Labs

February 21, 2025

1 Lab: Linear Regression

1.1 Set up IPython libraries for customizing notebook display

[1]:
```
from notebookfuncs import *
```

1.2 Import standard libraries

[2]:
```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
```

1.3 New imports

[3]:
```
import statsmodels.api as sm
```

1.4 Import statsmodels objects

[4]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor as␣
 ↪VIF
from statsmodels.stats.anova import anova_lm
```

1.5 Import ISLP objects

[5]:
```
from ISLP import load_data
from ISLP.models import ModelSpec as MS, summarize, poly
```

1.6 Import User Functions

[6]:
```
from userfuncs import *
```

1.7 Inspecting objects and namespaces

[7]:
```
dir()
```

[7]:
```
['Audio',
 'In',
```

```
'InteractiveShell',
'Latex',
'MS',
'Markdown',
'Math',
'Out',
'VIF',
'_',
'__',
'___',
'__builtin__',
'__builtins__',
'__doc__',
'__loader__',
'__name__',
'__package__',
'__spec__',
'_dh',
'_i',
'_i1',
'_i2',
'_i3',
'_i4',
'_i5',
'_i6',
'_i7',
'_ih',
'_ii',
'_iii',
'_oh',
'allDone',
'anova_lm',
'calculate_VIFs',
'check_symmetric',
'display',
'display_DFFITS_plot',
'display_cooks_distance_plot',
'display_hat_leverage_cutoffs',
'display_hat_leverage_plot',
'display_residuals_plot',
'display_studentized_residuals',
'dmatrices',
'exit',
'get_influence_points',
'get_ipython',
'get_results_df',
'identify_highest_VIF_feature',
```

```
 'identify_least_significant_feature',
 'influence_plot',
 'is_numeric_dtype',
 'is_pos_def',
 'is_symmetric_pos_def',
 'load_data',
 'np',
 'open',
 'pd',
 'perform_analysis',
 'poly',
 'printlatex',
 'printmd',
 'px',
 'quit',
 'sm',
 'smf',
 'standardize',
 'stats',
 'subplots',
 'summarize']
```

[8]:
```
A = np.array([3, 5, 11])
dir(A)
```

[8]:
```
['T',
 '__abs__',
 '__add__',
 '__and__',
 '__array__',
 '__array_finalize__',
 '__array_function__',
 '__array_interface__',
 '__array_prepare__',
 '__array_priority__',
 '__array_struct__',
 '__array_ufunc__',
 '__array_wrap__',
 '__bool__',
 '__buffer__',
 '__class__',
 '__class_getitem__',
 '__complex__',
 '__contains__',
 '__copy__',
 '__deepcopy__',
 '__delattr__',
```

```
'__delitem__',
'__dir__',
'__divmod__',
'__dlpack__',
'__dlpack_device__',
'__doc__',
'__eq__',
'__float__',
'__floordiv__',
'__format__',
'__ge__',
'__getattribute__',
'__getitem__',
'__getstate__',
'__gt__',
'__hash__',
'__iadd__',
'__iand__',
'__ifloordiv__',
'__ilshift__',
'__imatmul__',
'__imod__',
'__imul__',
'__index__',
'__init__',
'__init_subclass__',
'__int__',
'__invert__',
'__ior__',
'__ipow__',
'__irshift__',
'__isub__',
'__iter__',
'__itruediv__',
'__ixor__',
'__le__',
'__len__',
'__lshift__',
'__lt__',
'__matmul__',
'__mod__',
'__mul__',
'__ne__',
'__neg__',
'__new__',
'__or__',
'__pos__',
```

```
'__pow__',
'__radd__',
'__rand__',
'__rdivmod__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rfloordiv__',
'__rlshift__',
'__rmatmul__',
'__rmod__',
'__rmul__',
'__ror__',
'__rpow__',
'__rrshift__',
'__rshift__',
'__rsub__',
'__rtruediv__',
'__rxor__',
'__setattr__',
'__setitem__',
'__setstate__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__truediv__',
'__xor__',
'all',
'any',
'argmax',
'argmin',
'argpartition',
'argsort',
'astype',
'base',
'byteswap',
'choose',
'clip',
'compress',
'conj',
'conjugate',
'copy',
'ctypes',
'cumprod',
'cumsum',
'data',
```

```
'diagonal',
'dot',
'dtype',
'dump',
'dumps',
'fill',
'flags',
'flat',
'flatten',
'getfield',
'imag',
'item',
'itemset',
'itemsize',
'max',
'mean',
'min',
'nbytes',
'ndim',
'newbyteorder',
'nonzero',
'partition',
'prod',
'ptp',
'put',
'ravel',
'real',
'repeat',
'reshape',
'resize',
'round',
'searchsorted',
'setfield',
'setflags',
'shape',
'size',
'sort',
'squeeze',
'std',
'strides',
'sum',
'swapaxes',
'take',
'tobytes',
'tofile',
'tolist',
'tostring',
```

```
 'trace',
 'transpose',
 'var',
 'view']
```

[9]:
```
A.sum()
```

[9]:
```
19
```

1.8 Simple Linear Regression

1.8.1 We will use the Boston housing dataset which is in the package ISLP

[10]:
```
Boston = load_data("Boston")
Boston.columns
```

[10]:
```
Index(['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',
       'ptratio', 'lstat', 'medv'],
      dtype='object')
```

[11]:
```
len(Boston.columns)
```

[11]:
```
13
```

[12]:
```
Boston?
```

1.8.2 Use sm.OLS to fit a simple linear regression

[13]:
```
X = pd.DataFrame({"intercept": np.ones(Boston.shape[0]), "lstat":␣
 ↪Boston["lstat"]})
X.head()
```

[13]:

	intercept	lstat
0	1.0	4.98
1	1.0	9.14
2	1.0	4.03
3	1.0	2.94
4	1.0	5.33

1.8.3 Extract the response and fit the model.

[14]:
```
y = Boston["medv"]
model = sm.OLS(y, X)
results = model.fit()
```

[14]:
```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x7671c599af60>
```

1.8.4 Summarize the results using the ISLP method summarize

[15]:
```
summarize(results)
```

[15]:

	coef	std err	t	P>|t|
intercept	34.5538	0.563	61.415	0.0
lstat	-0.9500	0.039	-24.528	0.0

1.9 Using Transformations: Fit and Transform

[16]:
```
design = MS(["lstat"])
design = design.fit(Boston)
X = design.transform(Boston)
X.head()
```

[16]:

	intercept	lstat
0	1.0	4.98
1	1.0	9.14
2	1.0	4.03
3	1.0	2.94
4	1.0	5.33

[17]:
```
design = MS(["lstat"])
design = design.fit_transform(Boston)
X.head()
```

[17]:

	intercept	lstat
0	1.0	4.98
1	1.0	9.14
2	1.0	4.03
3	1.0	2.94
4	1.0	5.33

1.9.1 Full and exhaustive summary of the fit

[18]:
```
results.summary()
```

[18]:

Dep. Variable:	medv	**R-squared:**	0.544
Model:	OLS	**Adj. R-squared:**	0.543
Method:	Least Squares	**F-statistic:**	601.6
Date:	Fri, 21 Feb 2025	**Prob (F-statistic):**	5.08e-88
Time:	15:49:43	**Log-Likelihood:**	-1641.5
No. Observations:	506	**AIC:**	3287.
Df Residuals:	504	**BIC:**	3295.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
intercept	34.5538	0.563	61.415	0.000	33.448	35.659
lstat	-0.9500	0.039	-24.528	0.000	-1.026	-0.874

Omnibus:	137.043	**Durbin-Watson:**	0.892
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	291.373
Skew:	1.453	**Prob(JB):**	5.36e-64
Kurtosis:	5.319	**Cond. No.**	29.7

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

1.9.2 Fitted coefficients can be retrieved as the *params* attribute of results

[19]:
```
results.params
```

[19]:
```
intercept    34.553841
lstat        -0.950049
dtype: float64
```

1.9.3 Computing predictions

[20]:
```
design = MS(["lstat"])
new_df = pd.DataFrame({"lstat": [5, 10, 15]})
print(new_df)
design = design.fit(new_df)
newX = design.transform(new_df)
newX
```

```
   lstat
0      5
1     10
2     15
```

[20]:

	intercept	lstat
0	1.0	5
1	1.0	10
2	1.0	15

[21]:
```
new_predictions = results.get_prediction(newX)
new_predictions.predicted_mean
```

[21]:
```
array([29.80359411, 25.05334734, 20.30310057])
```

1.9.4 We can predict confidence intervals for the predicted values.

[22]:
```
new_predictions.conf_int(alpha=0.05)
```

[22]:
```
array([[29.00741194, 30.59977628],
       [24.47413202, 25.63256267],
       [19.73158815, 20.87461299]])
```

1.9.5 We can obtain prediction intervals for the values which are wider than the confidence intervals since they're for a specific instance of lstat by setting obs=True.

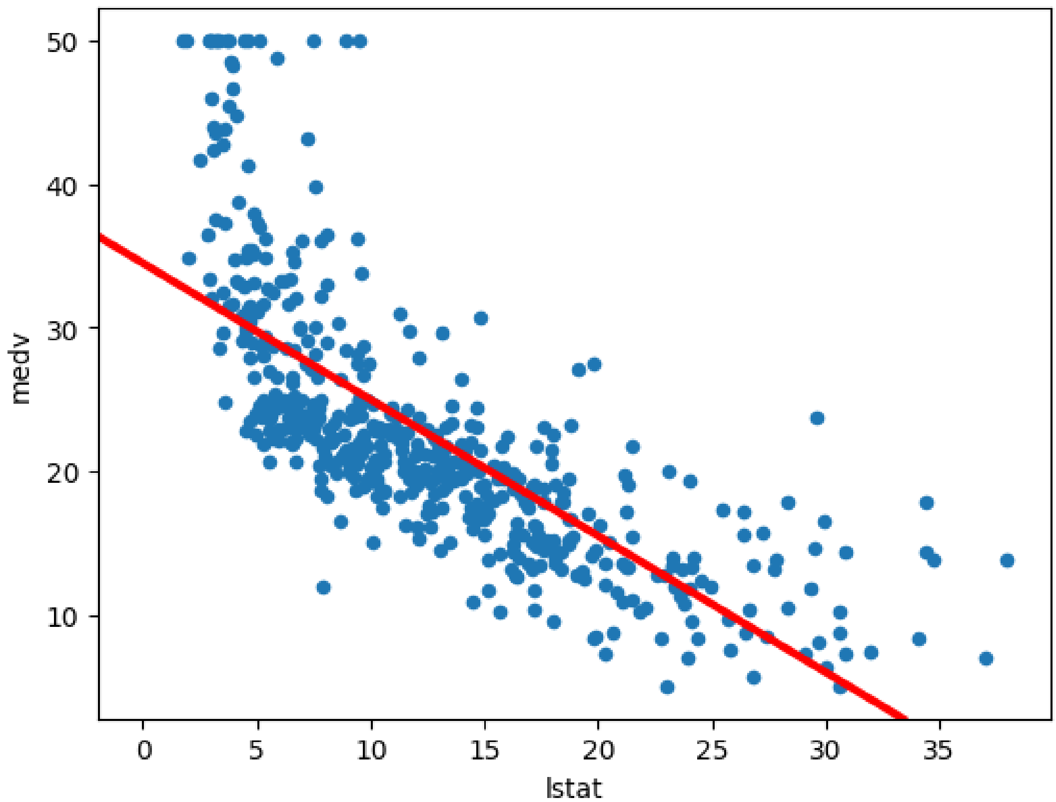
[23]:
```python
new_predictions.conf_int(obs=True, alpha=0.05)
```

[23]:
```
array([[17.56567478, 42.04151344],
       [12.82762635, 37.27906833],
       [ 8.0777421 , 32.52845905]])
```

1.9.6 Plot medv and lstat using DataFrame.plot.scatter() and add the regression line to the resulting plot.

[24]:
```python
ax = Boston.plot.scatter("lstat", "medv")
ax.axline(
    (ax.get_xlim()[0], results.params.iloc[0]),
    slope=results.params.iloc[1],
    color="r",
    linewidth=3,
)
```

[24]:
```
<matplotlib.lines.AxLine at 0x7671fcdac6b0>
```



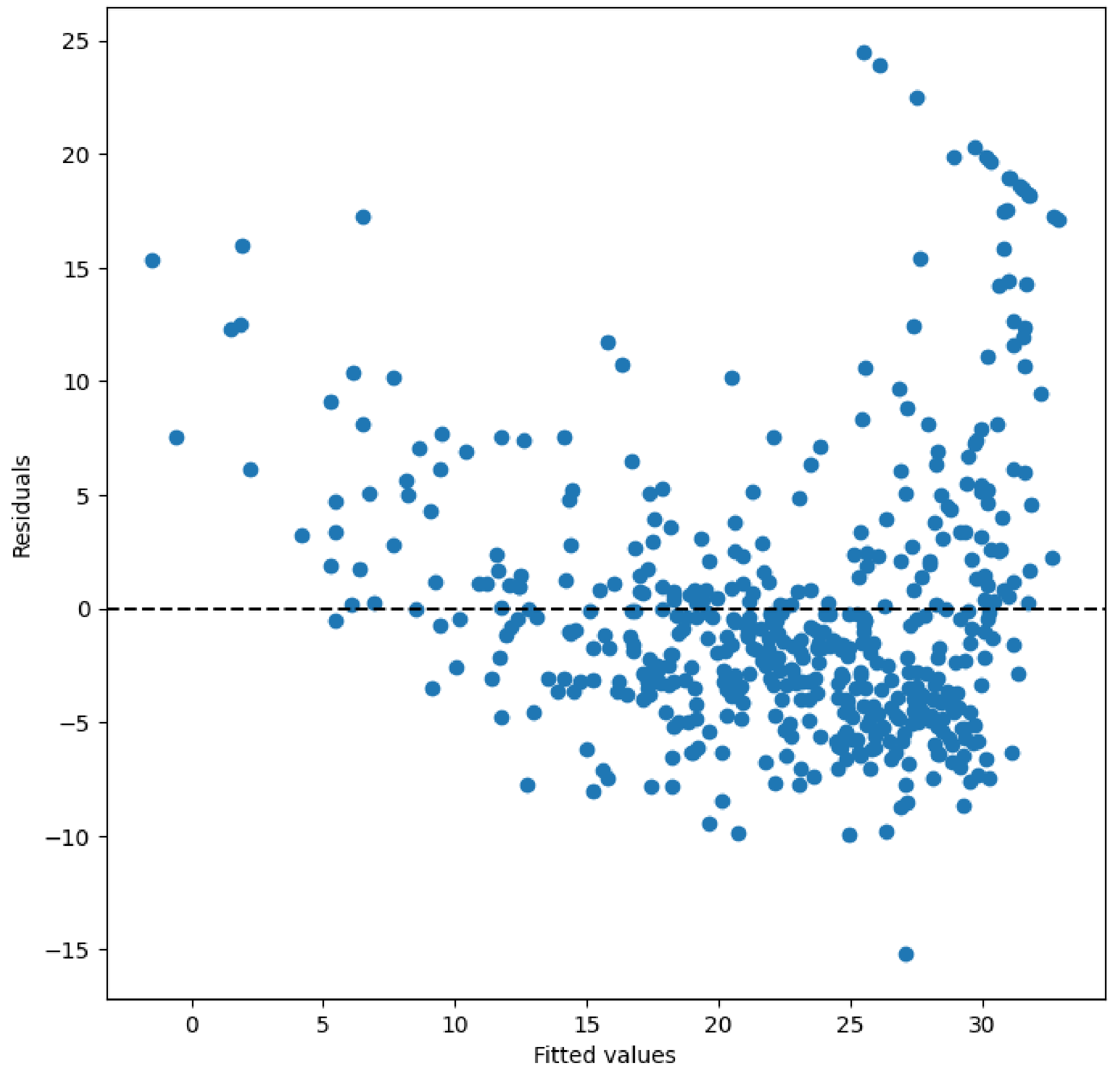
- There is some evidence of non-linearity in the relationship b/w lstat and medv.

1.9.7 Find the fitted values and residuals of the fit as attributes of the results object as *results.fittedvalues* and *results.resid*.

- The get_influence() method computes various influence measures of the regression.

[25]:
```
_, ax = subplots(figsize=(8, 8))
ax.scatter(results.fittedvalues, results.resid)
ax.set_xlabel("Fitted values")
ax.set_ylabel("Residuals")
ax.axhline(0, c="k", ls="--")
```

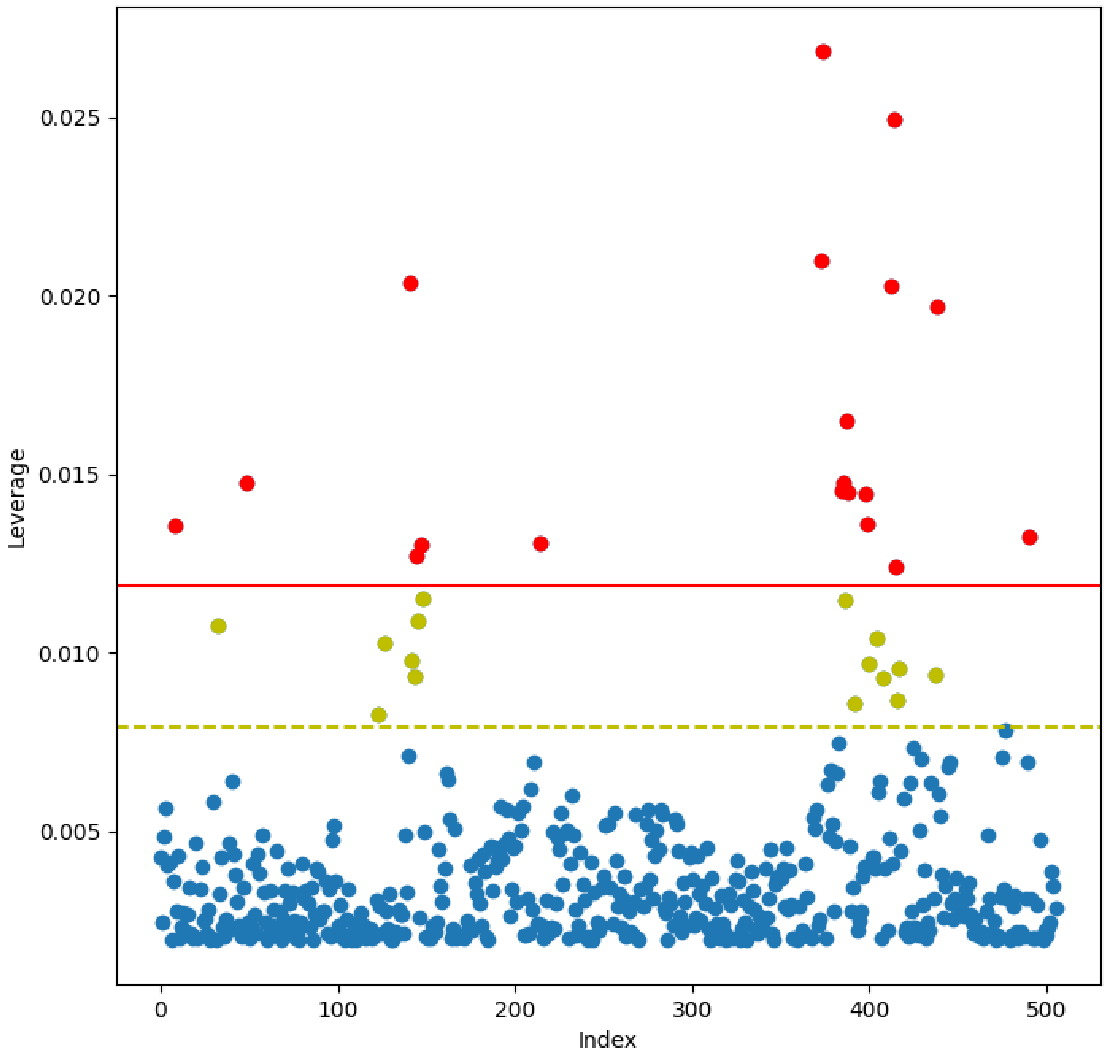
[25]:
```
<matplotlib.lines.Line2D at 0x7671aac54950>
```



- On the basis of the residual plot, there is some evidence of non-linearity.

1.9.8 Leverage statistics can be computed for any number of predictors using the hat_matrix_diag attribute of the value returned by the get_influence() method.

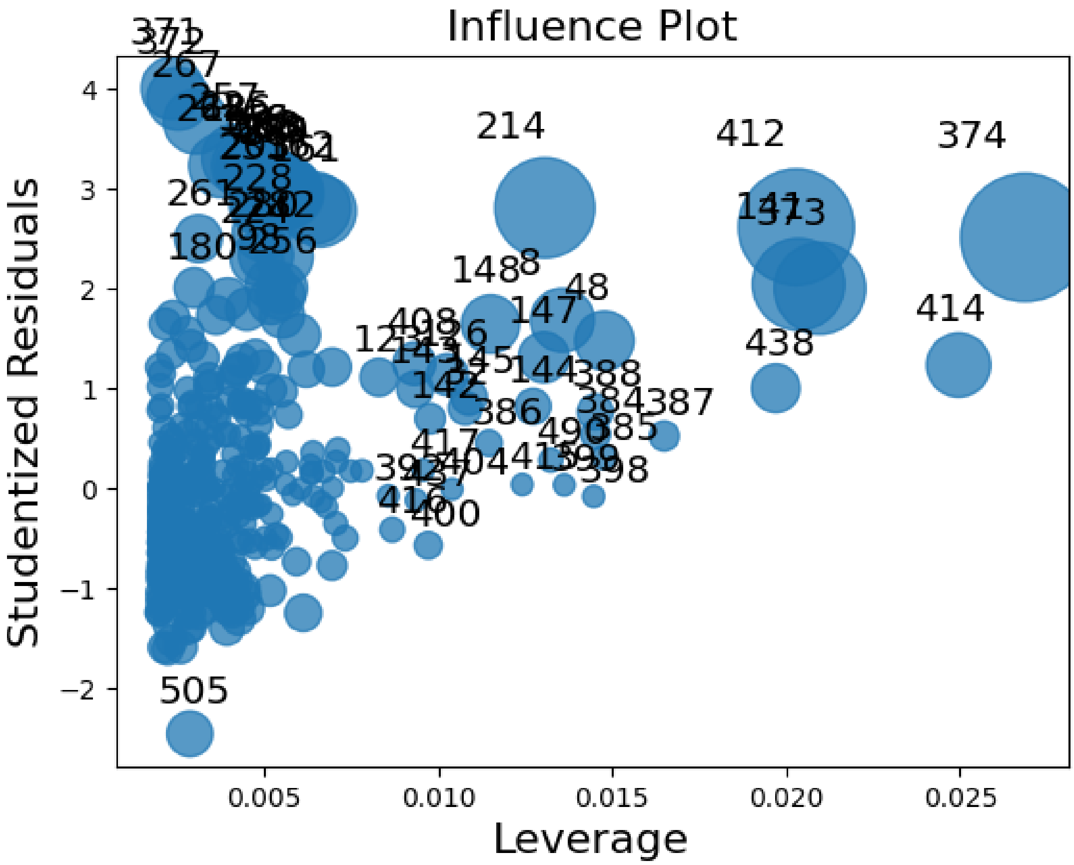
[26]:
```
display_hat_leverage_cutoffs(results)
```



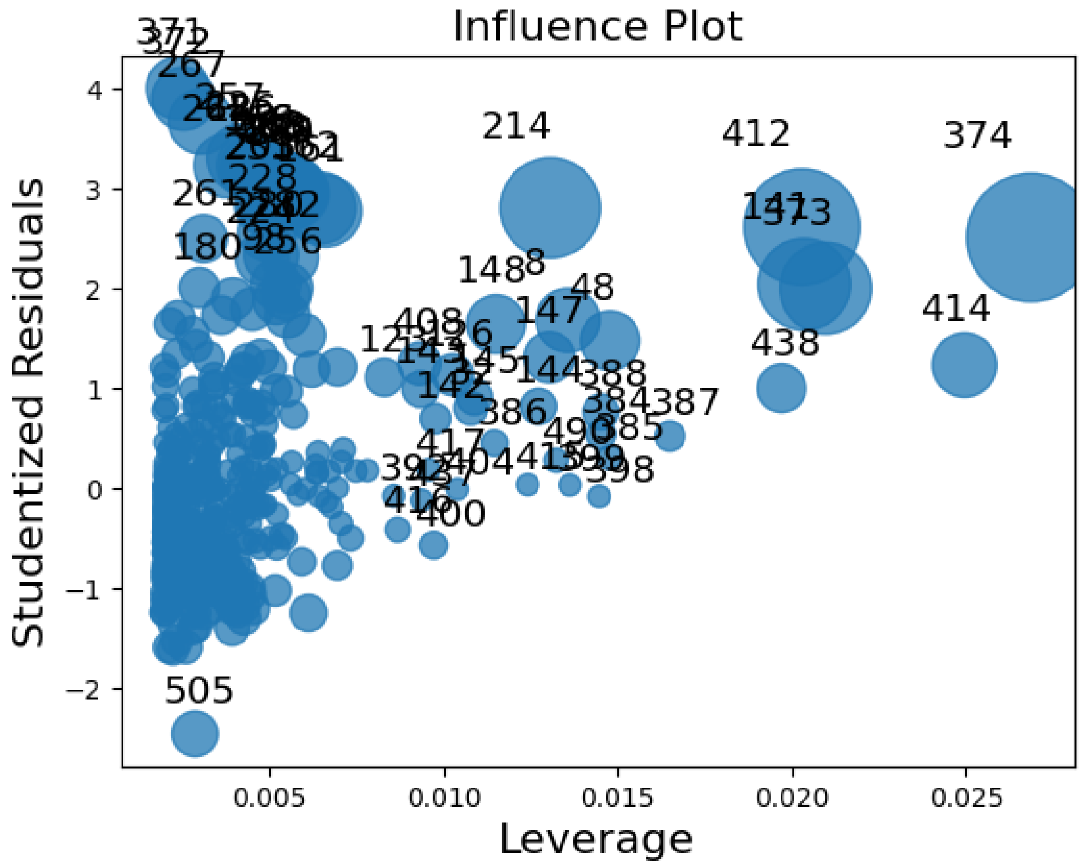
[27]:
```
display_hat_leverage_plot(results)
```

[28]:
```
display_cooks_distance_plot(results)
```

[28]:

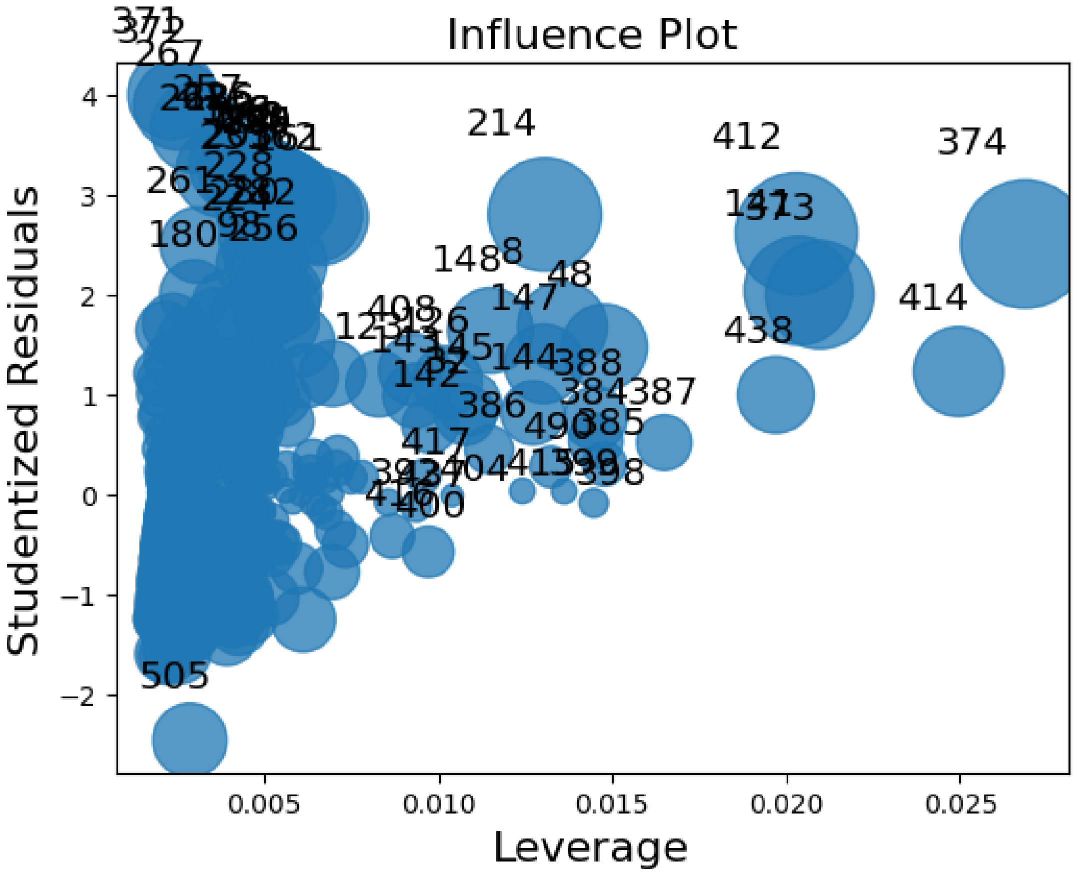


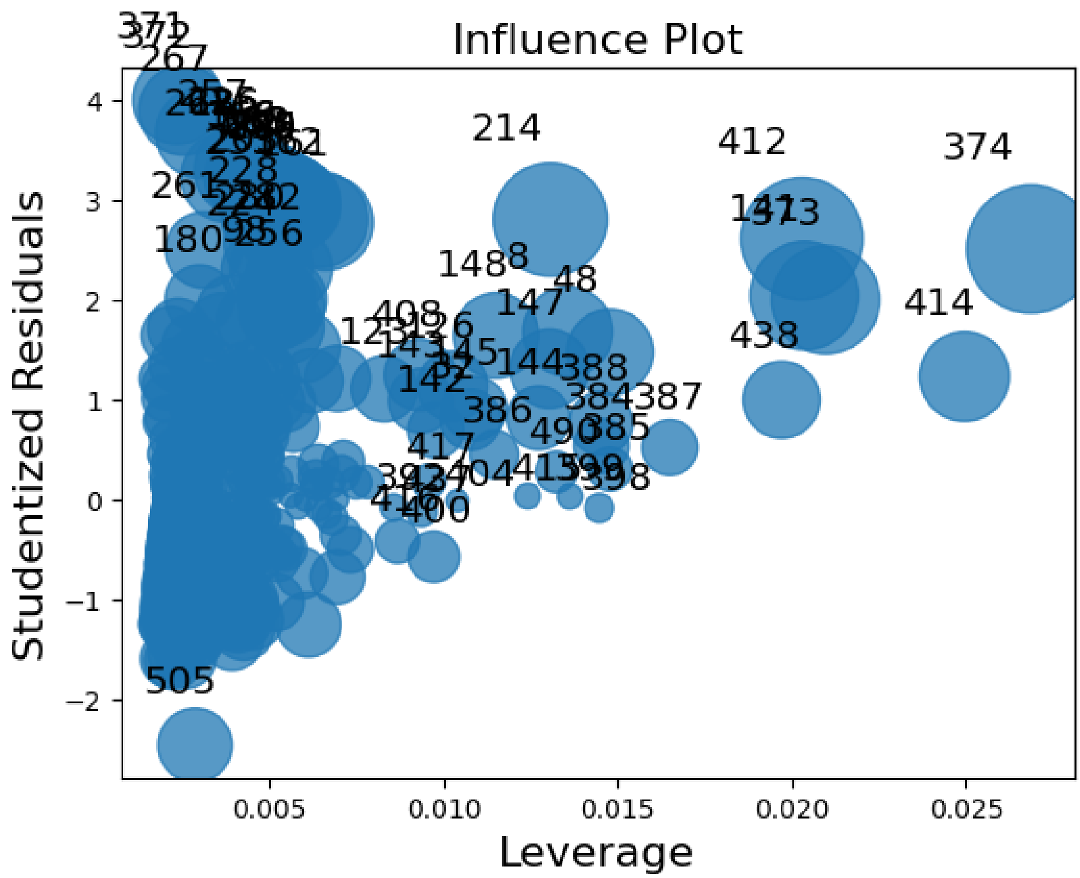
Influence Plot
Studentized Residuals
Leverage
4
3
2
1
0
−1
−2
0.005
0.010
0.015
0.020
0.025
214
412
374
414
438
505
148
147
48
387
400
398



[29]:
```
display_DFFITS_plot(results)
```

[29]:


Influence Plot
Studentized Residuals
Leverage
4
3
2
1
0
−1
−2
0.005
0.010
0.015
0.020
0.025
267
214
412
374
261
180
148
48
147
414
438
408
144
388
387
386
385
417
400
398
505



[30]:
```
inf_df, _ = get_influence_points(results)
inf_df
```

```
n = 506.0, p = 2
Average Hat Leverage: 0.003952569169960474
Hat Leverage  Cutoff = 2 * Average Hat Leverage = 0.007905138339920948
DFBetas Cutoff = 3 / sqrt(n) = 0.1333662673423161
DFFITS Cutoff = 2 * sqrt(p/n) = 0.1257389226923863
Cooks Distance Cutoff = 1.0
Cooks Distance p-value Cutoff = 0.05
Studentized Residuals Cutoff = 3.0
Studentized Residuals p-value Cutoff = 0.01
```

[30]:

	dfb_intercept	dfb_lstat	cooks_d	hat_diag	student_resid	dffits \
161	0.226022	-0.189640	0.025315	0.006609	2.776857	0.226503
162	0.225506	-0.188310	0.025219	0.006450	2.806416	0.226112
163	0.219862	-0.176384	0.024252	0.005359	3.024654	0.222011
166	0.217770	-0.172500	0.023921	0.005089	3.084034	0.220566
186	0.212939	-0.164025	0.023201	0.004589	3.201426	0.217377

195	0.221577	-0.179716	0.024534	0.005617	2.970018	0.223224
203	0.199749	-0.157618	0.020220	0.005013	2.853126	0.202515
204	0.221985	-0.180535	0.024602	0.005685	2.955977	0.223517
214	-0.197509	0.297576	0.051465	0.013063	2.807647	0.323011
225	0.211641	-0.161830	0.023017	0.004476	3.229640	0.216554
228	0.178647	-0.140414	0.016255	0.004938	2.573814	0.181309
233	0.196791	-0.154507	0.019680	0.004918	2.841931	0.199782
257	0.207834	-0.155541	0.022502	0.004180	3.306527	0.214221
261	0.128039	-0.084194	0.009646	0.003106	2.501385	0.139616
262	0.189166	-0.136023	0.019246	0.003742	3.231093	0.198020
267	0.184377	-0.119477	0.020008	0.003032	3.672331	0.202505
280	0.164168	-0.129770	0.013717	0.005047	2.335776	0.166365
283	0.220671	-0.177936	0.024384	0.005476	2.999671	0.222580
368	0.220170	-0.176971	0.024302	0.005402	3.015284	0.222227
369	0.217594	-0.172182	0.023894	0.005068	3.088724	0.220447
370	0.221623	-0.179808	0.024542	0.005625	2.968458	0.223257
371	0.155509	-0.078029	0.018381	0.002355	4.004703	0.194572
372	0.165278	-0.091836	0.018763	0.002529	3.901020	0.196431
374	-0.294291	0.401657	0.086162	0.026865	2.511537	0.417300
412	-0.253809	0.357605	0.070029	0.020290	2.615542	0.376405

	student_resid_pvalue	hat_influence	cooks_d_pvalue
161	0.002847	0.018353	0.975004
162	0.002602	0.018100	0.975097
163	0.001308	0.016208	0.976041
166	0.001077	0.015694	0.976364
186	0.000727	0.014692	0.977067
195	0.001560	0.016683	0.975766
203	0.002254	0.014302	0.979984
204	0.001632	0.016805	0.975699
214	0.002592	0.036676	0.949842
225	0.000660	0.014455	0.977247
228	0.005172	0.012709	0.983877
233	0.002333	0.013975	0.980513
257	0.000506	0.013821	0.977750
261	0.006344	0.007769	0.990401
262	0.000657	0.012090	0.980939
267	0.000133	0.011133	0.980191
280	0.009947	0.011789	0.986377
283	0.001418	0.016425	0.975912
368	0.001349	0.016290	0.975992
369	0.001061	0.015654	0.976391
370	0.001568	0.016697	0.975758
371	0.000036	0.009431	0.981788
372	0.000054	0.009866	0.981412
374	0.006166	0.067473	0.917459
412	0.004588	0.053070	0.932376

1.9.9 For a more conservative cutoff values for hat_diag, we have the following influence point(s):

[31]:
```
inf_df[inf_df["hat_diag"] > (3 * np.mean(inf_df["hat_diag"]))]
```

[31]:

	dfb_intercept	dfb_lstat	cooks_d	hat_diag	student_resid	dffits	\
374	-0.294291	0.401657	0.086162	0.026865	2.511537	0.417300	
412	-0.253809	0.357605	0.070029	0.020290	2.615542	0.376405	

	student_resid_pvalue	hat_influence	cooks_d_pvalue
374	0.006166	0.067473	0.917459
412	0.004588	0.053070	0.932376

1.9.10 Using DFFITS cutoff, we have the following influential points

[32]:
```
inf_df[inf_df["dffits"] > 2 * np.sqrt(len(results.params) / results.nobs)]
```

[32]:

	dfb_intercept	dfb_lstat	cooks_d	hat_diag	student_resid	dffits	\
161	0.226022	-0.189640	0.025315	0.006609	2.776857	0.226503	
162	0.225506	-0.188310	0.025219	0.006450	2.806416	0.226112	
163	0.219862	-0.176384	0.024252	0.005359	3.024654	0.222011	
166	0.217770	-0.172500	0.023921	0.005089	3.084034	0.220566	
186	0.212939	-0.164025	0.023201	0.004589	3.201426	0.217377	
195	0.221577	-0.179716	0.024534	0.005617	2.970018	0.223224	
203	0.199749	-0.157618	0.020220	0.005013	2.853126	0.202515	
204	0.221985	-0.180535	0.024602	0.005685	2.955977	0.223517	
214	-0.197509	0.297576	0.051465	0.013063	2.807647	0.323011	
225	0.211641	-0.161830	0.023017	0.004476	3.229640	0.216554	
228	0.178647	-0.140414	0.016255	0.004938	2.573814	0.181309	
233	0.196791	-0.154507	0.019680	0.004918	2.841931	0.199782	
257	0.207834	-0.155541	0.022502	0.004180	3.306527	0.214221	
261	0.128039	-0.084194	0.009646	0.003106	2.501385	0.139616	
262	0.189166	-0.136023	0.019246	0.003742	3.231093	0.198020	
267	0.184377	-0.119477	0.020008	0.003032	3.672331	0.202505	
280	0.164168	-0.129770	0.013717	0.005047	2.335776	0.166365	
283	0.220671	-0.177936	0.024384	0.005476	2.999671	0.222580	
368	0.220170	-0.176971	0.024302	0.005402	3.015284	0.222227	
369	0.217594	-0.172182	0.023894	0.005068	3.088724	0.220447	
370	0.221623	-0.179808	0.024542	0.005625	2.968458	0.223257	
371	0.155509	-0.078029	0.018381	0.002355	4.004703	0.194572	
372	0.165278	-0.091836	0.018763	0.002529	3.901020	0.196431	
374	-0.294291	0.401657	0.086162	0.026865	2.511537	0.417300	
412	-0.253809	0.357605	0.070029	0.020290	2.615542	0.376405	

	student_resid_pvalue	hat_influence	cooks_d_pvalue
161	0.002847	0.018353	0.975004
162	0.002602	0.018100	0.975097

163	0.001308	0.016208	0.976041
166	0.001077	0.015694	0.976364
186	0.000727	0.014692	0.977067
195	0.001560	0.016683	0.975766
203	0.002254	0.014302	0.979984
204	0.001632	0.016805	0.975699
214	0.002592	0.036676	0.949842
225	0.000660	0.014455	0.977247
228	0.005172	0.012709	0.983877
233	0.002333	0.013975	0.980513
257	0.000506	0.013821	0.977750
261	0.006344	0.007769	0.990401
262	0.000657	0.012090	0.980939
267	0.000133	0.011133	0.980191
280	0.009947	0.011789	0.986377
283	0.001418	0.016425	0.975912
368	0.001349	0.016290	0.975992
369	0.001061	0.015654	0.976391
370	0.001568	0.016697	0.975758
371	0.000036	0.009431	0.981788
372	0.000054	0.009866	0.981412
374	0.006166	0.067473	0.917459
412	0.004588	0.053070	0.932376

1.9.11 Using Cooks Distance, we have the following influential points

[33]:
```
inf_df[inf_df["cooks_d"] > 1.0]
```

[33]:
```
Empty DataFrame
Columns: [dfb_intercept, dfb_lstat, cooks_d, hat_diag, student_resid, dffits,
student_resid_pvalue, hat_influence, cooks_d_pvalue]
Index: []
```

1.9.12 Using Cooks Distance p-values, we have the following influential points

[34]:
```
inf_df[inf_df["cooks_d_pvalue"] < 0.05]
```

[34]:
```
Empty DataFrame
Columns: [dfb_intercept, dfb_lstat, cooks_d, hat_diag, student_resid, dffits,
student_resid_pvalue, hat_influence, cooks_d_pvalue]
Index: []
```

1.9.13 Using DFBeta for intercept, we have the following influential points

[35]:
```
inf_df[inf_df["dfb_intercept"] > (3 / np.sqrt(results.nobs))]
```

[35]:

	dfb_intercept	dfb_lstat	cooks_d	hat_diag	student_resid	dffits
161	0.226022	-0.189640	0.025315	0.006609	2.776857	0.226503
162	0.225506	-0.188310	0.025219	0.006450	2.806416	0.226112
163	0.219862	-0.176384	0.024252	0.005359	3.024654	0.222011
166	0.217770	-0.172500	0.023921	0.005089	3.084034	0.220566
186	0.212939	-0.164025	0.023201	0.004589	3.201426	0.217377
195	0.221577	-0.179716	0.024534	0.005617	2.970018	0.223224
203	0.199749	-0.157618	0.020220	0.005013	2.853126	0.202515
204	0.221985	-0.180535	0.024602	0.005685	2.955977	0.223517
225	0.211641	-0.161830	0.023017	0.004476	3.229640	0.216554
228	0.178647	-0.140414	0.016255	0.004938	2.573814	0.181309
233	0.196791	-0.154507	0.019680	0.004918	2.841931	0.199782
257	0.207834	-0.155541	0.022502	0.004180	3.306527	0.214221
262	0.189166	-0.136023	0.019246	0.003742	3.231093	0.198020
267	0.184377	-0.119477	0.020008	0.003032	3.672331	0.202505
280	0.164168	-0.129770	0.013717	0.005047	2.335776	0.166365
283	0.220671	-0.177936	0.024384	0.005476	2.999671	0.222580
368	0.220170	-0.176971	0.024302	0.005402	3.015284	0.222227
369	0.217594	-0.172182	0.023894	0.005068	3.088724	0.220447
370	0.221623	-0.179808	0.024542	0.005625	2.968458	0.223257
371	0.155509	-0.078029	0.018381	0.002355	4.004703	0.194572
372	0.165278	-0.091836	0.018763	0.002529	3.901020	0.196431

\

	student_resid_pvalue	hat_influence	cooks_d_pvalue
161	0.002847	0.018353	0.975004
162	0.002602	0.018100	0.975097
163	0.001308	0.016208	0.976041
166	0.001077	0.015694	0.976364
186	0.000727	0.014692	0.977067
195	0.001560	0.016683	0.975766
203	0.002254	0.014302	0.979984
204	0.001632	0.016805	0.975699
225	0.000660	0.014455	0.977247
228	0.005172	0.012709	0.983877
233	0.002333	0.013975	0.980513
257	0.000506	0.013821	0.977750
262	0.000657	0.012090	0.980939
267	0.000133	0.011133	0.980191
280	0.009947	0.011789	0.986377
283	0.001418	0.016425	0.975912
368	0.001349	0.016290	0.975992
369	0.001061	0.015654	0.976391
370	0.001568	0.016697	0.975758
371	0.000036	0.009431	0.981788
372	0.000054	0.009866	0.981412

1.9.14 Using DFBeta for lstat, we have the following influential points

[36]:
```
inf_df[inf_df["dfb_lstat"] > (3 / np.sqrt(results.nobs))]
```

[36]:

	dfb_intercept	dfb_lstat	cooks_d	hat_diag	student_resid	dffits
214	-0.197509	0.297576	0.051465	0.013063	2.807647	0.323011
374	-0.294291	0.401657	0.086162	0.026865	2.511537	0.417300
412	-0.253809	0.357605	0.070029	0.020290	2.615542	0.376405

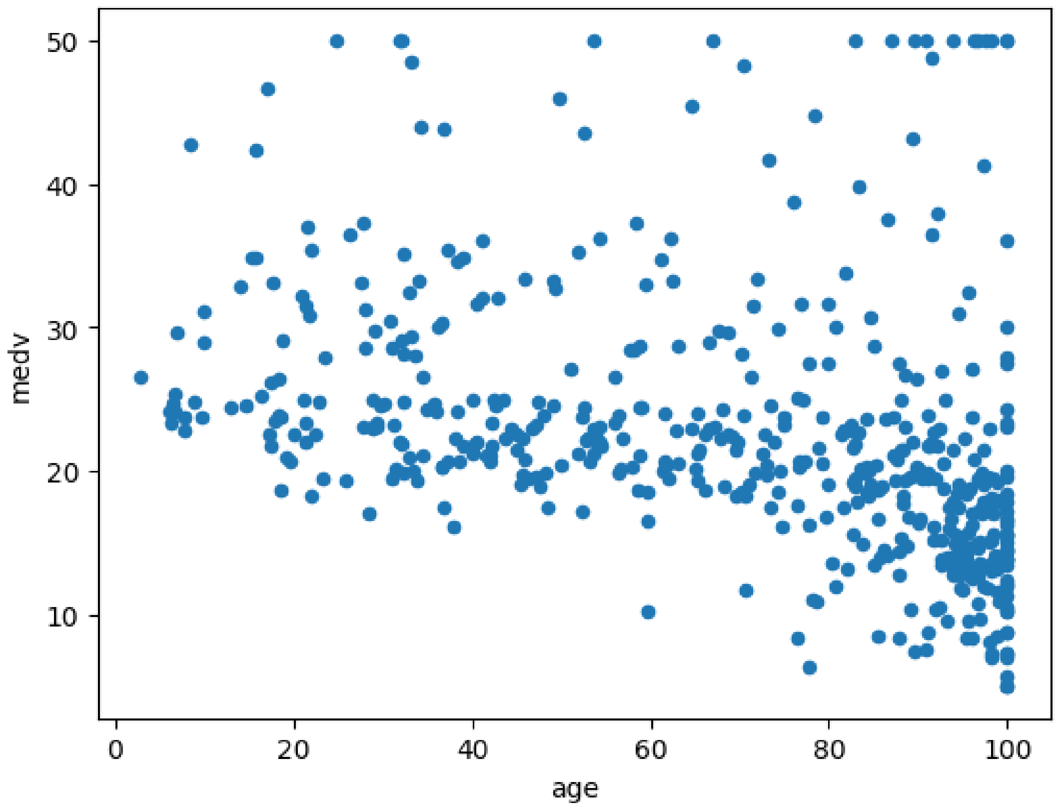
	student_resid_pvalue	hat_influence	cooks_d_pvalue
214	0.002592	0.036676	0.949842
374	0.006166	0.067473	0.917459
412	0.004588	0.053070	0.932376

1.9.15 Multiple linear regression

[37]:
```
Boston.plot.scatter("age", "medv")
X = MS(["lstat", "age"]).fit_transform(Boston)
model1 = sm.OLS(y, X)
results1 = model1.fit()
summarize(results1)
```

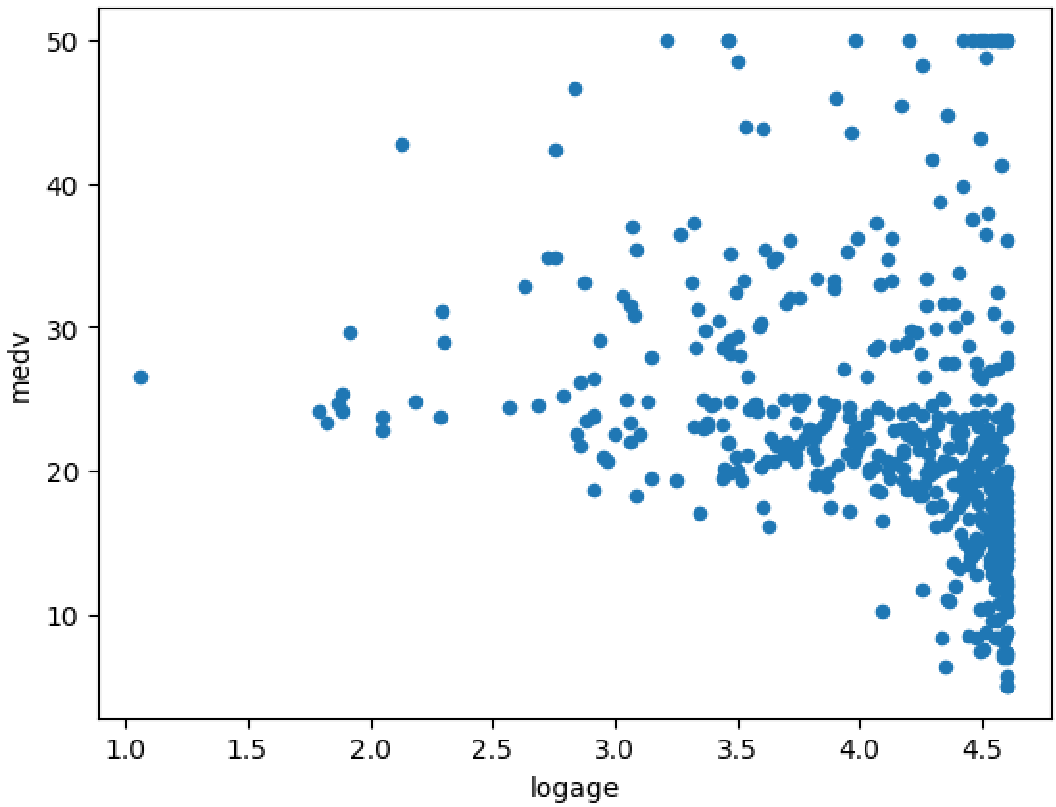
[37]:

	coef	std err	t	P>|t|
intercept	33.2228	0.731	45.458	0.000
lstat	-1.0321	0.048	-21.416	0.000
age	0.0345	0.012	2.826	0.005



[38]:
```
Boston["logage"] = np.log(Boston["age"])
Boston.plot.scatter("logage", "medv")
X = MS(["lstat", "logage"]).fit_transform(Boston)
model1 = sm.OLS(y, X)
resultslog = model1.fit()
print(summarize(resultslog))
```

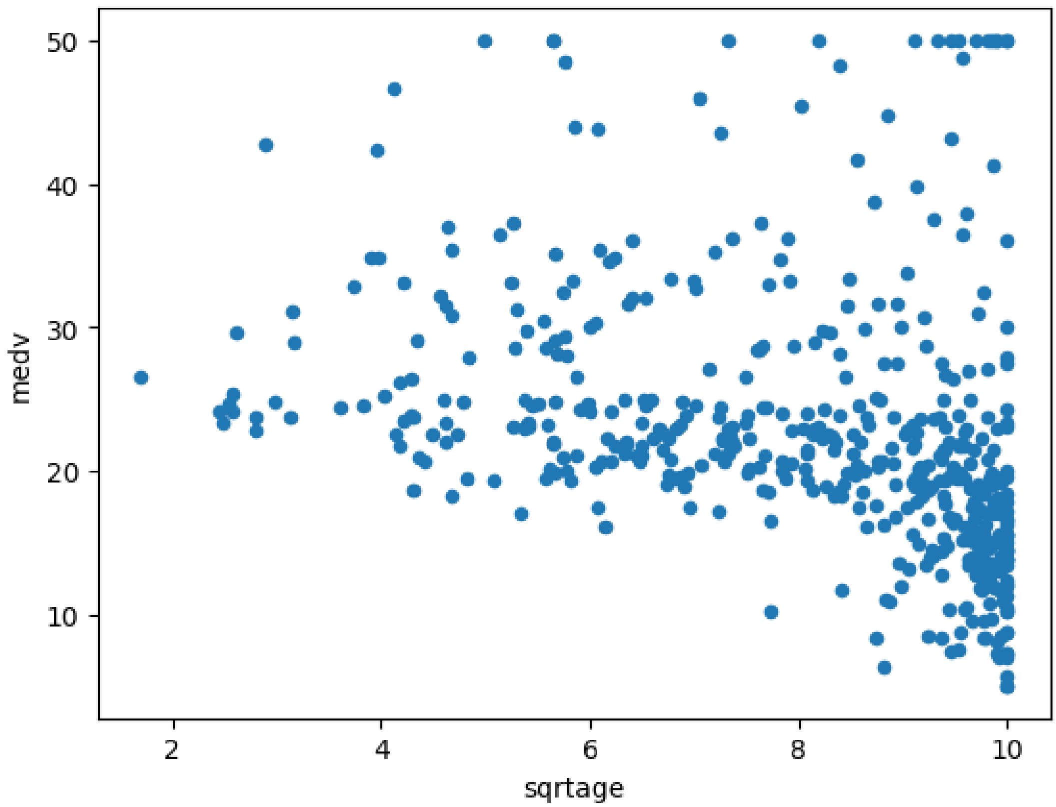
```
               coef  std err       t  P>|t|
intercept   30.2143    1.947  15.517   0.00
lstat       -1.0051    0.045 -22.213   0.00
logage       1.2312    0.529   2.327   0.02
```



[39]:
```
Boston["sqrtage"] = np.sqrt(Boston["age"])
Boston.plot.scatter("sqrtage", "medv")
X = MS(["lstat", "sqrtage"]).fit_transform(Boston)
model1 = sm.OLS(y, X)
resultssqrt = model1.fit()
summarize(resultssqrt)
```

[39]:

	coef	std err	t	P>|t|
intercept	31.8635	1.174	27.139	0.000
lstat	-1.0203	0.047	-21.703	0.000
sqrtage	0.4450	0.171	2.606	0.009



[40]:
```
Boston = Boston.drop(columns=["logage", "sqrtage"])
```

[40]:
```
        crim    zn  indus  chas    nox     rm   age     dis  rad  tax  \
0    0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222
..       …     …     …     …     …      …     …      …    …    …
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273

     ptratio  lstat  medv
0       15.3   4.98  24.0
1       17.8   9.14  21.6
2       17.8   4.03  34.7
3       18.7   2.94  33.4
4       18.7   5.33  36.2
```

```
..        ...    ...   ...
501       21.0   9.67  22.4
502       21.0   9.08  20.6
503       21.0   5.64  23.9
504       21.0   6.48  22.0
505       21.0   7.88  11.9

[506 rows x 13 columns]
```

[41]:
```
terms = Boston.columns.drop("medv")
terms
```

[41]:
```
Index(['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',
       'ptratio', 'lstat'],
      dtype='object')
```

[42]:
```
X = MS(terms).fit_transform(Boston)
model = sm.OLS(y, X)
results = model.fit()
summarize(results)
```

[42]:

	coef	std err	t	P>|t|
intercept	41.6173	4.936	8.431	0.000
crim	-0.1214	0.033	-3.678	0.000
zn	0.0470	0.014	3.384	0.001
indus	0.0135	0.062	0.217	0.829
chas	2.8400	0.870	3.264	0.001
nox	-18.7580	3.851	-4.870	0.000
rm	3.6581	0.420	8.705	0.000
age	0.0036	0.013	0.271	0.787
dis	-1.4908	0.202	-7.394	0.000
rad	0.2894	0.067	4.325	0.000
tax	-0.0127	0.004	-3.337	0.001
ptratio	-0.9375	0.132	-7.091	0.000
lstat	-0.5520	0.051	-10.897	0.000

- Age has a high p-value. So how about we drop it from the predictors?

[43]:
```
minus_age = Boston.columns.drop(["medv", "age"])
Xma = MS(minus_age).fit_transform(Boston)
model1 = sm.OLS(y, Xma)
summarize(model1.fit())
```

[43]:

	coef	std err	t	P>|t|
intercept	41.5251	4.920	8.441	0.000
crim	-0.1214	0.033	-3.683	0.000
zn	0.0465	0.014	3.379	0.001
indus	0.0135	0.062	0.217	0.829

```
chas         2.8528    0.868   3.287  0.001
nox        -18.4851    3.714  -4.978  0.000
rm           3.6811    0.411   8.951  0.000
dis         -1.5068    0.193  -7.825  0.000
rad          0.2879    0.067   4.322  0.000
tax         -0.0127    0.004  -3.333  0.001
ptratio     -0.9346    0.132  -7.099  0.000
lstat       -0.5474    0.048 -11.483  0.000
```

[44]:
```
np.unique(Boston["indus"])
```

[44]:
```
array([ 0.46,  0.74,  1.21,  1.22,  1.25,  1.32,  1.38,  1.47,  1.52,
        1.69,  1.76,  1.89,  1.91,  2.01,  2.02,  2.03,  2.18,  2.24,
        2.25,  2.31,  2.46,  2.68,  2.89,  2.93,  2.95,  2.97,  3.24,
        3.33,  3.37,  3.41,  3.44,  3.64,  3.75,  3.78,  3.97,  4.  ,
        4.05,  4.15,  4.39,  4.49,  4.86,  4.93,  4.95,  5.13,  5.19,
        5.32,  5.64,  5.86,  5.96,  6.06,  6.07,  6.09,  6.2 ,  6.41,
        6.91,  6.96,  7.07,  7.38,  7.87,  8.14,  8.56,  9.69,  9.9 ,
       10.01, 10.59, 10.81, 11.93, 12.83, 13.89, 13.92, 15.04, 18.1 ,
       19.58, 21.89, 25.65, 27.74])
```

Similarly, indus has a high p-value. Let's drop it as well. minus_age_indus = Boston.columns.drop(["medv", "age", "indus"]) Xmai = MS(minus_age_indus).fit_transform(Boston) model1 = sm.OLS(y, Xmai) results1 = model1.fit() summarize(results1)

We can also observe the F-statistic for the regression.

[45]:
```
(results1.fvalue, results1.f_pvalue)
```

[45]:
```
(308.9693351215988, 2.9820335524722154e-88)
```

1.9.16 Multivariate Goodness of Fit

1.9.17 We can access the individual components of results by name.

[46]:
```
dir(results1)
```

[46]:
```
['HC0_se',
 'HC1_se',
 'HC2_se',
 'HC3_se',
 '_HCCM',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
```

```
'__format__',
'__ge__',
'__getattribute__',
'__getstate__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__',
'_abat_diagonal',
'_cache',
'_data_attr',
'_data_in_cache',
'_get_robustcov_results',
'_get_wald_nonlinear',
'_is_nested',
'_transform_predict_exog',
'_use_t',
'_wexog_singular_values',
'aic',
'bic',
'bse',
'centered_tss',
'compare_f_test',
'compare_lm_test',
'compare_lr_test',
'condition_number',
'conf_int',
'conf_int_el',
'cov_HC0',
'cov_HC1',
'cov_HC2',
'cov_HC3',
'cov_kwds',
'cov_params',
```

```
'cov_type',
'df_model',
'df_resid',
'diagn',
'eigenvals',
'el_test',
'ess',
'f_pvalue',
'f_test',
'fittedvalues',
'fvalue',
'get_influence',
'get_prediction',
'get_robustcov_results',
'info_criteria',
'initialize',
'k_constant',
'llf',
'load',
'model',
'mse_model',
'mse_resid',
'mse_total',
'nobs',
'normalized_cov_params',
'outlier_test',
'params',
'predict',
'pvalues',
'remove_data',
'resid',
'resid_pearson',
'rsquared',
'rsquared_adj',
'save',
'scale',
'ssr',
'summary',
'summary2',
't_test',
't_test_pairwise',
'tvalues',
'uncentered_tss',
'use_t',
'wald_test',
'wald_test_terms',
'wresid']
```

- results.rsquared gives us the R2 and np.sqrt(results.scale) gives us the RSE.

[47]:
```
print("RSE", np.sqrt(results1.scale))
```

```
RSE 6.173136281359115
```

[48]:
```
("R", results1.rsquared)
```

[48]:
```
('R', 0.5512689379421002)
```

- Variance Inflation Factors are sometimes useful to assess the collinearity effect in our regression model.

1.9.18 Compute VIFs and List Comprehension

[49]:
```
vals = [VIF(X, i) for i in range(1, X.shape[1])]
print(vals)
```

```
[1.7674859154310127, 2.2984589077358097, 3.9871806307570994, 1.071167773758404,
4.369092622844793, 1.9125324374368868, 3.0882320397311966, 3.954036641628298,
7.445300760069838, 9.002157663471797, 1.7970595931297786, 2.8707765008417514]
```

[50]:
```
vif = pd.DataFrame({"vif": vals}, index=X.columns[1:])
print(vif)
("VIF Range:", np.min(vif), np.max(vif))
```

```
              vif
crim     1.767486
zn       2.298459
indus    3.987181
chas     1.071168
nox      4.369093
rm       1.912532
age      3.088232
dis      3.954037
rad      7.445301
tax      9.002158
ptratio  1.797060
lstat    2.870777
```

[50]:
```
('VIF Range:', 1.071167773758404, 9.002157663471797)
```

- The VIFs are not very large.

1.9.19 Interaction terms

[51]:
```
X = MS(["lstat", "age", ("lstat", "age")]).fit_transform(Boston)
model2 = sm.OLS(y, X)
results2 = model2.fit()
summarize(results2)
```

[51]:
```
                 coef  std err        t  P>|t|
intercept     36.0885    1.470   24.553  0.000
lstat         -1.3921    0.167   -8.313  0.000
age           -0.0007    0.020   -0.036  0.971
lstat:age      0.0042    0.002    2.244  0.025
```

[52]:
```
(results2.rsquared, " > ", results1.rsquared)
```

[52]:
```
(0.5557265450993936, ' > ', 0.5512689379421002)
```

- The interaction terms lstat:age are not statistically significant at 0.01 level of significance, and R2 does not significantly explain the variation in the model. Suffice to say, the interaction term can be dropped.

1.9.20 Non-linear transformation of the predictors

- The poly() function specifies the first argument term to be added to the model matrix

[53]:
```
X = MS([poly("lstat", degree=2), "age"]).fit_transform(Boston)
model3 = sm.OLS(y, X)
results3 = model3.fit()
summarize(results3)
```

[53]:
```
                              coef  std err        t  P>|t|
intercept                  17.7151    0.781   22.681    0.0
poly(lstat, degree=2)[0] -179.2279    6.733  -26.620    0.0
poly(lstat, degree=2)[1]   72.9908    5.482   13.315    0.0
age                         0.0703    0.011    6.471    0.0
```

The effectively 0 p-value associated with the quadratic term suggests an improved model. The R2 confirms it

[54]:
```
print(results3.rsquared, " > ", results2.rsquared)
```

```
0.6683791720749932  >  0.5557265450993936
```

- By default, poly() creates a basis matrix for inclusion in the model matrix whose columns are orthogonal polynomials which are designed for stable least squares computations. If we had included another argument, raw = True , the basis matrix would consist of lstat and lstat ** 2. Both represent quadratic polynomials. The fitted values would not change. Just the polynomial coefficients. The columns created by poly() do not include an intercept column. These are provided by MS().

1.9.21 Questions:

- What are orthogonal polynomials?
- http://home.iitk.ac.in/~shalab/regression/Chapter12-Regression-PolynomialRegression.pdf
- https://stats.stackexchange.com/questions/258307/raw-or-orthogonal-polynomial-regression

[55]:
```python
X = MS([poly("lstat", degree=2, raw=True), "age"]).fit_transform(Boston)
model3 = sm.OLS(y, X)
results3 = model3.fit()
summarize(results3)
```

[55]:

	coef	std err	t	P>|t|
intercept	41.2885	0.873	47.284	0.0
poly(lstat, degree=2, raw=True)[0]	-2.6883	0.131	-20.502	0.0
poly(lstat, degree=2, raw=True)[1]	0.0495	0.004	13.315	0.0
age	0.0703	0.011	6.471	0.0

[56]:
```python
print(results3.rsquared, " > ", results1.rsquared)
```

```
0.6683791720749932  >  0.5512689379421002
```

- Use the anova_lm() function to further quantify the superiority of the quadratic fit.

[57]:
```python
anova_lm(results1, results3)
```

[57]:

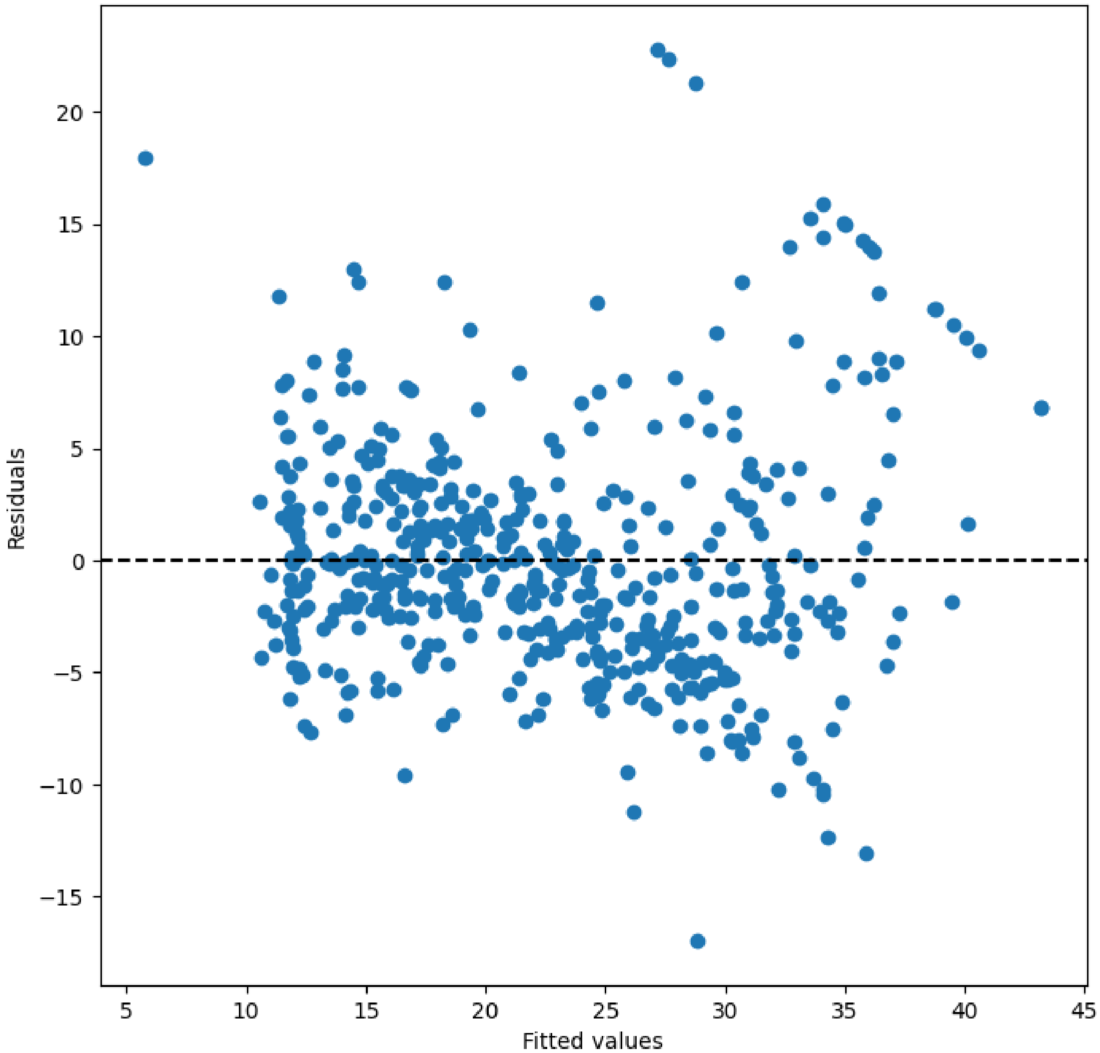
	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	503.0	19168.128609	0.0	NaN	NaN	NaN
1	502.0	14165.613251	1.0	5002.515357	177.278785	7.468491e-35

- results1 corresponds to the linear model containing predictors lstat and age only.
- results3 includes the quadratic term in lstat.
- The anova_lm() function performs a hypothesis test on the two models.
- H0: The quadratic term in the model is not needed.
- Ha: The larger model including the quadratic term is superior.
- Here, the F-statistic is 177.28 and the associated p-value is 0.
- The F-statistic is the t-statistic squared for the quadratic term in results3.
- These nested models differ by 1 degree of freedom.
- This provides very clear evidence that the quadratic term improves the model.
- The anova_lm() function can take more than two models as input.
- The comparison is successive pair-wise.
- That explains the NaNs in the first row of the output above, since there is no previous model with which to compare the output.

1.9.22 We can further plot the residuals of the regression against the fitted values to check of there still is a pattern discernible.

[58]:
```python
_, ax = subplots(figsize=(8, 8))
ax.scatter(results3.fittedvalues, results3.resid)
ax.set_xlabel("Fitted values")
ax.set_ylabel("Residuals")
ax.axhline(0, c="k", ls="--")
```

[58]:
```
<matplotlib.lines.Line2D at 0x7671a9e8b680>
```



1.9.23 We can also try and add the interaction term (lstat, age) to the regression and check the results.

[59]:
```
X = MS([poly("lstat", degree=2, raw=True), "age", ("lstat", "age")]).
 ↪fit_transform(
    Boston
)
model4 = sm.OLS(y, X)
results4 = model4.fit()
summarize(results4)
```

[59]:

	coef	std err	t	P>|t|
intercept	37.2658	1.250	29.816	0.0
poly(lstat, degree=2, raw=True)[0]	-2.2980	0.156	-14.723	0.0

```
poly(lstat, degree=2, raw=True)[1]    0.0584    0.004  14.015    0.0
age                                   0.1439    0.020   7.279    0.0
lstat:age                            -0.0079    0.002  -4.424    0.0
```

[60]:
```python
print(results4.rsquared, " > ", results3.rsquared)
```

```
0.6808467217930462  >  0.6683791720749932
```

- The R2 in the interaction model again does not exceedingly explain the variance in the model compared to simply having the quadratic term.

1.9.24 Qualitative Predictors

1.10 Carseats data

[61]:
```python
Carseats = load_data("Carseats")
Carseats.columns
```

[61]:
```
Index(['Sales', 'CompPrice', 'Income', 'Advertising', 'Population', 'Price',
       'ShelveLoc', 'Age', 'Education', 'Urban', 'US'],
      dtype='object')
```

[62]:
```python
Carseats.shape
```

[62]:
```
(400, 11)
```

[63]:
```python
Carseats.describe()
```

[63]:
```
            Sales   CompPrice      Income  Advertising  Population  \
count  400.000000  400.000000  400.000000   400.000000  400.000000
mean     7.496325  124.975000   68.657500     6.635000  264.840000
std      2.824115   15.334512   27.986037     6.650364  147.376436
min      0.000000   77.000000   21.000000     0.000000   10.000000
25%      5.390000  115.000000   42.750000     0.000000  139.000000
50%      7.490000  125.000000   69.000000     5.000000  272.000000
75%      9.320000  135.000000   91.000000    12.000000  398.500000
max     16.270000  175.000000  120.000000    29.000000  509.000000

            Price         Age   Education
count  400.000000  400.000000  400.000000
mean   115.795000   53.322500   13.900000
std     23.676664   16.200297    2.620528
min     24.000000   25.000000   10.000000
25%    100.000000   39.750000   12.000000
50%    117.000000   54.500000   14.000000
75%    131.000000   66.000000   16.000000
max    191.000000   80.000000   18.000000
```

- ModelSpec() generates dummy variables for categorical columns automatically. This is termed a one-hot encoding of the categorical feature.

- Their columns sum to one. To avoid collinearity with the intercept, the first column is dropped.

1.10.1 Below we fit a multiple regression model with interaction terms.

[64]:
```python
allvars = list(Carseats.columns.drop("Sales"))
y = Carseats["Sales"]
final = allvars + [("Income", "Advertising"), ("Price", "Age")]
X = MS(final).fit_transform(Carseats)
model = sm.OLS(y, X)
summarize(model.fit())
```

[64]:

	coef	std err	t	P>|t|
intercept	6.5756	1.009	6.519	0.000
CompPrice	0.0929	0.004	22.567	0.000
Income	0.0109	0.003	4.183	0.000
Advertising	0.0702	0.023	3.107	0.002
Population	0.0002	0.000	0.433	0.665
Price	-0.1008	0.007	-13.549	0.000
ShelveLoc[Good]	4.8487	0.153	31.724	0.000
ShelveLoc[Medium]	1.9533	0.126	15.531	0.000
Age	-0.0579	0.016	-3.633	0.000
Education	-0.0209	0.020	-1.063	0.288
Urban[Yes]	0.1402	0.112	1.247	0.213
US[Yes]	-0.1576	0.149	-1.058	0.291
Income:Advertising	0.0008	0.000	2.698	0.007
Price:Age	0.0001	0.000	0.801	0.424

- It can be seen that ShelvLoc is significant and a good shelving location is associated with high sales (relative to a bad location). Medium has a smaller coefficient than Good leading us to believe that it leads to higher sales than a bad location, but lesser than a good location.

[65]:
```python
allDone()
```

```
<IPython.lib.display.Audio object>
```