Simeon Sukinder (Sprem)

Classmates you worked with today:

- Anthony
- Arjun
- Ryan

Submission instruction: Please create a pdf via File -> Print (or cmd + P on mac), and upload it to Gradescope. No autograder since there are many correct ways to approach this question. You are not required to finish the entire worksheet provided that you were actively engaged during the entire class period. As long as you've collaborated, put in good effort and made reasonable progress during the lab period, you can expect to get full credit. So even if you decide to finish it at home, please submit what you have by the end of 50 minutes to make sure you get credit!

Part A: Are we automating racism? Video by Vox (~25 minutes)

The whole class will watch this video together:

https://www.youtube.com/watch?v=Ok5sKLXqynQ

By the way, the paper discussed around minute 14 is where project C2 comes from.

While you're watching the video, answer this question: What are two new or interesting things you learned from this video?

One interesting thing I learned from the video is that computers can be trained to understand what people focus on by using eye tracking data and recorded eye movements. This allows technology to predict where a person is likely to look. Another interesting thing I learned is that machines inherently aren't racist and that the "racist" outcomes actually stem from bias in training data.

Discussion as a group (5 minutes)

What is one thing you learned or find interesting from your classmates' answers?

Something I found interesting from my classmates' answers is that bias in technology stems from the humans who design it (as it's their choices, assumptions, and data that shape how the technology behaves).

Part B: Linear regression (20 minutes)

Let's revisit the ROUSes dataset.

In [1]:
```
import pandas as pd
import seaborn as sns
```

In [17]:
```
rouses = pd.read_csv('ROUSes.csv')
print(rouses.shape)
rouses.head()
```

```
(29, 4)
```

Out[17]:

	Age	Length	Weight	Temperament
0	9.5	4.1	93.2	Sleepy
1	12.0	4.1	97.7	Moody
2	14.5	4.6	120.3	No-nonsense
3	7.0	3.3	60.4	Moody
4	10.0	3.5	75.3	No-nonsense

Run the following code to drop the column `Temperament` from the table `rouses`.

In [18]:
```
rouses = rouses.drop(columns='Temperament')
```

Here our goal is to predict the `Weight` of ROUSes using numerical values. Start by answering the following questions:

- Classification or regression?
- Predictor variables?
- Target variables?
- Model: is this simiple or multiple linear regression?

This is a regression problem as we are trying to predict weight. Predictor variables include age and length. The target variable is weight. This is multiple linear regression as we are using more than one variable.

Exploratory analysis

Run the code `rouses.corr()`. This calculates the correlations between pairs of numerical columns.

In [19]:
```
rouses.corr()
```

Out[19]:

	Age	Length	Weight
Age	1.000000	0.934265	0.988924
Length	0.934265	1.000000	0.959640
Weight	0.988924	0.959640	1.000000

Based on the correlations, which variable do you expect to be more helpful in predicting `Weight`: `Age` or `Length`?

I would expect age to be more helpful in predicting weight.

Setting up the data

We usually split the dataset into train and test first and then split them into X and y. But for this exercise, let's go in a different order.

Split the table `rouses` into `X` and `y` by completing the following code.

```
y = rouses[...]
X = rouses.drop(columns=...)
```

In [20]:
```
y = rouses['Weight']
X = rouses.drop(columns='Weight')
```

Setting up regression pipeline

We'll write our machine learning code inside a function so that we can call it multiple times later. Copy the following code, then fill in the missing spots marked by `...`.

Hint: The six `...` are from these options: `X_train`, `X_test`, `y_train`, `y_test`.

```
def rouses_lr(X, y, seed=0):
    # Split X and y into X_train, X_test and y_train, y_test
    # using the same seed ensures that the same rows are picked
between X and y

    X_train = X.sample(frac=0.8, random_state=seed)
    X_test = X.drop(index=X_train.index)
    y_train = y.sample(frac=0.8, random_state=seed)
    y_test = y.drop(index=y_train.index)

    # Create "empty" model
    from sklearn.linear_model import LinearRegression

    lr = LinearRegression(fit_intercept=True)

    # Fit model to data (or train model)
    lr.fit(..., ...)

    # Save coefficients of the trained model
    coefs = pd.DataFrame(lr.coef_,
                         index=lr.feature_names_in_,
                        columns=['Coefficient vals'])

    # Save model performance on train and test
    coefs.loc['Train R2 score'] = lr.score(..., ...)
    coefs.loc['Test R2 score'] = lr.score(..., ...)
    return coefs
```

In [27]:
```
def rouses_lr(X, y, seed=0):
    # Split X and y into X_train, X_test and y_train, y_test
    # using the same seed ensures that the same rows are picked between X ar

    X_train = X.sample(frac=0.8, random_state=seed)
    X_test = X.drop(index=X_train.index)

    y_train = y.sample(frac=0.8, random_state=seed)
    y_test = y.drop(index=y_train.index)

    # Create "empty" model
    from sklearn.linear_model import LinearRegression

    lr = LinearRegression(fit_intercept=True)

    # Fit model to data (or train model)
    lr.fit(X_train, y_train)

    # Save coefficients of the trained model
    coefs = pd.DataFrame(lr.coef_,
                         index=lr.feature_names_in_,
                        columns=['Coefficient vals'])

    # Save model performance on train and test
    coefs.loc['Train R2 score'] = lr.score(X_train, y_train)
    coefs.loc['Test R2 score'] = lr.score(X_test, y_test)
    return coefs
```

Normalizing features

In class, we talked about interpreting linear regression coefficients, and mentioned an important caviat: To use the **magnitude of the coefficients** as an indication of relative feature importance, **the features have to be in the same range or scale**. Today, we'll see a few different ways to acheive that goal. The technical term for this is **feature normalization**, if you want to Google it for your project.

First, call the function `describe` on the table `X` to see the summary statistics of the numerical columns.

In [22]:
```
X.describe()
```

Out[22]:

	Age	Length
count	29.000000	29.000000
mean	13.362069	3.841379
std	12.042259	1.551386
min	1.000000	1.100000
25%	5.000000	3.100000
50%	10.000000	3.900000
75%	15.500000	4.600000
max	55.000000	8.000000

Question: Does `Age` and `Length` have the same ranges? Or the same means?

They do not have the same ranges or the same means.

We'll see how that affects the final model and interpretation of coefficients.

1. Standardization

The first method is called **standardization**. This means that each variable will have mean of 0 and standard deviation of 1 after this process. You can acheive this by doing the following steps:

1. Within each column, subtract its mean. For example, if the mean `Age` is 13,

subtract 13 from everyone's age.
2. Within each column, divide by its standard deviation. For example, if the std of `Age` 10, divide everyone's age by 10.

The code to achieve this is very simple. Make sure you understand what's going on, then copy and run the lines below.

```
X_standardized = X - X.mean() # Subtract by column mean
X_standardized = X_standardized/X_standardized.std() # Divide by
column std

X_standardized.describe()
```

In [23]:
```
X_standardized = X - X.mean()
X_standardized = X_standardized/X_standardized.std()
X_standardized.describe()
```

Out[23]:

	Age	Length
count	2.900000e+01	2.900000e+01
mean	-2.201304e-17	3.014830e-17
std	1.000000e+00	1.000000e+00
min	-1.026557e+00	-1.767052e+00
25%	-6.943937e-01	-4.778820e-01
50%	-2.791892e-01	3.778602e-02
75%	1.775357e-01	4.889955e-01
max	3.457651e+00	2.680584e+00

Before we move on, check that each feature in `X_standardized` now has zero mean and 1 stadard deviation as expected.

2. Min-max scaling

The second method is called **min-max scaling**. This makes all values in each column to fall between 0 and 1. The minimum value becomes 0, the maximum value becomes 1, and everything in between is scaled linearly.

Here is the code that does that. As before, make sure you understand what's going on.

```
X_minmax = X - X.min() # this turns minimum value of each column to
```

```
0
ranges = X.max() - X.min() # calculate the range per column
X_minmax = X_minmax/ranges # this turns maximum value to 1

X_minmax.describe()
```

In [24]:
```
X_minmax = X - X.min()
ranges = X.max() - X.min()
X_minmax = X_minmax/ranges
X_minmax.describe()
```

Out[24]:

	Age	Length
count	29.000000	29.000000
mean	0.228927	0.397301
std	0.223005	0.224839
min	0.000000	0.000000
25%	0.074074	0.289855
50%	0.166667	0.405797
75%	0.268519	0.507246
max	1.000000	1.000000

Before we move on, check that each feature in `X_minimax` now has minimum 0 and maximum 1 as expected.

Check the results!

Finally it's time to put everything together. Run the following code to check:

rows:

- the resulting coefficients (`Age` , `Length`) of the linear regression model, and the
- R^2 scores (`Train R2 score` , `Test R2 score`), for

columns: - the three different versions of feature tables (unnormalized `X`, standardized `X_standardized`, min-max scaled `X_minmax`), and - the unnormalized coefficients mulipled by the std of the unnormalized features (`Unnormalized * std`)

In [28]:
```
seed = 2024

results = pd.DataFrame()
```

```
for features, name in zip([X, X_standardized, X_minmax],
                          ["Unnormalized", "Standardized", "Min-max scaled"]):
    res = rouses_lr(features, y, seed)
    res = res.rename(columns ={'Coefficient vals': name})
    results = pd.concat((results, res), axis=1)

results['Unnormalized * std'] = results['Unnormalized']*X.std()
results
```

Out[28]:

	Unnormalized	Standardized	Min-max scaled	Unnormalized * std
Age	3.535627	42.576937	190.923857	42.576937
Length	19.701658	30.564874	135.941442	30.564874
Train R2 score	0.988407	0.988407	0.988407	NaN
Test R2 score	0.961811	0.961811	0.961811	NaN

The last column is sometimes called **beta** or **standardized** coefficients. Does it look similar to any of the other columns?

Question: Which feature is more important, according to the magnitude of the coefficients? Did that answer change for different versions of features?

The last column looks similar to the standardized one. Age is more important according to the magnitude of the coefficients. My answer did not change.

I hope this convinces you to normalize your features for your project if you decide to do a predictive model, and especially if you plan to do linear regression.

FYI, you get the same score regardless of feature normalization here. This is not the case for every dataset and every ML model. Often normalization improves accuracy of the model.