Lecture 13

# Feature Engineering and Sklearn

Transforming Data to Improve Our Models.

# Today's Roadmap

- Implementing Models in Code
- Sklearn
- Feature Engineering Overview
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

# Implementing Models in Code

- **Implementing Models in Code**
- Sklearn
- Feature Engineering Overview
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

# Performing Ordinary Least Squares in Python

We previously derived the OLS estimate for the optimal model parameters:

$$\hat{\theta} = \left(\mathbb{X}^\top \mathbb{X}\right)^{-1} \mathbb{X}^\top \mathbb{Y}$$

In Python:

| Transpose | Inverse | Matrix Multiplication |
| --- | --- | --- |
| `matrix.T` | `np.linalg.inv(matrix)` | `matrix_1 @ matrix_2` |

```
theta_hat = np.linalg.inv(X.T @ X) @ X.T @ Y
```

# Sklearn

- Implementing Models in Code
- **Sklearn**
- Feature Engineering Overview
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

# `sklearn`: a Standard Library for Model Creation

So far, we have been doing the "heavy lifting" of model creation ourselves – via calculus, ordinary least squares, or gradient descent

In research and industry, it is more common to rely on data science libraries for creating and training models. We will use Scikit-Learn, commonly called **`sklearn`**


```
import sklearn
my_model = linear_model.LinearRegression()
my_model.fit(X, y)
my_model.predict(X)
```

# `sklearn`: a Standard Library for Model Creation

**`sklearn`** uses an object-oriented programming paradigm. Different types of models are defined as their own classes. To use a model, we initialize an instance of the model class.

- Don't worry if you are not familiar with objects in Python. You can think of sklearn as allowing you to "copy" an existing template of a useful model.

# The `sklearn` Workflow

At a high level, there are three steps to creating an **`sklearn`** model:

**1** Initialize a new model instance
*Make a "copy" of the model template*

**2** Fit the model to the training data
*Save the optimal model parameters*

**3** Use fitted model to make predictions
*Fitted model outputs predictions for y*

# The `sklearn` Workflow

At a high level, there are three steps to creating an **`sklearn`** model:

| | | |
|---|---|---|
| **1** | Initialize a new model instance<br>*Make a "copy" of the model template* | `my_model = lm.LinearRegression()` |
| **2** | Fit the model to the training data<br>*Save the optimal model parameters* | `my_model.fit(X, y)` |
| **3** | Use fitted model to make predictions<br>*Fitted model makes predictions for y* | `my_model.predict(X)` |

To extract the fitted parameters: **`my_model.coef_`** and **`my_model.intercept_`**

# Feature Engineering Overview

- Implementing Models in Code
- Sklearn
- **Feature Engineering Overview**
- One Hot Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

# What is a feature?

A feature is an input to our model

- So far, we have just used the raw data as features

We can also create new features to use an inputs to our model

We can choose/create **x** any way we like as long as our model follows the form $\hat{y} = x^T \theta$

The rest of the lecture will discuss different techniques we can use to create **x**:

- How can we create features from **quantitative data**?
- How can we create features from **categorical data**?
- How can we create features from **text data**? (lecture 5&6)

# Feature Engineering

**Feature Engineering** is the process of **transforming** the raw features **into more informative features** that can be used in modeling or EDA tasks.

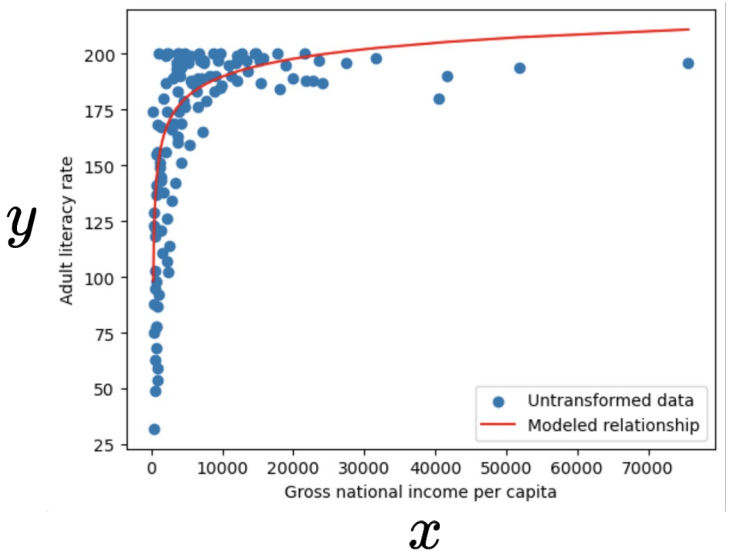
Feature engineering allows you to:

- Capture domain knowledge (e.g. periodicity or relationships between features).
- Express non-linear relationships using simple linear models.
- Encode non-numeric features to be used as inputs to models.
    - Example: Using the country of origin of a car as an input to modeling its efficiency.

# Transforming Features

Two observations:

- At the end of the Visualization lecture, we looked at transforming variables – we found that applying a transformation could help **linearize** a dataset
- In our work on modeling, we saw that linear **modeling works best** when our dataset has linear relationships




$$y^4 = m(\log x) + b$$

Putting ideas together:

**Feature engineering** = transforming features to improve model performance

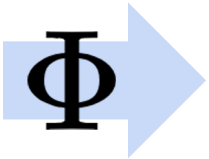
# Feature Functions

A **feature function** describes the transformations we apply to raw features in the dataset to create transformed features. Often, the dimension of the *featurized* dataset increases.

Example: a feature function that adds a squared feature to the design matrix

| | hp | mpg |
|---|---|---|
| **0** | 130.00 | 18.00 |
| **1** | 165.00 | 15.00 |
| **2** | 150.00 | 18.00 |
| **...** | ... | ... |
| **395** | 84.00 | 32.00 |
| **396** | 79.00 | 28.00 |
| **397** | 82.00 | 31.00 |

392 rows × 2 columns




| | hp | hp^2 | mpg |
|---|---|---|---|
| **0** | 130.00 | 16900.00 | 18.00 |
| **1** | 165.00 | 27225.00 | 15.00 |
| **2** | 150.00 | 22500.00 | 18.00 |
| **...** | ... | ... | ... |
| **395** | 84.00 | 7056.00 | 32.00 |
| **396** | 79.00 | 6241.00 | 28.00 |
| **397** | 82.00 | 6724.00 | 31.00 |

392 rows × 3 columns

Dataset of raw features:

$$\mathbb{X} \in \mathbb{R}^{n \times p}$$

After applying the feature function $\Phi$:

$$\Phi(\mathbb{X}) \in \mathbb{R}^{n \times p'}$$

# Feature Functions

A **feature function** describes the transformations we apply to raw features in the dataset to create transformed features. Often, the dimension of the *featurized* dataset increases.

Linear models trained on transformed data are sometimes written using the symbol $\Phi$ instead of X:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2$$
$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$



$$\hat{y} = \theta_0 + \theta_1 \phi_1 + \theta_2 \phi_2$$
$$\hat{\mathbb{Y}} = \Phi\theta$$

Shorthand for “the design matrix after feature engineering”

# Feature Functions

Designing feature functions is a major part of data science and machine learning.

- You'll have a chance to do lots of feature function design on the project.
- Fun fact: Much of the success of modern deep learning is because of its ability to automatically learn feature functions. See a course in deep learning for more.

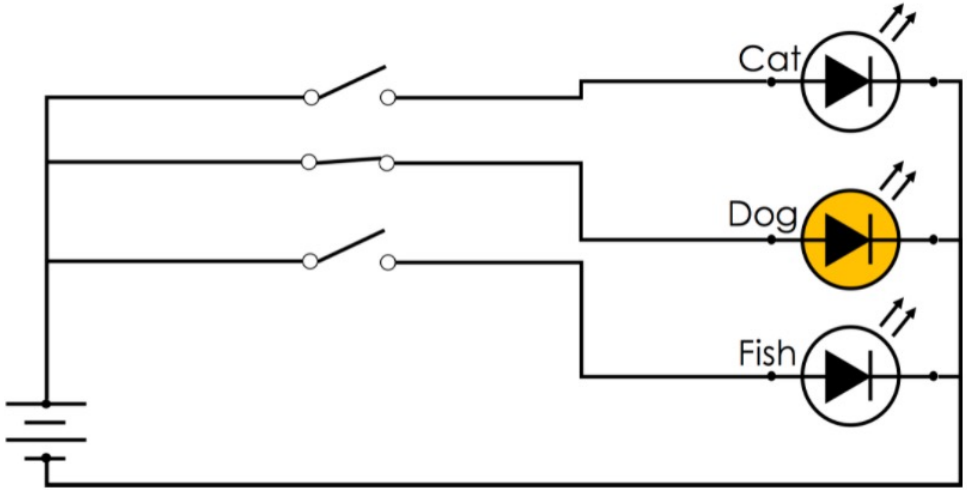
$$X \in \mathbb{R}^{n \times d} \longrightarrow \Phi \in \mathbb{R}^{n \times p}$$

# One Hot Encoding

- Implementing Models in Code
- Sklearn
- Feature Engineering Overview
- **One Hot Encoding**
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- Detecting Overfitting

# One-Hot Encoding

- One-Hot encoding, sometimes also called **dummy encoding**
- It is a simple mechanism to encode categorical data as real numbers such that the magnitude of each dimension is meaningful. Suppose a feature can take on $k$ distinct values
- For each distinct *possible* value, a new feature (dimension) is created. For each record, all the new features are set to zero except the one corresponding to the value in the original feature.
- The term one-hot encoding comes from a digital circuit encoding of a categorical state as particular "hot" wire:



# Regression Using Non-Numeric Features

We can also perform regression on non-numeric features. For example, for the tips dataset from last lecture, we might want to use the day of the week.

- One problem: Our linear model is always a linear combination of our features.

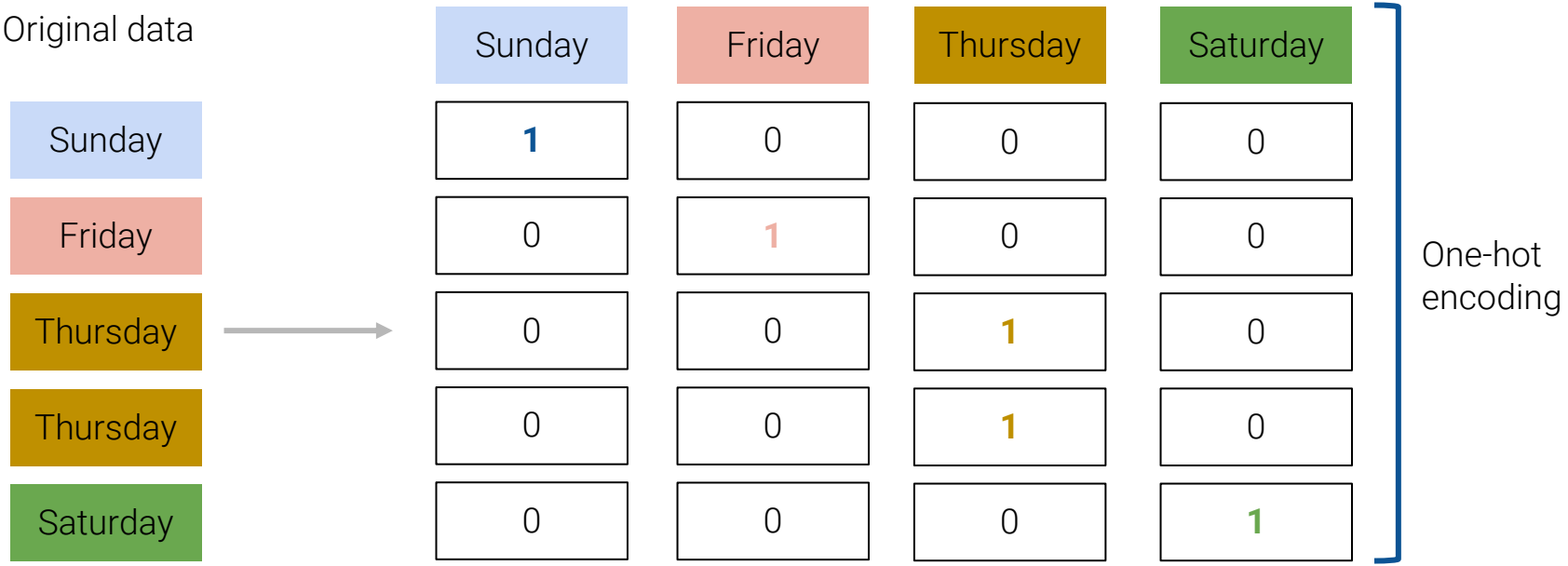
$$\hat{y} = \theta_1 \times bill + \theta_2 \times size + \theta_3 \times day$$

| total_bill | tip | sex | smoker | day | time | size |
|---|---|---|---|---|---|---|
| 28.97 | 3.00 | Male | Yes | Fri | Dinner | 2 |
| 17.81 | 2.34 | Male | No | Sat | Dinner | 4 |
| 13.37 | 2.00 | Male | No | Sat | Dinner | 2 |
| 15.69 | 1.50 | Male | Yes | Sun | Dinner | 2 |
| 15.48 | 2.02 | Male | Yes | Thur | Lunch | 2 |

# One-hot Encoding

One-hot encoding is a feature engineering technique to transform non-numeric data into numeric features for modeling

- Each category of a categorical variable gets its own feature
  - Value = 1 if a row belongs to the category
  - Value = 0 otherwise



# Regression Using the One-Hot Encoding

The one-hot encoded features can then be used in the design matrix to train a model

| | total_bill | size | day_Fri | day_Sat | day_Sun | day_Thur |
|---|---|---|---|---|---|---|
| **0** | 16.99 | 2 | 0.0 | 0.0 | 1.0 | 0.0 |
| **1** | 10.34 | 3 | 0.0 | 0.0 | 1.0 | 0.0 |
| **2** | 21.01 | 3 | 0.0 | 0.0 | 1.0 | 0.0 |
| **3** | 23.68 | 2 | 0.0 | 0.0 | 1.0 | 0.0 |
| **4** | 24.59 | 4 | 0.0 | 0.0 | 1.0 | 0.0 |

Raw features | One-hot encoded features

$$\hat{y} = \theta_1(\text{total\_bill}) + \theta_2(\text{size}) + \theta_3(\text{day\_Fri}) + \theta_4(\text{day\_Sat}) + \theta_5(\text{day\_Sun}) + \theta_6(\text{day\_Thur})$$

In shorthand: $\hat{y} = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 + \theta_4\phi_4 + \theta_5\phi_5 + \theta_6\phi_6$

# Using Non-Numeric Features: One Hot Encoding

One approach is to use what is known as a “one hot encoding.”

- Give every category its own feature, with value = 1 if that category applies to that row.
- Can do this using the get_dummies function. Then join with the original table with pd.concat.

| | total_bill | size | day | Thur | Fri | Sat | Sun |
|---|---|---|---|---|---|---|---|
| **193** | 15.48 | 2 | Thur | 1 | 0 | 0 | 0 |
| **90** | 28.97 | 2 | Fri | 0 | 1 | 0 | 0 |
| **25** | 17.81 | 4 | Sat | 0 | 0 | 1 | 0 |
| **26** | 13.37 | 2 | Sat | 0 | 0 | 1 | 0 |
| **190** | 15.69 | 2 | Sun | 0 | 0 | 0 | 1 |

```
dummies = pd.get_dummies(data['day'])
```

```
data_w_dummies = pd.concat([three_feature_data, dummies], axis=1)
```

# Fitting a Model

If we fit a linear model, the result is a 6 dimensional model.

- $\theta_1 = 0.093$: How much to weight the total bill.
- $\theta_2 = 0.187$: How much to weight the party size.
- $\theta_3 = 0.668$: How much to weight the fact that it is Thursday.
- $\theta_4 = 0.746$: How much to weight the fact that it is Friday.
- $\theta_5 = 0.621$: How much to weight the fact that it is Saturday.
- $\theta_6 = 0.732$: How much to weight the fact that it is Sunday.

Resulting prediction is: $\hat{y} = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 + \theta_4\phi_4 + \theta_5\phi_5 + \theta_6\phi_6$

```
from sklearn.linear_model import LinearRegression
f_with_day = LinearRegression(fit_intercept=False)
f_with_day.fit(data_w_dummies[["total_bill", "size", "Thur",
                               "Fri", "Sat", "Sun"]], data["tip"])
```

# Test Your Understanding

If we fit a linear model, the result is a 6 dimensional model.

- $\theta_1 = 0.093$: How much to weight the total bill.
- $\theta_2 = 0.187$: How much to weight the party size.
- $\theta_3 = 0.668$: How much to weight the fact that it is Thursday.
- $\theta_4 = 0.746$: How much to weight the fact that it is Friday.
- $\theta_5 = 0.621$: How much to weight the fact that it is Saturday.
- $\theta_6 = 0.732$: How much to weight the fact that it is Sunday.

Resulting prediction is: $\hat{y} = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 + \theta_4\phi_4 + \theta_5\phi_5 + \theta_6\phi_6$

To test your understanding, what tip would the model predict for a party of 3 with a $50 check eating on a Thursday?

# Test Your Understanding

If we fit a linear model, the result is a 6 dimensional model.

- $\theta_1 = 0.093$: How much to weight the total bill.
- $\theta_2 = 0.187$: How much to weight the party size.
- $\theta_3 = 0.668$: How much to weight the fact that it is Thursday.
- $\theta_4 = 0.746$: How much to weight the fact that it is Friday.
- $\theta_5 = 0.621$: How much to weight the fact that it is Saturday.
- $\theta_6 = 0.732$: How much to weight the fact that it is Sunday.

Resulting prediction is: $\hat{y} = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 + \theta_4\phi_4 + \theta_5\phi_5 + \theta_6\phi_6$

To test your understanding, what tip would the model predict for a party of 3 with a \$50 check eating on a Thursday?

$$\hat{y} = 0.093 \times 50 + 0.187 \times 3 + 0.668 = \$5.88$$

# Verifying in Python

If we fit a linear model, the result is a 6 dimensional model.

- $\theta_1 = 0.093$: How much to weight the total bill.
- $\theta_2 = 0.187$: How much to weight the party size.
- $\theta_3 = 0.668$: How much to weight the fact that it is Thursday.
- $\theta_4 = 0.746$: How much to weight the fact that it is Friday.
- $\theta_5 = 0.621$: How much to weight the fact that it is Saturday.
- $\theta_6 = 0.732$: How much to weight the fact that it is Sunday.

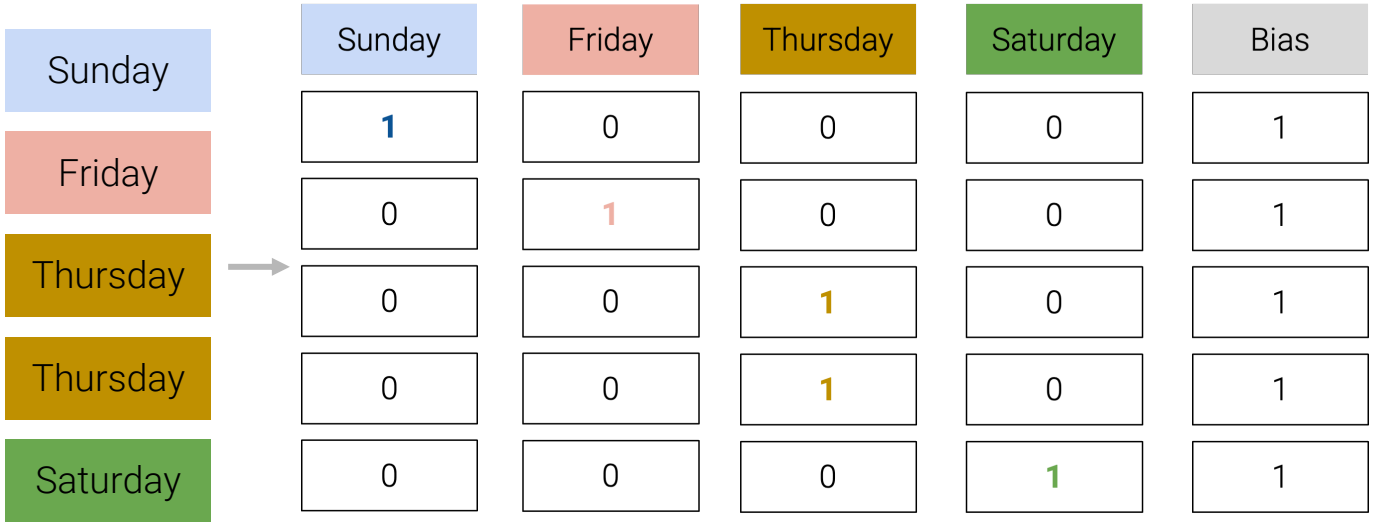
Resulting prediction is: $\hat{y} = \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 + \theta_4\phi_4 + \theta_5\phi_5 + \theta_6\phi_6$

To test your understanding, what tip would the model predict for a party of 3 with a \$50 check eating on a Thursday?

```
f_with_day.predict([[50, 3, 1, 0, 0, 0]])
```

# One-hot Encode Wisely!

Any set of one-hot encoded columns will always sum to a column of all ones.




The bias column is a linear combination of the OHE columns

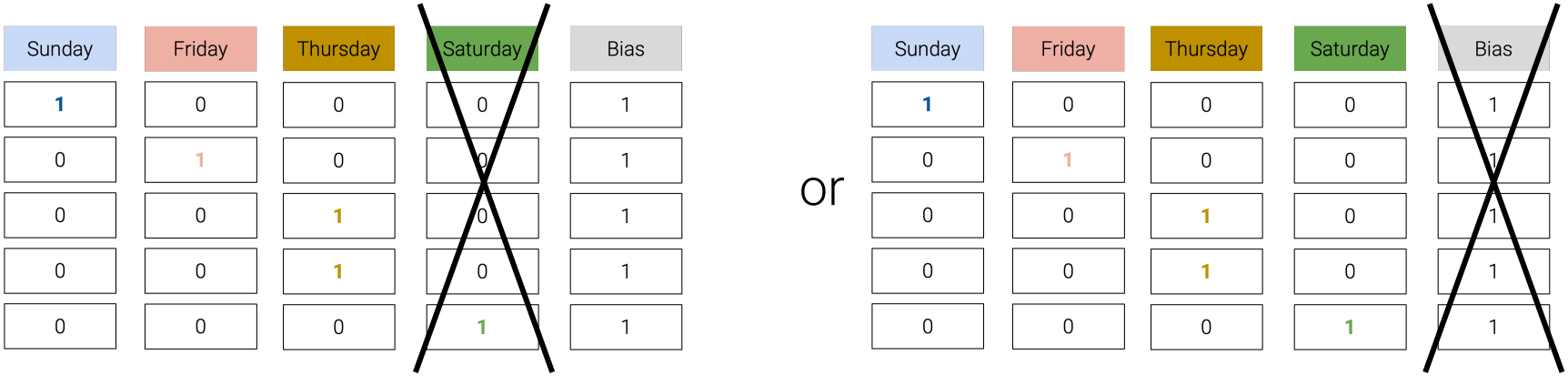
If we also include a bias column in the design matrix, there will be linear dependence in the model. $\mathbb{X}^\top\mathbb{X}$ is not invertible, and our OLS estimate $\hat{\theta} = (\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\mathbb{Y}$ fails.

How to resolve? Omit one of the one-hot encoded columns *or* do not include an intercept term

# One-hot Encode Wisely!

How to resolve? Omit one of the one-hot encoded columns *or* do not include an intercept term

Adjusted design matrices:



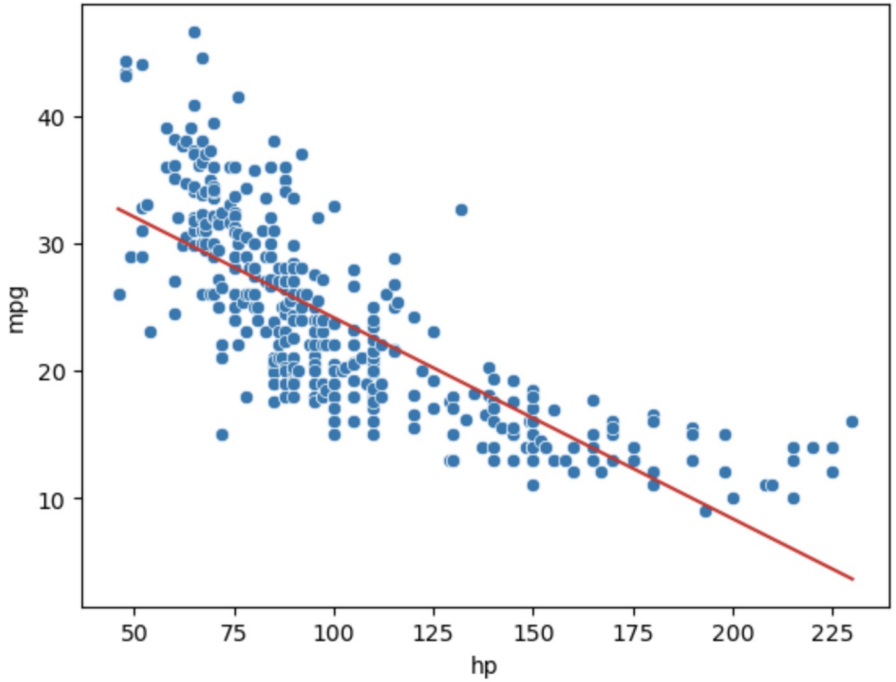

We still retain the same information – in both approaches, the omitted column is simply a linear combination of the remaining columns

# High Order Polynomial Example

- Implementing Models in Code
- Sklearn
- Feature Engineering Overview
- One Hot Encoding
- **High Order Polynomial Example**
- Variance and Training Error
- Overfitting
- Detecting Overfitting

# Accounting for Curvature

We've seen a few cases now where models with linear features have performed poorly on datasets with a clear non-linear curve.




$$\hat{y} = \theta_0 + \theta_1(\text{hp})$$

MSE: 23.94

When our model uses only a single linear feature (**hp**), it cannot capture non-linearity in the relationship

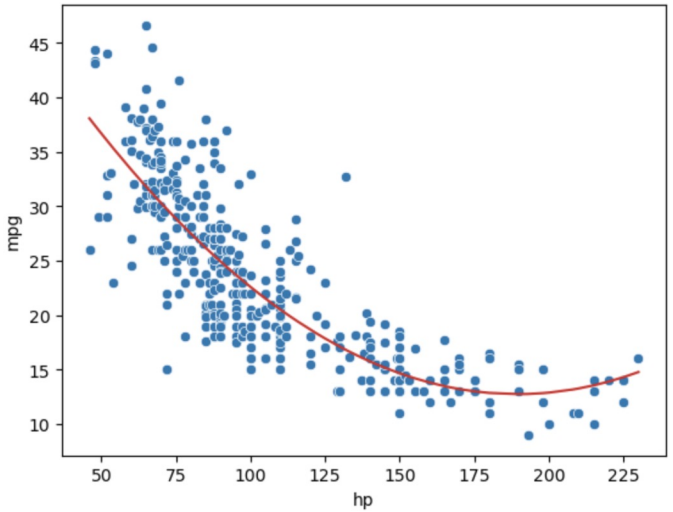
Solution: incorporate a non-linear feature!

# Polynomial Features

We create a new feature: the square of the **hp**

$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2)$$

This is still a **linear model**. Even though there are non-linear *features*, the model is linear with respect to $\theta$



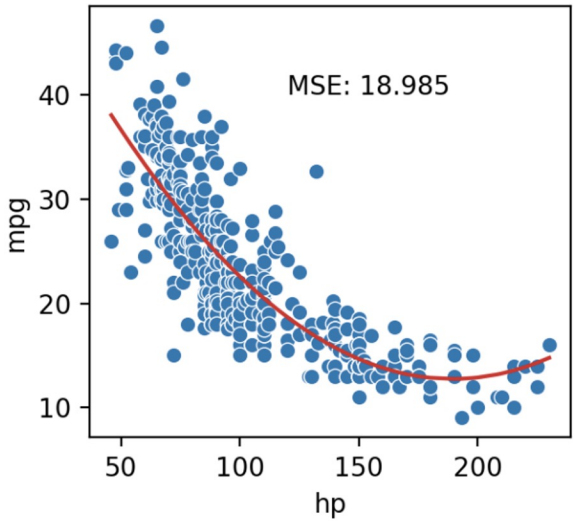

Degree of model: 2
MSE: 18.98

Looking a lot better: our predictions capture the curvature of the data.

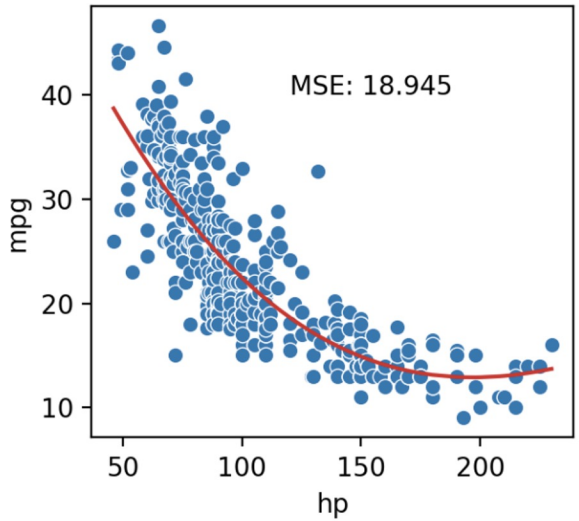
# Polynomial Features

What if we add more polynomial features?

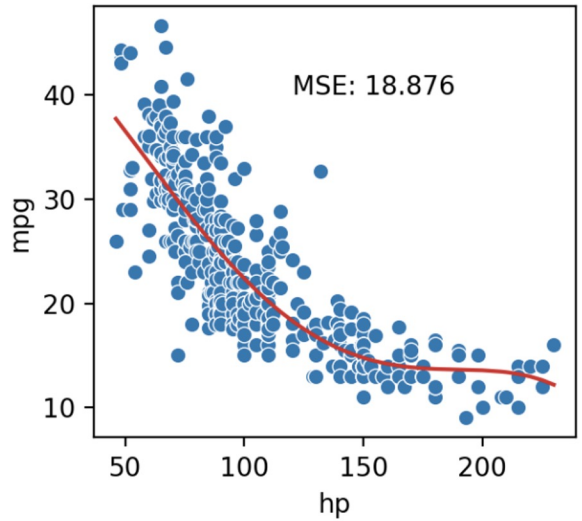
$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2)$




$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2) + \theta_3(\text{hp}^3)$




$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2) + \theta_3(\text{hp}^3) + \theta_4(\text{hp}^4)$

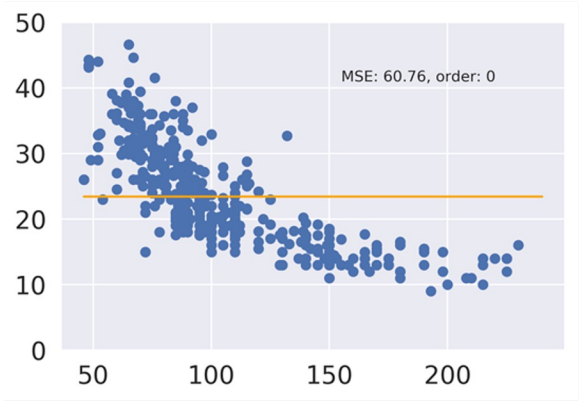



MSE continues to decrease with each additional polynomial term

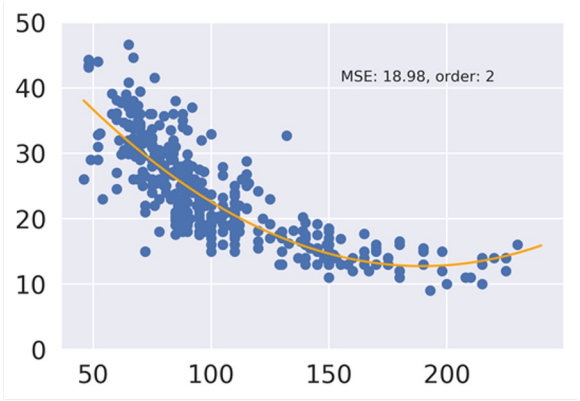
# Variance and Training Error

- Implementing Models in Code
- Sklearn
- Feature Engineering Overview
- One Hot Encoding
- High Order Polynomial Example
- **Variance and Training Error**
- Overfitting
- Detecting Overfitting

# Going Even Higher Order







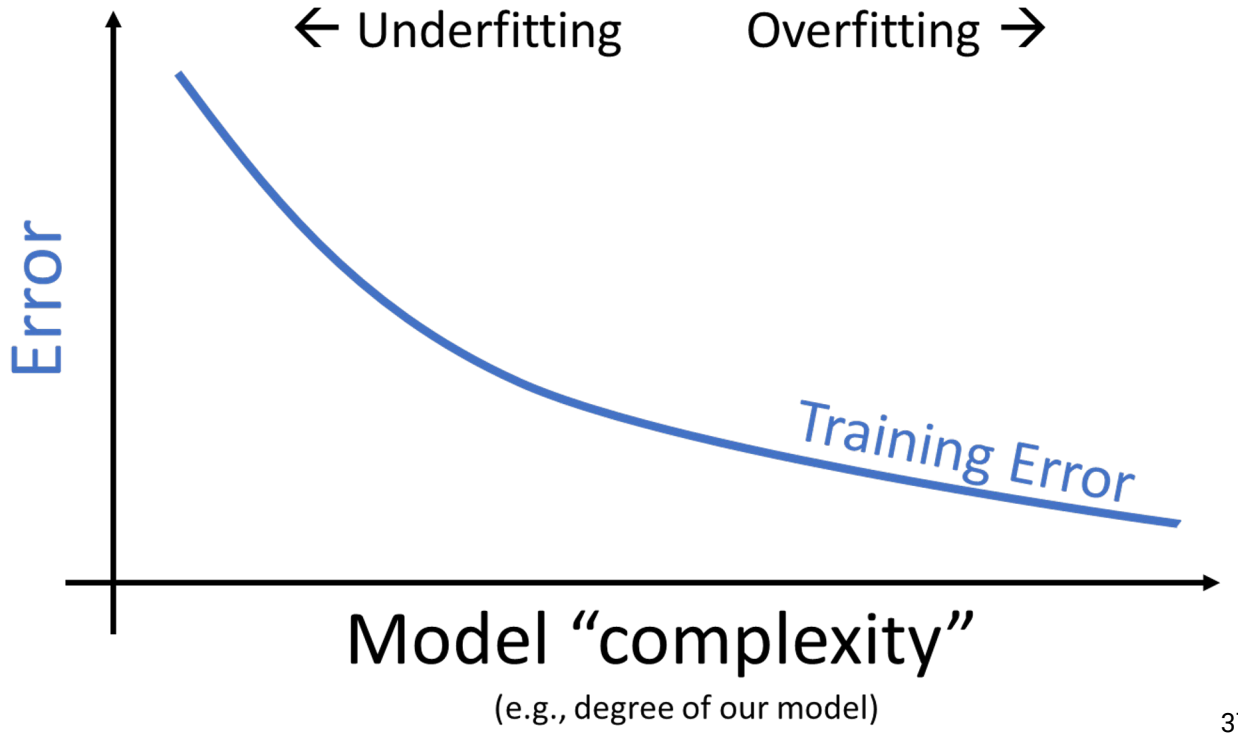




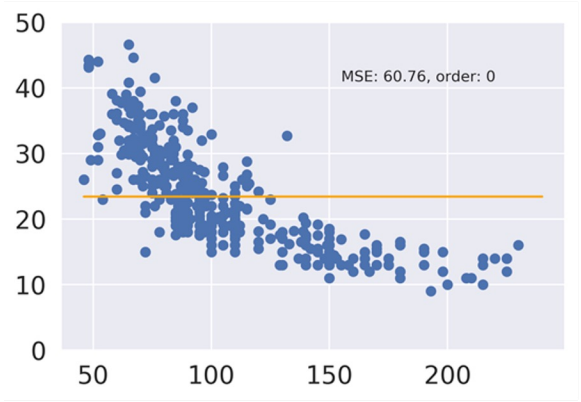
As we increase model complexity, MSE drops from 60.76 to 23.94 to … 18.43.

# Error vs. Complexity

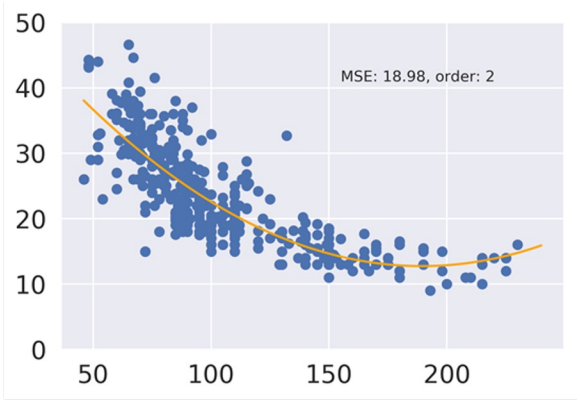
As we increase the complexity of our model, we see that the error on our training data (also called the **Training Error**) decreases.



# Going Even Higher Order











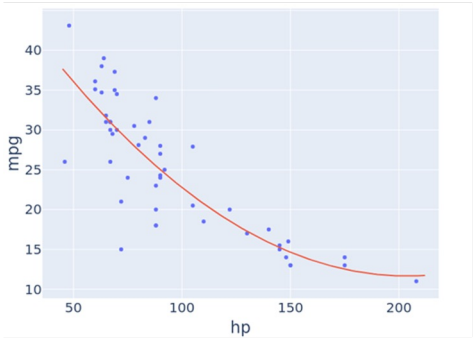
As we **increase model complexity**, **MSE drops** from 60.76 to 23.94 to … 18.43. **At the same time**, the fit curve grows increasingly erratic and **sensitive** to the data.

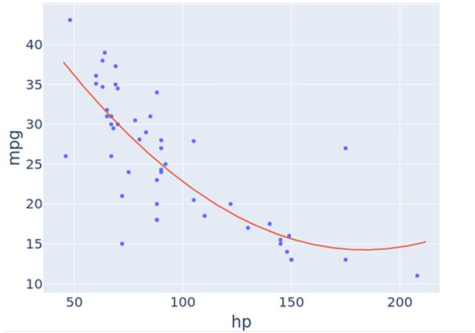
# Example on a Subset of the Data

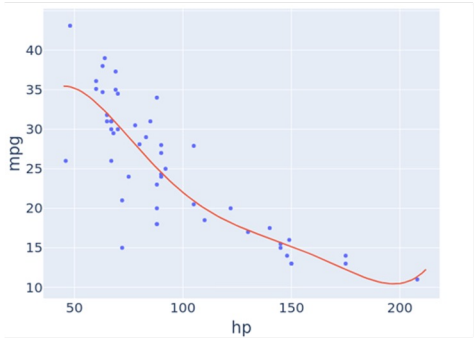
On top, we see the results of fitting two very similar datasets using an order 2 model ($\theta_1 + \theta_2\, x + \theta_3\, x^2$). The resulting fit (model parameters) is close.

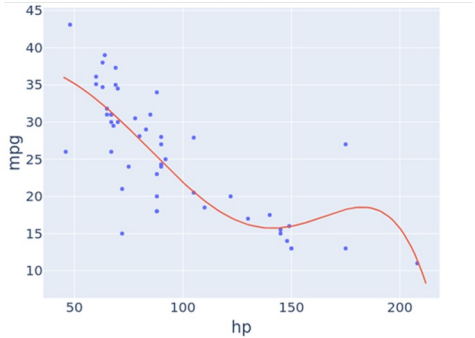
On bottom, we see the results of fitting the same datasets using an order 6 model ($\theta_1 + \ldots + \theta_7\, x^6$). We see very different predictions, especially for hp around 170.

In ML, this **sensitivity** to data is known as "**variance**".



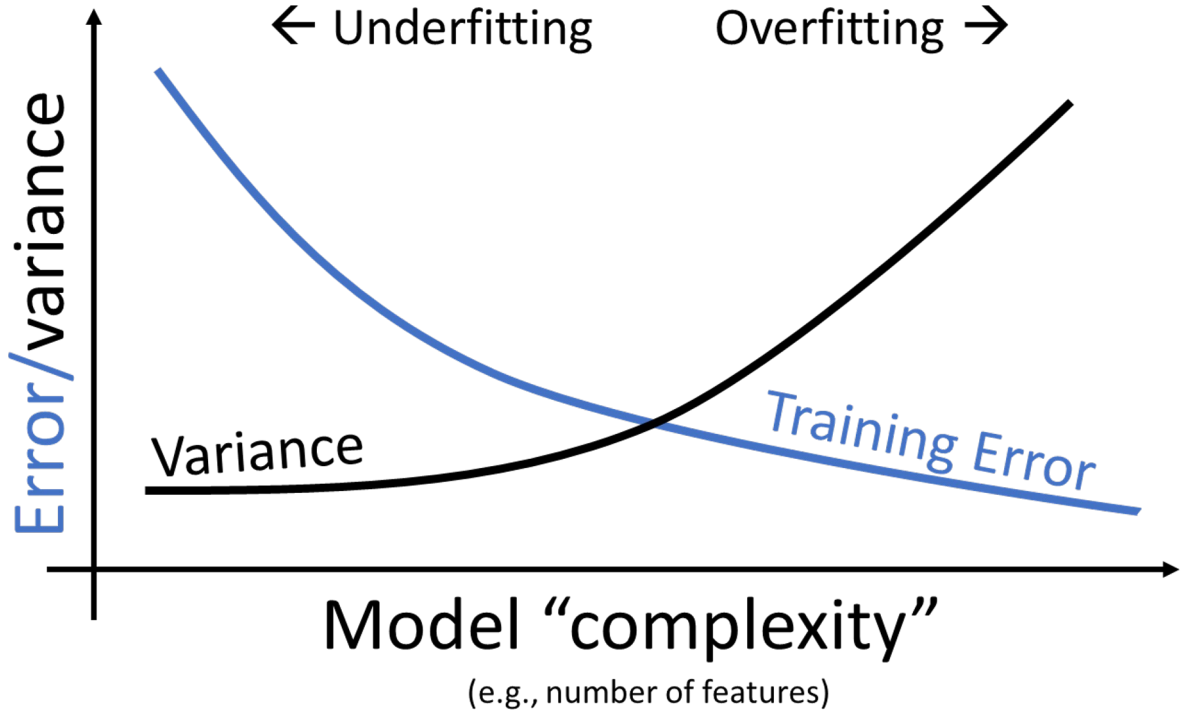



# Error vs. Complexity

As we increase the complexity of our model:

- **Training error** decreases.
- **Variance** increases.

How do we find a model that hits the “sweet spot”?



# Overfitting

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example
- One Hot Encoding
- Word Encoding
- High Order Polynomial Example
- Variance and Training Error
- **Overfitting**
- Detecting Overfitting

# Four Parameter Model with Four Data Points

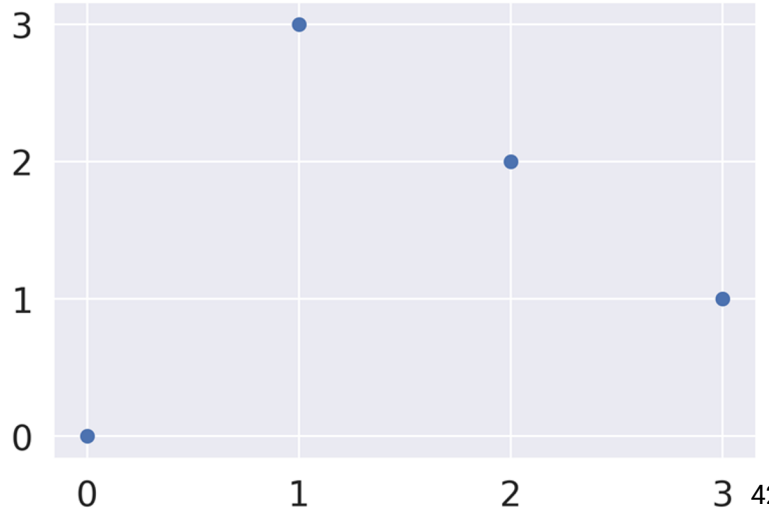
Algebra fact: Given **N** non-overlapping data points, we can always find a polynomial of degree **N-1** that goes through all those points.

Example: $x_1, y_1 = (0, 0), x_2, y_2 = (1, 3), x_3, y_3 = (2, 2), x_4, y_4 = (3, 1)$

There exist $\theta_0, \theta_1, \theta_2, \theta_3$ such that $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ goes through all of these points.

Just solve the system of equations:

$$\theta_0 = 0$$
$$\theta_0 + \theta_1 + \theta_2 + \theta_3 = 3$$
$$\theta_0 + 2\theta_1 + 4\theta_2 + 8\theta_3 = 2$$
$$\theta_0 + 3\theta_1 + 9\theta_2 + 27\theta_3 = 1$$



# Four Parameter Model with Four Data Points

Algebra fact: Given N non-overlapping data points, we can always find a polynomial of degree N-1 that goes through all those points.

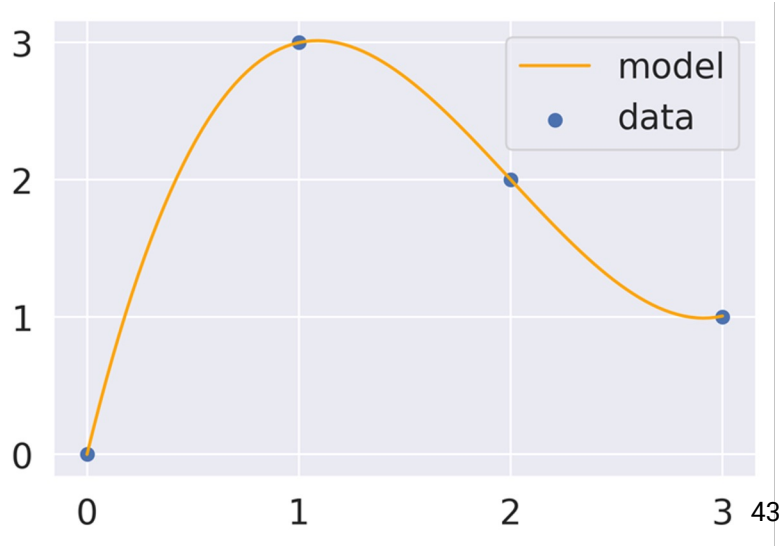
Example: $x_1, y_1 = (0, 0), x_2, y_2 = (1, 3), x_3, y_3 = (2, 2), x_4, y_4 = (3, 1)$

There exist $\theta_0, \theta_1, \theta_2, \theta_3$ such that $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ goes through all of these points, meaning **MSE = 0**.

Just solve the system of equations:

$$\begin{aligned} \theta_0 &= 0 \\ \theta_0 + \theta_1 + \theta_2 + \theta_3 &= 3 \\ \theta_0 + 2\theta_1 + 4\theta_2 + 8\theta_3 &= 2 \\ \theta_0 + 3\theta_1 + 9\theta_2 + 27\theta_3 &= 1 \end{aligned}$$

$$\begin{aligned} \theta_0 &= 0 \\ \theta_1 &= 19/3 \\ \theta_2 &= -4 \\ \theta_3 &= 2/3 \end{aligned}$$



# Reminder: Solving a System of Linear Equations is Equivalent to Matrix Inversion

Solving our linear equations is equivalent to a matrix inversion.

$$\hat{y} = \theta_0 + \theta_1 x^1 + \theta_2 x^2 + \theta_3 x^3$$
$$x_1, y_1 = (0, 0), x_2, y_2 = (1, 3),$$
$$x_3, y_3 = (2, 2), x_4, y_4 = (3, 1)$$

$$\theta_0 = 0$$
$$\theta_0 + \theta_1 + \theta_2 + \theta_3 = 3$$
$$\theta_0 + 2\theta_1 + 4\theta_2 + 8\theta_3 = 2$$
$$\theta_0 + 3\theta_1 + 9\theta_2 + 27\theta_3 = 1$$

Specifically, we're solving $\hat{Y} = \Phi\theta$ ,where $\hat{Y}$ is predictions, $\Phi$ is features, and $\theta$ is parameters.

$$\begin{bmatrix} 0 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

# The Danger of Overfitting

This principle generalizes. If we have **100 data points** with only **a single feature**, we can always generate 99 polynomial features from the original feature, then fit a **100 parameter model** $\Phi$ that perfectly fits our data.

- MSE is always zero.
- Model is totally useless!!!

The problem we're facing here is **overfitting**: Our model is effectively just memorizing existing data and cannot handle new situations at all.

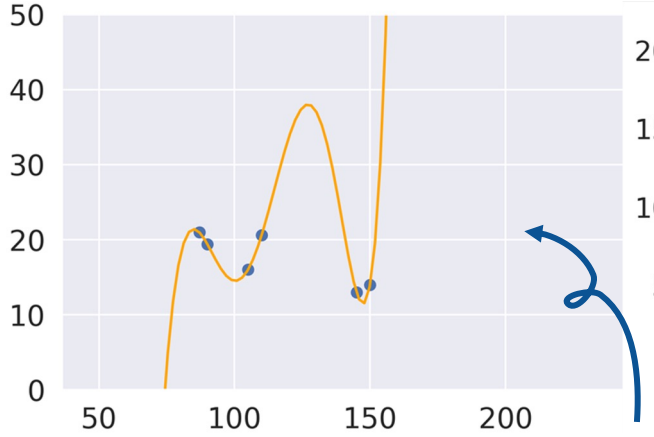
# Model Sensitivity in Action

Let's build an **order-5** model that perfectly fits **6 randomly chosen vehicles** from our fuel efficiency dataset.

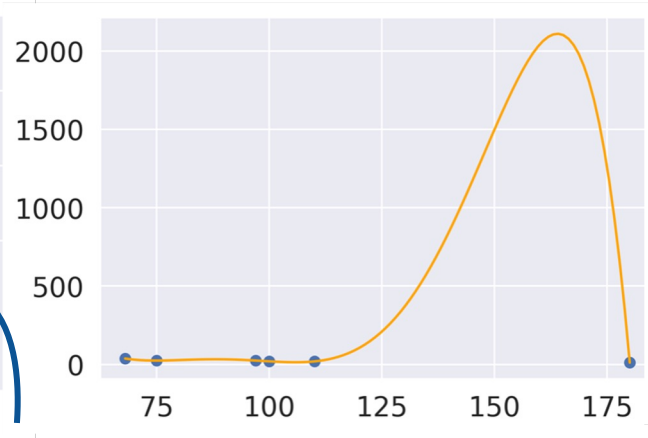
No matter which vehicles we pick, we'll almost always get an essentially* perfect fit.

(*With the caveat that real computers do not have infinite precision, and thus for even higher order models, this will break due to rounding errors.)

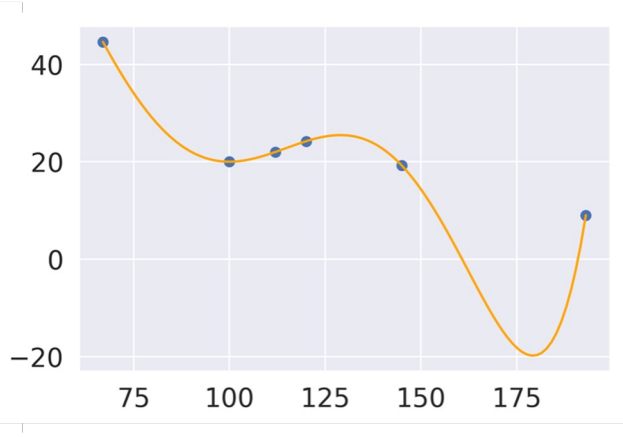
Order-5 model perfectly fit to Dataset 1




Order-5 model perfectly fit to Dataset 2




Order-5 model perfectly fit to Dataset 3



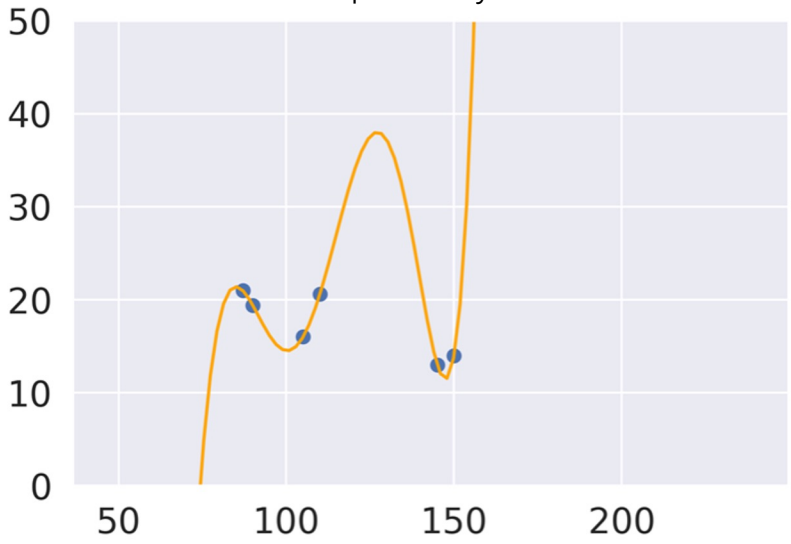

How well does one of these order-5 models **generalize** to the rest of the data?

# Comparing a Fit On Our Six Data Points with the Full Data Set

When overlaid on our full data set, we see that our predictions are terrible.

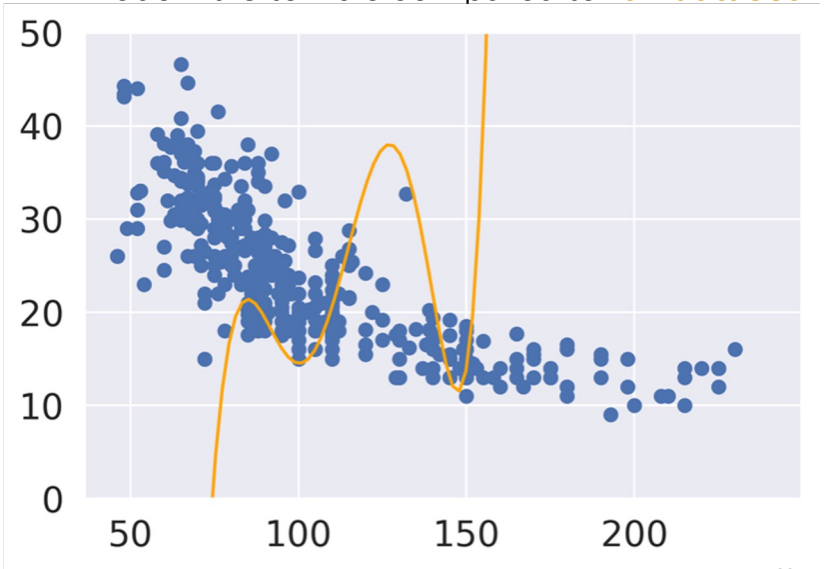
- Zero error on the **training set** (i.e. the set of data we used to train our model, Dataset 1).
- … but enormous error on a bigger sample of **real world data**.
- Since most data that we work with are just samples of some larger population, this is bad!

Order-5 model perfectly fit to **Dataset 1**




…

Model fit is terrible compared to **full dataset**



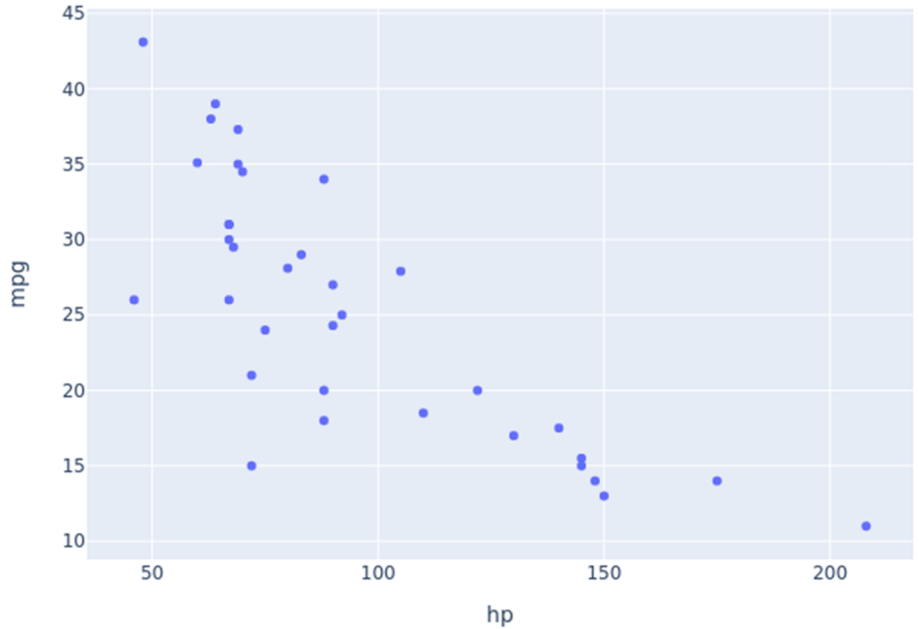
# Detecting Overfitting

- Fitting a Linear Parabolic Model
- Feature Engineering Overview
- High Dimensional Feature Engineering Example
- One Hot Encoding
- Word Encoding
- High Order Polynomial Example
- Variance and Training Error
- Overfitting
- **Detecting Overfitting**

# Our 35 Samples

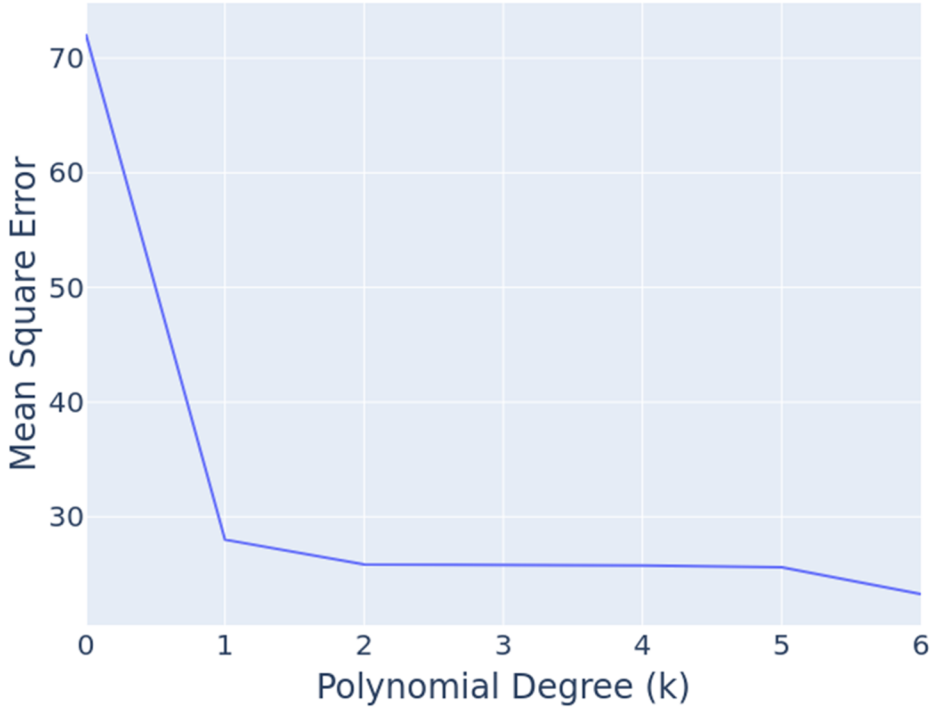
Consider a model fit on only the 35 data points.

- We'll try various degrees and try to find the one we like best.



# Fitting Various Degree Models

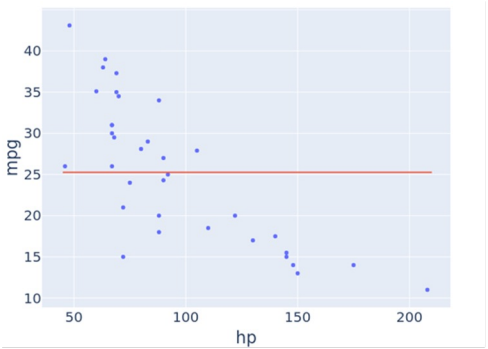
If we fit models of degree 0 through 7 of this model. The MSE is as shown below.

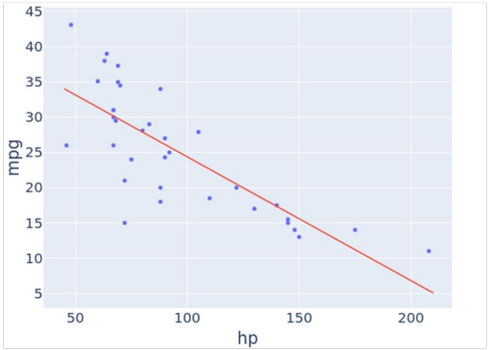


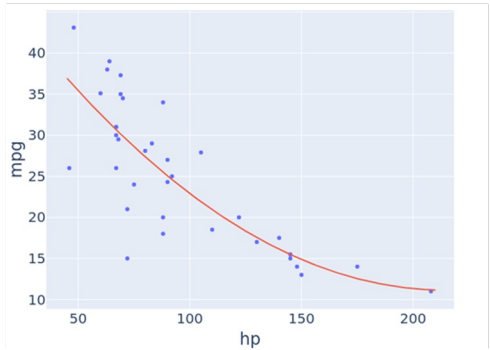

| k | MSE |
|---|---|
| 0 | 72.091396 |
| 1 | 28.002727 |
| 2 | 25.835769 |
| 3 | 25.831592 |
| 4 | 25.763052 |
| 5 | 25.609403 |
| 6 | 23.269001 |

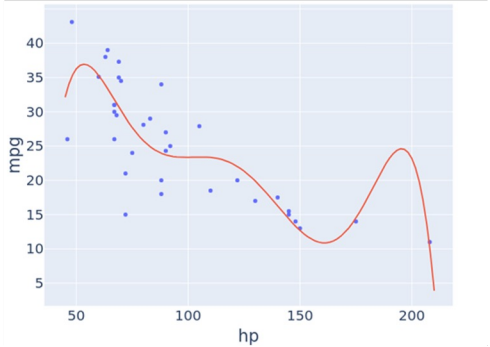
# Visualizing the Models

Below we show the order 0, 1, 2, and 6 models.



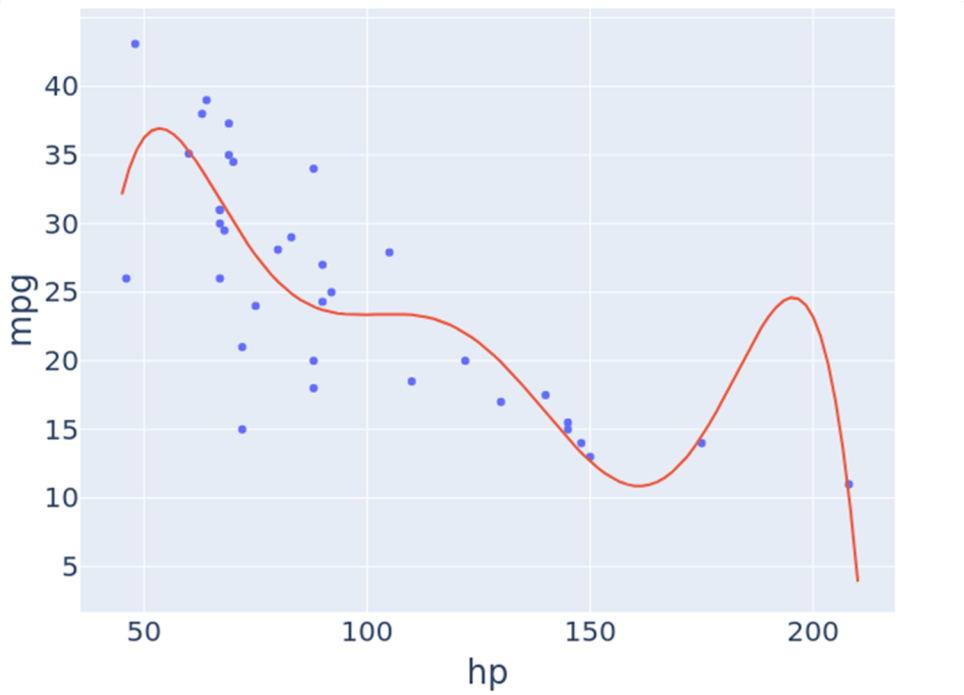




| k | MSE |
|---|---|
| 0 | 72.091396 |
| 1 | 28.002727 |
| 2 | 25.835769 |
| 3 | 25.831592 |
| 4 | 25.763052 |
| 5 | 25.609403 |
| 6 | 23.269001 |

# An Intuitively Overfit Model

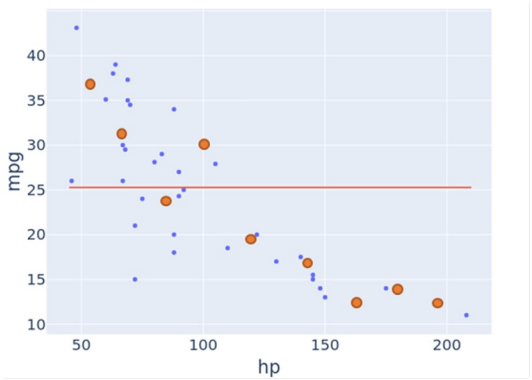
Intuitively, the degree 6 model below feels like it is overfit.

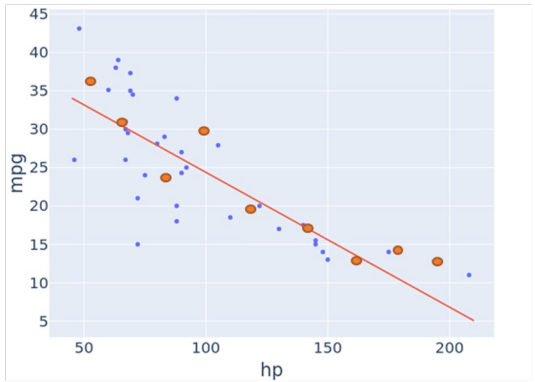
- More specifically: It seems that if we collect more data, i.e. draw more samples from the same distribution, we are worried this model will make poor predictions.

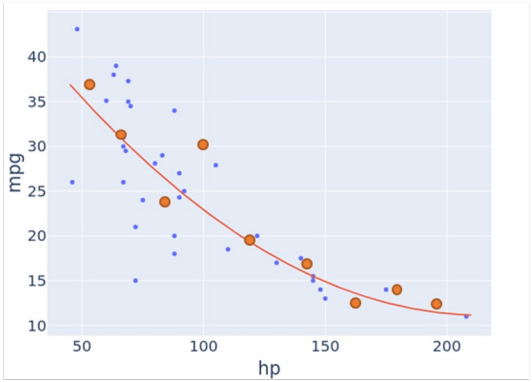


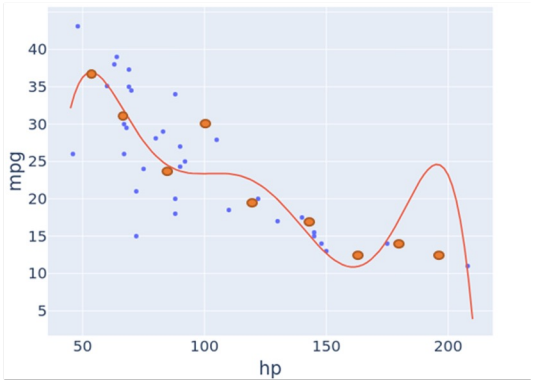
# Collecting More Data to Prove a Model is Overfit

Suppose we collect the 9 new orange data points. Can compute MSE for our original models **without refitting using the new orange data points**.




| k | MSE |
|---|---|
| 0 | 72.091396 |
| 1 | 28.002727 |
| 2 | 25.835769 |
| 3 | 25.831592 |
| 4 | 25.763052 |
| 5 | 25.609403 |
| 6 | 23.269001 |

Original 35 data points

| k | MSE |
|---|---|
| 0 | 69.198210 |
| 1 | 31.189267 |
| 2 | 27.387612 |
| 3 | 29.127612 |
| 4 | 34.198272 |
| 5 | 37.182632 |
| 6 | 53.128712 |

New 9 data points

# Collecting More Data to Prove a Model is Overfit

Suppose we have 7 models and don't know which is best.

- Can't necessarily trust the training error. We may have overfit!

We could wait for more data and see which of our 7 models does best on the new points.

- Unfortunately, that means we need to wait for more data. May be very expensive or time consuming.