```
import pandas as pd
from google.colab import drive
import numpy as np
import sklearn
from datetime import datetime
import matplotlib as matplot
from sklearn import preprocessing
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
import sklearn.metrics
import math
from xgboost import XGBRegressor

drive.mount('/content/gdrive')
# !ls "/content/gdrive/My Drive"
data_path = "/content/gdrive/My Drive/Master ADS/Week 8/store-sales-time-series-forecastin

def print_results_metrics(truth_values, predicted_values):
  print("RMSE: ", math.sqrt(sklearn.metrics.mean_squared_error(truth_values, predicted_val
  print("MAD: ", np.mean(np.absolute(predicted_values - np.mean(predicted_values))))
  print("MAE: ", sklearn.metrics.mean_absolute_error(truth_values, predicted_values))
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive
```

Task 1

Exercise 1

Load oil.csv. This file contains years worth of data of the daily oil price. However, the data is missing for a few days. Make sure that every day contains a value using any data imputation technique that you learned during the data preparation week or during the missing values imputation week.

```
oil_data = pd.read_csv(data_path + "oil.csv")
# checking missing values before applying imputation
print("Number of missing values before: {}".format(oil_data.isna().sum().sum()))
oil_data = oil_data.fillna(oil_data.mean())
# checking missing values before applying mean imputation
print("Number of missing values after: {}".format(oil_data.isna().sum().sum()))
```

```
Number of missing values before: 43
Number of missing values after: 0
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: FutureWarning: Droppi
  after removing the cwd from sys.path.
```

Exercise 2

Augment the data in test.csv and train.csv with the oil price

```
train_data = pd.read_csv(data_path + "train.csv")
test_data = pd.read_csv(data_path + "test.csv")
ground_truth_df = pd.read_csv(data_path + "submission.csv")

# train and test augmentation
# fill the missing values with the mean
# we have to do this because the oil.csv does not contain all the dates that train.csv has
train_data = pd.merge(train_data, oil_data, on=['date'], how='left')
train_data['dcoilwtico'].fillna(value = train_data['dcoilwtico'].mean(), inplace = True)
test_data = pd.merge(test_data, oil_data, on=['date'], how='left')
test_data['dcoilwtico'].fillna(value = test_data['dcoilwtico'].mean(), inplace = True)
```

```
train_data.head(10)
```

	id	date	store_nbr	family	sales	onpromotion	dcoilwtico
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0	67.714366
1	1	2013-01-01	1	BABY CARE	0.0	0	67.714366
2	2	2013-01-01	1	BEAUTY	0.0	0	67.714366
3	3	2013-01-01	1	BEVERAGES	0.0	0	67.714366
4	4	2013-01-01	1	BOOKS	0.0	0	67.714366
5	5	2013-01-01	1	BREAD/BAKERY	0.0	0	67.714366
6	6	2013-01-01	1	CELEBRATION	0.0	0	67.714366
7	7	2013-01-01	1	CLEANING	0.0	0	67.714366
8	8	2013-01-01	1	DAIRY	0.0	0	67.714366
9	9	2013-01-01	1	DELI	0.0	0	67.714366

```
test_data.head(10)
```

	id	date	store_nbr	family	onpromotion	dcoilwtico
0	3000888	2017-08-16	1	AUTOMOTIVE	0	46.8
1	3000889	2017-08-16	1	BABY CARE	0	46.8
2	3000890	2017-08-16	1	BEAUTY	2	46.8

Exercise 3

Note that the training set contains a ‘sales’ column while the test set doesnot. Use the training set to train a model of your choice and use that model to predict which values for sales should be in the test set. You can try training multiple models and compare their accuracy later.

Tranforming the data to a proper format

```
# instance of a label encoder
label_encoder = preprocessing.LabelEncoder()

# encoding family feature from train data
train_data['family'] = label_encoder.fit_transform(train_data['family'])

# encoding the family feature from test data
test_data['family']= label_encoder.fit_transform(test_data['family'])

# transforming the date to a numerical value
train_data['date'] = train_data['date'].apply(lambda x: datetime.fromisoformat(x).timestam
test_data['date'] = test_data['date'].apply(lambda x: datetime.fromisoformat(x).timestamp(
```

Random Forest model

```
# Initiate model
rf_regressor = RandomForestRegressor(n_estimators = 5, random_state = 0)
# Fit to training data
rf_regressor.fit(train_data.drop('sales', axis=1), train_data['sales'])
```

```
RandomForestRegressor(n_estimators=5, random_state=0)
```

Now we will make the predictions in the test set.

```
# Make predictions on test set
rf_prediction = rf_regressor.predict(test_data)
```

XGBRegressor model

Now we will make some changes in the data in order to be able to train a XGBRegressor model.

Preparation of the training data for XGBRegressor

```
# output of the training data
y_train = np.array(train_data['sales'])
# removing the columns that we dont need
train_data = train_data.drop('id', axis = 1)
train_data = train_data.drop('sales', axis = 1)

# store the names of the features in a list
feature_list = list(train_data.columns)
X_train = np.array(train_data)
print("Dimensions of the training set: {}".format(X_train.shape))
```

```
Dimensions of the training set: (3000888, 5)
```

Preparation of the test data for XGBRegressor

```
# removing the columns that we dont need
test_data = test_data.drop('id', axis = 1)

# store the names of the features in a list
feature_list = list(test_data.columns)
X_test = np.array(test_data)
print("Dimensions of the test set: {}".format(X_test.shape))
```

```
Dimensions of the test set: (28512, 5)
```

Training XGBRegressor model

```
xgB = XGBRegressor(n_estimators=30, objective='reg:squarederror')
xgB.fit(X_train, y_train, verbose= False)
```

```
XGBRegressor(n_estimators=30, objective='reg:squarederror')
```

Predicting values for the test set for XGBRegressor

```
xgbr_predictions = xgB.predict(X_test)
```

Exercise 4

Compare your prediction with the prediction found in submission.csv with 3 different methods:

- Root Mean Square Error (RMSE)1
- Mean Absolute Deviation
- Anoher Metri of your choise

Compare the three errors. Are they in agreement? Do you think any of themethods is objectively better than the others in this case?

```
truth_values = np.array(ground_truth_df['sales'])
```

Metrics for Random Forest

```
# comparing predicted values with ground truth and showing the metrics
print_results_metrics(truth_values, rf_prediction)
```

```
RMSE:  369.3718230551788
MAD:  626.5673612382647
MAE:  97.11466267240759
```

Metrics for GXBRegressor

```
# comparing predicted values with ground truth and showing the metrics
print_results_metrics(truth_values, xgbr_predictions)
```

```
RMSE:  720.8682103292491
MAD:  523.2903
MAE:  316.94458334800464
```

Conclusion

We trained two models (XGBRegressor and Random Forest) in order to check wheter either of them could give better results. With the metrics shown above we can see that models perform differently in some situations. For example, Random Forest got a better RMSE and MAD, but XGBRegressor obtained a better MAE score. All 3 methods of measuring the predictions are in agreement because they are all positive values. The scale might not be the same because of the way of measuring the error in each method. Finally, we believe that it is hard to see if one method is objectively better than the others in this specific case.

Task 2

Exercise 1

Determine which properties you want to consider privileged (e.g. age, gender,race, etc) and compute the following 3 fairness properties: (Note that these 3 metrics do not require a trained model)

- disparate impact ratio (DI ratio)

- statistical parity difference (P. diff.)
- consistency

What do these numbers tell you about the fairness of the dataset? Wouldyou say that the dataset is currently fair? If not, what numbers would youneed to see to judge a dataset to be fair?

```
# Installing the required libraries
!pip install sklearn
!pip install aif360
!pip install fairlearn
!pip install sdv
!pip install cgan
!pip install imbalanced-learn
!pip install scikit-learn==0.23.1
```

```
Requirement already satisfied: pandas>=0.25.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/d
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whee
Requirement already satisfied: sdv in /usr/local/lib/python3.7/dist-packages (0.1
Requirement already satisfied: tqdm<5,>=4.15 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: graphviz<1,>=0.13.2 in /usr/local/lib/python3.7/di
Requirement already satisfied: pandas<2,>=1.1.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: Faker<15,>=10 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: ctgan<0.6,>=0.5.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: numpy<2,>=1.20.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: deepecho<0.4,>=0.3.0.post1 in /usr/local/lib/pytho
Requirement already satisfied: rdt<1.3.0,>=1.2.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: sdmetrics<0.8,>=0.7.0.dev0 in /usr/local/lib/pytho
Requirement already satisfied: copulas<0.8,>=0.7.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: cloudpickle<3.0,>=2.1.0 in /usr/local/lib/python3.
Requirement already satisfied: scipy<2,>=1.5.4 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: matplotlib<4,>=3.4.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: torch<2,>=1.8.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: scikit-learn<2,>=0.24 in /usr/local/lib/python3.7/
Requirement already satisfied: packaging<22,>=20 in /usr/local/lib/python3.7/dist
Requirement already satisfied: torchvision<1,>=0.9.0 in /usr/local/lib/python3.7/
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.7/d
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/pytho
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyyaml<6,>=5.4.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: psutil<6,>=5.7 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: plotly<6,>=5.10.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/lo
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dis
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-pack
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whee
ERROR: Could not find a version that satisfies the requirement cgan (from version
ERROR: No matching distribution found for cgan
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whee
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.7/dist-
Requirement already satisfied: scikit-learn>=0.24 in /usr/local/lib/python3.7/dis
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/d
```

```
# Load all necessary packages
import sys

from aif360.datasets import BinaryLabelDataset
from aif360.datasets import AdultDataset, GermanDataset, CompasDataset
from aif360.metrics import BinaryLabelDatasetMetric
from aif360.metrics import ClassificationMetric
from aif360.metrics.utils import compute_boolean_conditioning_vector

from aif360.algorithms.preprocessing.optim_preproc_helpers.data_preproc_functions\
    import load_preproc_data_adult, load_preproc_data_german, load_preproc_data_compas

from aif360.algorithms.preprocessing.lfr import LFR

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

from IPython.display import Markdown, display
import matplotlib.pyplot as plt
import numpy as np
```

```
WARNING:root:No module named 'tempeh': LawSchoolGPADataset will be unavailable. To ir
pip install 'aif360[LawSchoolGPA]'
```

```
dataset_used = "german"
protected_attribute_used = 2 # 1, 2
if dataset_used == "german":
    dataset_orig = GermanDataset()
    if protected_attribute_used == 1:
        privileged_groups = [{'sex': 1}]
        unprivileged_groups = [{'sex': 0}]
    else:
        privileged_groups = [{'age': 1}]
        unprivileged_groups = [{'age': 0}]

    for i in range(1000):
        if (dataset_orig.labels[i] == 2.0):
            dataset_orig.labels[i] = 0
        else:
            dataset_orig.labels[i] = 1

    dataset_orig.favorable_label = 1
    dataset_orig.unfavorable_label = 0
```

```
# Initial disparities in the original datasets

metric_orig = BinaryLabelDatasetMetric(dataset_orig,
                                             unprivileged_groups=unprivileged_groups,
                                             privileged_groups=privileged_groups)


print("#### %s original dataset\n"%dataset_used)

# Calculate the Disperate Impact Ratio
print("Disparate impact (of original labels) between unprivileged and privileged groups = 

# Calculate the Statistical Parity Difference
print("Difference in statistical parity (of original labels) between unprivileged and priv

# Calculate the Concisteny
print("Individual fairness metric from Zemel et.al. that measures how similar the labels a
```

```
#### german original dataset

Disparate impact (of original labels) between unprivileged and privileged groups = 0
Difference in statistical parity (of original labels) between unprivileged and privi
Individual fairness metric from Zemel et.al. that measures how similar the labels are
```

What do these numbers tell us?

The disparate impact ratio is fair and legal when it's between 0.8 and 1.25, where 1 is optimal. Our DI is 0.794 which means that it is underneath 0.8 and thus not considered as fair by law.

The statistical parity difference is considered as fair when it's close to zero. The difference in our dataset is 0.149 which is relatively close to the optimum but every value closer to 0 will be better.

The consisteny is optimal when it's 1. The consisteny of our dataset is 0.68 which is not close to 1.

Is our dataset fair? Not really. Since it is not fair by law and the consisteny is not close to 1.

Exercise 2

Split the data into a 30/70 test and training set using stratification. Train a model using the training set and compute values the following 2 fairness metrics (in addition to the values of the previous 3 metrics (DI Ratio, P diff. and consistency)):

- Equalized odds
- Predictive parity

What do these results tell you? Compute the accuracy of the model.

```
#Splitting train and test set
dataset_orig_train, dataset_orig_test = dataset_orig.split([0.7], shuffle=True)


#Scaling the dataset
scale_orig = StandardScaler()

X_train = scale_orig.fit_transform(dataset_orig_train.features)
X_test = scale_orig.transform(dataset_orig_test.features)

y_train = dataset_orig_train.labels.ravel()
y_test = dataset_orig_test.labels.ravel()

#Logistic Regression Training for each dataset
log_reg = LogisticRegression()

#Fitting the training set
log_reg.fit(X_train, y_train)

#Predicting test set labels
y_test_pred = log_reg.predict(X_test)
y_test_pred_proba = log_reg.predict_proba(X_test)


#Create a new version of the test set with predicted class labels
testset_pred = dataset_orig_test.copy()
testset_pred.labels = y_test_pred

#Construction 2
#both original test dataset with actual labels and the test dataset combined with predicte
classified_metric = ClassificationMetric(dataset_orig_test,
                                                 testset_pred,
                                                 unprivileged_groups=unprivileged_groups,
                                                 privileged_groups=privileged_groups)

#Checking Equalized Odds: average odds differecence, which is the avg. of differences in F
aeo = classified_metric.average_odds_difference()
print("Average equalized odds difference between unprivileged and privileged groups = %f" 

#Predictive parity difference: PPV difference between privileged and unprivileged groups.
ppd = classified_metric.positive_predictive_value(privileged=False) - classified_metric.po
print("Predictive Parity difference between unprivileged and privileged groups = %f" % ppd
```

```
Average equalized odds difference between unprivileged and privileged groups = -0.21
Predictive Parity difference between unprivileged and privileged groups = -0.059045
```

```
print("Standard accuracy of logistic regression trained on German dataset without any miti
```

```
Standard accuracy of logistic regression trained on German dataset without any mitiga
```

What do these results tell us?

In our model, the average equalized odds (AEO) is around 0.20. the AEO is optimal when it's 0. This means that there is a worse prediction for the sub groups and unfairness in the dataset.

The accuracy of the model is 0.76

Exercise 3

Use one of the bias mitigation algorithms that are implemented in aif360 to improve the model fairness and compute the fairness metrics values. How have the values of all 5 fairness properties changed? Compute the accuracy and compare the value with the obtained in the previous question.

```
from aif360.algorithms.preprocessing.lfr import LFR


# LFR itself contains logistic regression since it uses sigmoid functions
lfr_obj =LFR(unprivileged_groups=unprivileged_groups,
         privileged_groups=privileged_groups,
         k=5, Ax=0.01, Ay=1.0, Az=50.0, verbose=1)

TR = lfr_obj.fit(dataset_orig_train, maxiter=5000, maxfun=5000)


#scaled dataset together with its labels is needed
dataset_orig_train.features = scale_orig.fit_transform(dataset_orig_train.features)
dataset_orig_test.features = scale_orig.transform(dataset_orig_test.features)


# Transform training data and align features
dataset_transf_train = TR.transform(dataset_orig_train)
# Before proceeding to the next step, make sure that LFR doesn't solve the bias
# using the trivial solution (converting all the labels to the preferable label)
from collections import Counter
c = Counter(dataset_transf_train.labels.ravel())
c
```

```
step: 0, loss: 6306.8327353958175, L_x: 630620.2621563061,  L_y: 0.627652866573587,
step: 250, loss: 6306.832735396124, L_x: 630620.2621563952,  L_y: 0.6276528664367913,
step: 500, loss: 6305.688914415635, L_x: 630507.5904722863,  L_y: 0.6109794377948053,
step: 750, loss: 6305.594447199212, L_x: 630498.113067416,  L_y: 0.6113200359297114,
step: 1000, loss: 6303.501273318389, L_x: 630287.8578050908,  L_y: 0.621512963465458,
step: 1250, loss: 6292.369638148416, L_x: 629170.2540122072,  L_y: 0.667057122543467
step: 1500, loss: 6252.162321508862, L_x: 625146.0543736017,  L_y: 0.701777768895753,
step: 1750, loss: 6252.162321508862, L_x: 625146.0543736017,  L_y: 0.701777768895753
step: 2000, loss: 6246.6065493795, L_x: 624585.623529826,  L_y: 0.7503140802095247,
```

```
step: 2250, loss: 6246.401675034555, L_x: 624119.7146284909,  L_y: 5.204528749515158,
step: 2500, loss: 6246.60626838619, L_x: 624585.5986193798,  L_y: 0.7502821913613327,
step: 2750, loss: 6244.13792962485, L_x: 624352.6162819872,  L_y: 0.6117668046097703,
step: 3000, loss: 6220.181908320987, L_x: 620936.8812304251,  L_y: 10.81309601673514
step: 3250, loss: 6220.181908320987, L_x: 620936.8812304251,  L_y: 10.81309601673514
step: 3500, loss: 6244.132263333159, L_x: 624352.049209714,  L_y: 0.6117712356504332,
step: 3750, loss: 6227.117876634045, L_x: 622642.2952562518,  L_y: 0.69492407152667
step: 4000, loss: 6216.000595921095, L_x: 621504.8405370493,  L_y: 0.95219055060144
step: 4250, loss: 6212.521226020962, L_x: 621126.2083211516,  L_y: 1.259142809446385,
step: 4500, loss: 3828.607507726485, L_x: 381755.3942123669,  L_y: 10.54395267477000
step: 4750, loss: 5470.2437573232055, L_x: 546889.9271834907,  L_y: 1.34448548829818
step: 5000, loss: 5470.2437573232055, L_x: 546889.9271834907,  L_y: 1.34448548829818
Counter({0.0: 424, 1.0: 276})
```

```
metric_transf_train = BinaryLabelDatasetMetric(dataset_transf_train,
                                               unprivileged_groups = unprivileged_groups,
                                               privileged_groups = privileged_groups)


# Calculate the Disperate Impact Ratio
print("Disparate impact (of original labels) between unprivileged and privileged groups = 

# Calculate the Statistical Parity Difference
print("Difference in statistical parity (of original labels) between unprivileged and priv

# Calculate the Concisteny
print("Individual fairness metric from Zemel et.al. that measures how similar the labels a
```

```
Disparate impact (of original labels) between unprivileged and privileged groups = 0
Difference in statistical parity (of original labels) between unprivileged and privil
Individual fairness metric from Zemel et.al. that measures how similar the labels are
```

```
# If the counter in the previous cell shows more than one class, proceed to this step
# Otherwise, you cannot train model


X_train_trans =dataset_transf_train.features
X_test_trans = dataset_orig_test.features

y_train_trans = dataset_transf_train.labels.ravel()
y_test_trans = dataset_orig_test.labels.ravel()
```

```
#Logistic Regression Training with the transformed dataset
trans_lr = LogisticRegression(solver='liblinear')

#fitting the model
trans_lr.fit(X_train_trans, y_train_trans)

#Predicting test set labels
y_test_trans_pred = trans_lr.predict(X_test_trans)
y_test_trans_pred_proba = trans_lr.predict_proba(X_test_trans)


# Create a new version of the transformed test set with predicted class labels
testset_pred_trans = dataset_orig_test.copy()
testset_pred_trans.labels = y_test_trans_pred


metric_trans_test = BinaryLabelDatasetMetric(testset_pred_trans,
                                             unprivileged_groups=unprivileged_groups,
                                             privileged_groups=privileged_groups)

classified_trans_test = ClassificationMetric(dataset_orig_test,
                                                 testset_pred_trans,
                                                 unprivileged_groups=unprivileged_groups,
                                                 privileged_groups=privileged_groups)



#Disparate Impact ratio between privileged and unprivileged groups.
deb_di_t = classified_trans_test.disparate_impact()
print("Disparate impact ratio between unprivileged and privileged groups = %f" % deb_di_t)

#Statistical parity difference between privileged and unprivileged groups.
deb_spd_t = classified_trans_test.statistical_parity_difference()
print("Statistical parity difference between unprivileged and privileged groups = %f" % de

#Individual Fairness: 1)Consistency, 2) Euclidean Distance between individuals.
print("Consistency of indivuals' predicted labels = %f" % metric_trans_test.consistency())

#Predictive parity difference: PPV difference between privileged and unprivileged groups.
deb_ppd_t = classified_trans_test.positive_predictive_value(privileged=False) - classified
print("Predictive Parity difference between unprivileged and privileged groups = %f" % deb

#Checking Equalized Odds: average odds differecence, which is the avg. of differences in F
deb_aeo_t = classified_trans_test.average_odds_difference()
print("Average equalized odds difference between unprivileged and privileged groups = %f"


print("Standard accuracy of logistic regression trained on test set with debiasing = %f" %
```

```
Disparate impact ratio between unprivileged and privileged groups = 1.299248
Statistical parity difference between unprivileged and privileged groups = 0.129305
Consistency of indivuals' predicted labels = 0.762667
Predictive Parity difference between unprivileged and privileged groups = -0.168155
Average equalized odds difference between unprivileged and privileged groups = 0.148
Standard accuracy of logistic regression trained on test set with debiasing = 0.5033
```

Compare the values

Now we can compare the values of the data before mitigation and the data after mitigation.

Disparate Impact Ratio

- Before Bias Mitigation: 0.794826
- After Bias Mitigation: 0.798922

Statistical Parity Difference

- Before Bias Mitigation: -0.149448
- After Bias Mitigation: -0.113971

Consistency

- Before Bias Mitigation: 0.681600
- After Bias Mitigation: 0.75400

Equalized odds

- Before Bias Mitigation: -0.076385
- After Bias Mitigation: -0.1199389

Predictive Parity Distance

- Before Bias Mitigation: -0.210636
- After Bias Mitigation: -0.214286

Accuracy

- Before Bias Mitigation: 0.763333
- After Bias Mitigation: 0.503

The accuracy of the model decreased by almost 0.26.

The performance of the DI, SPD and consistency increased. This means that the fairness of the dataset slightly increased by watching these values.

If we look to Equalized odds and preditive parity distance, the fairness decreased.

Question 4

Synthesise a new dataset by oversampling the underrepresented classes. For this, you can use any technique discussed in the lecture such as SMOTE or GANs. Train the model in exactly the same way (as you did in Exercise 2) on this new dataset. How have the values of all 5 fairness measures changed? Compute the accuracy of the model and compare the value with the accuracy value that was obtained in question 2.

```
from collections import Counter
german_data = GermanDataset()
c = Counter(german_data.labels.ravel())
c
```

```
Counter({1.0: 700, 2.0: 300})
```

```
from imblearn.over_sampling import SMOTE
```

```
# use SMOTE to oversample the underrepresented class
oversample = SMOTE()
X_OS, y_OS = oversample.fit_resample(X_train, y_train)
```

```
# logistic regression fit to oversampled data
logreg = LogisticRegression()
logreg.fit(X_OS, y_OS)
```

```
LogisticRegression()
```

```
y_test_smote_pred = logreg.predict(X_test)

testset_pred_smote = dataset_orig_test.copy()
testset_pred_smote.labels = y_test_smote_pred
```

```
metric_transf_train = BinaryLabelDatasetMetric(testset_pred_smote,
                                             unprivileged_groups = unprivileged_groups,
                                             privileged_groups = privileged_groups)

classified_metric = ClassificationMetric(dataset_orig_test,
                                                 testset_pred_smote,
                                                 unprivileged_groups=unprivileged_groups,
                                                 privileged_groups=privileged_groups)

print("Disparate impact ratio (of transformed labels) between unprivileged and privileged
print("Difference in statistical parity (of transformed labels) between unprivileged and p
print("Individual fairness metric 'consistency' that measures how similar the labels are f
ppd = classified_metric.positive_predictive_value(privileged=False) - classified_metric.po
print("Predictive Parity difference between unprivileged and privileged groups = %f" % ppd
aeo = classified_metric.average_odds_difference()
print("Average equalized odds difference between unprivileged and privileged groups = %f"
```

```
Disparate impact ratio (of transformed labels) between unprivileged and privileged gr
Difference in statistical parity (of transformed labels) between unprivileged and pri
Individual fairness metric 'consistency' that measures how similar the labels are for
Predictive Parity difference between unprivileged and privileged groups = -0.079667
Average equalized odds difference between unprivileged and privileged groups = -0.20
```

```
print("Accuracy: %.3f" % accuracy_score(y_test, y_test_smote_pred))
```

```
Accuracy: 0.747
```