Lab 6 (II): Use Pandas for Data Processing

Data Pre-processing

Data pre-processing (a.k.a. data cleaning or data wrangling) is the step of converting or mapping data from the initial raw form into another format, in order to make it ready for further analysis. In this lab, we will use Pandas for converting the text data to numeric data and manipulating the quantitative data.

Handle Missing Values

Value missing generates problem during the data analysis. In a Pandas data frame, the missing value will be represented by a mark 'nan'.

Consider the following data set:

Model	Photo Taking Speed	HDR Photo Taking Speed	Camera Speed Score	Camera Speed Score with flash
Samsung Galaxy S9+	0.7 sec	1 sec	nan	nan
Apple iPhone 8 Plus	0.95 sec	1.58 sec	nan	nan
Google Pixel 2 XL	0.951 sec	0.918 sec	700	843
Samsung Galaxy S9+	1 sec	1.1 sec	nan	nan
Motorola Moto Z2 Play	1.1 sec	2.1 sec	948	639
OnePlus 5	1.1 sec	1.5 sec	682	682
Google Pixel	1.1 sec	1.8 sec	nan	nan
…	…	…	…	…

*** Sample data file **smartphone_camspeed.csv**. You can find the data file on our course web page.*

We have different ways to handle the missing values, such as *dropping the data entry*, *replacing the missing values*, etc.

Dropping the data entry

Dropping the data entry may bring some impact to the data analysis result since the other fields in the entry are also deleted. To drop the data entry with missing data in any column, we can use the following code:

```
1  import pandas as pd
2  df = pd.read_csv('smartphone_camspeed.csv')
3  df.dropna(inplace=True)
```

We use the **dropna()** function to drop the rows with missing values. With the **inplace** parameter, we can write the result back to the original data frame.

The following code is used to drop the rows with missing values in the "Camera Speed Score" column. But you can still see some 'nan' in the column "Camera Speed Score with Flash".

```
1  import pandas as pd
2  df = pd.read_csv('smartphone_camspeed.csv')
3  df.dropna(subset=['Camera Speed Score'], inplace=True)
```

Replacing the missing values

Replacing data is better since no data is wasted. However, it is less accurate because we need to replace missing data with a guess.

In the following example, we replace the missing values in the 'Camera Speed Score' column with zero.

```
1  import pandas as pd
2  df = pd.read_csv(smartphone_camspeed.csv')
3  import numpy as np
4  df['Camera Speed Score'] = df['Camera Speed Score'].replace(np.nan, 0)
```

- We use the **Pandas' replace()** function to replace the missing values with *0*. (line 4)

Then, we get the following result:

Model	Photo Taking Speed	HDR Photo Taking Speed	Camera Speed Score	Camera Speed Score with flash
Samsung Galaxy S9+	0.7 sec	1 sec	0	nan
Apple iPhone 8 Plus	0.95 sec	1.58 sec	0	nan
Google Pixel 2 XL	0.951 sec	0.918 sec	700	843
Samsung Galaxy S9+	1 sec	1.1 sec	0	nan
Motorola Moto Z2 Play	1.1 sec	2.1 sec	948	639
OnePlus 5	1.1 sec	1.5 sec	682	682
Google Pixel	1.1 sec	1.8 sec	0	nan
…	…	…	…	…

Data Formatting

Data are usually collected from different places and stored in different formats. Bring the data into a common standard of expression allows users to make a meaningful comparison. But, the data type may be incorrectly established or may not be suitable for calculation. For example, the "Photo Taking Speed" column stores the data in string format in "#.# sec" form. We cannot simply find the fastest photo taking time.

Model	Photo Taking Speed	…
Samsung Galaxy S9+	0.7 sec	…
Apple iPhone 8 Plus	0.95 sec	…
Google Pixel 2 XL	0.951 sec	…

Samsung Galaxy S9+	1 sec	…
Motorola Moto Z2 Play	1.1 sec	…
OnePlus 5	1.1 sec	…
Google Pixel	1.1 sec	…
…	…	…

Therefore, we need to convert the data into float format so that we then can do a simple comparison. We use the following code to convert the data.

```
1  import pandas as pd
2  df = pd.read_csv('smartphone_camspeed.csv')

3  df['Photo Taking Speed'] = df['Photo Taking Speed'].str.replace('sec', '')

4  df['Photo Taking Speed'] = pd.to_numeric(df['Photo Taking Speed'])
```

We use **df['Photo Taking Speed'].str** to retrieve the values stored in the "Photo Taking Speed" as strings. Originally, the values are stored as objects. Then, we use **string's replace()** function to replace "sec" with *an empty string*. That means we delete the "sec" word from the value. (line 3)

The **Pandas' to_numeric()** function converts the values from object format to float format. (line 4)

You may check the data type by running the following statement in the console:

```
df['Photo Taking Speed'].head(5)
```

```
Out:
0    0.700
1    0.950
2    0.951
3    1.000
4    1.100
Name: Photo Taking Speed, dtype: float64
```

Data Normalization

Different columns of numerical data may have very different ranges, and direct comparison is often not meaningful. Normalization is a way to bring all data into a similar range for more useful comparison.

Consider the following data set.

Model	Photo Taking Speed	Camera Speed Score	…
Google Pixel 2 XL	0.951	700	…
Motorola Moto Z2 Play	1.1	948	…
OnePlus 5	1.1	682	…
BlackBerry Motion	1.13	689	…
Google Pixel 2	1.16	1109	…
Samsung Galaxy S8	1.2	281	…
…	…	…	…

If we plot a line chart that contains both “photo Taking Speed” column and “Camera Speed Score” column, we will find the line of “Photo Taking Speed” becomes a straight line and shows at the bottom of the chart. Before we start, type "**%matplotlib qt**" in the Spyder console.

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np

df = pd.read_csv('smartphone_camspeed.csv')
df.dropna(inplace=True)

df['Photo Taking Speed'] = df['Photo Taking Speed'].str.replace('sec', '')
df['Photo Taking Speed'] = pd.to_numeric(df['Photo Taking Speed'])

n = 5
index = np.arange(n)
fig, ax = plt.subplots()

bar_width = 0.35

ax.bar(index + bar_width, df['Photo Taking Speed'][0:n], bar_width,
                color='r',
                label='Photo Taking Speed')

ax.bar(index, df['Camera Speed Score'][0:n], bar_width,
                color='b',
                label='Camera Speed Score')

ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(df['Model'][0:n], rotation=45)
ax.legend()
plt.show()
```

A simple way to normalize a vector of positive data is to convert each data *x* into (*x-min*)/(*max-min*), where *min* is the minimum value in the vector, and *max* is the maximum value in the vector. E.g., we can create a new column named 'Camera Speed Score Norm' and show it:

```
df['Camera Speed Score Norm'] = (df['Camera Speed Score'] - df['Camera Speed Score'].min()) /
                                (df['Camera Speed Score'].max() - df['Camera Speed Score'].min())

ax.bar(index, df['Camera Speed Score Norm'][0:n], bar_width, color='b', label='Camera Speed Score Norm')
```

Categorical Values to Quantitative Values Encoding

We are going to discuss how to turn **categorical values** into **quantitative values** in Python. Most statistical programs/functions cannot take in objects or strings as input. And, they only take the numbers as their training data.

Consider the following data set:

Vehicle	Fuel Type	…
A	Gas	…
B	Diesel	…
C	Gas	…

D	Electricity	…
E	Diesel	…
F	Diesel	…

The **Fuel Type** column as a category list that includes 3 values – Gas, Diesel, and Electricity, which are in string format. For further analysis, we convert the values in the **Fuel Type** column into integer format. We encode the values by adding new columns corresponding to the categories, so as to get the following data set:

Vehicle	Fuel Type	Gas	Diesel	Electricity	…
1	Gas	1	0	0	…
2	Diesel	0	1	0	…
3	Gas	1	0	0	…
4	Electricity	0	0	1	…
5	Diesel	0	1	0	…
6	Diesel	0	1	0	…
…	…	…	…	…	…

In the table above, you can see that the **Gas** column of *vehicle 1* is equal to 1 and the rest columns are equal to 0 because the fuel type of the *vehicle 1* is gas. This encoding method is often called "**One-hot encoding**". Then, we use some technique for other vehicles.

In Python, we can use Pandas **get_dummies()** function to do so. Let's try the example code below:

```
1  import pandas as pd

2  df = pd.read_csv('vehicle.csv')

3  dummies = pd.get_dummies(df['Fuel Type'])

4  df = pd.merge(left=df, right=dummies, left_index=True, right_index=True)

5  print(df)
```

- We open a CSV file and store the data in a data frame named *df*. (line 2)
- A new data frame **dummies** is used for containing the quantitative values generated based on the "Fuel Type" column. (line 3)
- The Pandas **merge()** function combines two data frames – the rows are matched using the indices. (line 4)

The result is as follows:

```
           Fuel Type  Diesel  Electricity  Gas
Vehicle
1                Gas       0            0    1
2        Electricity       0            1    0
3                Gas       0            0    1
4             Diesel       1            0    0
5                Gas       0            0    1
6             Diesel       1            0    0
7             Diesel       1            0    0
...
```

Exercise:

Preparing product record

A data file retrieved from a company's database about the product including purchasing price and selling price. However, somehow the category and product fields are combined into a single field named CategoryProduct. The first four characters in this field represent the categories, and other characters are the name of the product. The following is the partial data. For the first record, the category is SEAF, the product name is Konbu.

Year	CategoryProduct	Purchasing Price	Selling Price
2010	SEAFKonbu	33	61.44
2010	SEAFInlagd Sill	37	68.4
2010	BEVESasquatch Ale	98	179.2
2010	GRAIFilo Mix	103	187.6
…	…	…	…

Write a program to read the data file **lab6_sampledata1.csv**, use the Pandas package to split the column **CategoryProduct** into two columns – **Category** and **Product**. Then, save the result to another csv file named **lab6_result.csv**.

Hint:

- Use String slices to break the **CategoryProduct** column into two.
- To create a new column with name '**Category**', just use **df['Category'] =** …
- Use **del df['CategoryProduct']** to delete the column in data frame **df**.