Machine Learning

Samatrix Consulting Pvt Ltd

Samatrix.io

Project – Credit Card Default

Project - Introduction

Project - Finance

- Predict whether a credit card user will default on monthly credit card payment based on annual income and monthly credit card balance

Project Steps Followed

- Define Project Goals/Objective
- Data Retrieval
- Data Cleansing
- Exploratory Data Analysis
- Data Modeling
- Result Analysis

Project - Introduction

- We have information about credit card balance and annual income for 10,000 individuals.
- Based on the data, we need to predict whether the individual will default on a monthly credit card balance

Project - Introduction

- Define Research Goals
  - Predict whether a credit card user will default on monthly credit card payment based on annual income and monthly credit card balance
- Data Set
  - The Data set can be downloaded
  - We have data about credit card balance and annual income for 10,000 individuals
  - By the end of the project, the learners will be able to learn the approaches required for Logistic Regression, and LDA

Import Libraries

Import the Libraries

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: import matplotlib as mpl

In [4]: import matplotlib.pyplot as plt

In [5]: import seaborn as sns
```

Load the Data

Load the Data

```
In [7]: ccdef = pd.read_excel('Data/Default.xlsx')
```

View the raw data

```
In [8]: ccdef.head()
Out[8]:
   Unnamed: 0 default student     balance       income
0           1      No      No  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
```

Dimension of the Data

```
In [9]: ccdef.shape
Out[9]: (10000, 5)
```

We get the dimension of the dataset. The dataset has 10000 rows and 5 columns.

Data Type

```
In [10]: ccdef.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
```

Samatrix.io

Data Type

- Our observations are as follows
  - NaN values do not present in the data set. Because of the Non-Null Count and number of rows in the dataset match.
  - There are 3 Input Variables and 1 Output Variable (default)
  - The data type of balance and income variables is float64. The data type of out variable (default) and student is object
  - Shows two input variables continuous (quantitative) data types.
  - Output variable as well as 1 input variable (student) are categorical (qualitative) data types
  - None of the columns contain the Null Values

Null Values

```
In [11]: ccdef.isnull().sum()
Out[11]:
Unnamed: 0    0
default       0
student       0
balance       0
income        0
dtype: int64
```

The dataset does not contain any null values

Samatrix.io

Exploratory Data Analysis

Statistical Analysis

```
In [13]:
ccdef.describe(include='all')
```

Out[13]:

	Unnamed: 0	balance	income
count	10000.00	10000.00	10000.00
mean	5000.50	835.37	33516.98
std	2886.90	483.71	13336.64
min	1.00	0.00	771.97
25%	2500.75	481.73	21340.46
50%	5000.50	823.64	34552.64
75%	7500.25	1166.31	43807.73
max	10000.00	2654.32	73554.23

We can see that the min value of balance is zero. We need to confirm how many zero values existing in the dataset.

For all other columns, the data cleaning is not required. However for categorical variables, the encoding is required.

Analysis of Zero Values in Predictors

```
In [14]: (ccdef.balance == 0).sum(axis=0)
Out[14]: 499
```

499 rows of the balance variable contain the zero value, which is possible. Hence we conclude the data cleaning steps are not required for the balance variable

Categorical Variable Analysis

```
In [15]: ccdef.student.value_counts()
Out[15]:
No      7056
Yes     2944
Name: student, dtype: int64
```

This confirms that the predictor student has only 2 possible values. Yes and No. The distribution of students vs non-students is given above.

Samatrix.io

Response Variable Analysis

```
In [16]: ccdef.default.value_counts()
Out[16]:
No      9667
Yes      333
Name: default, dtype: int64
```

This confirms that the response variable default has only 2 possible values. Yes and No. Data is highly skewed. Only 3.33% of the individuals in training data defaulted.

Encode Categorical Variables

Most machine learning models accept the numerical data only. It is necessary to pre-process the categorical variables. We need to convert the categorical variables into numbers. For any machine learning project, converting categorical data is an unavoidable activity.

We have created two dummy variable columns `student2` and `default2` after encoding the categorical data

```
In [17]: ccdef['default2'] =
ccdef.default.factorize()[0]


In [18]: ccdef['student2'] =
ccdef.student.factorize()[0]


In [19]: ccdef.head(3)
Out[19]:
   Unnamed: 0 default student  balance    income  default2  student2
0           1      No      No   729.53  44361.63         0         0
1           2      No     Yes   817.18  12106.13         0         1
2           3      No      No  1073.55  31767.14         0         0
```

Graphical Representation

Relationship between `balance` and `income` and the relationship between `default` and `balance`, and `default` and `income`, has been plotted.

We create a new data frame, `ccdef_df`, that includes 15% data for non defaulters and whole data for defaulters

```
In [20]: ccdef_dfno = ccdef[ccdef.default2 ==
0].sample(frac=0.15)

In [21]: ccdef_dfyes = ccdef[ccdef.default2 ==
1]

In [22]: ccdef_df = ccdef_dfno.append(ccdef_dfyes)
```

Samatrix.io

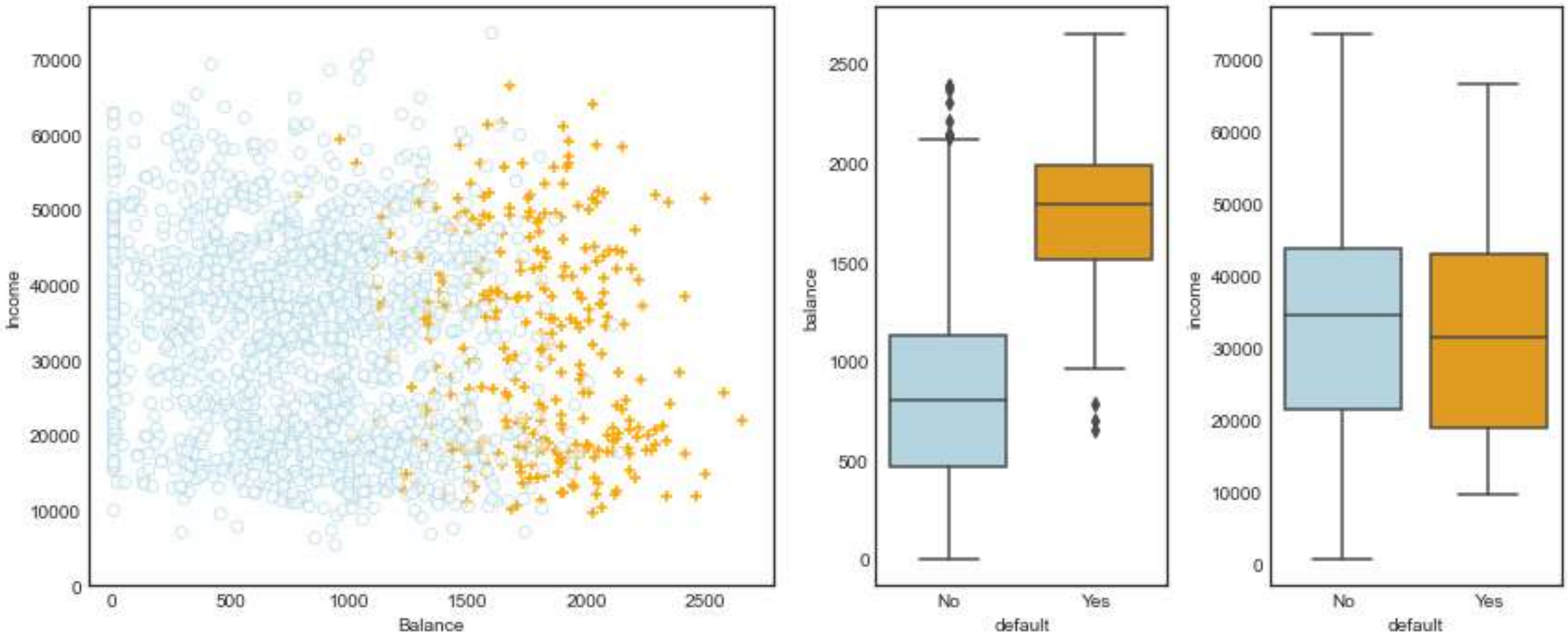
Graphical Representation

```
In [24]: 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(ccdef_df[ccdef_df.default ==
'Yes'].balance, ccdef_df[ccdef
    ...: _df.default == 'Yes'].income, s=40,
c='orange', marker='+', linewidths=1)
    ...: ax1.scatter(ccdef_df[ccdef_df.default ==
'No'].balance, ccdef_df[ccdef_
    ...: df.default == 'No'].income, s=40,
marker='o', linewidths=1,
                         edgecolors='lightblue',
```

Graphical Representation

```
    ...: 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=ccdef, orient='v', ax=ax2, palet
    ...: te=c_palette)
    ...: sns.boxplot(x='default’, y='income', data=ccdef,
orient='v', ax=ax3, palett
    ...: e=c_palette)
    ...: gs.tight_layout(plt.gcf())
```

Graphical Representation

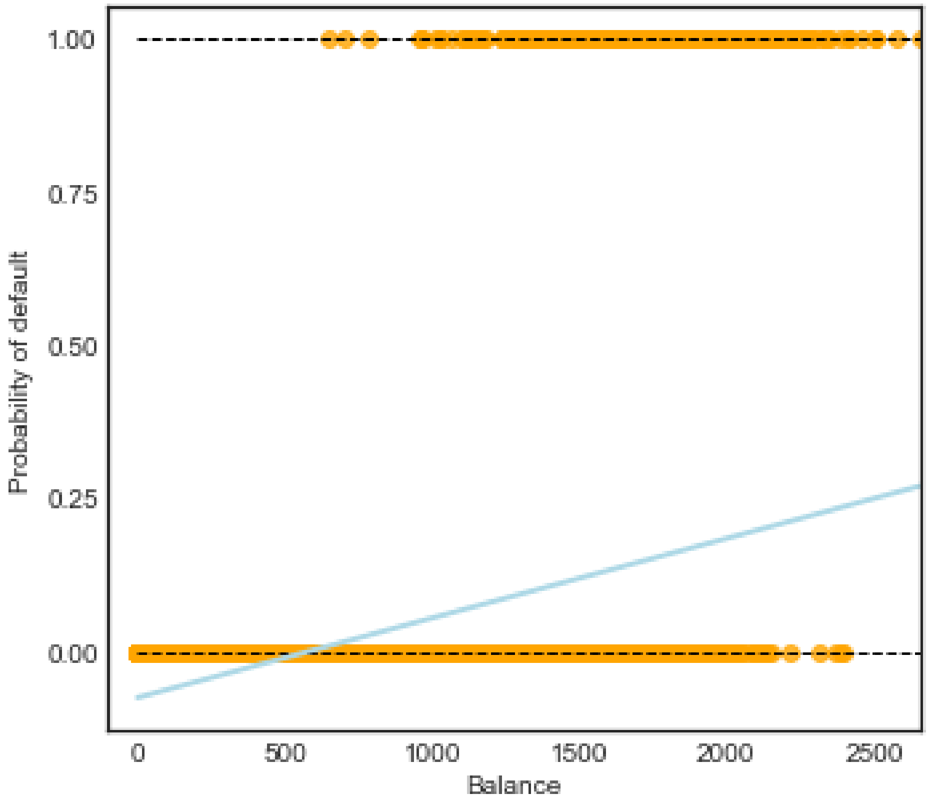


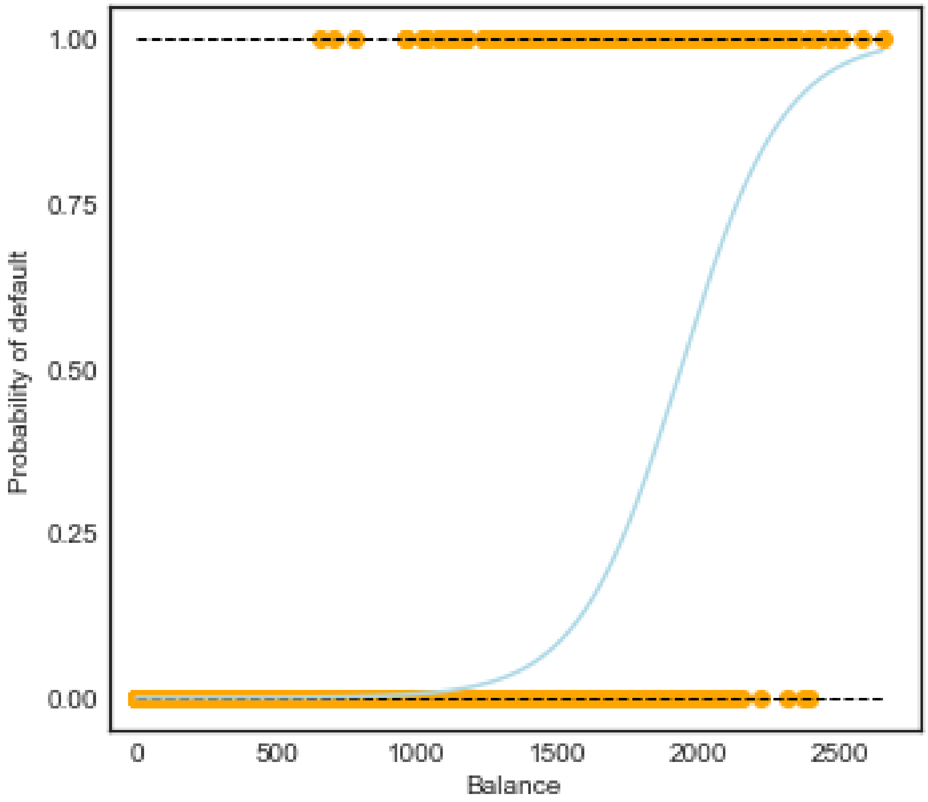
Samatrix.io

Data Modeling

Samatrix.io

Logistic Regression Using sklearn





Samatrix.io

Logistic Regression Using sklearn

Create training and test data.

Training Data

input data (X) – `balance`

output data (y) – `default2`

Test Data

create new data varies between min and max value of `balance`

```
In [25]: X_train = ccdef.balance.values.reshape(-
1,1)

In [26]: y = ccdef.default2

In [27]: X_test = np.arange(ccdef.balance.min(),
ccdef.balance.max()).reshape(-1 ,1)
```

Samatrix.io

Logistic Regression Using sklearn

Calculate probability using logistic regression

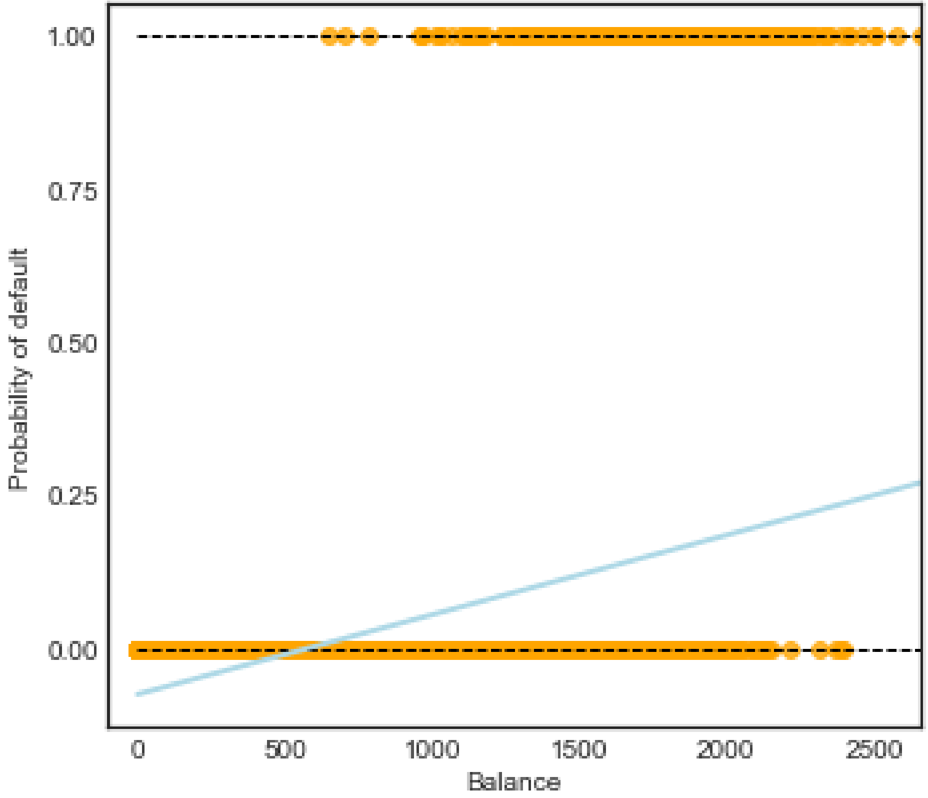
```
In [28]: import sklearn.linear_model as skl_lm

In [29]: clf = skl_lm.LogisticRegression(solver='newton-cg')

In [30]: clf.fit(X_train,y)
Out[30]: LogisticRegression(solver='newton-cg')

In [31]: prob = clf.predict_proba(X_test)
```

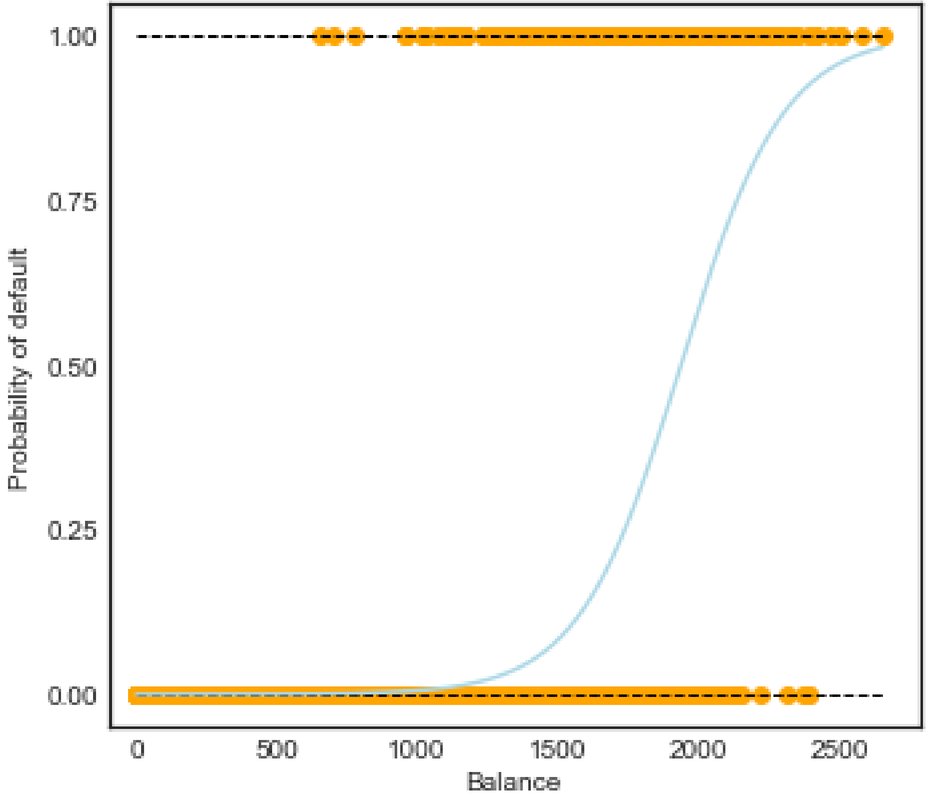
Logistic Regression Using sklearn

```
In [32]: fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))
    ...: sns.regplot(x=ccdef.balance, y=ccdef.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_inte
rval()[1], linestyles='dashed', lw=1)
    ...:     ax.hlines(0,
xmin=ax.xaxis.get_data_interval()[0],xmax=ax.xaxis.get_data_inte
rval()[1], linestyles='dashed', lw=1)
    ...:     ax.set_ylabel('Probability of default')
    ...:     ax.set_xlabel('Balance')
```

Samatrix.io

Logistic Regression Using sklearn





Samatrix.io

Logistic Regression Using sklearn

Print the values of coefficient $\hat{\beta}_0, \hat{\beta}_1$ and array of distinct classes that y takes

```
In [33]: print(clf)
LogisticRegression(solver='newton-cg')

In [34]: print('classes: ',clf.classes_)
classes:  [0 1]

In [35]: print('coefficients: ',clf.coef_)
coefficients:  [[0.00549892]]

In [36]: print('intercept :', clf.intercept_)
intercept : [-10.65133006]
```

Logistic Regression (X=Balance) Using statsmodel

```
In [37]: import statsmodels.api as sm

In [38]: import statsmodels.discrete.discrete_model as
sms

In [40]: X_train = sm.add_constant(ccdef.balance)
In [41]: est = sm.Logit(y.ravel(), X_train).fit()

Optimization terminated successfully.

         Current function value: 0.079823

         Iterations 10

In [42]: est.summary2().tables[1]

Out[42]:

               Coef.   Std.Err.          z         P>|z|
[0.025    0.975]

const    -10.65133   0.36117 -29.49129  3.72366e-191 -
11.35921 -9.94345
```

Logistic Regression (Dummy Variable) Using statsmodel

```
In [43]: X_train = sm.add_constant(ccdef.student2)
In [44]: y = ccdef.default2
In [45]: est = sms.Logit(y, X_train).fit()
Optimization terminated successfully.
         Current function value: 0.145434
         Iterations 7
In [46]: print(est.summary().tables[1].as_text())
=============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------
const         -3.5041      0.071    -49.554      0.000      -3.643      -3.366
```

Samatrix.io

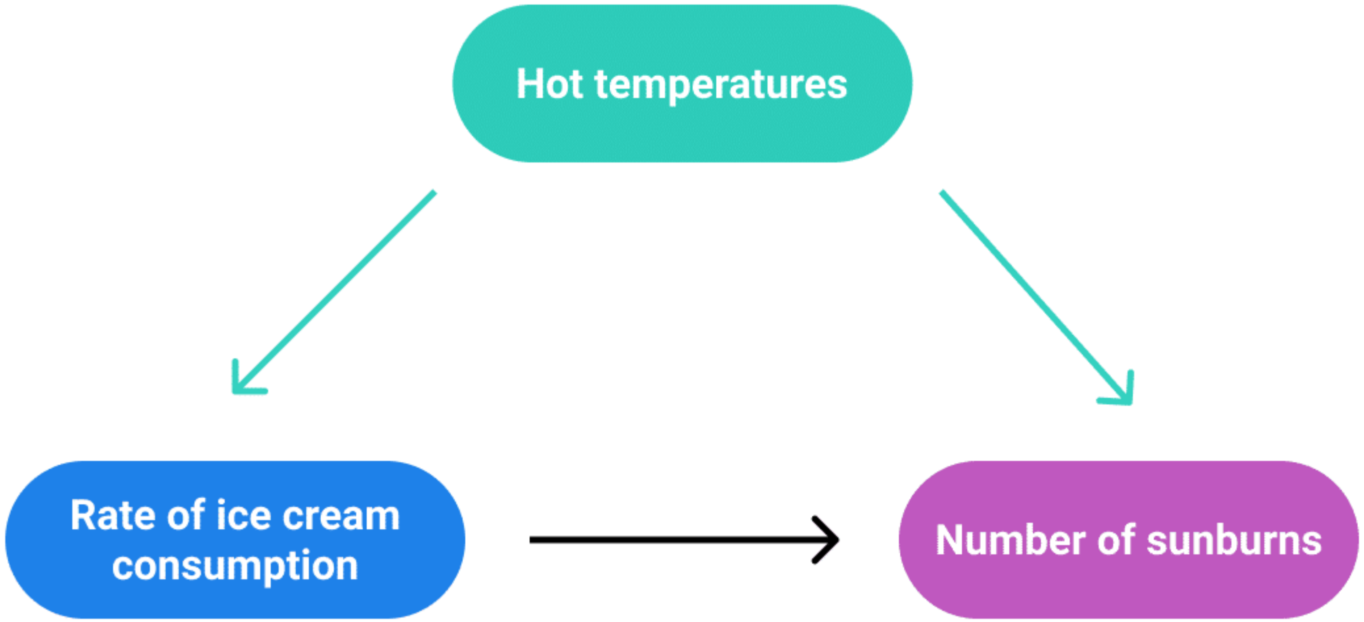
Multiple Logistic Regression

```
In [47]: X_train = sm.add_constant(ccdef[['balance',
'income', 'student2']])

In [48]: est = sms.Logit(y,
X_train).fit()
Optimization terminated successfully.
         Current function value: 0.078577
         Iterations 10

In [49]: print(est.summary().tables[1])
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -10.8690      0.492    -22.079      0.000     -11.834      -9.904
balance        0.0057      0.000     24.737      0.000       0.005       0.006
income      3.033e-06    8.2e-06      0.370      0.712    -1.3e-05    1.91e-05
student2      -0.6468      0.236     -2.738      0.006      -1.110      -0.184
```

Confounding Variable

- Confounding variables are those that affect other variables in a way that produces spurious or distorted associations between two variables.
- They confound the "true" relationship between two variables.
- a confounding variable is an unmeasured third variable that influences both the supposed cause and the supposed effect.
- It must be correlated with the independent variable. This may be a causal relationship, but it does not have to be.
- It must be causally related to the dependent variable.

Confounding variable



Confounding

Create balance and default vectors for students

```
In [50]: X_train = ccdef[ccdef.student ==
'Yes'].balance.values.reshape(-1,1)


In [51]: y = ccdef[ccdef.student == 'Yes'].default2
```

Create balance and default vectors for non- students

```
In [52]: X_train2 = ccdef[ccdef.student ==
'No'].balance.values.reshape(-1,1)


In [53]: y2 = ccdef[ccdef.student == 'No'].default2
```

Create test vector

```
In [54]: X_test = np.arange(ccdef.balance.min(),
ccdef.balance.max()).reshape(-1,1)
```

Confounding

Fit both dataset to Logistic Regression

```
In [55]: clf = skl_lm.LogisticRegression(solver='newton-cg’)

In [56]: clf2 = skl_lm.LogisticRegression(solver='newton-cg')

In [57]: clf.fit(X_train,y)
Out[57]: LogisticRegression(solver='newton-cg')

In [58]: clf2.fit(X_train2,y2)
Out[58]: LogisticRegression(solver='newton-cg’)
```

Calculate Probabilities

```
In [59]: prob = clf.predict_proba(X_test)

In [60]: prob2 = clf2.predict_proba(X_test)
```

Samatrix.io

Confounding

Confusion Matrix

```
In [61]:
ccdef.groupby(['student','default']).size().unstack('default')
Out[61]:
default      No   Yes
student
No         6850   206
Yes        2817   127
```

Samatrix.io

Confounding

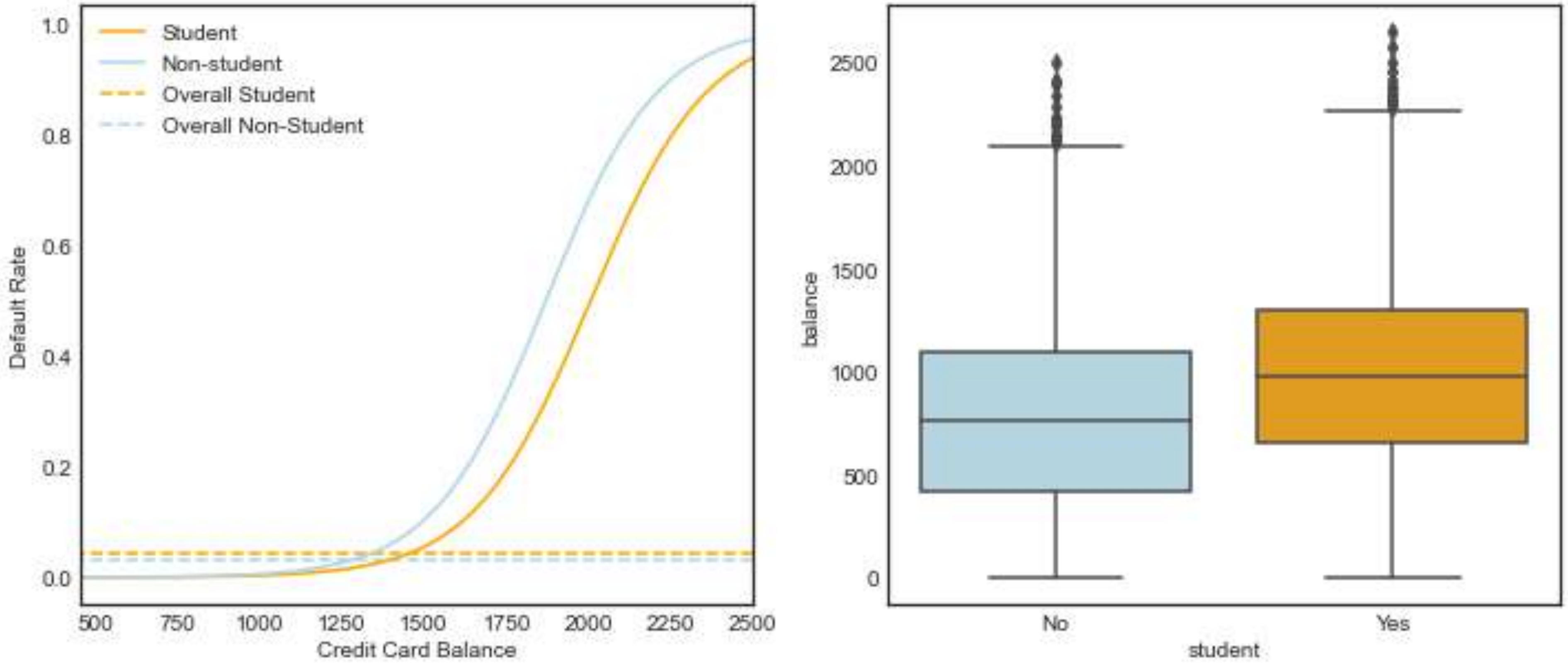
```
In [62]: fig, (ax1, ax2) = plt.subplots(1,2,
figsize=(12,5))
    ...: # Left plot
    ...: ax1.plot(X_test, prob[:,1],
color='orange', label='Student')
    ...: ax1.plot(X_test, prob2[:,1],
color='lightblue', label='Non-student')
    ...: ax1.hlines(127/2817, colors='orange',
label='Overall
Student',xmin=ax1.xaxis.get_data_interval()[0],xm
ax=ax1.xaxis.get_data_interval()[1],
linestyles='dashed')
    ...: ax1.hlines(206/6850, colors='lightblue',
```

Confounding

```
   ...: ax1.set_ylabel('Default Rate')
   ...: ax1.set_xlabel('Credit Card Balance')
   ...: ax1.set_yticks([0, 0.2, 0.4, 0.6, 0.8,
1.])
   ...: ax1.set_xlim(450,2500)
   ...: ax1.legend(loc=2)
   ...: # Right plot
   ...: sns.boxplot(x='student’, y='balance',
data=ccdef, orient='v',
ax=ax2,  palette=c_palette);
```

Samatrix.io

Confounding



Samatrix.io

Linear Discriminant Analysis

50% Threshold

```
In [63]: from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis

In [64]: X = ccdef[['balance', 'income', 'student2']]

In [65]: y = ccdef.default2

In [66]: lda = LinearDiscriminantAnalysis(solver='svd')

In [67]: y_pred = lda.fit(X, y).predict(X)

In [68]: ccdef_df = pd.DataFrame({'True default status':
y, 'Predicted default status': y_pred})
```

Samatrix.io

Linear Discriminant Analysis

```
In [69]: ccdef_df.replace(to_replace={0:'No',
1:'Yes'}, inplace=True)

In [70]: ccdef_df.groupby(['Predicted default
status','True default
status']).size().unstack('True default
status')


Out[70]:
True default status          No   Yes
Predicted default status
No                         9645   254
Yes                          22    79
```

Linear Discriminant Analysis

20% Threshold

```
In [71]: decision_prob =
0.2

In [72]: y_prob = lda.fit(X,
y).predict_proba(X)


In [73]: ccdef_df = pd.DataFrame({'True default status':
y,'Predicted default status': y_prob[:,1] >
decision_prob})



In [74]: ccdef_df.replace(to_replace={0:'No', 1:'Yes',
'True':'Yes', 'False':'No '}, inplace=True)
```

Samatrix.io

Linear Discriminant Analysis

```
In [75]: ccdef_df.groupby(['Predicted default
status','True default
status']).size().unstack('True default
status')

Out[75]:
True default status         No   Yes
Predicted default status
No                        9435   140
Yes                        232   193
```

Thanks

Samatrix Consulting Pvt Ltd

Samatrix.io