Introduction to preprocessing

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

What is data preprocessing?

- After exploratory data analysis and data cleaning
- Preparing data for modeling

- **Example:** transforming categorical features into numerical features (dummy variables)

Why preprocess?

- Transform dataset so it's suitable for modeling
- Improve model performance
- Generate more reliable results

Recap: exploring data with pandas

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

```
  Prop_ID                    Name  ... lat lon
0    B057  Salt Marsh Nature Trail  ... NaN NaN
1    B073                Lullwater  ... NaN NaN
2    B073                  Midwood  ... NaN NaN
3    B073                Peninsula  ... NaN NaN
4    B073                Waterfall  ... NaN NaN
```

Recap: exploring data with pandas

```python
print(hiking.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33 entries, 0 to 32
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
 --  ------          --------------  -----
 0   Prop_ID         33 non-null     object
 1   Name            33 non-null     object
 2   Location        33 non-null     object
 3   Park_Name       33 non-null     object
 4   Length          29 non-null     object
 5   Difficulty      27 non-null     object
 6   Other_Details   31 non-null     object
 7   Accessible      33 non-null     object
 8   Limited_Access  33 non-null     object
 9   lat             0 non-null      float64
 10  lon             0 non-null      float64
dtypes: float64(2), object(9)
memory usage: 3.0+ KB
```

Recap: exploring data with pandas

```
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
```

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
```

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.isna().sum())
```

```
A    1
B    2
C    2
dtype: int64
```

```
print(df.dropna(subset=["B"]))
```

```
     A    B    C
1  4.0  7.0  3.0
3  NaN  7.0  NaN
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.dropna(thresh=2))
```

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

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Working With Data Types

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Why are types important?

```
print(volunteer.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 665 entries, 0 to 664
Data columns (total 35 columns):
 #   Column              Non-Null Count  Dtype
 --  ------              --------------  -----
 0   opportunity_id      665 non-null    int64
 1   content_id          665 non-null    int64
 2   vol_requests        665 non-null    int64
 3   event_time          665 non-null    int64
 4   title               665 non-null    object
 ..  ...                 ...             ...
 34  NTA                 0 non-null      float64
dtypes: float64(13), int64(8), object(14)
memory usage: 182.0+ KB
```

- `object` : string/mixed types
- `int64` : integer
- `float64` : float
- `datetime64` : dates and times

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.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
 --  ------  --------------  -----
 0   A       3 non-null      int64
 1   B       3 non-null      object
 2   C       3 non-null      object
dtypes: int64(1), object(2)
memory usage: 200.0+ bytes
```

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
```

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Training and test sets

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Why split?

1. Reduces *overfitting*
2. Evaluate performance on a holdout set

Splitting up your dataset

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
   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

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

Stratified sampling

```
X_train,X_test,y_train,y_test = train_test_split(X, y, stratify=y, random_state=42)
```

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

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

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
```

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Standardization

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

What is standardization?

Standardization: transform *continuous* data to appear *normally distributed*

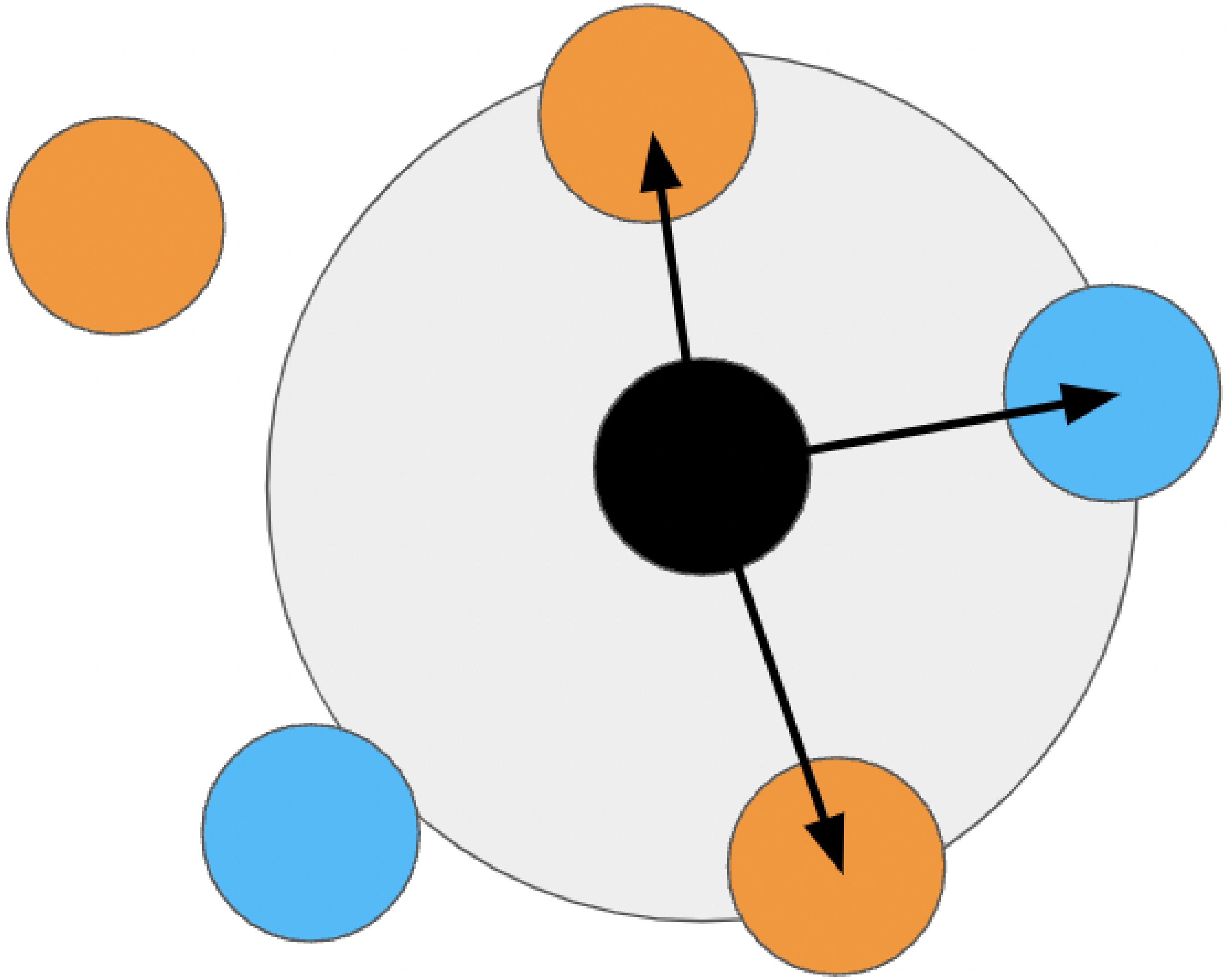
- `scikit-learn` models assume normally distributed data
- Using non-normal training data can introduce *bias*
- **Log normalization** and feature **scaling** in this course
- Applied to continuous numerical data

When to standardize: linear distances

- Model in *linear* space

Examples:

- k-Nearest Neighbors (kNN)
- Linear regression
- K-Means Clustering



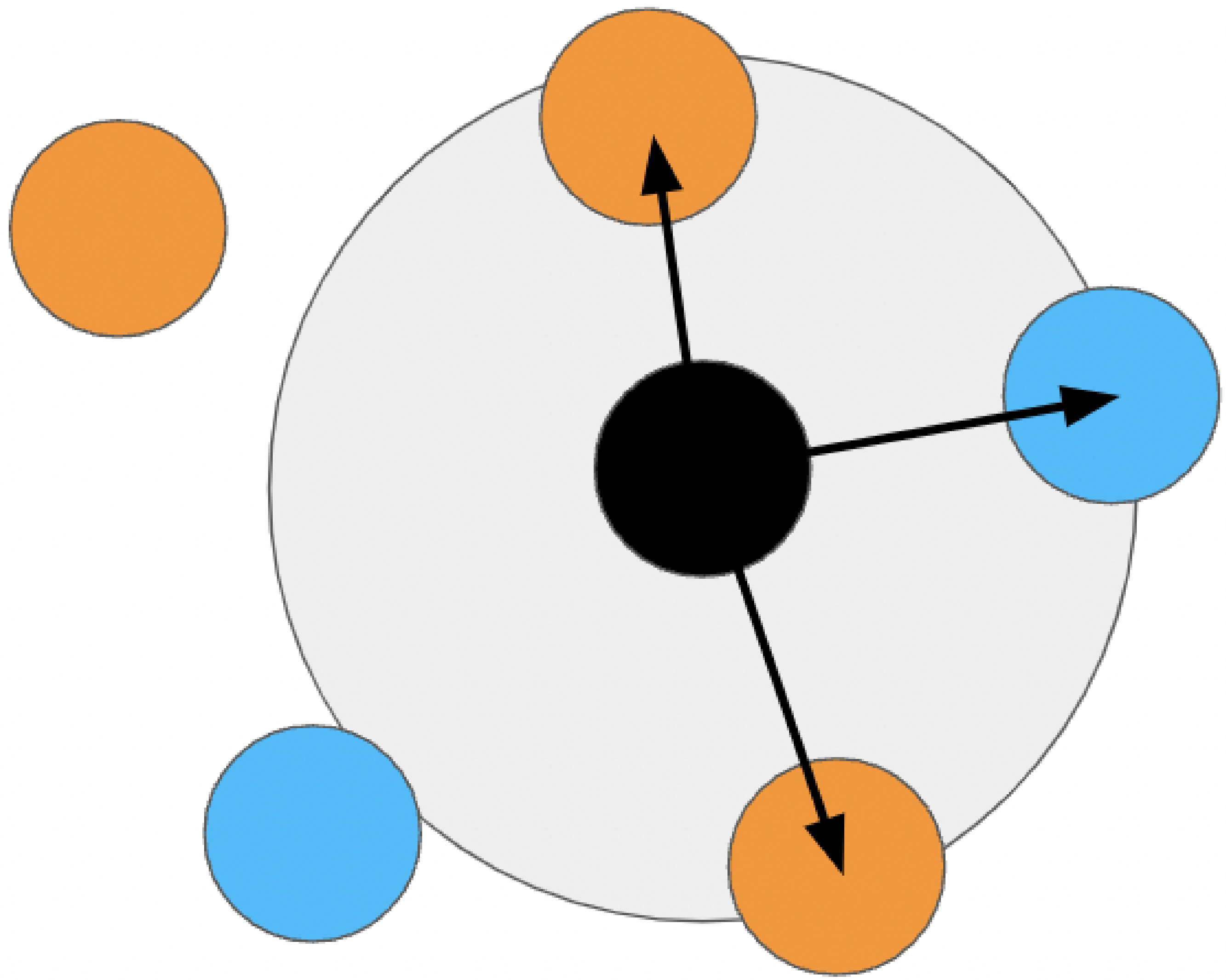
When to standardize: high variance

- Model in *linear* space

Examples:

- k-Nearest Neighbors (kNN)
- Linear regression
- K-Means Clustering

- Dataset features have *high variance*



When to standardize: different scales

- Features are on *different scales*

Example:

- Predicting house prices using *no. bedrooms* and *last sale price*

- Linearity assumptions

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Log normalization

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

What is log normalization?

- Useful for features with *high variance*
- Applies logarithm transformation
- Natural log using the constant e (≈ 2.718)

What is log normalization?

- Useful for features with *high variance*
- Applies logarithm transformation
- Natural log using the constant e (≈ 2.718)
- $e^{3.4} = 30$

- Captures relative changes, the magnitude of change, and keeps everything positive

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
```

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

```
col1     0.128958
log_2    2.262886
dtype: float64
```

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Scaling data

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

What is feature scaling?

- Features on different scales
- Model with linear characteristics
- Center features around 0 and transform to variance of 1
- 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

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

```python
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
```

```python
print(df_scaled.var())
```

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

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Standardized data and modeling

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

K-nearest neighbors

- **Data leakage:** non-training data is used to train the model

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

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42)
knn = KNeighborsClassifier()
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn.fit(X_train_scaled, y_train)
knn.score(X_test_scaled, y_test)
```

Let's practice!

Feature engineering

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

What is feature engineering?

Feature engineering: Creation of new features from existing ones

- Improve performance
- Insight into relationships between features
- Need to understand the data first!
- Highly 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

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

Let's practice!

Encoding categorical variables

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

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

```python
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
3          y           1
```

One-hot encoding

fav_color
blue
green
orange
green

Values: [blue, green, orange]

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

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

```
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
```

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Engineering numerical features

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

```python
print(temps)
```

```
     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
```

```python
temps["mean"] = temps.loc[:,"day1":"day3"].mean(axis=1)
print(temps)
```

```
     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(purchases)
```

```
                  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

```
purchases["date_converted"] = pd.to_datetime(purchases["date"])
purchases['month'] = purchases["date_converted"].dt.month
print(purchases)
```

```
               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
```

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Engineering text features

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Extraction

- **Regular expressions:** code to identify patterns

```python
import re
my_string = "temperature:75.6 F"
temp = re.search("\d+\.\d+", my_string)

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

```
75.6
```

- `\d+`
- `\.`
- `\d+`

Vectorizing text

TF/IDF: Vectorizes words based upon importance

- 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

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

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Feature selection

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

What is feature selection?

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

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

- Reducing noise
- Features are strongly statistically correlated
- Reduce overall variance

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Removing redundant features

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Redundant features

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

Scenarios for manual removal

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

Correlated features

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

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
```

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Selecting features using text vectors

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

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 ...]
```

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

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

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',
 ...
```

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

```
print(zipped_row)
```

```
{5: 0.1597882543332701,
 7: 0.26576432098763175,
 8: 0.18599931331925676,
 9: 0.26576432098763175,
 10: 0.13077355258450366,
 ...
```

Looking at word weights

```python
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,
 ...
```

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Dimensionality reduction

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

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

```python
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]
```

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

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

PCA caveats

- Difficult to interpret components
- End of preprocessing journey

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

UFOs and preprocessing

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Identifying areas for preprocessing

Important concepts to remember

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

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Categorical variables and standardization

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

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()`

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Engineering new features

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

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, slowly m...
12/31/2013 22:25	3 minutes	Three orange fireballs spotted by witness in E...

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

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Feature selection and modeling

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Feature selection and modeling

- Redundant features
- Text vector

Final thoughts

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

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

Congratulations!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

What you've learned

- Preparing data for modeling:
  - Missing data
  - Incorrect types
  - Standardize numerical values
  - Process categorical values
  - Feature engineering
  - Select features for modeling

Let's practice!

PREPROCESSING FOR MACHINE LEARNING IN PYTHON