LECTURE 30

# Feature Engineering

Transforming Data to Improve Our Models.

**CSCI 3022, Fall 2023**

Maribeth Oscamou

Content credit: Acknowledgments

# Announcements

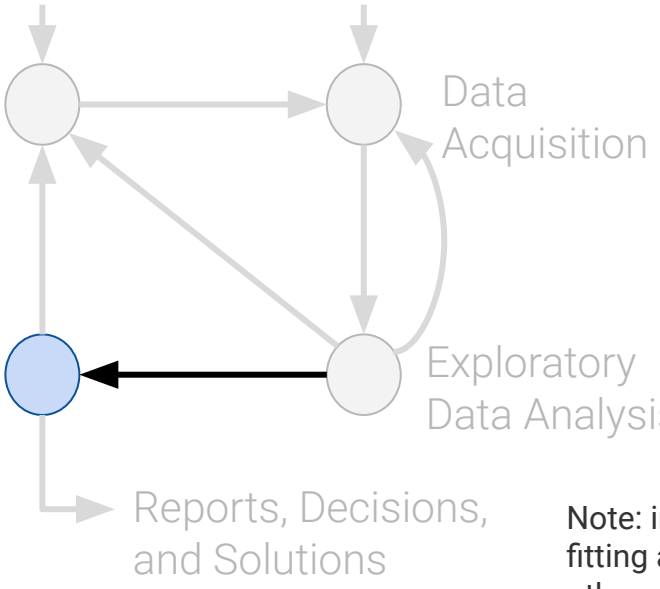
Project Part 1 due next Thursday at 11:59pm MT  (plan accordingly - no late submissions accepted)

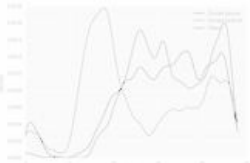
Last quiz (quiz 10) next Friday

# Goals for this Lecture

- **Feature Engineering**
  - **Incorporating Categorical Data with One Hot Encoding**
  - Polynomial Features
- Complexity and Overfitting
-

# Plan for Next Few Lectures: Modeling



Note: in this class, we call the overall process o
fitting and interpreting models *modeling*, while
others call it *machine learning*.

**(today)**

| Modeling I: Intro to Modeling, Simple Linear Regression | → | Modeling II: Different models, loss functions, linearization | → | Modeling III: Multiple Linear Regression |
|---|---|---|---|---|

# Three Reasons for Building Models

## Reason 1:

To explain **complex phenomena** occurring in the world we live in.

- How are the parents' average heights related to the children's average heights?
- How do an object's velocity and acceleration impact how far it travels?

Often times, we care about creating models that are **simple and interpretable**, allowing us to understand what the relationships between our variables are.

## Reason 2:

To make **accurate predictions** about unseen data.

- Can we predict if an email is spam or not?
- Can we generate a one-sentence summary of this 10-page long article?

Other times, we care more about making extremely accurate predictions, at the cost of having an uninterpretable model. These are sometimes called **black-box models**, and are common in fields like deep learning.

## Reason 3:

To make **causal inferences** about if one thing causes another thing.

- Can we conclude that smoking *causes* lung cancer?
- Does a job training program cause increases in employment and wage?

Much harder question because most statistical tools are designed to infer association not causation

This won't be the focus of this class, but will be if you go on to take more advanced classes (Stat 156, Data 102)

Most of the time, we want to strike a balance between **interpretability** and **accuracy**.

# Recall: Ordinary Least Squares

1. Choose a model

2. Choose a loss function

3. Fit the model

4. Evaluate model performance

**Multiple Linear Regression**

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p$$

L2 Loss

**Mean Squared Error (MSE)**

$$R(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Minimize average loss with linear algebra and/or geometry

Visualize, RMSE, Multiple $R^2$

In statistics, this model + loss is called **Ordinary Least Squares (OLS)**.

The solution to OLS are the minimizing loss for parameters $\hat{\theta}$, also called the **least squares estimate**.

# Feature Engineering

- **Feature Engineering**
  - **One Hot Encoding**
  - Polynomial Features
- Complexity and Overfitting

# 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.

Why doesn't sklearn doesn't have `SquareRegression/PolynomialRegression`.

- We can translate these into linear models with features that are polynomials of x.
- Feature engineering saves sklearn a lot of redundancy in their library.
- As you saw in homework, linear models have really nice properties.

# Feature Functions

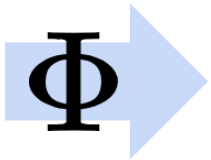
A **feature function** describes the transformations we apply to raw features in the dataset to create transformed features. Often, the dimension of the *featurized* dataset increases.

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

Dataset of raw features:



| | 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

After applying the feature function $\Phi$:

- As number of features grows, we can capture arbitrarily complex relationships.

# Feature Function

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 some models have the ability to automatically learn feature functions. See APPM 4720/5720( Applied Deep Learning) for more.

# The Modeling Process: Example with Housing Data

Consider a dataset of houses sold in the Bay Area in 2006

**How can we use this data to build a model to accurately predict the price of a house?**

| | date | city | price | br | lsqft | bsqft |
|---|---|---|---|---|---|---|
| **1576** | 2006-10-15 | Richmond | 80000.0 | 3.0 | 1900.0 | 870.0 |
| **620** | 2006-05-28 | Berkeley | 120000.0 | 1.0 | 6150.0 | 534.0 |
| **2472** | 2006-11-12 | Richmond | 125000.0 | 2.0 | 3920.0 | 951.0 |
| **1861** | 2006-05-14 | Richmond | 130000.0 | 3.0 | 2870.0 | 1607.0 |
| **2262** | 2006-01-01 | Richmond | 145000.0 | 1.0 | 3375.0 | 541.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **760** | 2006-10-22 | Piedmont | 3500000.0 | 4.0 | 10000.0 | 4362.0 |
| **696** | 2006-07-23 | Piedmont | 3625000.0 | 4.0 | 15000.0 | 3527.0 |
| **964** | 2006-11-05 | Lamorinda | 3650000.0 | 4.0 | 43560.0 | 4606.0 |
| **697** | 2006-01-15 | Piedmont | 3975000.0 | NaN | 15000.0 | 5974.0 |
| **688** | 2006-05-21 | Piedmont | 3980000.0 | 7.0 | 62890.0 | 7468.0 |

2762 rows × 6 columns

# The Modeling Process: Example with Housing Data

Consider a dataset of houses sold in the Bay Area in 2006

| | date | city | price | br | lsqft | bsqft |
|---|---|---|---|---|---|---|
| **1576** | 2006-10-15 | Richmond | 80000.0 | 3.0 | 1900.0 | 870.0 |
| **620** | 2006-05-28 | Berkeley | 120000.0 | 1.0 | 6150.0 | 534.0 |
| **2472** | 2006-11-12 | Richmond | 125000.0 | 2.0 | 3920.0 | 951.0 |
| **1861** | 2006-05-14 | Richmond | 130000.0 | 3.0 | 2870.0 | 1607.0 |
| **2262** | 2006-01-01 | Richmond | 145000.0 | 1.0 | 3375.0 | 541.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **760** | 2006-10-22 | Piedmont | 3500000.0 | 4.0 | 10000.0 | 4362.0 |
| **696** | 2006-07-23 | Piedmont | 3625000.0 | 4.0 | 15000.0 | 3527.0 |
| **964** | 2006-11-05 | Lamorinda | 3650000.0 | 4.0 | 43560.0 | 4606.0 |
| **697** | 2006-01-15 | Piedmont | 3975000.0 | NaN | 15000.0 | 5974.0 |
| **688** | 2006-05-21 | Piedmont | 3980000.0 | 7.0 | 62890.0 | 7468.0 |

2762 rows × 6 columns

**How can we use this data to build a model to accurately predict the price of a house?**

## Step 1: EDA

**Key Properties to Consider in EDA (Lec 6):**

**Structure** -- the "shape" of a data file

**Granularity** -- how fine/coarse is each datum

- Each record represents a sale of a home in the SF Bay Area during 2006.

**Scope** -- how (in)complete is the data

- If a home was sold twice during 2006, then there are two records in the table. And if a home in the Bay Area was not up for sale during this time, then it does not appear in the dataset.

**Temporality** -- how is the data situated in time

- This is all data from one year, 2006. During 2006 housing prices were relatively stable, so we don't need to account for temporal trends in price
-

**Faithfulness** -- how well does the data capture "reality"

`date` : Date the house sold

`price` : Price the house sold for

`city` : In this case we consider 4 cities: Richmond, Berkeley, Piedmont and a combination of 3 cities Lafayette, Moraga and Morinda that we name Lamorinda

`br` : Number of bedrooms

`lsqft` : Square feet of the lot

`bsqft` : Square feet of the house

# The Modeling Process: Example with Housing Data

Consider a dataset of houses sold in the Bay Area in 2006

`date` : Date the house sold

`price` : Price the house sold for

`city` : In this case we consider 4 cities: Richmond, Berkeley, Piedmont and a combination of 3 cities Lafayette, Moraga and Morinda that we name Lamorinda

`br` : Number of bedrooms

`lsqft` : Square feet of the lot

`bsqft` : Square feet of the house

**How can we use this data to build a model to accurately predict the price of a house?**

## Step 1: EDA

Do any of the features have a strong linear association with price?

Calculate correlation between price and each of the variables in the dataset:

```
sfh.corr()
```

| | price | br | lsqft | bsqft |
|---|---|---|---|---|
| **price** | 1.000000 | 0.435298 | 0.589539 | 0.755521 |
| **br** | 0.435298 | 1.000000 | 0.270283 | 0.669279 |
| **lsqft** | 0.589539 | 0.270283 | 1.000000 | 0.433104 |
| **bsqft** | 0.755521 | 0.669279 | 0.433104 | 1.000000 |

Sale price correlates most highly with house size, called `bsqft` for building square feet.

Next step: Confirm with a scatter plot of sale price against house size that the association is linear.

# The Modeling Process: Example with Housing Data

Consider a dataset of houses sold in the Bay Area in 2006

`date` : Date the house sold

`price` : Price the house sold for

`city` : In this case we consider 4 cities: Richmond, Berkeley, Piedmont and a combination of 3 cities Lafayette, Moraga and Morinda that we name Lamorinda

`br` : Number of bedrooms

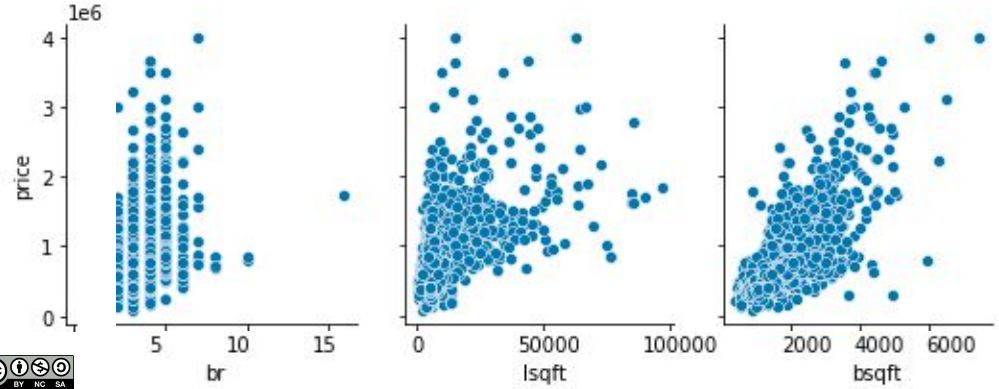
`lsqft` : Square feet of the lot

`bsqft` : Square feet of the house

**How can we use this data to build a model to accurately predict the price of a house?**

## Step 2: Visualization

Examine scatterplots of each of the features vs price to visually determine if price is linearly related to features available in the dataset




```
sfh.corr()
```

| | price | br | lsqft | bsqft |
|---|---|---|---|---|
| **price** | 1.000000 | 0.435298 | 0.589539 | 0.755521 |
| **br** | 0.435298 | 1.000000 | 0.270283 | 0.669279 |
| **lsqft** | 0.589539 | 0.270283 | 1.000000 | 0.433104 |
| **bsqft** | 0.755521 | 0.669279 | 0.433104 | 1.000000 |

**Notice - Issue with overplotting! (many datapoints crowded on top of one another, hard to visualize the underlying structure/patterns)**

- For the br data, we should use boxplots or violinplots instead

# The Modeling Process: Example with Housing Data

`date` : Date the house sold

`price` : Price the house sold for

`city` : In this case we consider 4 cities: Richmond, Berkeley, Piedmont and a combination of 3 cities Lafayette, Moraga and Morinda that we name Lamorinda

`br` : Number of bedrooms

`lsqft` : Square feet of the lot

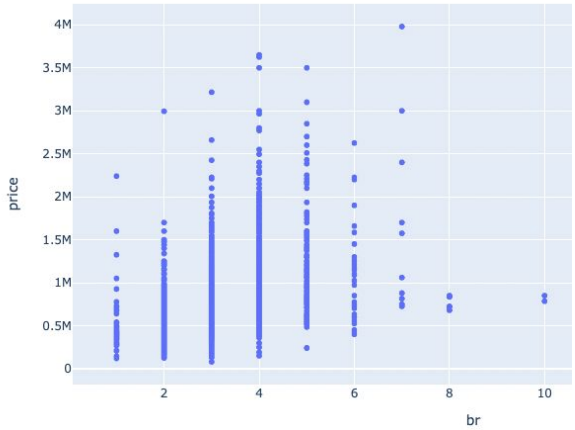
`bsqft` : Square feet of the house

Consider a dataset of houses sold in the Bay Area in 2006

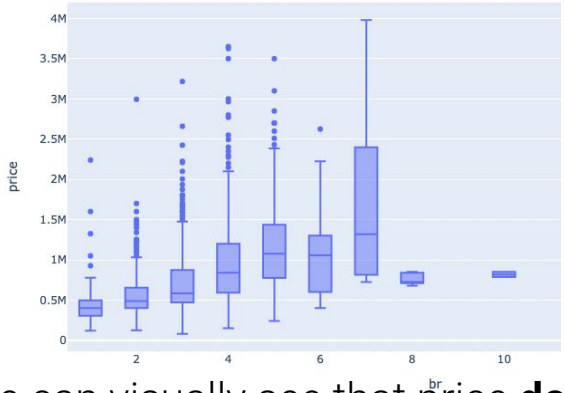
**How can we use this data to build a model to accurately predict the price of a house?**

## Step 2: Visualization

Notice the issue with overplotting due to discrete or categorical data:




Soln: Use boxplots (or violin plots) instead:




We can visually see that price **does increase** with the number of bedrooms

# The Modeling Process: Example with Housing Data

Consider a dataset of houses sold in the Bay Area in 2006

`date` : Date the house sold

`price` : Price the house sold for

`city` : In this case we consider 4 cities: Richmond, Berkeley, Piedmont and a combination of 3 cities Lafayette, Moraga and Morinda that we name Lamorinda

`br` : Number of bedrooms

`lsqft` : Square feet of the lot

`bsqft` : Square feet of the house

**How can we use this data to build a model to accurately predict the price of a house?**

## Step 3: Create v1 of a model

### First Version of Model: Using the predictor with the highest correlation to price

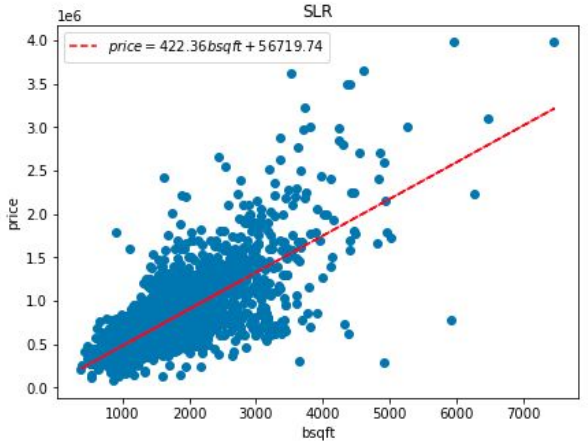
Let's create a model of the form:

$$\text{price} = \theta_0 + \theta_1 \text{bsqft}$$

```
import sklearn.linear_model as lm

X1 = sfh[['bsqft']]
y = sfh['price']

model1= lm.LinearRegression().fit(X1, y)
```

**Model Coefficient**

| Feature | |
|---|---|
| bsqft | 422.361285 |
| intercept | 56719.735791 |

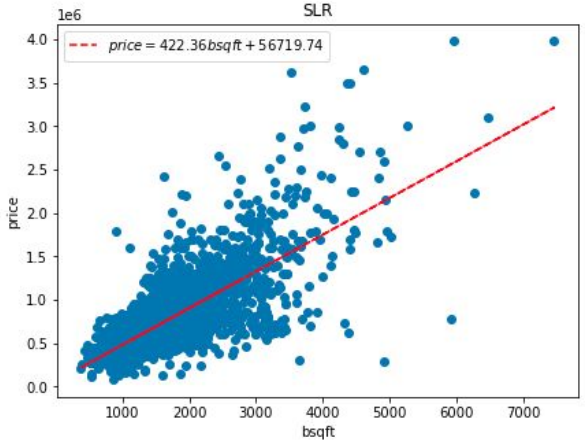


# Analyzing Model: Example with Housing Data

## First Version of Model: Using the predictor with the highest correlation to price

Let's create a model of the form:

$$\text{price} = \theta_0 + \theta_1 \text{bsqft}$$




**Calculate RMSE**

Analyze residuals

Multiple R^2

```
y=sfh['price']

y_hat = model1.predict(X1)

print(f"The RMSE of the model is {np.sqrt(np.mean((y-y_hat)**2))}")
```

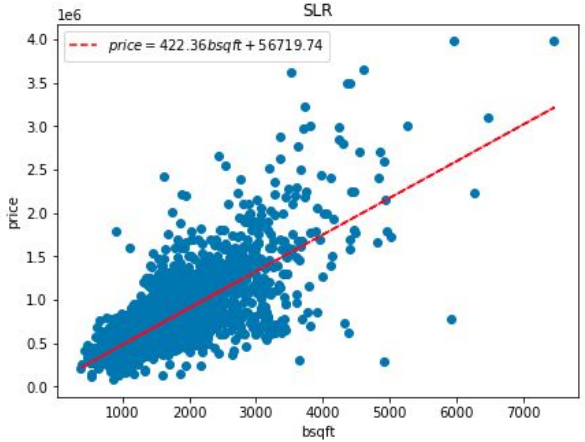
```
The RMSE of the model is 280574.2886385459
```

# Analyzing Model: Example with Housing Data

## First Version of Model: Using the predictor with the highest correlation to price

Let's create a model of the form:

$$\text{price} = \theta_0 + \theta_1 \text{bsqft}$$




**Analyze residuals**

Recall:

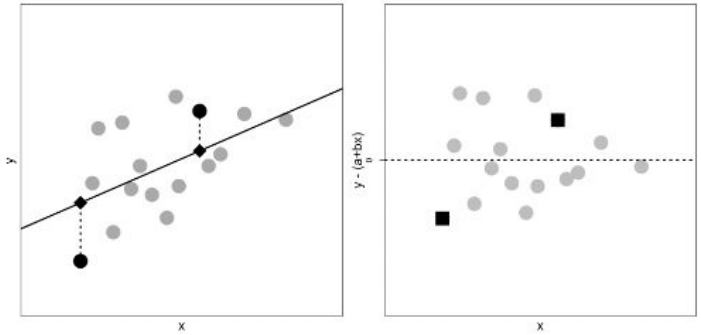


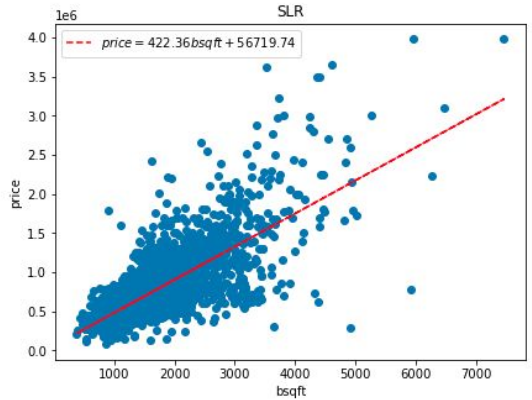

*Fig. 15.1* On the left is a scatter plot of $(x_i, y_i)$ pairs and a line that we use to estimate $y$ from $x$. Two specific points are represented by squares and their estimates by diamonds. On the right is a scatter plot of the errors: $y_i - (\theta_0 + \theta_1 x_i)$

# Analyzing Model: Example with Housing Data

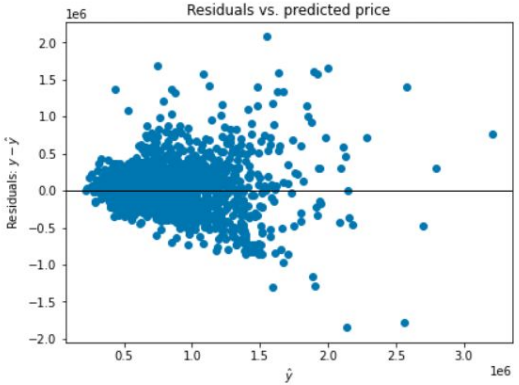
## First Version of Model: Using the predictor with the highest correlation to price

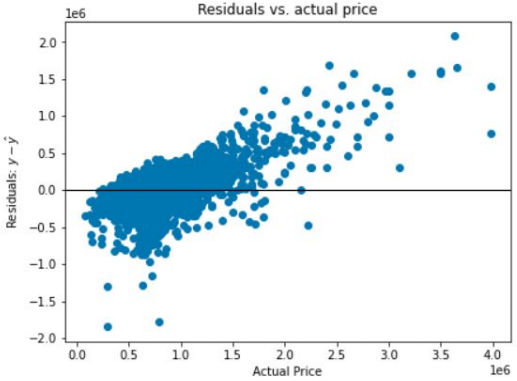
Let's create a model of the form:

$$\text{price} = \theta_0 + \theta_1 \text{bsqft}$$




**Analyze residuals**



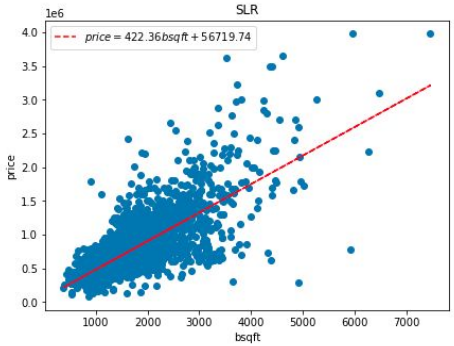

I). overestimating
II). underestimating

Poll: A) I, II B)II, I C), I, I D) II, II

Plotting residuals vs the actual price helps us see that our model is _____ the price of lower priced homes and

_____the price of higher priced homes.

# Analyzing Model: Example with Housing Data




## First Version of Model: Using the predictor with the highest correlation to price

Let's create a model of the form:

$$\text{price} = \theta_0 + \theta_1 \text{bsqft}$$

## Multiple R^2

We define the **multiple R²** value as the **proportion of variance** or our **fitted values** (predictions) $\hat{y}$ to our true values $y$.

$$R^2 = \frac{\text{variance of fitted values}}{\text{variance of } y} = \frac{\sigma_{\hat{y}}^2}{\sigma_y^2}$$

Also called the **correlation of determination**.

$R^2$ ranges from 0 to 1 and is effectively "the proportion of variance that the **model explains**."

```
r2_m1 = np.var(y_hat) / np.var(sfh["price"])

print('Multiple R^2 for model1:  ', r2_m1)
```

```
Multiple R^2 for model1:   0.570811414698371
```

```
#Built-in function to calculate this:
model1.score(X1, y)
```

```
0.5708114146983712
```

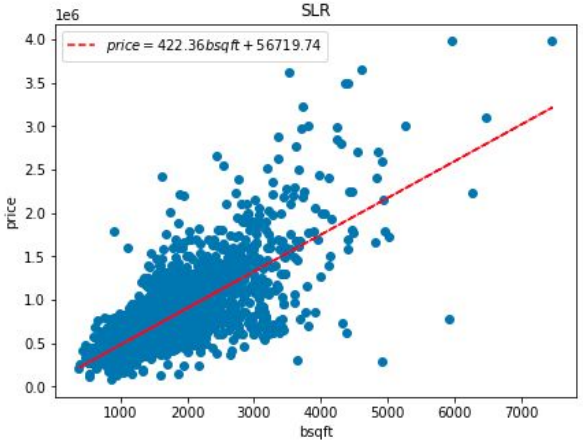
For OLS with an intercept term (e.g. $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p$), $R^2 = [r(y, \hat{y})]^2$ is equal to the square of correlation between $y, \hat{y}$.

- For SLR, $R^2 = r^2$, the correlation between $x, y$.

```
sfh.corr()
```

| | price | br | lsqft | bsqft |
|---|---|---|---|---|
| **price** | 1.000000 | 0.435298 | 0.589539 | 0.755521 |
| **br** | 0.435298 | 1.000000 | 0.270283 | 0.669279 |
| **lsqft** | 0.589539 | 0.270283 | 1.000000 | 0.433104 |
| **bsqft** | 0.755521 | 0.669279 | 0.433104 | 1.000000 |

# Analyzing Model: Example with Housing Data

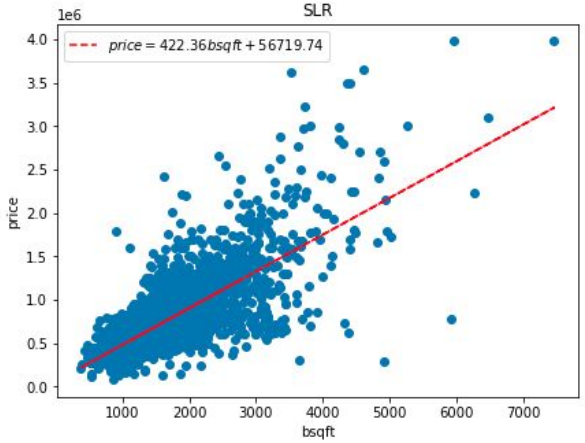



Improving the model

The relationship does look roughly linear, but the very large and expensive houses are far from the center of the distribution and can overly influence the model.

Let's try a log transformation of both variables to make the distributions of price and size more symmetric.

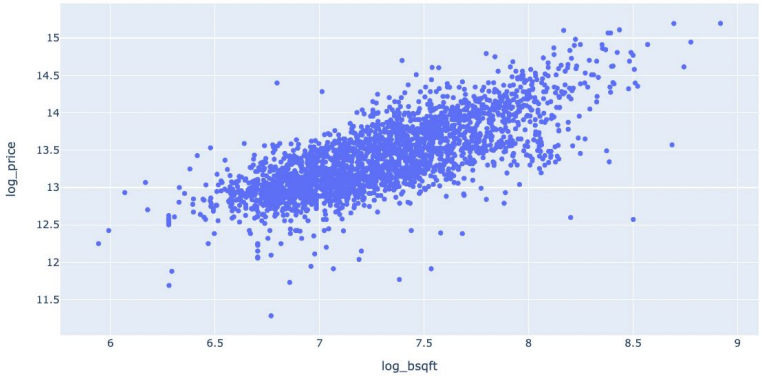
# Improving The Model: Feature Engineering




The relationship does look roughly linear, but the very large and expensive houses are far from the center of the distribution and can overly influence the model.

Let's try a ***log transformation of both variables*** to make the distributions of price and size more symmetric.

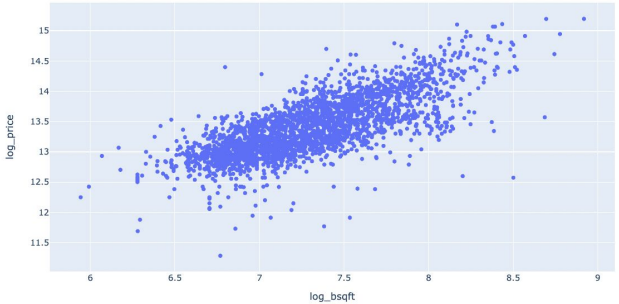
$$\log(\text{price}) = \theta_0 + \theta_1 \log(\text{bsqft})$$

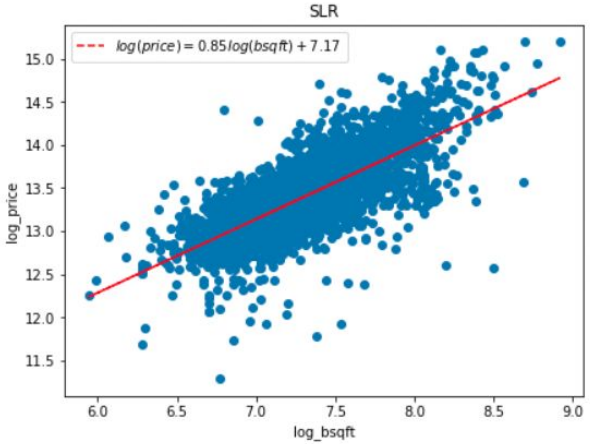


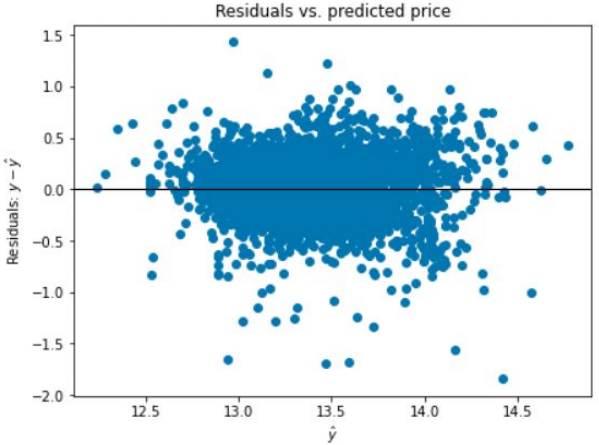
# Improving The Model: Feature Engineering

Let's try a ***log transformation of both variables*** to make the distributions of price and size more symmetric.

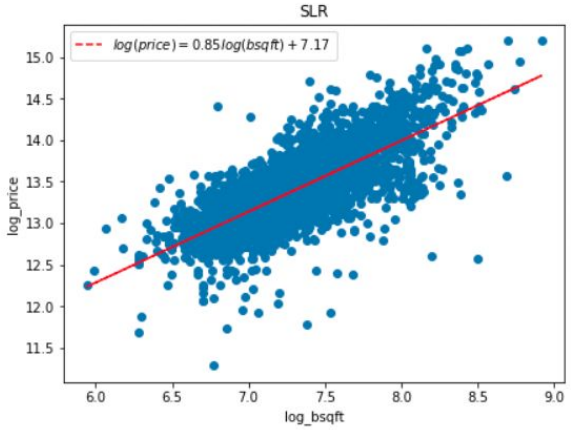
$$\log(\text{price}) = \theta_0 + \theta_1 \log(\text{bsqft})$$

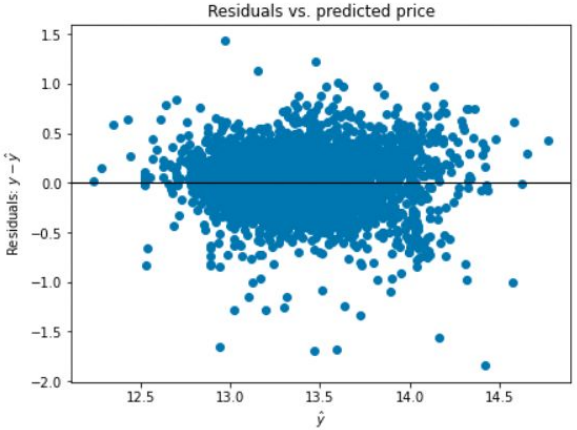


# Improving The Model: Adding More Features

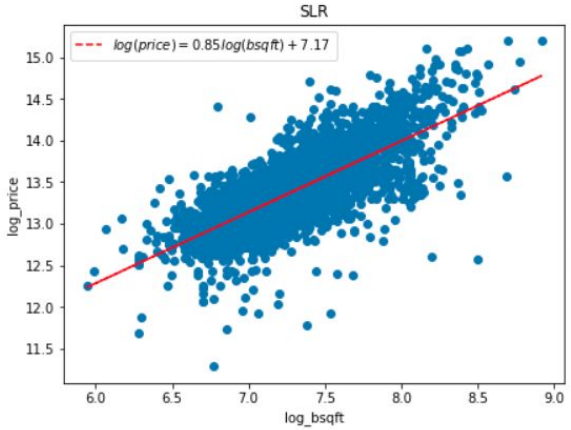


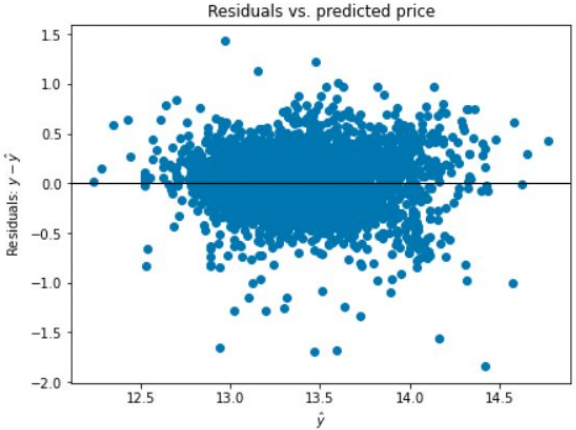




Choosing new features to add:

- We can plot residuals from the fitted model against a NEW variable (that is not in the model)
  - If we see patterns, that indicates we might want to include this additional feature or a transformation of it

# Improving The Model: Adding More Features



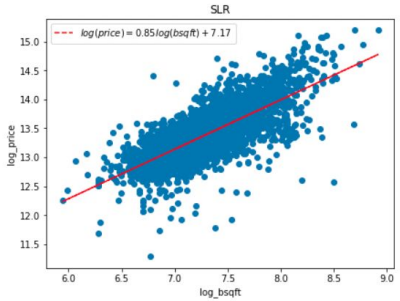




Choosing new features to add:

- We can plot residuals from the fitted model against a NEW variable (that is not in the model)
  - If we see patterns, that indicates we might want to include this additional feature or a transformation of it

Hard to see because of overplotting - switch to boxplots!

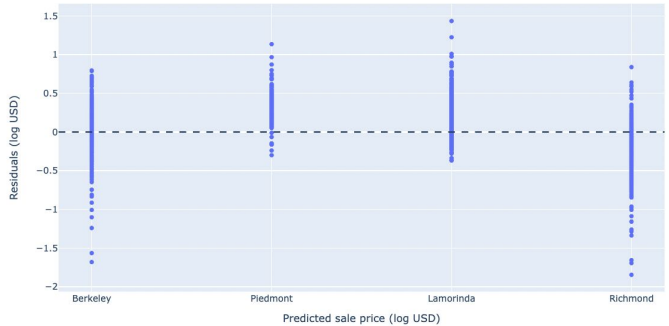
# Improving The Model: Adding More Features





Choosing new features to add:

- We can plot residuals from the fitted model against a NEW variable (that is not in the model)
  - If we see patterns, that indicates we might want to include this additional feature or a transformation of it

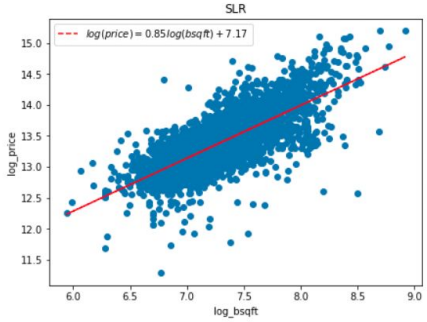


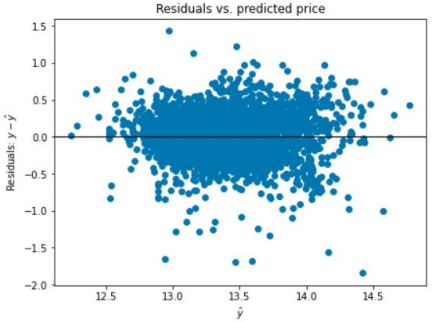

Hard to see patterns because of overplotting - switch to boxplots!

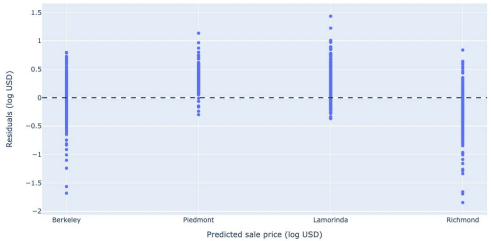
# Improving The Model: Adding More Features

Choosing new features to add:

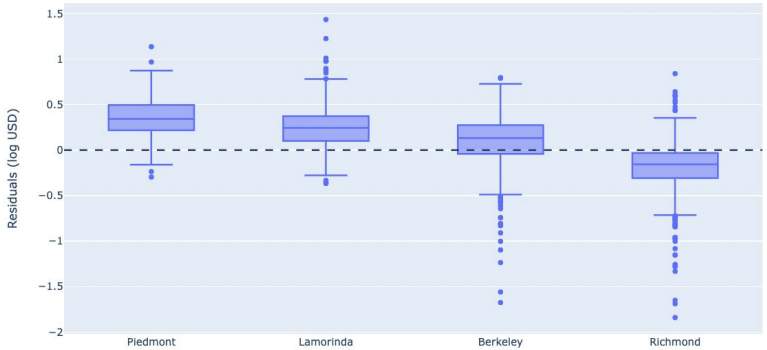
- We can plot residuals from the fitted model against a NEW variable (that is not in the model)
  - If we see patterns, that indicates we might want to include this additional feature or a transformation of it




Hard to see patterns because of overplotting - switch to boxplots!




Ideally, the median of each city's box plot lines up with 0 on the y-axis.

- Instead, more than 75% of the houses sold in Piedmont have positive errors, meaning the actual sale price is above the predicted value.
- At the other extreme, more than 75% of sale prices in Richmond fall below their predicted values.

These patterns suggest that we should include city in the model.

# One Hot Encoding

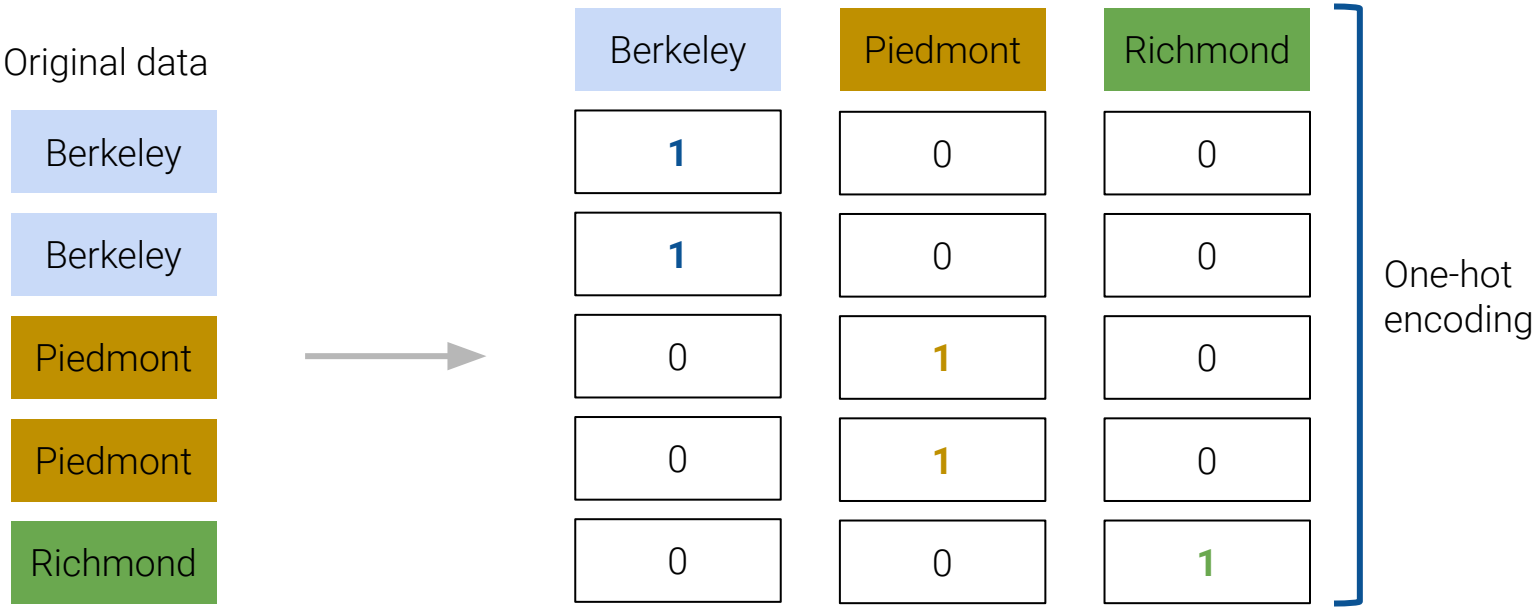
Gradient Descent in Higher Dimensions
Mini-Batch and Stochastic Gradient Descent
Feature Engineering
**One-Hot Encoding**
Higher-Order Polynomial Example
Overfitting
Variance and Training Error
[Extra] Convexity
[Extra] Deciding Overfitting

# One-hot Encoding

One-hot encoding is a feature engineering technique to transform qualitative 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

Use sklearn's `OneHotEncoder`! (documentation)



# Regression Using the One-Hot Encoding

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

```
from sklearn.preprocessing import OneHotEncoder

# Initialize a OneHotEncoder object
ohe = OneHotEncoder()

# Fit the encoder
ohe.fit(sfh[["city"]])

# Use the encoder to transform the raw "day" feature and put in a new dataframe

encoded_city_df = pd.DataFrame(ohe.transform(sfh[['city']]).todense(),
                               columns=ohe.get_feature_names_out(),
                               index = sfh.index)
```

| city | city_Berkeley | city_Lamorinda | city_Piedmont | city_Richmond |
|---|---|---|---|---|
| Richmond | 0.0 | 0.0 | 0.0 | 1.0 |
| Berkeley | 1.0 | 0.0 | 0.0 | 0.0 |
| Berkeley | 1.0 | 0.0 | 0.0 | 0.0 |
| Berkeley | 1.0 | 0.0 | 0.0 | 0.0 |
| Berkeley | 1.0 | 0.0 | 0.0 | 0.0 |
| ... | ... | ... | ... | ... |
| Berkeley | 1.0 | 0.0 | 0.0 | 0.0 |
| Piedmont | 0.0 | 0.0 | 1.0 | 0.0 |
| Berkeley | 1.0 | 0.0 | 0.0 | 0.0 |
| Piedmont | 0.0 | 0.0 | 1.0 | 0.0 |
| Piedmont | 0.0 | 0.0 | 1.0 | 0.0 |

Raw feature

One-hot encoded features

```
X = X_raw.join(encoded_city_df).drop(columns="city")
X.head()
```

| | log_bsqft | city_Berkeley | city_Lamorinda | city_Piedmont | city_Richmond |
|---|---|---|---|---|---|
| 0 | 7.020191 | 1.0 | 0.0 | 0.0 | 0.0 |
| 1 | 7.239933 | 1.0 | 0.0 | 0.0 | 0.0 |
| 2 | 7.632886 | 1.0 | 0.0 | 0.0 | 0.0 |
| 3 | 6.817831 | 1.0 | 0.0 | 0.0 | 0.0 |
| 4 | 6.845880 | 1.0 | 0.0 | 0.0 | 0.0 |

# Regression Using the One-Hot Encoding

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

Now, we can use `sklearn`'s `LinearRegression` class to fit a model to this design matrix.

We're fitting a model of this form:

$$log(price) = \theta_1 log(bsqft) + \theta_2(Berkeley) + \theta_3(Lamorinda) + \theta_4(Piedmont) + \theta_5(Richmond)$$

Notice, this is equivalent to fitting 4 models with the same slope, but the intercept term depending on city:

$$log(price) = \theta_1 log(bsqft) + \theta_2 \quad \text{(for houses in Berkeley)}$$

$$log(price) = \theta_1 log(bsqft) + \theta_3 \quad \text{(for houses in Lamorinda)}$$

$$log(price) = \theta_1 log(bsqft) + \theta_4 \quad \text{(for houses in Piedmont)}$$

$$log(price) = \theta_1 log(bsqft) + \theta_5 \quad \text{(for houses in Richmond)}$$

# Regression Using the One-Hot Encoding

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

Now, we can use `sklearn`'s `LinearRegression` class to fit a model to this design matrix.

We're fitting a model of this form:

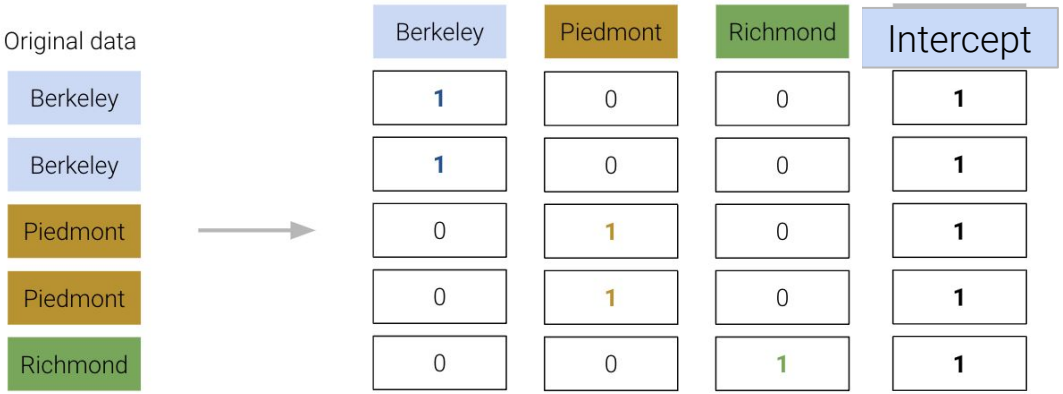
$$log(price) = \theta_1 log(bsqft) + \theta_2(Berkeley) + \theta_3(Lamorinda) + \theta_4(Piedmont) + \theta_5(Richmond)$$

```
y = sfh["log_price"]

ohe_model = lm.LinearRegression(fit_intercept=False)

#Since we are using one-hot encoding, tell sklearn to not add an additional bias column.
ohe_model.fit(X, y)

pd.DataFrame({"Feature":X.columns, "Model Coefficient":ohe_model.coef_}).set_index("Feature")
```

| Feature | Model Coefficient |
|---|---|
| log_bsqft | 0.596437 |
| city_Berkeley | 9.149616 |
| city_Lamorinda | 9.361655 |
| city_Piedmont | 9.484581 |
| city_Richmond | 8.811055 |

# One-hot Encode Wisely!

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




The intercept column is a linear combination of the OHE columns

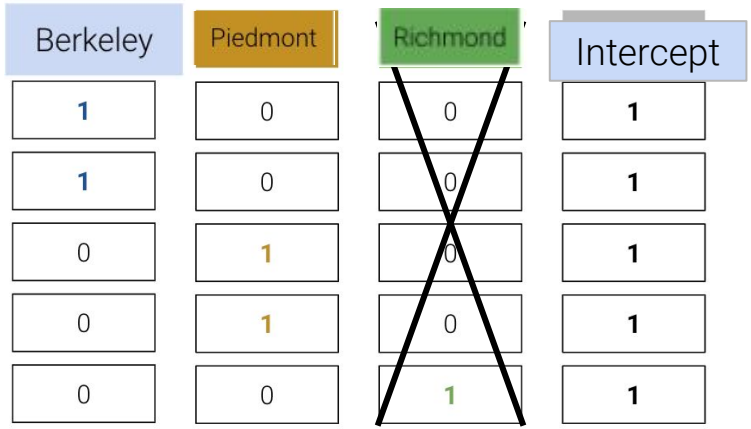
If we also include an intercept column in the design matrix, there will be linear dependence in the model. $\mathbb{X}^\top\mathbb{X}$ is not invertible, and our OLS estimate $\hat{\theta} = (\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^\top\mathbb{Y}$ fails.

How to resolve? Do not include an intercept term in your model

# 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:




or


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

# Pitfalls of Feature Engineering

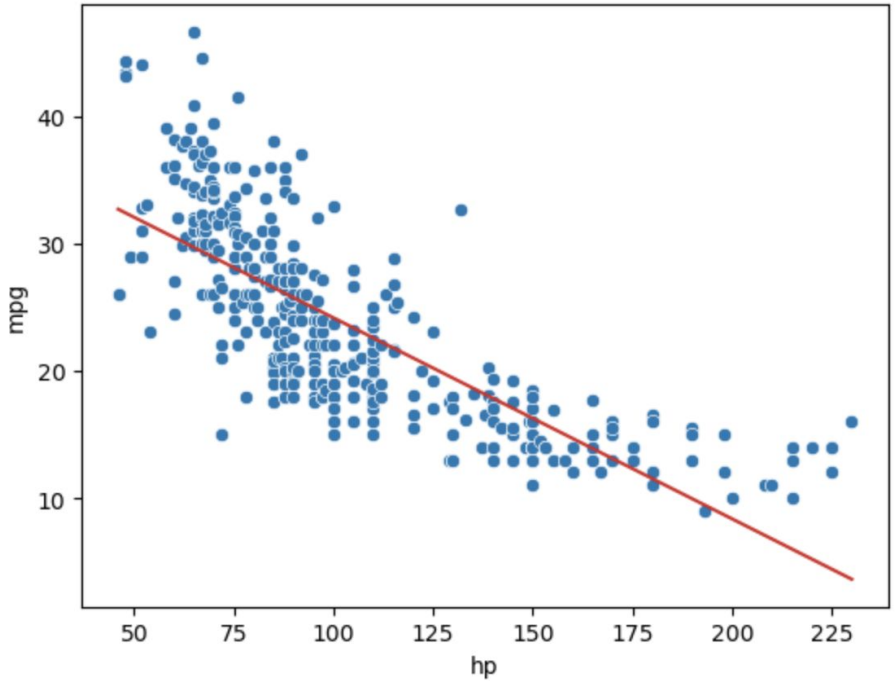
- Redundant features.
  - When one feature is a linear combination of the others, there isn't a unique solution for our parameter vector theta.
- Too many features.
  - When there are more features than datapoints, there isn't a unique solution for our parameter vector theta.
- Overfitting.
  - When we use too many complicated features, our model may fit to the "noise" in our collected data, and may not generalize to unseen data as well.

# Polynomial Features

- `sklearn`
- Feature Engineering
- One-Hot Encoding
- **Polynomial Features**
- Complexity and 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

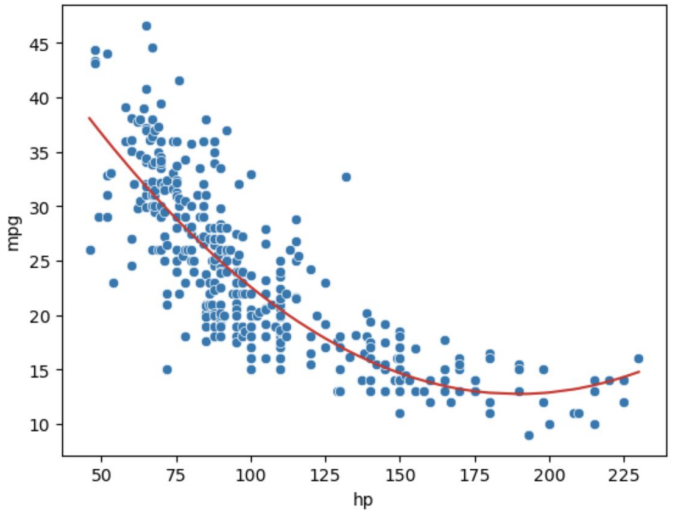
Solution: incorporate a non-linear feature!

# 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 $\theta$



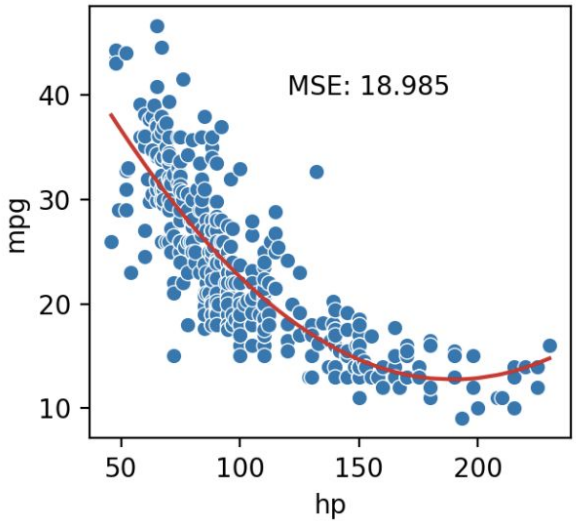

Degree of model: 2
MSE: 18.98

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

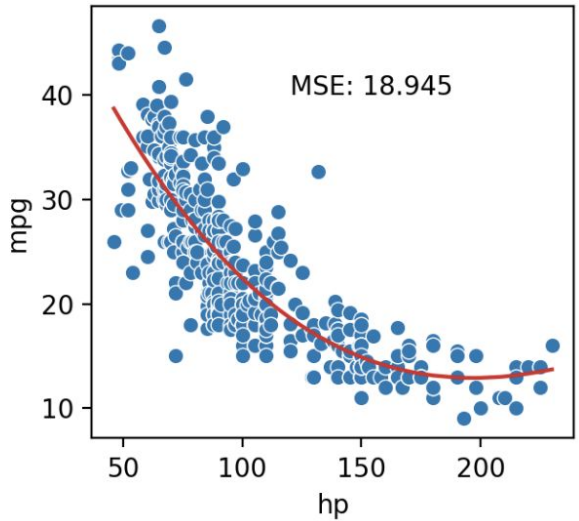
# Polynomial Features

What if we add more polynomial features?

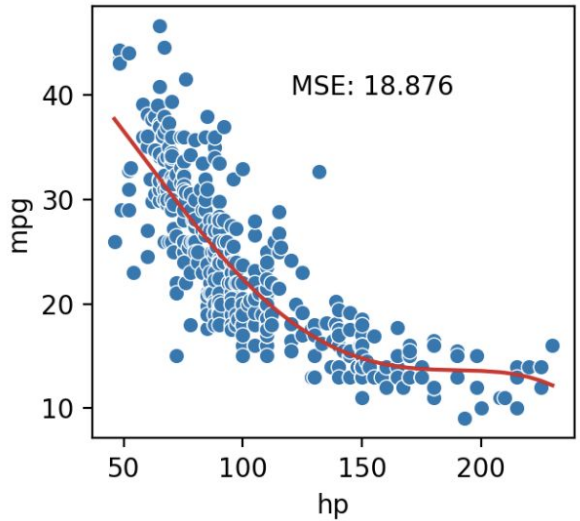
$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2)$




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




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

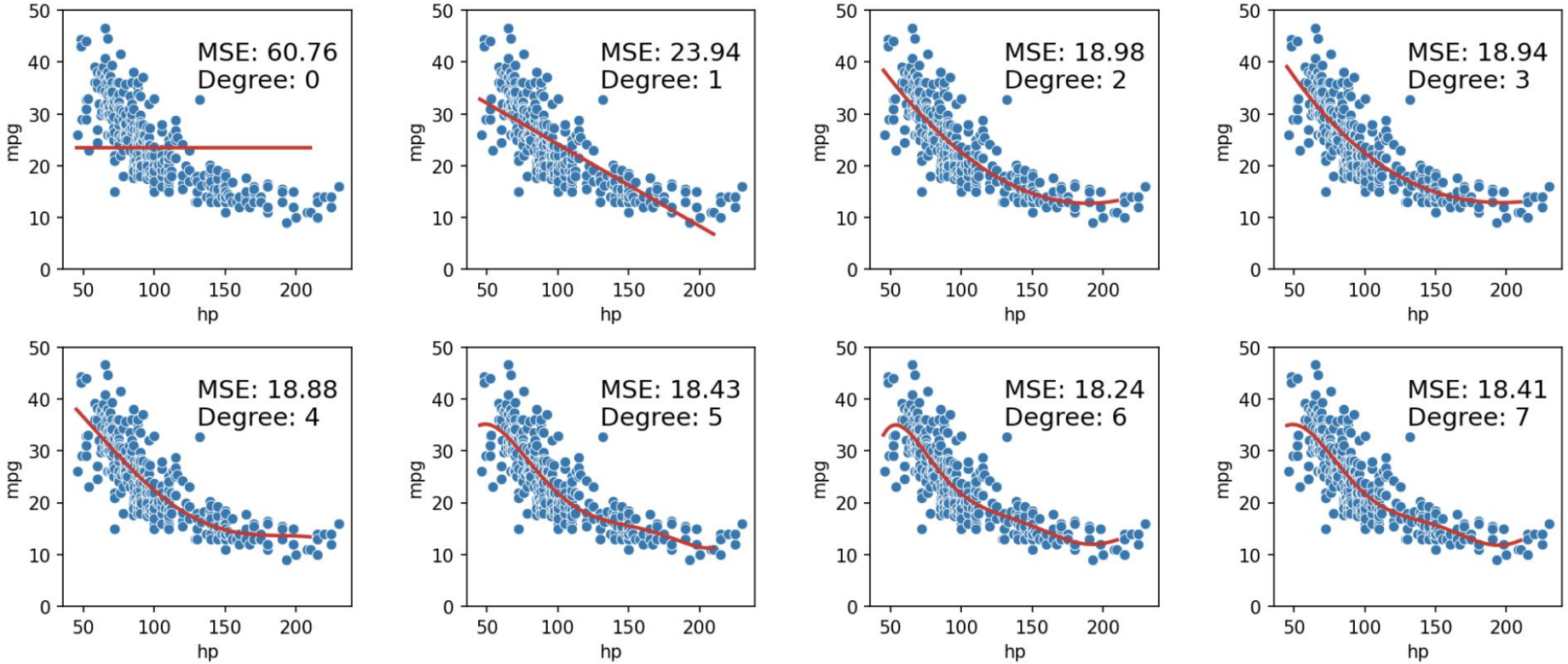



MSE continues to decrease with each additional polynomial term

# Complexity and Overfitting

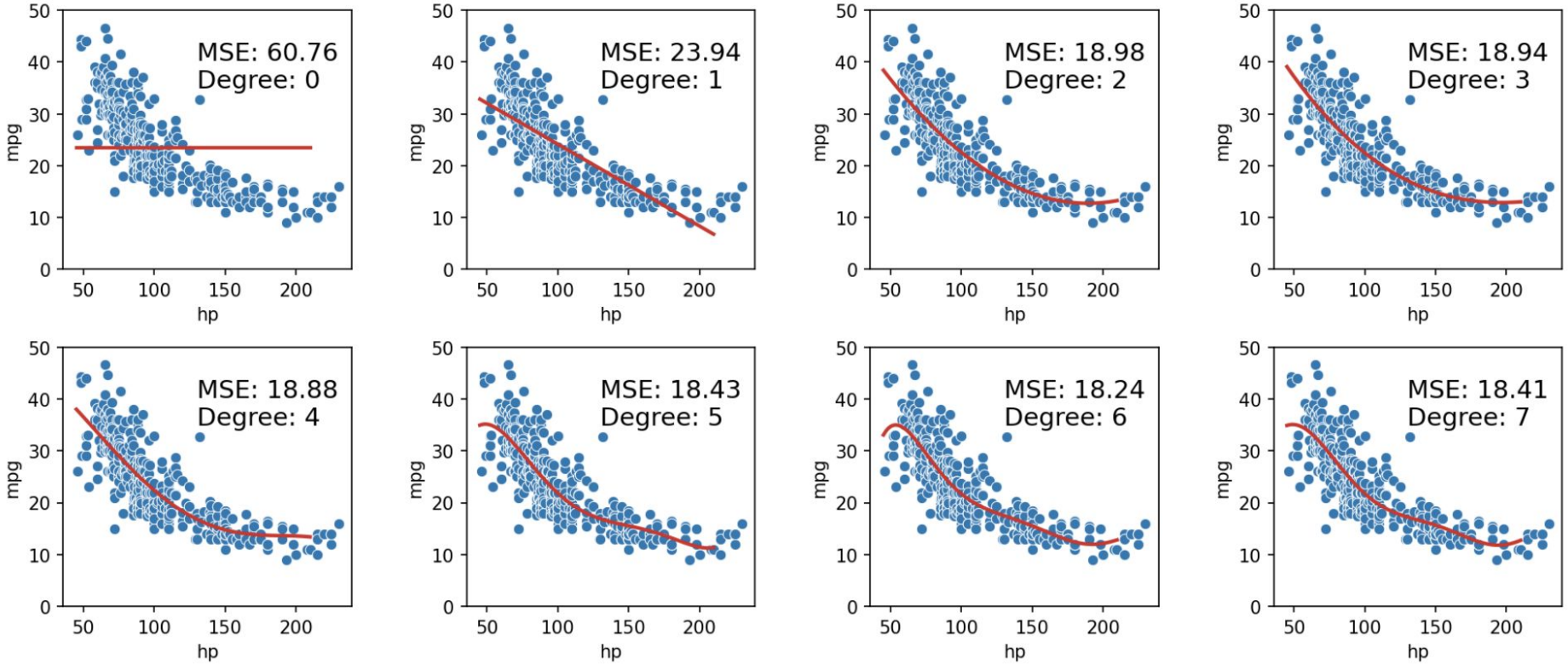
- `sklearn`
- Feature Engineering
- One-Hot Encoding
- Polynomial Features
- **Complexity and Overfitting**

# How Far Can We Take This?



# How Far Can We Take This?

## Poll: Which higher-order polynomial model do you think fits best?

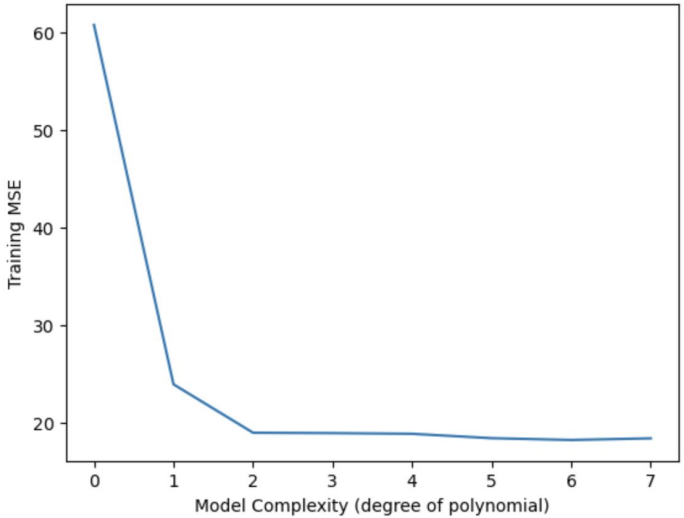


# Model Complexity

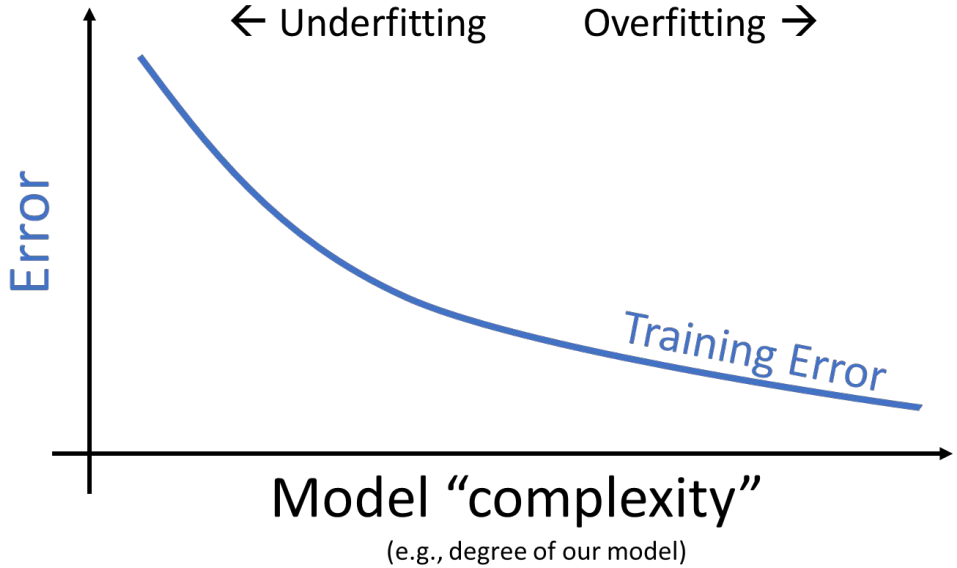
As we continue to add more and more polynomial features, the MSE continues to decrease

Equivalently: as the **model complexity** increases, its *training error* decreases

Our experiment using `vehicles`




General trend for an arbitrary dataset

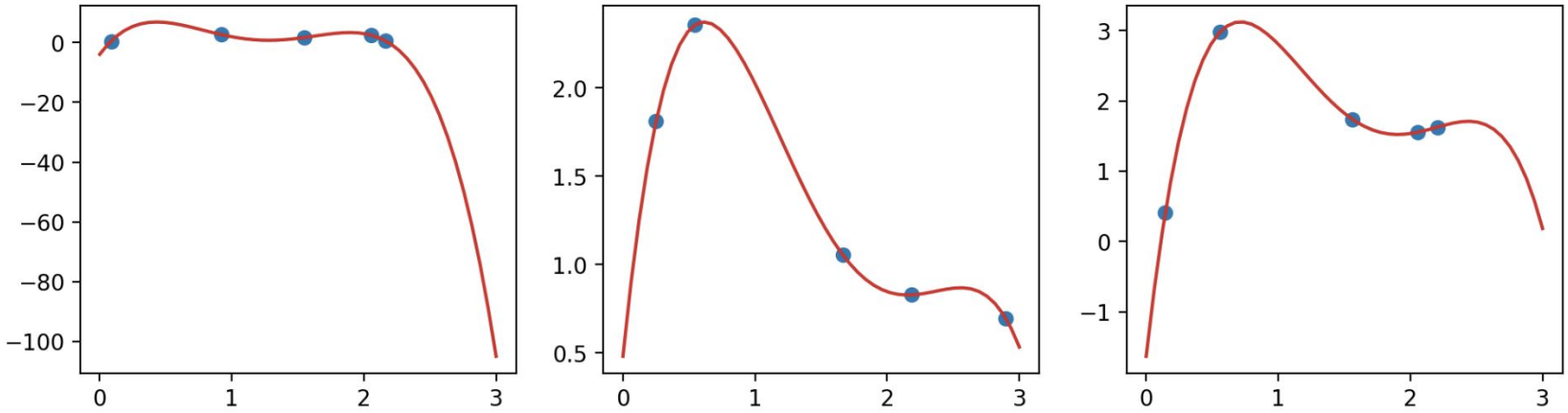



Seems like a good deal?

# An Extreme Example: Perfect Polynomial Fits

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

For example, there always exists a degree-4 polynomial curve that can perfectly model a dataset of 5 datapoints



# Model Performance on Unseen Data

Our `vehicle` models from before considered a somewhat artificial scenario – we trained the models on the *entire* dataset, then evaluated their ability to make predictions on this same dataset

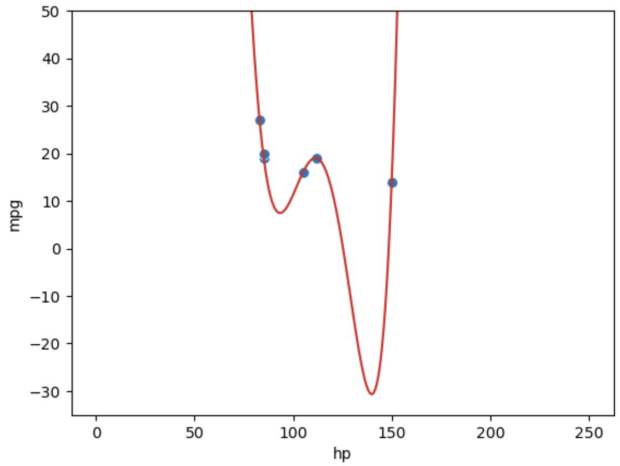
More realistic situation: we train the model on a *sample* from the population, then use it to make predictions on data it didn't encounter during training

# Model Performance on Unseen Data

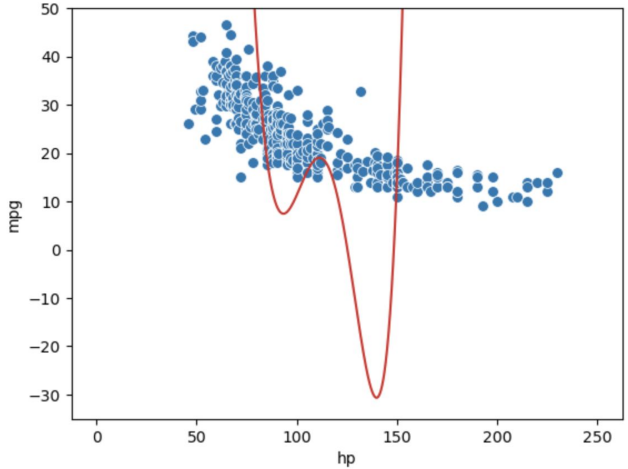
New (more realistic) example:

- We are given a training dataset of just 6 datapoints
- We want to train a model to then make predictions on a *different* set of points

We may be tempted to make a highly complex model (eg degree 5)




Complex model makes perfect predictions on the training data…



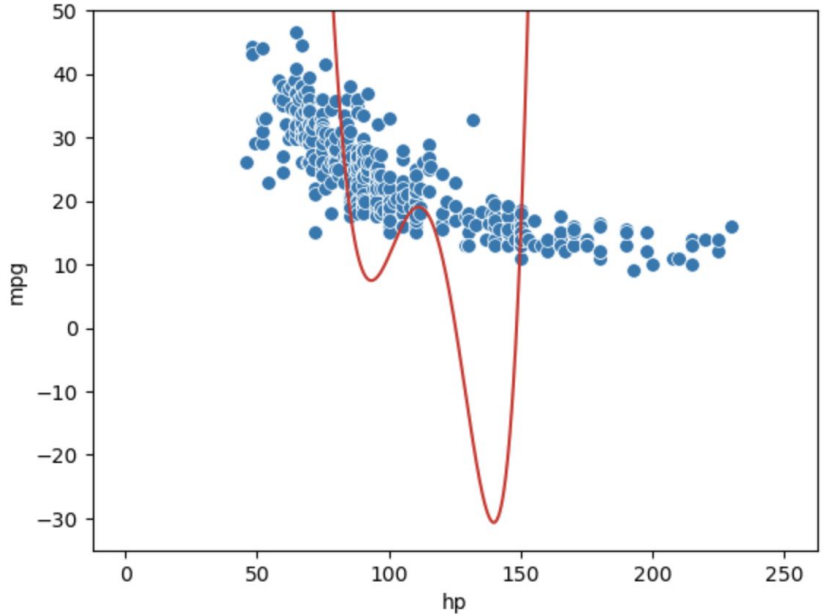

…but performs *horribly* on the rest of the population!

# Model Performance on Unseen Data

What went wrong?

- The complex model **overfit** to the training data – it essentially "memorized" these 6 training points
- The overfitted model does not **generalize** well to data it did not encounter during training

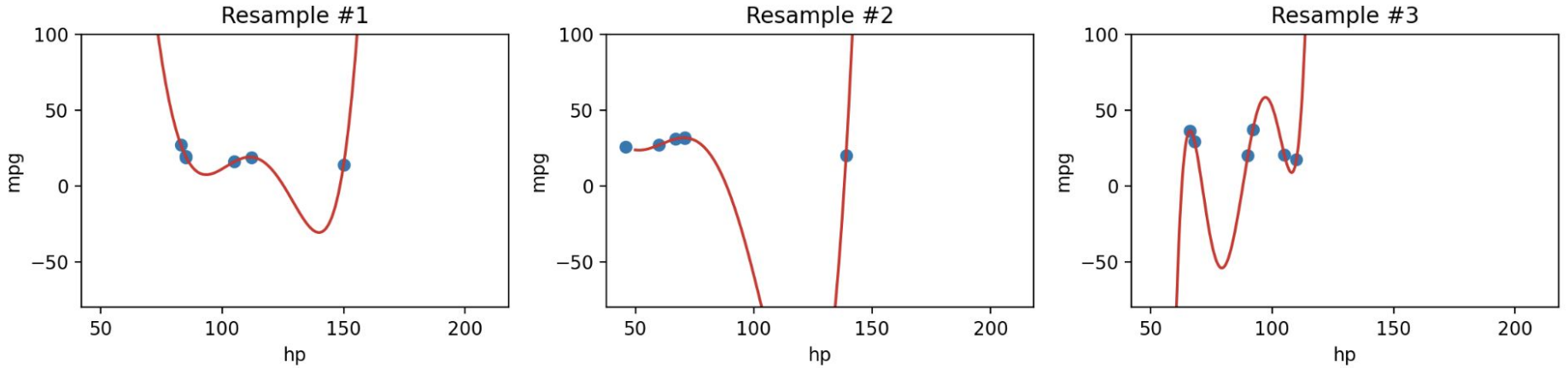
This is a problem: we want models that are generalizable to "unseen" data



# Model Variance

Complex models are sensitive to the specific dataset used to train them – they have high **variance**, because they will *vary* depending on what datapoints are used for training them

Our degree-5 model varies erratically when we fit it to different samples of 6 points from `vehicles`

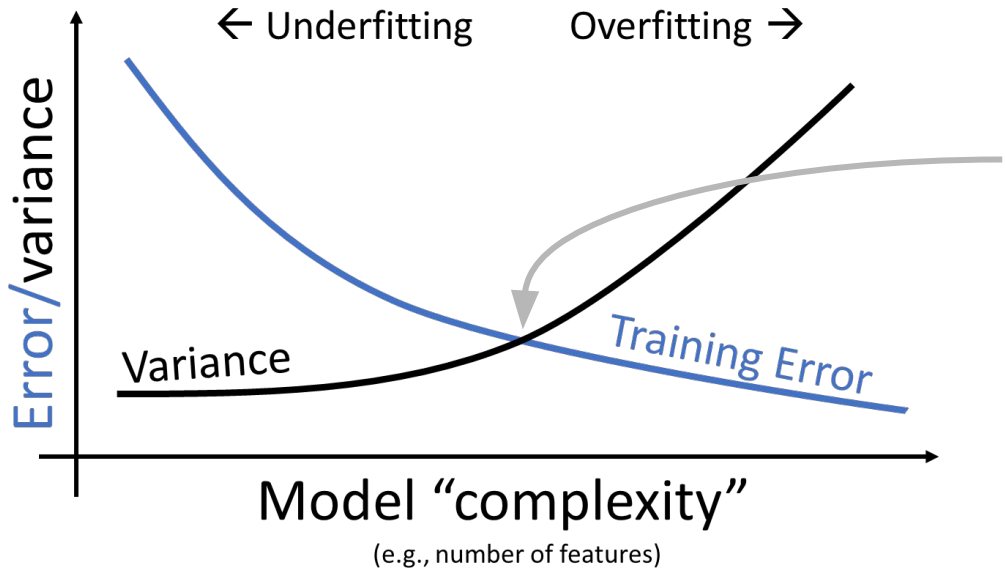



In Machine Learning, this **sensitivity** to data is known as **model variance**.

# Error, Variance, and Complexity

We face a dilemma:

- We know that we can **decrease training error** by increasing model complexity
- However, models that are *too* complex start to overfit and do not generalize well – their **high variance** means they can't be reapplied to new datasets




Our goal: find this "sweet spot"

Stay tuned for future lectures covering this!

2327602

# Appendix: Geometric Derivation

OLS Problem Formulation

- Multiple Linear Regression Model
- Mean Squared Error

**Geometric Derivation**

- Lin Alg Review: Orthogonality, Span
- Least Squares Estimate Proof

# Vector Notation

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p$$

$$= \theta_0 + \sum_{j=1}^{p} \theta_j x_j$$

To combine the two terms into one matrix operation, we can assume that there is an additional term $x_0 = 1$ in $\vec{x}$ and hence:

$$= x^T \theta \qquad \vec{x} = \begin{bmatrix} 1 \\ 0.4 \\ 0.8 \\ 1.5 \end{bmatrix} \quad \vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \qquad \vec{x}, \vec{\theta} \in \mathbb{R}^{(p+1)}$$

$$= \begin{bmatrix} 1 & 0.4 & 0.8 & 1.5 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \hat{y} \in \mathbb{R}$$

**NBA Data**

| | FG | AST | 3PA | PTS |
|---|---|---|---|---|
| 1 | 1.8 | 0.6 | 4.1 | 5.3 |
| 2 | 0.4 | 0.8 | 1.5 | 1.7 |
| 3 | 1.1 | 1.9 | 2.2 | 3.2 |
| 4 | 6.0 | 1.6 | 0.0 | 13.9 |
| 5 | 3.4 | 2.2 | 0.2 | 8.9 |
| 6 | 0.6 | 0.3 | 1.2 | 1.7 |

Rows correspond to individual players.

Note that:

$$\vec{x}^T \times \vec{\theta} = \vec{x}.\vec{\theta}$$

# Matrix Notation

To make predictions on all $n$ datapoints in our sample:

**Data**

| | FG | AST | 3PA | PTS |
|---|---|---|---|---|
| 1 | 1.8 | 0.6 | 4.1 | 5.3 |
| 2 | 0.4 | 0.8 | 1.5 | 1.7 |
| 3 | 1.1 | 1.9 | 2.2 | 3.2 |

$$\hat{y}_1 = x_1^T\theta \quad \text{where } x_1^T = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \end{bmatrix} \quad \text{Datapoint 1}$$

$$\hat{y}_2 = x_2^T\theta \quad \text{where } x_2^T = \begin{bmatrix} 1 & x_{21} & x_{22} & \dots & x_{2p} \end{bmatrix} \quad \text{Datapoint 2}$$

$$\vdots \qquad \vdots \qquad\qquad\qquad \vdots$$

$$\hat{y}_n = x_n^T\theta \quad \text{where } x_n^T = \begin{bmatrix} 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \quad \text{Datapoint n}$$

same $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$ for all preds

# Matrix Notation

To make predictions on all $n$ datapoints in our sample:

$$\begin{aligned}
\hat{y}_1 &= \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \end{bmatrix} \theta &= x_1^T \theta \\
\hat{y}_2 &= \begin{bmatrix} 1 & x_{21} & x_{22} & \dots & x_{2p} \end{bmatrix} \theta &= x_2^T \theta \\
&\vdots & & \vdots & \\
\hat{y}_n &= \begin{bmatrix} 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \theta &= x_n^T \theta
\end{aligned}$$

**n** row vectors, each with dimension **(p+1)**

**Data**

| | FG | AST | 3PA | PTS |
|---|---|---|---|---|
| 1 | 1.8 | 0.6 | 4.1 | 5.3 |
| 2 | 0.4 | 0.8 | 1.5 | 1.7 |
| 3 | 1.1 | 1.9 | 2.2 | 3.2 |

same $\theta =$ for all preds $\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

Expand out each data point's (transposed) input

# Matrix Notation

To make predictions on all $n$ datapoints in our sample:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ & & \vdots & & \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \theta$$

**n** row vectors, each with dimension **(p+1)**

**Data**

| | FG | AST | 3PA | PTS |
|---|---|---|---|---|
| 1 | 1.8 | 0.6 | 4.1 | 5.3 |
| 2 | 0.4 | 0.8 | 1.5 | 1.7 |
| 3 | 1.1 | 1.9 | 2.2 | 3.2 |

same $\theta$ for all preds $= \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

Vectorize predictions and parameters to encapsulate all n equations into a single matrix equation.

# Matrix Notation

To make predictions on all $n$ datapoints in our sample:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \mathbb{X} \theta$$

**Design matrix** with dimensions n x (p + 1)

**Data**

| | FG | AST | 3PA | PTS |
|---|---|---|---|---|
| 1 | 1.8 | 0.6 | 4.1 | 5.3 |
| 2 | 0.4 | 0.8 | 1.5 | 1.7 |
| 3 | 1.1 | 1.9 | 2.2 | 3.2 |

same $\theta =$ for all preds $\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$

# The Design Matrix $\mathbb{X}$

We can use linear algebra to represent our predictions of all $n$ data points at once.

One step in this process is to stack all of our input features together into a **design matrix**:

$$\mathbb{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ 1 & x_{31} & x_{32} & \dots & x_{3p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

What do the **rows** and **columns** of the design matrix represent in terms of the observed data?

Field Goals, Assists, 3-Point Attempts

| Bias | FG | AST | 3PA | PTS |
|---|---|---|---|---|
| 1 | 1.8 | 0.6 | 4.1 | 5.3 |
| 1 | 0.4 | 0.8 | 1.5 | 1.7 |
| 1 | 1.1 | 1.9 | 2.2 | 3.2 |
| 1 | 6.0 | 1.6 | 0.0 | 13.9 |
| 1 | 3.4 | 2.2 | 0.2 | 8.9 |
| ... | ... | ... | ... | ... |
| 1 | 4.0 | 0.8 | 0.0 | 11.5 |
| 1 | 3.1 | 0.9 | 0.0 | 7.8 |
| 1 | 3.6 | 1.1 | 0.0 | 8.9 |
| 1 | 3.4 | 0.8 | 0.0 | 8.5 |
| 1 | 3.8 | 1.5 | 0.0 | 9.4 |

Example design matrix
708 rows x (3+1) cols

# The Design Matrix $\mathbb{X}$

We can use linear algebra to represent our predictions of all $n$ data points at once.

One step in this process is to stack all of our input features together into a **design matrix**:

$$\mathbb{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ 1 & x_{31} & x_{32} & \dots & x_{3p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

A **row** corresponds to one **observation**, e.g., all (p+1) features for datapoint 3

A **column** corresponds to a **feature**, e.g. feature 1 for all n data points

Special all-ones feature often called the **bias/intercept**

| Bias | FG | AST | 3PA | PTS |
|---|---|---|---|---|
| 1 | 1.8 | 0.6 | 4.1 | 5.3 |
| 1 | 0.4 | 0.8 | 1.5 | 1.7 |
| 1 | 1.1 | 1.9 | 2.2 | 3.2 |
| 1 | 6.0 | 1.6 | 0.0 | 13.9 |
| 1 | 3.4 | 2.2 | 0.2 | 8.9 |
| ... | ... | ... | ... | ... |
| 1 | 4.0 | 0.8 | 0.0 | 11.5 |
| 1 | 3.1 | 0.9 | 0.0 | 7.8 |
| 1 | 3.6 | 1.1 | 0.0 | 8.9 |
| 1 | 3.4 | 0.8 | 0.0 | 8.5 |
| 1 | 3.8 | 1.5 | 0.0 | 9.4 |

Example design matrix
708 rows x (3+1) cols

# The Multiple Linear Regression Model using Matrix Notation

We can express our linear model on our entire dataset as follows:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ 1 & x_{31} & x_{32} & \dots & x_{3p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}$$

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

**Prediction vector** $\mathbb{R}^n$

**Design matrix** $\mathbb{R}^{n \times (p+1)}$

**Parameter vector** $\mathbb{R}^{(p+1)}$

Note that our **true output** is also a vector:

$$\mathbb{Y} \in \mathbb{R}^n$$

# [Linear Algebra] Vector Norms and the L2 Vector Norm

The **norm** of a vector is some measure of that vector's **size**.

- The two norms we need to know for Data 100 are the $L_1$ and $L_2$ norms (sound familiar?).
- Today, we focus on $L_2$ norm. We'll define the $L_1$ norm another day.

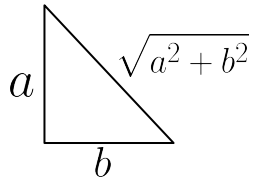
For the n-dimensional vector $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, the **L2 vector norm** is

$$||\vec{x}||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

# [Linear Algebra] The L2 Norm Is a Measure of Distance

$$||\vec{x}||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

In $\mathbb{R}^2$

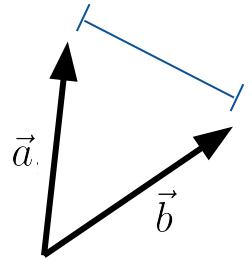



The L2 vector norm is a generalization of the Pythagorean theorem into ***n*** dimensions.

It can therefore be used as a measure of **distance** between two vectors.

- For n-dimensional vectors $\vec{a}, \vec{b}$ , their distance is $||\vec{a} - \vec{b}||_2$ .

In $\mathbb{R}^n$




Note: The square of the L2 norm of a vector is the sum of the squares of the vector's elements:

$$(||\vec{x}||_2)^2 = \sum_{i=1}^{n} x_i^2$$

Looks like Mean Squared Error!!

# Mean Squared Error with L2 Norms

We can rewrite mean squared error as a squared L2 norm:

$$R(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
$$= \frac{1}{n}||\mathbb{Y} - \hat{\mathbb{Y}}||_2^2$$

With our linear model $\hat{\mathbb{Y}} = \mathbb{X}\theta$ :

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

# Ordinary Least Squares

The **least squares estimate** $\hat{\theta}$ is the parameter that **minimizes** the objective function $R(\theta)$:

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

How should we interpret the OLS problem?

**A.** Minimize the mean squared error for the linear model $\hat{\mathbb{Y}} = \mathbb{X}\theta$

**B.** Minimize the **distance** between true and predicted values $\mathbb{Y}$ and $\hat{\mathbb{Y}}$

**C.** Minimize the **length** of the residual vector, $e = \mathbb{Y} - \hat{\mathbb{Y}} = \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ \vdots \\ y_n - \hat{y}_n \end{bmatrix}$

**D.** All of the above

**E.** Something else

# Ordinary Least Squares

The **least squares estimate** $\hat{\theta}$ is the parameter that **minimizes** the objective function $R(\theta)$:

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

How should we interpret the OLS problem?

**A.** Minimize the mean squared error for the linear model $\hat{\mathbb{Y}} = \mathbb{X}\theta$

**B.** Minimize the **distance** between true and predicted values $\mathbb{Y}$ and $\hat{\mathbb{Y}}$

**C.** Minimize the **length** of the residual vector, $e = \mathbb{Y} - \hat{\mathbb{Y}} = \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ \vdots \\ y_n - \hat{y}_n \end{bmatrix}$ Important for today

**D.** All of the above

**E.** Something else

# Today's Goal: Ordinary Least Squares

1. Choose a model ✅

Multiple Linear Regression

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

2. Choose a loss function ✅

L2 Loss

Mean Squared Error (MSE)

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

**3. Fit the model**

Minimize average loss with ~~calculus~~ geometry

The calculus derivation requires matrix calculus (out of scope, but here's a link if you're interested).

Instead, we will derive $\hat{\theta}$ using a **geometric argument**.

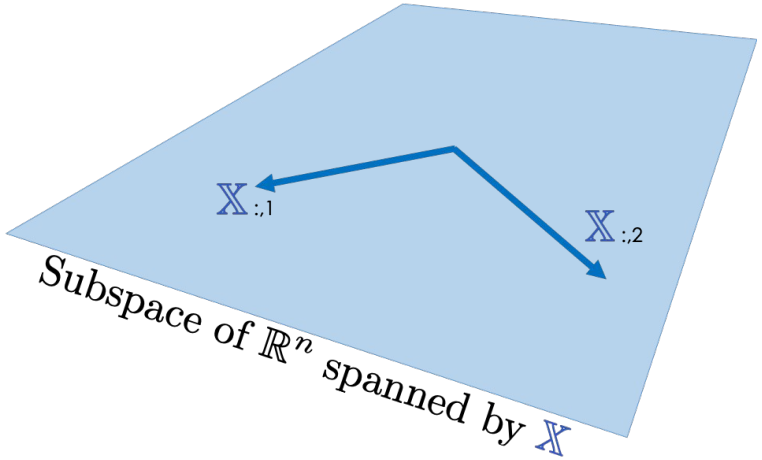
4. Evaluate model performance

Visualize,
~~Root MSE~~
Multiple $R^2$

# [Linear Algebra] Span

The set of all possible linear combinations of the columns of $\mathbb{X}$ is called the **span** of the columns of $\mathbb{X}$ (denoted $span(\mathbb{X})$), also called the **column space**.

- Intuitively, this is all of the vectors you can "reach" using the columns of $\mathbb{X}$.
- If each column of $\mathbb{X}$ has length $n$, $span(\mathbb{X})$ is a subspace of $\mathbb{R}^n$.



# A Linear Combination of Columns

$$\hat{\mathbb{Y}} = \mathbb{X}\,\theta$$

So far, we've thought of our model as horizontally stacked predictions per datapoint:

$$\underset{1}{\overset{n}{\begin{pmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{pmatrix}}} = \begin{bmatrix} — & x_1^T & — \\ — & x_2^T & — \\ & \vdots & \\ — & x_n^T & — \end{bmatrix} \underset{1}{\overset{p+1}{\begin{pmatrix} | \\ \theta \\ | \end{pmatrix}}}$$

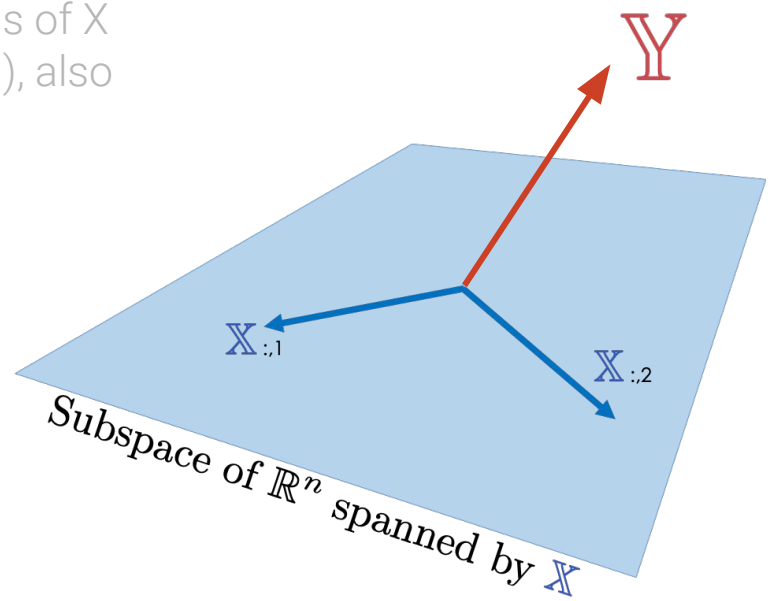
We can also think of $\hat{\mathbb{Y}}$ as a **linear combination of feature vectors**, scaled by **parameters**.

$$\underset{1}{\overset{n}{\begin{pmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{pmatrix}}} = \underset{p+1}{\overset{n}{\begin{pmatrix} | & | \\ \mathbb{X}_{:,1} & \mathbb{X}_{:,2} \\ | & | \end{pmatrix}}} \underset{1}{\overset{p+1}{\begin{pmatrix} | \\ \theta \\ | \end{pmatrix}}} = \theta_1 \begin{pmatrix} | \\ \mathbb{X}_{:,1} \\ | \end{pmatrix} + \theta_2 \begin{pmatrix} | \\ \mathbb{X}_{:,2} \\ | \end{pmatrix} + \ldots$$

# A Linear Combination of Columns

The set of all possible linear combinations of the columns of X is called the **span** of the columns of X (denoted $span(\mathbb{X})$), also called the **column space**.

- Intuitively, this is all of the vectors you can "reach" using the columns of X.
- If each column of X has length $n$, $span(\mathbb{X})$ is a subspace of $\mathbb{R}^n$.

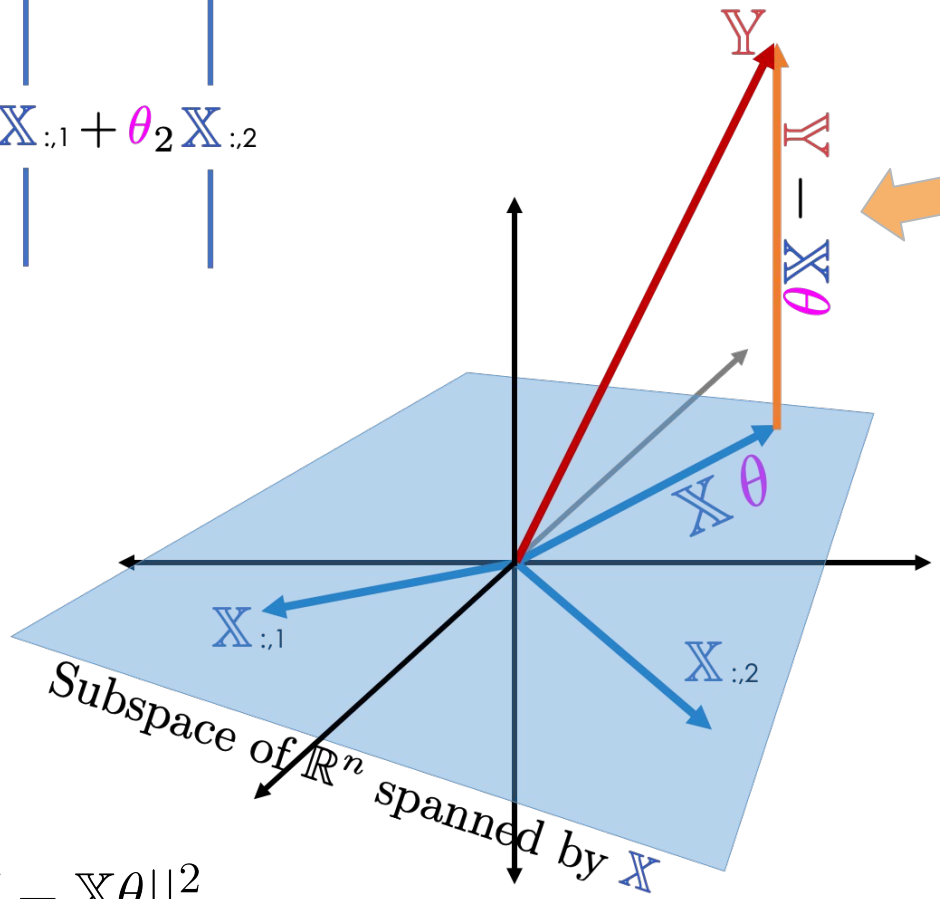



Our prediction $\hat{\mathbb{Y}} = \mathbb{X}\theta$ is a **linear combination** of the columns of $\mathbb{X}$. Therefore $\hat{\mathbb{Y}} \in span(\mathbb{X})$.

Interpret: Our linear prediction $\hat{\mathbb{Y}}$ will be in $span(\mathbb{X})$, even if the true values $\mathbb{Y}$ might not be.

Goal: Find the vector in $span(\mathbb{X})$ that is **closest** to $\mathbb{Y}$.

$$\begin{bmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{bmatrix}_{n \times 1} = \theta_1 \mathbb{X}_{:,1} + \theta_2 \mathbb{X}_{:,2}$$

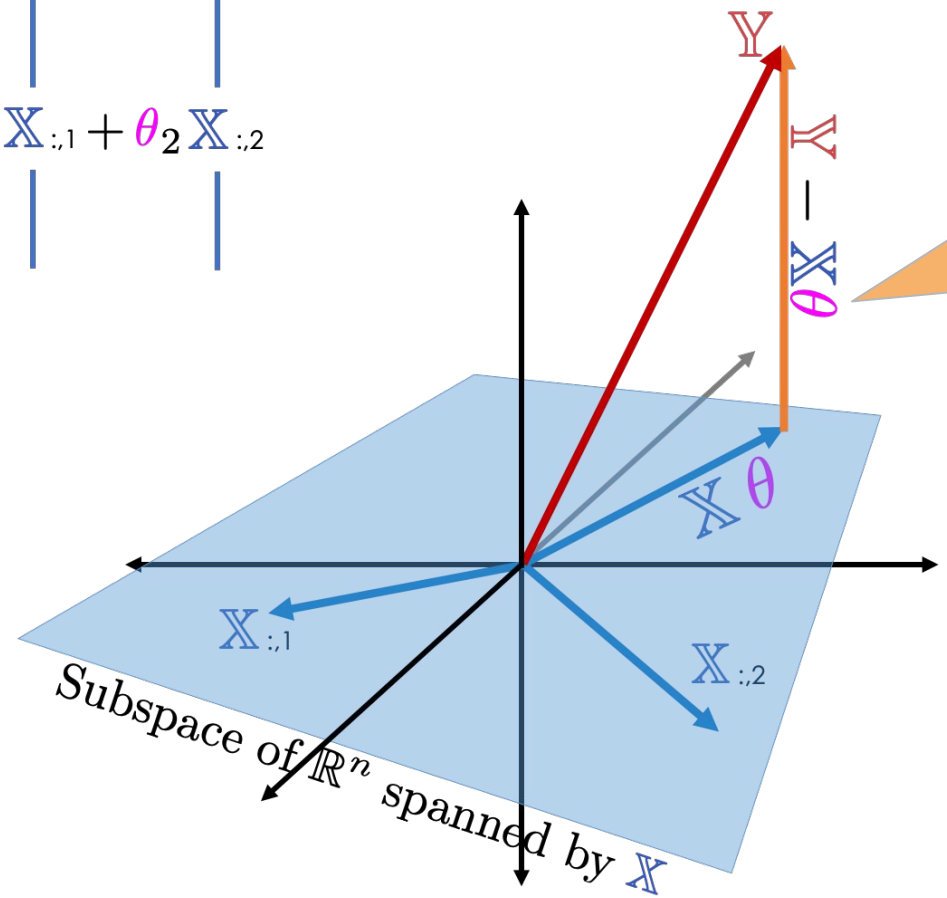



This is the residual vector, $e = \mathbb{Y} - \hat{\mathbb{Y}}$.

Goal:
Minimize the *L2* norm of the residual vector.
i.e., get the predictions $\hat{\mathbb{Y}}$ to be "as close" to our true $\mathbb{Y}$ values as possible.

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

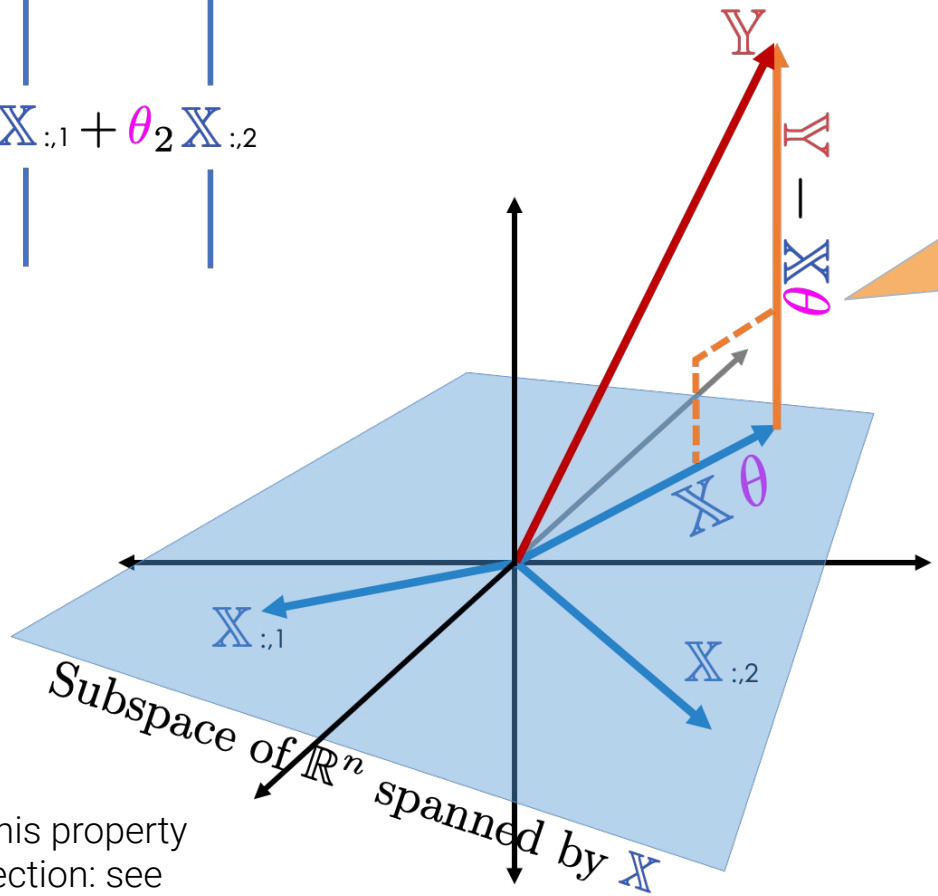
$$\begin{bmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{bmatrix}_{n \times 1} = \theta_1 \mathbb{X}_{:,1} + \theta_2 \mathbb{X}_{:,2}$$




How do we minimize this distance – the norm of the residual vector (squared)?

2327602

$$\begin{bmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{bmatrix}_{n \times 1} = \theta_1 \mathbb{X}_{:,1} + \theta_2 \mathbb{X}_{:,2} = \begin{bmatrix} | & | \\ \mathbb{X}_{:,1} & \mathbb{X}_{:,2} \\ | & | \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \mathbb{X}\theta$$

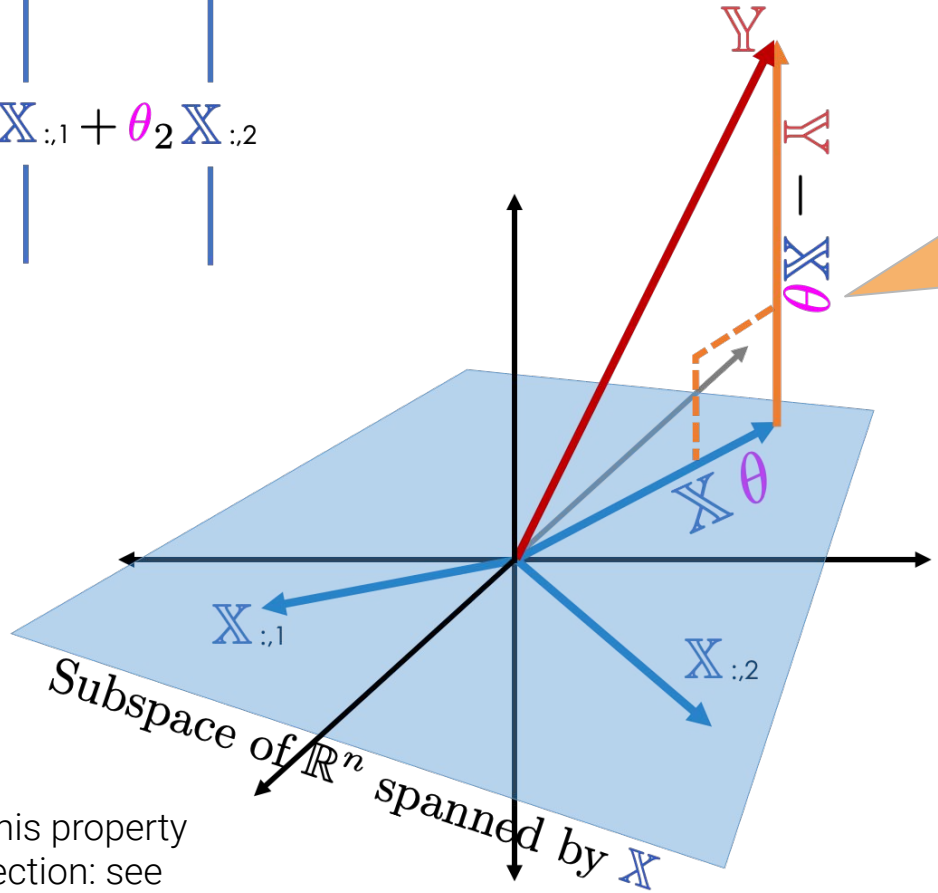



How do we minimize this distance – the norm of the residual vector (squared)?

The vector in $span(\mathbb{X})$ that is closest to $\mathbb{Y}$ is the **orthogonal projection** of $\mathbb{Y}$ onto $span(\mathbb{X})$.

We will not prove this property of orthogonal projection: see Khan Academy.

$$\begin{bmatrix} | \\ \hat{\mathbb{Y}} \\ | \end{bmatrix}_{n \times 1} = \theta_1 \mathbb{X}_{:,1} + \theta_2 \mathbb{X}_{:,2}$$




How do we minimize this distance – the norm of the residual vector (squared)?

The vector in $span(\mathbb{X})$ that is closest to $\mathbb{Y}$ is the **orthogonal projection** of $\mathbb{Y}$ onto $span(\mathbb{X})$.

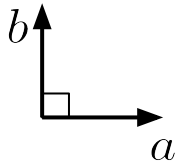
Thus, we should choose the $\theta$ that makes the residual vector **orthogonal** to $span(\mathbb{X})$.

We will not prove this property of orthogonal projection: see Khan Academy.

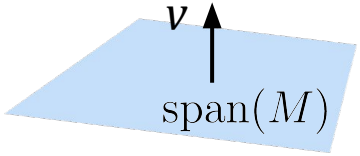
# [Linear Algebra] Orthogonality

**1.** Vector $a$ and Vector $b$ are **orthogonal** if and only if their dot product is 0: $a \cdot b = 0$

This is a generalization of the notion of two vectors in 2D being perpendicular.




**2.** A vector $v$ is **orthogonal** to $\text{span}(M)$, the span of the columns of a matrix $M$, if and only if $v$ is orthogonal to **each column** in $M$.




Let's express **2** in matrix notation. Let $v \in \mathbb{R}^{n \times 1}$, $M \in \mathbb{R}^{n \times d}$ where $M = \begin{bmatrix} | & | & & | \\ m_1 & m_2 & \dots & m_d \\ | & | & & | \end{bmatrix}$:

$$\begin{aligned} m_1^T v &= 0 \\ m_2^T v &= 0 \\ &\vdots \\ m_d^T v &= 0 \end{aligned}$$

$v$ is orthogonal to each column of $M$, $m_j \in \mathbb{R}^{n \times 1}$

$$\begin{bmatrix} m_1^T v \\ m_2^T v \\ \vdots \\ m_d^T v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\underbrace{M^T}_{M^T \in \mathbb{R}^{d \times n}} v = \underbrace{\vec{0}}_{\text{zero vector}}$$

**zero vector** ($d$-length vector full of 0s).

# Ordinary Least Squares Proof

The **least squares estimate** $\hat{\theta}$ is the parameter $\theta$ that minimizes the objective function $R(\theta)$:

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

Design matrix

$$M^T v = \vec{0}$$

Residual vector

Equivalently, this is the $\hat{\theta}$ such that the residual vector $\mathbb{Y} - \mathbb{X}\hat{\theta}$ is orthogonal to $span(\mathbb{X})$.

Definition of orthogonality of $\mathbb{Y} - \mathbb{X}\hat{\theta}$ to $span(\mathbb{X})$ (0 is the $\vec{0}$ vector)

$$\mathbb{X}^T\left(\mathbb{Y} - \mathbb{X}\hat{\theta}\right) = 0$$

Rearrange terms

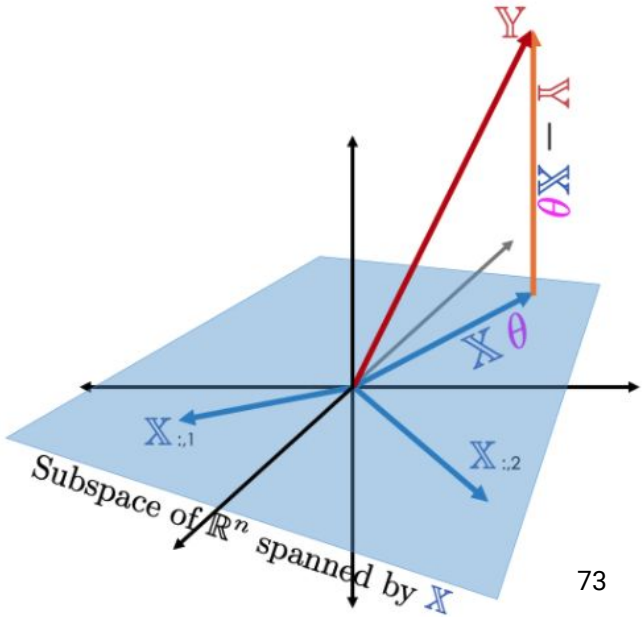
$$\mathbb{X}^T\mathbb{Y} - \mathbb{X}^T\mathbb{X}\hat{\theta} = 0$$

The **normal equation**

$$\mathbb{X}^T\mathbb{X}\hat{\theta} = \mathbb{X}^T\mathbb{Y}$$

If $\mathbb{X}^T\mathbb{X}$ is invertible

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



2327602

$$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$$

This result is so important that it deserves its own slide.

It is the **least squares estimate** and the solution to the normal equation $\mathbb{X}^T\mathbb{X}\hat{\theta} = \mathbb{X}^T\mathbb{Y}$.

2327602

$$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$$

This result is so important that it deserves its own slide.

It is the **least squares estimate** and the solution to the normal equation $\mathbb{X}^T\mathbb{X}\hat{\theta} = \mathbb{X}^T\mathbb{Y}$ .

# Least Squares Estimate

1. Choose a model

Multiple Linear Regression

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

2. Choose a loss function

L2 Loss

Mean Squared Error (MSE)

$$R(\theta) = \frac{1}{n}||\mathbb{Y} - \mathbb{X}\theta||_2^2$$

**3. Fit the model** ✅

Minimize average loss with ~~calculus~~ geometry

$$\hat{\theta} = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{Y}$$

4. Evaluate model performance

Visualize,
~~Root MSE~~
Multiple $R^2$