LAB 4 - Multiple Linear Regression Using k-fold Cross-Validation

In this lab, we will perform multiple linear regression with using the k-fold cross-validation technique. To do this, we will need the .csv provided with the assignment on Blackboard.

We will perform the same regression as LAB 3 (i.e. we will investigate the effects of age, experience and power to the player's salary), but this time we will perform it multiple times, once for each "fold". A fold is simply a slice of the data, formed by taking several rows from the original matrix.

We will select $k = 10$. This means that we must split our matrix into 10 equal parts. If the number of rows of our matrix is not divisible by 10, then the last fold should consist of the remaining number of rows.

(50 pts) You should perform a loop (pay close attention to how the loop iterates!!!), where:

- The next fold is selected as test data, and the rest becomes training data. You can use `numpy.delete()` to form the training data.

- The regression coefficients are calculated using the training data.

- The estimations for the *test* data are found. Simply compute $\hat{y} = X\hat{B}$ where X is the test data. The estimations should be stored inside an array for future use. Label these estimations as `cv_hat`.

- If the matrix has less remaining rows than the size of a fold, you should again follow the same steps listed above (using all remaining rows), but you should do so *outside* of the loop.

The loop iterates until all folds are used as test data. After the loop (and the calculations for the last fold) finishes, calculate and display MSE for the predictions stored in cv_hat. The calculation for MSE is as follows:

$$\frac{\sum_{i=1}^{n}(\hat{y_i} - y_i)^2}{n}$$

Where $\hat{y_i}$ are the predictions and are the original y (salary) values, and n is the total number of elements.

(15 pts) You should then do the regression one more time, using *all* of the data both as test and train data. Label your predictions as `y_hat`. Calculate and display the MSE for this set of predictions. The values should be as the following:

```
MSE with cross-validation:  8276648.562778284
MSE without cross-validation:  6468033.2283146065
```

(35 pts) Remember the estimate-error plot you did in the LAB 3? Now, you will perform the same plot twice:

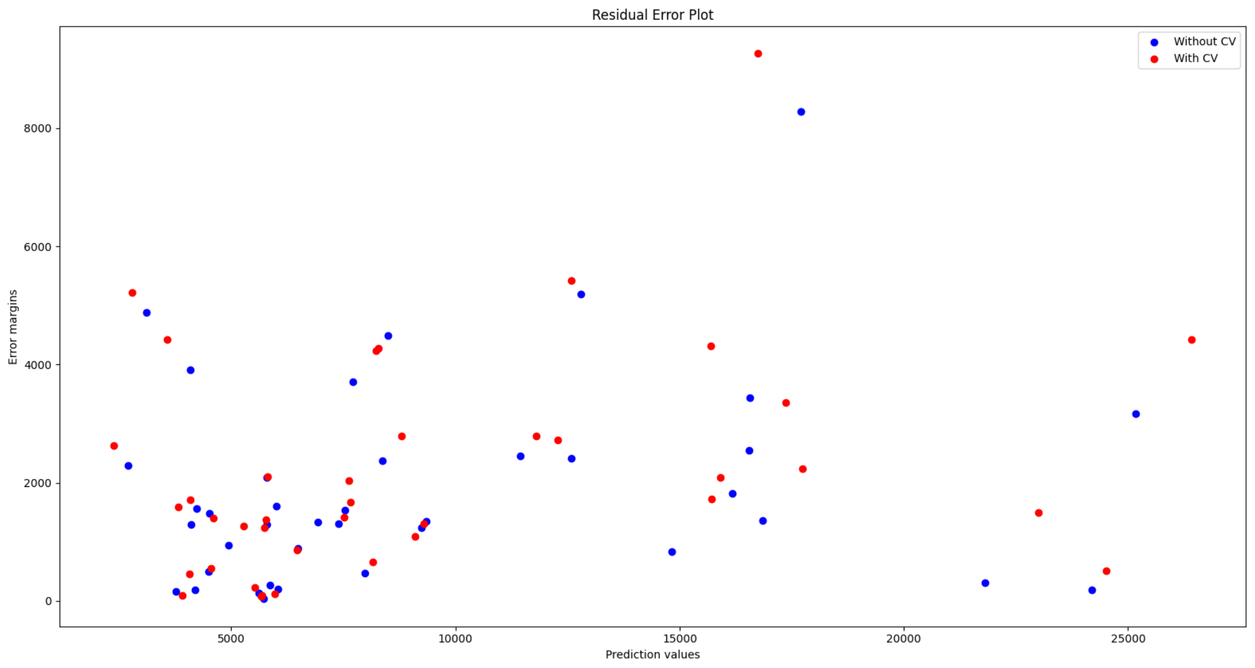
- One plot will be done using the prediction values from the first task, `cv_hat`. Calculate the errors for these predictions, then make a scatter plot using the predictions as your x-axis, and the error as your y-axis.

- The other plot will be done using the prediction values from Task 2, `y_hat`. Again, calculate the error and plot like you did previously.

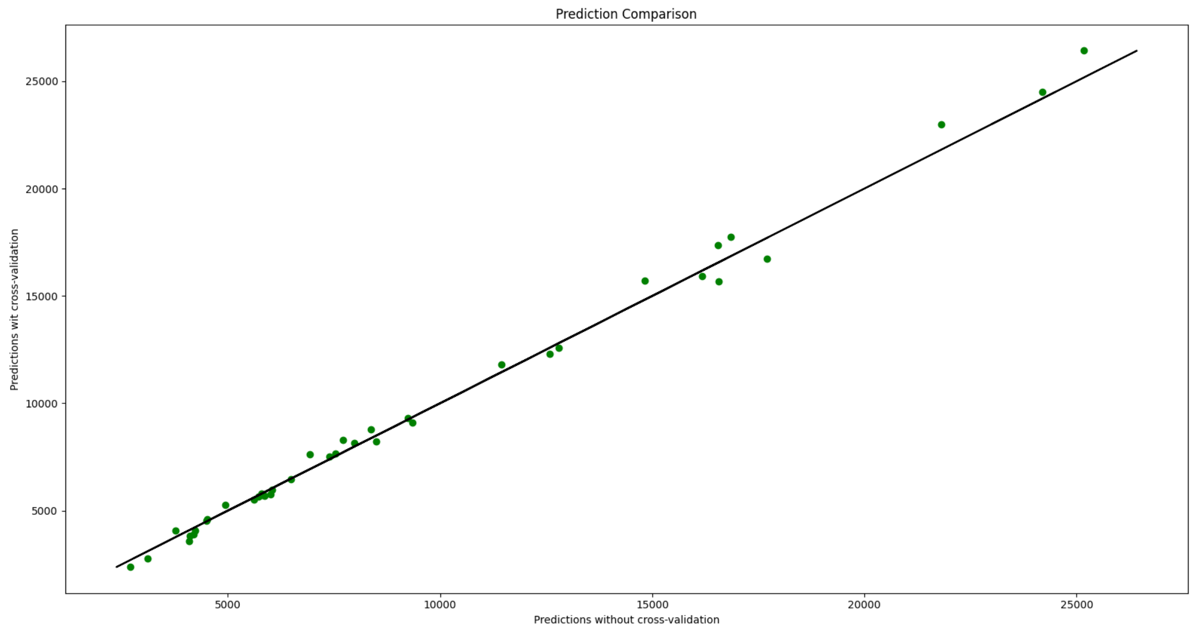
- Draw both plots on the same figure (window).

- Implement a legend so that we can see which plot belongs to which set of predictions.

The output should look like this:



Then, just to see the differences between predictions, we will plot `y_hat` against `cv_hat`. In a new window, make a scatter plot with `y_hat` as your x-axis and `cv_hat` as your y-axis. To be able to clearly see the differences, also plot a line on the same window at $y = x$. The output can be seen below:



As always, please use `numpy` arrays for any numerical arrays or matrices. Like previous labs, bypassing these implementations with the use of outside libraries will result in grade deductions.

Also, another important note: This implementation should work for every possible dataset, therefore avoid any implementations which are specifically designed for the given dataset.

Continuing with the insight section:

Cross-validation is a very useful technique which allows us to get better understanding about our model, especially if we have a limited sample size. Overfitting/underfitting issues can be monitored efficiently.

Notice that the MSE value for the second task was lower than the first one. This is because we created our model using some data, and if we use the *same* data to test, our error is going to be minimized (that was the goal when creating the model in the first place). In cross-validation, we take some data points as training data, so the rest will be unfamiliar for our model, and the error margin will be undeniably greater. However, we can more accurately determine if our model is effective or not.

In k-fold cross-validation, which we implemented today, we take equal portions of our data (k portions in total) and use them separately for testing. When a portion is used for testing, the rest becomes training data. Leave-one-out cross-validation (LOOCV) is an extreme case where k is equal to the total number of data points we have. In that case, a fold simply consists of a single row.