In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


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

In [2]:
```python
data = pd.read_csv('cleaned_loans_full_schema.csv')
print(data.shape)
data.head(2).T
```

```
(10000, 50)
```

Out[2]:

	0	1
state	NJ	HI
homeownership	MORTGAGE	RENT
individual_annual_income	90000	40000
individual_income_verification	Verified	Not Verified
individual_debt_to_income	18.01	5.04
joint_annual_income	90000	40000
joint_income_verification	Not Verified	Not Verified
joint_debt_to_income	18.01	5.04
delinq_2y	0	0
earliest_credit_line	2001	1996
inquiries_last_12m	6	1
total_credit_lines	28	30
open_credit_lines	10	14
total_credit_limit	70795	28800
total_credit_utilized	38767	4321
num_collections_last_12m	0	0
num_historical_failed_to_pay	0	1
total_collection_amount_ever	1250	0
current_installment_accounts	2	0
accounts_opened_24m	5	11
months_since_last_credit_inquiry	5	8
num_satisfactory_accounts	10	14
num_active_debit_accounts	2	3
total_debit_limit	11100	16500
num_total_cc_accounts	14	24
num_open_cc_accounts	8	14
num_cc_carrying_balance	6	4
num_mort_accounts	1	0
account_never_delinq_percent	92.9	100
tax_liens	0	0
public_record_bankrupt	0	1
loan_purpose	moving	debt_consolidation
application_type	individual	individual
loan_amount	28000	5000
term	60_month	36_month
interest_rate	14.07	12.61
installment	652.53	167.54
sub_grade	C3	C1
issue_month	Mar-18	Feb-18
loan_status	Current	Current
initial_listing_status	whole	whole
disbursement_method	Cash	Cash
balance	27015.9	4651.37
paid_total	1999.33	499.12
paid_principal	984.14	348.63
paid_interest	1015.19	150.49
grade	C	C
default	0	0
emp_length	3	10
paidPrinciple_to_loanAmnt_ratio	0.04	0.07

In [3]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   state                             10000 non-null  object 
 1   homeownership                     10000 non-null  object 
 2   individual_annual_income          10000 non-null  float64
 3   individual_income_verification    10000 non-null  object 
 4   individual_debt_to_income         10000 non-null  float64
 5   joint_annual_income               10000 non-null  float64
 6   joint_income_verification         10000 non-null  object 
 7   joint_debt_to_income              10000 non-null  float64
 8   delinq_2y                         10000 non-null  int64  
 9   earliest_credit_line              10000 non-null  int64  
 10  inquiries_last_12m                10000 non-null  int64  
 11  total_credit_lines                10000 non-null  int64  
 12  open_credit_lines                 10000 non-null  int64  
 13  total_credit_limit                10000 non-null  float64
 14  total_credit_utilized             10000 non-null  float64
 15  num_collections_last_12m          10000 non-null  int64  
 16  num_historical_failed_to_pay      10000 non-null  int64  
 17  total_collection_amount_ever      10000 non-null  int64  
 18  current_installment_accounts      10000 non-null  int64  
 19  accounts_opened_24m               10000 non-null  int64  
 20  months_since_last_credit_inquiry  10000 non-null  float64
 21  num_satisfactory_accounts         10000 non-null  int64  
 22  num_active_debit_accounts         10000 non-null  int64  
 23  total_debit_limit                 10000 non-null  float64
 24  num_total_cc_accounts             10000 non-null  int64  
 25  num_open_cc_accounts              10000 non-null  int64  
 26  num_cc_carrying_balance           10000 non-null  int64  
 27  num_mort_accounts                 10000 non-null  int64  
 28  account_never_delinq_percent      10000 non-null  float64
 29  tax_liens                         10000 non-null  int64  
 30  public_record_bankrupt            10000 non-null  int64  
 31  loan_purpose                      10000 non-null  object 
 32  application_type                  10000 non-null  object 
 33  loan_amount                       10000 non-null  int64  
 34  term                              10000 non-null  object 
 35  interest_rate                     10000 non-null  float64
 36  installment                       10000 non-null  float64
 37  sub_grade                         10000 non-null  object 
 38  issue_month                       10000 non-null  object 
 39  loan_status                       10000 non-null  object 
 40  initial_listing_status            10000 non-null  object 
 41  disbursement_method               10000 non-null  object 
 42  balance                           10000 non-null  float64
 43  paid_total                        10000 non-null  float64
 44  paid_principal                    10000 non-null  float64
 45  paid_interest                     10000 non-null  float64
 46  grade                             10000 non-null  object 
 47  default                           10000 non-null  int64  
 48  emp_length                        10000 non-null  float64
 49  paidPrinciple_to_loanAmnt_ratio   10000 non-null  float64
dtypes: float64(17), int64(20), object(13)
memory usage: 3.8+ MB
```

In [4]:
```python
data_loan_status = data.drop(columns= ['loan_status', 'sub_grade','balance'], axis= 1)
```

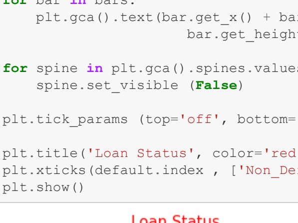
In [5]:
```python
default = round(100 * data_loan_status.default.value_counts(normalize= True, dropna= False), 2)

bars = plt.bar(default.index, default, align='center')
for bar in bars:
    plt.gca().text(bar.get_x() + bar.get_width()/2, 1.02 * bar.get_height(),
                   bar.get_height().round(2), ha= 'center', color= 'red', fontsize= 11)

for spine in plt.gca().spines.values():
    spine.set_visible (False)

plt.tick_params (top='off', bottom='off', left=False , right=False, labelleft=False, labelbottom='on')

plt.title('Loan Status', color='red', fontsize=15)
plt.xticks(default.index , ['Non_Default', 'Default'], fontsize=13)
plt.show()
```



In [6]:
```python
from itertools import combinations

def multicoll(df):
    lst= []
    comb = list(combinations(df.columns, 2))
    for tup in comb:
        corr_coef = abs(df.loc[tup[0] , tup[1]])
        if corr_coef > 0.70 :
            lst.append((tup[0] , tup[1], corr_coef))
    return lst
```

In [7]:
```python
X_corr = round(data_loan_status.drop(columns=['default'], axis= 0).corr(),2)
multicoll(X_corr)[:2]
```

Out[7]:
```
[('individual_annual_income', 'joint_annual_income', 0.85),
 ('individual_debt_to_income', 'joint_debt_to_income', 0.84)]
```

In [8]:
```python
multicoll_X = pd.DataFrame(multicoll(X_corr), columns= ['var1', 'var2','corr_coef'])
multicoll_X
```

Out[8]:

	var1	var2	corr_coef
0	individual_annual_income	joint_annual_income	0.85
1	individual_debt_to_income	joint_debt_to_income	0.84
2	total_credit_lines	open_credit_lines	0.76
3	total_credit_lines	num_satisfactory_accounts	0.76
4	total_credit_lines	num_total_cc_accounts	0.77
5	open_credit_lines	num_satisfactory_accounts	1.00
6	open_credit_lines	num_total_cc_accounts	0.71
7	open_credit_lines	num_open_cc_accounts	0.84
8	num_historical_failed_to_pay	tax_liens	0.87
9	num_satisfactory_accounts	num_total_cc_accounts	0.71
10	num_satisfactory_accounts	num_open_cc_accounts	0.84
11	num_active_debit_accounts	num_cc_carrying_balance	0.83
12	num_total_cc_accounts	num_open_cc_accounts	0.83
13	num_open_cc_accounts	num_cc_carrying_balance	0.80
14	loan_amount	installment	0.94
15	loan_amount	paid_interest	0.74
16	installment	paid_interest	0.71
17	paid_total	paid_principal	0.91
18	paid_principal	paidPrinciple_to_loanAmnt_ratio	0.74

Encoding

In [9]:
```python
categorical_features = data_loan_status.select_dtypes('object').columns.values
uniq_values = data_loan_status[categorical_features].nunique().sort_values(ascending= False)
uniq_values = pd.DataFrame(uniq_values).reset_index()
uniq_values.columns = ['categorical_feature', 'n_unique_values']
uniq_values
```

Out[9]:

	categorical_feature	n_unique_values
0	state	50
1	loan_purpose	12
2	grade	7
3	issue_month	3
4	joint_income_verification	3
5	individual_income_verification	3
6	homeownership	3
7	disbursement_method	2
8	initial_listing_status	2
9	term	2
10	application_type	2

One-Hot Encoding

In [10]:
```python
def One_hot_encoder(df, lst_cols):
    encoded_columns = pd.get_dummies(df[lst_cols])
    df = df.join(encoded_columns).drop(lst_cols, axis=1)

    return df
```

In [11]:
```python
lst_cols = ['issue_month','joint_income_verification','individual_income_verification'\
                  ,'homeownership','disbursement_method','initial_listing_status','term','application_
type', 'grade']

data_loan_status = One_hot_encoder(data_loan_status, lst_cols)
```

In [12]:
```python
data_loan_status.shape
```

Out[12]:
```
(10000, 65)
```

Split X, Y into train and test

In [13]:
```python
y1 = data_loan_status['default']
X1 = data_loan_status.drop(['default'], axis=1)
```

In [14]:
```python
from sklearn.model_selection import train_test_split
X1_train, X1_test, y1_train, y1_test= train_test_split(X1, y1, test_size=0.30, random_state= 42, strati
fy= y1)
```

In [15]:
```python
round(100 * y1_test.value_counts(normalize= True, dropna= False),2)
```

Out[15]:
```
0    99.93
1     0.07
Name: default, dtype: float64
```

In [16]:
```python
round(100 * y1_train.value_counts(normalize= True, dropna= False),2)
```

Out[16]:
```
0    99.93
1     0.07
Name: default, dtype: float64
```

Resampling

In [17]:
```python
df_majority = data_loan_status[data_loan_status.default == 0]  # 0 indicates non_default loans
df_minority = data_loan_status[data_loan_status.default == 1]  # 1 indicates default loans
```

In [18]:
```python
print('Default Loans Data: ', df_majority .shape)
print('Non_Default Loans Data: ',df_minority.shape)
```

```
Default Loans Data:  (9993, 65)
Non_Default Loans Data:  (7, 65)
```

In [19]:
```python
from sklearn.utils import resample

#Resample df of minority class
df_minority_resample = resample(df_minority, n_samples = 6500, replace = True, random_state= 42)
X1_train_oversampled = pd.concat([X1_train , df_minority_resample.drop('default',axis=1)])
y1_train_oversampled = pd.concat([y1_train, df_minority_resample['default']])
y1_train_oversampled.value_counts(normalize=True)
```

Out[19]:
```
0    0.518148
1    0.481852
Name: default, dtype: float64
```

In [20]:
```python
X1_train_oversampled.shape
```

Out[20]:
```
(13500, 64)
```

In [21]:
```python
y1_train_oversampled.shape
```

Out[21]:
```
(13500,)
```

In [22]:
```python
round(100 * y1_train_oversampled.value_counts(normalize= True, dropna= False),2)
```

Out[22]:
```
0    51.81
1    48.19
Name: default, dtype: float64
```

In [23]:
```python
round(100 * y1_test.value_counts(normalize= True, dropna= False),2)
```

Out[23]:
```
0    99.93
1     0.07
Name: default, dtype: float64
```

BinaryEncoder

In [24]:
```python
import category_encoders as ce

def binary_encoder(X, X_train, X_test,y_train, y_test, col):
    ce_bi = ce.BinaryEncoder(col).fit(X_train[col] , y_train)
    encoded_train = ce_bi.transform(X_train[col] , y_train)


    #X_test = ce_bi.transform(X_test[col] , y_test)
    encoded_test = ce_bi.transform(X_test[col] , y_test)


    return encoded_train, encoded_test
```

In [25]:
```python
encoded_train, encoded_test = binary_encoder(X1, X1_train_oversampled, X1_test, y1_train_oversampled, y
1_test\
                                              , ['state','loan_purpose'])
```

In [26]:
```python
X1_Train = pd.concat([X1_train_oversampled, encoded_train], axis=1, join='inner')
temp = X1_Train.drop(columns=['state','loan_purpose'], axis=1)
X1_Train = temp.copy()

y1_Train = y1_train_oversampled.copy()
```

In [27]:
```python
X1_Test = pd.concat([X1_test, encoded_test], axis=1, join='inner')
temp = X1_Test.drop(columns=['state','loan_purpose'], axis=1)
X1_Test = temp.copy()

y1_Test = y1_test.copy()
```

In [28]:
```python
print(X1_Test.shape)
print(y1_Test.shape)
```

```
(3000, 74)
(3000,)
```

In [29]:
```python
print(X1_Train.shape)
print(y1_Train.shape)
```

```
(13500, 74)
(13500,)
```

Scaling

When is the skewness too much?

- If the skewness is between -0.5 and 0.5, the data are fairly symmetrical
- If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data are moderately skewed
- If the skewness is less than -1 or greater than 1, the data are highly skewed

In [30]:
```python
data_skew = pd.DataFrame(round(data_loan_status.drop(columns=['default'],axis=1).skew()\
                               .sort_values(ascending= False),1)).reset_index()[:35]
data_skew.columns = ['feature' , 'skewness']
data_skew
```

Out[30]:

	feature	skewness
0	total_collection_amount_ever	74.6
1	tax_liens	68.2
2	num_historical_failed_to_pay	44.0
3	grade_G	28.8
4	grade_F	13.0
5	num_collections_last_12m	10.3
6	paid_total	5.9
7	grade_E	5.2
8	joint_income_verification_Verified	5.1
9	paidPrinciple_to_loanAmnt_ratio	4.3
10	joint_income_verification_Source Verified	4.1
11	paid_principal	3.9
12	disbursement_method_DirectPay	3.3
13	current_installment_accounts	2.9
14	delinq_2y	2.6
15	public_record_bankrupt	2.5
16	homeownership_OWN	2.1
17	grade_D	2.0
18	application_type_joint	2.0
19	individual_annual_income	1.8
20	total_credit_utilized	1.7
21	initial_listing_status_fractional	1.7
22	joint_annual_income	1.6
23	num_cc_carrying_balance	1.5
24	num_open_cc_accounts	1.4
25	total_credit_limit	1.4
26	num_total_cc_accounts	1.3
27	accounts_opened_24m	1.3
28	individual_income_verification_Verified	1.3
29	paid_interest	1.3
30	grade_A	1.2
31	inquiries_last_12m	1.2
32	num_mort_accounts	1.1
33	individual_debt_to_income	1.1
34	grade_C	1.1

'Box-Cox Power Transformation:

The statisticians George Box and David Cox developed a procedure to identify an appropriate exponent (Lambda = l) to use to transform data into a "normal shape." The Lambda value indicates the power to which all data should be raised. In order to do this, the Box-Cox power transformation searches from Lambda = -5 to Lamba = +5 until the best value is found. Table 1 shows some common Box-Cox transformations, where Y' is the transformation of the original data Y. Note that for Lambda = 0, the transformation is NOT Y (because this would be 1 for every value) but instead the logarithm of Y.' https://www.isixsigma.com/tools-templates/normality/making-data-normal-using-box-cox-power-transformation/

y = (xlmbda - 1) / lmbda, for lmbda != 0**

log(x), for lmbda = 0

Lambda -----> transformed Y

-2----> 1/power(Y,2)

-1----> 1/Y

-0.5----> 1/(Sqrt(Y))

0----> log(Y)

0.5----> Sqrt(Y)

1----> Y

2----> power(Y,2)

In [31]:
```python
from sklearn import preprocessing

scaler = preprocessing.MinMaxScaler().fit(X1_Train)
X1_train_scaled = scaler.transform(X1_Train)
X1_test_scaled = scaler.transform(X1_Test)
```

In [32]:
```python
print(X1_train_scaled.shape)
print(y1_Train.shape)
```

```
(13500, 74)
(13500,)
```

In [33]:
```python
print(X1_test_scaled .shape)
print(y1_Test.shape)
```

```
(3000, 74)
(3000,)
```

In [34]:
```python
y1_Test.value_counts()
```

Out[34]:
```
0    2998
1       2
Name: default, dtype: int64
```

Modeling

Evaluation

In [35]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, cla
ssification_report
results = pd.DataFrame(columns= ['model', 'Accuracy', 'Precision', 'Recall', 'F1-score', 'AUC'])


def evaluation(model_name, y_test, y_pred):
    print('\n{}\n'.format(model_name),classification_report(y_test, y_pred))
    eval_res = pd.Series([model_name] + [round(accuracy_score(y_test, y_pred ),2),
                              round(precision_score(y_test, y_pred, average='weighted'  ),2),
                              round(recall_score(y_test, y_pred ),2),
                              round(f1_score(y_test, y_pred ),2),
                              round(roc_auc_score(y_test, y_pred ),2)],
                  index= ['model', 'Accuracy', 'Precision', 'Recall', 'F1-score', 'AUC'])

    return eval_res.to_frame().T
```

In [36]:
```python
from sklearn.metrics import confusion_matrix

def conf_mat(y_true, y_pred, set_name,model_name):
    print(model_name,'-', 'Confusion Matrix on', set_name)
    cnf_table = pd.DataFrame(confusion_matrix(y_true, y_pred), index = ['Non_Default', 'Default'],
                columns=['predicted_Non_Default', 'predicted_Default'])
    return cnf_table
```

In [37]:
```python
model_features = ['model_name'] + X1_Train.columns.tolist()
features = pd.DataFrame(columns = model_features)

def importance(model_name, feature_importances_):
    feat_importance = pd.Series([model_name] + np.round(100 * feature_importances_,2).tolist(),
                  index= model_features)
    feat_importance = feat_importance.to_frame()

    return feat_importance
```

In [38]:
```python
def feature_selection(feature_importance, feature_name):
    rel_feat_importance = np.round(100 * (feature_importance / abs(feature_importance).max()),2)
    feat_imp = pd.DataFrame(pd.Series(rel_feat_importance, index = feature_name).sort_values(ascending=
 False)).reset_index()
    feat_imp.columns = ['Feature', 'Relative Importance']
    important_features = feat_imp [abs(feat_imp['Relative Importance']) > 5]\
    .sort_values(by='Relative Importance', ascending= True)
    return important_features
```

In [39]:
```python
def plot_importance(clf_name, important_features, fgsize):
    plt.figure(figsize= fgsize)
    plt.barh(y = important_features.Feature, width=important_features['Relative Importance'], align='ce
nter')
    plt.yticks(important_features.Feature, fontsize= 13)
    plt.xlabel('Relative Feature Importance', fontsize= 13)
    plt.title('Features Importance _ {}'.format(clf_name), color='red', fontsize=16)
    plt.show()
```

Models

Random Forest Classifier

In [40]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

#Setup the hyperparameter grid
param_grid = {'n_estimators': [100, 200, 300],
              'max_features': ['auto', 'sqrt'],       # Number of features to consider at every split
              'max_depth': [int(x) for x in np.linspace(10, 140, num = 14)],
              'bootstrap': [True, False] }


rf = RandomForestClassifier(random_state=42)

# Instantiate the GridSearchCV object
rf_cv= GridSearchCV(rf, param_grid, cv= 5)

# Fit it to the scaled train data
rf_cv.fit(X1_Train, y1_Train)
```

Out[40]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
             param_grid={'bootstrap': [True, False],
                         'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
                                       110, 120, 130, 140],
                         'max_features': ['auto', 'sqrt'],
                         'n_estimators': [100, 200, 300]})
```

In [41]:
```python
best_rf_cv = rf_cv.best_estimator_
y1_Train_pred = best_rf_cv.predict(X1_Train)
y1_Test_pred = best_rf_cv.predict(X1_Test)

eval_df = evaluation('Random Forest Clf on Train ', y1_Train, y1_Train_pred)
results = pd.concat([results, eval_df])
results = evaluation('Random Forest Clf on Test', y1_Test, y1_Test_pred)
results = pd.concat([results, eval_df])


feat_importance = importance('RF_Clf', best_rf_cv.feature_importances_).T
features = pd.concat([features,feat_importance])
```

```
Random Forest Clf on Train
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6995
           1       1.00      1.00      1.00      6505

    accuracy                           1.00     13500
   macro avg       1.00      1.00      1.00     13500
weighted avg       1.00      1.00      1.00     13500


Random Forest Clf on Test
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2998
           1       1.00      1.00      1.00         2

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

In [42]:
```python
conf_mat(y1_Train, y1_Train_pred, 'Train', 'Random Forest Clf')
```

```
Random Forest Clf - Confusion Matrix on Train
```

Out[42]:

	predicted_Non_Default	predicted_Default
Non_Default	6995	0
Default	0	6505

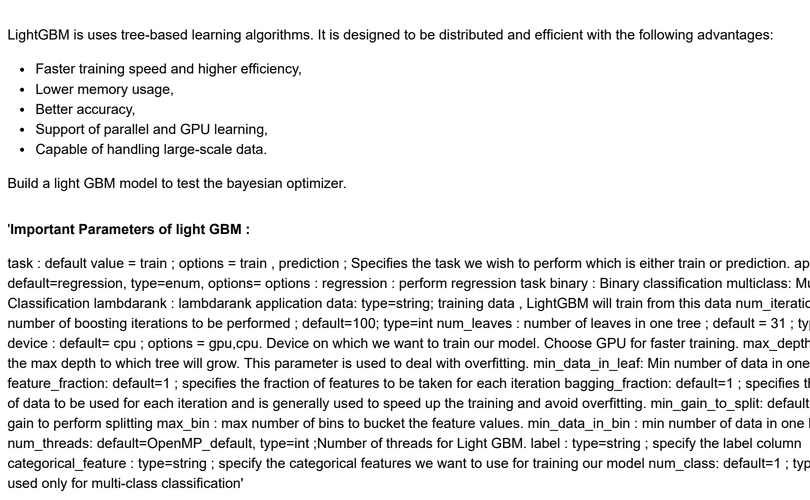
In [43]:
```python
conf_mat(y1_Test, y1_Test_pred, 'Test', 'Random Forest Clf')
```

```
Random Forest Clf - Confusion Matrix on Test
```

Out[43]:

	predicted_Non_Default	predicted_Default
Non_Default	2998	0
Default	0	2

In [44]:
```python
rf_clf_imp_feat = feature_selection(best_rf_cv.feature_importances_, X1_Test.columns)
plot_importance('Random Forest Classifier',rf_clf_imp_feat,(10,10))
```



Light Gradient Boosting Model

LightGBM is uses tree-based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency,
- Lower memory usage,
- Better accuracy,
- Support of parallel and GPU learning,
- Capable of handling large-scale data.

Build a light GBM model to test the bayesian optimizer.

'Important Parameters of light GBM :

task : default value = train ; options = train , prediction ; Specifies the task we wish to perform which is either train or prediction. application: default=regression, type=enum, options= options : regression : perform regression task binary : Binary classification multiclass: Multiclass Classification lambdarank : lambdarank application data: type=string; training data , LightGBM will train from this data num_iterations: number of boosting iterations to be performed ; default=100; type=int num_leaves : number of leaves in one tree ; default = 31 ; type =int device : default= cpu ; options = gpu,cpu. Device on which we want to train our model. Choose GPU for faster training. max_depth: Specify the max depth to which tree will grow. This parameter is used to deal with overfitting. min_data_in_leaf: Min number of data in one leaf. feature_fraction: default=1 ; specifies the fraction of features to be taken for each iteration bagging_fraction: default=1 ; specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting. min_gain_to_split: default=.1 ; min gain to perform splitting max_bin : max number of bins to bucket the feature values. min_data_in_bin : min number of data in one bin num_threads: default=OpenMP_default, type=int ;Number of threads for Light GBM. label : type=string ; specify the label column categorical_feature : type=string ; specify the categorical features we want to use for training our model num_class: default=1 ; type=int ; used only for multi-class classification'

https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

In [45]:
```python
import lightgbm


def lgb_eval(num_leaves, max_depth, lambda_l2 , lambda_l1, min_child_samples, min_data_in_leaf):

    #set the params grid
    params = {'objective': 'binary',
              'metric': 'auc',
              'is_unbalance': True,    #Train set is imbalanced
              'num_leaves': int(num_leaves),
              'max_depth': int(max_depth),
              'lambdal2': lambda_l2,    #L2 regularization
              'lambda_l1' :lambda_l1,   #L1 regularization
              'num_thread': 20,         #set this to the number of real CPU cores
              'min_child_samples': int(min_child_samples),
              'min_data_in_leaf': int(min_data_in_leaf),
              'learning_rate' : 0.03,
              'subsample_freq' : 5,     #frequency for bagging; perform bagging at every 5 iteration
              'bagging_seed': 42,
              'verbosity': -1           #controls the level of LightGBM's verbosity. verbosity<0 :Fatal

    }

    lgb_train = lightgbm.Dataset(X1_Train, y1_Train)

    #cross-validation with given paramaters.
    cv_result = lightgbm.cv(params,
                            train_set = lgb_train,
                            num_boost_round= 1000,        #Number of boosting iterations
                            early_stopping_rounds= 100,   #CV score needs to improve at least every 100
 rounds to continue.
                            stratified = True,            #perform stratified sampling.
                            nfold= 3
                            )

    return cv_result['auc-mean'][-1]
```

In [46]:
```python
from bayes_opt import BayesianOptimization

lgbBO = BayesianOptimization(lgb_eval,
                             {'num_leaves': (25,100),
                              'max_depth': (5,63),
                              'lambda_l2': (0.0, 0.05),
                              'lambda_l1': (0.0, 0.05),
                              'min_child_samples': (50, 100),
                              'min_data_in_leaf': (100, 200)}
                             )

lgbBO.maximize(n_iter= 10, init_points= 2)
```

```
|   iter    |  target   | lambda_l1 | lambda_l2 | max_depth | min_ch... | min_da... | num_le... |
-------------------------------------------------------------------------------------------------
[LightGBM] [Warning] Unknown parameter: lambdal2
[LightGBM] [Warning] min_data_in_leaf is set=146, min_child_samples=82 will be ignored. Current valu
e: min_data_in_leaf=146
|  1        |  1.0      |  0.0149   |  0.02439  |  43.31    |  82.11    |  146.6    |  84.2     |
|  2        |  1.0      |  0.004501 |  0.02758  |  49.8     |  50.83    |  117.1    |  71.55    |
|  3        |  1.0      |  0.003579 |  0.0133   |  11.59    |  93.09    |  197.1    |  25.13    |
|  4        |  1.0      |  0.0161   |  0.01918  |  6.451    |  53.46    |  100.9    |  25.22    |
|  5        |  1.0      |  0.03733  |  0.04811  |  8.829    |  51.39    |  196.3    |  99.39    |
|  6        |  1.0      |  0.01316  |  0.01716  |  62.73    |  99.76    |  101.4    |  95.37    |
|  7        |  1.0      |  0.01851  |  0.04029  |  60.44    |  56.18    |  199.9    |  29.71    |
|  8        |  1.0      |  0.01297  |  0.01765  |  5.249    |  98.37    |  191.9    |  98.43    |
|  9        |  1.0      |  0.03934  |  0.04321  |  62.91    |  99.91    |  199.7    |  25.68    |
|  10       |  1.0      |  0.04734  |  0.001363 |  5.119    |  98.24    |  110.0    |  25.12    |
|  11       |  1.0      |  0.04358  |  0.03736  |  5.501    |  98.52    |  102.7    |  97.0     |
|  12       |  1.0      |  0.00164  |  0.0422   |  57.52    |  50.74    |  105.5    |  26.72    |
=================================================================================================
```

In [47]:
```python
param = lgbBO.max
best_param = param['params']
best_param
```

Out[47]:
```
{'lambda_l1': 0.014904030431068804,
 'lambda_l2': 0.0243871016196213,
 'max_depth': 43.31105148516426,
 'min_child_samples': 82.11410102438798,
 'min_data_in_leaf': 146.58612589195332,
 'num_leaves': 84.19695877678835}
```

In [48]:
```python
for key, val in best_param.items():
    if (key != 'lambda_l1') & (key != 'lambda_l2'):
        best_param[key] = int(val)
```

In [49]:
```python
best_param
```

Out[49]:
```
{'lambda_l1': 0.014904030431068804,
 'lambda_l2': 0.0243871016196213,
 'max_depth': 43,
 'min_child_samples': 82,
 'min_data_in_leaf': 146,
 'num_leaves': 84}
```

In [50]:
```python
train_data = lightgbm.Dataset(X1_Train, label= y1_Train)

num_round= 4
best_lgb = lightgbm.train(best_param, train_data, num_round)
y1_Train_pred = best_lgb.predict(X1_Train)
y1_Test_pred = best_lgb.predict(X1_Test)

for i in np.arange(len(y1_Train_pred)):
    if y1_Train_pred[i] >= 0.50:
        y1_Train_pred[i]= 1
    else:
        y1_Train_pred[i]= 0


for i in np.arange(len(y1_Test_pred)):
    if y1_Test_pred[i] >= 0.50:
        y1_Test_pred[i]= 1
    else:
        y1_Test_pred[i]= 0


eval_df = evaluation('lightgbm on Train', y1_Train, y1_Train_pred)
results = pd.concat([results, eval_df])
eval_df = evaluation('lightgbm on Test', y1_Test, y1_Test_pred)
results = pd.concat([results, eval_df])
```

```
[LightGBM] [Warning] Find whitespaces in feature_names, replace with underlines
[LightGBM] [Warning] min_data_in_leaf is set=146, min_child_samples=82 will be ignored. Current valu
e: min_data_in_leaf=146
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.011460 sec
onds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3951
[LightGBM] [Info] Number of data points in the train set: 13500, number of used features: 69
[LightGBM] [Info] Start training from score 0.481852
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

lightgbm on Train
              precision    recall  f1-score   support

           0       1.00      0.96      0.98      6995
           1       0.96      1.00      0.98      6505

    accuracy                           0.98     13500
   macro avg       0.98      0.98      0.98     13500
weighted avg       0.98      0.98      0.98     13500


lightgbm on Test
              precision    recall  f1-score   support

           0       1.00      0.97      0.98      2998
           1       0.02      1.00      0.04         2

    accuracy                           0.97      3000
   macro avg       0.51      0.98      0.51      3000
weighted avg       1.00      0.97      0.98      3000
```

In [51]:
```python
conf_mat(y1_Train, y1_Train_pred, 'Train', 'LGB Model')
```

LGB Model - Confusion Matrix on Train

Out[51]:

	predicted_Non_Default	predicted_Default
Non_Default	6740	255
Default	0	6505

In [52]:
```python
conf_mat(y1_Test, y1_Test_pred, 'Test', 'LGB Model')
```

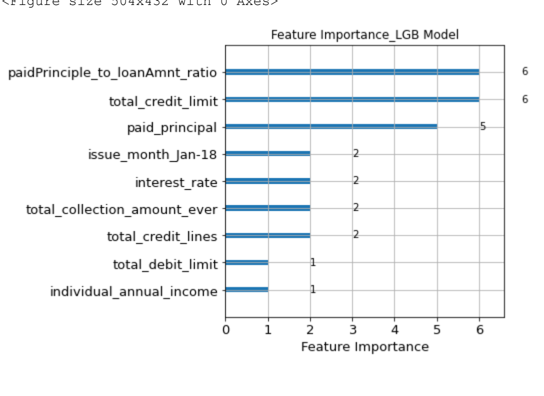
LGB Model - Confusion Matrix on Test

Out[52]:

	predicted_Non_Default	predicted_Default
Non_Default	2898	100
Default	0	2

In [53]:
```python
plt.figure(figsize= (7,6))
ax = lightgbm.plot_importance(best_lgb, max_num_features= 40, figsize=(5,5), title= 'Feature Importance
_LGB Model')
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.xlabel('Feature Importance', fontsize=13)
plt.ylabel('')
plt.show()
```

<Figure size 504x432 with 0 Axes>



Logistic regression

Logistic Regression is a discriminative classifier as we learn a soft boundary between/among classes.

C is the regularization strength (equivalent to 1/alpha from the Ridge case), and smaller values of C mean stronger regularization. As before, the regularization tries to prevent features from having terribly high weights, thus implementing a form of feature selection.

In [54]:
```python
from sklearn.linear_model import LogisticRegression
#the grid of parameters to search over
Cs = [0.001, 0.01, 0.1, 1, 10, 100]
param_grid = {'C': Cs}

#Instantiate a logistic regression classifier: logreg
logreg = LogisticRegression()
# Instantiate the GridSearchCV object: logreg_cv
logreg_cv = GridSearchCV(logreg, param_grid, cv=5)
# Fit it to the data
logreg_cv.fit(X1_Train, y1_Train)

print('Tuned Logistic Regression Parameter:{}'.format(logreg_cv.best_params_))
best_logreg_clf = logreg_cv.best_estimator_
```

```
Tuned Logistic Regression Parameter:{'C': 1}
```

In [55]:
```python
y1_Train_pred = best_logreg_clf.predict(X1_Train)
y1_Test_pred = best_logreg_clf.predict(X1_Test)


eval_df = evaluation('Logistic Regression on Train', y1_Train, y1_Train_pred)
results = pd.concat([results, eval_df])
eval_df = evaluation('Logistic Regression', y1_Test, y1_Test_pred)
results = pd.concat([results, eval_df])
```

```
Logistic Regression on Train
              precision    recall  f1-score   support

           0       1.00      0.88      0.94      6995
           1       0.89      1.00      0.94      6505

    accuracy                           0.94     13500
   macro avg       0.94      0.94      0.94     13500
weighted avg       0.95      0.94      0.94     13500


Logistic Regression
              precision    recall  f1-score   support

           0       1.00      0.88      0.94      2998
           1       0.01      1.00      0.01         2

    accuracy                           0.88      3000
   macro avg       0.50      0.94      0.47      3000
weighted avg       1.00      0.88      0.94      3000
```

In [56]:
```python
conf_mat(y1_Train, y1_Train_pred, 'Train', 'Logistic Regression')
```

Logistic Regression - Confusion Matrix on Train

Out[56]:

	predicted_Non_Default	predicted_Default
Non_Default	6173	822
Default	0	6505

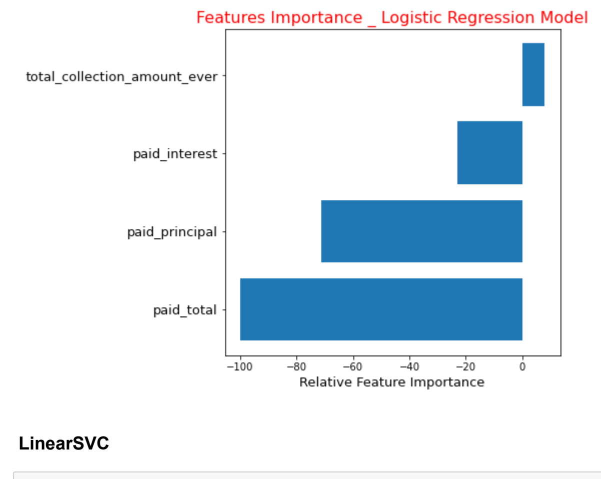
In [57]:
```python
conf_mat(y1_Test, y1_Test_pred, 'Test', 'Logistic Regression')
```

Logistic Regression - Confusion Matrix on Test

Out[57]:

	predicted_Non_Default	predicted_Default
Non_Default	2642	356
Default	0	2

In [58]:
```python
log_reg_imp_feat = feature_selection(best_logreg_clf.coef_[0], X1_Test.columns)
plot_importance('Logistic Regression Model',log_reg_imp_feat, (6,6))
```



LinearSVC

In [59]:
```python
from sklearn.svm import LinearSVC

lnr_svc = LinearSVC()    #class_weight={0 : 1 , 1: 49}
lnr_svc.fit(X1_Train, y1_Train)
y1_Train_pred = lnr_svc.predict(X1_Train)
y1_Test_pred = lnr_svc.predict(X1_Test)

eval_df = evaluation('LinearSVC on Train', y1_Train, y1_Train_pred)
results = pd.concat([results, eval_df])
eval_df = evaluation('LinearSVC on Test', y1_Test, y1_Test_pred)
results = pd.concat([results, eval_df])

feat_importance = importance('LinearSVC', lnr_svc.coef_[0]).T
features = pd.concat([features,feat_importance])
```

```
LinearSVC on Train
              precision    recall  f1-score   support

           0       0.78      0.96      0.86      6995
           1       0.94      0.72      0.81      6505

    accuracy                           0.84     13500
   macro avg       0.86      0.84      0.84     13500
weighted avg       0.86      0.84      0.84     13500


LinearSVC on Test
              precision    recall  f1-score   support

           0       1.00      0.95      0.98      2998
           1       0.01      1.00      0.03         2

    accuracy                           0.95      3000
   macro avg       0.51      0.98      0.50      3000
weighted avg       1.00      0.95      0.98      3000
```

In [60]:
```python
conf_mat(y1_Train, y1_Train_pred, 'Train', 'Linear SVC')
```

Linear SVC - Confusion Matrix on Train

Out[60]:

	predicted_Non_Default	predicted_Default
Non_Default	6714	281
Default	1851	4654

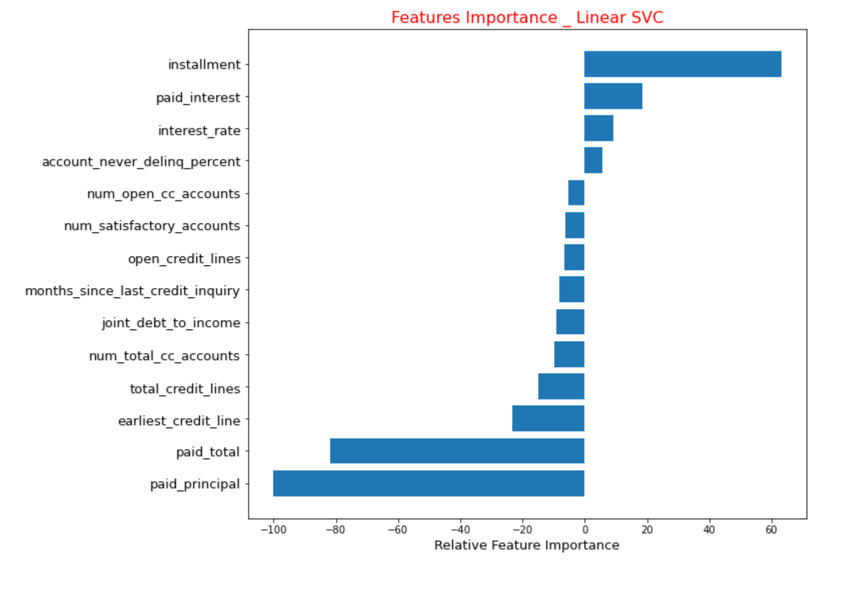
In [61]:
```python
conf_mat(y1_Test, y1_Test_pred, 'Test', 'Linear SVC')
```

Linear SVC - Confusion Matrix on Test

Out[61]:

	predicted_Non_Default	predicted_Default
Non_Default	2859	139
Default	0	2

In [62]:
```python
ln_svc_imp_feat = feature_selection(lnr_svc.coef_[0], X1_Test.columns)
plot_importance('Linear SVC', ln_svc_imp_feat, (10,9))
```



Ridge Classifier

This classifier first converts the target values into {-1, 1} and then treats the problem as a regression task (multi-output regression in the multiclass case).

In [63]:
```python
from sklearn.linear_model import RidgeClassifier
ridge_clf = RidgeClassifier(alpha=1.0)
ridge_clf.fit(X1_Train, y1_Train)

y1_Train_pred = ridge_clf.predict(X1_Train)
y1_Test_pred = ridge_clf.predict(X1_Test)


eval_df = evaluation('Ridge Classifier on Train', y1_Train, y1_Train_pred)
results = pd.concat([results, eval_df])
eval_df = evaluation('Ridge Classifier on Test', y1_Test, y1_Test_pred)
results = pd.concat([results, eval_df])
```

```
Ridge Classifier on Train
              precision    recall  f1-score   support

           0       1.00      0.94      0.97      6995
           1       0.94      1.00      0.97      6505

    accuracy                           0.97     13500
   macro avg       0.97      0.97      0.97     13500
weighted avg       0.97      0.97      0.97     13500


Ridge Classifier on Test
              precision    recall  f1-score   support

           0       1.00      0.94      0.97      2998
           1       0.01      1.00      0.02         2

    accuracy                           0.94      3000
   macro avg       0.51      0.97      0.49      3000
weighted avg       1.00      0.94      0.97      3000
```

In [64]:
```python
conf_mat(y1_Train, y1_Train_pred, 'Train', 'Ridge Clf')
```

Ridge Clf - Confusion Matrix on Train

Out[64]:

	predicted_Non_Default	predicted_Default
Non_Default	6556	439
Default	0	6505

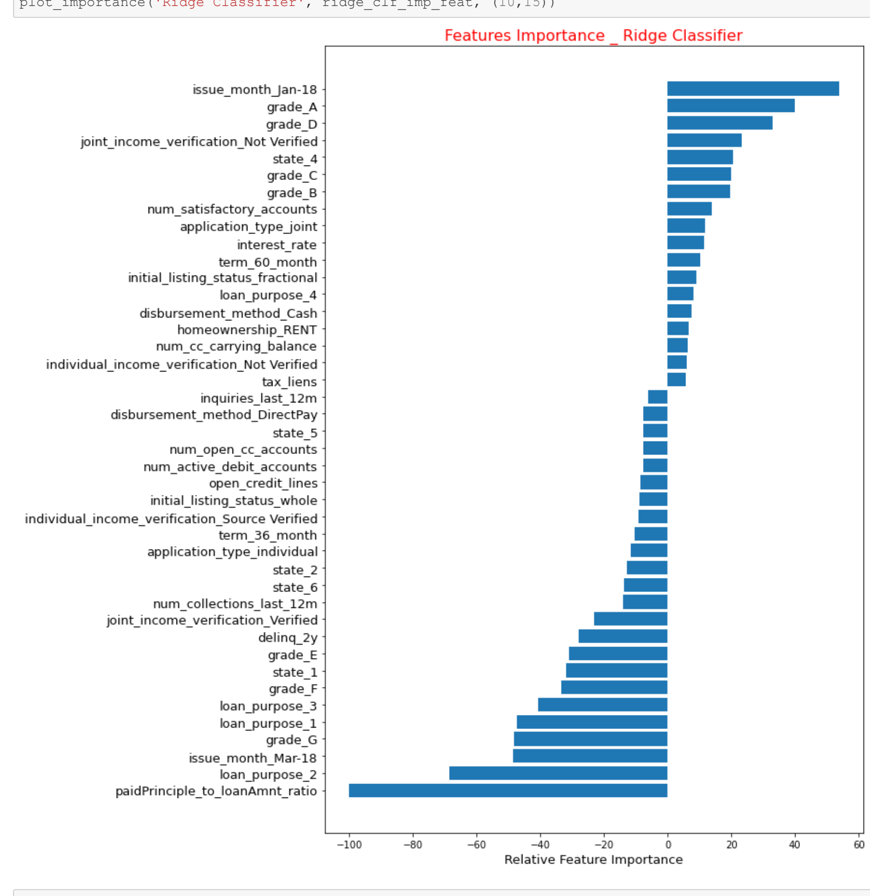
In [65]:
```python
conf_mat(y1_Test, y1_Test_pred, 'Test', 'Ridge Clf')
```

Ridge Clf - Confusion Matrix on Test

Out[65]:

	predicted_Non_Default	predicted_Default
Non_Default	2804	194
Default	0	2

In [66]:
```python
ridge_clf_imp_feat = feature_selection(ridge_clf.coef_[0], X1_Test.columns)
plot_importance('Ridge Classifier', ridge_clf_imp_feat, (10,15))
```



In [67]:
```python
results
```

Out[67]:

	model	Accuracy	Precision	Recall	F1-score	AUC
0	Random Forest Clf on Test	1	1	1	1	1
0	Random Forest Clf on Train	1	1	1	1	1
0	lightgbm on Train	0.98	0.98	1	0.98	0.98
0	lightgbm on Test	0.97	1	1	0.04	0.98
0	Logistic Regression on Train	0.94	0.95	1	0.94	0.94
0	Logistic Regression	0.88	1	1	0.01	0.94
0	LinearSVC on Train	0.84	0.86	0.72	0.81	0.84
0	LinearSVC on Test	0.95	1	1	0.03	0.98
0	Ridge Classifier on Train	0.97	0.97	1	0.97	0.97
0	Ridge Classifier on Test	0.94	1	1	0.02	0.97

In [ ]: