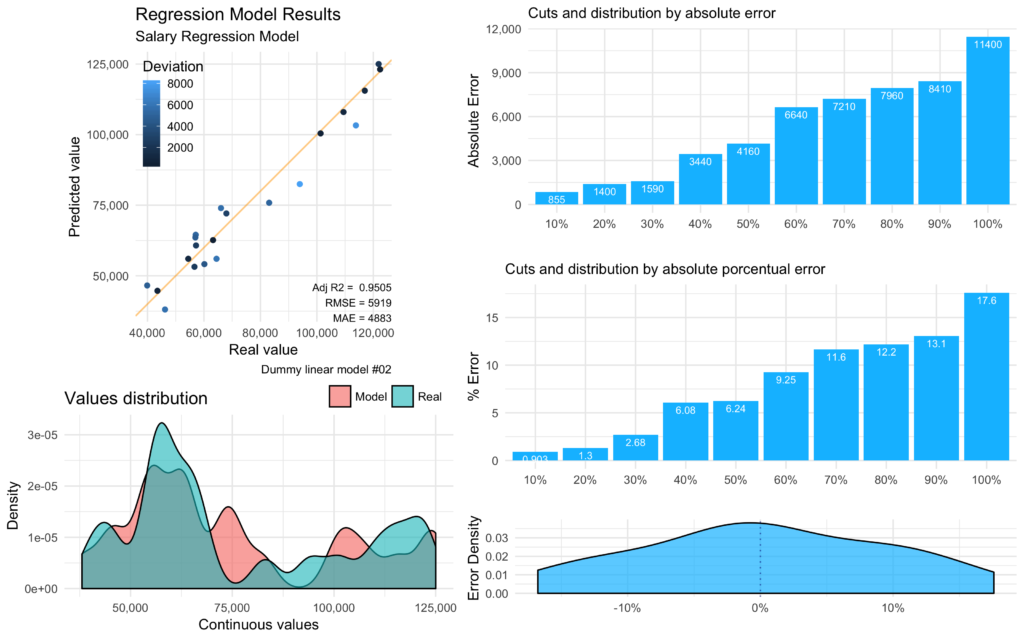
I’ve decided to create and share some new function to visualize and compare whole Linear Regression Models with one line of code. These plots will help us with our time invested in model selection and a general understanding of our results.

Where are we going with this post? Let’s take a quick look at the final output: a quick nice dashboard with everything you’d need to compare and evaluate if your regression model is looking good, compare with others, or get working on further improvements.  
[](https://datascienceplus.com/wp-content/uploads/2018/07/viz_full-3.png)

There’s a simple validator which will automatically label the model as a “regression” if the ‘tag’ values is numeric and have more than 6 unique values; if not, will run as a “classifier”.

**NOTE:** If you wish to replicate the following plots, you can download the dummy data used for the examples [here](https://bit.ly/2uSgiAn).

**Install the ‘lares’ library**

install.packages("lares")

**Regression results plot**

The most obvious plot to study for a linear regression model, you guessed it, is the regression itself. If we plot the predicted values vs the real values we can see how close they are to our reference line of 45° (intercept = 0, slope = 1). If we’d had a very sparse plot where we can see no clear tendency over that line, then we have a bad regression. On the other hand, if we have all our points over the line, I bet you gave the model your wished results!

Then, the Adjusted R2 on the plot gives us an easy parameter for us to compare models and how well did it fits our reference line. The nearer this value gets to 1, the better. Without getting too technical, if you add more and more useless variables to a model, this value will decrease; but, if you add useful variables, the Adjusted R-Squared will improve.

We also get the RMSE and MAE (Root-Mean Squared Error and Mean Absolute Error) for our regression’s results. MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. On the other side we have RMSE, which is a quadratic scoring rule that also measures the average magnitude of the error. It’s the square root of the average of squared differences between prediction and actual observation. Both metrics can range from 0 to ∞ and are indifferent to the direction of errors. They are negatively-oriented scores, which means lower values are better.

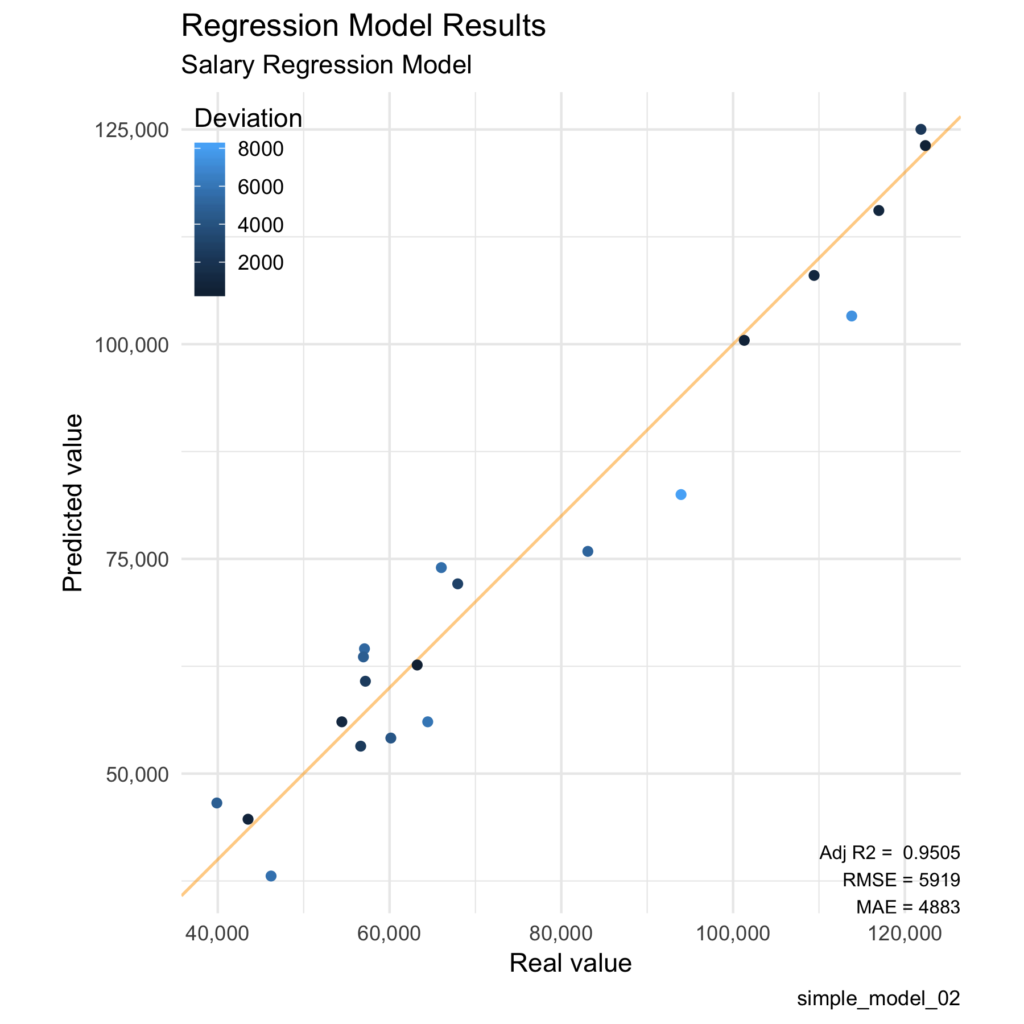
Having a data.frame with the ‘label’ and ‘prediction’ values (real value and obtained from the model value) in our environment, we can start plotting:

lares::mplot\_lineal(tag = results$label,

score = results$pred,

subtitle = "Salary Regression Model",

model\_name = "simple\_model\_02")

Which will give us (and save into our working directory if set to save = TRUE) the following plot:  
[](https://datascienceplus.com/wp-content/uploads/2018/07/viz_lineal-1.png)

**Errors plot**

If we achieved a good model, now we have to learn to deal with its errors. As I said in the other post, this is the easiest plot to explain to the C-level guys to understand the consequences of getting your model to production. For example, if we tell them that our regression has an Adjusted R2 value of 0.97 and a great p-value of 1e-14, probably, they’ll blink twice and give you an awkward face. But, if you tell them that you trained your model and when tested in an untouched-by-your-algorithm dataset you get only a 5% of error in 90% of the cases, that is much more worthy to sell (depending on the final goal)!

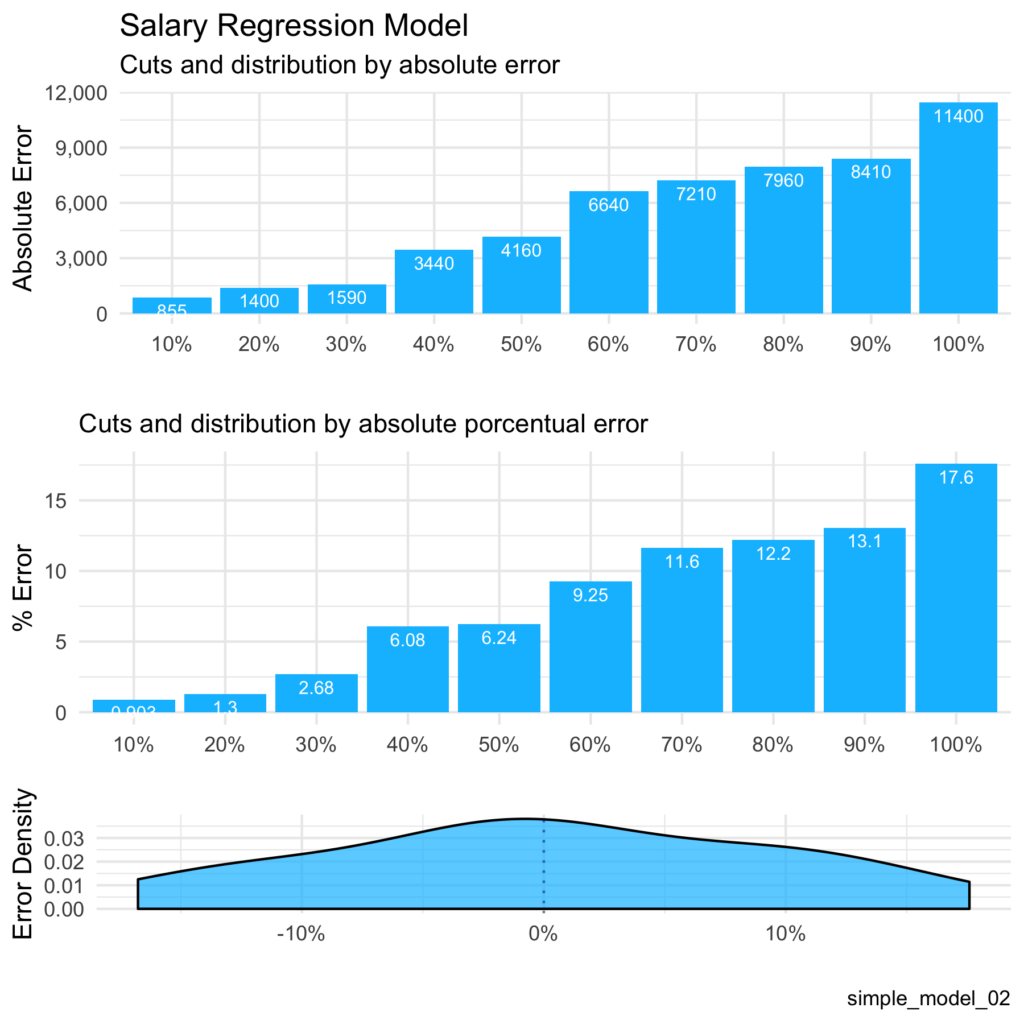
On the background, I calculate the error of each calculation, absolute and porcentual, and sort them from the smallest errors to the worst. Finally, I split them into n different buckets with the same amount of observations. As you can see, we can conclude how much of our test set have an error below a value or percentage with the first two plots. The third one is a density plot for the real porcentual error, with a 0 reference line, where we can see in which ranges our errors are most common.

lares::mplot\_cuts\_error(tag = results$label,

score = results$pred,

title = "Salary Regression Model",

model\_name = "simple\_model\_02")

Which gives us these three plots:  
[](https://datascienceplus.com/wp-content/uploads/2018/07/viz_ncuts_error-1.png)

**Distribution plot**

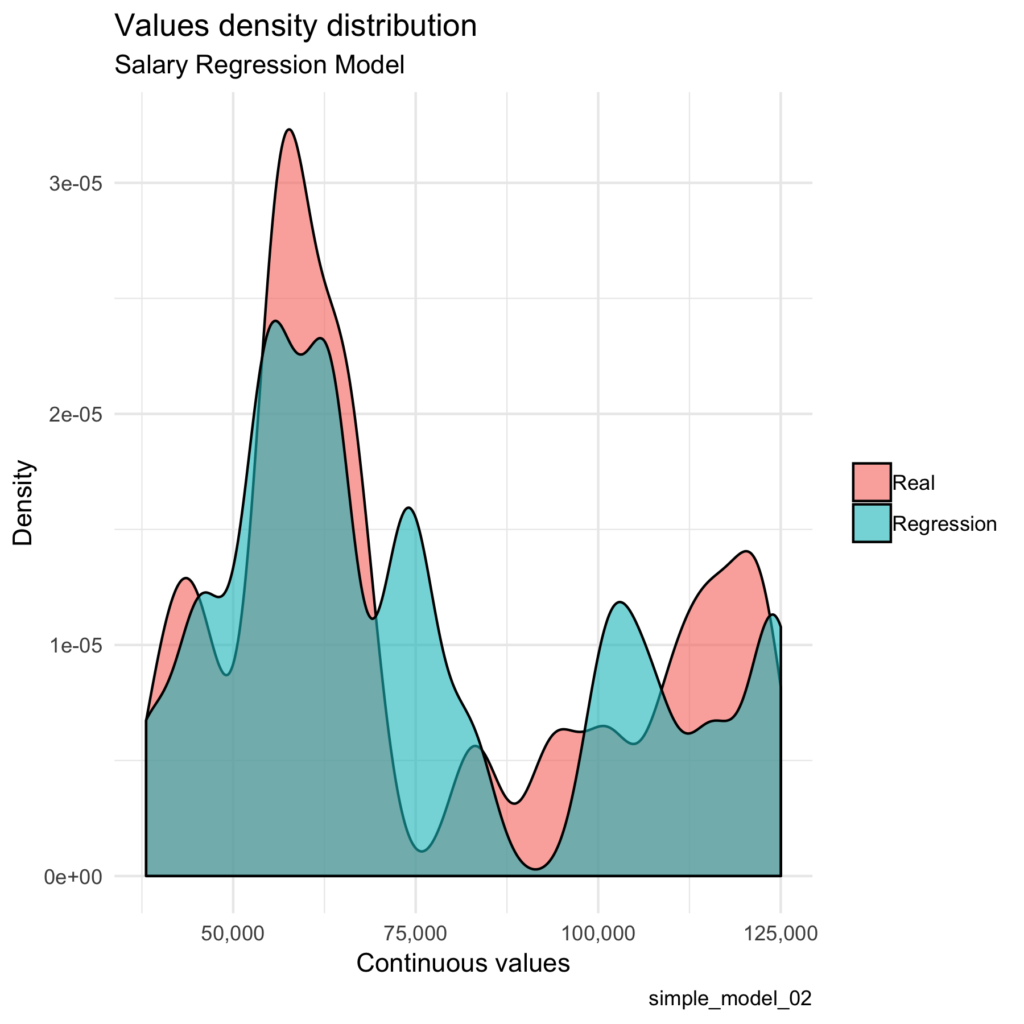
This is a simple comparison between the real values and the predicted values. The more similar this two curves are, the better. I give a smaller size to this plot because it gives us an idea of the whole picture more than a specific metric.

lares::mplot\_density(tag = results$label,

score = results$pred,

subtitle = "Salary Regression Model",

model\_name = "simple\_model\_02")

Plots:  
[](https://datascienceplus.com/wp-content/uploads/2018/07/viz_distribution.png)

**Final All-In Plot**

We do not have to plot non of the above functions to get this result (that’s the whole point of this post). We need to provide only two values: tag for real values and score for our model’s prediction or regression. The rest of the parameters may not be used.

lares::mplot\_full(tag = results$label,

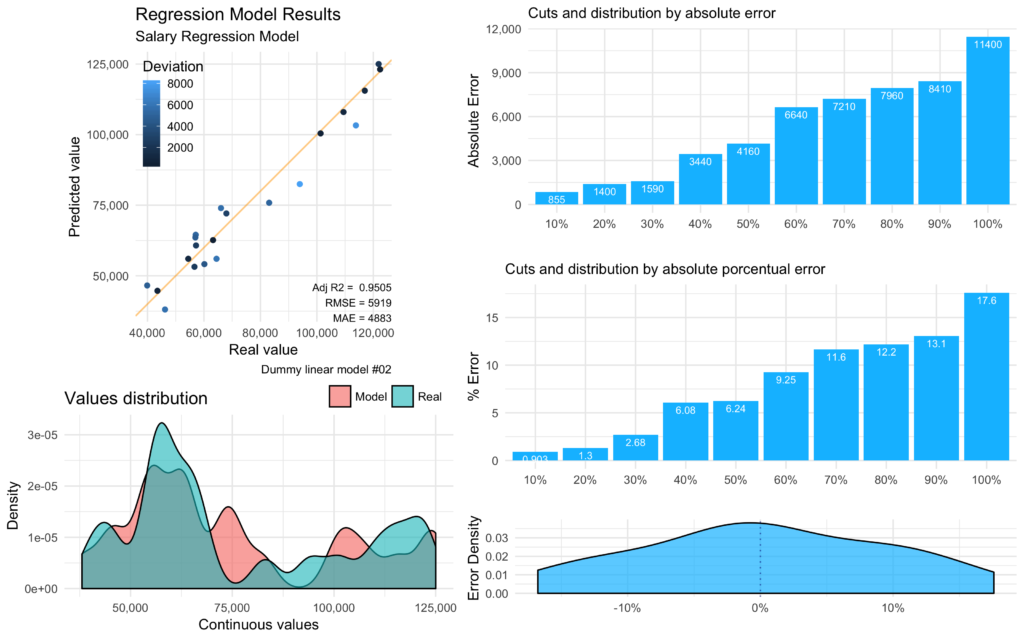
score = results$pred,

splits = 10,

subtitle = "Salary Regression Model",

model\_name = "simple\_model\_02",

save = T)

[](https://datascienceplus.com/wp-content/uploads/2018/07/viz_full-3.png)

These functions were adapted into my past mplot series functions from the lares library, merging some of them so they plot automatically, whether we have a categorical or a numerical input. Remember, if your tag values have more than 6 unique values, it’ll assume you are studying a regression.

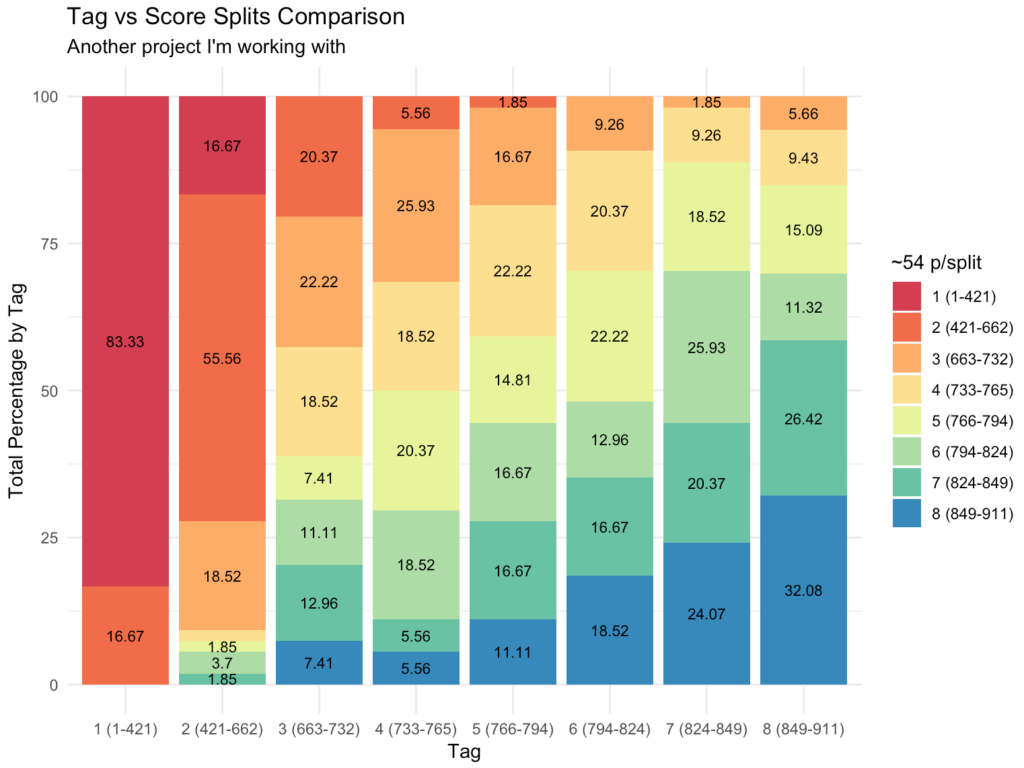
**BONUS: Splits by quantiles**

One of my favourite functions to study our model’s quality is the following. What it shows is the result of arranging all scores or predicted values in sorted quantiles, from worst to best, and see how the classification goes compared to our test set. Being a regression model, we won’t always need this plot but I though it might be useful for some cases.

lares::mplot\_splits(tag = results$label,

score = results$pred,

split = 8)

Which will give us something like this:  
[](https://datascienceplus.com/wp-content/uploads/2018/07/lares-splits.png)