practice

November 19, 2024

[2]:
```python
import kagglehub

# Download latest version
path = kagglehub.dataset_download("vikrishnan/boston-house-prices")

print("Path to dataset files:", path)
```

```
Downloading from
https://www.kaggle.com/api/v1/datasets/download/vikrishnan/boston-house-
prices?dataset_version_number=1…

100%|
                                                  |
12.8k/12.8k [00:00<00:00, 6.83MB/s]

Extracting files…
Path to dataset files: /Users/bruce/.cache/kagglehub/datasets/vikrishnan/boston-
house-prices/versions/1
```

[1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns


import matplotlib.pyplot as plt
import plotly.express as px
import plotly.io as pio
pio.renderers.default = 'iframe'
from plotly import graph_objects as go
```

[2]:
```python
path = '/Users/bruce/.cache/kagglehub/datasets/vikrishnan/boston-house-prices/
 ↪versions/1'
col_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',␣
 ↪'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
df = pd.read_csv(path+'/housing.csv', header=None, delimiter=r"\s+",␣
 ↪names=col_names)
```

[3]:
```python
df
```

[3]:
```
        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  296.0
1    0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2    0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3    0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4    0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0
..       …      …     …     …      …     …     …      …   …      …
501  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1  273.0
502  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1  273.0
503  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1  273.0
504  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1  273.0
505  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1  273.0

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90   5.33  36.2
..       …       …     …     …
501     21.0  391.99   9.67  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

[4]:
```
df.describe()
```

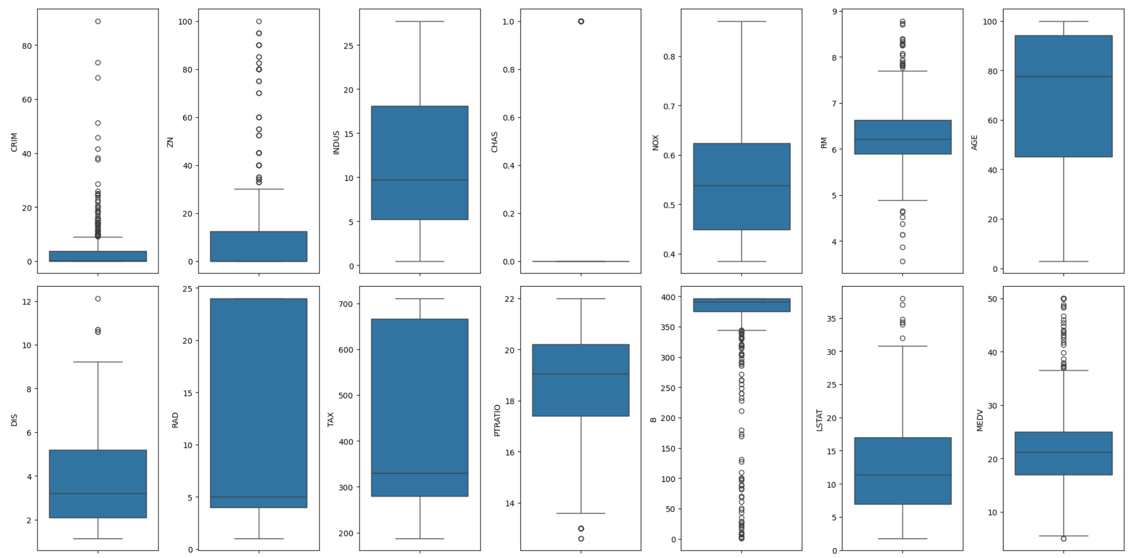
[4]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
```

```
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT        MEDV
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

[5]:
```
df.isna().any(axis=0)
```

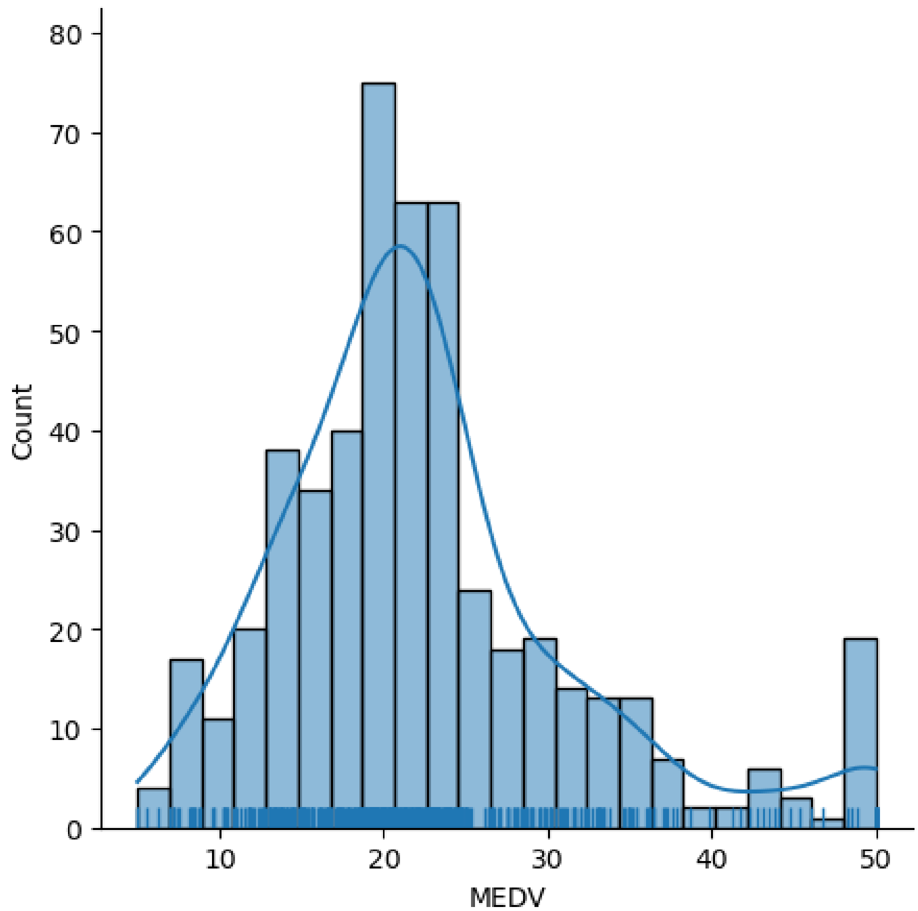
[5]:
```
CRIM       False
ZN         False
INDUS      False
CHAS       False
NOX        False
RM         False
AGE        False
DIS        False
RAD        False
TAX        False
PTRATIO    False
B          False
LSTAT      False
MEDV       False
dtype: bool
```

[6]:
```
fig, axs = plt.subplots(figsize=(20,10), ncols=7, nrows=2)
axs = axs.flatten()
for i, col in enumerate(col_names):
    sns.boxplot(data=df, y=col, ax=axs[i])
plt.tight_layout()
```



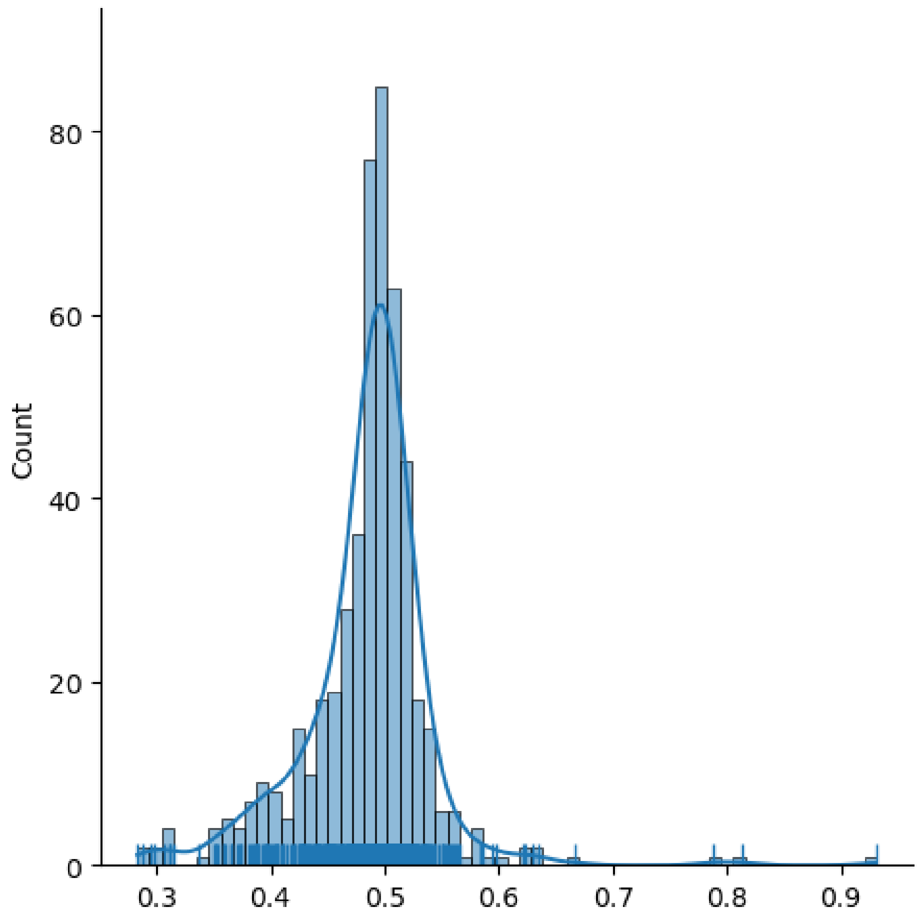
[253]:
```
sns.displot(df['MEDV'], rug=True, kde=True)
```

[253]: `<seaborn.axisgrid.FacetGrid at 0x2bc735870>`



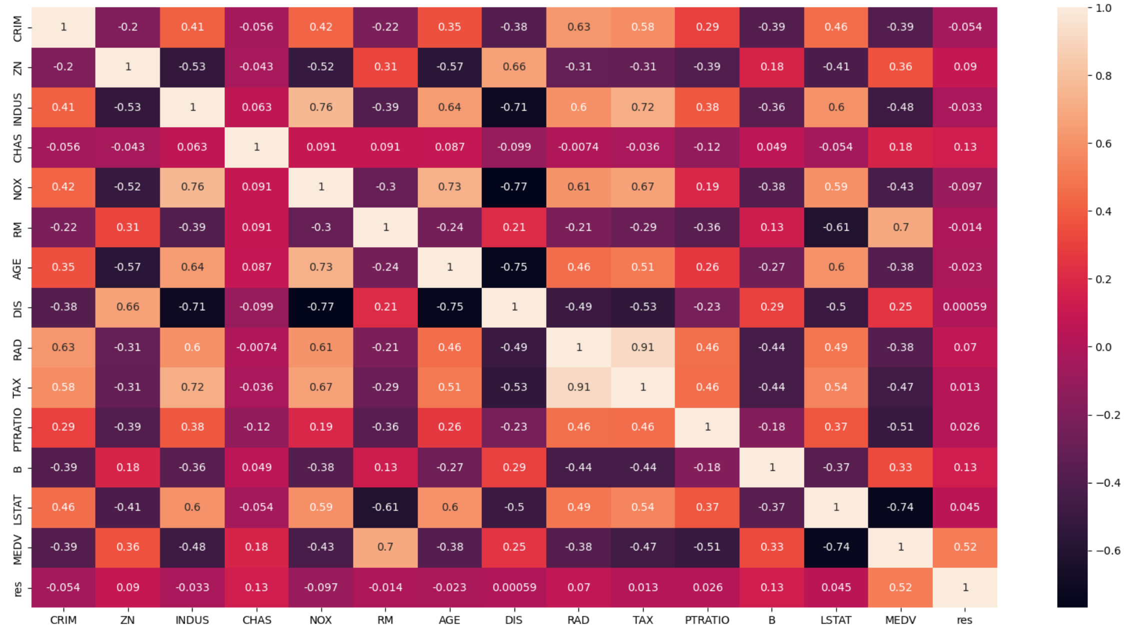
[255]:
```
sns.displot(np.log(df['MEDV']) / df['RM'], rug=True, kde=True)
```

[255]: <seaborn.axisgrid.FacetGrid at 0x2bca1aa70>

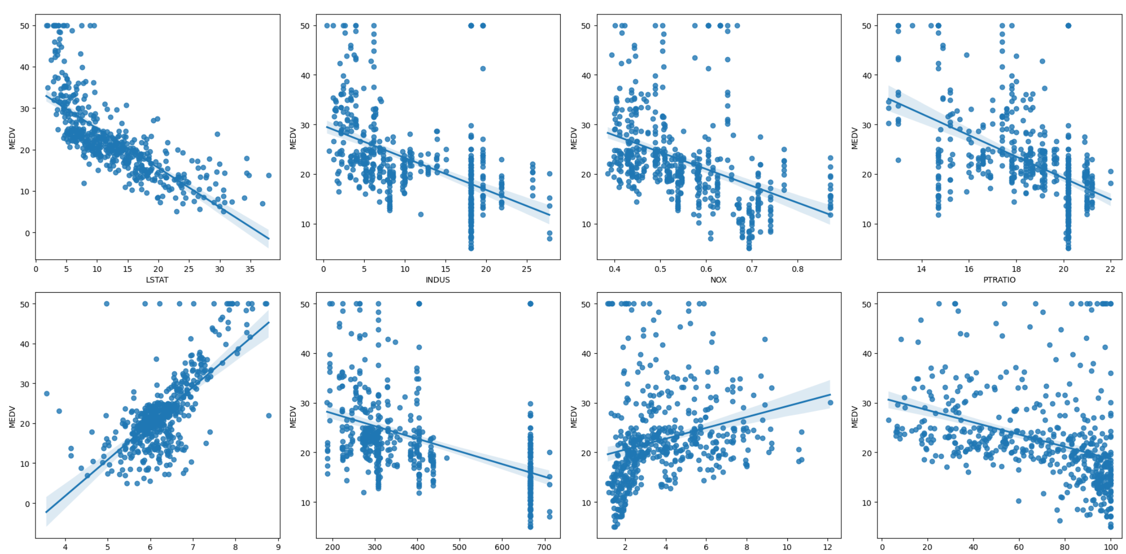


[258]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True)
#px.imshow(df.corr(), text_auto=True)
```

[258]: <Axes: >



[260]:
```python
fig, axs = plt.subplots(ncols=4, nrows=2, figsize=(20, 10))
axs = axs.flatten()
for i, k in enumerate(['LSTAT', 'INDUS', 'NOX', 'PTRATIO', 'RM', 'TAX', 'DIS', 'AGE']):
    sns.regplot(y=df['MEDV'], x=df[k], ax=axs[i])
plt.tight_layout()
```



```
[9]: df_norm = (df - df.mean()) / df.std()
```

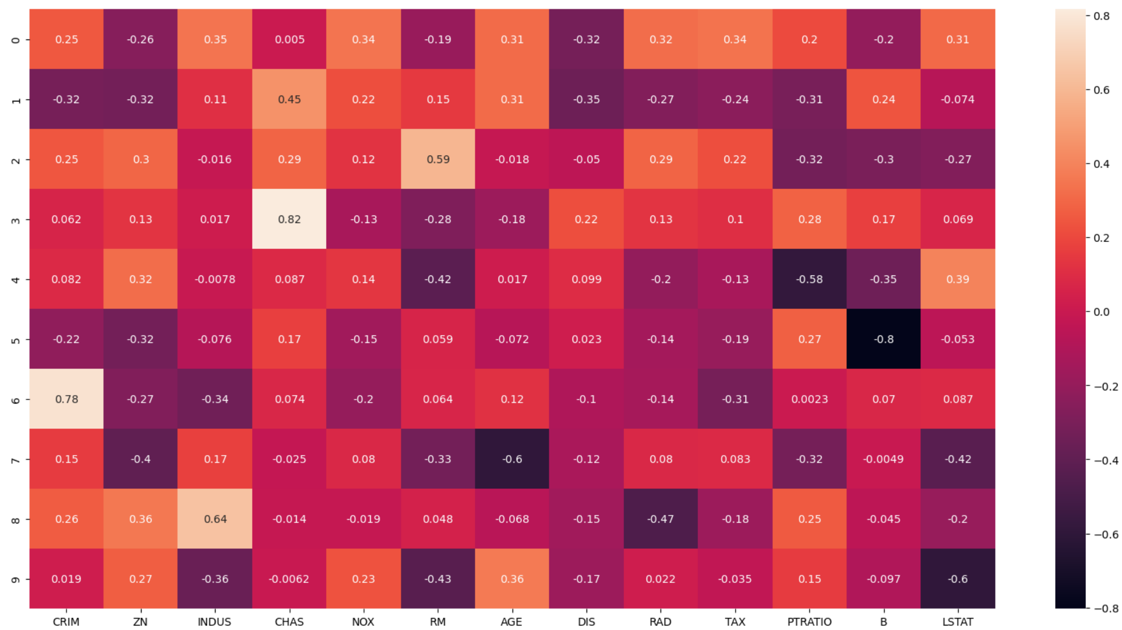
1 PCA

```
[10]: from sklearn import decomposition

      pca = decomposition.PCA(n_components=10)
      pca.fit(df_norm.drop(['MEDV'], axis=1))
```

```
[10]: PCA(n_components=10)
```

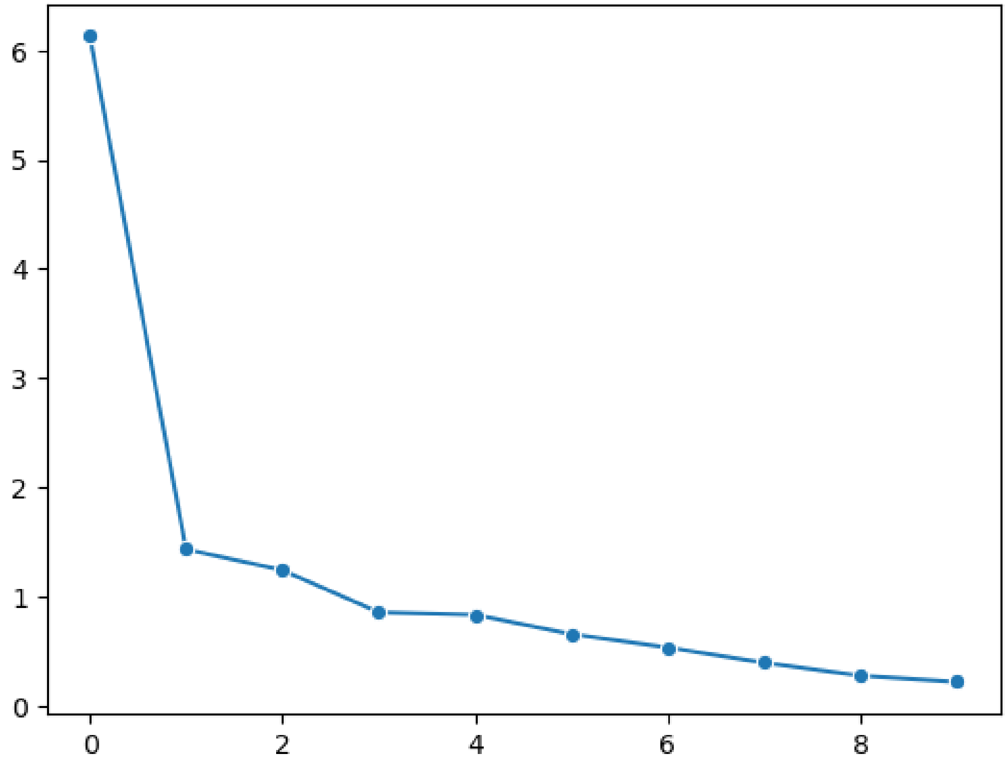
```
[17]: plt.figure(figsize=(20,10))
      sns.heatmap(pd.DataFrame(pca.components_, columns=col_names[:-1]), annot=True)
```

```
[17]: <Axes: >
```




```
[13]: px.imshow(pca.components_, x=col_names[:-1], text_auto=True)
```

```
[342]: sns.lineplot(data=pca.explained_variance_, marker='o')
       plt.show()
```



[232]:
```
df_pca = pd.DataFrame(pca.transform(df_norm.drop(['MEDV'], axis=1)))
df_pca['MEDV'] = df['MEDV']
```

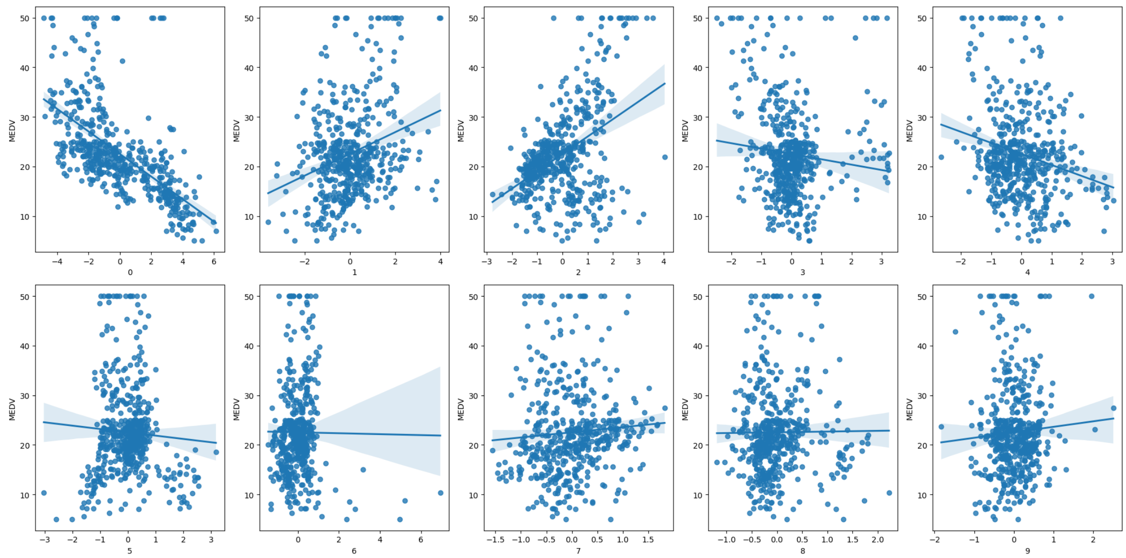
[233]:
```
df_pca.corr()[['MEDV']]
```

[233]:

	MEDV
0	-0.611745
1	0.285714
2	0.424334
3	-0.108814
4	-0.221845
5	-0.059122
6	-0.007503
7	0.071180
8	0.008551
9	0.056572
MEDV	1.000000

[261]:
```
fig, axs = plt.subplots(nrows=2, ncols=5, figsize=(20,10))
axs = axs.flatten()

for i in range(10):
```

```python
    sns.regplot(y=df_pca['MEDV'], x=df_pca[i], ax=axs[i])
plt.tight_layout()
```



[17]:
```python
px.scatter(x=df['LSTAT'], y=df['MEDV'])
```

[18]:
```python
px.scatter(x=df['LSTAT'], y=X_pca[:,0])
```

2 OLS

[18]:
```python
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
```

[19]:
```python
X = df[['LSTAT', 'DIS', 'TAX', 'PTRATIO', 'RM']]
y = df['MEDV']
X = sm.add_constant(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
 ↪shuffle=True)
```

[21]:
```python
import statsmodels.api as sm
```

[22]:
```python
model = sm.OLS(exog=X_train, endog=y_train)
results = model.fit() #_regularized(L1_wt=1, alpha=0) # L1_wt=1 means Lasso
```

[23]:
```python
results.summary()
```

[23]:

Dep. Variable:	MEDV	**R-squared:**	0.697
Model:	OLS	**Adj. R-squared:**	0.694
Method:	Least Squares	**F-statistic:**	206.5
Date:	Tue, 19 Nov 2024	**Prob (F-statistic):**	6.15e-114
Time:	22:14:49	**Log-Likelihood:**	-1378.6
No. Observations:	455	**AIC:**	2769.
Df Residuals:	449	**BIC:**	2794.
Df Model:	5		
Covariance Type:	nonrobust		

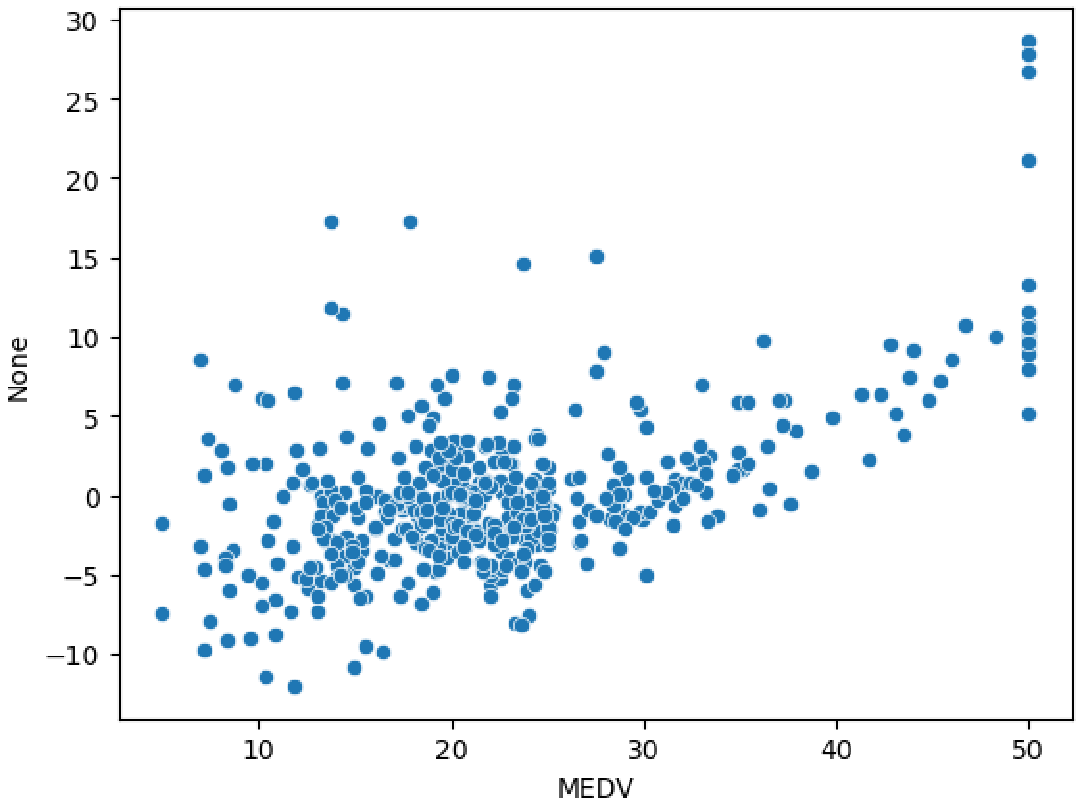
	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	21.7375	4.261	5.102	0.000	13.364	30.111
LSTAT	-0.6095	0.051	-11.908	0.000	-0.710	-0.509
DIS	-0.7325	0.140	-5.239	0.000	-1.007	-0.458
TAX	-0.0054	0.002	-2.835	0.005	-0.009	-0.002
PTRATIO	-0.8121	0.127	-6.380	0.000	-1.062	-0.562
RM	4.5379	0.451	10.067	0.000	3.652	5.424

Omnibus:	188.784	**Durbin-Watson:**	1.954
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1030.092
Skew:	1.727	**Prob(JB):**	2.08e-224
Kurtosis:	9.512	**Cond. No.**	7.99e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.99e+03. This might indicate that there are strong multicollinearity or other numerical problems.

[34]:
```
sns.scatterplot(x=y_train, y=results.resid)
```

[34]: `<Axes: xlabel='MEDV', ylabel='None'>`

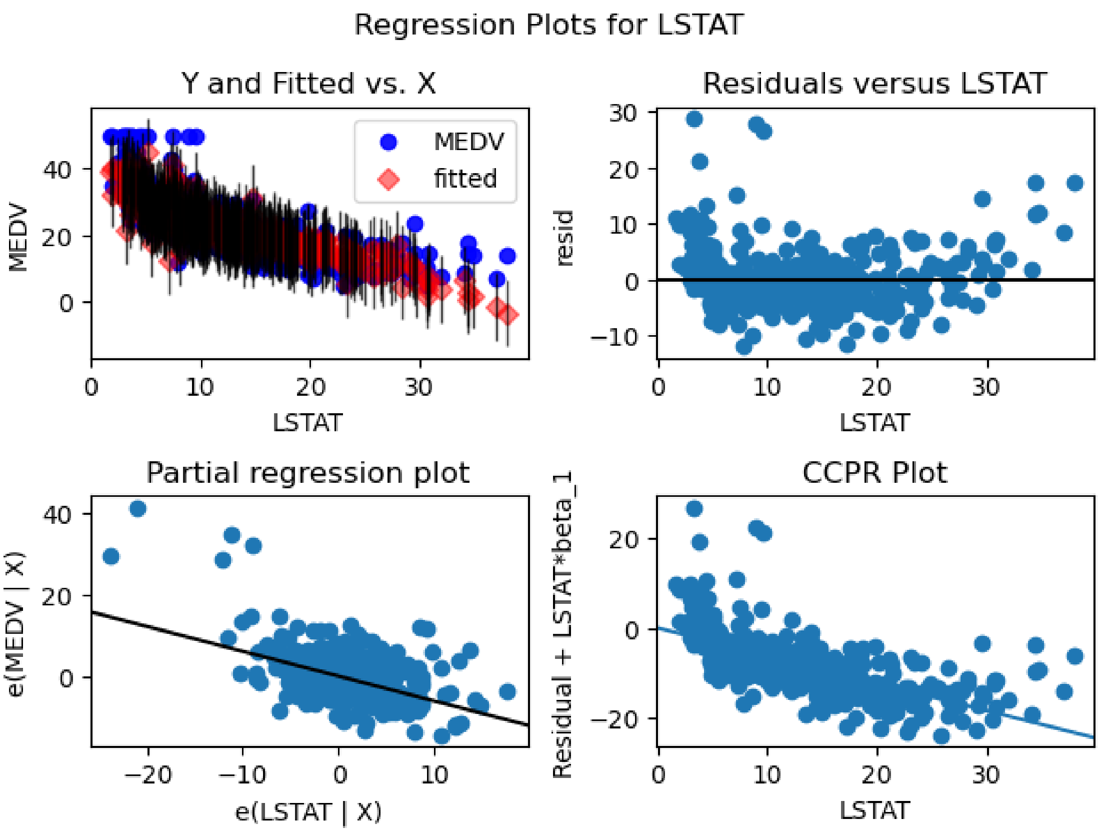


[24]:
```
from sklearn.metrics import mean_squared_error
```

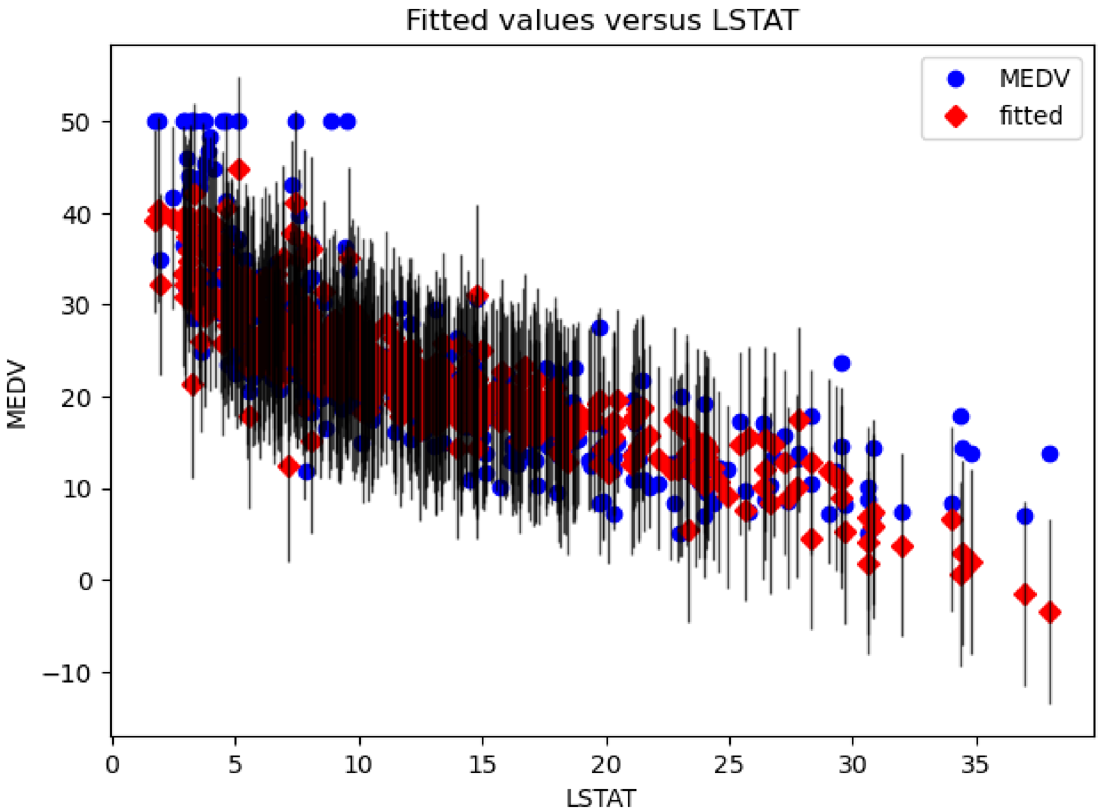
[25]:
```
mean_squared_error(results.predict(X_test), y_test)
```

[25]: 31.31090608217717

[39]:
```
fig = sm.graphics.plot_regress_exog(results, "LSTAT")
fig.tight_layout(pad=1.0)
```



[38]:
```
fig = sm.graphics.plot_fit(results, "LSTAT")
fig.tight_layout(pad=1.0)
```



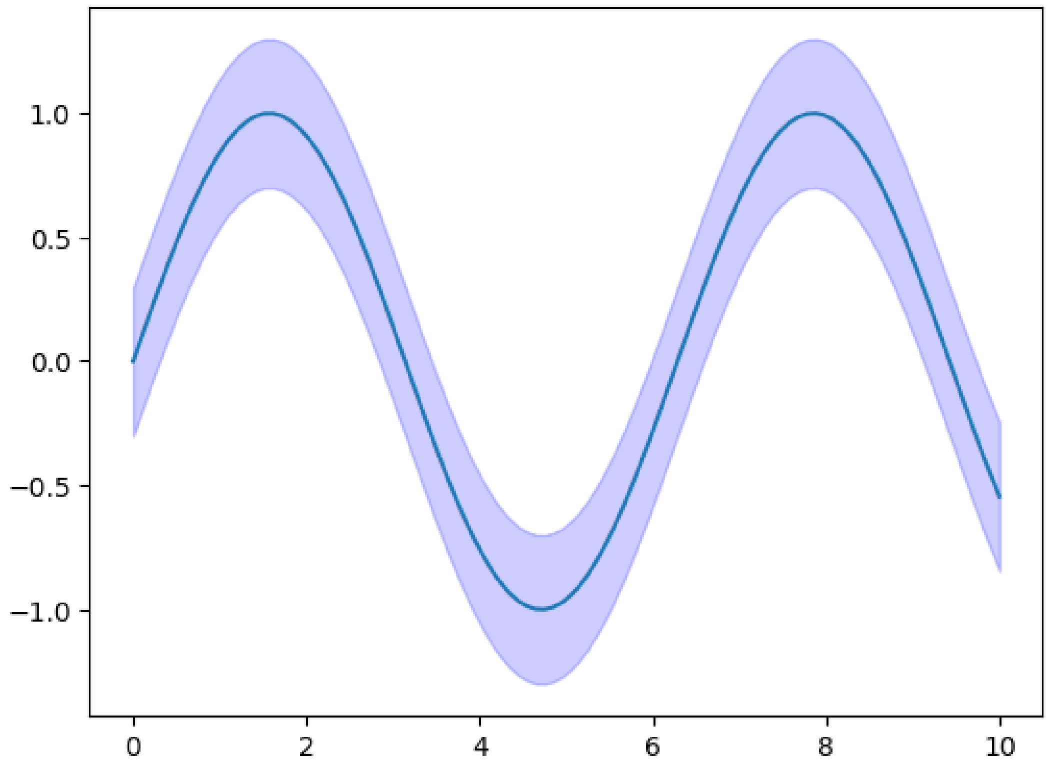
[42]:
```
conf = results.get_prediction(X_test).summary_frame(alpha=0.05).
    ↪join(X_test['LSTAT'])
conf.head(5)
```

[42]:
```
          mean   mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  \
7    19.697067  0.701645      18.318151      21.075983      9.694338
248  22.107377  0.556888      21.012946      23.201807     12.139881
119  20.647399  0.378258      19.904023      21.390775     10.712320
203  38.159159  0.650482      36.880792      39.437525     28.169794
424  15.031669  0.492971      14.062852      16.000487      5.077183

     obs_ci_upper  LSTAT
7       29.699796  19.15
248     32.074872   9.52
119     30.582478  13.61
203     48.148524   3.81
424     24.986155  17.16
```

[61]:
```
x = np.linspace(0, 10, 100)
y = np.sin(x)
y_upper = y + 0.3
```

```
y_lower = y - 0.3

ax = sns.lineplot(x=x, y=y)
ax.fill_between(x, y_lower, y_upper, alpha=0.2, color='b', label='Confidence')
```

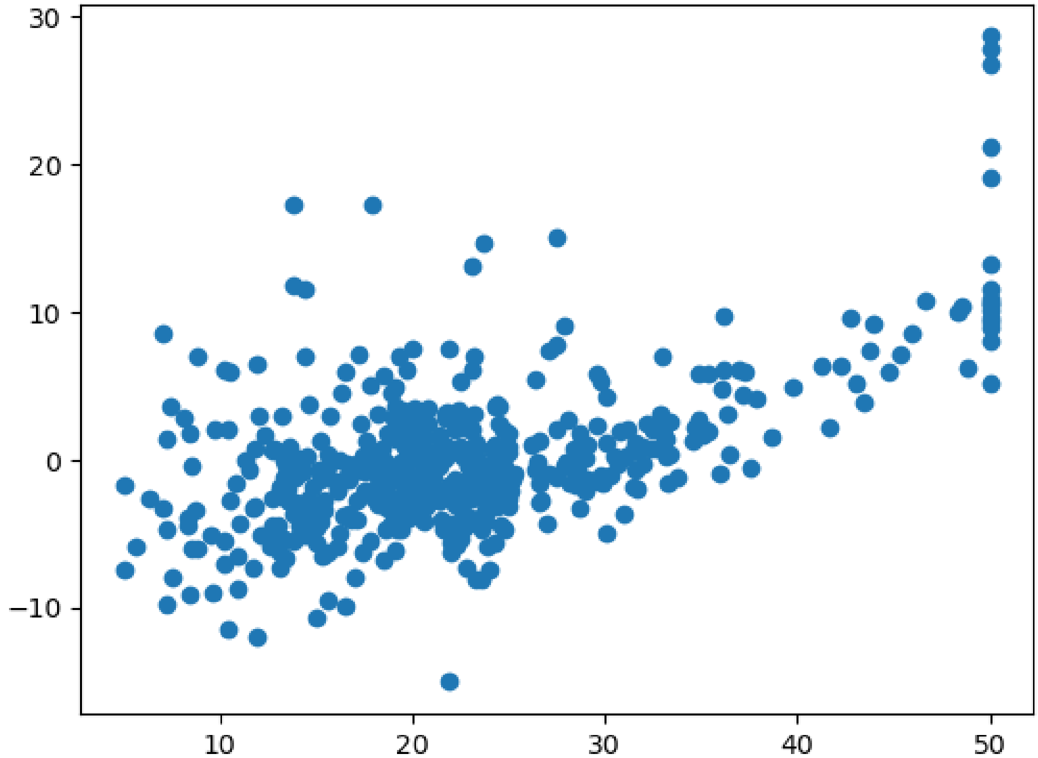
[61]: <matplotlib.collections.PolyCollection at 0x294e2f3d0>



[26]:
```
res = y - results.predict(X)
```

[33]:
```
plt.scatter(y, res)
```

[33]: <matplotlib.collections.PathCollection at 0x16a2caf80>

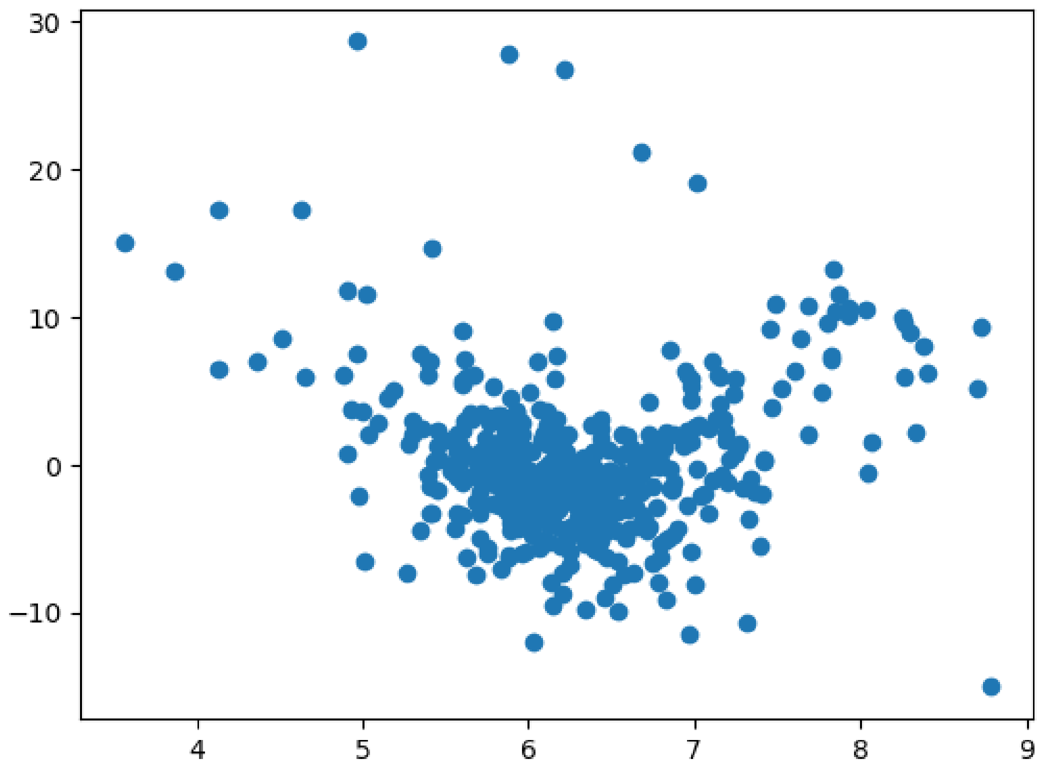



```
[28]: df['res'] = res
```

```
[29]: px.imshow(df.corr(), text_auto=True)
```

```
[30]: plt.scatter(df['RM'], res)
```

```
[30]: <matplotlib.collections.PathCollection at 0x16a032500>
```



3 GLM

see: https://www.statsmodels.org/stable/glm.html

[67]:
```
log_model = sm.GLM(endog=y_train, exog=X_train, family=sm.families.
  ↪Gaussian(link=sm.families.links.Log()))
log_results = log_model.fit()
```

[68]:
```
log_results.summary()
```

[68]:

Dep. Variable:	MEDV	**No. Observations:**	455
Model:	GLM	**Df Residuals:**	449
Model Family:	Gaussian	**Df Model:**	5
Link Function:	Log	**Scale:**	18.406
Method:	IRLS	**Log-Likelihood:**	-1305.3
Date:	Tue, 19 Nov 2024	**Deviance:**	8264.3
Time:	22:44:53	**Pearson chi2:**	8.26e+03
No. Iterations:	8	**Pseudo R-squ. (CS):**	0.9701
Covariance Type:	nonrobust		

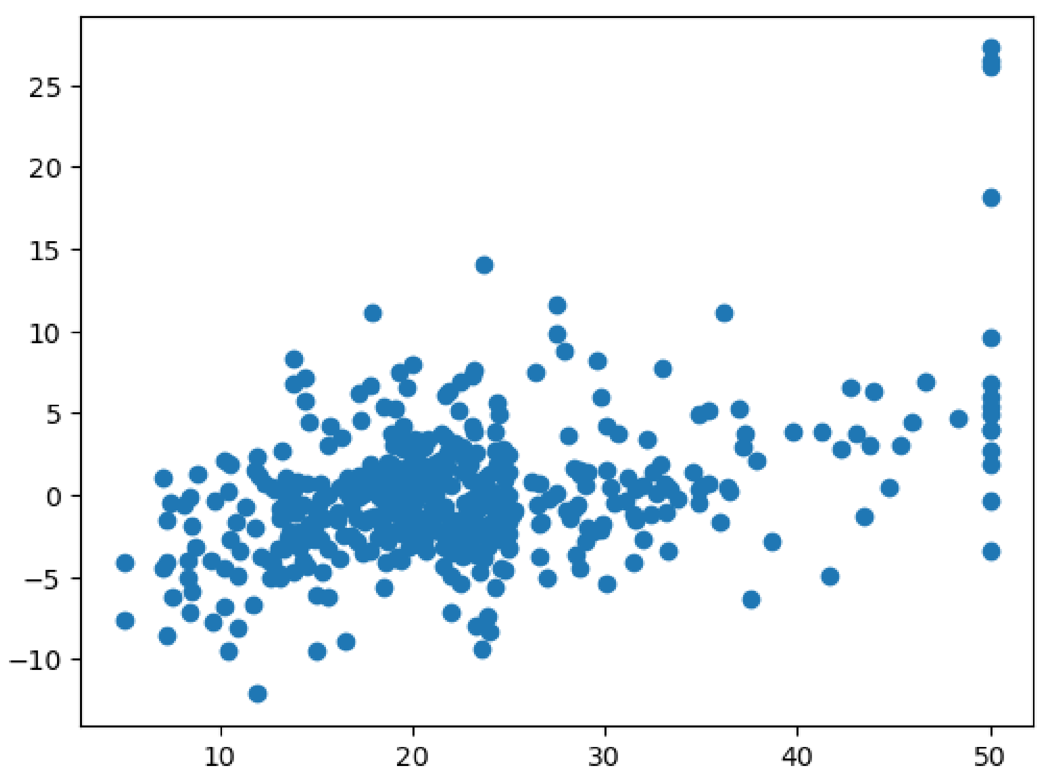
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	3.0235	0.152	19.882	0.000	2.725	3.322
LSTAT	-0.0385	0.002	-15.845	0.000	-0.043	-0.034
DIS	-0.0299	0.005	-6.333	0.000	-0.039	-0.021
TAX	-0.0001	7.68e-05	-1.612	0.107	-0.000	2.67e-05
PTRATIO	-0.0242	0.004	-5.610	0.000	-0.033	-0.016
RM	0.1783	0.015	12.033	0.000	0.149	0.207

[69]:
```
mean_squared_error(log_results.predict(X_test), y_test)
```

[69]: 28.42567629267149

[70]:
```
plt.scatter(y_train, log_results.resid_response)
```

[70]: <matplotlib.collections.PathCollection at 0x2950bd5a0>



[100]:
```
# --- Logit model ---
# sm.Logit(endog=, exog=)
```

[ ]:
```
```

4 RandomForest

[71]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

[72]:
```
rf_cv = GridSearchCV(RandomForestRegressor(),
                     param_grid={'n_estimators': [100, 300, 500],
                                 'max_depth': [3, 5, 10],
                                 'max_features': [1.0, 'sqrt']},
                     scoring='neg_mean_squared_error',
                     cv=5,
                     verbose=0,
                    )
```

[282]:
```
rf_cv.fit(X_train, y_train)
```

[282]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'max_depth': [3, 5, 10], 'max_features': [1.0, 'sqrt'],
                         'n_estimators': [100, 300, 500]},
             scoring='neg_mean_squared_error')
```

[292]:
```
#rf_cv.best_params_
pd.DataFrame(rf_cv.cv_results_).sort_values('rank_test_score')
```

[292]:
```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
17       0.266116      0.055677         0.009936        0.000291
15       0.045862      0.000494         0.002179        0.000047
16       0.141573      0.004142         0.006410        0.000597
14       0.336194      0.002969         0.009589        0.000132
9        0.039489      0.000439         0.001889        0.000120
13       0.208770      0.009322         0.006215        0.000291
12       0.070385      0.001201         0.002541        0.000160
6        0.054928      0.000717         0.002127        0.000042
11       0.201186      0.004776         0.007659        0.000229
8        0.264963      0.003692         0.007369        0.000204
10       0.119924      0.002824         0.004712        0.000202
7        0.157109      0.002981         0.004470        0.000101
0        0.051108      0.012603         0.001796        0.000204
1        0.138788      0.003875         0.004180        0.000202
```

```
2        0.227359      0.003771         0.006639        0.000158
4        0.110966      0.003778         0.004272        0.000219
3        0.038823      0.000665         0.002038        0.000118
5        0.180174      0.003536         0.006502        0.000248

   param_max_depth param_max_features param_n_estimators  \
17              10               sqrt                500
15              10               sqrt                100
16              10               sqrt                300
14              10                1.0                500
9                5               sqrt                100
13              10                1.0                300
12              10                1.0                100
6                5                1.0                100
11               5               sqrt                500
8                5                1.0                500
10               5               sqrt                300
7                5                1.0                300
0                3                1.0                100
1                3                1.0                300
2                3                1.0                500
4                3               sqrt                300
3                3               sqrt                100
5                3               sqrt                500

                                               params  split0_test_score  \
17  {'max_depth': 10, 'max_features': 'sqrt', 'n_e…          -7.448951
15  {'max_depth': 10, 'max_features': 'sqrt', 'n_e…          -7.070799
16  {'max_depth': 10, 'max_features': 'sqrt', 'n_e…          -7.201673
14  {'max_depth': 10, 'max_features': 1.0, 'n_esti…          -7.965726
9   {'max_depth': 5, 'max_features': 'sqrt', 'n_es…          -8.194504
13  {'max_depth': 10, 'max_features': 1.0, 'n_esti…          -7.966277
12  {'max_depth': 10, 'max_features': 1.0, 'n_esti…          -7.491151
6   {'max_depth': 5, 'max_features': 1.0, 'n_estim…          -8.616203
11  {'max_depth': 5, 'max_features': 'sqrt', 'n_es…          -8.532719
8   {'max_depth': 5, 'max_features': 1.0, 'n_estim…          -8.478953
10  {'max_depth': 5, 'max_features': 'sqrt', 'n_es…          -8.721183
7   {'max_depth': 5, 'max_features': 1.0, 'n_estim…          -8.421314
0   {'max_depth': 3, 'max_features': 1.0, 'n_estim…         -12.063792
1   {'max_depth': 3, 'max_features': 1.0, 'n_estim…         -11.976411
2   {'max_depth': 3, 'max_features': 1.0, 'n_estim…         -12.149863
4   {'max_depth': 3, 'max_features': 'sqrt', 'n_es…         -13.107641
3   {'max_depth': 3, 'max_features': 'sqrt', 'n_es…         -12.410374
5   {'max_depth': 3, 'max_features': 'sqrt', 'n_es…         -12.854066

    split1_test_score  split2_test_score  split3_test_score  \
17         -12.689259         -13.078331         -13.908736
```

15	-11.195586	-13.411535	-14.703290
16	-12.223230	-13.175793	-14.328282
14	-11.919455	-16.493886	-16.287242
9	-14.436129	-12.550843	-15.544291
13	-13.087440	-16.380269	-15.722388
12	-12.378430	-16.560149	-17.330716
6	-12.503808	-16.656268	-15.798471
11	-13.834109	-13.650729	-15.702470
8	-12.016245	-16.452318	-16.089019
10	-15.904770	-13.154462	-15.466326
7	-12.482940	-16.420491	-16.294311
0	-16.740463	-20.460051	-21.059465
1	-16.584222	-20.265343	-21.565937
2	-16.722707	-20.207380	-21.032545
4	-24.639471	-15.641116	-21.905448
3	-25.998506	-16.204044	-20.704805
5	-25.320643	-16.432822	-21.382617

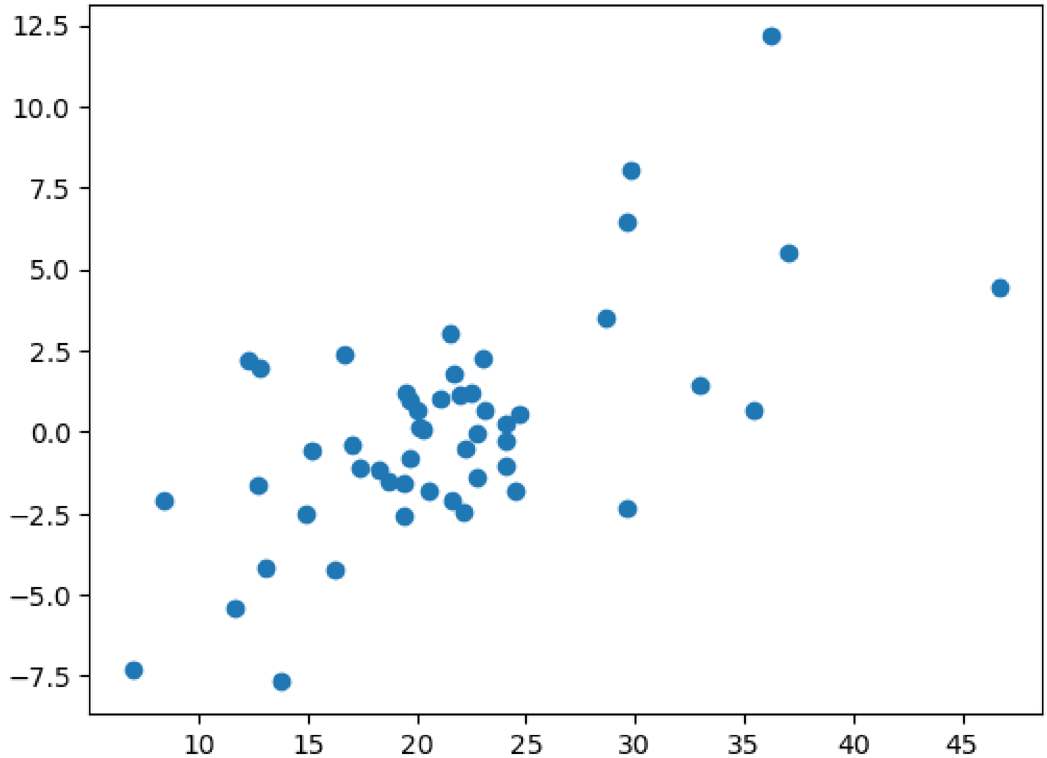
	split4_test_score	mean_test_score	std_test_score	rank_test_score
17	-10.563946	-11.537845	2.323252	1
15	-11.781437	-11.632529	2.592948	2
16	-11.280187	-11.641833	2.439421	3
14	-10.736027	-12.680467	3.290585	4
9	-12.773048	-12.699763	2.507238	5
13	-10.448648	-12.721004	3.171871	6
12	-9.919546	-12.735999	3.776354	7
6	-10.380804	-12.791111	3.075895	8
11	-12.433978	-12.830801	2.389827	9
8	-11.313827	-12.870072	3.020826	10
10	-11.712183	-12.991785	2.627193	11
7	-11.450183	-13.013848	3.039241	12
0	-13.521545	-16.769063	3.597713	13
1	-13.514683	-16.781319	3.710199	14
2	-14.242345	-16.870968	3.396199	15
4	-19.132695	-18.885274	4.152085	16
3	-19.179704	-18.899486	4.541096	17
5	-19.587182	-19.115466	4.248963	18

[293]:
```
mean_squared_error(rf_cv.predict(X_test), y_test)
```

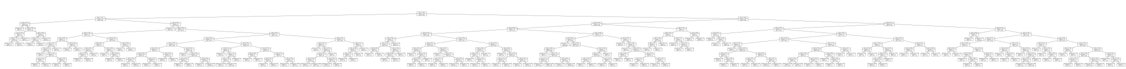
[293]: 10.781680971439544

[294]:
```
plt.scatter(y_test, y_test - rf.predict(X_test))
```

[294]: <matplotlib.collections.PathCollection at 0x2bf116320>



[300]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(200,10))
plot_tree(rf_cv.best_estimator_.estimators_[0])
plt.show()
```



[147]:
```
import xgboost
```

[172]:
```
bst = xgboost.XGBRegressor(n_estimators=5000, max_depth=3, learning_rate=1e-3)
```

[173]:
```
bst.fit(X_train, y_train)
```

[173]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.001, max_bin=None,
```

```
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=3, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=None,
multi_strategy=None, n_estimators=5000, n_jobs=None,
num_parallel_tree=None, random_state=None, …)
```

[174]:
```
mean_squared_error(bst.predict(X_test), y_test)
```

[174]: 11.443298078843423

[177]:
```
px.scatter(x=y_test, y=y_test - bst.predict(X_test))
```

5 OLS with formulas

[301]:
```
import statsmodels.formula.api as smf
```

[321]:
```
model_f = smf.ols(formula='np.log1p(MEDV) ~ LSTAT*RM + DIS + np.log1p(TAX) +␣
↪PTRATIO + 1', data=pd.concat([X_train, y_train], axis=1))
```

[322]:
```
result_f = model_f.fit()
result_f.summary()
```

[322]:

Dep. Variable:	np.log1p(MEDV)	**R-squared:**	0.770
Model:	OLS	**Adj. R-squared:**	0.767
Method:	Least Squares	**F-statistic:**	250.0
Date:	Wed, 16 Oct 2024	**Prob (F-statistic):**	1.53e-139
Time:	23:05:23	**Log-Likelihood:**	114.88
No. Observations:	455	**AIC:**	-215.8
Df Residuals:	448	**BIC:**	-186.9
Df Model:	6		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	3.6643	0.256	14.315	0.000	3.161	4.167
LSTAT	0.0326	0.010	3.342	0.001	0.013	0.052
RM	0.2044	0.023	8.966	0.000	0.160	0.249
LSTAT:RM	-0.0114	0.002	-6.981	0.000	-0.015	-0.008
DIS	-0.0235	0.005	-4.437	0.000	-0.034	-0.013
np.log1p(TAX)	-0.1444	0.030	-4.865	0.000	-0.203	-0.086
PTRATIO	-0.0246	0.005	-5.092	0.000	-0.034	-0.015

Omnibus:	53.724	**Durbin-Watson:**	2.103
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	286.224
Skew:	0.310	**Prob(JB):**	7.04e-63
Kurtosis:	6.836	**Cond. No.**	2.57e+03

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

[2] The condition number is large, 2.57e+03. This might indicate that there are strong multicollinearity or other numerical problems.

[ ]:

```

```

6 More

[7]:

```
# --- Flow ---
# 1. Motivation; 2. Observation; 3. Conclusion
```

[79]:

```
# --- k-fold CV for generalisation error ---
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score

diabetes = datasets.load_diabetes()
X = diabetes.data[:150]
y = diabetes.target[:150]
lasso = linear_model.Lasso()

cross_val_score(lasso, X, y, cv=3, scoring='neg_mean_squared_error')
```

[79]:

```
array([-3635.51152303, -3573.34242148, -6114.78229547])
```

[78]:

```
import sklearn
sklearn.metrics.SCORERS.keys()
```

[78]:

```
dict_keys(['explained_variance', 'r2', 'max_error', 'neg_median_absolute_error',
'neg_mean_absolute_error', 'neg_mean_absolute_percentage_error',
'neg_mean_squared_error', 'neg_mean_squared_log_error',
'neg_root_mean_squared_error', 'neg_mean_poisson_deviance',
'neg_mean_gamma_deviance', 'accuracy', 'top_k_accuracy', 'roc_auc',
'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted',
'balanced_accuracy', 'average_precision', 'neg_log_loss', 'neg_brier_score',
'adjusted_rand_score', 'rand_score', 'homogeneity_score', 'completeness_score',
'v_measure_score', 'mutual_info_score', 'adjusted_mutual_info_score',
'normalized_mutual_info_score', 'fowlkes_mallows_score', 'precision',
'precision_macro', 'precision_micro', 'precision_samples', 'precision_weighted',
'recall', 'recall_macro', 'recall_micro', 'recall_samples', 'recall_weighted',
'f1', 'f1_macro', 'f1_micro', 'f1_samples', 'f1_weighted', 'jaccard',
'jaccard_macro', 'jaccard_micro', 'jaccard_samples', 'jaccard_weighted'])
```

[88]:

```
# --- t-test example ---
# are MEDV means significantly different for each value of CHAS?
from scipy.stats import ttest_ind, ttest_1samp, ttest_rel
```

[86]:

```
df.groupby(['CHAS'])[['MEDV']].mean()
```

[86]:
```
          MEDV
CHAS
0     22.093843
1     28.440000
```

[89]:
```
ttest_ind(df[df['CHAS']==0]['MEDV'], df[df['CHAS']==1]['MEDV'])
```

[89]:
```
TtestResult(statistic=-3.996437466090509, pvalue=7.390623170519902e-05,
df=504.0)
```

[ ]:
```
# --- autoregresion ---
# Fitting an AR(p) model via OLS yields a biased estimate
```