Handling Missing Values and Data Cleaning

Transcription Description

Hello and welcome to section six of our course improving machine learning model performance, feature engineering and hyper parameter tuning. In this section we are going to cover a bunch of techniques and we will be covering them intuitively, we will not be going into too much detail, but we'll cover them enough so that you can apply them in your real-world machine learning projects. Specifically we are going to cover in this section identifying missing values, data cleaning and scaling, converting numerical features two categorical, outlier detection techniques, using log of numerical features to remove distribution skew, determining feature importance, deriving additional features from important features, model performance improvement with cross-validation, model performance improvement with hyper parameter tuning, grid search and random search to find best hyper parameters, this basically belongs to the category of model selection. So as you can see there are a bunch of techniques that we are going to cover in this section. In this specific video we are going to cover identifying missing values and data cleaning and scaling. So let's get started. Since we have covered most of these techniques before, so this section is mostly going to be about code walk through, they will be very little theory and slides. So let's start with importing a bunch of libraries as we always do, we will walk through a set of utility functions that we have written specifically for different type of pre-processing tasks that we will come across. What we will do is we will just skip the functions right now and as we have the need to call them we will come back to them and one by one, and then walk through the code. The dataset we are going to use for this example is called Iowa housing dataset. We have come across two housing datasets before, the Boston housing dataset and the California housing dataset, they were both regression problems and so is this one. Their difference is that this one has a lot more features than the previous datasets, so this is a good example to consider when we are going to do some feature engineering because there are quite a bunch of missing values in different features, so we'll get a lot of practice. This is the specific link for this competition, it's this competition has been created to specifically predict sales prices and that is feature engineering, and there are already 4970 teams, and purpose is to practice creative feature engineering in advance regression techniques by random forests and routine boosting. I already downloaded the dataset. The dataset has been placed in this path on my computer, yours will be different obviously. So I created a folder called Iowa housing and I'm going to read the raw DataFrame and the test because there are two different datasets, one is for testing where we are not given the target variable or the values of the target variable, and there is the actual training dataset where we are given the periphery, that's pretty normal in Kaggle competitions. The pandas flag to read CSV low-memory = false means pandas will process the CSV file in in chunks, which is useful for large files and it saves memory while processing. If we are sure about the data types that pandas is not going to get confused about data types it's a good safe practice to use low memory equal to false while using read CSV. So once we have the DataFrame we can just print its shape and information about it. So as you can see there are 80 columns in the desk and 81 in the raw frame, because they're the target column is also included in the raw frame whereas it's not in the test frame. Let's print the info, so here we have ID which is just a number or incrementing number, this is ms sub class everything is like 1460 which is the size of our data for the training set, and there are a bunch of missing values as you can see. We only have 91 values for this particular field and only seven values for houses with pools. Let's describe the data frame so as usual we get a bunch of counts minimum maximum value standard deviation and percentiles for each of the columns, all the 80 columns are not listed here, this sale price this is our target column. Now the first thing we do is we just copy the raw DataFrame into another DataFrame just so that we leave the origin one alone. And another important thing we need to do is to extract our target variable into a variable called Y, which is in this case sale price. We drop it from the DataFrame that we are going to process next, which is the DF DataFrame which we used to get the copy of the raw one. So we have written a function called extract and draw a target problem, the reason we have done that is because this is such a common task that we need to perform on almost all DataFrames or datasets we are going to use. So let's look at this function line by line, so it takes a DataFrame we have in it takes the name of the target column and another flag called in place equal to false by default with means that is not going to modify the incoming DataFrame, it's going to create a copy if in place equal to false, so we just check if the in place is there we just directly have our local variable DF 2.2 DF in otherwise we create a copy in the local variable DF. Now there could be two cases that the target variable is numeric type, which in this case it is because it's a regression problem, so we have a sale price which is a floating point value. So if it is not numeric type in the DataFrame then it is categorical type, for all categorical types what the pandas DataFrame stores is a subfield called cat, in each value of the variable which is categorical, and the cat also contains the codes which are the actual codes of the categories, or assigned to category. So what we do is we assign first temporarily the codes to the column in the target variable, and then we assign to y the values of the course. So codes also has another sub field called values which is assigned to Y, so Y now contains a bunch of values for each of the row of the target variable, if it is categorical, if it's not numeric. So is numeric d type this is a utility function which is available in pandas dot API dot types is a string type and is male type, these are very nice utility functions that you will find handy. Now on the other hand if it's numeric then we just simply assign it to Y and we drop them from the DataFrame itself, and we return the DataFrame and the target variable column as separate entity for a separate pandas series. Next what we are going to do is we're going to combine train and test sets into a single DataFrame, so that any pre-processing, cleaning, scaling and the like or any feature engineering we do would be on the combined datasets, so we don't have to repeat it for the test set. To do that we use PD dot concat for D F and D F test and we set the index. Reset index what it does is because test set indexes also start from zero and the train test indexes also start from zero, for both the indices it will just concatenate them and it will just number them starting from the Train and continuing on to the test. Another thing we do is we drop the ID field because we really don't require it, it has no predictive value. Before combining we also store the number of row in the Train test in the test set and we also store the IDs of the test set, because we'll be needing them when we have to submit our solution to Kaggle if we decide to do that. The next very important thing is to get a separate list of categorical and numerical columns, and the way we do that is we have a utility function called get category columns by type any pasta combine DataFrame. And so let's look at this function what it does, so here is the function and for column name and column values in DF dot items, so DF word items will return the name as well as the values for each column of the DataFrame. So if the column values are of a string type we append them as string with the column name and the type is string as a tuple in the list, if they are not a string and they are not numerical then we just categorize them as categorical. So when we print our categorical columns returned by our function most of them are string columns because if you look at there we have mostly these object types which are strings basically. And there are 43 out of 80 columns which are string type or categorical in nature, from this list what we can do is we can just get the first value of the tuple, which will give us the name of the column with v1 as a separate list, and we get this list. So these are our categorical columns and to get the numerical columns, what we can do is just anything that is not categorical in combined dot columns is going to be treated as numerical, we can do either this, or we can also get it from another function called get numerical columns, as this is numeric d type checked, so it's a kind of a counter part of the categorical function. The shape of the combined is 79 because we drop the ID column and the print the head of the combine just to see everything is in order so there are a lot of missing values still. Because of the constraints on the time length of our video we had to divide it into two parts, so in the next part of this video we will discuss handling missing values and scaling numerical features. So we will continue on the same notebook in the next video.

# Handling Missing Values and Scaling Numerical Features

# Handling missing values

# Scaling numerical features

Handling missing values and scaling numerical figures. So we are going to continue in the code from where we last left off, so let's continue. Now we are going to handle missing values. So this function handled missing values is going to a the DataFrame and the categorical columns in numerical columns. So the dictionary we are passing which default is null is just an option that we can use to set how we are going to replace the missing values if we want to, right now we are not using it so we just skip it, otherwise it is going to be just an amputation area. Now this loop is for numerical values, so if it is not numerical we just continue, otherwise we see if it's null, we just sum up all the null values of this column. We add another column in DataFrame which is a good practice in general, and it might improve your prediction results in certain cases, but it's a general good practice to add a separate column for any missing values that we find. So if the sum is nonzero we have missing values then we can just add another column with the same name and with underscore na appended at the end, append a boolean whether this wherever that column has missing values, and we create a filler which is going to be taken from their dictionary if we passed one, otherwise we just take the median. And we also record the filler which is the median value of that particular column with the column name in our dictionary, so that the final filled value will be returned. Now for the categorical values we go through the categorical columns we fill the missing values with the mode of that column. So the most frequently occurring value, after that we create a label encoder and we fit the values of that column which all the null values have been filled with the more previously, just on this line, and then we just label encode all the values. So we are using scikit-learn label encoder to encode all the values after we have reply missing varies with the mode. So this is the basic two things that this handle mission values function does, it will handle a lot of cases for different data sets which we can come across in terms of handling missing values. After calling we get the distance back and you can see that these were the columns that were filled with the median value, and some of them have zero median value, so they had missing values but the median turned out to be zero. Now after this we can find out the percentage of missing values and for this we have a utility function, so this will return a list and we can just sum up the list to get the total missing and the total values we can get from the NP dot product of the shape, which is basically rows x columns, and we just calculate the percentage. So as you can see we don't have any missing values. We already recorded the number of rows in the training set, so we can use that number to split the combined DataFrame again into training and test DataFrame. And we get back the same number of rows for training in test and we have increased our columns to 90 because we added defenders for any columns, we had 79 previously now we added 11 of them so we have 90 now. And just to verify if our training set now has any missing values, there we don’t. One of the things you can do is you can use the to underscore feather method which is now available in pandas to store the DataFrame in a very fast way into a file on disk, and you can just read it back. So this is one way to store the DataFrame on disk in a binary form, in this reddit, it's much faster than pickle files, so it will come in handy if you want to just save your DataFrame and load them again later. We spread our training set into a drain test spread with validation we call our test set as validation set because our original test set does not contain the target column, so we will just read that one separately. This test set is going to be called our validation set. Now just to a baseline model we will use random forests a regressor, and we just create a function to make predictions, and so what this function is going to do it is going to use mean square error because that's what the Kaggle competition uses for evaluation of the models in this particular case. So we have imported mean squared error from scikit-learn, so we're going to pass the y train the training set after splitting, and the prediction of our model on the training set, then again the same for the validation set, and the score of the model which is going to be the R square in this case, but we're not going to use that but we just want to print it just to see how our model is doing. And we just print it out, so he gets create a list of these four things, in the spin it out and if the model also has a know V score as the attribute we will also print the over B score is there. For the random forest we are going to use 1500 estimators because our dataset is relatively small, so it is a good practice to use a larger value of the number of estimators for small datasets because we only have like 1450 or 1460 rows, and we just create a constructor and we just chain our fit method to that constructor in this assign it to the variable our F model for the training set. So our baseline model gave us some values for the MSE this is for the validation set which is a pretty high number, because we did not do any scaling so this number is expected to be high, and for the OOB score it's point eight six. Next what we can do is we can apply scaling to the combined DataFrame that we have, and the way we do it is again we have this utility function scale numerical columns, so we are going to apply obviously the scaling new mega columns. By the way we also stratified on the overall quality because this categorical column gives us the overall quality of the house that we are trying to predict the value of. We've also used it here as well. So the scale numerical columns function is pretty simple we pass a mapper, which in this case is going to be none, and we passed DataFrame and we are going to come back to the mapper what it's all about when we are going to pass one. In this case since you are not passing the mapper, so it's just going to perform standard scaling which is if mapper is none, then we just used a constructor for the standard scalar, and the column of the DataFrames if it is a numerical type. We create this list and we use a DataFrame mapper which is going to take a tuple or a list of tuples of the column list and the function to be applied to all the values of the DataFrame based on the column. And it's going to reassign it to the DataFrame for each of the columns that it is applied the scaling to. So this is a transformation that is doing, we first create the mapper and then we just transform the whole DataFrame according to the column list, and that with those column lists will come into the transform names, and that list will be used to index into the DataFrame, and apply the transformation so that the scaling will be applied. This DataFrame mapper is part of sklearn under scope pandas library which is separate library that has a lot of utilities for working with pandas DataFrame and sklearn, and we have just used the DataFrame mapper from this library. You can install this library using pit stall. In this video we cover some important aspects of handling missing values, data cleaning and feature scaling but we are not done yet. In the next video we are going to look at techniques for handling outliers using the log of target variable and numerical values, and then removing the distribution skew from numerical variables.

# Handling Outliers and Removing Distribution Skew

* Handling outliers
* Converting numerical features to categorical
* Detecting distributed skew in numerical feature
* Using log and other transformation of numerical feature to remove skew
* Determining feature importance

Topic of this video is handling outliers, removing distributions skew and converting numerical features to categoricals, specifically we are going to cover handling outliers, converting numerical features two categorical, detecting distribution skew in numerical features, using log and other transformations of the numerical features to remove the skew, and determining feature importance. Now before we go into the code we just want to recap how a normal distribution looks like and what does a skew mean. A normal distribution as you all know looks like this, it's just like a bell curve for any feature or any column that we might have in our dataset, and a skewed distribution is like this. So it might be skewed to the left or skewed to the right, in the right case it's called positive skewness and the left case it is called negative skewness. We will have a relatively longer tail on the right side in case of right skewed and a relatively longer tail on the left side in case of depth like this one. So this was just a recap of what we mean by skew and what a positive and negative skew look like. Now it turns out that if we endure the skew and we try to normalize, or try to make our distribution closer to the bell curve as much as possible it results in better model performance, and that's what our goal is finally. So we will study some basic techniques of removing this skew. So let's go to the code now, as usual be import a bunch of libraries we have our utility functions. Now here is a function that we have written it's called pre-process underscore DF, pre-process DataFrame, what I have been doing in the previous video I am just building upon that, and I move the previous thing stuff where we are extracting and dropping the target column, and doing some other stuff the DataFrame like combining a train and test frame into one and so on, I have moved all of that code into this function, so that once we call this function with different parameters, so we have bunch of keyword arguments, we have more control and we can **reuse the code instead** of copying it in every cell and doing the same thing again and again when we are doing experimentation, by performing different type of feature engineering and so on. So this kind of function becomes very handy for that kind of iterative testing and performance improvement. So what we will do is we will introduce the code of this function, some of this code you have seen before but we will introduce new code as we have the need. So the first thing we do in this function is just have a pass drop target flag and if it is true then which is abstract and drop the target column as before. Another thing that we did not do before was NumPy has a function called log1p which basically adds one to a variable and then takes the log of that we are doing that for Y for our target value, and what this does is it removes the skew of the target variable and it's generally a good practice. That's what we do here for the target variable, otherwise we set Y to none because we might have handled the target variable ourselves outside the functions. This is what if log Y is equal to false means that we don't want this function to do anything to the target variable. Then we come to the combining the DataFrame train and test sets. Another thing we are doing here is that if DF test is not none and ID column is not done, then we are going to drop the ID column from the combine frame and also store the IDs in the test ID as before, but this time we are going to return this thing from this function otherwise we return nothing. Now we will come to this part of the code later in the video just skip it for now, this is basically converting some of the numerical columns to categorical columns, and we'll see why we need to do that. Then this is the same code as before, we're just getting the list of categorical and numerical columns respectively, and here is an interesting part, if remove students is true, what we are doing is we are applying all the numerical columns so numerical column if we remember is a list, and so we are indexing the combined DataFrame with that list, and to each of the numerical columns we are applying this lambda function which calls the skew function which we got from here. So in SciPy dot stats library we have a function that measures the skew, and also remember we have another function called boxcox1p. Now boxcox1p and log transformation, let's take a look at that the law of transformation is often used when the data has a positively skewed distribution, the square root transformation can be effective if we have counts of some variable with large variance in counts, and boxcox1p transformation allows us to apply different transformations, training from a square root to log and others as well in between, controlled by the parameter lambda, and we'll see how it does that. So coming back to the code, here we apply the lambda function and we call this - this - what it does is it measures the amount of skew in the distribution of that specific column. So for each numerical column we will measure the amount of skew as a number, and then for each of the columns skew the values of skew we sort the values in descending order, so ascending is equal to false, what this will give us is a dictionary called skewed columns which will contain the column name as well as the amount of skew. And we can create a temporary DataFrame for convenience out of this dictionary, and we have only one column in this called skew. So this skewness log is equal to wherever the skewness is greater than 4.0, which is just an arbitrary number based on the current dataset. We can pass this number as an argument but we haven't done so in this case, so that gives us the skewness log, and the skewness other variable will have all the values from the skewness where the students is less than or equal to 4. And then what we do is we have skewness features log variable which you get from the index of this genus log DataFrame and the unis feature other which we get from the Skinner's other dot index. And then we have this function called box-cox 1p. The documentation page of this function on SciPy website says this, so it basically computes the box-cox transformation of 1 plus X, so this is the formula, so y equals 1 plus X raised to the power lambda, minus 1 divided by lambda, if lambda not equal to 0, and log of 1 plus X which is essentially equal to NV dot log 1 P, which we already used for Y or target variable if lambda is equal to 0. So this is basically the formula. So now for the features we collected in the features log whose skewness was greater than 4 we apply boxcox1p with lambda equal to 0, so we will apply essentially a log of 1 plus X transformation, and for other one instead of taking square root where if we have lambda equal to 0.5 we just have some middle value of lambda like 0.15 or something instead of 0.5, so we don't really take the square root, we take a transformation which is somewhere between taking the log of 1 plus X and square root, so we have control this lambda is where. We could pass lambda and this range number as a parameter of this function again, but we haven't really done them in this case. You can do this as an exercise. So this is the basic transformation we do if our skewness is greater than 4.0, and where is this 4.0 is coming from I will just let you know. Then we call handle missing values and then finally we get dummies of Wonders function to actually apply one hot and according to all the categorical variables which we separated out here. We also did not do this thing in the previous video. And the last thing we do in this function we apply a scalar based on the scalar mapper that is passed, so we have this keyword argument called scalar mapper, and if it is not none we create a mapper using the DataFrame mapper, and we just call the scale numerical columns was in the mapper as none or the mapper which was passed by creating an instance of the DataFrame mapper, after fitting the past structure, so we'll have to pass a constructor of our scalar as an argument. So this is all from previous video we have seen this DataFrame ever before. And now we finally just return everything that we have processed, we have the number of train and test set variables as well test IDs are separately stored number of numerical columns categorical columns target variable DataFrame in the combined. So this is a very handy function that will allow us to call it again and again by varying the keyword arguments and experimenting with our feature engineering and modeling. So in the next part of this video we will use this function that we have just gone through to apply different type of feature engineering on our housing dataset.

# Handling Outliers and Removing Distribution Skew (Continued)

So let's continue with our notebook and apply the function that we just created on our dataset. So the first thing we do is we just don't use the skewness at all. If we do a DF info on our DataFrame we see that there are some variables like here built, here remodeled when the house was remodeled probably, and similarly years old and months old. These are basically years or months right and so they're not really numbers, they can be considered as categorical features. So what we should do is we should convert them into categorical features, and the way to do that is we first convert to cat columns to our utility function process DF, and we pass this array of those columns which are either here or months, and all the rest we pass is false. So this is the only change we are doing in this case. Code that we left out it's converted to cat column this not none what we do is we convert to STR type, so we convert those columns we do string type and later on they will be created, their null values will be level encoded, and later on when we do the one hot encoding, they will be converted into pure categories. So we did not pass any scalar or any mapper with standard scalar was used. Now if we look from outside that utility function and we won't take a look at our skew columns, what we can do is we can just directly call that piece of code that we saw in the utility function on our DF row, just to get this skewed columns. If we do that we get these values for the skewness as you can see there are many values like 24 14 12 and 9 in higher than four values, and we are printing them in a descending order. So they are values which are higher than four and there are values which are less than four. So four seems to be a reasonable limit for applying the log transformation, or the lambda equal to 0.15 transformation with box-cox 1b, so that's why we chose this. We should really be passing these as an arguments in we actually make our function more useful but for now this will do. Now this was about converting date based or year and month waste where I was two categorical now another thing is about detecting and handling outliers. For outliers we have this function that measures the interquartile minimum and maximum ranges for a numerical column, and because any of the columns that are outside that interquartile range they are to be considered as outliers. So let's see what this function does this is one of the functions that we have in our utility library, as you can see we passed a numerical columns to it and what it does is it goes through each of the column with name and values from the items as before for whole DataFrame and just continues if the column is not in the columns list, which will be numerical columns only, and then we have this quartile 75 for 25 is equal to NP dot percentile of the column values between 75 and 25, so this is the way to get the percentile ranges from NumPy. And then the interquartile range is what is 75 minus quadratic 25, and the minimum value is 25 minus interquartile range into 1.5, and the maximum value is quartile 75 plus IX, so we are actually going a little bit higher than that 1.5 times higher and lower respectively. So anything beyond this minimum and maximum value will be considered as real outlier and then we just populate our audition Airy with the tuple called minimum value and maximum value using column name as the key. So this is basically going to return all the values that fall within the interquartile range times 1.5, on either direction, and from 35 to 75 percentile so this is for the interquartile range. We can call this and see that we have got this interquartile range. For example just looking at one of the variables was like for example from living here, this has an interquartile range from 158 0.625 this is the lowest value and two seven four seven, and if we do a describe on this we see that the maximum value it is five six four two, and the meanest one five one five, and there's no missing value, the 25 percentile is 1129, 75 is 1776. Obviously this is five six four two and any values which are higher than 270 for 7.62 5 they will be considered outliers, and any value less than 158 in this case we have the minimum value 334 so there is no outlier on the lower side for this variable at least, and the median of this is 14 64 which is 50% tied. To deal with outliers one thing we can do is we can use something called robust scaler, that's available in scikit-learn, and the back page of this robust scalars is this. Scale features using statistics that are robust to outliers, this scaler removes the median and scales the data according to the quantile range for two IQR interquartile range, which is what we want and the IQR is the range between first quartile and the third quartile. So it already does scaling based on the median and IQR by default, and centering and scaling happen independently on each feature, and median Inlet of a trial range are then stored to be used on later data using the transform method. So at the time of fitting it stores the IQR range and the median. Outliers can often influence the sample mean variance in the negative way, in such cases the median and interquartile range often gives better results. So this is a very handy function which just like min/max scalar and standard scalar it already performs scaling based on interquartile ranges, which is exactly what we want. And this will help us in handling outliers better. So we able to use robust scaler, with those variables move to categorical the year and month variables we can just apply our model. This is the score we get our validation set score is 0.88 to be score is 0.87 to 5 and our validation set MSE is point zero one eight seven. So this MSE is now in the range of 0.00 something instead of those large numbers that we had in the previous video. The reason is that we applied log 1p transformation on our target label and we also apply the same kind of transformation on our numerical columns. The OOB score has improved because in the previous video our last OOB score was 0.86 one, although it may seem that the validation set score is a little bit down but since brand-new first has this OOB score which actually Donnelly creates a validation set and keeps on applying the model on that, so this score is very authentic in terms of gives us an overall quality of our model. So if this is improving so our model on the average is improving. Now let's do some quick experimentation and see how our score changes. So another thing we can do now is we can just have the remove skewness equal to true, so do we do this again, we haven't passed any scale mapper. Previously we had twenty eight eight two and point eight seven two five as OOB, now it's 0.873 little bit improvement and point gated three which is also an improvement, point zero one eight five seven is our MSE and point zero one eight seven four was our MSE before, so again there is an improvement because we have lesser MSE. Now let's add robust scaler as the mapper into the same thing. So we got almost the same score as before. Now it might be the case that our number 0.04 the skew value and applying different transformations based on that threshold, that may not be a good fit for this dataset, so we may need to play around with this but I'll leave that to you as an exercise. Now let's spend the feature importances, and to do that what we do is we create a series out of the render furnace models feature importances attribute, which is part of the random forest model and we give it the index equal to extreme columns, so the column names of the training set that we have after the split. Then we sort the values of the important theories by descending order and we do it in place, then we print the 50 top features out. So as you can see overall quality is highest in importance, ground living area is second, total based math square footage is third, garage area garage cars first of all. Now one thing here we notice this central air and the overall condition, and also the overall quality these seem to be categorical features, and there are other such features as well we encourage you to go through the list and see what kind of features are actually categorical, but since they have been given us as numbers, or in 64 or floor 64 or something like that, so we are considering them as numerical, but we should change and do categorical. For example the central air conditioning facility it has only two values 1 and 0, and so obviously it is a boolean value, so whether the facility has central air conditioning, and similarly the overall quality 10 values because we scale them so they look a bit odd, otherwise they are ranked from 1 to 10, and similarly goes for the poor or condition. So we should actually change these three at least in two categorical. What we can do is we just want to show you another technique the central air we will just simply add to the list of the months and the year categories, and so center layer is added to that list and it will be counted to the categorical first string type and then one hot encoded. Whereas for the overall quality and overall condition we do this we create additionally we pass a tuple 1 to 11 and the number of slots equal to 10. So this is basically we are creating a bucket and that's called binning, and the code we are going to call our function, if this history is passed as a set of tuples for each column which you want to bin or create buckets for, we call the bin numerical columns function, where we pass the DataFrame and the dictionary. What this does is it just creates a cause n P dot linspace would be divides the data according to the bins specified that we have, and we have three values of bins 0 pin 1 and pin 2, gets the tuple from the dictionary, and then it just creates and uses it to create the linear spacing. And then we pass those buckets to PD dot cut. This is another way to convert numerical very much to categorical. So again we get similar scores for our MSE and our OOB school. Other things you can try after we are done with our pre processing we can additionally apply a min/max scalar or standard scalar to our numerical values once they have been processed by the process function, so we can do that as well, just to try out things. So another version would be to just apply the standard scaling after we're done with pre-processing, so we pass robust scaler and once we are done we just apply a standard scalar as well to all the numerical variables that are left, and see if it makes any difference. So in this video we went through a bunch of techniques for scaling handling outliers and converting numerical features do categorical, using different techniques like binning and one hot encoding. In the next video we are going to derive additional features based on feature importances and see if our model improves by adding more features derived from the already existing features.

# Deriving Additional Features V32

# Adding features from a combination of other features

# Adding features obtained by mathematical transformation of existing features.

Deriving additional features from existing ones and adding them to our existing feature set. We are going to cover adding features from combination of other features, adding features obtained by applying mathematical transformations of existing features, for example taking square root or square or cube of a specific feature to add additional columns. So let's get started. Before we go to feature addition we need to fix the slide issue in our pre-process dia function from our last video. Those of you the kenai might have pointed out this problem. In our previous version of the function if we look at the previous videos version we can see that we are using PD dot get dummies to perform one hot encoding towards the loss of the function, before applying the scaling and after handling missing values, but the problem is that once we have converted our categorical features to a string type, also binning of numerical features from the given list, and we apply the skewness and we then just handle the missing values, what we will end up doing is we will convert everything into numerical, or everything into integer, before applying one hot encoding, whereas one hot encoding searches for either the object type or categorical type or string type features, and then converts them to one hot encoding, so basically our function get demeans is not really doing anything in this case. To fix this what we can do is after applying the binning and convert into string type, we are going to perform one hot encoding after calculating the categorical columns, and then we apply the missing values, and then finally we do the skew removal and then the scaling on numerical columns only, as we doing in this case here. As a result our one hot encoding features would already have been expanded due to one hot encoding the way it works, and then everything will be fine and we will have a much larger number of expanded feature set because of one hot encoding. Another thing we need to change in this function is we have added a new keyword argument new features function. This will be a function that will be called after the combined DataFrame has been obtained, as you can see here we are calling the new feature function, and this function will create new features before any further pre-processing is done. We have a couple of ways to add new features, either we can do it outside this function, or we can pass different types of flags to this function saying that what kind of derivation to perform on which columns and so on. A simple way is to just pass a function so that we have more control and we can just do whatever we want in terms of feature addition and new feature creation in this function, and then let the pre process DF proceed with further pre-processing. So that's the approach we have chosen. Let's continue where we left off in the last video and just reproduce the result so that we can just go on from there. So here we are not passing that new feature addition function, we just want to correct our bug about the 1 hot encoding to see if the results drastically change or not. So as you can see we end up with 659 features instead of the 80 or 90 features that we had before, and this is definitely due to the application of one hot encoding using period or get dummies. Now we can just directly apply the splitting and the random forests. So we got a point eight six zero OB score point eight six two where addition set score n point zero to one eigh. Now these are similar but a little bit less than our previous course, but we have applied one hot encoding and everything so we will go with this approach. Now let's start with adding new features so we are passing add new features one which is the first version of our function which just adds one new feature and let's see what it does. So these are our functions and the first function what it does is it just adds a new feature called total square footage, which is a sum of three other features, total basement square footage, first-floor square footage and the second floor square footage, so it's just some of all these three columns which makes sense because the total square footage of a house should be mentioned in one single column, but it's not in this case, so we add one column which just gives us a sum of the total square footage of the first floor, second floor and basement, so this is only one column that we are adding which depends on, or which is a combination of, these three columns by simple addition. Now before we apply this let's take a look at the feature importance is, we have written a function and added it to your utility functions it just prints the feature importances. This function just has that code that we were using to print feature importances in a single cell, previously we have just moved that code into this function. Now see there are additional features which were added as a result of one hot encoding which are listed in the order of importance and the ground living area is the most important one, let's add this one feature that we mentioned. All three scores have improved the OOB score the validation set MSE and the validation set score all three of em proof, so we are confident that we are going in the right direction. Now let's add additional features so let's see whatever new features to is doing, the new features to function adds an additional feature called overall grade of the house, and which is obtained by multiplying overall quality feature and the overall condition feature. So this is a categorical feature overall quality and overall condition is also categorical, we combine the two to obtain overall grade, this will reflect the overall grade of the house which is based on common sense, and we let the previous feature also there because we did improve our score as a result of adding it. Now as you can see our OOB score went up, our validation score went up and our MSE went down validation set, so we are actually making progress by adding each feature one by one. Now understate future importance is again, as you can observe total square footage as the highest importance, 0.6, overall great that we just had it is number two on importance, although it is much less than in terms of relative importance compared to total square footage, but it's still number two. So other two added features have actually popped up in importance as the highest ones. Now before going further with adding features we note one thing that this garage cars is a categorical variable, and it is being ranked pretty high by a random forest, so it's like number four on the list. So why don't we just convert it into categorical because right now is a number from the beginning. So we can do that by just passing it as a member of the list and where do cat calls with other members like central air, and the ones we added the year and months, we do that and just free when we adjust based from this. See we are still passing the add new features to just additional changes that we have added garish cars which was not there in the previous run, and we just continue with our feature importances. Now as you can see the garish cars when they are two in number and three in number, this is a result of one hot encoding. So the next thing we're going to add is total live area which is the sum of ground living area, plus garage area, plus lot area. As you can see there's a ground living area in plot area are pretty important and so is the garage area, so the total live area is expected to give us better performance. So let's see if it actually does not. This add new feature three they will be able to pass, now this is almost the same performance as before so we are by and large on the right track. Now the last thing we are going to try is the fourth version of this function. Here we are doing something interesting, we have created additional features from single feature. So for example the ground living area to is the square of the ground living area column ground living area cube is the cube of ground living area column, and the square SQ is same the square root of the ground living area so sometimes doing this type of mathematical transformation also helps. So we have selected two numerical columns, ground-living area which have been extracted from the importance list, and the garage area. So for these two columns we have added three more features square cube and square root respectively, and we're going to see if this makes any difference. So we got from point eight seven nine six two point eight eight on the validation set score, but other than that the MSE and the movie score pretty much remains the same. Now it seems that we have needed kind of a limit for the random forest model with these in our festa meters and the hybrid parameters, the default hybrid parameters basically. So in the next video we get to work on trying different models and also playing with the hyper parameters of our selected model, and see how we can improve the performance of our models after we are done with the feature Nehring. Now the thing about feature engineering is that you can try different combinations to see if any of them make any difference, and so you can add additional features, multiply them together, take the square or cube of an individual feature and so on, or any other type of mathematical transformation or interaction between features that you think makes sense, you can try it out. Feature engineering is a vast subject and there are a lot of different techniques based on statistical tests and so on, and we're not going to cover all of them in a short course but I encourage you to try out on your own. In the next video we are going to work on evaluating different models and also introduce ourselves to the concept of cross-validation.

Evaluating Different Models and Cross- Validation V33?

* Using multiple models on Iowa hosing dataset
* Evaluating models using k-fold cross-validation

The topic of this video is evaluating different models and cross-validation. We are going to cover using multiple models on Iowa housing dataset and we are also going to look at model evaluation using k-fold cross-validation. K-fold cross-validation is a technique where we split our training set into multiple splits, and one of the splits we set aside as a set and we iterate through the training set multiple times, like here for example in iteration 1, we have created 5 splits of our training set and the first one is a scientist test set and the rest 4 are considered as training set. We train on the training set and then we validate on the test set. So you can call it a validation set or test set, in iteration 2 we shift the test set to the next split, and the other 4 splits are considered to be training, and then the validation is performed on the second split, and so on in the third iteration it's the third split which is set aside as test and so on. So at the end of the fifth iteration all of our training set has been used for training as well as testing, each of the training example in this way we can evaluate how our model is performing, in our normal train test bed all we do is we just perform the Train desperate ones and if our test set has some peculiarities in the Trinity in the examples it might be the case that we might not generalize better, and our test set for example it contains samples which are easier to classify in some respect for our particular model. It gives us an idea of our models performance across all of the data. A variant of k-fold cross-validation, where k is basically the number of splits that we haven't to divide the data into, a variant of that is stratified k4 which is the same idea as we have been doing for a stratified train test fit, we want to have the same distribution of classes, this is mainly relevant to classification problems, so we have this first split containing the distribution of two classes that we have, equal to the other four classes. So each time we move the testing set to the next split make sure that the distribution of the classes stays the same in all the splits, so that the imbalance of the dataset in terms of classes is not present while validating. So this is a variant of the k-fold cross-validation. So in this Notebook we start again where we left off in the previous Notebook with all the feature engineering in place when we are calling our pre process D F function, and we perform a drain test plate. Now with our random forests regressor model we can pretty much the same results as before, we can run it again. Now we add two more models gradient boosting regressor with the same number of estimators which gives us better results then the random forest, so our MSE validation goes down from point zero one nine two point zero one five, and our score on validation goes up to point nine zero from 0.87 and there is no OOB spoor in radiant booster so we don't get that. A third model is called ElasticNet which is a combination of lasso and ridge regression, we covered last one education in Section three ElasticNet combines l1 and l2 regularization, and we pass a parameter called l1 ratio which is set to point nine in this case, and alpha parameter which is set to point zero one, we got these after a bit of experimentation with our dataset, and we get a better MSE on validation set of point zero one seven which is better than point zero one nine for random forests, and the score on validation set is point eight eight which is also a little bit better than random forest, but it's lesser than gradient boosting regressive. So out of the three models boosting regressor gives us the best performance. Now from sklearn model selection the import Crossville score function which is part of the same library as train test split that we have been using and another important function called K fold, which we will use for creating a special kind of split. For the random forest aggressor model we call it RF underscore model score CV and we pass it to the cross well score with our training set and the levels, and we pass another parameter called CV which is set to 5, this specifies the number of splits we are going to divide the data set into while performing cross-validation. So we get 5 scores as output, one for each iteration, and one of the prominent things about these 5 scores is that the first one is significantly lower for random forest model with n estimate is equal to 15 unit, this particular model on this data set the first split results in significantly lower score, whereas the other four are pretty much consistent. So we can interpret this as our model is not going to generalize 20% of the time with this average score oarsman's like that. So basically the first split is a bit difficult for this model, the nature of the training examples sense, for the gradient boosting case however again we have higher scores so point nine point eight the other four are close to 0.9, again the first one is significantly lower although it is higher than random forest, but lower than the other four iterations. For the ElasticNet first and the fifth one are pretty much the same, whereas the middle ones are a little bit higher. So one of the improvements we can do is to use the K fold function, we call its constructor, we specify the number of spreads, this is just going to create a split object which will be passed as a CV parameter to our cross val score, but the thing that we are doing is they're passing shuffle equal to true to this K fold constructor. What this will do is it will shuffle the data randomly before splitting, before each spread, over all our results should be improved by this change. We observe we get a significant increase in performance on the first plate from point eight one two point eight five, and the rest of this course are also reasonably consistent with the first one. So our k-fold split with shuffle equal to true had a positive effect on the random for us to get the performance. For the gradient boosting reversal again our first split performance went higher and it is now closer to the other four, so this also has better consistency in its performance due to the shuffle equal to true. In general the ElasticNet performance boosted up a little bit but there is still a little bit of inconsistency in ElasticNet, although it's better than before. So in this video we looked at the techniques for evaluating one instance of our model, in the next video we are going to look at model selection strategies, so we'll be creating or trying to vary the hyper parameters of our models, which is equivalent to creating multiple models or multiple instances with different parameters of a given model, and then we will be combining k-fold cross-validation with two strategies for model selection called grid search and random search.

# Model Selection Strategies v34

# Selecting best hyperparameter for a model on Iowa housing data.

# Grid search and randomized search for hyperparameter selection.

# Evaluating different models using K\_Fold cross-validation combined with hyperparameters selection.

# Recursively eliminating features to select the best model.

Model selection strategies, that is how to select the best model for a specific dataset. We are going to cover selecting best hyper parameters for a model for Iowa housing dataset and we will tune hyper parameters for different models, we will also cover grid search and randomized search which are strategies for hyper parameter selection, we will see how we can combine k-fold cross-validation with hyper parameter selection using grid search or random search, additionally we are going to cover another strategy for model selection which is called recursive feature elimination. So let's get started. Now the first thing we do after importing everything we load the data set into our DataFrame and we apply all the feature engineering that we have been doing in previous videos. So for example we are going to convert these into categorical columns, we are going to apply a robust scalar, we are going to bin certain columns that we have been doing previously, and we are also going to add the maximum number of features that we did in the previous video. So this is going to be a fully featured dataset that with all the feature Nina applied and everything. Now what we are going to be doing is given to do model selection and we are going to be tuning different hyper parameters using different techniques. Now if we ever try to tune our model, or shape our model, according to the performance that we get on the validation set that's going to affect the generalization of our model, because we are trying to then fit our model to the validation set, or you're going to change our model according to the validation set performance. We want to keep the test set totally aside so that we don't have any information leakage, we are going to respray the training set, so once we have this training set as recall recall the validation set start of validation set we call it the test set extraction via desk we do a trained Esprit. Now on the training set we split and this time we call our other partition the validation set, and so we again further reduced from 1314 to 1182 on the training set, the final training set that we're going to use, and 132 rows on the validation set. So this is how we divide the data into three tests training validation and tests, and we are not going to touch the test set for any type of model performance tuning, and we are finally going to test it on the test set, after we are done with the performance tuning and we have selected the best model. Now with this random forest regressor on this model as we've been doing before just within our first image is equal to 300 just to reopen our results. Again we apply a gradient boosting regressor with the same number of estimators as before, now when we apply ElasticNet with alpha equal to point zero one and L one ratio equal to 0.9 we get point zero one three MSE which is better than both random forests and gradient boosting, and we get a validation set score of 0.9 which is also better than both. Seems like our ElasticNet model is doing better than the played in boosting and random forests in this case. Now if we look at the random forest regressor model these are the hyper parameters that we can tune, for max features, max leaf nodes, minimum Beauty degrees and so on, and of course the number of estimators which is by default the pollutant. Similarly there are quite a few hyper parameters for gradient boosting regressor. Our goal now is to find the best combination of these hyper parameters for each model which will give us the best performance on our dataset, which is the Iowa housing data set. Now one approach to doing that is to use a brute force approach and what I mean by that is that we select the different parameters values, for example we say number of estimators 300 500 800 and so on, so these are six values which are basically arbitrarily chosen, the max features are 0.5 means 50% of the features for in 7.9 and auto means all the features, and minimum samples is two samples where we want to make the sec in to split the node when three samples are layer, when 10 samples are left and so on. And minimum samples which are required to call it a leaf are 1 3 and 10 respectively. Now let's take these four hyper parameters only there are others as well in the random forest regressor, but we are just taking these four four now. We assign different values and so we have these arrays and we create a dictionary where we have the hyper parameters name as the key and array of values as value. And we apply something called grid search CV which is combining grid search with cross-validation. What grid search does is it tries all combinations in a brute force way whatever we specify in the dictionary. So for example it will go with 300 0.5 2 and 1, then 300 point 5 2 and 3, 300 point 5 2 and 10, then 300 point 7 3 and 10 and so on, and while trying all combinations it will calculate the performance using cross-validation. So it's a really computationally intensive exhaustive technique which tries to find the best model, while for each instance of the model it chooses, meaning that each instance of the hyper parameter combination, creates a model instance and then it applies the model using cross-validation on the training set ,and then it chooses the best model out of that or the mean score of the cross-validation for that particular combination, and then it changes the combination and repeats. So it's really an exhaustive technique and that's why it takes a lot of time, and this is called a grid of values period of fiber parameters, so what we pass to the grid search CV function in scikit-learn is this as an argument, the first argument the model that we want to use, warm start equal to true just tells the model to pre use the previous values if we want to rerun it again and again, kind of reduces the time, so it doesn't have to Train from 0 every time, and number of jobs equal to 8, so I have 8 cores on my machine, and param grid is the parameter that is given the dicier of the parameters that are choose, and the number of jobs equal to 8 for this grid search CV. What this does is it tries to use all the cores or multiple cores, whatever I specify as number of jobs, while performing the grid search. So it's going to create different combinations and run them in parallel. Now what it returns is it's going to return a model which has all the predict and score and fit and everything, all those methods in with it, and we can just use it to fit and it will just fit the best model internally, which it found. Also we can time so for example I just noted the time before the start of my fitting and then also I subtracted the start time from the current time and divided by 60s, and after fitting we can score based on the models that we got on our validation set, and it took 78 minutes to run, we got 0.9 0 3 6 which is as you can see the MSE is also better and the validation set score is also better than our original model, so the grid search found us a better combination than our basic model with 1,500 trees. Now to get the report of the models that grid search found, I just took the report function which directly from scikit-learn documentation, this is a function that they have given, and we pass it this cross validation TV results field of the grid stretch model that we found, and we pass n top which is set to default as 3, so we want to look at the three top models. So we are going to get all the non 0 indices of the results the cv results using the flat non 0, and the results object has a key called rank test score, and which contains all the indexes of the models, so when it is equal to I, meaning that it is equal to the first rank models are ranked from 1 so when it is equal to 1, we get the candidates which are ranked to 1. So the horrid set might have found multiple models that it thinks should be ranked same rank, so we go through that list with that we have found and we just printed the mean test score on that list. So the mean test score is a standard deviation of the score is printed out from the results, for this particular candidate which is ranked one or two or three. Now when we call this function we get this result so these are the three models edit search found. So over rank one model has max features 0.7 minimum samples the one minimum samples split two and number of estimators equal to 300. Now one parameter that we could have set also is the scoring parameter, we will use it later on in our examples but for this we did not do that, but we can set the scoring parameters to be root-mean-square so that our model evaluation matches our actual model revelation that we are interested in, otherwise it will score the default scoring which is the R squared error. It's always a good idea to dump or to store this model as a pkl file using job lib.dump because for this video I did not obviously wait for 78 minutes, what I did was when I ran this model while recording the video I loaded the model back from disk, and got it into the same variable, and then I ran this report cell, so this is how I got it. So whenever you run long running tasks like grid search always try to save the results using joblib. Now another variation we can do is we can create the K folder split as we did before and we can then use the randomized search. This is the second technique that we are going to study. The randomized search is similar to grid search but much faster, the way it works is we specify the number of iterations, so this is the x is going to choose them different models, so it's going to choose like 20 different models. It's going to randomly sample from the distribution from the values we supply for different hyper parameters, for example in this case is just simple array, so it's going to randomly pick one from the array, and in the max features case it's going to be a random integer from 80 to 680 here, these numbers come from minimum of 80 features that we had originally, and we created 679 features in our dataset, if you remember by adding all those features and doing the feature engineering. And also applying one hot encoding and all. So after that and when we run it with the CV equal to K fold for the randomized CV we get a score of point nine zero and point zero one three which is pretty similar to the research case for gradient boosting reassesses time. And we again save it and then we reload it to generate the report and we get all these three models again. Similarly we can run randomized search for random forests together as well, similarly we have Elastic search which has multiple hyper parameters one of them is alpha, the l1 ratio which determines how much of l1 or l2 regularization we are going to apply, 0.9 means or 1 means that we are only going to apply l1 realization and Noel to, these values apply combination of l1 ratio and l2 ratio based on how much L 1 ratio is going to be. So for selection equal to cyclic and random, cyclic goes through all the features updating the coefficients in every iteration, random randomly selects features to update the coefficients. And when we run this with randomized CV we get a point 9 3 score on the validation set and a point 0 0 9 9, so our MSE goes further down. So looks like our ElasticNet is doing far better than the other two models on this dataset, and it's not even overfitting so as you can see in the case of ElasticNet its 0.92 score on the training set and 0.93 on the validation set. So you're not overfitting at all. Again we save and load again and then we do the ranking, it's mostly tilted towards l1 regularization and the alpha value is pretty much constant for all three, so point zero zero one seems to be a good value. So we create a model out of the hyper parameters of rank one yet again another model for random search elastic to for all the of model rank 2. Now if we apply this model the random sat ElasticNet one which is best or rank one on our test set, just finally on the 10% of the data that we set aside initially we get this performance point 0 1 - 2 which is again quite good. Now the last thing we want to study is called RFE. RFE, what it does is it recursively eliminates features, so first it does perform some feature analysis, calculates feature importance is like using PCA or other techniques, it ranks the features according to their importance, and then it starts eliminating features at the bottom of the list, so features which are not important it just eliminates them, calculates the models performance, eliminates other features, adds more features to eliminate till it finally gets to a point where the model performance stops improving, because of feature elimination, and then it starts putting features back. So this is also a very exhaustive and computationally intensive technique, it takes a lot of time. So RFE takes also the models that we haven't be used we did not have any other parameters for RFE, used the ElasticNet 1 model which was ranked one by our randomized search theory. So we got the RFE model and we just use it to evaluate. So in the next video we are going to apply these model selection strategies that we covered in this video on a classification task, which evaluates the credit risk for a financial institution.

RandomSearch RF:

MSE Training set = 0.00396983205076493, MSE Validation set = 0.02004034273698724, score Training Set = 0.9751957808419874, score on Validation Set = 0.8789535784327496

RandomSearch GB:

MSE Training set = 0.001540490180668893, MSE Validation set = 0.012579135646131103, score Training Set = 0.9903747424164416, score on Validation Set = 0.924020293646828

RandomSearch Elastic:

MSE Training set = 0.012659795133610232, MSE Validation set = 0.012284430576120135, score Training Set = 0.9208993405831606, score on Validation Set = 0.9258003527311827

RFE Elastic:

MSE Training set = 0.012659791759553351, MSE Validation set = 0.012284440103317461, score Training Set = 0.9208993616648701, score on Validation Set = 0.9258002951856028

In [ ]:

Without: df['TotalBath'] = df['BsmtFullBath'] + df['BsmtHalfBath'] + df['FullBath'] + df['HalfBath']

RandomSearch RF:

MSE Training set = 0.00396983205076493, MSE Validation set = 0.02004034273698724, score Training Set = 0.9751957808419874, score on Validation Set = 0.8789535784327496

RandomSearch GB:

MSE Training set = 0.001540490180668893, MSE Validation set = 0.012579135646131103, score Training Set = 0.9903747424164416, score on Validation Set = 0.924020293646828

RandomSearch Elastic:

MSE Training set = 0.012659795133610232, MSE Validation set = 0.012284430576120135, score Training Set = 0.9208993405831606, score on Validation Set = 0.9258003527311827

RFE Elastic:

MSE Training set = 0.012659791759553351, MSE Validation set = 0.012284440103317461, score Training Set = 0.9208993616648701, score on Validation Set = 0.9258002951856028

RandomSearch RF:

MSE Training set = 0.0036381698942017735, MSE Validation set = 0.019787731378900567, score Training Set = 0.9772680651886833, score on Validation Set = 0.8804793857178316

RandomSearch GB:

MSE Training set = 0.0008319160378759738, MSE Validation set = 0.014020953996315737, score Training Set = 0.994802040121429, score on Validation Set = 0.9153115128574789

RandomSearch Elastic:

MSE Training set = 0.012165762932362633, MSE Validation set = 0.012273048301029637, score Training Set = 0.923986141947591, score on Validation Set = 0.9258691032354569

RFE Elastic:

MSE Training set = 0.012640932442784322, MSE Validation set = 0.012311723476722877, score Training Set = 0.9210171980419076, score on Validation Set = 0.9256354998643676

In [1]:

<https://hk.saowen.com/a/948047ce2cef44b6ed29c5860d4dd46cd0f23470b1a234a6af2907fdae48f2fc>

RFE Elastic:without

New: Validation Set = Validation Set = 0.9256354998643676

Old: Validatiscore on Validation Set = 0.9258002951856028

MSE validation: Validation set = 0.012311723476722877

RandomSearch Elastic: with

def add\_new\_features5(df):

df['TotalSF'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']

df["OverallGrade"] = df["OverallQual"] \* df["OverallCond"]

df['TotalLivArea'] = df['GrLivArea'] + df['GarageArea']

df["GrLivArea-2"] = df["GrLivArea"] \*\* 2

df["GrLivArea-3"] = df["GrLivArea"] \*\* 3

df["GrLivArea-Sq"] = np.sqrt(df["GrLivArea"])

df["GarageArea-2"] = df["GarageArea"] \*\* 2

df["GarageArea-3"] = df["GarageArea"] \*\* 3

df["GarageArea-Sq"] = np.sqrt(df["GarageArea"])

df['TotalBath'] = df['BsmtFullBath'] + df['BsmtHalfBath'] + df['FullBath'] + df['HalfBath']

return df

|  |  |  |  |
| --- | --- | --- | --- |
|  | E1 | E2 | E3 |
| MSE train | 0.01216576 | 0.01304127 | 0.01304123 |
| MSE Valid | 0.0098082 | 0.0097837 | 0.00978493 |
| Score on Validation | 0.9307056 | 0.930878 | 0.9308700 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | E1 | E2 | E3 |
| MSE train | 0.01216576 | 0.01304127 | 0.01304123 |
| MSE Valid | 0.0098082 | 0.0097837 | 0.00978493 |
| Score on Validation | 0.9307056 | 0.930878 | 0.9308700 |

def add\_new\_features6(df):

df['TotalSF'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']

df["OverallGrade"] = df["OverallQual"] \* df["OverallCond"]

df['TotalLivArea'] = df['GrLivArea'] + df['GarageArea']

df["GrLivArea-2"] = df["GrLivArea"] \*\* 2

df["GrLivArea-3"] = df["GrLivArea"] \*\* 3

df["GrLivArea-Sq"] = np.sqrt(df["GrLivArea"])

df["GarageArea-2"] = df["GarageArea"] \*\* 2

df["GarageArea-3"] = df["GarageArea"] \*\* 3

df["GarageArea-Sq"] = np.sqrt(df["GarageArea"])

df['TotalBath'] = df['BsmtFullBath'] + df['BsmtHalfBath'] + df['FullBath'] + df['HalfBath']

df['LandscapeFactor']= df['LotShape'] \* df['LandContour']

return df

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ENET | ENET1 | ENET2 | ENET3 |
| MSE train | 0.01264093 | 0.012640931 | 0.0126409314 | 0.0121653044 |
| MSE Valid | 0.00973522631 | 0.009735226 | 0.009735226 | 0.009805426 |
| Score on Validation | 0.9312212 | 0.93122124 | 0.931221245 | 0.93072528 |

def add\_new\_features6(df):

df['TotalSF'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']

df["OverallGrade"] = df["OverallQual"] \* df["OverallCond"]

df['TotalLivArea'] = df['GrLivArea'] + df['GarageArea']

df["GrLivArea-2"] = df["GrLivArea"] \*\* 2

df["GrLivArea-3"] = df["GrLivArea"] \*\* 3

df["GrLivArea-Sq"] = np.sqrt(df["GrLivArea"])

df["GarageArea-2"] = df["GarageArea"] \*\* 2

df["GarageArea-3"] = df["GarageArea"] \*\* 3

df["GarageArea-Sq"] = np.sqrt(df["GarageArea"])

df['TotalBath'] = df['BsmtFullBath'] + df['BsmtHalfBath'] + df['FullBath'] + df['HalfBath']

df['LandscapeFactor']= df['LotShape'] \* df['LandContour']

df['BathroomRoomRatio'] = (df['FullBath'] + df['HalfBath'])/['BedroomAbvGr']

return df

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ENET | ENET1 | ENET2 | ENET3 |
| MSE train | 0.0126411909 | 0.0126405102 | 0.01264135075 | 0.0121653815 |
| MSE Valid | 0.009737526 | 0.0097324305 | 0.0097374058 | 0.009805343 |
| Score on Validation | 0.931204994 | 0.9312409976 | 0.9312058477 | 0.9307258755 |

Created simulation programs used in testing leading-edge computer chip designs, resulting in the first and only chip design that worked on first-pass hardware.

Created simulation using Simul8 to evaluate business processes including manufacturing, supply chain dynamics, call center analysis and logistics.

Created a simulation for Norske Skog to experiment with inventory levels, shipping modules, lead times and cost per metric tonne. This lead to evaluation to reduce costs and provide on-going decision support. This tool is still in use.

Designed a simulation for Mueller Industries to evaluate the delivery and storage requirements for their processing of scrap copper to the casting of that copper to copper tubes. The goal was to minimize inventory levels without exhausting the material needed for the casting facility.