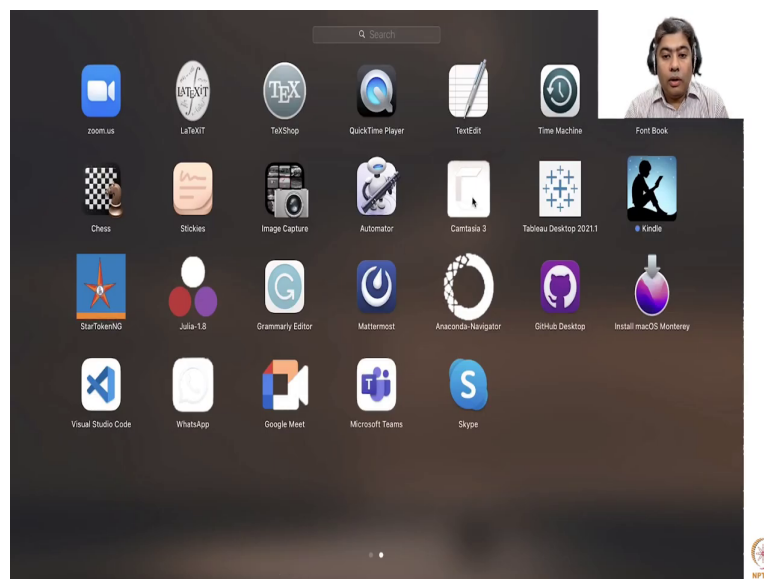


**Predictive Analytics - Regression and Classification**  
**Prof. Sourish Das**  
**Department of Mathematics**  
**Chennai Mathematical Institute**

**Lecture - 03**  
**Hands - on with Python Part - 1**

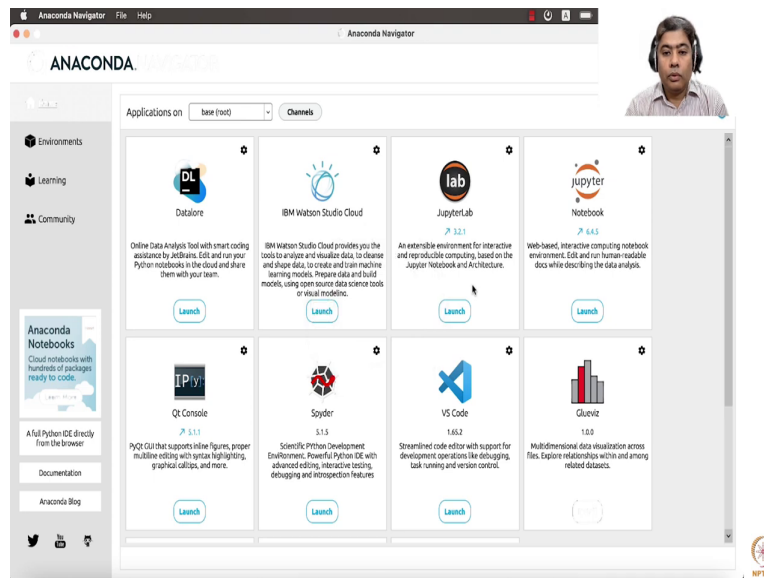
Welcome to the third part in the Predictive Analytics Regression Classification course. So, today we are going to discuss in this particular video that how to estimate betas of that we discussed in part A and part B the betas of the regression line using least square method in Python.

(Refer Slide Time: 00:47)



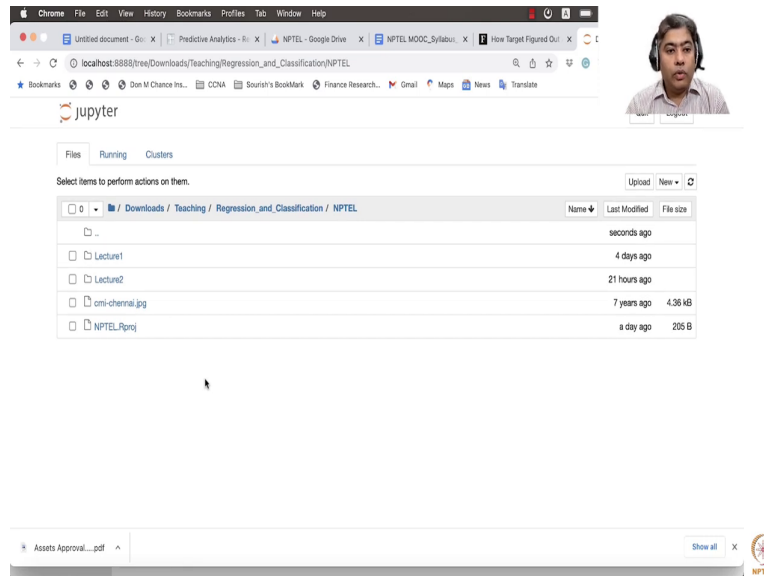
So, what I am going to do I am going to open my conda navigation.

(Refer Slide Time: 00:53)



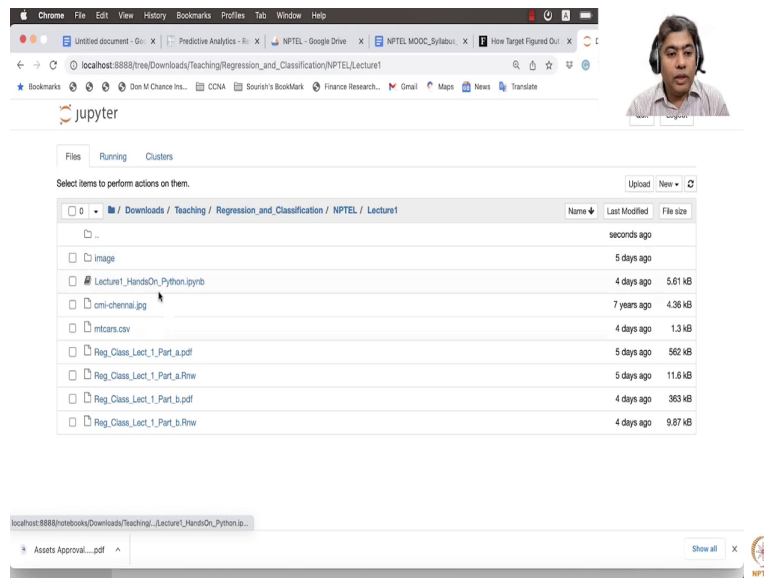
And then from there I will launch the Jupyter Notebook.

(Refer Slide Time: 01:01)



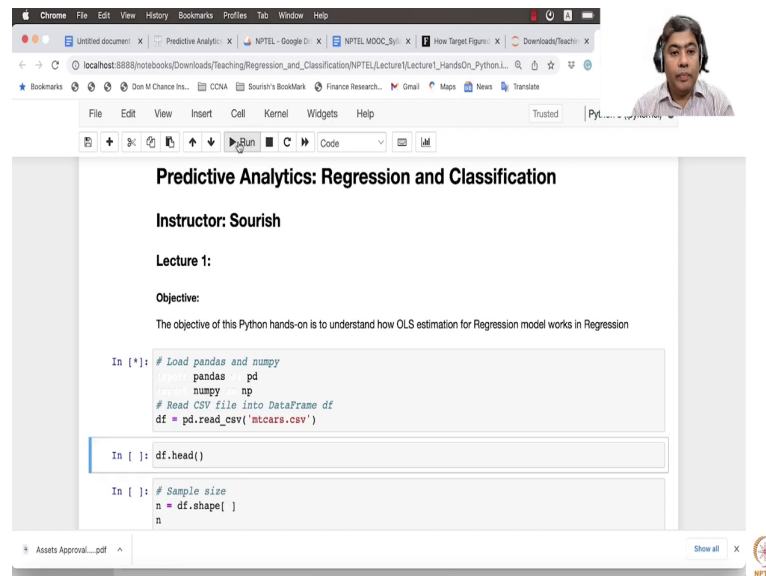
In the Jupyter Notebook I will go to the Lecture 1 of my things.

(Refer Slide Time: 01:09)



And in this folder I have both the Jupyter Notebook ready and as well as the mtcars as a csv file is there.

(Refer Slide Time: 01:28)



The screenshot displays a Jupyter Notebook interface within a web browser. The notebook title is "Predictive Analytics: Regression and Classification". The instructor is identified as Sourish. The current slide is "Lecture 1" with the objective: "The objective of this Python hands-on is to understand how OLS estimation for Regression model works in Regression". The code in the notebook includes:

```
In [*]: # Load pandas and numpy
import pandas as pd
import numpy as np
# Read CSV file into DataFrame df
df = pd.read_csv('mtcars.csv')

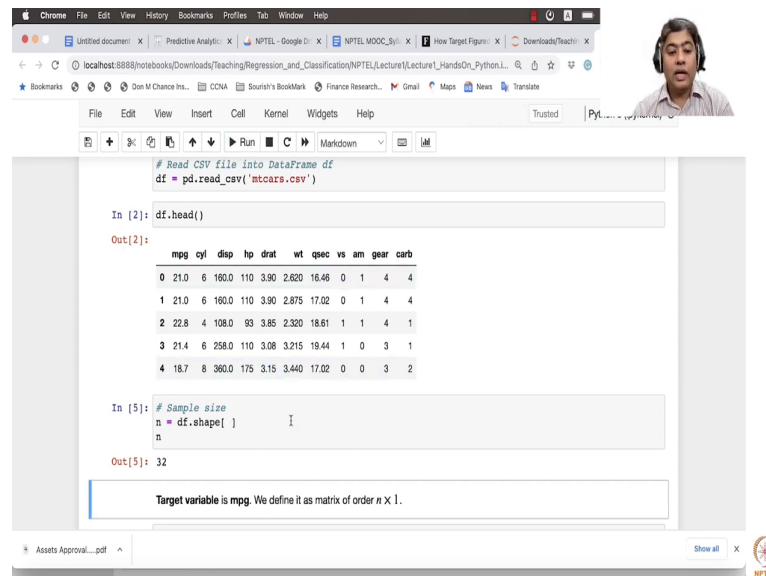
In [ ]: df.head()

In [ ]: # Sample size
n = df.shape[0]
n
```

The interface also shows a video feed of the instructor in the top right corner and an NPTEL logo in the bottom right corner.

So, both the files are there. So, data file and the notebook is there. So, here is the Lecture 1 the objective of this exercise is basically to figure out the using Python how OLS estimation for regression model works in regression. So, first what I have to do that I have to import pandas and numpy. So, pandas as pd import and import numpy as np. Now I have just going to run this, right.

(Refer Slide Time: 02:11)



```
# Read CSV file into DataFrame df
df = pd.read_csv('mtcars.csv')

In [2]: df.head()

Out[2]:
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	21.0	6	160.0	110	3.90	2.820	16.46	0	1	4	4
1	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	18.7	8	380.0	175	3.15	3.440	17.02	0	0	3	2

```
In [5]: # Sample size
n = df.shape[ ]
n

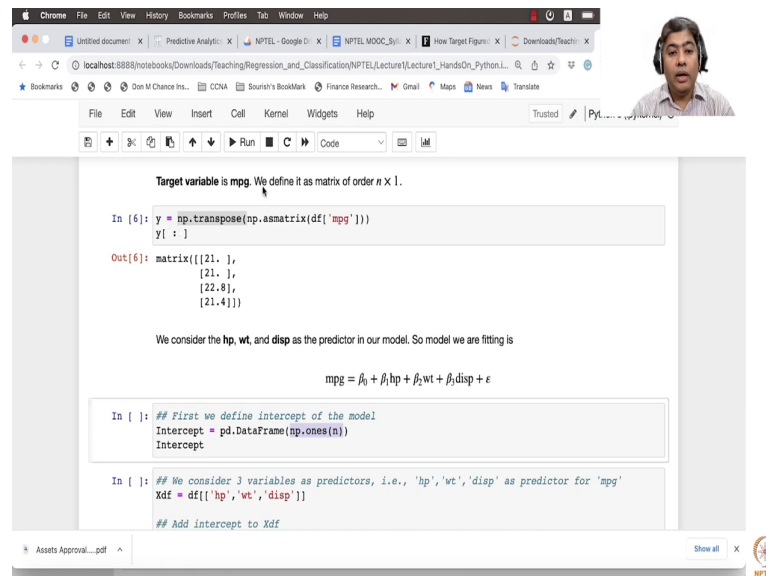
Out[5]: 32

Target variable is mpg. We define it as matrix of order n x 1.
```

And then in df equal to pd.read\_csv and within a quote unquote mtcars.csv is going to read the mtcars state has read the mtcars data set. So, I am just going to run the head. So, you can see it has printed the first five lines miles per gallon, cylinder displacement, horsepower all these lines and then if I just want to read the number of samples that has it has 32 samples if I read that as a df.

So, remember that df is the one data frame and this in this data frame if I just ask give me the shape and the first thing first object of the shape then it will give me the sample size or the number of rows. If I give 1 then it will give you the number of columns, see there are 11 columns. For the time being now we will just in n in the variable n we want to write give number of rows or the number of samples.

(Refer Slide Time: 03:33)



```
Target variable is mpg. We define it as matrix of order n x 1.

In [6]: y = np.transpose(np.asmatrix(df['mpg']))
        yf : 1

Out[6]: matrix([[21. ],
                [21. ],
                [22.8],
                [21.4]])

We consider the hp, wt, and disp as the predictor in our model. So model we are fitting is

mpg =  $\beta_0 + \beta_1 hp + \beta_2 wt + \beta_3 disp + \epsilon$ 

In [ ]: ## First we define intercept of the model
        Intercept = pd.DataFrame(np.ones(n))
        Intercept

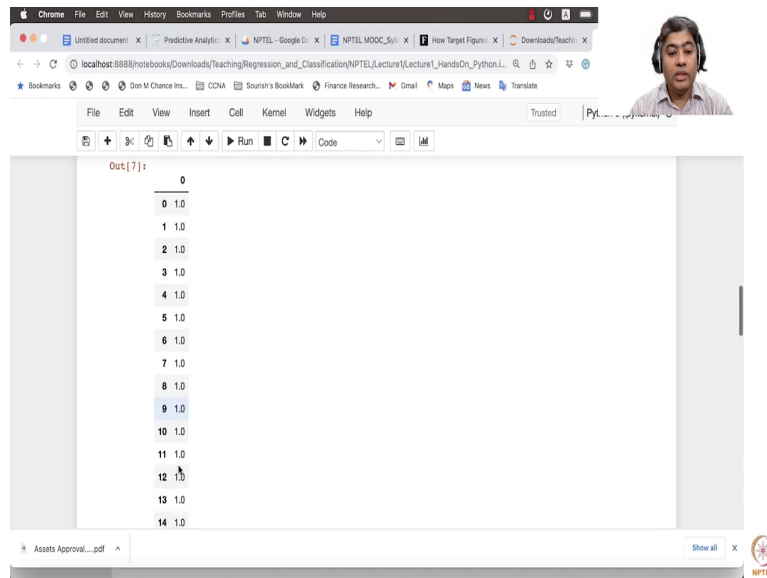
In [ ]: ## We consider 3 variables as predictors, i.e., 'hp', 'wt', 'disp' as predictor for 'mpg'
        Xdf = df[['hp', 'wt', 'disp']]
        ## Add intercept to Xdf
```

So, now, we are going to define the target variable of the response variable mpg. So, we define it as a matrix of order n cross 1. So, df we are going to take the df from the df we are going to take mpg and read that as matrix np.asmatrix will read it as a matrix. And then typically it gives like this kind of format. So, we need to transpose it and. So, that is why I am using np.transpose and then I am putting it in the y variable. So, and then we can see that the first four variable I am showing you here 21, 21, 22.8, 21.4 it is here, but overall this it is a vector of size 32 cross 1.

Next, I am going to fit this miles per gallon as a function of horsepower, weight and displacement. So, the model that I am going to fit is mpg equal to beta naught plus beta 1 horse power plus beta 2 weight plus beta 3 displacement plus a error term sometime it is

called white noise. So, first what I have to do I have to define a intercept terms for that what I will do is I will just going to define a intercept. So, I am saying that, ok.

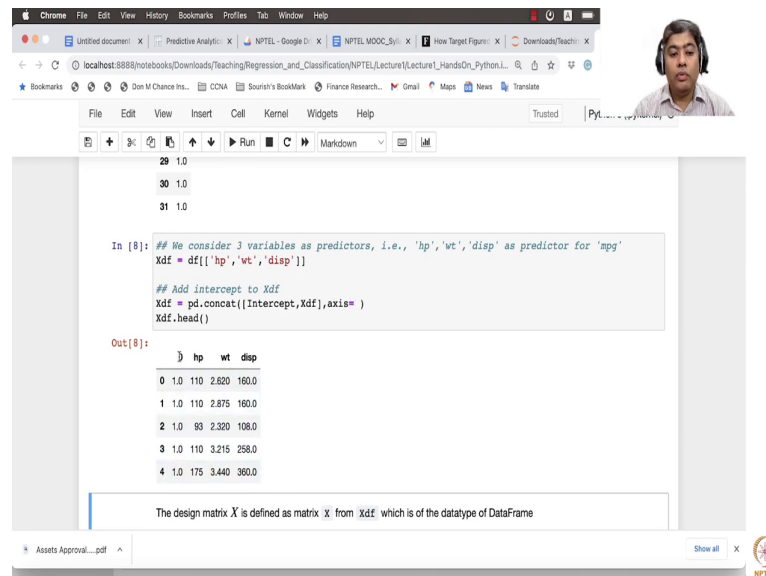
(Refer Slide Time: 05:27)



You just create a vector of ones and then put it in a data frame, ok.



(Refer Slide Time: 05:31)



```
In [8]: ## We consider 3 variables as predictors, i.e., 'hp','wt','disp' as predictor for 'mpg'
Xdf = df[['hp','wt','disp']]

## Add intercept to Xdf
Xdf = pd.concat([Intercept,Xdf],axis= )
Xdf.head()
```

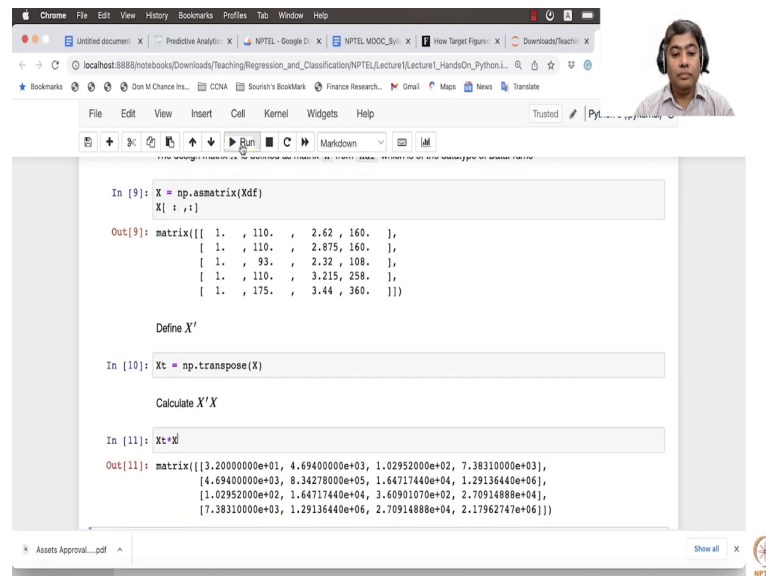
```
Out[8]:
```

	hp	wt	disp	
0	1.0	110	2.620	160.0
1	1.0	110	2.875	160.0
2	1.0	93	2.320	108.0
3	1.0	110	3.215	258.0
4	1.0	175	3.440	360.0

The design matrix  $X$  is defined as matrix  $X$  from `Xdf` which is of the datatype of DataFrame

So, you can see it is created a vector of 1s and put everything in a data frame then from data frame. I am just going to pick these three variables horsepower, weight and displacement and put them as a Xdf. And then I am taking intercept and Xdf join them as a data frame. So, now you can see the first column stands for intercept all once and then horsepower weight and displacement.

(Refer Slide Time: 06:07)



```
In [9]: X = np.asmatrix(Xdf)
X[:, :]
Out[9]: matrix([[ 1. , 110. , 2.62 , 160. ],
 [ 1. , 110. , 2.875, 160. ],
 [ 1. , 93. , 2.32 , 108. ],
 [ 1. , 110. , 3.215, 258. ],
 [ 1. , 175. , 3.44 , 360. ]])

Define X'

In [10]: Xt = np.transpose(X)

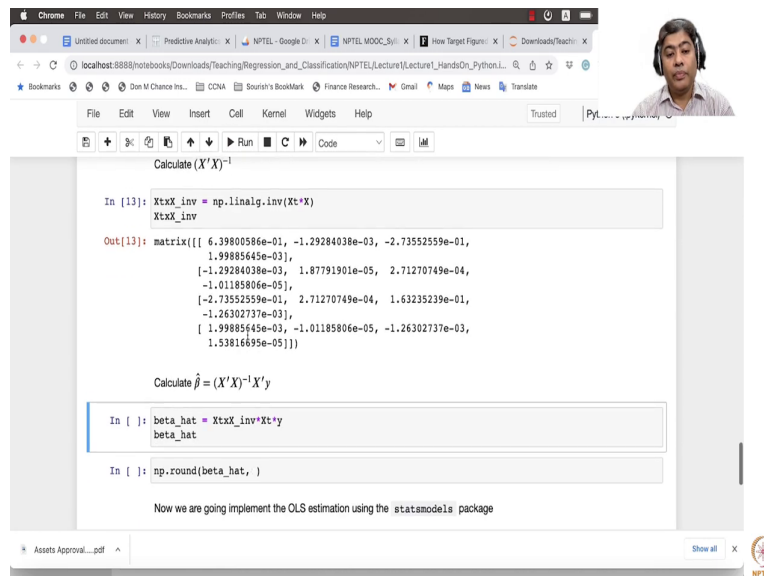
Calculate X'X

In [11]: Xt*X
Out[11]: matrix([[3.20000000e+01, 4.69400000e+03, 1.02952000e+02, 7.38310000e+03],
 [4.69400000e+03, 8.34278000e+05, 1.64717440e+04, 1.29136440e+06],
 [1.02952000e+02, 1.64717440e+04, 3.60901070e+02, 2.70914888e+04],
 [7.38310000e+03, 1.29136440e+06, 2.70914888e+04, 2.17962747e+06]])
```

And this particular data frame is ready to read as my design matrix  $X$  and all I am just doing `np.asmatrix` in `Xdf` and that will be my design matrix  $X$ . So, I am going to run this, right. So, now, you can see the same data frame has read as a matrix and I am just showing here the first five rows of the matrix it is actually a 32 cross 32 rows and 1, 2, 3, 4, 4 columns. So, it is a matrix of size 32 cross 4.

Then what I am going to do? I am going to calculate the take the transpose of this matrix as  $Xt$  and then I am going to calculate  $X$  transpose  $X$ .

(Refer Slide Time: 07:11)



```
Calculate  $(X'X)^{-1}$ 

In [13]: XtX_inv = np.linalg.inv(Xt*X)
XtX_inv

Out[13]: matrix([[ 6.39800596e-01, -1.29284038e-03, -2.73552559e-01,
 1.99885645e-03],
 [-1.29284038e-03, 1.87791901e-05, 2.71270749e-04,
 -1.01185806e-05],
 [-2.73552559e-01, 2.71270749e-04, 1.63235239e-01,
 -1.26302737e-03],
 [ 1.99885645e-03, -1.01185806e-05, -1.26302737e-03,
 1.53816695e-05]])

Calculate  $\hat{\beta} = (X'X)^{-1}X'y$ 

In [ ]: beta_hat = XtX_inv*Xt*y
beta_hat

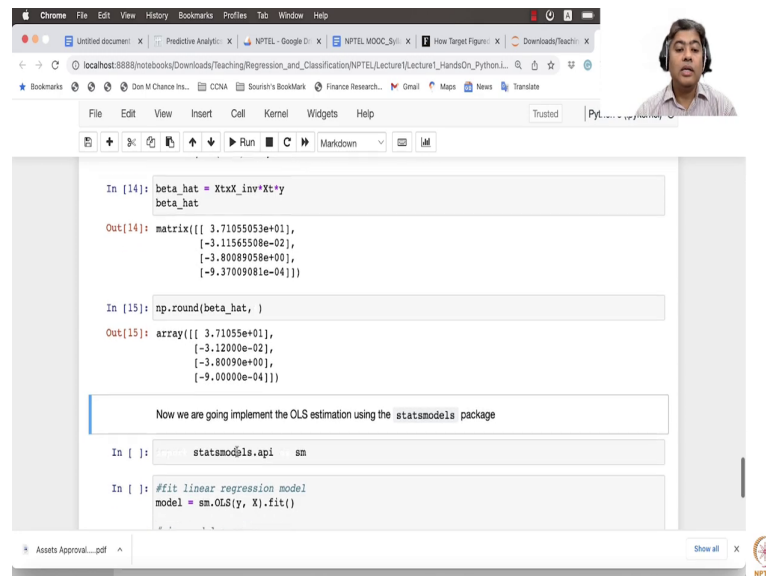
In [ ]: np.round(beta_hat, )

Now we are going to implement the OLS estimation using the statsmodels package
```

So, here  $X^T X$  is a 4 cross 4 matrix. So,  $X$  is a 32 cross 4. So,  $X^T$  will be 4 cross 32. So,  $X^T X$  will be 4 cross 4 matrix. So, my  $X^T X$  is 4 cross 4 matrix. Then I am going to take the inverse of that matrix  $X^T X$  and calculate  $X^T X$  inverse, ok. And then what I am going to do is yeah let me just calculate that.

So,  $X^T X$  inverse has this  $X^T X$  inverse. In fact, you can print it should not be a very difficult thing. Let me run it once more and you can see this is the  $X^T X$  inverse. Now remember that from my previous lectures of part A and part B previous part that  $\hat{\beta}$  is  $X^T X$  inverse  $X^T y$ . So, now, I have calculated  $X^T X$  inverse then it I just have to multiply it  $X^T$  and  $y$  that will give me the  $\hat{\beta}$ .

(Refer Slide Time: 08:24)



```
In [14]: beta_hat = XtX_inv*Xt*y
beta_hat
Out[14]: matrix([[ 3.71055053e+01],
                [-3.11565508e-02],
                [-3.80089058e+00],
                [-9.37009081e-04]])

In [15]: np.round(beta_hat, 4)
Out[15]: array([[ 3.71055e+01],
                [-3.12000e-02],
                [-3.80090e+00],
                [-9.00000e-04]])
```

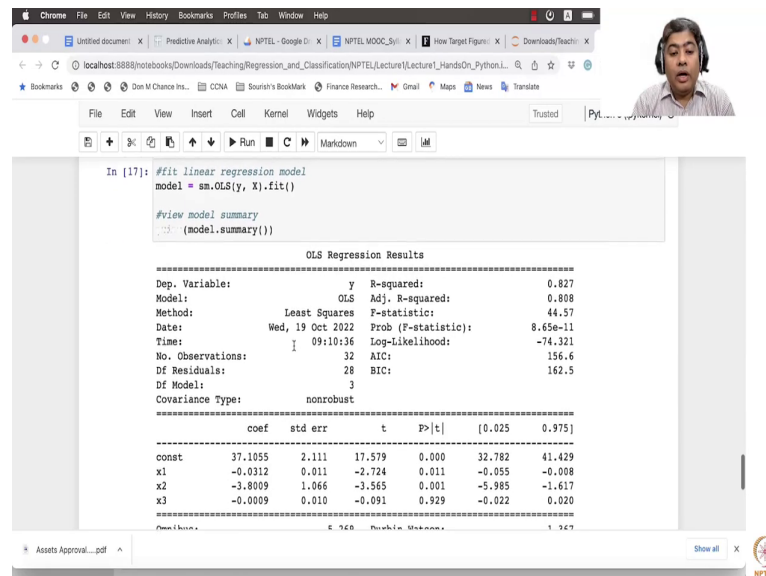
Now we are going to implement the OLS estimation using the statsmodels package

```
In [ ]: statsmodels.api  sm

In [ ]: #fit linear regression model
model = sm.OLS(y, X).fit()
```

So, if we do that. So, now, you can see this is my beta hat. Now I am just rounding it off to four decimal place this beta hat to just. So, that you know 3.71055 negative 3. So, this will be yeah. So, this is actually 37.1055 this is 0.00312 this is 3 negative 3.8 and this is negative 0.0009.

(Refer Slide Time: 08:58)



The screenshot shows a Jupyter Notebook interface with a Chrome browser window in the background. The notebook cell contains the following Python code:

```
In [17]: #fit linear regression model
model = sm.OLS(y, X).fit()

#view model summary
model.summary()
```

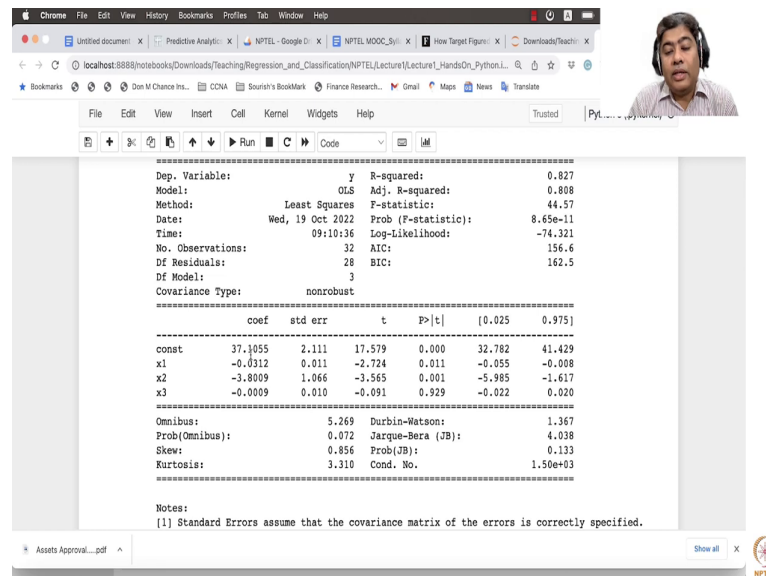
The output of the code is a detailed OLS Regression Results summary:

```
OLS Regression Results
=====
Dep. Variable:          y      R-squared:            0.827
Model:                  OLS    Adj. R-squared:       0.808
Method:                 Least Squares  F-statistic:         44.57
Date:                   Wed, 19 Oct 2022  Prob (F-statistic):    8.65e-11
Time:                   09:10:36    Log-Likelihood:     -74.321
No. Observations:      32          AIC:                156.6
Df Residuals:          28          BIC:                162.5
Df Model:               3
Covariance Type:       nonrobust
=====
               coef      std err          t      Pr>|t|    [0.025    0.975]
-----
const      37.1055      2.111      17.579   0.000    32.782    41.429
x1         -0.0312      0.011      -2.724   0.011    -0.055    -0.008
x2         -3.8009      1.066      -3.565   0.001    -5.985    -1.617
x3         -0.0009      0.010      -0.091   0.929    -0.022    0.020
=====
```

So, this is how you generally the beta hat estimation happens. But when going forward we are not going to implement every time like  $X^T X^{-1}$  and all the  $a$  then I am this will be little bit cumbersome. So, what we are going to do we are going to implement this OLS estimation using the stat models package in Python.

So, we are going to implement you know load the statmodels.api package as sm. So, I am going to run this and then from sm we are going to call OLS and provide the y vector and the X matrix and then just I am calling fit it will works just like the you know and very intuitively you can work with this. And then fit the model and then from the model you call the summary it will give you all the necessary summary for this thing.

(Refer Slide Time: 10:10)



```
Dep. Variable: y R-squared: 0.827
Model: OLS Adj. R-squared: 0.808
Method: Least Squares F-statistic: 44.57
Date: Wed, 19 Oct 2022 Prob (F-statistic): 8.65e-11
Time: 09:10:36 Log-Likelihood: -74.321
No. Observations: 32 AIC: 156.6
Df Residuals: 28 BIC: 162.5
Df Model: 3
Covariance type: nonrobust

=====
coef std err t P>|t| [0.025 0.975]
-----+-----+-----+-----+-----
const 37.1055 2.111 17.579 0.000 32.782 41.429
x1 -0.0312 0.011 -2.724 0.011 -0.055 -0.008
x2 -3.8009 1.066 -3.565 0.001 -5.985 -1.617
x3 -0.0009 0.010 -0.091 0.929 -0.022 0.020
=====

Omnibus: 5.269 Durbin-Watson: 1.367
Prob(Omnibus): 0.072 Jarque-Bera (JB): 4.038
Skew: 0.856 Prob(JB): 0.133
Kurtosis: 3.310 Cond. No. 1.50e+03
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

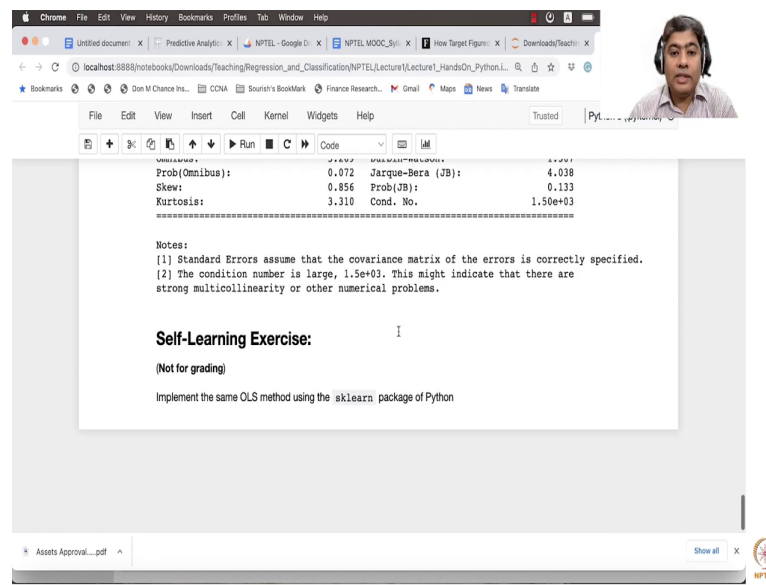
Now, you look at the there are lot of summary is being provided by stat models. In today's class we will not be able to discuss all the elements of this summary. But over the next you know 3 months we are going to learn all the summary of the regression models.

So, today we are going to discuss only the coefficient. Now coefficient first constant or the intercept is 37.1055. Now if you go back and this is you can see that it is 37.1055 indeed. Then if you look into the second one it is negative 0.0312 and if you look into this it is indeed negative 0.0312 then if you look into so the third one negative 3.8009. So, this is negative 3.8009.

Now we go back it is indeed negative 3.8009 because e to the power 00 it is just 1 basically. And then last one is negative after 4 0 there is a 9 and you can see in the fourth place after three 0's there is a 9 and yeah, it is here it is also after three 0's in the fourth place there will

be a 9 after decimal. So, you can see that OLS estimation that we did you know analytically we solve that analytics OLS estimation that is already provided in the statsmodel package. So, we can just simply call statsmodel package and work with it.

(Refer Slide Time: 12:05)



The screenshot shows a Jupyter Notebook interface with a Chrome browser window in the background. The notebook displays the following output:

```
Prob(Omnibus):    0.072    Jarque-Bera (JB):    4.038
Skew:            0.856    Prob(JB):            0.133
Kurtosis:        3.310    Cond. No.            1.50e+03
=====
```

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.5e+03. This might indicate that there are strong multicollinearity or other numerical problems.

**Self-Learning Exercise:**

(Not for grading)

Implement the same OLS method using the `sklearn` package of Python

Here is a self learning exercise it is not for grading you should try for yourself. Now in Python sklearn is a very popular package. In sklearn also you can fit OLS method for a simple linear regression models. So, your job is to implement OLS method using sklearn package of Python.

So, try yourself and check if the coefficient values are indeed matching with this numbers if your coefficient values from the sklearn are matching indeed matching with these numbers; that means, you have done it correctly. But if it is not matching with these numbers then you have not done it correctly something you must have made some mistakes. So, try yourself

good luck. In the next video we are going to discuss how to implement the same things using R.

Take care bye.