In [1]:

```
#Imports
import pandas as pd
import numpy as np

#Import visualization libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:

```
df = pd.read_excel('credit_card_defaults.xls')
df.head(4)
```

Out[2]:

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...
0	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...
1	1	20000	2	2	1	24	2	2	-1	-1	...
2	2	120000	2	2	2	26	-1	2	0	0	...
3	3	90000	2	2	2	34	0	0	0	0	...

4 rows × 25 columns

As we can see from the above cell, the columns of the dataframe are not properly labelled. We will have to do a little bit of cleaning, one step at a time.

In [3]:

```
df.columns
```

Out[3]:

```
Index(['Unnamed: 0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9',
       'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
       'X20', 'X21', 'X22', 'X23', 'Y'],
      dtype='object')
```

In [4]:

```
df.iloc[0].head(6)
```

Out[4]:

```
Unnamed: 0           ID
X1            LIMIT_BAL
X2                  SEX
X3            EDUCATION
X4             MARRIAGE
X5                  AGE
Name: 0, dtype: object
```

We'll remove the present columns and replace them with the values of the first row in the dataframe

In [5]:

```
df.columns = df.iloc[0]
df.columns
```

Out[5]:

```
Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT
2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'default payment next month'],
      dtype='object', name=0)
```

In [6]:

```
df.head(4)
```

Out[6]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BIL
0	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BIL
1	1	20000	2	2	1	24	2	2	-1	-1	...	
2	2	120000	2	2	2	26	-1	2	0	0	...	
3	3	90000	2	2	2	34	0	0	0	0	...	

4 rows × 25 columns

As we've successfully changed the names of the columns, we'll drop the first row of the dataframe.

In [7]:

```
df.describe().T
```

Out[7]:

	count	unique	top	freq
0				
ID	30001	30001	6800	1
LIMIT_BAL	30001	82	50000	3365
SEX	30001	3	2	18112
EDUCATION	30001	8	2	14030
MARRIAGE	30001	5	2	15964
AGE	30001	57	29	1605
PAY_0	30001	12	0	14737
PAY_2	30001	12	0	15730
PAY_3	30001	12	0	15764
PAY_4	30001	12	0	16455
PAY_5	30001	11	0	16947
PAY_6	30001	11	0	16286
BILL_AMT1	30001	22724	0	2008
BILL_AMT2	30001	22347	0	2506
BILL_AMT3	30001	22027	0	2870
BILL_AMT4	30001	21549	0	3195
BILL_AMT5	30001	21011	0	3506
BILL_AMT6	30001	20605	0	4020
PAY_AMT1	30001	7944	0	5249
PAY_AMT2	30001	7900	0	5396
PAY_AMT3	30001	7519	0	5968
PAY_AMT4	30001	6938	0	6408
PAY_AMT5	30001	6898	0	6703
PAY_AMT6	30001	6940	0	7173
default payment next month	30001	3	0	23364

In [8]:

```
df = df.drop(index=0)
```

In [9]:

```
assert(df.index == df['ID']).all(), \
'The index is not the same as the ID column'
```

In [10]:

```
# Since the ID columns is the same as the index, well drop it as it won't be useful in
df.drop(labels=['ID'], axis=1, inplace=True)
```

In [11]:

```
df.columns = df.columns.str.lower()
df.columns
```

Out[11]:

```
Index(['limit_bal', 'sex', 'education', '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',
       'default payment next month'],
      dtype='object', name=0)
```

In [12]:

```
def convert(data):
    for col in data.columns:
        data[col] = data[col].astype('int64')
    return data
df = convert(df)
```

In [13]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30000 entries, 1 to 30000
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   limit_bal                   30000 non-null  int64
 1   sex                         30000 non-null  int64
 2   education                   30000 non-null  int64
 3   marriage                    30000 non-null  int64
 4   age                         30000 non-null  int64
 5   pay_0                       30000 non-null  int64
 6   pay_2                       30000 non-null  int64
 7   pay_3                       30000 non-null  int64
 8   pay_4                       30000 non-null  int64
 9   pay_5                       30000 non-null  int64
 10  pay_6                       30000 non-null  int64
 11  bill_amt1                   30000 non-null  int64
 12  bill_amt2                   30000 non-null  int64
 13  bill_amt3                   30000 non-null  int64
 14  bill_amt4                   30000 non-null  int64
 15  bill_amt5                   30000 non-null  int64
 16  bill_amt6                   30000 non-null  int64
 17  pay_amt1                    30000 non-null  int64
 18  pay_amt2                    30000 non-null  int64
 19  pay_amt3                    30000 non-null  int64
 20  pay_amt4                    30000 non-null  int64
 21  pay_amt5                    30000 non-null  int64
 22  pay_amt6                    30000 non-null  int64
 23  default payment next month  30000 non-null  int64
dtypes: int64(24)
memory usage: 5.7 MB
```

In [14]:

```
df.describe().T
```

Out[14]:

	count	mean	std	min	25%	50%	75%	
0								
limit_bal	30000.0	167484.322667	129747.661567	10000.0	50000.00	140000.0	240000.00	10
sex	30000.0	1.603733	0.489129	1.0	1.00	2.0	2.00	
education	30000.0	1.853133	0.790349	0.0	1.00	2.0	2.00	
marriage	30000.0	1.551867	0.521970	0.0	1.00	2.0	2.00	
age	30000.0	35.485500	9.217904	21.0	28.00	34.0	41.00	
pay_0	30000.0	-0.016700	1.123802	-2.0	-1.00	0.0	0.00	
pay_2	30000.0	-0.133767	1.197186	-2.0	-1.00	0.0	0.00	
pay_3	30000.0	-0.166200	1.196868	-2.0	-1.00	0.0	0.00	
pay_4	30000.0	-0.220667	1.169139	-2.0	-1.00	0.0	0.00	
pay_5	30000.0	-0.266200	1.133187	-2.0	-1.00	0.0	0.00	
pay_6	30000.0	-0.291100	1.149988	-2.0	-1.00	0.0	0.00	
bill_amt1	30000.0	51223.330900	73635.860576	-165580.0	3558.75	22381.5	67091.00	9
bill_amt2	30000.0	49179.075167	71173.768783	-69777.0	2984.75	21200.0	64006.25	9
bill_amt3	30000.0	47013.154800	69349.387427	-157264.0	2666.25	20088.5	60164.75	16
bill_amt4	30000.0	43262.948967	64332.856134	-170000.0	2326.75	19052.0	54506.00	8
bill_amt5	30000.0	40311.400967	60797.155770	-81334.0	1763.00	18104.5	50190.50	9
bill_amt6	30000.0	38871.760400	59554.107537	-339603.0	1256.00	17071.0	49198.25	9
pay_amt1	30000.0	5663.580500	16563.280354	0.0	1000.00	2100.0	5006.00	8
pay_amt2	30000.0	5921.163500	23040.870402	0.0	833.00	2009.0	5000.00	16
pay_amt3	30000.0	5225.681500	17606.961470	0.0	390.00	1800.0	4505.00	8
pay_amt4	30000.0	4826.076867	15666.159744	0.0	296.00	1500.0	4013.25	6
pay_amt5	30000.0	4799.387633	15278.305679	0.0	252.50	1500.0	4031.50	4
pay_amt6	30000.0	5215.502567	17777.465775	0.0	117.75	1500.0	4000.00	5
default payment next month	30000.0	0.221200	0.415062	0.0	0.00	0.0	0.00	

In [15]:

```
df.isnull().sum().sum()
```

Out[15]:

```
0
```

DATA PREPROCESSING

Let's look at the unique values in the columns. The motive behind looking at unique values in a column is to identify the subcategory in each column.

In [16]:

```
df.columns = df.columns.str.lower()
df.columns
```

Out[16]:

```
Index(['limit_bal', 'sex', 'education', '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',
       'default payment next month'],
      dtype='object', name=0)
```

In [17]:

```
#Finding unique values in the SEX column
print('Sex ' + str(sorted(df['sex'].unique()[1:])))
```

```
Sex [1]
```

In [18]:

```
#Findning unique values in the EDUCATION column
print('Education ' + str(sorted(df['education'].unique())))
```

```
Education [0, 1, 2, 3, 4, 5, 6]
```

In [19]:

```
#Finding the unqiue values in the MARRIAGE column
print('Marriage ' + str(sorted(df['marriage'].unique())))
```

```
Marriage [0, 1, 2, 3]
```

In [20]:

```
#Unique values from the Pay_0 column
print('Pay_0 ' + str(sorted(df['pay_0'].unique())))
```

```
Pay_0 [-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8]
```

In [21]:

```
print('default.payment.next.month ' \
+ str(sorted(df['default payment next month'].unique())))
```

default.payment.next.month [0, 1]

In [22]:

```
"""The EDUCATION column has 7 unique values, but as per our data description,
   we have only 4 unique values, so we are going to club categories 0, 5, and 6 with ca
fill = (df.education==0) | (df.education==5) | (df.education==6)
df.loc[fill, 'education'] = 4
print('Education ' + str(sorted(df['education'].unique())))
```

Education [1, 2, 3, 4]

In [23]:

```
df.head(4)
```

Out[23]:

	limit_bal	sex	education	marriage	age	pay_0	pay_2	pay_3	pay_4	pay_5	...	bill_amt4
1	20000	2	2	1	24	2	2	-1	-1	-2	...	0
2	120000	2	2	2	26	-1	2	0	0	0	...	3272
3	90000	2	2	2	34	0	0	0	0	0	...	14331
4	50000	2	2	1	37	0	0	0	0	0	...	28314

4 rows × 24 columns

In [24]:

```
"""Similarly, in the MARRIAGE column, according to the data description, we should
have 3 unique values. But here, we have 4 values in our data. As per our data
description, the MARRIAGE column should have three subcategories. So, we combine
category 0 with category 2 (Single):"""
fill = (df.marriage == 0)
df.loc[fill, 'marriage'] = 2
print('Marriage ' + str(sorted(df['marriage'].unique())))
```

Marriage [1, 2, 3]

In [25]:

```
df = df.rename(mapper={'default payment next month':'default', 'pay_0':'pay_1'}, axis=1
df.columns
```

Out[25]:

```
Index(['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', 'defaul
t'],
      dtype='object', name=0)
```

Exploratory Data Analysis

Univariate Analysis

Univariate analysis is the simplest form of analysis where we analyze each feature (that is, each column of a DataFrame) and try to uncover the pattern or distribution of the data. We'll be analyzing categorical features(default, sex, education and marriage.)

In [26]:

```
df.head(3)
```

Out[26]:

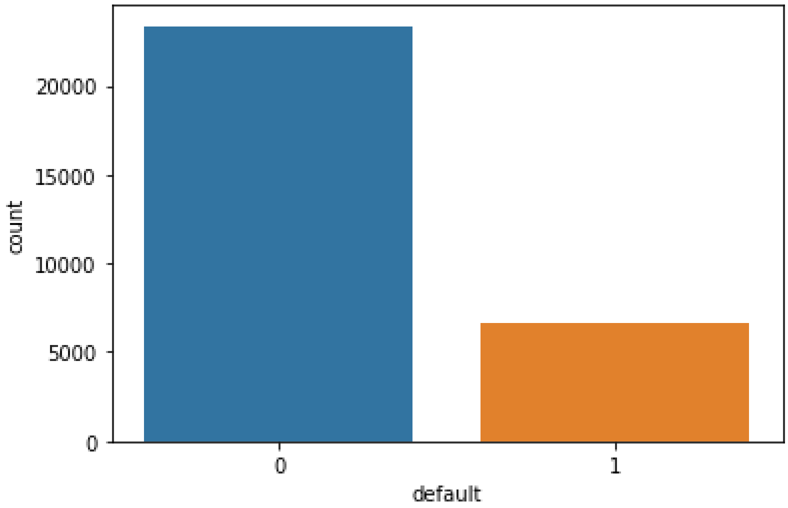
	limit_bal	sex	education	marriage	age	pay_1	pay_2	pay_3	pay_4	pay_5	...	bill_amt4
1	20000	2	2	1	24	2	2	-1	-1	-2	...	0
2	120000	2	2	2	26	-1	2	0	0	0	...	3272
3	90000	2	2	2	34	0	0	0	0	0	...	14331

3 rows × 24 columns

Let's begin with each of the variables one by one

In [27]:

```
#The default column
sns.countplot(x='default', data=df)
plt.figure(figsize=(9,5));
```



<Figure size 648x360 with 0 Axes>

In [28]:

```
df['default'].value_counts()
```

Out[28]:

```
0    23364
1     6636
Name: default, dtype: int64
```

In [29]:

```
(df['default'].value_counts() * 100)/30000
```

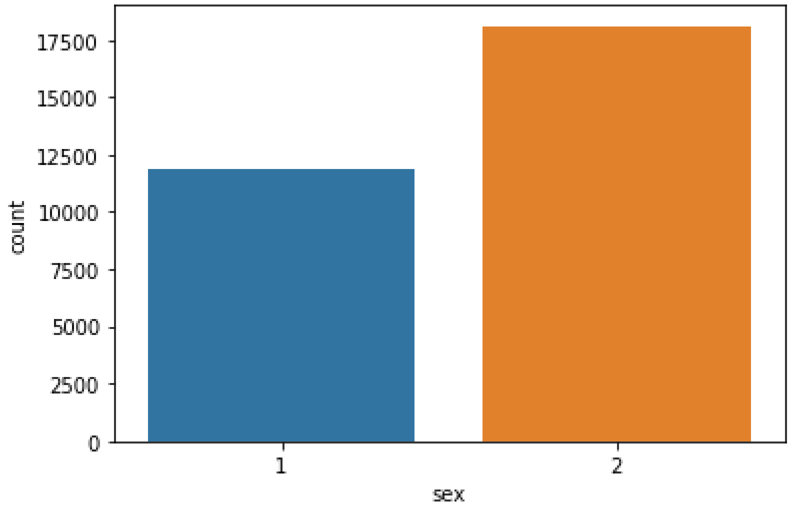
Out[29]:

```
0    77.88
1    22.12
Name: default, dtype: float64
```

From the preceding output, we see that around *6636* customers have defaulted out of *30,000* people, which is around *22%*.

In [30]:

```
#The sex column
sns.countplot(x='sex', data=df);
```



In the preceding output, 1 represents male and 2 represents female.

In [31]:

```
df.sex.value_counts()
```

Out[31]:

```
2    18112
1    11888
Name: sex, dtype: int64
```

In [32]:

```
(df.sex.value_counts()*100)/30000
```

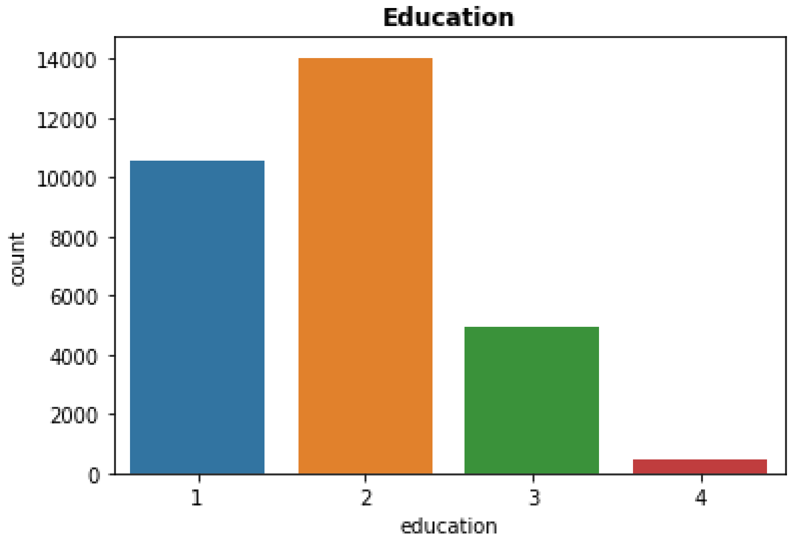
Out[32]:

```
2    60.373333
1    39.626667
Name: sex, dtype: float64
```

From the preceding output, there are 18112 female(about 60%) and 11888 male(approximately 40%) in the dataset.

In [33]:

```
#The education column
sns.countplot(x='education', data=df)
plt.title("Education", weight='bold');
```



From the preceding output, 1=Graduate School, 2=University, 3=High School, 4=Others which represents the highest qualification of customers.

In [34]:

```
df.education.value_counts()
```

Out[34]:

```
2    14030
1    10585
3     4917
4      468
Name: education, dtype: int64
```

In [35]:

```
(df.education.value_counts()*100)/30000
```

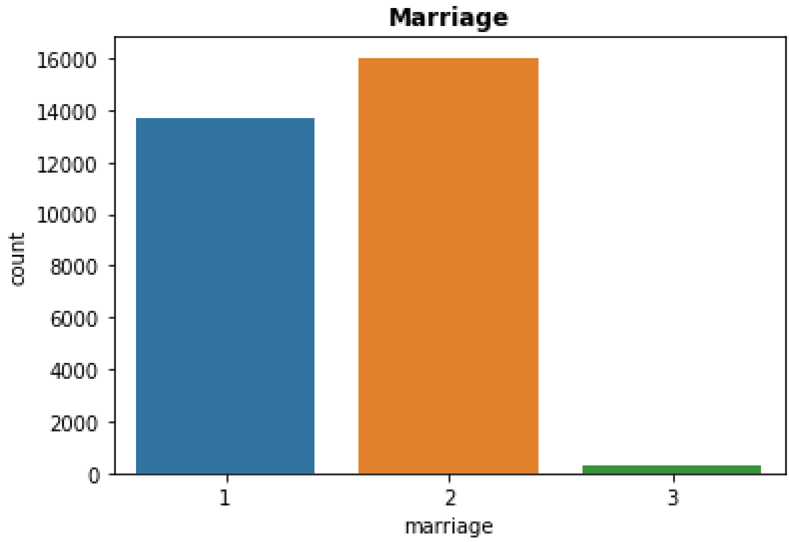
Out[35]:

```
2    46.766667
1    35.283333
3    16.390000
4     1.560000
Name: education, dtype: float64
```

From the preceding outputs, most of the customers either went to University or Graduate school.

In [36]:

```
#The marriage column
sns.countplot(x='marriage', data=df)
plt.title("Marriage", weight='bold');
```



In the preceding output, 1=Married, 2=Single, 3=Divorced. These represent the marital status of customers.

In [37]:

```
df.marriage.value_counts()
```

Out[37]:

```
2    16018
1    13659
3      323
Name: marriage, dtype: int64
```

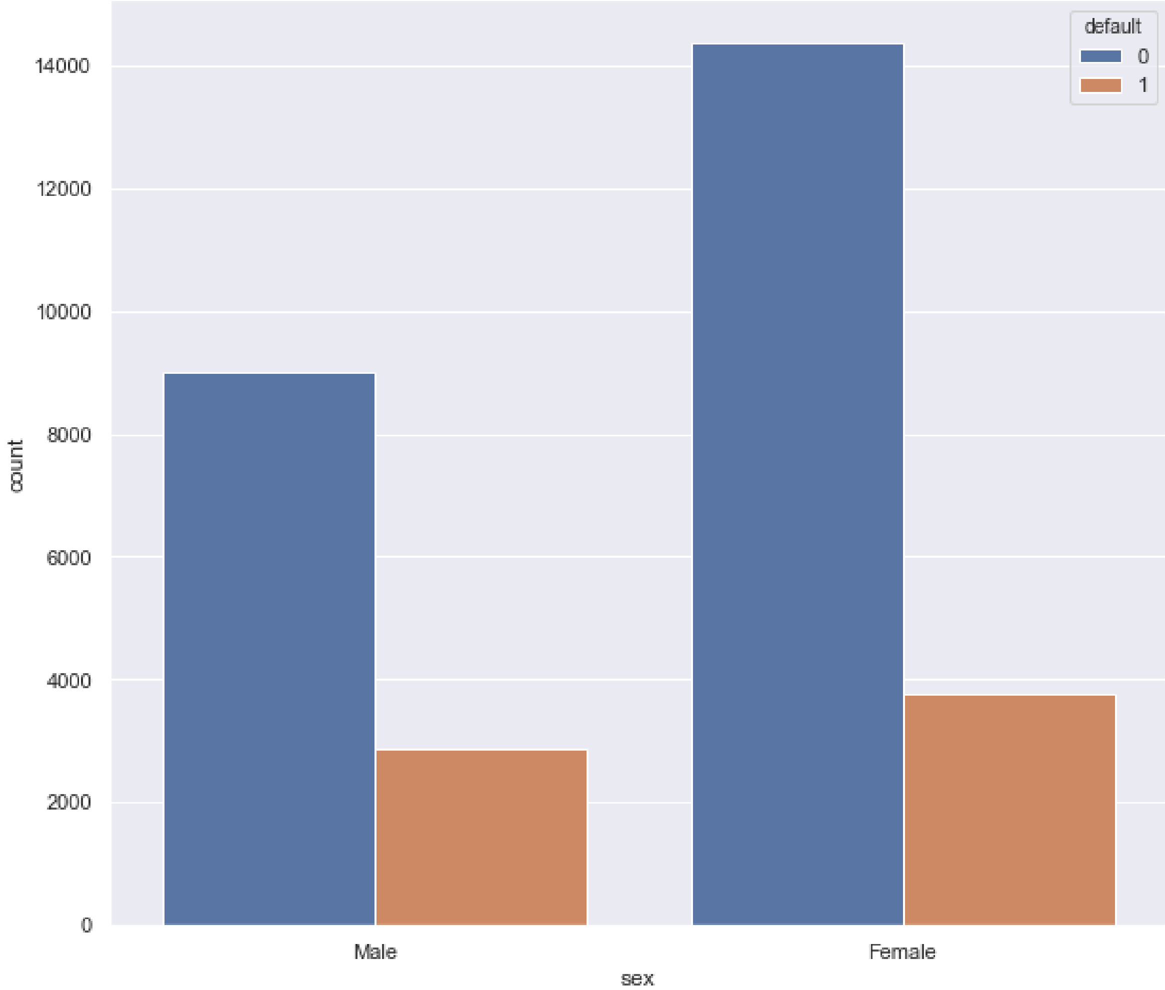
From the preceding output, most of the customers are Single and Married people.

Bivariate Analysis

Bivariate analysis is performed between two variables to look at their relationship.

In [38]:

```
""" We'll be seeing the relationship between the sex and default column to know compare
    the number of male customers who have defaulted to the number of female customers t
    have also defaulted."""
sns.set(rc={'figure.figsize':(10,9)})
edu = sns.countplot(x='sex', hue='default', data=df)
edu.set_xticklabels(['Male','Female'])
plt.show()
```



From the preceding output, we can see that more females have defaulted compared to the males, this graph does not give us the complete picture as there are more female customers than male customers.

In [39]:

```
pd.crosstab(df.sex, df.default, margins=True)
```

Out[39]:

default sex	0	1	All
1	9015	2873	11888
2	14349	3763	18112
All	23364	6636	30000

In [40]:

```
pd.crosstab(df.sex, df.default, normalize='index', margins=True)
```

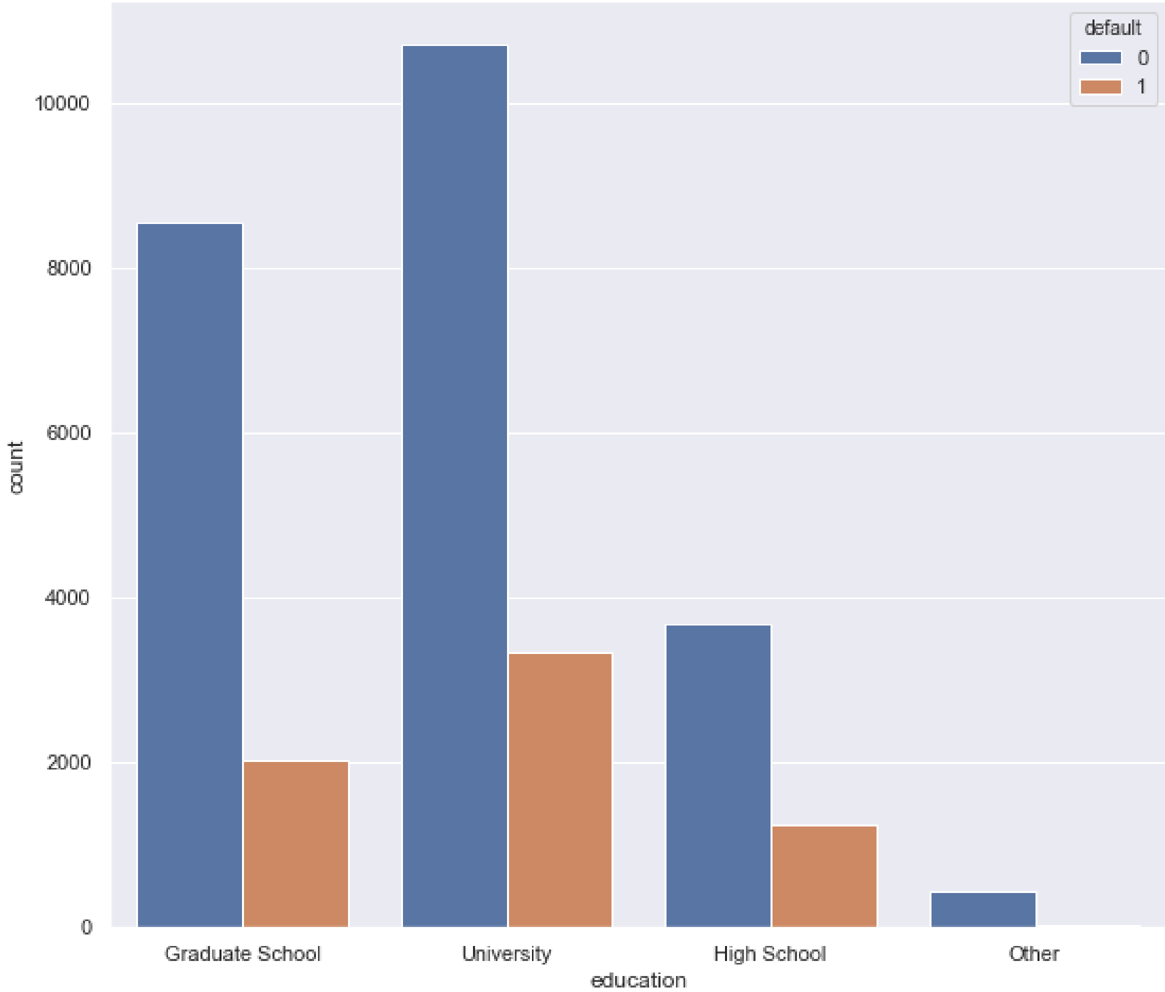
Out[40]:

default sex	0	1
1	0.758328	0.241672
2	0.792237	0.207763
All	0.778800	0.221200

The preceding output tells us that 24% of male customers have defaulted while 20% of female customers have defaulted.

In [41]:

```
# The education and default column
edu = sns.countplot(x='education', hue='default', data=df)
edu.set_xticklabels(['Graduate School','University', 'High School','Other'])
plt.show()
```



From the preceding output, the customers that went to University have defaulted the most followed by the ones who attended Graduate school. This graph does not tell the whole story, therefore we'll need to check the actual values and percentage.

In [42]:

```
pd.crosstab(df.education, df.default, margins=True)
```

Out[42]:

default education	0	1	All
1	8549	2036	10585
2	10700	3330	14030
3	3680	1237	4917
4	435	33	468
All	23364	6636	30000

In [43]:

```
pd.crosstab(df.education, df.default, normalize='index', margins=True)
```

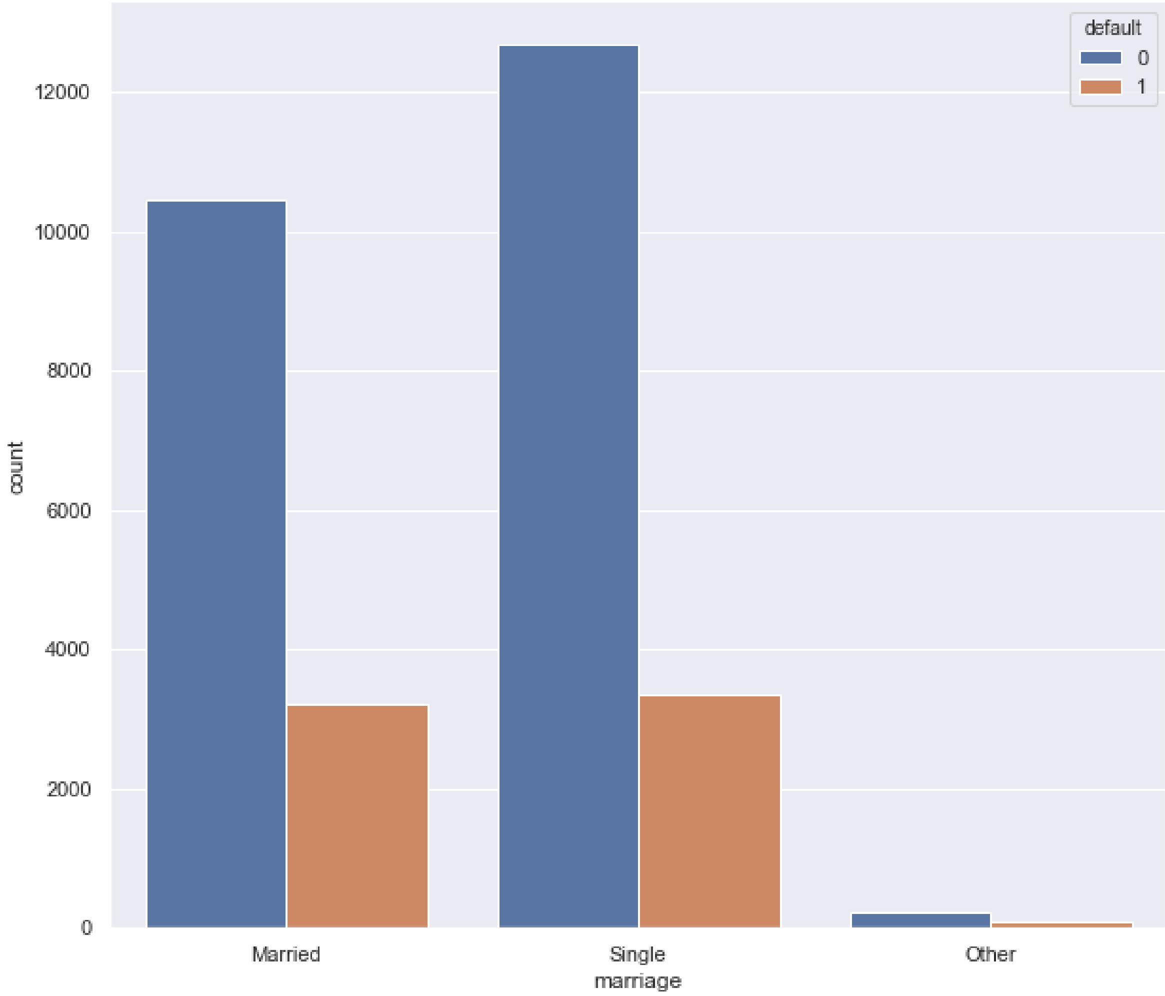
Out[43]:

default education	0	1
1	0.807652	0.192348
2	0.762651	0.237349
3	0.748424	0.251576
4	0.929487	0.070513
All	0.778800	0.221200

From the preceding output, *25%* of High school customers defaulted, 23% of University customers defaulted.

In [44]:

```
# The marriage and default column.
marriage = sns.countplot(x='marriage', hue='default', data=df)
marriage.set_xticklabels(['Married','Single','Other'])
plt.show()
```



In [45]:

```
pd.crosstab(df.marriage, df.default, margins=True)
```

Out[45]:

default	0	1	All
marriage			
1	10453	3206	13659
2	12672	3346	16018
3	239	84	323
All	23364	6636	30000

In [46]:

```
pd.crosstab(df.marriage, df.default, normalize='index', margins=True)
```

Out[46]:

default	0	1
marriage		
1	0.765283	0.234717
2	0.791110	0.208890
3	0.739938	0.260062
All	0.778800	0.221200

From the preceding output, most defaulters are divorced(26%), followed by the married(23%) people.

In [47]:

```
df.pay_1.values
```

Out[47]:

```
array([ 2, -1,  0, ...,  4,  1,  0], dtype=int64)
```

The Pay_1(the repayment status in the month of September 2005) and default column. For the pay_1 column, we won't be mapping the values as it won't be visible on the dataframe. -1=Paid on time, 1=Payment delay for 1 month, 2=Payment delay for 2 months thorough 8. 9 is for 9 months and above.

In [48]:

```
pd.crosstab(df.pay_1, df.default, margins=True)
```

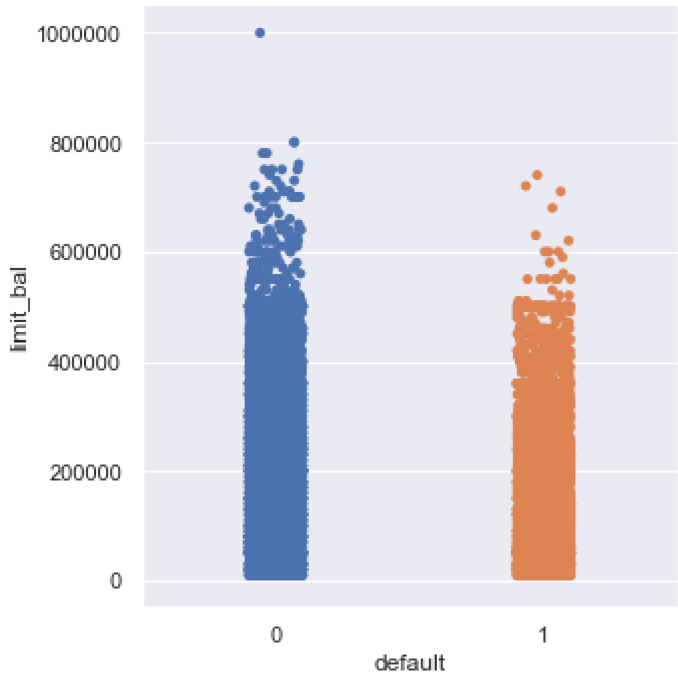
Out[48]:

default	0	1	All
pay_1			
-2	2394	365	2759
-1	4732	954	5686
0	12849	1888	14737
1	2436	1252	3688
2	823	1844	2667
3	78	244	322
4	24	52	76
5	13	13	26
6	5	6	11
7	2	7	9
8	8	11	19
All	23364	6636	30000

From the output of the crosstab function, we can see that the maximum count of defaults falls under subcategory 2—that is, a payment delay for the last 2 months. This implies that a customer who has missed payments for 2 continuous months has a high probability of default.

In [49]:

```
# Limit_Balance and default column
"""The balance is the amount given as credit. It includes both the individual consumer
credit and their family's (supplementary) credit."""
sns.set(rc={'figure.figsize':(15,15)})
sns.catplot(x='default', y='limit_bal', jitter=True, data=df);
```



From the above plot, we can infer that customers with higher balances have lower likelihood of default than those with lower balance amounts.

In [50]:

```
pd.crosstab(df.age, df.default).head(10)
```

Out[50]:

default	0	1
age		
21	53	14
22	391	169
23	684	247
24	827	300
25	884	302
26	1003	253
27	1164	313
28	1123	286
29	1292	313
30	1121	274

As we can see from the preceding output, age 27 and 29 have the highest defaults.

In [51]:

```
pd.crosstab(df.age, df.default, normalize='index', margins=True).head(10)
```

Out[51]:

default	0	1
age		
21	0.791045	0.208955
22	0.698214	0.301786
23	0.734694	0.265306
24	0.733807	0.266193
25	0.745363	0.254637
26	0.798567	0.201433
27	0.788084	0.211916
28	0.797019	0.202981
29	0.804984	0.195016
30	0.803584	0.196416

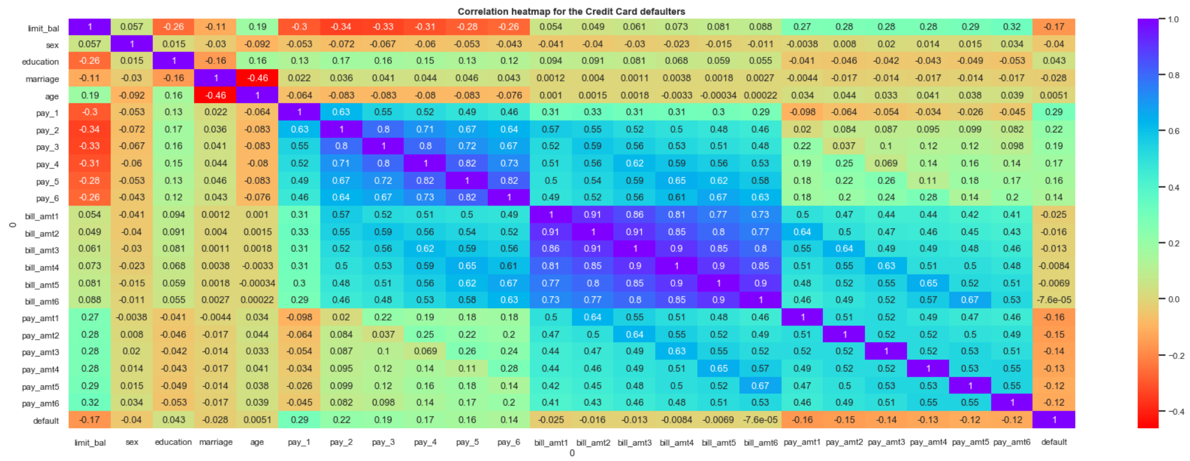
From the preceding output, we can see that even though the ages 27 and 29 had higher counts of defaults, the percentage-wise default count paints a different picture. Those customers of the age of 22 had a higher percentage of defaulters than non-defaulters.

In [52]:

```
sns.set(rc={'figure.figsize':(30,10)})
sns.set_context("talk", font_scale=0.7)
```

In [53]:

```
sns.heatmap(df.corr(method='spearman'), \
                cmap='rainbow_r', annot=True)
plt.title("Correlation heatmap for the Credit Card defaulters", weight="bold");
```



In [54]:

```
df['age'].corr(df['default'], method='spearman')
```

Out[54]:

```
0.005148863519844661
```

In [55]:

```
def correlation(data):
    for col in data.columns:
        print(col +"  " + str(data[col].corr(data['default'],method='spearman')))
correlation(df)
```

```
limit_bal  -0.16958627777128973
sex  -0.0399605777054416
education  0.0434254664517991
marriage  -0.028174099838732487
age  0.005148863519844661
pay_1  0.2922132153119903
pay_2  0.21691875073932657
pay_3  0.19477122842037808
pay_4  0.17368952787447228
pay_5  0.15904328252424124
pay_6  0.1425232152732447
bill_amt1  -0.025326827533909278
bill_amt2  -0.01555375617850131
bill_amt3  -0.012669908803039634
bill_amt4  -0.008357064590967862
bill_amt5  -0.00685122651830854
bill_amt6  -7.612488787045955e-05
pay_amt1  -0.16049312738844118
pay_amt2  -0.1509773960410106
pay_amt3  -0.13938802702711522
pay_amt4  -0.12797859795409017
pay_amt5  -0.11658708671179431
pay_amt6  -0.12144363905532163
default  1.0
```

From the preceding output, we can easily conclude that the DEFAULT column has a high positive correlation with PAY_1 (.29), which implies that if a customer has missed a payment in the first month, they have a higher chance of missing further payments in the consecutive months. Also, the DEFAULT column has the highest negative correlation with PAY_AMT1 (-.16), which implies that the higher the payment for the month of September 2005, the lower the chances of default.

Building A High Risk Profile

As a result of the analysis performed on the dataset, we can now build a profile of the customer who is most likely to default. With this predicted customer profile, credit card companies can take preventive steps (such as reducing credit limits or increasing the rate of interest) and can demand additional collateral from customers who are deemed to be high risk.

The customer who satisfies the majority of the following conditions can be classified as a high-risk customer. A high-risk customer is one who has a higher probability of default:

• A male customer is more likely to default than a female customer.
• People with a relationship status of other are more likely to default than married or single people.
• A customer whose highest educational qualification is a high-school diploma is more likely to default than a customer who has gone to graduate school or university.
• A customer who has delayed payment for 2 consecutive months has a higher probability of default.
• A customer who is 22 years of age has a higher probability of defaulting on payments than any other age group

In [ ]:

```
1
```