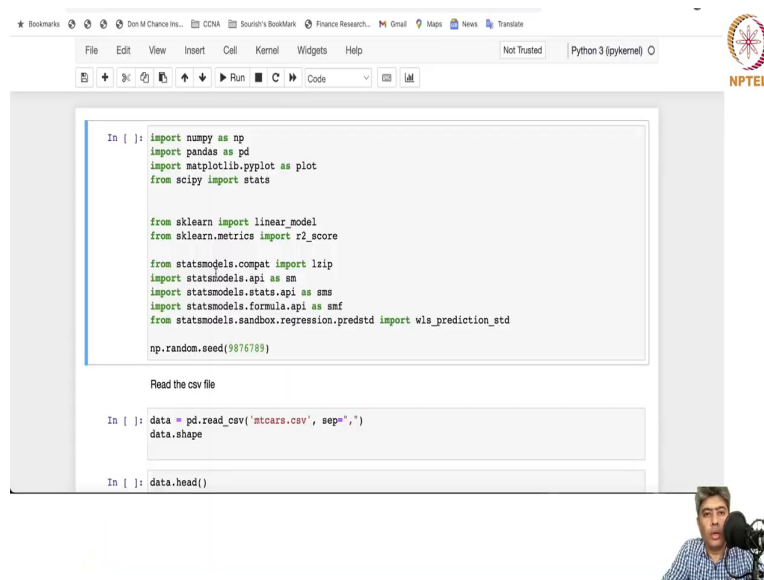


Predictive Analytics - Regression and Classification
Prof. Sourish Das
Department of Mathematics
Chennai Mathematical Institute

Lecture - 25
Hands-on with Python Part - 3

Hello guys, welcome back to hands on for lecture 7, in this hands on we are going to do some regression analysis using Python.

(Refer Slide Time: 00:33)



The image shows a Jupyter Notebook interface with a browser window at the top. The notebook contains the following code:

```
In [ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plot
from scipy import stats

from sklearn import linear_model
from sklearn.metrics import r2_score

from statsmodels.compat import lzip
import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.formula.api as smf
from statsmodels.sandbox.regression.predstd import wls_prediction_std

np.random.seed(9876789)

Read the csv file

In [ ]: data = pd.read_csv('mtcars.csv', sep=',')
data.shape

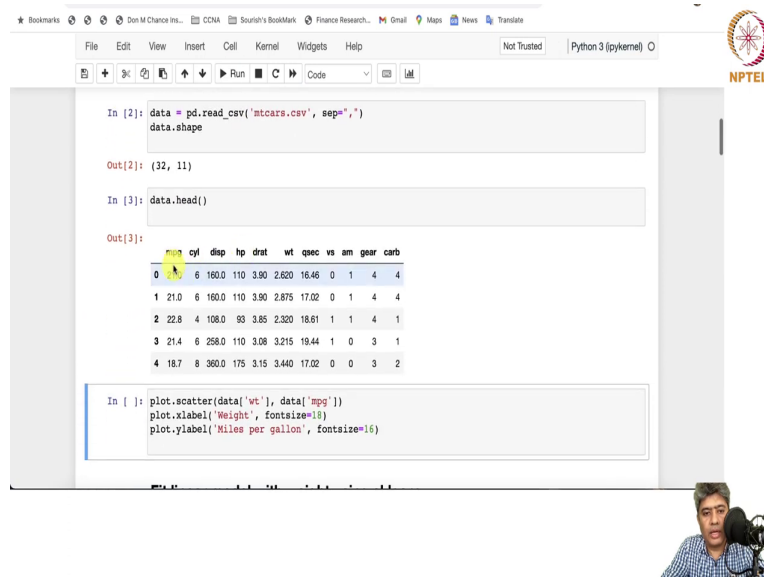
In [ ]: data.head()
```

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

So, let me start my Jupyter Notebook ok; so, in this we are going to use the mtcars dataset and here is the bunch of packages that I have called you know here I have bunch of packages numpy, pandas, matplotlib, library, scipy from sklearn, we imported linear models; from

sklearns matrices, we calculated r2 score. Then some other models from statmodels stats api. So, all these things we have computed, let me run it has no problem.

(Refer Slide Time: 01:30)



```
In [2]: data = pd.read_csv('mtcars.csv', sep=',')
data.shape

Out[2]: (32, 11)

In [3]: data.head()

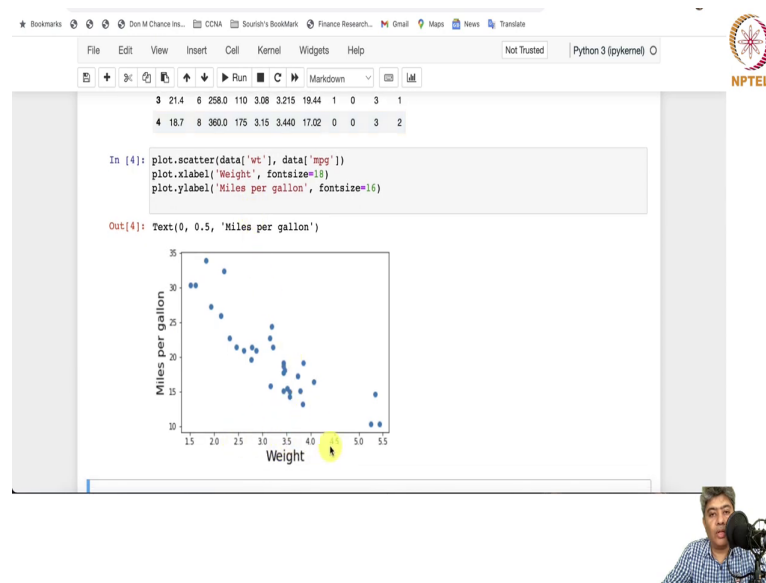
Out[3]:
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	16.0	6	160.0	110	3.90	2.620	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	360.0	175	3.15	3.440	17.02	0	0	3	2

```
In [ ]: plot.scatter(data['wt'], data['mpg'])
plot.xlabel('Weight', fontsize=18)
plot.ylabel('Miles per gallon', fontsize=16)
```

So, next is read the mtcars dataset, we created and it has 32 rows, 11 column as expected and head. So, it has first column is miles per gallon, second column is cylinder, third column is displacement, fourth column is horsepower, rears ratio, weight qsec, v shape or knot. automatic or manual, how many gears it has and carburettor.

(Refer Slide Time: 02:01)



So, the first we are going to plot weight versus miles per gallon and as expected on the x axis we put a weight and a on the y axis we put gallon miles per gallon. And it there is a negative relationship what we are seeing is as the weight of the car increases efficiency of the car drops. So, first we are going to fit a linear model with weight using a sklearn.

(Refer Slide Time: 02:30)

```
Fit linear model with weight using sklearn:

mpg = a + b weight + error
y = a + b x + error

In [ ]: x = data[['wt']]
        y = data[['mpg']]
        lm = linear_model.LinearRegression()
        fit_ml = lm.fit(X=x, y=y)

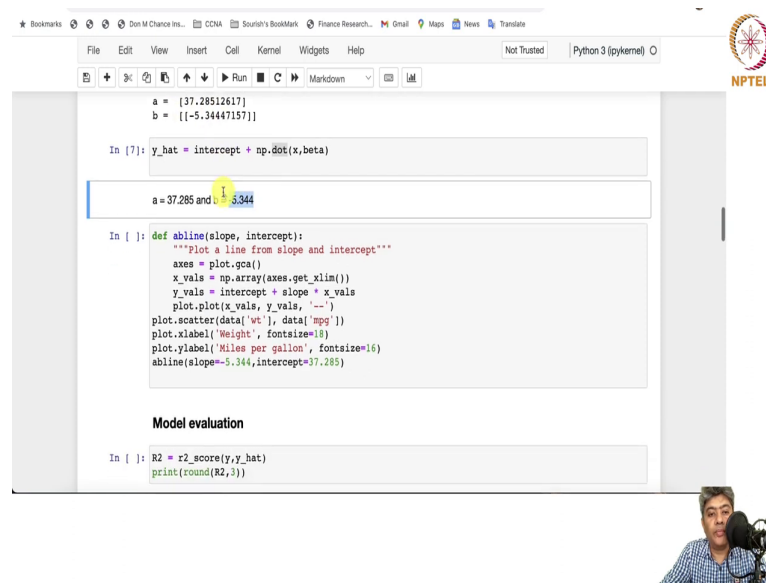
In [ ]: intercept = fit_ml.intercept_
        print('a = ', intercept)
        beta = fit_ml.coef_
        print('b = ', beta)

In [ ]: y_hat = intercept + np.dot(x, beta)

a = 37.285 and b = -5.344
```

So, first; so, we are going to fit miles per gallon as a function of a plus b times weight plus error. So, x is in the weight and y as the miles per gallon and from the linear model we are calling the linear regression module. Defining it as a lm and then we are going to fit lm fit and x we have to give this x and for all y we have to give this y, and that will be fitted as a first model ok.

(Refer Slide Time: 03:18)



```
★ Bookmarks  Don M Chance ins...  CCNA  Sourin's BookMark  Finance Reseach...  Gmail  Maps  News  Translate
File  Edit  View  Insert  Cell  Kernel  Widgets  Help  Not Trusted  Python 3 (pykernel) O
a = [37.28512617]
b = [[-5.34447157]]

In [7]: y_hat = intercept + np.dot(x,beta)

a = 37.285 and b = -5.344

In [ ]: def abline(slope, intercept):
        """Plot a line from slope and intercept"""
        axes = plot.gca()
        x_vals = np.array(axes.get_xlim())
        y_vals = intercept + slope * x_vals
        plot.plot(x_vals, y_vals, '-')
        plot.scatter(data['wt'], data['mpg'])
        plot.xlabel('Weight', fontsize=18)
        plot.ylabel('Miles per gallon', fontsize=16)
        abline(slope=-5.344,intercept=37.285)

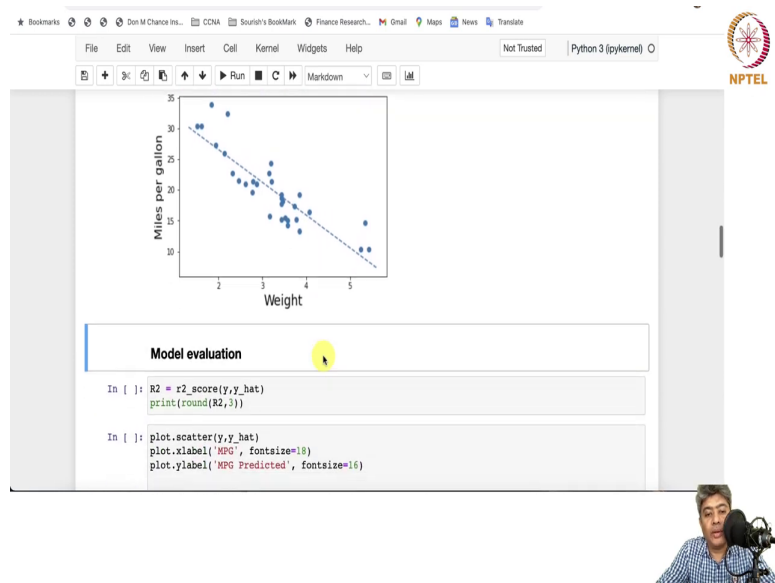
Model evaluation

In [ ]: R2 = r2_score(y,y_hat)
        print(round(R2,3))
```

So, it has done it has done and let from the fit_m1, here it is; we are extracting the intercept and calling it intercept and the coefficient beta and we take it as b equal to a equal to b equal to. So, we got a equal to 37.2 a and b equal to negative 5.344 and we are going to calculate y hat equal to intercept, plus in dot operation we are going to do x into beta.

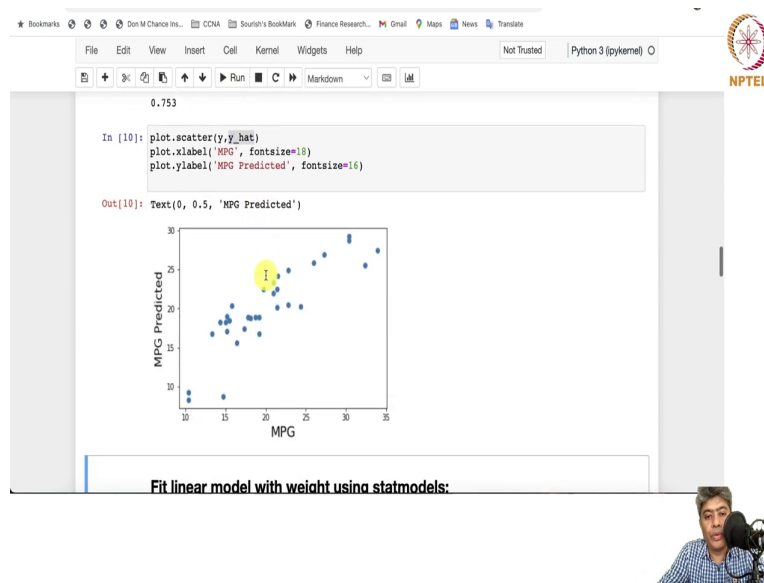
So, but it is a dot operation from; so, from numpai, we are going to call the dot operation. So, we ran it we took a equal to 37.285 and b equal to negative 5.344 from here. And then we are going to draw a straight line abline sometimes called through the plots. So, this we have written this small piece of function which, if you go give the slope value and the intercept value, it will plot the line from slope and intercept over the scatter plot.

(Refer Slide Time: 04:39)



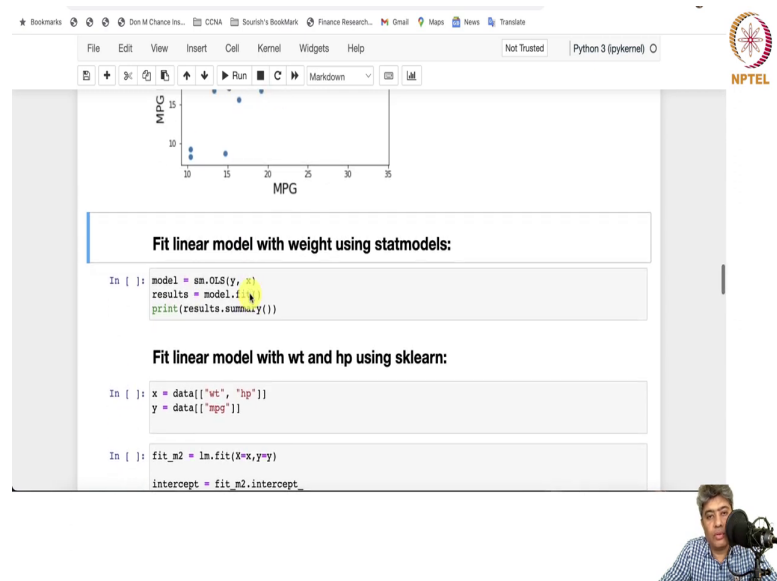
So, we run this; so, over the scatter plot we draw a straight line, and this is the best fitted OLS line.

(Refer Slide Time: 04:50)



Now, we are fitting a model evaluation; now, we are fitting r square, r square is 0.753. And then from the we have y and y hat; we plot y and y hat, y is the miles per original miles per gallon and y hat is the predicted miles per gallon or estimated miles per gallon. So, we can see that they are positively correlated; that means, the cars which has lower efficiency it is predicted as lower. But definitely there are some variability, more time it will be better the model prediction is we can say.

(Refer Slide Time: 05:36)



The screenshot shows a Jupyter Notebook interface with a scatter plot and two code blocks. The scatter plot displays MPG on the y-axis (ranging from 10 to 15) and an unlabeled x-axis (ranging from 10 to 35). The code blocks demonstrate fitting linear models using statsmodels and sklearn.

Fit linear model with weight using statmodels:

```
In [ ]: model = sm.OLS(y, X)
        results = model.fit()
        print(results.summary())
```

Fit linear model with wt and hp using sklearn:

```
In [ ]: x = data[["wt", "hp"]]
        y = data[["mpg"]]

In [ ]: fit_m2 = lm.fit(X=x, y=y)
        intercept = fit_m2.intercept_
```

The interface includes a browser address bar at the top with various bookmarks, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a toolbar with icons for file operations and execution. The Python 3 (ipykernel) environment is indicated in the top right corner. An NPTEL logo is visible in the bottom right corner of the notebook area.

So, now fit linear model using weight using stat model; so, that was we fit this model using; so, when we fit this model in the above yeah, here this was done using sk learn. Now, we are going to fit the same model using stat models; so, sm OLS, sm dot OLS.

(Refer Slide Time: 06:10)

Fit linear model with weight using statmodels:

```
In [11]: model = sm.OLS(y, x)
results = model.fit()
print(results.summary())
```

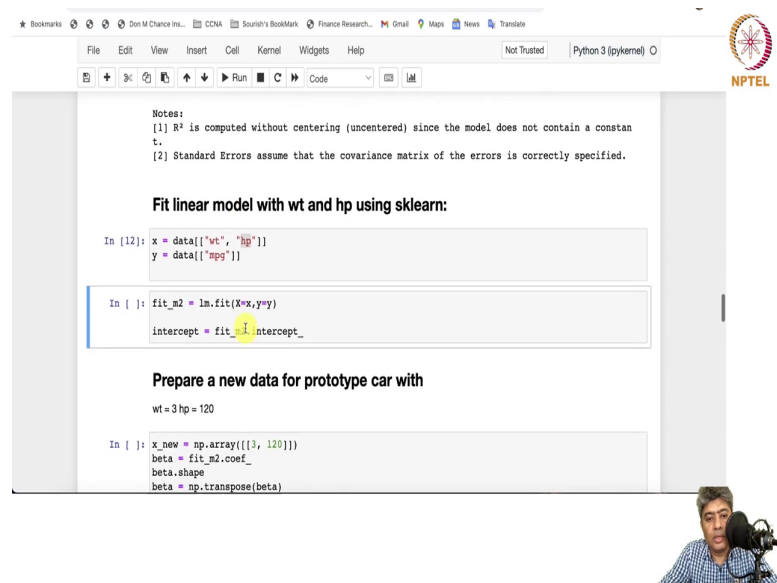
OLS Regression Results

Dep. Variable:	mpg	R-squared (uncentered):	0.720
Model:	OLS	Adj. R-squared (uncentered):	0.711
Method:	Least Squares	F-statistic:	79.58
Date:	Mon, 30 Jan 2023	Prob (F-statistic):	4.55e-10
Time:	16:17:21	Log-Likelihood:	-122.40
No. Observations:	32	AIC:	246.8
Df Residuals:	31	BIC:	248.3
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
wt	5.2916	0.593	8.921	0.000	4.082	6.501

Omnibus: 0.255 Durbin-Watson: 0.833
Prob(Omnibus): 0.890 Jarque-Bera (JB): 0.317
Skew: 0.189 Prob(JB): 0.854
Kurtosis: 2.692 Cond. No. 1.00

(Refer Slide Time: 06:20)



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

```
Notes:  
[1] R2 is computed without centering (uncentered) since the model does not contain a constant.  
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Fit linear model with wt and hp using sklearn:

```
In [12]: x = data[['wt', 'hp']]  
         y = data[['mpg']]
```

```
In [ ]: fit_m2 = lm.fit(X=x, y=y)  
        intercept = fit_m2.intercept_
```

Prepare a new data for prototype car with

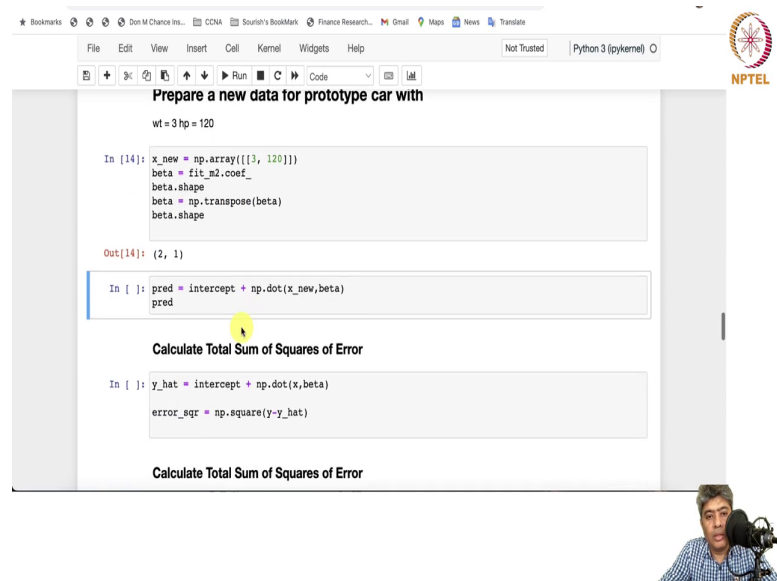
```
wt = 3  
hp = 120
```

```
In [ ]: x_new = np.array([[3, 120]])  
        beta = fit_m2.coef_  
        beta.shape  
        beta = np.transpose(beta)
```

The interface includes a browser window at the top with various tabs and a toolbar with icons for file operations, running, and code execution. The NPTEL logo is visible in the top right corner.

We if we run that, it gives print out more like a you know like a r kind of print out. Now, along with weight we want to put the horsepower; so, in the x we put along with weight we put the horsepower and we fit the model. Now, we are preparing a new data with a new prototype car with weight equal to 3 and horsepower equal to 120.

(Refer Slide Time: 06:47)



```
wt = 3 hp = 120

In [14]: x_new = np.array([[3, 120]])
         beta = fit_m2_coef_
         beta.shape
         beta = np.transpose(beta)
         beta.shape

Out[14]: (2, 1)

In [ ]: pred = intercept + np.dot(x_new, beta)
         pred

Calculate Total Sum of Squares of Error

In [ ]: y_hat = intercept + np.dot(x, beta)
         error_sqr = np.square(y - y_hat)

Calculate Total Sum of Squares of Error
```

So, in that we run; so, we got the alpha then beta; now, beta we have 2 coefficients; so, that is why it is 2 cross 1.

(Refer Slide Time: 07:00)

```
Out[15]: array([[21.78102425]])

Calculate Total Sum of Squares of Error

In [16]: y_hat = intercept + np.dot(x,beta)
         error_sqr = np.square(y-y_hat)

Calculate Total Sum of Squares of Error

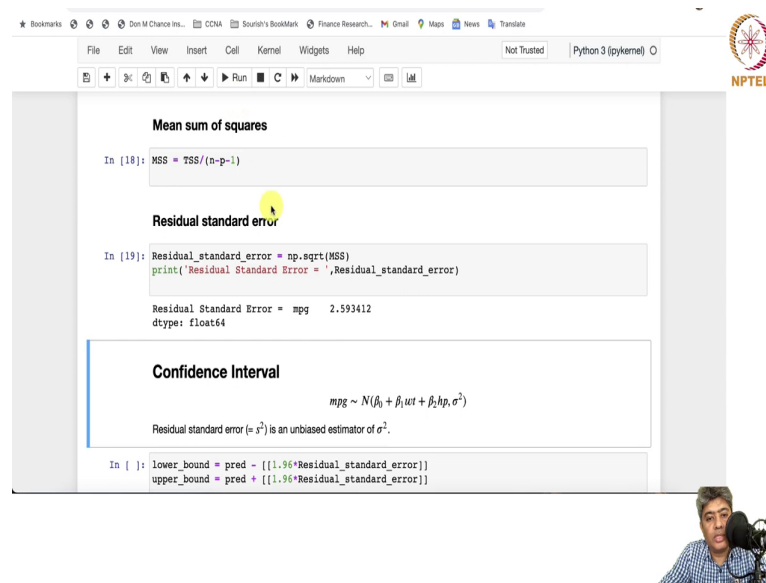
In [ ]: TSS = np.sum(error_sqr)
         n = x.shape[0]
         p = x.shape[1]

Mean sum of squares

In [ ]: MSS = TSS/(n-p-1)
```

Now, predicted values will be for that for a new prototype car which has 3 weights. And how weight equal to 3 meat and horsepower equal to 120 units. We will have a miles per gallon 20 mile 1.78. So, we can calculate the total sum of square; first you calculate y hat then y minus y hat and that. If you square it that will you give you error sum of squares ok, and then if you just take sum that will give you the total sum of squares.

(Refer Slide Time: 07:51)



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

- Mean sum of squares**
In [18]: `MSS = TSS/(n-p-1)`
- Residual standard error**
In [19]: `Residual_standard_error = np.sqrt(MSS)`
`print('Residual Standard Error = ', Residual_standard_error)`
Residual Standard Error = mpg 2.593412
dtype: float64
- Confidence Interval**
$$mpg \sim N(\beta_0 + \beta_1 wt + \beta_2 hp, \sigma^2)$$

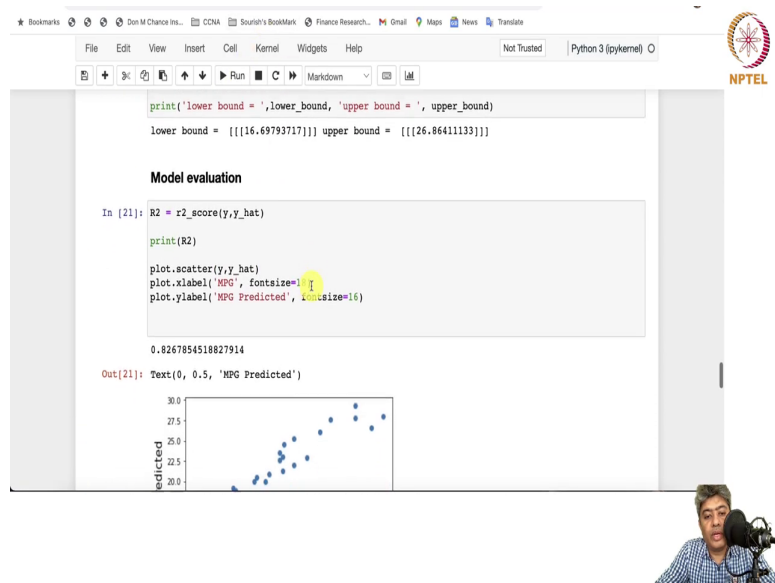
Residual standard error ($= s^2$) is an unbiased estimator of σ^2 .
In []: `lower_bound = pred - [1.96*Residual_standard_error]`
`upper_bound = pred + [1.96*Residual_standard_error]`

A small video inset of a person speaking is visible in the bottom right corner of the notebook interface.

And then if you just calculate divide that by n minus p minus 1 that will give you the mean sum of square. So, the residual standard error is just take the square root of mean sum of square that will give you the residual standard error ok. Now, so, the residual standard error for this model is 2.59. Now, miles per gallon; that means, for a normal beta naught plus beta 1 times weight plus beta 2 times horsepower comma sigma square.

So, residual standard error is an unbiased estimator of the sigma square. So, you can calculate the lower bound as predicted value minus 1.96 times residual standard error and upper bound would be predicted value plus 1.96 times residual standard error and we print the lower bound and the upper bound.

(Refer Slide Time: 08:53)



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

```
print('lower bound = ',lower_bound, 'upper bound = ', upper_bound)
lower bound = [[[16.69793717]]] upper bound = [[[26.8641133]]]
```

Model evaluation

```
In [21]: R2 = r2_score(y,y_hat)
print(R2)
plot.scatter(y,y_hat)
plot.xlabel('MPG', fontsize=16)
plot.ylabel('MPG Predicted', fontsize=16)
```

0.8267854518827914

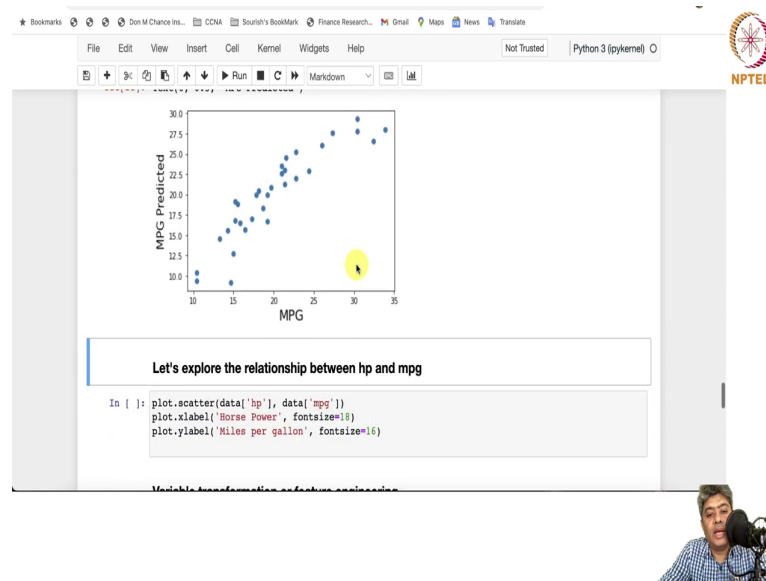
```
Out[21]: Text(0, 0.5, 'MPG Predicted')
```

The plot shows a scatter plot of predicted MPG values (y-axis) versus actual MPG values (x-axis). The y-axis is labeled 'MPG Predicted' and ranges from 20.0 to 30.0. The x-axis is labeled 'MPG' and ranges from 20.0 to 30.0. The data points are blue dots showing a strong positive correlation. A yellow circle highlights the number '16' in the code for plot.xlabel('MPG', fontsize=16).

NPTEL logo is visible in the top right corner of the notebook interface.

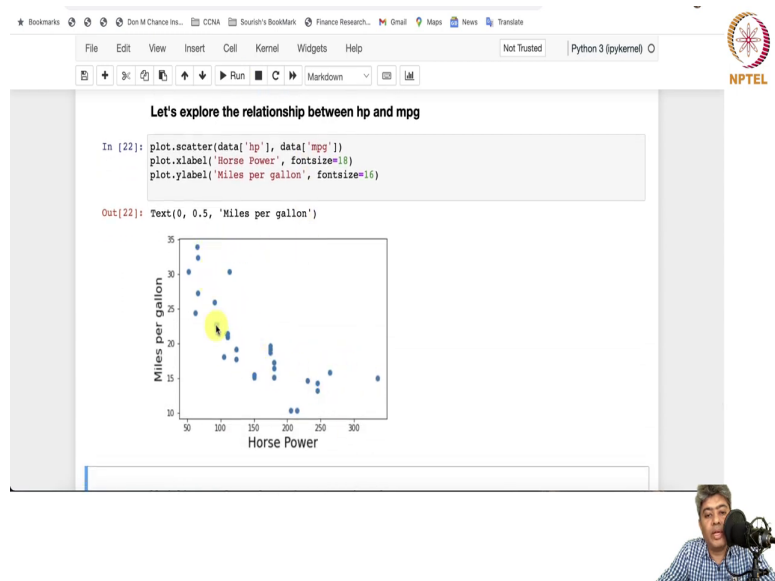
So, the lower bound will be 16.69 and upper bound will be 26.86.

(Refer Slide Time: 09:03)



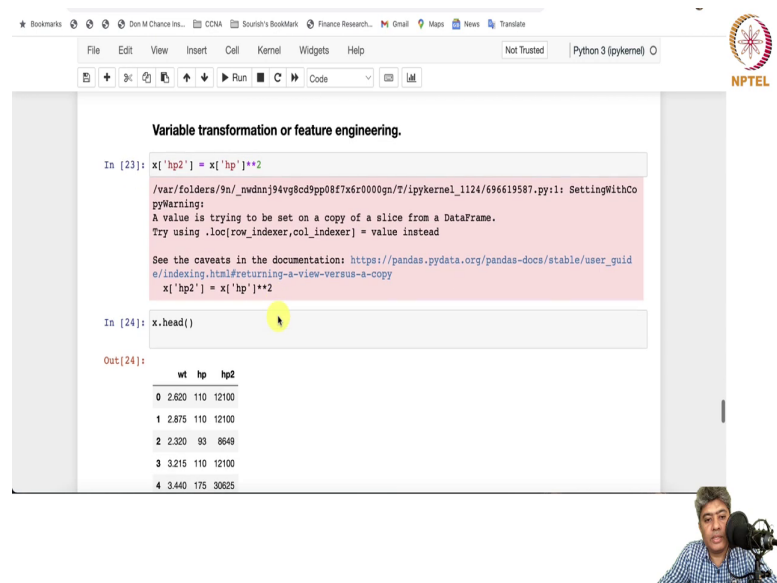
So, for next we fit a you know r square is 82.67 percent; so, along with weight when we put horsepower, the r square jump to 82.6 percent and it becomes slightly tighter.

(Refer Slide Time: 09:25)



Now, let us explore the relationship between horsepower and miles per gallon. So, what we are seeing the relationship is maybe there is a quadratic relationship.

(Refer Slide Time: 09:40)



The screenshot shows a Jupyter Notebook interface with a Python 3 (ipykernel) environment. The notebook title is "Variable transformation or feature engineering." The code cell contains the following code:

```
In [23]: x['hp2'] = x['hp']**2
```

A warning message is displayed below the code:

```
/var/folders/5n/_nwdnnj94vg8cd9pp08f7x6r0000gn/T/ipykernel_1124/696619587.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead


See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
x['hp2'] = x['hp']**2
```

The code cell is then executed, and the output is displayed:

```
In [24]: x.head()
```

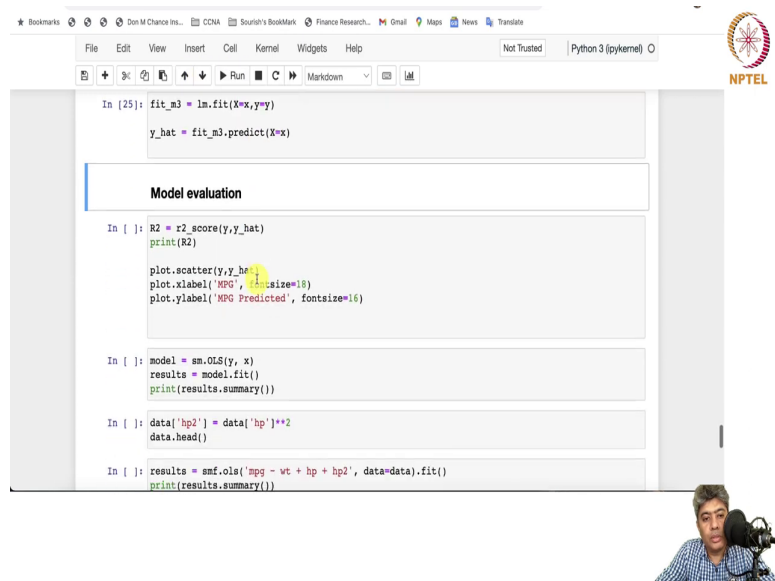
```
Out[24]:
```

	wt	hp	hp2
0	2.620	110	12100
1	2.875	110	12100
2	2.320	93	8649
3	3.215	110	12100
4	3.440	175	30625



So, we have to do some feature engineering; so, we add a new column by simply taking the square of the column. And then \hat{x} is \hat{x} is to have weight horsepower and horsepower square.

(Refer Slide Time: 09:57)



```
In [25]: fit_m3 = lm.fit(X=x,y=y)
y_hat = fit_m3.predict(X=x)

Model evaluation



In [ ]: R2 = r2_score(y,y_hat)
print(R2)

plot.scatter(y,y_hat)
plot.xlabel('MPG', fontsize=18)
plot.ylabel('MPG Predicted', fontsize=16)

In [ ]: model = sm.OLS(y, x)
results = model.fit()
print(results.summary())

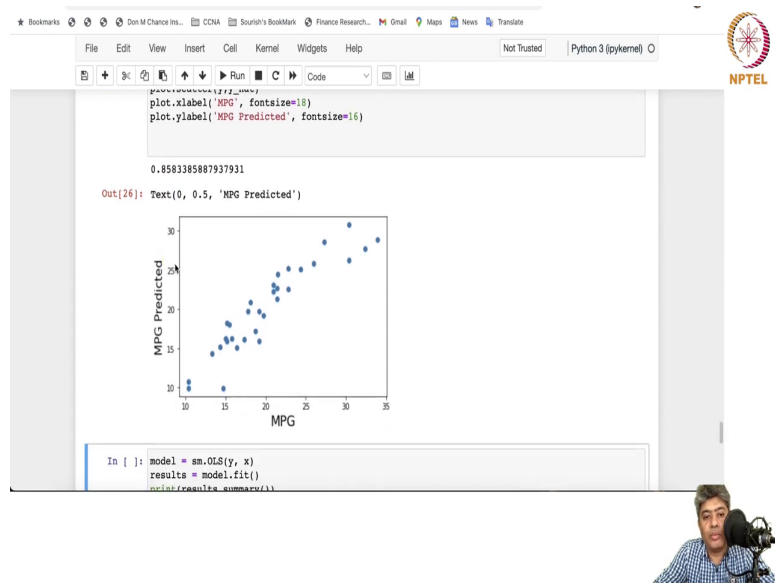
In [ ]: data['hp2'] = data['hp']**2
data.head()

In [ ]: results = smf.ols('mpg ~ wt + hp + hp2', data=data).fit()
print(results.summary())
```



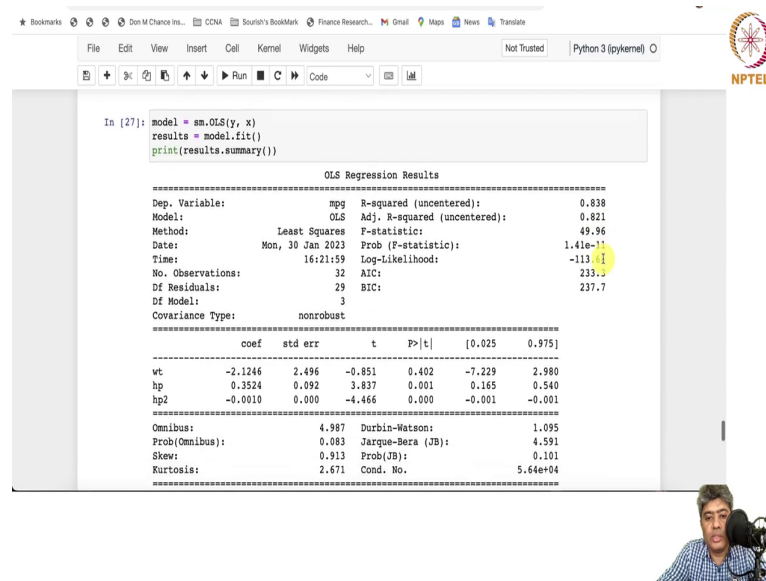
Now, we can fit the third model by just providing the proper x and y and use the predict to get the predicted values.

(Refer Slide Time: 10:13)



So, the model evaluation when we do the model evaluation, what we find that the whole and you know the r square be has gone up to 85.8 percent and it is becoming more tighter.

(Refer Slide Time: 10:31)



```
In [27]: model = sm.OLS(y, x)
results = model.fit()
print(results.summary())
```

OLS Regression Results

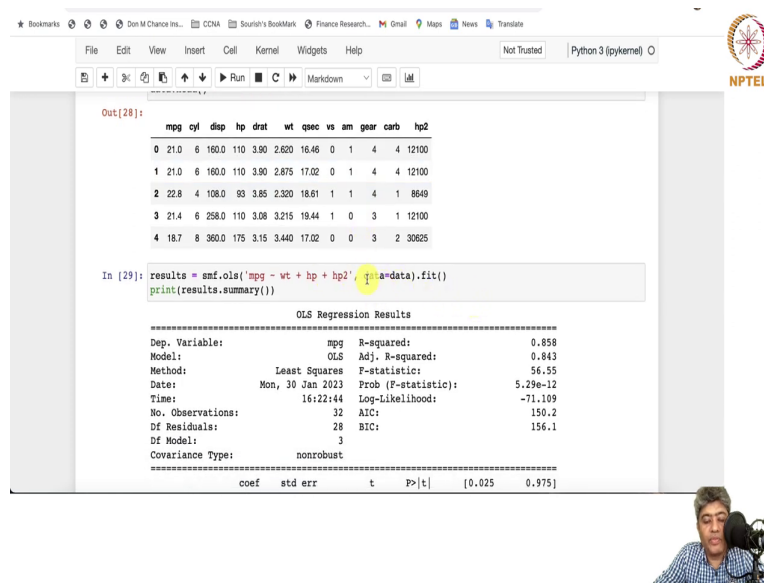
Dep. Variable:	mpg	R-squared (uncentered):	0.838
Model:	OLS	Adj. R-squared (uncentered):	0.821
Method:	Least Squares	F-statistic:	49.96
Date:	Mon, 30 Jan 2023	Prob (F-statistic):	1.41e-11
Time:	16:21:59	Log-Likelihood:	-113.1
No. Observations:	32	AIC:	233.2
Df Residuals:	29	BIC:	237.7
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
wt	-2.1246	2.496	-0.851	0.402	-7.229	2.980
hp	0.3524	0.092	3.837	0.001	0.165	0.540
hp2	-0.0010	0.000	-4.466	0.000	-0.001	-0.001

Omnibus: 4.987 Durbin-Watson: 1.095
Prob(Omnibus): 0.083 Jarque-Bera (JB): 4.591
Skew: 0.913 Prob(JB): 0.101
Kurtosis: 2.671 Cond. No. 5.64e+04

Now, and we fit the OLS way r square slightly different 83 percent we have just did r square is 82 percent. So, this is the sklearn, remember that sklearn's calculation is 85.8 percent; whereas, stat models is giving us a slightly different 83 percent ok.

(Refer Slide Time: 11:09)



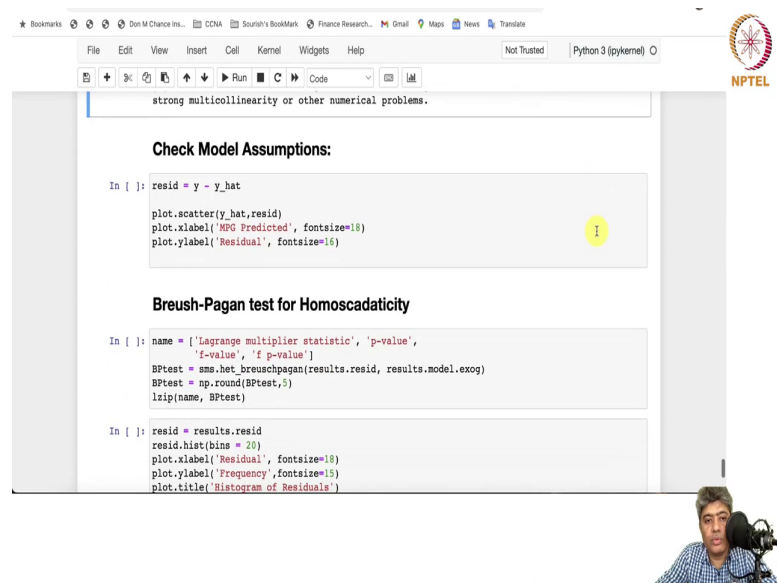
The screenshot shows a Jupyter Notebook interface with a Python kernel. The notebook contains a data table and a code cell that has been executed. The data table has 5 rows and 11 columns. The code cell contains a single line of Python code: `results = smf.ols('mpg ~ wt + hp + hp2', data=data).fit()` followed by `print(results.summary())`. The output of the code is a detailed OLS regression summary.

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	hp2	
0	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	12100
1	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	12100
2	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	8649
3	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	12100
4	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	30625

```
OLS Regression Results
=====
Dep. Variable:      mpg      R-squared:      0.858
Model:              OLS      Adj. R-squared: 0.843
Method:             Least Squares      F-statistic:    56.55
Date:               Mon, 30 Jan 2023      Prob (F-statistic): 5.29e-12
Time:               16:22:44      Log-Likelihood: -71.109
No. Observations:  32      AIC:            150.2
Df Residuals:      28      BIC:            156.1
Df Model:          3
Covariance Type:   nonrobust
=====
coef    std err      t      P>|t|      [0.025      0.975]
```

And this is our data set because you know dataset and then when we fit the OLS in this way interesting. When we fit the OLS in this way what we have seen that r square is not 85.5 percent. So, maybe this is the right way of fitting the model ok; so, previous paper is fitting you know the old way ok.

(Refer Slide Time: 11:38)



```
strong multicollinearity or other numerical problems.
```

Check Model Assumptions:

```
In [ ]: resid = y - y_hat

plot.scatter(y_hat, resid)
plot.xlabel('MPG Predicted', fontsize=18)
plot.ylabel('Residual', fontsize=16)
```

Breusch-Pagan test for Homoscedasticity

```
In [ ]: name = ['Lagrange multiplier statistic', 'p-value',
               'f-value', 'f p-value']
Bptest = sm.stats.breuschpagan(results.resid, results.model.exog)
Bptest = np.round(Bptest, 2)
lzip(name, Bptest)
```

```
In [ ]: resid = results.resid
resid.hist(bins = 20)
plot.xlabel('Residual', fontsize=18)
plot.ylabel('Frequency', fontsize=15)
plot.title('Histogram of Residuals')
```

And now, we have to do some we have to do some model as checking the model assumptions. If you want to check the model assumptions what you do? First you define the residuals and plot the residuals against the predicted value.

(Refer Slide Time: 11:55)

The screenshot shows a Jupyter Notebook interface with a Python kernel. The code cell contains the following:

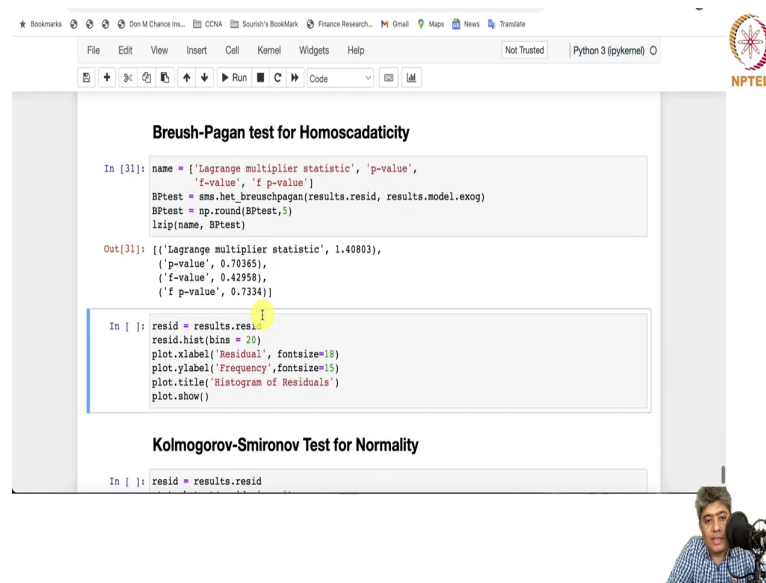
```
plot.xlabel('MPG Predicted', fontsize=18)
plot.ylabel('Residual', fontsize=16)
```

The output cell shows a scatter plot with 'MPG Predicted' on the x-axis (ranging from 10 to 30) and 'Residual' on the y-axis (ranging from -3 to 5). A single data point at approximately (15, 1) is highlighted with a yellow circle. Below the plot, the text 'Breusch-Pagan test for Homoscedasticity' is displayed. The input cell below contains the following code:

```
In [ ]: name = ['Lagrange multiplier statistic', 'p-value',
               'f-value', 'f p-value']
```

The NPTEL logo is visible in the top right corner of the notebook interface. A small video feed of a person is visible in the bottom right corner of the slide.

(Refer Slide Time: 12:05)



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

Breusch-Pagan test for Homoscedasticity

```
In [31]: name = ['Lagrange multiplier statistic', 'p-value',  
              'f-value', 'f p-value']  
BPtest = sms.het_breuschpagan(results.resid, results.model.exog)  
BPtest = np.round(BPtest,5)  
zip(name, BPtest)  
  
Out[31]: [('Lagrange multiplier statistic', 1.40803),  
          ('p-value', 0.70365),  
          ('f-value', 0.42958),  
          ('f p-value', 0.7334)]
```

```
In [ ]: resid = results.resid  
resid.hist(bins = 20)  
plot.xlabel('Residual', fontsize=18)  
plot.ylabel('Frequency', fontsize=18)  
plot.title('Histogram of Residuals')  
plot.show()
```

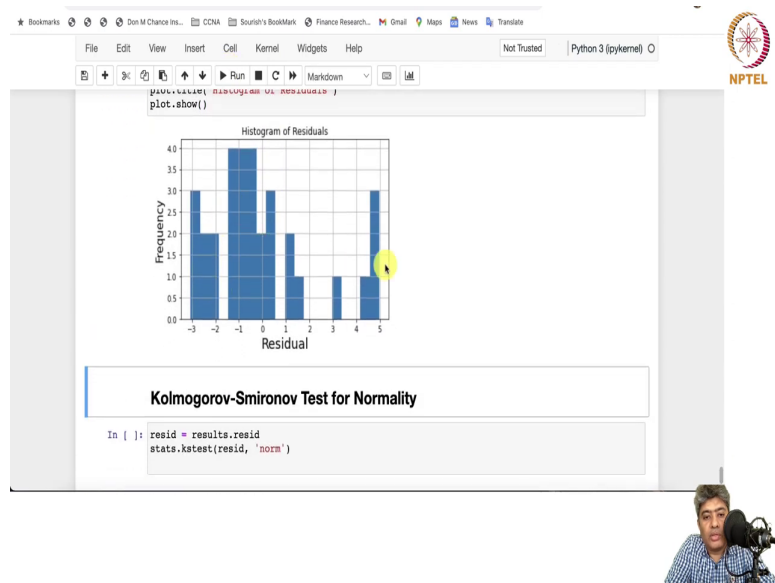
Kolmogorov-Smirnov Test for Normality

```
In [ ]: resid = results.resid
```

The NPTEL logo is visible in the top right corner of the notebook interface.

So, we see that there is not much going on; so, we can do a Breush Pagan test for a homogeneity and the p value is quite high. So, from the Breush Pagan test we cannot reject homoscedasticity, we can say safely that it is homoscedasticity behaviour, it has a homoscedasticity behaviour.

(Refer Slide Time: 12:23)



The screenshot displays a Jupyter Notebook interface. At the top, there is a browser address bar and a menu bar with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The notebook's code cell contains the following Python code:

```
plt.figure(figsize=(10,6))  
plt.hist(results.resid, bins=10, density=False, color='blue', edgecolor='black')  
plt.xlabel('Residual')  
plt.ylabel('Frequency')  
plt.show()
```

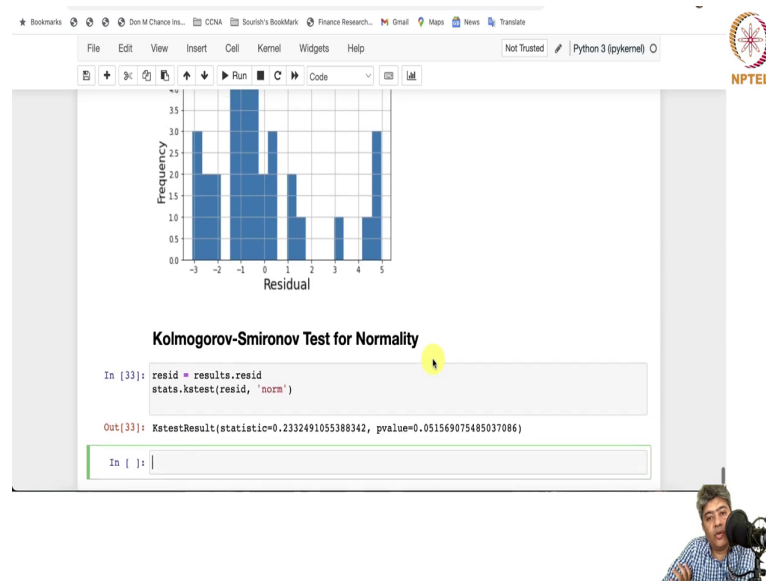
Below the code, a histogram titled "Histogram of Residuals" is shown. The x-axis is labeled "Residual" and ranges from -3 to 5. The y-axis is labeled "Frequency" and ranges from 0.0 to 4.0. The histogram consists of 10 blue bars with black outlines. A yellow mouse cursor is pointing at the bar for the residual value 5.

Below the histogram, a section titled "Kolmogorov-Smirnov Test for Normality" is displayed. The code cell below it contains the following Python code:

```
In [ ]: resid = results.resid  
stats.kstest(resid, 'norm')
```

In the bottom right corner of the slide, there is a small video thumbnail of a man speaking into a microphone.

(Refer Slide Time: 12:29)



The histogram is sparse; we can run a Kolmogorov Smironov test the p value is marginally small. So, in the previous also we saw that Kolmogorov Smironov test was marginally smaller giving us a indication that there could be questionably the normality could be questionable though other test could be fine; so, with this we will I will stop.