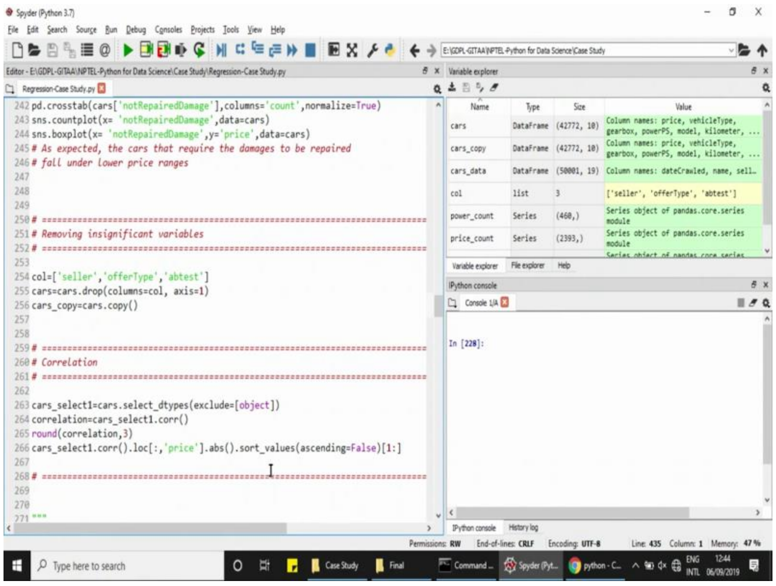
Python for Data Science
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 28
Case study on regression Part III

(Refer Slide Time: 00:15)



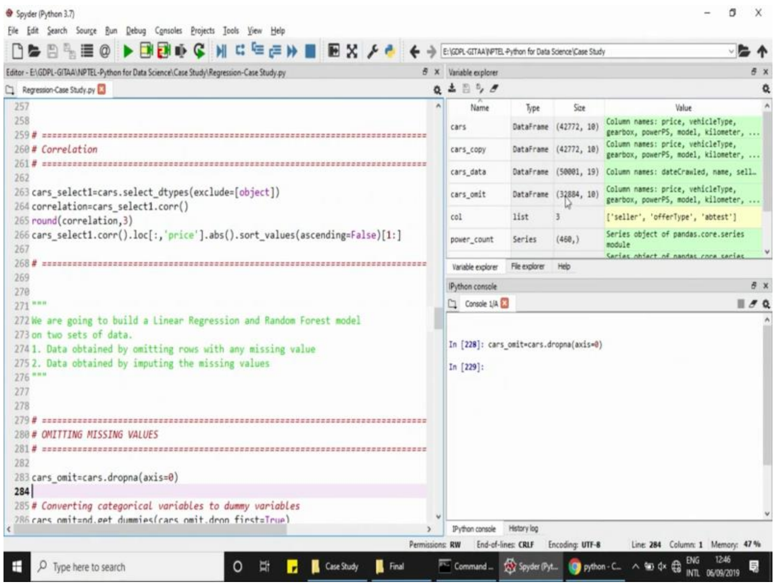
```
pd.crosstab(cars['notRepairedDamage'],columns='count',normalize=True)
sns.countplot(x= 'notRepairedDamage',data=cars)
sns.boxplot(x= 'notRepairedDamage',y='price',data=cars)
# As expected, the cars that require the damages to be repaired
# fall under lower price ranges



# =============================================================================
# Removing insignificant variables
# =============================================================================

col=['seller','offerType','abtest']
cars=cars.drop(columns=col, axis=1)
cars_copy=cars.copy()


# =============================================================================
# Correlation
# =============================================================================

cars_select1=cars.select_dtypes(exclude=[object])
correlation=cars_select1.corr()
round(correlation,3)
cars_select1.corr().loc[:,'price'].abs().sort_values(ascending=False)[1:]

# =============================================================================


"""
```

Name	Type	Size	Value
cars	DataFrame	(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
cars_copy	DataFrame	(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, sell…
col	list	3	['seller', 'offerType', 'abtest']
power_count	Series	(460,)	Series object of pandas.core.series module
price_count	Series	(2393,)	Series object of pandas.core.series module

```
In [228]:
```

So, now we have seen how to compute correlation. So, let us go ahead with model building.

(Refer Slide Time: 00:19)



```


# =============================================================================
# Correlation
# =============================================================================

cars_select1=cars.select_dtypes(exclude=[object])
correlation=cars_select1.corr()
round(correlation,3)
cars_select1.corr().loc[:,'price'].abs().sort_values(ascending=False)[1:]

# =============================================================================


"""
We are going to build a Linear Regression and Random Forest model
on two sets of data.
1. Data obtained by omitting rows with any missing value
2. Data obtained by imputing the missing values
"""


# =============================================================================
# OMITTING MISSING VALUES
# =============================================================================

cars_omit=cars.dropna(axis=0)

# Converting categorical variables to dummy variables
cars_omit=pd.get_dummies(cars_omit,drop_first=True)
```

Name	Type	Size	Value
cars	DataFrame	(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
cars_copy	DataFrame	(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, sell…
cars_omit	DataFrame	(32884, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
col	list	3	['seller', 'offerType', 'abtest']
power_count	Series	(460,)	Series object of pandas.core.series module

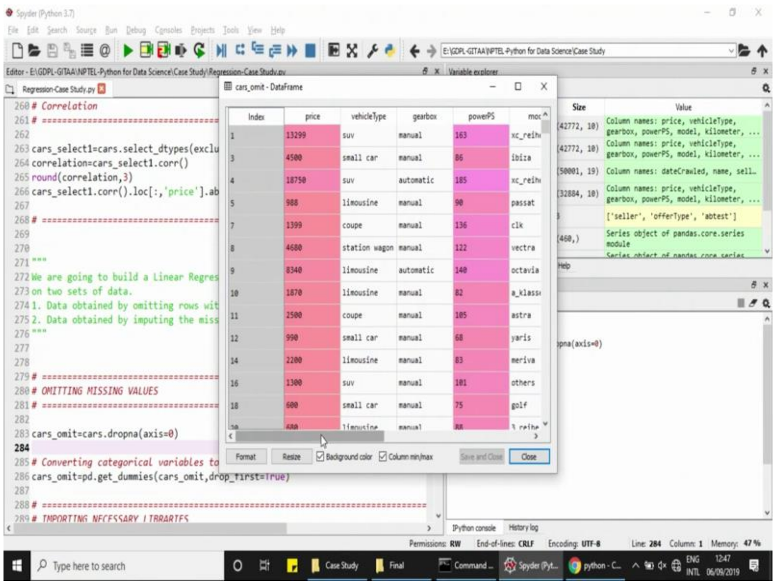
```
In [228]: cars_omit=cars.dropna(axis=0)

In [229]:
```

So, we are going to be looking into two algorithms; one is linear regression and other is the random forest regression and for each of these algorithms we are going to use two sets of data. Now, the first data is obtained by removing any missing value. So, any row with any number of missing value gets omitted. So, even if it is a single row with one missing value the entire row gets omitted. Now, the other data is where I impute the missing values. So, I am not going to let go of the missing cells. I am going to come up with some algorithm that is going to help me impute the missing value.

So, these are the two sets of data and for each of these set we are going to use a linear regression model and a random forest model. So, let us start with the first theta which is by omitting missing rows right. So, the command that you are going to use to drop any data with a missing value is dropna(). I am setting the axis as equal to 0. Because I want to drop any row that contains any number of missing value be it 1 or be it all cells missing right.

And once the command is executed so, the remaining data gets saved on to cars_omit. We started with 42772 records and after removing all rows which were missing it came down to 32884 and that is roughly about 10000 records that is been removed. So, once it is omitted cars_omit and you can see that around 10000 records have been removed.

(Refer Slide Time: 01:53)



```
# Correlation
# ====================================

cars_select1=cars.select_dtypes(exclu
correlation=cars_select1.corr()
round(correlation,3)
cars_select1.corr().loc[:,'price'].ab

# ====================================


"""
We are going to build a Linear Regres
on two sets of data.
1. Data obtained by omitting rows wit
2. Data obtained by imputing the miss
"""


# ====================================
# OMITTING MISSING VALUES
# ====================================

cars_omit=cars.dropna(axis=0)

# Converting categorical variables to
cars_omit=pd.get_dummies(cars_omit,drop_first=True)

# ====================================
# IMPORTING NECESSARY LIBRARIES
```

cars_omit - DataFrame

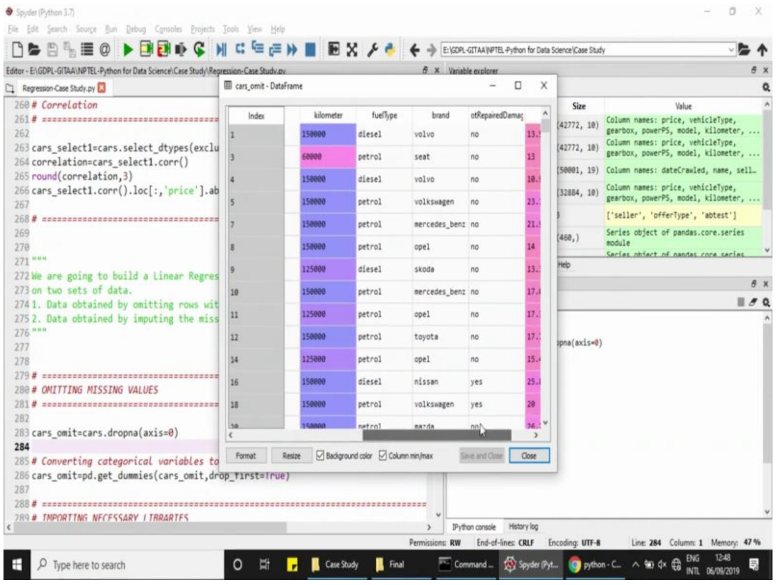
Index	price	vehicleType	gearbox	powerPS	moc
1	13299	suv	manual	163	xc_reihe
3	4500	small car	manual	86	ibiza
4	18750	suv	automatic	185	xc_reihe
5	988	limousine	manual	90	passat
7	1399	coupe	manual	136	clk
8	4680	station wagon	manual	122	vectra
9	8340	limousine	automatic	140	octavia
10	1870	limousine	manual	82	a_klasse
11	2500	coupe	manual	105	astra
12	990	small car	manual	68	yaris
14	2200	limousine	manual	83	meriva
16	1300	suv	manual	101	others
18	600	small car	manual	75	golf

Size	Value
(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
(50001, 19)	Column names: dateCrawled, name, sell…
(32884, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
3	['seller', 'offerType', 'abtest']
(460,)	Series object of pandas.core.series module

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
Format
Resize
Background color
Column min/max
Save and Close
Close
IPython console
History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 284 Column: 1 Memory: 47 %
Type here to search
Case Study
Final
Command ...
Spyder (Pyt...
python - C...
12:47 06/09/2019

So, we have columns that are categorical in nature for instance vehicle type, gearbox or model. Now, the values under these columns are strings in nature and none of the machine learning algorithms under scikit learn except columns which are strings by nature. So, I am going to dummy encode these categorical columns and I am going to generate newer columns out of it.

(Refer Slide Time: 02:15)



```
# Correlation
# ====================================

cars_select1=cars.select_dtypes(exclu
correlation=cars_select1.corr()
round(correlation,3)
cars_select1.corr().loc[:,'price'].ab

# ====================================


"""
We are going to build a Linear Regres
on two sets of data.
1. Data obtained by omitting rows wit
2. Data obtained by imputing the miss
"""


# ====================================
# OMITTING MISSING VALUES
# ====================================

cars_omit=cars.dropna(axis=0)

# Converting categorical variables to
cars_omit=pd.get_dummies(cars_omit,drop_first=True)

# =====================================================================
# IMPORTING NECESSARY LIBRARIES
```

cars_omit - DataFrame

Index	kilometer	fuelType	brand	otRepairedDamag	
1	150000	diesel	volvo	no	13.
3	60000	petrol	seat	no	13
4	150000	diesel	volvo	no	10.
5	150000	petrol	volkswagen	no	23.
7	150000	petrol	mercedes_benz	no	21.
8	150000	petrol	opel	no	14
9	125000	diesel	skoda	no	13.
10	150000	petrol	mercedes_benz	no	17.
11	125000	petrol	opel	no	17.
12	150000	petrol	toyota	no	17.
14	125000	petrol	opel	no	15.
16	150000	diesel	nissan	yes	25.
18	150000	petrol	volkswagen	yes	20
19	150000	petrol	mazda	no	26.

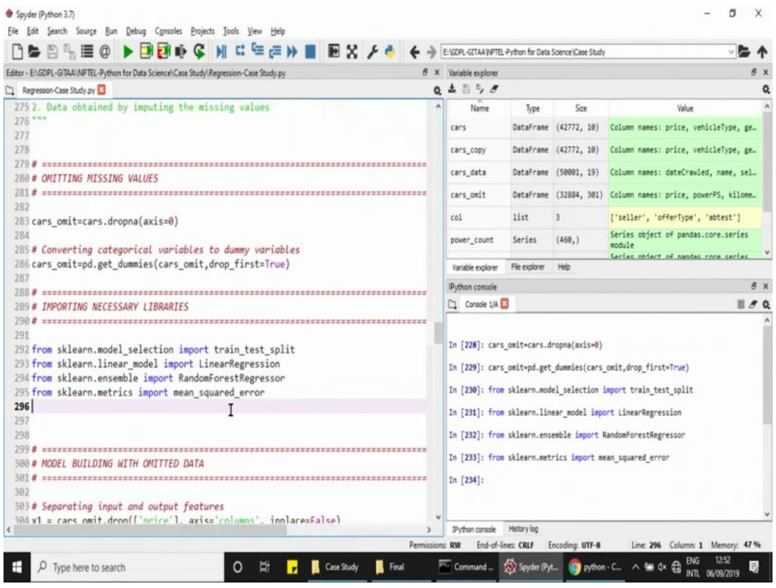
Size	Value
(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
(50001, 19)	Column names: dateCrawled, name, sell…
(32884, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer, ...
3	['seller', 'offerType', 'abtest']
(460,)	Series object of pandas.core.series module

For instance, let us say if your fuelType is diesel, petrol and you have CNG and LPG you have four categories here, then I am going to generate 4-1 which is 3 more columns out of this; one column will have fuelType_diesel, the other column will be fuelType_petrol and similarly, you have columns generated on it. Now, the value under each of these column will be 0 or 1; now, 0 if it is not present and 1 if the value is present.

For instance, let us take the column fuelType. Now, this column has several values. Let us say you have a column called fuelType_diesel and if diesel is present under this then that column will be then that cell will be marked as 1, otherwise it will be marked as 0. So, ideally we are trying to encode all of these categories as 0s or 1s alone and we are going to get rid of the strings. So, in order for us to do that I am going to use the get dummies function from the panda's library and I am going to drop the first category.

So, if I run it, you will see there are 301 columns in total. So, we started with 10 and then we have 301 more. And, the reason is because you have several categories under each column and especially variables like brand and model have several categories under them and hence you are getting more number of columns.

(Refer Slide Time: 03:47)



```
2. Data obtained by imputing the missing values
"""


# ==============================================================================
# OMITTING MISSING VALUES
# ==============================================================================

cars_omit=cars.dropna(axis=0)

# Converting categorical variables to dummy variables
cars_omit=pd.get_dummies(cars_omit,drop_first=True)

# ==============================================================================
# IMPORTING NECESSARY LIBRARIES
# ==============================================================================

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error



# ==============================================================================
# MODEL BUILDING WITH OMITTED DATA
# ==============================================================================

# Separating input and output features
x1 = cars_omit.drop(['price'], axis='columns', inplace=False)
```

Name	Type	Size	Value
cars	DataFrame	(42772, 10)	Column names: price, vehicleType, ge…
cars_copy	DataFrame	(42772, 10)	Column names: price, vehicleType, ge…
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, sel…
cars_omit	DataFrame	(32884, 301)	Column names: price, powerPS, kilome…
col	list	3	['seller', 'offerType', 'abtest']
power_count	Series	(460,)	Series object of pandas.core.series module

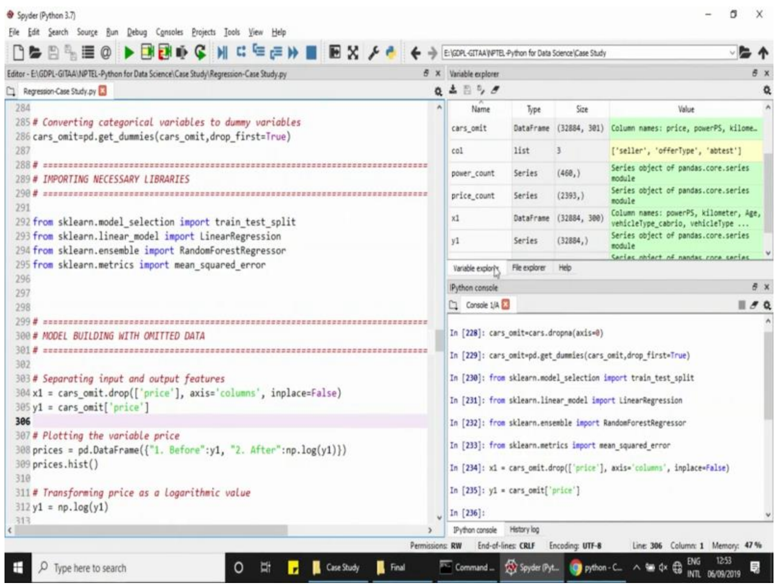
```
In [228]: cars_omit=cars.dropna(axis=0)

In [229]: cars_omit=pd.get_dummies(cars_omit,drop_first=True)

In [230]: from sklearn.model_selection import train_test_split

In [231]: from sklearn.linear_model import LinearRegression

In [232]: from sklearn.ensemble import RandomForestRegressor

In [233]: from sklearn.metrics import mean_squared_error

In [234]:
```

So, before we proceed let us just begin by importing some of the necessary libraries that we might need. Now, I am importing the train test split from sklearn.model_selection. Now, in now because we are going to go about building a model we will need a train set on which we are going to build a model and you also need a test set of data on which we are going to test it. So, given this data cars_omit, I am going to reserve a certain part of it for training and building my model I am going to reserve the remaining part to test my model on and for that you need the train_test_split function.

And I need the LinearRegression function from sklearn.linear_model and like how the name suggests this is for building a linear regression model. And similarly for a random forest model I want the RandomForestRegressor and this is a part of sklearn.ensemble package and apart from these two there are a few heuristics that I am going to calculate and one of it is mean squared error and that is a part of the sklearn.matrics package.

So, I am going to import these four libraries of packages before I proceed. So, let us begin by separating the input features and the output features.

(Refer Slide Time: 05:05)



```
# Converting categorical variables to dummy variables
cars_omit=pd.get_dummies(cars_omit,drop_first=True)

# ==============================================================================
# IMPORTING NECESSARY LIBRARIES
# ==============================================================================

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error



# ==============================================================================
# MODEL BUILDING WITH OMITTED DATA
# ==============================================================================

# Separating input and output features
x1 = cars_omit.drop(['price'], axis='columns', inplace=False)
y1 = cars_omit['price']

# Plotting the variable price
prices = pd.DataFrame({"1. Before":y1, "2. After":np.log(y1)})
prices.hist()

# Transforming price as a logarithmic value
y1 = np.log(y1)
```

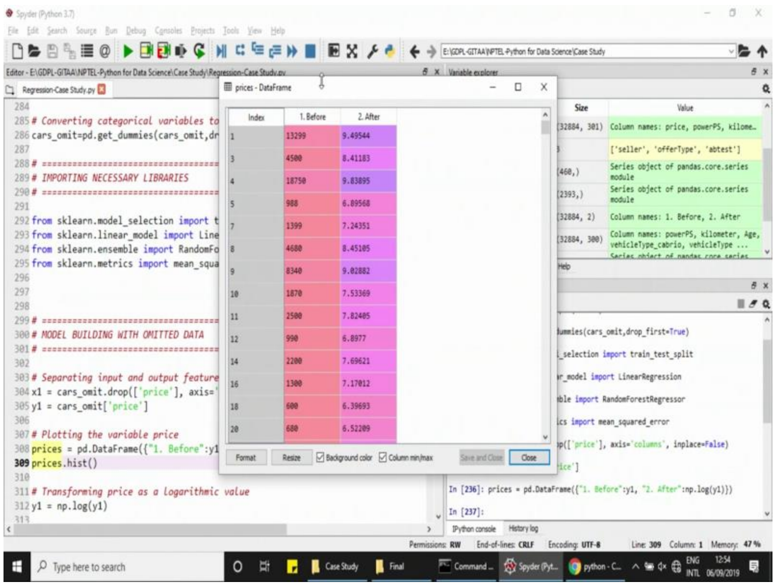
Name	Type	Size	Value
cars_omit	DataFrame	(32884, 301)	Column names: price, powerPS, kilome…
col	list	3	['seller', 'offerType', 'abtest']
power_count	Series	(460,)	Series object of pandas.core.series module
price_count	Series	(2393,)	Series object of pandas.core.series module
x1	DataFrame	(32884, 300)	Column names: powerPS, kilometer, Age, vehicleType_cabrio, vehicleType ...
y1	Series	(32884,)	Series object of pandas.core.series module

```
In [228]: cars_omit=cars.dropna(axis=0)

In [229]: cars_omit=pd.get_dummies(cars_omit,drop_first=True)

In [230]: from sklearn.model_selection import train_test_split

In [231]: from sklearn.linear_model import LinearRegression

In [232]: from sklearn.ensemble import RandomForestRegressor

In [233]: from sklearn.metrics import mean_squared_error

In [234]: x1 = cars_omit.drop(['price'], axis='columns', inplace=False)

In [235]: y1 = cars_omit['price']

In [236]:
```

Now, my input features I am going to call them as x1 and my output feature which is price I am going to call it as y1. So, from cars.omit I am going to drop only price and I am dropping column. So, I am just mentioning it as columns and I am just saying in place equal to false I do not want the changes to be reflected under cars_omit right.

So, let us just drop only price. So, basically x1 has everything apart from price. So, we so, cars_omit had 301 columns and one column alone we have removed which is price and hence you have 300 columns under x1 contrary to that we want y1 to contain only one column which is price. So, I am just saving cars_omit and within square indices I am just giving price which makes it y as only column.

(Refer Slide Time: 06:01)



```
# Converting categorical variables to
cars_omit=pd.get_dummies(cars_omit,dr

# ==================================
# IMPORTING NECESSARY LIBRARIES
# ==================================

from sklearn.model_selection import t
from sklearn.linear_model import Line
from sklearn.ensemble import RandomFo
from sklearn.metrics import mean_squa



# ==================================
# MODEL BUILDING WITH OMITTED DATA
# ==================================

# Separating input and output feature
x1 = cars_omit.drop(['price'], axis='
y1 = cars_omit['price']

# Plotting the variable price
prices = pd.DataFrame({"1. Before":y1
prices.hist()

# Transforming price as a logarithmic value
y1 = np.log(y1)
```

prices - DataFrame

Index	1. Before	2. After
1	13299	9.49544
3	4500	8.41183
4	18750	9.83895
5	988	6.89568
7	1399	7.24351
8	4680	8.45105
9	8340	9.02882
10	1870	7.53369
11	2500	7.82405
12	990	6.8977
14	2200	7.69621
16	1300	7.17012
18	600	6.39693
20	680	6.52209

Size	Value
(32884, 301)	Column names: price, powerPS, kilome…
	['seller', 'offerType', 'abtest']
(460,)	Series object of pandas.core.series module
(2393,)	Series object of pandas.core.series module
(32884, 2)	Column names: 1. Before, 2. After
(32884, 300)	Column names: powerPS, kilometer, Age, vehicleType_cabrio, vehicleType ...

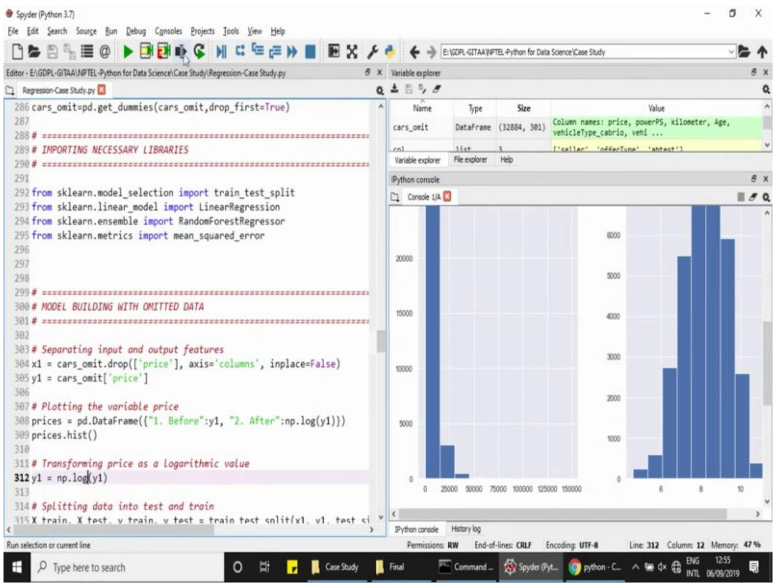
```
ummies(cars_omit,drop_first=True)
_selection import train_test_split
r_model import LinearRegression
ble import RandomForestRegressor
cs import mean_squared_error
p(['price'], axis='columns', inplace=False)
ice']
In [236]: prices = pd.DataFrame({"1. Before":y1, "2. After":np.log(y1)})

In [237]:
```

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
Variable explorer
Format
Resize
Background color
Column min/max
Save and Close
Close
Help
IPython console
History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 309 Column: 1 Memory: 47 %
Type here to search
12:54 06/09/2019

And, you have the same number of observations for price. Now, this is. So, x1 is a collection of all input features and y1 is my output feature which is price. Now, I am just going to plot the variable price now before I plot it I am just converting it to a data frame which says before and after. So, y1 and then I have transformed y1 as a log as a natural logarithm.

So, let me just run this command. So, basically you will have two column under prices one is before, the other is after. Before is what you have as y1 and after is after taking the natural logarithm. Now, the reason we are doing it, it will be clear when I plug the histogram which is below.

(Refer Slide Time: 06:53)

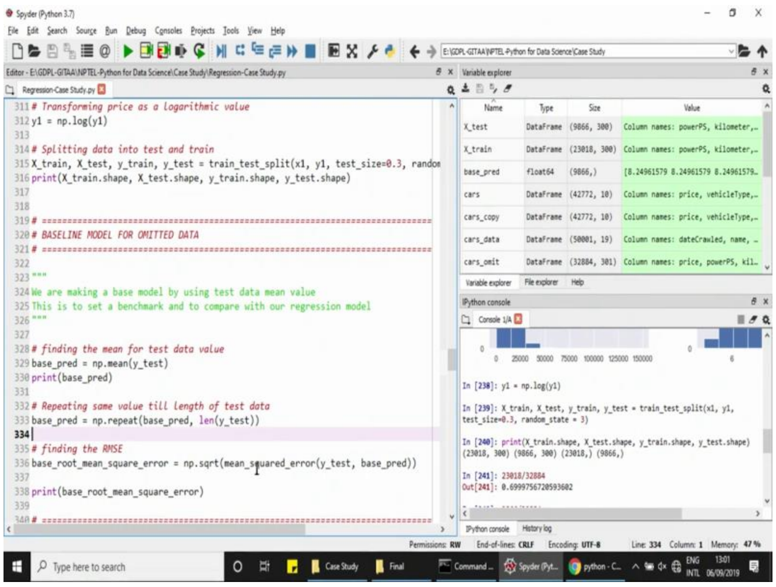


```
cars_omit=pd.get_dummies(cars_omit,drop_first=True)

# =====================================================================
# IMPORTING NECESSARY LIBRARIES
# =====================================================================

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error



# =====================================================================
# MODEL BUILDING WITH OMITTED DATA
# =====================================================================

# Separating input and output features
x1 = cars_omit.drop(['price'], axis='columns', inplace=False)
y1 = cars_omit['price']

# Plotting the variable price
prices = pd.DataFrame({"1. Before":y1, "2. After":np.log(y1)})
prices.hist()

# Transforming price as a logarithmic value
y1 = np.log(y1)

# Splitting data into test and train
X_train, X_test, y_train, y_test = train_test_split(x1, y1, test_si
```

Let me just drag the screen to show you the difference between plotting histogram with pi and plotting histogram with logarithmic of π. So, here you can see there is a nice bell shaped curve the moment you plot the x-axis that is natural logarithm of y whereas, on my left I have just done a histogram on just the price whereas, on the right I have a logarithm of price. Now, here you can see that the shape the right hand side is more bell shaped normal compared to the left; the left is most cute and this itself tells you that you know we should consider going further with our natural log on the prices and not just with price.

So, we are not going to regress price with the input features, rather we are going to regress the log of price with input features and this log is the natural log. So, let us now transform the price as a logarithmic value. So, I do np.log(y1). So, my entire y1 is now transformed right. This is because the range of price is very very huge because you want to avoid it you are taking the log.

(Refer Slide Time: 08:19)



```python
# Transforming price as a logarithmic value
y1 = np.log(y1)

# Splitting data into test and train
X_train, X_test, y_train, y_test = train_test_split(x1, y1, test_size=0.3, random
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)


# =============================================================================
# BASELINE MODEL FOR OMITTED DATA
# =============================================================================

"""
We are making a base model by using test data mean value
This is to set a benchmark and to compare with our regression model
"""

# finding the mean for test data value
base_pred = np.mean(y_test)
print(base_pred)

# Repeating same value till length of test data
base_pred = np.repeat(base_pred, len(y_test))

# finding the RMSE
base_root_mean_square_error = np.sqrt(mean_squared_error(y_test, base_pred))

print(base_root_mean_square_error)

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

Name	Type	Size	Value
X_test	DataFrame	(9866, 300)	Column names: powerPS, kilometer,…
X_train	DataFrame	(23018, 300)	Column names: powerPS, kilometer,…
base_pred	float64	(9866,)	[8.24961579 8.24961579 8.24961579…
cars	DataFrame	(42772, 10)	Column names: price, vehicleType,…
cars_copy	DataFrame	(42772, 10)	Column names: price, vehicleType,…
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, …
cars_omit	DataFrame	(32884, 301)	Column names: price, powerPS, kil…

```
In [238]: y1 = np.log(y1)

In [239]: X_train, X_test, y_train, y_test = train_test_split(x1, y1,
test_size=0.3, random_state = 3)

In [240]: print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
(23018, 300) (9866, 300) (23018,) (9866,)

In [241]: 23018/32884
Out[241]: 0.6999756720593602
```

Now, I am splitting my data into testing chain, I am just dragging the output window to the right just to show you the entire command. So, the train underscores test_split is the function which we have already imported. The input to it is the input set of features, the output set of features which is x1 and y1 respectively. Now, the test side I am giving it as 0.3 and which means 70 percent of the data will go to the train set and the remaining 30 percent will be retained as the test set and I am setting a random state of equal to 3.

Now, random state is a predefined algorithm it is called pseudo random number generator; you can give any value to this parameter and I am giving a value of 3. So, every time you run the same algorithm the same set of records will go to train and test. So, the output I am saving it as X_train X_test, y_train and y_test. So, the X's represent the input set of features and the y's that represent the output feature which is price. So, let me just run this command and once I run the command I am just printing the shape of each of these. So, my X_train is 23,018 and I can have 300 columns my y_train is 23018 again.

Similarly, my X_test is 9866 and I can have 300 columns and my y_test is 9866. So, this is the shape of the data if you do 23818 divided by 32884 that is roughly about 70 percent right. Similarly, if you do 9866 divided by 32884 that is a free about 30 percent right. You can infer the test train split ratio from here itself based on the number of observations.

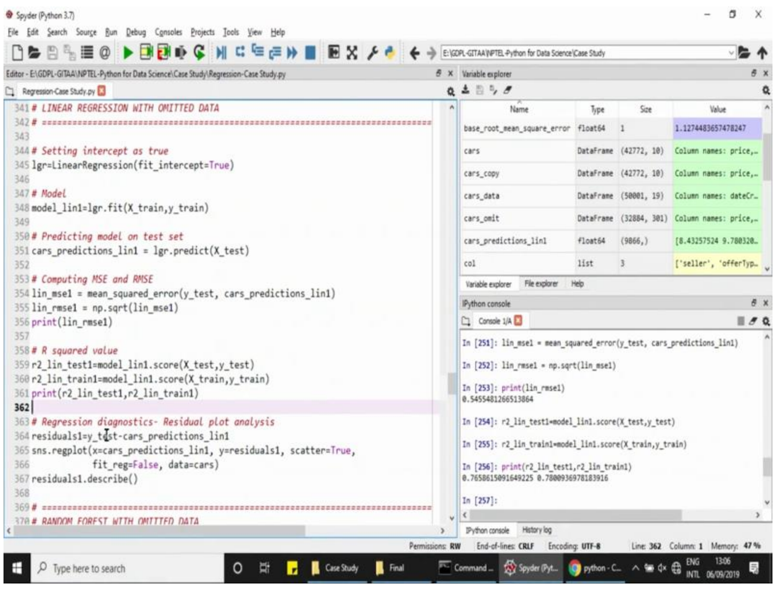
So, before we move further we want to build a baseline model for this data and in the baseline model the predicted value is basically replaced with the mean value of the test data and this is to actually set the benchmark for us and to compare our models that we are going to build in future. So, for this data we are going to build a regression and a random forest model and in order to compare the metrics from that model we also need a base value or a base metric. And, the base metric that we are going to use is the RMSE value for the base model.

So, the base model is where you replace the predicted value as the mean from the test value and that I am saving it as base_pred right and that comes out to be around 8.25 and I am going to. So, this is only one value and I wanted repeated to 9866 values which is the length of y_test. So, I am just replicating this value over 9866 records.

Now, I want to find the RMSE value. Now, RMSE is root means squared error and it is a like the name suggests is the square root of the mean squared error. So, it computes a between the test value and the predicted value squares them and divides them by the number of sample. Now, that is mean square error; the moment you take a root of that value you get the RMSE value which is the root mean square error. So, you already have a function called mean_square_error which we have imported earlier.

Now, the input to it is the test data right the y_test and the base predictions right. So, these are the two value that I am going to give and once I computed I am just going to print it. So, the base RMSE turns out to be 1.13 roughly. So, this is a benchmark for comparison. Any other models that are going to build in future should give you an RMSE that is less than this. So, that is the objective for us.

(Refer Slide Time: 12:11)



```
# LINEAR REGRESSION WITH OMITTED DATA
# ===============================================================================

# Setting intercept as true
lgr=LinearRegression(fit_intercept=True)

# Model
model_lin1=lgr.fit(X_train,y_train)

# Predicting model on test set
cars_predictions_lin1 = lgr.predict(X_test)

# Computing MSE and RMSE
lin_mse1 = mean_squared_error(y_test, cars_predictions_lin1)
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)

# R squared value
r2_lin_test1=model_lin1.score(X_test,y_test)
r2_lin_train1=model_lin1.score(X_train,y_train)
print(r2_lin_test1,r2_lin_train1)

# Regression diagnostics- Residual plot analysis
residuals1=y_test-cars_predictions_lin1
sns.regplot(x=cars_predictions_lin1, y=residuals1, scatter=True,
            fit_reg=False, data=cars)
residuals1.describe()

# ===============================================================================
# RANDOM FOREST WITH OMITTED DATA
```

Name	Type	Size	Value
base_root_mean_square_error	float64	1	1.1274483657478247
cars	DataFrame	(42772, 10)	Column names: price,…
cars_copy	DataFrame	(42772, 10)	Column names: price,…
cars_data	DataFrame	(50001, 19)	Column names: dateCr…
cars_omit	DataFrame	(32884, 301)	Column names: price,…
cars_predictions_lin1	float64	(9866,)	[8.43257524 9.780320…
col	list	3	['seller', 'offerTyp…

```
In [251]: lin_mse1 = mean_squared_error(y_test, cars_predictions_lin1)

In [252]: lin_rmse1 = np.sqrt(lin_mse1)

In [253]: print(lin_rmse1)
0.5455481266513864

In [254]: r2_lin_test1=model_lin1.score(X_test,y_test)

In [255]: r2_lin_train1=model_lin1.score(X_train,y_train)

In [256]: print(r2_lin_test1,r2_lin_train1)
0.7658615091649225 0.7800936978183916

In [257]:
```

So, now let us start with linear regression for this data. Now, I am going to call this the omitted data because I have removed all missing values from here. So, the function that you are going to invoke first is called the linear regression function. I am setting the intercept as true because I want an intercept for this and I am saving it on to an object called lgr. Once you do that you need to fit the model on the train X and train y; train X being the input features, train y being the output features for the train set of data. And, I am storing the model as model_linear_1; linear has represented by lin.

And, once I do that I want to predict my model on the test set of input features. So, my_test set of input features are the X_test and I am going to use the.predict function and I am saving my predictions on to cars_predictions_linear 1; linear again is represented by lin1. So, here you can see you have 9866 predictions right. Now, these are my predictions. Now, I am again going to compute my mean squared error.

Now, instead of giving a base predicted value, I am going to pass on the actually predicted values here and again the other parameter that I am passing is the test set of features that I have from my_test set. Now, again I need RMSE value. So, I am just computing the RMSE by taking a square root of this and if you print it you will see that my RMSE for this model which is a linear regression model is 0.54.

So, we started at a value of 1.12 with the base RMSE and we came down to about 0.54 that is roughly about a 50 percent drop right there is a 50 percent change from this value.

So, we have come down a long way from there we have started. So, we are at 0.54. Now, let us see if we can compute the R squared value. So, the R squared value tells you, so how good is your model able to explain the variability in the y. Now, let us do this for test and train.

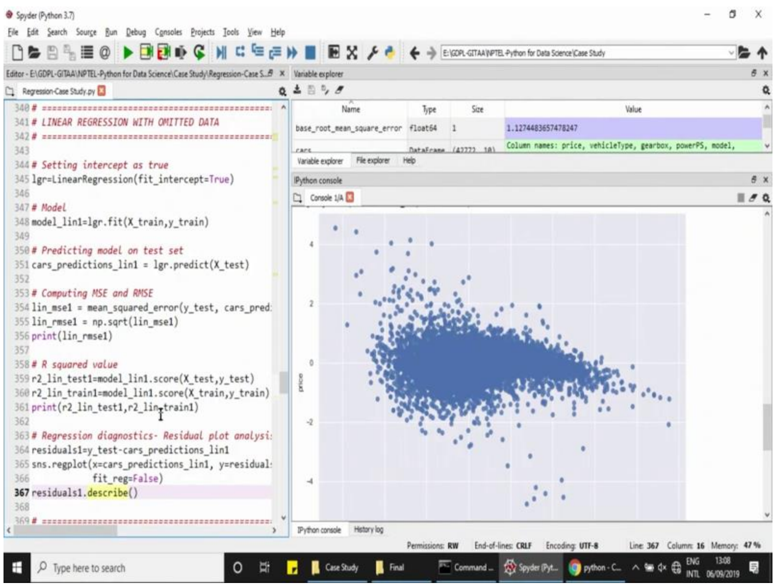
Now, I am going to call the function.score from the model_lin1 and the input to it is X_test and x y_test and similarly, if you want to r squared value for your train you just give the train input. Now, I am just calculating the value of for test and for train and let us just print it out.

So, if you print out both these values at test side for the test set my R square is 0.766 and for my_train set the R square is 0.780; cutting quite close, but the train seems to be definitely better and the test is not far behind, it is about 0.766. So, that is a close cut. So, this tells you that the model is good. As much as the variability that the model was able to capture in the train data, it is able to capture the same amount of variability if not more on the test data itself, on the test data as well.

Now, let us just do a quick regression diagnostics. Now, I am going to calculate something called residuals here. Now, the residual is nothing, but the difference between your test data and your prediction. Now, the residual is nothing, but the difference between your predicted value and your actual value. Now, your actual values under y_test and your predicted value is under is under cars_predictions_linear1.

Now, once you do this you can plot I am once you do this I am going to generate a residual plot using a simple scatter plot where my x is the predicted the fitted value, that is, cars_predictions_linear1 and my y is nothing, but the residuals.

(Refer Slide Time: 16:25)



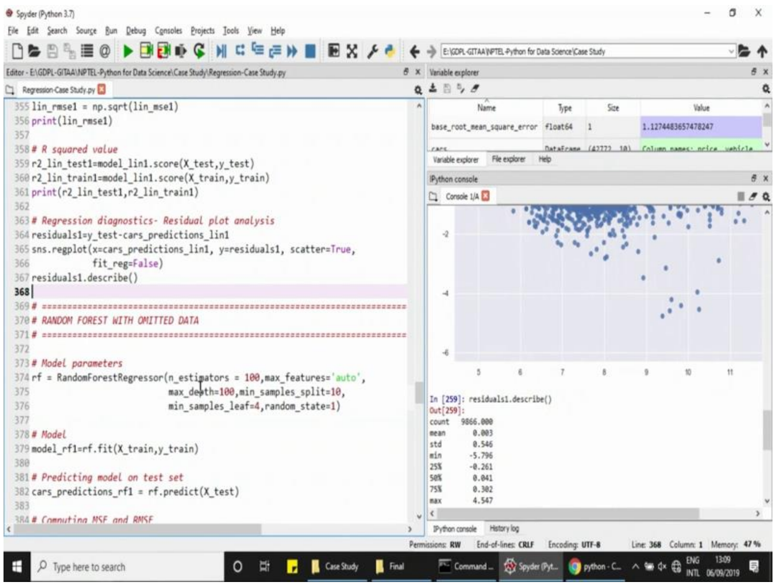
```
# ==============================================
# LINEAR REGRESSION WITH OMITTED DATA
# ==============================================

# Setting intercept as true
lgr=LinearRegression(fit_intercept=True)

# Model
model_lin1=lgr.fit(X_train,y_train)

# Predicting model on test set
cars_predictions_lin1 = lgr.predict(X_test)

# Computing MSE and RMSE
lin_mse1 = mean_squared_error(y_test, cars_pred
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)

# R squared value
r2_lin_test1=model_lin1.score(X_test,y_test)
r2_lin_train1=model_lin1.score(X_train,y_train)
print(r2_lin_test1,r2_lin_train1)

# Regression diagnostics- Residual plot analysi
residuals1=y_test-cars_predictions_lin1
sns.regplot(x=cars_predictions_lin1, y=residual
            fit_reg=False)
residuals1.describe()

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

Name	Type	Size	Value
base_root_mean_square_error	float64	1	1.1274483657478247
cars	DataFrame	(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model,

So, let us just quickly do a regression diagnostic. I am going to use the residual plot analysis. Now, I am going to generate a separate set of numbers or values that is called residuals1. Now, it is the difference between the y_test and the cars_prediction_linear1. Now, basically residuals are nothing, but the difference of your actual value or the observed value and the predicted value. So, that is called residuals and what I am going to do is I am going to generate a simple scatter plot where my x-axis is basically the fitted or the predicted values and my y-axis contains the residual values.

So, let us just do a quick scatter plot here. So, I am going to drag the output window little to show you how does the plot look. This is good because you want the residuals to be close to 0. Since, residuals are more or more of errors you want the errors to be less and that tells you that your predicted and your actual values are very very closer and this can also be seen from the describe function. So, if you do a residuals1.described this gives you the summary of the data.

(Refer Slide Time: 17:37)



```python
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)

# R squared value
r2_lin_test1=model_lin1.score(X_test,y_test)
r2_lin_train1=model_lin1.score(X_train,y_train)
print(r2_lin_test1,r2_lin_train1)

# Regression diagnostics- Residual plot analysis
residuals1=y_test-cars_predictions_lin1
sns.regplot(x=cars_predictions_lin1, y=residuals1, scatter=True,
            fit_reg=False)
residuals1.describe()

# =====================================================================
# RANDOM FOREST WITH OMITTED DATA
# =====================================================================

# Model parameters
rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
                           max_depth=100,min_samples_split=10,
                           min_samples_leaf=4,random_state=1)

# Model
model_rf1=rf.fit(X_train,y_train)

# Predicting model on test set
cars_predictions_rf1 = rf.predict(X_test)

# Computing MSE and RMSE
```

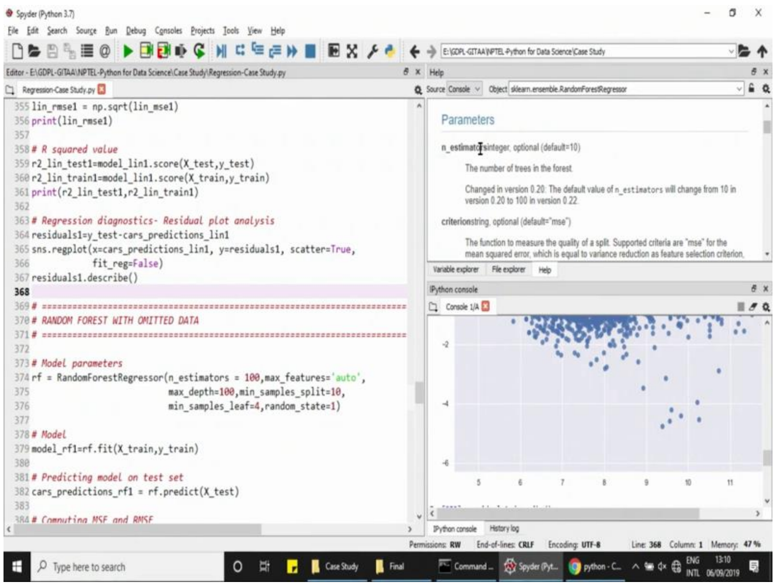
Name	Type	Size	Value
base_root_mean_square_error	float64	1	1.1274483657478247
cars	DataFrame	(42772, 10)	Column names: price, vehicle

```
In [259]: residuals1.describe()
Out[259]:
count    9866.000
mean        0.003
std         0.546
min        -5.796
25%        -0.261
50%         0.041
75%         0.302
max         4.547
```

And you can see that the mean of the data is around 0.003 that is very very less, it is almost 0 right and this itself a good this itself is a good indicator that your predictor and your actual value is a very very closer now. And this is the conclusion that it can actually draw from your residual analysis. So, I have just shown you for one linear regression model and you can just do it with the others as well.

So, let us move ahead with random forest model. Now, a random forest model has several inputs and we have already imported the function called randomForestRegressor. So, I have a couple of parameters here as input arguments. Let us just quickly go over what these mean.

(Refer Slide Time: 18:23)



```
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)

# R squared value
r2_lin_test1=model_lin1.score(X_test,y_test)
r2_lin_train1=model_lin1.score(X_train,y_train)
print(r2_lin_test1,r2_lin_train1)

# Regression diagnostics- Residual plot analysis
residuals1=y_test-cars_predictions_lin1
sns.regplot(x=cars_predictions_lin1, y=residuals1, scatter=True,
            fit_reg=False)
residuals1.describe()

# ======================================================================
# RANDOM FOREST WITH OMITTED DATA
# ======================================================================

# Model parameters
rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
                           max_depth=100,min_samples_split=10,
                           min_samples_leaf=4,random_state=1)

# Model
model_rf1=rf.fit(X_train,y_train)

# Predicting model on test set
cars_predictions_rf1 = rf.predict(X_test)

# Computing MSE and RMSE
```

Object: sklearn.ensemble.RandomForestRegressor

Parameters

n_estimatorsinteger, optional (default=10)

The number of trees in the forest

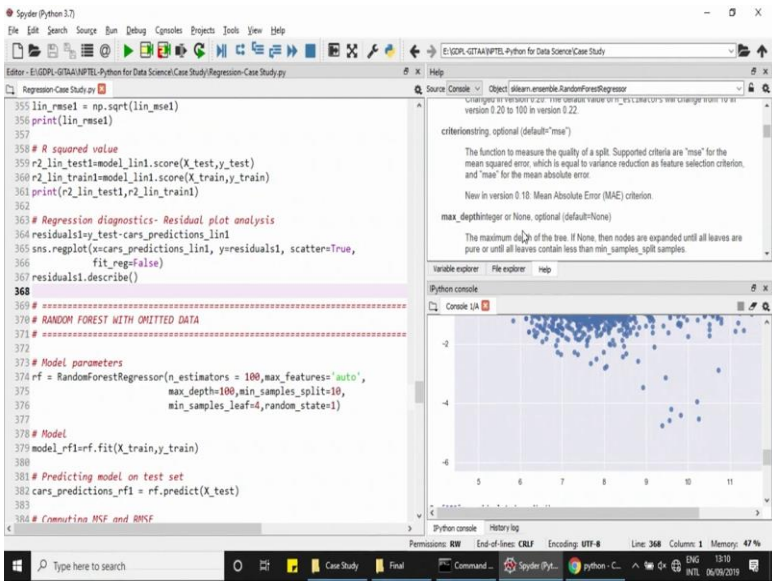
Changed in version 0.20: The default value of `n_estimators` will change from 10 in version 0.20 to 100 in version 0.22.

criterionstring, optional (default="mse")

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion.

So, I in my help toolbox I have typed a sklearn.ensemble.RandomForestRegressor and it gives you a quick description of what does the function do. But, if you come down it also tells you what are the input parameters and what do they mean. Also mentioned under the parameters are also the default values. So, let us begin with the first parameter it is n_estimators. Now, because this is a random forest tree you can have several trees and that is what n estimators does. It basically tells you the number of trees in your forest.

(Refer Slide Time: 19:01)



```
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)

# R squared value
r2_lin_test1=model_lin1.score(X_test,y_test)
r2_lin_train1=model_lin1.score(X_train,y_train)
print(r2_lin_test1,r2_lin_train1)

# Regression diagnostics- Residual plot analysis
residuals1=y_test-cars_predictions_lin1
sns.regplot(x=cars_predictions_lin1, y=residuals1, scatter=True,
            fit_reg=False)
residuals1.describe()

# ======================================================================
# RANDOM FOREST WITH OMITTED DATA
# ======================================================================

# Model parameters
rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
                           max_depth=100,min_samples_split=10,
                           min_samples_leaf=4,random_state=1)

# Model
model_rf1=rf.fit(X_train,y_train)

# Predicting model on test set
cars_predictions_rf1 = rf.predict(X_test)

# Computing MSE and RMSE
```

Object: sklearn.ensemble.RandomForestRegressor

version 0.20 to 100 in version 0.22.

criterionstring, optional (default="mse")

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "mae" for the mean absolute error.

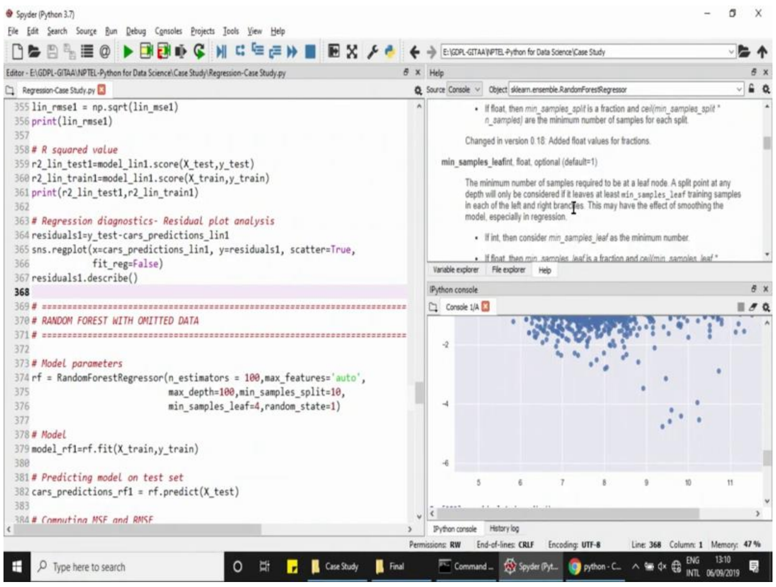
New in version 0.18: Mean Absolute Error (MAE) criterion.

max_depthinteger or None, optional (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

The next is maximum depth which is the depth of each tree; how deep do you want each tree to go and that is given by maximum depth.

(Refer Slide Time: 19:11)



```
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)

# R squared value
r2_lin_test1=model_lin1.score(X_test,y_test)
r2_lin_train1=model_lin1.score(X_train,y_train)
print(r2_lin_test1,r2_lin_train1)

# Regression diagnostics- Residual plot analysis
residuals1=y_test-cars_predictions_lin1
sns.regplot(x=cars_predictions_lin1, y=residuals1, scatter=True,
            fit_reg=False)
residuals1.describe()

# =============================================================================
# RANDOM FOREST WITH OMITTED DATA
# =============================================================================

# Model parameters
rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
                           max_depth=100,min_samples_split=10,
                           min_samples_leaf=4,random_state=1)

# Model
model_rf1=rf.fit(X_train,y_train)

# Predicting model on test set
cars_predictions_rf1 = rf.predict(X_test)

# Computing MSE and RMSE
```

- If float, then *min_samples_split* is a fraction and *ceil(min_samples_split * n_samples)* are the minimum number of samples for each split.

Changed in version 0.18: Added float values for fractions.

min_samples_leafint, float, optional (default=1)

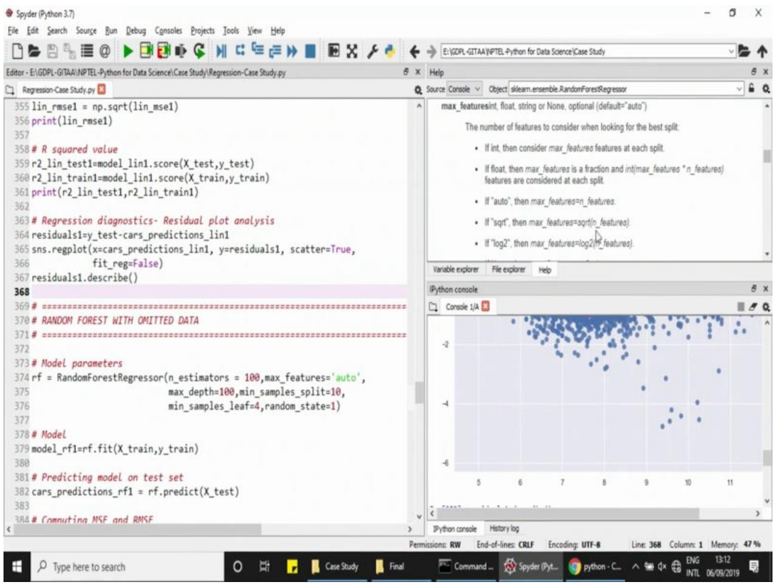
The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- If int, then consider *min_samples_leaf* as the minimum number.

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
Help
Source Console
Object sklearn.ensemble.RandomForestRegressor
Variable explorer
File explorer
Help
IPython console
Console 1/A
History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 368 Column: 1 Memory: 47 %
Type here to search

The minimum_samples_split is the minimum number of samples that is required for a node to split. So, let us say if you have only two observations in a node and you do not want that node to split any further right because it might not make any sense. So, in that case if you want to impose such conditions you can do it here. The next is minimum_samples_leaf and it is the minimum number of samples which is required to be at a leaf node.

(Refer Slide Time: 19:41)



```
lin_rmse1 = np.sqrt(lin_mse1)
print(lin_rmse1)

# R squared value
r2_lin_test1=model_lin1.score(X_test,y_test)
r2_lin_train1=model_lin1.score(X_train,y_train)
print(r2_lin_test1,r2_lin_train1)

# Regression diagnostics- Residual plot analysis
residuals1=y_test-cars_predictions_lin1
sns.regplot(x=cars_predictions_lin1, y=residuals1, scatter=True,
            fit_reg=False)
residuals1.describe()

# =============================================================================
# RANDOM FOREST WITH OMITTED DATA
# =============================================================================

# Model parameters
rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
                           max_depth=100,min_samples_split=10,
                           min_samples_leaf=4,random_state=1)

# Model
model_rf1=rf.fit(X_train,y_train)

# Predicting model on test set
cars_predictions_rf1 = rf.predict(X_test)

# Computing MSE and RMSE
```

Object: sklearn.ensemble.RandomForestRegressor

max_featuresint, float, string or None, optional (default="auto")

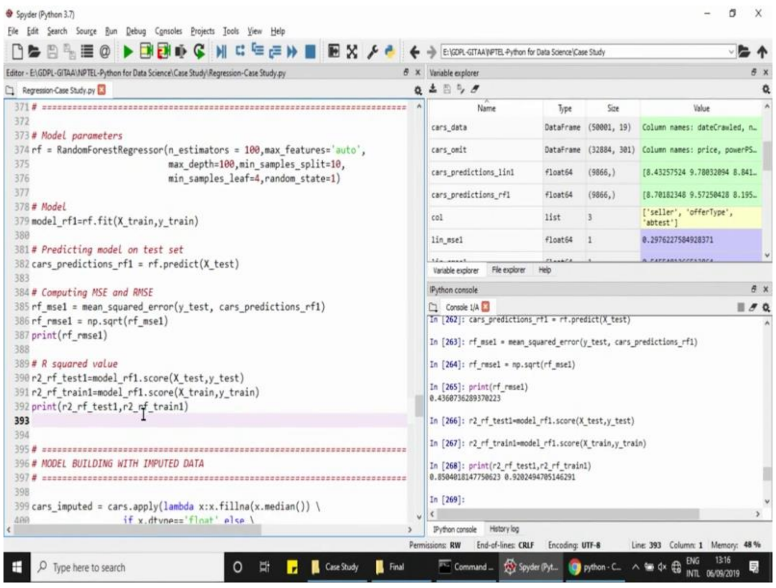
The number of features to consider when looking for the best split:

- If int, then consider *max_features* features at each split.
- If float, then *max_features* is a fraction and *int(max_features * n_features)* features are considered at each split.
- If "auto", then *max_features=n_features*.
- If "sqrt", then *max_features=sqrt(n_features)*.
- If "log2", then *max_features=log2(n_features)*.

And, the next input is maximum features which is basically the number of features the algorithm wants to consider to build the model or to build the tree right. Now, there are different options for this. If you give auto, then it automatically chooses otherwise you can give square root which is basically the square root of the number of features. So, each tree is built on different features.

So, there are other parameters as well which you can explore one by one and these are the main parameters that you usually start with. Again random state is equal to 1, because I have just chosen one random state, so that every time I run the model I get the same output.

(Refer Slide Time: 20:29)



```python
# ==========================================================================

# Model parameters
rf = RandomForestRegressor(n_estimators = 100,max_features='auto',
                           max_depth=100,min_samples_split=10,
                           min_samples_leaf=4,random_state=1)

# Model
model_rf1=rf.fit(X_train,y_train)

# Predicting model on test set
cars_predictions_rf1 = rf.predict(X_test)

# Computing MSE and RMSE
rf_mse1 = mean_squared_error(y_test, cars_predictions_rf1)
rf_rmse1 = np.sqrt(rf_mse1)
print(rf_rmse1)

# R squared value
r2_rf_test1=model_rf1.score(X_test,y_test)
r2_rf_train1=model_rf1.score(X_train,y_train)
print(r2_rf_test1,r2_rf_train1)


# ==========================================================================
# MODEL BUILDING WITH IMPUTED DATA
# ==========================================================================

cars_imputed = cars.apply(lambda x:x.fillna(x.median()) \
                 if x.dtype=='float' else \
```

Name	Type	Size	Value
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, n…
cars_omit	DataFrame	(32884, 301)	Column names: price, powerPS…
cars_predictions_lin1	float64	(9866,)	[8.43257524 9.78032094 8.841…
cars_predictions_rf1	float64	(9866,)	[8.70182348 9.57250428 8.195…
col	list	3	['seller', 'offerType', 'abtest']
lin_mse1	float64	1	0.2976227584928371

```
In [262]: cars_predictions_rf1 = rf.predict(X_test)

In [263]: rf_mse1 = mean_squared_error(y_test, cars_predictions_rf1)

In [264]: rf_rmse1 = np.sqrt(rf_mse1)

In [265]: print(rf_rmse1)
0.4360736289370223

In [266]: r2_rf_test1=model_rf1.score(X_test,y_test)

In [267]: r2_rf_train1=model_rf1.score(X_train,y_train)

In [268]: print(r2_rf_test1,r2_rf_train1)
0.8504018147750623 0.9202494705146291

In [269]:
```

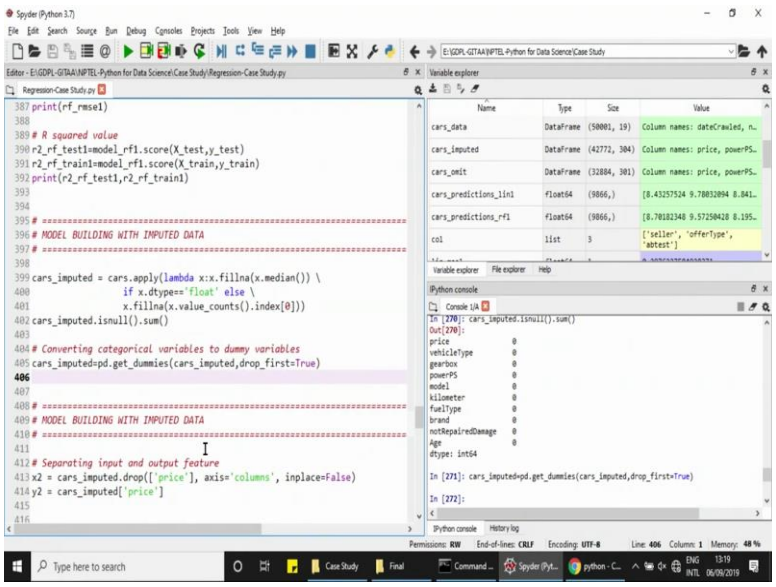
So, now let us just load the function. Now, rf is the model; now, I do rf.fit to fit the train data. Now, this will take some time, now the model was built using the train set of input and output features. Now, I am going to predict this model on the test set. Now, if you see under cars_predictions_rf1 you again have 9866 predictions ok. Now, we are again to compute the mean square error for this and the rmse turned out to be 0.44 roughly.

Now, this itself is a good indicator that your random forest is performing better than your linear regression model because the RMSE has come down further. Computer the R squared value here. So, let us just print out the R squared value. So, I have the R squared value here now. Now, the R squared value for the test is 0.85 and the a squared value for the train when the train is 0.92. So, definitely random forest is working a lot better compared to the linear regression model at least for this omitted data.

So, now let us go ahead and impute the missing cells in our data and let us now build the same linear and regression, now let us and let us build this linear regression model and random forest model on the data where these missing values have been imputed. So, I am using the same cars data that we are earlier started with. I am using a.apply function and within the apply function I am declaring a lambda function.

Now, the lambda function will fill the missing values with median if the data type of the column is float otherwise it will fill the missing cell with the most frequent category. Now, this the second condition will be applicable wherever your column type is object and the first condition will be applicable wherever your column type is float; either ways both cells will be imputed.

(Refer Slide Time: 22:27)



```
print(rf_rmse1)

# R squared value
r2_rf_test1=model_rf1.score(X_test,y_test)
r2_rf_train1=model_rf1.score(X_train,y_train)
print(r2_rf_test1,r2_rf_train1)


# =============================================================================
# MODEL BUILDING WITH IMPUTED DATA
# =============================================================================

cars_imputed = cars.apply(lambda x:x.fillna(x.median()) \
                  if x.dtype=='float' else \
                  x.fillna(x.value_counts().index[0]))
cars_imputed.isnull().sum()

# Converting categorical variables to dummy variables
cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)


# =============================================================================
# MODEL BUILDING WITH IMPUTED DATA
# =============================================================================

# Separating input and output feature
x2 = cars_imputed.drop(['price'], axis='columns', inplace=False)
y2 = cars_imputed['price']
```

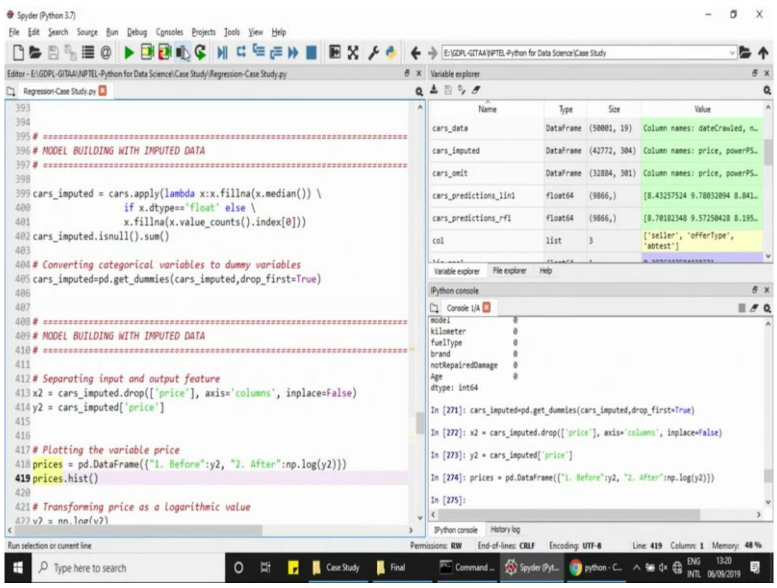
Name	Type	Size	Value
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, n…
cars_imputed	DataFrame	(42772, 304)	Column names: price, powerPS…
cars_omit	DataFrame	(32884, 301)	Column names: price, powerPS…
cars_predictions_lin1	float64	(9866,)	[8.43257524 9.78032094 8.841…
cars_predictions_rf1	float64	(9866,)	[8.70182348 9.57250428 8.195…
col	list	3	['seller', 'offerType', 'abtest']

```
In [270]: cars_imputed.isnull().sum()
Out[270]:
price               0
vehicleType         0
gearbox             0
powerPS             0
model               0
kilometer           0
fuelType            0
brand               0
notRepairedDamage   0
Age                 0
dtype: int64

In [271]: cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)

In [272]:
```

Now, if you impute and then you can check the number of null values under each you will see that none of the columns are missing values. So, in our data we had missing values only for the categorical columns and in that sense you are all these categorical columns are imputed with the most frequently occurring category.

Now, let us convert the categorical columns into dummy variables. So, I am going to use the same get dummies function. So, if you look at cars_imputed you again have 10 columns and when you do a dummy variable encoding, you get 304 columns. So, these are the number of columns that have been converted to dummy variables.

(Refer Slide Time: 23:13)



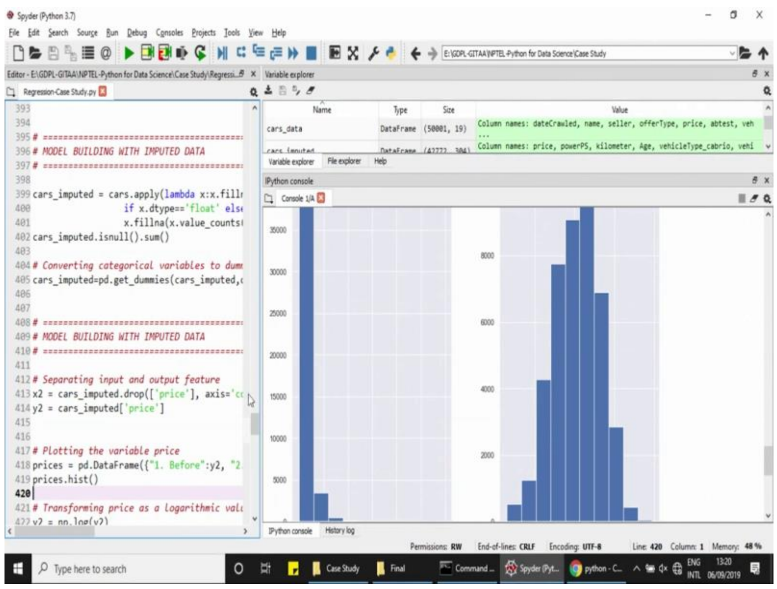
```
# ==============================================================================
# MODEL BUILDING WITH IMPUTED DATA
# ==============================================================================

cars_imputed = cars.apply(lambda x:x.fillna(x.median()) \
                  if x.dtype=='float' else \
                  x.fillna(x.value_counts().index[0]))
cars_imputed.isnull().sum()

# Converting categorical variables to dummy variables
cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)


# ==============================================================================
# MODEL BUILDING WITH IMPUTED DATA
# ==============================================================================

# Separating input and output feature
x2 = cars_imputed.drop(['price'], axis='columns', inplace=False)
y2 = cars_imputed['price']


# Plotting the variable price
prices = pd.DataFrame({"1. Before":y2, "2. After":np.log(y2)})
prices.hist()

# Transforming price as a logarithmic value
y2 = np.log(y2)
```

Name	Type	Size	Value
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, n…
cars_imputed	DataFrame	(42772, 304)	Column names: price, powerPS…
cars_omit	DataFrame	(32884, 301)	Column names: price, powerPS…
cars_predictions_lin1	float64	(9866,)	[8.43257524 9.78032094 8.841…
cars_predictions_rf1	float64	(9866,)	[8.70182348 9.57250428 8.195…
col	list	3	['seller', 'offerType', 'abtest']

```
model                0
kilometer            0
fuelType             0
brand                0
notRepairedDamage    0
Age                  0
dtype: int64

In [271]: cars_imputed=pd.get_dummies(cars_imputed,drop_first=True)

In [272]: x2 = cars_imputed.drop(['price'], axis='columns', inplace=False)

In [273]: y2 = cars_imputed['price']

In [274]: prices = pd.DataFrame({"1. Before":y2, "2. After":np.log(y2)})

In [275]:
```

So, let us again repeat the same drill where I separate the input features and the output set of features. Now, I am again plotting y2 what you get and np.log of y2.

(Refer Slide Time: 23:21)

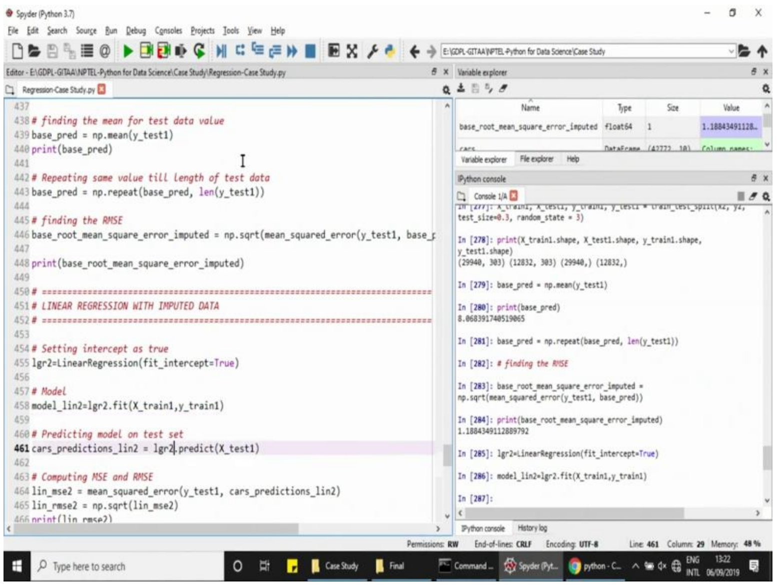


```
# ======================================
# MODEL BUILDING WITH IMPUTED DATA
# ======================================

cars_imputed = cars.apply(lambda x:x.fillr
                  if x.dtype=='float' else
                  x.fillna(x.value_counts(
cars_imputed.isnull().sum()

# Converting categorical variables to dumr
cars_imputed=pd.get_dummies(cars_imputed,c


# ======================================
# MODEL BUILDING WITH IMPUTED DATA
# ======================================

# Separating input and output feature
x2 = cars_imputed.drop(['price'], axis='cc
y2 = cars_imputed['price']


# Plotting the variable price
prices = pd.DataFrame({"1. Before":y2, "2.
prices.hist()

# Transforming price as a logarithmic valu
y2 = np.log(y2)
```

Name	Type	Size	Value
cars_data	DataFrame	(50001, 19)	Column names: dateCrawled, name, seller, offerType, price, abtest, veh ...
cars_imputed	DataFrame	(42772, 304)	Column names: price, powerPS, kilometer, Age, vehicleType_cabrio, vehi

And, if you plot a histogram you can again see that value for price is normally distributed and this is the same as before. So, all our predictions are going to be for a log transformed value of price. So, I am going ahead and I am transforming it to a natural logarithm.

(Refer Slide Time: 23:35)



```
# finding the mean for test data value
base_pred = np.mean(y_test1)
print(base_pred)

# Repeating same value till length of test data
base_pred = np.repeat(base_pred, len(y_test1))

# finding the RMSE
base_root_mean_square_error_imputed = np.sqrt(mean_squared_error(y_test1, base_p

print(base_root_mean_square_error_imputed)

# ==========================================================================
# LINEAR REGRESSION WITH IMPUTED DATA
# ==========================================================================

# Setting intercept as true
lgr2=LinearRegression(fit_intercept=True)

# Model
model_lin2=lgr2.fit(X_train1,y_train1)

# Predicting model on test set
cars_predictions_lin2 = lgr2.predict(X_test1)

# Computing MSE and RMSE
lin_mse2 = mean_squared_error(y_test1, cars_predictions_lin2)
lin_rmse2 = np.sqrt(lin_mse2)
print(lin_rmse2)
```

Name	Type	Size	Value
base_root_mean_square_error_imputed	float64	1	1.18843491128…
cars	DataFrame	(42772, 10)	Column names:

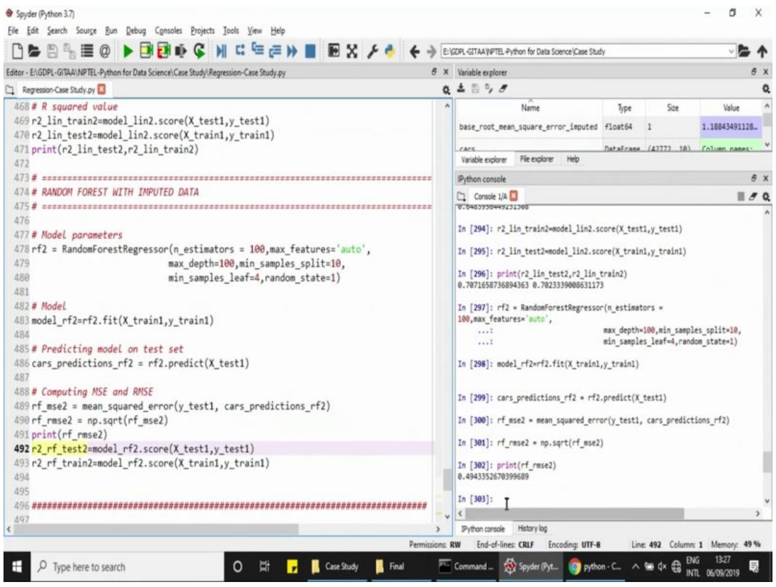
```
test_size=0.3, random_state = 3)

In [278]: print(X_train1.shape, X_test1.shape, y_train1.shape,
y_test1.shape)
(29940, 303) (12832, 303) (29940,) (12832,)

In [279]: base_pred = np.mean(y_test1)

In [280]: print(base_pred)
8.068391740519065

In [281]: base_pred = np.repeat(base_pred, len(y_test1))

In [282]: # finding the RMSE

In [283]: base_root_mean_square_error_imputed =
np.sqrt(mean_squared_error(y_test1, base_pred))

In [284]: print(base_root_mean_square_error_imputed)
1.1884349112889792

In [285]: lgr2=LinearRegression(fit_intercept=True)

In [286]: model_lin2=lgr2.fit(X_train1,y_train1)

In [287]:
```

And then I am going to split the data into test and train again and I am using the same split ratio which is 0.3 and if you print the shape you can see 70 percent of it has gone to train and remaining 30 percent have gone to test. And, here are the individual split up of the test set and the train set for the input and output features.

So, let us again start by building a baseline model. Now, here I am going to use the y_test1 I am again going to find the base RMSE for this. This is the same function as before and just that by instead of having y_test, I am doing y_test1. So, that is the only change that is here and in this case we have an RMSE value of 1.18.

So, again let us go ahead with linear regression I am going to repeat the same set of steps. I am building a model I am fitting the model with the train data and then I am predicting the test data. So, this is the RMSE value. So, I am going to use the same function again and if you print the RMSE you can see it is 0.648. Now, after imputing you can see that the RMSE value for the linear regression model has gone up right. It was earlier at around 0.56 for the same data with, but without cells which were missing. So, the error has gone slightly up. Now, let us go ahead with random forest model for the same data.

(Refer Slide Time: 25:01)

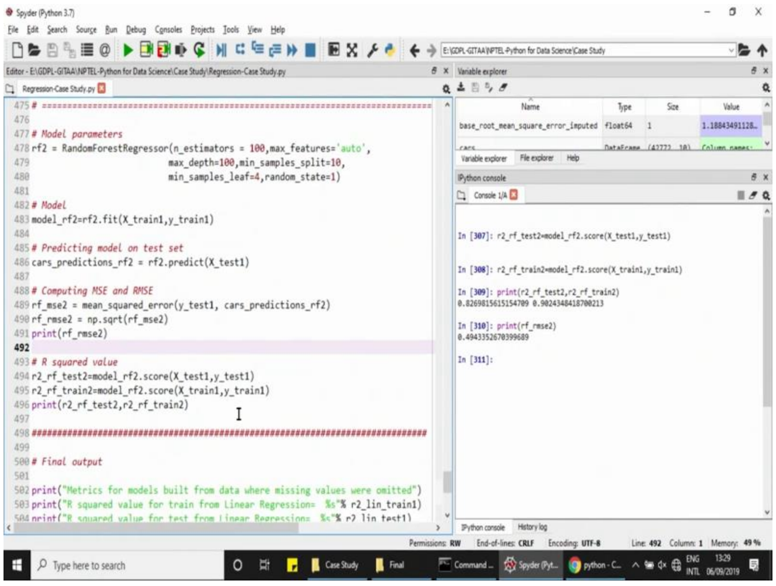


```
# R squared value
r2_lin_train2=model_lin2.score(X_test1,y_test1)
r2_lin_test2=model_lin2.score(X_train1,y_train1)
print(r2_lin_test2,r2_lin_train2)

# =============================================================================
# RANDOM FOREST WITH IMPUTED DATA
# =============================================================================

# Model parameters
rf2 = RandomForestRegressor(n_estimators = 100,max_features='auto',
                            max_depth=100,min_samples_split=10,
                            min_samples_leaf=4,random_state=1)

# Model
model_rf2=rf2.fit(X_train1,y_train1)

# Predicting model on test set
cars_predictions_rf2 = rf2.predict(X_test1)

# Computing MSE and RMSE
rf_mse2 = mean_squared_error(y_test1, cars_predictions_rf2)
rf_rmse2 = np.sqrt(rf_mse2)
print(rf_rmse2)
r2_rf_test2=model_rf2.score(X_test1,y_test1)
r2_rf_train2=model_rf2.score(X_train1,y_train1)


################################################################################
```

```
In [294]: r2_lin_train2=model_lin2.score(X_test1,y_test1)

In [295]: r2_lin_test2=model_lin2.score(X_train1,y_train1)

In [296]: print(r2_lin_test2,r2_lin_train2)
0.7071658736894363 0.7023339008631173

In [297]: rf2 = RandomForestRegressor(n_estimators =
100,max_features='auto',
     ...:                             max_depth=100,min_samples_split=10,
     ...:                             min_samples_leaf=4,random_state=1)

In [298]: model_rf2=rf2.fit(X_train1,y_train1)


In [299]: cars_predictions_rf2 = rf2.predict(X_test1)

In [300]: rf_mse2 = mean_squared_error(y_test1, cars_predictions_rf2)

In [301]: rf_rmse2 = np.sqrt(rf_mse2)

In [302]: print(rf_rmse2)
0.4943352670399689

In [303]:
```

I am again building a model with all parameters I have kept the parameters value the same as before just to standardize both the methods and I am again fitting the model on the train and predicting on the test. Again, this will take some time to run. So, the model has run and it is predicted on the test set. So, let us just compute the RMSE again and RMSE that you get in this case is 0.494. That is a lot less compared to what the linear regression model k with imputed data.

(Refer Slide Time: 25:33)

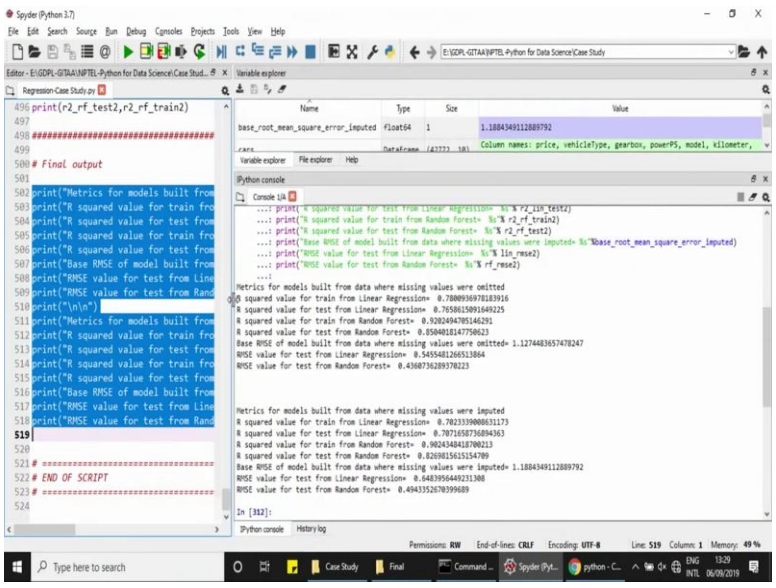


```
# =============================================================================

# Model parameters
rf2 = RandomForestRegressor(n_estimators = 100,max_features='auto',
                            max_depth=100,min_samples_split=10,
                            min_samples_leaf=4,random_state=1)

# Model
model_rf2=rf2.fit(X_train1,y_train1)

# Predicting model on test set
cars_predictions_rf2 = rf2.predict(X_test1)

# Computing MSE and RMSE
rf_mse2 = mean_squared_error(y_test1, cars_predictions_rf2)
rf_rmse2 = np.sqrt(rf_mse2)
print(rf_rmse2)

# R squared value
r2_rf_test2=model_rf2.score(X_test1,y_test1)
r2_rf_train2=model_rf2.score(X_train1,y_train1)
print(r2_rf_test2,r2_rf_train2)

################################################################################

# Final output

print("Metrics for models built from data where missing values were omitted")
print("R squared value for train from Linear Regression=  %s"% r2_lin_train1)
print("R squared value for test from Linear Regression=  %s"% r2_lin_test1)
```

```
In [307]: r2_rf_test2=model_rf2.score(X_test1,y_test1)


In [308]: r2_rf_train2=model_rf2.score(X_train1,y_train1)

In [309]: print(r2_rf_test2,r2_rf_train2)
0.8269815615154709 0.9024348418700213

In [310]: print(rf_rmse2)
0.4943352670399689

In [311]:
```

So, again let us compute the R squared value for this and R squared value for the random forest regression module after imputing turns out to be 0.83 for the test set and on the train set it is about 0.9. Now, that is a very good R squared value compared to the linear regression model that we had got earlier and similarly, if even if you look at the RMSE value we had got about 0.49 and that is a good value compared to the linear regression model on the imputed data. So, let us just print all of these and condense it together.

(Refer Slide Time: 26:07)



```
print(r2_rf_test2,r2_rf_train2)

########################################

# Final output

print("Metrics for models built from
print("R squared value for train fro
print("R squared value for test from
print("R squared value for train fro
print("R squared value for test from
print("Base RMSE of model built from
print("RMSE value for test from Line
print("RMSE value for test from Rand
print("\n\n")
print("Metrics for models built from
print("R squared value for train fro
print("R squared value for test from
print("R squared value for train fro
print("R squared value for test from
print("Base RMSE of model built from
print("RMSE value for test from Line
print("RMSE value for test from Rand


# ====================================
# END OF SCRIPT
# ====================================
```

Name	Type	Size	Value
base_root_mean_square_error_imputed	float64	1	1.1884349112889792
cars	DataFrame	(42772, 10)	Column names: price, vehicleType, gearbox, powerPS, model, kilometer,

```
    ...: print("R squared value for test from Linear Regression=  %s"% r2_lin_test2)
    ...: print("R squared value for train from Random Forest=  %s"% r2_rf_train2)
    ...: print("R squared value for test from Random Forest=  %s"% r2_rf_test2)
    ...: print("Base RMSE of model built from data where missing values were imputed= %s"%base_root_mean_square_error_imputed)
    ...: print("RMSE value for test from Linear Regression=  %s"% lin_rmse2)
    ...: print("RMSE value for test from Random Forest=  %s"% rf_rmse2)
    ...:
Metrics for models built from data where missing values were omitted
R squared value for train from Linear Regression=  0.7800936978183916
R squared value for test from Linear Regression=  0.7658615091649225
R squared value for train from Random Forest=  0.9202494705146291
R squared value for test from Random Forest=  0.8504018147750623
Base RMSE of model built from data where missing values were omitted= 1.1274483657478247
RMSE value for test from Linear Regression=  0.5455481266513864
RMSE value for test from Random Forest=  0.4360736289370223



Metrics for models built from data where missing values were imputed
R squared value for train from Linear Regression=  0.7023339008631173
R squared value for test from Linear Regression=  0.7071658736894363
R squared value for train from Random Forest=  0.9024348418700213
R squared value for test from Random Forest=  0.8269815615154709
Base RMSE of model built from data where missing values were imputed= 1.1884349112889792
RMSE value for test from Linear Regression=  0.6483956449231308
RMSE value for test from Random Forest=  0.4943352670399689

In [312]:
```

So, basically I am just printing these values, so that we can infer the message. So, I what I am printing here is basically a condensed form of whatever we have computed till now. I have taken two sets; one is a set of metric for the omitted data and the other set of metrics for the imputed data. Now, for the omitted data you can see that the R squared value for train from the linear regression is 0.78, again on test it is 0.76. So, it is more or less close and similarly, for random forest the R square on train is 0.92 and the R squared value on point on test is 0.85.

Now, if you look at the base RMSE value it is for 0.12 and for this data for linear regression the RMSE value turns out to be 0.54 and the RMSE value from random forests turns out to be 0.44 and these are also the heuristics that we have computed earlier. So, this tells you that definitely on at least we omitted data, the random forest is definitely performing a lot better compared to the linear regression model.

Now, moving further to the imputed set of data if you look at the R squared value from the linear regression on train it is 0.70 and it is more or less the same even on the test. And if you look at the r squared value from random forest on the train it is 0.9 and on the test it is a 0.826. So, that is again a good value and it is a lot better compared to your linear regression model. And the base RMSE for this data is at 1.18 and the RMSE value from the linear regression that you record this 0.64, but however, if you look at the RMSE from the random forest it is at 0.49.

So, definitely for the imputed data, there is a huge remarkable change in the RMSE value between your linear regression and random forest model. So, your random forest model is definitely a lot better than your linear regression model and this is very important because it is if the models are performing good for data where missing values are omitted because in that case you have a clean data you are ready full, but most often. But, more often than not problem comes only when you start impute in sells because you are not sure about with which value you suppose to impute and how the results are going to turn out right.

So, at least in this case the imputation looks like the imputation has not gone really wrong and random forests definitely is a lot better than linear regression. And, this is also very and it is also very important to impute values because you do not want to let go of information and you do not want to let go of the number of records that you have. In the earlier case where we did omit we lost about 10000 record, we had around 33000 on records. But, after imputing, you can see that random forest definitely able to capture the variations much better than linear regression.

So, it is a necessary that you do not let go of information that you already have, but at the same time there is a tradeoff between how much you can retain and how much you can leave.

So, thank you.