PROJECT REPORT

Project Group 11

Names:	Jay Bharadva	Sagar Bharatkumar Raval	Pratyush Sharma	Rushi Sharma
IDs:	015958449	015906683	015570399	015947113

INDEX OF CONTENTS

Loan-Defaulters

Colab Notebook Link (Click here)

Description

This project aims to build a classification model which can predict if the loan can be defaulted by a person given the loan and some personal information provided. Users can use this model as a reference tool for any financial institution in the decision-making process of whether to approve the loan to the loan appliers. By applying it, an institution can do the risk analysis maximizing the profit. Loans can be of various types, including

- Cash Loan
- Personal/Consumer Loan
- Revolving Loan

If a new applicant is less likely or not able to pay back the loan amount or credit, it is not an economical step to approve the loan. Thus, it helps financial institutions by avoiding the loss of funds.

Approach

Dataset & Pre-processing

- The Kaggle dataset we've used in this project is available here, https://www.kaggle.com/gauravduttakiit/loan-defaulter
- Dataset Dimension,

  Instances: **307511**

  Features: **122**
- To pre-process our data, we implemented a few methods in our code so that the quality of our results is not affected. We employed methods to deal with missing values and thus making our data cleaner for training, followed by testing.

- **Dropping Features**:
  - The training data that we are using has multiple features that have the potential to help predict loan defaulters. However, certain features have such a high amount of missing

values that they do not accurately represent the properties of the feature and we deemed them not ideal as a part of our learning dataset. Therefore, we are dropping the features that have more than 50% of values missing as a pre-processing approach. We can see in the notebook that compared to the initial 122 features, we have a total of 70 interesting features which we get after dropping features with a high amount of missing values.

- Before dropping:

```
In [4]: print ("application_data     :",application_data.shape)
        application_data     : (307511, 122)
```

- After Dropping:

```
In [11]: print ("application_data     :",application_data.shape)
         application_data     : (307511, 70)
```

- **Handling Missing Values**:
  - After dropping the majority missing values features in the dataset we are left with interesting features which may still have some missing values. To deal with this issue, we are handling those missing values with appropriate statistics after plotting data and judging the trend of missing values. We have three sub-approaches for our data.
  - **Replacing with Mean**:
    - For features with symmetric distribution and non-skewed data, we are replacing the null values with the mean of the feature's values. As an example, the feature AMT_GOODS_PRICE represents the price of goods for which a consumer loan is given. On the initial check, there are 278 missing values whose number is later converted to 0 after replacing the missing values with the mean.

```
Missing Values in 'AMT_GOODS_PRICE' Before :  278
Missing Values in 'AMT_GOODS_PRICE' After :  0
```

  - **Replacing with Mode**:
    - When the data distribution is skewed and data type as categories, we are replacing the null values with the mode of the feature's values. As an example, the feature NAME_TYPE_SUITE represents the individual/s who are accompanying the client when they are applying for the loan. On the initial check, out of the total 306219 values; there are 1292 missing values whose number is later converted to 0 after replacing the missing values with the mode.

```
Missing Values in 'NAME_TYPE_SUITE' Before :  1292
Missing Values in 'NAME_TYPE_SUITE' After :  0
```

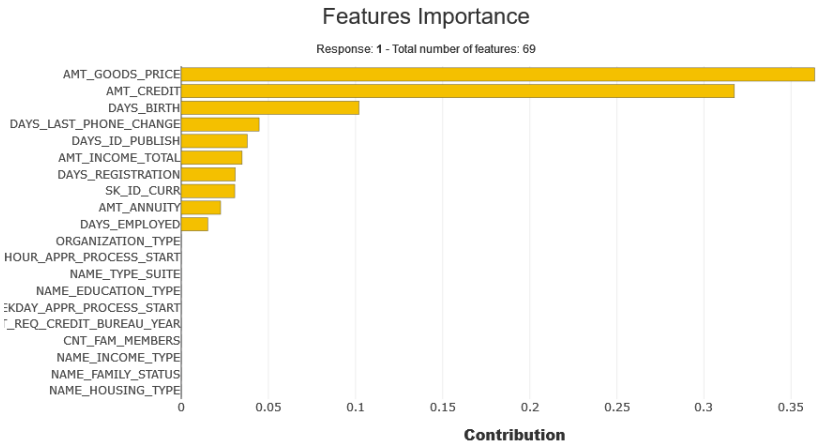
- **Replacing with Median**:
  - For skewed data and features with whole numbers as their values, we are replacing the null values with the median of the feature's values. As an example, the feature OBS_30_CNT_SOCIAL_CIRCLE represents the observation of the client's social surroundings within the last 30 days. On the initial check, out of the total 306490 values; there are 1021 missing values whose number is later converted to 0 after replacing the missing values with the median.

```
Missing Values in 'OBS_30_CNT_SOCIAL_CIRCLE' Before :  1021
Median :  0.0
Missing Values in 'OBS_30_CNT_SOCIAL_CIRCLE' After :  0
```

  - As there are multiple interesting features in the dataset with whole numbers as their values, we replace missing values for the following features with their respective median values:
    - OBS_60_CNT_SOCIAL_CIRCLE
    - AMT_REQ_CREDIT_BUREAU_HOUR
    - AMT_REQ_CREDIT_BUREAU_DAY
    - AMT_REQ_CREDIT_BUREAU_WEEK
    - AMT_REQ_CREDIT_BUREAU_MON
    - AMT_REQ_CREDIT_BUREAU_QRT
    - AMT_REQ_CREDIT_BUREAU_YEAR

Feature Selection

- We have used "Shapash" library to get the features from the dataset. Shapash needs to be installed by `!pip install shapash`.
- Shapash provides a Smart Explainer due to which the model can be easily understood.
- We got the features' importance as below,



- We have put features on y-axis and contribution of those features in training model on x-axis.
- From above listed features, we selected features with a contribution more than 0.015 i.e. from AMT_GOODS_PRICE to DAYS_EMPLOYED features from the list.
- We used the following methods in order to accomplish this task:
    1. “SmartExplainer” which allows us to perform many operations to make the results more understandable
    2. “compile” which performs the sorting of contributions, the reverse preprocessing steps and performs all the calculations necessary for a quick display of plots
    3. “features_importance” to plot the features which are more important

Splitting the dataset

- We have used “train_test_split” method to split the data into random train and test subsets

```
X_train, X_test, y_train, y_test = train_test_split(X_imp, y, test_size=0.2, stratify=y)
```

- Here, we kept test_size 0.2, thus we divided the data in test data subset and train data subset in 1:4 ratio
- The proportion of values in the sample produced will be the same as the dataset itself because we’ve passed “y” as stratify parameter value.

Classifier & Parameter Selection

- Module “tune-sklearn” integrates hyperparameters tuning and scikit-learn’s Classifier API. We’ve used an API named TuneGridSearchCV from tune-sklearn module in order to get the best combination of parameter values for different classifiers to get the highest accuracy in each case.
- The classifiers we checked during this task:

1) LogisticRegression (parametric classification model)
    - Parameters entered in and chosed by TuneGridSearchCV :-

```
Dictionary of parameters to test on LogisticRegression :
      {'solver': ['newton-cg',
       'lbfgs', 'liblinear'],
       'penalty': ['l2']}
Best parameters for Logistic Regression are :
      {'solver': 'lbfgs',
       'penalty': 'l2'}
which yields 91.92722306228964 % accuracy
```

2) RandomForestClassifier (supervised learning algorithm)
    - Parameters entered in and chosed by TuneGridSearchCV :-

```
Dictionary of parameters to test on RandomForestClassifier :
      {'n_estimators': [200, 500],
       'max_features': ['auto', 'sqrt', 'log2'],
       'class_weight': ['balanced'],
       'random_state': [42],
       'max_samples': [50000]}
Best parameters for RandomForestClassifier are :
      {'n_estimators': 200,
       'max_features': 'auto',
       'class_weight': 'balanced',
       'random_state': 42,
       'max_samples': 50000}
which yields 91.93004477888596 % accuracy
```

- As per the results above, we can observe that Random Forest Classifier gave slightly higher accuracy than Logistic Regression's on the **X_train** and **y_train** we got in the previous step.
- So, we're keeping Random Forest Classifier with the best parameter combination to proceed further.

Experimental Results

Evaluation Metrices

- Now, we are trying to get Evaluation Metrices using the whole data applying Random Forest Classifier.
- **Accuracy Score:** We've used accuracy_score method from sklearn.metrics library. Loan Defaulters prediction is a type of binary classification, thus the Accuracy Score will be equal to Jaccard Score.

```
Accuracy of RandomForestClassifier model : 0.983854236108627
```

- **Confusion Matrix:** It gives deeper insights by showing the correct and incorrect predictions on each class.

```
Confusion Matrix :
[[282686 0]
[ 4965 19860]]
```

- **Classification Report:** Gives details about Precision and Recall.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Precision measures how good our model is when the prediction is positive. Recall measures how good our model is at correctly predicting positive classes.

```
Classification Report :
              precision    recall  f1-score   support

           0       0.98      1.00      0.99    282686
           1       1.00      0.80      0.89     24825

    accuracy                           0.98    307511
   macro avg       0.99      0.90      0.94    307511
weighted avg       0.98      0.98      0.98    307511
```

Parameters

- TuneGridSearchCV chosed parameters for different classifiers as below,

1) Logistic Regression

penalty : ‘l2’

adds an L2 penalty equal to the square of the magnitude of coefficients. It can also be called Ridge Regression as it’s using l2 as a value of penalty.

solver : ‘lbfgs’

An algorithm used in optimization problem. The name stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno.

2) Random Forest Classifier

n_estimators : 200

It is the value of number of trees to be used in the forest.

max_features : ‘auto'

It is the number of features to be considered when classifiers finds the best split. In case of value ‘auto’, it will take max_features=sqrt(n_features).

class_weight : 'balanced'

It uses the values of y to adjust the weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)) automatically.

random_state : 42

It controls the randomness of the bootstrapping of the samples and the sampling of the features.