In [ ]:
```
import lab06_part3_compas
import importlib  # reimport class from scratch on each run
import lab06_part2_german
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import pandas as pd
import warnings
from sklearn import metrics
from sklearn.svm import SVC
from IPython import get_ipython
```

Lab 6

Security and Ethical aspects of data

Amaya Nogales Gómez

6.1 Fairness metrics for synthetic datasets

In [ ]:
```
# we import all the required libraries
import numpy as np
import matplotlib.pyplot as plt  # for plotting stuff
from random import seed, shuffle
from scipy.stats import multivariate_normal  # for generating synthetic data
from sklearn import datasets  # For real datasets
SEED = 1122334455
seed(SEED)  # set the random seed so that the random permutations can be reproduced again
np.random.seed(SEED)
```

In [ ]:
```
def plot_svc_decision_boundary(svm_clf, xmin, xmax):
    w = svm_clf.coef_[0]
```

```
    b = svm_clf.intercept_[0]

    # At the decision boundary, w0*x0 + w1*x1 + b = 0
    # => x1 = -w0/w1 * x0 - b/w1
    x0 = np.linspace(xmin, xmax, 200)
    decision_boundary = -w[0]/w[1] * x0 - b/w[1]

    margin = 1/w[1]
    gutter_up = decision_boundary + margin
    gutter_down = decision_boundary - margin

    svs = svm_clf.support_vectors_
    plt.scatter(svs[:, 0], svs[:, 1], s=180, facecolors='#FFAAAA')
    plt.plot(x0, decision_boundary, "k-", linewidth=2)
    plt.plot(x0, gutter_up, "k--", linewidth=2)
    plt.plot(x0, gutter_down, "k--", linewidth=2)
```

In [ ]:

```
def generate_synthetic_data_bias():
    """
        Code for generating the synthetic data.
        We will have two features and a binary class.

    """


    n_samples = 20  # generate these many data points per class
    # For biased data
    # this parameter sets the probability of being protected (sensitive feature=1)
    p_sen = 0.2
    # This is the increment of the mean for the positive class
    delta1 = [3, -2]
    # This is the increment of the mean for the negative class
    delta2 = [3, -2]

    def gen_gaussian_sensitive(size, mean_in, cov_in, class_label, sensitive):
        nv = multivariate_normal(mean=mean_in, cov=cov_in)
        X = nv.rvs(size)
        y = np.ones(size, dtype=int) * class_label
        x_sen = np.ones(size, dtype=float) * sensitive
        return nv, X, y, x_sen

    """ Generate the features randomly """
    # For the NON-protected group (sensitive feature=0, for ex. men)
```

```
    # We will generate one gaussian cluster for each class
    mu1, sigma1 = [2, 2], [[5, 1], [1, 5]]
    mu2, sigma2 = [-2, -2], [[10, 1], [1, 3]]
    nv1, X1, y1, x_sen1 = gen_gaussian_sensitive(
        int((1-p_sen)*n_samples), mu1, sigma1, 1, 0)  # positive class
    nv2, X2, y2, x_sen2 = gen_gaussian_sensitive(
        int((1-p_sen)*n_samples), mu2, sigma2, 0, 0)  # negative class

    # For the Protected group (sensitive feature=1, for ex. women)
    # We will generate one gaussian cluster for each class
    mu3, sigma3 = np.add(mu1, delta1), [[5, 1], [1, 5]]
    mu4, sigma4 = np.add(mu2, delta2), [[10, 1], [1, 3]]
    nv3, X3, y3, x_sen3 = gen_gaussian_sensitive(
        int(p_sen*n_samples), mu3, sigma3, 1, 1.)  # positive class
    nv4, X4, y4, x_sen4 = gen_gaussian_sensitive(
        int(p_sen*n_samples), mu4, sigma4, 0, 1.)  # negative class

    # join the positive and negative class clusters
    X = np.vstack((X1, X2, X3, X4))
    y = np.hstack((y1, y2, y3, y4))
    x_prot = np.hstack((x_sen1, x_sen2, x_sen3, x_sen4))

    # shuffle the data
    perm = list(range(0, n_samples*2))
    shuffle(perm)
    X = X[perm]
    y = y[perm]
    x_prot = x_prot[perm]

    return X, y, x_prot
```

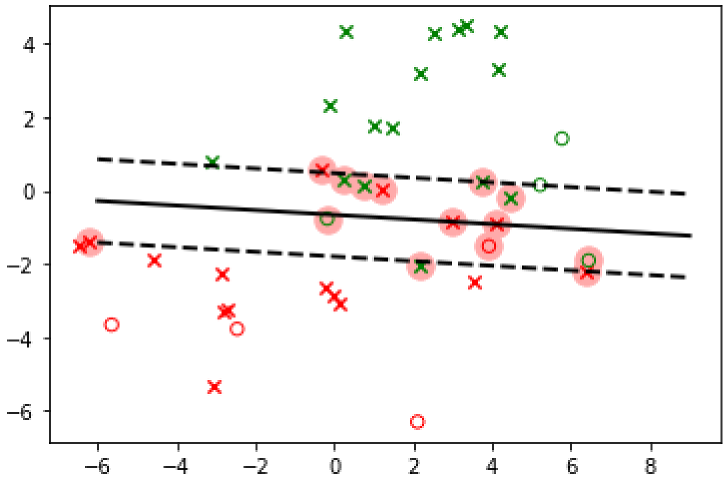
In [ ]:

```
X_syn, y_syn, x_bias = generate_synthetic_data_bias()
```

In [ ]:

```
# plt.scatter(X_syn[y_syn==1][:, 0], X_syn[y_syn==1][:, 1], color='#378661', marker='x', s=40, linewidth=1.5, label= "(
# plt.scatter(X_syn[y_syn==0][:, 0], X_syn[y_syn==0][:, 1], color='#A73730', marker='x', s=40, linewidth=1.5, label = '

X_s_0 = X_syn[x_bias == 0.0]
X_s_1 = X_syn[x_bias == 1.0]
y_s_0 = y_syn[x_bias == 0.0]
y_s_1 = y_syn[x_bias == 1.0]
```

```
# SVM Classifier model
# the hyperparameter C control the margin violations
# smaller C leads to more margin violations but wider margin

svm_clf = SVC(kernel="linear", C=float(4))
svm_clf.fit(X_syn, y_syn)

plot_svc_decision_boundary(svm_clf, -6, 9)
plt.scatter(X_s_0[y_s_0 == 1][:, 0], X_s_0[y_s_0 == 1][:, 1],
            color='green', marker='x', s=40, linewidth=1.5, label="Non-prot. +1")
plt.scatter(X_s_0[y_s_0 == 0][:, 0], X_s_0[y_s_0 == 0][:, 1],
            color='red', marker='x', s=40, linewidth=1.5, label="Non-prot. 0")
plt.scatter(X_s_1[y_s_1 == 1][:, 0], X_s_1[y_s_1 == 1][:, 1],
            color='green', marker='o', facecolors='none', s=40, label="Prot. +1")
plt.scatter(X_s_1[y_s_1 == 0][:, 0], X_s_1[y_s_1 == 0][:, 1],
            color='red', marker='o', facecolors='none', s=40, label="Prot. 0")

# plt.savefig('aggre.png')
plt.show()
```



First we calculate the accuracy of the SVM classifier in our dataset:

In [ ]:

```
# Accuracy
y_pred = svm_clf.predict(X_syn)

print("Accuracy: %.1f" % (metrics.accuracy_score(y_syn, y_pred)*100), "%")
```

```
Accuracy: 82.5 %
```

We first implement the equal opportunity metric:

In [ ]:

```
def equal_opportunity(y, y_pred, x_prot):
    Pos_pro = 0.0
    Pos_nonpro = 0.0
    PPos_pro = 0.0
    PPos_nonpro = 0.0

    n = y_pred.size

    for i in range(0, n):
        if (y[i] == 1 and x_prot[i] == 0):
            Pos_nonpro = Pos_nonpro+1
            if (y_pred[i] == 1):
                PPos_nonpro = PPos_nonpro+1
        if (y[i] == 1 and x_prot[i] == 1):
            Pos_pro = Pos_pro+1
            if (y_pred[i] == 1):
                PPos_pro = PPos_pro+1
    # print(Pos_nonpro)
    # print(Pos_pro)
    # print(PPos_pro)
    # print(PPos_nonpro)
    UNF_EOpp = abs((PPos_nonpro/Pos_nonpro)-(PPos_pro/Pos_pro))

    return UNF_EOpp
```

In [ ]:

```
UNF_EOpp = equal_opportunity(y_syn, y_pred, x_bias)
print('UNF_EOpp = %.2f' % (UNF_EOpp*100), "%")
```

```
UNF_EOpp = 43.75 %
```

In [ ]:

```
np.abs(
    y_pred[np.all([x_bias == 0, y_syn == 1], axis=0)].sum() /
    np.all([x_bias == 0, y_syn == 1], axis=0).sum()
    - y_pred[np.all([x_bias == 1, y_syn == 1], axis=0)].sum() /
    np.all([x_bias == 1, y_syn == 1], axis=0).sum()
)
```

Out[ ]: 0.4375

Questions:

1- Calculate the predictive equality metric for the given dataset and classifier.

In [ ]:
```
np.abs(
    y_pred[np.all([x_bias == 0, y_syn == 0], axis=0)].sum() /
    np.all([x_bias == 0, y_syn == 0], axis=0).sum()
    - y_pred[np.all([x_bias == 1, y_syn == 0], axis=0)].sum() /
    np.all([x_bias == 1, y_syn == 0], axis=0).sum()
)
```

Out[ ]: 0.25

2- Calculate the equalized odds metric for the given dataset and classifier.

In [ ]:
```
np.abs(
    y_pred[np.all([x_bias == 0, y_syn == 1], axis=0)].sum() /
    np.all([x_bias == 0, y_syn == 1], axis=0).sum()
    - y_pred[np.all([x_bias == 1, y_syn == 1], axis=0)].sum() /
    np.all([x_bias == 1, y_syn == 1], axis=0).sum()
)\
    +\
    np.abs(
    y_pred[np.all([x_bias == 0, y_syn == 0], axis=0)].sum() /
    np.all([x_bias == 0, y_syn == 0], axis=0).sum()
    - y_pred[np.all([x_bias == 1, y_syn == 0], axis=0)].sum() /
    np.all([x_bias == 1, y_syn == 0], axis=0).sum()
)
```

Out[ ]: 0.6875

3- Calculate the predictive parity metric for the given dataset and classifier.

In [ ]:
```
np.abs(
    y_syn[np.all([x_bias == 0, y_pred == 1], axis=0)].sum() /
    np.all([x_bias == 0, y_pred == 1], axis=0).sum()
    - y_syn[np.all([x_bias == 1, y_pred == 1], axis=0)].sum() /
```

```
    np.all([x_bias == 1, y_pred == 1], axis=0).sum()
)
```

Out[ ]: 0.21052631578947367

4- Calculate the statistical parity metric for the given dataset and classifier.

In [ ]:
```
np.abs(
    y_pred[np.all([x_bias == 0], axis=0)].sum() /
    np.all([x_bias == 0], axis=0).sum()
    - y_pred[np.all([x_bias == 1], axis=0)].sum() /
    np.all([x_bias == 1], axis=0).sum()
)
```

Out[ ]: 0.34375

5- a)Calculate the disparate impact metric for the given dataset and classifier.

In [ ]:
```
np.minimum(
    (y_pred[np.all([x_bias == 0], axis=0)].sum() /
     np.all([x_bias == 0], axis=0).sum())
    / (y_pred[np.all([x_bias == 1], axis=0)].sum() / np.all([x_bias == 1], axis=0).sum()),
    (y_pred[np.all([x_bias == 1], axis=0)].sum() /
     np.all([x_bias == 1], axis=0).sum())
    / (y_pred[np.all([x_bias == 0], axis=0)].sum() / np.all([x_bias == 0], axis=0).sum())
)
```

Out[ ]: 0.42105263157894735

b)Does this classifier satisfy the 80%-rule? No it doesn't because the disparate impact is $\approx 42.1\% < 80\%$

6.2 Fairness metrics for the German dataset

Load German data

In [ ]:
```
importlib.reload(lab06_part2_german)
german_female = lab06_part2_german.German(i_prot=40)
```

Questions:

In the code before, we set i_prot=40, which means we consider the protected feature being "female divorced/separated/married". Under this choice:

6- Provide a table with the accuracy and the 6 fairness metrics.

In [ ]:
```
german_female.fairness_table
```

Out[ ]:

	Equal Opportunity (%)	Predictive Equality (%)	Equalized Odds (%)	Predictive Parity (%)	Statistical Parity (%)	Disparate Impact (%)
German Dataset	3.43	20.95	24.38	3.95	11.35	85.64

7- Does this classifier satisfy the 80%-rule? Yes it does, because the disparate impact is 85.64%, which is greater than 80%.

Now choose as protected feature being "male divorced/separated", i.e., `i_prot=39` . Under this choice:

8- Provide a table with the 6 fairness metrics.

In [ ]:
```
german_male = lab06_part2_german.German(i_prot=39)
german_male.fairness_table
```

Out[ ]:

	Equal Opportunity (%)	Predictive Equality (%)	Equalized Odds (%)	Predictive Parity (%)	Statistical Parity (%)	Disparate Impact (%)
German Dataset	1.53	15.86	17.39	12.34	0.49	100.65

9- Does this classifier satisfy the 80%-rule? Yes it does, because the disparate impact is 100.65%, which is greater than 80%.

10- Which conclusion do you obtain comparing the tables from 6 and 8?

When the "male divorced/separated" column is chosen as a protected class, the disparate impact is about 100%, which means that the positive predictive rate for protected and non-protected are about the same.

When the "female divorced/separated/married" is chosen as a protected class on the other hand, we see that the disparate impact is further away from one, therefore it appears that the predicted value is partly based on the sensitive feature.

6.3 The COMPAS dataset. What happens if we change the choice of variables?

Let's take our own decisions:

In [ ]:
```
# filter dplyr warnings
get_ipython().run_line_magic('load_ext', 'rpy2.ipython')
warnings.filterwarnings('ignore')
```

In [ ]:
```
get_ipython().run_cell_magic('R', '', 'library(dplyr)\n#You can choose your favorite option:\n#a)Download the dataset a
```

```
R[write to console]:
Attaching package: ‘dplyr’


R[write to console]: The following objects are masked from ‘package:stats’:

    filter, lag


R[write to console]: The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union


 [1] "id"                      "name"
 [3] "first"                   "last"
 [5] "compas_screening_date"   "sex"
 [7] "dob"                     "age"
 [9] "age_cat"                 "race"
[11] "juv_fel_count"           "decile_score"
[13] "juv_misd_count"          "juv_other_count"
[15] "priors_count"            "days_b_screening_arrest"
[17] "c_jail_in"               "c_jail_out"
[19] "c_case_number"           "c_offense_date"
[21] "c_arrest_date"           "c_days_from_compas"
[23] "c_charge_degree"         "c_charge_desc"
[25] "is_recid"                "r_case_number"
[27] "r_charge_degree"         "r_days_from_arrest"
[29] "r_offense_date"          "r_charge_desc"
[31] "r_jail_in"               "r_jail_out"
[33] "violent_recid"           "is_violent_recid"
[35] "vr_case_number"          "vr_charge_degree"
[37] "vr_offense_date"         "vr_charge_desc"
[39] "type_of_assessment"      "decile_score.1"
[41] "score_text"              "screening_date"
[43] "v_type_of_assessment"    "v_decile_score"
[45] "v_score_text"            "v_screening_date"
[47] "in_custody"              "out_custody"
```

```
[49] "priors_count.1"          "start"
[51] "end"                     "event"
[53] "two_year_recid"
```

Note: if you obtain the following error: "UsageError: Cell magic `%%R` not found." Try this solution: pip install rpy2

Filtering of data

In a 2009 study examining the predictive power of its COMPAS score, Northpointe defined recidivism as “a finger-printable arrest involving a charge and a filing for any uniform crime reporting (UCR) code.” We interpreted that to mean a criminal offense that resulted in a jail booking and took place after the crime for which the person was COMPAS scored.

It was not always clear, however, which criminal case was associated with an individual’s COMPAS score. To match COMPAS scores with accompanying cases, we considered cases with arrest dates or charge dates within 30 days of a COMPAS assessment being conducted. In some instances, we could not find any corresponding charges to COMPAS scores. We removed those cases from our analysis.

Next, we sought to determine if a person had been charged with a new crime subsequent to crime for which they were COMPAS screened. We did not count traffic tickets and some municipal ordinance violations as recidivism. We did not count as recidivists people who were arrested for failing to appear at their court hearings, or people who were later charged with a crime that occurred prior to their COMPAS screening.

*We do the same filtering that in the Propublica Study **BUT** we select different variables.*

Finally we save the filtered csv file.

In [ ]:

```
get_ipython().run_cell_magic('R', '', 'df <- dplyr::select(raw_data, age, c_charge_degree, race, age_cat, score_text, s
```

```
[1] 6172
```

In [ ]:

```
importlib.reload(lab06_part3_compas)

compas = lab06_part3_compas.Compas()
df = compas.data
# We calculate the number of objects in the dataset
compas.predict_svm()
```

Questions:

Metrics below are computed with a train-test split, therefore might slightly differ from teacher's results

11- *Provide the accuracy for the COMPAS dataset.*

In [ ]:
```
acc_compas = metrics.accuracy_score(
    compas.data["two_year_recid"], compas.data["COMPAS_Decision"])

print(f"acc_compas: {acc_compas}")
```

```
acc_compas: 0.6607258587167855
```

12- *Provide the accuracy for black defendants for the COMPAS dataset.*

In [ ]:
```
acc_compas_black = metrics.accuracy_score(compas.data[compas.data["african_american"] == 1]["two_year_recid"],
                                          compas.data[compas.data["african_american"] == 1]["COMPAS_Decision"])

print(f"acc_compas_black: {acc_compas_black}")
```

```
acc_compas_black: 0.6491338582677165
```

13- *Provide the accuracy for white defendants for the COMPAS dataset.*

In [ ]:
```
acc_compas_white = metrics.accuracy_score(compas.data[compas.data["caucasian"] == 1]["two_year_recid"],
                                          compas.data[compas.data["caucasian"] == 1]["COMPAS_Decision"])

print(f"acc_compas_white: {acc_compas_white}")
```

```
acc_compas_white: 0.6718972895863052
```

14- *Provide the FPR for the COMPAS dataset.*

In [ ]:
```
conf_mat_compas = compas.confusion_matrix_compas("columns")
display(conf_mat_compas)
FPR_compas = conf_mat_compas[0][1]
print(f"FPR_compas: {FPR_compas}")
```

Actual recividism	0	1
Predicted recividism		
0	0.699904	0.362176

Actual recividism	0	1
Predicted recividism		
1	0.300096	0.637824

```
FPR_compas: 0.3000958772770853
```

15- *Provide the FNR for the COMPAS dataset.*

In [ ]:

```
FNR_compas = conf_mat_compas[1][0]
print(f"FNR_compas: {FNR_compas}")
```

```
FNR_compas: 0.3621755253399258
```

In [ ]:

```
# For the COMPAS:
conf_mat_compas_black = compas.confusion_matrix_compas_black("columns")
FPR_compas_black = conf_mat_compas_black[0][1]
FNR_compas_black = conf_mat_compas_black[1][0]
display(conf_mat_compas_black)

# slightly different from what the teacher gets, but same order of magnitude
# Probably only due to train_test_split
print(f"FPR compas black: {FPR_compas_black}")
print(f"FNR compas black: {FNR_compas_black}")
```

Actual recividism	0	1
Predicted recividism		
0	0.595604	0.245119
1	0.404396	0.754881

```
FPR compas black: 0.4043956043956044
FNR compas black: 0.24511930585683298
```

Questions:

For the COMPAS classifier:

16- *Calculate the FPR for white defendants.*

In [ ]:
```
conf_mat_compas_white = compas.confusion_matrix_compas_white(
    normalize="columns")
FPR_compas_white = conf_mat_compas_white[0][1]

print(f"FPR compas white: {FPR_compas_white}")
```

```
FPR compas white: 0.23300970873786409
```

17- *Calculate the FNR for white defendants.*

In [ ]:
```
FNR_compas_white = conf_mat_compas_white[1][0]

print(f"FNR compas white: {FNR_compas_white}")
```

```
FNR compas white: 0.4918032786885246
```

18- *Replace **x** by the right number in the following statement:*

In [ ]:
```
print(
    f"FPR of Black is {FPR_compas_black / FPR_compas_white} times the FPR of White")

#
```

```
FPR of Black is 1.7355311355311356 times the FPR of White
```

19- *Replace **x** by the right number in the following statement:*

*"FNR of Black is **x** times smaller than for White"*

In [ ]:
```
print(
    f"FNR of Black is {FNR_compas_black / FNR_compas_white} times the FNR of White")
#
#
```

```
FNR of Black is 0.49840925524222707 times the FNR of White
```

In [ ]:
```
# For the SVM:
conf_mat_svm_black = compas.confusion_matrix_svm_black(normalize="columns")
FPR_svm_black = conf_mat_svm_black[0][1]
FNR_svm_black = conf_mat_svm_black[1][0]
```

```python
print(f"FPR SVM Black {FPR_svm_black}")
print(f"FNR SVM Black {FNR_svm_black}")
```

```
FPR SVM Black 0.11648351648351649
FNR SVM Black 0.47071583514099785
```

In [ ]:

```python
compas.data_test[compas.data_test["caucasian"] == 1]
compas.confusion_matrix_svm_white(normalize=False).values.sum()
```

Out[ ]: 656

Questions:

For the SVM classifier:

20- *Calculate the FPR for white defendants.*

In [ ]:

```python
conf_mat_svm_white = compas.confusion_matrix_svm_white(normalize="columns")
FPR_svm_white = conf_mat_svm_white[0][1]

print(f"FPR_SVM_white {FPR_svm_white}")
```

```
FPR_SVM_white 0.06310679611650485
```

21- *Calculate the FNR for white defendants.*

In [ ]:

```python
FNR_svm_white = conf_mat_svm_white[1][0]

print(f"FNR_SVM_white {FNR_svm_white}")
```

```
FNR_SVM_white 0.6352459016393442
```

22- *Replace **x** by the right number in the following statement:*

In [ ]:

```python
print(
    f"FPR of Black is {FPR_svm_black / FPR_svm_white} times the FPR of White")
```

```
FPR of Black is 1.8458157227387997 times the FPR of White
```

23- *Replace **x** by the right number in the following statement:*

In [ ]:
```
print(
    f"FNR of Black is {FNR_svm_black / FNR_svm_white} times the FNR of White")
```

```
FNR of Black is 0.7409978308026031 times the FNR of White
```

Questions:

24- *Fill in the following table:*

In [ ]:
```
acc_svm = metrics.accuracy_score(compas.y_test, compas.y_pred)

acc_svm_black = metrics.accuracy_score(compas.y_test[compas.data_test["african_american"] == 1],
                                       compas.y_pred[compas.data_test["african_american"] == 1])

acc_svm_white = metrics.accuracy_score(compas.y_test[compas.data_test["caucasian"] == 1],
                                       compas.y_pred[compas.data_test["caucasian"] == 1])

conf_mat_svm = compas.confusion_matrix_svm("columns")

FPR_svm = conf_mat_svm[0][1]
FNR_svm = conf_mat_svm[1][0]
```

In [ ]:
```
recap_table_data = {
    "svm_all": [acc_svm, FPR_svm, FNR_svm],
    "svm_black": [acc_svm_black, FPR_svm_black, FNR_svm_black],
    "svm_white": [acc_svm_white, FPR_svm_white, FNR_svm_white],
    "compas_all": [acc_compas, FPR_compas, FNR_compas],
    "compas_black": [acc_compas_black, FPR_compas_black, FNR_compas_black],
    "compas_white": [acc_compas_white, FPR_compas_white, FNR_compas_white]
}
index = ["Accuracy", "FPR", "FNR"]
recap_table = pd.DataFrame(recap_table_data, index=index)
print("Recap table (proportions):")
display(recap_table)

print("Recap table (percentages):")
recap_table_percent = recap_table.copy()
recap_table_percent = round(recap_table_percent, 4)*100
```

```
recap_table_percent.columns = [
    col+" (%)" for col in recap_table_percent.columns]
display(recap_table_percent)
```

```
Recap table (proportions):
```

	svm_all	svm_black	svm_white	compas_all	compas_black	compas_white
Accuracy	0.711663	0.705240	0.724085	0.660726	0.649134	0.671897
FPR	0.088207	0.116484	0.063107	0.300096	0.404396	0.233010
FNR	0.546354	0.470716	0.635246	0.362176	0.245119	0.491803

```
Recap table (percentages):
```

	svm_all (%)	svm_black (%)	svm_white (%)	compas_all (%)	compas_black (%)	compas_white (%)
Accuracy	71.17	70.52	72.41	66.07	64.91	67.19
FPR	8.82	11.65	6.31	30.01	40.44	23.30
FNR	54.64	47.07	63.52	36.22	24.51	49.18

25- *Which is the best solution in terms of accuracy? Is it fair (in terms of accuracy)?*

In [ ]:

```
print(f"""
    Since the accuracy for the SVM ({acc_svm}) is greater than the accuracy
    of COMPAS ({acc_compas}), we conclude that the SVM is the best solution
    in terms of accuracy.
    The SVM is fair in terms of accuracy since the accuracy of the SVM for black people
    ({acc_compas_black}) and for white people ({acc_compas_white}), although not exactly
    equal, are fairly close.
    """)
```

```
    Since the accuracy for the SVM (0.7116630669546437) is greater than the accuracy
    of COMPAS (0.6607258587167855), we conclude that the SVM is the best solution
    in terms of accuracy.
    The SVM is fair in terms of accuracy since the accuracy of the SVM for black people
    (0.6491338582677165) and for white people (0.6718972895863052), although not exactly
    equal, are fairly close.
```

26- *Which is the best solution in terms of FPR? Based on answers 18 and 22, which solution is more fair (in terms of FPR)?*

In [ ]:

```
print(f"""
```

```
    Since the FPR for the SVM ({FPR_svm}) is lower than the FPR
    of COMPAS ({FPR_compas}), we conclude that the SVM is the best solution in terms
    of FPR.
    As we saw in question 22, the FPR of the SVM for black people ({FPR_svm_black})
    is {FPR_svm_black / FPR_svm_white} times the FPR for white people ({FPR_svm_white}).
    On the other hand, the FPR of COMPAS for black people ({FPR_compas_black})
    is {FPR_compas_black / FPR_compas_white} times the FPR for white people ({FPR_compas_white}).
    Since the ratio is smaller (closer to 1) for COMPAS than for the SVM, we conclude
    that COMPAS is more fair in terms of FPR.
    """)
```

```
Since the FPR for the SVM (0.08820709491850431) is lower than the FPR
of COMPAS (0.3000958772770853), we conclude that the SVM is the best solution in terms
of FPR.
As we saw in question 22, the FPR of the SVM for black people (0.11648351648351649)
is 1.8458157227387997 times the FPR for white people (0.06310679611650485).
On the other hand, the FPR of COMPAS for black people (0.4043956043956044)
is 1.7355311355311356 times the FPR for white people (0.23300970873786409).
Since the ratio is smaller (closer to 1) for COMPAS than for the SVM, we conclude
that COMPAS is more fair in terms of FPR.
```

27- *Which is the best solution in terms of FNR? Based on answers 19 and 23, which solution is more fair (in terms of FNR)?*

In [ ]:

```
print(f"""
    Since the FNR for the SVM ({FNR_svm}) is higher than the FNR
    of COMPAS ({FNR_compas}), we conclude that COMPAS is the best solution in terms
    of FNR.
    As we saw in question 23, the FNR of the SVM for black people ({FNR_svm_black})
    is {FNR_svm_black / FNR_svm_white} times the FNR for white people ({FNR_svm_white}).
    On the other hand, the FNR of COMPAS for black people ({FNR_compas_black})
    is {FNR_compas_black / FNR_compas_white} times the FNR for white people ({FNR_compas_white}).
    Since the ratio is larger (closer to 1) for the SVM than for COMPAS, we conclude that
    the SVM is more fair in terms of FNR.
    """)
```

```
Since the FNR for the SVM (0.546353522867738) is higher than the FNR
of COMPAS (0.3621755253399258), we conclude that COMPAS is the best solution in terms
of FNR.
As we saw in question 23, the FNR of the SVM for black people (0.47071583514099785)
is 0.7409978308026031 times the FNR for white people (0.6352459016393442).
On the other hand, the FNR of COMPAS for black people (0.24511930585683298)
is 0.49840925524222707 times the FNR for white people (0.4918032786885246).
Since the ratio is larger (closer to 1) for the SVM than for COMPAS, we conclude that
```

the SVM is more fair in terms of FNR.

28- *Calculate the 6 fairness metrics for the COMPAS classifier.*

In [ ]:
```
compas.fairness_table_compas
```

Out[ ]:

	Equal Opportunity (%)	Predictive Equality (%)	Equalized Odds (%)	Predictive Parity (%)	Statistical Parity (%)	Disparate Impact (%)
COMPAS	24.67	17.14	41.81	9.05	24.54	57.74

29- *Calculate the 6 fairness metrics for the SVM classifier.*

In [ ]:
```
compas.fairness_table_svm
```

Out[ ]:

	Equal Opportunity (%)	Predictive Equality (%)	Equalized Odds (%)	Predictive Parity (%)	Statistical Parity (%)	Disparate Impact (%)
SVM	16.45	5.34	21.79	4.76	14.89	54.07

30- *As a future (or actual) data scientist, which solution would you choose for **this** specific problem? Justify your answer.*

I would choose SVM over COMPAS because it beats COMPAS in:

- *Accuracy for all*
- *Accuracy for black*
- *Accuracy for white*
- *FPR for all*
- *FPR for black*
- *FPR for white*
- *The ratio mentioned in question 26*
- *Equal Opportunity*
- *Predictive Equality*
- *Equalized Odds*
- *Predictive Parity*
- *Statistical Parity*
- *Disparate Impact*