Lecture 02 - Student Notebook

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
from scipy import stats
from scipy.stats import skewnorm
import seaborn as sns
import numpy as np
from sklearn.feature_selection import mutual_info_classif,
mutual_info_regression
from sklearn.preprocessing import LabelEncoder
```

```python
DATA_DIR = "./../../data"
```

Download the data from the Drive folder and put it in the data folder.

```python
# Aggregated features
df = pd.read_csv('{}/aggregated_fc.csv'.format(DATA_DIR))
df.head()
```

```
   user  grade gender       category        year  sessions
time_in_problem  \
0     0   4.50    NaN            NaN  Y2-2018-19      19.0
23344.0
1     1   4.50      M  Suisse.Autres  Y2-2018-19      34.0
16984.0
2     2   5.25      M     Suisse.PAM  Y2-2018-19      53.0
23406.0
3     3   4.50      F  Suisse.Autres  Y2-2018-19      28.0
27371.0
4     4   4.75      F         France  Y2-2018-19      25.0
37873.0

   time_in_video  lecture_delay  content_anticipation
mean_playback_speed  \
0        29518.0   55068.387500              0.006061
0.968519
1        92278.0   -2883.367738              0.009091
1.122014
2       108013.0   10027.216667              0.237488
0.807090
3        81855.0   27596.864484              0.011879
0.500000
4        70400.0    -914.633333              0.290421
0.846794
```

```
   relative_video_pause  submissions  submissions_correct  
clicks_weekend  \
0              0.137436         30.0                 20.0           
168.0
1              0.361389         90.0                 59.0            
40.0
2              0.272210         61.0                 30.0           
946.0
3              0.151223         46.0                 32.0           
135.0
4              0.196403          3.0                  1.0           
584.0

   clicks_weekday
0           381.0
1          1794.0
2          1292.0
3           464.0
4           649.0
```

```
# Time series features
ts = pd.read_csv('{}/time_series_fc.csv'.format(DATA_DIR))
ts.head()
```

```
   week  user  sessions  time_in_problem  time_in_video  lecture_delay  
\
0     0     0       4.0           5682.0         6417.0  -24339.200000   

1     0     1       7.0            326.0        15525.0    4492.833333   

2     0     2       4.0           1224.0        12209.0   -8998.000000   

3     0     3      11.0           3517.0        26500.0  -33102.111111   

4     0     4       4.0           1294.0        12037.0   -9146.333333   


   content_anticipation  mean_playback_speed  relative_video_pause  \
0              0.015152             1.539474              0.315217   
1              0.090909             1.319288              0.345528   
2              0.060606             1.000000              0.230415   
3              0.045455             1.000000              0.301887   
4              0.181818             1.184140              0.267606   

   submissions  submissions_correct  clicks_weekend  clicks_weekday  
0          8.0                  4.0            12.0           102.0  
1          7.0                  4.0            40.0           227.0  
2         13.0                  8.0             1.0           258.0  
3         17.0                 10.0            10.0           141.0  
4          3.0                  1.0           140.0            46.0  
```

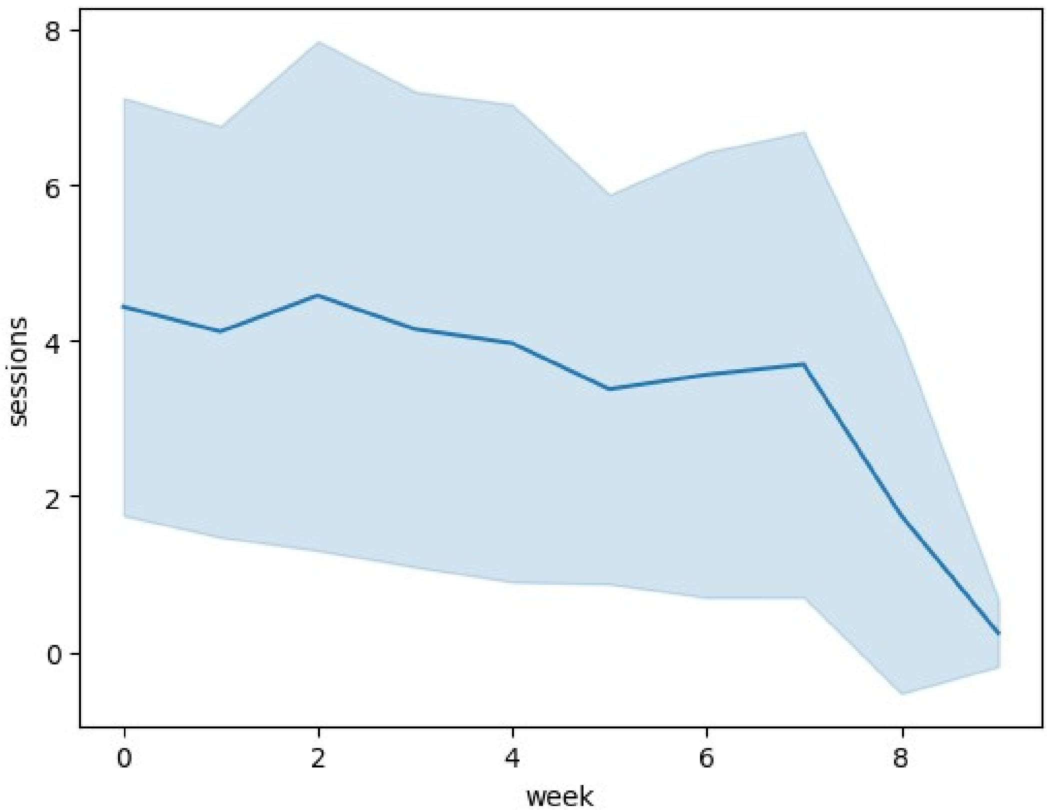
Some useful functions

```python
def plot_features(df, hue = None):
    continuous_cols = list(df._get_numeric_data().columns)
    categorical_cols =
list(df.select_dtypes(include=['O']).columns.values)

    rows = np.ceil(len(df.columns)/3).astype(int)
    fig, axes = plt.subplots(rows, 3, figsize=(15,5*rows))
    for i, col in enumerate(df.columns):
        ax = axes[i // 3, i % 3]
        if col in continuous_cols:
            sns.histplot(data=df, x = col, ax=ax,  kde=True, hue= hue)

        elif col in categorical_cols:
            sns.countplot(data=df, x=col, ax=ax, hue = hue)
        else:
            print(col)
        ax.set(xlabel=col, ylabel='Count', title= 'Distribution
{}'.format(col))

    fig.tight_layout()
    plt.show()


def plot_time_series(df, hue=None):
    continuous_cols = list(df._get_numeric_data().columns)

    rows = np.ceil(len(continuous_cols)/3).astype(int)
    fig, axes = plt.subplots(rows, 3, figsize=(15,5*rows))
    for i, col in enumerate(continuous_cols):
        ax = axes[i // 3, i % 3]
        sns.lineplot(data=df, x="week", y=col, ax = ax, errorbar='sd',
hue=hue)
        ax.set(xlabel="week", ylabel=col, title= 'Time series
{}'.format(col))

    fig.tight_layout()
    plt.show()
```

Example Questions

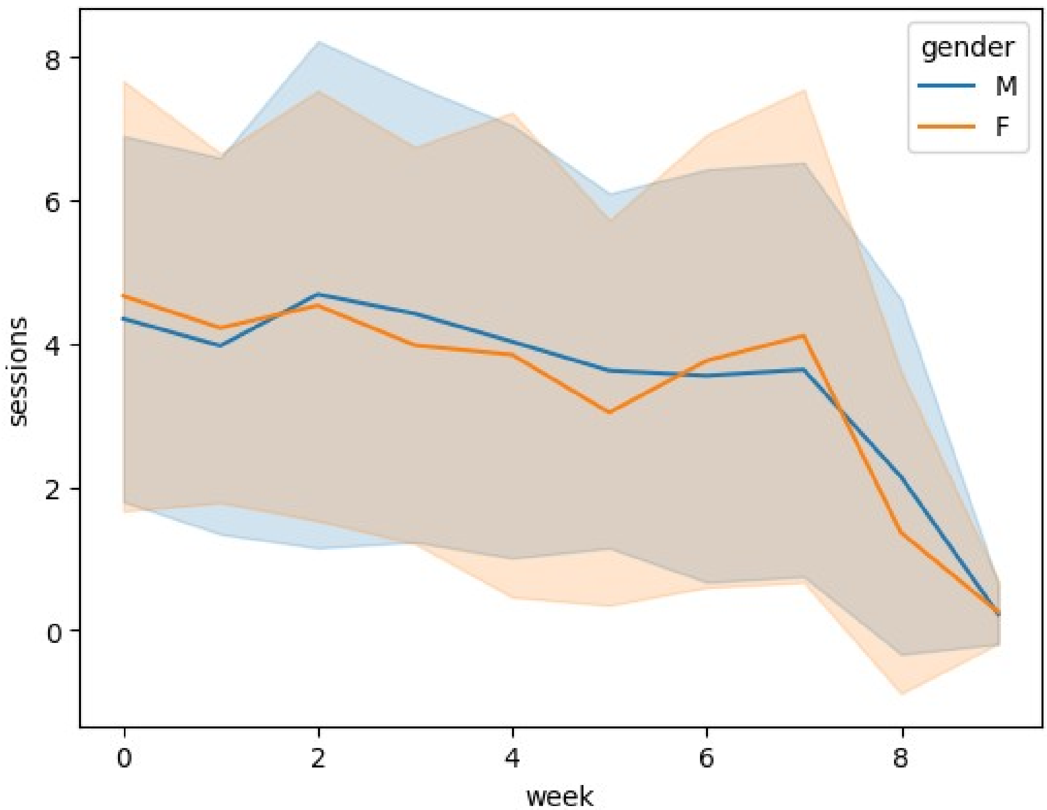
H1: Students will work more at the beginning of the semester (due to decreasing motivation over the course of the semester).

```python
ax = sns.lineplot(data=ts, x="week", y="sessions", errorbar='sd')
plt.show()
```



H2: There is no difference between males and females in terms of the number of sessions.

```
ts = ts.merge(df[['user','gender']], how='left', on='user')

ax = sns.lineplot(data=ts, x="week", y="sessions",errorbar='sd', hue =
'gender')
```



Your turn

```
import requests

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

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

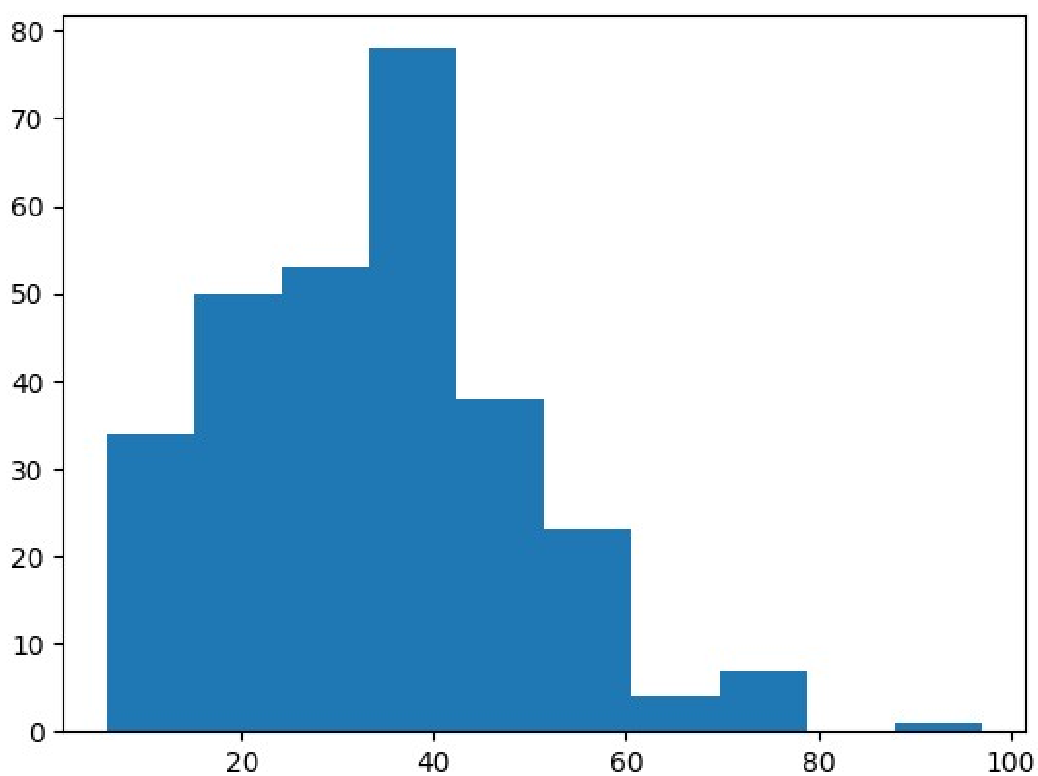
```
Your name:  Paola
```

```
### Write briefly your question or hypothesis as a string
rq = """
This is an example hypothesis
"""

### Share it with us
send(rq, 1)
```

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

```python
### Plot it and share it with us

# Example plot (do a better one!)
plt.hist(df.sessions)

send(plt, 2)
plt.show()

### Discuss briefly as a string what you observed: can you confirm
your hypothesis?
hy = """This is an example discussion"""

### Share it with us
send(hy, 3)
```

```
<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]>
```

Lecture 03 - Student Notebook

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

We extended the data with extra features. The feature description is found here.

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

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

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

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

# Parse the aggregated and time series data
df = pd.read_csv('{}/aggregated_extended_fc.csv'.format(DATA_DIR))
df = df.fillna('NaN')
list(df.columns)
display(df)

df_byweek = pd.read_csv('{}/fc_long_extended.csv'.format(DATA_DIR))
display(df_byweek)
```

```
     user  ch_num_sessions  ch_time_in_prob_sum  ch_time_in_video_sum
\
0       0              1.9               2334.4                2951.8

1       1              3.4               1698.4                9227.8

2       2              5.3               2340.6               10801.3

3       3              2.8               2737.1                8185.5

4       4              2.5               3787.3                7040.0

..    ...              ...                  ...                   ...

283   293              3.5               8127.5                 113.4
```

```
284   294             2.2               2452.4               4623.1

285   296             0.9               1643.2               1932.4

286   297             1.4               2718.6                360.3

287   298             0.9                  0.1               1954.9


     ch_ratio_clicks_weekend_day  ch_total_clicks_weekend  \
0                       0.850000                     16.8
1                       0.567500                      4.0
2                      26.562274                     94.6
3                       3.691250                     13.5
4                       1.543889                     58.4
..                           ...                      ...
283                     0.632304                     28.9
284                    18.147762                     36.4
285                     0.000000                      0.4
286                     0.180000                      2.0
287                     0.597368                     15.9

     ch_total_clicks_weekday  ch_time_sessions_mean
ch_time_sessions_std  \
0                       38.1            1392.858333
790.762032
1                      179.4            3068.720238
1257.504407
2                      129.2            1750.289268
1024.134043
3                       46.4           20203.590260
656.052901
4                       64.9            3373.908333
1363.320365
..                       ...                    ...
...
283                     20.6            7963.627500
1001.514794
284                     71.3            3614.055952
853.195566
285                     31.2             926.916667
616.918475
286                     15.3             346.437500
122.017326
287                      3.3             350.758333
266.095738

     bo_delay_lecture  ...  la_weekly_prop_watched_mean  \
```

```
0        55068.387500  ...                      0.245714
1        -2883.367738  ...                      0.748868
2        10027.216667  ...                      0.354487
3        27596.864484  ...                      0.370000
4         -914.633333  ...                      0.030000
..                ...  ...                           ...
283          0.000000  ...                      0.000000
284      16834.900000  ...                      0.140530
285     -12860.522222  ...                      0.069231
286          0.000000  ...                      0.000000
287          0.000000  ...                      0.000000

     la_weekly_prop_interrupted_mean
la_weekly_prop_interrupted_std  \
0                           0.024286                             0.0

1                           0.074683                             0.0

2                           0.026667                             0.0

3                           0.014286                             0.0

4                           0.000000                             0.0

..                               ...                             ...

283                         0.000000                             0.0

284                         0.011111                             0.0

285                         0.023077                             0.0

286                         0.000000                             0.0

287                         0.000000                             0.0


     la_weekly_prop_replayed_mean  la_weekly_prop_replayed_std  \
0                        0.010000                          0.0
1                        0.066456                          0.0
2                        0.059915                          0.0
3                        0.020000                          0.0
4                        0.020000                          0.0
..                            ...                          ...
283                      0.000000                          0.0
284                      0.000000                          0.0
285                      0.000000                          0.0
286                      0.000000                          0.0
287                      0.000000                          0.0
```

```
     la_frequency_action_video_play  grade  gender       category
year
0                          0.179203   4.50     NaN            NaN  Y2-
2018-19
1                          0.332424   4.50       M  Suisse.Autres  Y2-
2018-19
2                          0.284407   5.25       M     Suisse.PAM  Y2-
2018-19
3                          0.108774   4.50       F  Suisse.Autres  Y2-
2018-19
4                          0.199775   4.75       F         France  Y2-
2018-19
..                              ...    ...     ...            ...
...
283                        0.034080   5.25       M         France  Y3-
2019-20
284                        0.186649   5.25       F         France  Y3-
2019-20
285                        0.028596   6.00       F         France  Y3-
2019-20
286                        0.032353   5.00       M     Suisse.PAM  Y3-
2019-20
287                        0.127182   4.00       M         France  Y3-
2019-20

[288 rows x 38 columns]

      Unnamed: 0  week  user  ch_num_sessions  ch_time_in_prob_sum  \
0             10     0     1              7.0                326.0
1             11     1     1              4.0                350.0
2             12     2     1              5.0               4577.0
3             13     3     1              4.0                259.0
4             14     4     1              3.0                480.0
...          ...   ...   ...              ...                  ...
2335        2835     5   293              2.0               9315.0
2336        2836     6   293              3.0                 86.0
2337        2837     7   293              3.0               3675.0
2338        2838     8   293              5.0              10956.0
2339        2839     9   293              1.0                  0.0

      ch_time_in_video_sum  ch_ratio_clicks_weekend_day  \
0                  15525.0                     5.675000
1                   8411.0                     0.000000
2                   8691.0                     0.000000
3                  12055.0                     0.000000
4                  13235.0                     0.000000
...                    ...                          ...
2335                   0.0                     0.513514
2336                 549.0                     4.333333
```

```
2337                   0.0                     0.000000
2338                   0.0                     0.000000
2339                   0.0                     0.000000

      ch_total_clicks_weekend  ch_total_clicks_weekday
ch_time_sessions_mean  \
0                        40.0                    227.0
1931.285714
1                         0.0                    207.0
2190.250000
2                         0.0                    167.0
2106.200000
3                         0.0                    239.0
3078.500000
4                         0.0                    197.0
4116.666667
...                       ...                      ...
...
2335                     37.0                     19.0
4657.500000
2336                      3.0                     13.0
211.666667
2337                      0.0                     41.0
1225.000000
2338                      0.0                     53.0
601.600000
2339                     14.0                      0.0
62893.000000

      ...  la_seek_len_std  la_pause_dur_std  la_time_speeding_up_mean
\
0     ...       146.564097        188.175709                 65.173554

1     ...         8.486253         78.639644                 47.872928

2     ...        63.484419        105.108022                 64.533835

3     ...        31.535282         75.997314                 58.085308

4     ...        10.594150        202.504038                 78.057143

...   ...              ...               ...                       ...

2335  ...         0.000000          0.000000                  0.000000

2336  ...         0.000000        116.639044                 13.000000

2337  ...         0.000000          0.000000                  0.000000
```

```
2338  ...        0.000000         0.000000                  0.000000

2339  ...        0.000000         0.000000                  0.000000


      la_time_speeding_up_std  la_weekly_prop_watched_mean  \
0                  150.807752                     0.600000
1                   67.365584                     0.800000
2                   81.772612                     1.000000
3                   86.465139                     0.769231
4                  140.802708                     1.000000
...                       ...                          ...
2335                 0.000000                     0.000000
2336                 9.000000                     0.000000
2337                 0.000000                     0.000000
2338                 0.000000                     0.000000
2339                 0.000000                     0.000000

      la_weekly_prop_interrupted_mean  la_weekly_prop_interrupted_std
\
0                            0.100000                             0.0

1                            0.000000                             0.0

2                            0.000000                             0.0

3                            0.000000                             0.0

4                            0.285714                             0.0

...                               ...                             ...

2335                         0.000000                             0.0

2336                         0.000000                             0.0

2337                         0.000000                             0.0

2338                         0.000000                             0.0

2339                         0.000000                             0.0


      la_weekly_prop_replayed_mean  la_weekly_prop_replayed_std  \
0                         0.000000                          0.0
1                         0.100000                          0.0
2                         0.000000                          0.0
3                         0.153846                          0.0
4                         0.285714                          0.0
```

```
...                         ...                         ...
2335                   0.000000                         0.0
2336                   0.000000                         0.0
2337                   0.000000                         0.0
2338                   0.000000                         0.0
2339                   0.000000                         0.0

      la_frequency_action_video_play
0                           0.400749
1                           0.391304
2                           0.359281
3                           0.359833
4                           0.390863
...                              ...
2335                        0.000000
2336                        0.312500
2337                        0.000000
2338                        0.000000
2339                        0.000000

[2340 rows x 36 columns]
```

Predicting student performance early on

In this task, we are interested in predicting course grade early on during the semester. This type of information can be useful for an instructor in order to be able to provide intervention to struggling students. We will use again the category as a random effect. We will need to train a separate model for each week (i.e. predicting after 1 week of the course, after 2 weeks of the course, after 3 weeks, etc.). However, we will use the same equation for all models.

First, we create a dataframe containing information about the user.

```python
# parse the necessary data frames
df_ui = (df.loc[:,['user','grade','gender','category','year']]).copy()

# compute pass/fail label
df_ui['passed'] = df_ui.loc[:,'grade'] >= 4
df_ui.loc[:,'passed'] = df_ui.loc[:,'passed'].replace(True,1)
df_ui.loc[:,'passed'] = df_ui.loc[:,'passed'].replace(False,0)
display(df_ui)
```

```
     user  grade gender       category        year  passed
0       0   4.50    NaN            NaN  Y2-2018-19       1
1       1   4.50      M  Suisse.Autres  Y2-2018-19       1
2       2   5.25      M     Suisse.PAM  Y2-2018-19       1
3       3   4.50      F  Suisse.Autres  Y2-2018-19       1
4       4   4.75      F         France  Y2-2018-19       1
..    ...    ...    ...            ...         ...     ...
283   293   5.25      M         France  Y3-2019-20       1
```

```
284   294   5.25      F         France  Y3-2019-20       1
285   296   6.00      F         France  Y3-2019-20       1
286   297   5.00      M     Suisse.PAM  Y3-2019-20       1
287   298   4.00      M         France  Y3-2019-20       1

[288 rows x 6 columns]
```

Next, we reformat the data frame to contain values by week and user.

```
df_byuser = df_byweek.sort_values(by=['user',
'week']).reset_index(drop=True)
display(df_byuser)
```

```
      Unnamed: 0  week  user  ch_num_sessions  ch_time_in_prob_sum  \
0             10     0     1              7.0                326.0
1             11     1     1              4.0                350.0
2             12     2     1              5.0               4577.0
3             13     3     1              4.0                259.0
4             14     4     1              3.0                480.0
...          ...   ...   ...              ...                  ...
2335        2835     5   293              2.0               9315.0
2336        2836     6   293              3.0                 86.0
2337        2837     7   293              3.0               3675.0
2338        2838     8   293              5.0              10956.0
2339        2839     9   293              1.0                  0.0

      ch_time_in_video_sum  ch_ratio_clicks_weekend_day  \
0                  15525.0                     5.675000
1                   8411.0                     0.000000
2                   8691.0                     0.000000
3                  12055.0                     0.000000
4                  13235.0                     0.000000
...                    ...                          ...
2335                   0.0                     0.513514
2336                 549.0                     4.333333
2337                   0.0                     0.000000
2338                   0.0                     0.000000
2339                   0.0                     0.000000

      ch_total_clicks_weekend  ch_total_clicks_weekday
ch_time_sessions_mean  \
0                        40.0                    227.0
1931.285714
1                         0.0                    207.0
2190.250000
2                         0.0                    167.0
2106.200000
3                         0.0                    239.0
3078.500000
4                         0.0                    197.0
```

```
4116.666667
...                        ...                        ...
...
2335                      37.0                       19.0
4657.500000
2336                       3.0                       13.0
211.666667
2337                       0.0                       41.0
1225.000000
2338                       0.0                       53.0
601.600000
2339                      14.0                        0.0
62893.000000

      ...  la_seek_len_std  la_pause_dur_std  la_time_speeding_up_mean  \
0     ...       146.564097        188.175709                 65.173554

1     ...         8.486253         78.639644                 47.872928

2     ...        63.484419        105.108022                 64.533835

3     ...        31.535282         75.997314                 58.085308

4     ...        10.594150        202.504038                 78.057143

...   ...              ...               ...                       ...

2335  ...         0.000000          0.000000                  0.000000

2336  ...         0.000000        116.639044                 13.000000

2337  ...         0.000000          0.000000                  0.000000

2338  ...         0.000000          0.000000                  0.000000

2339  ...         0.000000          0.000000                  0.000000


      la_time_speeding_up_std  la_weekly_prop_watched_mean  \
0                  150.807752                     0.600000
1                   67.365584                     0.800000
2                   81.772612                     1.000000
3                   86.465139                     0.769231
4                  140.802708                     1.000000
...                       ...                          ...
2335                 0.000000                     0.000000
2336                 9.000000                     0.000000
2337                 0.000000                     0.000000
```

```
2338                  0.000000                             0.000000
2339                  0.000000                             0.000000

      la_weekly_prop_interrupted_mean  la_weekly_prop_interrupted_std
\
0                            0.100000                             0.0

1                            0.000000                             0.0

2                            0.000000                             0.0

3                            0.000000                             0.0

4                            0.285714                             0.0

...                               ...                             ...

2335                         0.000000                             0.0

2336                         0.000000                             0.0

2337                         0.000000                             0.0

2338                         0.000000                             0.0

2339                         0.000000                             0.0


      la_weekly_prop_replayed_mean  la_weekly_prop_replayed_std  \
0                         0.000000                          0.0
1                         0.100000                          0.0
2                         0.000000                          0.0
3                         0.153846                          0.0
4                         0.285714                          0.0
...                            ...                          ...
2335                      0.000000                          0.0
2336                      0.000000                          0.0
2337                      0.000000                          0.0
2338                      0.000000                          0.0
2339                      0.000000                          0.0

      la_frequency_action_video_play
0                           0.400749
1                           0.391304
2                           0.359281
3                           0.359833
4                           0.390863
...                              ...
2335                        0.000000
```

```
2336                            0.312500
2337                            0.000000
2338                            0.000000
2339                            0.000000

[2340 rows x 36 columns]
```

We can now create a model that predicts the exam grade after a specific number of weeks of the course. We will use 5 weeks and 10 weeks.

Step 1: We will write a function that aggregates the features for all weeks.

```
def aggregate_features(df_ui, df_byuser, week_nr):

    df_weeknr = df_byuser[df_byuser['week'] < week_nr]
    df_return = df_weeknr.groupby(['user']).mean()
    df_return['user'] = df_return.index

    # Return df with aggregated features
    df_return =
df_return.set_index('user').join(df_ui.set_index('user'))
    df_return.reset_index()

    return df_return
```

Step 2: We will split the data into a training and test set (20% users in the test set, stratified by pass/fail label). In our case, **data stratification** refers to choosing a sample with the same ratio of pass/fail as the initial dataset, so our training set and our test set are both representative of our original population. If you are interested, you can read more about stratifying test sets here.

```
# perform train/test split
df_week5 = aggregate_features(df_ui, df_byuser, 5)
df_train5, df_test5 = train_test_split(df_week5, test_size=0.2,
random_state=0,  stratify=df_week5['passed'])

df_week10 = aggregate_features(df_ui, df_byuser, 10)
df_train10, df_test10 = train_test_split(df_week10, test_size=0.2,
random_state=0,  stratify=df_week10['passed'])
```

Step 3: We will now train our model on the training data for 5 and 10 weeks. We will use the following formula: `grade ~ (1|category) + wa_num_subs_perc_correct`

```
# Train a multi-regression model for weeks 5 and 10
# Initialize model instance using 1 predictor with random intercepts
and slopes
model5 = Lmer("grade ~ (1|category) +  wa_num_subs_perc_correct",
data=df_train5, family='gaussian')
model10 = Lmer("grade ~ (1|category) +  wa_num_subs_perc_correct",
data=df_train10, family='gaussian')
```

```python
# Fit the models
print(model5.fit())
print(model10.fit())
```

```
Formula: grade~(1|category)+wa_num_subs_perc_correct

Family: gaussian	 Inference: parametric

Number of observations: 187	 Groups: {'category': 5.0}

Log-likelihood: -295.857 	 AIC: 591.714

Random effects:

                 Name    Var    Std
category  (Intercept)  0.072  0.269
Residual               1.349  1.161

No random effect correlations specified

Fixed effects:

                          Estimate  2.5_ci  97.5_ci     SE       DF
T-stat  \
(Intercept)                  3.813   3.404    4.222  0.209   12.154
18.273
wa_num_subs_perc_correct     0.553  -0.160    1.266  0.364  182.752
1.520

                          P-val  Sig
(Intercept)                0.00  ***
wa_num_subs_perc_correct   0.13
Formula: grade~(1|category)+wa_num_subs_perc_correct

Family: gaussian	 Inference: parametric

Number of observations: 187	 Groups: {'category': 5.0}

Log-likelihood: -293.932 	 AIC: 587.863

Random effects:

                 Name    Var    Std
category  (Intercept)  0.075  0.273
Residual               1.322  1.150

No random effect correlations specified

Fixed effects:
```

```
                          Estimate  2.5_ci  97.5_ci     SE       DF
T-stat  \
(Intercept)                  3.694   3.288    4.101  0.207   11.588
17.810
wa_num_subs_perc_correct     1.037   0.201    1.873  0.427  183.054
2.431

                          P-val  Sig
(Intercept)               0.000  ***
wa_num_subs_perc_correct  0.016    *
```

Step 4: We predict on the test data and check the accuracy.

```python
# predict on the test data for weeks 5, 10
predictions5 = model5.predict(df_test5, verify_predictions=False)
rmse5 = mean_squared_error(df_test5['grade'], predictions5,
squared=False)

predictions10 = model10.predict(df_test10, verify_predictions=False)
rmse10 = mean_squared_error(df_test10['grade'], predictions10,
squared=False)

print(rmse5)
print(rmse10)
```

```
1.1346320524664855
1.1203193591371505
```

Your Turn

We are interested in predicting pass/fail (denoted by `passed` in the dataframe) instead of the grade.

1. Adjust the equations of `model5` and `model10` to predict pass/fail instead of the grade.
2. Train the two models, evalute their accuracy on the test data set, and send us the RMSE.

```python
import requests

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

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

```
---------------------------------------------------------------------
-----
KeyboardInterrupt                          Traceback (most recent call
last)
Input In [8], in <cell line: 6>()
      1 import requests
      3 exec(requests.get("https://courdier.pythonanywhere.com/get-
send-code").content)
      5 npt_config = {
      6     'session_name': 'lecture-03',
      7     'session_owner': 'mlbd',
----> 8     'sender_name': input("Your name: "),
      9 }

File
/usr/local/lib/python3.8/dist-packages/ipykernel/kernelbase.py:1177,
in Kernel.raw_input(self, prompt)
   1173 if not self._allow_stdin:
   1174     raise StdinNotImplementedError(
   1175         "raw_input was called, but this frontend does not
support input requests."
   1176     )
-> 1177 return self._input_request(
   1178     str(prompt),
   1179     self._parent_ident["shell"],
   1180     self.get_parent("shell"),
   1181     password=False,
   1182 )

File
/usr/local/lib/python3.8/dist-packages/ipykernel/kernelbase.py:1219,
in Kernel._input_request(self, prompt, ident, parent, password)
   1216             break
   1217 except KeyboardInterrupt:
   1218     # re-raise KeyboardInterrupt, to truncate traceback
-> 1219     raise KeyboardInterrupt("Interrupted by user") from None
   1220 except Exception:
   1221     self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

```python
# Define the model equations
model5 = Lmer("passed ~ (1|category) +  wa_num_subs_perc_correct",
data=df_train5, family='binomial')
model10 = Lmer("passed ~ (1|category) +  wa_num_subs_perc_correct",
data=df_train10, family='binomial')

# Fit the models
print(model5.fit())
print(model10.fit())
```

```python
# predict on the test data for weeks 5, 10
predictions5 = model5.predict(df_test5, verify_predictions=False)
rmse5 = str(mean_squared_error(df_test5['passed'], predictions5,
squared=False))

predictions10 = model10.predict(df_test10, verify_predictions=False)
rmse10 = str(mean_squared_error(df_test10['passed'], predictions10,
squared=False))

print(rmse5)
print(rmse10)

# share the RMSEs with us
send(rmse5, 1)
send(rmse10, 2)
```

```
Formula: passed~(1|category)+wa_num_subs_perc_correct

Family: binomial  Inference: parametric

Number of observations: 187  Groups: {'category': 5.0}

Log-likelihood: -125.311     AIC: 256.623

Random effects:

                 Name    Var    Std
category  (Intercept)  0.026  0.161

No random effect correlations specified

Fixed effects:

                          Estimate  2.5_ci  97.5_ci     SE     OR
OR_2.5_ci  \
(Intercept)                  0.516  -0.078    1.111  0.303  1.676
0.925
wa_num_subs_perc_correct    -0.287  -1.551    0.976  0.645  0.750
0.212

                          OR_97.5_ci   Prob  Prob_2.5_ci  Prob_97.5_ci
\
(Intercept)                    3.036  0.626        0.481         0.752

wa_num_subs_perc_correct       2.655  0.429        0.175         0.726


                          Z-stat  P-val Sig
```

```
(Intercept)                1.703  0.089   .
wa_num_subs_perc_correct  -0.445  0.656
Formula: passed~(1|category)+wa_num_subs_perc_correct

Family: binomial  Inference: parametric

Number of observations: 187  Groups: {'category': 5.0}

Log-likelihood: -125.330    AIC: 256.659

Random effects:

                 Name    Var    Std
category  (Intercept)  0.031  0.177

No random effect correlations specified

Fixed effects:

                          Estimate  2.5_ci  97.5_ci     SE     OR
OR_2.5_ci  \
(Intercept)                  0.308  -0.279    0.896  0.300  1.361
0.757
wa_num_subs_perc_correct     0.307  -1.190    1.804  0.764  1.359
0.304

                          OR_97.5_ci   Prob  Prob_2.5_ci  Prob_97.5_ci
\
(Intercept)                    2.449  0.576        0.431         0.710

wa_num_subs_perc_correct       6.075  0.576        0.233         0.859


                          Z-stat  P-val Sig
(Intercept)                1.029  0.303
wa_num_subs_perc_correct   0.402  0.688
0.4913802057143117
0.4840872132280756
Variable npt_config is not defined
Variable npt_config is not defined
```

Extension: if you still have time: can you improve the accuracy of the model by adding more features? Send us an explanation of why you have chosen the specific features along with the RMSE of your model.

```
# Explain briefly: what features are you adding and why?

exp = """This is an example discussion"""
```

```python
### Share it with us
send(exp, 3)
```

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

```python
# Define the model equations

# YOUR CODE HERE

# Fit the models
print(model5.fit())
print(model10.fit())

# predict on the test data for weeks 5, 10
predictions5 = model5.predict(df_test5, verify_predictions=False)
rmse5 = str(mean_squared_error(df_test5['passed'], predictions5,
squared=False))

predictions10 = model10.predict(df_test10, verify_predictions=False)
rmse10 = str(mean_squared_error(df_test10['passed'], predictions10,
squared=False))

print(rmse5)
print(rmse10)

# share the RMSEs with us
send(rmse5, 4)
send(rmse10, 5)
```

```
Formula: grade~(1|category)+wa_num_subs_perc_correct

Family: gaussian  Inference: parametric

Number of observations: 187 Groups: {'category': 5.0}

Log-likelihood: -295.857    AIC: 591.714

Random effects:

                 Name    Var    Std
category  (Intercept)  0.072  0.269
Residual               1.349  1.161

No random effect correlations specified

Fixed effects:

                          Estimate  2.5_ci  97.5_ci     SE       DF
T-stat  \
(Intercept)                  3.813   3.404    4.222  0.209   12.154
```

```
18.273
wa_num_subs_perc_correct     0.553  -0.160    1.266  0.364  182.752
1.520

                          P-val  Sig
(Intercept)                0.00  ***
wa_num_subs_perc_correct   0.13
Formula: grade~(1|category)+wa_num_subs_perc_correct

Family: gaussian Inference: parametric

Number of observations: 187 Groups: {'category': 5.0}

Log-likelihood: -293.932    AIC: 587.863

Random effects:

                 Name    Var    Std
category  (Intercept)  0.075  0.273
Residual               1.322  1.150

No random effect correlations specified

Fixed effects:

                          Estimate  2.5_ci  97.5_ci     SE       DF
T-stat  \
(Intercept)                  3.694   3.288    4.101  0.207   11.588
17.810
wa_num_subs_perc_correct     1.037   0.201    1.873  0.427  183.054
2.431

                          P-val  Sig
(Intercept)               0.000  ***
wa_num_subs_perc_correct  0.016    *
3.4999036181293426
3.4996060841843972

<Response [200]>
```

Lecture 4 - Student Notebook

Prelimilaries: Imports and stuff

We extended the data with extra features. The feature description is found here.

The features were calculated per week in the time_series_extended. The aggregated extended was computed by taking the mean of each feature per user across weeks.

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

from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score, balanced_accuracy_score
from sklearn.preprocessing import MinMaxScaler, normalize
from sklearn.linear_model import LogisticRegression

import statsmodels.api as sm
import statsmodels.formula.api as smf

from scipy.spatial.distance import pdist, cdist, squareform

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

Section 1: Pre-processing

Read the data

```
# Parse the aggregated data frame
df_lq = pd.read_csv('{}/aggregated_extended_fc.csv'.format(DATA_DIR))
ts = pd.read_csv('{}/time_series_extended_fc.csv'.format(DATA_DIR))
```

Clean the data

We remove inactive students that did not click during weekdays and weekend for the fist 5 weeks of the semester.

```
def remove_inactive_students(df, ts):
    df = df.fillna('NaN')

    #find all users weeks with 0 clicks on weekends and 0 clicks on 
weekdays during the first weeks of the semester
    df_first = ts[ts.week < 5]
```

```
    rows =
np.where(np.logical_and(df_first.ch_total_clicks_weekend==0,
df_first.ch_total_clicks_weekday == 0).to_numpy())[0]
    df_zero = df_first.iloc[rows,:]
    dropusers = np.unique(df_zero.user)

    ts = ts[ts.user.isin(dropusers)==False]
    df = df[df.user.isin(dropusers)==False]
    return df, ts

df_lq, ts = remove_inactive_students(df_lq, ts)
# print(df_lq.columns)

display(df_lq)
```

```
     user  ch_num_sessions  ch_time_in_prob_sum  ch_time_in_video_sum
\
1       1              3.4               1698.4                9227.8

2       2              5.3               2340.6               10801.3

4       4              2.5               3787.3                7040.0

5       5              2.5               2568.9                3718.9

6       6              4.2               5475.2                9711.6

..    ...              ...                  ...                   ...

272   281              2.2               2600.3                3071.8

276   285              1.8               2111.4                2130.4

277   286              1.9               2224.8                2408.1

281   291              4.0               6898.3                5657.2

283   293              3.5               8127.5                 113.4


     ch_ratio_clicks_weekend_day  ch_total_clicks_weekend  \
1                       0.567500                      4.0
2                      26.562274                     94.6
4                       1.543889                     58.4
5                       0.009677                     31.2
6                       0.476263                    105.1
..                           ...                      ...
272                     0.213467                     20.6
276                     0.236626                      7.0
```

```
277                    0.326239                    20.5
281                    0.080266                   131.5
283                    0.632304                    28.9

     ch_total_clicks_weekday  ch_time_sessions_mean
ch_time_sessions_std  \
1                    179.4           3068.720238
1257.504407
2                    129.2           1750.289268
1024.134043
4                     64.9           3373.908333
1363.320365
5                     50.8           1753.647500
1190.793589
6                     46.7          20410.677619
1561.548415
..                     ...                   ...
...
272                  122.6           2429.026667
336.303590
276                   21.2           1378.954762
544.967013
277                   46.7           1744.663333
664.335107
281                   15.7           2323.840952
1623.425778
283                   20.6           7963.627500
1001.514794

     bo_delay_lecture  ...  la_weekly_prop_watched_mean  \
1        -2883.367738  ...                     0.748868
2        10027.216667  ...                     0.354487
4         -914.633333  ...                     0.030000
5        12406.195238  ...                     0.330000
6       -13723.616667  ...                     0.080000
..                ...  ...                          ...
272     -69623.272700  ...                     0.616667
276      23125.010000  ...                     0.352244
277       4812.464286  ...                     0.261205
281      -5626.100000  ...                     0.033333
283          0.000000  ...                     0.000000

     la_weekly_prop_interrupted_mean
la_weekly_prop_interrupted_std  \
1                           0.074683                             0.0

2                           0.026667                             0.0

4                           0.000000                             0.0
```

```
5                             0.097619                             0.0

6                             0.010000                             0.0

..                                 ...                             ...

272                           0.027894                             0.0

276                           0.057692                             0.0

277                           0.014286                             0.0

281                           0.000000                             0.0

283                           0.000000                             0.0


     la_weekly_prop_replayed_mean  la_weekly_prop_replayed_std  \
1                        0.066456                          0.0
2                        0.059915                          0.0
4                        0.020000                          0.0
5                        0.047619                          0.0
6                        0.000000                          0.0
..                            ...                          ...
272                      0.015385                          0.0
276                      0.047619                          0.0
277                      0.000000                          0.0
281                      0.000000                          0.0
283                      0.000000                          0.0

     la_frequency_action_video_play  grade  gender       category
year
1                          0.332424   4.50       M  Suisse.Autres  Y2-
2018-19
2                          0.284407   5.25       M     Suisse.PAM  Y2-
2018-19
4                          0.199775   4.75       F         France  Y2-
2018-19
5                          0.261962   4.00       M     Suisse.PAM  Y2-
2018-19
6                          0.229185   4.25       F         France  Y2-
2018-19
..                              ...    ...     ...            ...
...
272                        0.205969   5.75       M     Suisse.PAM  Y3-
2019-20
276                        0.027091   4.00       F         France  Y3-
2019-20
277                        0.137474   4.50       M         France  Y3-
```

```
2019-20
281                        0.173431   5.50       M         France  Y3-
2019-20
283                        0.034080   5.25       M         France  Y3-
2019-20

[234 rows x 38 columns]
```

```
display(ts)
```

```
      week  user  ch_num_sessions  ch_time_in_prob_sum
ch_time_in_video_sum  \
1        0     1              7.0                326.0
15525.0
2        0     2              4.0               1224.0
12209.0
4        0     4              4.0               1294.0
12037.0
5        0     5              2.0               1324.0
4440.0
6        0     6              3.0               1773.0
14462.0
...    ...   ...              ...                  ...
...
2864     9   281              0.0                  0.0
0.0
2868     9   285              0.0                  0.0
0.0
2869     9   286              0.0                  0.0
0.0
2873     9   291              0.0                  0.0
0.0
2875     9   293              1.0                  0.0
0.0

      ch_ratio_clicks_weekend_day  ch_total_clicks_weekend  \
1                        5.675000                     40.0
2                      258.000000                      1.0
4                        0.328571                    140.0
5                        0.000000                    119.0
6                        1.411765                    102.0
...                           ...                      ...
2864                     0.000000                      0.0
2868                     0.000000                      0.0
2869                     0.000000                      0.0
2873                     0.000000                      0.0
2875                     0.000000                     14.0

      ch_total_clicks_weekday  ch_time_sessions_mean
ch_time_sessions_std  \
```

```
1                        227.0           1931.285714
1648.472515
2                        258.0           2780.250000
2297.110400
4                         46.0           3043.750000
344.374342
5                          0.0           2882.000000
2827.000000
6                        144.0           5411.666667
2459.581039
...                        ...                   ...
...
2864                       0.0              0.000000
0.000000
2868                       0.0              0.000000
0.000000
2869                       0.0              0.000000
0.000000
2873                       0.0              0.000000
0.000000
2875                       0.0          62893.000000
0.000000

      ...  la_seek_len_std  la_pause_dur_std  la_time_speeding_up_mean
\
1     ...       146.564097        188.175709                 65.173554

2     ...        33.419147         39.702700                  0.000000

4     ...       159.612354        228.274335                 67.941176

5     ...        99.684654        182.336877                 51.760000

6     ...       116.878539        135.989183                  0.000000

...   ...              ...               ...                       ...

2864  ...         0.000000          0.000000                  0.000000

2868  ...         0.000000          0.000000                  0.000000

2869  ...         0.000000          0.000000                  0.000000

2873  ...         0.000000          0.000000                  0.000000

2875  ...         0.000000          0.000000                  0.000000


      la_time_speeding_up_std  la_weekly_prop_watched_mean  \
```

```
1                  150.807752                           0.6
2                    0.000000                           0.6
4                  111.514074                           0.3
5                  146.965446                           0.0
6                    0.000000                           0.6
...                       ...                           ...
2864                 0.000000                           0.0
2868                 0.000000                           0.0
2869                 0.000000                           0.0
2873                 0.000000                           0.0
2875                 0.000000                           0.0

      la_weekly_prop_interrupted_mean  la_weekly_prop_interrupted_std
\
1                                 0.1                             0.0

2                                 0.0                             0.0

4                                 0.0                             0.0

5                                 0.0                             0.0

6                                 0.1                             0.0

...                               ...                             ...

2864                              0.0                             0.0

2868                              0.0                             0.0

2869                              0.0                             0.0

2873                              0.0                             0.0

2875                              0.0                             0.0


      la_weekly_prop_replayed_mean  la_weekly_prop_replayed_std  \
1                              0.0                          0.0
2                              0.0                          0.0
4                              0.2                          0.0
5                              0.0                          0.0
6                              0.0                          0.0
...                            ...                          ...
2864                           0.0                          0.0
2868                           0.0                          0.0
2869                           0.0                          0.0
2873                           0.0                          0.0
2875                           0.0                          0.0
```

```
      la_frequency_action_video_play
1                           0.400749
2                           0.370656
4                           0.252688
5                           0.411765
6                           0.353659
...                              ...
2864                        0.000000
2868                        0.000000
2869                        0.000000
2873                        0.000000
2875                        0.000000

[2340 rows x 35 columns]
```

Prepare data for classification

Add a pass/fail label

```
# We first add a column to the dataframe containing the outcome variable
# compute pass/fail label
df_lq['passed'] = df_lq.grade >= 4
df_lq['passed'] = df_lq['passed'].astype(int)
```

Remove "bad" features and Split Data

```
# We then split the data in a train-test split (stratified by the outcome variable)
X = df_lq.drop(['user','grade', 'gender', 'category', 'year', 'passed'], axis=1)
y = df_lq['passed']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=0, stratify=y) # split train and validation data set
```

Print pass/fail proportions

```
# The class proportions in train and validation sets are the same, thanks to the stratification on y
print(y_train.value_counts(normalize=True))
print(y_val.value_counts(normalize=True))
```

```
1    0.604278
0    0.395722
Name: passed, dtype: float64
1    0.595745
0    0.404255
Name: passed, dtype: float64
```

Define Evaluation Metrics (will see later in the slides)

```
def compute_scores(clf, X_train, y_train, X_test, y_test, roundnum = 3):
```

```python
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy =  balanced_accuracy_score(y_test, y_pred)

    y_pred_proba = clf.predict_proba(X_test)[:,1]
    auc = roc_auc_score(y_test, y_pred_proba)

    return round(accuracy,roundnum), round(auc,roundnum)
```

Section 2: Decision Trees

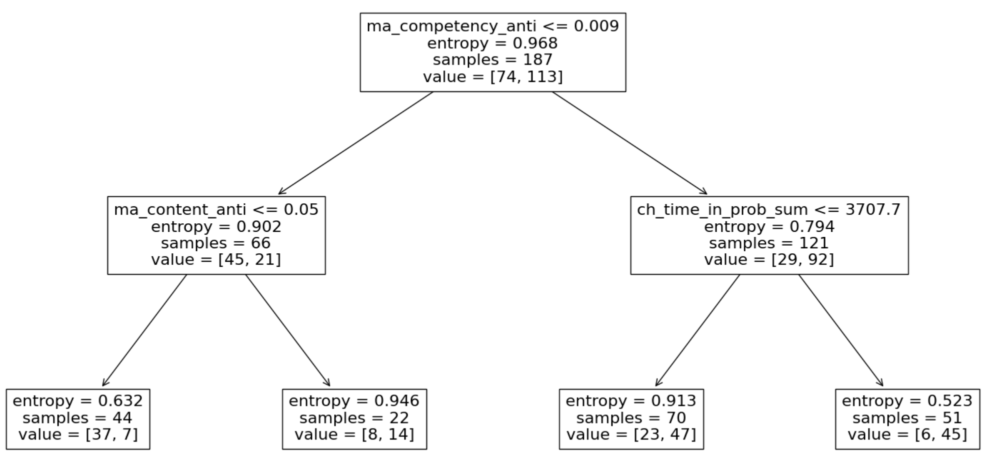
Compute a decision tree of max depth 2 over all the features

```python
clf = tree.DecisionTreeClassifier(max_depth=2, random_state=0, 
criterion='entropy')
accuracy, auc = compute_scores(clf, X_train, y_train, X_val, y_val)
print("Decision tree. Balanced Accuracy = {}, AUC = 
{}".format(accuracy, auc))
```

```
Decision tree. Balanced Accuracy = 0.577, AUC = 0.602
```

Visualize the decision tree

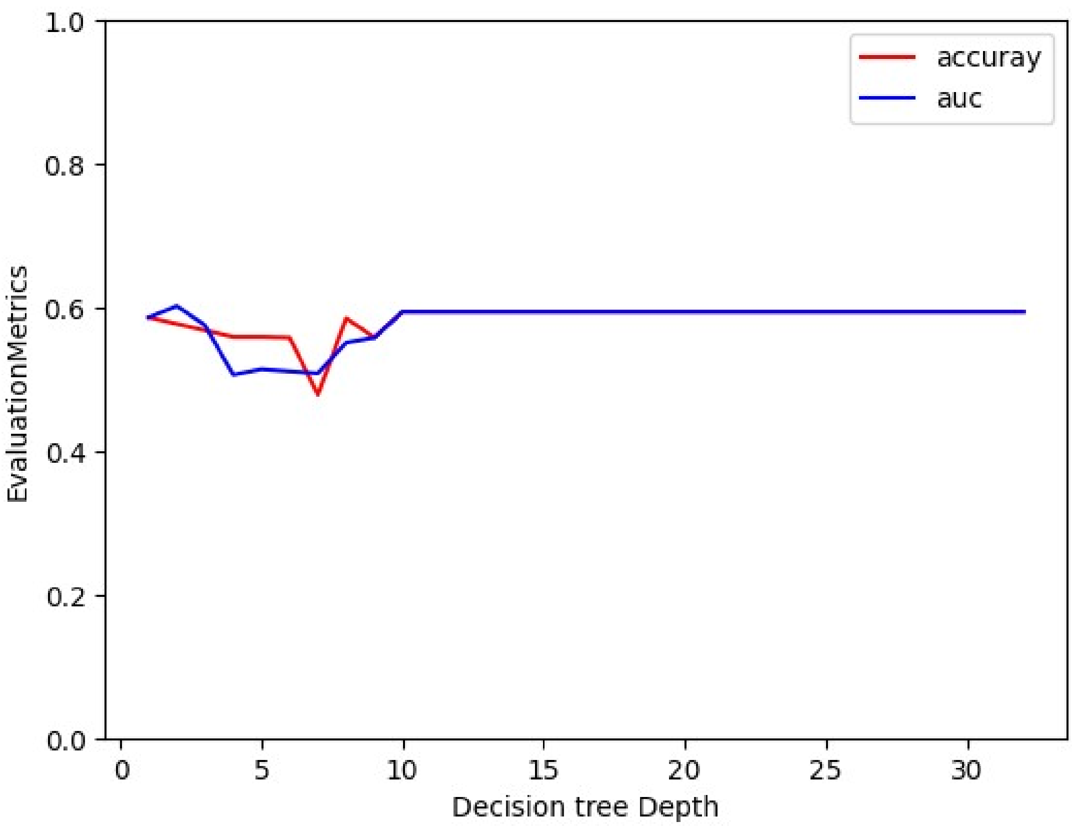
```python
plt.figure(figsize=(20, 10))
tree.plot_tree(clf, feature_names=X_train.columns);
```



Does depth improves perfromance ?

```python
# We can change the max depth
accuracy_list = []
auc_list = []
for depth in range(1,len(X_train.columns)):
    clf = tree.DecisionTreeClassifier(max_depth=depth, random_state=0, 
criterion='entropy')
    accuracy, auc = compute_scores(clf, X_train, y_train, X_val, 
```

```
y_val)
    accuracy_list.append(accuracy)
    auc_list.append(auc)
    # print("Decision tree. Depth = {}, Balanced Accuracy = {}, AUC = 
{}".format(depth, accuracy, auc))
x = list(range(1,len(X_train.columns)))
plt.plot(x, accuracy_list, 'r', label = 'accuray')
plt.plot(x, auc_list, 'b', label = 'auc')
plt.xlabel("Decision tree Depth")
plt.ylabel("EvaluationMetrics")
plt.ylim([0,1])
plt.legend()
plt.show()
```



Section 3: Random Forests

Next, we will use a random forest classifier instead of a decision tree.

```
rf = RandomForestClassifier(n_estimators=100, random_state=0, 
criterion='entropy') # create a Random Forest
accuracy, auc = compute_scores(rf, X_train, y_train, X_val, y_val)
print("Random Forest. Balanced Accuracy = {}, AUC = 
{}".format(accuracy, auc))
```

```
Random Forest. Balanced Accuracy = 0.507, AUC = 0.603
```

For a single tree, in fact, keeping a low depth is necessary to avoid overfitting and to reduce the variance. Random forests, instead, can have a higher depth, and consequently a lower bias, since the variance is reduced in the aggregation step.

In this case, decision trees seem to perform better than random forests. A reason for this behavior could be that the single tree is already very "stable", i.e. it will change a little in response to little changes in the data. If this was the case, the submodels in the ensemble forest would be all very similar to the single tree, if they were allowed to choose among all the features at every split. Since, though, only a random subset of features is considered at each split, some subtrees would choose bad splits and have overall bad performances.

Section 4: K-Nearest Neighbors

We only use the euclidean distance since all our features are numerical

```
feature = 'ch_time_in_prob_sum'

# Compute the pairwise distance matrix for all the elements of the
training set
X_train_dist = squareform(pdist(X_train[feature].to_numpy().reshape(-
1,1), metric='euclidean'))

# Compute the distance between all elements of the training set and of
the validation set
X_val_dist = cdist(X_val[feature].to_numpy().reshape(-1,1),
X_train[feature].to_numpy().reshape(-1,1), metric='euclidean')

X_train_dist
```

```
array([[   0. ,  181.9,  932.5, ..., 1821.3, 1884.6, 2220.8],
       [ 181.9,    0. ,  750.6, ..., 1639.4, 2066.5, 2402.7],
       [ 932.5,  750.6,    0. , ...,  888.8, 2817.1, 3153.3],
       ...,
       [1821.3, 1639.4,  888.8, ...,    0. , 3705.9, 4042.1],
       [1884.6, 2066.5, 2817.1, ..., 3705.9,    0. ,  336.2],
       [2220.8, 2402.7, 3153.3, ..., 4042.1,  336.2,    0. ]])
```

```
print('Training set size:', X_train.shape)
print('Validation set size:', X_val.shape)
print('Training pairwise distances size:', X_train_dist.shape)
print('Validation distances size:', X_val_dist.shape)
```

```
Training set size: (187, 33)
Validation set size: (47, 33)
Training pairwise distances size: (187, 187)
Validation distances size: (47, 187)
```

```
knn = KNeighborsClassifier(n_neighbors=5, metric='precomputed')

accuracy, auc = compute_scores(knn, X_train_dist, y_train, X_val_dist, y_val)
print("k-nearest neighbors. Balanced Accuracy = {}, AUC = {}".format(accuracy, auc))
```

```
k-nearest neighbors. Balanced Accuracy = 0.533, AUC = 0.548
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/neighbors/_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Section 5: Logistic regression

We normalize the data data using the MinMaxScaler such that all the features are on the same scale.

```
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_val_scaled  = scaler.transform(X_val)

clf = LogisticRegression(random_state=0)
accuracy, auc = compute_scores(clf, X_train_scaled, y_train, X_val_scaled, y_val)
print("Logistic Regression. Balanced Accuracy = {}, AUC = {}".format(accuracy, auc))
```

```
Logistic Regression. Balanced Accuracy = 0.577, AUC = 0.626
```

Section 6: Time Series - Your Turn

Build a classifier that can predict whether students pass the course after half of the course (5 weeks). You will need to use the data frame **ts** for this task. You can use kNN, RF, or decision tree. Train your model on the training data and predict on the test data.

- Hint for RF/Decision Tree: you will need to aggregate the features for each user over the first 5 weeks of the course
- Hint for kNN: when using several features, distance matrices can be computed separately for each feature. They can then be summed up to a overall distance

matrix. Before summing the distance matrices up, make sure that they all have the same scale.

```
import requests

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

npt_config = {
    'session_name': 'lecture-04',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}

# Consider only data up to the 5th week
ts = ts[ts.week <= 5]

# Train-test split done on the users, so that all the rows
corresponding to one user go into the same set.
users = ts.user.unique()
y = df_lq.passed
users_train, users_val, y_train, y_val = train_test_split(users, y,
test_size=0.2, random_state=0, stratify=y)

X_train = ts[ts.user.isin(users_train)]
X_val = ts[ts.user.isin(users_val)]

# Sort indexes to make label arrays consistent with the data
y_train = y_train.sort_index()
y_val = y_val.sort_index()
```

Decision Tree/Random Forest

```
## AGGREGATION
X_train = X_train.groupby(['user']).mean()
X_train['user'] = X_train.index
X_val = X_val.groupby(['user']).mean()
X_val['user'] = X_val.index

# Train the classifier
rf = RandomForestClassifier(n_estimators=100, random_state=0,
criterion='entropy') # create a Random Forest
accuracy, auc = compute_scores(rf, X_train, y_train, X_val, y_val)
print("Random Forest. Balanced Accuracy = {}, AUC =
{}".format(accuracy, auc))
```

```
Random Forest. Balanced Accuracy = 0.568, AUC = 0.581
```

```
# Compute accuracy and AUC of the classifier
accuracy, auc = #your code here
```

```
result = "My Classifier (Decision Tree/Random Forest). Balanced
Accuracy = {}, AUC = {}".format(accuracy, auc)
print(result)

#send(result, 1)
```

```
  Input In [116]
    accuracy, auc = #your code here
                    ^
SyntaxError: invalid syntax
```

K-Nearest Neighbors (harder challenge)

```
# Compute pairwise distance matrix for each feature f. You can choose
the features yourself
from sklearn.preprocessing import normalize

# Compute the pairwise distance matrix for all the elements of the
training set
X_train_dist1 =
squareform(pdist(X_train["ch_num_sessions"].to_numpy().reshape(-1,1),
metric='euclidean'))

# Compute the distance between all elements of the training set and of
the validation set
X_val_dist1 = cdist(X_val["ch_num_sessions"].to_numpy().reshape(-1,1),
X_train["ch_num_sessions"].to_numpy().reshape(-1,1),
metric='euclidean')
X_val_dist1 = normalize(X_val_dist1, axis=1, norm='l1')

X_train_dist1 = normalize(X_train_dist1, axis=1, norm='l1')
X_train_dist1
```

```
array([[0.        , 0.01085209, 0.00522508, ..., 0.01045016,
0.00924437,
        0.00602894],
       [0.00941094, 0.        , 0.00487975, ..., 0.00034855,
0.00139421,
        0.00418264],
       [0.0089717 , 0.00966184, 0.        , ..., 0.0089717 ,
0.00690131,
        0.00138026],
       ...,
       [0.00960828, 0.00036955, 0.00480414, ..., 0.        ,
0.00110865,
        0.00406504],
       [0.01011879, 0.00175979, 0.00439947, ..., 0.00131984, 0.
,
        0.00351958],
```

```
       [0.01006036, 0.00804829, 0.00134138, ..., 0.0073776 ,
0.00536553,
        0.        ]])
```

```
# Compute the pairwise distance matrix for all the elements of the
training set
X_train_dist2 =
squareform(pdist(X_train["ch_time_in_prob_sum"].to_numpy().reshape(-
1,1), metric='euclidean'))

# Compute the distance between all elements of the training set and of
the validation set
X_val_dist2 = cdist(X_val["ch_time_in_prob_sum"].to_numpy().reshape(-
1,1), X_train["ch_time_in_prob_sum"].to_numpy().reshape(-1,1),
metric='euclidean')
X_val_dist2 = normalize(X_val_dist2, axis=1, norm='l1')

X_train_dist2 = normalize(X_train_dist2, axis=1, norm='l1')
X_train_dist2
```

```
array([[0.00000000e+00, 2.06485160e-03, 8.11324131e-03, ...,
        1.35817134e-03, 8.10591149e-05, 9.02601868e-03],
       [2.02504818e-03, 0.00000000e+00, 9.98189339e-03, ...,
        6.93057835e-04, 2.10454475e-03, 1.08770755e-02],
       [4.79762865e-03, 6.01864388e-03, 0.00000000e+00, ...,
        5.60076040e-03, 4.74969570e-03, 5.39755532e-04],
       ...,
       [1.35681879e-03, 7.05976507e-04, 9.46198043e-03, ...,
        0.00000000e+00, 1.43779718e-03, 1.03738488e-02],
       [8.09606759e-05, 2.14330470e-03, 8.02242783e-03, ...,
        1.43748264e-03, 0.00000000e+00, 8.93409672e-03],
       [4.95685190e-03, 6.09081405e-03, 5.01273310e-04, ...,
        5.70272385e-03, 4.91233637e-03, 0.00000000e+00]])
```

```
# Sum up the distance matrices (don't forget the scaling)
X_train_dist = np.array(X_train_dist1) + np.array(X_train_dist2)
X_val_dist = np.array(X_val_dist1) + np.array(X_val_dist2)

# Compute the AUC and accuracy for kNN
knn = KNeighborsClassifier(n_neighbors=5, metric='precomputed')
accuracy, auc = compute_scores(knn, X_train_dist, y_train, X_val_dist,
y_val)

result = "K-Nearest Neighbors. Balanced Accuracy = {}, AUC =
{}".format(accuracy, auc)
print(result)

#send(result, 2)
```

```
K-Nearest Neighbors. Balanced Accuracy = 0.604, AUC = 0.604
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/neighbors/
_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this
behavior will change: the default value of `keepdims` will become
False, the `axis` over which the statistic is taken will be
eliminated, and the value None will no longer be accepted. Set
`keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Lecture 5 - Student Notebook

We first load and clean the data.

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

from sklearn.model_selection import train_test_split, cross_validate,
GridSearchCV, ParameterGrid
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score,
classification_report, confusion_matrix, auc
from sklearn.utils import resample

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

# Parse the aggregated student data frame.
# This data is from an EPFL Linear Algebra flipped classroom. df_lq is
aggregated features for the last week of student performance.
# ts represents the students' time series features.

df_lq = pd.read_csv('{}/aggregated_extended_fc.csv'.format(DATA_DIR))
ts = pd.read_csv('{}/time_series_extended_fc.csv'.format(DATA_DIR))

def remove_inactive_students(df, ts):
    """
    Filter the students (removing the ones that are inactive) to
proceed with analysis on students who have participated during the
entire class.
    Inputs: df, ts
    Outputs: filtered df, ts
    """
    # Fill all NaNs with strings to make them easier to process
    df = df.fillna('NaN')

    # Find all users weeks with 0 clicks on weekends and 0 clicks on
weekdays during the first weeks of the semester
    df_first = ts[ts.week < 5]
    rows =
np.where(np.logical_and(df_first.ch_total_clicks_weekend==0,
df_first.ch_total_clicks_weekday==0).to_numpy())[0]
    df_zero = df_first.iloc[rows, :]
    dropusers = np.unique(df_zero.user)

    # Drop users with no activity
    ts = ts[~ts.user.isin(dropusers)]
```

```
    df = df[~df.user.isin(dropusers)]
    return df, ts

df_lq, ts = remove_inactive_students(df_lq, ts)
```

The `compute_scores` function computes the performance of classifiers with accuracy + AUC. We will use this evaluation function for all our experiments.

```
def compute_scores(clf, X_train, y_train, X_test, y_test, roundnum=3, 
report=False):
    """
    Train clf (binary classification) model on X_train and y_train, 
predict on X_test. Evaluate predictions against ground truth y_test.
    Inputs: clf, training set (X_train, y_train), test set (X_test, 
y_test)
    Inputs (optional): roundnum (number of digits for rounding 
metrics), report (print scores)
    Outputs: accuracy, AUC
    """
    # Fit the clf predictor (passed in as an argument)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    # Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    # Calculate roc AUC score
    AUC = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1])

    # Print classification results
    if report:
        print(classification_report(y_test, y_pred))

    return round(accuracy, roundnum), round(AUC, roundnum)
```

We compute the pass/fail label of the students in the dataframe to use for the experiments. We will use the aggregated dataframe (df_lq) for all our experiments. If students have a grade higher than or equal to 4, they have passed the class.

```
df_lq['passed'] = (df_lq.grade >= 4).astype(int)
```

We are interested in model selection and assessment. We will use a random forest model for all our evaluations. For our evaluations, we will investigate behavioral features only.

```
# Filter out demographic features
features = [x for x in df_lq.columns if x not in ['user', 'week', 
'grade', 'gender', 'category', 'year', 'passed']]
print(features)
```

```
['ch_num_sessions', 'ch_time_in_prob_sum', 'ch_time_in_video_sum', 
'ch_ratio_clicks_weekend_day', 'ch_total_clicks_weekend',
```

```
'ch_total_clicks_weekday', 'ch_time_sessions_mean',
'ch_time_sessions_std', 'bo_delay_lecture', 'bo_reg_peak_dayhour',
'bo_reg_periodicity_m1', 'ma_competency_strength',
'ma_competency_anti', 'ma_content_anti', 'ma_student_shape',
'ma_student_speed', 'mu_speed_playback_mean',
'mu_frequency_action_relative_video_pause', 'wa_num_subs',
'wa_num_subs_correct', 'wa_num_subs_avg', 'wa_num_subs_perc_correct',
'la_pause_dur_mean', 'la_seek_len_std', 'la_pause_dur_std',
'la_time_speeding_up_mean', 'la_time_speeding_up_std',
'la_weekly_prop_watched_mean', 'la_weekly_prop_interrupted_mean',
'la_weekly_prop_interrupted_std', 'la_weekly_prop_replayed_mean',
'la_weekly_prop_replayed_std', 'la_frequency_action_video_play']

# Only keep behavioral features in X.
X = df_lq[features]

# Our binary indicator variable is based on our evaluation criteria:
pass/fail.
y = df_lq['passed']
```

Your Turn 1: Model Assessment

In a first experiment, we are interested in assessing the generalizability of the trained model on to new data. We use two different methods to do so: a train-test split and a cross validation. Run the two methods and assess their accuracy/AUC:

- What can you observe?
- Where do the differences come from?

Train-Test Split

We split the data in a train-test split (stratified by the outcome variable) and obtain the accuracy and AUC.

```
# The train-test split is 80:20 (as shown by the 0.2 test_size
argument).
# We choose a random_state to replicate the results in the same split
every time we run this notebook.
# The stratify argument ensures a proportionate number of passes/fails
are in the training set and the test set.

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Let's initialize a RandomForestClassifier to make our model
predictions.

clf = RandomForestClassifier(random_state=42)

# We can use our compute_scores function to evaluate the results of
our train-test split classifier.
```

```
accuracy, AUC = compute_scores(clf, X_train, y_train, X_test, y_test)

print(f'Accuracy for train-test setting: {accuracy}')
print(f'AUC for train-test setting: {AUC}')
```

```
Accuracy for train-test setting: 0.723
AUC for train-test setting: 0.723
```

Cross Validation

We use a 10-fold cross validation to obtain accuracy and AUC.

```
# Initialize a new Random Forest predictor for our cross-validation
comparison.

clf = RandomForestClassifier(random_state=42)

# With the cross_validate function, the SciKit Learn library
automatically uses stratification across folds with the "cv" argument.

# In the background, it's using the StratifiedKFold function with 10
folds.
# We pass in our desired metrics ("accuracy", "roc_auc") for
evaluation in the "scoring" argument.

scores = cross_validate(clf, X, y, cv=3, scoring=['accuracy',
'roc_auc'])

print(f'Mean accuracy with cross-validation:
{scores["test_accuracy"].mean():.3f}')
print(f'Mean AUC with cross-validation:
{scores["test_roc_auc"].mean():.3f}')
```

```
Mean accuracy with cross-validation: 0.667
Mean AUC with cross-validation: 0.660
```

Your solution 1

Write your answers in the following cell:

```
import requests

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

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

```
# YOUR TURN: what differences can you observe in the metrics
(accuracy, AUC) between train-test and cross validation? Where do
these differences come from?

### Share the answer with us
cv_tts = ""
send(cv_tts, 1)
```

```
Your name:  Paola
```

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

Your Turn 2: Model Selection

Of course, when training ML models, we want to tune their hyperparameters in order to optimize the performance. In order to tune the hyperparameters of a model, we need to do further splits of our data set. In the following, we present an incorrect example. Your task is to:

- Explain why it is incorrect.
- Describe how it could be fixed.

```
# We compute a grid search across the following parameter space
parameters = {
    'n_estimators': [20, 50, 100],
    'criterion': ['entropy', 'gini'],
    'max_depth': np.arange(3, 7),
    'min_samples_split': [2],
    'min_samples_leaf': [1],
}

# Perform 10-fold cross-validation to identify the best
hyperparameters, selecting the ones with the highest accuracy
clf = GridSearchCV(RandomForestClassifier(random_state=42),
parameters, cv=10, scoring=['accuracy', 'roc_auc'], refit='accuracy')
clf.fit(X, y)
```

```
GridSearchCV(cv=10, estimator=RandomForestClassifier(random_state=42),
             param_grid={'criterion': ['entropy', 'gini'],
                         'max_depth': array([3, 4, 5, 6]),
                         'min_samples_leaf': [1], 'min_samples_split':
[2],
                         'n_estimators': [20, 50, 100]},
             refit='accuracy', scoring=['accuracy', 'roc_auc'])
```

```
clf.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_leaf': 1,
```

```
 'min_samples_split': 2,
 'n_estimators': 50}
```

```
accuracy = clf.cv_results_['mean_test_accuracy'][clf.best_index_]
AUC = clf.cv_results_['mean_test_roc_auc'][clf.best_index_]

print(f'Accuracy for train-validation-test setting: {accuracy:.3f}')
print(f'AUC for train-validation-test setting: {AUC:.3f}')
```

```
Accuracy for train-validation-test setting: 0.726
AUC for train-validation-test setting: 0.707
```

Your Solution 2

```
# YOUR TURN: Explain why it is incorrect.
answer = ""
send(answer, 2)
```

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

```
# YOUR TURN: Describe how it could be fixed.
answer = ""
send(answer, 3)
```

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

Lecture 6 - Student Notebook

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. The dataset involves 4,217 middle-school students practicing an electronic tutor that teaches and evaluates students in grade-school math, with a total of 525,534 trials. The student data are in a comma-delimited text file with one row per trial. The columns should correspond to a trial's user id, the order id (timestamp), the skill name, and and whether the student produced a correct response in the trial. More information on the platform can be found here.

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.

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

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
```

To keep things simpler for demonstration purposes, we will focus on the following 6 skills in this lecture:
`'Circle Graph', 'Venn Diagram', 'Mode', 'Division Fractions', 'Finding Percents', 'Area Rectangle'`

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

BKT Models - Training & Prediction

We will use a train-test setting (20% of students in the test set). The `create_iterator` function creates 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, we appropriately initialize a scikit-learn's GroupShuffleSplit iterator with 80% training set size and non-overlapping groups, then return the iterator.

```
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)
```

Next, we train a BKT model for each skill on the training data set and then predict on the test data set. We obtain `df_preds`, a data frame containing the predictions for each user and skill in the test data set. We output the overall RMSE and AUC scores.

```
rmse_bkt, auc_bkt = [], []
df_preds = pd.DataFrame()
# Train a BKT model for each skill
for skill in skills_subset:
    print("--", skill, "--")
    skill_data = data[data['skill_name'] == skill]
    for iteration, (train_index, test_index) in
enumerate(create_iterator(skill_data)):
        # Split data in training and test sets
        X_train, X_test = skill_data.iloc[train_index],
skill_data.iloc[test_index]
        # Initialize and fit the model
        model = Model(seed=0)
        %time model.fit(data=X_train)
        # Compute predictions
        preds = model.predict(data=X_test)[['user_id', 'skill_name',
'correct', 'correct_predictions']]
        df_preds = df_preds.append(preds)

# Print the the resulting dataframe
display(df_preds)

# Compute overall RMSE and AUC
rmse = mean_squared_error(df_preds.correct,
df_preds.correct_predictions, squared = False)
AUC = roc_auc_score(df_preds.correct, df_preds.correct_predictions)
print('RMSE:', rmse, 'AUC:', AUC)
```

```
-- Circle Graph --
CPU times: user 356 ms, sys: 236 µs, total: 356 ms
Wall time: 114 ms
```

```
/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)
```

```
-- Venn Diagram --
CPU times: user 2.25 s, sys: 13.1 ms, total: 2.26 s
Wall time: 1.18 s
-- Mode --
CPU times: user 152 ms, sys: 0 ns, total: 152 ms
Wall time: 88 ms

/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)
/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

-- Division Fractions --
CPU times: user 1.22 s, sys: 2.45 ms, total: 1.22 s
Wall time: 592 ms
-- Finding Percents --

/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

CPU times: user 544 ms, sys: 0 ns, total: 544 ms
Wall time: 283 ms
-- Area Rectangle --

/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

CPU times: user 785 ms, sys: 5.46 ms, total: 790 ms
Wall time: 391 ms

/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

        user_id      skill_name  correct  correct_predictions
3969      64525    Circle Graph        1              0.47467
3970      64525    Circle Graph        1              0.64102
3971      64525    Circle Graph        1              0.68944
3972      64525    Circle Graph        0              0.69915
3973      64525    Circle Graph        1              0.69539
...         ...             ...      ...                  ...
337153    96264  Area Rectangle        1              0.89258
337154    96264  Area Rectangle        1              0.97978
```

```
337159    96270  Area Rectangle        1              0.89258
337167    96292  Area Rectangle        1              0.89258
337169    96295  Area Rectangle        1              0.89258

[9551 rows x 4 columns]

RMSE: 0.3565418731257616 AUC: 0.865118941112136
```

Your Turn - Training & Prediction

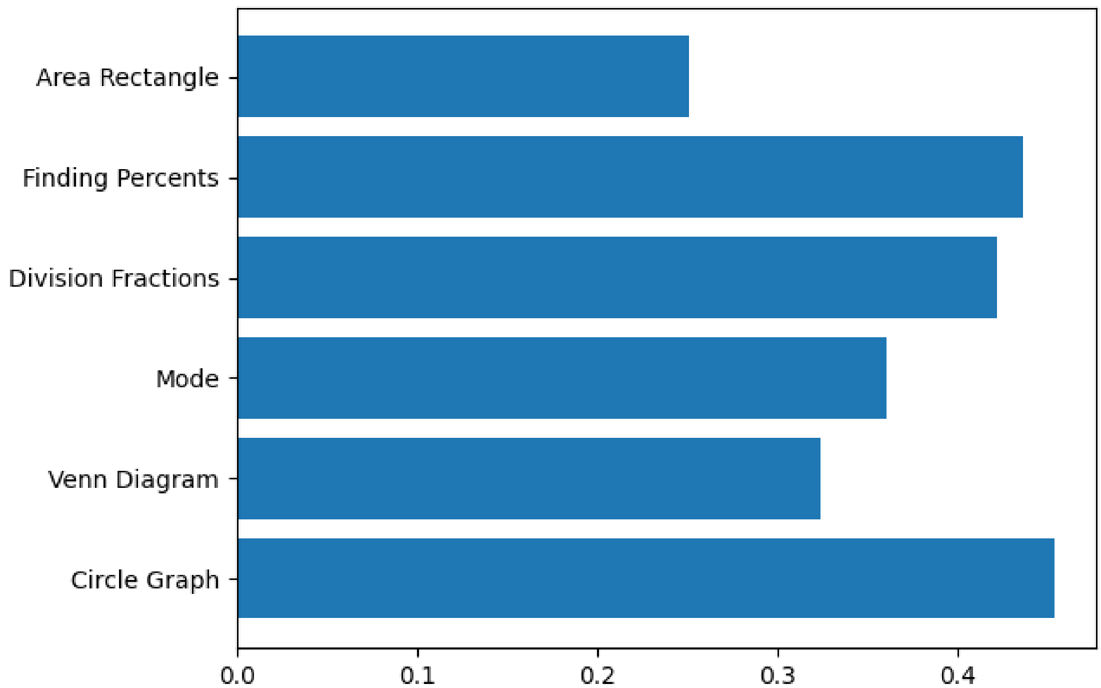
Next, we assume that the RMSE and AUC might differ depending on the skill. Your task is to:

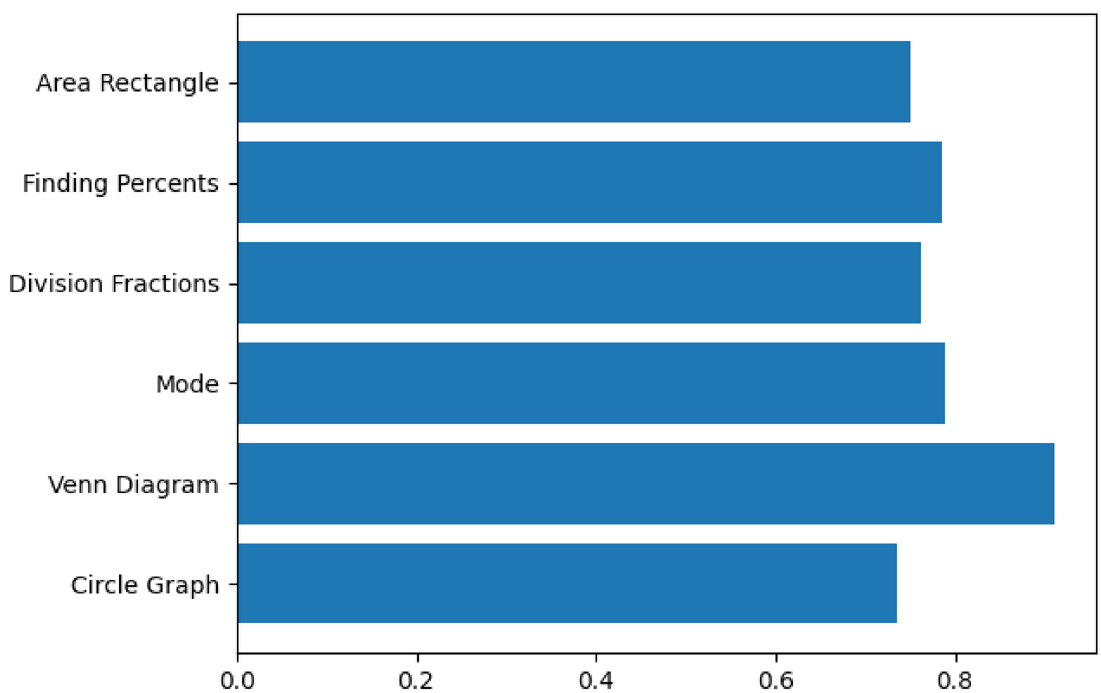
1. Compute one of the metrics (RMSE or AUC) separately for each skill.
2. Compute the mean of the selected metric (+ standard deviation) over all skills.
3. Create a visualization that displays: the mean of the metric (+ standard deviation) over all skills *and* the metric per skill.
4. Discuss your findings.

```
import requests

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

npt_config = {
    'session_name': 'lecture-06',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}

# YOUR TURN: Your code for computing the metrics goes here
rmse_bkt, auc_bkt = [], []
df_preds = pd.DataFrame()
# Train a BKT model for each skill
for skill in skills_subset:
    print("--", skill, "--")
    skill_data = data[data['skill_name'] == skill]
    for iteration, (train_index, test_index) in
enumerate(create_iterator(skill_data)):
        # Split data in training and test sets
        X_train, X_test = skill_data.iloc[train_index],
skill_data.iloc[test_index]
        # Initialize and fit the model
        model = Model(seed=0)
        %time model.fit(data=X_train)
        # Compute predictions
        preds = pd.DataFrame(model.predict(data=X_test)[['user_id',
'skill_name', 'correct', 'correct_predictions']])

        rmse_bkt.append(mean_squared_error(preds.correct,
preds.correct_predictions, squared = False))
        auc_bkt.append(roc_auc_score(preds.correct,
```

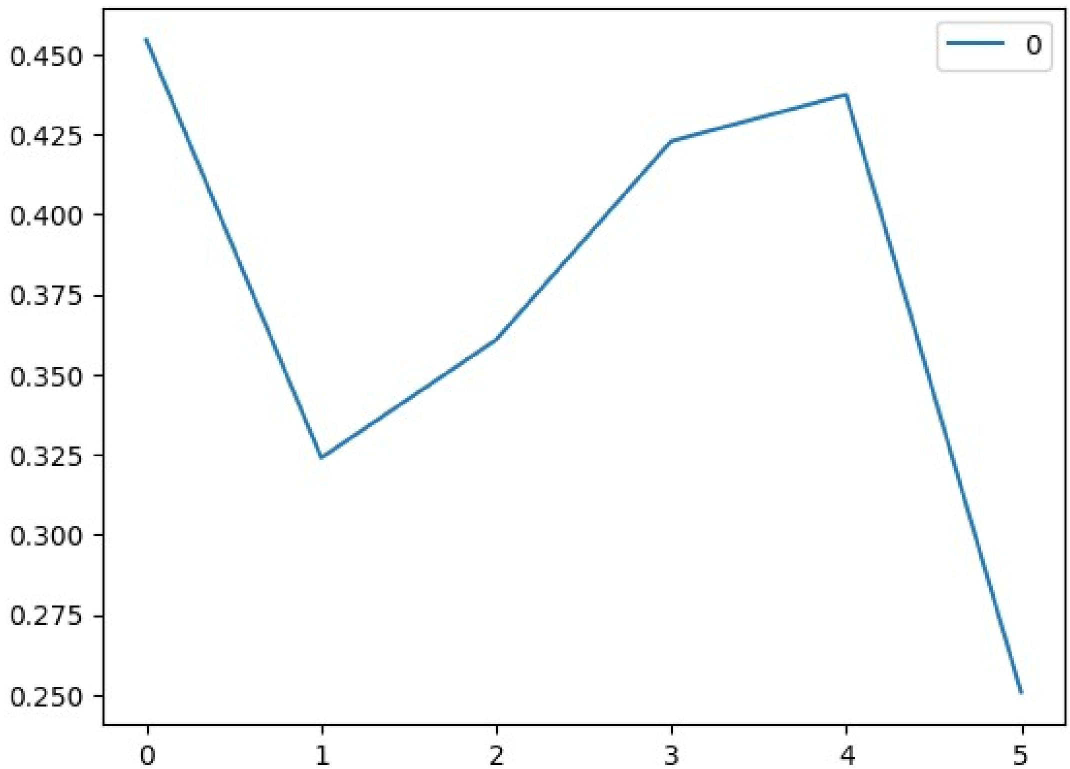
```
preds.correct_predictions))


# Print the the resulting dataframe
print(rmse_bkt, " mean: ", np.mean(rmse_bkt), " std: ",
np.std(rmse_bkt))
print(auc_bkt, " mean: ", np.mean(auc_bkt), " std: ", np.std(auc_bkt))
### Share your metric visualization plot with us
plt.barh(skills_subset, rmse_bkt)
#send(plt, 1)
plt.show()

plt.barh(skills_subset, auc_bkt)
plt.show()
### Share your analysis of the metric
metric_discussion = ""
#send(metric_discussion, 2)
```

```
-- Circle Graph --
CPU times: user 474 ms, sys: 0 ns, total: 474 ms
Wall time: 181 ms
-- Venn Diagram --
CPU times: user 1.35 s, sys: 9.04 ms, total: 1.35 s
Wall time: 684 ms
-- Mode --
CPU times: user 331 ms, sys: 330 µs, total: 331 ms
Wall time: 104 ms
-- Division Fractions --
CPU times: user 311 ms, sys: 0 ns, total: 311 ms
Wall time: 185 ms
-- Finding Percents --
CPU times: user 374 ms, sys: 688 µs, total: 375 ms
Wall time: 182 ms
-- Area Rectangle --
CPU times: user 492 ms, sys: 4.14 ms, total: 496 ms
Wall time: 281 ms
[0.45449455569075925, 0.3239651928883658, 0.3608313946406462,
0.42288019162563645, 0.43737425456401574, 0.25082638310355526]  mean:
0.37506199541882973  std:  0.07156210560799578
[0.7350582833877455, 0.9113705191861753, 0.7902208201892744,
0.7630769230769231, 0.7851426101029809, 0.7506945950032886]  mean:
0.7892606251577314  std:  0.05779204823042674
```






```
ax = sns.lineplot(data=pd.DataFrame(rmse_bkt), errorbar='sd')
plt.show()
```



0
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0
1
2
3
4
5

Lecture 7 - Student Notebook

In this exercises, you will create and interpret learning curves and compare the performance of different knowledge tracing models. We will use the same ASSISTments data set as for lecture 6.

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.

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

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 to obtain train, test, and validation folds of our data.

```
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)
```

BKT Models - Learning Curves

Last week, we have seen how to use BKT to predict the probability that a student will solve a task correctly. In addition, we can also use this type of model to compute learning curves and in this way analyze the learning activity (in our case the skills).

We first 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, but just checking interpretation). To keep things simpler on the following 6 skills for this exercise:
`'Circle Graph', 'Venn Diagram', 'Mode', 'Division Fractions', 'Finding Percents', 'Area Rectangle'`

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

```
# Initialize the model
model = Model(seed=0)

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

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']
```

```
CPU times: user 6.49 s, sys: 1.02 ms, total: 6.49 s
Wall time: 3.32 s
```

```
predictions.head()
```

	user_id	skill_name	y_true	y_pred_bkt
3957	14	Circle Graph	0	0.45897
3958	14	Circle Graph	1	0.33319
3959	14	Circle Graph	0	0.56200
3960	14	Circle Graph	0	0.43364
3961	14	Circle Graph	0	0.31410

Next, we create a function that computes the learning curve (observed or predicted) for us by averaging over the success rate of all users at a given opportunity.

```
def avg_y_by_x(x, y):
    '''
    Compute average learning curve and number of students over the
number of opportunities.
    x is the number of opportunities.
    y the success rates of the users (can be predicted success rate or
true success rate).
    '''
    # 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
```

Then, we create a function for plotting learning curve and a bar chart with the number of students per opportunity for a given skill.

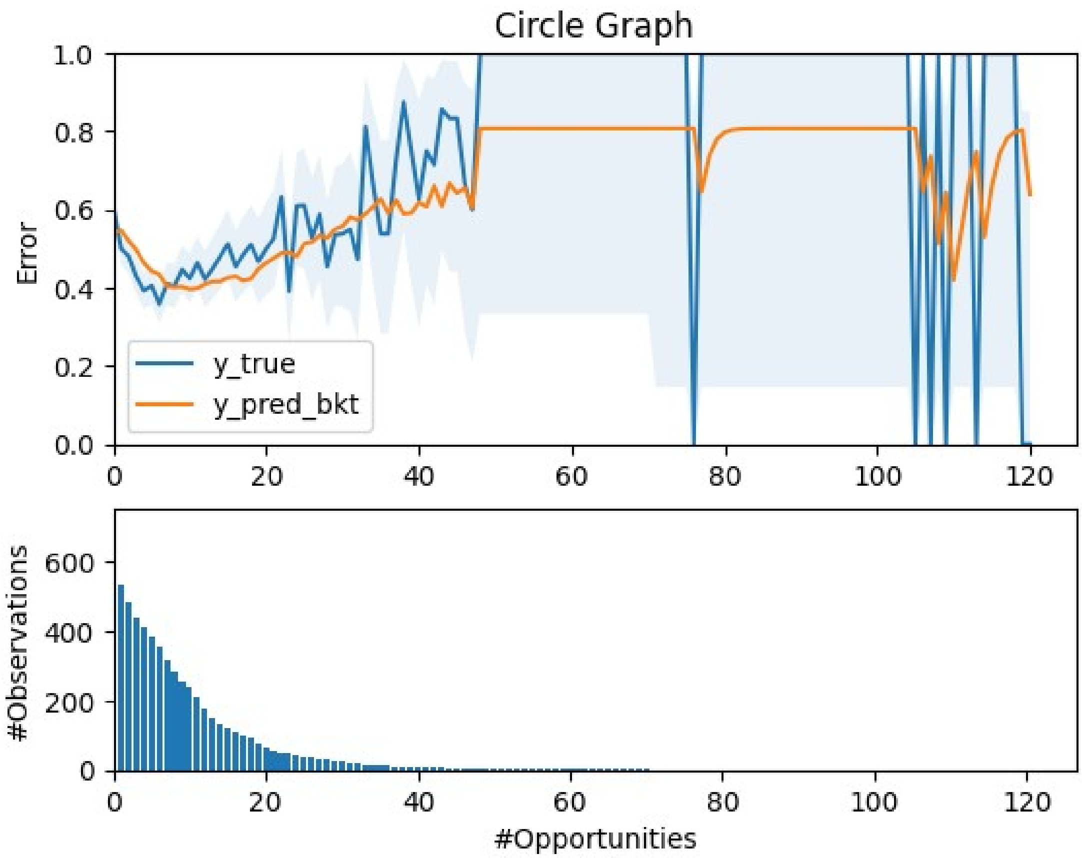
```
def plot_learning_curve(skill_name):
    '''
    Plot learning curve using BKT model for skill `skill_name`.
    '''
    preds = predictions[predictions['skill_name'] == skill_name] #
Retrieve predictions for the current skill
```

```
    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
    return plt
```

We then plot the learning curve and number of opportunities per student for skill `Circle Graph`.

```
plt = plot_learning_curve('Circle Graph')
plt.show()
```

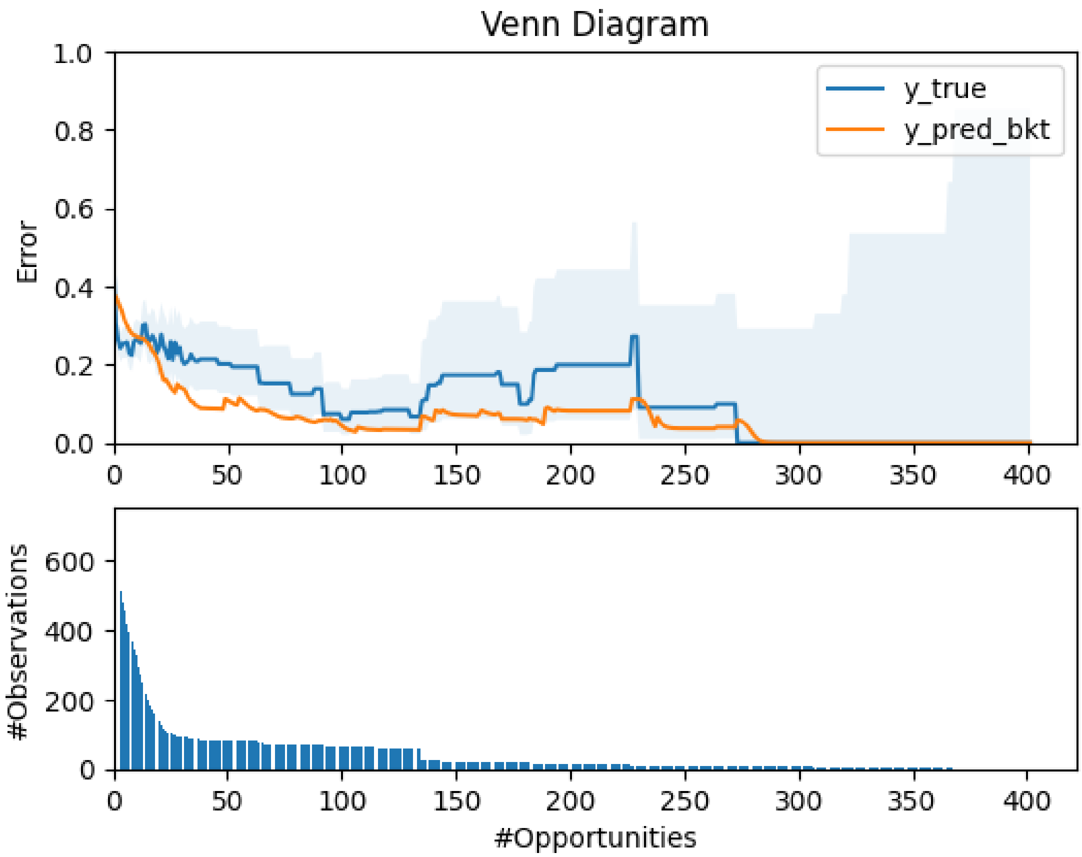


Your Turn 1 - Learning Curves

Visualize and interpret the learning curves and number of students per opportunity for two selected skills. You can choose from the remaining five skills: `'Venn Diagram'`, `'Mode'`, `'Division Fractions'`, `'Finding Percents'`, `'Area Rectangle'`. Send us your visualizations as well as the discussion.

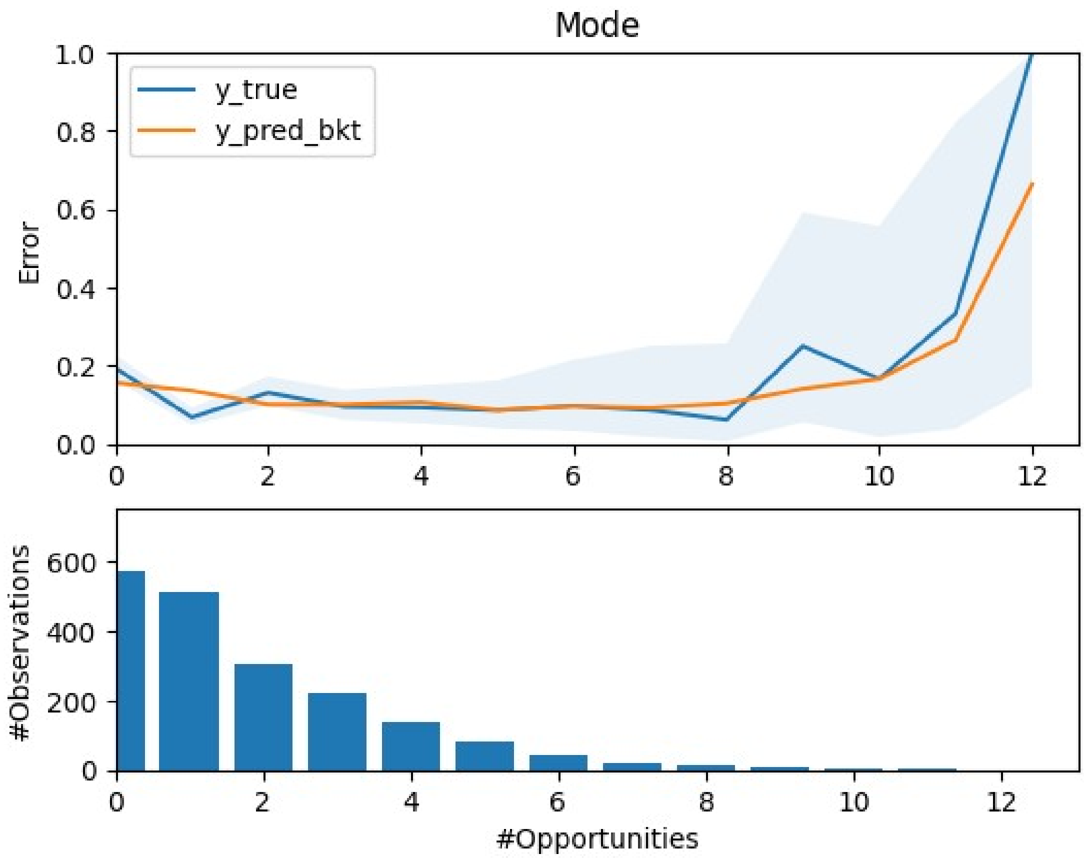
```
# YOUR TURN: Visualize the learning curve for the first skill.

first_skill_name = "Venn Diagram" # replace the skill name with one of the 5 skills above
plt = plot_learning_curve(first_skill_name)

### Share the plot with us
```

```
#send(plt, 1)
plt.show()
```




```
# YOUR TURN: What is your analysis about the learning curve for the
first skill?

### Share your analysis of the learning curve with us
first_skill_interpretation = "Write your interpretation here"
send(first_skill_interpretation, 4)

# YOUR TURN: Visualize the learning curve for the second skill.

second_skill_name = "Mode" # replace the skill name with one of the 5
skills above
plt = plot_learning_curve(second_skill_name)

### Share the plot with us
send(plt, 2)
plt.show()
```

```
Variable npt_config is not defined
```




```
# YOUR TURN: What is your analysis about the learning curve for the second skill?

### Share your analysis of the learning curve with us
second_skill_interpretation = "Write your interpretation here"
send(second_skill_interpretation, 6)
```

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.

```
# 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()
```

```
      user_id  order_id    skill_name  correct  correct_predictions  \
3957       14  21617623  Circle Graph        0              0.45897
3958       14  21617632  Circle Graph        1              0.33319
3959       14  21617641  Circle Graph        0              0.56200
3960       14  21617650  Circle Graph        0              0.43364
3961       14  21617659  Circle Graph        0              0.31410

      state_predictions  aux  prev_attempts  correct_aux
before_correct_num  \
3957            0.55462    1              0            0
0
3958            0.33498    1              1            1
0
3959            0.73454    1              2            1
1
3960            0.51039    1              3            1
1
3961            0.30164    1              4            1
1

      before_wrong_num
3957                 0
3958                 1
3959                 1
3960                 2
3961                 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 29.7 s, sys: 261 ms, total: 29.9 s
Wall time: 30 s

        user_id  order_id          skill_name  correct
correct_predictions  \
53382     53167  26451283                Mode        1
0.84330
53383     53167  26451302                Mode        1
0.92807
157253    53167  32517498  Division Fractions        0
0.59832
157254    53167  32517627  Division Fractions        1
0.43335
157255    53167  32517648  Division Fractions        1
```

```
0.73088

        state_predictions  aux  prev_attempts  correct_aux  \
53382             0.89625    1              0            1
53383             0.99413    1              1            2
157253            0.68806    1              0            0
157254            0.45102    1              1            1
157255            0.87852    1              2            2

        before_correct_num  before_wrong_num  afm_predictions
53382                    0                 0          0.85507
53383                    1                 0          0.86928
157253                   0                 0          0.62272
157254                   0                 1          0.64312
157255                   1                 1          0.66302
```

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 12s, sys: 375 ms, total: 1min 12s
Wall time: 1min 27s

        user_id  order_id          skill_name  correct
correct_predictions  \
53382     53167  26451283                Mode        1
0.84330
53383     53167  26451302                Mode        1
0.92807
157253    53167  32517498  Division Fractions        0
0.59832
157254    53167  32517627  Division Fractions        1
0.43335
157255    53167  32517648  Division Fractions        1
0.73088

        state_predictions  aux  prev_attempts  correct_aux  \
53382             0.89625    1              0            1
53383             0.99413    1              1            2
157253            0.68806    1              0            0
157254            0.45102    1              1            1
157255            0.87852    1              2            2

        before_correct_num  before_wrong_num  afm_predictions
pfa_predictions
53382                    0                 0          0.85507
0.83721
53383                    1                 0          0.86928
0.87290
157253                   0                 0          0.62272
0.61971
157254                   0                 1          0.64312
0.62047
157255                   1                 1          0.66302
0.65994
```

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 2.97 s, sys: 13.2 ms, total: 2.98 s
Wall time: 1.45 s
--Venn Diagram--

/tmp/ipykernel_520/1614788260.py:13: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

CPU times: user 1.3 s, sys: 857 µs, total: 1.3 s
Wall time: 603 ms
--Mode--

/tmp/ipykernel_520/1614788260.py:13: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

CPU times: user 620 ms, sys: 0 ns, total: 620 ms
Wall time: 290 ms
--Division Fractions--

/tmp/ipykernel_520/1614788260.py:13: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

CPU times: user 763 ms, sys: 6.22 ms, total: 770 ms
Wall time: 382 ms
--Finding Percents--

/tmp/ipykernel_520/1614788260.py:13: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)
```

```
CPU times: user 582 ms, sys: 0 ns, total: 582 ms
Wall time: 289 ms
--Area Rectangle--

/tmp/ipykernel_520/1614788260.py:13: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

CPU times: user 492 ms, sys: 0 ns, total: 492 ms
Wall time: 205 ms

/tmp/ipykernel_520/1614788260.py:13: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

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

      before_correct_num  before_wrong_num  afm_predictions
pfa_predictions  \
3969                   0                 0          0.48266
0.46224
3970                   1                 0          0.49251
0.48999
3971                   2                 0          0.50236
0.51780
3972                   3                 0          0.51221
0.54551
3973                   3                 1          0.52205
0.54598

      bkt_predictions
3969          0.45193
3970          0.63194
3971          0.68854
3972          0.70022
3973          0.69561
```

```
X_test.to_csv('x_test_07.csv.gz', compression = 'gzip', index = False)
```

Your Turn 2 - 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) such that the metrics can be easily compared.
2. Interpret your results and discuss your observations.

```
# 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('x_test_07.csv.gz', compression = 'gzip')

X_test["skill_name"].unique()

array(['Circle Graph', 'Venn Diagram', 'Mode', 'Division Fractions',
       'Finding Percents', 'Area Rectangle'], dtype=object)

Circle_Graph = X_test[X_test["skill_name"]=="Circle Graph"]
Venn_Diagram = X_test[X_test["skill_name"]=="Venn Diagram"]
Mode = X_test[X_test["skill_name"]=="Mode"]
Division_Fractions = X_test[X_test["skill_name"]=="Division
Fractions"]
Finding_Percents = X_test[X_test["skill_name"]=="Finding Percents"]
Area_Rectangle = X_test[X_test["skill_name"]=="Area Rectangle"]

from sklearn.metrics import mean_squared_error, roc_auc_score
RMSE, AUC = [], []
predictions_list = ["afm_predictions", "pfa_predictions",
"bkt_predictions"]

for predictions in predictions_list:
    RMSE.append(mean_squared_error(X_test.correct,
X_test[predictions], squared = False))
    AUC.append(roc_auc_score(X_test.correct, X_test[predictions]))

# Visualize plots
#send(plt, 3)
plt.show()

interpretation = "Write your interpretation here"

send(interpretation, 4)
```

Lecture 8 - Student Version

Recurrent neural networks can handle time series data of different lengths. In this demo notebook we will first look deeper into Deep Knowledge Tracing, before showing examples of different types of neural network models for tracing and time series tasks. The learning objectives of this notebook are as follows:

1. Explore the differences between deep learning architectures for time-series data with LSTMs, GRUs and RNNs.

2. Implement hyperparameter tuning for a deep learning pipeline.

3. Contrast two behavioral time-series data settings: a model that makes a prediction at every time interval vs. a model that makes an overall prediction at the end of the time series.

If you are using EPFL's Noto, this notebook will need to use the `tensorflow` kernel for the dependencies to be installed appropriately. Change the kernel in the upper right corner of Noto. Select `tensorflow`.

```
# 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 tensorflow as tf

# In this demo, we use a lot of SciKit-Learn functions, as imported
below.
from sklearn import feature_extraction, model_selection
from sklearn.metrics import mean_squared_error, roc_auc_score,
balanced_accuracy_score
from sklearn.model_selection import ParameterGrid, train_test_split
from sklearn.preprocessing import MinMaxScaler

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

# Setting this variable to true will train the DKT model fitting,
evaluation and
# hyperparameter tuning from scratch, which will take ~1 hour on
Colab.
train_from_scratch = False

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)
```

Deep Knowledge Tracing (DKT)

We begin by loading the data of the ASSISTments dataset (that we have explored in previous lectures).

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.

```
data = pd.read_csv(DATA_DIR + 'assistments.csv',
low_memory=False).dropna()
data.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 students and skills in the dataset.

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

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

Data Preparation

Since the data needs to be fed into the model in batches, we need to specify in advance how many elements per batch the DKT model will receive. DKT also requires that all sequences need to be of the same length in order to be used as model input.

Given that students have different number of opportunities across skills, we need to define a scheme such that the sequences will be the same length. We choose to pad our values to the maximum sequence length and determine a masking value (for the model to ignore) for those entries that are introduced as a padding into the student's sequences.

```
def prepare_seq(df):
    '''
    Extract user_id sequence in preparation for DKT. The output of
this function
    feeds into the prepare_data() function.
    '''
    # Enumerate skill id as a categorical variable
    # (i.e. [32, 12, 32, 45] -> [0, 1, 0, 2])
    df['skill'], skill_codes = pd.factorize(df['skill_name'],
sort=True)

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

    # 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:],))

    # 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):
    '''
    Manipulate the data sequences into the right format for DKT with
padding by batch
    and encoding categorical features.
    '''

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

    # 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)
        )
    )

    # Pad sequences to the appropriate length 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)
```

Your Turn (In-Class Discussion)

What do these hyperparameters mean?

```
# Specify the model hyperparameters. Full descriptions included in the
demo notebook!
params = {}

params['batch_size'] = 32
params['mask_value'] = -1.0
params['verbose'] = 1
params['best_model_weights'] = 'weights/bestmodel'
params['optimizer'] = 'adam'
params['recurrent_units'] = 16
params['epochs'] = 20
params['dropout_rate'] = 0.1
```

We then split the data into a train, a validation and a test set.

```
# Obtain indexes for training and test sets
train_index, test_index = next(create_iterator(data))

# Split the data into training and test
X_train, X_test = data.iloc[train_index], data.iloc[test_index]

# Obtain indexes for training and validation sets
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]

# Build TensorFlow sequence datasets for training, validation, and
test data
```

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

# Prepare the training, validation, and test data in the DKT input
format
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)

# Calculate the length of each of the train-test-val sets and store as
parameters
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'])
```

```
2023-06-29 18:59:51.688341: 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
2023-06-29 18:59:51.688412: W
tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to
cuInit: UNKNOWN ERROR (303)
2023-06-29 18:59:51.688457: 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
2023-06-29 18:59:51.689086: 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

First, we train DKT using an LSTM architecture and default parameter settings. We use a validation set to monitor prediction accuracy of the model and store the model with the best weights.

Considering that we padded the sequences such that all have the same length, we need to remove predictions for the time steps that are based on padded data. To this end, we implement a function called get_target.

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

```
    Adjust y_true and y_pred to ignore predictions made using padded
values.
    '''
    # 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 and evaluating the model, we will monitor the following performance metrics. Please, note that we need to process our targets before using the default TensorFlow metric functions.

```
class AUC(tf.keras.metrics.AUC):
    # Our custom AUC calls our get_target function first to remove
predictions on padded values,
    # then computes a standard AUC metric.
    def __init__(self):
        # We use a super constructor here just to make our metric name
pretty!
        super(AUC, self).__init__(name='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):
    # Our custom RMSE calls our get_target function first to remove
predictions on padded values,
    # then computes a standard RMSE metric.
    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):
    # Our custom binary cross entropy loss calls our get_target
function first
    # to remove predictions on padded values, then computes standard
binary cross-entropy.
    y_true, y_pred = get_target(y_true, y_pred)
    return tf.keras.losses.binary_crossentropy(y_true, y_pred)
```

We define an LSTM and a GRU model.

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

    # Create an LSTM model architecture
    inputs = tf.keras.Input(shape=(None, nb_features), name='inputs')

    # We use a masking layer here to ignore our masked padding values
    x = tf.keras.layers.Masking(mask_value=params['mask_value'])
(inputs)

    # This LSTM layer is the crux of the model; we use our parameters
to specify
    # what this layer should look like (# of recurrent_units, fraction
of dropout).
    x = tf.keras.layers.LSTM(params['recurrent_units'],
return_sequences=True, dropout=params['dropout_rate'])(x)

    # We use a dense layer with the sigmoid function activation to map
our predictions
    # between 0 and 1.
    dense = tf.keras.layers.Dense(nb_skills, activation='sigmoid')

    # The TimeDistributed layer takes the dense layer predictions and
applies the sigmoid
    # activation function to all time steps.
    outputs = tf.keras.layers.TimeDistributed(dense, name='outputs')
(x)
    model = tf.keras.models.Model(inputs=inputs, outputs=outputs,
name='DKT')

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

    return model

# Create our DKT model using an LSTM
dkt_lstm = create_model_lstm(features_depth, skill_depth, params)

def create_model_gru(nb_features, nb_skills, params):

    # Create a GRU model architecture
    inputs = tf.keras.Input(shape=(None, nb_features), name='inputs')

    # We use a masking layer here to ignore our masked padding values
    x = tf.keras.layers.Masking(mask_value=params['mask_value'])
(inputs)
```

```
    # This GRU layer is the crux of the model; we use our parameters
to specify
    # what this layer should look like (# of recurrent_units, fraction
of dropout).
    x = tf.keras.layers.GRU(params['recurrent_units'],
return_sequences=True, dropout=params['dropout_rate'])(x)

    # We use a dense layer with the sigmoid function activation to map
our predictions
    # between 0 and 1.
    dense = tf.keras.layers.Dense(nb_skills, activation='sigmoid')

    # The TimeDistributed layer takes the dense layer predictions and
applies the sigmoid
    # activation function to all time steps.
    outputs = tf.keras.layers.TimeDistributed(dense, name='outputs')
(x)
    model = tf.keras.models.Model(inputs=inputs, outputs=outputs,
name='DKT')

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

    return model

# Create our DKT model using a GRU
dkt_gru = create_model_gru(features_depth, skill_depth, params)
```

Model Fitting and Evaluation

Next we train the models and then evaluate them on the test data.

```
# This cell takes 8 minutes to run. On default, we will not run the
training experiments below.
# However, if you would like to run it from scratch, you can modify
train_from_scratch=True
# at the beginning of the notebook.

if train_from_scratch:
  # This line tells our training procedure to only save the best
version of the model at any given time.
  ckp_callback =
tf.keras.callbacks.ModelCheckpoint(params['best_model_weights'],

save_best_only=True, save_weights_only=True)
```

```
  # Let's fit our LSTM model on the training data. This cell takes 8 
minutes to run.
  history = dkt_lstm.fit(tf_train, epochs=params['epochs'], 
steps_per_epoch=params['train_size']-1,
                        validation_data=tf_val, 
validation_steps=params['val_size'],
                        callbacks=[ckp_callback], 
verbose=params['verbose'])

if train_from_scratch:
  # We load the LSTM model with the best performance, and evaluate it 
on the test set.
  dkt_lstm.load_weights(params['best_model_weights'])
  dkt_lstm.evaluate(tf_test, steps=params['test_size'], 
verbose=params['verbose'], return_dict=True)

if train_from_scratch:
  # This line tells our training procedure to only save the best 
version of the model at any given time.
  ckp_callback = 
tf.keras.callbacks.ModelCheckpoint(params['best_model_weights'],

save_best_only=True, save_weights_only=True)

  # Let's fit our GRU model on the training data. This cell takes 8 
minutes to run.
  history = dkt_gru.fit(tf_train, epochs=params['epochs'], 
steps_per_epoch=params['train_size']-1,
                        validation_data=tf_val, 
validation_steps=params['val_size'],
                        callbacks=[ckp_callback], 
verbose=params['verbose'])

if train_from_scratch:
  # We load the GRU model with the best performance, and evaluate it 
on the test set.
  dkt_gru.load_weights(params['best_model_weights'])
  dkt_gru.evaluate(tf_test, steps=params['test_size'], 
verbose=params['verbose'], return_dict=True)
```

Hyperparameter Tuning

As we have seen, we need to specify a lot of hyperparameters. In a next step, we perform a small grid search for the number of recurrent units in the LSTM: {8, 16, 32, 64}.

```
# Modify the dictionary of parameters so that each parameter maps to a 
list of possibilities.
# In this case, we're only searching over the recurrent_units and 
leaving the rest of the
# parameters fixed to their default values.
params_space = {param: [value] for param, value in params.items()}
```

```
params_space['recurrent_units'] = [8, 16, 32, 64]
params_grid = ParameterGrid(params_space)

# For each combination of candidate parameters, fit a model on the
training set
# and evaluate it on the validation set (as we've seen in Lecture 5).

# NOTE: This cell will take 40 minutes to run from scratch.
if train_from_scratch:
  results = {}

  # For each parameter setting in the grid search of parameters
  for params_i in params_grid:

      # Create a LSTM model with the specific parameter setting
params_i
      dkt_lstm = create_model_lstm(features_depth, skill_depth,
params_i)

      save_model_name = params_i['best_model_weights'] +
str(params_i['recurrent_units'])

      # Save the best version of the model through the training epochs
      ckp_callback =
tf.keras.callbacks.ModelCheckpoint(save_model_name,

save_best_only=True, save_weights_only=True)

      # Fit the model on the training data with the appropriate
parameters
      dkt_lstm.fit(tf_train,
                   epochs=params_i['epochs'],
                   steps_per_epoch=params_i['train_size']-1,
                   validation_data=tf_val,
                   validation_steps=params_i['val_size'],
                   callbacks=[ckp_callback],
                   verbose=params_i['verbose'])

      # Evaluate the model performance
      results[params_i['recurrent_units']] = dkt_lstm.evaluate(tf_val,

steps=params_i['val_size'],

verbose=params_i['verbose'],

return_dict=True)

if train_from_scratch:
  # Sort candidate parameters according to their accuracy
```

```
  results = sorted(results.items(), key=lambda x: x[1]['auc'],
reverse=True)

  # Obtain the best parameters
  best_params = results[0][0]
  best_params

if train_from_scratch:
  # Load the best model variant from the hyperparameter gridsearch
  dkt_lstm.load_weights(params['best_model_weights'] +
str(best_params))
  dkt_lstm.evaluate(tf_test, steps=params['test_size'],
                      verbose=params['verbose'],
                      return_dict=True)
```

Tracing and Time-Series Experiments

Next, we perform experiments with recurrent neural networks for tracing as well as the time series task. We first load the data for the tracing task. It stems from a massive open online course (MOOC) hosted by EPFL. We first load the features as well as the labels to predict.

```
mooc_feat = pd.read_csv(DATA_DIR + 'mooc_feat.csv', low_memory=False)
mooc_feat.columns
```

```
Index(['user_id', 'week', 'TotalClicksVideoLoad',
'AvgWatchedWeeklyProp',
       'StdWatchedWeeklyProp', 'AvgReplayedWeeklyProp',
       'StdReplayedWeeklyProp', 'AvgInterruptedWeeklyProp',
       'StdInterruptedWeeklyProp', 'TotalClicksVideoConati',
       'FrequencyEventVideo', 'FrequencyEventLoad',
'FrequencyEventVideoPlay',
       'FrequencyEventVideoPause', 'FrequencyEventVideoStop',
       'FrequencyEventVideoSeekBackward',
'FrequencyEventVideoSeekForward',
       'FrequencyEventVideoSpeedChange', 'AvgSeekLength',
'StdSeekLength',
       'AvgPauseDuration', 'StdPauseDuration', 'AvgTimeSpeedingUp',
       'StdTimeSpeedingUp', 'RegPeakTimeDayHour', 'RegPeriodicityM1',
       'DelayLecture', 'TotalClicks', 'NumberOfSessions',
'TotalTimeSessions',
       'AvgTimeSessions', 'StdTimeBetweenSessions', 'StdTimeSessions',
       'TotalClicksWeekday', 'TotalClicksWeekend',
'RatioClicksWeekendDay',
       'TotalClicksVideoChen', 'TotalClicksProblem',
'TotalTimeProblem',
       'TotalTimeVideo', 'CompetencyAlignment',
'CompetencyAnticipation',
       'ContentAlignment', 'ContentAnticipation'],
      dtype='object')
```

```
mooc_quizzes = pd.read_csv(DATA_DIR + 'mooc_quizzes.csv',
low_memory=False)
display(mooc_quizzes)
```

```
       user_id  week  quiz_correct
0         1593     0      0.929825
1         1593     1           NaN
2         1593     2      0.807141
3         1593     3      0.960000
4         1593     4      0.900000
...        ...   ...           ...
59685  3353959     5           NaN
59686  3353959     6           NaN
59687  3353959     7           NaN
59688  3353959     8           NaN
59689  3353959     9           NaN

[59690 rows x 3 columns]
```

Tracing: Data Preparation

```
# Normalize all the features with min-max scaling
scaler = MinMaxScaler()
mooc_feat.iloc[:, 2:] = scaler.fit_transform(mooc_feat.iloc[:, 2:])

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

```
Number of unique students in the dataset: 4352
```

In this analysis, we want to predict **weekly quiz performance** of the students. We perform the following preprocessing steps to prepare our data:

- First, we observe from the data frame mooc_quizzes that quite a number of students have not solved quizzes in all weeks. We will use a mask to ignore weeks for students with missing quiz answers. We create a new data frame df_y (the outcome), where we replace NaNs (for quiz_correct) with -1. We also create a data frame df_x, where we replace the according input feature values with -1.

- Second, we bring df_y and df_x to an appropriate shape.

  df_y should become a NumPy array of size:

  `size(df_y) = num_of_students * num_of_weeks`

  df_x should become a NumPy array of size:

  `size(df_x) = num_of_students * num_of_weeks * num_of_features`.

We create a data frame `df_x`, where we ignore weeks for students with missing quiz answers by filling in the appropriate feature values with -1.

```
num_features = 42
num_index = mooc_feat.shape[1] - num_features

# Mask df_x values
mask = mooc_quizzes.quiz_correct.isna().values
mask = np.concatenate([np.zeros((mask.shape[0], num_index),
dtype=bool),
                       mask[:, None].repeat(num_features, axis=1)],
axis=1)
df_x = mooc_feat.mask(mask, -1)
df_x
```

```
----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Input In [38], in <cell line: 8>()
      5 mask = mooc_quizzes.quiz_correct.isna().values
      6 mask = np.concatenate([np.zeros((mask.shape[0], num_index),
dtype=bool),
      7                        mask[:, None].repeat(num_features,
axis=1)], axis=1)
----> 8 df_x = mooc_feat.mask(mask, -1)
      9 df_x

File
/usr/local/lib/python3.8/dist-packages/pandas/util/_decorators.py:311,
in
deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*arg
s, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File
/usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:10976, in
DataFrame.mask(self, cond, other, inplace, axis, level, errors,
try_cast)
  10963 @deprecate_nonkeyword_arguments(
  10964     version=None, allowed_args=["self", "cond", "other"]
  10965 )
   (...)
  10974     try_cast=lib.no_default,
  10975 ):
> 10976     return super().mask(cond, other, inplace, axis, level,
errors, try_cast)
```

```
File
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:9346, in
NDFrame.mask(self, cond, other, inplace, axis, level, errors,
try_cast)
   9343 if not hasattr(cond, "__invert__"):
   9344     cond = np.array(cond)
-> 9346 return self.where(
   9347     ~cond,
   9348     other=other,
   9349     inplace=inplace,
   9350     axis=axis,
   9351     level=level,
   9352     errors=errors,
   9353 )

File
/usr/local/lib/python3.8/dist-packages/pandas/util/_decorators.py:311,
in
deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*arg
s, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File
/usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:10961, in
DataFrame.where(self, cond, other, inplace, axis, level, errors,
try_cast)
  10948 @deprecate_nonkeyword_arguments(
  10949     version=None, allowed_args=["self", "cond", "other"]
  10950 )
   (...)
  10959     try_cast=lib.no_default,
  10960 ):
> 10961     return super().where(cond, other, inplace, axis, level,
errors, try_cast)

File
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:9310, in
NDFrame.where(self, cond, other, inplace, axis, level, errors,
try_cast)
   9302 if try_cast is not lib.no_default:
   9303     warnings.warn(
   9304         "try_cast keyword is deprecated and will be removed in
a "
```

```
   9305         "future version.",
   9306         FutureWarning,
   9307         stacklevel=find_stack_level(),
   9308     )
-> 9310 return self._where(cond, other, inplace, axis, level,
errors=errors)

File
/usr/local/lib/python3.8/dist-packages/pandas/core/generic.py:9054, in
NDFrame._where(self, cond, other, inplace, axis, level, errors)
   9052         cond = np.asanyarray(cond)
   9053     if cond.shape != self.shape:
-> 9054         raise ValueError("Array conditional must be same shape
as self")
   9055     cond = self._constructor(cond,
**self._construct_axes_dict())
   9057 # make sure we are boolean

ValueError: Array conditional must be same shape as self
```

We create `df_y` (the outcome), where we replace NaNs (for quiz_correct) with -1.

```
df_y = mooc_quizzes.fillna(-1)
df_y
```

We bring `df_y` and `df_x` to an appropriate shape.

```
num_weeks = df_y.week.nunique()
df_y = df_y.quiz_correct.values.reshape(-1, num_weeks, 1)
df_x = df_x.iloc[:, num_index:].values.reshape(-1, num_weeks,
num_features)
```

We then split the data into train, test, and validation data set.

```
# Split the MOOC data into training and test sets.
df_x_train, df_x_test, df_y_train, df_y_test = train_test_split(
                                                        df_x, df_y,
test_size=0.2,

random_state=0)

# Split the training dataset into validation and training sets.
df_x_train_val, df_x_val, df_y_train_val, df_y_val = train_test_split(
                                                        df_x_train,
df_y_train,
                                                        test_size=0.2,
random_state=0)
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
```

```
last)
Input In [35], in <cell line: 2>()
      1 # Split the MOOC data into training and test sets.
      2 df_x_train, df_x_test, df_y_train, df_y_test =
train_test_split(
----> 3                                                         df_x,
df_y, test_size=0.2,
      4
random_state=0)
      6 # Split the training dataset into validation and training
sets.
      7 df_x_train_val, df_x_val, df_y_train_val, df_y_val =
train_test_split(
      8
df_x_train, df_y_train,
      9
test_size=0.2, random_state=0)

NameError: name 'df_x' is not defined
```

Tracing: Model Creation

Next, we build an LSTM model for predicting student performance on the MOOC.

```
# We use the default hyperparameters, as described in detail in the
DKT model creation section.
params = {}
params['batch_size'] = 32
params['mask_value'] = -1.0
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['recurrent_units'] = 32 # Number of RNN units
params['epochs'] = 20 # Number of epochs to train
params['dropout_rate'] = 0.1 # Dropout rate

def create_model_lstm_MOOC(nb_features, nb_skills, params):

    # Create an LSTM model architecture.
    inputs = tf.keras.Input(shape=(None, nb_features), name='inputs')

    # We use a masking layer here to ignore our masked padding values
    x = tf.keras.layers.Masking(mask_value=params['mask_value'])
(inputs)

    # This LSTM layer is the crux of the model; we use our parameters
to specify
    # what this layer should look like (# of recurrent_units, fraction
of dropout).
    x = tf.keras.layers.LSTM(params['recurrent_units'],
```

```
                                  return_sequences=True,
                                  dropout=params['dropout_rate'])(x)


    # We use a dense layer with the linear function activation to map
our predictions
    # on a linear scale. Note that this has changed from a sigmoid
activated dense layer
    # in the previous LSTM function.
    dense = tf.keras.layers.Dense(nb_skills, activation='linear')
    outputs = tf.keras.layers.TimeDistributed(dense, name='outputs')
(x)
    model = tf.keras.models.Model(inputs=inputs, outputs=outputs,
name='DKT')

    # Compile the model with our loss functions, optimizer, and
metrics.
    model.compile(loss=tf.keras.losses.MSE,
                  optimizer=params['optimizer'],
                  metrics=[tf.keras.metrics.RootMeanSquaredError()])

    return model

dkt_lstm = create_model_lstm_MOOC(num_features, 1, params)
```

Tracing: Model Fitting and Evaluation

```
# This model takes less than 5 minutes to train on Noto (< 1 minute on
Colab).

# We save only the best model during the training process.
ckp_callback =
tf.keras.callbacks.ModelCheckpoint(params['best_model_weights'],
                                                  save_best_only=True,
save_weights_only=True)

# Fit the DKT LSTM on DSP1 data.
history = dkt_lstm.fit(df_x_train_val, df_y_train_val,
epochs=params['epochs'],
                       validation_data=(df_x_val, df_y_val),
                       callbacks=[ckp_callback],
verbose=params['verbose'])

Epoch 1/20
120/120 [==============================] - 7s 22ms/step - loss: 0.0360
- root_mean_squared_error: 0.3023 - val_loss: 0.0201 -
val_root_mean_squared_error: 0.2244
Epoch 2/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0197
- root_mean_squared_error: 0.2238 - val_loss: 0.0179 -
val_root_mean_squared_error: 0.2114
```

```
Epoch 3/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0184
- root_mean_squared_error: 0.2161 - val_loss: 0.0175 -
val_root_mean_squared_error: 0.2092
Epoch 4/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0180
- root_mean_squared_error: 0.2137 - val_loss: 0.0176 -
val_root_mean_squared_error: 0.2099
Epoch 5/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0180
- root_mean_squared_error: 0.2141 - val_loss: 0.0171 -
val_root_mean_squared_error: 0.2070
Epoch 6/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0176
- root_mean_squared_error: 0.2112 - val_loss: 0.0170 -
val_root_mean_squared_error: 0.2059
Epoch 7/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0177
- root_mean_squared_error: 0.2118 - val_loss: 0.0169 -
val_root_mean_squared_error: 0.2056
Epoch 8/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0176
- root_mean_squared_error: 0.2113 - val_loss: 0.0175 -
val_root_mean_squared_error: 0.2089
Epoch 9/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0174
- root_mean_squared_error: 0.2105 - val_loss: 0.0168 -
val_root_mean_squared_error: 0.2051
Epoch 10/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0173
- root_mean_squared_error: 0.2098 - val_loss: 0.0168 -
val_root_mean_squared_error: 0.2048
Epoch 11/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0173
- root_mean_squared_error: 0.2098 - val_loss: 0.0167 -
val_root_mean_squared_error: 0.2046
Epoch 12/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0174
- root_mean_squared_error: 0.2102 - val_loss: 0.0167 -
val_root_mean_squared_error: 0.2041
Epoch 13/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0172
- root_mean_squared_error: 0.2089 - val_loss: 0.0166 -
val_root_mean_squared_error: 0.2038
Epoch 14/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0174
- root_mean_squared_error: 0.2099 - val_loss: 0.0179 -
val_root_mean_squared_error: 0.2118
Epoch 15/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0173
```

```
- root_mean_squared_error: 0.2098 - val_loss: 0.0166 -
val_root_mean_squared_error: 0.2039
Epoch 16/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0171
- root_mean_squared_error: 0.2083 - val_loss: 0.0166 -
val_root_mean_squared_error: 0.2040
Epoch 17/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0172
- root_mean_squared_error: 0.2092 - val_loss: 0.0168 -
val_root_mean_squared_error: 0.2052
Epoch 18/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0171
- root_mean_squared_error: 0.2085 - val_loss: 0.0164 -
val_root_mean_squared_error: 0.2027
Epoch 19/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0170
- root_mean_squared_error: 0.2080 - val_loss: 0.0166 -
val_root_mean_squared_error: 0.2036
Epoch 20/20
120/120 [==============================] - 1s 10ms/step - loss: 0.0170
- root_mean_squared_error: 0.2080 - val_loss: 0.0164 -
val_root_mean_squared_error: 0.2027
```

```
# Load the best performing model and evaluate the performance.
dkt_lstm.load_weights(params['best_model_weights'])
dkt_lstm.evaluate(df_x_test, df_y_test, verbose=params['verbose'],
return_dict=True)
```

```
38/38 [==============================] - 0s 3ms/step - loss: 0.0172 -
root_mean_squared_error: 0.2052
```

```
{'loss': 0.017214568331837654, 'root_mean_squared_error':
0.2052297443151474}
```

Time Series: Data Preparation

We can modify our model to predict after n weeks whether students will pass or fail the class.

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

```
   user_id  label-pass-fail
0     1593              0.0
1     1626              1.0
2     1787              1.0
3     1824              1.0
4     1836              1.0
```

We choose `n = 5` weeks and therefore drop all the data from weeks 5 through 10. Since this problem refers to early peformance prediction, we can only train on weeks 1 through 4 of student data.

```
n = 5
```

We preprocess our data for this task:

- `mooc_labels` should become a NumPy array of size `num_of_students`.
- `df_x` should become a NumPy array of size `num_of_students * n * num_of_features`.

```
df_x_binary = df_x[:, :n, :]
df_y_binary = mooc_labels['label-pass-fail'].values.reshape(-1, 1)
```

Finally, we split the data into train/validation/test sets. We do a stratified split (on label-pass-fail) so that the classes are representatively balanced across each of our dataset divisions.

```
# Split into training and test sets.
df_x_binary_train, df_x_binary_test, df_y_binary_train,
df_y_binary_test = train_test_split(

df_x_binary,

df_y_binary,

test_size=0.2,

random_state=0,

stratify=df_y_binary)

# Split training into training and validation sets.
df_x_binary_train_val, df_x_binary_val, df_y_binary_train_val,
df_y_binary_val = train_test_split(

df_x_binary_train,

df_y_binary_train,

test_size=0.2,

random_state=0,

stratify=df_y_binary_train)
```

Time Series: Model Creation

Now, we can again create an lstm model, which takes the features up to week 5 as an input and predicts the pass/fail label.

Your Turn (Code)

Fill in the create_model function for time-series prediction using an LSTM below. You can refer to the DKT task and the above tracing task for example code.

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

    # Create an LSTM model architecture.
    inputs = ...

    # YOUR CODE HERE

    # Compile the model with our loss functions, optimizer, and
metrics.
    model.compile(loss=tf.keras.losses.binary_crossentropy,
                  optimizer=params['optimizer'],
                  metrics=[tf.keras.metrics.AUC(), 'binary_accuracy'])

    return model

time_series_lstm = create_model_lstm_mooc_binary(num_features, 1,
params)
```

Time Series: Model Fitting and Evaluation

```
# This model should take ~30 seconds to train.

# We save only the best model during the training process.
ckp_callback =
tf.keras.callbacks.ModelCheckpoint(params['best_model_weights'],
                                   save_best_only=True,
save_weights_only=True)

# Fit the DKT LSTM on DSP1 data.
history = time_series_lstm.fit(df_x_binary_train_val,
                               df_y_binary_train_val,
                               epochs=params['epochs'],
                               validation_data=(df_x_binary_val,
df_y_binary_val),
                               callbacks=[ckp_callback],
                               verbose=params['verbose'])
```

To evaluate performance of the model, we can also use `predict` instead of `evaluate` to get the actual predictions of the model. We can then compute any evaluation metric based on the true labels and the model predictions.

```python
# Load the best version of the the trained model and evaluate its
performance on the test_set.
time_series_lstm.load_weights(params['best_model_weights'])
predictions = time_series_lstm.predict(df_x_binary_test)
bac = balanced_accuracy_score(df_y_binary_test, predictions>0.5)
auc = roc_auc_score(df_y_binary_test,predictions)
print("Balanced accuracy: ", bac)
print("AUC: ", auc)
```

```python
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: "),
}
```

```python
### Share the bac with us
bac_time_series = bac
send(bac_time_series, 1)
```

Time Series: Hyperparameter Tuning

Your Turn (Code)

```python
# Modify the dictionary of parameters so that each parameter maps to a
list of possibilities.
# You can tune any hyperparameter that you want. We advice to stay
with a small grid...
params_space = ...
```

```python
# Conduct the gridsearch over hyperparameters.
# This cell should take ~3 minutes to run.
results = {}

# For each parameter setting in the grid search of parameters
for params_i in params_grid:
    ...

# Sort candidate parameters according to their accuracy
results = sorted(results.items(), key=lambda x: x[1]
['binary_accuracy'], reverse=True)

# Obtain the best parameters
best_params = results[0][0]
best_params
```

```python
# Load the best model variant from the hyperparameter gridsearch
time_series_lstm.load_weights(params['best_model_weights'] +
```

```
str(best_params))
predictions = time_series_lstm.predict(df_x_binary_test)
bac = balanced_accuracy_score(df_y_binary_test, predictions>0.5)
auc = roc_auc_score(df_y_binary_test,predictions)
print("Balanced accuracy: ", bac)
print("AUC: ", auc)

### Share the bac with us
bac_hyperparam_tuning = bac
send(bac_hyperparam_tuning, 2)
```

Student Notebook - Lecture 10

Clustering algorithms are important techniques for structural discovery in the data. In these lecture, we will solve two tasks. In a first task, you will observe and discuss performance of K-Means clustering on synthetic data. In the second task, you will yourself cluster students of a flipped classroom using spectral clustering.

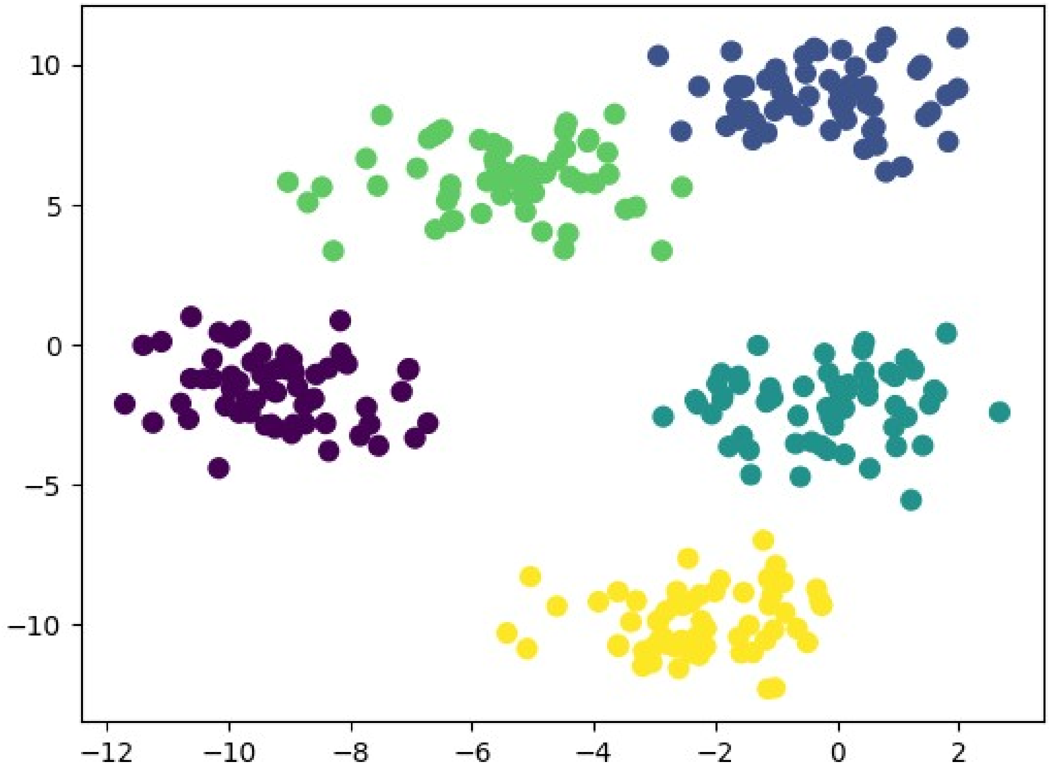
```
#Important imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from scipy.spatial import distance
from scipy.sparse.csgraph import laplacian
from scipy import linalg

from sklearn.datasets import make_blobs, make_circles, make_moons
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score,
rand_score
from sklearn.metrics.pairwise import pairwise_kernels
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import spectral_embedding

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

K-Means Clustering - Examples

K-Means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. K-means clustering minimizes within-cluster variances (squared Euclidean distances). In a first step, we look at K-Means clustering in detail. We first generate a synthetic example data set and write a function able to extract the intermediate cluster assignments from the K-Means algorithm.

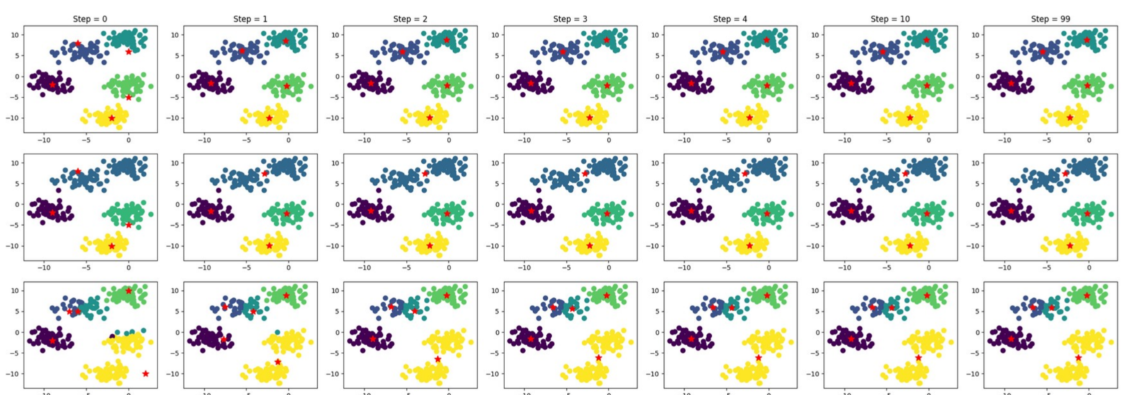
```
n_samples = 300
X, y= make_blobs(n_samples=n_samples, centers=5,
                 cluster_std=1.2, random_state=2010)
plt.scatter(X[:, 0], X[:, 1], s=50, c=y)
```

```
<matplotlib.collections.PathCollection at 0x7fa7c63e5610>
```




```
def getKMeansSteps(X, k, centroids):

    y_pred = []
    intermediate_centers = []
    k_means = KMeans(n_clusters=k, max_iter=1, init=centroids,
n_init=1)
    c_hat = centroids
    for i in range(100):
        intermediate_centers.append(c_hat)
        y_hat = k_means.fit_predict(X)
        c_hat = k_means.cluster_centers_
        y_pred.append(y_hat)
        k_means = KMeans(n_clusters=k, max_iter=1, init=c_hat,
n_init=1)

    return y_pred, intermediate_centers

c1 = np.array([[-9,-2], [-6, 8], [0, 6], [0, -5], [-2, -10]])
y_pred_1, centers_1 = getKMeansSteps(X, 5, c1)

c2 = np.array([[-9,-2], [-6, 8], [0, -5], [-2, -10]])
y_pred_2, centers_2 = getKMeansSteps(X, 4, c2)

c3 = np.array([[-9,-2], [-7, 5], [-6, 5], [0, 10], [2, -10]])
y_pred_3, centers_3 = getKMeansSteps(X, 5, c3)
```

```
steps = [0, 1, 2, 3, 4, 10, 99]

fig, ax = plt.subplots(3, 7, figsize=(30, 10))

ind = 0
for i in steps:
    ax[0,ind].scatter(X[:, 0], X[:, 1], s=50, c = y_pred_1[i]);
    ax[0,ind].plot(centers_1[i].transpose()[0],
centers_1[i].transpose()[1], marker='*', color = 'red', ls='none',
ms=10)
    ax[0,ind].set_title('Step = ' + str(i))
    ind = ind+1

ind = 0
for i in steps:
    ax[1,ind].scatter(X[:, 0], X[:, 1], s=50, c = y_pred_2[i]);
    ax[1,ind].plot(centers_2[i].transpose()[0],
centers_2[i].transpose()[1], marker='*', color = 'red', ls='none',
ms=10)
    ind = ind+1

ind = 0
for i in steps:
    ax[2,ind].scatter(X[:, 0], X[:, 1], s=50, c = y_pred_3[i]);
    ax[2,ind].plot(centers_3[i].transpose()[0],
centers_3[i].transpose()[1], marker='*', color = 'red', ls='none',
ms=10)
    ind = ind+1

plt.show()
```



Your Turn - Task 1

The above plots are three examples of the K-Means algorithm on the same synthetic data set with k=5 clusters. Each row corresponds to one example run of the K-Means algorithm. Each column shows the centroids (red stars) as well as the cluster assignments after an intermediate step of the algorithm. Specifically, we visualize the following time steps: 0, 1, 2, 3, 4, 10, and 99. Note that the maximum number of iterations was 100, so time step 99 corresponds to the final solution. What do you observe? Does K-Means recover the original clusters? Discuss your observations and send them to us.

```
observation_example1 = ""
send(observation_example1, 1)
```

```
observation_example2 = ""
send(observation_example2, 2)
```

```
observation_example3 = ""
send(observation_example3, 3)
```

Spectral Clustering

In contrast to K-Means, spectral clustering makes no assumption about the form/shape of the clusters. The different data points are treated as nodes of graphs and the clustering is done based on connectivity of the graph. In a first step, we use spectral clustering to cluster the two simulated data sets. We again assume that the correct number of clusters is known a-priori. We will use an unnormalized Laplacian for all our experiments.

We compute the pairwise similarity matrix using the radial basis function or Gaussian kernel, defined as:

$$s_{ij} = s(x_i, x_j) = \exp\left(-\gamma \| x_i - x_j \|^2\right)$$

where γ is a hyperparameter that must be tuned, controlling the width of the kernel.

Once we have the similarity matrix S, we need to compute the adjacency matrix W.

```
def get_adjacency(S, connectivity='full'):
    """
    Computes the adjacency matrix
    :param S: np array of similarity matrix
    :param connectivity: type of connectivity
    :return: adjacency matrix
    """

    if(connectivity=='full'):
        adjacency = S
    elif(connectivity=='epsilon'):
        epsilon = 0.5
        adjacency = np.where(S > epsilon, 1, 0)
    else:
        raise RuntimeError('Method not supported')

    return adjacency
```

We then can implement the spectral clustering algorithm, giving the adjacency matrix W as an input.

```python
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
```

For spectral clustering, we can for example use the eigengap heuristic or the Silhouette score to determine the optimal number of clusters. Next, we write functions to compute spectral clustering for a varying number of k and visualize these two heuristics.

```python
def plot_metrics(n_clusters_list, metric_dictionary):
    """
    Plots metric dictionary (auxilary function)
    [Optional]

    :param n_clusters_list: List of number of clusters to explore
    :param metric_dictionary:
    """
    fig = plt.figure(figsize=(12, 10), dpi=80)
    i = 1

    for metric in metric_dictionary.keys():
        plt.subplot(3, 2, i)
```

```
        if metric == 'Eigengap':
            clusters = len(n_clusters_list)
            eigenvals_sorted = metric_dictionary[metric]
            plt.scatter(range(1, len(eigenvals_sorted[:clusters * 2])
+ 1), eigenvals_sorted[:clusters * 2])
            plt.xlabel('Eigenvalues')
            plt.xticks(range(1, len(eigenvals_sorted[:clusters * 2]) +
1))
        else:
            plt.plot(n_clusters_list, metric_dictionary[metric], '-o')
            plt.xlabel('Number of clusters')
            plt.xticks(n_clusters_list)
        plt.ylabel(metric)
        i += 1


def get_heuristics_spectral(A, n_clusters_list, plot=True):
    """
    Calculates heuristics for optimal number of clusters with Spectral
Clustering

    :param A: affinity matrix
    :param n_clusters_list: List of number of clusters to explore
    :plot: bool, plot the metrics if true
    """
    silhouette_list = []
    distortion_list = []
    bic_list = []
    eigengap_list = []
    davies_bouldin_list = []

    for k in n_clusters_list:

        kmeans, proj_X, eigenvals_sorted = spectral_clustering(A, k)
        y_pred = kmeans.labels_

        if k == 1:
            silhouette = np.nan
        else:
            silhouette = silhouette_score(proj_X, y_pred)
        silhouette_list.append(silhouette)


    metric_dictionary = {
                         'Silhouette': silhouette_list,
                         'Eigengap': eigenvals_sorted,
                        }

    if(plot):
```

```
        plot_metrics(n_clusters_list, metric_dictionary)
    else:
        return metric_dictionary
```

Spectral Clustering on Flipped Classroom Data

Given the favorable properties of spectral clustering, we will use it to cluster the students of our flipped classroom data set. We first parse and preprocess the data.

```
df = pd.read_csv('{}/aggregated_extended_fc.csv'.format(DATA_DIR))
df = df.fillna('NaN')
df.head()
```

```
   user  ch_num_sessions  ch_time_in_prob_sum  ch_time_in_video_sum  \
0     0              1.9               2334.4                2951.8
1     1              3.4               1698.4                9227.8
2     2              5.3               2340.6               10801.3
3     3              2.8               2737.1                8185.5
4     4              2.5               3787.3                7040.0

   ch_ratio_clicks_weekend_day  ch_total_clicks_weekend  \
0                     0.850000                     16.8
1                     0.567500                      4.0
2                    26.562274                     94.6
3                     3.691250                     13.5
4                     1.543889                     58.4

   ch_total_clicks_weekday  ch_time_sessions_mean
ch_time_sessions_std  \
0                     38.1            1392.858333
790.762032
1                    179.4            3068.720238
1257.504407
2                    129.2            1750.289268
1024.134043
3                     46.4           20203.590260
656.052901
4                     64.9            3373.908333
1363.320365

   bo_delay_lecture  ...  la_weekly_prop_watched_mean  \
0      55068.387500  ...                     0.245714
1      -2883.367738  ...                     0.748868
2      10027.216667  ...                     0.354487
3      27596.864484  ...                     0.370000
4       -914.633333  ...                     0.030000

   la_weekly_prop_interrupted_mean  la_weekly_prop_interrupted_std  \
0                         0.024286                             0.0
1                         0.074683                             0.0
```

```
2                          0.026667                                     0.0
3                          0.014286                                     0.0
4                          0.000000                                     0.0

   la_weekly_prop_replayed_mean  la_weekly_prop_replayed_std  \
0                      0.010000                          0.0
1                      0.066456                          0.0
2                      0.059915                          0.0
3                      0.020000                          0.0
4                      0.020000                          0.0

   la_frequency_action_video_play  grade  gender       category
year
0                        0.179203   4.50     NaN            NaN  Y2-
2018-19
1                        0.332424   4.50       M  Suisse.Autres  Y2-
2018-19
2                        0.284407   5.25       M     Suisse.PAM  Y2-
2018-19
3                        0.108774   4.50       F  Suisse.Autres  Y2-
2018-19
4                        0.199775   4.75       F         France  Y2-
2018-19

[5 rows x 38 columns]
```

Specifically, we are interested in clustering the students based on their behavior in the course. We investigate two different type of behaviors. The first behavior is related to students effort. We use the following three features as indicators: `ch_time_in_prob_sum`, `ch_time_in_video_sum`, `ch_total_clicks_weekend`, `ch_total_clicks_weekday`. We sum up the time in problems and videos to obtain the total time spent on the platform. Similarly, we also sum up the number of clicks in problems and videos to obtain the total number of clicks.

The second behavior is related to students proactivity in the course. Use the following two features as indicators of how proactive the students are: `ma_content_anti`, `bo_delay_lecture` and follow the previous steps.

```
df['ch_time_sum'] = df.ch_time_in_prob_sum + df.ch_time_in_video_sum
df['ch_total_clicks'] = df.ch_total_clicks_weekend +
df.ch_total_clicks_weekday
```

Your Turn - Task 2

Pick one of the two behaviors (effort or proactivity) and use spectral clustering to cluster students according to this behavior.

In a first step you will need to normalize or standardize the features.

```
# Data standardization/normalization
from sklearn.preprocessing import normalize
time_normed = np.linalg.norm(df['ch_time_sum'])
clicks_normed = np.linalg.norm(df['ch_total_clicks'])
```

Next, compute the pairwise similarity matrices separately for each feature using a Gaussian kernel. We can then simply sum up the similarity matrices up to obtain the overall similarity matrix S.

```
# Pariwise kernels
# Hint: use scikit-learn pairwise_kernels
S_time_normed = pairwise_kernels(np.array(time_normed).reshape(-1,1))
S_clicks_normed = pairwise_kernels(np.array(clicks_normed).reshape(-
1,1))
total_S = S_time_normed+S_clicks_normed
```

Next, we compute the adjacency matrix W.

```
# Compute adjacency matrix
# Hint: get_adjacency function above
W = get_adjacency(total_S)
```

Finally, we perform a spectral clustering for k=2,...,10 and compute the Silhouette score as well as the eigengap heuristic.

```
# Compute spectral clustering, heuristics, and visualization
# Hint: get_heuristics_spectral function above
get_heuristics_spectral(W, [2,3,4,5,6,7,8,9,10])
```

```
/usr/local/lib/python3.8/dist-packages/scipy/sparse/linalg/_eigen/
arpack/arpack.py:1592: RuntimeWarning: k >= N for N * N square matrix.
Attempting to use scipy.linalg.eigh instead.
  warnings.warn("k >= N for N * N square matrix. "

----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Input In [54], in <cell line: 3>()
      1 # Compute spectral clustering, heuristics, and visualization
      2 # Hint: get_heuristics_spectral function above
----> 3 get_heuristics_spectral(W, [2,3,4,5,6,7,8,9,10])

Input In [36], in get_heuristics_spectral(A, n_clusters_list, plot)
     41 davies_bouldin_list = []
     43 for k in n_clusters_list:
---> 45     kmeans, proj_X, eigenvals_sorted = spectral_clustering(A,
k)
     46     y_pred = kmeans.labels_
     48     if k == 1:
```

```
Input In [35], in spectral_clustering(W, n_clusters, random_state)
     26 # Cluster the points using k-means clustering
     27 kmeans = KMeans(n_clusters=n_clusters, random_state =
random_state)
---> 28 kmeans.fit(proj_X)
     30 return kmeans, proj_X, eigenvals_sorted

File
/usr/local/lib/python3.8/dist-packages/sklearn/cluster/_kmeans.py:1146
, in KMeans.fit(self, X, y, sample_weight)
   1112 """Compute k-means clustering.
   1113
   1114 Parameters
   (...)
   1135     Fitted estimator.
   1136 """
   1137 X = self._validate_data(
   1138     X,
   1139     accept_sparse="csr",
   (...)
   1143     accept_large_sparse=False,
   1144 )
-> 1146 self._check_params(X)
   1147 random_state = check_random_state(self.random_state)
   1148 sample_weight = _check_sample_weight(sample_weight, X,
dtype=X.dtype)

File
/usr/local/lib/python3.8/dist-packages/sklearn/cluster/_kmeans.py:947,
in KMeans._check_params(self, X)
    945 # n_clusters
    946 if X.shape[0] < self.n_clusters:
--> 947     raise ValueError(
    948         f"n_samples={X.shape[0]} should be >=
n_clusters={self.n_clusters}."
    949     )
    951 # tol
    952 self._tol = _tolerance(X, self.tol)

ValueError: n_samples=1 should be >= n_clusters=2.
```

```
# What do you observe? What is the optimal number of clusters? Do both
metrics agree?
observation = ""
send(observation, 4)
```

Your Turn - Task 3

If you have time, replicate the analyses for the second feature group.

```
# Replicate analysis for second feature group
```

```python
# What do you observe? What is the optimal number of clusters? Do both
metrics agree?
observation = ""
send(observation, 5)
```

Student Notebook - Lecture 11

In this lecture, we will investigate different methods for clustering time series:

- Aggregating the data
- Using distance metrics that can handle vectors (e.g. Euclidean distance)
- Using dynamic time warping

We will use spectral clustering for all experiments. Furthermore, we will again use a synthetic data set to explore the characteristics of the different approaches.

Using our synthetic data, we are interested in exploring procrastination. For this purpose, we will cluster the data of 30 high-school students based on their usage of an academic learning platform. The dataset contains the number of hours per biweek of the year that each student spent on the platform.

The dataset is described by the following columns:

- student id: unique student identifier-
- biweek of the year: number of the biweek of the school year. Biweek 0 refers to the first two wekeks of the school year.
- hours: number of hours the student spent on the platform for that particular biweek-
- student type: expert tagging of student behavior, where (1) is procrastinators, (2) regular students, and (3) precrastinators. We will use the expert label as ground truth for the clustering.

```
#Important imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tslearn.metrics import cdist_dtw
from sklearn.preprocessing import StandardScaler

from scipy.spatial import distance
from scipy.sparse.csgraph import laplacian
from scipy import linalg
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.metrics import silhouette_score
from sklearn.metrics.pairwise import pairwise_kernels
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import spectral_embedding

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

```
df = pd.read_csv('{}/hours_biweek_students.csv'.format(DATA_DIR))
df.head()
```

```
   student_id  biweek_of_year      hours  student_type
0           0               0  39.915507             3
1           0               1  32.356082             3
2           0               2  17.456692             3
3           0               3  16.012725             3
4           0               4  15.859812             3
```

In a first step, we extract a time series (of biweeks) for each student.

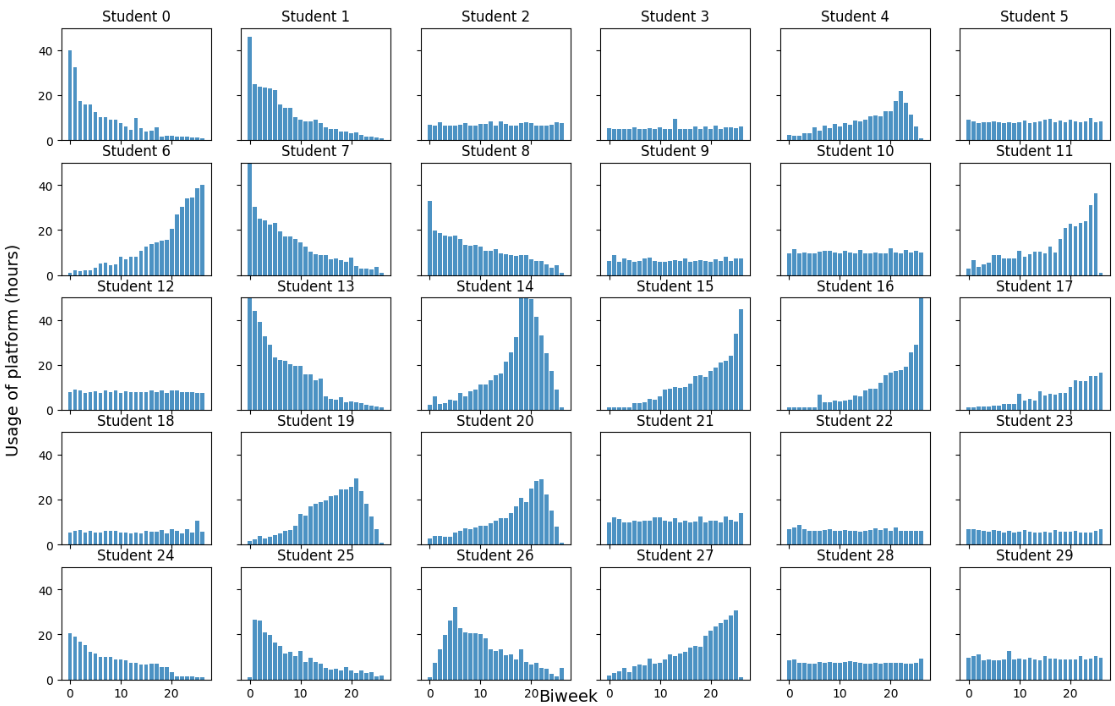
```
def get_time_series(df):
    """
    reshapes DataFrame from long to wide and returns an np.array
    :param df: pd.DataFrame with data in long format
    :return: np.array with reshaped data
    """
    df_array = (df.sort_values(['student_id', 'biweek_of_year'],
ascending=True)
                .groupby('student_id')
                .agg({'hours': lambda x: list(x)}))

    data = np.asarray(df_array.hours.values.tolist())
    return data


data = get_time_series(df)
data.shape
```

```
(30, 27)
```

We then plot the time series data for each student. The three student types are visually very well separable.

```
def plot_students(data):
    """
    Plot the students time-series
    :param data: np.array with students' time-series
    :return:
    """
    students, biweeks = data.shape
    fig, axs = plt.subplots(5, 6, figsize=(16, 10), sharex=True,
                            sharey=True, facecolor='w', edgecolor='k')
    axs = axs.ravel()
    for i in range(students):
        axs[i].bar(range(biweeks), data[i], alpha=0.8)
        axs[i].set_ylim([0, 50])
        axs[i].set_title('Student {0}'.format(i))
    fig.text(0.5, 0.09, 'Biweek', va='center', ha='center',
fontsize=14)
```

```
    fig.text(0.09, 0.5, 'Usage of platform (hours)', va='center',
ha='center', rotation='vertical', fontsize=14)


plot_students(data)
```



Next, we implement some helper functions needed to perform spectral clustering. Specifically, we provide the following functions:

- get_adjacency: computes the adjacency matrix W from a pairwise similarity matrix S
- spectral_clustering: performs spectral clustering for a given number of clusters k, based on an adjacency matrix W
- get_heuristics_spectral: performs spectral clustering for k=2,...,n clusters and computes the Silhouette score and eigengap heuristic for each k
- plot_metrics: visualizes the heuristics for the number of clusters

```
def get_adjacency(S, connectivity='full'):
    """
    Computes the adjacency matrix
    :param S: np array of similarity matrix
    :param connectivity: type of connectivity
    :return: adjacency matrix
    """

    if(connectivity=='full'):
        adjacency = S
    elif(connectivity=='epsilon'):
        epsilon = 0.5
        adjacency = np.where(S > epsilon, 1, 0)
```

```python
    else:
        raise RuntimeError('Method not supported')

    return adjacency

def spectral_clustering(W, n_clusters, random_state=111):
    """
    Spectral clustering
    :param W: np array of adjacency matrix
    :param n_clusters: number of clusters
    :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

def plot_metrics(n_clusters_list, metric_dictionary):
    """
    Plots metric dictionary (auxilary function)
    [Optional]

    :param n_clusters_list: List of number of clusters to explore
    :param metric_dictionary:
    """
    fig = plt.figure(figsize=(12, 10), dpi=80)
    i = 1

    for metric in metric_dictionary.keys():
        plt.subplot(3, 2, i)
```

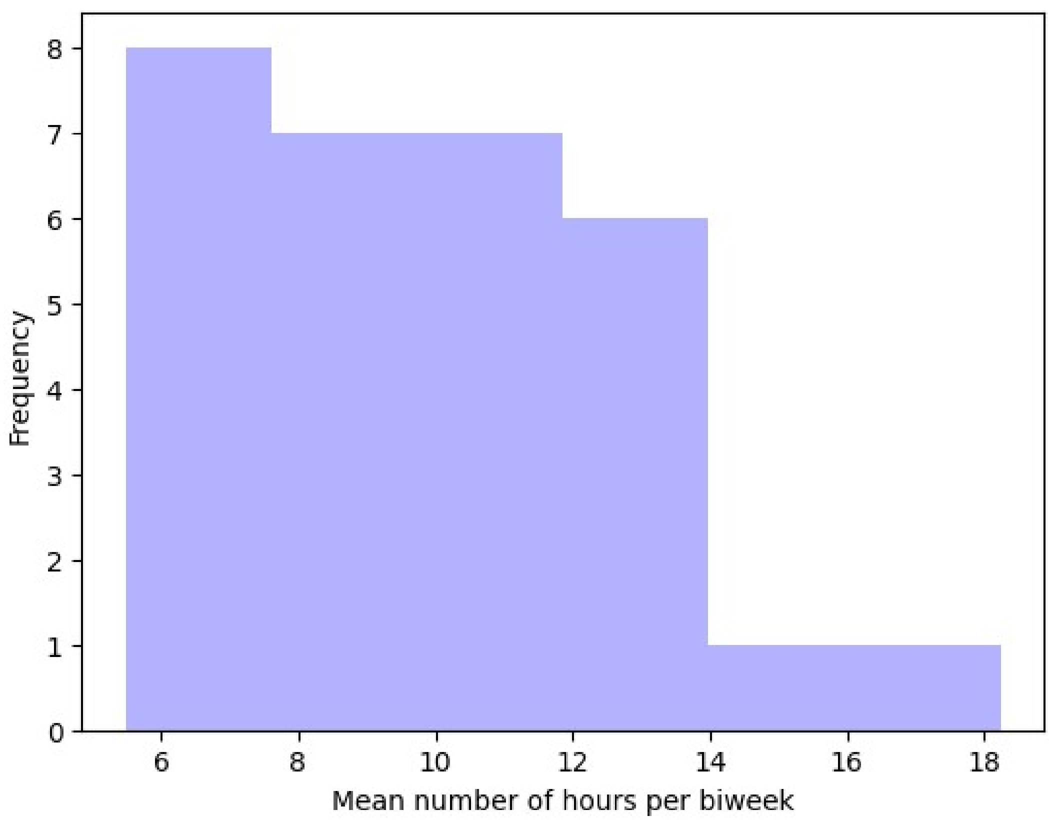
```python
        if metric == 'Eigengap':
            clusters = len(n_clusters_list)
            eigenvals_sorted = metric_dictionary[metric]
            plt.scatter(range(1, len(eigenvals_sorted[:clusters * 2])
+ 1), eigenvals_sorted[:clusters * 2])
            plt.xlabel('Eigenvalues')
            plt.xticks(range(1, len(eigenvals_sorted[:clusters * 2]) +
1))
        else:
            plt.plot(n_clusters_list, metric_dictionary[metric], '-o')
            plt.xlabel('Number of clusters')
            plt.xticks(n_clusters_list)
        plt.ylabel(metric)
        i += 1

def get_heuristics_spectral(W, n_clusters_list, plot=True):
    """
    Calculates heuristics for optimal number of clusters with Spectral
Clustering

    :param W: np array of adjacency matrix
    :param n_clusters_list: List of number of clusters to explore
    :plot: bool, plot the metrics if true
    """
    silhouette_list = []
    eigengap_list = []

    df_labels = pd.DataFrame()

    for k in n_clusters_list:

        kmeans, proj_X, eigenvals_sorted = spectral_clustering(W, k)
        y_pred = kmeans.labels_
        df_labels[str(k)] = y_pred

        if k == 1:
            silhouette = np.nan
        else:
            silhouette = silhouette_score(proj_X, y_pred)
        silhouette_list.append(silhouette)


    metric_dictionary = {
                         'Silhouette': silhouette_list,
                         'Eigengap': eigenvals_sorted,
                        }

    if(plot):
        plot_metrics(n_clusters_list, metric_dictionary)
        return df_labels
```

```python
    else:
        return df_labels, metric_dictionary
```

1 - Aggregated Data

The first method we will explore is aggregating features over time. In our example, we will use the mean for the aggreation. We therefore first compute the mean value of our feature (number of hours per biweek) over the whole time series.

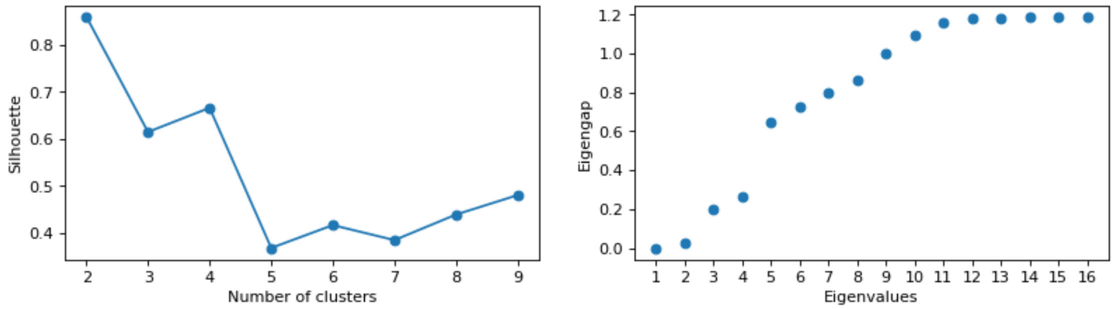
```python
# compute the average of the feature over the whole time series
aggregated_data = np.mean(data, axis = 1)

# plot the histogram of the feature for all students
plt.hist(aggregated_data, bins = 6, alpha = 0.3, color = 'blue')
plt.xlabel('Mean number of hours per biweek')
plt.ylabel('Frequency');
```



We then again build a similarity matrix and a similarity graph and perform spectral clustering for k=2,...10 clusters. We visualize the Silhouette score and the eigengap heuristic.

```python
S = pairwise_kernels(aggregated_data.reshape(-1,1), metric='rbf', gamma=1)

W =  get_adjacency(S)
```

```
n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```



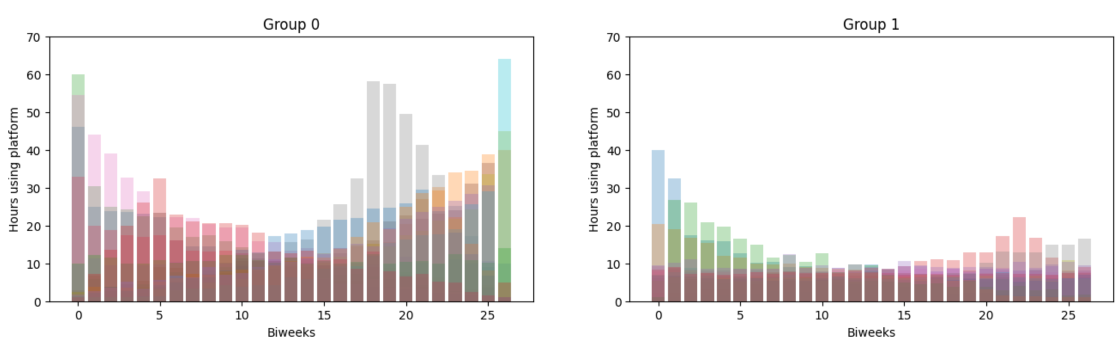
Next, we want to visualize the time series of the students in the different clusters. We implement a function `view_clusters`, which visualizes the average behavior for each cluster. We also implement a function `plot_students_group`, which visualizes the time series of the students in each group.

```
def view_clusters(data, labels, ylim = 70, xlabel= 'Biweeks'):
    """
    visualize the different time-series of students belonging to each
cluster.
    :param data: np.array with students' time-series
    :param labels: np.array predicted labels from clustering model
    :return:
    """
    _, biweeks = data.shape
    clusters = np.unique(labels).shape[0]
    fig, axs = plt.subplots(1, clusters, figsize=(16, 4),
facecolor='w', edgecolor='k')
    axs = axs.ravel()

    for i in range(clusters):
        students_cluster = data[labels == i]
        number_students = students_cluster.shape[0]
        for student in range(number_students):
            axs[i].bar(range(biweeks), students_cluster[student],
alpha=0.3)

        axs[i].set_ylim([0, ylim])
        axs[i].set_title('Group {0}'.format(i))
        axs[i].set_ylabel('Hours using platform')
        axs[i].set_xlabel(xlabel)

def plot_students_group(data, labels):
    """
    Plot the students time-series
    :param data: np.array with students' time-series
    :param labels: pd.Series indicating the labels of the students
    :return:
```

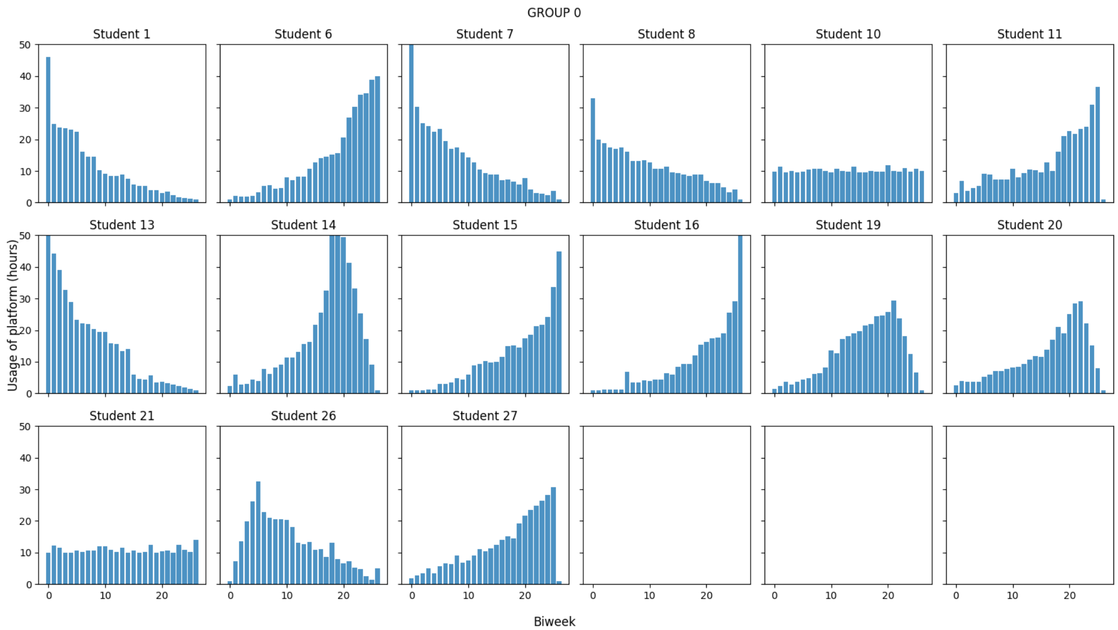
```
    """
    for group in np.unique(labels):
        subdata = data[labels==group]
        subindex = labels[labels==group].index
        students, biweeks = subdata.shape

        rows = int(np.ceil(students/6))
        fig, axs = plt.subplots(rows, 6, figsize=(16, rows*3),
sharex=True,
                                sharey=True, facecolor='w', edgecolor='k')

        axs = axs.ravel()
        for i in range(students):
            axs[i].bar(range(biweeks), subdata[i], alpha=0.8)
            axs[i].set_ylim([0, 50])
            axs[i].set_title('Student {0}'.format(subindex[i]))

        fig.suptitle('GROUP {}'.format(group))
        fig.supxlabel('Biweek')
        fig.supylabel('Usage of platform (hours)')
        plt.tight_layout()
        plt.show()
```

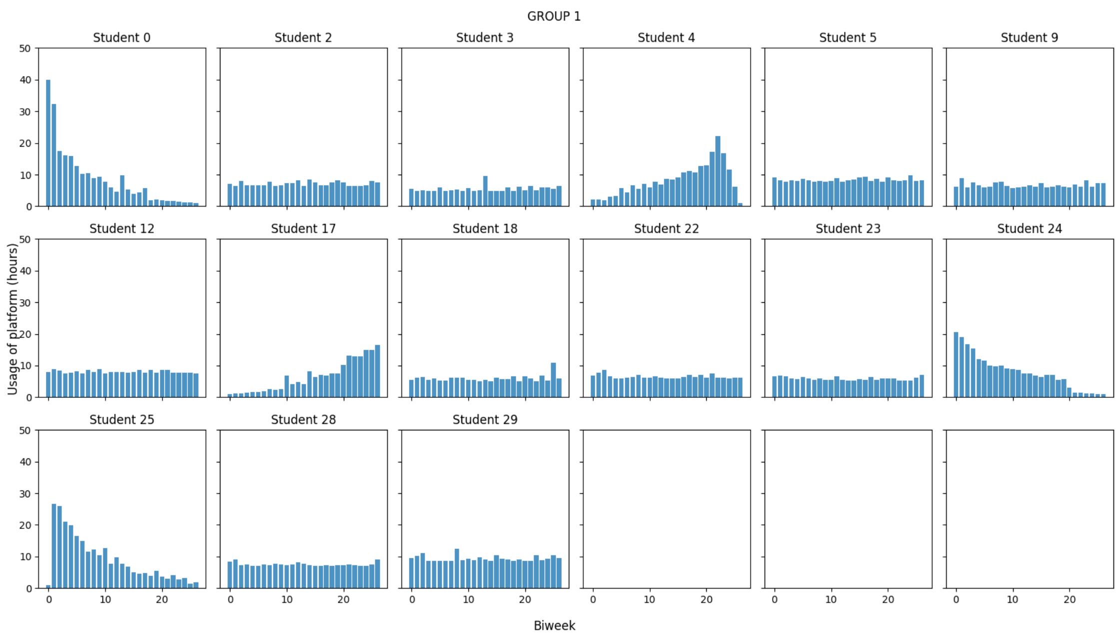
Both the Silhouette score and the eigengap heuristic suggest that the optimal number of clusters is 2. We visualize the mean behavior as well as the time series data of the students in each group.

```
k = 2
view_clusters(data, df_labels[str(k)])
```




```
plot_students_group(data, df_labels[str(k)])
```





Your Turn - Task 1

Discuss your observations and send them to us through the SpeakUp Chat (or through this notebook):

- Can you interpret the obtained clusters?
- Is the approach able to retrieve the procrastination patterns? If not, why not?

```
# Notebook option
answer = """
Can you interpret the obtained clusters?
"""
```

```python
send(answer, 11)

answer = """
Is the approach able to retrieve the procrastination patterns? If not,
why not?
"""
send(answer, 12)
```

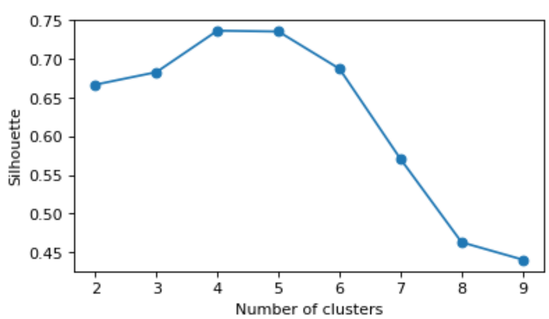
2 - Assuming fixed time intervals

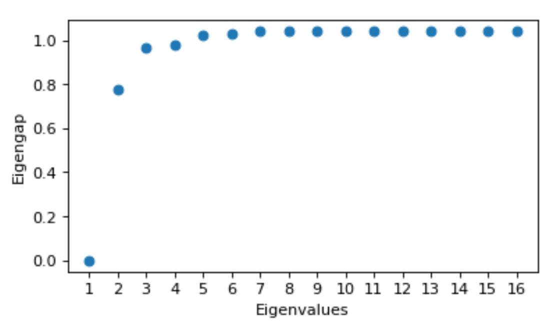
Given the fact that all the students have the same number of biweeks (worth a year), the time series of each student has the same length. We can therefore simply use the Euclidean distance to compute the pairwise distances. In order to avoid clustering by the absolute number of hours and capture students individual differences over the semester (i.e. students who work more at the beginning of the semester and then less over the course of the semester) we normalize the data for each student (i.e. within the student's time series).

```python
X = data
norms = np.linalg.norm(X, axis=1)
data_normalized = X / norms[:, np.newaxis]
```

We then again perform a spectral clustering and visualize the heuristics.

```python
S = pairwise_kernels(data_normalized, metric='rbf', gamma=1)
W =  get_adjacency(S)

n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```





Your Turn - Task 2

Discuss your observations and send them to us through the SpeakUp Chat (or through this notebook):

- What is the optimal number of clusters k*?
- Can you interpret the obtained clusters?Visualize the average cluster behavior as well as the time series per student of feach group for k*.

Hint: make use of the functions `view_clusters` and `plot_students_group`.

```python
# Notebook option
answer = """
What is the optimal number of clusters k*?
"""
send(answer, 21)

answer = """
Can you interpret the obtained clusters?Visualize the average cluster
behavior as well as the time series per student of feach group for k*.
"""
send(answer, 22)

# YOUR VISUALIZATION CODE HERE

send(plt, 23)
```

3- Dynamic Time Warping

Dynamic time warping allows us to align to sequences in an optimal way by choosing a window size w larger than 0.

We first implement a distance function for computing the dynamic time warping distance for a fixed window size w.

```python
def get_distance_matrix(X, metric='euclidean', window=2):
    """
    calculates distance matrix given a metric
    :param X: np.array with students' time-series
    :param metric: str distance metric to compute
    :param window: int for DTW
    :return: np.array with distance matrix
    """
    norms = np.linalg.norm(X, axis=1)
    data_normalized = X / norms[:, np.newaxis]

    if metric == 'dtw':
        distance_matrix = cdist_dtw(data_normalized,
                                    global_constraint='sakoe_chiba',
                                    sakoe_chiba_radius=window)
    else:
        distance_vector = distance.pdist(data_normalized, metric)
        distance_matrix = distance.squareform(distance_vector)
    return distance_matrix
```

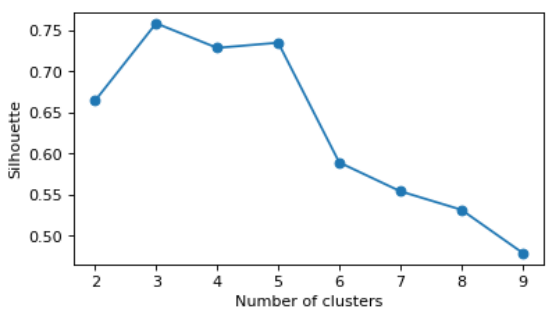
We then also implement a function that computes the similarity matrix for us based on the pairwise distances.

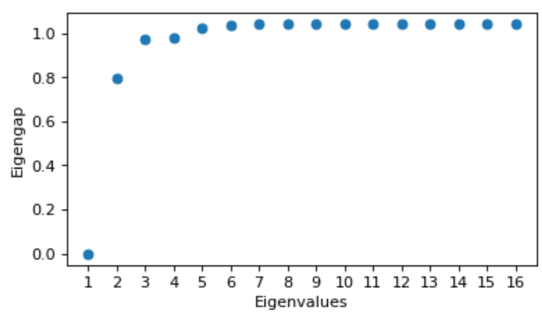
```python
def get_affinity_matrix(D, gamma=1):
    """
    calculates affinity matrix from distance matrix
    :param D: np.array distance matrix
```

```
    :param gamma: float coefficient for Gaussian Kernel
    :return:
    """
    S = np.exp(-gamma * D ** 2)
    return S
```

We then compute pairwise distances using a window size of 6. Subsequently, we compute the similarity matrix and the adjacency matrix and then again perform spectral clustering and visualize the cluster heuristics.

```
D = get_distance_matrix(data, metric='dtw', window=6)

S = get_affinity_matrix(D)
W = get_adjacency(S)

n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```





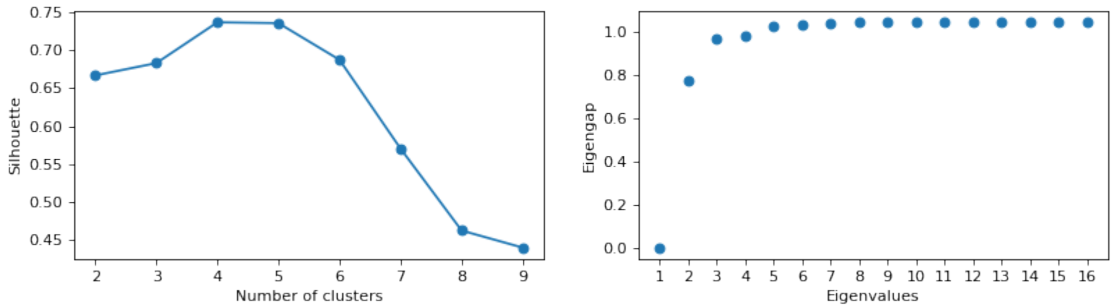
Your Turn - Task 3

Investigate the clustering results when using DTW (with w = 6). Discuss your observations and send them to us through the SpeakUp Chat (or through this notebook):

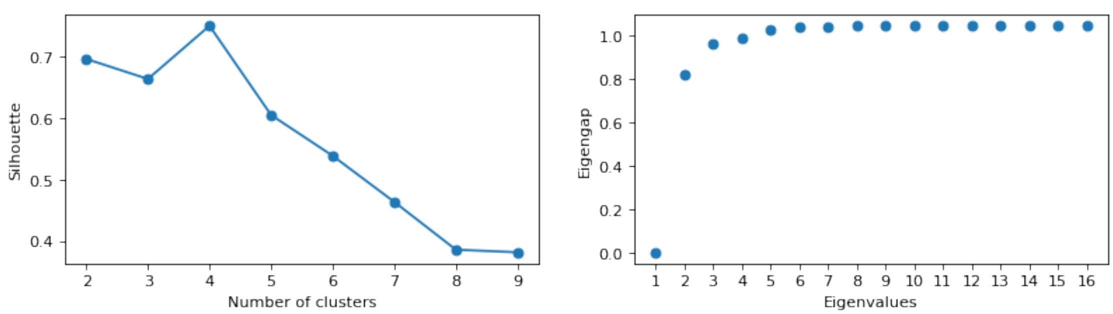
- What is the optimal number of clusters k*?
- Can you interpret the obtained cluster? Visualize the average cluster behavior as well as the time series per student of feach group for k*.
- How do the results change for w = 0 and w = 27?

```
# Notebook option
answer = """
What is the optimal number of clusters k*?
"""
send(answer, 31)

answer = """
Can you interpret the obtained cluster? Visualize the average cluster
behavior as well as the time series per student of feach group for k*.
"""
send(answer, 32)

answer = """
```

```
How do the results change for w = 0 and w = 27?
"""
send(answer, 33)

# YOUR VISUALIZATION CODE HERE
send(plt, 34)

# windows size 0
D = get_distance_matrix(data, metric='dtw', window=0)
S = get_affinity_matrix(D)
W = get_adjacency(S)
n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```




```
# YOUR VISUALIZATION CODE HERE
send(plt, 35)

# windows size 27
D = get_distance_matrix(data, metric='dtw', window=27)
S = get_affinity_matrix(D)
W = get_adjacency(S)
n_cluster_list = range(2, 10)
df_labels = get_heuristics_spectral(W, n_cluster_list)
```




```
# YOUR VISUALIZATION CODE HERE
send(plt, 36)
```

Your Turn - Task 4

Discuss the clustering results and send us your observations:

- What is the optimal number of clusters?

- Can you interpret the obtained clusters? Hint: you can use the function `view_clusters` for visualization

```
answer = """
What is the optimal number of clusters?
"""
send(answer, 41)

answer = """
Can you interpret the obtained clusters?
"""
send(answer, 42)

#YOUR VISUALIZATION CODE HERE
send(plt, 43)
```

Student 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. Below, you will find functions for computing three popular fairness metrics:

- demographic parity
- equalized odds
- predictive value parity

```python
# 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/"
```

Fairness Definition 1: 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. In other words, the probability of a positive outcome (denoted as PPP) should be the same independent of the value of the protected attribute.

We first write a function `compute_ppp` that calculates the PPP for a given population.

```python
# For demographic parity, we compare the difference between the PPPs
of the sensitive attributes.
def compute_ppp(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
```

Fairness Definition 2: Equalized Odds

Our second definition of fairness is called **equalized odds**. This definition requires that the true positive rates (TPR) as well as the false positive rates (FPR) are equal accross values of the sensitive attribute. 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]
```

Fairness Definition 3: Predictive Value Parity

Predictive value-parity equalizes the probability of a positive outcome, given a positive prediction (PPV) and the probability of a negative outcome given a negative prediction (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]
```

2 - Fairness Evaluation Example

We will evaluate fairness of a model predicting whether a student will pass or fail a flipped classroom course. To this end, we use the same flipped classroom data set as in the previous lectures. We will first load the data set.

```
# 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]
```

```
# 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']]
```

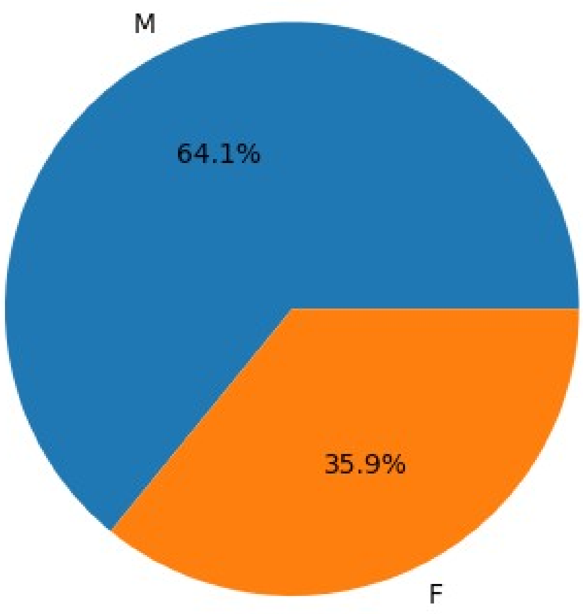
```python
# Load and process ground truth grades, which are between 0 to 6
# Recieving a score 4 or higher is passing, so we can convert these
grades to a binary pass/fail variable {0, 1}
y = [1 if grade >= 4 else 0 for grade in demographics['grade']]

demographics.insert(0, 'y', y)
demographics.insert(1, 'y_pred', y_pred)

display(demographics)
```

```
     y  y_pred  index gender country_diploma continent_diploma
year_diploma  \
0    0       0      0      M          France            Europe
2018.0
1    0       0      1      M          France            Europe
2018.0
2    1       1      2    NaN             NaN               NaN
NaN
3    1       1      3      M          France            Europe
2018.0
4    1       1      4      M          France            Europe
2018.0
..  ..     ...    ...    ...             ...               ...
...
209  0       0    105      M          France            Europe
2018.0
210  0       0    106      M          Suisse            Europe
2019.0
211  1       1    107      M          Suisse            Europe
2018.0
212  1       1    108      M          France            Europe
2018.0
213  1       1    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 16 columns]
```

We first start by analyzing, whether our data set is imbalanced with respect to protected attributes. In the following, we will always focus on gender (the analysis could be conducted in exactly the same way for other protected attributes.

```
val_counts = 
demographics.gender.value_counts()/np.sum(demographics.gender.value_co
unts())
labels = val_counts.index.to_list()
plt.pie(val_counts, labels = labels,autopct='%1.1f%%')
plt.show()
```



We observe that the data set is imbalanced with only 35.9% of the students identifying as female.

Next, we also look at the prevalence, i.e. the proportion of positive cases to overall cases.

```
prev = demographics['y'].mean()
print(prev)
```

```
0.6261682242990654
```

```
prev_gender = demographics.groupby('gender')['y'].mean()
print(prev_gender)
```

```
gender
F    0.657143
M    0.568000
Name: y, dtype: float64
```

We observe that the pevalence is higher for female students.

3 - Your Task

Evaluate fairness of the model by computing one of the fairness metrics discussed in class. Send us the following:

- Why did you choose this metric? Why do you think it is appropriate for the given use case?
- Is the classifier fair with respect to your selected metric? If not, what consequences might this have?

```
# YOUR TURN: FILL IN CODE HERE

# Get PPP for males (in case of demographic parity)
ppp_m = compute_ppp(demographics[demographics['gender'] == 'M'])

# Get PPP for females (in case of demographic parity)
ppp_f = ''

# Print values
print(ppp_m, ppp_f)
```

```
import requests

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

npt_config = {
    'session_name': 'lecture-12',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}
```

```
# YOUR TURN: Why did you choose this metric?
# Why do you think it is appropriate for the given use case?

argument = ''
send(argument, 1)
```

```
# YOUR TURN: Discuss your results. Is the classifier fair with respect to your selected metric?
# If not, what consequences might this have?

interpretation = ''
send(interpretation, 2)
```

Student Notebook - Lecture 13

This notebook provides an introduction to explaining the predictions of your neural network model. Building upon last week's fairness lecture, this lecture on explainability is especially relevant to the ethical concerns of modeling human data. Explainable AI aims to answer the question: why did my black box model make prediction y for features x?

To do this, we look at two different classes of AI explainability: global surrogate models (estimating the whole black box) and local surrogate models (explaining one instance's prediction). In this notebook, we will investigate using **LIME** to explain neural network models.

The material for this notebook is inspired by a great book on Interpretable Machine Learning by Christopher Molnar.

Note that this notebook will need to be run on a kernel with Tensorflow and explainability packages installed. To run the notebook, choose the kernel `Tensorflow` on the top right of Noto.

Missing files? Make sure that you have copied all the (private, anonymized) data and models from the explainability folder of the MLBD Lecture Drive that we shared with you.

```
# Load standard imports for the rest of the notebook.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

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

# Load explainability imports.
from lime import lime_tabular
import os

# 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')

import requests

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

npt_config = {
    'session_name': 'lecture-13',
    'session_owner': 'mlbd',
```

```
    'sender_name': input("Your name: "),
}
```

```
Your name:  Yessir
```

Data Preprocessing

We begin by loading the model for predictions, as well as features and labels in the right formats for our model. This model predicts overall pass / fail performance for students in an EPFL MOOC.

The input to our model involves features regarding student behavior on a learning platform over 10 weeks. We have seen these features before in lecture 8, when we were using deep knowledge tracing to make predictions on data. The model output is a probability of pass/fail, where 0 is pass and 1 is fail. In the predict functions (predict_fn) for the explainability methods, we flip the model performance, so 1 is pass and 0 is fail.

Our model is a bidirectional LSTM with an accuracy of 97% and a balanced accuracy of 94%. We have 8769 users with 25 features each over 10 weeks.

```
model_name = "{}/explainability/model".format(DATA_DIR)
loaded_model = tf.keras.models.load_model(model_name)
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Input In [4], in <cell line: 2>()
      1 model_name = "{}/explainability/model".format(DATA_DIR)
----> 2 loaded_model = tf.keras.models.load_model(model_name)

File /opt/tensorflow/lib/python3.8/site-packages/keras/utils/traceback_utils.py:67, in filter_traceback.<locals>.error_handler(*args, **kwargs)
     65 except Exception as e:  # pylint: disable=broad-except
     66   filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67   raise e.with_traceback(filtered_tb) from None
     68 finally:
     69   del filtered_tb

File /opt/tensorflow/lib/python3.8/site-packages/keras/saving/save.py:204, in load_model(filepath, custom_objects, compile, options)
    202 if isinstance(filepath_str, str):
    203   if not tf.io.gfile.exists(filepath_str):
--> 204     raise IOError(f'No file or directory found at {filepath_str}')
    206   if tf.io.gfile.isdir(filepath_str):
    207     return saved_model_load.load(filepath_str, compile, options)
```

```
OSError: No file or directory found at
./../../data//explainability/model
```

```
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
```

Your Turn: Local Interpretable Model Explanations (LIME)

LIME gives us scores for the most important features for each prediction. We can examine these scores and derive which features of X were important for a particular prediction y.

Interpreting the LIME Plot: LIME explanations help us deduce which features were important in the model making this prediction for this specific student, and how much each feature contributed positively or negatively towards the ultimate prediction (scores on the y-axis). The colors indicate how much a feature contributed towards the model prediction in terms of failing (red) or passing (green). The descriptions of the feature names mentioned in recent papers from the lab (1, 2) are below.

Set	Feature	Description
Regularity	DelayLecture	The average delay in viewing video lectures after they are released to students.
	RegPeakTimeDayHour	The extent to which students' activities are centered around a particular hour of the day.
	RegPeriodicityDayHour	The extent to which the hourly pattern of user's activities repeats over days.
Engagement	NumberOfSessions	The number of unique online sessions the student has participated in.
	RatioClicksWeekendDay	The ratio between the number of clicks in the weekend and the weekdays
	AvgTimeSessions	The average of the student's time per session.
	TotalTimeSessions	The sum of the student's time in sessions.
	StdTimeSessions	The standard deviation of student's time in sessions.
	StdTimeBetweenSessions	The standard deviation of the time between sessions of each user.
	TotalClicks	The number of clicks that a student has made overall.
	TotalClicksProblem	The number of clicks that a student has made on problems this week.
	TotalClicksVideo	The number of clicks that a student has made on videos this week.
	TotalClicksWeekday	The number of clicks that a student has made on the weekdays.
	TotalClicksWeekend	The number of clicks that a student has made on the weekends.
	TotalTimeProblem	The total (cumulative) time that a student has spent on problem events.
	TotalTimeVideo	The total (cumulative) time that a student has spent on video events.
Control	TotalClicksVi	The number of times a student loaded a video.

Set	Feature	Description
	deoLoad	
	TotalClicksVideo	The number of times a student clicked on a video (load, pause, play, forward).
	AvgWatchedWeeklyProp	The ratio of videos watched over the number of videos available.
	StdWatchedWeeklyProp	The standard deviation of videos watched over the number of videos available.
	AvgReplayedWeeklyProp	The ratio of videos replayed over the number of videos available.
	StdReplayedWeeklyProp	The standard deviation of videos replayed over the number of videos available.
	AvgInterruptedWeeklyProp	The ratio of videos interrupted over the number of videos available.
	StdInterruptedWeeklyProp	The standard deviation of videos interrupted over the number of videos available.
	FrequencyEventVideo	The frequency between every Video action and the following action.
	FrequencyEventLoad	The frequency between every Video.Load action and the following action.
	FrequencyEventPlay	The frequency between every Video.Play action and the following action.
	FrequencyEventPause	The frequency between every Video.Pause action and the following action.
	FrequencyEventStop	The frequency between every Video.Stop action and the following action.
	FrequencyEventSeekBackward	The frequency between every Video.SeekBackward action and the following action.
	FrequencyEventSeekForward	The frequency between every Video.SeekForward action and the following action.
	FrequencyEventSpeedChange	The frequency between every Video.SpeedChange action and the following action.
	AvgSeekLength	The student's average seek length (seconds).
	StdSeekLength	The student's standard deviation for seek length (seconds).

Set	Feature	Description
	AvgPauseDuration	The student's average pause duration (seconds).
	StdPauseDuration	The student's standard deviation for pause duration (seconds).
	AvgTimeSpeedingUp	The student's average time using Video.SeekForward actions (seconds).
	StdTimeSpeedingUp	The student's standard deviation of time using Video.SeekForward actions (seconds).
Participation	CompetencyStrength	The extent to which a student passes a quiz getting the maximum grade with few attempts.
	Competency Alignment	The number of problems this week that the student has passed.
	Competency Anticipation	The extent to which the student approaches a quiz provided in subsequent weeks.
	ContentAlignment	The number of videos this week that have been watched by the student.
	ContentAnticipation	The number of videos covered by the student from those that are in subsequent weeks.
	StudentSpeed	The average time passed between two consecutive attempts for the same quiz.
	StudentShape	The extent to which the student receives the maximum quiz grade on the first attempt.

```
# 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)

# Here is a plotting utility for the LIME results.
def plot_lime(exp):
    s = 'fail' if labels[instance] else 'pass'
```

```python
    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]).reshape(1,250))
[0][0]
    prediction = np.round(1-prediction, 2)
    print("Student #: ", instance)
    print("Ground Truth Model Prediction: ", 1-labels[instance], "-",
s)
    print("Black Box Model Prediction: ", prediction, "-", 'pass' if
prediction > 0.5 else 'fail')

# YOUR TURN: Choose a student to explain (by index #). Note that there
are 8,769 students.
instance = ...

# We call the explainer on a student instance.
exp = explainer.explain_instance(features.iloc[instance], predict_fn,
num_features=10)

# YOUR TURN: Plot the LIME results
plot_lime(...)

send(plt, 1)

lime_explanation = """

Write your interpretation here

"""

send(lime_explanation, 2)
```