LAB 3 - Multiple Linear Regression

In this lab, we will perform multiple linear regression in Python. To do this, we will need the .csv provided with the assignment on Blackboard.

Again, this is the same .csv file we used in the previous labs. This time, we will investigate the effects of age, experience and power (x_1, x_2 and x_3) to the player's salary (y).

Instructions:

- (20 pts) In order to implement multiple linear regression, we need to have our independent variables as a matrix (X). Therefore, you will need to extract each independent variable column ("Age", "Experience" and "Power" in this case) and concatenate them together (along with a ones column, so you should have 4 columns in total) to form the correct matrix, which should look like this:

$$\mathbf{X} = \begin{bmatrix} 1 & x_{21} & x_{31} & \dots & x_{k1} \\ 1 & x_{22} & x_{32} & \dots & x_{k2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{2n} & x_{3n} & \dots & x_{kn} \end{bmatrix}$$

Here, instead of having one single variable as a vector, we have multiple vectors forming a matrix. Concatenating vectors can be done in many ways, which we experimented with, while working on the Python exercises. The easiest method would be to use the `column_stack` function within the `numpy` package.

- (30 pts) For each column in the matrix, we obtain a different coefficient. The aim is to find coefficients $\hat{\beta} = \{\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_n\}$ such that:

$$y = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots\dots + \hat{\beta}_n x_n, \text{ where}$$

$x_i,\ i = 1, 2, 3, \dots, n$ is our set of input variables, and
$\hat{\beta}_j,\ j = 0, 1, 2, \dots, n$ is the set of estimated coefficients.

The formula to manually calculate the coefficients is given below:

$$\widehat{\boldsymbol{\beta}} = [\boldsymbol{X}'\boldsymbol{X}]^{-1}\boldsymbol{X}'\boldsymbol{y}$$

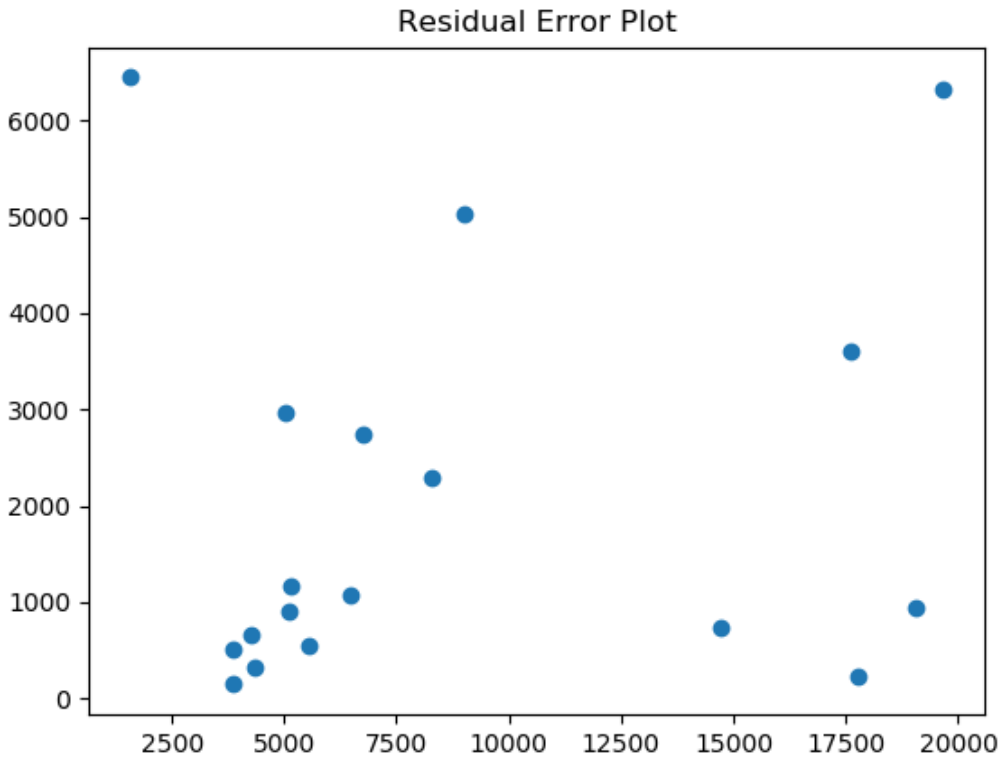
(Note that this is a linear algebra equation and you should use the linear algebra functions within `numpy` in order to compute the result.)

- (25 pts) After calculating the coefficients, we need to plot the results. Unlike single linear regression, we don't have a regression *line*, but a regression *hyperplane* (with 4 dimensions), which unfortunately cannot be plotted. We can, however, plot the *errors* for each observation.

    - Make a scatter plot such that the x-axis corresponds to the predicted y values, and the y-axis corresponds to the error for each predicted value.

    - The formulas for predicted y values and error margins are:
        - Predictions for y (salary): $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$
        - Error margins for predictions: $|\hat{\boldsymbol{u}}| = |\boldsymbol{y} - \hat{\boldsymbol{y}}|$

    The output of the plot should look like this:



- (25 pts) Calculate and display the R^2 score for this prediction by using the same calculations as in the previous lab.

Then, change your input matrix X by simply adding an extra column to it. This extra column should have random integers from -1000 to 1000. You can use the function

`numpy.random.randint(low, high, size)`

to create the vector. You can easily use the function `column_stack` to append this column to the existing matrix X.

Calculate and display the R^2 score once more. Here is an example of a desired output:

```
Showing original results:
R^2 score: 0.8080911002741027
Showing results with an added random column:
R^2 score: 0.8172598291906547
```

The first score is the absolute true value. The second one, since it includes random numbers, is going to be different each time you run the program. But it won't ever be *less* than the original, which is the point of this section.

Important note: Similar to the previous labs, these instructions require you to implement the calculations manually. Any submissions which bypass such calculations will receive 0 points from corresponding parts.

Now, for the insight section:

All the basics from previous labs are also employed here. The only difference is that since we have multiple input vectors (in the previous labs it was only one, namely “Age”, but now, it is the combination of “Age”, “Experience” and “Power”), we combine all of them into a matrix, and our coefficient calculation now takes the form of *linear algebra* equations.

Please note that we could not plot our model directly, since it was a multi-dimensional model. We could see the error margins for the model, though. It is still valuable to be able to visualize *certain* aspects of the model, in addition to analyzing the numbers, and that is what we did here. Most popular models are multi-dimensional, and we usually come up with some metric to define how well these models performed. We then visualize those metrics.

The last section is particularly interesting for illustrating how training works. We created a random column of numbers between -1000 and 1000. This column obviously has no relationship with a player’s salary, **but including it increased the R^2 score! Think about why that might be.**

The thing is, it is still not a good idea to include such vectors. The reason is, even though your training error decreased, your test error is very likely to increase! That is not a guarantee, since the random numbers might align with your model so well that it could be beneficial to have those numbers, but that would be pure luck and wouldn’t be the case overall. In a usual case, since your model is now catered to these random numbers, it won’t be able to correctly estimate salaries for a new batch of players.

Feel free to employ the train-test concepts on your code: Divide your data into two parts (train set should preferably be bigger), train your data using the train set, and then come up with predictions using the test set. Then, you can see how prediction behavior changes with an added random column. You can also try running the same code multiple times and see how randomness affect your results.

We will move on to different multi-dimensional models in the future labs.