Interactions, Inference and Diagnostics

Last modified: 2026-11-19

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

Source

This material was part of the Auto lab page, which now keeps only the lab exercises. The sections follow [1, Ch. 3].

1 Fitted models on the Auto data

Every lab and example below reports 95% confidence intervals[1]. These helpers are defined once, here, so no page depends on another page's code. Each takes all its inputs as arguments.

1.0.1 Python

```
import numpy as np


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

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

```
    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)
```

1.0.2 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) {
  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.
```

```r
# 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)
}
```

The examples on this page use four least squares fits to the Auto data [2], each of a car's miles per gallon (`mpg`) on some of its features:

- the one-feature fit, on `horsepower`;
- the two-feature fit, on `horsepower` and `weight`;
- the quadratic[2] fit, on `horsepower` and its square[3];
- the origin fit, on `weight` and indicators[4] for European and Japanese cars.

The Auto lab makes the same fits step by step.

1.0.3 Python

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from scipy import stats

# ISLP's load_data("Auto") reads this same file.
Auto = pd.read_csv("data/auto.csv")
mpg = Auto["mpg"]
y = mpg.to_numpy()
horsepower = Auto["horsepower"].to_numpy(float)
weight = Auto["weight"].to_numpy(float)
n = len(y)
origin = Auto["origin"].to_numpy()
european = (origin == 2).astype(float)
japanese = (origin == 3).astype(float)
place = Auto["origin"].map({1: "American", 2: "European", 3: "Japanese"})
origin_columns = pd.get_dummies(place, drop_first=True, dtype=float)
```

[2] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-quadratic-cubic
[3] https://morrison-lab.github.io/mds/algebra-basics.html#def-square
[4] https://morrison-lab.github.io/mds/indicator-functions.html#def-indicator-function

```
def fit(columns):
    Phi = np.column_stack([np.ones(n)] + columns)
    beta_hat = np.linalg.lstsq(Phi, y, rcond=None)[0]
    return beta_hat, Phi

def mean_squared_error(beta_hat, Phi):
    return np.mean((Phi @ beta_hat - y) ** 2)

def rmse(beta_hat, Phi):
    return np.sqrt(mean_squared_error(beta_hat, Phi))

def r_squared(beta_hat, Phi):
    sst = np.sum((y - y.mean()) ** 2)
    return 1 - mean_squared_error(beta_hat, Phi) * n / sst

def ols_summary(beta_hat, Phi, y_obs=y):
    """Residual degrees of freedom, the estimated noise variance, and (Phi'Phi)^{-1}."""
    df = len(y_obs) - Phi.shape[1]
    sigma2 = np.sum((y_obs - Phi @ beta_hat) ** 2) / df
    # The pseudoinverse gives (Phi'Phi)^{-1} without forming Phi'Phi,
    # which is badly conditioned for high-degree polynomials
    pinv = np.linalg.pinv(Phi)
    return df, sigma2, pinv @ pinv.T

def coef_ci(beta_hat, Phi):
    """95% interval for each coefficient: the estimate plus or minus a t quantile times its standard er
    df, sigma2, PtP_inv = ols_summary(beta_hat, Phi)
    half = stats.t.ppf(0.975, df) * np.sqrt(sigma2 * np.diag(PtP_inv))
    return np.column_stack([beta_hat - half, beta_hat + half])

def contrast_ci(beta_hat, Phi, c):
    """95% interval for the linear combination c'beta of the coefficients."""
    df, sigma2, PtP_inv = ols_summary(beta_hat, Phi)
    half = stats.t.ppf(0.975, df) * np.sqrt(sigma2 * c @ PtP_inv @ c)
    return c @ beta_hat - half, c @ beta_hat + half

def mean_ci(beta_hat, Phi, Phi_new, prediction=False, y_obs=y):
    """95% interval at the rows of Phi_new: for the mean response, or with prediction=True for one new r
    df, sigma2, PtP_inv = ols_summary(beta_hat, Phi, y_obs)
    # The variance of the fitted value, plus the noise of one new response for a prediction interval
    variance = sigma2 * (np.sum((Phi_new @ PtP_inv) * Phi_new, axis=1) + prediction)
    half = stats.t.ppf(0.975, df) * np.sqrt(variance)
    fitted_new = Phi_new @ beta_hat
    return fitted_new - half, fitted_new + half

def bootstrap_r2_ci(Phi, B=2000, seed=571):
    """95% percentile interval for R^2 from B resamples of the cars, refitted each time."""
    rng = np.random.default_rng(seed)
    r2 = np.empty(B)
    for b in range(B):
        rows = rng.integers(0, n, n)
        beta_b = np.linalg.lstsq(Phi[rows], y[rows], rcond=None)[0]
        sse = np.sum((y[rows] - Phi[rows] @ beta_b) ** 2)
        r2[b] = 1 - sse / np.sum((y[rows] - y[rows].mean()) ** 2)
    return np.quantile(r2, [0.025, 0.975])
```

```python
beta_simple, Phi_simple = fit([horsepower])
beta_multiple, Phi_multiple = fit([horsepower, weight])
beta_quadratic, Phi_quadratic = fit([horsepower, horsepower ** 2])
beta_origin_weight, Phi_origin_weight = fit([weight, european, japanese])
fits = {
    "one-feature": beta_simple,
    "two-feature": beta_multiple,
    "quadratic": beta_quadratic,
    "origin": beta_origin_weight,
}
for name, beta in fits.items():
    print(f"{name}: {np.round(beta, 4)}")
```

```
one-feature: [39.9359 -0.1578]
two-feature: [ 4.56402e+01 -4.73000e-02 -5.80000e-03]
quadratic: [ 5.69001e+01 -4.66200e-01  1.20000e-03]
origin: [ 4.37322e+01 -7.00000e-03  9.70900e-01  2.32710e+00]
```

1.0.4 R

```r
library(tidymodels) # nolint: undesirable_function_linter

auto <- readr::read_csv("data/auto.csv", show_col_types = FALSE)
lm_spec <- linear_reg() |> set_engine("lm")
fit_simple <- lm(mpg ~ horsepower, data = auto)
fit_multiple <- lm(mpg ~ horsepower + weight, data = auto)
fit_quadratic <- lm(mpg ~ horsepower + I(horsepower^2), data = auto)
fit_origin_weight <- lm(mpg ~ weight + factor(origin), data = auto)
fits <- list(
  one_feature = fit_simple,
  two_feature = fit_multiple,
  quadratic = fit_quadratic,
  origin = fit_origin_weight
)
fits |> map(coef)
```

```
$one_feature
(Intercept)  horsepower
 39.9358610  -0.1578447

$two_feature
 (Intercept)   horsepower       weight
45.640210840 -0.047302863 -0.005794157

$quadratic
    (Intercept)      horsepower I(horsepower^2)
   56.900099702    -0.466189630     0.001230536

$origin
    (Intercept)          weight factor(origin)2 factor(origin)3
   43.732236152    -0.007027083     0.970905585     2.327149903
```

2 Interaction terms

Exercise 2.1 (Does the horsepower slope depend on weight?)**.** In the two-feature fit, how much does the predicted `mpg` change per extra horsepower for a light car (2000 pounds), and for a heavy car (4000 pounds)?

Solution

Solution 2.1. The fitted model is $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 \,\mathrm{hp} + \hat{\beta}_2 \, w$, where hp is horsepower and w is weight. Differentiating in hp gives $\hat{\beta}_1$, which does not involve w. So the change is -0.047 miles per gallon per horsepower for the light car and for the heavy car alike.
The same holds for every pair of features in a model that is a sum of one-feature terms: the effect of one feature on the prediction is the same at every value of the others. James et al. [1, Sec. 3.3.2], p. 94 calls this the *additive* assumption. To let the horsepower slope change with weight, the design matrix needs a column that combines the two features.

Unit sales in 200 markets, with the advertising budget in each market, make up the `Advertising` data [1, Sec. 2.1]. The data are in `data/advertising.csv`[5], one row per market. The file is a copy of the `Advertising` data from the book's website, statlearning.com[6]. The columns are:

- `sales`: the units sold, in thousands.
- `TV`: the television advertising budget, in thousands of dollars.
- `radio`: the radio advertising budget, in thousands of dollars.
- `newspaper`: the newspaper advertising budget, in thousands of dollars.

Dataset 1: The advertising and sales (`Advertising`) data

Definition 2.1 (Interaction term (product term, cross term))**.** An **interaction term** (also called a **product term** or a **cross term**) between two features x_1 and x_2 is their product $x_1 x_2$, used as an extra column of the design matrix [3]. It is a basis function of two inputs[a]. With an interaction term, the feature map is $\phi(\underset{\sim}{x}) = [1, x_1, x_2, x_1 x_2]^\top$, and the model is

$$h(\underset{\sim}{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2.$$

The slope of h in x_1, with x_2 held fixed, is

$$\frac{\partial h}{\partial x_1} = \beta_1 + \beta_3 x_2,$$

so it changes with x_2. In the same way, the slope in x_2 is $\beta_2 + \beta_3 x_1$. If $\beta_3 = 0$, the model is additive, as in Solution 2.1.

[a] https://morrison-lab.github.io/sds/model-validation.html#def-data-set

Note 1: Other names for interactions

Marketing calls an interaction a *synergy* effect [1, Sec. 3.3.2], p. 95, and epidemiology calls it *effect modification*. These notes say *interaction*.
In R's model formulas, `x1:x2` is the product column alone, and `x1 * x2` is short for `x1 + x2 + x1:x2`.

Example 2.1 (Television and radio advertising)**.** In the Advertising data (Dataset 1), television advertising may sell more when radio advertising is also high. Fit `sales` on `TV`, `radio` and their interaction:

2.0.1 Python

```
advertising = pd.read_csv("data/advertising.csv")
tv = advertising["TV"].to_numpy(float)
radio = advertising["radio"].to_numpy(float)
sales = advertising["sales"].to_numpy(float)
ones = np.ones(len(sales))

Phi_tv_radio = np.column_stack([ones, tv, radio, tv * radio])
beta_tv_radio = np.linalg.lstsq(Phi_tv_radio, sales, rcond=None)[0]
coef_names = ["intercept", "TV", "radio", "TV:radio"]
pd.Series(beta_tv_radio, index=coef_names).round(4)
```

```
intercept    6.7502
TV           0.0191
radio        0.0289
TV:radio     0.0011
dtype: float64
```

2.0.2 R

```
advertising <- readr::read_csv("data/advertising.csv", show_col_types = FALSE)
fit_tv_radio <- lm(sales ~ TV * radio, data = advertising)
fit_additive <- lm(sales ~ TV + radio, data = advertising)
parameters::parameters(fit_tv_radio) |>
  parameters::print_md(digits = 4)
```

Parameter	Coefficient	SE	95% CI	t(196)	p
(Intercept)	6.7502	0.2479	(6.2614, 7.2391)	27.2328	$< .001$
TV	0.0191	0.0015	(0.0161, 0.0221)	12.6990	$< .001$
radio	0.0289	0.0089	(0.0113, 0.0464)	3.2408	0.001
TV × radio	0.0011	5.2420e-05	(0.0010, 0.0012)	20.7266	$< .001$

The slope[a] of `sales` in `TV`, with `radio` held fixed, is 19.1 plus 1.1 times the radio budget, in units per thousand dollars of television advertising. So television advertising sells more where the radio budget is higher. The model with the interaction has $R^2 = 96.8\%$, against 89.7% for the additive model [1, Sec. 3.3.2].

```
library(ggplot2) # nolint: undesirable_function_linter
quartile_probs <- c(0.25, 0.75)
radio_quartiles <- advertising$radio |> quantile(quartile_probs)
tv_range <- advertising$TV |> range()
lines_tv_radio <- expand.grid(TV = tv_range, radio = radio_quartiles)
lines_tv_radio$sales <- predict(fit_tv_radio, lines_tv_radio)
lines_tv_radio$radio_budget <- factor(lines_tv_radio$radio)
advertising |>
  ggplot() +
  aes(x = TV, y = sales) +
  geom_point(colour = "grey50") +
  geom_line(
    data = lines_tv_radio,
    aes(colour = radio_budget),
    linewidth = 1
  ) +
  labs(colour = "Radio budget")
```

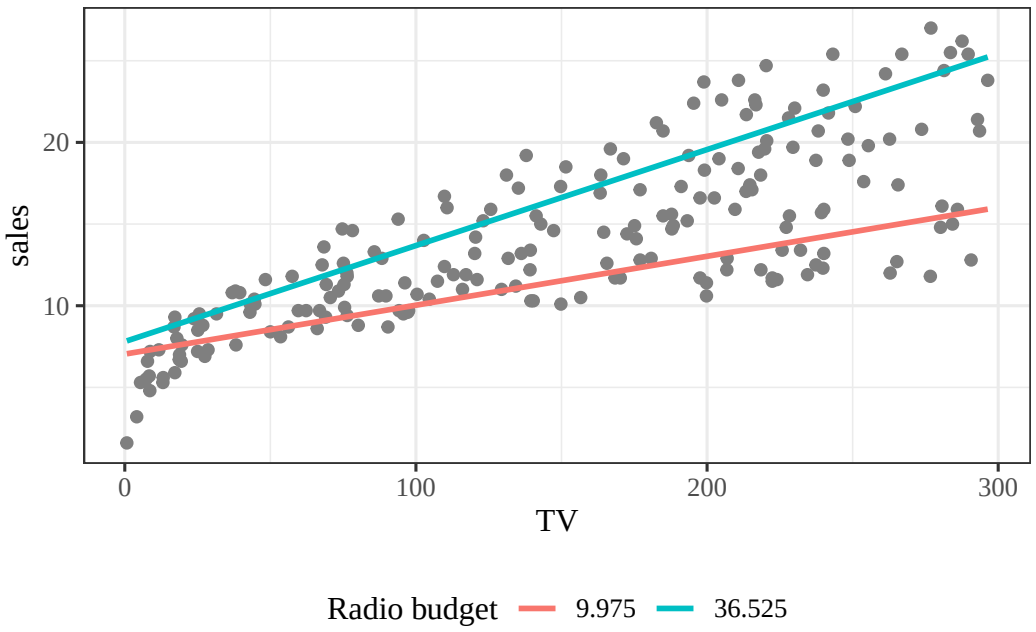


Figure 1: Sales against television advertising in the Advertising data, with the fitted lines of the interaction model at the lower and upper quartiles of the radio budget.

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

Example 2.2 (Weight matters more in newer cars)**.** The Auto data record each car's model `year`, from 1970 to 1982, as 70 to 82. Fit `mpg` on `weight`, `year` and their interaction:

2.0.3 Python

```
year = Auto["year"].to_numpy(float)
beta_weight_year, Phi_weight_year = fit([weight, year, weight * year])
print(beta_weight_year)
```

```
[-1.10451919e+02  2.75465227e-02  2.04047640e+00 -4.57932342e-04]
```

```
def weight_slope(model_year):
    """Change in predicted mpg per 1000 pounds, in a given model year."""
    return 1000 * (beta_weight_year[1] + beta_weight_year[3] * model_year)

print(weight_slope(70), weight_slope(82))
```

```
-4.508741178179798 -10.003929277689418
```

```
from scipy import stats

# The covariance of the coefficients is sigma^2 (Phi'Phi)^-1.
# Phi'Phi = R'R for the QR factor R of Phi, so (Phi'Phi)^-1 = R^-1 R^-T,
# which is more accurate than inverting Phi'Phi, since the weight-by-year column is large.
df_wy = n - Phi_weight_year.shape[1]
rse_wy = np.sqrt(np.sum((y - Phi_weight_year @ beta_weight_year) ** 2) / df_wy)
R_inv = np.linalg.inv(np.linalg.qr(Phi_weight_year, mode="r"))
cov_wy = rse_wy ** 2 * R_inv @ R_inv.T
t_wy = stats.t.ppf(0.975, df=df_wy)

def combination_ci(a):
    """95% confidence interval for a'beta: the estimate plus or minus t standard errors."""
    half = t_wy * np.sqrt(a @ cov_wy @ a)
    return np.array([a @ beta_weight_year - half, a @ beta_weight_year + half])

ci_interaction = combination_ci(np.array([0, 0, 0, 1.0]))
ci_slope_70 = combination_ci(1000 * np.array([0, 1, 0, 70.0]))
ci_slope_82 = combination_ci(1000 * np.array([0, 1, 0, 82.0]))
print(ci_interaction, ci_slope_70, ci_slope_82)
```

```
[-0.00057407 -0.00034179] [-5.1754346  -3.84204775] [-10.94509211  -9.06276645]
```

```
# The ratio of the two slopes is nonlinear in the coefficients, so bootstrap it:
# resample cars with replacement, refit, and recompute the ratio each time.
rng_wy = np.random.default_rng(571)
ratio_wy = np.empty(2000)
for b in range(2000):
    rows = rng_wy.integers(0, n, n)
    beta_b = np.linalg.lstsq(Phi_weight_year[rows], y[rows], rcond=None)[0]
    ratio_wy[b] = (beta_b[1] + 82 * beta_b[3]) / (beta_b[1] + 70 * beta_b[3])
ratio_ci_wy = np.quantile(ratio_wy, [0.025, 0.975])
print(ratio_ci_wy)
```

```
[1.81334968 2.77069719]
```

```python
# A function, so the plotting calls' return values are not printed.
def plot_weight_year():
    grid = np.linspace(weight.min(), weight.max(), 100)
    fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
    ax.scatter(weight, y, s=12, alpha=0.4, color="grey");
    for model_year, colour in [(70, "C0"), (82, "C3")]:
        # The design matrix of the fitted line for this model year.
        Phi_grid = np.column_stack([np.ones_like(grid), grid,
                                    np.full_like(grid, model_year), grid * model_year])
        line = Phi_grid @ beta_weight_year
        # Half-width of the 95% confidence band: t * sqrt(phi' cov phi) at each weight.
        half = t_wy * np.sqrt(np.sum((Phi_grid @ cov_wy) * Phi_grid, axis=1))
        ax.fill_between(grid, line - half, line + half, color=colour, alpha=0.3);
        ax.plot(grid, line, color=colour, label=f"19{model_year}");
    ax.set_xlabel("Weight (pounds)");
    ax.set_ylabel("Miles per gallon");
    fig.legend(title="Fitted line", loc="outside lower center", ncol=2);
    plt.show()

plot_weight_year()
```

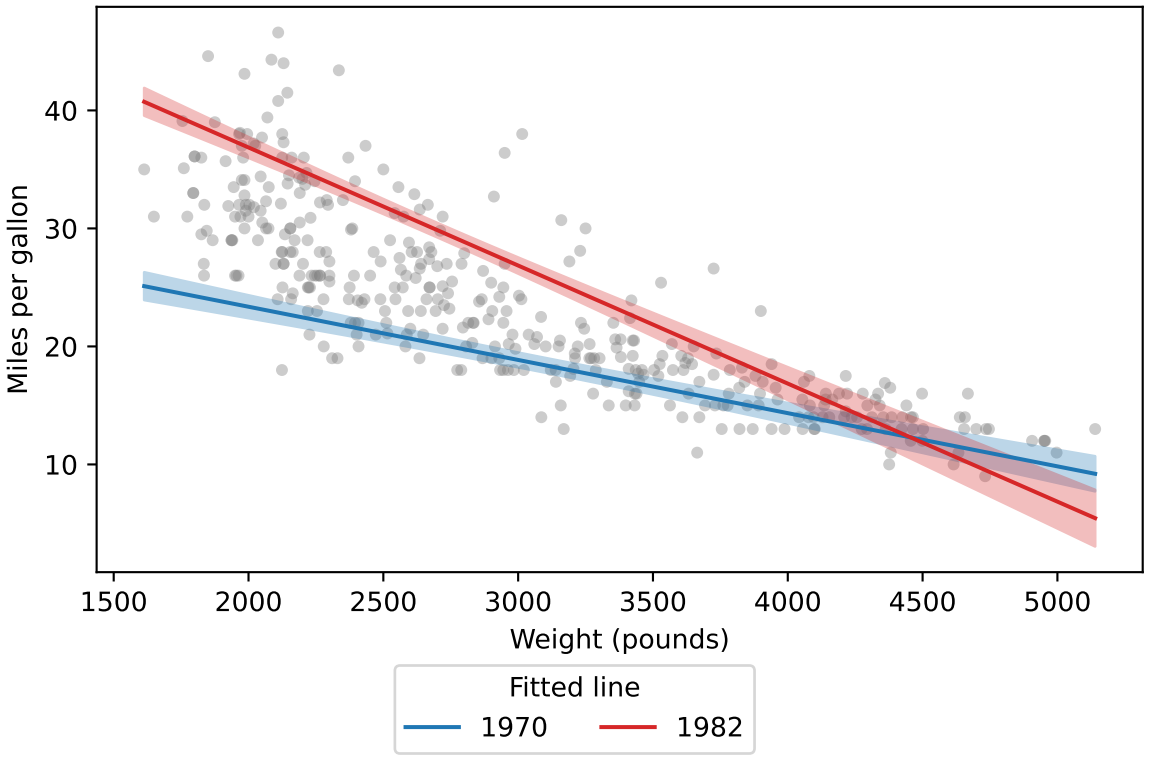


Figure 2: Fuel economy against weight for the Auto data, with the fitted lines of the interaction model for model years 1970 and 1982. Each shaded ribbon is the 95% confidence band for the mean fuel economy of cars of that model year.

2.0.4 R

```r
fit_weight_year <- lm(mpg ~ weight * year, data = auto)
parameters::parameters(fit_weight_year) |>
  parameters::print_md(digits = 4)
```

Parameter	Coefficient	SE	95% CI	t(388)	p
(Intercept)	-110.4519	12.9470	(-135.9069, -84.9969)	-8.5311	< .001
weight	0.0275	0.0044	(0.0189, 0.0362)	6.2415	< .001
year	2.0405	0.1718	(1.7027, 2.3783)	11.8756	< .001
weight × year	-0.0005	5.9071e-05	(-0.0006, -0.0003)	-7.7522	< .001

```r
weight_slope <- function(model_year) {
  b <- coef(fit_weight_year)
  1000 * (b[["weight"]] + b[["weight:year"]] * model_year)
}
c(weight_slope(70), weight_slope(82))
```

```
[1]  -4.508741 -10.003929
```

```r
# 95% confidence interval for the weight slope in a given model year:
# the slope is the linear combination a'b, with variance a' vcov a.
slope_ci <- function(model_year) {
  a <- 1000 * c(0, 1, 0, model_year)
  half <- qt(0.975, df.residual(fit_weight_year)) *
    sqrt(drop(t(a) %*% vcov(fit_weight_year) %*% a))
  sum(a * coef(fit_weight_year)) + c(-1, 1) * half
}
ci_interaction <- confint(fit_weight_year)["weight:year", ]
rbind(interaction = ci_interaction,
      slope_1970 = slope_ci(70), slope_1982 = slope_ci(82))
```

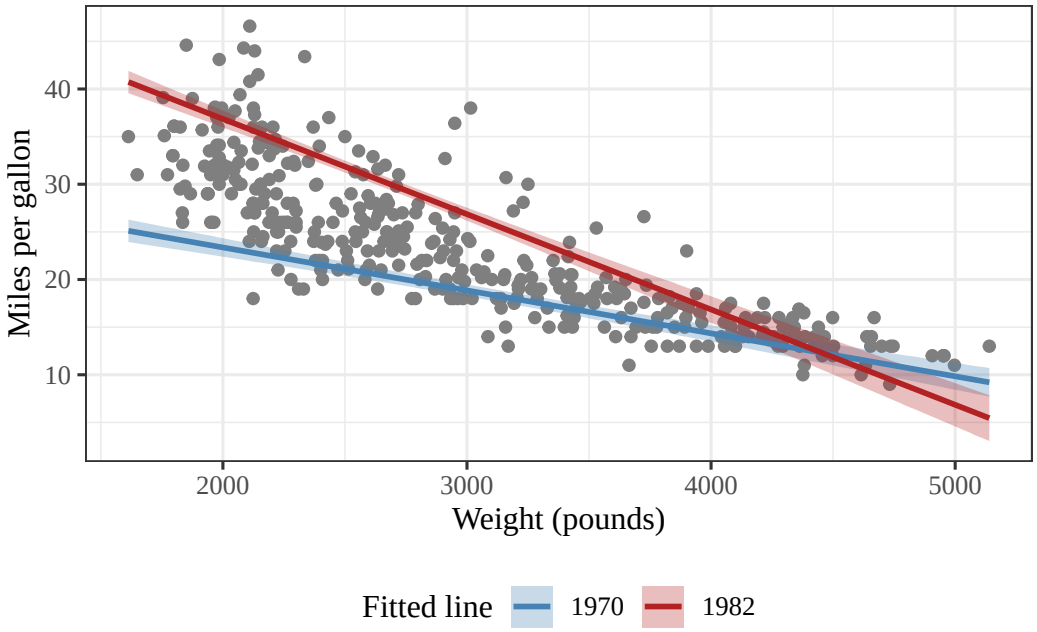
```
                    2.5 %        97.5 %
interaction -5.740726e-04 -0.0003417921
slope_1970  -5.175435e+00 -3.8420477515
slope_1982  -1.094509e+01 -9.0627664467
```

```r
# The ratio of the two slopes is nonlinear in the coefficients, so bootstrap it:
# resample cars with replacement, refit, and recompute the ratio each time.
set.seed(571)
ratio_wy <- replicate(2000, {
  resampled <- auto[sample(nrow(auto), replace = TRUE), ]
  b <- coef(lm(mpg ~ weight * year, data = resampled))
  slope_82 <- b[["weight"]] + 82 * b[["weight:year"]]
  slope_70 <- b[["weight"]] + 70 * b[["weight:year"]]
  slope_82 / slope_70
})
ratio_ci_wy <- quantile(ratio_wy, c(0.025, 0.975))
ratio_ci_wy
```

```
    2.5%    97.5%
1.814234 2.777316
```

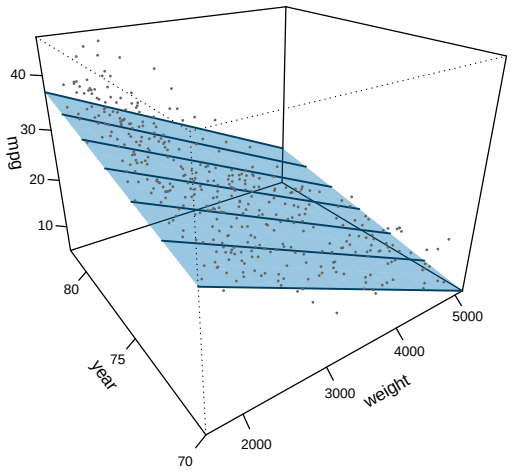
```r
library(ggplot2)

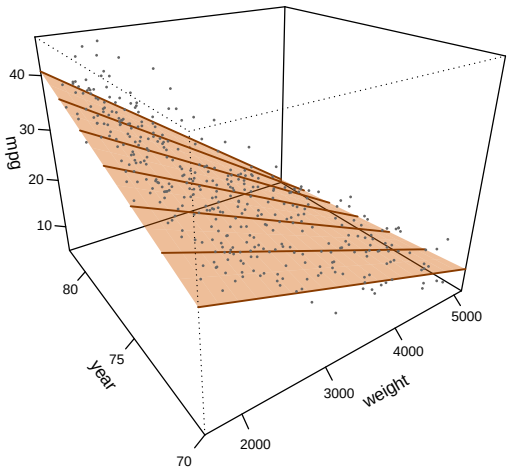
weight_year_grid <- expand.grid(
  weight = seq(min(auto$weight), max(auto$weight), length.out = 100),
  year = c(70, 82)
)
# predict() returns the fit and the ends of its 95% confidence band.
band <- predict(fit_weight_year, weight_year_grid, interval = "confidence")
weight_year_grid <- cbind(weight_year_grid, band)
ggplot(auto, aes(weight, mpg)) +
  geom_point(colour = "grey50") +
  geom_ribbon(data = weight_year_grid,
              aes(y = NULL, ymin = lwr, ymax = upr, fill = factor(year + 1900)),
              alpha = 0.3) +
  geom_line(data = weight_year_grid,
            aes(y = fit, colour = factor(year + 1900)), linewidth = 1) +
  scale_colour_manual(values = c("steelblue", "firebrick")) +
  scale_fill_manual(values = c("steelblue", "firebrick")) +
  labs(x = "Weight (pounds)", y = "Miles per gallon",
       colour = "Fitted line", fill = "Fitted line")
```

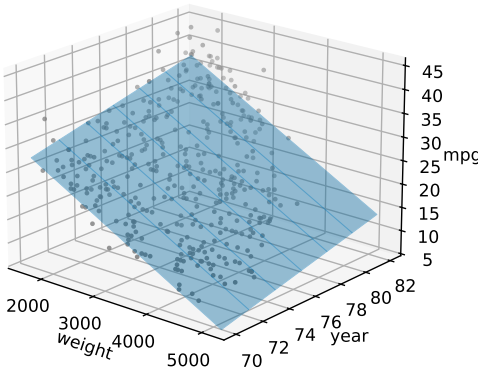


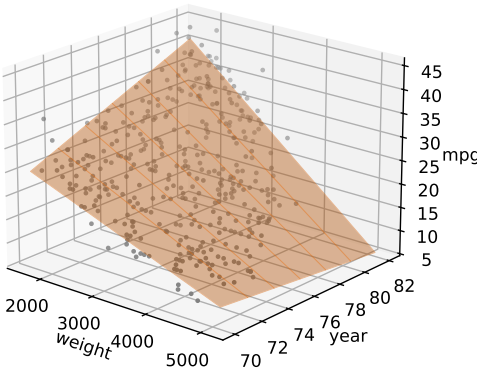
The interaction coefficient is $\hat{\beta}_3 = -4.6 \times 10^{-4}$, with a 95% confidence interval from -5.7×10^{-4} to -3.4×10^{-4}. By Definition 2.1, the slope in weight is $\hat{\beta}_1 + \hat{\beta}_3 \cdot \text{year}$. Per 1000 pounds, it is -4.5 miles per gallon for a 1970 car (interval -5.2 to -3.8) and -10.0 for a 1982 car (interval -10.9 to -9.1). The intervals for the coefficient and the slopes are model-based t intervals. In the fitted model, the estimated slope in weight for 1982 cars is about 2.2 times the estimated slope for 1970 cars (both slopes are negative), with a 95% bootstrap percentile interval for the ratio from 1.8 to 2.8. All of these intervals describe which cars were drawn, and assume the model's form is right.

Example 2.3 (The interaction term twists the fitted plane)**.** Fit `mpg` on `weight` and `year` twice: once without the interaction term, as an additive model, and once with it, as in Example 2.2.
Figure 3 draws both fitted surfaces over the cars.









(a) Without the interaction term: the surface is a plane, and its slices at every second value of `year` are parallel.

(b) With the interaction term: the surface twists, and its slices at every second value of `year` fan out.

Figure 3: The fitted surfaces of two models for `mpg` over `weight` and `year`, with the cars as gray points.

In the additive model (Figure 3a), the slope in `weight` is -6.6 miles per gallon per 1000 pounds in every model year. So the slices of the surface at fixed values of `year` are parallel lines, and the surface is a plane. With the interaction term (Figure 3b), the slope in `weight` is $\hat{\beta}_1 + \hat{\beta}_3 \cdot \text{year}$, which goes from -4.5 per 1000 pounds in 1970 to -10.0 in 1982. The slices at different years have different slopes, so the surface twists and is no longer a plane.

Definition 2.2 (Hierarchy principle (hierarchical principle))**.** The **hierarchy principle** (also called the **hierarchical principle**) says that a model with an interaction $x_1 x_2$ should also include the main

effects x_1 and x_2 [1, Sec. 3.3.2].

Example 2.4 (Weight and year in the Auto data)**.** The model of `mpg` with the interaction `weight` × `year` and the main effects `weight` and `year` follows the principle. The model of `mpg` with `weight` × `year` and `weight` but without `year` does not. It forces the effect of model year on `mpg` to be zero for a car that weighs nothing, which no car does.

Proposition 2.1 (A model with its main effects is unchanged by shifting a feature)**.** *Call a function*[a] *a* model function *if it has the form* $b_0 + b_1x_1 + b_2x_2 + b_3x_1x_2$*, for some coefficients* b_0, b_1, b_2, b_3*. Shifting* x_1 *by a constant*[b] c*, to* $x_1' = x_1 + c$*, turns every model function into a model function of* x_1' *and* x_2*.*

[a] https://morrison-lab.github.io/mcs/functions.html#def-function
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function

Proof

Proof. For a shift of x_1, substitute $x_1 = x_1' - c$.

$$b_0 + b_1(x_1' - c) + b_2x_2 + b_3(x_1' - c)x_2 = (b_0 - cb_1) + b_1x_1' + (b_2 - cb_3)x_2 + b_3x_1'x_2,$$

which has the form of a model function[a] in x_1' and x_2. The same substitution, with the roles of x_1 and x_2 exchanged, shows the same for a shift of x_2. □

[a] https://morrison-lab.github.io/mcs/functions.html#def-function

Example 2.5 (Centering weight in the Auto data)**.** Take the model of `mpg` with `weight` × `year` and both main effects. Subtract a constant[a] c from `weight`, as when centering `weight` at a typical value. By Proposition 2.1, the model is still a model function[b] of the shifted `weight` and `year`. The fitted model does not change, because adding c back to `weight` undoes the shift, so the model function[c]s before and after the shift are the same. Only the coefficients change, and the coefficient of `year` becomes the effect of model year on `mpg` for a car with `weight` $= c$.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[b] https://morrison-lab.github.io/mcs/functions.html#def-function
[c] https://morrison-lab.github.io/mcs/functions.html#def-function

Proposition 2.2 (Shifting a feature can create a dropped main effect)**.** *Let a function*[a] *without the main effect of* x_1 *have the form* $b_0 + b_2x_2 + b_3x_1x_2$*, with* $b_3 \neq 0$*. Shifting* x_2 *by a constant*[b] $c \neq 0$*, to* $x_2' = x_2 + c$*, gives a function with a nonzero main effect of* x_1*.*

[a] https://morrison-lab.github.io/mcs/functions.html#def-function
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function

Proof

Proof. Substitute $x_2 = x_2' - c$.

$$b_0 + b_2(x_2' - c) + b_3x_1(x_2' - c) = (b_0 - cb_2) + b_2x_2' + b_3x_1x_2' - cb_3x_1.$$

Here $-cb_3x_1$ is a main effect of x_1, with the coefficient $-cb_3$, which is nonzero because $b_3 \neq 0$ and $c \neq 0$. So the shifted function[a] is not of the form without the main effect of x_1. □

[a] https://morrison-lab.github.io/mcs/functions.html#def-function

Example 2.6 (Shifting weight without the main effect of year)**.** Take the model of `mpg` with `weight` and `weight` × `year`, but without `year`. Add a constant[a] $c \neq 0$ to `weight`. By Proposition 2.2, the shifted model has a main effect of `year`, with the coefficient $-c$ times that of `weight` × `year`. So the model that followed the hierarchy principle before the shift no longer does.

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

3 Inference for the coefficients

This section follows [1, Sec. 3.1.2], and takes the matrix form of the results from the rme notes on linear models[7].

3.1 How precise is a coefficient?

Exercise 3.1 (How much would the slope change with new data?)**.** The one-feature fit has a slope[a] of -0.158 miles per gallon per horsepower to 392 cars. Suppose we measured 392 different cars of the same kinds and fitted the same model. Would we get the same slope?

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

Solution

Solution 3.1. No. Each response is $y_i = \phi(\underset{\sim}{x}_i) \cdot \vec{\beta} + \varepsilon_i$, and the noise terms ε_i differ from one sample to the next. The estimate $\hat{\vec{\beta}} = (\mathbf{D}^\top \mathbf{D})^{-1} \mathbf{D}^\top \underset{\sim}{y}$ is a function of $\underset{\sim}{y}$, so it is a random variable too.

Exercise 3.2 (How much would the slope change with new data? (part 2))**.** Use the setup of Exercise 3.1.
What would describe how much the slope[a] changes from one sample of cars to another?

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

Solution

Solution 3.2. The standard deviation[a] of $\hat{\beta}_1$ over repeated samples[b] describes how much the slope[c] changes. A small standard deviation shows that one sample pins the slope down well.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-sd
[b] https://morrison-lab.github.io/sds/exploratory-descriptive.html#def-sample
[c] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope

3.2 The covariance of the least squares estimate

Exercise 3.3 (The mean and covariance of the least squares estimate)**.** Suppose $y = \mathbf{D}\vec{\beta} + \vec{\varepsilon}$, where $\mathbf{D}$ is a fixed $n \times p^*$ design matrix with full column rank, $\mathbb{E}[\vec{\varepsilon}] = \mathbf{0}_{n \times 1}$ and $\text{Cov}(\vec{\varepsilon}) = \tilde{\sigma}^2 \mathbf{I}_n$. Find the mean $\mathbb{E}\left[\hat{\vec{\beta}}\right]$ and the covariance matrix[a] $\text{Cov}\left(\hat{\vec{\beta}}\right)$ of the least squares estimate $\hat{\vec{\beta}} = (\mathbf{D}^\top \mathbf{D})^{-1} \mathbf{D}^\top \underset{\sim}{y}$.

Hint
First write $\hat{\vec{\beta}} - \vec{\beta}$ as a matrix times $\vec{\varepsilon}$.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x

[7] https://morrison-lab.github.io/rme/chapters/Linear-models-overview.html#inference-for-individual-predictor-coefficients

Solution

Solution 3.3. Let $\mathbf{A} = (\mathbf{D}^\top \mathbf{D})^{-1}\mathbf{D}^\top$, a fixed $p^* \times n$ matrix. Substitute the model for $\underset{\sim}{y}$:

$$\begin{aligned}
\hat{\vec{\beta}} &= \mathbf{A}(\mathbf{D}\vec{\beta} + \vec{\varepsilon}) && \text{(substitute } \underset{\sim}{y} = \mathbf{D}\vec{\beta} + \vec{\varepsilon}\text{)} \\
&= (\mathbf{D}^\top \mathbf{D})^{-1}\mathbf{D}^\top \mathbf{D}\vec{\beta} + \mathbf{A}\vec{\varepsilon} && \text{(distribute)} \\
&= \vec{\beta} + \mathbf{A}\vec{\varepsilon} && ((\mathbf{D}^\top \mathbf{D})^{-1}\mathbf{D}^\top \mathbf{D} = \mathbf{I}_{p^*})
\end{aligned}$$

The mean follows from linearity of expectation:

$$\begin{aligned}
\mathbb{E}\left[\hat{\vec{\beta}}\right] &= \vec{\beta} + \mathbf{A}\,\mathbb{E}[\vec{\varepsilon}] && \text{(linearity; } \mathbf{A} \text{ is fixed)} \\
&= \vec{\beta} && (\mathbb{E}[\vec{\varepsilon}] = \mathbf{0})
\end{aligned}$$

So $\hat{\vec{\beta}}$ is unbiased. For the covariance, $\hat{\vec{\beta}} - \mathbb{E}\left[\hat{\vec{\beta}}\right] = \mathbf{A}\vec{\varepsilon}$:

$$\begin{aligned}
\mathrm{Cov}\left(\hat{\vec{\beta}}\right) &= \mathbb{E}\left[(\mathbf{A}\vec{\varepsilon})(\mathbf{A}\vec{\varepsilon})^\top\right] && \text{(definition of covariance)} \\
&= \mathbb{E}\left[\mathbf{A}\vec{\varepsilon}\vec{\varepsilon}^\top \mathbf{A}^\top\right] && ((\mathbf{A}\vec{\varepsilon})^\top = \vec{\varepsilon}^\top \mathbf{A}^\top) \\
&= \mathbf{A}\,\mathbb{E}[\vec{\varepsilon}\vec{\varepsilon}^\top]\mathbf{A}^\top && \text{(linearity; } \mathbf{A} \text{ is fixed)} \\
&= \mathbf{A}\,(\sigma^2 \mathbf{I}_n)\,\mathbf{A}^\top && (\mathbb{E}[\vec{\varepsilon}\vec{\varepsilon}^\top] = \mathrm{Cov}(\vec{\varepsilon}), \text{ since } \mathbb{E}[\vec{\varepsilon}] = \mathbf{0}) \\
&= \sigma^2 (\mathbf{D}^\top \mathbf{D})^{-1}\mathbf{D}^\top \mathbf{D}(\mathbf{D}^\top \mathbf{D})^{-1} && (\mathbf{A}^\top = \mathbf{D}(\mathbf{D}^\top \mathbf{D})^{-1}, \text{ as } \mathbf{D}^\top \mathbf{D} \text{ is symmetric}) \\
&= \sigma^2 (\mathbf{D}^\top \mathbf{D})^{-1} && (\mathbf{D}^\top \mathbf{D}(\mathbf{D}^\top \mathbf{D})^{-1} = \mathbf{I}_{p^*})
\end{aligned}$$

Theorem 3.1 (Mean and covariance of the least squares estimate)**.** *If* $\underset{\sim}{y} = \mathbf{D}\vec{\beta} + \vec{\varepsilon}$ *with* $\mathbf{D}$ *fixed and of full column rank,* $\mathbb{E}[\vec{\varepsilon}] = \mathbf{0}_{n\times 1}$ *and* $\mathrm{Cov}(\vec{\varepsilon}) = \sigma^2 \mathbf{I}_n$*, then*

$$\mathbb{E}\left[\hat{\vec{\beta}}\right] = \vec{\beta}, \qquad \mathrm{Cov}\left(\hat{\vec{\beta}}\right) = \sigma^2 \underbrace{(\mathbf{D}^\top \mathbf{D})^{-1}}_{p^* \times p^*}.$$

In particular, $\mathrm{Var}\left(\hat{\beta}_j\right) = \sigma^2 [(\mathbf{D}^\top \mathbf{D})^{-1}]_{jj}$.

Proof

Proof. By Solution 3.3. □

Example 3.1 (Checking the slope's variance by simulation)**.**
Take the design matrix of the one-feature fit (an intercept and `horsepower` for the 392 Auto cars), treat its fitted coefficients as the true $\vec{\beta}$, and take σ to be its residual standard error. Simulate 2000 new response vectors $\underset{\sim}{y} = \mathbf{D}\vec{\beta} + \vec{\varepsilon}$ with $\vec{\varepsilon} \sim \mathrm{N}(\mathbf{0}, \sigma^2 \mathbf{I}_n)$, refit each one, and compare the standard deviation of the 2000 slopes with $\sqrt{\sigma^2 [(\mathbf{D}^\top \mathbf{D})^{-1}]_{11}}$ from Theorem 3.1:

3.2.1 Python

```python
residuals_simple = y - Phi_simple @ beta_simple
sigma_simple = np.sqrt(np.sum(residuals_simple ** 2) / (n - 2))
PtP_inv = np.linalg.inv(Phi_simple.T @ Phi_simple)
sd_theory = sigma_simple * np.sqrt(PtP_inv[1, 1])

rng = np.random.default_rng(571)
n_sims = 2000
sim_slopes = np.empty(n_sims)
sim_ses = np.empty(n_sims)
for sim in range(n_sims):
    y_sim = Phi_simple @ beta_simple + rng.normal(0, sigma_simple, n)
    b_sim = PtP_inv @ Phi_simple.T @ y_sim
    rse_sim = np.sqrt(np.sum((y_sim - Phi_simple @ b_sim) ** 2) / (n - 2))
    sim_slopes[sim] = b_sim[1]
    sim_ses[sim] = rse_sim * np.sqrt(PtP_inv[1, 1])

from scipy import stats

# The simulated slopes are exactly Gaussian, so (N - 1) s^2 / sd^2 follows
# a chi-squared distribution with N - 1 degrees of freedom;
# inverting that statement gives a 95% interval for the true standard deviation.
sd_sim = np.std(sim_slopes, ddof=1)
sd_sim_ci = sd_sim * np.sqrt((n_sims - 1) / stats.chi2.ppf([0.975, 0.025], df=n_sims - 1))
print(sd_theory, sd_sim, sd_sim_ci)
```

```
0.006445500517685036 0.006482351093736395 [0.00628751 0.00668974]
```

3.2.2 R

```r
design <- model.matrix(fit_simple)
sigma_simple <- summary(fit_simple)$sigma
xtx_inv <- solve(crossprod(design))
sd_theory <- sigma_simple * sqrt(xtx_inv[2, 2])

set.seed(571)
sim_fits <- replicate(2000, {
  y_sim <- fitted(fit_simple) + rnorm(nrow(design), 0, sigma_simple)
  coef(summary(lm(y_sim ~ design - 1)))[2, c("Estimate", "Std. Error")]
})
sd_sim <- sd(sim_fits["Estimate", ])
n_sims <- ncol(sim_fits)
# The simulated slopes are exactly Gaussian, so (N - 1) s^2 / sd^2 follows
# a chi-squared distribution with N - 1 degrees of freedom;
# inverting that statement gives a 95% interval for the true standard deviation.
chi_sq <- qchisq(c(0.975, 0.025), df = n_sims - 1)
sd_sim_ci <- sd_sim * sqrt((n_sims - 1) / chi_sq)
c(theory = sd_theory, simulated = sd_sim,
  lower = sd_sim_ci[1], upper = sd_sim_ci[2])
```

```
     theory   simulated       lower       upper
0.006445501 0.006367362 0.006175978 0.006571075
```

The formula gives 0.00645, and the 2000 simulated slopes have standard deviation 0.00648, with a 95% confidence interval from 0.00629 to 0.00669. The interval measures only the simulation error of using 2000 draws, and it uses the exact chi-squared distribution of a Gaussian sample's variance. The formula's value lies inside it, so the two agree to within simulation error.

3.3 Standard errors

Definition 3.1 (Standard error of a coefficient)**.** The **standard error** of $\hat{\beta}_j$ is its standard deviation over repeated samples,

$$\mathrm{SE}\left(\hat{\beta}_j\right) = \sigma\sqrt{[(\mathbf{D}^\top\mathbf{D})^{-1}]_{jj}},$$

by Theorem 3.1.

Example 3.2 (Standard errors when σ is known)**.** In the simulation of Example 3.1, the noise level σ is known, so we can compute the standard errors of Definition 3.1 exactly:

3.3.1 Python

```
se_known_sigma = sigma_simple * np.sqrt(np.diag(PtP_inv))
print(se_known_sigma)
```

```
[0.71749866 0.0064455 ]
```

3.3.2 R

```
se_known_sigma <- sigma_simple * sqrt(diag(xtx_inv))
se_known_sigma
```

```
(Intercept)  horsepower
0.717498656 0.006445501
```

The slope's standard error is 0.00645. It is one fixed number for the whole simulation, and it matches the spread of the 2000 simulated slopes, whose standard deviation is 0.00648.

Definition 3.2 (Estimated standard error)**.** In practice the noise level σ in Definition 3.1 is unknown. Replacing it by the residual standard error, $\hat{\sigma} = \mathrm{RSE}$, gives the **estimated standard error**

$$\widehat{\mathrm{se}}\left(\hat{\beta}_j\right) = \hat{\sigma}\sqrt{[(\mathbf{D}^\top\mathbf{D})^{-1}]_{jj}}.$$

Software output, and most books, call $\widehat{\mathrm{se}}\left(\hat{\beta}_j\right)$ "the standard error" too [1, Sec. 3.1.2], p. 75.

Example 3.3 (The slope's estimated standard error for the Auto data)**.**
For the one-feature fit, compute $\widehat{\mathrm{se}}\left(\hat{\beta}_1\right)$ from Definition 3.2, and compare it with the standard error in the table from R's parameters package:

3.3.3 Python

```
se_simple = sigma_simple * np.sqrt(np.diag(PtP_inv))
print(se_simple)
```

```
[0.71749866 0.0064455 ]
```

3.3.4 R

```
se_by_hand <- sigma_simple * sqrt(diag(xtx_inv))
se_by_hand
```

```
(Intercept)  horsepower
0.717498656 0.006445501
```

```
parameters::parameters(fit_simple) |>
  parameters::print_md(digits = 4)
```

Parameter	Coefficient	SE	95% CI	t(390)	p
(Intercept)	39.9359	0.7175	(38.5252, 41.3465)	55.6598	$< .001$
horsepower	-0.1578	0.0064	(-0.1705, -0.1452)	-24.4891	$< .001$

The residual standard error is $\hat{\sigma} = 4.91$, and $\widehat{\mathrm{se}}\left(\hat{\beta}_1\right) = 0.00645$, the same value as the standard error column of the parameters table. It also equals the standard error of Example 3.2, because that simulation set σ to this same $\hat{\sigma}$. The slope $\hat{\beta}_1 = -0.158$ is about 24 standard errors away from 0. Unlike the standard error, the estimate changes from one data set to the next. Across the 2000 simulated data sets of Example 3.1, $\widehat{\mathrm{se}}\left(\hat{\beta}_1\right)$ ranges from 0.0057 to 0.00727, around the standard error 0.00645 of Example 3.2.

3.4 The t distribution of a coefficient

Theorem 3.2 (The standardized coefficient has a t distribution)**.** *Under the assumptions of Theorem 3.1, suppose also that the noise is Gaussian, $\vec{\varepsilon} \sim \mathrm{N}(\mathbf{0}, \sigma^2 \mathbf{I}_n)$. Then for each coefficient,*

$$T_j = \frac{\hat{\beta}_j - \beta_j}{\widehat{\mathrm{se}}\left(\hat{\beta}_j\right)} \sim t_{n-p^*},$$

the t distribution[a] with $n - p^*$ degrees of freedom [1, Sec. 3.1.2], p. 77.

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

The proof needs the distribution of $\hat{\sigma}^2$ and its independence from $\hat{\vec{\beta}}$, which these notes do not cover. James et al. [1, Sec. 3.1.2] states the result for simple linear regression, and the rme notes[8] state it for multiple regression.

Example 3.4 (Checking the t distribution by simulation)**.** For each of the 2000 simulated samples of Example 3.1, compute $T_1 = (\hat{\beta}_1 - \beta_1)/\widehat{\mathrm{se}}\left(\hat{\beta}_1\right)$. By Theorem 3.2, about 5% of them should land outside $\pm t_{n-2}(0.975)$, the 97.5th percentile of t_{n-2}:

3.4.1 Python

```
from scipy import stats

t_sim = (sim_slopes - beta_simple[1]) / sim_ses
t_crit = stats.t.ppf(0.975, df=n - 2)
n_outside = int(np.sum(np.abs(t_sim) > t_crit))
# An exact (Clopper-Pearson) 95% interval for the proportion outside.
outside_ci = stats.binomtest(n_outside, len(t_sim)).proportion_ci(
    confidence_level=0.95, method="exact")
outside_lo, outside_hi = outside_ci.low, outside_ci.high
print(t_crit, n_outside / len(t_sim), outside_lo, outside_hi)
```

```
1.9660653313011052 0.0495 0.04041037792654773 0.05993731615842642
```

[8] https://morrison-lab.github.io/rme/chapters/Linear-models-overview.html#wald-tests-and-confidence-intervals

3.4.2 R

```
t_sim <- (sim_fits["Estimate", ] - coef(fit_simple)[["horsepower"]]) /
  sim_fits["Std. Error", ]
t_crit <- qt(0.975, df = fit_simple$df.residual)
n_outside <- sum(abs(t_sim) > t_crit)
# binom.test gives an exact (Clopper-Pearson) 95% interval
# for the proportion outside.
outside_ci <- binom.test(n_outside, length(t_sim))$conf.int
c(t_crit = t_crit, outside = n_outside / length(t_sim),
  lower = outside_ci[1], upper = outside_ci[2])
```

```
    t_crit    outside      lower      upper
1.96606533 0.04400000 0.03543624 0.05393046
```

With $n - 2 = 390$ degrees of freedom, $t_{n-2}(0.975) = 1.966$, and 5.0% of the simulated T_1 values land outside $\pm$ 1.966, with an exact 95% confidence interval for that proportion from 4.0% to 6.0%. The interval covers the nominal 5%, and it measures only the simulation error of 2000 draws.

3.5 Confidence intervals

Exercise 3.4 (An interval that catches the true coefficient)**.** Use Theorem 3.2 to find an interval, computed from the data, that contains the true β_j with probability 0.95.

Solution

Solution 3.4. Write $c = t_{n-p^*}(0.975)$ for the 97.5th percentile of t_{n-p^*}. By Theorem 3.2, $T_j \sim t_{n-p^*}$, and the t distribution is symmetric about 0, so

$$\begin{aligned}
0.95 &= \mathbb{P}\big(-c \le T_j \le c\big) && \text{(definition of } c\text{; symmetry)} \\
&= \mathbb{P}\left(-c \le \frac{\hat\beta_j - \beta_j}{\widehat{\mathrm{se}}\big(\hat\beta_j\big)} \le c\right) && \text{(definition of } T_j\text{)} \\
&= \mathbb{P}\big(-c\,\widehat{\mathrm{se}}\big(\hat\beta_j\big) \le \hat\beta_j - \beta_j \le c\,\widehat{\mathrm{se}}\big(\hat\beta_j\big)\big) && \text{(multiply by } \widehat{\mathrm{se}}\big(\hat\beta_j\big) > 0\text{)} \\
&= \mathbb{P}\big(\hat\beta_j - c\,\widehat{\mathrm{se}}\big(\hat\beta_j\big) \le \beta_j \le \hat\beta_j + c\,\widehat{\mathrm{se}}\big(\hat\beta_j\big)\big) && \text{(rearrange for } \beta_j\text{)}
\end{aligned}$$

So the interval $\hat\beta_j \pm c\,\widehat{\mathrm{se}}\big(\hat\beta_j\big)$ contains β_j with probability 0.95. The interval is random, and β_j is fixed.

Theorem 3.3 (The coefficient interval has probability 0.95 of containing the coefficient)**.** *Let* $c = t_{n-p^*}(0.975)$*. Under the assumptions of Theorem 3.2,*

$$\mathbb{P}\big(\hat\beta_j - c\,\widehat{\mathrm{se}}\big(\hat\beta_j\big) \le \beta_j \le \hat\beta_j + c\,\widehat{\mathrm{se}}\big(\hat\beta_j\big)\big) = 0.95.$$

The interval is random, and β_j *is fixed.*

Proof

Proof. This is the solution to Exercise 3.4: see Solution 3.4. □

Example 3.5 (Coverage of the coefficient interval in simulation)**.** Simulate 5000 data sets of $n = 30$ points from $y = 1 + 2x + \varepsilon$ with $\varepsilon \sim \mathrm{N}(0, 1)$. For each, build the 95% interval for the slope of Definition 3.3 and check whether it contains the true slope 2.

```r
# Save and restore the random seed, so this example does not change later random output.
ci_old_seed <- if (exists(".Random.seed", globalenv())) get(".Random.seed", globalenv())
set.seed(571)
ci_covered <- replicate(5000, {
  x <- runif(30)
  y <- 1 + 2 * x + rnorm(30)
  ci <- confint(lm(y ~ x))["x", ]
  ci[1] <= 2 && 2 <= ci[2]
})
if (!is.null(ci_old_seed)) assign(".Random.seed", ci_old_seed, globalenv())
mean(ci_covered)
```

```
[1] 0.9532
```

The intervals contain the true slope in 95.3% of the data sets, close to the stated 95%.

Definition 3.3 (Confidence interval for a coefficient)**.** The 95% confidence interval[a] for β_j is

$$\hat{\beta}_j \pm t_{n-p^*}(0.975)\,\widehat{\mathrm{se}}\big(\hat{\beta}_j\big).$$

Over repeated samples, 95% of the intervals built this way contain the true β_j, by Theorem 3.3. For large n, $t_{n-p^*}(0.975)$ is close to 2, and James et al. [1, Sec. 3.1.2], p. 75 writes the interval as $\hat{\beta}_j \pm 2\,\widehat{\mathrm{se}}\big(\hat{\beta}_j\big)$.

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

Example 3.6 (A confidence interval for the horsepower slope)**.**
Compute the 95% confidence interval[a] for the slope[b] of the one-feature fit from Definition 3.3, and compare it with the 95% confidence interval columns of the table from R's parameters package:

3.5.1 Python

```python
ci_slope = beta_simple[1] + np.array([-1, 1]) * t_crit * se_simple[1]
print(ci_slope)
```

```
[-0.17051701 -0.14517246]
```

3.5.2 R

```r
coef(fit_simple)[["horsepower"]] + c(-1, 1) * t_crit * se_by_hand[[2]]
```

```
[1] -0.1705170 -0.1451725
```

```r
parameters::parameters(fit_simple) |>
  parameters::print_md(digits = 4, ci_digits = 4)
```

Parameter	Coefficient	SE	95% CI	t(390)	p
(Intercept)	39.9359	0.7175	(38.5252, 41.3465)	55.6598	< .001
horsepower	-0.1578	0.0064	(-0.1705, -0.1452)	-24.4891	< .001

The interval runs from -0.171 to -0.145 miles per gallon per horsepower. Every value in the interval is negative, so the interval excludes a slope of zero.

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

3.6 Testing whether a coefficient is zero

Definition 3.4 (The t-test for a coefficient)**.** To test the null hypothesis[a] $H_0 : \beta_j = 0$ (feature j adds nothing to the model that already contains the other features) against $H_A : \beta_j \neq 0$, compute the **t-statistic**

$$t_j = \frac{\hat{\beta}_j}{\widehat{\mathrm{se}}\big(\hat{\beta}_j\big)}.$$

Under H_0, t_j has the t_{n-p^*} distribution, by Theorem 3.2. The p-value[b] is the probability, under H_0, of a statistic at least as far from 0:

$$\begin{aligned} p &= \mathbb{P}\big(|T_{n-p^*}| \geq |t_j|\big) \\ &= 2\,\mathbb{P}\big(T_{n-p^*} \geq |t_j|\big). \end{aligned}$$

A small p-value is evidence against H_0 [1, Sec. 3.1.2], p. 77.

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

Example 3.7 (Does horsepower matter once weight is in the model?)**.**
For the two-feature fit, compute the t-statistic and p-value for `horsepower` from Definition 3.4, and compare them with the table from R's parameters package:

3.6.1 Python

```
p_star = Phi_multiple.shape[1]
rse_multiple = np.sqrt(
    np.sum((y - Phi_multiple @ beta_multiple) ** 2) / (n - p_star)
)
se_multiple = rse_multiple * np.sqrt(
    np.diag(np.linalg.inv(Phi_multiple.T @ Phi_multiple))
)
t_multiple = beta_multiple / se_multiple
p_multiple = 2 * stats.t.sf(np.abs(t_multiple), df=n - p_star)
print(t_multiple)
```

```
[ 57.5396503   -4.26725251 -11.53463263]
```

```
print(p_multiple)
```

```
[2.31711323e-192 2.48848204e-005 1.12436236e-026]
```

3.6.2 R

```
parameters::parameters(fit_multiple) |>
  parameters::print_md(digits = 4)
```

Parameter	Coefficient	SE	95% CI	t(389)	p
(Intercept)	45.6402	0.7932	(44.0807, 47.1997)	57.5397	$< .001$
horsepower	-0.0473	0.0111	(-0.0691, -0.0255)	-4.2673	$< .001$
weight	-0.0058	0.0005	(-0.0068, -0.0048)	-11.5346	$< .001$

For `horsepower`, t = -4.27 with $n - p^* = 389$ degrees of freedom, so $p = 2.5 \times 10^{-5}$. Even with `weight` held fixed, the data are unlikely under $H_0 : \beta_{\text{hp}} = 0$. The t-statistic for `horsepower` is much smaller in size than in the one-feature fit, where it was -24.5, because `horsepower` and `weight` carry much of the same information.

3.7 Is any feature useful?

Exercise 3.5 (Why not just check each t-test?)**.** Suppose a model has $p = 100$ features, and none of them is related to the response:

$$\begin{aligned}\beta_1 &= \beta_2 \\ &= \cdots \\ &= \beta_{100} \\ &= 0.\end{aligned}$$

We run the t-test of Definition 3.4 on each coefficient at level 0.05. About how many of the 100 p-values will fall below 0.05?

Solution

Solution 3.5. Under $H_0 : \beta_j = 0$, each p-value falls below 0.05 with probability 0.05. So we expect about $100 \times 0.05 = 5$ of them to fall below 0.05 by chance, even though no feature is related to the response.

Exercise 3.6 (Why not just check each t-test? (part 2))**.** Use the setup of Exercise 3.5.
What does that say about using the separate t-tests to decide whether any feature is related to the response?

Solution

Solution 3.6. If we declared a relationship whenever any one p-value was small, we would almost always find one. We need a single test of all the coefficients together, whose chance of a false alarm is 0.05 no matter how many features there are [1, Sec. 3.2.2], p. 86.

Definition 3.5 (The F-test that any feature is useful)**.** To test

$$H_0 : \beta_j = 0 \text{ for every } j \in \{1, \dots, p\} \quad \text{against} \quad H_A : \text{at least one } \beta_j \neq 0,$$

compute the **F-statistic**

$$F = \frac{(\text{TSS} - \text{RSS})/p}{\text{RSS}/(n - p - 1)},$$

where $\text{TSS} = \sum_i (y_i - \bar{y})^2$ is the total sum of squares and $\text{RSS} = \sum_i (y_i - \hat{y}_i)^2$ is the residual sum of squares [1, Sec. 3.2.2], p. 84. If H_0 holds and the noise is Gaussian, F has the F distribution[a] with p and $n - p - 1$ degrees of freedom [1, Sec. 3.2.2], p. 85. The p-value is $\mathbb{P}\big(F_{p,n-p-1} \geq F\big)$.
The numerator is the drop in squared error per feature, and the denominator is $\hat{\sigma}^2$, the squared residual standard error. Under H_0, both estimate σ^2, so F is near 1; when some feature is useful, the numerator is larger, and so is F.

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

Example 3.8 (The F-test for horsepower and weight)**.**
For the two-feature fit, compute F and its p-value from Definition 3.5, and compare them with the F-statistic that `broom::glance()` reports in R:

3.7.1 Python

```
tss = np.sum((y - y.mean()) ** 2)
rss_multiple = np.sum((y - Phi_multiple @ beta_multiple) ** 2)
p_features = 2
f_multiple = ((tss - rss_multiple) / p_features) / (
    rss_multiple / (n - p_features - 1)
)
print(f_multiple, stats.f.sf(f_multiple, p_features, n - p_features - 1))
```

```
467.91015349994683 3.059606299669705e-104
```

3.7.2 R

```
broom::glance(fit_multiple) |>
  dplyr::select(statistic, p.value, df, df.residual)
```

```
# A tibble: 1 x 4
  statistic   p.value    df df.residual
      <dbl>     <dbl> <dbl>       <int>
1      468. 3.06e-104     2         389
```

$F = 467.9$ on 2 and 389 degrees of freedom, far above 1. Its p-value is below 10^{-100}, so at least one of `horsepower` and `weight` is related to `mpg`.

3.8 Testing a group of features

Definition 3.6 (The partial F-test for a group of features)**.** To test whether a group of q features adds anything to a model that already contains the other $p-q$, fit the model twice: once with all p features, with residual sum of squares RSS, and once without the q features, with residual sum of squares RSS_0. The **partial F-statistic** is

$$F = \frac{(\text{RSS}_0 - \text{RSS})/q}{\text{RSS}/(n-p-1)}$$

[1, Sec. 3.2.2], p. 85. If the q coefficients are all 0 and the noise is Gaussian, F has the $F_{q,n-p-1}$ distribution. Definition 3.5 is the case $q = p$, where the smaller model has only an intercept and $\text{RSS}_0 = \text{TSS}$. With $q = 1$, the partial F-statistic is the square of that feature's t-statistic.

The rme notes derive this test and connect it to the likelihood ratio test; see the partial F-test[9].

Example 3.9 (Does origin matter once weight is known?)**.**
The origin fit fits `mpg` on `weight` and the two origin indicators. Test whether the two indicators together add anything to `weight` alone, so $q = 2$ and $p = 3$:

3.8.1 Python

[9] https://morrison-lab.github.io/rme/chapters/Linear-models-overview.html#sec-partial-F-test

```
beta_weight, Phi_weight = fit([weight])
rss_0 = np.sum((y - Phi_weight @ beta_weight) ** 2)
rss_1 = np.sum((y - Phi_origin_weight @ beta_origin_weight) ** 2)
f_origin = ((rss_0 - rss_1) / 2) / (rss_1 / (n - 3 - 1))
print(f_origin, stats.f.sf(f_origin, 2, n - 3 - 1))
```

```
6.126802214577667 0.002400578748351605
```

3.8.2 R

```
fit_weight <- lm(mpg ~ weight, data = auto)
anova(fit_weight, fit_origin_weight) |>
  parameters::parameters() |>
  parameters::print_md(footer = "")
```

Parameter	RSS	Sum_Squares	df	df (error)	Mean_Square	F	p
Model 1: mpg ~ weight	7321.23			390			
Model 2: mpg ~ weight + factor(origin)	7097.10	224.14	2	388	112.07	6.13	0.002

$F = 6.13$ on 2 and 388 degrees of freedom, with $p = 0.0024$. The data are unlikely if origin adds nothing once weight is known, even though the Auto lab found that weight explains much of the raw gap between origins.

3.9 Confidence and prediction intervals

Exercise 3.7 (Two questions about a 98-horsepower car)**.** The one-feature fit predicts 24.5 miles per gallon for a car with 98 horsepower. Which sources of error make that number uncertain as an estimate[a] of the *average* `mpg` of all 98-horsepower cars?

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

Solution

Solution 3.7. In Exercise 3.7, the average `mpg` at 98 horsepower is $\phi(\underset{\sim}{x}_0) \cdot \vec{\beta}$, with $\phi(\underset{\sim}{x}_0) = [1, 98]^\top$. Its estimate[a] $\phi(\underset{\sim}{x}_0) \cdot \hat{\vec{\beta}}$ is off only because $\hat{\vec{\beta}}$ differs from $\vec{\beta}$: the reducible error of the error decomposition.

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

Exercise 3.8 (Two questions about a 98-horsepower car (part 2))**.** Use the setup of Exercise 3.7. Which sources of error make that number uncertain as a guess at the `mpg` of *one particular* 98-horsepower car?

Solution

Solution 3.8. One car's `mpg` is $y_0 = \phi(\underset{\sim}{x}_0) \cdot \vec{\beta} + \varepsilon_0$. The guess has the same error from $\hat{\vec{\beta}}$, plus the car's own noise ε_0, whose variance[a] is the residual variance.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance

Exercise 3.9 (Two questions about a 98-horsepower car (part 3))**.** Use the setup of Exercise 3.7. Which of the two should we be less sure about?

Solution

Solution 3.9. We should be less sure about the single car, because its error adds the noise ε_0 to the error in $\hat{\vec{\beta}}$ [1, Sec. 3.2.2], p. 90.

Exercise 3.10 (The variance of a prediction)**.** Under the assumptions of Theorem 3.1, let $\phi_0 = \phi(\underset{\sim}{x}_0)$ be the feature vector of a new input, let $\hat{y}_0 = \phi_0 \cdot \hat{\vec{\beta}}$, and let $y_0 = \phi_0 \cdot \vec{\beta} + \varepsilon_0$, where ε_0 has mean 0 and variance σ^2 and is independent of the training noise $\vec{\varepsilon}$. Find
$\mathrm{Var}(\hat{y}_0)$, and

Solution

Solution 3.10. Here $\underset{\sim}{x}_0$ is the new input[a] of Exercise 3.10, ϕ_0 is its feature vector, ε_0 is the noise in the new outcome[b] Y_0 at $\underset{\sim}{x}_0$, and σ^2 is the noise variance[c] of Theorem 3.1.
$\hat{Y}_0 = \phi_0 \cdot \hat{\vec{\beta}}$ is a linear combination[d] of the entries of $\hat{\vec{\beta}}$ with fixed weights ϕ_0, and Theorem 3.1 gives its covariance, so

$$\begin{aligned}
\mathrm{Var}\left(\hat{Y}_0\right) &= {\phi_0}^\top \mathrm{Cov}\left(\hat{\vec{\beta}}\right) \phi_0 && \text{(variance of a fixed linear combination)} \\
&= {\phi_0}^\top \sigma^2 (\mathbf{D}^\top \mathbf{D})^{-1} \phi_0 && \text{(covariance of } \hat{\vec{\beta}}\text{)} \\
&= \sigma^2 {\phi_0}^\top (\mathbf{D}^\top \mathbf{D})^{-1} \phi_0 && \text{(move the noise variance } \sigma^2 \text{ out)} \\
&= \sigma^2 h_0 && \text{(let } h_0 \text{ be the leverage of the new input, } {\phi_0}^\top (\mathbf{D}^\top \mathbf{D})^{-1} \phi_0\text{)}
\end{aligned}$$

[a] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[b] https://morrison-lab.github.io/sds/model-validation.html#def-data-set
[c] https://morrison-lab.github.io/pds/variance-covariance.html#def-variance
[d] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-linear-combination

Exercise 3.11 (The variance of a prediction (part 2))**.** Use the setup of Exercise 3.10.
$\mathrm{Var}(y_0 - \hat{y}_0)$.

Solution

Solution 3.11. $Y_0 - \hat{Y}_0 = \phi_0 \cdot \vec{\beta} + \varepsilon_0 - \hat{Y}_0$, and $\phi_0 \cdot \vec{\beta}$ is a constant[a]. The new noise ε_0 is independent[b] of $\hat{\vec{\beta}}$, which depends only on the training noise, so

$$\begin{aligned}
\mathrm{Var}\left(Y_0 - \hat{Y}_0\right) &= \mathrm{Var}(\varepsilon_0) + \mathrm{Var}\left(\hat{Y}_0\right) && \text{(constants add no variance; independent terms)} \\
&= \sigma^2 + \sigma^2 h_0 && \text{(@sol-prediction-variance)} \\
&= \sigma^2 (1 + h_0) && \text{(factor out } \sigma^2\text{)}
\end{aligned}$$

The extra 1 is the residual variance of Solution 3.8.

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-constant-function
[b] https://morrison-lab.github.io/pds/independence.html#def-indpt

Theorem 3.4 (The variance of a prediction)**.** *Under the assumptions of Theorem 3.1, let* $\phi_0 = \phi(\underset{\sim}{x}_0)$ *be the feature vector of a new input and* $\hat{y}_0 = \phi_0 \cdot \hat{\vec{\beta}}$*. Then*

$$\mathrm{Var}(\hat{y}_0) = \sigma^2 {\phi_0}^\top (\mathbf{D}^\top \mathbf{D})^{-1} \phi_0.$$

Proof

Proof. This is the solution to Exercise 3.10: see Solution 3.10. □

Example 3.10 (The variance of a prediction at a new input)**.** Fit a line to the inputs $x = 1, 2, 3, 4, 5$ with noise variance $\sigma^2 = 4$, and predict at $x_0 = 3.5$, so $\phi_0 = [1, 3.5]^\top$.

```
pv_Phi <- cbind(1, 1:5)
pv_phi0 <- c(1, 3.5)
pv_sigma2 <- 4
pv_h0 <- drop(t(pv_phi0) %*% solve(t(pv_Phi) %*% pv_Phi) %*% pv_phi0)
c(h0 = pv_h0, var_prediction = pv_sigma2 * pv_h0)
```

```
            h0 var_prediction
         0.225          0.900
```

By Theorem 3.4, $\mathrm{Var}(\hat{y}_0) = 4 \times 0.225 = 0.9$. The variance is smallest at the center of the inputs, $x_0 = 3$, and grows as x_0 moves away from it.

Theorem 3.5 (The variance of a prediction error)**.** *In the setting of Theorem 3.4, let* $y_0 = \phi_0 \cdot \vec{\beta} + \varepsilon_0$, *where* ε_0 *has mean* 0 *and variance* σ^2 *and is independent of the training noise. Then*

$$\mathrm{Var}(y_0 - \hat{y}_0) = \sigma^2\big(1 + \phi_0{}^\top (\mathbf{D}^\top \mathbf{D})^{-1} \phi_0\big).$$

Proof

Proof. This is the solution to Exercise 3.11: see Solution 3.10. □

Example 3.11 (The variance of a prediction error)**.** Continue Example 3.10, with $\sigma^2 = 4$ and $\phi_0 = [1, 3.5]^\top$ for the inputs $x = 1, \ldots, 5$.

```
pe_Phi <- cbind(1, 1:5)
pe_phi0 <- c(1, 3.5)
pe_sigma2 <- 4
pe_h0 <- drop(t(pe_phi0) %*% solve(t(pe_Phi) %*% pe_Phi) %*% pe_phi0)
c(var_prediction = pe_sigma2 * pe_h0, var_error = pe_sigma2 * (1 + pe_h0))
```

```
var_prediction      var_error
           0.9            4.9
```

By Theorem 3.5, $\mathrm{Var}(y_0 - \hat{y}_0) = 4 \times (1 + 0.225) = 4.9$. Of this, 4 is the new response's own noise, which no amount of training data removes.

Definition 3.7 (Confidence interval for the mean response)**.** The 95% confidence interval for the mean response $\phi_0 \cdot \vec{\beta}$ at a new input $\underset{\sim}{x}_0$ is

$$\hat{y}_0 \pm t_{n-p^*}(0.975)\, \hat{\sigma} \sqrt{\phi_0{}^\top (\mathbf{D}^\top \mathbf{D})^{-1} \phi_0},$$

where $\hat{y}_0 = \phi_0 \cdot \hat{\vec{\beta}}$ and $\hat{\sigma}$ is the residual standard error. It is Definition 3.3 with the variance of Theorem 3.4 in place of $\mathrm{Var}\big(\hat{\beta}_j\big)$.

Example 3.12 (Average fuel economy at 98 horsepower)**.**
For the one-feature fit and $\phi_0 = [1, 98]^\top$, compute the confidence interval of Definition 3.7, and compare it with R's `predict(..., interval = "confidence")`:

3.9.1 Python

```
# PtP_inv and sigma_simple come from the covariance example,
# and t_crit from the t-distribution example, earlier in this chapter.
phi_0 = np.array([1, 98])
y_hat_0 = phi_0 @ beta_simple
h_0 = phi_0 @ PtP_inv @ phi_0
mean_ci = y_hat_0 + np.array([-1, 1]) * t_crit * sigma_simple * np.sqrt(h_0)
print(y_hat_0, mean_ci)
```

```
24.467077152512417 [23.97307896 24.96107534]
```

3.9.2 R

```
predict(fit_simple, tibble::tibble(horsepower = 98), interval = "confidence")
```

```
       fit      lwr      upr
1 24.46708 23.97308 24.96108
```

The interval runs from 23.97 to 24.96 miles per gallon. It is narrow, because 392 cars pin down the line well near 98 horsepower.

Definition 3.8 (Prediction interval)**.** The 95% **prediction interval** for a single new response y_0 at input $\underset{\sim}{x}_0$ is

$$\hat{y}_0 \pm t_{n-p^*}(0.975)\,\hat{\sigma}\sqrt{1 + \phi_0{}^\top (\mathbf{D}^\top \mathbf{D})^{-1}\phi_0}.$$

If the noise is Gaussian, it contains the new response y_0 with probability 0.95 [4].

Example 3.13 (One car at 98 horsepower)**.** Compute the prediction interval of Definition 3.8 at 98 horsepower, compare it with R's `predict(..., interval = "prediction")`, and draw both intervals across the range of horsepower:

3.9.3 Python

```
pred_int = y_hat_0 + np.array([-1, 1]) * t_crit * sigma_simple * np.sqrt(1 + h_0)
print(pred_int)
```

```
[14.80939607 34.12475823]
```

```
# A function, so the plotting calls' return values are not printed.
def plot_prediction_bands():
    grid = np.linspace(horsepower.min(), horsepower.max(), 100)
    Phi_grid = np.column_stack([np.ones_like(grid), grid])
    fitted_grid = Phi_grid @ beta_simple
    h_grid = np.sum(Phi_grid @ PtP_inv * Phi_grid, axis=1)
    half_mean = t_crit * sigma_simple * np.sqrt(h_grid)
    half_pred = t_crit * sigma_simple * np.sqrt(1 + h_grid)
    fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
    ax.scatter(horsepower, y, s=12, alpha=0.4, color="grey");
    ax.plot(grid, fitted_grid, color="C3", label="Fitted line");
    ax.fill_between(grid, fitted_grid - half_mean, fitted_grid + half_mean,
                    color="C3", alpha=0.3, label="95% confidence band");
    ax.plot(grid, fitted_grid - half_pred, color="C0", linestyle="--",
            label="95% prediction band");
    ax.plot(grid, fitted_grid + half_pred, color="C0", linestyle="--");
    ax.set_xlabel("Horsepower");
    ax.set_ylabel("Miles per gallon");
    fig.legend(loc="outside lower center", ncol=3);
    plt.show()

plot_prediction_bands()
```

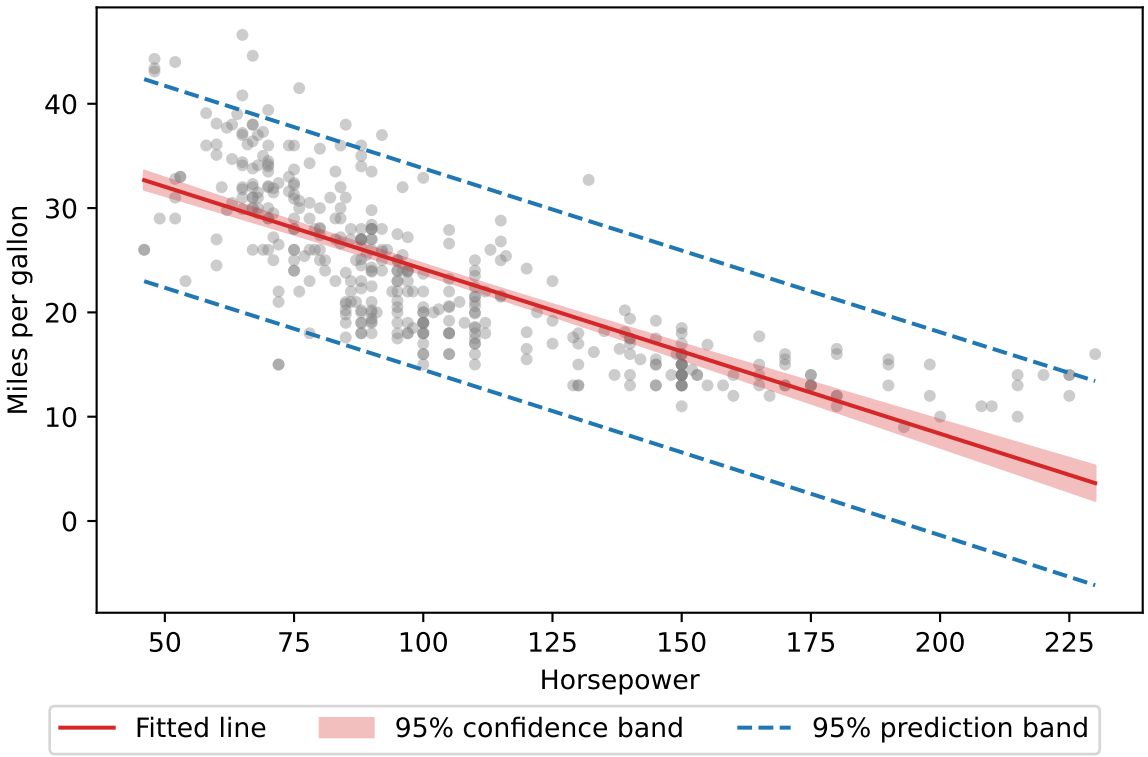


Figure 4: Fuel economy against horsepower for the Auto data, with the least squares line, its 95% confidence band for the mean response, and its 95% prediction band.

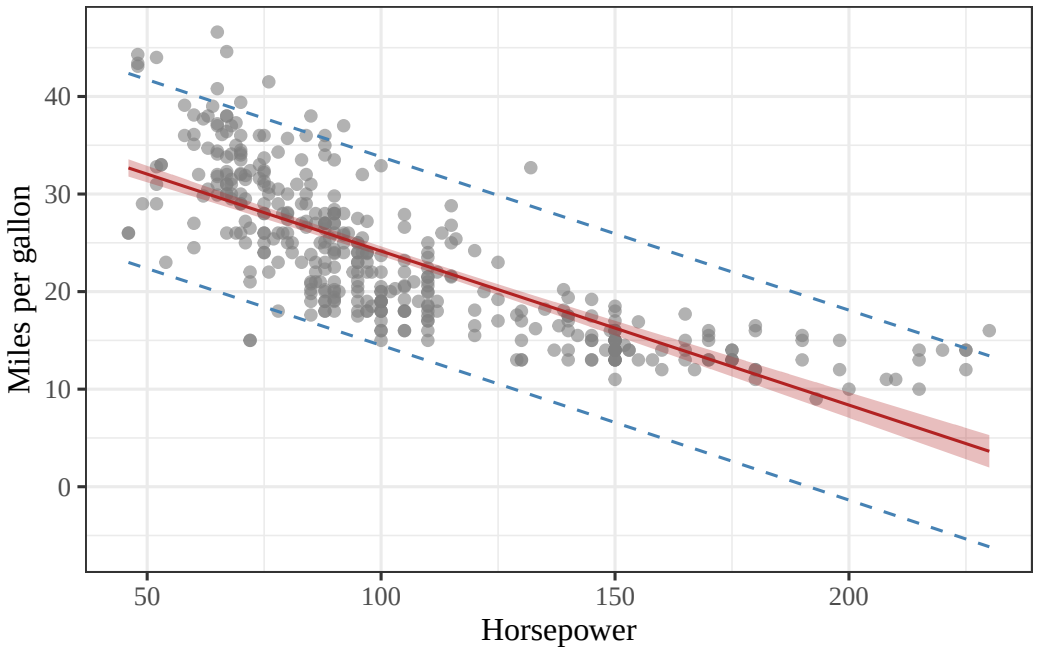
3.9.4 R

```
predict(fit_simple, tibble::tibble(horsepower = 98), interval = "prediction")
```

```
       fit     lwr      upr
1 24.46708 14.8094 34.12476
```

```
library(ggplot2)

grid <- tibble::tibble(horsepower = seq(min(auto$horsepower),
                                        max(auto$horsepower), length.out = 100))
band_mean <- predict(fit_simple, grid, interval = "confidence")
band_pred <- predict(fit_simple, grid, interval = "prediction")
bands <- tibble::tibble(grid, fit = band_mean[, "fit"],
                        mean_lwr = band_mean[, "lwr"], mean_upr = band_mean[, "upr"],
                        pred_lwr = band_pred[, "lwr"], pred_upr = band_pred[, "upr"])
ggplot(bands, aes(horsepower)) +
  geom_point(data = auto, aes(y = mpg), colour = "grey50", alpha = 0.6) +
  geom_ribbon(aes(ymin = mean_lwr, ymax = mean_upr),
              fill = "firebrick", alpha = 0.3) +
  geom_line(aes(y = fit), colour = "firebrick") +
  geom_line(aes(y = pred_lwr), colour = "steelblue", linetype = "dashed") +
  geom_line(aes(y = pred_upr), colour = "steelblue", linetype = "dashed") +
  labs(x = "Horsepower", y = "Miles per gallon")
```



The prediction interval at 98 horsepower runs from 14.8 to 34.1 miles per gallon, about 20 times as wide as the confidence interval of Example 3.12. The number of cars controls the width of the confidence band, which would shrink toward the line with more data. The prediction band would not shrink below about $\pm 2\hat{\sigma}$, because no amount of data removes a new car's own noise.

Theorem 3.6 (The prediction interval is wider than the confidence interval)**.** *At the same input x_0 and the same level, the prediction interval of Definition 3.8 is wider than the confidence interval for the mean response of Definition 3.7. With $h_0 = \phi_0^{\top}(\mathbf{D}^{\top}\mathbf{D})^{-1}\phi_0$, the ratio of their widths is $\sqrt{(1+h_0)/h_0}$ [1, Sec. 3.2.2], p. 90.*

Proof

Proof. Both intervals have the center $\hat{y}_0$ and use the same $t_{n-p^*}(0.975)$ and $\hat{\sigma}$. Their half-widths are

$$t_{n-p^*}(0.975)\,\hat{\sigma}\sqrt{h_0} \quad \text{and} \quad t_{n-p^*}(0.975)\,\hat{\sigma}\sqrt{1+h_0}.$$

Because $1 + h_0 > h_0$, the second is larger. The ratio of the two is $\sqrt{(1+h_0)/h_0}$. The extra 1 is the variance of the new response's own noise. □

Example 3.14 (Both intervals at every car)**.** For each car of the `Auto` data, take its own horsepower as x_0. Compute the width of the confidence interval for the mean response and of the prediction interval, and check that the prediction interval is wider at every car, with the width ratio of Theorem 3.6:

3.9.5 Python

```
# Phi_simple, PtP_inv, sigma_simple and t_crit come from the examples earlier in this chapter.
h_cars = np.sum(Phi_simple @ PtP_inv * Phi_simple, axis=1)
width_mean = 2 * t_crit * sigma_simple * np.sqrt(h_cars)
width_pred = 2 * t_crit * sigma_simple * np.sqrt(1 + h_cars)
ratio_cars = width_pred / width_mean
print(np.all(width_pred > width_mean), ratio_cars.min(), ratio_cars.max())
```

```
True 5.883031030489702 19.82234421472044
```

3.9.6 R

```
conf_cars <- predict(fit_simple, auto, interval = "confidence")
pred_cars <- predict(fit_simple, auto, interval = "prediction")
width_conf <- conf_cars[, "upr"] - conf_cars[, "lwr"]
width_pred <- pred_cars[, "upr"] - pred_cars[, "lwr"]
c(all_wider = all(width_pred > width_conf),
  min_ratio = min(width_pred / width_conf),
  max_ratio = max(width_pred / width_conf))
```

```
all_wider min_ratio max_ratio
 1.000000  5.883031 19.822344
```

The prediction interval is wider than the confidence interval at 392 of the 392 cars. The ratio of widths runs from 5.9 to 19.8. It is smallest where h_0 is largest, at the extreme horsepower values, where the fitted line is least certain. Even there, the new car's own noise makes the prediction interval wider.

4 Checking the model

This section follows the list of potential problems in [1, Sec. 3.3.3]. The rme notes on diagnostics[10] go further, with quantile-quantile (QQ) plots, Cook's distance and formal tests.

4.1 Non-linearity

Exercise 4.1 (Plot the residuals)**.** For the one-feature fit, plot each car's residual $e_i = y_i - \hat{y}_i$ against its fitted value $\hat{y}_i$.

[10] https://morrison-lab.github.io/rme/chapters/Linear-models-overview.html#sec-diagnose-LMs

Solution

Solution 4.1.

4.1.1 Python

```
# A function, so the plotting calls' return values are not printed.
def plot_residuals(beta_hat, Phi, label):
    fitted_values = Phi @ beta_hat
    fig, ax = plt.subplots(figsize=(6, 4), layout="constrained")
    ax.scatter(fitted_values, y - fitted_values, s=12, alpha=0.5);
    ax.axhline(0, color="grey", linewidth=1);
    ax.set_xlabel(f"Fitted mpg ({label})");
    ax.set_ylabel("Residual");
    plt.show()

plot_residuals(beta_simple, Phi_simple, "straight line")
```

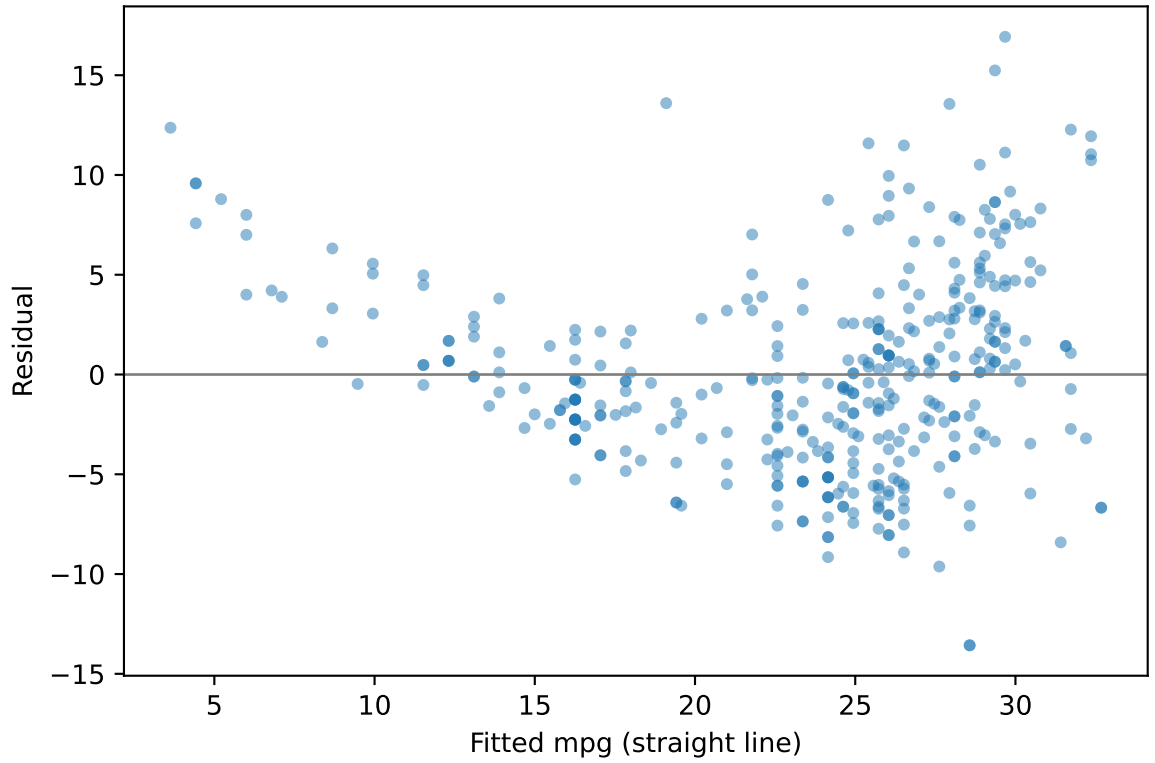
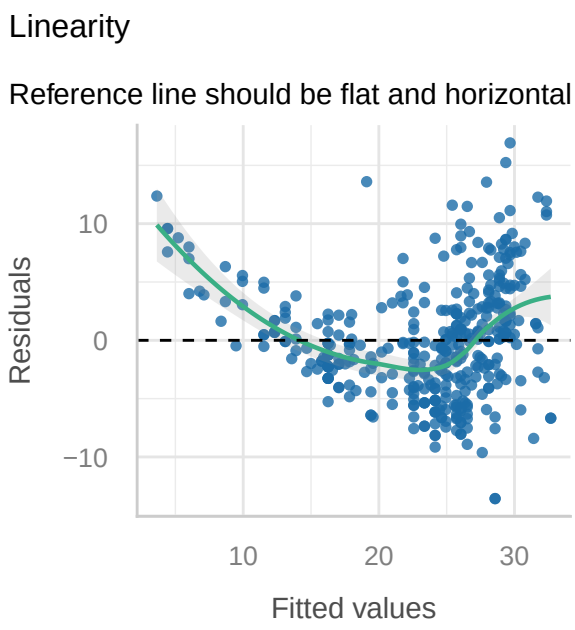


Figure 5: Residuals against fitted values for the straight-line fit of mpg on horsepower.

4.1.2 R

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



Exercise 4.2 (Plot the residuals (part 2))**.** Use the setup of Exercise 4.1.
If the straight line captured the relationship, what would the plot look like?

Solution

Solution 4.2. If the line captured the relationship, the residuals would scatter evenly above and below 0 at every fitted value, with no pattern.

Exercise 4.3 (Plot the residuals (part 3))**.** Use the setup of Exercise 4.1.
What do you see instead?

Solution

Solution 4.3. Instead they form a U: mostly positive at both ends and mostly negative in the middle. The line is too high in the middle of the horsepower range and too low at both ends, so the relationship is curved [1, Sec. 3.3.3], pp. 100–101.

Definition 4.1 (Residual plot)**.** A **residual plot** shows the residuals $e_i = y_i - \hat{y}_i$ against the fitted values $\hat{y}_i$ (or, in simple linear regression, against the feature). For a model that captures the relationship, the points scatter around 0 with no visible pattern; a curve in the plot is a sign of non-linearity [1, Sec. 3.3.3], p. 101.

Example 4.1 (The residual plot after adding horsepower^2)**.** The residual plot for the quadratic fit:

4.1.3 Python

```
plot_residuals(beta_quadratic, Phi_quadratic, "quadratic")
```

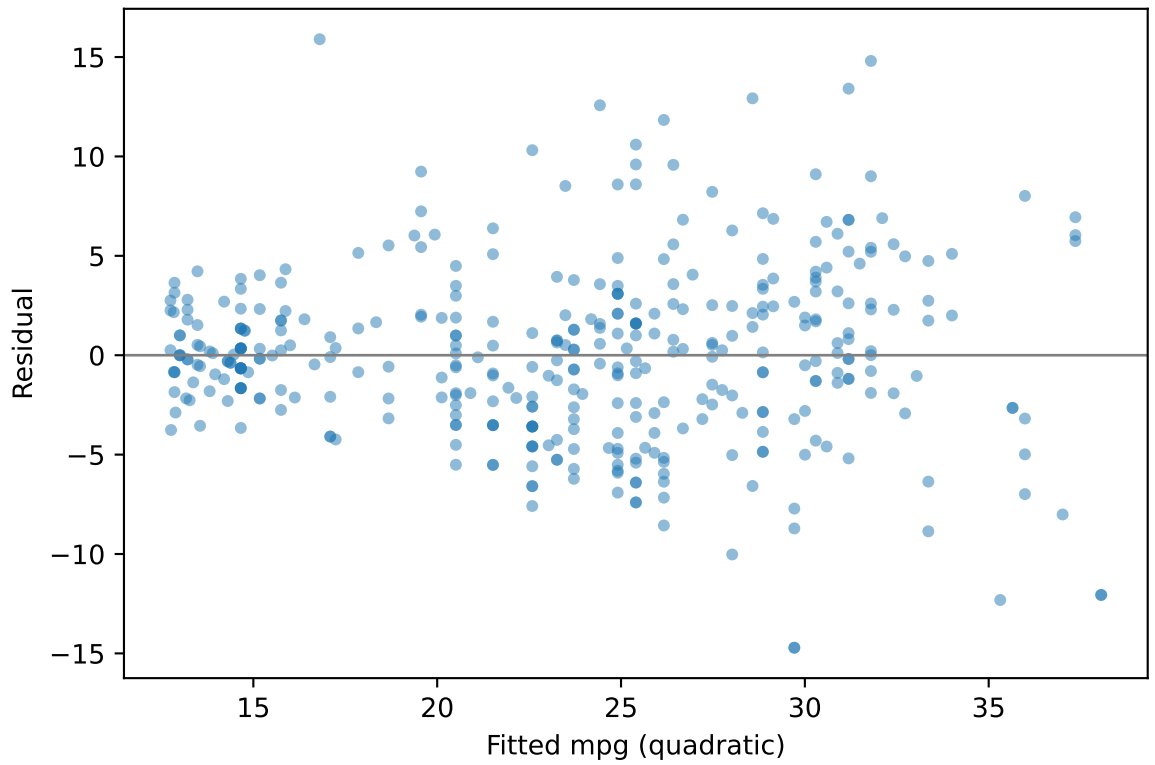
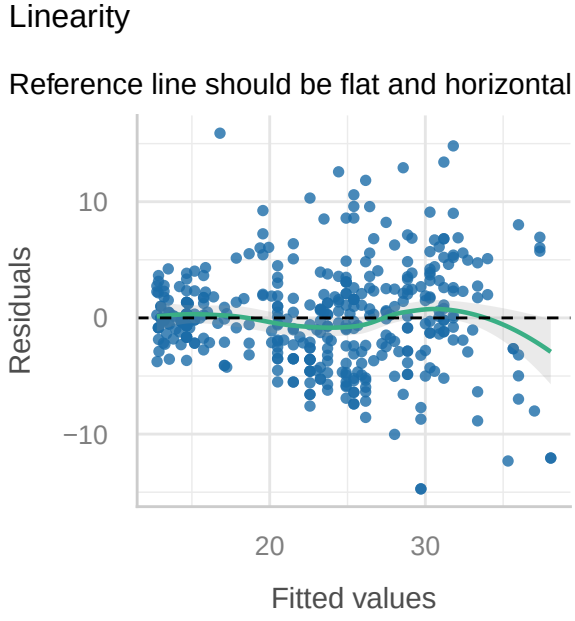


Figure 6: Residuals against fitted values for the quadratic fit of mpg on horsepower.

4.1.4 R

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



The U shape is gone, so the quadratic term captured the curve.

4.2 Non-constant variance

Exercise 4.4 (Does the spread change?)**.** In the residual plot of Example 4.1, split the cars into thirds by fitted value, and compute the standard deviation of the residuals in each third.

Solution

Solution 4.4.

4.2.1 Python

```
def spread_by_third(fitted_values, residuals):
    """Standard deviation of the residuals in each third of the fitted values."""
    cuts = np.quantile(fitted_values, [1 / 3, 2 / 3])
    third = np.searchsorted(cuts, fitted_values)
    return np.array([np.std(residuals[third == k], ddof=1) for k in range(3)])

fitted_quadratic = Phi_quadratic @ beta_quadratic
spread_quadratic = spread_by_third(fitted_quadratic, y - fitted_quadratic)
print(spread_quadratic)
```

```
[2.85532811 4.46613435 5.36238522]
```

```
# The spreads have no simple interval, so resample the cars.
# Each resample refits the quadratic, finds its own thirds, and recomputes the spreads.
def bootstrap_spread_ci(B=2000, seed=571):
    """95% percentile intervals for the three spreads and for the highest third's spread over the lowe
    rng = np.random.default_rng(seed)
    spreads = np.empty((B, 3))
    for b in range(B):
        rows = rng.integers(0, n, n)
        beta_b = np.linalg.lstsq(Phi_quadratic[rows], y[rows], rcond=None)[0]
        fitted_b = Phi_quadratic[rows] @ beta_b
        spreads[b] = spread_by_third(fitted_b, y[rows] - fitted_b)
    return (np.quantile(spreads, [0.025, 0.975], axis=0),
            np.quantile(spreads[:, 2] / spreads[:, 0], [0.025, 0.975]))

spread_ci_quadratic, spread_ratio_ci = bootstrap_spread_ci()
print(spread_ci_quadratic.T)
```

```
[[2.25359029 3.54058612]
 [3.74873411 5.05624762]
 [4.56392342 6.10287934]]
```

```
print(spread_quadratic[2] / spread_quadratic[0], spread_ratio_ci)
```

```
1.878027678194057 [1.42727074 2.46554721]
```

4.2.2 R

```r
spread_of_thirds <- function(fitted_values, residuals) {
  third <- cut(fitted_values, quantile(fitted_values, c(0, 1 / 3, 2 / 3, 1)),
               include.lowest = TRUE)
  tapply(residuals, third, sd)
}
spread_by_third <- function(model) {
  spread_of_thirds(fitted(model), residuals(model))
}
spread_quadratic_r <- spread_by_third(fit_quadratic)
spread_quadratic_r
```

```
[12.7,20.5] (20.5,26.4] (26.4,38.1]
   2.855328    4.466134    5.362385
```

```r
# The spreads have no simple interval, so resample the cars.
# Each resample refits the quadratic,
# finds its own thirds, and recomputes the spreads.
boot_spread_ci <- function(model, data, n_boot = 2000) {
  set.seed(571)
  design <- model.matrix(model)
  y <- data$mpg
  spreads <- vapply(seq_len(n_boot), \(b) {
    rows <- sample(nrow(design), replace = TRUE)
    residuals_b <- lm.fit(design[rows, ], y[rows])$residuals
    spread_of_thirds(y[rows] - residuals_b, residuals_b)
  }, numeric(3))
  list(spreads = apply(spreads, 1, quantile, probs = c(0.025, 0.975)),
       ratio = quantile(spreads[3, ] / spreads[1, ], c(0.025, 0.975)))
}
spread_ci_r <- boot_spread_ci(fit_quadratic, auto)
t(spread_ci_r$spreads)
```

```
                2.5%    97.5%
[13.5,22.4] 2.260491 3.576468
(22.4,26.9] 3.706911 5.048198
(26.9,36.7] 4.588323 6.087379
```

```r
spread_quadratic_r[[3]] / spread_quadratic_r[[1]]
```

```
[1] 1.878028
```

```r
spread_ci_r$ratio
```

```
    2.5%    97.5%
1.418753 2.458572
```

Exercise 4.5 (Does the spread change? (part 2))**.** Use the setup of Exercise 4.4. Is the spread of the residuals the same across the range of fitted values?

Solution

Solution 4.5. No. The standard deviation[a] grows from 2.9 (95% bootstrap interval 2.3 to 3.5) in the lowest third to 5.4 (4.6 to 6.1) in the highest, so the plot fans out like a funnel. The highest third's spread is 1.9 times the lowest third's (1.4 to 2.5), an interval that excludes 1, the ratio for equal spreads. Each interval is the 2.5th to 97.5th percentile over 2,000 resamples of the cars, refitting the quadratic and recomputing the thirds each time.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-sd

Exercise 4.6 (Does the spread change? (part 3))**.** Use the setup of Exercise 4.4. Which assumption of Theorem 3.1 does that bear on?

Solution

Solution 4.6. Theorem 3.1 assumes $\mathrm{Cov}(\vec{\varepsilon}) = \sigma^2 \mathbf{I}_n$, the same variance[a] σ^2 for every car. A funnel says the noise variance grows with the fitted value, which pds[b] calls *heteroskedasticity*. Then the standard errors[c] estimated[d] as in Definition 3.2, and the intervals and tests built on them, are no longer right [1, Sec. 3.3.3], p. 103.

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

Example 4.2 (Fitting log(mpg) instead)**.** A common remedy for a funnel is to model a concave function of the response, such as $\log y$, which shrinks large responses more than small ones [1, Sec. 3.3.3], p. 103. Refit the quadratic model with log(mpg) as the response, and repeat Exercise 4.4:

4.2.3 Python

```
log_y = np.log(y)
beta_log = np.linalg.lstsq(Phi_quadratic, log_y, rcond=None)[0]
fitted_log = Phi_quadratic @ beta_log
spread_log = spread_by_third(fitted_log, log_y - fitted_log)
print(spread_log)
```

```
[0.15802602 0.18322063 0.18559353]
```

```
print(spread_quadratic[2] / spread_quadratic[0], spread_log[2] / spread_log[0])
```

```
1.878027678194057 1.1744492221667266
```

```
# Bootstrap: resample cars, refit the quadratic model to each response,
# and recompute the ratio of the highest third's spread to the lowest third's.
rng_log = np.random.default_rng(571)
ratio_boot = np.empty((2000, 2))
for b in range(2000):
    rows = rng_log.integers(0, n, n)
    for j, response in enumerate([y, log_y]):
        beta_b = np.linalg.lstsq(Phi_quadratic[rows], response[rows], rcond=None)[0]
        fitted_b = Phi_quadratic[rows] @ beta_b
        spread_b = spread_by_third(fitted_b, response[rows] - fitted_b)
        ratio_boot[b, j] = spread_b[2] / spread_b[0]
# Columns: original scale, log scale; rows: 2.5th and 97.5th percentiles.
ratio_ci = np.quantile(ratio_boot, [0.025, 0.975], axis=0)
print(ratio_ci)
```

```
[[1.42727074 0.93581492]
 [2.46554721 1.4846279 ]]
```

4.2.4 R

```
fit_log <- lm(log(mpg) ~ horsepower + I(horsepower^2), data = auto)
spread_by_third(fit_log)
```

```
[2.53,2.99] (2.99,3.24]  (3.24,3.7]
  0.1580260   0.1832206   0.1855935
```

```r
# Bootstrap: resample cars, refit the quadratic model to each response,
# and recompute the ratio of the highest third's spread to the lowest third's.
set.seed(571)
ratio_boot <- replicate(2000, {
  boot_cars <- auto[sample(nrow(auto), replace = TRUE), ]
  spread_orig <- spread_by_third(
    lm(mpg ~ horsepower + I(horsepower^2), data = boot_cars))
  spread_log <- spread_by_third(
    lm(log(mpg) ~ horsepower + I(horsepower^2), data = boot_cars))
  c(original = spread_orig[[3]] / spread_orig[[1]],
    log = spread_log[[3]] / spread_log[[1]])
})
# Rows: 2.5th and 97.5th percentiles; columns: original scale, log scale.
apply(ratio_boot, 1, quantile, c(0.025, 0.975))
```

```
      original      log
2.5%  1.418753 0.931214
97.5% 2.458572 1.467118
```

On the original scale, the highest third's residual spread was 1.88 times the lowest third's, with a 95% bootstrap percentile interval from 1.43 to 2.47. On the log scale, the ratio is 1.17, with an interval from 0.94 to 1.48. Equal spread in the two thirds is a ratio of 1: the original-scale interval excludes it, and the log-scale interval includes it. Each interval comes from 2000 resamples of the cars, refitting the model each time, and describes which cars were drawn. Another remedy, when the variance of each response is known up to a constant, is weighted least squares, which gives each squared residual a weight inversely proportional to that response's variance [1, Sec. 3.3.3], p. 103.

4.3 Correlated errors

Exercise 4.7 (Plot the residuals in time order)**.** The rows of the Auto data are sorted by model `year`. For the quadratic fit, compute the correlation between each residual and the next one in the file, and the average residual for each model year. Theorem 3.1 assumes the noise terms are uncorrelated. Do the residuals look uncorrelated?

Solution

Solution 4.7.

4.3.1 Python

```python
def lag_correlation(residuals):
    """Correlation between each residual and the next one in the file."""
    return np.corrcoef(residuals[:-1], residuals[1:])[0, 1]

residuals_quadratic = y - fitted_quadratic
lag_quadratic = lag_correlation(residuals_quadratic)
year_means = pd.Series(residuals_quadratic).groupby(year).mean()
year_means_min, year_means_max = year_means.min(), year_means.max()
last_negative_year = int(year_means.index[year_means < 0].max())
print(lag_quadratic)
```

```
0.4602676830246073
```

```python
print(year_means)
```

```
70.0   -0.616227
71.0   -1.750040
```

```
72.0   -1.938788
73.0   -2.506749
74.0   -2.170697
75.0   -2.926681
76.0   -2.024674
77.0    0.326999
78.0    0.439857
79.0    1.808373
80.0    5.245947
81.0    2.593976
82.0    4.701654
dtype: float64
```

```
# Each year's average residual, with a 95% t interval from the cars of that year
residuals_by_year = pd.Series(residuals_quadratic).groupby(year)
year_table = pd.DataFrame({"mean": residuals_by_year.mean(),
                           "sem": residuals_by_year.sem(),
                           "df": residuals_by_year.count() - 1})
year_table["half"] = stats.t.ppf(0.975, year_table["df"]) * year_table["sem"]
year_table["lower"] = year_table["mean"] - year_table["half"]
year_table["upper"] = year_table["mean"] + year_table["half"]
year_table["year"] = year_table.index.astype(int)
print(year_table[["year", "mean", "lower", "upper"]].to_string(index=False))
```

```
 year      mean     lower     upper
   70 -0.616227 -1.903516  0.671061
   71 -1.750040 -3.100018 -0.400061
   72 -1.938788 -3.191400 -0.686177
   73 -2.506749 -3.560732 -1.452767
   74 -2.170697 -3.193823 -1.147571
   75 -2.926681 -4.515174 -1.338188
   76 -2.024674 -3.075580 -0.973768
   77  0.326999 -0.683219  1.337218
   78  0.439857 -0.710148  1.589862
   79  1.808373  0.316701  3.300046
   80  5.245947  3.100536  7.391358
   81  2.593976  1.185572  4.002381
   82  4.701654  3.459409  5.943899
```

```
beta_with_year, Phi_with_year = fit([horsepower, horsepower ** 2, year])
lag_with_year = lag_correlation(y - Phi_with_year @ beta_with_year)
print(lag_with_year)
```

```
0.3075231574213514
```

```python
# Neighboring residuals are correlated, so resampling single cars would break the order.
# Instead resample blocks of 20 consecutive cars, refit the model, and recompute the correlation.
def block_bootstrap_lag_ci(Phi, block=20, B=20000, seed=571):
    """95% percentile interval for the neighboring-residual correlation from resampled blocks of cars."""
    rng = np.random.default_rng(seed)
    n_blocks = int(np.ceil(n / block))
    lag = np.empty(B)
    for b in range(B):
        starts = rng.integers(0, n - block + 1, n_blocks)
        rows = (starts[:, None] + np.arange(block)).ravel()[:n]
        beta_b = np.linalg.lstsq(Phi[rows], y[rows], rcond=None)[0]
        lag[b] = lag_correlation(y[rows] - Phi[rows] @ beta_b)
    return np.quantile(lag, [0.025, 0.975])

lag_ci_quadratic = block_bootstrap_lag_ci(Phi_quadratic)
lag_ci_with_year = block_bootstrap_lag_ci(Phi_with_year)
print(lag_ci_quadratic, lag_ci_with_year)
```

```
[0.31733878 0.50875017] [0.1433561  0.38124989]
```

4.3.2 R

```r
lag_correlation <- function(e) cor(head(e, -1), tail(e, -1))
lag_correlation(residuals(fit_quadratic))
```

```
[1] 0.4602677
```

```r
tapply(residuals(fit_quadratic), auto$year, mean)
```

```
        70         71         72         73         74         75         76
-0.6162271 -1.7500396 -1.9387883 -2.5067494 -2.1706968 -2.9266811 -2.0246741
        77         78         79         80         81         82
 0.3269993  0.4398570  1.8083732  5.2459472  2.5939761  4.7016539
```

```r
# Each year's average residual,
# with a 95% t interval from the cars of that year
sapply(split(residuals(fit_quadratic), auto$year), function(e) {
  half <- qt(0.975, length(e) - 1) * sd(e) / sqrt(length(e))
  c(mean = mean(e), lower = mean(e) - half, upper = mean(e) + half)
}) |>
  t()
```

```
         mean      lower      upper
70 -0.6162271 -1.9035155  0.6710612
71 -1.7500396 -3.1000185 -0.4000607
72 -1.9387883 -3.1914000 -0.6861766
73 -2.5067494 -3.5607316 -1.4527671
74 -2.1706968 -3.1938231 -1.1475706
75 -2.9266811 -4.5151741 -1.3381881
76 -2.0246741 -3.0755802 -0.9737680
77  0.3269993 -0.6832193  1.3372179
78  0.4398570 -0.7101477  1.5898617
79  1.8083732  0.3167006  3.3000458
80  5.2459472  3.1005363  7.3913580
81  2.5939761  1.1855717  4.0023805
82  4.7016539  3.4594092  5.9438985
```

```r
fit_with_year <- lm(mpg ~ horsepower + I(horsepower^2) + year, data = auto)
lag_correlation(residuals(fit_with_year))
```

```
[1] 0.3075232
```

```
# Neighboring residuals are correlated,
# so resampling single cars would break the order.
# Instead resample blocks of 20 consecutive cars,
# refit the model, and recompute the correlation.
block_bootstrap_lag_ci <- function(model, block = 20, n_boot = 20000) {
  set.seed(571)
  design <- model.matrix(model)
  y <- model.response(model.frame(model))
  n_blocks <- ceiling(nrow(design) / block)
  lag <- vapply(seq_len(n_boot), \(b) {
    starts <- sample(nrow(design) - block + 1, n_blocks, replace = TRUE)
    rows <- head(as.vector(outer(0:(block - 1), starts, "+")), nrow(design))
    lag_correlation(lm.fit(design[rows, ], y[rows])$residuals)
  }, numeric(1))
  unname(quantile(lag, c(0.025, 0.975)))
}
block_bootstrap_lag_ci(fit_quadratic)
```

```
[1] 0.3175578 0.5078073
```

```
block_bootstrap_lag_ci(fit_with_year)
```

```
[1] 0.1445910 0.3820917
```

```
year_table_py <- py$year_table
year_min <- year_table_py[which.min(year_table_py$mean), ]
year_max <- year_table_py[which.max(year_table_py$mean), ]
```

No. Neighboring residuals have correlation 0.46 (95% bootstrap interval 0.32 to 0.51). The interval comes from 20,000 resamples of blocks of 20 consecutive cars, which keep the order of the file, with the quadratic refitted and the correlation recomputed on each resample. The average[a] residual is negative for every year up to 1976 and positive for every later year (Figure 7), ranging from -2.9 (1975, 95% interval -4.5 to -1.3) to 5.2 (1980, 3.1 to 7.4): the model over-predicts the older cars and under-predicts the newer ones. Each year's interval is a 95% t interval for the average residual of the cars of that year, and it excludes 0 in 10 of the 13 years. James et al. [1, Sec. 3.3.3], p. 102 calls runs of same-signed neighboring residuals *tracking*. With correlated errors, the estimated standard errors of Definition 3.2 are too small, and the confidence intervals too narrow [1, Sec. 3.3.3], p. 101.

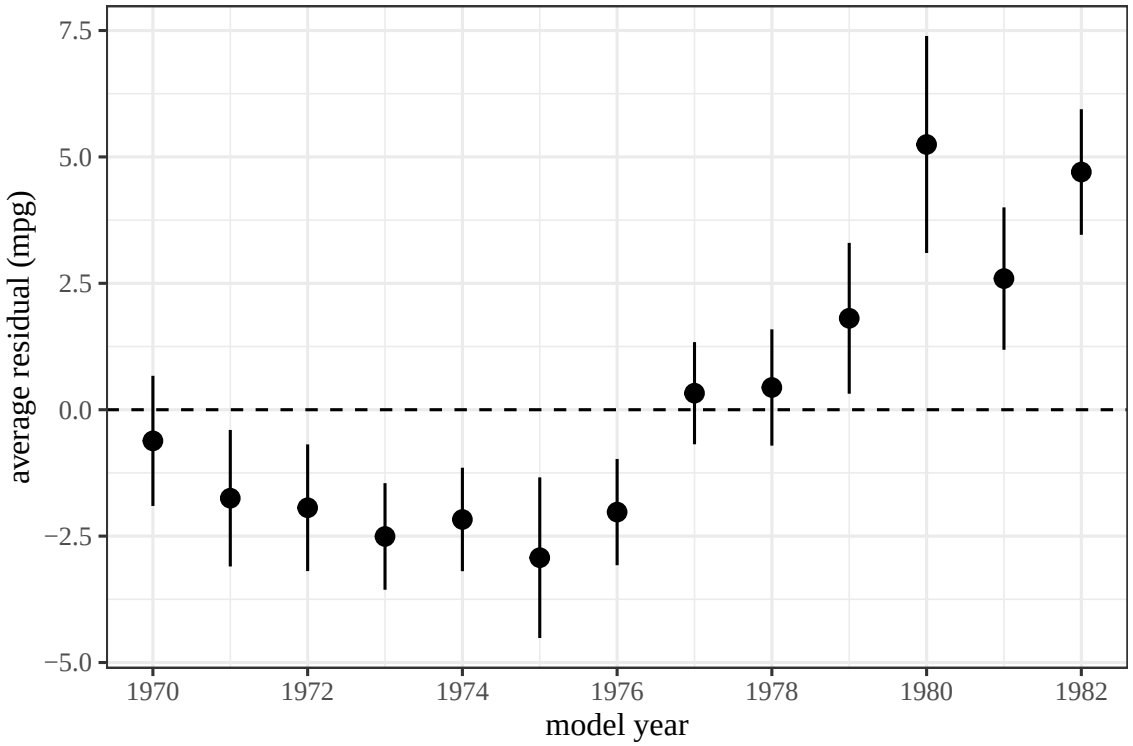


Figure 7: The average[b] residual of the quadratic fit for each year, with each year's 95% t interval.

Part of the cause is a missing feature. Adding `year` to the model lowers the neighboring correlation to 0.31 (0.14 to 0.38), but does not remove it.

[a] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-mean
[b] https://morrison-lab.github.io/mds/linear-algebra-vectors.html#def-mean

4.4 Outliers

Definition 4.2 (Standardized residual)**.** The **standardized residual** of observation i is its residual divided by its estimated standard deviation,

$$r_i = \frac{e_i}{\hat{\sigma}\sqrt{1 - h_i}},$$

where h_i is the leverage and $\hat{\sigma}$ is the residual standard error. Dividing by $\sqrt{1 - h_i}$ accounts for high-leverage points having smaller residuals, because they pull the fit toward themselves.

Example 4.3 (Standardized residuals of the quadratic fit)**.**
Compute the standardized residuals of the quadratic fit, and look at the car with the highest leverage:

4.4.1 Python

```
PtP_inv_quadratic = np.linalg.inv(Phi_quadratic.T @ Phi_quadratic)
h_quadratic = np.sum(Phi_quadratic @ PtP_inv_quadratic * Phi_quadratic, axis=1)
sigma_quadratic = np.sqrt(np.sum(residuals_quadratic ** 2) / (n - 3))
r_quadratic = residuals_quadratic / (sigma_quadratic * np.sqrt(1 - h_quadratic))
top_leverage = np.argmax(h_quadratic)
print(Auto.loc[top_leverage, ["name", "horsepower", "mpg"]])
```

```
name          pontiac grand prix
horsepower                   230
mpg                         16.0
Name: 115, dtype: object
```

```
print(h_quadratic[top_leverage], residuals_quadratic[top_leverage],
      r_quadratic[top_leverage])
```

```
0.09359709306399572 1.2281554548362656 0.2949319391422815
```

4.4.2 R

```
r_quadratic <- rstandard(fit_quadratic)
top_leverage <- which.max(hatvalues(fit_quadratic))
auto |>
  dplyr::mutate(id = dplyr::row_number()) |>
  dplyr::filter(id == top_leverage) |>
  dplyr::select(id, name, horsepower, mpg)
```

```
# A tibble: 1 x 4
     id name                horsepower   mpg
  <int> <chr>                    <dbl> <dbl>
1   116 pontiac grand prix         230    16
```

```
c(leverage = hatvalues(fit_quadratic)[[top_leverage]],
  residual = residuals(fit_quadratic)[[top_leverage]],
  standardized = r_quadratic[[top_leverage]])
```

```
    leverage     residual standardized
  0.09359709   1.22815545   0.29493194
```

The car with the highest leverage has $h_i = 0.094$, residual $e_i = 1.23$ and standardized residual $r_i = 0.29$. R's `rstandard()` gives the same r_i for every car: the largest difference from the Python values is 4×10^{-13}, which is rounding error.

> **i** Other names for standardized residuals
>
> James et al. [1, Sec. 3.3.3], p. 104 calls r_i a *studentized* residual, and R computes it with `rstandard()`. Other sources keep *studentized* for the version that estimates σ without observation i, which R computes with `rstudent()`. These notes say *standardized*.

Definition 4.3 (Outlier)**.** An **outlier** is an observation whose response is far from what the model predicts. A common rule of thumb flags observation i as a possible outlier when its standardized residual (Definition 4.2) has $|r_i| > 3$ [1, Sec. 3.3.3], p. 104.

Example 4.4 (Outliers in the quadratic fit)**.** List the cars whose standardized residuals in Example 4.3 have $|r_i| > 3$:

4.4.3 Python

```
outliers = np.abs(r_quadratic) > 3
print(Auto.loc[outliers, ["name", "horsepower", "mpg"]].assign(r=r_quadratic[outliers]))
```

```
                    name  horsepower   mpg         r
152      mercury monarch          72  15.0 -3.372439
153        ford maverick          72  15.0 -3.372439
320            mazda glc          65  46.6  3.397282
327  honda civic 1500 gl          67  44.6  3.076382
330        datsun 280-zx         132  32.7  3.646286
```

4.4.4 R

```
auto |>
  dplyr::mutate(id = dplyr::row_number(), r = r_quadratic) |>
  dplyr::filter(abs(r) > 3) |>
  dplyr::select(id, name, horsepower, mpg, r)
```

```
# A tibble: 5 x 5
     id name                horsepower   mpg     r
  <int> <chr>                    <dbl> <dbl> <dbl>
1   153 mercury monarch             72  15   -3.37
2   154 ford maverick               72  15   -3.37
3   321 mazda glc                   65  46.6  3.40
4   328 honda civic 1500 gl         67  44.6  3.08
5   331 datsun 280-zx              132  32.7  3.65
```

5 of the 392 cars have $|r_i| > 3$, and the largest is $|r_i| = 3.65$. Of these, 3 get more miles per gallon than the model predicts, and 2 get fewer. Under Gaussian noise with constant variance, about 1.1 of 392 cars would have $|r_i| > 3$ by chance, so there are more than chance alone would give. Before removing an outlier, check whether it is a recording error; if it is not, it may be a sign that the model is missing something, such as a feature [1, Sec. 3.3.3], p. 104.

4.5 High-leverage points

Example 4.5 (The highest-leverage car)**.** The leverage h_i measures how unusual car i's features are. Its average is p^*/n, and James et al. [1, Sec. 3.3.3], p. 106 flags points whose leverage greatly exceeds that average. Find the highest-leverage car in the one-feature fit:

4.5.1 Python

```
h_simple = np.sum(Phi_simple @ PtP_inv * Phi_simple, axis=1)
top = np.argmax(h_simple)
print(Auto.loc[top, ["name", "horsepower"]], h_simple[top], 2 / n)
```

```
name          pontiac grand prix
horsepower                   230
Name: 115, dtype: object 0.02975300179092146 0.00510204081632653
```

4.5.2 R

```r
h_simple <- hatvalues(fit_simple)
auto |>
  dplyr::mutate(id = dplyr::row_number()) |>
  dplyr::filter(id == which.max(h_simple)) |>
  dplyr::select(id, name, horsepower)
```

```
# A tibble: 1 x 3
     id name                horsepower
  <int> <chr>                    <dbl>
1   116 pontiac grand prix         230
```

```r
c(max = max(h_simple), average = mean(h_simple))
```

```
        max     average
0.029753002 0.005102041
```

The pontiac grand prix, with 230 horsepower, has leverage 0.0298, about 5.8 times the average 2/392. Its horsepower is the largest in the data, far from the sample mean of 104. A high-leverage point is worth checking: since $\hat{y} = Hy$, a change in its response moves its own fitted value by h_i times that change, more than the same change would for any other car.

4.6 Collinearity

Definition 4.4 (Variance inflation factor)**.** Let $\mathbf{D}$ have an intercept[a] column[b] and p features $x_1, \ldots, x_p$, and let R_j^2 be the coefficient of determination of the regression of feature x_j on the other $p-1$ features (with an intercept). The **variance inflation factor** (VIF) [5] of feature x_j is

$$\mathrm{VIF}_j \overset{\text{def}}{=} \frac{1}{1 - R_j^2}.$$

It is at least 1, and equals 1 exactly when the other features cannot predict x_j at all, in the sample ($R_j^2 = 0$). It grows without bound as x_j approaches a linear combination of the other features ($R_j^2 \to 1$).

[a] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-intercept
[b] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix-column

Example 4.6 (Collinearity of horsepower and weight)**.**
Definition 4.4 defines the variance inflation factor, $\mathrm{VIF}_j = 1/(1-R_j^2)$. Compute it for `horsepower` and `weight` in the two-feature fit:

4.6.1 Python

```python
r_hp_weight = np.corrcoef(horsepower, weight)[0, 1]
vif_hp_weight = 1 / (1 - r_hp_weight ** 2)
print(vif_hp_weight)
```

```
3.959227866518292
```

```python
# Bootstrap: resample cars with replacement and recompute the VIF each time.
rng_vif = np.random.default_rng(571)
vif_boot = np.empty(2000)
for b in range(2000):
    rows = rng_vif.integers(0, len(horsepower), len(horsepower))
    r_boot = np.corrcoef(horsepower[rows], weight[rows])[0, 1]
    vif_boot[b] = 1 / (1 - r_boot ** 2)
# The 2.5th and 97.5th percentiles of the resampled VIFs.
vif_ci = np.quantile(vif_boot, [0.025, 0.975])
print(vif_ci)
```

```
[3.28696834 4.78555867]
```

4.6.2 R

```r
r2_hp <- summary(lm(horsepower ~ weight, data = auto))$r.squared
1 / (1 - r2_hp)
```

```
[1] 3.959228
```

```r
# Bootstrap: resample cars with replacement and recompute the VIF each time.
set.seed(571)
vif_boot <- replicate(2000, {
  rows <- sample(nrow(auto), replace = TRUE)
  1 / (1 - cor(auto$horsepower[rows], auto$weight[rows])^2)
})
# The 2.5th and 97.5th percentiles of the resampled VIFs.
quantile(vif_boot, c(0.025, 0.975))
```

```
    2.5%    97.5%
3.267544 4.794859
```

With only two features, R_j^2 is the squared correlation, so both features have the same VIF, 3.96. The variance of each coefficient is 3.96 times what it would be if the two features were uncorrelated. A 95% bootstrap percentile interval for the VIF, from 2000 resamples of the cars, runs from 3.29 to 4.79. The bootstrap fits here because the VIF is a nonlinear function of the correlation, and the interval describes which cars were drawn. The interval lies entirely below the common warning level of 5 to 10 [1, Sec. 3.3.3], p. 108, and the VIF explains why the `horsepower` standard error grows from 0.0064 in the one-feature fit to 0.011 in Example 3.7.

Remark (Variance inflation from collinear features)

Remark 4.1 (Variance inflation from collinear features). In a model with an intercept, the variance of the least squares estimate of the coefficient of feature x_j is

$$\operatorname{Var}\left(\hat{\beta}_j\right) = \frac{\sigma^2}{(n-1)s_{x_j}^2}\,\mathrm{VIF}_j,$$

where σ^2 is the error variance and $s_{x_j}^2$ is the sample variance of x_j. The formula follows from the covariance of the least squares estimate (Theorem 3.1), and James et al. [1, Sec. 3.3.3] defines the VIF. The factor $\sigma^2/((n-1)s_{x_j}^2)$ is the variance of the slope in a simple linear regression on x_j alone, so VIF_j (Definition 4.4) is the factor by which the other features inflate the variance of $\hat{\beta}_j$. Features that are nearly collinear, such as the powers of one feature in the remark on stability of expansions, have large VIFs and imprecise coefficients.

5 Choosing features

Adapted from [1, Sec. 3.2.1].

The features that a model keeps change the meaning of each of its coefficients. The chapter Subset Selection covers ways to choose the features, such as forward selection.

5.1 A coefficient depends on the other features

Exercise 5.1 (Does faster acceleration raise fuel economy?)**.** In the Auto data, `acceleration` is the time in seconds a car takes to go from 0 to 60 miles per hour, so a larger value means a slower car. Fit `mpg` on `acceleration` alone.

Solution

Solution 5.1.

5.1.1 Python

```
acceleration = Auto["acceleration"].to_numpy(float)
beta_acc, Phi_acc = fit([acceleration])
# Each coefficient with its 95% confidence interval
print(np.column_stack([beta_acc, coef_ci(beta_acc, Phi_acc)]))
```

```
[[4.8332498  0.80576513 8.86073448]
 [1.19762419 0.94245663 1.45279175]]
```

5.1.2 R

```
# The table includes each coefficient's 95% confidence interval
lm(mpg ~ acceleration, data = auto) |>
  parameters::parameters() |>
  parameters::print_md(digits = 3)
```

Parameter	Coefficient	SE	95% CI	t(390)	p
(Intercept)	4.833	2.048	(0.806, 8.861)	2.359	0.019
acceleration	1.198	0.130	(0.942, 1.453)	9.228	< .001

```
coef_ci_acc <- py$coef_ci(py$beta_acc, py$Phi_acc)
```

Exercise 5.2 (Does faster acceleration raise fuel economy? (part 2))**.** Use the setup of Exercise 5.1. What is the sign of the slope[a]?

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

Solution

Solution 5.2. On its own, `acceleration` has slope[a] 1.198, so slower cars get more miles per gallon.

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

Exercise 5.3 (Does faster acceleration raise fuel economy? (part 3))**.** Use the setup of Exercise 5.1. Is it significant?

Solution

Solution 5.3. The 95% confidence interval[a] for the slope[b] is 0.942 to 1.453, and $p = 1.8 \times 10^{-18}$, so the slope is significant at the 5% level.

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

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

Exercise 5.4 (Does faster acceleration raise fuel economy? (part 4))**.** Use the setup of Exercise 5.1. Add `horsepower` to the model. What happens to the `acceleration` coefficient?

Solution

Solution 5.4.

5.1.3 Python

```
beta_acc_hp, Phi_acc_hp = fit([acceleration, horsepower])
# Each coefficient with its 95% confidence interval
print(np.column_stack([beta_acc_hp, coef_ci(beta_acc_hp, Phi_acc_hp)]))
```

```
[[52.55932488 47.47298392 57.64566585]
 [-0.60977221 -0.8464122  -0.37313222]
 [-0.18796643 -0.20492766 -0.17100519]]
```

5.1.4 R

```
# The table includes each coefficient's 95% confidence interval
lm(mpg ~ acceleration + horsepower, data = auto) |>
  parameters::parameters() |>
  parameters::print_md(digits = 3)
```

Parameter	Coefficient	SE	95% CI	t(389)	p
(Intercept)	52.559	2.587	(47.473, 57.646)	20.316	< .001
acceleration	-0.610	0.120	(-0.846, -0.373)	-5.066	< .001
horsepower	-0.188	0.009	(-0.205, -0.171)	-21.788	< .001

```
coef_ci_acc_hp <- py$coef_ci(py$beta_acc_hp, py$Phi_acc_hp)
```

With `horsepower` in the model, the `acceleration` coefficient is -0.610 (-0.846 to -0.373). It has changed sign. Each coefficient's interval is its t interval, the estimate[a] plus or minus a t quantile times its standard error[b].

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

Exercise 5.5 (Does faster acceleration raise fuel economy? (part 5))**.** Use the setup of Exercise 5.1. Explain the change.

Solution

Solution 5.5.

5.1.5 Python

```
print(np.corrcoef(acceleration, horsepower)[0, 1])
```

```
-0.6891955103342359
```

```
# bootstrap_corr_ci() is one of the interval helpers
corr_ci_acc_hp = bootstrap_corr_ci(acceleration, horsepower)
print(corr_ci_acc_hp)
```

```
[-0.73918546 -0.63352643]
```

5.1.6 R

```
cor(auto$acceleration, auto$horsepower)
```

```
[1] -0.6891955
```

```
# boot_cor_ci() is one of the interval helpers
boot_cor_ci(auto$acceleration, auto$horsepower)
```

```
[1] -0.7378244 -0.6332126
```

The two features have correlation[a] -0.69 (95% bootstrap interval -0.74 to -0.63, from 20,000 resamples of the cars): cars with small engines are slow. In the one-feature fit, `acceleration` stands in for engine power, and gets the credit for the association between engine power and `mpg`. Holding `horsepower` fixed, a slower car gets fewer miles per gallon, not more. James et al. [1, Sec. 3.2.1], p. 83 calls such a feature a *surrogate*. Their example is a regression of shark attacks on ice cream sales at a beach: the slope[b] is positive, because hot weather brings both, and ice cream sales are no longer significant once temperature is in the model. As in the two-feature fit, a feature's coefficient depends on which other features are in the model.

[a] https://morrison-lab.github.io/pds/variance-covariance.html#def-correlation
[b] https://morrison-lab.github.io/mds/algebra-exponentials.html#def-slope

[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[11].

[2] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning with applications in R*, 2nd ed. Springer, 2021. Available: https://www.statlearning.com/

[3] Wikipedia contributors, "Interaction (statistics) — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Interaction_(statistics)

[4] Wikipedia contributors, "Prediction interval — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Prediction_interval

[5] Wikipedia contributors, "Variance inflation factor — Wikipedia, the free encyclopedia." 2026. Available: https://en.wikipedia.org/wiki/Variance_inflation_factor

[11] https://doi.org/10.1007/978-3-031-38747-0