

# Credit Card

## Lead Prediction

### 組員名單

- 王之璇
- 陳品臻
- 蘇柏羽
- 王育綸
- 黃馨毅





## 專案動機

1. 找尋潛在商機
- 2 降低人力成本
3. 提升客戶好感度

# “問題討論

- 何種因素(特徵值), 會影響到人們是否有興趣
- 基於因素(特徵值), 建立模型, 並利用模型預測潛在顧客的與否

Processing

資料處理

建立模型

Demo

總結



了解資料

資料前處理

資料比較

1 資料處理

2 建立模型

3 DEMO

4 總結

# 資料類型

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 245725 entries, 0 to 245724
```

```
Data columns (total 11 columns):
```

#	Column	Non-Null Count	Dtype
0	ID	245725 non-null	object
1	Gender	245725 non-null	object
2	Age	245725 non-null	int64
3	Region_Code	245725 non-null	object
4	Occupation	245725 non-null	object
5	Channel_Code	245725 non-null	object
6	Vintage	245725 non-null	int64
7	Credit_Product	216400 non-null	object
8	Avg_Account_Balance	245725 non-null	int64
9	Is_Active	245725 non-null	object
10	Is_Lead	245725 non-null	int64

```
dtypes: int64(4), object(7)
```

```
memory usage: 20.6+ MB
```

## 了解資料

ID

客戶名稱

Gender

客戶性別

Age

客戶年齡

Region\_Code

郵遞區號

Occupation

職業類別

Channel\_Code

銀行代號

Vintage

客戶存在  
(月份)

Credit\_Product

是否已有  
銀行服務

Avg\_Account  
\_Balance

資產

Is\_Active

是否活躍

Is\_Lead

是否有興趣



了解資料

資料前處理

資料比較

1 資料處理

2 建立模型

3 DEMO

4 總結



# 類別資料

Gender

```
#Gender_mapping 數字
Gender_mapping = {'Male':1, 'Female':0}
dataset['Gender'] = dataset['Gender'].map(Gender_mapping)
```

Occupation

```
#將值轉換
occupation_mapping={'Entrepreneur':0,"Salaried":1,"Self_Employed":2,"Other":3}
traindata['Occupation'] = traindata['Occupation'].map(occupation_mapping)
traindata.head()
```

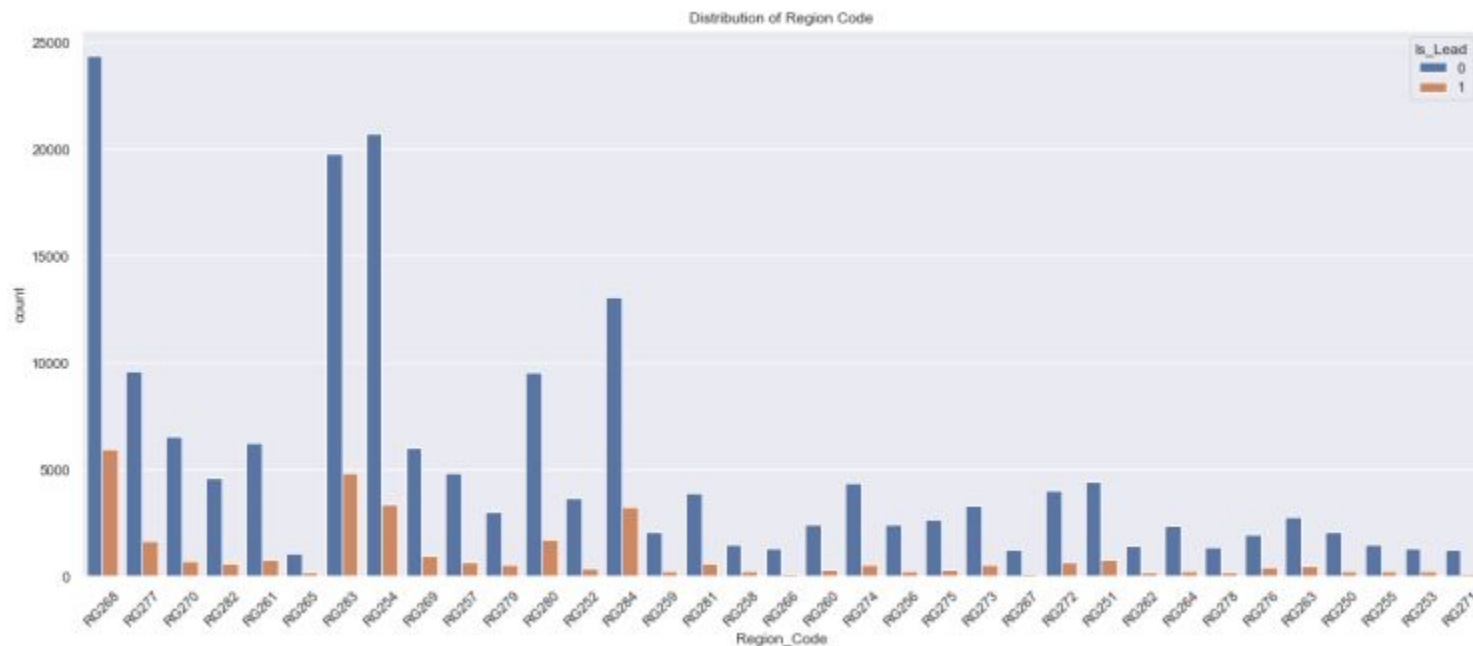
Channel\_code

```
#Channel_Code_mapping 數字
Channel_Code_mapping = {'X1':2, 'X3':1, 'X2':0, 'X4':0}
dataset['Channel_Code'] = dataset['Channel_Code'].map(Channel_Code_mapping)
```

Is\_Active

```
#Is_Active_mapping 數字
Is_Active_mapping = {'Yes':1, 'No':0}
dataset['Is_Active'] = dataset['Is_Active'].map(Is_Active_mapping)
```

# Region\_code



```
#依年齡區間做mapping function
```

```
Region_mapping={"RG268":1,"RG283":2,"RG254":3,"RG284":4,"RG277":5,"RG280":6,"RG269":7,  
                "RG270":8,"RG261":9,"RG257":10,"RG251":11,"RG282":12,"RG274":13,"RG272":14,  
                "RG281":15,"RG273":16,"RG252":17,"RG279":18,"RG263":19,"RG275":20,"RG260":21,  
                "RG256":22,"RG264":23,"RG276":24,"RG259":25,"RG250":26,"RG255":27,"RG258":28,  
                "RG253":29,"RG278":30,"RG262":31,"RG266":32,"RG265":33,"RG271":34,"RG267":35 }
```

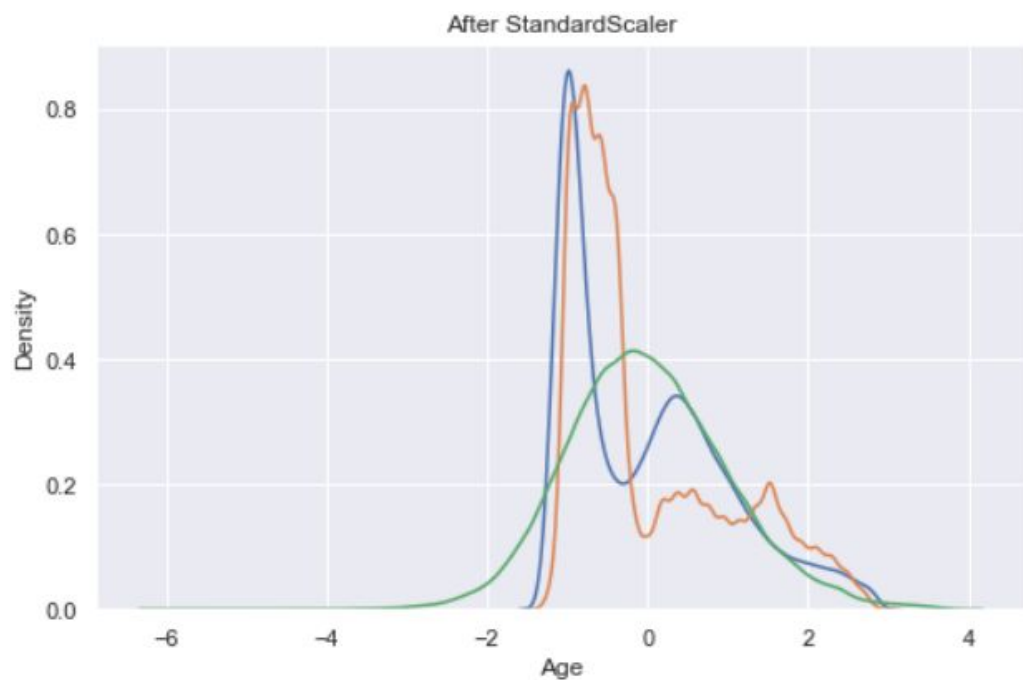
```
traindata['Region_Code'] = traindata['Region_Code'].map(Region_mapping)
```

```
#目前的datafram
```

```
traindata.head()
```

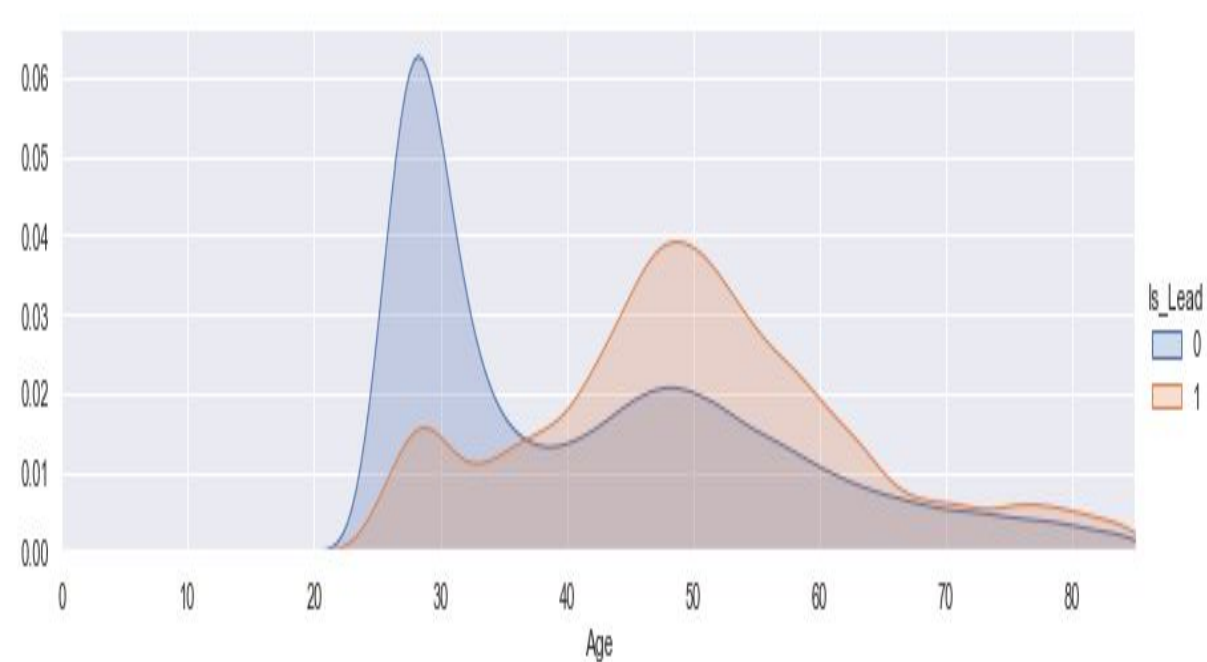
# Age

## “方法一



將Age標準化

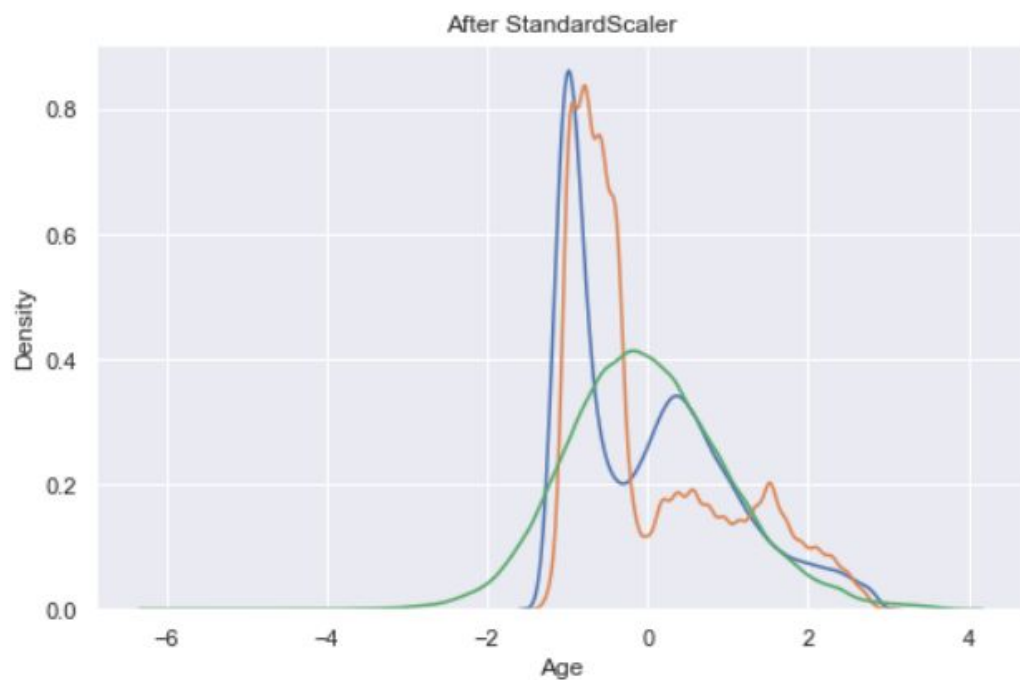
## “方法二



將Age按圖依特徵區分

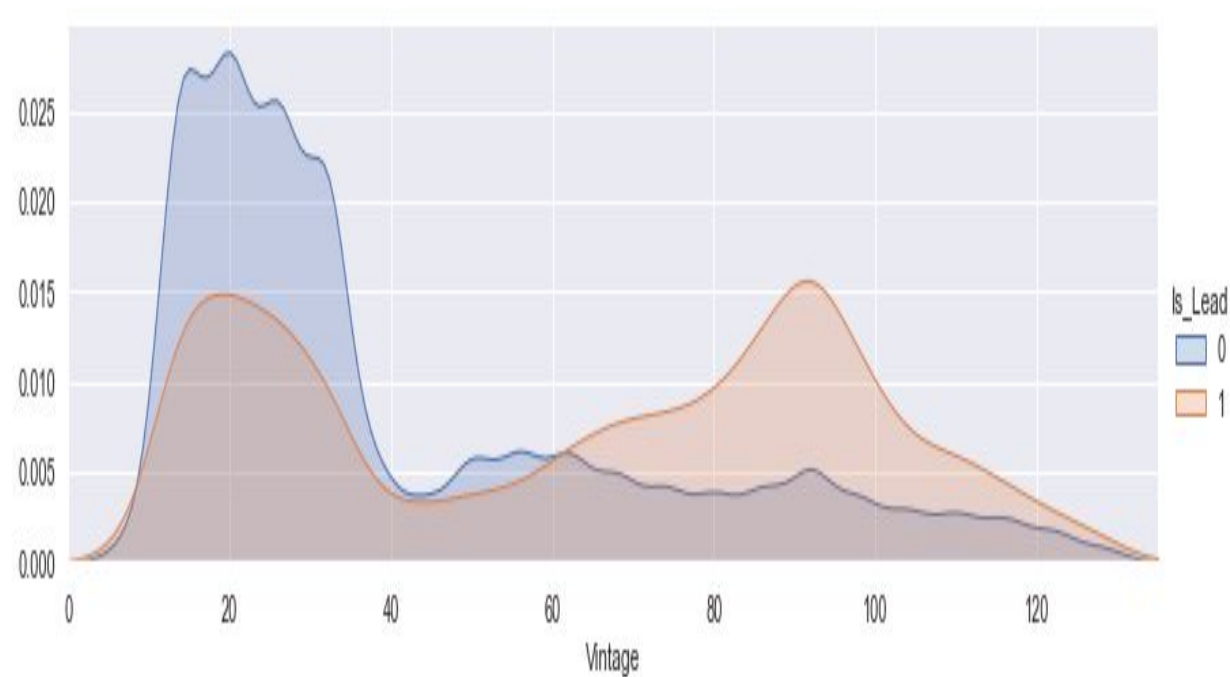
# Vintage

## “方法一



將Vintage標準化

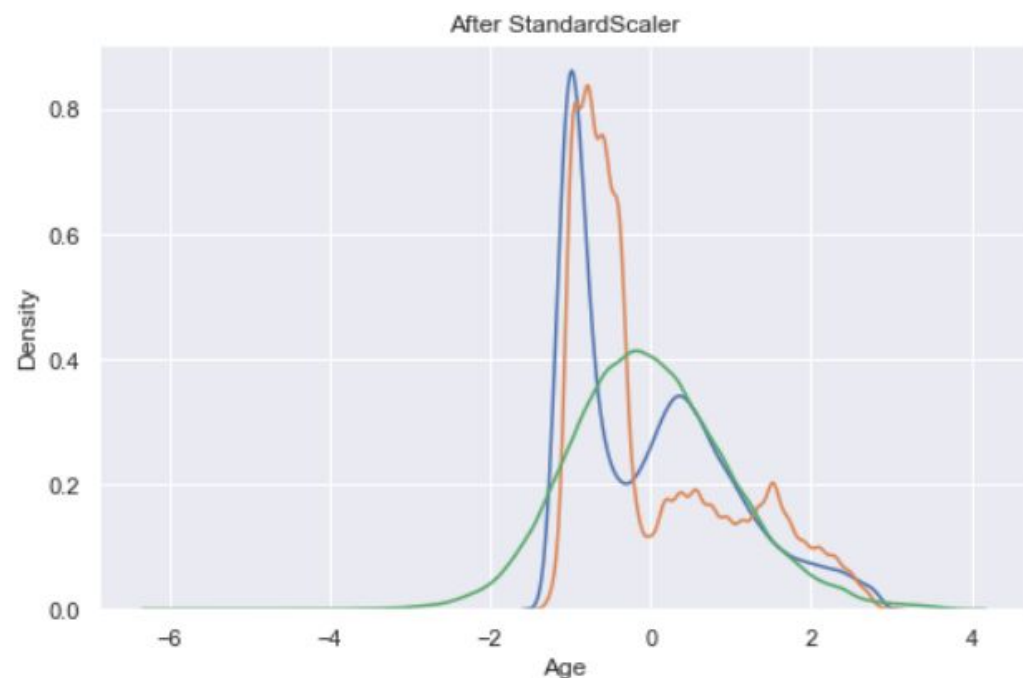
## “方法二



將Vintage按圖依特徵區分

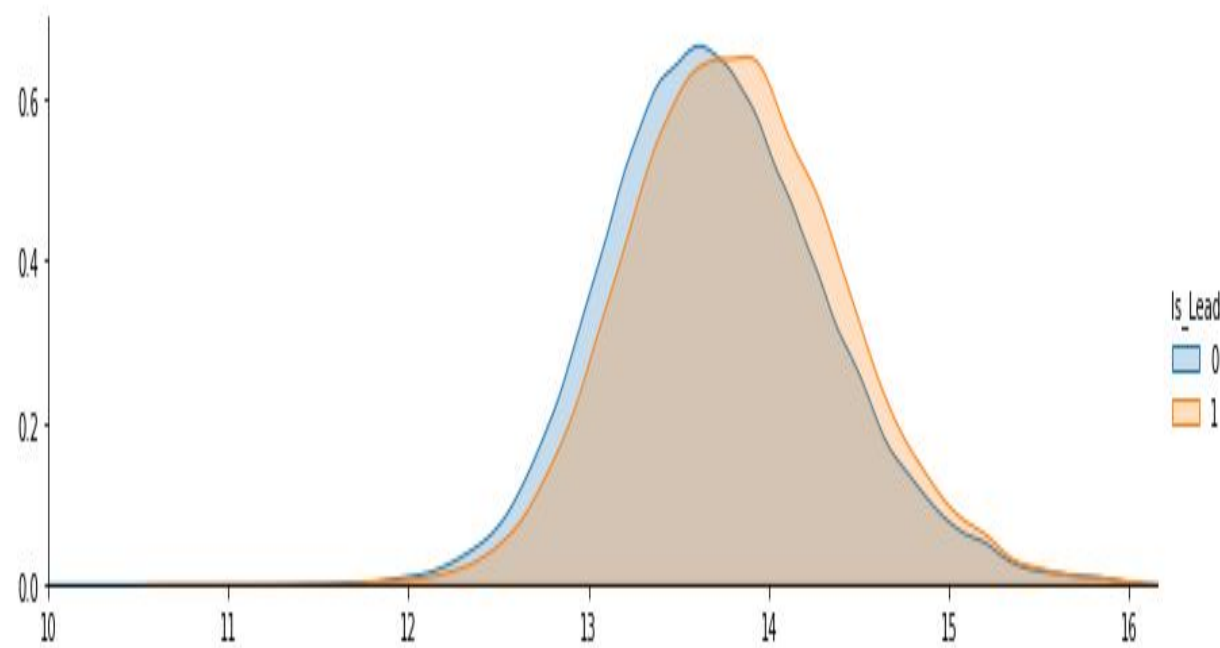
# Avg\_Account\_Balance

## “方法一



將Avg\_Account\_Balance 標準化

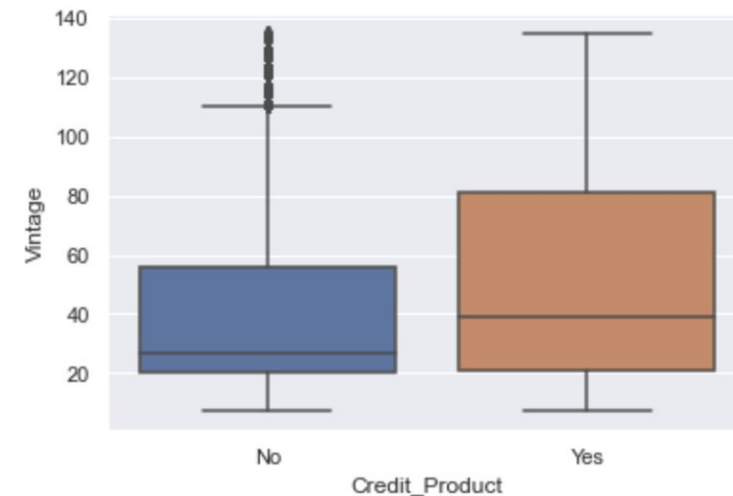
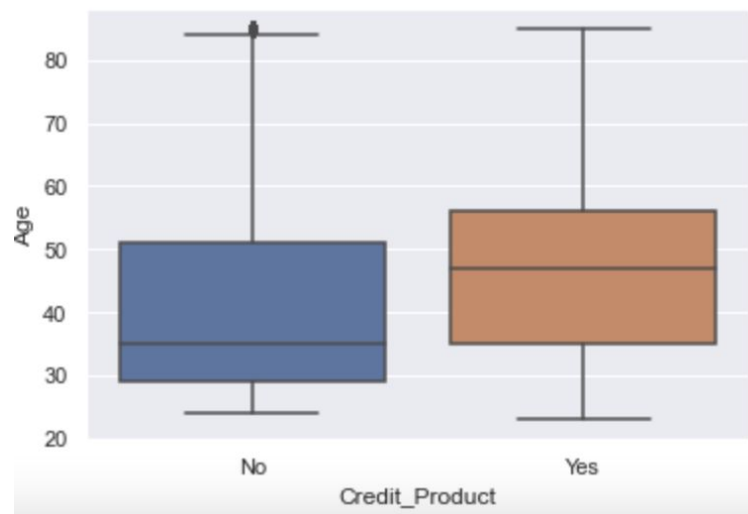
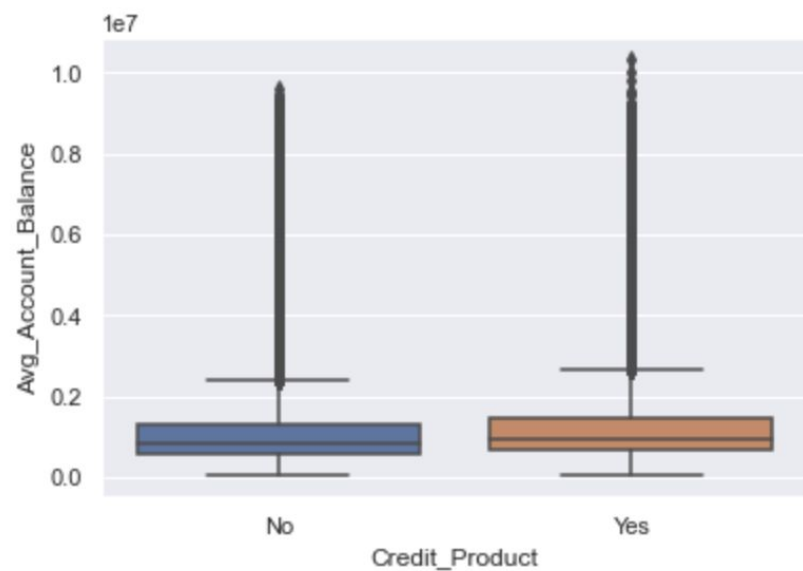
## “方法二



依據四分位數分

# 空值處理

“方法一



# 空值處理

## “方法一

```
# 找NA
null_indices = dataset[dataset['Credit_Product'].isnull() == True].index
```

```
# 補值函式
def fill_na(data, null_indices):
    for i in null_indices:
        if (55 <= dataset['Age'].iloc[i] <= 60) and (60 <= dataset['Age'].iloc[i] <= 70):
            dataset['Credit_Product'].iloc[i] = "Yes"
        elif 30 <= dataset['Age'].iloc[i] <= 40:
            dataset['Credit_Product'].iloc[i] = "No"
        else:
            dataset['Credit_Product'].iloc[i] = "Unkown"
```

```
# 補值
fill_na(dataset, null_indices)
```

```
Credit_Product_mapping = {'Unkown':2, 'Yes':1, 'No':0 }
```

模型準確  
度:84.79%

## “方法二

```
#missingvalue
null_indices=traindata[traindata['Credit_Product'].isnull()==True].index
```

刪除NaN

```
def fill_na(traindata, null_indices):
    for i in null_indices:
        if (55 <= traindata['Age'].iloc[i] <= 60) and (60 <= traindata['Age'].iloc[i] <= 70):
            traindata['Credit_Product'].iloc[i] = "Yes"
        elif 30 <= traindata['Age'].iloc[i] <= 40:
            traindata['Credit_Product'].iloc[i] = "No"
        else:
            traindata['Credit_Product'].iloc[i] = "Unknow"
```

```
fill_na(traindata, null_indices)
```

```
Credit_mapping={'No':0, 'Yes':1}
```

模型準確  
度:80.77%



## 資料比較

	同一參數		
	資料離散化	連續資料標準化	只對數值資料標準
LogisticRegression	85.04%	85.03%	85.03%
SGDClassifier	84.77%	84.96%	85.03%
KNeighborsClassifier	85.19%	85.46%	85.31%
GaussianNB	80.64%	80.15%	80.15%
MultinomialNB	84.63%	84.63%	--
BernoulliNB	82.26%	84.64%	82.17%
DecisionTreeClassifier	85.87%	86.01%	86.01%
RandomForestClassifier	85.88%	86.17%	86.13%
GradientBoostingClassifier	85.93%	86.12%	86.12%
MLPClassifier	85.89%	86.09%	86.10%
DNN	EPOCH:15->86% .0.865809559822	EPOCH:6->86% 0.869071900844	0.8764449358
Xgboost	85.85%	86.02%	86.02





了解資料

資料前處理

資料比較

1 資料處理

2 建立模型

3 DEMO

4 總結

# 取數值&類別的欄位

```
# 取數值 & 類別的欄位
data_num_cols = dataset_data._get_numeric_data().columns
data_cat_cols = dataset_data.columns.difference(data_num_cols)

print("Numeric columns: ", data_num_cols)
print()
print("Categorical columns: ", data_cat_cols)
```

```
Numeric columns:  Index(['Age', 'Vintage', 'Avg_Account_Balance'], dtype='object')
```

```
Categorical columns:  Index(['Credit_Product', 'Is_Active', 'Occupation'], dtype='object')
```

# 類別欄位值轉數值

```
from sklearn.preprocessing import LabelEncoder  
  
label = LabelEncoder()  
data_cat_data = data_cat_data.apply(LabelEncoder().fit_transform)
```

```
data_num_data_s.reset_index(drop=True, inplace=True)  
data_cat_data.reset_index(drop=True, inplace=True)  
#df = pd.concat([df1, df2], axis=1)  
  
data_new = pd.concat([data_num_data_s, data_cat_data], axis = 1)
```

# 數值欄位數值標準化

#利用StandardScaler標準化

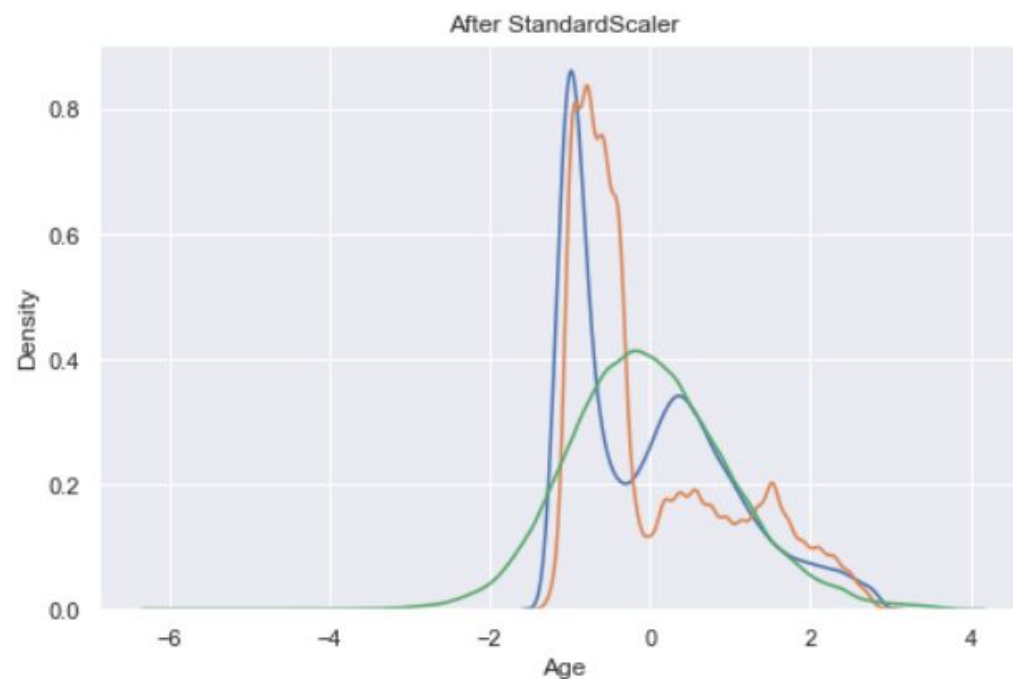
```
from sklearn import preprocessing
s_scaler = preprocessing.StandardScaler()

data_num_data_s = s_scaler.fit_transform(data_num_data)

data_num_data_s = pd.DataFrame(data_num_data_s, columns = data_num_cols)

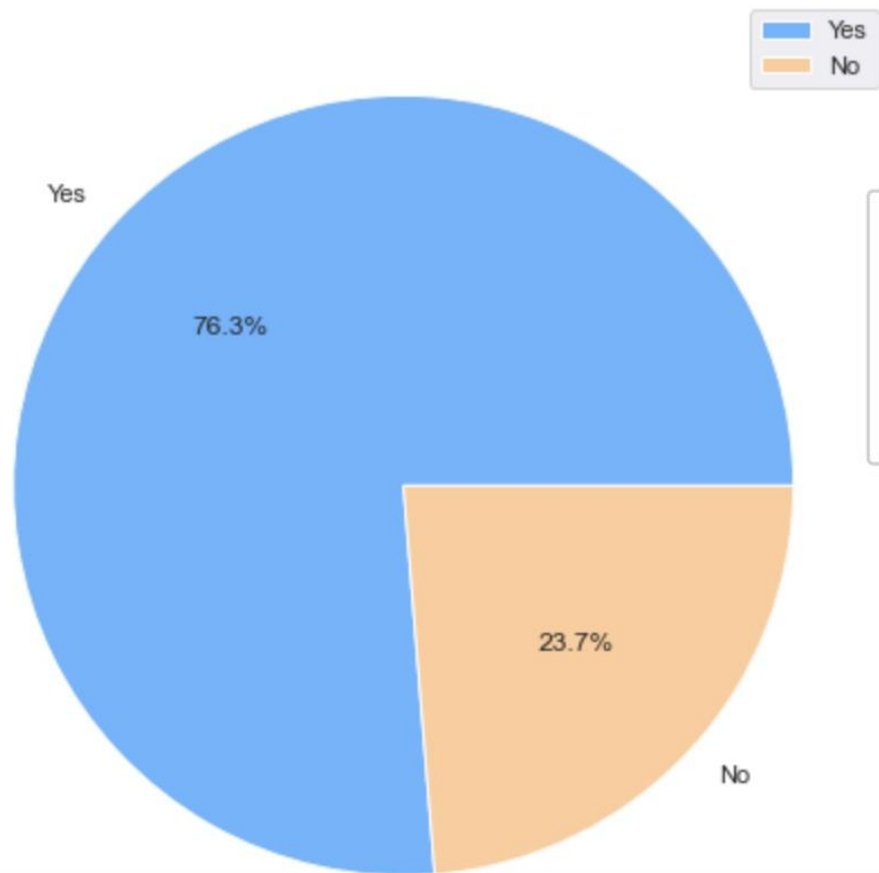
fig, (ax1) = plt.subplots(ncols=1, figsize=(8, 5))
ax1.set_title('After StandardScaler')

sns.kdeplot(data_num_data_s['Age'], ax=ax1)
sns.kdeplot(data_num_data_s['Vintage'], ax=ax1)
sns.kdeplot(data_num_data_s['Avg_Account_Balance'], ax=ax1);
```



## STOME

數據樣本中，發現有正/負資料比例佔不平均，竟而影響到機器學習之模型失效，使準確率下降。在實際生活案例中，存在問題的樣本一般會遠少於正常樣本。故，透過Synthetic Minority Oversampling Technique，在少數類樣本之間利用差值來產生額外的樣本




```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy='minority')
X_sm, y_sm = smote.fit_resample(traindata_data, traindata_target)
```

## 將資料分成訓練集(TrainDataset)以及測試資料集(TestDataset)

```
# 因為測試資料沒有答案無法判斷模型的優劣所以用此方法判斷模型優劣
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X_sm, y_sm, test_size=0.2, random_state=15, stratify=y_sm)
```



訓練資料80% 測試資料  
20%



SMOTE

建立模型

模型比較

1 資料處理

2 建立模型

3 DEMO

4 總結

# 設定預測目標變數與解釋變數

In [7]:

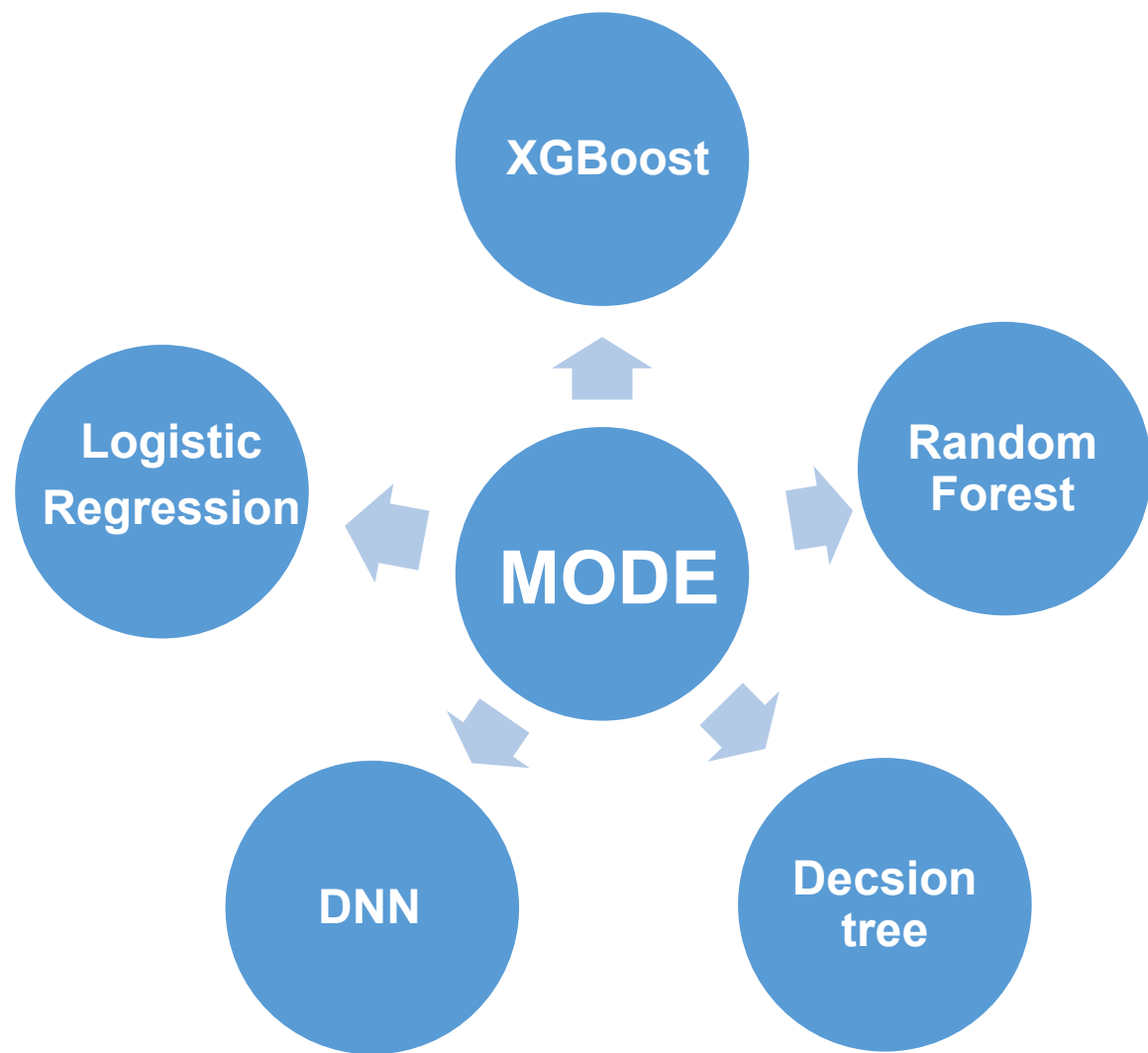
```
#設定預測目標變數與解釋變數
dataset_data = dataset.drop('Is_Lead', axis = 1)
dataset_target = dataset['Is_Lead']

#Survived為series, 加入[]轉成dataframe
dataset_target2 = dataset[['Is_Lead']]

dataset_data.shape, dataset_target2.shape
```



## 建立模型

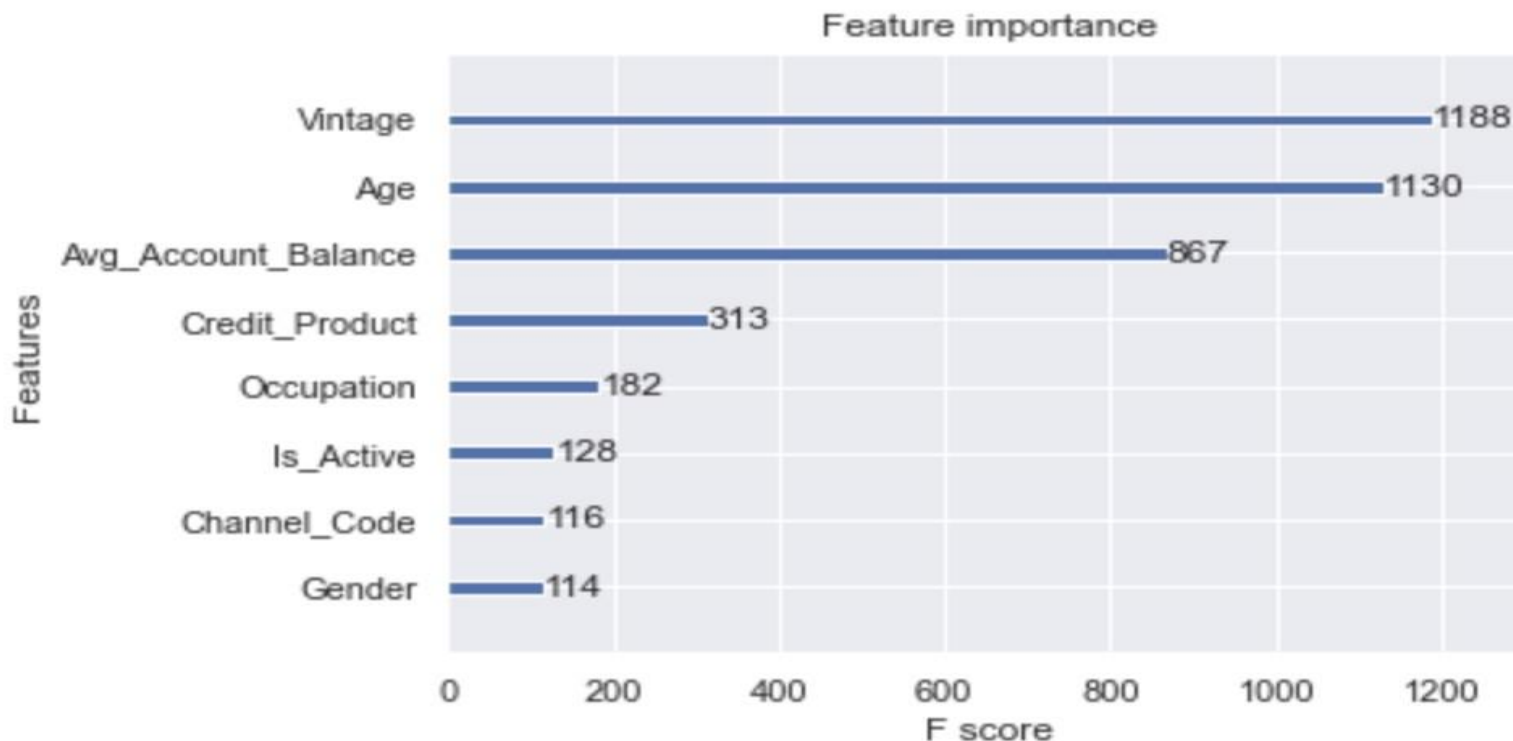


# 資料比較

	同一參數						調整參數
	資料離散化	連續資料標準化	只對數值資料標準化	資料離散化 SMOTE	連續資料標準化 SMOTE	只對數值資料標準化 SMOTE	
LogisticRegression	85.04%	85.03%	85.03%	71.26%	70.81%	70.93%	85.04
SGDClassifier	84.77%	84.96%	85.03%	71.75%	70.87%	70.17%	
KNeighborsClassifier	85.19%	85.46%	85.31%	72.51%	73.62%	73.68%	
GaussianNB	80.64%	80.15%	80.15%	69.16%	70.10%	69.88%	
MultinomialNB	84.63%	84.63%	--	70.60%	70.64%	--	
BernoulliNB	82.26%	84.64%	82.17%	68.45%	69.84%	68.54%	
DecisionTreeClassifier	85.87%	86.01%	86.01%	74.94%	77.32%	78.12%	86.3
RandomForestClassifier	85.88%	86.17%	86.13%	75.84%	79.66%	80.32%	86.3
GradientBoostingClassifier	85.93%	86.12%	86.12%	80.69%	81.47%	80.32%	
MLPClassifier	85.89%	86.09%	86.10%	73.00%	75.57%	74.79%	
DNN	EPOCH:15->86% .0.865809559822	EPOCH:6->86% 0.869071900844	0.8764449358	0.7623839974	0.7918781638	0.7857	
Xgboost	85.85%	86.02%	86.02	83.70%	90.85%	91.08%	89.6



# 特徵值比較



由特徵值可以看到Region\_code, Gender, Channel\_Code比較不重要

所以將這三個欄位刪除

## 模型比較

	資料離散化	連續資料標準化	資料離散化 SMOTE	連續資料離散化 SMOTE
XGBoost	85.85%	86.02%	83.70%	<u>89.60%</u>
Randomforest	85.88%	86.17%	75.84%	79.66%
Decision Tree	85.87%	86.01%	74.94%	77.32%
CatBoost	85.80%	86.12%	無法做出	無法做出
Logistic Regression	85.04%	85.03%	71.26%	70.81%

**XGBoost**預測的預測準確度最高，所以我們採用此方法



Linebot實現

1 資料處理

1 建立模型

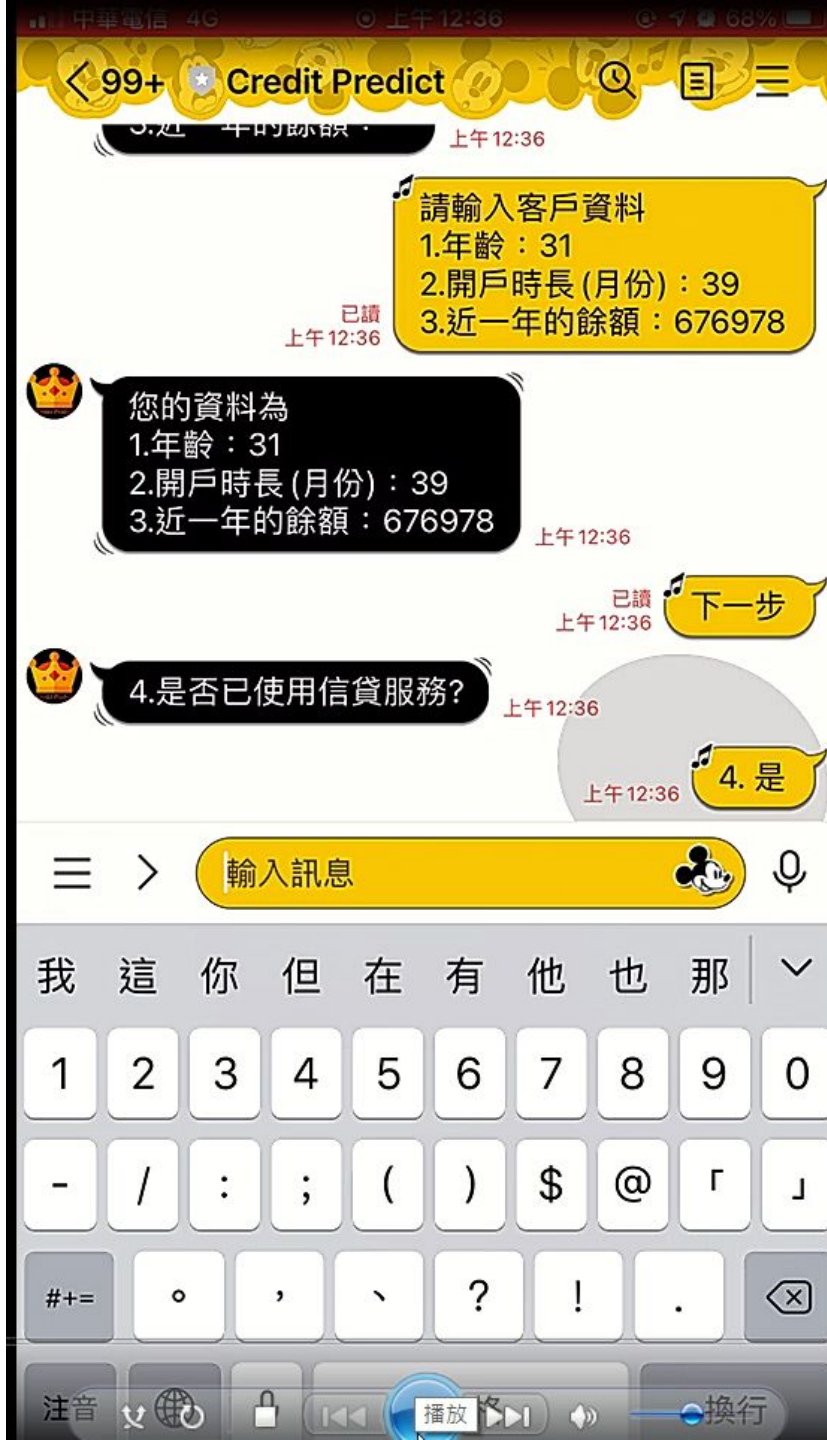
3 DEMO

4 總結

## Linebot實現



## Linebot實現





結語

1 資料處理

2 建立模型

3 Demo

4 總結



## User Experience Innovation Design

1. 使用者研究
2. 概念發想
3. 情境模擬
4. 原型製作
5. 策略藍圖

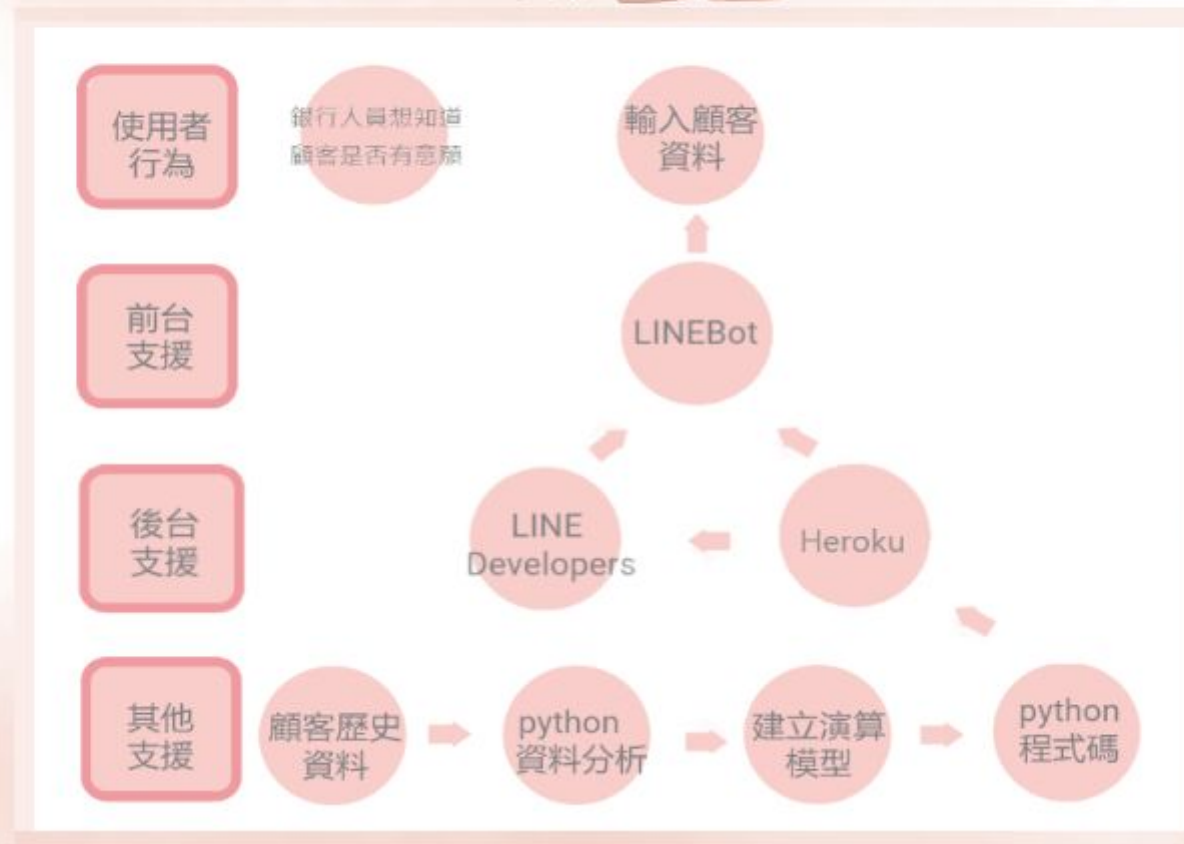
## KJ Method



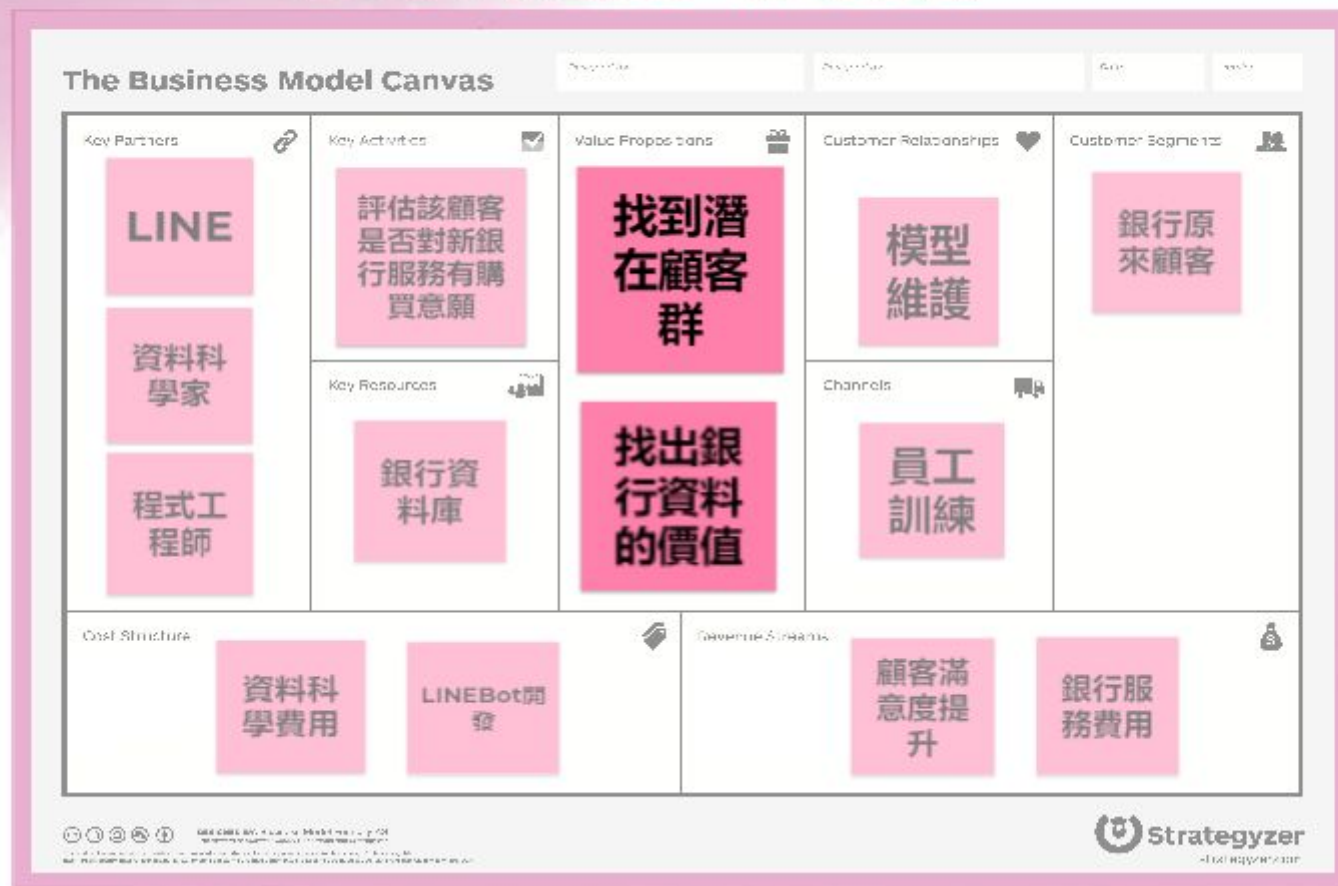
## 概念發想



## 服務藍圖



# Business model



Q&A

*Thank you for  
watching*