**Data Analysis + Python (5)** \_ October 05, 2023

Jihyun Lee

jerry\_jane@naver.com

https://medium.com/@pig30n77

Data Preprocessing

Meaning of data preprocessing

Data preprocessing means processing data suitable for the analysis to be performed.  
  
Data collected for analysis, including data generated in the field, are often not suitable for analysis.  
Missing values or outliers exist, or many variables that are not suitable for analysis tools degrade the quality of analysis results.  
  
To prevent this, a data preprocessing process is performed before the analysis is performed.

The types of data preprocessing are as follows.

* Cleaning Data : Treatment of missing values, checking and refining outliers, etc
* Data integration : Combining different data files, etc
* Data conversion : Scaling, Summary, etc
* Reduce data : Reducing variables, labeling, etc
* Processing unbalanced data :Under-sampling, over-sampling, etc
* Data segmentation : TRAIN, TEST data segmentation, etc

Perform data EDA, convert data according to the purpose, and look at the preprocessing performed one by one before starting the analysis.

Check and refine outliers  
  
the meaning of outliers

It refers to data in which observations and values differ significantly.

Outliers may occur due to variability in measurement, errors in experiments, or abnormalities in measuring equipment.

When collecting data, the value added by replacing the missing value may be identified as an outlier.  
  
Outliers can degrade the performance of the analysis model or adversely affect the analysis results, so it is recommended to check and remove them before analysis.

|  |
| --- |
| Outliers marked with values of a different format than the type of observation |
| weight\_number = pd.DataFrame({'Weight' : [60.0, 55.5, 'ERROR' , 70.5]})  weight\_number |
|  |
|  |

|  |
| --- |
| Outliers represented by values of the same type as the type of observation |
| score\_number = pd.DataFrame({'Score':[90, 85, 999, 100]})  score\_number |
|  |
|  |

|  |
| --- |
| Outliers caused by data collection errors |
| import pandas as pd  temp\_rate = pd.DataFrame({'Living room temperature' :[22.4, 22.3, 2345, 22.1]})  temp\_rate |
|  |
|  |

|  |
| --- |
| real observations that differ markedly from those observed |
| game\_time = pd.DataFrame({'Game time by date' : [2, 1 , 17 , 4]})  game\_time |
|  |
|  |

Outliers can exist in various forms.  
If outliers are displayed in a different format than the type of observation, the column may be in a different format from the observation.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
|  |  |  |  |

In the case #1, the observation is a float value, but the entire column is stored in the 'object' type due to an outlier entered as 'ERROR'. In this case, it is possible to check whether an abnormal value exists only by the form of the column.

For 2 and 3, it is an outlier stored as a value that is the same format as the observed value, but will not occur in the real environment.  
  
It is an extreme value that can affect the entire model, and if it is not refined, a completely different value from the observed value may be derived if calculations such as average are performed.

In the case of 4, it is actually a collected value, but it is a very unique value, and it is recommended to refine it because it can adversely affect the entire model as in the case of 2 and 3.

Check Outliers

Methods of identifying outliers vary depending on the data.  
If the criteria for outliers are determined through the data definition, outliers can be identified by filtering according to the rules.

If there is a generally known section of data such as test results (0-100 points), outliers can be identified by identifying values that deviate from the section of the data.

But otherwise, analysts should set the criteria for judging outliers.  
  
In general, for numerical variables, the 'IQR' method can be used as a criterion for determining outliers.

IQR(Inter Quantile Range)

:Using the outlier determination method of "Box plot" as it is

When the data is designated in ascending order and divided into exactly four equal parts, the three points that are divided, 25%, 50%, and 75%, are called the Q1(first), Q2(second), and Q3(third quartiles), respectively.

At this time, the difference between the value of Q3 and the value of Q1 is called IQR.

|  |
| --- |
| IQR = Q3 - Q1 |

Box Plot displays the range from Q3 to Q3 + 1.5 \* IQR and Q1 to Q1 - 1.5 \* IQR as 'beard'.

|  |
| --- |
| Beard |
| Q3 ~ Q3 + 1.5 \* IQR |
| Q1 ~ Q1 - 1.5 \* IQR |

And values outside the range of these beard are considered outliers and indicated by dots.

|  |
| --- |
|  |

Let's explore outliers in the 'color\_intensity' column of the win dataset using the IQR method.

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  from sklearn.datasets import load\_wine  wine\_load = load\_wine()  wine = pd.DataFrame(wine\_load.data, columns=wine\_load.feature\_names)  wine['Class'] = wine\_load.target  wine['Class'] = wine['Class'].map({0:'class\_0', 1:'class\_1', 2:'class\_2'}) |
| **import matplotlib.pyplot as plt**  **plt.boxplot(wine['color\_intensity'], whis=1.5)**  **plt.title('color\_intensity')**  **plt.show()** |
|  |
|  |

Four outliers are visible on the beard of Q3. Now, make a code with a function that brings outliers and check the location and value of outliers.

|  |
| --- |
| import numpy as np  def outliers\_iqr(dt, col):  quartile\_1, quartile\_3 = np.percentile(dt[col], [25, 75])    iqr = quartile\_3 - quartile\_1    lower\_whis = quartile\_1 - (iqr \* 1.5)  upper\_whis = quartile\_3 + (iqr \* 1.5)    outliers =dt[(dt[col] > upper\_whis) | (dt[col] < lower\_whis)]  return outliers[[col]]  outliers = outliers\_iqr(wine, 'color\_intensity')  outliers |
|  |
|  |

The method using IQR risks considering too much data as outliers.

Rather than unconditionally using IQR\*1.5 rules to explore and refine outliers, it is also a good idea to look at the type of data and adjust the value multiplied by IQR as needed.

Outlier purification  
  
Methods of purifying outliers include deleting rows with outliers and purifying outliers to appropriate values.

Refining outliers is convenient to perform in the same way as purifying missing values.

Remove Outliers  
  
If the number of data is very large and the rows with outliers are not large compared to the total data, the entire row with outliers can be deleted.

Since the number of data available for analysis decreases, it should be done after comparing the number of rows to be deleted with the length of the entire dataset.

|  |
| --- |
| drop\_outliers = wine.drop(index=outliers.index)  print('Original : ' , wine.shape)  print('Drop outliers :', drop\_outliers.shape) |
|  |
|  |
| Data shape before and after deleting outliers |

Outlier Replacement  
  
As a method of replacing outliers with other values, a method such as replacing missing values may be used.  
  
It can be done simply by making outliers NULL and replacing them with missing values.

Replace the outliers in the 'color\_intensity' column of the 'WINE' dataset with the average value.

|  |
| --- |
| #Change Outliers to NaN  wine.loc[outliers.index, 'color\_intensity'] = np.NaN  #Change outliers 'NaN' to mean value  wine['color\_intensity'] = wine['color\_intensity'].fillna(wine['color\_intensity'].mean())  wine.loc[outliers.index, 'color\_intensity'] |
|  |
|  |

Categorical variable processing  
  
Categorical variables(Categorical Data) are variables collected by dividing them into several categories, ranges, and sequences, and are also called Qualitative variables.

Categorical variables are mainly string data and are stored in object or category types from 'DataFrame'.

Even numerical values have no mathematical meaning, so the concept of mathematical operations such as addition cannot be applied.This includes nominal variables such as gender and occupation, and ordinal variables such as education level and satisfaction.

Categorical variables cannot be used directly by most analytical tools whose values are mathematical operations, requiring special treatment.  
  
We introduce how to turn categorical variables into dummy variables into forms that the model can understand.

A dummy variable is a variable that changes each category in a categorical variable to a column and fills it with 1 or 0 depending on whether the value of the original column belongs to that category.

It enables mathematical operation by changing categorical variables that cannot be mathematically operated to numbers in the form of 'True/False'.

|  |
| --- |
| Categorical\_variables = pd.DataFrame({'original\_data' : ['A', 'B', 'A' , 'C', 'C', 'B','C']})  Categorical\_variables |
|  |
|  |

|  |
| --- |
| #new\_A, new\_B, new\_c  new\_A = [1,0,1,0,0,0,0]  new\_B = [0,1,0,0,0,1,0]  new\_C = [0,0,0,1,1,0,1]  Categorical\_variables['COL\_A'] = new\_A  Categorical\_variables['COL\_B'] = new\_B  Categorical\_variables['COL\_C'] = new\_C  Categorical\_variables |
|  |
|  |

|  |
| --- |
| pd.get\_dummies(data, columns = [’categorical1’, ‘categorical2’]) |

Using the 'Class' column of iris data, we practice replacing categorical numbers with dummy variables.

|  |
| --- |
| import pandas as pd  from sklearn.datasets import load\_iris  iris = load\_iris()  iris = pd.DataFrame(iris.data, columns=iris.feature\_names)  iris['Class'] = load\_iris().target  iris['Class'] = iris['Class'].map({0:'Setosa', 1:'Versicolour', 2:'Virginica'})  iris\_dummy = pd.get\_dummies(iris, columns = ['Class'])  iris\_dummy |
|  |
|  |
|  |

\*Reference books: Finishing with one Python book, data analysis expert, big data analysis engineer practical preparation, SD Edu \_ [Author] Desa ramen, red fish, spare code

파이썬 한권으로 끝내기, 데이터분석전문가, 빅데이터분석기사 실기대비 ,SD에듀 \_[저자] 데싸라면, 빨간색 물고기, 자투리코드