Chapter 01 - Data exploration and cleaning

December 4, 2022

1 Chapter 01: Data exploration and cleaning

Welcome to your first jupyter notebook! The first thing to know about Jupyter notebooks is that there are two kinds of cells. This is a markdown cell.

There are a lot of different ways to mark up the text in markdown cells, including **bold** and *italics*.

The next one will be a `code` cell.

[ ]:
```
import pandas as pd
```

[ ]:
```
df = pd.read_excel('data_cards.xlsx')
```

1.1 Verificación de la integridad de datos

Es vital entender qué significa cada fila y columna de un dataset. En este caso, se trata del nivel de crédito y datos financieros, cada fila representa una cuenta bancaria. También se necesita elaborar o recurrir al diccionario de datos que es suna descripción del dataset de estudio, en este se detallan sus características y contexto.

[ ]:
```
df.shape
```

[ ]:
```
df.info()
```

[ ]:
```
df.columns.values.tolist()
```

[ ]:
```
df.head()
```

Ahora verificamos si la columna ID tiene tantos valores únicos como filas en el dataset. El método "nunique()" hace un conteo de los valores únicos en el dataset y, como se puede ver, la cantidad de filas registradas en el dataset es mayor. Esto significa que existen valores duplicados en la columna **ID**

[ ]:
```
df['ID'].nunique()
```

Ahora podemos usar el método **value_counts()** para hacer una lista de los valores únicos en la columna **ID** junto a la cantidad de veces que se repite cda uno de ellos.

[ ]:
```
type(id_counts)
```

[ ]:
```
id_counts = df['ID'].value_counts()
id_counts.head(10)
```

En efecto, existen filas con **ID** repetidos, pero si aplicamos la función value_counts() De los 29687 registros, 29374 solo están presentes una vez, en cambio 313 se repiten 2 veces. Nótese que no existen casos con un ID que se repite más de 2 veces.

[ ]:
```
id_counts.value_counts()
```

Resumiendo el proceso anterior…

La función values_counts se encarga de contar las veces que un valor se repite a lo largo de toda la columna. Luego creamos una máscara booleana (boolean mask) para filtrar los valores que se repiten 2 veces. Como es de esperar, la variable dupe_mask y id_counts tienen la misma longitud.

[ ]:
```
df = pd.read_excel('../Para revisar/data_cards.xlsx')
id_counts = df['ID'].value_counts()
#id_counts.head()
dupe_mask = id_counts == 2
dupe_mask[0:5]
```

El resultado que obtenido en la última entrada refleja a un dataframe con una columna que se identifica con el "ID" de la tabla original. A este tipo de estructura también se conoce como "series" de pandas.

[ ]:
```
type(dupe_mask)
```

Dado que se requiere trabajar solo con las posiciones del índide, estos serán apartados con el método index

[ ]:
```
id_counts.index[:5]
```

Existen 313 "ID" repetidos

[ ]:
```
# Aplicación del filtro
dupe_ids = id_counts.index[dupe_mask]
# Transformación a una lista
dupe_ids = list(dupe_ids)
# Consulta de la longitud de la lista
len(dupe_ids)
```

Verificación de los resultados

[ ]:
```
dupe_ids[:5]
```

Filtrando los primeros tres valores duplicados en el dataset

[ ]:
```
df.loc[df['ID'].isin(dupe_ids[:3]),:]
```

Por otro lado, si se pretende hallar todos los registro repetidos, no será necesario adicionar segmentaciones.

[ ]:
```
df.loc[df['ID'].isin(dupe_ids),:].head()
```

One approach to deal with this issue would be to find rows that have all zeros, except for the first column, which has the IDs. These would be invalid data in any case, and it may be that if we get rid of all of these, we would also solve our problem of duplicate IDs. We can find the entries of the DataFrame that are equal to zero by creating a Boolean matrix that is the same size as the whole DataFrame, based on the "is equal to zero" condition.

[ ]:
```
df_zero_mask = df == 0
```

[ ]:
```
feature_zero_mask = df_zero_mask.iloc[:,1:].all(axis=1)
```

[ ]:
```
sum(feature_zero_mask)
```

El cálculo anterior nos dice que existen 315 filas que contienen 0 en todas columnas excepto la primera. Ya que es más grande que las filas duplicadas (313), podríamos eliminarlas solucionando el problema de los registros duplicados.

[ ]:
```
df_clean_1 = df.loc[~feature_zero_mask,:]
```

[ ]:
```
df_clean_1.shape
```

[ ]:
```
df_clean_1['ID'].nunique()
```

Con el problema solucuonado, se procede a guardar el resultado como un archivo .csv

[ ]:
```
df_clean_1.to_csv('../Para revisar/data_cards.csv', index = False)
```

Evaluación de los resultación

[ ]:
```
df_clean_1.info()
```

[ ]:
```
df_clean_1['PAY_1'].head(5)
```

[ ]:
```
df_clean_1['PAY_1'].value_counts()
```

The preceding output reveals the presence of two undocumented values: 0 and –2, as well as the reason this column was imported by pandas as an object data type, instead of int64 as we would expect for integer data: there is a 'Not available' string present in this column, symbolizing missing data.

[ ]:
```
valid_pay_1_mask = df_clean_1['PAY_1'] != 'Not available'
valid_pay_1_mask[:5]
```

We see that 26,664 rows do not have the value 'Not available' in the PAY_1 column. We saw from the value count that 3,021 rows do have this value. Does this make sense? From Figure 1.23 we know there are 29,685 entries (rows) in the dataset, and 29,685 – 3,021 = 26,664, so this checks out.

[ ]:
```
sum(valid_pay_1_mask)
```

```
# Es importante que después de la tranformación se escriba el método "copy"
# para evitar conflictos de encademamiento en modificaciones posteriores
df_clean_2 = df_clean_1.loc[valid_pay_1_mask,:].copy()
```

```
df_clean_2.shape
```

```
df_clean_2['PAY_1'].value_counts()
```

Cambio de tipo de columna

```
df_clean_2['PAY_1'] = df_clean_2['PAY_1'].astype('int64')
df_clean_2[['PAY_1', 'PAY_2']].info()
```

```
df_clean_2.to_csv('../Exploratory_Analysis/df_clean_2.csv', index = False)
```

1.2 Exploring the Credit Limit and Demographic Features

```
import pandas as pd
import matplotlib.pyplot as plt #import plotting package
import matplotlib as mpl #additional plotting functionality
mpl.rcParams['figure.dpi'] = 100 #400 #high resolution figures
```

This imports matplotlib and uses .rcParams to set the resolution (dpi = dots per inch) for a nice crisp image; you may not want to worry about this last part unless you are preparing things for presentation, as it could make the images quite large in your notebook.

```
df_clean_2 = pd.read_csv('../Para revisar/df_clean_2.csv')
```

Gráfico para una visualización preliminar de los datos

```
df_clean_2[['LIMIT_BAL', 'AGE']].hist()
```

Estadísticas descriptivas

```
df_clean_2[['LIMIT_BAL', 'AGE']].describe()
```

```
df_clean_2['EDUCATION'].value_counts()
```

```
df_clean_2['EDUCATION'].replace(to_replace=[0, 5, 6],\
                                value=4, inplace=True)
df_clean_2['EDUCATION'].value_counts()
```

Note that here we make this change in place (inplace=True). This means that, instead of returning a new DataFrame, this operation will make the change on the existing DataFrame.

```
df_clean_2['MARRIAGE'].value_counts()
```

```
df_clean_2['MARRIAGE'].replace(to_replace=[0],\
                                value=3, inplace=True)
df_clean_2['MARRIAGE'].value_counts()
```

Después de limpiar los datos, guardaremos los cambios

[ ]:
```
df_clean_2.to_csv('../Exploratory_Analysis/df_clean_2.csv', index=False)
```

1.3 Deep Dive: Categorical Features

[ ]:
```
df_clean_2 = pd.read_csv('../Para revisar/df_clean_2.csv')
df_clean_2.groupby('EDUCATION').agg({'default payment next '\
'month':'mean'})\
.plot.bar(legend=False)
plt.ylabel('Default rate')
plt.xlabel('Education level: ordinal encoding')
plt.show()
```

1.4 Implementing OHE for a Categorical Feature

Implementar un modelo OHE consiste en tranformar el procesado de una variable después de haber sido cargada

[ ]:
```
import pandas as pd
import matplotlib as mpl #additional plotting functionality
mpl.rcParams['figure.dpi'] = 400 #high resolution figures
```

[ ]:
```
df_clean_2 = pd.read_csv('../Para revisar/df_clean_2.csv')
```

Creación de una columna vacía que contenga el texto *none*

[ ]:
```
df_clean_2['EDUCATION_CAT'] = 'none'
```

Examinamos los resultados

[ ]:
```
df_clean_2[['EDUCATION', 'EDUCATION_CAT']].head(10)
```

Creamos un diccionario con las descirpciones correspondientes

[ ]:
```
cat_mapping = {1: "graduate school",\
               2: "university",\
               3: "high school",\
               4: "others"}
```

Usamos la función map

[ ]:
```
df_clean_2['EDUCATION_CAT'] = df_clean_2['EDUCATION']\
                                .map(cat_mapping)
df_clean_2[['EDUCATION', 'EDUCATION_CAT']].head(10)
```

Aplicación de la técnica OHE (One hot encoding)

[ ]:
```
edu_ohe = pd.get_dummies(df_clean_2['EDUCATION_CAT'])
edu_ohe.head(10)
```

[ ]:
```
# Unión de los dataframes
df_with_ohe = pd.concat([df_clean_2, edu_ohe], axis=1)
df_with_ohe[['EDUCATION_CAT', 'graduate school',\
'high school', 'university', 'others']].head(10)
```

Guardando los resultados

[ ]:
```
df_with_ohe.to_csv('../Para revisar/Chapter_1_cleaned_data.csv', index=False)
```

1.5 Exploring the Financial History Features in the Dataset

[35]:
```
import pandas as pd
import matplotlib.pyplot as plt #import plotting package
import matplotlib as mpl #additional plotting functionality
mpl.rcParams['figure.dpi'] = 150 #high resolution figures
import numpy as np
```

[2]:
```
df = pd.read_csv('../Para revisar/Chapter_1_cleaned_data.csv')
```

[6]:
```
df.columns.values
```

[6]:
```
array(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_1',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',
       'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6',
       'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5',
       'PAY_AMT6', 'default payment next month', 'EDUCATION_CAT',
       'graduate school', 'high school', 'others', 'university'],
      dtype=object)
```

1.5.1 Análisis del estado actual de pago

Primero indicamos las columnas de interés en una lista

[7]:
```
pay_feats = ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5','PAY_6']
```

Filtramos y resumimos las variables de interés

[8]:
```
df[pay_feats].describe()
```

[8]:

	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	\
count	26664.000000	26664.000000	26664.000000	26664.000000	26664.000000	
mean	-0.017777	-0.133363	-0.167679	-0.225023	-0.269764	
std	1.126769	1.198640	1.199165	1.167897	1.131735	
min	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	
25%	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	0.000000	0.000000	0.000000	
max	8.000000	8.000000	8.000000	8.000000	8.000000	

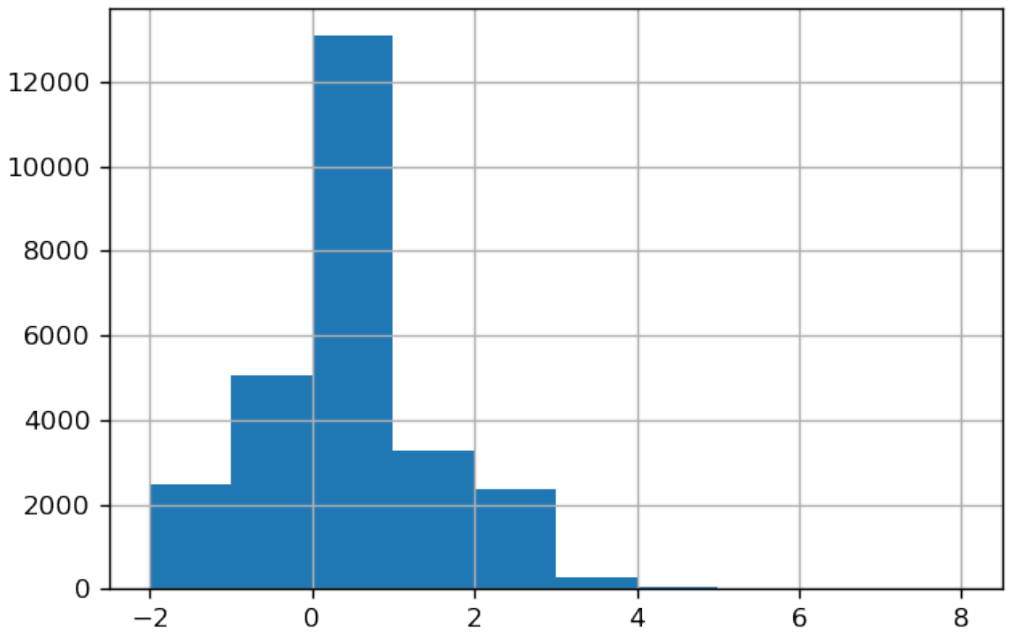
```
              PAY_6
count  26664.000000
mean      -0.293579
std        1.150229
min       -2.000000
25%       -1.000000
50%        0.000000
75%        0.000000
max        8.000000
```

[11]:
```
# PAY_1
df[pay_feats[0]].value_counts().sort_index()
```

[11]:
```
-2     2476
-1     5047
 0    13087
 1     3261
 2     2378
 3      292
 4       63
 5       23
 6       11
 7        9
 8       17
Name: PAY_1, dtype: int64
```

Representación gráfica

[22]:
```
df[pay_feats[0]].hist()
plt.show()
```



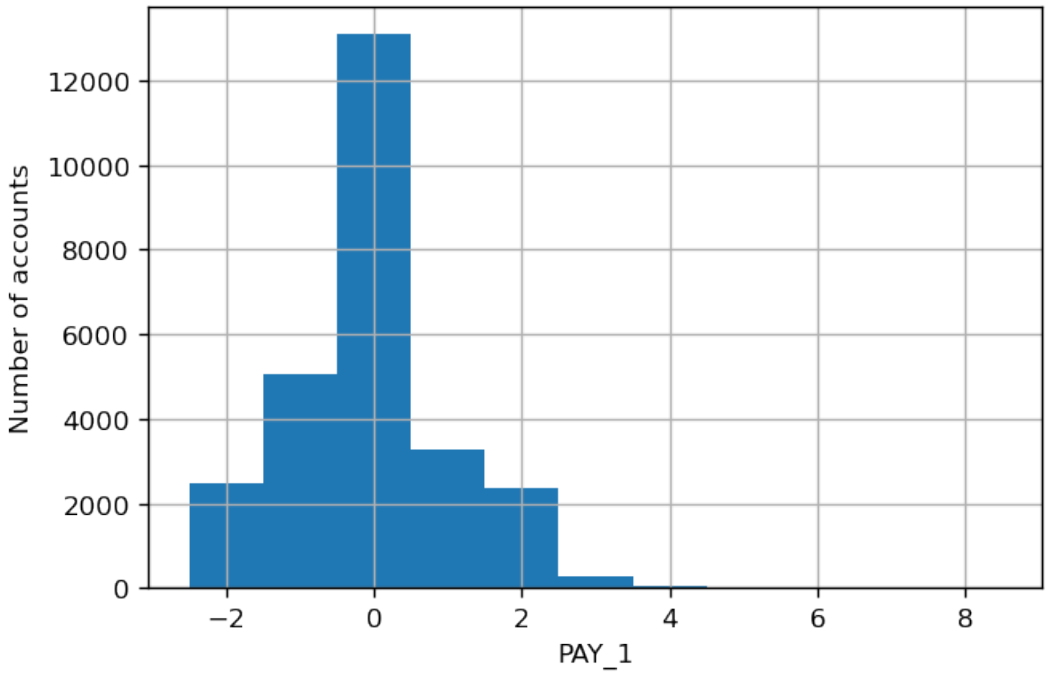
We'll create an array of 12 numbers, which will result in 11 bins, each one centered around 1 of the unique values of PAY_1

[24]:
```
pay_1_bins = np.array(range(-2,10))- 0.5
pay_1_bins
```

[24]:
```
array([-2.5, -1.5, -0.5,  0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,
        8.5])
```

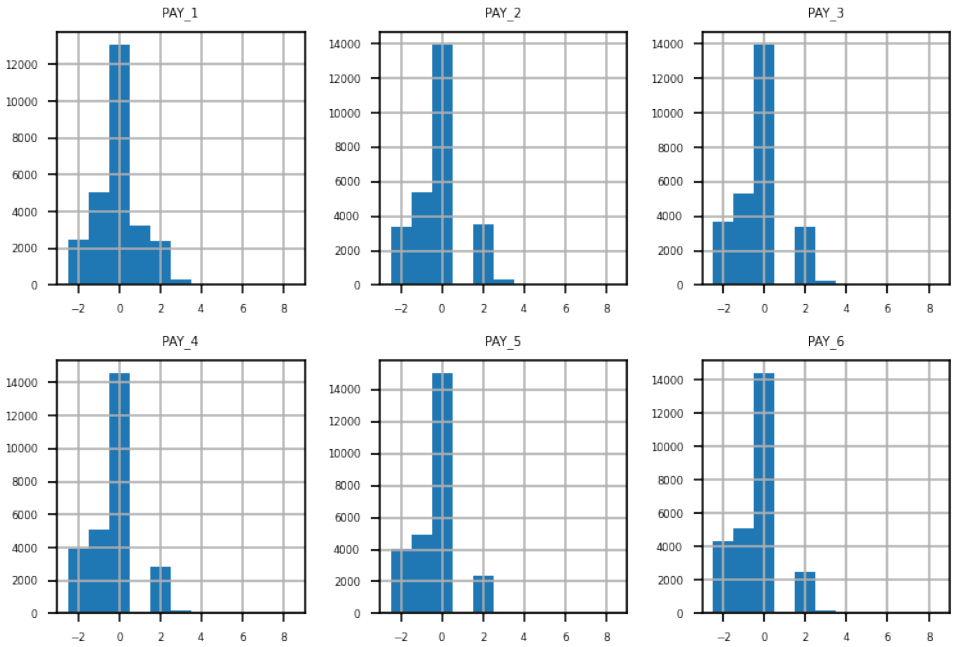
[26]:
```
# Las barras están al centro de cada número
df[pay_feats[0]].hist(bins = pay_1_bins)
plt.xlabel('PAY_1')
plt.ylabel('Number of accounts')
```

[26]:
```
Text(0, 0.5, 'Number of accounts')
```



Múltiples gráficos

[50]:
```
# Tamaño de la fuente
mpl.rcParams['font.size'] = 4
# Múltiples gráficos
pay_1_bins = np.array(range(-2,10)) - 0.5
#print(pay_1_bins)
df[pay_feats].hist(bins = pay_1_bins, layout=(2,3))
plt.show()
```



[44]:
```
df.loc[df['PAY_2']==2, ['PAY_2', 'PAY_3']].head()
```

[44]:

	PAY_2	PAY_3
0	2	-1
1	2	0
13	2	2
15	2	0
47	2	2

[ ]:
```

```