Linear Regression Lab on the Boston and Carseats Data

Last modified: 2026-11-19

```
# The Python chunks on this page import numpy, pandas, matplotlib, scipy and statsmodels;
# reticulate provisions them.
library(reticulate)
py_require(c("numpy", "pandas", "matplotlib", "scipy", "statsmodels"))
f3 <- function(v) sprintf("%.3f", v)
# A positive number in scientific notation, for display math: 2.4\times10^{2}.
sci <- function(v) {
  e <- floor(log10(v))
  m <- signif(v / 10^e, 2)
  if (m >= 10) {
    m <- m / 10
    e <- e + 1
  }
  sprintf("%.1f\\times10^{%d}", m, e)
}
```

1 The book's Chapter 3 lab

This page holds the code lab for the Linear Regression chapter, on the Boston and Carseats data. The lab on the Auto data covers the same topics on different data.

To work the lab without the solutions, download a stub with only the exercises:

- Quarto
- Jupyter notebook[1]
- Python[2]
- R[3]

Each solution shows the code in Python, in R with tidymodels, and in base R.

2 Helper functions

Every lab and example below reports 95% confidence intervals[4]. These helpers are defined once, here, so no page depends on another page's code. Each takes all its inputs as arguments.

2.0.1 Python

```
import numpy as np


def bootstrap_corr_ci(a, b, B=20000, seed=571):
```

[1] https://morrison-lab.github.io/lds/chapters/lab-stubs/linear-regression-lab-stub.ipynb
[2] https://morrison-lab.github.io/lds/chapters/lab-stubs/linear-regression-lab-stub.py
[3] https://morrison-lab.github.io/lds/chapters/lab-stubs/linear-regression-lab-stub.R
[4] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval

```
    """95% percentile interval for the correlation of a and b.

    Resamples the cases with replacement B times,
    which mimics drawing new cases from the same population,
    and recomputes the correlation each time.
    """
    rng = np.random.default_rng(seed)
    a, b = np.asarray(a, float), np.asarray(b, float)
    corr = np.empty(B)
    for i in range(B):
        rows = rng.integers(0, len(a), len(a))
        corr[i] = np.corrcoef(a[rows], b[rows])[0, 1]
    return np.quantile(corr, [0.025, 0.975])


def bootstrap_vif_ci(features, B=20000, seed=571):
    """95% percentile interval for each feature's VIF.

    The VIF of feature j is the jth diagonal entry
    of the inverse of the features' correlation matrix,
    which gives the same value as 1 / (1 - R^2) without a regression per feature.
    """
    rng = np.random.default_rng(seed)
    F = np.asarray(features, float)
    vifs = np.empty((B, F.shape[1]))
    for b in range(B):
        rows = rng.integers(0, len(F), len(F))
        vifs[b] = np.diag(np.linalg.inv(np.corrcoef(F[rows], rowvar=False)))
    return np.quantile(vifs, [0.025, 0.975], axis=0)
```

2.0.2 R

```
# Formats an interval's endpoints for the prose
ci_text <- function(bounds, digits) {
  sprintf("%.*f to %.*f", digits, bounds[[1]], digits, bounds[[2]])
}

# 95% percentile interval for R^2 from resampled cars.
# Resampling rows with replacement mimics
# drawing a new set of cars from the same population.
# The refits use lm.fit directly, because 2,000 workflow fits would be slow.
boot_r2_ci <- function(formula, data, n_boot = 2000) {
  set.seed(571)
  design <- model.matrix(formula, data)
  y <- model.response(model.frame(formula, data))
  r2 <- vapply(seq_len(n_boot), \(b) {
    rows <- sample(nrow(design), replace = TRUE)
    sse <- sum(lm.fit(design[rows, ], y[rows])$residuals^2)
    1 - sse / sum((y[rows] - mean(y[rows]))^2)
  }, numeric(1))
  unname(quantile(r2, c(0.025, 0.975)))
}

# 95% percentile interval for the correlation of a and b
boot_cor_ci <- function(a, b, n_boot = 20000) {
```

```r
  set.seed(571)
  corr <- vapply(seq_len(n_boot), \(i) {
    rows <- sample(length(a), replace = TRUE)
    cor(a[rows], b[rows])
  }, numeric(1))
  unname(quantile(corr, c(0.025, 0.975)))
}

# 95% percentile interval for each feature's VIF.
# The VIF of feature j is the jth diagonal entry
# of the inverse of the features' correlation matrix,
# which gives the same value as 1 / (1 - R^2) without a regression per feature.
boot_vif_ci <- function(features, n_boot = 20000) {
  set.seed(571)
  vifs <- vapply(seq_len(n_boot), \(b) {
    resample <- features[sample(nrow(features), replace = TRUE), ]
    diag(solve(cor(resample)))
  }, numeric(ncol(features)))
  apply(vifs, 1, quantile, probs = c(0.025, 0.975))
}

# The 95% interval for a linear combination w'beta
# of a fitted model's coefficients:
# the estimate plus or minus a t quantile times its standard error,
# from the coefficients' covariance matrix
lincomb_ci <- function(model, w) {
  estimate <- sum(w * coef(model))
  half <- qt(0.975, df.residual(model)) * sqrt(drop(t(w) %*% vcov(model) %*% w))
  c(estimate = estimate, lower = estimate - half, upper = estimate + half)
}
```

These exercises follow the lab of [1, Sec. 3.6], on the Boston and Carseats data, step by step. The Python code follows the book's Chapter 3 lab notebook[5], which fits models with `statsmodels`. That notebook builds its design matrices with `ModelSpec` from the `ISLP` package; here we build them with `pandas`, because installing `ISLP` also installs PyTorch. The R tidymodels code follows the Chapter 3 lab of ISLR-tidymodels-labs[6].

3 Simple linear regression

Exercise 3.1 (Lab: fit a line to the Boston data)**.** Download the Boston data from `data/boston.csv`[a] and save it as `data/boston.csv` next to your code. It has 506 Boston neighborhoods from the 1970 census [2]. The copy here is the one in the Python package `ISLP` and the R package `ISLR2`. The response `medv` is the median home value, in thousands of dollars, and `lstat` is the percentage of the population with lower socioeconomic status.
Fit `medv` on `lstat`. Report each coefficient with its standard error[b].

Hint
Read the data with the data frame functions, then fit a model with a formula and read its coefficients and standard errors[c] from the results.

[a] https://morrison-lab.github.io/lds/chapters/data/boston.csv
[b] https://morrison-lab.github.io/sds/estimation.html#def-SE
[c] https://morrison-lab.github.io/sds/estimation.html#def-SE

[5] https://github.com/intro-stat-learning/ISLP_labs/blob/main/Ch03-linreg-lab.ipynb
[6] https://github.com/Morrison-Lab/ISLR-tidymodels-labs/blob/main/03-linear-regression.qmd

Solution

Solution 3.1.

3.0.1 Python

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm

# ISLP's load_data("Boston") reads this same file.
Boston = pd.read_csv("data/boston.csv")
medv = Boston["medv"]

X_simple = pd.DataFrame({"intercept": np.ones(Boston.shape[0]),
                         "lstat": Boston["lstat"]})
boston_simple = sm.OLS(medv, X_simple).fit()
# Each coefficient with its standard error and 95% confidence interval
print(pd.concat([boston_simple.params, boston_simple.bse, boston_simple.conf_int()], axis=1)
      .set_axis(["coef", "std err", "lower", "upper"], axis=1))
```

```
                coef   std err      lower      upper
intercept  34.553841  0.562627  33.448457  35.659225
lstat      -0.950049  0.038733  -1.026148  -0.873951
```

3.0.2 R (tidymodels)

```r
library(tidymodels)

boston <- read.csv("data/boston.csv")
lm_spec <- linear_reg() |> set_engine("lm")

tm_boston_simple <- lm_spec |> fit(medv ~ lstat, data = boston)
# Each coefficient with its standard error and 95% confidence interval
tidy(tm_boston_simple, conf.int = TRUE)
```

```
# A tibble: 2 x 7
  term        estimate std.error statistic   p.value conf.low conf.high
  <chr>          <dbl>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>
1 (Intercept)   34.6      0.563       61.4 3.74e-236    33.4     35.7
2 lstat         -0.950    0.0387     -24.5 5.08e- 88    -1.03    -0.874
```

3.0.3 R (base)

```r
boston <- read.csv("data/boston.csv")
lm_boston_simple <- lm(medv ~ lstat, data = boston)
# The table includes each coefficient's standard error
# and 95% confidence interval
parameters::parameters(lm_boston_simple) |>
  parameters::print_md(digits = 3)
```

Parameter	Coefficient	SE	95% CI	t(504)	p
(Intercept)	34.554	0.563	(33.448, 35.659)	61.415	$< .001$
lstat	-0.950	0.039	(-1.026, -0.874)	-24.528	$< .001$

The fitted line is $\widehat{\texttt{medv}} = 34.55 - 0.95 \times \texttt{lstat}$, with standard errors 0.563 and 0.039. The 95% confidence intervals, the estimate plus or minus a t quantile times its standard error, are 33.45 to 35.66 for the intercept and -1.026 to -0.874 for the slope. Each extra percentage point of `lstat` goes with a median home value about 950 dollars lower (95% confidence interval 874 to 1026 dollars lower).

Exercise 3.2 (Lab: fit a line to the Boston data (part 2))**.** Use the setup of Exercise 3.1.
Predict `medv` for `lstat` values 5, 10 and 15.

Hint

A fitted model predicts at new values of the feature when you pass them in a data frame with the same column name.

Solution

Solution 3.2.

3.0.4 Python

```
new_X = pd.DataFrame({"intercept": np.ones(3), "lstat": [5, 10, 15]})
new_predictions = boston_simple.get_prediction(new_X)
print(new_predictions.predicted_mean)
```

```
[29.80359411 25.05334734 20.30310057]
```

3.0.5 R (tidymodels)

```
new_lstat <- tibble(lstat = c(5, 10, 15))
predict(tm_boston_simple, new_data = new_lstat)
```

```
# A tibble: 3 x 1
  .pred
  <dbl>
1  29.8
2  25.1
3  20.3
```

3.0.6 R (base)

```
new_lstat <- tibble::tibble(lstat = c(5, 10, 15))
predict(lm_boston_simple, new_lstat)
```

```
       1        2        3
29.80359 25.05335 20.30310
```

At `lstat` = 10, the predicted `medv` is 25.05.

Exercise 3.3 (Lab: fit a line to the Boston data (part 3))**.** Use the setup of Exercise 3.1.
Give the 95% confidence interval[a] for the mean[b] response at each value.

Hint

The prediction object of the fit holds the confidence interval[c] of the mean[d] response at each new value.

[a] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval

[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean
[c] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval
[d] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean

Solution

Solution 3.3.

3.0.7 Python

```
print(new_predictions.conf_int(alpha=0.05))
```

```
[[29.00741194 30.59977628]
 [24.47413202 25.63256267]
 [19.73158815 20.87461299]]
```

3.0.8 R (tidymodels)

```
predict(tm_boston_simple, new_data = new_lstat, type = "conf_int")
```

```
# A tibble: 3 x 2
  .pred_lower .pred_upper
        <dbl>       <dbl>
1        29.0        30.6
2        24.5        25.6
3        19.7        20.9
```

3.0.9 R (base)

```
predict(lm_boston_simple, new_lstat, interval = "confidence")
```

```
       fit      lwr      upr
1 29.80359 29.00741 30.59978
2 25.05335 24.47413 25.63256
3 20.30310 19.73159 20.87461
```

The confidence interval[a] for the mean[b] response at `lstat` $= 10$ is (24.47, 25.63).

[a] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample-mean

Exercise 3.4 (Lab: fit a line to the Boston data (part 4))**.** Use the setup of Exercise 3.1. Give the 95% prediction interval at each value.

Hint

The prediction object of the fit also holds the prediction interval at each new value.

Solution

Solution 3.4.

3.0.10 Python

```
print(new_predictions.conf_int(obs=True, alpha=0.05))
```

```
[[17.56567478 42.04151344]
 [12.82762635 37.27906833]
 [ 8.0777421  32.52845905]]
```

3.0.11 R (tidymodels)

```
predict(tm_boston_simple, new_data = new_lstat, type = "pred_int")
```

```
# A tibble: 3 x 2
  .pred_lower .pred_upper
        <dbl>       <dbl>
1       17.6         42.0
2       12.8         37.3
3        8.08        32.5
```

3.0.12 R (base)

```
predict(lm_boston_simple, new_lstat, interval = "prediction")
```

```
       fit       lwr      upr
1 29.80359 17.565675 42.04151
2 25.05335 12.827626 37.27907
3 20.30310  8.077742 32.52846
```

The prediction interval at `lstat` = 10 is (12.83, 37.28).

Exercise 3.5 (Lab: fit a line to the Boston data (part 5))**.** Use the setup of Exercise 3.1. Why is the prediction interval wider?

Hint

Compare what the confidence interval[a] covers with what the prediction interval covers.

[a] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval

Solution

Solution 3.5. The confidence interval for the mean covers only the estimation error[a] in the line. The prediction interval also covers the noise of one new neighborhood around that line, so it is much wider.

[a] https://morrison-lab.github.io/sds/estimation.html#def-estimation-error

Exercise 3.6 (Lab: fit a line to the Boston data (part 6))**.** Use the setup of Exercise 3.1. Draw a scatter plot of `medv` against `lstat` with the fitted line.

Hint

The plotting functions draw the scatter plot, and a fitted line over the data comes from predictions on a grid of feature values.

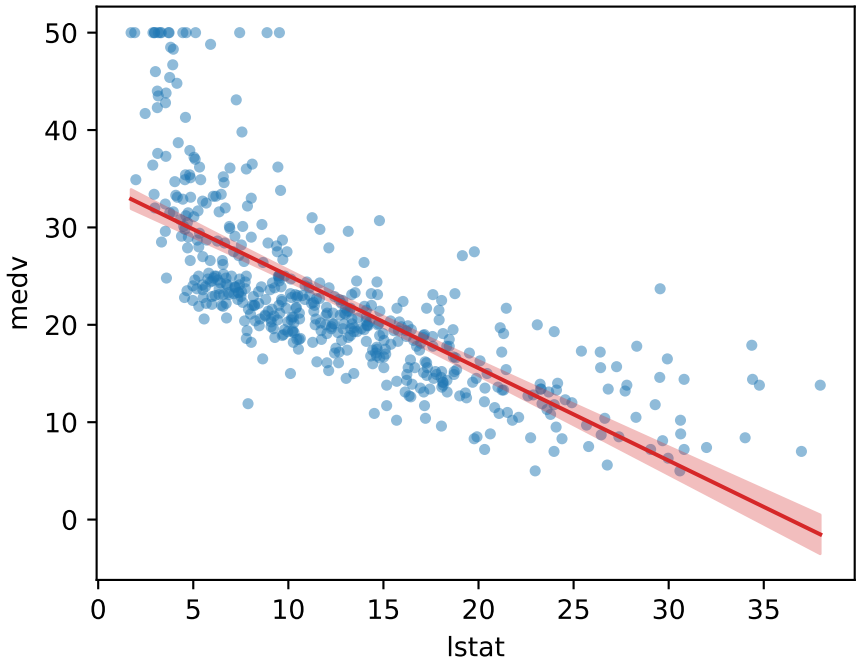
Solution

Solution 3.6.

3.0.13 Python

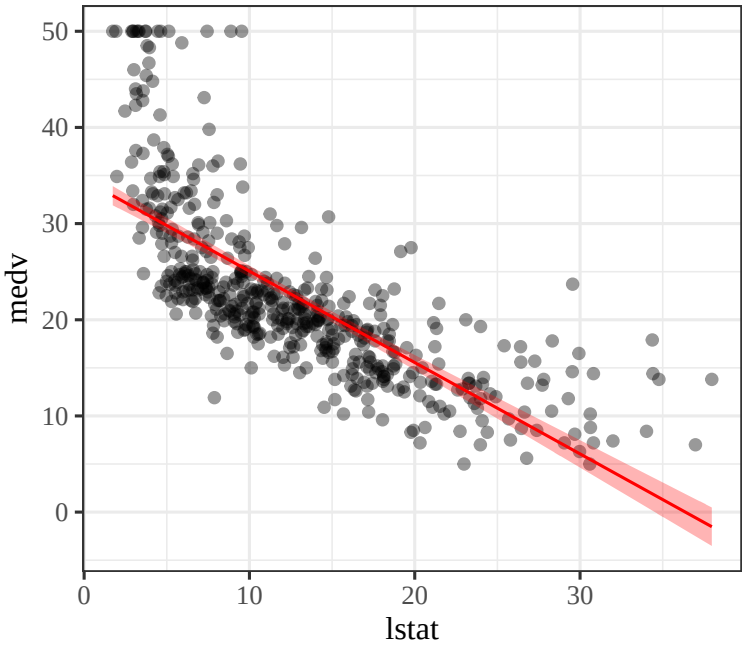
```
import matplotlib.pyplot as plt

lstat_grid = np.linspace(Boston["lstat"].min(), Boston["lstat"].max(), 100)
# The band is the 95% confidence interval for the mean medv at each lstat
band = boston_simple.get_prediction(
    pd.DataFrame({"intercept": 1.0, "lstat": lstat_grid})).conf_int(alpha=0.05)
fig, ax = plt.subplots(figsize=(4.5, 3.5), layout="constrained")
ax.scatter(Boston["lstat"], medv, s=10, alpha=0.5);
ax.fill_between(lstat_grid, band[:, 0], band[:, 1], color="C3", alpha=0.3);
ax.plot(lstat_grid,
        boston_simple.params["intercept"] + boston_simple.params["lstat"] * lstat_grid,
        color="C3");
ax.set_xlabel("lstat");
ax.set_ylabel("medv");
plt.show()
```



3.0.14 R (tidymodels)

```
# The band is the 95% confidence interval for the mean medv at each lstat
lstat_grid <- tibble(lstat = seq(min(boston$lstat), max(boston$lstat),
                                 length.out = 100))
boston_band <- bind_cols(
  lstat_grid,
  predict(tm_boston_simple, new_data = lstat_grid),
  predict(tm_boston_simple, new_data = lstat_grid, type = "conf_int")
)
ggplot() +
  geom_point(data = boston, aes(lstat, medv), alpha = 0.4) +
  geom_ribbon(data = boston_band,
              aes(lstat, ymin = .pred_lower, ymax = .pred_upper),
              fill = "red", alpha = 0.3) +
  geom_line(data = boston_band, aes(lstat, .pred), colour = "red")
```



3.0.15 R (base)

```
sjPlot::plot_model(lm_boston_simple, type = "pred", terms = "lstat",
                   show.data = TRUE)
```

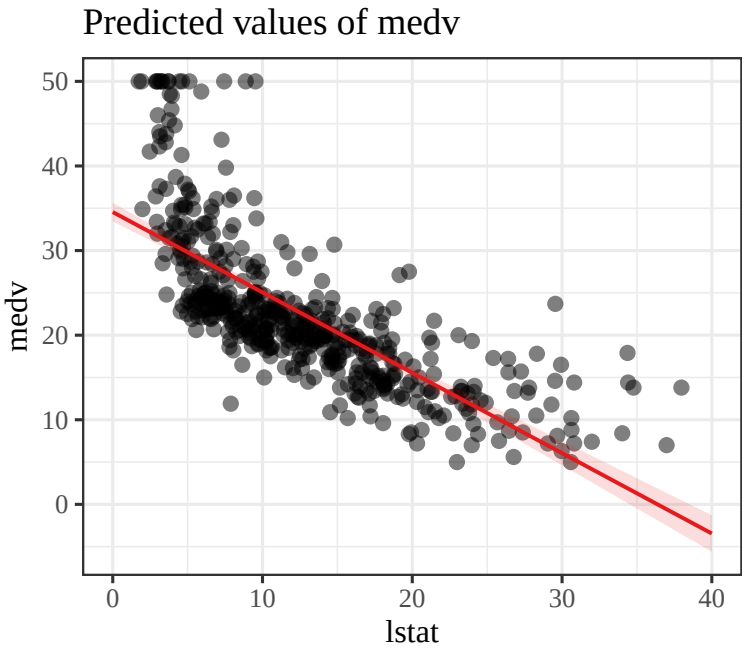


Figure 1: Typical home value against `lstat` for the Boston data, with a straight line inside a shaded band.

In Figure 1, the line falls as `lstat` rises, and the points curve away from it at both ends.

Exercise 3.7 (Lab: fit a line to the Boston data (part 7))**.** Use the setup of Exercise 3.1. Draw the residual plot.

Hint

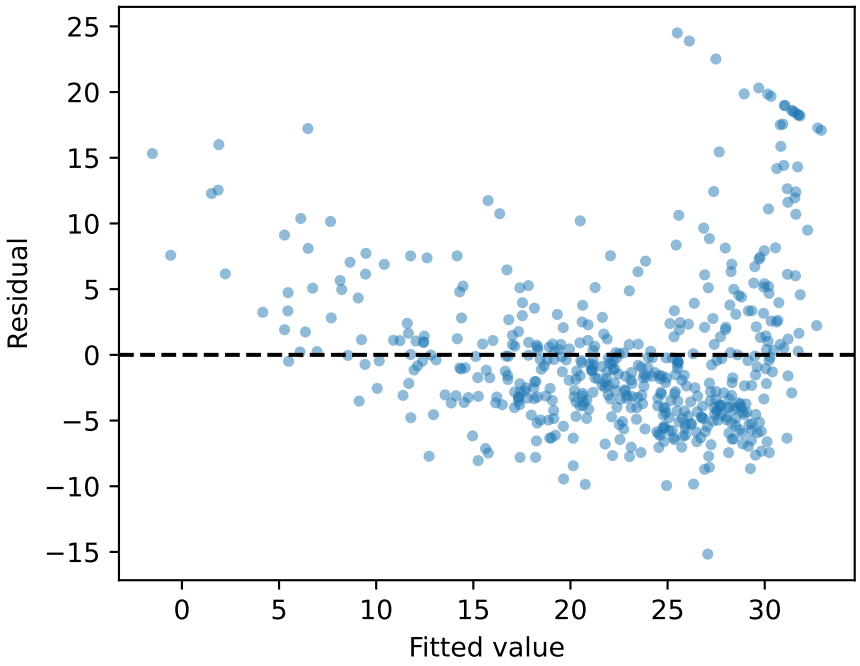
The residuals and fitted values of a fit are the two axes of the residual plot.

Solution

Solution 3.7.

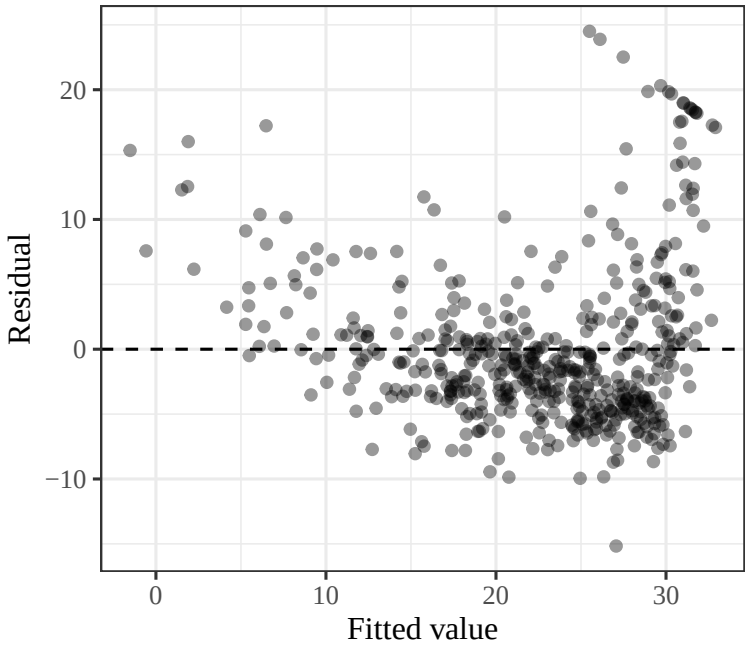
3.0.16 Python

```
fig, ax = plt.subplots(figsize=(4.5, 3.5), layout="constrained")
ax.scatter(boston_simple.fittedvalues, boston_simple.resid, s=10, alpha=0.5);
ax.axhline(0, color="k", linestyle="--");
ax.set_xlabel("Fitted value");
ax.set_ylabel("Residual");
plt.show()
```



3.0.17 R (tidymodels)

```
augment(tm_boston_simple, new_data = boston) |>
  ggplot(aes(.pred, .resid)) +
  geom_point(alpha = 0.4) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(x = "Fitted value", y = "Residual")
```



3.0.18 R (base)

```
performance::check_model(lm_boston_simple, check = "linearity") |>
  plot()
```

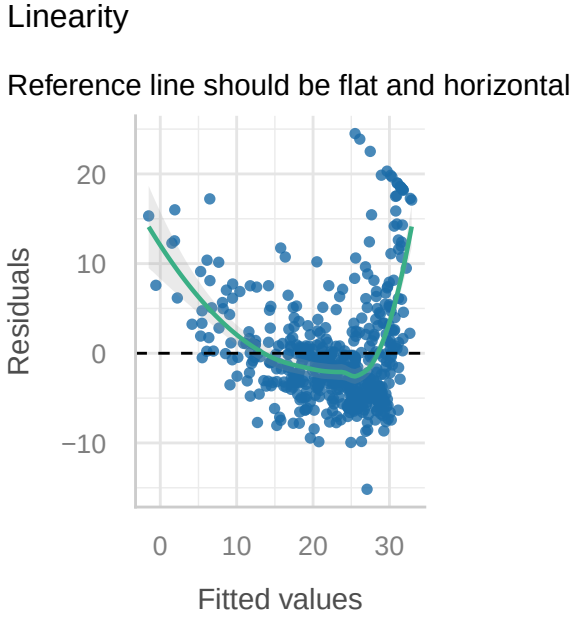


Figure 2: The gaps between the data and the line, against the height the line gives, for the fit of `medv` on `lstat`.

In Figure 2, the residual plot is a U shape.

Exercise 3.8 (Lab: fit a line to the Boston data (part 8))**.** Use the setup of Exercise 3.1. What do the plots say about a straight line for these data?

Hint

A straight line suits the data when the residual plot shows no pattern.

Solution

Solution 3.8. The points curve: they lie above the line at both ends and below it in the middle. The residual plot shows the same pattern as a U shape, the sign of non-linearity in a residual plot. Exercise 6.1 adds a squared term for it.

4 Multiple linear regression

Exercise 4.1 (Lab: many features on the Boston data)**.** Fit `medv` on `lstat` and `age`, the percentage of owner-occupied homes built before 1940. How does the `lstat` coefficient compare with that of Exercise 3.1?

Hint

A formula names each feature after a `+`, and the coefficient of `lstat` is in the results.

Solution

Solution 4.1.

4.0.1 Python

```
import pandas as pd
import statsmodels.api as sm

X_lstat_age = sm.add_constant(Boston[["lstat", "age"]])
boston_lstat_age = sm.OLS(medv, X_lstat_age).fit()
print(boston_lstat_age.summary2().tables[1])
```

```
            Coef.  Std.Err.          t          P>|t|     [0.025     0.975]
const   33.222761  0.730847  45.457881  2.943785e-180  31.786872  34.658650
lstat   -1.032069  0.048191 -21.416330   8.419554e-73  -1.126748  -0.937389
age      0.034544  0.012225   2.825605   4.906776e-03   0.010525   0.058564
```

4.0.2 R (tidymodels)

```
library(tidymodels)

# Each coefficient with its standard error and 95% confidence interval
tm_boston_lstat_age <- lm_spec |> fit(medv ~ lstat + age, data = boston)
tidy(tm_boston_lstat_age, conf.int = TRUE)
```

```
# A tibble: 3 x 7
  term        estimate std.error statistic   p.value conf.low conf.high
  <chr>          <dbl>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>
```

```
1 (Intercept)  33.2       0.731      45.5  2.94e-180  31.8      34.7
2 lstat        -1.03      0.0482    -21.4  8.42e- 73  -1.13     -0.937
3 age           0.0345    0.0122      2.83 4.91e-  3   0.0105    0.0586
```

4.0.3 R (base)

```
# The table includes each coefficient's standard error
# and 95% confidence interval
lm_boston_lstat_age <- lm(medv ~ lstat + age, data = boston)
parameters::parameters(lm_boston_lstat_age) |>
  parameters::print_md(digits = 3)
```

Parameter	Coefficient	SE	95% CI	t(503)	p
(Intercept)	33.223	0.731	(31.787, 34.659)	45.458	< .001
lstat	-1.032	0.048	(-1.127, -0.937)	-21.416	< .001
age	0.035	0.012	(0.011, 0.059)	2.826	0.005

The `lstat` coefficient moves from -0.950 (95% confidence interval -1.026 to -0.874) to -1.032 (-1.127 to -0.937). It is now the change in `medv` per point of `lstat` among neighborhoods with the same `age`.

Exercise 4.2 (Lab: many features on the Boston data (part 2))**.** Use the setup of Exercise 4.1. Fit `medv` on all 12 other columns.

> **Hint**
> A formula can list the other columns by name, or build the formula as text from a list of the column names.

Solution

Solution 4.2.

4.0.4 Python

```
X_all = sm.add_constant(Boston.drop(columns="medv"))
boston_all = sm.OLS(medv, X_all).fit()
print(boston_all.summary2().tables[1])
```

```
             Coef.  Std.Err.          t         P>|t|      [0.025      0.975]
const    41.617270  4.936039   8.431308  3.794467e-16  31.919001  51.315539
crim     -0.121389  0.033000  -3.678399  2.605302e-04  -0.186227  -0.056550
zn        0.046963  0.013879   3.383750  7.719966e-04   0.019694   0.074233
indus     0.013468  0.062145   0.216715  8.285201e-01  -0.108633   0.135569
chas      2.839993  0.870007   3.264333  1.173458e-03   1.130614   4.549373
nox     -18.758022  3.851355  -4.870500  1.502073e-06 -26.325116 -11.190928
rm        3.658119  0.420246   8.704714  4.808025e-17   2.832425   4.483813
age       0.003611  0.013329   0.270882  7.865948e-01  -0.022579   0.029800
dis      -1.490754  0.201623  -7.393779  6.170894e-13  -1.886899  -1.094608
rad       0.289405  0.066908   4.325417  1.843595e-05   0.157945   0.420864
tax      -0.012682  0.003801  -3.336504  9.124487e-04  -0.020150  -0.005214
ptratio  -0.937533  0.132206  -7.091443  4.630167e-12  -1.197290  -0.677776
lstat    -0.552019  0.050659 -10.896814  6.392118e-25  -0.651553  -0.452485
```

4.0.5 R (tidymodels)

```
tm_boston_all <- lm_spec |> fit(medv ~ ., data = boston)
tidy(tm_boston_all, conf.int = TRUE)
```

```
# A tibble: 13 x 7
   term         estimate std.error statistic  p.value conf.low conf.high
   <chr>           <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
 1 (Intercept)  41.6       4.94        8.43  3.79e-16  31.9     51.3
 2 crim         -0.121     0.0330     -3.68  2.61e- 4  -0.186   -0.0565
 3 zn            0.0470    0.0139      3.38  7.72e- 4   0.0197   0.0742
 4 indus         0.0135    0.0621      0.217 8.29e- 1  -0.109    0.136
 5 chas          2.84      0.870       3.26  1.17e- 3   1.13     4.55
 6 nox         -18.8       3.85       -4.87  1.50e- 6 -26.3    -11.2
 7 rm            3.66      0.420       8.70  4.81e-17   2.83     4.48
 8 age           0.00361   0.0133      0.271 7.87e- 1  -0.0226   0.0298
 9 dis          -1.49      0.202      -7.39  6.17e-13  -1.89    -1.09
10 rad           0.289     0.0669      4.33  1.84e- 5   0.158    0.421
11 tax          -0.0127    0.00380    -3.34  9.12e- 4  -0.0202  -0.00521
12 ptratio      -0.938     0.132      -7.09  4.63e-12  -1.20    -0.678
13 lstat        -0.552     0.0507    -10.9   6.39e-25  -0.652   -0.452
```

4.0.6 R (base)

```
lm_boston_all <- lm(medv ~ ., data = boston)
parameters::parameters(lm_boston_all) |>
  parameters::print_md(digits = 3)
```

Parameter	Coefficient	SE	95% CI	t(493)	p
(Intercept)	41.617	4.936	(31.919, 51.316)	8.431	< .001
crim	-0.121	0.033	(-0.186, -0.057)	-3.678	< .001
zn	0.047	0.014	(0.020, 0.074)	3.384	< .001
indus	0.013	0.062	(-0.109, 0.136)	0.217	0.829
chas	2.840	0.870	(1.131, 4.549)	3.264	0.001
nox	-18.758	3.851	(-26.325, -11.191)	-4.870	< .001
rm	3.658	0.420	(2.832, 4.484)	8.705	< .001
age	0.004	0.013	(-0.023, 0.030)	0.271	0.787
dis	-1.491	0.202	(-1.887, -1.095)	-7.394	< .001
rad	0.289	0.067	(0.158, 0.421)	4.325	< .001
tax	-0.013	0.004	(-0.020, -0.005)	-3.337	< .001
ptratio	-0.938	0.132	(-1.197, -0.678)	-7.091	< .001
lstat	-0.552	0.051	(-0.652, -0.452)	-10.897	< .001

The fit has 12 features besides the intercept[a].

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept

Exercise 4.3 (Lab: many features on the Boston data (part 3))**.** Use the setup of Exercise 4.1. Which coefficients have p-values above 0.05?

Hint

The summary of the fit holds a p-value[a] for each coefficient, and a mask keeps those above 0.05.

[a] https://morrison-lab.github.io/sds/inference.html#def-p-value

Solution

Solution 4.3. `indus` and `age` have p-values above 0.05. With the other features in the model, the data give no clear sign that either one adds anything.

Exercise 4.4 (Lab: many features on the Boston data (part 4))**.** Use the setup of Exercise 4.1. Compute the variance inflation factor for each feature of the full model.

Hint

The variance inflation factor of each feature comes from the design matrix of the fit.

Solution

Solution 4.4.

4.0.7 Python

```
from statsmodels.stats.outliers_influence import variance_inflation_factor as VIF

vif = pd.DataFrame({"vif": [VIF(X_all, i) for i in range(1, X_all.shape[1])]},
                   index=X_all.columns[1:])

# A VIF has no formula for its interval, so bootstrap_vif_ci
# (in the interval helpers) resamples the neighborhoods
vif[["lower", "upper"]] = bootstrap_vif_ci(Boston.drop(columns="medv")).T
print(vif)
```

```
              vif     lower      upper
crim     1.767486  1.555240   2.553470
zn       2.298459  2.026309   2.745879
indus    3.987181  3.390477   5.125450
chas     1.071168  1.049384   1.158832
nox      4.369093  4.046508   5.051079
rm       1.912532  1.670261   2.353259
age      3.088232  2.755750   3.634929
dis      3.954037  3.566747   4.638972
rad      7.445301  6.198520   9.890067
tax      9.002158  7.247533  12.301629
ptratio  1.797060  1.635727   2.078554
lstat    2.870777  2.453071   3.642755
```

4.0.8 R (tidymodels)

```
# A VIF has no formula for its interval, so boot_vif_ci
# (in the interval helpers) resamples the neighborhoods

# VIF_j = 1 / (1 - R^2) from regressing feature j on the other features
predictors <- setdiff(names(boston), "medv")
vif_interval <- boot_vif_ci(boston[predictors])
tibble(
  feature = predictors,
  vif = map_dbl(predictors, \(feature) {
    r_squared <- lm_spec |>
      fit(reformulate(setdiff(predictors, feature), feature), data = boston) |>
      glance() |>
      pull(r.squared)
    1 / (1 - r_squared)
  }),
  lower = vif_interval[1, ],
  upper = vif_interval[2, ]
)
```

```
# A tibble: 12 x 4
   feature   vif lower upper
   <chr>   <dbl> <dbl> <dbl>
 1 crim     1.77  1.55  2.55
 2 zn       2.30  2.02  2.74
 3 indus    3.99  3.39  5.11
 4 chas     1.07  1.05  1.16
 5 nox      4.37  4.05  5.05
 6 rm       1.91  1.66  2.34
 7 age      3.09  2.75  3.64
 8 dis      3.95  3.57  4.65
 9 rad      7.45  6.18  9.91
10 tax      9.00  7.22 12.2
11 ptratio  1.80  1.63  2.08
12 lstat    2.87  2.45  3.63
```

4.0.9 R (base)

```
# A VIF has no formula for its interval, so boot_vif_ci
# (in the interval helpers) resamples the neighborhoods

# VIF_j = 1 / (1 - R^2) from regressing feature j on the other features
predictors <- setdiff(names(boston), "medv")
vif_boston <- sapply(predictors, function(feature) {
  others <- reformulate(setdiff(predictors, feature), feature)
  1 / (1 - summary(lm(others, data = boston))$r.squared)
})
vif_boston
```

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

```
vif_boston_ci <- boot_vif_ci(boston[predictors])
t(vif_boston_ci)
```

```
            2.5%     97.5%
crim    1.552895  2.550535
```

```
zn      2.020795  2.744689
indus   3.394594  5.108599
chas    1.048955  1.159317
nox     4.045030  5.049063
rm      1.662013  2.343394
age     2.752418  3.635210
dis     3.570608  4.649746
rad     6.181721  9.913714
tax     7.218113 12.244461
ptratio 1.634732  2.081573
lstat   2.446677  3.627501
```

The variance inflation factors of the model with every feature run from 1.07 to 9.00.

Exercise 4.5 (Lab: many features on the Boston data (part 5))**.** Use the setup of Exercise 4.1. Which feature has the largest, and what is it?

Hint

Look for the largest of the variance inflation factors, and name the feature that has it.

Solution

Solution 4.5.

```
# The same resampling gives an interval for the correlation of two features.
# boot_cor_ci() is one of the interval helpers
cor(boston$tax, boston$rad)
```

```
[1] 0.9102282
```

```
boot_cor_ci(boston$tax, boston$rad)
```

```
[1] 0.8804014 0.9349518
```

The largest variance inflation factor is 9.00 (95% bootstrap interval 7.22 to 12.24), for `tax`, the property tax rate. Its coefficient has about 9 times the variance[a] it would have if that feature were uncorrelated with the others. The next largest is 7.45 (6.18 to 9.91), for `rad`, the access to radial highways. The two are strongly correlated with each other (correlation[b] 0.91, 95% bootstrap interval 0.88 to 0.93). Each bootstrap interval is the 2.5th to 97.5th percentile over 20,000 resamples of the 506 neighborhoods, with the statistic recomputed on each resample. Confidence intervals[c] for the coefficients in the tables above are t intervals, the estimate[d] plus or minus a t quantile times its standard error[e].

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-correlation
[c] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval
[d] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[e] https://morrison-lab.github.io/sds/estimation.html#def-SE

5 Interaction terms

Exercise 5.1 (Lab: an interaction on the Boston data)**.** Fit `medv` on `lstat`, `age` and their product lstat × age, the interaction of the two features. Report the coefficients with their p-values.

Hint

In a formula, `*` adds both features and their product.

Solution

Solution 5.1.

5.0.1 Python

```
import statsmodels.api as sm

X_interaction = sm.add_constant(Boston[["lstat", "age"]])
X_interaction["lstat:age"] = Boston["lstat"] * Boston["age"]
boston_interaction = sm.OLS(medv, X_interaction).fit()
print(boston_interaction.summary2().tables[1])
```

```
              Coef.  Std.Err.          t         P>|t|     [0.025     0.975]
const      36.088536  1.469835  24.552773  4.907116e-88  33.200749  38.976323
lstat      -1.392117  0.167456  -8.313352  8.780730e-16  -1.721117  -1.063117
age        -0.000721  0.019879  -0.036262  9.710878e-01  -0.039777   0.038336
lstat:age   0.004156  0.001852   2.244283  2.524911e-02   0.000518   0.007794
```

5.0.2 R (tidymodels)

```
library(tidymodels)

interaction_recipe <- recipe(medv ~ lstat + age, data = boston) |>
  step_interact(~ lstat:age)
tm_boston_interaction <- workflow() |>
  add_recipe(interaction_recipe) |>
  add_model(lm_spec) |>
  fit(boston)
# Each coefficient with its standard error and 95% confidence interval
tidy(tm_boston_interaction, conf.int = TRUE)
```

```
# A tibble: 4 x 7
  term         estimate std.error statistic  p.value  conf.low conf.high
  <chr>           <dbl>     <dbl>     <dbl>    <dbl>     <dbl>     <dbl>
1 (Intercept) 36.1        1.47      24.6    4.91e-88 33.2       39.0
2 lstat       -1.39       0.167     -8.31   8.78e-16 -1.72      -1.06
3 age         -0.000721   0.0199    -0.0363 9.71e- 1 -0.0398     0.0383
4 lstat_x_age  0.00416    0.00185    2.24   2.52e- 2  0.000518   0.00779
```

5.0.3 R (base)

```
lm_boston_interaction <- lm(medv ~ lstat * age, data = boston)
# The table includes each coefficient's standard error
# and 95% confidence interval
parameters::parameters(lm_boston_interaction) |>
  parameters::print_md(digits = 3)
```

Parameter	Coefficient	SE	95% CI	t(502)	p
(Intercept)	36.089	1.470	(33.201, 38.976)	24.553	$< .001$
lstat	-1.392	0.167	(-1.721, -1.063)	-8.313	$< .001$
age	-7.209e-04	0.020	(-0.040, 0.038)	-0.036	0.971
lstat × age	0.004	0.002	(5.177e-04, 0.008)	2.244	0.025

The interaction coefficient is 0.0042 (95% confidence interval 0.0005 to 0.0078), with p-value 0.025.

Exercise 5.2 (Lab: an interaction on the Boston data (part 2))**.** Use the setup of Exercise 5.1. Is there evidence that the slope[a] on `lstat` depends on `age`?

Hint
Read the p-value[b] of the interaction term in the summary of the fit.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[b] https://morrison-lab.github.io/sds/inference.html#def-p-value

Solution

Solution 5.2.

```
# The slope on lstat, beta_1 + beta_3 age,
# is a linear combination of the coefficients,
# so its 95% interval comes from lincomb_ci(),
# one of the interval helpers,
# with the weights (0, 1, 0, age)
slope_age_20 <- lincomb_ci(lm_boston_interaction, c(0, 1, 0, 20))
slope_age_90 <- lincomb_ci(lm_boston_interaction, c(0, 1, 0, 90))
```

That is modest evidence that the slope[a] on `lstat` depends on `age`. The fitted slope on `lstat` is $\hat{\beta}_1 + \hat{\beta}_3 \cdot \text{age}$. It is about -1.31 (95% confidence interval[b] -1.57 to -1.05) in a neighborhood where 20% of homes were built before 1940, and about -1.02 (-1.11 to -0.92) where 90% were. Each interval is a t interval, the estimate[c] plus or minus a t quantile times its standard error[d], and the standard error of the slope comes from the coefficients' estimated[e] covariance matrix[f]. The main effect of `age` has p-value 0.97, but the hierarchy principle keeps it in the model alongside its interaction.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope
[b] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval
[c] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[d] https://morrison-lab.github.io/sds/estimation.html#def-SE
[e] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[f] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x

6 Non-linear terms

Exercise 6.1 (Lab: a curve in `lstat`)**.** Add lstat^2 to the model of Exercise 4.1, so the features are `lstat`, lstat^2 and `age`. Report the coefficients with their p-values.

Hint
Wrap the square in `I(...)` inside the formula so that `**` is read as a power. In R, write `^` instead of `**`.

Solution

Solution 6.1.

6.0.1 Python

```
import statsmodels.api as sm

X_quadratic = sm.add_constant(Boston[["lstat", "age"]])
X_quadratic["lstat^2"] = Boston["lstat"] ** 2
boston_quadratic = sm.OLS(medv, X_quadratic).fit()
print(boston_quadratic.summary2().tables[1])
```

```
             Coef.  Std.Err.          t          P>|t|     [0.025     0.975]
const    41.288537  0.873212  47.283544  4.859728e-187  39.572938  43.004137
lstat    -2.688300  0.131122 -20.502230   2.569236e-68  -2.945917  -2.430684
age       0.070254  0.010857   6.471052   2.316450e-10   0.048924   0.091585
```

```
lstat^2   0.049489  0.003717  13.314608   7.468491e-35   0.042186   0.056791
```

6.0.2 R (tidymodels)

```
library(tidymodels)

quadratic_recipe <- recipe(medv ~ lstat + age, data = boston) |>
  step_mutate(lstat2 = lstat^2)
tm_boston_quadratic <- workflow() |>
  add_recipe(quadratic_recipe) |>
  add_model(lm_spec) |>
  fit(boston)
# Each coefficient with its standard error and 95% confidence interval
tidy(tm_boston_quadratic, conf.int = TRUE)
```

```
# A tibble: 4 x 7
  term        estimate std.error statistic   p.value conf.low conf.high
  <chr>          <dbl>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>
1 (Intercept)  41.3      0.873       47.3  4.86e-187  39.6      43.0
2 lstat        -2.69     0.131      -20.5  2.57e- 68  -2.95     -2.43
3 age           0.0703   0.0109       6.47 2.32e- 10   0.0489    0.0916
4 lstat2        0.0495   0.00372     13.3  7.47e- 35   0.0422    0.0568
```

6.0.3 R (base)

```
lm_boston_quadratic <- lm(medv ~ lstat + I(lstat^2) + age, data = boston)
# The table includes each coefficient's standard error
# and 95% confidence interval
parameters::parameters(lm_boston_quadratic) |>
  parameters::print_md(digits = 3)
```

Parameter	Coefficient	SE	95% CI	t(502)	p
(Intercept)	41.289	0.873	(39.573, 43.004)	47.284	$< .001$
lstat	-2.688	0.131	(-2.946, -2.431)	-20.502	$< .001$
lstat^2	0.049	0.004	(0.042, 0.057)	13.315	$< .001$
age	0.070	0.011	(0.049, 0.092)	6.471	$< .001$

The coefficient of lstat^2 is 0.0495 (95% confidence interval 0.0422 to 0.0568, the estimate plus or minus a t quantile times its standard error), with p-value 7.5×10^{-35}. The positive sign means the fitted curve flattens as `lstat` grows, as the scatter plot of Solution 3.6 suggested.

Exercise 6.2 (Lab: a curve in `lstat` (part 2))**.** Use the setup of Exercise 6.1.
Compare the two models with the partial F-test.

> **Hint**
> The `anova_lm` function compares two nested models with an F test. In R, `anova()` does the same.

Solution

Solution 6.2.

6.0.4 Python

```
from statsmodels.stats.anova import anova_lm

print(anova_lm(boston_lstat_age, boston_quadratic))
```

```
   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
```

6.0.5 R (tidymodels)

```
# anova() compares models only when their responses have the same name,
# so fit the smaller model through a workflow too.
tm_boston_lstat_age_wf <- workflow() |>
  add_recipe(recipe(medv ~ lstat + age, data = boston)) |>
  add_model(lm_spec) |>
  fit(boston)
anova(extract_fit_engine(tm_boston_lstat_age_wf),
      extract_fit_engine(tm_boston_quadratic)) |>
  tidy()
```

```
# A tibble: 2 x 7
  term                      df.residual    rss    df sumsq statistic   p.value
  <chr>                           <dbl>  <dbl> <dbl> <dbl>     <dbl>     <dbl>
1 ..y ~ lstat + age                 503 19168.    NA   NA        NA  NA
2 ..y ~ lstat + age + lstat2        502 14166.     1 5003.      177.  7.47e-35
```

6.0.6 R (base)

```
anova_boston <- anova(lm_boston_lstat_age, lm_boston_quadratic)
anova_boston |>
  parameters::parameters() |>
  parameters::print_md(footer = "")
```

Parameter	RSS	Sum_Squares	df	df (error)	Mean_Square	F	p
Model 1: medv ~ lstat + age	19168.13			503			
Model 2: medv ~ lstat + I(lstat^2) + age	14165.61	5002.52	1	502	5002.52	177.28	< .001

The partial F-statistic, with $q = 1$, is 177.3, with p-value 7.5×10^{-35}.

Exercise 6.3 (Lab: a curve in `lstat` (part 3))**.** Use the setup of Exercise 6.1.
What does it say about the squared term?

Hint

Read the p-value[a] of the F test: a small value says the larger model fits better than the smaller one.

[a] https://morrison-lab.github.io/sds/inference.html#def-p-value

Solution

Solution 6.3. The squared term clearly improves the fit. Since $q = 1$, the F-statistic is the square of the t-statistic of lstat^2 ($13.31^2 = 177.3$).

Exercise 6.4 (Lab: a curve in `lstat` (part 4))**.** Use the setup of Exercise 6.1.
Draw the residual plot of the new model.

Hint

The residuals and fitted values of the new fit are the two axes of its residual plot.

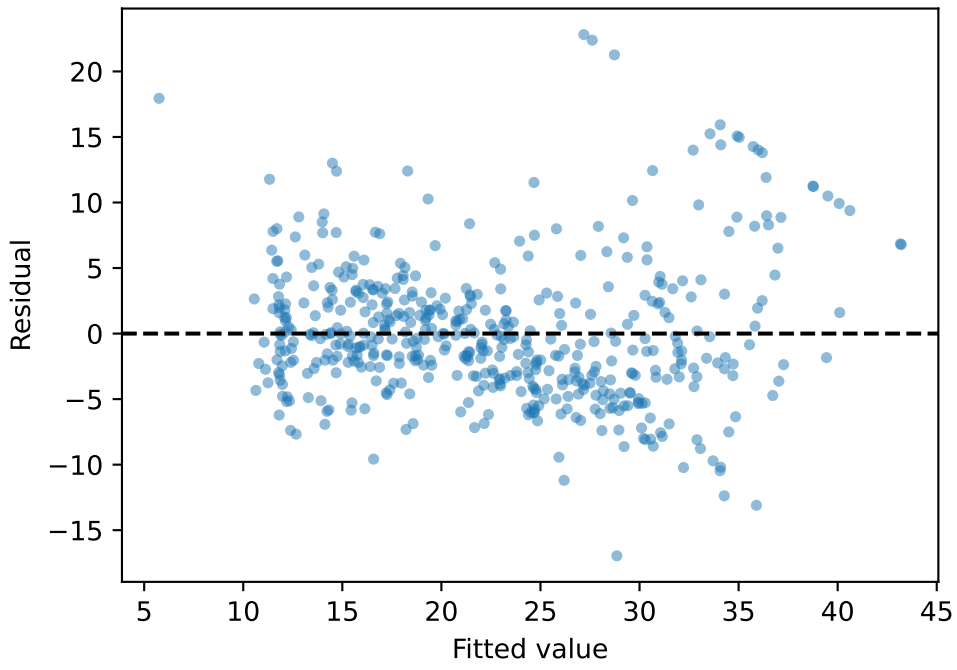
Solution

Solution 6.4.

6.0.7 Python

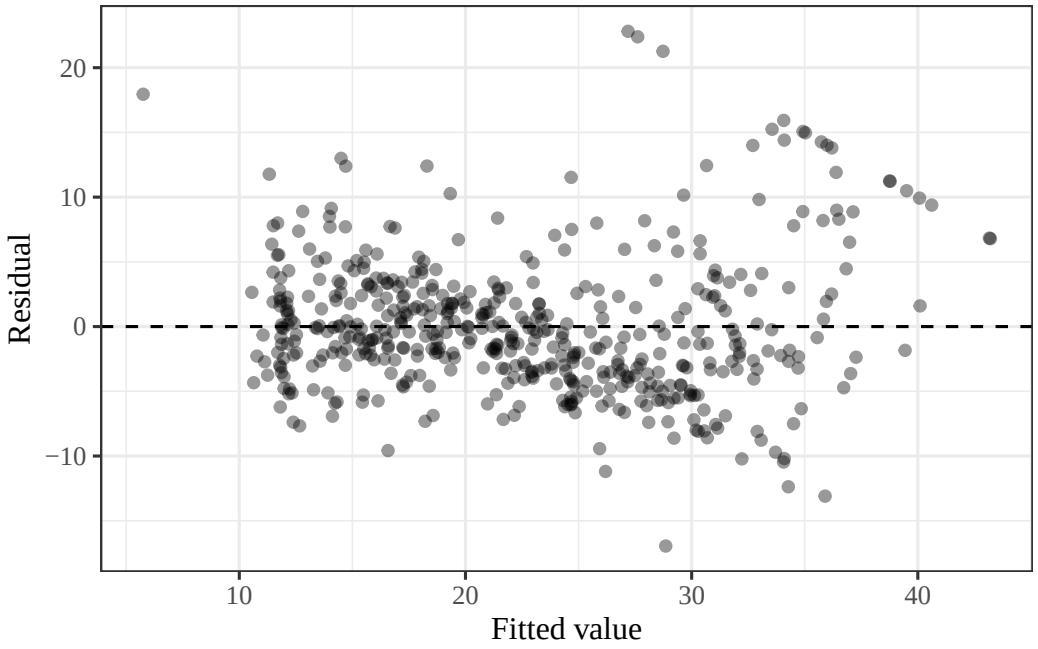
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(5, 3.5), layout="constrained")
ax.scatter(boston_quadratic.fittedvalues, boston_quadratic.resid, s=10, alpha=0.5);
ax.axhline(0, color="k", linestyle="--");
ax.set_xlabel("Fitted value");
ax.set_ylabel("Residual");
plt.show()
```



6.0.8 R (tidymodels)

```
augment(tm_boston_quadratic, new_data = boston) |>
  ggplot(aes(.pred, .resid)) +
  geom_point(alpha = 0.4) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(x = "Fitted value", y = "Residual")
```



6.0.9 R (base)

```
performance::check_model(lm_boston_quadratic, check = "linearity") |>
  plot()
```

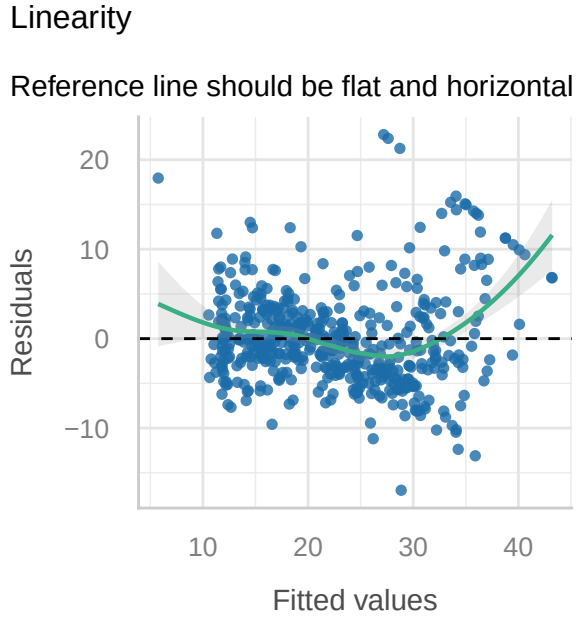


Figure 3: The gaps between the data and the curve, against the height the curve gives, for the fit of `medv` on `lstat`, lstat^2 and `age`.

Figure 3 shows the gaps scattered around zero with no clear U shape.

Exercise 6.5 (Lab: a curve in `lstat` (part 5))**.** Use the setup of Exercise 6.1.
Compare it with the residual plot of Exercise 3.1.

Hint
Compare whether the residual plot still shows the curve that the residual plot of Exercise 3.1 showed.

Solution

Solution 6.5. The U shape of the first residual plot is gone: the residuals now scatter around 0 across the fitted values, though their spread grows at the highest fitted values.

7 Qualitative predictors

Exercise 7.1 (Lab: a category with three levels)**.** Download the Carseats data from `data/carseats.csv`[a] and save it as `data/carseats.csv` next to your code. It is simulated data on sales of child car seats at 400 stores, from the book [1]. `Sales` is unit sales in thousands. `ShelveLoc` is the quality of the shelf space for the car seats at each store: `Bad`, `Medium` or `Good`.
Fit `Sales` on all the other columns, plus the interactions Income × Advertising and Price × Age.

Hint
In a formula, `Income:Advertising` adds only the product, and a column of text is treated as a category automatically.

[a] https://morrison-lab.github.io/lds/chapters/data/carseats.csv

Solution

Solution 7.1.

7.0.1 Python

```
import pandas as pd
import statsmodels.api as sm

# ISLP's load_data("Carseats") reads this same file.
Carseats = pd.read_csv("data/carseats.csv")
sales = Carseats["Sales"]

X_carseats = pd.get_dummies(Carseats.drop(columns="Sales"), drop_first=True, dtype=float)
X_carseats["Income:Advertising"] = Carseats["Income"] * Carseats["Advertising"]
X_carseats["Price:Age"] = Carseats["Price"] * Carseats["Age"]
X_carseats = sm.add_constant(X_carseats)

carseats_fit = sm.OLS(sales, X_carseats).fit()
# The table includes each coefficient's standard error
# and 95% confidence interval
print(carseats_fit.summary2().tables[1])
```

```
                      Coef.  Std.Err.   ...    [0.025    0.975]
```

```
const               6.575565  1.008747  ...  4.592239  8.558892
CompPrice           0.092937  0.004118  ...  0.084840  0.101034
Income              0.010894  0.002604  ...  0.005773  0.016015
Advertising         0.070246  0.022609  ...  0.025794  0.114699
Population          0.000159  0.000368  ... -0.000564  0.000883
Price              -0.100806  0.007440  ... -0.115434 -0.086179
Age                -0.057947  0.015951  ... -0.089307 -0.026586
Education          -0.020852  0.019613  ... -0.059414  0.017709
ShelveLoc_Good      4.848676  0.152838  ...  4.548177  5.149175
ShelveLoc_Medium    1.953262  0.125768  ...  1.705986  2.200538
Urban_Yes           0.140160  0.112402  ... -0.080837  0.361156
US_Yes             -0.157557  0.148923  ... -0.450360  0.135245
Income:Advertising  0.000751  0.000278  ...  0.000204  0.001298
Price:Age           0.000107  0.000133  ... -0.000155  0.000369

[14 rows x 6 columns]
```

7.0.2 R (tidymodels)

```r
library(tidymodels)

carseats <- read.csv("data/carseats.csv")
carseats_recipe <- recipe(Sales ~ ., data = carseats) |>
  step_dummy(all_nominal_predictors()) |>
  step_interact(~ Income:Advertising + Price:Age)

tm_carseats <- workflow() |>
  add_recipe(carseats_recipe) |>
  add_model(lm_spec) |>
  fit(carseats)
# Each coefficient with its standard error and 95% confidence interval
tidy(tm_carseats, conf.int = TRUE)
```

```
# A tibble: 14 x 7
   term                estimate std.error statistic   p.value conf.low conf.high
   <chr>                  <dbl>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>
 1 (Intercept)          6.58e+0  1.01         6.52  2.22e- 10  4.59e+0  8.56
 2 CompPrice            9.29e-2  0.00412     22.6   1.64e- 72  8.48e-2  0.101
 3 Income               1.09e-2  0.00260      4.18  3.57e-  5  5.77e-3  0.0160
 4 Advertising          7.02e-2  0.0226       3.11  2.03e-  3  2.58e-2  0.115
 5 Population           1.59e-4  0.000368     0.433 6.65e-  1 -5.64e-4  0.000883
 6 Price               -1.01e-1  0.00744    -13.5   1.74e- 34 -1.15e-1 -0.0862
 7 Age                 -5.79e-2  0.0160      -3.63  3.18e-  4 -8.93e-2 -0.0266
 8 Education           -2.09e-2  0.0196      -1.06  2.88e-  1 -5.94e-2  0.0177
 9 ShelveLoc_Good       4.85e+0  0.153       31.7   1.38e-109  4.55e+0  5.15
10 ShelveLoc_Medium     1.95e+0  0.126       15.5   1.34e- 42  1.71e+0  2.20
11 Urban_Yes            1.40e-1  0.112        1.25  2.13e-  1 -8.08e-2  0.361
12 US_Yes              -1.58e-1  0.149       -1.06  2.91e-  1 -4.50e-1  0.135
13 Income_x_Advertisi~  7.51e-4  0.000278     2.70  7.29e-  3  2.04e-4  0.00130
14 Price_x_Age          1.07e-4  0.000133     0.801 4.24e-  1 -1.55e-4  0.000369
```

7.0.3 R (base)

```r
carseats <- read.csv("data/carseats.csv", stringsAsFactors = TRUE)
lm_carseats <- lm(Sales ~ . + Income:Advertising + Price:Age, data = carseats)
# The table includes each coefficient's standard error
# and 95% confidence interval
parameters::parameters(lm_carseats) |>
  parameters::print_md(digits = 3)
```

Parameter	Coefficient	SE	95% CI	t(386)	p
(Intercept)	6.576	1.009	(4.592, 8.559)	6.519	< .001
CompPrice	0.093	0.004	(0.085, 0.101)	22.567	< .001
Income	0.011	0.003	(0.006, 0.016)	4.183	< .001
Advertising	0.070	0.023	(0.026, 0.115)	3.107	0.002
Population	1.592e-04	3.679e-04	(-5.640e-04, 8.825e-04)	0.433	0.665
Price	-0.101	0.007	(-0.115, -0.086)	-13.549	< .001
ShelveLoc (Good)	4.849	0.153	(4.548, 5.149)	31.724	< .001
ShelveLoc (Medium)	1.953	0.126	(1.706, 2.201)	15.531	< .001
Age	-0.058	0.016	(-0.089, -0.027)	-3.633	< .001
Education	-0.021	0.020	(-0.059, 0.018)	-1.063	0.288
Urban (Yes)	0.140	0.112	(-0.081, 0.361)	1.247	0.213
US (Yes)	-0.158	0.149	(-0.450, 0.135)	-1.058	0.291
Income × Advertising	7.510e-04	2.784e-04	(2.037e-04, 0.001)	2.698	0.007
Price × Age	1.068e-04	1.333e-04	(-1.554e-04, 3.689e-04)	0.801	0.424

The fit has 14 coefficients: an intercept, the 10 other columns with `ShelveLoc` expanded into two indicators, and the two products.

Exercise 7.2 (Lab: a category with three levels (part 2))**.** Use the setup of Exercise 7.1. Which indicator columns does the software build for `ShelveLoc`?

Hint

The category terms of a formula name one indicator column for each level except the baseline; read the names in the summary.

Solution

Solution 7.2.

7.0.4 Python

```python
print(X_carseats.columns.tolist())
```

```
['const', 'CompPrice', 'Income', 'Advertising', 'Population', 'Price', 'Age', 'Education', 'ShelveLoc_
```

7.0.5 R (tidymodels)

```r
carseats_recipe |> prep() |> bake(new_data = NULL) |> names()
```

```
 [1] "CompPrice"            "Income"               "Advertising"
 [4] "Population"           "Price"                "Age"
 [7] "Education"            "Sales"                "ShelveLoc_Good"
[10] "ShelveLoc_Medium"     "Urban_Yes"            "US_Yes"
[13] "Income_x_Advertising" "Price_x_Age"
```

7.0.6 R (base)

```
contrasts(carseats$ShelveLoc)
```

```
       Good Medium
Bad       0      0
Good      1      0
Medium    0      1
```

The software builds an indicator for `Good` and one for `Medium`.

Exercise 7.3 (Lab: a category with three levels (part 3))**.** Use the setup of Exercise 7.1. Which level has no column of its own?

Hint

The baseline is the level whose indicator is missing from the summary.

Solution

Solution 7.3. `Bad` has no column of its own: it is the baseline, where both indicators are 0. Each tool drops the first level in alphabetical order, and `Bad` comes first.

Exercise 7.4 (Lab: a category with three levels (part 4))**.** Use the setup of Exercise 7.1. Interpret the coefficients of the two `ShelveLoc` indicators.

Hint

Each coefficient of an indicator is a difference from the baseline, with the other features held fixed.

Solution

Solution 7.4. The `Good` coefficient is 4.85 (95% confidence interval[a] 4.55 to 5.15). A store with a good shelf location is predicted to sell about 4,850 more car seats (95% confidence interval 4,550 to 5,150 seats) than a store with a bad one, with the other features held fixed. The `Medium` coefficient is 1.95 (1.71 to 2.20), so a medium location sits between the two, about 1,950 seats above a bad one (1,710 to 2,200 seats). Each interval is a t interval, the estimate[b] plus or minus a t quantile times its standard error[c].

[a] https://morrison-lab.github.io/sds/inference.html#def-confidence-interval
[b] https://morrison-lab.github.io/sds/estimation.html#def-estimate
[c] https://morrison-lab.github.io/sds/estimation.html#def-SE

References

[1] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An introduction to statistical learning: With applications in Python*. Springer, 2023. doi: 10.1007/978-3-031-38747-0[7].

[2] D. Harrison and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air," *Journal of Environmental Economics and Management*, vol. 5, no. 1, pp. 81–102, 1978, doi: 10.1016/0095-0696(78)90006-2[8].

[7] https://doi.org/10.1007/978-3-031-38747-0
[8] https://doi.org/10.1016/0095-0696(78)90006-2