```
# import libraries
# data loading, manipulation and plotting
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# data split
from sklearn.model_selection import train_test_split

# label encoding
from sklearn.preprocessing import LabelEncoder

# ignore warnings
import warnings
warnings.filterwarnings('ignore')

# data scalling and standardization
from sklearn.preprocessing import StandardScaler

# model evaluation
from  sklearn.metrics  import  accuracy_score, precision_score, recall_score, confusion_matrix, classification_report
```

```
# Read data in python
df = pd.read_csv("loan_repayment_data.csv")
df.head(2)
```

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.ut
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	5:
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	7

```
# Consise Summery / dataset information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
```

```
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   object
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64
 11  delinq.2yrs        9578 non-null   int64
 12  pub.rec            9578 non-null   int64
 13  not.fully.paid     9578 non-null   int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

```
# Statistical Summary
df.describe()
```

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	re
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9.578000e+03	957
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.767197	1.691396e+04	4
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.930377	3.375619e+04	2
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.000000	3.187000e+03	2
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.958333	8.596000e+03	4
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.000000	1.824950e+04	7
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	11

Checking Missing values

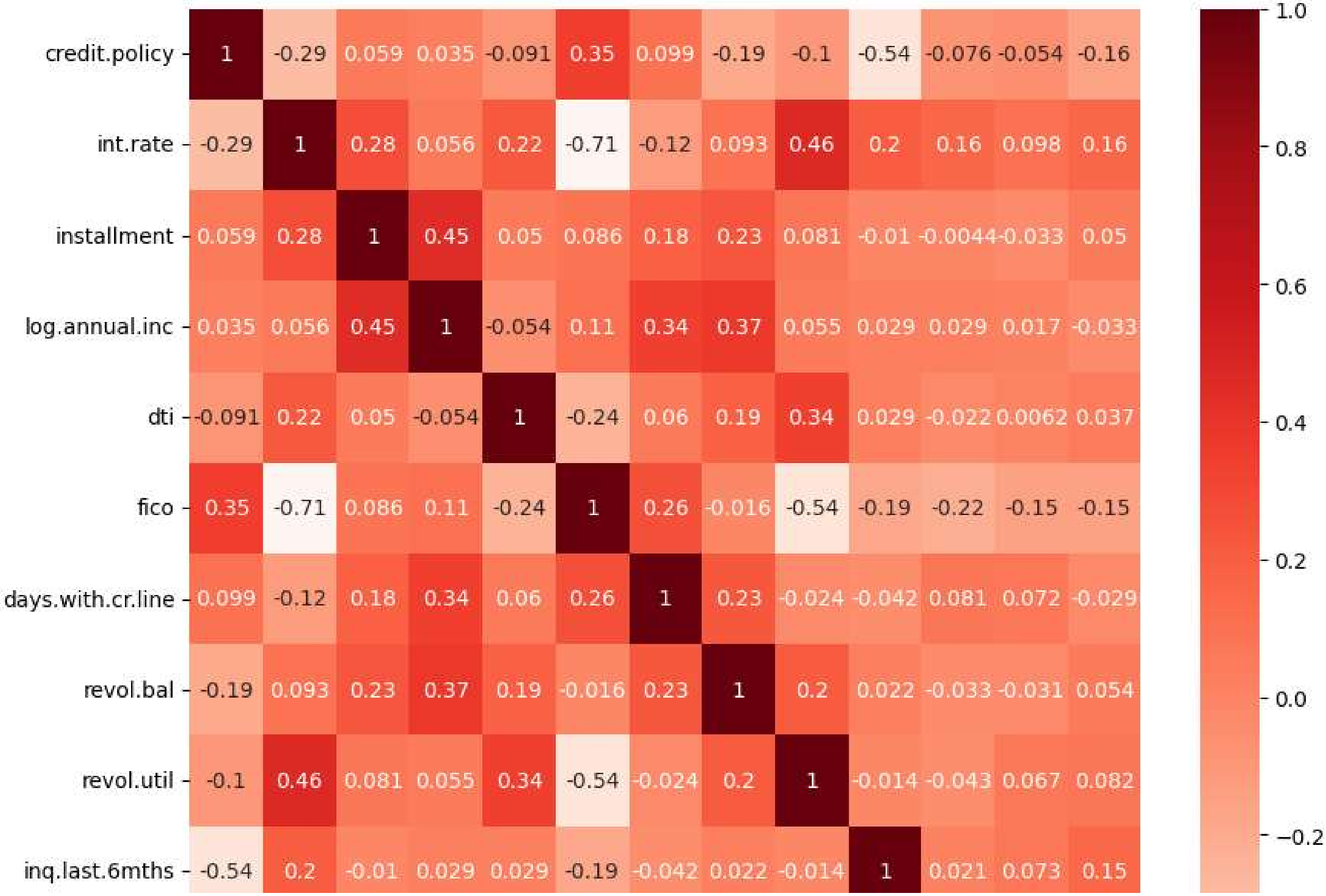
```
# Checking For Null Values
df.isnull().sum()#.sum()
```

```
credit.policy        0
purpose              0
int.rate             0
installment          0
log.annual.inc       0
dti                  0
fico                 0
days.with.cr.line    0
revol.bal            0
revol.util           0
inq.last.6mths       0
delinq.2yrs          0
pub.rec              0
not.fully.paid       0
dtype: int64
```

Our DataFrame contain Zero Null values.

Checking Correlation

```
plt.figure(figsize = (10, 10))
sns.heatmap(df.corr(), annot = True, cmap = "Reds")
plt.show()
```



We can see that init rate, credit policy, fico and inq.last.6mths has corresponding grater impact on target class(not.gully.paid)

Categorical independent variable

Now lets deal with categorical data, **Purpose** attribute/variable.

```
# unique values in purpose attribute
df.purpose.value_counts()
```

```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

It has 6 unique values. lets convert these labels into numeric form.

Encoding We will be using **Label Encoder** to convert labels available in purpose attribute.

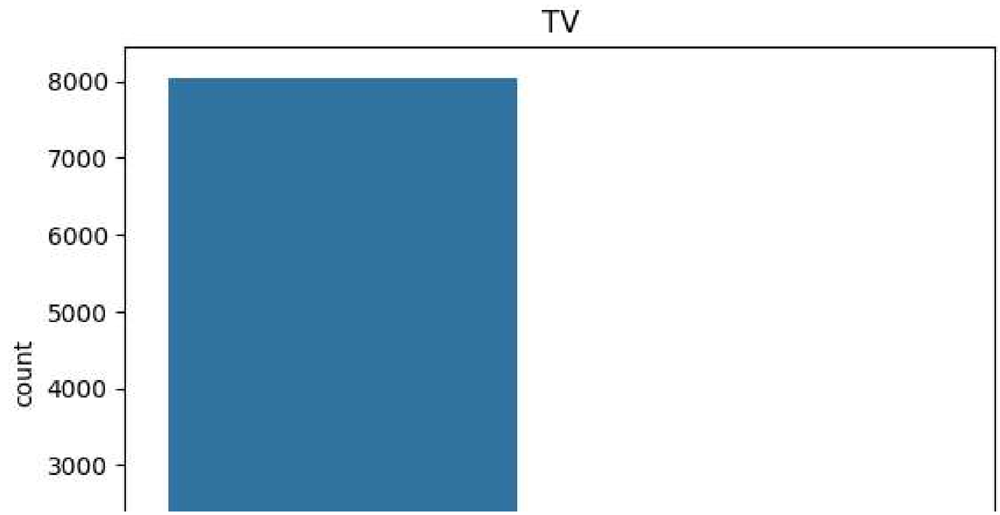
It will Encode purpose labels with value between 0 and n_classes-1(5).

```
df['purpose']=LabelEncoder().fit_transform(df['purpose'])
df.head(2)
```

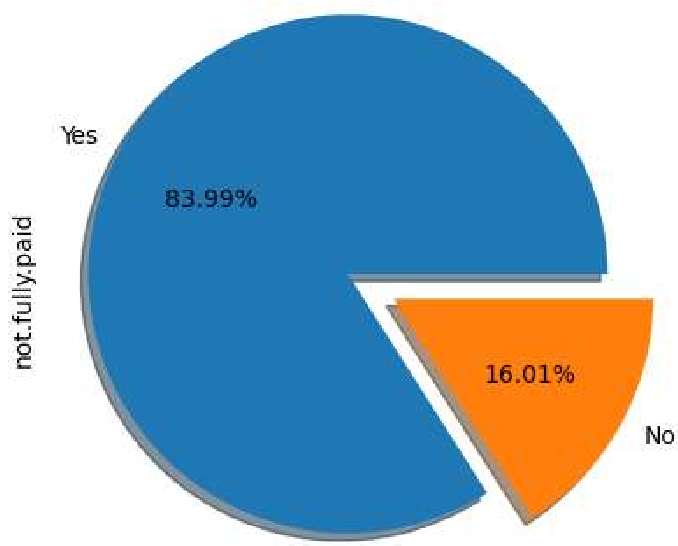
	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.
0	1	2	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	
1	1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	

Checking distribution of target variable

```
sns.countplot(data = df, x = 'not.fully.paid')
plt.title('TV')
plt.show()
```




```
labels = 'Yes', 'No'
ex = [0.1, 0.1]
df['not.fully.paid'].value_counts().plot.pie(labels = labels, autopct = '%1.2f%%', shadow = True, explode = ex)
plt.show()
```




```
# Checking duplicates
```

```
duplicates = df[df.duplicated()]
print("Duplicates: ", len(duplicates))
duplicates
```

```
Duplicates:  0
```

credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.las

In our data we don't have any duplicates.

first split

- training(70%) and testing(30%)

```
# Dropping target class
X = df.drop('not.fully.paid',axis=1)
y = df['not.fully.paid']
```

```
X_train1, X_test, y_train1, y_test = train_test_split(X, y , test_size=0.3,random_state=0, stratify=y)
```

```
print(X_train1.shape)
print(X_test.shape)
print(y_train1.shape)
print(y_test.shape)
```

```
(6704, 13)
(2874, 13)
(6704,)
(2874,)
```

Second split

- training(70%) and validation(30%)

```
X_train, X_val, y_train, y_val = train_test_split(X_train1, y_train1, test_size=0.3, random_state=0)
print(X_train.shape)
print(X_val.shape)
```

```
print(y_train.shape)
print(y_val.shape)
```

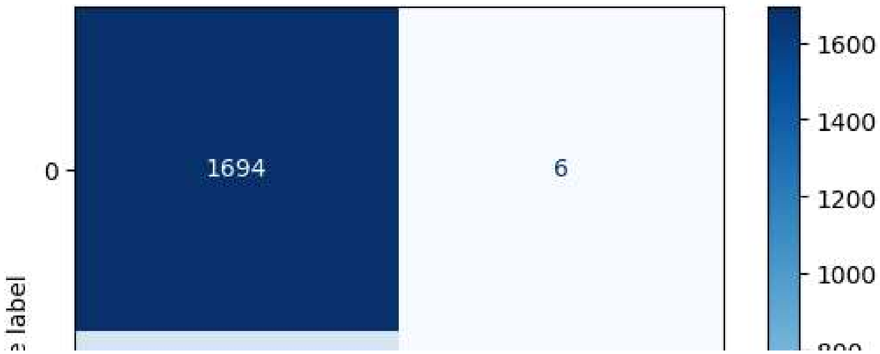
```
(4692, 13)
(2012, 13)
(4692,)
(2012,)
```

Applying ML Algorithms

```
# Logistic Regression Algorithm
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train) # fit a model
y_pred = classifier.predict(X_val) # predictions
print("Accuracy of Random Forest model is ", accuracy_score(y_val, y_pred) * 100, "%.")
```

```
Accuracy of Random Forest model is  84.44333996023857 %.
```

```
# Performance Evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_val, y_pred)

# Plotting Confusion Matrix
display = ConfusionMatrixDisplay(cm)
display.plot(cmap = "Blues")
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_val, y_pred))
```

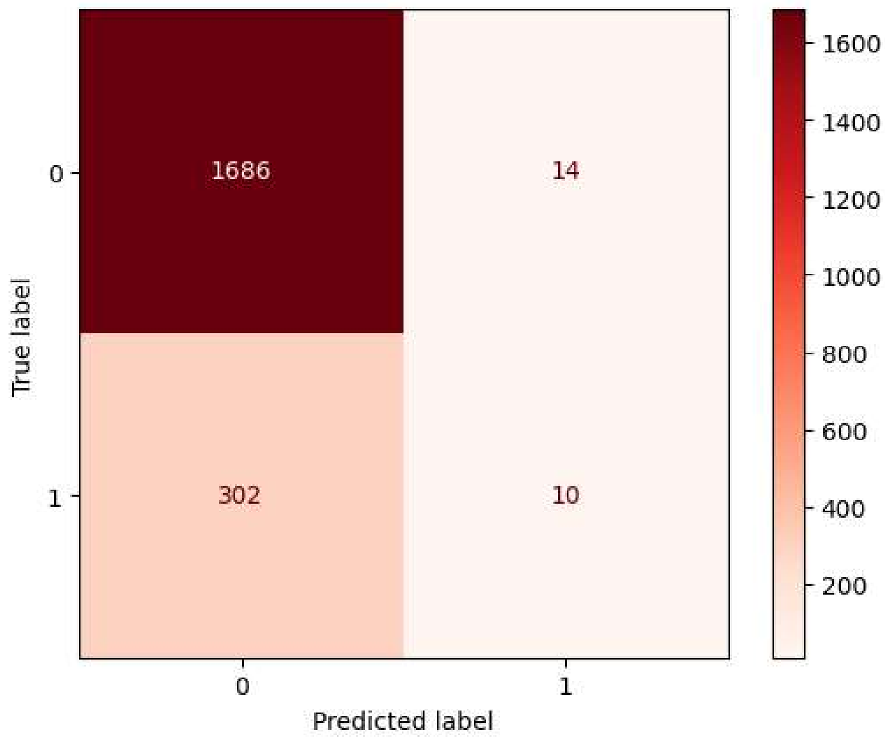
```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      1700
           1       0.45      0.02      0.03       312

    accuracy                           0.84      2012
   macro avg       0.65      0.51      0.47      2012
weighted avg       0.79      0.84      0.78      2012
```

```
# Random Forest Classifier Algorithm
from sklearn.ensemble import RandomForestClassifier

RFCmodel = RandomForestClassifier(n_estimators = 30, criterion = 'gini',
                                  min_samples_split = 10, random_state = 42, verbose = True)

RFCmodel.fit(X_train, y_train)
y_pred = RFCmodel.predict(X_val)
from sklearn import metrics
print("Accuracy of Random Forest model is ", metrics.accuracy_score(y_val, y_pred) * 100, "%.")
```

```
Accuracy of Random Forest model is  84.29423459244532 %.
```

```
# Performance Evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_val, y_pred)

# Plotting Confusion Matrix
display = ConfusionMatrixDisplay(cm)
```

```
display = ConfusionMatrixDisplay(cm)
display.plot(cmap = "Reds")
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.91      1700
           1       0.42      0.03      0.06       312

    accuracy                           0.84      2012
   macro avg       0.63      0.51      0.49      2012
weighted avg       0.78      0.84      0.78      2012
```

We Found that the Best Model for this DataSet is Random Forest with

1. Gini used

2. Recall is 84%

Accuracy of 78%.

as training and test set is .7 and .3.