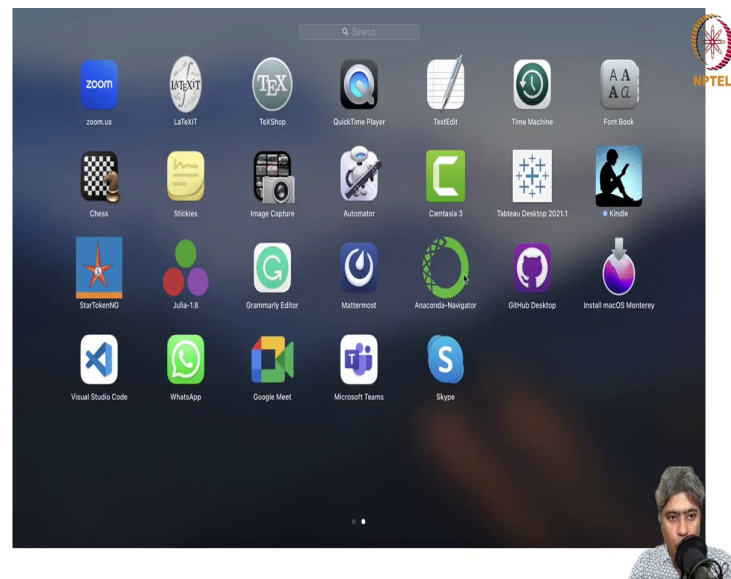


Predictive Analytics - Regression and Classification
Prof. Sourish Das
Department of Mathematics
Chennai Mathematical Institute

Lecture - 34
Hands on with Python: Handle multicollinearity with Ridge correction

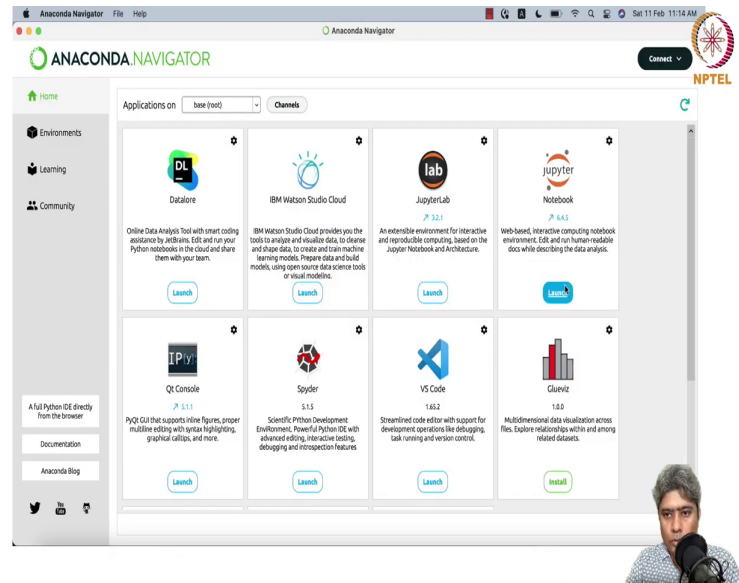
Hello all, welcome to lecture 10 on Predictive regression, Predictive Analytics, Regression and Classification course. In this lecture we will be doing hands-on with Python. So, first we will open our.

(Refer Slide Time: 00:37)



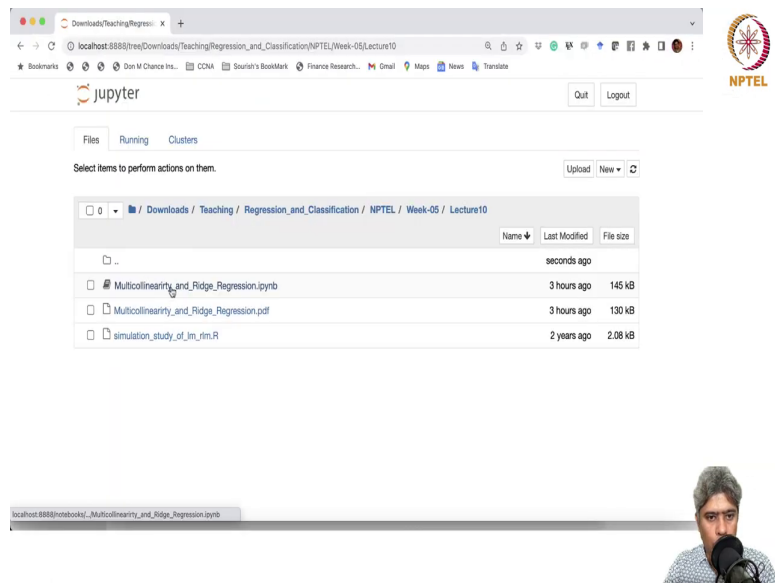
I have a notebook, particular notebook, let me open that notebook.

(Refer Slide Time: 00:50)



So, we launch Jupyter notebook.

(Refer Slide Time: 00:56)

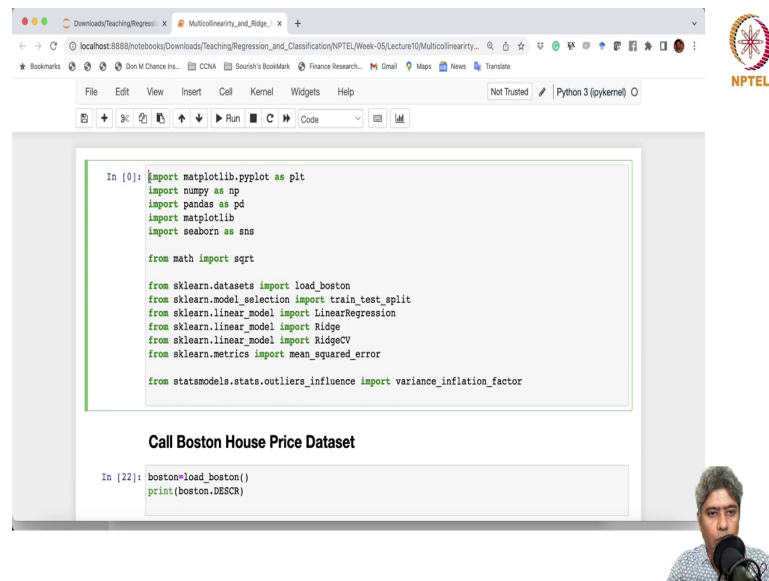


The screenshot displays a web browser window with a JupyterLab interface. The address bar shows the URL: localhost:8888/tree/Downloads/Teaching/Regression_and_Classification/NPTEL/Week-05/Lecture10. The JupyterLab header includes the 'jupyter' logo and 'Quit' and 'Logout' buttons. Below the header, there are tabs for 'Files', 'Running', and 'Clusters'. A message states 'Select items to perform actions on them.' with 'Upload' and 'New' buttons. The main area shows a file browser view for the path: Downloads / Teaching / Regression_and_Classification / NPTEL / Week-05 / Lecture10. A table lists the files in the directory:

Name	Last Modified	File size
..	seconds ago	
Multicollinearity_and_Ridge_Regression.ipynb	3 hours ago	145 kB
Multicollinearity_and_Ridge_Regression.pdf	3 hours ago	130 kB
simulation_study_of_lm_lm.R	2 years ago	2.08 kB

At the bottom of the browser window, a small video thumbnail shows a person speaking into a microphone. The address bar at the very bottom of the browser shows the file path: localhost:8888/notebook/Multicollinearity_and_Ridge_Regression.ipynb.

(Refer Slide Time: 00:58)



```
In [0]: import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib
import seaborn as sns

from math import sqrt

from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from sklearn.metrics import mean_squared_error

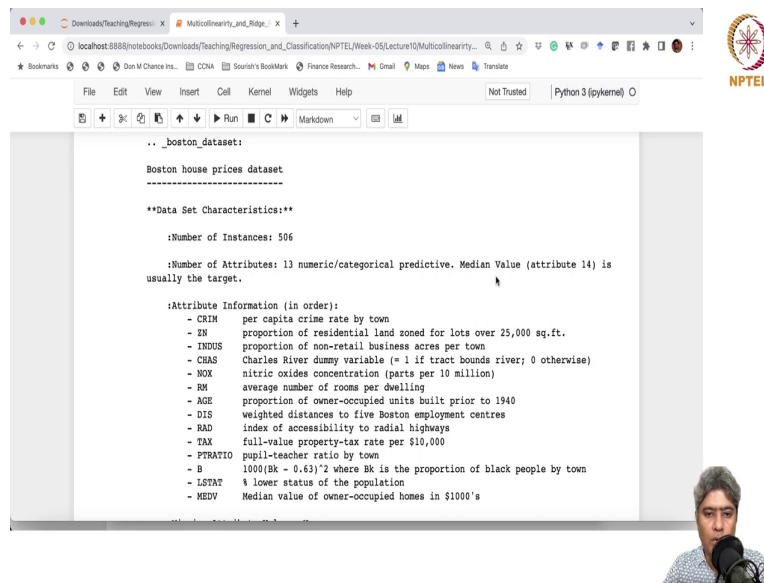
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Call Boston House Price Dataset

```
In [22]: boston=load_boston()
print(boston.DESCR)
```

First thing what we are doing here, we are calling bunch of we are calling bunch of packages and we are loading those that matplot library for plotting numpy pandas, matplot library, seaborn, math, we are importing square root function from math. Then from sklearn data sets, we are importing boston data sets, then from our sklearn we are model selection, we are importing train test split functions, linear regression, ridge, RidgeCV and mean squared error and variance inflation factor.

(Refer Slide Time: 01:55)

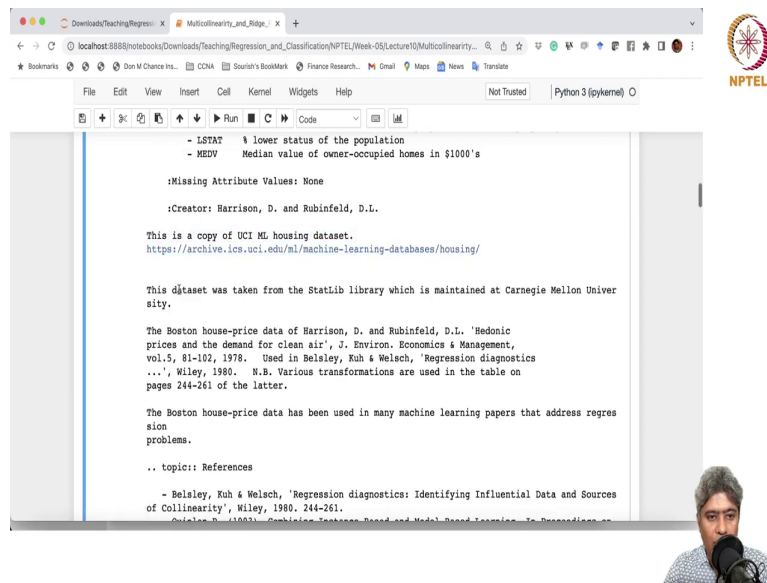


The screenshot shows a Jupyter Notebook interface with a terminal window displaying the following text:

```
.._boston_dataset:
Boston house prices dataset
-----
**Data Set Characteristics:**
:Number of Instances: 506
:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.
:Attribute Information (in order):
- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX nitric oxides concentration (parts per 10 million)
- RM average number of rooms per dwelling
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per $10,000
- PTRATIO pupil-teacher ratio by town
- B 1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
- LSTAT % lower status of the population
- MEDV Median value of owner-occupied homes in $1000's
```

So, we are going to work with Boston house price data. So, let me just compile all these things. So, now they are all compiled. Now, we are going to run call the Boston house price data. Now, in the Boston house price data we are, going to have lots of predictors and many of these predictors are correlated ok. So, now in the boston data sets we have about 500 instances. The number of attributes is median value of the house of a house.

(Refer Slide Time: 02:51)



```
Download/Teaching/Regre... x Multicollinearity_and_Ridge... x
localhost:8888/notebooks/Downloads/Teaching/Regression_and_Classification/NPTEL/Week-05/Lecture10/Multicollinearity...
Bookmarks Don M Chance Ins... CCNA Sourin's Bookmark Finance Research... Gmail Maps News Translate
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)
+ - Run Code
- LSTAT % lower status of the population
- MEDV Median value of owner-occupied homes in $1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression problems.

.. topic:: References
- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
- Gelman, D. (1993). 'Combining Expertise: Bayes and Model-Based Learning in Regression'
```

And then bunch of features are available, CRIM crim stands for per capital crime rate by town, ZN stands for proportion of residential land zone for lots over 25,000 square feet. INDUS stands for proportion of non retail business acres per town. So; that means, how much commercial activities are happening.

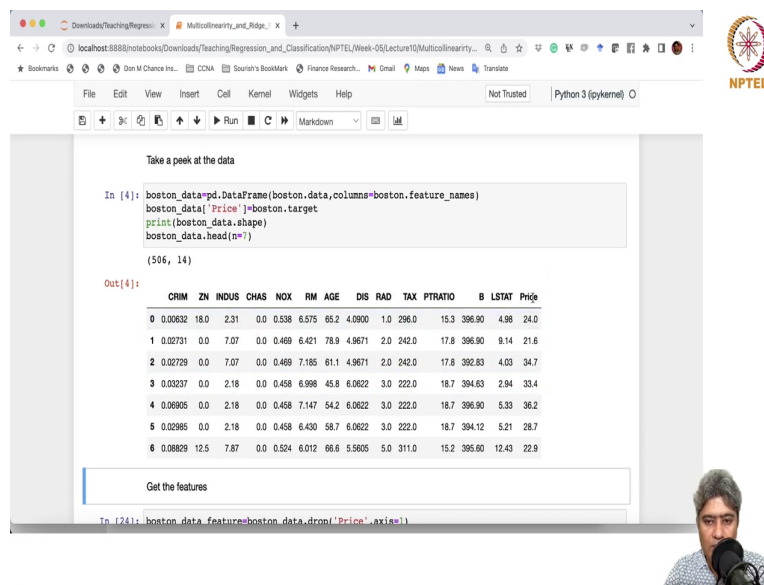
Charles rivers dummy variable, if it is equal to one; that means, tract bounds river and 0 otherwise. NOX stands for nitric oxide concentration, so basically in that area of the house, what is the air pollution level? RM stands for average number of rooms per dwelling. AGE stands for the proportion of owner occupied units built prior to 1940.

So, weighted distance to five Boston employment centres. Index of accessibility to radial highways, full value of property tax, pupil teacher ratio in the town that you have good the

school is, then lower status of the population median value of the owner occupied homes. So, what is the value of the owner occupied homes? That is what we are going to predict.

So, here is the data set actually was first available from the UCL machine learning housing data set typically it is known as also Boston housing data set.

(Refer Slide Time: 04:46)



```
In [4]: boston_data=pd.DataFrame(boston_data,columns=boston.feature_names)
boston_data['Price']=boston.target
print(boston_data.shape)
boston_data.head(n=7)
```

(506, 14)

```
Out[4]:
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.80	12.43	22.9

```
In [24]: boston_data.features=boston_data.drop('Price',axis=1)
```

Here is the reference of the data set. So, let us take a peek of the data, if you run. So, here is the it has 506, instances and 14 cases. So, 2 3. So, this CRIM to L stat this is essentially these are all feature variable.

(Refer Slide Time: 05:30)

```
In [5]: boston_data_feature=boston_data.drop('Price',axis=1)
boston_data_feature.head(n=7)

Out[5]:
```

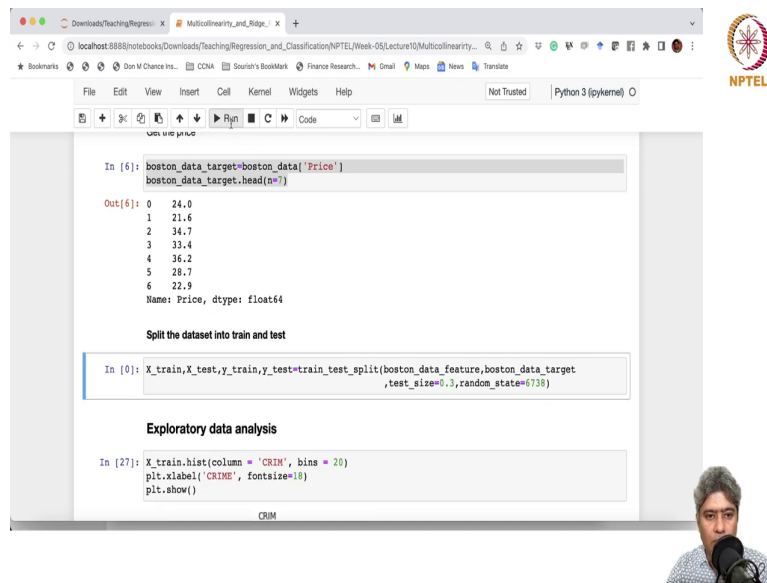
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.06032	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43

```
In [25]: boston_data_target=boston_data['Price']
boston_data_target.head(n=7)

Out[25]: 0    24.0
1    31.6
```

And price is the last column price is the predictor the dependent or target variable. So, if we just get the features.

(Refer Slide Time: 05:40)



The screenshot shows a Jupyter Notebook window with the following content:

```
In [6]: boston_data_target=boston_data['Price']
boston_data_target.head(n=7)

Out[6]: 0    24.0
        1    21.6
        2    34.7
        3    33.4
        4    36.2
        5    28.7
        6    22.9
        Name: Price, dtype: float64
```



Split the dataset into train and test

```
In [0]: X_train,X_test,y_train,y_test=train_test_split(boston_data_feature,boston_data_target
, test_size=0.3,random_state=6738)
```

Exploratory data analysis

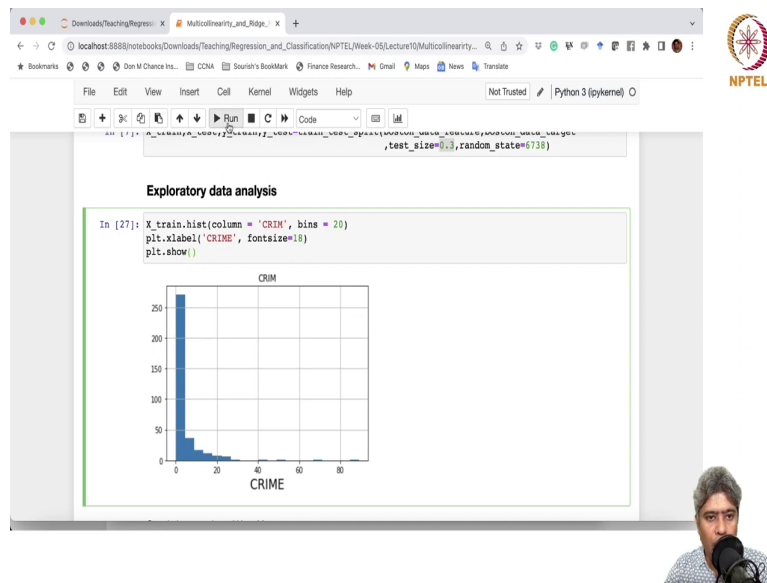
```
In [27]: X_train.hist(column = 'CRIM', bins = 20)
plt.xlabel('CRIME', fontsize=18)
plt.show()
```

CRIM



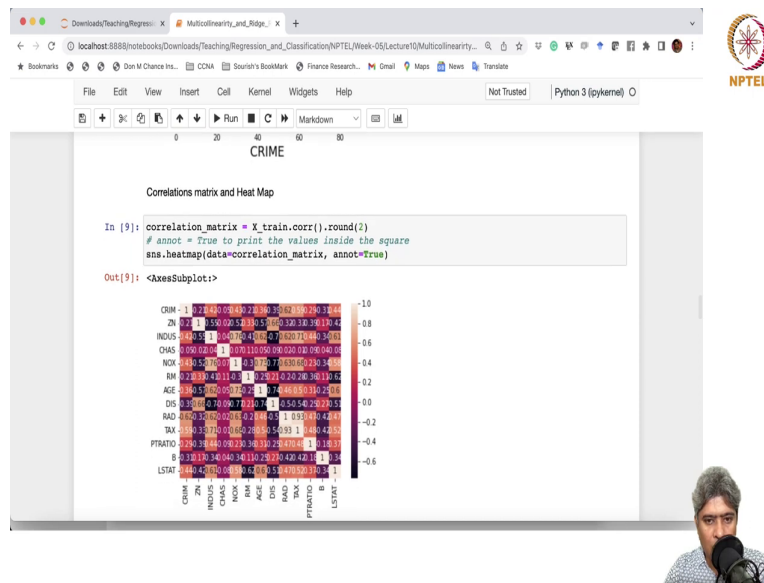
So, here we are just getting the features and in this piece of code we are just running we are getting the price. Here we are in this place we are just you know splitting the data into train and test.

(Refer Slide Time: 06:06)



Here we are splitting the data into train and test and then we are doing some exploratory data analysis on the. So, one thing in the training and testing we are keeping 30 percent of the data for test and 70 percent of the data for training. Now, we are drawing the doing some exploratory data analysis on the training data set.

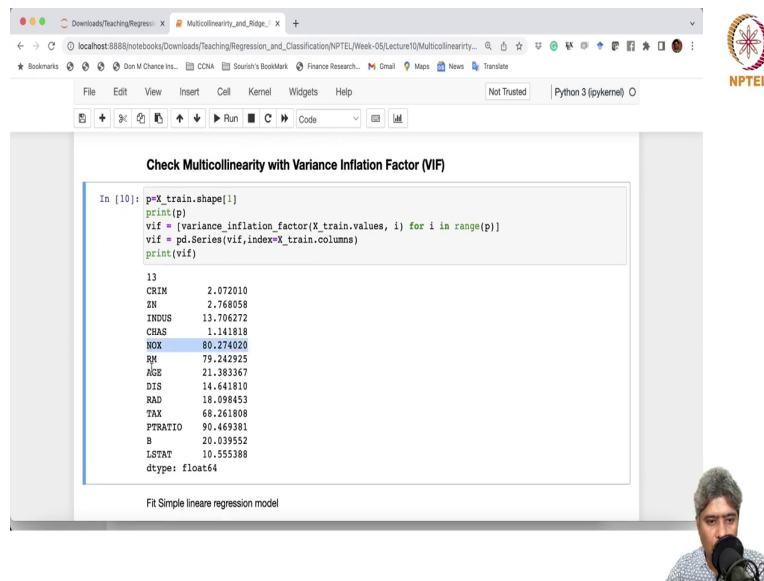
(Refer Slide Time: 06:41)



Here we are drawing the correlation matrix. If you look into the correlation matrix you will find an interesting phenomena. So, this dark purple which are near negative 0.7. So, with this DIS we are seeing that CRIME INDUS, NOX, age all these are quite strongly correlated. So, the so, there are quite a few variables are strongly correlated with DIS. What was the DIS? Let us go up and check it out. Weighted distance to five Boston employment centre.

So, how far it is from the employment centre like all offices that has strong correlation with how close it to river, nitrogen oxide level, age of the house and you know all other variables.

(Refer Slide Time: 08:02)



```
In [10]: px_train.shape[1]
print(p)
vif = [variance_inflation_factor(X_train.values, i) for i in range(p)]
vif = pd.Series(vif, index=X_train.columns)
print(vif)

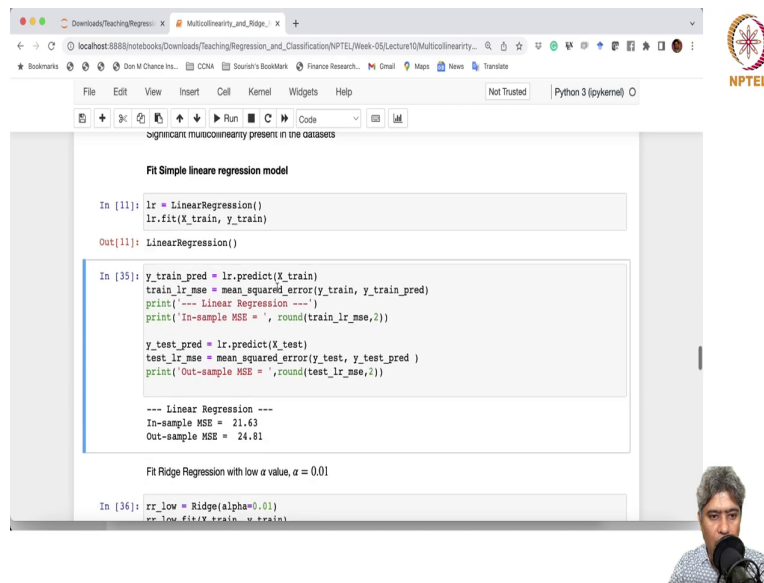
13
CRIM      2.072010
ZN        2.760558
INDUS    13.706272
CHAS      1.141818
NOX      80.274020
RM        79.242925
AGE       21.383367
DIS       14.641810
RAD        18.998453
TAX        68.261808
PTRATIO   90.469381
B         20.039552
LSTAT     10.555388
dtype: float64
```

Fit Simple linear regression model

So, here I am writing taking the X and then I am just running the variance inflation factor on this for a given values and then here all the vif are being printed.

So, what we are seeing for knocks the high its and RM the variance inflation factor is very high. PTRATIO, p tra PT Ratio ok. What is PT ratio? Let us check it out. What was the PT ratio? Pupil teachers ratio has very strong correlation with has our you know multicollinearity variance inflation factor. So, there is a quite significant variance inflation factor indicates that there is a quite significant multicollinearity present.

(Refer Slide Time: 09:15)



```
significant multicollinearity present in the datasets

Fit Simple linear regression model

In [11]: lr = LinearRegression()
lr.fit(X_train, y_train)

Out[11]: LinearRegression()

In [35]: y_train_pred = lr.predict(X_train)
train_lr_mse = mean_squared_error(y_train, y_train_pred)
print("--- Linear Regression ---")
print("In-sample MSE = ", round(train_lr_mse,2))

y_test_pred = lr.predict(X_test)
test_lr_mse = mean_squared_error(y_test, y_test_pred )
print("Out-sample MSE = ",round(test_lr_mse,2))

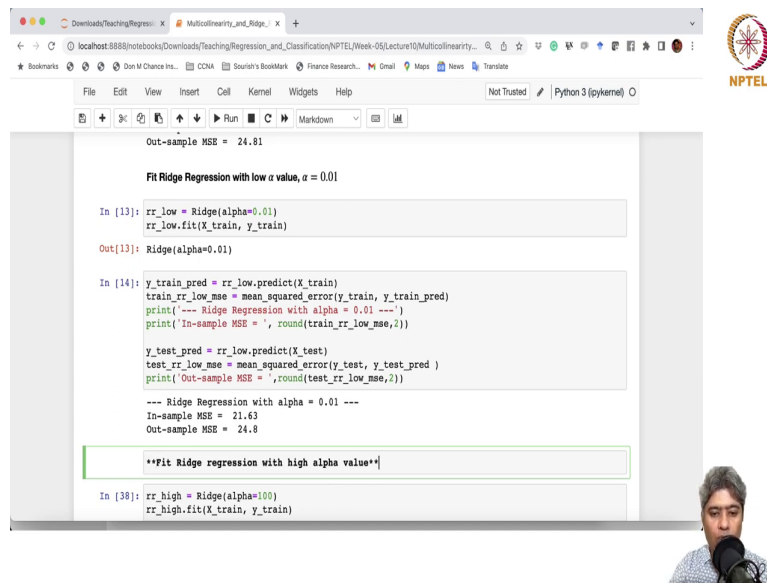
--- Linear Regression ---
In-sample MSE = 21.63
Out-sample MSE = 24.81

Fit Ridge Regression with low  $\alpha$  value,  $\alpha = 0.01$ 

In [36]: rr_low = Ridge(alpha=0.01)
rr_low.fit(X_train, y_train)
```

So, significant, multicollinearity present in the data set ok. So, first we fit simple linear regression model, first we fit simple linear regression in the model just we call lr, lr dot fit x train y train and then we did prediction and then from the predicted value we here in this line we calculate the mean square error in sample mean square error and out of sample mean square error.

(Refer Slide Time: 10:11)



```
Out-sample MSE = 24.81

Fit Ridge Regression with low alpha value, alpha = 0.01

In [13]: rr_low = Ridge(alpha=0.01)
rr_low.fit(X_train, y_train)

Out[13]: Ridge(alpha=0.01)

In [14]: y_train_pred = rr_low.predict(X_train)
train_rr_low_mse = mean_squared_error(y_train, y_train_pred)
print('--- Ridge Regression with alpha = 0.01 ---')
print('In-sample MSE = ', round(train_rr_low_mse,2))

y_test_pred = rr_low.predict(X_test)
test_rr_low_mse = mean_squared_error(y_test, y_test_pred)
print('Out-sample MSE = ', round(test_rr_low_mse,2))

--- Ridge Regression with alpha = 0.01 ---
In-sample MSE = 21.63
Out-sample MSE = 24.8

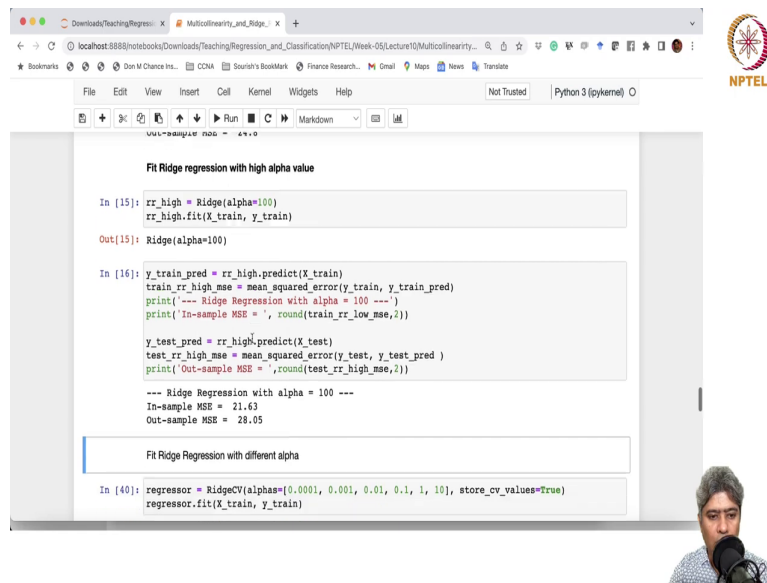
**Fit Ridge regression with high alpha value**

In [38]: rr_high = Ridge(alpha=100)
rr_high.fit(X_train, y_train)
```

So, In sample mean square error is 21.63 and Out of sample mean square error is 24.81. Now, we are going to fit ridge regression with alpha value to be 0.01, adhocly chosen alpha value to be 0.01. So, we are we just fit that model and now for that fitted regression model the in sample and out of the sample RMSE we calculated 21.63 and 24.8.

So, for adhocly chosen alpha gives us some value which is some in sample and on some out of sample RMSE which is close to the linear regression. Now, if we fit a ridge regression with a very high alpha value means lot of penalty there will be too much smoothing we choose alpha equal to 100.

(Refer Slide Time: 11:23)



```
Fit Ridge regression with high alpha value

In [15]: rr_high = Ridge(alpha=100)
         rr_high.fit(X_train, y_train)

Out[15]: Ridge(alpha=100)

In [16]: y_train_pred = rr_high.predict(X_train)
         train_rr_high_mse = mean_squared_error(y_train, y_train_pred)
         print("--- Ridge Regression with alpha = 100 ---")
         print('In-sample MSE = ', round(train_rr_lov_mse,2))

         y_test_pred = rr_high.predict(X_test)
         test_rr_high_mse = mean_squared_error(y_test, y_test_pred )
         print('Out-sample MSE = ',round(test_rr_high_mse,2))

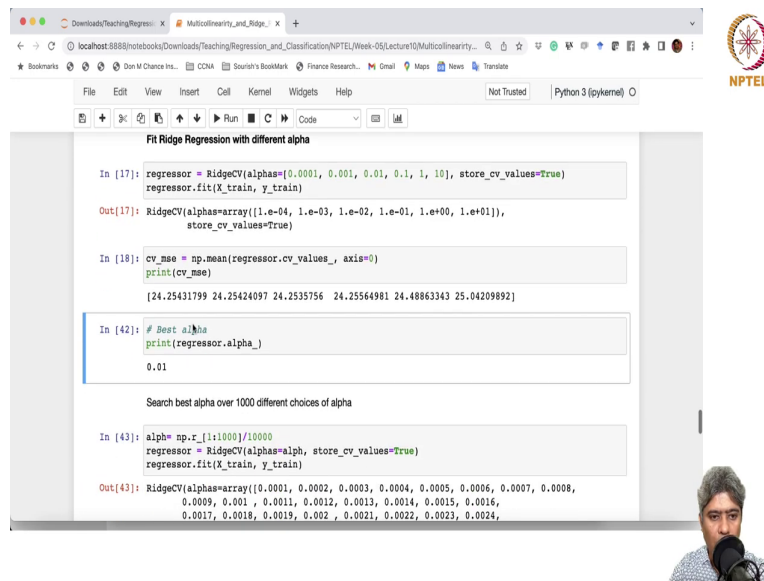
         --- Ridge Regression with alpha = 100 ---
         In-sample MSE = 21.63
         Out-sample MSE = 28.05

Fit Ridge Regression with different alpha

In [40]: regressor = RidgeCV(alphas=[0.0001, 0.001, 0.01, 0.1, 1, 10], store_cv_values=True)
         regressor.fit(X_train, y_train)
```

So, what happens? If we do that.

(Refer Slide Time: 11:34)



```
Fit Ridge Regression with different alpha

In [17]: regressor = RidgeCV(alphas=[0.0001, 0.001, 0.01, 0.1, 1, 10], store_cv_values=True)
regressor.fit(X_train, y_train)

Out[17]: RidgeCV(alphas=array([1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01]),
store_cv_values=True)

In [18]: cv_mse = np.mean(regressor.cv_values_, axis=0)
print(cv_mse)

[24.25431799 24.25424097 24.2535756 24.25564981 24.48863343 25.04209892]

In [42]: # Best alpha
print(regressor.alpha_)

0.01

Search best alpha over 1000 different choices of alpha

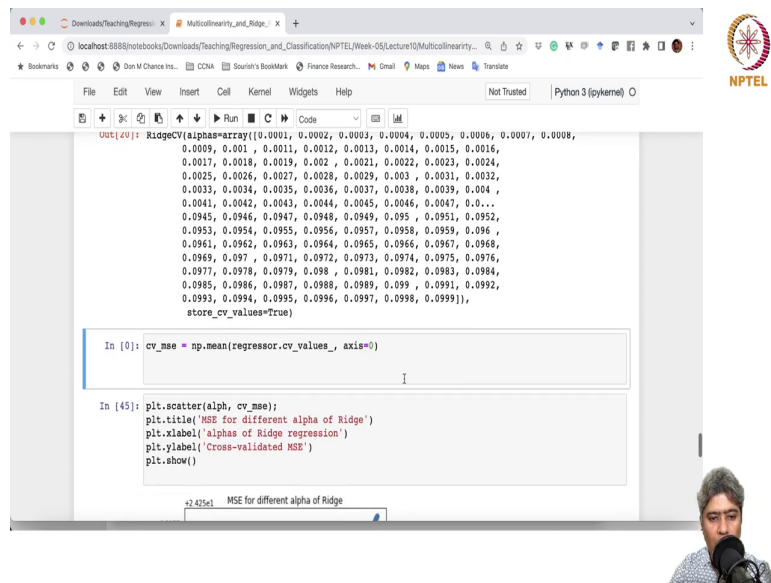
In [43]: alpha = np.linspace(0.0001, 10000, 10000)
regressor = RidgeCV(alphas=alpha, store_cv_values=True)
regressor.fit(X_train, y_train)

Out[43]: RidgeCV(alphas=array([0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008,
0.0009, 0.001, 0.0011, 0.0012, 0.0013, 0.0014, 0.0015, 0.0016,
0.0017, 0.0018, 0.0019, 0.002, 0.0021, 0.0022, 0.0023, 0.0024,
```

So, what we are seeing that out of sample RMSE now is too high in sample is somewhere in the same range 2.21.63 for some reason. But out of sample RMSE now is too high we cannot really you know do that. So, if we regression with a different value of alpha; different value of alpha cross validated k fold cross validation method.

So, RidgeCV essentially cross implement the regression ridge regression with k fold cross validation and it computes the ca computation c v m e like you know mean square error these are the mean square error we are seeing.

(Refer Slide Time: 12:31)



The screenshot shows a Jupyter Notebook interface with a browser window at the top. The browser address bar shows a local host path. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running and saving. The code cell contains the following Python code:

```
Out[40]: RidgeCV(alpha=array([0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008,
0.0009, 0.001, 0.0011, 0.0012, 0.0013, 0.0014, 0.0015, 0.0016,
0.0017, 0.0018, 0.0019, 0.002, 0.0021, 0.0022, 0.0023, 0.0024,
0.0025, 0.0026, 0.0027, 0.0028, 0.0029, 0.003, 0.0031, 0.0032,
0.0033, 0.0034, 0.0035, 0.0036, 0.0037, 0.0038, 0.0039, 0.004,
0.0041, 0.0042, 0.0043, 0.0044, 0.0045, 0.0046, 0.0047, 0.0048,
0.0049, 0.005, 0.0051, 0.0052, 0.0053, 0.0054, 0.0055, 0.0056, 0.0057, 0.0058, 0.0059, 0.006,
0.0061, 0.0062, 0.0063, 0.0064, 0.0065, 0.0066, 0.0067, 0.0068,
0.0069, 0.007, 0.0071, 0.0072, 0.0073, 0.0074, 0.0075, 0.0076,
0.0077, 0.0078, 0.0079, 0.008, 0.0081, 0.0082, 0.0083, 0.0084,
0.0085, 0.0086, 0.0087, 0.0088, 0.0089, 0.009, 0.0091, 0.0092,
0.0093, 0.0094, 0.0095, 0.0096, 0.0097, 0.0098, 0.0099]),
store_cv_values=True)

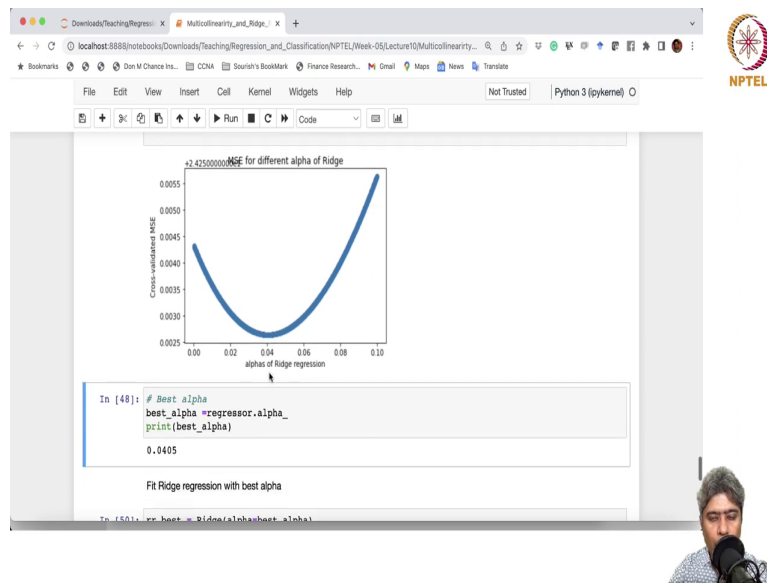
In [0]: cv_mse = np.mean(regressor.cv_values_, axis=0)

In [45]: plt.scatter(alpha, cv_mse);
plt.title('MSE for different alpha of Ridge')
plt.xlabel('alpha of Ridge regression')
plt.ylabel('Cross-validated MSE')
plt.show()
```

At the bottom of the notebook, a small thumbnail of a person speaking into a microphone is visible.

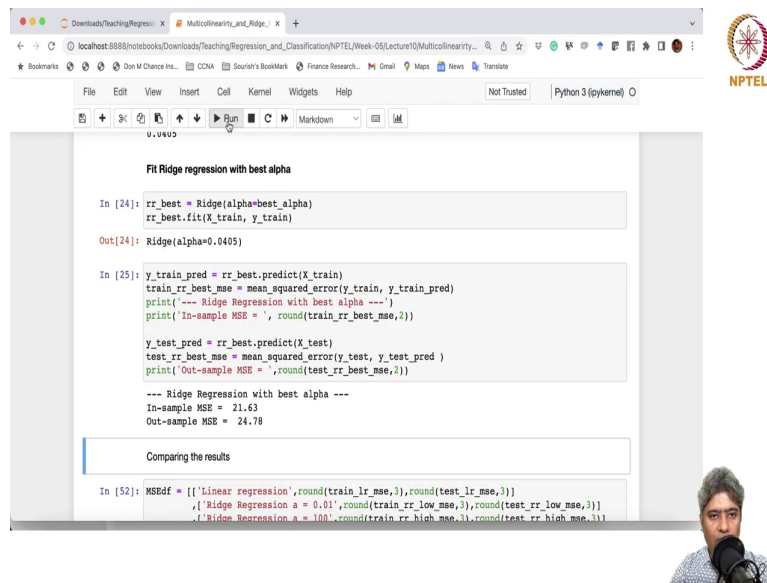
So, the best fit alpha is somewhere in the range of 0.01. So, kind of now what we can do why we choose only these values? Somewhere between 0 point. So, we choose this 0 point different alpha 0.0001, 0.001, 0.01, 0.11 and 10 these are the alphas. So, instead we choose thousands of values different choices of alpha where alpha will be taking values from 0.001 to 0.01 effectively.

(Refer Slide Time: 13:22)



And now we calculate the mean square error; mean square error plotted against different values of alpha. So, remember the, and what we are seeing that it is actually minimum at near 0.04.

(Refer Slide Time: 13:42)



```
Fit Ridge regression with best alpha

In [24]: rr_best = Ridge(alpha=best_alpha)
rr_best.fit(X_train, y_train)

Out[24]: Ridge(alpha=0.0405)

In [25]: y_train_pred = rr_best.predict(X_train)
train_rr_best_mse = mean_squared_error(y_train, y_train_pred)
print('--- Ridge Regression with best alpha ---')
print('In-sample MSE = ', round(train_rr_best_mse,2))

y_test_pred = rr_best.predict(X_test)
test_rr_best_mse = mean_squared_error(y_test, y_test_pred )
print('Out-sample MSE = ',round(test_rr_best_mse,2))

--- Ridge Regression with best alpha ---
In-sample MSE = 21.63
Out-sample MSE = 24.78

Comparing the results

In [52]: MSEdf = [['Linear regression',round(train_lr_mse,3),round(test_lr_mse,3)]
,{'Ridge Regression a = 0.01',round(train_rr_low_mse,3),round(test_rr_low_mse,3)}]
,['Ridge Regression a = 100',round(train_rr_high_mse,3),round(test_rr_high_mse,3)]
```

So, the best alpha that we can have is 0.0405. So, best alpha that we can have is 0.0405. So, we decide to fit the ridge regression with best alpha to fit the ridge regression with best alpha.

(Refer Slide Time: 14:13)

you can use statmodels package to check the standard error of the coefficients

```
y_test_pred = rr_best.predict(X_test)
test_rr_best_mse = mean_squared_error(y_test, y_test_pred)
print('Out-sample MSE = ', round(test_rr_best_mse, 2))

--- Ridge Regression with best alpha ---
In-sample MSE = 21.63
Out-sample MSE = 24.78

Comparing the results

In [26]: MSEdf = [['Linear regression', round(train_lr_mse, 3), round(test_lr_mse, 3)],
                ['Ridge Regression a = 0.01', round(train_rr_low_mse, 3), round(test_rr_low_mse, 3)],
                ['Ridge Regression a = 100', round(train_rr_high_mse, 3), round(test_rr_high_mse, 3)],
                ['Ridge Regression best a = 0.0404', round(train_rr_best_mse, 3), round(test_rr_best_mse, 3)]]
MSEdf = pd.DataFrame(MSEdf, columns=['Method', 'In-Sample', 'Out-Sample'])
print(MSEdf)
```

	Method	In-Sample	Out-Sample
0	Linear regression	21.627	24.806
1	Ridge Regression a = 0.01	21.627	24.800
2	Ridge Regression a = 100	23.883	28.048
3	Ridge Regression best a = 0.0404	21.629	24.784

****Self assessed assignment**** check if ridge correction helps to reduce standard error

And then in sample and out of the sample RMSE comes out to be something like, that and then here I am I have written a small piece of code to compare the results ok. So, here four models that we have fitted simple linear regression model, Ridge Regression model with alpha to be 0.01, Ridge Regression with alpha to be 100 and then we did a search on alpha a grid search on alpha and the best alpha on that grid search give us 0.0404.

Now, if you look in samples are all hovering around the same value 21.62 out of the sample is giving us best at 0.04 24.784. The other simple linear regression and ridge regression is not much different, but so we are only able to so; that means, in the ridge correction is not in this particular case ridge correction is not going to make a huge prediction accuracy going to improve much of a prediction accuracy.

If your prediction is the main target when in this particular example ridge correction is not going to improve a lot, but the purpose of ridge correction is not to make a best prediction.

The purpose of ridge correction is to make the reduce the multicollinearity in the data, so that you can do a good statistical inference. So, you have to keep that in mind what is the purpose of the tool. If you hope that you will do ridge correction and; that means, almost all your reduce your predictive accuracy increase your predictive accuracy and reduce your mean square error to significantly that will not happen.

So, the purpose of the ridge correction remember that to reduce the standard error. So, my suggestion is go check in the next you should check try that if the it has be able to reduce the date it is a self assessment assignment; self assessed assignment that check if ridge correction helps to reduce standard error. So, that is the purpose of the ridge regression. With that we will stop here and see you in the next video.