Preprocessing for Machine Learning in Python

Sunday, 30 August 2020 12:49 PM

Introduction to Data Preprocessing

What is data preprocessing?

- Beyond cleaning and exploratory data analysis
- Prepping data for modeling
- Modeling in Python requires numerical input

Refresher on Pandas basics

```
import pandas as pd
hiking = pd.read_json("datasets/hiking.json")
print(hiking.head())
```

```
  Accessible Difficulty      Length Limited_Access
0          Y       None   0.8 miles              N
1          N       Easy    1.0 mile              N
2          N       Easy  0.75 miles              N
3          N       Easy   0.5 miles              N
4          N       Easy   0.5 miles              N
```

your dataset has categorical variables, you'll need to transform them

Refresher on Pandas basics

```
print(hiking.columns)
```

```
Index(['Accessible','Difficulty',
     'Length','Limited_Access',
     'Location','Name',
     'Other_Details','Park_Name',
     'Prop_ID','lat','lon'],
dtype='object')
```

```
print(hiking.dtypes)
```

```
Accessible         object
Difficulty         object
Length             object
Limited_Access     object
Location           object
Name               object
Other_Details      object
Park_Name          object
Prop_ID            object
lat               float64
lon               float64
dtype: object
```

Refresher on Pandas basics

```
print(wine.describe())
```

```
              Type     Alcohol    ...  Alcalinity of ash
count  178.000000  178.000000    ...   178.000000
mean     1.938202   13.000618    ...    19.494944
std      0.775035    0.811827    ...     3.339564
min      1.000000   11.030000    ...    10.600000
25%      1.000000   12.362500    ...    17.200000
50%      2.000000   13.050000    ...    19.500000
75%      3.000000   13.677500    ...    21.500000
max      3.000000   14.830000    ...    30.000000
```

Removing missing data

```
print(df)
```

```
     A    B    C
0  1.0  NaN  2.0
1  4.0  7.0  3.0
2  7.0  NaN  NaN
3  NaN  7.0  NaN
4  5.0  9.0  7.0
```

```
print(df.dropna())
```

```
     A    B    C
1  4.0  7.0  3.0
4  5.0  9.0  7.0
```

Removing missing data

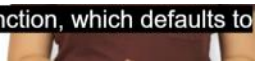
```
print(df)
```

```
     A    B    C
0  1.0  NaN  2.0
1  4.0  7.0  3.0
2  7.0  NaN  NaN
3  NaN  7.0  NaN
4  5.0  9.0  7.0
```

```
print(df.drop([1, 2, 3]))
```

```
     A    B    C
0  1.0  NaN  2.0
4  5.0  9.0  7.0
```

You can drop specific rows by passing index labels to the drop function, which defaults to dropping rows.



Removing missing data

```
print(df)
```

```
     A    B    C
0  1.0  NaN  2.0
1  4.0  7.0  3.0
2  7.0  NaN  NaN
3  NaN  7.0  NaN
4  5.0  9.0  7.0
```

```
print(df.drop("A", axis=1))
```

```
     B    C
0  NaN  2.0
1  7.0  3.0
2  NaN  NaN
3  7.0  NaN
4  9.0  7.0
```

The first parameter is the column name, in this case A.

We have to specify axis=1 in order to designate that we want to drop a column

What if we want to drop rows where data is missing in a particular column?

We can do this with the help of boolean indexing, which is a way to filter a dataframe based on certain values.

Instead of indexing a dataframe using column or row names, you can set

a condition to filter your dataframe by to return a specific set of data.

a condition to filter your dataframe by to return a specific set of data.

For example, if we wanted only rows in this dataframe where column B

is equal to 7, we can filter it by selecting where column B is equal to 7

Removing missing data

```
print(df)
```

```
print(df[df["B"] == 7])
```

```
     A    B    C
0  1.0  NaN  2.0
1  4.0  7.0  3.0
2  7.0  NaN  NaN
3  NaN  7.0  NaN
4  5.0  9.0  7.0
```

```
     A    B    C
1  4.0  7.0  3.0
3  NaN  7.0  NaN
```

Removing missing data

```
print(df)
```

```
     A    B    C
0  1.0  NaN  2.0
1  4.0  7.0  3.0
2  7.0  NaN  NaN
3  NaN  7.0  NaN
4  5.0  9.0  7.0
```

```
print(df["B"].isnull().sum())
```

```
2
```

So we have 2 missing values.

To filter those out, we can simply use the notnull method on column B as a boolean index.

```
print(df[df["B"].notnull()])
```

```
     A    B    C
1  4.0  7.0  3.0
3  NaN  7.0  NaN
4  5.0  9.0  7.0
```

Why are types important?

```
print(volunteer.dtypes)
```

```
opportunity_id      int64
content_id          int64
vol_requests        int64
...                   ...
summary            object
is_priority        object
category_id       float64
```

- object: string/mixed types
- int64: integer
- float64: float
- datetime64 (or timedelta): datetime

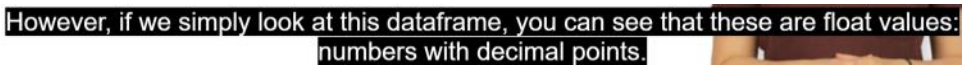
Converting column types

```
print(df)
```

```
   A        B    C
0  1   string  1.0
1  2  string2  2.0
2  3  string3  3.0
```

```
print(df.dtypes)
```

```
A      int64
B     object
C     object
dtype: object
```



Converting column types

```
print(df)
```

```
   A        B    C
0  1   string  1.0
1  2  string2  2.0
2  3  string3  3.0
```

```
df["C"] = df["C"].astype("float")
print(df.dtypes)
```

```
A      int64
B     object
C    float64
dtype: object
```

Splitting up your dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
   X_train y_train
0      1.0       n
1      4.0       n
       ...
5      5.0       n
6      6.0       n

   X_test y_test
0     9.0      y
1     1.0      n
2     4.0      n
```

Stratified sampling

- 100 samples, 80 class 1 and 20 class 2
- Training set: 75 samples, 60 class 1 and 15 class 2
- Test set: 25 samples, 20 class 1 and 5 class 2

Stratified sampling

```
# Total "labels" counts
y["labels"].value_counts()
```

```
class1    80
class2    20
Name: labels, dtype: int64
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y, stratify=y)
```

Stratified sampling

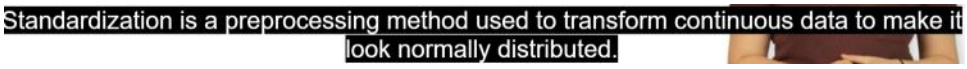
```
y_train["labels"].value_counts()
```

```
class1    60
class2    15
Name: labels, dtype: int64
```

```
y_test["labels"].value_counts()
```

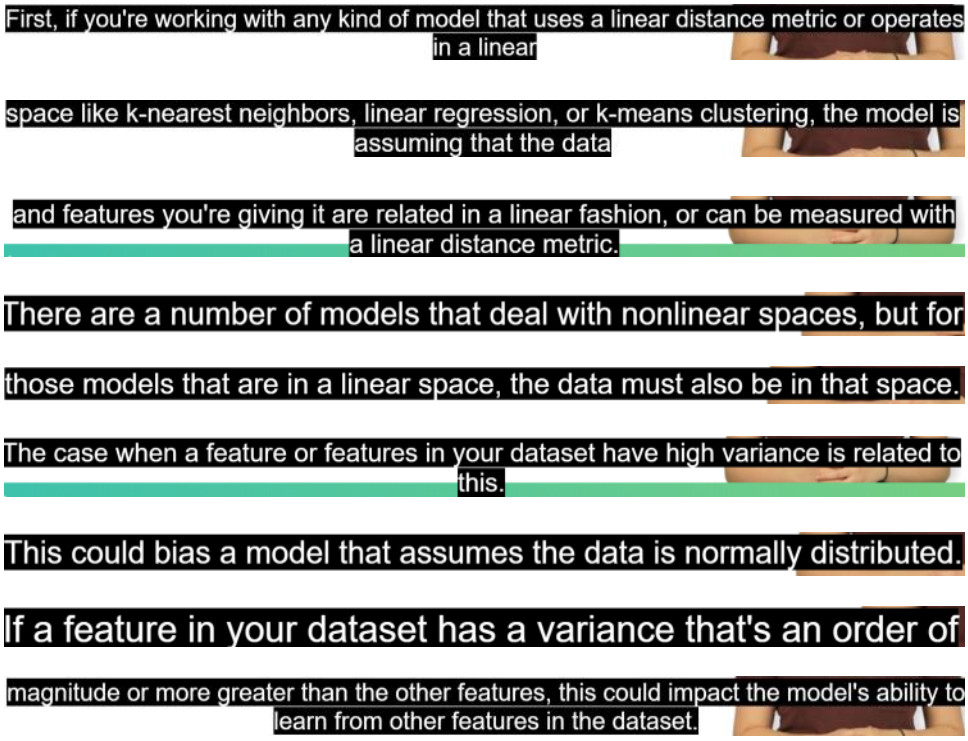
```
class1    20
class2    5
Name: labels, dtype: int64
```

Standardizing Data



What is standardization?

- Scikit-learn models assume normally distributed data
- Log normalization and feature scaling in this course
- Applied to continuous numerical data



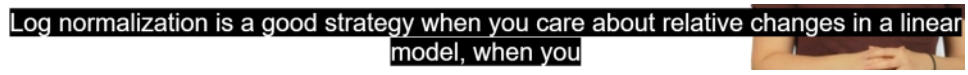
When to standardize: models

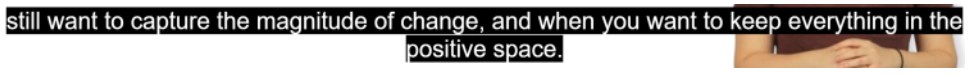
- Model in linear space
- Dataset features have high variance
- Dataset features are continuous and on different scales
- Linearity assumptions

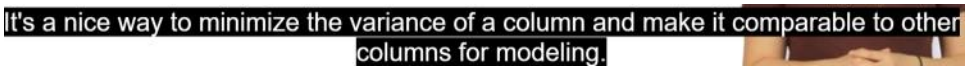
What is log normalization?

- Applies log transformation
- Natural log using the constant _e_ (2.718)
- Captures relative changes, the magnitude of change, and keeps everything in the positive space

Number	Log
30	3.4
300	5.7
3000	8







Log normalization in Python

```
print(df)
```

```
   col1   col2
0  1.00    3.0
1  1.20   45.5
2  0.75   28.0
3  1.60  100.0
```

```
print(df.var())
```

```
col1       0.128958
col2    1691.729167
dtype: float64
```

If we check the variance of the columns, you can see that column 2 has a significantly higher variance than column 1.

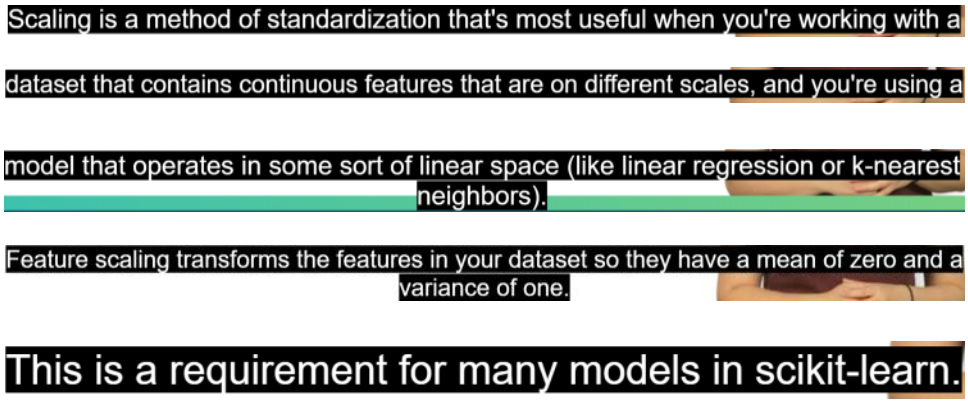
To apply log normalization to column 2, we need the log function from numpy.

```
import numpy as np
df["log_2"] = np.log(df["col2"])
print(df)
```

```
   col1   col2     log_2
0  1.00    3.0  1.098612
1  1.20   45.5  3.817712
2  0.75   28.0  3.332205
3  1.60  100.0  4.605170
```

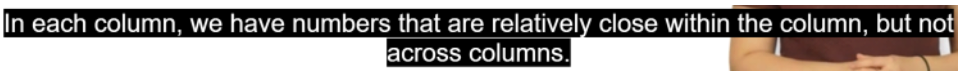
```
print(np.var(df[["col1", "log_2"]]))
```

```
col1      0.096719
log_2     1.697165
dtype: float64
```



What is feature scaling?

- Features on different scales
- Model with linear characteristics
- Center features around 0 and transform to unit variance
- Transforms to approximately normal distribution



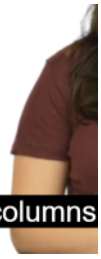
How to scale data

```
print(df)
```

```
   col1  col2   col3
0  1.00  48.0  100.0
1  1.20  45.5  101.3
2  0.75  46.2  103.5
3  1.60  50.0  104.0
```

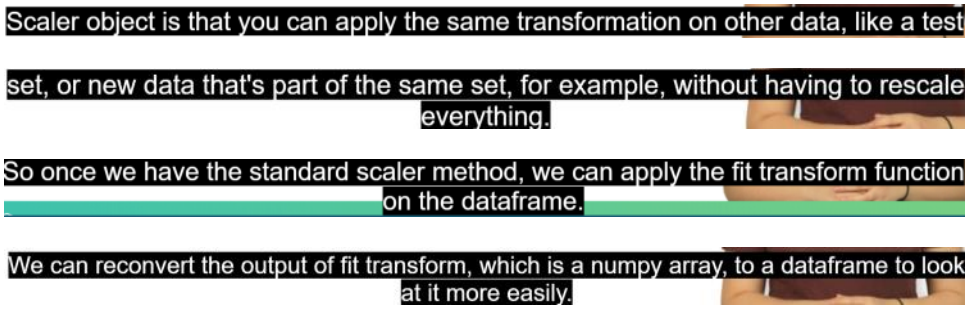
```
print(df.var())
```

```
col1    0.128958
col2    4.055833
col3    3.526667
dtype: float64
```



How to scale data

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df),
                                    columns=df.columns)
```




```
print(df_scaled)
```

```
       col1      col2      col3
0 -0.442127  0.329683 -1.352726
1  0.200967 -1.103723 -0.553388
2 -1.245995 -0.702369  0.799338
3  1.487156  1.476409  1.106776
```

```
print(df.var())
```

```
col1    1.333333
col2    1.333333
col3    1.333333
dtype: float64
```

K-nearest neighbors

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```
# Preprocessing first
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

```
knn.score(X_test, y_test)
```

Feature Engineering

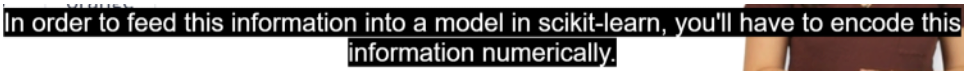
What is feature engineering?

- Creation of new features based on existing features
- Insight into relationships between features
- Extract and expand data
- Dataset-dependent

Feature engineering scenarios

Id	Text
1	"Feature engineering is fun!"
2	"Feature engineering is a lot of work."
3	"I don't mind feature engineering."

user	fav_color
1	blue
2	green
3	orange



Another common example is with timestamps.

Feature engineering scenarios

Id	Date
4	July 30 2011
5	January 29 2011
6	February 05 2011

user	test1	test2	test3
1	90.5	89.6	91.4
2	65.5	70.6	67.3
3	78.1	80.7	81.8

Categorical variables

```
   user subscribed fav_color
0     1          y      blue
1     2          n     green
2     3          n    orange
3     4          y     green
```

Encoding binary variables - Pandas

```
print(users["subscribed"])
```

```
0    y
1    n
2    n
3    y
Name: subscribed, dtype: object
```

```
print(users[["subscribed", "sub_enc"]])
```

```
  subscribed  sub_enc
0          y        1
1          n        0
2          n        0
3          y        1
```

```
users["sub_enc"] = users["subscribed"].apply(lambda val:
                                             1 if val == "y" else 0)
```

Encoding binary variables - scikit-learn

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
users["sub_enc_le"] = le.fit_transform(users["subscribed"])
```

```
print(users[["subscribed", "sub_enc_le"]])
```

```
  subscribed  sub_enc_le
0          y           1
1          n           0
2          n           0
```

One-hot encoding

fav_color
blue
green
orange
green

fav_color_enc
[1, 0, 0]
[0, 1, 0]
[0, 0, 1]
[0, 1, 0]

Values: [blue, green, orange]

- blue: [1, 0, 0]
- green: [0, 1, 0]
- orange: [0, 0, 1]

```
print(users["fav_color"])
```

```
0      blue
1     green
2    orange
3     green
Name: fav_color, dtype: object
```

```
print(pd.get_dummies(users["fav_color"]))
```

```
   blue  green  orange
0     1      0       0
1     0      1       0
2     0      0       1
3     0      1       0
```

```
print(df)
```

```
     city  day1  day2  day3
0     NYC  68.3  67.9  67.8
1      SF  75.1  75.5  74.9
2      LA  80.3  84.0  81.3
3  Boston  63.0  61.0  61.2
```

```
columns = ["day1", "day2", "day3"]
df["mean"] = df.apply(lambda row: row[columns].mean(), axis=1)
print(df)
```

```
     city  day1  day2  day3   mean
0     NYC  68.3  67.9  67.8  68.00
1      SF  75.1  75.5  74.9  75.17
2      LA  80.3  84.0  81.3  81.87
3  Boston  63.0  61.0  61.2  61.73
```

Dates

```
print(df)
```

```
                date purchase
0       July 30 2011   $45.08
1   February 01 2011   $19.48
2    January 29 2011   $76.09
3      March 31 2012   $32.61
4   February 05 2011   $75.98
```

Dates

```
df["date_converted"] = pd.to_datetime(df["date"])
```

```
df["month"] = df["date_converted"].apply(lambda row: row.month)
```

```
print(df)
```

```
              date purchase date_converted  month
0      July 30 2011   $45.08     2011-07-30      7
1  February 01 2011   $19.48     2011-02-01      2
2   January 29 2011   $76.09     2011-01-29      1
3     March 31 2012   $32.61     2012-03-31      3
4  February 05 2011   $75.98     2011-02-05      2
```

Extraction

```
import re
```

```
my_string = "temperature:75.6 F"
```

Here we have a string, and we want to extract the temperature digit from it.

```
pattern = re.compile("\d+\.\d+")
```

"backslash d" means that we want to grab digits, and the "plus" means we want to grab as many as possible.

"backslash period" means we want to grab the decimal point, and then there's another "backslash d plus" at the end to grab the digits on the right-hand side of the decimal.

```
temp = re.match(pattern,
                my_string)
```

```
print(float(temp.group(0))
```

```
75.6
```

Vectorizing text

- tf = term frequency
- idf = inverse document frequency

Vectorizing text

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
print(documents.head())
```

```
0    Building on successful events last summer and ...
1               Build a website for an Afghan business
2    Please join us and the students from Mott Hall...
3    The Oxfam Action Corps is a group of dedicated...
4    Stop 'N' Swap reduces NYC's waste by finding n...
```

```
tfidf_vec = TfidfVectorizer()
text_tfidf = tfidf_vec.fit_transform(documents)
```

Text classification

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Selecting features for modeling

What is feature selection?

- Selecting features to be used for modeling
- Doesn't create new features
- Improve model's performance

Scikit-learn has several methods for automated feature selection, such as choosing a variance threshold and using univariate statistical tests, but we won't cover those here.

Most of the methods we'll cover in this chapter are more on the manual side,

When to select features

city	state	lat	long
hico	tx	31.982778	-98.033333
mackinaw city	mi	45.783889	-84.727778
winchester	ky	37.990000	-84.179722

Redundant features

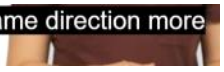
- Remove noisy features
- Remove correlated features
- Remove duplicated features

Correlated features

- Statistically correlated: features move together directionally
- Linear models assume feature independence
- Pearson correlation coefficient

Linear models in particular assume that features are independent of each other and if features are strongly correlated, that could introduce bias into your model

The Pearson correlation coefficient is a measure of this directionality: a score closer to 1 between pairs of features means that they move together in the same direction more strongly, a



score closer to 0 means features are not correlated, and a score close to -1 means they are

We can easily check this in Pandas.

Correlated features

```
print(df)
```

```
        A     B     C
0    3.06  3.92  1.04
1    2.76  3.40  1.05
2    3.24  3.17  1.03
...
```

```
print(df.corr())
```

```
          A         B         C
A  1.000000  0.787194  0.543479
B  0.787194  1.000000  0.565468
C  0.543479  0.565468  1.000000
```

To check the correlations within a dataset, we can use the "corr" method on this dataframe.

You can see that features A and B score close to 1 (0.78), so we should likely drop one of those features.

After you've vectorized your text, the vocabulary and weights will be stored in the vectorizer.

To pull out the vocabulary list, which you'll need in order to look at word weights, you can use the vocabulary attribute.

Looking at word weights

```
print(tfidf_vec.vocabulary_)
```

```
{'200': 0,
 '204th': 1,
 '33rd': 2,
 'ahead': 3,
 'alley': 4,
 ...
```

```
print(text_tfidf[3].data)
```

```
[0.19392702 0.20261085 0.2491
 0.31957651 0.18599931 ...]
```

```
print(text_tfidf[3].indices)
```

```
[ 31 102  20  70   5 ...]
```

To reverse the vocabulary dictionary, you can swap the key-value pairs by grabbing the items from the vocabulary dictionary and reversing the order.

Looking at word weights

```
vocab = {v:k for k,v in
   tfidf_vec.vocabulary_.items()}
```

```
print(vocab)
```

```
{0: '200',
 1: '204th',
 2: '33rd',
 3: 'ahead',
 4: 'alley',
 ...
```

Finally, we can also zip together the row indices and weights, pass it into the dict function, and turn that into a dictionary.

```
zipped_row =
dict(zip(text_tfidf[3].indices,
text_tfidf[3].data))
```

```
print(zipped_row)
```

```
{5: 0.1597882543332701,
 7: 0.2657643209876317 5,
 8: 0.1859993133192567 6,
 9: 0.2657643209876317 5,
 10: 0.1307735525845036 6,
 ...
```

Looking at word weights

```
def return_weights(vocab, vector, vector_index):
```

```
zipped = dict(zip(vector[vector_index].indices,
                  vector[vector_index].data))
```

```
return {vocab[i]:zipped[i] for i in vector[vector_index].indices}
```

```
print(return_weights(vocab, text_tfidf, 3))
```

```
{'and': 0.1597882543332701,
 'are': 0.26576432098763175,
 'at': 0.18599931331925676,
 ...
```

We'll pass in the reversed vocab list, the vector, and the row we want to retrieve data for.

We'll do row zipping to a dictionary in the function.

And finally, we'll return a dictionary mapping the word to its score.

So if we pass in the reversed vocabulary list (vocab), the text_tfidf vector, and the index for the 4th row (3), we now have a mapping of scores to words.

Dimensionality reduction and PCA

- Unsupervised learning method
- Combines/decomposes a feature space
- Feature extraction - here we'll use to reduce our feature space
- Principal component analysis
- Linear transformation to uncorrelated space
- Captures as much variance as possible in each component

PCA in scikit-learn

```
from sklearn.decomposition import PCA
pca = PCA()
df_pca = pca.fit_transform(df)
```

```
print(df_pca)
```

```
[88.4583, 18.7764, -2.2379, ..., 0.0954, 0.0361, -0.0034],
[93.4564, 18.6709, -1.7887, ..., -0.0509, 0.1331, 0.0119],
[-186.9433, -0.2133, -5.6307, ..., 0.0332, 0.0271, 0.0055]
```

```
print(pca.explained_variance_ratio_)
```

```
[0.9981, 0.0017, 0.0001, 0.0001, ...]
```

PCA caveats

- Difficult to interpret components
- End of preprocessing journey

Putting it all together

Important concepts to remember

- Missing data: `dropna()` and `notnull()`
- Types: `astype()`
- Stratified sampling: `train_test_split(X, y, stratify=y)`

Categorical variables

```
    state country       type
295    az      us      light
296    tx      us  formation
297    nv      us   fireball
```

- One-hot encoding: `pd.get_dummies()`

Standardization

- `var()`
- `np.log()`

UFO feature engineering

date	length_of_time	desc
6/16/2013 21:00	5 minutes	Sabino Canyon Tucson Arizona night UFO sighting.
9/12/2005 22:35	5 minutes	Star like objects hovering in sky&#44 slowly m...
12/31/2013 22:25	3 minutes	Three orange fireballs spotted by witness in E...

- Dates: `.month` or `.hour` attributes
- Regex: `\d` and `.group()`
- Text: tf-idf and `TfidfVectorizer`

Feature selection and modeling

- Redundant features
- Text vector

Final thoughts

- Iterative processes
- Know your dataset
- Understand your modeling task