Credit Risk Analysis

Implementation of Machine Learning Algorithms for strategizing on a credit risk policy.

Learning Algorithms Used-

XGBoost

&

Neural Networks

Project By-

Parth Ghumare
Sakshi Saxena
Sumana Akavaram

Executive Summary

	Train			Test1			Test 2			Overall		
	#Accepted	Default Rate	Revenue	#Accepted	Default Rate	Revenue	#Accepted	Default Rate	Revenue	#Accepted	Default Rate	Revenue
Conservative 0.3	28208	0.037	51.49	6003	0.065	9.68	7289	0.042	12.55	41500	0.042	73.73
Aggressive 0.443	31030	0.063	63.822	0.443	0.099	11.96	8130	0.072	15.80	45764	0.070	91.58

Use Case: We propose 2 strategies, Aggressive and Conservative wherein we take different thresholds to accept customers to propose the trade off between risk and revenue.

1. In our conservative strategy, we keep our threshold to accept customers with probability of default under 0.3 and display revenue
2. In our aggressive strategy, we keep our threshold to accept customers with probability of default under 0.443 and display revenue

It is observed that with the Aggressive strategy although the revenue is high, risk of defaulting is also higher and hence a company can analyze the risk to revenue tradeoff and decide which strategy to adopt.

Data

- The aim of this project is to use the monthly customer profile data to forecast the likelihood that a customer will default on their credit card balance in the future.
- The target variable is "1" if the customer defaults, else "0".
- A default event occurs if the customer does not make the required payment within 120 days of the date of their most recent statement.

Month	# of observations	Default Rate
2017 / 3	5639	24.24%
2017 / 4	5659	23.20%
2017 / 5	5635	24.13%
2017 / 6	5785	23.56%
2017 / 7	5921	26.46%
2017 / 8	5957	25.15%
2017 / 9	6116	25.31%
2017 / 10	6292	25.94%
2017 / 11	6338	26.44%
2017 / 12	6632	27.43%
2018 / 1	6921	28.32%
2018 / 2	7575	27.96%
2018 / 3	8505	28.46%
Total	82975	26.08%

Features

Categories	No.of features
Delinquency Variables	96
Spend Variables	22
Payment Variables	3
Balance Variables	40
Risk Variables	28

	Feature	Minimum value	Maximum value	1 percentile	5 percentile	99 percentile	95 percentile	Median value	Mean value	Missing Value
0	P_2	-3.356653e-01	1.009993	0.007486	0.217058	1.005432	0.974097	0.681664	0.648862	1.220850
1	D_45	1.242293e-05	1.568436	0.002673	0.008129	0.991412	0.754665	0.153975	0.237275	0.069901
2	S_3	-3.409995e-01	3.047587	0.004744	0.063045	1.013916	0.618140	0.165817	0.230616	18.415185
3	B_9	1.537601e-07	14.308021	0.000241	0.001172	1.000241	0.650838	0.028686	0.192319	0.000000
4	D_42	-2.559827e-04	4.183892	0.002611	0.007144	1.049500	0.582117	0.117838	0.185634	81.238927
5	D_50	-7.652144e-01	34.889139	0.002456	0.025821	0.982222	0.445565	0.108740	0.170394	57.672793

Sampling

```
In [160]: #test-train-test split

train_start_date = '2017-05'
train_end_date = '2018-01'

# Define the start and end dates for the test 1 set
test_start_date = '2017-03'
test_end_date = '2017-04'

# Define the start and end dates for the test 2 setYear-Month
test1_start_date = '2018-02'
test1_end_date = '2018-03'

# Split the data into training and test sets
train_final_data = final_data[(final_data['Year-Month'] >= train_start_date) & (final_data['Year-Month'] <= train_end_date)]
test_final_data = final_data[(final_data['Year-Month'] >= test_start_date) & (final_data['Year-Month'] <= test_end_date)]
test2_final_data = final_data[(final_data['Year-Month'] >= test1_start_date) & (final_data['Year-Month'] <= test1_end_date) ]
```

	Time Period	# Obs	Default
Train	2017 May - 2018 Jan	55597	25.94%
Test 1	2017 March - 2017 April	11298	23.72%
Test 2	2018 Feb - 2018 March	16080	28.23%

Data Processing – One Hot Encoding

In [394]:
```
#One hot encoding

one_hot = pd.get_dummies(final_data[['D_63', 'D_64']])

# Combine the one hot encoded data with the original dataframe
final_data = pd.concat([final_data, one_hot], axis=1)

# Drop the original categorical columns
final_data.drop(['D_63', 'D_64', 'customer_ID'], axis=1, inplace=True)
```

In [395]:
```

from sklearn.preprocessing import OrdinalEncoder
for col in['B_30','B_38','D_114','D_116','D_117','D_120','D_126','D_66','D_68']:
    col_dummies=pd.get_dummies(final_data[col],prefix=col)
    final_data=pd.concat([final_data,col_dummies],axis=1)
    final_data.drop(col,axis=1,inplace=True)
```

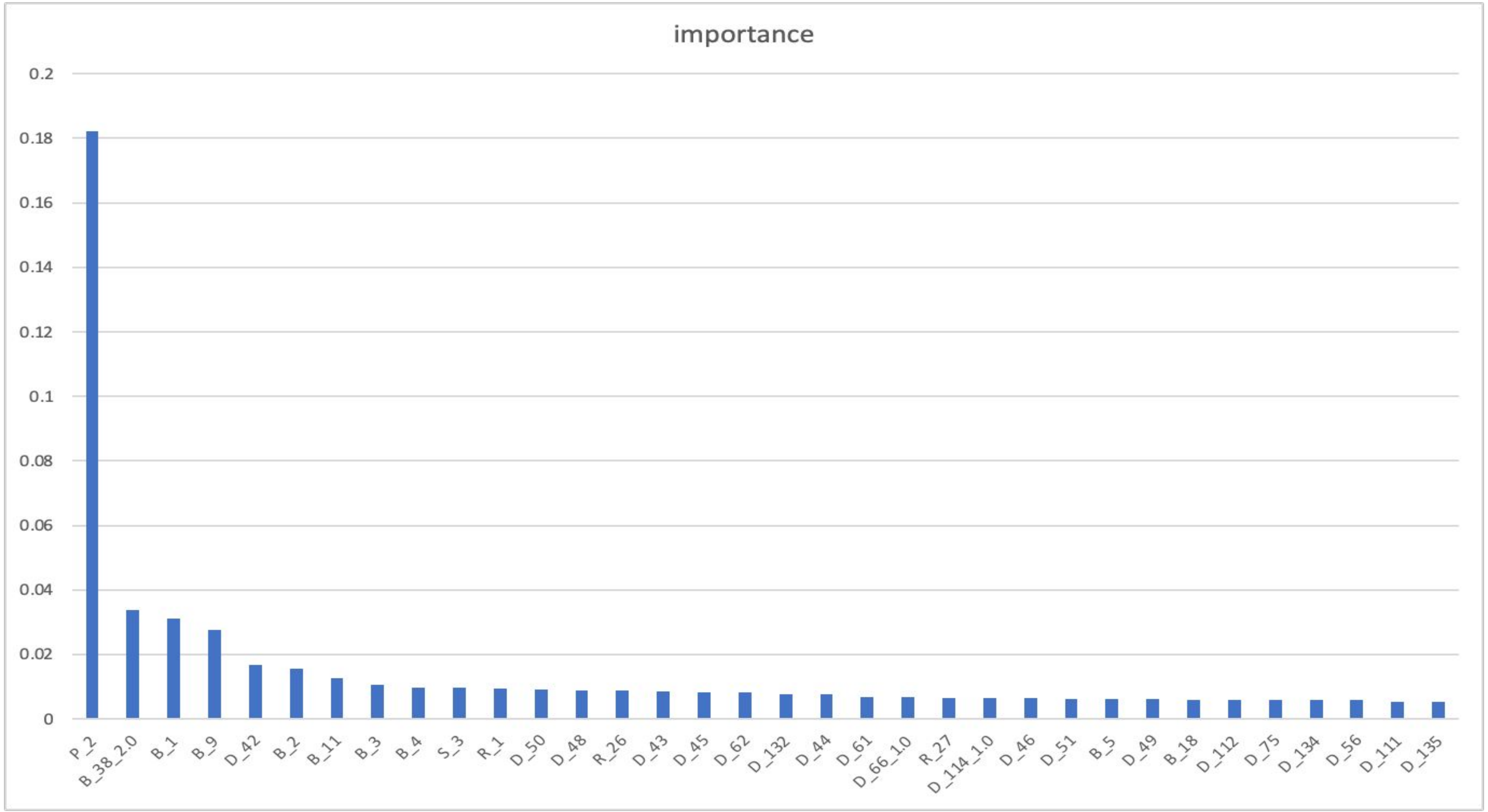
	target	S_2	P_2	D_39	B_1	B_2	R_1	S_3	D_41	B_3	...	D_126_1.0	D_66_0.0	D_66_1.0	D_68_0.0	D_68_1.0
0	0	2018-01-11	0.909811	0.005715	0.002829	1.004798	0.008175	0.098882	0.001853	0.003238	...	1	0	0	0	0
1	0	2017-11-24	0.873512	0.003067	0.008903	1.004783	0.001023	0.119332	0.004865	0.001811	...	1	0	0	0	0
2	0	2017-07-12	0.891656	0.000802	0.009997	0.811041	0.003540	NaN	0.009034	0.000626	...	1	0	0	0	0
6	0	2017-06-10	0.328983	0.038574	0.049463	0.115654	0.004654	0.416112	0.003223	0.081001	...	0	0	0	0	0

Feature Selection

In [172]:
```
#Feature_Importance
importance = model.feature_importances_

print(importance)

feature_importance_df = pd.DataFrame(list(zip(X_train.columns, importance)), columns=['feature', 'importance'])

# sort the dataframe in descending order of importance
feature_importance_df = feature_importance_df.sort_values(by='importance', ascending=False)

# print the dataframe
print(feature_importance_df)

feature_importance_df.to_csv("Feature_Importance_1.csv")
```

```
df_selected_features = set(feature_importance[feature_importance['importance'] >
0.005]['feature']) | set(feature_importance_df[feature_importance_df['importance']>
0.005]['feature'])
df_selected_features= list(df_selected_features)
df_selected_features
```

Feature Selection

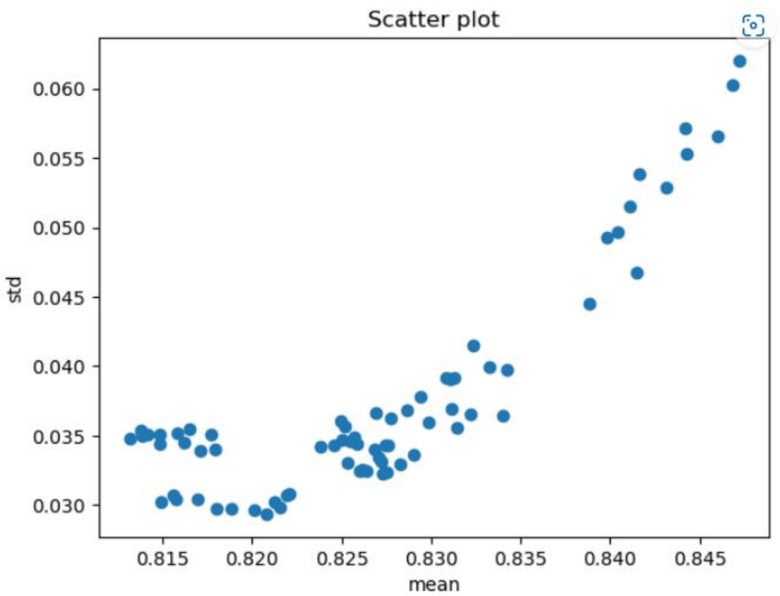


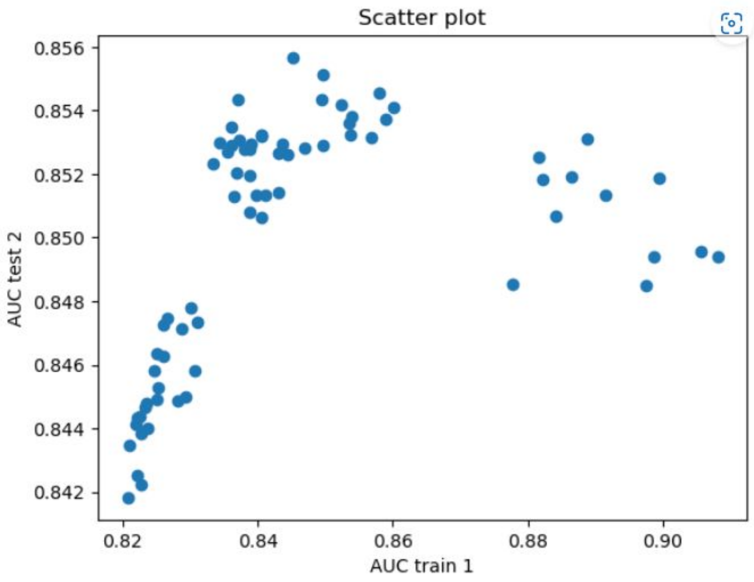
XGBoost - Grid Search

```
In [177]: # Subset X_train to include only selected features
X_train_selected = X_train[df_selected_features]
X_test_selected= X_test[df_selected_features]
X_test1_selected= X_test1[df_selected_features]

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 300],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.5, 0.8],
    'colsample_bytree': [0.5, 1.0],
    'scale_pos_weight': [1, 5, 10]
}

# Create an XGBoost classifier object
xgb_model = xgb.XGBClassifier(random_state=52,objective='binary:logistic', eval_metric='auc')

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, n_jobs=-1,scoring='roc_auc')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train_selected, y_train)
```

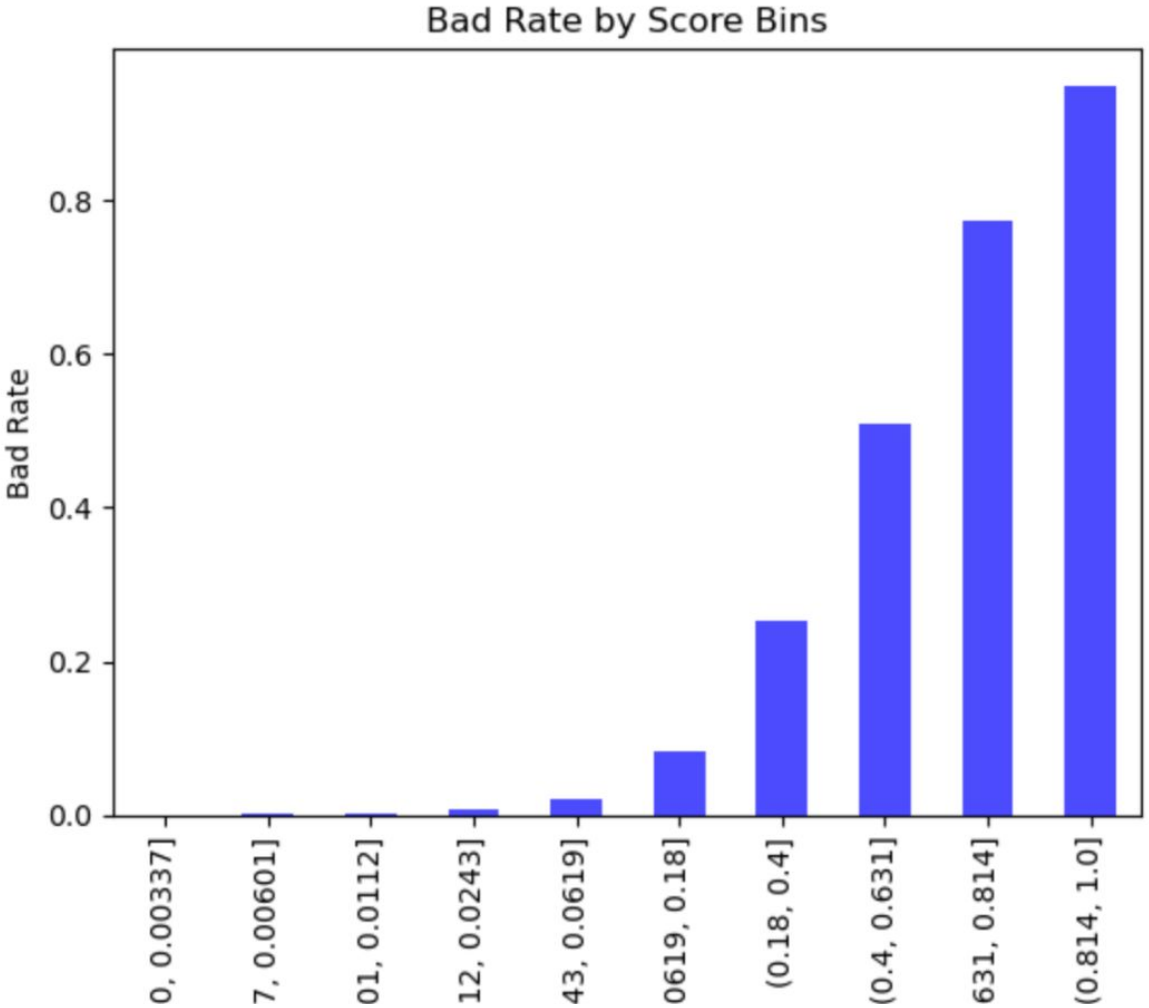
XGBoost - Grid Search

- In the first one, X_Axis shows Average AUC, and Y-Axis shows Standard Deviation of AUC.In the second one, X-Axis is AUC of train sample and Y-Axis is AUC of Test 2 sample.
- Based on the first scatter plot (Average AUC vs. Standard Deviation of AUC), the ideal model would have a high average AUC and a low standard deviation. This would indicate that the model is consistently performing well across different samples and is not overly sensitive to the particular split of the data.
- Based on the second scatter plot (AUC of Train vs. AUC of Test 2), the ideal model would have high AUCs for both the train and test 2 samples, indicating that the model is not overfitting to the training data and is able to generalize well to new data.





In [56]:
```python
import matplotlib.pyplot as plt

ax = stat.plot(kind='bar', y='Bad Rate', color='blue', alpha=0.7, legend=False)
ax.set_xlabel("Score Bins")
ax.set_ylabel("Bad Rate")
ax.set_title("Bad Rate by Score Bins")

# display the chart
plt.show()
```



XGBoost – Final Model

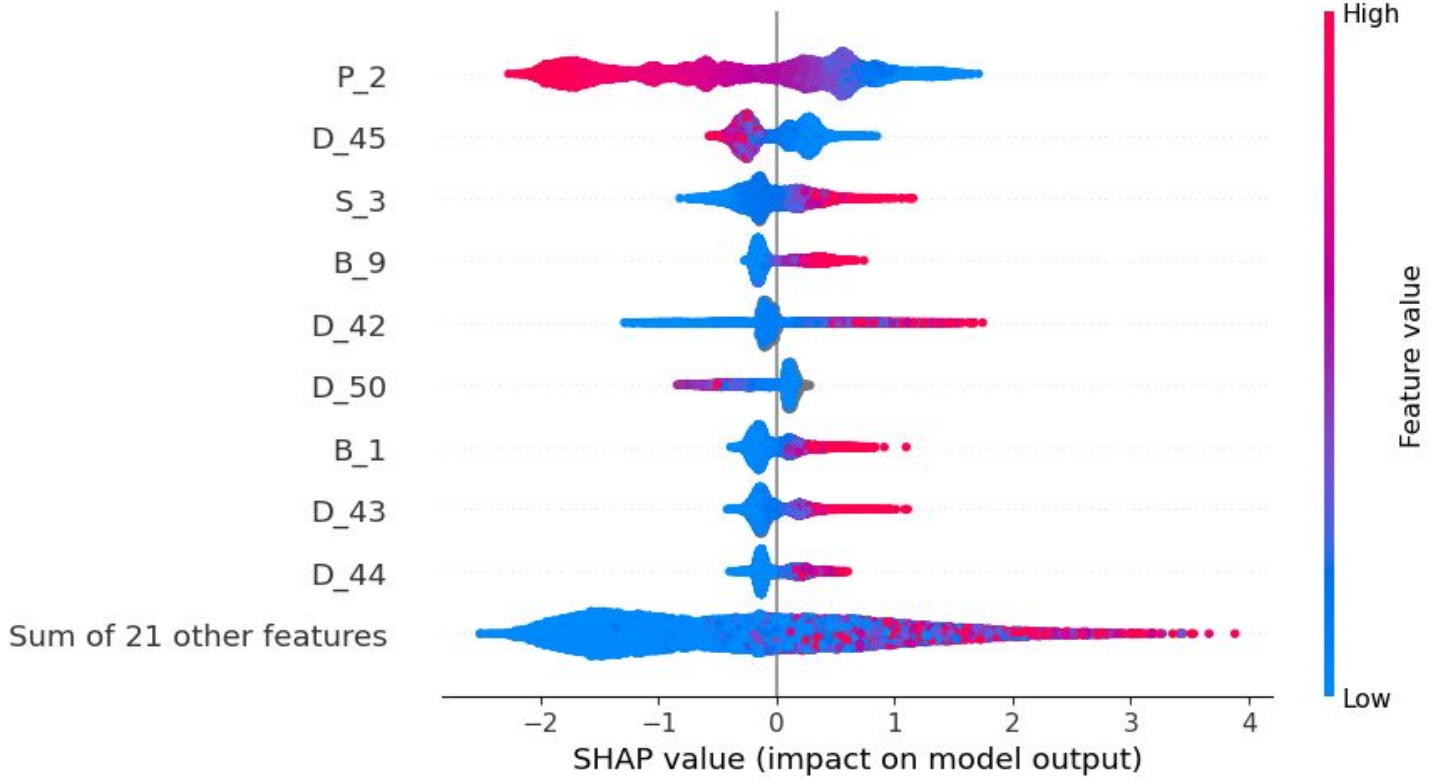
[53]:

```
#running xgb on hyper parameters
params = {
    'learning_rate': 0.1,
    'n_estimators': 100,
    'subsample': 0.8,
    'colsample_bytree': 0.5,
    'scale_pos_weight': 1
}

# Train the model
xgb_best_model = xgb.XGBClassifier(**params)
xgb_best_model.fit(X_train_selected, y_train)
# Evaluate the model
auc_score = roc_auc_score(y_test, xgb_best_model.predict_proba(X_test_selected)[:, 1])
print("AUC score on test data:", auc_score)

auc_score1 = roc_auc_score(y_test1, xgb_best_model.predict_proba(X_test1_selected)[:, 1])
print("AUC score on test 2 data:", auc_score1)
```

```
AUC score on test data: 0.9178918613955542
AUC score on test 2 data: 0.9411621654763803
```

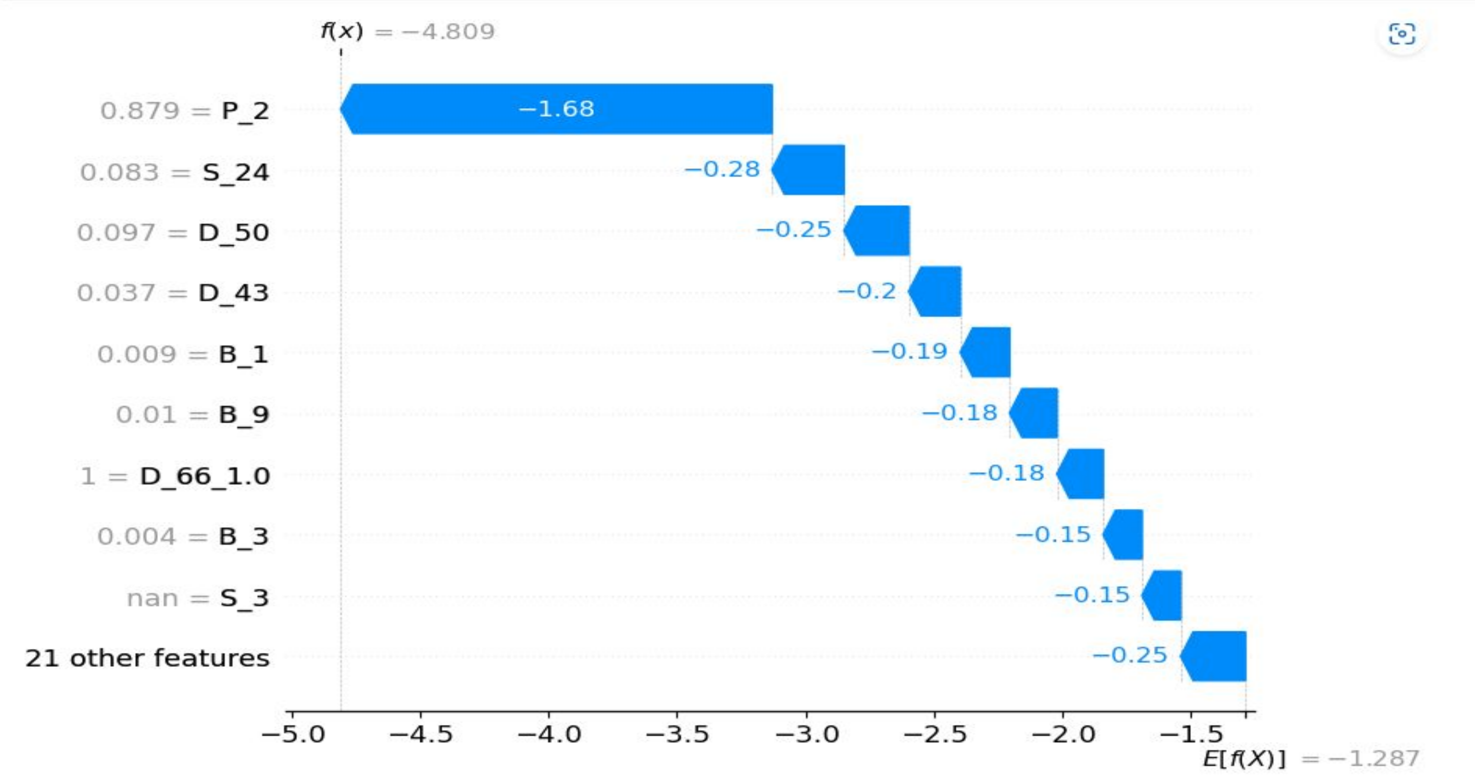
XGBoost – SHAP Analysis

```
import shap
shap.initjs()
explainer = shap.Explainer(xgb_model)
shap_values = explainer(X_test1)
shap.plots.beeswarm(shap_values)
```



XGBoost – SHAP Analysis

```
explainer = shap.Explainer(xgb_model)
shap_values = explainer(X_test1)
shap.plots.waterfall(shap_values[0])
```



Neural Network – Data Processing

Standardization is the process of scaling the input features to have a mean of 0 and standard deviation of 1.
Outlier treatment involves identifying and removing or transforming these extreme values to improve the accuracy and reliability of statistical analysis.
Normalization is a process of transforming a dataset so that its values are on a similar scale, allowing for fair comparisons between different features.

```
X_train_normalized = sc.transform(X_train_selected)
X_test_normalized = sc.transform(X_test_selected)
```

```
#convert to pandas DF
X_train_normalized = pd.DataFrame(X_train_normalized,columns=X_train_selected.columns)
X_test_normalized = pd.DataFrame(X_test_normalized,columns=X_test_selected.columns)
```

```
#Outlier Treatment
X_train_normalized.describe(percentiles=[0.01,0.99]).transpose()
```

	count	mean	std	min	1%	50%	99%	max
R_1	55972.0	-5.890299e-17	1.000009	-0.350764	-0.350285	-0.324694	4.174615	13.115675
D_112	55920.0	1.146754e-16	1.000009	-2.338156	-2.332852	0.429613	0.445687	0.446003
D_42	9875.0	-6.997497e-17	1.000051	-0.793189	-0.781205	-0.271615	3.460219	17.072298

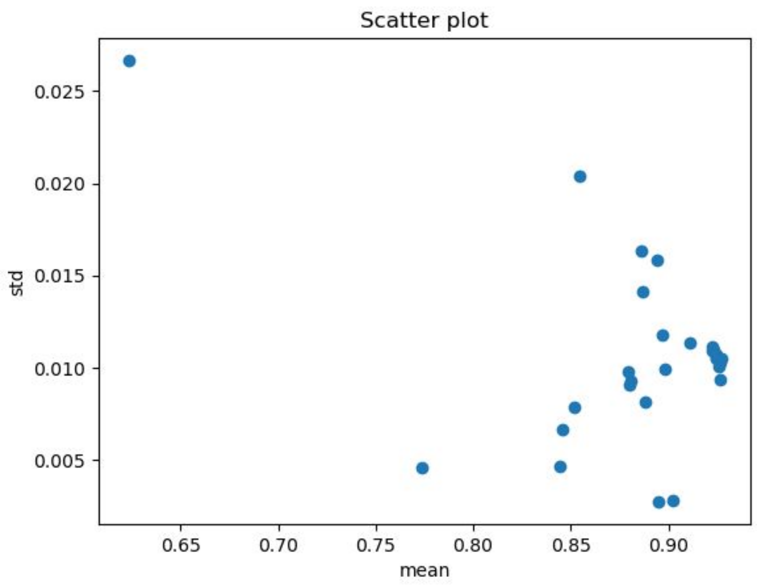
```
#Missing Value Imputation
X_train_normalized.fillna(0,inplace=True)
X_test_normalized.fillna(0,inplace=True)
X_test1_normalized.fillna(0,inplace=True)
```

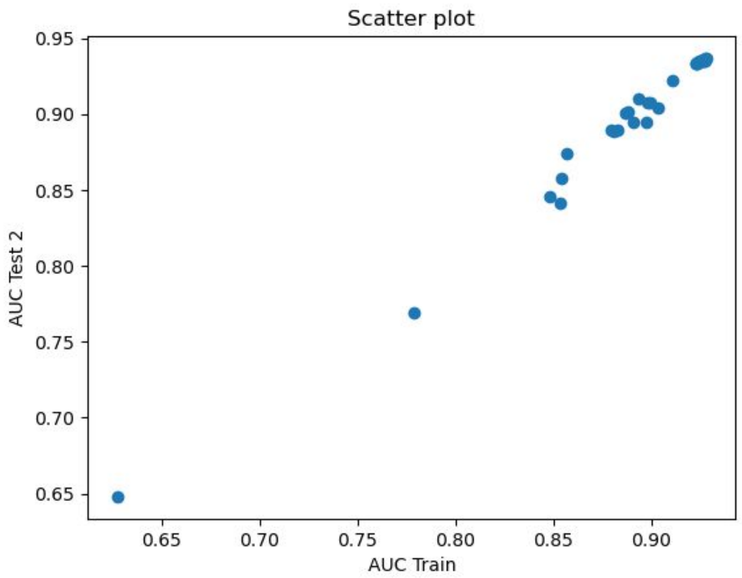
Neural Network - Grid Search

```python
# Define the neural network model
def create_model(hidden_layers=2, nodes_per_layer=4, activation='relu', dropout_rate=0.5):
    model = Sequential()
    for i in range(hidden_layers):
        if i == 0:
            model.add(Dense(nodes_per_layer, activation=activation, input_shape=(X_train_normalized.shape[1],)))
        else:
            model.add(Dense(nodes_per_layer, activation=activation))
        if dropout_rate > 0:
            model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Create a KerasClassifier with default parameters
nn_model = KerasClassifier(build_fn=create_model, epochs=20, batch_size=32, verbose=0)

# Define the hyperparameters for the grid search
params = {
    'hidden_layers': [2, 4],
    'nodes_per_layer': [4, 6],
    'activation': ['relu', 'tanh'],
    'dropout_rate': [0,0.5],
    'batch_size': [100, 10000]
}

# Perform grid search
grid_search = GridSearchCV(nn_model, param_grid=params, cv=5, scoring='roc_auc')
grid_search.fit(X_train_normalized, y_train)
```

Neural Network - Grid Search

- In the first one, X_Axis shows Average AUC, and Y-Axis shows Standard Deviation of AUC.In the second one, X-Axis is AUC of train sample and Y-Axis is AUC of Test 2 sample.
- Based on the first scatter plot (Average AUC vs. Standard Deviation of AUC), the ideal model would have a high average AUC and a low standard deviation. This would indicate that the model is consistently performing well across different samples and is not overly sensitive to the particular split of the data.
- Based on the second scatter plot (AUC of Train vs. AUC of Test 2), the ideal model would have high AUCs for both the train and test 2 samples, indicating that the model is not overfitting to the training data and is able to generalize well to new data.





Neural Network – Final Model

```
# Print the results
print("Best AUC:", grid_search.best_score_)
print("Best parameters:", grid_search.best_params_)
```

```
Best AUC: 0.9254176916886298
Best parameters: {'activation': 'relu', 'batch_size': 100, 'dropout_rate': 0, 'hidden_layers': 2, 'nodes_per_layer'
: 6}
```

```
#Best Neural Network
# Define the neural network model with the best parameters
model_best_nn = Sequential()
model_best_nn.add(Dense(6, activation='relu', input_shape=(X_train_normalized.shape[1],)))
model_best_nn.add(Dropout(0))
model_best_nn.add(Dense(6, activation='relu'))
model_best_nn.add(Dropout(0))
model_best_nn.add(Dense(1, activation='sigmoid'))

# Compile the model
model_best_nn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model on the training data
model_best_nn.fit(X_train_normalized, y_train, batch_size=100, epochs=20, verbose=0)
```

```
<keras.callbacks.History at 0x29cfbc730>
```

```
# Evaluate the model
y_pred_nn = model_best_nn.predict(X_test_normalized)



# Evaluate the model on the test data
loss, accuracy = model_best_nn.evaluate(X_test_normalized, y_test)
print(f"Test loss: {loss:.3f}")
print(f"Test accuracy: {accuracy:.3f}")
```

```
354/354 [==============================] - 0s 241us/step
354/354 [==============================] - 0s 281us/step - loss: 0.3094 - accuracy: 0.8594
```

Final Model : XGBoost

XGBoost:

Hyperparameters:

learning_rate: 0.1
n_estimators: 100
subsample: 0.8
colsample_bytree: 0.5
scale_pos_weight: 1

AUC obtained:

Training Sample: 0.929
Test Sample 1: 0.917
Test Sample 2: 0.941

Strategy - Code

```
#def revenue(TH,x,y,TR):
def revenue(TH,x,y):
    df = pd.DataFrame(y)
    df['Prob of 1'] = xgb_best_model.predict_proba(x)[:, 1]
    df['Accepted_customers'] = (df['Prob of 1'] < TH).astype(int)
    df = df[df['Accepted_customers'] == 1]
    balance_feature = x['B_9']
    spend_feature = x['S_3']
    df['B9_Balance'] = balance_feature
    df['S3_Spend'] = spend_feature
    df = df.dropna()
    length = len(df['target'])
    RR = nm.mean(df['target'])
    df = df.drop(df[df['target'] == 1].index)
    df = df.reset_index(drop=True)

    return(length,RR,(sum(df['B9_Balance'])*0.02) + (sum(df['S3_Spend'])*0.001))


a,b,c = revenue(0.56,X_test1_selected,y_test1)
print("#Accepted: ",a)
print("Default Rate: ",b)
print("Revenue: ",c)
```

Strategy - Analysis of Revenue

	Train			Test 1			Test 2		
Threshold	#Total Accepted	Default Rate	Revenue	#Total Accepted	Default Rate	Revenue	#Total Accepted	Default Rate	Revenue
0.1	22801	0.009	31.00	4918	0.028	6.55	5803	0.014	7.733
0.2	25925	0.022	41.79	5556	0.048	8.20	6677	0.025	10.48
0.3	28208	0.037	51.49	6003	0.065	9.68	7289	0.042	12.55
0.4	30142	0.054	59.83	6415	0.087	11.33	7878	0.062	14.77
0.5	32228	0.078	68.69	6845	0.113	12.86	8488	0.083	17.51
0.6	34421	0.109	77.12	7279	0.141	14.37	9130	0.112	19.84
0.7	37010	0.148	84.94	7739	0.174	15.50	9889	0.150	22.23
0.8	40071	0.198	91.68	8282	0.211	17.03	10870	0.202	24.65
0.9	43587	0.256	94.988	8817	0.247	18.15	12033	0.266	26.27
1	45736	0.290	95.31	9030	0.263	18.30	13067	0.32	26.75