In this notebook

Many college courses conclude by giving students the opportunity to evaluate the course and the instructor anonymously. In the article “Beauty in the Classroom: Professors’ Pulchritude and Putative Pedagogical Productivity” (PDF), authors Daniel Hamermesh and Amy M. Parker suggest (based on a data set of teaching evaluation scores collected at UT Austin) that student evaluation scores can partially be predicted by features unrelated to teaching, such as the physical attractiveness of the instructor.

In this notebook, we will use this data to try and predict a course- and instructor-specific “baseline” score (excluding the effect of teaching quality), against which to measure instructor performance.

Attribution

Parts of this lab are based on a lab assignment from the OpenIntro textbook “Introductory Statistics with Randomization and Simulation” that is released under a Creative Commons Attribution-ShareAlike 3.0 Unported license. The book website is at https://www.openintro.org/book/isrs/.

Data

The data were gathered from end of semester student evaluations for a large sample of professors from the University of Texas at Austin. In addition, six students looked at a photograph of each professor in the sample, and rated the professors’ physical appearance. More specifically:

> Each of the professors’ pictures was rated by each of six undergraduate students: Three women and three men, with one of each gender being a lower-division, two upper-division students (to accord with the distribution of classes across the two levels). The raters were told to use a 10 (highest) to 1 rating scale, to concentrate on the physiognomy of the professor in the picture, to make their ratings independent of age, and to keep 5 in mind as an average.

We are using a slightly modified version of the original data set from the published paper. The dataset was released along with the textbook “Data Analysis Using Regression and Multilevel/Hierarchical Models” (Gelman and Hill, 2007).)

Setup

We will start by importing relevant libraries, setting up our notebook, reading in the data, and checking that it was loaded correctly.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn import metrics
from sklearn import model_selection
from sklearn.linear_model import LinearRegression

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

```
!wget 'https://www.openintro.org/stat/data/evals.csv' -O 'evals.csv'
```

```
--2022-06-02 11:50:08--  https://www.openintro.org/stat/data/evals.csv
Resolving www.openintro.org (www.openintro.org)... 192.185.65.127
Connecting to www.openintro.org (www.openintro.org)|192.185.65.127|:443... connected.
HTTP request sent, awaiting response... 200 OK
```

```
Length: 55004 (54K) [text/csv]
Saving to: ‘evals.’csv

evals.csv           100%[===================>]  53.71K  --.-KB/s    in 0.09s

2022-06-02 11:50:08 (578 KB/s) - ‘evals.’csv saved [55004/55004]
```

```
df = pd.read_csv('evals.csv')
df.head()
df.columns
df.shape
```

```
   score          rank     ethnicity  gender language  age  cls_perc_eval  \
0    4.7  tenure track      minority  female  english   36       55.81395
1    4.1  tenure track      minority  female  english   36       68.80000
2    3.9  tenure track      minority  female  english   36       60.80000
3    4.8  tenure track      minority  female  english   36       62.60163
4    4.6       tenured  not minority    male  english   59       85.00000

   cls_did_eval  cls_students cls_level  ...   cls_credits bty_f1lower  \
0            24            43     upper  ...  multi credit           5
1            86           125     upper  ...  multi credit           5
2            76           125     upper  ...  multi credit           5
3            77           123     upper  ...  multi credit           5
4            17            20     upper  ...  multi credit           4

   bty_f1upper  bty_f2upper  bty_m1lower  bty_m1upper  bty_m2upper  bty_avg  \
0            7            6            2            4            6      5.0
1            7            6            2            4            6      5.0
2            7            6            2            4            6      5.0
3            7            6            2            4            6      5.0
4            4            2            2            3            3      3.0

   pic_outfit pic_color
0  not formal     color
1  not formal     color
2  not formal     color
3  not formal     color
4  not formal     color

[5 rows x 21 columns]
```

```
Index(['score', 'rank', 'ethnicity', 'gender', 'language', 'age',
       'cls_perc_eval', 'cls_did_eval', 'cls_students', 'cls_level',
       'cls_profs', 'cls_credits', 'bty_f1lower', 'bty_f1upper', 'bty_f2upper',
       'bty_m1lower', 'bty_m1upper', 'bty_m2upper', 'bty_avg', 'pic_outfit',
       'pic_color'],
      dtype='object')
```

```
(463, 21)
```

Each row in the data frame represents a different course, and columns represent features of the courses and professors. Here’s the data dictionary:

variable	description
score	average professor evaluation score: (1) very unsatisfactory - (5) excellent.
rank	rank of professor: teaching, tenure track, tenured.
ethnicity	ethnicity of professor: not minority, minority.
gender	gender of professor: female, male.
language	language of school where professor received education: english or non-english.
age	age of professor.
cls_perc_eval	percent of students in class who completed evaluation.
cls_did_eval	number of students in class who completed evaluation.
cls_students	total number of students in class.
cls_level	class level: lower, upper.
cls_profs	number of professors teaching sections in course in sample: single, multiple.
cls_credits	number of credits of class: one credit (lab, PE, etc.), multi credit.
bty_f1lower	beauty rating of professor from lower level female: (1) lowest - (10) highest.
bty_f1upper	beauty rating of professor from upper level female: (1) lowest - (10) highest.
bty_f2upper	beauty rating of professor from second upper level female: (1) lowest - (10) highest.
bty_m1lower	beauty rating of professor from lower level male: (1) lowest - (10) highest.
bty_m1upper	beauty rating of professor from upper level male: (1) lowest - (10) highest.
bty_m2upper	beauty rating of professor from second upper level male: (1) lowest - (10) highest.
bty_avg	average beauty rating of professor.
pic_outfit	outfit of professor in picture: not formal, formal.
pic_color	color of professor's picture: color, black & white.

Source: OpenIntro book.

Note that:

- `score` is the target variable - this is what we want our model to predict. We expect that the score is a function of the teaching quality, characteristics of the course, and non-teaching related characteristics of the instructor. However, the "true" teaching quality for each course is not known.
- the variables that begin with a `cls_` prefix are features that relate to the course. These features could potentially affect student evaluations: for example, students may rank one-credit lab courses more highly than multi-credit lecture courses.
- variables such as `rank`, `ethnicity`, `gender`, `language`, `age`, and the variables with a `bty_` prefix are features that relate to the instructor. They do not necessarily to the quality of instruction! These features may also affect student evaluations: for example, students may rate instructors more highly if they are physically attractive.
- variables with the `pic_` prefix describe the photograph that was shown to the students who provided the `bty_` scores. This should have no effect on the student evaluations, since those were evaluations by students who were enrolled in the course (not the students who were shown the photograph and asked to provide an attractiveness score.) (For your reference: on the bottom of page 7 of the paper, the authors describe why they include this variable and how they used it)

Explore data

As always, start by exploring the data:

```
df.describe()
```

```
            score         age  cls_perc_eval  cls_did_eval  cls_students  \
count  463.000000  463.000000     463.000000    463.000000    463.000000
mean     4.174730   48.365011      74.427788     36.624190     55.177106
```

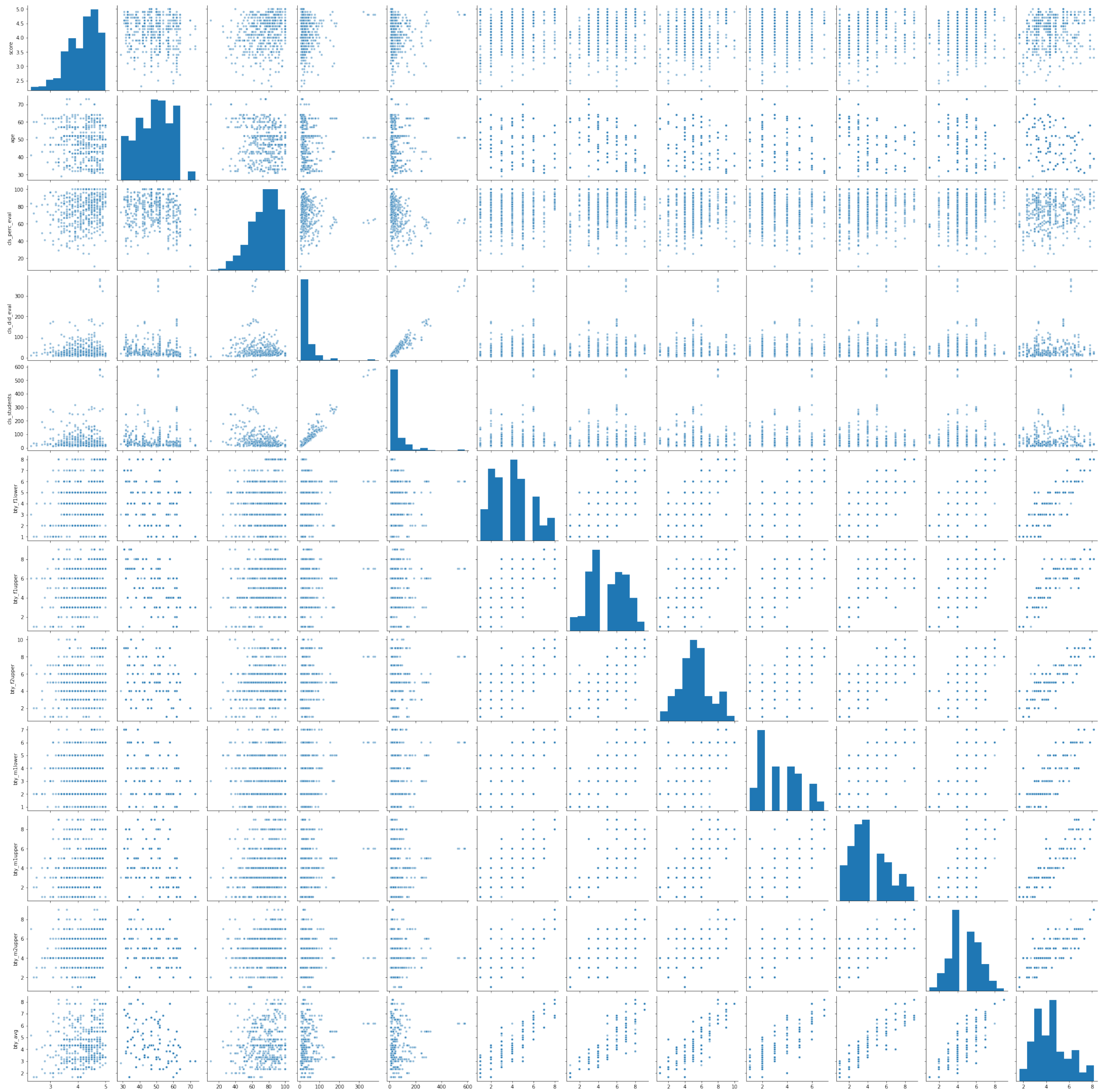
```
std      0.543865   9.802742    16.756311    45.018481    75.072800
min      2.300000  29.000000    10.416670     5.000000     8.000000
25%      3.800000  42.000000    62.696165    15.000000    19.000000
50%      4.300000  48.000000    76.923080    23.000000    29.000000
75%      4.600000  57.000000    87.249170    40.000000    60.000000
max      5.000000  73.000000   100.000000   380.000000   581.000000

       bty_f1lower  bty_f1upper  bty_f2upper  bty_m1lower  bty_m1upper  \
count   463.000000   463.000000   463.000000   463.000000   463.000000
mean      3.963283     5.019438     5.213823     3.412527     4.146868
std       1.873936     1.934437     2.018224     1.637102     2.110586
min       1.000000     1.000000     1.000000     1.000000     1.000000
25%       2.000000     4.000000     4.000000     2.000000     3.000000
50%       4.000000     5.000000     5.000000     3.000000     4.000000
75%       5.000000     7.000000     6.000000     5.000000     5.000000
max       8.000000     9.000000    10.000000     7.000000     9.000000

       bty_m2upper     bty_avg
count   463.000000  463.000000
mean      4.751620    4.417844
std       1.575266    1.527380
min       1.000000    1.667000
25%       4.000000    3.167000
50%       5.000000    4.333000
75%       6.000000    5.500000
max       9.000000    8.167000
```

```python
sns.pairplot(df, plot_kws={'alpha':0.5, 'size': 0.1})
```

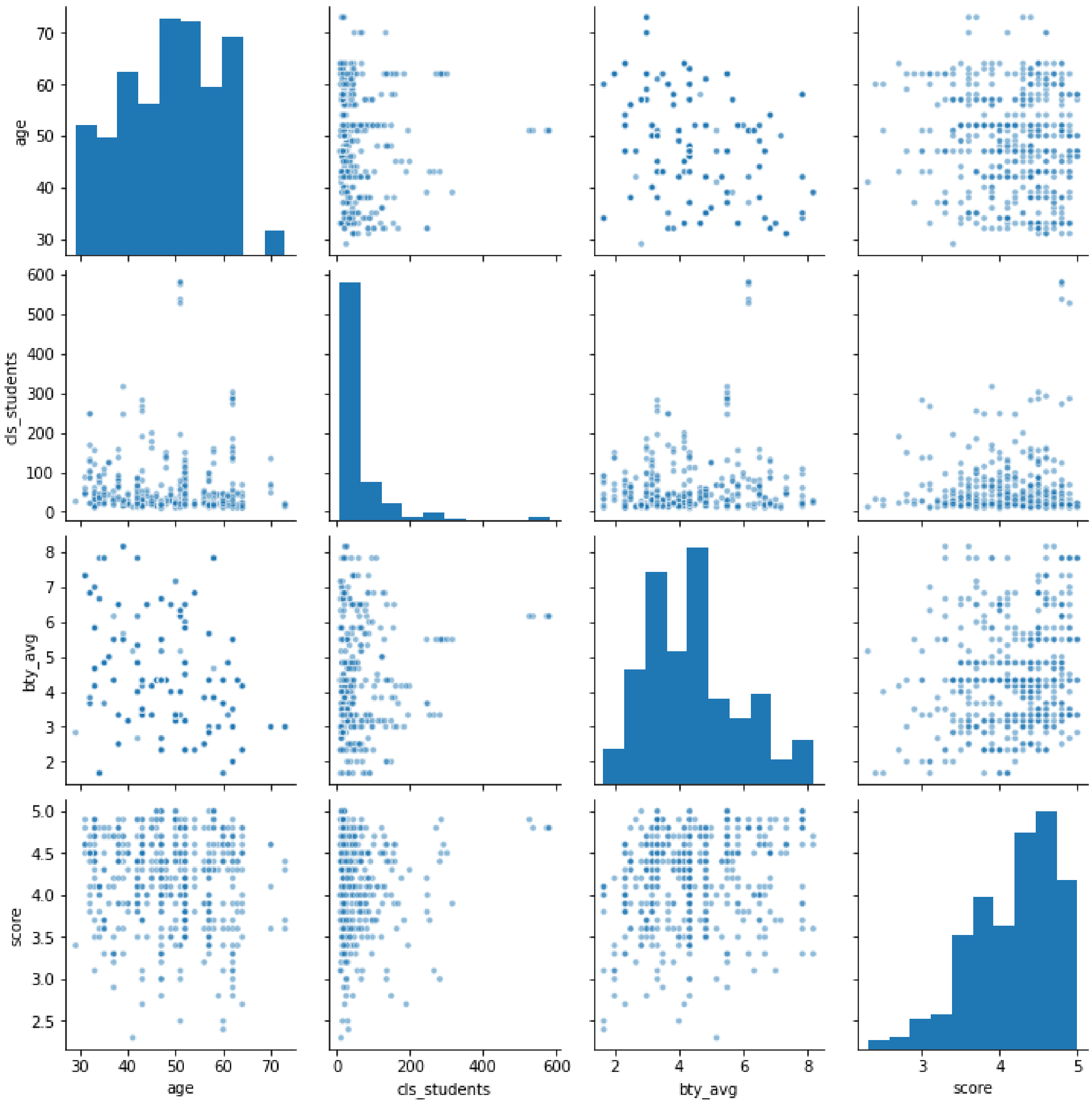
```
<seaborn.axisgrid.PairGrid at 0x7feaa0f60c70>
```



With so many numeric variables, the pair plot is hard to read. We can create a pairplot excluding some variables that we don't expect to be useful for visualization: `cls_perc_eval`, `cls_did_eval`. We will also exclude the individual attractiveness ratings `bty_f1lower`, `bty_f1upper`, `bty_f2upper`, `bty_m1lower`, `bty_m1upper`, `bty_m2upper`, since the overall attractiveness rating is still represented by `bty_avg`.

```
sns.pairplot(df, vars=['age', 'cls_students', 'bty_avg', 'score'], plot_kws={'alpha':0.5,
    'size': 0.1})
```

```
<seaborn.axisgrid.PairGrid at 0x7fea3dcd4460>
```



As part of our exploration of the data, we can also examine the effect of non-numeric variables related to the instructor and the class: `rank`, `ethnicity`, `gender`, `language`, `cls_level`, `cls_profs`, `cls_credits`.

```
for feature in ['rank', 'ethnicity', 'gender', 'language', 'cls_level', 'cls_profs',
    'cls_credits']:
    df.groupby([feature])['score'].describe()
```

```
              count      mean       std  min  25%   50%  75%  max
rank
teaching      102.0  4.284314  0.498263  3.3  3.9  4.40  4.7  5.0
tenure track  108.0  4.154630  0.561104  2.3  3.7  4.35  4.6  4.9
tenured       253.0  4.139130  0.550262  2.4  3.8  4.20  4.6  5.0
```

```
              count      mean       std  min    25%   50%    75%  max
```

```
ethnicity
minority       64.0  4.071875  0.581588  2.7  3.675  4.05  4.525  5.0
not minority  399.0  4.191228  0.536505  2.3  3.850  4.30  4.600  5.0
```

```
        count      mean       std  min  25%  50%  75%  max
gender
female  195.0  4.092821  0.563814  2.3  3.7  4.1  4.5  5.0
male    268.0  4.234328  0.521896  2.4  3.9  4.3  4.6  5.0
```

```
             count      mean       std  min  25%   50%  75%  max
language
english      435.0  4.189655  0.547183  2.3  3.9  4.30  4.6  5.0
non-english   28.0  3.942857  0.434979  3.4  3.6  3.75  4.4  4.8
```

```
           count      mean       std  min  25%  50%  75%  max
cls_level
lower      157.0  4.238217  0.592532  2.5  3.8  4.4  4.7  5.0
upper      306.0  4.142157  0.515104  2.3  3.8  4.2  4.5  5.0
```

```
           count      mean       std  min  25%  50%  75%  max
cls_profs
multiple   306.0  4.184641  0.551177  2.4  3.8  4.3  4.6  5.0
single     157.0  4.155414  0.530529  2.3  3.8  4.3  4.6  5.0
```

```
              count      mean       std  min  25%  50%  75%  max
cls_credits
multi credit  436.0  4.147018  0.542464  2.3  3.8  4.2  4.6  5.0
one credit     27.0  4.622222  0.334357  3.5  4.5  4.7  4.9  5.0
```

Discussion Question 1 Describe the relationship between `score` and the overall attractiveness rating `bty_avg`. Is there an apparent correlation? If so, is it a positive or a negative correlation? What about `age` and `cls_students`, do they appear to be correlated with `score`?

Also describe the relationship between `score` and the categorical variables you explored above that are related to characteristics of the *instructor*: `rank`, `ethnicity`, `gender`, `language`. Which of these variables have an apparent correlation with `score`? Is it a positive or a negative correlation?

Are any of the apparent relationships you observed unexpected to you? Explain.

Encoding categorical variables

To represent a categorical variable (with no inherent ordering) in a regression, we can use *one hot encoding*. It works as follows:

- For a categorical variable x with values $1, \cdots, M$
- Represent with M binary features: $\phi_1, \phi_2, \cdots, \phi_M$
- Model in regression $w1_1\phi_1 + \cdots + w_M\phi_M$

We can use the `get_dummies` function in `pandas` for one hot encoding. Create a copy of the dataframe with all categorical variables transformed into indicator ("dummy") variables, and save it in a new data frame called `df_enc`.

Compare the columns of the `df` data frame versus the `df_enc` data frame.

```python
df_enc = pd.get_dummies(df)
df_enc.columns
```

```
Index(['score', 'age', 'cls_perc_eval', 'cls_did_eval', 'cls_students',
       'bty_f1lower', 'bty_f1upper', 'bty_f2upper', 'bty_m1lower',
       'bty_m1upper', 'bty_m2upper', 'bty_avg', 'rank_teaching',
       'rank_tenure track', 'rank_tenured', 'ethnicity_minority',
       'ethnicity_not minority', 'gender_female', 'gender_male',
       'language_english', 'language_non-english', 'cls_level_lower',
       'cls_level_upper', 'cls_profs_multiple', 'cls_profs_single',
       'cls_credits_multi credit', 'cls_credits_one credit',
       'pic_outfit_formal', 'pic_outfit_not formal', 'pic_color_black&white',
       'pic_color_color'],
      dtype='object')
```

Split data

Next, we split the encoded data into a training set (70%) and test set (30%). We will be especially interested in evaluating the model performance on the test set. Since it was not used to train the model parameters (intercept and coefficients), the performance on this data gives us a better idea of how the model may perform on new data.

We'll use the `train_test_split` method in `sklearn`'s `model_selection` module. Since it randomly splits the data, we'll pass a random "state" into the function that makes the split repeatable (same split every time we run this notebook) and ensures that everyone in the class will have exactly the same split.

```python
train, test = model_selection.train_test_split(df_enc, test_size=0.3, random_state=9)
# why 9? see https://dilbert.com/strip/2001-10-25
train.shape
test.shape
```

```
(324, 31)
```

```
(139, 31)
```

Simple linear regression

Now we are finally ready to train a regression model.

Since the article is nominally abou the attractiveness of the instructor, we will train the simple linear regression on the `bty_avg` feature.

In the cell that follows, write code to

- use `sklearn` to fit a simple linear regression model on the training set, using `bty_avg` as the feature on which to train. Save your fitted model in a variable `reg_simple`.
- print the intercept and coefficient of the model.
- use `predict` on the fitted model to estimate the evaluation score on the training set, and save this array in `y_pred_train`.
- use `predict` on the fitted model to estimate the evaluation score on the test set, and save this array in `y_pred_test`.

Then run the cell after that one, which will show you the training data, the test data, and your regression line.

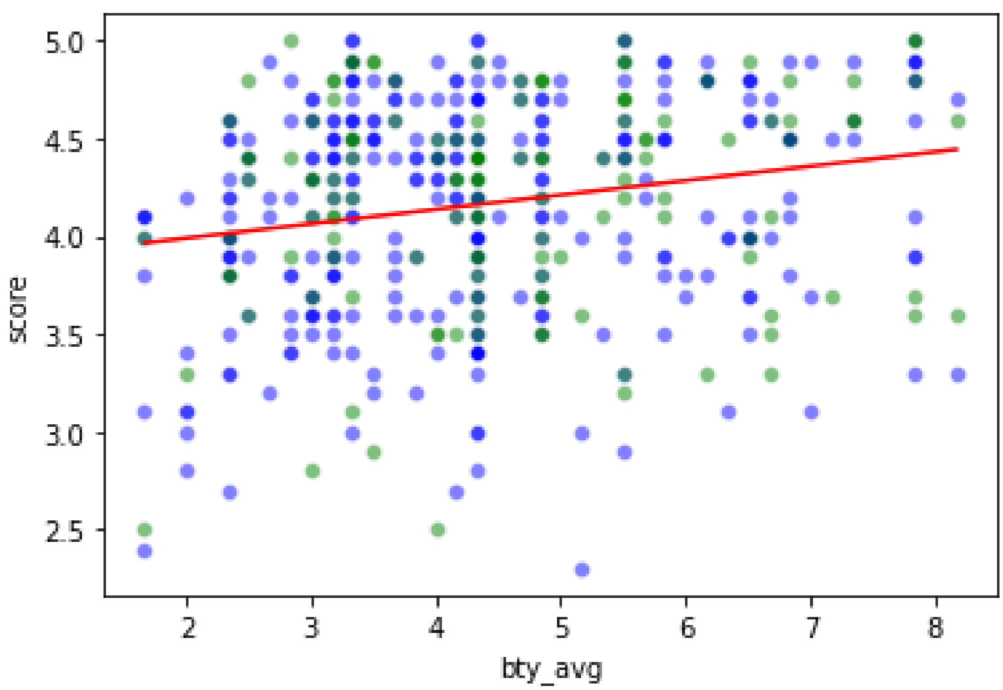
```
reg_simple = LinearRegression().fit(train[['bty_avg']], train['score'])
reg_simple.coef_
reg_simple.intercept_

y_pred_train = reg_simple.predict(train[['bty_avg']])
y_pred_test = reg_simple.predict(test[['bty_avg']])
```

```
array([0.07367795])
```

```
3.842544653270749
```

```
sns.scatterplot(data=train, x="bty_avg", y="score", color='blue', alpha=0.5);
sns.scatterplot(data=test, x="bty_avg", y="score", color='green', alpha=0.5);
sns.lineplot(data=train, x="bty_avg", y=y_pred_train, color='red');
```



Evaluate simple linear regression performance

Next, we will evaluate our model performance.

In the following cell, write a *function* to compute key performance metrics for your model:

- compute the R2 score on your training data
- compute the MSE on your training data
- compute the MSE, divided by the sample variance of `score`, on your training data. Recall that this metric tells us the ratio of average error of your model to average error of prediction by mean.
- and compute the same three metrics for your test set

```
def regression_performance(y_true_train, y_pred_train, y_true_test, y_pred_test):

    r2_train = metrics.r2_score(y_true_train, y_pred_train)
```

```
    mse_train = metrics.mean_squared_error(y_true_train, y_pred_train)
    norm_mse_train = metrics.mean_squared_error(y_true_train,
        y_pred_train)/(np.std(y_true_train)**2)

    r2_test = metrics.r2_score(y_true_test, y_pred_test)
    mse_test = metrics.mean_squared_error(y_true_test, y_pred_test)
    norm_mse_test = metrics.mean_squared_error(y_true_test,
        y_pred_test)/(np.std(y_true_test)**2)

    #print("Training:   %f %f %f" % (r2_train, mse_train, norm_mse_train))
    #print("Test:       %f %f %f" % (r2_test, mse_test, norm_mse_test))

    return [r2_train, mse_train, norm_mse_train, r2_test, mse_test, norm_mse_test]
```

Call your function to print the performance of the simple linear regression. Is a simple linear regression on `bty_avg` better than a "dumb" model that predicts the mean value of `score` for all samples?

```
vals = regression_performance(train['score'], y_pred_train, test['score'], y_pred_test)
```

Multiple linear regression

Next, we'll see if we can improve model performance using multiple linear regression, with more features included.

To start, we need to decide which features to use as input to our model. One possible approach is to use every feature in the dataset excluding the target variable, `score`.

You can build and view this list of features by running:

```
features = df_enc.columns.drop(['score'])
features
```

```
Index(['age', 'cls_perc_eval', 'cls_did_eval', 'cls_students', 'bty_f1lower',
       'bty_f1upper', 'bty_f2upper', 'bty_m1lower', 'bty_m1upper',
       'bty_m2upper', 'bty_avg', 'rank_teaching', 'rank_tenure track',
       'rank_tenured', 'ethnicity_minority', 'ethnicity_not minority',
       'gender_female', 'gender_male', 'language_english',
       'language_non-english', 'cls_level_lower', 'cls_level_upper',
       'cls_profs_multiple', 'cls_profs_single', 'cls_credits_multi credit',
       'cls_credits_one credit', 'pic_outfit_formal', 'pic_outfit_not formal',
       'pic_color_black&white', 'pic_color_color'],
      dtype='object')
```

In the following cell, write code to

- use `sklearn` to fit a linear regression model on the training set, using the `features` array as the list of features to train on. Save your fitted model in a variable `reg_multi`.
- print a table of the features used in the regression and the coefficient assigned to each. If you have saved your fitted regression in a variable named `reg_multi`, you can create and print this table with:

```
df_coef = pd.DataFrame(data =
                        {'feature': features,
                         'coefficient': reg_multi.coef_})
df_coef
```

```
reg_multi = LinearRegression().fit(train[features], train['score'])
df_coef = pd.DataFrame(data =
                        {'feature': features,
                         'coefficient': reg_multi.coef_})
df_coef
```

```
                     feature  coefficient
0                        age    -0.009493
1              cls_perc_eval     0.004385
2               cls_did_eval     0.002983
3               cls_students    -0.001427
4                bty_f1lower     7.022264
5                bty_f1upper     7.052930
6                bty_f2upper     7.032261
7                bty_m1lower     6.959419
8                bty_m1upper     6.999172
9                bty_m2upper     6.978719
10                   bty_avg   -42.015575
11             rank_teaching     0.084572
12         rank_tenure track    -0.085993
13              rank_tenured     0.001421
14        ethnicity_minority    -0.131245
15    ethnicity_not minority     0.131245
16             gender_female    -0.121054
17               gender_male     0.121054
18          language_english     0.085742
19      language_non-english    -0.085742
20           cls_level_lower    -0.006088
21           cls_level_upper     0.006088
22        cls_profs_multiple     0.004518
23          cls_profs_single    -0.004518
24  cls_credits_multi credit    -0.283913
25    cls_credits_one credit     0.283913
26         pic_outfit_formal     0.039920
27     pic_outfit_not formal    -0.039920
28     pic_color_black&white     0.065689
29           pic_color_color    -0.065689
```

Discussion Question 2 Look at the list of features and coefficients, especially those related to the attractiveness ratings.

Are these results surprising, based on the results of the simple linear regression? Explain your answer.

Effect of collinearity

Note especially the coefficients associated with each of the individual attractiveness rankings, and the coefficient associated with the average attractiveness ranking. Each of these features separately seems to have a large effect; however, because they are strongly *collinear*, they cancel one another out.

(You should be able to see the collinearity clearly in the pairplot you created.)

In the following cell, write code to

- create a new `features` array, that drops the *individual* attractiveness rankings in addition to the `score` variable (but do *not* drop the average attractiveness ranking)
- use `sklearn` to fit a linear regression model on the training set, using the new `features` array as the list of features to train on. Save your fitted model in a variable `reg_avgbty`.
- print a table of the features used in the regression and the coefficient assigned to each.

```
features = df_enc.columns.drop(['score',
    'bty_f1lower', 'bty_f1upper', 'bty_f2upper',
    'bty_m1lower', 'bty_m1upper', 'bty_m2upper'])
reg_avgbty = LinearRegression().fit(train[features], train['score'])

df_coef = pd.DataFrame(data =
                        {'feature': features,
                         'coefficient': reg_avgbty.coef_})
df_coef
```

```
                     feature  coefficient
0                        age    -0.009297
1              cls_perc_eval     0.004900
2               cls_did_eval     0.003737
3               cls_students    -0.001740
4                    bty_avg     0.040577
5              rank_teaching     0.075283
6          rank_tenure track    -0.059429
7               rank_tenured    -0.015854
8         ethnicity_minority    -0.111830
9     ethnicity_not minority     0.111830
10             gender_female    -0.097201
11               gender_male     0.097201
12          language_english     0.083435
13      language_non-english    -0.083435
14           cls_level_lower    -0.012437
15           cls_level_upper     0.012437
16        cls_profs_multiple     0.009897
17          cls_profs_single    -0.009897
18   cls_credits_multi credit    -0.285089
19     cls_credits_one credit     0.285089
20         pic_outfit_formal     0.053104
21     pic_outfit_not formal    -0.053104
22   pic_color_black&white     0.076615
23           pic_color_color    -0.076615
```

Discussion Question 3 Given the model parameters you have found, which is associated with the strongest effect (on average) on the evaluation score:

- Instructor ethnicity
- Instructor gender

(Note that in general, we cannot use the coefficient to compare the effect of features that have a different range. But both ethnicity and gender are represented by binary one hot-encoded variables.)

Evaluate multiple regression model performance

Evaluate the performance of your `reg_avgbty` model. In the next cell, write code to:

- use the `predict` function on your fitted regression to find $\hat{y}$ for all samples in the *training* set, and save this in an array called `y_pred_train`
- use the `predict` function on your fitted regression to find $\hat{y}$ for all samples in the *test* set, and save this in an array called `y_pred_test`
- call the `regression_performance` function you wrote in a previous cell, and print the performance metrics on the training and test set.

```
y_pred_train = reg_avgbty.predict(train[features])
y_pred_test = reg_avgbty.predict(test[features])

vals = regression_performance(train['score'], y_pred_train, test['score'], y_pred_test)
```

Discussion Question 4 Based on the analysis above, what portion of the variation in instructor teaching evaluation can be explained by the factors unrelated to teaching performance, such as the physical characteristics of the instructor?

Discussion Question 5 Based on the analysis above, is your model better at predicting instructor teaching scores than a "dumb" model that just assigns the mean teaching score to every instructor? Explain.

Discussion Question 6 Suppose you are hired by the ECE department to develop a classifer that will identify high-performing faculty, who will then be awarded prizes for their efforts.

Based on the analysis above, do you think it would be fair to use scores on teaching evaluations as an input to your classifier? Explain your answer.

Exploring unexpected correlation

There are some features that we do *not* expect to be correlated with the instructor's score.

For example, consider the "features" related to the photograph used by the students who rated the instructor's attractiveness.

There is no reason that characteristics of an instructor's photograph - whether it was in black and white or color, how the instructor was dressed in the photograph - should influence the ratings of students in the instructor's class. (These students did not even see the photograph.)

We're going to explore this more… in the next lesson.