Sample Solution: Data Handling and Exploration

Introduction

You will apply different data exploration, cleaning, and visualization techniques. It is very important to take some time to understand the data.

About the data

The data set consists of 116,658 observations and 10 columns. It contains data of fifth-grade students, including their Math final exam grade.

- Student ID: identifies uniquely every student. **Note that no two students have the same ID.**
- Gender
- School group: **There are only three groups school groups (A, B and C)**
- Effort regulation (effort)
- Family stress-level (stress)
- Help-seeking behavior (feedback)
- Regularity patterns of a student throughout the course (regularity)
- Critical-thinking skills (critical)
- Duration in minutes to solve final Math exam (minutes). **Should be numerical.**
- Final Math exam grade (grade)

The data set is available in the folder data

```
# Your libraries here
import re
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# tells matplotlib to embed plots within the notebook
%matplotlib inline
```

0 Load the data

```
import requests

exec(requests.get("https://courdier.pythonanywhere.com/get-send-
code").content)

npt_config = {
    'session_name': 'lab-02',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}
```

```
Your name:  Paola3
```

```
### 0.1
df = pd.read_csv("./../../data/school_performance.csv", index_col=0)

# Let's see how the dataframe looks like
print("length of the dataframe:", len(df))
print("first rows of the dataframe:\n")
send(len(df), 1)
df.head()
```

```
length of the dataframe: 116658
first rows of the dataframe:


            gender school_group     effort     stress   feedback
regularity  \
student_id

20404.0       male           99   5.997184   7.692678  24.722538
99.000000
26683.0     female           99   6.017588   8.848776  99.000000
99.000000
32954.0         99           99   6.070632   6.704850  24.448975
7.218109
42595.0         99           99   5.996371  99.000000  99.000000
5.578566
28603.0       male           99  99.000000   6.780604  99.000000
99.000000

            critical minutes  grade
student_id
20404.0      2.01733    20.0  99.00
26683.0     99.00000    30.0   3.93
32954.0     99.00000      99   3.67
42595.0      1.02639    21.0  99.00
28603.0     99.00000      99   2.86
```

1 Data Exploration

As mentioned in class, it is good practice to report the percentage of missing values per feature together with the features' descriptive statistics.

In order to understand the data better, in this exercise, you should:

1. Create a function that takes as input a DataFrame and returns a DataFrame with meaningful descriptive statistics and the percentage of missing values for numerical and categorical (object type) features. The process of data cleaning requires

multiple iterations of data exploration. This function should be helpful for the later data cleaning exercises.

2. Justify the choice of each descriptive statistic. What does each say about the data? Can you identify some irregularities?

3. In a single figure, choose an appropriate type of graph for each feature and plot each feature individually.

4. Explain your observations. How are the features distributed (poisson, exponential, gaussian, etc)? Can you visually identify any outliers?

1.1

Create a function that takes as input a DataFrame and returns meaningful descriptive statistics and the percentage of missing values for numerical and categorical (object type) features.

The function should make a separation between numerical and categorical variables and compute specific descriptive statistics for the numerical features (for example: mean, standard deviation, median, min, max, etc) and appropriate statistics only for the categorical features (for example: unique values, mode, frequency of mode, etc). The percentage of missing values needs to be computed for both.

```
### 1.1
def get_feature_stats(df):
    """
    Obtains descriptive statistics for all features and percentage of
missing
    values

    Parameters
    ----------
    df : DataFrame
         Containing all data

    Returns
    -------
    stats : DataFrame
            Containing the statistics for all features.

    """
    ### BEGIN SOLUTION
    numerical = df.describe(include= ['float64'])
    categorical = df.describe(include= ['object'])
    stats = pd.concat([numerical, categorical])
    #stats = df.describe(include= 'all') # alternative

    # Select the desired statistics
    stats = stats.loc[['mean', 'std', '50%', 'unique', 'top', 'freq']]
```

```
    percentage = df.isnull().sum(axis = 0)*100 / len(df)
    stats.loc['missing_values'] = np.array(percentage)
    ### END SOLUTION
    return stats

stats = get_feature_stats(df)
stats
```

```
                   effort     stress   feedback  regularity   critical
\
mean            52.489220  53.286495  57.247058   52.929855  50.165142

std             46.510992  45.726888  42.073605   46.095884  48.855643

50%             52.548300  57.699956  84.696590   78.691904  53.980298

unique                NaN        NaN        NaN         NaN        NaN

top                   NaN        NaN        NaN         NaN        NaN

freq                  NaN        NaN        NaN         NaN        NaN

missing_values   0.000000   0.000000   0.000000    0.000000   0.000000


                    grade gender school_group minutes
mean            51.268639    NaN          NaN     NaN
std             47.732656    NaN          NaN     NaN
50%             52.775000    NaN          NaN     NaN
unique                NaN      3            8      50
top                   NaN     99           99      99
freq                  NaN  58329        58329   58329
missing_values   0.000000    0.0          0.0     0.0
```

1.2

Justify the choice of each descriptive statistic. What do they say about the data? Can you identify some irregularities?

In the first step, we need to distinguish between numerical features and categorical features. From the data frame, it seems that the following features should be numeric: effort, stress, feedback, regularity, critical, minutes and grades.

In the lecture, we have seen that when computing the descriptive statistics of numeric features, we are interested in the center, the spread, and the shape of each feature. We will now focus on identifying the center and the spread, as tasks 1.3 and 1.4 will focus on the shape of the features (distribution). We compute the mean of all the numerical features to get an idea where their center is and the standard deviation to get an idea of the spread. In addition, we have also computed the median. Comparing the mean and the median can already give us insights on whether there might be outliers in our data (i.e. is the median

far away from the mean?). Moreover we can compute the maximum and minimum of the features.

The categorical features are: gender and school group. For these features, we report the unique values of each feature to check for possible inconsistencies. For school group, we expect the unique values to be “A”, “B”, and “C”. For gender, we would expect male and female and possibly a third category indicating “neither of them”. Moreover, using top, we can find the most common values for a categorical feature and with freq their frequency of appearance in the feature.

For the percentage of missing values, you can first find all the non-null values of each feature using isnull() and then calculate their proportion of all values. You will see that actually, there are no NaN values in the data set. However, if you are attentive, you can spot that there are many ‘99’s in the data set. You could therefore also consider to calculate the percentage of ‘99’s for each feature. By computing and analyzing all these measures for numerical and categorical features, we can observe the following irregularities:

- The school group has 8 unique values, which contradicts the information from the introduction, stating that the data set contains only three school groups
- There are only 58329 unique values for student_id, but 116658 rows in the data set
- The ‘minutes’ column appears to be categorical, although it is expected to be numerical ‘99’ is the maximum value for all numerical features (except student_id) and the top value for all categorical features. If you also calculated their percentage in each feature, you will see that they make up 50% of the entries for each feature (except for student_id)!

1.3

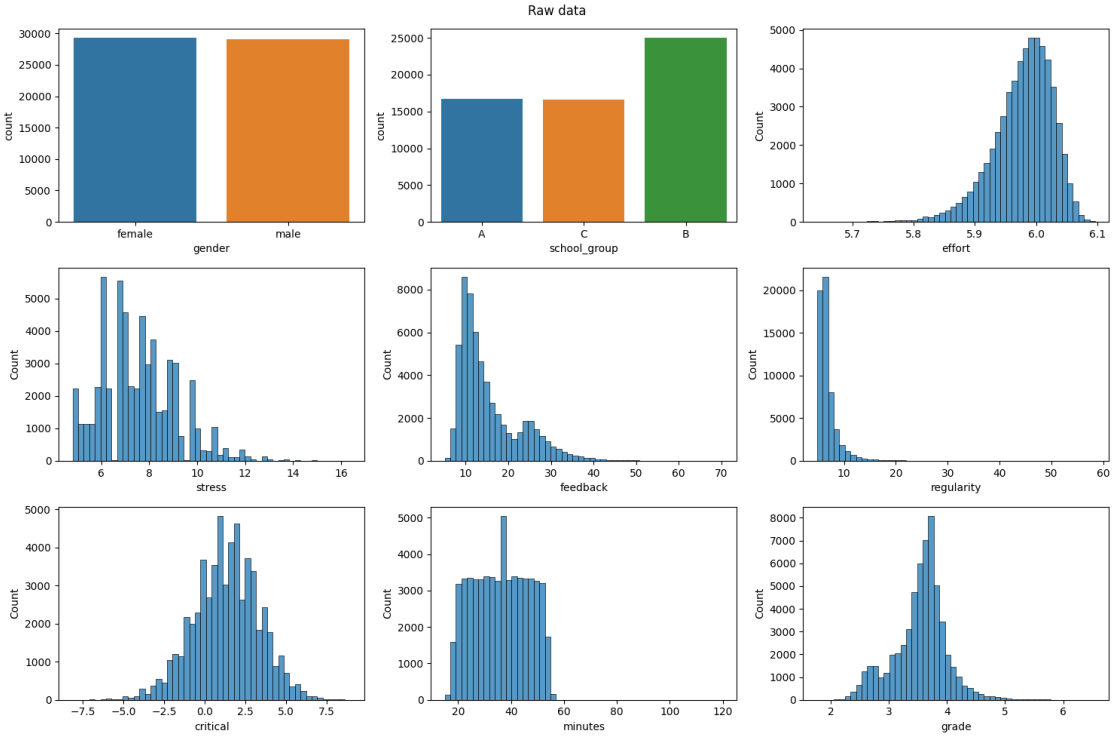
In a single figure, choose an appropriate type of graph for each feature and plot each feature individually.

For this task, it is important to choose the appropriate type of graphs for each of the 9 main features (we excluded the student_id here, because plotting them would not provide us with any interesting information).

For the categorical data an appropriate type of graph would be barplots, in order to visualize the distribution of the different values for the features. For the numerical data on the other hand, you could use histograms to visualize the distribution of the different values for the features.

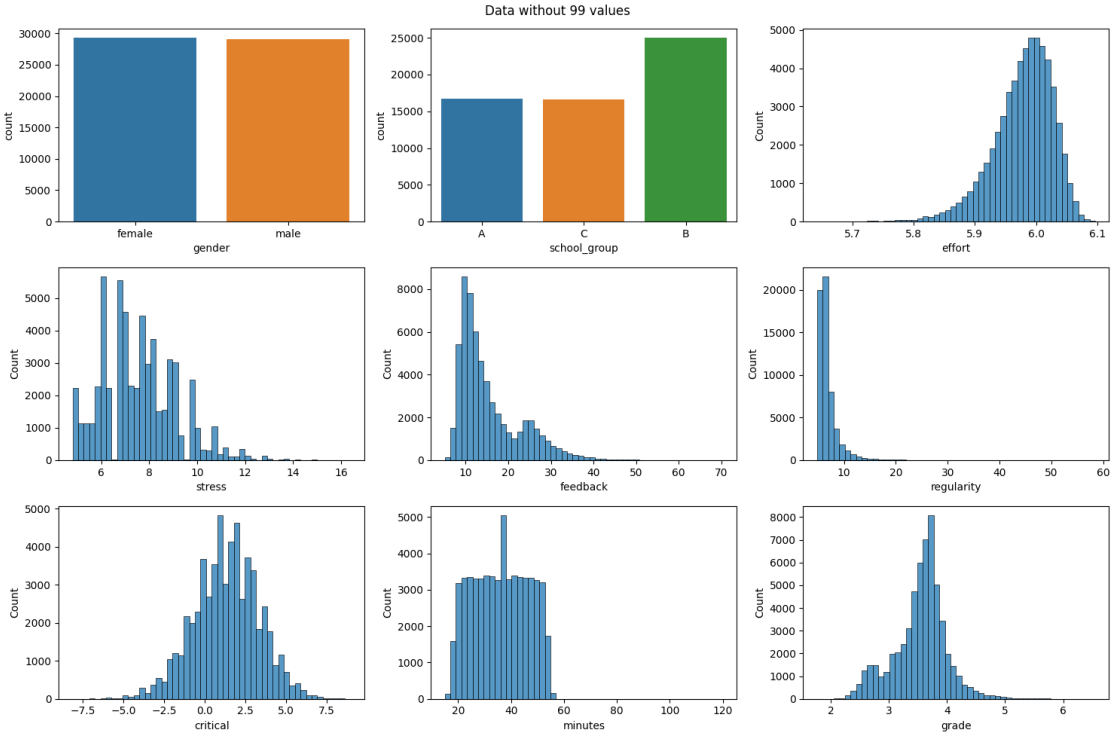
Especially for the histograms **it is important that you adjust the plots (bin size) appropriately**, in order to be able to gain some insights. For instance, you could consider adjusting the sizes of the bins or the scales for the x- and y-axes. For this data set in particular, you might have noticed that the appearance of the ‘99’s may distort the plots, making it difficult to see the distribution of the valid data. One way to deal with this is to go back to the plots, once the ‘99s’ have been handled. Finally, you should use subplots to display all the nine plots in the same figure.

```
### 1.3

def plot_features(df):
    """
    Plots all features individually in the same figure

    Parameters
    ----------
    df : DataFrame
         Containing all data

    Hint
    ------
    To have multiple plots in a single figure see pyplot.figure

    """
    df = df.copy()

    def plot_features(df, title):
        continuous_cols = list(df._get_numeric_data().columns)
        categorical_cols = list(set(df.columns) -
set(continuous_cols))
        fig, axes = plt.subplots(3, 3, figsize=(15,10))
        for i, col in enumerate(df.columns):
            ax = axes[i // 3, i % 3]
            data = df[~df[col].isna()]
            if col in continuous_cols:
                sns.histplot(data=data[col], bins=50, ax=ax) #Filter
out nan values in the features
            elif col in categorical_cols:
                sns.countplot(data=data, x=col, ax=ax)
            else:
                print(col)
        fig.suptitle(title)
        fig.tight_layout()


    plot_features(df,"Raw data")
    plt.show()
    # For plotting purposes, we removed the 99 values from the
numerical features
    # to see the distributions more clearly
    df[(df == 99) | (df == '99')] = np.nan

    plot_features(df, "Data without 99 values")
    return plt


send(plot_features(df),13)
```



<string>:57: MatplotlibDeprecationWarning: savefig() got unexpected keyword argument "quality" which is no longer supported as of 3.3 and will become an error in 3.6

<Response [200]>



1.4

Explain your observations. How are the features distributed (poisson, exponential, gaussian, etc)? Can you visually identify outliers?

From the plots it can be clearly seen that the '99's seem to be outliers. Once you have appropriately adjusted your plots, you will be able to identify the following distributions for the features:

- Gender: The data set is balanced between males and females (uniform). School groups: Also the school groups appear to be rather balanced (roughly uniform)
- Effort: The distribution is slightly left skewed, could be weibull.
- Stress: The distribution is slightly right skewed, could be roughly gaussian or gamma distributed
- Feedback: The distribution is right skewed, could be gamma distributed
- Regularity: The distribution is right skewed, could be log normal.
- Critical: The distribution looks gaussian
- Minutes: The distribution looks roughly uniform
- Grades: The distribution looks roughly gaussian

2 Data Cleaning

Using your findings from the previous section, carefully continue to explore the data set and do the following:

1. Create a function to handle the missing values
2. Justify your decisions to treat the missing values
3. Create a function to handle the inconsistent data
4. Justify your decisions to treat the inconsistent data

2.1

Create a function to handle the missing values

```
### 2.1
def handle_missing_values(df):
    """
    Identifies and removes all missing values

    Parameters
    ----------
    df : DataFrame
      Containing missing values

    Returns
    -------
```

```
    df : DataFrame
      Without missing values

    Hint:
    -----
    Try to understand the pattern in the missing values
    """
    ### BEGIN SOLUTION
    df = df.replace([99,'99'], np.nan)
    df = df.groupby(['student_id']).first()
    ### END SOLUTION
    return df

df = handle_missing_values(df)
send(len(df.columns), 21.1)
print("number of columns: ", len(df.columns))
```

```
number of columns:  9
```

```
# take a look at the new dataframe stats and compare it with the
original
get_feature_stats(df)
```

```
                  effort    stress   feedback  regularity  critical
grade  \
mean            5.978440  7.572990  15.494115    6.859709  1.330283
3.537279
std             0.048722  1.552971   7.330119    2.169821  2.005023
0.456478
50%             5.985351  7.401787  12.843072    6.227957  1.372255
3.600000
unique               NaN       NaN        NaN         NaN       NaN
NaN
top                  NaN       NaN        NaN         NaN       NaN
NaN
freq                 NaN       NaN        NaN         NaN       NaN
NaN
missing_values  0.000000  0.000000   0.000000    0.000000  0.000000
0.000000

                gender school_group minutes
mean               NaN          NaN     NaN
std                NaN          NaN     NaN
50%                NaN          NaN     NaN
unique               2            7      49
top             female            b    30.0
freq             29295         8414    1740
missing_values     0.0          0.0     0.0
```

2.2

Justify your decisions to treat the missing values. Are there missing values? If so, how are the missing values encoded? Why are there missing values? Is there a pattern in the values missing?

As we have seen from the previous analyses, the '99's seem to encode missing values in the data set. Furthermore, we have seen that there are double as many entries as unique values for student_id in the data set. This can be further investigated by inspecting a few examples of student_id and exploring their entries. By doing this, we can see that for each student_id, there are 2 entries in the data set containing complementary data (i.e., if one entry contains '99's, the real values can be found in the other entry of the same student_id). Hence by combining the data from both entries of a student_id, we can replace the '99's with meaningful values and reduce the size of the dataframe by half.

2.3

Create a function to handle the inconsistent data

```
### 2.3
def handle_inconsistent_data(df):
    """
    Identifies features with inconsistent data types and transforms
features
    to the correct data type (numerical, object).

    Parameters
    ----------
    df : DataFrame
      Containing inconsistent data

    Returns
    -------
    df : DataFrame
       With consistent data. All columns must be either numerical or
categorical

    Hint:
    -----
    Don't forget to convert the features into the correct data type
    """
    ### BEGIN SOLUTION
    mapping_time = {'1 hr': 60, '2hrs': 120, '2 hours': 120, '30 min':
30,
                    '45 min': 45, '60 minutes': 60, '1.5 hours':90 }
    mapping_group = {'a':'A', 'b':'B', 'c':'C', 'aa': 'A', 'Bb':'B',
'cc':'C'}

    df = df.replace({'minutes': mapping_time, 'school_group':
mapping_group})
```

```
    df['minutes'] = pd.to_numeric(df['minutes'])
    ### END SOLUTION
    return df

df = handle_inconsistent_data(df)
print(len(df))
print(df.head())
print(get_feature_stats(df))
```

```
58329
            gender school_group    effort    stress   feedback
regularity  \
student_id

1.0         female            A  5.974496  9.688888  24.563935
6.639488
2.0           male            A  5.982265  9.788799  18.722110
5.705770
3.0           male            C  6.011487  7.847762  15.577682
5.821650
4.0         female            B  5.838975  6.155117  18.597183
5.137559
5.0         female            C  6.013486  6.848094  12.498195
6.447001

            critical  minutes  grade
student_id
1.0        -1.795853     60.0   3.41
2.0         0.952679    120.0   2.66
3.0         2.913822    120.0   3.80
4.0         2.481461     30.0   3.53
5.0         2.015520     45.0   3.88
                effort    stress   feedback  regularity  critical  \
mean          5.978440  7.572990  15.494115    6.859709  1.330283
std           0.048722  1.552971   7.330119    2.169821  2.005023
50%           5.985351  7.401787  12.843072    6.227957  1.372255
unique             NaN       NaN        NaN         NaN       NaN
top                NaN       NaN        NaN         NaN       NaN
freq               NaN       NaN        NaN         NaN       NaN
missing_values  0.000000  0.000000   0.000000    0.000000  0.000000

                minutes     grade  gender school_group
mean          36.041300  3.537279     NaN          NaN
std           10.183264  0.456478     NaN          NaN
50%           36.000000  3.600000     NaN          NaN
unique              NaN       NaN       2            3
top                 NaN       NaN  female            B
freq                NaN       NaN   29295        25007
missing_values   0.000000  0.000000     0.0          0.0
```

2.4

Justify your decisions to treat the inconsistent data. Were there columns with inconsistent data types? How did you identify them?

As we have seen from the previous analyses, in addition to the missing values encoded as '99's, there are two additional features with inconsistent data.

First, there is the school group feature, for which we found 8 types of unique values, instead of the expected three. One of the expected types, the '99's, has already been treated in the previous task. To find the remaining ones, you can for instance apply the 'unique' function to the school_group column. Doing so, you will see that instead of the original school groups (A,B and C) there are also rows with misspelled entries (a, b, c, aa, Bb, cc). To treat these inconsistencies you can replace the misspelled entries with their correct values.

Second, there is the minutes feature, which was expected to be numerical, but seems to be categorical. By inspecting the values for this feature, we find that the entries are very inconsistent, with values including strings that describe the units in different forms ('min', 'minutes', 'hrs' or 'hours') as well as values provided in hours. To treat this inconsistent data, we decide to transform all the values into minutes and remove the strings, before transforming this feature to a numeric column.

3 Visualization

After cleaning the data, we can try to understand or extract insights from it. To do so, in this last section, you will do the following:

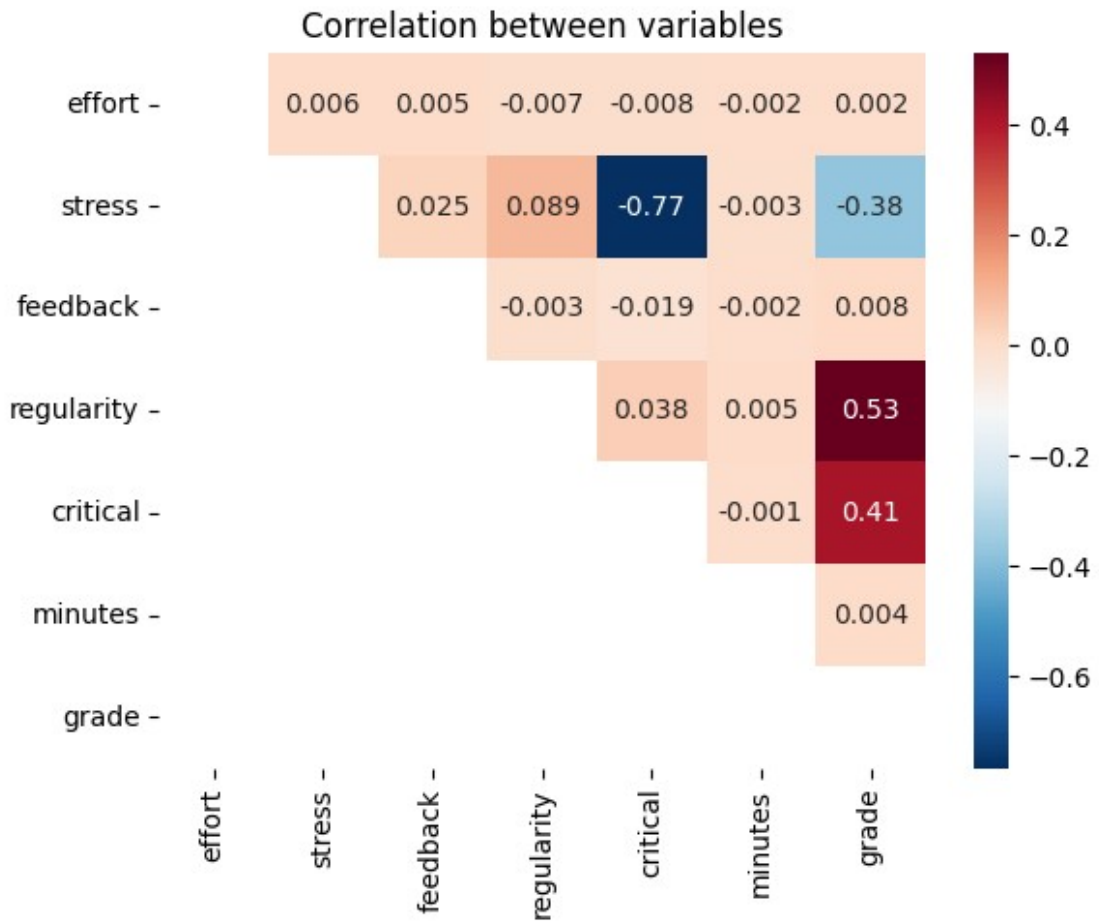
1. Create a function to show the relationship between numerical features.
2. Interpret your findings. What is correlation useful for? What insights can you get from it?
3. Select an appropriate type of graph to explore the relationship between grade, school group, and any other meaningful feature
4. Interpret your findings. What are some factors that seem to influence the grade of the students? Which features do not seem to affect the outcome?

3.1

Create a function to show the linear correlation between features.

```
### 3.1
import seaborn as sns
def plot_correlation(df):
    """
    Builds upper triangular heatmap with pearson correlation between
numerical variables

    Instructions
```

```
    ------------
    The plot must have:
    - An appropiate title
    - Only upper triangular elements
    - Annotated values of correlation coefficients rounded to three
significant
    figures
    - Negative correlation must be blue and possitive correlation red.


    Parameters
    ----------
    df : DataFrame with data


    """
    ### BEGIN SOLUTION
    corr = np.round(df.corr(method='pearson'), 3)
    mask = np.tril(corr)
    ax = plt.axes()
    heatmap = sns.heatmap(corr, annot=True, mask=mask, cmap='RdBu_r')
    ax.set_title('Correlation between variables')
    plt.show()
    ### END SOLUTION

send(plot_correlation(df),31)
```




```
Datatype not supported
```

3.2

Interpret your findings. What is correlation useful for? What insights can you get from it?

Pearson correlation can be useful to analyze LINEAR relationships between two variables. In case there is a linear dependency between variables, correlation allows us to quantify how strong this dependency is. However, it should also be noticed that strong correlations do not necessarily imply a causality of the observed effects. Moreover, note that Pearson correlation is only applicable to the numerical and not the categorical features. For the categorical data, other methods such as Chi square can be used to find associations between variables.

If generated correctly, the heatmap with the Pearson correlations between the numerical features will provide the following insights:

- Stress is negatively correlated with critical thinking
- Grade is positively correlated with regularity
- Grade is positively correlated with critical thinking

- Grade is negatively correlated with stress

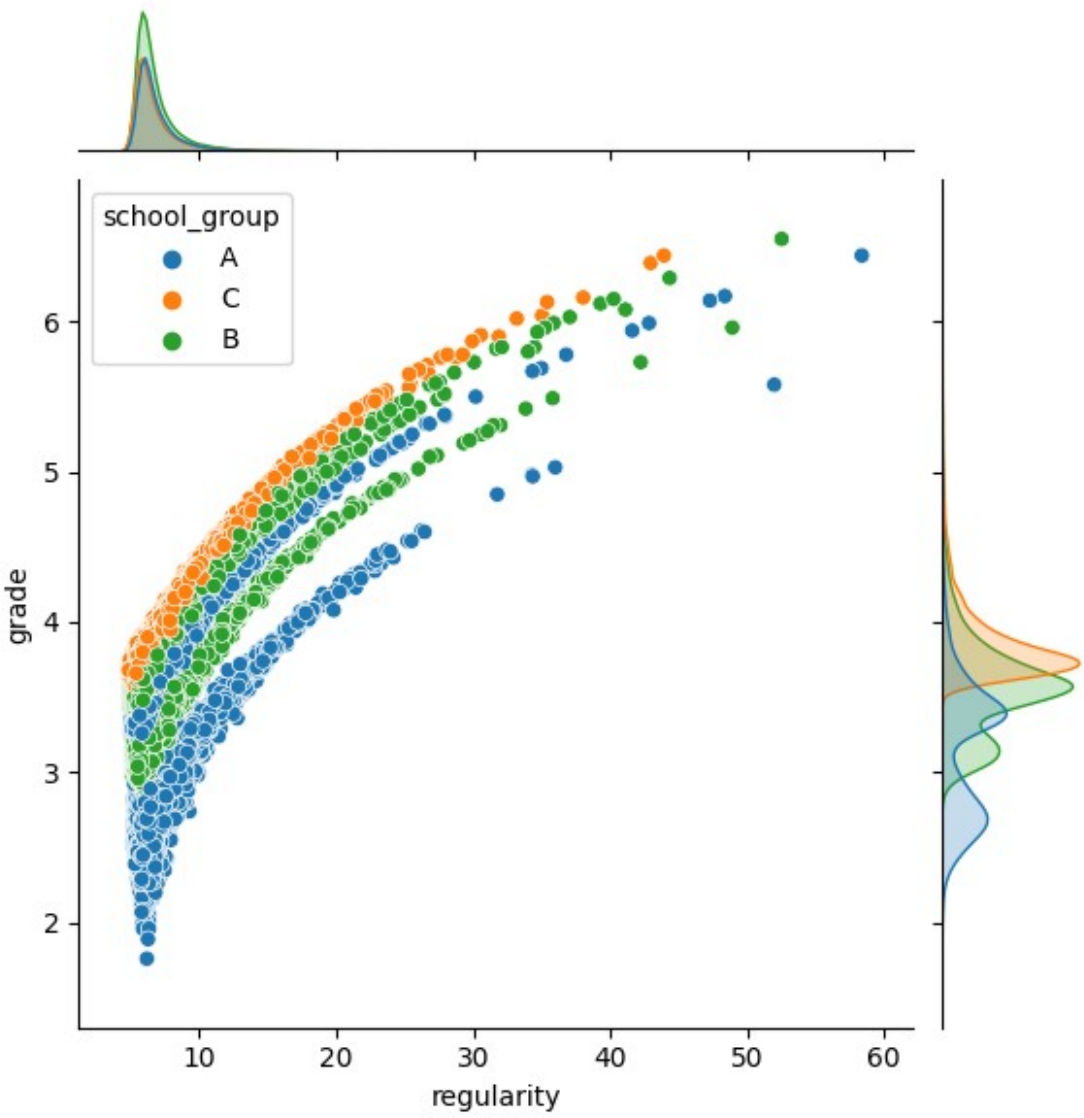
3.3

Select an appropriate type of graph to explore the relationship between grade, school group, and any other meaningful feature.

The plot should show the three features together. One way of doing this is using different colors to show the school grade (categorial feature)

```
### 3.3
def plot_grades(df):
    """
    Visualizes the relationship between grade,  school group and other
meaningful
    feature

    Parameters
    ----------
    df : DataFrame with data

    """
    ### BEGIN SOLUTION
    sns.jointplot(data = df, y = 'grade', x = 'regularity', hue =
'school_group')
    ### END SOLUTION

send(plot_grades(df),33)
```

```
Datatype not supported
```



3.4

Interpret your findings. What are some factors that seem to influence the grade of the students? Which features do not seem to affect the outcome?

The idea of this task was to use a plot that allows you to study the relationships of three variables at the same time. For this task, different solutions are possible, here we present an example using Seaborn's jointplot and choosing regularity as the third feature. We plot regularity on the x-axis and grade on the y-axis, and choose the school group as the hue. This will generate a scatter plot with different colors for the school groups and additionally adds the density distributions for each group at the borders of the plot.

For this particular choice of plot and variables, we can observe that there is a general trend of better grades with higher regularity independent of the school group. This is in line with one of our observations from the Pearson correlations. However, here we also see that for

the same regularity, students in school group C generally have higher grades than students from the other two groups. This is an interesting observation which could be further investigated in more analyses.

Lab 03 - Extended Exercises

Explaining and predicting student performance

We recommend using Noto for this lecture tutorial, where we've already installed the dependencies of the pymer4 package and statsmodels.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Import the linear regression model class
from pymer4.models import Lm

# Import the lmm model class
from pymer4.models import Lmer

# Import Gaussian modeling
import statsmodels.formula.api as smf

import scipy as sp
from scipy import stats

# Data directory
DATA_DIR = "./../../data/"
```

```
import requests

exec(requests.get("https://courdier.pythonanywhere.com/get-send-
code").content)

npt_config = {
    'session_name': 'lab-03',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}
```

```
Your name:  Paola
```

Introduction

The data has already been cleaned and it comes from 29 students in 3 different groups in a course of 26 weeks.

In this lab you will explore different models to explain the quiz grade.

```
# Load data
df= pd.read_csv(f'{DATA_DIR}grades_in_time.csv.gz')
send(len(df),0)
df.head()
```

	student	week	studying_hours	group	quiz_grade
0	0	0	39.9	3	6.1
1	0	1	32.4	3	7.0
2	0	2	17.5	3	6.9
3	0	3	16.0	3	7.0
4	0	4	15.9	3	7.2

```
df.describe(include='all')
```

	student	week	studying_hours	group	quiz_grade
count	810.000000	810.000000	810.000000	810.000000	810.000000
mean	14.500000	13.000000	10.050617	1.933333	6.931975
std	8.660789	7.793693	8.270041	0.772199	1.336888
min	0.000000	0.000000	1.000000	1.000000	1.200000
25%	7.000000	6.000000	5.700000	1.000000	6.400000
50%	14.500000	13.000000	7.800000	2.000000	7.200000
75%	22.000000	20.000000	11.100000	3.000000	7.800000
max	29.000000	26.000000	64.000000	3.000000	10.100000

Task 1: Linear Model

1.1 Preprocess the data to run a regression model to explain the effect of studying hours on quiz grade.

```
from sklearn.preprocessing import StandardScaler
columns_to_scale = ['studying_hours']
X = df[columns_to_scale]
scaler = StandardScaler()
X = scaler.fit_transform(X)

simple_X = pd.DataFrame(X, columns = columns_to_scale)
df_scaled = pd.concat([simple_X, df[['quiz_grade', 'week', 'student',
'group']]], axis=1)

df_scaled.head(2)
```

	studying_hours	quiz_grade	week	student	group
0	3.611569	6.1	0	0	3
1	2.704121	7.0	1	0	3

1.2 Explain your preprocessing steps

```
answer = """
Standardization helps correctly compare multiple variables (in
different units) and
reduce multicollinearity.
In this case, we only have one variable feature. Thus, it is not
necessary but
```

```
it is a good practice.
"""

send(answer, 12)
```

```
<Response [200]>
```

1.3 Run a regression model to explain the effect of studying hours on quiz grade.

```
model1_str = """quiz_grade ~ 0 + studying_hours """ ## Write your
model here
send(model1_str,13)
model = Lm(model1_str, data=df_scaled, family='gaussian')

# Fit the models
print(model.fit())
```

```
Formula: quiz_grade~0+studying_hours

Family: gaussian  Estimator: OLS

Std-errors: non-robust     CIs: standard 95%     Inference: parametric


Number of observations: 810 R^2: 0.007      R^2_adj: 0.006

Log-likelihood: -2729.422   AIC: 5460.844   BIC: 5465.541

Fixed effects:

                Estimate  2.5_ci  97.5_ci     SE   DF  T-stat  P-val
Sig
studying_hours     0.603   0.118    1.089  0.247  809   2.439  0.015
*
```

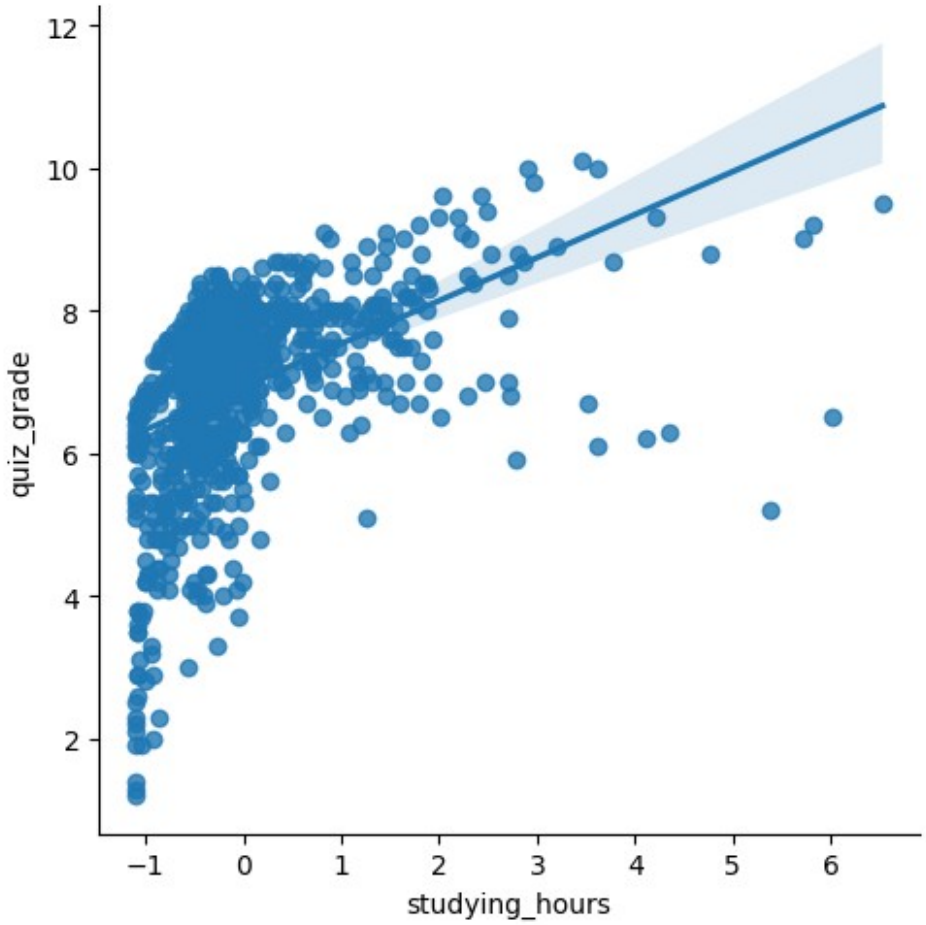
```
model1_str = """quiz_grade ~ 1 + studying_hours """ ## Write your
model here
send(model1_str,13)
model = Lm(model1_str, data=df_scaled, family='gaussian')

# Fit the models
print(model.fit())
```

```
Formula: quiz_grade~1+studying_hours

Family: gaussian  Estimator: OLS

Std-errors: non-robust     CIs: standard 95%     Inference: parametric


Number of observations: 810 R^2: 0.204      R^2_adj: 0.203
```

```
Log-likelihood: -1291.688   AIC: 2587.376   BIC: 2596.770

Fixed effects:

                Estimate  2.5_ci  97.5_ci     SE   DF   T-stat  P-val
Sig
Intercept          6.932   6.850    7.014  0.042  808  165.288    0.0
***
studying_hours     0.603   0.521    0.686  0.042  808   14.384    0.0
***
```

```
sns.lmplot(x="studying_hours", y="quiz_grade", data=df_scaled)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f22a9dd6820>
```



1.3 What model family (poisson, logistic, etc) did you use and why?

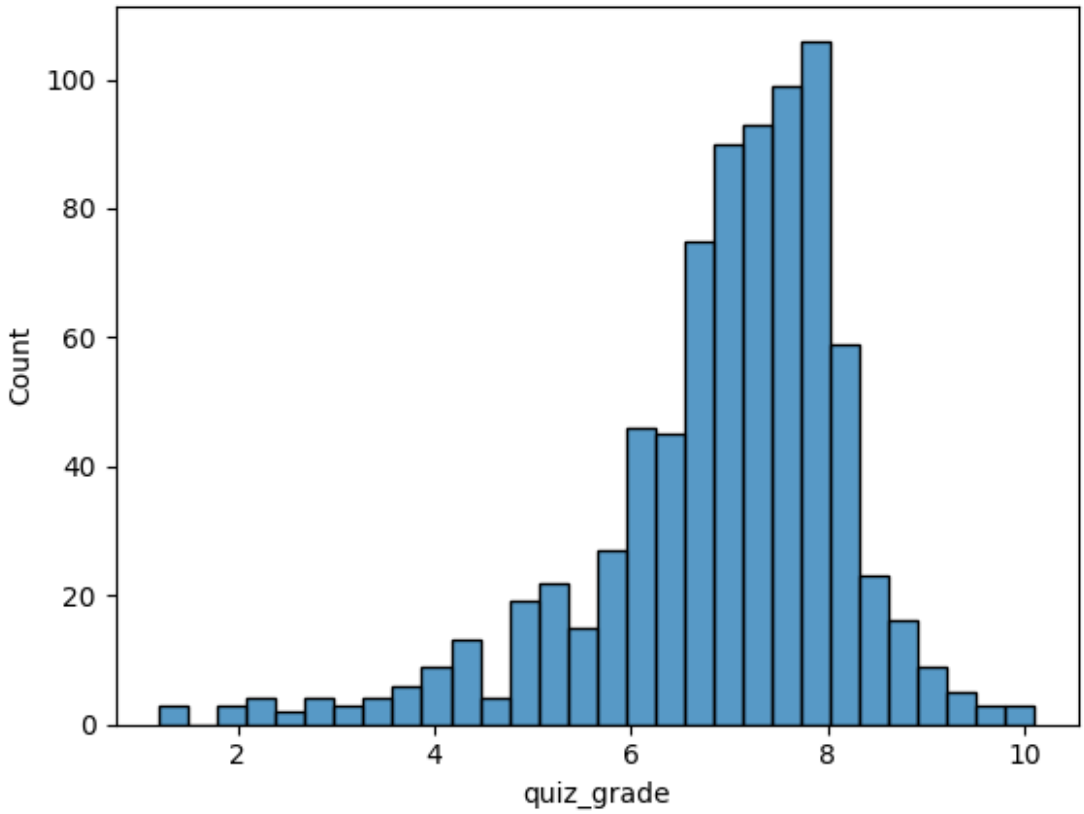
```
answer = """
Gaussian is a good approximation becauset the dependent feature (y) is
continuous (not discrete or binary).
"""
```

```
send(answer, 13)
```

```
<Response [200]>
```

```
sns.histplot(df_scaled["quiz_grade"])
```

```
<AxesSubplot:xlabel='quiz_grade', ylabel='Count'>
```



1.4 Interpret the regression results.

Do the variables have a positive or negative effect? Is it significant?

```
answer = """
The number of studying hours has a positive and significant effect.
"""

send(answer, 14)
```

```
<Response [200]>
```

1.5 Is this an appropriate method? Explain why or why not.

```
answer = """
No, because group differences are not taken into account.
"""
```

```
send(answer, 15)
```

```
<Response [200]>
```

Task 2: Linear Model with Fixed Effects

2.1 Run a regression model to explain the effect of studying hours on quiz grade. Add fixed effects for group.

```
df_scaled['group'] = df_scaled['group'].astype(str)

model = Lm("""quiz_grade ~ 1 + studying_hours  + group""",
data=df_scaled, family='gaussian')

# Fit the models
print(model.fit())
```

```
Formula: quiz_grade~1+studying_hours+group

Family: gaussian  Estimator: OLS

Std-errors: non-robust     CIs: standard 95%     Inference: parametric


Number of observations: 810 R^2: 0.237      R^2_adj: 0.234

Log-likelihood: -1274.546   AIC: 2557.091   BIC: 2575.879

Fixed effects:

                Estimate  2.5_ci  97.5_ci     SE   DF  T-stat  P-val
Sig
Intercept          6.586   6.445    6.727  0.072  806  91.707    0.0
***
group[T.2]         0.551   0.357    0.745  0.099  806   5.580    0.0
***
group[T.3]         0.469   0.260    0.679  0.107  806   4.394    0.0
***
studying_hours     0.647   0.564    0.730  0.042  806  15.294    0.0
***
```

2.2 Interpret the regression results.

What changed? What does it mean to have group fixed effects?

```
answer = """
 Group fixed effects allow us to difference out any constant
differences between groups,
 and focus only on changes within each entity over time.
"""
```

```
send(answer, 22)
```

```
<Response [200]>
```

Task 3: Linear Model with Random Effects

3.1 Run a regression model to explain the effect of studying hours on quiz grade. Add random intercept for group.

```
model = Lmer("""quiz_grade ~ 1 + (1|group) + studying_hours """,
data=df_scaled, family='gaussian')

# Fit the models
print(model.fit())
```

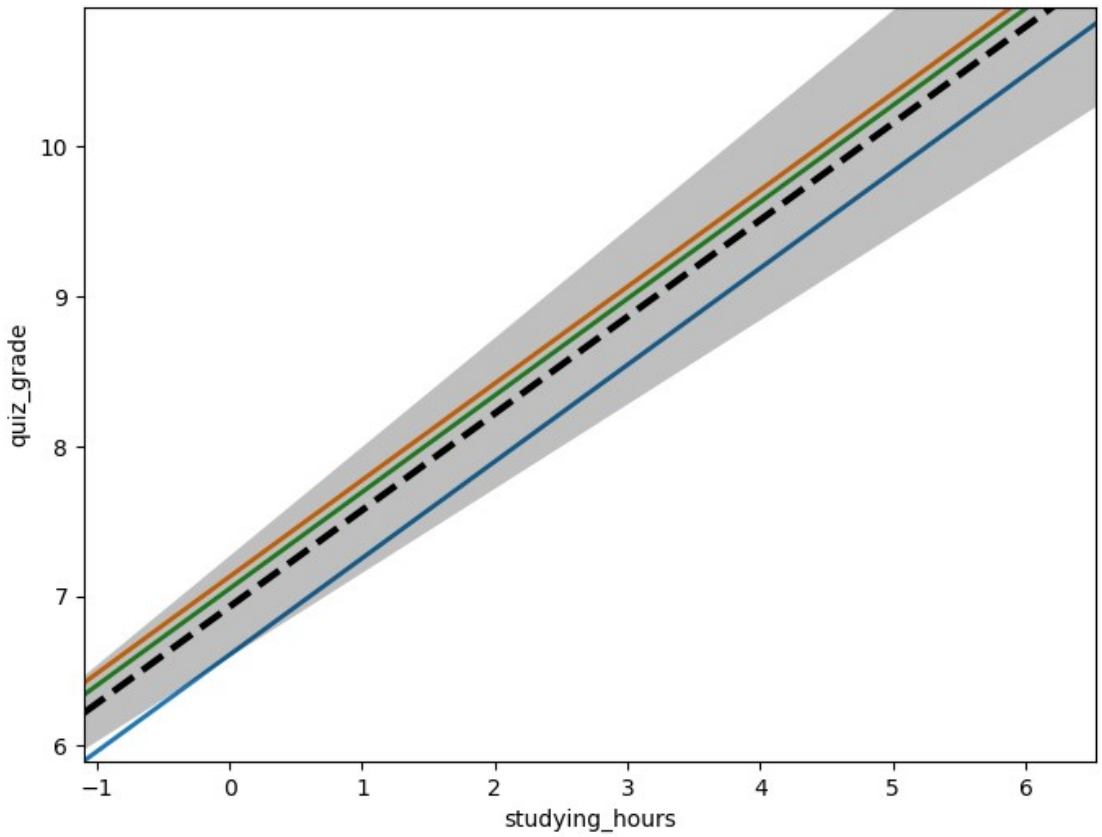
```
Formula: quiz_grade~1+(1|group)+studying_hours

Family: gaussian  Inference: parametric

Number of observations: 810  Groups: {'group': 3.0}

Log-likelihood: -1282.909    AIC: 2565.817

Random effects:

                 Name    Var    Std
group     (Intercept)  0.084  0.289
Residual               1.369  1.170

No random effect correlations specified

Fixed effects:

                Estimate  2.5_ci  97.5_ci     SE       DF  T-stat  P-
val  Sig
(Intercept)        6.927   6.589    7.264  0.172    2.004  40.249
0.001  ***
studying_hours     0.645   0.562    0.728  0.042  807.849  15.263
0.000  ***
```

3.2 Plot the regression lines

Hint: You may use model.plot

```
model.plot("studying_hours", plot_ci=True)
```

```
<AxesSubplot:xlabel='studying_hours', ylabel='quiz_grade'>
```



3.3 Run a regression model to explain the effect of studying hours on quiz grade. Add slope for group.

```
model = Lmer("""quiz_grade ~ 1  + (0 + studying_hours|group) """,
data=df_scaled, family='gaussian')

# Fit the models
print(model.fit())
```

```
Formula: quiz_grade~1+(0+studying_hours|group)

Family: gaussian  Inference: parametric

Number of observations: 810  Groups: {'group': 3.0}

Log-likelihood: -1230.242    AIC: 2460.484

Random effects:

                     Name    Var    Std
group     studying_hours  0.362  0.602
Residual                  1.198  1.094

No random effect correlations specified
```

```
Fixed effects:

             Estimate  2.5_ci  97.5_ci     SE       DF   T-stat  P-val
Sig
(Intercept)     6.853   6.766     6.94  0.044  739.487  154.658    0.0
***
```

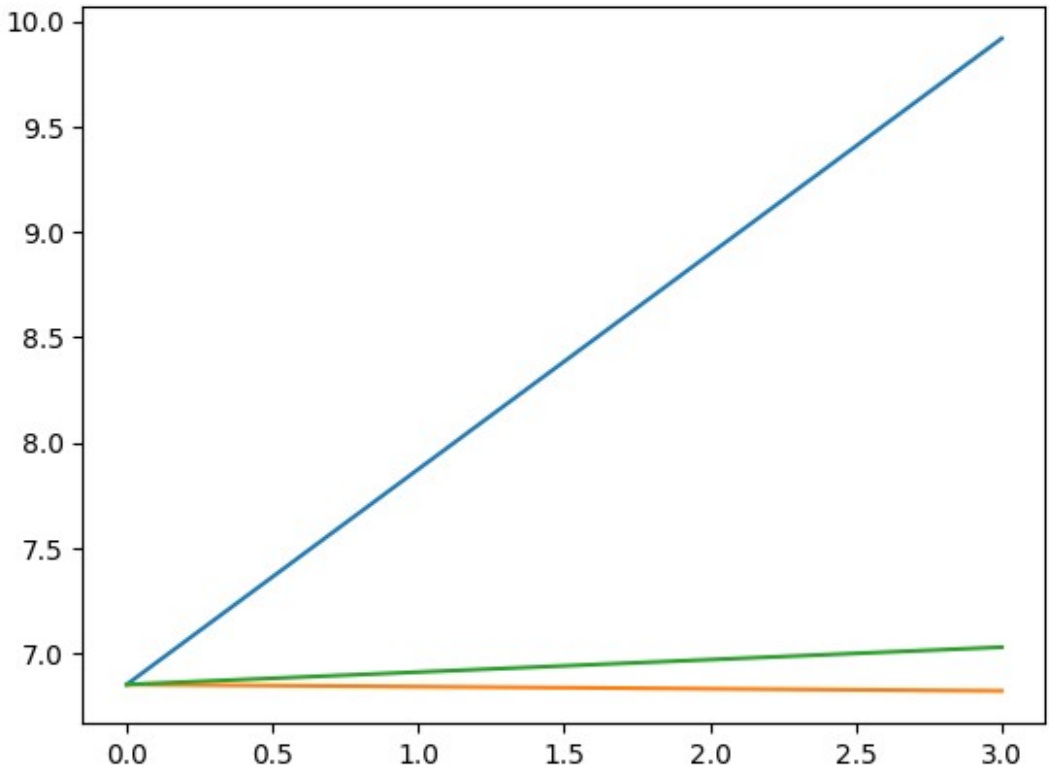
3.4 Plot the regression lines

Hint: You may use model.ranef

```
intercept = model.coefs.Estimate[0]

model.ranef.head()
```

```
   studying_hours
1        1.021431
2       -0.010058
3        0.059044
```

```
x = np.linspace(0, 3,4)
for i, row in model.ranef.iterrows():
    sns.lineplot(x=x, y=intercept + row['studying_hours']*x)
```



3.5 Run a regression model to explain the effect of studying hours on quiz grade. Add random intercept and slope for group.

```
model = Lmer("""quiz_grade ~ (1 + studying_hours|group) """,
data=df_scaled, family='gaussian')

# Fit the models
print(model.fit())
```

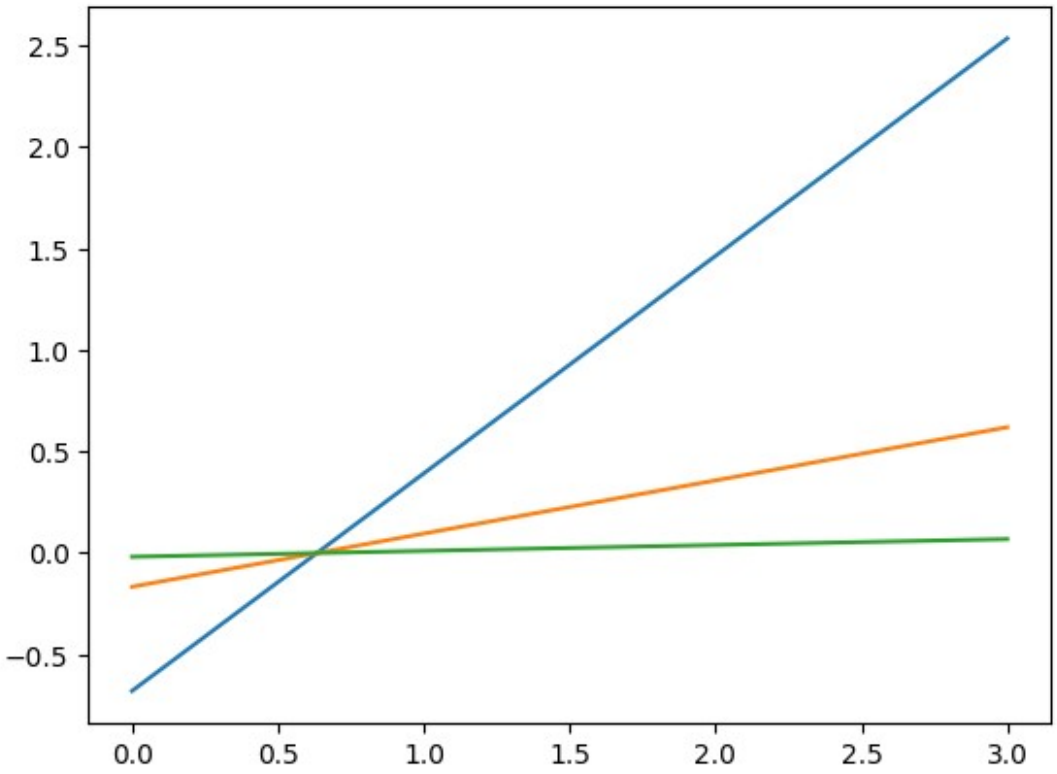
```
boundary (singular) fit: see help('isSingular')

Formula: quiz_grade~(1+studying_hours|group)

Family: gaussian  Inference: parametric

Number of observations: 810 Groups: {'group': 3.0}

Log-likelihood: -1205.548   AIC: 2411.096

Random effects:

                     Name    Var    Std
group         (Intercept)  0.165  0.406
group      studying_hours  0.410  0.640
Residual                   1.124  1.060

               IV1              IV2  Corr
group  (Intercept)  studying_hours  -1.0

Fixed effects:

             Estimate  2.5_ci  97.5_ci    SE       DF   T-stat  P-val
Sig
(Intercept)      7.18   7.082    7.278  0.05  359.077  143.976    0.0
***
```

```
model.ranef
```

```
   X.Intercept.  studying_hours
1     -0.679018        1.071420
2     -0.166013        0.261951
3     -0.018387        0.029013
```

3.6 Plot the regression lines

```
x = np.linspace(0, 3,4)
for i, row in model.ranef.iterrows():
    sns.lineplot(x=x, y=row['X.Intercept.'] + row['studying_hours']*x)
```



3.7 Interpret the regression results.

What changed? What does it mean to have group random effects?

```
answer = """
Effects are fixed if they are interesting in themselves
or random if there is interest in the underlying population.
With intercept random effects, we assumed that every group has a
different starting
point (y-intercept) and with slope random effects we assume that every
group has a different rate.
"""

send(answer, 37)
```

```
<Response [200]>
```

Task 4: Mixed Model with Time Interaction

4.1 Again, run a regression model to explain the effect of studying hours on quiz grade. Add random intercept and slope for groups AND interaction between the number of stuyding hours and time (weeks).

```
model = Lmer("""quiz_grade ~  (1 + studying_hours*week|group) """,
data=df_scaled, family='gaussian')
```

```
# Fit the models
print(model.fit())
```

```
boundary (singular) fit: see help('isSingular')

Formula: quiz_grade~(1+studying_hours*week|group)

Family: gaussian  Inference: parametric

Number of observations: 810  Groups: {'group': 3.0}

Log-likelihood: -845.308    AIC: 1690.616

Random effects:

                         Name    Var    Std
group             (Intercept)  1.701  1.304
group           studying_hours  1.512  1.230
group                     week  0.012  0.109
group     studying_hours:week  0.005  0.071
Residual                       0.446  0.668

                  IV1                   IV2   Corr
group     (Intercept)       studying_hours -0.733
group     (Intercept)                 week -0.978
group     (Intercept)  studying_hours:week -0.062
group  studying_hours                 week  0.609
group  studying_hours  studying_hours:week -0.564
group            week  studying_hours:week  0.263

Fixed effects:

             Estimate  2.5_ci  97.5_ci     SE       DF   T-stat  P-val
Sig
(Intercept)     7.142   7.075     7.21  0.034  212.364  207.848    0.0
***
```

4.2 Interpret the regression results.

```
answer = """
The variance of quiz grade by groups is estimated as 1.600 + 1.515 +
0.12 + 0.005 + 0.445 (from the residual) = 3.685
The studying hours by groups explain a big part of the variance (41%)
but the interaction between studying hours and weeks
explains much less of the variance (0.1%).
We also observe a high correlation between random effects (intercept
and slope) within each group.
Studying hours is negatively correlated (-0.73) with the intercept as
well as weeks (-0.97).
The interaction term is weakly correlated with the intercept.
```

```
When analyzing the coefficients (with model.ranef)
we observe that studying hours has a greater coefficient for group 1
(in comparison to the other groups)
and the week has a greater coefficient for group 3 (in comparison to
the other groups).
"""

send(answer, 42)
```

```
<Response [200]>
```

```
model.ranef
```

```
   X.Intercept.  studying_hours      week  studying_hours.week
1     -1.352402        2.053030  0.091559            -0.072844
2     -1.427752        0.598642  0.108035            -0.000913
3     -1.175201        0.065531  0.129008             0.096540
```

Lab 04- Extended Exercises on Classification and Pipelines

```
import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import adjusted_mutual_info_score
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel,
mutual_info_classif
from sklearn import model_selection
from sklearn.metrics import roc_auc_score, balanced_accuracy_score,
confusion_matrix, ConfusionMatrixDisplay, silhouette_score
from sklearn.model_selection import train_test_split
```

You are the Senior Data Scientist in a learning platform called LernTime. You have realized that many users stop using the platform and want to increase user retention. For this purpose, you decide to build a model to predict whether a student will stop using the learning platform or not.

Your data science team built a data frame in which each row contains the aggregated features per student (calculated over the first 5 weeks of interactions) and the feature `dropout` indicates whether the student stopped using the platform (1) or not (0) before week 10.

The dataframe is in the file `lerntime.csv` and contains the following features:

- `video_time`: total video time (in minutes)
- `num_sessions` total number of sessions
- `num_quizzes`: total number of quizzes attempts
- `reading_time`: total theory reading time
- `previous_knowledge`: standardized previous knowledge
- `browser_speed`: standardized browser speed
- `device`: whether the student logged in using a smartphone (1) or a computer (-1)
- `topics`: the topics covered by the user
- `education`: current level of education (0: middle school, 1: high school, 2: bachelor, 3: master, 4: Ph.D.).
- `dropout`: whether the student stopped using the platform (1) or not (0) before week 5.

The newest data scientist created two models with an excellent performance. As a Senior Data Scientist, you are suspicious of the results and decide to revise the code.

Your task is to:

a) Identify the mistakes. In the first cell, add a comment above each line in which you identify an error and explain the error.

b) In the second cell, you must correct the code.

```
df = pd.read_csv('data/lerntime_dropout.csv')

y = df['dropout']
X = df[['video_time', 'num_sessions', 'num_quizzes', 'reading_time',
       'previous_knowledge', 'browser_speed']]

len(df)
```

```
300
```

Task A) Identify the mistakes in the code

In the following cell, add a comment above each line in which you identify an error and explain the why it is erroneous. Please start each of your comments with `#ERROR:`. For example:

```
#ERROR: the RMSE of the model is printed instead of the AUC

print("The AUC of the model is: {}".format(rmse))
```

You may assume that:

- all the features are continous and numerical.
- the features have already been cleaned and processed.

```
# ERROR: Train-test split should be done before preprocessing steps 1.
and 2. to avoid data leakage,
# fitting both scaler and selector only on X_train
## 1. Scale the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

## 2. Feature selection (Lasso)
print(X.shape)
lasso = Lasso(alpha=0.1, random_state=0).fit(X, y)
selector = SelectFromModel(lasso, prefit = True)
X = selector.transform(X)
print(X.shape)

## 3. Split the data
# ERROR: The split should be done before the feature selection or
transformation
# to avoid data leaking
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=0)

## Model 1
```

```
clf = RandomForestClassifier(n_estimators=10, max_depth=1,
random_state=0)
# ERROR: Fit should only be called on the train set
clf.fit(X,y)
preds = clf.predict(X_test)
# ERROR: The adjusted mutual information is not an appropriate score
for classification, since it would give
# a perfect score even if the predictions are the complete opposite of
y_test
print("Score model 1:
{}".format(np.round(adjusted_mutual_info_score(preds, y_test), 2)))

## Model 2
clf = RandomForestClassifier(n_estimators=1000, max_depth=None,
random_state=0)
# ERROR: Fit should only be called on the train set
clf.fit(X,y)
preds = clf.predict(X_test)
# ERROR: The adjusted mutual information is not an appropriate score
for classification, since it would give
# a perfect score even if the predictions are the complete opposite of
y_test
print("Score model 2:
{}".format(np.round(adjusted_mutual_info_score(preds, y_test), 2)))

# ERROR: The second model has just more complexity and can hence
better overfit to the test set, which leaked during training
## Discussion
# Our second model achieved perfect results with unseen data and
outperforms the first model.
## This is because we increased the number of estimators.
```

```
(300, 3)
(300, 3)
Score model 1: 0.05
Score model 2: 1.0
```

Task B) Correct the code

Correct all the erroneous code in the following cell.

```
## 1. Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=0)

## 2. Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

## 3. Feature selection (Lasso)
```

```
print(X_train.shape)
lasso = Lasso(alpha=0.1, random_state=0).fit(X_train, y_train)
selector = SelectFromModel(lasso, prefit = True)
X_train = selector.transform(X_train)
X_test = selector.transform(X_test)
print(X_train.shape)

## Model 1
clf = RandomForestClassifier(n_estimators=10, max_depth=1,
random_state=0)
clf.fit(X_train, y_train)
preds = clf.predict(X_test)
print("Score model 1:
{}".format(np.round(balanced_accuracy_score(preds, y_test), 2)))

## Model 2
clf = RandomForestClassifier(n_estimators=1000, max_depth=None,
random_state=0)
clf.fit(X_train, y_train)
preds = clf.predict(X_test)
print("Score model 2:
{}".format(np.round(balanced_accuracy_score(preds, y_test), 2)))

## Discussion
# Our first model outperformed the second model.
# However, it is not clear why because we change the number of
estimators and the maximum depth at the same time
```

```
(240, 3)
(240, 3)
Score model 1: 0.9
Score model 2: 0.81
```

Task C) Re-write your code using pipelines.

Hint: Go over sklearn-pipeline-introduction.

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedGroupKFold

scalers =  [
    StandardScaler(),
    'passthrough'] # none

feature_selectors = [
    SelectFromModel(Lasso(alpha=0.1, random_state=0)),
    'passthrough'
]

steps = [('scaler', StandardScaler()), # preprocessing steps
```

```
        ('feature_selector', SelectFromModel(Lasso(alpha=0.1,
random_state=0))), # Feature selection
        ('clf', RandomForestClassifier(random_state=0))]
# Model

param_grid = {
    'scaler': scalers,
    'feature_selector':feature_selectors,
    'clf__n_estimators': [10,1000],
    'clf__max_depth':[1, None]
}

pipeline = Pipeline(steps)

search = GridSearchCV(pipeline, param_grid, n_jobs=-1, cv = 5, scoring
= "balanced_accuracy")
search.fit(X, y)
print("Best parameter (CV score=%0.2f):" % search.best_score_)
print(search.best_params_)
```

```
Best parameter (CV score=0.81):
{'clf__max_depth': None, 'clf__n_estimators': 1000,
'feature_selector': SelectFromModel(estimator=Lasso(alpha=0.1,
random_state=0)), 'scaler': StandardScaler()}
```

```
results = pd.DataFrame(search.cv_results_)
results.sort_values('rank_test_score')[[
       'param_clf__max_depth', 'param_clf__n_estimators',
       'param_feature_selector', 'param_scaler', 'params',
'mean_test_score', 'std_test_score',
       'rank_test_score']]
```

```
   param_clf__max_depth param_clf__n_estimators  \
12                 None                     1000
14                 None                     1000
13                 None                     1000
15                 None                     1000
8                  None                       10
9                  None                       10
10                 None                       10
11                 None                       10
0                     1                       10
1                     1                       10
2                     1                       10
3                     1                       10
4                     1                     1000
5                     1                     1000
6                     1                     1000
7                     1                     1000
```

```
                                param_feature_selector
param_scaler  \
12  SelectFromModel(estimator=Lasso(alpha=0.1, ran...
StandardScaler()
14                                        passthrough
StandardScaler()
13  SelectFromModel(estimator=Lasso(alpha=0.1, ran...
passthrough
15                                        passthrough
passthrough
8   SelectFromModel(estimator=Lasso(alpha=0.1, ran...
StandardScaler()
9   SelectFromModel(estimator=Lasso(alpha=0.1, ran...
passthrough
10                                        passthrough
StandardScaler()
11                                        passthrough
passthrough
0   SelectFromModel(estimator=Lasso(alpha=0.1, ran...
StandardScaler()
1   SelectFromModel(estimator=Lasso(alpha=0.1, ran...
passthrough
2                                         passthrough
StandardScaler()
3                                         passthrough
passthrough
4   SelectFromModel(estimator=Lasso(alpha=0.1, ran...
StandardScaler()
5   SelectFromModel(estimator=Lasso(alpha=0.1, ran...
passthrough
6                                         passthrough
StandardScaler()
7                                         passthrough
passthrough

                                                 params  mean_test_score
\
12  {'clf__max_depth': None, 'clf__n_estimators': ...         0.806935

14  {'clf__max_depth': None, 'clf__n_estimators': ...         0.806935

13  {'clf__max_depth': None, 'clf__n_estimators': ...         0.801380

15  {'clf__max_depth': None, 'clf__n_estimators': ...         0.801380

8   {'clf__max_depth': None, 'clf__n_estimators': ...         0.770950

9   {'clf__max_depth': None, 'clf__n_estimators': ...         0.770950
```

```
10  {'clf__max_depth': None, 'clf__n_estimators': ...         0.770950

11  {'clf__max_depth': None, 'clf__n_estimators': ...         0.770950

0   {'clf__max_depth': 1, 'clf__n_estimators': 10,...         0.624183

1   {'clf__max_depth': 1, 'clf__n_estimators': 10,...         0.624183

2   {'clf__max_depth': 1, 'clf__n_estimators': 10,...         0.624183

3   {'clf__max_depth': 1, 'clf__n_estimators': 10,...         0.624183

4   {'clf__max_depth': 1, 'clf__n_estimators': 100...         0.594164

5   {'clf__max_depth': 1, 'clf__n_estimators': 100...         0.594164

6   {'clf__max_depth': 1, 'clf__n_estimators': 100...         0.594164

7   {'clf__max_depth': 1, 'clf__n_estimators': 100...         0.594164
```

	std_test_score	rank_test_score
12	0.045989	1
14	0.045989	1
13	0.043601	3
15	0.043601	3
8	0.049943	5
9	0.049943	5
10	0.049943	5
11	0.049943	5
0	0.040118	9
1	0.040118	9
2	0.040118	9
3	0.040118	9
4	0.055740	13
5	0.055740	13
6	0.055740	13
7	0.055740	13

```
results[['split0_test_score',
       'split1_test_score', 'split2_test_score', 'split3_test_score',
       'split4_test_score', 'mean_test_score',
'std_test_score',]].sort_values('std_test_score')
```

	split0_test_score	split1_test_score	split2_test_score	\
0	0.676471	0.583333	0.666667	
1	0.676471	0.583333	0.666667	
2	0.676471	0.583333	0.666667	
3	0.676471	0.583333	0.666667	

```
13           0.788646           0.809524           0.730159
15           0.788646           0.809524           0.730159
12           0.788646           0.837302           0.730159
14           0.788646           0.837302           0.730159
8            0.747606           0.718254           0.753968
9            0.747606           0.718254           0.753968
10           0.747606           0.718254           0.753968
11           0.747606           0.718254           0.753968
4            0.617647           0.527778           0.658730
5            0.617647           0.527778           0.658730
6            0.617647           0.527778           0.658730
7            0.617647           0.527778           0.658730

    split3_test_score  split4_test_score  mean_test_score
std_test_score
0            0.583333           0.611111         0.624183
0.040118
1            0.583333           0.611111         0.624183
0.040118
2            0.583333           0.611111         0.624183
0.040118
3            0.583333           0.611111         0.624183
0.040118
13           0.813492           0.865079         0.801380
0.043601
15           0.813492           0.865079         0.801380
0.043601
12           0.813492           0.865079         0.806935
0.045989
14           0.813492           0.865079         0.806935
0.045989
8            0.769841           0.865079         0.770950
0.049943
9            0.769841           0.865079         0.770950
0.049943
10           0.769841           0.865079         0.770950
0.049943
11           0.769841           0.865079         0.770950
0.049943
4            0.527778           0.638889         0.594164
0.055740
5            0.527778           0.638889         0.594164
0.055740
6            0.527778           0.638889         0.594164
0.055740
7            0.527778           0.638889         0.594164
0.055740
```

Lab 05 - Extended Exercises on Model Evaluation

Predicting student performance

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from sklearn.pipeline import Pipeline
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score, train_test_split

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler,  OneHotEncoder

# Data directory
DATA_DIR = "./../../data/"

import requests

exec(requests.get("https://courdier.pythonanywhere.com/get-send-
code").content)

npt_config = {
    'session_name': 'lab-05',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}
```

```
Your name:  Paola
```

Introduction

The data has already been cleaned and it comes from 29 students in 3 different groups in a course of 26 weeks.

You already used this data in week 03.

In this lab you will explore different models to predict the quiz grade.

```
# Load data
df= pd.read_csv(f'{DATA_DIR}grades_in_time.csv.gz')
df.head()
```

```
   student  week  studying_hours  group  quiz_grade
0        0     0            39.9      3         6.1
1        0     1            32.4      3         7.0
```

```
2        0     2            17.5      3         6.9
3        0     3            16.0      3         7.0
4        0     4            15.9      3         7.2
```

```
df.describe(include='all')
```

```
          student        week  studying_hours       group  quiz_grade
count  810.000000  810.000000      810.000000  810.000000  810.000000
mean    14.500000   13.000000       10.050617    1.933333    6.931975
std      8.660789    7.793693        8.270041    0.772199    1.336888
min      0.000000    0.000000        1.000000    1.000000    1.200000
25%      7.000000    6.000000        5.700000    1.000000    6.400000
50%     14.500000   13.000000        7.800000    2.000000    7.200000
75%     22.000000   20.000000       11.100000    3.000000    7.800000
max     29.000000   26.000000       64.000000    3.000000   10.100000
```

Task 1: Predict the quiz grade using the studying hours and the group.

1.1 Split the data. 80% to train and the rest to test.

```
X = df[['studying_hours', 'group']]
y = df['quiz_grade']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state=0)
```

1.2 Preprocess the data

Recall that group is a categorical feature.

Hint: Use ColumnTransformer.

```
preprocessor = ColumnTransformer([
    ('categorical', OneHotEncoder(handle_unknown='ignore', drop =
'first'), ['group']),
    ('numerical', MinMaxScaler(),['studying_hours'])
])

preprocessor.fit_transform(X_train)
```

```
array([[0.        , 0.        , 0.07482993],
       [0.        , 1.        , 0.13945578],
       [0.        , 0.        , 0.11904762],
       ...,
       [1.        , 0.        , 0.08333333],
       [0.        , 0.        , 0.30782313],
       [0.        , 1.        , 0.16156463]])
```

1.3 Create a pipeline (including the preprocessing steps) to predict the quiz grade using the studying hours and the group.

1. Use the model ElasticNet for the regression task.
2. Calculate the mean sqaured error of the prediction.

Hint: Integrate the ColumnTransformer as a pipeline step

```
pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('model', ElasticNet())
])

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
error = round(mean_squared_error(y_test, y_pred),3)
print(f"Mean Squared Error = {error}")
```

```
Mean Squared Error = 1.582
```

1.4 Compute the cross validation score

```
# Fit a pipeline with transformers and an estimator to the training
data
pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('model', ElasticNet())
])

(-1)*np.mean(cross_val_score(pipe, X, y, cv = 5, scoring =
'neg_mean_squared_error'))
```

```
1.8575008478128328
```

1.5 Does the score in 1.3 differ from the score in 1.4? Why?

Answer = 1.3 is one fold and 1.4 is the average of multiple folds.

1.6 What is wrong with data split?

Answer: We are using the future sometimes to predict the past

```
df.iloc[X_test.index][['week','student']]
```

```
     week  student
613    19       22
202    13        7
55      1        2
478    19       17
27      0        1
..    ...      ...
749    20       27
71     17        2
49     22        1
416    11       15
240    24        8

[162 rows x 2 columns]
```

Task 2: Time Validation

2.1 Train with the first 25 weeks and predict week 26.

Hint: You may re-use your pipeline

```
df_train = df.query('week < 26')
df_test = df.query('week == 26')

X_train = df_train[['studying_hours', 'group','week']]
y_train = df_train['quiz_grade']

X_test = df_test[['studying_hours', 'group','week']]
y_test = df_test['quiz_grade']

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
error = round(mean_squared_error(y_test, y_pred),3)
print(f"Mean Squared Error = {error}")
```

```
Mean Squared Error = 1.86
```

2.2 Time splits

Would the model also be able to predict week 16 from all the previous weeks?

What about week 5 from the previous weeks?

Create all the data splits so that the model predicts the next week given the information from the previous weeks.

```
time_splits = [tuple([list(df.query('week < @i').index),
list(df.query('week == @i').index)]) for i in range(4,27)]
```

2.3 Using the previously created splits, calculate the cross validation score

```
X = df[['studying_hours', 'group']]
y = df['quiz_grade']

errors = (-1)*cross_val_score(pipe, X, y, cv = time_splits, scoring =
'neg_mean_squared_error')
np.mean(errors)
```

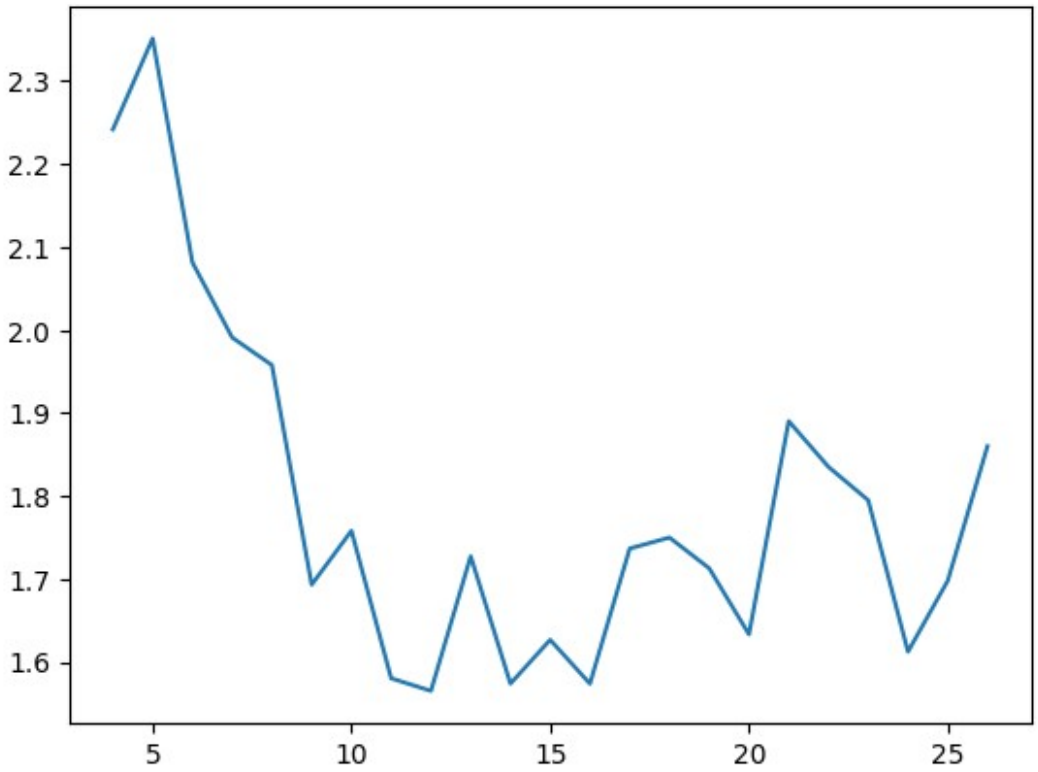
```
1.7933446924429217
```

2.4 How does the error differ from the error of 2.1? Why?

Answer = It is much higher. On the following plot we see that the more weeks (more information), the smaller the error

```
sns.lineplot(y = errors, x = list(range(4,27)))
```

```
<AxesSubplot:>
```



Task 3: Nested cross-validation

Now imagine we want to optimize the hyperparameters for the model.

We will "ignore" time for now and take the mean studying hours and quiz grade.

```
df_agg = df.groupby('student').mean()

X = df_agg[['studying_hours', 'group']]
y = df_agg['quiz_grade']
```

3.1 Gridsearch with cross validation

ElasticNet has two interesting parameters: alpha and l1_ratio.

Run a GridSearch to explore the following values:

- alpha = 0.1 and 1
- l1_ratio = 0.1, 0.5 and 1

What is the best score (smallest error)?

```
param_grid = {'model__alpha': [0.1, 1],
              'model__l1_ratio': [0.1, 0.5, 1]}

search = GridSearchCV(pipe, param_grid, n_jobs=-1, cv =
KFold(n_splits=4, shuffle=True, random_state=123) ,
```

```
                      scoring = 'neg_mean_squared_error')
search.fit(X,y)
```

```
GridSearchCV(cv=KFold(n_splits=4, random_state=123, shuffle=True),
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('categorical',

OneHotEncoder(drop='first',

handle_unknown='ignore'),

['group']),

('numerical',

MinMaxScaler(),

['studying_hours'])])),
                                       ('model', ElasticNet())]),
             n_jobs=-1,
             param_grid={'model__alpha': [0.1, 1],
                         'model__l1_ratio': [0.1, 0.5, 1]},
             scoring='neg_mean_squared_error')
```

```
pd.DataFrame(search.cv_results_)
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.084765      0.040885         0.124522        0.042845
1       0.190319      0.072629         0.176377        0.086030
2       0.250369      0.046773         0.152884        0.110980
3       0.224204      0.042654         0.126463        0.106406
4       0.248766      0.049980         0.101051        0.065776
5       0.197975      0.066664         0.053716        0.045144

  param_model__alpha param_model__l1_ratio  \
0                0.1                   0.1
1                0.1                   0.5
2                0.1                     1
3                  1                   0.1
4                  1                   0.5
5                  1                     1

                                       params  split0_test_score  \
0  {'model__alpha': 0.1, 'model__l1_ratio': 0.1}          -0.357376
1  {'model__alpha': 0.1, 'model__l1_ratio': 0.5}          -0.382699
2    {'model__alpha': 0.1, 'model__l1_ratio': 1}          -0.382699
3    {'model__alpha': 1, 'model__l1_ratio': 0.1}          -0.382699
4    {'model__alpha': 1, 'model__l1_ratio': 0.5}          -0.382699
5      {'model__alpha': 1, 'model__l1_ratio': 1}          -0.382699
```

```
   split1_test_score  split2_test_score  split3_test_score
mean_test_score  \
0          -0.261542          -0.108918          -0.346856        -
0.268673
1          -0.283716          -0.121981          -0.342687        -
0.282771
2          -0.289399          -0.126909          -0.340410        -
0.284854
3          -0.289399          -0.126909          -0.340410        -
0.284854
4          -0.289399          -0.126909          -0.340410        -
0.284854
5          -0.289399          -0.126909          -0.340410        -
0.284854

   std_test_score  rank_test_score
0        0.099440                1
1        0.099285                2
2        0.096989                3
3        0.096989                3
4        0.096989                3
5        0.096989                3
```

```
(-1)*search.best_score_
```

```
0.2686728849527697
```

3.2 Why is the error from the best model in 3.1 biased?

Answer = We are using the same data to tune model parameters and evaluate model performance.

3.3 Improve 3.1 to have an unbiased estimation of the generalization error

Hint: Use nested cross-validation

```
inner_cv = KFold(n_splits=4, shuffle=True, random_state=123)
outer_cv = KFold(n_splits=3, shuffle=True, random_state=123)

param_grid = {'model__alpha': [0.1, 1],
              'model__l1_ratio': [0.1, 0.5, 1]}

search = GridSearchCV(pipe, param_grid, n_jobs=-1,
                      cv = inner_cv, scoring =
'neg_mean_squared_error')
errors = (-1)* cross_val_score(search, X=X, y=y,
                               cv=outer_cv)

np.mean(errors)
```

```
0.27392204035478573
errors
array([0.301807  , 0.266251  , 0.25370813])
```

Lab 6 - Bayesian Knowledge Tracing (BKT) and Variants

This tutorial is partially based on the pyBKT model tutorial and the Jupyter notebooks available on GitHub at https://github.com/CAHLR/pyBKT.

One notable application of machine learning in education is represented **knowledge inference models**, which aim to understand how well a student is learning concepts or skills. Being able to monitor this knowledge makes it possible to improve and personalize online learning platforms or intelligent tutoring systems, by focusing on areas the student is weak in and accelerating learning of certain concepts.

In this tutorial, we study a range of popular models for modelling students' knowledge based on **Bayesian Knowledge Tracing (BKT)**. BKT was introduced in 1995 as a means to model students' knowledge as a **latent variable** in online learning environments. Specifically, the environment can maintain an estimate of the **probability that the student has learned a set of skills**, which is statistically equivalent to a 2-node dynamic Bayesian network.

For this tutorial, we will rely on a Python implementation of the Bayesian Knowledge Tracing algorithm and more recent variants, estimating student cognitive mastery from problem solving sequences, known under the name of **pyBKT**. This package can be used to define and fit many BKT variants.

These variants are derived from a range of papers published in the educational data mining literature and, in this tutorial, we will provide you with the main notions and implementation details needed to investigate BKT models in practice.

Expected Tasks

- Follow the pyBKT getting started showcase.
- Solve a range of exercises on BKT models.

Learning Objectives

- Instantiate and run a pipeline on BKT models.
- Conduct fine-grained analyses on specific learning skills.
- Understand and experiment with different variants of BKT.
- Compare the performance of BKT setups under different evaluation methods.
- Inspect the influence of a BKT variant on the internal BKT parameters.

More information on the PyBKT is provided in the corresponding Github repository.

```
# Traditional packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

```
import math

%matplotlib inline
```

Introduction

BKT models operationalize the learning of a student as a **Markov process**, building upon the idea that, while students interact with an educational environment, their skill in a given concept improves. To move the theory behind BKT into practice, variables related to forgetting, learning, guessing, slipping, and so on need to be modelled, controlling for instance how fast and how well learning is happening for the student.

The BKT model assumes that the student's knowledge can be estimated by means of standardized questions, which can be answered correctly or incorrectly, on a concept or combination of concepts. BKT also assumes that initially a student may not know about a concept, but their knowledge gets better with learning and practice related to that concept. The following concepts will be

- P_0 is the initial probability of mastering that concept (skill).
- P_F is the probability that the student forgot something previously learned on the concept (skill).
- P_L is the probability that the student has learned something that was previous not known on the concept (skill).
- P_S is the probability that the student gave a wrong answer even though they had learned the concept (skill).
- P_G is the probability that the student guessed the right answer while not knowing the concept (skill).

In this tutorial, we will use a dataset of the student's responses to questions in a test, along with whether they answered correctly or incorrectly, and we will use a BKT model to find the values of the above probabilities.

The ASSISTments data set

ASSISTments is a free tool for assigning and assessing math problems and homework. Teachers can select and assign problem sets. Once they get an assignment, students can complete it at their own pace and with the help of hints, multiple chances, and immediate feedback. Teachers get instant results broken down by individual student or for the whole class. Please, find more information on the platform here.

In this tutorial, we will play with a simplified version of a dataset collected from the ASSISTments tool, saved on a CSV files with the following columns:

- user_id: The ID of the student doing the problem.
- template_id: The ID of the template in ASSISTment (assistments with the same template ID have similar questions).

- assistment_id: The ID of the ASSISTment (an assistment consists of one or more problems).
- order_id: These IDs are chronological and refer to the id of the original problem log.
- problem_id: The ID of the problem.
- skill_name: Skill name associated with the problem.
- correct: 1 if correct on the first attempt, 0 if incorrect on the first attempt or asked for help.
- ms_first_response: The time in milliseconds for the student's first response.
- attempt_count: Number of student attempts on this problem.
- hint_count: Number of student hints asked by the student on this problem.
- hint_total: Number of possible hints to be asked on this problem.

```
DATA_DIR = "./../../data/"
as_data = pd.read_csv(DATA_DIR + 'as_supersmall.csv',
encoding='latin', low_memory=False)

as_data.head(10)
```

```
   user_id  template_id  assistment_id  order_id  problem_id
skill_name  \
0    70733        30060          33175  35278766       51460  Box and
Whisker
1    70733        30060          33182  35278780       51467  Box and
Whisker
2    70733        30059          33107  35278789       51392  Box and
Whisker
3    70733        30060          33187  35278802       51472  Box and
Whisker
4    70733        30059          33111  35278810       51396  Box and
Whisker
5    70872        30059          33136  32268742       51421  Box and
Whisker
6    70872        30799          33144  32268764       51429  Box and
Whisker
7    72059        30799          33155  33409110       51440  Box and
Whisker
8    72059        30060          33181  33409165       51466  Box and
Whisker
9    72059        30060          33168  33409366       51453  Box and
Whisker

   correct  ms_first_response  attempt_count  hint_count  hint_total
0        0               9575              2           0           4
1        1               6422              1           0           4
2        0              11365              3           0           3
3        1               4412              1           0           4
4        1               6902              1           0           3
5        1               7281              1           0           3
6        1               7234              1           0           3
```

```
7         0              38290              2           0           3
8         0               8366              4           0           4
9         1               9661              1           0           4
```

Before delving into the pyBKT description and showcase, we invite you to spend some time to explore the toy dataset presented in this tutorial, e.g., how many students/problems/skills are included, examine the skills in more detail etc. Here, you could therefore add one or more cells to perform your exploration.

The pyBKT Package

In this tutorial, we use the pyBKT package, a Python implementation of the Bayesian Knowledge Tracing algorithm and variants, estimating student cognitive mastery from problem solving sequences. First, we install the package:

```
%pip install pyBKT
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: pyBKT in /usr/local/lib/python3.8/dist-
packages (1.4)
Requirement already satisfied: requests in
/usr/local/lib/python3.8/dist-packages (from pyBKT) (2.28.1)
Requirement already satisfied: sklearn in
/usr/local/lib/python3.8/dist-packages (from pyBKT) (0.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-
packages (from pyBKT) (1.22.4)
Requirement already satisfied: pandas in
/usr/local/lib/python3.8/dist-packages (from pyBKT) (1.4.3)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.8/dist-packages (from pandas->pyBKT) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.8/dist-packages (from pandas->pyBKT) (2022.2.1)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.8/dist-packages (from requests->pyBKT)
(2022.6.15)
Requirement already satisfied: charset-normalizer<3,>=2 in
/usr/local/lib/python3.8/dist-packages (from requests->pyBKT) (2.1.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.8/dist-packages (from requests->pyBKT)
(1.26.12)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.8/dist-packages (from requests->pyBKT) (3.3)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.8/dist-packages (from sklearn->pyBKT) (1.0.2)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-
packages (from python-dateutil>=2.8.1->pandas->pyBKT) (1.14.0)
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.8/dist-packages (from scikit-learn->sklearn-
>pyBKT) (1.1.0)
```

```
Requirement already satisfied: scipy>=1.1.0 in
/usr/local/lib/python3.8/dist-packages (from scikit-learn->sklearn-
>pyBKT) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.8/dist-packages (from scikit-learn->sklearn-
>pyBKT) (3.1.0)
Note: you may need to restart the kernel to use updated packages.
```

Then, we can import the core class provided by the package, that is Model.

```
from pyBKT.models import Model
```

The first step is to construct a BKT model. To be instantiated, a BKT model requires a series of parameters, whose default value and meaning is provided below (e.g., num_fits, seed, defaults, and any model variant(s) that may be used). Each parameter can be modified during fit/crossvalidation time too.

- **Defaults generic parameters**:
  - num_fits (5) is the number of initialization fits used for the BKT model.
  - defaults (None) is a dictionary that can be used to pass values different than the default ones during initialization.
  - parallel (True) indicates whether the computation will use multi-threading.
  - skills ('.*') is a regular expression used to indicate the skills the BKT model will be run on.
  - seed (random.randint(0, 1e8)) is a seed that can be setup to enable reproducible experiments.
  - folds (5) is the number of folds used in case of cross-validation.
  - forgets (False) indicates whether the model will consider that the student may give a wrong answer even though they had learned the concept.
- **Defaults additional parameters**:
  - order_id ('order_id') is the name of the CSV column for the chronological IDs that refer to the original problem log.
  - skill_name ('skill_name') is the name of the CSV column for the skill name associated with the problem.
  - correct ('correct') is the name of the CSV column for the correct / incorrect label on the first attempt.
  - user_id ('user_id') is the name of the CSV column for the ID of the student doing the problem.
  - multilearn ('template_id') is the name of the column for checking whether there is a multi-skill object.
  - multiprior ('correct') is the name of the CSV column for mapping multi-prior knowledge.
  - multigs ('template_id') is the name of the CSV column corresponding to the desired guess/slip classes.

- **Initializers for learnable parameters**:
  - 'prior' (None, no inizialization) is the initial probability of answering the question correct.
  - 'learns' (None, no inizialization) is the probability that the student has learned something that was previous not known.
  - 'guesses' (None, no inizialization) is the probability that the student guessed the right answer while not knowing the concept.
  - 'slips' (None, no inizialization) is the probability that the student gave a wrong answer even though they had learned the concept.
  - 'forgets' (None, no inizialization) is the probability that the student forgot something previously learned.

If you have doubts on the meaning of certain parameters, please ask to TAs or move on the next examples (they will help you understand).

```
model = Model(seed=0)
model
```

```
Model(parallel=True, num_fits=5, seed=0, defaults=None)
```

The Model class is inspired by scikit-learn and, therefore, provides a range of methods a model can be called with:

- The **fit** method fits a BKT model given model and data information. Takes arguments skills, number of initialization fits, default column names (i.e. correct, skill_name), parallelization, and model types.
- The **predict** method predicts using the trained BKT model and test data information. Takes test data path or DataFrame as arguments. Returns a dictionary mapping skills to predicted values for those skills. Note that the predicted values are a tuple of (correct_predictions, state_predictions).
- The **evaluate** method evaluates a BKT model given model and data information. Takes a metric and data path or DataFrame as arguments. Returns the value of the metric for the given trained model tested on the given data.
- The **crossvalidate** method crossvalidates (trains and evaluates) the BKT model. Takes the data, metric, and any arguments that would be passed to the fit function (skills, number of initialization fits, default column names, parallelization, and model types) as arguments.

We will show a range of examples for each of the above methods.

Fitting and evaluating a model

```
model = Model(seed=0)
%time model.fit(data=as_data, skills='Box and Whisker')
%time model.evaluate(data=as_data, metric='auc')
```

```
CPU times: user 682 ms, sys: 12 ms, total: 694 ms
Wall time: 311 ms
```

```
CPU times: user 152 ms, sys: 0 ns, total: 152 ms
Wall time: 94.6 ms
```

```
0.5674965857758167
```

First, we have fitted a BKT model on the 'Box and Whisker' skill and, then, evaluate the corresponding **training AUC** (0.64). Note that we have run the BKT fitting process on the full dataset, to understand how well the BKT model can fit the data. Evaluation methods like cross-validation will be presented later in this notebook. Furthermore, the default metric displayed is RMSE, but pyBKT supports AUC ('auc'), RMSE ('rmse'), and accuracy ('accuracy') as metrics. We will also see how to add other metrics.

For each skill, you can get the learned parameters for 'prior', 'learns', 'guesses', 'slips', and 'forgets'. Specifically:

- **prior** (P_0¿ the prior probability of "knowing".
- **forgets** (P_F¿: the probability of transitioning to the "not knowing" state given "known".
- **learns** (P_L): the probability of transitioning to the "knowing" state given "not known".
- **slips** (P_S): the probability of picking incorrect answer, given "knowing" state.
- **guesses** (P_G): the probability of guessing correctly, given "not knowing" state.

```
model.coef_
```

```
{'Box and Whisker': {'prior': 0.8533977868556804,
  'learns': array([0.19638124]),
  'guesses': array([0.24682242]),
  'slips': array([0.21955686]),
  'forgets': array([0.])}}
```

```
model.params()
```

			value
skill	param	class	
Box and Whisker	prior	default	0.85340
	learns	default	0.19638
	guesses	default	0.24682
	slips	default	0.21956
	forgets	default	0.00000

We could initialize the prior knowledge to $1e-40$ for Box and Whisker, before fitting the model.

```
model = Model(seed=0)

model.coef_ = {'Box and Whisker': {'prior': 1e-40}}
model.coef_
```

```
{'Box and Whisker': {'prior': 1e-40}}
```

Then, we can fit the model and observe the resulting AUC score. How does it compares to the AUC score of the previous model.

```
%time model.fit(data=as_data, skills='Box and Whisker')
%time model.evaluate(data=as_data, metric='auc')
```

```
CPU times: user 261 ms, sys: 0 ns, total: 261 ms
Wall time: 130 ms
CPU times: user 112 ms, sys: 0 ns, total: 112 ms
Wall time: 48.8 ms
```

```
0.5535245298875933
```

```
model.params()
```

```
                                  value
skill           param   class
Box and Whisker prior   default 0.00000
                learns  default 0.56145
                guesses default 0.62351
                slips   default 0.22136
                forgets default 0.00000
```

You can also train simple BKT models on different skills in the data set.

```
model = Model(seed=0)
%time model.fit(data=as_data, skills=['Box and Whisker', 'Scatter
Plot'])
%time model.evaluate(data=as_data, metric='auc')
```

```
CPU times: user 248 ms, sys: 0 ns, total: 248 ms
Wall time: 149 ms
CPU times: user 86.6 ms, sys: 0 ns, total: 86.6 ms
Wall time: 54.3 ms
```

```
0.6625740160222918
```

And, then, observed the learned parameters for each skill. Note that, when multiple skills are passed to fit, the method will run a fitting procedure for each skill, separately (in this case, we will have two BKT models).

```
model.params()
```

```
                                  value
skill           param   class
Box and Whisker prior   default 0.96146
                learns  default 0.01810
                guesses default 0.00036
                slips   default 0.23364
                forgets default 0.00000
Scatter Plot    prior   default 0.48923
                learns  default 0.56128
```

```
                guesses default 0.71140
                slips   default 0.03956
                forgets default 0.00000
```

You can also enable forgetting, by setting the corresponding parameter in the fit method.

```
model = Model(seed=0)
%time model.fit(data=as_data, skills='Box and Whisker', forgets=True)
%time model.evaluate(data=as_data, metric='auc')
```

```
CPU times: user 73.4 ms, sys: 0 ns, total: 73.4 ms
Wall time: 49.4 ms
CPU times: user 125 ms, sys: 0 ns, total: 125 ms
Wall time: 39.8 ms
```

```
0.5721189200546275
```

```
model.params()
```

```
                                  value
skill           param   class
Box and Whisker prior   default 0.72657
                learns  default 0.25751
                guesses default 0.36213
                slips   default 0.19171
                forgets default 0.01321
```

Or train a multiguess and slip BKT model on the same skills in the data set. The **multigs** model fits a different guess/slip rate for each class. Note that, with *multigs=True*, the guess and slip classes will be specified by the *template_id*. You can specify a custom column mapping by doing *multigs='column_name'*.

```
model = Model(seed=0)
%time model.fit(data=as_data, skills=['Box and Whisker'],
multigs=True)
%time model.evaluate(data=as_data, metric='auc')
```

```
CPU times: user 173 ms, sys: 0 ns, total: 173 ms
Wall time: 62.1 ms
CPU times: user 128 ms, sys: 0 ns, total: 128 ms
Wall time: 116 ms
```

```
0.7049059775186469
```

And finally, we show the BKT paramaters. By enabling *multigs*, the guess and slip classes will be specified by the template_id and, by setting *multigs=True*, the guess and slip classes will be specified by default by the template_id classes. Note that assistments with the same template ID have similar questions. What could you observe by looking at the different learned guesses and slips values below?

```
model.params()
```

```
                                value
skill           param   class
Box and Whisker prior   default 0.09283
                learns  default 0.07509
                guesses 30059   0.75371
                        30060   0.59704
                        30799   0.70751
                        63446   0.00000
                        63447   0.16724
                        63448   1.00000
                slips   30059   0.01865
                        30060   0.16663
                        30799   0.04409
                        63446   1.00000
                        63447   0.99707
                        63448   0.00000
                forgets default 0.00000
```

The **multilearn** model fits a different learn rate (and forget rate if enabled) rate for each class specified. Note that, with multilearn=True, the learn classes are specified by the *template_id*. You can specify a custom column mapping by doing *multilearn='column_name'*.

```
model = Model(seed=0)
%time model.fit(data=as_data, skills=['Box and Whisker'],
multilearn=True)
%time model.evaluate(data=as_data, metric='auc')
```

```
CPU times: user 108 ms, sys: 0 ns, total: 108 ms
Wall time: 55.3 ms
CPU times: user 119 ms, sys: 3.37 ms, total: 122 ms
Wall time: 40.4 ms
```

```
0.5694925937598487
```

Looking at the parameters, we will observe a `leanrs` score for each template_id (the class column in the paras dataframe). In this case, what could you observe by looking at the different `leanrs` values below?

```
model.params()
```

```
                                value
skill           param   class
Box and Whisker prior   default 0.94327
                learns  30059   0.02049
                        30060   0.42985
                        30799   0.25272
                        63446   0.84344
                        63447   0.66021
                        63448   0.25677
                guesses default 0.05763
                slips   default 0.23095
```

```
                forgets 30059   0.00000
                        30060   0.00000
                        30799   0.00000
                        63446   0.00000
                        63447   0.00000
                        63448   0.00000
```

You can also combine multiple variants, and use a different column to specify the different learn and forget classes. In this case, we use user_id, assuming that we are interested in learning the parameters for each student, and we also enable forgetting.

```
model = Model(seed=0)
%time model.fit(data=as_data, skills=['Box and Whisker'],
forgets=True, multilearn='user_id')
%time model.evaluate(data=as_data, metric='auc')
```

```
CPU times: user 777 ms, sys: 1.33 ms, total: 778 ms
Wall time: 368 ms
CPU times: user 106 ms, sys: 362 µs, total: 106 ms
Wall time: 121 ms
```

```
0.6405609832965647
```

Once we run a BKT model with `forgets=True` and `multilearn='user_id'`, we will observe individual scores for each student, as shown below.

```
model.params()
```

```
                                  value
skill           param   class
Box and Whisker prior   default 0.05051
                learns  70733   0.08695
                        70872   0.11881
                        72059   0.07787
                        79748   0.11881
                        79750   0.13948
                        79769   0.10011
                        81641   0.02619
                        82482   0.12312
                        82533   0.16158
                        83169   0.11881
                        84316   0.07571
                        85725   0.13006
                        86489   0.09430
                        91260   0.11881
                        91301   0.11881
                        91406   0.16158
                        91409   0.11659
                        91436   0.01063
                        91459   0.10580
                        96217   0.10079
```

```
          96242   0.07787
          96248   0.06080
          96249   0.06670
          96266   0.11881
          96268   0.08741
          96292   0.07408
          96294   0.11825
          96296   0.11028
guesses default 0.70234
slips   default 0.05860
forgets 70733   0.31890
          70872   0.23699
          72059   0.35471
          79748   0.23699
          79750   0.22281
          79769   0.26551
          81641   0.66799
          82482   0.26814
          82533   0.19007
          83169   0.23699
          84316   0.35968
          85725   0.23422
          86489   0.34166
          91260   0.23699
          91301   0.23699
          91406   0.19007
          91409   0.28950
          91436   0.78903
          91459   0.32115
          96217   0.27475
          96242   0.35471
          96248   0.40840
          96249   0.42812
          96266   0.23699
          96268   0.26268
          96292   0.36373
          96294   0.27470
          96296   0.26723
```

The best performing models are typically those that combine several useful variants, such as the multilearn and multiguess/slip class variants. After this lab session, you might be interested in testing with other skills and see whether this observations is true for other skills as well.

Make predictions

As we said, the predict method can be executed on the trained BKT model, obtaining a dictionary mapping skills to predicted values for those skills, namely correct_predictions (each score is between 0 and 1 that measures the extent to which the model thinks that the student will answer correctly to that question) and state_predictions (each score between 0

and 1 that measures the extent to which the student has mastered that skill, after that question).

Note that, in the example below, we have run the BKT fitting process on the full dataset, to understand how well the BKT model can fit the data. Evaluation methods like cross-validation will be presented slightly after in this notebook.

```
model = Model(seed=0)
%time model.fit(data=as_data, skills='Box and Whisker')
%time model.evaluate(data=as_data, metric='auc')
%time preds = model.predict(data=as_data)
```

```
CPU times: user 83.1 ms, sys: 4.64 ms, total: 87.7 ms
Wall time: 48.6 ms
CPU times: user 137 ms, sys: 0 ns, total: 137 ms
Wall time: 57.2 ms
CPU times: user 195 ms, sys: 0 ns, total: 195 ms
Wall time: 98 ms
```

```
preds[preds['skill_name']=='Box and Whisker'][['user_id', 'correct',
'correct_predictions', 'state_predictions']]
```

	user_id	correct	correct_predictions	state_predictions
0	70733	0	0.66780	0.81542
1	70733	1	0.59418	0.69635
2	70733	0	0.74813	0.94535
3	70733	1	0.70860	0.88142
4	70733	1	0.77085	0.98210
..	...	...	...	...
219	96296	1	0.78191	1.00000
220	96296	1	0.78191	1.00000
221	96296	1	0.78191	1.00000
222	96296	1	0.78191	1.00000
223	96296	1	0.78191	1.00000

```
[224 rows x 4 columns]
```

Note that, if the BKT model is asked to predict on skills not included in the training set, the output predictions for that skills will be a best effort guess of 0.5 for both the correct and state predictions.

Extend the evaluation

The pyBKT package makes also possible to extend the range of metrics you can compute while evaluating a BKT model. To this end, you need to define a custom function that, given true_vals (true values for the correct target) and pred_vals (the predicted values for the correct target), computes and returns the score corresponding to the desired metric.

```
def mae(true_vals, pred_vals):
    return np.mean(np.abs(true_vals - pred_vals))
```

```
%time model.evaluate(data=as_data, metric=mae)
```

```
CPU times: user 200 ms, sys: 1.41 ms, total: 201 ms
Wall time: 60.9 ms
```

```
0.3699436448390422
```

Perform cross validation

Finally, the pyBKT package offers also a cross-validation method. You can specify the number of folds, a seed, and a metric (one of the three default ones, namely 'rmse', 'auc' or 'accuracy', or a custom Python function as we have see above). Furthermore, simialrly to the fit method, arguments for cross-validation a BKT variant and for defining the data path/data and skill names are accepted.

```
model = Model(seed=0)
%time model.crossvalidate(data=as_data, skills='Box and Whisker',
folds=5, metric='auc')
```

```
CPU times: user 3.06 s, sys: 3.31 ms, total: 3.06 s
Wall time: 1.5 s
```

```
                    auc
skill
Box and Whisker 0.54551
```

In this showcase, we just opted to five folds due to the time constraints. In the other cases, you need to select an appropriate number of folds based on the data you are dealing with, as discussed in the lectures.

Exercises

That's your turn! We ask you to complete the following exercises. In case you do not finish them during the lab session, please feel free to complete later, at your earliest convenience. TAs are happy to address any question or doubts you might have.

Kindly note that the following exercises have the goal of supporting you in getting familiar with the library functions, and may not fully represent the sequences of steps and the design choices made in a real-world or homework scenario. Elements concerning the latter scenarios will be discussed during the session. Furthermore, due to running time constraints, the following exercises will be run in a train-test split or full data set mode, while we leave the opportunity to run them under a cross-validation setting after this lab session.

In all your models, we ask you to set the *seed* to 0, to let you reproduce the same results across different runs.

Note that the expected running time may vary according to the device or environment.

Question 1 [expected total time for BKT fitting: 2 mins]

- Fit a BKT model with default parameters on the full data set, only for the skill 'Addition and Subtraction Integers'.
- Compute the correct predictions from the BKT model, by using the predict method of the Model class.
- Manually calculate the RMSE between the true correct value and the predicted correct value (refer to Slide 51 of Lecture 4 to get the RMSE formula).
- Compare with the RMSE returned by the evaluate method of the BKT model.

```
### EXERCISE CELL ###
def rmse(y_true, y_pred):
    return np.sqrt(np.mean((y_true - y_pred) ** 2))

model = Model(seed=0)
%time model.fit(data=as_data, skills='Addition and Subtraction
Integers')
preds = model.predict(data=as_data)

preds_filtered = preds[preds['skill_name'].str.contains('Addition and
Subtraction Integers')]
manual_training_rmse = rmse(preds_filtered['correct'],
preds_filtered['correct_predictions'])

print('Manual RMSE:', manual_training_rmse)
print('pyBKT RMSE:', model.evaluate(data=as_data, metric='rmse'))
```

```
CPU times: user 2.53 s, sys: 15.3 ms, total: 2.55 s
Wall time: 1.28 s
Manual RMSE: 0.46390773499967014
pyBKT RMSE: 0.46390773499967064
```

Question 2 [expected total time for BKT fitting: 7 mins]

- Perform a user-based train-test split of the data, with 20% of the users in the test set.
- Fit the two BKT model variants on the training set, only for the skill 'Addition and Subtraction Integers'.
  - default;
  - forgets=True;
- Which model variant listed below has the highest test AUC for 'Addition and Subtraction Integers' in the test set?

```
### EXERCISE CELL ###
users = as_data['user_id'].unique()
users_train = list(np.random.choice(users, int(len(users) * 0.8),
replace=False))
users_test = list(set(users) - set(users_train))

X_train, X_test = as_data[as_data['user_id'].isin(users_train)],
as_data[as_data['user_id'].isin(users_test)]
```

```
models = {}

model = Model(seed=0, num_fits=1)
%time model.fit(data=X_train, skills='Addition and Subtraction
Integers')
models['simple'] = model.evaluate(data=X_test, metric='auc')

model = Model(seed=0, num_fits=1)
%time model.fit(data=X_train, skills='Addition and Subtraction
Integers', forgets=True)
models['forgets'] = model.evaluate(data=X_test, metric='auc')

df = pd.DataFrame(models.items())
df.columns = ['Model Type', 'AUC']
df.set_index('Model Type')
```

```
CPU times: user 447 ms, sys: 4.42 ms, total: 451 ms
Wall time: 182 ms
CPU times: user 733 ms, sys: 0 ns, total: 733 ms
Wall time: 388 ms
```

Model Type	AUC
simple	0.55297
forgets	0.53747

Question 3 [expected total time for BKT fitting: 3 mins]

- Bin values in the ms_first_response column in *as_data* to categories ('less than 10s', 'less than 20s', 'less than 30s','less than 40s', 'less than 50s', 'other').
- Fit BKT models with different learn rates, according to the ms_first_response categories above, on the full data set, only for the skill 'Addition and Subtraction Integers'. You need to play with the multilearn parameter of the BKT fit method.
- Create a bar plot to show the P_L (learns) value for each ms_first_response category above. You basically need to play with the dataframe returned by model.params(), to prepare the data to be shown in the plot.
- Does binned response time influence the P_L parameter for the skill 'Addition and Subtraction Integers'? Which bin result in the highest P_L scores?

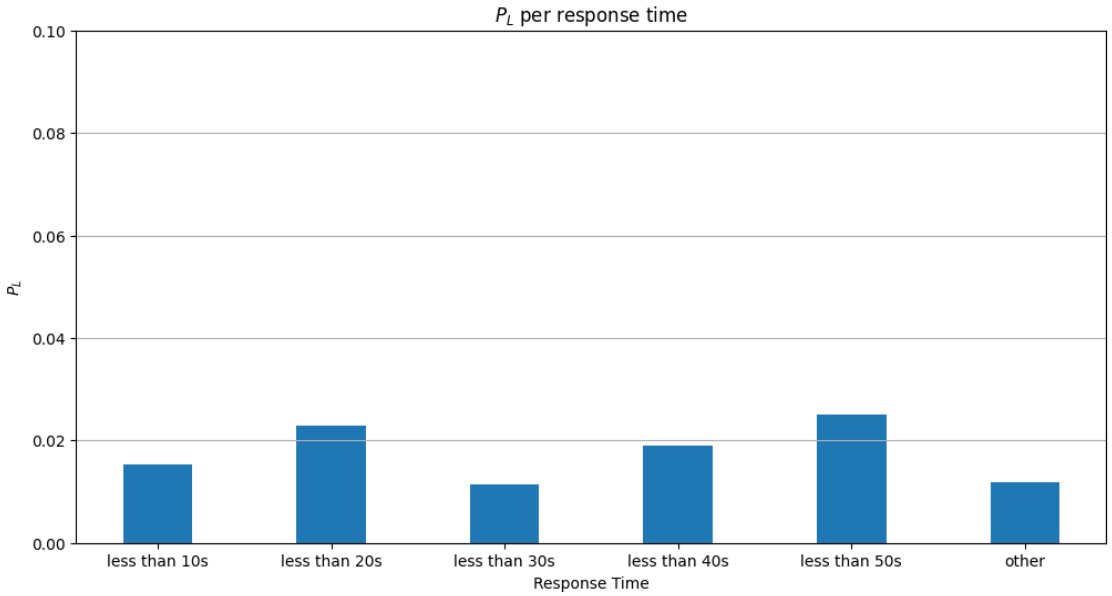
```
### EXERCISE CELL ###

skill = 'Addition and Subtraction Integers'

# Binning
learn_maps = {0: 'less than 10s', 1: 'less than 20s', 2: 'less than
30s', 3: 'less than 40s', 4: 'less than 50s'}
as_data['bin_s_first_response'] = (as_data['ms_first_response'] // (10
* 1000)).map(learn_maps).fillna('other')
```

```
# Modelling
model = Model(seed=0, num_fits=1)
%time model.fit(data=as_data, skills=skill,
multilearn='bin_s_first_response')
params = model.params().reset_index()

params
```

```
CPU times: user 496 ms, sys: 0 ns, total: 496 ms
Wall time: 272 ms
```

	skill	param	class	value
0	Addition and Subtraction Integers	prior	default	0.74974
1	Addition and Subtraction Integers	learns	less than 10s	0.01536
2	Addition and Subtraction Integers	learns	less than 20s	0.02294
3	Addition and Subtraction Integers	learns	less than 30s	0.01146
4	Addition and Subtraction Integers	learns	less than 40s	0.01898
5	Addition and Subtraction Integers	learns	less than 50s	0.02503
6	Addition and Subtraction Integers	learns	other	0.01188
7	Addition and Subtraction Integers	guesses	default	0.37005
8	Addition and Subtraction Integers	slips	default	0.20730
9	Addition and Subtraction Integers	forgets	less than 10s	0.00000
10	Addition and Subtraction Integers	forgets	less than 20s	0.00000
11	Addition and Subtraction Integers	forgets	less than 30s	0.00000
12	Addition and Subtraction Integers	forgets	less than 40s	0.00000
13	Addition and Subtraction Integers	forgets	less than 50s	0.00000
14	Addition and Subtraction Integers	forgets	other	0.00000

```
plt.figure(figsize = (12, 6))
plt.title(r'$P_{L}$ per response time')
params_learns = params[params['param'] ==
'learns'].sort_values(by='class').copy()
labels = params_learns['class']
values = params_learns['value']
plt.bar(labels, values, width=0.4)
plt.ylabel(r'$P_{L}$')
plt.xlabel('Response Time')
plt.ylim([0, .1])
plt.grid(axis='y')
plt.show()
```



Question 4 [expected total time for BKT fitting: 8 mins]

- Use the same bins ms_first_response to categories ('less than 10s', 'less than 20s', 'less than 30s','less than 40s', 'less than 50s', 'other').
- Fit a BKT model with template-id multilearn (default), on the full data set, only for the skill 'Addition and Subtraction Integers'.
- Fit a BKT model with binned-response-time-based multilearn, on the full data set, only for the skill 'Addition and Subtraction Integers'.
- Does the binned-response-time-based multilearn improve the AUC of the model compared to the default template_id-based multilearn?

```
### EXERCISE CELL ###
skill = 'Addition and Subtraction Integers'

model = Model(seed=0, num_fits=1)
%time model.fit(data=as_data, skills=skill, multilearn=True)
default_multilearn_auc = model.evaluate(data=as_data, metric='auc')

model = Model(seed=0, num_fits=1)
%time model.fit(data=as_data, skills=skill,
multilearn='bin_s_first_response')
time_multilearn_auc = model.evaluate(data=as_data, metric='auc')

'AUC Improvement using Response Time:', time_multilearn_auc -
default_multilearn_auc
```

```
CPU times: user 455 ms, sys: 17.5 ms, total: 473 ms
Wall time: 162 ms
CPU times: user 420 ms, sys: 8.13 ms, total: 428 ms
Wall time: 205 ms

('AUC Improvement using Response Time:', -0.0031074282332050895)
```

Summary

In this tutorial, we have seen several important aspects of Bayesian Knowledge Tracing (BKT). We have shown how a typical data set for knowledge tracing should look like. We have illustrated how BKT models can be trained on different skills. We have shown how different variants of BKT can help you improve the goodness of your model. Many of the ideas described in this tutorial can be adapted to other data sets and projects. Finally, we have shown some examples of predictions and evaluations, covering also cross-validation. If you are interested in the implementation details of the different variants, we invite you to explore the codebase stored in the pyBKT Github repository.

Lab 07 Solution - Knowledge Tracing

Introduction

During the last lectures and lab session, you have dealt into one notable application of machine learning in education, namely knowledge tracing. Machine-learning models optimized for this task aim to understand how well a student is learning a portfolio of skills. Monitoring this knowledge by means of automated models allows to personalize online learning platforms, focusing the assessment on skills the student is weak in and accelerating learning of certain skills.

You are asked to work on the ASSISTment data set presented last week and to complete the following tasks:

- Compare three knowledge tracing models (BKT, AFM, PFA) in terms of AUC and RMSE.
- Generate and discuss the learning curves for a BKT model on a specific set of skills.

You can use pyBKT and pyAFM throughout this tutorial.

```
# Principal package imports
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import scipy as sc

# Scikit-learn package imports
from sklearn import feature_extraction, model_selection, metrics

### YOUR ADDITIONAL IMPORT STATEMENTS BELOW (please, do not make any
imports elsewhere in the notebook) ###

# PyBKT package imports
from pyBKT.models import Model

# PyAFM package imports
from pyafm.custom_logistic import CustomLogistic
```

The Data Set

ASSISTments is a free tool for assigning and assessing math problems and homework. Teachers can select and assign problem sets. Once they get an assignment, students can complete it at their own pace and with the help of hints, multiple chances, and immediate feedback. Teachers get instant results broken down by individual student or for the whole class. More information on the platform can be found here.

We will play with a simplified version of a dataset collected from the ASSISTments tool, saved in a CSV file with the following columns:

Name	Description
user_id	The ID of the student who is solving the problem.
order_id	The temporal ID (timestamp) associated with the student's answer to the problem.
problem_id	The ID of the problem.
skill_name	The name of the skill associated with the problem.
correct	The student's performance on the problem: 1 if the problem's answer is correct at the first attempt, 0 otherwise.
prior_success	The number of prior problems on that skill the student correctly answered at the first attempt.
prior_failure	The number of prior problems on that skill the student wrongly answered at the first attempt.

Load the data set.

```
DATA_DIR = "./../../data/"
data = pd.read_csv(DATA_DIR + 'as_hw_cmp.csv')
```

As a first step, we compute the total number of interactions, the number of unique students, and the number of unique skills.

```
len(data.index), len(data['user_id'].unique()),
len(data['skill_name'].unique())
```

```
(26409, 1014, 3)
```

We then also take a look at the skills included in the data set.

```
data['skill_name'].unique()
```

```
array(['Circle Graph', 'Venn Diagram', 'Mode'], dtype=object)
```

Finally, we show the first ten rows of the data dataframe, to have an idea of how the data looks like.

```
data.head(20)
```

```
    user_id  order_id  problem_id    skill_name  correct
prior_success  \
0        14  21617623       93383  Circle Graph        0
0
1        14  21617632       93407  Circle Graph        1
0
2        14  21617641       93400  Circle Graph        0
1
3        14  21617650       93419  Circle Graph        0
```

1
4 14 21617659 93420 Circle Graph 0
1
5 14 21617667 93415 Circle Graph 0
1
6 14 21617675 93423 Circle Graph 0
1
7 14 21617692 57695 Circle Graph 0
1
8 14 21617731 58596 Circle Graph 1
1
9 14 21617749 57647 Circle Graph 0
2
10 14 21617805 58566 Circle Graph 1
2
11 14 21617825 58551 Circle Graph 0
3
12 64525 28186893 92320 Circle Graph 1
0
13 64525 28187093 92335 Circle Graph 1
1
14 64525 32413158 92327 Circle Graph 1
2
15 64525 33022751 93432 Circle Graph 0
3
16 64525 33023039 93447 Circle Graph 1
3
17 64525 33023131 93448 Circle Graph 1
4
18 64525 33023183 93429 Circle Graph 1
5
19 64525 33023245 57689 Circle Graph 0
6

	prior_failure
0	0
1	1
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	7
10	8
11	8
12	0
13	0
14	0

```
15              0
16              1
17              1
18              1
19              1
```

1 Knowledge Tracing: Model Performance Comparison

In this section, we ask you to evaluate (i) a Bayesian Knowledge Tracing (BKT) model, (ii) an Additive Factor Model (AFM), and (iii) a Performance Factor Analysis (PFA) model on the skills 'Circle Graph', 'Venn Diagram', and 'Mode', by performing a user-stratified 10-fold cross validation and monitoring the Root Mean Squared Error (RMSE) and the Area Under the ROC Curve (AUC) as performance metrics. Then, we ask you to visually report the RMSE and AUC scores achieved by the three student's models in the user-stratified 10-fold cross validation, in such a way that the models' performance can be easily and appropriately compared against each other.

For your convenience, you will be guided in completing this section through seven main tasks:

- Task 1.1: Group k-fold initialization.
- Task 1.2: BKT evaluation.
- Task 1.3: AFM evaluation.
- Task 1.4: PFA evaluation.
- Task 1.5: Performance metrics plotting.
- Task 1.6: Performance metrics discussion.

Task 1.1

Given that the main objective of this homework section is to evaluate three student's knowledge tracing models under a `user-stratified 10-fold cross validation`, in this task, we ask you to complete the body of a function named `create_iterator`. This function should create an iterator object able to split student's interactions included in `data` in `10 folds` such that the same student does not appear in two different folds. To do so, you can appropriately initialize a scikit-learn's GroupKFold iterator with non-overlapping groups and returning the iterator, i.e., `model_selection.GroupKFold(...).split(...)`.

For convenience, we present you an illustrative example assuming that (i) you have four data samples and that (ii) the first two data samples belong to group 0 and the last two data samples belong to group 2. The data samples associated with a group should not appear in multiple folds or, in other words, the data samples associated with a group should appear all in the same fold. Please, find below a way to use the scikit-learn's GroupKFold object to

create folds that meet this property (here, we simulate this scenario by considering only a 2-fold creation strategy):

```
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]]) y = np.array([1, 2, 3, 4]) groups = np.array([0, 0, 2, 2]) group_kfold = model_selection.GroupKFold(n_splits=2).split(X, y, groups)
```

Finally, we provide an illustrative example not related with the task on how this iterator can be then used to generate training and test folds:

```
for train_index, test_index in group_kfold:     print('TRAIN:', train_index, 'TEST:', test_index)     X_train, X_test = X[train_index], X[test_index]     y_train, y_test = y[train_index], y[test_index]     print(X_train, '-', X_test, '-', y_train, '-', y_test)
```

The above for loop generates the following output. It can be observed that the data samples belonging to a group all appear in the same fold, as expected.

```
TRAIN: [0 1] TEST: [2 3] [[1 2] [3 4]] - [[5 6] [7 8]] - [1 2] - [3 4]

TRAIN: [2 3] TEST: [0 1] [[5 6] [7 8]] - [[1 2] [3 4]] - [3 4] - [1 2]
```

Please, find more information about the GroupKFold iterator in the scikit-learn documentation.

```
def create_iterator(data):
    '''
    Create an iterator to split interactions in data in 10 folds, with the same student not appearing in two diverse folds.
    :param data:        Dataframe with student's interactions.
    :return:            An iterator.
    '''
    ### YOUR CODE HERE ###

    # Both passing a matrix with the raw data or just an array of indexes works
    X = np.arange(len(data.index))
    # Groups of interactions are identified by the user id (we do not want the same user appearing in two folds)
    groups = data['user_id'].values
    return model_selection.GroupKFold(n_splits=10).split(X, groups=groups)
```

Let's check the output of this function and a few properties of the iterator.

```
tested_user_ids = set()
for iteration, (train_index, test_index) in enumerate(create_iterator(data)):
    user_ids = data['user_id'].unique()
    train_user_ids = data.iloc[train_index]['user_id'].unique()
    test_user_ids = data.iloc[test_index]['user_id'].unique()
```

```
    print('Iteration:', iteration)
    print('Intersection between train and test user ids:',
set(train_user_ids) & set(test_user_ids))
    print('All user ids in train and test user union:',
len(set(train_user_ids).union(set(test_user_ids))) == len(user_ids))
    print('User ids tested more than once:', set(tested_user_ids) &
set(test_user_ids))
    tested_user_ids = tested_user_ids.union(set(test_user_ids))
    print()
```

```
Iteration: 0
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()

Iteration: 1
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()

Iteration: 2
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()

Iteration: 3
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()

Iteration: 4
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()

Iteration: 5
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()

Iteration: 6
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()

Iteration: 7
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()
```

```
Iteration: 8
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()

Iteration: 9
Intersection between train and test user ids: set()
All user ids in train and test user union: True
User ids tested more than once: set()
```

```
len(tested_user_ids) == len(user_ids)
```

```
True
```

On a given iteration, no user appears in both training and test sets. The union of the users in both training and test sets given us the full set of user ids in the dataset. Each user appears in the test set exactly once.

Task 1.2

In this task, we ask you to evaluate a `BKT model` with all default parameters, namely `Model(seed=0)` in pyBKT, through a `10-fold user-stratified cross-validation`, computing the following performance metrics: `RMSE` and `AUC`. To do so, you should use the `create_iterator` function, defined in Task 1.2, to create the training and test set for each fold, starting from the interactions in `data`.

No plotting is needed, it is enough to print the scores for each metric in the cell.

Please, note that this task may require a long running time (e.g., about 40 to 90 minutes), depending on your implementation. Just as an indication, on a Dell XPS 13, one fold lasts around 7 minutes.

```
### YOUR CODE HERE (please, feel free to add extra cells to solve this
task, after this first one) ###
rmse_bkt, auc_bkt = [], []
for iteration, (train_index, test_index) in
enumerate(create_iterator(data)):
    # Split data in training and test sets
    X_train, X_test = data.iloc[train_index], data.iloc[test_index]
    # Initialize and fit the model
    model = Model(seed=0)
    %time model.fit(data=X_train)
    # Compute RMSE
    train_rmse = model.evaluate(data=X_train, metric='rmse')
    test_rmse = model.evaluate(data=X_test, metric='rmse')
    rmse_bkt.append(test_rmse)
    # Compute AUC
    train_auc = model.evaluate(data=X_train, metric='auc')
```

```
    test_auc = model.evaluate(data=X_test, metric='auc')
    auc_bkt.append(test_auc)
    # Print progress
    print('Iteration:', iteration, 'RMSE', (train_rmse, test_rmse),
'AUC', (train_auc, test_auc))
```

```
Wall time: 6min 30s
Iteration: 0 RMSE (0.3619326883451476, 0.3627377393397434) AUC
(0.8768635724803968, 0.8804119587687254)
Wall time: 8min 1s
Iteration: 1 RMSE (0.3623758521598453, 0.35652947004237456) AUC
(0.8777360935826011, 0.871462831416175)
Wall time: 7min 37s
Iteration: 2 RMSE (0.3614677583161969 3, 0.3685555348264736) AUC
(0.8785462996317557, 0.8645564407924567)
Wall time: 6min 32s
Iteration: 3 RMSE (0.36123289139100484, 0.3765145626874925) AUC
(0.8785240813856332, 0.8692037292316623)
Wall time: 7min 24s
Iteration: 4 RMSE (0.3612297575354389, 0.37934386367709344) AUC
(0.8772757259188736, 0.8785882024159103)
Wall time: 6min 37s
Iteration: 5 RMSE (0.36384158637003744, 0.3501882134966003) AUC
(0.8766541779809577, 0.885324470100699)
Wall time: 7min 37s
Iteration: 6 RMSE (0.36361727299904206, 0.3499285472259435) AUC
(0.8749834945119513, 0.8995316915740698)
Wall time: 5min 59s
Iteration: 7 RMSE (0.3614092428844653, 0.3662077388390171) AUC
(0.8796324144055722, 0.8541902758970246)
Wall time: 8min 12s
Iteration: 8 RMSE (0.36266788598266114, 0.36405419887577556) AUC
(0.8765203013778682, 0.884975143324418)
Wall time: 8min 17s
Iteration: 9 RMSE (0.3633233826910702, 0.35314683335159985) AUC
(0.8788799587872089, 0.8733444928021428)
```

Finally, we show the mean and the standard deviation of the RMSE and AUC across folds.

```
'RMSE', np.mean(rmse_bkt), np.std(rmse_bkt)
```

```
('RMSE', 0.3627206702362114, 0.009824846598894346)
```

```
'AUC', np.mean(auc_bkt), np.std(auc_bkt)
```

```
('AUC', 0.8761589236323284, 0.011948209404677711)
```

Task 1.3

In this task, we ask you to evaluate an `AFM model` with all default parameters (e.g., no custom bounds, default l2 regularization, and fit_intercept=True) through a `10-fold user-stratified cross-validation`, computing the following performance metrics: RMSE and AUC. To do so, exactly as you should have done for the BKT model in Task 1.3, you should use the `create_iterator` function, defined in Task 1.2, to create the training and test set for each fold, starting from the interactions in `data`.

No plotting is needed, it is enough to print the scores for each metric in the cell.

The following cells include some utility functions that are needed to generate the `X` and `y` data in a format that is accepted by pyAFM model objects. To complete this task, you can build on top of the `X` and `y` created for you with the following cells. Please, refer to Tutorial 6 for further information on pyAFM.

```
def read_as_student_step(data):
    skills, opportunities, corrects, user_ids = [], [], [], []

    for row_id, (_, row) in enumerate(data.iterrows()):

        # Get attributes for the current interaction
        user_id = row['user_id']
        skill_name = row['skill_name']
        correct = row['correct']
        prior_success = row['prior_success']
        prior_failure = row['prior_failure']

        # Update the number of opportunities this student had with
this skill
        opportunities.append({skill_name: prior_success +
prior_failure})

        # Update information in the current
        skills.append({skill_name: 1})

        # Answer info
        corrects.append(correct)

        # Student info
        user_ids.append({user_id: 1})

    return (skills, opportunities, corrects, user_ids)

def prepare_data_afm(skills, opportunities, corrects, user_ids):

    sv = feature_extraction.DictVectorizer()
    qv = feature_extraction.DictVectorizer()
    ov = feature_extraction.DictVectorizer()
    S = sv.fit_transform(user_ids)
```

```
    Q = qv.fit_transform(skills)
    O = ov.fit_transform(opportunities)
    X = sc.sparse.hstack((S, Q, O))
    y = np.array(corrects)

    return (X.toarray(), y)
```

Prepare the X and y arrays to be used to evaluate the AFM model.

```
%time skills, opportunities, corrects, user_ids =
read_as_student_step(data)
%time X, y = prepare_data_afm(skills, opportunities, corrects,
user_ids)
```

```
Wall time: 3.25 s
Wall time: 161 ms
```

```
### YOUR CODE HERE (please, feel free to add extra cells to solve this
task, after this first one) ###
rmse_afm, auc_afm = [], []
for iteration, (train_index, test_index) in
enumerate(create_iterator(data)):
    # Split data in training and test sets
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # Initialize and fit the model
    afm = CustomLogistic()
    %time afm.fit(X_train, y_train)
    # Make predictions
    y_train_pred = afm.predict_proba(X_train)
    y_test_pred = afm.predict_proba(X_test)
    # Compute RMSE
    train_rmse = metrics.mean_squared_error(y_train, y_train_pred,
squared=False)
    test_rmse = metrics.mean_squared_error(y_test, y_test_pred,
squared=False)
    rmse_afm.append(test_rmse)
    # Compute AUC
    train_auc = metrics.roc_auc_score(y_train, y_train_pred)
    test_auc = metrics.roc_auc_score(y_test, y_test_pred)
    auc_afm.append(test_auc)
    # Print progress
    print('Iteration:', iteration, 'RMSE', (train_rmse, test_rmse),
'AUC', (train_auc, test_auc))
```

```
Wall time: 12.4 s
Iteration: 0 RMSE (0.3588795022813869, 0.45202131504519033) AUC
(0.8573893360851267, 0.6130214085464276)
Wall time: 9.38 s
Iteration: 1 RMSE (0.358820570463362, 0.4140457202925771) AUC
(0.8599132460943775, 0.6829414951498932)
```

```
Wall time: 11.4 s
Iteration: 2 RMSE (0.36066012598738856, 0.41602149060121024) AUC
(0.8557095628637055, 0.6852909560866977)
Wall time: 11.5 s
Iteration: 3 RMSE (0.35944514606534345, 0.44356724973709066) AUC
(0.8562528592307947, 0.6138551113690778)
Wall time: 12.3 s
Iteration: 4 RMSE (0.3575889560213242, 0.4341065247494938) AUC
(0.8578727813677345, 0.7078581737186332)
Wall time: 11.3 s
Iteration: 5 RMSE (0.36185077745423777, 0.3972852615444283) AUC
(0.8538340075189449, 0.7296433207291607)
Wall time: 15.1 s
Iteration: 6 RMSE (0.3592706434559734, 0.42327898547747866) AUC
(0.8566766924289391, 0.713221927205779)
Wall time: 12.7 s
Iteration: 7 RMSE (0.3579997163243513, 0.4150620859259244) AUC
(0.8608480096454952, 0.7035631340449515)
Wall time: 12.4 s
Iteration: 8 RMSE (0.35934539076985417, 0.412558505056829) AUC
(0.8565544841259949, 0.7380982429105336)
Wall time: 11.8 s
Iteration: 9 RMSE (0.36080917079239705, 0.39844331773368125) AUC
(0.8564244224956705, 0.6960964178105123)
```

Finally, we show the mean and the standard deviation of the RMSE and AUC across folds.

```
'RMSE', np.mean(rmse_afm), np.std(rmse_afm)
```

```
('RMSE', 0.4206390456163904, 0.0170239025795021)
```

```
'AUC', np.mean(auc_afm), np.std(auc_afm)
```

```
('AUC', 0.6883590187571667, 0.040906170725414366)
```

Task 1.4

In this task, we ask you to evaluate a `PFA model` with all default parameters (e.g., no custom bounds, default l2 regularization, and fit_intercept=True) through a `10-fold user-stratified cross-validation`, computing the following performance metrics: `RMSE` and `AUC`. To do so, exactly as you should have done for the BKT and AFM models in Task 1.3 and 1.4, you should use the `create_iterator` function, defined in Task 1.2, to create the training and test set for each fold, starting from the interactions in `data`.

No plotting is needed, it is enough to print the scores for each metric in the cell.

The following cells include some utility functions that are needed to generate the `X` and `y` data in a format that is accepted by pyAFM model objects. To complete this task, you can

build on top of the X and y created for you with the following cells. Please, refer to Tutorial 6 for further information on pyAFM.

```
def read_as_success_failure(data):
    n_succ, n_fail = [], []

    # Create the n_succ and n_fail variables required by pyAFM
    for i, row in data.iterrows():
        n_succ.append({row['skill_name']: int(row['prior_success'])})
        n_fail.append({row['skill_name']: int(row['prior_failure'])})

    return n_succ, n_fail

def prepare_data_pfa(skills, corrects, user_ids, n_succ, n_fail):

    s = feature_extraction.DictVectorizer()
    q = feature_extraction.DictVectorizer()
    succ = feature_extraction.DictVectorizer()
    fail = feature_extraction.DictVectorizer()
    S = s.fit_transform(user_ids)
    Q = q.fit_transform(skills)
    succ = succ.fit_transform(n_succ)
    fail = fail.fit_transform(n_fail)
    X = sc.sparse.hstack((S, Q, succ, fail))
    y = np.array(corrects)

    return (X.toarray(), y)
```

Prepare the X and y arrays to be used to evaluate the PFA model.

```
%time n_succ, n_fail = read_as_success_failure(data)
%time X, y = prepare_data_pfa(skills, corrects, user_ids, n_succ,
n_fail)
```

```
Wall time: 2.45 s
Wall time: 202 ms
```

```
### YOUR CODE HERE (please, feel free to add extra cells to solve this
task, after this first one) ###
rmse_pfa, auc_pfa = [], []
for iteration, (train_index, test_index) in
enumerate(create_iterator(data)):
    # Split data in training and test sets
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    # Initialize and fit the model
    pfa = CustomLogistic()
    %time pfa.fit(X_train, y_train)
    # Make predictions
    y_train_pred = pfa.predict_proba(X_train)
    y_test_pred = pfa.predict_proba(X_test)
```

```
    # Compute RMSE
    train_rmse = metrics.mean_squared_error(y_train, y_train_pred,
squared=False)
    test_rmse = metrics.mean_squared_error(y_test, y_test_pred,
squared=False)
    rmse_pfa.append(test_rmse)
    # Compute AUC
    train_auc = metrics.roc_auc_score(y_train, y_train_pred)
    test_auc = metrics.roc_auc_score(y_test, y_test_pred)
    auc_pfa.append(test_auc)
    # Print progress
    print('Iteration:', iteration, 'RMSE', (train_rmse, test_rmse),
'AUC', (train_auc, test_auc))
```

```
Wall time: 14.8 s
Iteration: 0 RMSE (0.3592612212713895, 0.41275331953657246) AUC
(0.8559670398622193, 0.7395491745249625)
Wall time: 18 s
Iteration: 1 RMSE (0.35885738244034426, 0.40379001344655385) AUC
(0.8588426867798364, 0.7158009486936392)
Wall time: 16.1 s
Iteration: 2 RMSE (0.3607598183660183, 0.40249078543884526) AUC
(0.8543376406280749, 0.7588350189847844)
Wall time: 15.4 s
Iteration: 3 RMSE (0.3597105765032166, 0.4269537850499162) AUC
(0.8548760790495966, 0.6866375244866865)
Wall time: 22.8 s
Iteration: 4 RMSE (0.35764536305385825, 0.4241454765292951) AUC
(0.856519188631309, 0.7512259604689446)
Wall time: 16.6 s
Iteration: 5 RMSE (0.362098709421017, 0.38602406435072795) AUC
(0.8523760148856947, 0.778039312118549)
Wall time: 19 s
Iteration: 6 RMSE (0.35953901533416016, 0.4052646257412933) AUC
(0.8551210748717755, 0.7672193517267152)
Wall time: 14.7 s
Iteration: 7 RMSE (0.3581739548411531, 0.4019915603509969) AUC
(0.8595009440354477, 0.7540208189912775)
Wall time: 17.3 s
Iteration: 8 RMSE (0.3595624198884521, 0.4019961800046421) AUC
(0.855078464451322, 0.7843115114157678)
Wall time: 16.1 s
Iteration: 9 RMSE (0.36104152124203504, 0.3871205028813061) AUC
(0.8552466760104702, 0.7484679444258453)
```

Finally, we show the mean and the standard deviation of the RMSE and AUC across folds.

```
'RMSE', np.mean(rmse_pfa), np.std(rmse_pfa)
```

```
('RMSE', 0.40525303133301493, 0.012702984340139625)
```

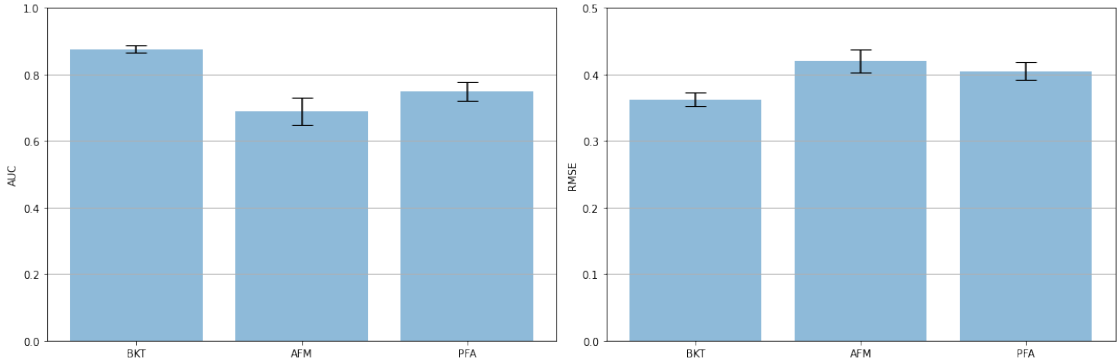
```
'AUC', np.mean(auc_pfa), np.std(auc_pfa)
```

```
('AUC', 0.7484107565837173, 0.027615946214022805)
```

Task 1.5

In this task, we ask you to visually report the RMSE and AUC scores achieved by the three student's models in the user-stratified 10-fold cross validation performed in Task 1.2, 1.3, and 1.4 respectively, in such a way that the models' performances can be easily and appropriately compared against each other.

```
### YOUR CODE HERE (please, feel free to add extra cells to solve this
task, after this first one) ###

m = {'AUC': {'BKT': auc_bkt, 'AFM': auc_afm, 'PFA': auc_pfa}, 'RMSE':
{'BKT': rmse_bkt, 'AFM': rmse_afm, 'PFA': rmse_pfa}}
limits = {'AUC': 1, 'RMSE': 0.5}

plt.figure(figsize=(15, 5))

for metric_idx, metric_key in enumerate(m.keys()):

    # Create the subplot for the current metric
    plt.subplot(1, len(m), metric_idx + 1)

    # Compute means, standard deviations, and labels
    means, errors, labels = [], [], []
    for model_key, model_scores in m[metric_key].items():
        means.append(np.mean(model_scores))
        errors.append(np.std(model_scores))
        labels.append(model_key)

    # Plot values
    x_pos = np.arange(len(labels))
    plt.bar(x_pos, means, yerr=errors, align='center', alpha=0.5,
ecolor='black', capsize=10)

    # Make decorations
    plt.grid(axis='y')
    plt.xticks(x_pos, labels)
    plt.ylabel(metric_key)
    plt.ylim(0, limits[metric_key])

# Show the plot
plt.tight_layout()
plt.show()
```



Task 1.6 Please compare and discuss the performance metric scores achieved by the student's models.

From the left plot, it can be observed that the BKT model exhibited a higher AUC score (over 0.87) and a lower standard deviation of AUC score across folds (0.01) with respect to AFM and PFA models, indicating that the predictive power of the BKT model is higher and more stable across folds than the one of the other two models, when AUC is considered. Specifically, the average BKT AUC is 0.12-point higher (0.87 vs 0.75) the average PFA AUC and 0.18-point higher (0.87 vs 0.69) than the average AFM AUC. Comparing the latter two models between each other, PFA outperforms AFM in terms of average AUC (0.75 vs 0.69, 0.06-point higher), though the difference in AUC between PFA and AFM is smaller than the difference of both with BKT. Furthermore, PFA exhibited a smaller standard deviation across folds (0.027) with respect to AFM (0.04). It can be hence observed that including information on both prior success and failure (PFA) allows the model to perform better and have a more stable performance across folds, with respect to the case where only general information on the opportunities is considered (AFM).

Similarly, the right plot highlights that the BKT model performs better, on average, with respect to AFM and PFA in terms of RMSE, i.e., the RMSE score is lower for the BKT model (0.36). AFM and PFA were able to achieve an average RMSE score of 0.42 (0.06-point higher than BKT) and 0.405 (0.045-point higher than BKT) respectively, showing again how using more fine-grained information on the failure and success of a student can lead to better predictions, but not as much as BKT. The standard deviation in performance across folds is quite small for all three models. Specifically, while being already the top performer in terms of average RMSE, the BKT model exhibited also the lowest standard deviation (0.009). AFM and PFA showed slightly higher standard deviations (0.018 and 0.12), showing that their performance were less stable across folds for the former. In general, the lower the RMSE is, the lower the standard deviation of the model performance is too. This strengthen the good performance shown by BKT, especially.

Based on our results, we can generally observe that the characteristics and properties of the BKT model allow us to make better estimations of student's skill mastery, compared to the two other considered models. One reason behind this finding might be that the skills at hand seem to meet well BKT assumptions, specifically that (i) knowledge can be divided into different skills, (ii) the definition of skills is accurate enough, (iii) each task corresponds to a single skill, and (iv) there is no connection between the skills (we should

mention that there is no "extensive" connection, though for certain skills some interrelationships exist). For AFM and PFA ones, leveraging prior failure and success opportunities (PFA) and fed them into the model leads to good performance with respect to AFM, but not as much we can with BKT. One reason behind the performance differences across models (and a step for further improvement of all of them) might lie also on the fact that we considered only a basic BKT (no forgetting or other advanced modelling) and we have not tuned certain aspects (e.g., bounds and regularizations in PFA and AFM).

2 Knowledge Tracing: Learning Curves Comparison

In this section, you should fit a Bayesian Knowledge Tracing (BKT) model on the three skills included in the `data` data set, and compute the corresponding predictions. Then, for each skill included in the data dataframe, you should visually report and discuss (i) the learning curve and (ii) the bar plot representing the number of students who reached a given number of opportunities for that skill, obtained with the BKT model fitted on the above-mentioned skills, in such a way that they can be easily and appropriately compared. No comparison with other baseline model is required.

For your convenience, you will be guided in completing this section through three main tasks:

- Task 2.1: BKT fit and prediction.
- Task 2.2: Learning curves and bar plots generation.
- Task 2.3: Learning curves and bar plots discussion.

Task 2.1

In this task, we ask you to fit a BKT model with all default parameters, i.e., `Model(seed=0)` in pyBKT, on the full `data` data set (no split into train and test set needed as we are not assessing predictive performance of the model here). Once you BKT model is fitted, we ask you to create a dataframe named `predictions` with four columns `user_id`, `skill_name`, `y_true`, `y_pred_bkt`. This dataframe should include one row per interaction in `data`, where user_id is the id of the student associated with that interaction, skill_name is the name of the skill involved in that interaction, y_true is the student's performance on that interaction (1 if correct at the first attempt, 0 otherwise), and y_pred_bkt is the prediction made by the pre-trained BKT model for that interaction.

Please, note that this task may require a long running time (e.g., about 10 to 20 minutes), depending on your implementation. Just as an indication, on a Dell XPS 13, the fit process lasts around 7 minutes.

```
### YOUR CODE HERE (please, feel free to add extra cells to solve this
task, after this first one) ###

# Initialize the model
```

```
model = Model(seed=0)

# Fit the model on the entire dataset
%time model.fit(data=data)
```

```
Wall time: 6min 40s
```

```
# Make predictions
predictions = model.predict(data=data)[['user_id', 'skill_name',
'correct', 'correct_predictions']]

# Rename the dataframe columns as per instructions
predictions.columns = ['user_id', 'skill_name', 'y_true',
'y_pred_bkt']
```

As a double check, we show the first ten rows of the predictions dataframe.

```
predictions.head()
```

	user_id	skill_name	y_true	y_pred_bkt
0	14	Circle Graph	0	0.46431
1	14	Circle Graph	1	0.33660
2	14	Circle Graph	0	0.56816
3	14	Circle Graph	0	0.44325
4	14	Circle Graph	0	0.31982

Task 2.2

In this task, for each skill, we ask you to visually report and discuss (i) the `learning curve` and (ii) the `bar plot` representing the number of students who reached a given number of opportunities (similar to the visualizations done in Tutorial 6), obtained by the BKT model fitted on that skill, in such a way that they can be easily and appropriately compared. To do so, we ask you to use the predictions you stored in the dataframe `predictions`.

No comparison with other baseline model is required.

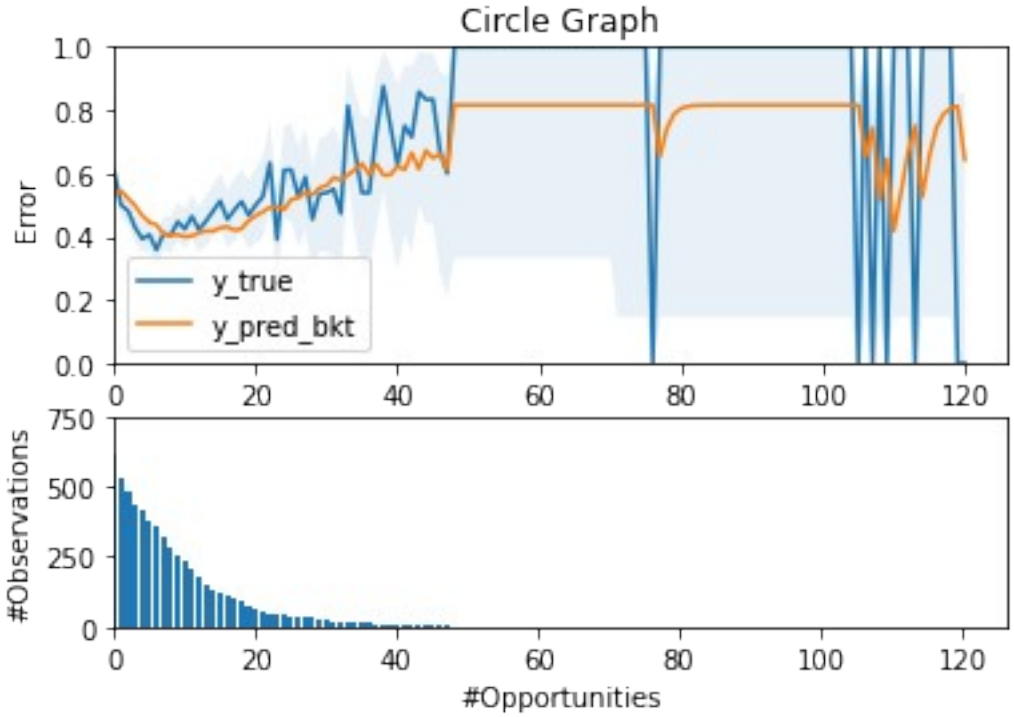
Please, refer to Tutorial 6 for further information on learning curve and bar plotting for student's knowledge tracing models.

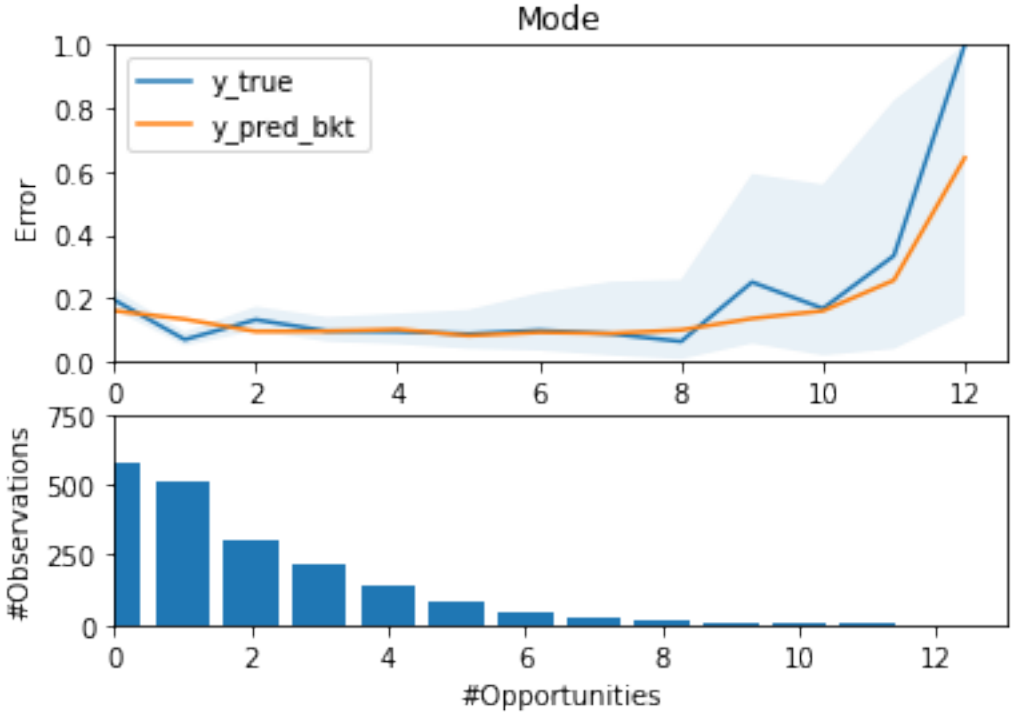
```
### YOUR CODE HERE (please, feel free to add extra cells to solve this
task, after this first one) ###
def avg_y_by_x(x, y):
    # Transform lists into arrays
    x = np.array(x)
    y = np.array(y)

    # Sort the integer id representing the number of opportunities in
increasing order
```

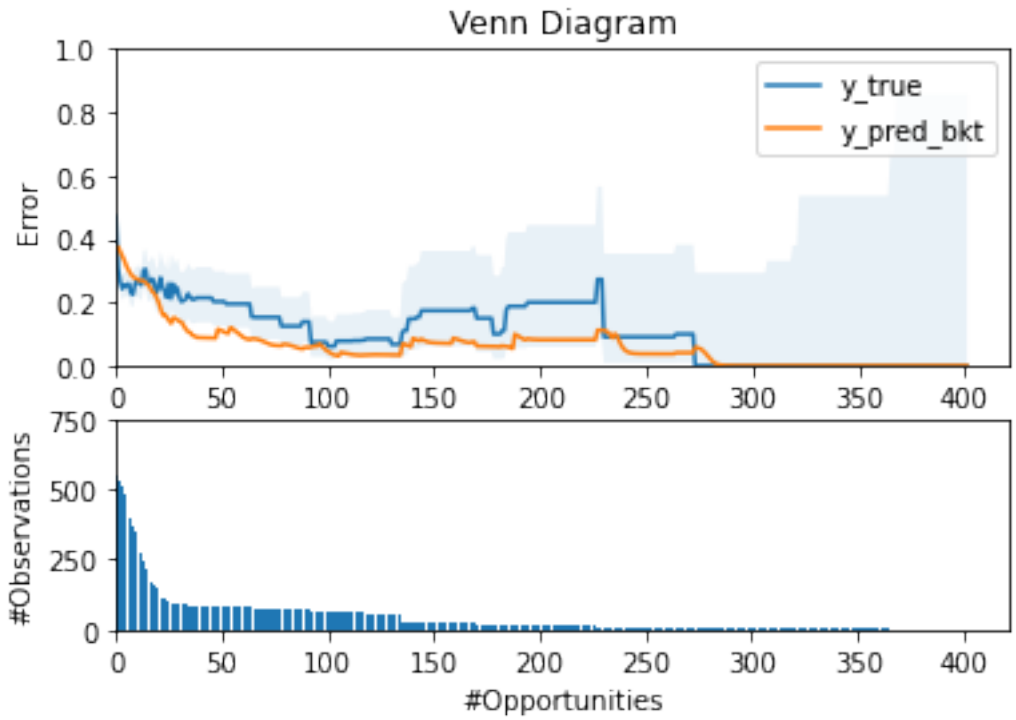
```
    xs = sorted(list(set(x)))

    # Supporting lists to store the:
    # - xv: integer identifier of the number of opportunities
    # - yv: average value across students at that number of
opportunities
    # - lcb and ucb: lower and upper confidence bound
    # - n_obs: number of observartions present at that number of
opportunities (on per-skill plots, it is the #students)
    xv, yv, lcb, ucb, n_obs = [], [], [], [], []

    # For each integer identifier of the number of opportunities
0, ...
    for v in xs:
        ys = [y[i] for i, e in enumerate(x) if e == v] # We retrieve
the values for that integer identifier
        if len(ys) > 0:
            xv.append(v) # Append the integer identifier of the number
of opportunities
            yv.append(sum(ys) / len(ys)) # Append the average value
across students at that number of opportunities
            n_obs.append(len(ys)) # Append the number of observartions
present at that number of opportunities


            # Prepare data for confidence interval computation
            unique, counts = np.unique(ys, return_counts=True)
            counts = dict(zip(unique, counts))

            if 0 not in counts:
                counts[0] = 0
            if 1 not in counts:
                counts[1] = 0

            # Calculate the 95% confidence intervals
            ci = sc.stats.beta.interval(0.95, 0.5 + counts[0], 0.5 +
counts[1])
            lcb.append(ci[0])
            ucb.append(ci[1])

    return xv, yv, lcb, ucb, n_obs

### YOUR CODE HERE (please, feel free to add extra cells to solve this
task, after this first one) ###

for plot_id, skill_name in enumerate(data['skill_name'].unique()): #
For each skill under consideration

    preds = predictions[predictions['skill_name'] == skill_name] #
Retrieve predictions for the current skill
```

```
    xp = []
    yp = {}
    for col in preds.columns: # For y_true and and y_pred_bkt columns,
initialize an empty list for curve values
        if 'y_' in col:
            yp[col] = []

    for user_id in preds['user_id'].unique(): # For each user
        user_preds = preds[preds['user_id'] == user_id] # Retrieve the
predictions on the current skill for this user
        xp += list(np.arange(len(user_preds))) # The x-axis values go
from 0 to |n_opportunities|-1
        for col in preds.columns:
            if 'y_' in col: # For y_true and and y_pred_bkt columns
                yp[col] += user_preds[col].tolist() # The y-axis value
is the success rate for this user at that opportunity

    fig, axs = plt.subplots(2, 1, gridspec_kw={'height_ratios': [3,
2]}) # Initialize the plotting figure

    lines = []
    for col in preds.columns:
        if 'y_' in col: # For y_true and and y_pred_bkt columns
            x, y, lcb, ucb, n_obs = avg_y_by_x(xp, yp[col]) #
Calculate mean and 95% confidence intervals for success rate
            y = [1-v for v in y] # Transform success rate in error
rate
            if col == 'y_true': # In case of ground-truth data, we
also show the confidence intervals
                axs[0].fill_between(x, lcb, ucb, alpha=.1)
            model_line, = axs[0].plot(x, y, label=col) # Plot the
curve
            lines.append(model_line) # Store the line to then set the
legend

    # Make decorations for the learning curve plot
    axs[0].set_title(skill_name)
    axs[0].legend(handles=lines)
    axs[0].set_ylabel('Error')
    axs[0].set_ylim(0, 1)
    axs[0].set_xlim(0, None)

    # Plot the number of observations per number of opportunities bars
and make decorations
    axs[1].set_xlabel('#Opportunities')
    axs[1].bar([i for i in range(len(n_obs))], n_obs)
    axs[1].set_ylabel('#Observations')
    axs[1].set_ylim(0, 750)
    axs[1].set_xlim(0, None)
```

```
# Plot the learning curve and the bar plot
plt.show()
```







Task 2.3 Please discuss all visualizations (learning curves and bar plots) obtained with the BKT model.

First, we discuss each skill separately. Then, given that we are showing the same aspects for the three skills, we provide few more observations aimed at comparing the findings we obtained across the skills under consideration.

- "**Circle Graph**". From the error rate in the ground-truth data (y_true, blue), it can be observed that this skill appears quite hard for students, with an initial error rate of around 0.60 in the first opportunity. The error rate goes slightly down, as we would usually expect, as much as the students play with the skill, reaching the lowest error rate of 0.38 at around 7 opportunities. However, starting from #opportunities around equal to 7, the error rate starts going up till 0.80 after 50 opportunities. It seems that a large part of the students managed to master a bit this skill in the first opportunities, but then they start facing difficulties in mastering it. This behavior might be due to the fact that the skill is ill-defined or that the problems become too difficult or are not aligned well with the skill. After 50 opportunities, the error rate starts jumping between 0 and a 1 just because there are only few students. In terms of confidence interval (blue area), it can be observed that the error rate is quite stable at the earlier stage. Starting from around 7 opportunities, the error rate is less stable probably due to the same reasons we provided above. The confidence interval finally becomes large when only few students keep playing with this skill for a higher number of opportunities. When it comes to consider the error rate obtained by the BKT model estimations (y_pred_bkt, orange), it can be observed that the model tends to overestimate a bit the error rate during the first opportunities (the orange line is above the blue one, the model tends to predict more errors than

expected), while the opposite pattern is observed for higher numbers of opportunities (the model tends to predict less errors than expected). However, in general, the model fits well the ground-truth data. Looking at the bar plot at the bottom, students appear reasonably engaged with this skill only for few opportunities (particularly, starting from 35 opportunities, the number of involved students is low, and the error rate starts jumping more).

- "**Mode**". Based on the patterns of the ground-truth data (y_true, blue), the error rate observed for this skill at the earlier stages is of around 0.20. The students thus are already good in problems involving this skill, somehow. While this might depend on the intrinsic knowledge/background of the students about this skill, this behavior might also appear due to the fact that the skill is easier and easier to master in general or that the problems presented to the students on that skill are too easy, for instance. While the error rate goes slightly down when the number of opportunities increases, the difference between the initial error rate and the error rate experienced (for instance) after 5 or 6 opportunities is relatively small (around 0.10 of error rate in the latter case). This observation might be justified by the fact that students played with the skill only for few opportunities (so there is no so much room for improving more) and by the already low initial error rate (students might use those opportunities just to refine their knowledge, with few errors still happening). The going-up behavior at the end appears mainly due to the very low number of students involved at that point. The blue area shows us that the confidence interval is very small for the first three or four opportunities (so behavior is more consistent among students), while it starts increasing later probably also due to the peculiar patterns behind students who interact more with this skill. The BKT model (y_pred_bkt, orange) is able to fit the ground-truth data very well across the number of opportunities, especially till around 8 opportunities. One important observation comes from the bar plot, which shows that the number of students involved in this skill drastically goes down especially after three/four opportunities.

- "**Venn Diagram**". This third skill appears like another somehow easy skill since the beginning (y_true, blue), but a bit more harder than "Mode". Except for a few fluctuations in the very first opportunities (where the curve slightly increases, maybe because they are just the first opportunities or due to the type/difficulty of problems proposed at that point), the error rate goes down till around 0.08, reached at around 100 opportunities. Probably, once only those students struggling more with this skill tend to stay for a higher number of opportunities, the error rate goes up again. The confidence interval for the ground-truth data (blue area) is relatively high (if compared with the one observed for the other two skills during the first opportunities). Naturally, it increases even more when the number of students reaching that number of opportunities is very low (#opportunities > 130). Comparing the ground-truth estimations with the BKT estimations (y_pred_bkt, orange), except for the first opportunities, it can be observed that BKT tends to consistently underestimate the error rate and the BKT error rate is often outside the confidence interval of actual data. The number of students per opportunity naturally

goes down as observed for the other skills, but interestingly it can be observed that a set of students seems to still keep playing with this skill from around 30 opportunites till 100 opportunities stably (the bar plot values appear somehow similar in that range).

Overall, comparing the considered three skills, "Circle Graph" seems to be really going up (e.g., ill-defined skill or due to the type/difficulty of problems), while, for "Mode" and "Venn Diagram", the going up later seems to come only from the very low number of students. Especially but not only for the "Venn Diagram" skill, the BKT generally tends to under-estimate the error rate. One reason behind this observation could be that the BKT model we adopted did not consider forgetting, for instance. In terms of number of opportunities, students experienced a lower maximum number of opportunities on the "Mode" skill, compared to the others. Compared to "Circle Graph", the skill "Venn Diagram" seems to involve lots of students even for higher number of opportunities (e.g., #opportunities > 30).

Lab 8 Solution - DKT Model Comparison

In this exercises, you will compare the performance of different knowledge tracing models. We will use the same ASSISTments data set as for lecture 7.

The ASSISTments data sets are often used for benchmarking knowledge tracing models. We will play with a simplified data set that contains the following columns:

Name	Description
user_id	The ID of the student who is solving the problem.
order_id	The temporal ID (timestamp) associated with the student's answer to the problem.
skill_name	The name of the skill associated with the problem.
correct	The student's performance on the problem: 1 if the problem's answer is correct at the first attempt, 0 otherwise.

Note that this notebook will need to use the tensorflow kernel. Change the kernel in the upper right corner of Noto. Select `tensorflow`.

We first load the data set.

```
# Principal package imports
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import scipy as sc

# Scikit-learn package imports
from sklearn import feature_extraction, model_selection
from sklearn.metrics import mean_squared_error, roc_auc_score

# PyBKT package imports
from pyBKT.models import Model
# Import the lmm model class
from pymer4.models import Lmer

# Tensorflow
import tensorflow as tf

DATA_DIR = "./../../data/"

assistments = pd.read_csv(DATA_DIR + 'assistments.csv',
low_memory=False).dropna()
assistments.head()
```

```
   user_id  order_id       skill_name  correct
0    64525  33022537  Box and Whisker        1
1    64525  33022709  Box and Whisker        1
2    70363  35450204  Box and Whisker        0
3    70363  35450295  Box and Whisker        1
4    70363  35450311  Box and Whisker        0
```

Next, we print the number of unique students and skills in this data set.

```
print("Number of unique students in the dataset:",
len(set(assistments['user_id'])))
print("Number of unique skills in the dataset:",
len(set(assistments['skill_name'])))
```

```
Number of unique students in the dataset: 4151
Number of unique skills in the dataset: 110
```

We also implement a utility function that splits the data in two folds, making sure that all interactions of a student land in the same fold. We will use this function later when comparing predictive performance of the different models.

```
def create_iterator(data):
    '''
    Create an iterator to split interactions in data into train and
test, with the same student not appearing in two diverse folds.
    :param data:        Dataframe with student's interactions.
    :return:            An iterator.
    '''
    # Both passing a matrix with the raw data or just an array of
indexes works
    X = np.arange(len(data.index))
    # Groups of interactions are identified by the user id (we do not
want the same user appearing in two folds)
    groups = data['user_id'].values
    return model_selection.GroupShuffleSplit(n_splits=1,
train_size=.8, test_size=0.2, random_state=0).split(X, groups=groups)
```

Additive Factors Model (AFM) and Performance Factors Analysis (PFA)

The AFM and PFA models are both based on logistic regression and item response theory (IRT). Specifically, they compute the probability that a student will solve a task correctly based on the number of previous attempts the student had at the corresponding skill (in case of AFM) and based on the correct and wrong attempts at the corresponding skill (in case of PFA), respectively. We therefore first preprocess the data to compute these variables. For demonstration purposes, we will continue on the small subset of the data set containing six skills.

```
skills_subset = ['Circle Graph', 'Venn Diagram', 'Mode', 'Division
Fractions', 'Finding Percents', 'Area Rectangle']
data = assistments[assistments['skill_name'].isin(skills_subset)]
```

```
print("Skill set:", set(data['skill_name']))
print("Number of unique students in the subset:",
len(set(data['user_id'])))
print("Number of unique skills in the subset:",
len(set(data['skill_name'])))
```

```
Skill set: {'Finding Percents', 'Venn Diagram', 'Area Rectangle',
'Mode', 'Circle Graph', 'Division Fractions'}
Number of unique students in the subset: 1527
Number of unique skills in the subset: 6
```

```
# Data processing
# Number of attempts before current
def preprocess_data(data):
    data.loc[:, 'aux'] = 1
    data.loc[:, 'prev_attempts'] =
data.sort_values('order_id').groupby(['user_id', 'skill_name'])
['aux'].cumsum() -1

    # Number of correct and incorrect attempts before current attempt
    data.loc[:, 'correct_aux'] =
data.sort_values('order_id').groupby(['user_id', 'skill_name'])
['correct'].cumsum()
    data.loc[:, 'before_correct_num'] =
data.sort_values('order_id').groupby(['user_id', 'skill_name'])
['correct_aux'].shift(periods=1, fill_value=0)
    data.loc[:, 'before_wrong_num'] = data['prev_attempts'] -
data['before_correct_num']
    return data

data = preprocess_data(data)
data.head()
```

```
/tmp/ipykernel_857/3193894825.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data.loc[:, 'aux'] = 1
/tmp/ipykernel_857/3193894825.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data.loc[:, 'prev_attempts'] =
data.sort_values('order_id').groupby(['user_id', 'skill_name'])
```

```
['aux'].cumsum() -1
/tmp/ipykernel_857/3193894825.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data.loc[:, 'correct_aux'] =
data.sort_values('order_id').groupby(['user_id', 'skill_name'])
['correct'].cumsum()
/tmp/ipykernel_857/3193894825.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data.loc[:, 'before_correct_num'] =
data.sort_values('order_id').groupby(['user_id', 'skill_name'])
['correct_aux'].shift(periods=1, fill_value=0)
/tmp/ipykernel_857/3193894825.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data.loc[:, 'before_wrong_num'] = data['prev_attempts'] -
data['before_correct_num']
```

	user_id	order_id	skill_name	correct	aux	prev_attempts	\
3957	14	21617623	Circle Graph	0	1	0	
3958	14	21617632	Circle Graph	1	1	1	
3959	14	21617641	Circle Graph	0	1	2	
3960	14	21617650	Circle Graph	0	1	3	
3961	14	21617659	Circle Graph	0	1	4	

	correct_aux	before_correct_num	before_wrong_num
3957	0	0	0
3958	1	0	1
3959	1	1	1
3960	1	1	2
3961	1	1	3

Next, we split the data into a training and a test data set.

```
# Obtain indexes
train_index, test_index = next(create_iterator(data))
```

```
# Split the data
X_train, X_test = data.iloc[train_index], data.iloc[test_index]
```

Next, we fit an AFM model to the training data and predict on the test data. Note that the implementation below only works for a one-to-one correspondance of task and skill, i.e. when a task is associated to exactly one skill. In case of a data set containing tasks with multiple skills, we would need to use the pyAFM package. A tutorial on using pyAFM can be found here.

```
# Initialize and fit the model
model = Lmer("correct ~ (1|user_id) + (1|skill_name) + (0 +
prev_attempts|skill_name)", data=X_train, family='binomial')
%time model.fit()
# Compute predictions
X_test['afm_predictions'] = model.predict(data=X_test,
verify_predictions=False)
X_test.head()
```

```
Formula: correct~(1|user_id)+(1|skill_name)+(0+prev_attempts|
skill_name)

Family: binomial  Inference: parametric

Number of observations: 40258    Groups: {'user_id': 1221.0,
'skill_name': 6.0}

Log-likelihood: -16797.782  AIC: 33603.565

Random effects:

                      Name     Var     Std
user_id         (Intercept) 2.56000 1.60000
skill_name      (Intercept) 0.68300 0.82700
skill_name.1  prev_attempts 0.00500 0.06900

No random effect correlations specified

Fixed effects:

CPU times: user 24.6 s, sys: 80 ms, total: 24.7 s
Wall time: 24.7 s

/tmp/ipykernel_857/736728277.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
```

```
  X_test['afm_predictions'] = model.predict(data=X_test,
verify_predictions=False)
```

```
      user_id  order_id    skill_name  correct  aux  prev_attempts  \
3969    64525  28186893  Circle Graph        1    1              0
3970    64525  28187093  Circle Graph        1    1              1
3971    64525  32413158  Circle Graph        1    1              2
3972    64525  33022751  Circle Graph        0    1              3
3973    64525  33023039  Circle Graph        1    1              4

      correct_aux  before_correct_num  before_wrong_num
afm_predictions
3969            1                   0                 0
0.48266
3970            2                   1                 0
0.49251
3971            3                   2                 0
0.50236
3972            3                   3                 0
0.51221
3973            4                   3                 1
0.52205
```

Next, we fit a PFA model to the data. Again, this implementation works for one-to-one correspondance and tasks with multiple skills would require the use of pyAFM.

```
# Initialize and fit the model
model = Lmer("correct ~ (1|user_id) + (1|skill_name) + (0 +
before_correct_num|skill_name) + (0 + before_wrong_num|skill_name)",
data=X_train, family='binomial')
%time model.fit()
# Compute predictions
X_test['pfa_predictions'] = model.predict(data=X_test,
verify_predictions=False)
X_test.head()
```

```
Formula: correct~(1|user_id)+(1|skill_name)+(0+before_correct_num|
skill_name)+(0+before_wrong_num|skill_name)

Family: binomial  Inference: parametric

Number of observations: 40258    Groups: {'user_id': 1221.0,
'skill_name': 6.0}

Log-likelihood: -16385.969  AIC: 32781.939

Random effects:

                          Name     Var     Std
user_id             (Intercept) 1.74800 1.32200
```

```
skill_name           (Intercept) 0.69900 0.83600
skill_name.1  before_correct_num 0.02600 0.16200
skill_name.2    before_wrong_num 0.00000 0.01000

No random effect correlations specified

Fixed effects:

CPU times: user 1min 10s, sys: 208 ms, total: 1min 10s
Wall time: 1min 10s

/tmp/ipykernel_857/1100232259.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  X_test['pfa_predictions'] = model.predict(data=X_test,
verify_predictions=False)

      user_id  order_id    skill_name  correct  aux  prev_attempts  \
3969    64525  28186893  Circle Graph        1    1              0
3970    64525  28187093  Circle Graph        1    1              1
3971    64525  32413158  Circle Graph        1    1              2
3972    64525  33022751  Circle Graph        0    1              3
3973    64525  33023039  Circle Graph        1    1              4

      correct_aux  before_correct_num  before_wrong_num
afm_predictions  \
3969            1                   0                 0
0.48266
3970            2                   1                 0
0.49251
3971            3                   2                 0
0.50236
3972            3                   3                 0
0.51221
3973            4                   3                 1
0.52205

      pfa_predictions
3969          0.46224
3970          0.48999
3971          0.51780
3972          0.54551
3973          0.54598
```

Deep Knowledge Tracing (DKT)

Knowledge tracing is one of the key research areas for empowering personalized education. It is a task to model students' mastery level of a skill based on their historical learning trajectories. In recent years, a recurrent neural network model called deep knowledge tracing (DKT) has been proposed to handle the knowledge tracing task and literature has shown that DKT generally outperforms traditional methods.

Next, we will create and evaluate DKT models on top of a TensorFlow framework. For those who are not familiar with this framework, we recommended to follow the official tutorials.

We continue to work with the small subset (six skills of the data). Furthermore, we will continue to use the same train test split as before.

Data preparation

A DKT model is characterized by the following main three components:

- **Input**: the one-hot encoded observations at varying time steps.
- **Network**: a recurrent neural network that processes the one-hot encoded observations in a time-wise manner.
- **Output**: the probabilities for answering skill (or item) correct at the varying time steps.

The first step to enable a DKT experimental pipeline requires to prepare the input and output data to be fed into the model during the training and evaluation phases. TensorFlow has an API, called TF Dataset, that supports writing descriptive and efficient input pipelines. Dataset usage follows a common pattern: (i) create a source dataset from your input data, (ii) apply dataset transformations to preprocess the data, (iii) iterate over the dataset and process the elements. Iteration happens in a streaming fashion, so the full dataset does not need to fit into memory.

```
def prepare_seq(df):
    # Step 1 - Enumerate skill id
    df['skill'], skill_codes = pd.factorize(df['skill_name'],
sort=True)

    # Step 2 - Cross skill id with answer to form a synthetic feature
    df['skill_with_answer'] = df['skill'] * 2 + df['correct']

    # Step 3 - Convert to a sequence per user id and shift features 1
timestep
    seq = df.groupby('user_id').apply(lambda r:
(r['skill_with_answer'].values[:-1], r['skill'].values[1:],
r['correct'].values[1:],))

    # Step 4- Get max skill depth and max feature depth
    skill_depth = df['skill'].max()
    features_depth = df['skill_with_answer'].max() + 1
```

```
    return seq, features_depth, skill_depth

def prepare_data(seq, params, features_depth, skill_depth):

    # Step 1 - Get Tensorflow Dataset
    dataset = tf.data.Dataset.from_generator(generator=lambda: seq,
output_types=(tf.int32, tf.int32, tf.float32))

    # Step 2 - Encode categorical features and merge skills with
labels to compute target loss.
    dataset = dataset.map(
        lambda feat, skill, label: (
            tf.one_hot(feat, depth=features_depth),
            tf.concat(values=[tf.one_hot(skill, depth=skill_depth),
tf.expand_dims(label, -1)], axis=-1)
        )
    )

    # Step 3 - Pad sequences per batch
    dataset = dataset.padded_batch(
        batch_size=params['batch_size'],
        padding_values=(params['mask_value'], params['mask_value']),
        padded_shapes=([None, None], [None, None]),
        drop_remainder=True
    )

    return dataset.repeat(), len(seq)
```

The data needs to be fed into the model in batches. Therefore, we need to specify in advance how many elements per batch our DKT will receive. Furthermore, all sequences should be of the same length in order to be fed into the model. Given that students have different number of opportunities across skills, we need to define a masking value for those entries that are introduced as a padding into the student's sequences.

```
params = {}
params['batch_size'] = 32
params['mask_value'] = -1.0
```

We are now ready to encode the data and split into a training, validation, and test set.

```
# Obtain indexes for necessary validation set
train_val_index, val_index = next(create_iterator(X_train))
# Split the training data into training and validation
X_train_val, X_val = X_train.iloc[train_val_index],
X_train.iloc[val_index]

seq, features_depth, skill_depth = prepare_seq(data)
seq_train = seq[X_train.user_id.unique()]
seq_val = seq[X_train_val.user_id.unique()]
```

```
seq_test = seq[X_test.user_id.unique()]

tf_train, length = prepare_data(seq_train, params, features_depth,
skill_depth)
tf_val, val_length  = prepare_data(seq_val, params, features_depth,
skill_depth)
tf_test, test_length = prepare_data(seq_test, params, features_depth,
skill_depth)

params['train_size'] = int(length // params['batch_size'])
params['val_size'] = int(val_length // params['batch_size'])
params['test_size'] = int(test_length // params['batch_size'])
```

```
/tmp/ipykernel_857/2886616435.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['skill'], skill_codes = pd.factorize(df['skill_name'], sort=True)
/tmp/ipykernel_857/2886616435.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df['skill_with_answer'] = df['skill'] * 2 + df['correct']
2023-04-14 13:32:18.681226: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot
open shared object file: No such file or directory
2023-04-14 13:32:18.681342: W
tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to
cuInit: UNKNOWN ERROR (303)
2023-04-14 13:32:18.681445: I
tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver
does not appear to be running on this host (noto.epfl.ch):
/proc/driver/nvidia/version does not exist
2023-04-14 13:32:18.682747: I
tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow
binary is optimized with oneAPI Deep Neural Network Library (oneDNN)
to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
```

Model Creation

Next, we create and compile the model. To do so, we first define the necessary parameters.

```
params['verbose'] = 1 # Verbose = {0,1,2}
params['best_model_weights'] = 'weights/bestmodel' # File to save the
model
params['optimizer'] = 'adam' # Optimizer to use
params['backbone_nn'] = tf.keras.layers.RNN # Backbone neural network
params['recurrent_units'] = 16 # Number of RNN units
params['epochs'] = 10  # Number of epochs to train
params['dropout_rate'] = 0.3 # Dropout rate
```

Considering that we padded the sequences such that all have the same length, we need to remove predictions on the time step associated with padding. We also need to mach each output with a specific skill. To this end, we implement a function calle get_target.

```
def get_target(y_true, y_pred, mask_value=params['mask_value']):

    # Get skills and labels from y_true
    mask = 1. - tf.cast(tf.equal(y_true, mask_value), y_true.dtype)
    y_true = y_true * mask

    skills, y_true = tf.split(y_true, num_or_size_splits=[-1, 1],
axis=-1)

    # Get predictions for each skill
    y_pred = tf.reduce_sum(y_pred * skills, axis=-1, keepdims=True)

    return y_true, y_pred
```

While training the model, we will monitor the following evaluation metrics.

```
class AUC(tf.keras.metrics.AUC):
    def update_state(self, y_true, y_pred, sample_weight=None):
        true, pred = get_target(y_true, y_pred)
        super(AUC, self).update_state(y_true=true, y_pred=pred,
sample_weight=sample_weight)

class RMSE(tf.keras.metrics.RootMeanSquaredError):
    def update_state(self, y_true, y_pred, sample_weight=None):
        true, pred = get_target(y_true, y_pred)
        super(RMSE, self).update_state(y_true=true, y_pred=pred,
sample_weight=sample_weight)

def CustomBinaryCrossEntropy(y_true, y_pred):
    y_true, y_pred = get_target(y_true, y_pred)
    return tf.keras.losses.binary_crossentropy(y_true, y_pred)
```

We are now ready to create the model.

```
def create_model(nb_features, nb_skills, params):

    # Create the model architecture
    inputs = tf.keras.Input(shape=(None, nb_features), name='inputs')
    x = tf.keras.layers.Masking(mask_value=params['mask_value'])
(inputs)
    x = tf.keras.layers.LSTM(params['recurrent_units'],
return_sequences=True, dropout=params['dropout_rate'])(x)
    dense = tf.keras.layers.Dense(nb_skills, activation='sigmoid')
    outputs = tf.keras.layers.TimeDistributed(dense, name='outputs')
(x)
    model = tf.keras.models.Model(inputs=inputs, outputs=outputs,
name='DKT')

    # Compile the model
    model.compile(loss=CustomBinaryCrossEntropy,
                  optimizer=params['optimizer'],
                  metrics=[AUC(), RMSE()])

    return model

model = create_model(features_depth, skill_depth, params)

model.summary()
```

```
Model: "DKT"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 inputs (InputLayer)          [(None, None, 12)]        0

 masking (Masking)            (None, None, 12)          0

 lstm (LSTM)                  (None, None, 16)          1856

 outputs (TimeDistributed)    (None, None, 5)           85

=================================================================
Total params: 1,941
Trainable params: 1,941
Non-trainable params: 0
_________________________________________________________________
```

Model Fitting and Evaluation

Finally, we fit the model on the training data and evaluate it on the test data. We are using a callback for the model, i.e. we store the best model (on the validation set) and then use this model for prediction.

```
ckp_callback =
tf.keras.callbacks.ModelCheckpoint(params['best_model_weights'],
```

```
save_best_only=True, save_weights_only=True)
history = model.fit(tf_train, epochs=params['epochs'],
steps_per_epoch=params['train_size'],
                    validation_data=tf_val,  validation_steps =
params['val_size'],
                    callbacks=[ckp_callback],
verbose=params['verbose'])
```

```
Epoch 1/10
38/38 [==============================] - 13s 219ms/step - loss: 0.8645
- auc: 0.5299 - root_mean_squared_error: 0.6772 - val_loss: 0.8615 -
val_auc: 0.5734 - val_root_mean_squared_error: 0.6777
Epoch 2/10
38/38 [==============================] - 8s 212ms/step - loss: 0.8592
- auc: 0.5792 - root_mean_squared_error: 0.6667 - val_loss: 0.8556 -
val_auc: 0.5861 - val_root_mean_squared_error: 0.6668
Epoch 3/10
38/38 [==============================] - 7s 174ms/step - loss: 0.8537
- auc: 0.5893 - root_mean_squared_error: 0.6551 - val_loss: 0.8489 -
val_auc: 0.5883 - val_root_mean_squared_error: 0.6549
Epoch 4/10
38/38 [==============================] - 6s 166ms/step - loss: 0.8473
- auc: 0.5919 - root_mean_squared_error: 0.6434 - val_loss: 0.8426 -
val_auc: 0.5862 - val_root_mean_squared_error: 0.6453
Epoch 5/10
38/38 [==============================] - 7s 183ms/step - loss: 0.8432
- auc: 0.5906 - root_mean_squared_error: 0.6345 - val_loss: 0.8379 -
val_auc: 0.5863 - val_root_mean_squared_error: 0.6389
Epoch 6/10
38/38 [==============================] - 7s 174ms/step - loss: 0.8390
- auc: 0.5918 - root_mean_squared_error: 0.6286 - val_loss: 0.8344 -
val_auc: 0.5876 - val_root_mean_squared_error: 0.6352
Epoch 7/10
38/38 [==============================] - 6s 169ms/step - loss: 0.8356
- auc: 0.5934 - root_mean_squared_error: 0.6251 - val_loss: 0.8323 -
val_auc: 0.5873 - val_root_mean_squared_error: 0.6338
Epoch 8/10
38/38 [==============================] - 6s 165ms/step - loss: 0.8348
- auc: 0.5925 - root_mean_squared_error: 0.6242 - val_loss: 0.8305 -
val_auc: 0.5890 - val_root_mean_squared_error: 0.6322
Epoch 9/10
38/38 [==============================] - 7s 188ms/step - loss: 0.8341
- auc: 0.5942 - root_mean_squared_error: 0.6228 - val_loss: 0.8284 -
val_auc: 0.5909 - val_root_mean_squared_error: 0.6307
Epoch 10/10
38/38 [==============================] - 7s 180ms/step - loss: 0.8319
- auc: 0.5956 - root_mean_squared_error: 0.6215 - val_loss: 0.8271 -
val_auc: 0.5908 - val_root_mean_squared_error: 0.6302
```

We evaluate on the test data set and print the results.

```
model.load_weights(params['best_model_weights'])
metrics_dkt_small = model.evaluate(tf_test, verbose=params['verbose'],
steps = params['test_size'])
```

```
9/9 [==============================] - 1s 46ms/step - loss: 0.6262 -
auc: 0.6545 - root_mean_squared_error: 0.5911
```

```
# Binary cross entropy, AUC, RMSE
metrics_dkt_small
```

```
[0.6262012124061584, 0.6544973850250244, 0.5910719037055969]
```

BKT

We first also fit a BKT model to this data set using the same train/test split as above.

```
df_preds = pd.DataFrame()

# Train a BKT model for each skill
for skill in skills_subset:
    print("--{}--".format(skill))
    X_train_skill = X_train[X_train['skill_name'] == skill]
    X_test_skill = X_test[X_test['skill_name'] == skill]
    # Initialize and fit the model
    model = Model(seed=0)
    %time model.fit(data=X_train_skill)
    preds = model.predict(data=X_test_skill) [['user_id', 'order_id',
'skill_name', 'correct', 'prev_attempts',
       'before_correct_num', 'before_wrong_num', 'afm_predictions',
'pfa_predictions', 'correct_predictions']]
    df_preds = df_preds.append(preds)

X_test = df_preds
X_test.columns = ['user_id', 'order_id', 'skill_name', 'correct',
'prev_attempts',
       'before_correct_num', 'before_wrong_num', 'afm_predictions',
'pfa_predictions', 'bkt_predictions']
X_test.head()
```

```
--Circle Graph--
CPU times: user 6.39 s, sys: 0 ns, total: 6.39 s
Wall time: 6.78 s
--Venn Diagram--
CPU times: user 5 s, sys: 0 ns, total: 5 s
Wall time: 5.38 s
--Mode--
CPU times: user 8.29 s, sys: 0 ns, total: 8.29 s
Wall time: 8.69 s
--Division Fractions--
CPU times: user 6.7 s, sys: 0 ns, total: 6.7 s
Wall time: 6.98 s
```

```
--Finding Percents--
CPU times: user 7 s, sys: 0 ns, total: 7 s
Wall time: 7.48 s
--Area Rectangle--
CPU times: user 3.68 s, sys: 0 ns, total: 3.68 s
Wall time: 3.88 s
```

	user_id	order_id	skill_name	correct	prev_attempts	before_correct_num	before_wrong_num	afm_predictions	pfa_predictions	bkt_predictions
3969	64525	28186893	Circle Graph	1	0	0	0	0.48266	0.46224	0.44987
3970	64525	28187093	Circle Graph	1	1	1	0	0.49251	0.48999	0.63161
3971	64525	32413158	Circle Graph	1	2	2	0	0.50236	0.51780	0.68926
3972	64525	33022751	Circle Graph	0	3	3	0	0.51221	0.54551	0.70119
3973	64525	33023039	Circle Graph	1	4	3	1	0.52205	0.54598	0.69646

```
X_test.to_csv(DATA_DIR + 'x_test_08.csv.gz', compression = 'gzip',
index = False)
```

Your Turn 1 - Model Comparison on Subset

Up to now, we have compared model performance on a subset of the data. Your task is to compare and discuss performance of the different models:

1. Visualize the overall RMSE and AUC of the four models (AFM, PFA, BKT, DKT) such that the metrics can be easily compared.
2. Interpret your results and discuss your observations.

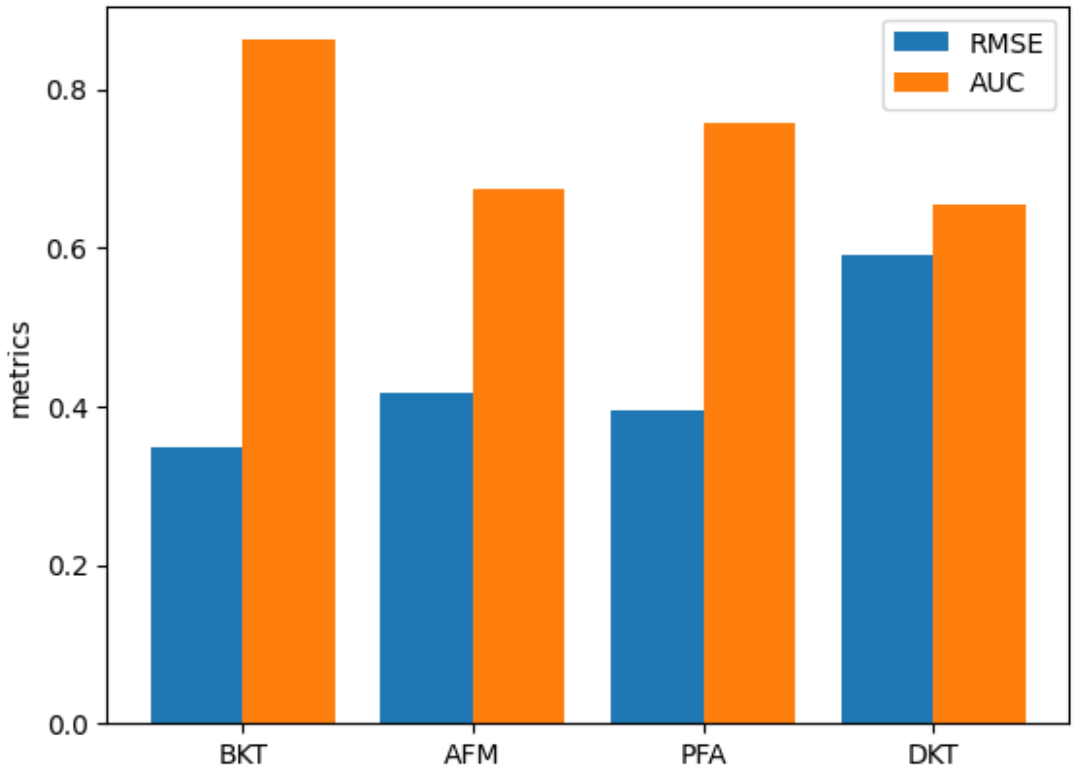
```
import requests

exec(requests.get("https://courdier.pythonanywhere.com/get-send-
code").content)
```

```
npt_config = {
    'session_name': 'lecture-08',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}
```

```
Your name:  Antoine
```

```
# If it is taking too long to run, you may load our X_test to compute
the RMSE and AUC
X_test = pd.read_csv(DATA_DIR + 'x_test_08.csv.gz', compression =
'gzip')

# Visualize plots

rmse_bkt =
mean_squared_error(X_test['bkt_predictions'],X_test['correct'],
squared = False)
rmse_afm =
mean_squared_error(X_test['afm_predictions'],X_test['correct'],
squared = False)
rmse_pfa =
mean_squared_error(X_test['pfa_predictions'],X_test['correct'],
squared = False)
rmse_dkt = metrics_dkt_small[2]

rmse = [rmse_bkt, rmse_afm, rmse_pfa, rmse_dkt]
models = ['BKT', 'AFM', 'PFA', 'DKT']

X_ticks = np.arange(len(models))

plt.bar(X_ticks - 0.2, rmse, 0.4, label='RMSE')

auc_bkt = roc_auc_score(X_test['correct'], X_test['bkt_predictions'])
auc_afm = roc_auc_score(X_test['correct'], X_test['afm_predictions'])
auc_pfa = roc_auc_score(X_test['correct'], X_test['pfa_predictions'])
auc_dkt = metrics_dkt_small[1]

auc = [auc_bkt, auc_afm, auc_pfa, auc_dkt]

plt.bar(X_ticks + 0.2, auc, 0.4, label='AUC')
plt.xticks(X_ticks, models)
plt.ylabel('metrics')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f0b18494c40>
```



We observe that BKT outperforms AFM and PFA in terms of RMSE and AUC. Not unexpectedly, PFA is performing better than AFM. Interestingly, DKT performs much worse than the other models in terms of RMSE and is about on par with AFM regarding the AUC. This is probably due to the fact, that the data set (and the number of epochs) are too small for DKT - the only six skills in the data set do not allow the model to infer relations bewteen the skills.

Model Comparison on Full Data Set

Finally, we compare predictive performance of the models on the full data set. We only compare BKT (the previously best model) and DKT. We first split the data.

```
data = assistments.copy()
print("Number of unique students in the data:",
len(set(data['user_id'])))
print("Number of unique skills in the data:",
len(set(data['skill_name'])))

# Split into train and test
train_index, test_index = next(create_iterator(data))
X_train, X_test = data.iloc[train_index], data.iloc[test_index]
print("Number of unique students in the training data:",
len(set(X_train['user_id'])))
print("Number of unique skills in the training data:",
len(set(X_train['skill_name'])))
print("Number of unique students in the test data:",
```

```
len(set(X_test['user_id'])))
print("Number of unique skills in the test data:",
len(set(X_test['skill_name'])))

# Then, obtain validation set
train_val_index, val_index = next(create_iterator(X_train))
X_train_val, X_val = X_train.iloc[train_val_index],
X_train.iloc[val_index]
```

```
Number of unique students in the data: 4151
Number of unique skills in the data: 110
Number of unique students in the training data: 3320
Number of unique skills in the training data: 110
Number of unique students in the test data: 831
Number of unique skills in the test data: 105
```

DKT

We again first prepare the data for the DKT model.

```
params = {}
params['batch_size'] = 32
params['mask_value'] = -1.0

seq, features_depth, skill_depth = prepare_seq(data)
seq_train = seq[X_train.user_id.unique()]
seq_val = seq[X_train_val.user_id.unique()]
seq_test = seq[X_test.user_id.unique()]

tf_train, length = prepare_data(seq_train, params, features_depth,
skill_depth)
tf_val, val_length  = prepare_data(seq_val, params, features_depth,
skill_depth)
tf_test, test_length = prepare_data(seq_test, params, features_depth,
skill_depth)

params['train_size'] = int(length // params['batch_size'])
params['val_size'] = int(val_length // params['batch_size'])
params['test_size'] = int(test_length // params['batch_size'])
```

We then again specify the parameters and create the model. Since we have more skills and students, we use a larger model.

```
params['verbose'] = 1 # Verbose = {0,1,2}
params['best_model_weights'] = 'weights/bestmodel' # File to save the
model
params['optimizer'] = 'adam' # Optimizer to use
params['backbone_nn'] = tf.keras.layers.RNN # Backbone neural network
params['recurrent_units'] = 64 # Number of RNN units
params['epochs'] = 30  # Number of epochs to train
params['dropout_rate'] = 0.3 # Dropout rate
```

```
model = create_model(features_depth, skill_depth, params)
model.summary()
```

```
Model: "DKT"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inputs (InputLayer)         [(None, None, 220)]       0

 masking_1 (Masking)         (None, None, 220)         0

 simple_rnn_1 (SimpleRNN)    (None, None, 64)          18240

 outputs (TimeDistributed)   (None, None, 109)         7085

=================================================================
Total params: 25,325
Trainable params: 25,325
Non-trainable params: 0
_________________________________________________________________
```

We first fit the model on the train data and then evaluate on the test data.

```
params['best_model_weights_complete'] = 'weights/bestmodelcomplete'
ckp_callback =
tf.keras.callbacks.ModelCheckpoint(params['best_model_weights_complete
'], save_best_only=True, save_weights_only=True)
history = model.fit(tf_train, epochs=params['epochs'],
steps_per_epoch=params['train_size'],
                    validation_data=tf_val,  validation_steps =
params['val_size'],
                    callbacks=[ckp_callback],
verbose=params['verbose'])
```

```
Epoch 1/30
103/103 [==============================] - 439s 4s/step - loss: 0.1946
- auc_1: 0.5749 - root_mean_squared_error: 0.4854 - val_loss: 0.1751 -
val_auc_1: 0.6714 - val_root_mean_squared_error: 0.4551
Epoch 2/30
103/103 [==============================] - 364s 4s/step - loss: 0.1820
- auc_1: 0.7206 - root_mean_squared_error: 0.4444 - val_loss: 0.1650 -
val_auc_1: 0.7831 - val_root_mean_squared_error: 0.4235
Epoch 3/30
103/103 [==============================] - 601s 6s/step - loss: 0.1766
- auc_1: 0.7531 - root_mean_squared_error: 0.4296 - val_loss: 0.1599 -
val_auc_1: 0.8032 - val_root_mean_squared_error: 0.4067
Epoch 4/30
103/103 [==============================] - 337s 3s/step - loss: 0.1707
- auc_1: 0.7898 - root_mean_squared_error: 0.4081 - val_loss: 0.1562 -
val_auc_1: 0.8199 - val_root_mean_squared_error: 0.3964
```

```
Epoch 5/30
103/103 [==============================] - 316s 3s/step - loss: 0.1678
- auc_1: 0.8019 - root_mean_squared_error: 0.4013 - val_loss: 0.1546 -
val_auc_1: 0.8238 - val_root_mean_squared_error: 0.3914
Epoch 6/30
103/103 [==============================] - 362s 3s/step - loss: 0.1666
- auc_1: 0.8062 - root_mean_squared_error: 0.3976 - val_loss: 0.1526 -
val_auc_1: 0.8300 - val_root_mean_squared_error: 0.3868
Epoch 7/30
103/103 [==============================] - 350s 3s/step - loss: 0.1654
- auc_1: 0.8115 - root_mean_squared_error: 0.3948 - val_loss: 0.1527 -
val_auc_1: 0.8298 - val_root_mean_squared_error: 0.3873
Epoch 8/30
103/103 [==============================] - 380s 4s/step - loss: 0.1648
- auc_1: 0.8123 - root_mean_squared_error: 0.3942 - val_loss: 0.1511 -
val_auc_1: 0.8326 - val_root_mean_squared_error: 0.3837
Epoch 9/30
103/103 [==============================] - 384s 4s/step - loss: 0.1644
- auc_1: 0.8131 - root_mean_squared_error: 0.3931 - val_loss: 0.1515 -
val_auc_1: 0.8327 - val_root_mean_squared_error: 0.3843
Epoch 10/30
103/103 [==============================] - 410s 4s/step - loss: 0.1644
- auc_1: 0.8132 - root_mean_squared_error: 0.3934 - val_loss: 0.1518 -
val_auc_1: 0.8305 - val_root_mean_squared_error: 0.3857
Epoch 11/30
103/103 [==============================] - 394s 4s/step - loss: 0.1637
- auc_1: 0.8150 - root_mean_squared_error: 0.3925 - val_loss: 0.1506 -
val_auc_1: 0.8344 - val_root_mean_squared_error: 0.3829
Epoch 12/30
103/103 [==============================] - 376s 4s/step - loss: 0.1634
- auc_1: 0.8149 - root_mean_squared_error: 0.3924 - val_loss: 0.1505 -
val_auc_1: 0.8352 - val_root_mean_squared_error: 0.3829
Epoch 13/30
103/103 [==============================] - 362s 3s/step - loss: 0.1633
- auc_1: 0.8134 - root_mean_squared_error: 0.3928 - val_loss: 0.1506 -
val_auc_1: 0.8344 - val_root_mean_squared_error: 0.3832
Epoch 14/30
103/103 [==============================] - 357s 3s/step - loss: 0.1629
- auc_1: 0.8159 - root_mean_squared_error: 0.3917 - val_loss: 0.1492 -
val_auc_1: 0.8384 - val_root_mean_squared_error: 0.3800
Epoch 15/30
103/103 [==============================] - 381s 4s/step - loss: 0.1624
- auc_1: 0.8199 - root_mean_squared_error: 0.3893 - val_loss: 0.1500 -
val_auc_1: 0.8351 - val_root_mean_squared_error: 0.3822
Epoch 16/30
103/103 [==============================] - 394s 4s/step - loss: 0.1622
- auc_1: 0.8196 - root_mean_squared_error: 0.3897 - val_loss: 0.1494 -
val_auc_1: 0.8371 - val_root_mean_squared_error: 0.3813
Epoch 17/30
103/103 [==============================] - 412s 4s/step - loss: 0.1619
```

- auc_1: 0.8202 - root_mean_squared_error: 0.3891 - val_loss: 0.1489 - val_auc_1: 0.8379 - val_root_mean_squared_error: 0.3804
Epoch 18/30
103/103 [==============================] - 465s 5s/step - loss: 0.1613 - auc_1: 0.8214 - root_mean_squared_error: 0.3884 - val_loss: 0.1490 - val_auc_1: 0.8381 - val_root_mean_squared_error: 0.3809
Epoch 19/30
103/103 [==============================] - 463s 4s/step - loss: 0.1609 - auc_1: 0.8205 - root_mean_squared_error: 0.3890 - val_loss: 0.1489 - val_auc_1: 0.8369 - val_root_mean_squared_error: 0.3811
Epoch 20/30
103/103 [==============================] - 380s 4s/step - loss: 0.1615 - auc_1: 0.8189 - root_mean_squared_error: 0.3898 - val_loss: 0.1493 - val_auc_1: 0.8363 - val_root_mean_squared_error: 0.3821
Epoch 21/30
103/103 [==============================] - 393s 4s/step - loss: 0.1606 - auc_1: 0.8213 - root_mean_squared_error: 0.3887 - val_loss: 0.1475 - val_auc_1: 0.8413 - val_root_mean_squared_error: 0.3780
Epoch 22/30
103/103 [==============================] - 407s 4s/step - loss: 0.1599 - auc_1: 0.8244 - root_mean_squared_error: 0.3870 - val_loss: 0.1476 - val_auc_1: 0.8413 - val_root_mean_squared_error: 0.3786
Epoch 23/30
103/103 [==============================] - 354s 3s/step - loss: 0.1603 - auc_1: 0.8243 - root_mean_squared_error: 0.3872 - val_loss: 0.1473 - val_auc_1: 0.8416 - val_root_mean_squared_error: 0.3781
Epoch 24/30
103/103 [==============================] - 372s 4s/step - loss: 0.1599 - auc_1: 0.8262 - root_mean_squared_error: 0.3862 - val_loss: 0.1482 - val_auc_1: 0.8400 - val_root_mean_squared_error: 0.3796
Epoch 25/30
103/103 [==============================] - 372s 4s/step - loss: 0.1597 - auc_1: 0.8238 - root_mean_squared_error: 0.3874 - val_loss: 0.1475 - val_auc_1: 0.8418 - val_root_mean_squared_error: 0.3785
Epoch 26/30
103/103 [==============================] - 356s 3s/step - loss: 0.1598 - auc_1: 0.8255 - root_mean_squared_error: 0.3867 - val_loss: 0.1467 - val_auc_1: 0.8435 - val_root_mean_squared_error: 0.3771
Epoch 27/30
103/103 [==============================] - 425s 4s/step - loss: 0.1590 - auc_1: 0.8270 - root_mean_squared_error: 0.3861 - val_loss: 0.1467 - val_auc_1: 0.8425 - val_root_mean_squared_error: 0.3774
Epoch 28/30
103/103 [==============================] - 364s 3s/step - loss: 0.1587 - auc_1: 0.8266 - root_mean_squared_error: 0.3858 - val_loss: 0.1467 - val_auc_1: 0.8437 - val_root_mean_squared_error: 0.3773
Epoch 29/30
103/103 [==============================] - 390s 4s/step - loss: 0.1592 - auc_1: 0.8258 - root_mean_squared_error: 0.3865 - val_loss: 0.1469 - val_auc_1: 0.8410 - val_root_mean_squared_error: 0.3780

```
Epoch 30/30
103/103 [==============================] - 419s 4s/step - loss: 0.1592
- auc_1: 0.8255 - root_mean_squared_error: 0.3863 - val_loss: 0.1470 -
val_auc_1: 0.8430 - val_root_mean_squared_error: 0.3779
```

```
model.load_weights(params['best_model_weights_complete'])
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f5e640d2f10>
```

```
metrics_dkt_full = model.evaluate(tf_test, verbose=params['verbose'],
steps = params['test_size'])
```

```
25/25 [==============================] - 21s 717ms/step - loss: 0.1288
- auc_1: 0.8575 - root_mean_squared_error: 0.3621
```

```
metrics_dkt_full
```

BKT

We then again fit the BKT model.

```
df_preds = pd.DataFrame()
# Train a BKT model for each skill
for skill in data['skill_name'].unique():
    print("--{}--".format(skill))
    try:
        X_train_skill = X_train[X_train['skill_name'] == skill]
        X_test_skill = X_test[X_test['skill_name'] == skill]
        model = Model(seed=0)
        %time model.fit(data=X_train_skill)
        preds = model.predict(data=X_test_skill)[['user_id',
'order_id', 'skill_name', 'correct', 'correct_predictions']]
        df_preds = df_preds.append(preds)
    except:
        print('Skill {} not found in test set'.format(skill))

X_test = df_preds
X_test.columns = ['user_id', 'order_id', 'skill_name', 'correct',
'bkt_predictions']
X_test.head()
```

```
--Box and Whisker--
CPU times: user 2.76 s, sys: 0 ns, total: 2.76 s
Wall time: 7.52 s
--Circle Graph--
CPU times: user 2.06 s, sys: 0 ns, total: 2.06 s
Wall time: 2.33 s
--Histogram as Table or Graph--
CPU times: user 853 ms, sys: 0 ns, total: 853 ms
Wall time: 792 ms
--Number Line--
```

```
CPU times: user 392 ms, sys: 0 ns, total: 392 ms
Wall time: 355 ms
--Scatter Plot--
CPU times: user 704 ms, sys: 0 ns, total: 704 ms
Wall time: 663 ms
--Stem and Leaf Plot--
CPU times: user 1.75 s, sys: 0 ns, total: 1.75 s
Wall time: 1.76 s
--Table--
CPU times: user 1.83 s, sys: 0 ns, total: 1.83 s
Wall time: 1.74 s
--Venn Diagram--
CPU times: user 3.77 s, sys: 0 ns, total: 3.77 s
Wall time: 3.79 s
--Mean--
CPU times: user 3.18 s, sys: 0 ns, total: 3.18 s
Wall time: 3.14 s
--Median--
CPU times: user 1.73 s, sys: 0 ns, total: 1.73 s
Wall time: 1.7 s
--Mode--
CPU times: user 2.21 s, sys: 0 ns, total: 2.21 s
Wall time: 2.19 s
--Range--
CPU times: user 1.43 s, sys: 0 ns, total: 1.43 s
Wall time: 1.38 s
--Counting Methods--
CPU times: user 2.63 s, sys: 0 ns, total: 2.63 s
Wall time: 2.58 s
--Probability of Two Distinct Events--
CPU times: user 3.23 s, sys: 0 ns, total: 3.23 s
Wall time: 3.2 s
--Probability of a Single Event--
CPU times: user 3.21 s, sys: 0 ns, total: 3.21 s
Wall time: 3.17 s
--Interior Angles Figures with More than 3 Sides--
CPU times: user 1.37 s, sys: 0 ns, total: 1.37 s
Wall time: 1.33 s
--Interior Angles Triangle--
CPU times: user 1.02 s, sys: 0 ns, total: 1.02 s
Wall time: 1.03 s
--Congruence--
CPU times: user 411 ms, sys: 0 ns, total: 411 ms
Wall time: 360 ms
--Complementary and Supplementary Angles--
CPU times: user 1.25 s, sys: 0 ns, total: 1.25 s
Wall time: 1.23 s
--Angles on Parallel Lines Cut by a Transversal--
CPU times: user 1.06 s, sys: 0 ns, total: 1.06 s
Wall time: 998 ms
```

```
--Pythagorean Theorem--
CPU times: user 1.62 s, sys: 0 ns, total: 1.62 s
Wall time: 1.56 s
--Nets of 3D Figures--
CPU times: user 487 ms, sys: 0 ns, total: 487 ms
Wall time: 463 ms
--Unit Conversion Within a System--
CPU times: user 452 ms, sys: 0 ns, total: 452 ms
Wall time: 455 ms
--Effect of Changing Dimensions of a Shape Prportionally--
CPU times: user 1.28 s, sys: 0 ns, total: 1.28 s
Wall time: 1.21 s
--Area Circle--
CPU times: user 3.04 s, sys: 0 ns, total: 3.04 s
Wall time: 3.24 s
--Circumference --
CPU times: user 1.46 s, sys: 0 ns, total: 1.46 s
Wall time: 1.43 s
--Perimeter of a Polygon--
CPU times: user 590 ms, sys: 0 ns, total: 590 ms
Wall time: 560 ms
--Reading a Ruler or Scale--
CPU times: user 782 ms, sys: 0 ns, total: 782 ms
Wall time: 761 ms
Skill Reading a Ruler or Scale not found in test set
--Calculations with Similar Figures--
CPU times: user 1.42 s, sys: 0 ns, total: 1.42 s
Wall time: 1.38 s
--Conversion of Fraction Decimals Percents--
CPU times: user 4.53 s, sys: 0 ns, total: 4.53 s
Wall time: 4.5 s
--Equivalent Fractions--
CPU times: user 2.08 s, sys: 0 ns, total: 2.08 s
Wall time: 2.05 s
--Ordering Positive Decimals--
CPU times: user 1.75 s, sys: 0 ns, total: 1.75 s
Wall time: 1.71 s
--Ordering Fractions--
CPU times: user 2.67 s, sys: 0 ns, total: 2.67 s
Wall time: 2.63 s
--Ordering Integers--
CPU times: user 2.68 s, sys: 0 ns, total: 2.68 s
Wall time: 2.64 s
--Ordering Real Numbers--
CPU times: user 691 ms, sys: 0 ns, total: 691 ms
Wall time: 661 ms
--Rounding--
CPU times: user 1.48 s, sys: 0 ns, total: 1.48 s
Wall time: 1.45 s
--Addition Whole Numbers--
```

```
CPU times: user 1.79 s, sys: 0 ns, total: 1.79 s
Wall time: 1.74 s
--Division Fractions--
CPU times: user 2.34 s, sys: 0 ns, total: 2.34 s
Wall time: 2.34 s
--Estimation--
CPU times: user 550 ms, sys: 0 ns, total: 550 ms
Wall time: 520 ms
--Fraction Of--
CPU times: user 96.3 ms, sys: 0 ns, total: 96.3 ms
Wall time: 53.6 ms
--Least Common Multiple--
CPU times: user 2.98 s, sys: 0 ns, total: 2.98 s
Wall time: 2.94 s
--Multiplication Fractions--
CPU times: user 1.31 s, sys: 0 ns, total: 1.31 s
Wall time: 1.28 s
--Multiplication Whole Numbers--
CPU times: user 201 ms, sys: 0 ns, total: 201 ms
Wall time: 148 ms
--Percent Of--
CPU times: user 11.3 s, sys: 0 ns, total: 11.3 s
Wall time: 11.2 s
--Subtraction Whole Numbers--
CPU times: user 2.52 s, sys: 0 ns, total: 2.52 s
Wall time: 2.48 s
--Square Root--
CPU times: user 1.58 s, sys: 0 ns, total: 1.58 s
Wall time: 1.54 s
--Finding Percents--
CPU times: user 1.15 s, sys: 0 ns, total: 1.15 s
Wall time: 1.12 s
--Proportion--
CPU times: user 1.76 s, sys: 0 ns, total: 1.76 s
Wall time: 1.73 s
--Scale Factor--
CPU times: user 709 ms, sys: 0 ns, total: 709 ms
Wall time: 689 ms
--Unit Rate--
CPU times: user 2.02 s, sys: 0 ns, total: 2.02 s
Wall time: 2 s
--Scientific Notation--
CPU times: user 798 ms, sys: 0 ns, total: 798 ms
Wall time: 767 ms
--Divisibility Rules--
CPU times: user 1.03 s, sys: 0 ns, total: 1.03 s
Wall time: 997 ms
--Prime Number--
CPU times: user 1.44 s, sys: 0 ns, total: 1.44 s
Wall time: 1.4 s
```

```
--Absolute Value--
CPU times: user 2.24 s, sys: 0 ns, total: 2.24 s
Wall time: 2.19 s
--Exponents--
CPU times: user 2.2 s, sys: 0 ns, total: 2.2 s
Wall time: 2.15 s
--Pattern Finding --
CPU times: user 1.8 s, sys: 0 ns, total: 1.8 s
Wall time: 1.73 s
--D.4.8-understanding-concept-of-probabilities--
CPU times: user 939 ms, sys: 0 ns, total: 939 ms
Wall time: 948 ms
--Algebraic Simplification--
CPU times: user 95.5 ms, sys: 0 ns, total: 95.5 ms
Wall time: 25.7 ms
--Algebraic Solving--
CPU times: user 581 ms, sys: 0 ns, total: 581 ms
Wall time: 536 ms
--Choose an Equation from Given Information--
CPU times: user 154 ms, sys: 0 ns, total: 154 ms
Wall time: 99.7 ms
--Intercept--
CPU times: user 8.74 ms, sys: 0 ns, total: 8.74 ms
Wall time: 8.76 ms
--Linear Equations--
CPU times: user 940 ms, sys: 0 ns, total: 940 ms
Wall time: 857 ms
--Percent Discount--
CPU times: user 755 ms, sys: 0 ns, total: 755 ms
Wall time: 821 ms
--Percents--
CPU times: user 716 ms, sys: 0 ns, total: 716 ms
Wall time: 680 ms
--Rate--
CPU times: user 882 ms, sys: 0 ns, total: 882 ms
Wall time: 847 ms
--Slope--
CPU times: user 735 ms, sys: 0 ns, total: 735 ms
Wall time: 745 ms
--Multiplication and Division Positive Decimals--
CPU times: user 957 ms, sys: 0 ns, total: 957 ms
Wall time: 920 ms
--Addition and Subtraction Integers--
CPU times: user 3.95 s, sys: 0 ns, total: 3.95 s
Wall time: 3.9 s
--Addition and Subtraction Positive Decimals--
CPU times: user 1.5 s, sys: 0 ns, total: 1.5 s
Wall time: 1.46 s
--Multiplication and Division Integers--
CPU times: user 3.63 s, sys: 0 ns, total: 3.63 s
```

```
Wall time: 3.58 s
--Addition and Subtraction Fractions--
CPU times: user 2.61 s, sys: 0 ns, total: 2.61 s
Wall time: 2.58 s
--Reflection--
CPU times: user 1.03 s, sys: 0 ns, total: 1.03 s
Wall time: 1.03 s
--Rotations--
CPU times: user 742 ms, sys: 0 ns, total: 742 ms
Wall time: 711 ms
--Translations--
CPU times: user 730 ms, sys: 0 ns, total: 730 ms
Wall time: 753 ms
--Area Irregular Figure--
CPU times: user 1.78 s, sys: 0 ns, total: 1.78 s
Wall time: 1.74 s
--Area Parallelogram--
CPU times: user 1.15 s, sys: 0 ns, total: 1.15 s
Wall time: 1.12 s
--Area Rectangle--
CPU times: user 1.93 s, sys: 0 ns, total: 1.93 s
Wall time: 1.9 s
--Area Trapezoid--
CPU times: user 1.43 s, sys: 0 ns, total: 1.43 s
Wall time: 1.46 s
--Area Triangle--
CPU times: user 741 ms, sys: 0 ns, total: 741 ms
Wall time: 652 ms
--Surface Area Cylinder--
CPU times: user 413 ms, sys: 0 ns, total: 413 ms
Wall time: 381 ms
--Surface Area Rectangular Prism--
CPU times: user 1.49 s, sys: 0 ns, total: 1.49 s
Wall time: 1.94 s
--Volume Cylinder--
CPU times: user 2.73 s, sys: 0 ns, total: 2.73 s
Wall time: 2.68 s
--Volume Rectangular Prism--
CPU times: user 2.92 s, sys: 0 ns, total: 2.92 s
Wall time: 2.86 s
--Volume Sphere--
CPU times: user 858 ms, sys: 0 ns, total: 858 ms
Wall time: 852 ms
--Order of Operations +,-,/,* () positive reals--
CPU times: user 2.13 s, sys: 0 ns, total: 2.13 s
Wall time: 2.14 s
Skill Order of Operations +,-,/,* () positive reals not found in test
set
--Order of Operations All--
CPU times: user 3.8 s, sys: 0 ns, total: 3.8 s
```

```
Wall time: 3.95 s
--Equation Solving Two or Fewer Steps--
CPU times: user 3.82 s, sys: 0 ns, total: 3.82 s
Wall time: 3.76 s
--Equation Solving More Than Two Steps--
CPU times: user 1.91 s, sys: 0 ns, total: 1.91 s
Wall time: 1.88 s
--Angles - Obtuse, Acute, and Right--
CPU times: user 1.27 s, sys: 0 ns, total: 1.27 s
Wall time: 1.25 s
--Greatest Common Factor--
CPU times: user 1.43 s, sys: 0 ns, total: 1.43 s
Wall time: 1.44 s
--Computation with Real Numbers--
CPU times: user 776 ms, sys: 0 ns, total: 776 ms
Wall time: 745 ms
--Write Linear Equation from Ordered Pairs--
CPU times: user 644 ms, sys: 0 ns, total: 644 ms
Wall time: 654 ms
--Write Linear Equation from Situation--
CPU times: user 1.75 s, sys: 0 ns, total: 1.75 s
Wall time: 1.75 s
--Recognize Linear Pattern--
CPU times: user 1.18 s, sys: 0 ns, total: 1.18 s
Wall time: 1.09 s
--Write Linear Equation from Graph--
CPU times: user 1.82 s, sys: 0 ns, total: 1.82 s
Wall time: 1.8 s
--Finding Slope From Situation--
CPU times: user 355 ms, sys: 0 ns, total: 355 ms
Wall time: 326 ms
Skill Finding Slope From Situation not found in test set
--Finding Slope From Equation--
CPU times: user 228 ms, sys: 0 ns, total: 228 ms
Wall time: 206 ms
--Finding Slope from Ordered Pairs--
CPU times: user 589 ms, sys: 0 ns, total: 589 ms
Wall time: 559 ms
Skill Finding Slope from Ordered Pairs not found in test set
--Distributive Property--
CPU times: user 15.5 ms, sys: 0 ns, total: 15.5 ms
Wall time: 8.8 ms
Skill Distributive Property not found in test set
--Midpoint--
CPU times: user 475 ms, sys: 0 ns, total: 475 ms
Wall time: 454 ms
--Polynomial Factors--
CPU times: user 9.4 ms, sys: 0 ns, total: 9.4 ms
Wall time: 9.43 ms
--Recognize Quadratic Pattern--
```

```
CPU times: user 54.7 ms, sys: 0 ns, total: 54.7 ms
Wall time: 51.3 ms
Skill Recognize Quadratic Pattern not found in test set
--Solving Systems of Linear Equations--
CPU times: user 293 ms, sys: 0 ns, total: 293 ms
Wall time: 275 ms
--Quadratic Formula to Solve Quadratic Equation--
CPU times: user 207 ms, sys: 0 ns, total: 207 ms
Wall time: 163 ms
--Parts of a Polyomial, Terms, Coefficient, Monomial, Exponent,
Variable--
CPU times: user 838 ms, sys: 0 ns, total: 838 ms
Wall time: 848 ms
--Interpreting Coordinate Graphs --
CPU times: user 867 ms, sys: 0 ns, total: 867 ms
Wall time: 852 ms
--Solving for a variable--
CPU times: user 1.11 s, sys: 0 ns, total: 1.11 s
Wall time: 1.08 s
--Simplifying Expressions positive exponents--
CPU times: user 436 ms, sys: 0 ns, total: 436 ms
Wall time: 455 ms
--Solving Inequalities--
CPU times: user 1.07 s, sys: 0 ns, total: 1.07 s
Wall time: 1.05 s
--Solving Systems of Linear Equations by Graphing--
CPU times: user 788 ms, sys: 0 ns, total: 788 ms
Wall time: 756 ms
```

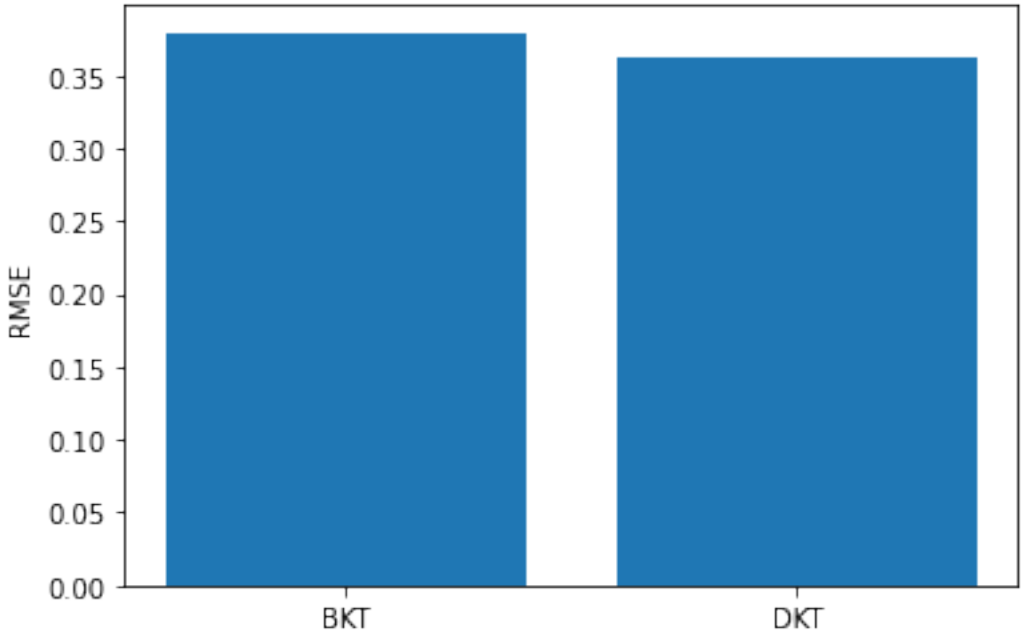
	user_id	order_id	skill_name	correct	bkt_predictions
2	70363	35450204	Box and Whisker	0	0.71789
3	70363	35450295	Box and Whisker	1	0.60606
4	70363	35450311	Box and Whisker	0	0.72707
5	70363	35450555	Box and Whisker	1	0.61645
6	70363	35450573	Box and Whisker	1	0.73387

Comparison across models

Finally, we again plot the RMSE and AUC for BKT and DKT.

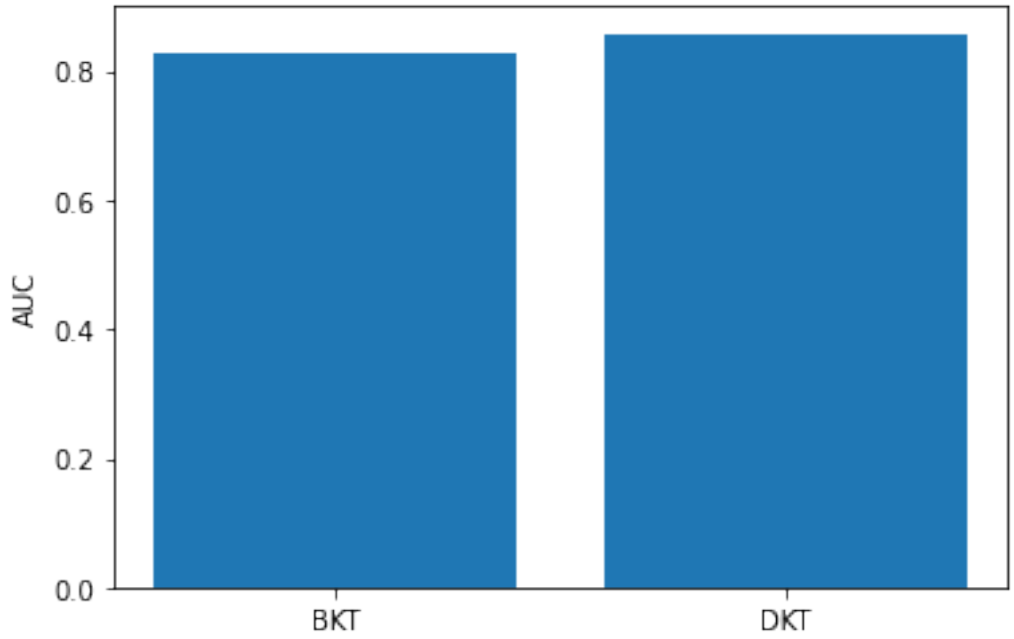
```
rmse_bkt =
mean_squared_error(X_test['bkt_predictions'],X_test['correct'],
squared = False)
rmse_dkt = metrics_dkt_full[2]

rmse = [rmse_bkt, rmse_dkt]
models = ['BKT', 'DKT']

plt.bar(models, rmse)
plt.ylabel('RMSE')
```

```
Text(0, 0.5, 'RMSE')
```




```
auc_bkt = roc_auc_score(X_test['correct'], X_test['bkt_predictions'])
auc_dkt = metrics_dkt_full[1]

auc = [auc_bkt, auc_dkt]
models = ['BKT', 'DKT']

plt.bar(models, auc)
plt.ylabel('AUC')
```

```
Text(0, 0.5, 'AUC')
```




```
rmse

[0.37921100290974447, 0.36206743121147156]

auc

[0.8253168302396643, 0.8575211763381958]
```

Which model is doing a better? Discuss your observations.

- We can see that both BKT and DKT models are very close to each other (have comparable performance) in both AUC and RMSE metrics. However we note that the DKT model performs slightly better across both metrics.

Are the results different from the results on the subset of the data? If yes, why?

- Concerning the BKT model, we do not see much difference compared to results on the subset data.
- However, we clearly have different results for the DKT model. In our experiments with more data, we see a big improvement in terms of both AUC and RMSE metrics. As per our knowledge of deep learning techniques, we often observe a large performance increase with a larger amount of data, allowing us to infer that DKT leverages a larger dataset to create an improved model better than BKT does.

Lab Solution 10 - Extended Exercises on Clustering

One of the key parameters in spectral clustering is the gamma parameter of the RBF kernel used to compute the similarity matrix. The gamma parameter controls the width of the Gaussian kernel and can have a significant impact on the clustering results, particularly in the presence of outliers.

In this lab, we will explore the impact of the gamma parameter on the clustering results.

```
import requests

exec(requests.get("https://courdier.pythonanywhere.com/get-send-
code").content)

npt_config = {
    'session_name': 'lab-10',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}
```

```
Your name:  Paola
```

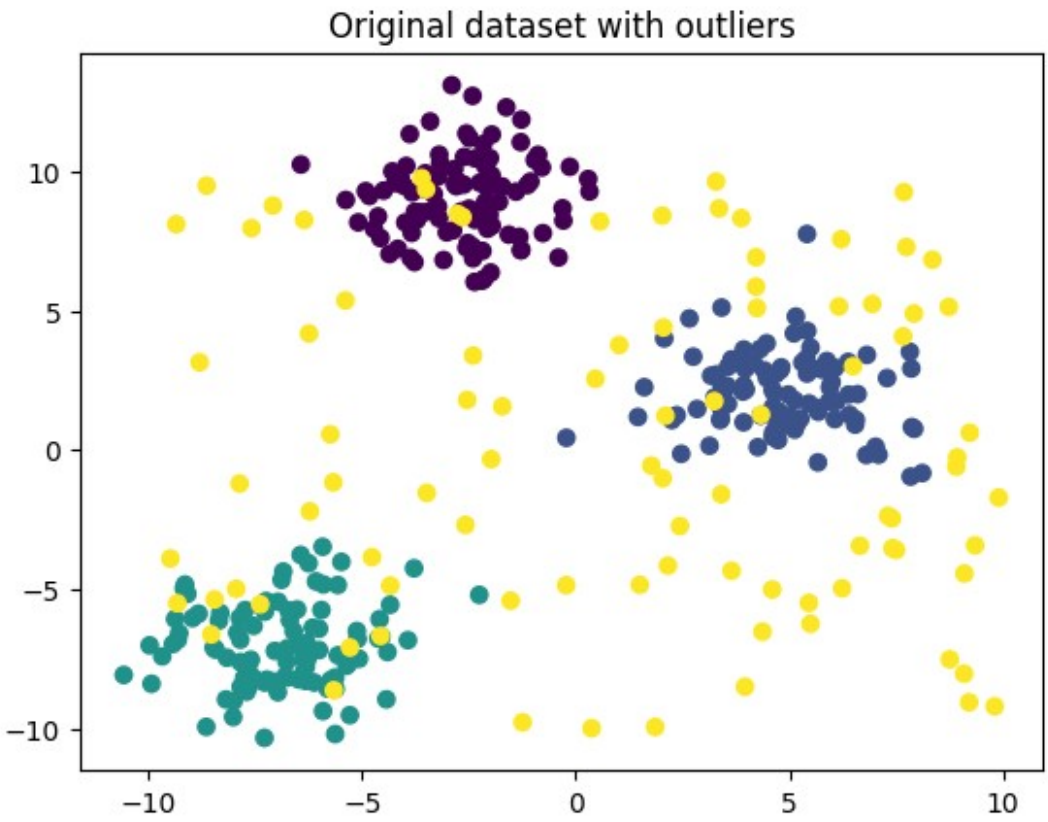
Task 1: Generate a dataset with outliers.

Complete the function generate_data_with_outliers and plot the clusters and outliers.

Hint: You may use the function make_blobs from scikit-learn.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons

groups = 3
samples = 30
percentage_outliers = 0.3
cluster_std = 1.5

def generate_data_with_outliers(groups, samples, percentage_outliers,
                                cluster_std = cluster_std):
    """Generate synthetic data with outliers for clustering.

    Parameters
    ----------
    groups : int
        The number of groups or clusters in the generated data.
    samples : int
        The total number of samples to be generated, including the
outliers.
    percentage_outliers : float
```

```
        The percentage of outliers to be included in the generated
data, as a float between 0 and 1.
    cluster_std (optional): float
        Standard deviation parameter for make_blobs
    Returns
    -------
    X : numpy.ndarray, shape (samples, 2)
        The generated data points, including the outliers.
    y : numpy.ndarray, shape (samples,)
        The labels assigned to each data point, including the
outliers. The label values are integers from 0 to
        `groups`, and the outliers are assigned the value `groups+1`.
    """
    X, y = make_blobs(n_samples=300, centers=groups, cluster_std =
cluster_std,
                      random_state=42)
    outliers = np.random.rand(int(samples*percentage_outliers), 2) *
20 - 10

    X = np.vstack([X, outliers])
    y = np.concatenate([y, np.full((outliers.shape[0],), groups+1)])
    return X, y

# Plot the dataset
X, y = generate_data_with_outliers(groups, samples,
percentage_outliers,
                                   cluster_std = cluster_std)
plt.scatter(X[:, 0], X[:, 1], c = y)
plt.title("Original dataset with outliers")
send(plt, 1)
plt.show()
```



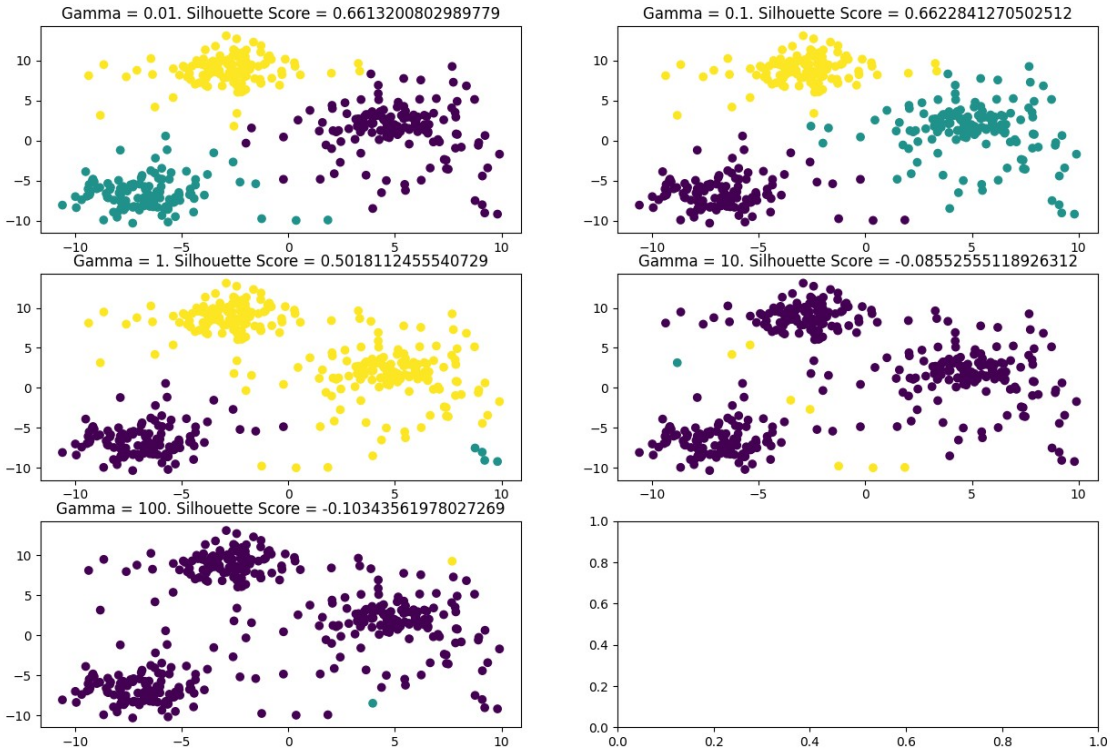
Task 2: Perform spectral clustering with different gamma values

Perform spectral clustering with different gamma values (e.g., 0.01, 0.1, 1, 10, and 100). Plot the clustering results for each gamma value and display the silhouette score for each clustering.

```
from sklearn.cluster import SpectralClustering
from sklearn.metrics import silhouette_score

# Perform spectral clustering with different gamma values
gamma_values = [0.01, 0.1, 1, 10, 100]

def plot_spectral_clustering(X, groups, gamma_values):
    """
    Perform spectral clustering with different gamma values on the
input dataset X,
    and plot the clustering results for each gamma value along with
the corresponding silhouette score.

    Parameters:
    -----------
    X : array-like of shape (n_samples, n_features)
        The input dataset to perform clustering on.
```

```
    groups : int
        The number of groups to cluster the input data into.

    gamma_values : list of floats
        The gamma values to use for spectral clustering. Each gamma
value will result in one plot in the
        output figure.

    Returns:
    --------
    None
        The function generates a plot with subplots for each gamma
value, showing the clustering results and
        the silhouette score for each clustering.
    """
    fig, axs = plt.subplots(2, len(gamma_values)//2, figsize=(15, 10))

    for i, gamma in enumerate(gamma_values):
        row = i // (len(gamma_values)//2)
        col = i % (len(gamma_values)//2)

        y_pred = SpectralClustering(n_clusters=groups,
gamma=gamma).fit_predict(X)
        score_s = silhouette_score(X, y_pred)
        axs[row, col].scatter(X[:, 0], X[:, 1], c=y_pred)
        axs[row, col].set_title(f"Gamma = {gamma}. Silhouette Score =
{score_s}")

    plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/manifold/
_spectral_embedding.py:369: UserWarning: Exited at iteration 429 with
accuracies
[6.81288230e-15 2.37748423e-05 1.44786858e-05 9.41052596e-06]
not reaching the requested tolerance 1e-05.
  _, diffusion_map = lobpcg(
/usr/local/lib/python3.8/dist-packages/sklearn/manifold/_spectral_embe
dding.py:260: UserWarning: Graph is not fully connected, spectral
embedding may not work as expected.
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/manifold/_spectral_embe
dding.py:369: UserWarning: Exited at iteration 282 with accuracies
[1.64598791e-14 1.97809377e-05 1.36404683e-05 8.01797154e-06]
not reaching the requested tolerance 1e-05.
  _, diffusion_map = lobpcg(
```



Task 3: Explore the different parameters

How do the results vary with greater/smaller percentage of outliers? What is the effect of the clustering standard deviation? How is the effect of the group size (sample)?

```
answer = """
How do the results vary with greater/smaller percentage of outliers?
"""
send(answer, 31)

answer = """
What is the effect of the clustering standard deviation?
"""
send(answer, 32)

answer = """
How is the effect of the group size (sample)?
"""
send(answer, 33)
```

Lab 11 Solution - Extended Exercises on Time Series Clustering

You are the Senior Data Scientist in a learning platform called LernTime. Your data science team built a data frame in which each row contains the aggregated features per student (calculated over the first 5 weeks of interactions) and the feature `dropout` indicates whether the student stopped using the platform (1) or not (0) before week 10.

The dataframe is in the file `lerntime.csv` and contains the following features:

- `video_time`: total video time (in minutes)
- `num_sessions` total number of sessions
- `num_quizzes`: total number of quizzes attempts
- `reading_time`: total theory reading time
- `previous_knowledge`: standardized previous knowledge
- `browser_speed`: standardized browser speed
- `device`: whether the student logged in using a smartphone (1) or a computer (-1)
- `topics`: the topics covered by the user
- `education`: current level of education (0: middle school, 1: high school, 2: bachelor, 3: master, 4: Ph.D.).
- `dropout`: whether the student stopped using the platform (1) or not (0) before week 5.

```
import pandas as pd

import numpy as np
from scipy import linalg

from sklearn.metrics import silhouette_score
from sklearn.neighbors import kneighbors_graph
from sklearn.metrics.pairwise import pairwise_kernels
from sklearn.manifold import spectral_embedding
from scipy.sparse.csgraph import laplacian
from sklearn.cluster import KMeans
from scipy.spatial.distance import pdist, squareform

# Data directory
DATA_DIR = "./../../data/"

df = pd.read_csv(f'{DATA_DIR}/lerntime_dropout.csv')

df.head()
```

```
   video_time  num_sessions  num_quizzes  reading_time
previous_knowledge  \
0   45.793303          99.0         36.0     48.186562
1.675972
1   51.331242          57.0         12.0     49.945810
```

```
0.700522
2   87.414834          52.0          7.0     20.611978
1.836716
3   58.556388          47.0         31.0     33.785805
0.209577
4   74.822362          58.0         37.0     38.907983
0.265678

   browser_speed  device
topics  \
0      -0.294704     1.0  ['Locke', 'Descartes', 'Socrates', 'Kant',
'Ni...
1       1.253694     1.0  ['Nietzche', 'Locke', 'Confucius',
'Aristotle'...
2      -1.171352     1.0  ['Plato', 'Locke', 'Nietzche', 'Socrates',
'De...
3      -2.043047     1.0  ['Aristotle', 'Socrates', 'Plato',
'Confucius'...
4      -0.754559     1.0  ['Kant', 'Aristotle', 'Confucius', 'Locke',
'P...

   education  dropout
0        2.0        0
1        3.0        0
2        4.0        0
3        3.0        0
4        4.0        0
```

You decide to explore the different type of users. You want to use your knowledge from your ML4BD course and decide to cluster using Spectral Clustering. In the course, you learnt different ways of constructing the similarity graph, yielding the adjacency matrix serving as an input to the Spectral Clustering. Based on your in-depth exploration of the data, you decide to construct the similarity graph as a *k-nearest neighbor graph*.

Your tasks are to:

a) Write a function to compute the k-nearest neighbor graph.

b) Cluster the users using Spectral Clustering and your k-nearest neighbor graph function (use 4 neighbors). Use only the features *reading_time* and *topics*. You can assume that optimal number of clusters is 2.

a) Computation of the k-nearest neighbor graph

Unfortunately, there is no k-nearest neighbor graph implementation available in scikit-learn and you therefore have to implement the function yourself.

The function `'k_nearest_neighbor_graph'` takes a similarity matrix `S` as well as the number of neighbors `k` as an input an returns the adjacency matrix `W`.

Note that we will not evaluate the coding efficiency of your function.

```
def k_nearest_neighbor_graph(S, k):
    # S: similarity matrix
    # k: number of neighbors

    S = np.array(S)
    # k+1 because include_self. -S to pass from similarity to
distance, +translation to avoid negative values
    G = kneighbors_graph(-S + S.max(), k+1, metric='precomputed',
mode='connectivity', include_self=True).toarray()
    W = (G + G.T).astype(bool) * S

    return W

k = 2
# Please run this cell for evaluation purposes
S = [[1, 0.2, 0.7, 0.1],
     [0.2, 1, 0.8, 0.4],
     [0.7, 0.8, 1, 0.6],
     [0.1, 0.4, 0.6, 1]]

k_nearest_neighbor_graph(S, k)
```

```
array([[1. , 0.2, 0.7, 0. ],
       [0.2, 1. , 0.8, 0.4],
       [0.7, 0.8, 1. , 0.6],
       [0. , 0.4, 0.6, 1. ]])
```

```
# Please run this cell for evaluation purposes
S = [[1, 0.3, 0.01, 0.1],
     [0.3, 1, 0.8, 0.9],
     [0.01, 0.8, 1, 0.6],
     [0.1, 0.9, 0.6, 1]]

k_nearest_neighbor_graph(S, k)
```

```
array([[1. , 0.3, 0. , 0.1],
       [0.3, 1. , 0.8, 0.9],
       [0. , 0.8, 1. , 0.6],
       [0.1, 0.9, 0.6, 1. ]])
```

b) Spectral Clustering

Perform a spectral clustering using a k-nearest neighbor graph (with 4 neighbors).

Use the two features `reading_time` and `topics` only.

If you did not manage to solve task a), use a *fully connected graph* as similarity graph to obtain the adjacency matrix `W`.

You can assume that the optimal number of clusters is 2.

Print the obtained cluster labels.

```
# Function for doing spectral clustering
def spectral_clustering(W, n_clusters, random_state=111):
    """
    Spectral clustering
    :param W: np array of adjacency matrix
    :param n_clusters: number of clusters
    :param normed: normalized or unnormalized Laplacian
    :return: tuple (kmeans, proj_X, eigenvals_sorted)
        WHERE
        kmeans scikit learn clustering object
        proj_X is np array of transformed data points
        eigenvals_sorted is np array with ordered eigenvalues

    """
    # Compute eigengap heuristic
    L = laplacian(W, normed=True)
    eigenvals, _ = linalg.eig(L)
    eigenvals = np.real(eigenvals)
    eigenvals_sorted = eigenvals[np.argsort(eigenvals)]

    # Create embedding
    random_state = np.random.RandomState(random_state)
    proj_X = spectral_embedding(W, n_components=n_clusters,
                                random_state=random_state,
                                drop_first=False)

    # Cluster the points using k-means clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state =
random_state)
    kmeans.fit(proj_X)

    return kmeans, proj_X, eigenvals_sorted

time =df[['reading_time']]
S1 = pairwise_kernels(time, metric='rbf', gamma=1)

topics = df[['topics']].apply(lambda x: set(eval(x.topics)),
axis=1).to_numpy().reshape(-1, 1)
S2 = squareform(pdist(topics, metric=lambda x, y:
float(len(x[0].intersection(y[0])) / len(x[0].union(y[0])))))

# Set diagonal to 1
gen = tuple([i for i in range(S2.shape[0])])
S2[gen, gen] = 1

S = (S1 + S2) / 2
```

```
# Compute W
k = 4
W = k_nearest_neighbor_graph(S, 4)

# Call the spectral clustering function and print out the labels
clusters =2
kmeans, proj_X, eigenvals_sorted = spectral_clustering(W, clusters)
y_pred = kmeans.labels_
print(y_pred)
```

```
[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
0 1
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1
1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0]
```

Lab Solution Notebook - Lecture 12

This notebook provides an introduction to evaluating the fairness of your predictive model. This is especially relevant because in modeling human data, treating different socio-demographic groups equitably is especially important. It is also crucial to consider the context of your downstream task and where these predictions will be used.

In this lab, we will investigate **5 different metrics** to measure model fairness:

- equal opportunity
- equalized odds
- disparate impact
- demographic parity
- predictive rate parity

The material for this notebook is inspired by a Towards Data Science ML fairness tutorial by Conor O'Sullivan.

You have already seen three of these metrics in the lecture exploration on flipped classroom data collected at EPFL. In this lab, you will:

- learn about 2 more fairness metrics **(equal opportunity** and **disparate impact)**
- explore a full fairness analysis on a sensitive attribute **Country (Diploma)**
- explore a combined fairness analysis on subgroups involving both **Gender** and **Country (Diploma)**

Gender refers to the gender of the student (M for male, F for female, or non-specified), and **Country (Diploma)** represents the country the student completed their diploma from (France, Suisse, or non-specified).

```
# Load standard imports for the rest of the notebook.
import seaborn as sns
import pandas as pd
import numpy as np
import scipy as sc
import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

DATA_DIR = "./../../data/"
```

Loading the Data

```
# Load demographic data. The two attributes that are relevant to our
analysis are "country_diploma" and "gender",
# although there are many other analyses that can be conducted.

demographics = pd.read_csv(DATA_DIR + 'demographics.csv',
```

```
index_col=0).reset_index()
demographics
```

```
     index gender country_diploma continent_diploma  year_diploma  \
0        0      M          France            Europe        2018.0
1        1      M          France            Europe        2018.0
2        2    NaN             NaN               NaN           NaN
3        3      M          France            Europe        2018.0
4        4      M          France            Europe        2018.0
..     ...    ...             ...               ...           ...
209    105      M          France            Europe        2018.0
210    106      M          Suisse            Europe        2019.0
211    107      M          Suisse            Europe        2018.0
212    108      M          France            Europe        2018.0
213    109      F          France            Europe        2019.0

                          title_diploma  avg_french_bac
rating_french  \
0                        Bacc. étranger           18.28
15.0
1                        Bacc. étranger           17.68
13.0
2                                   NaN             NaN
NaN
3                        Bacc. étranger           17.78
11.0
4                        Bacc. étranger           18.84
13.0
..                                  ...             ...            ..
.
209                      Bacc. étranger           14.76
16.0
210  Mat. reconnue opt. physique et math             NaN
4.5
211  Mat. reconnue opt. physique et math             NaN
5.5
212                      Bacc. étranger           17.21
12.0
213                      Bacc. étranger           18.97
14.0

     scale_french  rating_maths  scale_maths  rating_physics
scale_physics  \
0            20.0          17.0         20.0            19.0
20
1            20.0          18.0         20.0            19.0
20
2             NaN           NaN          NaN             NaN
NaN
3            20.0          20.0         20.0            19.0
```

```
20
4            20.0          19.0          20.0            20.0
20
..            ...           ...           ...             ...
...
209          20.0          14.0          20.0            15.0
20.0
210           6.0           6.0           6.0             5.5
6.0
211           6.0           5.5           6.0             5.5
6.0
212          20.0          17.0          20.0            18.0
20.0
213          20.0          18.0          20.0            18.0
20.0

     grade
0     2.50
1     1.75
2     4.50
3     4.50
4     4.50
..     ...
209   2.75
210   3.25
211   5.75
212   5.50
213   5.25

[214 rows x 14 columns]
```

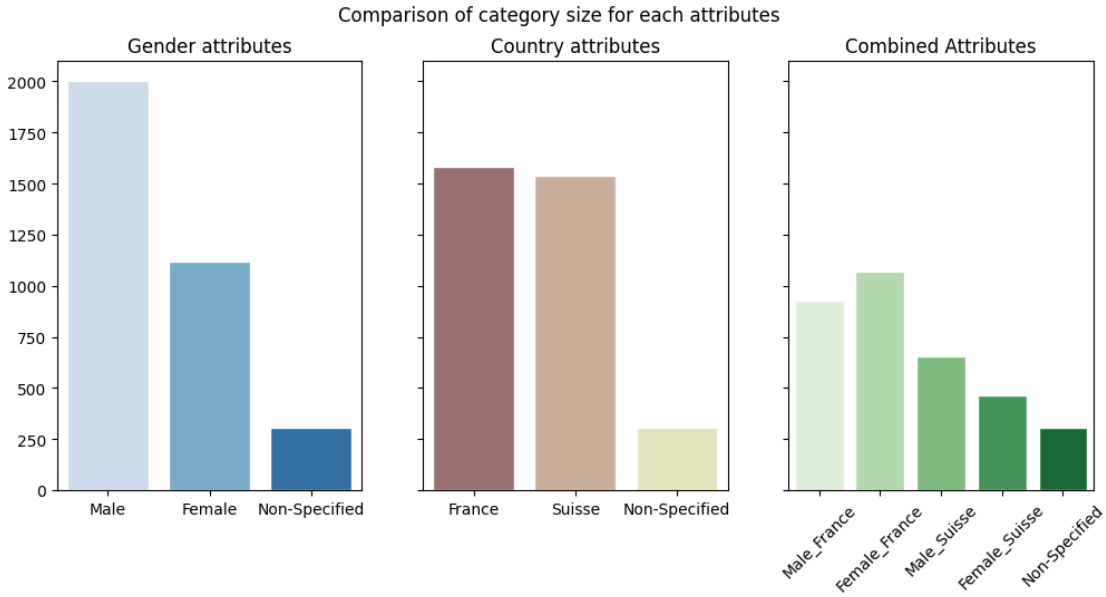
```
demographics['country_diploma'].unique()
```

```
array(['France', nan, 'Suisse'], dtype=object)
```

```
# We've run a BiLSTM model on the data using a 10-fold cross
validation, generating predictions for all 214 students.
predictions = pd.read_csv(DATA_DIR + 'model_predictions.csv')

# convert predictions between [0, 1] to binary variable for pass /
fail {0, 1}
y_pred = [1 if grade < 0.5 else 0 for grade in predictions['grade']]

# Load and process ground truth grades, which are between 0 to 6
# Recieving a score 4 or higher is passing, so we can convert these
grades to a binary pass/fail variable {0, 1}
y = [1 if grade >= 4 else 0 for grade in demographics['grade']]

demographics.insert(0, 'y', y)
demographics.insert(1, 'y_pred', y_pred)
```

```
Number of data in each caterogies
fig, axs = plt.subplots(1, 3, figsize=(12, 5), sharey=True)

M_size = demographics[demographics['gender'] == 'M'].size
F_size = demographics[demographics['gender'] == 'F'].size

France_size = demographics[demographics['country_diploma'] ==
'France'].size
Suisse_size = demographics[demographics['country_diploma'] ==
'Suisse'].size

France_M_size = demographics[(demographics['country_diploma'] ==
'France') &
                                        (demographics['gender'] ==
'M')].size
Suisse_M_size = demographics[(demographics['country_diploma'] ==
'Suisse') &
                                        (demographics['gender'] ==
'M')].size
France_F_size = demographics[(demographics['country_diploma'] ==
'France') &
                                        (demographics['gender'] ==
'F')].size
Suisse_F_size = demographics[(demographics['country_diploma'] ==
'Suisse') &
                                        (demographics['gender'] ==
'F')].size

none_size =
demographics[(demographics['country_diploma'].isna())].size

gender_cat = ['Male', 'Female', 'Non-Specified']
sns.barplot(x=gender_cat, y=[M_size, F_size, none_size], palette =
'Blues', edgecolor = 'w', ax=axs[0])
axs[0].set_title("Gender attributes")

country_cat = ['France', 'Suisse', 'Non-Specified']
sns.barplot(x=country_cat, y=[France_size, Suisse_size, none_size],
palette = 'pink', edgecolor = 'w', ax=axs[1])
axs[1].set_title("Country attributes")

gender_country_cat = ['Male_France', 'Female_France', 'Male_Suisse',
'Female_Suisse', 'Non-Specified']
sns.barplot(x=gender_country_cat, y=[France_M_size, Suisse_M_size,
France_F_size, Suisse_F_size, none_size], palette = 'Greens',
edgecolor = 'w', ax=axs[2])
axs[2].set_title("Combined Attributes")

fig.suptitle("Comparison of category size for each attributes")
```

```
plt.xticks(rotation=45)
plt.show()
```



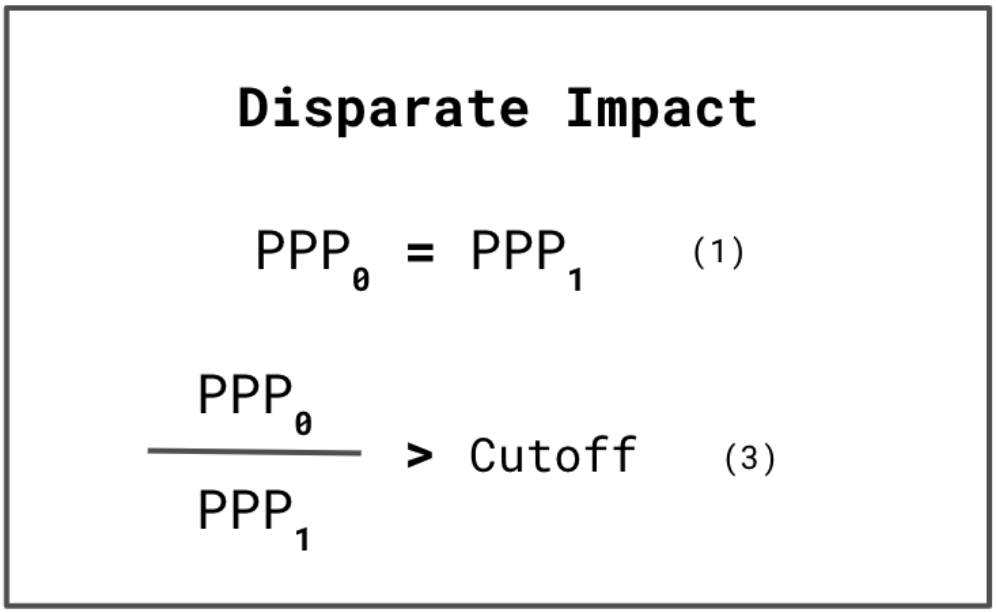
Measuring Fairness

Now, we will move into analyzing methods to measure fairness.

Accuracy is not an ideal measure of fairness. We can base the accuracy calculation on the confusion matrix below. This is a standard confusion matrix used to compare model predictions to the actual target variable. Here Y=1 is a positive prediction (student passes the course) and Y=0 is a negative prediction (student fails the course). We will also be referring back to this matrix when we calculate the other fairness metrics.

$$\%\ \text{predicted as positive (PPP)} = \frac{TP + FP}{N}$$

We can use the confusion matrix to calculate accuracy. Accuracy is the number of true negatives and true positives over the total number of observations. In other words, accuracy is the percentage of correct predictions.

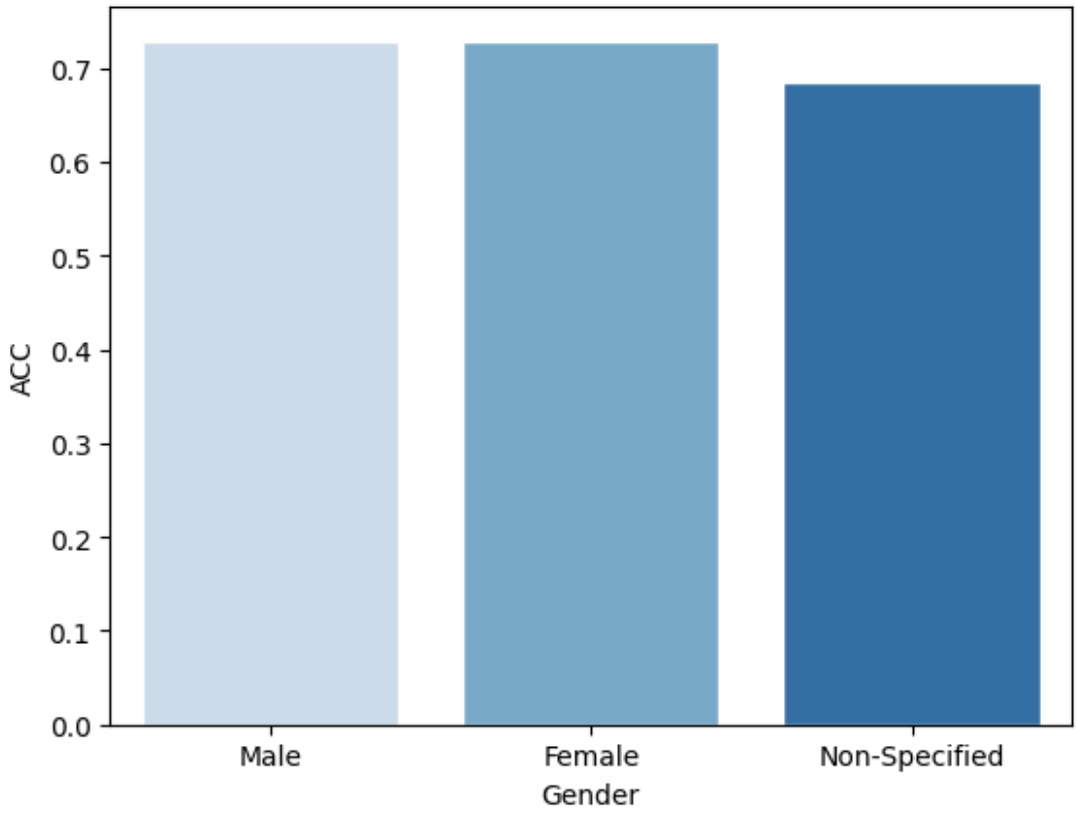



```
def accuracy(df):
    """Calculate accuracy through the confusion matrix."""

    # Confusion Matrix
    cm = confusion_matrix(df['y'],df['y_pred'])
    TN, FP, FN, TP = cm.ravel()

    # Total population
    N = TP + FP + FN + TN

    # Accuracy
    ACC = (TP + TN) / N

    return ACC

print("Overall Accuracy:", np.round(accuracy(demographics), 3))
```

```
Overall Accuracy: 0.724
```

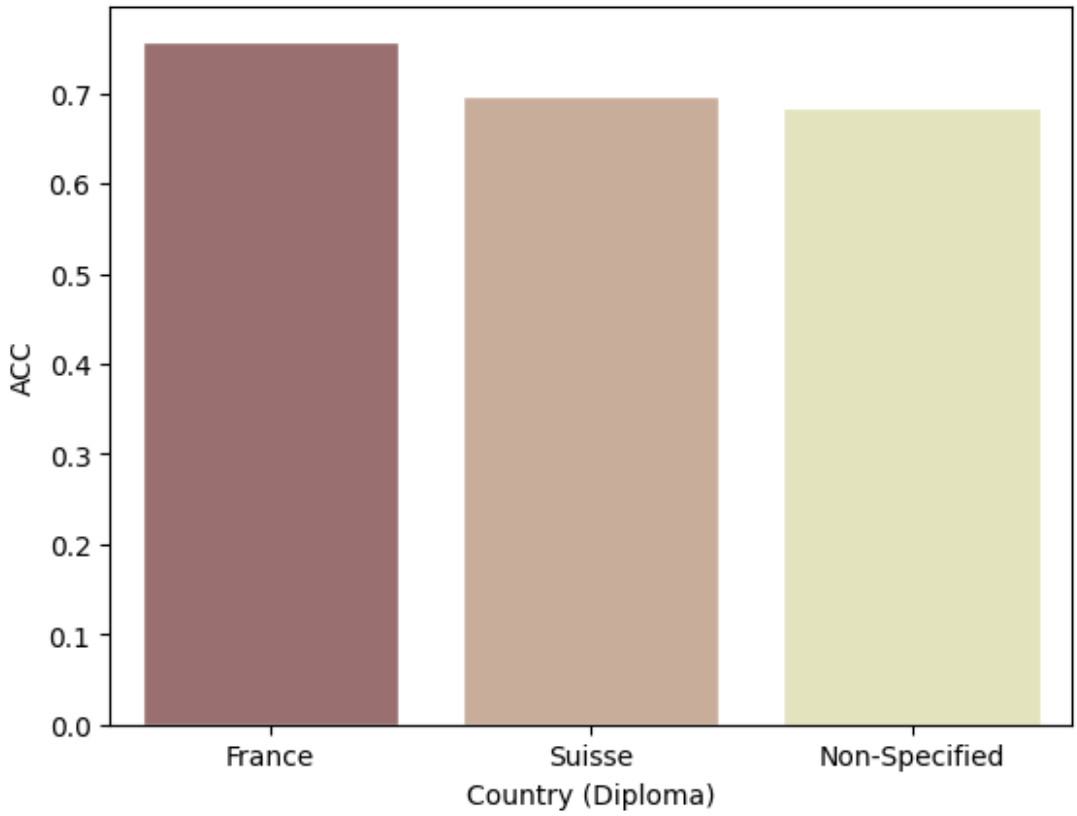
```
gender_df = pd.DataFrame()
gender_df['Gender'] = ['Male', 'Female', 'Non-Specified']
gender_df['ACC'] =
[np.round(accuracy(demographics[demographics['gender'] == 'M']), 3),

np.round(accuracy(demographics[demographics['gender'] == 'F']), 3),

np.round(accuracy(demographics[demographics['gender'].isna()]), 3)]
```

```
sns.barplot(x='Gender', y='ACC', data=gender_df, palette = 'Blues',
edgecolor = 'w')
plt.show()
```



We can see that Male and Female have the same accuracy. The accuracy seems to not depend on the gender.

```
country_df = pd.DataFrame()
country_df['Country (Diploma)'] = ['France', 'Suisse', 'Non-
Specified']
country_df['ACC'] =
[np.round(accuracy(demographics[demographics['country_diploma'] ==
'France']), 3),

np.round(accuracy(demographics[demographics['country_diploma'] ==
'Suisse']), 3),

np.round(accuracy(demographics[demographics['country_diploma'].isna()]
), 3)]


sns.barplot(x='Country (Diploma)', y='ACC', data=country_df, palette =
'pink', edgecolor = 'w')
plt.show()
```

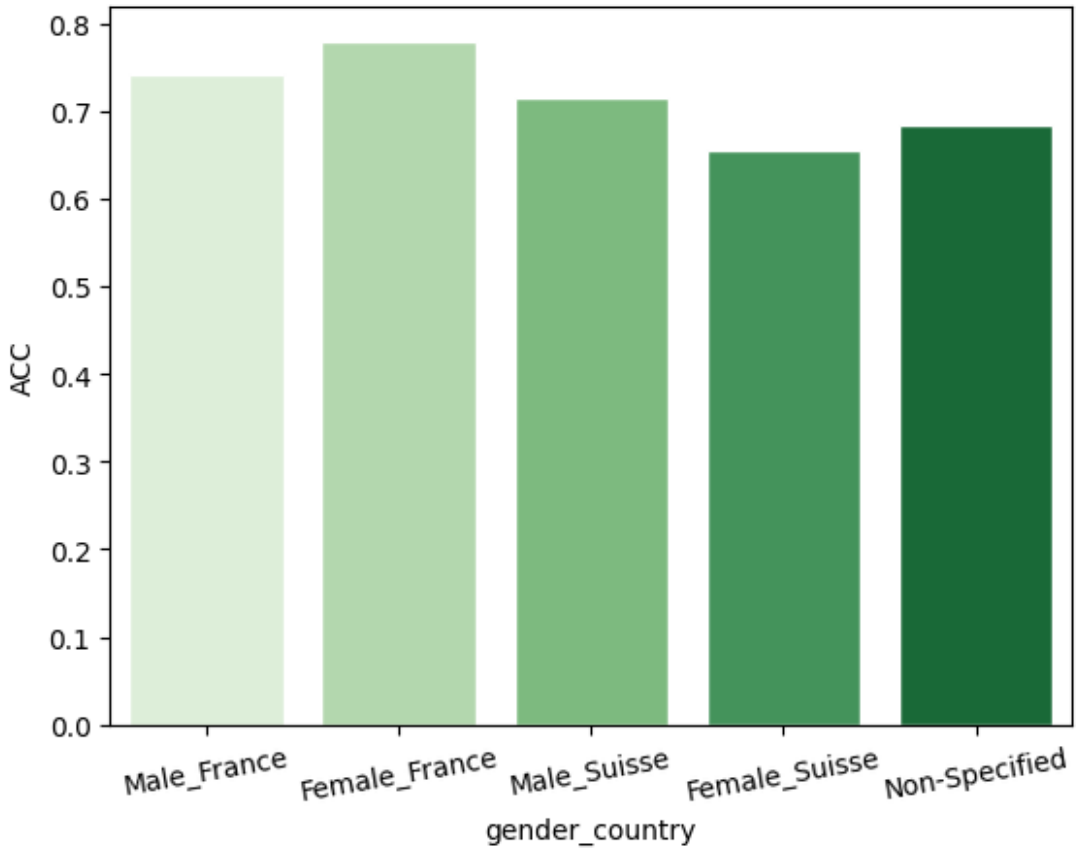


We see a slightly better accuracy when predicting student from France than from Switzerland.

Combined Attributes

```
combined_df = pd.DataFrame()
combined_df["gender_country"] = ['Male_France', 'Female_France',
'Male_Suisse', 'Female_Suisse', 'Non-Specified']

combined_df['ACC'] = [

np.round(accuracy(demographics[(demographics['country_diploma'] ==
'France') &

(demographics['gender'] == 'M')]), 3),

np.round(accuracy(demographics[(demographics['country_diploma'] ==
'France') &

(demographics['gender'] == 'F')]), 3),

np.round(accuracy(demographics[(demographics['country_diploma'] ==
'Suisse') &

(demographics['gender'] == 'M')]), 3),
```

```
np.round(accuracy(demographics[(demographics['country_diploma'] ==
'Suisse') &

(demographics['gender'] == 'F')]), 3),

np.round(accuracy(demographics[demographics['country_diploma'].isna()]
), 3),
                    ]

sns.barplot(x='gender_country', y='ACC', data=combined_df, palette =
'Greens', edgecolor = 'w')
plt.xticks(rotation = 10)
plt.show()
```



Interestingly, male from France and Switzerland have similar accuracy, but concerning Female, the difference is more significante. Neverless, as expected from the previous graph, Male and Female from France have both better accuracy than their corresponding category in Switzerland.

Fairness Definition 1: Equal opportunity

To better capture the benefits of a model we can use the true positive rate (TPR). You can see how we calculate TPR below. The denominator is the number of actual positives. The numerator is the number of correctly predicted positives. In other words, TPR is the percentage of actual positives that were correctly predicted as positive.

$$\%\ \text{predicted as positive (PPP)} = \frac{TP + FP}{N}$$

Under **equal opportunity** we consider a model to be fair if the TPRs of the privileged and unprivileged groups are equal. In practice, we will give some leeway for statistic uncertainty. We can require the differences to be less than a certain cutoff (Equation 2). For our analysis, we have taken the ratio. In this case, we require the ratio to be larger than some cutoff (Equation 3). This ensures that the TPR for the unprivileged group is not significantly smaller than for the privileged group.

Disparate Impact

$$PPP_0 = PPP_1 \quad (1)$$

$$\frac{PPP_0}{PPP_1} > \text{Cutoff} \quad (3)$$

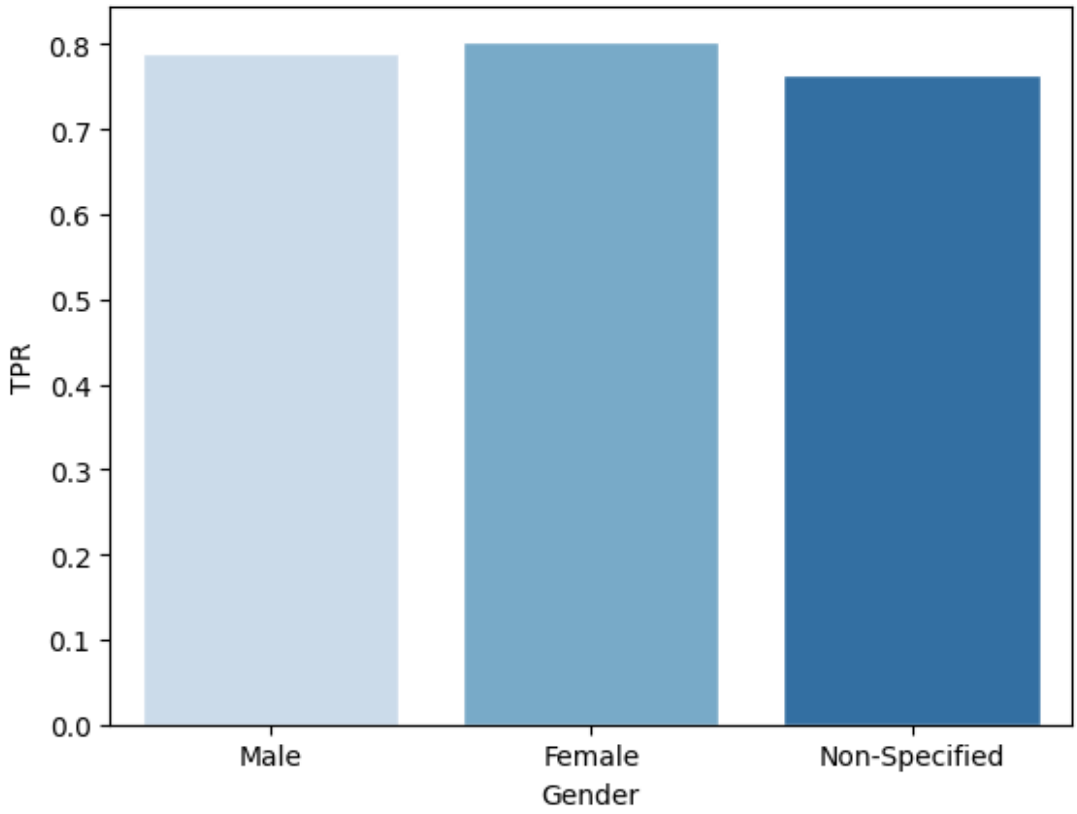
```
def true_positive_rate(df):
    """Calculate equal opportunity (true positive rate)."""
```

```
    # Confusion Matrix
    cm = confusion_matrix(df['y'],df['y_pred'])
    TN, FP, FN, TP = cm.ravel()

    # Total population
    N = TP + FP + FN + TN

    # True positive rate
    TPR = TP / (TP + FN)

    return TPR
print("Overall Equal Opportunity:",
np.round(true_positive_rate(demographics), 3))
```

```
Overall Equal Opportunity: 0.791
```

Sensitive Attribute: Gender

```
gender_df['TPR'] =
[np.round(true_positive_rate(demographics[demographics['gender'] ==
'M']), 3),

np.round(true_positive_rate(demographics[demographics['gender'] ==
'F']), 3),

np.round(true_positive_rate(demographics[demographics['gender'].isna()
]), 3)]

sns.barplot(x='Gender', y='TPR', data=gender_df, palette = 'Blues',
edgecolor = 'w')
plt.show()
```




```
# For equal opportunity, we directly compare the difference between
TPRs of the sensitive attributes.
# We define our significance cutoff at 0.1, stating any difference
below 10% can be attributed to random chance.

def stats_eq_opp(df, attr, stat='TPR', cutoff=0.1, indexs=[0, 1]):
    TPR_0, TPR_1 = df[stat][indexs[0]], df[stat][indexs[1]]
    equal_opp = np.abs(np.round(TPR_1 - TPR_0, 3))
    equal_opp_ratio = np.round(np.minimum(TPR_0, TPR_1) /
np.maximum(TPR_0, TPR_1), 3)

    print('Sensitive Attr:', attr, '\n')

    print('------------------------------------')
    print('|Equal Opportunity| < Cutoff?', np.abs(equal_opp) > cutoff)
    print('------------------------------------')
    print('TPR0 (', df[attr][indexs[0]], ') =', TPR_0)
    print('TPR1 (', df[attr][indexs[1]], ') =', TPR_1)
    print('Equal Opportunity:', equal_opp)
    print('Cutoff:', cutoff)


    print('\n------------------------------------')
    print('Equal Opportunity Ratio?', equal_opp_ratio)
    print('------------------------------------')
```

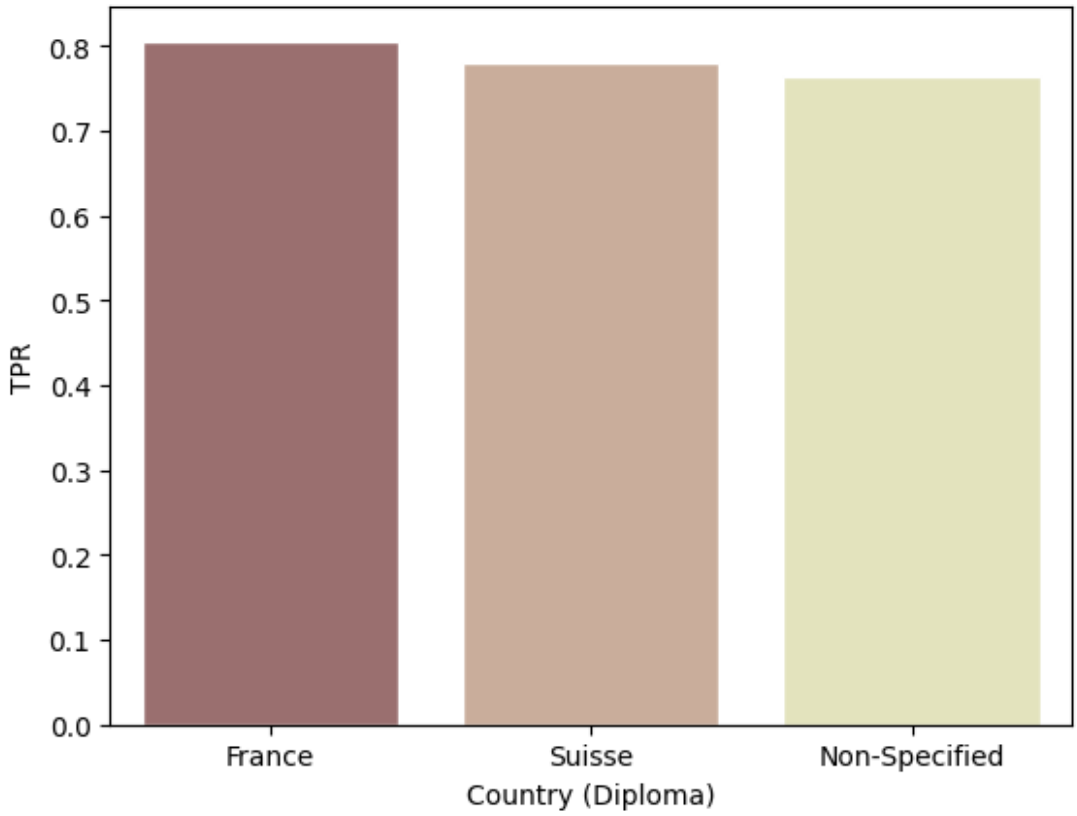
```
stats_eq_opp(gender_df, 'Gender')
```

```
Sensitive Attr: Gender

-------------------------------------
|Equal Opportunity| < Cutoff? False
-------------------------------------
TPR0 ( Male ) = 0.789
TPR1 ( Female ) = 0.804
Equal Opportunity: 0.015
Cutoff: 0.1

-------------------------------------
Equal Opportunity Ratio? 0.981
-------------------------------------
```

We have an Equal Opportunity is smaller than the Cutoff, hence we consider our model as fair. Also the Equal Opportunity Ratio is bigger than the Cutoff and close to 1, meaning that there is no unpriviledge group.

Sensitive Attribute: Country

```
country_df['TPR'] =
[np.round(true_positive_rate(demographics[demographics['country_diplom
a'] == 'France']), 3),

np.round(true_positive_rate(demographics[demographics['country_diploma
'] == 'Suisse']), 3),

np.round(true_positive_rate(demographics[demographics['country_diploma
'].isna()]), 3)]


sns.barplot(x='Country (Diploma)', y='TPR', data=country_df, palette =
'pink', edgecolor = 'w')
plt.show()
```




```
stats_eq_opp(country_df, 'Country (Diploma)')
```

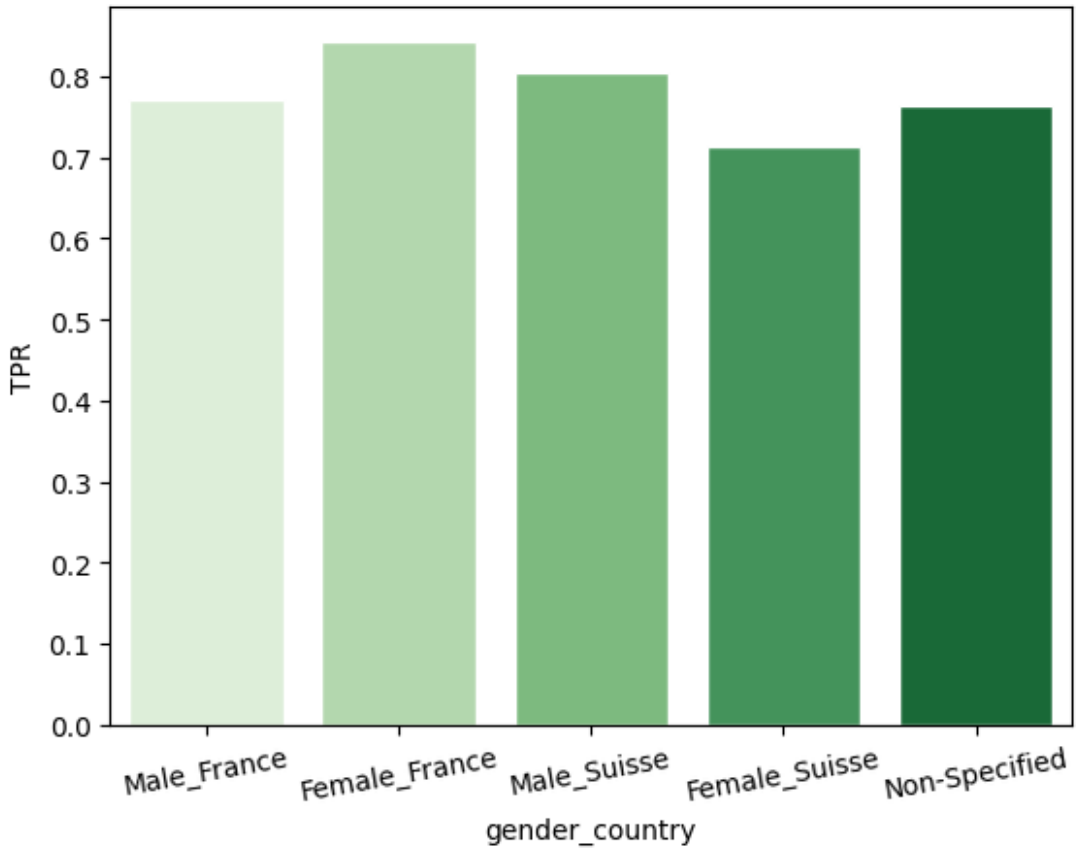
```
Sensitive Attr: Country (Diploma)

-------------------------------------
|Equal Opportunity| < Cutoff? False
-------------------------------------
TPR0 ( France ) = 0.806
TPR1 ( Suisse ) = 0.78
Equal Opportunity: 0.026
Cutoff: 0.1

-------------------------------------
Equal Opportunity Ratio? 0.968
-------------------------------------
```

We have an Equal Opportunity is smaller than the Cutoff, hence we consider our model as fair. Also the Equal Opportunity Ratio is bigger than the Cutoff and close to 1, meaning that there is no unpriviledge group.

Combined Attributes

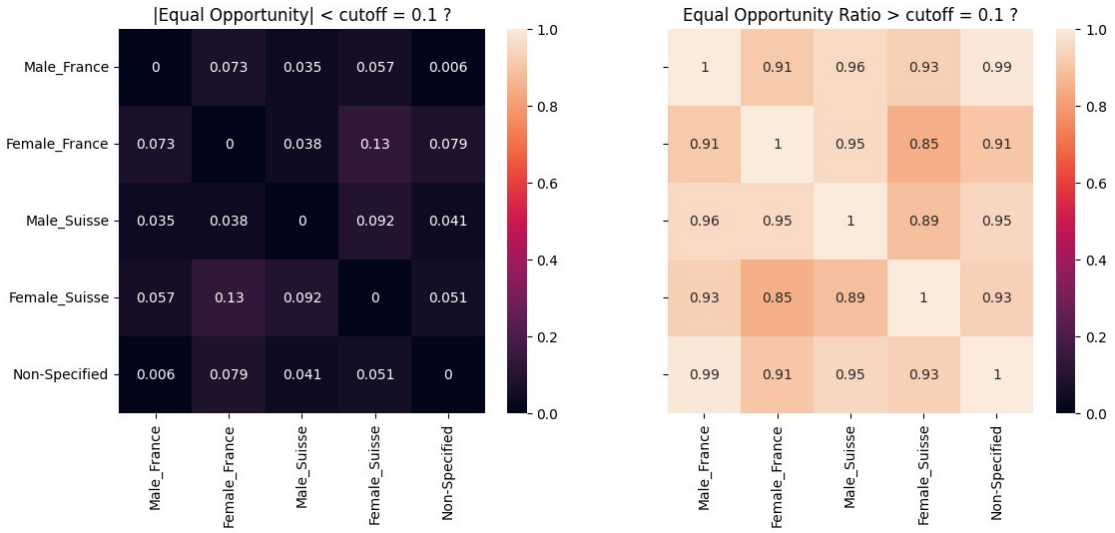
```
combined_df['TPR'] = [

np.round(true_positive_rate(demographics[(demographics['country_diplom
a'] == 'France') &
```

```
(demographics['gender'] == 'M')]), 3),

np.round(true_positive_rate(demographics[(demographics['country_diplom
a'] == 'France') &

(demographics['gender'] == 'F')]), 3),

np.round(true_positive_rate(demographics[(demographics['country_diplom
a'] == 'Suisse') &

(demographics['gender'] == 'M')]), 3),

np.round(true_positive_rate(demographics[(demographics['country_diplom
a'] == 'Suisse') &

(demographics['gender'] == 'F')]), 3),

np.round(true_positive_rate(demographics[demographics['country_diploma
'].isna()]), 3),
                    ]

sns.barplot(x='gender_country', y='TPR', data=combined_df, palette =
'Greens', edgecolor = 'w')
plt.xticks(rotation = 10)
plt.show()
```




```
def get_heatmap(df, attr, func, stat='TPR', cutoff=0.1):
    size = df[attr].size
    data = df[stat]
    heatmap = np.zeros((size, size))
    for i in range(size):
        for j in range(size):
            heatmap[i, j] = func([data[i], data[j]])

    return heatmap

fig, ax = plt.subplots(1, 2, figsize=(13, 5), sharey=True)

heatmap = get_heatmap(combined_df, 'gender_country', lambda x:
np.round(np.abs(x[0] - x[1]), 3))
sns.heatmap(heatmap, ax=ax[0],
xticklabels=combined_df['gender_country'],
yticklabels=combined_df['gender_country'], annot=True, vmin=0, vmax=1)
ax[0].set_title("|Equal Opportunity| < cutoff = 0.1 ?")

heatmap = get_heatmap(combined_df, 'gender_country', lambda x:
np.round(np.minimum(x[0], x[1]) / np.maximum(x[0], x[1]), 3))
sns.heatmap(heatmap, ax=ax[1],
xticklabels=combined_df['gender_country'],
yticklabels=combined_df['gender_country'], annot=True, vmin=0, vmax=1)
```

```
ax[1].set_title("Equal Opportunity Ratio > cutoff = 0.1 ?")

plt.show()
```



False negative rate (FNR)

In some cases, you may want to capture the negative consequences of a model. In FNR, the denominator gives the number of actual positives. Except now we have the number of incorrectly predicted negatives as the numerator. In other words, the FNR is the percentage of actual positives incorrectly predicted as negative.

The FNR can be interpreted as the percentage of people who have wrongfully not benefitted from the model.

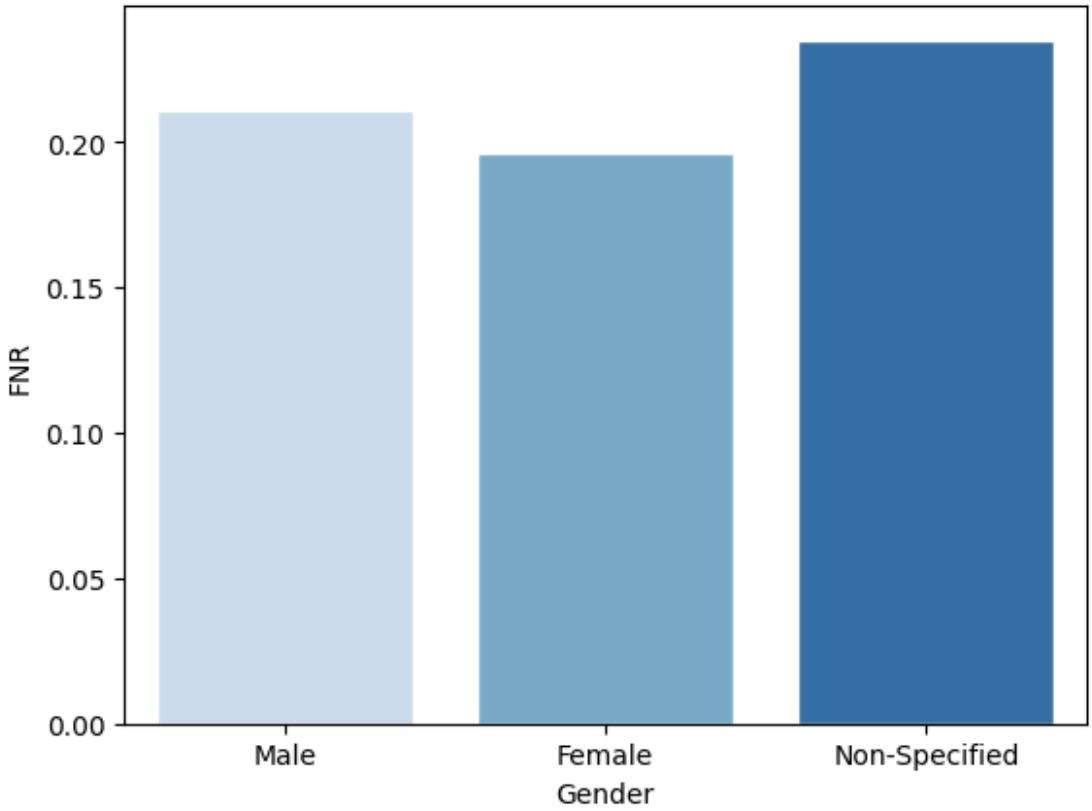
$$\%\ \text{predicted as positive (PPP)} = \frac{TP + FP}{N}$$

```
def false_negative_rate(df):
    """Calculate false negative rate"""

    # Confusion Matrix
    cm = confusion_matrix(df['y'],df['y_pred'])
    TN, FP, FN, TP = cm.ravel()

    # False negative rate
```

```
    FNR = FN / (TP + FN)

    return FNR
print("Overall FNR:", np.round(false_negative_rate(demographics), 3))
```

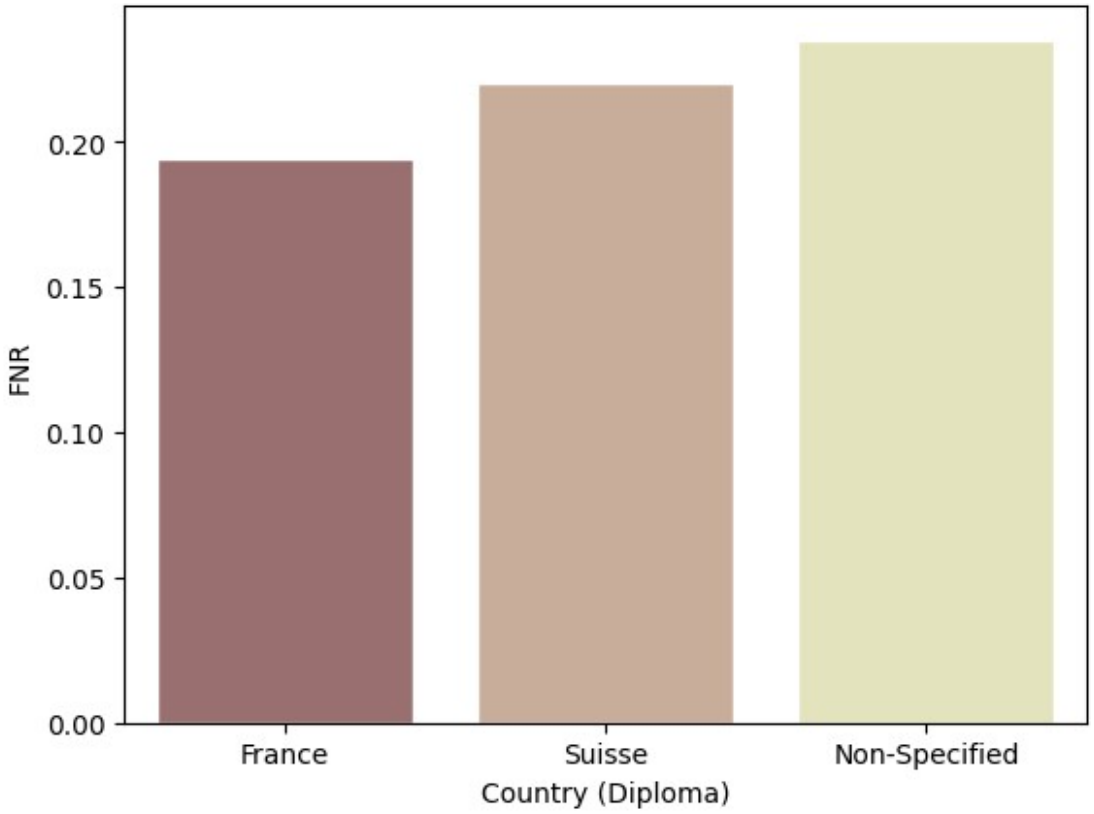
```
Overall FNR: 0.209
```

```
gender_df['FNR'] =
[np.round(false_negative_rate(demographics[demographics['gender'] ==
'M']), 3),

np.round(false_negative_rate(demographics[demographics['gender'] ==
'F']), 3),

np.round(false_negative_rate(demographics[demographics['gender'].isna(
)]), 3)]

sns.barplot(x='Gender', y='FNR', data=gender_df, palette = 'Blues',
edgecolor = 'w')
plt.show()
```




```
country_df['FNR'] =
[np.round(false_negative_rate(demographics[demographics['country_diplo
ma'] == 'France']), 3),
```

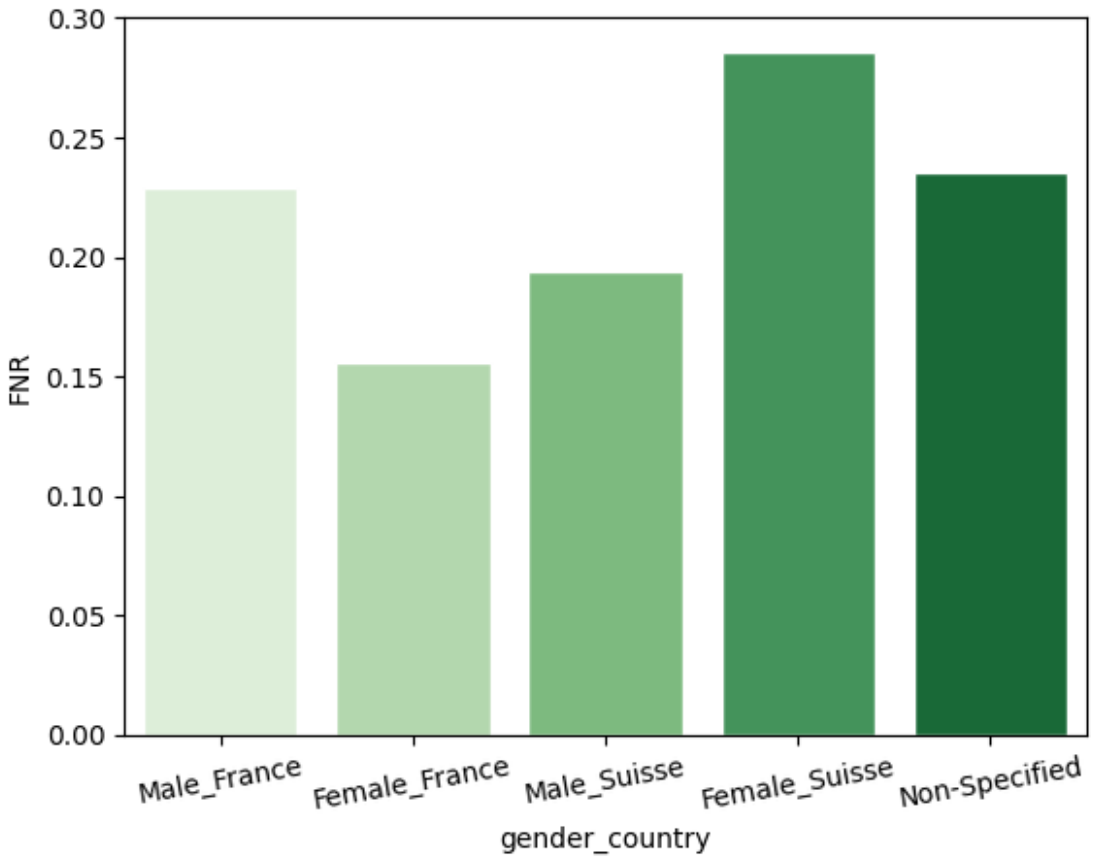
```
np.round(false_negative_rate(demographics[demographics['country_diplom
a'] == 'Suisse']), 3),

np.round(false_negative_rate(demographics[demographics['country_diplom
a'].isna()]), 3)]


sns.barplot(x='Country (Diploma)', y='FNR', data=country_df, palette =
'pink', edgecolor = 'w')
plt.show()
```



Combined Attributes

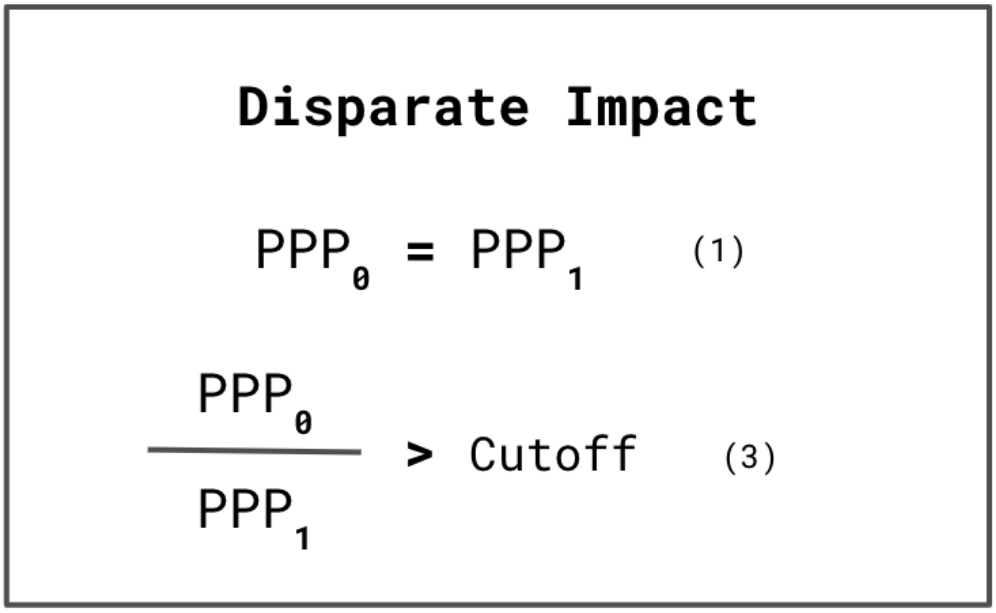
```
combined_df['FNR'] = [

np.round(false_negative_rate(demographics[(demographics['country_diplo
ma'] == 'France') &

(demographics['gender'] == 'M')]), 3),

np.round(false_negative_rate(demographics[(demographics['country_diplo
ma'] == 'France') &

(demographics['gender'] == 'F')]), 3),
```

```
np.round(false_negative_rate(demographics[(demographics['country_diplo
ma'] == 'Suisse') &

(demographics['gender'] == 'M')]), 3),

np.round(false_negative_rate(demographics[(demographics['country_diplo
ma'] == 'Suisse') &

(demographics['gender'] == 'F')]), 3),

np.round(false_negative_rate(demographics[demographics['country_diplom
a'].isna()]), 3),
                    ]

sns.barplot(x='gender_country', y='FNR', data=combined_df, palette =
'Greens', edgecolor = 'w')
plt.xticks(rotation = 10)
plt.show()
```

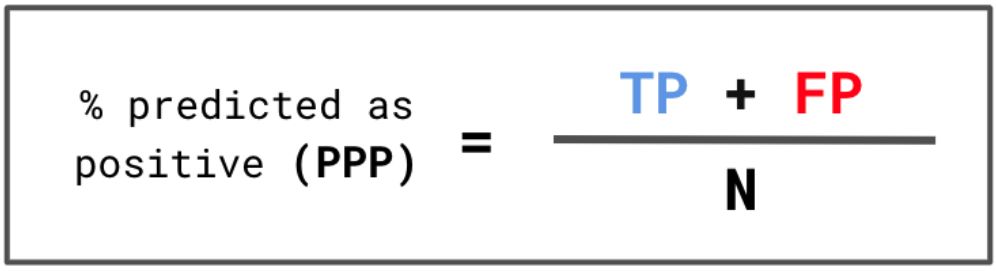


Fairness Definition 2: Equalized Odds

Another way we can capture the benefits of a model is by looking at false positive rates (FPR). As seen in Figure 10, the denominator is the number of actual negatives. This means the TPR is the percentage of actual negatives incorrectly predicted as positive. This can be interpreted as the percentage of people who have wrongfully benefited from the model.



This leads us to the second definition of fairness, **equalized odds**. Like with equal opportunity, this definition requires that the TPRs are equal. Now we also require that the FPRs are equal. This means equalized odds can be thought of as a stricter definition of fairness. It also makes sense that for a model to be fair overall benefit should be equal. That is a similar percentage of the groups should both rightfully and wrongfully benefit.



An advantage of equalized odds is that it does not matter how we define our target variable. Suppose instead we had Y = 0 leads to a benefit. In this case the interpretations of TPR and FPR swap. TPR now captures the wrongful benefit and FPR now captures the rightful benefit. Equalized odds already uses both of these rates so the interpretation

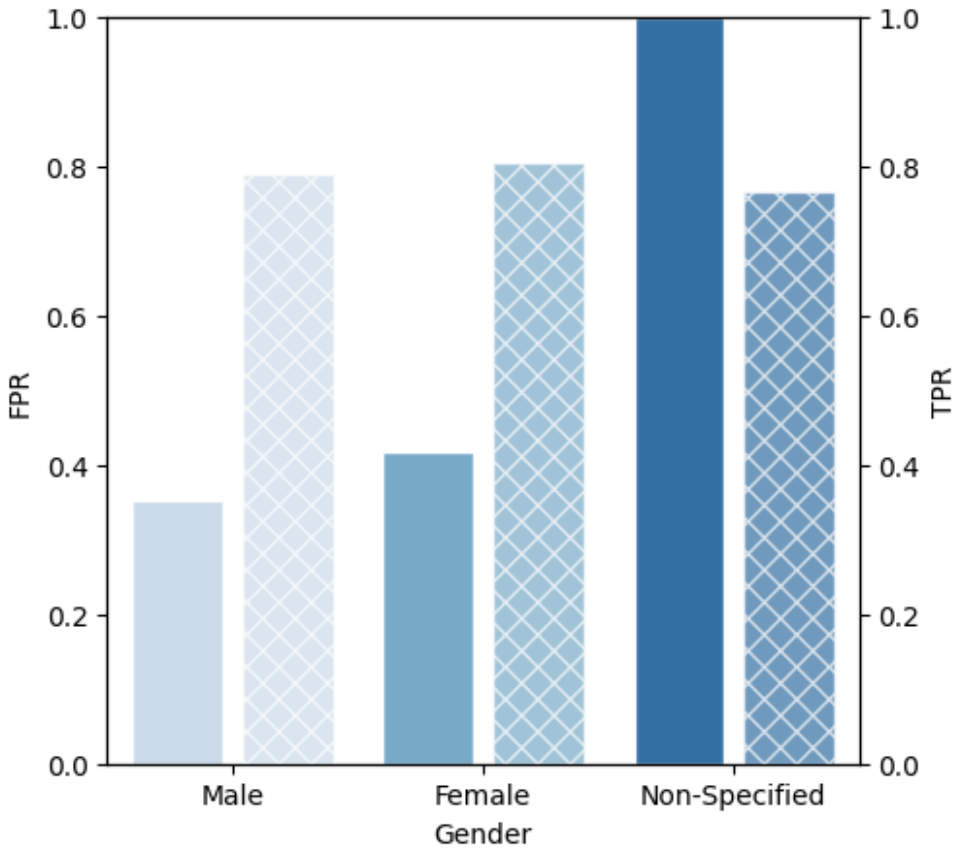
remains the same. In comparison, the interpretation of equal opportunity changes as it only considers TPR.

```
def equalized_odds(df):
    """Calculate FPR and TPR for subgroup of population"""

    # Confusion Matrix
    cm = confusion_matrix(df['y'],df['y_pred'])
    TN, FP, FN, TP = cm.ravel()

    # True positive rate
    TPR = TP / (TP + FN)

    # False positive rate
    FPR = FP / (FP + TN)

    return [TPR, FPR]

equal_odds = equalized_odds(demographics)
print("Overall TPR:", np.round(equal_odds[0], 3))
print("Overall FPR:", np.round(equal_odds[1], 3))
```

```
Overall TPR: 0.791
Overall FPR: 0.388
```

```
gender_df['TPR'] =
[np.round(equalized_odds(demographics[demographics['gender'] == 'M'])
[0], 3),

np.round(equalized_odds(demographics[demographics['gender'] == 'F'])
[0], 3),

np.round(equalized_odds(demographics[demographics['gender'].isna()])
[0], 3)]

gender_df['FPR'] =
[np.round(equalized_odds(demographics[demographics['gender'] == 'M'])
[1], 3),

np.round(equalized_odds(demographics[demographics['gender'] == 'F'])
[1], 3),

np.round(equalized_odds(demographics[demographics['gender'].isna()])
[1], 3)]

plt.figure(figsize=(5, 5))
ax = sns.barplot(x='Gender', y='FPR', data=gender_df, palette =
'Blues', edgecolor = 'w')
width_scale = 0.45
for bar in ax.containers[0]:
```

```
    bar.set_width(bar.get_width() * width_scale)
ax.set_ylim([0, 1])

ax2 = ax.twinx()
sns.barplot(x='Gender', y='TPR', data=gender_df, palette = 'Blues',
edgecolor = 'w', alpha=0.7, hatch='xx', ax=ax2)
for bar in ax2.containers[0]:
    x = bar.get_x()
    w = bar.get_width()
    bar.set_x(x + w * (1- width_scale))
    bar.set_width(w * width_scale)
ax2.set_ylim([0, 1])

plt.show()
```

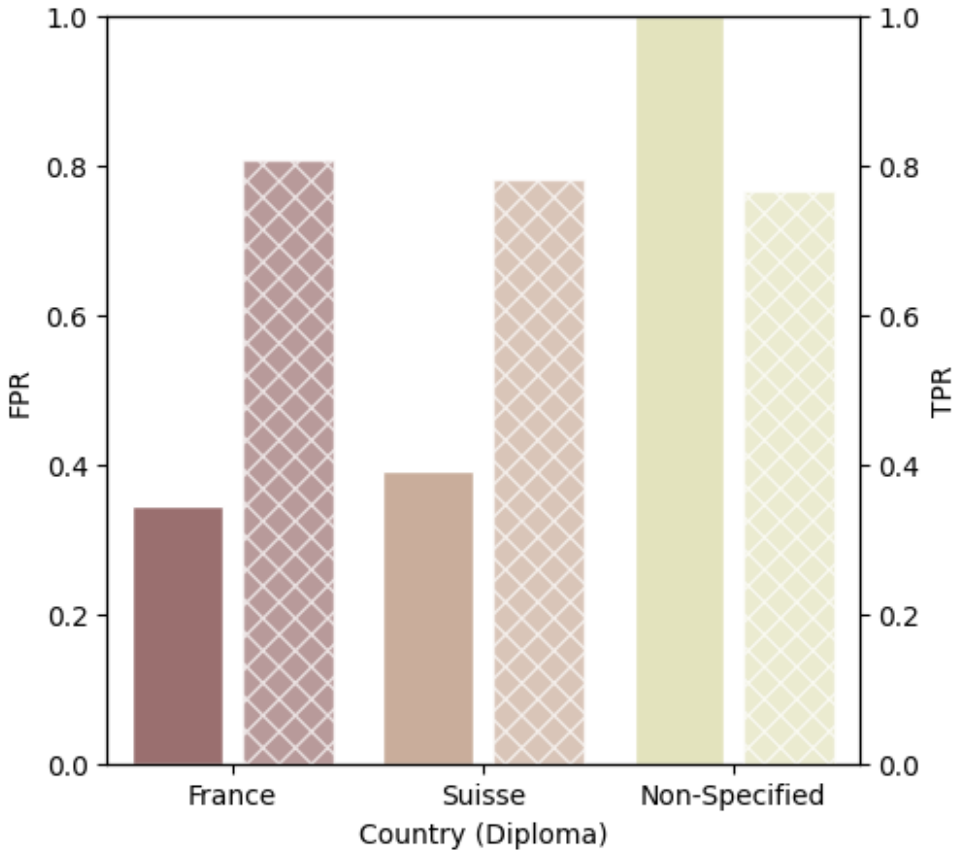



```
gender_df[['Gender', 'TPR', 'FPR']][:-1]
```

	Gender	TPR	FPR
0	Male	0.789	0.352
1	Female	0.804	0.417

```
country_df['TPR'] =
[np.round(equalized_odds(demographics[demographics['country_diploma']
== 'France'])[0], 3),
```

```
np.round(equalized_odds(demographics[demographics['country_diploma']
== 'Suisse'])[0], 3),

np.round(equalized_odds(demographics[demographics['country_diploma'].i
sna()])[0], 3)]
country_df['FPR'] =
[np.round(equalized_odds(demographics[demographics['country_diploma']
== 'France'])[1], 3),

np.round(equalized_odds(demographics[demographics['country_diploma']
== 'Suisse'])[1], 3),

np.round(equalized_odds(demographics[demographics['country_diploma'].i
sna()])[1], 3)]

plt.figure(figsize=(5, 5))
ax = sns.barplot(x='Country (Diploma)', y='FPR', data=country_df,
palette = 'pink', edgecolor = 'w')
width_scale = 0.45
for bar in ax.containers[0]:
    bar.set_width(bar.get_width() * width_scale)
ax.set_ylim([0, 1])

ax2 = ax.twinx()
sns.barplot(x='Country (Diploma)', y='TPR', data=country_df, palette =
'pink', edgecolor = 'w', alpha=0.7, hatch='xx', ax=ax2)
for bar in ax2.containers[0]:
    x = bar.get_x()
    w = bar.get_width()
    bar.set_x(x + w * (1- width_scale))
    bar.set_width(w * width_scale)
ax2.set_ylim([0, 1])

plt.show()
```




```
country_df[['Country (Diploma)', 'TPR', 'FPR']][:-1]
```

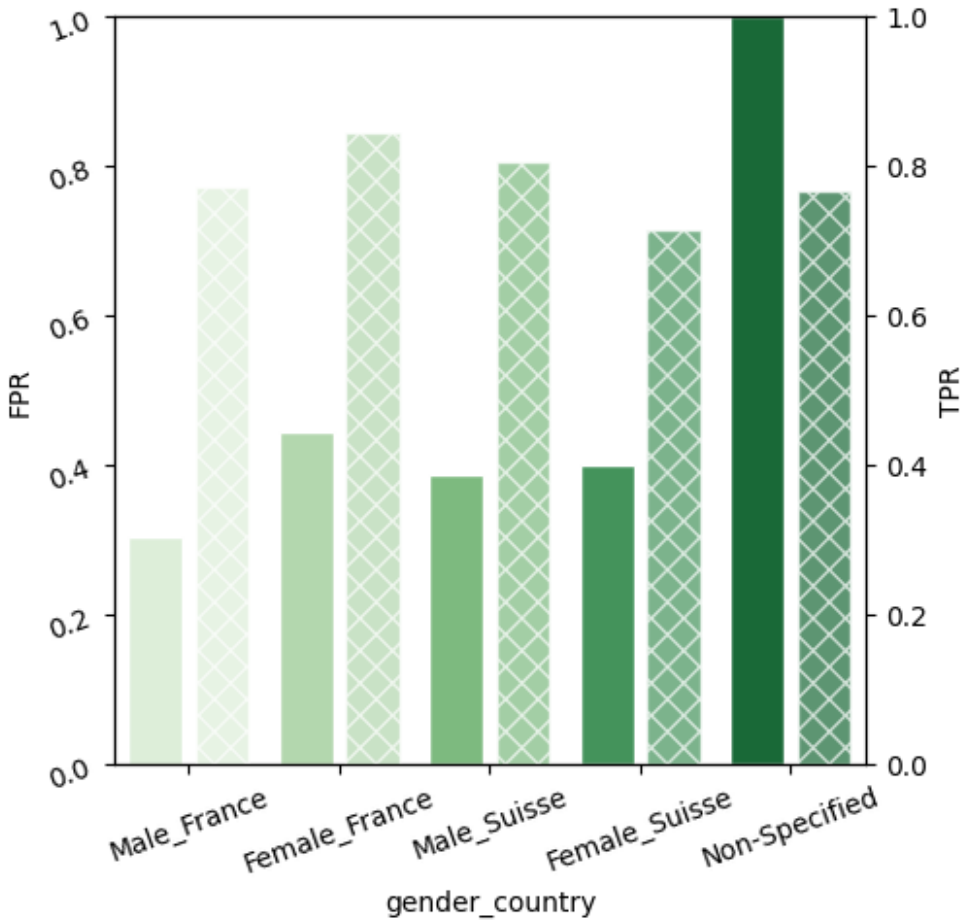
```
  Country (Diploma)    TPR    FPR
0            France  0.806  0.344
1            Suisse  0.780  0.391
```

Combined Attributes

```
combined_df['TPR'] = [

np.round(equalized_odds(demographics[(demographics['country_diploma']
== 'France') &

(demographics['gender'] == 'M')])[0], 3),

np.round(equalized_odds(demographics[(demographics['country_diploma']
== 'France') &

(demographics['gender'] == 'F')])[0], 3),

np.round(equalized_odds(demographics[(demographics['country_diploma']
== 'Suisse') &

(demographics['gender'] == 'M')])[0], 3),
```

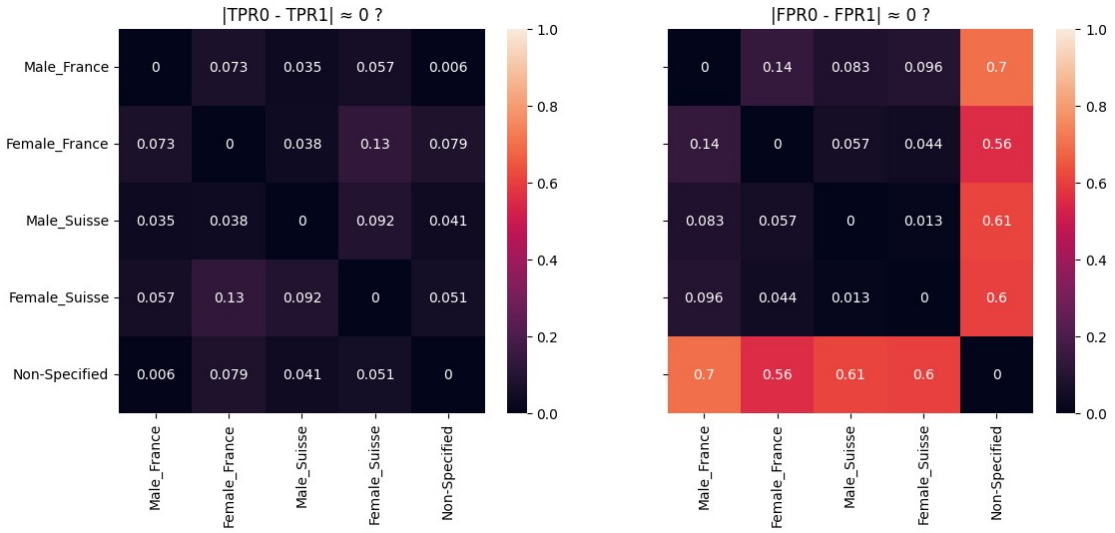
```
np.round(equalized_odds(demographics[(demographics['country_diploma']
== 'Suisse') &

(demographics['gender'] == 'F')])[0], 3),

np.round(equalized_odds(demographics[demographics['country_diploma'].i
sna()])[0], 3),
                    ]

combined_df['FPR'] = [

np.round(equalized_odds(demographics[(demographics['country_diploma']
== 'France') &

(demographics['gender'] == 'M')])[1], 3),

np.round(equalized_odds(demographics[(demographics['country_diploma']
== 'France') &

(demographics['gender'] == 'F')])[1], 3),

np.round(equalized_odds(demographics[(demographics['country_diploma']
== 'Suisse') &

(demographics['gender'] == 'M')])[1], 3),

np.round(equalized_odds(demographics[(demographics['country_diploma']
== 'Suisse') &

(demographics['gender'] == 'F')])[1], 3),

np.round(equalized_odds(demographics[demographics['country_diploma'].i
sna()])[1], 3),
                    ]
plt.figure(figsize=(5, 5))
ax = sns.barplot(x='gender_country', y='FPR', data=combined_df,
palette = 'Greens', edgecolor = 'w')
width_scale = 0.45
for bar in ax.containers[0]:
    bar.set_width(bar.get_width() * width_scale)
ax.set_ylim([0, 1])

ax2 = ax.twinx()
sns.barplot(x='gender_country', y='TPR', data=combined_df, palette =
'Greens', edgecolor = 'w', alpha=0.7, hatch='xx', ax=ax2)
for bar in ax2.containers[0]:
    x = bar.get_x()
    w = bar.get_width()
```

```
    bar.set_x(x + w * (1- width_scale))
    bar.set_width(w * width_scale)
ax2.set_ylim([0, 1])

ax.tick_params(labelrotation=20)
plt.show()
```




```
fig, ax = plt.subplots(1, 2, figsize=(13, 5), sharey=True)

heatmap = get_heatmap(combined_df, 'gender_country', lambda x:
np.round(np.abs(x[0] - x[1]), 3), stat='TPR')
sns.heatmap(heatmap, ax=ax[0],
xticklabels=combined_df['gender_country'],
yticklabels=combined_df['gender_country'], annot=True, vmin=0, vmax=1)
ax[0].set_title("|TPR0 - TPR1| ≈ 0 ?")

heatmap = get_heatmap(combined_df, 'gender_country', lambda x:
np.round(np.abs(x[0] - x[1]), 3), stat='FPR')
sns.heatmap(heatmap, ax=ax[1],
xticklabels=combined_df['gender_country'],
yticklabels=combined_df['gender_country'], annot=True, vmin=0, vmax=1)
ax[1].set_title("|FPR0 - FPR1| ≈ 0 ?")
```

```
plt.show()
```

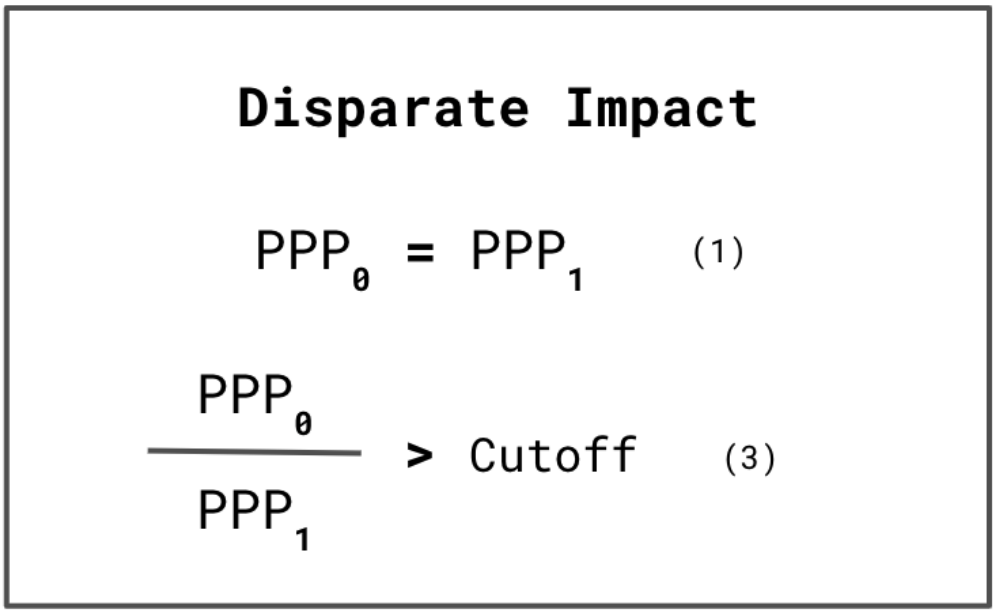


Fairness Definition 3: Disparate Impact

Our third definition of fairness is disparate impact (DI). We start by calculating the PPP rates seen below. This is the percentage of people who have either been correctly (TP) or incorrectly (FP) predicted as positive. We can interpret this as the percentage of people who will benefit from the model.

$$\text{\% predicted as positive (PPP)} = \frac{TP + FP}{N}$$

Under DI we consider a model to be fair if we have equal PPP rates. Again, in practice, we use a cutoff to give some leeway. This definition is supposed to represent the legal concept of disparate impact. In the US there is a legal precedent to set the cutoff to 0.8. That is the PPP for the unprivileged group must not be less than 80% of that of the privileged group.



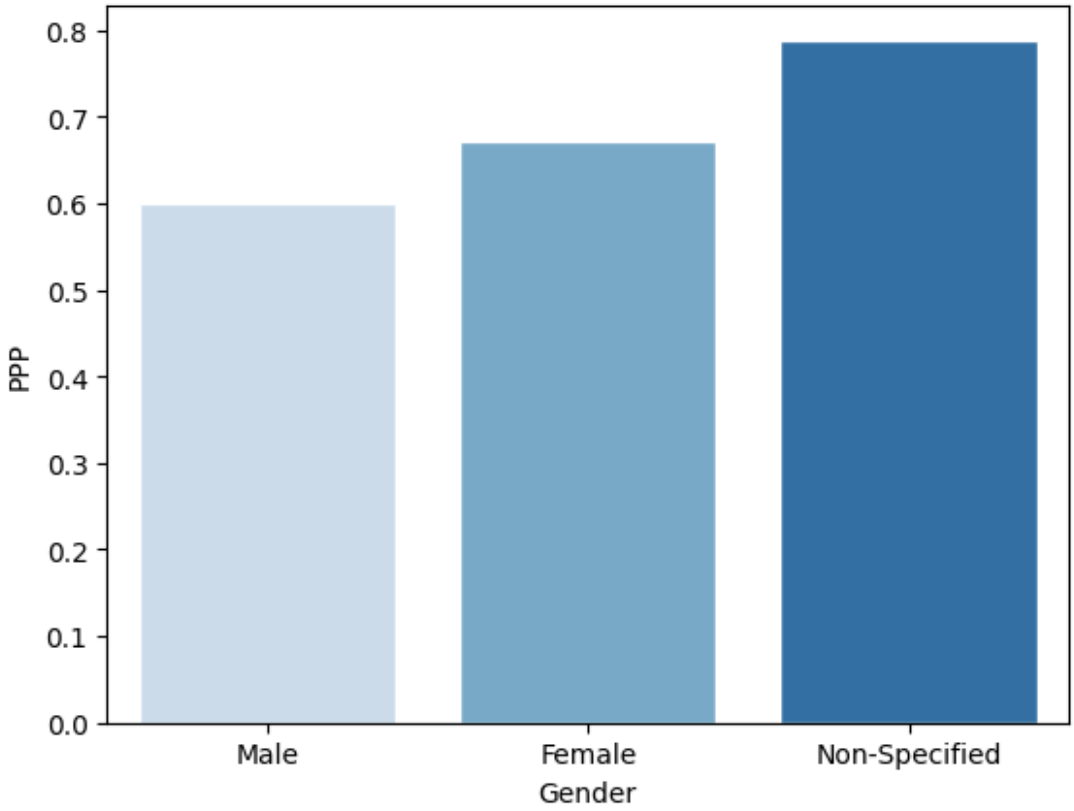

```
def disparate_impact(df):
    """Calculate PPP for subgroup of population"""

    # Confusion Matrix
    cm = confusion_matrix(df['y'],df['y_pred'])
    TN, FP, FN, TP = cm.ravel()

    # Total population
    N = TP + FP + FN + TN

    # predicted as positive
    PPP = (TP + FP) / N

    return PPP

print("Overall PPP:", np.round(disparate_impact(demographics), 3))
```

```
Overall PPP: 0.64
```

Sensitive Attribute: Gender

```
gender_df['PPP'] =
[np.round(disparate_impact(demographics[demographics['gender'] ==
'M']), 3),

np.round(disparate_impact(demographics[demographics['gender'] ==
'F']), 3),

np.round(disparate_impact(demographics[demographics['gender'].isna()])
, 3)]
```

```
sns.barplot(x='Gender', y='PPP', data=gender_df, palette = 'Blues',
edgecolor = 'w')
plt.show()
```



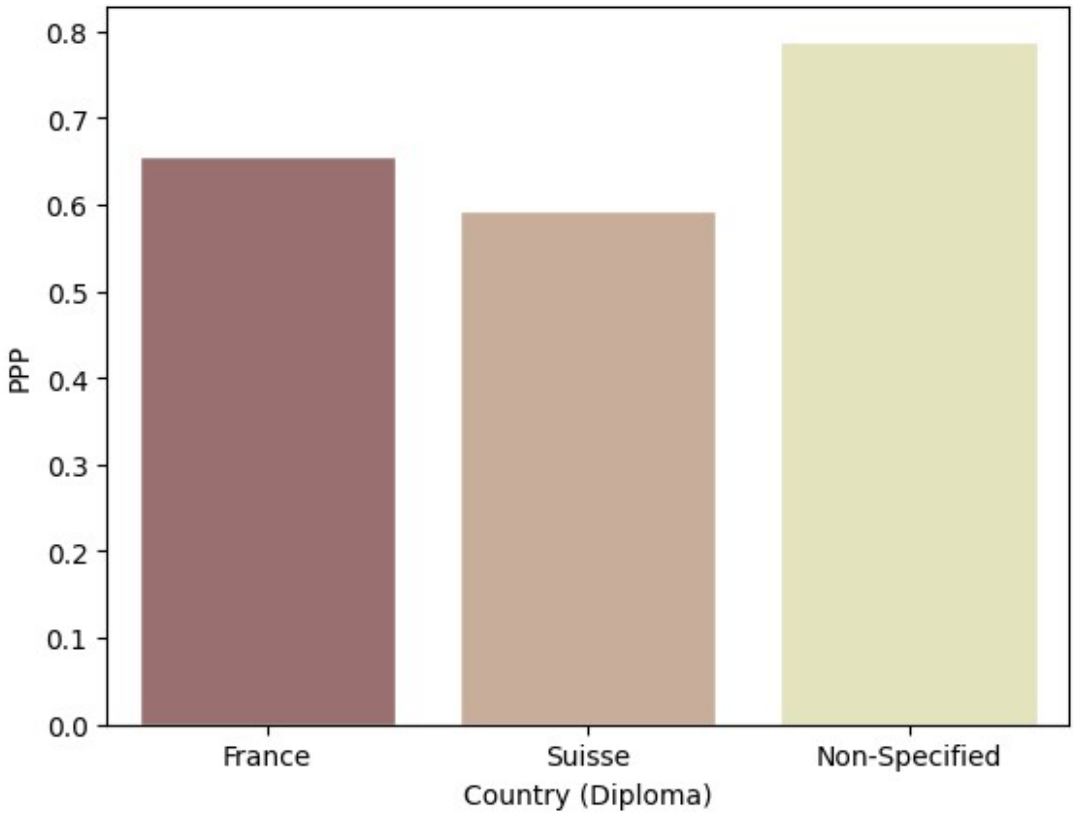

```
# For disparate impact, we compare the ratio between the PPPs of the
sensitive attributes.
# We define our significance cutoff at 0.1, stating any difference
below 10% can be attributed to random chance.

def stats_ppp(df, attr, cutoff=0.1):
    PPP_0, PPP_1 = df['PPP'][0], df['PPP'][1]
    ppp_ratio = np.round(np.minimum(PPP_0, PPP_1) / np.maximum(PPP_0,
PPP_1), 3)

    print('Sensitive Attr:', attr, '\n')

    print('--------------------------------------')
    print('Disparate Impact > Cutoff?', np.abs(ppp_ratio) > cutoff)
    print('--------------------------------------')
    print('PPP0 (', df[attr][0], ') =', PPP_0)
    print('PPP1 (', df[attr][1], ') =', PPP_1)
    print('PPP_Ratio:', ppp_ratio)
    print('Cutoff:', cutoff)
```

```
stats_ppp(gender_df, 'Gender')
```

```
Sensitive Attr: Gender

------------------------------------
Disparate Impact > Cutoff? True
------------------------------------
PPP0 ( Male ) = 0.6
PPP1 ( Female ) = 0.671
PPP_Ratio: 0.894
Cutoff: 0.1
```

Sensitive Attribute: Country (Diploma)

```
country_df['PPP'] =
[np.round(disparate_impact(demographics[demographics['country_diploma'
] == 'France']), 3),

np.round(disparate_impact(demographics[demographics['country_diploma']
== 'Suisse']), 3),

np.round(disparate_impact(demographics[demographics['country_diploma']
.isna()]), 3)]


sns.barplot(x='Country (Diploma)', y='PPP', data=country_df, palette =
'pink', edgecolor = 'w')
plt.show()
```



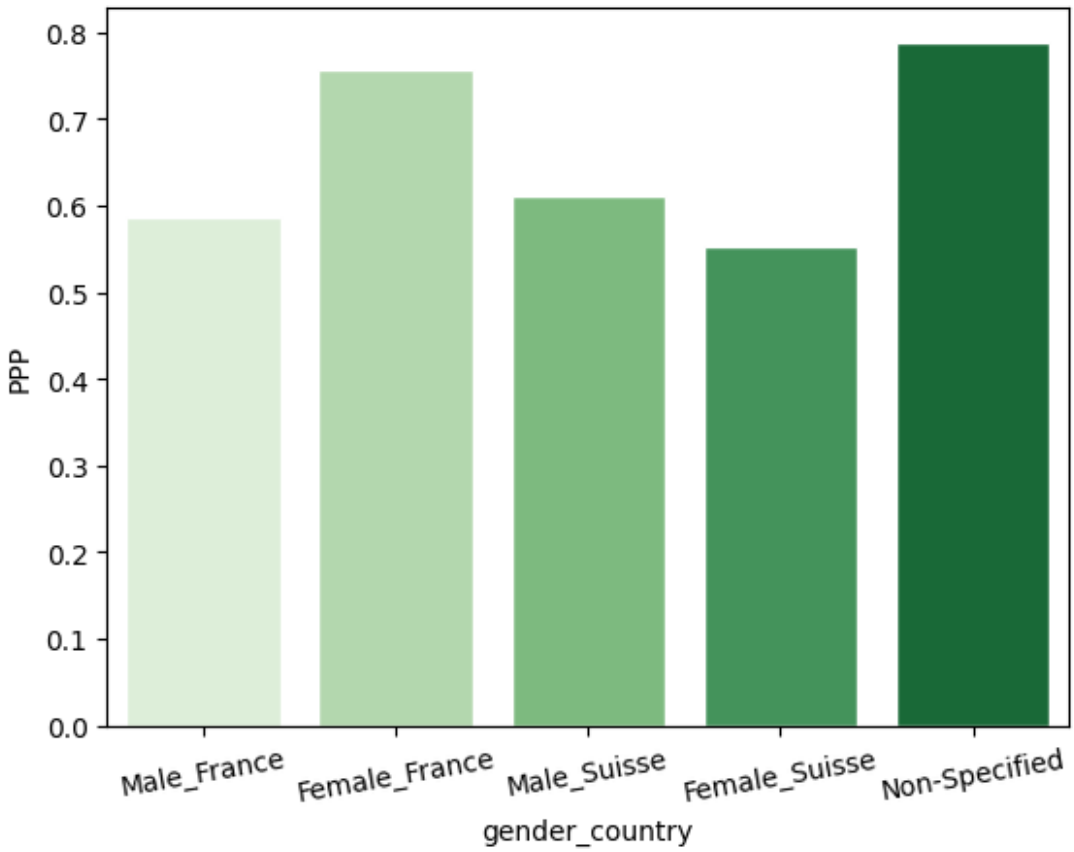

```
stats_ppp(country_df, 'Country (Diploma)')
```

```
Sensitive Attr: Country (Diploma)

-----------------------------------
Disparate Impact > Cutoff? True
-----------------------------------
PPP0 ( France ) = 0.657
PPP1 ( Suisse ) = 0.594
PPP_Ratio: 0.904
Cutoff: 0.1
```

Combined Attributes

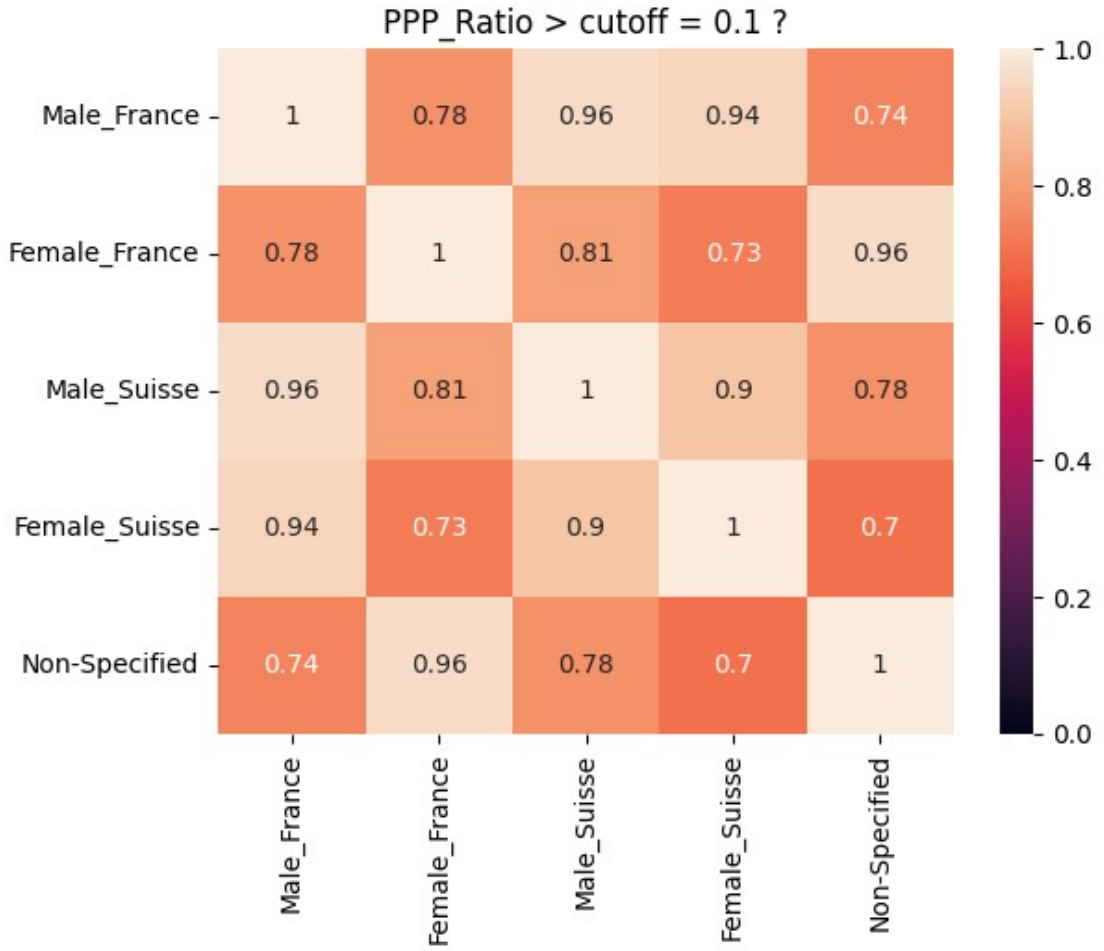
```
combined_df['PPP'] = [

np.round(disparate_impact(demographics[(demographics['country_diploma'
] == 'France') &

(demographics['gender'] == 'M')]), 3),

np.round(disparate_impact(demographics[(demographics['country_diploma'
] == 'France') &

(demographics['gender'] == 'F')]), 3),
```

```
np.round(disparate_impact(demographics[(demographics['country_diploma'
] == 'Suisse') &

(demographics['gender'] == 'M')]), 3),

np.round(disparate_impact(demographics[(demographics['country_diploma'
] == 'Suisse') &

(demographics['gender'] == 'F')]), 3),

np.round(disparate_impact(demographics[demographics['country_diploma']
.isna()]), 3),
                    ]


sns.barplot(x='gender_country', y='PPP', data=combined_df, palette =
'Greens', edgecolor = 'w')
plt.xticks(rotation=10)
plt.show()
```




```
heatmap = get_heatmap(combined_df, 'gender_country', lambda x:
np.round(np.minimum(x[0], x[1]) / np.maximum(x[0], x[1]), 3),
stat='PPP')
sns.heatmap(heatmap, xticklabels=combined_df['gender_country'],
```

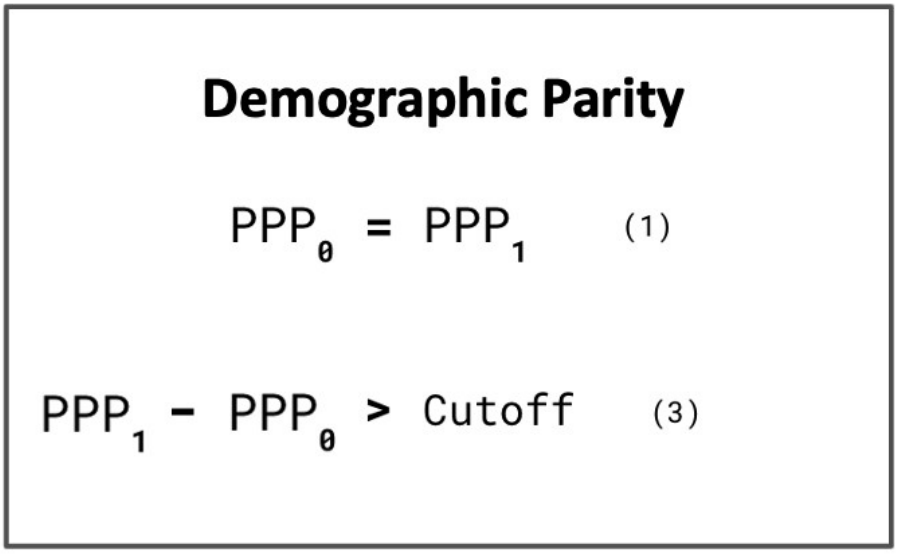
```
yticklabels=combined_df['gender_country'], annot=True, vmin=0, vmax=1)
plt.title("PPP_Ratio > cutoff = 0.1 ?")
plt.show()
```



Fairness Definition 4: Demographic Parity

Demographic Parity states that the proportion of each segment of a protected class (e.g. gender) should receive the positive outcome at equal rates. A positive outcome is the preferred decision, such as in our case, passing a class.

Demographic Parity is very similar to Disparate Impact, with the only difference being that it measures the difference between PPPs instead of the ratio.



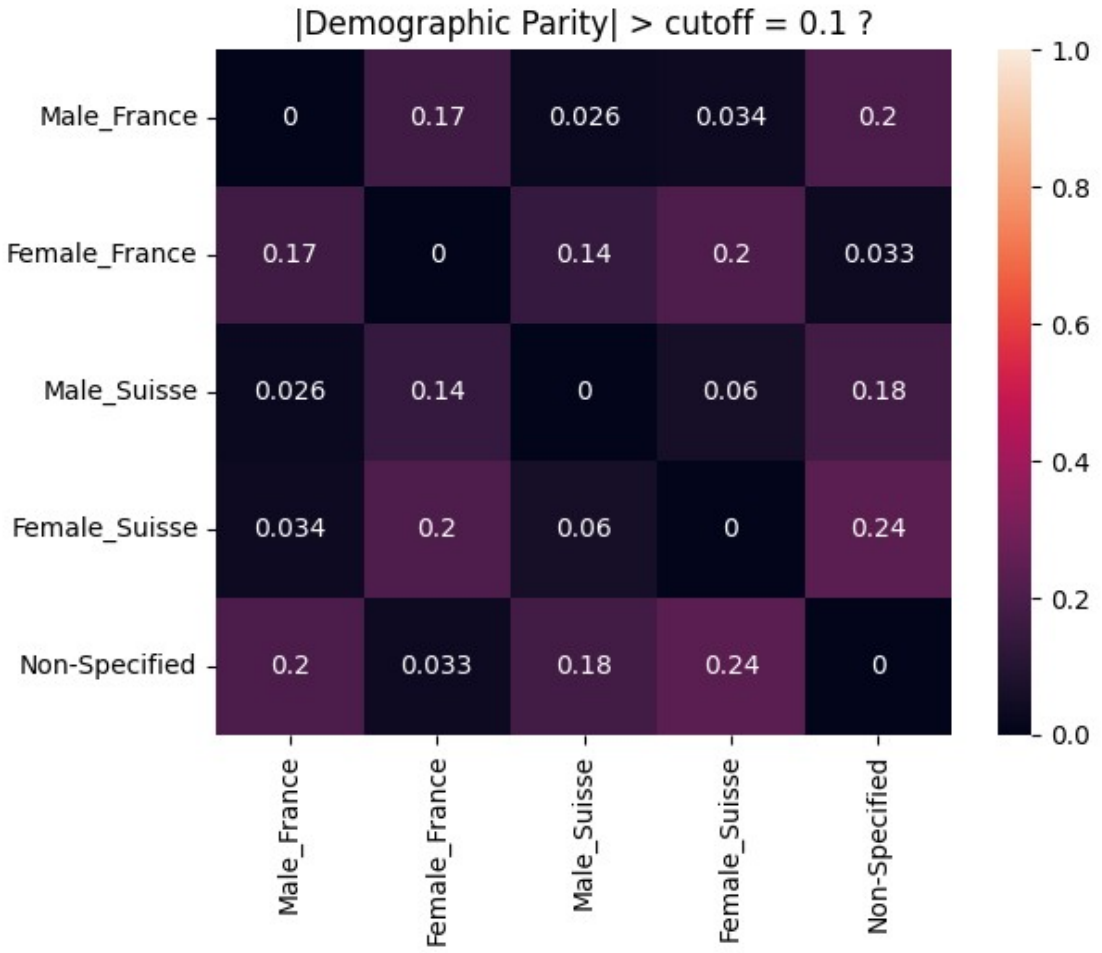

```
# For demographic parity, we compare the difference between the PPPs
of the sensitive attributes.
# We define our significance cutoff at 0.1, stating any difference
below 10% can be attributed to random chance.

def stats_dp(df, attr, cutoff=0.1):
    PPP_0, PPP_1 = df['PPP'][0], df['PPP'][1]
    ppp_diff = np.round(np.abs(PPP_1 - PPP_0), 3)

    print('Sensitive Attr:', attr, '\n')

    print('------------------------------------')
    print('|Demographic Parity| > Cutoff?', np.abs(ppp_diff) > cutoff)
    print('------------------------------------')
    print('PPP0 (', df[attr][0], ') =', PPP_0)
    print('PPP1 (', df[attr][1], ') =', PPP_1)
    print('PPP_Diff:', ppp_diff)
    print('Cutoff:', cutoff)

stats_dp(gender_df, 'Gender')
```

```
Sensitive Attr: Gender

------------------------------------
|Demographic Parity| > Cutoff? False
------------------------------------
PPP0 ( Male ) = 0.6
PPP1 ( Female ) = 0.671
PPP_Diff: 0.071
Cutoff: 0.1
```

```
stats_dp(country_df, 'Country (Diploma)')
```

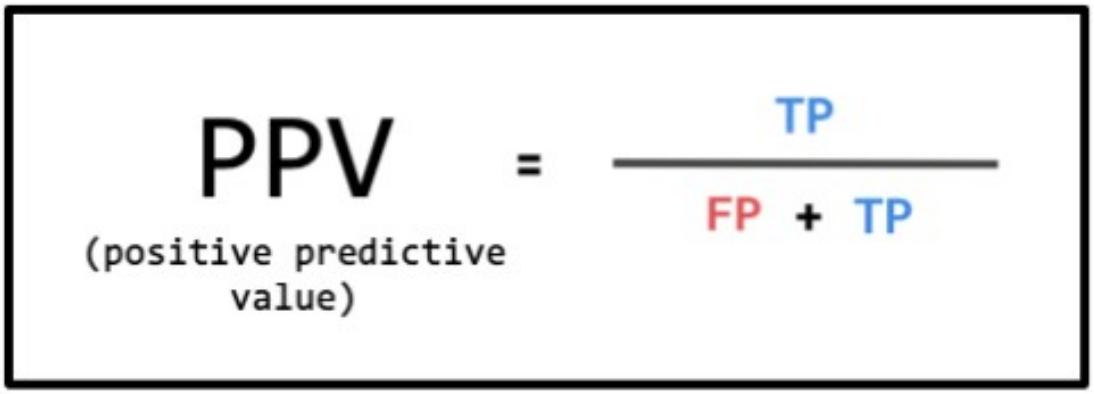
```
Sensitive Attr: Country (Diploma)

-----------------------------------
|Demographic Parity| > Cutoff? False
-----------------------------------
PPP0 ( France ) = 0.657
PPP1 ( Suisse ) = 0.594
PPP_Diff: 0.063
Cutoff: 0.1
```

```
heatmap = get_heatmap(combined_df, 'gender_country', lambda x:
np.round(np.abs(x[0] - x[1]), 3), stat='PPP')
sns.heatmap(heatmap, xticklabels=combined_df['gender_country'],
yticklabels=combined_df['gender_country'], annot=True, vmin=0, vmax=1)
plt.title("|Demographic Parity| > cutoff = 0.1 ?")
plt.show()
```



Fairness Definition 5: Predictive Value Parity

Predictive value-parity equalizes the chance of success, given a positive prediction (PPV) or negative prediction (NPV).



Predictive Rate Parity

PPV = NPV

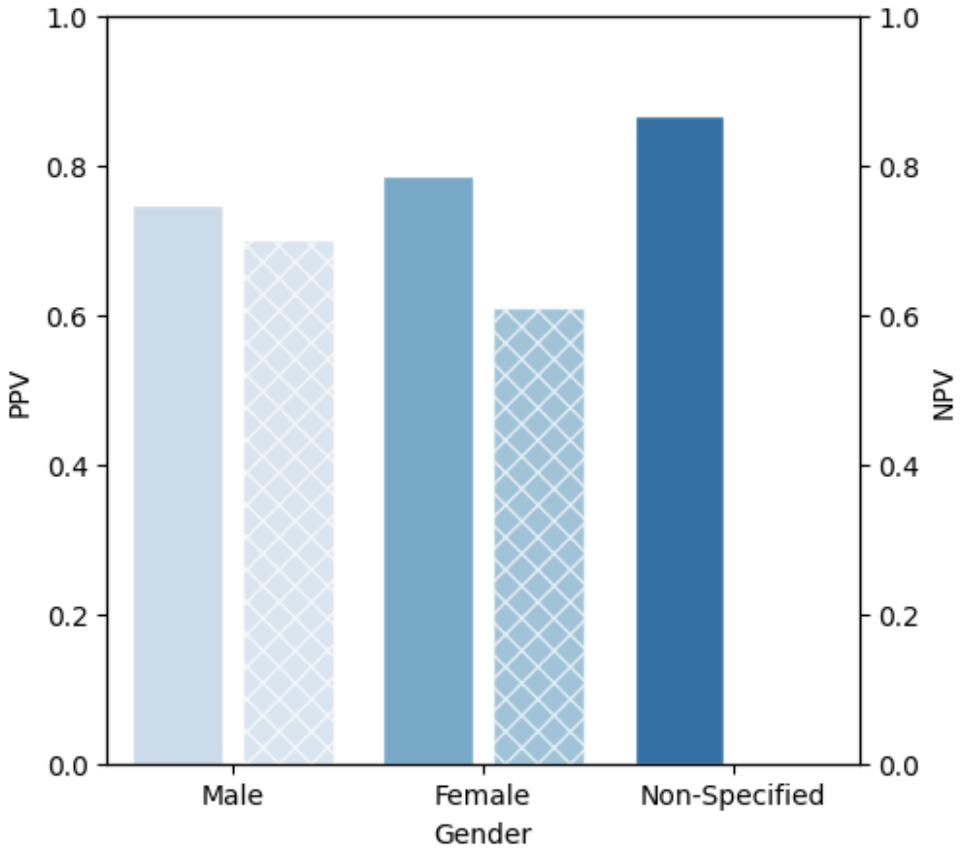
```
def predictive_value_parity(df):
    """Calculate predictive value parity scores"""

    # Confusion Matrix
    cm = confusion_matrix(df['y'],df['y_pred'])
    TN, FP, FN, TP = cm.ravel()

    # Positive Predictive Value
    PPV = TP / (FP + TP)

    # Negative Predictive Value
    NPV = TN / (FN + TN)

    return [PPV, NPV]

print("Overal PPV:", np.round(predictive_value_parity(demographics)
[0], 3))
```

```
print("Overal NPV:", np.round(predictive_value_parity(demographics)
[1], 3))
```

```
Overal PPV: 0.774
Overal NPV: 0.636
```

Sensitive Attribute: Gender

```
gender_df['PPV'] =
[np.round(predictive_value_parity(demographics[demographics['gender']
== 'M'])[0], 3),

np.round(predictive_value_parity(demographics[demographics['gender']
== 'F'])[0], 3),

np.round(predictive_value_parity(demographics[demographics['gender'].i
sna()])[0], 3)]

gender_df['NPV'] =
[np.round(predictive_value_parity(demographics[demographics['gender']
== 'M'])[1], 3),

np.round(predictive_value_parity(demographics[demographics['gender']
== 'F'])[1], 3),

np.round(predictive_value_parity(demographics[demographics['gender'].i
sna()])[1], 3)]

plt.figure(figsize=(5, 5))
ax = sns.barplot(x='Gender', y='PPV', data=gender_df, palette =
'Blues', edgecolor = 'w')
width_scale = 0.45
for bar in ax.containers[0]:
    bar.set_width(bar.get_width() * width_scale)
ax.set_ylim([0, 1])

ax2 = ax.twinx()
sns.barplot(x='Gender', y='NPV', data=gender_df, palette = 'Blues',
edgecolor = 'w', alpha=0.7, hatch='xx', ax=ax2)
for bar in ax2.containers[0]:
    x = bar.get_x()
    w = bar.get_width()
    bar.set_x(x + w * (1- width_scale))
    bar.set_width(w * width_scale)
ax2.set_ylim([0, 1])

plt.show()
```



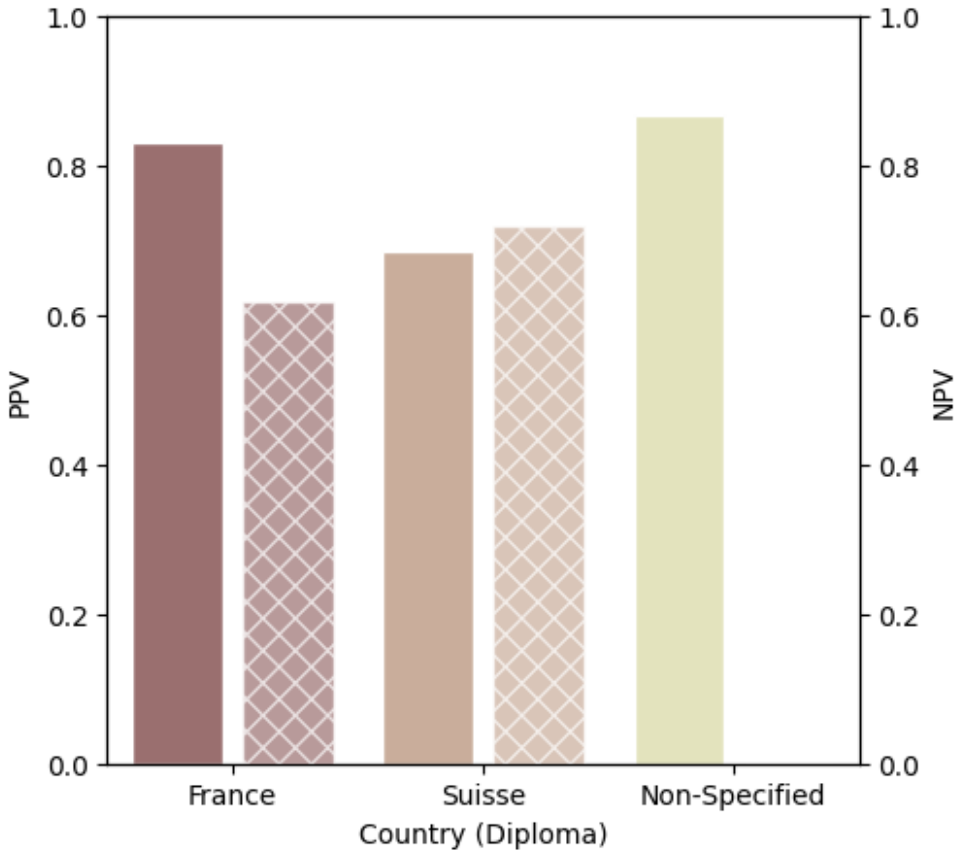

```
gender_df[['Gender', 'PPV', 'NPV']][:-1]
```

```
   Gender    PPV    NPV
0    Male  0.747  0.700
1  Female  0.787  0.609
```

Sensitive Attribute: Country (Diploma)

```
country_df['PPV'] =
[np.round(predictive_value_parity(demographics[demographics['country_d
iploma'] == 'France'])[0], 3),

np.round(predictive_value_parity(demographics[demographics['country_di
ploma'] == 'Suisse'])[0], 3),

np.round(predictive_value_parity(demographics[demographics['country_di
ploma'].isna()])[0], 3)]
country_df['NPV'] =
[np.round(predictive_value_parity(demographics[demographics['country_d
iploma'] == 'France'])[1], 3),

np.round(predictive_value_parity(demographics[demographics['country_di
ploma'] == 'Suisse'])[1], 3),

np.round(predictive_value_parity(demographics[demographics['country_di
```

```
ploma'].isna()])[1], 3)]

plt.figure(figsize=(5, 5))
ax = sns.barplot(x='Country (Diploma)', y='PPV', data=country_df,
palette = 'pink', edgecolor = 'w')
width_scale = 0.45
for bar in ax.containers[0]:
    bar.set_width(bar.get_width() * width_scale)
ax.set_ylim([0, 1])

ax2 = ax.twinx()
sns.barplot(x='Country (Diploma)', y='NPV', data=country_df, palette =
'pink', edgecolor = 'w', alpha=0.7, hatch='xx', ax=ax2)
for bar in ax2.containers[0]:
    x = bar.get_x()
    w = bar.get_width()
    bar.set_x(x + w * (1- width_scale))
    bar.set_width(w * width_scale)
ax2.set_ylim([0, 1])

plt.show()
```



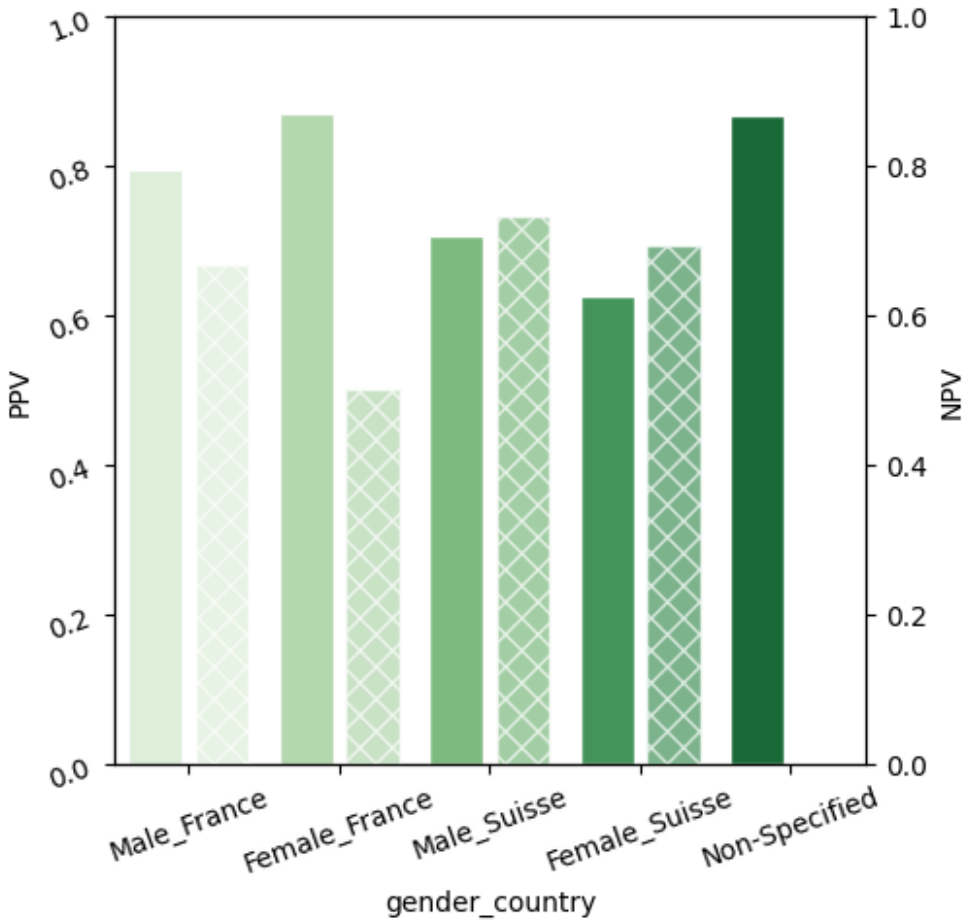

```
country_df[['Country (Diploma)', 'PPV', 'NPV']][:-1]
```

```
  Country (Diploma)    PPV    NPV
0            France  0.831  0.618
1            Suisse  0.684  0.718
```

Sensitive Attribute: Combine

```
combined_df['PPV'] = [

np.round(predictive_value_parity(demographics[(demographics['country_d
iploma'] == 'France') &

(demographics['gender'] == 'M')])[0], 3),

np.round(predictive_value_parity(demographics[(demographics['country_d
iploma'] == 'France') &

(demographics['gender'] == 'F')])[0], 3),

np.round(predictive_value_parity(demographics[(demographics['country_d
iploma'] == 'Suisse') &

(demographics['gender'] == 'M')])[0], 3),

np.round(predictive_value_parity(demographics[(demographics['country_d
iploma'] == 'Suisse') &

(demographics['gender'] == 'F')])[0], 3),

np.round(predictive_value_parity(demographics[demographics['country_di
ploma'].isna()])[0], 3),
                    ]

combined_df['NPV'] = [

np.round(predictive_value_parity(demographics[(demographics['country_d
iploma'] == 'France') &

(demographics['gender'] == 'M')])[1], 3),

np.round(predictive_value_parity(demographics[(demographics['country_d
iploma'] == 'France') &

(demographics['gender'] == 'F')])[1], 3),

np.round(predictive_value_parity(demographics[(demographics['country_d
iploma'] == 'Suisse') &

(demographics['gender'] == 'M')])[1], 3),

np.round(predictive_value_parity(demographics[(demographics['country_d
```

```
iploma'] == 'Suisse') &

(demographics['gender'] == 'F')])[1], 3),

np.round(predictive_value_parity(demographics[demographics['country_di
ploma'].isna()])[1], 3),
                    ]

plt.figure(figsize=(5, 5))
ax = sns.barplot(x='gender_country', y='PPV', data=combined_df,
palette = 'Greens', edgecolor = 'w')
width_scale = 0.45
for bar in ax.containers[0]:
    bar.set_width(bar.get_width() * width_scale)
ax.set_ylim([0, 1])

ax2 = ax.twinx()
sns.barplot(x='gender_country', y='NPV', data=combined_df, palette =
'Greens', edgecolor = 'w', alpha=0.7, hatch='xx', ax=ax2)
for bar in ax2.containers[0]:
    x = bar.get_x()
    w = bar.get_width()
    bar.set_x(x + w * (1- width_scale))
    bar.set_width(w * width_scale)
ax2.set_ylim([0, 1])

ax.tick_params(labelrotation=20)
plt.show()
```




```
combined_df[['gender_country', 'PPV', 'NPV']][:-1]
```

```
  gender_country    PPV    NPV
0    Male_France  0.794  0.667
1  Female_France  0.871  0.500
2    Male_Suisse  0.707  0.731
3  Female_Suisse  0.625  0.692
```

Interpretations

Country Attribute

We can see that in the Accuracy, Equal Opportunity (TPR) and Disparate Impact (PPP) metrics, the France category is greater than the Suisse category. However, they have close enough TPR/FRP to have Equalized Odds.

Also, the PPP_Ratio is bigger than the cutoff, and we have Demographic Parity since the difference between France and Suisse are smaller than the cutoff.

However, in the France category, there is a significance difference between PPV and NPV values, which show us that there is no Predictive Rate Parity.

Combined Attributes

Now looking at the Combined Attributes, we can see a significant difference in accuracy between Female_France and Female_Suisse.

We also see this difference in TPR leading to an Equal Opportunity bigger than the cutoff. For equalized odds, the difference in FPR between Female_France and Female_Suisse is very small, but still apparent. We cannot say that they have Equalized odds.

Conversely, looking at Male_France and Female_France, the difference in TPR is small but not in FPR.

For Disparate Impact, we see a significant difference in PPP between Female_France and Female_Suisse, although the ratio is bigger than the cutoff.

The main difference with the Gender/Country individual analyses vs. the combined subgroup analysis is when we see result from the Demographic Parity. A majority of |Demographic Parity| values are bigger than the cutoff. This can be caused by the different sizes of each category, as seen at the beginning of the notebook.

Lastly, regarding Predictive Value Parity, we have a significant difference between PPV and NPV in both Male_France and Female_France category, meaning that they don't have an equal chance of success given a positive or negative prediction. This disparity didn't exist in the individual analysis, which is why it is very important to run combined attribute analyses for fairness.

Fairness Metrics (Overall)

```
gender_df
```

	Gender	ACC	TPR	FNR	FPR	PPP	PPV	NPV
0	Male	0.728	0.789	0.211	0.352	0.600	0.747	0.700
1	Female	0.729	0.804	0.196	0.417	0.671	0.787	0.609
2	Non-Specified	0.684	0.765	0.235	1.000	0.789	0.867	0.000

```
country_df
```

	Country (Diploma)	ACC	TPR	FNR	FPR	PPP	PPV	NPV
0	France	0.758	0.806	0.194	0.344	0.657	0.831	0.618
1	Suisse	0.698	0.780	0.220	0.391	0.594	0.684	0.718
2	Non-Specified	0.684	0.765	0.235	1.000	0.789	0.867	0.000

```
combined_df
```

	gender_country	ACC	TPR	FNR	FPR	PPP	PPV	NPV
0	Male_France	0.741	0.771	0.229	0.304	0.586	0.794	0.667
1	Female_France	0.780	0.844	0.156	0.444	0.756	0.871	0.500
2	Male_Suisse	0.716	0.806	0.194	0.387	0.612	0.707	0.731
3	Female_Suisse	0.655	0.714	0.286	0.400	0.552	0.625	0.692
4	Non-Specified	0.684	0.765	0.235	1.000	0.789	0.867	0.000

Lab 13 Solution - Explainability

This week, we have seen an introduction to several explainability methods, used for peeking into the black-box of your neural network model and seeing what your model finds important while making predictions. Building upon last week's topics on fairness, this lecture on explainability is especially relevant to the ethical concerns of modeling human data.

In our lecture demo, we have seen at two different classes of AI explainability: global surrogate models (estimating the whole black box) with **Partial Dependence Plots (PDP)** and local surrogate models (explaining one instance's prediction) with **LIME**. We now want to examine whether these explanations align with each other.

The question we aim to answer with this lab:

> If we run a sample of local explanations on a random subset of our students, does it align with the global explanations for our model?

If you use noto for this notebook, don't forget to use the **Tensorflow** kernel.

```
# Load standard imports for the rest of the notebook.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

DATA_DIR = "./../../data/"

# Load explainability imports.
from lime import lime_tabular
import shap

# Suppress TF warnings during import
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
# Set log level to DEBUG again
tf.get_logger().setLevel('DEBUG')

model_name = "{}/explainability/model".format(DATA_DIR)
loaded_model = tf.keras.models.load_model(model_name)

features =
pd.read_csv('{}/explainability/mooc_features.csv'.format(DATA_DIR))
labels =
pd.read_csv('{}/explainability/mooc_labels.csv'.format(DATA_DIR))['0']

features.shape, labels.shape
```

```
((8679, 250), (8679,))
```

```
# For 8,679 students, we have 10 weeks of data with 25 features per
week.

display(features)
```

```
      RegPeakTimeDayHour_InWeek1  RegPeriodicityM1_InWeek1  \
0                       3.178054              1.000000e+00
1                       7.058606              3.041330e+00
2                       5.703059              3.092002e+00
3                       6.929695              2.435539e+00
4                      12.712215              1.000000e+00
...                          ...                       ...
8674                    0.980829              1.224647e-16
8675                    0.980829              1.224647e-16
8676                    0.980829              1.224647e-16
8677                    0.980829              1.224647e-16
8678                    0.980829              1.224647e-16

      DelayLecture_InWeek1  TotalClicks_InWeek1
NumberOfSessions_InWeek1  \
0                -518326.0                  1.0
0.0
1                -497116.5                 34.0
3.0
2                -481356.0                  7.0
0.0
3                -427158.0                 20.0
2.0
4                -517640.0                  4.0
1.0
...                    ...                  ...
...
8674             -518394.0                  0.0
0.0
8675             -518394.0                  0.0
0.0
8676             -518394.0                  0.0
0.0
8677             -518394.0                  0.0
0.0
8678             -518394.0                  0.0
0.0

      TotalTimeSessions_InWeek1  AvgTimeSessions_InWeek1  \
0                           0.0                 0.000000
1                        5423.0              1807.666667
2                           0.0                 0.000000
3                        4804.0              2402.000000
4                         863.0               863.000000
...                         ...                      ...
```

```
8674                          0.0                 0.000000
8675                          0.0                 0.000000
8676                          0.0                 0.000000
8677                          0.0                 0.000000
8678                          0.0                 0.000000

      StdTimeBetweenSessions_InWeek1  StdTimeSessions_InWeek1  \
0                                0.0                 0.000000
1                            90701.5              1158.870811
2                                0.0                 0.000000
3                                0.0               998.000000
4                                0.0                 0.000000
...                              ...                      ...
8674                             0.0                 0.000000
8675                             0.0                 0.000000
8676                             0.0                 0.000000
8677                             0.0                 0.000000
8678                             0.0                 0.000000

      TotalClicksWeekday_InWeek1  ...  TotalTimeVideo_InWeek10  \
0                            1.0  ...                      0.0
1                           26.0  ...                  10683.0
2                            7.0  ...                      0.0
3                           12.0  ...                   5325.0
4                            4.0  ...                      0.0
...                          ...  ...                      ...
8674                         0.0  ...                      0.0
8675                         0.0  ...                      0.0
8676                         0.0  ...                      0.0
8677                         0.0  ...                      0.0
8678                         0.0  ...                      0.0

      CompetencyAnticipation_InWeek10  ContentAlignment_InWeek10  \
0                                 0.0                        0.0
1                                 0.0                        0.8
2                                 0.0                        0.0
3                                 0.0                        1.0
4                                 0.0                        0.0
...                               ...                        ...
8674                              0.0                        0.0
8675                              0.0                        0.0
8676                              0.0                        0.0
8677                              0.0                        0.0
8678                              0.0                        0.0

      ContentAnticipation_InWeek10  StudentSpeed_InWeek10  \
0                              0.0                  16.00
1                              0.0                 558.00
2                              0.0                  16.00
3                              0.0                2074.25
```

```
4                                0.0                  16.00
...                              ...                    ...
8674                             0.0                  16.00
8675                             0.0                  16.00
8676                             0.0                  16.00
8677                             0.0                  16.00
8678                             0.0                  16.00

      TotalClicksVideoLoad_InWeek10  AvgWatchedWeeklyProp_InWeek10  \
0                               0.0                            0.0
1                              16.0                            0.8
2                               0.0                            0.0
3                              16.0                            1.0
4                               0.0                            0.0
...                             ...                            ...
8674                            0.0                            0.0
8675                            0.0                            0.0
8676                            0.0                            0.0
8677                            0.0                            0.0
8678                            0.0                            0.0

      AvgReplayedWeeklyProp_InWeek10  TotalClicksVideoConati_InWeek10
\
0                                0.0                              0.0

1                                0.2                             16.0

2                                0.0                              0.0

3                                0.0                             16.0

4                                0.0                              0.0

...                              ...                              ...

8674                             0.0                              0.0

8675                             0.0                              0.0

8676                             0.0                              0.0

8677                             0.0                              0.0

8678                             0.0                              0.0


      FrequencyEventLoad_InWeek10
0                        0.000000
1                        0.666667
```

```
2                         0.000000
3                         0.301887
4                         0.000000
...                            ...
8674                      0.000000
8675                      0.000000
8676                      0.000000
8677                      0.000000
8678                      0.000000

[8679 rows x 250 columns]
```

```
# For our true labels, we have a pass (0) or fail (1) performance
indicator. We only use these labels after obtaining model
# explanations, to try to understand how our model performs against
the ground truth.

# There are 8,679 students in this MOOC course.

display(labels)
```

```
0       1.0
1       0.0
2       1.0
3       0.0
4       1.0
       ...
8674    1.0
8675    1.0
8676    1.0
8677    1.0
8678    1.0
Name: 0, Length: 8679, dtype: float64
```

```
# This function returns a (NUM OF INSTANCES, 2) array of probability
of pass in first column and
# probability of failing in another column, which is the format LIME
requires.
predict_fn = lambda x: np.array([[1-loaded_model.predict(x)],

[loaded_model.predict(x)]]).reshape(2,-1).T

class_names = ['pass', 'fail']

# We initialize the LIME explainer on our training data.
explainer = lime_tabular.LimeTabularExplainer(
      training_data=np.array(features),
      feature_names=features.columns,
      class_names=class_names,
      mode='classification',
      discretize_continuous=True)
```

Select a subset of students

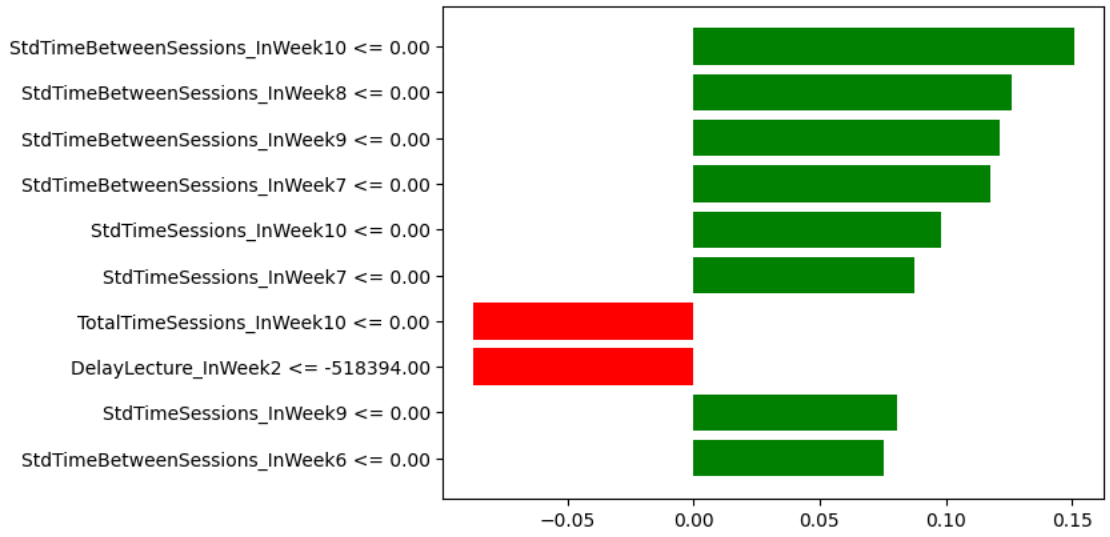
```
# We select 10 random students as instances from our dataset to
explain.

instances = range(10, 100, 10)
```

Generate local explanations with LIME (for multiple students)

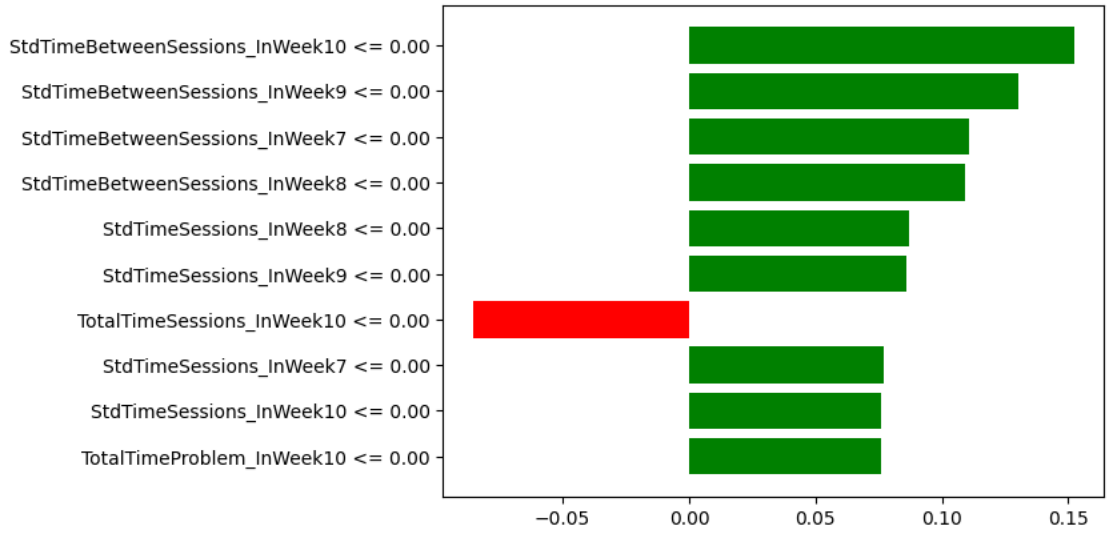
```
def plot_lime(exp, instance_id):
    s = 'fail' if labels[instance_id] else 'pass'
    label = exp.available_labels()[0]
    expl = exp.as_list(label=label)
    fig = plt.figure(facecolor='white')
    vals = [x[1] for x in expl]
    names = [x[0] for x in expl]
    vals.reverse()
    names.reverse()
    colors = ['green' if x > 0 else 'red' for x in vals]
    pos = np.arange(len(expl)) + .5
    plt.barh(pos, vals, align='center', color=colors)
    plt.yticks(pos, names)
    prediction =
loaded_model.predict(np.array(features.iloc[instance_id]).reshape(1,25
0))[0][0]
    prediction = np.round(1-prediction, 2)
    print("Student #: ", instance_id)
    print("Ground Truth Model Prediction: ", 1-labels[instance_id],
"-", s)
    print("Black Box Model Prediction: ", prediction, "-", 'pass' if
prediction > 0.5 else 'fail')

def DataFrame_all(explainers,instances,real_labels):
    df=pd.DataFrame({})
    class_names=['pass', 'fail']
    dfl=[]
    for i,exp in enumerate(explainers):
        this_label=exp.available_labels()[0]
        l=[]
        l.append(("exp number",instances[i]))
        l.append(("real value",'fail' if real_labels[instances[i]]
else 'pass'))
        l.extend(exp.as_list(label=this_label))
        dfl.append(dict(l))
    df=pd.concat((df, pd.DataFrame(dfl)))
    return df

explainers = []
for instance_id in instances:
    # This line calls our LIME explainer on a student instance.
```

```
explainers.append(explainer.explain_instance(features.iloc[instance_id
], predict_fn, num_features=10))

for exp, instance_id in zip(explainers, instances):
    plot_lime(exp, instance_id)
    plt.show()
```

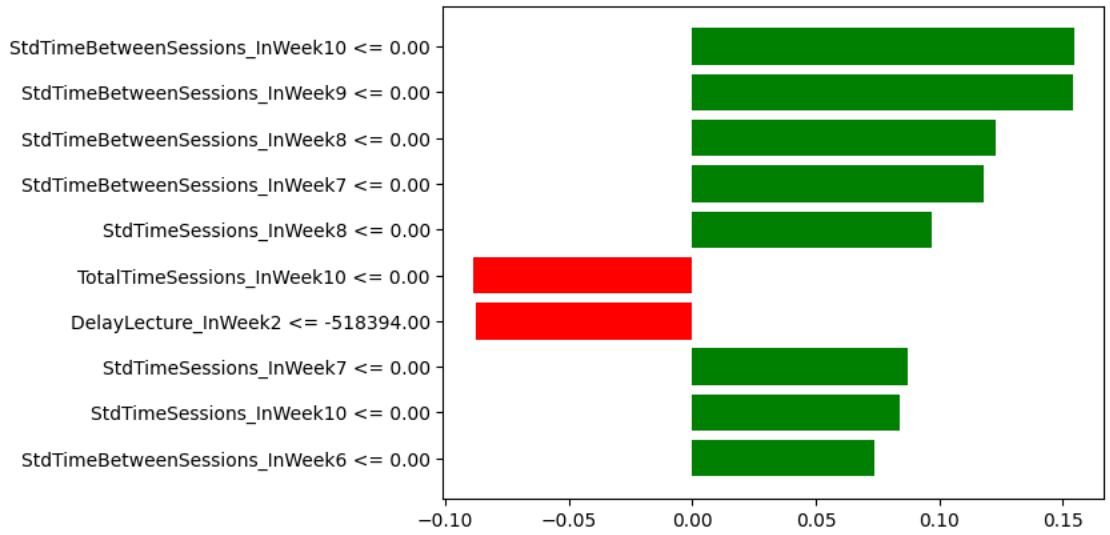
```
Student #:  10
Ground Truth Model Prediction:  1.0 - pass
Black Box Model Prediction:  0.39 - fail
```



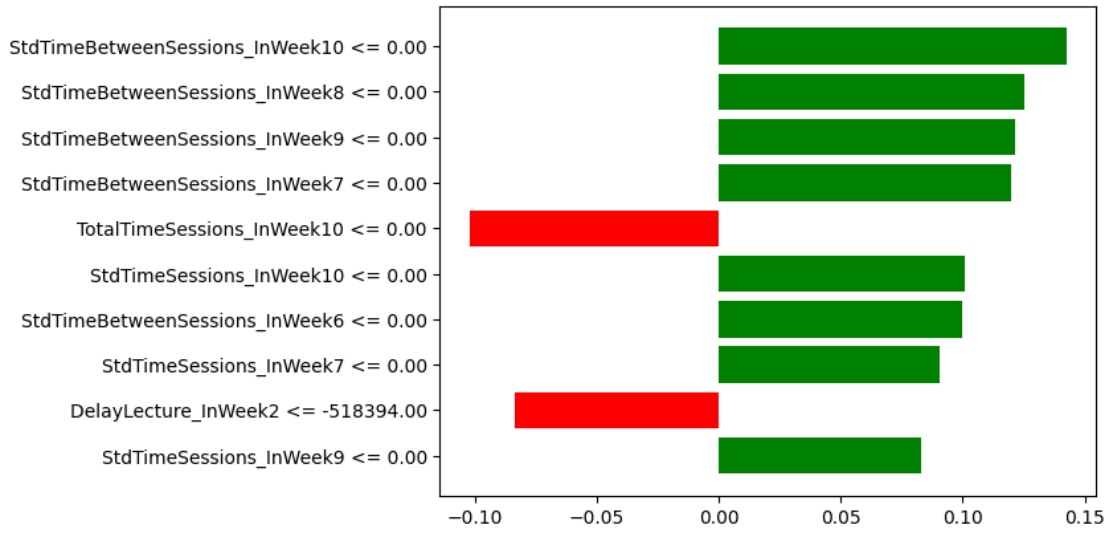

```
Student #:  20
Ground Truth Model Prediction:  0.0 - fail
Black Box Model Prediction:  0.35 - fail
```



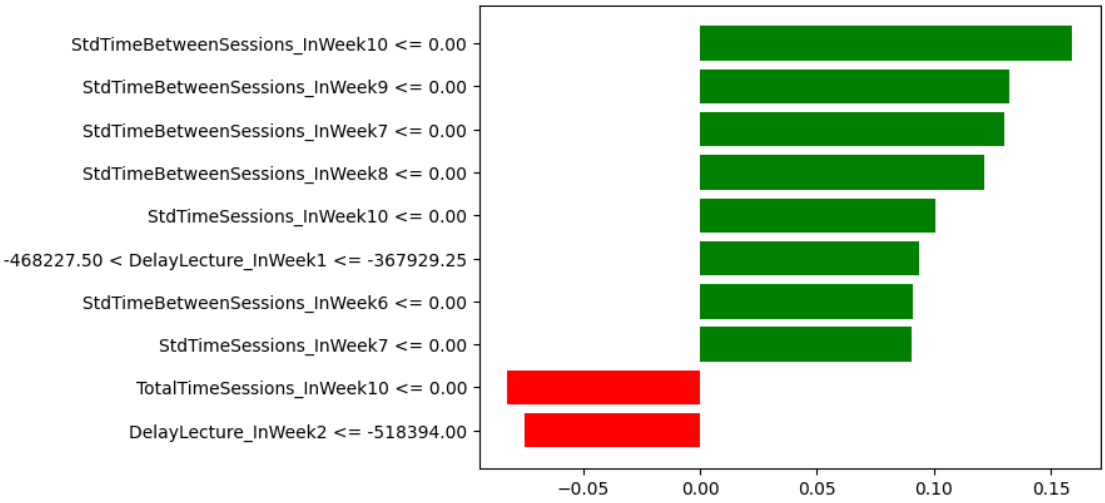
Student #: 30
Ground Truth Model Prediction: 0.0 - fail
Black Box Model Prediction: 0.4 - fail



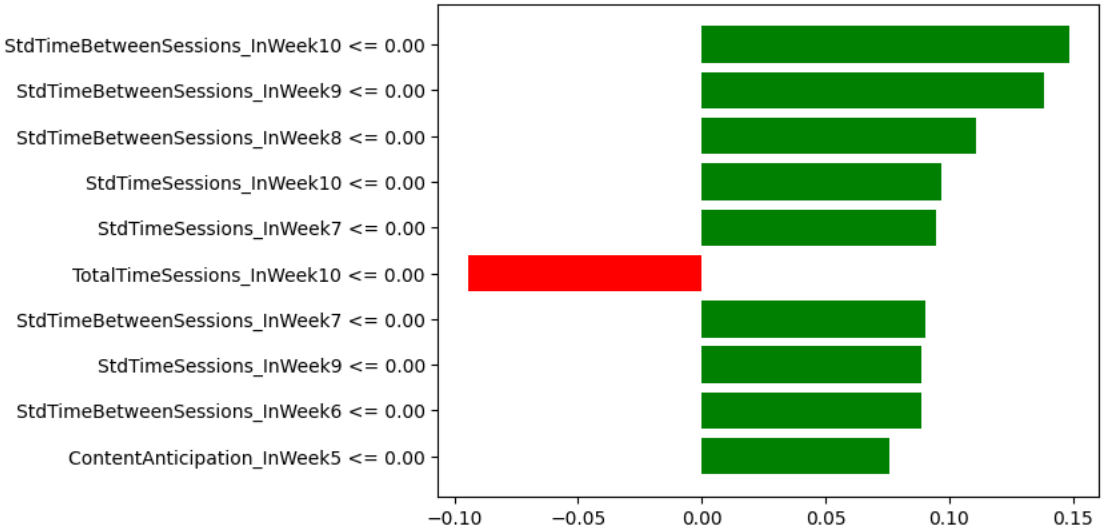
Student #: 40
Ground Truth Model Prediction: 0.0 - fail
Black Box Model Prediction: 0.4 - fail



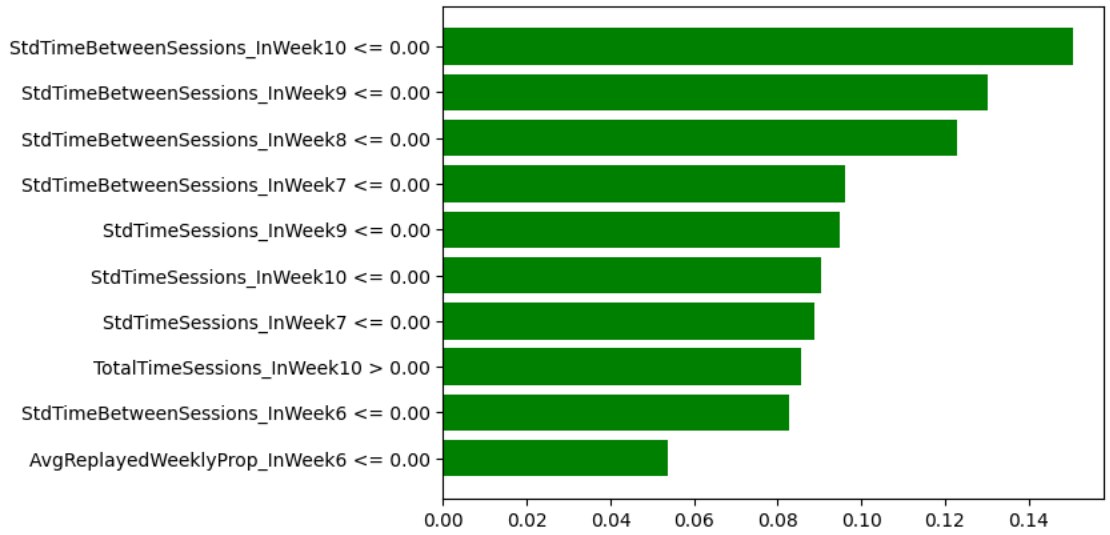
Student #: 50
Ground Truth Model Prediction: 0.0 - fail
Black Box Model Prediction: 0.4 - fail



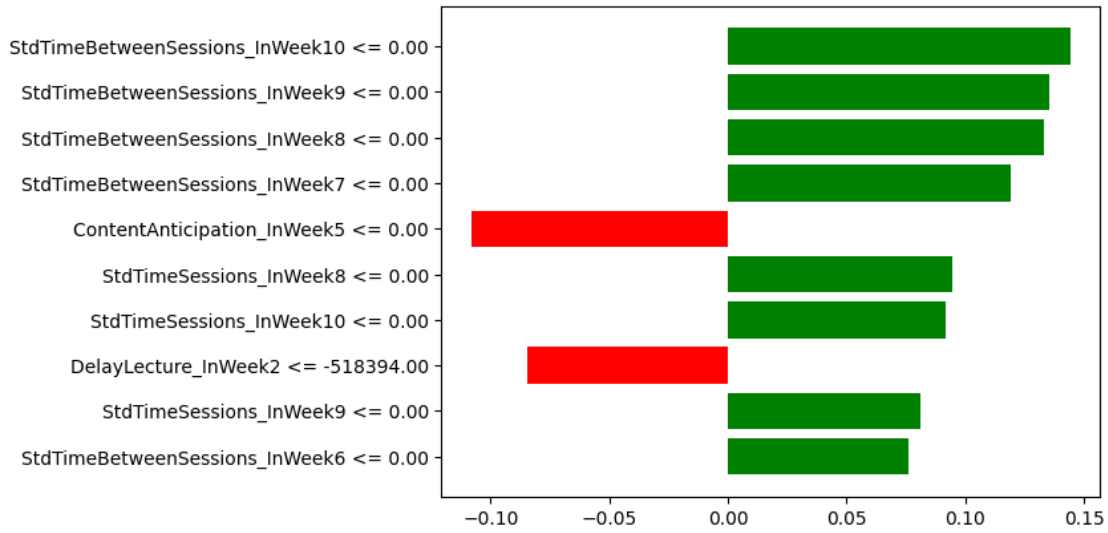
Student #: 60
Ground Truth Model Prediction: 0.0 - fail
Black Box Model Prediction: 0.31 - fail



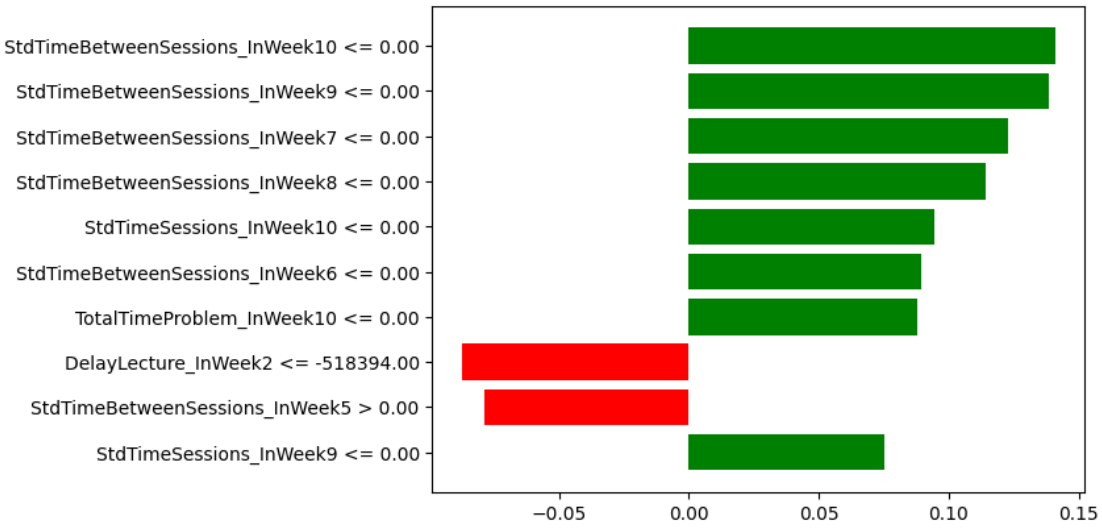
Student #: 70
Ground Truth Model Prediction: 0.0 - fail
Black Box Model Prediction: 0.38 - fail



Student #: 80
Ground Truth Model Prediction: 0.0 - fail
Black Box Model Prediction: 0.38 - fail



Student #: 90
Ground Truth Model Prediction: 1.0 - pass
Black Box Model Prediction: 0.31 - fail




```
# aggregate LIME feature importances (across students) into dataframe
for later analysis
```

```
df = DataFrame_all(explainers,instances,labels)
```

```
df
```

```
   exp number real value  StdTimeBetweenSessions_InWeek10 <= 0.00  \
0          10       pass                                0.150923
1          20       fail                                0.152395
2          30       fail                                0.154577
3          40       fail                                0.142618
4          50       fail                                0.159107
5          60       fail                                0.148449
6          70       fail                                0.150597
7          80       fail                                0.144249
8          90       pass                                0.141011

   StdTimeBetweenSessions_InWeek8 <= 0.00  \
0                                0.126338
1                                0.109334
2                                0.122779
3                                0.125340
4                                0.121358
5                                0.110948
6                                0.122806
7                                0.133345
8                                0.114114

   StdTimeBetweenSessions_InWeek9 <= 0.00  \
0                                0.121119
1                                0.130033
2                                0.153992
3                                0.121347
```

```
4                                   0.132354
5                                   0.138386
6                                   0.130208
7                                   0.135314
8                                   0.138561

   StdTimeBetweenSessions_InWeek7 <= 0.00  StdTimeSessions_InWeek10 <=
0.00  \
0                                   0.117679
0.098031
1                                   0.110975
0.075733
2                                   0.117917
0.083868
3                                   0.119601
0.100853
4                                   0.129841
0.100829
5                                   0.090478
0.096831
6                                   0.095982
0.090394
7                                   0.119371
0.091452
8                                   0.122756
0.094477

   StdTimeSessions_InWeek7 <= 0.00  TotalTimeSessions_InWeek10 <= 0.00
\
0                          0.087427                             -0.087278

1                          0.076877                             -0.085394

2                          0.087141                             -0.088555

3                          0.090338                             -0.102021

4                          0.090613                             -0.082264

5                          0.094840                             -0.094360

6                          0.088851                                   NaN

7                               NaN                                   NaN

8                               NaN                                   NaN


   DelayLecture_InWeek2 <= -518394.00  StdTimeSessions_InWeek9 <= 0.00
```

\
0 -0.087077 0.080462

1 NaN 0.086216

2 -0.087496 NaN

3 -0.083523 0.083167

4 -0.075243 NaN

5 NaN 0.088659

6 NaN 0.094751

7 -0.084579 0.081167

8 -0.087317 0.075152

StdTimeBetweenSessions_InWeek6 <= 0.00 StdTimeSessions_InWeek8 <= 0.00 \
0 0.075370 NaN
1 NaN 0.087222
2 0.073993 0.096822
3 0.100059 NaN
4 0.090900 NaN
5 0.088641 NaN
6 0.082641 NaN
7 0.076269 0.094555
8 0.089556 NaN

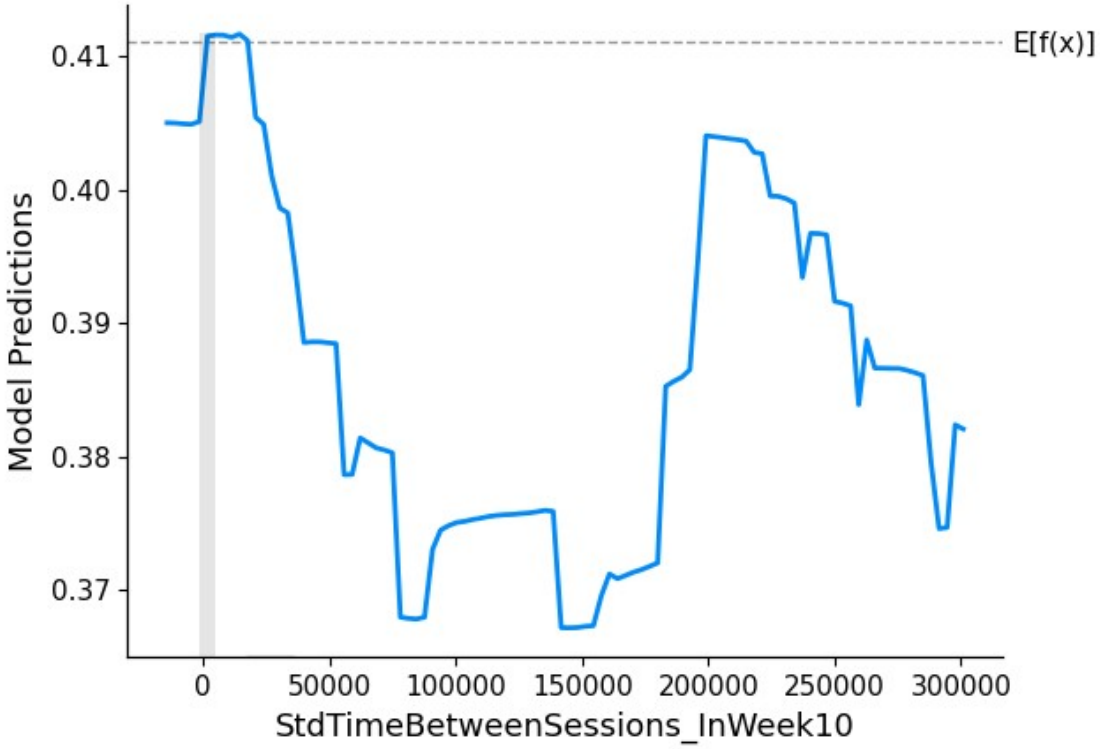
TotalTimeProblem_InWeek10 <= 0.00 \
0 NaN
1 0.075704
2 NaN
3 NaN
4 NaN
5 NaN
6 NaN

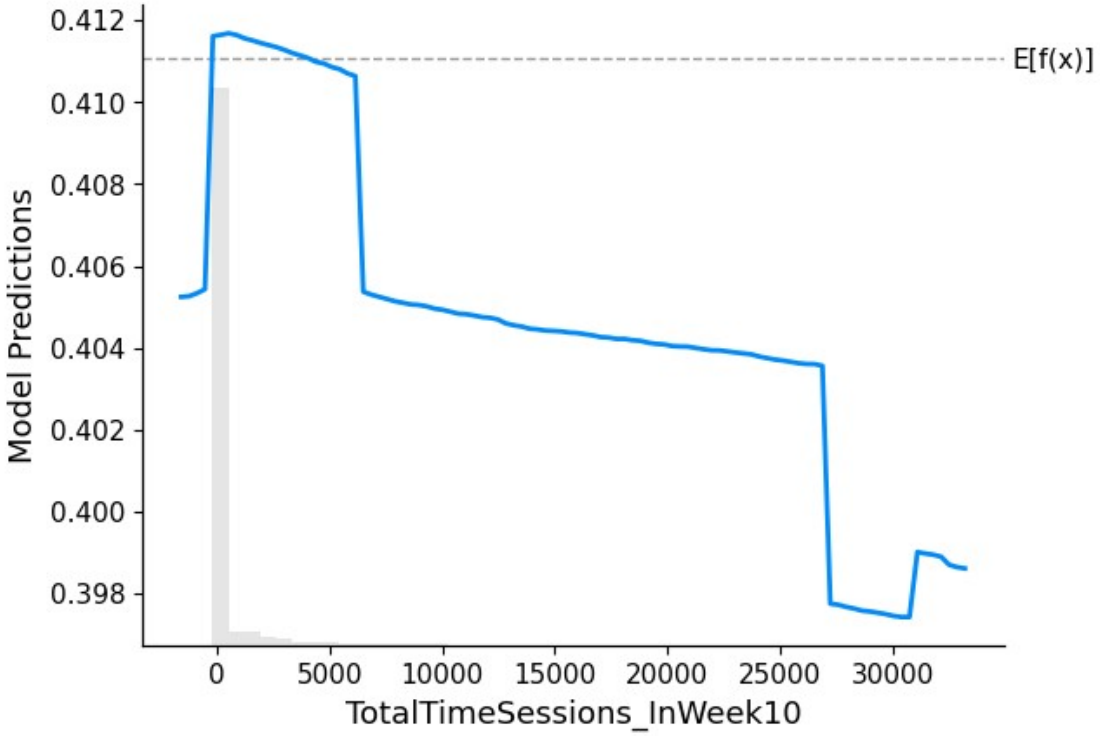
```
7                                 NaN
8                            0.087920

   -468227.50 < DelayLecture_InWeek1 <= -367929.25  \
0                                                NaN
1                                                NaN
2                                                NaN
3                                                NaN
4                                           0.093482
5                                                NaN
6                                                NaN
7                                                NaN
8                                                NaN

   ContentAnticipation_InWeek5 <= 0.00  TotalTimeSessions_InWeek10 >
0.00  \
0                                    NaN
NaN
1                                    NaN
NaN
2                                    NaN
NaN
3                                    NaN
NaN
4                                    NaN
NaN
5                               0.075763
NaN
6                                    NaN
0.085568
7                              -0.108007
NaN
8                                    NaN
NaN

   AvgReplayedWeeklyProp_InWeek6 <= 0.00  \
0                                     NaN
1                                     NaN
2                                     NaN
3                                     NaN
4                                     NaN
5                                     NaN
6                                0.053681
7                                     NaN
8                                     NaN

   StdTimeBetweenSessions_InWeek5 > 0.00
0                                     NaN
1                                     NaN
2                                     NaN
```

```
3                                    NaN
4                                    NaN
5                                    NaN
6                                    NaN
7                                    NaN
8                              -0.078919
```

Generate global explanations with PDP

```
# We generate the PDP plot against a background distribution of all
the points available in the feature set.
# While a minimal background distribution would let us run this
analysis faster (i.e. 300 points), we recommend
# plotting with a much larger point distribution (all the students) if
you use this in other situations for
# improved accuracy and a more global understanding of your model's
behavior.

background_distribution = features

# This function converts our data to the right format for the PDP
explainer.

predict_fn = lambda x: (1 - loaded_model.predict(x)).flatten()

# Based on the above analysis, select at least two features to analyze
with PDP.

features_to_analyze = ['StdTimeBetweenSessions_InWeek10',
'TotalTimeSessions_InWeek10']

for feature in features_to_analyze:
    feat = list(features.columns).index(feature)
    # Create a partial dependence plot from the background
distribution.
    fig = shap.plots.partial_dependence(
      feat, predict_fn, background_distribution, ice=False,
ylabel='Model Predictions',
      model_expected_value=True, feature_expected_value=False,
show=True
    )
```


E[f(x)]
0.41
0.40
0.39
0.38
0.37
Model Predictions
0
50000
100000
150000
200000
250000
300000
StdTimeBetweenSessions_InWeek10


E[f(x)]
0.412
0.410
0.408
0.406
0.404
0.402
0.400
0.398
Model Predictions
0
5000
10000
15000
20000
25000
30000
TotalTimeSessions_InWeek10

Comparing Global and Local Explanations

Choose two features selected as important by the LIME explanations and interpret the PDP plot. Do the LIME explanations' important features correspond with the PDP analysis of that feature?

TotalTimeProblem_InWeek10

StdTimeBetweenSessions_InWeek10

We can see in both of these cases that a peak at both features at 0 is important for predicting student success.

This often refers to students who have not had session interactions or solved problems at certain weeks.

We can infer that the model thinks that students who do nothing in week 10 are likely to fail and students who do a lot at week 10 have varying predictions (by analyzing the PDP plot in more detail). According to the PDP plot, there is a strong peak at 0, which tells us that this information applies to a lot of students.

In this way, both the LIME and PDP explanations are aligned.