432 Class 04 Slides

thomaselove.github.io/432

2021-02-11

Today's Agenda

- Data Load from .Rds built in Class 3
- Fitting models with `lm`
    - Incorporating an interaction between factors
    - Incorporating polynomial terms
    - Incorporating restricted cubic splines
- Evaluating results in a testing sample with `yardstick`

Setup

```
knitr::opts_chunk$set(comment = NA)
options(width = 60)

library(here); library(magrittr)
library(janitor); library(knitr)
library(patchwork); library(broom)
library(rsample); library(yardstick)

library(rms)              ## new today: from Frank Harrell

library(tidyverse)

theme_set(theme_bw())
options(dplyr.summarise.inform = FALSE)
```

From Class 3

We developed the `week2` data and performed a simple imputation for it (into `week2im`) in Class 3. Here, we'll read in those saved results, and then split into testing and training samples, as we did in Class 3.

```
week2 <- readRDS(here("data", "week2.Rds"))

week2im <- readRDS(here("data", "week2im.Rds"))

set.seed(432)
week2im_split <- initial_split(week2im, prop = 3/4)

train_w2im <- training(week2im_split)
test_w2im <- testing(week2im_split)
```

Codebook for useful `week2` variables

- 894 subjects in Cleveland-Elyria with `bmi` and no history of diabetes

Variable	Description
`bmi`	(outcome) Body-Mass index in kg/m^2.
`inc_imp`	income (imputed from grouped values) in \$
`fruit_day`	average fruit servings consumed per day
`drinks_wk`	average alcoholic drinks consumed per week
`female`	sex: $1 =$ female, $0 =$ male
`exerany`	any exercise in the past month: $1 =$ yes, $0 =$ no
`genhealth`	self-reported overall health (5 levels)
`race_eth`	race and Hispanic/Latinx ethnicity (5 levels)

- plus `ID`, `SEQNO`, `hx_diabetes` (all 0), `MMSA` (all Cleveland-Elyria)
- See Chapter 2 of the Course Notes for details on the variables

Class 03 and Class 04

In Class 03, we fit two models to predict `bmi`, using `exerany` and `health`, one with an interaction term and one without.

```r
m_1 <- lm(bmi ~ exerany + health, data = train_w2im)
m_1int <- lm(bmi ~ exerany * health, data = train_w2im)
```

In Class 04, today, we'll fit models incorporating a covariate.

The covariate we'll add is `fruit_day`, a quantity (servings/day).

- `m_2` and `m_2int` will add a linear term for `fruit_day`
- `m_3` and `m_3int` instead add a quadratic polynomial in `fruit_day`
- `m_4` and `m_4int` instead add a restricted cubic spline in `fruit_day`

Giving away the ending: We'll see that none of these augmented models will clearly improve the fit in our testing sample over the performance of `m_1` and `m_1int`.

Adding in the covariate `fruit_day` to `m_1`

```r
m_2 <- lm(bmi ~ fruit_day + exerany + health,
          data = train_w2im)
```

- How well does this model fit the training data?

mod	r.sq	adj.r.sq	sigma	df	df.res	AIC	BIC
m_1	0.082	0.075	6.00	5	665	4316.0	4347.5
m_2	0.090	0.081	5.98	6	664	4312.6	4348.7
m_1int	0.098	0.085	5.96	9	661	4312.6	4362.2

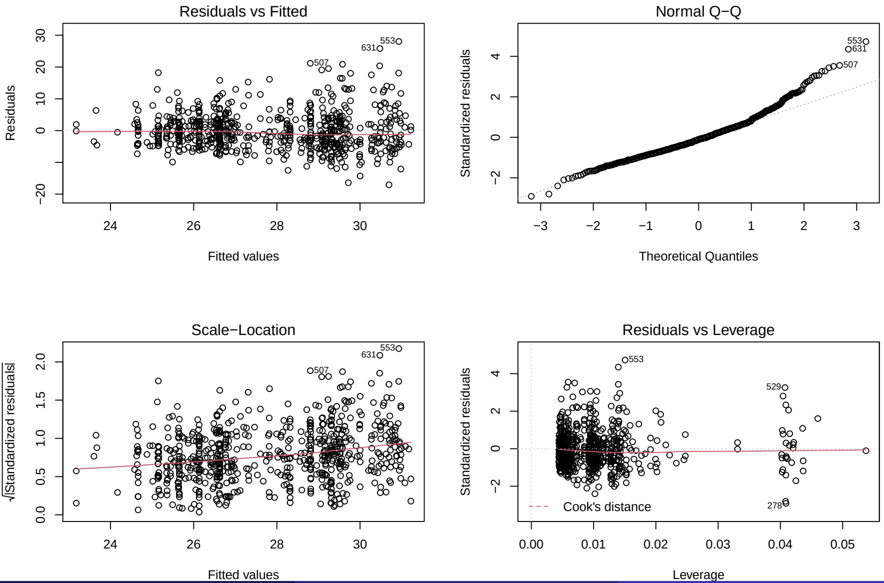
- Also available in `glance` for a model fit with `lm` are `statistic`, `p.value`, `logLik`, and `deviance`.

Tidied summary of m_2 coefficients

```
tidy(m_2, conf.int = TRUE, conf.level = 0.90) %>%
    kable(digits = c(0,2,2,2,3,2,2))
```

term	estimate	std.error	statistic	p.value	conf.low	conf.high
(Intercept)	27.34	0.81	33.93	0.000	26.01	28.67
fruit_day	-0.50	0.22	-2.30	0.022	-0.85	-0.14
exerany	-1.19	0.55	-2.15	0.032	-2.10	-0.28
healthVG	0.97	0.69	1.40	0.162	-0.17	2.11
healthG	3.65	0.72	5.09	0.000	2.47	4.84
healthF	3.64	0.88	4.16	0.000	2.20	5.08
healthP	3.92	1.32	2.96	0.003	1.74	6.09

`m_2` Residual Plots (non-constant variance?)



Who is that poorest fit case?

Plot suggests we look at row 553

```
train_w2im %>% slice(553) %>%
    select(ID, bmi, fruit_day, exerany, health) %>% kable()
```

ID	bmi	fruit_day	exerany	health
959	58.98	0.1	0	F

What is unusual about this subject?

```
train_w2im %$% sort(bmi) %>% tail()
```

```
[1] 49.04 49.98 50.46 50.85 56.31 58.98
```

What if we included the interaction term?

```r
m_2int <- lm(bmi ~ fruit_day + exerany * health,
             data = train_w2im)
```

Compare `m_2int` fit to previous models...

mod	r.sq	adj.r.sq	sigma	df	df.res	AIC	BIC
m_1	0.082	0.075	6.00	5	665	4316.0	4347.5
m_2	0.090	0.081	5.98	6	664	4312.6	4348.7
m_1int	0.098	0.085	5.96	9	661	4312.6	4362.2
m_2int	0.106	0.093	5.94	10	660	4308.2	4362.3

- `m_1` = no `fruit_day`
- `m_2` = `fruit_day` included
- `int` = `exerany*health` interaction included

ANOVA comparison of `m_2` and `m_2int`

```r
anova(m_2, m_2int)
```

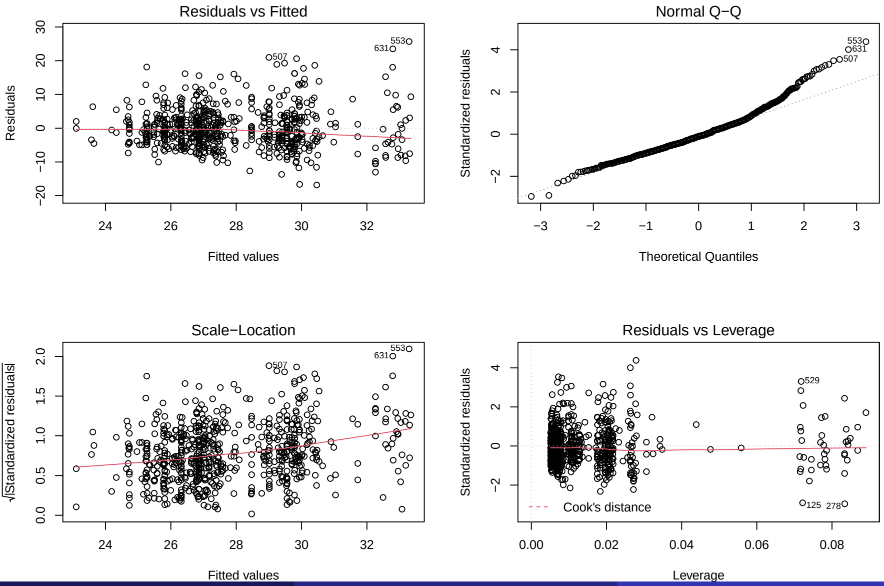
```
Analysis of Variance Table

Model 1: bmi ~ fruit_day + exerany + health
Model 2: bmi ~ fruit_day + exerany * health
  Res.Df   RSS Df Sum of Sq      F  Pr(>F)
1    664 23724
2    660 23288  4    436.03 3.0893 0.01551 *
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Tidied summary of `m_2int` coefficients

```
tidy(m_2int, conf.int = TRUE, conf.level = 0.90) %>%
    rename(se = std.error, t = statistic, p = p.value) %>%
    kable(digits = c(0,2,2,2,3,2,2))
```

term	estimate	se	t	p	conf.low	conf.high
(Intercept)	26.50	1.67	15.87	0.000	23.75	29.25
fruit_day	-0.54	0.22	-2.51	0.012	-0.90	-0.19
exerany	-0.14	1.76	-0.08	0.935	-3.04	2.76
healthVG	1.03	1.85	0.56	0.578	-2.02	4.08
healthG	3.98	1.83	2.18	0.030	0.97	6.98
healthF	6.85	1.91	3.59	0.000	3.70	9.99
healthP	4.58	2.38	1.92	0.055	0.66	8.50
exerany:healthVG	0.02	2.00	0.01	0.993	-3.27	3.31
exerany:healthG	-0.23	1.99	-0.12	0.906	-3.51	3.04
exerany:healthF	-5.07	2.17	-2.33	0.020	-8.65	-1.49
exerany:healthP	-0.61	2.92	-0.21	0.835	-5.42	4.21

Residual plots for model `m_2int`?



Which of the four models fits best?

In the training sample, we have...

mod	r.sq	adj.r.sq	sigma	df	df.res	AIC	BIC
m_1	0.082	0.075	6.00	5	665	4316.0	4347.5
m_2	0.090	0.081	5.98	6	664	4312.6	4348.7
m_1int	0.098	0.085	5.96	9	661	4312.6	4362.2
m_2int	0.106	0.093	5.94	10	660	4308.2	4362.3

- The interaction models look better by Adjusted R^2 and σ; AIC likes `m_2int` while BIC likes `m1`. What to do?
- More importantly, the testing sample cannot judge between models accurately. Our models have already *seen* that data.
- For fairer comparisons, consider the (held out) testing sample...

Model predictions of `bmi` in the testing sample

We'll use `augment` from the `broom` package...

```
m1_test_aug <- augment(m_1, newdata = test_w2im)
m1int_test_aug <- augment(m_1int, newdata = test_w2im)
m2_test_aug <- augment(m_2, newdata = test_w2im)
m2int_test_aug <- augment(m_2int, newdata = test_w2im)
```

This adds fitted values (predictions) and residuals (errors) ...

```
m1_test_aug %>% select(ID, bmi, .fitted, .resid) %>%
    slice(1:2) %>% kable()
```

ID	bmi	.fitted	.resid
11	27.17	25.29124	1.878756
15	27.09	29.06438	-1.974377

What will the `yardstick` package do?

For each subject in the testing set, we will need:

- estimate = model's prediction of that subject's `bmi`
- truth = the `bmi` value observed for that subject

Calculate a summary of the predictions across the n test subjects, such as:

- R^2 = square of the correlation between truth and estimate
- `mae` = mean absolute error ...

$$mae = \frac{1}{n} \sum |truth - estimate|$$

- `rmse` = root mean squared error ...

$$rmse = \sqrt{\frac{1}{n} \sum (truth - estimate)^2}$$

Testing Results (using R^2)

We can use the yardstick package and its rsq() function.

```
testing_r2 <- bind_rows(
    rsq(m1_test_aug, truth = bmi, estimate = .fitted),
    rsq(m1int_test_aug, truth = bmi, estimate = .fitted),
    rsq(m2_test_aug, truth = bmi, estimate = .fitted),
    rsq(m2int_test_aug, truth = bmi, estimate = .fitted)) %>%
    mutate(model = c("m_1", "m_1int", "m_2", "m_2int"))
testing_r2 %>% kable(dig = 4)
```

.metric	.estimator	.estimate	model
rsq	standard	0.0828	m_1
rsq	standard	0.0881	m_1int
rsq	standard	0.0782	m_2
rsq	standard	0.0829	m_2int

Mean Absolute Error?

Consider the mean absolute prediction error . . .

```
testing_mae <- bind_rows(
    mae(m1_test_aug, truth = bmi, estimate = .fitted),
    mae(m1int_test_aug, truth = bmi, estimate = .fitted),
    mae(m2_test_aug, truth = bmi, estimate = .fitted),
    mae(m2int_test_aug, truth = bmi, estimate = .fitted)) %>%
    mutate(model = c("m_1", "m_1int", "m_2", "m_2int"))
testing_mae %>% kable(dig = 2)
```

.metric	.estimator	.estimate	model
mae	standard	4.45	m_1
mae	standard	4.46	m_1int
mae	standard	4.41	m_2
mae	standard	4.42	m_2int

Root Mean Squared Error?

How about the square root of the mean squared prediction error, or RMSE?

```
testing_rmse <- bind_rows(
   rmse(m1_test_aug, truth = bmi, estimate = .fitted),
   rmse(m1int_test_aug, truth = bmi, estimate = .fitted),
   rmse(m2_test_aug, truth = bmi, estimate = .fitted),
   rmse(m2int_test_aug, truth = bmi, estimate = .fitted)) %>%
   mutate(model = c("m_1", "m_1int", "m_2", "m_2int"))
testing_rmse %>% kable(digits = 3)
```

.metric	.estimator	.estimate	model
rmse	standard	6.095	m_1
rmse	standard	6.079	m_1int
rmse	standard	6.110	m_2
rmse	standard	6.096	m_2int

Other Summaries for Numerical Predictions

Within the `yardstick` package, there are several other summaries, including:

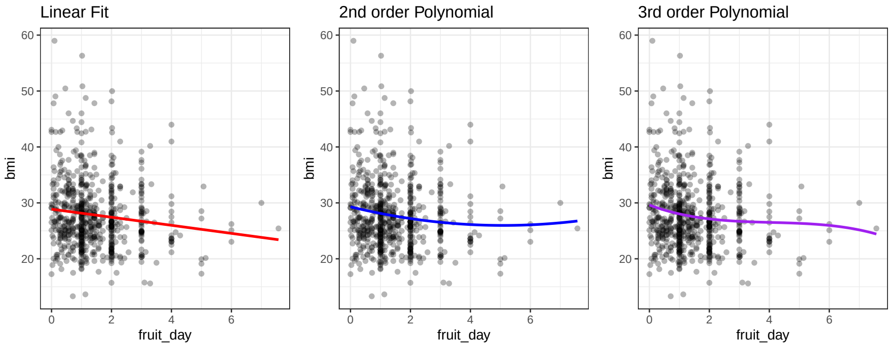
- `rsq_trad()` = defines R^2 using sums of squares.
    - The `rsq()` measure we showed a few slides ago is a squared correlation coefficient and is guaranteed to fall in (0, 1).
- `mape()` = mean absolute percentage error
- `mpe()` = mean percentage error
- `huber_loss()` = Huber loss (often used in robust regression), which is less sensitive to outliers than `rmse()`.
- `ccc()` = concordance correlation coefficient, which attempts to measure both consistency/correlation (like `rsq()`) and accuracy (like `rmse()`).

See the yardstick home page for more details.

Incorporating a non-linear term for `fruit_day`

Suppose we wanted to include a polynomial term for `fruit_day`:

```
lm(bmi ~ fruit_day, data = train_w2im)
lm(bmi ~ poly(fruit_day, 2), data = train_w2im)
lm(bmi ~ poly(fruit_day, 3), data = train_w2im)
```



Polynomial Regression

A polynomial in the variable `x` of degree D is a linear combination of the powers of `x` up to D.

For example:

- Linear: $y = \beta_0 + \beta_1 x$
- Quadratic: $y = \beta_0 + \beta_1 x + \beta_2 x^2$
- Cubic: $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$
- Quartic: $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4$
- Quintic: $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5$

Fitting such a model creates a **polynomial regression**.

Raw Polynomials vs. Orthogonal Polynomials

Predict `bmi` using `fruit_day` with a polynomial of degree 2.

```
(temp1 <- lm(bmi ~ fruit_day + I(fruit_day^2),
             data = train_w2im))
```

```
Call:
lm(formula = bmi ~ fruit_day + I(fruit_day^2), data = train_w2


Coefficients:
   (Intercept)        fruit_day  I(fruit_day^2)
       29.2991          -1.3079          0.1284
```

This uses raw polynomials. Predicted `bmi` for `fruit_day = 2` is

```
bmi = 29.2991 - 1.3079 (fruit_day) + 0.1284 (fruit_day^2)
    = 29.2991 - 1.3079 (2) + 0.1284 (4)
    = 27.1969
```

Does the raw polynomial match our expectations?

```r
temp1 <- lm(bmi ~ fruit_day + I(fruit_day^2),
            data = train_w2im)
```

```r
augment(temp1, newdata = tibble(fruit_day = 2)) %>%
    kable(digits = 4)
```

fruit_day	.fitted
2	27.1969

and this matches our "by hand" calculation. But it turns out most regression models use *orthogonal* rather than raw polynomials...

Fitting an Orthogonal Polynomial

Predict `bmi` using `fruit_day` with an *orthogonal* polynomial of degree 2.

```
(temp2 <- lm(bmi ~ poly(fruit_day,2), data = train_w2im))
```

```
Call:
lm(formula = bmi ~ poly(fruit_day, 2), data = train_w2im)

Coefficients:
        (Intercept)  poly(fruit_day, 2)1
              27.84               -20.33
poly(fruit_day, 2)2
               7.21
```

This looks very different from our previous version of the model.

- What happens when we make a prediction, though?

Prediction in the Orthogonal Polynomial Model

Remember that in our raw polynomial model, our "by hand" and "using R" calculations both concluded that the predicted `bmi` for a subject with `fruit_day` = 2 was 27.1969.

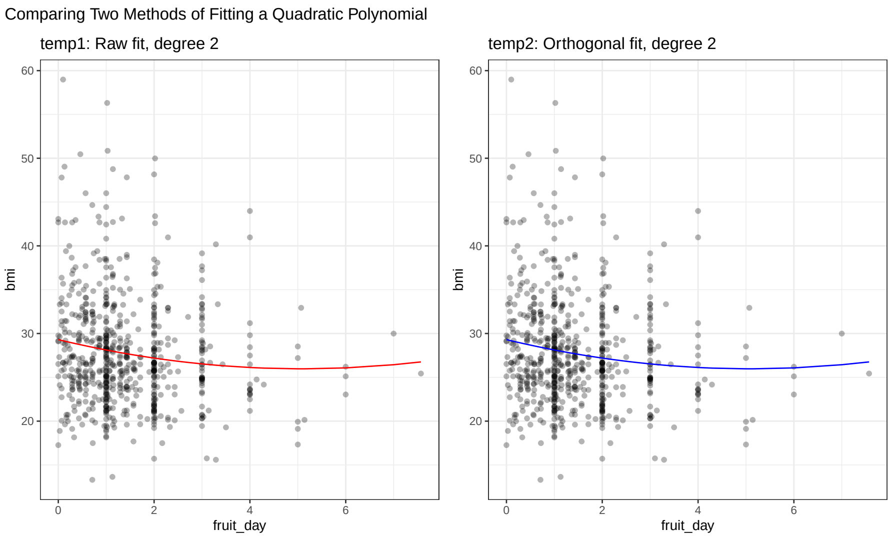
Now, what happens with the orthogonal polynomial model `temp2` we just fit?

```
augment(temp2, newdata = data.frame(fruit_day = 2)) %>%
    kable(digits = 4)
```

fruit_day	.fitted
2	27.1969

- No change in the prediction.

Fits of raw vs orthogonal polynomials



- The two models are, in fact, identical.

Why do we use orthogonal polynomials?

- The main reason is to avoid having to include powers of our predictor that are highly collinear.
- Variance Inflation Factor assesses collinearity. . .

```
vif(temp1)          ## from rms package
```

```
     fruit_day I(fruit_day^2)
      6.259193       6.259193
```

- Orthogonal polynomial terms are uncorrelated with one another, easing the process of identifying which terms add value to our model.

```
vif(temp2)
```

```
poly(fruit_day, 2)1 poly(fruit_day, 2)2
                  1                   1
```

Why orthogonal rather than raw polynomials?

The tradeoff is that the raw polynomial is a lot easier to explain in terms of a single equation in the simplest case.

Actually, we'll usually avoid polynomials in our practical work, and instead use splines, which are more flexible and require less maintenance, but at the cost of pretty much requiring you to focus on visualizing their predictions rather than their equations.

Adding a Second Order Polynomial to our Models

```
m_3 <- lm(bmi ~ poly(fruit_day,2) + exerany + health,
          data = train_w2im)
```

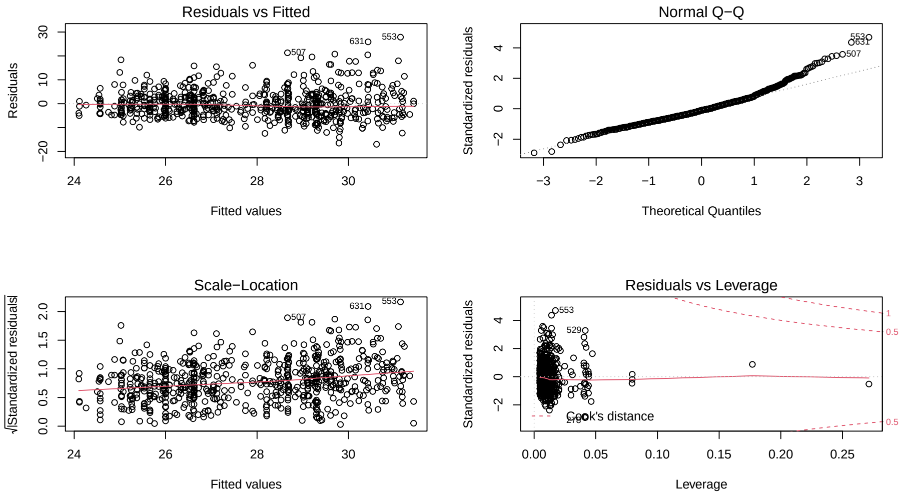
- Comparison to other models without the interaction...

mod	r.sq	adj.r.sq	sigma	df	df.res	AIC	BIC
m_1	0.0823	0.0754	6.00	5	665	4316.0	4347.5
m_2	0.0896	0.0813	5.98	6	664	4312.6	4348.7
m_3	0.0903	0.0807	5.98	7	663	4314.1	4354.7

Tidied summary of `m_3` coefficients

term	est	se	t	p	conf.low	conf.high
(Intercept)	26.58	0.75	35.35	0.000	25.35	27.82
poly(fruit_day, 2)1	-14.08	6.09	-2.31	0.021	-24.12	-4.05
poly(fruit_day, 2)2	4.41	6.06	0.73	0.467	-5.58	14.40
exerany	-1.12	0.56	-2.01	0.045	-2.04	-0.20
healthVG	0.96	0.69	1.39	0.165	-0.18	2.11
healthG	3.64	0.72	5.07	0.000	2.46	4.83
healthF	3.66	0.88	4.18	0.000	2.22	5.11
healthP	3.92	1.32	2.97	0.003	1.75	6.10

m_3 Residual Plots



Add in the interaction

```
m_3int <- lm(bmi ~ poly(fruit_day,2) + exerany * health,
             data = train_w2im)
```

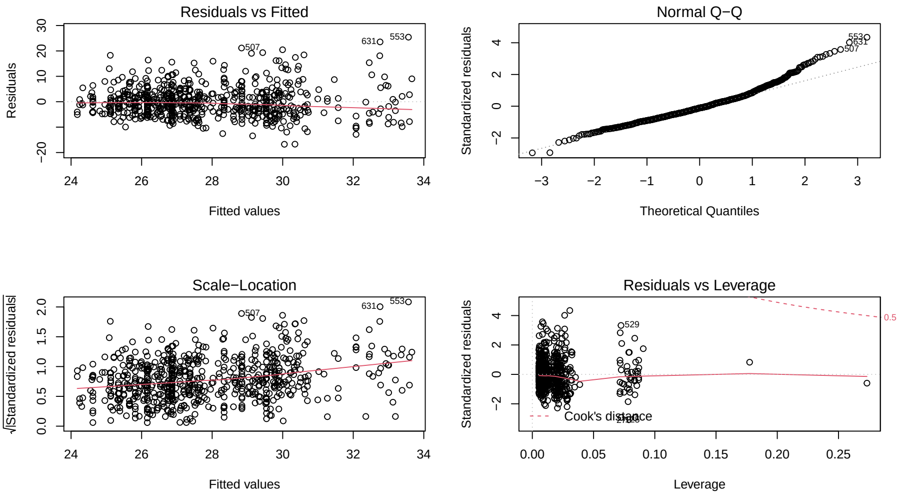
- Comparison to other models with the interaction...

mod	r.sq	adj.r.sq	sigma	df	df.res	AIC	BIC
m_1int	0.0977	0.0854	5.96	9	661	4312.6	4362.2
m_2int	0.1063	0.0928	5.94	10	660	4308.2	4362.3
m_3int	0.1074	0.0925	5.94	11	659	4309.4	4368.0

Tidied summary of `m_3int` coefficients

term	est	se	t	p	conf.low	conf.high
(Intercept)	25.64	1.65	15.53	0.000	22.92	28.36
poly(fruit_day, 2)1	-15.42	6.08	-2.54	0.011	-25.43	-5.41
poly(fruit_day, 2)2	5.34	6.03	0.89	0.376	-4.59	15.28
exerany	-0.03	1.76	-0.02	0.987	-2.94	2.88
healthVG	1.04	1.85	0.56	0.574	-2.01	4.10
healthG	3.99	1.83	2.19	0.029	0.99	7.00
healthF	6.93	1.91	3.62	0.000	3.78	10.07
healthP	4.60	2.38	1.93	0.054	0.68	8.52
exerany:healthVG	-0.01	2.00	0.00	0.997	-3.30	3.28
exerany:healthG	-0.27	1.99	-0.14	0.891	-3.55	3.00
exerany:healthF	-5.15	2.17	-2.37	0.018	-8.73	-1.57
exerany:healthP	-0.61	2.92	-0.21	0.835	-5.42	4.21

`m_3int` Residual Plots



How do models m_3 and m_3int do in testing?

```r
m3_test_aug <- augment(m_3, newdata = test_w2im)
m3int_test_aug <- augment(m_3int, newdata = test_w2im)

testing_r2 <- bind_rows(
    rsq(m1_test_aug, truth = bmi, estimate = .fitted),
    rsq(m2_test_aug, truth = bmi, estimate = .fitted),
    rsq(m3_test_aug, truth = bmi, estimate = .fitted),
    rsq(m1int_test_aug, truth = bmi, estimate = .fitted),
    rsq(m2int_test_aug, truth = bmi, estimate = .fitted),
    rsq(m3int_test_aug, truth = bmi, estimate = .fitted)) %>%
    mutate(model = c("m_1", "m_2", "m_3", "m_1int",
                     "m_2int", "m_3int"))
```

- I've hidden my calculations for RMSE and MAE here.

Results comparing all six models (testing)

```
bind_cols(testing_r2 %>% select(model, rsquare = .estimate),
          testing_rmse %>% select(rmse = .estimate),
          testing_mae %>% select(mae = .estimate)) %>%
    kable(digits = c(0, 4, 3, 3))
```

model	rsquare	rmse	mae
m_1	0.0828	6.095	4.447
m_2	0.0782	6.110	4.411
m_3	0.0764	6.116	4.430
m_1int	0.0881	6.079	4.458
m_2int	0.0829	6.096	4.425
m_3int	0.0806	6.105	4.444

- Did the polynomial term in `m_3` and `m_3int` improve our predictions?

Splines

- A **linear spline** is a continuous function formed by connecting points (called **knots** of the spline) by line segments.
- A **restricted cubic spline** is a way to build highly complicated curves into a regression equation in a fairly easily structured way.
- A restricted cubic spline is a series of polynomial functions joined together at the knots.
    - Such a spline gives us a way to flexibly account for non-linearity without over-fitting the model.
    - Restricted cubic splines can fit many different types of non-linearities.
    - Specifying the number of knots is all you need to do in R to get a reasonable result from a restricted cubic spline.

The most common choices are 3, 4, or 5 knots.

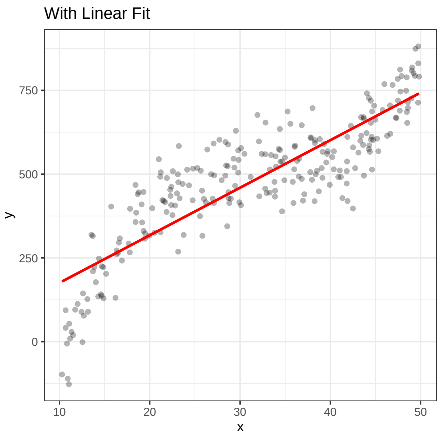
- 3 Knots, 2 degrees of freedom, allows the curve to "bend" once.
- 4 Knots, 3 degrees of freedom, lets the curve "bend" twice.
- 5 Knots, 4 degrees of freedom, lets the curve "bend" three times.

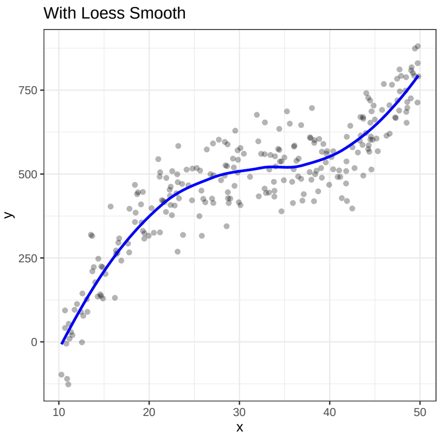
A simulated data set

```r
set.seed(4322021)

sim_data <- tibble(
    x = runif(250, min = 10, max = 50),
    y = 3*(x-30) - 0.3*(x-30)^2 + 0.05*(x-30)^3 +
        rnorm(250, mean = 500, sd = 70)
)

head(sim_data, 2)
```

```
# A tibble: 2 x 2
      x     y
  <dbl> <dbl>
1  42.5  397.
2  35.9  414.
```

The sim_data, plotted.

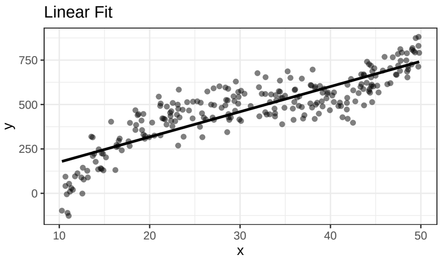


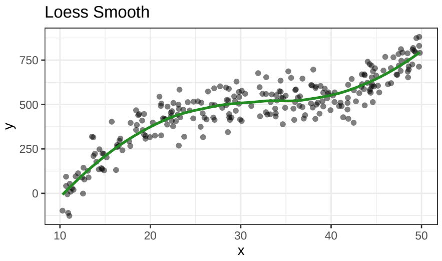


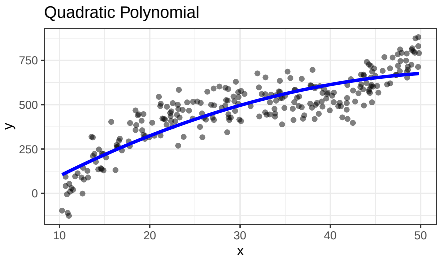
Fitting Restricted Cubic Splines with `lm` and `rcs`

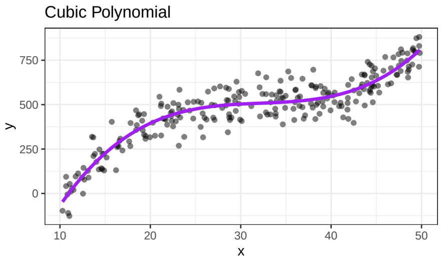
```
sim_linear <- lm(y ~ x, data = sim_data)
sim_poly2  <- lm(y ~ poly(x, 2), data = sim_data)
sim_poly3  <- lm(y ~ poly(x, 3), data = sim_data)
sim_rcs3   <- lm(y ~ rcs(x, 3), data = sim_data)
sim_rcs4   <- lm(y ~ rcs(x, 4), data = sim_data)
sim_rcs5   <- lm(y ~ rcs(x, 5), data = sim_data)
```

Looking at the Polynomial Fits

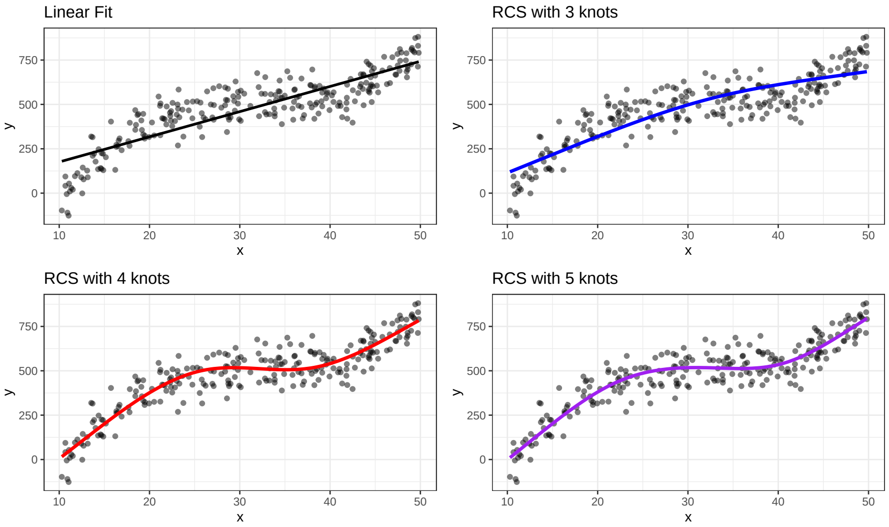








Looking at the Restricted Cubic Spline Fits



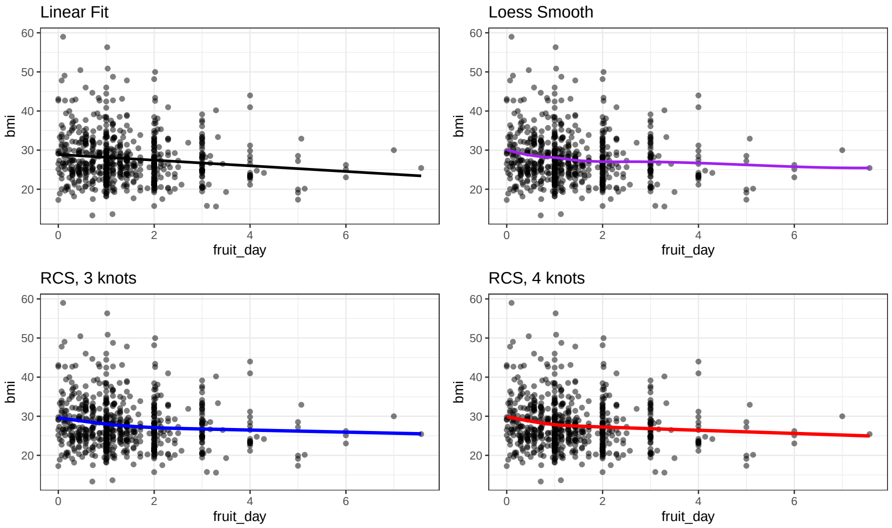
Fitting Restricted Cubic Splines with `lm` and `rcs`

For most applications, three to five knots strike a nice balance between complicating the model needlessly and fitting data pleasingly. Let's consider a restricted cubic spline model for `bmi` based on `fruit_day` again, but now with:

- in `temp3`, 3 knots, and
- in `temp4`, 4 knots,

```
temp3 <- lm(bmi ~ rcs(fruit_day, 3), data = train_w2im)
temp4 <- lm(bmi ~ rcs(fruit_day, 4), data = train_w2im)
```

Spline models for `bmi` and `fruit_day`



Let's try an RCS with 4 knots

```
m_4 <- lm(bmi ~ rcs(fruit_day, 4) + exerany + health,
          data = train_w2im)

m_4int <- lm(bmi ~ rcs(fruit_day, 4) + exerany * health,
          data = train_w2im)
```

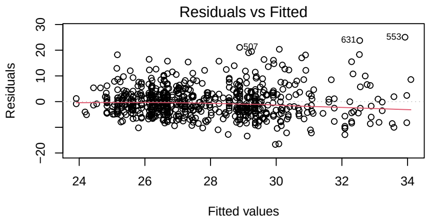
Comparing 4 models including the `exerany*health` interaction...

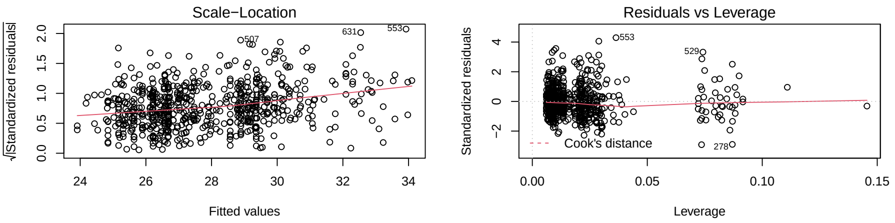
mod	fruit	r.sq	adj.r.sq	sigma	df	AIC	BIC
m_1int	not in	0.0977	0.0854	5.964	9	4312.6	4362.2
m_2int	linear	0.1063	0.0928	5.940	10	4308.2	4362.3
m_3int	poly(2)	0.1074	0.0925	5.941	11	4309.4	4368.0
m_4int	rcs(4)	0.1083	0.0920	5.942	12	4310.7	4373.8

Tidied summary of `m_4int` coefficients

term	est	se	t	p	lo90	hi90
(Intercept)	27.08	1.76	15.42	0.000	24.19	29.97
rcs(fruit_day, 4)fruit_day	-1.78	1.30	-1.37	0.170	-3.92	0.36
rcs(fruit_day, 4)fruit_day'	3.78	5.58	0.68	0.499	-5.41	12.97
rcs(fruit_day, 4)fruit_day''	-9.31	16.81	-0.55	0.580	-37.00	18.39
exerany	0.10	1.77	0.06	0.956	-2.82	3.02
healthVG	1.06	1.86	0.57	0.566	-1.99	4.12
healthG	4.04	1.83	2.21	0.027	1.03	7.05
healthF	7.02	1.91	3.67	0.000	3.87	10.17
healthP	4.65	2.38	1.95	0.051	0.73	8.57
exerany:healthVG	-0.01	2.00	-0.01	0.994	-3.31	3.28
exerany:healthG	-0.32	1.99	-0.16	0.872	-3.60	2.96
exerany:healthF	-5.23	2.18	-2.40	0.017	-8.81	-1.64
exerany:healthP	-0.66	2.92	-0.22	0.822	-5.48	4.16

`m_4int` Residual Plots





How do models `m_4` and `m_4int` do in testing?

model	rsquare	rmse	mae
m_1	0.0828	6.095	4.447
m_2	0.0782	6.110	4.411
m_3	0.0764	6.116	4.430
m_4	0.0756	6.120	4.423
m_1int	0.0881	6.079	4.458
m_2int	0.0829	6.096	4.425
m_3int	0.0806	6.105	4.444
m_4int	0.0783	6.117	4.447

I'll note that there's a fair amount of very repetitive code in the R Markdown file to create that table.

- What are our conclusions?

Next Week

- Using the `ols` modeling structure (from the `rms` package) to fit and assess linear regression models
- The Spearman ρ^2 plot, and some thoughts on how to spend data / degrees of freedom on nonlinearity
- What if we want to build models for a binary outcome, rather than a quantitative one?

This weekend, please get started finding and ingesting a Project 1 data set.