Should This Loan be Approved or Denied?
— Loan Default Prediction

Group Members:

Jianing Hu
Shiqi Tang
Shuyang Ning
Yuchen Wu

GitHub Repository:
https://github.com/JoanneT17/5293_group_project_loan

CONTENTS

INTRODUCTION

Source

U.S. Small Business Administration (SBA)

Background

The U.S. SBA was founded in 1953 on the principle of promoting and assisting small enterprises in the U.S. credit market.

There have been many success stories of start-ups receiving SBA loan guarantees such as FedEx and Apple Computer. However, there have also been stories of small businesses and/or start-ups that have defaulted on their SBA-guaranteed loans.

Purpose

Help loan officer make decisions about whether to approve a loan to a small business.

89.9w rows   27 columns

Selected Columns

NAICS	North American industry classification system code
ApprovalFY	Fiscal year of commitment
Term	Loan term in month
NewExist	1=existing, 2=new
FranchiseCode	00000 or 00001= no franchise

```
# drop duplication
df.drop_duplicates(subset=None,keep='first',inplace=True)
```

```
# select columns
df = df[['NAICS', 'ApprovalFY', 'Term', 'NewExist',
         'FranchiseCode', 'UrbanRural',
         'RevLineCr', 'MIS_Status', 'GrAppv']]
# drop na (relatively NOT large data loss)
df.dropna(inplace=True)
```

```
# keep first 2 digits of NAICS
df.NAICS = pd.to_numeric(df.NAICS.astype(str).str[:2])
# New Exist = 0, 1 (Delet NewExist = 0.0)
df.NewExist = df.NewExist.astype(int)
df = df[(df.NewExist == 1) | (df.NewExist == 2)]
df.NewExist[df.NewExist == 1] = 0
df.NewExist[df.NewExist == 2] = 1
# Franchise Code = 0, 1
df.FranchiseCode[df.FranchiseCode <= 1] = 0
df.FranchiseCode[df.FranchiseCode > 1] = 1
df = df.rename(columns={"FranchiseCode":"HasFranchise"})
```

89.9w rows 27 columns

Selected Columns

UrbanRural	1=urban 2=rural 0=undefined
RevLineCr	Revolving line of credit Y=yes, N=no
GrAppv	Gross amount of loan approved
MIS_Status	Loan status CHGOFF=default, PIF = full paid

```
# RevLineCr = 0, 1
df.RevLineCr.replace(['N','0','Y','T'],[0,0,1,1],
                     inplace=True)
df = df[(df.RevLineCr == 0) | (df.RevLineCr == 1)]
df.RevLineCr = pd.to numeric(df.RevLineCr)
```

```
# $, A, ...
df.GrAppv = df.GrAppv.apply(lambda x: x.strip('$'))
df.GrAppv = df.GrAppv.apply(lambda x : x.replace(',',''))
df.GrAppv = pd.to_numeric(df.GrAppv)
df.ApprovalFY[df.ApprovalFY == "1976A"] =
    df.ApprovalFY[df.ApprovalFY == "1976A"].apply(lambda x: x.strip('A'))
# Change type
df = df.astype({"GrAppv":'int', "ApprovalFY":'int'})
```

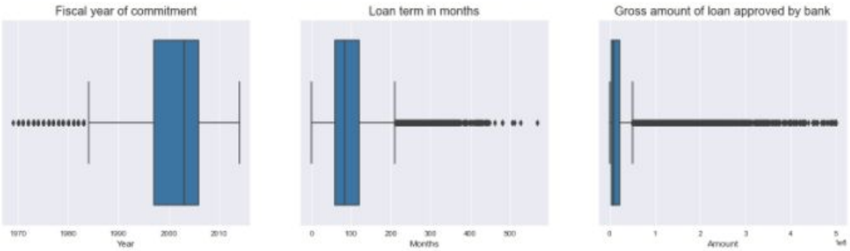
```
# Default
df.MIS_Status.replace(['P I F', 'CHGOFF'],[0, 1],
                      inplace=True)
df = df.rename(columns={"MIS_Status":"Default"})
df.Default = pd.to_numeric(df.Default)
```

EXPLORATORY DATA ANALYSIS

	ApprovalFY	Term	GrAppv
count	891424.000000	891424.000000	8.914240e+05
mean	2001.163105	110.712274	1.927831e+05
std	5.908215	78.863264	2.828811e+05
min	1969.000000	0.000000	1.000000e+03
25%	1997.000000	60.000000	3.500000e+04
50%	2003.000000	84.000000	9.000000e+04
75%	2006.000000	120.000000	2.250000e+05
max	2014.000000	569.000000	5.000000e+06

1. Checking the outliner for numerical variables

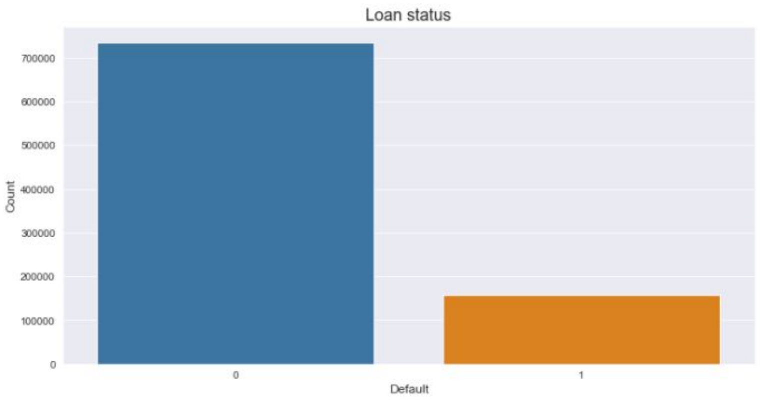
   All these numerical variables have outliers.



2. Analysis On Target Value

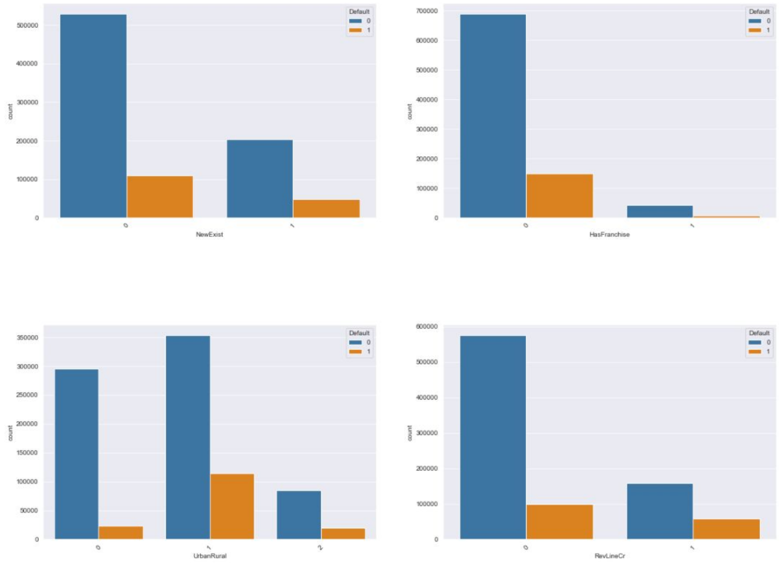
a. Imbalanced

Imbalance Ratio: 4.66



b. Univariate Analysis

i. Categorical variables

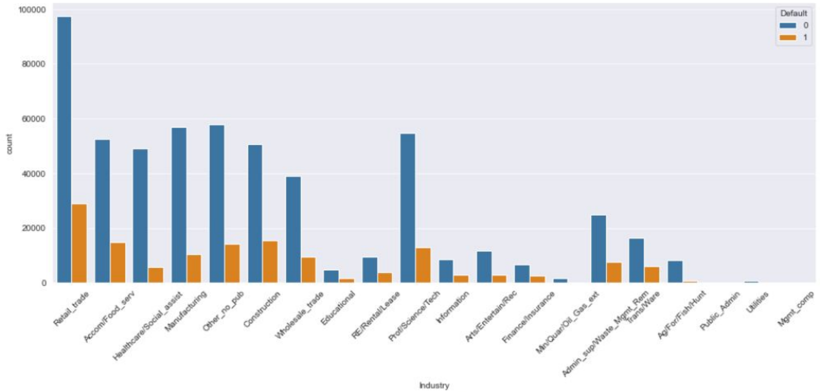


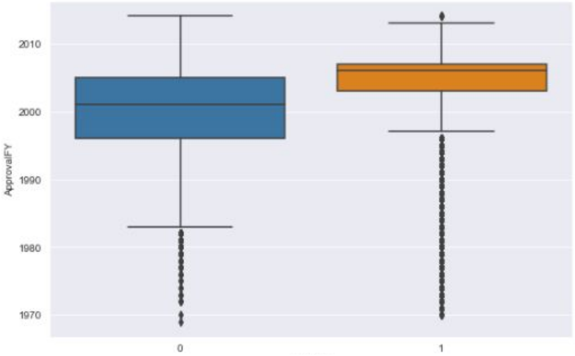
2. Analysis On Target Values

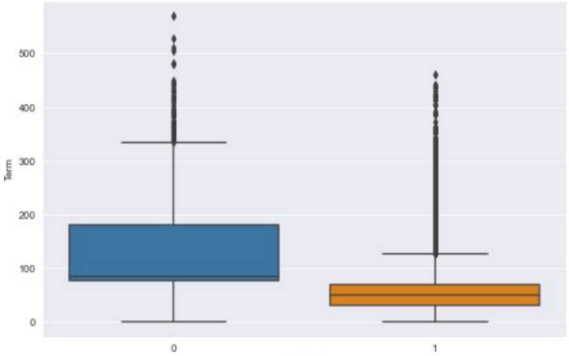
b. Univariate Analysis

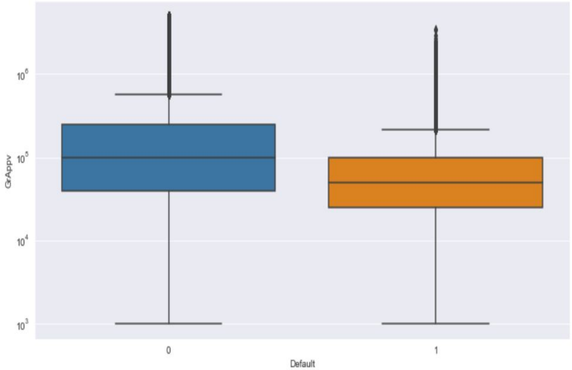
i. Categorical variables

ii. Numerical Variables









3. Correlation

The high correlation between the left variables may affects the performance of our model created by the algorithms, like logistic regression and KNN.

Correlated variables: ‘Paid in full’ dataframe (Coef > = 0.3)

	Var1	Var2	Correlation
46	UrbanRural	ApprovalFY	0.75
74	GrAppv	Term	0.49
9	ApprovalFY	NAICS	0.48
45	UrbanRural	NAICS	0.43
55	RevLineCr	ApprovalFY	0.39
56	RevLineCr	Term	0.34

Correlated variables: ‘Default’ dataframe (Coef > = 0.3)

	Var1	Var2	Correlation
46	UrbanRural	ApprovalFY	0.61
9	ApprovalFY	NAICS	0.52
74	GrAppv	Term	0.50
45	UrbanRural	NAICS	0.36
56	RevLineCr	Term	0.30

BUILD MODEL

General Algorithm

Determine X and Y from selected columns

Split data into train and test sets

Build initial model

Improve model using grid search

Check accuracy of best model in test data

	precision	recall	f1-score	support
0	0.853	0.976	0.910	183357
1	0.661	0.219	0.330	39499
accuracy			0.842	222856
macro avg	0.757	0.598	0.620	222856
weighted avg	0.819	0.842	0.807	222856

Logistic Regression

X using all columns except 'default', all object in dummy forms

Y 'default'

```
# Scale the feature values prior to modeling
scale = StandardScaler()
X_scaled = scale.fit_transform(X)

X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.25)
```

Initialize model

LogisticRegression(random_state).fit(training data)

get accuracy on testing data

```
# Improve model using grid search
from sklearn.model_selection import GridSearchCV

grid={"C":np.logspace(-3,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
logreg_cv=GridSearchCV(log_reg,grid,cv=10)
logreg_cv.fit(X_train, y_train)

print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

Best model : c = 10, penalty = ridge

BUILD MODEL

General Algorithm

Determine X and Y from selected columns

Split data into train and test sets

Build initial model

Improve model using grid search

Check accuracy of best model in test data

```
              precision    recall  f1-score   support

           0      0.937     0.965     0.951    183514
           1      0.812     0.697     0.750     39342

    accuracy                          0.918    222856
   macro avg      0.875     0.831     0.851    222856
weighted avg      0.915     0.918     0.916    222856
```

KNN

X using all columns except ‘default’, ‘NAICS’, ‘Industry’

Y ‘default’

```
# split Train Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y, random_state=520)
```

Initialize model

KNeighborsClassifier().fit(training data)

get accuracy on testing data

```
params = {
    'n_neighbors': [1,2,3,4,5],
    'weights': ['uniform', 'distance']
}
grid_search = GridSearchCV(neighbors.KNeighborsClassifier(),
                          param_grid=params,
                          refit=True,
                          cv=5, n_jobs=-1, verbose=1, scoring = "balanced_accuracy").fit(X_train,y_train)

print(f'bt best hyperparams      : {grid_search.best_params_}')
print(f'bt best mean cv accuracy : {grid_search.best_score_:.5f}')
```

Best model : n_neighbors = 3, weight = ‘uniform’

BUILD MODEL

General Algorithm

Determine X and Y from selected columns

Split data into train and test sets

Build initial model

Improve model using grid search

Check accuracy of best model in test data

	precision	recall	f1-score	support
0	0.963	0.968	0.966	183215
1	0.849	0.828	0.838	39641
accuracy			0.943	222856
macro avg	0.906	0.898	0.902	222856
weighted avg	0.943	0.943	0.943	222856

Decision Tree

X using all columns except 'default', don't use object data(Industry is substituted by NAICS)

Y 'default'

```
# Split Train Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=520)
```

Initialize model

DecisionTreeClassifier().fit(training data)

get accuracy on testing data

```
# Improve Model
from sklearn.model_selection import GridSearchCV
params = {
    'max_depth': [5, 10, 20, 50, 100, 150],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"]
}
grid_search = GridSearchCV(DecisionTreeClassifier(),
                           param_grid=params,
                           refit=True,
                           cv=5, n_jobs=-1, verbose=1, scoring = "balanced_accuracy").fit(X_train,y_train)
print(f'bt best hyperparams: {grid_search.best_params_}')
print(f'bt best mean cv accuracy: {grid_search.best_score_:.5f}')
```

Best model : max_depth = 20 min_samples_leaf = 50

criterion = 'entropy'

BUILD MODEL

General Algorithm

Determine X and Y from selected columns

Split data into train and test sets

Build initial model

Improve model using grid search

Check accuracy of best model in test data

	precision	recall	f1-score	support
0	0.956	0.971	0.963	183215
1	0.856	0.792	0.823	39641
accuracy			0.939	222856
macro avg	0.906	0.882	0.893	222856
weighted avg	0.938	0.939	0.938	222856

Random Forest

X using all columns except 'default', don't use object data(Industry is substituted by NAICS)

Y 'default'

```
# Split Train Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=520)
```

Initialize model

RandomForestClassifier().fit(training data)

get accuracy on testing data

```
# use cross validation to improve
from sklearn.model_selection import GridSearchCV
params = {
    'n_estimators': [50, 100, 120],
    'max_depth': [15, 20, 25],
    'criterion': ["gini", "entropy"]
}
grid_search = GridSearchCV(RandomForestClassifier(),
                           param_grid=params,
                           refit=True,
                           cv=5, n_jobs=-1, verbose=1, scoring = "balanced_accuracy").fit(X_train,y_train)

print(f'bt best hyperparams      : {grid_search.best_params_}')
print(f'bt best mean cv accuracy : {grid_search.best_score_:.5f}')
```

Best model : max_depth = 20 n_etimators = 120

criterion = 'entropy'

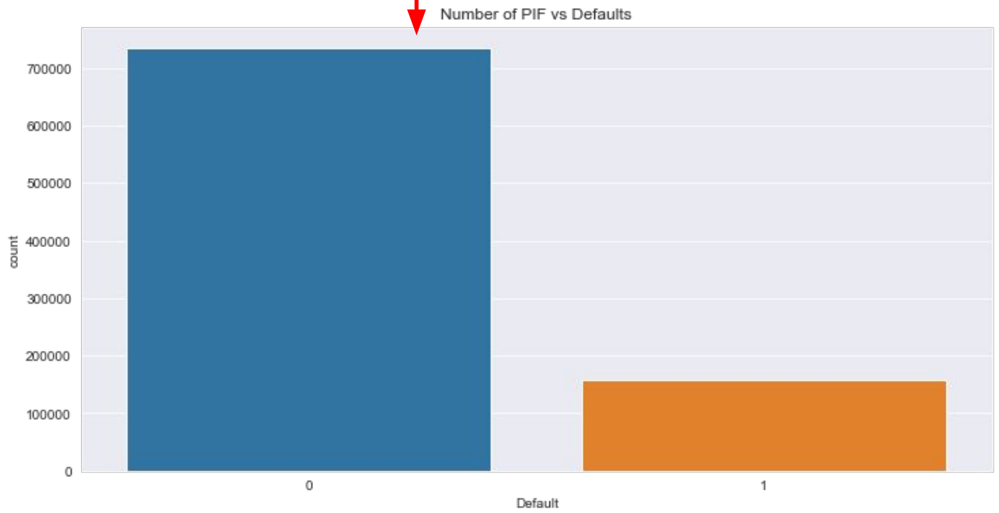
EVALUATION & MODEL SELECTION

- By comparison, we conclude that **Decision Tree and Random Forest perform better** in this case.
- As the dataset is slightly **unbalanced**, we will look at **balanced accuracy** when comparing decision tree model and random forest model.
- Balanced accuracy is calculated as:

(Sensitivity + Specificity) / 2

Where sensitivity is the true positive rate, and specificity is the true negative rate.

	Accuracy	Training Time
Logistic Regression	84.2%	4.16s
KNN	91.8%	135.67s
Decision Tree	94.3%	17.31s
Random Forest	93.9%	155.74s



Decision Tree VS. Random Forest

Decision Tree:

```
print(classification_report(y_test, y_pred_imp, digits=3))
```

```
              precision    recall  f1-score   support

           0      0.963     0.968     0.966    183215
           1      0.849     0.828     0.838     39641

    accuracy                          0.943    222856
   macro avg      0.906     0.898     0.902    222856
weighted avg      0.943     0.943     0.943    222856
```

```
print("Balanced Accuracy: ",metrics.balanced_accuracy_score(y_test, y_pred_imp))
```

```
Balanced Accuracy:  0.8978866718146674
```

Random Forest:

```
print(classification_report(y_test, y_rfcimb_pred, digits=3))
```

```
              precision    recall  f1-score   support

           0      0.956     0.971     0.963    183215
           1      0.856     0.792     0.823     39641

    accuracy                          0.939    222856
   macro avg      0.906     0.882     0.893    222856
weighted avg      0.938     0.939     0.938    222856
```

```
print("Balanced Accuracy: ",metrics.balanced_accuracy_score(y_test, y_rfcimb_pred))
```

```
Balanced Accuracy:  0.8817420671543394
```

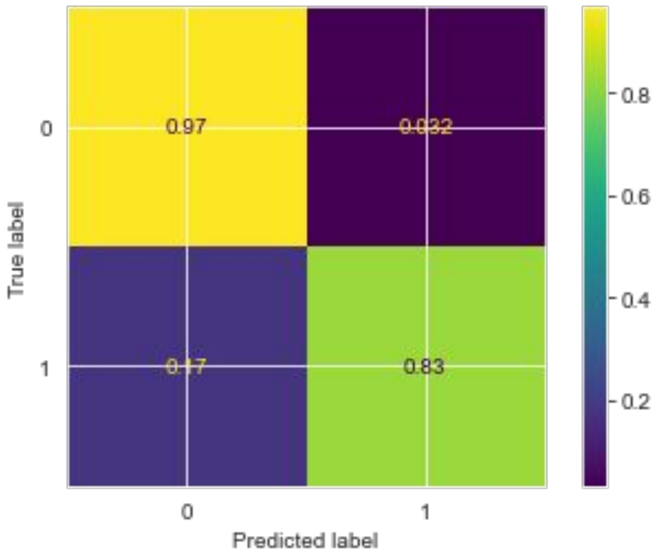
- Balanced accuracy for decision tree model: 89.8%.
- Balanced accuracy for random forest model: 88.2%.

Conclusion

- Accuracy wise: Decision Tree performs better than Random Forest both in accuracy score and balanced accuracy;
- Training cost wise: Random Forest has a higher training time than a single Decision Tree.
- Random Forest is suitable for situations when we have a large dataset, and interpretability is not a major concern.
- Decision trees are much easier to interpret and understand. Since a random forest combines multiple decision trees, it becomes more difficult to interpret.

→ **We recommend Decision Tree Model in this case when predicting whether the loan should be accepted or denied.**

Further Evaluation




```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_imp)
TN = cm[0][0]
FN = cm[1][0]
TP = cm[1][1]
FP = cm[0][1]

# we use the WACC matric from the article
# http://store.ectap.ro/articole/1421.pdf

WACC = 0.25*(TP/(TP+FN))+0.75*(TN/(TN+FP))
print('The WACC rank of our model is: '+str(WACC))
```

```
The WACC rank of our model is: 0.9330412536542431
```

IMPROVEMENT

1. Limitation

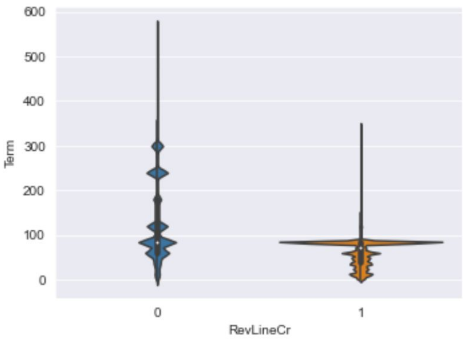
a. 'Loan Term == 84' occupied a large proportion of the data

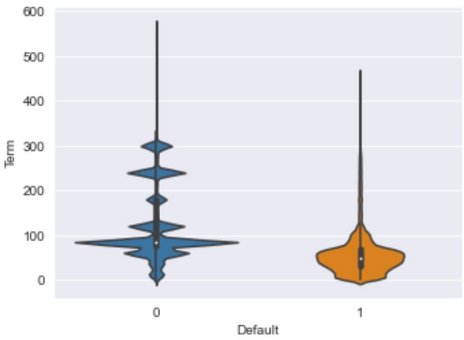
```
df.loc[df.RevLineCr == 1].Term.value_counts().head()
```

```
84    95709
60    15763
12     9908
36     8123
48     5740
Name: Term, dtype: int64
```

```
df.loc[df.Default == 0].Term.value_counts().head()
```

```
84     225747
60      86639
240     84705
120     75941
300     44354
Name: Term, dtype: int64
```



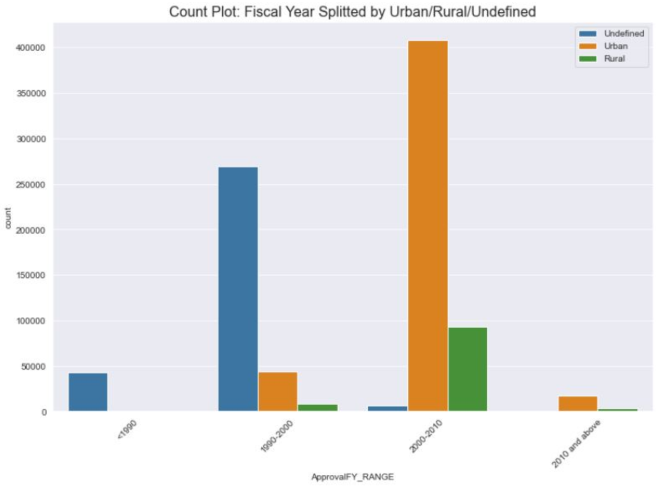


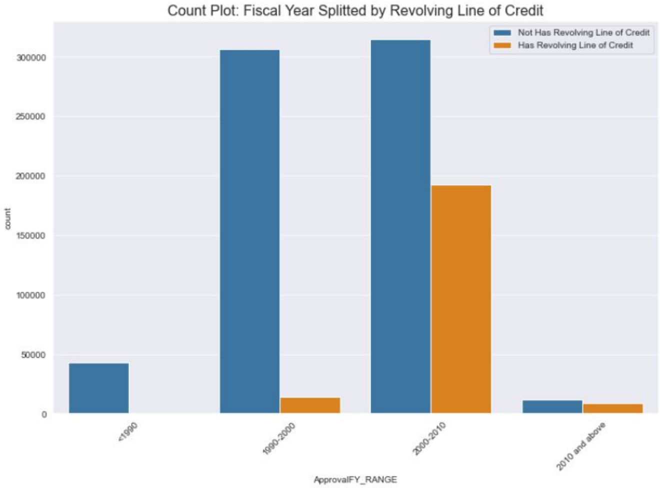
1. Limitation

b. The data is too out-to-date

i. Most of the places are undecided

ii. The usage of the revolving line of credit is becoming popular after millennium





c. More factors may influence a small enterprise default a business loan or not:

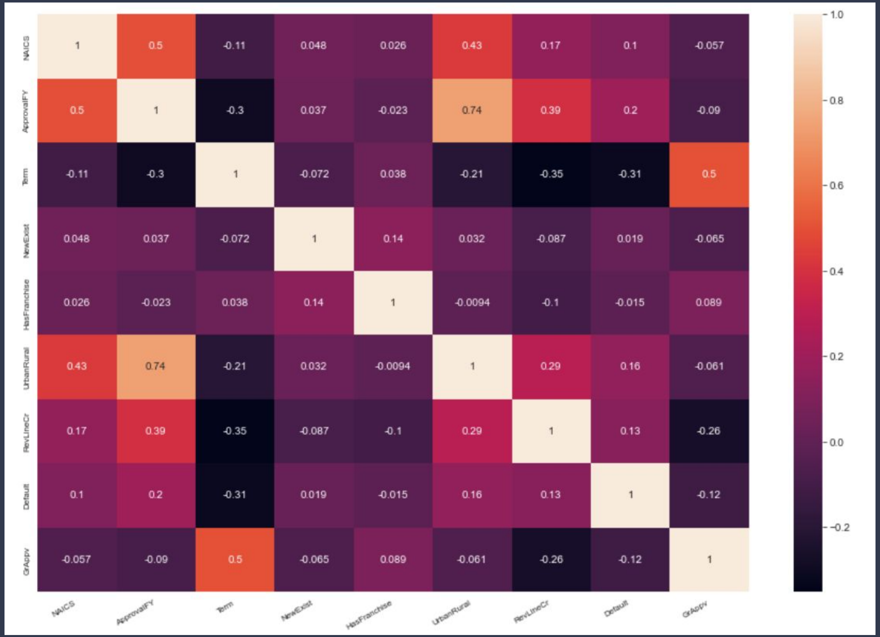
i. The credibility of the company's lender

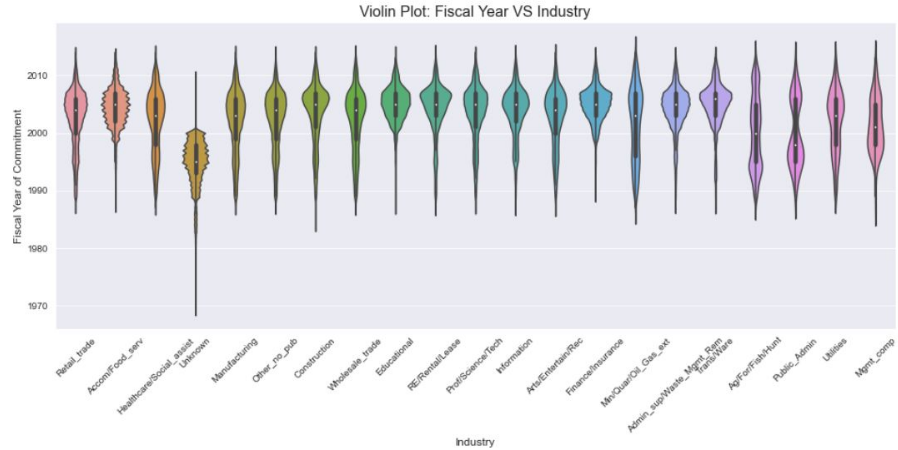
ii. The market situation of each industry for each year (This can be obtained by the GDP proportion of each industry occupied)

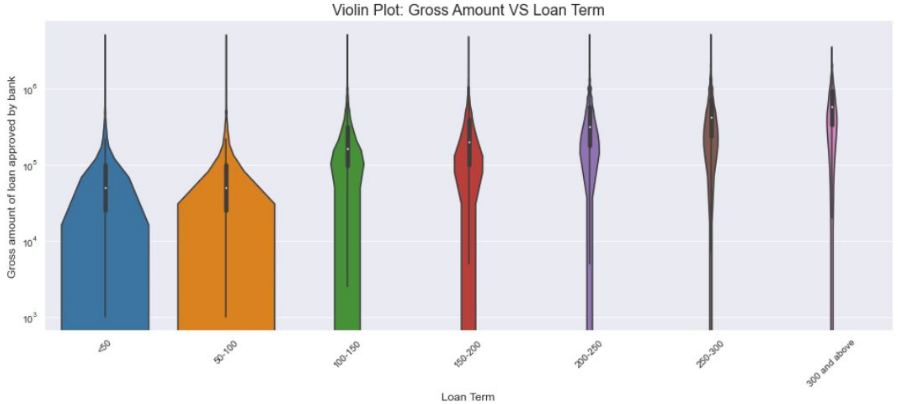
iii. …

IMPROVEMENT

2. Multicollinearity







3. Add auto-encoder model

We can also create the decoder layers of the auto-encoder neural network to further improve our model.