4_LinearRegression

August 27, 2024

[6]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Analysing the alcohol content of white wines using linear regression

df = pd.read_csv("winequality-white.csv", sep=";")
df.describe()
```

[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar
count	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415
std	0.843868	0.100795	0.121020	5.072058
min	3.800000	0.080000	0.000000	0.600000
25%	6.300000	0.210000	0.270000	1.700000
50%	6.800000	0.260000	0.320000	5.200000
75%	7.300000	0.320000	0.390000	9.900000
max	14.200000	1.100000	1.660000	65.800000

	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	4898.000000	4898.000000	4898.000000	4898.000000
mean	0.045772	35.308085	138.360657	0.994027
std	0.021848	17.007137	42.498065	0.002991
min	0.009000	2.000000	9.000000	0.987110
25%	0.036000	23.000000	108.000000	0.991723
50%	0.043000	34.000000	134.000000	0.993740
75%	0.050000	46.000000	167.000000	0.996100
max	0.346000	289.000000	440.000000	1.038980

	pH	sulphates	alcohol	quality
count	4898.000000	4898.000000	4898.000000	4898.000000
mean	3.188267	0.489847	10.514267	5.877909
std	0.151001	0.114126	1.230621	0.885639

```
min       2.720000      0.220000      8.000000      3.000000
25%       3.090000      0.410000      9.500000      5.000000
50%       3.180000      0.470000     10.400000      6.000000
75%       3.280000      0.550000     11.400000      6.000000
max       3.820000      1.080000     14.200000      9.000000
```

[8]:
```
# Drop quality as we are measuring alcohol content
df.drop(['quality'], axis=1, inplace=True)

# Split the data into explanatory variables (columns 0 to 9) and a response␣
 ↪variable (alcohol, column 10)
x = df.iloc[:,0:10]
y = df.iloc[:,10]

# Split explanatory and response variables into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=42)

# Fit a linear regression model to the training data
reg = linear_model.LinearRegression()
reg.fit(x_train, y_train)

print("Coefficients:" ,reg.coef_)
print("Intercept:", reg.intercept_)
```

```
Coefficients: [ 5.21769000e-01  9.41817911e-01  4.05432209e-01  2.40677212e-01
 -3.40163473e-01 -3.46346533e-03  3.28399093e-04 -6.87572769e+02
  2.48011734e+00  1.03876560e+00]
Intercept: 680.1454546075421
```

[12]:
```
# Calculate error statistics on the testing data and print them
y_pred = reg.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
r2s = r2_score(y_test, y_pred)

print("MSE = ", mse)
print("R2s = ", r2s)
```

```
MSE =  0.14358293516027423
R2s =  0.9068068867998738
```

[14]:
```
# Compare observed and predicted values
df_comp = pd.DataFrame();
df_comp['observed'] = y_test
df_comp['predicted'] = y_pred
df_comp['residual'] = df_comp['observed'] - df_comp['predicted']
df_comp
```

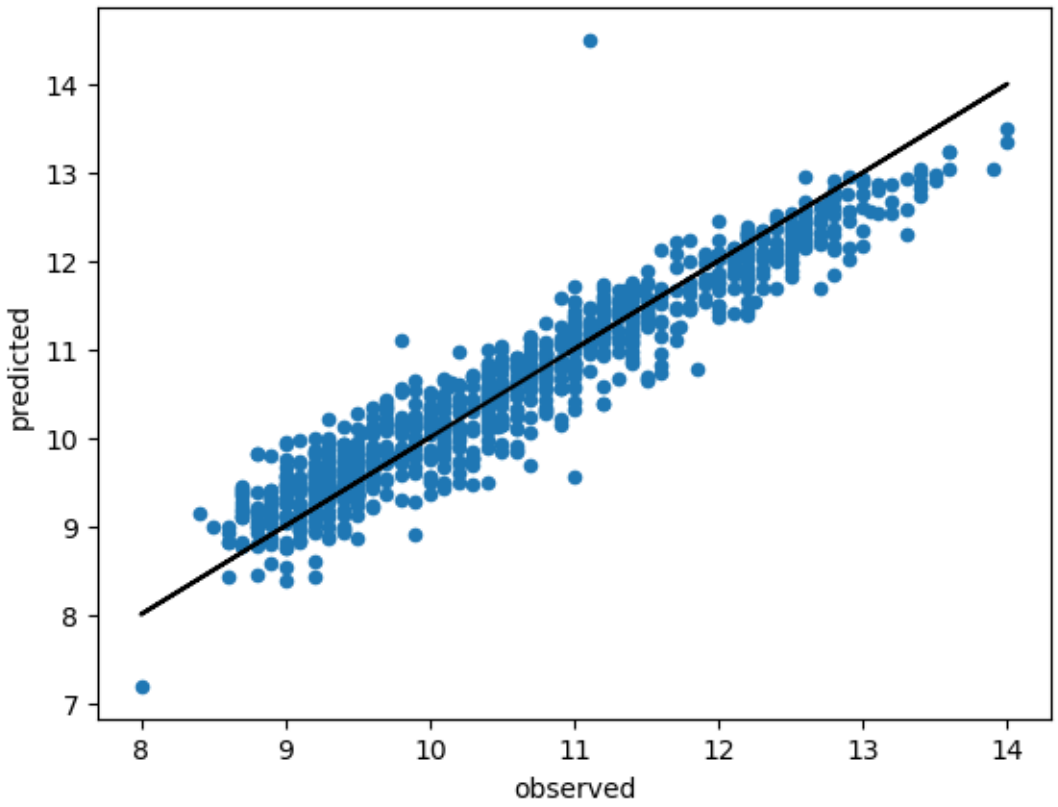
[14]:

	observed	predicted	residual
4656	10.966667	11.192224	-0.225557
3659	13.200000	12.868885	0.331115
907	12.200000	12.007936	0.192064
4352	10.700000	11.122002	-0.422002
3271	12.000000	12.065315	-0.065315
…	…	…	…
3207	10.900000	10.554635	0.345365
1539	10.800000	10.940283	-0.140283
964	9.500000	9.409926	0.090074
168	9.300000	9.400363	-0.100363
3661	10.000000	10.331225	-0.331225

[980 rows x 3 columns]

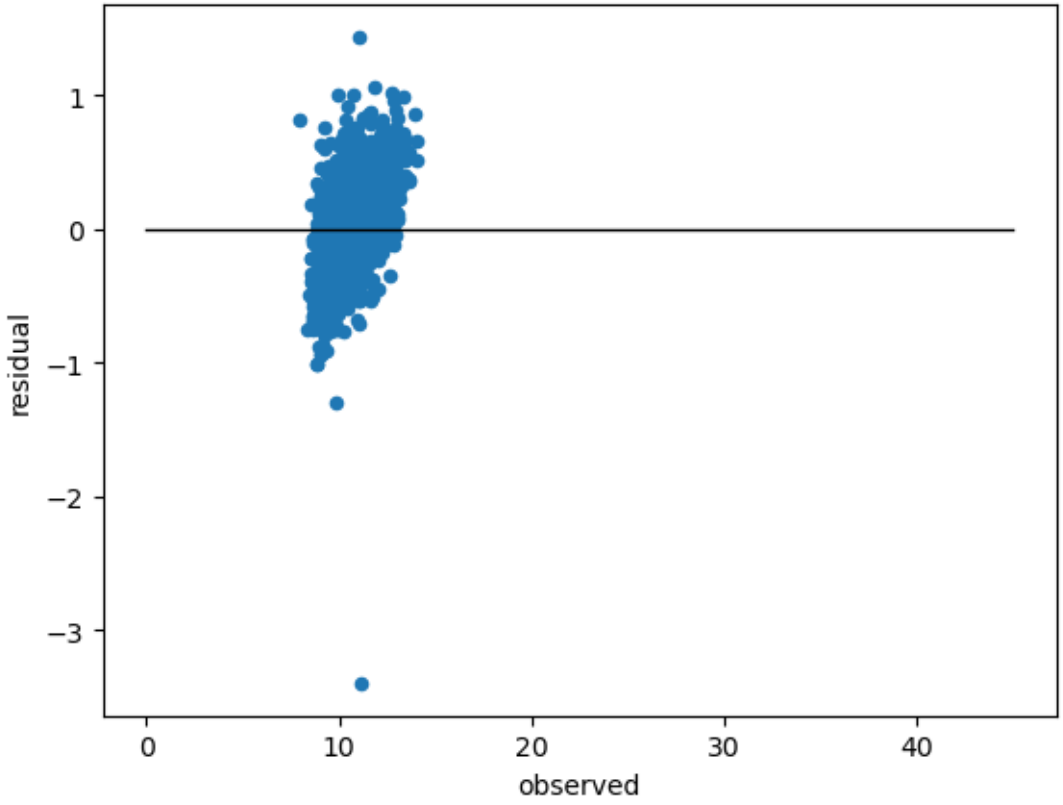
[16]:
```
# Creating a plot of the linear regression
plt.figure()
df_comp.plot.scatter(x='observed', y='predicted') # Values added to the plot
plt.plot(df_comp['observed'], df_comp['observed'], color="black")  # Line of␣
 ↪perfect prediction
plt.show()
```

<Figure size 640x480 with 0 Axes>



[18]:
```
# Creating a plot of residuals
plt.figure()
df_comp.plot.scatter(x='observed', y='residual')
plt.plot([0,45], [0,0], color='black', linewidth=1)
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```



[20]:
```
# Train a scaled version of the model to figure out the most important␣
 ↪parameters
scaler = StandardScaler()
x_std = scaler.fit_transform(x_train)
reg_scaled = linear_model.LinearRegression()
reg_scaled.fit(x_std, y_train)
print("Coefficients:" ,reg_scaled.coef_)
print("Intercept:", reg_scaled.intercept_)
```

```
Coefficients: [ 0.44056889  0.09568203  0.0485474   1.23675709 -0.00741374
-0.0577527
```

```
  0.01381322 -2.07736606  0.37242449  0.11797831]
Intercept: 10.508840394759204
```

[22]:
```
# Sorting values based on their importance to predict alcohol content
df_importance = pd.DataFrame()
df_importance['column'] = x_test.columns
df_importance['abs_scaled_coef'] = np.abs(reg_scaled.coef_)
df_importance.sort_values(by=['abs_scaled_coef'], ascending=False)
```

[22]:

	column	abs_scaled_coef
7	density	2.077366
3	residual sugar	1.236757
0	fixed acidity	0.440569
8	pH	0.372424
9	sulphates	0.117978
1	volatile acidity	0.095682
5	free sulfur dioxide	0.057753
2	citric acid	0.048547
6	total sulfur dioxide	0.013813
4	chlorides	0.007414

1 Questions

1.1 What is the regression equation for estimating the amount of alcohol in white wine?

The regression equation is:

y = 10.508840394759204 + 0.44056889 * fixed acidity + 0.09568203 * volatiole acidity + 0.0485474 * citric acid + 1.23675709 * residual sugar − 0.00741374 * chlorides − 0.0577527 * free sulfur dioxide + 0.01381322 * total sulfur dioxide − 2.07736606 * density + 0.37242449 * pH + 0.11797831 * sulphates

1.2 What are the five most useful variables for estimating the trait values?

Order from most useful to least useful, the top 5 variables are: Density, Residual Sugar, Fixed Acidity, pH, Sulphates.

1.3 Provide a validation-based error estimate for your model. As the data set is large, use split validation that divides the data set into separate training and testing sets.

Split validation-based mean squared error is 0.14358293516027423 and R squared 0.9068068867998738. When the model is performing well, you will see a low mean squared error value and high (close to 1) R squared value.