Inconsistent Data

This lesson will focus on some of the common inconsistencies present in datasets and how to deal with them using pandas.

Inconsistency in data arises due to errors in collecting data. For instance, if the data was collected from multiple sources, or if the data was collected by multiple people who did not follow the same format of collecting data, then there is a high chance of inconsistencies in the data.

In this lesson, we will be cleaning the Credit Cards Default Dataset. This dataset is a very good example of the kind of inconsistencies that are present in most datasets.

Credit cards default dataset

The documented details of individual columns are mentioned below. But we will see that our dataset will not be consistent with this format.

```
# There are 25 variables:

# ID: ID of each client
# LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementa
# GENDER: Gender (1=male, 2=female)
# EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
# MARRIAGE: Marital status (1=married, 2=single, 3=others)
# AGE: Age in years
# PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=p
# PAY_2: Repayment status in August, 2005 (scale same as above)
# PAY_3: Repayment status in July, 2005 (scale same as above)
# PAY_4: Repayment status in June, 2005 (scale same as above)
# PAY_5: Repayment status in May, 2005 (scale same as above)
# PAY_6: Repayment status in April, 2005 (scale same as above)
# BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
# BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)

# BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
# BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
# BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
# BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
# PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
# PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
# PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
# PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
# PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
# PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
# default.payment.next.month: Default payment (1=yes, 0=no)
```

Let's load the dataset.

```
import pandas as pd

# Read Data
df = pd.read_csv('credit_card.csv')
# Print head and column names
print(df.head())
print(df.columns)
```

Just by looking at the output, we can see that pandas keeps serial numbers for us automatically, and since we have IDs in the `ID` column, we do not need the first column, we can remove it. We can use the `drop` function to drop columns by specifying their names.

```
import pandas as pd
df = pd.read_csv('credit_card.csv')
# Drop Extra Column
df = df.drop(columns = ['Unnamed: 0'])
print(df.head())
```

We have used the `drop` function on **line 4** and dropped the extra first column as can be seen from the output of **line 5**.

Categorical variables

The usual check for inconsistent data is usually the number of unique values that categorical variables take.

```python
import pandas as pd
df = pd.read_csv('credit_card.csv')

df = df.drop(columns = ['Unnamed: 0'])

# GENDER: Gender (1=male, 2=female)
# EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
# MARRIAGE: Marital status (1=married, 2=single, 3=others)
categorical_variables = ['GENDER','EDUCATION','MARRIAGE']
print('Number of unique values of categorical variables')
print(df[categorical_variables].nunique())

# Print Unique values
print('Unique values of EDUCATION : ', df['EDUCATION'].unique())
print('Unique values of MARRIAGE : ', df['MARRIAGE'].unique())

# Find inconsistent values and update
condition = (df['EDUCATION'] == 0 ) | (df['EDUCATION'] == 5 ) | (df['EDUCATION'] == 6 )
df.loc[condition,'EDUCATION'] = 4
print('Unique values of EDUCATION after changing: ', df['EDUCATION'].unique())

# Change MARRIAGE values
condition = df['MARRIAGE'] == 0
df.loc[condition,'MARRIAGE'] = 3
print('Unique values of MARRIAGE after changing: ', df['MARRIAGE'].unique())
```

In **line 11**, we selected the categorical variables that we had written in a list in **line 10**, and used the function `nunique` to output the number of unique values that each variable takes. By looking at the output of **line 11**, we see that `EDUCATION` and `MARRIAGE` have more unique values than specified. Therefore, we print these unique values by using the `unique` function in **lines 14-15**.

In the case of `EDUCATION`, the label `0` is undocumented and `5` and `6` indicate unknown values. We can replace all three of these values with `4` as that will indicate unknown or other levels of education than `1` (graduate school), `2` (university), and `3` (high school). To accomplish this, we define `condition` to be the rows where the values of `EDUCATION` is `0`, `5` or `6` in **line 18**. The operator `|` means that we will select the row if any of the three conditions is true. In **line 19**, we use `loc` to specify the location where we will be making the changes. We give `condition` to specify the rows and `EDUCATION` to specify the column. We place `4` at these locations.

In the same way, in **line 24** we have placed `3` at the places which had `0` in the

MARRIAGE columns as 0 is undocumented. In **lines 20** and **25**, we check the

unique values in EDUCATION and MARRIAGE respectively to confirm our changes.

Payment delay variables

Now let's check the payment delay variables for inconsistencies. We can use the describe function to see if the columns have values outside the scale by looking at the minimum and maximum values instead of looking at the unique values for each column.

```
import pandas as pd
df = pd.read_csv('credit_card.csv')
df = df.drop(columns = ['Unnamed: 0'])

condition = (df['EDUCATION'] == 0 ) | (df['EDUCATION'] == 5 ) | (df['EDUCATION'] == 6 )
df.loc[condition,'EDUCATION'] = 4

condition = df['MARRIAGE'] == 0
df.loc[condition,'MARRIAGE'] = 3

# PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=p
# PAY_2: Repayment status in August, 2005 (scale same as above)
# PAY_3: Repayment status in July, 2005 (scale same as above)
# PAY_4: Repayment status in June, 2005 (scale same as above)
# PAY_5: Repayment status in May, 2005 (scale same as above)
# PAY_6: Repayment status in April, 2005 (scale same as above)

df.rename(columns = {'PAY_0':'PAY_1'},inplace=True)

payment_variables = ['PAY_1','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6']

print('Output of Describe')
print(df[payment_variables].describe())

condition = df['PAY_1'] < 0
df.loc[condition,'PAY_1'] = 0

condition = df['PAY_2'] < 0
df.loc[condition,'PAY_2'] = 0

condition = df['PAY_3'] < 0
df.loc[condition,'PAY_3'] = 0

condition = df['PAY_4'] < 0
df.loc[condition,'PAY_4'] = 0

condition = df['PAY_5'] < 0
df.loc[condition,'PAY_5'] = 0

condition = df['PAY_6'] < 0
df.loc[condition,'PAY_6'] = 0

print('Output of Describe after replacing values')
print(df[payment_variables].describe())
```

When we look at the names of the payment variables we see an inconsistency that the names follow a pattern. Therefore, we change the name of the variable for September from `PAY_0` and to `PAY_1` in **line 18**.

We write the names of these payment variables in a list in **line 20**. Then we use the function `describe` on these columns in **line 23**. We see that the minimum value for all of these columns is -2, which probably indicates that the payment was duly paid. To deal with this, we decide to replace all negative values in these columns with 0.

We locate rows that have negative values in the PAY_1 column on **line 25**, and then set such cells to `0` on **line 26**. The same process is repeated for `PAY_2`, `PAY_3`, ..., `PAY_6` columns as well.

This was a lesson on how to deal with inconsistent data. In the next lesson, we will see how to deal with *outliers* in the data.