

**Python for Data Science**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 44**  
**Case study on regression Part II**

(Refer Slide Time: 00:15)

The screenshot shows the Spyder Python IDE with a script named 'Regression-Case Study.py'. The code performs several data cleaning steps on the 'cars' dataset:

- Line 76: `sum(cars['yearOfRegistration'] > 2018)`
- Line 77: `sum(cars['yearOfRegistration'] < 1950)`
- Line 78: `sns.regplot(x='yearOfRegistration', y='price', scatter=True, fit_reg=False, data=cars)`
- Line 80: `# Working range- 1950 and 2018`
- Line 82: `# Variable price`
- Line 83: `price_count=cars['price'].value_counts().sort_index()`
- Line 84: `sns.distplot(cars['price'])`
- Line 85: `cars['price'].describe()`
- Line 86: `sns.boxplot(y=cars['price'])`
- Line 87: `sum(cars['price'] > 150000)`
- Line 88: `sum(cars['price'] < 100)`
- Line 89: `# Working range- 100 and 150000`
- Line 92: `# Variable powerPS`
- Line 93: `power_count=cars['powerPS'].value_counts().sort_index()`
- Line 94: `sns.distplot(cars['powerPS'])`
- Line 95: `cars['powerPS'].describe()`
- Line 96: `sns.boxplot(y=cars['powerPS'])`
- Line 97: `sns.regplot(x='powerPS', y='price', scatter=True, fit_reg=False, data=cars)`
- Line 98: `sum(cars['powerPS'] > 500)`
- Line 99: `sum(cars['powerPS'] < 10)`
- Line 100: `# Working range- 10 and 500`
- Line 104: `# =====`
- Line 105: `# Working range of data`
- Line 106: `# =====`

The Variable explorer on the right shows the following variables:

Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, price, atest, vehicleType, yearOfReg ...
cars_data	DataFrame	(50001, 19)	Column names: dateCreated, name, seller, offerType, price, atest, veh ...
col	list	5	['name', 'dateCreated', 'dateCreated', 'postalCode', 'lastSeen']
power_count	Series	(480,)	Series object of pandas.core.series module
price_count	Series	(2393,)	Series object of pandas.core.series module
yearwise_count	Series	(97,)	Series object of pandas.core.series module

So, now let us incorporate these cleaning ranges.

(Refer Slide Time: 00:18)

The screenshot shows the Spyder Python IDE with the same script. The code now applies the cleaning ranges to the 'cars' dataset:

- Line 89: `# Working range- 100 and 150000`
- Line 90: `# =====`
- Line 92: `# Variable powerPS`
- Line 93: `power_count=cars['powerPS'].value_counts().sort_index()`
- Line 94: `sns.distplot(cars['powerPS'])`
- Line 95: `cars['powerPS'].describe()`
- Line 96: `sns.boxplot(y=cars['powerPS'])`
- Line 97: `sns.regplot(x='powerPS', y='price', scatter=True, fit_reg=False, data=cars)`
- Line 98: `sum(cars['powerPS'] > 500)`
- Line 99: `sum(cars['powerPS'] < 10)`
- Line 100: `# Working range- 10 and 500`
- Line 101: `# =====`
- Line 104: `# =====`
- Line 105: `# Working range of data`
- Line 106: `# =====`
- Line 107: `# =====`
- Line 108: `# Working range of data`
- Line 109: `# =====`
- Line 110: `cars = cars[`
- Line 111:  `(cars.yearOfRegistration <= 2018)`
- Line 112:  `& (cars.yearOfRegistration >= 1950)`
- Line 113:  `& (cars.price >= 100)`
- Line 114:  `& (cars.price <= 150000)`
- Line 115:  `& (cars.powerPS >= 10)`
- Line 116:  `& (cars.powerPS <= 500)]`
- Line 117: `# ~6700 records are dropped`
- Line 118: `# =====`

The Variable explorer on the right shows the updated variables:

Name	Type	Size	Value
cars	DataFrame	(41772, 14)	Column names: seller, offerType, price, atest, vehicleType, yearOfReg ...
cars_data	DataFrame	(50001, 19)	Column names: dateCreated, name, seller, offerType, price, atest, veh ...
col	list	5	['name', 'dateCreated', 'dateCreated', 'postalCode', 'lastSeen']
power_count	Series	(480,)	Series object of pandas.core.series module
price_count	Series	(2393,)	Series object of pandas.core.series module
yearwise_count	Series	(97,)	Series object of pandas.core.series module

So, what I am trying to do here is from the cars data I am accessing yearOfRegistration and putting a bound on it. The upper bound for yearOfRegistration is 2018 and the lower

bound is 1950, both ranges inclusive. So, from cars I have accessed yearOfRegistration. Now, earlier I have used square braces this is the other way of accessing it this is also fine or if you are comfortable with the other way of accessing then that is also fine.

So, next is price, the lower bound for price is 100 and the upper bound for price is 1,50,000 and similarly for powerPS the lower bound is 10 and the upper bound is 500. So, if you see here, we started with around 49531 after removing duplicates; and now let us see how many records are we losing after this. So, we are roughly losing about 6700 records here. So, all these changes have been affected into cars. Also with yearOfRegistration it is hard to find out what is the age of the car.

(Refer Slide Time: 01:20)

```

89 # Working range- 100 and 150000
90
91
92 # Variable powerPS
93 power_count=cars['powerPS'].value_counts()
94 sns.distplot(cars['powerPS'])
95 cars['powerPS'].describe()
96 sns.boxplot(y=cars['powerPS'])
97 sns.regplot(x='powerPS', y='price',
98             fit_reg=False, data=cars)
99 sum(cars['powerPS'] > 500)
100 sum(cars['powerPS'] < 10)
101 # Working range- 10 and 500
102
103
104 # Working range of data
105 # =====
106 # Working range of data
107 # =====
108 # Working range of data
109
110 cars = cars[
111     (cars.yearOfRegistration <= 2018)
112     & (cars.yearOfRegistration >= 1950)
113     & (cars.price >= 100)
114     & (cars.price <= 150000)
115     & (cars.powerPS >= 10)
116     & (cars.powerPS <= 500)]
117 # ~6700 records are dropped
118

```

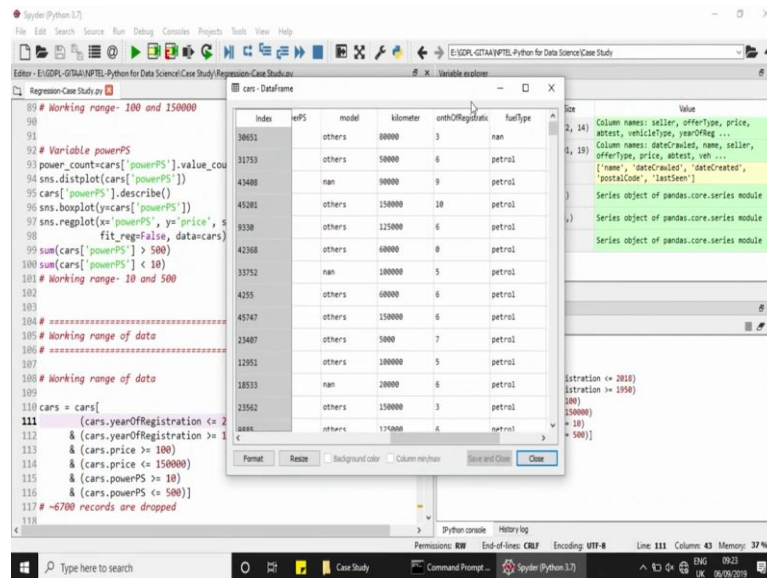
Index	vehicleType	yearOfRegistration	gearbox	powerPS
30651	limousine	1951	nan	39
31753	limousine	1951	manual	58
43488	cabrio	1951	manual	54
45201	suv	1953	manual	72
9338	limousine	1954	nan	158
42388	limousine	1955	manual	38
33752	limousine	1955	automatic	228
4255	limousine	1955	manual	56
45747	station wagon	1955	manual	75
23487	cabrio	1955	manual	213
12951	limousine	1956	manual	45
18533	suv	1956	manual	75
23562	suv	1957	automatic	288
1881	limousine	1957	nan	181

Column names: seller, offerType, price, attest, vehicleType, yearOfReg ...  
Column names: dateCrawled, name, seller, offerType, price, attest, veh ...  
['name', 'dateCrawled', 'dateCreated', 'postalCode', 'lastSeen']  
Series object of pandas.core.series module  
Series object of pandas.core.series module  
Series object of pandas.core.series module

117 # ~6700 records are dropped

So, what we are going to do is we are going to subtract the yearOfRegistration from 2018 and add the monthOfRegistration.

(Refer Slide Time: 01:34)



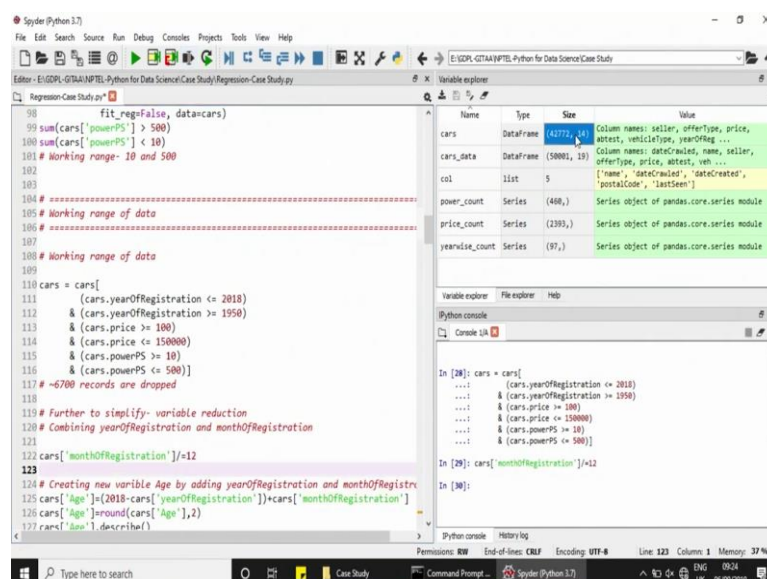
The screenshot shows the Spyder Python IDE interface. The main window displays a Jupyter Notebook with the following code:

```
89 # Working range- 100 and 150000
90
91 # Variable powerPS
92 power_count=cars['powerPS'].value_count()
93 sns.distplot(cars['powerPS'])
94 cars['powerPS'].describe()
95 sns.boxplot(y=cars['powerPS'])
96 sns.regplot(x='powerPS', y='price',
97             fit_reg=False, data=cars)
98
99 sum(cars['powerPS'] > 500)
100 sum(cars['powerPS'] < 10)
101 # Working range- 10 and 500
102
103
104 # Working range of data
105 # Working range of data
106 # Working range of data
107
108 # Working range of data
109
110 cars = cars[
111     (cars.yearOfRegistration <= 2018)
112     & (cars.yearOfRegistration >= 1950)
113     & (cars.price >= 100)
114     & (cars.price <= 150000)
115     & (cars.powerPS >= 10)
116     & (cars.powerPS <= 500)]
117 # ~6700 records are dropped
118
```

The Variable Explorer window is open, showing the 'cars' DataFrame with columns: index, yearPS, model, kilometer, monthOfRegistration, fuelType. The 'cars' DataFrame has 5000 rows and 6 columns. The 'monthOfRegistration' column is highlighted.

So, the monthOfRegistration we are going to divide it by 12 which gives you months in decimals, and I am going to create another variable called age that will contain the difference between the year 2018 and the yearOfRegistration. Now, that is the first step. So, I am going to subtract the yearOfRegistration from the year 2018 and then I am going to add the monthOfRegistration which is monthOfRegistration by 12. So, let us start by doing this.

(Refer Slide Time: 02:09)



The screenshot shows the Spyder Python IDE interface. The main window displays a Jupyter Notebook with the following code:

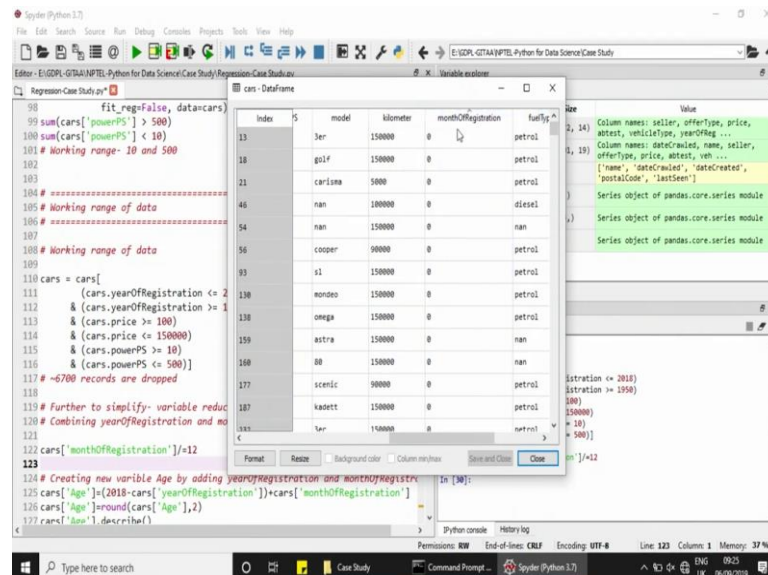
```
98 fit_reg=False, data=cars)
99 sum(cars['powerPS'] > 500)
100 sum(cars['powerPS'] < 10)
101 # Working range- 10 and 500
102
103
104 # Working range of data
105 # Working range of data
106 # Working range of data
107
108 # Working range of data
109
110 cars = cars[
111     (cars.yearOfRegistration <= 2018)
112     & (cars.yearOfRegistration >= 1950)
113     & (cars.price >= 100)
114     & (cars.price <= 150000)
115     & (cars.powerPS >= 10)
116     & (cars.powerPS <= 500)]
117 # ~6700 records are dropped
118
119 # Further to simplify- variable reduction
120 # Combining yearOfRegistration and monthOfRegistration
121
122 cars['monthOfRegistration']/=12
123
124 # Creating new variable Age by adding yearOfRegistration and monthOfRegistration
125 cars['Age']=(2018-cars['yearOfRegistration']+cars['monthOfRegistration'])
126 cars['Age']=round(cars['Age'],2)
127 cars['Age'].describe()

```

The Variable Explorer window is open, showing the 'cars' DataFrame with columns: Name, Type, Size, Value. The 'cars' DataFrame has 5000 rows and 6 columns. The 'cars' DataFrame is highlighted.

So, first we are going to divide the monthOfRegistration by 12. So, let us just do that and see if the changes are reflected.

(Refer Slide Time: 02:17)



The screenshot shows the Spyder Python IDE interface. The main editor displays a Jupyter notebook with the following code:

```
98 fit_reg=False, datacars)
99 sum(cars['powerPS'] > 500)
100 sum(cars['powerPS'] < 10)
101 # Working range- 10 and 500
102
103 # Working range of data
104 # Working range of data
105 # Working range of data
106 # Working range of data
107
108 cars = cars[
109     (cars.yearOfRegistration <= 2018)
110     & (cars.yearOfRegistration >= 1990)
111     & (cars.price >= 10000)
112     & (cars.price <= 150000)
113     & (cars.powerPS >= 10)
114     & (cars.powerPS <= 500)]
115 # ~6700 records are dropped
116
117 # Further to simplify- variable reduction
118 # Combining yearOfRegistration and monthOfRegistration
119 cars['Age'] = (2018 - cars['yearOfRegistration']) + cars['monthOfRegistration']
120 cars['Age'] = round(cars['Age'], 2)
121 cars['Age'].describe()
```

The Dataframe viewer on the right shows a table with columns: Index, S, model, kilometer, monthOfRegistration, fuelType. The first few rows are:

Index	S	model	kilometer	monthOfRegistration	fuelType
13	3er	150000	0	petrol	
18	golf	150000	0	petrol	
21	carina	5000	0	petrol	
46	nan	100000	0	diesel	
54	nan	150000	0	nan	
56	cooper	90000	0	petrol	
93	s1	150000	0	petrol	
130	mondeo	150000	0	petrol	
138	omega	150000	0	petrol	
159	astra	150000	0	nan	
160	00	150000	0	nan	
167	scenic	90000	0	petrol	
187	kadett	150000	0	petrol	
188	tao	100000	0	petrol	

So, whichever entries were 0 earlier in terms of integer, they become 0 with decimal. So, again when you add your yearOfRegistration the difference between 2018 and the yearOfRegistration, when you add it with the monthOfRegistration these entries will again be reflected as 0 under age. So, indirectly we are actually reducing the effect of 0 because if you try and use the monthOfRegistration as integer itself. Because if you try and use the monthOfRegistration as integer itself it is going to create some problems because 0 is not a valid integer to represent any of the month and the months are being represented by numbers 1 to 12, right.

So, because we wanted to get rid of 0s, we are trying to divide each of the entries by 12 which means any entry which has a monthOfRegistration of 0 will automatically remain a 0 and that when you add it with the age which is a difference between the year 2018 and the yearOfRegistration and that will eventually go off, right.

So, we are not letting go of observations which are marked as 0 under monthOfRegistration, at the same time you are also making sure that it does not affect our analysis. So, we have converted the monthOfRegistration as decimals. So, I have just divided by 12. Now, I am going to create a new variable called age in line 125 and it is

the difference between the year 2018 minus the yearOfRegistration for that particular record and I am adding the monthOfRegistration here.

(Refer Slide Time: 04:17)

The screenshot shows the Spyder Python IDE with a Jupyter notebook open. The notebook contains the following code:

```

101 # Working range- 10 and 500
102
103
104 # Working range of data
105
106 # Working range of data
107
108 # Working range of data
109
110 cars = cars[
111     (cars.yearOfRegistration <= 2
112     & (cars.yearOfRegistration >= 1
113     & (cars.price >= 100)
114     & (cars.price <= 150000)
115     & (cars.powerPS >= 10)
116     & (cars.powerPS <= 500))]
117 # ~6700 records are dropped
118
119 # Further to simplify- variable reduc
120 # Combining yearOfRegistration and mo
121
122 cars['monthOfRegistration']//12
123
124 # Creating new variable Age by adding
125 cars['Age']=(2018-cars['yearOfRegistr
126 cars['Age']=round(cars['Age'],2)
127 cars['Age'].describe()
128
129
130 # Dropping yearOfRegistration and monthOfRegistration
131 cars=cars.drop(columns=['yearOfRegistration', 'monthOfRegistration'], axis=1)

```

The 'cars - DataFrame' window shows the following data:

Index	Registration	FuelType	Brand	OfferedPrice	Age
0	diesel	bmw	nan	15.25	
1	diesel	volvo	no	13.5	
2	67	diesel	volkswagen	nan	15.9167
3	petrol	seat	no	13	
4	67	diesel	volvo	no	10.9167
5	67	petrol	volkswagen	no	23.1667
6	67	petrol	mercedes_benz	no	21.9167
7	petrol	opel	no	14	
8	33	diesel	skoda	no	13.3333
9	333	petrol	mercedes_benz	no	17.0833
10	33	petrol	opel	no	17.3333
11	petrol	toyota	no	17.75	
12	petrol	bmw	yes	2	
13	67	petrol	bmw	no	15.4167

You will see that an extra column gets added to cars and here you will see the different values of age, because we have added the months here you will see values which are in decimals. So, we are going to round off the decimal places to 2, using the round function. So, the input to the round function is the column that you want to round off and to how many places do you want to round it off to.

(Refer Slide Time: 04:40)

The screenshot shows the Spyder Python IDE with the Jupyter notebook updated with the following code:

```

122 cars['monthOfRegistration']//12
123
124 # Creating new variable Age by adding
125 cars['Age']=(2018-cars['yearOfRegistr
126 cars['Age']=round(cars['Age'],2)
127 cars['Age'].describe()
128
129
130 # Dropping yearOfRegistration and monthOfRegistration
131 cars=cars.drop(columns=['yearOfRegistration', 'monthOfRegistration'], axis=1)

```

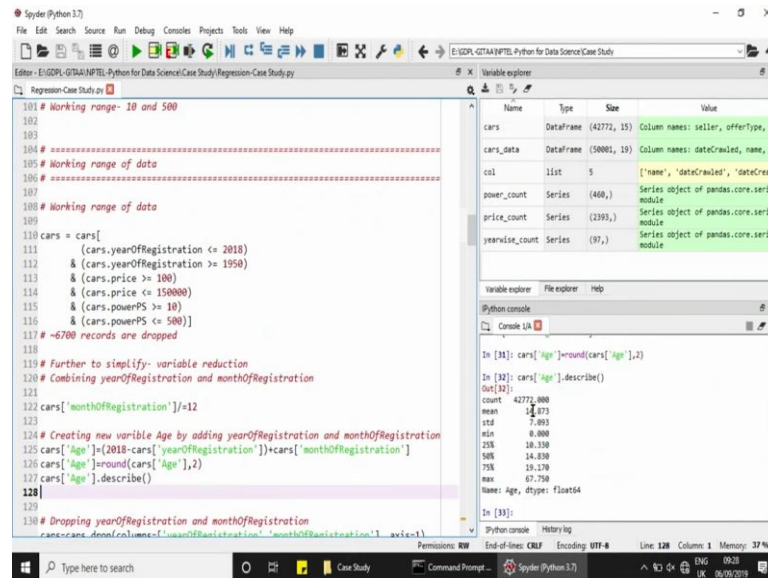
The 'cars - DataFrame' window shows the following data:

Index	Registration	FuelType	Brand	OfferedPrice	Age
0	diesel	bmw	nan	15.25	
1	diesel	volvo	no	13.5	
2	67	diesel	volkswagen	nan	15.92
3	petrol	seat	no	13	
4	67	diesel	volvo	no	10.92
5	67	petrol	volkswagen	no	23.17
6	67	petrol	mercedes_benz	no	21.92
7	petrol	opel	no	14	
8	33	diesel	skoda	no	13.33
9	333	petrol	mercedes_benz	no	17.08
10	33	petrol	opel	no	17.33
11	petrol	toyota	no	17.75	
12	petrol	bmw	yes	2	
13	67	petrol	bmw	no	15.42



I am effecting these changes in the same column itself. So, again let us just open and see, yes. So, all these values have been rounded off to 2 decimal places. So, let us just quickly do a.describe.

(Refer Slide Time: 04:57)



The screenshot shows the Spyder Python IDE interface. The editor window displays a Python script for data cleaning and feature engineering. The code includes comments and steps for filtering data, creating a new 'Age' variable, and dropping unnecessary columns. The variable explorer on the right shows the state of the data after the first set of operations. The console window at the bottom shows the output of the describe() method for the 'Age' variable.

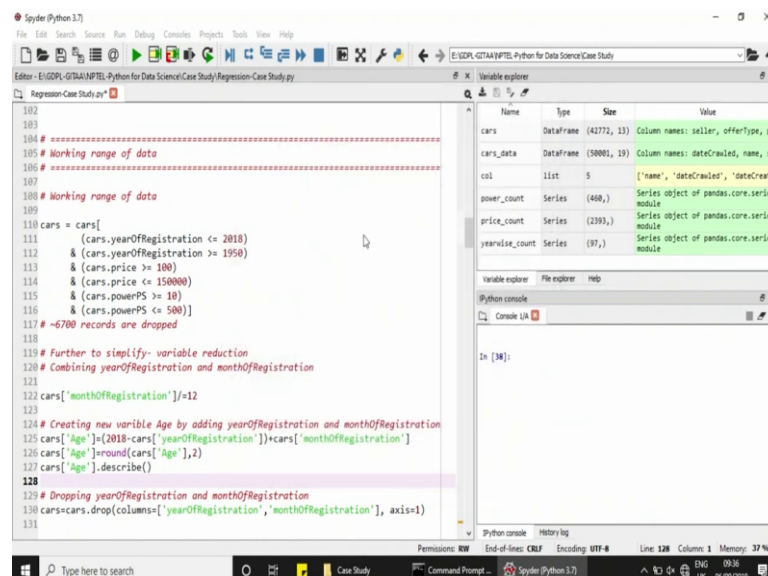
```
101 # Working range- 10 and 500
102
103
104 # Working range of data
105 # Working range of data
106 # Working range of data
107
108 # Working range of data
109
110 cars = cars[
111     (cars.yearOfRegistration <= 2018)
112     & (cars.yearOfRegistration >= 1950)
113     & (cars.price <= 100)
114     & (cars.price <= 150000)
115     & (cars.powerPS >= 10)
116     & (cars.powerPS <= 500)]
117 # ~6700 records are dropped
118
119 # Further to simplify- variable reduction
120 # Combining yearOfRegistration and monthOfRegistration
121 cars['monthOfRegistration']//12
122
123
124 # Creating new variable Age by adding yearOfRegistration and monthOfRegistration
125 cars['Age']=(2018-cars['yearOfRegistration'])*cars['monthOfRegistration']
126 cars['Age']=round(cars['Age'],2)
127 cars['Age'].describe()
128
129 # Dropping yearOfRegistration and monthOfRegistration
130 cars=cars.drop(columns=['yearOfRegistration', 'monthOfRegistration'], axis=1)
```

Name	Type	Size	Value
cars	Dataframe	(42772, 15)	Column names: seller, offerType, p...
cars_data	Dataframe	(50001, 19)	Column names: dateCrawled, name, s...
col	list	5	['name', 'dateCrawled', 'dateCreat...
power_count	Series	(460,)	Series object of pandas.core.series module
price_count	Series	(2393,)	Series object of pandas.core.series module
yearwise_count	Series	(97,)	Series object of pandas.core.series module

```
In [31]: cars['Age']=round(cars['Age'],2)
In [32]: cars['Age'].describe()
Out[32]:
count    42772.000
mean      14.873
std        7.093
min         0.000
25%      10.330
50%      14.830
75%      19.170
max       67.750
Name: Age, dtype: float64
```

So, you will see that the count is again 42772 and the mean and the mean age or the average of age is 14.87 with a standard deviation of 7 and minimum is 0, maximum is 67, and the median lies at 14.83. So, the mean and the median are not very far off, the mean is at 14.87 and the median is at 14.83. So, we know the data is not very skewed.

(Refer Slide Time: 05:27)



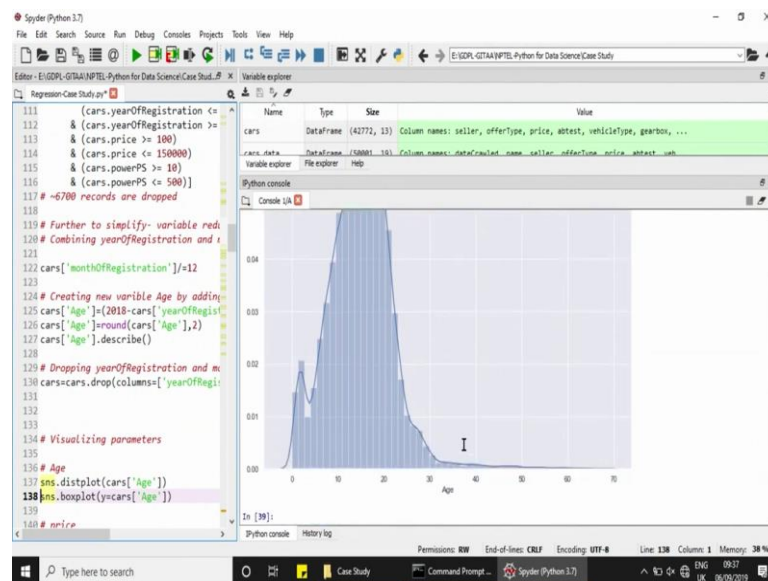
This screenshot shows the Spyder Python IDE after the second set of operations. The code in the editor continues from the previous state, dropping the 'yearOfRegistration' and 'monthOfRegistration' columns. The variable explorer on the right shows the updated state of the 'cars' dataframe. The console window is empty, indicating that the describe() method has not been executed again in this state.

```
102
103
104 # Working range of data
105 # Working range of data
106 # Working range of data
107
108 # Working range of data
109
110 cars = cars[
111     (cars.yearOfRegistration <= 2018)
112     & (cars.yearOfRegistration >= 1950)
113     & (cars.price >= 100)
114     & (cars.price <= 150000)
115     & (cars.powerPS >= 10)
116     & (cars.powerPS <= 500)]
117 # ~6700 records are dropped
118
119 # Further to simplify- variable reduction
120 # Combining yearOfRegistration and monthOfRegistration
121 cars['monthOfRegistration']//12
122
123
124 # Creating new variable Age by adding yearOfRegistration and monthOfRegistration
125 cars['Age']=(2018-cars['yearOfRegistration'])*cars['monthOfRegistration']
126 cars['Age']=round(cars['Age'],2)
127 cars['Age'].describe()
128
129 # Dropping yearOfRegistration and monthOfRegistration
130 cars=cars.drop(columns=['yearOfRegistration', 'monthOfRegistration'], axis=1)
131
```

Name	Type	Size	Value
cars	Dataframe	(42772, 13)	Column names: seller, offerType, p...
cars_data	Dataframe	(50001, 19)	Column names: dateCrawled, name, s...
col	list	5	['name', 'dateCrawled', 'dateCreat...
power_count	Series	(460,)	Series object of pandas.core.series module
price_count	Series	(2393,)	Series object of pandas.core.series module
yearwise_count	Series	(97,)	Series object of pandas.core.series module

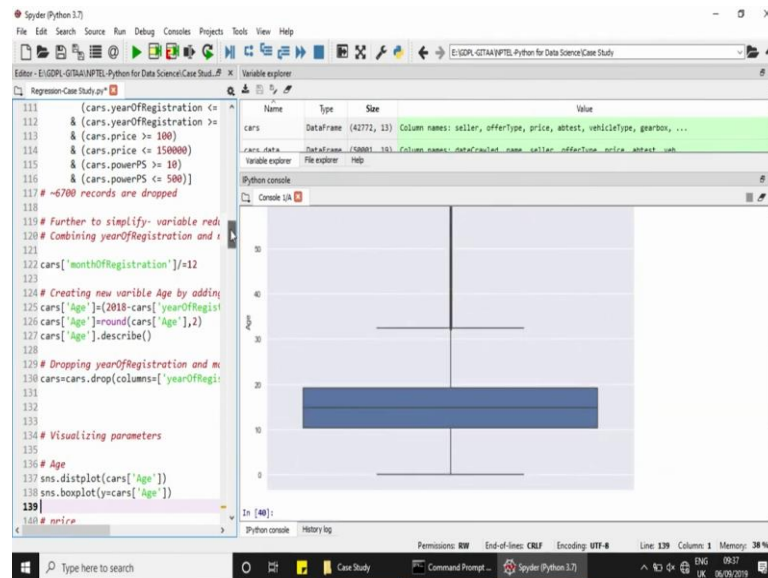
So, now that we have incorporated the information of yearOfRegistration and monthOfRegistration into 8. We can drop off these columns because adding them is only going to make it very redundant. So, I am going to use the cars.drop function to clear. So, I am going to use the cars.drop function to drop them and I am giving the column names as a list, and I am giving access is equal to 1 because these are because these are columns. Now, once you drop them you can see that the number of columns have reduced by 2 and the year and monthOfRegistration have disappeared, right.

(Refer Slide Time: 06:01)



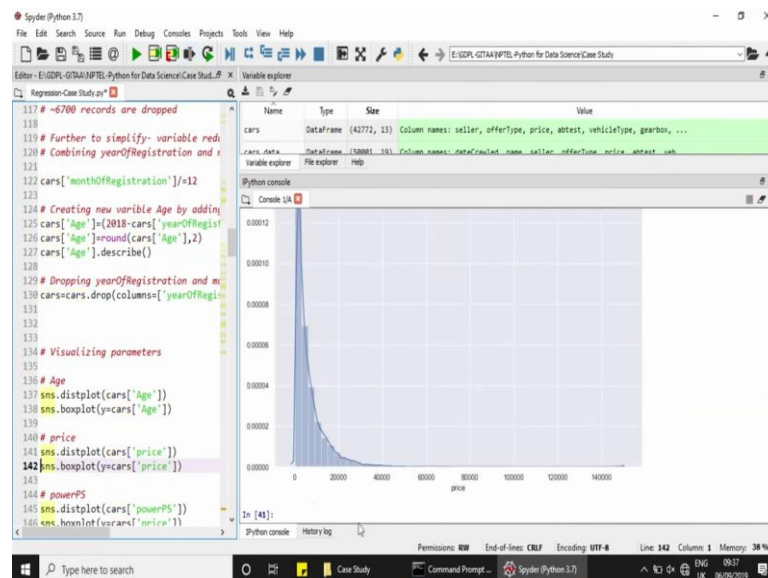
So, now let us start by visualizing parameters I am going to first start with visualizing age. So, let us see how does the histogram look for age. So, if you look at age here you can see a slightly skewed distribution, it is not extremely skewed, let me just drag let me just drag the box to show you how does the distribution look. Again you can fit a density curve over it. So, now, again let us plot a box plot here for age.

(Refer Slide Time: 06:37)



Now, we can see a clear cut box here, though above the whiskers there are a few extreme values or outliers, right. But even this was not possible in the earlier case.

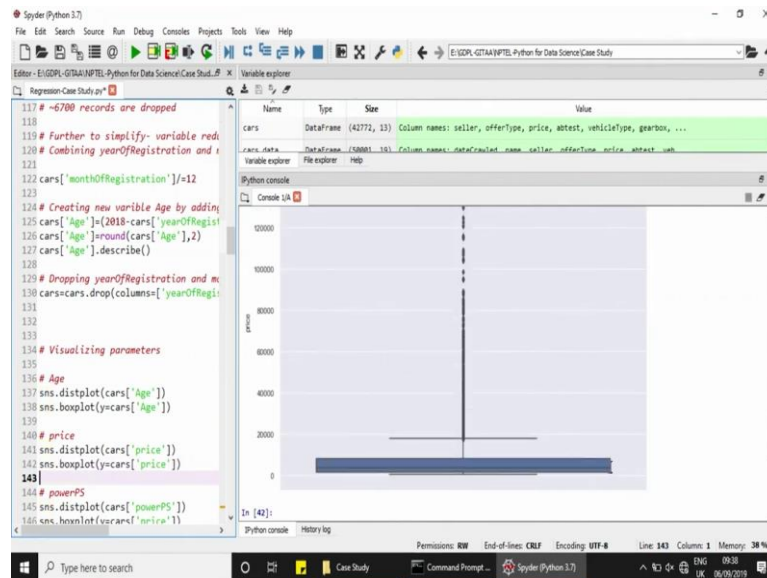
(Refer Slide Time: 06:51)



So, let us now look at the histogram for price. Now, for price here you will see histogram, even this was not seen earlier, but now we are able to see a few more ranges.



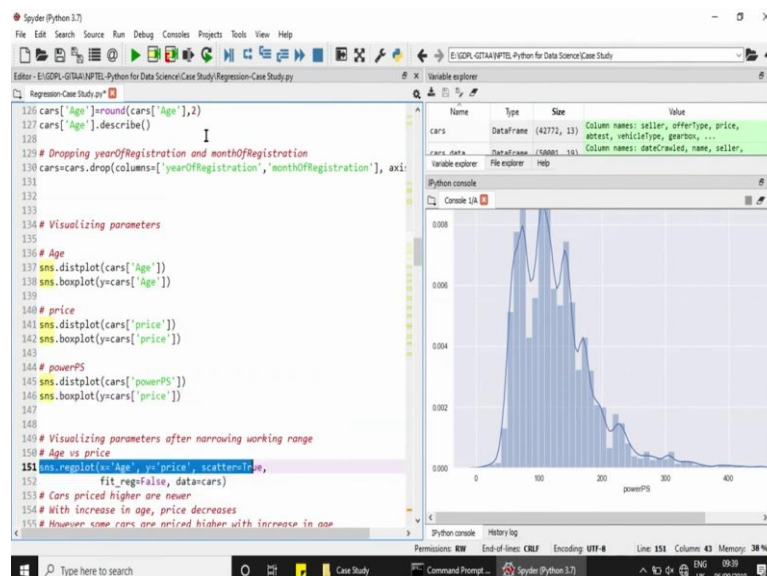
(Refer Slide Time: 07:00)



And if you plot the box plot for the variable price you will see a small box here, earlier what we could really see was just a line, right. Now, this is better, but again you still have a few extreme values outside the whiskers.

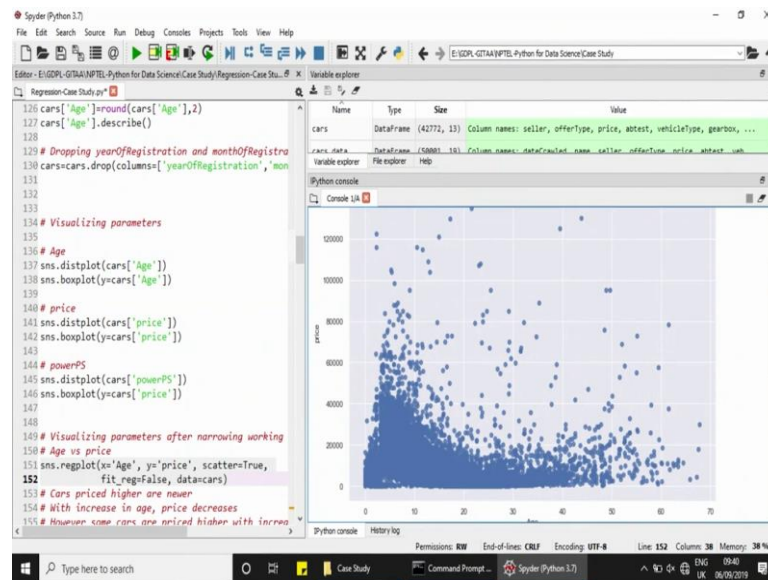
If you look at powerPS; if you look at powerPS you will see that the histogram is now better, better than earlier definitely because earlier it was all bundled up to one side and you can see that the density curve is also being fit here and you will see more ranges to powerPS than it was earlier.

(Refer Slide Time: 07:29)



So, now, that we have seen the individual distributions and we have seen the individual histograms and box plot for the 3 variables, let us see what their effect is on price. So, let us start by plotting. So, let us start by plotting age versus price here. I am using the regplot that is under c1 package again here.

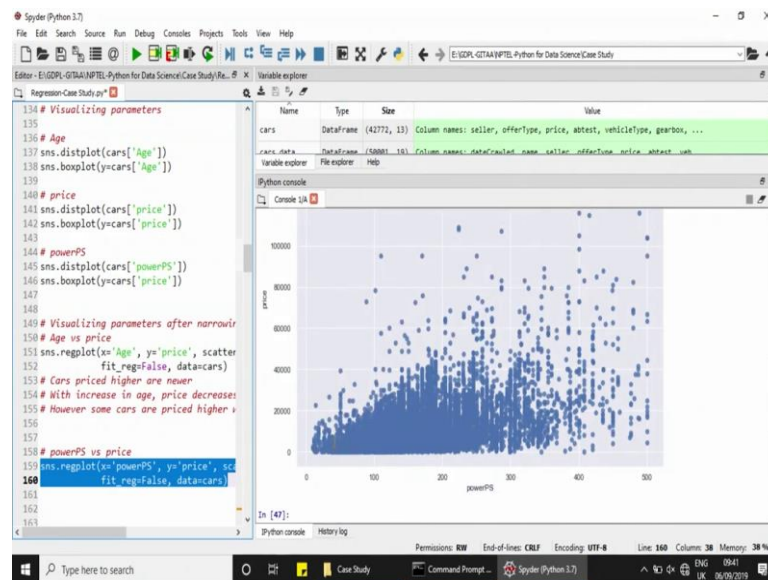
(Refer Slide Time: 08:08)



So, let me just drag this output window to the left to show you how does the plot look. So, you can see that it is a scatter plot and so you can see that it is a scatter plot; earlier when you try to plot yearOfRegistration versus price you would have seen that all these values were bunched up together in one end and we could not really see a clear cut effect of age on price, but here it is much better.

We can see that cars which are priced higher are fairly newer, so you have age on the x axis price on the y axis and cars which are priced higher are fairly new. But cars, but there are cars which are priced a little higher and are still older, so we can treat them as vintage. And on a on a very general note with increase in age the price also drops, right. So, this is the effect of age on price. Yes, it does affect a price to a great deal. But now, let us see the effect of powerPS versus price.

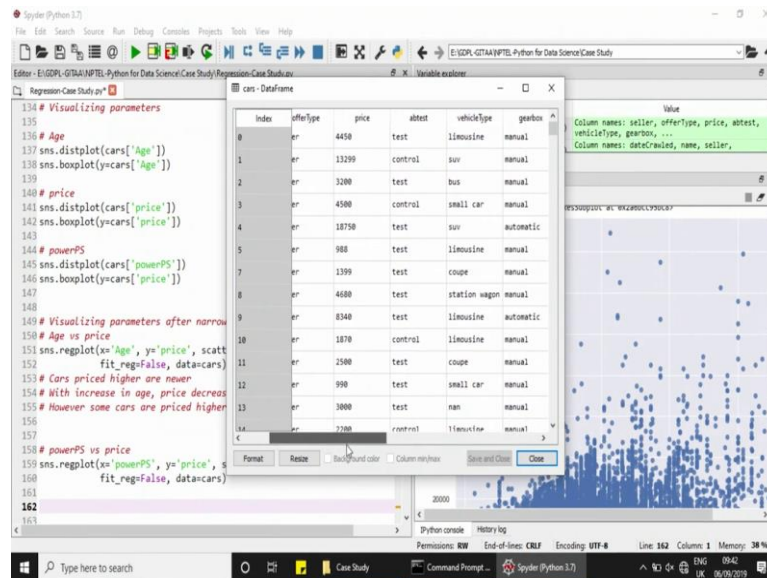
(Refer Slide Time: 09:32)



So, I am again using the same plot to plot powerPS with price. My power is on the x axis and my price is again on the y axis. So, here we can see with increase in power the price increases and that is very very clear from this plot because earlier we were not able to infer the effect of powerPS on price, we were not sure, but now that we have arrived at a very very narrow range of working values we can see an effect of powerPS on price. So, with increase in powerPS, the price eventually increases.

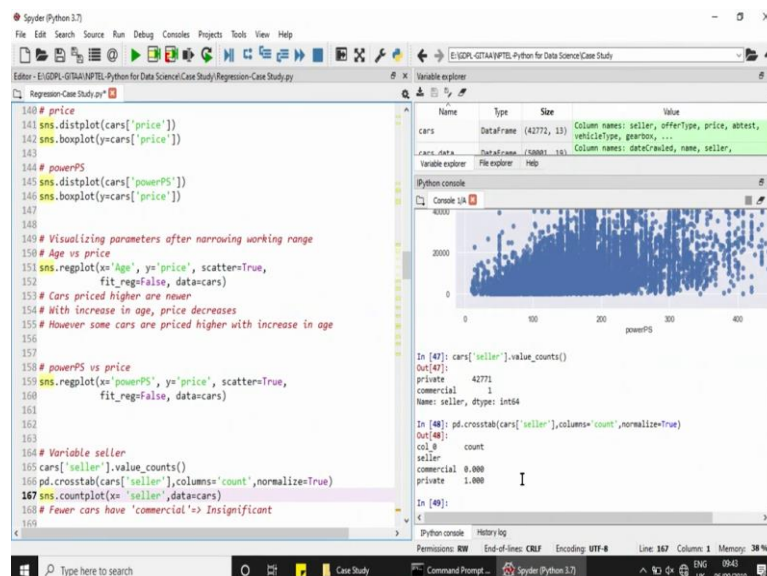
So, these were the effect of powerPS, age on price. Of course, age is a combination of yearOfRegistration and monthOfRegistration. So, two variables have been incorporated into a single in the form of a single variable. Now, having done this we are also going to check what are the effect of other variables on price, right.

(Refer Slide Time: 10:38)



But before that we have to even narrow down if these variables, if these, but mostly the categorical variables if they have enough if there are enough categories or if there is enough count under each category for us to even retain them. So, you again have seller offer type then you have abtest and vehicleType gearbox and there are many more categorical variables. So, now let us see what is the individual frequency count under each of these categories.

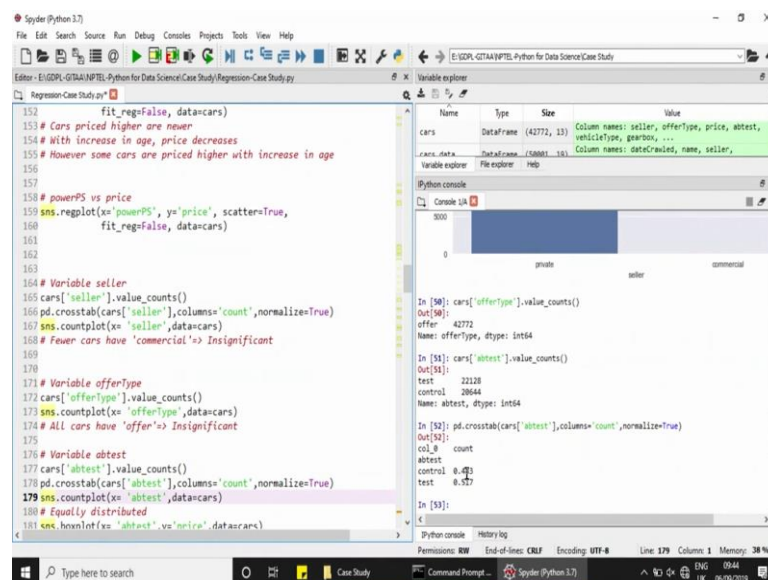
(Refer Slide Time: 11:12)



So, I first I am, so I first taken seller the variable seller and if you look at the value count you will see private is 42771 and commercial is only 1. So, the seller variable has two categories, private and commercial, but one of the category has only one observation. So, we are going to drop it and if you are interested from a proportion point of view then you can clearly see that private almost accounts for 100 percent of the proportion.

So, now that we have seen all this, we can clearly say that the variable seller is insignificant. So, we will also see how to remove it later on, but just let us identify all the variables which are insignificant and remove them together.

(Refer Slide Time: 12:00)

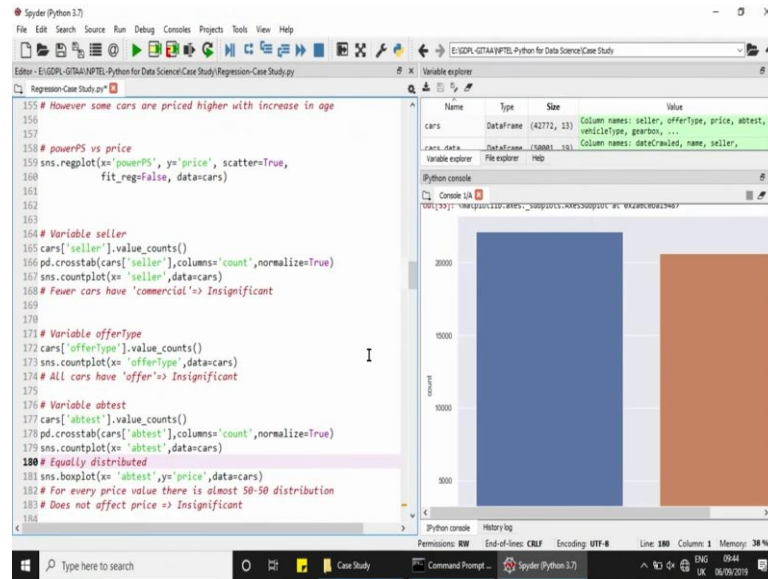


So, the next variable is offer type. So, in your earlier data you would have seen two offer type, one is offer, the other is request. And in this case all the 4 all the 42772 records are of the type offer, so we do not have any of the records under request. So, again this variable is again going to be insignificant.

Now, moving further let us look at the variable abtest. So, under abtest if you do a value count you can see there is an equal split between the categories, test and control, right. And from a proportion point of view 51 percent, roughly about 52 percent is under test and the remaining 48 percent is under control, right. So, this is more or less, it is a closed cut its more or less equal and that can also be seen from the bar graph here, right.

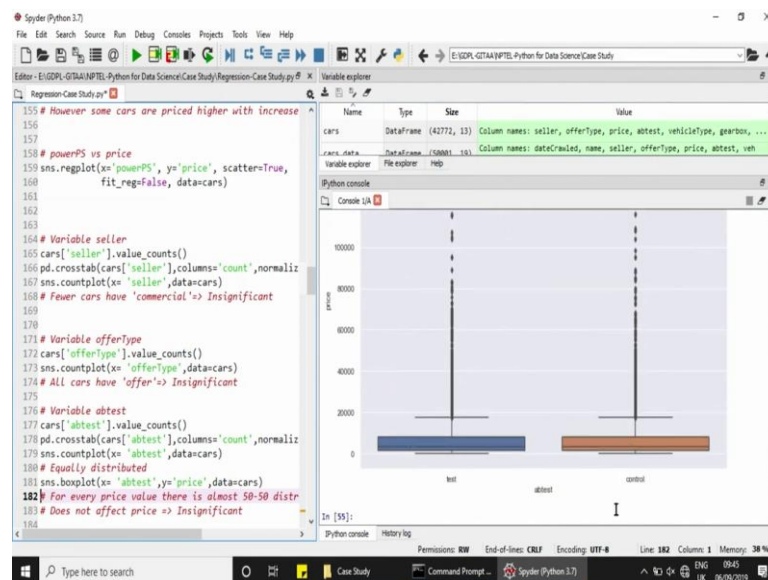


(Refer Slide Time: 12:50)



It is cutting very very closely both the categories. So, both the categories are cutting very very closely. So, they are more or less equally distributed, but let us say their effect on price

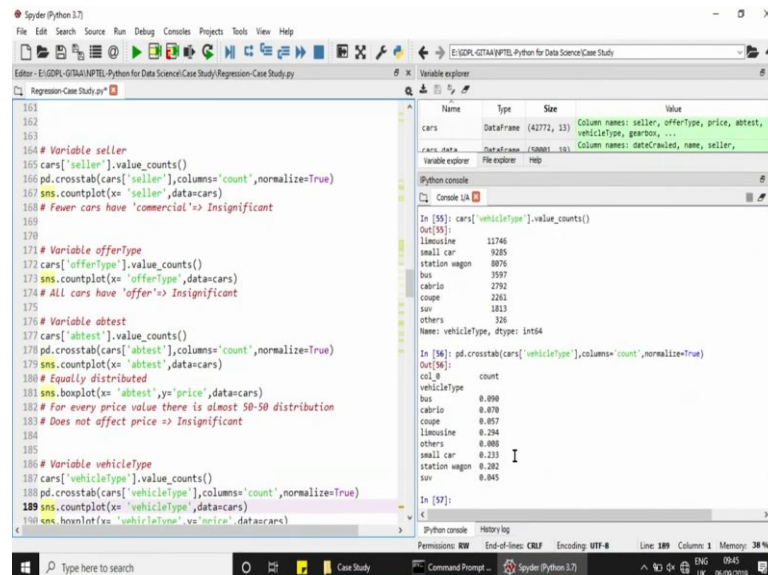
(Refer Slide Time: 13:06)



But even from the box plot it seems like the effect is more or less the same. So, there is nothing much you can tell whether between categories test and control. They seem to have the same median or, they seem to have the same median for price. So, there is nothing much that we can infer even from the box plot. So, for every price value there is

almost a 50-50 distribution. So, we cannot really conclude anything from abtest. So, we are going to treat it as insignificant.

(Refer Slide Time: 13:37)



The screenshot shows the Spyder Python IDE interface. The main editor displays a Python script for data analysis. The code includes comments and function calls for exploring the 'cars' dataset, specifically focusing on the 'seller', 'offerType', 'abtest', and 'vehicleType' variables. The variable explorer on the right shows the 'cars' variable as a DataFrame with 42772 rows and 13 columns. The Python console shows the output of the code execution, displaying the value counts for the 'vehicleType' variable.

```
161
162
163
164 # Variable seller
165 cars['seller'].value_counts()
166 pd.crosstab(cars['seller'], columns='count', normalize=True)
167 sns.countplot(x='seller', data=cars)
168 # Fewer cars have 'commercial' => Insignificant
169
170
171 # Variable offerType
172 cars['offerType'].value_counts()
173 sns.countplot(x='offerType', data=cars)
174 # All cars have 'offer' => Insignificant
175
176 # Variable abtest
177 cars['abtest'].value_counts()
178 pd.crosstab(cars['abtest'], columns='count', normalize=True)
179 sns.countplot(x='abtest', data=cars)
180 # Equally distributed
181 sns.boxplot(x='abtest', y='price', data=cars)
182 # For every price value there is almost 50-50 distribution
183 # Does not affect price => Insignificant
184
185
186 # Variable vehicleType
187 cars['vehicleType'].value_counts()
188 pd.crosstab(cars['vehicleType'], columns='count', normalize=True)
189 sns.countplot(x='vehicleType', data=cars)
190 sns.lmplot(x='vehicleType', y='price', data=cars)
```

Variable explorer:

Name	Type	Size	Value
cars	DataFrame	(42772, 13)	Column names: seller, offerType, price, abtest, vehicleType, gearbox, ...
cars_data	DataFrame	(50001, 16)	Column names: dataCrawled, name, seller, ...

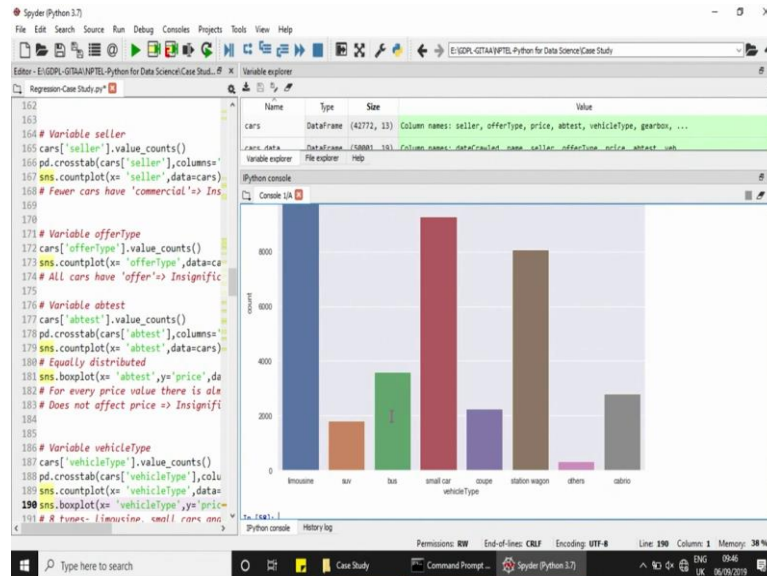
Python console:

```
In [55]: cars['vehicleType'].value_counts()
Out[55]:
limousine    11746
small car    9285
station wagon 8076
bus          2997
cabrio       2792
coupe        2261
suv          1813
others       326
Name: vehicleType, dtype: int64

In [56]: pd.crosstab(cars['vehicleType'], columns='count', normalize=True)
Out[56]:
col_# count
vehicleType
cabrio    0.890
coupe     0.879
limousine 0.294
others    0.888
small car 0.233
station wagon 0.282
suv       0.845
```

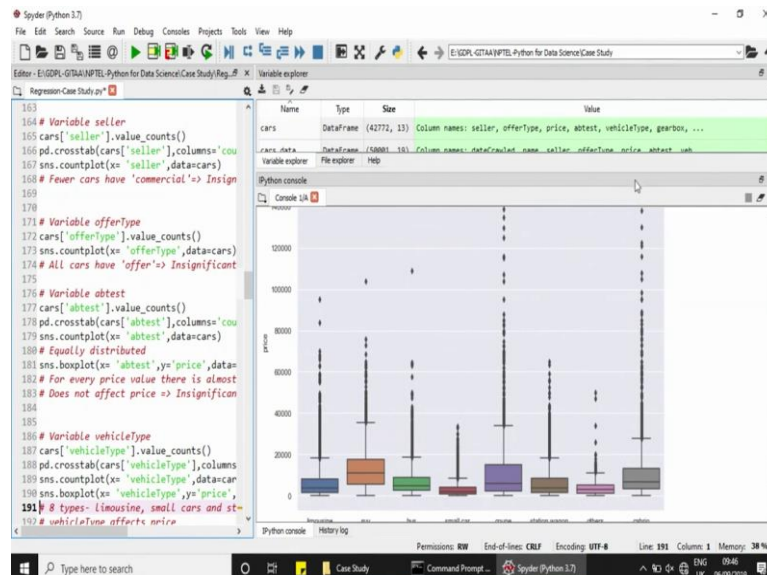
This is vehicleType and if you do a value counts for them you will see that the category topping the charts is limousine, followed by that you have small car, and then station wagon, and then you have bus, and cabrio, coupe and others, SUV, and then others, right. So, for a very proportion based account you can see that limousine is the highest with about 29 percent and let us just try to understand the proportion or the frequency better.

(Refer Slide Time: 14:04)



From the products see more easier to understand that limousine is contributing the most and followed by that you have small cars, station wagon and bus, right. So, this is the order. It is more easier to understand from the plot than from the proportion count when under each variable you have several categories, ok.

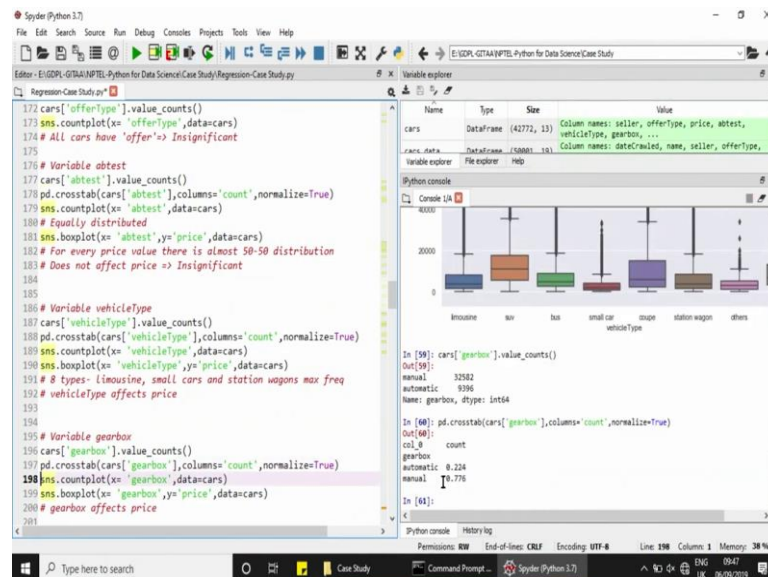
(Refer Slide Time: 14:28)



So, let us just see its effect on price. So, I am plotting a box plot of vehicleType with price. I have just dragged the output window to show you how does the screen look. So, for different categories under vehicleType, you can see that there are different price

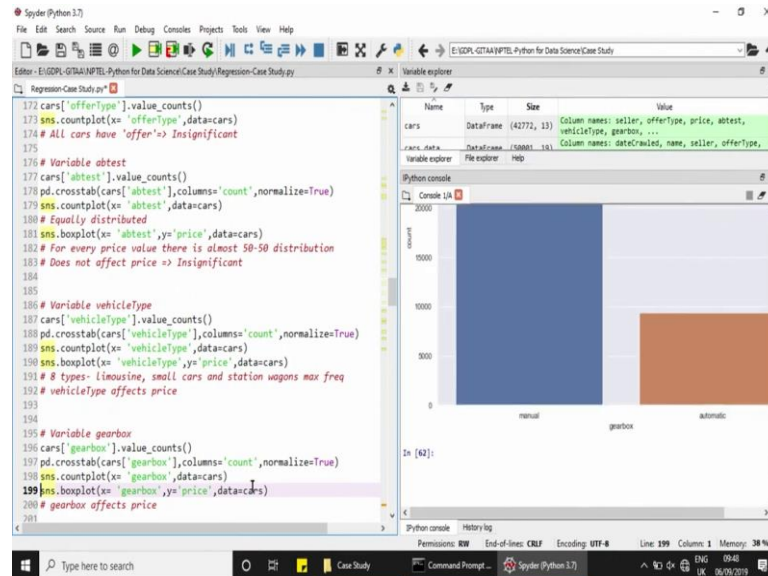
ranges, right. So, it is all it is all different, there is nothing constant about each category and limousine again is very very different from you know coupe or the bus or station wagon. So, the price are different for different categories. So, yes, vehicleType does affect price and we are going to retain it in our model.

(Refer Slide Time: 15:17)



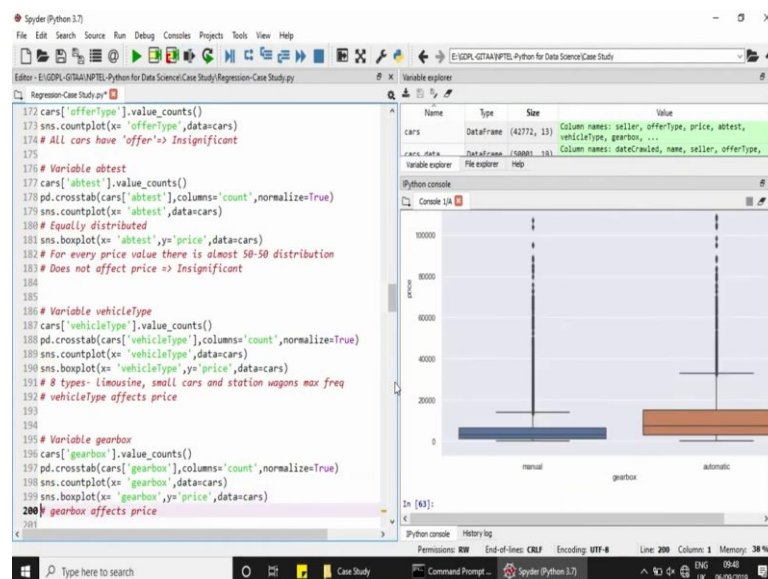
So, the next variable is gearbox. Let us look at the count for gearbox. You have 32582 records under manual and the remaining 9396 under automatic and these are the two categories that are available. And if you want a proportion based view you can see that manual counts for about 78 percent roughly and automatic accounts for roughly about 22 percent, right. And again we can; and again we can do a count plot for it which will also show you that manual gearbox are the most frequent.

(Refer Slide Time: 15:47)



And from your box plot let us see if the gearbox has an effect on price.

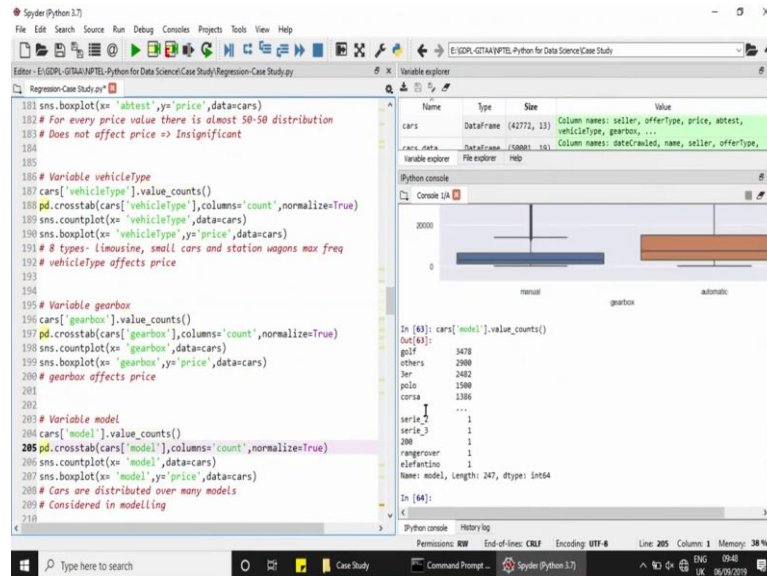
(Refer Slide Time: 15:53)



And yes, it does because manual is priced lower compared to automatic, right and that is evident from the box plot itself. So, yes, gearbox does affect price and we are going to retain gearbox in our final model.

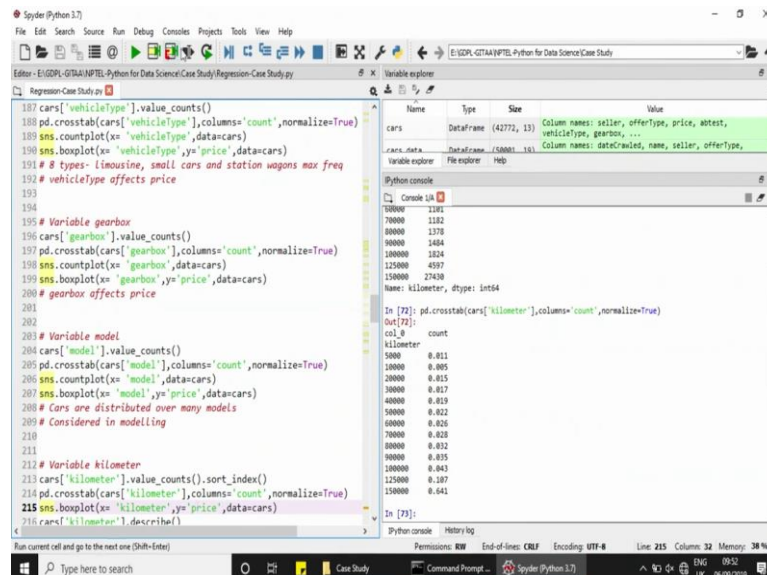


(Refer Slide Time: 16:14)



The next is the variable model which is the model of the car and if you look at the frequency count you can see that you have about 247 models, right and of course, we cannot know the value count for all, but the model golf seems to be the most frequently occurring. So, moving further we are going to look into the variable kilometer.

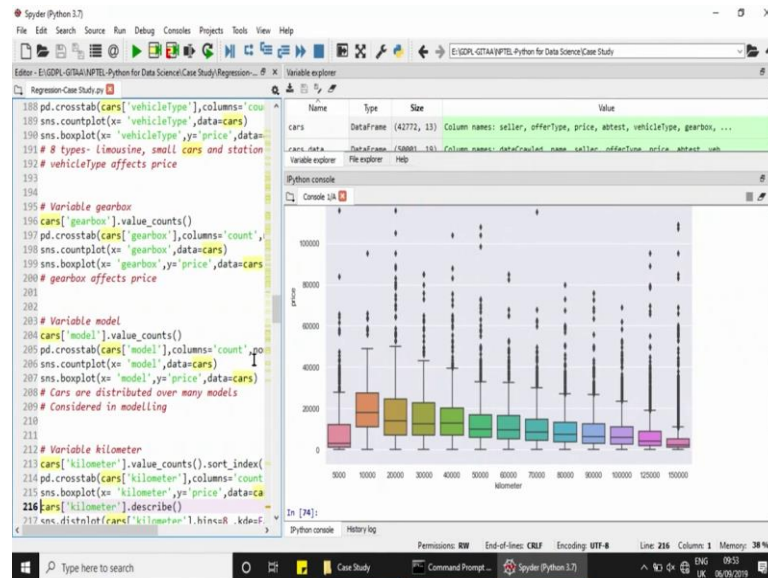
(Refer Slide Time: 16:36)



Now, to check the value counts I have sorted out the kilometers in the increasing order based on the index using the sort index function and you can see that the kilometer which is equal to 150000 contributes the maximum and in terms of and in terms of proportion if

you see it contributes to about roughly 64 percent. So, it is the most frequently occurring kilometer. So, most of the cars have a kilometer of 150000. And let us see if its effect on price and I am doing a boxplot for it.

(Refer Slide Time: 17:10)



So, you can see here that different kilometers affect the price in a very very different sense and kilometers, so if cars that have traveled lower kilometers like 10000 and 20000 are going for are being sold or go to rent, lease for a much higher price. But whereas, cars which have gone whereas, cars which have had traveled for a lot of kilometers eventually their price drops, except for this one variable, except for this one case where the kilometer is at 5000 and the median of the price is very very less.

So, this is the only outlier in terms of the behavior, but otherwise with increase in kilometer you will see that the price also decreases. So, eventually when the car has traveled more, you will see that there is more wear and tear, and hence the price drops, ok. So, this is a take home message with this variable.

(Refer Slide Time: 18:09)

```
200 # gearbox affects price
201
202
203 # Variable model
204 cars['model'].value_counts()
205 pd.crosstab(cars['model'], columns='count', normalize=True)
206 sns.countplot(x='model', y='price', data=cars)
207 # Cars are distributed over many models
208 # Considered in modelling
209
210
211 # Variable kilometer
212 cars['kilometer'].value_counts().sort_index()
213 pd.crosstab(cars['kilometer'], columns='count', normalize=True)
214 sns.boxplot(x='kilometer', y='price', data=cars)
215 cars['kilometer'].describe()
216 sns.distplot(cars['kilometer'], bins=8, kde=False)
217 sns.regplot(x='kilometer', y='price', scatter=True,
218             fit_reg=False, data=cars)
219 # Considered in modelling
220
221
222 # Variable fuelType
223 cars['fuelType'].value_counts()
224 pd.crosstab(cars['fuelType'], columns='count', normalize=True)
225 sns.countplot(x='fuelType', y='price', data=cars)
226 # fuelType affects price
227
```

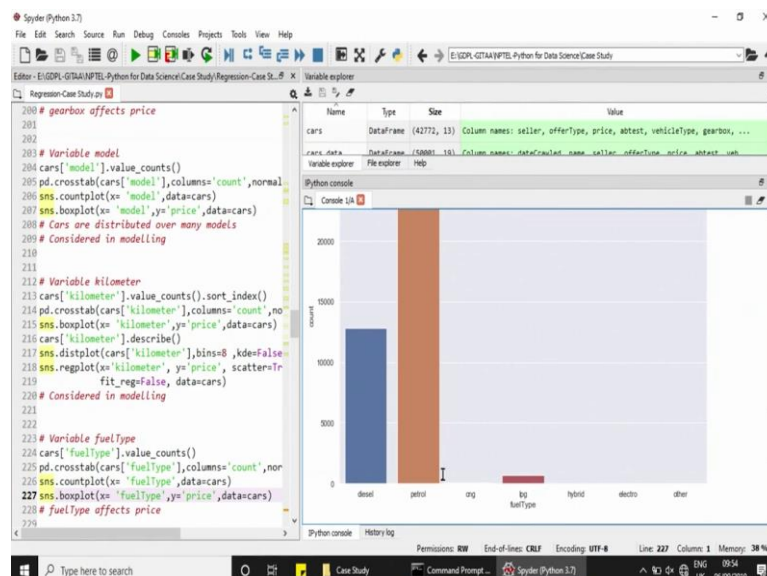
Name	Type	Size	Value
cars	DataFrame	(42772, 13)	Column names: seller, offerType, price, abtest, vehicleType, gearbox, ...
cars_data	DataFrame	(50801, 18)	Column names: dataCrawled, name, seller, offerType, price, ...

```
In [74]: cars['fuelType'].value_counts()
Out[74]:
petrol    26589
diesel    12854
lpg        690
cng        70
hybrid    36
electro    18
other       6
Name: fuelType, dtype: int64

In [75]: pd.crosstab(cars['fuelType'], columns='count', normalize=True)
Out[75]:
fuelType count
cng        0.002
diesel     0.329
electro    0.000
hybrid     0.001
lpg        0.017
other      0.000
petrol     0.668
```

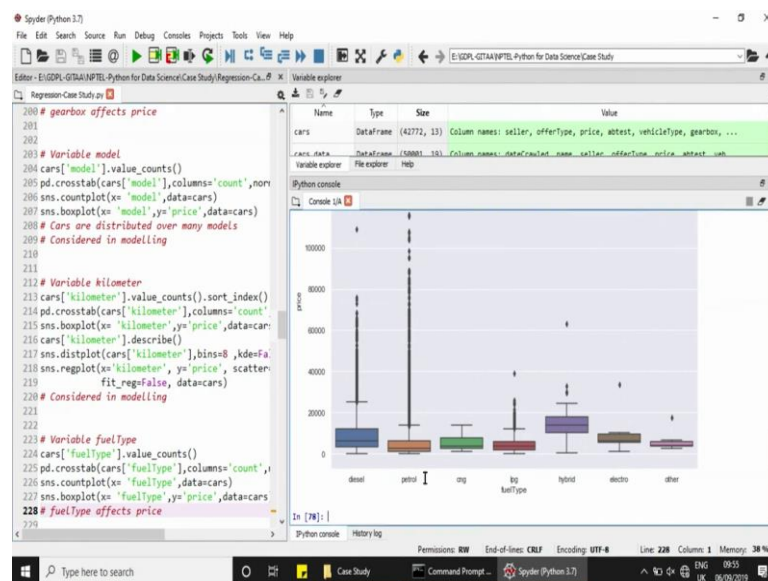
So, let us now look at the variable fuelType. So, we are going to retain kilometer and now let us look at the variable fuelType, and if you do a value counts for it you will see that petrol is topping with a frequency of over 26500, followed by that you have diesel and then you have LPG and then you have CNG, CNG hybrid electro and others, they all contribute for very very small fraction. And in terms of proportion, you will see petrol is leading with about 66 percent.

(Refer Slide Time: 18:45)



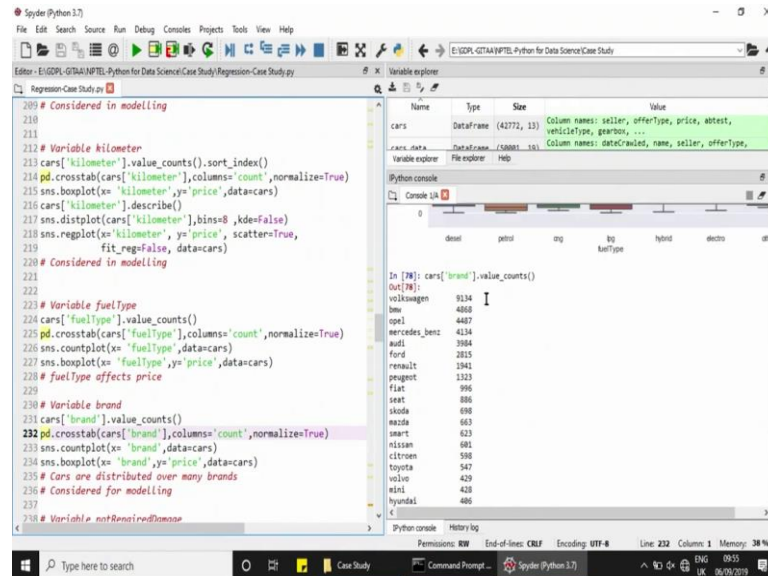
Now, let us just understand this better in terms of a bar plot. On my right you have the bar plot. So, you can see that here diesel, you can see here that petrol is contributing the maximum followed by that you have diesel and then you have LPG, but we cannot even see the bars for the others because they are very very less. And the effect of and the count of petrol and diesel is actually smearing out the effect of others which is obvious. So, now, let us just try to understand if fuelType affects price.

(Refer Slide Time: 19:18)



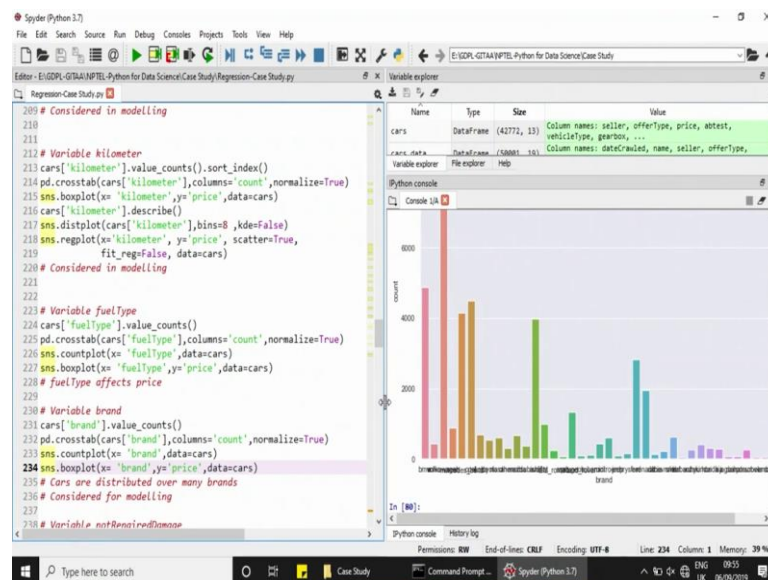
Now, definitely it does, so from the. So, definitely it affects price and that is clear from the box plot because different categories of fuelTypes has different price ranges and that is very very clear from the boxplot itself. So, yes, we are going to retain fuelType in our modeling. Let us move on to the variable brand.

(Refer Slide Time: 19:39)



So, if you do a frequency count you will see a whole lot of brands here, but the one that stopping the chart is Volkswagen with about 9134 occurrences and followed by that you have BMW and Opel and Mercedes Benz and so on and so forth, right.

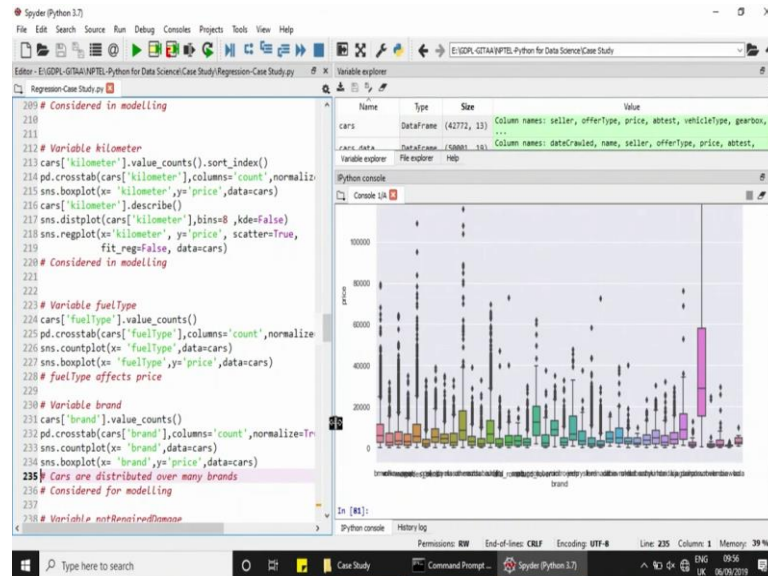
(Refer Slide Time: 20:01)



So, again if you do a count plot you will see different ranges for it. This is how the count plot looks for all the brands and to understand if brand would it definitely affect price, we are doing going for a box plot.

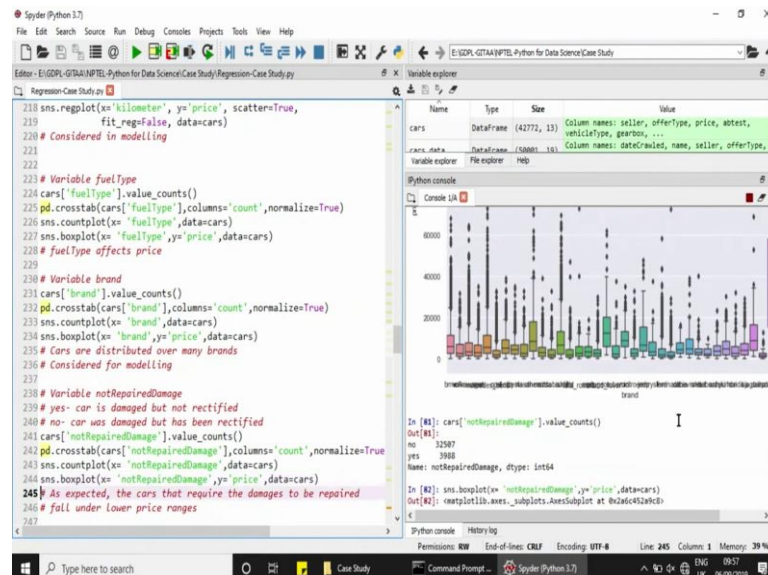


(Refer Slide Time: 20:14)



It is very very clear from the box plot that all the brands do have an effect, but were not able to see which of the boxes correspond to which brand that is not evident from the box plot, but on an overall note we can see that different brands have different price ranges. So, the last variable that we are going to look into is not repaired or damaged.

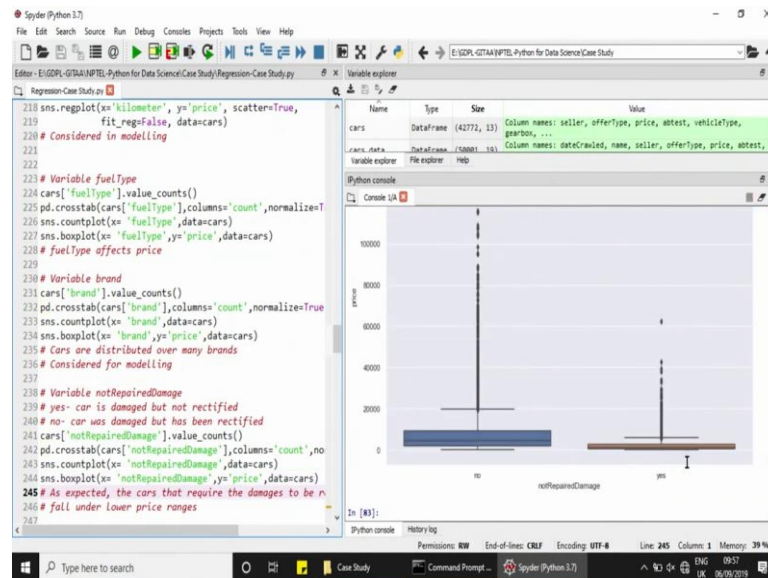
(Refer Slide Time: 20:33)



Now, since these are secondhand cars, there is already some damage that has happened to these cars. Now, this variable will tell you whether has that damage being rectified or has that damage is just being retained and not looked after, right. Now, if there are two

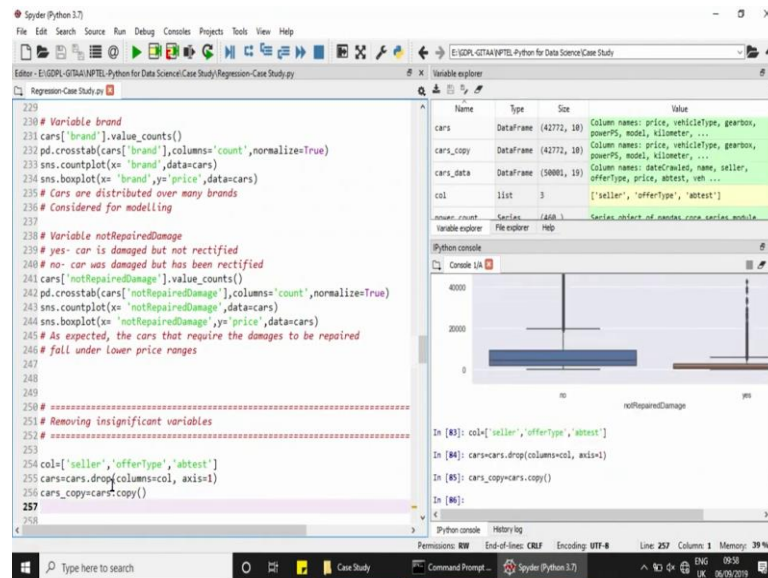
levels to, so there are two categories to this variable, one is yes, the other is no. When you have yes, it means that the car was damaged, but the damage has not been rectified and the second category says that no the car was damaged, but the damage has been looked after, right. So, these are the two levels. So, let us look at the value counts for this. So, the category with no is 32507 and the category yes is 30988.

(Refer Slide Time: 21:27)



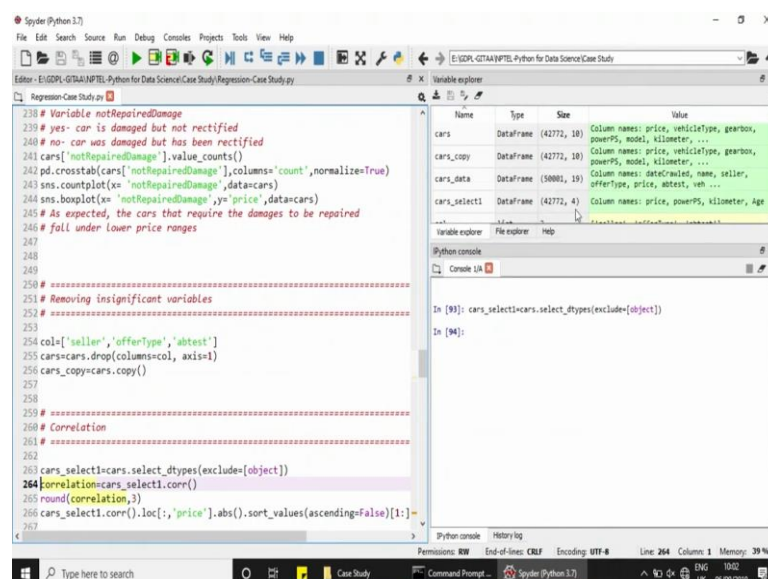
But if you want to see their effect on price you will have to do a box plot which eventually tells you that cars where the damage has not been rectified are under the category yes and they have a lower price range. So, cars which where the damage has not been looked after too, are going are being sold or quoted for a lower price range compared to the cars where the damage has been rectified. So, there is a clear cut difference between the two categories and their effect on price, and hence we are going to retain the variable notRepairedDamage, right.

(Refer Slide Time: 22:00)



So, now, I am going to just remove this insignificant variables here which are seller offer type and abtest from the car. So, I am just collecting their names as a list and I am using that cars.drop function, and you can see that from 13 we are down to 10 and that is from the cars data frame, right. So, before we go further let us just make a copy of this data here. So, this is just to make sure that we do not end up losing the analysis that we have done till now.

(Refer Slide Time: 22:30)



So, before we move further let us look at the correlation between our numerical variables and I am using `.select_dtypes` which will select a certain ranges of data based on the data type that we have, and for this under the `exclude` parameter I am excluding all variables which are of the type object.

(Refer Slide Time: 22:52)

The screenshot shows the Spyder Python IDE. The main editor displays a DataFrame named `cars_select1` with the following columns: `price`, `powerPS`, `kilometer`, and `Age`. The variable explorer on the right shows the `cars_select1` DataFrame with its columns listed. The code in the editor includes comments and a `.select_dtypes(exclude='object')` operation.

Index	price	powerPS	kilometer	Age
0	4438	158	150000	15.25
1	13299	163	150000	13.5
2	3200	181	150000	15.92
3	4500	86	60000	13
4	18750	185	150000	18.92
5	900	90	150000	23.17
7	1399	136	150000	21.92
8	4600	122	150000	14
9	8340	140	121000	13.33
10	1870	82	150000	17.88
11	2500	185	121000	17.33
12	990	68	150000	17.75
13	3000	116	150000	2
14	2200	83	121000	15.42

Once I do that you can see that `cars_select1` gets displayed and we have 4, columns which are `price`, `powerPS`, `kilometer` and `age` and I want to calculate the correlation between them.

(Refer Slide Time: 22:57)

The screenshot shows the Spyder Python IDE. The main editor displays the correlation matrix for the selected variables. The variable explorer on the right shows the `cars_select1` DataFrame with its columns listed. The code in the editor includes comments and a `.corr()` operation.

	price	powerPS	kilometer	Age
price	1.000	0.575	-0.440	-0.336
powerPS	0.575	1.000	-0.816	-0.151
kilometer	-0.440	-0.816	1.000	0.292
Age	-0.336	-0.151	0.292	1.000

So, I am using the `corr` function for this and I am rounding off the correlation to 3 decimal places. So, you have the variables again on the x and on the y axis, you have the same variables, it is a matrix, it is a correlation matrix and from here it is little hard to interpret which of the variables have the highest correlation value.

So, I have cars underscore `select.corr` which returns the correlation and I am from there I am doing a `location` function based on the price, right and this just returns the price and the first column. So, all that returns is the first column, right. And I am just taking the absolute value of it and I am just sorting them in descending order.

So, I want values after price and you can see that this is the correlation and powerPS has the highest correlation which is about 0.575 followed by that you have kilometer at 0.44 and age is correlated to price by the value of 0.336, right. These values are not high except for powerPS which shows a reasonable correlation. This is only, this method was to only check if there is some, if there is a heavy dependency of one of these variables or one of these numerical variables with price and that was the only reason to check the correlation.

So, except for probably powerPS, the other two variables do not have a very very heavy influence on price. Again the value of the powerPS, again the correlation value of powerPS and price is less, but it is not too less, it is referring about 0.57 and that is ok, but it is not very very strongly correlated to price.