# Python Feature Engineering Cookbook

Stephen CUI[1]

December 15, 2022

[1]cuixuanStephen@gmail.com

# Contents

# Chapter 1

# Imputing Missing Data

Missing data, that is, the absence of values for certain observations, is an unavoidable problem in most data sources.

The act of replacing missing data with statistical estimates of missing values is called imputation. The goal of any imputation technique is to produce a complete dataset. There are multiple imputation methods that we can use, depending on whether the data is missing at random, the proportion of missing values, and the machine learning model we intend to use.

## 1.1 Technical requirements

We will use the Credit Approval dataset from the UCI Machine Learning Repository (https://archive.ics.uci.edu/). To prepare the dataset, follow these steps:

1. Visit https://archive.ics.uci.edu/ml/machine-learning-databases/credit-screening/.

2. Click on **crx.data** to download the data.

```
import random
import numpy as np
import pandas as pd
data = pd.read_csv('data/crx.data', header=None)
varname = [f"A{s}" for s in range(1, 17)]
data.columns = varname
data = data.replace("?", np.nan)
data["A2"] = data["A2"].astype("float")
data["A14"] = data["A14"].astype("float")
data["A16"] = data["A16"].map({"+": 1, "-": 0})
data.rename(columns={"A16":"target"}, inplace=True)
random.seed(37)
values = list(set([random.randint(0, len(data)) for p in range(0, 100)]))
data.loc[values, ["A3", "A8", "A9", "A10"]] = np.nan
data.to_csv('data/credit_approval_uci.csv', index=False)
```

**Tip**

Make sure you store the dataset in the same folder from which you will execute the code in the recipes.

## 1.2 Removing observations with missing data

**Complete Case Analysis (CCA)**, also called list-wise deletion of cases, consists of discarding observations with missing data. CCA can be applied to both categorical and numerical variables. With CCA, we preserve the distribution of the variables after the imputation, provided the data is missing at random and only in a small proportion of observations. However, if data is missing for many variables, CCA may lead to the removal of a large portion of the dataset.

### 1.2.1 How to do it...

```
import matplotlib.pyplot as plt
import pandas as pd
data = pd.read_csv('data/credit_approval_uci.csv')
with plt.style.context('ggplot'):
    data.isnull().mean().sort_values(ascending=True).plot.bar(rot=45)
    plt.ylabel("Proportion of missing data")
    plt.title("Proportion of missing data per variable")
```

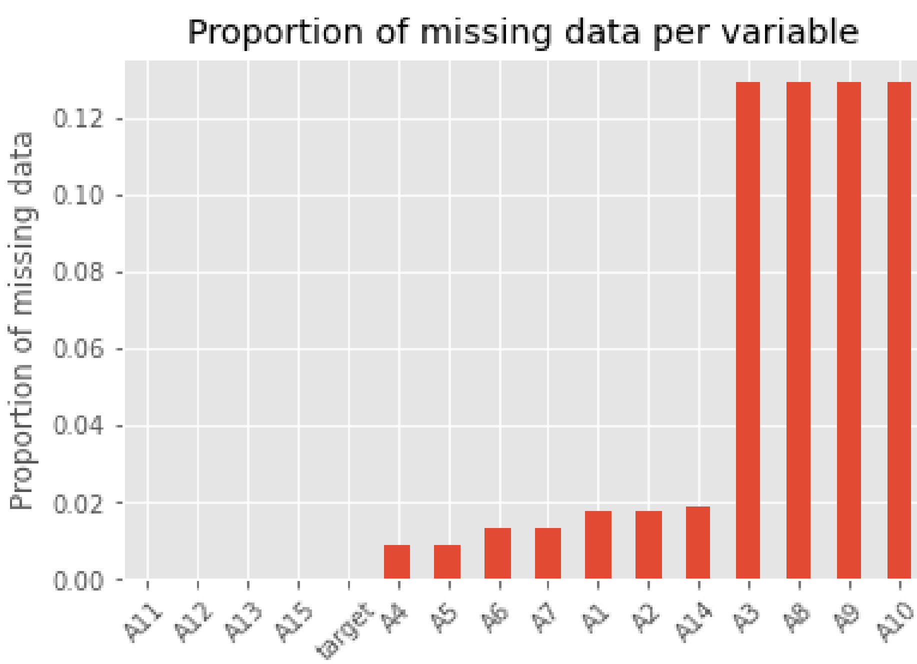


Figure 1.1: Proportion of missing data per variable

```
data_cca = data.dropna()
print(f"Total number of observations: {len(data)}")
print(f"Number of observations without missing data: {len(data_cca)}")
from feature_engine.imputation import DropMissingData
cca = DropMissingData(variables=None, missing_only=True)
cca.fit(data)
print(cca.variables_)
data_cca = cca.transform(data)
```

**Tip**

The dropna() method drops observations with any missing value by default. We can remove observations with missing data in a subset of variables like this: data.dropna(subset=["A3","A4"]).

**Tip**

To remove observations with missing data in a subset of variables, use DropMissingData(variables=['A3', 'A4']). To remove observations with missing values in at least 5% of the variables, use DropMissingData(threshold=0.95).

## 1.3 Performing mean or median imputation

Mean or median imputation consists of replacing missing values with the mean or median variable. The mean or median is calculated using a train set, and these values are used to impute missing data in train and test sets, as well as in all future data we intend to use with the machine learning model. Scikit-learn and feature-engine transformers learn the mean or median from the train set and store these parameters for future use out of the box.

**Tip**

Use mean imputation if variables are normally distributed and median imputation otherwise. Mean and median imputation may distort the distribution of the original variables if there is a high percentage of missing data.

### 1.3.1 How to do it...

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from feature_engine.imputation import MeanMedianImputer

data = pd.read_csv('data/credit_approval_uci.csv')

X_train, X_test, y_train, y_test = train_test_split(data.drop('target',
                                                              axis='columns'),
                                                    data['target'],
                                                    test_size=.3,
                                                    random_state=39)
numeric_vars = X_train.select_dtypes(exclude='O').columns.to_list()
numeric_vars
median_values = X_train[numeric_vars].median().to_dict()
median_values
for df in [X_train, X_test]:
# unable to modify X_train and X_test
# because df pointer to different address
#     df = df.fillna(value=median_values)
    df.fillna(value=median_values, inplace=True)
# equivalent to following codes
# X_train = X_train.fillna(value=median_values)
# X_test = X_test.fillna(value=median_values)
```

```python
X_train, X_test, y_train, y_test = train_test_split(data.drop('target',
                                                              axis='columns'),
                                                    data['target'],
                                                    test_size=.3,
                                                    random_state=39)
remaining_vars = [var for var in X_test.columns if var not in numeric_vars]
remaining_vars
imputer = SimpleImputer(strategy='median')
ct = ColumnTransformer([('imputer', imputer, numeric_vars)],
                       remainder='passthrough')
ct.fit(X_train)
ct.named_transformers_.imputer.statistics_
# warning: unable to perform median imputation
# for df in [X_train, X_test]:
#     df = ct.transform(df)
# return NumPy arrays
X_train = ct.transform(X_train)
X_test = ct.transform(X_test)
X_train = pd.DataFrame(X_train, columns=numeric_vars + remaining_vars)
X_train
X_test = pd.DataFrame(X_test, columns=numeric_vars + remaining_vars)
X_test

X_train, X_test, y_train, y_test = train_test_split(data.drop('target',
                                                              axis='columns'),
                                                    data['target'],
                                                    test_size=.3,
                                                    random_state=39)
imputer = MeanMedianImputer(imputation_method='median', variables=numeric_vars)
imputer.fit(X_train)
imputer.imputer_dict_
X_train = imputer.transform(X_train)
X_test = imputer.transform(X_test)
print(X_train[numeric_vars].isnull().any().any())
print(X_test[numeric_vars].isnull().any().any())
```

### 1.3.2 How it works...

The mean or median values should be learned from the train set variables. Thus, we divided the dataset into train and test sets using scikit-learn's `train_test_split()` function. The function takes the predictor variables, the target, the fraction of observations to retain in the test set, and a `random_state` value for reproducibility as arguments. We obtained a train set with 70% of the original observations and a test set with 30% of the original observations. The 70:30 split was done at random.

To replace the missing values using scikit-learn, we used `SimpleImputer()` with strategy set to ”median”. To impute only numerical variables, we used `ColumnTransformer()`, which takes the imputer and the numerical variable names in a list as parameters. With the passthrough argument set to ”remainder”, we make `ColumnTransformer()` **return all the variables in the final output, the imputed ones followed by the remaining ones.**

**Tip**

SimpleImputer() operates on the entire DataFrame and returns NumPy arrays. To impute just a subset of variables, we need to use ColumnTransformer(). In contrast, MeanMedianImputer() can take an entire DataFrame and yet it will only impute the specified variables, returning a pandas DataFrame.

## 1.4 Imputing categorical variables

Categorical variables usually contain strings as values, instead of numbers. We replace missing data in categorical variables with the most frequent category, or with a different string. Frequent categories are estimated using the train set and then used to impute values in the train, test, and future datasets. Thus, we need to learn and store these values, which we can do using scikit-learn and feature- engine's out-of-the-box transformers.

### 1.4.1 How to do it...