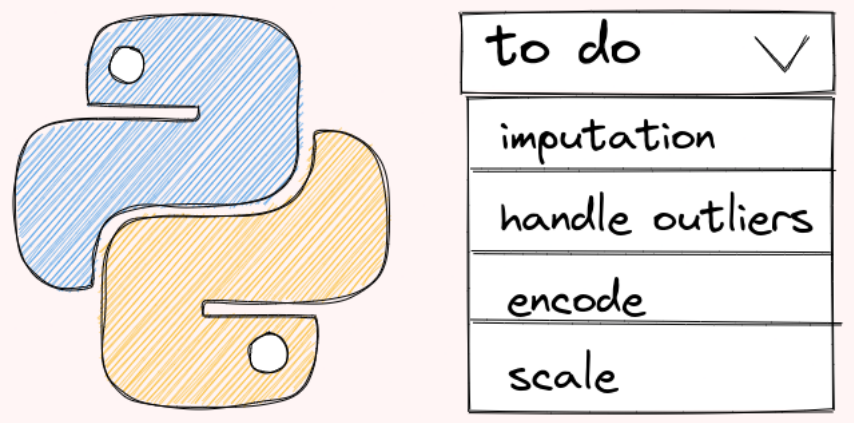
Efficient Python Tricks and Tools for Data Scientists - By Khuyen Tran

Feature Engineer

This section covers some libraries for feature engineering.



Split Data in a Stratified Fashion in scikit-learn

Normally, after using scikit-learn's `train_test_split`, the proportion of values in the sample will be different from the proportion of values in the entire dataset.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import numpy as np

X, y = load_iris(return_X_y=True)
np.bincount(y)
```

```
array([50, 50, 50])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)
```

```
# Get count of each class in the train set

np.bincount(y_train)
```

```
array([37, 34, 41])
```

```
# Get count of each class in the test set

np.bincount(y_test)
```

```
array([13, 16,  9])
```

If you want to keep the proportion of classes in the sample the same as the proportion of classes in the entire dataset, add `stratify=y`.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0, stratify=y)
```

```
np.bincount(y_train)
```

```
array([37, 37, 38])
```

```
np.bincount(y_test)
```

```
array([13, 13, 12])
```

Drop Correlated Features

```
$ pip install feature_engine
```

If you want to remove the correlated variables from a dataframe, use feature_engine.DropCorrelatedFeatures.

```
import pandas as pd
from sklearn.datasets import make_classification
from feature_engine.selection import DropCorrelatedFeatures

# make dataframe with some correlated variables
X, y = make_classification(
        n_samples=1000,
        n_features=6,
        n_redundant=3,
        n_clusters_per_class=1,
        class_sep=2,
        random_state=0,
    )

# trabsform arrays into pandas df and series
colnames = ["var_" + str(i) for i in range(6)]
X = pd.DataFrame(X, columns=colnames)
```

```
X.columns
```

```
Index(['var_0', 'var_1', 'var_2', 'var_3', 'var_4', 'var_5'],
dtype='object')
```

```
X[["var_0", "var_1", "var_2"]].corr()
```

	var_0	var_1	var_2
var_0	1.000000	0.938936	0.874845
var_1	0.938936	1.000000	0.654745
var_2	0.874845	0.654745	1.000000

Drop the variables with a correlation above 0.8.

```
tr = DropCorrelatedFeatures(variables=None, method="pearson",
threshold=0.8)

Xt = tr.fit_transform(X)

tr.correlated_feature_sets_
```

```
[{'var_0', 'var_1', 'var_2'}]
```

```
Xt.columns
```

```
Index(['var_0', 'var_3', 'var_4', 'var_5'], dtype='object')
```

Link to feature-engine.

Similarity Encoding for Dirty Categories Using dirty_cat

```
$ pip install dirty-cat
```

To capture the similarities among dirty categories when encoding categorical variables, use dirty_cat's `SimilarityEncoder` .

To understand how `SimilarityEncoder` works, let's start with the employee_salaries dataset.

```
from dirty_cat.datasets import fetch_employee_salaries
from dirty_cat import SimilarityEncoder

X = fetch_employee_salaries().X
X.head(10)
```

	gender	department	department_name	division	assignment_category	employee_positio
0	F	POL	Department of Police	MSB Information Mgmt and Tech Division Records...	Fulltime-Regular	Office Services Coordinator
1	M	POL	Department of Police	ISB Major Crimes Division Fugitive Section	Fulltime-Regular	Master Police Offi
2	F	HHS	Department of Health and Human Services	Adult Protective and Case Management Services	Fulltime-Regular	Social Worker IV
3	M	COR	Correction and Rehabilitation	PRRS Facility and Security	Fulltime-Regular	Resident Supervis
4	M	HCA	Department of Housing and Community Affairs	Affordable Housing Programs	Fulltime-Regular	Planning Specialis

	gender	department	department_name	division	assignment_category	employee_positio
5	M	POL	Department of Police	PSB 6th District Special Assignment Team	Fulltime-Regular	Police Officer III
6	F	FRS	Fire and Rescue Services	EMS Billing	Fulltime-Regular	Accountant/Audit
7	M	HHS	Department of Health and Human Services	Head Start	Fulltime-Regular	Administrative Specialist II
8	M	FRS	Fire and Rescue Services	Recruit Training	Fulltime-Regular	Firefighter/Rescue
9	F	POL	Department of Police	FSB Traffic Division Automated Traffic Enforce...	Fulltime-Regular	Police Aide

```
dirty_column = "employee_position_title"
X_dirty = df[dirty_column].values
X_dirty[:7]
```

```
array(['Office Services Coordinator', 'Master Police Officer',
       'Social Worker IV', 'Resident Supervisor II',
       'Planning Specialist III', 'Police Officer III',
       'Accountant/Auditor II'], dtype=object)
```

We can see that titles such as 'Master Police Officer' and 'Police Officer III' are similar. We can use `SimilaryEncoder` to encode these categories while capturing their similarities.

```
enc = SimilarityEncoder(similarity="ngram")
X_enc = enc.fit_transform(X_dirty[:10].reshape(-1, 1))
X_enc
```

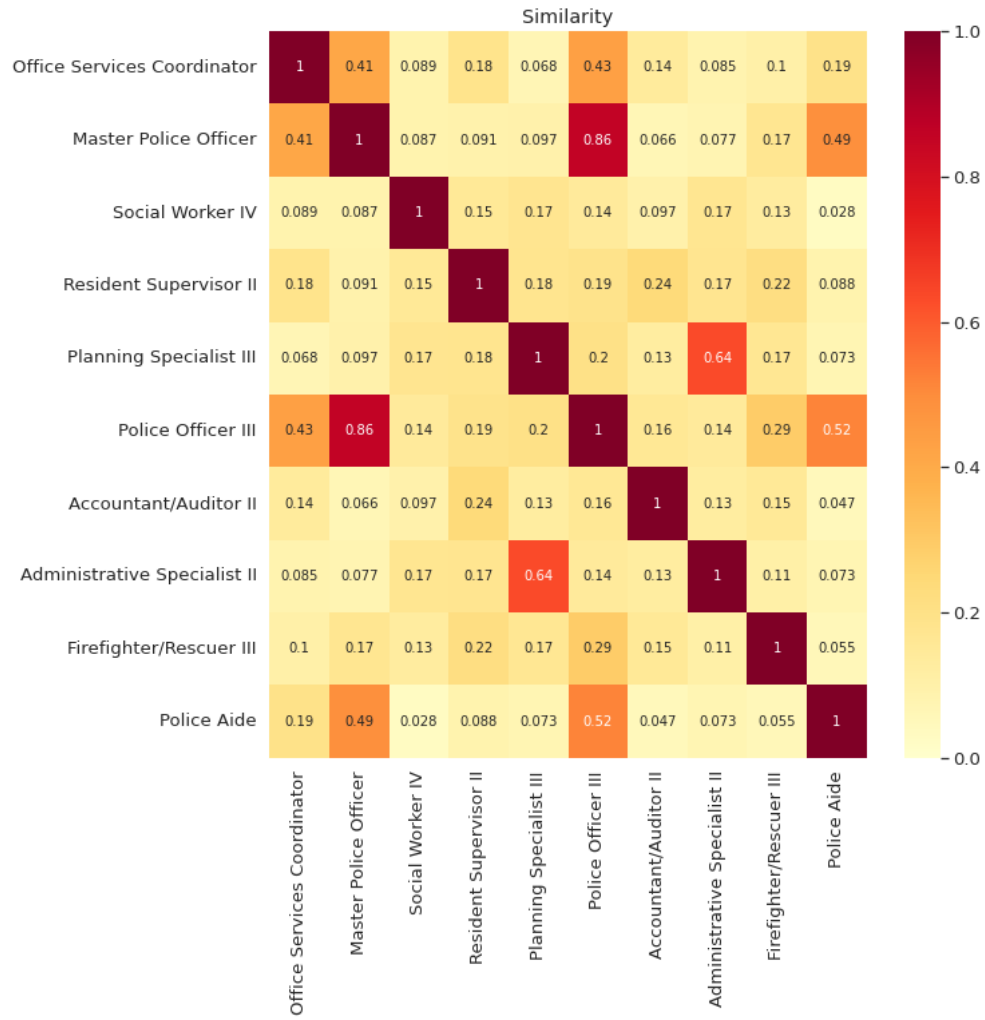
```
array([[0.05882353, 0.03125   , 0.02739726, 0.19008264, 1.        ,
        0.01351351, 0.05555556, 0.20535714, 0.08088235, 0.032     ],
       [0.008     , 0.02083333, 0.056     , 1.        , 0.19008264,
        0.02325581, 0.23076923, 0.56      , 0.01574803, 0.02777778],
       [0.03738318, 0.07317073, 0.05405405, 0.02777778, 0.032     ,
        0.0733945 , 0.        , 0.0625    , 0.06542056, 1.        ],
       [0.11206897, 0.07142857, 0.09756098, 0.01574803, 0.08088235,
        0.07142857, 0.03125   , 0.08108108, 1.        , 0.06542056],
```

```
       [0.04761905, 0.3539823 , 0.06976744, 0.02325581, 0.01351351,
        1.        , 0.02      , 0.09821429, 0.07142857, 0.0733945 ],
       [0.0733945 , 0.05343511, 0.14953271, 0.56      , 0.20535714,
        0.09821429, 0.26086957, 1.        , 0.08108108, 0.0625    ],
       [1.        , 0.05      , 0.06451613, 0.008     , 0.05882353,
        0.04761905, 0.01052632, 0.0733945 , 0.11206897, 0.03738318],
       [0.05      , 1.        , 0.03378378, 0.02083333, 0.03125   ,
        0.3539823 , 0.02631579, 0.05343511, 0.07142857, 0.07317073],
       [0.06451613, 0.03378378, 1.        , 0.056     , 0.02739726,
        0.06976744, 0.        , 0.14953271, 0.09756098, 0.05405405],
       [0.01052632, 0.02631579, 0.        , 0.23076923, 0.05555556,
        0.02      , 1.        , 0.26086957, 0.03125   , 0.        ]])
```

Cool! Let's create a heatmap to understand the correlation between the encoded features.

```
import seaborn as sns
import numpy as np
from sklearn.preprocessing import normalize
from IPython.core.pylabtools import figsize

def plot_similarity(labels, features):

    normalized_features = normalize(features)

    # Create correction matrix
    corr = np.inner(normalized_features, normalized_features)

    # Plot
    figsize(10, 10)
    sns.set(font_scale=1.2)
    g = sns.heatmap(corr, xticklabels=labels, yticklabels=labels,
vmin=0,
        vmax=1, cmap="YlOrRd", annot=True, annot_kws={"size": 10})

    g.set_xticklabels(labels, rotation=90)
    g.set_title("Similarity")


def encode_and_plot(labels):

    enc = SimilarityEncoder(similarity="ngram") # Encode
    X_enc = enc.fit_transform(labels.reshape(-1, 1))

    plot_similarity(labels, X_enc) # Plot
```

```
encode_and_plot(X_dirty[:10])
```



As we can see from the matrix above,

- The similarity between the same strings such as 'Office Services Coordinator' and 'Office Services Coordinator' is 1
- The similarity between somewhat similar strings such as 'Office Services Coordinator' and 'Master Police Officer' is 0.41
- The similarity between two very different strings such as 'Social Worker IV' and 'Polic Aide' is 0.028

Link to dirty-cat.

Link to my full article about dirty-cat.

Snorkel — Programmatically Build Training Data in Python

```
$ pip install snorkel
```

Imagine you try to determine whether a job posting is fake or not. You come up with some assumptions about a fake job posting, such as:

- If a job posting has few to no descriptions about the requirements, it is likely to be fake.
- If a job posting does not include any company profile or logo, it is likely to be fake.
- If the job posting requires some sort of education or experience, it is likely to be real.

```
import pandas as pd
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)


train_df = pd.read_pickle(
    "https://github.com/khuyentran1401/Data-science/blob/master/feature_engineering/snorkel_example/train_fake_jobs.pkl?raw=true"
)
train_df.head(5)
```

	job_id	title	location	department	salary_range	company_profile	des
12276	12277	Big Data Analyst	GB, WSM, London	Product Ops	NaN	Founded in 2010 by a team from Google's London...	Qubit: Cu Edge Big Engineeri ...
14680	14681	Instructional Advocate	US, GA, Savannah	NaN	NaN	We are an after-school program committed to as...	21st Cent Commun Learning is an ...
16518	16519	Software Developer	US, FL, Gainesville	NaN	NaN	352 Inc. is a full-service digital agency crea...	We partn great clie build sma

	job_id	title	location	department	salary_range	company_profile	des
15478	15479	Internship in India	IN, , Bangalore	NaN	NaN		London is paced city culture, d
16348	16349	Web Developer Backend Microservices (m/f)	DE, BE, 10969	Engineering	NaN	airfy prägt sicheres und einfach zu bedienende...	Design ar develop a microserv platform

How do you test which of these features are the most accurate in predicting fraud?

That is when Snorkel comes in handy. Snorkel is an open-source Python library for programmatically building training datasets without manual labeling.

To learn how Snorkel works, start with giving a meaningful name to each value:

```
from snorkel.labeling import labeling_function, PandasLFApplier,
LFAnalysis

FAKE = 1
REAL = 0
ABSTAIN = -1
```

We assume that:

- Fake companies don’t have company profiles or logos
- Fake companies are found in a lot of fake job postings
- Real job postings often requires a certain level of experience and education

Let’s test those assumptions using Snorkel’s `labeling_function` decorator. The `labeling_function` decorator allows us to quickly label instances in a dataset using functions.

```
@labeling_function()
def no_company_profile(x: pd.Series):
    return FAKE if x.company_profile == "" else ABSTAIN


@labeling_function()
def no_company_logo(x: pd.Series):
    return FAKE if x.has_company_logo == 0 else ABSTAIN
```

```
@labeling_function()
def required_experience(x: pd.Series):
    return REAL if x.required_experience else ABSTAIN


@labeling_function()
def required_education(x: pd.Series):
    return REAL if x.required_education else ABSTAIN
```

`ABSTAIN` or `-1` tells Snorkel not to make any conclusion about the instance that doesn't satisfy the condition.

Next, we will use each of these labeling functions to label our training dataset:

```
lfs = [
    no_company_profile,
    no_company_logo,
    required_experience,
    required_education,
]

applier = PandasLFApplier(lfs=lfs)
L_train = applier.apply(df=train_df)
```

```
100%|██████████████████████████████████████████████████| 13410/13410
[00:02<00:00, 5849.25it/s]
```

Now that we have created the labels using each labeling function, we can use `LFAnalysis` to determine the accuracy of these labels.

```
LFAnalysis(L=L_train,
lfs=lfs).lf_summary(Y=train_df.fraudulent.values)
```

	j	Polarity	Coverage	Overlaps	Conflicts	Correct	Incorrect	Emp. Acc.
no_company_profile	0	[1]	0.186204	0.186204	0.186204	459	2038	0.183821
no_company_logo	1	[1]	0.205742	0.205742	0.205742	459	2300	0.166365
required_experience	2	[0]	1.000000	1.000000	0.244295	12741	669	0.950112
required_education	3	[0]	1.000000	1.000000	0.244295	12741	669	0.950112

Details of the statistics in the table above:

- **Polarity**: The set of unique labels this LF outputs (excluding abstains)
- **Coverage**: The fraction of the dataset that is labeled

- **Overlaps**: The fraction of the dataset where this LF and at least one other LF agree
- **Conflicts**: The fraction of the dataset where this LF and at least one other LF disagree
- **Correct**: The number of data points this LF labels correctly
- **Incorrect**: The number of data points this LF labels incorrectly
- **Empirical** Accuracy: The empirical accuracy of this LF

Link to Snorkel.

My full article about Snorkel.