Model Evaluation

Tells us how our model performs in the real world.

In-Sample evaluation does not tell us how the model can predict new data. It only tell us how the data predicts for existing data.

Solution:

Split into train and test data.

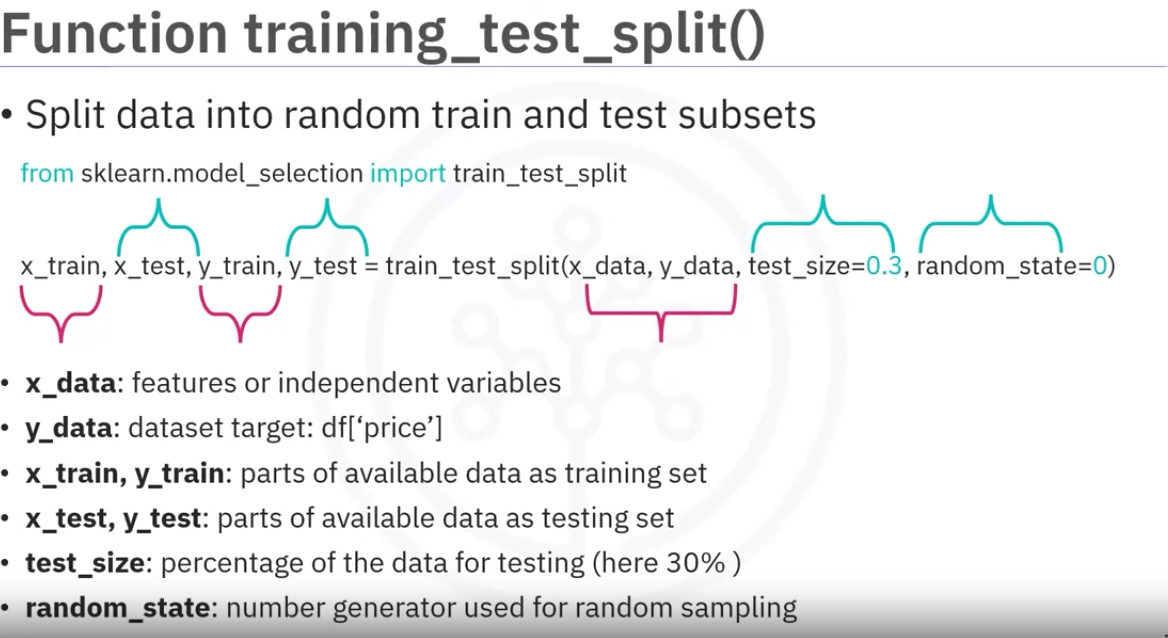
training 70%

testing 30%

WE train a model

Then asses the performance

When the testing is complete, then we use complete data to train the model.



Generalization performance:  
Generalization error is a measure of how well our data does at predicting previously unseen data.

The error when obtained using our testing data is an approximation of this error.



As we can see the predicted values are a lot closer to the actual values in the insample evaluation but with the train test split, the gap is larger. This is what happens in the real world environment.

Using a lot of data for training gives us an accurate means of determining how well our model will perform in the real world,

but the precision of the performance will be low.

Let's clarify this with an example. The center of this bull's eye represents the correct generalization error. Let's say we take a random sample of the data using 90% of the data for training and 10% for testing. The first time we experiment,

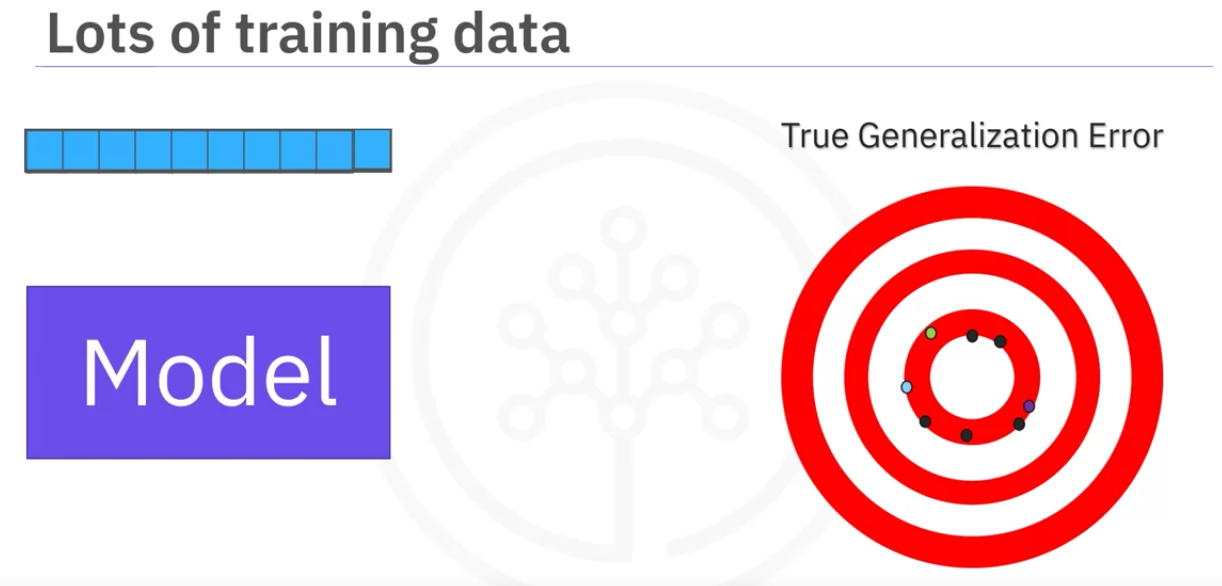
we get a good estimate of the training data. If we experiment again, training the model with a different combination of samples, we also get a good result, but the results will be

different relative to the first time we run the experiment.

Repeating the experiment again with a different combination of training and testing samples, the results are relatively

close to the generalization error, but distinct from each other. Repeating the process, we get good approximation of the generalization error, but the precision is poor, i.e.

all the results were extremely different from one another.





If we use fewer data points to train the model and more to test the model, the accuracy of the generalization performance will be less, but the model will have good precision. The figure above demonstrates this.

All our error estimates are relatively close together but they are further away from the true generalization performance. To overcome this problem, we use cross validation. One of the most common out-of-sample evaluation metrics is cross validation.

Cross Validation:

Most common out-of-sample evaluation metrics

Most effective use of data (each observation is used for both training and testing)

The data is divided into 4 folds. Each time when we train we pick three folds as training and 1 fold as test data.

For Example:

f1f2 f3 training data

f4 testing data

f1f2 f4 training data

f3 test data

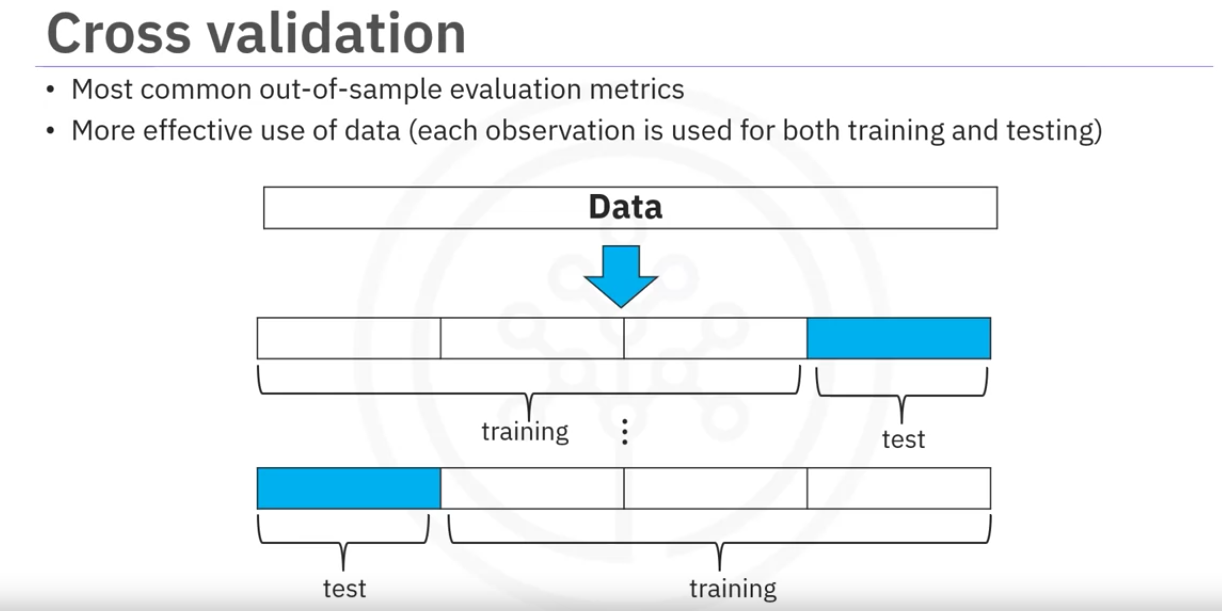
f1 f3 f4 training

f2 test

f2 f3 f4 training

f1 test.

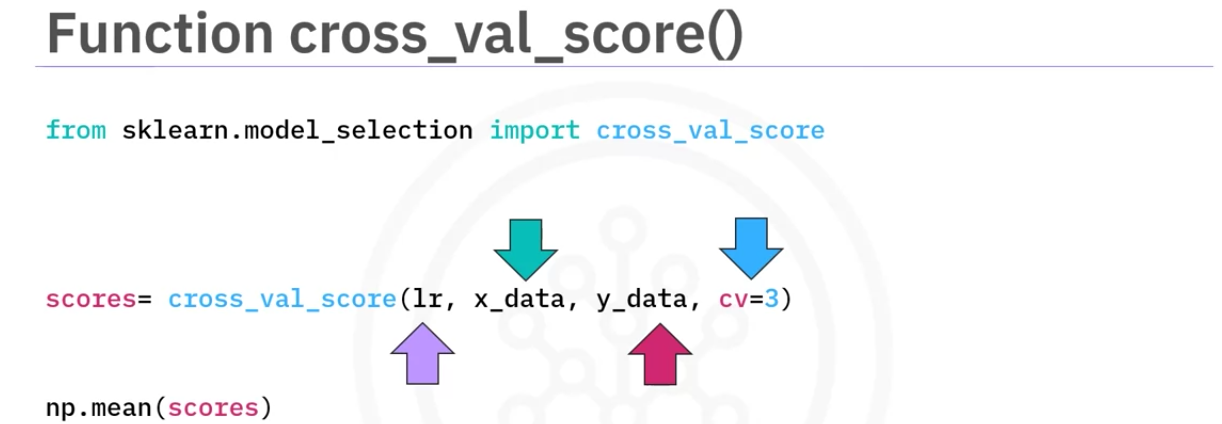
In this way we can have all the data for taining and also testing.



At then end we use the average results to estimate of out-of sample testing.

Applying cross validataion:

Using cross\_val\_score()



lr- model to be used. in this case linear regression object used to train

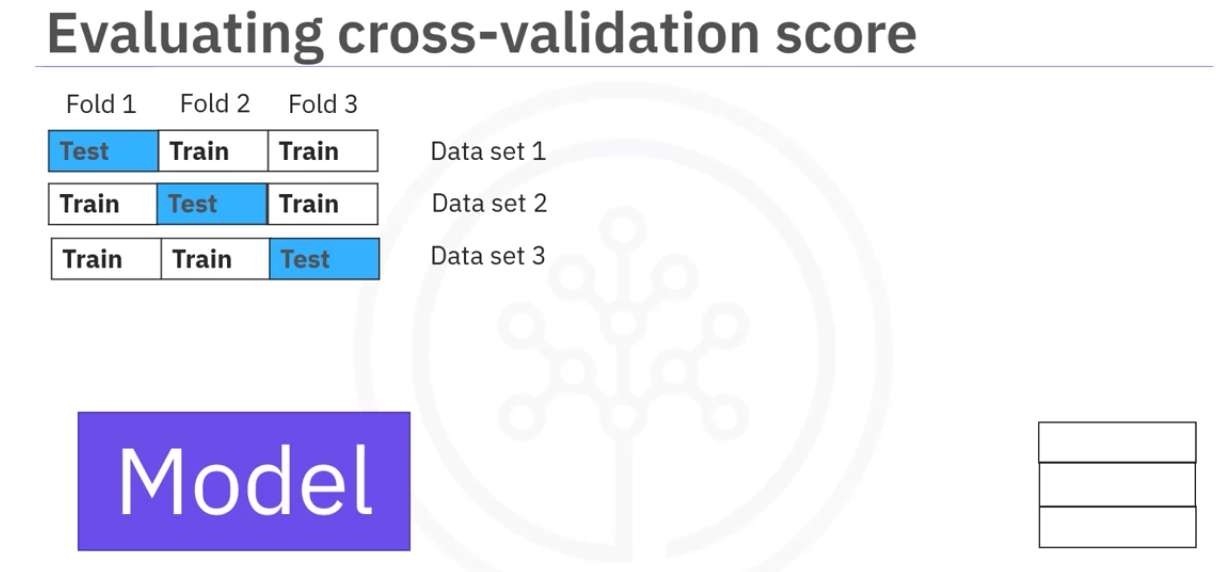
x\_data - independent variable

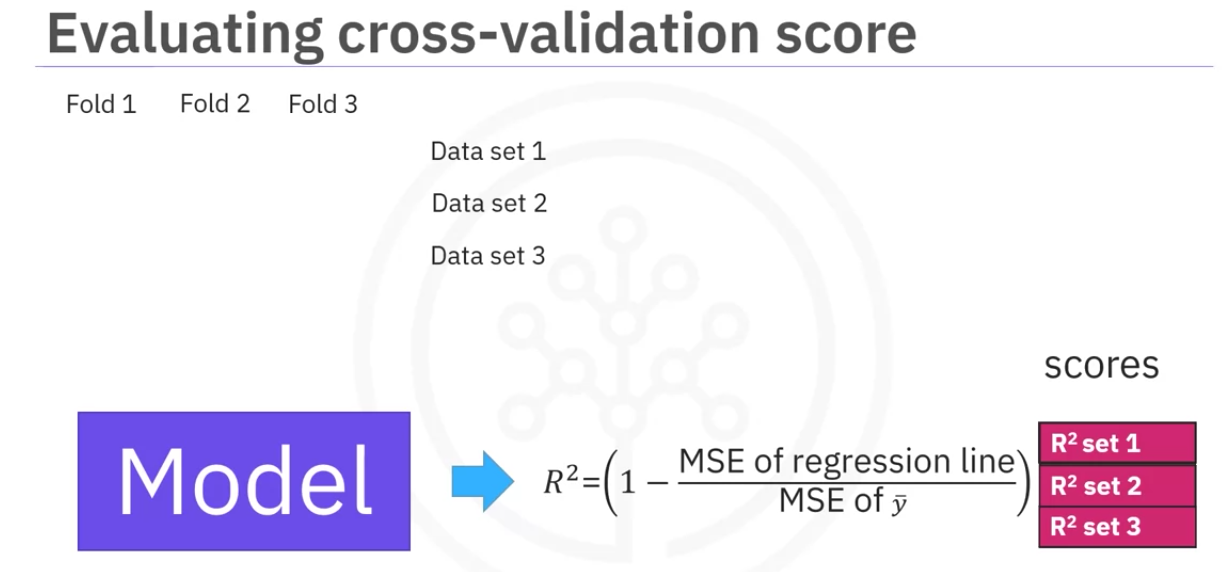
y\_data - dependent variable

cv-number of splits.

The result will be an numpy array of scores.one for each partition,

Then we can calculate mean.

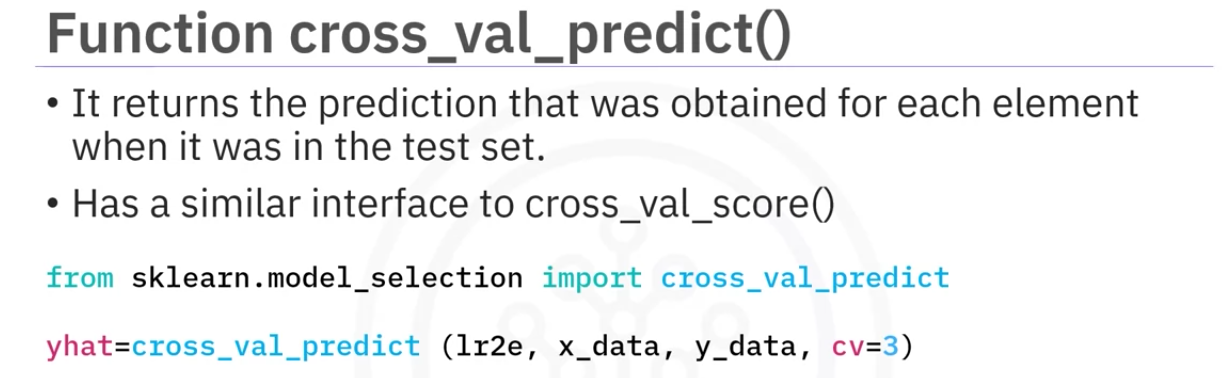


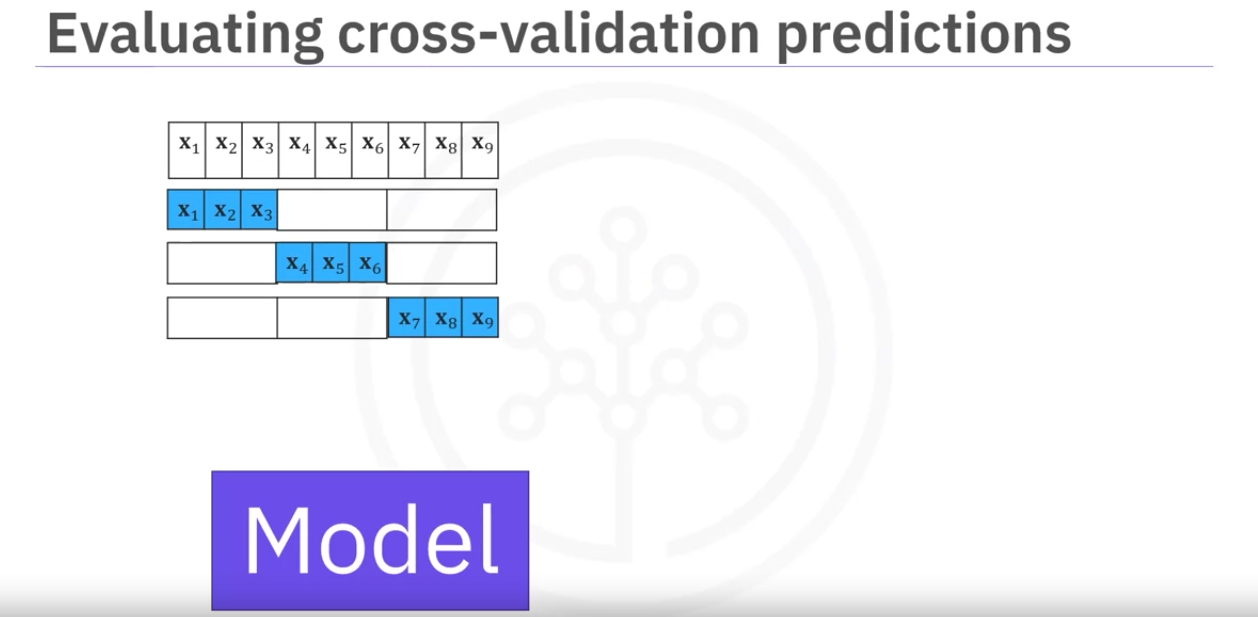


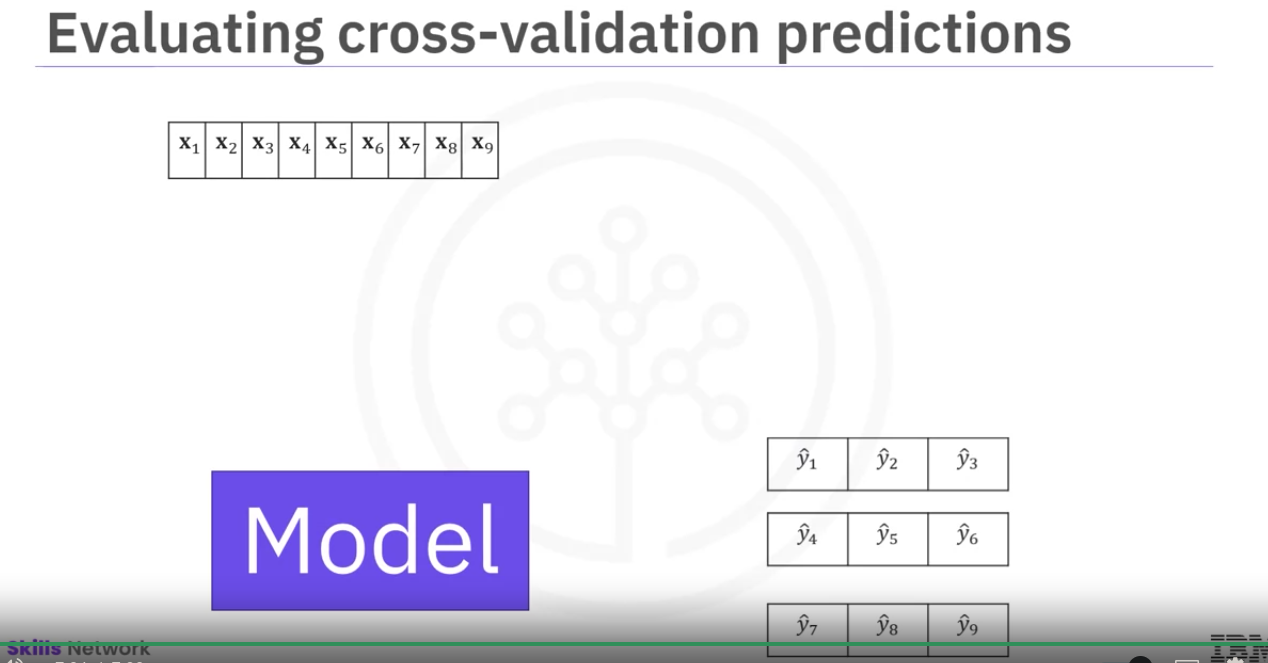
As we can see in the image data will be split into the folds then the data is trained and tested and r-sqared is recorded for each batch then the mean can be calculated from it,

To know the actual predicted values before the r-square is calculated, we need use th cross\_val\_predict()

It returns the prediction that was obtained for each element when it was in the test set.



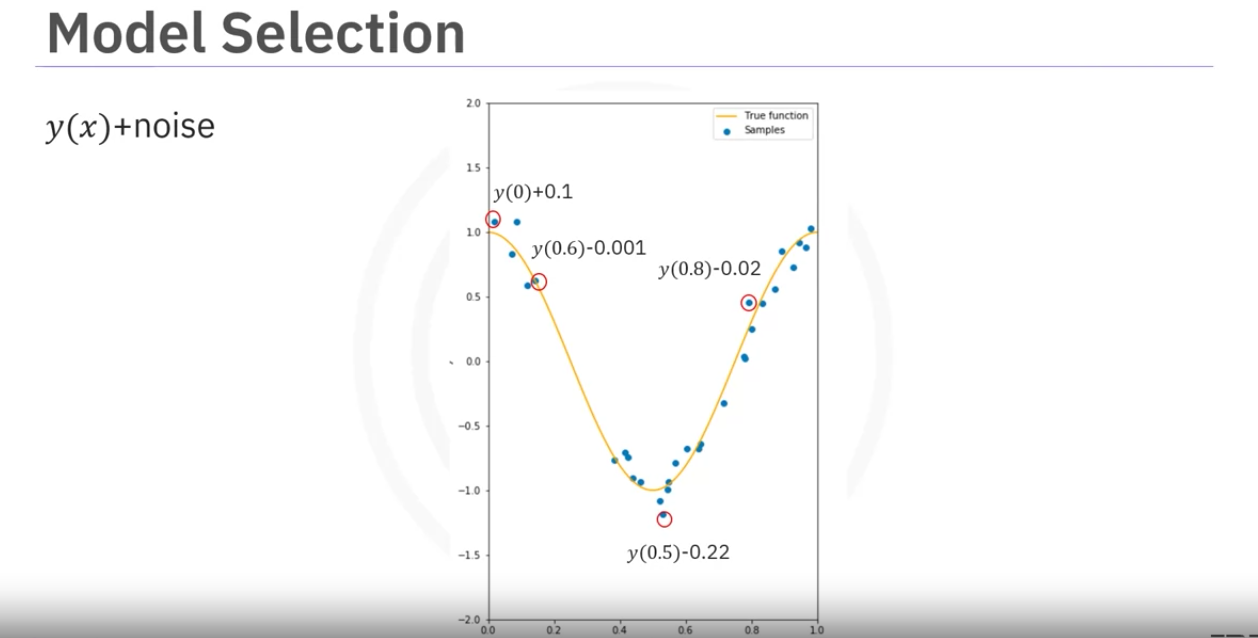




The split is done in a similar fashion. Then each partition is fed to the model and then predicton are found. The predicted values for each test will be loaded into an nd array.

Each prediction of a partition will have the same size as the test data for that partition.

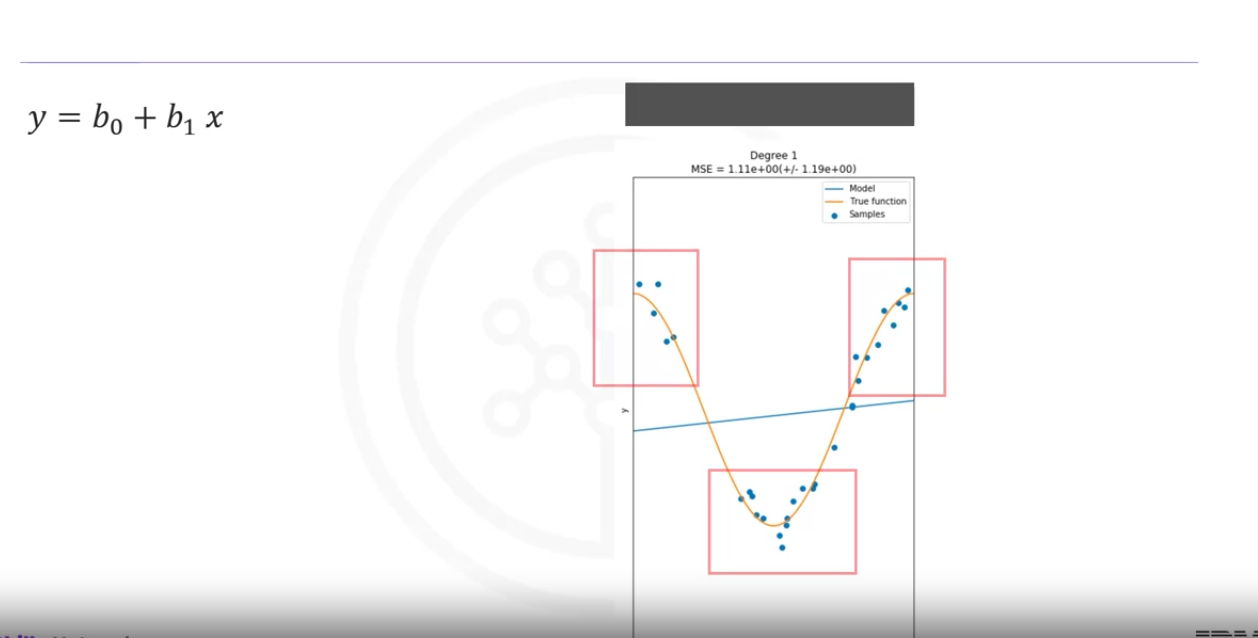
Overfitting , Underfitting



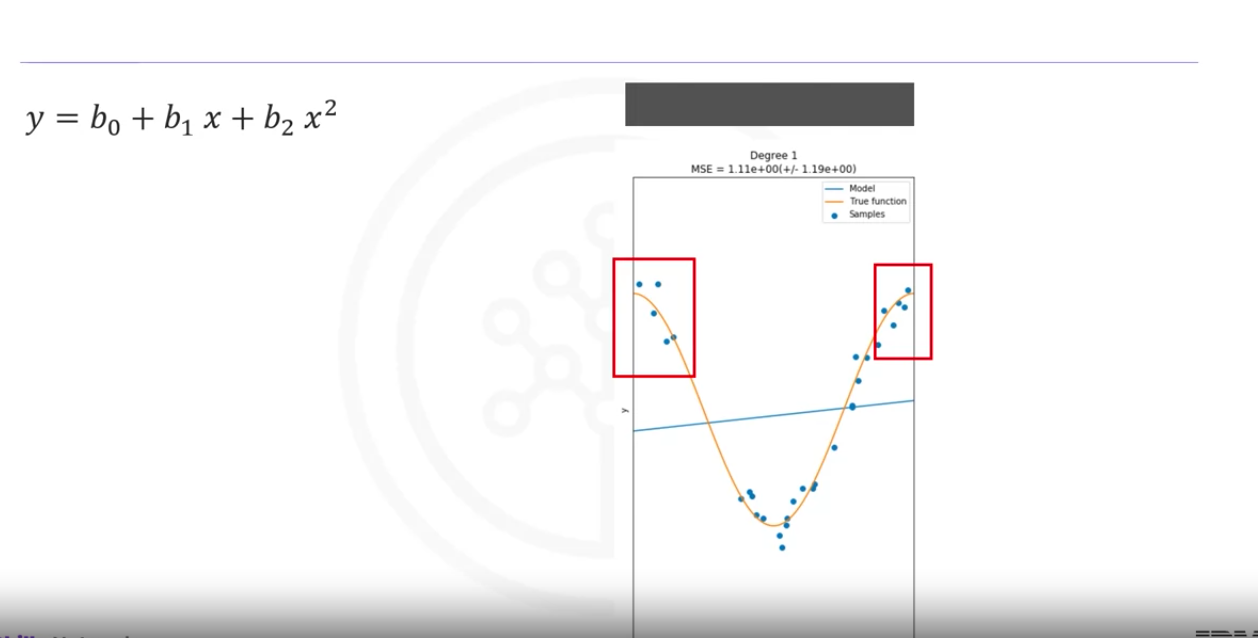
Consider the above polynomial function + some noise.

The goal of the model selection is to determine the order of best estimate of the function.

If we try to fit the function with linear function we the line is not complex enough to fit the data. So we get a lot of errors. This is called underfitting. We use very less anount of data to in this method to fit the model and the model will not perform very well.



if we increase the order the model is better bit stil its not flexible enough and it exibits underfitting.



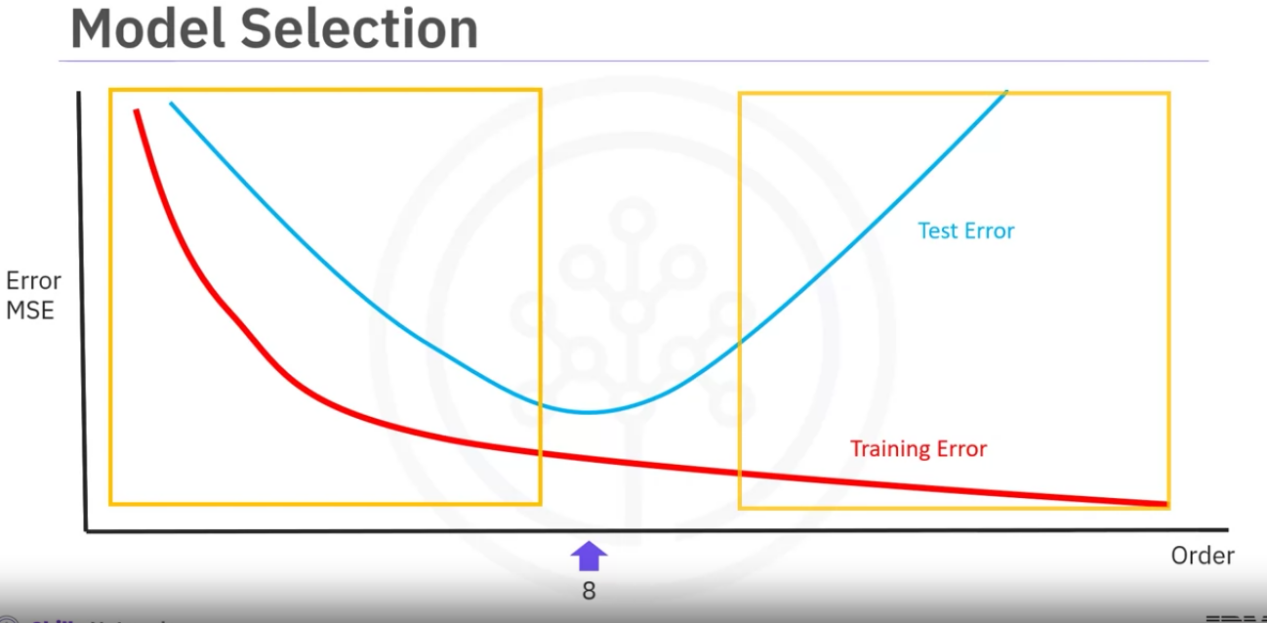
The following is and example of polynomial function with a degree 8 to fit the data.



WE see that the model does well predicting the data even at the inflextion points.

Increassing it to a degree 16, The model does extremely well at tracking the training points but it performs poorlyat estimating the function. (too much taining data. not much test data). This is called overfitting. The where the model is too flexible to fit the noise rather than the function.

Lets see the MSE for each polynomial order:



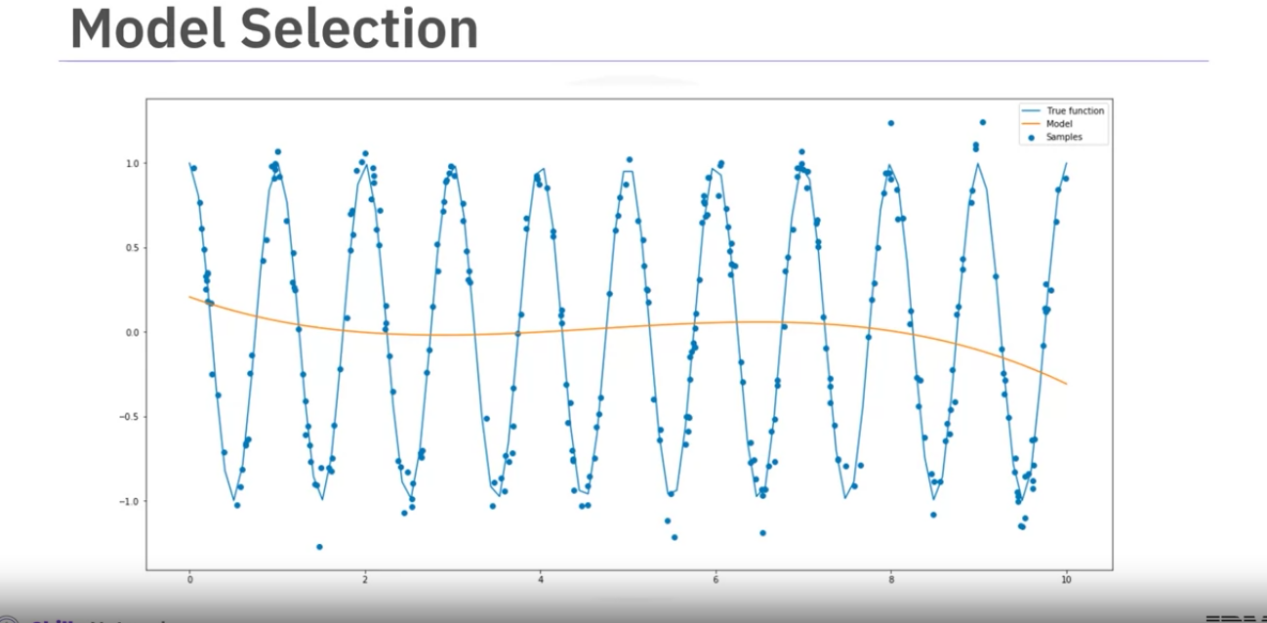
As we can see the training error comes down as the order increases.But the test error is lowest at in the middle which is considered the optimum fit.

The left of optimum fit is underfitting and the right of optimum fit is overfitting.

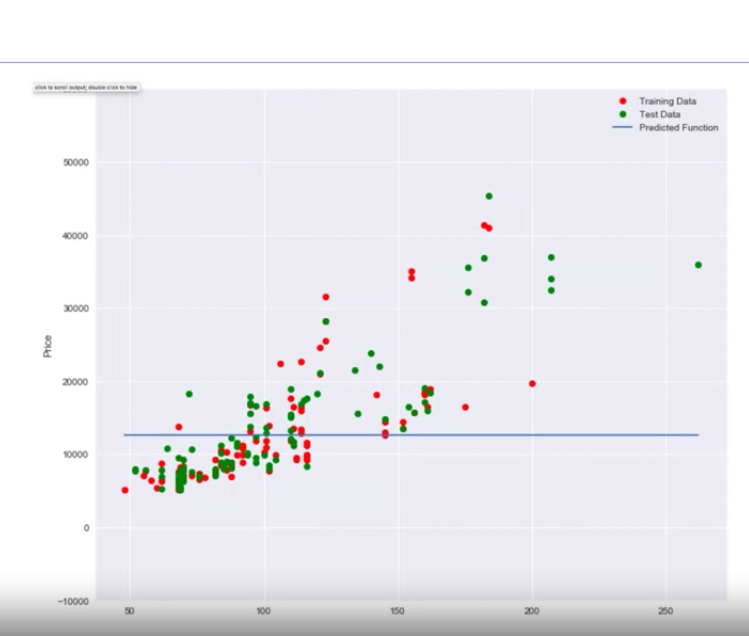
Even with the optimum fit there will be some errors. One reason for the errors could be because of the noise. The noise is random so we cannot predict it, This is sometimes called irreducible error.

Sometimes there are other sources of errors.For example our polynomial assumption is maybe wrong. Our sample maybe from a diffent function For Example in the following plot the data is generated from a sine wave not from a polynomial .

For real data the model maybe too difficult to fit or we may not have enough data to estimate fuction.

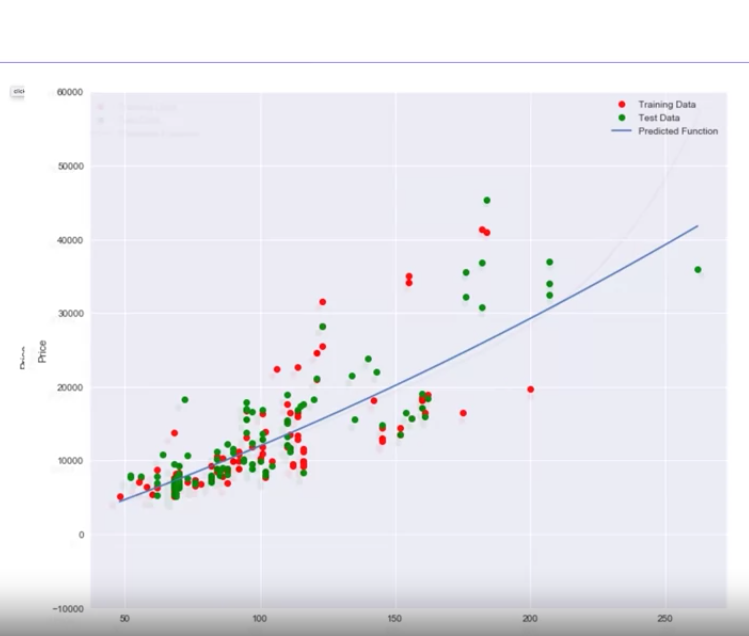


Lets use another polynomial using horsepower.





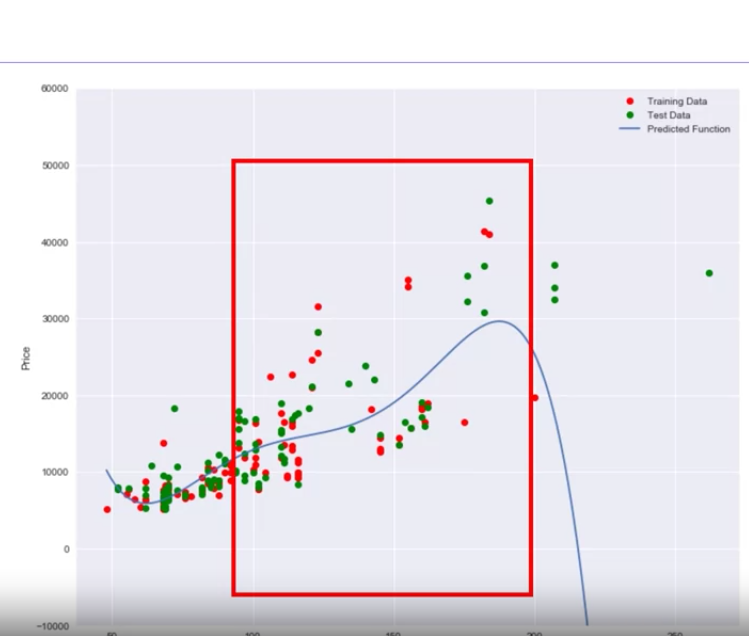
If we just use mean of the data, the model does not perform well. A linear function does fit the function better.



A 2nd order polynomial looks similar to the linear function.



A third order function also seems to increase like the previous two orders.



A fourth order polynomial has at around 200HP the price suddenly decreases.

This seems errorneous.

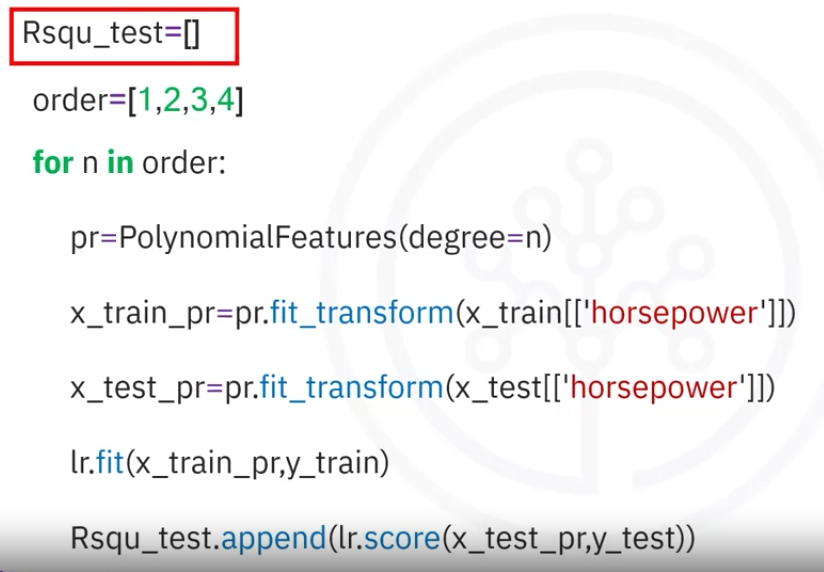
Lets looks at the r-squared for these models.



The closer the r-squred to 1 the better.

The optimal r-squared here is order 3 . The r-squred decreases drastically when the order is 4.

Calculating r-squared for differnt orders:



# Introduction to Ridge Regression

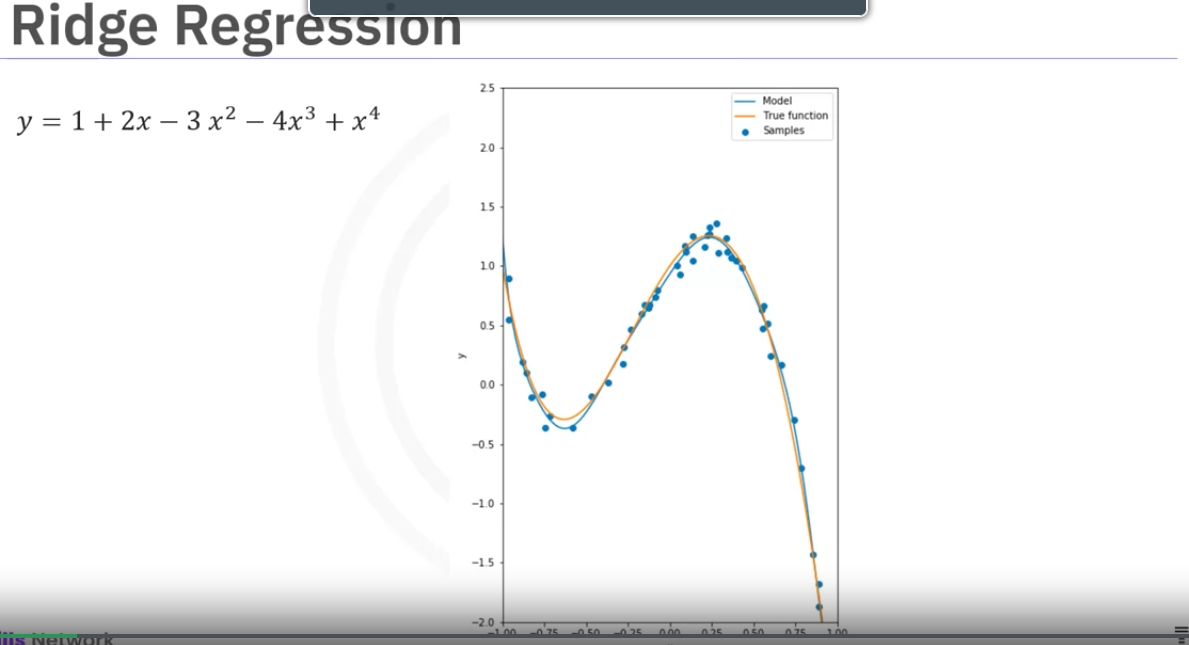
For models with multiple independent features and ones with polynomial feature extrapolation, it is common to have colinear combinations of features. Left unchecked, this multicollinearity of features can lead the model to overfit the training data. To control this, the feature sets are typically regularized using hyperparameters.

Ridge regression is the process of regularizing the feature set using the hyperparameter alpha. The upcoming video shows how Ridge regression can be utilized to regularize and reduce standard errors and avoid over-fitting while using a regression model.

Ridge Rigression

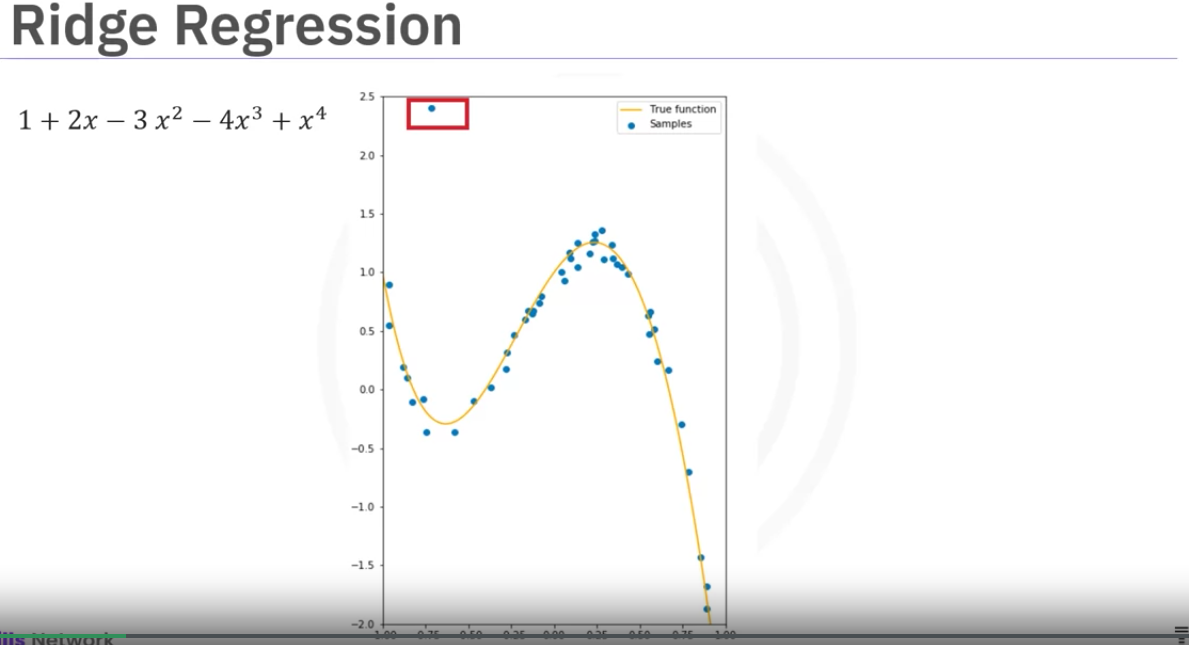
Ridge rigression prevents overfitting.

We will focus of polyniomial regression where over fitting is a huge problem where you have multiple independent variables or features.

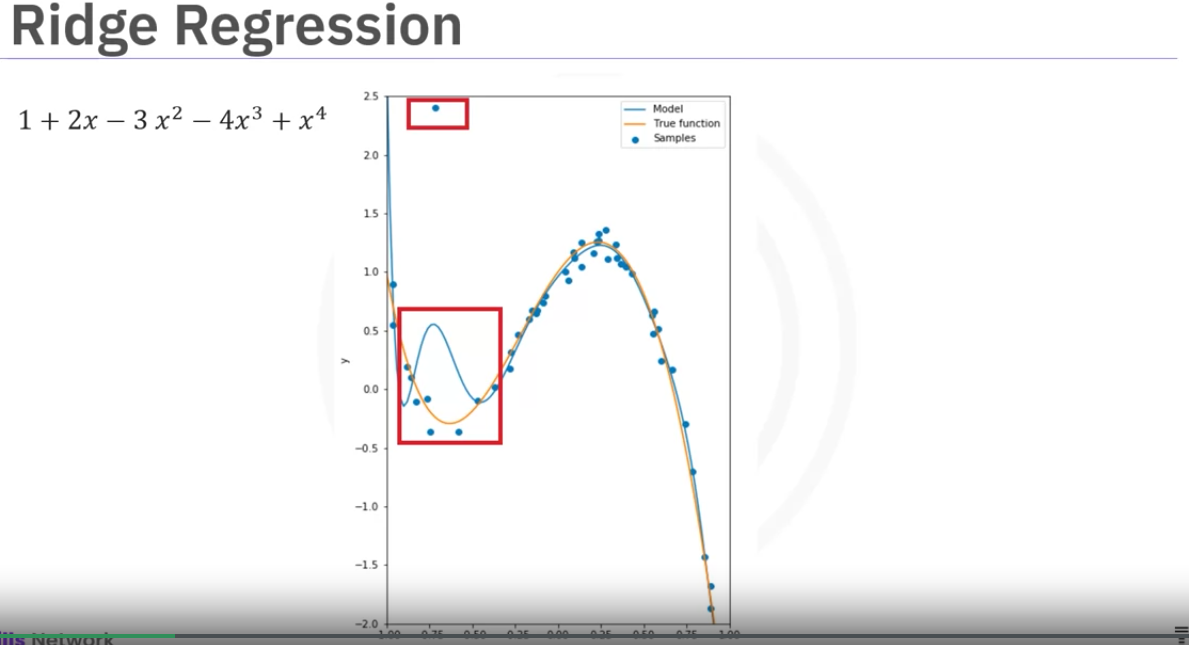


yellow line is the 4th order polynomial. Blue(the estimated function) line looks like it fits the true function.

In real world data there are some outliers.

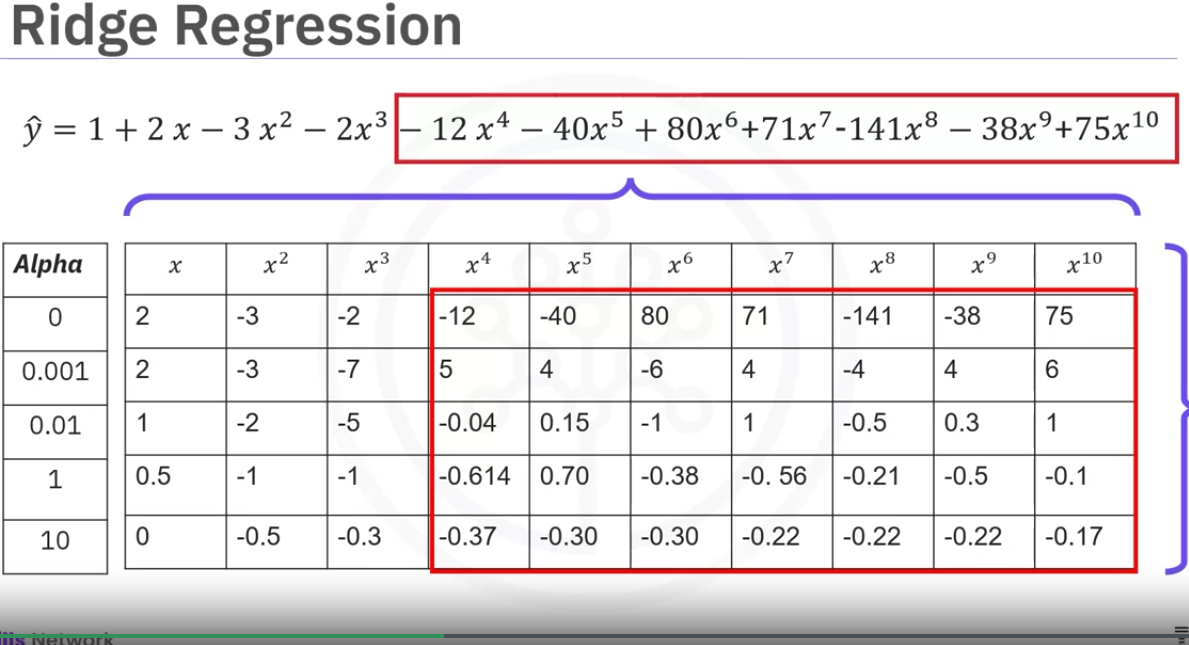


But we can try and fit a 10th order polynomial.



The estimated function in blue is incorrect,

Its not a good estimate of the actual function in orange.



when we look at the expression for the estimated function, we can see a lot of coefficients with large magnitude.

This is evident in very high order polynomials

This kind of shows that there is overfitting.

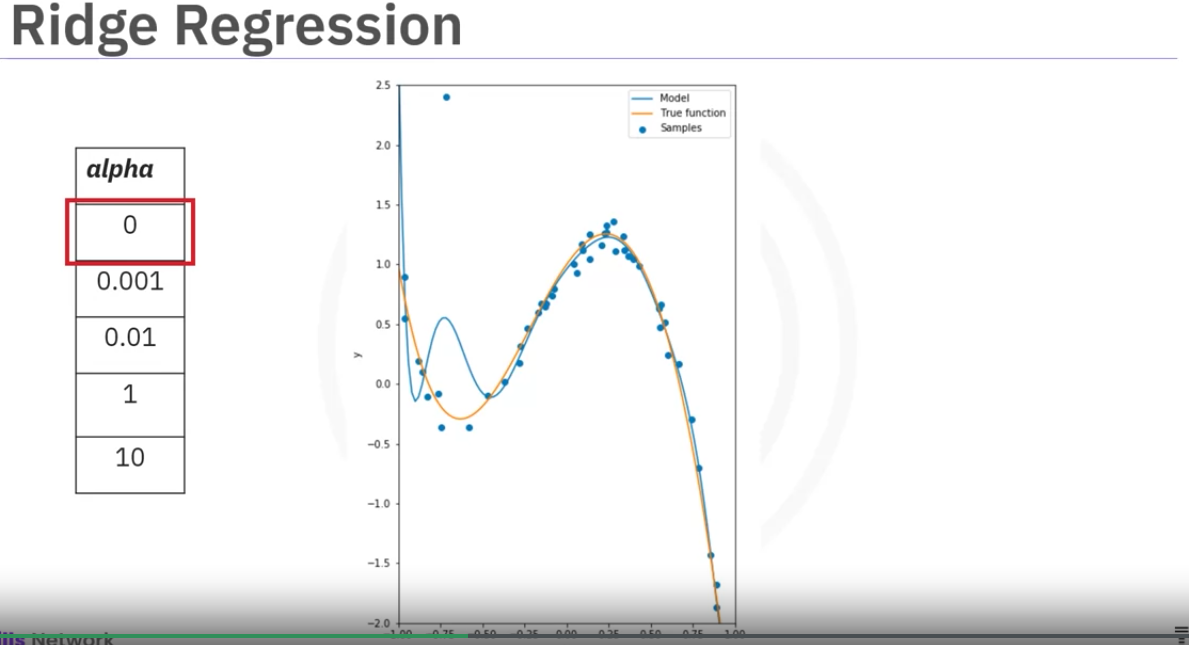
Ridge rigression controls the magnitude and therefore prevents us from overfitting. This by providing a parameter called alpha.We select alpha before fitting the model.

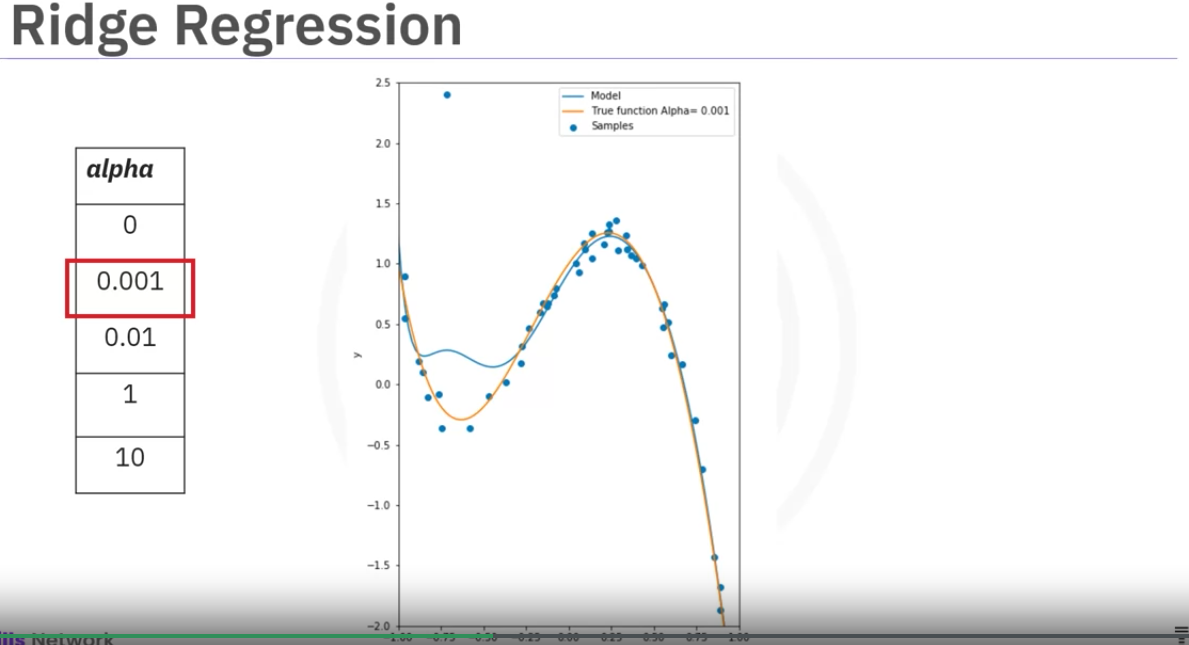
The table in the image shows that how the increasing alpha is changing the magnitudes.

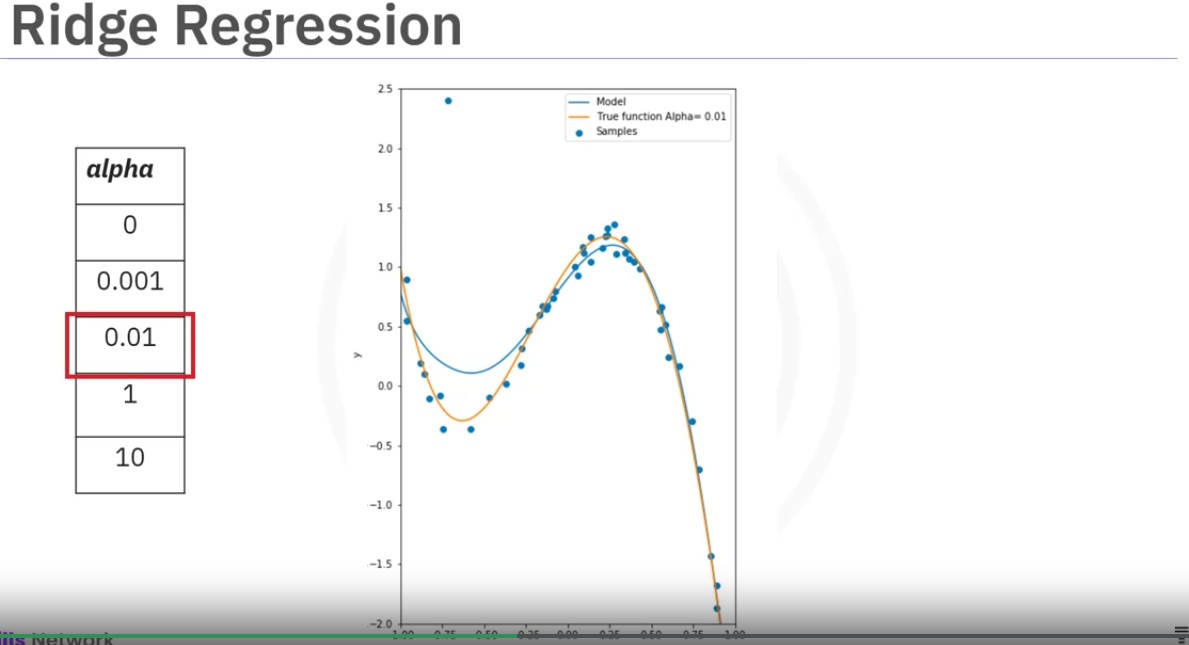
But alpha must be selected carefully.If the alpha is too large, the coefficients near 0 and will underfit the data.

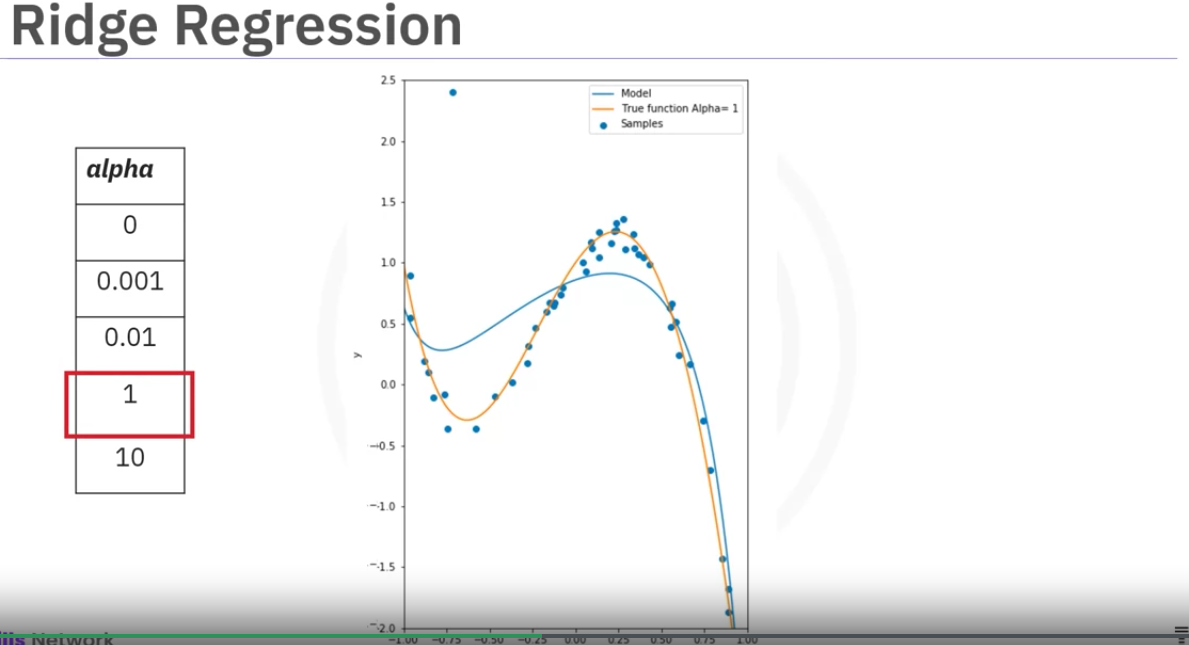
if the alpha is too small, the coefficients too large and data will overfit.

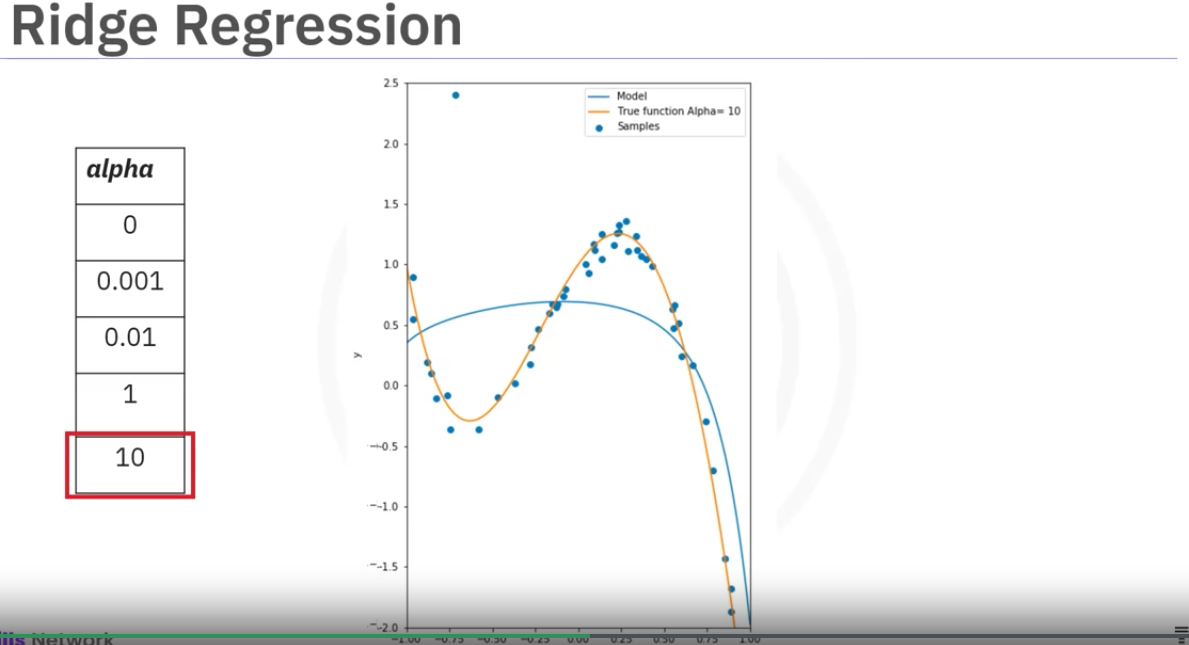
Graphs when using different alpha values:

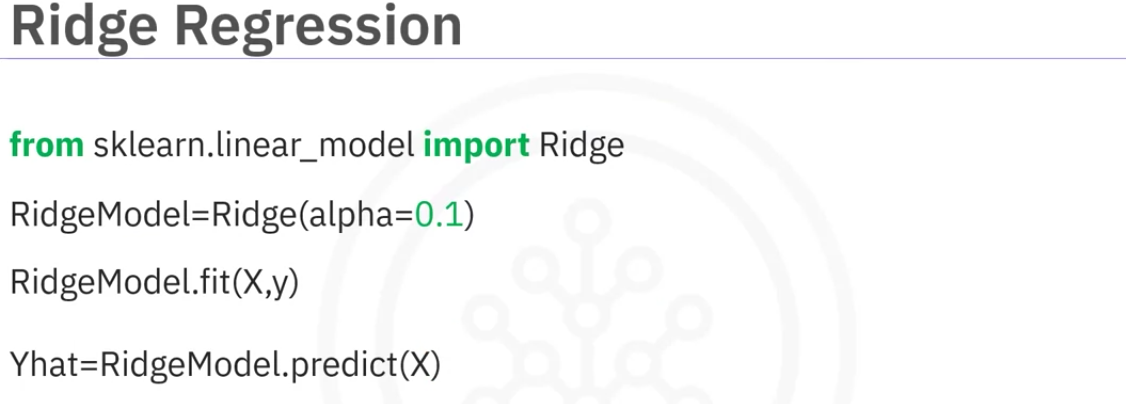




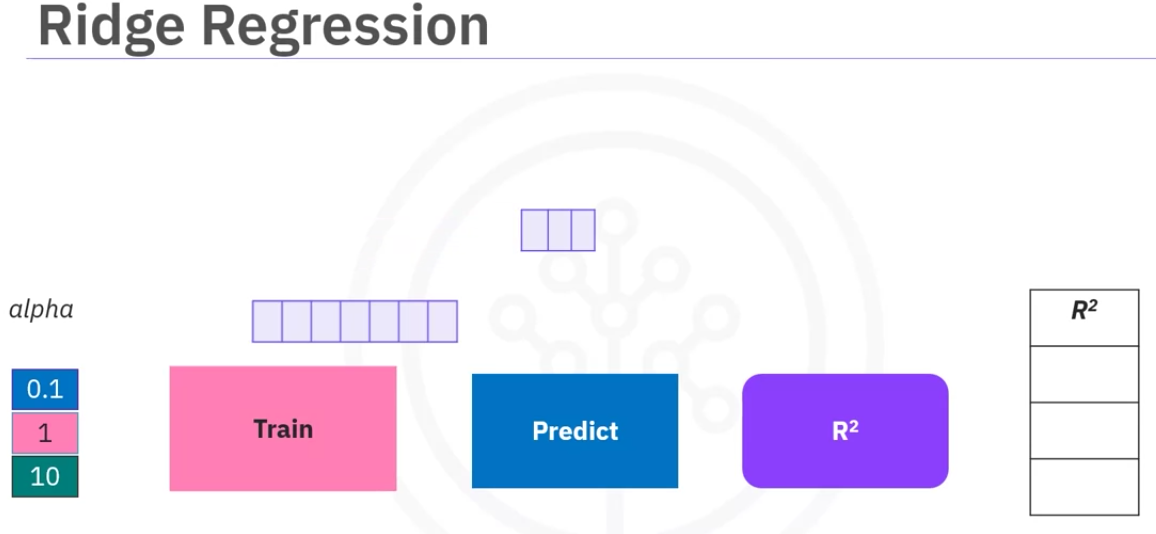








Note: The ridge model has its own score method similar to a LinearRegression model



In order to select alpha correctly, we use cross validation.

Using Ridge rigression.

We first use the training data to train the model. Then we use the second set called validation set for prediction. Validation data is similar to test data but its used to find hyper parameters like alpha.

Note: its very confusing to understand the test vs validation dataset. Watch the following video to understand better,

<https://www.youtube.com/watch?v=dSCFk168vmo>

<https://www.youtube.com/watch?v=XCYlRBf18YI>

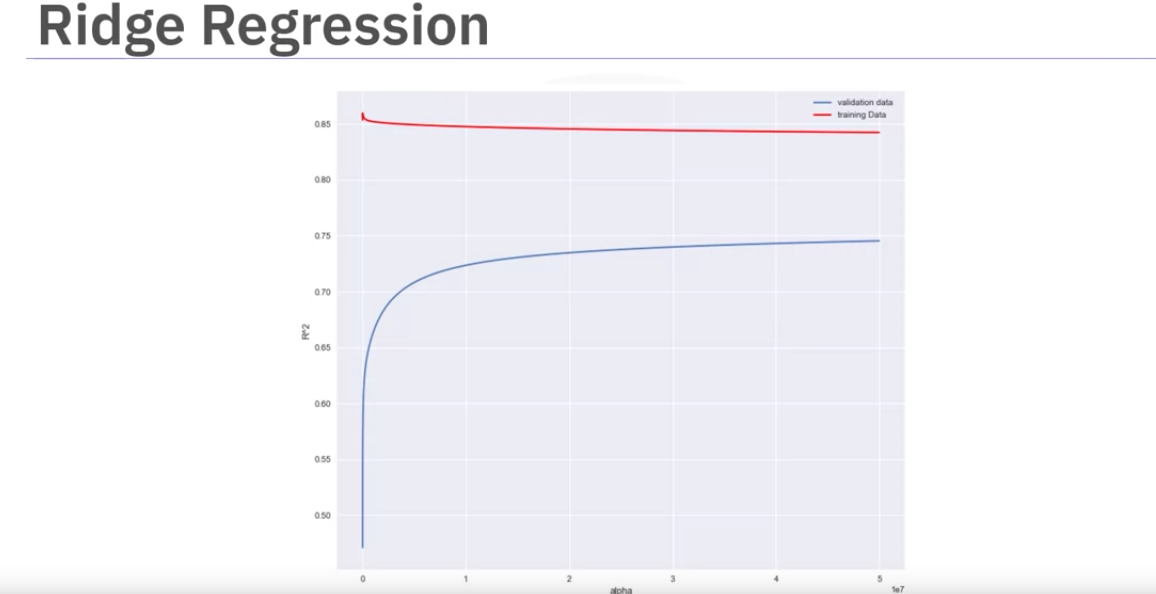
Also watch videos on ridge regression.

Test data is mostly in-sample data for which we know the results.

We use different alpha values and find the r-squared values,

Then we pick the alpha for which the r-squared is maximum.

We can also pick the values based on MSE aswell.



Overfitting is even worse if there are a lot of features for X.

The graphs shows the r-squared values on vertical axis and alpha values on the x axis,

WE use 2nd order polynomial on multiple features of car dataset in the above graph.

Red is the r-squared when the training data is used to test the model.

The blue line shows the r-squared values is calculated by using validation data.

As we can see the r-squred value as the alpha incerases the r-squared value increases for the validation data and converges approcimately at 0.75.

In this case we select the max value of alpha because running alpha with more higher values will have little impact.

Conversly, the r-squared values if the test data decreases. This is because the term alpha prevents overfitting.This may improve the results in the unseen data but the model has worse performance on the test data.

Grid Search

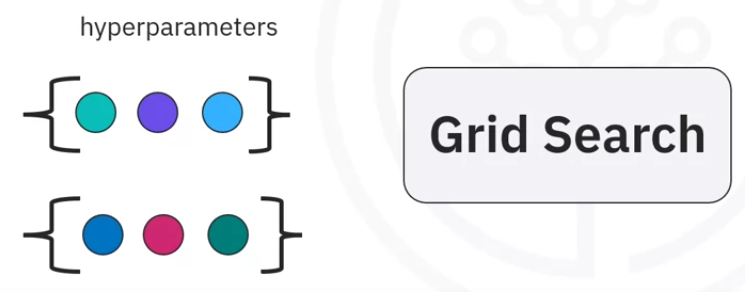
Allows us to seach through multiple free parameters with few lines of code.

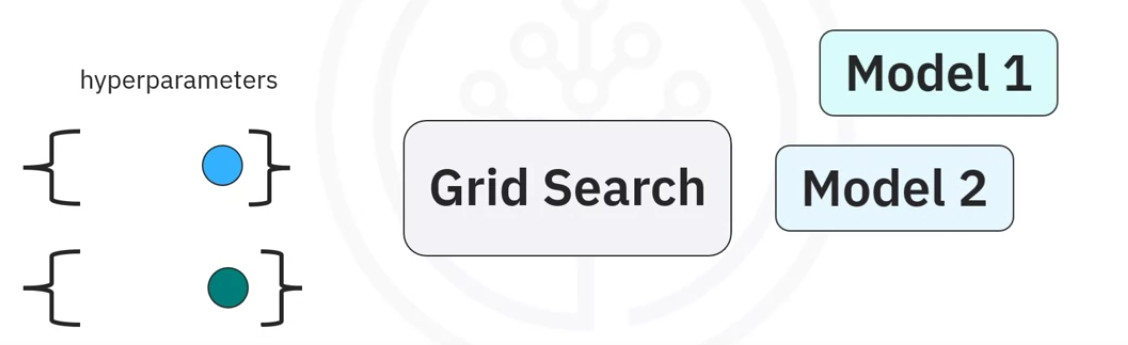
Parameters like alpha are not part of fitting or training processs. These values are called hyper parameters.

Scikit-learn has a way in which these parameters are iterated through using crossvalidation called Grid Search.

Gridsearch takes the model or objects we want to train and different values of the hyperparameters.

It then calculates the MSE or r-squared for various hyperparameter values, allowing you to choose the best values.



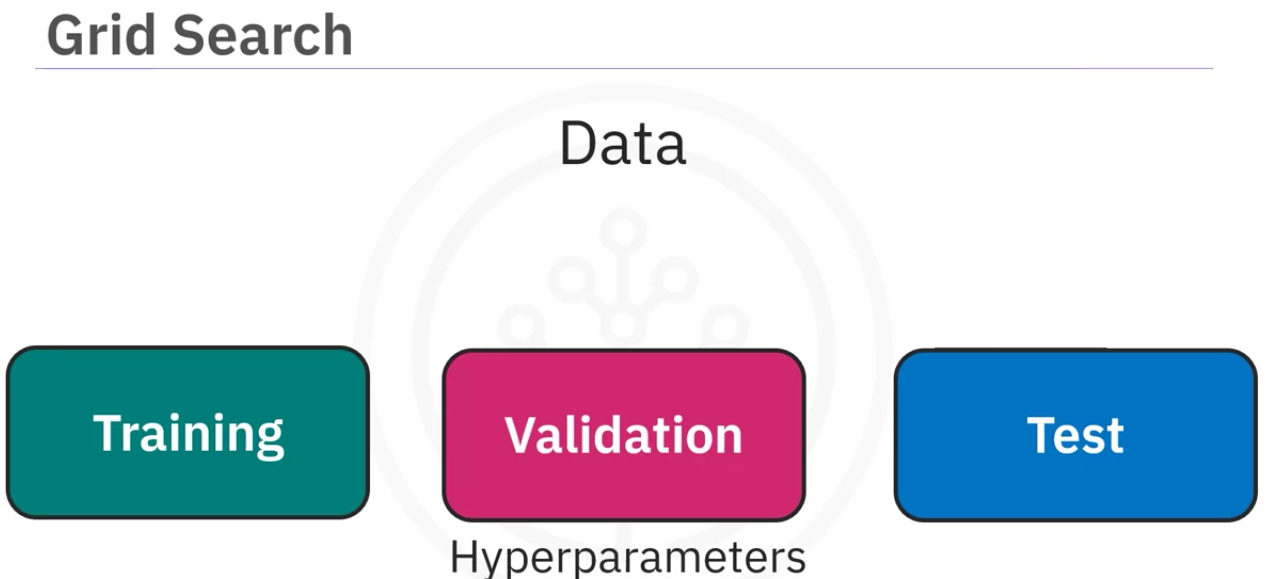


Grid Search selects each combination of the hyper parameters and creates a model for each combination.

Then MES and R-squares is calculated.

We select the hyper parameter that produces the smallest error.

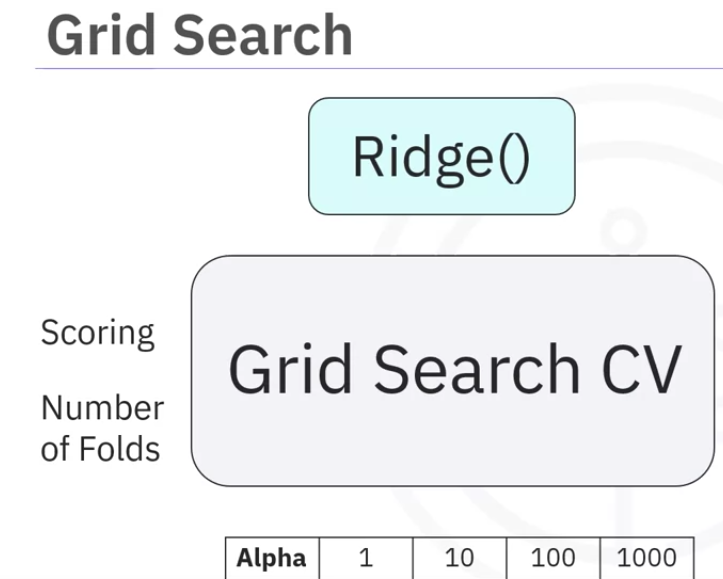
To select the hyper parameter we split the data into 3 set. Training set, validation set , test set,



After training the model we validate the model for the best hyperparameter values. Then we test the model.

For the Rigde regression we will focus on alpha and normalize hyper parameters.

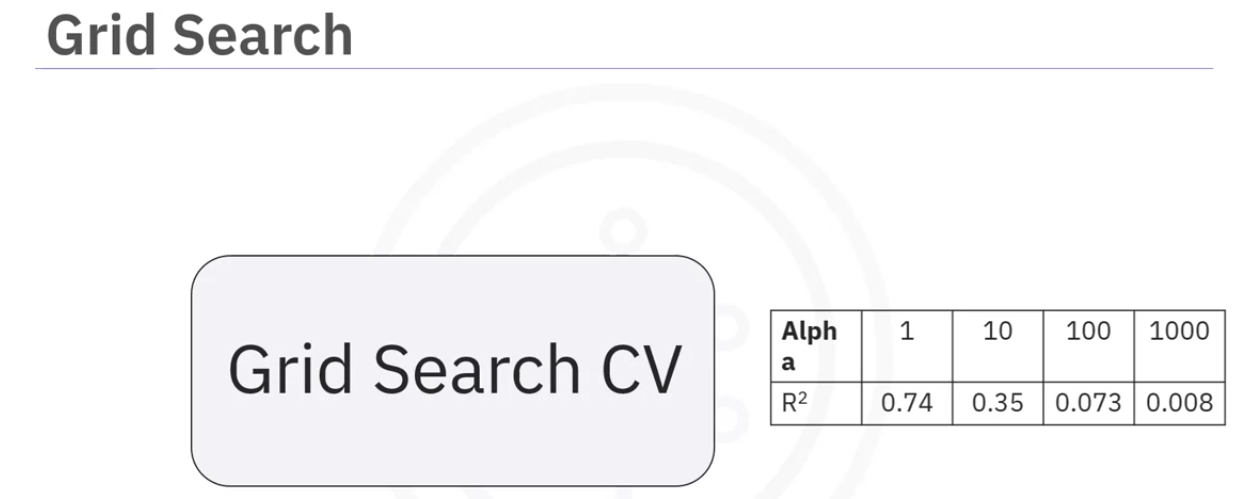
Grid search takes the model that we are going to use , scoring method(in this case , r-squared), number of folds and the hyper parameters.



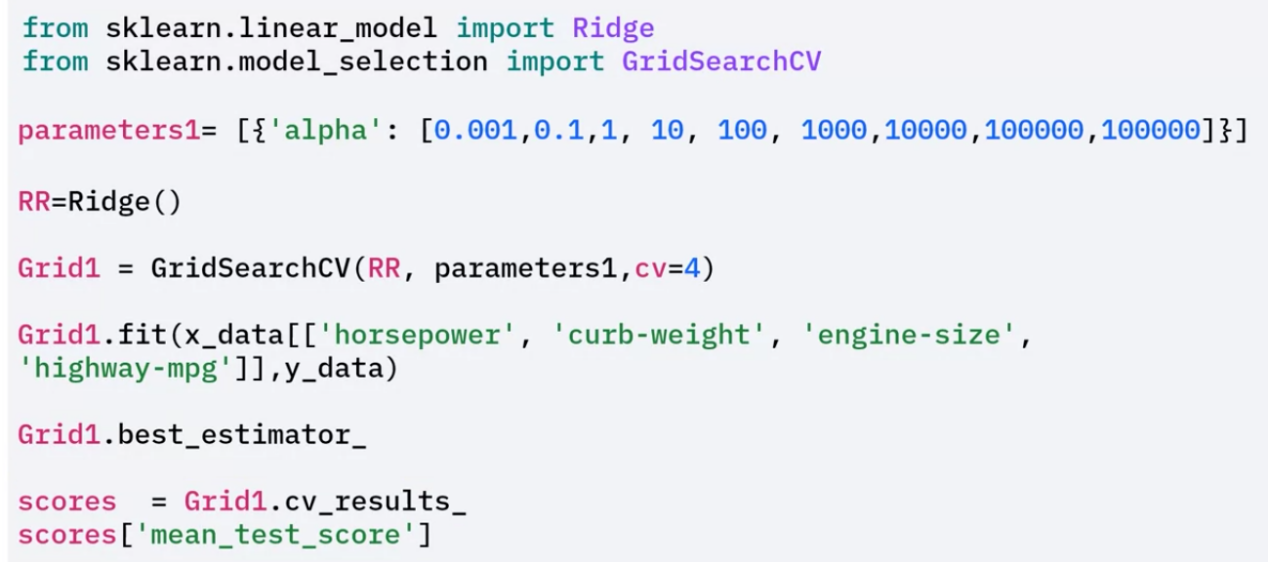
The hyperparameters are a list that contains dictionary with differnt values for that parameter.



The output object will hold the differnt r-squared values for each combination of hyper parameters.

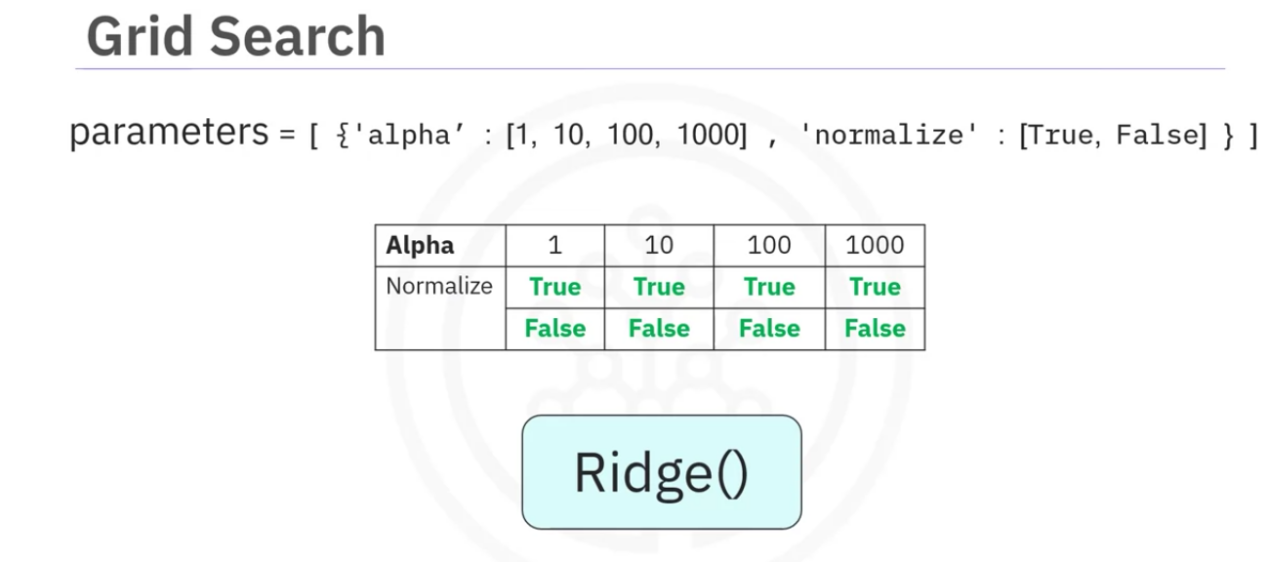


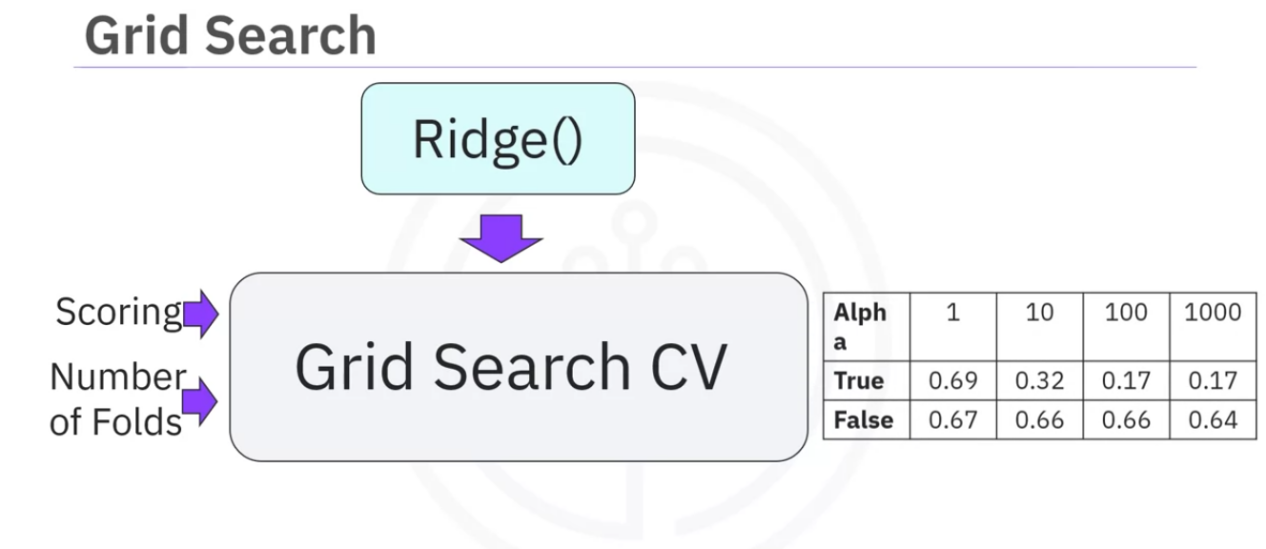
Usage:



we can use th best\_estimator\_ to get the best hyper parameter values and cv\_results\_ to get all details, and then we can look at info like MSE,

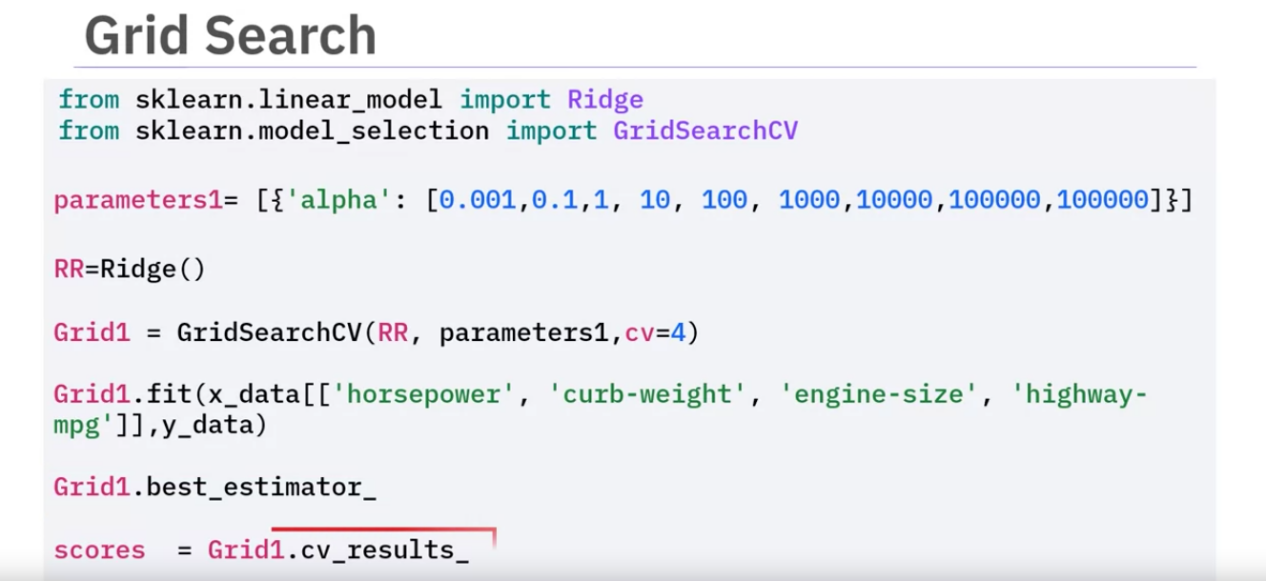
The following images show how we can setup multiple hyperparameters.





As we can see each combinataion of alpha and normalize paramters are taken and we find the r-squared values,

Code:

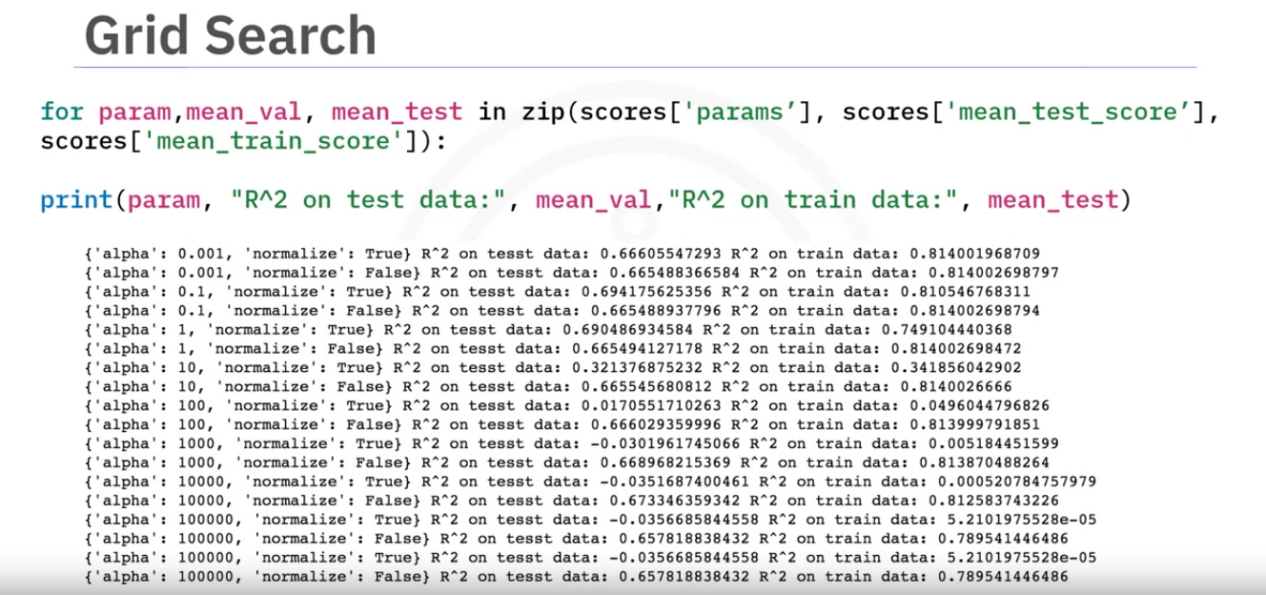


Note the data we use for the grid search fit method is not the split data. we need to use the whole data. Which is a little wrong. We need to use the train data for fitting and test data for checking the score.

Checkout this video for better understanding.

https://www.youtube.com/watch?v=ATnZmBxIvmQ

scores is dictionary that contains paramete, training score and test score.

We can display all the details like foloowing image,

# Lesson Summary

Congratulations! You have completed this lesson. At this point in the course, you know:

How to split your data using the train\_test\_split() method into training and test sets. You use the training set to train a model, discover possible predictive relationships, and then use the test set to test your model to evaluate its performance.

How to use the generalization error to measure how well your data does at predicting previously unseen data.

How to use cross-validation by splitting the data into folds where you use some of the folds as a training set, which we use to train the model, and the remaining parts are used as a test set, which we use to test the model. You iterate through the folds until you use each partition for training and testing. At the end, you average results as the estimate of out-of-sample error.

How to pick the best polynomial order and problems that arise when selecting the wrong order polynomial by analyzing models that underfit and overfit your data.

Select the best order of a polynomial to fit your data by minimizing the test error using a graph comparing the mean square error to the order of the fitted polynomials.

You should use ridge regression when there is a strong relationship among the independent variables.

That ridge regression prevents overfitting.

Ridge regression controls the magnitude of polynomial coefficients by introducing a hyperparameter, alpha.

To determine alpha, you divide your data into training  and validation data. Starting with a small value for alpha, you train the model, make a prediction using the validation data, then calculate the R-squared and store the values. You repeat the value for a larger value of alpha. You repeat the process for different alpha values, training the model, and making a prediction. You select the value of alpha that maximizes R-squared.

That grid search allows you to scan through multiple hyperparameters using the Scikit-learn library, which iterates over these parameters using cross-validation. Based on the results of the grid search method, you select optimum hyperparameter values.

The GridSearchCV() method takes in a dictionary as its argument where the key is the name of the hyperparameter, and the values are the hyperparameter values you wish to iterate over.