網路資料科技應用期末報告

信用卡用戶流失預測

日四B 06122247 吳英緩

目錄

01 研究方法

01-1. 研究動機
01-2. 資料來源
01-3. 研究流程

➢01-1. 研究動機

在信用卡氾濫的現在，多樣的選擇下，你有沒
有苦惱過要不要申辦？
又或者已經報廢幾張信用卡了呢？

阻止客戶離開之前，你需要知道，
誰將離開，什麼時候離開，為什麼離開

➢01-1. 研究動機

▲ 分析客戶流失特徵及客戶流失預測

▲ 探討此公司可以如何訂定對策「防止客戶流失」

➢01-2.資料來源

- ▲ 資料來源：

  kaggle上的公開數據集

  「Credit Card customers」

- ▲ 來源網站：

  Leaps.analyttica.com

- ▲ 資料集名稱：

  BankChurners.csv

  總共有21個欄位

欄位名稱	欄位內容
CLIENTNUM	持有該帳戶的用戶唯一識別編號
Attrition_Flag	客戶是否續約信用卡服務
Customer_Age	用戶年齡
Gender	用戶性別
Dependent_count	家庭人數
Education_Level	教育水平
Marital_Status	婚姻狀況
Income_Category	收入分布
Card_Category	持有卡片類型，分為藍、銀、金、鉑金卡
Months_on_book	每月帳單數量
Total_Relationship_Count	用戶持有銀行業務數量
Months_Inactive_12_mon	用戶不活躍月份數
Contacts_Count_12_mon	過去一年中與銀行接觸次數
Credit_Limit	信用卡額度
Total_Revolving_Bal	循環信用總量
Avg_Open_To_Buy	過去一年平均開放的信貸限額
Total_Amt_Chng_Q4_Q1	第四季度與第一季度間交易金額變化
Total_Trans_Amt	過去一年總交易金額
Total_Trans_Ct	過去一年總交易次數
Total_Ct_Chng_Q4_Q1	第四季度與第一季度間交易數量變化
Avg_Utilization_Ratio	平均卡片利用率

➢01-3.研究流程

02

數　據　分　析

02-1. 目標變數分析

02-2. 特徵變數分析

➢02. 數據分析—匯入套件

▲ 匯入套件

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as ex
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import plotly.offline as pyo
pyo.init_notebook_mode()
sns.set_style('darkgrid')
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score as f1
from sklearn.metrics import confusion_matrix
import scikitplot as skplt
plt.rc('figure',figsize=(18,9))
%pip install imbalanced-learn
from imblearn.over_sampling import SMOTE
```

▲ 事先安裝套件

```
pip install xgboost
pip install delayed
pip install scikit-plot
pip install examples
pip install hiredis
pip install redis
pip install plotly
pip install imbalanced-learn
```

➢02. 數據分析—匯入資料

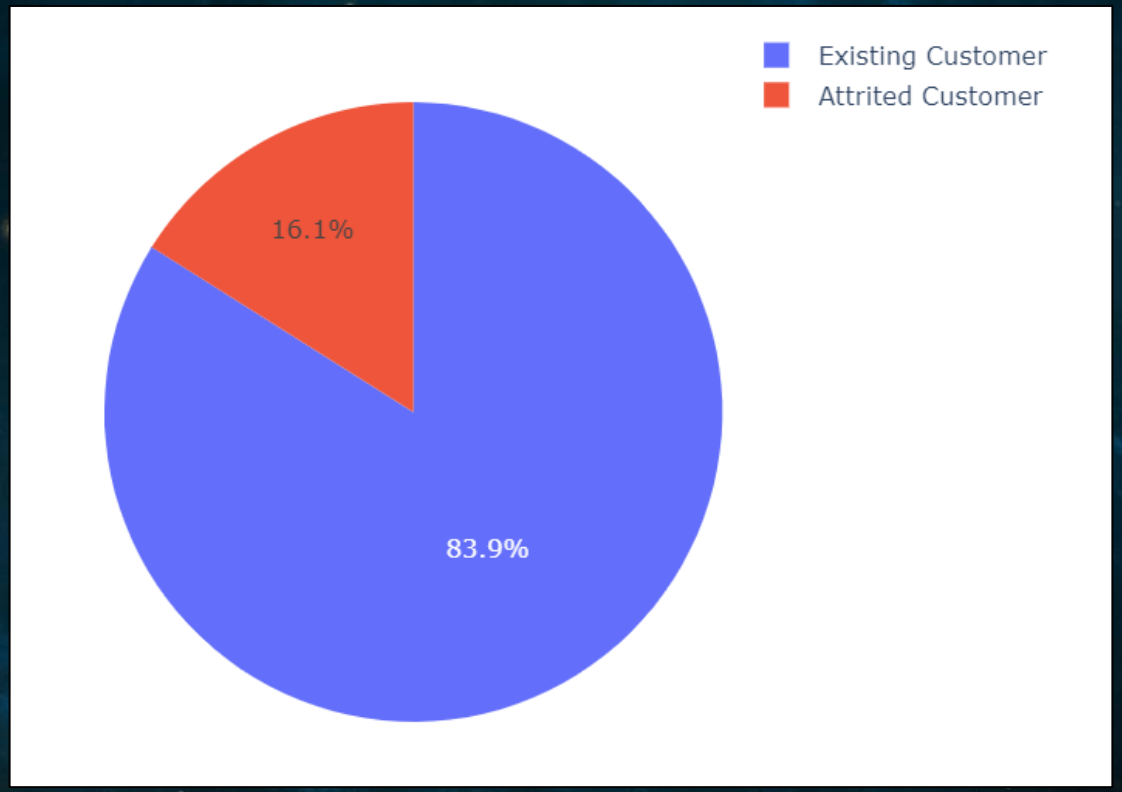
```
c_data = pd.read_csv('BankChurners.csv')
c_data = c_data[c_data.columns[:-2]]
c_data
```

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_bo
0	768805383	Existing Customer	45	M	3	High School	Married	$60K-80K$	Blue	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K-120K$	Blue	
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K-80K$	Blue	
...	...	...	...	...	...	...	...	...	...	
10122	772366833	Existing Customer	50	M	2	Graduate	Single	$40K-60K$	Blue	
10123	710638233	Attrited Customer	41	M	2	Unknown	Divorced	$40K-60K$	Blue	
10124	716506083	Attrited Customer	44	F	1	High School	Married	Less than $40K	Blue	
10125	717406983	Attrited Customer	30	M	2	Graduate	Unknown	$40K-60K$	Blue	
10126	714337233	Attrited Customer	43	F	2	Graduate	Married	Less than $40K	Silver	

10127 rows × 21 columns

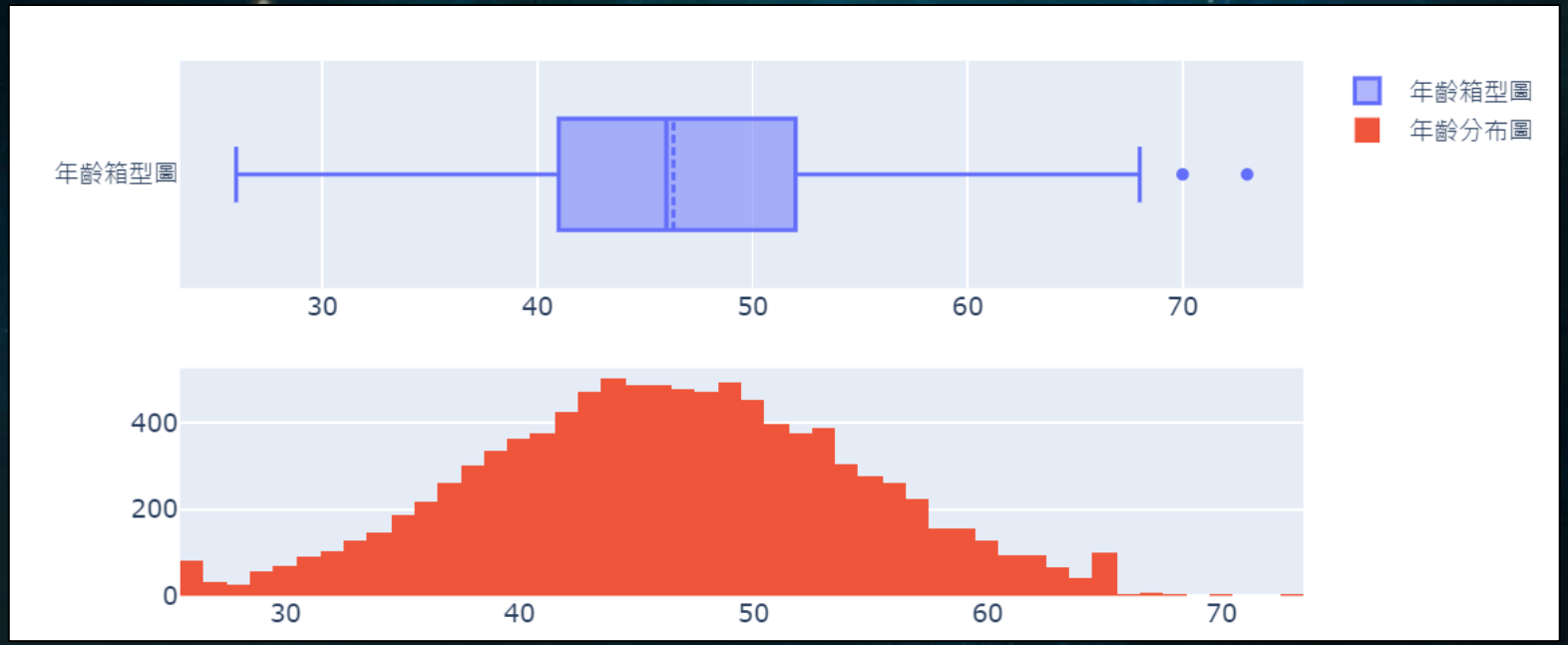
➢02-1.目標變數分析— Attrition_Flag

```
fig=ex.pie(c_data,names='Attrition_Flag',title='信用卡客戶流失比率',width=550, height=500)
fig.show()
```



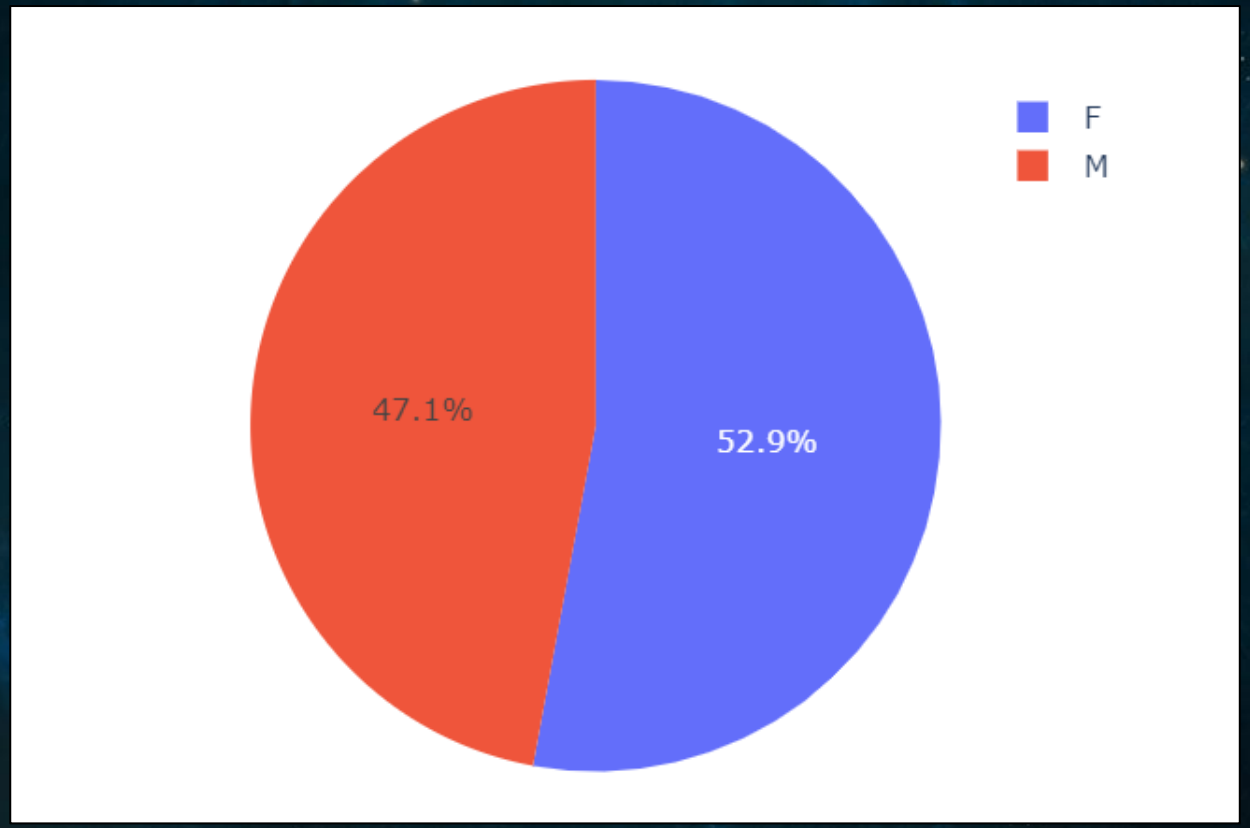
➢02-2.特徵變數分析—Customer_Age

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Customer_Age'],name='年齡箱型圖',boxmean=True)
tr2=go.Histogram(x=c_data['Customer_Age'],name='年齡分布圖')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=438, width=750, title_text="用戶的年齡分布")
fig.show()
```



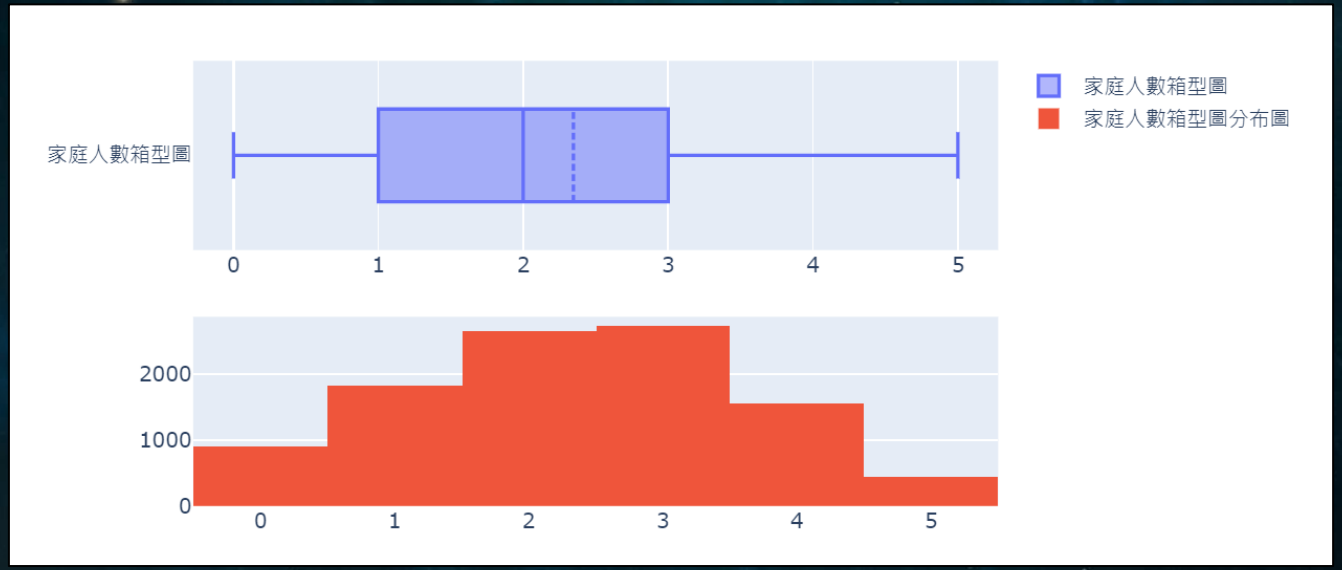
➢02-2.特徵變數分析— Gender

```
fig=ex.pie(c_data,names='Gender',title='用戶性別比',width=450, height=450)
fig.show()
```



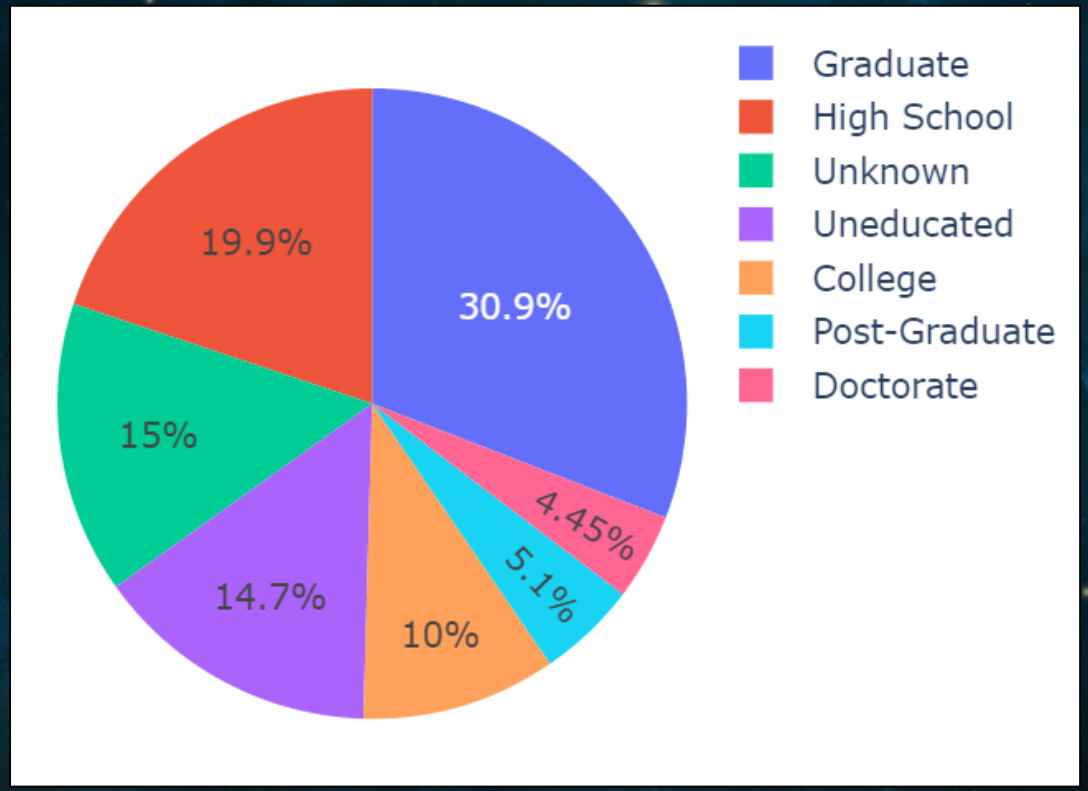
➢02-2.特徵變數分析—Dependent_count

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Dependent_count'],name='家庭人數箱型圖',boxmean=True)
tr2=go.Histogram(x=c_data['Dependent_count'],name='家庭人數箱型圖分布圖')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=438, width=750, title_text="家庭人數")
fig.show()
```



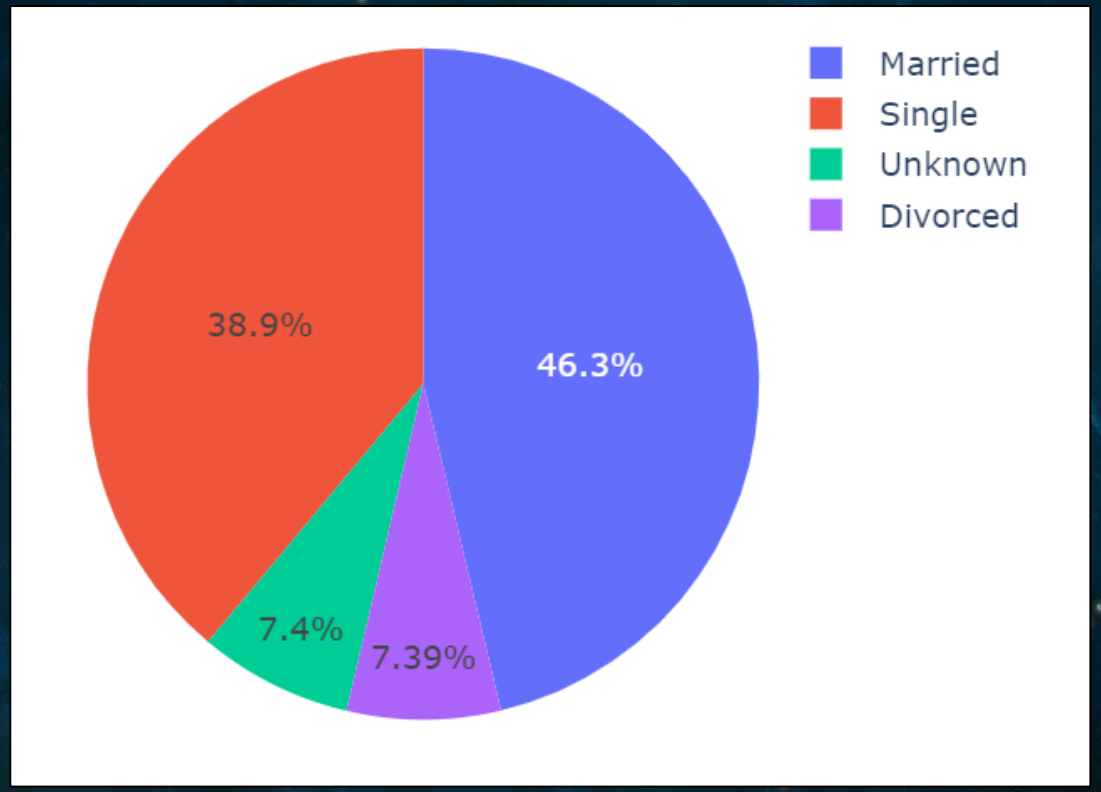
➢02-2.特徵變數分析—Education_Level

```
fig=ex.pie(c_data,names='Education_Level',title='教育程度',width=450, height=450)
fig.show()
```



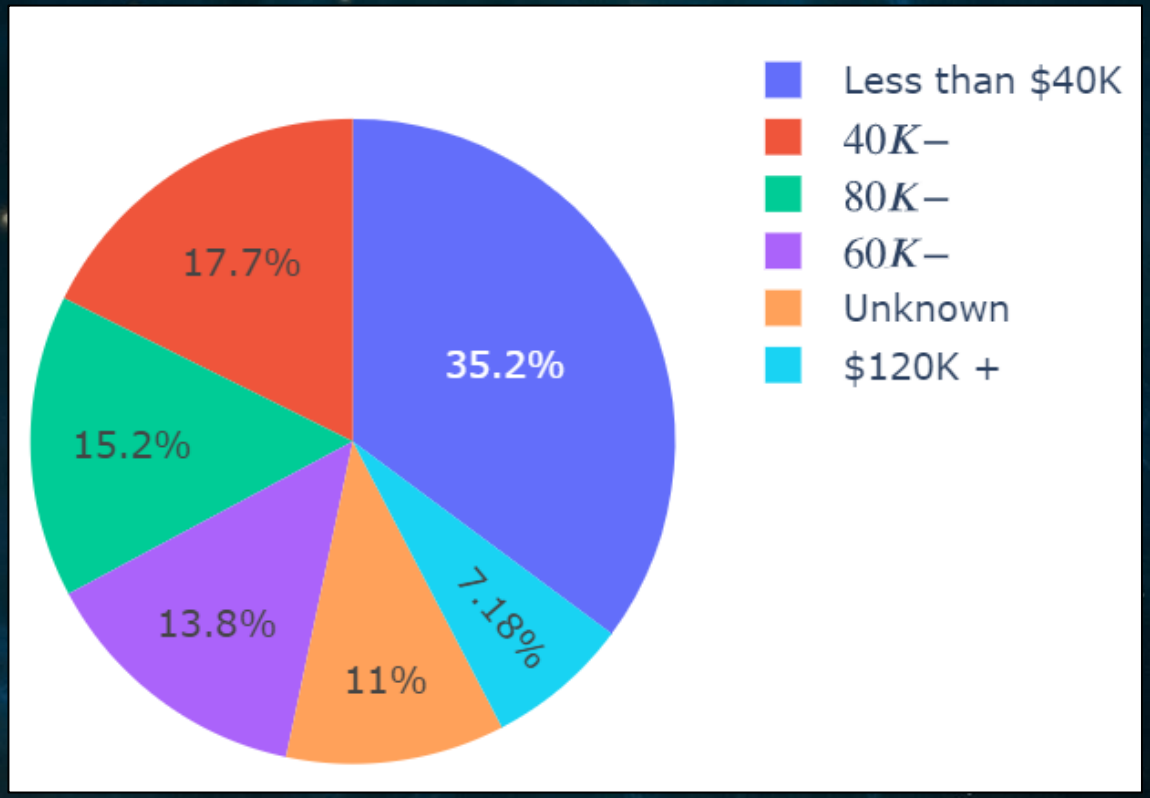
➢02-2.特徵變數分析—Education_Level

```
fig=ex.pie(c_data,names='Marital_Status',title='婚姻狀態',width=450, height=450)
fig.show()
```



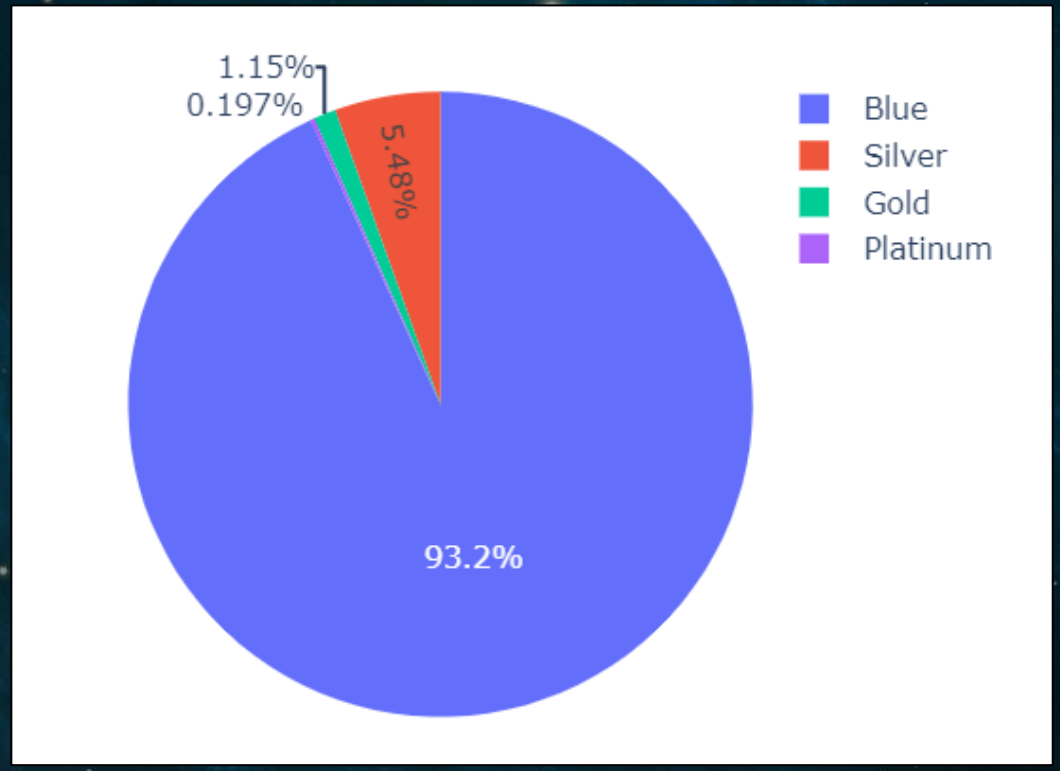
➢02-2.特徵變數分析—Income_Category

```
fig=ex.pie(c_data,names='Income_Category',title='收入狀況',width=450, height=450)
fig.show()
```



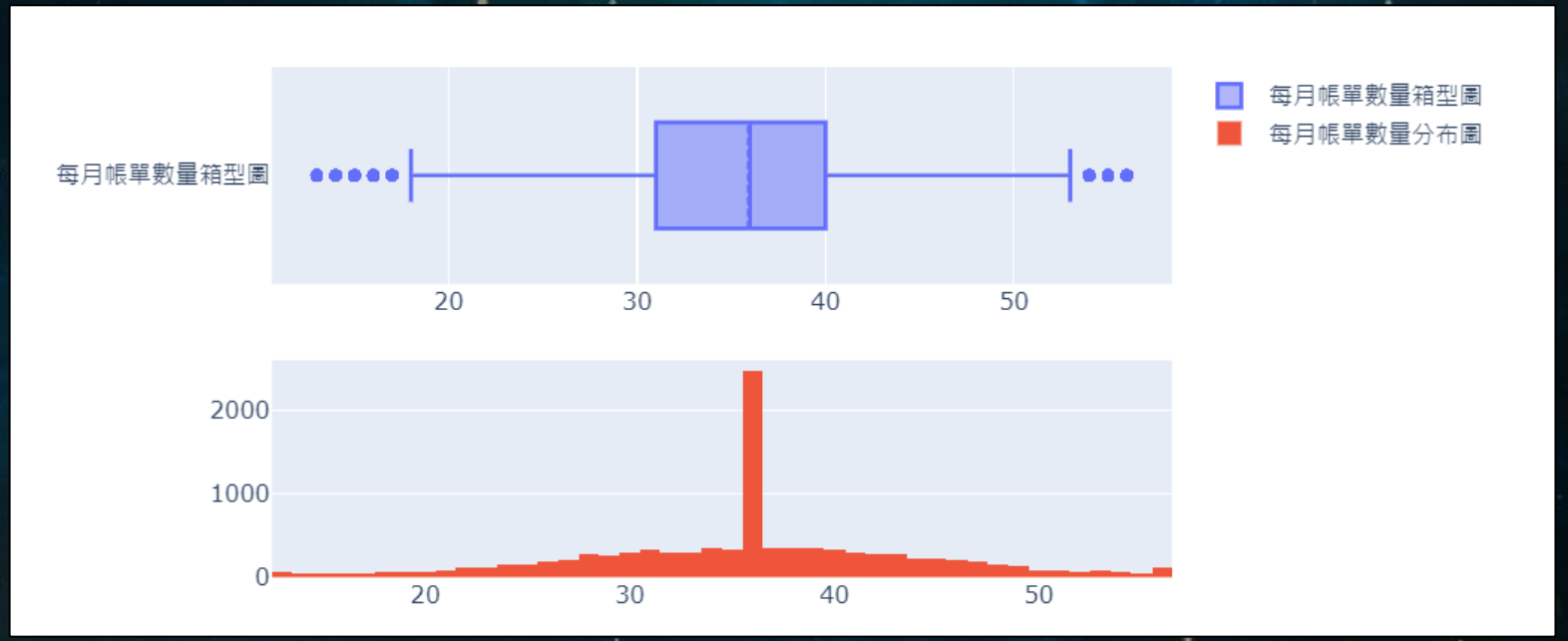
➢02-2.特徵變數分析—Card_Category

```
fig=ex.pie(c_data,names='Card_Category',title='卡片等級',width=450, height=450)
fig.show()
```



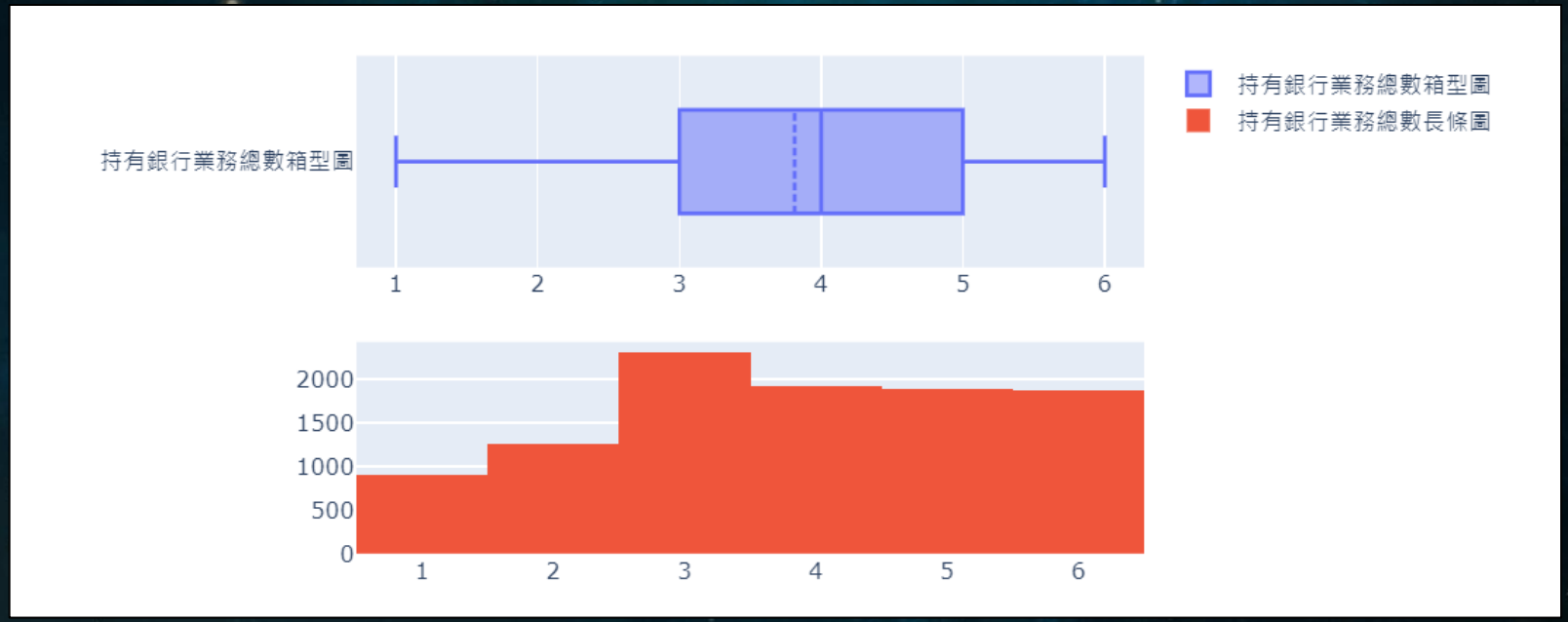
02-2.特徵變數分析—Months_on_book

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Months_on_book'],name='每月帳單數量箱型圖',boxmean=True)
tr2=go.Histogram(x=c_data['Months_on_book'],name='每月帳單數量分布圖')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=438, width=750, title_text="每月帳單數量")
fig.show()
```



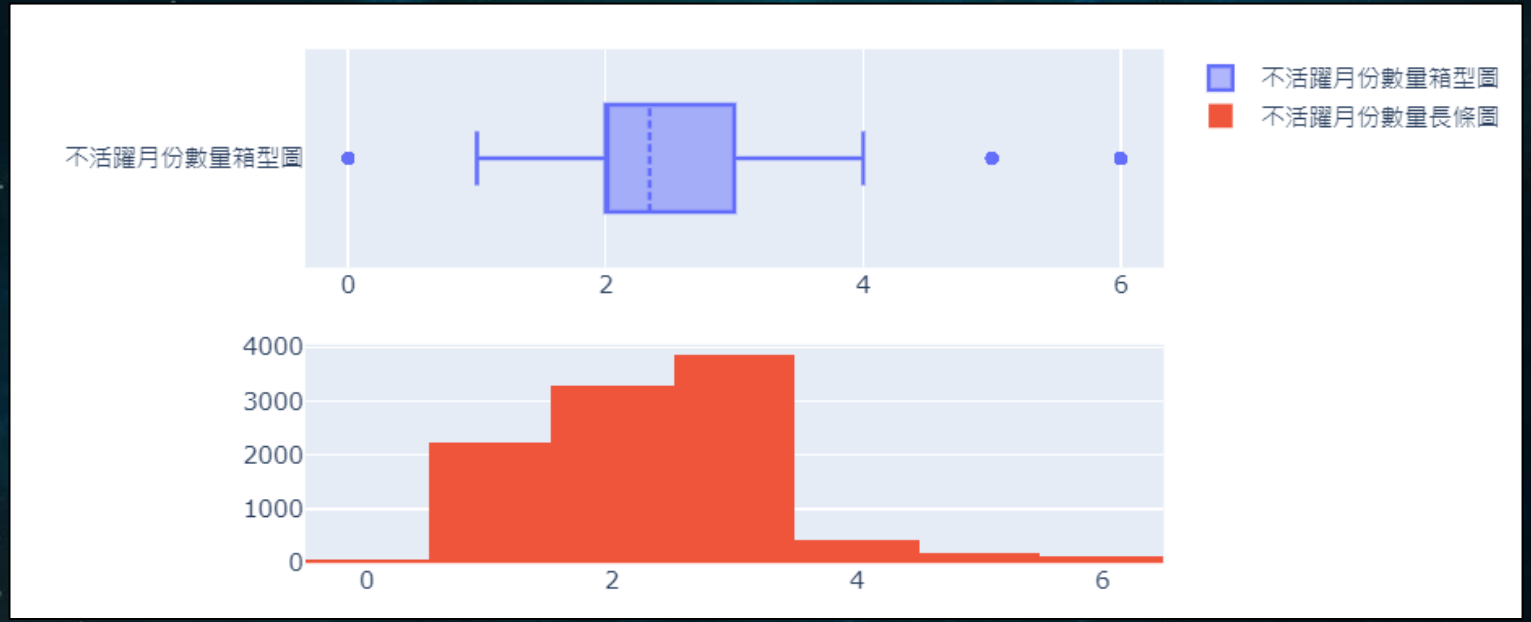
➢02-2.特徵變數分析—Total_Relationship_Count

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Total_Relationship_Count'],name='持有銀行業務總數箱型圖',boxmean=True)
tr2=go.Histogram(x=c_data['Total_Relationship_Count'],name='持有銀行業務總數長條圖')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=438, width=750, title_text="持有銀行業務總數")
fig.show()
```



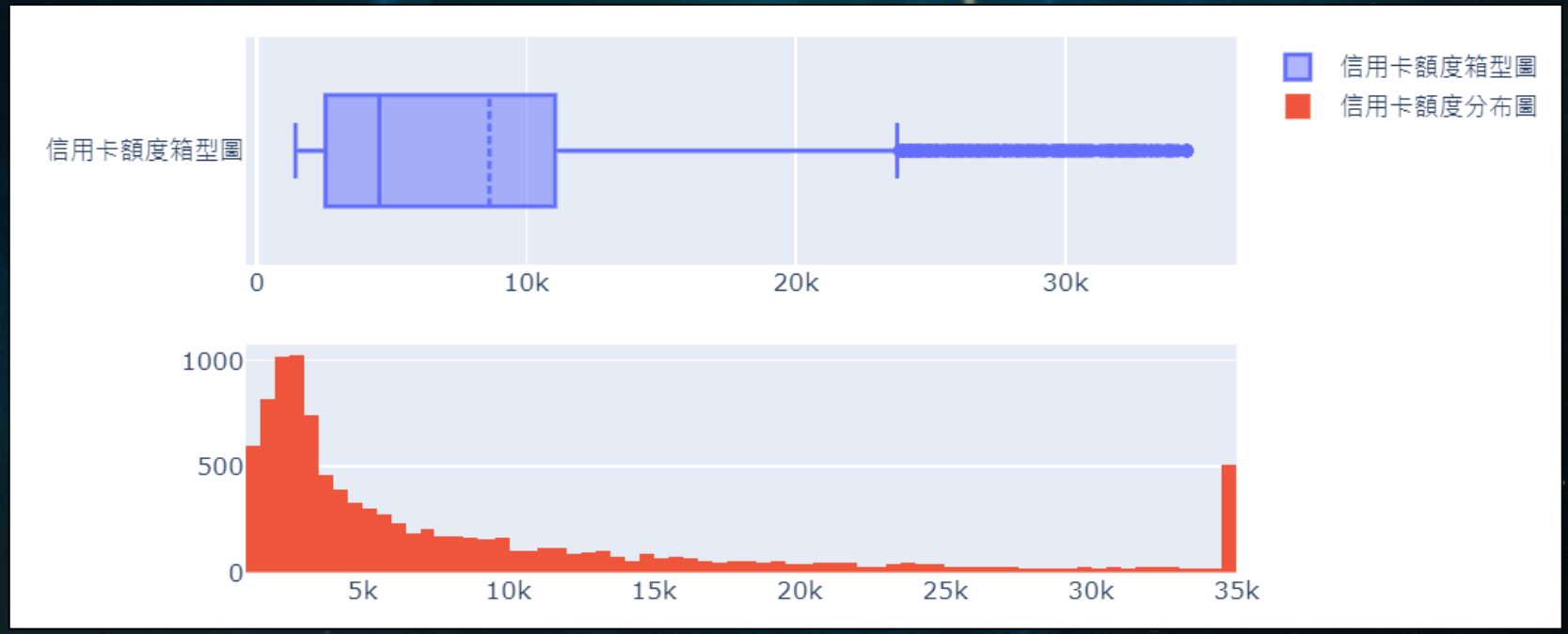
➢02-2.特徵變數分析—Months_Inactive_12_mon

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Months_Inactive_12_mon'],name='不活躍月份數量箱型圖',boxmean=True)
tr2=go.Histogram(x=c_data['Months_Inactive_12_mon'],name='不活躍月份數量長條圖')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=438, width=750, title_text="不活躍月份數量")
fig.show()
```



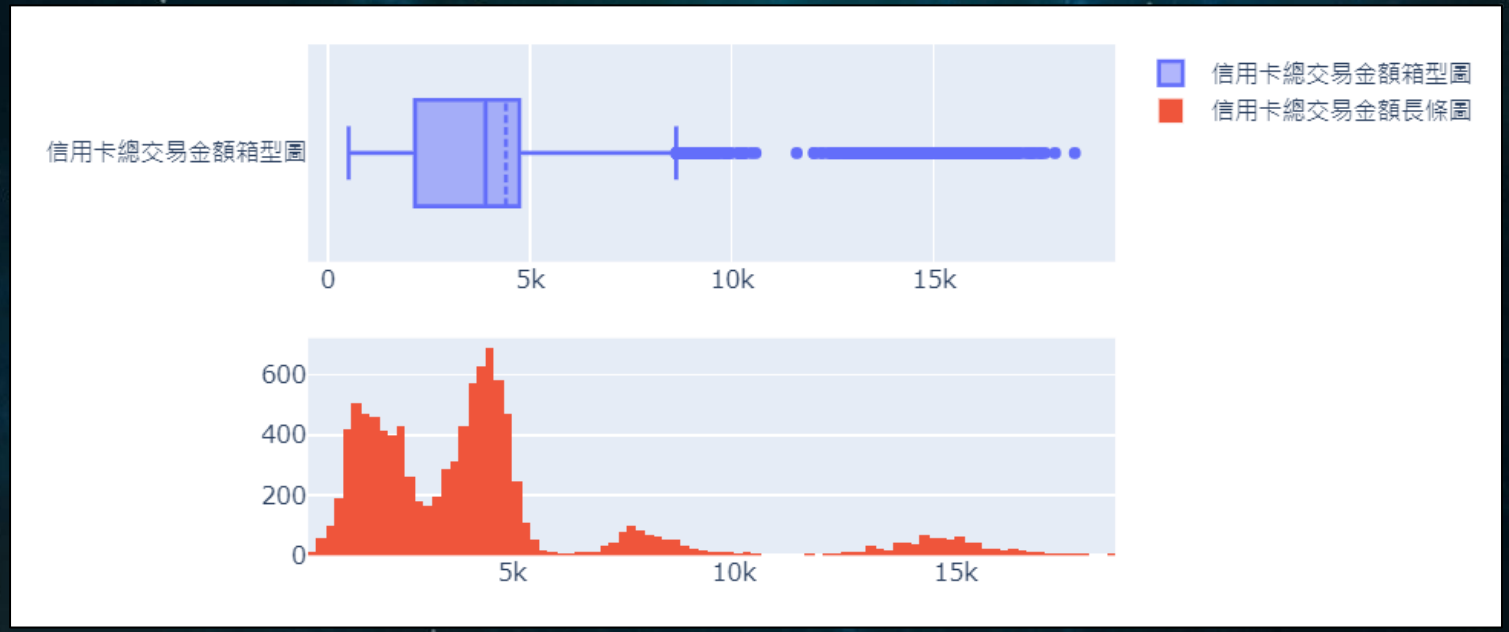
➢02-2.特徵變數分析—Credit_Limit

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Credit_Limit'],name='信用卡額度箱型圖',boxmean=True)
tr2=go.Histogram(x=c_data['Credit_Limit'],name='信用卡額度分布圖')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=438, width=750, title_text="信用卡額度")
fig.show()
```



➢02-2.特徵變數分析—Total_Trans_Amt

```
fig = make_subplots(rows=2, cols=1)

tr1=go.Box(x=c_data['Total_Trans_Amt'],name='信用卡總交易金額箱型圖',boxmean=True)
tr2=go.Histogram(x=c_data['Total_Trans_Amt'],name='信用卡總交易金額長條圖')

fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)

fig.update_layout(height=438, width=750, title_text="信用卡總交易金額")
fig.show()
```



03

預　測　模　型

03-1. 資料預處理
03-2. 主成分分析
03-3. 建置模型
03-4. 模型分析

➢03-1. 資料預處理 — 類別型資料轉換

```
c_data
```

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_or
0	768805383	1	45	1	3	3	1	2	0	
1	818770008	1	49	0	5	2	2	4	0	
2	713982108	1	51	1	3	2	1	3	0	
3	769911858	1	40	0	4	3	3	4	0	
4	709106358	1	40	1	3	5	1	2	0	
...	...	...	...	...	...	...	...	...	...	...
10122	772366833	1	50	1	2	2	2	1	0	
10123	710638233	0	41	1	2	6	0	1	0	
10124	716506083	0	44	0	1	3	1	4	0	
10125	717406983	0	30	1	2	2	3	1	0	
10126	714337233	0	43	0	2	2	1	4	3	

10127 rows × 21 columns

➢03-1. 資料預處理 — 分割資料集

```
c_data
```

y

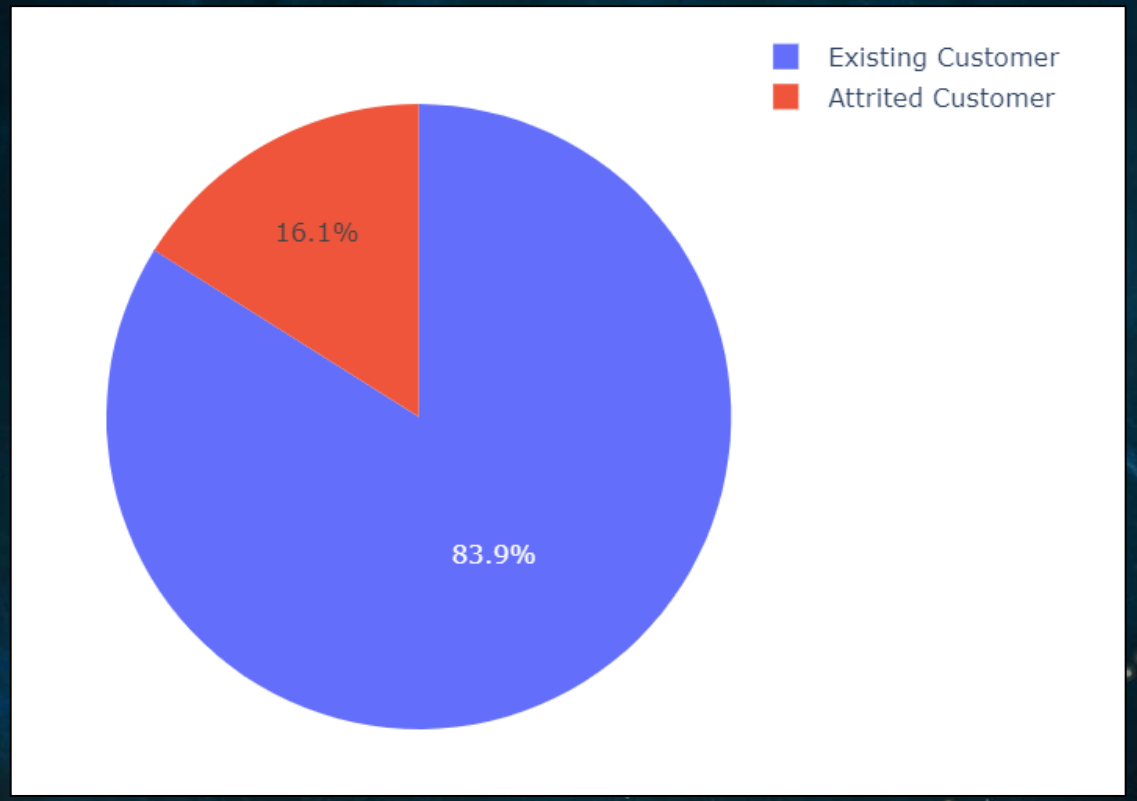
x

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_or
0	768805383	1	45	1	3	3	1	2	0	
1	818770008	1	49	0	5	2	2	4	0	
2	713982108	1	51	1	3	2	1	3	0	
3	769911858	1	40	0	4	3	3	4	0	
4	709106358	1	40	1	3	5	1	2	0	
...	...	...	...	...	...	...	...	...	...	
10122	772366833	1	50	1	2	2	2	1	0	
10123	710638233	0	41	1	2	6	0	1	0	
10124	716506083	0	44	0	1	3	1	4	0	
10125	717406983	0	30	1	2	2	3	1	0	
10126	714337233	0	43	0	2	2	1	4	3	

10127 rows × 21 columns

➢03-1. 資料預處理 — 處理失衡資料

▲ 續約用戶 : 未續約用戶 = 83.9 : 16.1



➢03-1. 資料預處理 — 處理失衡資料

- SMOTE（Synthetic Minority Oversampling Technique）— 樣本合成方法

```
from imblearn.over_sampling import SMOTE

print("Before OverSampling, counts of label '1': {}".format(sum(y==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y==0)))

sm = SMOTE()
x_train, y_train = sm.fit_sample(x_train, y_train.ravel())

print("After OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train==0)))
```

```
Before OverSampling, counts of label '1': 8500
Before OverSampling, counts of label '0': 1627

After OverSampling, counts of label '1': 5917
After OverSampling, counts of label '0': 5917
```

➢03-1. 資料預處理 — 資料標準化

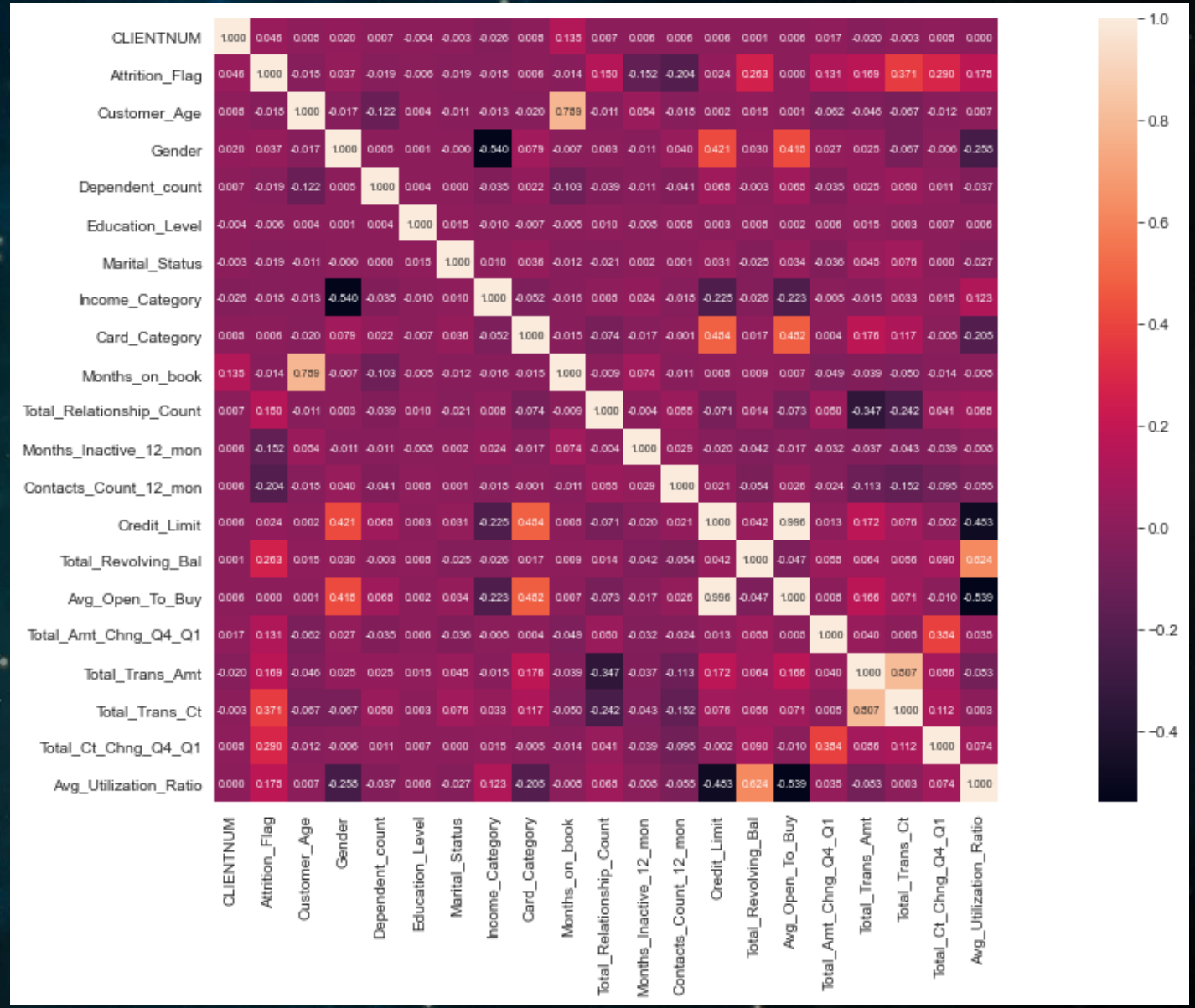
- 為了提升模型的收斂速度與精準度，將資料標準化

```
#標準化
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(x_train)
X_train_std = sc.transform(x_train)
X_test_std = sc.transform(x_test)
```

➢03-2. 主成分分析— 資料相關性

▲ 熱度圖

```
#熱度圖
cols = c_data.columns
cm = np.corrcoef(c_data[cols].values.T)
#sns.set(font_scale=1.5)
hm = sns.heatmap(cm,cbar = True, annot = True,
                 square = True, fmt='.3f',
                 annot_kws={'size':7},
                 yticklabels=cols,
                 xticklabels=cols)
plt.tight_layout()
plt.savefig('correlation.png',dpi=300)
plt.show()
```



➢03-2. 主成分分析— 資料相關性

- 取絕對值查看各特徵變數與目標變數之相關係數，並由大至小排序。

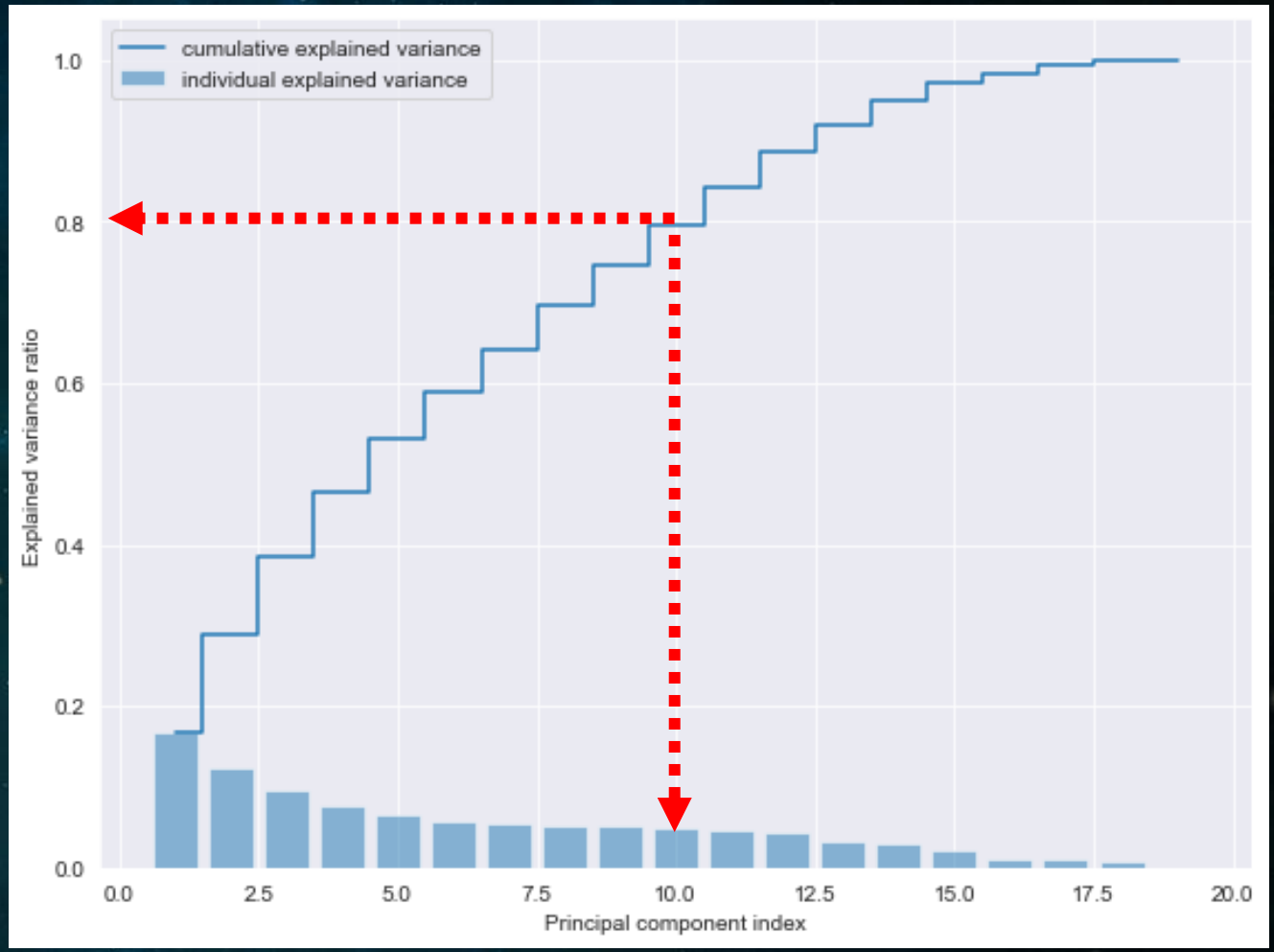
```
#查看相關係數
most_correlated = c_data.corr().abs()['Attrition_Flag'].sort_values(ascending=False)
most_correlated = most_correlated[:15]
print(most_correlated)
```

```
Attrition_Flag              1.000000
Total_Trans_Ct              0.371403
Total_Ct_Chng_Q4_Q1         0.290054
Total_Revolving_Bal         0.263053
Contacts_Count_12_mon       0.204491
Avg_Utilization_Ratio       0.178410
Total_Trans_Amt             0.168598
Months_Inactive_12_mon      0.152449
Total_Relationship_Count    0.150005
Total_Amt_Chng_Q4_Q1        0.131063
CLIENTNUM                   0.046430
Gender                      0.037272
Credit_Limit                0.023873
Dependent_count             0.018991
Marital_Status              0.018597
Name: Attrition_Flag, dtype: float64
```

➢03-2. 主成分分析—疊加變異解釋率

- 主成分依序保留1~20特徵之疊加變異解釋率。

```
plt.figure(figsize=(8,6))
pca = PCA(n_components = None)
X_train_pca = pca.fit_transform(X_train_std)
var_ratio = pca.explained_variance_ratio_
cum_var_ratio = np.cumsum(var_ratio)

plt.bar(range(1,20),var_ratio,alpha=0.5,
       label = "individual explained variance")
plt.step(range(1,20),cum_var_ratio,where="mid",
        label = "cumulative explained variance")
plt.ylabel("Explained variance ratio")
plt.xlabel("Principal component index")
plt.legend(loc="best")
plt.tight_layout()
plt.show
```



- 主成分保留10維，建置模型

```
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
```

➢03-3. 建置模型—模型選擇

▲ 選擇常見的分類模型：以下皆為監督式學習

演算法模型	假設	優點	缺點
樸素貝葉斯	自變數需獨立	當自變數是類別型時成效好、收斂速度快	自變數有關聯會影響成效
SVM 支持向量機	無	可利用 Kernel 技巧分離非線性的資料	訓練時間長、需決定模型的核心方法(Kernal)
✓Logistic Regression	各個特徵皆獨立、為二類評定模型	分類時計算量小、速度快、存儲資源低	只能處理兩分類問題
決策樹（分類與回歸樹）	無	解釋性高，可同時處理類別+數值型資料	容易過擬合
✓ RandomForest	無	採用多個決策樹的投票機制來改善決策樹	可能有很多相似的決策樹，掩蓋了真實的結果
✓ XGBoost	無	精度、靈活性高、正則化項、支持並行化	調參複雜，不適合處理超高維資料

➢03-3. 建置模型—Logistic Regression

- Logistic Regression 模型之預測正確率、F1-score和混淆矩陣

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train_pca,y_train)
print("train accuracy:%.3f" % lr.score(X_train_pca,y_train))
print("test accuracy:%.3f" % lr.score(X_test_pca,y_test))
```

```
train accuracy:0.796
test accuracy:0.808
```

```
from sklearn.metrics import precision_score,recall_score,f1_score
y_pred = lr.predict(X_test_pca)
print("Precision: %.3f" % precision_score(y_true=y_test,y_pred=y_pred))
print("Recall: %.3f" % recall_score(y_true=y_test,y_pred=y_pred))
print("F1: %.3f" % f1_score(y_true=y_test,y_pred=y_pred))
```

```
Precision: 0.941
Recall: 0.822
F1: 0.878
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true=y_test, y_pred=y_pred)
```

```
array([[ 357,  131],
       [ 453, 2098]], dtype=int64)
```

訓練集預測正確率	測試集預測正確率
79.6%	80.8%

Precision	Recall	F1-score
94.1%	82.2%	87.8%

	1	0
1	357	131
0	453	2098

➢03-3. 建置模型—RandomForest

- RandomForest模型之預測正確率、F1-score和混淆矩陣

```
#隨機森林模型
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(criterion='entropy',
                                n_estimators=10,
                                random_state=1)
forest.fit(X_train_pca,y_train)
print('train_score:',forest.score(X_train_pca,y_train))#forest.score正確率
print('test_score:',forest.score(X_test_pca,y_test))#forest.score正確率
```

```
train_score: 0.9957976130442091
test_score: 0.8604804211911813
```

```
y_pred = forest.predict(X_test_pca)
print("Precision: %.3f" % precision_score(y_true=y_test,y_pred=y_pred))
print("Recall: %.3f" % recall_score(y_true=y_test,y_pred=y_pred))
print("F1: %.3f" % f1_score(y_true=y_test,y_pred=y_pred))
```

```
Precision: 0.948
Recall: 0.882
F1: 0.914
```

```
confusion_matrix(y_true=y_test, y_pred=y_pred)
```

```
array([[ 365,  123],
       [ 301, 2250]], dtype=int64)
```

訓練集預測正確率	測試集預測正確率
99.5%	86%

Precision	Recall	F1-score
94.8%	88.2%	91.4%

	1	0
1	365	123
0	301	2250

➢03-3. 建置模型—RandomForest

- RandomForest模型之決策樹數量參數

```
#隨機森林模型
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(criterion='entropy',
                                n_estimators=10,
                                random_state=1)
forest.fit(X_train_pca,y_train)
print('train_score:',forest.score(X_train_pca,y_train))#forest.score正確率
print('test_score:',forest.score(X_test_pca,y_test))#forest.score正確率
```

```
train_score: 0.9957976130442091
test_score: 0.8604804211911813
```

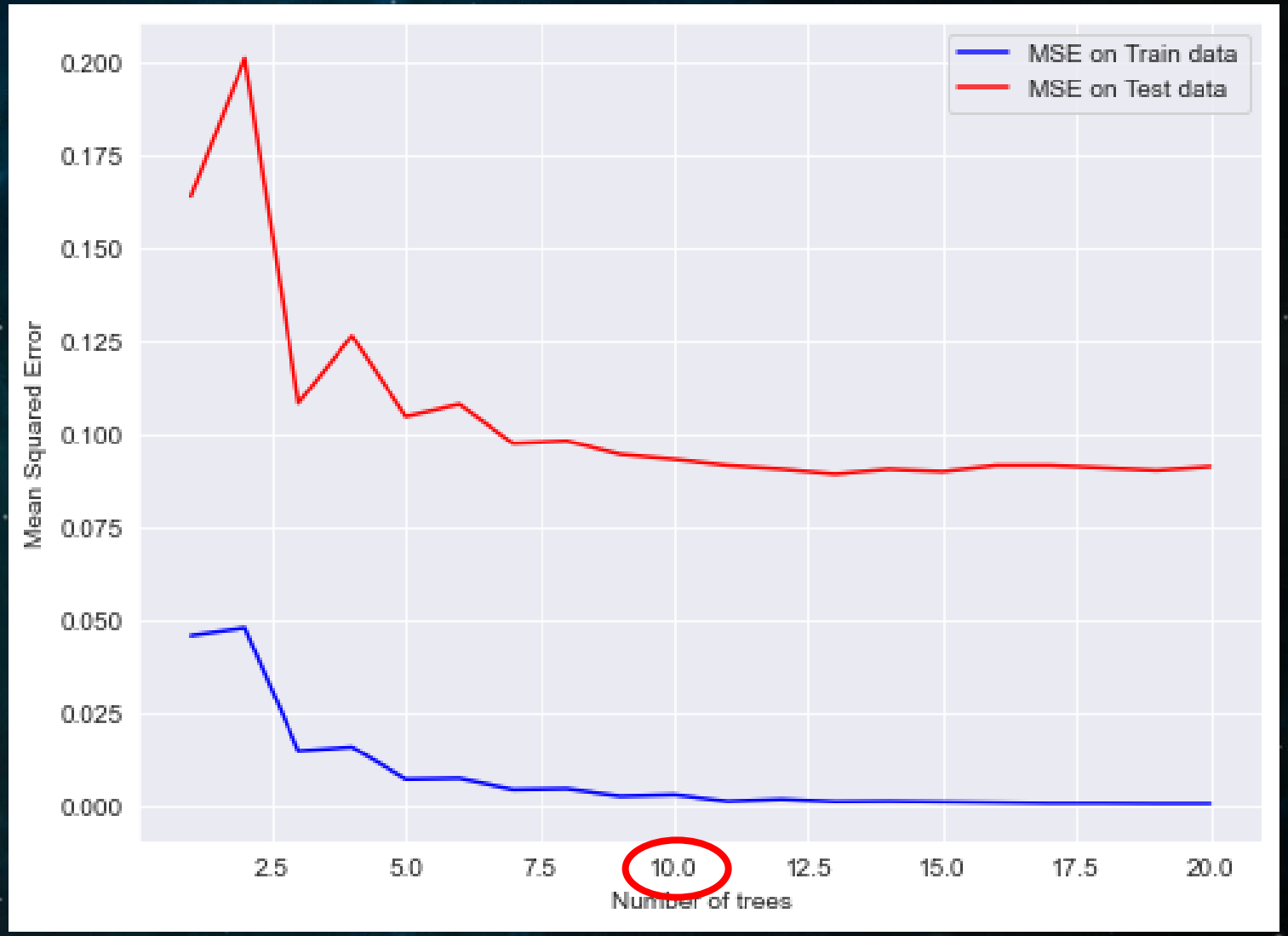
```
rf = RandomForestClassifier(criterion='entropy',
                            n_estimators=1,
                            random_state=1)
from sklearn.metrics import mean_squared_error
trees, train_loss, test_loss = [], [], []
for iter in range(20):
    rf.fit(X_train_pca, y_train)
    y_train_predicted = rf.predict(X_train_pca)
    y_test_predicted = rf.predict(X_test_pca)
    mse_train = mean_squared_error(y_train, y_train_predicted)
    mse_test = mean_squared_error(y_test, y_test_predicted)
    print("Iteration: {} Train mse: {} Test mse: {}".format(iter, mse_train, mse_test))
    trees += [rf.n_estimators]
    train_loss += [mse_train]
    test_loss += [mse_test]
    rf.n_estimators += 1
```

```
Iteration: 0 Train mse: 0.04572197007900487 Test mse: 0.16386969397828233
Iteration: 1 Train mse: 0.04782316355690032 Test mse: 0.2010529779532741
Iteration: 2 Train mse: 0.014624306606152295 Test mse: 0.10825929582099375
Iteration: 3 Train mse: 0.015716927214657925 Test mse: 0.1263573543928924
Iteration: 4 Train mse: 0.0070600100857286935 Test mse: 0.10463968410661402
Iteration: 5 Train mse: 0.007312153303076148 Test mse: 0.1079302402105956
Iteration: 6 Train mse: 0.004286434694906707 Test mse: 0.09740046067785456
Iteration: 7 Train mse: 0.0045385779122541605 Test mse: 0.09805857189865087
Iteration: 8 Train mse: 0.0025214321734745334 Test mse: 0.09443896018427114
Iteration: 9 Train mse: 0.0029416708690536224 Test mse: 0.09312273774267851
Iteration: 10 Train mse: 0.0010926206085056312 Test mse: 0.09147745969068773
Iteration: 11 Train mse: 0.0016809547823163557 Test mse: 0.09049029285949325
Iteration: 12 Train mse: 0.0010085728693898135 Test mse: 0.08917407041790062
Iteration: 13 Train mse: 0.0010926206085056312 Test mse: 0.09049029285949325
Iteration: 14 Train mse: 0.0009245251302739956 Test mse: 0.08983218163869694
Iteration: 15 Train mse: 0.0007564296520423601 Test mse: 0.09147745969068773
Iteration: 16 Train mse: 0.0005883341738107245 Test mse: 0.09147745969068773
Iteration: 17 Train mse: 0.0005883341738107245 Test mse: 0.09081934846989141
Iteration: 18 Train mse: 0.0005042864346949068 Test mse: 0.0901612372490951
Iteration: 19 Train mse: 0.0005042864346949068 Test mse: 0.09114840408028957
```

➢03-3. 建置模型—RandomForest

- RandomForest模型
  決策樹數量影響

```
plt.figure(figsize=(8,6))
plt.plot(trees, train_loss, color="blue", label="MSE on Train data")
plt.plot(trees, test_loss, color="red", label="MSE on Test data")
plt.xlabel("Number of trees")
plt.ylabel("Mean Squared Error");
plt.legend()
```



➢03-3. 建置模型—XGBoost

- XGBoost模型之預測正確率、F1-score和混淆矩陣

```
#xgboost模型
import xgboost as xgb
from xgboost import XGBClassifier
xgbc = XGBClassifier()
xgbc.fit(X_train_pca,y_train)
print('train_score:',xgbc.score(X_train_pca,y_train))#xgbc.score正確率
print('test_score:',xgbc.score(X_test_pca,y_test))#xgbc.score正確率
```

```
C:\Users\nachu\anaconda3\lib\site-packages\xgboost\sklearn.py:888: Use

The use of label encoder in XGBClassifier is deprecated and will be re
the following: 1) Pass option use_label_encoder=False when constructin
as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].

[16:06:06] WARNING: C:/Users/Administrator/workspace/xgboost-win64_rel
3.0, the default evaluation metric used with the objective 'binary:log
ly set eval_metric if you'd like to restore the old behavior.
train_score: 0.9927718944360396
test_score: 0.8874629812438302
```

```
y_pred = xgbc.predict(X_test_pca)
print("Precision: %.3f" % precision_score(y_true=y_test,y_pred=y_pred))
print("Recall: %.3f" % recall_score(y_true=y_test,y_pred=y_pred))
print("F1: %.3f" % f1_score(y_true=y_test,y_pred=y_pred))
```

```
Precision: 0.949
Recall: 0.915
F1: 0.932
```

```
confusion_matrix(y_true=y_test, y_pred=y_pred)
```

```
array([[ 362,  126],
       [ 216, 2335]], dtype=int64)
```

訓練集預測正確率	測試集預測正確率
99.2%	88.7%

Precision	Recall	F1-score
94.9%	91.5%	93.2%

	1	0
1	363	126
0	216	2335

➢03-4. 模型分析—3種模型比較

▲ 3種模型之正確率和F1-score的比較

模型名稱	訓練資料集	測試資料集	F1-score
Logistic Regression	79.9%	80.8%	87.8%
RandomForest	99.5%	86%	91.4%
XGBoost	99.2%	88.7%	93.2%

➢03-4. 模型分析—模型重要特徵

▲ 查看模型重要特徵程式碼

```
imp_rfc=forest.feature_importances_
imp_xgbc=xgbc.feature_importances_
names=c_data.columns[1:]
names
```

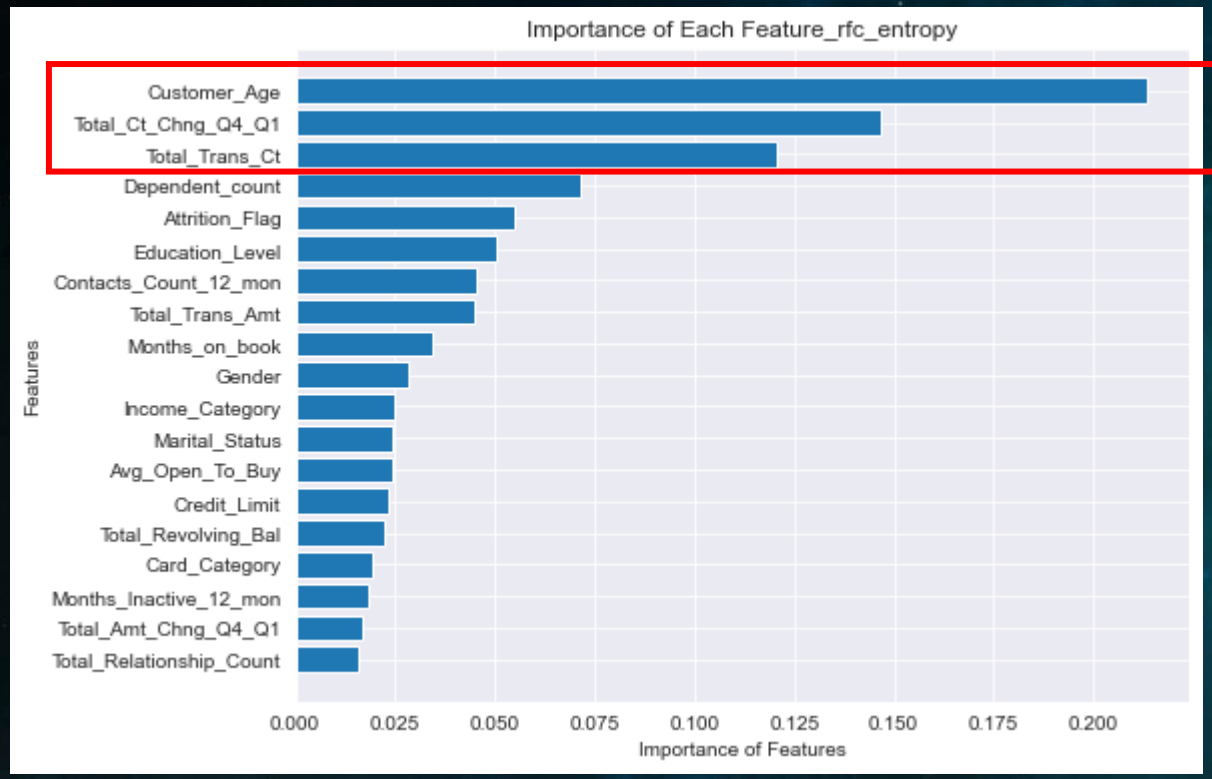
```
Index(['Attrition_Flag', 'Customer_Age', 'Gender', 'Dependent_count',
       'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Category',
       'Months_on_book', 'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'],
      dtype='object')
```

```
plt.figure(figsize=(8,6))
zip(imp_rfc,names)
imp_rfc, names= zip(*sorted(zip(imp_rfc,names)))
plt.barh(range(len(names)),imp_rfc,align='center')
plt.yticks(range(len(names)),names)
plt.xlabel('Importance of Features')
plt.ylabel('Features')
plt.title('Importance of Each Feature_rfc_entropy')
plt.show()
```

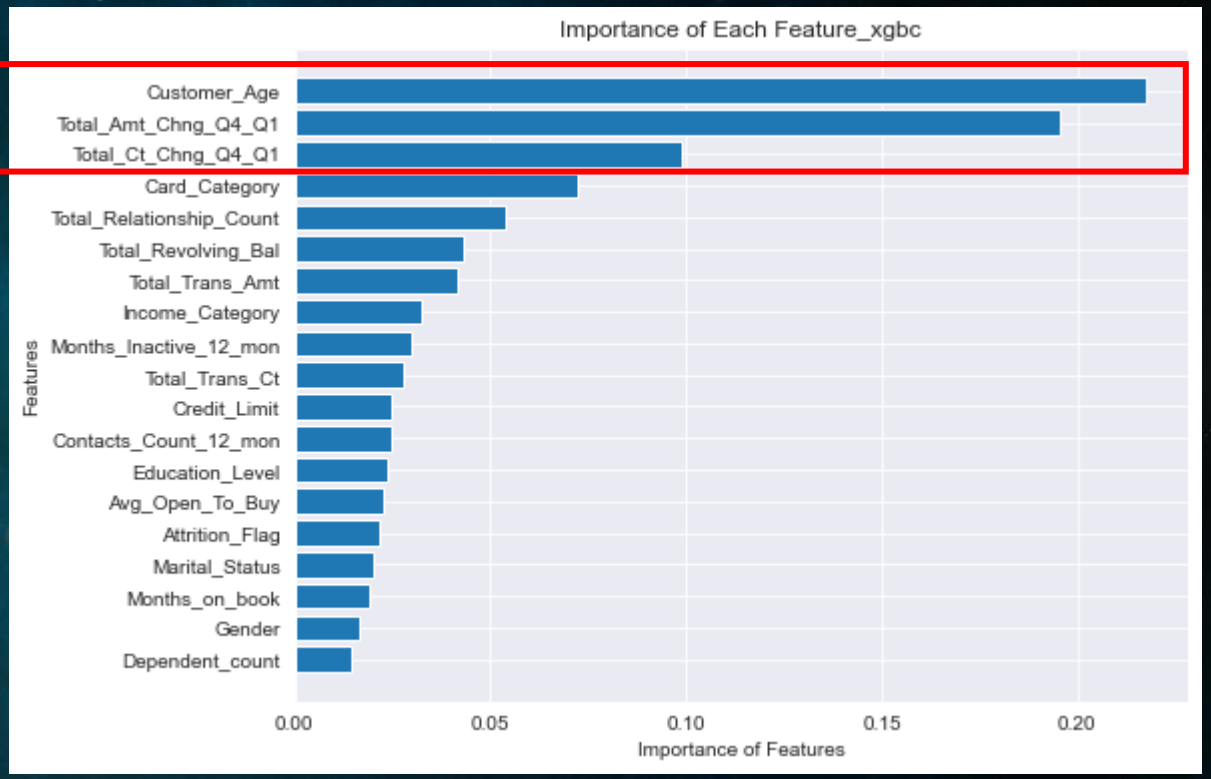
```
plt.figure(figsize=(8,6))
zip(imp_xgbc,names)
imp_xgbc, names= zip(*sorted(zip(imp_xgbc,names)))
plt.barh(range(len(names)),imp_xgbc,align='center')
plt.yticks(range(len(names)),names)
plt.xlabel('Importance of Features')
plt.ylabel('Features')
plt.title('Importance of Each Feature_xgbc')
plt.show()
```

➢03-4. 模型分析—模型重要特徵

▲ RandomForest模型之重要特徵



▲ XGBoost模型之重要特徵

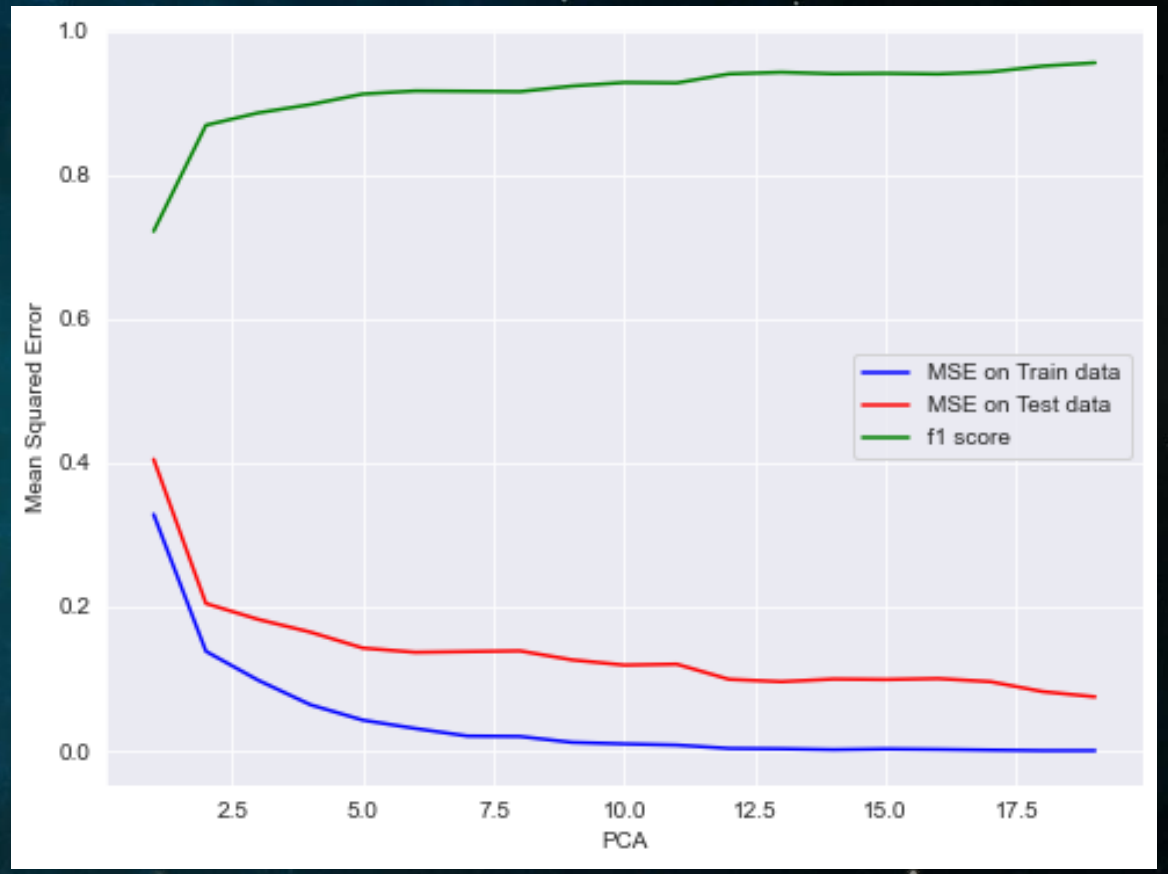


➢03-4. 模型分析—主成分疊代之MSE及F1

▲ XGBoost模型之主成分疊代之MSE及F1-score

```
from sklearn.metrics import mean_squared_error
pca_number, train_loss, test_loss ,f1 = [], [], [],[]
for i in range(1,20):
    pca = PCA(n_components=i)
    X_train_pca = pca.fit_transform(X_train_std)
    X_test_pca = pca.transform(X_test_std)
    xgbc.fit(X_train_pca,y_train)
    y_train_predicted = xgbc.predict(X_train_pca)
    y_test_predicted = xgbc.predict(X_test_pca)
    mse_train = mean_squared_error(y_train, y_train_predicted)
    mse_test = mean_squared_error(y_test, y_test_predicted)
    f1score=f1_score(y_true=y_test,y_pred=y_test_predicted)
    print("pca: {} Train mse: {} Test mse: {} f1: {}".format(i, mse_train, mse_test,f1score))
    pca_number += [i]
    train_loss += [mse_train]
    test_loss += [mse_test]
    f1 += [f1score]
```

```
plt.figure(figsize=(8,6))
plt.plot(pca_number, train_loss, color="blue", label="MSE on Train data")
plt.plot(pca_number, test_loss, color="red", label="MSE on Test data")
plt.plot(pca_number, f1, color="green", label="f1 score")
plt.xlabel("PCA")
plt.ylabel("Mean Squared Error");
plt.legend()
```

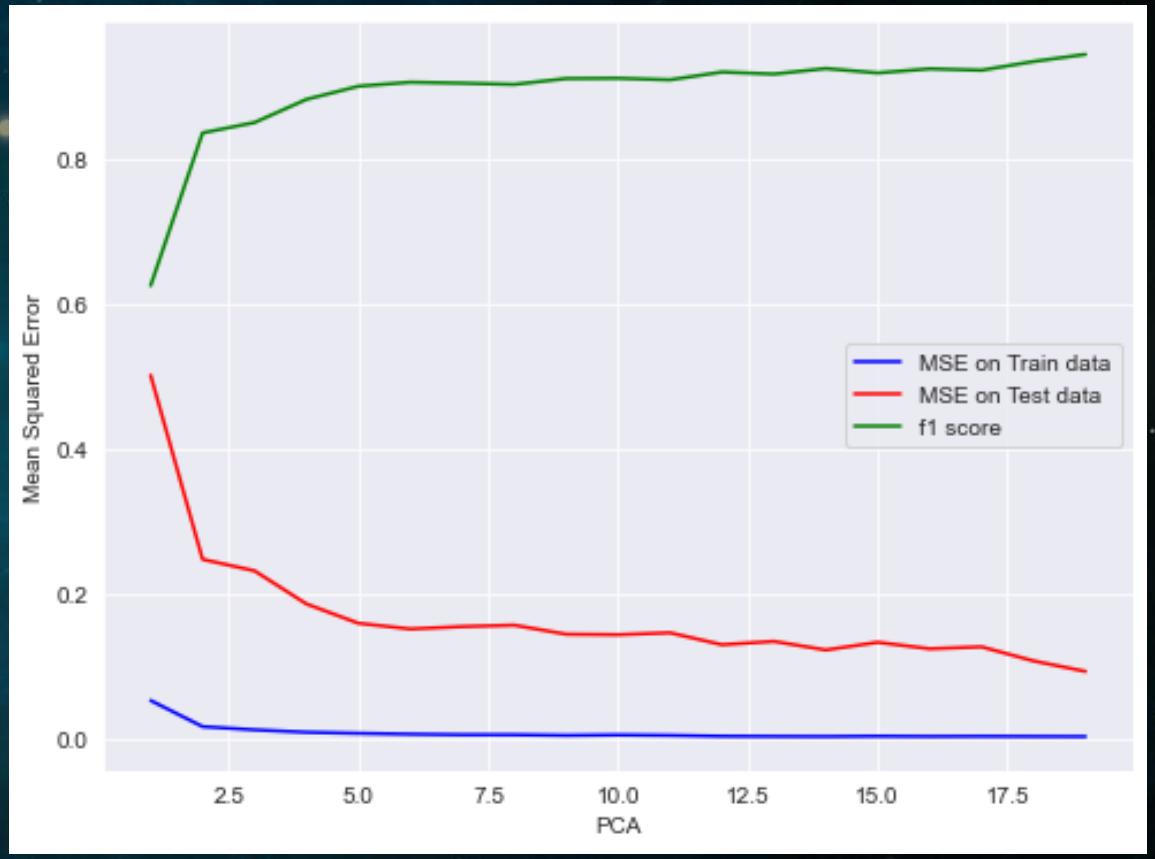


➢03-4. 模型分析—主成分疊代之MSE及F1

▲ RandomForest模型之主成分疊代之MSE及F1-score

```
from sklearn.metrics import mean_squared_error
pca_number2, train_loss2, test_loss2,f1_2 = [], [], [],[]
for i in range(1,20):
    pca = PCA(n_components=i)
    X_train_pca = pca.fit_transform(X_train_std)
    X_test_pca = pca.transform(X_test_std)
    forest.fit(X_train_pca,y_train)
    y_train_predicted = forest.predict(X_train_pca)
    y_test_predicted = forest.predict(X_test_pca)
    mse_train = mean_squared_error(y_train, y_train_predicted)
    mse_test = mean_squared_error(y_test, y_test_predicted)
    f1score=f1_score(y_true=y_test,y_pred=y_test_predicted)
    print("pca: {} Train mse: {} Test mse: {} f1: {}".format(i, mse_train, mse_test,f1score))
    pca_number2 += [i]
    train_loss2 += [mse_train]
    test_loss2 += [mse_test]
    f1_2 += [f1score]
```

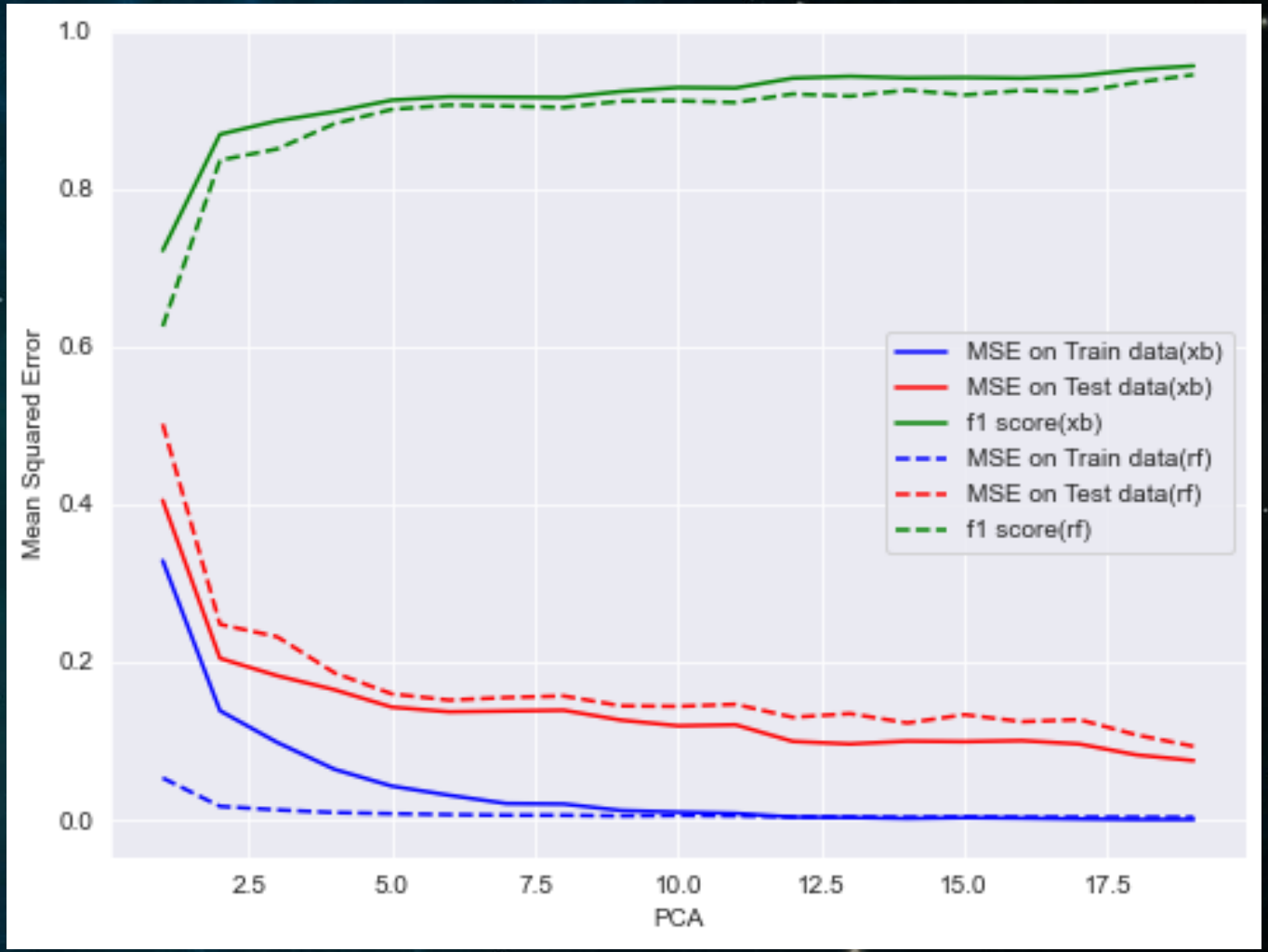
```
plt.figure(figsize=(8,6))
plt.plot(pca_number, train_loss, color="blue", label="MSE on Train data")
plt.plot(pca_number, test_loss, color="red", label="MSE on Test data")
plt.plot(pca_number, f1, color="green", label="f1 score")
plt.xlabel("PCA")
plt.ylabel("Mean Squared Error");
plt.legend()
```



➢03-4. 模型分析—主成分疊代之MSE及F1

- XGBoost和RandomForest模型主成分疊代之MSE及F1-score

```
plt.figure(figsize=(8,6))
plt.plot(pca_number, train_loss, color="blue", label="MSE on Train data(xb)")
plt.plot(pca_number, test_loss, color="red", label="MSE on Test data(xb)")
plt.plot(pca_number, f1, color="green", label="f1 score(xb)")
plt.plot(pca_number2, train_loss2, color="blue", label="MSE on Train data(rf)",linestyle='--')
plt.plot(pca_number2, test_loss2, color="red", label="MSE on Test data(rf)",linestyle='--')
plt.plot(pca_number2, f1_2, color="green", label="f1 score(rf)",linestyle='--')
plt.xlabel("PCA")
plt.ylabel("Mean Squared Error");
plt.legend()
```



04

報　告　總　結

04-1. 報告結論
04-2. 報告心得

➢04-1. 報告結論

▲ 可以分別針對年長者、小資族、信用卡額度較低的用戶
提出相關現金回饋或紅利以增加信用卡續約之意願。

▲ 持有銀行業務總數低之不續約機率高→提升銀行業務品質

▲ 若是這間公司需要預測用戶流失，可使用「XGBoost」建置模型進行分類

➢04-2. 報告心得

▲ 日四B 吳英緩

心得
1. 將過去機器學習的學習經驗學以致用
2. 挑選適當的模型與參數調整十分重要
3. 希望未來能更熟捻各類模型與提升調參能力

分工
1. 預測模型建置、評估
2. 程式碼校正
3. 書面及簡報排版調整

感謝您的細心聆聽

Thank you for your time and attention