-- coding: utf-8 --

""" Created on Tue Apr 26 15:40:18 2019

@author: Saiharshavardhan, Sarathi, Soujanya, Antony, Amit, Praveen, Siddharth

"""

#%% """ SCOPE OF THIS PROJECT 1.IMPORTING THE REQUIRED LIBRARIES 2.READING THE DATASET 3.FINDING AND REMOVING THE MISSING VALUE COLUMNS 4.FILTERING THE DATA WITH REQUIRED COLUMNS 5.PLOTTING CORRULATION MATRIX 6.DROPPING THE ROWS HAVING NULL VALUES 7.DATA VIZUALIZATION 8.TRANSFORMATION FUNCTION DEFINITION 9.DATA SPLIT 10.CREATE ROC 11.CROSS VALIDATION 12.MODELLING 13.FITTING 14.CONFUSION MATRIX

"""

In [1]:

```
#iMPORTING LIBRATIES
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from scipy.stats import boxcox
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
##Reading the dataset
df = pd.read_csv("G:/Analytics/Project Lending Data/XYZCorp_LendingData.txt", sep = '\t',na
df.head(3)
```

Out[2]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installmer
0	1077501	1296599	5000.0	5000.0	4975.0	36 months	10.65	162.8
1	1077430	1314167	2500.0	2500.0	2500.0	60 months	15.27	59.8
2	1077175	1313524	2400.0	2400.0	2400.0	36 months	15.96	84.3

3 rows × 73 columns

In [3]:

```
#3.FINDING AND REMOVING THE MISSING VALUE COLUMNS
##Finding the the count and percentage of values that are missing in the dataframe.
df_null = pd.DataFrame({'Count': df.isnull().sum(), 'Percent': 100*df.isnull().sum()/len(df

##printing columns with null count more than 0
df_null[df_null['Count'] > 0]

df1 = df.dropna(axis=1, thresh=int(0.50*len(df)))
df1.head(5)

##Finding the the count and percentage of values that are missing in the dataframe.
df_null = pd.DataFrame({'Count': df1.isnull().sum(), 'Percent': 100*df1.isnull().sum()/len(


##printing columns with null count more than 0
df_null[df_null['Count'] > 0]
```

Out[3]:

	Count	Percent
emp_title	49443	5.776261
emp_length	43061	5.030673
title	33	0.003855
revol_util	446	0.052105
last_pymnt_d	8862	1.035318
next_pymnt_d	252971	29.553757
last_credit_pull_d	50	0.005841
collections_12_mths_ex_med	56	0.006542
tot_coll_amt	67313	7.863953
tot_cur_bal	67313	7.863953
total_rev_hi_lim	67313	7.863953

In [4]:

```
#4.FILTERING THE DATA WITH REQUIRED COLUMNS
dff = df1.filter(['loan_amnt','term','int_rate','installment','grade','sub_grade','emp_leng
                  'annual_inc','verification_status','purpose','dti','delinq_2yrs','defau
dff.dtypes
```

Out[4]:

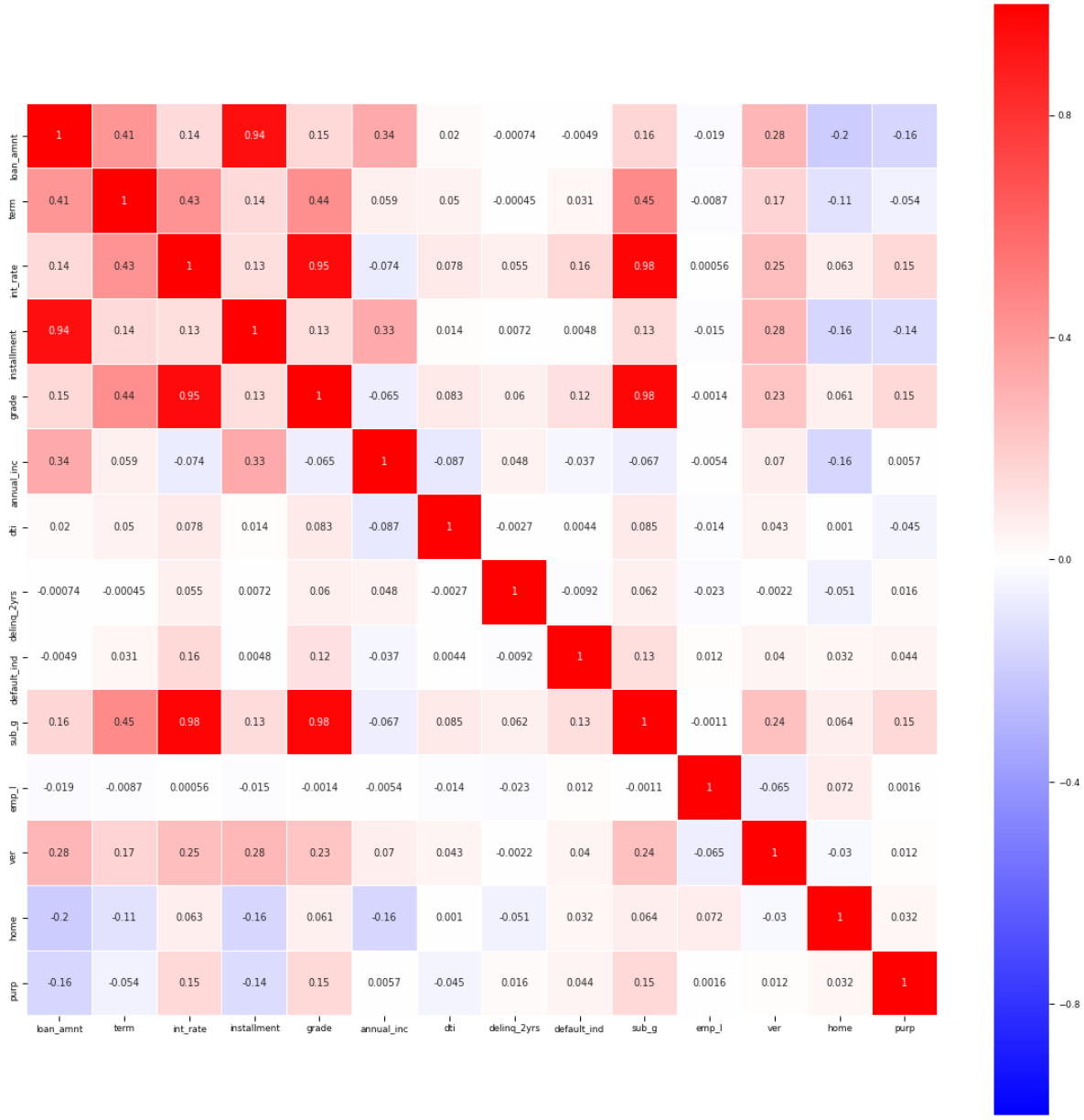
```
loan_amnt             float64
term                   object
int_rate              float64
installment           float64
grade                  object
sub_grade              object
emp_length             object
home_ownership         object
annual_inc            float64
verification_status    object
purpose                object
dti                   float64
delinq_2yrs           float64
default_ind             int64
issue_d                object
dtype: object
```

In [5]:

```
#%%
#5.PLOTTING CORRULATION MATRIX
plt.figure(figsize=(20,20))
sns.set_context("paper", font_scale=1)
##correllation matrix and changing the categorical data to category for the plot.
sns.heatmap(dff.assign(grade=dff.grade.astype('category').cat.codes,
                       sub_g=dff.sub_grade.astype('category').cat.codes,
                       term=dff.term.astype('category').cat.codes,
                      emp_l=dff.emp_length.astype('category').cat.codes,
                       ver =dff.verification_status.astype('category').cat.codes,
                      home=dff.home_ownership.astype('category').cat.codes,
                      purp=dff.purpose.astype('category').cat.codes).corr(),
                       annot=True, cmap='bwr',vmin=-1, vmax=1, square=True, linewidths=0.
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x5a80080940>
```



In [6]:

```
#%%
#6.DROPPING THE ROWS HAVING NULL VALUES
##printing the count and null values in the dataframe
dflc_null = pd.DataFrame({'Count': dff.isnull().sum(), 'Percent': 100*dff.isnull().sum()/le
dflc_null[dflc_null['Count'] > 0]
## dropping the null rows since we have sufficient amount of data and there is no need to f
dff.dropna(axis=0)
#%%

#dff.drop(['installment','sub_grade','verification_status','term'], axis=1, inplace = True)

## printing unique statuses in the loan status column (dependent variable)
dff['default_ind'].unique()
```

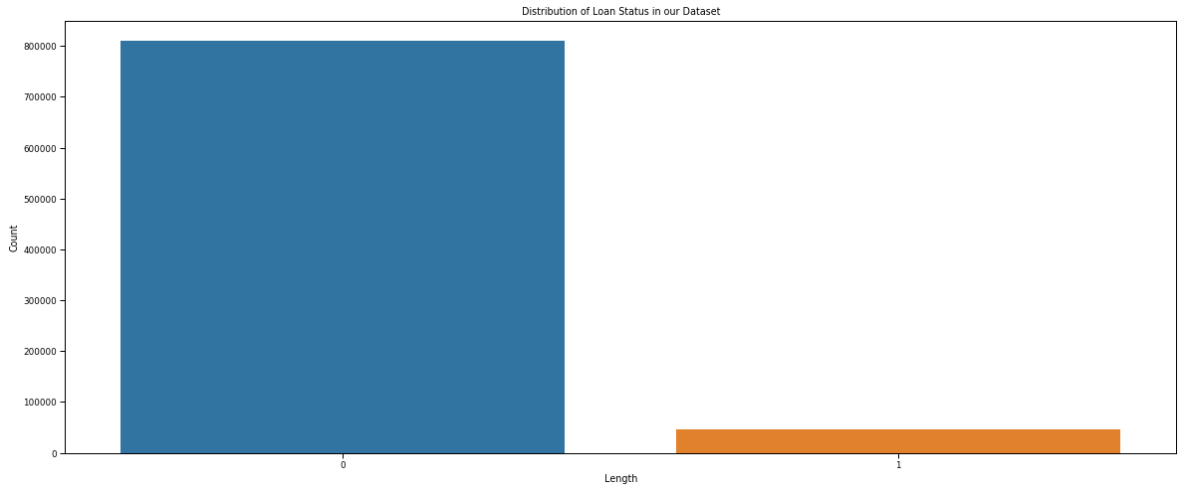
Out[6]:

```
array([0, 1], dtype=int64)
```

#7.DATA VIZUALIZATION

In [7]:

```
#Distribution of the loan status values
m =dff['default_ind'].value_counts()
m = m.to_frame()
m.reset_index(inplace=True)
m.columns = ['Loan Status','Count']
plt.subplots(figsize=(20,8))
sns.barplot(y='Count', x='Loan Status', data=m)
plt.xlabel("Length")
plt.ylabel("Count")
plt.title("Distribution of Loan Status in our Dataset")
plt.show()
```



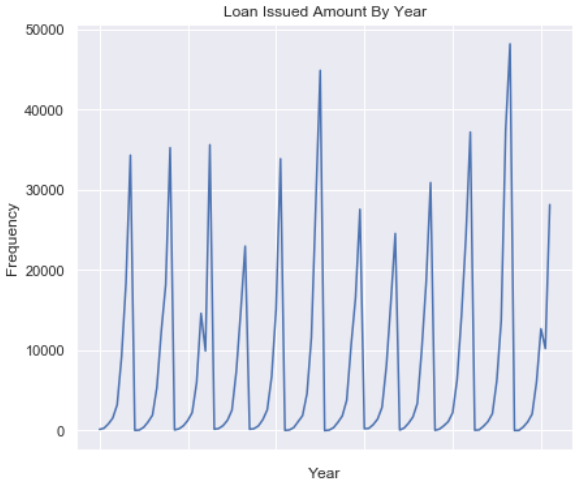
In [8]:

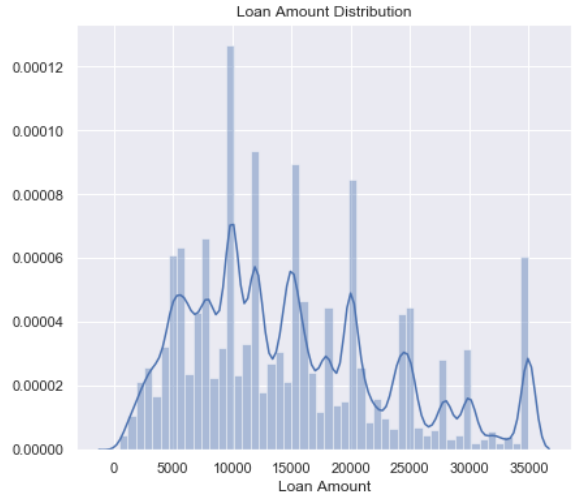
```
#Loan Distribution by Year and Amount
year_dist = dff.groupby(['issue_d']).size()

plt.figure(figsize=(15,6))
sns.set()

ax1 = plt.subplot(1, 2, 1)
ax1 = year_dist.plot()
ax1 = plt.title('Loan Issued Amount By Year')
ax1 = plt.xlabel('Year')
ax1 = plt.ylabel('Frequency')

ax2 = plt.subplot(1, 2, 2)
ax2 = sns.distplot(df['loan_amnt'])
ax2 = plt.title('Loan Amount Distribution')
ax2 = plt.xlabel('Loan Amount')

plt.figure(figsize=(15,6))
ax = sns.countplot(y=df['verification_status'],order = df['verification_status'].value_cour
ax = plt.title('Loan Verification Status Distribution')

"""
# remove "months" from "36 months" and convert it to int type
df_LC["term"]= df_LC['term'].map(lambda x: x.rstrip('months'))
df_LC["term"]= df_LC["term"].astype("int")
"""
```

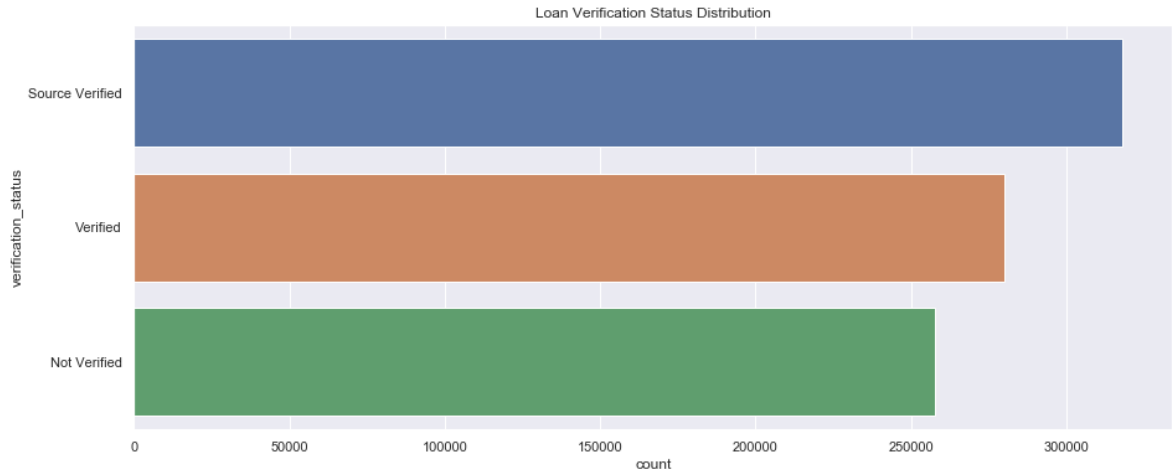
```
C:\Users\acer\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is deprecat
ed; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be
interpreted as an array index, `arr[np.array(seq)]`, which will result eithe
r in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[8]:

```
'\n# remove "months" from "36 months" and convert it to int type\ndf_LC["ter
m"]= df_LC[\'term\'].map(lambda x: x.rstrip(\'months\'))\ndf_LC["term"]= df_
LC["term"].astype("int")\n'
```





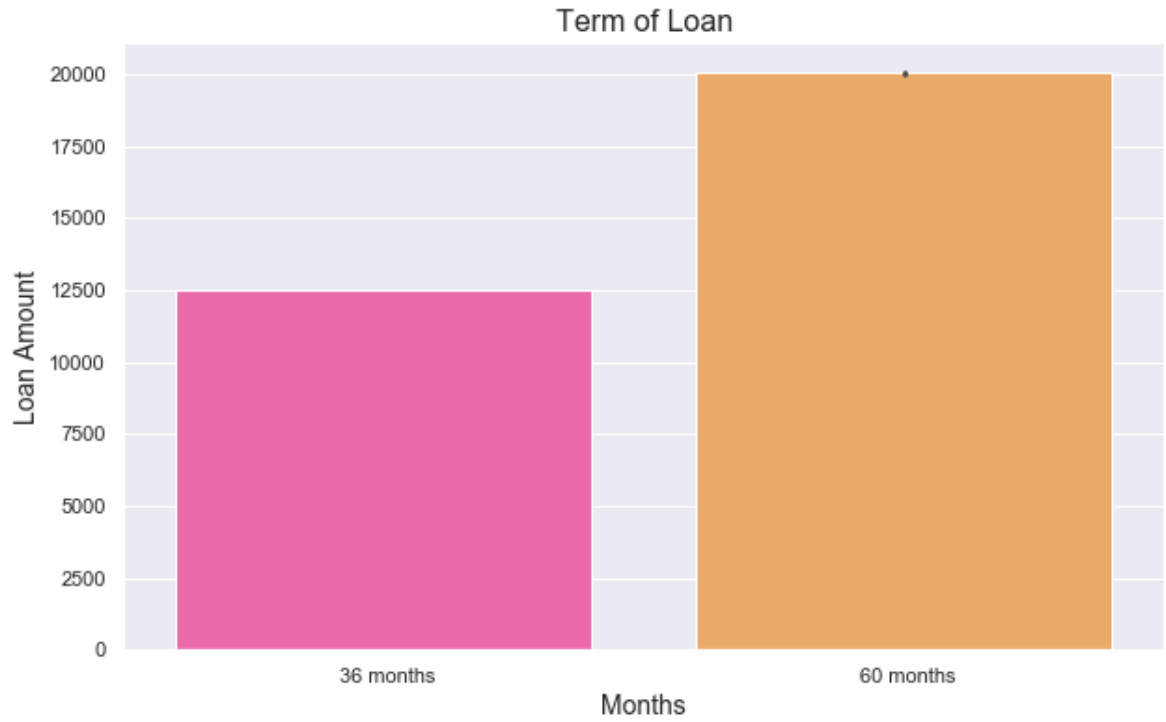


In [9]:

```
# Bar plot of Term (Loan taken for number of months)
plt.figure(figsize=(10,6))
sns.barplot("term", "loan_amnt", data=dff, palette='spring')
plt.title("Term of Loan", fontsize=16)
plt.xlabel("Months", fontsize=14)
plt.ylabel("Loan Amount", fontsize=14)
```

Out[9]:

```
Text(0, 0.5, 'Loan Amount')
```

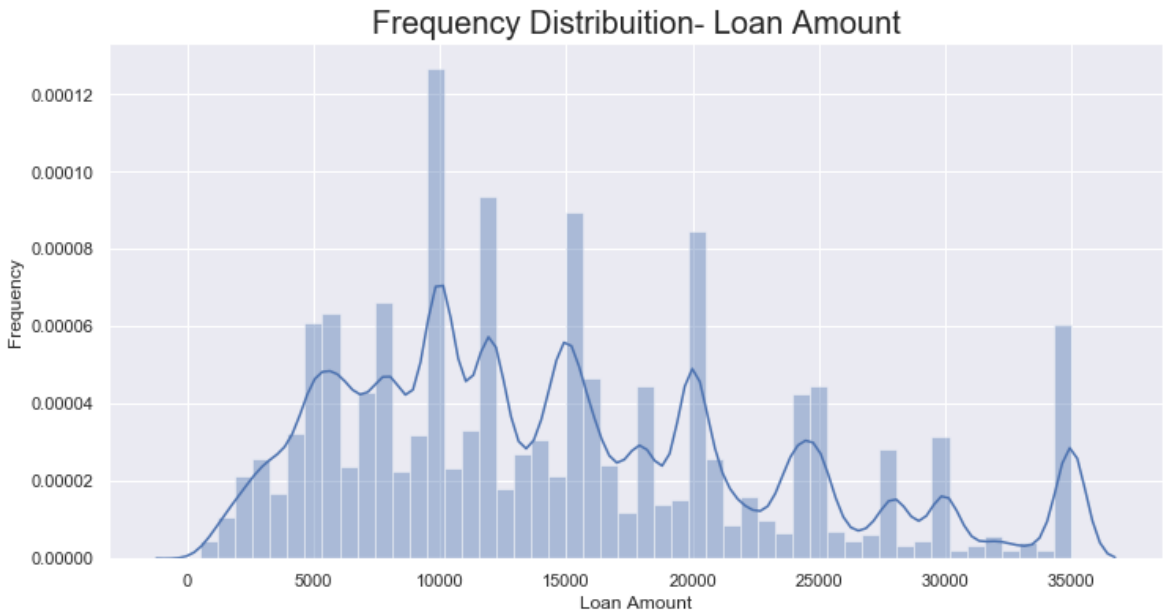


In [10]:

```
# Frequency distribution of Loan Amount
#THis is previously done
plt.figure(figsize=(12,6))
g = sns.distplot(dff["loan_amnt"])
g.set_xlabel("Loan Amount", fontsize=12)
g.set_ylabel("Frequency", fontsize=12)
g.set_title("Frequency Distribuition- Loan Amount", fontsize=20)
```

Out[10]:

```
Text(0.5, 1.0, 'Frequency Distribuition- Loan Amount')
```

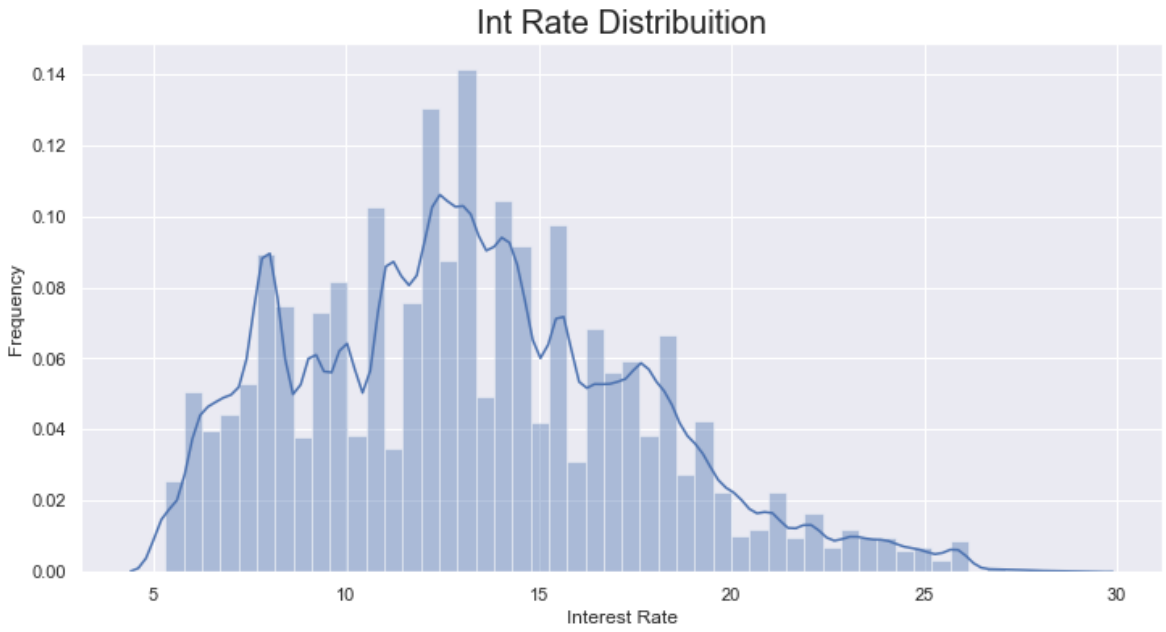


In [11]:

```
# Frequency distribution of Interest Rate
plt.figure(figsize=(12,6))
g = sns.distplot(dff["int_rate"])
g.set_xlabel("Interest Rate", fontsize=12)
g.set_ylabel("Frequency", fontsize=12)
g.set_title("Int Rate Distribuition", fontsize=20)
```

Out[11]:

```
Text(0.5, 1.0, 'Int Rate Distribuition')
```

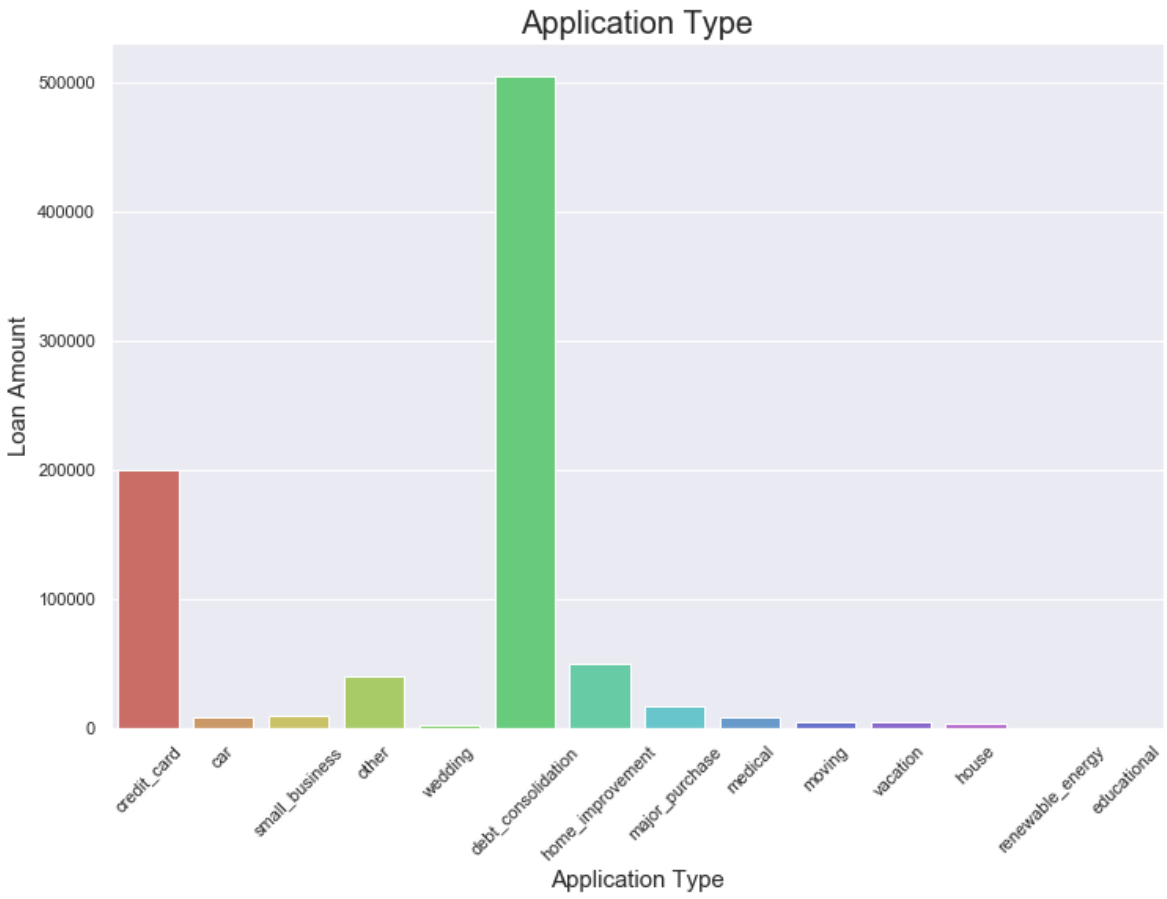


In [12]:

```
# Application Type and Loan Amount
plt.figure(figsize = (12,8))
g = sns.countplot(x="purpose",data=dff,
                  palette='hls')
g.set_xticklabels(g.get_xticklabels(),rotation=45)
g.set_title("Application Type", fontsize=20)
g.set_xlabel("Application Type", fontsize=15)
g.set_ylabel("Loan Amount", fontsize=15)
```

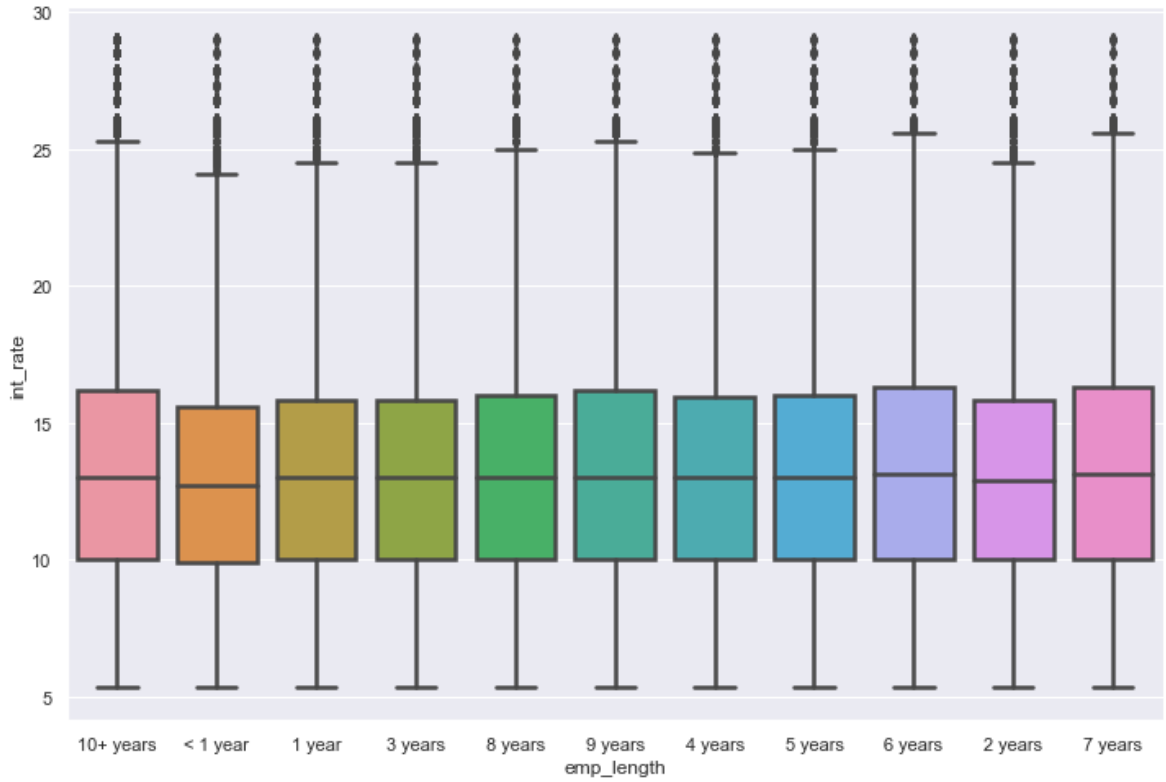
Out[12]:

```
Text(0, 0.5, 'Loan Amount')
```



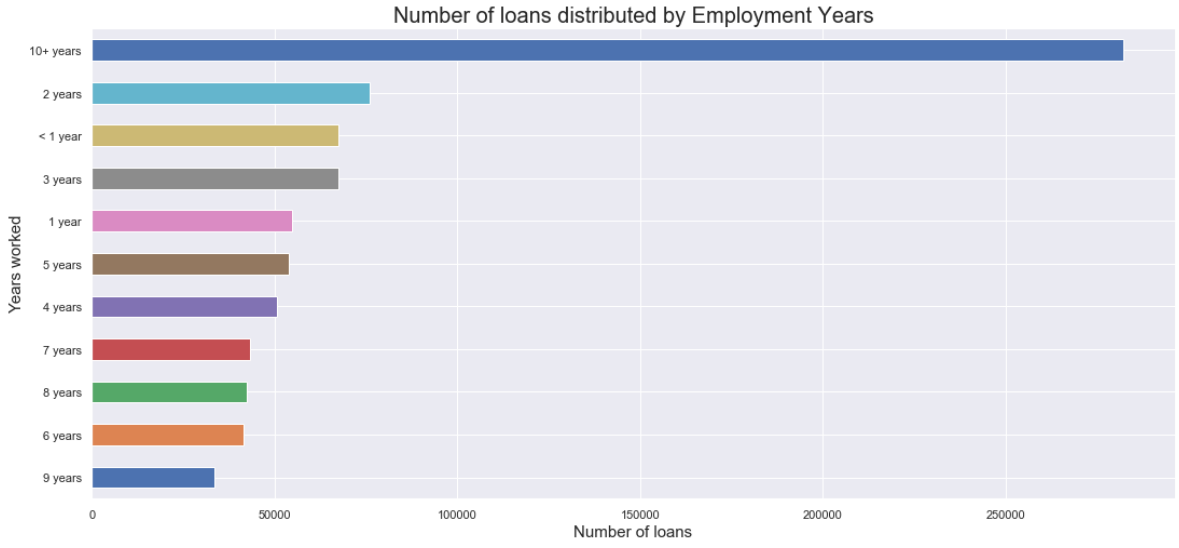
In [13]:

```
# Boxplot of Employment Length and Inter
plt.figure(figsize = (12,8))
ax = sns.boxplot(x="emp_length" , y= "int_rate", data=dff, linewidth=2.5)
plt.show()
```



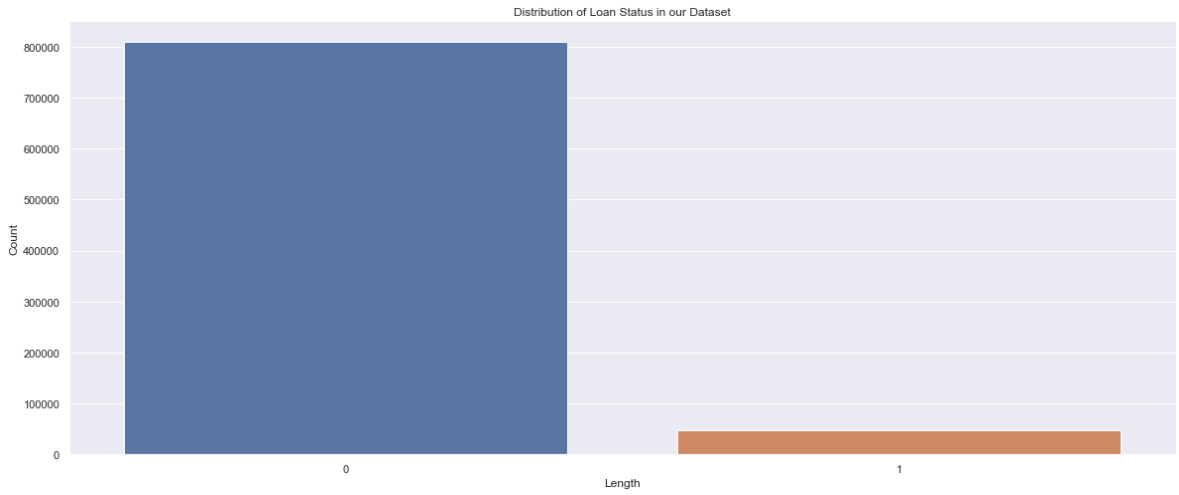
In [14]:

```
# Employment Length and Number of Loans
dff['emp_length'].value_counts().sort_values().plot(kind='barh',figsize=(18,8))
plt.title('Number of loans distributed by Employment Years',fontsize=20)
plt.xlabel('Number of loans',fontsize=15)
plt.ylabel('Years worked',fontsize=15);
```



In [15]:

```
#Distribution of the loan status values¶
m =dff['default_ind'].value_counts()
m = m.to_frame()
m.reset_index(inplace=True)
m.columns = ['Loan Status','Count']
plt.subplots(figsize=(20,8))
sns.barplot(y='Count', x='Loan Status', data=m)
plt.xlabel("Length")
plt.ylabel("Count")
plt.title("Distribution of Loan Status in our Dataset")
plt.show()
```



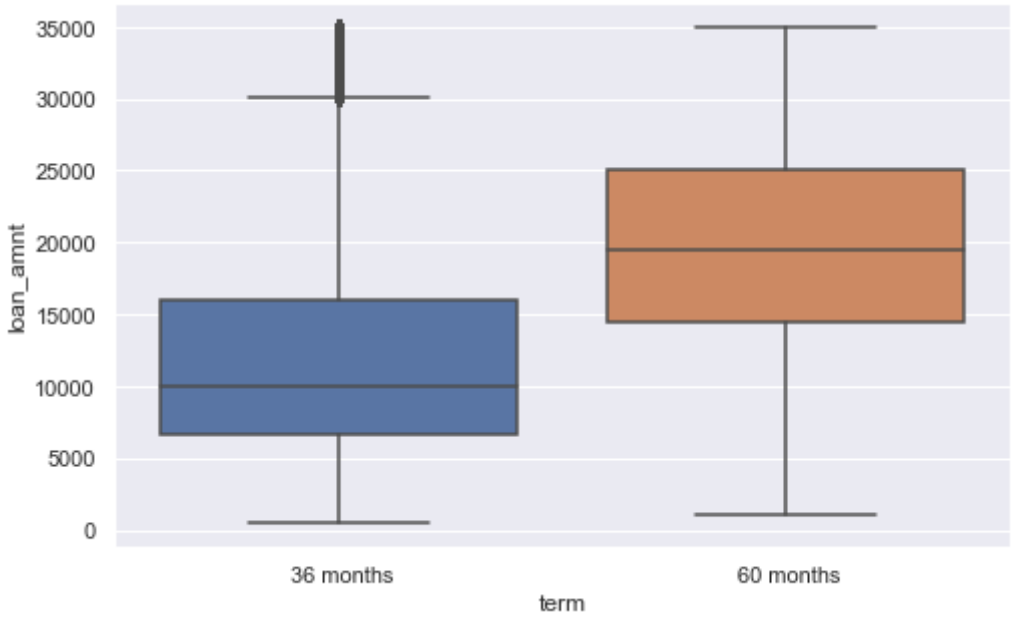
In [16]:

```
# Boxplot of Term and Loan Amount
print("Loan Amount Distribution BoxPlot")
plt.figure(figsize=(8,5))
sns.boxplot(x=df.term, y=dff.loan_amnt)
```

```
Loan Amount Distribution BoxPlot
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x5a889e7f60>
```

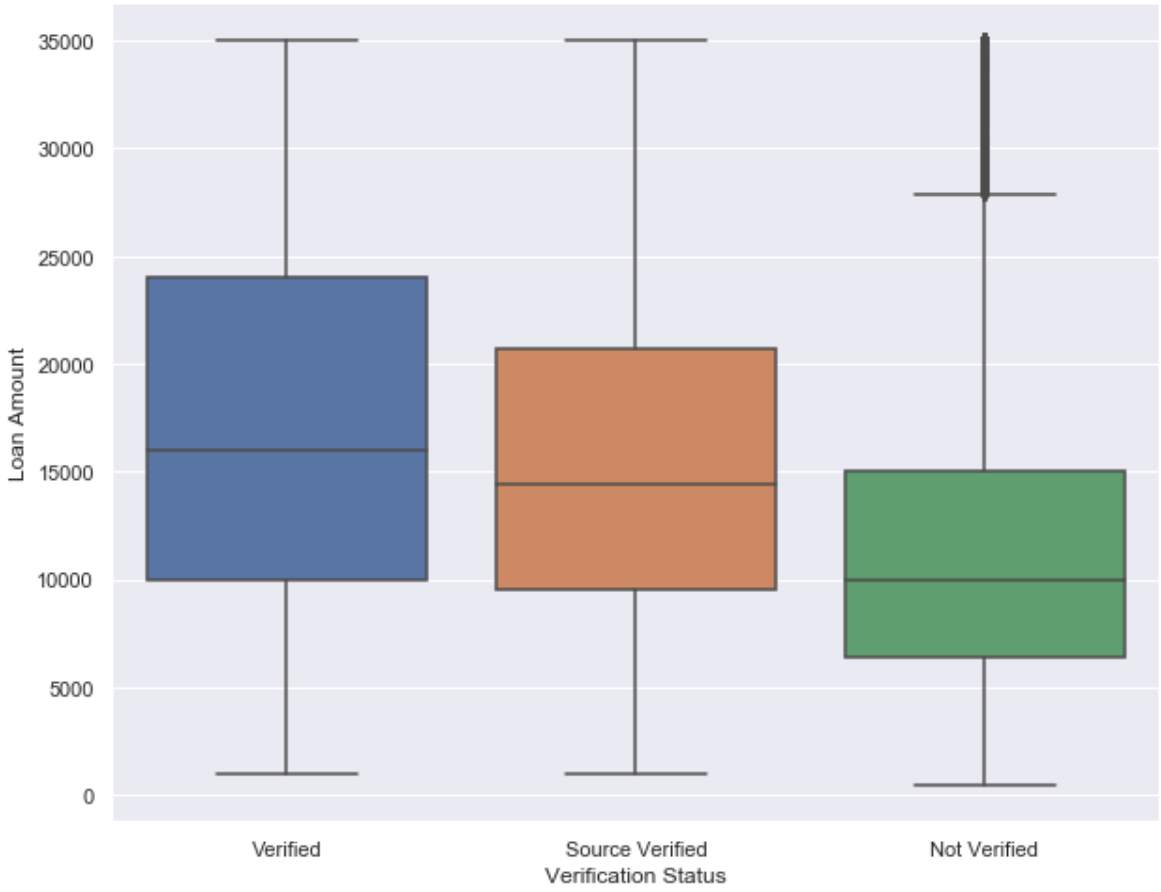


In [17]:

```
# Boxplot of Verification Status and Loan Amount
plt.figure(figsize=(10,8))
sns.boxplot(x=dff.verification_status, y=dff.loan_amnt)
plt.xlabel("Verification Status")
plt.ylabel("Loan Amount")
```

Out[17]:

```
Text(0, 0.5, 'Loan Amount')
```



In [18]:

```
# Crosstabulation of Purpose and Loan Status
purp_loan= ['purpose', 'default_ind']
cm = sns.light_palette("green", as_cmap=True)
cross=pd.crosstab(dff[purp_loan[0]], df[purp_loan[1]]).style.background_gradient(cmap = cm)
cross
cm
```

Out[18]:

```
<matplotlib.colors.LinearSegmentedColormap at 0x5a8850fba8>
```

In [19]:

```
#%%
#8.TRANSFORMATION FUNCTION DEFINITION
numerical = dff.columns[dff.dtypes == 'float64']
for i in numerical:
    if dff[i].min() > 0:
        transformed, lamb = boxcox(dff.loc[df[i].notnull(), i])
        if np.abs(1 - lamb) > 0.02:
            dff.loc[df[i].notnull(), i] = transformed
#%%
#One Hot Encoding
dff = pd.get_dummies(dff, drop_first=True)

list(dff)
```

Out[19]:

```
['loan_amnt',
 'int_rate',
 'installment',
 'annual_inc',
 'dti',
 'delinq_2yrs',
 'default_ind',
 'term_ 60 months',
 'grade_B',
 'grade_C',
 'grade_D',
 'grade_E',
 'grade_F',
 'grade_G',
 'sub_grade_A2',
 'sub_grade_A3',
 'sub_grade_A4',
 'sub_grade_A5',
```

In [20]:

```
#%%
#9.DATA SPLIT
traindata, testdata = train_test_split(dff, stratify=dff['default_ind'],test_size=.4, randc
testdata.reset_index(drop=True, inplace=True)
traindata.reset_index(drop=True, inplace=True)

#We'll now scale the data so that each column has a mean of zero and unit standard deviatio
sc = StandardScaler()
Xunb = traindata.drop('default_ind', axis=1)
yunb = traindata['default_ind']
numerical = Xunb.columns[(Xunb.dtypes == 'float64') | (Xunb.dtypes == 'int64')].tolist()
Xunb[numerical] = sc.fit_transform(Xunb[numerical])
#%%
##checking the shape of train data
yunb.shape
```

Out[20]:

```
(513581,)
```

In [21]:

```
#%%
#10.CREATE ROC
def createROC(models, X, y, Xte, yte):
    false_p, true_p = [], [] ##false postives and true positives

    for i in models.keys():  ##dict of models
        models[i].fit(X, y)

        fp, tp, threshold = roc_curve(yte, models[i].predict_proba(Xte)[:,1]) ##roc_curve f

        true_p.append(tp)
        false_p.append(fp)
    return true_p, false_p ##returning the true postive and false positive
```

#%%

""" Let's try some models on the train dataset With 3 fold cross validation. We are going to use the following 4 machine learning algorithms:

Linear Discriminant Analysis Multinomial Naive Bayes Random Forest (tree based model) Logistic Regression
"""

In [22]:

```
#%%
#11.CROSS VALIDATION
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB

models = {
          'LDA': LinearDiscriminantAnalysis(), #Remove # for crossvalidation in case of sys
          'MNB': MultinomialNB(),
          'RF': RandomForestClassifier(n_estimators=100),
          'LR': LogisticRegression(C=1)}

unbalset = {}
for i in models.keys():
    scores = cross_val_score(models[i], Xunb - np.min(Xunb) + 1,
                                     yunb, cv=3)
    unbalset[i] = scores
    print(i, scores, np.mean(scores))
#%%
#Creating Test
Xte = testdata.drop('default_ind', axis=1)
yte = testdata['default_ind']
numerical = Xte.columns[(Xte.dtypes == 'float64') | (Xte.dtypes == 'int64')].tolist()
Xte[numerical] = sc.fit_transform(Xte[numerical])
```

```
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:3
88: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:3
88: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:3
88: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")

LDA [0.94305908 0.94256775 0.94216469] 0.9425971737197795
MNB [0.94571103 0.94571624 0.9457104 ] 0.9457125555716598
RF [0.9454657  0.94544754 0.94545338] 0.9454555366722666

C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)

LR [0.94571687 0.94572792 0.94571624] 0.9457203440293603
```

In [23]:

```
#%%
#12.MODELLING
#Computing the ROC curves for the models and finding the true positive and false positives.
tp_unbalset, fp_unbalset = createROC(models, Xunb - np.min(Xunb) + 1, yunb, Xte - np.min(Xt
```

```
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:3
88: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
```

In [24]:

```
#%%
#13.FITTING
#Fitting LR to the test set.
model =  LogisticRegression(C=1)
model.fit(Xunb, yunb)
predict = model.predict(Xte) #prediction of Xte which can be used to test against yte (test

m = yte.to_frame()
m['default_ind'].value_counts()
```

```
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
```

Out[24]:

```
0    323801
1     18587
Name: default_ind, dtype: int64
```

#%%

"""

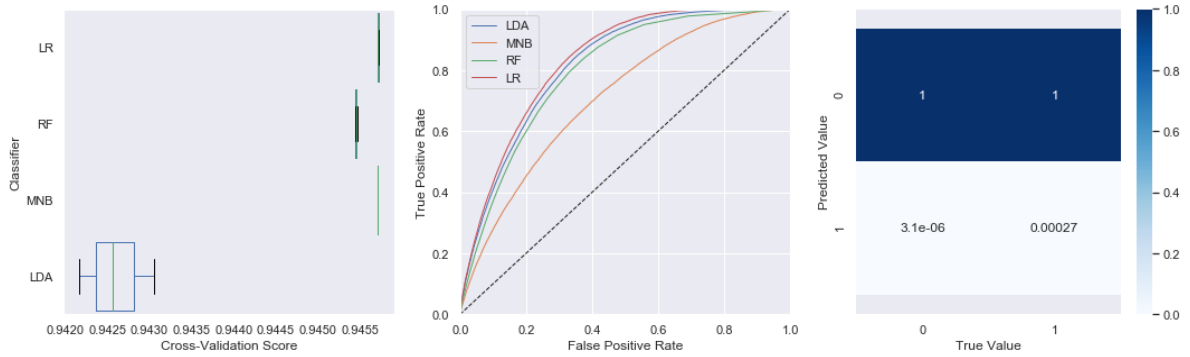
#We will now plot the cross-validation scores, ROC curves and confusion matrix of random forest model. X axis is the true value and Y axis is the predicted value. """

In [25]:

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18,5))

ax = pd.DataFrame(unbalset).boxplot(widths=(0.9,0.9,0.9,0.9), grid=False, vert=False, ax=ax
ax.set_ylabel('Classifier')
ax.set_xlabel('Cross-Validation Score')

for i in range(0, len(tp_unbalset)):
    axes[1].plot(fp_unbalset[i], tp_unbalset[i], lw=1)

axes[1].plot([0, 1], [0, 1], '--k', lw=1)
axes[1].legend(models.keys())
axes[1].set_ylabel('True Positive Rate')
axes[1].set_xlabel('False Positive Rate')
axes[1].set_xlim(0,1)
axes[1].set_ylim(0,1)

cm = confusion_matrix(yte, predict).T
cm = cm.astype('float')/cm.sum(axis=0)

ax = sns.heatmap(cm, annot=True, cmap='Blues', ax=axes[2]);
ax.set_xlabel('True Value')
ax.set_ylabel('Predicted Value')
ax.axis('equal')
```

Out[25]:

```
(0.0, 2.0, 2.0, 0.0)
```

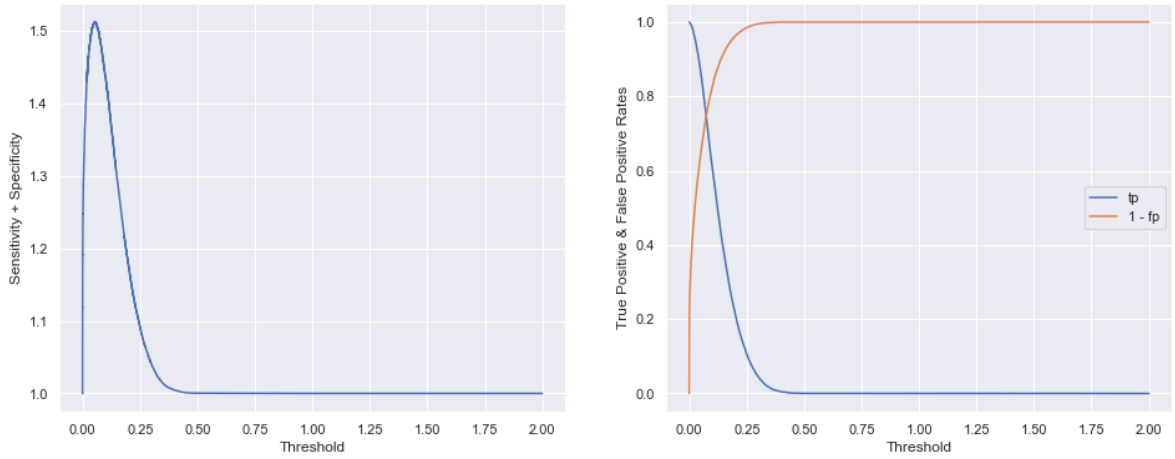


In [26]:

```
#%%
#Doing Again because of imbalance
fp, tp, threshold = roc_curve(yte, model.predict_proba(Xte)[:,1]) #getting false and true p
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16,6))

ax[0].plot(threshold, tp + (1 - fp))
ax[0].set_xlabel('Threshold')
ax[0].set_ylabel('Sensitivity + Specificity')

ax[1].plot(threshold, tp, label="tp")
ax[1].plot(threshold, 1 - fp, label="1 - fp")
ax[1].legend()
ax[1].set_xlabel('Threshold')
ax[1].set_ylabel('True Positive & False Positive Rates')
```

Out[26]:

```
Text(0, 0.5, 'True Positive & False Positive Rates')
```



In [27]:

```
#%%
##finding the optimal threshold for the model
function = tp + (1 - fp)
index = np.argmax(function)

optimal_threshold = threshold[np.argmax(function)]
print('optimal threshold:', optimal_threshold)
```

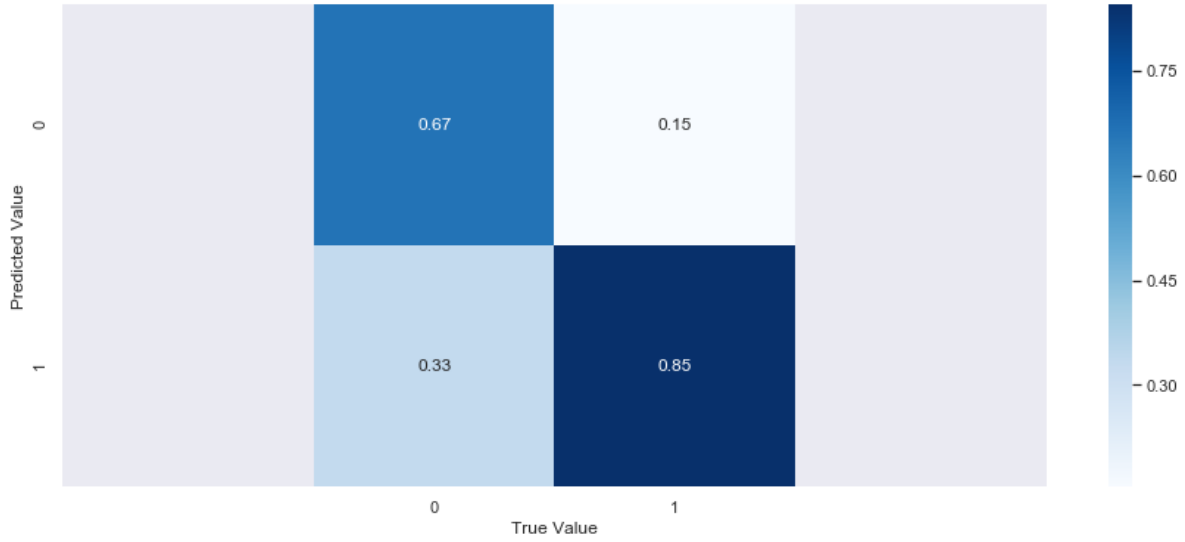
```
optimal threshold: 0.05411412432199966
```

In [28]:

```
#%%
#Now using this threshold for the model:

predict = model.predict_proba(Xte)[:,1]
predict = np.where(predict >= optimal_threshold, 1, 0)

fig, axes = plt.subplots(figsize=(15,6))

cm = confusion_matrix(yte, predict).T
cm = cm.astype('float')/cm.sum(axis=0)

ax = sns.heatmap(cm, annot=True, cmap='Blues');
ax.set_xlabel('True Value')
ax.set_ylabel('Predicted Value')
ax.axis('equal')
```

Out[28]:

```
(0.0, 2.0, 2.0, 0.0)
```



In [29]:

```
#%%
#Part2 : Balancing the training dataset and creating a new model
#Now we will try to use a balanced dataset with equal amount of zeroes and 1's. The followi

y_default = traindata[traindata['default_ind'] == 1]
n_paid = traindata[traindata['default_ind'] == 0].sample(n=len(y_default), random_state=17)

##creating a new dataframe for balanced set
data = y_default.append(n_paid)

##creating the independent and dependent array
Xbal = data.drop('default_ind', axis=1)
ybal = data['default_ind']
## scaling it again
numerical = Xbal.columns[(Xbal.dtypes == 'float64') | (Xbal.dtypes == 'int64')].tolist()
Xbal[numerical] = sc.fit_transform(Xbal[numerical])
```

In [30]:

```
#%%
#Training the model on the balanced set

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB

models = {'LDA': LinearDiscriminantAnalysis(),
          'MNB': MultinomialNB(),
          'RF': RandomForestClassifier(n_estimators=100),
          'LR': LogisticRegression(C=1)}

balset = {}
for i in models.keys():
    scores = cross_val_score(models[i], Xbal - np.min(Xbal) + 1,
                                    ybal, scoring='roc_auc', cv=3)
    balset[i] = scores
    print(i, scores, np.mean(scores))
```

```
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:3
88: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:3
88: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\discriminant_analysis.py:3
88: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")

LDA [0.82431604 0.82879743 0.82370319] 0.8256055520964383
MNB [0.70742412 0.70884694 0.70555438] 0.7072751486164114
RF [0.80772244 0.81259873 0.80792869] 0.8094166243193227

C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)

LR [0.82626127 0.83097701 0.82541921] 0.8275524940678701
```

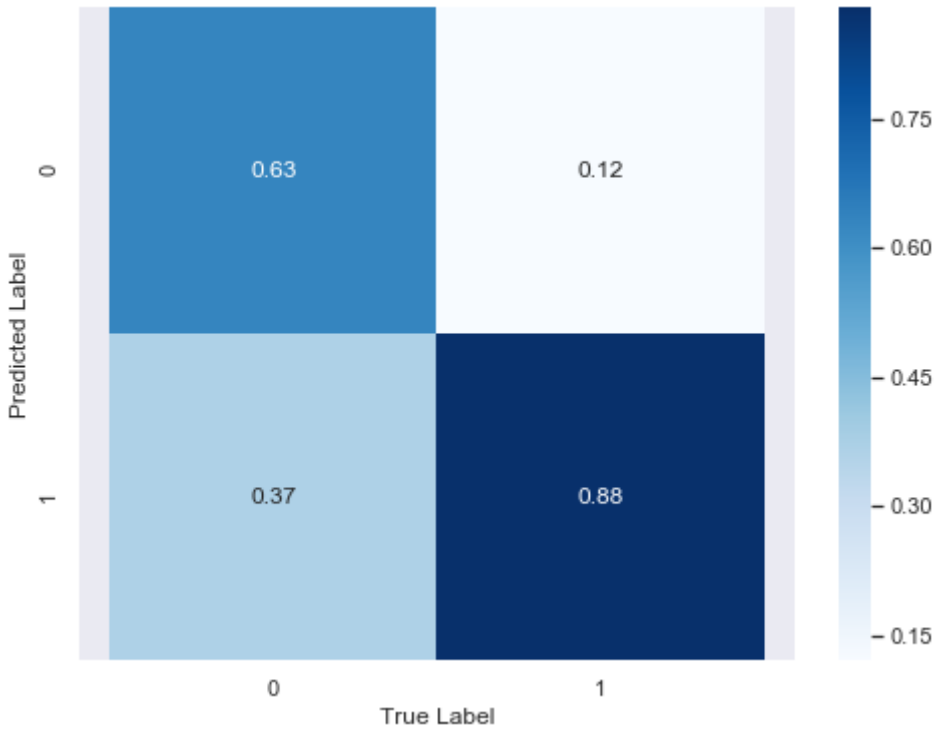
In [31]:

```
#%%
model = LogisticRegression(C=1)
model.fit(Xbal, ybal)
predict = model.predict(Xte)
predict = model.predict(Xte)
fig, axes = plt.subplots(figsize=(8,6))
cm = confusion_matrix(yte, predict).T
cm = cm.astype('float')/cm.sum(axis=0)
ax = sns.heatmap(cm, annot=True, cmap='Blues');
ax.set_xlabel('True Label')
ax.set_ylabel('Predicted Label')
ax.axis('equal')
```

```
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:4
33: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specif
y a solver to silence this warning.
  FutureWarning)
```

Out[31]:

```
(0.0, 2.0, 2.0, 0.0)
```



In [39]:

```
#%%

#14.CONFUSION MATRIX

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
cm
confusion_matrix(yte, predict)

results = confusion_matrix(yte, predict)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score :',accuracy_score(yte, predict))
print ('Report : ')
print (classification_report(yte, predict))
```

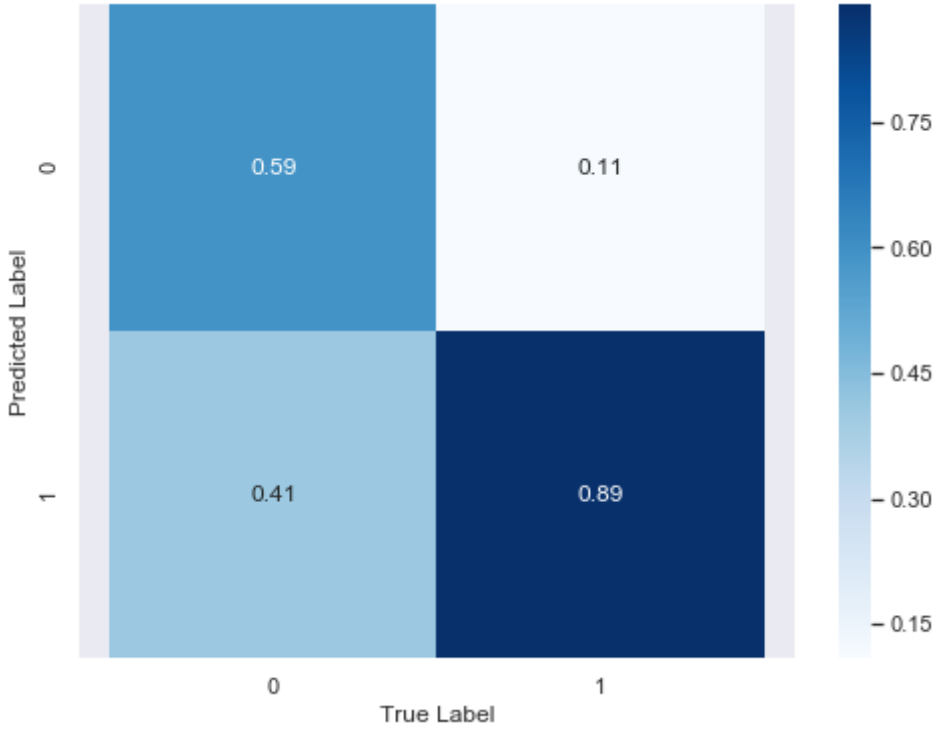
```
Confusion Matrix :
[[204205 119596]
 [  2256  16331]]
Accuracy Score : 0.6441113590429571
Report :
              precision    recall  f1-score   support

           0       0.99      0.63      0.77    323801
           1       0.12      0.88      0.21     18587

   micro avg       0.64      0.64      0.64    342388
   macro avg       0.55      0.75      0.49    342388
weighted avg       0.94      0.64      0.74    342388
```

In [40]:

```
#15.DOING WITH RANDOM FOREST
model = RandomForestClassifier(n_estimators=100)
model.fit(Xbal, ybal)
predict = model.predict(Xte)
predict = model.predict(Xte)
fig, axes = plt.subplots(figsize=(8,6))
cm = confusion_matrix(yte, predict).T
cm = cm.astype('float')/cm.sum(axis=0)
ax = sns.heatmap(cm, annot=True, cmap='Blues');
ax.set_xlabel('True Label')
ax.set_ylabel('Predicted Label')
ax.axis('equal')
```

Out[40]:

```
(0.0, 2.0, 2.0, 0.0)
```

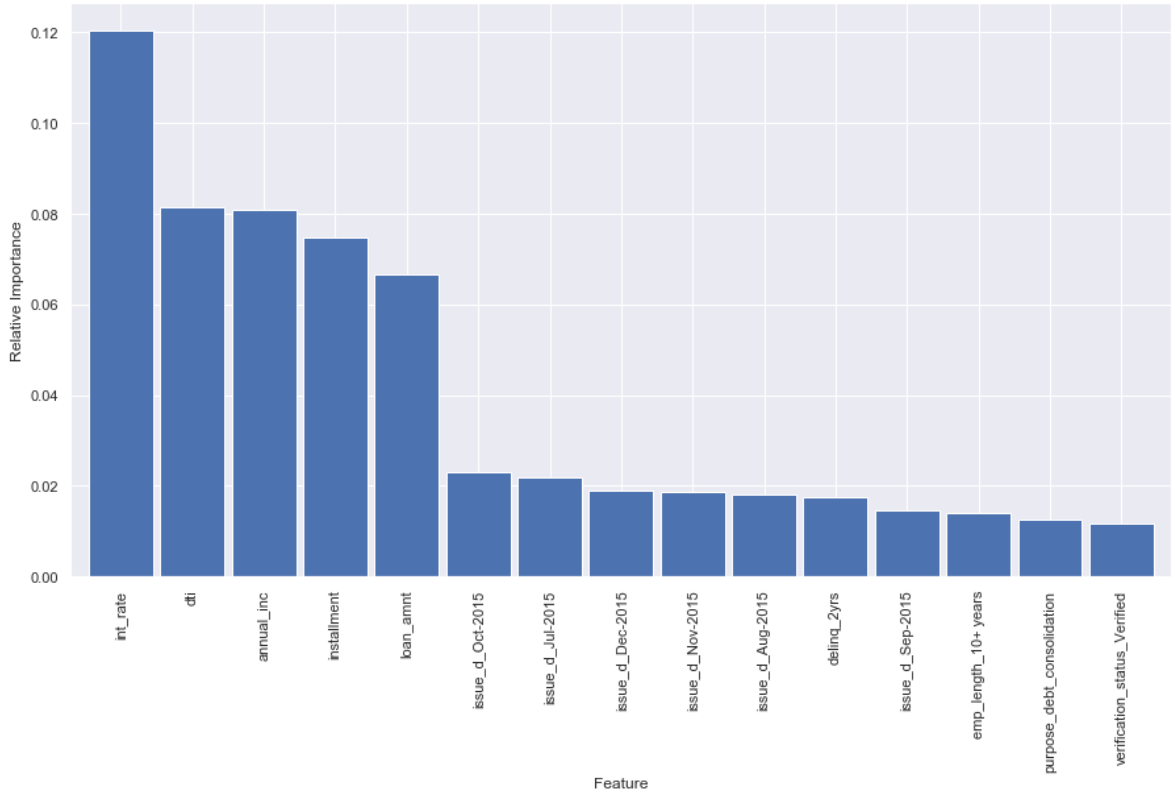


In [42]:

```
#%%
r = pd.DataFrame(columns=['Feature','Importance'])
ncomp = 15
r['Feature'] = feat_labels = Xbal.columns
r['Importance'] = model.feature_importances_
r.set_index(r['Feature'], inplace=True)
ax = r.sort_values('Importance', ascending=False)[:ncomp].plot.bar(width=0.9, legend=False,
ax.set_ylabel('Relative Importance')
```

Out[42]:

```
Text(0, 0.5, 'Relative Importance')
```



In [43]:

```
#%%

#16.CONFUSION MATRIX FOR RANDOM FOREST

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
cm
confusion_matrix(yte, predict)

results = confusion_matrix(yte, predict)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score :',accuracy_score(yte, predict))
print ('Report : ')
print (classification_report(yte, predict))
```

```
Confusion Matrix :
[[192377 131424]
 [  2034  16553]]
Accuracy Score : 0.610214143019031
Report :
              precision    recall  f1-score   support

           0       0.99      0.59      0.74    323801
           1       0.11      0.89      0.20     18587

   micro avg       0.61      0.61      0.61    342388
   macro avg       0.55      0.74      0.47    342388
weighted avg       0.94      0.61      0.71    342388
```