Multiple Logistic Regression

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

#input balance and income (Float)
#output default (Categorical data)
```

In [2]:
```
tbl=pd.read_excel("Default1.xlsx")
tbl
```

Out[2]:

	Unnamed: 0	default	student	balance	income
0	1	No	Yes	729.526495	44361.625074
1	2	No	Yes	817.180407	12106.134700
2	3	No	No	1073.549164	31767.138947
3	4	No	No	529.250605	35704.493935
4	5	No	No	785.655883	38463.495879
...	...	...	...	...	...
9995	9996	No	No	711.555020	52992.378914
9996	9997	No	No	757.962918	19660.721768
9997	9998	No	No	845.411989	58636.156984
9998	9999	No	No	1569.009053	36669.112365
9999	10000	No	Yes	200.922183	16862.952321

10000 rows × 5 columns

In [3]:
```
tbl.head()
```

Out[3]:

	Unnamed: 0	default	student	balance	income
0	1	No	Yes	729.526495	44361.625074
1	2	No	Yes	817.180407	12106.134700
2	3	No	No	1073.549164	31767.138947
3	4	No	No	529.250605	35704.493935
4	5	No	No	785.655883	38463.495879

In [4]:
```
tbl.shape
```

Out[4]:
```
(10000, 5)
```

In [5]:
```
tbl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  10000 non-null  int64  
 1   default     10000 non-null  object 
 2   student     10000 non-null  object 
 3   balance     10000 non-null  float64
 4   income      10000 non-null  float64
dtypes: float64(2), int64(1), object(2)
memory usage: 390.8+ KB
```

In [6]:
```
tbl.describe()
```

Out[6]:

	Unnamed: 0	balance	income
count	10000.00000	10000.000000	10000.000000
mean	5000.50000	835.374886	33516.981876
std	2886.89568	483.714985	13336.639563
min	1.00000	0.000000	771.967729
25%	2500.75000	481.731105	21340.462903
50%	5000.50000	823.636973	34552.644802
75%	7500.25000	1166.308386	43807.729272
max	10000.00000	2654.322576	73554.233495

In [7]:
```
tbl.isnull().sum
```

Out[7]:
```
<bound method NDFrame._add_numeric_operations.<locals>.sum of       Unname
d: 0  default  student  balance  income
0          False    False    False    False   False
1          False    False    False    False   False
2          False    False    False    False   False
3          False    False    False    False   False
4          False    False    False    False   False
...          ...      ...      ...      ...     ...
9995       False    False    False    False   False
9996       False    False    False    False   False
9997       False    False    False    False   False
9998       False    False    False    False   False
9999       False    False    False    False   False

[10000 rows x 5 columns]>
```

Statistical Analysis

In [8]:
```
#unique => default value 2 that means Yes or No

tbl.describe(include='all')
```

Out[8]:

	Unnamed: 0	default	student	balance	income
count	10000.00000	10000	10000	10000.000000	10000.000000
unique	NaN	2	2	NaN	NaN
top	NaN	No	No	NaN	NaN
freq	NaN	9667	7055	NaN	NaN
mean	5000.50000	NaN	NaN	835.374886	33516.981876
std	2886.89568	NaN	NaN	483.714985	13336.639563
min	1.00000	NaN	NaN	0.000000	771.967729
25%	2500.75000	NaN	NaN	481.731105	21340.462903
50%	5000.50000	NaN	NaN	823.636973	34552.644802
75%	7500.25000	NaN	NaN	1166.308386	43807.729272
max	10000.00000	NaN	NaN	2654.322576	73554.233495

Analysis of Zero Values in Predictors

In [9]:
```
#499 rows of the balance variable contain is 0 value

#499 default

(tbl.balance==0).sum(axis=0)
```

Out[9]:
```
499
```

Categorical Variable Analysis

In [10]:
```
tbl.student.value_counts()
```

Out[10]:
```
student
No     7055
Yes    2945
Name: count, dtype: int64
```

Response Variable Analysis

In [11]:
```
tbl.default.value_counts()
```

Out[11]:
```
default
No     9667
Yes     333
Name: count, dtype: int64
```

Encode Categorical Variables

In [12]:
```
tbl['default2']=tbl.default.factorize()[0]

tbl['student2']=tbl.student.factorize()[0]

tbl.head(3)
```

Out[12]:

	Unnamed: 0	default	student	balance	income	default2	student2
0	1	No	Yes	729.526495	44361.625074	0	0
1	2	No	Yes	817.180407	12106.134700	0	0
2	3	No	No	1073.549164	31767.138947	0	1

Graphical Representation

In [13]:
```
tbl_dfno=tbl[tbl.default2==0].sample(frac=0.15)

tbl_dfyes=tbl[tbl.default2==1]

tbl_df=tbl_dfno._append(tbl_dfyes)
```

In [14]:
```
tbl_df
```

Out[14]:

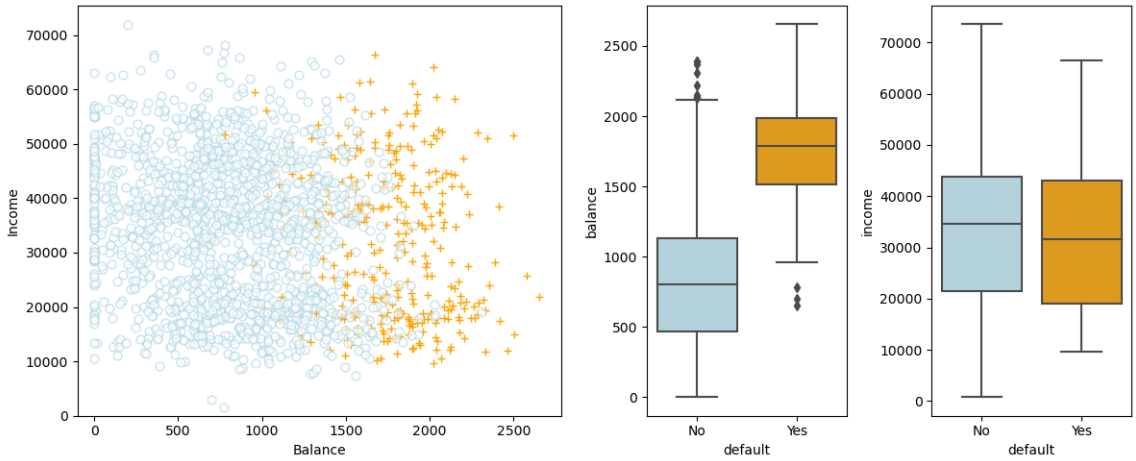
	Unnamed: 0	default	student	balance	income	default2	student2
6702	6703	No	No	512.513526	44178.840504	0	1
4875	4876	No	No	0.000000	37825.221415	0	1
7279	7280	No	No	1074.791401	25715.177454	0	1
758	759	No	Yes	638.691983	18148.301705	0	0
8434	8435	No	No	0.000000	48025.616855	0	1
...	...	...	...	...	...	...	...
9912	9913	Yes	No	2148.898454	44309.917173	1	1
9921	9922	Yes	Yes	1627.898323	17546.997016	1	0
9949	9950	Yes	No	1750.253150	51578.940163	1	1
9951	9952	Yes	No	1515.606239	48688.512086	1	1
9978	9979	Yes	No	2202.462395	47287.257108	1	1

1783 rows × 7 columns

In [15]:
```
fig=plt.figure(figsize=(12,5))
gs=mpl.gridspec.GridSpec(1,4)
ax1=plt.subplot(gs[0,:2])
ax2=plt.subplot(gs[0,2:3])
ax3=plt.subplot(gs[0,3:4])
ax1.scatter(tbl_df[tbl_df.default=='Yes'].balance,
            tbl_df[tbl_df.default=='Yes'].income,s=40,c='orange',marker='+
ax1.scatter(tbl_df[tbl_df.default=='No'].balance,
            tbl_df[tbl_df.default=='No'].income,s=40,marker='o',linewidths
            edgecolors='lightblue',facecolors='white',alpha=.6)


#--------------------------------------------------------------------

ax1.set_ylim(ymin=0)
ax1.set_ylabel('Income')
ax1.set_xlim(xmin=-100)
ax1.set_xlabel('Balance')
c_palette={'No':'lightblue','Yes':'Orange'}
sns.boxplot(x='default',y='balance',data=tbl,orient='v',ax=ax2,palette=c_pa
sns.boxplot(x='default',y='income',data=tbl,orient='v',ax=ax3,palette=c_pal
gs.tight_layout(plt.gcf())
```



Logistic Regression Using sklearn

In [17]:
```
X_train=tbl.balance.values.reshape(-1,1)
```

In [18]:
```
y=tbl.default2
```

In [19]:
```
X_test=np.arange(tbl.balance.min(),tbl.balance.max()).reshape(-1,1)
```

Logistic Regression Using sklearn

In [20]:
```
import sklearn.linear_model as skl_lm
```

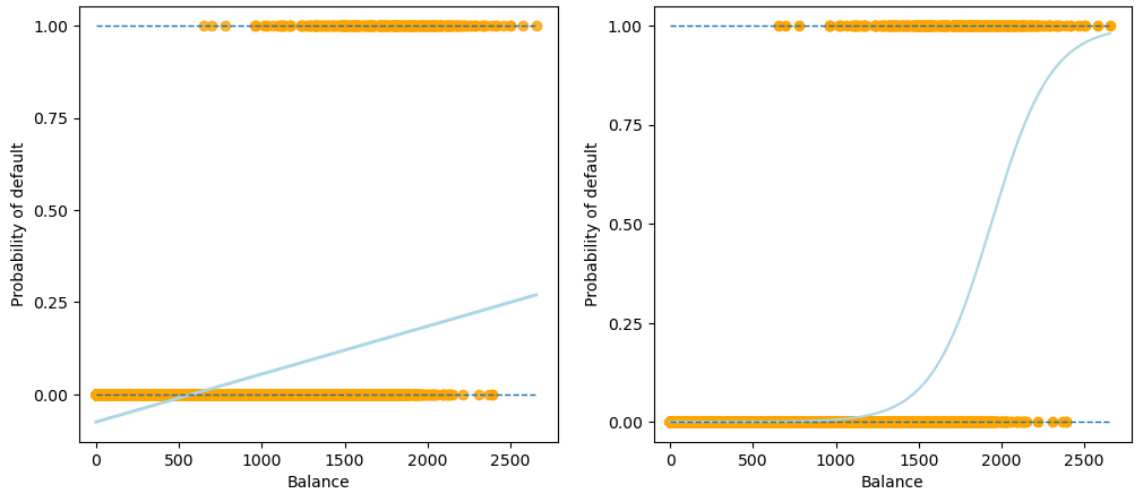
In [21]:
```
clf=skl_lm.LogisticRegression(solver='newton-cg')
```

In [22]:
```
clf.fit(X_train,y)
```

Out[22]:
```
LogisticRegression(solver='newton-cg')
```

In [23]:
```
prob=clf.predict_proba(X_test)
```

In [24]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(12,5))
sns.regplot(x=tbl.balance,
            y=tbl.default2,
            order=1,ci=None,
            scatter_kws={'color':'orange'},
            line_kws={'color':'lightblue','lw':2},
            ax=ax1)
ax2.scatter(X_train,y,color='orange')
ax2.plot(X_test,prob[:,1],color='lightblue')
for ax in fig.axes:
    ax.hlines(1,xmin=ax.xaxis.get_data_interval()[0],xmax=ax.xaxis.get_data
    ax.hlines(0,xmin=ax.xaxis.get_data_interval()[0],xmax=ax.xaxis.get_data
    ax.set_ylabel('Probability of default')
    ax.set_xlabel('Balance')
    ax.set_yticks([0,0.25,0.5,0.75,1.])
    ax.set_xlim(xmin=-100)
```



In [25]:
```
#Print the values of coefficient predicted beta0,beta1 and array of distinc
```

In [26]:
```
print(clf)
```

```
LogisticRegression(solver='newton-cg')
```

In [27]:
```
print('classes:',clf.classes_)
```

```
classes: [0 1]
```

In [28]:
```
print('coefficients:',clf.coef_)
```

```
coefficients: [[0.00549892]]
```

In [29]:
```
print('intercept:',clf.intercept_)
```

```
intercept: [-10.65133019]
```

Logistic Regression (X=Balance) Using statsmodel

In [30]:
```
import statsmodels.api as sm
import statsmodels.discrete.discrete_model as sms
```

In [31]:
```
X_train=sm.add_constant(tbl.balance)
est=sm.Logit(y.ravel(),X_train).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.079823
         Iterations 10
```

In [32]:
```
est.summary2().tables[1]
```

Out[32]:

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-10.651331	0.361169	-29.491287	3.723665e-191	-11.359208	-9.943453
balance	0.005499	0.000220	24.952404	2.010855e-137	0.005067	0.005931

Logistic Regression (Dummy Variable) Using statsmodel

In [33]:
```
X_train=sm.add_constant(tbl.student2)
y=tbl.default2
est=sms.Logit(y,X_train).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.145436
         Iterations 7
```

In [34]:
```
print(est.summary().tables[1].as_text())
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.
975]
------------------------------------------------------------------------------
const         -3.0996      0.091    -34.169      0.000      -3.277      -
2.922
student2      -0.4044      0.115     -3.516      0.000      -0.630      -
0.179
==============================================================================
```

Multiple Logistic Regression

In [35]:
```
X_train=sm.add_constant(tbl[['balance','income','student2']])

est=sms.Logit(y,X_train).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.078577
         Iterations 10
```

In [36]:
```
print(est.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.
975]
------------------------------------------------------------------------------
const        -11.5158      0.438    -26.299      0.000     -12.374     -1
0.658
balance        0.0057      0.000     24.736      0.000       0.005
0.006
income      3.033e-06    8.2e-06      0.370      0.712    -1.3e-05    1.91
e-05
student2       0.6468      0.236      2.738      0.006       0.184
1.110
==============================================================================
```

In [ ]:
```

```