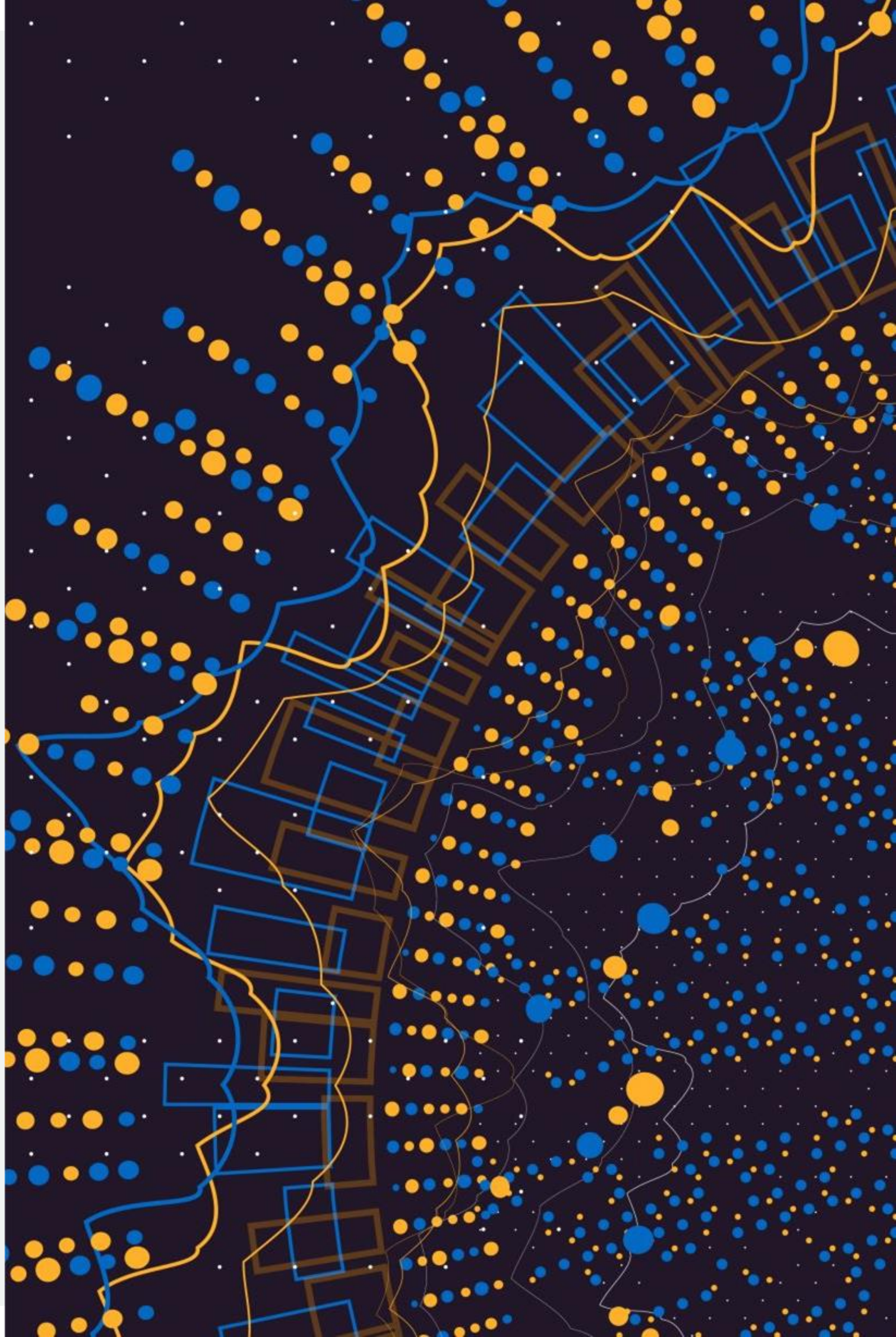
FINANCIAL DISTRESS PREDICTION

a

machine learning portfolio

Adnan



Today's Agenda

- Introduction
- Objective
- The Dataset
- EDA / Feature Engineering
- ML Models
- Model Exploration
- Next Steps

OBJECTIVE

- What constitutes Financial Distress?
  - Credit / Delinquency Risk

- Why?
  - Risk Management
  - Credit Decisions
  - Portfolio Management
  - Compliance
  - Customer Service
- The Metric (AUC , F1)

"Do you have any other collateral...
besides this e-mail from a Nigerian prince?"

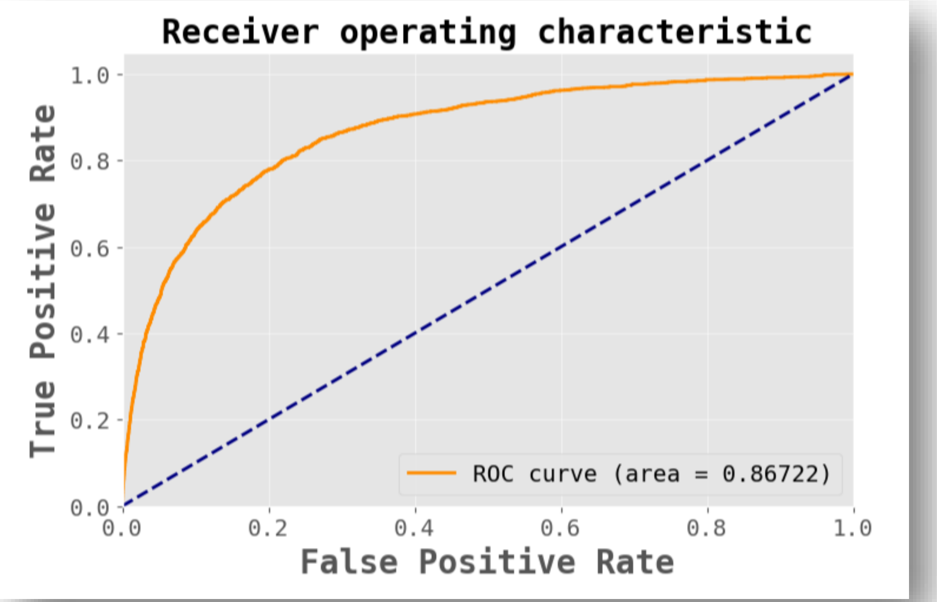
THE METRIC – AUC-ROC

(AREA UNDER THE RECEIVER OPERATING CHARACTERISTIC CURVE)

- The Model's Ability to Distinguish between TPR and FPR.
  - TPR: Actual Positives correctly identified by model
  - FPR: Actual Negatives correctly identified by model
- The ROC curve is created by varying the decision threshold of the model and plotting TPR against FPR at each threshold.
- The higher the AUC-ROC score, the better the model's predictive performance.
- Default Threshold in Sckitlearn is **0.5**

```
ypred_train = gs.best_estimator_.predict_proba(X_train)[:, 1]
ypred = gs.best_estimator_.predict_proba(X_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, ypred, pos_label=1)
roc_auc = auc(fpr, tpr)
```

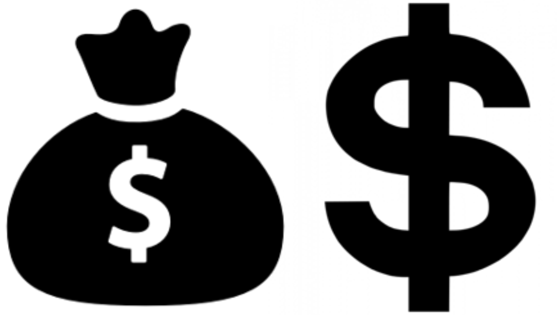
- No single industry standard threshold for classification problems, including financial distress prediction.
- Threshold is often chosen based on a balance of the true positive rate (TPR) and false positive rate (FPR) of the model, which can be obtained from the ROC curve



THE DATASET

give me some credit

kaggle



Feature	Description	Data Type
age	Age of borrower in years	Integer
MonthlyIncome	Monthly Income	Real
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.)	Integer
DebtRatio	Monthly debt payments, alimony, living costs divided by monthy gross income	Percentage
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits	Percentage
NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards)	Integer
NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days past due but no worse in the last 2 years.	Integer
NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days past due but no worse in the last 2 years.	Integer
NumberOfTimes90DaysLate	Number of times borrower has been 90 days or more past due.	Integer
SeriousDlqin2yrs	Person experienced 90 days past due delinquency or worse	Categorical (Y/N)

PROJECT PLAN

Data Import

- Load Data
- Reindex Data
- Describe Data

Exploratory Data Analysis

- Study Variables
- Outlier/ Null Removal
- Data Correlations

Feature Engineering

- Data Relations
- Custom Features
- Remove Unwanted Features

Basic Model

- Trial All Models
- Find Best Scaler / Sampler for Data

Modeling

- GridSearch
- Plot The AUC
- Get Permutation Importance

Final Models & Ensemble

- Plot History of Epochs
- Voting Classify best models

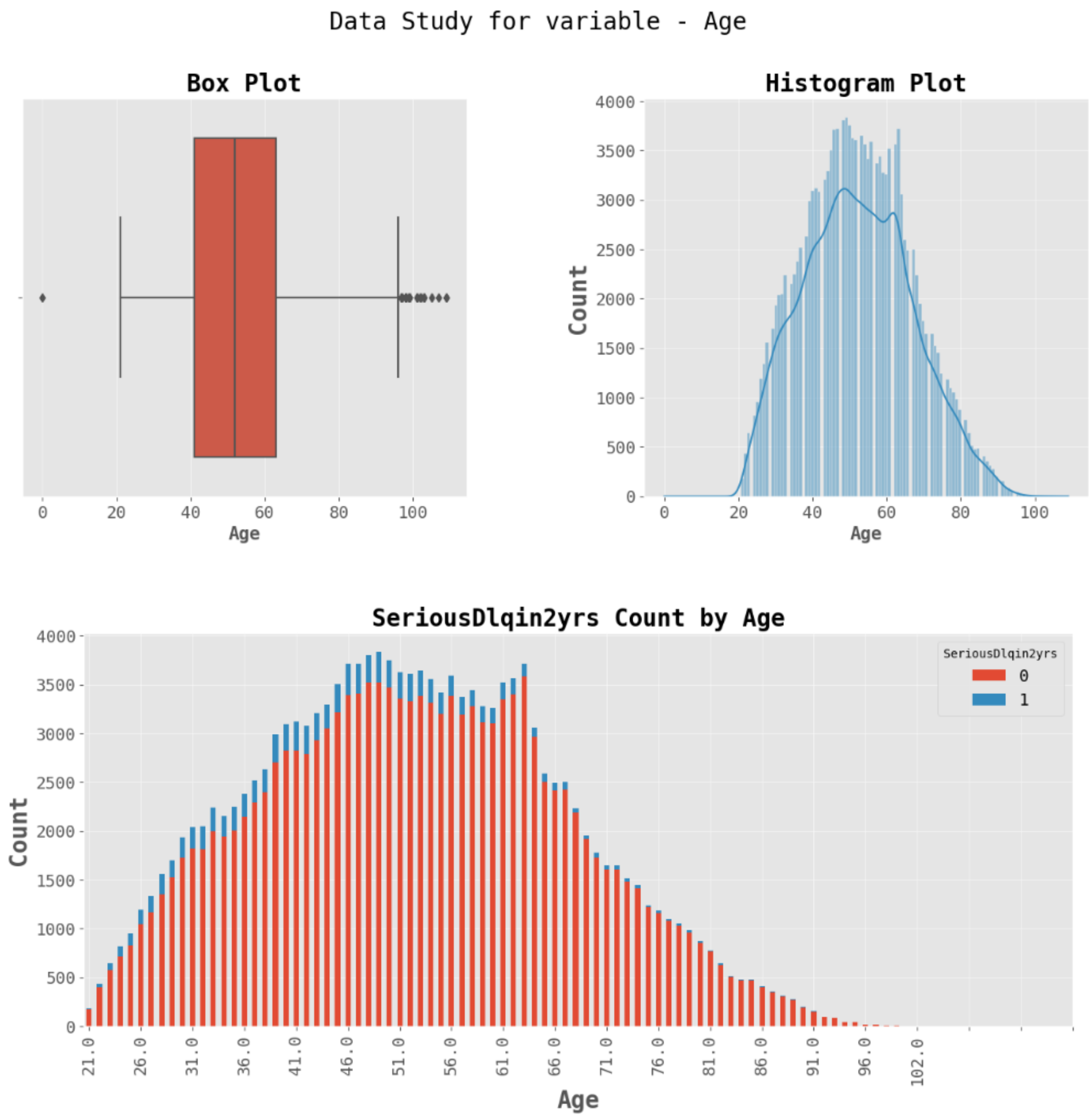
Model Study & Interpretation

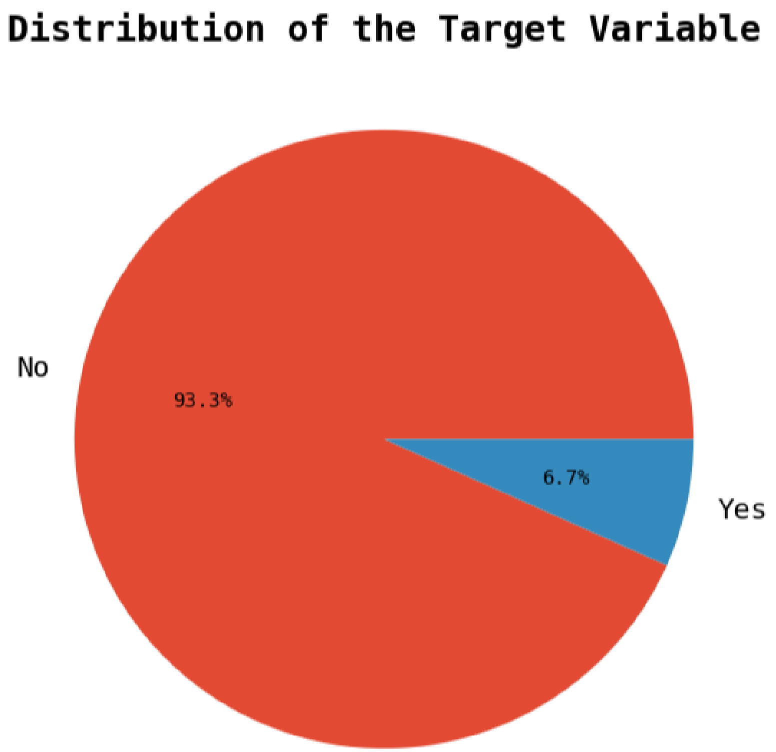
- Plot AUC of all models
- LIME & SHAP

Pipeline & Conclusion

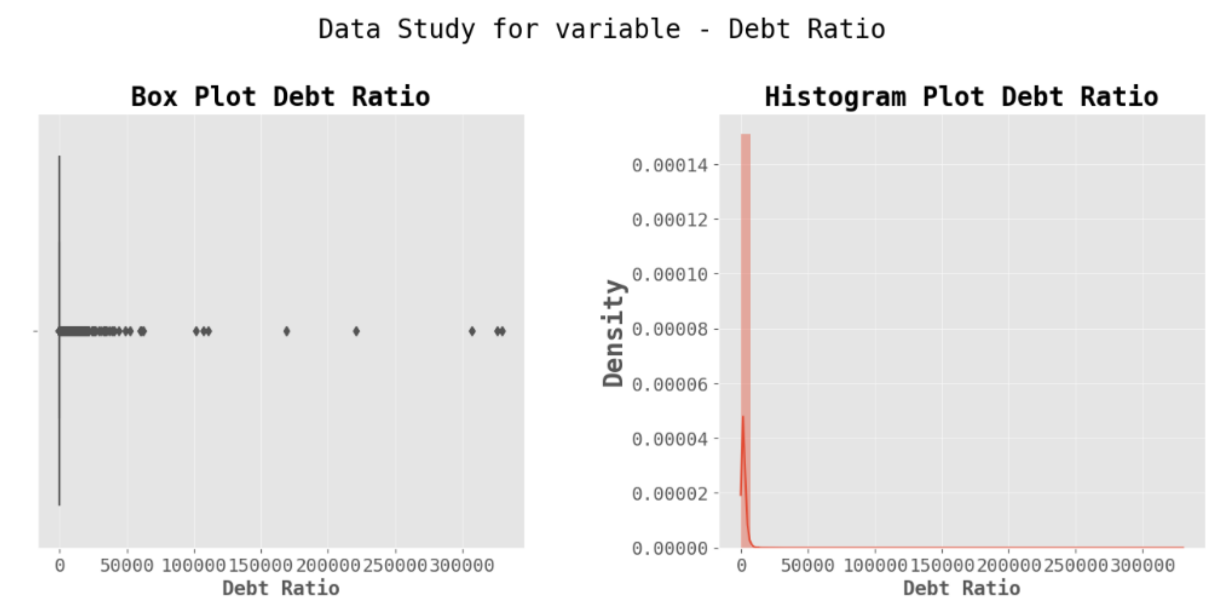
- Function Transform all Functions
- Build Pipeline
- Conclusions

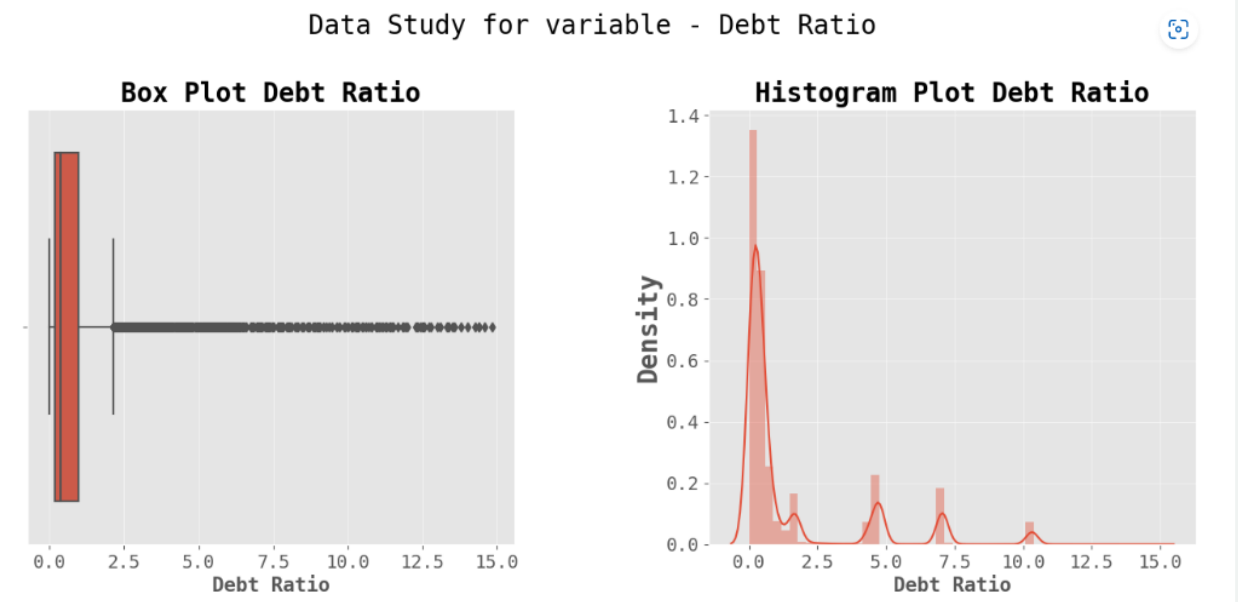
EXPLORATORY DATA ANALYSIS

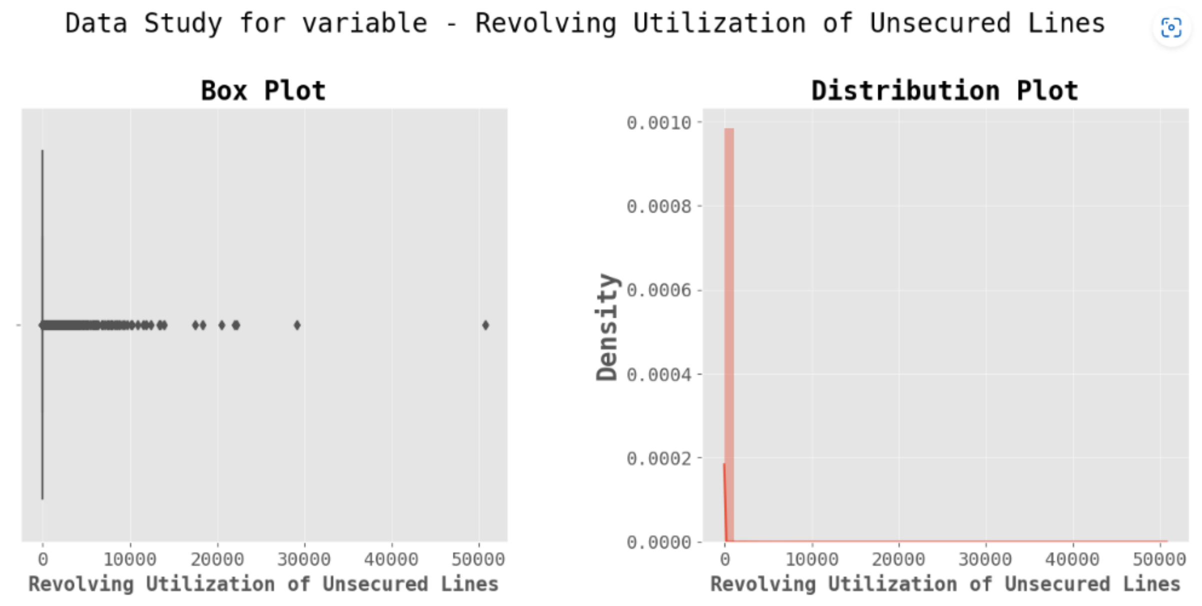


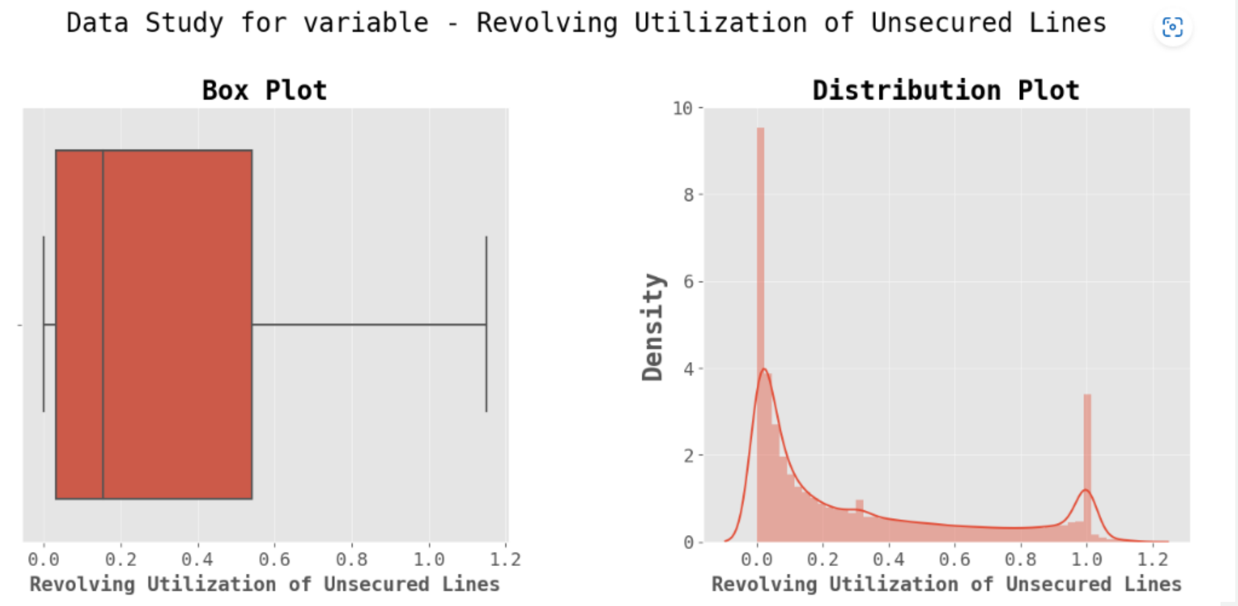


EXPLORATORY DATA ANALYSIS









FEATURE ENGINEERING

Debt-per-person: This feature
feature could be useful for prec

```
[88]: df['debt_per_person'] = df['I
      df['debt_per_person'] = df['
      mean_debt_per_person = df['de
      df['debt_per_person'].fillna
```

Delinquency_ratio: This featu
limit across all open credit lines
credit-limit ratios are generally

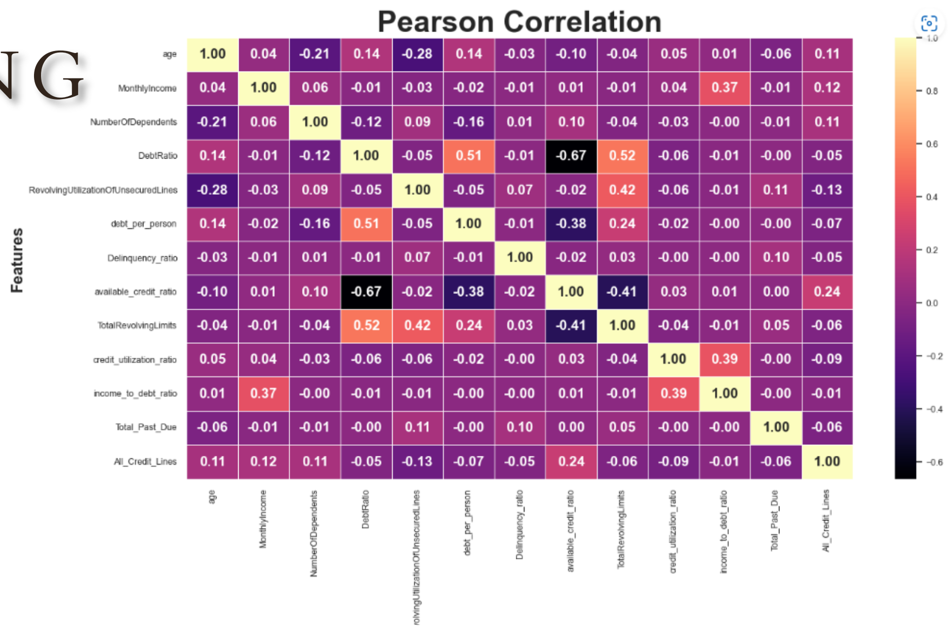
```
[89]: df['TotalCreditLines'] = df[
      df['Delinquency_ratio'] = df|


      df['Delinquency_ratio'] = df|
      mean_Delinquency_ratio = df[
      df['Delinquency_ratio'].filli
```

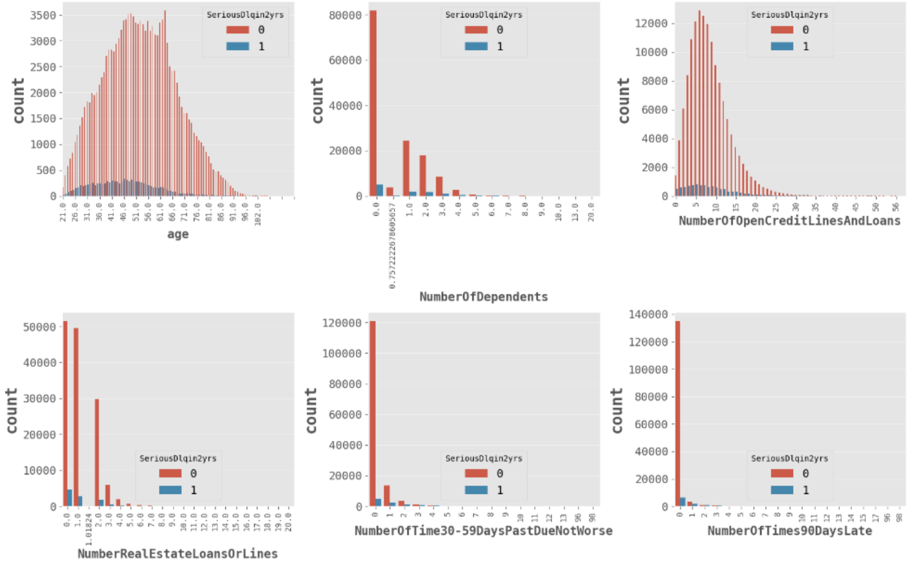
Available credit ratio: This fea
credit limit across all open cred
available credit ratios are gene

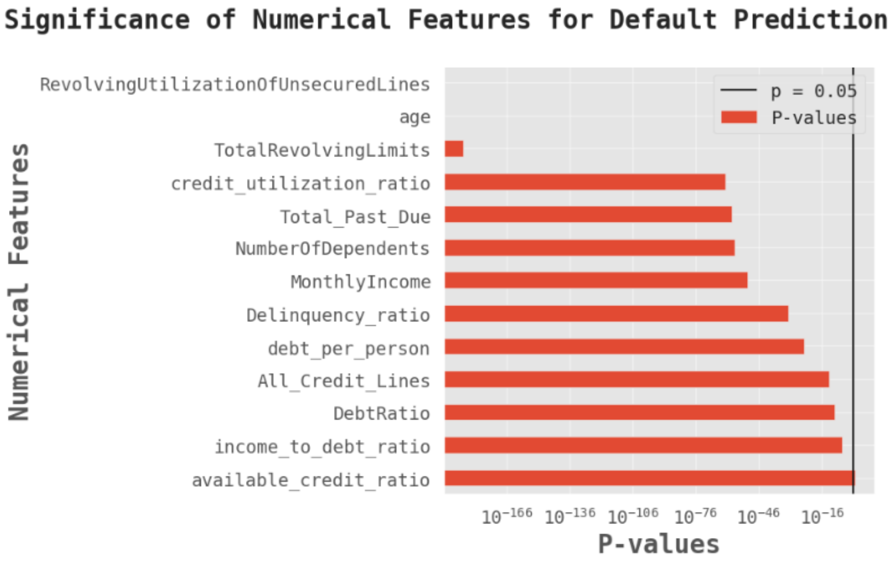
```
[90]: df['available_credit_ratio']
      df['available_credit_ratio']
      mean_available_credit_ratio =
      df['available_credit_ratio'].
```

TotalRevolvingLimits: This fe
typically calculated by summin

```
[91]: df['TotalRevolvingLimits'] =
```







We can also see all the p values are on or below 0.05 showing originality in the variables.

feature	VIF
age	4.644384
MonthlyIncome	1.296428
NumberOfDependents	1.478024
DebtRatio	1.410657
RevolvingUtilizationOfUnsecuredLines	1.795841
NumberOfOpenCreditLinesAndLoans	4.593384
NumberRealEstateLoansOrLines	2.319815
NumberOfTime30-59DaysPastDueNotWorse	42.593421
NumberOfTime60-89DaysPastDueNotWorse	95.116295
NumberOfTimes90DaysLate	74.187992
SeriousDlqin2yrs	1.188257

Delinquency_ratio	1.023010
available_credit_ratio	3.153409
TotalRevolvingLimits	2.165645
credit_utilization_ratio	1.233292
income_to_debt_ratio	1.382536
Total_Past_Due	1.032038
All_Credit_Lines	1.126240

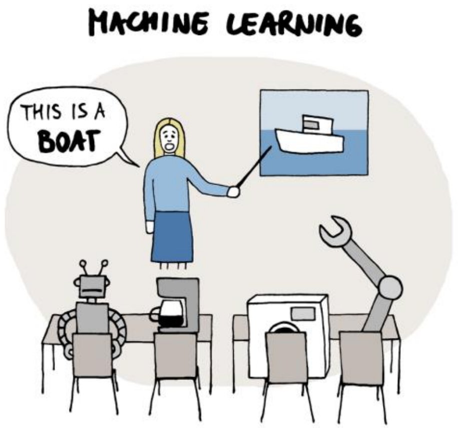
```python
def clean_credit_data(df):
    """
    This function takes in the dataframe and does all basic and EDA cleani
    """
    # Make a copy of the dataframe to avoid modifying the original
    df_clean = df.copy()

    # Remove unwanted column
    df_clean = df_clean.drop('Unnamed: 0', axis=1)

    # Reordering df
    new_order = ['age', 'MonthlyIncome', 'NumberOfDependents', 'DebtRatio'
                 'RevolvingUtilizationOfUnsecuredLines', 'NumberOfOpenCred
                 'NumberRealEstateLoansOrLines', 'NumberOfTime30-59DaysPas
                 'NumberOfTime60-89DaysPastDueNotWorse', 'NumberOfTimes90D
    df_clean = df_clean.reindex(columns=new_order)
```

```python
def feature_engineering(df):
    """
    This function takes in the dataframe and does all feature
    """

    df_feature = df.copy() # create a copy of the original dat

    # debt_per_person feature
    df_feature['debt_per_person'] = df_feature['DebtRatio'] /
    df_feature['debt_per_person'] = df_feature['debt_per_perso
    mean_debt_per_person = df_feature['debt_per_person'].mean(
    df_feature['debt_per_person'].fillna(mean_debt_per_person,
```

FINAL ML DATASET

```
Data columns (total 14 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   age                                   150000 non-null  float64
 1   MonthlyIncome                         150000 non-null  float64
 2   NumberOfDependents                    150000 non-null  float64
 3   DebtRatio                             150000 non-null  float64
 4   RevolvingUtilizationOfUnsecuredLines  150000 non-null  float64
 5   SeriousDlqin2yrs                      150000 non-null  category
 6   debt_per_person                       150000 non-null  float64
 7   Delinquency_ratio                     150000 non-null  float64
 8   available_credit_ratio                150000 non-null  float64
 9   TotalRevolvingLimits                  150000 non-null  float64
 10  credit_utilization_ratio              150000 non-null  float64
 11  income_to_debt_ratio                  150000 non-null  float64
 12  Total_Past_Due                        150000 non-null  float64
 13  All_Credit_Lines                      150000 non-null  float64
dtypes: category(1), float64(13)
memory usage: 15.0 MB
```

"The machine learning algorithm wants to know if we'd like a dozen wireless mice to feed the Python book we just bought."

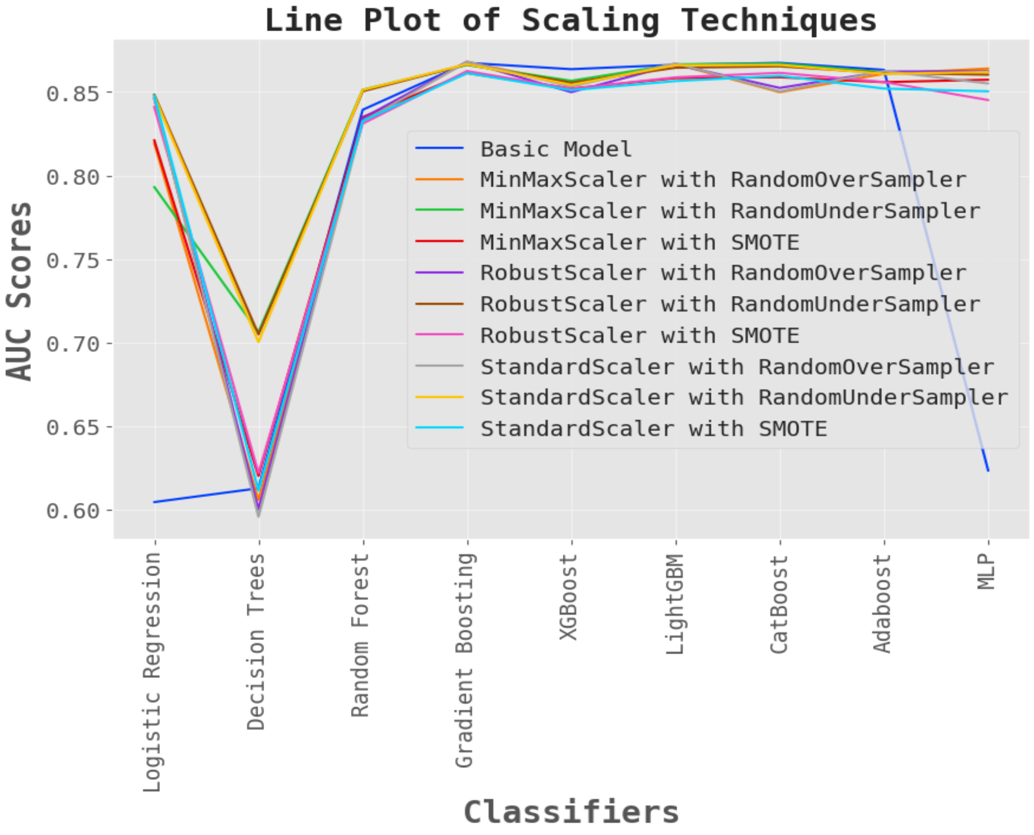


```python
def get_auc_scores(X_train, X_test, y_train, y_test, trial_name):
    classifiers = {
        'Logistic Regression': LogisticRegression(random_state=42, n_jobs=-1),
        'Decision Trees': DecisionTreeClassifier(random_state=42),
        'Random Forest': RandomForestClassifier(random_state=42, n_jobs=-1),
        'Gradient Boosting': GradientBoostingClassifier(),
#         'SVM': SVC(probability=True) #This was removed because it is very expensive computationally
        'XGBoost': XGBClassifier(random_state=42, n_jobs=-1),
        'LightGBM': LGBMClassifier(random_state=42, n_jobs=-1),
        'CatBoost': CatBoostClassifier(random_state=42, verbose=False, thread_count=-1),
        'Adaboost': AdaBoostClassifier(random_state=42),
        'MLP': MLPClassifier(random_state=42)
    }
```

	Model	Basic Model
0	Logistic Regression	0.604587
1	Decision Trees	0.612859
2	Random Forest	0.839380
3	Gradient Boosting	0.867459
4	XGBoost	0.863670
5	LightGBM	0.866314
6	CatBoost	0.867437
7	Adaboost	0.863255
8	MLP	0.623372

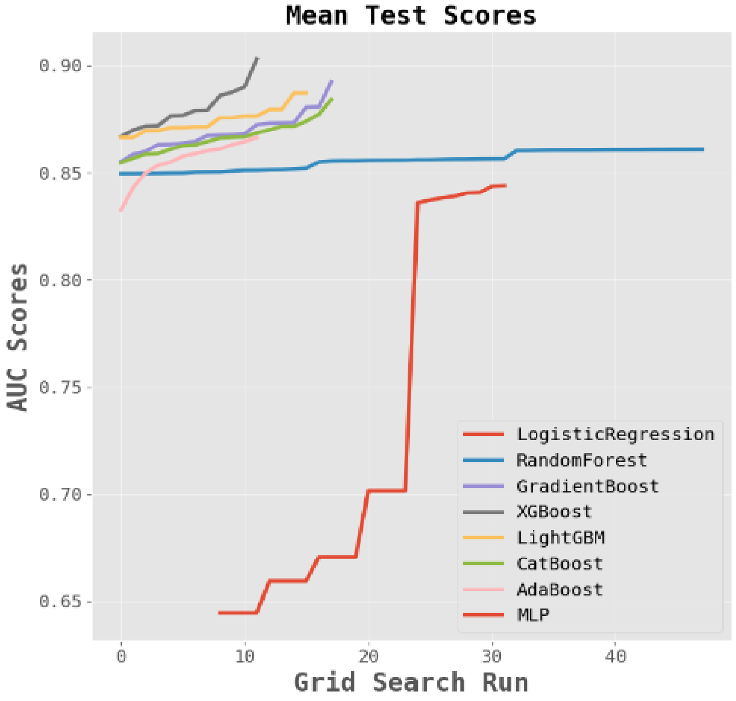
THE BASIC MODELS

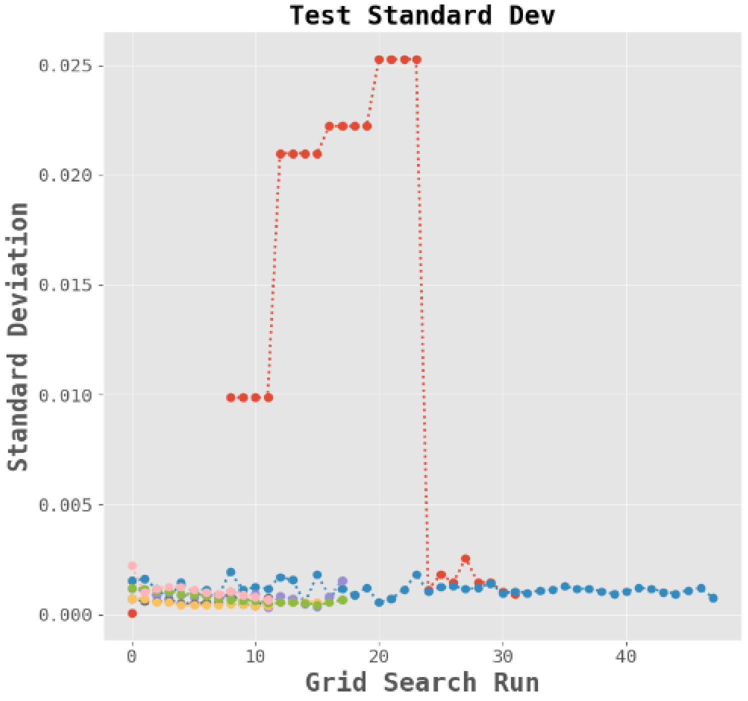
```python
scalers = [MinMaxScaler(), RobustScaler(), StandardScaler()]
samplers = [RandomOverSampler(), RandomUnderSampler(), SMOTE()]

results_list = []

# Loop through each scaler and sampler combination
for scaler in scalers:
    scaler_name = type(scaler).__name__
    for sampler in samplers:
        sampler_name = type(sampler).__name__

        # Scale and resample the data
        scaler.fit(X_train)
        X_train_scaled = scaler.transform(X_train)
        X_test_scaled = scaler.transform(X_test)
        X_train_resampled, y_train_resampled = sampler.fit_resample(X_train_
```



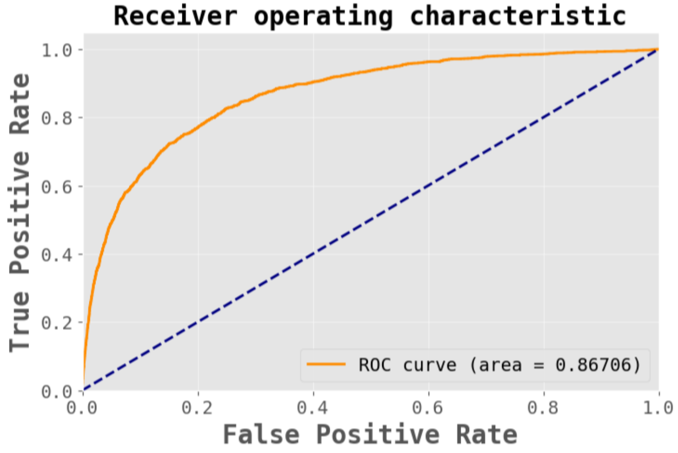
TUNED MODELS

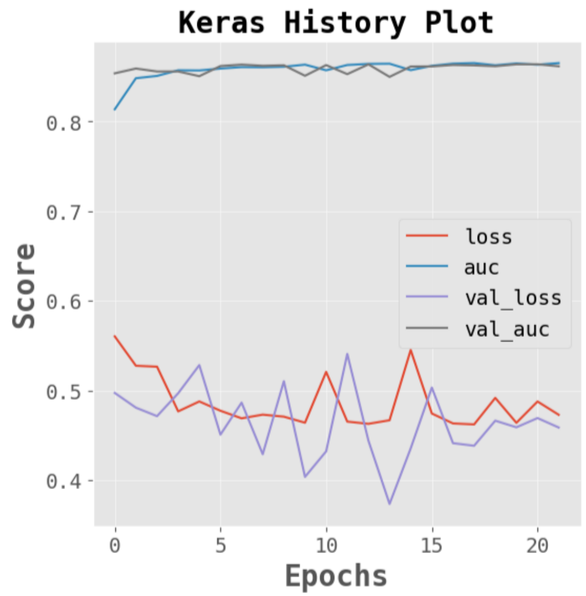
```python
def compute_model(model, params):
    """
    This function does a grid search with a 3 fold Stratified K fo
    the best model, the fpr, tpr and roc_auc values.
    """
    skf = StratifiedKFold(n_splits=3)

    gs = GridSearchCV(model, params, cv=skf, n_jobs=-1, verbose=1,
    gs.fit(X_train, y_train)

    model_stats = pd.DataFrame(gs.cv_results_)
    model_stats = model_stats.sort_values(by='rank_test_score', as

    ypred_train = gs.best_estimator_.predict_proba(X_train)[:, 1]
    ypred = gs.best_estimator_.predict_proba(X_test)[:, 1]

    fpr, tpr, thresholds = roc_curve(y_test, ypred, pos_label=1)
    roc_auc = auc(fpr, tpr)

    result = {'fpr': fpr, 'tpr': tpr, 'roc_auc': roc_auc}

    y_pred = gs.best_estimator_.predict(X_test)
    print(classification_report(y_test, y_pred))

    return result, gs.best_estimator_,model_stats
```






```python
%%time
# LightGBM
lgb_model = LGBMClassifier(random_state=42, n_jobs=-1)
lgb_param_grid = {
    'learning_rate': [0.1, 0.05],
    'n_estimators': [150, 250],
    'max_depth': [3, 4],
    'num_leaves': [31, 63]
}

model_name = 'LightGBM'
result, best_estimator, lgb_model_stats = compute_model(lgb_model, lgb_param_grid)
model_results[model_name] = [result, best_estimator]
plot_roc_curve(result, best_estimator)

# Compute permutation importance using Eli5
perm = PermutationImportance(best_estimator, random_state=42)
perm.fit(X_train_d, y_train)
eli5.show_weights(perm, feature_names=X_train_d.columns.tolist())
```





PERMUTATION IMPORTANCE

Weight	Feature
0.0650 ± 0.0007	RevolvingUtilizationOfUnsecuredLines
0.0446 ± 0.0006	Delinquency_ratio
0.0203 ± 0.0009	age
0.0069 ± 0.0002	TotalRevolvingLimits
0.0033 ± 0.0002	Total_Past_Due
0.0015 ± 0.0002	MonthlyIncome
0.0002 ± 0.0002	NumberOfDependents
0.0000 ± 0.0001	credit_utilization_ratio
-0.0000 ± 0.0000	income_to_debt_ratio
-0.0002 ± 0.0002	debt_per_person
-0.0003 ± 0.0004	DebtRatio
-0.0009 ± 0.0002	available_credit_ratio
-0.0098 ± 0.0008	All_Credit_Lines

Logistic Regression

Weight	Feature
0.0317 ± 0.0016	Delinquency_ratio
0.0287 ± 0.0014	Total_Past_Due
0.0157 ± 0.0008	RevolvingUtilizationOfUnsecuredLines
0.0102 ± 0.0001	TotalRevolvingLimits
0.0054 ± 0.0006	age
0.0034 ± 0.0002	All_Credit_Lines
0.0027 ± 0.0003	MonthlyIncome
0.0026 ± 0.0001	credit_utilization_ratio
0.0016 ± 0.0003	DebtRatio
0.0012 ± 0.0003	available_credit_ratio
0.0011 ± 0.0003	income_to_debt_ratio
0.0004 ± 0.0002	NumberOfDependents
0.0002 ± 0.0001	debt_per_person

Random Forest

Weight	Feature
0.0751 ± 0.0009	RevolvingUtilizationOfUnsecuredLines
0.0681 ± 0.0007	Delinquency_ratio
0.0279 ± 0.0007	Total_Past_Due
0.0191 ± 0.0003	TotalRevolvingLimits
0.0185 ± 0.0006	age
0.0177 ± 0.0009	income_to_debt_ratio
0.0172 ± 0.0003	All_Credit_Lines
0.0150 ± 0.0007	available_credit_ratio
0.0143 ± 0.0009	DebtRatio
0.0128 ± 0.0004	MonthlyIncome
0.0091 ± 0.0003	credit_utilization_ratio
0.0062 ± 0.0005	debt_per_person
0.0013 ± 0.0001	NumberOfDependents

Gradient Boosting

Weight	Feature
0.0783 ± 0.0016	Total_Past_Due
0.0630 ± 0.0009	RevolvingUtilizationOfUnsecuredLines
0.0394 ± 0.0011	Delinquency_ratio
0.0260 ± 0.0004	income_to_debt_ratio
0.0181 ± 0.0006	credit_utilization_ratio
0.0100 ± 0.0006	All_Credit_Lines
0.0090 ± 0.0008	available_credit_ratio
0.0090 ± 0.0008	TotalRevolvingLimits
0.0087 ± 0.0006	age
0.0072 ± 0.0003	DebtRatio
0.0048 ± 0.0007	MonthlyIncome
0.0026 ± 0.0004	debt_per_person
0.0025 ± 0.0007	NumberOfDependents

MLP

Weight	Feature
0.0579 ± 0.0011	RevolvingUtilizationOfUnsecuredLines
0.0336 ± 0.0013	Delinquency_ratio
0.0289 ± 0.0009	Total_Past_Due
0.0203 ± 0.0006	age
0.0182 ± 0.0006	All_Credit_Lines
0.0157 ± 0.0006	available_credit_ratio
0.0156 ± 0.0010	income_to_debt_ratio
0.0135 ± 0.0004	MonthlyIncome
0.0113 ± 0.0008	DebtRatio
0.0092 ± 0.0003	credit_utilization_ratio
0.0086 ± 0.0008	TotalRevolvingLimits
0.0059 ± 0.0006	debt_per_person
0.0022 ± 0.0002	NumberOfDependents

XGBoost

Weight	Feature
0.0570 ± 0.0016	RevolvingUtilizationOfUnsecuredLines
0.0464 ± 0.0009	Delinquency_ratio
0.0243 ± 0.0010	Total_Past_Due
0.0096 ± 0.0007	All_Credit_Lines
0.0090 ± 0.0003	age
0.0041 ± 0.0003	MonthlyIncome
0.0025 ± 0.0004	DebtRatio
0.0013 ± 0.0002	income_to_debt_ratio
0.0010 ± 0.0002	credit_utilization_ratio
0.0009 ± 0.0003	TotalRevolvingLimits
0.0008 ± 0.0003	available_credit_ratio
0.0007 ± 0.0001	debt_per_person
0.0003 ± 0.0001	NumberOfDependents

AdaBoost

Weight	Feature
0.0555 ± 0.0014	RevolvingUtilizationOfUnsecuredLines
0.0450 ± 0.0011	Delinquency_ratio
0.0241 ± 0.0007	Total_Past_Due
0.0179 ± 0.0009	age
0.0152 ± 0.0007	All_Credit_Lines
0.0122 ± 0.0006	income_to_debt_ratio
0.0105 ± 0.0006	available_credit_ratio
0.0097 ± 0.0003	MonthlyIncome
0.0095 ± 0.0006	DebtRatio
0.0068 ± 0.0003	TotalRevolvingLimits
0.0061 ± 0.0005	credit_utilization_ratio
0.0061 ± 0.0004	debt_per_person
0.0018 ± 0.0002	NumberOfDependents

LightGBM

Weight	Feature
0.0612 ± 0.0013	Total_Past_Due
0.0551 ± 0.0011	RevolvingUtilizationOfUnsecuredLines
0.0243 ± 0.0011	Delinquency_ratio
0.0210 ± 0.0007	age
0.0196 ± 0.0008	All_Credit_Lines
0.0145 ± 0.0003	available_credit_ratio
0.0134 ± 0.0004	MonthlyIncome
0.0125 ± 0.0006	TotalRevolvingLimits
0.0100 ± 0.0007	income_to_debt_ratio
0.0097 ± 0.0004	DebtRatio
0.0088 ± 0.0006	credit_utilization_ratio
0.0039 ± 0.0002	debt_per_person
0.0022 ± 0.0003	NumberOfDependents

CatBoost

MODEL ENSEMBLES

	model_name	roc_auc	best_model
5	CatBoost	0.867221	<catboost.core.CatBoostClassifier object at 0x...
6	AdaBoost	0.867055	(DecisionTreeClassifier(max_depth=2, random_st...
4	LightGBM	0.866887	LGBMClassifier(max_depth=4, n_estimators=250, ...
3	XGBoost	0.866121	XGBClassifier(base_score=None, booster=None, c...
7	MLP	0.863816	MLPClassifier(hidden_layer_sizes=(30,), random...
2	GradientBoosting	0.863280	([DecisionTreeRegressor(criterion='friedman_ms...
1	RandomForest	0.862225	(DecisionTreeClassifier(criterion='entropy', m...
0	Logistic Regression	0.849474	LogisticRegression(C=1, class_weight={0: 1, 1:...

```
estimators = []
for index, row in results.head(4).iterrows():
    variable_name = row['model_name'] + '_model'
    variable_value = row['best_model']
    globals()[variable_name] = variable_value
    estimators.append((row['model_name'], variable_value))
```

```
%%time
#VotingClassifier
vc = VotingClassifier(estimators=estimators, voting = 'soft')

# Fit Training Data
vc_scores = cross_val_score(vc, X_train, y_train, cv=5, scoring=
```

CatBoost

Model AUC Score: 0.86722

Weight	Feature
0.1062 ± 0.0011	RevolvingUtilizationOfUnsecuredLines
0.0801 ± 0.0007	TotalRevolvingLimits
0.0789 ± 0.0009	income_to_debt_ratio
0.0765 ± 0.0008	available_credit_ratio
0.0751 ± 0.0011	Total_Past_Due
0.0725 ± 0.0003	Delinquency_ratio
0.0662 ± 0.0002	DebtRatio
0.0569 ± 0.0009	credit_utilization_ratio
0.0458 ± 0.0008	All_Credit_Lines
0.0453 ± 0.0012	age
0.0435 ± 0.0005	MonthlyIncome
0.0285 ± 0.0006	debt_per_person
0.0047 ± 0.0005	NumberOfDependents

Weight	Feature
0.0525 ± 0.0010	RevolvingUtilizationOfUnsecuredLines
0.0319 ± 0.0014	Delinquency_ratio
0.0275 ± 0.0009	Total_Past_Due
0.0188 ± 0.0008	age
0.0160 ± 0.0008	All_Credit_Lines
0.0107 ± 0.0004	MonthlyIncome
0.0106 ± 0.0005	available_credit_ratio
0.0097 ± 0.0008	income_to_debt_ratio
0.0072 ± 0.0005	DebtRatio
0.0061 ± 0.0006	TotalRevolvingLimits
0.0047 ± 0.0004	credit_utilization_ratio
0.0040 ± 0.0002	debt_per_person
0.0017 ± 0.0002	NumberOfDependents

KerasClassifier

Model AUC Score: 0.86182

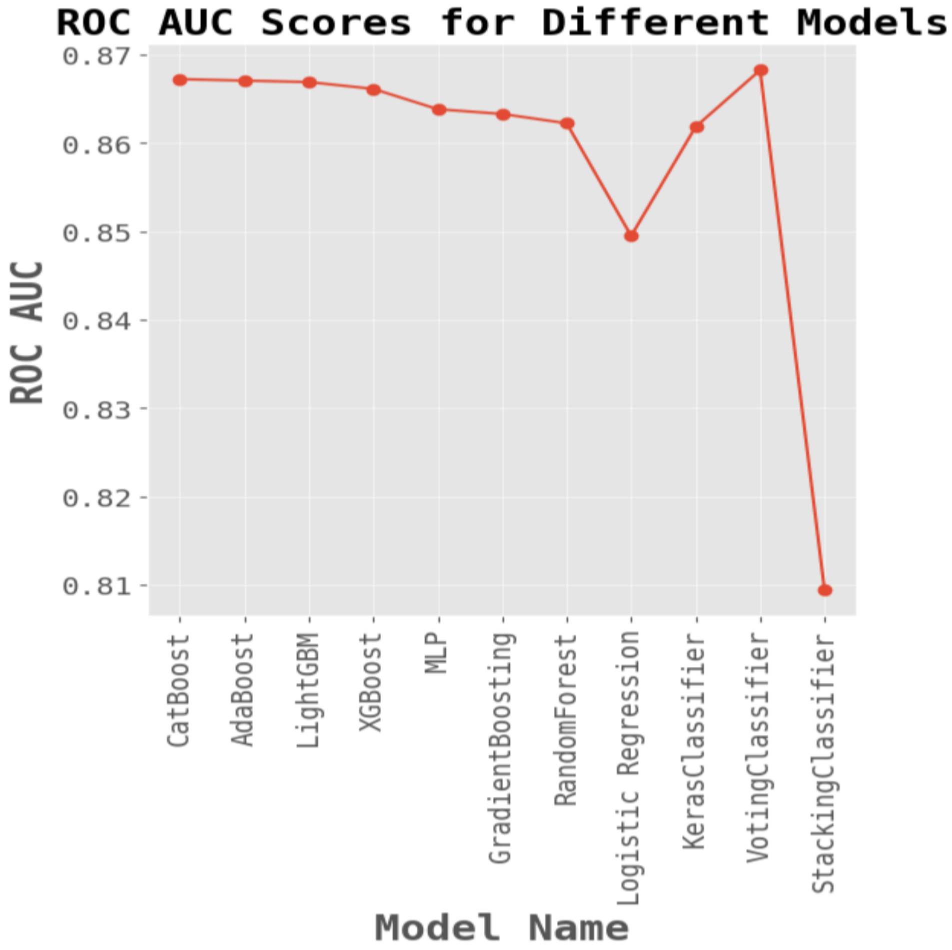
Stacking Classifier

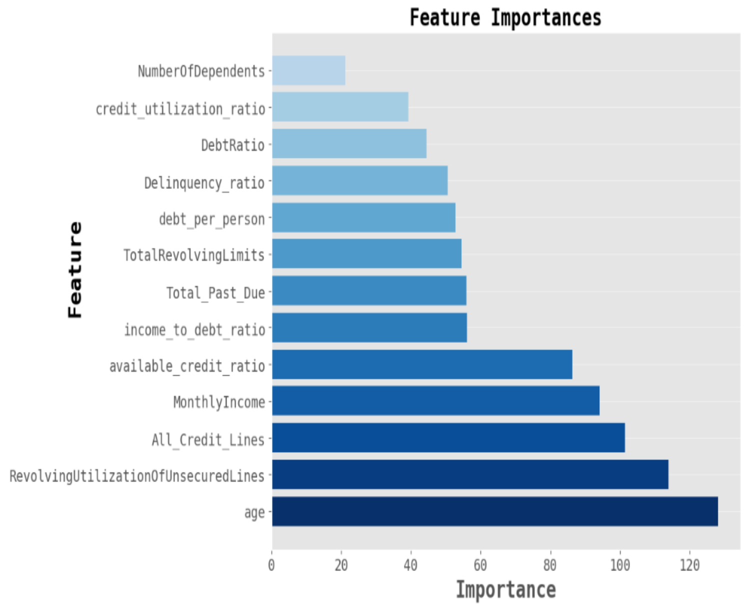
Model AUC Score: 0.80936

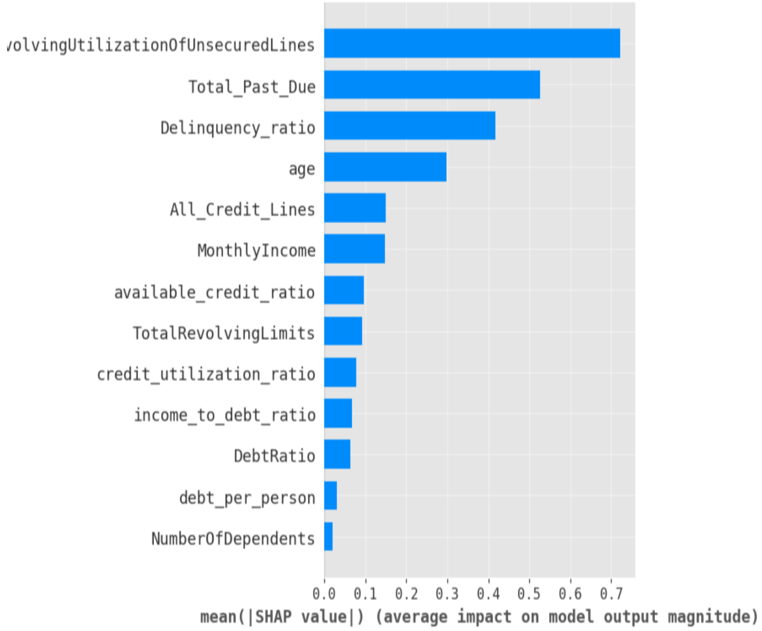
Voting Classifier

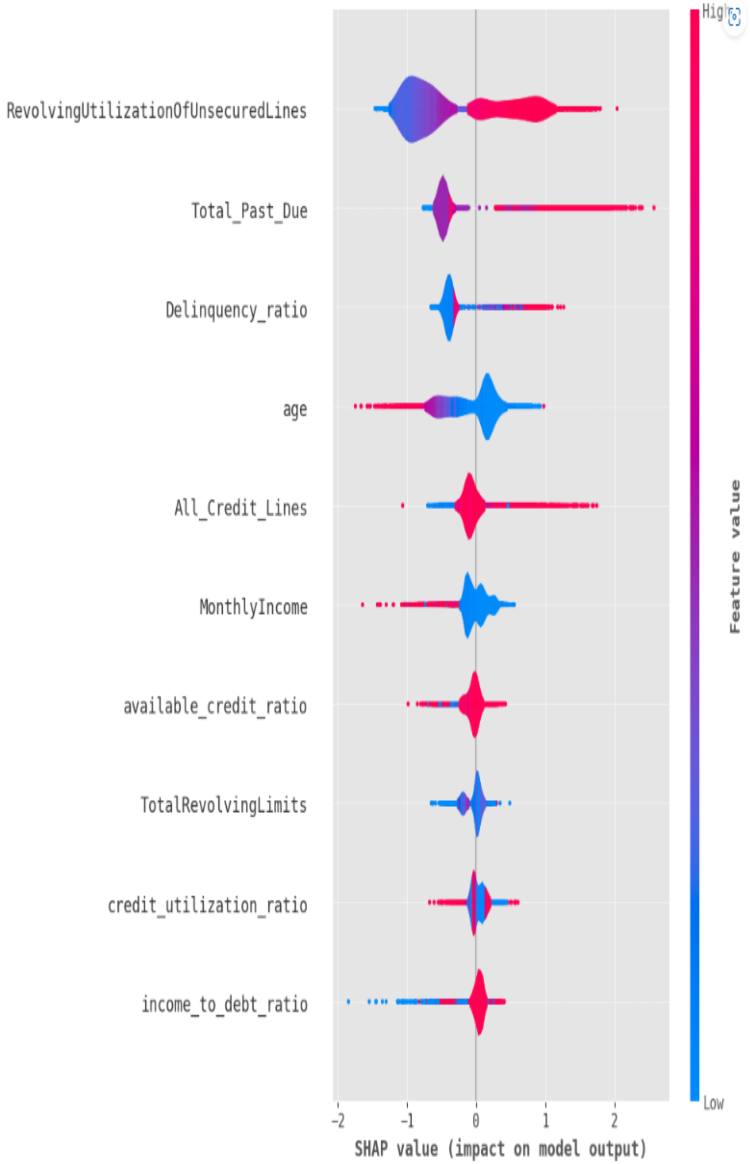
Model AUC Score: 0.86823

FINAL SCORES



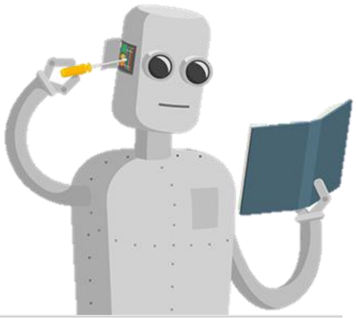






SHAP Summary Plots

MODEL TESTING & PIPELINE



```python
model = results['best_model'][9]
```

```python
df_test = pd.read_csv('cs-test.csv')
```

```python
df_test.drop('Unnamed: 0', axis=1, inplace=True)
```

```python
X_test = df_test.drop(columns=['SeriousDlqin2yrs'])
pred = model.predict_proba(X_test)[:,1]
pred
```

```python
clean_credit_data_ft = FunctionTransformer(clean_credit_data)
```

```python
feature_engineering_ft = FunctionTransformer(feature_engineering)
```

```python
steps = [('Data_Cleaning', clean_credit_data_ft), # step 1
         ('Feature_Engineering', feature_engineering_ft), # step 2
         ('Scaler', RobustScaler()), # step 3
         ('ROS', RandomOverSampler()), # step 4
         ('VotingClassifier', VotingClassifier(estimators=estimators, voting='soft'))] # step 5

pipeline = Pipeline(steps)
```

```python
# Use the loaded pipeline to make predictions

df_test = pd.read_csv('cs-test.csv') # Loading test dataset
df_test.drop('Unnamed: 0', axis=1, inplace=True)
X_test = df_test.drop(columns=['SeriousDlqin2yrs'])
y_pred = loaded_pipeline.predict(X_test)
y_pred
```

```
Pipeline
```

```
Pipeline(steps=[('Data_Cleaning',
                 FunctionTransformer(func=<function clean_credit_data at 0x00000232B2DED9D0>)),
                ('Feature_Engineering',
                 FunctionTransformer(func=<function feature_engineering at 0x00000232C4DE5A60>)),
                ('Scaler', RobustScaler()), ('ROS', RandomOverSampler()),
                ('VotingClassifier',
                 VotingClassifier(estimators=[('CatBoost',
                                               <catboost.core.CatBoostClas...
                                                             grow_policy=None,
                                                             importance_type=None,
                                                             interaction_constraints=None,
                                                             learning_rate=0.1,
                                                             max_bin=None,
                                                             max_cat_threshold=None,
                                                             max_cat_to_onehot=None,
                                                             max_delta_step=None,
                                                             max_depth=5,
                                                             max_leaves=None,
                                                             min_child_weight=None,
                                                             missing=nan,
                                                             monotone_constraints=None,
                                                             n_estimators=250,
                                                             n_jobs=None,
                                                             num_parallel_tree=None,
                                                             predictor=None,
                                                             random_state=42, ...))],
                                  voting='soft'))])
```

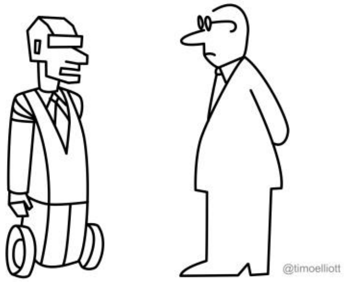
CONCLUSIONS / NEXT STEPS

INSIGHTS

- Objective & Purpose
- Kaggle Leadership Board 43rd Position
- Boosting Algorithms
- CatBoost
- Voting Classifier vs. Stacking Classifier
- Functions for Pipeline

NOW WHAT?

- Find score on test dataset
- Check Score change with feature/parameter tweak.
- Deploy the model
- Check with a large Deep Learning Model
- Check if F1 score model is possible.



"The good news is I have discovered inefficiencies. The bad news is that you're one of them."