Case Study

The objective of this case study is to create a delinquency model which can predict in terms of probabilities for each loan transaction, whether a consumer will be delinquent on his mortgage repayments in the next month.

Delinquency is a condition that arises when an activity or situation does not occur at its schedule or expected date; that is, it occurs later than expected.

In terms of Machine Learning, we will create a binary classifier to predict the probability of delinquency for each consumer.

Executive Summary

We ran multiple machine and deep learning models using sklearn library and H2O api. We found out that Logistic Regression has high Type I error and the Stacked Ensemble classifier improves the classification accuracy and reduces false positives.

The followings are the main characteristics of consumers that will delinquent on their mortgages:

- Consumers with loans originated before 2008 are more likely to miss a payment, especially consumers with loans from 2005 and 2006.
- Consumers with high levels of exposure (Total Exposure) are more likely to be delinquent.
- Consumers with low levels of estimate disposable income are more likely to be delinquent.
- Consumers with an interest rate between 4.5 and 5.5 are more likely to begin delinquent, especially loans with interest rates of 4.35
- The most likely consumers to become delinquent are consumers with properties in ZH, NB, and NH regions.

Exploratory Data Analysis

The data is about 3,000 consumers and is split into 4 files. In total the dataset has 415,229 records of loans from 2003 to 2013.

The dataset has numerical, categorical, dates, and missing values. Some date pre-processing needs to be done before doing any statistical analysis.

```
path = '/Users/andy42i/Documents/Documentos/Python/case_study_data/'
data01 = pd.read_csv(path + 'data01.csv', sep =',', header = 0)
data02 = pd.read_csv(path + 'data02.csv', sep =',', header = 0)
data03 = pd.read_csv(path + 'data03.csv', sep =',', header = 0)
data04 = pd.read_csv(path + 'data04.csv', sep =',', header = 0)
```

```
data01.head(5)
```

	DebtID	ConsumerID	LoanAge	InterestRate	NumberMonthsInArrears	CurrentBalance	EstDisposableIncome	ArrearsBalance	TotalExposure	IndexedTotalIn
0	2162571	353223	13	5.55	NaN	110000.0	389.0	NaN	110000.0	317
1	2162571	353223	14	5.55	NaN	110000.0	389.0	NaN	110000.0	317
2	2162571	353223	19	5.55	NaN	110000.0	389.0	NaN	110000.0	317
3	2162571	353223	23	5.55	NaN	110000.0	389.0	NaN	110000.0	317
4	2162571	353223	28	5.55	NaN	110000.0	389.0	NaN	110000.0	317

```
data01.dtypes
```

```
DebtID                    int64
ConsumerID                int64
LoanAge                   int64
InterestRate            float64
NumberMonthsInArrears   float64
CurrentBalance          float64
EstDisposableIncome     float64
ArrearsBalance          float64
TotalExposure           float64
IndexedTotalIncome      float64
OriginalBalance         float64
MaturityDate             object
LoanOriginationDate      object
PropertyRegion           object
ReportDate               object
ConsumerAge               int64
OriginalPropertyValue   float64
PropertyIndexFactor     float64
ForeclosureValue        float64
dtype: object
```

Data Pre-processing

First, we append the 4 datasets into a complete one. We also converted Maturity Date, Loan Origination Date, and Report Date from object to data type and DebitID from file data04 from float to int.

We need to convert dates from object to data type

```
data01['MaturityDate'] = data01['MaturityDate'].astype('datetime64')
data01['LoanOriginationDate'] = data01['LoanOriginationDate'].astype('datetime64')
data01['ReportDate'] = data01['ReportDate'].astype('datetime64')

data02['MaturityDate'] = data02['MaturityDate'].astype('datetime64')
data02['LoanOriginationDate'] = data02['LoanOriginationDate'].astype('datetime64')
data02['ReportDate'] = data02['ReportDate'].astype('datetime64')

data03['MaturityDate'] = data03['MaturityDate'].astype('datetime64')
data03['LoanOriginationDate'] = data03['LoanOriginationDate'].astype('datetime64')
data03['ReportDate'] = data03['ReportDate'].astype('datetime64')

data04['MaturityDate'] = data04['MaturityDate'].astype('datetime64')
data04['LoanOriginationDate'] = data04['LoanOriginationDate'].astype('datetime64')
data04['ReportDate'] = data04['ReportDate'].astype('datetime64')
data04['DebtID'] = data04['DebtID'].astype(int)
```

```
data = data01.append([data02, data03,data04])
data.shape
```

```
(415229, 19)
```

Feature Engineering

We calculated some extra features to better understand consumer behaviour and delinquency:

1. Term – is the length of the loan, i.e., is the difference between Maturity Data and Loan Origination Date.
2. preTime – is the number of months between the date of the Last Report Date and the Loan Origination Date.
3. postTime – is the number of months between the date of the Maturity Date and Last Report Date.
4. Year – we extract the year the loan was originated.
5. Month - we extract the month the loan was originated.
6. StartConsumerAge – age of the consumer when the loan was originated.
7. CurrentConsumerAge – age of the consumer at the time of the last report date.
8. DiffConsumerAge - is the difference between the current consumer age and the start consumer age.
9. GroupConsumerAge – we created 5 age segments.

```
from datetime import datetime

def diff_month(d1, d2):
    return (d1.year - d2.year) * 12 + d1.month - d2.month

df_agg['MaturityDate'] = pd.to_datetime(df_agg['MaturityDate'])
df_agg['LoanOriginationDate'] = pd.to_datetime(df_agg['LoanOriginationDate'])

df_agg['Term'] = (df_agg['MaturityDate'].apply(lambda x: x.year) - df_agg['LoanOriginationDate'].apply(lambda x:
```

```
df_agg['ReportDate'] = pd.to_datetime(df_agg['ReportDate'])
df_agg['LoanOriginationDate'] = pd.to_datetime(df_agg['LoanOriginationDate'])

df_agg['preTime'] = (df_agg['ReportDate'].apply(lambda x: x.year) - df_agg['LoanOriginationDate'].apply(lambda x:
```

```
df_agg['postTime'] = (df_agg['MaturityDate'].apply(lambda x: x.year) - df_agg['ReportDate'].apply(lambda x: x.yea
```

```
df_agg['Year'] = df_agg['LoanOriginationDate'].apply(lambda x: x.year)
df_agg['Month'] = df_agg['LoanOriginationDate'].apply(lambda x: x.month)
```

```
df_agg['GroupConsumerAge'] = pd.DataFrame(pd.qcut(df_agg['CurrentConsumerAge'],5,labels= False))
```

```
df_agg['DiffConsumerAge'] = df_agg['CurrentConsumerAge'] - df_agg['StartConsumerAge']
```

Checking Missing Values

There are 4 variables that have missing values: Number of Months in Arrears, Estimated Disposable Income, Arrears Balance and Foreclosure Value.

For Number of Months in Arrears and Arrears Balance we fill missing values with zero. For Estimated Disposable Income and Foreclosure Value we created an imputer object with the strategy set to the median.

```
df_agg.isna().sum()
```

```
DebtID                      0
ConsumerID                  0
PropertyRegion              0
MaturityDate                0
LoanOriginationDate         0
StartConsumerAge            0
CurrentConsumerAge          0
InterestRate                0
NumberMonthsInArrears    6563
CurrentBalance              0
EstDisposableIncome        70
ArrearsBalance           6563
TotalExposure               0
OriginalBalance             0
OriginalPropertyValue       0
ForeclosureValue           69
ReportDate                  0
Term                        0
preTime                     0
postTime                    0
Year                        0
Month                       0
status                      0
dtype: int64
```

```
#Fill NaN with 0 for NumberMonthsInArrears
df_agg['NumberMonthsInArrears'].fillna(0, inplace=True)
#Fill NaN with 0 for ArrearsBalance
df_agg['ArrearsBalance'].fillna(0, inplace=True)
```

```
from sklearn.preprocessing import Imputer, StandardScaler

imputer = Imputer(strategy='median')

# Train on the training features
imputer.fit(X_train)

# Transform both training and testing data
X_train = pd.DataFrame(imputer.transform(X_train), columns=X_train.columns)
X_test = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)
```

```
/Users/andy42i/anaconda3/lib/python3.6/site-packages/sklearn/utils/deprecation.py:58: DeprecationWarning: Class
Imputer is deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22. Import impute.Simple
Imputer from sklearn instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Target Variable

We created a binary variable called status that indicates whether the consumer has pending payments or not. We used Number of Months in Arrears variable because we are trying to predict delinquency. We define a delinquent consumer if the consumer has missed a payment and has pending balance.

There are 1,669 instances with missed payments which represent almost 20% of all the dataset. This means that our dataset is highly imbalanced. We will remove these two variables before running the machine learning and deep learning models however, we will keep them for the statistical and exploratory analysis.

```
df_agg['status'] = df_agg["NumberMonthsInArrears"].apply(lambda x: 'Yes' if x > 0 else 'No')
df_agg['status'].value_counts(normalize=False, dropna=False)
```

```
No     6862
Yes    1669
Name: status, dtype: int64
```

```
df_agg['status'].value_counts(normalize=True, dropna=False)
```

```
No     0.804361
Yes    0.195639
Name: status, dtype: float64
```

In total 736 consumers have positive arrears balance and 635 have one or more months in arrears.

How many consumers have arrears balance

```
arrears = df_agg[df_agg.ArrearsBalance > 0]
len(np.unique(arrears.ConsumerID))
```

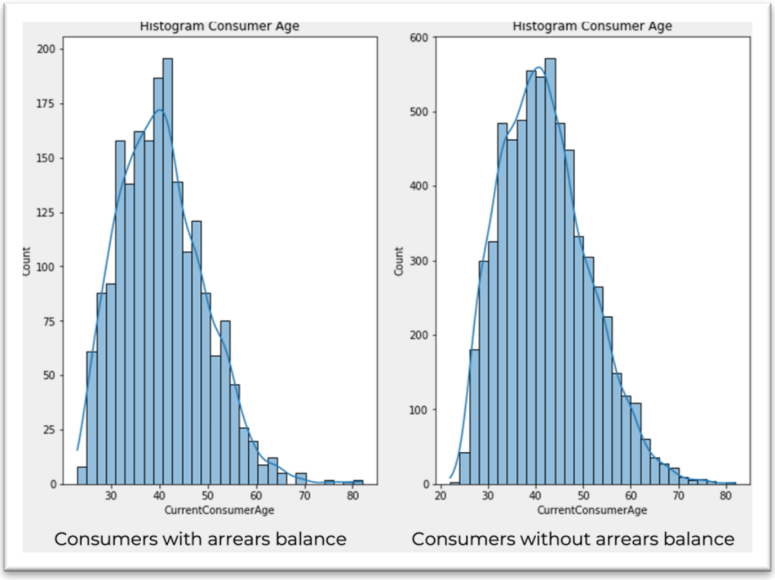
```
736
```

```
monthsarrears = df_agg[df_agg.NumberMonthsInArrears > 0]
len(np.unique(monthsarrears.ConsumerID))
```

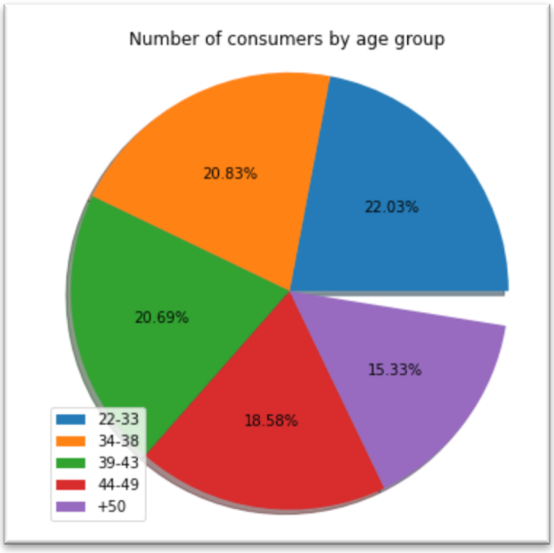
```
635
```

Below, we describe the characteristics of delinquent consumers and non-delinquent.

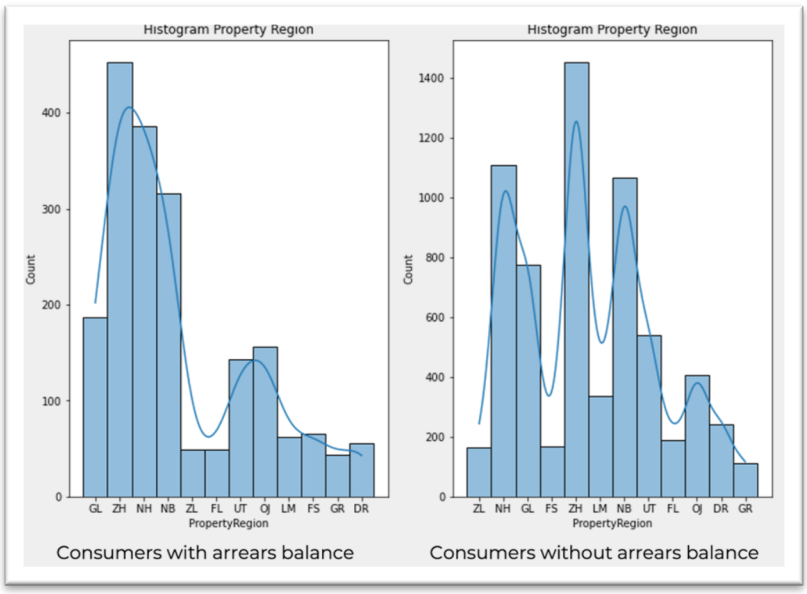
The distribution of both types of consumers is very similar. There is no big difference in mean age between delinquent and non-delinquent consumers. The former is 40 while the latter is 42.



Analysing consumers by group age, we can see that the age group between 22 and 33 has 22% of consumers with pending balance.



The property region distribution is also very similar for both types of customers. The top three property regions for delinquent consumer and non-delinquent are ZH, NH, and NB. Property regions GL, UT, and OJ have the similar number of delinquent consumers however, this pattern changes for non-delinquent consumers where GL has more consumers compared to UT, and OJ.



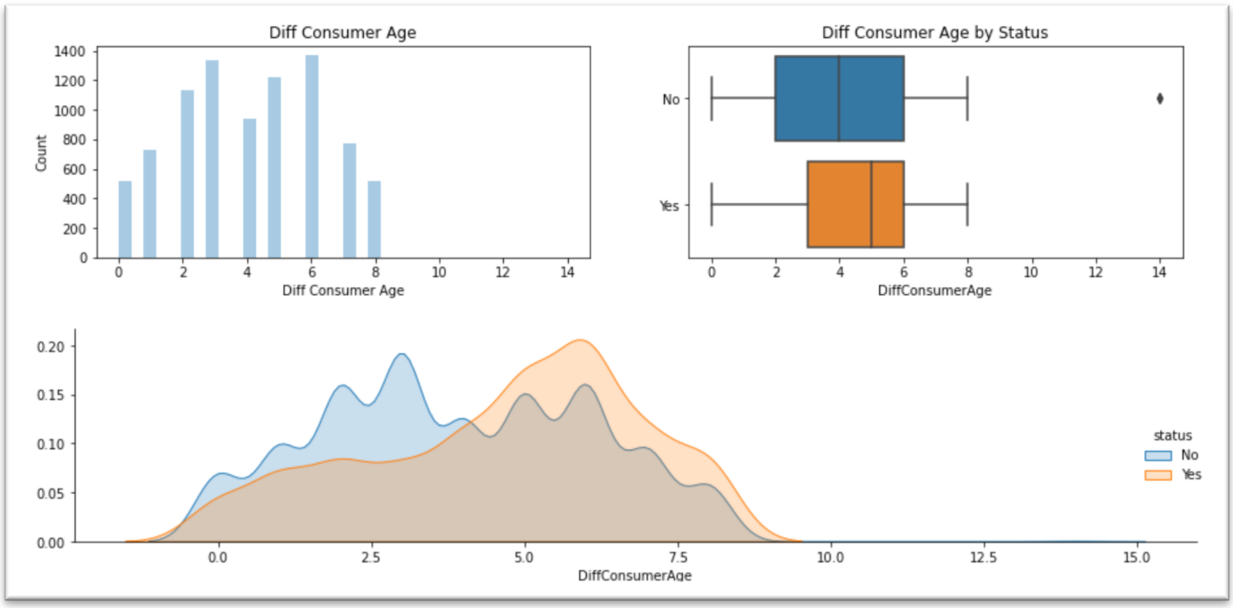
Checking Correlations Among Features

To further analyse the correlations between our target variable and the other features, we created a function that compares the distribution of a given variable against the distribution of the target variable.

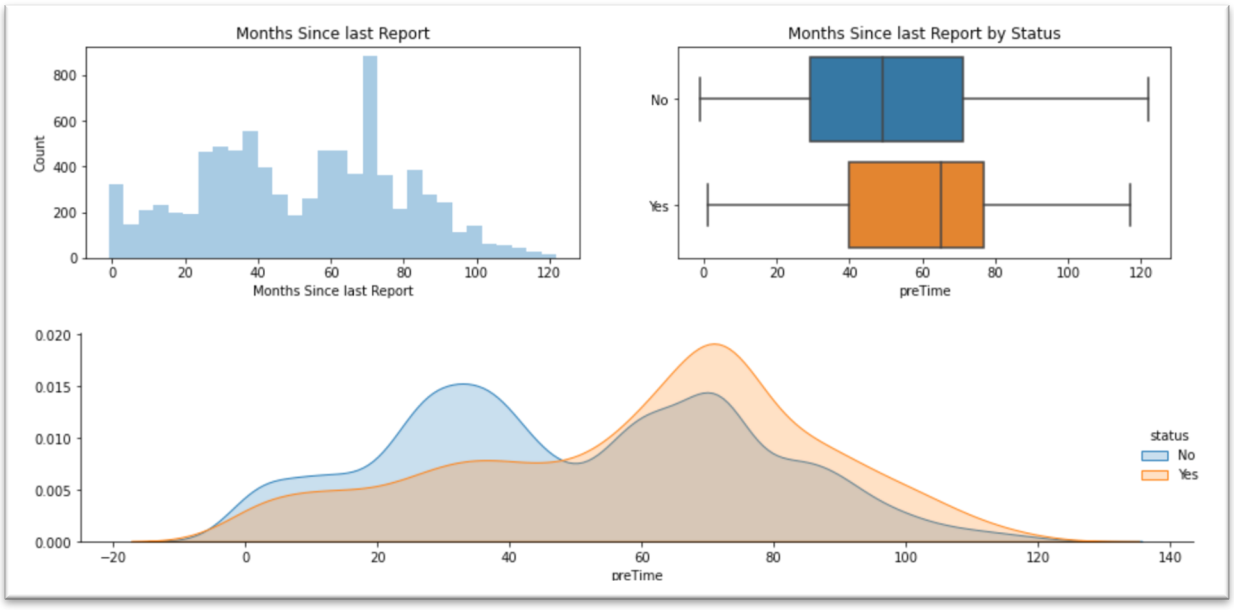
```
# make general plots to examine each feature
def plot_var(col_name, full_name, continuous):

    data = df_agg

    fig, (ax1, ax2) = plt.subplots(1, 2, sharex=False, figsize=(15,3))
    # plot1: counts distribution of the variable

    if continuous:
        sns.distplot(data.loc[data[col_name].notnull(), col_name], kde=False, ax=ax1)
    else:
        sns.countplot(data[col_name], order=sorted(data[col_name].unique()), color='#5975A4', saturation=1, ax=ax
    ax1.set_xlabel(full_name)
    ax1.set_ylabel('Count')
    ax1.set_title(full_name)


    # plot2: bar plot of the variable grouped by loan_status
    if continuous:
        sns.boxplot(x=col_name, y='status', data=data, ax=ax2)
        ax2.set_ylabel('')
        ax2.set_title(full_name + ' by Status')
    else:
        Charged_Off_rates = data.groupby(col_name)['status'].value_counts(normalize=True)[:,'Yes']
        sns.barplot(x=Charged_Off_rates.index, y=Charged_Off_rates.values, color='#5975A4', saturation=1, ax=ax2)
        ax2.set_ylabel('Fraction of Delinquent')
        ax2.set_title('Delinquent by ' + full_name)
        ax2.set_xlabel(full_name)

    # plot3: kde plot of the variable gropued by loan_status
    if continuous:
        facet = sns.FacetGrid(data, hue = 'status', size=3, aspect=4)
        facet.map(sns.kdeplot, col_name, shade=True)
        facet.add_legend()
    else:
        fig = plt.figure(figsize=(12,4))
        sns.countplot(x=col_name, hue='status', data=data, order=sorted(data[col_name].unique()) )

    plt.tight_layout()
```

Below, we highlight some interesting findings:

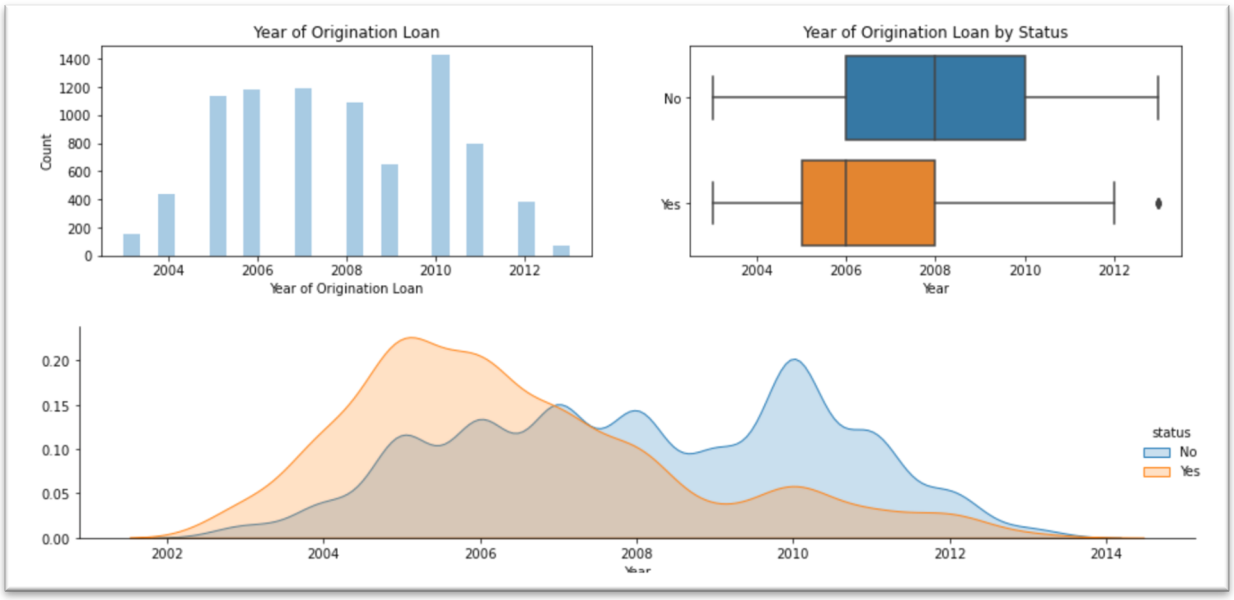
We observe different distributions when comparing the difference in age and the target variable. The age difference for delinquent consumer is bigger than for non-delinquent. That means that as time passes, consumers are more likely to become delinquent compared to recent consumers.



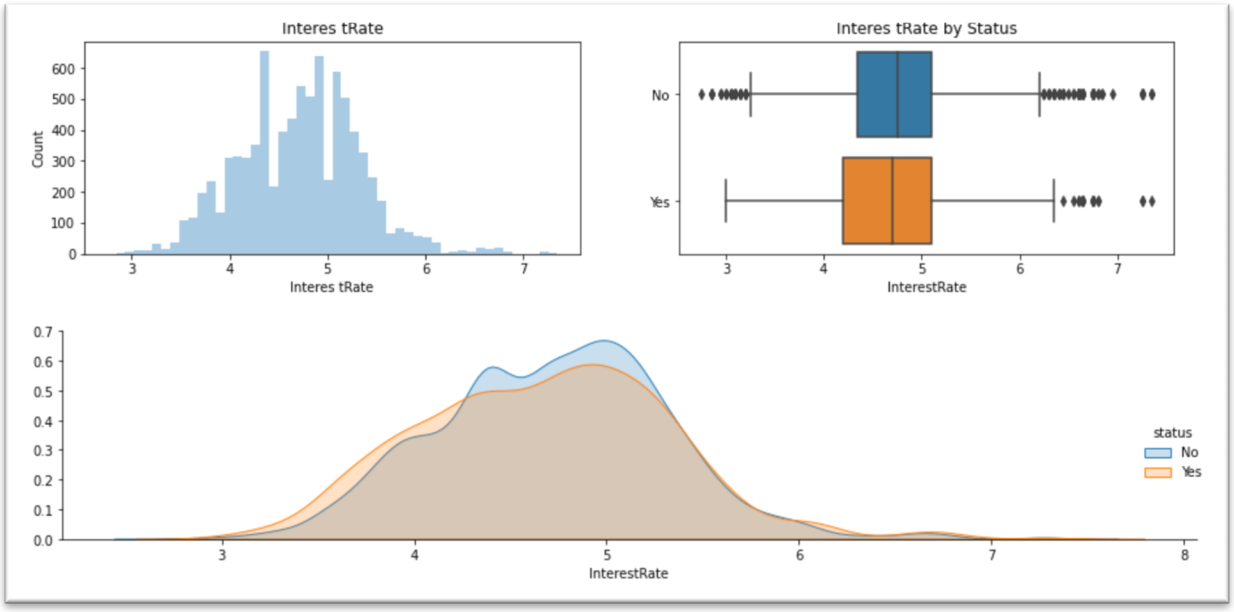
We observe a similar pattern when comparing with preTime variable; that is, the number of months between the report date and the loan origination date. This suggests that recent consumers are less likely to miss a payment.



In relation to the year of origination of the loan, consumers that originated a loan before 2008 are more likely to become delinquent.



Finally, consumers with an interest rate between 4.5 and 5.5 are more likely to become delinquent.

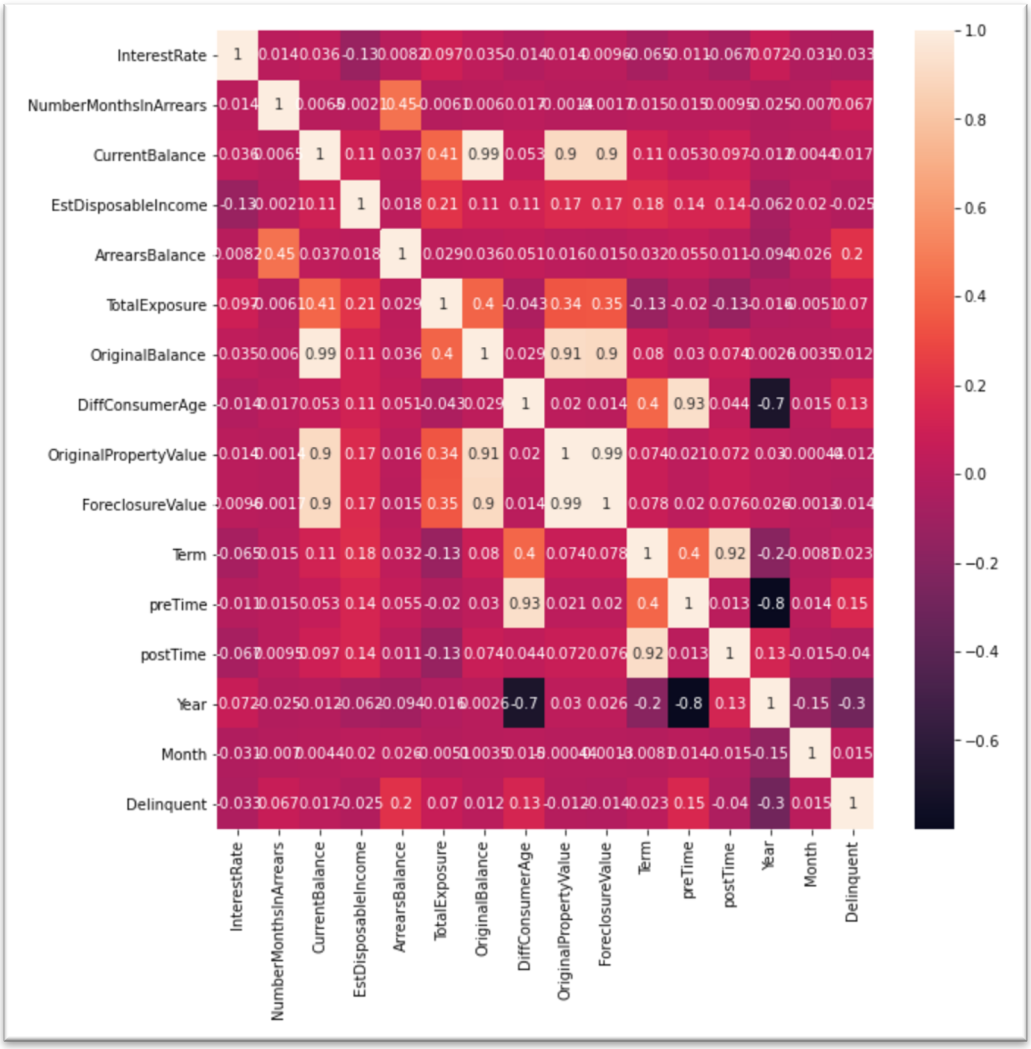


In the following, we compare the correlation between variables. To do so, we calculate Pearson correlation to evaluate whether there is statistical evidence for a linear relationship among the variables.

The linearly correlated features are:

Variable 1	Variable 2	Correlation Value
Current Balance	Original Balance	0.99
Current Balance	Original Property Value	0.9
Current Balance	Foreclosure Value	0.9
Current Balance	Total Exposure	0.41
Original Balance	Original Property Value	0.91
Original Balance	Foreclosure Value	0.9
Number Months in Arrears	Arrears Balance	0.45
Total Exposure	Original Balance	0.4
Diff Consumer Age	Term	0.4
Diff Consumer Age	preTime	0.93

We will remove some of these variables to avoid multicollinearity. Also, we can observe that there is no strong evidence of correlation between the variables and the target variable.



Model Design

We are going to further clean the remaining data and get them ready for the machine learning model. First, we convert the categorical variables to dummy variables and we eliminate the first one to avoid collinearity.

```
dummy_list = ['PropertyRegion','GroupConsumerAge','Year','Month']
df = pd.get_dummies(df_agg, columns=dummy_list, drop_first=True)
```

We scale the data using StandardScaler function. We do this because features are measured in different units, and they cover different ranges. Some ML models, such as SVM, KNN that consider distance measures between observations are therefore significantly affected by the range of the features and scaling allows them to learn. While some methods, such as Linear Regression and Ransom Forest do not actually require feature scaling, it's still best practice to take this step when we are comparing multiple algorithms.

```
# Create an imputer object with a median filling strategy
scaler = StandardScaler()

# Train on the training features
scaler.fit(X_train)

# Transform both training and testing data
X_train = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns)
X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
```

In the following sections, we present different solutions to estimate the probability of a consumer will miss a payment.

Logistic Regression with SGD

The first model we present is a Logistic Regression with stochastic gradient descent training to improve the performance of the model. Gradient Descent is the process of minimising a function by following the gradients of the cost function. This technique updates the coefficients in every iteration to minimize the error of the model on our training data.

We set up a pipeline to run multiple models with different parameters and we used grid search to find the best parameters.

From the results, the best model is when we use L1 as the penalty function and alpha = 0.001 as the regularization term. The model achieves 72.9% training accuracy.

```
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline

pipeline_sgdlr = Pipeline([
    ('model', SGDClassifier(loss='log', max_iter=1000, tol=1e-3, random_state=0, warm_start=False))
])

param_grid_sgdlr  = {
    'model__alpha': [10**-5, 10**-3, 10**-1,10**1 ,10**2],
    'model__penalty': ['l1', 'l2']
}

grid_sgdlr = GridSearchCV(estimator=pipeline_sgdlr, param_grid=param_grid_sgdlr, scoring='roc_auc', n_jobs=-1, pr

grid_sgdlr.fit(X_train, y_train)

Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:    5.9s finished

GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('model', SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=1000,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=0, shuffle=True, tol=0.001,
       validation_fraction=0.1, verbose=0, warm_start=False))]),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'model__alpha': [1e-05, 0.001, 0.1, 10, 100], 'model__penalty': ['l1', 'l2']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
       scoring='roc_auc', verbose=1)

sgdlr_estimator = grid_sgdlr.best_estimator_
print('Best score: ', grid_sgdlr.best_score_)
print('Best parameters set: \n', grid_sgdlr.best_params_)

Best score:  0.7291721523745747
Best parameters set:
 {'model__alpha': 0.001, 'model__penalty': 'l1'}
```

Given the best model, we proceed to analyse the feature coefficients. In the table below we show the top 10 features.

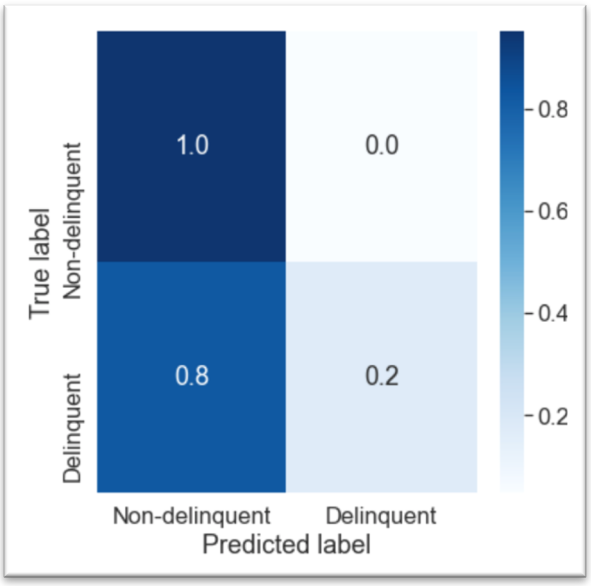
We can see that the odds of a delinquent consumer with a loan from 2010 is exp(-1.024) = 0.3591. In terms of percent change, we can say that the odds for a delinquent consumer with a loan from 2010 are 64% less than the odds for other consumers. This is aligned with the results from the exploratory analysis where we point out that consumers that originated a loan before 2008 are more likely to be delinquent.

The coefficient for Difference in Consumer Age (DiffConsumerAge) says that we will see 29.5% increase in odd of missing payments.

Variable	Coefficient	Odds	Percent Change
Year_2010	-1.024	0.359155	-0.640845
Year_2011	-0.922	0.397723	-0.602277
Year_2009	-0.673	0.510176	-0.489824
Year_2008	-0.643	0.525713	-0.474287
Year_2012	-0.559	0.571781	-0.428219
preTime	-0.550	0.576950	-0.423050
Year_2007	-0.351	0.703984	-0.296016
Year_2013	-0.331	0.718205	-0.281795

DiffConsumerAge	0.259	1.295634	0.295634
PropertyRegion_OJ	0.191	1.210459	0.210459

Analysing the results with the confusion matrix, we can see that the model is classifying incorrectly delinquent consumers. We falsely predicted 80% of delinquent consumers (Type I error). This could be because the dataset is highly imbalanced, and the classifier is biased towards the majority class.



In the next section, we will try to improve the classification accuracy by running multiple machine learning and deep learning models.

H2O

For our next experiment, we use an open-source API to run multiple Machine Learning and Deep Learning models called H2O.

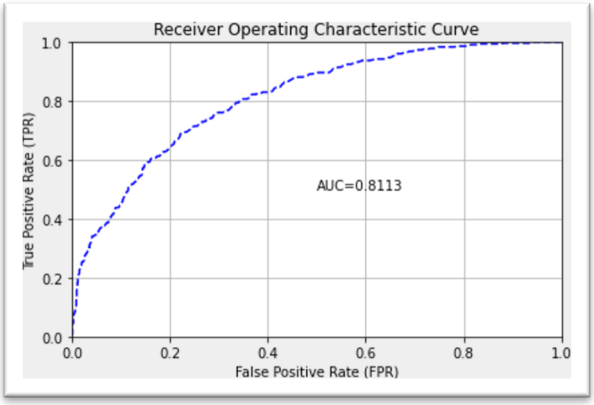
The best model is a Stack Ensemble. Stacked Ensemble method is a supervised ensemble machine learning algorithm that finds the optimal combination of a collection of prediction algorithms using a process called stacking.

The second-best model is Gradient Boosting Machines. Boosting is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set.

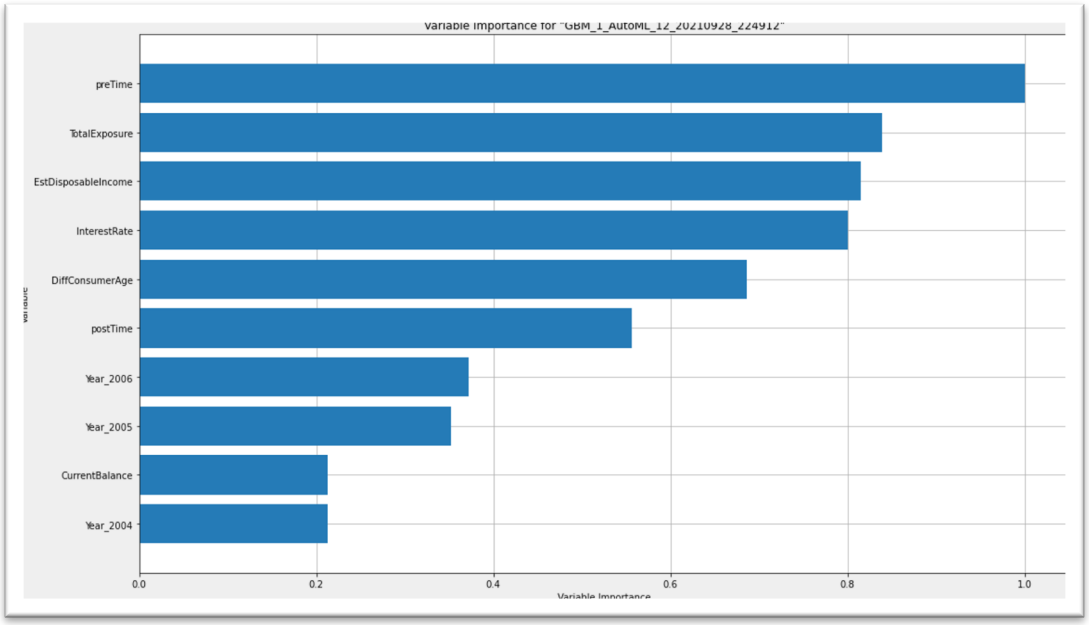
Below we show the performance of the top models

model_id	auc	logloss	aucpr	mean_per_class_error	rmse	mse
StackedEnsemble_BestOfFamily_1_AutoML_12_20210928_224912	0.819608	0.414296	0.589052	0.263282	0.363676	0.13226
GBM_1_AutoML_12_20210928_224912	0.808811	0.428902	0.569375	0.27869	0.370899	0.137566
XGBoost_1_AutoML_12_20210928_224912	0.806298	0.425726	0.566949	0.266178	0.3692	0.136308
DRF_1_AutoML_12_20210928_224912	0.778745	1.39719	0.551199	0.290081	0.379905	0.144328
GBM_2_AutoML_12_20210928_224912	0.746441	0.501775	0.483162	0.311444	0.4037	0.162974
GBM_4_AutoML_12_20210928_224912	0.744934	0.512418	0.460848	0.308607	0.408879	0.167182
XGBoost_2_AutoML_12_20210928_224912	0.740646	0.49093	0.486842	0.326054	0.399175	0.159341
GLM_1_AutoML_12_20210928_224912	0.738924	0.474045	0.456854	0.319089	0.391551	0.153312
GBM_3_AutoML_12_20210928_224912	0.733511	0.514572	0.447677	0.32504	0.409881	0.168002

The Stacked Ensemble achieves an AUC of 81.96% which improves the classification accuracy compared to the first model we presented before, the Logistic Regression with stochastic gradient descent.



The variable importance is shown in the plot below. We can see that the number of months between the loan origination date and the report date is the most important feature followed by Total Exposure, Estimated Disposable Income, and Interest Rate.



To get more insights into how our model works, we use SHAP values. SHAP is based the explanations on shapely values - measures of contributions each feature has in the model.

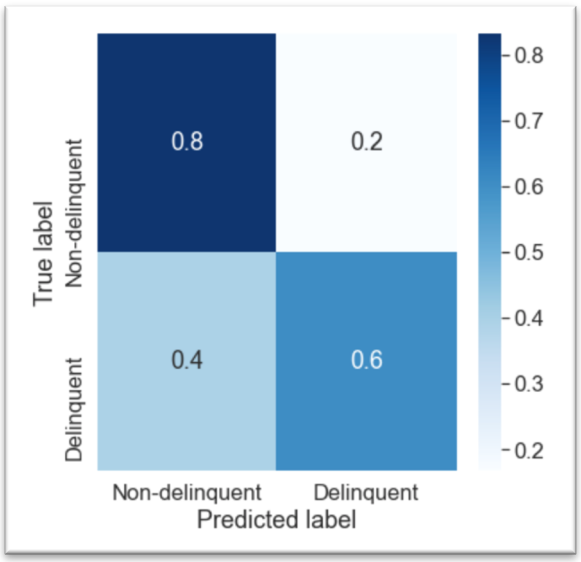
It demonstrates the following information:

- Feature importance: Variables are ranked in descending order.
- Impact: The horizontal location shows whether the effect of that value is associated with a higher or lower prediction.
- Value: Colour shows whether that variable is high (in red) or low (in blue) for that observation.

The image below shows the positive and negative relationships of the predictors with the target variable. For example, A high level of the "Difference in consumer age" has a high and positive impact on delinquent consumers. The "high" comes from the red colour, and the "positive" impact is shown on the X-axis. Similarly, we will say the high levels of "Total Exposure" are positively correlated to delinquent consumers.

We can also see those consumers with loans from 2005 and 2006 are more likely to be delinquent. Also, consumers with low levels of estimate disposable income are more likely to be delinquent.

The confusion matrix shows an improvement in Type I error, we can see that the model is classifying incorrectly 40% of delinquent consumers as non-delinquent (the logistic regression that falsely predicted 80%).



We created clusters of consumers. We can see that they look very similar however, there is some local structure. The local structure is more evident with delinquent consumers.

This local structure is given by the property region and interest rate where the most likely consumers to become delinquent are consumers with properties in ZH, NB, and NH regions with interest rates of 4.35 and 3.95.