Lecture #12: PCA and Review

aka STAT109A, AC209A, CSCIE-109A

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Gumb

Lecture Outline: PCA and Review

- Bayes: Simulating a Posterior
- Big Data and High Dimensionality
- Principal Components Analysis (PCA)
- PCA for Visualization
- PCA for Regression (PCR)
- Review

Bayesian Computation Methods

- When a posterior distribution has a well-known distribution (for example, when you are using a conjugate prior), then inferences are easy to develop.
  - You can get closed form solutions for specific estimators for θ: like the posterior mean, posterior median, or posterior mode
- If the posterior is not well known, or if there are a lot of parameters, then inferences on the posterior are not so easy
- We need a numerical/computational way to calculate inferences (estimators, credibility intervals, and hypothesis tests).
- Simulating from the posterior may be the way to go!

Simulating from a Posterior

- If the Posterior distribution has a closed form, but is not a well-known distribution (Normal-Gamma anyone?), then it may be easier to simulate directly from the distribution to get estimates.
- If the posterior can be broken into conditional and marginal distributions, this makes life easier.
- To simulate in this situation:
  1. Collect *nsims* values of θ_1 from the marginal posterior distribution of $f(\theta_1/X)$
  2. For each simulated value of θ_1, select a value for θ_2 from the conditional posterior distribution of $f(\theta_2/\theta_1,X)$

Estimates from a Simulated Posterior

- Calculating the posterior mean:
  - Simply just need to find the empirical estimate of the mean of the simulated distribution of observations of θ.
- Calculating the posterior mode,:
  - Not so simple ☹. Why?
  - Need to fit an entire empirical distribution/curve, and then find the mode of that (aka: bump-hunting).
- How to calculate the credible interval?
  - It's simple: just calculate the desired quantiles from the empirical/simulated distribution.
  - Note: this is equivalent to extracting the Confidence Interval from the empirical bootstrap distribution!

Normal-Normal Model Simulation

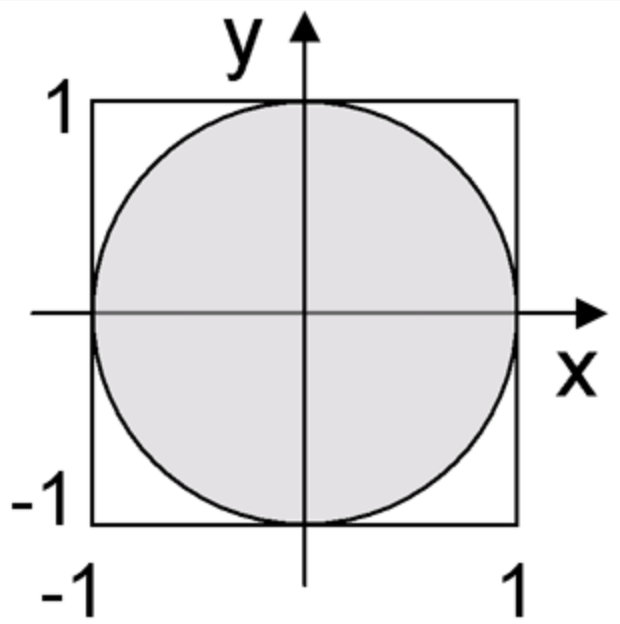
- What is the joint posterior dist. for (μ,σ^2) when $X_i \sim$ Normal, and the priors are $\mu|\sigma^2 \sim$Normal and $1/\sigma^2 \sim$Gamma?
- It's Normal-Gamma of course! What parameters?
  - $\mu/\sigma^2, X \sim N[m_n=(p_0 m_0+n\bar{x})/(p_0+n), \sigma^2/(p_0+n)]$
  - $1/\sigma^2/X \sim Gamma[(v_0+n)/2, SS_n/2 = (SS_0+SS+(np_0)*(\bar{x}-m_0)^2/(p_0+n)/2]$
- How to simulate the distribution? First sample a σ^2 from the Gamma posterior, then a μ/σ^2 from the Normal posterior.
- How do we make inferences from this distribution?
- Calculate some estimates or the credible interval

Markov Chain Monte Carlo

- *Markov Chain Monte Carlo* (MCMC) methods are algorithms to simulate from a probability distribution (like a posterior) that is usually a joint distribution.
- There are MANY examples, and a whole course could be dedicated to these algorithms. Here are a few:
  - Adaptive Rejection: sample from a distribution based on throwing darts at it (essentially).
  - Gibbs Sampling: sample from a distribution based on all of the conditional distributions.
  - Metropolis-Hastings: based on a random walk through a proposal joint density, and a method for rejecting possible moves.
  - And many more…
- We will illustrate the idea here, and then implement them in a future lecture/section (maybe).

Markov Chain Monte Carlo

- Throw a dart at the (x,y) plane based on proposal distribution (simplest one to think of is the uniform).
- If the point is within the distribution, accept x as a random observation. Otherwise, reject it and throw again.
- Simple example: collect a random sample of points from the unit circle:



Gibbs Sampling

- If all of the conditional distributions of the variables are known, then we can use Gibbs sampling. So for 3 parameters, we know $f(\theta_1|\theta_2,\theta_3)$, $f(\theta_2|\theta_1,\theta_3)$, and $f(\theta_3|\theta_1,\theta_2)$.
- The algorithm is as follows:

  0) Select an initial set of values of θ_1,θ_2 , and θ_3.

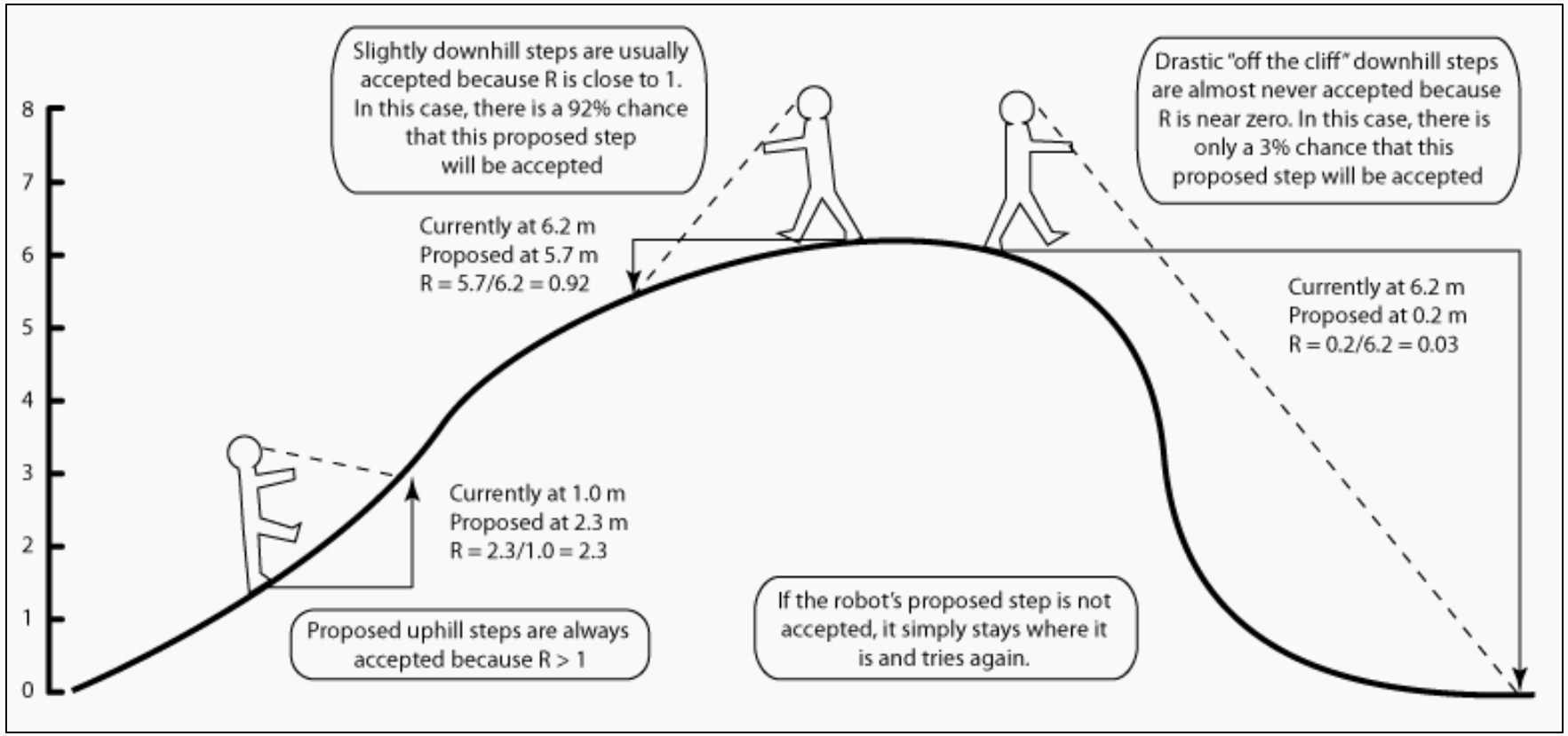
  1) Choose a new θ_1^* from $f(\theta_1|\theta_2,\theta_3)$.

  2) Choose a new θ_2^* from $f(\theta_2|\theta_1^*,\theta_3)$.

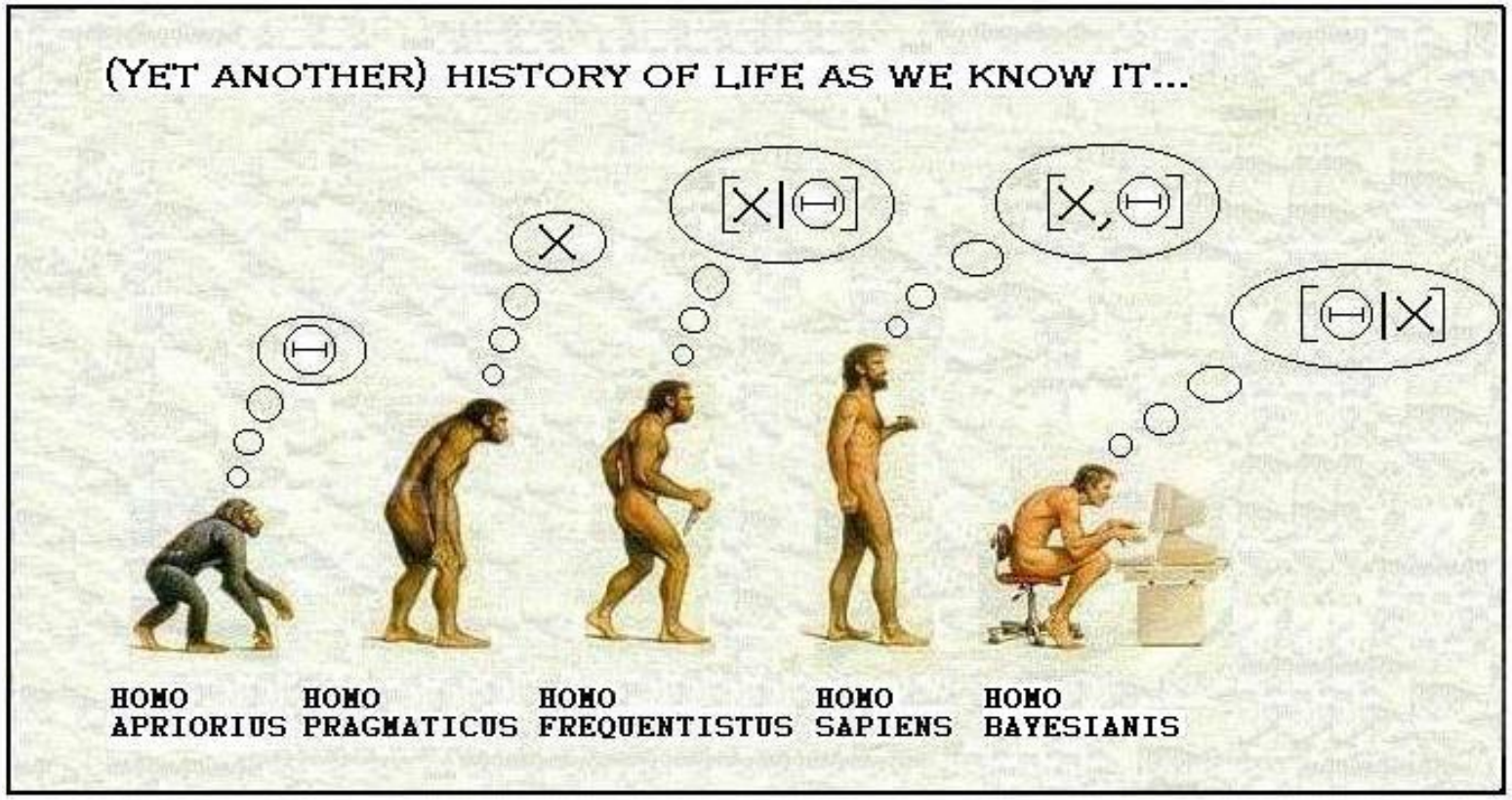
  3) Choose a new θ_3^* from $f(\theta_3|\theta_1^*,\theta_2^*)$.
- Repeat steps 1-3 until…
- After a burn in period (~100 iterations), then keep the remaining results of each iteration as a joint realization of $(\theta_1,\theta_2,\theta_3)$ until you reach the desired number of realizations.
- See any issues with this method? It will be an issue with the next method (metro-hasty) as well.

Metropolis-Hastings Illustration

- Start at an initial point, x. Propose a next point x^*, and accept it as the new spot at the rate of $f(x^*)/f(x)$ (if $f(x^*) \geq f(x)$, then always accept the move). Otherwise, stay in the same spot.
- Like a random walk through the distribution…



Memes of the day



Lecture Outline: PCA and Review

- Bayes: Simulating a Posterior
- **Big Data and High Dimensionality**
- Principal Components Analysis (PCA)
- PCA for Visualization
- PCA for Regression (PCR)
- Review

What is ‘Big Data’?

In the world of Data Science, the term *Big Data* gets thrown around a lot. What does *Big Data* mean?

A rectangular data set has two dimensions: number of observations (n) and the number of predictors (p). Both can play a part in defining a problem as a *Big Data* problem.

What are some issues when:

- n is big (and p is small to moderate)?
- p is big (and n is small to moderate)?
- n and p are both big?

When *n* is big

A very large sample size can pose computational a challenge.

- Algorithms can take forever to finish. Estimating the coefficients of a regression model, especially one that does not have closed form (like LASSO), can take a while.
- Model selection and hyperparameters tuning, especially when using cross-validation, exacerbates the problem.

What can we do to fix this computational issue?

- Perform 'preliminary' steps (model selection, tuning, etc.) on a subset of the training data set. 10% or less can be justified.

Note: statistical inference is not important when *n* is VERY big.

Keep in mind, big *n* doesn't solve everything

The era of Big Data (aka, large *n*) can help us answer lots of interesting scientific and application-based questions, but it does not fix everything.

Remember the old adage: "**garbage in; garbage out**". If the data are not representative of the population, then modeling results can be terrible. Random sampling ensures representative data.

Xiao-Li Meng does a wonderful job describing the subtleties involved (WARNING: it's a little technical, but digestible): https://www.youtube.com/watch?v=8YLdIDOMEZs

When p is big

When the number of predictors is large (many interactions, polynomial terms, etc.), lots of issues can occur.

What are some issues that may arise?

- Matrices may not be invertible (issue in OLS).
- Multicollinearity is likely to be present
- Models are susceptible to overfitting

This situation is called *High Dimensionality* and needs to be accounted for when performing data analysis and modeling.

When Does High Dimensionality Occur?

The problem of high dimensionality can occur when the number of parameters exceeds or is close to the number of observations. This can happen when we consider lots of interaction terms, like in our previous example. But this can also happen when the number of main effects is high.

For example:

- When we are performing polynomial regression with a high degree and a large number of predictors.
- When the predictors are genomic markers (and possible interactions) in a computational biology problem.
- When the predictors are the counts of all English words appearing in a text.

How Does sklearn handle unidentifiability?

In a parametric approach: if we have an over-specified model ($p > n$), the parameters are unidentifiable: we only need $n - 1$ predictors to perfectly predict every observation ($n - 1$ because of the intercept).

So, what happens to the 'extra' parameter estimates (the extra $\beta's$)?

The remaining $p - (n - 1)$ predictors' coefficients can be estimated to be *anything!* Thus, there are an infinite number of sets of estimates that will give us identical predictions. There is not one unique set of $\beta's$!

Perfect Multicollinearity

The $p > n$ situation leads to perfect collinearity of the predictor set. But this can also occur with redundant predictors.

Let's see what sklearn does in this situation:

```
regress_sqft = sk.linear_model.LinearRegression().fit(X = homes[['sqft']], y = homes['price'])
print("Intercept =",regress_sqft.intercept_.round(2),", Slope =",regress_sqft.coef_[0].round(4))

regress_sqft2 = sk.linear_model.LinearRegression().fit(X = homes[['sqft','sqft']], y = homes['price'])
print("Intercept =",regress_sqft2.intercept_.round(2),", Slopes =",regress_sqft2.coef_.round(4))
```

```
Intercept = 247438.24 , Slope = 589.7823
Intercept = 247438.24 , Slopes = [294.8911 294.8911]
```

Lecture Outline: PCA and Review

- Bayes: Simulating a Posterior
- Big Data and High Dimensionality
- **Principal Components Analysis (PCA)**
- PCA for Visualization
- PCA for Regression (PCR)
- Review

In high dimensional space, data are sparse

Consider the example where we add many interaction and/or polynomial terms to our design matrix.

As the number of dimensions increase the feature space is soon dominated by empty space.

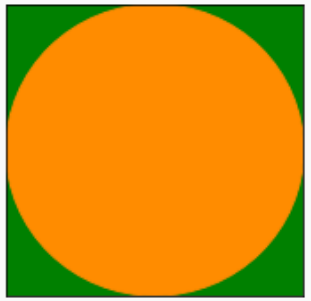
We say that the resulting feature space is ‘sparse.’

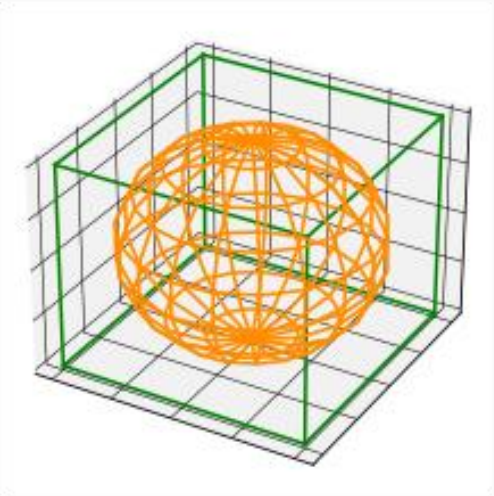
One consequence is that the number of data points required for your machine learning algorithms to remain effective increases exponentially.

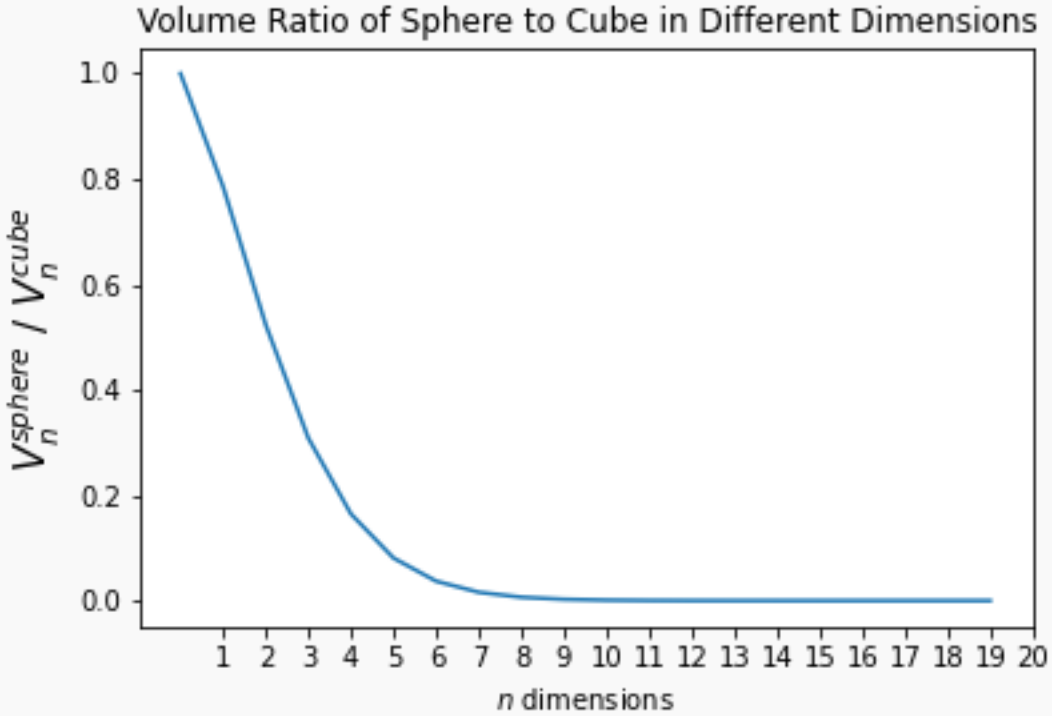
Don’t trust your spatial intuitions in high dimensions

Strange things happen in high dimensions, as illustrated by the example of the volume ration of an *n*-sphere to an *n*-cube rapidly *vanishing to zero* as *n* increases. This is disconcerting, but what implications does it have for doing data science?

$$V_n^{sphere} = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}+1\right)} \qquad V_n^{cube} = 2^n$$







In high dimensional space, there are no ‘neighbors’

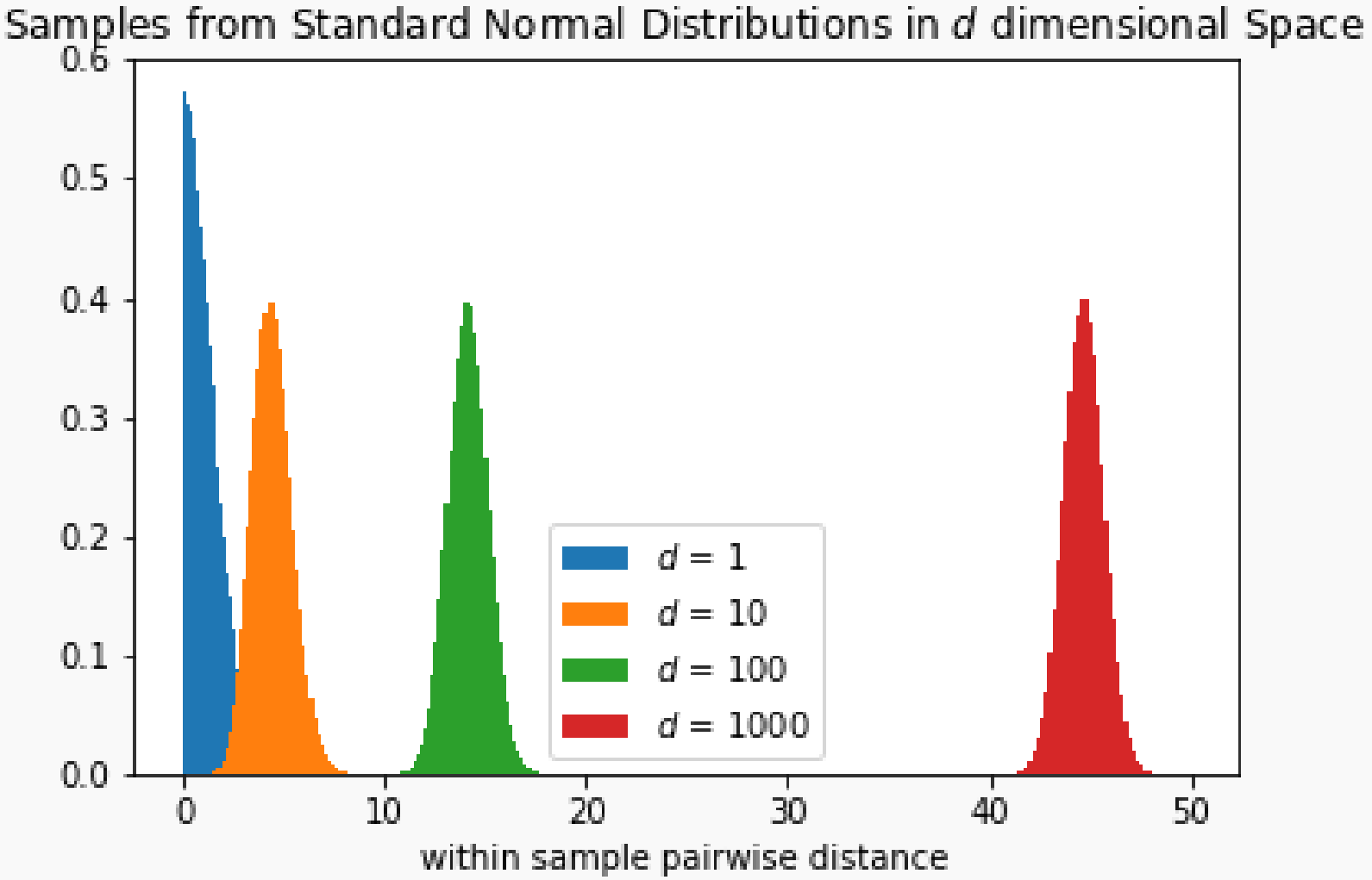
Many algorithms rely on the notion that nearby points in the feature space are ‘similar.’ And this similarity is used to inform predictions.

k-NN is one such algorithm where this idea of nearness = similarity is quite explicit.

It’s also true that, as the dimensionality increases, the calculation of the pairwise distances required by *k*-NN quickly become intractable as there are more dimensions to consider in each distance calculation. But this is the least of your worries!

In high dimensional space, all points are far away from one another. The notion of ‘neighbors’ begins to break down.

It’s lonely in high dimensional space

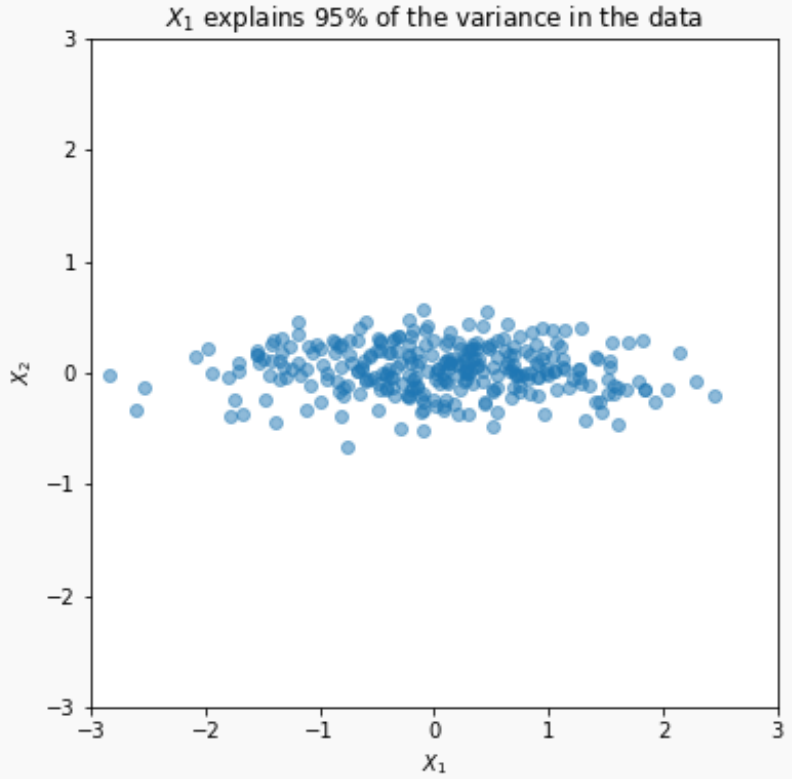


High Dimensionality and Overfitting

- More predictors means more degrees of freedom.
- The more degrees of freedom, the easier it is for our model to fit to noise in the training data, leading to poor generalization. This is overfitting.
- If some predictors are highly correlated, then there is redundancy in the dataset.
- Eliminating redundant predictors and those providing little to no new information will help prevent overfitting while retaining as much of the original information as possible.

Which predictors should be removed or retained?

One strategy is to remove those predictors which explain the least amount of variance in the data, or, equivalently, retain those *X*s that explain the most variance of the predictor set.

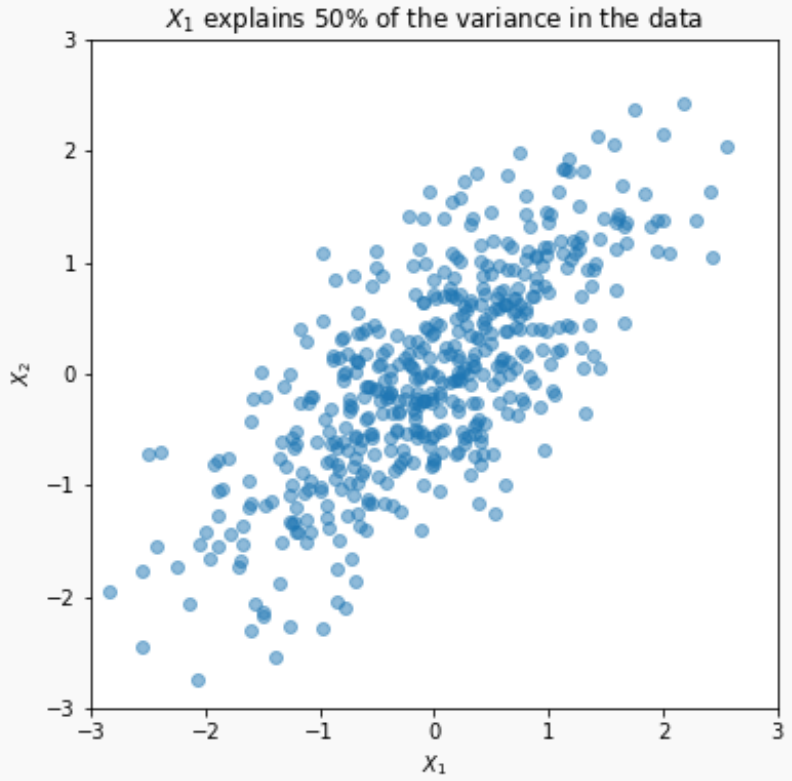


In this example, almost all the variance of the predictors is in X_1.

Keeping only this predictor would retain most of the information in the predictor set while cutting the dimensionality in half.

An example with moderate correlation

If the variance explained by our predictors is very similar, then this metric will not help us decide which predictor(s) to keep.

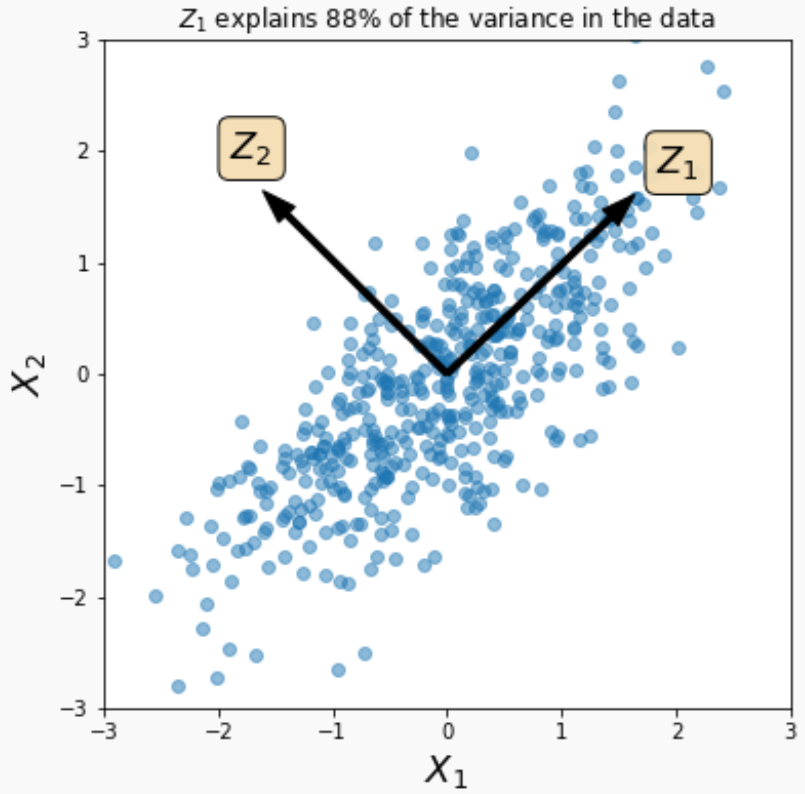


Here the variances of X_1 and X_2 are the same!

But we can clearly see there are *some directions in the predictor space* that have more spread than others. They just don't lie along the axes (i.e., the basis vectors)

Directions of maximum variance

The vector Z_1 represents the direction of maximum variance in the predictor space. Z_2 is orthogonal and captures the remaining unexplained variance.

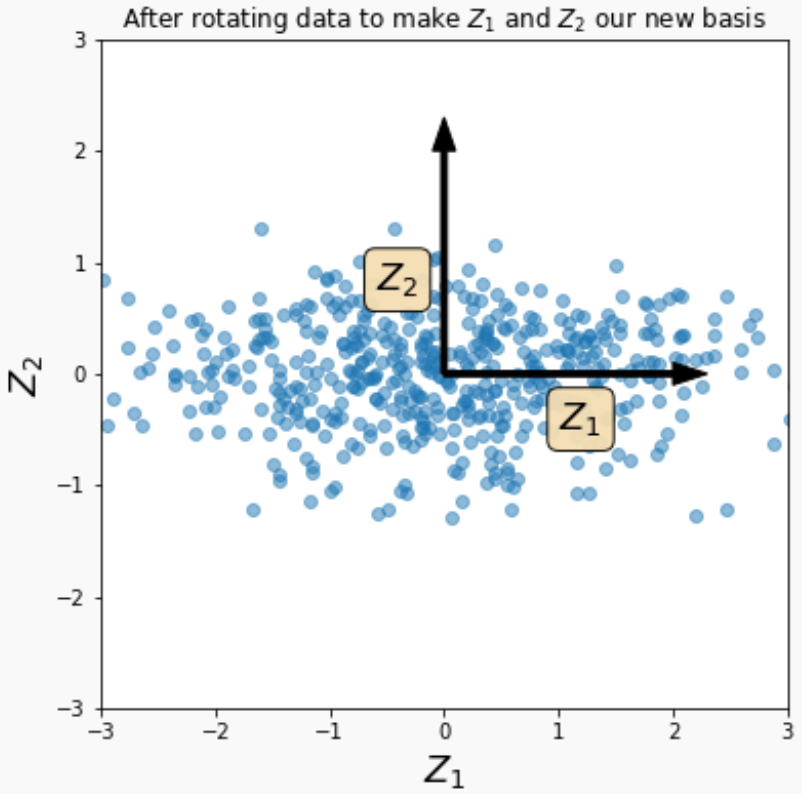


If our data were represented in terms of Z_1 and Z_2 then we could decide to keep only the Z_1 component.

This would again cut the dimensionality in half while retaining the most information.

Rotation for a change of basis

We can rotate our data so that the orthogonal Z_1 and Z_2 vectors now lie along the axes. This is equivalent to a change of basis from X_1 and X_2 to Z_1 and Z_2



We can think of Z_1 and Z_2 as *a new set of predictors*. The decision of which predictor to keep to retain the most variance is once again clear.

Great! But how did we discover the optimal vectors Z_1 and Z_2!?

A Framework for Linear Dimensionality Reduction

Our strategy will be to create a new, smaller set of predictors by taking linear combinations of the original predictors.

For $m < p$ we choose $Z_1, Z_2, \ldots, Z_m$, where each Z_i is a linear combination of the original p predictors:

$$Z_i = \sum_{j=1}^{p} \phi_{ji} X_j$$

for fixed constants Φ_{ji}. Then we can build a linear regression regression model using the new predictors

$$Y = \beta_0 + \beta_1 Z_1 + \cdots + \beta_m Z_m + \varepsilon$$

Notice that this model has a smaller number ($m+1 < p+1$) of parameters.

A Framework for Linear Dimensionality Reduction (cont.)

Our method of dimensionality reduction includes 2 steps:

1. Determine an optimal set of new predictors $z_1, \ldots, z_m$ for $m < p$.
2. Express each observation in the data in terms of these new predictors. The transformed data will have m columns rather than p.

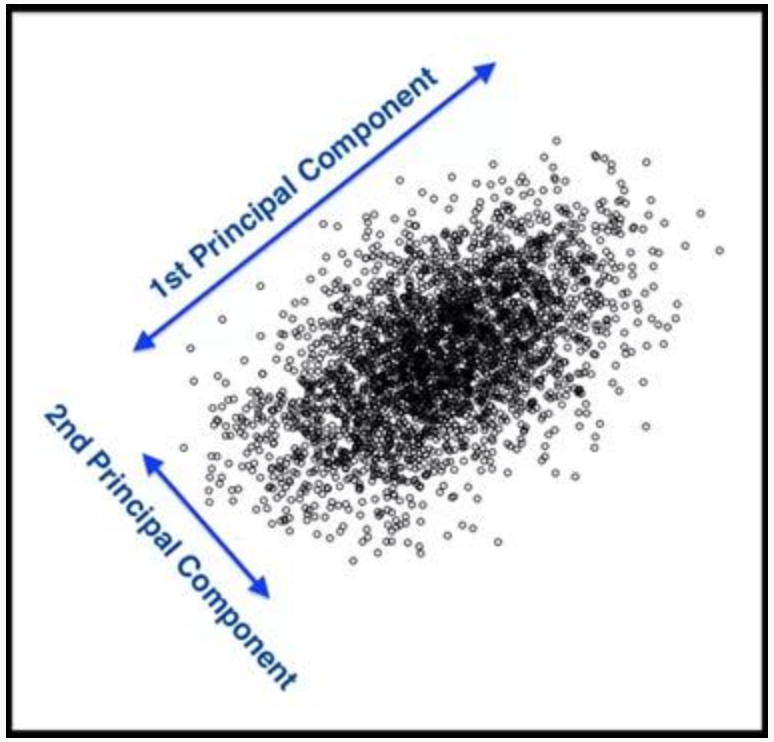
Thereafter, we can fit a model using the new predictors.

The method for deciding what is meant by an "optimal" set of new predictors can differ according to application. We will continue to explore a way to create new predictors that captures the **most variation** in the observed predictor set.

Big data = Bad Time

Principal Components Analysis (PCA)

Principal Components Analysis (PCA) is a method to identify a new set of predictors, as linear combinations of the original ones, that captures the maximum amount of variance in the observed data (predictors).



PCA (cont.)

PCA produces a list of p **principal components** $Z_1,\ldots, Z_p$ such that:

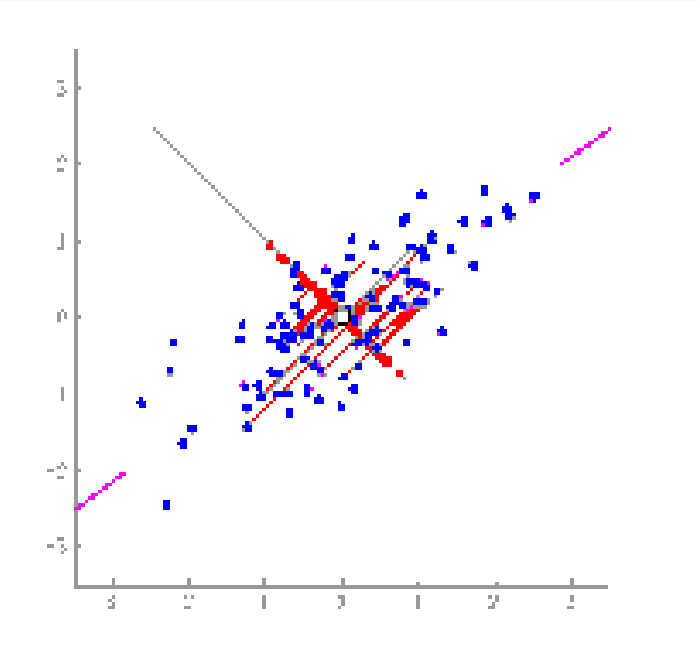
I. Each Z_i is a linear combination of the original predictors
II. The Z_i's are orthonormal (pairwise orthogonal with unit length)
III. The Z_i's are ordered based on the amount of variance explained

That is, the observed data shows more variance in the direction of Z_1 than in the direction of Z_2.

To perform dimensionality reduction, we select the top m principal components and express our data in terms of these predictors.

The Intuition Behind PCA (cont.)

Transforming our observed data means **projecting** our dataset onto the space defined by the top *m* PCA components. These components become our new predictors.



The animation shows projections onto several candidate vectors in red.

You can see the points retain the most spread when projected onto what turns out to be the first PCA component.

The Math behind PCA (for those whose are curious)

Let $\boldsymbol{X}$ be the n x p matrix with columns X_1,..., X_p (our original predictors), each standardized to have mean zero and variance one, and without the intercept)

Let let $\boldsymbol{W}$ be the p x p matrix whose columns are the eigenvectors of the covariance matrix, $\boldsymbol{X}^T\boldsymbol{X}$

Let $\boldsymbol{Z}$ be the n x p matrix with columns Z_1,..., Z_p (the full set of principal components):

$$\mathbf{Z}_{n\times p} = \mathbf{X}_{n\times p}\mathbf{W}_{p\times p}$$

Implementation of PCA using linear algebra (optional)

To implement PCA yourself you can perform the following steps:

I. [ideally] standardize your predictors (so they each have mean = 0, sd = 1).

II. Calculate the eigenvectors of $\boldsymbol{X^T X}$ and and arrange them as columns in order of descending eigenvalues in a new matrix, $\boldsymbol{W}$.

III. Use matrix multiplication to project $\boldsymbol{X}$ onto the eigenvectors to create the new feature matrix, $\boldsymbol{Z = XW}$

Note: this is not efficient from a computational perspective. This can be sped up using Cholesky decomposition.

PCA is easy to perform in Python using sklearn's decomposition.PCA class.

PCA example in sklearn

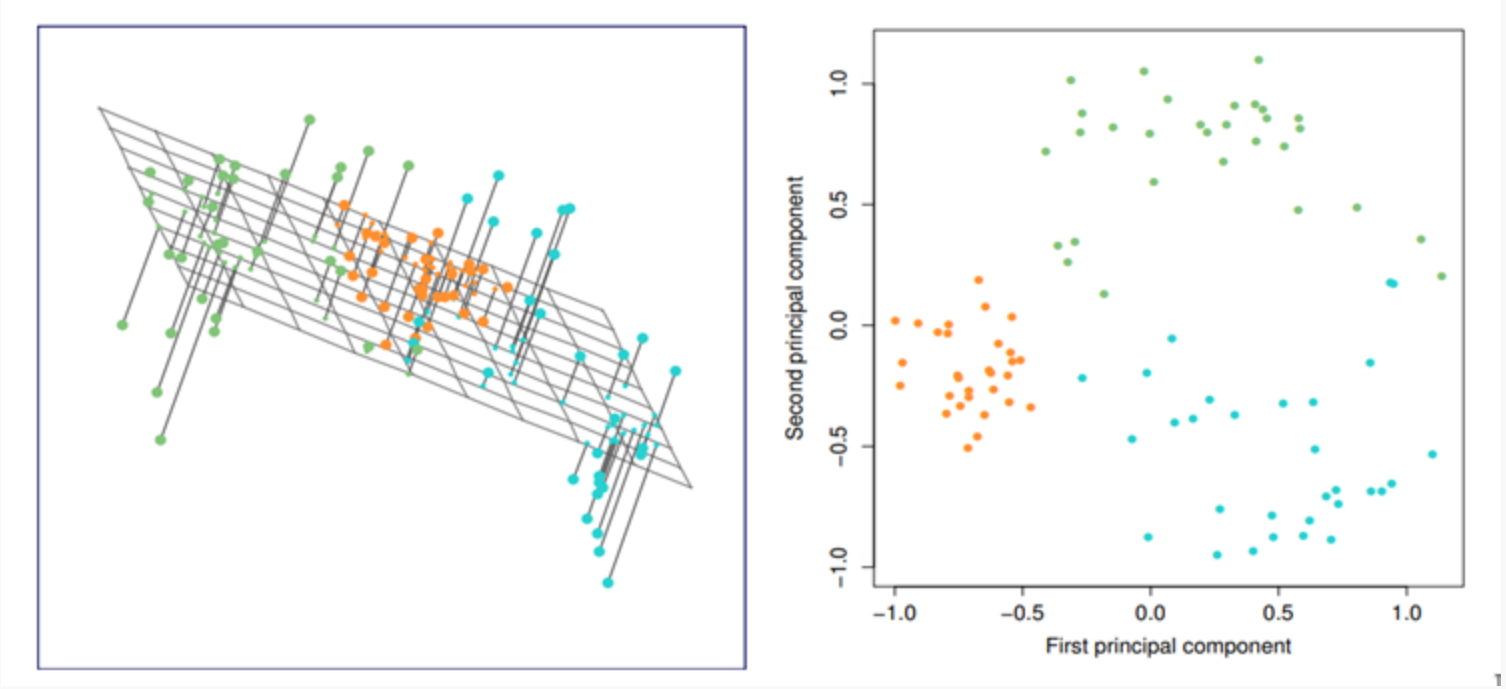
```
X = homes[['sqft','beds','baths','lotsize','dist']]

pca = PCA().fit(X)

pcaX = pca.transform(X)
W = pca.components_.T


print("First PCA Component vector (w1):",W[0,:].round(7))
print("Second PCA Component vector (w2):",W[1,:].round(7))

print("Variance explained by each component:",pca.explained_variance_ratio_.round(7))
```

```
First PCA Component vector (w1): [ 3.237999e-01  9.461243e-01 -7.237000e-04  9.225000e-04  1.009800e-03]
Second PCA Component vector (w2): [ 4.902000e-04  1.144900e-03  6.274349e-01 -7.753376e-01 -7.194070e-02]
Variance explained by each component: [9.097255e-01 9.027410e-02 2.000000e-07 1.000000e-07 1.000000e-07]
```

An Alternative Interpretation of PCA

We've seen an interpretation of PCA as finding the directions in the predictor space along which the data varies the most.

An alternative interpretation is that PCA finds a low-dimensional linear surface which is *closest* to the data points.



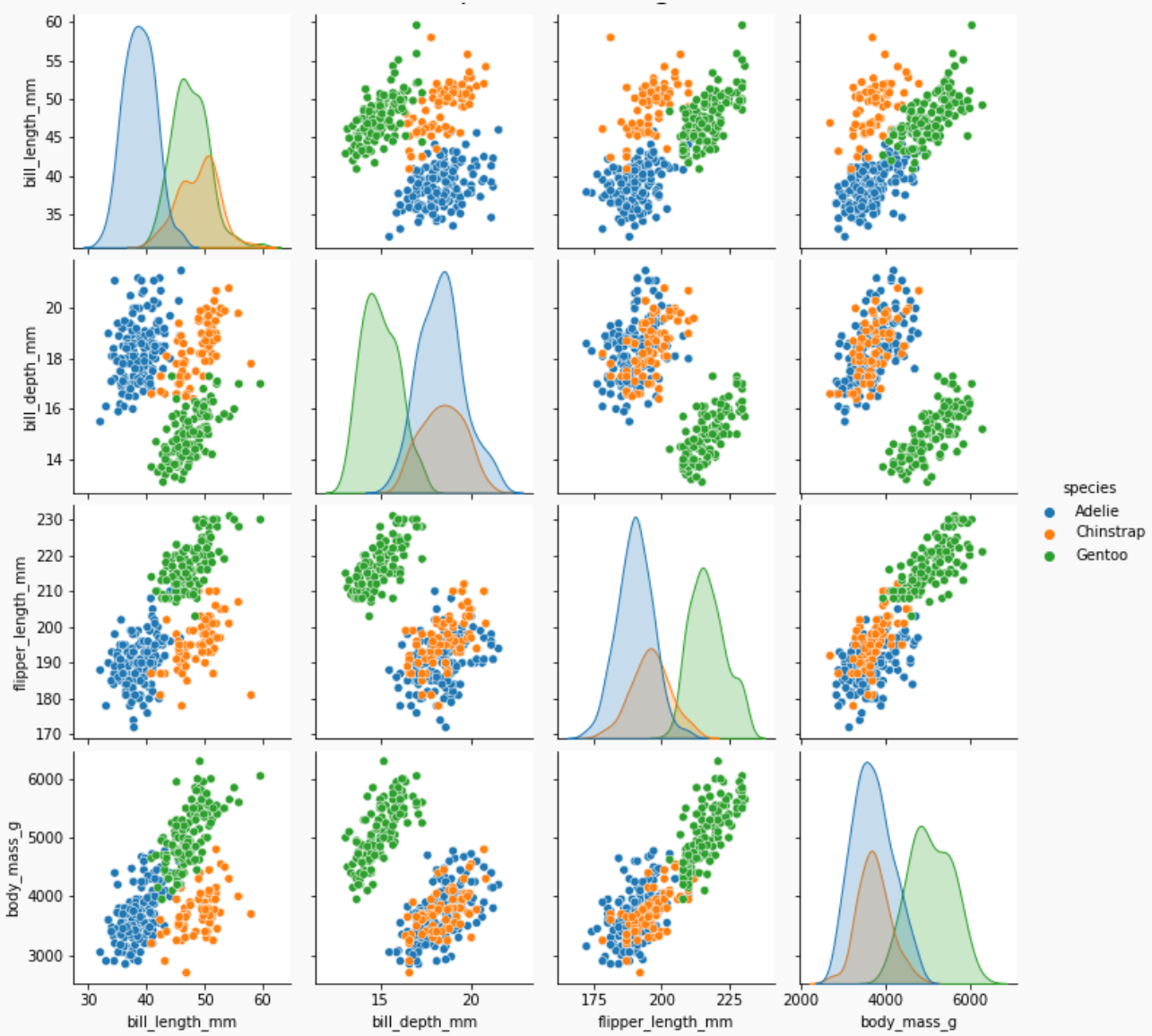
Lecture Outline: PCA and Review

- Bayes: Simulating a Posterior
- Big Data and High Dimensionality
- Principal Components Analysis (PCA)
- **PCA for Visualization**
- PCA for Regression (PCR)
- Review

Visualizing high dimensional data

- Visualization is often the quickest ways to gain insight into the relationships and patterns within a dataset.
- But how can we possibly visualize data beyond 2 or 3 dimensions?
- One option is to compromise and look only at small ‘slices’ of the predictor space at time.
- For example, we can create a series of pair-wise 2-D scatter plots between all the predictors.

3 Penguin Species: Pair-plot of features

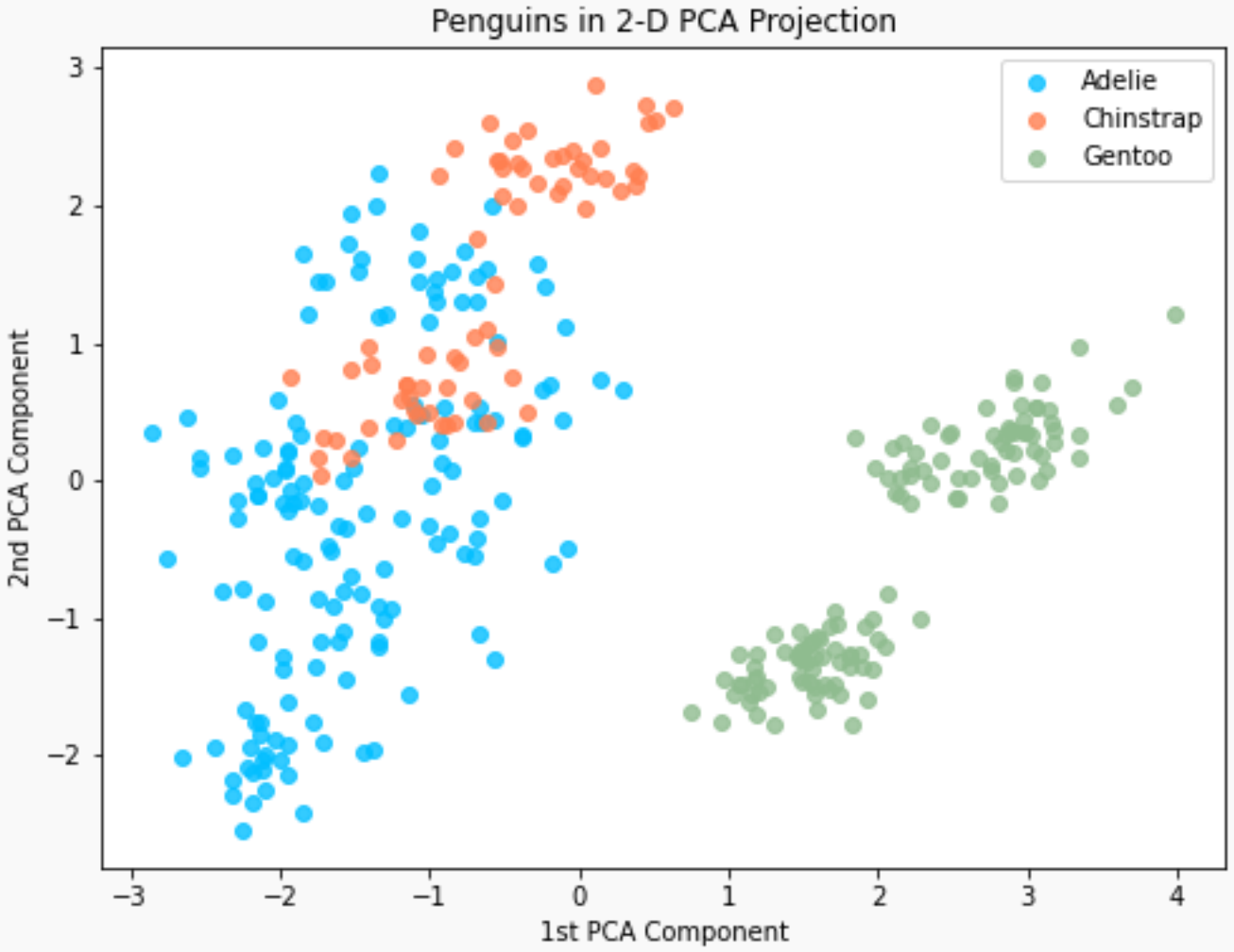


We have 4 measurements (4 *X*s) from 3 species of penguins: body mass, flipper length, bill length, and bill depth.

We can see some separation of the species in a few of the scatter plots. But what about relationships involving more than just 2 predictors?

This approach also becomes unwieldy with more than just a few predictors.

3 Penguin Species: Visualized with PCA



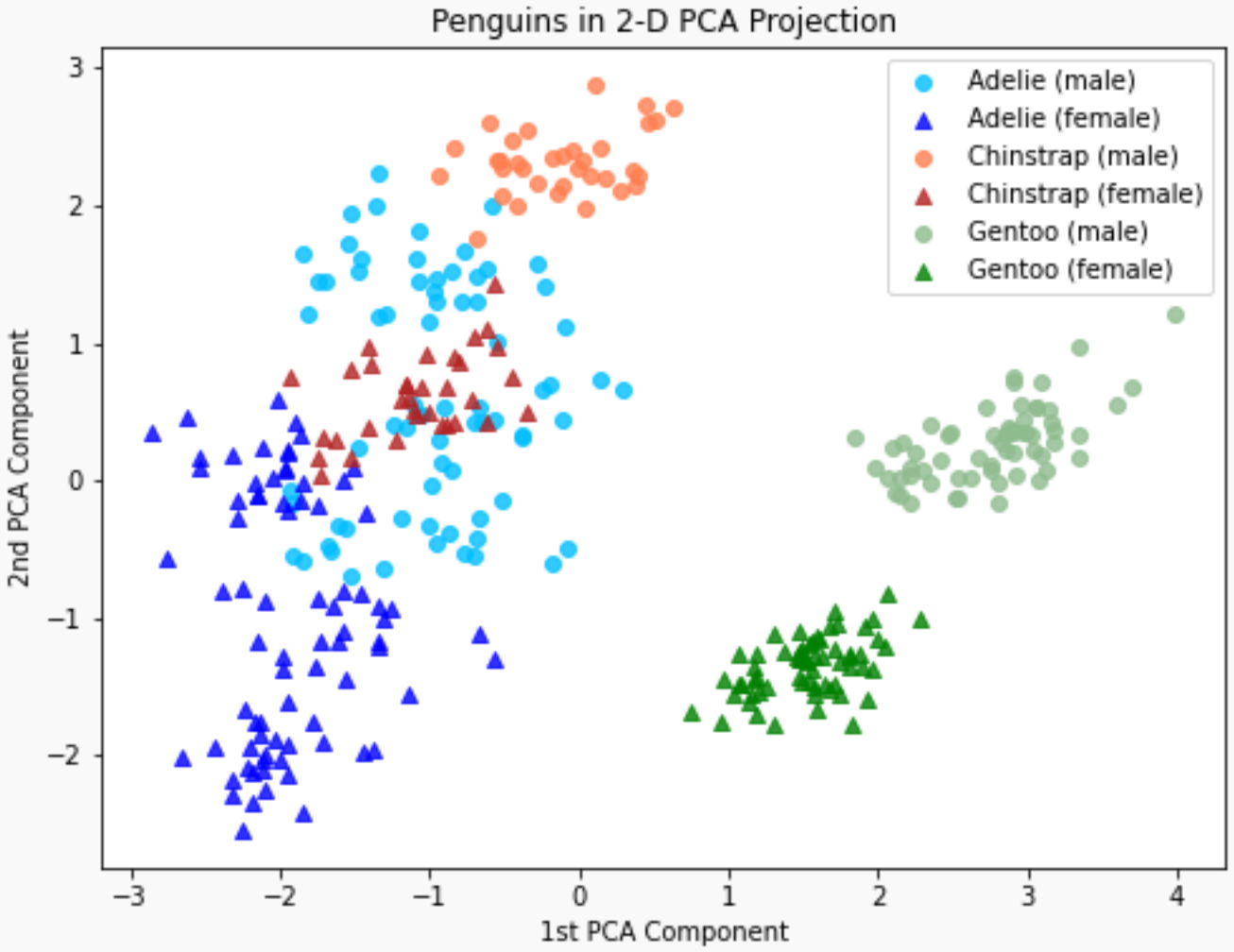
Here is the data projected onto the first 2 principle components.

Perhaps you're not convinced this is an improvement?

It might not look too different from the previous plots.

But notice that there are also some distinct clusters within species...

3 Penguin Species: Visualized with PCA



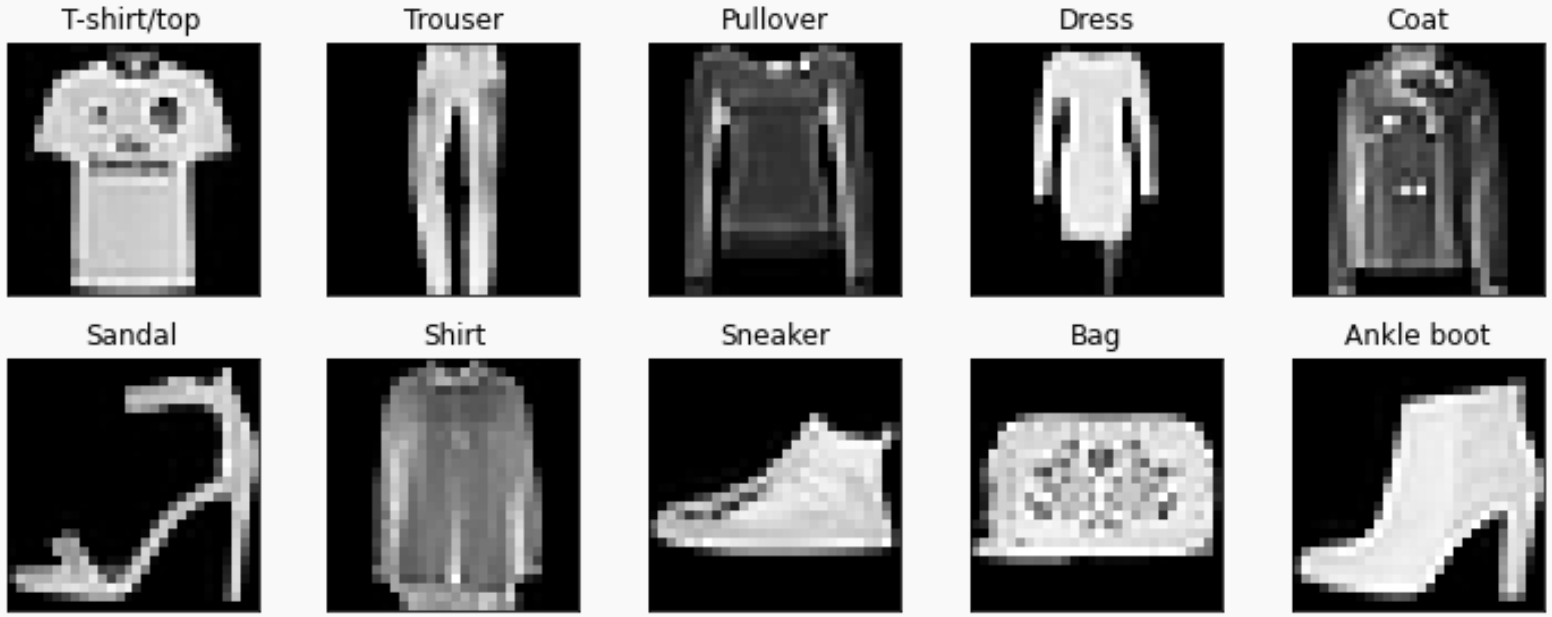
It turns out that these within species clusters correspond to male and female penguins.

Now let's try another example with a much higher dimensional dataset...

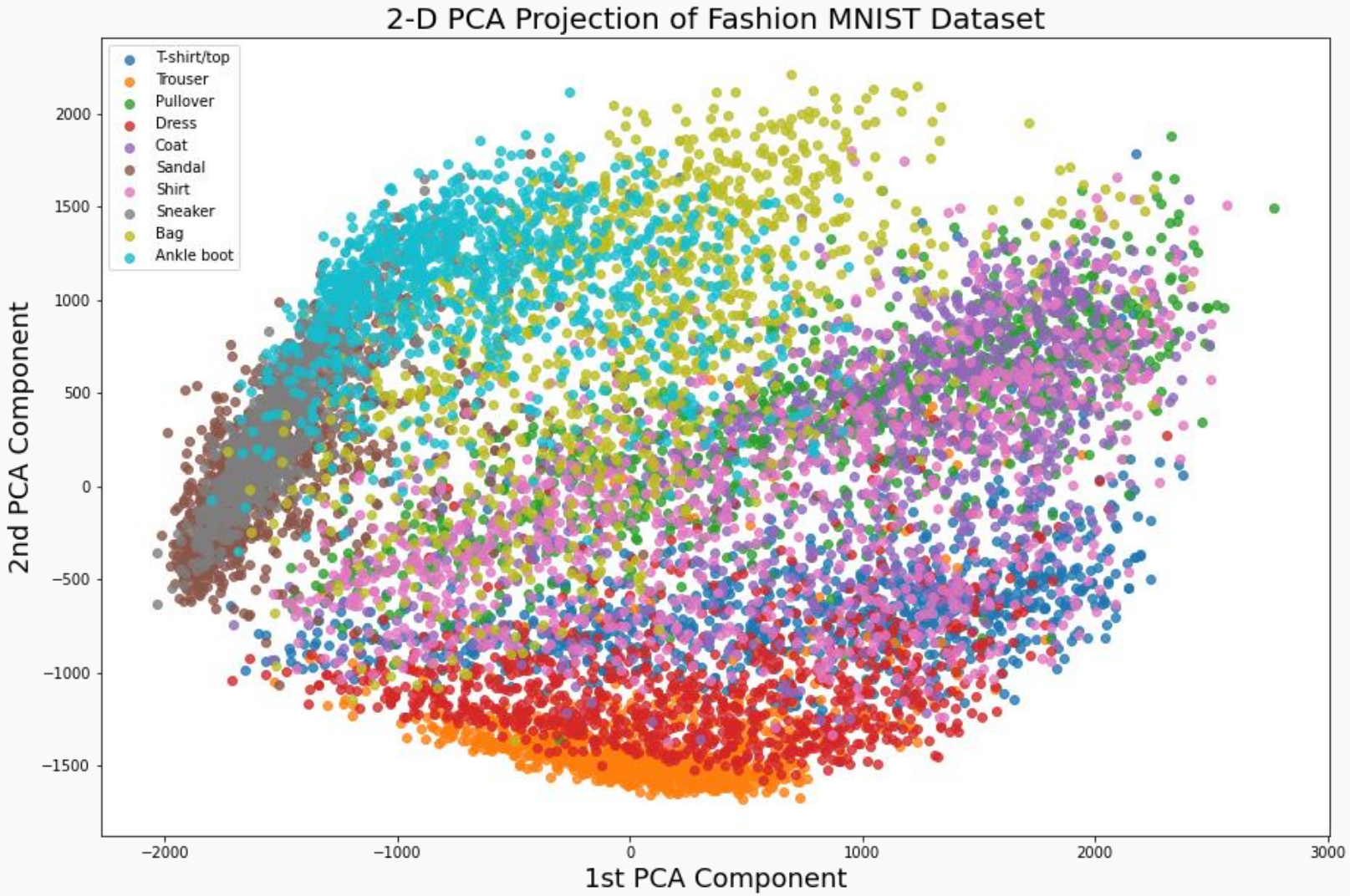
Images as high dimensional data

The **Fashion MNIST** data set consists of thousands of 28x28 grayscale images of articles of clothing from 10 different categories. Each pixel is essentially a feature. 28 x 28 = 784 features!

And obviously plotting pair-wise scatter plots of pixel values wouldn't be insightful.



PCA for visualizing image data

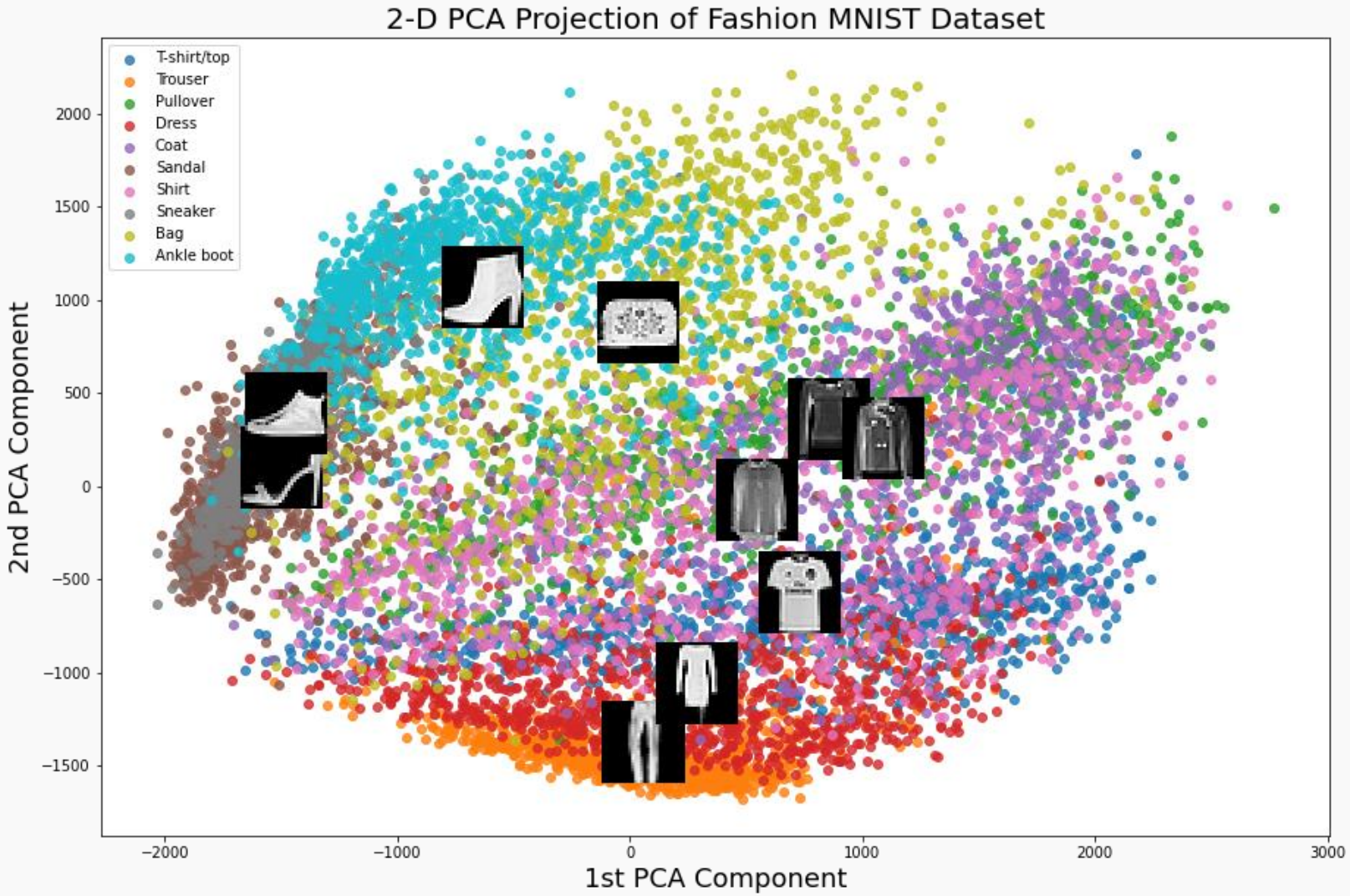


Projecting onto the top 2 principal components allows us to visualize the dataset.

While there exists considerable overlap, we see that most categories are grouped together.

And remember, PCA never actually saw the category labels!

PCA for visualizing image data



Displaying examples at the center of their respective cluster we can appreciate that similar categories are close to one another in the projected space.

All this was done by finding linear combinations of pixels that explained the most variance in the data!

Lecture Outline: PCA and Review

- Bayes: Simulating a Posterior
- Big Data and High Dimensionality
- Principal Components Analysis (PCA)
- PCA for Visualization
- **PCA for Regression (PCR)**
- Review

PCA for Regression (PCR)

PCA is easy in Python, so how can we use it for regression modeling in a real-life problem?

If we use all p of the new Z_i, then we have *not* reduced the dimensionality. To do so, we must select only the first m PCA variables, $Z_1,\ldots,Z_m$, to use as predictors for some $m < p$.

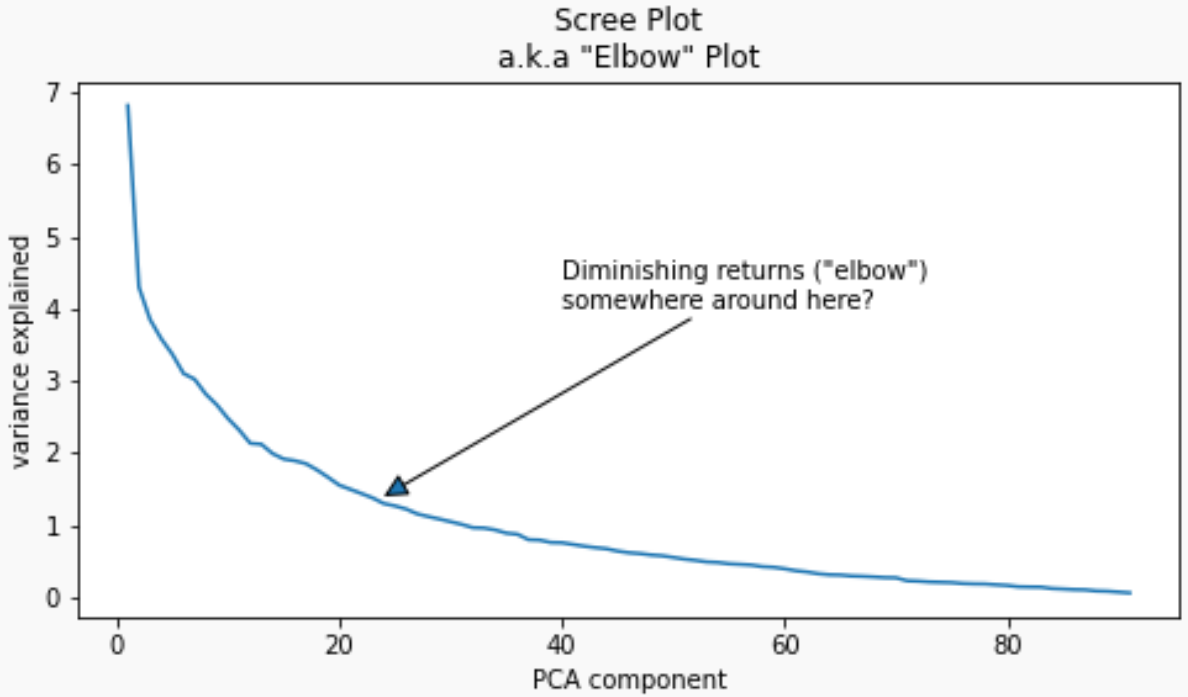
The choice of m is important and can vary from application to application. It depends on various things, like how collinear the predictors are, how truly related they are to the response, etc.

So how should we decide on a value of m?

How many components to keep?

One approach for deciding on the number of components to keep is to just “eyeball” it.

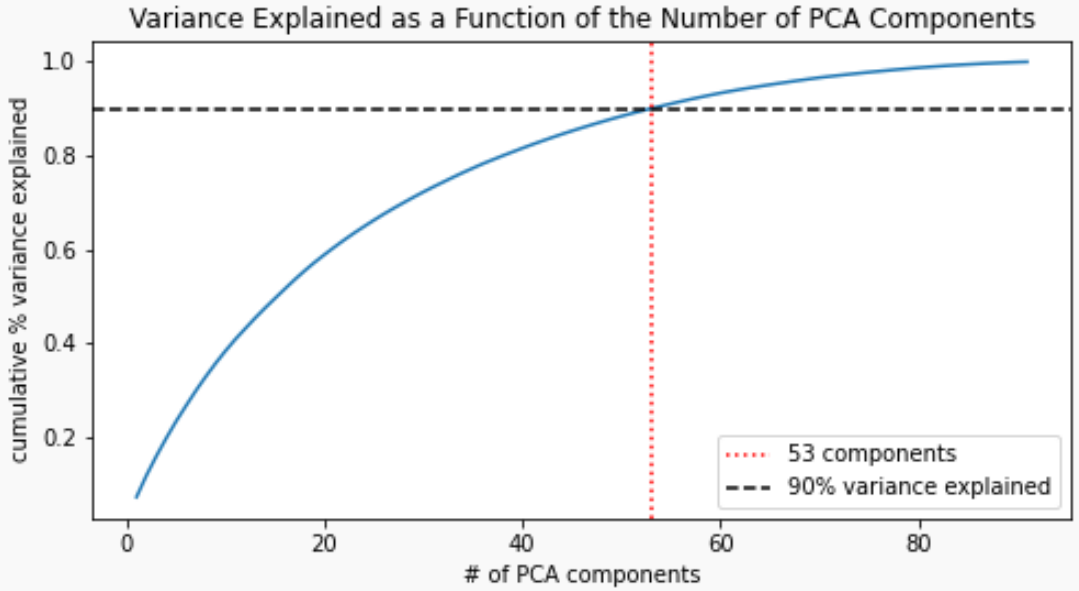
We look for a point of diminishing returns, or “elbow”, in a plot showing the variance explained by each component.



How many components to keep? (cont.)

Another approach is to specify how much of the total variance we want explained by our *m* components. For example, 90% of the total variance.

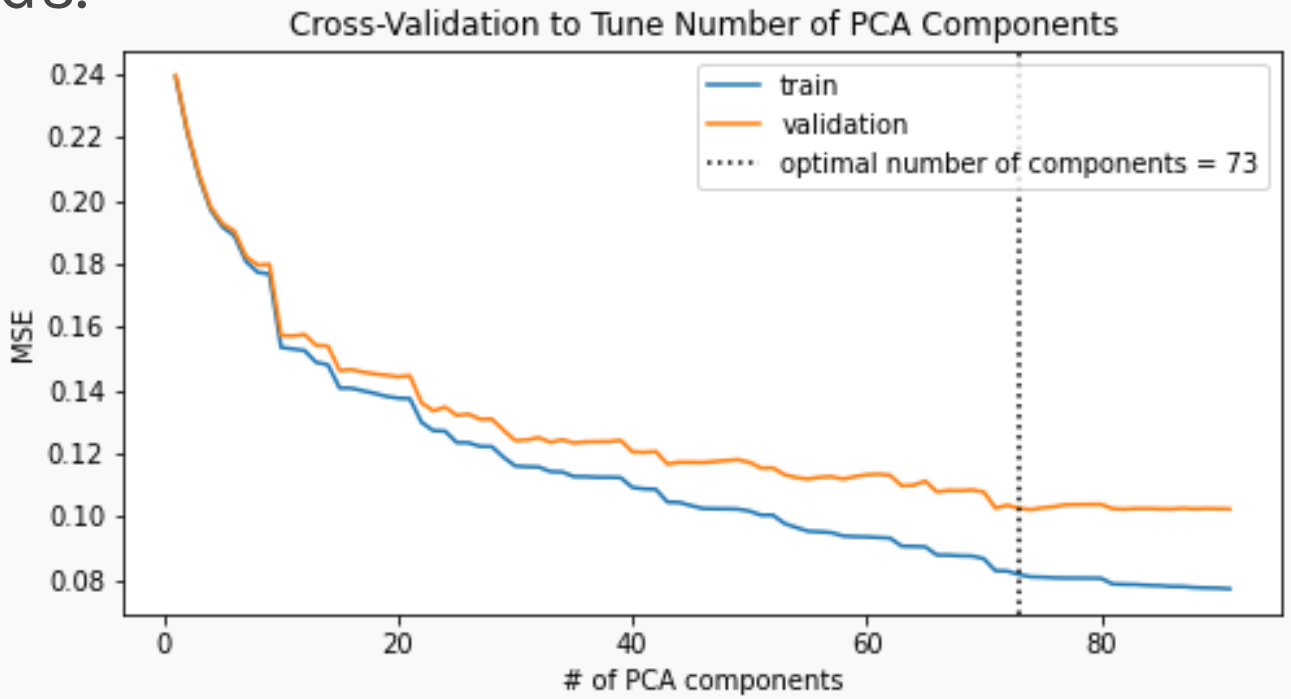
We then inspect the cumulative variance explained by each additional component and stop when we meet out goal.



Cross-validation to tune # of components in PCR

The two previous strategies **did not consider model performance!**

If modeling is your end goal, then you can use cross-validation to *tune* the number of components like any other hyperparameter, selecting the number that received the best mean validation error across all folds.



Interpreting the Results of PCR

A PCR can be interpreted in terms of original predictors…very carefully.

Each estimated β coefficient in the PCR can be *distributed* across the predictors via the associated component vector, *w*.

An example is worth a thousand words:

```
pcaX_df = pd.DataFrame(pcaX, columns=[f'PCA{i}' for i in range(1,X.shape[1]+1)])

PCR_simple = sk.linear_model.LinearRegression().fit(X = pcaX_df[['PCA1']], y = homes['price'])
print("PCA Intercept =",PCR_simple.intercept_.round(2),", PCA Slope =",PCR_simple.coef_[0].round(4))

print("First PCA Component vector (w1):",W[0,:])
```

```
PCA Intercept = 1244.2 , PCA Slope = 0.2049
First PCA Component vector (w1): [ 3.23799939e-01  9.46124307e-01 -7.23671683e-04  9.22493237e-04
  1.00975436e-03]
```

So how can this be transformed back to the original variables?

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 Z_1 = \hat{\beta}_0 + \hat{\beta}_1(\vec{w}_1^T X) = \hat{\beta}_0 + \left(\hat{\beta}_1 \vec{w}_1^T\right) X$$

$$= 1244.2 + 0.2049 \cdot (0.3238X_1 + 0.9461X_2 - 0.00072\, X_3 + 0.00092X_4 + 0.00101X_5)$$

$$= 1244.2 + 0.0663X_1 + 0.194X_2 - 0.00015\, X_3 + 0.00019X_4 + 0.00021X_5$$

Interpreting the Results of PCR

You can always put the PCR coefficients in terms of the original predictors.

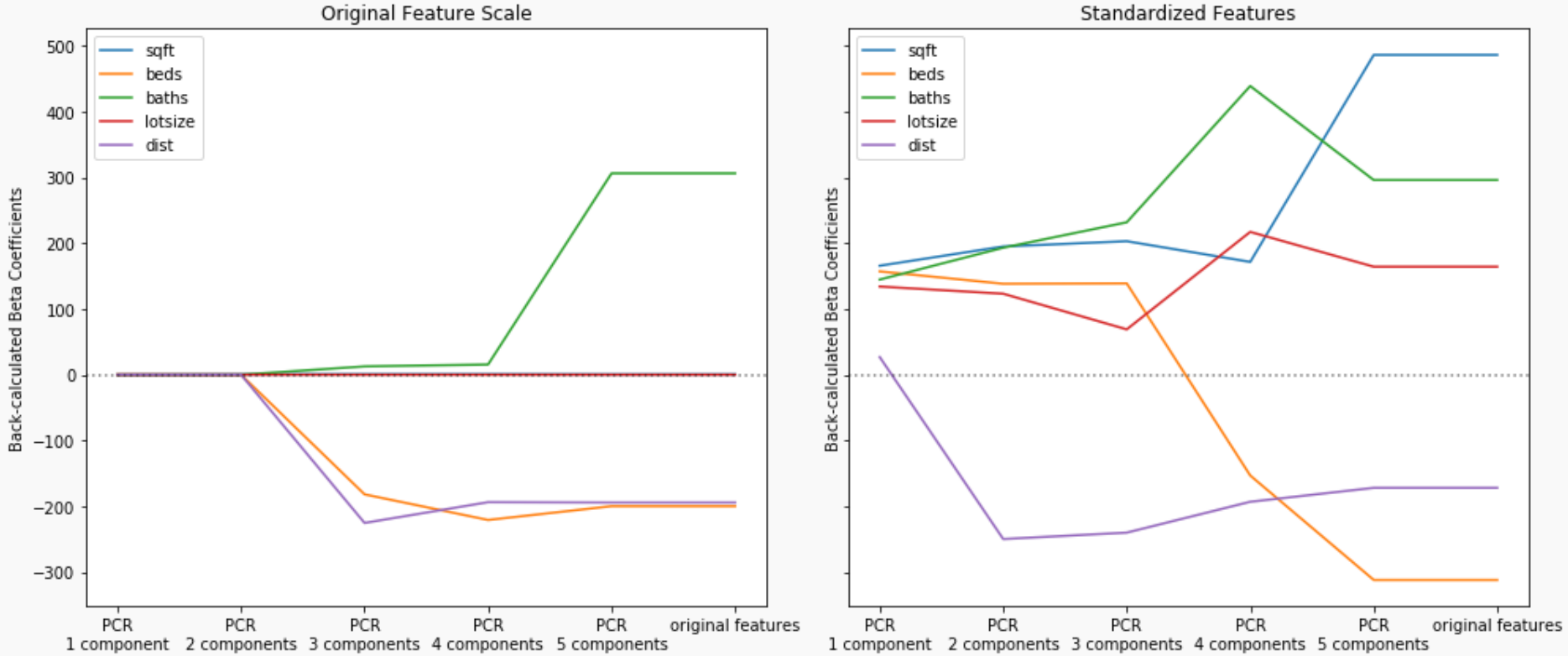
$$\hat{y} = \boldsymbol{Z}\beta_Z = (\boldsymbol{X}\boldsymbol{W}_m)\beta_Z = \boldsymbol{X}(\boldsymbol{W}_m\beta_Z) = \boldsymbol{X}\beta_X$$

$\boldsymbol{X}$ is our original dataset, centered to have a mean of zero
$\boldsymbol{W}_m$ is the matrix whose columns are the first m eigenvectors of $\boldsymbol{X}^\top\boldsymbol{X}$
$\boldsymbol{Z}_m$ is the transformed dataset created by projecting $\boldsymbol{X}$ onto the principal components contained in $\boldsymbol{W}_m$
β_Z are the coefficients for the PCR model
β_X are coefficients in terms of the original predictors

When $m = p$ then the recovered coefficients are identical to those from a model fit on the original predictors.

Coefficient ‘paths’ as more components enter the PCR

Coefficients for the PCR model using all components are identical to coefficients found when fitting on the original predictors.



Notice the effect standardizing has on the coefficients.

A few notes on using PCA

- PCA cons:
    1. PCA *knows nothing about the response variable*. Component vectors as predictors might not be ordered from best to worst!
    2. *Direct* interpretation of coefficients in PCR is lost, so if easy interpretation is important to you, perhaps steer clear.
    3. PCA can often fail to improve the predictive power of a model.
- PCA pros:
    1. Can help avoid the curse of dimensionality and overfitting.
    2. Easy to visualize how predictive your features are of the response variable, especially in the classification setting.
    3. Reduces multicollinearity, and so can improve the computational time when fitting models.

Lecture Outline: PCA and Review

- Bayes: Simulating a Posterior
- Big Data and High Dimensionality
- Principal Components Analysis (PCA)
- PCA for Visualization
- PCA for Regression (PCR)
- **Review**

Hypothesis Testing

Hypothesis testing is a formal process through which we evaluate the validity of a statistical hypothesis by considering evidence **for** or against the hypothesis gathered by **random sampling** of the data. The steps are:

1. State the hypotheses, typically a **null hypothesis**, H_0 and an **alternative hypothesis**, H_A, that is the negation of the former.
2. Choose a type of analysis, i.e. how to use sample data to evaluate the null hypothesis. Typically, this involves choosing a single test statistic.
3. **Sample** data and compute the test statistic.
4. Use the value of the test statistic (or the p-value) to either reject or not reject the null hypothesis.
5. Restate the conclusion in context of the problem.

p-value

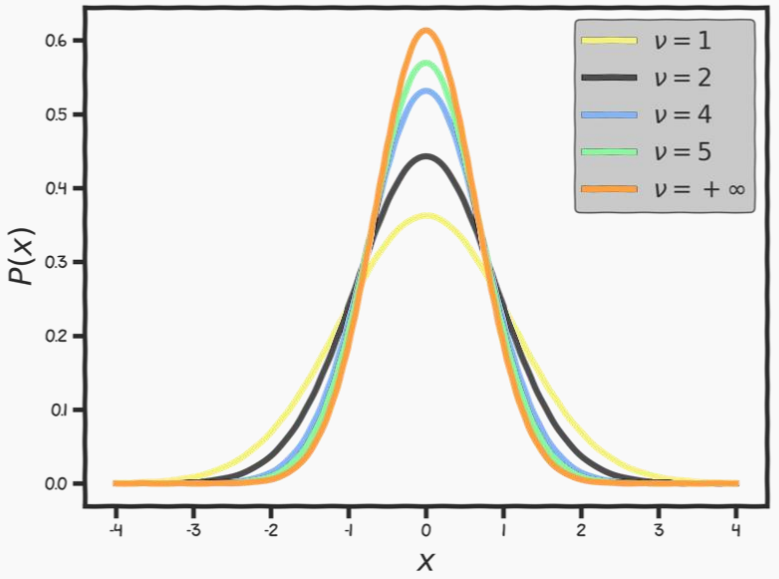
How extreme is extreme for a t -test statistic? We determine the probability of observing our test statistic, or a more extreme one, if the H_0 were true.

We call this probability the p-value:

$$p - value = P(|t_{df=n-p-1}| \geq |t - stat|)$$

Small p-value indicates that it is unlikely to observe such a substantial association between the predictor and the response due to chance. It is common to use p-value<0.05 as the threshold for significance.

To calculate the p-value we use the cumulative distribution function (CDF) of the student-t. `stats model` a python library has a build-in function `stats.t.cdf()` which can be used to calculate this.



Student's t-distribution, where ν is the degrees of freedom (number of data points minus (number of predictors + 1) = $n - (p + 1)$).

Hypothesis Testing via statsmodels

1. State Hypotheses:

$H_0: \beta_1 = 0, \qquad H_A: \beta_1 \neq 0$

2. Choose test statistic: (t-test)

3. Sample data and estimate:

$$t = \frac{\hat{\beta}_1 - 0}{\widehat{SE}\left(\hat{\beta}_1\right)} = \frac{0.5898}{0.023} = 25.211$$

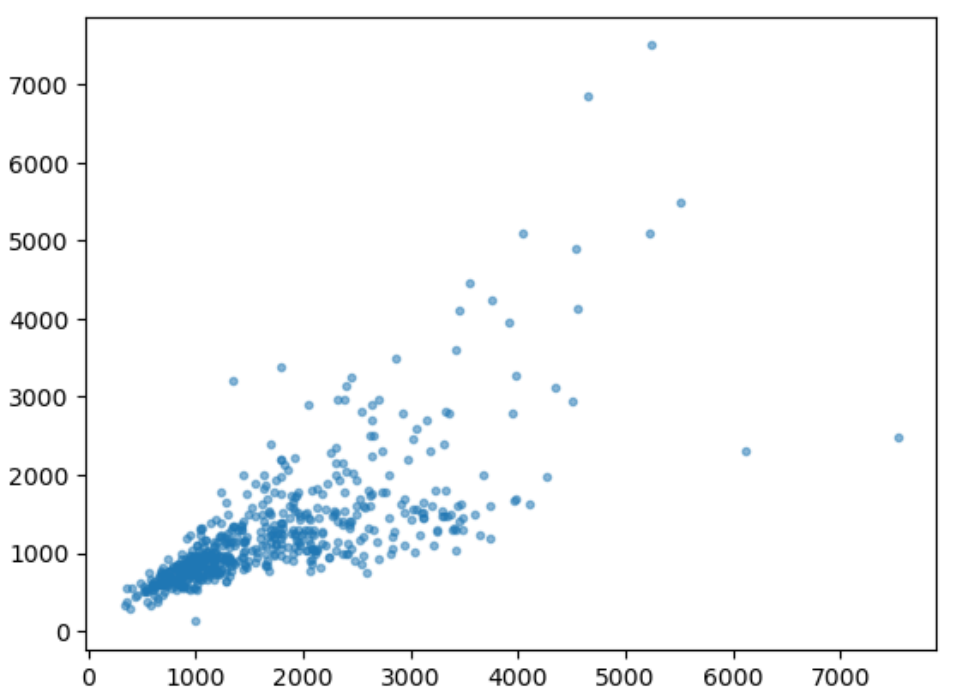
4. Reject or do not reject the H_0

$$p = P\left(\left|t_{df=n-2=590}\right| > 25.211\right) < 0.001$$

since p-value< 0.05, reject H_0

5. Restate the conclusion in context:

Evidence suggests that housing prices are truly [positively] associated with size of the home.



	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	247.4382	45.388	5.452	0.000	158.296	336.581
sqft	0.5898	0.023	25.211	0.000	0.544	0.636

Omnibus:	325.423	**Durbin-Watson:**	1.725
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	4390.598
Skew:	2.123	**Prob(JB):**	0.00
Kurtosis:	15.648	**Cond. No.**	3.95e+03

Permutation Tests: not just a side note

The **permutation test** allows us to directly use the data and statistic we care about but not have to bring along the baggage of distributional assumptions.

In classical hypothesis testing, the steps are:

1. State Hypotheses (H_0 and H_A)
2. Choose test statistic (often a t-test or z-test)
3. Sample/collect your data and estimate
4. Reject or do not reject the H_0 hypothesis
5. Restate the conclusion in context of the problem

The process for the permutation test is exactly the same, just the way the sampling distribution is built is through a form of resampling from the data rather than relying on probability theory.

Permutation Tests: the hypotheses

The most general form of the hypotheses:

H_0: The distribution of outcomes (Y) is not related to the value of X.

H_A: The distribution of outcomes (Y) is associated with the value of X.

Assumptions:

- Independence of observations.

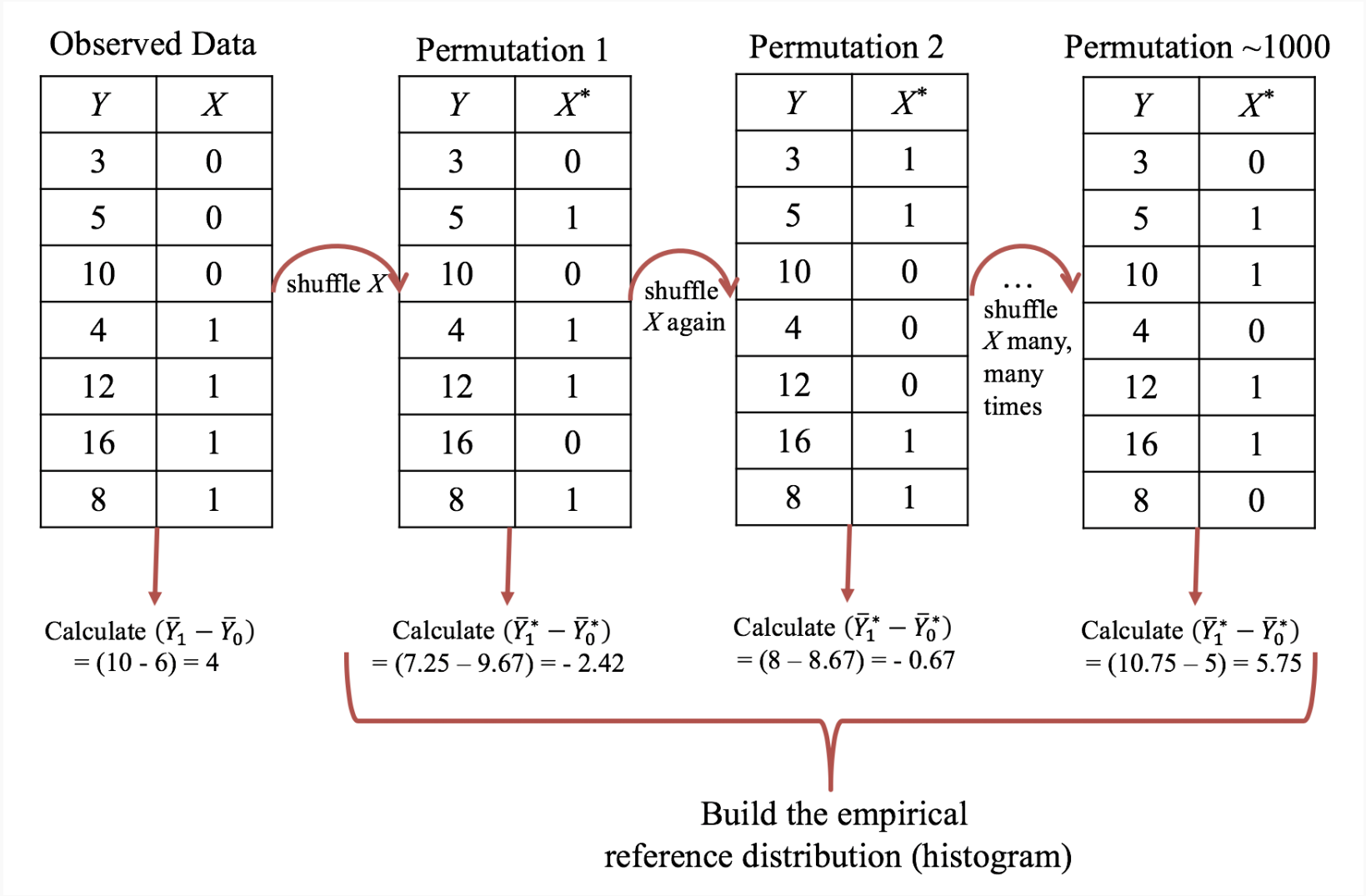
Test statistic: estimated slope, $\hat{\beta}_1$, between Y and X (or, any other statistic).

If you want a different set of hypotheses (comparison of means, comparison of medians, etc.), then that (i) will inform what test statistic to use and (ii) require you to possibly make more assumptions (similar shapes and spreads).

Permutation Tests: the main idea

- Just like in the bootstrap, the permutation test empirically creates a sampling distribution by using the data to build many potential **new** samples of data.
- But unlike the bootstrap, the permutation test relies on the null hypothesis (of **no effect** or **no difference**) when performing the resampling.
- In the two-sample case, if the null hypothesis is true, this results in **exchangeability** across observations: if the predictor variable were to change, this should have no bearing on the outcome/response measurement.
- In practice, all that you need to do is randomly 'redistribute' the responses across the predictors, thus preserving *n* but artificially forcing there to be no relationship between response (Y) and group status (X).

Permutation Tests: a diagram (for a binary predictor)



Permutation Tests: calculating the p-value

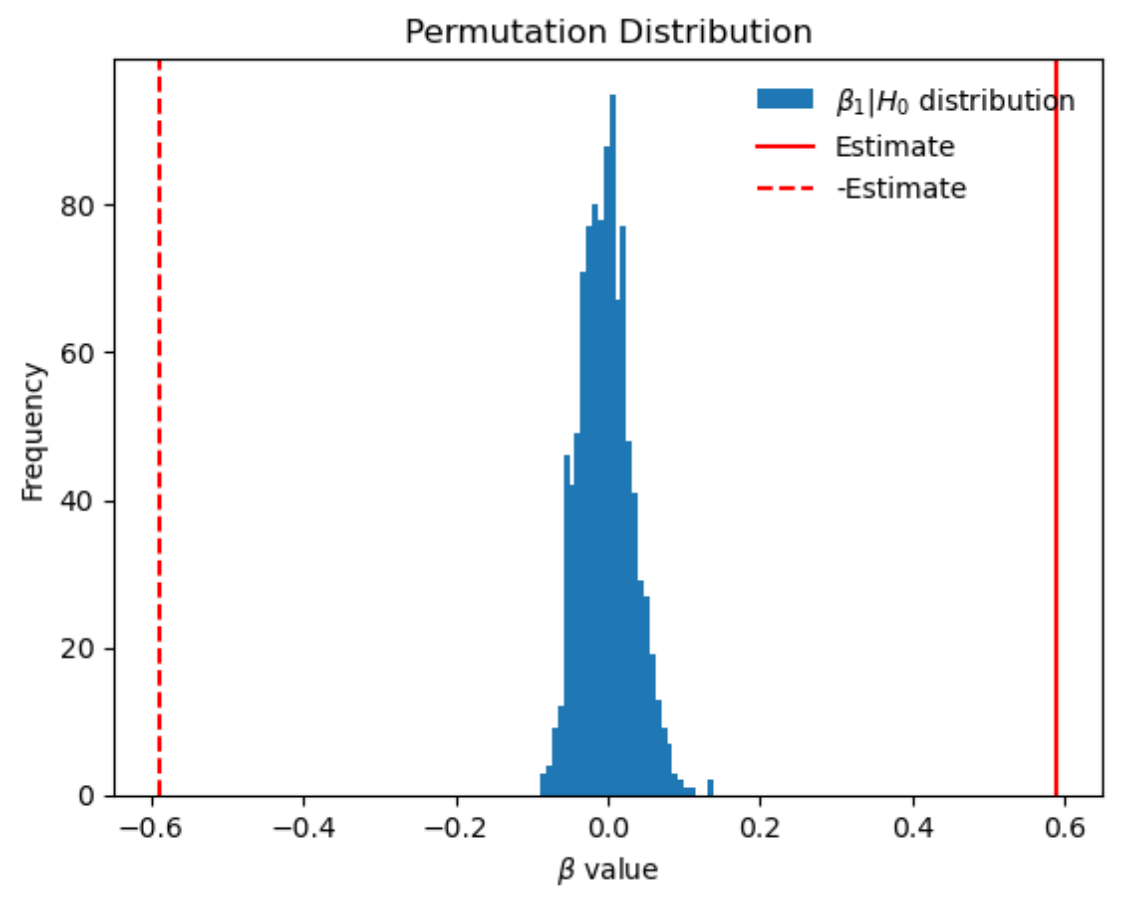
- Once the test statistic is calculated many, many times via re-permuting the observed responses across groups (via re-shuffling the group variable), then the observed statistic is compared for its **extremity** in that permutation reference/sampling distribution, in empirical probability terms. More specifically, we calculate how often the permuted statistic (calculated under the null) is more extreme than what was actually observed in the data.

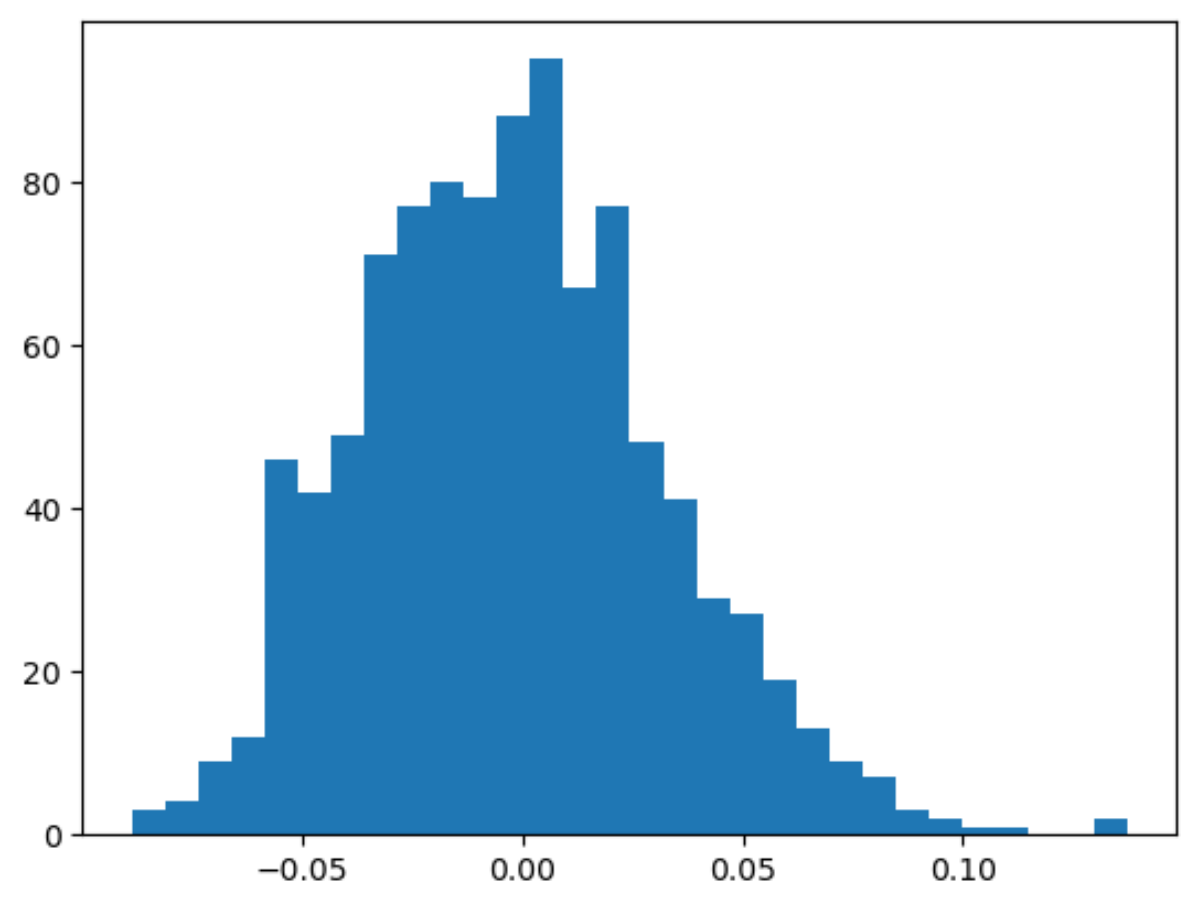
- Note: there is no distributional assumptions about the observations! Thus, the permutation test is an alternative to the -test if normality looks iffy.

Permutation Tests: an example

```
#permuations
nsims = 1000
X = homes['sqft']
y = homes[['price']]
indices = np.arange(0,len(homes))
beta1_permute = []
for i in np.arange(0,nsims):
    np.random.shuffle(indices)
    y_permute = y.iloc[indices]
    permute_ols = sk.linear_model.LinearRegression().fit(X = homes[['sqft']], y = y_permute)
    beta1_permute.append(permute_ols.coef_[0][0])
```

Permutation Tests: reference distribution





Permutation vs. bootstrap

What are the main differences between the bootstrap and permutation methods to performing resampling?

- **The goal:** bootstrapping is done to estimate (like calculate CIs) while permutation-ing is done to test a specific hypothesis.

- **The implementation:** just like in classic methods, estimation (bootstrapping) is performed without a null hypothesis and thus uses an approach the relies on whatever world the data lives in (presumably H_A), while hypothesis testing (permutation testing) is performed strictly assuming the null hypothesis is true.

That perspective on the implementation is why the bootstrapping just samples from the observed data directly (not caring about Type I or Type II error), while permutation testing carefully resamples under the null condition (thus attempting to preserve Type I error).

Note: 'inverting' a bootstrapped CI to perform testing is reasonable…but could lead to inflated (or deflated) Type I error.

Interpreting Interactions

Write out the model statement:

Interpret the coefficient estimates:

Does it appear that there is truly an interaction effect? [tricky]

```
interaction_ols = smf.ols(formula = "price ~ sqft * type",
                          data = homes).fit()
interaction_ols.summary()
```

OLS Regression Results

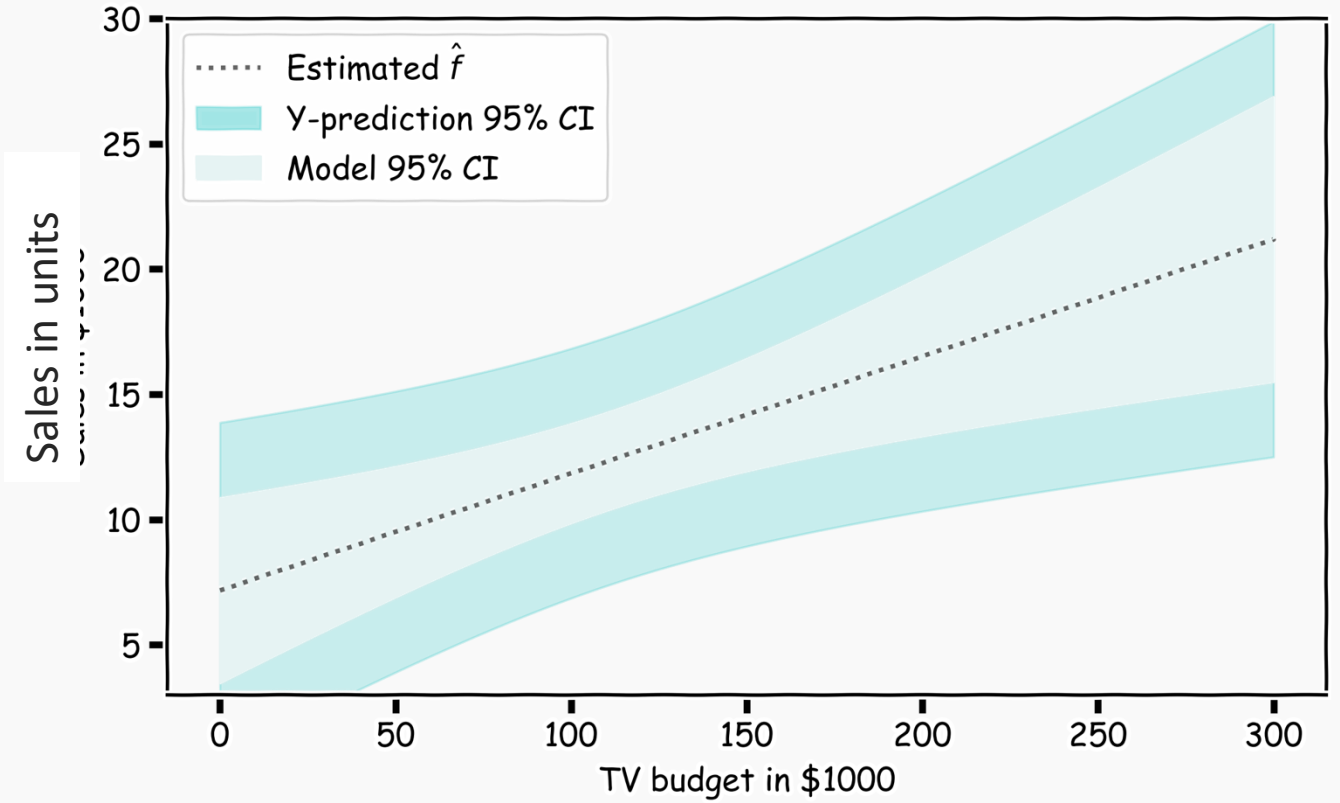
Dep. Variable:	price	**R-squared:**	0.738
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	234.5
Date:	Wed, 15 Oct 2025	**Prob (F-statistic):**	4.70e-165
Time:	07:19:03	**Log-Likelihood:**	-4386.6
No. Observations:	592	**AIC:**	8789.
Df Residuals:	584	**BIC:**	8824.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	170.5182	52.997	3.217	0.001	66.430	274.606
type[T.multifamily]	142.0626	154.846	0.917	0.359	-162.060	446.185
type[T.singlefamily]	-708.8103	111.900	-6.334	0.000	-928.586	-489.035
type[T.townhouse]	-34.2107	191.736	-0.178	0.858	-410.787	342.365
sqft	0.6659	0.040	16.516	0.000	0.587	0.745
sqft:type[T.multifamily]	-0.2863	0.062	-4.615	0.000	-0.408	-0.164
sqft:type[T.singlefamily]	0.4769	0.057	8.298	0.000	0.364	0.590
sqft:type[T.townhouse]	0.0543	0.107	0.509	0.611	-0.155	0.264

Omnibus:	196.241	**Durbin-Watson:**	1.842
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1512.812
Skew:	1.248	**Prob(JB):**	0.00
Kurtosis:	10.423	**Cond. No.**	2.56e+04

Uncertainty in predicting a new Y

- For a given x, we have a distribution of models $f(x)$
- For each of these $f(x)$, the prediction for $y \sim N(f(x), \sigma_\epsilon)$
- The prediction intervals are then …



Confidence in predicting $\hat{y}$

Even if we knew $f(x)$, the response value cannot be predicted perfectly because of the random error in the model (irreducible error).

How much will Y vary from $\hat{Y}$? We use prediction intervals to answer this question.