# Introduction to Data Science

BRIAN D'ALESSANDRO

ADJUNCT PROFESSOR, NYU

FALL 2018

*Fine Print: these slides are, and always will be a work in progress. The material presented herein is original, inspired, or borrowed from others' work. Where possible, attribution and acknowledgement will be made to content's original source. Do not distribute, except for as needed as a pedagogical tool in the subject of Data Science.*

# DISCUSSION

What do we mean by data having structure?

Why do we want to understand it?

What do we do with it?

# INFORMATION

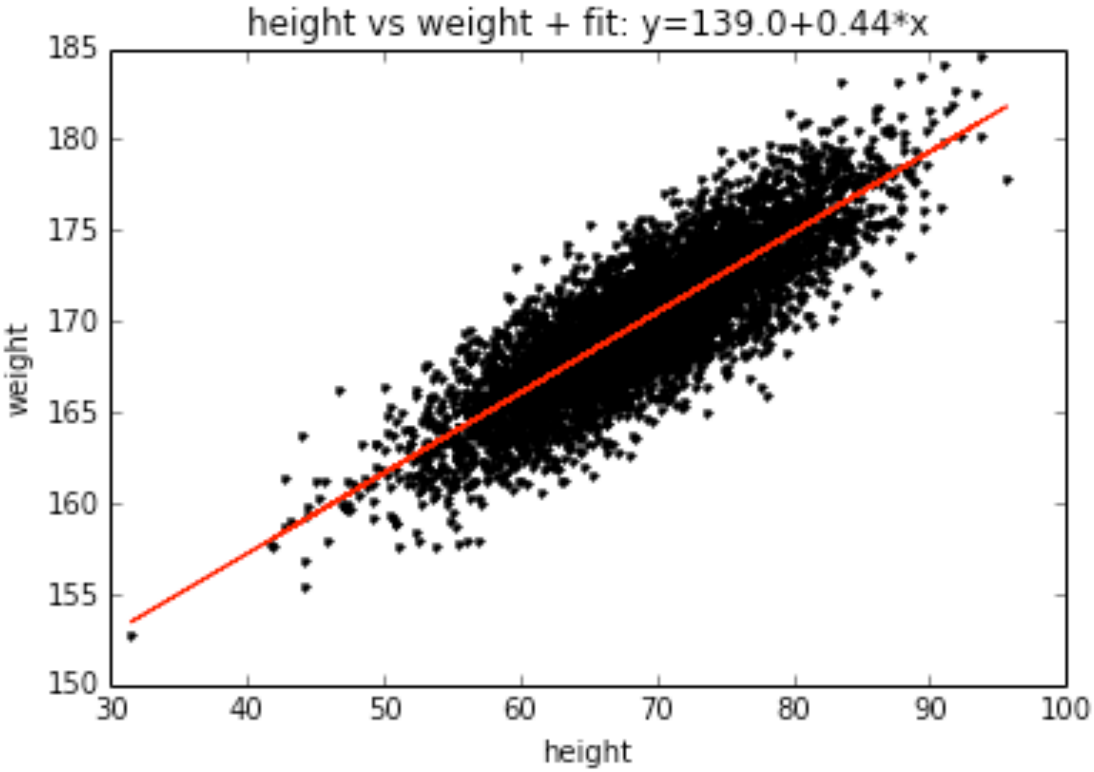
**any quantity that can reduce uncertainty about another quantity**

**If a carnival operator wanted to reduce the uncertainty about your weight, what information might he use?**

# REDUCING UNCERTAINTY

## Let’s assume our carnival friend is also a data scientist

Step 1 – Collect data, height vs. weight, regress



By regressing we can learn:

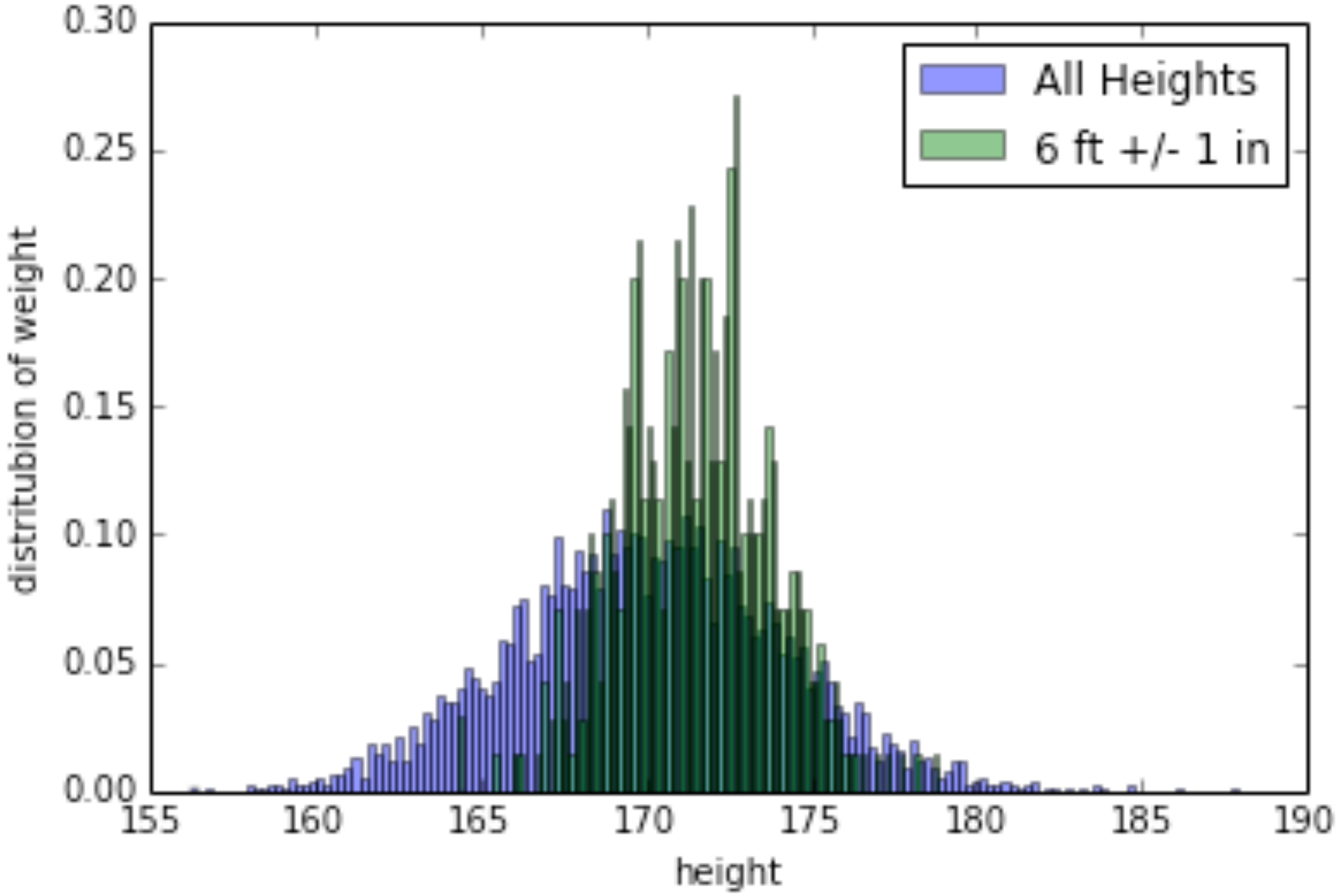
***E[Weight|Height]***

as opposed to

***E[Weight].***

# REDUCING UNCERTAINTY

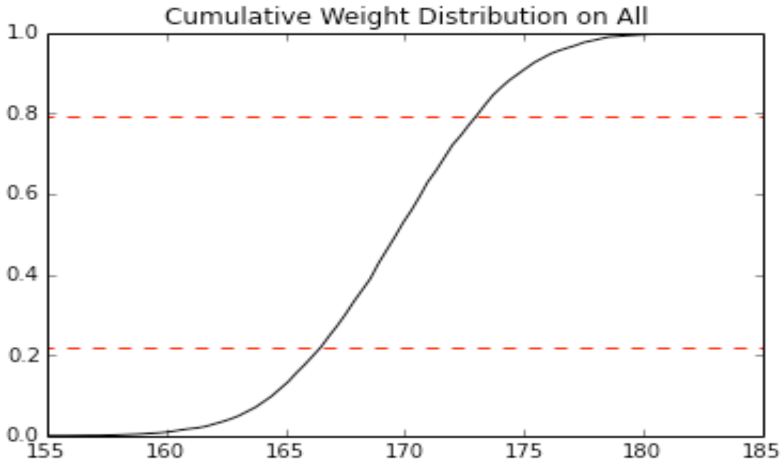
Step 2 – Based on prior knowledge, use the person's height to get a better range of possible weights.



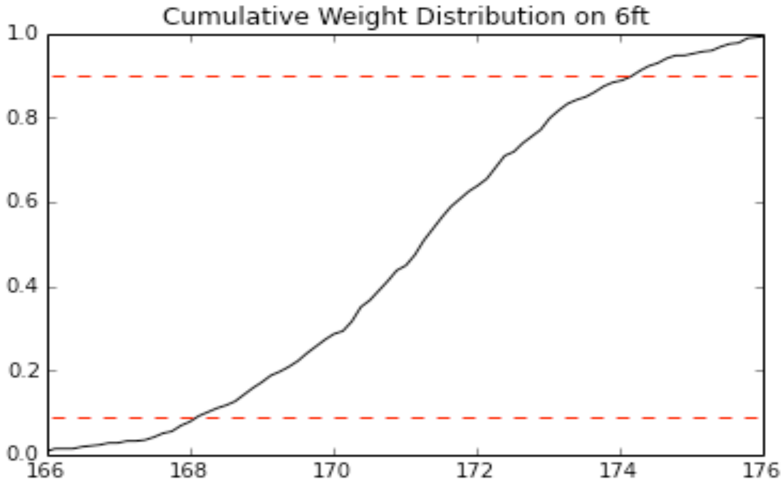
# REDUCING UNCERTAINTY

Step 3 – Guess E[W|H=72]. With a more informed, conditional expectation, he can reduce losses by 50% (40/100 vs. 20/100). That could be a lot of $$$.

***Error Rate =***
***1-P(Correct | Guess E[W])***
***= 40%***



***Error Rate =***
***P(Correct | Guess E[W]|H=72)***
***= 20%***



# EXPLORATORY ANALYSIS : FINDING STRUCTURE

NYU – Intro to Data Science

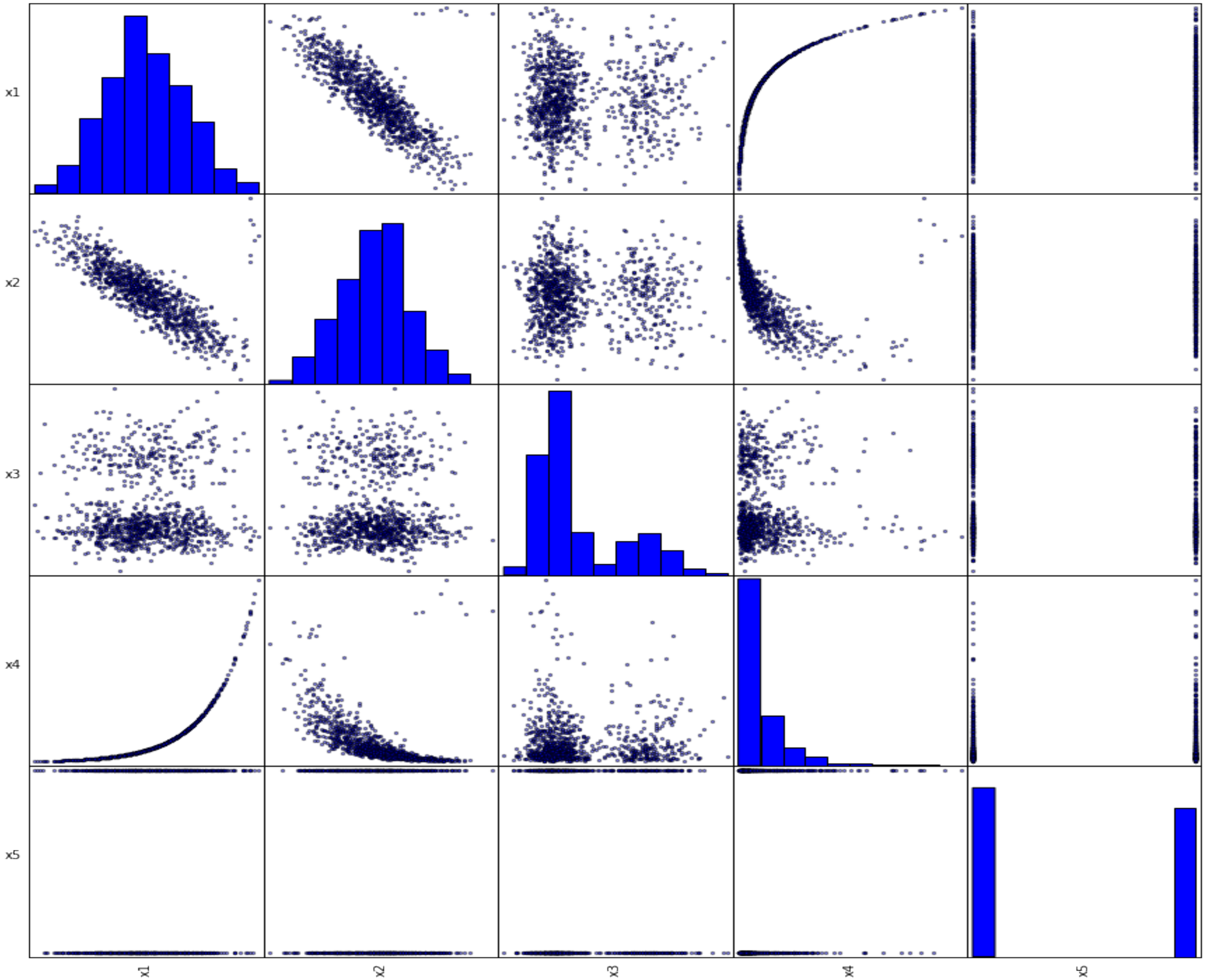
# TYPES OF STRUCTURE

**Univariate** – does a single variable follow a predictable pattern?

**Pairwise** (i.e., between 2 variables)

**Global** (i.e., across a matrix, or within a set of variables)

**Supervised** (technically includes the above, but with special emphasis on a single target variable)


x1
x2
x3
x4
x5
x1
x2
x3
x4
x5

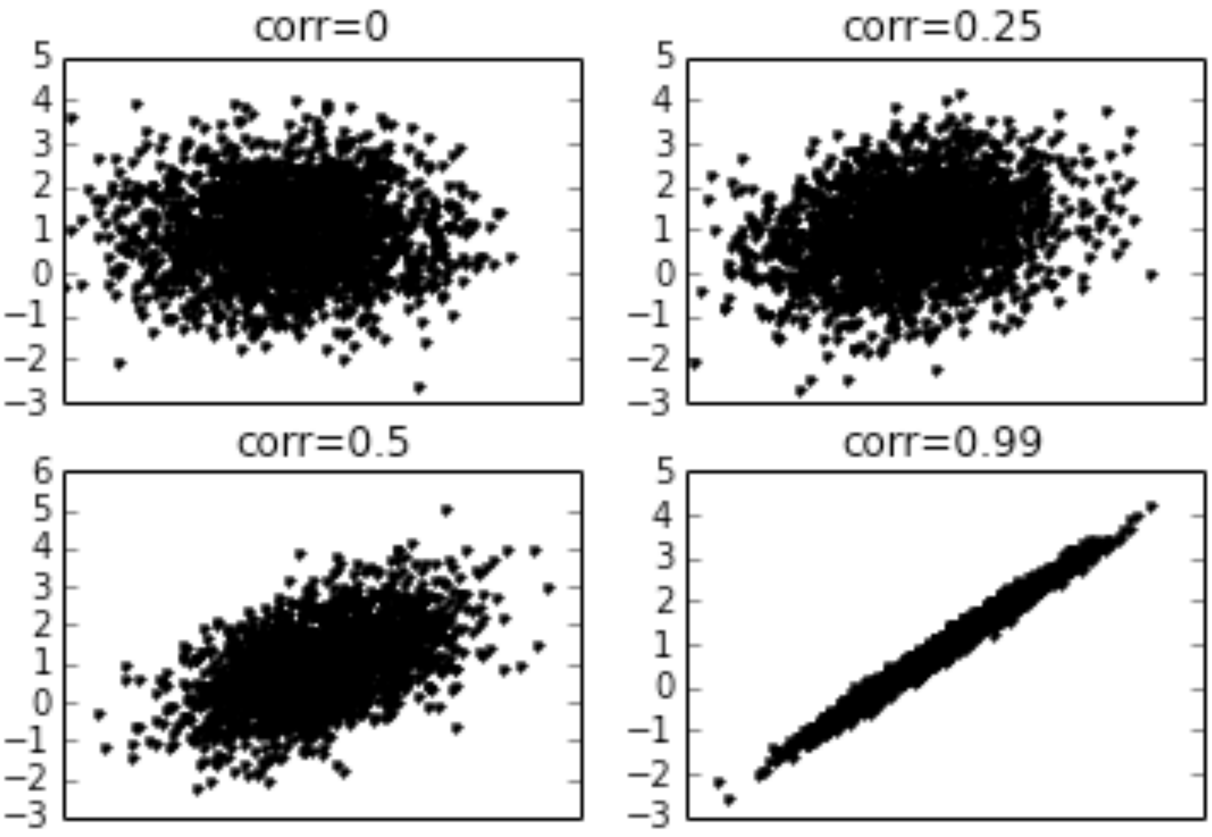
# TYPES OF STRUCTURE

**Pairwise** (i.e., between 2 variables)

# COVARIANCE AND CORRELATION

These are probably the most used and thought of metrics when considering pairwise structure. These are statistical quantities and have a fairly intuitive geometric interpretation.

Scatter of Bi-Variate Normally Distributed Variables with various Correlations



NYU – Intro to Data Science

# COVARIANCE

**Covariance**

$$\begin{aligned}\sigma(x,y) &= \mathrm{E}\left[\left(x-\mathrm{E}\left[x\right]\right)\left(y-\mathrm{E}\left[y\right]\right)\right]\\ &= \mathrm{E}\left[xy-x\,\mathrm{E}\left[y\right]-\mathrm{E}\left[x\right]y+\mathrm{E}\left[x\right]\mathrm{E}\left[y\right]\right]\\ &= \mathrm{E}\left[xy\right]-\mathrm{E}\left[x\right]\mathrm{E}\left[y\right]-\mathrm{E}\left[x\right]\mathrm{E}\left[y\right]+\mathrm{E}\left[x\right]\mathrm{E}\left[y\right]\\ &= \mathrm{E}\left[xy\right]-\mathrm{E}\left[x\right]\mathrm{E}\left[y\right].\end{aligned}$$

**Sample Covariance (between Xj and Xk)**

$$q_{jk}=\frac{1}{N-1}\sum_{i=1}^{N}\left(x_{ij}-\bar{x}_j\right)\left(x_{ik}-\bar{x}_k\right)$$

*Note: I have decided to pull equations often from wikipedia, as I feel wp represents a crowd-sourced vote on notation standardization: http://en.wikipedia.org/wiki/Covariance*

# CORRELATION

Aka: Pearson-Product Moment Correlation Coefficient. This is just the covariance normalized by the variance of each variable. This scales correlation to the interval [-1,1], which makes it a very intuitive tool for analysis and reporting.

**Correlation**

$$\rho_{X,Y} = \mathrm{corr}(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y},$$

**Sample Correlation**

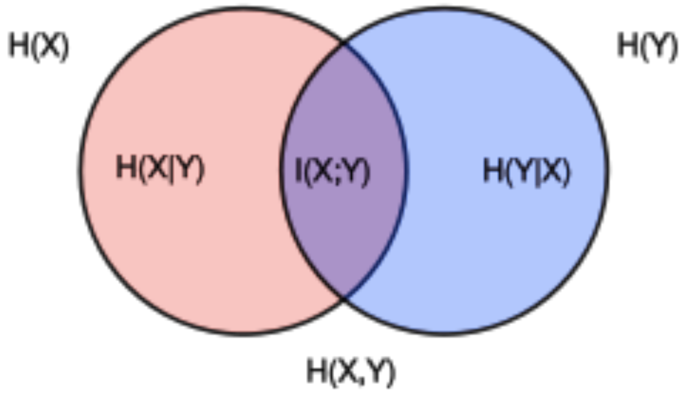
$$r_{xy} = \frac{\sum\limits_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{(n-1)s_x s_y} = \frac{\sum\limits_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum\limits_{i=1}^{n}(x_i-\bar{x})^2 \sum\limits_{i=1}^{n}(y_i-\bar{y})^2}},$$

**Almost any programming language has standard functions for these formulas, but it doesn't hurt to understand them!**

*Equation Source: http://en.wikipedia.org/wiki/Covariance*

# MUTUAL INFORMATION

This comes from Information Theory, is used often for feature importance ranking and is related to important aspects of Decision Tree algorithms.



**Mutual Information**

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right),$$

*Image and formula source: http://en.wikipedia.org/wiki/Mutual_information*

# MUTUAL INFORMATION

Let’s break this down

**Mutual Information**

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right),$$

This is in the form of E[F], i.e., E[X]=Σp(F)*F.
In this case, what is X?

NYU – Intro to Data Science

# MUTUAL INFORMATION

Let’s break this down

**Mutual Information**

$$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\boxed{\log\left(\frac{p(x,y)}{p(x)\,p(y)}\right)},$$

F=log(p(x,y)/(p(x)*p(y))….but what is this?

# MUTUAL INFORMATION

Let’s break this down

**Mutual Information**

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right),$$

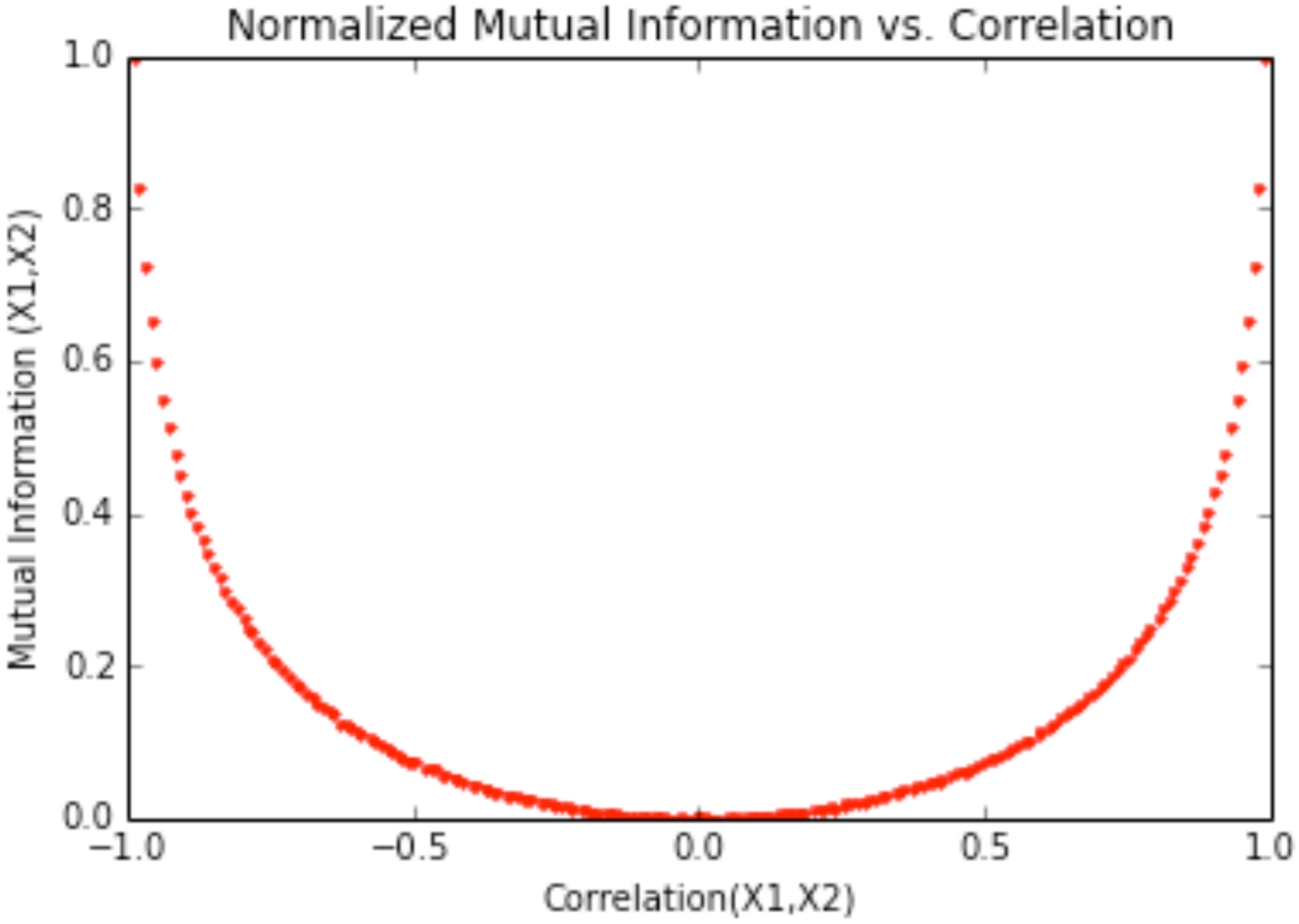
F=log(p(x,y)/(p(x)*p(y))….but what is this?

This is a quantity with the following properties:

- If X,Y are independent, F=0
- If X,Y are completely dependent, F is at a maximum
- F is symmetric (F(x,y)=F(y,x))

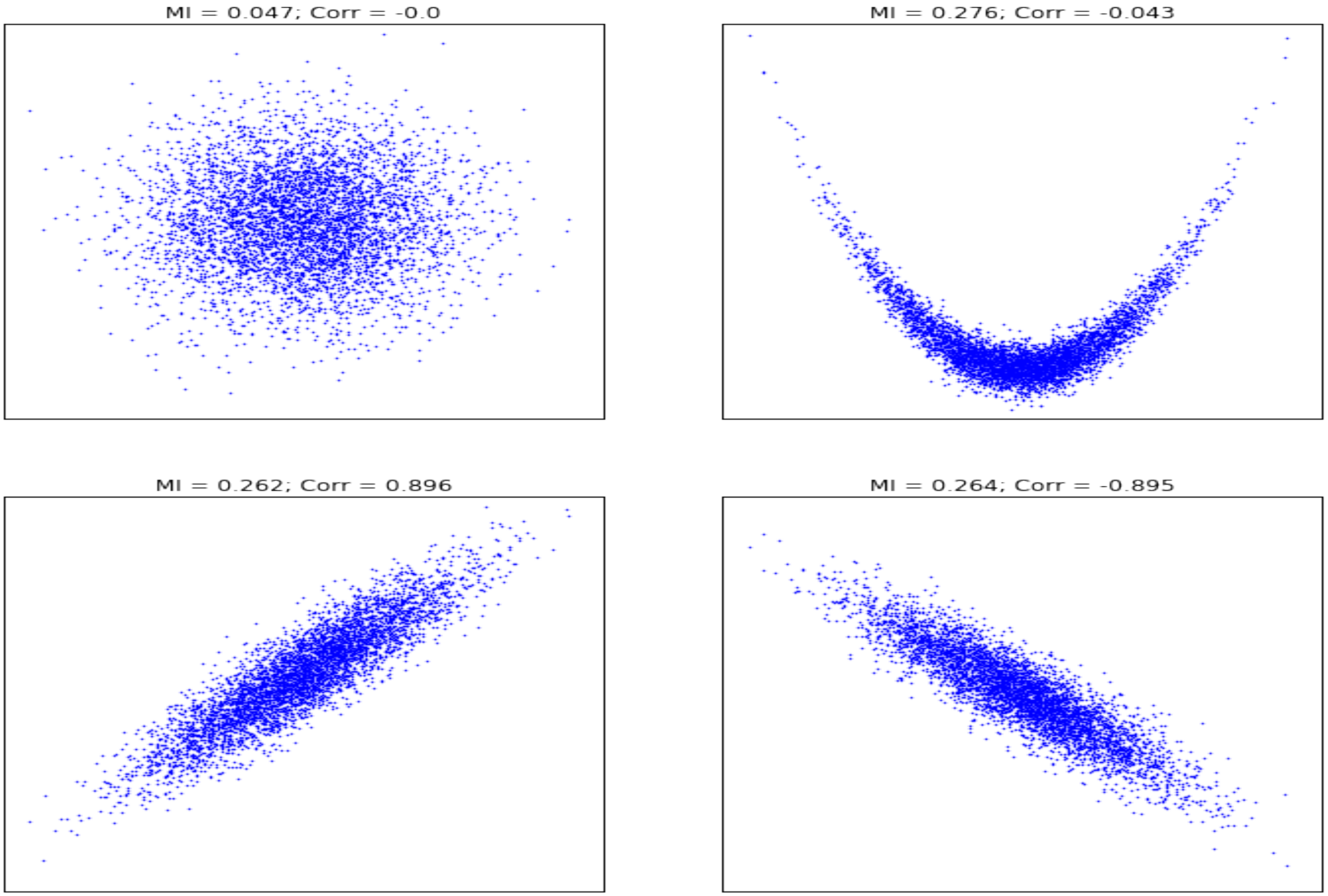
# MI VS CORRELATION

Scikit-learn has functions for MI and normalized Mutual Information. We can see that MI and correlation are monotonically related concepts, though MI is strictly positive so does not indicate negative dependencies.



# MI VS CORRELATION

We can see the difference by looking at both quantities for different types of variable dependencies.



# MI VS CORRELATION

Mutual Information

- Can capture non-linear dependencies better
- Works naturally with categorical data

Correlation Coefficient

- Expresses negative dependencies
- Well understood and intuitive (easy to communicate)

# PUTTING THESE TO USE

- General understanding of dependencies in data
- Validating assumptions for statistical modeling
- Feature ranking and selection
- Decision Tree Algorithms

# TYPES OF STRUCTURE

**Global** (i.e., across a matrix, or within a set of variables)

# DATA IS MULTIVARIATE

We usually want to go beyond pairwise similarity and understand how much “information” is actually embedded in a matrix.

We also might want to know what groups of features are related.

In an Nx2 matrix, this problem reduces to the pairwise methods just discussed.

# SINGULAR VALUE DECOMPOSITION

Although this is not a linear algebra course, this equation happens so often in data analysis that it is worth learning (again).

$$\begin{array}{ccccc} X & = & U & \Sigma & V^T \\ \mathit{NxM} & & \mathit{NxM} & \mathit{MxM} & (\mathit{MxM})^T \end{array}$$

We will not dive into all of the theoretical aspects of SVD and related topics. Our goal here is to understand it intuitively and be able to use it as a tool for solving other common Data Science problems.

# SINGULAR VALUE DECOMPOSITION

Lets go through this piece by piece:

$U$

U holds the left singular vectors.  Each row contains k elements that correspond to the latent factors of each row of X. U is orthonormal.

$\Sigma$

Σ is a diagonal matrix that holds the singular values (in descending order). The singular values are essentially weights that determine how much that latent factor contributes to the matrix.

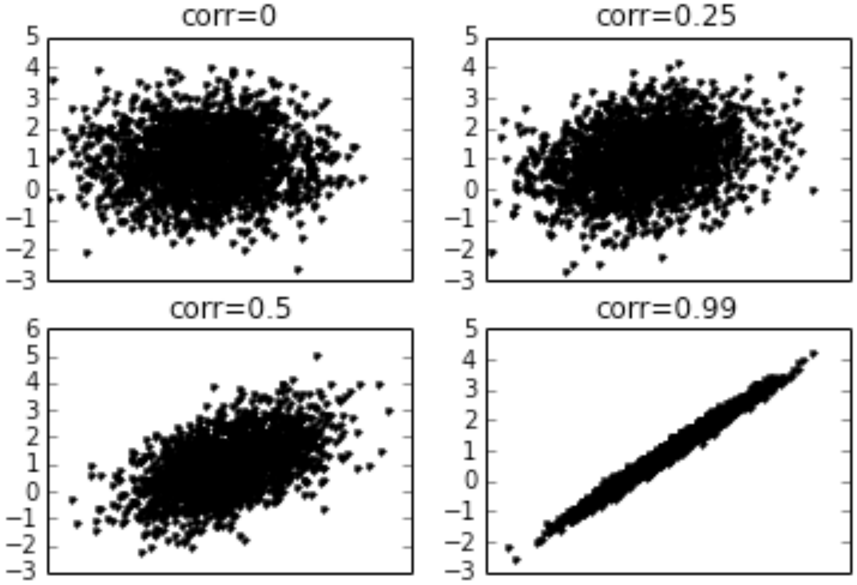
$V^T$

V holds the right singular vectors.  Each row contains k elements that correspond to the latent factors of each column of X. V is orthonormal.

# SINGULAR VALUE DECOMPOSITION

How does this relate to structure?

Lets recall these scatter plots. Each plot can be expressed as an Nx2 Matrix. The SVD is a tool that can help us understand how much information or structure is actually in the matrix.

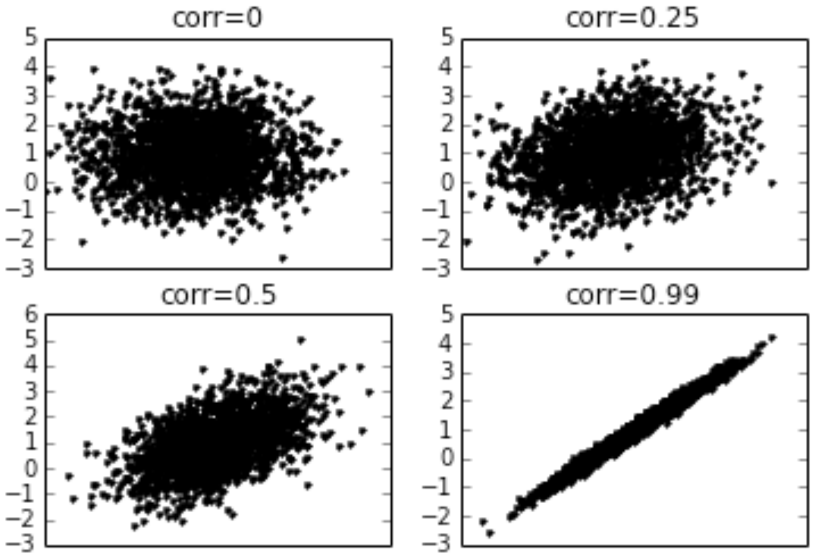


More independence of columns = more information or degrees of freedom.

Less independence of columns = less information or degrees of freedom.

# SINGULAR VALUE DECOMPOSITION

Using Scipy we can easily compute the SVD.:
U,Sig,Vt = scipy.linalg.svd(matrix)



$$\Sigma = \begin{bmatrix} 45 & \\ & 44 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 50 & \\ & 38 \end{bmatrix}$$

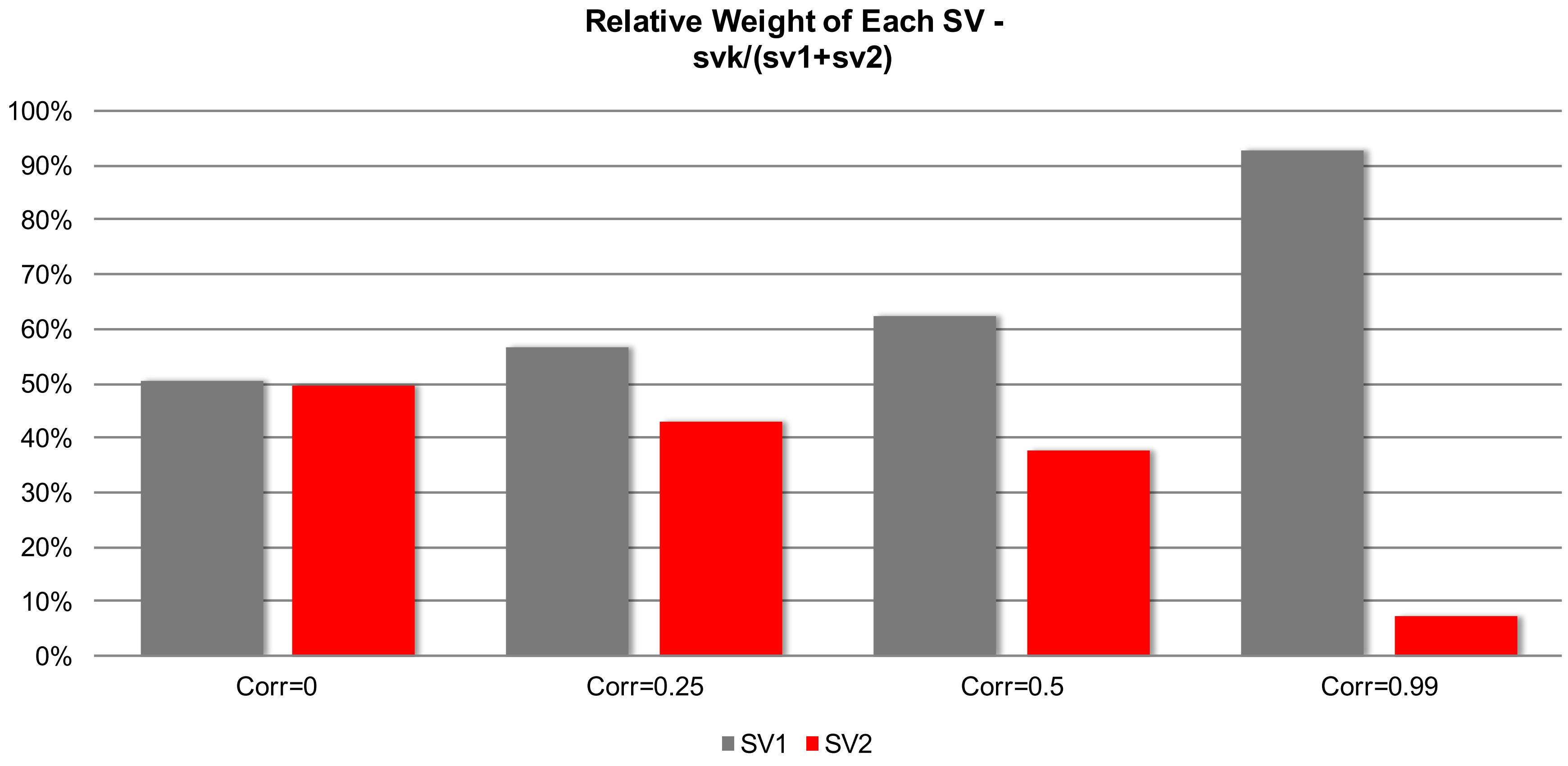
$$\Sigma = \begin{bmatrix} 53 & \\ & 32 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 62 & \\ & 5 \end{bmatrix}$$

The magnitude of the singular-values are generally determined by the magnitude of the values in X. The relative difference between singular values is a function of the level of independence of the columns.

# SINGULAR VALUE DECOMPOSITION

We can see that less independence of the columns leads to singular values with more skew from each other.



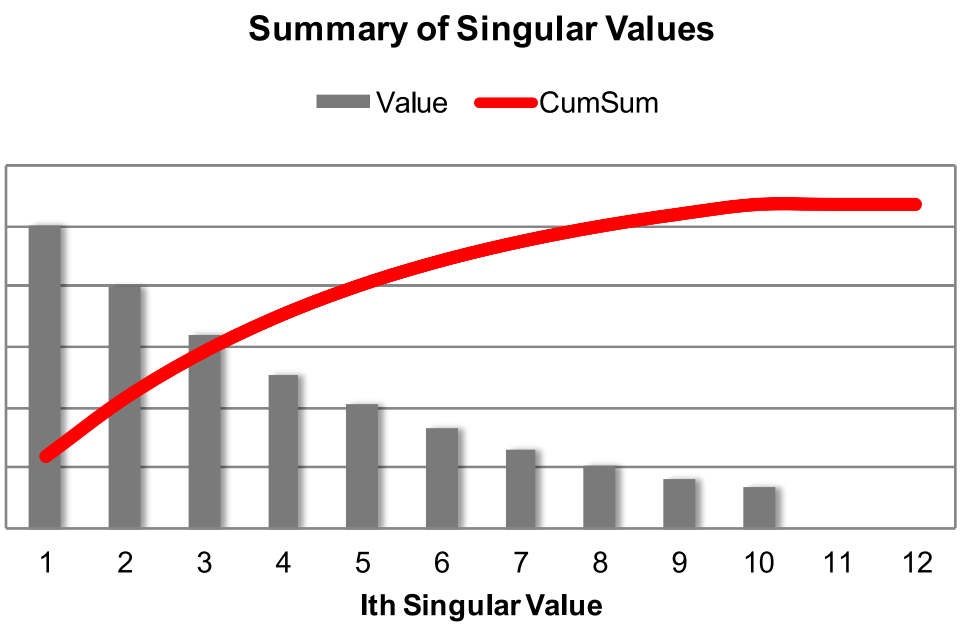
NYU – Intro to Data Science

# SINGULAR VALUE DECOMPOSITION

This idea generalizes to more dimensions, which is where the SVD is extremely useful.

$$\Sigma = \begin{pmatrix}
1 & & & & & & & & & & & \\
 & 0.8 & & & & & & & & & & \\
 & & 0.64 & & & & & & & & & \\
 & & & 0.512 & & & & & & & & \\
 & & & & 0.4096 & & & & & & & \\
 & & & & & 0.32768 & & & & & & \\
 & & & & & & 0.262144 & & & & & \\
 & & & & & & & 0.2097152 & & & & \\
 & & & & & & & & 0.16777216 & & & \\
 & & & & & & & & & 0.134217728 & & \\
 & & & & & & & & & & 0 & \\
 & & & & & & & & & & & 0
\end{pmatrix}$$

The skew of the singular values, and the shape of the cumulative sum curve can give us a sense of the degree of independence in a multi-dimensional matrix.



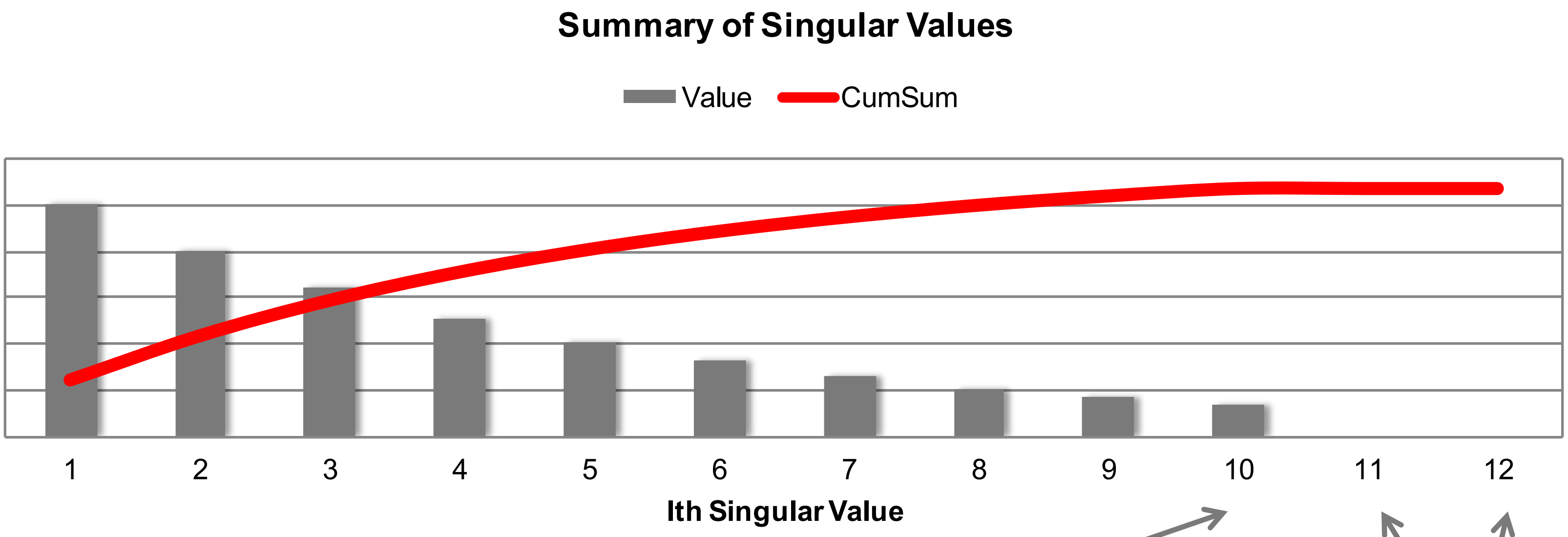
# AN INCREDIBLY USEFUL APPLICATION OF SVD

One of the most powerful applications of the SVD is creating a low rank approximation of a data matrix. Many of the following methods are based on using the SVD to get a low rank approximation.

- **Data Compression**
- **Dimensionality Reduction**
- **Recommender Systems**
- **Clustering in High Dimensions**

# THE LOW RANK APPROXIMATION

The rank of a matrix is the size of the largest number of independent columns of a matrix. The rank can be found by counting the number of singular values >0.



These SVs are greater than 0, but they are small. The add very little information to this matrix, and could even represent noise. What would happen if we set them to zero?

These SVs are zero so the rank of this matrix is 10

NYU – Intro to Data Science

# THE LOW RANK APPROXIMATION

We can build a matrix $X_k$ that approximates our original matrix by doing the following:

1. Compute the SVD of X
2. Truncate U, Σ and V to take only the k highest columns & singular values
3. Multiply back together to get $X_k$

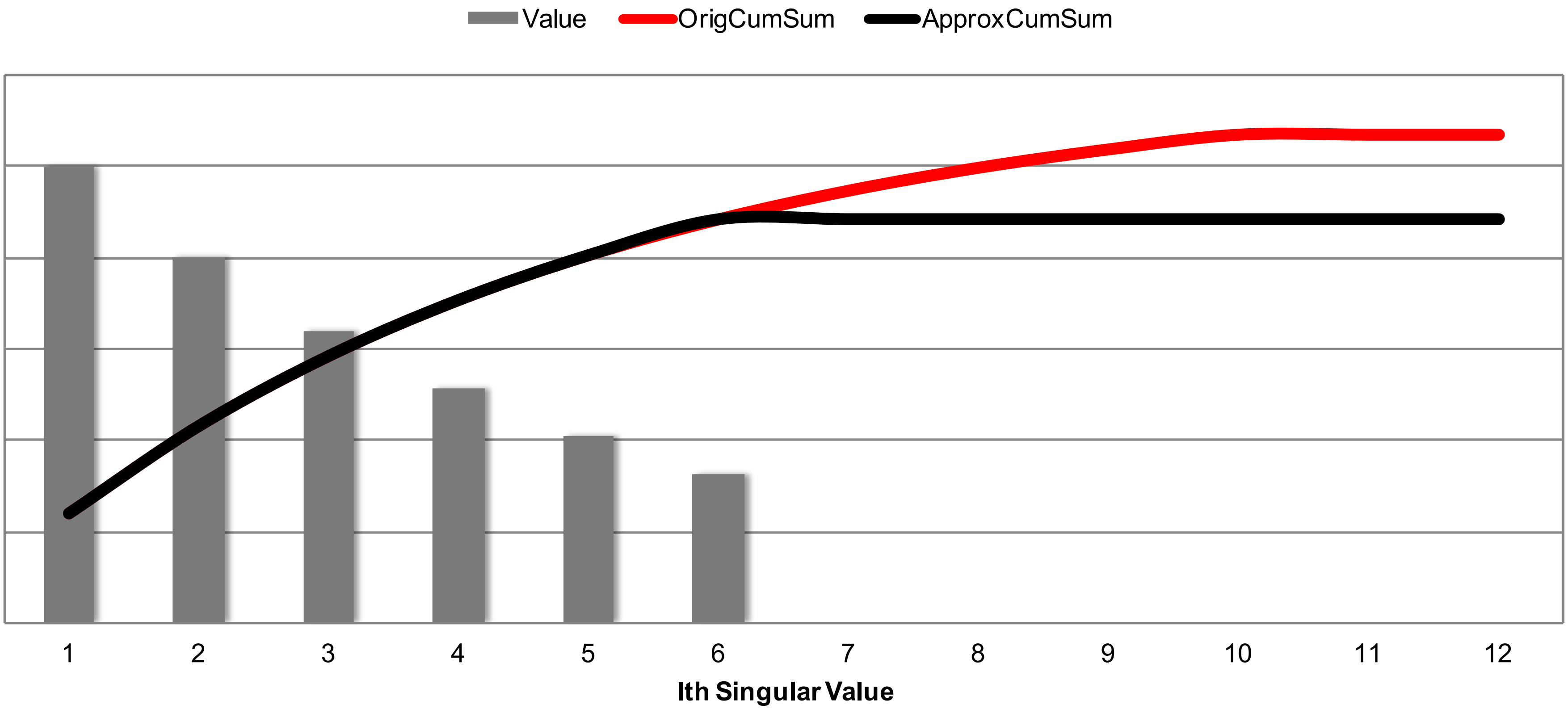
**Low Rank Approximation – K<<M**

$$\underset{NxM}{X_k} = \underset{NxK}{U_k}\,\underset{KxK}{\Sigma_k}\,\underset{(MxK)^T}{V_k^{\,T}}$$

# THE LOW RANK APPROXIMATION

Our original matrix had 12 columns with a rank of 10. In our approximation, we decided to use k=6. The cumulative sum of singular values is related to the amount of information contained in the matrix. By using k=6, we lost some information, but not half.

**What do we gain with the low rank approximation?**
**What do we lose?**



NYU – Intro to Data Science

# ONE BENEFIT OF RANK-K SVD

## Some facts

- A floating point number uses 8 bytes of memory
- An MxN matrix of floating point numbers uses 8*M*N bytes of memory.

*Instead of storing X, lets use $X_k$, a low rank approximation*

$U_k$ …this is 8*N*K bytes

$\Sigma_k$ …this is 8*1*K bytes

$V_k^T$ …this is 8*K*K bytes

→

When separated…

8*K*(N+M+1) bytes

*If you choose K<<M, you could save a ton of space…*

i.e. k=100,N=1MM,M=10k, the full X is 80 GB, whereas storing the decomposed components of $X_k$ would only be 0.8 GB!

# THE COST

This result comes from the Eckart-Young-Mirsky Theorem

If X is an nxm matrix and $X_k$ is the rank-*k* approximation derived from the SVD, then the sum-of-squares error between the entries of X and $X_k$ equals the square-root of the sum-of-squares of the singular values > k.

i.e.

$$\|X - X_k\|_F = \sqrt{\sum_{r=k+1}^{min(m,n)} \sigma_r^2}$$

Where the Frobenius norm is defined as:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}^2}$$

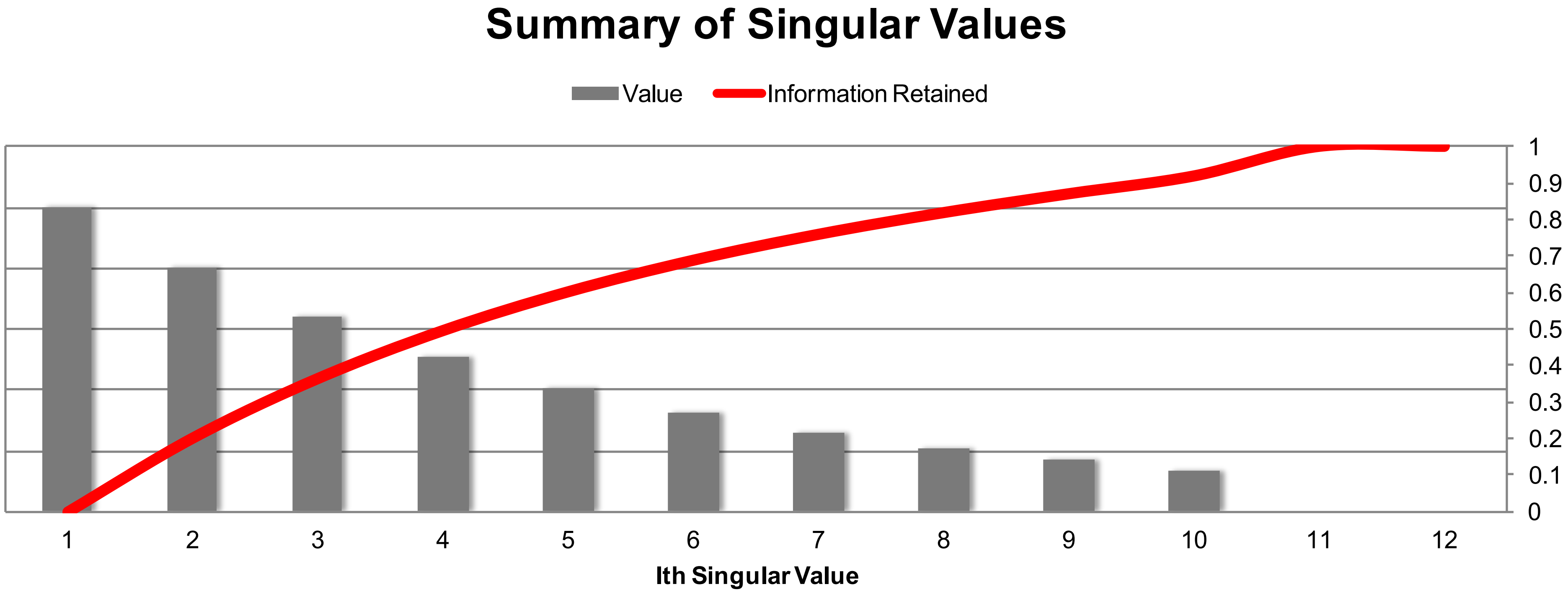
# THE COST - EXAMPLE

Given a matrix with the Singular Values shown below, the "Information Retained" shows:

$$1 - \frac{\|X - X_k\|_F}{\|X\|_F}$$

This is very similar to the $R^2$ metric in linear regression, and it tells us how well our rank-*k* approximation "fits" the the original matrix from a least squares sense.

Analyzing a rank-*k* approximation like this gives us a tool to choose *k*.



NYU – Intro to Data Science
Copyright: Brian d'Alessandro, all rights reserved

# DIMENSIONALITY REDUCTION

Sometimes a dataset has too many features for various algorithms (or storage systems to handle). Often times there is also a lot of redundancy of the data. The SVD gives us a way to reduce the number of variables without losing too much information.

Compute rank-k SVD $X_k = U_k \Sigma_k V_k^T$

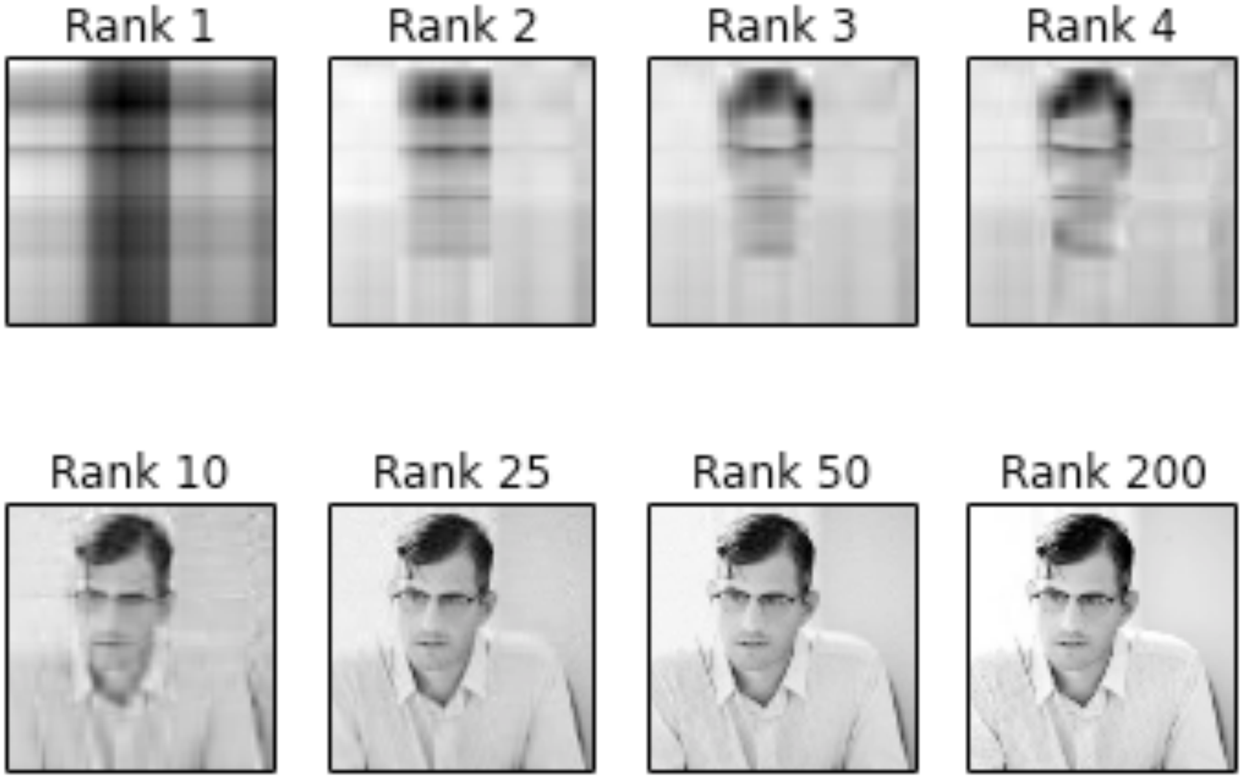
We can create an NxK reduced matrix with $X\ V_k$ This effectively becomes our new "X" matrix, and we do any analysis (clustering, modeling, etc.) with the reduced matrix.

We can project any new data into the reduced space by, $X'\ V_k$ and then use this in our algorithms.

Again, the optimal choice of K is dependent on the problem and will be a tradeoff between information loss vs. constraint tolerance.

# IMAGE COMPRESSION EXAMPLE IN PYTHON

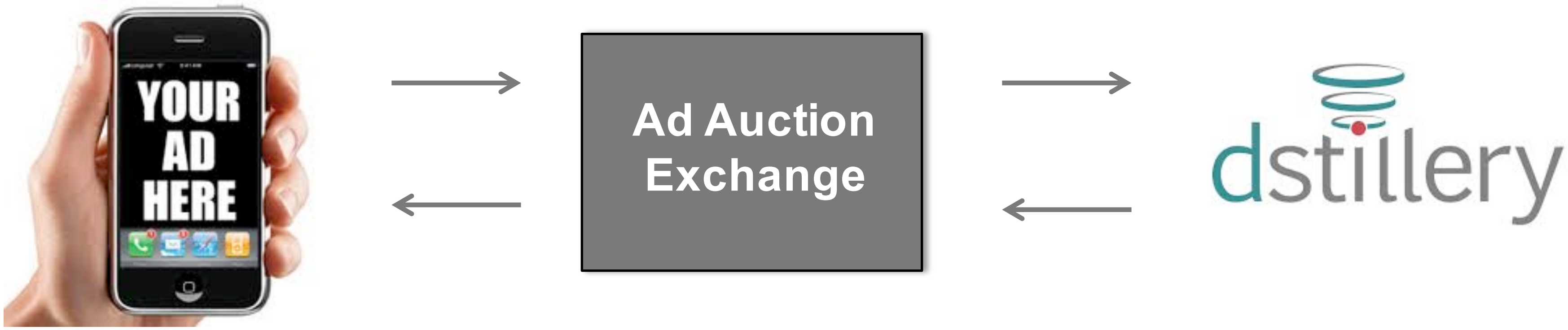
Images can be thought of as matrices of pixel color values. We can use SVD to build faces with different ranks. The full matrix here has rank=200 but we can see with a rank as low as 10 we capture most of the information we need to recognize the image.



# BUILDING AN EFFICIENT LOOKUP TABLE – THE CONTEXT

At Dstillery, we operate a mobile advertising bidding system.

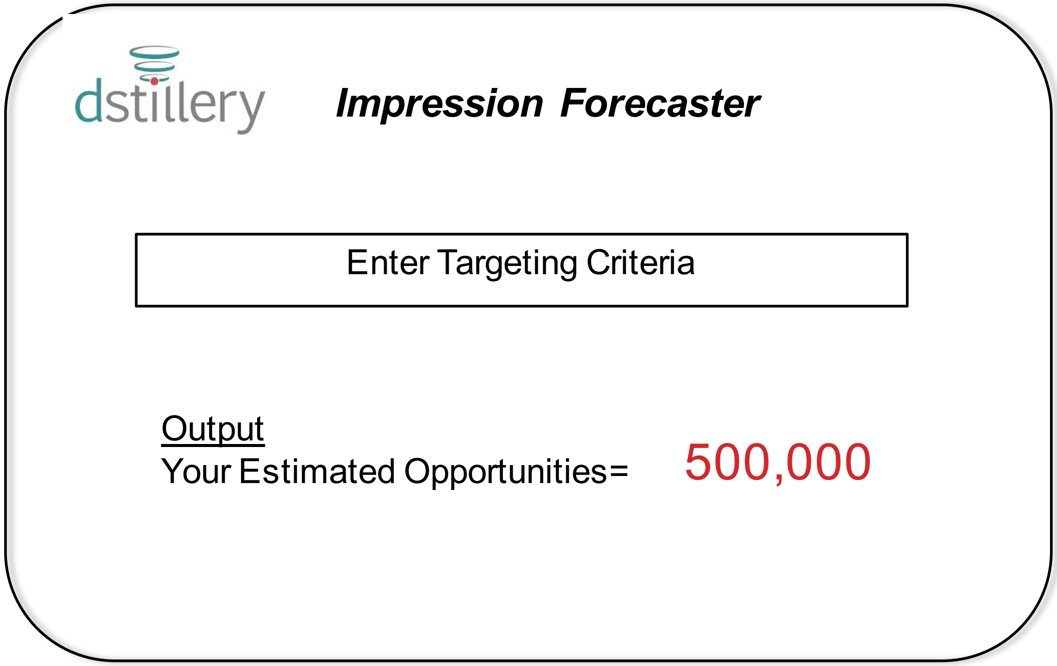
A user initiates an ad call by using an App. The Exchange receives the ad call and passes it to Dstillery as an option to bid.



If Dstillery places the highest bid, it sends an ad unit to the exchange to be displayed on the phone.

# BUILDING AN EFFICIENT LOOKUP TABLE – THE PROBLEM

Dstillery would like to provide its clients with a web application for forecasting total impression opportunities in a given period. The client would enter a set of targeting criteria (location, app name, device type, etc.) and Dstillery would provide an estimate.



dstillery

***Impression Forecaster***

Enter Targeting Criteria

Output

Your Estimated Opportunities= 500,000

*Seems simple enough, but wait!*

# BUILDING AN EFFICIENT LOOKUP TABLE – THE NAÏVE SOLUTION

An easy, brute force way to forecast is to just count the number of bid opportunities that happened in the past and use that as a forecast for the future.

| State | Zip | Census | App | Device Type | Cnt |
|---|---|---|---|---|---|
| NY | 11222 | 1001 | Candy | Phone | 678 |
| NY | 10003 | 1002 | HideMe | Phone | 77,794 |
| NJ | 08096 | 2001 | Whoa | Tablet | 78,143 |
| NJ | 08910 | 2002 | Stoopid | Phone | 1,112 |
| PA | 07919 | 3001 | Flava | Tablet | 59,787 |
| PA | 09199 | 3002 | BigStuff | Tablet | 5,000 |
| … | … | … | … | … | … |
| TX | 77450 | 9982 | Scream | Phone | 66,953 |

This is easy to accomplish in a single map-reduce query. The challenge is we end up with a table that has ~ 100 MM rows. The problem is that: **Querying a 100 MM record table with a live web interface is way too slow!**

# BUILDING AN EFFICIENT LOOKUP TABLE – EFFICIENT SOLUTION (1)

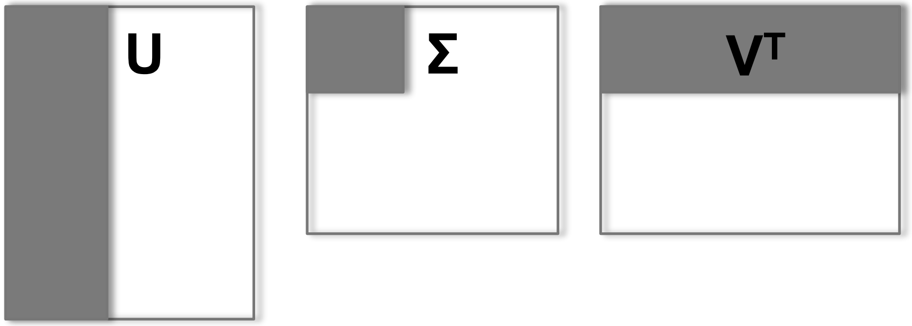
Step 1: Convert our table from a compound key-value format to a matrix.

| State | Zip | Census | App | Device Type | Cnt |
|---|---|---|---|---|---|
| NY | 11222 | 1001 | Candy | Phone | 678 |
| NY | 10003 | 1002 | HideMe | Phone | 77,794 |
| NJ | 08096 | 2001 | Whoa | Tablet | 78,143 |
| NJ | 08910 | 2002 | Stoopid | Phone | 1,112 |
| PA | 07919 | 3001 | Flava | Tablet | 59,787 |
| PA | 09199 | 3002 | BigDaddy | Tablet | 5,000 |
| … | … | … | … | … | … |
| TX | 77450 | 9982 | Scream | Phone | 66,953 |

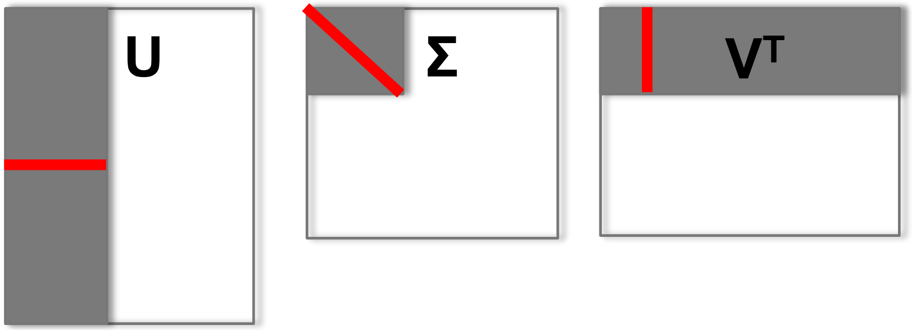
|  | App-Device 1 | App-Device 2 | … | App-Device M |
|---|---|---|---|---|
| State,Zip,Census 1 | 1000 | 10,000 | … | 90 |
| State,Zip,Census 2 | 500 | 4000 | … | 900 |
| State,Zip,Census 3 | 2400 | 34000 | … | 800 |
| State,Zip,Census 4 | 900 | 10,000 | … | 50 |
| … | … | … | … | … |
| State,Zip,Census N | 50 | 900 | … | 100 |

# BUILDING AN EFFICIENT LOOKUP TABLE – EFFICIENT SOLUTION (2)

Step 2: Use SVD to decompose the matrix…store the top k rows/columns of the decomposed matrices.



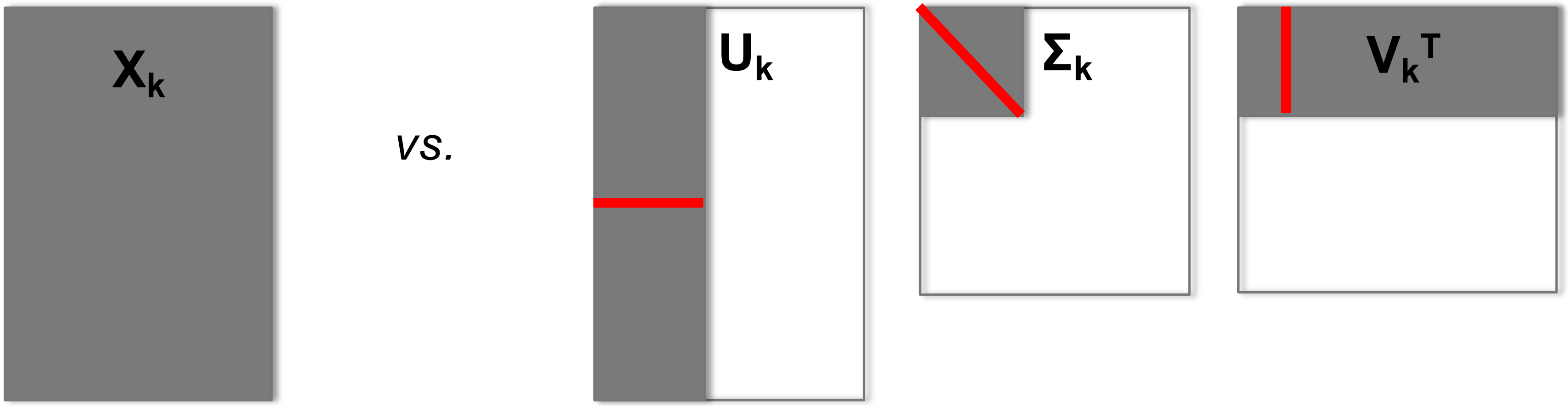
Step 3: During query time, look up appropriate row/columns from decomposed matrix. Multiply to get estimate*: Forecast* = *Σ σ$_i$*u$_i$*v$_i$*



NYU – Intro to Data Science

# BUILDING AN EFFICIENT LOOKUP TABLE – COST/BENEFIT ANALYSIS

1. The original key-value table had 100MM rows
2. For U and V we have K, 10k element vectors
3. Each scan of original table costs O(100MM)
4. Scanning U and V costs 2*K*O(10k)
5. Business requirement on accuracy is that we are right in the order of magnitude (i.e., log-scale), which means we can keep K<<10k



# TO BE CONTINUED…

Other applications of SVD are in recommender systems as well as clustering.

These will be presented in a later lecture.