housing-case-study-using-rfe

December 4, 2023

0.1 Model Selection using RFE (Housing Case Study)

0.1.1 Importing and Understanding Data

[1]:
```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import pandas as pd
import numpy as np
```

[3]:
```
# Importing Housing.csv
housing = pd.read_csv('Housing.csv')
```

[4]:
```
# Looking at the first five rows
housing.head()
```

[4]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

0.1.2 Data Preparation

[5]:
```
# List of variables to map

varlist =  ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']
```

```
# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
```

[6]:
```
# Check the housing dataframe now

housing.head()
```

[6]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom
0	13300000	7420	4	2	3	1	0
1	12250000	8960	4	4	4	1	0
2	12250000	9960	3	2	2	1	0
3	12215000	7500	4	2	2	1	0
4	11410000	7420	4	1	2	1	1

	basement	hotwaterheating	airconditioning	parking	prefarea
0	0	0	1	2	1
1	0	0	1	3	0
2	1	0	0	2	1
3	1	0	1	3	1
4	1	0	1	2	0

	furnishingstatus
0	furnished
1	furnished
2	semi-furnished
3	furnished
4	furnished

0.1.3 Dummy Variables

The variable `furnishingstatus` has three levels. We need to convert these levels into integer as well. For this, we will use something called `dummy variables`.

[7]:
```
# Get the dummy variables for the feature 'furnishingstatus' and store it in a␣
 ↪new variable - 'status'

status = pd.get_dummies(housing['furnishingstatus'])

# Check what the dataset 'status' looks like
status.head()
```

[7]:

	furnished	semi-furnished	unfurnished
0	1	0	0
1	1	0	0
2	0	1	0
3	1	0	0
4	1	0	0

Now, you don't need three columns. You can drop the `furnished` column, as the type of furnishing can be identified with just the last two columns where — - `00` will correspond to `furnished` - `01` will correspond to `unfurnished` - `10` will correspond to `semi-furnished`

[8]:
```
# Let's drop the first column from status df using 'drop_first = True'
status = pd.get_dummies(housing['furnishingstatus'], drop_first = True)

# Add the results to the original housing dataframe
housing = pd.concat([housing, status], axis = 1)

# Now let's see the head of our dataframe.
housing.head()
```

[8]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	\
0	13300000	7420	4	2	3	1	0	
1	12250000	8960	4	4	4	1	0	
2	12250000	9960	3	2	2	1	0	
3	12215000	7500	4	2	2	1	0	
4	11410000	7420	4	1	2	1	1	

	basement	hotwaterheating	airconditioning	parking	prefarea	\
0	0	0	1	2	1	
1	0	0	1	3	0	
2	1	0	0	2	1	
3	1	0	1	3	1	
4	1	0	1	2	0	

	furnishingstatus	semi-furnished	unfurnished
0	furnished	0	0
1	furnished	0	0
2	semi-furnished	1	0
3	furnished	0	0
4	furnished	0	0

[9]:
```
# Drop 'furnishingstatus' as we have created the dummies for it
housing.drop(['furnishingstatus'], axis = 1, inplace = True)

housing.head()
```

[9]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	semi-furnished	unfurnished
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	0	0
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	0	0
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	1	0
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	0	0
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	0	0

0.2 Splitting the Data into Training and Testing Sets

[10]:
```
from sklearn.model_selection import train_test_split

# We specify this so that the train and test data set always have the same␣
 ↪rows, respectively

df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.
 ↪3, random_state = 100)
```

0.2.1 Rescaling the Features

We will use MinMax scaling.

[11]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

[12]:
```
# Apply scaler() to all the columns except the 'yes-no' and 'dummy' variables
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']

df_train[num_vars] = scaler.fit_transform(df_train[num_vars])

df_train.head()
```

[12]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom
359	0.169697	0.155227	0.4	0.0	0.000000	1	0

```
19   0.615152  0.403379       0.4        0.5  0.333333         1          0
159  0.321212  0.115628       0.4        0.5  0.000000         1          1
35   0.548133  0.454417       0.4        0.5  1.000000         1          0
28   0.575758  0.538015       0.8        0.5  0.333333         1          0

     basement  hotwaterheating  airconditioning   parking  prefarea  \
359         0                0                0  0.333333         0
19          0                0                1  0.333333         1
159         1                0                1  0.000000         0
35          0                0                1  0.666667         0
28          1                1                0  0.666667         0

     semi-furnished  unfurnished
359               0            1
19                1            0
159               0            0
35                0            0
28                0            1
```

0.2.2 Dividing into X and Y sets for the model building

[13]:
```
y_train = df_train.pop('price')
X_train = df_train
```

0.3 Building our model

This time, we will be using the **LinearRegression function from SciKit Learn** for its compatibility with RFE (which is a utility from sklearn)

0.3.1 RFE

Recursive feature elimination

[14]:
```
# Importing RFE and LinearRegression
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

[15]:
```
# Running RFE with the output number of the variable equal to 10
lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, 10)             # running RFE
rfe = rfe.fit(X_train, y_train)
```

[16]:
```
list(zip(X_train.columns,rfe.support_,rfe.ranking_))
```

[16]:
```
[('area', True, 1),
 ('bedrooms', True, 1),
 ('bathrooms', True, 1),
 ('stories', True, 1),
 ('mainroad', True, 1),
 ('guestroom', True, 1),
 ('basement', False, 3),
 ('hotwaterheating', True, 1),
 ('airconditioning', True, 1),
 ('parking', True, 1),
 ('prefarea', True, 1),
 ('semi-furnished', False, 4),
 ('unfurnished', False, 2)]
```

[17]:
```
col = X_train.columns[rfe.support_]
col
```

[17]:
```
Index(['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',
       'hotwaterheating', 'airconditioning', 'parking', 'prefarea'],
      dtype='object')
```

[18]:
```
X_train.columns[~rfe.support_]
```

[18]:
```
Index(['basement', 'semi-furnished', 'unfurnished'], dtype='object')
```

0.3.2 Building model using statsmodel, for the detailed statistics

[19]:
```
# Creating X_test dataframe with RFE selected variables
X_train_rfe = X_train[col]
```

[20]:
```
# Adding a constant variable
import statsmodels.api as sm
X_train_rfe = sm.add_constant(X_train_rfe)
```

[21]:
```
lm = sm.OLS(y_train,X_train_rfe).fit()   # Running the linear model
```

[22]:
```
#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.669
Model:                            OLS   Adj. R-squared:                  0.660
Method:                 Least Squares   F-statistic:                     74.89
Date:                Tue, 09 Oct 2018   Prob (F-statistic):           1.28e-82
Time:                        13:15:31   Log-Likelihood:                374.65
No. Observations:                 381   AIC:                            -727.3
```

```
Df Residuals:                      370   BIC:                            -683.9
Df Model:                           10
Covariance Type:            nonrobust
==============================================================================
===
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
---
const              0.0027      0.018      0.151      0.880      -0.033
0.038
area               0.2363      0.030      7.787      0.000       0.177
0.296
bedrooms           0.0661      0.037      1.794      0.074      -0.006
0.139
bathrooms          0.1982      0.022      8.927      0.000       0.155
0.242
stories            0.0977      0.019      5.251      0.000       0.061
0.134
mainroad           0.0556      0.014      3.848      0.000       0.027
0.084
guestroom          0.0381      0.013      2.934      0.004       0.013
0.064
hotwaterheating    0.0897      0.022      4.104      0.000       0.047
0.133
airconditioning    0.0711      0.011      6.235      0.000       0.049
0.093
parking            0.0637      0.018      3.488      0.001       0.028
0.100
prefarea           0.0643      0.012      5.445      0.000       0.041
0.088
==============================================================================
Omnibus:                       86.105   Durbin-Watson:                   2.098
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              286.069
Skew:                           0.992   Prob(JB):                     7.60e-63
Kurtosis:                       6.753   Cond. No.                         13.2
==============================================================================

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

`Bedrooms` is insignificant in presence of other variables; can be dropped

[23]:
```
X_train_new = X_train_rfe.drop(["bedrooms"], axis = 1)
```

Rebuilding the model without `bedrooms`

[24]:
```
# Adding a constant variable
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_new)
```

[25]:
```
lm = sm.OLS(y_train,X_train_lm).fit()   # Running the linear model
```

[26]:
```
#Let's see the summary of our linear model
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.666
Model:                            OLS   Adj. R-squared:                  0.658
Method:                 Least Squares   F-statistic:                     82.37
Date:                Tue, 09 Oct 2018   Prob (F-statistic):           6.67e-83
Time:                        13:15:31   Log-Likelihood:                373.00
No. Observations:                 381   AIC:                            -726.0
Df Residuals:                     371   BIC:                            -686.6
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0242      0.013      1.794      0.074      -0.002       0.051
area           0.2367      0.030      7.779      0.000       0.177       0.297
bathrooms      0.2070      0.022      9.537      0.000       0.164       0.250
stories        0.1096      0.017      6.280      0.000       0.075       0.144
mainroad       0.0536      0.014      3.710      0.000       0.025       0.082
guestroom      0.0390      0.013      2.991      0.003       0.013       0.065
hotwaterheating     0.0921      0.022      4.213      0.000       0.049       0.135
airconditioning     0.0710      0.011      6.212      0.000       0.049       0.094
parking        0.0669      0.018      3.665      0.000       0.031       0.103
prefarea       0.0653      0.012      5.513      0.000       0.042       0.089
==============================================================================
```

```
Omnibus:                       91.542   Durbin-Watson:                   2.107
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              315.402
Skew:                           1.044   Prob(JB):                     3.25e-69
Kurtosis:                       6.938   Cond. No.                         10.0
==============================================================================

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

[27]:
```
X_train_new.columns
```

[27]:
```
Index(['const', 'area', 'bathrooms', 'stories', 'mainroad', 'guestroom',
       'hotwaterheating', 'airconditioning', 'parking', 'prefarea'],
      dtype='object')
```

[28]:
```
X_train_new = X_train_new.drop(['const'], axis=1)
```

[29]:
```
# Calculate the VIFs for the new model
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X = X_train_new
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[29]:

	Features	VIF
0	area	4.52
3	mainroad	4.26
2	stories	2.12
7	parking	2.10
6	airconditioning	1.75
1	bathrooms	1.58
8	prefarea	1.47
4	guestroom	1.30
5	hotwaterheating	1.12

0.4 Residual Analysis of the train data

So, now to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like.

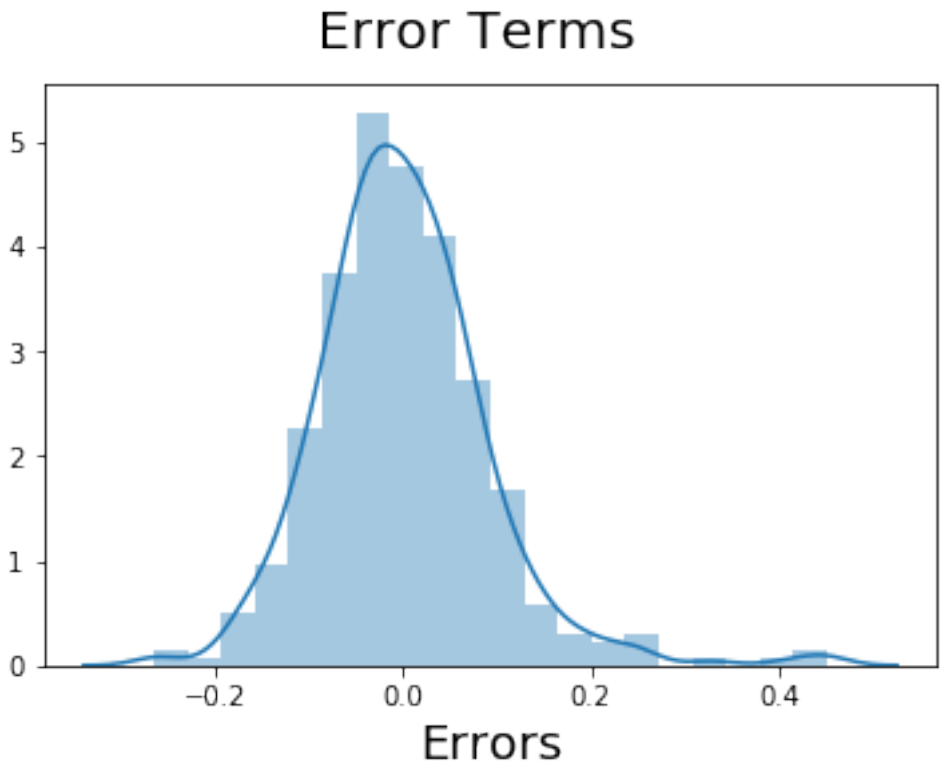
[30]:
```
y_train_price = lm.predict(X_train_lm)
```

[31]:
```
# Importing the required libraries for plots.
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[32]:
```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)                         # X-label
```

```
C:\Users\admin\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462:
UserWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

[32]:
```
Text(0.5,0,'Errors')
```



0.5 Making Predictions

Applying the scaling on the test sets

[33]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']

df_test[num_vars] = scaler.transform(df_test[num_vars])
```

Dividing into X_test and y_test

[34]:
```
y_test = df_test.pop('price')
X_test = df_test
```

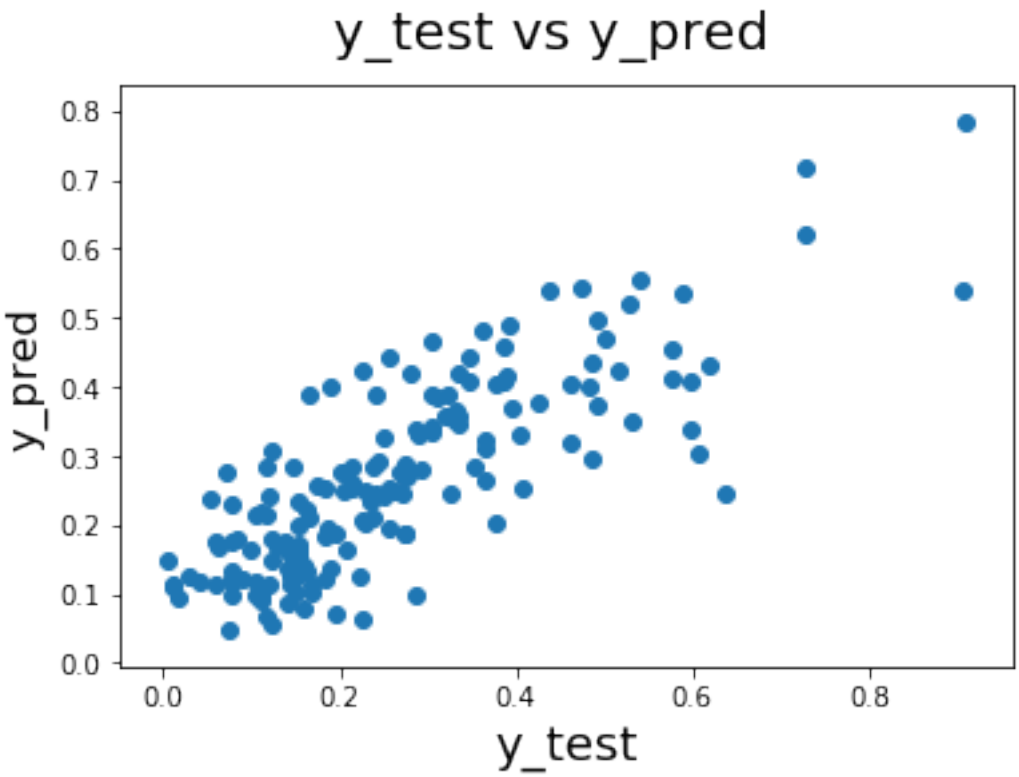
[35]:
```
# Now let's use our model to make predictions.

# Creating X_test_new dataframe by dropping variables from X_test
X_test_new = X_test[X_train_new.columns]

# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)
```

[36]:
```
# Making predictions
y_pred = lm.predict(X_test_new)
```

0.6 Model Evaluation

[37]:
```
# Plotting y_test and y_pred to understand the spread.
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)                          # Y-label
```

[37]:
```
Text(0,0.5,'y_pred')
```


y_test vs y_pred
y_pred
y_test
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.0
0.2
0.4
0.6
0.8