Notebook 4 - Machine Learning

May 22, 2024

Reference Guide for R (student resource) - Check out our reference guide for a full listing of useful R commands for this project.

0.1 Data Science Project: Use data to determine the best and worst colleges for conquering student debt.

0.1.1 Notebook 4: Machine Learning

Does college pay off? We'll use some of the latest data from the US Department of Education's College Scorecard Database to answer that question.

In this notebook (the 4th of 4 total notebooks), you'll use R to add polynomial terms to your multiple regression models (i.e. polynomial regression). Then, you'll use the principles of machine learning to tune models for a prediction task on *unseen* data.

[4]:
```r
## Run this code but do not edit it. Hit Ctrl+Enter to run the code.
# This command downloads a useful package of R commands
library(coursekata)
```

```
  CourseKata packages
coursekata 0.15.0
 dslabs                0.8.0        Metrics

 0.1.4
 Lock5withR            1.2.2        lsr

 0.5.2
 fivethirtyeightdata 0.1.0        mosaic

 1.9.1
 fivethirtyeight       0.6.2        supernova

 3.0.0
```

0.1.2 The Dataset (`four_year_colleges.csv`)

General description - In this notebook, we'll be using the `four_year_colleges.csv` file, which only includes schools that offer four-year bachelors degrees and/or higher graduate degrees. Community colleges and trade schools often have different goals (e.g. facilitating transfers, direct career education) than institutions that offer four-year bachelors degrees. By comparing four-year colleges only to other four-year colleges, we'll have clearer analyses and conclusions.

This data is a subset of the US Department of Education's College Scorecard Database. The data is current as of the 2020-2021 school year.

Description of all variables: See here

Detailed data file description: See here

```
[5]: ## Run this code but do not edit it. Hit Ctrl+Enter to run the code.
     # This command downloads data from the file 'colleges.csv' and stores it in an␣
      ↪object called `dat`
     dat <- read.csv('https://skewthescript.org/s/four_year_colleges.csv')
```

0.1.3 1.0 - Motivating non-linear regression

So far, we've focused entirely on **linear regression** and **multiple linear regression** models, which use linear functions to relate predictors (e.g. `net_tuition`,`grad_rate`,`pct_PELL`) to the outcome (`default_rate`).

In this notebook, we're going to investigate ways to model **non-linear** relationships. To make this task a bit more manageable at the start, let's reduce the size of our dataset by taking a random sample of 20 colleges from the `dat` dataframe. We will store our sample in a new R dataframe called `sample_dat`.

```
[6]: ## Run this code but do not edit it
     # create a dataset to train the model with 20 randomly selected observations
     set.seed(2)
     sample_dat <- sample(dat, size = 20)
```

Note: When getting a random sample, we'll get different results each time we run our code because it's … well … random. This can be quite annoying. So, in the code above, we used the command `set.seed(2)`. This ensures that each time the code is executed, we get the same results for our random sample - the results stored in seed `2`. We could have also set the seed to `1` or `3` or `845` or `12345`. The seed numbers serve merely as a unique ID that corresponds to a certain result from a random draw. By setting a certain seed, we'll always get a certain random draw.

1.1 Let's take a look at our sample data set. Print out the `head` and `dim` of `sample_dat`.

```
[7]: # Your code goes here
     head(sample_dat)
```

A data.frame: 6 × 27

	OPEID	name	city	state	region
	<int>	<chr>	<chr>	<chr>	<chr>
975	379400	Saint Martin's University	Lacey	WA	Far West
710	316100	Northeastern State University	Tahlequah	OK	Rockies & Southw
774	330400	Muhlenberg College	Allentown	PA	Northeast
416	231800	Spring Arbor University	Spring Arbor	MI	Midwest
392	223400	Adrian College	Adrian	MI	Midwest
273	189200	University of Iowa	Iowa City	IA	Midwest

[8]:
```
# Your code goes here
dim(sample_dat)
```

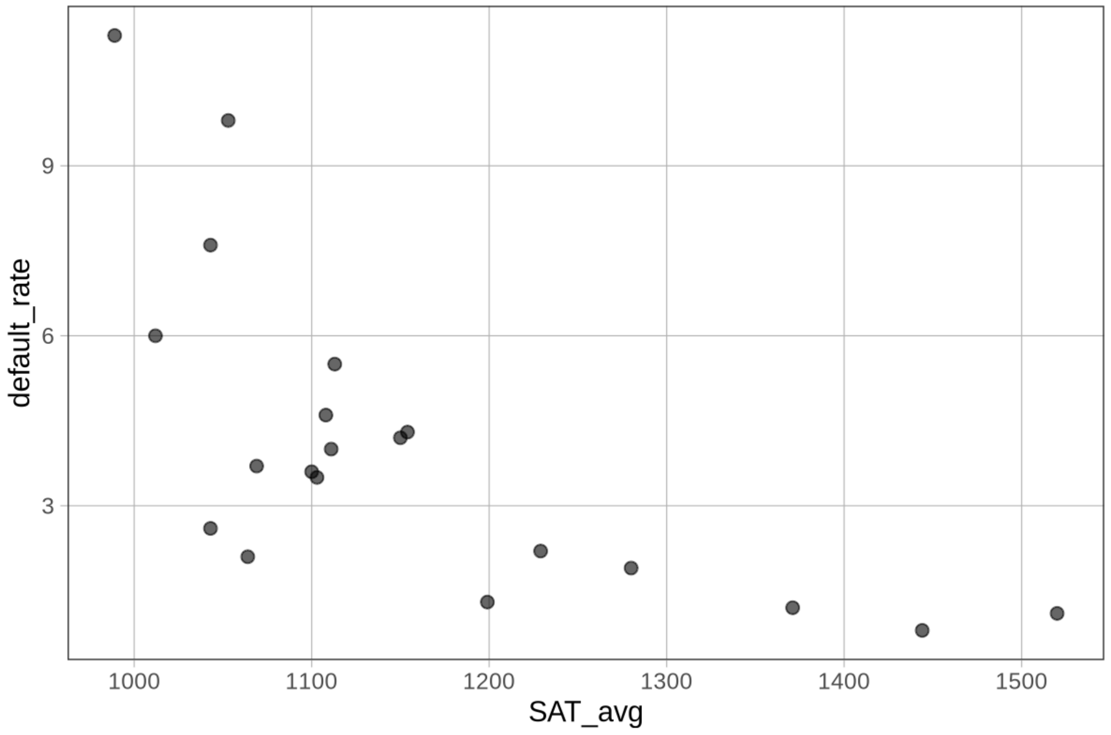
1. 20 2. 27

Check yourself: The dimensions of `sample_dat` should be 20 rows and 27 columns.

In prior notebooks, we focused on institutional and economic predictors of student loan default rates. In this notebook, we'll begin by analyzing an *academic* variable: `SAT_avg`. This variable shows the average SAT score of students who matriculate to a college.

The following code creates a scatterplot of the relationship between `SAT_avg` (predictor) and `default_rate` (outcome) from the dataset `sample_dat`:

[9]:
```
## Run this code but do not edit it
# create scatterplot: default_rate ~ SAT_avg
gf_point(default_rate ~ SAT_avg, data = sample_dat)
```



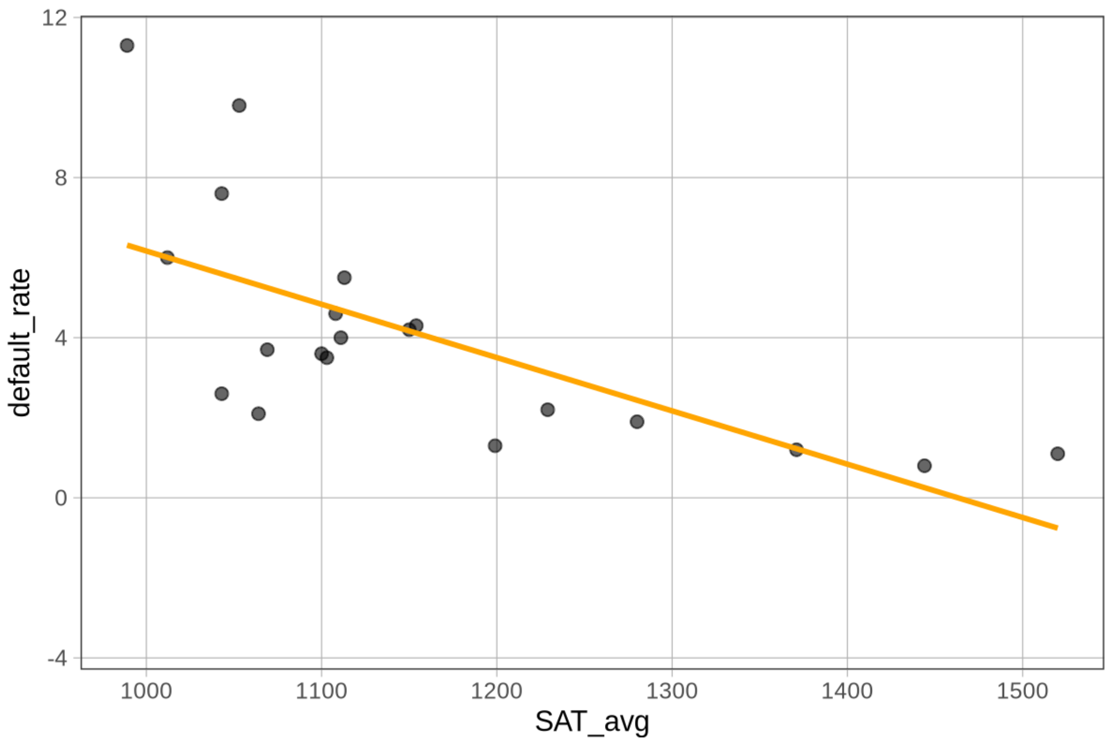
1.2 - Describe the direction of the relationship between `SAT_avg` and `default_rate`. Is it positive or negative? Why do you think this is?

Double-click this cell to type your answer here: The relationship between SAT_avg and default_rate is a moderately strong negative non linear relationship with no potential outliers. It is negative probably because the higher SAT average score you get, the smarter you are and if you are smarter you are less likely to default because you will land a well paying job/ got scholarships so you have to pay less, so easier to pay back due to academic excellence.

1.3 - Create the same scatterplot as above, but with the simple linear model between `default_rate` (outcome) and `SAT_avg` (predictor) overlayed on top.

Hint: Recall the `gf_lm` command from notebook 2.

```
[10]: # Your code goes here
gf_point(default_rate ~ SAT_avg, data = sample_dat)%>% gf_lm(color = "orange")
```



1.4 - Would you say that this model provides a "good" fit for this dataset? Explain.

Double-click this cell to type your answer here: I would not say this model is a "good" fit for this dataset because the data points vary highly from the LSRL.

1.5 - Use the `lm` command to fit the linear regression model, where we use `SAT_avg` (predictor) to predict `default_rate` (outcome) in the dataset `sample_dat`. Store the model in a variable named `sat_model_1` and use the `summary` command to print out information about the model fit.

```
[11]: # Your code goes here
sat_model_1 <- lm(default_rate ~ SAT_avg, data = sample_dat)
summary(sat_model_1)
```

```
Call:
lm(formula = default_rate ~ SAT_avg, data = sample_dat)

Residuals:
    Min      1Q  Median      3Q     Max
-3.2133 -1.2490 -0.0765  0.6196  4.9881

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 19.480574   3.989422   4.883  0.00012 ***
SAT_avg     -0.013315   0.003421  -3.893  0.00107 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.156 on 18 degrees of freedom
Multiple R-squared:  0.4571,	Adjusted R-squared:  0.4269
F-statistic: 15.15 on 1 and 18 DF,  p-value: 0.001067
```

Check yourself: The R^2 value shown in the model summary should be 0.4571

1.6 - Does the model's R^2 value indicate that this model provides a strong fit for this dataset? Explain.

Double-click this cell to type your answer here: The model does not indicate a strong fit for this dataset because nogt a very high amount, only 45.71% of the actual default_rate is accounted for by the LSRL.

1.7 - If this model were curved, rather than linear, do you believe the R^2 could be higher? Explain.

Double-click this cell to type your answer here: Yes, because the data looks exponential, so a curved model would fit better.

0.1.4 2.0 - Polynomial regression

Recall that simple linear regression follows this formula:

$$\hat{y} = \beta_0 + \beta_1 x$$

Where:

- β_0 is the intercept
- β_1 is the slope (coefficient of x)
- $\hat{y}$ is the predicted `default_rate`
- x is the value of `SAT_avg`

If we want to capture the curvature in a scatter plot by creating a non-linear model, we can use a technique called **polynomial regression**. For example, we could use a degree 2 polynomial (quadratic), which looks like this:

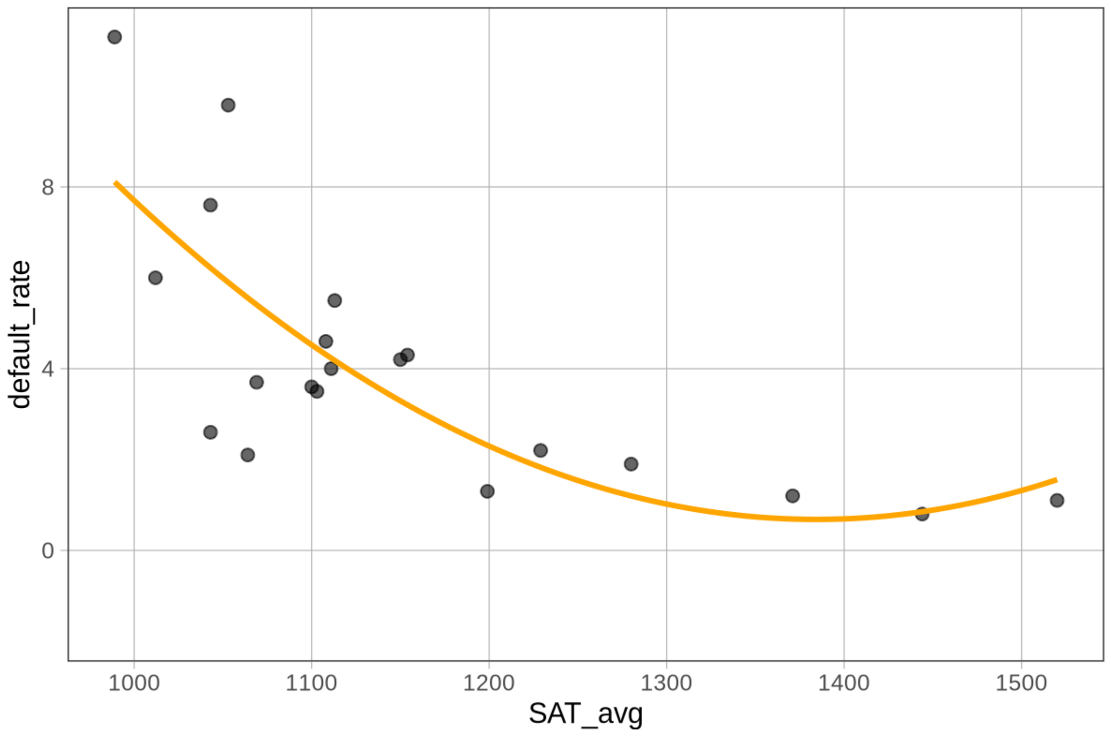
$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2$$

Where:

- β_0 is the intercept
- β_1 is the coefficient of x (linear term)
- β_2 is the coefficient of x^2 (squared term)
- $\hat{y}$ is the predicted `default_rate`
- x is the `SAT_avg`

Below, we visualize the fit of this degree-2 polynomial (quadratic) model between `SAT_avg` and `default_rate`:

[12]:
```
## Run this code but do not edit it
# create scatterplot: default_rate ~ SAT_avg, with degree 2 polynomial model␣
↪overlayed
gf_point(default_rate ~ SAT_avg, data = sample_dat) %>% gf_lm(formula = y ~␣
↪poly(x, 2), color = "orange")
```



2.1 - Make a prediction: Will this polynomial regression model have a higher or lower R^2 value than the linear regression model? Justify your reasoning.

Double-click this cell to type your answer here: Higher R^2 value than the linear regression model because more of the default_rates are accounted by the line.

Let's test your prediction. To do so, we'll first need to fit the polynomial model. We can fit a degree 2 polynomial to the data using the `poly()` function inside of the `lm()` function. Run the cell below to see how it's done.

```
[13]: ## Run this code but do not edit it
      # degree 2 polynomial model for default_rate ~ SAT_avg
      sat_model_2 <- lm(default_rate ~ poly(SAT_avg, 2), data = sample_dat)
      sat_model_2
```

```
Call:
lm(formula = default_rate ~ poly(SAT_avg, 2), data = sample_dat)

Coefficients:
      (Intercept)  poly(SAT_avg, 2)1  poly(SAT_avg, 2)2
            4.065             -8.391              4.355
```

The equation for this model would be

$$\hat{y} = 4.065 - 8.391x + 4.355x^2$$

Where:

- $\beta_0 = 4.065$ is the intercept
- $\beta_1 = -8.391$ is the coefficient of x, the linear term
- $\beta_2 = 4.355$ is the coefficient of x^2, the squared term
- $\hat{y}$ is the predicted `default_rate`
- x is the `SAT_avg`

2.2 - Use the `summary` command on `sat_model_2` to see summary information about the quadratic model.

```
[14]: # Your code goes here
      summary(sat_model_2)
```

```
Call:
lm(formula = default_rate ~ poly(SAT_avg, 2), data = sample_dat)

Residuals:
    Min      1Q  Median      3Q     Max
-3.6183 -0.9604  0.1192  0.9562  3.9014

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)         4.0650     0.4361   9.321  4.3e-08 ***
poly(SAT_avg, 2)1  -8.3909     1.9504  -4.302 0.000483 ***
poly(SAT_avg, 2)2   4.3553     1.9504   2.233 0.039280 *
---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.95 on 17 degrees of freedom
Multiple R-squared:  0.5802,	Adjusted R-squared:  0.5308
F-statistic: 11.75 on 2 and 17 DF,  p-value: 0.0006251
```

Check yourself: The R^2 value shown in the model summary should be 0.5802

2.3 - How does this model's R^2 value compare to that of the linear model? Was your prediction right? Explain.

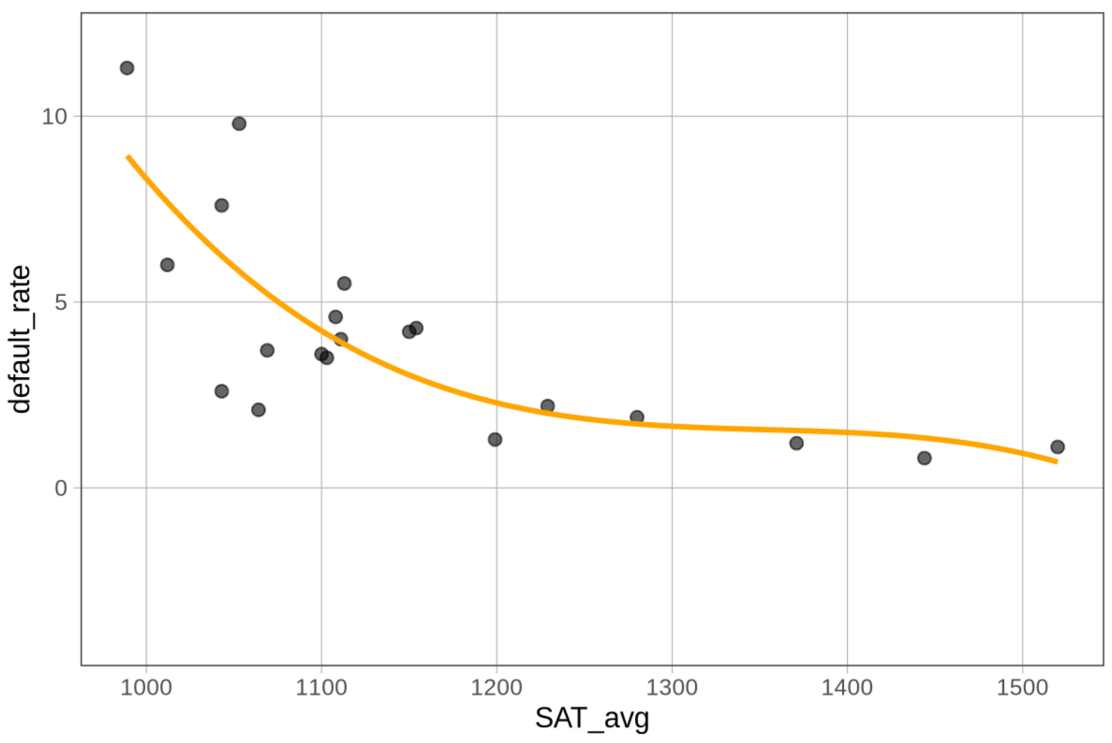
Double-click this cell to type your answer here: This model's R^2 value is higher than the linear model because .5802 is greater than .4571. Yes, my prediction was right because more of the default_rates are accounted by the line.

This analysis raises a natural question: Why stop at degree 2? By raising the degree, we can add more curves to our model, potentially better fitting the data! Let's visualize what happens when we increase the degree in our polynomial regression models.

Degree 3 Polynomial Model

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

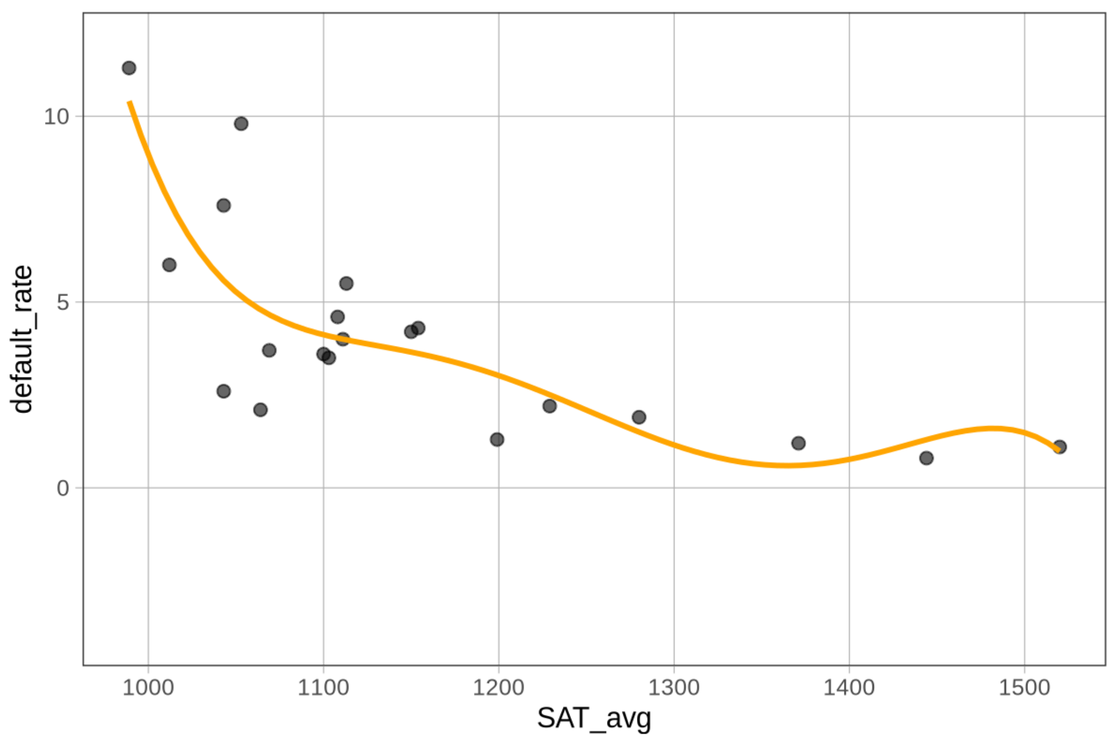
[15]:
```
## Run this code but do not edit it
# create scatterplot: default_rate ~ SAT_avg, with degree 3 polynomial model␣
↪overlayed
gf_point(default_rate ~ SAT_avg, data = sample_dat) %>% gf_lm(formula = y␣
↪~poly(x, 3), color = "orange") + ylim(-4,12)
```



Degree 5 Polynomial Model

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + +\beta_5 x^5$$

```
[16]: ## Run this code but do not edit it
      # create scatterplot: default_rate ~ SAT_avg, with degree 5 polynomial model␣
       ↪overlayed
      gf_point(default_rate ~ SAT_avg, data = sample_dat) %>% gf_lm(formula = y␣
       ↪~poly(x, 5), color = "orange") + ylim(-4,12)
```



Degree 12 Polynomial Model

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + +\beta_5 x^5 + +\beta_6 x^6 + ... + \beta_{12} x^{12}$$

Note: The following code is pre-run, to save computer space.

[17]:
```
## Note: This code was pre-run, to save computer space
# create scatterplot: default_rate ~ SAT_avg, with degree 12 polynomial model␣
↪overlayed
# gf_point(default_rate ~ SAT_avg, data = sample_dat) %>% gf_smooth(method =␣
↪"lm", formula = y ~poly(x,12), color = "orange") + ylim(-4,14)
```

2.4 - Examine each plot for the polynmial models with degrees 3, 5, 12. Which model do you think would have the largest R^2 value? Why?

Double-click this cell to type your answer here: The 12 degree polynomial model would have the largest R^2 value because more of the actual default_rates are accounted by the model.

To determine which polynomial model fits the data the best, we will fit models for each degree (3, 5, 12).

[18]:
```
## Run this code but do not edit it
# degree 3, 5, and 12 polynomial models for default_rate ~ SAT_avg
sat_model_3 <- lm(default_rate ~ poly(SAT_avg, 3), data = sample_dat)
sat_model_5 <- lm(default_rate ~ poly(SAT_avg, 5), data = sample_dat)
```

```
sat_model_12 <- lm(default_rate ~ poly(SAT_avg, 12), data = sample_dat)
```

Now we can compare each model's R^2 value. Normally, we use the `summary` command and read the R^2 value. However, since we've fit so many models, we don't want to print out the entire summary for each one.

Instead, we'll use commands like this: `summary(sat_model_1)$r.squared`. The `$` operator is used to extract just the `r.squared` element from the full `summary`. We execute this command for each model, then print the results for ease of comparison.

[19]:
```
## Run this code but do not edit it
# r-squared value for each model
r2_sat_model_1 <- summary(sat_model_1)$r.squared
r2_sat_model_2 <- summary(sat_model_2)$r.squared
r2_sat_model_3 <- summary(sat_model_3)$r.squared
r2_sat_model_5 <- summary(sat_model_5)$r.squared
r2_sat_model_12 <- summary(sat_model_12)$r.squared

# print each model's r-squared value
print(paste("The R squared value for the degree 1 model is", r2_sat_model_1))
print(paste("The R squared value for the degree 2 model is", r2_sat_model_2))
print(paste("The R squared value for the degree 3 model is", r2_sat_model_3))
print(paste("The R squared value for the degree 5 model is", r2_sat_model_5))
print(paste("The R squared value for the degree 12 model is", r2_sat_model_12))
```

```
[1] "The R squared value for the degree 1 model is 0.457055427196517"
[1] "The R squared value for the degree 2 model is 0.580193597490376"
[1] "The R squared value for the degree 3 model is 0.60314009577391"
[1] "The R squared value for the degree 5 model is 0.647445002110733"
[1] "The R squared value for the degree 12 model is 0.775449820710613"
```

Check yourself: The R^2 for the degree 5 model should be about 0.647

2.5 - The degree 12 model has the highest R^2 value. Does that mean it's the "best" model? Why or why not?

Hint: Think about which model would do the best for predicting *the rest of the data* from the original full dataset.

Double-click this cell to type your answer here: It's not the "best" model because it only predicts the current points on the graphy highly well, the rest of the data might not be predicted well.

0.1.5 3.0 - Prediction, model tuning, & machine learning

In prior notebooks, we've used our models to make inferences about default rates. However, sometimes in data science, we care more about predictions than we do about inferences. In particular, many data science tasks ask for making accurate predictions on *new* data - data that hadn't yet been collected when we first fit the model. This process of building models to predict new data, especially when it's automated, is called **machine learning.**

The key to machine learning is building models that make accurate predictions on **test** data - unseen data that weren't used when fitting the model. Let's see how this works. First, let's create a test dataset of 10 randomly sampled colleges. Importantly, these are colleges that **our models didn't see while fitting**:

[20]:
```
## Run this code but do not edit it
# create a data set to test the model with 10 new, randomly selected␣
↪observations
# not used to train the model
set.seed(23)
test_dat <- sample(dat, size = 10)
```

3.1 - Use the `head` command on the `test_dat` data set.

[21]:
```
# Your code goes here
head(test_dat)
```

A data.frame: 6 × 27

	OPEID <int>	name <chr>	city <chr>	state <chr>	reg <c
925	364600	Texas Woman's University	Denton	TX	Ro
284	1025600	Benedictine College	Atchison	KS	Mi
456	244900	Avila University	Kansas City	MO	Mi
615	291400	Catawba College	Salisbury	NC	Sou
913	363200	Texas A & M University-College Station	College Station	TX	Ro
1015	2136600	Wisconsin Lutheran College	Milwaukee	WI	Mi

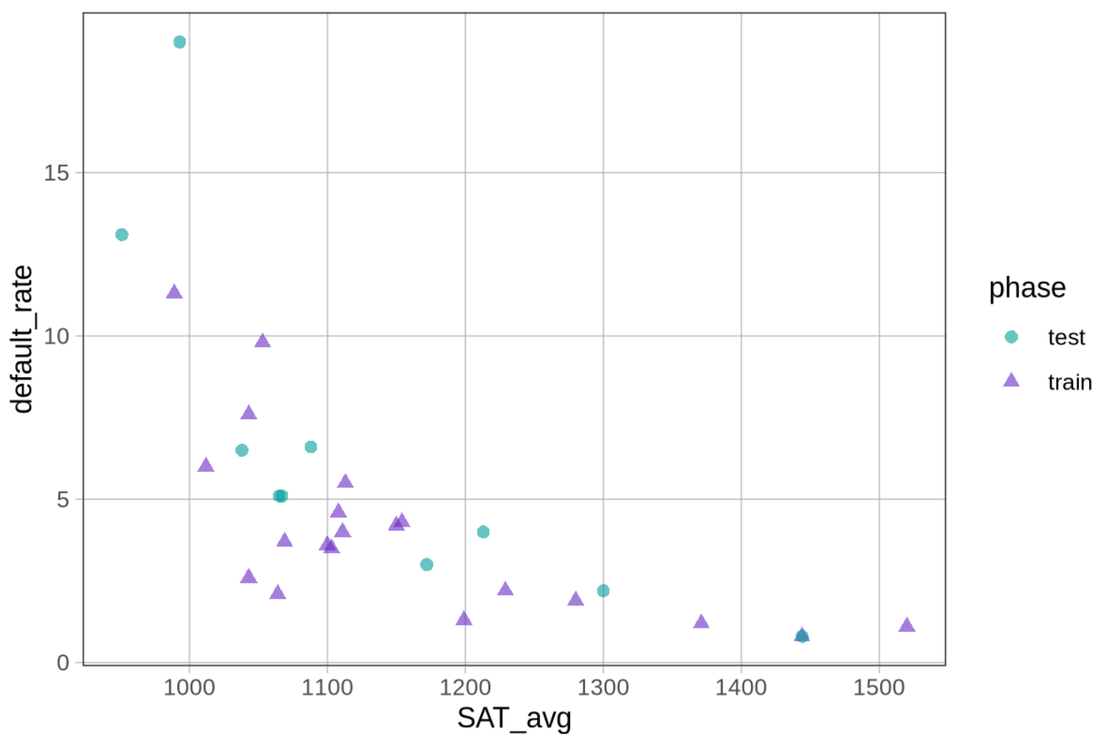
The following code visualizes the new test data alongside the training data (the data we used to originally fit our models).

[22]:
```
## Run this code but do not edit it
# label train and test sets
sample_dat$phase <- "train"
test_dat$phase <- "test"

# concatenate two datasets
full_dat <- rbind(sample_dat, test_dat)

# create scatterplot: default_rate ~ SAT_avg, with degree 5 polynomial model␣
↪overlayed
gf_point(default_rate ~ SAT_avg, data = full_dat, color = ~phase, shape =␣
↪~phase)
```

```
Warning message:
"The `scale_name` argument of `discrete_scale()` is deprecated as of
ggplot2
3.5.0."
```



3.2 - Of all the polynomial models we fit before, which do you think would do best in predicting the default rates in the test dataset?

Note: Use your gut and intution here. No calculations required.

Double-click this cell to type your answer here: 3 degree polynomial.

Let's see how good one of our models is at predicting default rates. The R code in the next cell uses the `predict` function to make predictions on the test dataset. In this case, output shows the predicted default rates for the 10 test set colleges, as predicted by our degree 5 model.

[23]:
```r
## Run this code but do not edit it
# get predictions for degree 5 model
pred_deg5 <- predict(sat_model_5, newdata = data.frame(SAT_avg =␣
 ↪test_dat$SAT_avg))
pred_deg5
```

1 4.71195777211723 **2** 2.80117510935355 **3** 4.76610631136996 **4** 5.83135112143679 **5** 1.1525539865398 **6** 3.41638983019025 **7** 9.85190742866518 **8** 18.1060265766056 **9** 1.29744229099948 **10** 4.28267983941371

So, how can we interpret these values? Well, the last college in our test set is `University of New Orleans`, which has an `SAT_avg` value of `1088` and a `default_rate` of `6.6`. It's shown here on the graph, alongside our degree 5 model.

Note: The following code is pre-run.

[24]:
```
### Run this code but do not edit it
## create scatterplot: default_rate ~ SAT_avg, with degree 5 polynomial model␣
 ↪overlayed
#gf_point(default_rate ~ SAT_avg, data = sample_dat, color = ~phase, shape =␣
 ↪~phase) %>% gf_lm(formula = y ~poly(x, 5), color = "orange") %>%␣
 ↪gf_point(default_rate ~ SAT_avg, data = full_dat, color = ~phase, shape =␣
 ↪~phase) + ylim(0,19)
```

Our degree 5 model's predicted default rate for this first data point was `4.28`. That means that our degree 5 model under-estimates the actual value for default rate by…

$$6.6 - 4.28 = 2.32$$

The model's prediction and error is visualized in the plot below:

So, its predicted default rate is "off" by about 2 percentage points! This is pretty amazing, considering the model had only 20 training data values and the `University of New Orleans` wasn't included among them. This is the power of machine learning! Predicting previously *unseen* data!

This is just one prediction. We're really interested in how this model performed across all its predictions. For that, let's measure its R^2 (prediciton strength) on the test set!

We can use the `cor` function to correlate the predictions with the actual default rates (r) and then square that value to get R^2, which gets us the prediction strength!

[25]:
```
## Run this code but do not edit it
# Get correlation between predicted and actual default rates in test set
cor(test_dat$default_rate, pred_deg5) ^ 2
```

```
0.616546630372038
```

We can now repeat this same process for all polynomial degrees.

[26]:
```
## Run this code but do not edit it
# Storing test set predictions for all models
pred_deg1 <- predict(sat_model_1, newdata = data.frame(SAT_avg =␣
 ↪test_dat$SAT_avg))
pred_deg2 <- predict(sat_model_2, newdata = data.frame(SAT_avg =␣
 ↪test_dat$SAT_avg))
pred_deg3 <- predict(sat_model_3, newdata = data.frame(SAT_avg =␣
 ↪test_dat$SAT_avg))
pred_deg5 <- predict(sat_model_5, newdata = data.frame(SAT_avg =␣
 ↪test_dat$SAT_avg))
pred_deg12 <- predict(sat_model_12, newdata = data.frame(SAT_avg =␣
 ↪test_dat$SAT_avg))

# print each model's r-squared value
print(paste("The test R squared value for the degree 1 model is",␣
 ↪cor(test_dat$default_rate, pred_deg1) ^ 2))
```

```
print(paste("The test R squared value for the degree 2 model is",␣
  ↪cor(test_dat$default_rate, pred_deg2) ^ 2))
print(paste("The test R squared value for the degree 3 model is",␣
  ↪cor(test_dat$default_rate, pred_deg3) ^ 2))
print(paste("The test R squared value for the degree 5 model is",␣
  ↪cor(test_dat$default_rate, pred_deg5) ^ 2))
print(paste("The test R squared value for the degree 12 model is",␣
  ↪cor(test_dat$default_rate, pred_deg12) ^ 2))
```

```
[1] "The test R squared value for the degree 1 model is 0.55824446697698"
[1] "The test R squared value for the degree 2 model is 0.70025122602337"
[1] "The test R squared value for the degree 3 model is 0.733729012084851"
[1] "The test R squared value for the degree 5 model is 0.616546630372038"
[1] "The test R squared value for the degree 12 model is 0.176121561427524"
```

Check yourself: The R^2 for the degree 5 model should be about 0.6165

3.3 - Compare the R^2 estimates for each model. Which models did well? Which models did poorly? Why do you think this is?

Double-click this cell to type your answer here: The 3 degree polynomial model fits the best because it has the highest R^2 value so the highest percentage of the default_rate data is accounted by the model. I think this is because that model is following the general trend of the data and not specific patterns within intervals.

3.4 - In machine learning, the central goal is to build our models so as to avoid "underfitting" and "overfitting" our models to the training data. What do you think these terms mean? Which of our models were underfit? Which do you think were overfit? Explain.

Double-click this cell to type your answer here: Underfitting means the model is not a good predictor in general. Overfitting means, the model is only a good predictor for a few points in the data set.

Recall that we built our polynomial models here with just one predictor: x (`SAT_avg`). Yet, those models could end up being quite complex...

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

Now, imagine that we wanted to bring in multiple predictors (x_1 = `SAT_avg`, x_2 = `net_tuition`, x_3 = `grad_rate`) for muliple regression. Plus, imagine that we decided to add in some polynomial terms for each of these predictors. We could end up with a model that looks ever more complicated, with literally hundreds of terms...

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \beta_4 x_2 + \beta_5 x_2^2 + \beta_5 x_2^3 + \beta_6 x_3 + \beta_7 x_3^2 + \ldots$$

3.5 - Is it always good to add more predictors and add more polynomial terms to your model? Explain why or why not.

Double-click this cell to type your answer here: No because too many could cause overfitting.

0.1.6 4.0 - In-class prediction competition

Now you have all the tools you need to build very powerful prediction models! This means that it's time for a friendly competition :)

The code below takes the full dataset and splits it into larger train and test datasets. 80% of the colleges will go into the train dataset. 20% will go into the test dataset. Because we all are setting the same seed (2024), everyone will get the exact same train and test sets:

[27]:
```
## Run but do not edit this code

# set training data to be 80% of all colleges
train_size <- floor(0.8 * nrow(dat))

## sample row indeces
set.seed(2024)
train_ind <- sample(seq_len(nrow(dat)), size = train_size)

train <- dat[train_ind, ]
test <- dat[-train_ind, ]
```

[28]:
```
dim(train)
```

1. 842 2. 26

[29]:
```
dim(test)
```

1. 211 2. 26

Now it's time to compete!

Goal: Create the most accurate prediction model of colleges' default rates.

Evaluation: Whichever student has the highest R^2 on the test set wins.

Guidelines Save your best model as an object called `my_model`. You are only allowed to fit models on the train set (not on the test set). You may use as many predictors and as many polynomial terms as you'd like. Just be warned: Don't fall into the trap of overfitting! Choose only the most important variables and keep your models simple, so that you can generalize well to the test set. Periodically test your model on the test set and then make adjustments as necessary.

Go!

[48]:
```
## Your code goes here
# Replace 'grad_rate + poly(SAT_avg,3)' with your own combo of variables and␣
 ↪poly terms
my_model <- lm(default_rate ~ poly(grad_rate,7) +␣
 ↪poly(SAT_avg,3)+poly(pct_fed_loan,7)+poly(pct_PELL,15)+poly(med_fam_income,3)+poly(net_tuit
 ↪data = train)
```

[47]:
```
# run this code to get the R^2 value on the test set from your model
test_predictions = predict(my_model, newdata = test)
print(paste("The test R^2 value was: ", cor(test$default_rate,␣
 ↪test_predictions) ^ 2))
```

```
[1] "The test R^2 value was:  0.712485802490887"
```

0.1.7 5.0 - NATIONWIDE prediction competition

Competition: We're hosting a *nationwide* competition to see which student can build the best model for predicting student loan default rates at different colleges. Here's an article about last year's winners.

Evaluation: Across the country, all students are using the same train and test sets as you did in the prior exercise to fit and evaluate their models. Your goal: Build a model that gives the best predictions on this test set. The student models that produce the highest R^2 value on the test set will be announced as champions!

Submission Process (due June 7, 2024 at 11:59pm CT): 1. Print and have a parent/guardian sign the media release form. This form gives permission to feature you and publish your results, in the event that you're a finalist! Take a picture or scan the signed form and submit it during as a part of Step #2 (below). 2. Submit this google form (note: you'll have to log into a google account), which allows you to upload your media release form, model, and notebook. This counts as your final submission.

Notes to avoid disqualification: - Do not change the seed (2024) in the code block that splits the data into the train and test sets. Using the common seed of 2024 will ensure everyone across the country has the exact same train/test split. - Make sure your model is fit using the `train` data. In other words, it should look like: `my_model <- lm(default_rate ~ ..., data = train)`. - Make sure your find the R^2 value on the `test` data, using the provided code. - There are ways to "cheat" on this competition by looking directly at the test set data values and designing your model to predict those values exactly (or approximately). However, based on the design of your model (which we'll see when you share your notebook), it's pretty easy for us to tell if you've done this. So, don't do it! Your submission will be discarded.

0.1.8 Feedback (Required)

Please take 2 minutes to fill out this anonymous notebook feedback form, so we can continue improving this notebook for future years!