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()
```

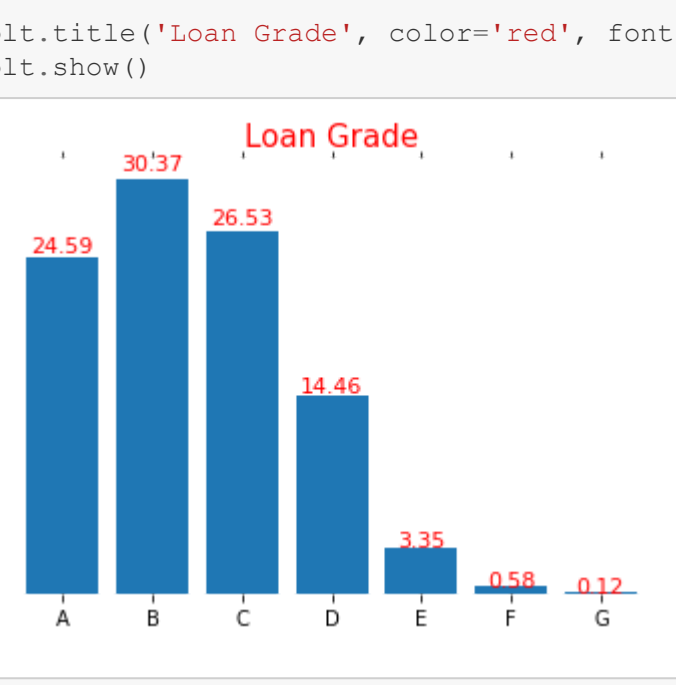
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 50 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   state                             10000 non-null  object 
 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_grade = data.drop(columns= ['interest_rate', 'paid_interest'], axis=1).copy()
data_grade = data.drop(columns\
  = ['interest_rate','sub_grade', 'paid_interest','balance','paid_total','paid_principal','paidPrincipl
e_to_loanAmnt_ratio'], axis=1).copy()
```

In [5]:

```python
grade = round(100 * data_grade.grade.value_counts(normalize= True, dropna= False), 2).sort_index()

bars = plt.bar(grade.index, grade, 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 Grade', color='red', fontsize=15)
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_grade.drop(columns=['grade'], axis= 0).corr(),2)
multicoll_X = pd.DataFrame(multicoll(X_corr), columns= ['var1', 'var2','corr_coef'])
multicoll_X
```

Out[7]:

	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

Encoding

In [8]:

```python
categorical_features = data_grade.drop(columns= ['grade'], axis= 1).select_dtypes('object').columns.val
ues
uniq_values = data_grade[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[8]:

	categorical_feature	n_unique_values
0	state	50
1	loan_purpose	12
2	loan_status	6
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 [9]:

```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 [10]:

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

data_grade = One_hot_encoder(data_grade, lst_cols)
data_grade.shape
```

Out[10]:

```
(10000, 55)
```

Split X, Y into train and test

In [11]:

```python
#data_grade['grade'].replace({'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7}, inplace= True)
```

In [12]:

```python
y2 = data_grade['grade']
X2 = data_grade.drop(['grade'], axis=1)
```

In [13]:

```python
from sklearn.model_selection import train_test_split
X2_train, X2_test, y2_train, y2_test= train_test_split(X2, y2, test_size=0.30, random_state= 42, strati
fy= y2)
```

In [14]:

```python
y_train_test = pd.DataFrame(pd.concat([round(100 * y2_test.value_counts(normalize= True, dropna= False
),2)\
                                        ,  round(100 * y2_train.value_counts(normalize= True, dropna= Fa
lse),2)], axis= 1))
y_train_test.columns= ['y_test_grade', 'y_train_grade']
y_train_test
```

Out[14]:

	y_test_grade	y_train_grade
B	30.37	30.37
C	26.53	26.53
A	24.60	24.59
D	14.47	14.46
E	3.33	3.36
F	0.57	0.59
G	0.13	0.11

BinaryEncoder

In [15]:

```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 [16]:

```python
#encoded_train, encoded_test = binary_encoder(X2, X2_train, X2_test, y2_train, y2_test\
                                            # , ['state', 'sub_grade', 'loan_purpose', 'loan_statu
s'])
encoded_train, encoded_test = binary_encoder(X2, X2_train, X2_test, y2_train, y2_test\
                                             , ['state', 'loan_purpose', 'loan_status'])
```

In [17]:

```python
X2_Train = pd.concat([X2_train, encoded_train], axis=1, join='inner')
#temp = X2_Train.drop(columns=['state', 'sub_grade', 'loan_purpose', 'loan_status'], axis=1)
temp = X2_Train.drop(columns=['state', 'loan_purpose', 'loan_status'], axis=1)
X2_Train = temp.copy()

y2_Train = y2_train.copy()
```

In [18]:

```python
X2_Test = pd.concat([X2_test, encoded_test], axis=1, join='inner')
#temp = X2_Test.drop(columns=['state', 'sub_grade', 'loan_purpose', 'loan_status'], axis=1)
temp = X2_Test.drop(columns=['state', 'loan_purpose', 'loan_status'], axis=1)
X2_Test = temp.copy()

y2_Test = y2_test.copy()
```

In [19]:

```python
print(X2_Test.shape)
print(y2_Test.shape)
```

```
(3000, 67)
(3000,)
```

In [20]:

```python
print(X2_Train.shape)
print(y2_Train.shape)
```

```
(7000, 67)
(7000,)
```

Resampling

In [21]:

```python
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X2_Train_smote, y2_Train_smote = smote.fit_resample(X2_Train, y2_Train)
```

In [22]:

```python
X2_Train_smote.shape
```

Out[22]:

```
(14882, 67)
```

In [23]:

```python
y2_Train_smote.shape
```

Out[23]:

```
(14882,)
```

In [24]:

```python
round(100 * y2_Train_smote.value_counts(normalize= True, dropna= False),2)
```

Out[24]:

```
D    14.29
E    14.29
G    14.29
B    14.29
F    14.29
C    14.29
A    14.29
Name: grade, dtype: float64
```

Scaling

I adopted the MinMaxScaler.

In [25]:

```python
from sklearn import preprocessing

scaler = preprocessing.MinMaxScaler().fit(X2_Train_smote)
X2_Train_smote_scaled = scaler.transform(X2_Train_smote)
X2_Test_scaled = scaler.transform(X2_Test)
```

In [26]:

```python
print(X2_Train_smote_scaled.shape)
print(y2_Train_smote.shape)
```

```
(14882, 67)
(14882,)
```

In [27]:

```python
y2_Train_smote.value_counts()
```

Out[27]:

```
D    2126
E    2126
G    2126
B    2126
F    2126
C    2126
A    2126
Name: grade, dtype: int64
```

In [28]:

```python
y2_Test.value_counts()
```

Out[28]:

```
B    911
C    796
A    738
D    434
E    100
F     17
G      4
Name: grade, dtype: int64
```

Modeling

In [29]:

```python
#Rename for simplicity
X_train = X2_Train_smote_scaled.copy()
y_train = y2_Train_smote.copy()

X_test = X2_Test_scaled.copy()
y_test = y2_Test.copy()
```

In [30]:

```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, average='weighted'),2),
                            round(f1_score(y_test, y_pred, average='weighted'),2),
                            round(roc_auc_score(y_test, y_pred, multi_class ='ovr' ),2)],
                    index= ['model', 'Accuracy', 'Precision', 'Recall', 'F1-score', 'AUC'])

    return eval_res.to_frame().T
```

In [31]:

```python
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV


def grid_cv(model, grid, X, y):
    cv = RepeatedStratifiedKFold(n_splits= 10, n_repeats= 5, random_state= 42)
    grid_search_cv = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accurac
y',error_score=0)
    fitted_model = grid_search_cv.fit(X,y)

    return fitted_model.best_score_, fitted_model.best_params_
```

In [32]:

```python
from sklearn.metrics import confusion_matrix

def conf_mat(y_true, y_pred, set_name):
    print('Confusion Matrix on ', set_name)
    cnf_table = pd.DataFrame(confusion_matrix(y_true, y_pred), index = ['A','B', 'C','D','E','F','G'],
                columns=['predicted_A', 'predicted_B', 'predicted_C', 'predicted_D', 'predicted_E','pre
dicted_F'\
                         ,'predicted_G'])
    return cnf_table
```

In [33]:

```python
def acc_score(y_true, y_pred, set_name):
    acc = np.round(100 *accuracy_score(y_true, y_pred),2)
    print('Accuracy Score on {} : {}'.format(set_name, acc))
    print('\nClassification Report on', set_name)
    class_report = classification_report(y_true, y_pred)
    return class_report
```

In [34]:

```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']) > 2]\
    .sort_values(by='Relative Importance', ascending= True)
    return important_features
```

In [50]:

```python
def feature_selection2(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']) > 20]\
    .sort_values(by='Relative Importance', ascending= True)
    return important_features
```

In [36]:

```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 Importance', fontsize= 13)
    plt.title('Relative Features Importance _ {}'.format(clf_name), color='red', fontsize=16)
    plt.show()
```

Multinomial Logistic Regression

Sometimes, you can see useful differences in performance or convergence with different solvers (solver).

solver in ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

Regularization (penalty) can sometimes be helpful.

penalty in ['none', 'l1', 'l2', 'elasticnet']

Note: not all solvers support all regularization terms.

The C parameter controls the penality strength, which can also be effective.

C in [100, 10, 1.0, 0.1, 0.01]

Mean of classification accuracy across all folds and repeats in cross validation is 97%.

In [37]:

```python
from sklearn.linear_model import LogisticRegression

Cs = [0.001, 0.01, 0.1, 1, 10, 100]
param_grid = {'C': Cs}
multi_log_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs')
best_score, best_param = grid_cv(multi_log_reg, param_grid, X_train, y_train)
print('best score: ', np.round(best_score,2))
print('best param: ', best_param)
```

```
best score:  0.64
best param:  {'C': 100}
```

In [38]:

```python
best_multi_log_reg = LogisticRegression(C = best_param['C'], multi_class='multinomial', solver='lbfgs')
best_multi_log_reg.fit(X_train, y_train)

y_pred_train = best_multi_log_reg.predict(X_train)
y_pred_test = best_multi_log_reg.predict(X_test)
```

In [39]:

```python
print(acc_score(y_train, pd.Series(y_pred_train), 'Train Set'))
conf_mat(y_train, pd.Series(y_pred_train),'Train Set')
```

```
Accuracy Score on Train Set : 65.02

Classification Report on Train Set
              precision    recall  f1-score   support

           A       0.67      0.73      0.70      2126
           B       0.47      0.48      0.47      2126
           C       0.43      0.40      0.41      2126
           D       0.52      0.45      0.48      2126
           E       0.70      0.63      0.66      2126
           F       0.77      0.88      0.82      2126
           G       0.94      0.98      0.96      2126

    accuracy                           0.65     14882
   macro avg       0.64      0.65      0.64     14882
weighted avg       0.64      0.65      0.64     14882

Confusion Matrix on  Train Set
```

Out[39]:

	predicted_A	predicted_B	predicted_C	predicted_D	predicted_E	predicted_F	predicted_G
A	1548	457	89	32	0	0	0
B	467	1021	530	89	15	2	2
C	194	557	850	408	89	25	3
D	88	144	427	958	333	150	26
E	3	8	74	299	1340	359	43
F	0	0	4	55	149	1866	52
G	0	0	0	2	1	30	2093

In [40]:

```python
print(acc_score(y_test, pd.Series(y_pred_test), 'Test Set'))
conf_mat(y_test, pd.Series(y_pred_test), 'Test Set')
```

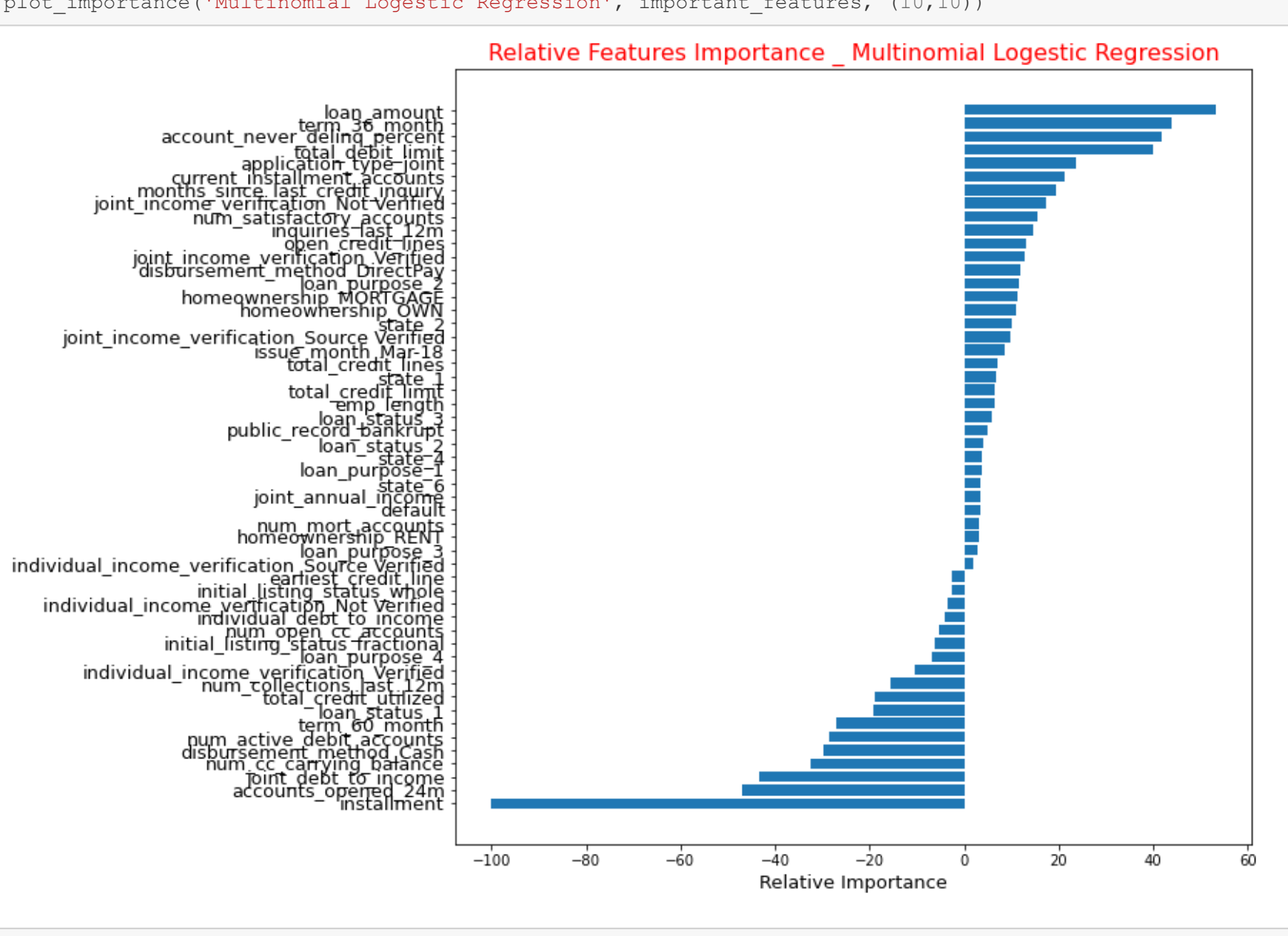
```
Accuracy Score on Test Set : 47.97

Classification Report on Test Set
              precision    recall  f1-score   support

           A       0.66      0.68      0.67       738
           B       0.46      0.47      0.47       911
           C       0.42      0.46      0.44       796
           D       0.36      0.28      0.31       434
           E       0.24      0.20      0.22       100
           F       0.22      0.24      0.23        17
           G       0.00      0.00      0.00         4

    accuracy                           0.48      3000
   macro avg       0.34      0.33      0.33      3000
weighted avg       0.47      0.48      0.48      3000

Confusion Matrix on  Test Set
```

Out[40]:

	predicted_A	predicted_B	predicted_C	predicted_D	predicted_E	predicted_F	predicted_G
A	504	189	42	3	0	0	0
B	182	426	251	44	4	1	3
C	61	233	364	118	14	4	2
D	17	67	180	121	42	6	1
E	2	3	29	43	20	3	0
F	0	1	1	8	3	4	0
G	0	1	0	1	2	0	0

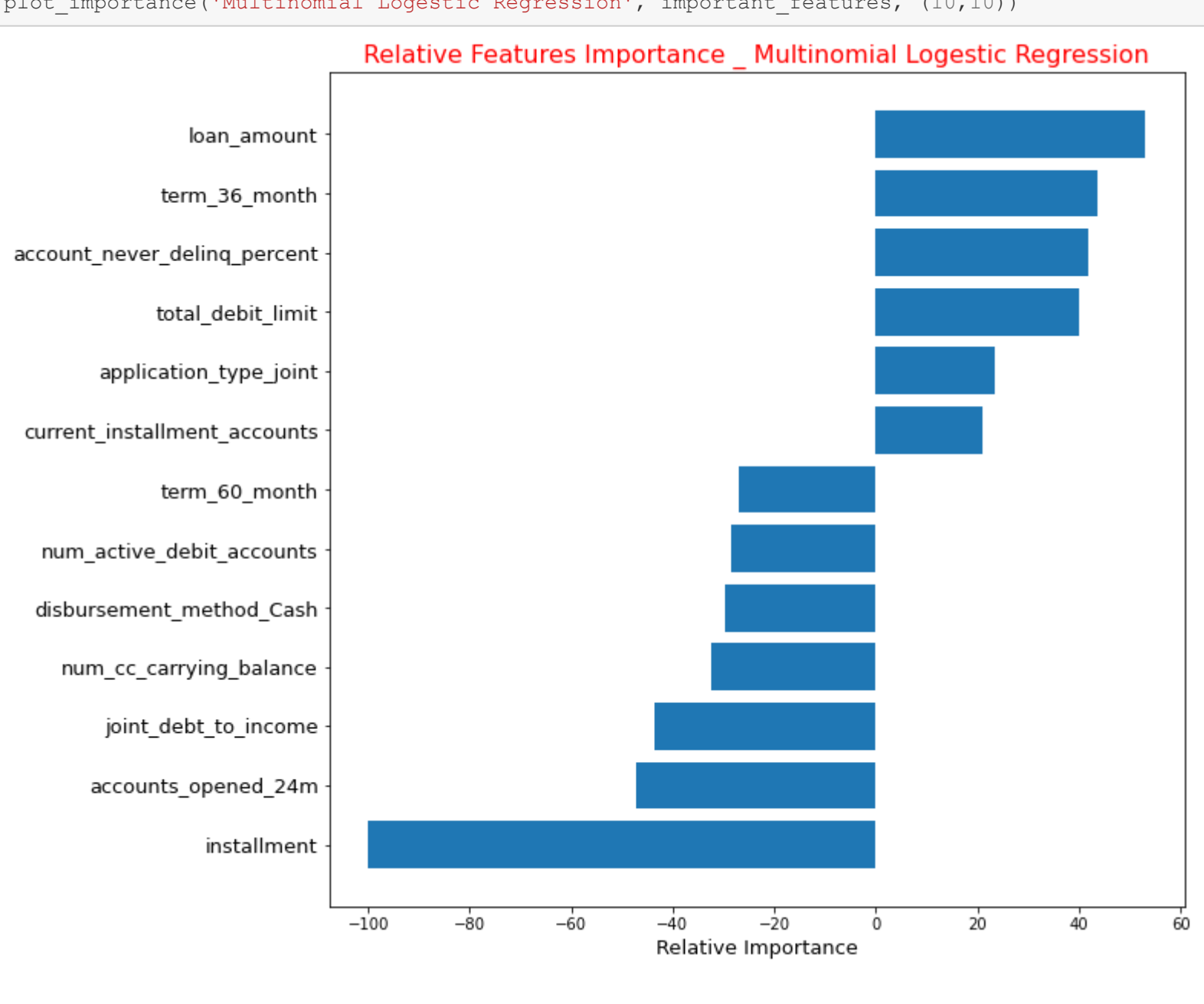
In [41]:

```python
important_features = feature_selection(best_multi_log_reg.coef_[0], X2_Train_smote.columns)
plot_importance('Multinomial Logistic Regression', important_features, (10,10))
```



In [51]:

```python
important_features = feature_selection2(best_multi_log_reg.coef_[0], X2_Train_smote.columns)
plot_importance('Multinomial Logistic Regression', important_features, (10,10))
```



Random Forest Classifier

In [43]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
```

In [44]:

```python
param_grid = {'max_features': ['auto', 'sqrt'],        # Number of features to consider at every split
              'max_depth': [int(x) for x in np.linspace(10, 140, num = 9)],
              'bootstrap': [True, False] }

rf_clf = RandomForestClassifier(random_state=42)
best_score, best_param = grid_cv(rf_clf, param_grid, X_train, y_train)
print('best score: ', np.round(best_score,2))
print('best param: ', best_param)
```

```
best score:  0.78
best param:  {'bootstrap': False, 'max_depth': 42, 'max_features': 'auto'}
```

In [45]:

```python
best_rf_clf = RandomForestClassifier(random_state=42)
best_rf_clf.fit(X_train, y_train)

y_pred_train = best_rf_clf.predict(X_train)
y_pred_test = best_rf_clf.predict(X_test)
```

In [46]:

```python
print(acc_score(y_train, pd.Series(y_pred_train), 'Train Set'))
conf_mat(y_train, pd.Series(y_pred_train),'Train Set')
```

```
Accuracy Score on Train Set : 100.0

Classification Report on Train Set
              precision    recall  f1-score   support

           A       1.00      1.00      1.00      2126
           B       1.00      1.00      1.00      2126
           C       1.00      1.00      1.00      2126
           D       1.00      1.00      1.00      2126
           E       1.00      1.00      1.00      2126
           F       1.00      1.00      1.00      2126
           G       1.00      1.00      1.00      2126

    accuracy                           1.00     14882
   macro avg       1.00      1.00      1.00     14882
weighted avg       1.00      1.00      1.00     14882

Confusion Matrix on  Train Set
```

Out[46]:

	predicted_A	predicted_B	predicted_C	predicted_D	predicted_E	predicted_F	predicted_G
A	2126	0	0	0	0	0	0
B	0	2126	0	0	0	0	0
C	0	0	2126	0	0	0	0
D	0	0	0	2126	0	0	0
E	0	0	0	0	2126	0	0
F	0	0	0	0	0	2126	0
G	0	0	0	0	0	0	2126

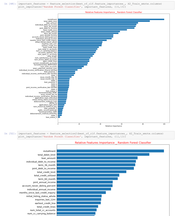
In [47]:

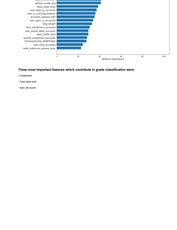
```python
print(acc_score(y_test, pd.Series(y_pred_test), 'Test Set'))
conf_mat(y_test, pd.Series(y_pred_test), 'Test Set')
```

```
Accuracy Score on Test Set : 48.0

Classification Report on Test Set
              precision    recall  f1-score   support

           A       0.61      0.76      0.67       738
           B       0.45      0.44      0.45       911
           C       0.44      0.44      0.44       796
           D       0.36      0.29      0.32       434
           E       0.16      0.08      0.11       100
           F       0.00      0.00      0.00        17
           G       0.00      0.00      0.00         4

    accuracy                           0.48      3000
   macro avg       0.29      0.29      0.28      3000
weighted avg       0.46      0.48      0.47      3000

Confusion Matrix on  Test Set
```

Out[47]:

	predicted_A	predicted_B	predicted_C	predicted_D	predicted_E	predicted_F	predicted_G
A	558	144	34	2	0	0	0
B	235	400	216	55	5	0	0
C	87	236	347	114	12	0	0
D	38	89	158	127	21	1	0
E	2	12	33	43	8	2	0
F	0	2	4	8	3	0	0
G	0	2	0	2	0	0	0





Three most important features which contribute to grade classification were:

- Installment
- Total debit limit
- term 36 month