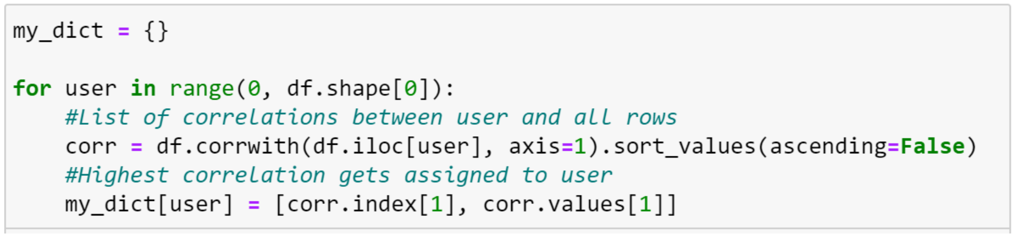
DS-GA 1001 05/12/2021

Data analysis Project 2

Question 1.1) For every user in the given data, find its most correlated user

First, I created a dictionary with a every user as keys and two values corresponding to each key: the highest correlated user and its correlation. The dictionary is filled using a for loop that, for every user, loops over every other user calculating its correlation and storing the maximum correlated users and its corresponding correlation.

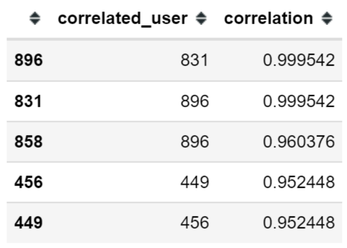
In order to calculate the correlation between two users I used the function corrwith.



```
my_dict = {}

for user in range(0, df.shape[0]):
    #List of correlations between user and all rows
    corr = df.corrwith(df.iloc[user], axis=1).sort_values(ascending=False)
    #Highest correlation gets assigned to user
    my_dict[user] = [corr.index[1], corr.values[1]]
```

Question 1.2: What is the pair of the most correlated users in the data?

To answer this question, I simply accessed the position of the dictionary that contained the highest correlation and extracted the users corresponding to it.



	correlated_user	correlation
896	831	0.999542
831	896	0.999542
858	896	0.960376
456	449	0.952448
449	456	0.952448

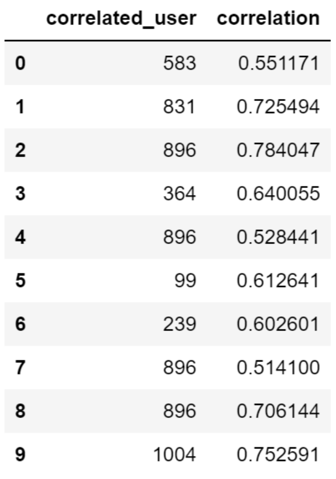
Question 1.3: What is the value of this highest correlation?

Similarly, I extracted the highest correlation following the previously mentioned method.

```
The correlation between the highest correlated users is: 0.9995424261495214
```

Question 1.4: For users 0, 1, 2, **…,** 9, print their most correlated users.

Data is most clearly displayed when stored in a dataframe, so I stored the data in the dictionary into a dataframe, sort it and used the head function to show the 10 first highest values.



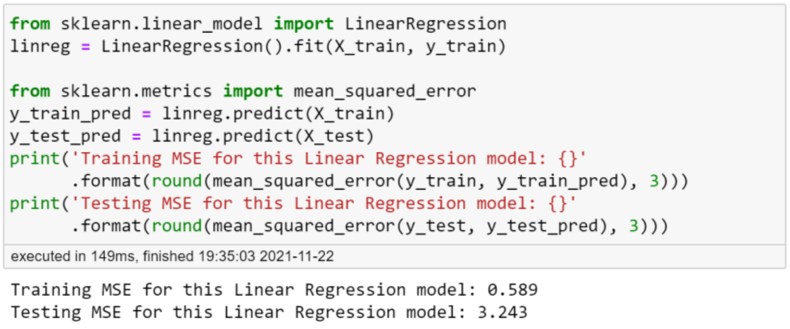
	correlated_user	correlation
0	583	0.551171
1	831	0.725494
2	896	0.784047
3	364	0.640055
4	896	0.528441
5	99	0.612641
6	239	0.602601
7	896	0.514100
8	896	0.706144
9	1004	0.752591

Before starting any modeling, I divided the data into features and target variables. Features being the ratings of the movies and targets being the answers to the more personal questions.

Then, using the features and target variables, I separated them again into training and testing data frames in order to be able to both train and properly assess the model's accuracy.

Model 1: Linear Regression

I used the sklearn library in Python in order to quickly train the model with the training data and obtain the training and testing errors.



```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression().fit(X_train, y_train)

from sklearn.metrics import mean_squared_error
y_train_pred = linreg.predict(X_train)
y_test_pred = linreg.predict(X_test)
print('Training MSE for this Linear Regression model: {}'
      .format(round(mean_squared_error(y_train, y_train_pred), 3)))
print('Testing MSE for this Linear Regression model: {}'
      .format(round(mean_squared_error(y_test, y_test_pred), 3)))
```

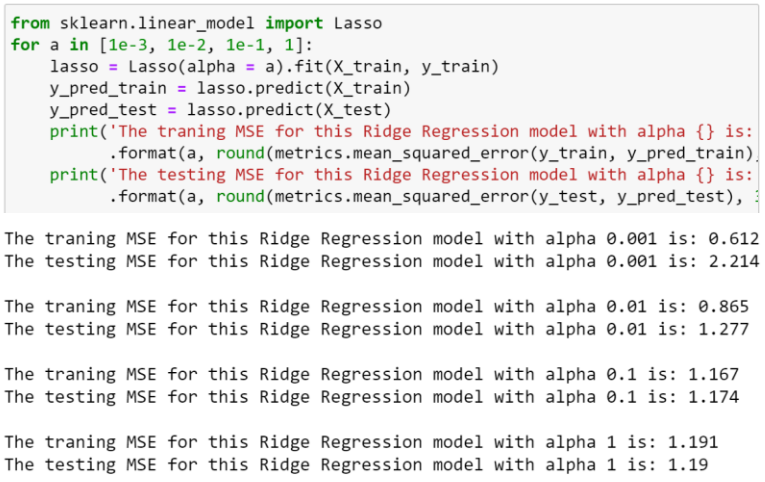
executed in 149ms, finished 19:35:03 2021-11-22

```
Training MSE for this Linear Regression model: 0.589
Testing MSE for this Linear Regression model: 3.243
```

Training error is much smaller than testing error, which might indicate a potential overfitting issue.

Model 2: Ridge Regression

Similarly, I used the same library to perform Ridge Regression, but this time I trained the model under a for loop with different regularization values.



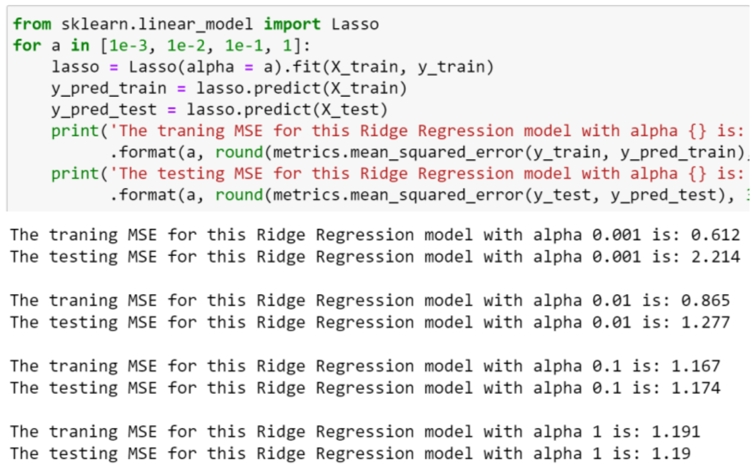
```
from sklearn.linear_model import Lasso
for a in [1e-3, 1e-2, 1e-1, 1]:
    lasso = Lasso(alpha = a).fit(X_train, y_train)
    y_pred_train = lasso.predict(X_train)
    y_pred_test = lasso.predict(X_test)
    print('The traning MSE for this Ridge Regression model with alpha {} is:
          .format(a, round(metrics.mean_squared_error(y_train, y_pred_train).
    print('The testing MSE for this Ridge Regression model with alpha {} is:
          .format(a, round(metrics.mean_squared_error(y_test, y_pred_test),
```

```
The traning MSE for this Ridge Regression model with alpha 0.001 is: 0.612
The testing MSE for this Ridge Regression model with alpha 0.001 is: 2.214

The traning MSE for this Ridge Regression model with alpha 0.01 is: 0.865
The testing MSE for this Ridge Regression model with alpha 0.01 is: 1.277

The traning MSE for this Ridge Regression model with alpha 0.1 is: 1.167
The testing MSE for this Ridge Regression model with alpha 0.1 is: 1.174

The traning MSE for this Ridge Regression model with alpha 1 is: 1.191
The testing MSE for this Ridge Regression model with alpha 1 is: 1.19
```

The previously mentioned potential overfitting is compensated using a regularization value. In this case, a value of alpha=10 is optimal as it reduces the testing error (the ability for the model to generalize)

<u>Model 3: Lasso Regression</u>

Similarly to Ridge Regression, I used the same library to perform Ridge Regression and trained the model under a for loop with different regularization values.



```python
from sklearn.linear_model import Lasso
for a in [1e-3, 1e-2, 1e-1, 1]:
    lasso = Lasso(alpha = a).fit(X_train, y_train)
    y_pred_train = lasso.predict(X_train)
    y_pred_test = lasso.predict(X_test)
    print('The traning MSE for this Ridge Regression model with alpha {} is:
          .format(a, round(metrics.mean_squared_error(y_train, y_pred_train),
    print('The testing MSE for this Ridge Regression model with alpha {} is:
          .format(a, round(metrics.mean_squared_error(y_test, y_pred_test),
```

```
The traning MSE for this Ridge Regression model with alpha 0.001 is: 0.612
The testing MSE for this Ridge Regression model with alpha 0.001 is: 2.214

The traning MSE for this Ridge Regression model with alpha 0.01 is: 0.865
The testing MSE for this Ridge Regression model with alpha 0.01 is: 1.277

The traning MSE for this Ridge Regression model with alpha 0.1 is: 1.167
The testing MSE for this Ridge Regression model with alpha 0.1 is: 1.174

The traning MSE for this Ridge Regression model with alpha 1 is: 1.191
The testing MSE for this Ridge Regression model with alpha 1 is: 1.19
```

Yet again, we can see that the applying regularization helps with the overfitting of the model. In this case, the optimal value for the regularization method is 0.1