Module 4 Assignment- resubmission

November 14, 2024

1 Module 4 - Reducing unfairness in learning algorithm applications

1.0.1 Assignment overview

In this assignment, you are tasked to create a classifer to predict the estimated income of individuals in the Kaggle Adult Income Dataset. This dataset is known to be biased towards certain groups. You will try some strategies to create a more fair classifier.

For this assignment, it is possible to work in **groups of up to 2 students**. Read the instructions carefully, as they may assign tasks to specific students.

1.0.2 Group members

Leave blanks if group has less than 2 members: - Student 1: Yanxin Liang 50798412 - Student 2: Yelia Ye 89657605

1.0.3 Learning Goals:

After completing this week's lecture and tutorial work, you will be able to: 1. Discuss the consequences of erroneous (biased) data on the training of learning algorithms and how it impacts its end users
2. Discuss potential ethical implications in errors in feature selection, model selection 3. Describe strategies for reducing algorithmic bias 4. Apply strategies to reduce unfairness in a predictive model trained on an unbalanced dataset 5. Describe advantages and limitations of the strategies used to reduce unfairness in predictive models

1.0.4 Libraries

Here are some libraries you will need for this assignment. `imblearn` and `aif360` are new ones, you can install it by running the cell below. Comment out this line after one execution:

[ ]:
```
#!pip install imblearn
#!pip install aif360
```

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

```python
from sklearn.metrics import accuracy_score
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
from imblearn.over_sampling import SMOTENC
import matplotlib.pyplot as plt
from aif360.algorithms.postprocessing import EqOddsPostprocessing
from aif360.datasets import BinaryLabelDataset
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    #plot_confusion_matrix,
    f1_score,
    make_scorer,
    ConfusionMatrixDisplay,
    accuracy_score, precision_score, recall_score, roc_auc_score,
 ↪confusion_matrix
)

import warnings
warnings.filterwarnings('ignore')
```

1.0.5 Dataset

The dataset you will use for this assignment is the Kaggle Adult Income Dataset. You may visit the source page for more information about this dataset.

The dataset includes 15 columns: 14 of them are demographics and other features to describe a person, and one (the target variable), is their income. The income variable is binary and has the two possible values `<=50K` or `>50K`.

Let's start by importing the dataset and taking a look (you are free to add other lines if you want more details):

[ ]:
```python
df = pd.read_csv("adult.csv")
df.head()
```

[ ]:
```
   age  workclass  fnlwgt     education  educational-num      marital-status  \
0   25    Private  226802          11th                7       Never-married
1   38    Private   89814       HS-grad                9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-spouse
3   44    Private  160323  Some-college               10  Married-civ-spouse
4   18          ?  103497  Some-college               10       Never-married

          occupation relationship   race gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black   Male             0             0
1    Farming-fishing      Husband  White   Male             0             0
2    Protective-serv      Husband  White   Male             0             0
```

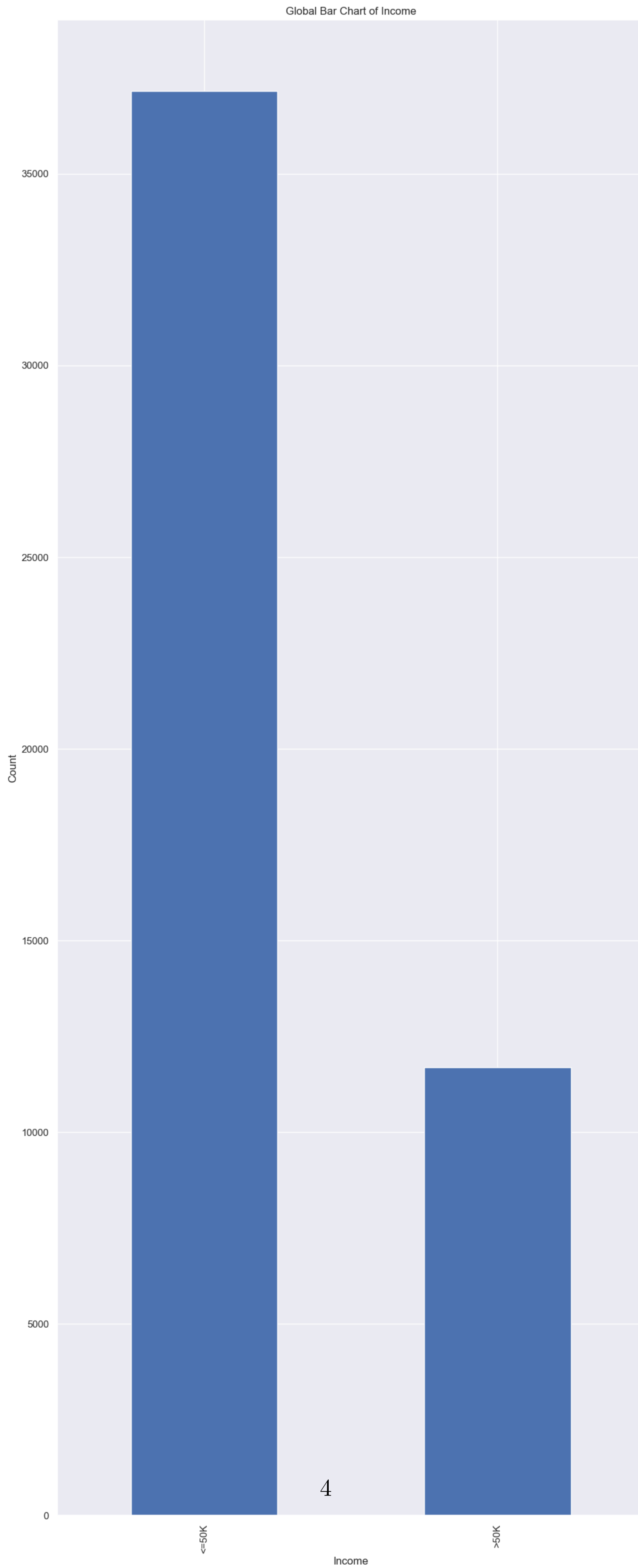
```
3  Machine-op-inspct      Husband  Black    Male          7688              0
4                  ?     Own-child  White   Female            0              0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
```

Unfortunately, this dataset is notoriously biased in the association between income and other demographic information, such as race and gender. Let's see how.

Question 1 Create the following 3 bar charts: - A global bar chart of the target variable - A bar chart of the target variable divided by gender - A bar chart of the target variable divided by race

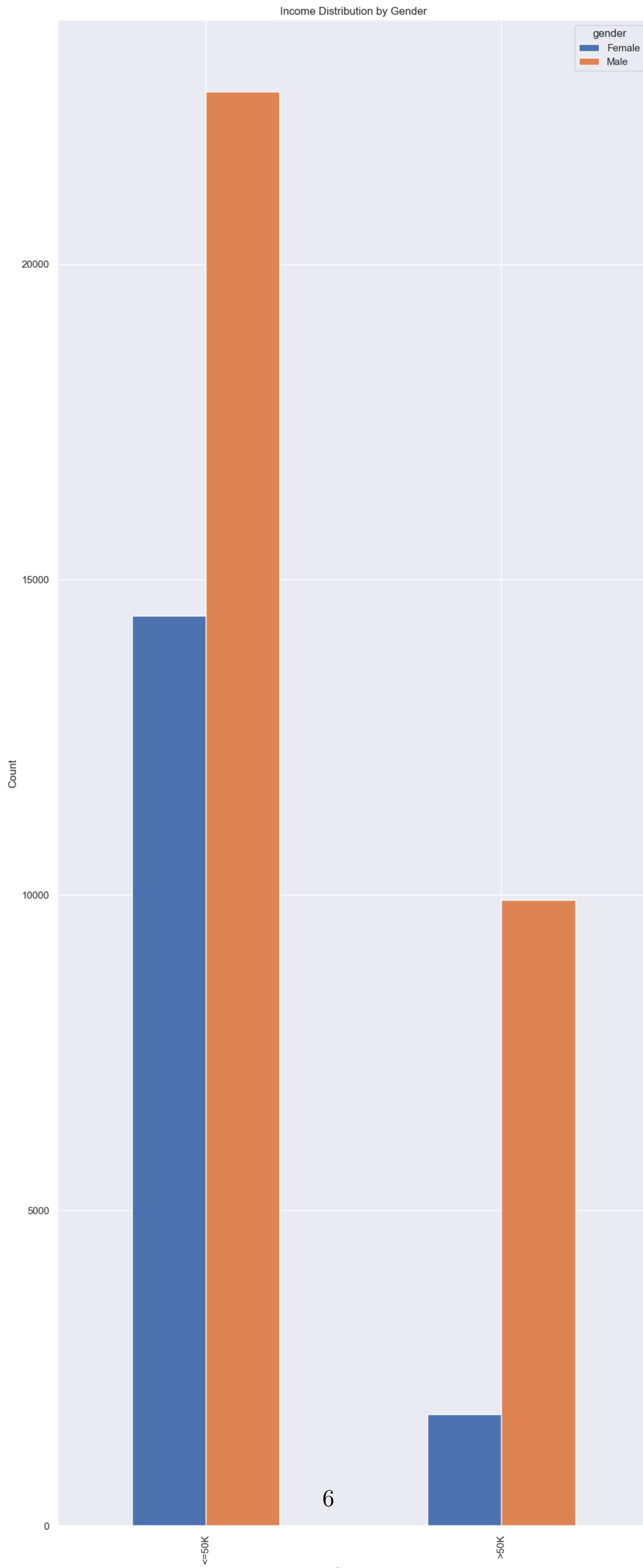
Comment on the results. Is the target variable balanced? Is the target variable balanced across protected groups?

[ ]:
```
df['income'].value_counts().plot(kind='bar')
plt.title('Global Bar Chart of Income')
plt.xlabel('Income')
plt.ylabel('Count')
plt.show()
```

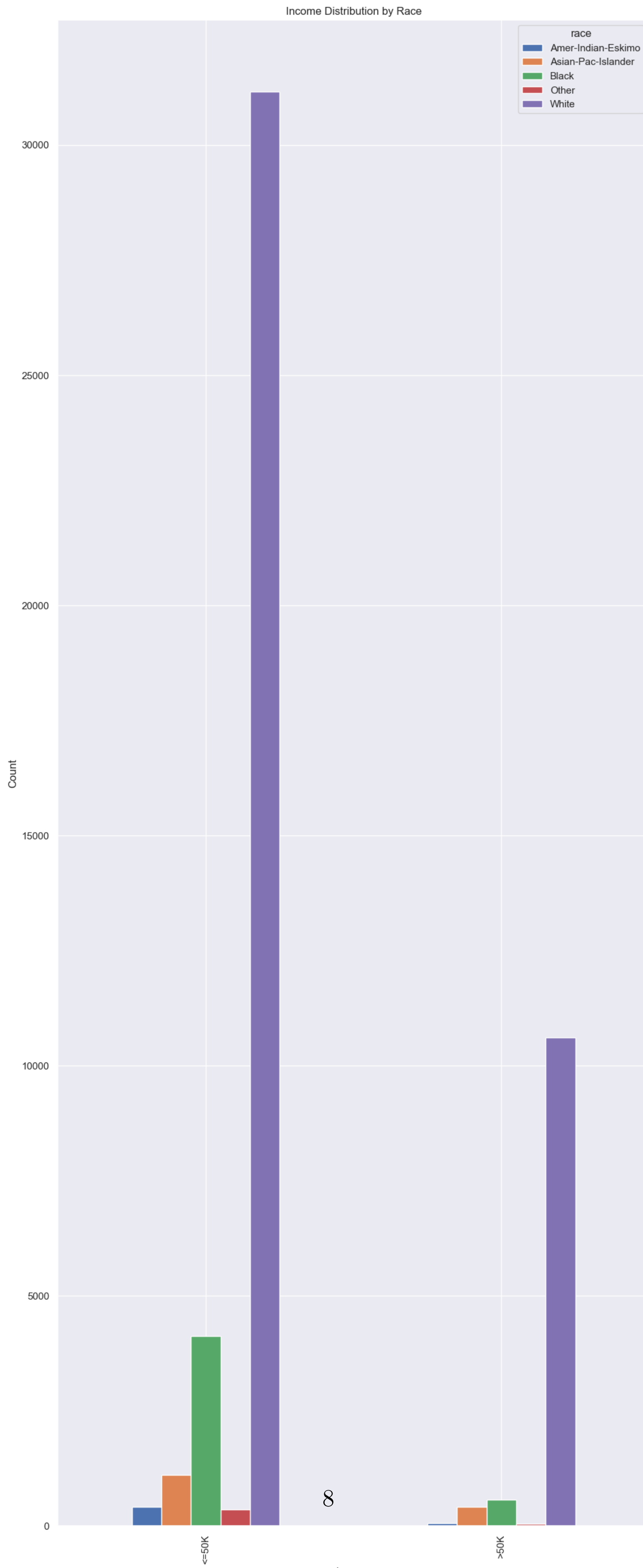


Global Bar Chart of Income
Count
35000
30000
25000
20000
15000
10000
5000
0
<=50K
>50K
Income

[ ]:
```python
df.groupby(['income', 'gender']).size().unstack().plot(kind='bar')
plt.title('Income Distribution by Gender')
plt.xlabel('Income')
plt.ylabel('Count')
plt.show()
```


Income Distribution by Gender
gender
Female
Male
20000
15000
10000
5000
0
Count
<=50K
>50K
Income

[ ]:
```
df.groupby(['income', 'race']).size().unstack().plot(kind='bar')
plt.title('Income Distribution by Race')
plt.xlabel('Income')
plt.ylabel('Count')
plt.show()
```


Income Distribution by Race
race
Amer-Indian-Eskimo
Asian-Pac-Islander
Black
Other
White
30000
25000
20000
15000
10000
5000
0
Count
<=50K
>50K
Income

From the first bar chart, we can clearly see that the income is imbalanced. The majority of individuals earn <=50K, with far fewer earning >50K. This imbalance indicates that the dataset could lead to biased predictions, as the model might favor predicting the majority class, which is <=50K.

When we split the income by gender, we can also see an imbalance. In class >50K, the proportions of male are much higer than female. This shows that gender might be a factor leading income inequality, and a biased predictions might be made which prefer to predict female into a lower income class.

The income distribution in different racial groups is also imbalanced. The white population dominates the dataset in both <=50K and >50K classes. In contrast, Amer-Indian-Eskimo, Asian-Pac-Islander, Black, and Other have significantly fewer data in both income categories. This bar chart shows that the minority groups are not only underrepresented but also have lower possibility to earn higher incomes.

In conclusion, the target variable isn't balanced and it is not balanced across protected groups.

1.0.6 A biased classifier

We can expect that a classifier trained on this kind of data will show some problematic behaviors when assigning an individual to a predicted income level. Let's visualize this using a random forest classifier.

[ ]:
```
# STEP 1
# Run this cell create training and test sets
train_df, test_df = train_test_split(df, test_size=0.3, random_state=123)

X_train, y_train = (
    train_df.drop(columns=["income"]),
    train_df["income"],
)
X_test, y_test = (
    test_df.drop(columns=["income"]),
    test_df["income"],
)
```

[ ]:
```
# STEP 2
# Run this cell to do the necessary dataset preprocessing (encoding of␣
 ↪categorical features).
# Note that, since we are using a tree based classifier, we don't need to scale␣
 ↪the
# numerical features.

categorical_feats = ["workclass",
                     "education",
                     "marital-status",
```

```
                      "occupation",
                      "relationship",
                      "race",
                      "gender",
                      "native-country",
                      ]  # Apply one-hot encoding
passthrough_feats = ["age",
                "fnlwgt",
                "educational-num",
                "capital-gain",
                "capital-loss",
                "hours-per-week"
                ]  # Numerical - no need to scale
target = "income"

ct = make_column_transformer(
    (
        make_pipeline(OneHotEncoder(handle_unknown="ignore",drop="if_binary")),
        categorical_feats,
    ),  # OHE on categorical features
    ("passthrough", passthrough_feats)  # no transformations on numerical␣
 ↪features
)

X_train_transformed = ct.fit_transform(X_train).toarray()

column_names = list(
    ct.named_transformers_["pipeline"].get_feature_names_out(
        categorical_feats
    )
) + passthrough_feats

X_test_transformed = ct.transform(X_test).toarray()
```

[ ]:
```
# You may use this lines to check the result
#pd.DataFrame(X_train_transformed, columns=column_names)
#pd.DataFrame(X_test_transformed, columns=column_names)
```

[ ]:
```
# STEP 3
# Run this cell to train a random forest classifer. The hyperparameters have␣
 ↪been pre-selected

from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=0, max_depth = 19, n_estimators =␣
 ↪100).fit(X_train_transformed, y_train)
```

How good is this classifier? Let's check its accuracy, by running the cells below:

[ ]:
```
clf.score(X_train_transformed, y_train)
```

[ ]: 0.907046125946942

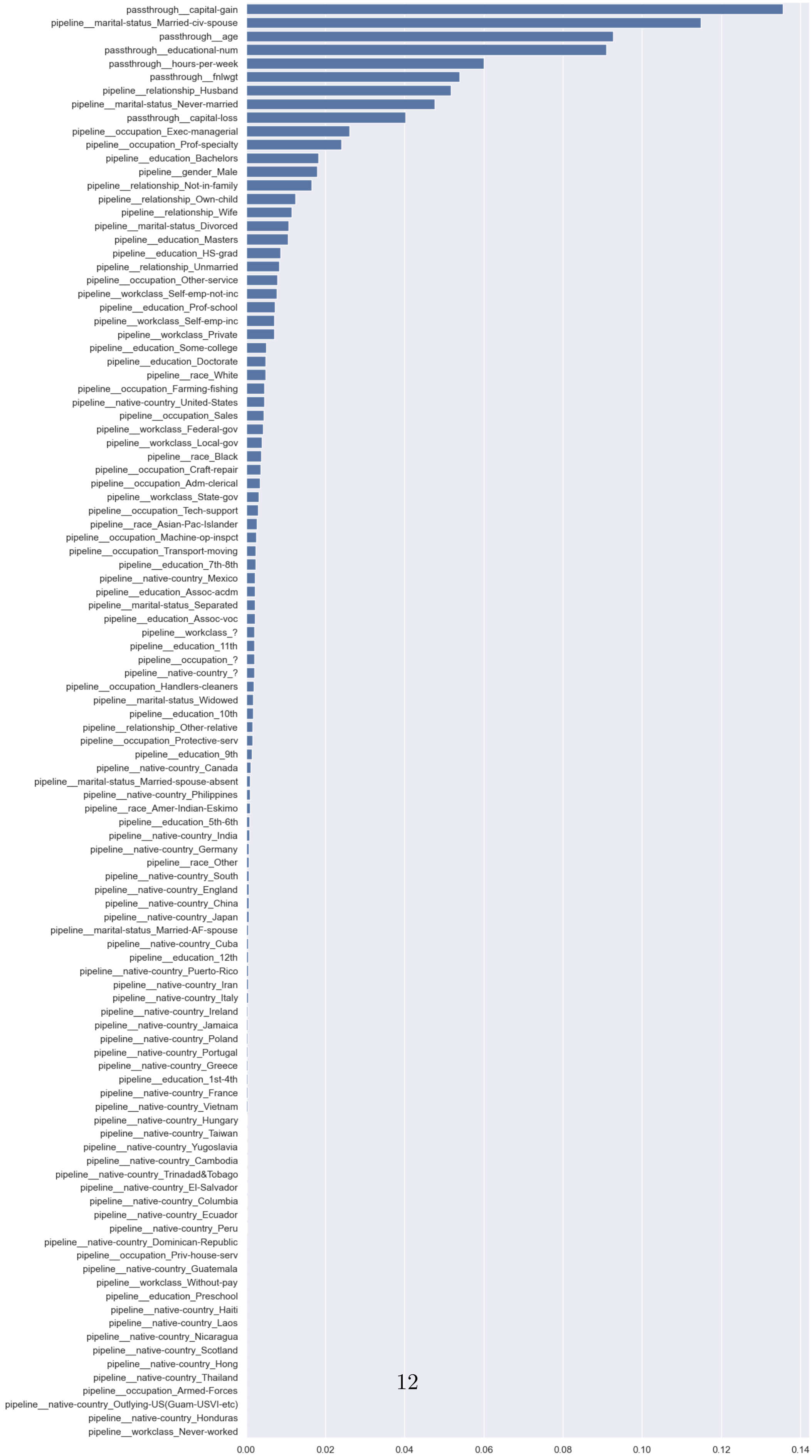
[ ]:
```
clf.score(X_test_transformed, y_test)
```

[ ]: 0.8614618166928274

Finally, let's see what features are considered important by the classifier.

[ ]:
```
import seaborn as sns

feature_importances = clf.feature_importances_

# Sort the feature importances from greatest to least using the sorted indices
sorted_indices = feature_importances.argsort()[::-1]
sorted_feature_names = ct.get_feature_names_out()[sorted_indices]
sorted_importances = feature_importances[sorted_indices]

# # Create a bar plot of the feature importances
sns.set(rc={'figure.figsize':(11.7,30)})
sns.barplot(x=sorted_importances, y=sorted_feature_names)
```

[ ]: <Axes: >


passthrough__capital-gain
pipeline__marital-status_Married-civ-spouse
passthrough__age
passthrough__educational-num
passthrough__hours-per-week
passthrough__fnlwgt
pipeline__relationship_Husband
pipeline__marital-status_Never-married
passthrough__capital-loss
pipeline__occupation_Exec-managerial
pipeline__occupation_Prof-specialty
pipeline__education_Bachelors
pipeline__gender_Male
pipeline__relationship_Not-in-family
pipeline__relationship_Own-child
pipeline__relationship_Wife
pipeline__marital-status_Divorced
pipeline__education_Masters
pipeline__education_HS-grad
pipeline__relationship_Unmarried
pipeline__occupation_Other-service
pipeline__workclass_Self-emp-not-inc
pipeline__education_Prof-school
pipeline__workclass_Self-emp-inc
pipeline__workclass_Private
pipeline__education_Some-college
pipeline__education_Doctorate
pipeline__race_White
pipeline__occupation_Farming-fishing
pipeline__native-country_United-States
pipeline__occupation_Sales
pipeline__workclass_Federal-gov
pipeline__workclass_Local-gov
pipeline__race_Black
pipeline__occupation_Craft-repair
pipeline__occupation_Adm-clerical
pipeline__workclass_State-gov
pipeline__occupation_Tech-support
pipeline__race_Asian-Pac-Islander
pipeline__occupation_Machine-op-inspct
pipeline__occupation_Transport-moving
pipeline__education_7th-8th
pipeline__native-country_Mexico
pipeline__education_Assoc-acdm
pipeline__marital-status_Separated
pipeline__education_Assoc-voc
pipeline__workclass_?
pipeline__education_11th
pipeline__occupation_?
pipeline__native-country_?
pipeline__occupation_Handlers-cleaners
pipeline__marital-status_Widowed
pipeline__education_10th
pipeline__relationship_Other-relative
pipeline__occupation_Protective-serv
pipeline__education_9th
pipeline__native-country_Canada
pipeline__marital-status_Married-spouse-absent
pipeline__native-country_Philippines
pipeline__race_Amer-Indian-Eskimo
pipeline__education_5th-6th
pipeline__native-country_India
pipeline__native-country_Germany
pipeline__race_Other
pipeline__native-country_South
pipeline__native-country_England
pipeline__native-country_China
pipeline__native-country_Japan
pipeline__marital-status_Married-AF-spouse
pipeline__native-country_Cuba
pipeline__education_12th
pipeline__native-country_Puerto-Rico
pipeline__native-country_Iran
pipeline__native-country_Italy
pipeline__native-country_Ireland
pipeline__native-country_Jamaica
pipeline__native-country_Poland
pipeline__native-country_Portugal
pipeline__native-country_Greece
pipeline__education_1st-4th
pipeline__native-country_France
pipeline__native-country_Vietnam
pipeline__native-country_Hungary
pipeline__native-country_Taiwan
pipeline__native-country_Yugoslavia
pipeline__native-country_Cambodia
pipeline__native-country_Trinadad&Tobago
pipeline__native-country_El-Salvador
pipeline__native-country_Columbia
pipeline__native-country_Ecuador
pipeline__native-country_Peru
pipeline__native-country_Dominican-Republic
pipeline__occupation_Priv-house-serv
pipeline__native-country_Guatemala
pipeline__workclass_Without-pay
pipeline__education_Preschool
pipeline__native-country_Haiti
pipeline__native-country_Laos
pipeline__native-country_Nicaragua
pipeline__native-country_Scotland
pipeline__native-country_Hong
pipeline__native-country_Thailand
pipeline__occupation_Armed-Forces
pipeline__native-country_Outlying-US(Guam-USVI-etc)
pipeline__native-country_Honduras
pipeline__workclass_Never-worked
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

Question 2 What are the most important features for this classifier? Do they include protected characteristics, such as race or gender?

The most important features are Capital-gain, Marrial-status, Age, Education-num, and hours-per-week in this classifier. Marrial-status may contain information about protected characteristics such as gender, age, and family status. Education-num may contain information about protected characteristics such as gender, age, and race. The feature Age is directly related to the protected characteristics of age. Although Gender and Race don't rank as high as the features mentioned before, they are also protected characteristics.They can still introduce bias into the prediction model. They may not directly influence predictions in a very strong way, but they might affect predictions when they combined with other features. It's important to be mindful of these features to prevent to introduce biases into the prediction model.

Question 3 From Assignment 3, we have learned that a classifier may perform well in terms of accuracy, but being unfair to protected groups in the dataset. Use what you have learned in Assignment 3 and **evaluate this classifier for fairness in treating the two gender groups included in this dataset.** In particular, do the following:

- Compute the 6 fairness metrics and the Average Distance from the Reference on training and test sets. You may reuse portions of code you have included in Assignment 3.
- Comment on the results, providing an interpretation for each computed metric; how different is the treatment of the two groups? Is one (or more) of the metrics particularly concerning?

Here is a recap of the fairness metrics: 1. *Predicted Positive Rate Disparity (PPRD)*, whether the numbers of positive predictions are on par across groups. 2. *Predicted Positive Group Rate Disparity (PPGRD)*, whether the rates of positive predictions are on par across groups. 3. *False Discovery Rate Disparity (FDRD)*, whether the ratios of false positives to predicted positives are on par across groups. 4. *False Positive Rate Disparity (FPRD)*, whether the ratios of false positives to actual negatives are on par across groups. 5. *False Omission Rate Disparity (FORD)*, whether the ratios of false negatives to predicted negatives are on par across groups. 6. *False Negative Rate Disparity (FNRD)*, whether the ratios of false negatives to actual positives are on par across groups.

[ ]:
```
# Your answer here (you may add more cells)

def fairness_metrics(model,test_x,test_y,gender_exist=True,is_TEST = False):
  if gender_exist == True:
    x1 = test_x
  else:
    if is_TEST == False:
        x1 = pd.DataFrame(X_train_transformed, columns=column_names)
    else:
       x1 = pd.DataFrame(X_test_transformed, columns=column_names)


  dict={}
```

```
    pred = model.predict(test_x)


    group_female = (x1["gender_Male"] == 0.0).tolist()
    group_male = (x1["gender_Male"] == 1.0).tolist()


    cm_female = confusion_matrix(test_y[group_female], pred[group_female])
    cm_male = confusion_matrix(test_y[group_male], pred[group_male])

    TNw, FPw, FNw, TPw = cm_male.ravel()
    TNb, FPb, FNb, TPb = cm_female.ravel()

   #w represents male,  b represents female

    dict["PPRD"] = (TPb + FPb) / (TPw + FPw)
    dict["PPGRD"] = ((TPb + FPb)/ (TPb + FPb + TNb + FNb)) / ((TPw + FPw) /(TPw +␣
 ↪FPw + TNw + FNw))
    dict["FDRD"]= (FPb / (TPb + FPb))/(FPw / (TPw + FPw))
    dict["FPRD"] = (FPb / (TNb + FPb))/(FPw / (TNw + FPw))
    dict["FORD"] = (FNb / (TNb + FNb))/(FNw / (TNw + FNw))
    dict["FNRD"] = (FNb / (TPb + FNb))/(FNw / (TPw + FNw))


    AVG_D = sum(abs(value - 1) for value in dict.values()) / 6

    dict["Average Distance from Reference"] = AVG_D
    dict["Accuracy"] = model.score(test_x, test_y)


    metrics_df =  pd.DataFrame(list(dict.items()), columns=['Metric', 'Value'])
    return metrics_df
```

[ ]:
```
results_dict = {}

results_dict = {
 "Training_set": fairness_metrics(clf,pd.DataFrame(X_train_transformed,␣
 ↪columns=column_names),y_train),
 "Testing_set": fairness_metrics(clf,pd.DataFrame(X_test_transformed,␣
 ↪columns=column_names),y_test)
}

results_df = pd.concat(
    {name: result.set_index('Metric')['Value'] for name, result in results_dict.
 ↪items()}, axis=1).T
```

```
results_df
```

```
[ ]: Metric            PPRD     PPGRD      FDRD      FPRD      FORD      FNRD  \
     Training_set  0.172842  0.349042  0.370874  0.101238  0.265089  0.883967
     Testing_set   0.140122  0.281567  0.817173  0.179615  0.357543  1.237044

     Metric        Average Distance from Reference  Accuracy
     Training_set                         0.642825  0.907046
     Testing_set                          0.576837  0.861462
```

Predicted Positive Rate Difference (PPRD) calculates the ratio of the number of positive predictions between the two groups. In the training set, PPRD is 0.172842, and in the testing set, it is 0.140122. Both of the PPRD are less than 1, this shows that there exist imbalance in our prediction. This shows that the model prefer to predict the male into the class >50K than the female group.

Predicted Positive Group Rate Disparity (PPGRD) calculates the the ratio of the positive predictions rate between the two groups. In the training set, PPGRD is 0.349042, and in the testing set, it is 0.281567. Same as what we conclude for PPRD, there is an imbalance between male and female group. The female group is predicted with a lower positive rate compared to the other group.

False Discovery Rate Disparity (FDRD) calculates the ratio of false positives (predict as >50K but actual is <=50K) rate between the two groups. In the training set, FDRD is 0.370874, and in the testing set, it is 0.817173. The results are less than 1, which means the male group has a higher rate of incorrect positive predictions.

False Positive Rate Disparity (FPRD) calculates the ratio of the proportion of false positives among all incorrect predictions between the two groups. In the training set, FPRD is 0.101238, and in the testing set, it is 0.179615. The result shows that the model has a much higher proportion of false positives among all incorrect predictions for male group.

False Omission Rate Disparity (FORD) calculates the ratio of the proportion of false negatives among all predicted negatives between the two groups. In the training set, FORD is 0.265089, and in the testing set, it is 0.357543. The result shows that the female group is more frequently misclassified as negative, which raising concerns about bias.

False Negative Rate Disparity (FNRD) calculates the ratio of the proportion of actual positives incorrectly predicted as negatives between the two groups. In the training set, FNRD is 0.883967, and in the testing set, it is 1.237044. Since FNRD is close to 1 in the training set but exceeds 1 in the testing set, it indicates that, especially in the testing data, the female group experiences a higher rate of false negatives. This means that the model is more likely to predict a low income the female group.

The Average Distance from Reference aggregates these values and measures the overall fairness of the model. For training dataset, the Average Distance is 0.6428, and the Average Distanc for testing dataset is 0.5768. The value of the average distance is high and shows that there is fairness problem, particularly, the disparities in PPGRD and FNRD. There exist bias that the model prefer to predict female as low income.

1.1 Debiasing techniques: dropping protected characteristics

A first idea to fix this issue could be dropping the protected characteristics from our dataset before training the classifier. Let's try this out and see if there is any improvement.

Question 4

1. Drop race, gender and native country from training and test set (we are focusing on gender but we will drop race and native country for good measure).
2. Transform the cleaned dataset using one-hot encoding.
3. Re-train the random forest classifier.
4. Compare accuracy and fairness of this new classifier to the previous one. Do we see any improvement? How do you explain the changes you see (or lack thereof)? Note that, to compare fairness, you will need to have a way to identify the gender of each sample, even though you are not using this feature for classification.
5. Create a new plot of the feature importance according to this classifier. Do you see any changes from the first one?

Hint: steps 2, 3 and 5 can be completed by tweaking the starting code given at the beginning of this assignment. Ask a TA or instructor if you need help in doing that.

[ ]:
```
#Drop race, gender and native country from training and test set
new_train = train_df.drop(columns=["race","gender","native-country"])
new_test = test_df.drop(columns=["race","gender","native-country"])
```

[ ]:
```
#transform the dataset using OHE
new_categorical_feats = ["workclass",
                         "education",
                         "marital-status",
                         "occupation",
                         "relationship"
                         ]  # Apply one-hot encoding



new_ct = make_column_transformer(
    (
        make_pipeline(OneHotEncoder(handle_unknown="ignore",drop="if_binary")),
        new_categorical_feats,
    ),  # OHE on categorical features
    ("passthrough", passthrough_feats)  # no transformations on numerical␣
 ↪features
)

new_X_train_transformed = new_ct.fit_transform(new_train).toarray()

new_column_names = list(
    new_ct.named_transformers_["pipeline"].get_feature_names_out(
        new_categorical_feats
    )
```

```
) + passthrough_feats

new_X_test_transformed = new_ct.transform(new_test).toarray()
```

[ ]:
```
new_clf = RandomForestClassifier(random_state=0, max_depth = 19, n_estimators =␣
↪100).fit(new_X_train_transformed, y_train)
```

[ ]:
```
new_train_metrics = fairness_metrics(new_clf, pd.
↪DataFrame(new_X_train_transformed, columns=new_column_names), y_train,␣
↪gender_exist=False,is_TEST=False)
new_test_metrics = fairness_metrics(new_clf, pd.
↪DataFrame(new_X_test_transformed, columns=new_column_names), y_test,␣
↪gender_exist=False,is_TEST=True)
results_dict2 = {}
results_dict2 = results_dict.copy()
results_dict2["Drop characteristics Training Set"] = new_train_metrics
results_dict2["Drop characteristics Test Set"] = new_test_metrics

results_df2 = pd.concat({name: result.set_index('Metric')['Value'] for name,␣
↪result in results_dict2.items()}, axis=1).T

results_df2
```

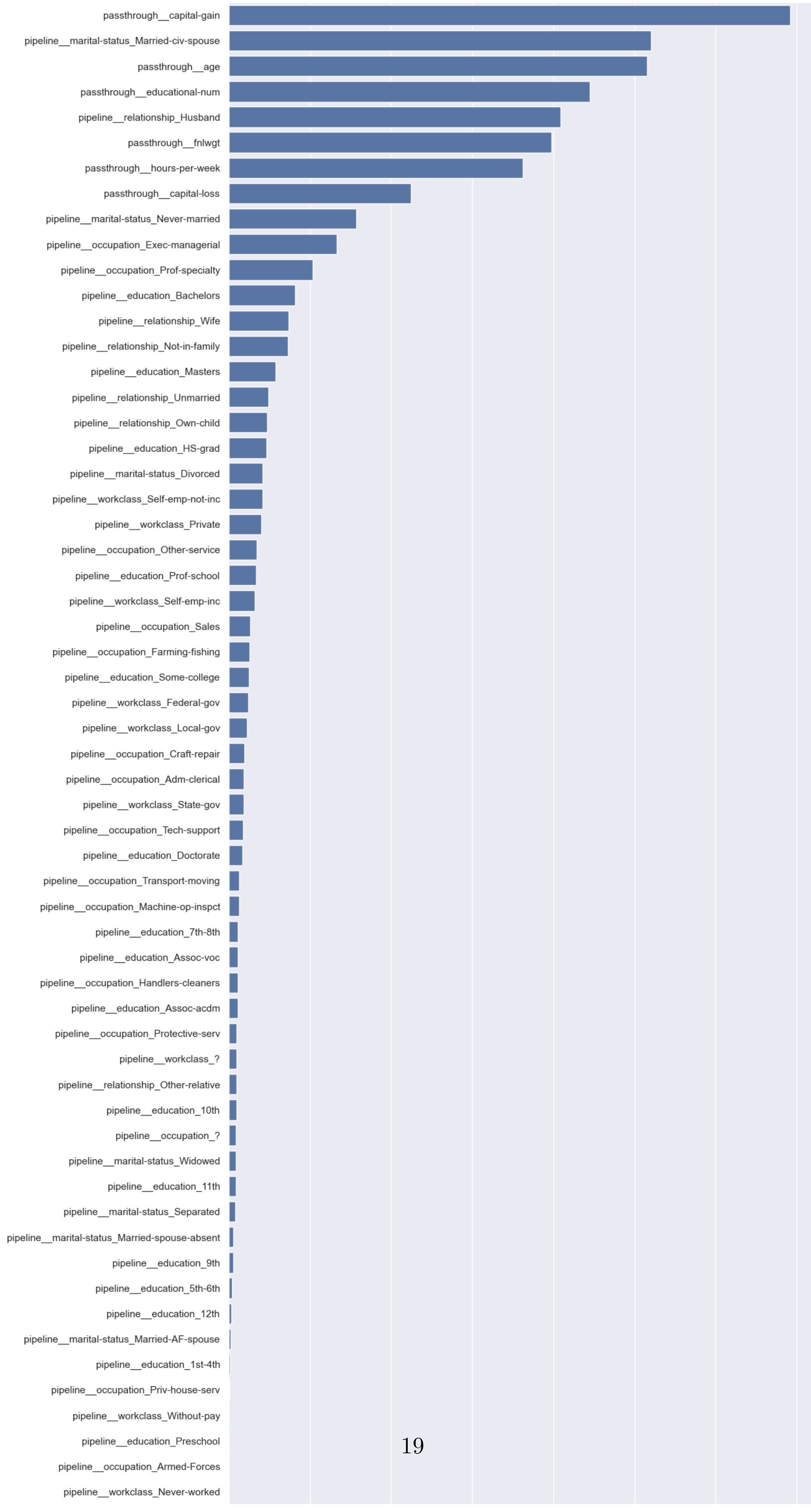
[ ]:
```
Metric                                 PPRD     PPGRD      FDRD      FPRD  \
Training_set                       0.172842  0.349042  0.370874  0.101238
Testing_set                        0.140122  0.281567  0.817173  0.179615
Drop characteristics Training Set  0.175075  0.353552  0.279112  0.077174
Drop characteristics Test Set      0.150408  0.302237  0.985436  0.232500

Metric                                 FORD      FNRD  \
Training_set                       0.265089  0.883967
Testing_set                        0.357543  1.237044
Drop characteristics Training Set  0.255225  0.853163
Drop characteristics Test Set      0.351786  1.214334

Metric                             Average Distance from Reference  Accuracy
Training_set                                              0.642825  0.907046
Testing_set                                               0.576837  0.861462
Drop characteristics Training Set                         0.667783  0.919009
Drop characteristics Test Set                             0.531995  0.861530
```

[ ]:
```
def draw_feature_importance(model,transformer):
    feature_importances_func = model.feature_importances_
    sorted_indices_func = feature_importances_func.argsort()[::-1]
    sorted_feature_names_func = transformer.
↪get_feature_names_out()[sorted_indices_func]
    sorted_importances_func = feature_importances_func[sorted_indices_func]
```

```
    return sns.barplot(x=sorted_importances_func, y=sorted_feature_names_func)
```

[ ]:
```
draw_feature_importance(new_clf,new_ct)
```

[ ]: <Axes: >


passthrough__capital-gain
pipeline__marital-status_Married-civ-spouse
passthrough__age
passthrough__educational-num
pipeline__relationship_Husband
passthrough__fnlwgt
passthrough__hours-per-week
passthrough__capital-loss
pipeline__marital-status_Never-married
pipeline__occupation_Exec-managerial
pipeline__occupation_Prof-specialty
pipeline__education_Bachelors
pipeline__relationship_Wife
pipeline__relationship_Not-in-family
pipeline__education_Masters
pipeline__relationship_Unmarried
pipeline__relationship_Own-child
pipeline__education_HS-grad
pipeline__marital-status_Divorced
pipeline__workclass_Self-emp-not-inc
pipeline__workclass_Private
pipeline__occupation_Other-service
pipeline__education_Prof-school
pipeline__workclass_Self-emp-inc
pipeline__occupation_Sales
pipeline__occupation_Farming-fishing
pipeline__education_Some-college
pipeline__workclass_Federal-gov
pipeline__workclass_Local-gov
pipeline__occupation_Craft-repair
pipeline__occupation_Adm-clerical
pipeline__workclass_State-gov
pipeline__occupation_Tech-support
pipeline__education_Doctorate
pipeline__occupation_Transport-moving
pipeline__occupation_Machine-op-inspct
pipeline__education_7th-8th
pipeline__education_Assoc-voc
pipeline__occupation_Handlers-cleaners
pipeline__education_Assoc-acdm
pipeline__occupation_Protective-serv
pipeline__workclass_?
pipeline__relationship_Other-relative
pipeline__education_10th
pipeline__occupation_?
pipeline__marital-status_Widowed
pipeline__education_11th
pipeline__marital-status_Separated
pipeline__marital-status_Married-spouse-absent
pipeline__education_9th
pipeline__education_5th-6th
pipeline__education_12th
pipeline__marital-status_Married-AF-spouse
pipeline__education_1st-4th
pipeline__occupation_Priv-house-serv
pipeline__workclass_Without-pay
pipeline__education_Preschool
pipeline__occupation_Armed-Forces
pipeline__workclass_Never-worked
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

After removing race, gender, and native country from the dataset, the accuracy of the classifier on the test set remains nearly the same, changing only slightly from 0.86146 to 0.86153. This suggests that these protected features didn't improve the predictive power, as the model still performs well without those protected characteristics. When we look at the fairness metrics, we can see some small improvements. For example, the False Positive Rate Disparity (FPRD) changes from 0.1796 to 0.2325 on the test set, and the False Negative Rate Disparity (FNRD) changes from 1.2370 to 1.2143. The model without those protected characteristics is a little fairer.

The fairness metrics are still not fully balanced, which still showing some bias. It might because other features in the dataset, like education level, marital status may still contain the information of race or gender. For example, the Predicted Positive Group Rate Disparity (PPGRD) is still at 0.3022 on the test set, suggesting that the model still prefer to predict the male groups as positive.

For the test set, Average Distance decreased from 0.5768 to 0.5320 which gave evidence of that removing protected characteristics helps reduce some of the unfairness between groups.

The updated feature importance plot also shows that some features have become more important. The features like marital status and capital gain become more important after race, gender, and native country removed. And the features capital_gain and hours_per_week is the most important features for the new classifier.

In summary, dropping protected features does make the model a little fairer, but it still can't remove bias completely, since the other features in the dataset can still influence the predictions.

1.2 Debiasing techniques: undersampling

As you should have seen when exploring the dataset, the groups of males and females who make more or less than \$50k are of very different sizes. This alone may have a significant impact on the way the classifier is trained, by teaching it that some groups are much more likely to make more than \$50k than others.

Let's try to fix this problem by creating a more balanced training set.

Question 5

1. Run the cell below to create a new training set by selecting a subset of samples from the original one, in which the groups of males and females who make more or less than \$50k are of equal size. To use the maximum number of training samples possible, the size of each group should be equal to the size of the smallest of these groups in the original dataset. **What is the size of each group, and of the final training set?**
2. Separate features from target, and transform the cleaned dataset using one-hot encoding. **Remeber to re-transform the test set accordingly!**
3. Re-train the random forest classifier.
4. Compare accuracy and fairness of this new classifier to the previous ones. Do we see any improvement? How do you explain the changes you see (or lack thereof)? Pay particular attention to the difference in results on the training and test set: can you explain these results?

5. Create a new plot of the feature importance according to this classifier. Do you see any changes from the previous ones?

[ ]:
```
# Check the distribution of gender and income
gender_distribution = train_df['gender'].value_counts()
income_distribution = train_df['income'].value_counts()

# Create balanced subsets
balanced_subsets = []
smallest = train_df.shape[0]

# Finding size of smallest subset by gender and income
for gender_category in gender_distribution.index:
    for income_category in income_distribution.index:
        if train_df[(train_df['gender'] == gender_category) &␣
 ↪(train_df['income'] == income_category)].shape[0] < smallest:
            smallest = train_df[(train_df['gender'] == gender_category) &␣
 ↪(train_df['income'] == income_category)].shape[0]

# Sampling subsets
for gender_category in gender_distribution.index:
    for income_category in income_distribution.index:
        subset = train_df[(train_df['gender'] == gender_category) &␣
 ↪(train_df['income'] == income_category)]
        subset = subset.sample(smallest)  # Sample to match the minimum count
        balanced_subsets.append(subset)

# Merge the balanced subsets to create the final balanced dataset
balanced_df = pd.concat(balanced_subsets)
```

[ ]:
```
#step 2: transformation

X_train_undersampling, y_train_undersampling = (
    balanced_df.drop(columns=["income"]),
    balanced_df["income"],
)

ct_undersampling = make_column_transformer(
    (
        make_pipeline(OneHotEncoder(handle_unknown="ignore",drop="if_binary")),
         categorical_feats,
    ), # OHE on categorical features
    ("passthrough", passthrough_feats) # no transformations on numerical␣
 ↪features
)
```

```
X_train_under_transformed = ct_undersampling.
 ↪fit_transform(X_train_undersampling).toarray()

column_names_under = list(
    ct_undersampling.named_transformers_["pipeline"].get_feature_names_out(
        categorical_feats
    )
) + passthrough_feats

X_test_under_transformed = ct_undersampling.transform(X_test).toarray()

#step 3: retrain the classifier

clf_undersampling = RandomForestClassifier(random_state=0, max_depth = 19,␣
 ↪n_estimators = 100).fit(X_train_under_transformed, y_train_undersampling)
```

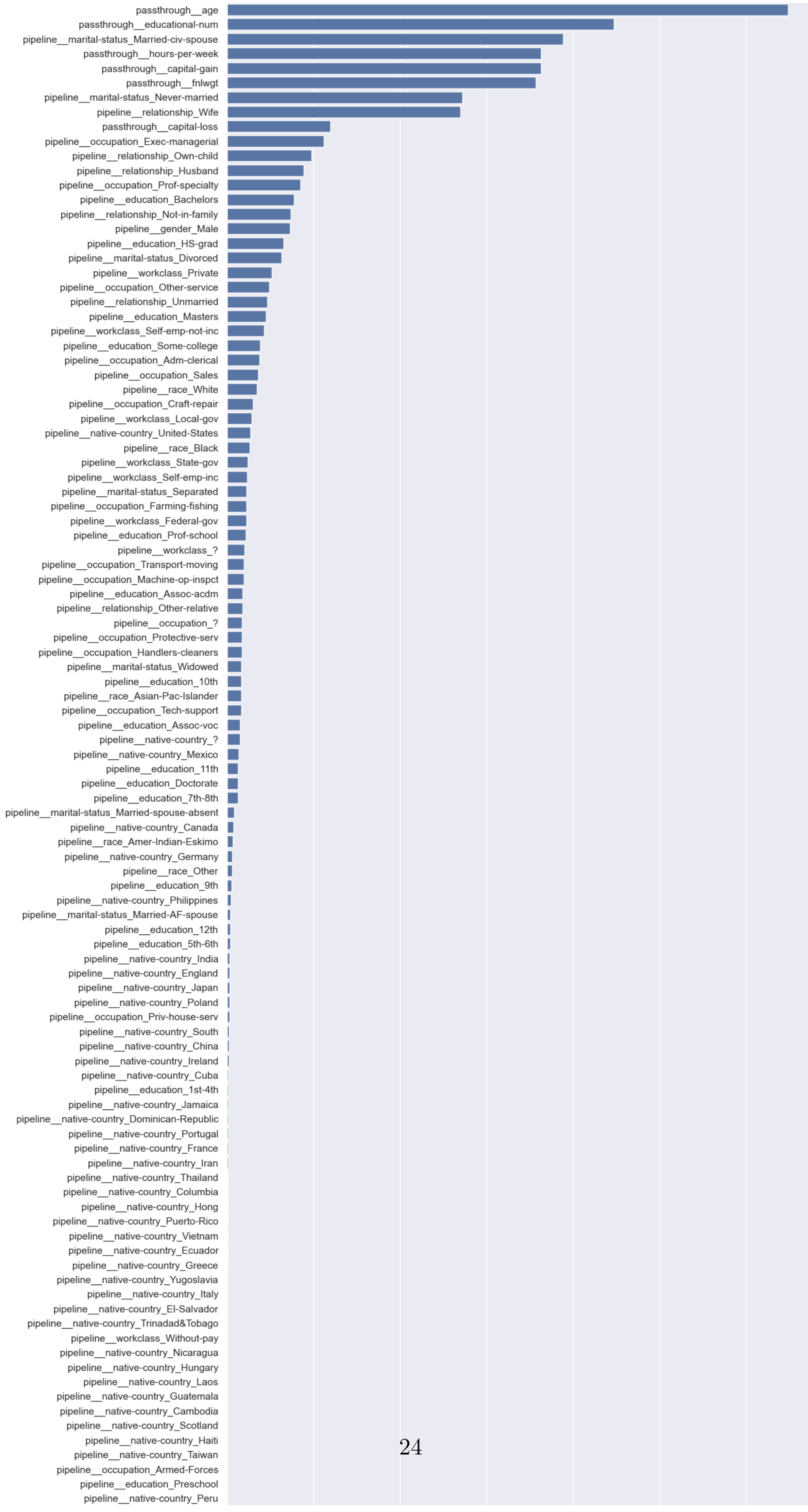
[ ]:
```
#compare
under_train_metrics = fairness_metrics(clf_undersampling, pd.
 ↪DataFrame(X_train_under_transformed, columns=column_names_under),␣
 ↪y_train_undersampling)
under_test_metrics = fairness_metrics(clf_undersampling, pd.
 ↪DataFrame(X_test_under_transformed, columns=column_names_under), y_test)

results_dict3 = {}
results_dict3 = results_dict2.copy()
results_dict3["Undersampling Training Set"] = under_train_metrics
results_dict3["Undersampling Test Set"] = under_test_metrics

results_df3 = pd.concat({name: result.set_index('Metric')['Value'] for name,␣
 ↪result in results_dict3.items()}, axis=1).T

results_df3
```

[ ]:
```
Metric                                 PPRD     PPGRD      FDRD      FPRD  \
Training_set                       0.172842  0.349042  0.370874  0.101238
Testing_set                        0.140122  0.281567  0.817173  0.179615
Drop characteristics Training Set  0.175075  0.353552  0.279112  0.077174
Drop characteristics Test Set      0.150408  0.302237  0.985436  0.232500
Undersampling Training Set         0.979054  0.979054  0.374032  0.366197
Undersampling Test Set             0.255802  0.514022  1.449859  0.581772

Metric                                 FORD      FNRD  \
Training_set                       0.265089  0.883967
Testing_set                        0.357543  1.237044
Drop characteristics Training Set  0.255225  0.853163
Drop characteristics Test Set      0.351786  1.214334
Undersampling Training Set         0.410194  0.419355
```

```
Undersampling Test Set             0.230591  0.878020

Metric                             Average Distance from Reference  Accuracy
Training_set                                               0.642825  0.907046
Testing_set                                                0.576837  0.861462
Drop characteristics Training Set                          0.667783  0.919009
Drop characteristics Test Set                              0.531995  0.861530
Undersampling Training Set                                 0.412019  0.971777
Undersampling Test Set                                     0.498275  0.808776
```

[ ]:
```
draw_feature_importance(clf_undersampling,ct_undersampling)
```

[ ]: <Axes: >



passthrough__age
passthrough__educational-num
pipeline__marital-status_Married-civ-spouse
passthrough__hours-per-week
passthrough__capital-gain
passthrough__fnlwgt
pipeline__marital-status_Never-married
pipeline__relationship_Wife
passthrough__capital-loss
pipeline__occupation_Exec-managerial
pipeline__relationship_Own-child
pipeline__relationship_Husband
pipeline__occupation_Prof-specialty
pipeline__education_Bachelors
pipeline__relationship_Not-in-family
pipeline__gender_Male
pipeline__education_HS-grad
pipeline__marital-status_Divorced
pipeline__workclass_Private
pipeline__occupation_Other-service
pipeline__relationship_Unmarried
pipeline__education_Masters
pipeline__workclass_Self-emp-not-inc
pipeline__education_Some-college
pipeline__occupation_Adm-clerical
pipeline__occupation_Sales
pipeline__race_White
pipeline__occupation_Craft-repair
pipeline__workclass_Local-gov
pipeline__native-country_United-States
pipeline__race_Black
pipeline__workclass_State-gov
pipeline__workclass_Self-emp-inc
pipeline__marital-status_Separated
pipeline__occupation_Farming-fishing
pipeline__workclass_Federal-gov
pipeline__education_Prof-school
pipeline__workclass_?
pipeline__occupation_Transport-moving
pipeline__occupation_Machine-op-inspct
pipeline__education_Assoc-acdm
pipeline__relationship_Other-relative
pipeline__occupation_?
pipeline__occupation_Protective-serv
pipeline__occupation_Handlers-cleaners
pipeline__marital-status_Widowed
pipeline__education_10th
pipeline__race_Asian-Pac-Islander
pipeline__occupation_Tech-support
pipeline__education_Assoc-voc
pipeline__native-country_?
pipeline__native-country_Mexico
pipeline__education_11th
pipeline__education_Doctorate
pipeline__education_7th-8th
pipeline__marital-status_Married-spouse-absent
pipeline__native-country_Canada
pipeline__race_Amer-Indian-Eskimo
pipeline__native-country_Germany
pipeline__race_Other
pipeline__education_9th
pipeline__native-country_Philippines
pipeline__marital-status_Married-AF-spouse
pipeline__education_12th
pipeline__education_5th-6th
pipeline__native-country_India
pipeline__native-country_England
pipeline__native-country_Japan
pipeline__native-country_Poland
pipeline__occupation_Priv-house-serv
pipeline__native-country_South
pipeline__native-country_China
pipeline__native-country_Ireland
pipeline__native-country_Cuba
pipeline__education_1st-4th
pipeline__native-country_Jamaica
pipeline__native-country_Dominican-Republic
pipeline__native-country_Portugal
pipeline__native-country_France
pipeline__native-country_Iran
pipeline__native-country_Thailand
pipeline__native-country_Columbia
pipeline__native-country_Hong
pipeline__native-country_Puerto-Rico
pipeline__native-country_Vietnam
pipeline__native-country_Ecuador
pipeline__native-country_Greece
pipeline__native-country_Yugoslavia
pipeline__native-country_Italy
pipeline__native-country_El-Salvador
pipeline__native-country_Trinadad&Tobago
pipeline__workclass_Without-pay
pipeline__native-country_Nicaragua
pipeline__native-country_Hungary
pipeline__native-country_Laos
pipeline__native-country_Guatemala
pipeline__native-country_Cambodia
pipeline__native-country_Scotland
pipeline__native-country_Haiti
pipeline__native-country_Taiwan
pipeline__occupation_Armed-Forces
pipeline__education_Preschool
pipeline__native-country_Peru
0.00
0.02
0.04
0.06
0.08
0.10
0.12

After balancing the dataset through undersampling, the size of each group is 1249 and the that of final training set is 4996. This balance helps ensure the equal representation across male and female groups and also can reducing bias from an imbalanced dataset.

This improve the fairness metrics. We can see a more fairness results between two groups. For example, PPGRD is much closer to 1 right now from 0.2816 (original model) to 0.5014 (with undersampling) on the test set. FNRD is also more closer to 1 from 1.2370 to 0.9841 on the test set. Although fairness metrics on the training set is improved, the accuracy of the model slightly decreases from 0.86 to 0.81 on the test dataset.

The Average Distance improved on both training and testing datasets. For the training set, it decreased from 0.643 to 0.412, and on the testing set, it changes from 0.577 to 0.498. These decreases indicate that undersampling makes the classifier fairer.

The feature importance plot shows a change in the order of the imortant features. The features like capital_gain, education_num, and marital_status are still in the top of most important features. The feature age is the most important feature in this case. In conclusion, we can say that undersampling improves fairness by balancing the training data, but it may lose some of the accuracy.

The discrepancy between the training and test set performance in undersampling suggests underfitting. With undersampling, the model was trained on a smaller, balanced dataset, which helped improve fairness on the training set by giving equal representation to each group. However, this approach reduced the diversity of data available, which likely made it difficult for the model to generalize to new, unseen data, as seen in the decreased accuracy on the test set (from 0.86 to 0.81).

1.3 Debiasing techniques: oversampling (with SMOTE)

Another way to create a more balanced training set, but without sacrificing training samples, is by *oversampling*, which means artificially increasing the size of the training set with "fake" samples. This can be achieved mainly in two ways: 1. By resampling (replicating) samples from the original training set, or 2. By introducing artificial *new* samples, similar enough to those included in the original training set

The Synthetic Minority Oversampling Technique (SMOTE) seen in class falls in the second group. In this portion of the assignment, you will create a more balanced dataset using SMOTE (specifically SMOTENC, a version of SMOTE that allows working with categorical variables).

Question 6

1. Run the cell below to create a more balanced training set using SMOTE. Note that a large portion of code is replicated to guarantee that the correct data is used, and not one modified in previous cells. The actual rebalancing all happens in the last 2 lines.
2. Explore the new training set, and provide the following information: what is the size of the new training set? Is the target variable balanced? How many samples are classified as >\$50, and how many as <=\$50k? Is the target variable balanced across protected groups, or at

least more balanced than before? How many males and females are classified as >\$50, and how many as <=\$50k?
3. Re-train the random forest classifier.
4. Compare accuracy and fairness of this new classifier to the previous ones. Do we see any improvement? How do you explain the changes you see (or lack thereof)? Pay particular attention to the difference in results on the training and test set: can you explain these results?
5. Create a new plot of the feature importance according to this classifier. Do you see any changes from the previous ones?

[ ]:
```
from imblearn.over_sampling import SMOTENC

X_train, y_train = (
    train_df.drop(columns=["income"]),
    train_df["income"],
)
X_test, y_test = (
    test_df.drop(columns=["income"]),
    test_df["income"],
)

oversample = SMOTENC(categorical_features=categorical_feats, random_state=0)

X_train_SMOTE, y_train_SMOTE = oversample.fit_resample(X_train, y_train)
```

[ ]:
```
# Transformation applied after oversampling

categorical_feats = ["workclass",
                     "education",
                     "marital-status",
                     "occupation",
                     "relationship",
                     "race",
                     "gender",
                     "native-country",
                     ]  # Apply one-hot encoding
passthrough_feats = ["age",
                "fnlwgt",
                "educational-num",
                "capital-gain",
                "capital-loss",
                "hours-per-week"
                ]  # Numerical - no need to scale
target = "income"


ctSMOTE = make_column_transformer(
```

```
    (
        ␣
↪OneHotEncoder(handle_unknown="ignore",drop="if_binary",sparse_output=False),
        categorical_feats,
    ),  # OHE on categorical features
    ("passthrough", passthrough_feats)  # no transformations on numerical␣
↪features
)

X_train_transformed = ctSMOTE.fit_transform(X_train_SMOTE)
X_test_transformed = ctSMOTE.transform(X_test)

# Column names, if needed
column_names = list(
    ctSMOTE.named_transformers_["onehotencoder"].get_feature_names_out(
        categorical_feats
    )
) + passthrough_feats

# X_train_transformed and X_test_transformed can now be used to answer the␣
↪questions above
```

[ ]:
```
clf_SMOTE = RandomForestClassifier(random_state=0, max_depth = 19, n_estimators␣
↪= 100).fit(X_train_transformed, y_train_SMOTE)
```

[ ]:
```
results_dict4 = {}
results_dict4 = results_dict3.copy()
results_dict4["SMOTE Training Set"] = fairness_metrics(clf_SMOTE, pd.
↪DataFrame(X_train_transformed, columns=column_names), y_train_SMOTE)
results_dict4["SMOTE Test Set"] = fairness_metrics(clf_SMOTE, pd.
↪DataFrame(X_test_transformed, columns=column_names), y_test)

results_df4 = pd.concat({name: result.set_index('Metric')['Value'] for name,␣
↪result in results_dict4.items()}, axis=1).T

results_df4
```

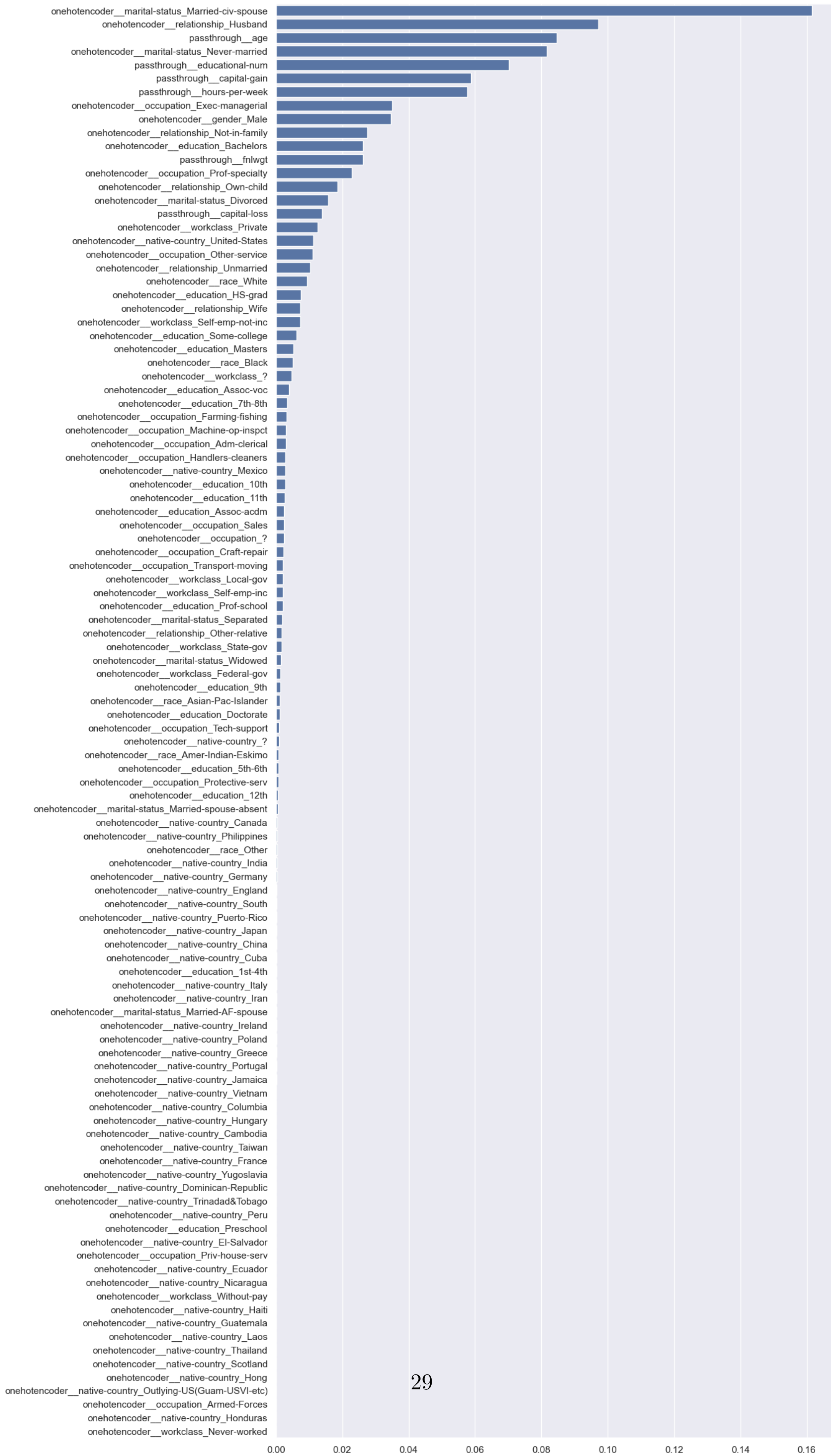
[ ]:

Metric	PPRD	PPGRD	FDRD	FPRD	\
Training_set	0.172842	0.349042	0.370874	0.101238	
Testing_set	0.140122	0.281567	0.817173	0.179615	
Drop characteristics Training Set	0.175075	0.353552	0.279112	0.077174	
Drop characteristics Test Set	0.150408	0.302237	0.985436	0.232500	
Undersampling Training Set	0.979054	0.979054	0.374032	0.366197	
Undersampling Test Set	0.255802	0.514022	1.449859	0.581772	
SMOTE Training Set	0.072110	0.239060	0.770999	0.087805	
SMOTE Test Set	0.103781	0.208543	0.655867	0.106772	

```
Metric                              FORD      FNRD  \
Training_set                    0.265089  0.883967
Testing_set                     0.357543  1.237044
Drop characteristics Training Set  0.255225  0.853163
Drop characteristics Test Set   0.351786  1.214334
Undersampling Training Set      0.410194  0.419355
Undersampling Test Set          0.230591  0.878020
SMOTE Training Set              0.364001  3.173955
SMOTE Test Set                  0.499459  2.105907

Metric                           Average Distance from Reference  Accuracy
Training_set                                            0.642825  0.907046
Testing_set                                             0.576837  0.861462
Drop characteristics Training Set                       0.667783  0.919009
Drop characteristics Test Set                           0.531995  0.861530
Undersampling Training Set                              0.412019  0.971777
Undersampling Test Set                                  0.498275  0.808776
SMOTE Training Set                                      0.939997  0.920682
SMOTE Test Set                                          0.755248  0.841944
```

[ ]:
```
draw_feature_importance(clf_SMOTE,ctSMOTE)
```

[ ]: <Axes: >


onehotencoder__marital-status_Married-civ-spouse
onehotencoder__relationship_Husband
passthrough__age
onehotencoder__marital-status_Never-married
passthrough__educational-num
passthrough__capital-gain
passthrough__hours-per-week
onehotencoder__occupation_Exec-managerial
onehotencoder__gender_Male
onehotencoder__relationship_Not-in-family
onehotencoder__education_Bachelors
passthrough__fnlwgt
onehotencoder__occupation_Prof-specialty
onehotencoder__relationship_Own-child
onehotencoder__marital-status_Divorced
passthrough__capital-loss
onehotencoder__workclass_Private
onehotencoder__native-country_United-States
onehotencoder__occupation_Other-service
onehotencoder__relationship_Unmarried
onehotencoder__race_White
onehotencoder__education_HS-grad
onehotencoder__relationship_Wife
onehotencoder__workclass_Self-emp-not-inc
onehotencoder__education_Some-college
onehotencoder__education_Masters
onehotencoder__race_Black
onehotencoder__workclass_?
onehotencoder__education_Assoc-voc
onehotencoder__education_7th-8th
onehotencoder__occupation_Farming-fishing
onehotencoder__occupation_Machine-op-inspct
onehotencoder__occupation_Adm-clerical
onehotencoder__occupation_Handlers-cleaners
onehotencoder__native-country_Mexico
onehotencoder__education_10th
onehotencoder__education_11th
onehotencoder__education_Assoc-acdm
onehotencoder__occupation_Sales
onehotencoder__occupation_?
onehotencoder__occupation_Craft-repair
onehotencoder__occupation_Transport-moving
onehotencoder__workclass_Local-gov
onehotencoder__workclass_Self-emp-inc
onehotencoder__education_Prof-school
onehotencoder__marital-status_Separated
onehotencoder__relationship_Other-relative
onehotencoder__workclass_State-gov
onehotencoder__marital-status_Widowed
onehotencoder__workclass_Federal-gov
onehotencoder__education_9th
onehotencoder__race_Asian-Pac-Islander
onehotencoder__education_Doctorate
onehotencoder__occupation_Tech-support
onehotencoder__native-country_?
onehotencoder__race_Amer-Indian-Eskimo
onehotencoder__education_5th-6th
onehotencoder__occupation_Protective-serv
onehotencoder__education_12th
onehotencoder__marital-status_Married-spouse-absent
onehotencoder__native-country_Canada
onehotencoder__native-country_Philippines
onehotencoder__race_Other
onehotencoder__native-country_India
onehotencoder__native-country_Germany
onehotencoder__native-country_England
onehotencoder__native-country_South
onehotencoder__native-country_Puerto-Rico
onehotencoder__native-country_Japan
onehotencoder__native-country_China
onehotencoder__native-country_Cuba
onehotencoder__education_1st-4th
onehotencoder__native-country_Italy
onehotencoder__native-country_Iran
onehotencoder__marital-status_Married-AF-spouse
onehotencoder__native-country_Ireland
onehotencoder__native-country_Poland
onehotencoder__native-country_Greece
onehotencoder__native-country_Portugal
onehotencoder__native-country_Jamaica
onehotencoder__native-country_Vietnam
onehotencoder__native-country_Columbia
onehotencoder__native-country_Hungary
onehotencoder__native-country_Cambodia
onehotencoder__native-country_Taiwan
onehotencoder__native-country_France
onehotencoder__native-country_Yugoslavia
onehotencoder__native-country_Dominican-Republic
onehotencoder__native-country_Trinadad&Tobago
onehotencoder__native-country_Peru
onehotencoder__education_Preschool
onehotencoder__native-country_El-Salvador
onehotencoder__occupation_Priv-house-serv
onehotencoder__native-country_Ecuador
onehotencoder__native-country_Nicaragua
onehotencoder__workclass_Without-pay
onehotencoder__native-country_Haiti
onehotencoder__native-country_Guatemala
onehotencoder__native-country_Laos
onehotencoder__native-country_Thailand
onehotencoder__native-country_Scotland
onehotencoder__native-country_Hong
onehotencoder__native-country_Outlying-US(Guam-USVI-etc)
onehotencoder__occupation_Armed-Forces
onehotencoder__native-country_Honduras
onehotencoder__workclass_Never-worked
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16

[ ]:
```
train_df["gender"].value_counts()
```

[ ]:
```
gender
Male      22866
Female    11323
Name: count, dtype: int64
```

[ ]:
```
X_train_SMOTE["gender"].value_counts()
```

[ ]:
```
gender
Male      39925
Female    12043
Name: count, dtype: int64
```

[ ]:
```
y_train_SMOTE.value_counts()
```

[ ]:
```
income
<=50K    25984
>50K     25984
Name: count, dtype: int64
```

[ ]:
```
print(train_df.groupby(['gender', 'income']).size().unstack(fill_value=0))
```

```
income  <=50K  >50K
gender
Female  10074  1249
Male    15910  6956
```

[ ]:
```
x2=X_train_SMOTE

x2['income'] = y_train_SMOTE

# Check distribution by gender and income
gender_income_counts = x2.groupby(['gender', 'income']).size().
 ↪unstack(fill_value=0)

print(gender_income_counts)
```

```
income  <=50K   >50K
gender
Female  10074   1969
Male    15910  24015
```

The size of training set is 51968. The target variable is balanced, with 25984 samples for both income levels (<=\$50K and >\$50K). The target variable is not balanced across protected groups. 24015 males and 1969 female are classified as >\$50 while 15910 males and 10074 female are classified as <=\$50k. SMOTE increased the balance between two income level by increased the number of individuals with >50K income, especially for males, but it has not balanced the gender distribution in income.

This classifier didn't show much improvement in accuracy or fairness. It clearly struggles with overfitting, which you can see from the significant difference between training accuracy (0.9207) and testing accuracy (0.8419). This issue arises because SMOTE, an oversampling method that generates artificial data points, may not add enough diversity to the dataset. The Average Distance from Reference is the highest among all techniques, at 0.9400 for the training set and 0.75525 for the testing set. Additionally, the PPRD and FNRD values are far from 1, indicating a considerable imbalance in positive predictions across gender groups. Overall, this suggests that SMOTE performs poorly in terms of fairness.

The important features in this case are marital status, relationship, age, educational level (in numerical form), capital gain, and hours per week. While these features are generally consistent with those identified by other methods, marital status stands out as the most important feature, replacing capital gain and age. Notably, capital gain is not as influential as it is in other methods.

1.4 Equalized odd post processing

An alternative to the methods seen so far (which may produce unsatisfactory results), is applying post-processing to the predictions of the classifier, so that they optimize equalized odds (whether the TPR and FPR are on par across groups).

`aif360`, a popular open-source library dedicated to detecting and mitigating bias in machine learning models, includes `EqOddsPostprocessing`, a function to performe equalized odds post-processing. The function is slightly more intricate to use than others you have used so far (typically from `sklearn`), so we will see together how to apply it on the test (you may try and replicate this on the training set for your own practice).

[ ]:
```
# Run this cell to reset training and test sets (and clear accidental prior␣
↪changes)

X_train, y_train = (
    train_df.drop(columns=["income"]),
    train_df["income"],
)
X_test, y_test = (
    test_df.drop(columns=["income"]),
    test_df["income"],
)
```

[ ]:
```
# Run this cell to do the necessary dataset preprocessing (encoding of␣
↪categorical features).
# Note that, since we are using a tree based classifier, we don't need to scale␣
↪the
# numerical features.

categorical_feats = ["workclass",
                     "education",
```

```
                      "marital-status",
                      "occupation",
                      "relationship",
                      "race",
                      "gender",
                      "native-country",
                      ]  # Apply one-hot encoding
passthrough_feats = ["age",
                "fnlwgt",
                "educational-num",
                "capital-gain",
                "capital-loss",
                "hours-per-week"
                ]  # Numerical - no need to scale
target = "income"

ct = make_column_transformer(
    (
        make_pipeline(OneHotEncoder(handle_unknown="ignore",drop="if_binary")),
        categorical_feats,
    ),  # OHE on categorical features
    ("passthrough", passthrough_feats)  # no transformations on numerical␣
 ↪features
)

X_train_transformed = ct.fit_transform(X_train).toarray()
X_test_transformed = ct.transform(X_test).toarray()
```

[ ]:
```
# Convert numpy arrays to pandas dataframes

column_names = list(
    ct.named_transformers_["pipeline"].get_feature_names_out(
        categorical_feats
    )
) + passthrough_feats

X_train_df = pd.DataFrame(X_train_transformed, columns=column_names)
X_test_df = pd.DataFrame(X_test_transformed, columns=column_names)
```

[ ]:
```
# Train RandomForestClassifier
clf = RandomForestClassifier(random_state=0, max_depth = 19, n_estimators =␣
 ↪100).fit(X_train_df, y_train)

# Get predictions for test set
y_pred = clf.predict(X_test_df)

# So far, all this is the same as the biased classifier we started with
```

```
[ ]: # Convert test data into a BinaryLabelDataset, necessary to work in aif360

from aif360.datasets import BinaryLabelDataset

X_test_df = X_test_df.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)

y_binary = y_test.map({'>50K': 1, '<=50K': 0})  # Map categorical values to␣
 ↪binary

test_bld = BinaryLabelDataset(df=pd.concat([X_test_df, y_binary], axis=1),
                              label_names=['income'],
                              protected_attribute_names=['gender_Male'])
```

```
[ ]: # Create another dataset with predicted labels for comparison
test_pred_bld = test_bld.copy()

# Convert to binary label (e.g., class 2 is positive, others are negative)
y_pred_binary = np.where(y_pred == '>50K', 1, 0)

test_pred_bld.labels = y_pred_binary.reshape(-1, 1)
```

```
[ ]: from aif360.algorithms.postprocessing import EqOddsPostprocessing

# Initialize EqOddsPostprocessing
eq_odds = EqOddsPostprocessing(unprivileged_groups=[{'gender_Male': 0}],
                               privileged_groups=[{'gender_Male': 1}])
```

```
[ ]: # Fit the EqOddsPostprocessing model
eq_odds = eq_odds.fit(test_bld, test_pred_bld)

# Get new fair predictions
fair_pred_bld = eq_odds.predict(test_pred_bld)

# Convert predictions back to array
fair_predictions = fair_pred_bld.labels
```

```
[ ]: fair_predictions_cat = np.where(fair_predictions == 1, '>50K', '<=50K')
fair_predictions_cat
```

```
[ ]: array([['<=50K'],
       ['<=50K'],
       ['<=50K'],
       …,
       ['<=50K'],
       ['<=50K'],
       ['<=50K']], dtype='<U5')
```

`fair_predictions_cat` now includes the post-processed predictions, after equalized odds postprocessing.

Question 7 Compute accuracy and fairness of this new predictions, and compare the results to the previous ones. Do we see any improvement? Is this technique more or less effective than the others tried before?

[ ]:
```
def fairness_metrics_odd(test_x,test_y,pred):
  dict={}

  group_female = (test_x["gender_Male"] == 0.0).tolist()
  group_male = (test_x["gender_Male"] == 1.0).tolist()


  cm_female = confusion_matrix(test_y[group_female], pred[group_female])
  cm_male = confusion_matrix(test_y[group_male], pred[group_male])

  TNw, FPw, FNw, TPw = cm_male.ravel()
  TNb, FPb, FNb, TPb = cm_female.ravel()

 #w represents male,  b represents female

  dict["PPRD"] = (TPb + FPb) / (TPw + FPw)
  dict["PPGRD"] = ((TPb + FPb)/ (TPb + FPb + TNb + FNb)) / ((TPw + FPw) /(TPw +␣
↪FPw + TNw + FNw))
  dict["FDRD"]= (FPb / (TPb + FPb))/(FPw / (TPw + FPw))
  dict["FPRD"] = (FPb / (TNb + FPb))/(FPw / (TNw + FPw))
  dict["FORD"] = (FNb / (TNb + FNb))/(FNw / (TNw + FNw))
  dict["FNRD"] = (FNb / (TPb + FNb))/(FNw / (TPw + FNw))


  AVG_D = sum(abs(value - 1) for value in dict.values()) / 6

  dict["Average Distance from Reference"] = AVG_D
  dict["Accuracy"] = accuracy_score(test_y,pred)


  metrics_df =  pd.DataFrame(list(dict.items()), columns=['Metric', 'Value'])
  return metrics_df
```

[ ]:
```

results_dict5 = {}
results_dict5 = results_dict4.copy()
results_dict5["Equalized odd post processing"] =␣
↪fairness_metrics_odd(X_test_df,y_test,fair_predictions_cat)
```

```
results_df5 = pd.concat({name: result.set_index('Metric')['Value'] for name,␣
↪result in results_dict5.items()}, axis=1).T

results_df5
```

```
[ ]: Metric                             PPRD     PPGRD      FDRD      FPRD  \
Training_set                       0.172842  0.349042  0.370874  0.101238
Testing_set                        0.140122  0.281567  0.817173  0.179615
Drop characteristics Training Set  0.175075  0.353552  0.279112  0.077174
Drop characteristics Test Set      0.150408  0.302237  0.985436  0.232500
Undersampling Training Set         0.979054  0.979054  0.374032  0.366197
Undersampling Test Set             0.255802  0.514022  1.449859  0.581772
SMOTE Training Set                 0.072110  0.239060  0.770999  0.087805
SMOTE Test Set                     0.103781  0.208543  0.655867  0.106772
Equalized odd post processing      0.280530  0.563710  2.215682  0.975009

Metric                             FORD      FNRD  \
Training_set                       0.265089  0.883967
Testing_set                        0.357543  1.237044
Drop characteristics Training Set  0.255225  0.853163
Drop characteristics Test Set      0.351786  1.214334
Undersampling Training Set         0.410194  0.419355
Undersampling Test Set             0.230591  0.878020
SMOTE Training Set                 0.364001  3.173955
SMOTE Test Set                     0.499459  2.105907
Equalized odd post processing      0.312454  0.987442

Metric                             Average Distance from Reference  Accuracy
Training_set                                              0.642825  0.907046
Testing_set                                               0.576837  0.861462
Drop characteristics Training Set                         0.667783  0.919009
Drop characteristics Test Set                             0.531995  0.861530
Undersampling Training Set                                0.412019  0.971777
Undersampling Test Set                                    0.498275  0.808776
SMOTE Training Set                                        0.939997  0.920682
SMOTE Test Set                                            0.755248  0.841944
Equalized odd post processing                             0.516090  0.835870
```

The accuracy of the new prediction is 0.8359, which is lower than most previous predictions. This decrease in accuracy is a result of the fairness-accuracy trade-off.

However, it demonstrates improved fairness compared to other methods. The FPRD (0.9750) and FNRD (0.9874) are the closest to 1, indicating that the False Positive Rates (FPR) and False Negative Rates (FNR) are nearly equal across gender groups. This suggests that the model's ability to correctly or incorrectly classify outcomes is more consistent between males and females.

On the other hand, PPRD (0.2805) and PPGRD (0.5637) are still far from 1, meaning that positive prediction parity and group-level prediction rates are not yet perfectly aligned. Despite these values being better than those of other methods, they show that there is still room for improvement.

Additionally, the FDRD (2.2157) indicates a significant fairness issue. The false discovery rate is much higher for females than for males, suggesting that more females are wrongly classified as earning >$50K, which introduces new bias against females. The Average Distance from Reference (AVD) is 0.5161, which is lower than that of upsampling but higher than that of other techniques.

Equalized Odds Postprocessing involves a trade-off between fairness and accuracy. In this case, the model's accuracy with Equalized Odds Postprocessing is 0.8359, which is slightly lower than most other methods. However, this approach achieved improved fairness across both training and test sets. It brought key fairness metrics, such as False Positive Rate Disparity (FPRD) and False Negative Rate Disparity (FNRD), closer to 1 on both sets, indicating more balanced outcomes across gender groups.

Compared to other debiasing techniques, Equalized Odds Postprocessing was more effective than dropping features and SMOTE, as it provided a better balance of fairness and accuracy. Although undersampling led to better fairness on the training set, it caused a larger drop in test accuracy and did not generalize fairness improvements as effectively to the test data. Thus, Equalized Odds Postprocessing proved to be a more reliable option for maintaining fairness on unseen data while managing the accuracy-fairness trade-off.

1.5 Final remarks

Question 8 Based on the results seen so far, provide an overall evaluation of our debiasing efforts. In particular, try answering the following questions: 1. What do you think was the most successful technique? Which one was the least successful? 2. If you found that bias still persists after attempting a debiasing strategy, what do you think is the reason? What could be done to fix this problem?

(max 400 words)

Although undersampling improved fairness metrics on the training set, it did not generalize as effectively to the test set, where accuracy dropped from 0.86 to 0.81. This suggests that undersampling led to underfitting by reducing data diversity, which limited the model's ability to perform well on unseen data. While undersampling helped balance the data in training, the ADR on the test set remained relatively high, indicating that fairness improvements were largely restricted to the training data and did not carry over well to new samples.

Equalized Odds Postprocessing was more effective in improving fairness across both the training and test sets. This technique adjusted predictions to ensure that False Positive and False Negative Rates were balanced between groups. As a result, metrics like False Positive Rate Disparity (FPRD) and False Negative Rate Disparity (FNRD) became closer to 1, showing that fairness improved across groups on both the training and test sets. Although accuracy decreased slightly to 0.8359, Equalized Odds maintained a more consistent ADR across both training and test sets, showing that it achieved more balanced fairness with better generalization than undersampling.

The least effective method was SMOTE. It didn't helpe with fairness even increase bias. Using SMOTE strategy also lose some accuracy. SMOTE didn't perform well, since SMOTE strategy only balanced the target feature, but it does not figure the imbalanced between different groups. This led to overfitting, with high training accuracy but lower test accuracy. The ADR for SMOTE was higher on the test set, reflecting poor fairness and performance on unseen data due to overfitting to the synthetic samples.

After using those three strategies, we found that bias still persists. One of the reasons of this might be there are some features still containing the information about protected features, even we drop the protected characteristics. Another reason is the original data might be strongly imbalanced, even we use the strategies to reduce some of the bias, the classifier is still unfair. In order to fix this problem, we can try to use some other debiasing strategies, such as adversarial debiasing and equalized odds post-processing. We can also try to use multiple debiasing strategies on the classifier.

2 Final thoughts

1) If you have completed this assignment in a group, please write a detailed description of how you divided the work and how you helped each other completing it:

We discussed our thoughts on the questions and shared our ideas with each other. And we divide the work so that each person could focus on specific parts of the assignment. We used GitHub to share our progress and make sure everything stayed organized. Once we completed all the questions, we came together to review the answers as a group, making sure that everything was clear and correct.

2) Have you used ChatGPT or a similar Large Language Model (LLM) to complete this homework? Please describe how you used the tool. We will never deduct points for using LLMs for completing homework assignments, but this helps us understand how you are using the tool and advise you in case we believe you are using it incorrectly.

We used ChatGPT to help us analyze and troubleshoot errors in our code when we couldn't resolve them on our own. ChatGPT provided suggestions on where the issues might be and how to fix them. We also used ChatGPT to help us correct some grammar mistakes in some parts of our written responses.

3) Have you struggled with some parts (or all) of this homework? Do you have pending questions you would like to ask? Write them down here!

We noticed that some debiasing strategies improved fairness but reduced accuracy. Is there a standard approach for determining the acceptable balance between fairness and accuracy?