Imports

In [8]:
```
from sqlalchemy import create_engine
import pymysql
import pandas as pd
import numpy as np
import pandas_profiling
from matplotlib import pyplot as plt
```

In [9]:
```
db_connection_str = 'mysql+pymysql://deepanalytics:Sqlcask1234!@34.73.222.197/deepanalytics'
db_connection = create_engine(db_connection_str)
df = pd.read_sql('SELECT * FROM credit', con=db_connection)
```

Inspect Data

In [10]:
```
#df.to_csv("creditOne.csv") #quick eye-ball check of data
```

In [11]:
```
df.head(3)
```

Out[11]:

	MyUnknownColumn	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	X18
0	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BIL
1	1	20000	female	university	1	24	2	2	-1	-1	...	0	0	0
2	2	120000	female	university	2	26	-1	2	0	0	...	3272	3455	3261

3 rows × 25 columns

In [12]:
```
#rpt = pandas_profiling.ProfileReport(df)
#rpt.to_file(output_file="CreditOne_ProfileReport.html")
```

Let's use row 0 as the headers instead

In [13]:
```
headers = df.iloc[0] #grab first row to be new headers
df = df[1:] #entire df except first row
df.columns = headers #set new column names
df.head(3)
```

Out[13]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_
1	1	20000	female	university	1	24	2	2	-1	-1	...	0	0	0	
2	2	120000	female	university	2	26	-1	2	0	0	...	3272	3455	3261	
3	3	90000	female	university	2	34	0	0	0	0	...	14331	14948	15549	

3 rows × 25 columns

In [14]:
```
df.describe()
```

Out[14]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	BILL_AM
count	30003	30003	30003	30003	30003	30003	30003	30003	30003	30003	...	30003	30003	30003	30
unique	30002	83	4	7	6	56	12	13	13	13	...	21890	21912	20604	29
top	187	50000	female	university	2	29	0	0	0	0	...	0	0	0	
freq	2	3397	18217	14107	16088	1619	14626	15830	15963	16566	...	3218	3530	4026	4

4 rows × 25 columns

In [15]:
```
df.sort_values('AGE',ascending = False)
```

Out[15]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	BILL_AMT
202	202	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	X17	X
203	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	BILL_AMT
18445	18240	440000	male	graduate school	1	79	0	0	0	0	...	447112	438187	44754	
28345	25142	210000	male	university	1	75	0	0	0	0	...	203776	205901	21009	
488	247	250000	female	university	1	75	0	-1	-1	-1	...	1010	5572	79	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
27834	27631	10000	female	university	2	21	2	2	2	3	2	9768	8430	2073	
1426	1225	20000	male	university	2	21	0	0	-1	0	0	1300	0	0	
21788	21577	20000	female	graduate school	2	21	0	0	0	0	0	4725	9135	1835	
2416	2213	10000	male	university	2	21	0	0	0	0	0	8880	9580	909	
16832	15729	50000	female	graduate school	2	21	0	0	0	0	0	22365	12702	413	

30203 rows × 25 columns

In [16]:
```
df.shape
```

Out[16]:
```
(30203, 25)
```

Besides those top two rows, there's probably other character issues in here. Let's attack them all at once by turning them into NaNs

In [17]:
```
#numeric column list - start with all columns
numeric_columns = list(df.columns)
#make a list of columns we couldn't modify
char_cols = ['SEX','EDUCATION','default payment next month']
numeric_columns = [x for x in numeric_columns if x not in char_cols]
print(numeric_columns)
```
```
['ID', 'LIMIT_BAL', 'MARRIAGE', 'AGE', 'PAY_0', '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']
```

In [18]:
```
# coerce all numberic columns to make chars go to NaN
for x in numeric_columns:
    df[x] = pd.to_numeric(df[x], errors='coerce')
```
```
<ipython-input-18-3bd852e839cc>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[x] = pd.to_numeric(df[x], errors='coerce')
```

In [19]:
```
df.dtypes
```

Out[19]:
```
ID                          float64
LIMIT_BAL                   float64
SEX                          object
EDUCATION                    object
MARRIAGE                    float64
AGE                         float64
PAY_0                       float64
PAY_2                       float64
PAY_3                       float64
PAY_4                       float64
PAY_5                       float64
PAY_6                       float64
BILL_AMT1                   float64
BILL_AMT2                   float64
BILL_AMT3                   float64
BILL_AMT4                   float64
BILL_AMT5                   float64
BILL_AMT6                   float64
PAY_AMT1                    float64
PAY_AMT2                    float64
PAY_AMT3                    float64
PAY_AMT4                    float64
PAY_AMT5                    float64
PAY_AMT6                    float64
default payment next month   object
dtype: object
```

... it cast to float64 because NaN values still exist, which int64 can't support

In [20]:
```
#if I hadn't already caught them all, this could catch other stray chars... but shouldn't need it after conversion
#df = df.replace(/^.*?$/, np.nan)
```

Remove null values

In [21]:
```
df.shape
```

Out[21]:
```
(30203, 25)
```

In [22]:
```
df.dropna(inplace=True)
df.shape
```
```
<ipython-input-22-1e4642348ac9>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.dropna(inplace=True)
```

Out[22]:
```
(30203, 25)
```

Duplicates

In [23]:
```
df.duplicated().any()
```

Out[23]:
```
True
```

In [24]:
```
print(df[df.duplicated()].shape)
#df[df.duplicated()]
```
```
(201, 25)
```

In [25]:
```
df = df.drop_duplicates()
df.duplicated().any()
```

Out[25]:
```
False
```

In [26]:
```
df.info()
```
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30002 entries, 1 to 30002
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          30002 non-null  float64
 1   LIMIT_BAL                   30002 non-null  float64
 2   SEX                         30002 non-null  object
 3   EDUCATION                   30002 non-null  object
 4   MARRIAGE                    30002 non-null  float64
 5   AGE                         30002 non-null  float64
 6   PAY_0                       30002 non-null  float64
 7   PAY_2                       30002 non-null  float64
 8   PAY_3                       30002 non-null  float64
 9   PAY_4                       30002 non-null  float64
 10  PAY_5                       30002 non-null  float64
 11  PAY_6                       30002 non-null  float64
 12  BILL_AMT1                   30002 non-null  float64
 13  BILL_AMT2                   30002 non-null  float64
 14  BILL_AMT3                   30002 non-null  float64
 15  BILL_AMT4                   30002 non-null  float64
 16  BILL_AMT5                   30002 non-null  float64
 17  BILL_AMT6                   30002 non-null  float64
 18  PAY_AMT1                    30002 non-null  float64
 19  PAY_AMT2                    30002 non-null  float64
 20  PAY_AMT3                    30002 non-null  float64
 21  PAY_AMT4                    30002 non-null  float64
 22  PAY_AMT5                    30002 non-null  float64
 23  PAY_AMT6                    30002 non-null  float64
 24  default payment next month  30002 non-null  object
dtypes: float64(23), object(2)
memory usage: 6.0+ MB
```

Auto-adjust data types using export/import

In [27]:
```
#df.to_csv('creditOne_cleaned.csv') #export for re-import to fix data types
```

In [28]:
```
#df = pd.read_csv('creditOne_cleaned.csv', index_col=0)
#print(df.shape)
```

In [29]:
```
# I don't like how since we removed the first row, the index now starts at 1... fix it
df.reset_index(drop=True, inplace=True)
df.head()
```

Out[29]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_
0	1.0	20000.0	female	university	1.0	24.0	2.0	2.0	-1.0	-1.0	...	0.0	0.0	0.0	
1	2.0	120000.0	female	university	2.0	26.0	-1.0	2.0	0.0	0.0	...	3272.0	3455.0	3261.0	
2	3.0	90000.0	female	university	2.0	34.0	0.0	0.0	0.0	0.0	...	14331.0	14948.0	15549.0	
3	4.0	50000.0	female	university	1.0	37.0	0.0	0.0	0.0	0.0	...	28314.0	28959.0	29547.0	
4	5.0	50000.0	male	university	1.0	57.0	-1.0	0.0	-1.0	0.0	...	20940.0	19146.0	19131.0	

5 rows × 25 columns

Hmmm... everything is still a float64. Let's try to force 'int64'

In [30]:
```
#quick little function to make a list out of the numeric columns ('col1':'int64', ...)
def Convert_List(set_type):
    res_dct = {lst[i]: type for i in range(0, len(lst), 1)}
    return res_dct

convert_dict = Convert_List(numeric_columns, 'int64')
print(convert_dict)
```
```
{'ID': 'int64', 'LIMIT_BAL': 'int64', 'MARRIAGE': 'int64', 'AGE': 'int64', 'PAY_0': 'int64', 'PAY_2': 'int64', 'PAY_3': 'int64', 'PAY_4': 'int64', 'PAY_5': 'int64', 'PAY_6': 'int64', 'BILL_AMT1': 'int64', 'BILL_AMT2': 'int64', 'BILL_AMT3': 'int64', 'BILL_AMT4': 'int64', 'BILL_AMT5': 'int64', 'BILL_AMT6': 'int64', 'PAY_AMT1': 'int64', 'PAY_AMT2': 'int64', 'PAY_AMT3': 'int64', 'PAY_AMT4': 'int64', 'PAY_AMT5': 'int64', 'PAY_AMT6': 'int64'}
```

If you don't assign to df2 then it doesn't actually save it. Inplace is not an option

In [31]:
```
df2 = df.astype(convert_dict)
```

In [32]:
```
df2.rename(columns={"default payment next month":"DEFAULT"}, inplace=True)
```

In [33]:
```
df2.dtypes
```

Out[33]:
```
ID                  int64
LIMIT_BAL           int64
SEX                object
EDUCATION          object
MARRIAGE            int64
AGE                 int64
PAY_0               int64
PAY_2               int64
PAY_3               int64
PAY_4               int64
PAY_5               int64
PAY_6               int64
BILL_AMT1           int64
BILL_AMT2           int64
BILL_AMT3           int64
BILL_AMT4           int64
BILL_AMT5           int64
BILL_AMT6           int64
PAY_AMT1            int64
PAY_AMT2            int64
PAY_AMT3            int64
PAY_AMT4            int64
PAY_AMT5            int64
PAY_AMT6            int64
DEFAULT            object
dtype: object
```

In [34]:
```
df2.to_csv('creditOne_cleaned_formatted.csv') #export for re-import in next round
#don't forget to import with index_col=0
    #df = pd.read_csv('creditOne_cleaned_formatted.csv', index_col=0)
        #check for format type and your index (and that no unnamed columns exist)
```

DATA IS CLEANED!

In [35]:
```
df2.groupby(['DEFAULT']).count()
```

Out[35]:

DEFAULT	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT5	BILL_AMT6	BILL_
default	6636	6636	6636	6636	6636	6636	6636	6636	6636	6636	...	6636	6636	
not default	23364	23364	23364	23364	23364	23364	23364	23364	23364	23364	...	23364	23364	2

2 rows × 24 columns

In [36]:
```
df2.groupby(['SEX']).count()
```

Out[36]:

SEX	ID	LIMIT_BAL	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT5	BILL_AMT6	BILL_AM
female	18112	18112	18112	18112	18112	18112	18112	18112	18112	18112	...	18112	18112	181
male	11890	11890	11890	11890	11890	11890	11890	11890	11890	11890	...	11890	11890	118

2 rows × 24 columns

In [37]:
```
df2.groupby('AGE')['DEFAULT'].count()
df2.groupby('AGE')['DEFAULT'].value_counts().unstack(level=1).plot.bar(stacked=True, figsize=(9,5))
```

Out[37]:
```
<AxesSubplot:xlabel='AGE'>
```


In [41]:
```
df2.groupby('AGE')['DEFAULT'].value_counts().unstack(level=1).plot.barh(stacked=True, figsize=(9,5))
```

Out[41]:
```
<AxesSubplot:xlabel='AGE'>
```


In [ ]:
```

```