

Data Pre-Processing with Python



About the Speakers

Name: Sarah Nooravi



Email: sarah@operam.com

Linkedin: <https://www.linkedin.com/in/snooravi/>

Education:

- BS Mathematics/Economics, Minor Statistics
- MS Mechanical Engineering

Current Jobs:

- Data Scientist at Operam/Fox
- DataViz Instructor at Trilogy

Fun Project: Movie Similarity using Scripts, Predicting Movie Success

About the Speakers



Name: Randy Lao

Email: randy@ideassn.org

LinkedIn: <https://www.linkedin.com/in/randylaosat/>

Education: BS Computer Science and Engineering

Current Jobs:

- Marketing Intern at IDEAS (Int'l Data Science and Engineering Association)
- DataViz Assistant Instructor at Trilogy Education
- Claims Assistant at Affiliated Physicians IPA

Fun Project: Predicting Employee Turnover

Overview



- I. Why Pre Processing?
 - A. Garbage in Garbage out!
- II. Working with Numerical Data
 - A. Data Cleaning
 - B. Data Discretization
 - C. Data Transformation
- III. Working with Categorical Data
 - A. LabelEncoding with Pandas and Sklearn
 - B. OneHotEncoding with Pandas and Sklearn
- IV. Working with Text Data
 - A. Intro to Natural Language Processing
 - B. CountVectorizer
- V. Dimensionality Reduction

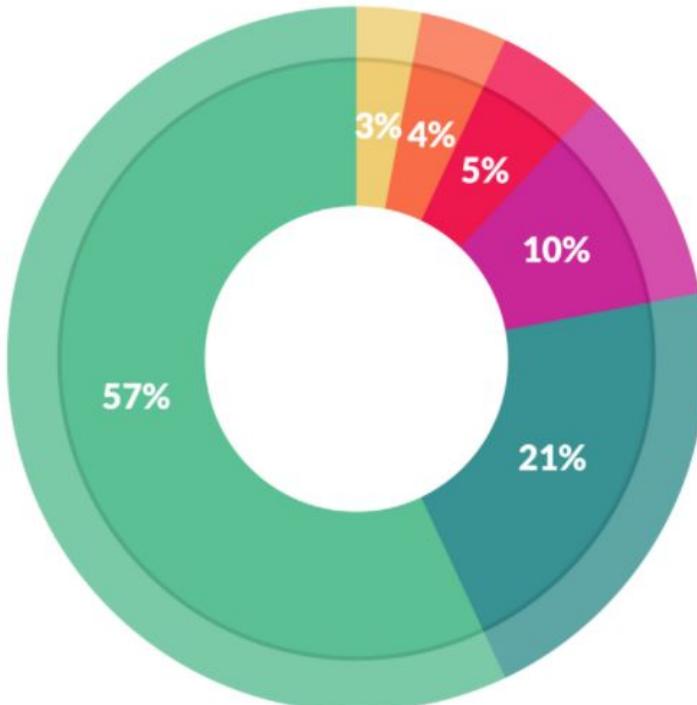
But....Real World Data is Messy

Real data is...

- Incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
- Noisy: containing errors (data entry/collection) or outliers
- Inconsistent: containing discrepancies in codes or names



What is the least enjoyable part of data science?



- *Building training sets: 10%*
- *Cleaning and organizing data: 57%*
- *Collecting data sets: 21%*
- *Mining data for patterns: 3%*
- *Refining algorithms: 4%*
- *Other: 5%*

"The Data is not going to clean itself..."

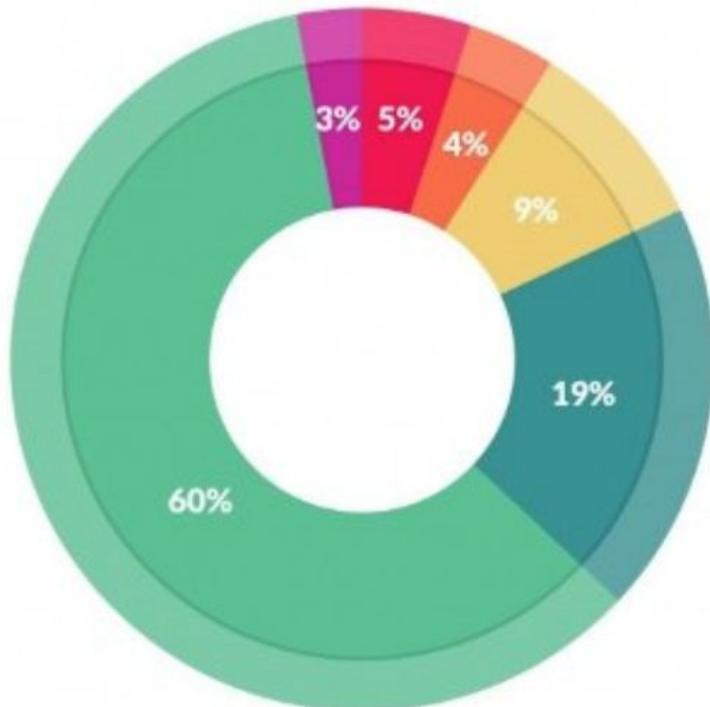
What is the least enjoyable part of data science?

CAUTION: BAD DATA



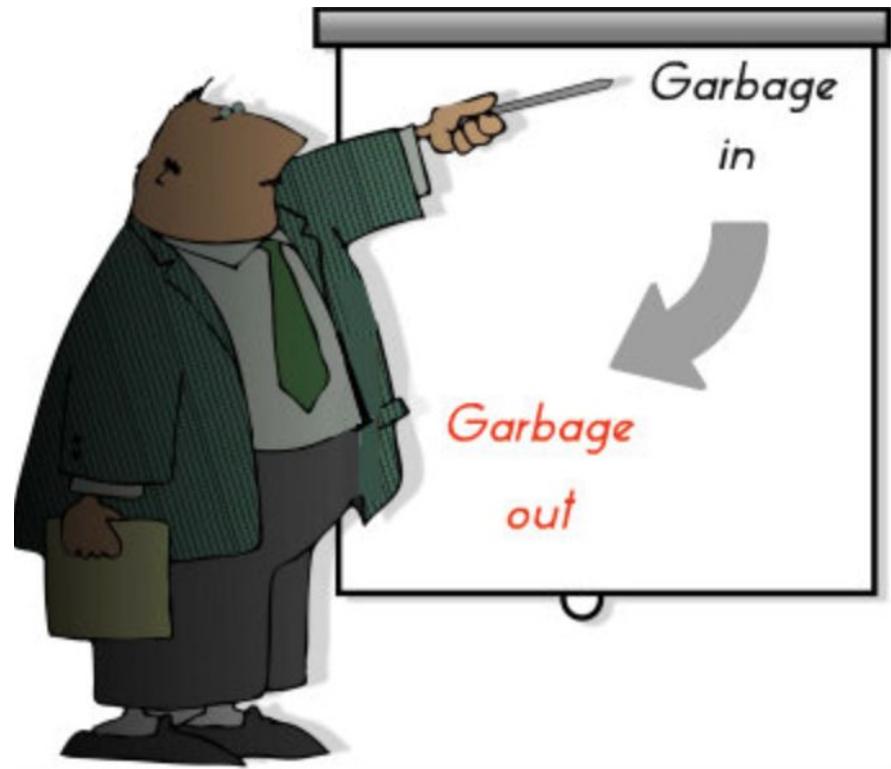
BAD DATA QUALITY
MAY RESULT IN
FRUSTRATION AND
LEAD TO DROP
KICKING YOUR
COMPUTER

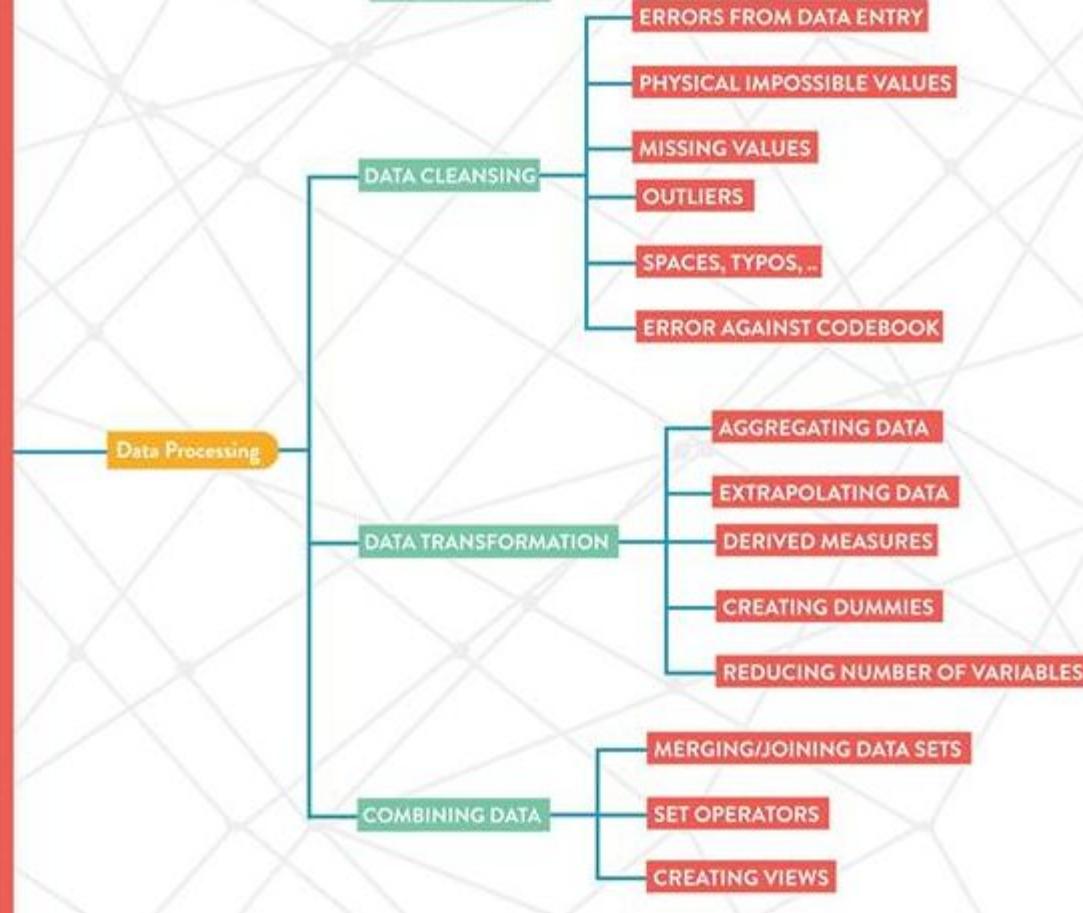
What data scientist spend the most time doing..



- *Building training sets: 3%*
- *Cleaning and organizing data: 60%*
- *Collecting data sets; 19%*
- *Mining data for patterns: 9%*
- *Refining algorithms: 4%*
- *Other: 5%*

We all know the saying...





Data Preprocessing Workflow

Types of data

Numerical data
or quantitative data

=numerical values

Discrete data

If you can count it,
then it is discrete:



Continuous data

If you can measure it,
then it is continuous:

*length



*weight



*temperature



etc

Categorical data
or qualitative data

=categories

Nominal data

If you can brand it,
then it is nominal:

Gender Colour

Female
 Male

Blue
 Red
 Green
 Orange

Ordinal data

If you can order or rank
it, then it is ordinal:

- Always
- Usually
- Sometimes
- Rarely
- Never

Working with Numerical Data

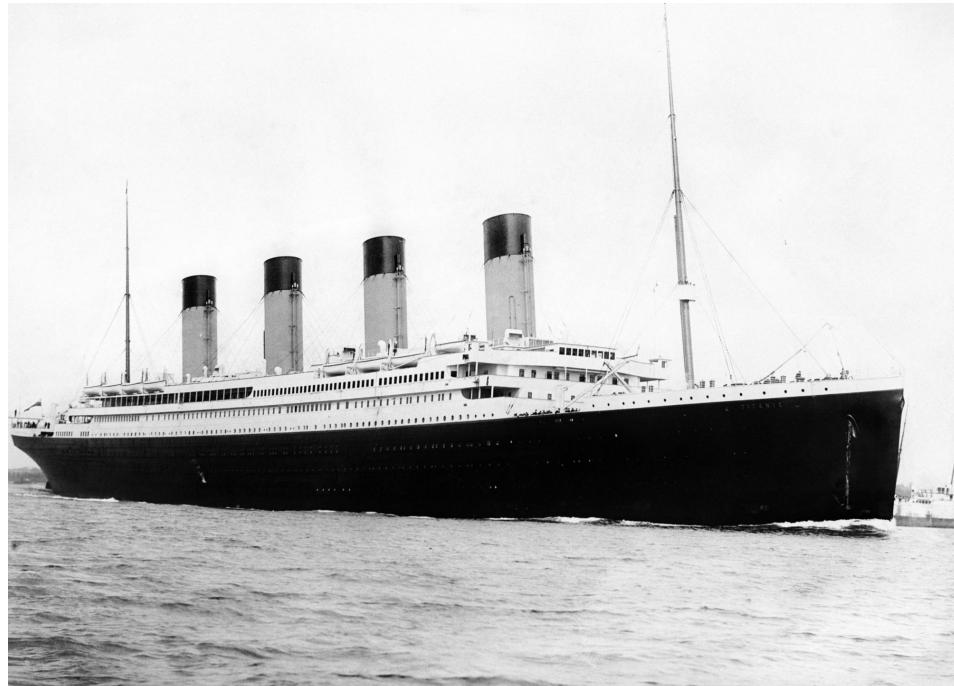
Example Dataset - Titanic

In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to **predict which passengers survived the tragedy.**

<https://www.kaggle.com/c/titanic>

Data Dictionary

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	



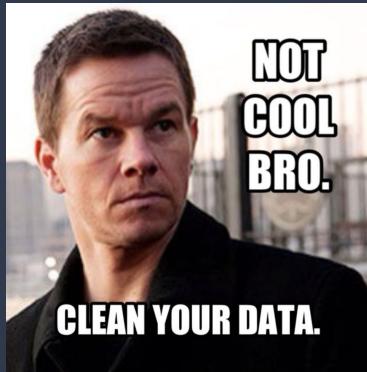
C = Cherbourg, Q = Queenstown, S = Southampton

Data Cleaning

Data cleansing plays an important role in modeling.

It's important because you will:

1. Understand your data
2. Know why those values exist
3. Know why those values are missing
4. Find relationships between features



"Clean up on Row 3" - Data Janitor

First Steps..Load and look at your data

```
# manipulating dataframes
import pandas as pd
```

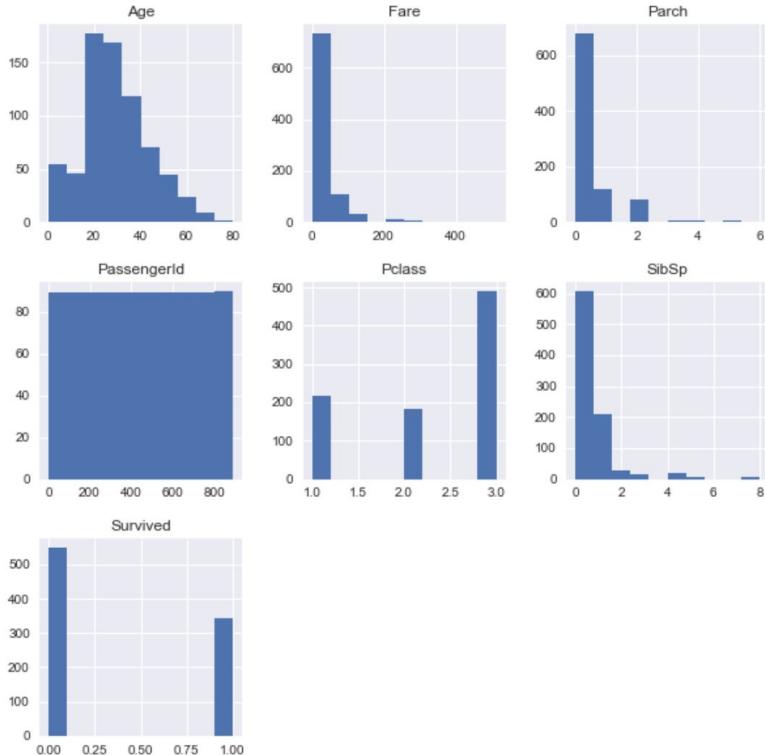
```
# visualizing libraries
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

```
titanic_train = pd.read_csv('train.csv')
```

```
titanic_train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Allen, Mr. William Henry	male	35.0	1	0	113803	53.1000	C123	S
4	5	0	3				0	0	373450	8.0500	NaN	S

First Steps..Explore and Visualize your data



`training.describe()`

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

While we are here, we can do a visual check that there are no errors or outliers.

First Steps..Get a sense for Data Types

PassengerId	Survived	Pclass			Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3		Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85	C	
2	3	1	3		female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	
4	5	0	3		Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
titanic_train.dtypes
```

```
PassengerId      int64
Survived        int64
Pclass          int64
Name            object
Sex             object
Age            float64
SibSp          int64
Parch          int64
Ticket         object
Fare            float64
Cabin          object
Embarked        object
dtype: object
```

- Numerical discrete: PassengerId, Survived, Pclass, Age, Parch
- Numerical continuous: Fare
- Categorical nominal: Name, Sex, Ticket, Embarked
- Categorical ordinal: Cabin
- Text: None

Check for Missing Data

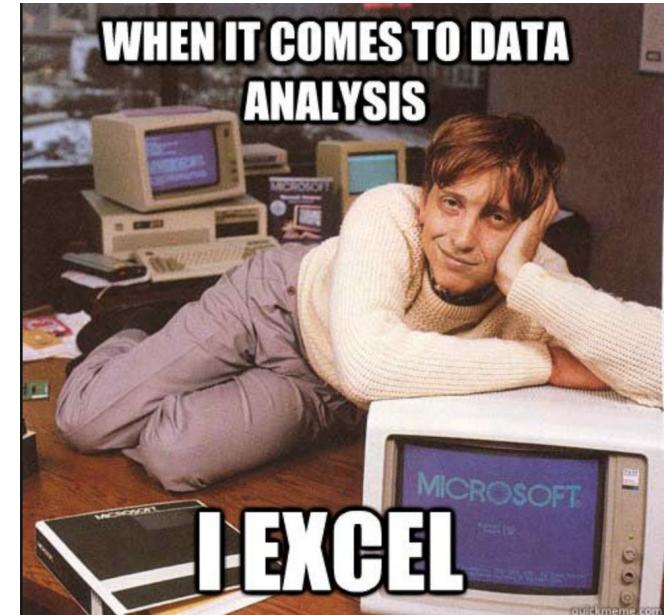
Models can not handle missing data

Simplest solution:

- Remove observations/features that have missing data

But, this might not be the ideal solution...

- If the data is “randomly” missing then you could potentially lose a lot of information from your data
- Even worse, if the data is “non-randomly” missing then you could potentially include biases
- Another approach
- Replace missing values with another value
- Tips: Impute it with either the mean, median, or frequency



Ask yourself: Why is the data missing? How much is missing?

Check for Missing Data

```
titanic_train.count()
```

```
PassengerId    891
Survived       891
Pclass          891
Name           891
Sex            891
Age            714
SibSp          891
Parch          891
Ticket         891
Fare           891
Cabin          204
Embarked       889
dtype: int64
```

Missing data:

- Age
- Cabin
- Embarked

Options:

1. Drop observations with missing values
2. Fill in with mean, median, or mode
3. Fill with random values

Handling Missing Data - Drop

```
titanic_drop_all_missing = titanic_train.dropna(how='any') # drop row if missing in any column
titanic_drop_all_missing.head()
```

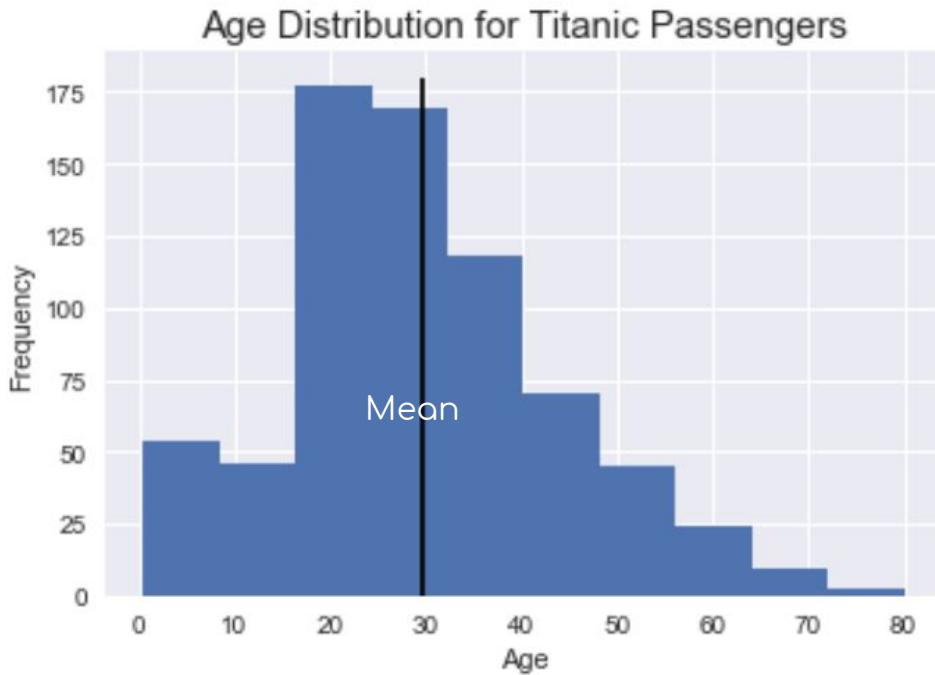
```
titanic_drop_age_missing = titanic_train.dropna(subset = ['Age'], how='all') # drop row if missing in age column
titanic_drop_age_missing.head()
```

```
print('dropping only rows with a missing age value results in', len(titanic_drop_age_missing), 'total observations')
print('dropping only rows with a missing cabin value results in', len(titanic_drop_cabin_missing), 'total observations')
print('dropping any row with a missing value results in', len(titanic_drop_all_missing), 'total observations')
```

```
dropping only rows with a missing age value results in 714 total observations
dropping only rows with a missing cabin value results in 204 total observations
dropping any row with a missing value results in 183 total observations
```

80% of our data was LOST!!!

Handling Missing Data - Impute Age



```
titanic_train.groupby(['Pclass', 'Sex'])['Age'].mean()
```

```
Pclass   Sex
1      female    34.611765
       male     41.281386
2      female    28.722973
       male     30.740707
3      female    21.750000
       male     26.507589
Name: Age, dtype: float64
```

```
titanic_train.groupby(['Pclass', 'Sex', 'Title'])['Age'].mean()
```

```
Pclass   Sex     Title
1      female  Miss    30.000000
          Mrs     40.882353
          male   Master   5.306667
          Mr      41.580460
2      female  Miss    22.390625
          Mrs     33.682927
          male   Master   2.258889
          Mr      32.768293
3      female  Miss    16.123188
          Mrs     33.515152
          male   Master   5.350833
          Mr      28.724891
Name: Age, dtype: float64
```

Detour - Feature Engineering

```
def title(x):
    if 'Mr.' in x:
        return 'Mr'
    elif 'Mrs.' in x:
        return 'Mrs'
    elif 'Master' in x:
        return 'Master'
    elif 'Miss.' in x:
        return 'Miss'
```

```
titanic_train['Title'] = titanic_train.Name.apply(title)
titanic_train.Title.value_counts()
```

```
Mr      517
Miss    182
Mrs     125
Master   40
Name: Title, dtype: int64
```

For this dataset, we are able to engineer at least three new features, namely: title, family size and single rider.

```
titanic_train['FamilySize'] = titanic_train['SibSp'] + titanic_train['Parch'] + 1
titanic_train.FamilySize.value_counts()
```

```
1      537
2      161
3      102
4       29
6       22
5       15
7       12
11      7
8       6
Name: FamilySize, dtype: int64
```

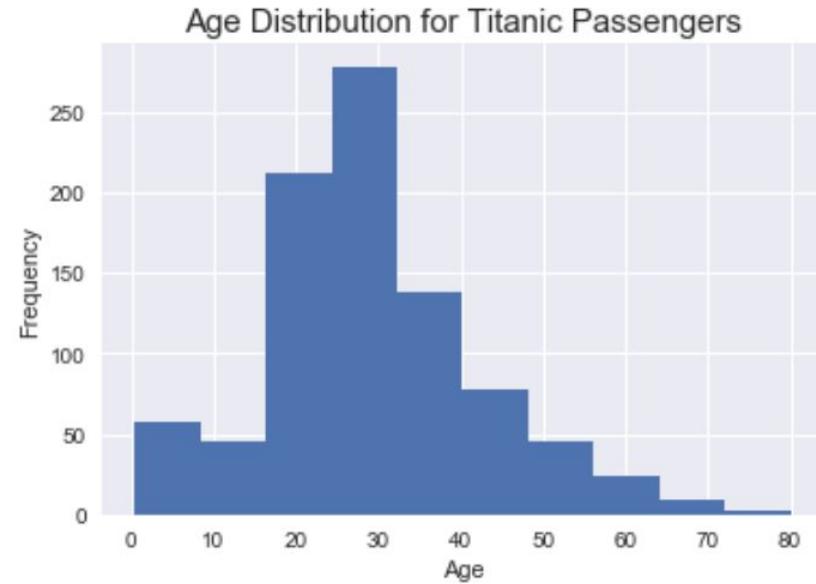
```
titanic_train['IsAlone'] = 0
titanic_train.loc[titanic_train['FamilySize'] == 1, 'IsAlone'] = 1
titanic_train.IsAlone.value_counts()
```

```
1      537
0      354
Name: IsAlone, dtype: int64
```

Handling Missing Data - Impute Age

```
def impute_age(dataset):
    for pclass in [1,2,3]:
        for sex in ['male','female']:
            for title in ['Miss', 'Mr', 'Master', 'Mrs']:
                ds = dataset[dataset['Pclass'] == pclass]
                ds = ds[ds['Sex'] == sex]
                ds = ds[ds['Title'] == title]
                median = ds['Age'].median()
                dataset.loc[
                    (dataset['Age'].isnull()) &
                    (dataset['Pclass'] == pclass) &
                    (dataset['Title'] == title) &
                    (dataset['Sex'] == sex),
                    'Age'] = median
```

```
impute_age(titanic_train)
```



Detour 2 - Data Discretization (Binning)

After we have imputed all the age values, we can use a technique called '**binning**' that can help users place values into groups so as to allow for more vigorous customization of datasets.

```
age_bins = [0,18,120]
age_labels = ['minor', 'adult']

for dataset in full_dataset:
    dataset['Child'] = pd.cut(dataset['Age'], age_bins, labels=age_labels)
titanic_train.Child.value_counts()

adult      715
minor     176
Name: Child, dtype: int64
```

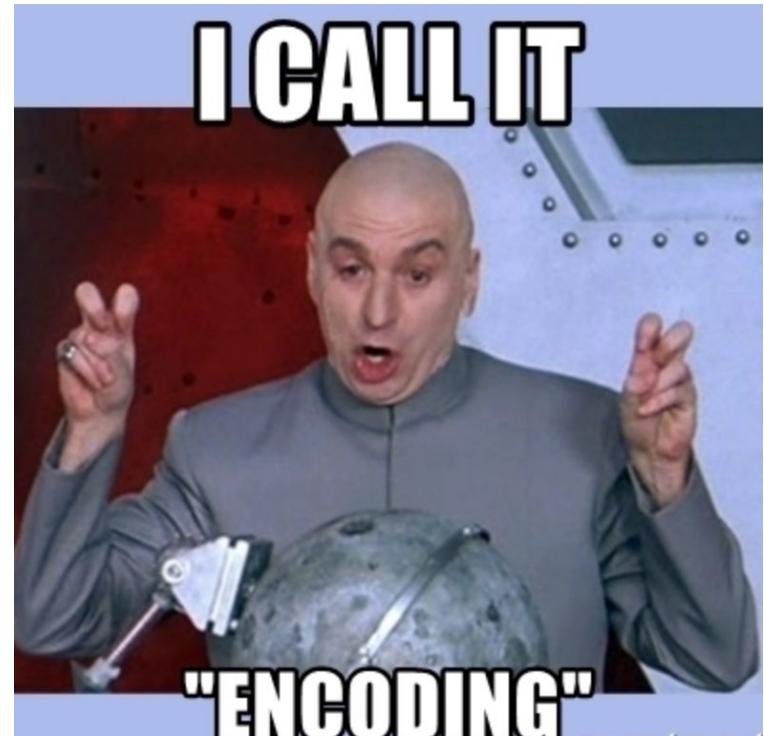
Encoding Categorical Variables

Why Encoding?

Models can only handle numeric features!

Must convert categorical and ordinal features into numeric features:

- Create dummy features
- Transform a categorical feature into a set of dummy features, each representing a unique category
- In the set of dummy features, 1 indicates the observation belongs to that category and 0 otherwise.



Approach 1 - Find and Replace

```
titanic_train.Sex.value_counts()
```

```
male      577  
female    314  
Name: Sex, dtype: int64
```

```
# using map function  
for dataset in full_dataset:  
    dataset['Sex'] = dataset['Sex'].map({'female': 0, 'male': 1}).astype(int)
```

```
titanic_train.Sex.value_counts()
```

```
1      577  
0      314  
Name: Sex, dtype: int64
```

```
cleanup_nums = {'Child': {'Adult': 0, 'Minor': 1},  
                'Mother': {'Mother': 0, 'Not Mother': 1}}
```

```
# using replace  
for dataset in full_dataset:  
    dataset.replace(cleanup_nums, inplace=True)
```

If you review the [replace documentation](#), you can see that it is a powerful command that has many options. For our uses, we are going to create a mapping dictionary that contains each column to process as well as a dictionary of the values to translate.

Approach 2 - Label Encoding using Pandas

```
titanic_train.Embarked.value_counts()
```

```
S     646  
C     168  
Q      77  
Name: Embarked, dtype: int64
```

```
for dataset in full_dataset:  
    dataset[ "Embarked" ] = dataset[ "Embarked" ].astype( 'category' )  
titanic_train.dtypes
```

```
PassengerId      int64  
Survived         int64  
Pclass           int64  
Name            object  
Sex              int64  
Age             float64  
SibSp            int64  
Parch            int64  
Ticket          object  
Fare             float64  
Cabin          object  
Embarked        category  
Title            int64  
Mother           int64  
FamilySize       int64  
IsAlone          int64  
Child            int64  
dtype: object
```

```
for dataset in full_dataset:  
    dataset[ "Embarked" ] = dataset[ "Embarked" ].cat.codes  
titanic_train.Embarked.value_counts()
```

```
2     646  
0     168  
1      77  
Name: Embarked, dtype: int64
```

Label encoding is simply converting each value in a column to a number. This results in the same output as the find and replace method (Approach 1).



Approach 2 - LabelEncoder with Sklearn

```
from sklearn.preprocessing import LabelEncoder
```

```
titanic_train.Title.value_counts()
```

```
Mr      517  
Miss    182  
Mrs     125  
Master   40  
Other    27  
Name: Title, dtype: int64
```

```
title_enc = LabelEncoder()  
titanic_train['Title'] = title_enc.fit_transform(titanic_train['Title'].astype(str))  
titanic_test['Title'] = title_enc.transform(titanic_test['Title'].astype(str))
```

```
titanic_train.Title.value_counts()
```

```
2      517  
1      182  
3      125  
0       40  
4       27  
Name: Title, dtype: int64
```

Sklearn Workflow:

1. Import
2. Instantiate
3. Fit
4. Transform

Advantages and Disadvantages

LabelEncoder:

The approaches mentioned have the advantage of being straightforward and easy to implement.

However, the numeric values can be “misinterpreted” by algorithms. For example, the value of 0 is obviously less than the value of 3 but does that really mean that being a “Mrs” must have 3X more weight in our calculation than being a “Master”. I don’t think so..



Approach 3 - OneHotEncoder with Pandas

The basic strategy is to convert each category value into a new column and assigns a 1 or 0 (True/False) value to the column.

This has the benefit of not weighing a value improperly but does have the downside of adding more columns to the data set

This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs with the standard kernels.

```
pd.get_dummies(titanic_train, columns=[ "Embarked" ])
```

Embarked	Embarked_0	Embarked_1	Embarked_2
2	0	0	1
0	1	0	0
2	0	0	1
2	0	0	1
2	0	0	1

Approach 3 - OneHotEncoder with Sklearn

```
titanic_train['Embarked'].value_counts()
```

```
2    646  
0    168  
1     77  
Name: Embarked, dtype: int64
```

```
missing_cols = set( titanic_train.columns ) - set( titanic_test.columns )  
for c in missing_cols:  
    titanic_test[c] = 0  
titanic_test = titanic_test[titanic_test.columns]
```

```
from sklearn.preprocessing import OneHotEncoder  
embarked_enc = OneHotEncoder()  
labels = embarked_enc.fit_transform(titanic_train['Embarked'].reshape(-1,1))  
embarked_enc.transform(titanic_test['Embarked'].reshape(-1,1))
```

```
/Users/snooravi/anaconda/lib/python3.6/site-packages/ipykernel/_main_.py:3: FutureWarning: reshape is deprecated and will raise in a subsequent release. Please use .values.reshape(...) instead  
    app.launch_new_instance()  
/Users/snooravi/anaconda/lib/python3.6/site-packages/ipykernel/_main_.py:4: FutureWarning: reshape is deprecated and will raise in a subsequent release. Please use .values.reshape(...) instead
```

```
<418x3 sparse matrix of type '<class 'numpy.float64'>'  
with 418 stored elements in Compressed Sparse Row format>
```

```
titanic_train.Embarked.head(3)
```

```
0    2  
1    0  
2    2  
Name: Embarked, dtype: int8
```

```
labels
```

```
array([[ 0.,  0.,  1.],  
       [ 1.,  0.,  0.],  
       [ 0.,  0.,  1.],  
       ...,  
       [ 0.,  0.,  1.],  
       [ 1.,  0.,  0.],  
       [ 0.,  1.,  0.]])
```

Data Transformation

Why Feature Scaling?

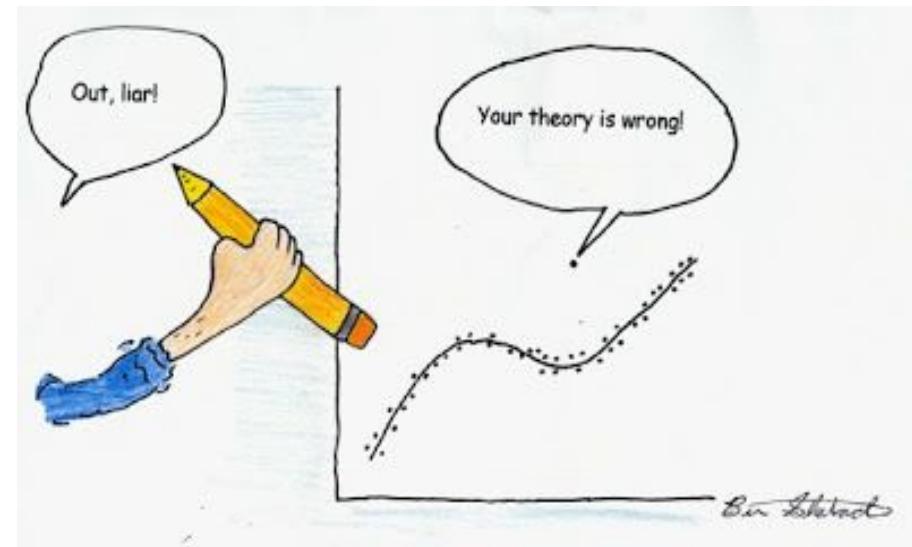
Models don't perform well when the numerical features have different scales.

Two common approaches:

1. Min-Max Scaling (Normalization)
 - a. Features are scaled from 0-1
2. StandardScaler (Standardization)
 - a. Features are scaled with a mean of 0 and has unit variance

Tips:

- When faced with OUTLIERS, it's better to use StandardScaler



Normalization

Sklearn Definition: Normalization is the process of scaling individual samples to have unit norm.

In other words: Normalization refers to rescaling real valued numeric attributes into the range 0 and 1.

Methods:

- L1 Normalization

$$L_1 \text{ norm: } \|\mathbf{e}\|_1 = \left[\sum_i |e_i|^1 \right]$$

- L2 Normalization

$$\|\mathbf{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$$

- MinMaxScaling

$$v' = \frac{v - \min_v}{\max_v - \min_v} (new_max - new_min) + new_min$$

```
>>> X = [[ 1., -1.,  2.],
...        [ 2.,  0.,  0.],
...        [ 0.,  1., -1.]]
>>> X_normalized = preprocessing.normalize(X, norm='l2')

>>> X_normalized
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])
```

Create scaler

```
minmax_scale = preprocessing.MinMaxScaler(feature_range=(0, 1))
```

Scale feature

```
x_scale = minmax_scale.fit_transform(x)
```

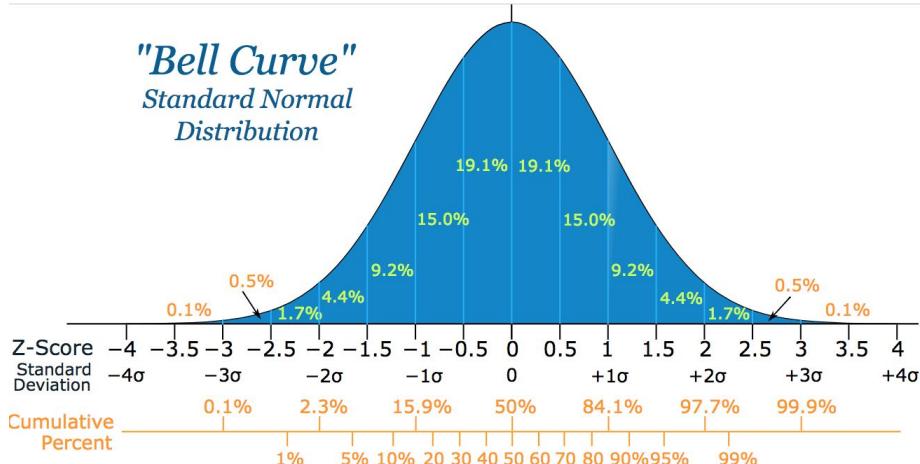
Show feature

```
x_scale
```

Standardization

Sklearn Definition: Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance

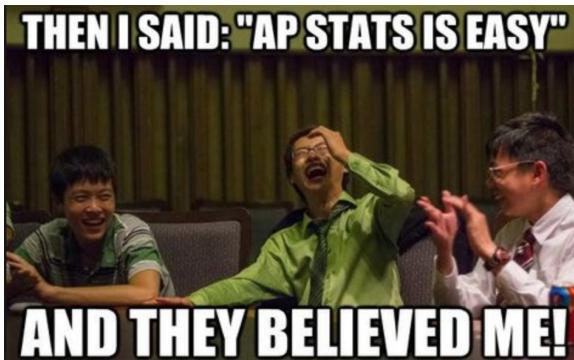
```
>>> from sklearn import preprocessing  
>>> import numpy as np  
>>> X_train = np.array([[ 1., -1.,  2.],  
...                      [ 2.,  0.,  0.],  
...                      [ 0.,  1., -1.]])  
>>> X_scaled = preprocessing.scale(X_train)  
  
>>> X_scaled  
array([[ 0. ..., -1.22...,  1.33...],  
       [ 1.22...,  0. ..., -0.26...],  
       [-1.22...,  1.22..., -1.06...]])
```



Other Transformations

Depending on the data you have, you might want to perform one of the following common transformations.

Remember to back-transform to the original units when reporting final results! To perform a back-transform you apply the *inverse* mathematical function.



Common Transformations

Method	Math Operation	Good for:	Bad for:
Log	$\ln(x)$ $\log_{10}(x)$	Right skewed data $\log_{10}(x)$ is especially good at handling higher order powers of 10 (e.g. 1000, 100000)	Zero values Negative values
Square root	\sqrt{x}	Right skewed data	Negative values
Square	x^2	Left skewed data	Negative values
Cube root	$x^{1/3}$	Right skewed data Negative values	Not as effective at normalizing as log transform
Reciprocal	$1/x$	Making small values bigger and big values smaller	Zero values Negative values

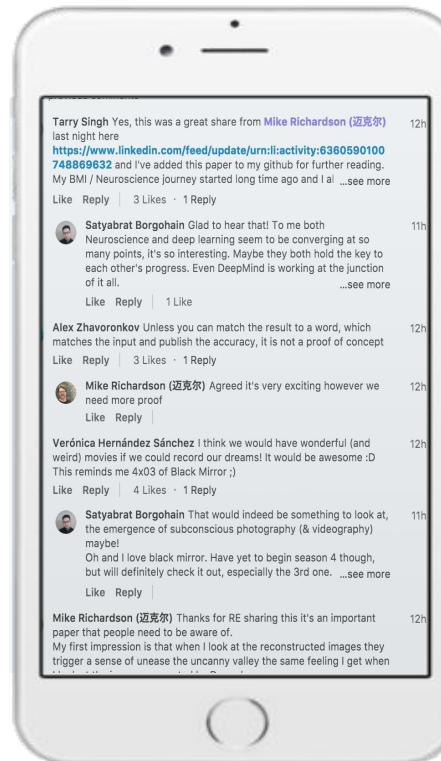


Why Text Data is so Important?

- With the advent of social media, text is **EVERWHERE**.
- We see it in Reddit comments, web pages, Twitter feeds, status updates, etc...
- Many companies hold text data in the forms of medical records, complaint logs, and repair records...

Use Case of Text Data:

- In business, understanding customer feedback requires the understanding of text.
- Which can be seen through product reviews, email messages, and comments.



Handling Text Data

Text data is notorious for being called “dirty” or “unstructured” data

It must undergo some steps of pre-processing in order to derive meaning from it

GOAL:

Want to convert our set of documents into a feature matrix our model can understand

Terminologies

- Text-Preprocessing: transforming text into readable and meaningful data
- Document: one piece of text (could be a single sentence, a paragraph, a blog post, or youtube comment..)
- Token: a word
- Corpus: a collection of documents

Example Dataset - Mercari's challenge

Mercari's challenging you to build an algorithm that automatically suggests the right product prices.

You'll be provided user-inputted text descriptions of their products, including details like product category name, brand name, and item condition.

<https://www.kaggle.com/c/mercari-price-suggestion-challenge>



First Steps..Load and look at your data

Dataset Features

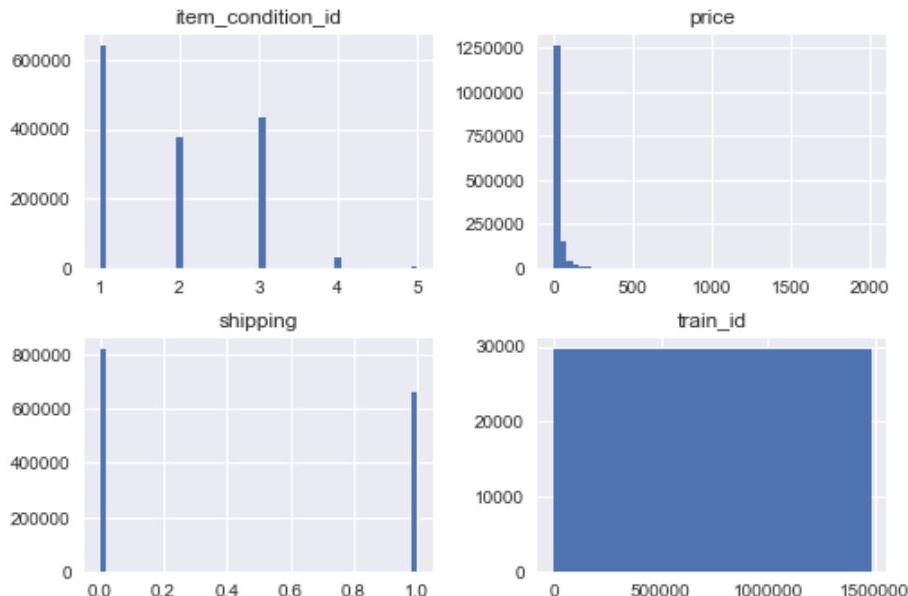
- **ID:** the id of the listing
- **Name:** the title of the listing
- **Item Condition:** the condition of the items provided by the seller
- **Category Name:** category of the listing
- **Brand Name:** brand of the listing
- **Shipping:** whether or not shipping cost was provided
- **Item Description:** the full description of the item
- **Price:** the price that the item was sold for. This is the target variable that you will predict. The unit is USD.

```
import pandas as pd
```

```
training = pd.read_csv('train.tsv', delimiter='\t')  
training.head()
```

train_id	name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet
1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...
2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...
3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete with certificate of authenticity

First Steps..Explore and Visualize your data



```
training.describe()
```

	train_id	item_condition_id	price	shipping
count	1.482535e+06	1.482535e+06	1.482535e+06	1.482535e+06
mean	7.412670e+05	1.907380e+00	2.673752e+01	4.472744e-01
std	4.279711e+05	9.031586e-01	3.858607e+01	4.972124e-01
min	0.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00
25%	3.706335e+05	1.000000e+00	1.000000e+01	0.000000e+00
50%	7.412670e+05	2.000000e+00	1.700000e+01	0.000000e+00
75%	1.111900e+06	3.000000e+00	2.900000e+01	1.000000e+00
max	1.482534e+06	5.000000e+00	2.009000e+03	1.000000e+00

First Steps..Get a sense for Data Types

train_id		name	item_condition_id	category_name	brand_name	price	shipping	item_description
0	0	MLB Cincinnati Reds T Shirt Size XL	3	Men/Tops/T-shirts	NaN	10.0	1	No description yet
1	1	Razer BlackWidow Chroma Keyboard	3	Electronics/Computers & Tablets/Components & P...	Razer	52.0	0	This keyboard is in great condition and works ...
2	2	AVA-VIV Blouse	1	Women/Tops & Blouses/Blouse	Target	10.0	1	Adorable top with a hint of lace and a key hol...
3	3	Leather Horse Statues	1	Home/Home Décor/Home Décor Accents	NaN	35.0	1	New with tags. Leather horses. Retail for [rm]...
4	4	24K GOLD plated rose	1	Women/Jewelry/Necklaces	NaN	44.0	0	Complete with certificate of authenticity

```
training.dtypes
```

```
train_id          int64
name            object
item_condition_id  int64
category_name    object
brand_name      object
price           float64
shipping         int64
item_description object
dtype: object
```

- Numerical discrete: train_id, item_condition_id, shipping
- Numerical continuous: price
- Categorical nominal: category_name, brand_name
- Categorical ordinal: None
- Text: item_description

Handling Missing Data

```
training.count()
```

```
train_id          1482535
name             1482535
item_condition_id 1482535
category_name    1476208
brand_name       849853
price            1482535
shipping         1482535
item_description 1482531
dtype: int64
```

```
mercari['item_description'] = mercari.item_description.fillna('')
```

Missing data:

- Category name
- Brand name
- Item descriptions

Options:

1. Drop observations with missing values
2. Fill in with mean, median, or mode
3. **Fill in with empty string**

Let's look at a few samples of item description

```
mercari.item_description[0]
```

```
'No description yet'
```

```
mercari.item_description[1]
```

```
'This keyboard is in great condition and works like it came out of the box. All of the ports are tested and work perfectly. The lights are customizable via the Razer Synapse app on your PC.'
```

```
mercari.item_description[0]
```

```
"KEEP YOUR DRINKS AS COLD AS SCIENCE ALLOWS WITH THE YETI RAMBLER 30 OZ TUMBLER If you want to have cold things stay cold for hours throughout the day, and HOT things HOT for hours throughout the day...THEN THIS YETI IS WHAT YOU NEED! !!DON'T MISS OUR BLOWOUT SALE, while supplies last!! Good for Sweet Tea, Bloody Marys, Root Beer Floats, Large Coffee, Cold Water, Wine COLORS: •Hot Pink •Black •Matte Black •Tiffany Blue •Mint Green •Carolina Blue This Yeti is made with: ✓Strong durable 18/8 Stainless Steel ✓Double-Wall Vacuum Insulation ✓Sweat Free Design ✓BPA FREE STORE'S GUARANTEES: ✓BEST CUSTOMER SERVICE ✓LESS THAN 24 HOUR SHIPMENT TO US PS ✓LOW PRICES ✓HIGH QUALITY PRODUCTS ✓EMPATHETIC COMMUNICATION & ✓FAST COMMUNICATION TO CUSTOMERS If you would like more special presents then look at my store for Authentic Apple 1 Meter 3 Feet Chargers, Authentic Apple 2 Meter 6 Feet Chargers, Authentic Apple Headphones/Earpods/Earpods, Lumee, iPhone 6/6s Cases. FriendlyTronics Coupons"
```

Especially when working with text data, you want to print out a few examples of what you will be working with as the preprocessing steps will fully depend on the data.

Consider: how you handle tweets that are short with hashtags and slang will be very different than the way we would handle text from books.

Brainstorm on Pre-processing Steps

Things to look for:

- Removing punctuation
- Removing numbers
- Removing stop words
- Part of speech tagging
- Lemmatization/Stemming
- Handling short text
- N-grams (red, wine, or red wine)

Especially crucial for social media and emails:

- Chat word removal
- Word standardization (happpy → happy)
- Website urls removal

Libraries to help with this:

- from string import punctuation
- import sklearn.preprocessing
- from nltk.corpus import stopwords
- from nltk.stem import WordNetLemmatizer
- from nltk.stem.porter import PorterStemmer

Apply Pre-processing Steps

after looking at a few descriptions maybe the following will suffice:

1. punctuation removal
2. lowercase
3. stop word removal
4. remove digits

Examples:

```
def remove_punctuation(x):
    for each in punctuation_symbols:
        x = x.replace(each)
    return x
```

```
def remove_digits(x):
    x = ''.join([i for i in x if not i.isdigit()])
    return x
```

```
mercari.item_description[1482524]
```

```
'■nwot ■perfect condition ■barely worn ■no flaws ■not nike stadium athletics ■no pockets great leisure wear'
```

still not perfect...

CountVectorizer - Definition

Sklearn: Converts a collection of text documents into a matrix of token counts.

In other words:

CountVectorizer will count the word frequencies of words in a corpus. The importance of a word should increase with the number of times it occurs

```
In [7]: test_record=['This is amazing.Check it out.']}
```

```
In [8]: test_dtm=vectorizer.transform(test_record)
```

```
In [9]: pd.DataFrame(test_dtm.toarray(),columns=vectorizer.get_feature_names())
```

```
Out[9]:
```

	amazing	and	delivered	ever	fast	higher	is	much	price	product	safe	than
0	0	1	0	0	0	0	0	1	0	0	0	0

CountVectorizer - Parameters

Parameters to consider:

ngram_range : tuple (min_n, max_n)

The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of n such that $\text{min_n} \leq n \leq \text{max_n}$ will be used.

max_df : float in range [0.0, 1.0] or int, default=1.0

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

min_df : float in range [0.0, 1.0] or int, default=1

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

The most frequent words are not the most descriptive.

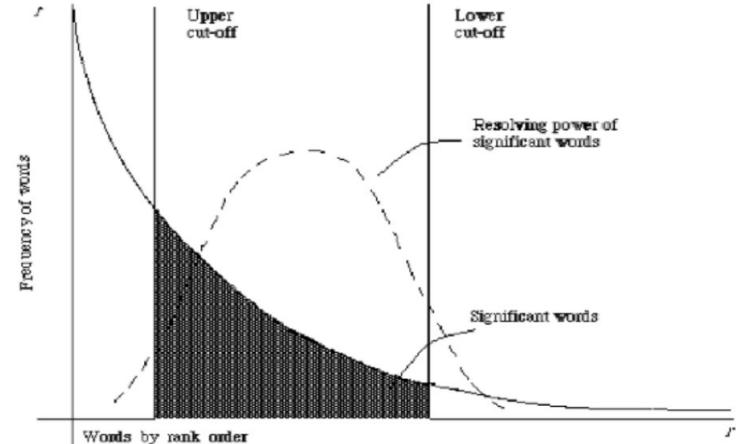


Figure 2.1. A plot of the hyperbolic curve relating f , the frequency of occurrence and r , the rank order (Adapted from Schatz¹⁴, page 120)

(from van Rijsbergen 79)

Apply CountVectorizer

This is great!

```
from sklearn.feature_extraction.text import CountVectorizer

min_df = 0.01
max_df = 0.90
ngram_range = 2 # accounting for bi-grams

print('feature space will include words if they appear in a minimum of ', int(min_df*len(mercari)))
print('feature space will include words if they appear in a maximum of ', int(max_df*len(mercari)))

feature space will include words if they appear in a minimum of 1482
feature space will include words if they appear in a maximum of 133428

# instantiate
countvec = CountVectorizer(stop_words='english', min_df=min_df, max_df=max_df, ngram_range=(0,2))

# fit and transform the item description
countvec_matrix = countvec.fit_transform(mercari.item_description)
```

```
'excellent condition',
'extra',
'fast',
'feel',
'feel free',
'firm',
'fit',
'fits',
'flaws',
'free',
'free home',
'free shipping',
'gently',
'gift',
'gold',
'good',
'good condition',
'gray',
'great',
'great condition',
```

Feature Space

Your new feature space resembles a “Bag of Words” model

	adjustable	ask	authentic	available	baby	bag	beautiful	best	big	black	...	white	womens	wore	work	works	worn	worn times	xl	xs	zip
0	0	0	3	0	0	0	0	1	0	2	...	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	

5 rows × 231 columns

Dimensionality Reduction

Why Dimensionality Reduction?

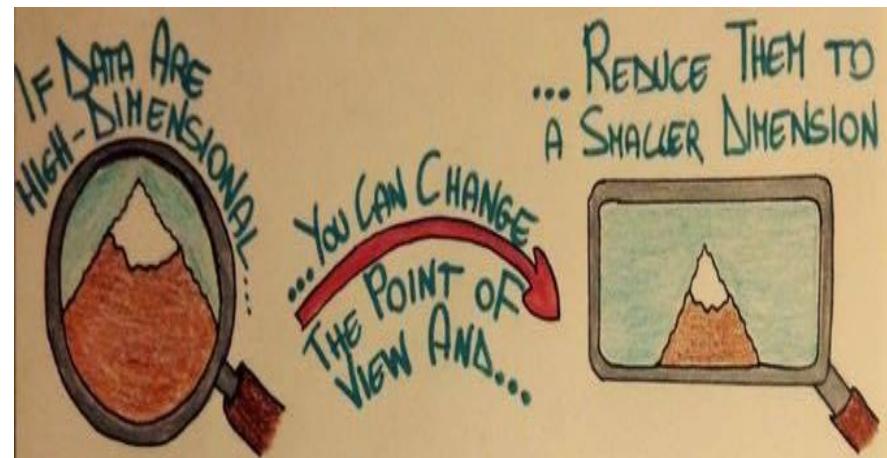
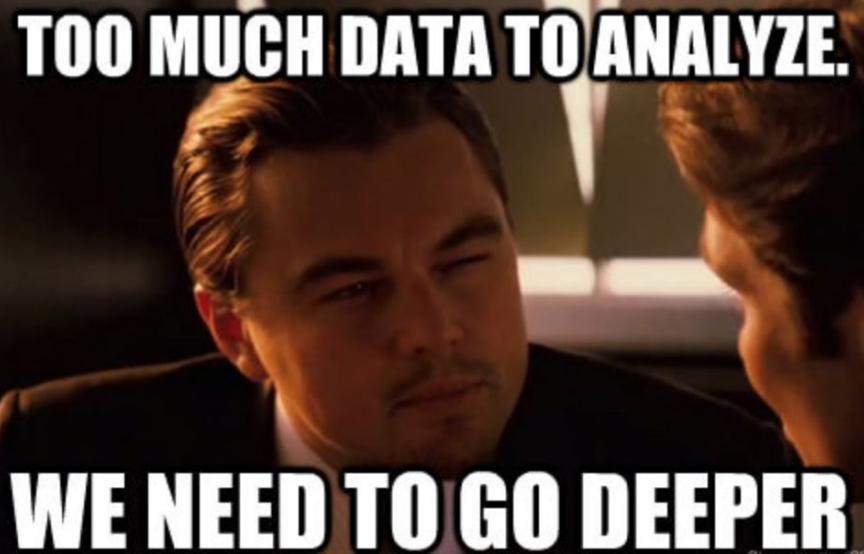
Want to reduce complexity, while not losing too much information

How does Dimensionality Reduction help us?

- Speeds up model training
- Good for data visualization
- Sometimes increases model performance

Disadvantages:

- Lose some data in the process



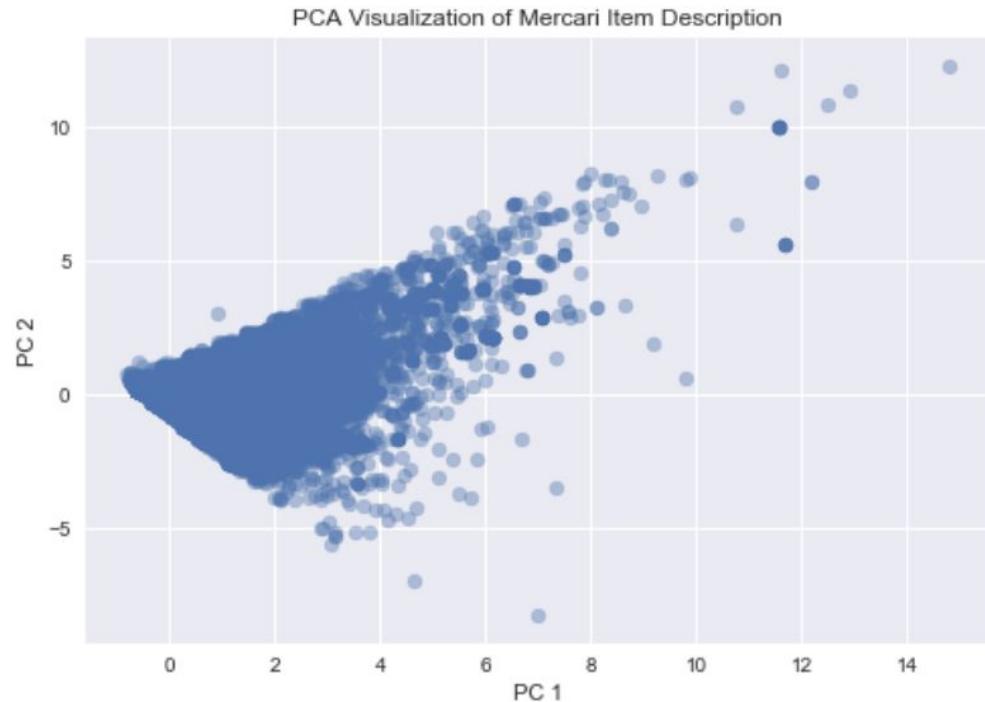
Dimensionality Reduction

Direct Method:

- Feature Selection: select a minimum set of attributes that is sufficient for the task

Indirect Methods:

- Principal Component Analysis:
PCA
- Singular Value Decomposition:
SVD



Summary



Working with Numerical Data:

- Data Cleaning
 - Handling Missing Data
 - Imputation
- Data Transformation
 - Normalization, Standardization and other common Transformation
- Data Discretization (binning)
- Feature Engineering

Working with Categorical Data:

- Encoding using LabelEncoder and OneHotEncoding (Pandas and Sklearn)

Working with Text Data:

- Stopword removal, lemmatization, stemming, n-grams, min_df, max_df
- CountVectorizer
- Dimensionality Reduction

Additional Resources

- Advantages and Disadvantages for Different Dimensionality Reduction Methods:
[here](#)
- Principal Component Analysis: [here](#)
- MinMaxScaling: [here](#)
- Categorical Encoding: [here](#)
- Data Pre-Processing: [here](#)
- Quick and Dirty Analysis with Pandas: [here](#)
- Data Transformation: [here](#)
- Data Transformations: [here](#)

**MIN MAX
SCALING**

Rescales feature values to between 0 and 1

Original value x_i Minimum value in feature

Rescaled value $x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$ Maximum value in feature

Chris Albon

Code and Slides

For access to all the code used to produce this presentation and the slides, see: [here](#)

Thank You!

