**Data processing.**

**One Hot Encoding.**

categorical variables contain the label values rather than numerical values. The number of possible values is often limited to a fixed set. Categorical variables are often called nominal. Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

This implies that category data must be transformed into numerical data

One hot encoding is an instance of Feature Engineering.  Feature Engineering is the process of extracting features from raw data using the domain knowledge of the problem. These features can be used to improve the performance of machine learning algorithms and if the performance increase then it will give the best accuracy. We can also say that feature engineering is the same as applied machine learning. Feature engineering is the most important art in machine learning which creates a huge difference between a good model and a bad model.

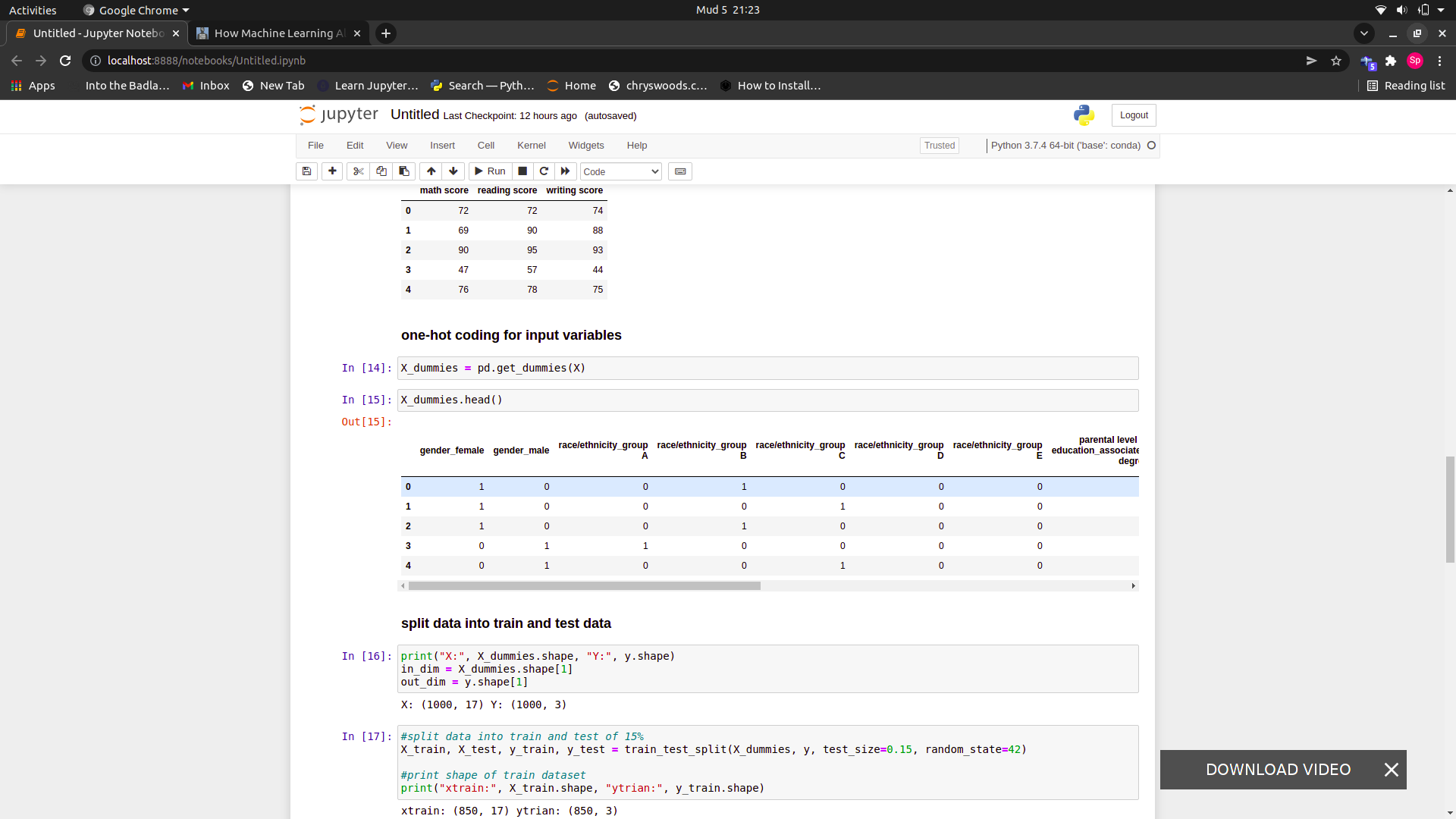
I.e In the “gender” variable example, there are two categories, and, therefore, two binary variables are needed. A “1” value is placed in the binary variable for the gender and “0” values for the other gender.

**Splitting data into X-variables and y-variables**

Machine learning algorithms are described as mapping function that turns input variables (X) into the output variable (Y)

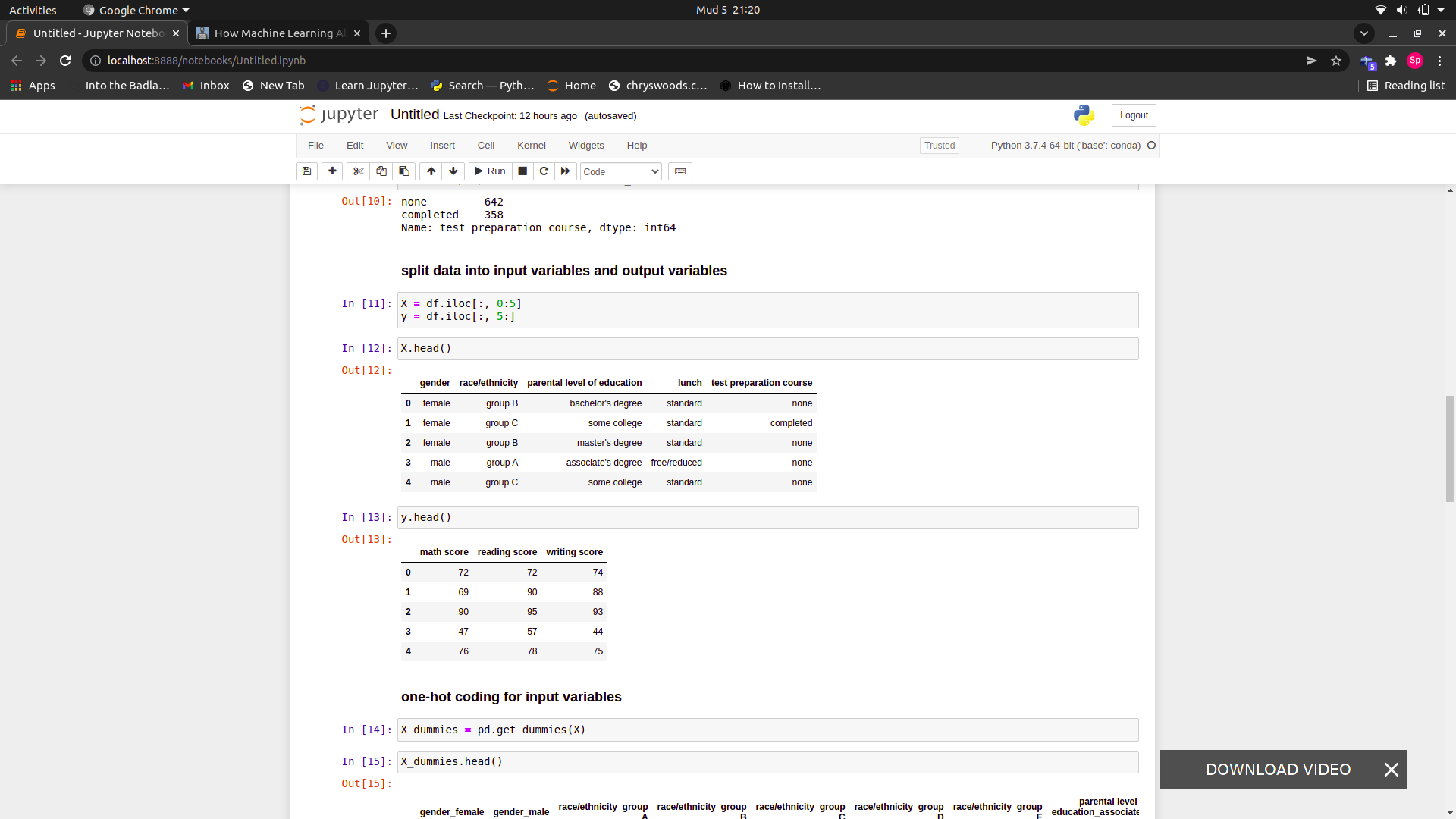
The project is a linear regression problem model which is an instance of supervised machine learning, I.e it has a target variable.

I used pandas, ‘get dummies’ function to encode input variables



Regressions used to predict the outcome of a given sample when the output variable is in the form of real values. For example, a regression model might process input data to predict the amount of rainfall, the height of a person, etc.

code used to split data int X and y variables is below.



# Building Neural Network

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria.

I used Keras to build my neural network

# Keras is a simple tool for constructing a neural network. It is a high-level framework based on tensorflow, theano or cntk back ends.

Sequential specifies to keras that we are creating model sequentially and the output of each layer we add is input to the next layer we specify.

model.add is used to add a layer to our neural network. We need to specify as an argument what type of layer we want. The Dense is used to specify the fully connected layer. The arguments of Dense are output dimension which is 3 in the first case, input dimension which is 17, from shape of our X training data, for input dimension and the activation function to be used which is relu in this case.

Adding a second layer was reducing performance of the the matrix so, I defaulted to using one layer.

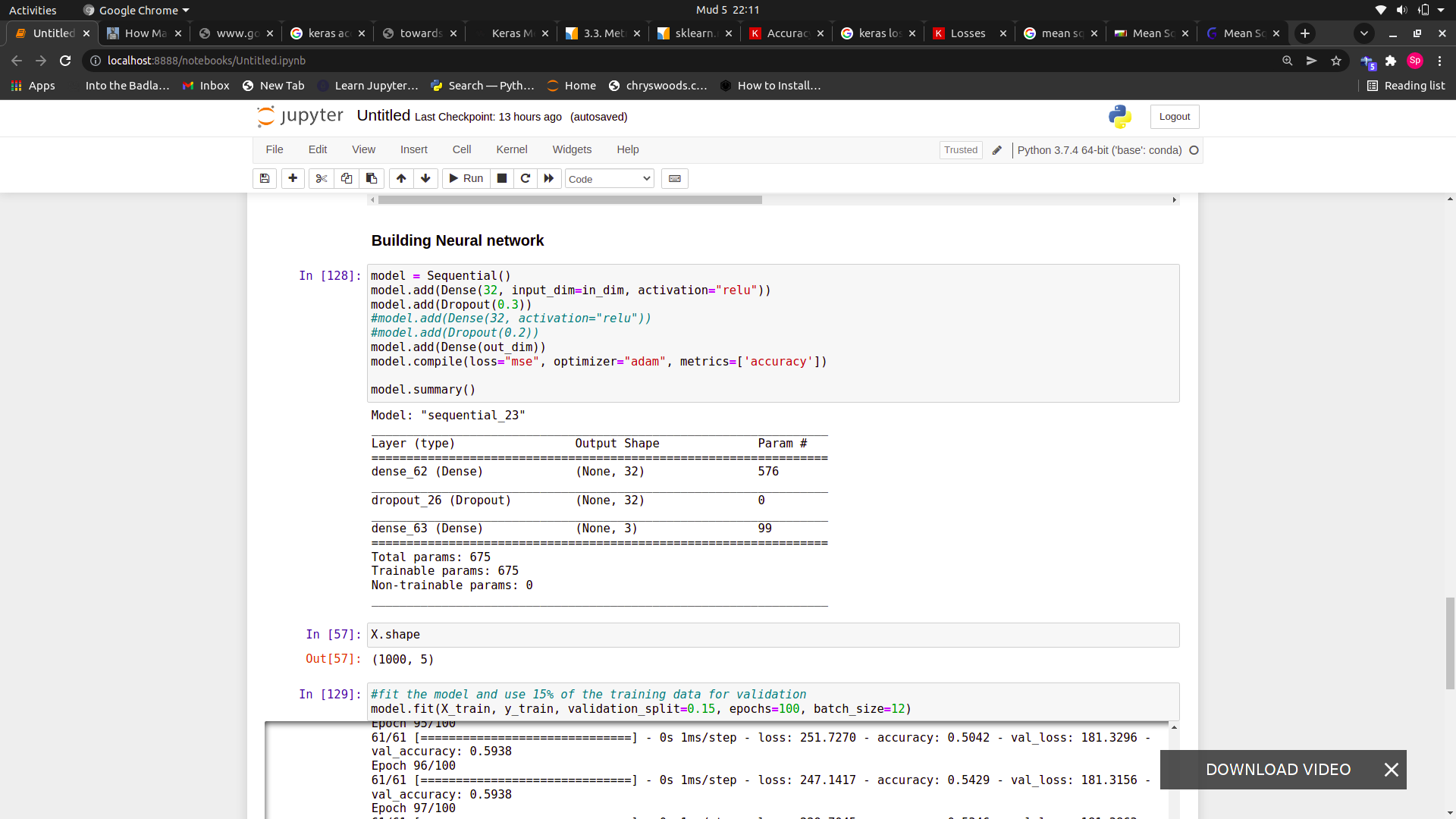
**Dropout** is a technique for preventing overfitting in a model. At each update of the training phase, Dropout works by setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0. Dropout layer only applies when training is set to True such that no values are dropped during inference. When using model.fit, training is automatically set to True, and in other cases, you can manually set the kwarg to True when calling the layer.

I tried dropout from 0.1 to 0.5, and accuracy drops from 0.4 so I decide to use a dropout of 0.3

For validation I decided on using Mean squared error and accuracy score. Since its regression problem I need to use validation strategies of regresion challenges.

The [**mean\_squared\_error**](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error) function computes [mean square error](https://en.wikipedia.org/wiki/Mean_squared_error), a risk metric corresponding to the expected value of the squared (quadratic) error or loss. It tells how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the “errors”) and squaring them. The squaring is necessary to remove any negative signs. It also gives more weight to larger differences. It’s called the [mean](https://www.statisticshowto.com/mean/)squared error as you’re finding the average of a set of errors. The lower the MSE, the better the forecast.

The code for buliding my model is below:

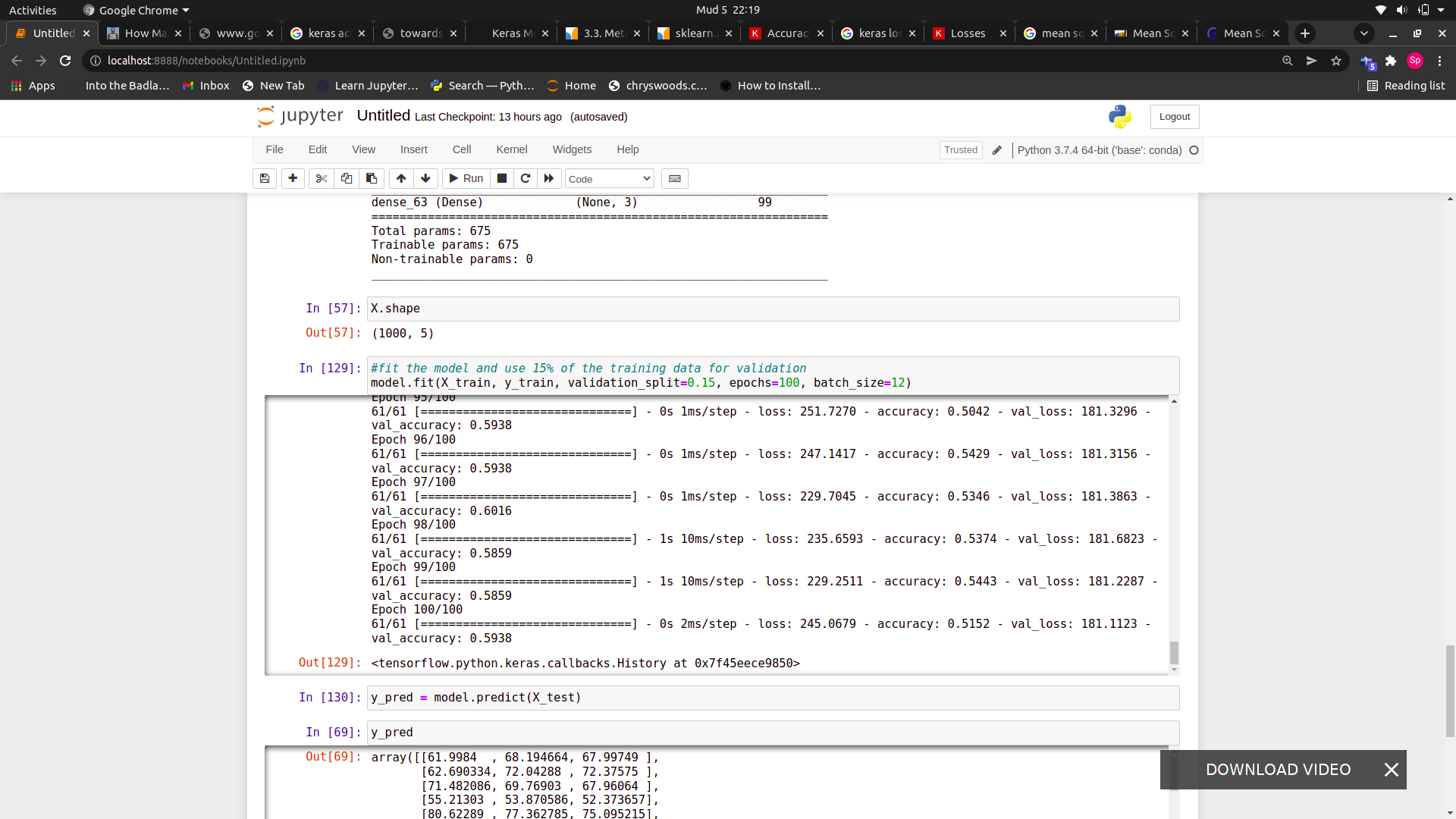


Keras can separate a portion of your training data into a validation dataset and evaluate the performance of your model on that validation dataset each epoch.

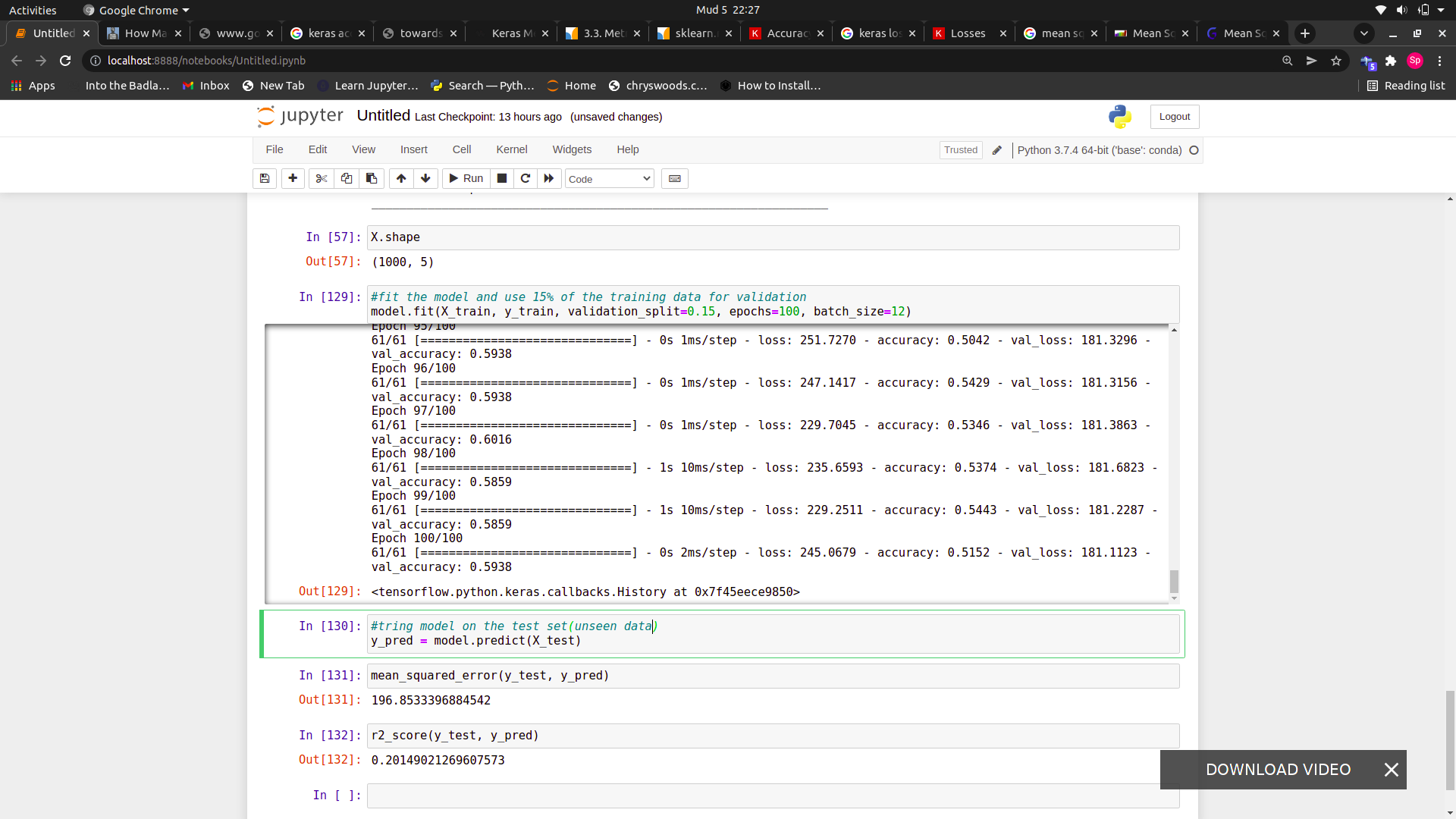
You can do this by setting the validation\_split argument on the fit() function to a percentage of the size of your training dataset.

When I was fitting on the model I used a validation portion of 15 % of the training data.

At the end of traing the model had an accuracy of 59% and a MSE of 181.



Trying performance of the model on test data the model MSE on unseen data was 196, which show the model is performance was good though need improvement.



The model need more data to improve on its performance.