Assignment A3.1

Analyzing a sharing economy dataset

By Matheus Paro

The Dataset

The dataset is composed of millions of entries of money borrowed and their statuses. Along with these entries, you have the amount of money lent, the interest and other relevant information regarding this transaction.

For this Assignment I will attempt to find the key fields that determines if the loan will become Defaulted and see how well can we predict this happening.

Field Name	Table	Remote Field Name
# Loan Number	Master_Loan_Summary.csv	loan_number
# Amount Borrowed	Master_Loan_Summary.csv	amount_borrowed
# Term	Master_Loan_Summary.csv	term
# Borrower Rate	Master_Loan_Summary.csv	borrower_rate
# Installment	Master_Loan_Summary.csv	installment
Abc Grade	Master_Loan_Summary.csv	grade
Origination Date	Master_Loan_Summary.csv	origination_date
Abc Listing Title	Master_Loan_Summary.csv	listing_title
# Principal Balance	Master_Loan_Summary.csv	principal_balance
# Principal Paid	Master_Loan_Summary.csv	principal_paid
# Interest Paid	Master_Loan_Summary.csv	interest_paid
# Late Fees Paid	Master_Loan_Summary.csv	late_fees_paid
# Debt Sale Proceeds Received	Master_Loan_Summary.csv	debt_sale_proceeds_received
Last Payment Date	Master_Loan_Summary.csv	last_payment_date
Next Payment Due Date	Master_Loan_Summary.csv	next_payment_due_date
# Days Past Due	Master_Loan_Summary.csv	days_past_due
Abc Loan Status Description	Master_Loan_Summary.csv	loan_status_description
Abc Data Source	Master_Loan_Summary.csv	data_source

Main Variables

If we take a look at the dataset we can determine that some are more important. For instance, the amount of money lent and the interest are the main pillars of the loan compared to the last date that a payment was installed.

For the simplicity of this assignment, we will analyse these loans with 3 main variables in mind, assuming these are the most important ones:

Amount Borrowed - Borrower Rate - Days Past Due

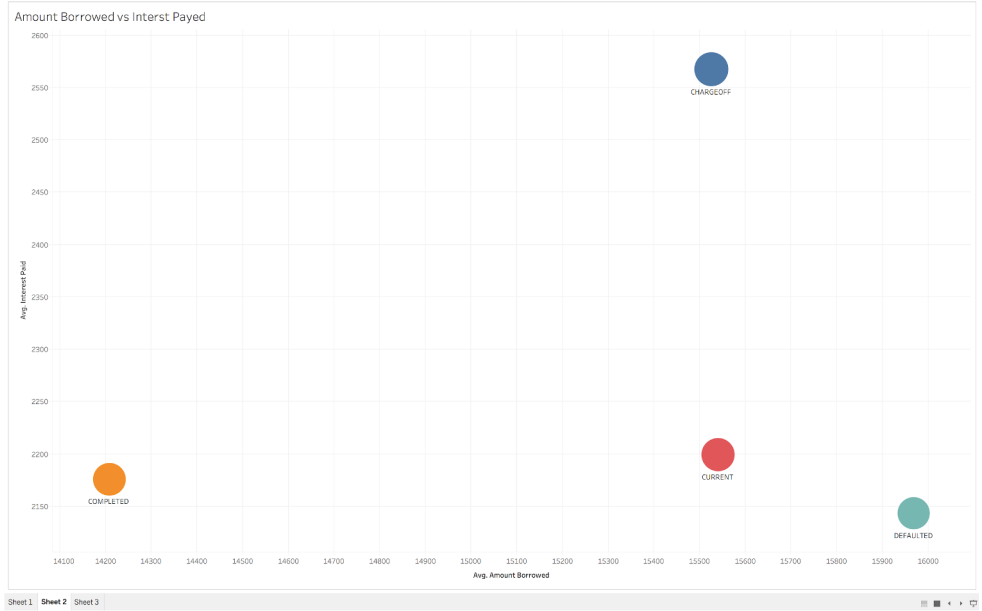
And our main dimension will be Loan Status Description, that will help us Identify how does these different inputs behave, so that then we can later predict the outcomes.

Here we can Visualize how the status of the loan changes depending on the amount they borrowed and the rate at which they borrowed it at. We can observe from this the defaulted ones are the ones that takes the biggest amount with the higher rates. (around 16000 and 0.176)

Completed ones generally stay at a lower rate, at 0.133 and a net amount of 14200, which is about 2 thousand less.

On this graph we can see that depending on how much interest paid, compared to the amount borrowed is also a strong determiner if someone will default or not.

Normally the ones that do default pay the less compared to the rest. Next let's analyse the time that they haven't been paying since that influences directly with why they are paying so less in interest.



This graph shows us the amount of days that the different groups of lending entries have past due in their payments. We can observe that those that been classified charge off, are those that haven't played in an average of 700 days.

Those that defaulted were only behind on their payment by 60 days. Current is around 40, but must be ensured that doesn’t slip into becoming at the level of the Defaulted posing a risk to the financial institution.

Predicting

In order to try to predict what the outcome for new loan entries would be, we will use python to manipulate our dataset and train a model. For this we will be using sklearn, pandas and numpy. In this assignment we will use the Random Tree Algorithm to try to predict fraud or not.

All the code can be found at the following link:

https://colab.research.google.com/drive/1QmAcs6mX2I-0uWSEQgSnNvEiTaSa0eoX?usp=sharing

Manipulating Dataset

```
[7] ds = pd.read_csv(DIR)
    #Taking out useless variables
    ds = ds.drop(["loan_number", "term", "grade", "origination_date", "listing_title", "last_payment_date", "next_payment_due_date", "data_source"], axis = 1)
    ds.head()
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (7) have mixed types.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

	amount_borrowed	borrower_rate	installment	principal_balance	principal_paid	interest_paid	late_fees_paid	debt_sale_proceeds_received	days_past_due	loan_status_description
0	27050.0	0.1099	885.46	0.0	27050.0	4702.53	0.0	0.0	0	COMPLETED
1	4800.0	0.1099	157.13	0.0	4800.0	357.52	0.0	0.0	0	COMPLETED
2	12000.0	0.0762	373.94	0.0	12000.0	1397.54	0.0	0.0	0	COMPLETED
3	12000.0	0.1199	398.52	0.0	12000.0	2346.48	0.0	0.0	0	COMPLETED
4	12000.0	0.0662	368.45	0.0	12000.0	1263.95	0.0	0.0	0	COMPLETED

In order to train our model, we need to have only scalar values, as we will not being those and that information.

We simply do that by telling pandas to drop the columns that we want. This is the resulting dataset.

Manipulating Dataset

```
#Creating a dataset for every Grade
status = {"CHARGEOFF": 1,"COMPLETED": 0,"CURRENT": None, "DEFAULTED": 1}
datasets = {}
for des, value in status.items():
    mask = ds["loan_status_description"].isin([des])
    datasets[des] = ds.loc[mask]
    if des != "CURRENT":
        datasets[des]["result"] = value
    datasets[des] = datasets[des].drop(["loan_status_description"], axis =1)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  if __name__ == '__main__':
```

```
[9] print(datasets["DEFAULTED"].sample())
```

```
         amount_borrowed  borrower_rate  ...  days_past_due  result
2171058          15000.0         0.1095  ...              0       1

[1 rows x 10 columns]
```

As we do not need the current loans for training since they have no result yet, we will divide all different statuses and add their respective values.

In this case we give the value for 1 if the person defaulted or got charged off, and 0 if they completed the loan payment. The result is a dictionary with each dataset for the different categories. Now we will join them.

That is done with the following code:

```
all_dataset = pd.concat([datasets["CHARGEOFF"], datasets["DEFAULTED"], datasets["COMPLETED"]])
```

Splitting Data

Now all we need to do is divide the training and test dataset so that we can train our model.

We will also Scale our values as it improves with biases towards certain variables.

Now we have our arrays ready to be fit in the model.

```
[10] #splitting inputs and outputs
     all_dataset = pd.concat([datasets["CHARGEOFF"], datasets["DEFAULTED"], datasets["COMPLETED"]])
     X = all_dataset.iloc[:, :-1].values
     y = all_dataset.iloc[:, -1].values

[11] #splitting dataset
     from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

[12] #scaling our values
     from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)

     X_train

     array([[-0.96120624,  1.5637401 , -0.79738562, ..., -0.13931302,
             -0.20223961, -0.35802068],
            [ 1.07447544, -0.47022071,  1.30343548, ...,  3.28923691,
             -0.20223961,  0.53836444],
            [-0.16723069, -1.13831003, -0.14073774, ..., -0.13931302,
             -0.20223961, -0.35802068],
            ...,
            [ 0.11931688,  0.2943704 ,  0.37595059, ...,  1.04603217,
              2.25416437,  2.74307636],
            [-0.99105494,  0.03826949, -1.17542443, ..., -0.13931302,
             -0.20223961, -0.35802068],
            [-0.47765722, -0.80240957, -0.42999585, ..., -0.13931302,
             -0.20223961, -0.35802068]])
```

Splitting Data

Now all we need to do is divide the training and test dataset so that we can train our model.

We will also Scale our values as it improves with biases towards certain variables.

Now we have our arrays ready to be fit in the model.

```
[10] #splitting inputs and outputs
     all_dataset = pd.concat([datasets["CHARGEOFF"], datasets["DEFAULTED"], datasets["COMPLETED"]])
     X = all_dataset.iloc[:, :-1].values
     y = all_dataset.iloc[:, -1].values

[11] #splitting dataset
     from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

[12] #scaling our values
     from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)

     X_train

     array([[-0.96120624,  1.5637401 , -0.79738562, ..., -0.13931302,
             -0.20223961, -0.35802068],
            [ 1.07447544, -0.47022071,  1.30343548, ...,  3.28923691,
             -0.20223961,  0.53836444],
            [-0.16723069, -1.13831003, -0.14073774, ..., -0.13931302,
             -0.20223961, -0.35802068],
            ...,
            [ 0.11931688,  0.2943704 ,  0.37595059, ...,  1.04603217,
              2.25416437,  2.74307636],
            [-0.99105494,  0.03826949, -1.17542443, ..., -0.13931302,
             -0.20223961, -0.35802068],
            [-0.47765722, -0.80240957, -0.42999585, ..., -0.13931302,
             -0.20223961, -0.35802068]])
```

Training and Accuracy

By importing the model library and fitting our data to the model I have successfully trained the Tree Classifier.

If I compare the predicted results to the actual results we can know what the accuracy was for our model, and even a confusion matrix.

The accuracy for this model is an astounding 99.8%.

The confusion matrix is very well divided in terms of the false negatives and false positives. In this scenario we value more the False Negatives, where the people that actually don't pay weren't detected, and that is the 312 cases.

```
[14] #Training Model
     from sklearn.tree import DecisionTreeClassifier
     classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
     classifier.fit(X_train, y_train)

     DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                            max_depth=None, max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort='deprecated',
                            random_state=0, splitter='best')

[15] #Making a Confusion matrix with the validation set
     y_pred = classifier.predict(X_test)
     from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)

     [[293370    312]
      [   298  78864]]
     0.998639270043235
```

Visualizing cases

Now the final part of the code is for seeing what the model is classifying and looking at the details for that case.

We will feed in cases from the “Current” loans, so that we have no way of knowing if it will complete or default.

Visualizing cases - Default

```
from termcolor import colored
#predicting one sample of the current loans
sample = datasets["CURRENT"].sample()

print("amount_borrowed: " , int(sample['amount_borrowed']))
print("borrower_rate: " , float(sample['borrower_rate']))
print("installment: " , int(sample['installment']))
print("principal_balance: " , int(sample['principal_balance']))
print("principal_paid: " , int(sample['principal_paid']))
print("interest_paid: " , int(sample['interest_paid']))
print("late_fees_paid: " , float(sample['late_fees_paid']))
print("debt_sale_proceeds_received: " ,float( sample['debt_sale_proceeds_received']))
print("days_past_due: " , int(sample['days_past_due']))
print("\n")
prediction = classifier.predict(sc.transform(sample))
if prediction[0] == 1:
    print(colored("Prediction: More likely to Default", 'red'))
else:
    print(colored("Prediction: More likely to Pay their Debts", 'green'))
```

```
amount_borrowed:  2400
borrower_rate:  0.1602
installment:  84
principal_balance:  1960
principal_paid:  439
interest_paid:  234
late_fees_paid:  0.0
debt_sale_proceeds_received:  0.0
days_past_due:  60


Prediction: More likely to Default
```

In this case, the model judged the Lendee to be more likely to default. So more action should be taken to prevent this.

Visualizing cases - Complete

```
from termcolor import colored
#predicting one sample of the current loans
sample = datasets["CURRENT"].sample()

print("amount_borrowed: " , int(sample['amount_borrowed']))
print("borrower_rate: " , float(sample['borrower_rate']))
print("installment: " , int(sample['installment']))
print("principal_balance: " , int(sample['principal_balance']))
print("principal_paid: " , int(sample['principal_paid']))
print("interest_paid: " , int(sample['interest_paid']))
print("late_fees_paid: " , float(sample['late_fees_paid']))
print("debt_sale_proceeds_received: " ,float( sample['debt_sale_proceeds_received']))
print("days_past_due: " , int(sample['days_past_due']))
print("\n")
prediction = classifier.predict(sc.transform(sample))
if prediction[0] == 1:
    print(colored("Prediction: More likely to Default", 'red'))
else:
    print(colored("Prediction: More likely to Pay their Debts", 'green'))
```

```
amount_borrowed:  20000
borrower_rate:  0.0959
installment:  641
principal_balance:  16035
principal_paid:  3964
interest_paid:  1167
late_fees_paid:  0.0
debt_sale_proceeds_received:  0.0
days_past_due:  0


Prediction: More likely to Pay their Debts
```

In this case, the model judged the Lendee to be more likely to pay back their debts.

In my opinion the Borrower rate is relatively low and compensates for the amount that was borrowed, and therefore making it a safer bet that the person will pay back.