**Model Development**

In this video we will examine model development by trying to predict the price of a car using our dataset. In this module, you'll learn about simple and multiple linear regression, model evaluation using visualization, polynomial regression and pipelines, R-squared and MSE for in-sample evaluation, prediction and decision making, and how you can determine a fair value for a used car. A model or estimator can be thought of as a mathematical equation used to predict the value given one or more other values. Relating one or more independent variables or features to dependent variables. For example, you input a car model's highway miles per gallon as the independent variable or feature, the output of the model or dependent variable is the price. Usually, the more relevant data you have, the more accurate your model is. For example, you input multiple independent variables or features to your model. Therefore, your model may predict a more accurate price for the car. To understand why more data is important, consider the following situation. You have two almost identical cars. Pink cars sell for significantly less. You want to use your model to determine the price of two cars, one pink, one red. If your models independent variables or features do not include color, your model will predict the same price for cars that may sell for much less. In addition to getting more data, you can try different types of models. In this course, you will learn about simple linear regression, multiple linear regression and polynomial regression.

**Linear Regression and Multiple Linear Regression**

In this video, we'll be talking about simple linear regression and multiple linear regression. Linear regression will refer to one independent variable to make a prediction. Multiple linear regression will refer to multiple independent variables to make a prediction. Simple linear regression or SLR is a method to help us understand the relationship between two variables. The predictor independent variable x and the target dependent variable y. We would like to come up with a linear relationship between the variables shown here. The parameter b\_0 is the intercept. The parameter b\_1 is the slope. When we fit or train the model, we will come up with these parameters. This step requires lots of math, so we will not focus on this part. Let's clarify the prediction step. It's hard to figure out how much a car costs, but the highway miles per gallon is in the owner's manual. If we assume there is a linear relationship between these variables, we can use this relationship to formulate a model to determine the price of the car. If the highway miles per gallon is 20, we can input this value into the model to obtain a prediction of $22,000. In order to determine the line, we take data points from our data set marked in red here. We then use these training points to fit our model. The results of the training points are the parameters. We usually store the data points into data frame or numpy arrays. The value we would like to predict is called the target that we store in the array y. We store the independent variable in the data frame or array x. Each sample corresponds to a different row in each data frame or array. In many cases, many factors influence how much people pay for a car. For example, make or how old the car is. In this model, this uncertainty is taken into account by assuming a small random value is added to the point on the line. This is called noise. The figure on the left shows the distribution of the noise. The vertical axis shows the value added and the horizontal axis illustrates the probability that the value will be added. Usually a small positive value is added or a small negative value. Sometimes, large values are added. But for the most part, the values added are near zero. We can summarize the process like this. We have a set of training points. We use these training points to fit or train the model and get parameters. We then use these parameters in the model. We now have a model. We use the hat on the y to denote the model is an estimate. We can use this model to predict values that we haven't seen. For example, we have no car with 20 highway miles per gallon. We can use our model to make a prediction for the price of this car. But don't forget our model is not always correct. We can see this by comparing the predicted value to the actual value. We have a sample for 10 highway miles per gallon but the predictive value does not match the actual value. If the linear assumption is correct, this error is due to the noise. But there can be other reasons. To fit the model in Python, first we import linear model from sklearn then create a linear regression object using the constructor. We define the predictor variable and target variable then use the method fit to fit the model and find the parameters b\_0 and b\_1. The input are the features and the targets. We can obtain prediction using the method predict. The output is an array. The array has the same number of samples as the input x. The intercept b\_0 is an attribute of the object lm. The slope b\_1 is also an attribute of the object lm. The relationship between price and highway miles per gallon is given by this equation in bold, price equals 38,423.31 minus 821.73 times highway miles per gallon, like the equation we discussed before. Multiple linear regression is used to explain the relationship between one continuous target y variable and two or more predictor x variables. If we have for example 4 predictor variables then b\_0 intercept x equal zero b \_1 the coefficient or parameter of x\_1, b\_2 the coefficient of parameter x\_2 and so on. If there are only two variables then we can visualize the values. Consider the following function: The variables x\_1 and x\_2 can be visualized on a 2D plane. Let's do an example on the next slide. The table contains different values of the predictor variables x\_1 and x\_2. The position of each point is placed on the 2D plane color-coded accordingly. Each value of the predictor variables x\_1 and x\_2 will be mapped to a new value y, y hat. The new values of y y hat are mapped in the vertical direction with height proportional to the value that y hat takes. We can fit the multiple linear regression as follows. We can extract the four predictor variables and store them in the variable z then train the model as before using the method train with the features or dependent variables and the targets colon. We can also obtain a prediction using the method predict. In this case, the input is an array or data frame with four columns. The number of rows corresponds to the number of samples. The output is an array with the same number of elements as number of samples. The intercept is an attribute of the object and the coefficients are also attributes. It is helpful to visualize the equation, replacing the dependent variable names with actual names. This is identical to the form we discussed earlier.

**Model Evaluation using Visualization**

In this video, we'll look at Model Evaluation using Visualization. Regression plots are a good estimate of the relationship between two variables, the strength of the correlation, and the direction of the relationship (positive or negative). The horizontal axis is the independent variable. The vertical axis is the dependent variable. Each point represents a different target point. The fitted line represents the predicted value. There are several ways to plot a regression plot. A simple ways to use regplot from the seaborn library. First, "import seaborn." Then use the "regplot" function. The parameter x is the name of the column that contains the independent variable or feature. The parameter y, contains the name of the column that contains the name of the dependent variable or target. The parameter data is the name of the dataframe. The result is given by the plot. The residual plot represents the error between the actual value. Examining the predicted value and actual value we see a difference. We obtain that value by subtracting the predicted value, and the actual target value. We then plot that value on the vertical axis with the independent variable as the horizontal axis. Similarly, for the second sample, we repeat the process. Subtracting the target value from the predicted value. Then plotting the value accordingly. Looking at the plot gives us some insight into our data. We expect to see the results to have zero mean, distributed evenly around the x axis with similar variance. There is no curvature. This type of residual plot suggests a linear plot is appropriate. In this residual plot, there is a curvature. The values of the error change with x. For example, in the region, all the residual errors are positive. In this area, the residuals are negative. In the final location, the error is large. The residuals are not randomly separated. This suggests the linear assumption is incorrect. This plot suggests a nonlinear function. We will deal with this in the next section. In this plot, we see the variance of the residuals increases with x. Therefore, our model is incorrect. We can use seaborn to create a residual plot. First, "import seabourn." We use the "residplot" function. The first parameter is a series of dependent variable or feature. The second parameter is a series of dependent variable or target. We see in this case, the residuals have a curvature. A distribution plot counts the predicted value versus the actual value. These plots are extremely useful for visualizing models with more than one independent variable or feature. Let's look at a simplified example. We examined the vertical axis. We then count and plot the number of predicted points that are approximately equal to one. We then, count and plot the number of predicted points that are approximately equal to two. We repeat the process. For predicted points, they are approximately equal to three. Then we repeat the process for the target values. In this case, all the target values are approximately equal to two. The values of the targets and predicted values are continuous. A histogram is for discrete values. Therefore, pandas will convert them to a distribution. The vertical axis is scaled to make the area under the distribution equal to one. This is an example of using a distribution plot. The dependent variable or feature is price. The fitted values that result from the model are in blue. The actual values are in red. We see the predicted values for prices in the range from 40,000 to 50,000 are inaccurate. The prices in the region from 10,000 to 20,000 are much closer to the target value. In this example, we use multiple features or independent variables. Comparing it to the plot on the last slide, we see predicted values are much closer to the target values. Here's the code to create a distribution plot. The actual values are used as a parameter. We wanted distribution instead of a histogram. So we want the hist parameters set to false. The color is red. The label is also included. The predicted values are included for the second plot. The rest of the parameters are set accordingly.

**Polynomial Regression and Pipelines**

In this video, we will cover polynomial regression and pipelines. What do we do when a linear model is not the best fit for our data? Let's look into another type of regression model. The polynomial regression. We transform our data into a polynomial, then use linear regression to fit the parameter. Then we will discuss pipelines. Pipelines are a way to simplify your code. Polynomial regression is a special case of the general linear regression. This method is beneficial for describing curvilinear relationships. What is a curvilinear relationship? It's what you get by squaring or setting higher order terms of the predictor variables in the model transforming the data. The model can be quadratic, which means that the predictor variable in the model is squared. We use a bracket to indicate it as an exponent. This is a second order polynomial regression, with a figure representing the function. The model can be cubic, which means that the predictor variable is cubed. This is the third order polynomial regression. We see by examining the figure that the function has more variation. There also exists higher order polynomial regressions. When a good fit hasn't been achieved by second or third order. We can see in figures how much the graphs change, when we change the order of the polynomial regression. The degree of the regression makes a big difference and can result in a better fit If you pick the right value. In all cases, the relationship between the variable and the parameter is always linear. Let's look at an example from our data where we generate a polynomial regression model. In Python we do this by using the polyfit function. In this example, we develop a third order polynomial regression model base. We can print out the model. Symbolic form for the model is given by the following expression: negative 1.557 x1 cubed plus 204.8 x one squared plus 8965 x1 plus 1.37 times 10 to the power of five. We could also have multi-dimensional polynomial linear regression. The expression can get complicated. Here are just some of the terms for a two dimensional second order polynomial. Numpy's polyfit function cannot perform this type of regression. We use the preprocessing library in scikit-learn to create a polynomial feature object. The constructor takes the degree of the polynomial as a parameter. Then we transform the features into a polynomial feature with the fit underscore transform method. Let's do a more intuitive example. Consider the features shown here. Applying the method we transform the data, we now have a new set of features that are a transformed version of our original features. As the dimension of the data gets larger, we may want to normalize multiple features in scikit-learn. Instead we can use the preprocessing module to simplify many tasks. For example, we can standardize each feature simultaneously. We import StandardScaler. We train the object, fit the scale object, then transform the data into a new data frame on array x\_scale. There are more normalization methods available in the preprocessing library as well as other transformations. We can simplify our code by using a pipeline library. There are many steps to getting a prediction. For example, normalization, polynomial transform, and linear regression. We simplify the process using a pipeline. Pipeline sequentially perform a series of transformations. The last step carries out a prediction. First we import all the modules we need, then we import the library pipeline. We create a list of tuples, the first element in the tuple contains the name of the estimator model. The second element contains model constructor. We input the list in the pipeline constructor. We now have a pipeline object. We can train the pipeline by applying the train method to the pipeline object. We can also produce a prediction as well. The method normalizes the data, performs a polynomial transform, then outputs a prediction.

**Measures for In-Sample Evaluation**

Now that we’ve seen how we can evaluate a model by using visualization, we want to numerically evaluate our models. Let’s look at some of the measures that we use for in-sample evaluation. These measures are a way to numerically determine how good the model fits on our data. Two important measures that we often use to determine the fit of a model are: Mean Square Error (MSE), and R-squared. To measure the MSE, we find the difference between the actual value y and the predicted value yhat then square it. In this case, the actual value is 150; the predicted value is 50. Subtracting these points we get 100. We then square the number. We then take the Mean or average of all the errors by adding then all together and dividing by the number of samples. To find the MSE in Python, we can import the “mean\_Squared\_error” from “scikit-learn.metrics”. The “mean\_Squared\_error” function gets two inputs: the actual value of target variable and the predicted value of target variable. R-squared is also called the coefficient of determination. It’s a measure to determine how close the data is to the fitted regression line. So how close is our actual data to our estimated model? Think about it as comparing a regression model to a simple model, i.e., the mean of the data points. If the variable x is a good predictor our model should perform much better than just the mean. In this example the average of the data points 𝑦 | is 6. Coefficient of Determination R^2 is 1 minus the ratio of the MSE of the regression lined divided by the MSE of the average of the data points. For the most part, it takes values between 0 and 1. Let’s look at a case where the line provides a relatively good fit. The blue line represents the regression line. The blue squares represent the MSE of the regression line. The red line represents the average value of the data points. The red squares represent the MSE of the red line. We see the area of the blue squares is much smaller than the area of the red squares. In this case, because the line is a good fit, the Mean squared error is small, therefore the numerator is small. The Mean squared error of the line is relatively large, as such the numerator is large. A small number divided by a larger number is an even smaller number. Taken to an extreme this value tends to zero. If we Plug in this value from the previous slide for R^2, we get a value near one, this means the line is a good fit for the data. Here is an example of a line that does not fit the data well. If we just examine the area of the red squares compared to the blue squares, we see the area is almost identical. The ratio of the areas is close to one. In this case the R^2 is near zero. This line performs about the same as just using the average of the data points, therefore, this line did not perform well. We find the R-squared value in Python by using the score method, in the linear regression object. From the value that we get from this example, we can say that approximately 49.695% of the variation of price is explained by this simple linear model. Your R^2 value is usually between 0 and 1. If your R^2 is negative, it can be due to over fitting that we will discuss in the next module.

**Prediction and Decision Making**

The first thing you should do is make sure your model results make sense. You should always use visualization, numerical measures for evaluation, and comparing between different models. Let's look at an example of prediction. If you recall we trained the model using the fit method. Now, we want to find out what the price would be for a car that has a highway miles per gallon of 30. Plugging this value into the predict method gives us a resulting price of $13,771.30 this seems to make sense. For example, the value is not negative, extremely high, or extremely low. We can look at the coefficients by examining the coeff underscore attribute. If you recall the expression for the simple linear model that predicts price from highway miles per gallon. This value corresponds to the multiple of the highway miles per gallon feature. As such an increase of one unit in highway miles per gallon the value of the car decreases approximately 821 dollars. This value also seems reasonable

sometimes your model will produce values that don't make sense. For example, if we plot the model out for highway miles per gallon in the ranges of 0 to 100 we get negative values for the price. This could be because the values in that range are not realistic the linear assumption is incorrect or we don't have data for cars in that range. In this case it is unlikely that a car will have fuel mileage in that range so our model seems valid. To generate a sequence of values in a specified range import numpy then use the numpy arrange function to generate the sequence the sequence starts at 1 and increments by 1 until we reach 100. The first parameter is the starting point of the sequence. The second parameter is the endpoint plus one of the sequence the final parameter is the step size between elements in the sequence. In this case it's one so we increment the sequence one step at a time from one to two and so on. We can use the output to predict new values the output is a numpy array many of the values are negative. Using a regression plot to visualize your data is the first method you should try. See the labs for examples of how to plot polynomial regression for this example the effect of the independent variable is evident in this case. The data trends down as the dependent variable increases the plot also shows some non-linear behavior.

Examining the residual plot we see in this case, the residuals have a curvature suggesting non-linear behavior. A distribution plot is a good method for multiple linear regression. For example, we see the predicted values for prices in the range from thirty thousand to fifty thousand are inaccurate this suggests a non-linear model may be more suitable or we need more data in this range. The mean square error is perhaps the most intuitive numerical measure for determining if a model is good or not. Let's see how different measures of mean square error impact the model. The figure shows an example of a mean square error of 3495. This example has a mean square error of three thousand six hundred and fifty two. The final plot has a mean square error of twelve thousand eight hundred and seventy. As the square error increases the targets get further from the predicted points. As we discussed r squared is another popular method to evaluate your model. It tells you how well your line fits into the model. r squared values range from zero to one. r squared tells us what percent of the variability in the dependent variable is accounted for by the regression on the independent variable. An r squared of 1 means that all movements of another dependent variable are completely explained by movements in the independent variables. In this plot we see the target points in red and the predicted line in blue. An r squared of 0.9986 the model appears to be a good fit. That means that more than 99 of the variability of the predicted variable is explained by the independent variables. This model has an r squared of 0.9226 there still is a strong linear relationship model is still a good fit. An r squared of 0806 of the data we can visually see that the values are scattered around the line. They are still close to the line and we can say that 80 percent of the variability of the predicted variable is explained by the independent variables. And an r squared 0.61 means that approximately 61 percent of the observed variation can be explained by the independent variables. An acceptable value for r squared depends on what field you are studying and what your use case is. Falcon Miller 1992 suggests that an acceptable r squared value should be at least 0.1. Does a lower mean square error imply better fit? Not necessarily MSE4 and MLR model will be smaller than the MSE for an SLR model. Since the errors of the data will decrease when more variables are included in the model. Polynomial regression will also have a smaller MSE than regular regression in the next section we will look at more accurate ways to evaluate the model.