Machine Learning in Banking Sector

PROJECT OBJECTIVE:

Build a ML model to perform focused digital marketing by predicting the potential customers who will convert from liability customers to asset customers.

CONTEXT:

Bank XYZ has a growing customer base where majority of them are liability customers (depositors) vs borrowers (asset customers). The bank is interested in expanding the borrowers base rapidly to bring in more business via loan interests. A campaign that the bank ran in last quarter showed an average single digit conversion rate. In the last town hall, the marketing head mentioned that digital transformation being the core strength of the business strategy, how to devise effective campaigns with better target marketing to increase the conversion ratio to double digit with same budget as per last campaign. You as a data scientist asked to develop machine learning model to identify potential borrowers to support focused marketing.

DATA DESCRIPTION: The data consists of the following attributes:

1. ID: Customer ID
2. Age Customer's approximate age.
3. CustomerSince: Customer of the bank since. [unit is masked]
4. HighestSpend: Customer's highest spend so far in one transaction. [unit is masked]
5. ZipCode: Customer's zip code.
6. HiddenScore: A score associated to the customer which is masked by the bank as an IP.
7. MonthlyAverageSpend: Customer's monthly average spend so far. [unit is masked]
8. Level: A level associated to the customer which is masked by the bank as an IP.
9. Mortgage: Customer's mortgage. [unit is masked]
10. Security: Customer's security asset with the bank. [unit is masked]
11. FixedDepositAccount: Customer's fixed deposit account with the bank. [unit is masked]
12. InternetBanking: if the customer uses internet banking.
13. CreditCard: if the customer uses bank's credit card.
14. LoanOnCard: if the customer has a loan on credit card

In [168]:
```
#Import Libraries
```

In [169]:
```
# To enable plotting graphs in Jupyter notebook
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [170]:
```
# Load customer data present in CSV file
data1 = pd.read_csv("Data1.csv")
data2 = pd.read_csv("Data2.csv")
```

In [171]:
```
# Shape and size of data
print(data1.shape)
print(data2.shape)
```

```
(5000, 8)
(5000, 7)
```

In [172]:
```
# Merging two data frames. Use Pandas merge function to merge two data frames
cust_data=data1.merge(data2, how='inner', on='ID')
```

In [107]:
```
# Explore final shape of data
print(cust_data.shape)
```

```
(5000, 14)
```

In [108]:
```
# Explore data types
cust_data.dtypes
```

Out[108]:
```
ID                      int64
Age                     int64
CustomerSince           int64
HighestSpend            int64
ZipCode                 int64
HiddenScore             int64
MonthlyAverageSpend   float64
Level                   int64
Mortgage                int64
Security                int64
FixedDepositAccount     int64
InternetBanking         int64
CreditCard              int64
LoanOnCard            float64
dtype: object
```

Comment: As all data attributes are quantitative data, we don't need data transformation here

In [109]:
```
# Data description
cust_data.describe().transpose()
```

Out[109]:

	count	mean	std	min	25%	50%	7
ID	5000.0	2500.500000	1443.520003	1.0	1250.75	2500.5	3750
Age	5000.0	45.338400	11.463166	23.0	35.00	45.0	55
CustomerSince	5000.0	20.104600	11.467954	-3.0	10.00	20.0	30
HighestSpend	5000.0	73.774200	46.033729	8.0	39.00	64.0	98
ZipCode	5000.0	93152.503000	2121.852197	9307.0	91911.00	93437.0	94608
HiddenScore	5000.0	2.396400	1.147663	1.0	1.00	2.0	3
MonthlyAverageSpend	5000.0	1.937938	1.747659	0.0	0.70	1.5	2
Level	5000.0	1.881000	0.839869	1.0	1.00	2.0	3
Mortgage	5000.0	56.498800	101.713802	0.0	0.00	0.0	101
Security	5000.0	0.104400	0.305809	0.0	0.00	0.0	0
FixedDepositAccount	5000.0	0.060400	0.238250	0.0	0.00	0.0	0
InternetBanking	5000.0	0.596800	0.490589	0.0	0.00	1.0	1
CreditCard	5000.0	0.294000	0.455637	0.0	0.00	0.0	1
LoanOnCard	4980.0	0.096386	0.295149	0.0	0.00	0.0	0

In [110]:
```
# Dropping ID as it doesn't have any impact on learning
cust_data = cust_data.drop(columns='ID')
```

In [111]:
```
cust_data.shape
```

Out[111]:
```
(5000, 13)
```

In [112]:
```
# Check for null value
cust_data.isnull().sum()
```

Out[112]:
```
Age                     0
CustomerSince           0
HighestSpend            0
ZipCode                 0
HiddenScore             0
MonthlyAverageSpend     0
Level                   0
Mortgage                0
Security                0
FixedDepositAccount     0
InternetBanking         0
CreditCard              0
LoanOnCard             20
dtype: int64
```

Comment: LoanOnCard attribute has 20 null data, which is 0.4% only. Secondly, it is the target class hence we can't repplace null value using mean or mode. We can remove these data from our dataset.

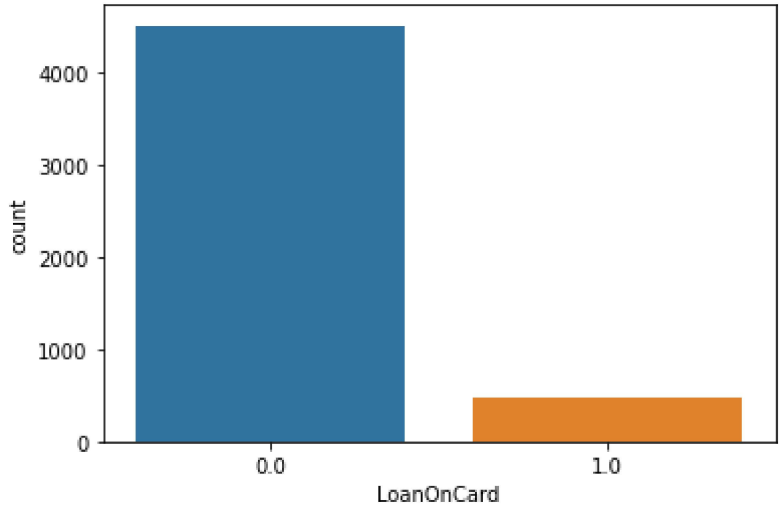
In [128]:
```
# Using Panda's dropna function to drop rows having null values.
cust_data = cust_data.dropna()
```

In [129]:
```
#Explore Size after null value removal
cust_data.shape
```

Out[129]: (4980, 11)

Exploratory Data Analysis

In [130]:
```
# Let explore how data is distributed as per target class.
sns.countplot(x = 'LoanOnCard',  data = cust_data);
```



This shows clearly data is highly imbalanced.

Calculate target class data percentage

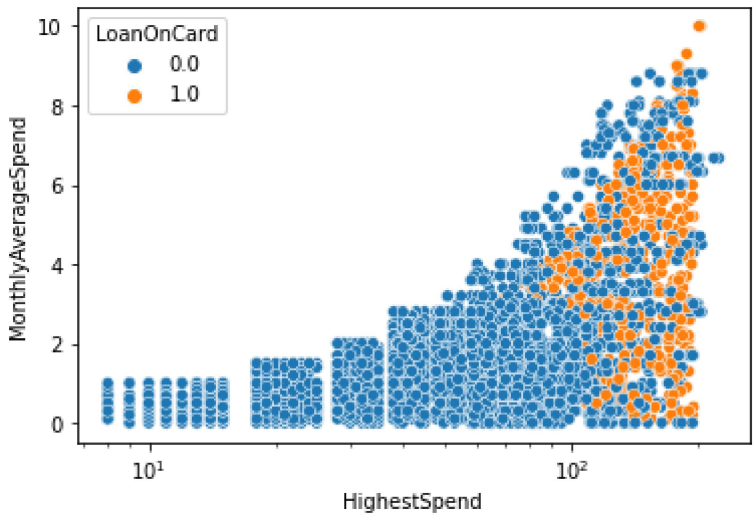
In [131]:
```
n_true = len(cust_data.loc[cust_data['LoanOnCard'] == 1.0])
n_false = len(cust_data.loc[cust_data['LoanOnCard'] == 0.0])
print("Number of true cases: {0} ({1:2.2f}%)".format(n_true, (n_true / (n_tru
print("Number of false cases: {0} ({1:2.2f}%)".format(n_false, (n_false / (n_
```

```
Number of true cases: 480 (9.64%)
Number of false cases: 4500 (90.36%)
```

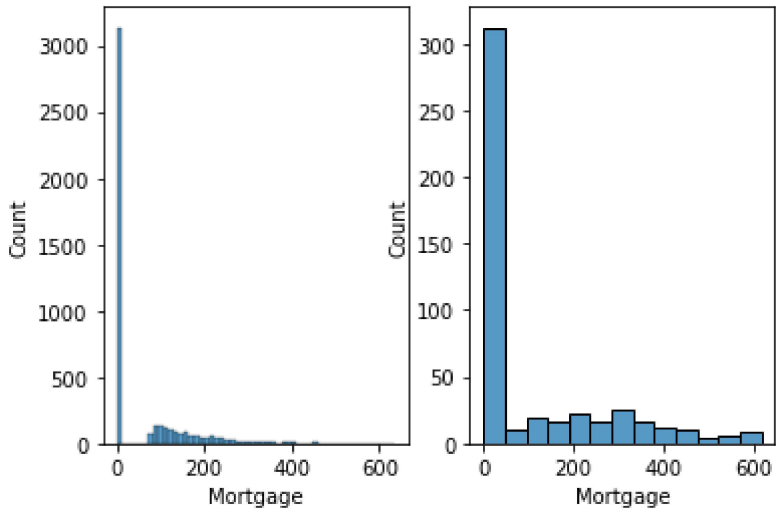
Comment: Data imbalance is a typical problem in machine learning. Later we shall use it's impact when we develop ML models.

In [132]:
```
# Scatter plot to see how data points are distributed for "MonthlyAverageSpen
g = sns.scatterplot(x="HighestSpend", y="MonthlyAverageSpend", hue="LoanOnCar
                    data=cust_data,legend='full')
g.set(xscale="log")
```

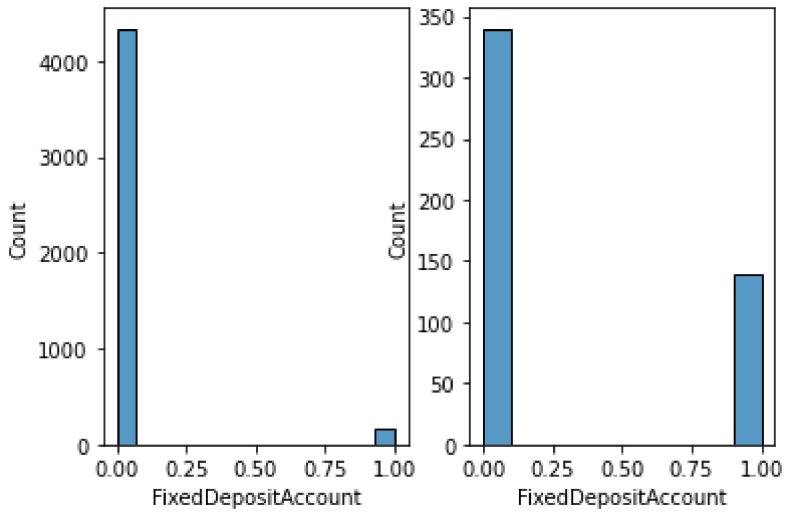
Out[132]: [None]



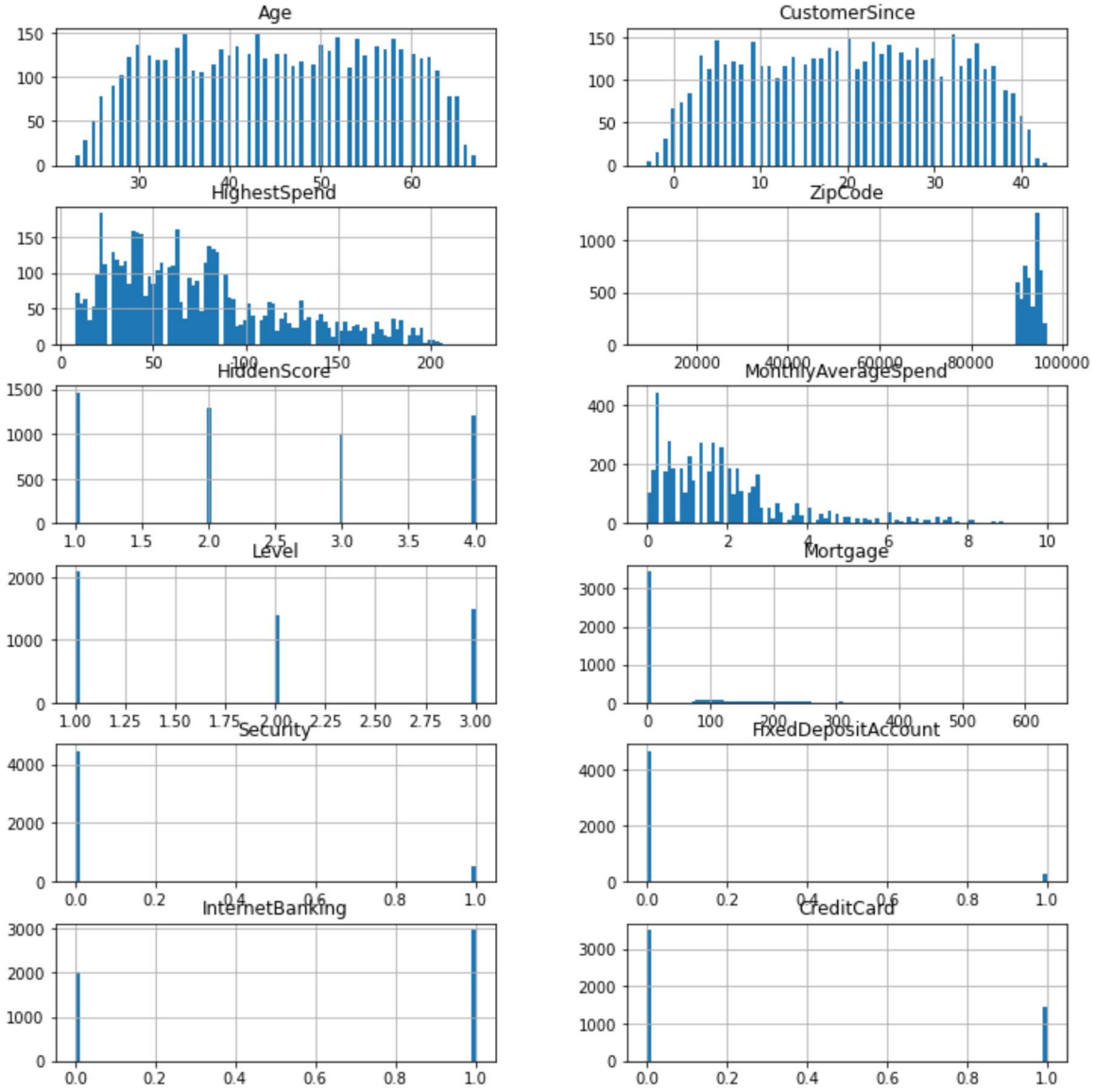
In [133]:
```
fig, ax = plt.subplots(1, 2)
sns.histplot(cust_data.loc[cust_data.LoanOnCard == 0.0, 'Mortgage'], ax = ax[
sns.histplot(cust_data.loc[cust_data.LoanOnCard == 1.0, 'Mortgage'], ax = ax[
plt.show()
```



In [119]:
```
fig, ax = plt.subplots(1, 2)
sns.histplot(cust_data.loc[cust_data.LoanOnCard == 0.0, 'FixedDepositAccount'
sns.histplot(cust_data.loc[cust_data.LoanOnCard == 1.0, 'FixedDepositAccount'
plt.show()
```

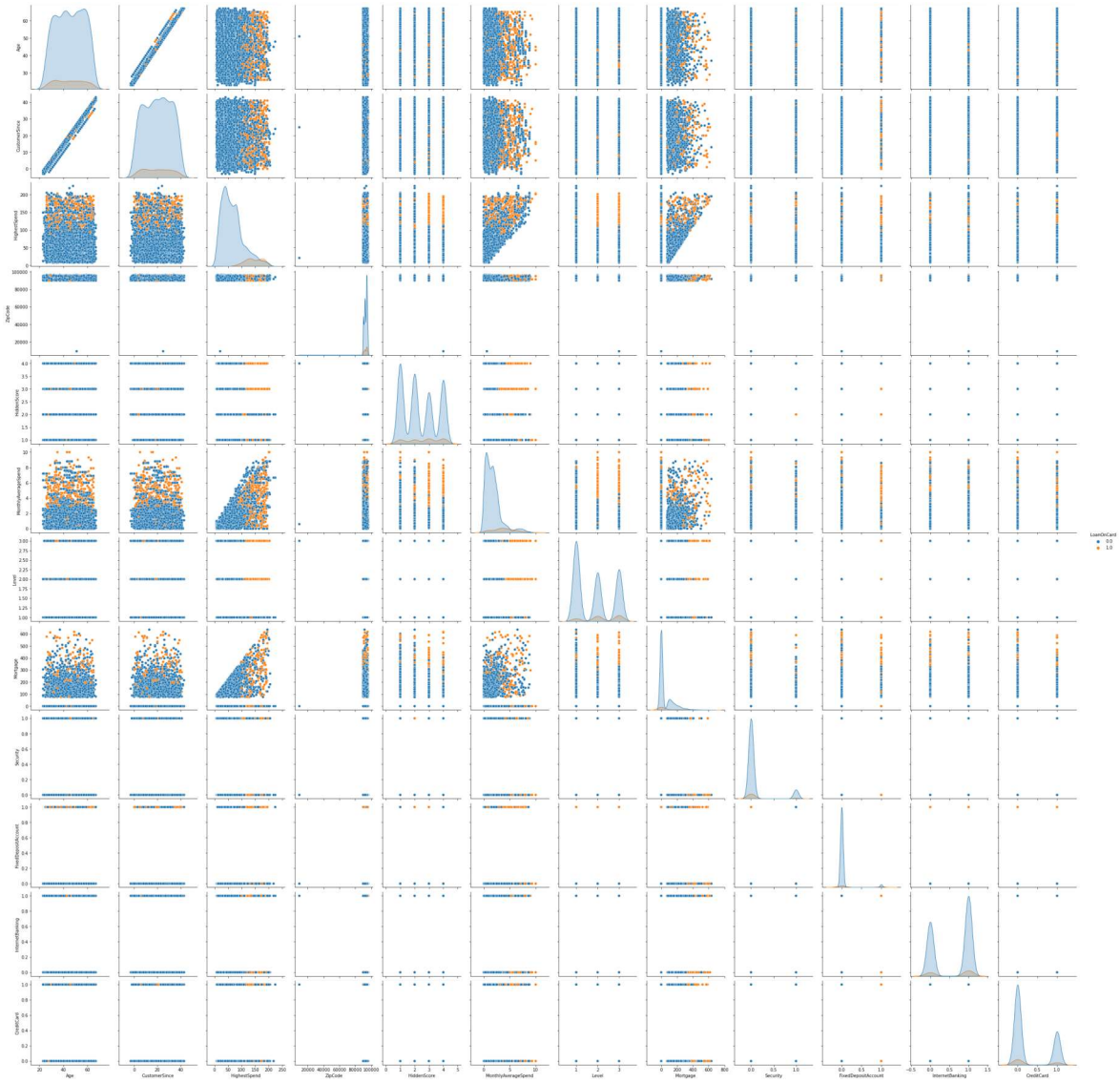


In [120]:
```
columns = list(cust_data)[0:-1] # Excluding Outcome column which has only
cust_data[columns].hist(stacked=False, bins=100, figsize=(12,30), layout=(14,
# Histogram of first 8 columns
```



In [121]:
```
sns.pairplot(cust_data, height=3, hue = 'LoanOnCard')
```

Out[121]: <seaborn.axisgrid.PairGrid at 0x1f7fed0dca0>



Zipcode doesn't have any significance with other dependant variables and on learning, hence drop it from dependant variable list.

Age and customer Since have similar information content. Will verify through correlation analysis

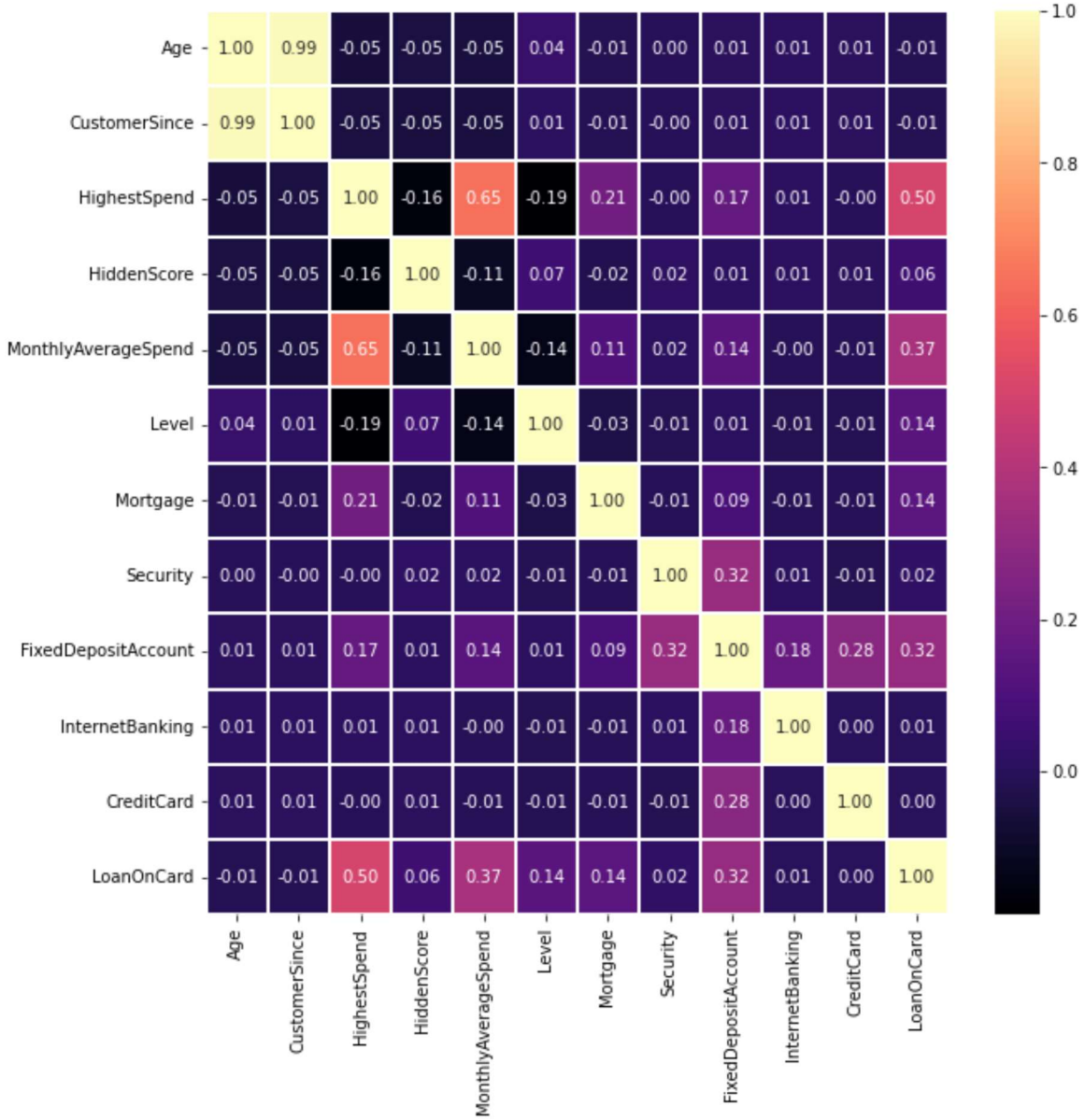
In [122]:
```
cust_data = cust_data.drop(columns='ZipCode')
```

In [123]:
```
#Correlation analysis
corr = cust_data.corr()
corr
```

Out[123]:

	Age	CustomerSince	HighestSpend	HiddenScore	MonthlyAverage
Age	1.000000	0.994208	-0.054951	-0.045289	-0.0
CustomerSince	0.994208	1.000000	-0.046092	-0.051456	-0.0
HighestSpend	-0.054951	-0.046092	1.000000	-0.158357	0.6
HiddenScore	-0.045289	-0.051456	-0.158357	1.000000	-0.1
MonthlyAverageSpend	-0.051896	-0.049918	0.646109	-0.109180	1.0
Level	0.042750	0.014545	-0.188909	0.065762	-0.1
Mortgage	-0.013272	-0.011380	0.207236	-0.021396	0.1
Security	0.000323	-0.000469	-0.002284	0.019061	0.0
FixedDepositAccount	0.007744	0.010085	0.169535	0.014327	0.1
InternetBanking	0.011227	0.011355	0.014202	0.010900	-0.0
CreditCard	0.007344	0.008779	-0.002780	0.010784	-0.0
LoanOnCard	-0.008147	-0.007801	0.502626	0.061761	0.3

In [124]:
```
#heatmap
fig,ax = plt.subplots(figsize=(10, 10))
sns.heatmap(cust_data.corr(), ax=ax, annot=True, linewidths=0.05, fmt= '.2f',
plt.show()
```



As "Age" and "customerSince" are highly correlated, we can drop 1. I am dropping "Age"

In [ ]:
```
cust_data = cust_data.drop(columns='Age')
```

In [ ]:
```
cust_data.shape
```

In [ ]:
```
cust_data.head(10)
```

Spliting the data

We will use 70% of data for training and 30% for testing.

In [ ]:
```
from sklearn.model_selection import train_test_split

X = cust_data.drop('LoanOnCard',axis=1)     # Predictor feature columns (8 X
Y = cust_data['LoanOnCard']   # Predicted class (1=True, 0=False) (1 X m)

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, rand
# 1 is just any random seed number

x_train.head()
```

Logistic Regression

In [ ]:
```
# import model and matrics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score,
# Fit the model on train
model = LogisticRegression(solver="liblinear")
model.fit(x_train, y_train)
#predict on test
y_predict = model.predict(x_test)
coef_df = pd.DataFrame(model.coef_)
coef_df['intercept'] = model.intercept_
print(coef_df)
```

In [ ]:
```
model_score = model.score(x_test, y_test)
print(model_score)
```

In [36]:
```
# performance
print(f'Accuracy Score: {accuracy_score(y_test,y_predict)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_test, y_predict)}')
print(f'Area Under Curve: {roc_auc_score(y_test, y_predict)}')
print(f'Recall score: {recall_score(y_test,y_predict)}')
print(f'Precision score: {precision_score(y_test,y_predict)}')
print(f'f1 score: {f1_score(y_test,y_predict)}')
```

```
Accuracy Score: 0.9451137884872824
Confusion Matrix:
[[1326   15]
 [  67   86]]
Area Under Curve: 0.7754529104706761
Recall score: 0.5620915032679739
Precision score: 0.8514851485148515
f1 score: 0.6771653543307087
```

For minority class, the above model is able to predict 86 correctly, out of 153. Although the accuracy is high, still the model is not a good model. We need to handle the unbalanced data

Weighted Logistic Regression to handle class inbalance

In [143]:
```
# define class weights
w = {0:1, 1:2}

# Fit the model on train
model_weighted = LogisticRegression(solver="liblinear", class_weight=w)
model_weighted.fit(x_train, y_train)
#predict on test
y_predict = model_weighted.predict(x_test)
```

In [144]:
```
print(f'Accuracy Score: {accuracy_score(y_test,y_predict)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_test, y_predict)}')
print(f'Area Under Curve: {roc_auc_score(y_test, y_predict)}')
print(f'Recall score: {recall_score(y_test,y_predict)}')
print(f'Precision score: {precision_score(y_test,y_predict)}')
print(f'f1 score: {f1_score(y_test,y_predict)}')
```

```
Accuracy Score: 0.9444444444444444
Confusion Matrix:
[[1305   36]
 [  47  106]]
Area Under Curve: 0.8329824099662237
Recall score: 0.6928104575163399
Precision score: 0.7464788732394366
f1 score: 0.71864406779661
```

Although the accuracy decreases, AUC and recall increases significantly, hence, it is a better model. Hence we select "model_weighted".

Train Naive bayes algorithm

In [145]:
```
from sklearn.naive_bayes import GaussianNB # using Gaussian algorithm from Na

# create the model
diab_model = GaussianNB()

diab_model.fit(x_train, y_train)
```

Out[145]:
```
GaussianNB()
```

Performance with training data

In [146]:
```
diab_train_predict = diab_model.predict(x_train)

from sklearn import metrics

print("Model Accuracy: {0:.4f}".format(metrics.accuracy_score(y_train, diab_t
print()
```

```
Model Accuracy: 0.8867
```

Performance with testing data

In [147]:
```
y_predict = diab_model.predict(x_test)

from sklearn import metrics

print("Model Accuracy: {0:.4f}".format(metrics.accuracy_score(y_test, y_predi
print()
```

```
Model Accuracy: 0.8809
```

In [148]:
```
# performance
print(f'Accuracy Score: {accuracy_score(y_test,y_predict)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_test, y_predict)}')
print(f'Area Under Curve: {roc_auc_score(y_test, y_predict)}')
print(f'Recall score: {recall_score(y_test,y_predict)}')
```

```
Accuracy Score: 0.8808567603748326
Confusion Matrix:
[[1230  111]
 [  67   86]]
Area Under Curve: 0.7396587270254859
Recall score: 0.5620915032679739
```

Use of class prior for inbalanced data

In [149]:
```
diab_model_cp = GaussianNB(priors=[0.1, 0.9])
#diab_model.class_prior_ = [0.9, 0.1]
diab_model_cp.fit(x_train, y_train.ravel())
y_predict = diab_model_cp.predict(x_test)
```

In [150]:
```
# performance
print(f'Accuracy Score: {accuracy_score(y_test,y_predict)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_test, y_predict)}')
print(f'Area Under Curve: {roc_auc_score(y_test, y_predict)}')
print(f'Recall score: {recall_score(y_test,y_predict)}')
```

```
Accuracy Score: 0.8172690763052208
Confusion Matrix:
[[1080  261]
 [  12  141]]
Area Under Curve: 0.8634688774838792
Recall score: 0.9215686274509803
```

Support Vector Machines

In [151]:
```
from sklearn import svm
clf = svm.SVC(gamma=0.25, C=10)
clf.fit(x_train , y_train)
y_predict = clf.predict(x_test)
```

In [152]:
```
### gamma is a measure of influence of a data point. It is inverse of distanc
### C is penalty of wrong classifications
```

In [153]:
```
print(f'Accuracy Score: {accuracy_score(y_test,y_predict)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_test, y_predict)}')
print(f'Area Under Curve: {roc_auc_score(y_test, y_predict)}')
print(f'Recall score: {recall_score(y_test,y_predict)}')
print(f'Precision score: {precision_score(y_test,y_predict)}')
print(f'f1 score: {f1_score(y_test,y_predict)}')
```

```
Accuracy Score: 0.8989290495314591
Confusion Matrix:
[[1337    4]
 [ 147    6]]
Area Under Curve: 0.5181164188270387
Recall score: 0.0392156862745098
Precision score: 0.6
f1 score: 0.0736196319018405
```

In [154]:
```
from scipy.stats import zscore
XScaled  = X.apply(zscore)  # convert all attributes to Z scale
XScaled.describe()
```

Out[154]:

	CustomerSince	HighestSpend	HiddenScore	MonthlyAverageSpend	Level	
count	4.980000e+03	4.980000e+03	4.980000e+03	4.980000e+03	4.980000e+03	4.9
mean	-8.493875e-17	-3.322420e-16	3.342262e-16	1.652404e-16	5.178366e-16	
std	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00	1.000100e+00	1.0
min	-2.015918e+00	-1.429540e+00	-1.216634e+00	-1.108414e+00	-1.048644e+00	
25%	-8.822859e-01	-7.565844e-01	-1.216634e+00	-7.083749e-01	-1.048644e+00	
50%	-1.026117e-02	-2.138784e-01	-3.448588e-01	-2.511878e-01	1.417474e-01	
75%	8.617636e-01	5.242016e-01	5.269162e-01	3.345832e-01	1.332139e+00	4.
max	1.995396e+00	3.259439e+00	1.398691e+00	4.606425e+00	1.332139e+00	5.6

In [155]:
```
x_trains, x_tests, y_trains, y_tests = train_test_split(XScaled, Y, test_size
```

In [156]:
```
clf = svm.SVC(gamma=0.25, C=10)
clf.fit(x_trains , y_trains)
y_predicts = clf.predict(x_tests)
```

In [157]:
```
print(f'Accuracy Score: {accuracy_score(y_tests,y_predicts)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_tests, y_predicts)}')
print(f'Area Under Curve: {roc_auc_score(y_tests, y_predicts)}')
print(f'Recall score: {recall_score(y_tests,y_predicts)}')
print(f'Precision score: {precision_score(y_tests,y_predicts)}')
print(f'f1 score: {f1_score(y_tests,y_predicts)}')
```

```
Accuracy Score: 0.9745649263721553
Confusion Matrix:
[[1332    9]
 [  29  124]]
Area Under Curve: 0.9018730534719481
Recall score: 0.8104575163398693
Precision score: 0.9323308270676691
f1 score: 0.8671328671328671
```

Decision Tree Classifier

In [158]:
```
# Build decision tree model
from sklearn.tree import DecisionTreeClassifier

dTree = DecisionTreeClassifier(criterion = 'gini', random_state=1)
dTree.fit(x_train, y_train)
```

Out[158]:
```
DecisionTreeClassifier

DecisionTreeClassifier(random_state=1)
```

In [159]:
```
# Scoring our DT
print(dTree.score(x_train, y_train))
print(dTree.score(x_test, y_test))
```

```
1.0
0.9812583668005355
```

In [160]:
```
y_predict = dTree.predict(x_test)
print(f'Accuracy Score: {accuracy_score(y_test,y_predict)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_test, y_predict)}')
print(f'Area Under Curve: {roc_auc_score(y_test, y_predict)}')
print(f'Recall score: {recall_score(y_test,y_predict)}')
print(f'Precision score: {precision_score(y_test,y_predict)}')
print(f'f1 score: {f1_score(y_test,y_predict)}')
```

```
Accuracy Score: 0.9812583668005355
Confusion Matrix:
[[1335    6]
 [  22  131]]
Area Under Curve: 0.9258674386980743
Recall score: 0.8562091503267973
Precision score: 0.9562043795620438
f1 score: 0.903448275862069
```

In [161]:
```
#Reducing over fitting (Regularization)
dTreeR = DecisionTreeClassifier(criterion = 'gini', max_depth = 5, random_sta
dTreeR.fit(x_train, y_train)
print(dTreeR.score(x_train, y_train))
print(dTreeR.score(x_test, y_test))
```

```
0.9905335628227194
0.9825970548862115
```

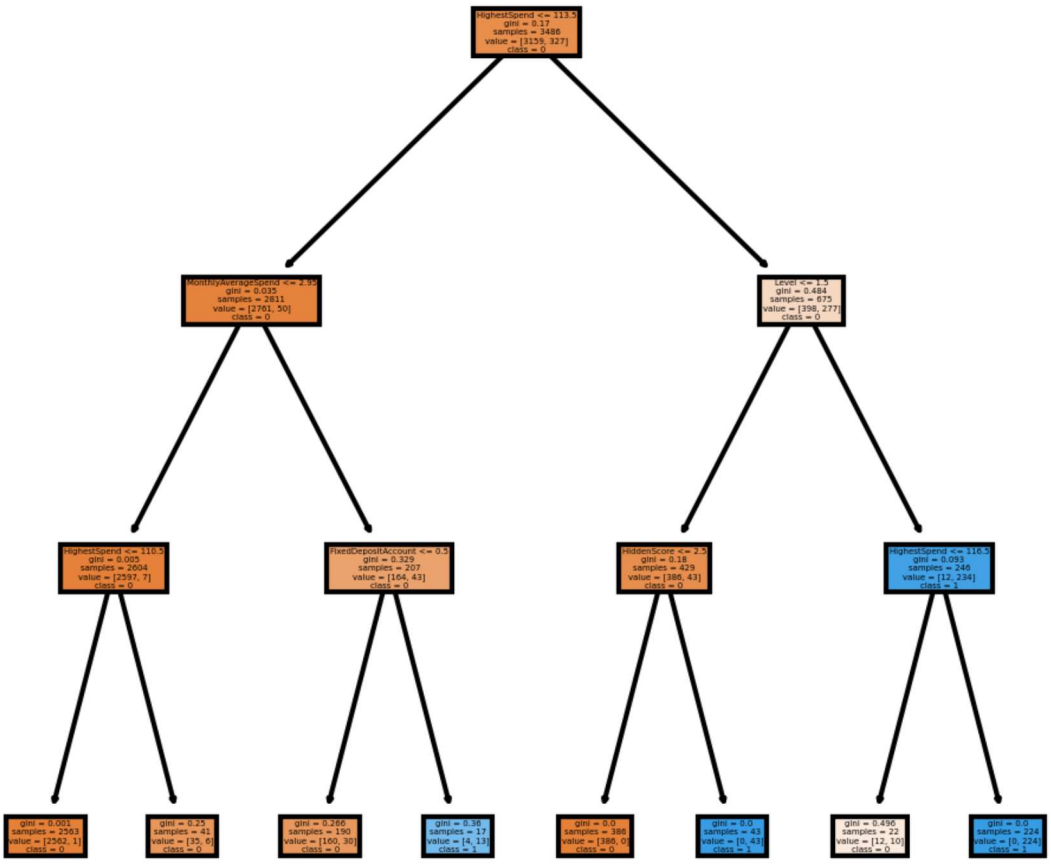
In [162]:
```
y_predictR = dTreeR.predict(x_test)
print(f'Accuracy Score: {accuracy_score(y_test,y_predictR)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_test, y_predictR)}')
print(f'Area Under Curve: {roc_auc_score(y_test, y_predictR)}')
print(f'Recall score: {recall_score(y_test,y_predictR)}')
print(f'Precision score: {precision_score(y_test,y_predictR)}')
print(f'f1 score: {f1_score(y_test,y_predictR)}')
```

```
Accuracy Score: 0.9825970548862115
Confusion Matrix:
[[1335    6]
 [  20  133]]
Area Under Curve: 0.9324033864104927
Recall score: 0.869281045751634
Precision score: 0.9568345323741008
f1 score: 0.910958904109589
```

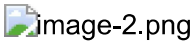
In [163]:

```
# Decision Tree Visualize
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
dTreeR3 = DecisionTreeClassifier(criterion = 'gini', max_depth = 3, random_st
dTreeR3.fit(x_train, y_train)
fn = list(x_train)
cn = ['0', '1']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4, 4), dpi=300)
plot_tree(dTreeR3, feature_names = fn, class_names=cn, filled = True)

fig.savefig('tree.png')
```



Ensemble Learning: Random forest classifier



In [164]:
```
from sklearn.ensemble import RandomForestClassifier
rfcl = RandomForestClassifier(random_state=1)
rfcl = rfcl.fit(x_train, y_train)
y_predict = rfcl.predict(x_test)
```

In [165]:
```
# performance
print(f'Accuracy Score: {accuracy_score(y_test,y_predict)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_test, y_predict)}')
print(f'Area Under Curve: {roc_auc_score(y_test, y_predict)}')
print(f'Recall score: {recall_score(y_test,y_predict)}')
print(f'Precision score: {precision_score(y_test,y_predict)}')
print(f'f1 score: {f1_score(y_test,y_predict)}')
```

```
Accuracy Score: 0.9859437751004017
Confusion Matrix:
[[1340    1]
 [  20  133]]
Area Under Curve: 0.934267666798263
Recall score: 0.869281045751634
Precision score: 0.9925373134328358
f1 score: 0.926829268292683
```

Unbalanced Data Handelling

In [60]:
```
# Install imbalanced-learn if you have not used before
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\admin\anaconda3
\lib\site-packages (0.9.1)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\admin\anacon
da3\lib\site-packages (from imbalanced-learn) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\anaco
nda3\lib\site-packages (from imbalanced-learn) (2.2.0)
Requirement already satisfied: joblib>=1.0.0 in c:\users\admin\anaconda3\li
b\site-packages (from imbalanced-learn) (1.1.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\admin\anaconda3\lib
\site-packages (from imbalanced-learn) (1.7.3)
Requirement already satisfied: numpy>=1.17.3 in c:\users\admin\anaconda3\li
b\site-packages (from imbalanced-learn) (1.21.5)
```

In [61]:
```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from collections import Counter
# summarize class distribution
counter = Counter(Y)
print(counter)
# define pipeline
over = SMOTE(sampling_strategy=0.3,random_state=1) #sampling_strategy=0.1,ran
under = RandomUnderSampler(sampling_strategy=0.5)
steps = [ ('o', over),('u', under)]
pipeline = Pipeline(steps=steps)
# transform the dataset
Xb, Yb = pipeline.fit_resample(XScaled, Y)
# summarize the new class distribution
counter = Counter(Yb)
print(counter)
```

```
Counter({0.0: 4500, 1.0: 480})
Counter({0.0: 2700, 1.0: 1350})
```

In [62]:
```
x_trainb, x_testb, y_trainb, y_testb = train_test_split(Xb, Yb, test_size=0.3
# 1 is just any random seed number
```

SVM with balanced Data

In [63]:
```
clf = svm.SVC(gamma=0.25, C=10)
clf.fit(x_trainb , y_trainb)
y_predictb = clf.predict(x_testb)
```

In [64]:
```
# performance
print(f'Accuracy Score: {accuracy_score(y_testb,y_predictb)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_testb, y_predictb)}')
print(f'Area Under Curve: {roc_auc_score(y_testb, y_predictb)}')
print(f'Recall score: {recall_score(y_testb,y_predictb)}')
print(f'Precision score: {precision_score(y_testb,y_predictb)}')
print(f'f1 score: {f1_score(y_testb,y_predictb)}')
```

```
Accuracy Score: 0.9777777777777777
Confusion Matrix:
[[797  18]
 [  9 391]]
Area Under Curve: 0.977707055214724
Recall score: 0.9775
Precision score: 0.9559902200488998
f1 score: 0.9666254635352286
```

Random Forest classifier with Balanced Data

In [65]:
```
rfcl = RandomForestClassifier(random_state=1)
rfcl = rfcl.fit(x_trainb, y_trainb)
y_predict = rfcl.predict(x_testb)
```

In [66]:
```
# performance
print(f'Accuracy Score: {accuracy_score(y_testb,y_predict)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_testb, y_predict)}')
print(f'Area Under Curve: {roc_auc_score(y_testb, y_predict)}')
print(f'Recall score: {recall_score(y_testb,y_predict)}')
print(f'Precision score: {precision_score(y_testb,y_predict)}')
print(f'f1 score: {f1_score(y_testb,y_predict)}')
```

```
Accuracy Score: 0.9901234567901235
Confusion Matrix:
[[812   3]
 [  9 391]]
Area Under Curve: 0.986909509202454
Recall score: 0.9775
Precision score: 0.9923857868020305
f1 score: 0.9848866498740554
```

Chosing hyperparameter using Grid Search

In [67]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.25,0.01],'kernel': ['r
grid = GridSearchCV(svm.SVC(),param_grid,refit=True,verbose=2)
grid.fit(x_trainb,y_trainb)
print(grid.best_estimator_)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=
0.9s
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=
0.8s
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=
0.9s
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=
0.8s
[CV] END .....................C=0.1, gamma=1, kernel=rbf; total time=
0.9s
[CV] END .....................C=0.1, gamma=1, kernel=poly; total time=
0.1s
[CV] END .....................C=0.1, gamma=1, kernel=poly; total time=
0.1s
[CV] END .....................C=0.1, gamma=1, kernel=poly; total time=
0.1s
[CV] END .....................C=0.1, gamma=1, kernel=poly; total time=
0.1s
```

Pickle the model

In [68]:
```
# Pickle model file
import pickle
filename = 'finalized_model.sav'
pickle.dump(rfcl, open(filename, 'wb'))
```

Load model from pickle file and use

In [69]:
```
# Checking the pickle model
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.predict(x_testb)
# performance
print(f'Accuracy Score: {accuracy_score(y_testb,result)}')
print(f'Confusion Matrix: \n{confusion_matrix(y_testb, result)}')
print(f'Area Under Curve: {roc_auc_score(y_testb, result)}')
print(f'Recall score: {recall_score(y_testb,result)}')
print(f'Precision score: {precision_score(y_testb,result)}')
print(f'f1 score: {f1_score(y_testb,result)}')
```

```
Accuracy Score: 0.9901234567901235
Confusion Matrix:
[[812   3]
 [  9 391]]
Area Under Curve: 0.986909509202454
Recall score: 0.9775
Precision score: 0.9923857868020305
f1 score: 0.9848866498740554
```

Conclusion:

We have built a model using logistic regression, Support vector machine and Random forest classifier. This data set is highly imbalance hence accuracy can't a good measure, Hence we have used precision, Recall, and AUC for determining better model.

We use class weight technique to handle un balanced data and observe that the model performance improved by considering class weight.

Scaling/data transformation plays a major role when we work on SVM.

We have also explored undersampling and oversampling technique like SMOTE to handle data imbalance.

Hyper parameter tuning using Grid Search

We have also seen how to systematically improve a model.

In [ ]:
```
```