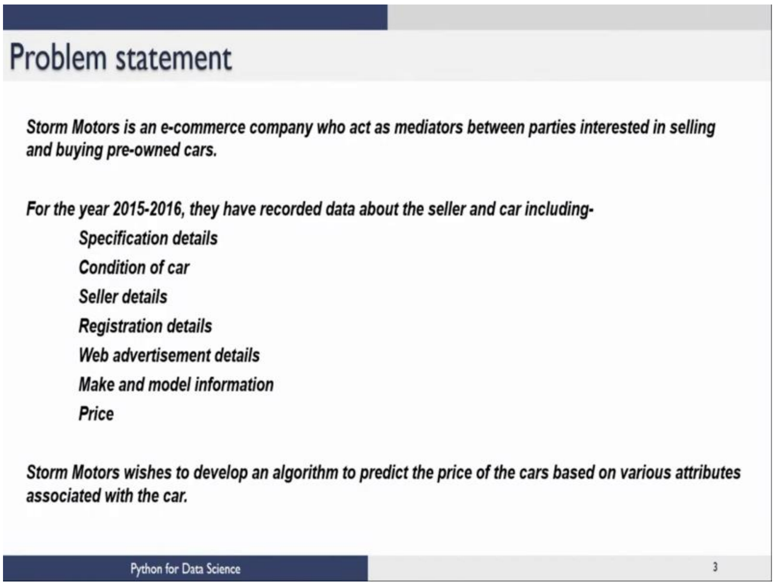
Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 43
Case study on regression

Welcome to the Case study on regression. In this case study we are going to take a problem that is on predicting price of pre owned cars.

(Refer Slide Time: 00:19)



Problem statement

Storm Motors is an e-commerce company who act as mediators between parties interested in selling and buying pre-owned cars.

For the year 2015-2016, they have recorded data about the seller and car including-

- *Specification details*
- *Condition of car*
- *Seller details*
- *Registration details*
- *Web advertisement details*
- *Make and model information*
- *Price*

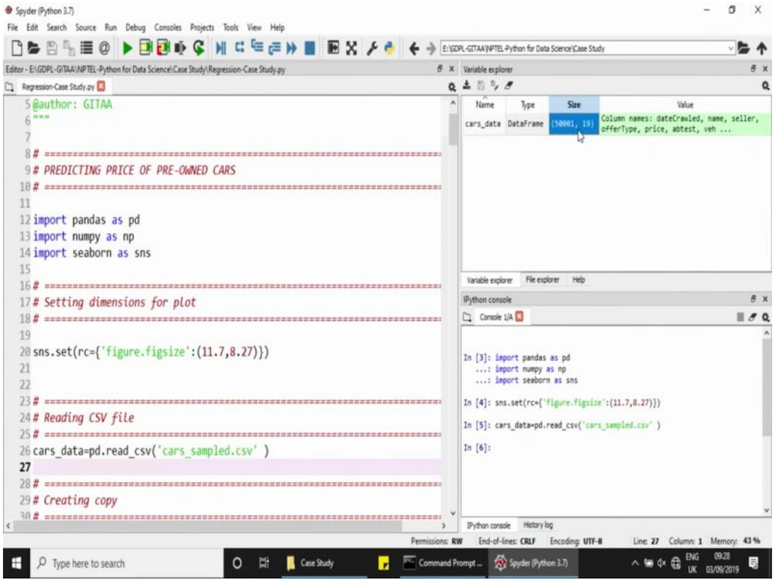
Storm Motors wishes to develop an algorithm to predict the price of the cars based on various attributes associated with the car.

Python for Data Science 3

So, let us go ahead and take a look at the problem statement. Now storm motors they are any commerce company and they act as mediators between parties who are interested in selling or buying a pre owned cars. Now these are second hand cars and storm motors act as mediators. Now in specific for the year 2015 to 2016, storm motors they have recorded the data about the seller and car details.

Now, these set of details includes specification details, condition of the car, seller details, registration details, web advertisement details, make and model information and price. Now these are several buckets and there are specific parameters or variables that storm motors as collected and each of these variables will definitely fall into one of these buckets. Now storm motors what they want to do is that, they want to develop an algorithm that will help them predict the price of the pre-own cars and this can be based on the various attributes that are associated with the car.

(Refer Slide Time: 01:23)



```
@author: GITAA
"""


# =============================================================================
# PREDICTING PRICE OF PRE-OWNED CARS
# =============================================================================

import pandas as pd
import numpy as np
import seaborn as sns

# =============================================================================
# Setting dimensions for plot
# =============================================================================

sns.set(rc={'figure.figsize':(11.7,8.27)})


# =============================================================================
# Reading CSV file
# =============================================================================
cars_data=pd.read_csv('cars_sampled.csv' )

# =============================================================================
# Creating copy
# =============================================================================
```

Name	Type	Size	Value
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...

```
In [3]: import pandas as pd
   ...: import numpy as np
   ...: import seaborn as sns

In [4]: sns.set(rc={'figure.figsize':(11.7,8.27)})

In [5]: cars_data=pd.read_csv('cars_sampled.csv' )

In [6]:
```

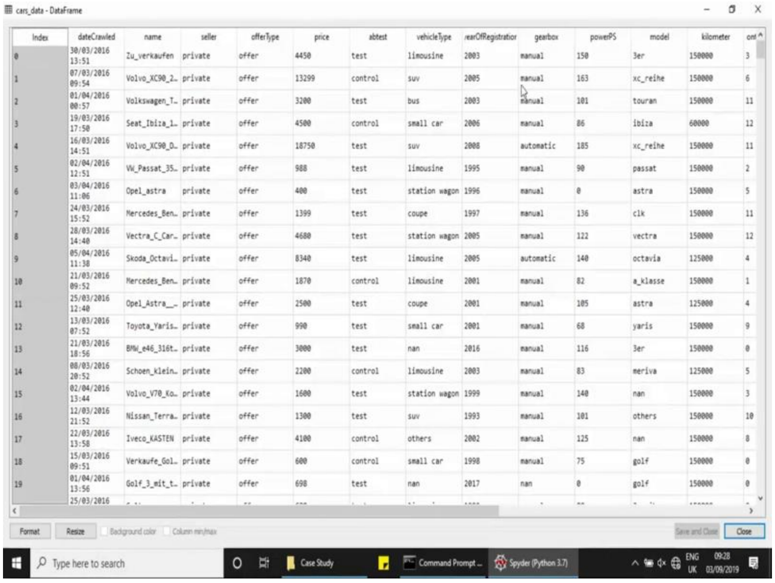
Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study
Editor - E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study\Regression-Case Study.py
Regression-Case Study.py
Variable explorer
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 27 Column: 1 Memory: 43 %
Type here to search
Case Study
Command Prompt ...
Spyder (Python 3.7)
ENG UK 09:28 03/09/2019

So, let us see how to solve this case study in python. So, let us start by importing a few required packages, first we have going to import pandas for reading all the files from the CSV format, then I am going to do a few numerical operation and hence I am going to import numpy as well. And I am going to visualize the data and derive some insides and going to import seaborn for it. So, I am importing pandas as pd numpy as np and see born as sns. So, let us just import these packages.

So, these packages have been imported now. Now I am setting a dimensions for all the plots that I am going to be generating, I am using the function sns touch set is a function from the seaborn package and within the parenthesis, I am giving the figure size and this sets the dimensions for rather. And this will set the dimensions for all our plots. So, let us begin by importing the CSV file. So, you have been given a data called cars_sample.csv. Now I have already set my working directory and the data is also residing in my working directory.

So, I am just going to right away read it using the pd.read_csv function. So, the data has been read and we have 50001 records and 19 columns let us just open and see what does the data have to say.

(Refer Slide Time: 03:01)

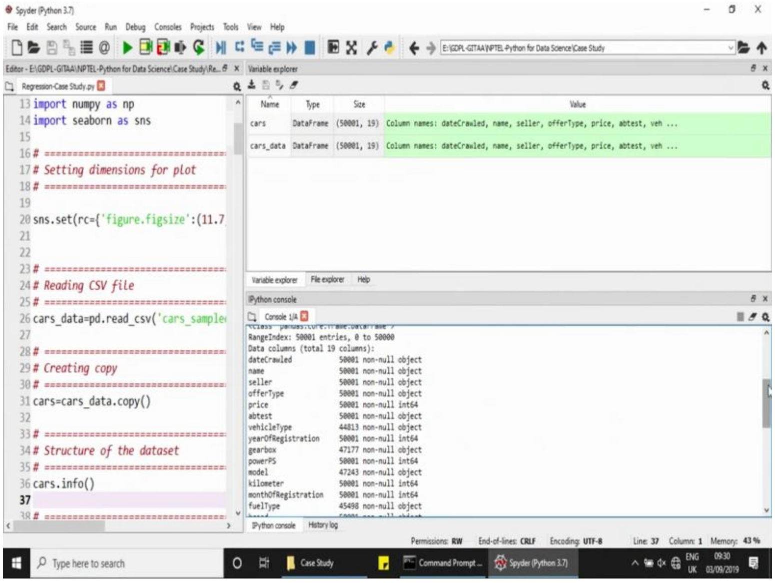


cars_data - DataFrame

Index	dateCrawled	name	seller	offerType	price	abtest	vehicleType	rearOfRegistratior	gearbox	powerPS	model	kilometer	ont
0	30/03/2016 13:51	Zu_verkaufen	private	offer	4450	test	limousine	2003	manual	150	3er	150000	3
1	07/03/2016 09:54	Volvo_XC90_2…	private	offer	13299	control	suv	2005	manual	163	xc_reihe	150000	6
2	01/04/2016 00:57	Volkswagen_T…	private	offer	3200	test	bus	2003	manual	101	touran	150000	11
3	19/03/2016 17:50	Seat_Ibiza_1…	private	offer	4500	control	small car	2006	manual	86	ibiza	60000	12
4	16/03/2016 14:51	Volvo_XC90_D…	private	offer	18750	test	suv	2008	automatic	185	xc_reihe	150000	11
5	02/04/2016 12:51	VW_Passat_35…	private	offer	988	test	limousine	1995	manual	90	passat	150000	2
6	03/04/2016 11:06	Opel_astra	private	offer	400	test	station wagon	1996	manual	0	astra	150000	5
7	24/03/2016 15:52	Mercedes_Ben…	private	offer	1399	test	coupe	1997	manual	136	clk	150000	11
8	28/03/2016 14:40	Vectra_C_Car…	private	offer	4680	test	station wagon	2005	manual	122	vectra	150000	12
9	05/04/2016 11:38	Skoda_Octavi…	private	offer	8340	test	limousine	2005	automatic	140	octavia	125000	4
10	21/03/2016 09:52	Mercedes_Ben…	private	offer	1870	control	limousine	2001	manual	82	a_klasse	150000	1
11	25/03/2016 12:40	Opel_Astra__…	private	offer	2500	test	coupe	2001	manual	105	astra	125000	4
12	13/03/2016 07:52	Toyota_Yaris…	private	offer	990	test	small car	2001	manual	68	yaris	150000	9
13	21/03/2016 18:56	BMW_e46_316t…	private	offer	3000	test	nan	2016	manual	116	3er	150000	0
14	08/03/2016 20:52	Schoen_klein…	private	offer	2200	control	limousine	2003	manual	83	meriva	125000	5
15	02/04/2016 13:44	Volvo_V70_Ko…	private	offer	1600	test	station wagon	1999	manual	140	nan	150000	3
16	12/03/2016 21:52	Nissan_Terra…	private	offer	1300	test	suv	1993	manual	101	others	150000	10
17	22/03/2016 13:58	Iveco_KASTEN	private	offer	4100	control	others	2002	manual	125	nan	150000	8
18	15/03/2016 09:51	Verkaufe_Gol…	private	offer	600	control	small car	1998	manual	75	golf	150000	0
19	01/04/2016 13:56	Golf_3_mit_t…	private	offer	698	test	nan	2017	nan	0	golf	150000	0
	25/03/2016												

So, you have index from 0 to 50,001 you have the date crawled you have the name of the car; the seller type, the offerType the price you have a b test then vehicle type and year of registration and there are couple of other variables here, all these have been explain to you before the description of the variables. So, now, that we have read let us just create a copy of it. So, you have already seen how to create a copy of your data. So, you can do a deep copy and any change that you made to cars will not be reflected back in cars_data.

(Refer Slide Time: 03:39)



```
import numpy as np
import seaborn as sns

# =================================
# Setting dimensions for plot
# =================================

sns.set(rc={'figure.figsize':(11.7


# =================================
# Reading CSV file
# =================================
cars_data=pd.read_csv('cars_sample

# =================================
# Creating copy
# =================================
cars=cars_data.copy()

# =================================
# Structure of the dataset
# =================================
cars.info()

# =================================
```

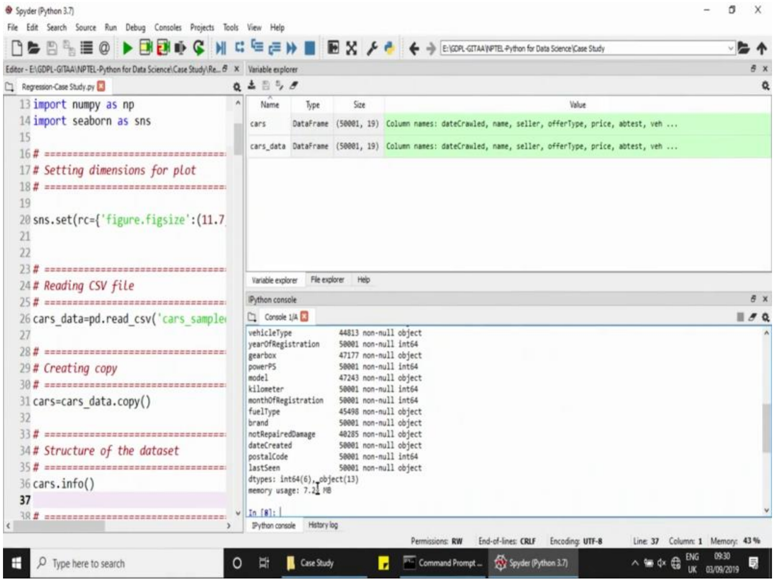
Name	Type	Size	Value
cars	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50001 entries, 0 to 50000
Data columns (total 19 columns):
dateCrawled            50001 non-null object
name                   50001 non-null object
seller                 50001 non-null object
offerType              50001 non-null object
price                  50001 non-null int64
abtest                 50001 non-null object
vehicleType            44813 non-null object
yearOfRegistration     50001 non-null int64
gearbox                47177 non-null object
powerPS                50001 non-null int64
model                  47243 non-null object
kilometer              50001 non-null int64
monthOfRegistration    50001 non-null int64
fuelType               45498 non-null object
brand                  50001 non-null object
```

So, I made a deep copy and I saved it as cars and we are going to be using cars to work here after. So, let us start by looking at the structure of the data, I am using the cars.info. So, this gives you the structure of the data. Now this tells me that is the data frame it has entries ranging from 0 to 50,000 and the total number of columns are 19 here what you also have is a number of non null objects and what is the data type of each of the column.

So, date crawled has 50,001non null object it is an object data type similarly you have name for seller type, offerType, price and abtest all these are full and if you take vehicle type that is 44,813 non null entries. So, you have some missing values there and similarly even in gearbox model and fuel type this will give you just of all the column names and under each column the number of non null entries and what is the data type of each of the columns.

(Refer Slide Time: 04:49)


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study
Editor - E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study\Re...
Regression-Case Study.py
Variable explorer
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 37 Column: 1 Memory: 43 %
Type here to search
Case Study
Command Prompt
Spyder (Python 3.7)
ENG UK 09:30 03/09/2019

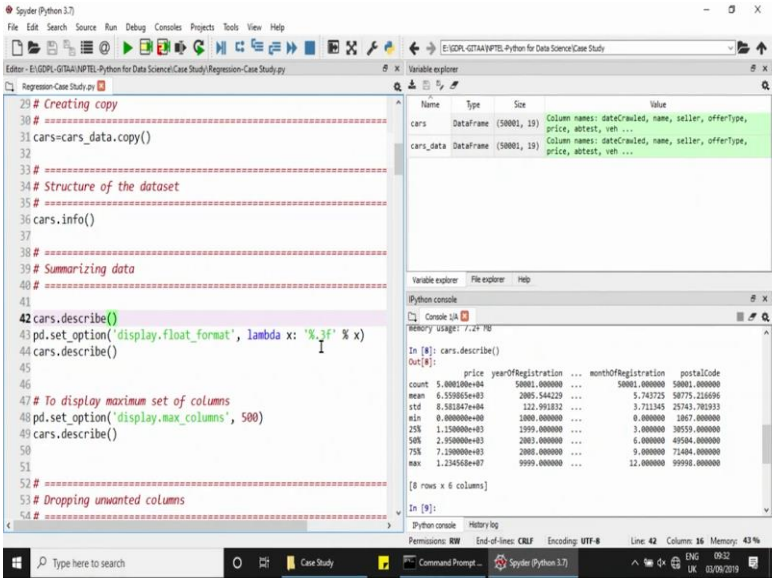
```
import numpy as np
import seaborn as sns

# ==================================
# Setting dimensions for plot
# ==================================

sns.set(rc={'figure.figsize':(11.7


# ==================================
# Reading CSV file
# ==================================
cars_data=pd.read_csv('cars_sample

# ==================================
# Creating copy
# ==================================
cars=cars_data.copy()

# ==================================
# Structure of the dataset
# ==================================
cars.info()

# ==================================
```

Name	Type	Size	Value
cars	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...

```
vehicleType            44813 non-null object
yearOfRegistration     50001 non-null int64
gearbox                47177 non-null object
powerPS                50001 non-null int64
model                  47243 non-null object
kilometer              50001 non-null int64
monthOfRegistration    50001 non-null int64
fuelType               45498 non-null object
brand                  50001 non-null object
notRepairedDamage      40285 non-null object
dateCreated            50001 non-null object
postalCode             50001 non-null int64
lastSeen               50001 non-null object
dtypes: int64(6), object(13)
memory usage: 7.2 MB

In [8]:
```

So, far as we have 6 columns which are of in 64 data type and there are 13 column which are of object data type. So, now, let us summarize the data I am going to be using the function cars.describe for this.

(Refer Slide Time: 05:03)



Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study\Regression-Case Study.py
Regression-Case Study.py
E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study
Variable explorer
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Type here to search
Case Study
Command Prompt ...
Spyder (Python 3.7)
ENG UK 09:32 03/09/2019

```
# Creating copy
# =============================================================================
cars=cars_data.copy()

# =============================================================================
# Structure of the dataset
# =============================================================================
cars.info()

# =============================================================================
# Summarizing data
# =============================================================================

cars.describe()
pd.set_option('display.float_format', lambda x: '%.3f' % x)
cars.describe()


# To display maximum set of columns
pd.set_option('display.max_columns', 500)
cars.describe()


# =============================================================================
# Dropping unwanted columns
# =============================================================================
```

Name	Type	Size	Value
cars	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...

```
memory usage: 7.2+ MB

In [8]: cars.describe()
Out[8]:
              price  yearOfRegistration  ...  monthOfRegistration   postalCode
count  5.000100e+04        50001.000000  ...         50001.000000 50001.000000
mean   6.559865e+03         2005.544229  ...             5.743725 50775.216696
std    8.581847e+04          122.991832  ...             3.711345 25743.701933
min    0.000000e+00         1000.000000  ...             0.000000  1067.000000
25%    1.150000e+03         1999.000000  ...             3.000000 30559.000000
50%    2.950000e+03         2003.000000  ...             6.000000 49504.000000
75%    7.190000e+03         2008.000000  ...             9.000000 71404.000000
max    1.234568e+07         9999.000000  ...            12.000000 99998.000000

[8 rows x 6 columns]

In [9]:
```

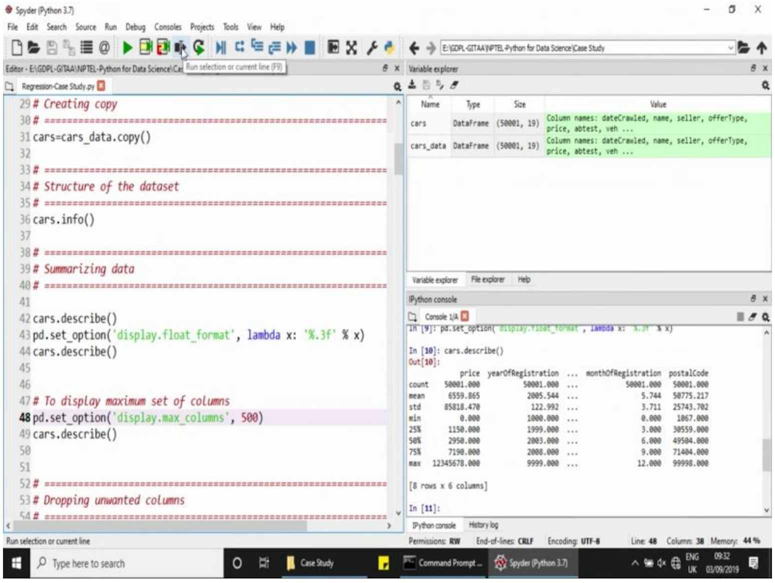
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 42 Column: 16 Memory: 43 %

So, when I used cars.describe you can see that there is a. So, when I use a we are do a.describe function, you can see the summary is given only for a few variables and it is not given for every variable there are some dots that you see in between; now to get rid of this you can use the.

So, when you do a summary you can see the summary is given when you do a.describe, you can see the description gives you the output in a scientific notation what you also see is that it displays the summary of only a few variables and the remaining variables are displayed as '…'. So, let us see how to get rid of the scientific notation first, I am going to use the function.set_option.

Within parenthesis what you have to give is displayed.float format because these are all float values and it is for these that you want to change the output. Now there is a lambda function that we are declaring inside. So, what I am saying here is converted to a three decimal place float value. So, let us just run this function again and then let us rerun cars.describe.

(Refer Slide Time: 06:23)



```
# Creating copy
# ================================================================
cars=cars_data.copy()

# ================================================================
# Structure of the dataset
# ================================================================
cars.info()

# ================================================================
# Summarizing data
# ================================================================

cars.describe()
pd.set_option('display.float_format', lambda x: '%.3f' % x)
cars.describe()


# To display maximum set of columns
pd.set_option('display.max_columns', 500)
cars.describe()


# ================================================================
# Dropping unwanted columns
# ================================================================
```

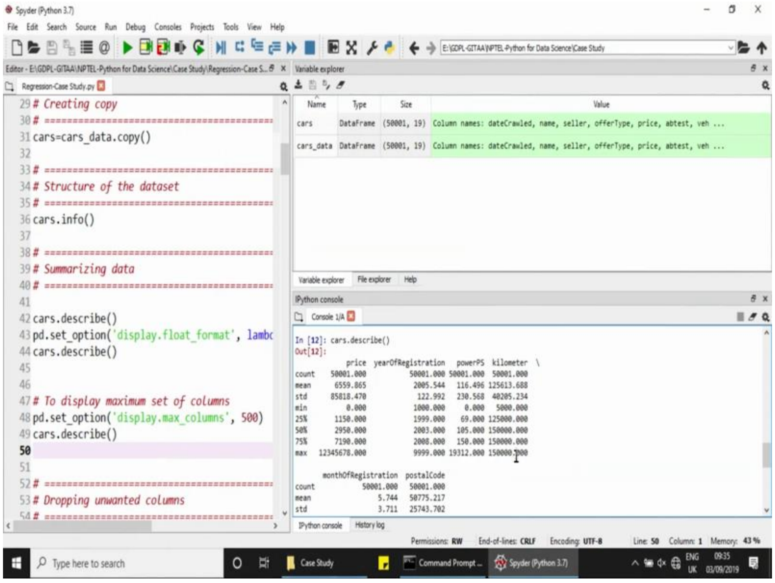
Name	Type	Size	Value
cars	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...

```
In [9]: pd.set_option('display.float_format', lambda x: '%.3f' % x)

In [10]: cars.describe()
Out[10]:
              price  yearOfRegistration  ...  monthOfRegistration  postalCode
count     50001.000           50001.000  ...            50001.000   50001.000
mean       6559.865            2005.544  ...                5.744   50775.217
std       85818.470             122.992  ...                3.711   25743.702
min           0.000            1000.000  ...                0.000    1067.000
25%        1150.000            1999.000  ...                3.000   30559.000
50%        2950.000            2003.000  ...                6.000   49504.000
75%        7190.000            2008.000  ...                9.000   71404.000
max    12345678.000            9999.000  ...               12.000   99998.000

[8 rows x 6 columns]

In [11]:
```

So, here you can see all the values have been rounded off to three decimal places, but we still and not able to get the description for all the columns. So, for that you need to use again the same function pd.set_option. So, within parenthesis you can give the parameter display.max_columns and it will display the maximum number of columns and the next parameter is basically how many of a columns you wanted to be display.

So, I have given a big number here 500, but usually you will not be seen 500 columns in a data, but this is just to set the maximum number of columns that can be displayed.

(Refer Slide Time: 07:03)



```
# Creating copy
# ==========================================
cars=cars_data.copy()

# ==========================================
# Structure of the dataset
# ==========================================
cars.info()

# ==========================================
# Summarizing data
# ==========================================

cars.describe()
pd.set_option('display.float_format', lambc
cars.describe()


# To display maximum set of columns
pd.set_option('display.max_columns', 500)
cars.describe()


# ==========================================
# Dropping unwanted columns
# ==========================================
```

Name	Type	Size	Value
cars	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...

```
In [12]: cars.describe()
Out[12]:
              price  yearOfRegistration   powerPS  kilometer  \
count     50001.000           50001.000 50001.000  50001.000
mean       6559.865            2005.544   116.496 125613.688
std       85818.470             122.992   230.568  40205.234
min           0.000            1000.000     0.000   5000.000
25%        1150.000            1999.000    69.000 125000.000
50%        2950.000            2003.000   105.000 150000.000
75%        7190.000            2008.000   150.000 150000.000
max    12345678.000            9999.000 19312.000 150000.000

       monthOfRegistration  postalCode
count            50001.000   50001.000
mean                 5.744   50775.217
std                  3.711   25743.702
```

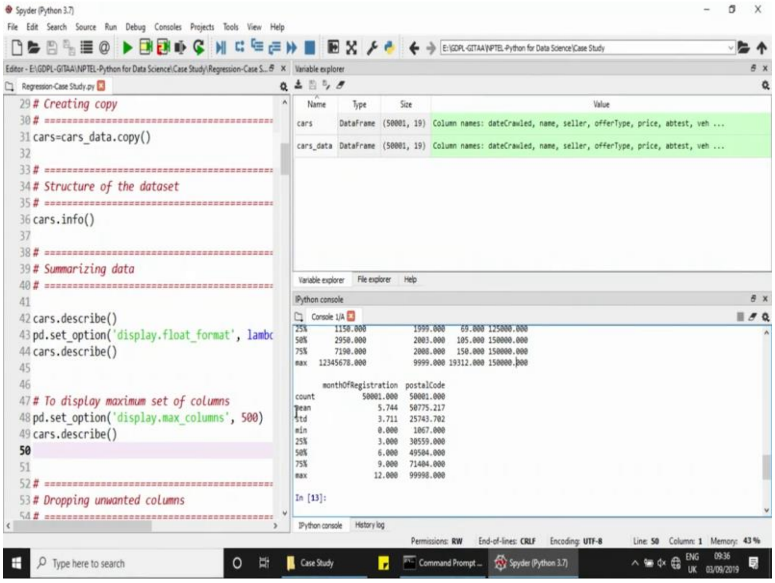
So, let us do that and let us read on cars.describe. So, now, you can see that all your columns are displayed here and you have a summary for all of it. Now price tells you that there are 50,000. So, under price if you check the count there are 50,001 records.

So, all the records having field and the mean is about 6559 and there is a standard there is a huge standard deviation, it is about 85,818 the minimum value is 0 the first quartile value is 1150, the second quartile which is also the median is 2950, the third quartile is 7190, but the maximum is very very huge and if you look at the difference between the mean and the median which is the second quartile, you can see there is a huge difference between the mean and the median there is itself shows you that price is very very secured.

And if you take care of registration you have 50,001 non null values the mean of year of registration is 2005, but this does not make sense because year of registration is an integer and it cannot be rounded off to a decimal.

But if you take the minimum of it you can see there are years ranging from 1000. Similarly, if you look at powerPS it again has 50,001 non null values, the mean being around 117 and if you look at the median, the mean being around 117, but if you look at the minimum value the minimum is 0 which does not make sense again and the first quartile is at around 70, the second quarter which is the median is around 105 and maximum is very very high, the maximum is around 19312 horse power. Now then you have kilometer, kilometer has again 50,001 non null entries, it is mean is about 1,25,613 kilometers, the minimum is again 5,000, the maximum being 1,50,000.

(Refer Slide Time: 08:57)



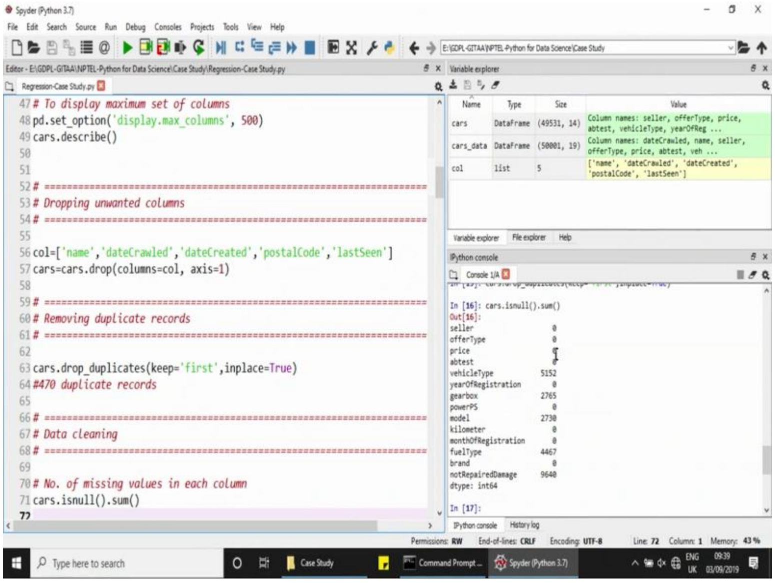
```
# Creating copy
# ==============================================
cars=cars_data.copy()

# ==============================================
# Structure of the dataset
# ==============================================
cars.info()

# ==============================================
# Summarizing data
# ==============================================

cars.describe()
pd.set_option('display.float_format', lambc
cars.describe()


# To display maximum set of columns
pd.set_option('display.max_columns', 500)
cars.describe()


# ==============================================
# Dropping unwanted columns
# ==============================================
```

Name	Type	Size	Value
cars	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...

```
25%       1150.000            1999.000    69.000 125000.000
50%       2950.000            2003.000   105.000 150000.000
75%       7190.000            2008.000   150.000 150000.000
max   12345678.000            9999.000 19312.000 150000.000

       monthOfRegistration  postalCode
count            50001.000   50001.000
mean                 5.744   50775.217
std                  3.711   25743.702
min                  0.000    1067.000
25%                  3.000   30559.000
50%                  6.000   49504.000
75%                  9.000   71404.000
max                 12.000   99998.000

In [13]:
```

Next you have month of registration. Month of registration does not have any missing values the minimum again is 0. But 0 does not make sense and postal code is something that we will be getting rid off because we are not going to be using for this case study, but you can you are free to use this variable if you want to do a more spatial based analysis right. So, again even for postal code, they are all integers and you cannot have decimal values.

(Refer Slide Time: 09:23)



```
# To display maximum set of columns
pd.set_option('display.max_columns', 500)
cars.describe()


# ==============================================================
# Dropping unwanted columns
# ==============================================================

col=['name','dateCrawled','dateCreated','postalCode','lastSeen']
cars=cars.drop(columns=col, axis=1)

# ==============================================================
# Removing duplicate records
# ==============================================================

cars.drop_duplicates(keep='first',inplace=True)
#470 duplicate records

# ==============================================================
# Data cleaning
# ==============================================================

# No. of missing values in each column
cars.isnull().sum()
```

Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, price, abtest, vehicleType, yearOfReg ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
col	list	5	['name', 'dateCrawled', 'dateCreated', 'postalCode', 'lastSeen']

```
In [16]: cars.isnull().sum()
Out[16]:
seller                    0
offerType                 0
price                     0
abtest                    0
vehicleType            5152
yearOfRegistration        0
gearbox                2765
powerPS                   0
model                  2730
kilometer                 0
monthOfRegistration       0
fuelType               4467
brand                     0
notRepairedDamage      9640
dtype: int64

In [17]:
```

So, let us just start by dropping unwanted columns up front. Now you have variables like name date crawled date created postal code and last seen, now we are not going to using these variables for our analysis and I am going to just drop them. So, I am creating a list of all the variable names here and once I run it all of them are stored under col and I am just going to use the.drop function and to drop them.

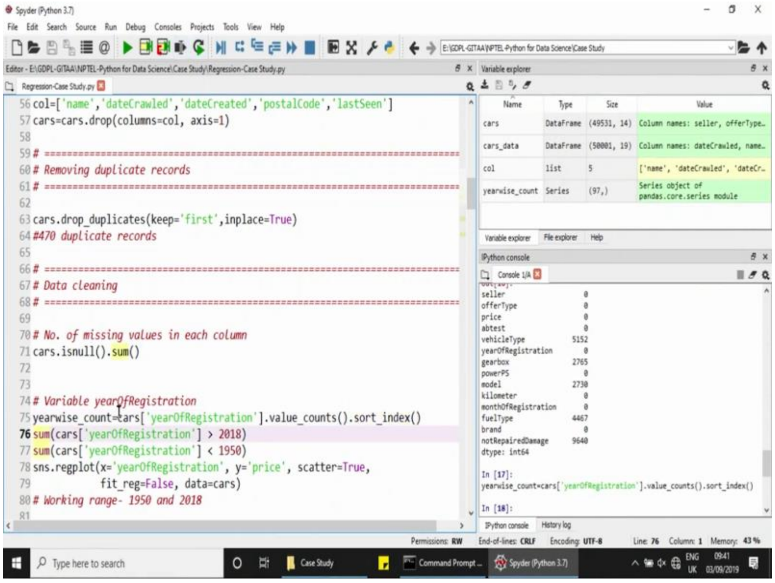
So, I do a cars.drop and I say columns is equal to col. So, these are the columns I want to drop, since these are columns I am giving the axis as 1. So, from 19 you can see the number of columns have come to 14, I have dropped 5 columns here. Now, we are going to see if there are any duplicate records in our data right and if there are any duplicate records, we are just going to keep the first occurrence of that such records. So, we started out with 50,001 records let us see when you do a.drop_duplicates how many are we will using. So, one 50,001 it is come down to 49531. Now that we have removed it let us get into data cleaning now.

So, my first task is to compute the number of missing values under each column, now from the.info function we saw not all column were filled right. So, now, we are going to find the number of missing values and each column and later on we will come up the method of how to logically fill such missing values. So, the first step is to basically calculate the number of missing values under each column. So, I am using the isnull function. So, isnull will basically return a data which is true or false.

It will mark the cells which are missing with true, but I am going to sum of the occurrences of the trues. So, I am just going to do it.sum for it. So, if you run the function you can see on my right that seller of a type price, ab test do not have any missing values now vehicle type has 5152 missing values, year of registration is again full, gearbox; however, has 2765 records missing.

And under model you can see you have about 2730 records missing. And under fuel you can see you have about 4467 records missing and about 9640 records are missing and not repaired or damaged. So, this is a column wise count of the number of cells that are missing.

(Refer Slide Time: 12:01)



```
col=['name','dateCrawled','dateCreated','postalCode','lastSeen']
cars=cars.drop(columns=col, axis=1)

# ==============================================================================
# Removing duplicate records
# ==============================================================================

cars.drop_duplicates(keep='first',inplace=True)
#470 duplicate records

# ==============================================================================
# Data cleaning
# ==============================================================================

# No. of missing values in each column
cars.isnull().sum()


# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018
```

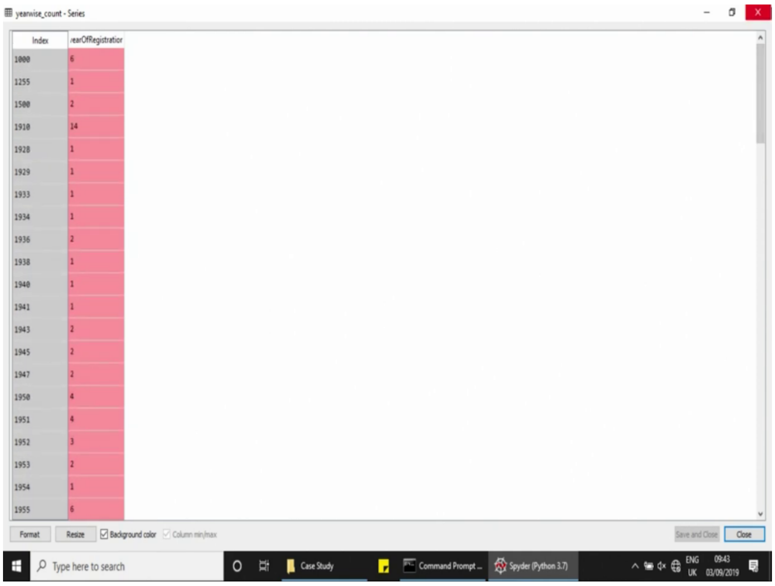
Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType…
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name…
col	list	5	['name', 'dateCrawled', 'dateCr…
yearwise_count	Series	(97,)	Series object of pandas.core.series module

```
seller                 0
offerType              0
price                  0
abtest                 0
vehicleType         5152
yearOfRegistration     0
gearbox             2765
powerPS                0
model               2730
kilometer              0
monthOfRegistration    0
fuelType            4467
brand                  0
notRepairedDamage   9640
dtype: int64

In [17]:
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()

In [18]:
```

So, now, let us just take year of registration first. So, in 975 what I am trying to do is I am trying to find year wise count based on this variable. Now, you can use the.value_counts here and I am sorting the index to make sure that, here year of registration is sorted based on the years and not on the counts. So, by default if you do a.value_counts, you would have seen that.

So, it gives you the category with the highest frequency on top, but I do not want that I basically wanted sorted placed on the years. So, let us just run this, this is too big output. So, I am saving it onto year wise under count variable.

(Refer Slide Time: 12:39)



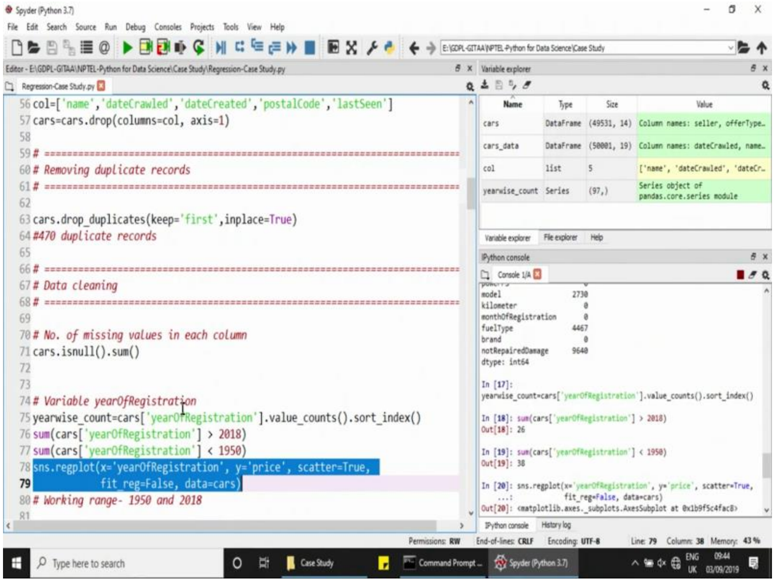
yearwise_count - Series

Index	earOfRegistratior
1000	6
1255	1
1500	2
1910	14
1928	1
1929	1
1933	1
1934	1
1936	2
1938	1
1940	1
1941	1
1943	2
1945	2
1947	2
1950	4
1951	4
1952	3
1953	2
1954	1
1955	6

Now, if you open and see. So, let us just see what this data frame has. It has index and it has year of registration index is the set of all years and year of registration on the that column you basically have the frequencies of each of these years. So, you have years that are ranging from 1000 and then you have year from 1910 and what you can also see if you scroll down is that, you have years which are in the future that is which are after 2019.

So, most of these do not make sense so, that is considerable amount of cleaning that we will have to do for this column. So, let us see how to come up with the strategy to clean them.

(Refer Slide Time: 13:19)



Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study
Editor - E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study\Regression-Case Study.py
Regression-Case Study.py
Variable explorer
File explorer
Help
IPython console
Console 1/A
History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 79 Column: 38 Memory: 43 %
Type here to search
Case Study
Command Prompt ...
Spyder (Python 3.7)
ENG UK 09:44 03/09/2019

```
col=['name','dateCrawled','dateCreated','postalCode','lastSeen']
cars=cars.drop(columns=col, axis=1)

# ==============================================================================
# Removing duplicate records
# ==============================================================================

cars.drop_duplicates(keep='first',inplace=True)
#470 duplicate records

# ==============================================================================
# Data cleaning
# ==============================================================================

# No. of missing values in each column
cars.isnull().sum()


# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018
```

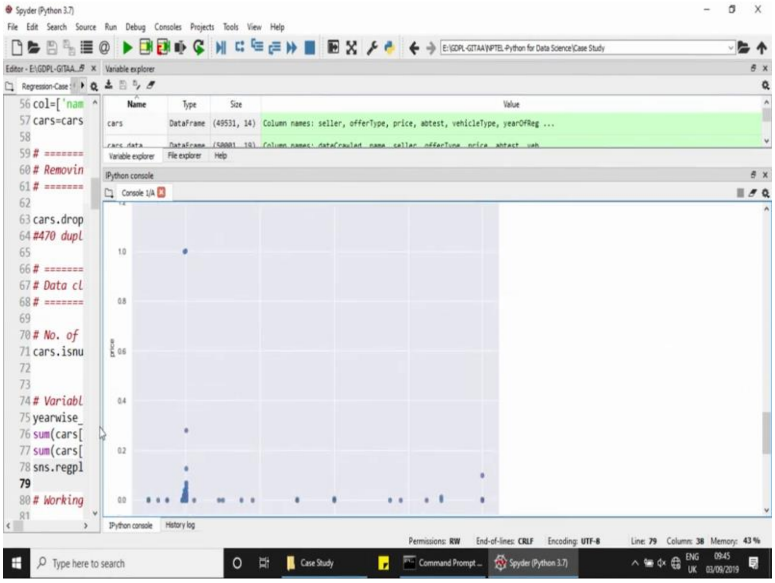
Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType…
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name…
col	list	5	['name', 'dateCrawled', 'dateCr…
yearwise_count	Series	(97,)	Series object of pandas.core.series module

```
model                  2730
kilometer                 0
monthOfRegistration       0
fuelType               4467
brand                     0
notRepairedDamage      9640
dtype: int64

In [17]:
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()

In [18]: sum(cars['yearOfRegistration'] > 2018)
Out[18]: 26

In [19]: sum(cars['yearOfRegistration'] < 1950)
Out[19]: 38

In [20]: sns.regplot(x='yearOfRegistration', y='price', scatter=True,
    ...:             fit_reg=False, data=cars)
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1b9f5c4fac8>
```

So, if you look at the sum of records which are greater than 2018 then that is only 26 right. This is 2 lesser number and we cannot predict in the future. We are in 2019, but I am not considering that because it is occurred only twice and it would not make sense to consider it. So, if you look at the number of cars where registration is greater than 2018.

Now, you have about 26 of them 26 records have a year that is greater than 2018. Now similarly if you do for a lower and check what you can also see is, I am taking the limit 1950. So, if you look at the cars that are made before or that are registered before in 1950, then you have about 38 records of that kind right. Now these are too less correct and they are going to smear or effect or you know sway the effect of the model. So, we are going to get rid of them. So, the working range that I am going to set is between 1950 and 2018. So, just to reconfirm this if you do a scatter plot I am using the sns.regplot.

(Refer Slide Time: 14:31)

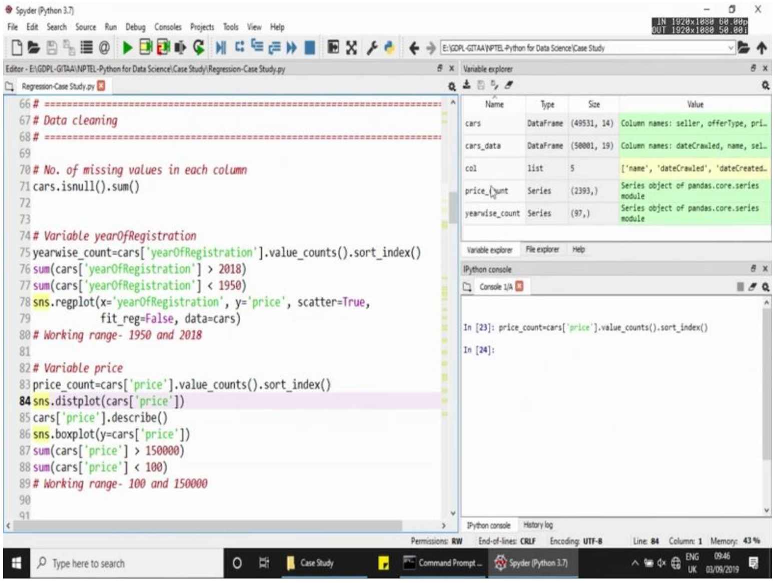


```
col=['nam
cars=cars

# =======
# Removin
# =======

cars.drop
#470 dupl

# =======
# Data cl
# =======

# No. of
cars.isnu


# Variabl
yearwise_
sum(cars[
sum(cars[
sns.regpl

# Working
```

Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, price, abtest, vehicleType, yearOfReg ...

If you look at this scatter plot, I have the year of registration on x axis and I have the price on the y axis, let me just drag it to the size just to show you a better plot. So, you can see that the plot is not explaining anything you have all only you have only dots here and there and that is because there are values which are very very high and they are just smearing out the effect on the other points. So, we have to clean this column to really understand what is the effect of year of registration on price.

(Refer Slide Time: 15:09)



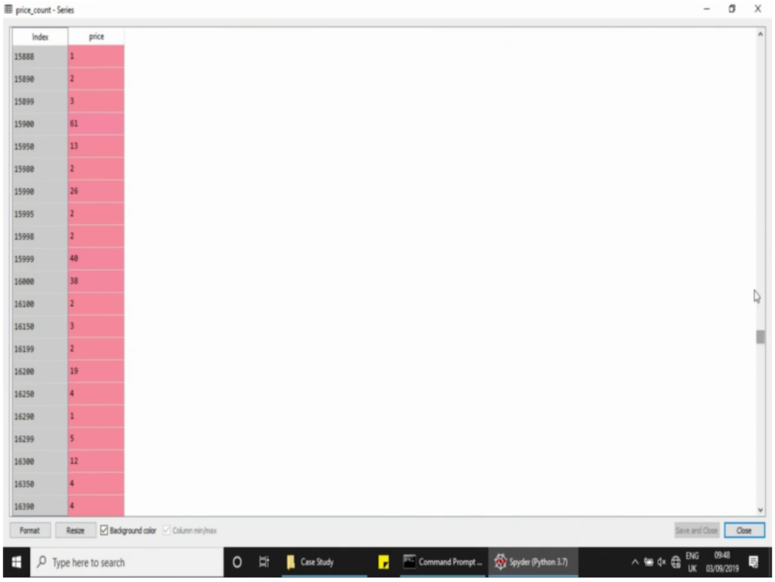
```
# ==============================================================================
# Data cleaning
# ==============================================================================

# No. of missing values in each column
cars.isnull().sum()


# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018

# Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000
```

Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, pri…
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, sel…
col	list	5	['name', 'dateCrawled', 'dateCreated…
price_count	Series	(2393,)	Series object of pandas.core.series module
yearwise_count	Series	(97,)	Series object of pandas.core.series module

```
In [23]: price_count=cars['price'].value_counts().sort_index()

In [24]:
```

So, that is as far as year of registration goes, the next is the variable price. Now if you again do a.value counts and then you sort based on the ascending order of the price and I am just saving it on to the variable price_count.

(Refer Slide Time: 15:25)

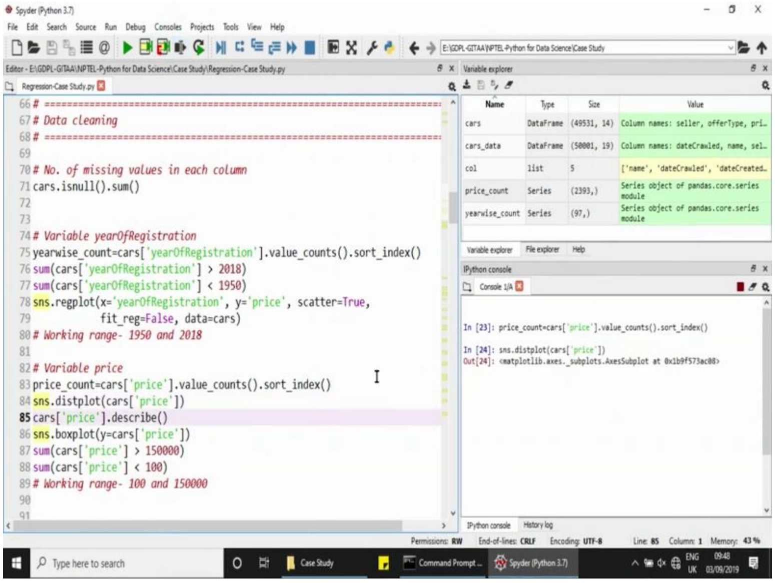


Index	price
15888	1
15890	2
15899	3
15900	61
15950	13
15980	2
15990	26
15995	2
15998	2
15999	40
16000	38
16100	2
16150	3
16199	2
16200	19
16250	4
16290	1
16299	5
16300	12
16350	4
16390	4

So, the left is basically the index and that is the value of the price of the car and the right is basically the frequencies of each occurrence. Now that is a too huge range. So, we have to do tweak it down and even price equal to 0 we will not make any sense, I mean it can be because if you want to just sell it for a few dollars or a few for a very low amount you can.

But then we are looking to generalize this model for a workable range of data and these values are these values can be very very extreme right. So, there is nothing wrong in selling a car for a lower price, but then we would like to generalize a model that would come up with the better prediction, but however, these if you have these values when they are going to smear the effect out. So, we are going to do check on this as well and we are going to arrive at a working range that is easy for us to understand right.

(Refer Slide Time: 16:29)



Editor - E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study\Regression-Case Study.py

```
# ==============================================================================
# Data cleaning
# ==============================================================================

# No. of missing values in each column
cars.isnull().sum()


# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018

# Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000
```

Variable explorer

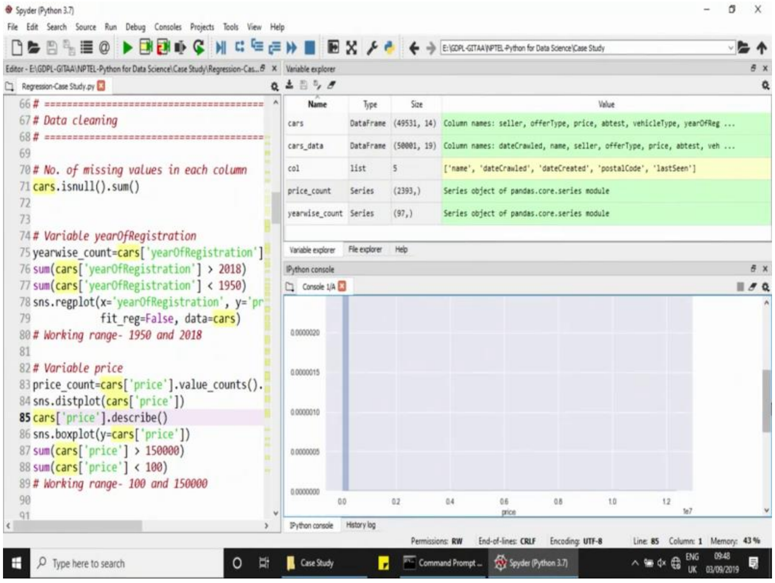
Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, pri…
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, sel…
col	list	5	['name', 'dateCrawled', 'dateCreated…
price_count	Series	(2393,)	Series object of pandas.core.series module
yearwise_count	Series	(97,)	Series object of pandas.core.series module

IPython console

```
In [23]: price_count=cars['price'].value_counts().sort_index()

In [24]: sns.distplot(cars['price'])
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1b9f573ac08>
```

So, now if you do histogram which is using the distplot right.

(Refer Slide Time: 16:31)



Editor - E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study\Regression-Cas...

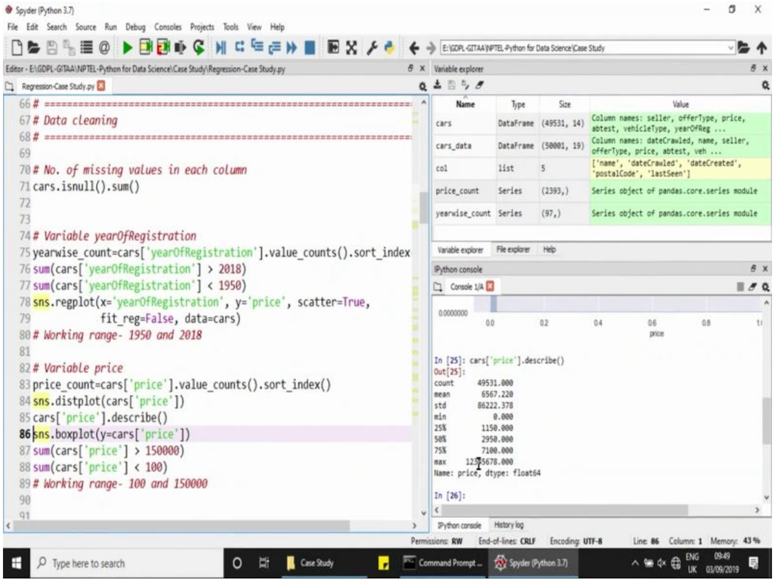
```
# ======================================
# Data cleaning
# ======================================

# No. of missing values in each column
cars.isnull().sum()


# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration']
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='pr
            fit_reg=False, data=cars)
# Working range- 1950 and 2018

# Variable price
price_count=cars['price'].value_counts().
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000
```

Variable explorer

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

You can see that because there are a lot of entries under the zero prize category. So, this is something that we have to work on.

(Refer Slide Time: 16:47)



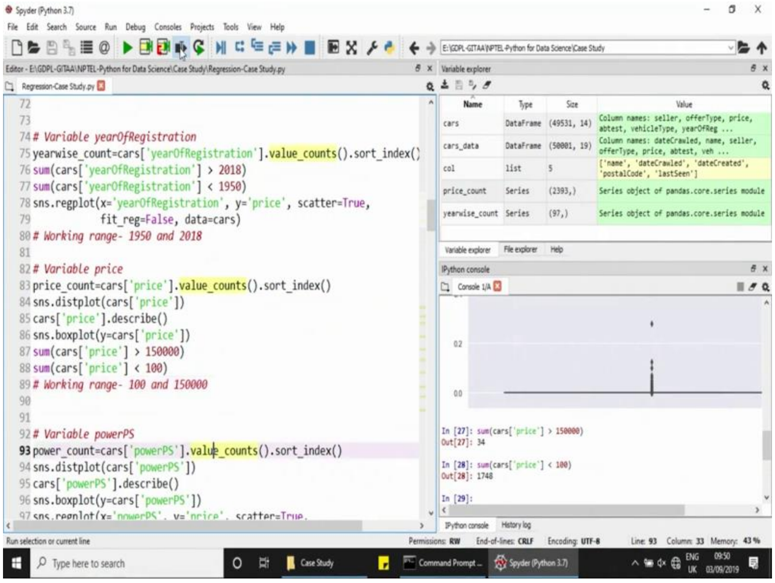
```
# ==========================================================================
# Data cleaning
# ==========================================================================

# No. of missing values in each column
cars.isnull().sum()


# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018

# Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000
```

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

```
In [25]: cars['price'].describe()
Out[25]:
count       49531.000
mean         6567.220
std         86222.378
min             0.000
25%          1150.000
50%          2950.000
75%          7100.000
max      12345678.000
Name: price, dtype: float64

In [26]:
```

So, let us just quickly reiterated by even looking at the describe. So, describe also tells you that the mean is about 6567, the median is way of the medium is around 2950. Now there is a huge different than this itself accounts for the skewness in the data and the minimum here you can see is again 0 and the maximum value is also given. So, the range is really really wide and we have to narrow it down to come up with the generalized model.

(Refer Slide Time: 17:15)



```
# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort_index()
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018

# Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000


# Variable powerPS
power_count=cars['powerPS'].value_counts().sort_index()
sns.distplot(cars['powerPS'])
cars['powerPS'].describe()
sns.boxplot(y=cars['powerPS'])
sns.regplot(x='powerPS', y='price', scatter=True,
```

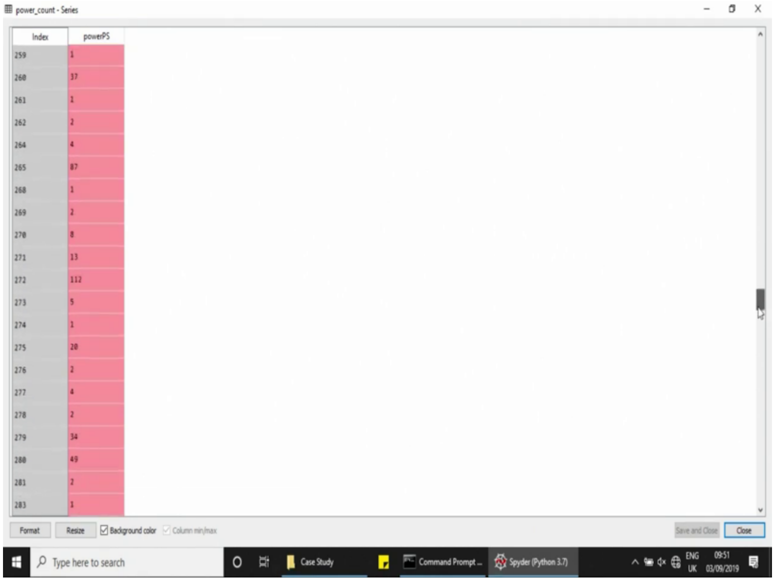
Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, price, abtest, vehicleType, yearOfReg ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
col	list	5	['name', 'dateCrawled', 'dateCreated', 'postalCode', 'lastSeen']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

```
In [27]: sum(cars['price'] > 150000)
Out[27]: 34

In [28]: sum(cars['price'] < 100)
Out[28]: 1748

In [29]:
```

So, let us just quickly do a box plot. Now a box plot also tells you that there are some outliers here right if I do a box plot for the price, you can see you cannot even see the box. In fact, what you just see is a line. Now this itself tells you that there are considerable outliers in your data which are very very extreme in nature and hence you not able to really see the behavior of the variable. So, let us just do a quick check of the range.

So, I am setting the range between 100 to lakh and 50,000 dollars. So, the first is to see how many cars are price above 1,50,000 dollars and that is about 34 and if you do a price check on the lower end you will see that 1748 cars a price lower than 100, but this is because 100 dollars to lakh and 50s decent range to work with. So, the next variable that we are going to look into is power ps. Now again if you do a.values count and then you sort it and now I have saved it under power count.

(Refer Slide Time: 18:27)



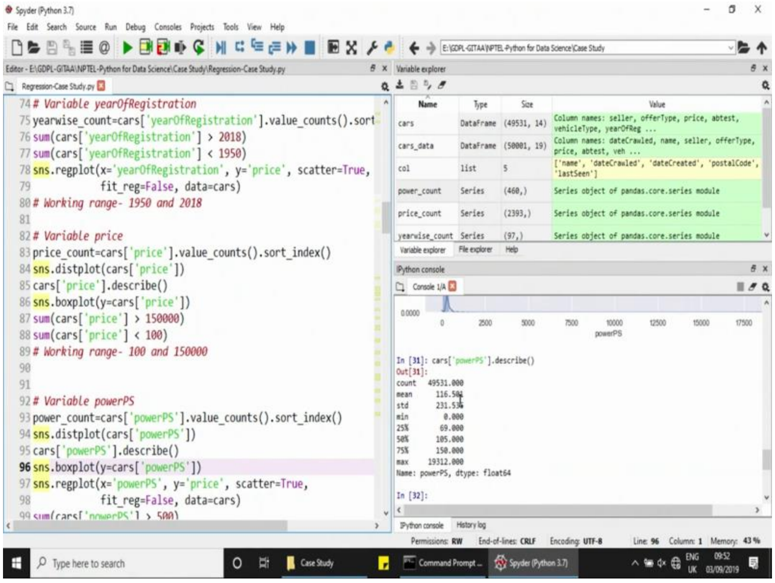
power_count - Series

Index	powerPS
259	1
260	37
261	1
262	2
264	4
265	87
268	1
269	2
270	8
271	13
272	112
273	5
274	1
275	20
276	2
277	4
278	2
279	34
280	49
281	2
283	1

So, here again you will see that the power values are on the extreme left and the right hand side is the frequencies of each. So, 5533 records have a power 0; again we have the same problem because the range is too diverse and we cannot really inform much. So, we have to again narrow it down. Somewhere in between you have a lot of occurrences of once some of them are two extreme for instance you have 19,312 horsepower that is very very extreme.

And which is occurred only once and so, we have to narrow down the range of power PS as well. So, let us just do a distplot just to see if there is any skewness yes there is you can see that from here itself because of entries that are of 0. So, this is something that we have to take into account we have to clean it up.

(Refer Slide Time: 19:23)



```
# Variable yearOfRegistration
yearwise_count=cars['yearOfRegistration'].value_counts().sort
sum(cars['yearOfRegistration'] > 2018)
sum(cars['yearOfRegistration'] < 1950)
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018


# Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000



# Variable powerPS
power_count=cars['powerPS'].value_counts().sort_index()
sns.distplot(cars['powerPS'])
cars['powerPS'].describe()
sns.boxplot(y=cars['powerPS'])
sns.regplot(x='powerPS', y='price', scatter=True,
            fit_reg=False, data=cars)
sum(cars['powerPS'] > 500)
```

Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, price, abtest, vehicleType, yearOfReg ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
col	list	5	['name', 'dateCrawled', 'dateCreated', 'postalCode', 'lastSeen']
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['powerPS'].describe()
Out[31]:
count    49531.000
mean       116.50
std        231.53
min          0.000
25%         69.000
50%        105.000
75%        150.000
max      19312.000
Name: powerPS, dtype: float64

In [32]:
```

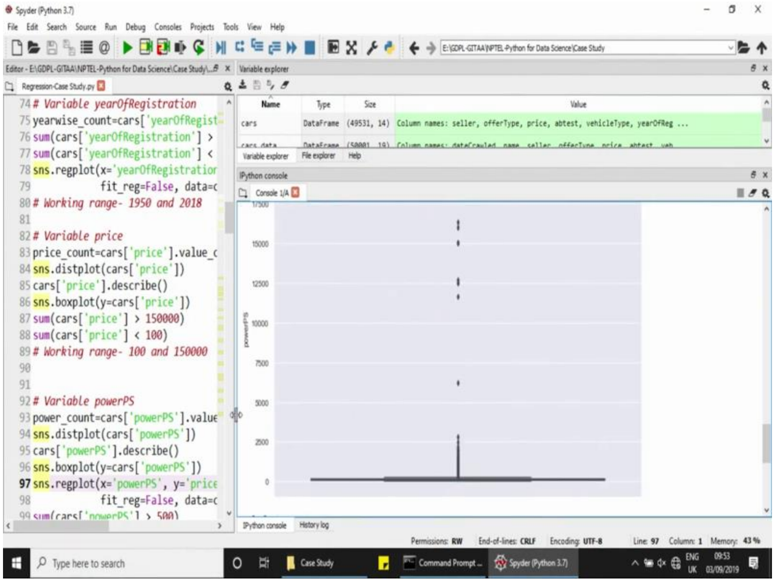
Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study\Regression-Case Study.py
Regression-Case Study.py
E:\GDPL-GITAA\NPTEL-Python for Data Science\Case Study
Variable explorer
Variable explorer File explorer Help
IPython console
Console 1/A
0.0000
0
2500
5000
7500
10000
12500
15000
17500
powerPS
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 96 Column: 1 Memory: 43 %
Type here to search
Case Study
Command Prompt ...
Spyder (Python 3.7)
ENG UK
09:52
03/09/2019

And if you do a.describe, this will also show you this skewness we have already done this before just reiterating it here. So, the mean is about 116 and if you look at the minimum it is 0 and if you look at the first quartile it is around 76 it is around 69 and if you look at the median, that is around 105. Though the mean and median are not far away the standard deviation is quite huge.

Now for a mean value of 116 if you are going to deviate by around 200 units then that is a lot right. So, though the mean and the median are not very far, but they are still far there is still some skewness. But if you look at the range of powerPS it is too diverse and you also have values that are around 0.

So, you just even if your mean and median are very very close you also have to take into account the standard deviation in the data and in this case it is very very huge right. So, we are going to take up this we are going to tweak the range down, but before we do that let us just quickly see if this is also getting reflected in the box plot.

(Refer Slide Time: 20:27)

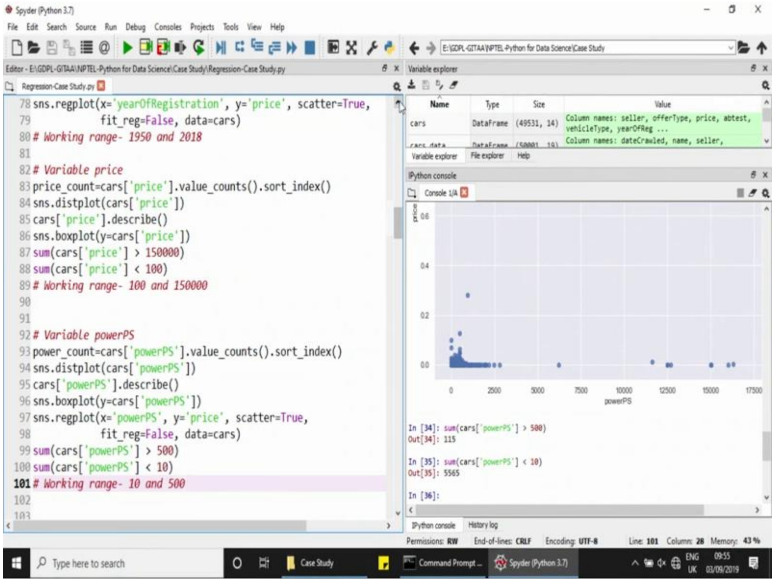


```
# Variable yearOfRegistration
yearwise_count=cars['yearOfRegist
sum(cars['yearOfRegistration'] >
sum(cars['yearOfRegistration'] <
sns.regplot(x='yearOfRegistration
            fit_reg=False, data=c
# Working range- 1950 and 2018


# Variable price
price_count=cars['price'].value_c
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000



# Variable powerPS
power_count=cars['powerPS'].value
sns.distplot(cars['powerPS'])
cars['powerPS'].describe()
sns.boxplot(y=cars['powerPS'])
sns.regplot(x='powerPS', y='price
            fit_reg=False, data=c
sum(cars['powerPS'] > 500)
```

Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, price, abtest, vehicleType, yearOfReg ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh

This plot is a lot better compared to the price plot that we are earlier did there we were just able to see a line at least here you are able to see a small box. So, yes of course, you have some extreme values here that are actually compressing the box. So, we will have to deal with this, we have to come with a workable range of data. So, let us just plot and see if powerPS has an effect on price before we clean it up.

(Refer Slide Time: 21:03)



```
sns.regplot(x='yearOfRegistration', y='price', scatter=True,
            fit_reg=False, data=cars)
# Working range- 1950 and 2018


# Variable price
price_count=cars['price'].value_counts().sort_index()
sns.distplot(cars['price'])
cars['price'].describe()
sns.boxplot(y=cars['price'])
sum(cars['price'] > 150000)
sum(cars['price'] < 100)
# Working range- 100 and 150000



# Variable powerPS
power_count=cars['powerPS'].value_counts().sort_index()
sns.distplot(cars['powerPS'])
cars['powerPS'].describe()
sns.boxplot(y=cars['powerPS'])
sns.regplot(x='powerPS', y='price', scatter=True,
            fit_reg=False, data=cars)
sum(cars['powerPS'] > 500)
sum(cars['powerPS'] < 10)
# Working range- 10 and 500
```

Name	Type	Size	Value
cars	DataFrame	(49531, 14)	Column names: seller, offerType, price, abtest, vehicleType, yearOfReg ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller,

```
In [34]: sum(cars['powerPS'] > 500)
Out[34]: 115

In [35]: sum(cars['powerPS'] < 10)
Out[35]: 5565

In [36]:
```

Again it is all bundled up together on the lower end and I think that is because of the values that are 0, we have to clean it up before we further do anything with this variable because otherwise we will not be able to see the effect of this variable on price. So, let us just fixe a range. So, I am fixing a range between 10 and 500, now I have arrived at all these ranges by trial and error also the idea is that you do not want to let go of too many records correct.

So, I am going to check the number of cars that have a powerPS of greater than 500 and that is about 115 and if you take the lower range and if you take the lower value the number of cars that are less than 10 it is about 5565. These are from trial and error and that is considerable reading that has been done to even check you know what is the minimum power that you need to start any vehicle.

So, the working range year of registration is between 19 to 50, the working range for year of registration is between 19 50 to 2018 and for price we are stick into 100 to 1, 50,000 dollars and the working range for powerPS variable is from 10 to 500. So, now, that we have checked the working range for three variables we are now going to clean the data by giving these variables and any further modifications we are going to do in the clean data.