Final Project

ADS 505 Applied Data Science for Business

Team 6: Lai Ieng Chan, Ben Earnest, Saba Alemayehu

Import of Packages and Libraries

```
pip install dmba
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: dmba in /usr/local/lib/python3.7/dist-packages (0.1.0)
```

```
%matplotlib inline
from pathlib import Path
from sklearn import preprocessing
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
import statsmodels.api as sm
import matplotlib.pylab as plt
import seaborn as sns
from dmba import classificationSummary, gainsChart, liftChart
from dmba.metric import AIC_score
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LassoCV, BayesianRidge
import statsmodels.formula.api as sm
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
import matplotlib.pylab as plt
from dmba import plotDecisionTree, classificationSummary, regressionSummary
from dmba import regressionSummary, exhaustive_search
from dmba import backward_elimination, forward_selection, stepwise_selection
from dmba import adjusted_r2_score, AIC_score, BIC_score
from sklearn.neural_network import MLPClassifier
from dmba import classificationSummary
from sklearn.preprocessing import MinMaxScaler
from imblearn.under_sampling import NearMiss
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report
from sklearn import metrics
```

```
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report, confusion_matrix

import warnings
warnings.filterwarnings('ignore')
```

```
no display found. Using non-interactive Agg backend
```

Load Data

```
df = pd.DataFrame(pd.read_csv('/content/cross_sell_dataset (1).tab', sep='\t'))
df
```

	cross_buy	acad_title	age	calls	complaints	customer_tenure_months	directmai
0	0	0	60	0	0	221	
1	0	0	55	0	0	227	
2	0	0	61	0	0	221	
3	0	0	70	0	0	222	
4	0	1	61	0	0	227	
...	...	...	...	...	...	...	
99995	1	0	53	0	0	206	
99996	1	0	25	0	0	206	
99997	1	0	19	0	0	205	
99998	1	0	58	0	0	204	
99999	1	0	55	0	0	205	

100000 rows × 35 columns

```
# Summary Information about the variables and their types in the data:
data_desc = pd.DataFrame(pd.read_csv('/content/Data Set Description.tab - Data Set Descriptio
data_desc
```

8	logins_desktop	Desktop Logins	Number of logins in the last 180 days
9	logins_mobile	Mobile Logins	Number of mobile sessions in the last 180 days
10	nr_products	Number of Products	Total number of products (accounts)
11	outflows	Outflows	Total volume of outflows from savings account ...
12	prod_loan	Loans	Number of consumer loan accounts
13	prod_mortgages	Mortgages	Number of mortgage accounts
14	prod_brokerage	Brokerage	Number of investment accounts
15	prod_pensionplan	Pension Plan	Number of long term savings plans
16	prod_savings	Savings	Number of savings accounts
17	relocations	Relocations	Number of relocations/address changes in the l...
18	volume_debit	Total Debit	Total balances of all debit (savings) accounts...
19	volume_debit_6months	Total Debit Six Months	Credit balance all of products from 6 months a...
20	Marketing Efforts	NaN	NaN
21	directmails	Direct Mailing	Total number of mailing in the last year
22	giro_mailing	Giro Mailing	Received an email about opening a checking acc...
23	Customer Characteristics	NaN	NaN
24	acad_title	Academic Title	Does the customer have an academic title: 1 (y...
25	age	Age	Customer's age in years
26	joint_account	Joint Account	Customer has a joint bank account: 1 (yes), 0 ...
27	gender	Gender	Customer's gender: 1 (male), 0 (female)
28	marital_status	Marital Status	Customer's marital status: divorced, married, ...
29	occupation	Occupation	Customer's occupation: white-collar worker, se...
30	member_get_member_active	Get Member Active	Customer recommended a customer: 1 (yes), 0 (no)
31	member_get_member_passive	Get Member Passive	Customer was recommended by a customer: 1 (yes...

32	ext_city_size	City Size	City size: 1 (< 5.000 inhabitants), 2 (5.000-

Exploratory Data Analysis

Initial Investigation into the Dataset and the Response Variable

```
# View columns, dimensions, and data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 35 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   cross_buy                  100000 non-null  int64
 1   acad_title                 100000 non-null  int64
 2   age                        100000 non-null  int64
 3   calls                      100000 non-null  int64
 4   complaints                 100000 non-null  int64
 5   customer_tenure_months     100000 non-null  int64
 6   directmails                100000 non-null  int64
 7   gender                     99998 non-null   float64
 8   joint_account              99998 non-null   float64
 9   inflows                    99527 non-null   float64
 10  last_acc_opening_days      100000 non-null  int64
 11  logins_desktop             100000 non-null  int64
 12  logins_mobile              100000 non-null  int64
 13  marital_status             100000 non-null  object
 14  member_get_member_active   100000 non-null  int64
 15  member_get_member_passive  100000 non-null  int64
 16  nr_products                100000 non-null  int64
 17  occupation                 48725 non-null   object
 18  outflows                   99527 non-null   float64
 19  prod_loan                  100000 non-null  int64
 20  prod_mortgages             100000 non-null  int64
 21  prod_brokerage             100000 non-null  int64
 22  prod_pensionplan           100000 non-null  int64
 23  prod_savings               100000 non-null  int64
 24  relocations                100000 non-null  int64
 25  volume_debit               100000 non-null  float64
 26  volume_debit_6months       97002 non-null   float64
```

```
 27  ext_city_size            97338 non-null   float64
 28  ext_house_size           96953 non-null   float64
 29  ext_purchase_power       95435 non-null   float64
 30  ext_share_new_houses     97338 non-null   float64
 31  ext_share_new_cars       84845 non-null   float64
 32  ext_car_power            89866 non-null   float64
 33  ext_living_duration      90935 non-null   float64
 34  giro_mailing             100000 non-null  int64
dtypes: float64(13), int64(20), object(2)
memory usage: 26.7+ MB
```

```
# The dimension of the dataset
print('Number of Rows:', df.shape[0])
print('Number of Columns:', df.shape[1])
```

```
Number of Rows: 100000
Number of Columns: 35
```

```
# check for missing values
df.isnull().sum()
```

```
cross_buy                     0
acad_title                    0
age                           0
calls                         0
complaints                    0
customer_tenure_months        0
directmails                   0
gender                        2
joint_account                 2
inflows                     473
last_acc_opening_days         0
logins_desktop                0
logins_mobile                 0
marital_status                0
member_get_member_active      0
member_get_member_passive     0
nr_products                   0
occupation                51275
outflows                    473
prod_loan                     0
prod_mortgages                0
prod_brokerage                0
prod_pensionplan              0
prod_savings                  0
relocations                   0
volume_debit                  0
volume_debit_6months       2998
ext_city_size              2662
ext_house_size             3047
ext_purchase_power         4565
ext_share_new_houses       2662
ext_share_new_cars        15155
ext_car_power             10134
```

```
ext_living_duration          9065
giro_mailing                    0
dtype: int64
```

Response variable - cross_buy

```
# Check Target Class Distribution
# 0 = Customers didn't open a checking account
# 1 = Customers opened a checking account
df['cross_buy'].value_counts()
```

```
0    90000
1    10000
Name: cross_buy, dtype: int64
```

```
# Plot Class Distribution
sns.countplot(data = df, x = 'cross_buy')
plt.xlabel('cross_buy')
plt.ylabel('Frequency')
plt.title('cross_buy Distribution')
plt.show()
```

"cross_buy" tells us if an existing customer opened a checking account. From here we see that 10,000 customers out of the 10,000 customers did.

For this problem, the class imbalancement might have an effect since 90% of the data are in one class and there aren't enough negative and positive classes for training. We should resample the data so that more customers would seemed to open the bank account.

Note, the cross buy rate for the existing customers is 10%. So our baseline suggests randomly selecting customers to contact for checking accounts will result in a 10% success rate.

Investigation in other Independent Variables

Selected Numerical features

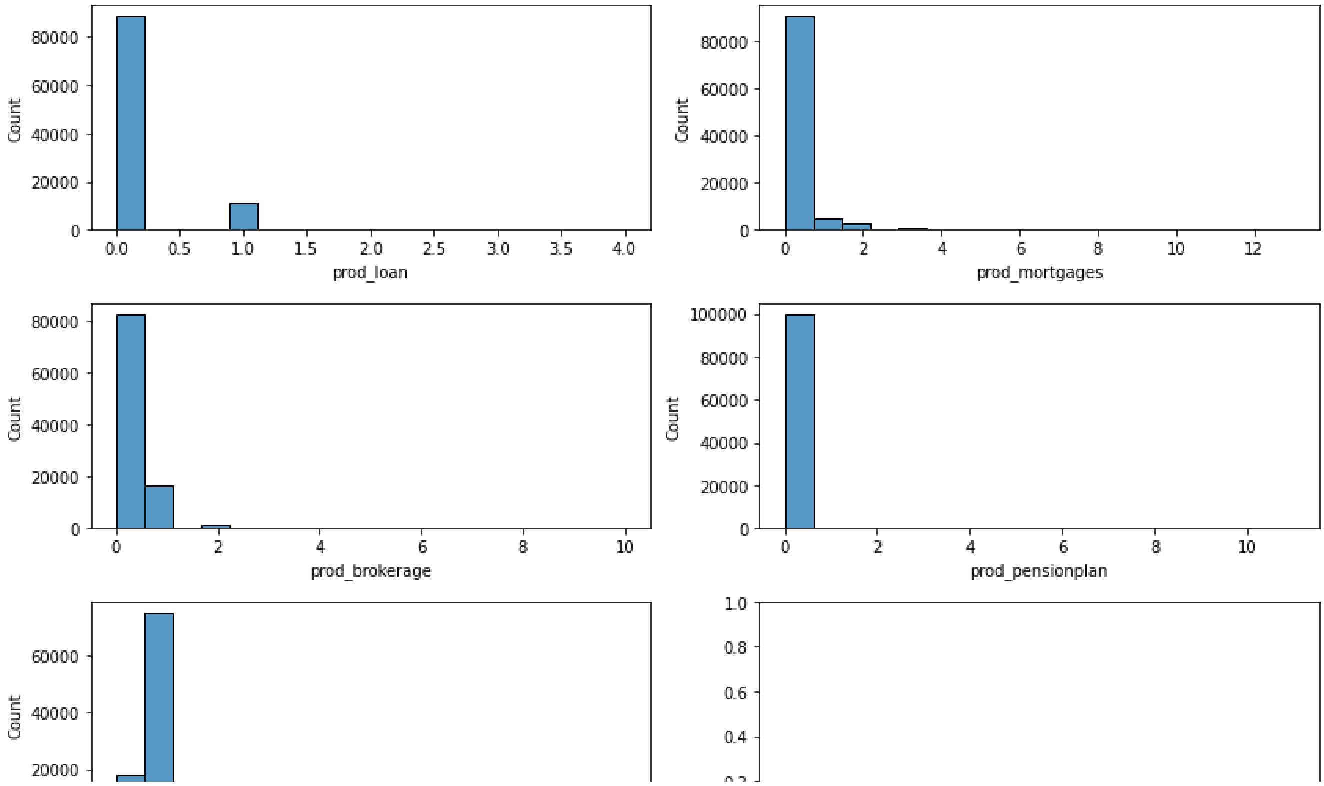
```
# Subsetting the selected numerical features into a dataset
num_features = df[['age','calls', 'complaints', 'customer_tenure_months', 'nr_products', 'dir

# Retunr decription of the numerical features
num_features.describe()
```

	age	calls	complaints	customer_tenure_months	nr_product
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.00000
mean	49.892260	0.104600	0.003530	140.181700	1.43338
std	14.534085	0.564395	0.078598	74.901654	0.79857
min	18.000000	0.000000	0.000000	0.000000	1.00000
25%	39.000000	0.000000	0.000000	78.000000	1.00000
50%	51.000000	0.000000	0.000000	160.000000	1.00000
75%	60.000000	0.000000	0.000000	201.000000	2.00000

```
# Plot Features
%matplotlib inline
f, axs = plt.subplots(2, 2, figsize = (12, 8))

# Customer's age in years
sns.histplot(data = df, x = 'age', ax = axs[0,0])

# Number of months since customer onboarding
sns.histplot(data = df, x = 'customer_tenure_months', ax = axs[0,1])

# Number of complaints in the last year
sns.histplot(data = df, x = 'complaints', ax = axs[1,0])

# Number of calls customers get in the last 180 days
sns.histplot(data = df, x = 'calls', ax = axs[1,1])

axs[0, 0].title.set_text("Age Distribution")
axs[0, 1].title.set_text("Customer Onboarding Time Distribution")
axs[1, 0].title.set_text("Number of Complaints Customers Filed")
axs[1, 1].title.set_text("Number of Calls Customers Get")

plt.tight_layout()
```

Financial Products that customers own:

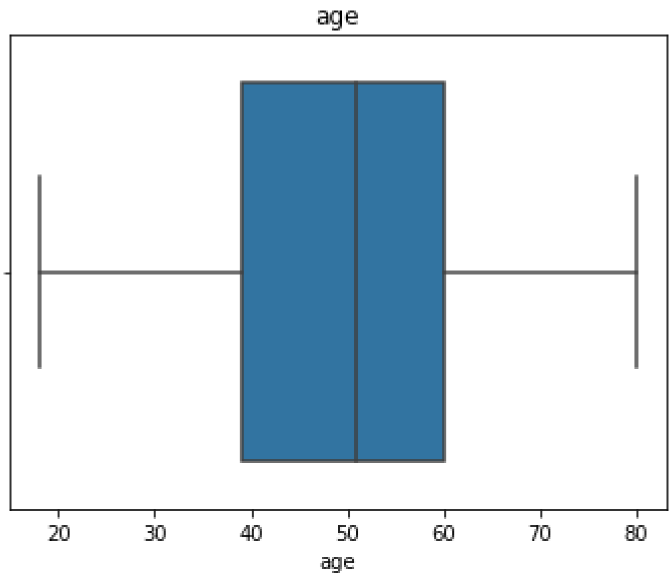
```
# Plot the financial products that consumers have
%matplotlib inline

f, axs = plt.subplots(3, 2, figsize = (12, 8))

# Distribution of number of loan accounts consumers have
sns.histplot(data = df, x = 'prod_loan', ax = axs[0,0])

# Distribution of number of mortgage accounts consumers have
sns.histplot(data = df, x = 'prod_mortgages', ax = axs[0,1])

# Distribution of number of investment accounts consumers have
sns.histplot(data = df, x = 'prod_brokerage', ax = axs[1,0])

# Distribution of number of long-term savings accounts consumers have
sns.histplot(data = df, x = 'prod_pensionplan', ax = axs[1,1])

# Distribution of number of savings accounts consumers have
sns.histplot(data = df, x = 'prod_savings', ax = axs[2,0])

plt.tight_layout()
```

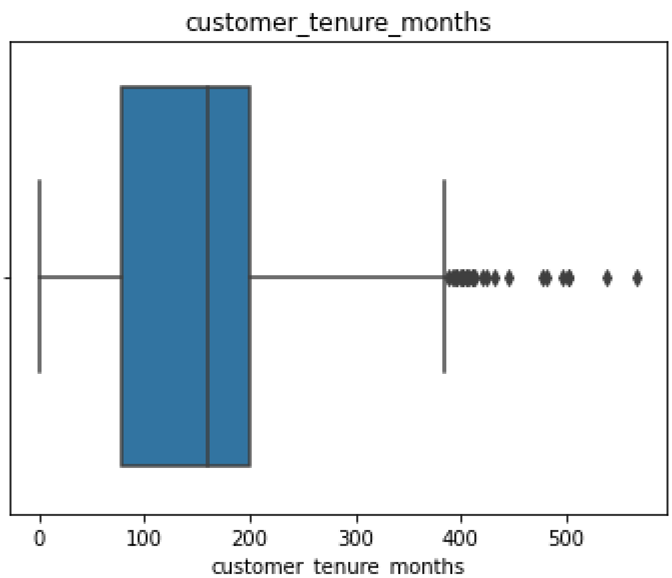


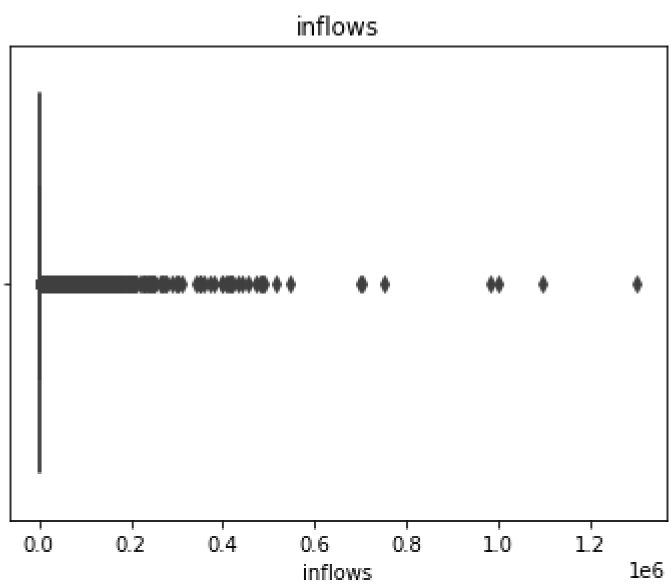
We can see that most customers own a savings account but not the other financial accounts.

Visualizing Outliers

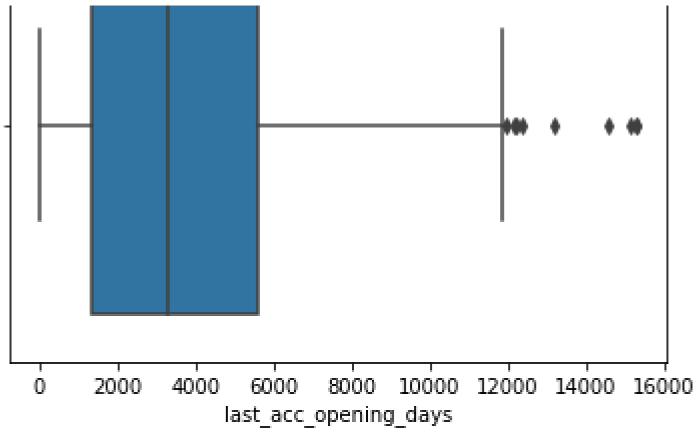
```
# Visualizing each variables outlier
col = df[['age','customer_tenure_months','inflows',
                 'last_acc_opening_days','logins_desktop',
                 'outflows','volume_debit','volume_debit_6months']]

for i in col:
  n=1
  plt.figure(figsize=(20,20))
  plt.subplot(4,3,1)
  sns.boxplot(df[i])
  plt.title(i)
  plt.show()
  n=n+1
```


age
20
30
40
50
60
70
80
age

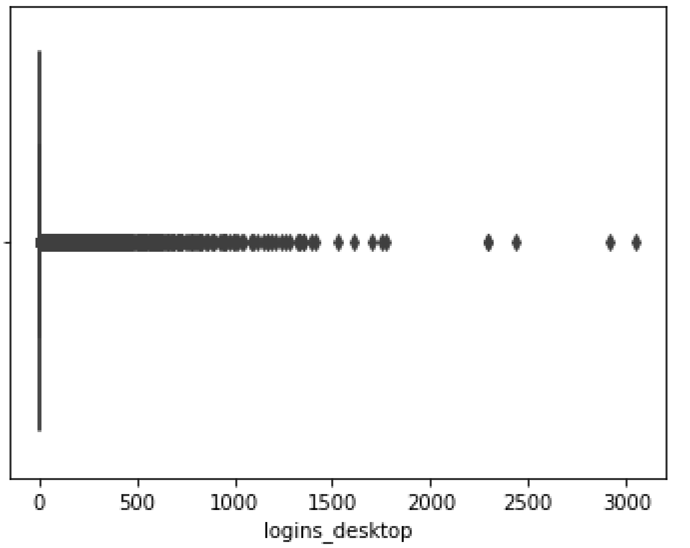

customer_tenure_months
0
100
200
300
400
500
customer_tenure_months


inflows
0.0
0.2
0.4
0.6
0.8
1.0
1.2
inflows
1e6

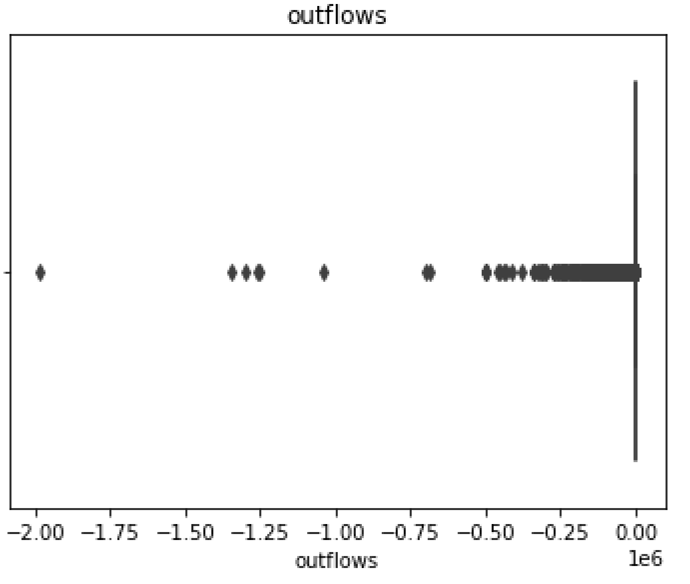

last_acc_opening_days



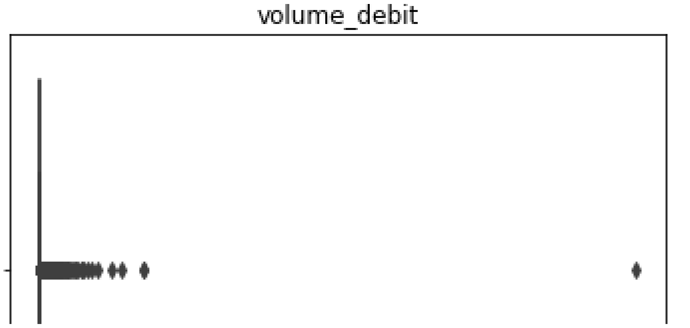
logins_desktop

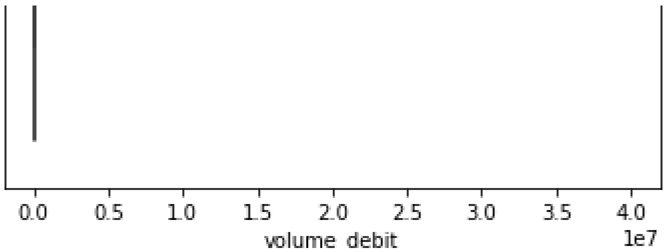


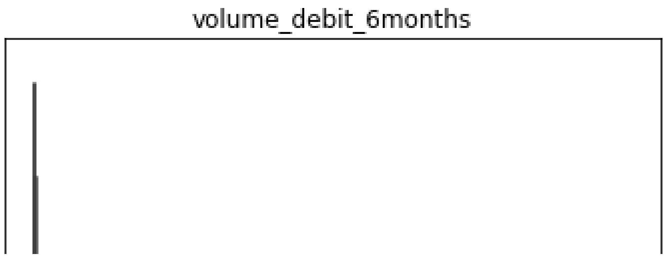
outflows



volume_debit

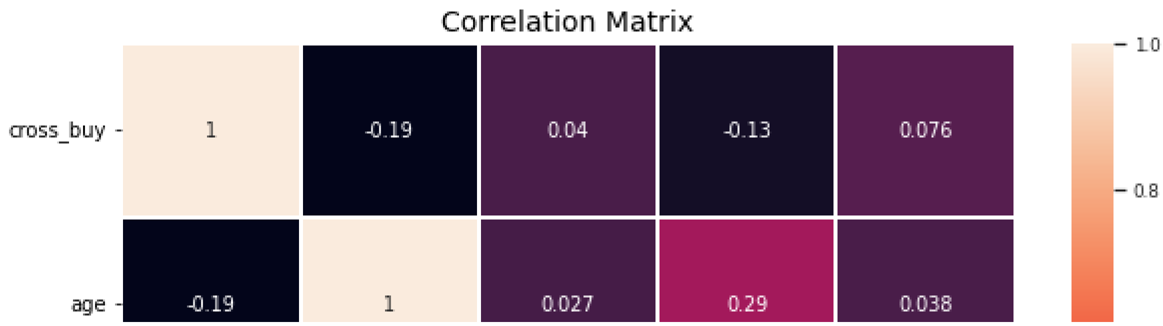






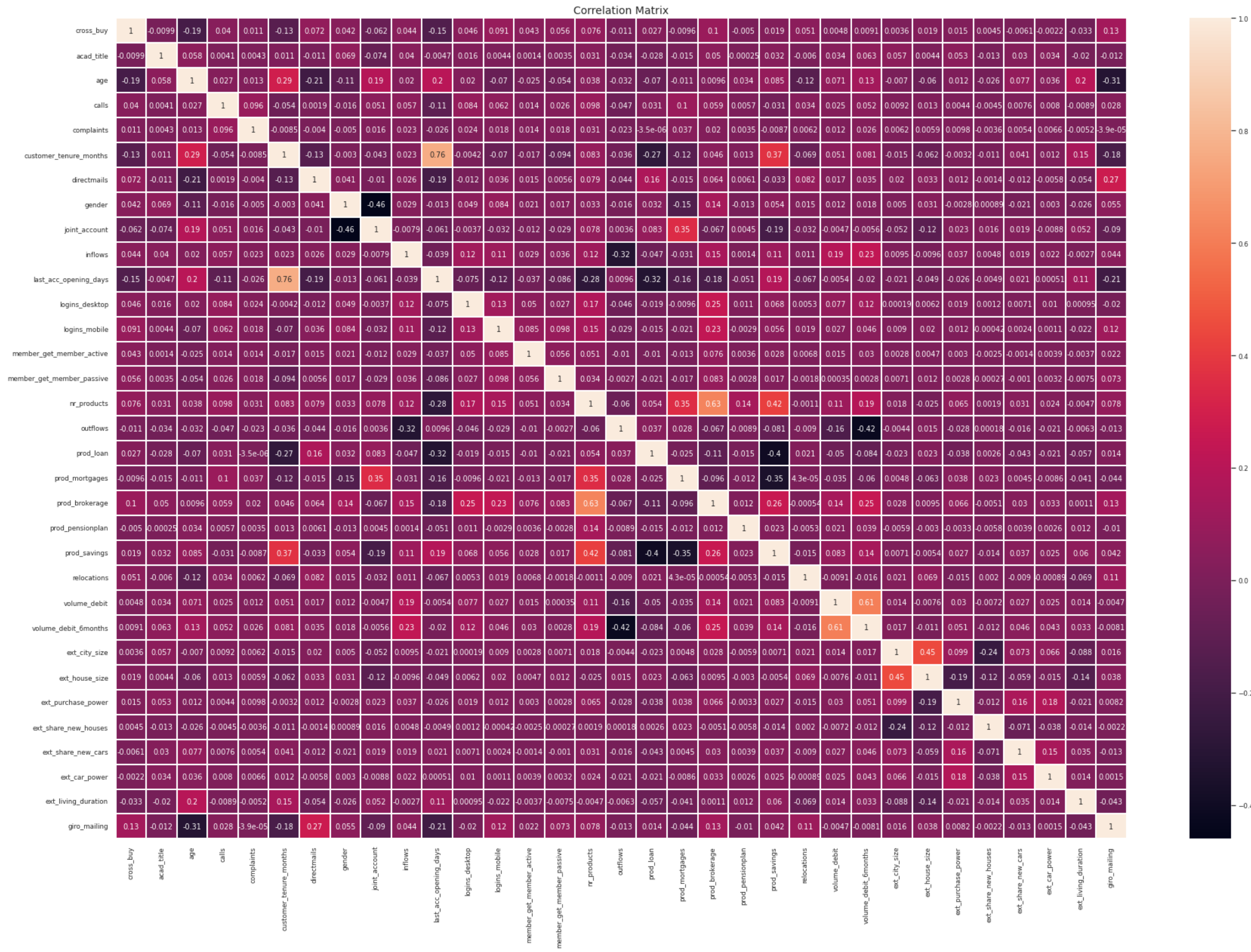
Correlation Matrix

```
# Selecting more relevant variables to compute the correlation matrix
df_corr = df[['cross_buy','age', 'calls', 'customer_tenure_months', 'nr_products']]

corrMatrix = df_corr.corr()
plt.subplots(figsize = (10,8))
sns.set(font_scale = .8)
sns.heatmap(corrMatrix, annot = True, linewidths = 1)
plt.title(r'Correlation Matrix', fontsize = 14)
plt.savefig('Correlation Matrix')
```




```
corrMatrix = df.corr()
plt.subplots(figsize = (30,20))
sns.set(font_scale = .8)
sns.heatmap(corrMatrix, annot = True, linewidths = 1)
plt.title(r'Correlation Matrix', fontsize = 14)
```

```
Text(0.5, 1.0, 'Correlation Matrix')
```



Pre-processing Data

```
# Check for missing values
df.isnull().sum()
```

```
cross_buy                    0
acad_title                   0
age                          0
calls                        0
complaints                   0
customer_tenure_months       0
directmails                  0
gender                       2
joint_account                2
inflows                    473
last_acc_opening_days        0
logins_desktop               0
logins_mobile                0
marital_status               0
member_get_member_active     0
member_get_member_passive    0
nr_products                  0
```

```
occupation                  51275
outflows                      473
prod_loan                       0
prod_mortgages                  0
prod_brokerage                  0
prod_pensionplan                0
prod_savings                    0
relocations                     0
volume_debit                    0
volume_debit_6months         2998
ext_city_size                2662
ext_house_size               3047
ext_purchase_power           4565
ext_share_new_houses         2662
ext_share_new_cars          15155
ext_car_power               10134
ext_living_duration          9065
giro_mailing                    0
dtype: int64
```

```
# Columns including occupationshould be dropped since it contains mostly NaN values and
# won't provide a lot of meaningful information.
df = df.drop(columns=['occupation'])
```

The curse of dimensionality means that the error increases with the increase in the number of features. Since we have 35 features, there is a need to select the important features and remove the irrelevant ones for better performance of the model.

Regarding account balances, we have four features that describes the account balances:

- inflows - It describes the total volume of inflows on savings account
- outflows - It describes the total volume of outflows on savings account
- volume_debit - It describes the total balances of all debit (savings) accounts
- volume_debit_6months - It describes the total balances of all debit (savings) accounts in six months

Since we are interested in knowing the account balances that customers have which will have a relationship with whether a customer will open a checking account, we will use the most relevant feature which is volume_debit, and drop the other features.

Regarding the days since the account opened, we have features that described it:

- customer_tenure_months - Number of months since customer onboarding
- last_acc_opening_days - Number of days since last account opening

We selected the customer_tenure_months since that gives us a better idea of the time customers are with the bank. Thus, we will remove the other feature.

```
# Since we will be using the total balances of all accounts, we can drop the redundant featur
df = df.drop(columns=['inflows','outflows','volume_debit_6months'])

# Also dropping duplicate feature last_acc_opening_days
df = df.drop(columns=['last_acc_opening_days'])
```

As mentioned above, most customers own 0 financial products except savings account, we would drop the irrelevant features and keep the prod_savings feature.

```
# Dropping the features of financial products that majority of consumers do not have since th
df = df.drop(columns=['prod_loan','prod_mortgages','prod_brokerage','prod_pensionplan'])
```

Looking atthe external features including the following:

- ext_city_size - City size
- ext_house_size - Average number of households per building in the residential block
- ext_purchase_power - Average purchase power in the residential block
- ext_share_new_houses - Share of new buildings in the residential block
- ext_share_new_cars - Share of new vehicle registrations in the residential block
- ext_car_power - Predominant vehicle category in the neighborhood
- ext_living_duration - Average duration of residence in the customer's building

The only features that are more relevant features to whether a customer will open a checking account will be the purchase power pf clients. The other features including city size, average number of households, etc. is a lot less relevant in impacting someone to open a checking account. Thus, we will remove the irrelevant features.

```
# Dropping irrelevant external factors that are less likely to affect if the consumer will op
df = df.drop(columns=['ext_city_size','ext_house_size','ext_share_new_houses','ext_share_new_
```

Lastly, there are some other less irrelevant features that we consider removing prior to the training of model:

- relocations - Number of relocations/address changes in the last year
- acad_title - Whether the customer have an academic title
- logins_desktop - Number of logins in the last 180 days
- logins_mobile - Number of mobile sessions in the last 180 days

```
# Dropping other less irrelevant factors to finalize the dataset for modeling
df = df.drop(columns=['relocations','acad_title','logins_desktop','logins_mobile','directmail
```

```
df
```

	cross_buy	age	calls	complaints	customer_tenure_months	gender	joint_account
0	0	60	0	0	221	0.0	0.0
1	0	55	0	0	227	0.0	1.0
2	0	61	0	0	221	1.0	0.0
3	0	70	0	0	222	0.0	0.0
4	0	61	0	0	227	1.0	0.0
...	...	...	...	...	...	...	...
99995	1	53	0	0	206	0.0	0.0
99996	1	25	0	0	206	0.0	0.0
99997	1	19	0	0	205	1.0	0.0
99998	1	58	0	0	204	1.0	0.0
99999	1	55	0	0	205	0.0	1.0

100000 rows × 15 columns

```
# Converting categorical variables to dummies

# Treat marital_status and ext_purchase_power as categorical, then convert to dummy variables
df['marital_status'] = df['marital_status'].astype('category')

df = pd.get_dummies(df, columns=['marital_status'], drop_first=True)
df.head(20)
```

	cross_buy	age	calls	complaints	customer_tenure_months	gender	joint_account	m
0	0	60	0	0	221	0.0	0.0	
1	0	55	0	0	227	0.0	1.0	
2	0	61	0	0	221	1.0	0.0	
3	0	70	0	0	222	0.0	0.0	
4	0	61	0	0	227	1.0	0.0	
5	0	67	0	0	227	0.0	1.0	
6	0	55	0	0	227	0.0	1.0	
7	0	78	0	0	227	0.0	1.0	
8	0	58	0	0	227	1.0	0.0	
9	0	47	0	0	221	1.0	0.0	
10	0	65	0	0	221	0.0	1.0	
11	0	56	0	0	227	1.0	0.0	
12	0	63	0	0	222	0.0	0.0	
13	0	54	0	0	227	1.0	0.0	

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

```
7.0    22197
6.0    16750
5.0    14529
4.0    12370
3.0    10802
1.0     9491
2.0     9296
Name: ext_purchase_power, dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 20 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   cross_buy                 100000 non-null  int64  
 1   age                       100000 non-null  int64  
 2   calls                     100000 non-null  int64  
 3   complaints                100000 non-null  int64  
 4   customer_tenure_months    100000 non-null  int64  
 5   gender                    99998 non-null   float64
 6   joint_account             99998 non-null   float64
 7   member_get_member_active  100000 non-null  int64  
```

```
 8   member_get_member_passive  100000 non-null  int64
 9   nr_products                100000 non-null  int64
 10  prod_savings               100000 non-null  int64
 11  volume_debit               100000 non-null  float64
 12  ext_purchase_power         95435 non-null   float64
 13  giro_mailing               100000 non-null  int64
 14  marital_status_divorced    100000 non-null  uint8
 15  marital_status_married     100000 non-null  uint8
 16  marital_status_separated   100000 non-null  uint8
 17  marital_status_single      100000 non-null  uint8
 18  marital_status_unmarried   100000 non-null  uint8
 19  marital_status_widowed     100000 non-null  uint8
dtypes: float64(4), int64(10), uint8(6)
memory usage: 11.3 MB
```

```
# Converting categorical variables to dummies

# Treat marital_status and ext_purchase_power as categorical, then convert to dummy variables
df['marital_status'] = df['marital_status'].astype('category')
df['ext_purchase_power'] = df['ext_purchase_power'].astype('category')

df = pd.get_dummies(df, columns=['marital_status','ext_purchase_power'], drop_first=True)
df
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

4 frames

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'marital_status'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364 
   3365         if is_scalar(key) and isna(key) and not self.hasnans:

KeyError: 'marital_status'
```

```
# Check for missing values again
df.isnull().sum()
```

```
cross_buy                     0
age                           0
calls                         0
complaints                    0
customer_tenure_months        0
gender                        2
joint_account                 2
member_get_member_active      0
member_get_member_passive     0
nr_products                   0
prod_savings                  0
volume_debit                  0
ext_purchase_power         4565
giro_mailing                  0
marital_status_divorced       0
marital_status_married        0
marital_status_separated      0
marital_status_single         0
marital_status_unmarried      0
marital_status_widowed        0
dtype: int64
```

```
# imputation of missing data with mean imputer
mean_imputer = SimpleImputer(strategy='mean')
df[['gender','joint_account']] = pd.DataFrame(mean_imputer.fit_transform(df[['gender','joint_
```

```
# imputation of missing data with mode imputer
mode_imputer = SimpleImputer(strategy='most_frequent')
df[['ext_purchase_power']] = pd.DataFrame(mode_imputer.fit_transform(df[['ext_purchase_power'
```

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

```
cross_buy                    0
age                          0
calls                        0
complaints                   0
customer_tenure_months       0
gender                       0
joint_account                0
member_get_member_active     0
member_get_member_passive    0
nr_products                  0
prod_savings                 0
volume_debit                 0
ext_purchase_power           0
giro_mailing                 0
marital_status_divorced      0
marital_status_married       0
marital_status_separated     0
```

```
marital_status_single       0
marital_status_unmarried    0
marital_status_widowed      0
dtype: int64
```

```
# Partition the data into training (60%) and validation (40%). Use seed = 1.
y = df['cross_buy']
X = df.drop(columns=['cross_buy'])

# Resampling the cross_buy variables using over-sampling techniques
nm = NearMiss()
X_res, y_res = nm.fit_resample(X, y)


# Split the data in training and valid data
X_train, X_valid, y_train, y_valid = train_test_split(X_res, y_res, train_size=0.4, random_st


# Scaling the variables
norm = MinMaxScaler().fit(X_train)
X_train = norm.transform(X_train)
X_valid = norm.transform(X_valid)
```

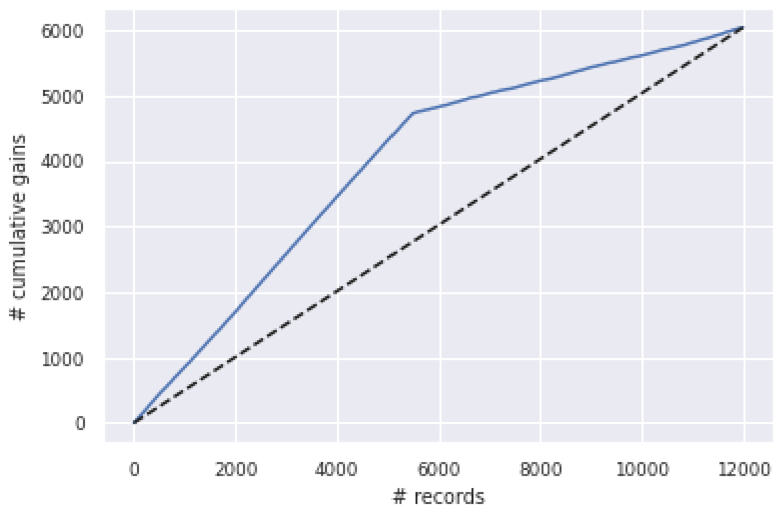
Modeling

Logistic Regression Model

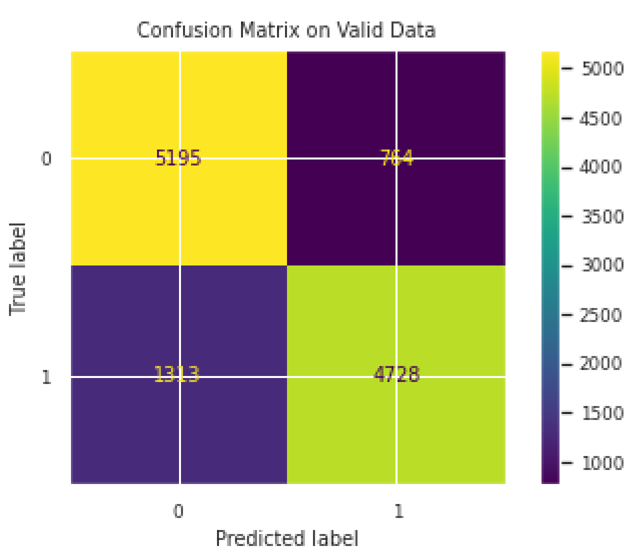
```
# Training logistic regression model
model = LogisticRegression()
grid = {"C": np.logspace(-3,3,7),
      "penalty":["l1", "l2"],# l1 lasso l2 ridge
      "solver": ['liblinear']}

logreg_cv = GridSearchCV(model, grid, cv=10)
logreg_cv.fit(X_train, y_train)

print("tuned hpyerparameters :(best parameters) ", logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
log_pred =  logreg_cv.predict(X_valid)

# Evaluation
print(classification_report(y_valid,log_pred))

# Plot the cumulative gains chart of the expected spending
gains_df = pd.DataFrame({'actual': y_valid,
                         'prob': log_pred})
gains_df = gains_df.sort_values(by=['prob'], ascending=False).reset_index(drop=True)
gainsChart(gains_df.actual)
```

```
gainsChart(gains_df.actual)
plt.show()
```

```
tuned hpyerparameters :(best parameters)  {'C': 10.0, 'penalty': 'l1', 'solver': 'liblir
accuracy : 0.8215
              precision    recall  f1-score   support

           0       0.80      0.87      0.83      5959
           1       0.86      0.78      0.82      6041

    accuracy                           0.83     12000
   macro avg       0.83      0.83      0.83     12000
weighted avg       0.83      0.83      0.83     12000
```




```
# Plot Confusion Matrix
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(logreg_cv, X_valid, y_valid)
plt.title('Confusion Matrix on Valid Data')
plt.show()
```



```
# Train logistic regression model
log = LogisticRegression(penalty="l2", C=1e42, solver='liblinear')
log.fit(X_train, y_train)

# Training performance
classificationSummary(y_train, log.predict(X_train))

# Validation performance
classificationSummary(y_valid, log.predict(X_valid))

print(classification_report(y_valid,log.predict(X_valid)))
```

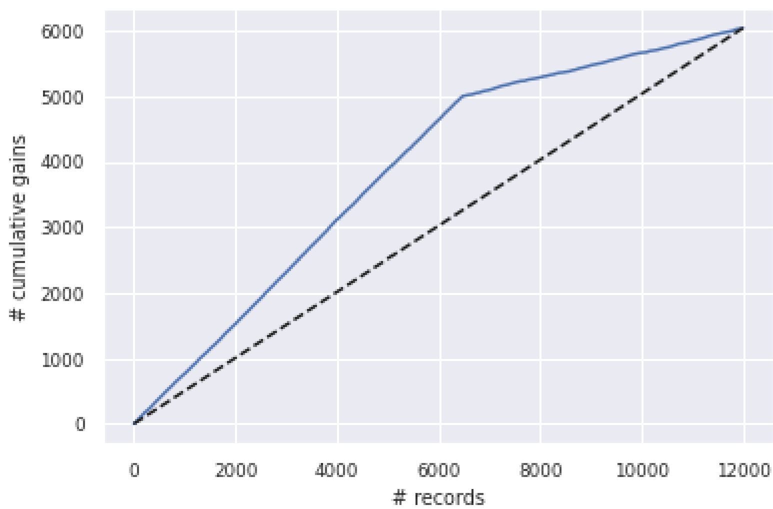
```
Confusion Matrix (Accuracy 0.8631)

       Prediction
Actual    0    1
     0 3962   79
     1 1016 2943
Confusion Matrix (Accuracy 0.8649)

       Prediction
Actual    0    1
     0 5838  121
     1 1500 4541
              precision    recall  f1-score   support

           0       0.80      0.98      0.88      5959
           1       0.97      0.75      0.85      6041

    accuracy                           0.86     12000
   macro avg       0.88      0.87      0.86     12000
weighted avg       0.89      0.86      0.86     12000
```

Support Vector Machine

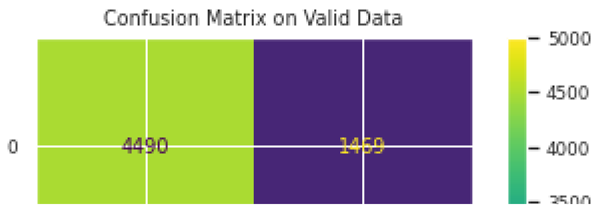
```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)

#Making Predictions
Svm_pred = svclassifier.predict(X_valid)

#Evaluating the Algorithm
classificationSummary(y_valid, svclassifier.predict(X_valid))
print(classification_report(y_valid,Svm_pred))

# Plot the cumulative gains chart of the expected spending
gains_df = pd.DataFrame({'actual': y_valid,
                         'prob': Svm_pred})
```

```
gains_df = gains_df.sort_values(by=['prob'], ascending=False).reset_index(drop=True)
gainsChart(gains_df.actual)
plt.show()
```

```
Confusion Matrix (Accuracy 0.7908)

       Prediction
Actual    0    1
     0 4490 1469
     1 1041 5000
              precision    recall  f1-score   support

           0       0.81      0.75      0.78      5959
           1       0.77      0.83      0.80      6041

    accuracy                           0.79     12000
   macro avg       0.79      0.79      0.79     12000
weighted avg       0.79      0.79      0.79     12000
```




```
# Plot Confusion Matrix
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(svclassifier, X_valid, y_valid)
plt.title('Confusion Matrix on Valid Data')
plt.show()
```




```
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['linear']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(X_train, y_train)
```

```
[CV 2/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.767 total time=   1.5s
[CV 3/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.782 total time=   1.5s
[CV 4/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.771 total time=   1.4s
[CV 5/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.771 total time=   1.5s
[CV 1/5] END .......C=1, gamma=1, kernel=linear;, score=0.787 total time=   2.0s
[CV 2/5] END .......C=1, gamma=1, kernel=linear;, score=0.783 total time=   1.9s
[CV 3/5] END .......C=1, gamma=1, kernel=linear;, score=0.782 total time=   2.3s
[CV 4/5] END .......C=1, gamma=1, kernel=linear;, score=0.786 total time=   2.0s
[CV 5/5] END .......C=1, gamma=1, kernel=linear;, score=0.769 total time=   2.2s
[CV 1/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.787 total time=   2.1s
[CV 2/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.783 total time=   1.9s
[CV 3/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.782 total time=   2.3s
[CV 4/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.786 total time=   2.0s
[CV 5/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.769 total time=   2.3s
[CV 1/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.787 total time=   2.1s
[CV 2/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.783 total time=   1.9s
[CV 3/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.782 total time=   2.3s
[CV 4/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.786 total time=   2.0s
[CV 5/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.769 total time=   2.3s
[CV 1/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.787 total time=   2.0s
[CV 2/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.783 total time=   1.9s
[CV 3/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.782 total time=   3.2s
[CV 4/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.786 total time=   2.5s
[CV 5/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.769 total time=   2.3s
[CV 1/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.787 total time=   2.1s
[CV 2/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.783 total time=   1.9s
[CV 3/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.782 total time=   2.3s
[CV 4/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.786 total time=   2.1s
[CV 5/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.769 total time=   2.3s
[CV 1/5] END ......C=10, gamma=1, kernel=linear;, score=0.787 total time=   5.1s
[CV 2/5] END ......C=10, gamma=1, kernel=linear;, score=0.783 total time=   4.7s
[CV 3/5] END ......C=10, gamma=1, kernel=linear;, score=0.782 total time=   6.7s
[CV 4/5] END ......C=10, gamma=1, kernel=linear;, score=0.786 total time=   4.7s
[CV 5/5] END ......C=10, gamma=1, kernel=linear;, score=0.769 total time=   5.9s
[CV 1/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.787 total time=   5.3s
```

```
[CV 2/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.783 total time=   4.6s
[CV 3/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.782 total time=   6.6s
[CV 4/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.786 total time=   4.7s
[CV 5/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.769 total time=   5.9s
[CV 1/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.787 total time=   5.2s
[CV 2/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.783 total time=   4.9s
[CV 3/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.782 total time=   7.3s
[CV 4/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.786 total time=   4.8s
[CV 5/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.769 total time=   6.0s
[CV 1/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.787 total time=   5.1s
[CV 2/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.783 total time=   4.6s
[CV 3/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.782 total time=   6.6s
[CV 4/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.786 total time=   4.7s
[CV 5/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.769 total time=   5.9s
[CV 1/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.787 total time=   5.2s
[CV 2/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.783 total time=   4.7s
[CV 3/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.782 total time=   6.7s
[CV 4/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.786 total time=   4.7s
[CV 5/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.769 total time=   5.9s
[CV 1/5] END .....C=100, gamma=1, kernel=linear;, score=0.787 total time=   8.2s
[CV 2/5] END .....C=100, gamma=1, kernel=linear;, score=0.783 total time=   8.7s
[CV 3/5] END .....C=100, gamma=1, kernel=linear;, score=0.782 total time=   9.5s
[CV 4/5] END .....C=100, gamma=1, kernel=linear;, score=0.786 total time=   6.2s
```

```
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
best_grid = grid.best_estimator_
```

```
{'C': 1000, 'gamma': 1, 'kernel': 'linear'}
SVC(C=1000, gamma=1, kernel='linear')
```

```
grid_predictions = best_grid.predict(X_valid)

# print classification report
print(classification_report(y_valid, grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.80      5959
           1       0.79      0.82      0.80      6041

    accuracy                           0.80     12000
   macro avg       0.80      0.80      0.80     12000
weighted avg       0.80      0.80      0.80     12000
```

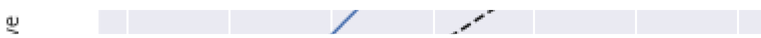
Linear Discriminant Analysis

```
from sklearn.model_selection import RepeatedStratifiedKFold
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

model = LinearDiscriminantAnalysis()
model.fit(X_train, y_train)

classificationSummary(y_valid, model.predict(X_valid))
print(classification_report(y_valid, model.predict(X_valid)))


# Plot the cumulative gains chart of the expected spending
gains_df = pd.DataFrame({'actual': y_valid,
                         'prob': model.predict(X_valid)})
gains_df = gains_df.sort_values(by=['prob'], ascending=False).reset_index(drop=True)
gainsChart(gains_df.actual)
plt.show()

# Plot Confusion Matrix
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model, X_valid, y_valid)
plt.title('Confusion Matrix on Valid Data')
plt.show()
```

```
Confusion Matrix (Accuracy 0.7965)

       Prediction
Actual    0    1
     0 4750 1209
     1 1233 4808
              precision    recall  f1-score   support

           0       0.79      0.80      0.80      5959
           1       0.80      0.80      0.80      6041

    accuracy                           0.80     12000
   macro avg       0.80      0.80      0.80     12000
weighted avg       0.80      0.80      0.80     12000
```

Naive Bayes Classifiers



```
from sklearn.naive_bayes import MultinomialNB
NB_bayes = MultinomialNB(alpha=0.001)
NB_bayes.fit(X_train, y_train)
classificationSummary(y_valid, NB_bayes.predict(X_valid))
print(classification_report(y_valid, NB_bayes.predict(X_valid)))
# Plot the cumulative gains chart of the expected spending
gains_df = pd.DataFrame({'actual': y_valid,
                         'prob': NB_bayes.predict(X_valid)})
gains_df = gains_df.sort_values(by=['prob'], ascending=False).reset_index(drop=True)
gainsChart(gains_df.actual)
plt.show()

#
# Plot Confusion Matrix
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(NB_bayes, X_valid, y_valid)
plt.title('Confusion Matrix on Valid Data')
plt.show()
```