Credit Risk Report

-“Crafting a New Strategy Through Machine Learning”

Presenters

Arjun Madhusoodanan

Prannay Khushalani

Ragunath Natarajan

Executive Summary

- Empowering the credit approval process using Artificial Intelligence
- Our data-driven model minimizes defaults and maximizes revenue

Strategy type	Threshold	Train			Test 1			Test 2		
		# Total	Default Rate	Revenue	# Total	Default Rate	Revenue	# Total	Default Rate	Revenue
Aggressive Strategy	0.72	52,366	10%	2.3 B	11,350	13%	0.5 B	11,323	13%	0.5 B
Conservative Strategy	0.20	41,379	1%	1.8 B	8,831	3%	0.4 B	8,824	3%	0.4 B

Data

- Dataset includes credit card applications with historical data ranging from 1 to 13 months.

- April 2018 originations: Start of financial year Selection of April 2018 originations enables analysis of credit risk dynamics over time.

- Goal is to develop precise predictive models for strategic decision-making.

- Aim is to mitigate risk and optimize opportunities in credit card management

All Applications with ‘n’ months of historical data	No of observations	Default rate
13	77262	23.16%
12	2106	38.13%
11	1216	43.42%
10	1279	47.77%
9	1306	46.94%
8	1188	43.52%
7	1058	43.67%
6	1096	40.05%
5	931	40.71%
4	950	39.79%
3	1155	36.19%
2	1196	31.61%
1	1039	33.21%

Features

- Dataset includes diverse range of raw features categorized into six key groups: Delinquency, Spend, Payment, Balance, Risk, and Other
- Features offer insights into credit card holder behavior and risk
- Objective is to develop a more accurate and effective credit card strategy

Feature category	No. of features
Delinquency	96
Spend	22
Payment	3
Balance	40
Risk	28
Other (Category, ID)	2

Feature Engineering

Feature category	No. of Engineered features
Delinquency	122
Spend	22
Payment	3
Balance	48
Risk	28

Numeric Features
Average of all the months data available for the customers

Categorical Features
Percentage of times the one-hot encoded feature columns are 1 in the last 13 months

Summary statistics for the top 5 features by SHAP values in final XGBoost model									
Features	min	1 Percentile	5 Percentile	Median	99 Percentile	max	mean	std	% missing
P_2_12Mo_Avg	-0.34	0.0651	0.2370	0.6778	1.00	1.01	0.65	0.24	0.01
B_1_12Mo_Avg	-0.12	0.0039	0.0049	0.0386	0.93	1.32	0.13	0.20	0.00
R_1_12Mo_Avg	0.00	0.0029	0.0038	0.0059	0.75	1.60	0.08	0.16	0.00
D_44_12Mo_Avg	0.00	0.0031	0.0039	0.0072	0.95	3.00	0.12	0.21	0.04
D_46_12Mo_Avg	-1.25	0.1840	0.3542	0.4611	0.86	2.83	0.48	0.11	0.17

Data Processing/ One hot encoding

```python
# Finding the number of unique values in categorical column mentioned in the data set
for i in ['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_66', 'D_68']:
    print(f'No. of unique values in {i}: {master_2575[i].nunique()} ---- {master_2575[i].unique()}')
```

```
No. of unique values in B_30: 3 ---- [ 0.  1.  2. nan]
No. of unique values in B_38: 7 ---- [ 1.  3.  2.  7.  6.  4.  5. nan]
No. of unique values in D_114: 2 ---- [ 1.  0. nan]
No. of unique values in D_116: 2 ---- [ 0. nan  1.]
No. of unique values in D_117: 7 ---- [-1.  4.  3.  5.  6. nan  2.  1.]
No. of unique values in D_120: 2 ---- [ 0. nan  1.]
No. of unique values in D_126: 3 ---- [ 0.  1. -1. nan]
No. of unique values in D_63: 6 ---- ['CO' 'CR' 'CL' 'XZ' 'XM' 'XL']
No. of unique values in D_64: 4 ---- ['O' 'U' 'R' nan '-1']
No. of unique values in D_66: 2 ---- [nan  1.  0.]
No. of unique values in D_68: 7 ---- [ 6.  5.  2.  3. nan  4.  1.  0.]
```

```python
dummy_df = pd.get_dummies(cat_2575[['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63','D_64', 'D_66', 'D_68']])
```

	B_30_0.0	B_30_1.0	B_30_2.0	B_38_1.0	B_38_2.0	B_38_3.0	B_38_4.0	B_38_5.0	B_38_6.0	B_38_7.0	...	D_64_U	D_66_0.0	D_66_1.0	D_68_0.0	D_68_1.0
0	True	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False
1	True	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False

```python
dummy_df = dummy_df.astype(int)
# Converting the boolean type to binary 1 and 0
```

```python
dummy_df.head()
```

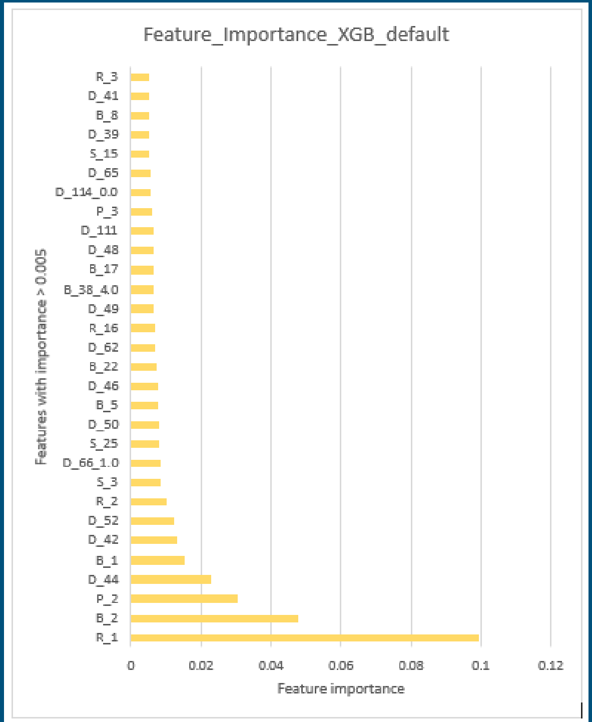
	B_30_0.0	B_30_1.0	B_30_2.0	B_38_1.0	B_38_2.0	B_38_3.0	B_38_4.0	B_38_5.0	B_38_6.0	B_38_7.0	...	D_64_U	D_66_0.0	D_66_1.0	D_68_0.0	D_68_1.0
0	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0
2	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0

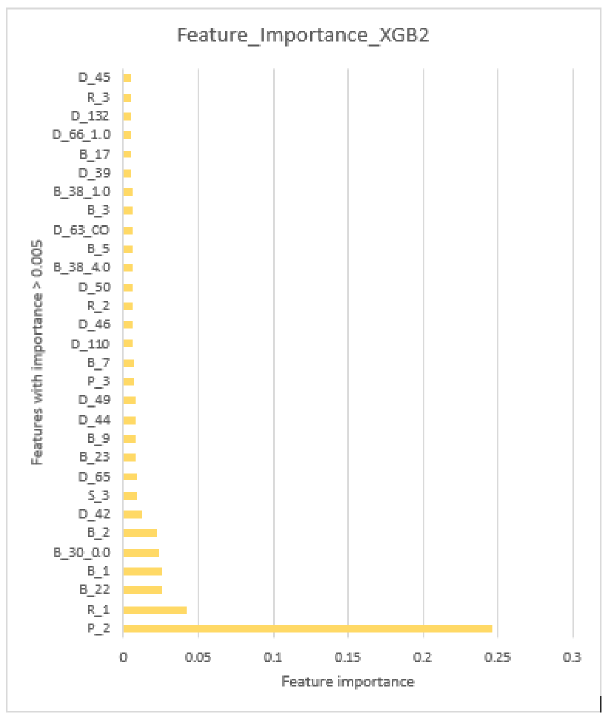
11 Categorical features processed with One-Hot Encoding

The datatypes of some categorical columns were converted from numerical to categorical

45 Indicator features created retaining the original categorical feature name suffixed with discrete values from each category

Feature Selection





Feature importance was calculated from two XGBoost models trained on the training dataset

Selected features have a feature importance of higher than 0.5% in any of the two models.

Feature category	No. of Engineered features	Selected features
Delinquency	122	18
Spend	22	3
Payment	3	2
Balance	48	13
Risk	28	4

Total features selected: 40

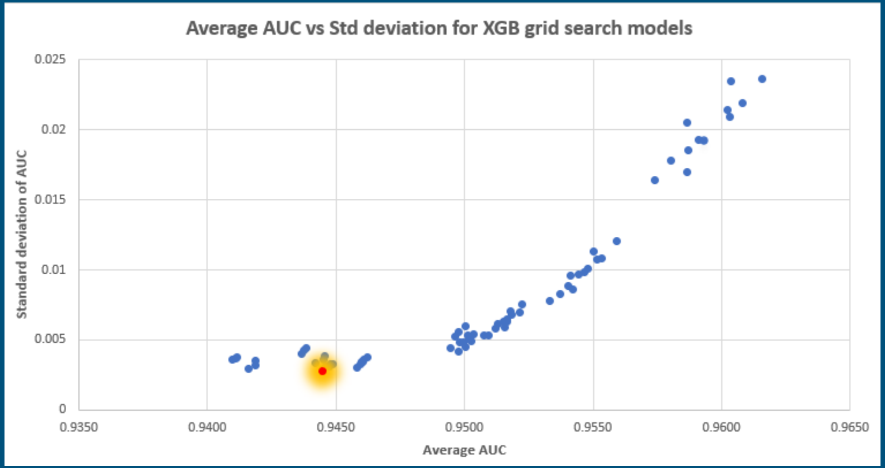
Script: CRisk_XGBmodel_26Mar2000, Feat_imp_XGB2

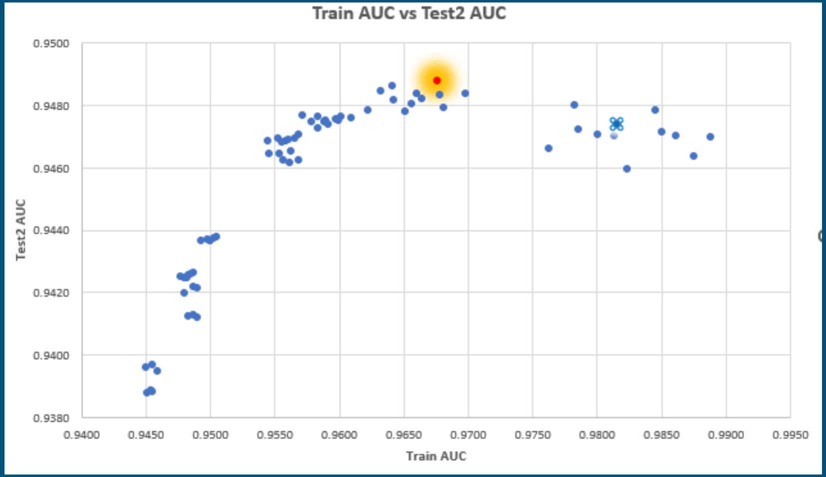
XGBoost-Grid Search

Hyperparameters	Description
n_estimators	No of trees
learning_rate	Learning rate
subsample	Percentage of observations used in each tree
colsample_bytree	Percentage of features used in each tree
min_child_weight	Weight of default observations

```python
# Doing grid search for building the best XGB using the shortlisted features

Grid_Search_Results = pd.DataFrame(columns = ["Model Number", "Number Trees", "Learning Rate", "Percent Observations",
                                              "Percent Features", 'Weight of observations',"AUC Train", "AUC Test 1",
                                              "AUC Test 2"])

Counter = 0
for num_trees in [50, 100, 300]:
    for learning_rate in [0.01, 0.1]:
        for Per_obs in [0.5,0.8]:
            for Per_feat in [0.5,1]:
                for Weit_obs in [1,5,10]:
                    xgb_instance = XGBClassifier(n_estimators=num_trees, learning_rate=learning_rate, subsample=Per_obs, colsample_bytree=Per_feat, min_child_
                    model = xgb_instance.fit(X_train_xgb3, y_train_xgb3)

                    Grid_Search_Results.loc[Counter,"Model Number"] = Counter
                    Grid_Search_Results.loc[Counter,"Number Trees"] = num_trees
                    Grid_Search_Results.loc[Counter,"Learning Rate"] = learning_rate
                    Grid_Search_Results.loc[Counter,"Percent Observations"] = Per_obs
                    Grid_Search_Results.loc[Counter,"Percent Features"] = Per_feat
                    Grid_Search_Results.loc[Counter,"Weight of observations"] = Weit_obs
                    Grid_Search_Results.loc[Counter,"AUC Train"] = roc_auc_score(y_train_xgb3, model.predict_proba(X_train_xgb3)[:,1])
                    Grid_Search_Results.loc[Counter,"AUC Test 1"] = roc_auc_score(Y_xgb3_test_1, model.predict_proba(X_xgb3_test_1)[:,1])
                    Grid_Search_Results.loc[Counter,"AUC Test 2"] = roc_auc_score(Y_xgb3_test_2, model.predict_proba(X_xgb3_test_2)[:,1])

                    Counter = Counter + 1
```

XGBoost-Grid Search





Best model selected

Model Number	Number Trees	Learning Rate	Percent Observations	Percent Features	Weight of observations	AUC Train	AUC Test 1	AUC Test 2	Avg AUC	Std AUC
✓ 2	50	0.01	0.5	0.5	10	0.9477	0.9433	0.9425	0.9445	0.002786
46	100	0.1	0.8	1	5	0.9676	0.9491	0.9488	0.9551	0.010744

Script: xgb3_grid_results

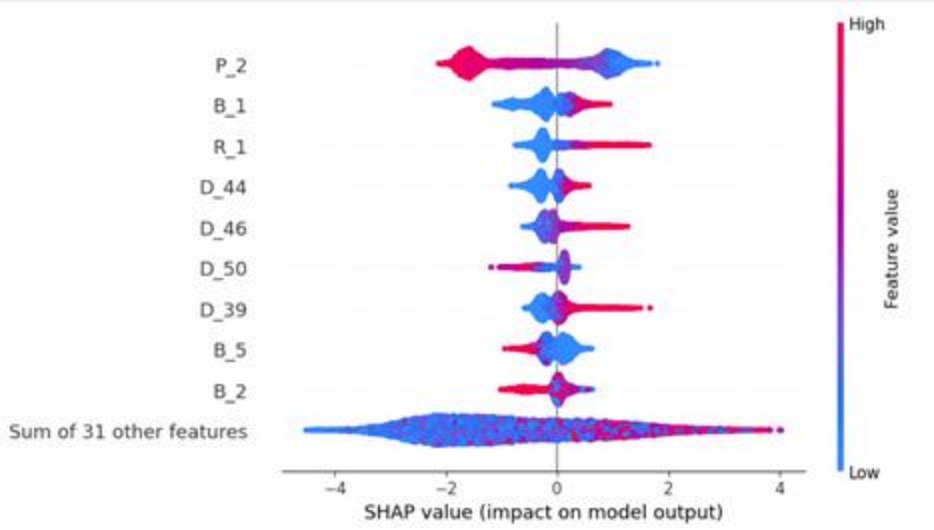
XGBoost- Final Model

Parameters	Values
Number of trees	50
Learning Rate	0.01
Percent Observations	0.8
Percent features	1
Weight of Observations	10

Area Under ROC Curve	
Train	0.94
Test1	0.94
Test2	0.94

XGBoost-SHAP Analysis

Top 5 Features based on SHAP Analysis
P_2
B_1
R_1
D_44
D_46

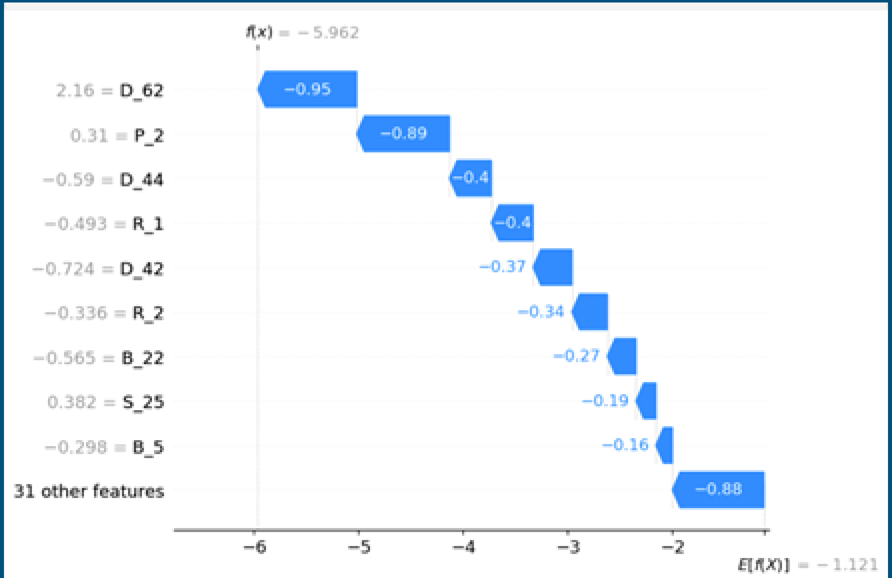


XGBoost-SHAP Analysis

- D_62 and P_2 are the most influential variables with a negative impact of 0.95 and 0.89 respectively.
- All the Variables here show a negative impact

To increase score:

- Selecting Relevant features
- Optimizing Model Parameters
- Iterative model improvement



Neural Network- Data Processing

1	**Outlier treatment**	• Capping the numerical features at 1 and 99 percentile
2	**Feature scaling**	• Standardizing the features • z = (x - u) / s
3	**Missing value imputation**	• Replace the missing values with 0

```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)

# scale features
X_train = pd.DataFrame(sc.transform(X_train), columns = X_train.columns)
```

```python
outlier = pd.DataFrame(columns = ["Column Name", "P1", "P99"])

counter = 0
for feature in num_feat:
  outlier.loc[counter, "Column Name"] = feature
  outlier.loc[counter, "P1"] = X_train[feature].quantile(0.01)
  outlier.loc[counter, "P99"] = X_train[feature].quantile(0.99)
  counter = counter + 1
```

```python
# For missing value imputation, we replace all missing values with 0
X_train.fillna(0,inplace=True)
X_test_1.fillna(0,inplace=True)
X_test_2.fillna(0,inplace=True)
```

```python
for counter in range (outlier.shape[0]):
  X_train[outlier.loc[counter, "Column Name"]] = np.where(X_train[outlier.loc[counter, "Column Name"]] < outlier.loc[counter, "P1
                                                 outlier.loc[counter, "P1"], X_train[outlier.loc[counter, "Column Name"]])

  X_train[outlier.loc[counter, "Column Name"]] = np.where(X_train[outlier.loc[counter, "Column Name"]] > outlier.loc[counter, "P9
                                                 outlier.loc[counter, "P99"], X_train[outlier.loc[counter, "Column Name"]])
```

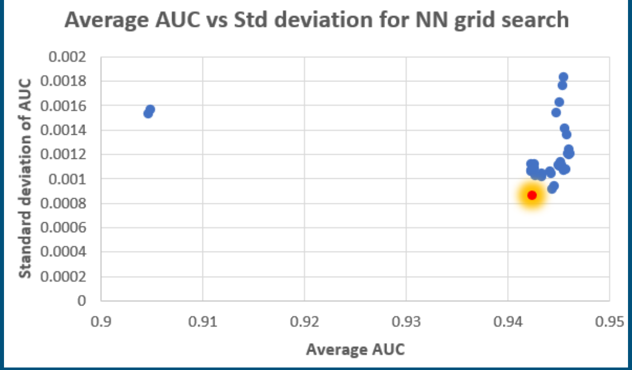
Neural Network- Grid Search

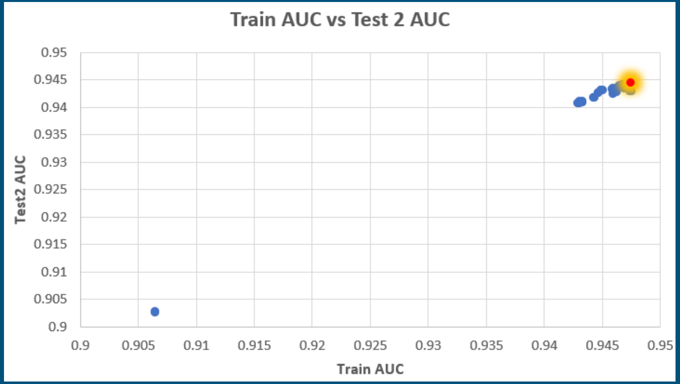
Parameters:

1. Number of Nodes: Varying the number in the hidden layers allows us to control the model's capacity to capture complex patterns in data
2. Activation Function- The choice of function influences the non-linear transformation applied to input data.
3. Hidden Layers- this determines the depth of the neural network and its ability to learn hierarchical representations of data.
4. Dropout- This is a regularization technique that will randomly deactivate a fraction of neurons during training, thus preventing overfitting.
5. Batch Size- This defines the number of samples processed before updating the model's parameters, and affects the optimization process and computational efficiency.

```python
Grid_Search_Results = pd.DataFrame(columns = ["Model Number", "Number of Nodes", "Activation Function", "Hidden layers",
                                              "Drop out","Batch Size","AUC Train", "AUC Test 1", "AUC Test 2"])

Counter = 0
for num_nodes in [4,6]:
    for activation in ['relu','tanh']:
        for hidden_layer in [2,4]:
            for dropout in [0.5,0]:
                model=Sequential()
                for _ in range(hidden_layer):
                    model.add(Dense(units=num_nodes, kernel_initializer='glorot_uniform',activation = activation))
                    model.add(Dropout(dropout))

                model.add(Dense(units=1,kernel_initializer='glorot_uniform', activation = 'sigmoid'))
                model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy', 'FalseNegatives'])

                for batch_size in [100,10000]:
                    model.fit(X_train,Y_train,batch_size=batch_size,epochs=20,verbose=0)

                    Grid_Search_Results.loc[Counter,"Model Number"] = Counter
                    Grid_Search_Results.loc[Counter,"Number of Nodes"] = num_nodes
                    Grid_Search_Results.loc[Counter,"Activation Function"] = activation
                    Grid_Search_Results.loc[Counter,"Hidden layers"] = hidden_layer
                    Grid_Search_Results.loc[Counter,"Drop out"] = dropout
                    Grid_Search_Results.loc[Counter,"Batch Size"] = batch_size

                    Grid_Search_Results.loc[Counter,"AUC Train"] = roc_auc_score(Y_train, model.predict(X_train))
                    Grid_Search_Results.loc[Counter,"AUC Test 1"] = roc_auc_score(Y_test_1, model.predict(X_test_1))
                    Grid_Search_Results.loc[Counter,"AUC Test 2"] = roc_auc_score(Y_test_2, model.predict(X_test_2))

                    Counter = Counter + 1


# Number of hidden layers: 2, 4
# Nodes in each hidden layer: 4, 6
# Activation function for hidden layers: ReLu, Tanh
# Dropout regularization for hidden layers: 50%, 100% (no dropout)
# Batch size: 100, 10000
# Epoch: 20
```

Neural Network- Grid Search





Model Number	Number of Nodes	Activation Function	Hidden layers	Drop out	Batch Size	AUC Train	AUC Test 1	AUC Test 2	Avg AUC	Std AUC
23	6	relu	4	0	10000	0.947	0.946	0.944	0.946	0.001212
24	6	tanh	2	0.5	100	0.943	0.943	0.941	0.942	0.000865

Excel: NN_Gridsearch1

Neural Network- Final Model

Parameter	Values
No. of Hidden layers	2
Nodes in each hidden layers	6
Activation function for hidden layers	Tanh
Dropout regularization for hidden layers	50%
Batch size	100
Epoch	20

Area Under Curve	
Train	0.943
Test1	0.942
Test2	0.941

Final Model

- We are choosing XGBoost as the final model as this gives consistent results
- Neural network AUC values are fluctuating when it is run separately
- XGBoost is more robust as can handle NA values and doesn’t require outlier treatment and feature scaling

```
In [23]: xgb4 = XGBClassifier(n_estimators=50,learning_rate=0.01,subsample=0.5,colsample_bytree=0.5,min_child_weight=10)

         xgb4.fit(X_train, y_train)

C:\ProgramData\Anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning: The use of label encoder in XGBClassifier is de
precated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=Fa
lse when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_
class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[05:24:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.

Out[23]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                       colsample_bynode=1, colsample_bytree=0.5, gamma=0, gpu_id=-1,
                       importance_type='gain', interaction_constraints='',
                       learning_rate=0.01, max_delta_step=0, max_depth=6,
                       min_child_weight=10, missing=nan, monotone_constraints='()',
                       n_estimators=50, n_jobs=4, num_parallel_tree=1, random_state=0,
```

Strategy

- Optimum threshold depends on True positive, True negative, False positive & False negative
- Features S_5 (0.088) & B_23 (0.18) were used as estimate of monthly spend & monthly balance.
- Estimated average monthly revenue = (0.088 * 0.02 + 0.18 * 0.001) * 10^6 = 3688 dollars per month per customer
- Estimate the default rate and annual revenue based on number of non-default customers for various thresholds and determine the conservative and aggressive threshold

Strategy

```python
# Function that calculates the default rate and average annual revenue based on threshold input for the probability of default
def defrate_Monrevenue (threshold,actual_y,pred_prob, est_mon_bal=0.18, est_mon_spe=0.088):
    prob_to_binary = pred_prob.apply(lambda x: 0 if x < threshold else 1)
    tot_cus_CC_approved = (prob_to_binary==0).sum()
    def_cus = ((actual_y==1) & (prob_to_binary==0)).sum()
    non_def_cus = ((actual_y==0) & (prob_to_binary==0)).sum()
    def_rate = ((actual_y==1) & (prob_to_binary==0)).sum() * 100 / (prob_to_binary==0).sum()

    annual_revenue = (prob_to_binary==0).sum() * 12 * (est_mon_bal*0.02 + est_mon_spe * 0.001) * 10**6
    return (tot_cus_CC_approved, def_cus, non_def_cus, def_rate, annual_revenue)
```

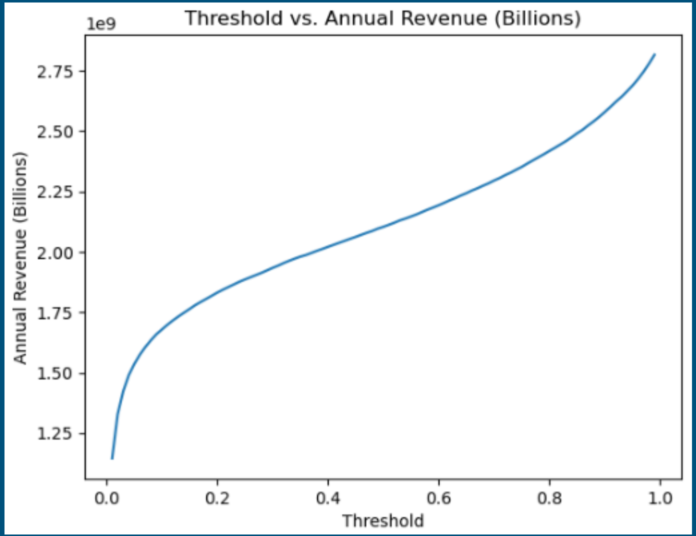
```python
test_thresh = [i / 100 for i in range(1, 73)]  # Range from 0.01 to 0.72 in steps of 0.01
```

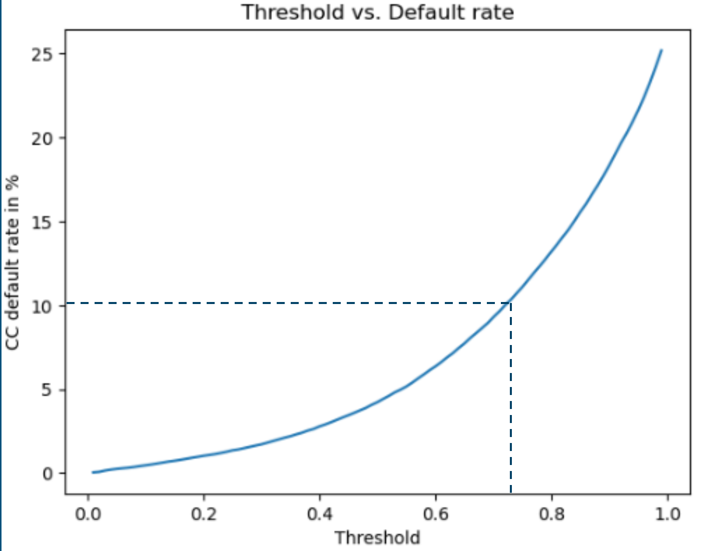
```python
counter = 0

for threshold1 in test_thresh:
    tot_customers, def_customers, non_def_cus, def_rate, annual_revenue = defrate_Monrevenue(threshold1, XGB_actual_predprob['Act
    strategy_gridsearch.loc[counter,'Threshold']= threshold1
    strategy_gridsearch.loc[counter,'Def_Rate']= def_rate
    strategy_gridsearch.loc[counter,'Annual_Revenue']= annual_revenue
    strategy_gridsearch.loc[counter,'Total_customers']= tot_customers
    strategy_gridsearch.loc[counter,'Defaulted_customers']= def_customers
    strategy_gridsearch.loc[counter,'NonDefaulted_customers']= non_def_cus

    counter = counter + 1
```

```python
strategy_gridsearch
```

	Threshold	Def_Rate	Annual_Revenue	Total_customers	Defaulted_customers	NonDefaulted_customers
0	0.01	0.042525	1144769952.0	25867	11	25856

Strategy





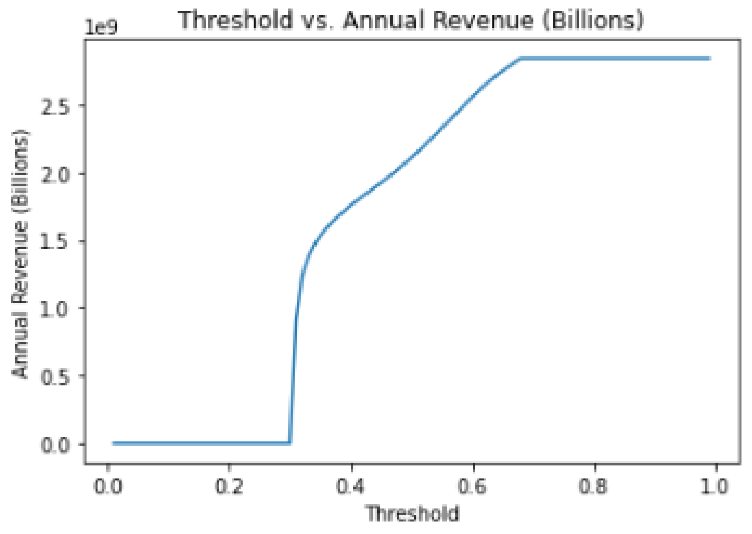
- The annual revenue increases rapidly initially, then proceeds slowly and again picking up momentum at the end
- Default rate < 10% constraint

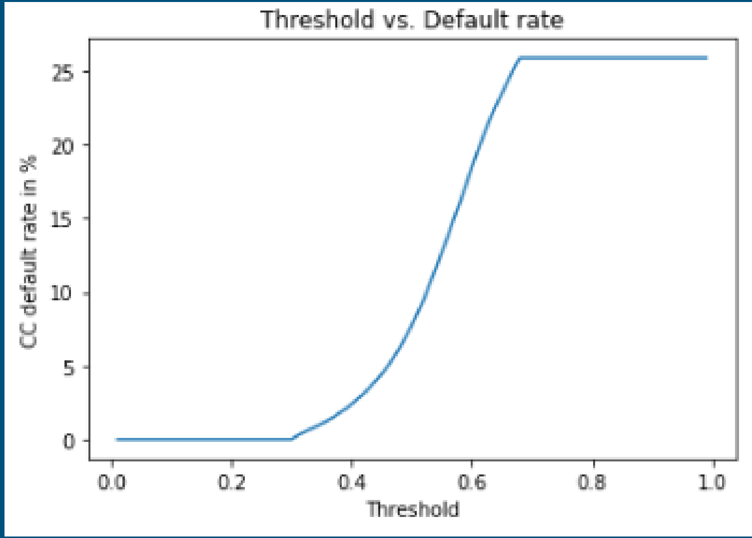
Conclusion

Based on train set		Default rate	Annual revenue
Aggressive threshold	0.72	~10%	2.3 B
Conservative threshold	0.2	~1%	1.8 B

Back up slides

Strategy





- The annual revenue increases rapidly initially, then proceeds slowly and again picking up momentum at the end
- Default rate < 10% constraint

Strategy

```
# Function that calculates the default rate and average annual revenue based on threshold input for the probability of default
def defrate_Monrevenue (threshold,actual_y,pred_prob, est_mon_bal=0.18, est_mon_spe=0.088):
    prob_to_binary = pred_prob.apply(lambda x: 0 if x < threshold else 1)
    tot_cus_CC_approved = (prob_to_binary==0).sum()
    def_cus = ((actual_y==1) & (prob_to_binary==0)).sum()
    non_def_cus = ((actual_y==0) & (prob_to_binary==0)).sum()
    def_rate = ((actual_y==1) & (prob_to_binary==0)).sum() * 100 / (prob_to_binary==0).sum()

    annual_revenue = (prob_to_binary==0).sum() * 12 * (est_mon_bal*0.02 + est_mon_spe * 0.001) * 10**6
    return (tot_cus_CC_approved, def_cus, non_def_cus, def_rate, annual_revenue)
```