ISLP_36

February 25, 2024

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

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor as␣
↪VIF
from statsmodels.stats.anova import anova_lm

from ISLP import load_data
from ISLP.models import (ModelSpec as MS, summarize, poly)
```

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

[ ]:
```
['In',
 'MS',
 'Out',
 'VIF',
 '_',
 '__',
 '___',
 '__builtin__',
 '__builtins__',
 '__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '__vsc_ipynb_file__',
 '_dh',
 '_i',
 '_i1',
 '_i2',
 '_i3',
 '_ih',
 '_ii',
```

```
 '_iii',
 '_oh',
 'anova_lm',
 'exit',
 'get_ipython',
 'load_data',
 'np',
 'open',
 'pd',
 'poly',
 'quit',
 'sm',
 'subplots',
 'summarize']
```

[ ]:
```
A = np.array([1, 3, 5])
dir(A)
```

[ ]:
```
['T',
 '__abs__',
 '__add__',
 '__and__',
 '__array__',
 '__array_finalize__',
 '__array_function__',
 '__array_interface__',
 '__array_prepare__',
 '__array_priority__',
 '__array_struct__',
 '__array_ufunc__',
 '__array_wrap__',
 '__bool__',
 '__class__',
 '__class_getitem__',
 '__complex__',
 '__contains__',
 '__copy__',
 '__deepcopy__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__divmod__',
 '__dlpack__',
 '__dlpack_device__',
 '__doc__',
 '__eq__',
 '__float__',
```

```
'__floordiv__',
'__format__',
'__ge__',
'__getattribute__',
'__getitem__',
'__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']
```

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

[ ]:
```
9
```

Simple Linear Regression

[ ]:
```
Boston = load_data("Boston")
```

[ ]:
```
Boston.columns
```

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

[ ]:
```
Boston?
```

```
Type:            DataFrame
String form:
crim    zn  indus  chas    nox     rm   age     dis  rad  tax  \
           0    0.00632  18.0   2.3 <…> 0   5.64  23.9
           504     21.0   6.48  22.0
           505     21.0   7.88  11.9

           [506 rows x 13 columns]
Length:          506
File:            ~/Documents/CS/CS543_ML/mlvenv/lib/python3.10/site-
packages/pandas/core/frame.py
Docstring:
Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
pandas data structure.

Parameters
----------
data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame
    Dict can contain Series, arrays, constants, dataclass or list-like objects.
If
    data is a dict, column order follows insertion-order. If a dict contains
Series
    which have an index defined, it is aligned by its index.

    .. versionchanged:: 0.25.0
       If data is a list of dicts, column order follows insertion-order.

index : Index or array-like
    Index to use for resulting frame. Will default to RangeIndex if
    no indexing information part of input data and no index provided.
columns : Index or array-like
    Column labels to use for resulting frame when data does not have them,
```

```
    defaulting to RangeIndex(0, 1, 2, …, n). If data contains column labels,
    will perform column selection instead.
dtype : dtype, default None
    Data type to force. Only a single dtype is allowed. If None, infer.
copy : bool or None, default None
    Copy data from inputs.
    For dict data, the default of None behaves like ``copy=True``.  For
DataFrame
    or 2d ndarray input, the default of None behaves like ``copy=False``.
    If data is a dict containing one or more Series (possibly of different
dtypes),
    ``copy=False`` will ensure that these inputs are not copied.

    .. versionchanged:: 1.3.0

See Also
--------
DataFrame.from_records : Constructor from tuples, also record arrays.
DataFrame.from_dict : From dicts of Series, arrays, or dicts.
read_csv : Read a comma-separated values (csv) file into DataFrame.
read_table : Read general delimited file into DataFrame.
read_clipboard : Read text from clipboard into DataFrame.

Notes
-----
Please reference the :ref:`User Guide <basics.dataframe>` for more information.

Examples
--------
Constructing DataFrame from a dictionary.

>>> d = {'col1': [1, 2], 'col2': [3, 4]}
>>> df = pd.DataFrame(data=d)
>>> df
   col1  col2
0     1     3
1     2     4

Notice that the inferred dtype is int64.

>>> df.dtypes
col1    int64
col2    int64
dtype: object

To enforce a single dtype:

>>> df = pd.DataFrame(data=d, dtype=np.int8)
```

```
>>> df.dtypes
col1    int8
col2    int8
dtype: object
```

Constructing DataFrame from a dictionary including Series:

```
>>> d = {'col1': [0, 1, 2, 3], 'col2': pd.Series([2, 3], index=[2, 3])}
>>> pd.DataFrame(data=d, index=[0, 1, 2, 3])
   col1  col2
0     0   NaN
1     1   NaN
2     2   2.0
3     3   3.0
```

Constructing DataFrame from numpy ndarray:

```
>>> df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
…                      columns=['a', 'b', 'c'])
>>> df2
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

Constructing DataFrame from a numpy ndarray that has labeled columns:

```
>>> data = np.array([(1, 2, 3), (4, 5, 6), (7, 8, 9)],
…                   dtype=[("a", "i4"), ("b", "i4"), ("c", "i4")])
>>> df3 = pd.DataFrame(data, columns=['c', 'a'])
…
>>> df3
   c  a
0  3  1
1  6  4
2  9  7
```

Constructing DataFrame from dataclass:

```
>>> from dataclasses import make_dataclass
>>> Point = make_dataclass("Point", [("x", int), ("y", int)])
>>> pd.DataFrame([Point(0, 0), Point(0, 3), Point(2, 3)])
   x  y
0  0  0
1  0  3
2  2  3
```

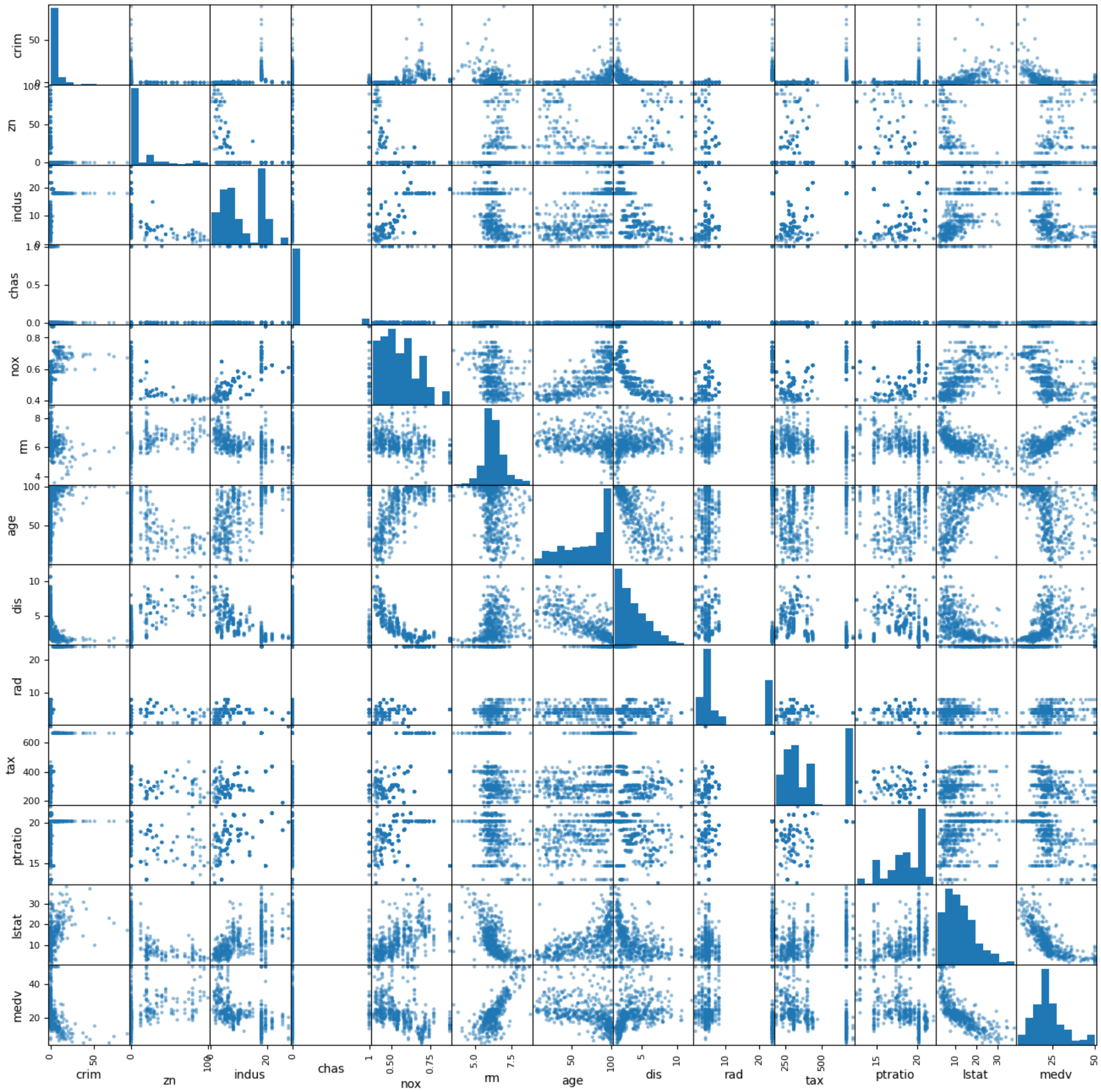
[ ]:
```
pd.plotting.scatter_matrix(Boston, figsize=(16,16))
```

[ ]:
```
array([[<Axes: xlabel='crim', ylabel='crim'>,
        <Axes: xlabel='zn', ylabel='crim'>,
        <Axes: xlabel='indus', ylabel='crim'>,
        <Axes: xlabel='chas', ylabel='crim'>,
        <Axes: xlabel='nox', ylabel='crim'>,
        <Axes: xlabel='rm', ylabel='crim'>,
        <Axes: xlabel='age', ylabel='crim'>,
        <Axes: xlabel='dis', ylabel='crim'>,
        <Axes: xlabel='rad', ylabel='crim'>,
        <Axes: xlabel='tax', ylabel='crim'>,
        <Axes: xlabel='ptratio', ylabel='crim'>,
        <Axes: xlabel='lstat', ylabel='crim'>,
        <Axes: xlabel='medv', ylabel='crim'>],
       [<Axes: xlabel='crim', ylabel='zn'>,
        <Axes: xlabel='zn', ylabel='zn'>,
        <Axes: xlabel='indus', ylabel='zn'>,
        <Axes: xlabel='chas', ylabel='zn'>,
        <Axes: xlabel='nox', ylabel='zn'>,
        <Axes: xlabel='rm', ylabel='zn'>,
        <Axes: xlabel='age', ylabel='zn'>,
        <Axes: xlabel='dis', ylabel='zn'>,
        <Axes: xlabel='rad', ylabel='zn'>,
        <Axes: xlabel='tax', ylabel='zn'>,
        <Axes: xlabel='ptratio', ylabel='zn'>,
        <Axes: xlabel='lstat', ylabel='zn'>,
        <Axes: xlabel='medv', ylabel='zn'>],
       [<Axes: xlabel='crim', ylabel='indus'>,
        <Axes: xlabel='zn', ylabel='indus'>,
        <Axes: xlabel='indus', ylabel='indus'>,
        <Axes: xlabel='chas', ylabel='indus'>,
        <Axes: xlabel='nox', ylabel='indus'>,
        <Axes: xlabel='rm', ylabel='indus'>,
        <Axes: xlabel='age', ylabel='indus'>,
        <Axes: xlabel='dis', ylabel='indus'>,
        <Axes: xlabel='rad', ylabel='indus'>,
        <Axes: xlabel='tax', ylabel='indus'>,
        <Axes: xlabel='ptratio', ylabel='indus'>,
        <Axes: xlabel='lstat', ylabel='indus'>,
        <Axes: xlabel='medv', ylabel='indus'>],
       [<Axes: xlabel='crim', ylabel='chas'>,
        <Axes: xlabel='zn', ylabel='chas'>,
        <Axes: xlabel='indus', ylabel='chas'>,
        <Axes: xlabel='chas', ylabel='chas'>,
        <Axes: xlabel='nox', ylabel='chas'>,
        <Axes: xlabel='rm', ylabel='chas'>,
```

```
 <Axes: xlabel='age', ylabel='chas'>,
 <Axes: xlabel='dis', ylabel='chas'>,
 <Axes: xlabel='rad', ylabel='chas'>,
 <Axes: xlabel='tax', ylabel='chas'>,
 <Axes: xlabel='ptratio', ylabel='chas'>,
 <Axes: xlabel='lstat', ylabel='chas'>,
 <Axes: xlabel='medv', ylabel='chas'>],
[<Axes: xlabel='crim', ylabel='nox'>,
 <Axes: xlabel='zn', ylabel='nox'>,
 <Axes: xlabel='indus', ylabel='nox'>,
 <Axes: xlabel='chas', ylabel='nox'>,
 <Axes: xlabel='nox', ylabel='nox'>,
 <Axes: xlabel='rm', ylabel='nox'>,
 <Axes: xlabel='age', ylabel='nox'>,
 <Axes: xlabel='dis', ylabel='nox'>,
 <Axes: xlabel='rad', ylabel='nox'>,
 <Axes: xlabel='tax', ylabel='nox'>,
 <Axes: xlabel='ptratio', ylabel='nox'>,
 <Axes: xlabel='lstat', ylabel='nox'>,
 <Axes: xlabel='medv', ylabel='nox'>],
[<Axes: xlabel='crim', ylabel='rm'>,
 <Axes: xlabel='zn', ylabel='rm'>,
 <Axes: xlabel='indus', ylabel='rm'>,
 <Axes: xlabel='chas', ylabel='rm'>,
 <Axes: xlabel='nox', ylabel='rm'>,
 <Axes: xlabel='rm', ylabel='rm'>,
 <Axes: xlabel='age', ylabel='rm'>,
 <Axes: xlabel='dis', ylabel='rm'>,
 <Axes: xlabel='rad', ylabel='rm'>,
 <Axes: xlabel='tax', ylabel='rm'>,
 <Axes: xlabel='ptratio', ylabel='rm'>,
 <Axes: xlabel='lstat', ylabel='rm'>,
 <Axes: xlabel='medv', ylabel='rm'>],
[<Axes: xlabel='crim', ylabel='age'>,
 <Axes: xlabel='zn', ylabel='age'>,
 <Axes: xlabel='indus', ylabel='age'>,
 <Axes: xlabel='chas', ylabel='age'>,
 <Axes: xlabel='nox', ylabel='age'>,
 <Axes: xlabel='rm', ylabel='age'>,
 <Axes: xlabel='age', ylabel='age'>,
 <Axes: xlabel='dis', ylabel='age'>,
 <Axes: xlabel='rad', ylabel='age'>,
 <Axes: xlabel='tax', ylabel='age'>,
 <Axes: xlabel='ptratio', ylabel='age'>,
 <Axes: xlabel='lstat', ylabel='age'>,
 <Axes: xlabel='medv', ylabel='age'>],
[<Axes: xlabel='crim', ylabel='dis'>,
```

```
 <Axes: xlabel='zn', ylabel='dis'>,
 <Axes: xlabel='indus', ylabel='dis'>,
 <Axes: xlabel='chas', ylabel='dis'>,
 <Axes: xlabel='nox', ylabel='dis'>,
 <Axes: xlabel='rm', ylabel='dis'>,
 <Axes: xlabel='age', ylabel='dis'>,
 <Axes: xlabel='dis', ylabel='dis'>,
 <Axes: xlabel='rad', ylabel='dis'>,
 <Axes: xlabel='tax', ylabel='dis'>,
 <Axes: xlabel='ptratio', ylabel='dis'>,
 <Axes: xlabel='lstat', ylabel='dis'>,
 <Axes: xlabel='medv', ylabel='dis'>],
[<Axes: xlabel='crim', ylabel='rad'>,
 <Axes: xlabel='zn', ylabel='rad'>,
 <Axes: xlabel='indus', ylabel='rad'>,
 <Axes: xlabel='chas', ylabel='rad'>,
 <Axes: xlabel='nox', ylabel='rad'>,
 <Axes: xlabel='rm', ylabel='rad'>,
 <Axes: xlabel='age', ylabel='rad'>,
 <Axes: xlabel='dis', ylabel='rad'>,
 <Axes: xlabel='rad', ylabel='rad'>,
 <Axes: xlabel='tax', ylabel='rad'>,
 <Axes: xlabel='ptratio', ylabel='rad'>,
 <Axes: xlabel='lstat', ylabel='rad'>,
 <Axes: xlabel='medv', ylabel='rad'>],
[<Axes: xlabel='crim', ylabel='tax'>,
 <Axes: xlabel='zn', ylabel='tax'>,
 <Axes: xlabel='indus', ylabel='tax'>,
 <Axes: xlabel='chas', ylabel='tax'>,
 <Axes: xlabel='nox', ylabel='tax'>,
 <Axes: xlabel='rm', ylabel='tax'>,
 <Axes: xlabel='age', ylabel='tax'>,
 <Axes: xlabel='dis', ylabel='tax'>,
 <Axes: xlabel='rad', ylabel='tax'>,
 <Axes: xlabel='tax', ylabel='tax'>,
 <Axes: xlabel='ptratio', ylabel='tax'>,
 <Axes: xlabel='lstat', ylabel='tax'>,
 <Axes: xlabel='medv', ylabel='tax'>],
[<Axes: xlabel='crim', ylabel='ptratio'>,
 <Axes: xlabel='zn', ylabel='ptratio'>,
 <Axes: xlabel='indus', ylabel='ptratio'>,
 <Axes: xlabel='chas', ylabel='ptratio'>,
 <Axes: xlabel='nox', ylabel='ptratio'>,
 <Axes: xlabel='rm', ylabel='ptratio'>,
 <Axes: xlabel='age', ylabel='ptratio'>,
 <Axes: xlabel='dis', ylabel='ptratio'>,
 <Axes: xlabel='rad', ylabel='ptratio'>,
```

```
 <Axes: xlabel='tax', ylabel='ptratio'>,
 <Axes: xlabel='ptratio', ylabel='ptratio'>,
 <Axes: xlabel='lstat', ylabel='ptratio'>,
 <Axes: xlabel='medv', ylabel='ptratio'>],
[<Axes: xlabel='crim', ylabel='lstat'>,
 <Axes: xlabel='zn', ylabel='lstat'>,
 <Axes: xlabel='indus', ylabel='lstat'>,
 <Axes: xlabel='chas', ylabel='lstat'>,
 <Axes: xlabel='nox', ylabel='lstat'>,
 <Axes: xlabel='rm', ylabel='lstat'>,
 <Axes: xlabel='age', ylabel='lstat'>,
 <Axes: xlabel='dis', ylabel='lstat'>,
 <Axes: xlabel='rad', ylabel='lstat'>,
 <Axes: xlabel='tax', ylabel='lstat'>,
 <Axes: xlabel='ptratio', ylabel='lstat'>,
 <Axes: xlabel='lstat', ylabel='lstat'>,
 <Axes: xlabel='medv', ylabel='lstat'>],
[<Axes: xlabel='crim', ylabel='medv'>,
 <Axes: xlabel='zn', ylabel='medv'>,
 <Axes: xlabel='indus', ylabel='medv'>,
 <Axes: xlabel='chas', ylabel='medv'>,
 <Axes: xlabel='nox', ylabel='medv'>,
 <Axes: xlabel='rm', ylabel='medv'>,
 <Axes: xlabel='age', ylabel='medv'>,
 <Axes: xlabel='dis', ylabel='medv'>,
 <Axes: xlabel='rad', ylabel='medv'>,
 <Axes: xlabel='tax', ylabel='medv'>,
 <Axes: xlabel='ptratio', ylabel='medv'>,
 <Axes: xlabel='lstat', ylabel='medv'>,
 <Axes: xlabel='medv', ylabel='medv'>]], dtype=object)
```




```
[ ]: X = pd.DataFrame({'intercept': np.ones(Boston.shape[0]), 'lstat':␣
      ↪Boston['lstat']})
     X[:10]
```

```
[ ]:    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
     5        1.0   5.21
     6        1.0  12.43
     7        1.0  19.15
```

```
8         1.0  29.93
9         1.0  17.10
```

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

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

[ ]:

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

[ ]:
```
design = MS(['lstat'])
design = design.fit(Boston)
X = design.transform(Boston)
X[:10]
```

[ ]:

	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
5	1.0	5.21
6	1.0	12.43
7	1.0	19.15
8	1.0	29.93
9	1.0	17.10

[ ]:
```
design = MS(['lstat'])
X = design.fit_transform(Boston)
X[:10]
```

[ ]:

	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
5	1.0	5.21
6	1.0	12.43
7	1.0	19.15
8	1.0	29.93
9	1.0	17.10

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

[ ]:

Dep. Variable:	medv	**R-squared:**	0.544
Model:	OLS	**Adj. R-squared:**	0.543
Method:	Least Squares	**F-statistic:**	601.6
Date:	Sun, 25 Feb 2024	**Prob (F-statistic):**	5.08e-88
Time:	13:49:19	**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.

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

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

[ ]:
```
new_df = pd.DataFrame({'lstat':[5,10,15]})
newX = design.transform(new_df)
newX
```

[ ]:
```
   intercept  lstat
0        1.0      5
1        1.0     10
2        1.0     15
```

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

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

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

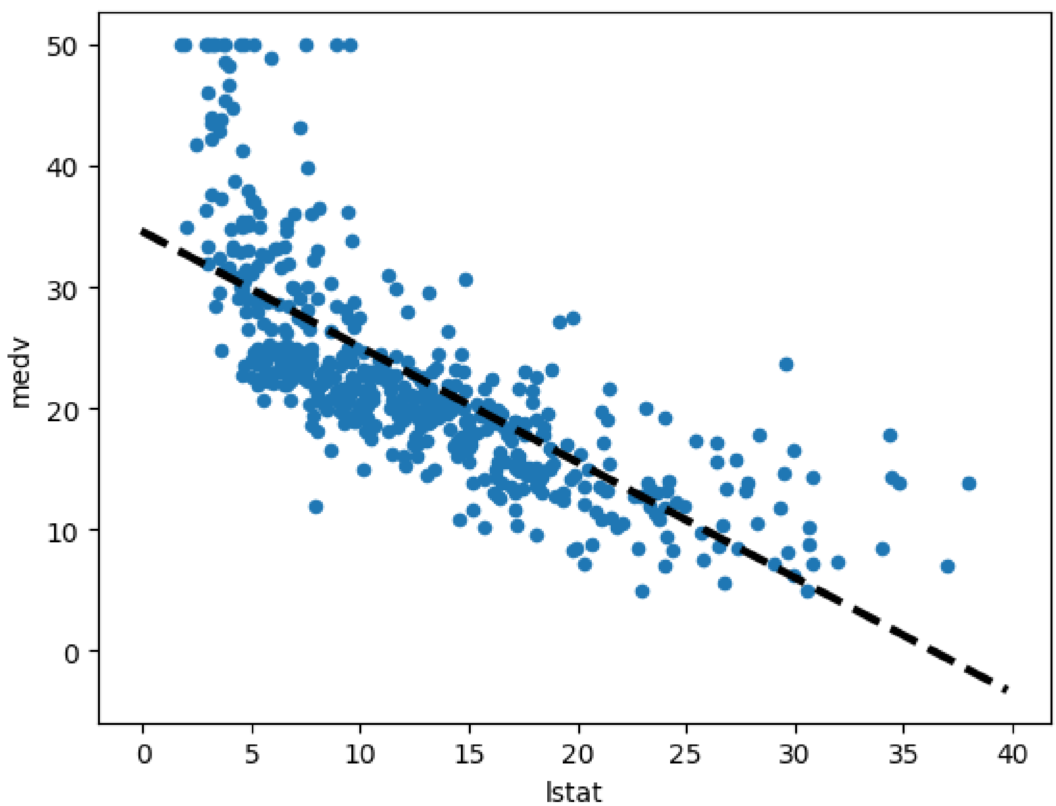
[ ]:
```
array([[29.00741194, 30.59977628],
       [24.47413202, 25.63256267],
       [19.73158815, 20.87461299]])
```

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

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

[ ]:
```
def abline(ax, b, m,  *args, **kwargs):
    xlim = ax.get_xlim()
    ylim = [m * xlim[0] + b, m * xlim[1] + b]
    ax.plot(xlim, ylim, *args, **kwargs)
```

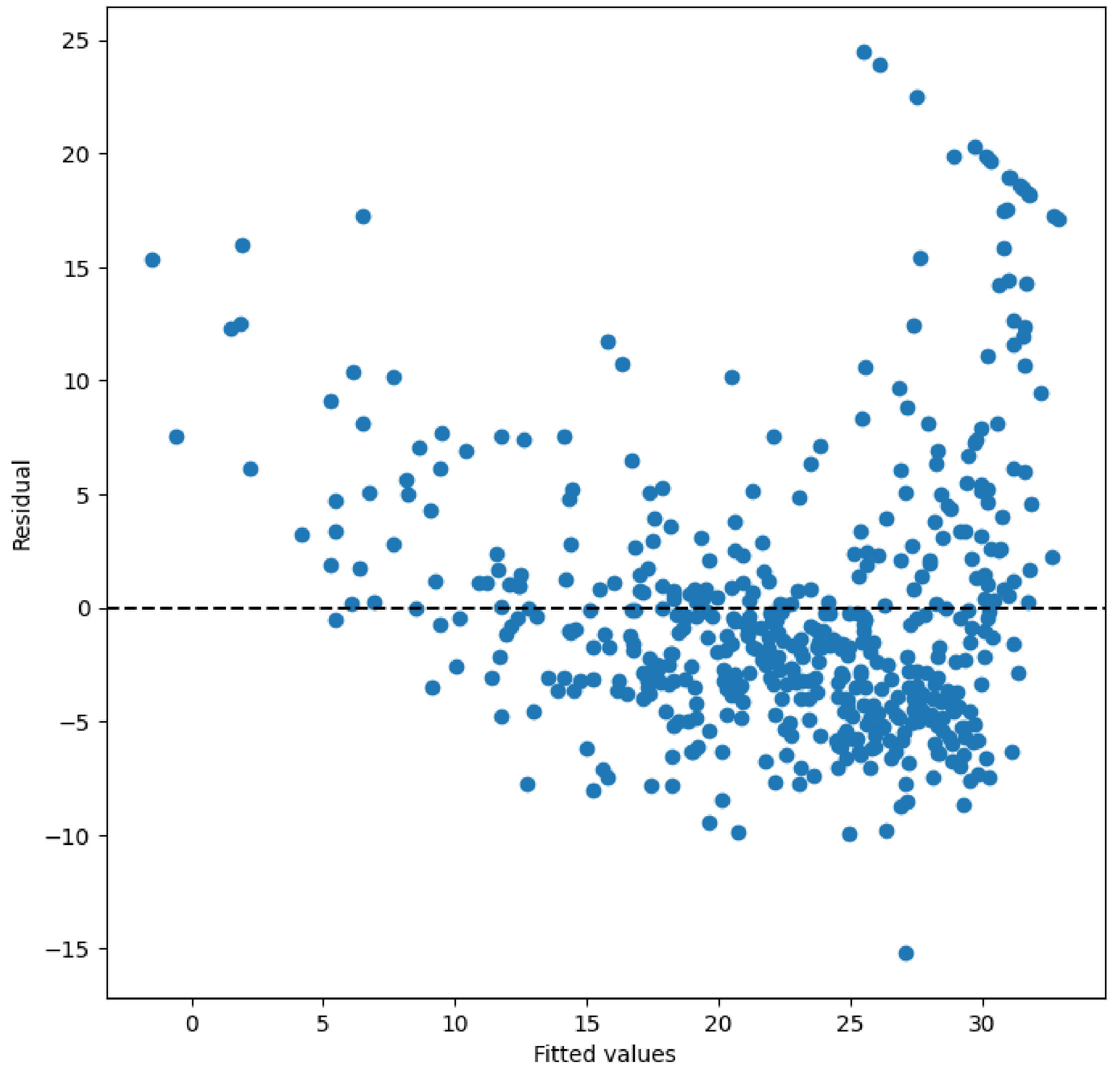
[ ]:
```
ax = Boston.plot.scatter('lstat','medv')
abline(ax,
       results.params[0],
       results.params[1],
       'k--',
       linewidth=3)
```



[ ]:
```
ax = subplots(figsize=(8,8))[1]
ax.scatter(results.fittedvalues, results.resid)
ax.set_xlabel('Fitted values')
```

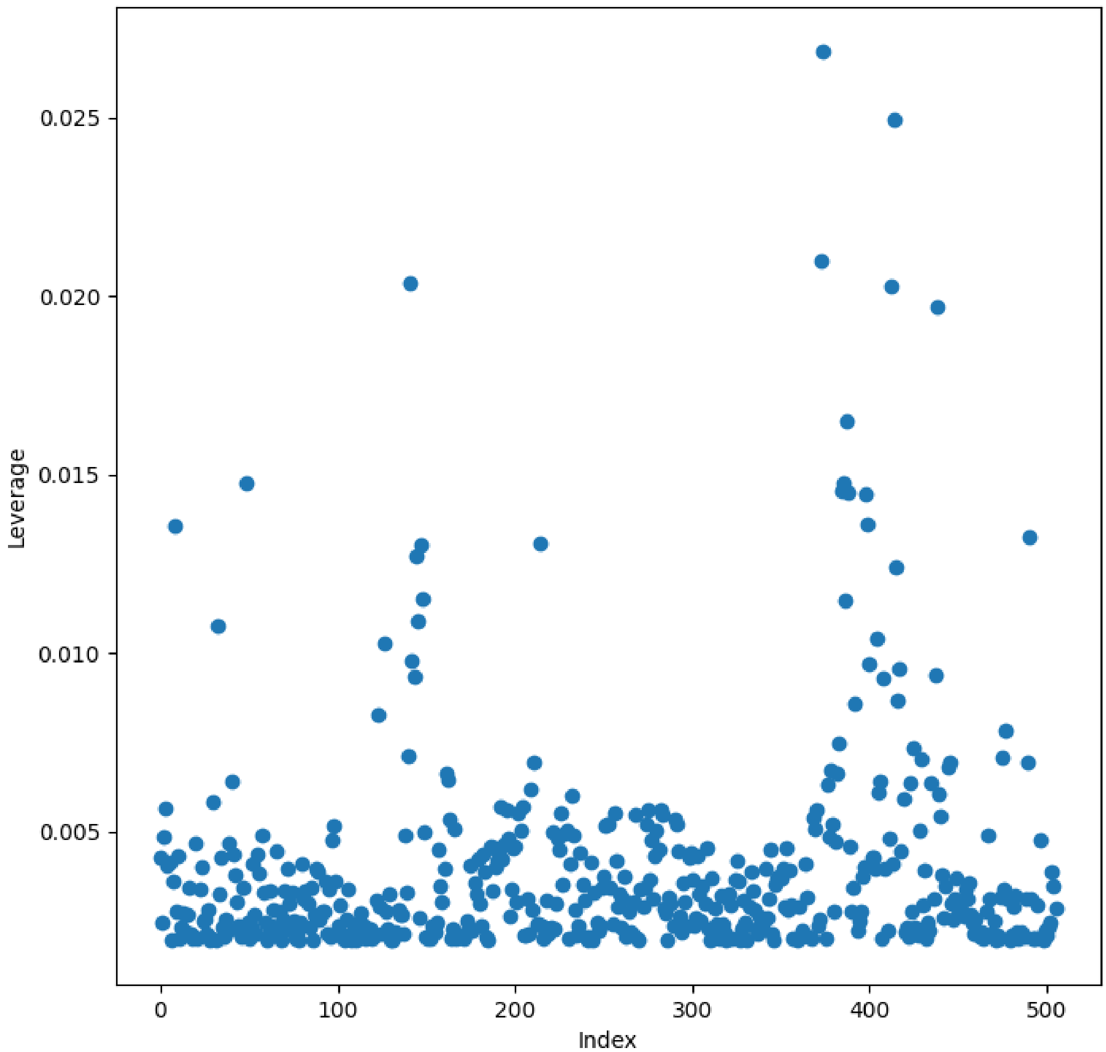
```
ax.set_ylabel('Residual')
ax.axhline(0, c='k', ls='--')
```

[ ]: <matplotlib.lines.Line2D at 0x7fd8ff844100>




```
[ ]: infl = results.get_influence()
ax = subplots(figsize=(8,8))[1]
ax.scatter(np.arange(X.shape[0]), infl.hat_matrix_diag)
ax.set_xlabel('Index')
ax.set_ylabel('Leverage')
np.argmax(infl.hat_matrix_diag)
```

[ ]: 374



[ ]:
```
X = MS(['lstat','age']).fit_transform(Boston)
model1 = sm.OLS(y, X)
results1 = model1.fit()
summarize(results1)
```

[ ]:

	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

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

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

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

[ ]:

	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

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

[ ]:

	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

[ ]:
```
vals = [VIF(X,i) for i in range(1, X.shape[1])]
vif = pd.DataFrame({'vif':vals}, index=X.columns[1:])
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
```

[ ]:
```python
vals = []
for i in range(1, X.values.shape[1]):
    vals.append(VIF(X.values, i))
```

[ ]:
```python
vals
```

[ ]:
```
[1.7674859154310116,
 2.2984589077358097,
 3.9871806307570994,
 1.0711677737584038,
 4.369092622844793,
 1.9125324374368873,
 3.0882320397311984,
 3.954036641628298,
 7.445300760069838,
 9.002157663471797,
 1.7970595931297797,
 2.8707765008417514]
```

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

[ ]:
```
              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
```

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

[ ]:

	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

[ ]:

```
anova_lm(results1, results3)
```

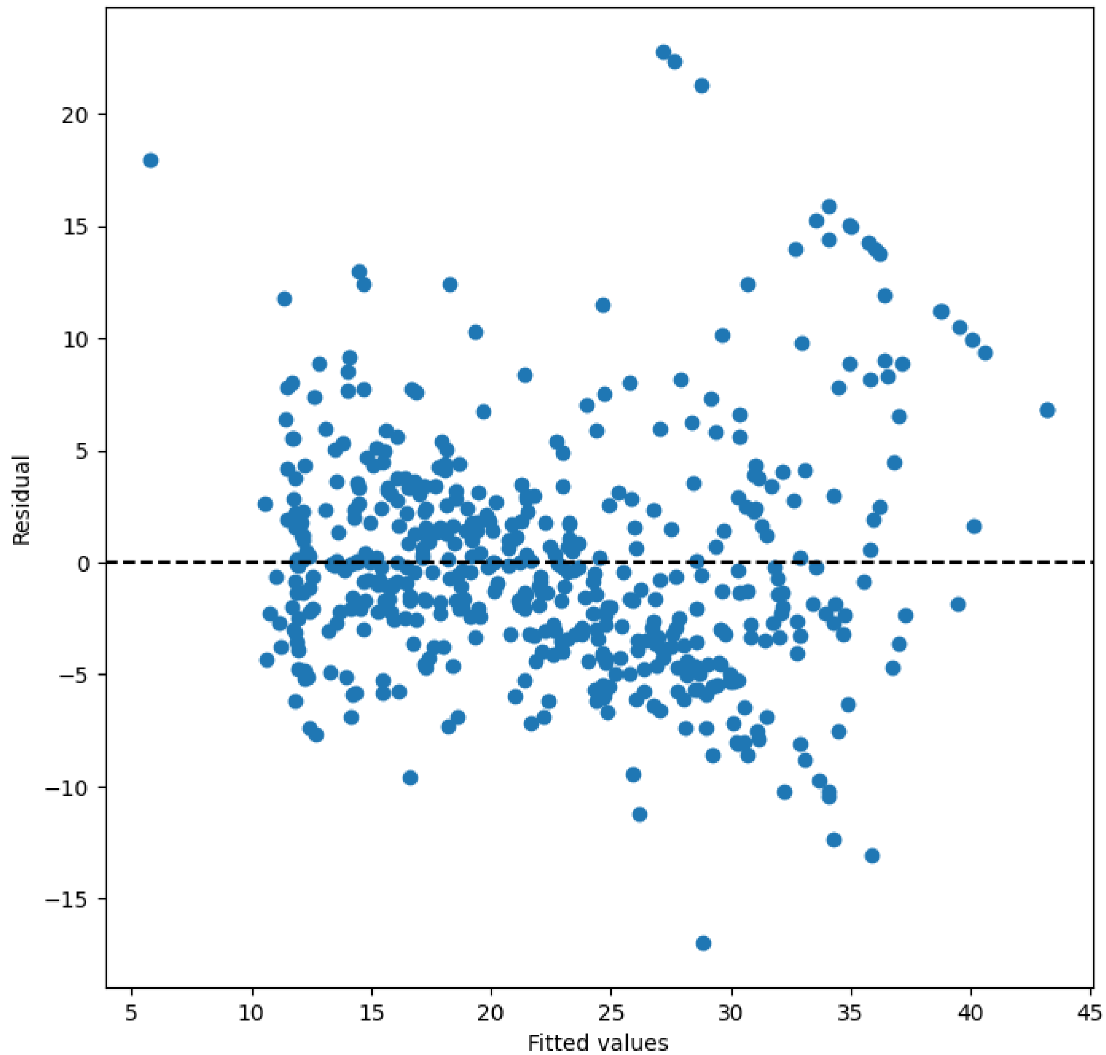
[ ]:

	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

[ ]:

```
ax = subplots(figsize=(8,8))[1]
ax.scatter(results3.fittedvalues, results3.resid)
ax.set_xlabel('Fitted values')
ax.set_ylabel('Residual')
ax.axhline(0, c='k', ls='--')
```

[ ]: <matplotlib.lines.Line2D at 0x7fd8fc463fa0>



[ ]:
```
Carseats = load_data('Carseats')
Carseats.columns
```

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

[ ]:
```
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())
```

[ ]:
```
                     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
```

1 Applied Homework

[ ]:
```python
auto = load_data('Auto')
```

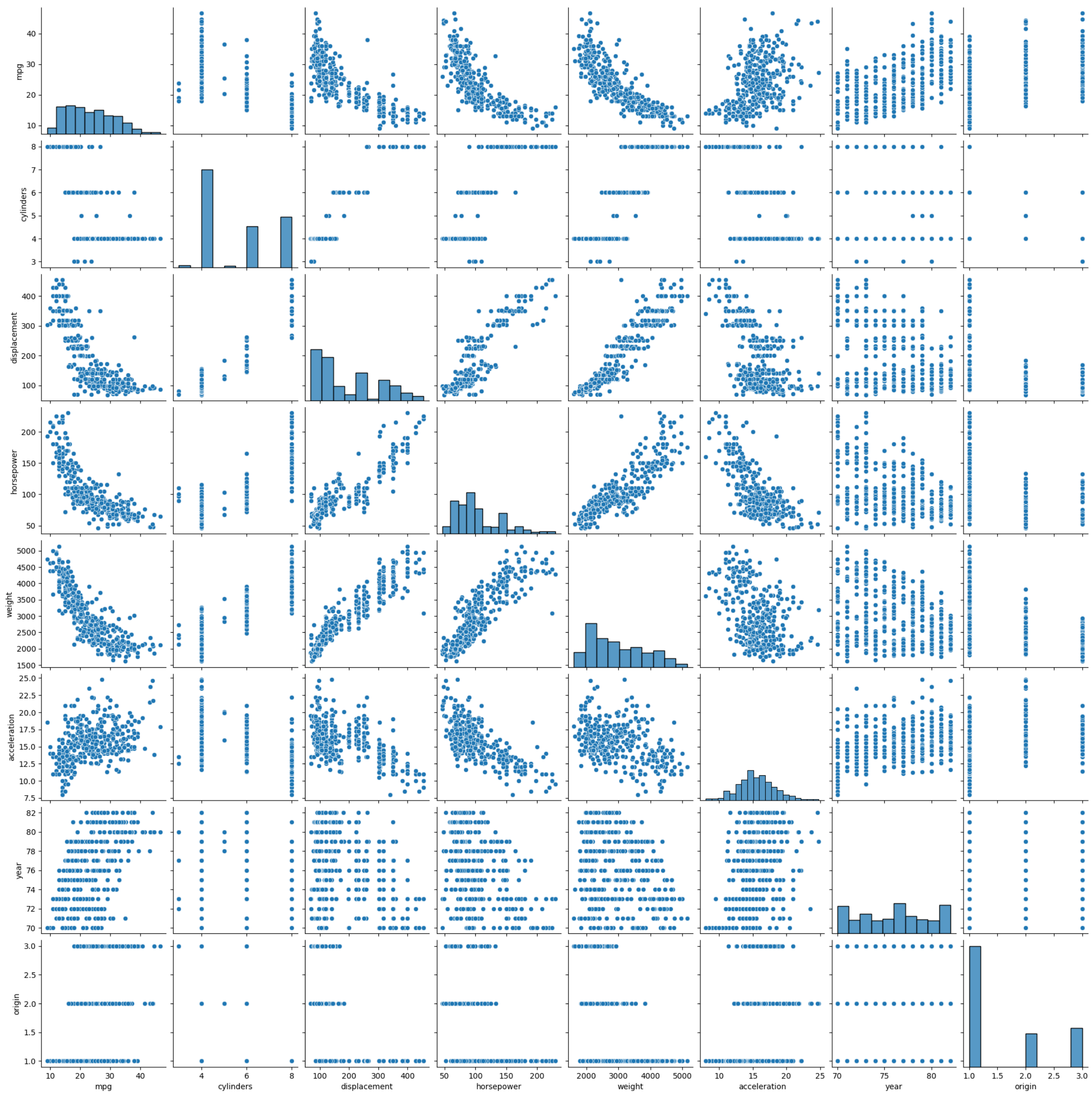
[ ]:
```python
auto
```

[ ]:
```
      mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0    18.0          8         307.0         130    3504          12.0    70
1    15.0          8         350.0         165    3693          11.5    70
2    18.0          8         318.0         150    3436          11.0    70
3    16.0          8         304.0         150    3433          12.0    70
4    17.0          8         302.0         140    3449          10.5    70
..    …          …           …         …       …           …   …
387  27.0          4         140.0          86    2790          15.6    82
388  44.0          4          97.0          52    2130          24.6    82
389  32.0          4         135.0          84    2295          11.6    82
390  28.0          4         120.0          79    2625          18.6    82
391  31.0          4         119.0          82    2720          19.4    82

     origin                       name
0         1  chevrolet chevelle malibu
1         1          buick skylark 320
2         1         plymouth satellite
3         1              amc rebel sst
4         1                ford torino
..      …                        …
387       1            ford mustang gl
388       2                  vw pickup
389       1              dodge rampage
390       1                ford ranger
```

```
391        1                   chevy s-10

[392 rows x 9 columns]
```

[ ]:
```python
sns.pairplot(auto)
```

[ ]: `<seaborn.axisgrid.PairGrid at 0x7fd8fc4a9f90>`



[ ]:
```python
X = pd.DataFrame({'intercept':np.ones(auto.shape[0]),
                  'horsepower': auto.horsepower})
y = auto.mpg
```

[ ]:
```python
model4 = sm.OLS(y, X)
```

[ ]:
```
results4 = model4.fit()
```

[ ]:
```
summarize(results4)
```

[ ]:

	coef	std err	t	P>|t|
intercept	39.9359	0.717	55.660	0.0
horsepower	-0.1578	0.006	-24.489	0.0

[ ]:
```
design1 = MS(['horsepower'])
design1 = design1.fit(auto)
X1 = design1.transform(auto)
X1

results4.summary()
```

[ ]:

Dep. Variable:	mpg	**R-squared:**	0.606
Model:	OLS	**Adj. R-squared:**	0.605
Method:	Least Squares	**F-statistic:**	599.7
Date:	Sun, 25 Feb 2024	**Prob (F-statistic):**	7.03e-81
Time:	19:46:37	**Log-Likelihood:**	-1178.7
No. Observations:	392	**AIC:**	2361.
Df Residuals:	390	**BIC:**	2369.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
intercept	39.9359	0.717	55.660	0.000	38.525	41.347
horsepower	-0.1578	0.006	-24.489	0.000	-0.171	-0.145

Omnibus:	16.432	**Durbin-Watson:**	0.920
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	17.305
Skew:	0.492	**Prob(JB):**	0.000175
Kurtosis:	3.299	**Cond. No.**	322.

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

[ ]:
```
new_df1 = pd.DataFrame({'horsepower': [98]})
newX1 = design1.transform(new_df1)
newX1
```

[ ]:

	intercept	horsepower
0	1.0	98

[ ]:
```
new_predictions1 = results4.get_prediction(newX1)
new_predictions1.predicted_mean
```

[ ]: array([24.46707715])

[ ]:
```python
new_predictions1.conf_int(alpha=0.05)
```

[ ]:
```
array([[23.97307896, 24.96107534]])
```

[ ]:
```python
new_predictions1.conf_int(obs=True, alpha=0.05)
```

[ ]:
```
array([[14.80939607, 34.12475823]])
```

(a)

i. Is there a relationship between the predictor and the response? Yes, there is a negative relationship between mpg and horsepower. As horsepower increases, mpg decreases.

ii. How strong is the relationship between the predictor and the reponse? The relationship has an R^2 value of 0.605 and an adjusted R^2 of 0.605. This is not a strong relationship.

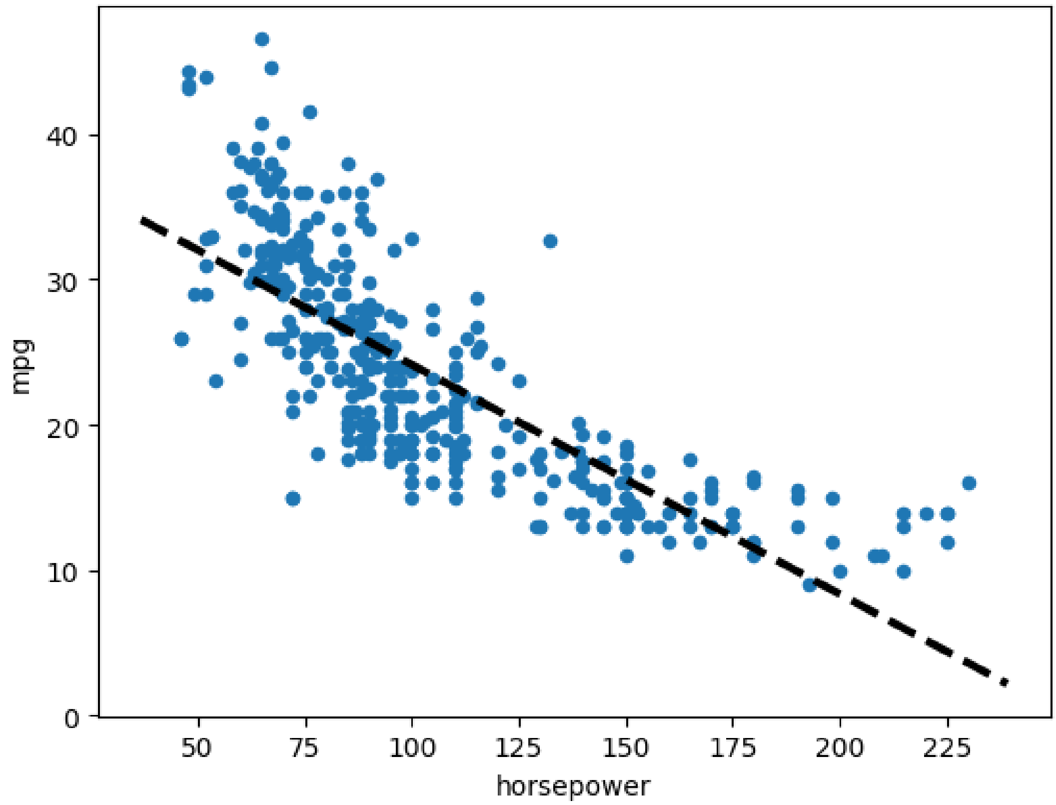
iii. Is the relationship between the predictor and the repsonse positive or negative? The relationship between the predictor and the response if negative.

iiii. What is the predicted mpg associated with a horsepower of 98? What are the associated 95% confidence and prediction intervals? The predicted mpg with a horsepower of 98hp is 24.47mpg. The 95% confidence interval is between 23.97 and 24.97mpg. The 95% prediction interval is between 14.81 and 34.12 mpg.

(b)

Plot the response and the predictor in a new set of axes ax. Use the ax.aline() method function defined int he lab to display the least squares regression line.

[ ]:
```python
ax1 = auto.plot.scatter('horsepower','mpg')
abline(ax1,
       results4.params[0],
       results4.params[1],
       'k--',
       linewidth=3)
```

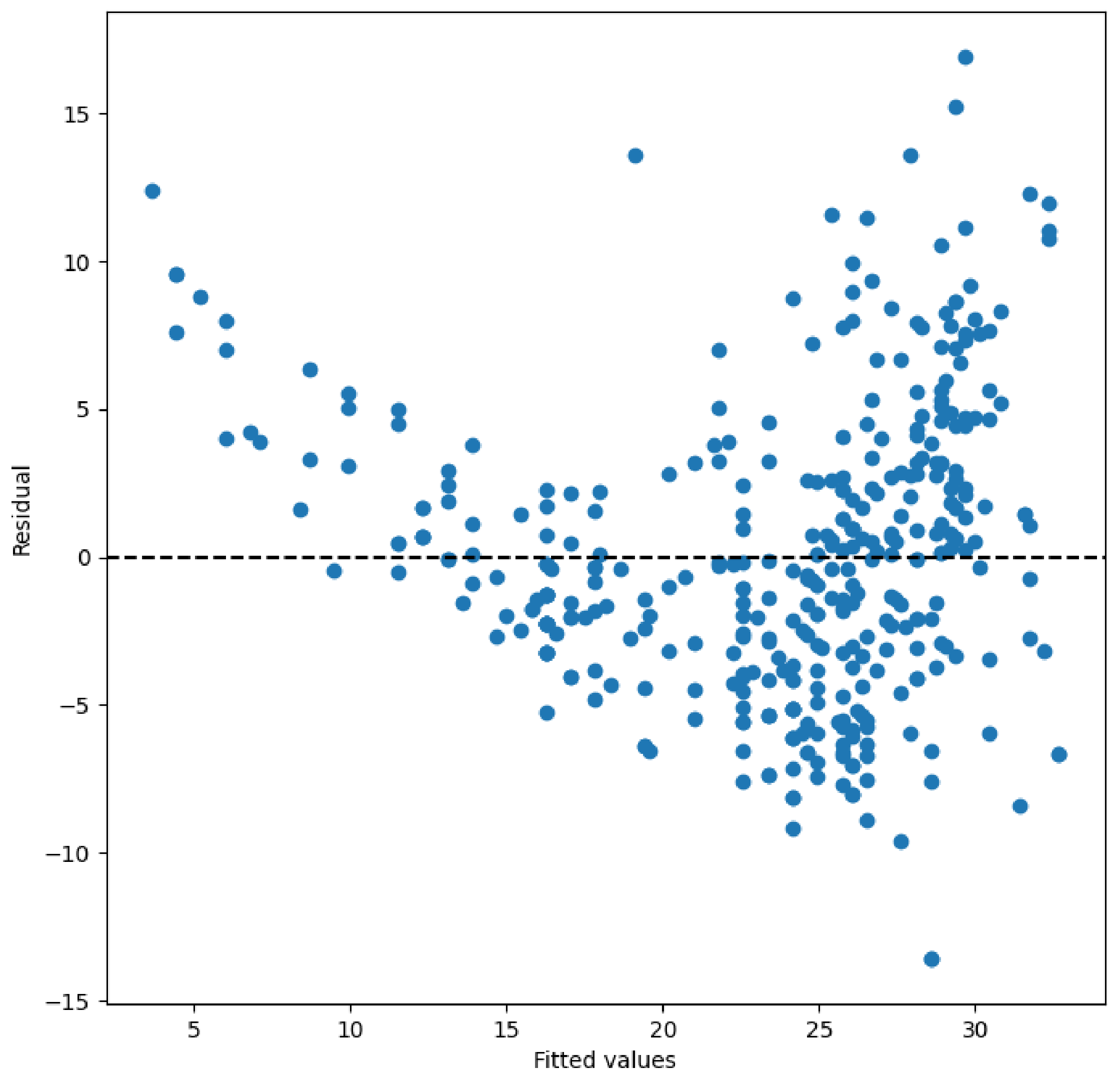


(c)

Produce some diagnostic plots of the least squares regression fit as described in the lab. Comment on any problems you see with the fit. The residual plot identifies a non linearity in the data, as the trend is "u shaped" and not straight. Utilizing a nonlinear transformation would most likly result in a better performing model.

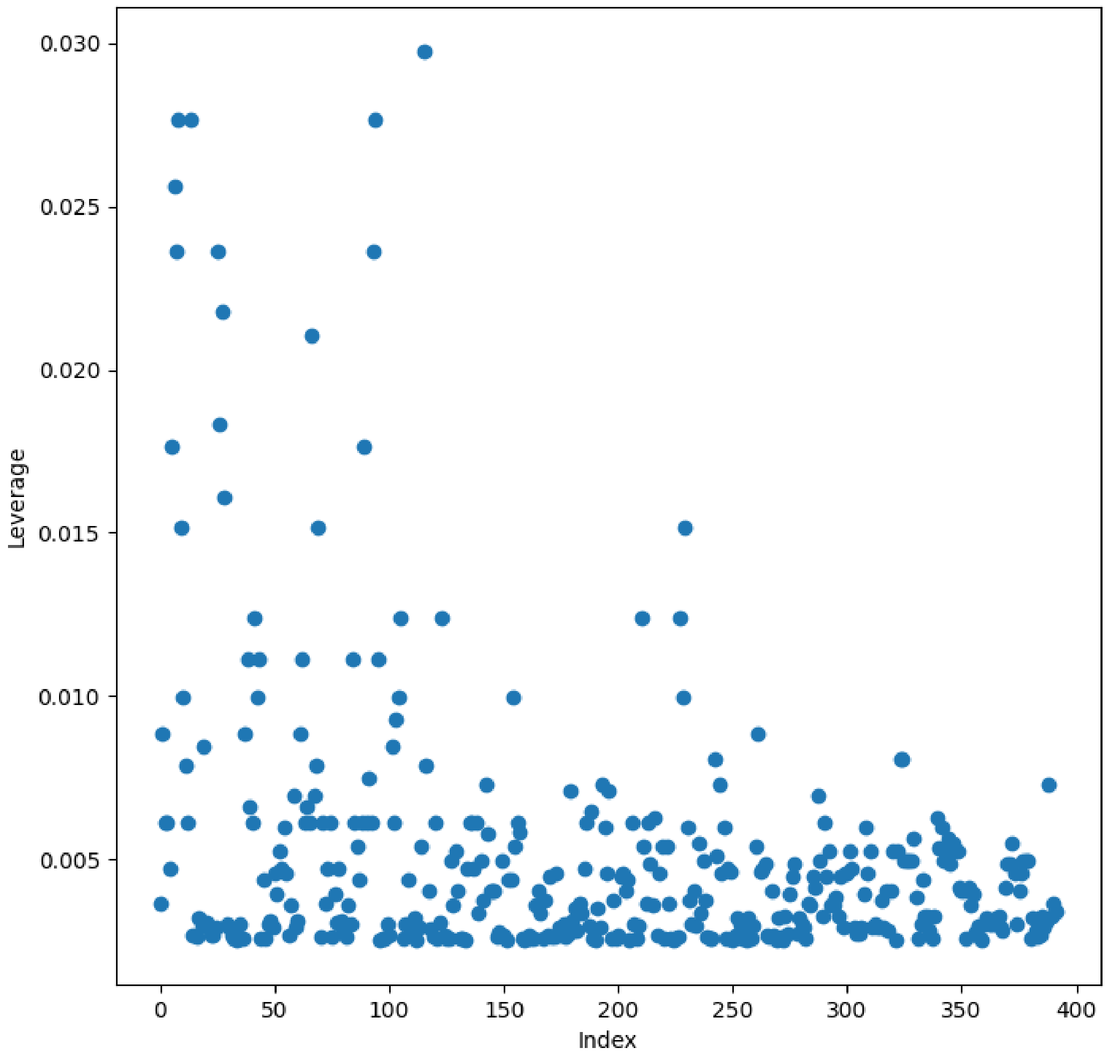
[ ]:
```
ax1 = subplots(figsize=(8,8))[1]
ax1.scatter(results4.fittedvalues, results4.resid)
ax1.set_xlabel('Fitted values')
ax1.set_ylabel('Residual')
ax1.axhline(0, c='k', ls='--')
```

[ ]: <matplotlib.lines.Line2D at 0x7fd8f7125cf0>



[ ]:
```
infl1 = results4.get_influence()
ax1 = subplots(figsize=(8,8))[1]
ax1.scatter(np.arange(X1.shape[0]), infl1.hat_matrix_diag)
ax1.set_xlabel('Index')
ax1.set_ylabel('Leverage')
np.argmax(infl1.hat_matrix_diag)
```

[ ]: 115


0.030
0.025
0.020
0.015
0.010
0.005
Leverage
0
50
100
150
200
250
300
350
400
Index

[ ]: