In [544]:
```
import pandas as pd
import numpy as np
```

In [545]:
```
pl=pd.read_csv('prosperLoanData.csv')
```

In [546]:
```
pd.set_option("display.max_columns", len(pl.columns))
pl.sample(5)
```

Out[546]:

	ListingKey	ListingNumber	ListingCreationDate	CreditGrade	Ter
21449	4BF53419418329010686881	317493	2008-04-24 12:19:43.413000000	AA	36
28491	7AA3353875603235900D6AC	555491	2012-01-30 16:45:42.943000000	NaN	36
23886	CB65356128839372776E7F6	654950	2012-10-16 12:10:25.363000000	NaN	36
84737	3D953598941100644F6EE0F	1116449	2014-01-04 10:19:59.303000000	NaN	60
113765	DE393586895011134F40215	868954	2013-08-13 11:46:15.580000000	NaN	60

In [547]:
```
pl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 81 columns):
ListingKey                             113937 non-null object
ListingNumber                          113937 non-null int64
ListingCreationDate                    113937 non-null object
CreditGrade                            28953 non-null object
Term                                   113937 non-null int64
LoanStatus                             113937 non-null object
ClosedDate                             55089 non-null object
BorrowerAPR                            113912 non-null float64
BorrowerRate                           113937 non-null float64
LenderYield                            113937 non-null float64
EstimatedEffectiveYield                84853 non-null float64
EstimatedLoss                          84853 non-null float64
EstimatedReturn                        84853 non-null float64
ProsperRating (numeric)                84853 non-null float64
ProsperRating (Alpha)                  84853 non-null object
ProsperScore                           84853 non-null float64
ListingCategory (numeric)              113937 non-null int64
BorrowerState                          108422 non-null object
Occupation                             110349 non-null object
EmploymentStatus                       111682 non-null object
EmploymentStatusDuration               106312 non-null float64
IsBorrowerHomeowner                    113937 non-null bool
CurrentlyInGroup                       113937 non-null bool
GroupKey                               13341 non-null object
DateCreditPulled                       113937 non-null object
CreditScoreRangeLower                  113346 non-null float64
CreditScoreRangeUpper                  113346 non-null float64
FirstRecordedCreditLine                113240 non-null object
CurrentCreditLines                     106333 non-null float64
OpenCreditLines                        106333 non-null float64
TotalCreditLinespast7years             113240 non-null float64
OpenRevolvingAccounts                  113937 non-null int64
OpenRevolvingMonthlyPayment            113937 non-null float64
InquiriesLast6Months                   113240 non-null float64
TotalInquiries                         112778 non-null float64
CurrentDelinquencies                   113240 non-null float64
AmountDelinquent                       106315 non-null float64
DelinquenciesLast7Years                112947 non-null float64
PublicRecordsLast10Years               113240 non-null float64
PublicRecordsLast12Months              106333 non-null float64
RevolvingCreditBalance                 106333 non-null float64
BankcardUtilization                    106333 non-null float64
AvailableBankcardCredit                106393 non-null float64
TotalTrades                            106393 non-null float64
TradesNeverDelinquent (percentage)     106393 non-null float64
TradesOpenedLast6Months                106393 non-null float64
DebtToIncomeRatio                      105383 non-null float64
IncomeRange                            113937 non-null object
IncomeVerifiable                       113937 non-null bool
StatedMonthlyIncome                    113937 non-null float64
LoanKey                                113937 non-null object
TotalProsperLoans                      22085 non-null float64
```

```
TotalProsperPaymentsBilled            22085 non-null float64
OnTimeProsperPayments                 22085 non-null float64
ProsperPaymentsLessThanOneMonthLate   22085 non-null float64
ProsperPaymentsOneMonthPlusLate       22085 non-null float64
ProsperPrincipalBorrowed              22085 non-null float64
ProsperPrincipalOutstanding           22085 non-null float64
ScorexChangeAtTimeOfListing           18928 non-null float64
LoanCurrentDaysDelinquent             113937 non-null int64
LoanFirstDefaultedCycleNumber         16952 non-null float64
LoanMonthsSinceOrigination            113937 non-null int64
LoanNumber                            113937 non-null int64
LoanOriginalAmount                    113937 non-null int64
LoanOriginationDate                   113937 non-null object
LoanOriginationQuarter                113937 non-null object
MemberKey                             113937 non-null object
MonthlyLoanPayment                    113937 non-null float64
LP_CustomerPayments                   113937 non-null float64
LP_CustomerPrincipalPayments          113937 non-null float64
LP_InterestandFees                    113937 non-null float64
LP_ServiceFees                        113937 non-null float64
LP_CollectionFees                     113937 non-null float64
LP_GrossPrincipalLoss                 113937 non-null float64
LP_NetPrincipalLoss                   113937 non-null float64
LP_NonPrincipalRecoverypayments       113937 non-null float64
PercentFunded                         113937 non-null float64
Recommendations                       113937 non-null int64
InvestmentFromFriendsCount            113937 non-null int64
InvestmentFromFriendsAmount           113937 non-null float64
Investors                             113937 non-null int64
dtypes: bool(3), float64(50), int64(11), object(17)
memory usage: 68.1+ MB
```

In [548]:
```
missing= pl.isnull().sum()
missing
```

Out[548]:
```
ListingKey                                  0
ListingNumber                               0
ListingCreationDate                         0
CreditGrade                             84984
Term                                        0
LoanStatus                                  0
ClosedDate                              58848
BorrowerAPR                                25
BorrowerRate                                0
LenderYield                                 0
EstimatedEffectiveYield                 29084
EstimatedLoss                           29084
EstimatedReturn                         29084
ProsperRating (numeric)                 29084
ProsperRating (Alpha)                   29084
ProsperScore                            29084
ListingCategory (numeric)                   0
BorrowerState                            5515
Occupation                               3588
EmploymentStatus                         2255
EmploymentStatusDuration                 7625
IsBorrowerHomeowner                         0
CurrentlyInGroup                            0
GroupKey                               100596
DateCreditPulled                            0
CreditScoreRangeLower                     591
CreditScoreRangeUpper                     591
FirstRecordedCreditLine                   697
CurrentCreditLines                       7604
OpenCreditLines                          7604
                                        ...
TotalProsperLoans                       91852
TotalProsperPaymentsBilled              91852
OnTimeProsperPayments                   91852
ProsperPaymentsLessThanOneMonthLate     91852
ProsperPaymentsOneMonthPlusLate         91852
ProsperPrincipalBorrowed                91852
ProsperPrincipalOutstanding             91852
ScorexChangeAtTimeOfListing             95009
LoanCurrentDaysDelinquent                   0
LoanFirstDefaultedCycleNumber           96985
LoanMonthsSinceOrigination                  0
LoanNumber                                  0
LoanOriginalAmount                          0
LoanOriginationDate                         0
LoanOriginationQuarter                      0
MemberKey                                   0
MonthlyLoanPayment                          0
LP_CustomerPayments                         0
LP_CustomerPrincipalPayments                0
LP_InterestandFees                          0
LP_ServiceFees                              0
LP_CollectionFees                           0
LP_GrossPrincipalLoss                       0
```

```
LP_NetPrincipalLoss                    0
LP_NonPrincipalRecoverypayments        0
PercentFunded                          0
Recommendations                        0
InvestmentFromFriendsCount             0
InvestmentFromFriendsAmount            0
Investors                              0
dtype: int64
```

Dropping unnecessary columns of the data and the columns which have more than 80% of missing values

In [549]:
```
to_drop = ['ListingKey','ListingNumber','ListingCreationDate','GroupKey','LoanKey
            'MemberKey','LoanFirstDefaultedCycleNumber','LoanOriginationDate','Da
           'TotalProsperLoans','TotalProsperPaymentsBilled','OnTimeProsperPayment
           'ProsperPaymentsOneMonthPlusLate','ProsperPrincipalBorrowed','ProsperP
           'ScorexChangeAtTimeOfListing']
```

In [550]:
```
pl.drop(to_drop,inplace=True, axis=1)
```

In [551]:
```
# Extracting the numerical features of the data
numeric=pl.select_dtypes(exclude=['object'])
```

In [552]:
```
# See the numeric data
numeric.sample(5)
```

Out[552]:

	Term	BorrowerAPR	BorrowerRate	LenderYield	EstimatedEffectiveYield	Estimated
97188	36	0.33051	0.2909	0.2809	0.26810	0.1425
60629	36	0.06726	0.0605	0.0505	0.05000	0.0074
49773	36	0.16215	0.1550	0.1250	NaN	NaN
42600	36	0.09030	0.0769	0.0669	0.06546	0.0174
47738	36	0.24246	0.2049	0.1949	0.19040	0.0890

In [553]:
```
numeric.describe()
```

Out[553]:

	Term	BorrowerAPR	BorrowerRate	LenderYield	EstimatedEffectiveYie
count	113937.000000	113912.000000	113937.000000	113937.000000	84853.000000
mean	40.830248	0.218828	0.192764	0.182701	0.168661
std	10.436212	0.080364	0.074818	0.074516	0.068467
min	12.000000	0.006530	0.000000	-0.010000	-0.182700
25%	36.000000	0.156290	0.134000	0.124200	0.115670
50%	36.000000	0.209760	0.184000	0.173000	0.161500
75%	36.000000	0.283810	0.250000	0.240000	0.224300
max	60.000000	0.512290	0.497500	0.492500	0.319900

In [554]:
```
# Numeric data information

numeric.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 53 columns):
Term                                  113937 non-null int64
BorrowerAPR                           113912 non-null float64
BorrowerRate                          113937 non-null float64
LenderYield                           113937 non-null float64
EstimatedEffectiveYield               84853 non-null float64
EstimatedLoss                         84853 non-null float64
EstimatedReturn                       84853 non-null float64
ProsperRating (numeric)               84853 non-null float64
ProsperScore                          84853 non-null float64
ListingCategory (numeric)             113937 non-null int64
EmploymentStatusDuration              106312 non-null float64
IsBorrowerHomeowner                   113937 non-null bool
CurrentlyInGroup                      113937 non-null bool
CreditScoreRangeLower                 113346 non-null float64
CreditScoreRangeUpper                 113346 non-null float64
CurrentCreditLines                    106333 non-null float64
OpenCreditLines                       106333 non-null float64
TotalCreditLinespast7years            113240 non-null float64
OpenRevolvingAccounts                 113937 non-null int64
OpenRevolvingMonthlyPayment           113937 non-null float64
InquiriesLast6Months                  113240 non-null float64
TotalInquiries                        112778 non-null float64
CurrentDelinquencies                  113240 non-null float64
AmountDelinquent                      106315 non-null float64
DelinquenciesLast7Years               112947 non-null float64
PublicRecordsLast10Years              113240 non-null float64
PublicRecordsLast12Months             106333 non-null float64
RevolvingCreditBalance                106333 non-null float64
BankcardUtilization                   106333 non-null float64
AvailableBankcardCredit               106393 non-null float64
TotalTrades                           106393 non-null float64
TradesNeverDelinquent (percentage)    106393 non-null float64
TradesOpenedLast6Months               106393 non-null float64
DebtToIncomeRatio                     105383 non-null float64
IncomeVerifiable                      113937 non-null bool
StatedMonthlyIncome                   113937 non-null float64
LoanCurrentDaysDelinquent             113937 non-null int64
LoanMonthsSinceOrigination            113937 non-null int64
LoanOriginalAmount                    113937 non-null int64
MonthlyLoanPayment                    113937 non-null float64
LP_CustomerPayments                   113937 non-null float64
LP_CustomerPrincipalPayments          113937 non-null float64
LP_InterestandFees                    113937 non-null float64
LP_ServiceFees                        113937 non-null float64
LP_CollectionFees                     113937 non-null float64
LP_GrossPrincipalLoss                 113937 non-null float64
LP_NetPrincipalLoss                   113937 non-null float64
LP_NonPrincipalRecoverypayments       113937 non-null float64
PercentFunded                         113937 non-null float64
Recommendations                       113937 non-null int64
```

```
InvestmentFromFriendsCount          113937 non-null int64
InvestmentFromFriendsAmount         113937 non-null float64
Investors                           113937 non-null int64
dtypes: bool(3), float64(41), int64(9)
memory usage: 43.8 MB
```

In [555]:
```
# Converting boolean numeric datatype to categorical datatype
convert_bool=['IsBorrowerHomeowner','CurrentlyInGroup',
              'IncomeVerifiable']
pl[convert_bool]=pl[convert_bool].astype('object')
```

In [556]:
```
pl.IsBorrowerHomeowner.dtype
```

Out[556]: dtype('O')

In [557]:
```
from sklearn.preprocessing import Imputer
my_imputer=Imputer()
p_nf= my_imputer.fit_transform(numeric)
```

In [558]:
```
columns=numeric.columns
```

In [559]:
```
p_nf=pd.DataFrame(p_nf,columns=columns)
```

In [560]:
```
p_nf=p_nf.join(pl['LoanStatus'])
```

Since our aim is to predict the defaulted LoanStatsus class, we are not concerned with the variable type 'current', hence we are deleting the observations corresponding to 'current' LoanStatus

In [561]:
```
p_nf=p_nf[p_nf['LoanStatus']!='Current']
```

In [562]:
```
p_nf.describe()
```

Out[562]:

	Term	BorrowerAPR	BorrowerRate	LenderYield	EstimatedEffectiveYield
count	57361.000000	57361.000000	57361.000000	57361.000000	57361.000000
mean	37.199142	0.223846	0.201634	0.191509	0.173659
std	7.649302	0.087876	0.080952	0.080409	0.056742
min	12.000000	0.006530	0.000000	-0.010000	-0.182700
25%	36.000000	0.152110	0.136400	0.125900	0.168661
50%	36.000000	0.217390	0.198000	0.185000	0.168661
75%	36.000000	0.295540	0.269900	0.259900	0.175700
max	60.000000	0.512290	0.497500	0.492500	0.319900

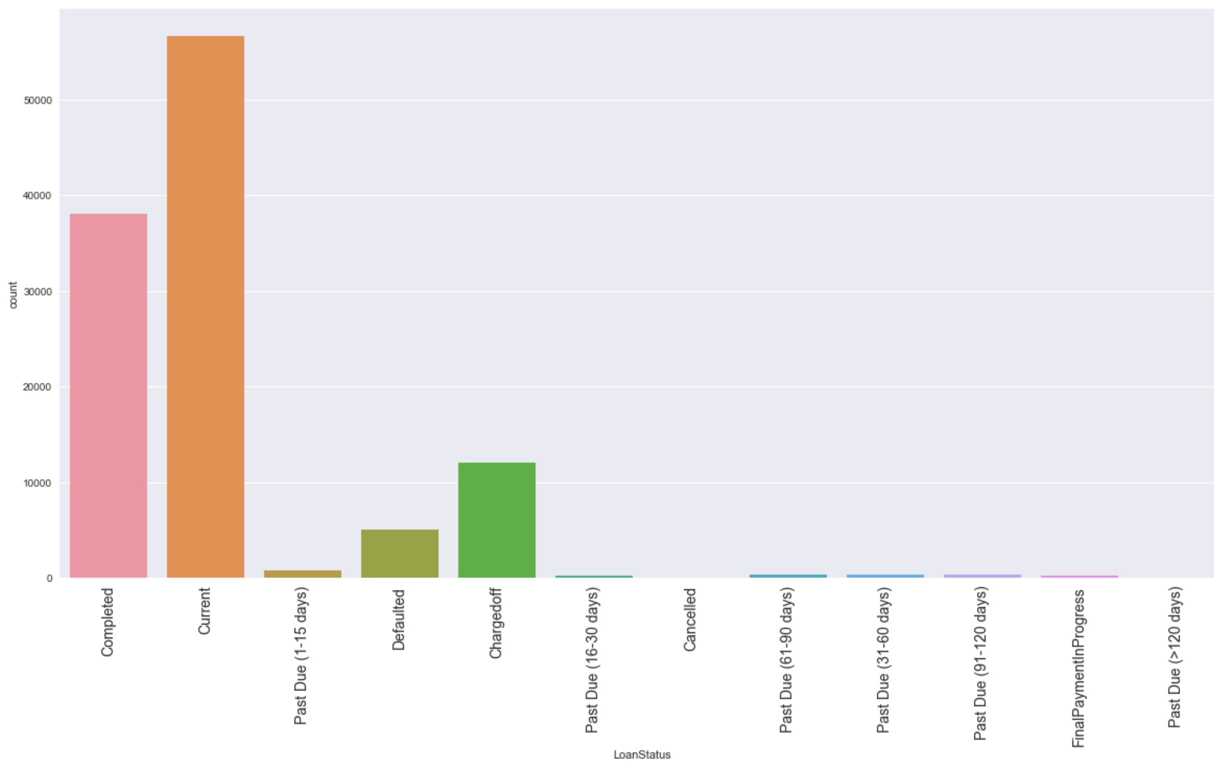
Visualization Section

Target Variable : Loan Status

In [563]:
```
ymulti=pl['LoanStatus']
```

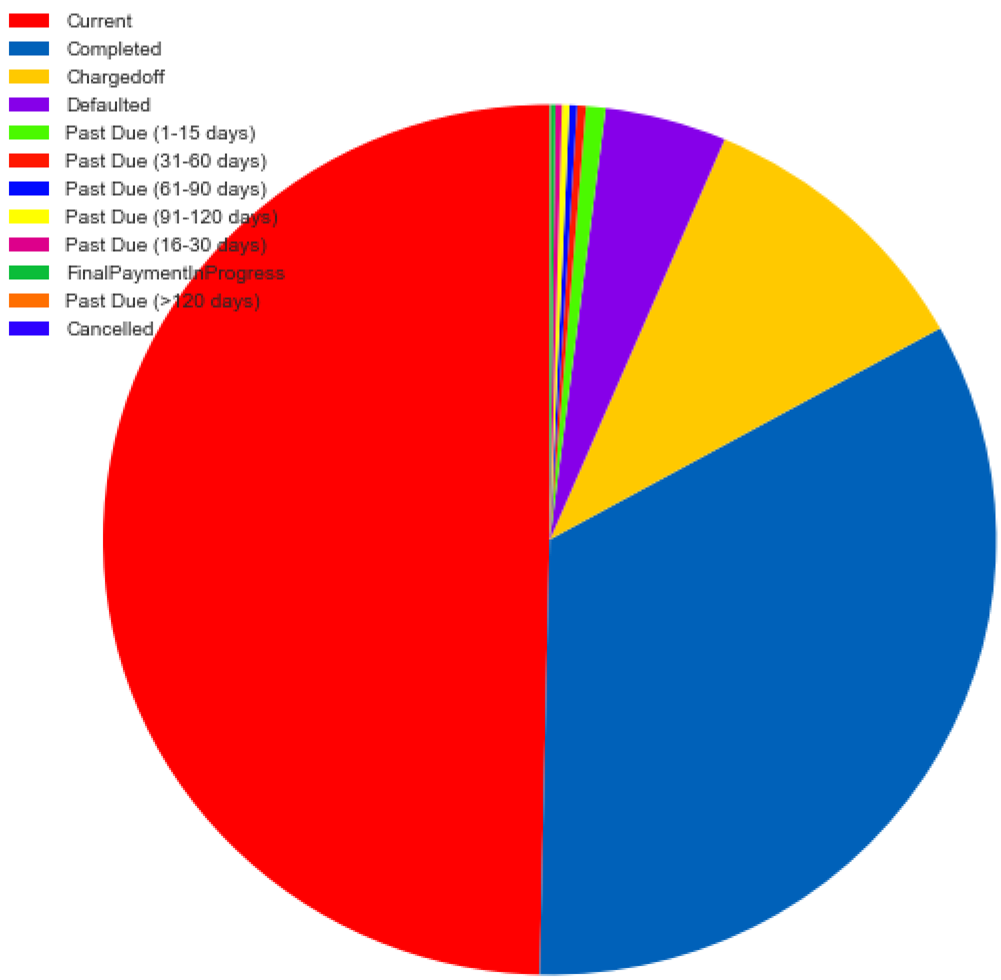
In [564]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
fig = plt.figure(figsize=(20, 10))

ax1 = fig.add_subplot()
plt.xticks(fontsize=14, rotation=90)
sns.countplot(ymulti)
```

Out[564]: <matplotlib.axes._subplots.AxesSubplot at 0x242be757978>



In [565]:
```
#Plot a histogram of frequencies
from random import shuffle


slices = [1,2,3] * 4 + [20, 25, 30] * 2
shuffle(slices)

fig = plt.figure(figsize=[10, 10])
ax = fig.add_subplot(111)

cmap = plt.cm.prism
colors = cmap(np.linspace(0., 1., len(slices)))


sizes= pl.LoanStatus.value_counts()
labels = [r'Current', r'Completed', r'Chargedoff', r'Defaulted',
          r'Past Due (1-15 days)', r'Past Due (31-60 days)', r'Past Due (61-90 da
          r'Past Due (16-30 days)', r'FinalPaymentInProgress', r'Past Due (>120 d
patches, texts = plt.pie(sizes, colors=colors, startangle=90)
plt.legend(patches, labels, loc="best")
print(sizes)
```

```
Current                   56576
Completed                 38074
Chargedoff                11992
Defaulted                  5018
Past Due (1-15 days)        806
Past Due (31-60 days)       363
Past Due (61-90 days)       313
Past Due (91-120 days)      304
Past Due (16-30 days)       265
FinalPaymentInProgress      205
Past Due (>120 days)         16
Cancelled                     5
Name: LoanStatus, dtype: int64
```



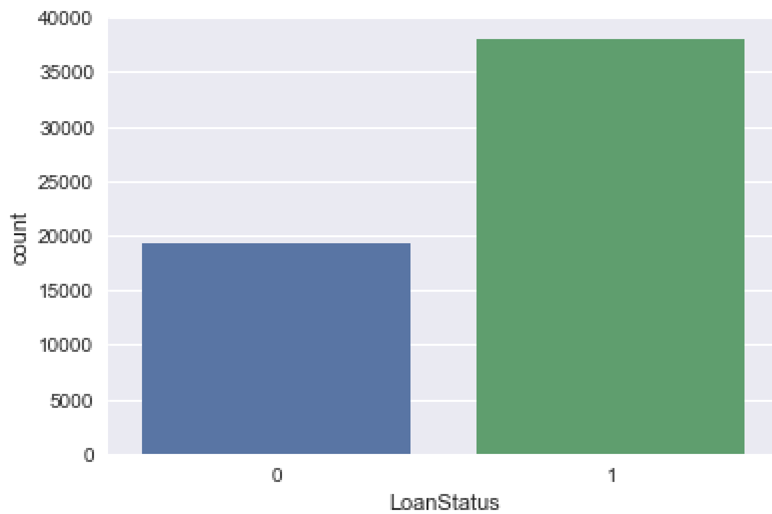
From the above visualization, Current loan status ranks the highest with almost nearly half of the dataset, followed by 'Completed' loan status with almost 39000 counts. Chargedoff and default follows the ranking list and the rest are fractionally distributed.

Binary transformation of Loan Status

In [566]:
```
data_new=(p_nf['LoanStatus']=='Completed').astype(int)
```

In [567]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
fig = plt.figure()

ax1 = fig.add_subplot(111)

sns.countplot(data_new)
```

Out[567]: <matplotlib.axes._subplots.AxesSubplot at 0x24297363940>



The current status of loan is removed from the vizualization as that is not part of the goals of the project. We only focus on historical data to build a predictive model.Further, all other classes of loan status are divided into binary format, categorizing them into good loans vs. bad loans. The one status is Completed which is a good loan and the other class has the mixture of all other sub classees such as Chargedoff,Defaulted,Past Due (1-15 days), Past Due (31-60 days), Past Due (61-90 days), Past Due (91-120 days), Past Due (16-30 days), FinalPaymentInProgress, Past Due (>120 days), Cancelled and these are bad loans.

From the above bar chart, we can conclude that the majority class is of 1 i.e completed and the other class of 0 has about half the data of the completed class. Our sole focus is on the class '0' which is bad loans and is considered to be a true class for our analysis.

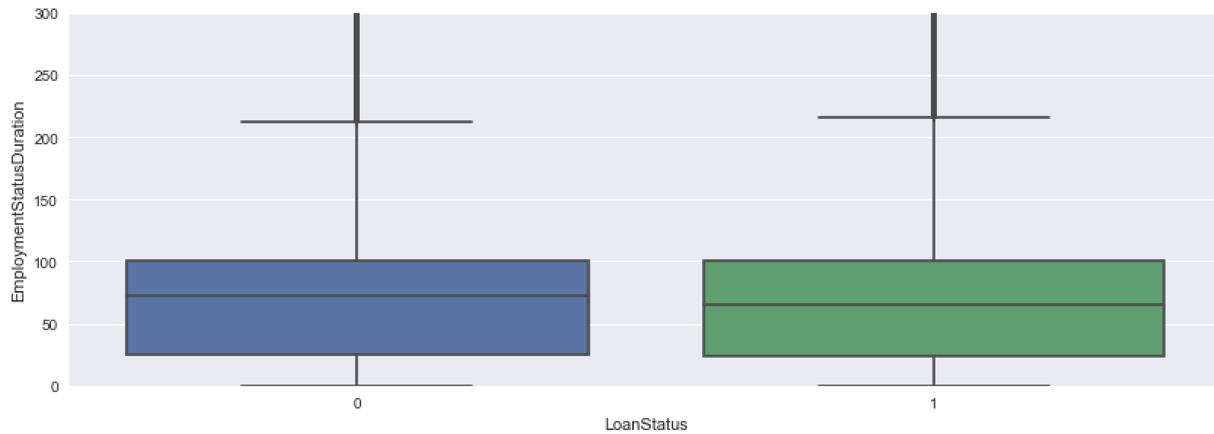
Exploring Numeric information of the data

In [568]:
```
p_nf["LoanStatus"]=(p_nf['LoanStatus']=='Completed').astype(int)
```

Employment status duration

In [569]:
```
fig = plt.figure(figsize=(30, 10))

ax2 = fig.add_subplot(222)
sns.boxplot(x="LoanStatus", y="EmploymentStatusDuration", data=p_nf).set_ylim([0,
```

Out[569]: (0, 300)

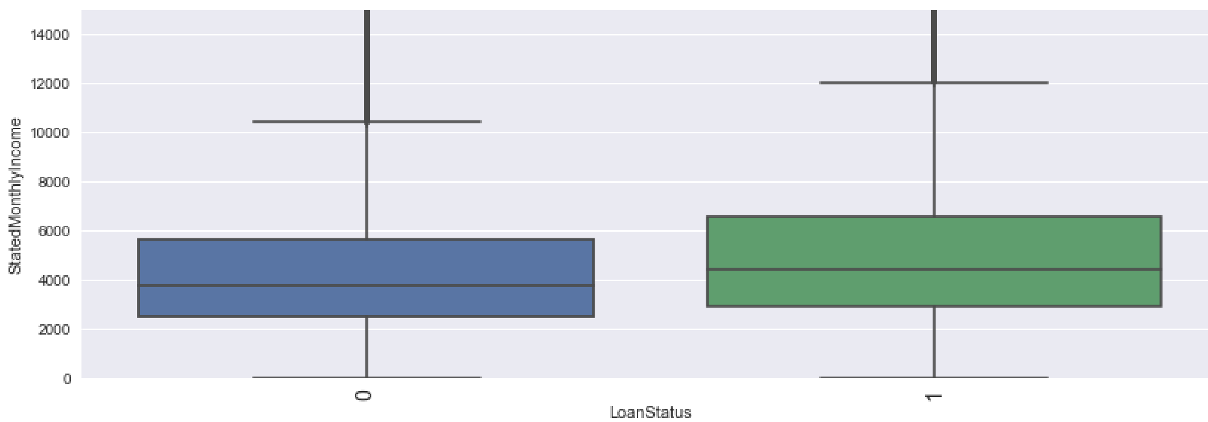


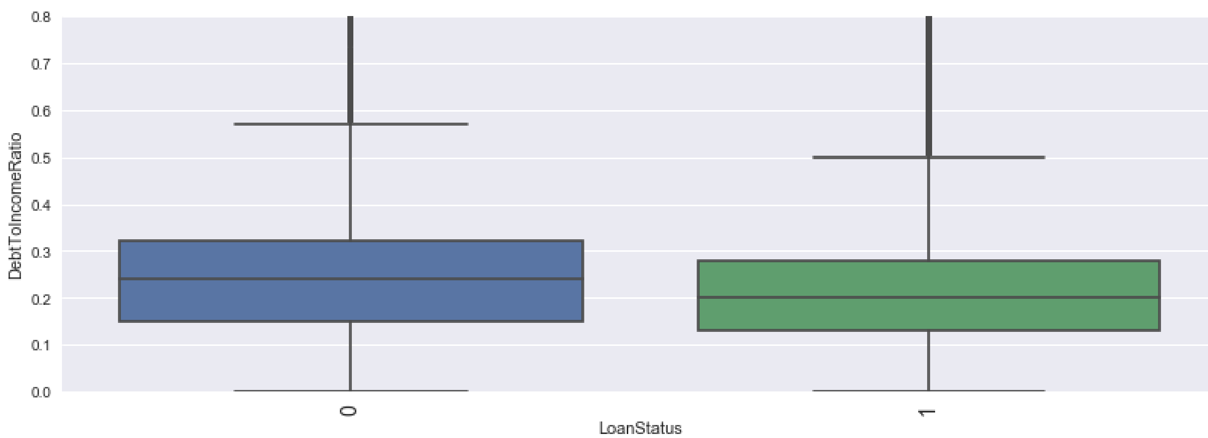
The box plot shows the median status duration falls around 50 for both categories of loans. The lower and upper qaurtile including the range also dosen't really differ for Loan Status. It is quite evident from this plot that there is hardly a relationship between EmploymentStatusDuration and loan default.

Income metrics - Stated Monthly Income and Debt to Income Ratio

In [570]:
```
# Box plot for Stated Montly Income variable
fig = plt.figure(figsize=(30, 10))

ax1 = fig.add_subplot(221)
sns.boxplot(x="LoanStatus", y="StatedMonthlyIncome", data=p_nf).set_ylim([0,15000
plt.xticks(fontsize=14, rotation=90)

# Box plot for Debt to Income ratio variable

fig = plt.figure(figsize=(30, 10))

ax2 = fig.add_subplot(222)
sns.boxplot(x="LoanStatus", y="DebtToIncomeRatio", data=p_nf).set_ylim([0,0.8])
plt.xticks(fontsize=14, rotation=90)
```

Out[570]:
```
(array([0, 1]), <a list of 2 Text xticklabel objects>)
```



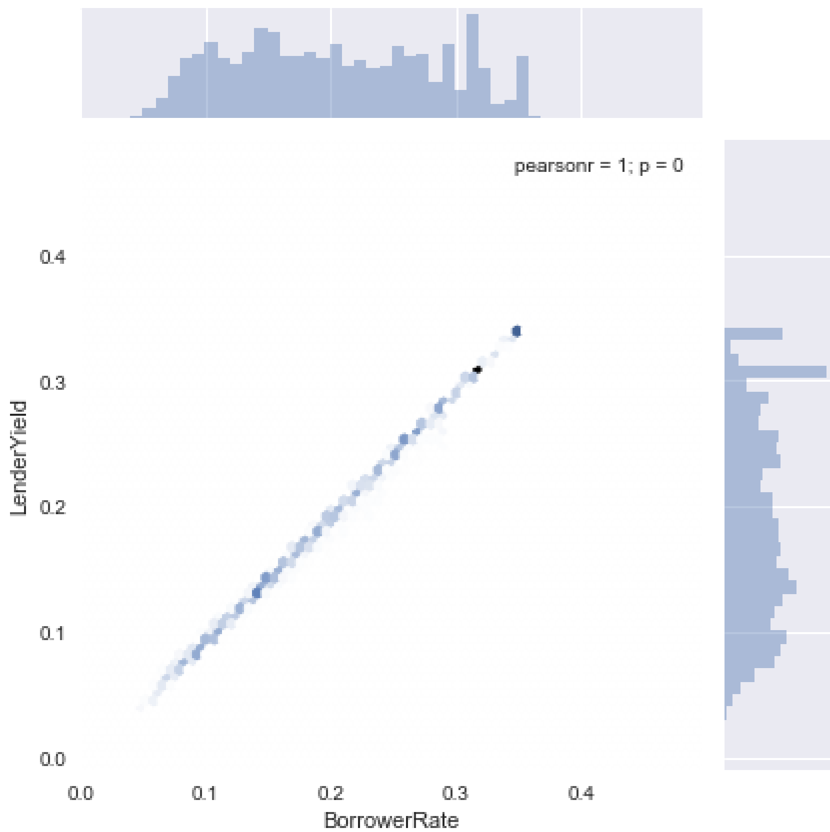


We observed the state monthly income and the debt to income ratio have a relationship with default i.e People with higher stated incomes defaulted less often than those with lower incomes. The range for monthly income for good loan status is also higher comparitively. People with higher debt to income ratio have more defaulted loans.

Borrower Rate and Lender Yield

In [571]:
```
fig = plt.figure(figsize=(50, 30))

sns.jointplot(x='BorrowerRate',y='LenderYield',kind='hex', data=p_nf)
```

Out[571]:
```
<seaborn.axisgrid.JointGrid at 0x242bde76f98>

<matplotlib.figure.Figure at 0x242bde767f0>
```

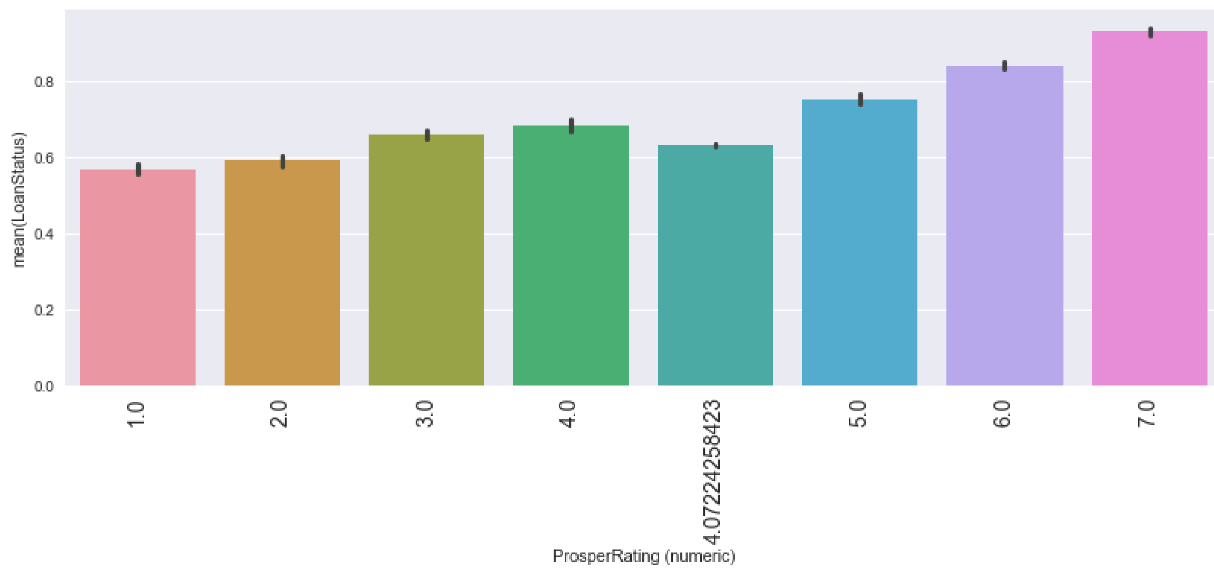


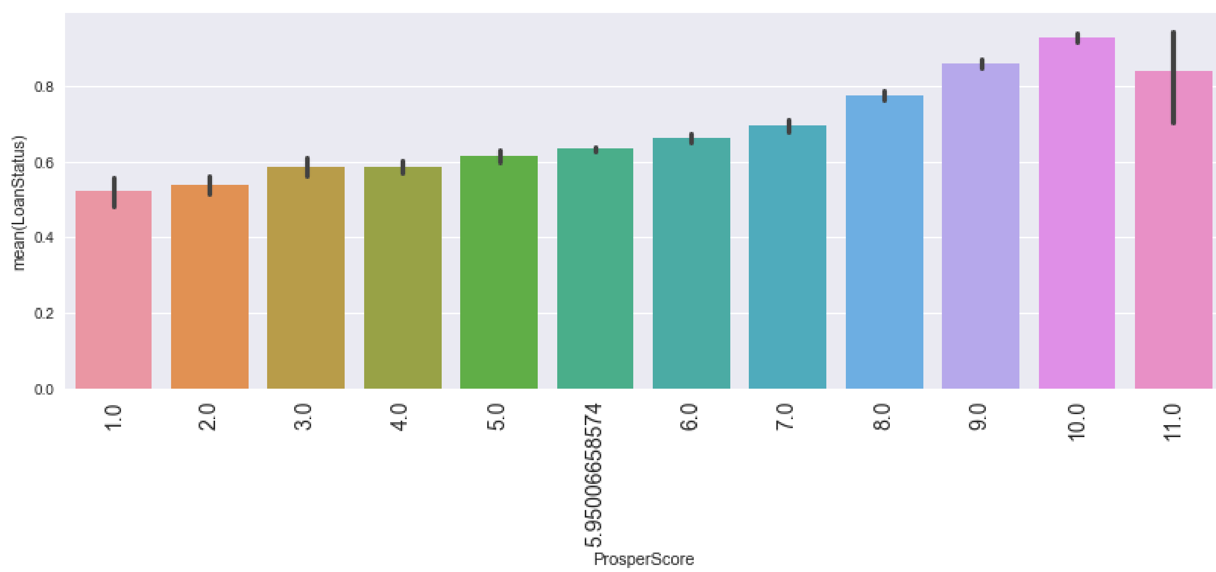
The above jointplot shows almost a perfect positive correlation between LenderYield and BorrowerRate. Higher the rate, higher is the lender yield. This makes sense to the data.

Credit scores: Prosper rating, Prosper Score, Credit Lower Range, Credit Upper Range

In [572]:
```
# ProsperRatingnumeric
fig = plt.figure(figsize=(30, 10))
ax1 = fig.add_subplot(221)
sns.barplot(x="ProsperRating (numeric)", y="LoanStatus", data=p_nf)
plt.xticks(fontsize=14, rotation=90)

#ProsperScore
fig = plt.figure(figsize=(30, 10))
ax2 = fig.add_subplot(222)
sns.barplot(x="ProsperScore", y="LoanStatus", data=p_nf)
plt.xticks(fontsize=14, rotation=90)
```

Out[572]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 <a list of 12 Text xticklabel objects>)
```

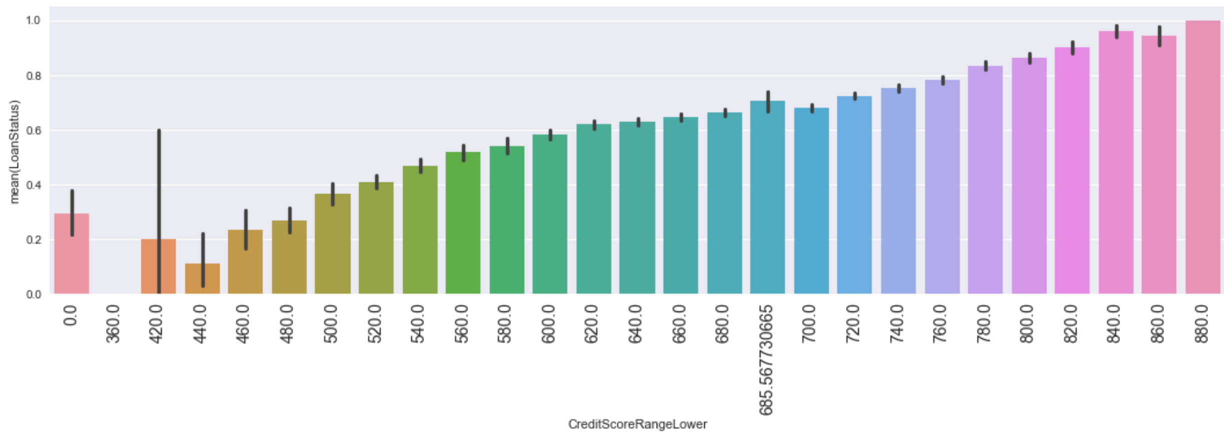


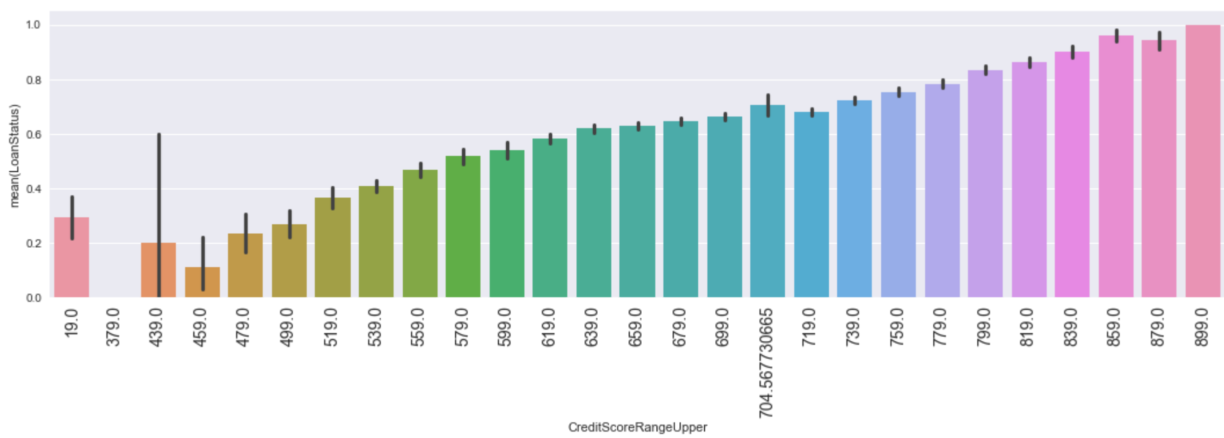


Higher the Prosper rating and ProsperScore, better the status of the loan being completed. Both the Prosper rating and Prosper score are linearly dependent on loan status and are doing pretty good in predicting the default. We can observe that as the rating number increases the prabability of loan being defaulted increases.There is some unusual behavior noticed at ProsperScore valued 11, it seems to default more than it's predecessors.

In [573]:
```
fig = plt.figure(figsize=(40, 10))
ax1 = fig.add_subplot(223)
sns.barplot(x="CreditScoreRangeLower", y="LoanStatus", data=p_nf)
plt.xticks(fontsize=14, rotation=90)

fig = plt.figure(figsize=(40, 10))
ax2 = fig.add_subplot(224)
sns.barplot(x="CreditScoreRangeUpper", y="LoanStatus", data=p_nf)
plt.xticks(fontsize=14, rotation=90)
```

Out[573]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26]),
 <a list of 27 Text xticklabel objects>)
```



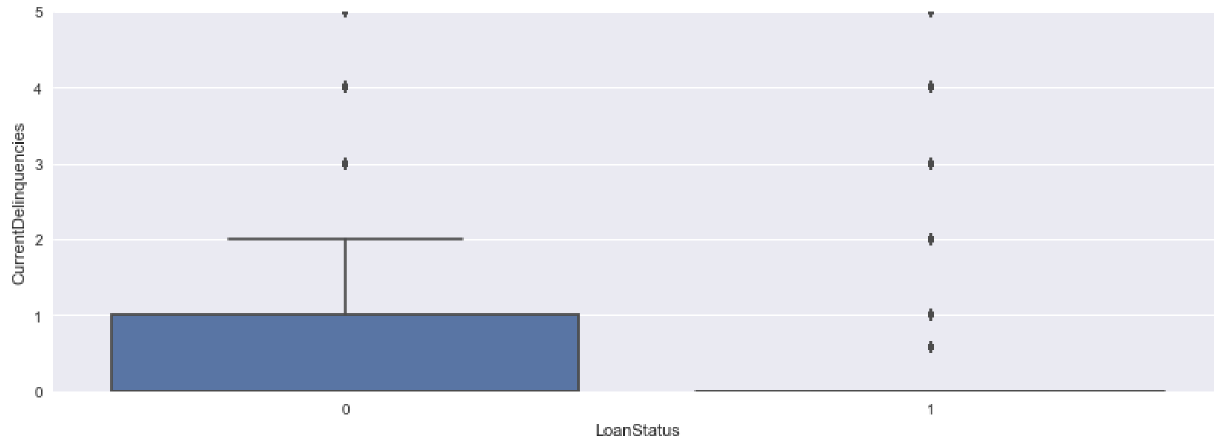


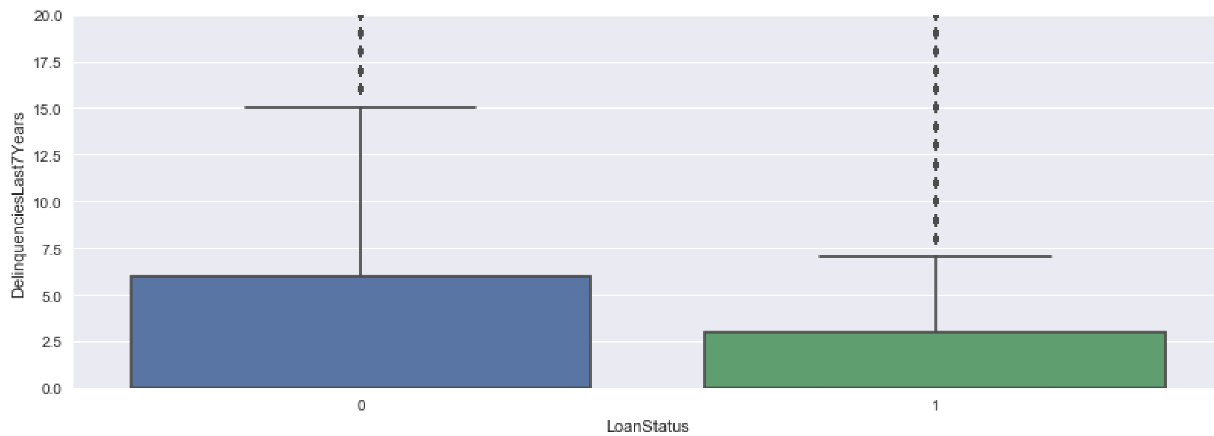
It should be noted here that the credit score "range" seems to be constant and behaves the same.

Credit History

In [574]:
```
fig = plt.figure(figsize=(30, 10))

ax1 = fig.add_subplot(221)
sns.boxplot(x="LoanStatus", y="CurrentCreditLines", data=p_nf).set_ylim([0,30])

fig = plt.figure(figsize=(30, 10))
ax2 = fig.add_subplot(222)
sns.boxplot(x="LoanStatus", y="TotalCreditLinespast7years", data=p_nf).set_ylim([

fig = plt.figure(figsize=(30, 10))
ax3 = fig.add_subplot(223)
sns.boxplot(x="LoanStatus", y="CurrentDelinquencies", data=p_nf).set_ylim([0,5])

fig = plt.figure(figsize=(30, 10))
ax4 = fig.add_subplot(224)
sns.boxplot(x="LoanStatus", y="DelinquenciesLast7Years", data=p_nf).set_ylim([0,2
```

Out[574]: (0, 20)



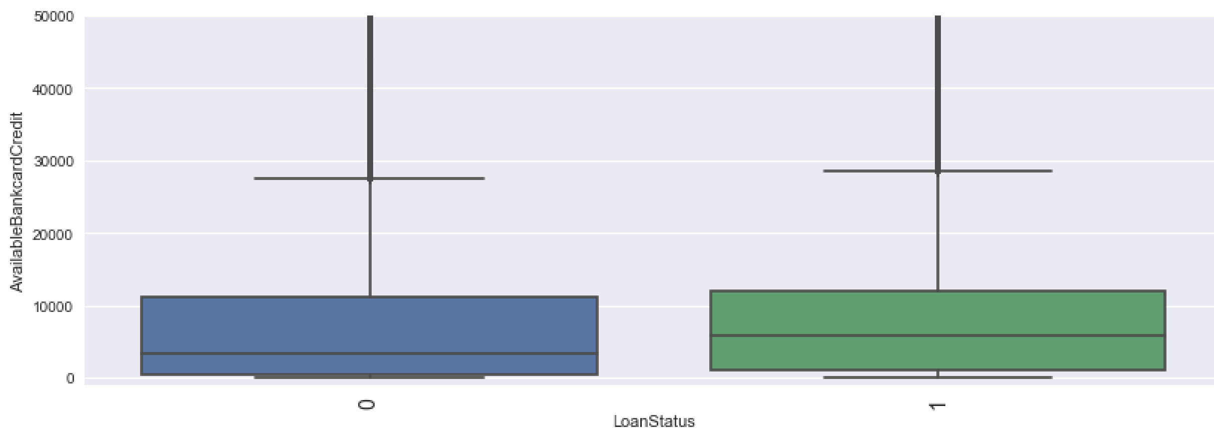


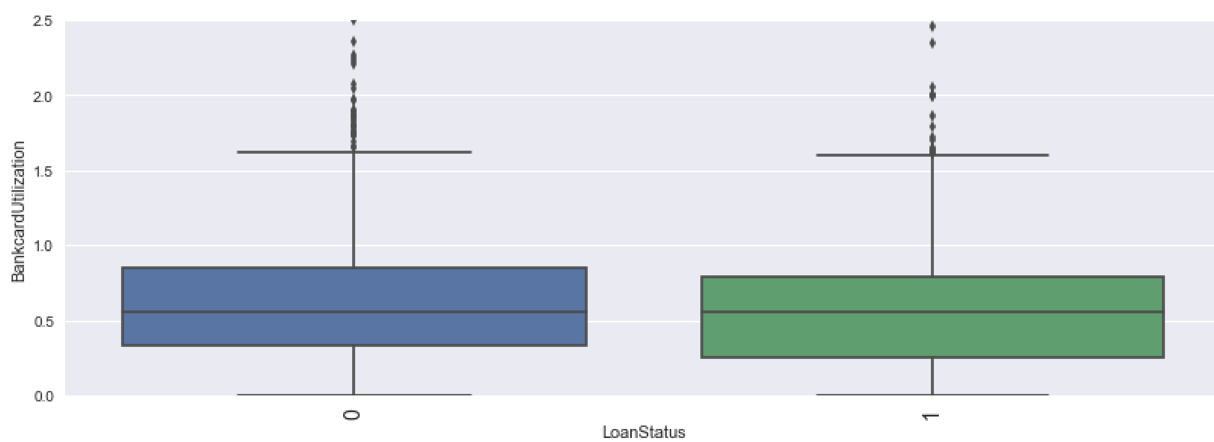
Credit Lines dosen't have significant relationship with Loan Status. However Delinquencies does seem to have some mild relationship.

Credit Information

In [575]:
```
fig = plt.figure(figsize=(30, 10))
ax1 = fig.add_subplot(221)
sns.boxplot(x="LoanStatus", y="AvailableBankcardCredit", data=p_nf).set_ylim([-10
plt.xticks(fontsize=14, rotation=90)

fig = plt.figure(figsize=(30, 10))
ax2 = fig.add_subplot(222)
sns.boxplot(x="LoanStatus", y="BankcardUtilization", data=p_nf).set_ylim([0,2.5])
plt.xticks(fontsize=14, rotation=90)
```

Out[575]:
```
(array([0, 1]), <a list of 2 Text xticklabel objects>)
```



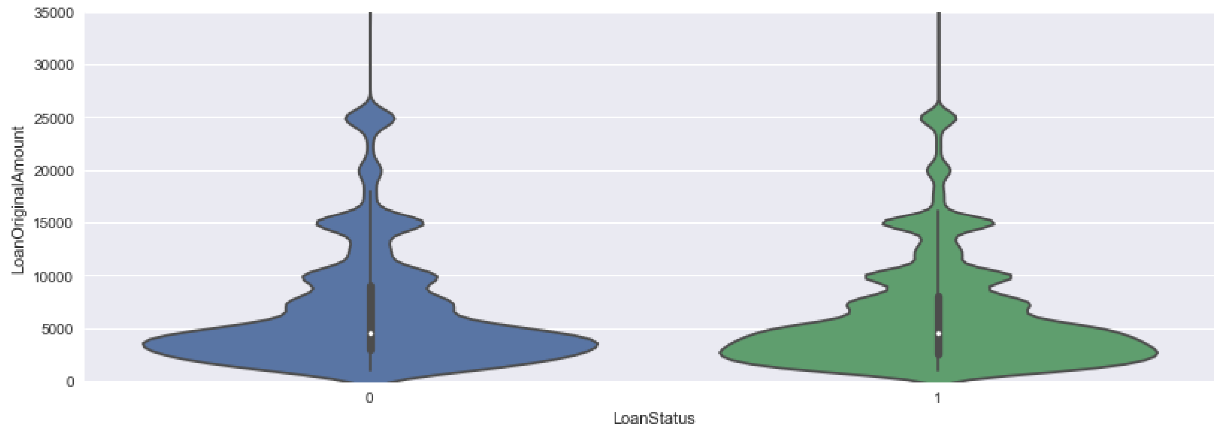


Loans that are subject to default has lower available bank card credit. Bank card utilization seem to be constant with Loan Status, however lower the proportion of bankcard utilization, lower the probaability of default.

Loan Characteristics

In [576]:
```
fig = plt.figure(figsize=(30, 10))
ax1 = fig.add_subplot(221)
sns.violinplot(x="LoanStatus", y="LoanOriginalAmount", data=p_nf).set_ylim([0,350

fig = plt.figure(figsize=(30, 10))
ax2 = fig.add_subplot(222)
sns.violinplot(x="LoanStatus", y="Term", data=p_nf).set_ylim([10, 60])
```

Out[576]: (10, 60)



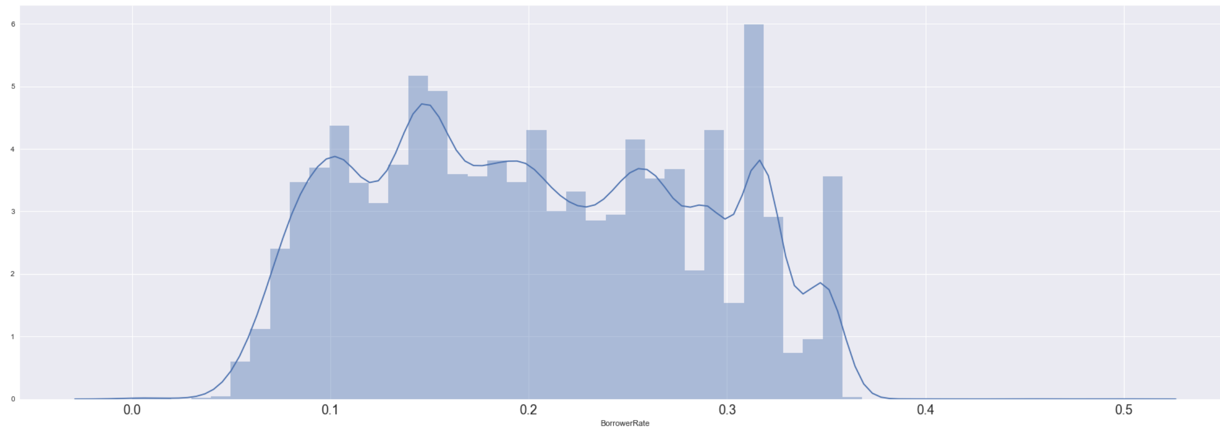
The violin plot helps in explaining the distribution of most of the points involved in the plot. In the above plots, the distribution appears to be similar for both variables against Loan Status.

Borrower Rate

In [577]:
```
fig = plt.figure(figsize=(30, 10))

sns.distplot(p_nf["BorrowerRate"])
plt.xticks(fontsize=18)
```

```
C:\Program Files\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetool
s.py:20: VisibleDeprecationWarning: using a non-integer number instead of an in
teger will result in an error in the future
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```

Out[577]:
```
(array([-0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6]),
 <a list of 8 Text xticklabel objects>)
```



In [578]:
```
p_nf['BorrowerRate'].describe()
```

Out[578]:
```
count    57361.000000
mean         0.201634
std          0.080952
min          0.000000
25%          0.136400
50%          0.198000
75%          0.269900
max          0.497500
Name: BorrowerRate, dtype: float64
```

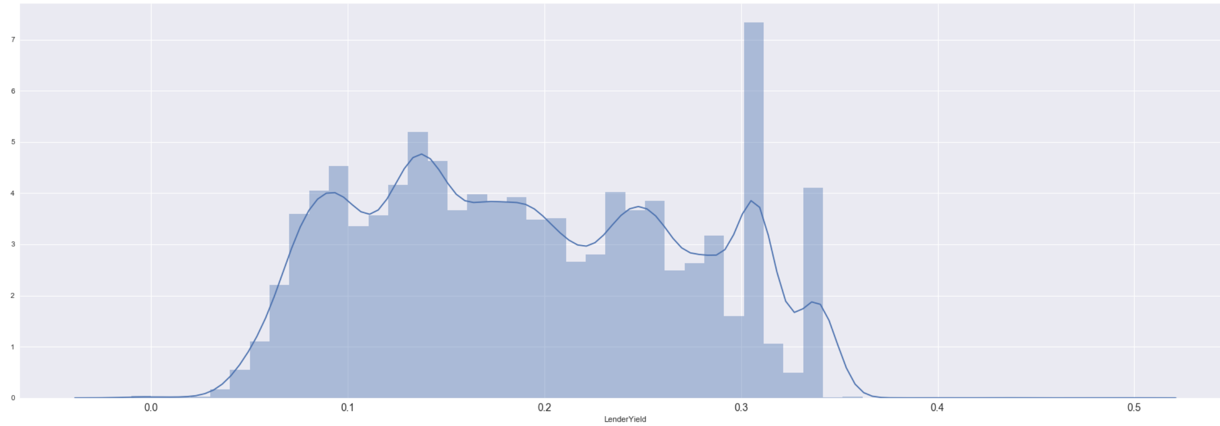
Borrower Rate is normally distributed with a mean interest rate of 20.16%. Upper quartile is paying an interest upto 27%, while the lower quartile is paying upto 17% interest rate.

LenderYield

In [579]:
```
fig = plt.figure(figsize=(30, 10))

sns.distplot(p_nf["LenderYield"])
plt.xticks(fontsize=14)
```

```
C:\Program Files\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetool
s.py:20: VisibleDeprecationWarning: using a non-integer number instead of an in
teger will result in an error in the future
  y = X[:m/2+1] + np.r_[0,X[m/2+1:],0]*1j
```

Out[579]:
```
(array([-0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6]),
 <a list of 8 Text xticklabel objects>)
```



In [580]:
```
p_nf.drop('LoanStatus', inplace=True, axis=1)
```

Lender yield distribution is normally distributed, mean of this distribution is around 0.19 and there is an yield with sharp peak ranging around 0.5, seems like this yield on the loan is with higher interest rate.

2) Exploring the categorical information of the data

In [581]:
```
p_c=pl.select_dtypes(include=['object'])
p_c.columns
```

Out[581]:
```
Index(['CreditGrade', 'LoanStatus', 'ClosedDate', 'ProsperRating (Alpha)',
       'BorrowerState', 'Occupation', 'EmploymentStatus',
       'IsBorrowerHomeowner', 'CurrentlyInGroup', 'FirstRecordedCreditLine',
       'IncomeRange', 'IncomeVerifiable', 'LoanOriginationQuarter'],
      dtype='object')
```

In [582]:
```
p_c.drop(['ProsperRating (Alpha)','ClosedDate'],inplace=True, axis=1)
```

```
C:\Program Files\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  if __name__ == '__main__':
```

In [583]:
```
p_c.isnull().sum()
```

Out[583]:
```
CreditGrade                84984
LoanStatus                     0
BorrowerState               5515
Occupation                  3588
EmploymentStatus            2255
IsBorrowerHomeowner            0
CurrentlyInGroup               0
FirstRecordedCreditLine      697
IncomeRange                    0
IncomeVerifiable               0
LoanOriginationQuarter         0
dtype: int64
```

In [584]:
```
p_c.fillna('Unknown', inplace=True)
```

```
C:\Program Files\Anaconda3\lib\site-packages\pandas\core\frame.py:2842: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  downcast=downcast, **kwargs)
```

In [585]:
```
p_c.isnull().sum()
```

Out[585]:
```
CreditGrade                0
LoanStatus                 0
BorrowerState              0
Occupation                 0
EmploymentStatus           0
IsBorrowerHomeowner        0
CurrentlyInGroup           0
FirstRecordedCreditLine    0
IncomeRange                0
IncomeVerifiable           0
LoanOriginationQuarter     0
dtype: int64
```

In [586]:
```
p_c=p_c[p_c['LoanStatus']!='Current']
```

In [587]:
```
p_c["LoanStatus"]=(p_c['LoanStatus']=='Completed').astype(int)
```

Listing Category

The data type of this variable is numeric by default because of different loan types being distinguished by numbers. It is important to convert this to categorical type for better interpretability and avoiding false ordinality.

In [588]:
```
data=pl[pl['LoanStatus']!='Current']
```

In [589]:
```
data["LoanStatus"]=(data['LoanStatus']=='Completed').astype(int)
```

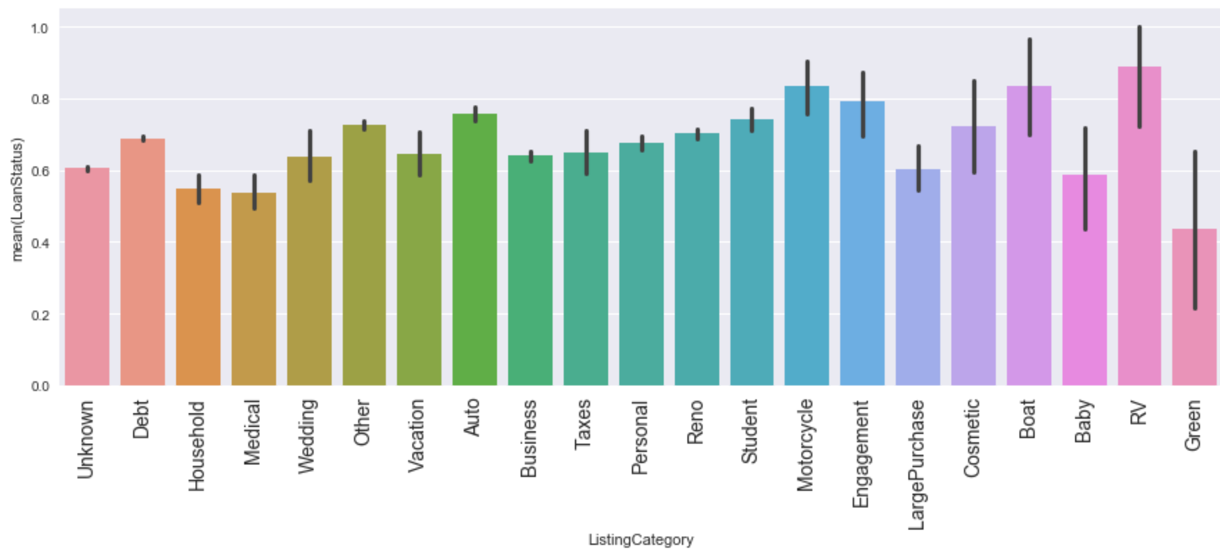
```
C:\Program Files\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  if __name__ == '__main__':
```

In [590]:
```
data.replace(to_replace={"ListingCategory (numeric)":
                         {0: "Unknown", 1: "Debt", 2: "Reno", 3: "Business", 4: "
                          5: "Student", 6: "Auto", 7: "Other", 8: "Baby", 9: "Boa
                         10: "Cosmetic", 11: "Engagement", 12: "Green", 13: "Hous
                         14: "LargePurchase", 15: "Medical", 16: "Motorcycle", 17
                         18: "Taxes", 19: "Vacation", 20: "Wedding"}}, inplace=Tr

data.rename(index=str, columns={"ListingCategory (numeric)": "ListingCategory"},
```

```
C:\Program Files\Anaconda3\lib\site-packages\pandas\core\generic.py:3485: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  regex=regex)
C:\Program Files\Anaconda3\lib\site-packages\pandas\core\frame.py:2834: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  **kwargs)
```

In [591]:
```
fig = plt.figure(figsize=(15, 5))

sns.barplot(x="ListingCategory", y="LoanStatus", data=data)

ax1 = fig.add_subplot(figsize=(30, 10))
plt.xticks(fontsize=14, rotation=90)
```

Out[591]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20]), <a list of 21 Text xticklabel objects>)
```



This visualization gives insights about probability of types of loans that are subject to good/ bad loans. Considering a threshold of 80% that makes excellent loans, loans such as 'Motorcycle','Engagement','Boat','RV' tend to complete their loans within the specified time.

It is evident to see 'Student' loans which is ususally a huge sum taken, to get completed almost 70% of their time.

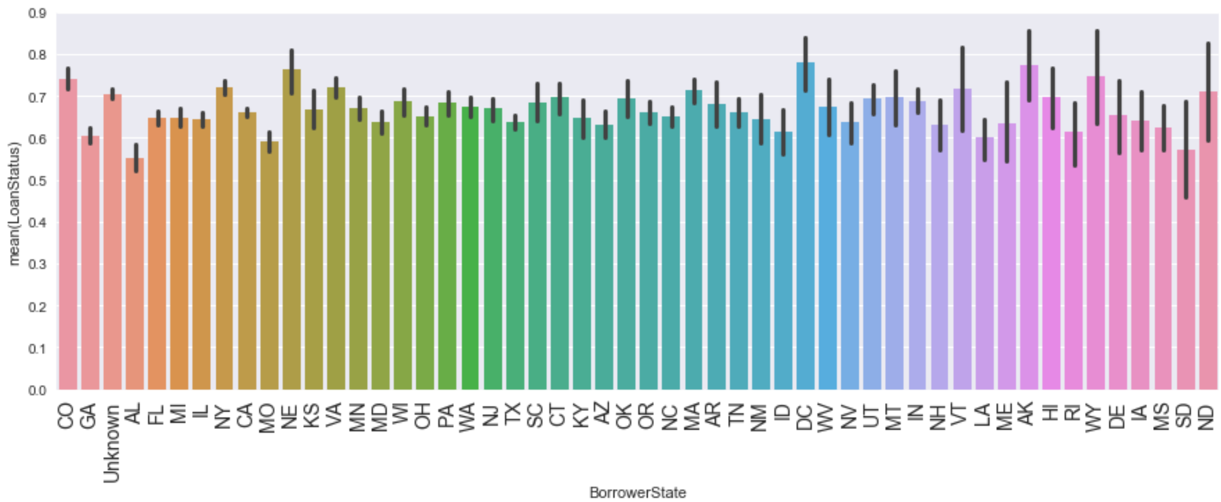
Loans such as 'Household','Medical','LargerPurchase','Baby' complete their loans close to 60% of their time.These loans fall under common categories and usually have huge sums of loans taken, reason for which accounts to 60%.

'Green' loans are subjected to default almost 60% of their time. These loans are probably deemed as bad loans.

Borrower State

In [592]:
```
fig = plt.figure(figsize=(15, 5))

sns.barplot(x="BorrowerState", y="LoanStatus", data=p_c)


ax1 = fig.add_subplot(figsize=(30, 10))
plt.xticks(fontsize=14, rotation=90)
```

Out[592]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
         17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
         34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
         51]), <a list of 52 Text xticklabel objects>)
```

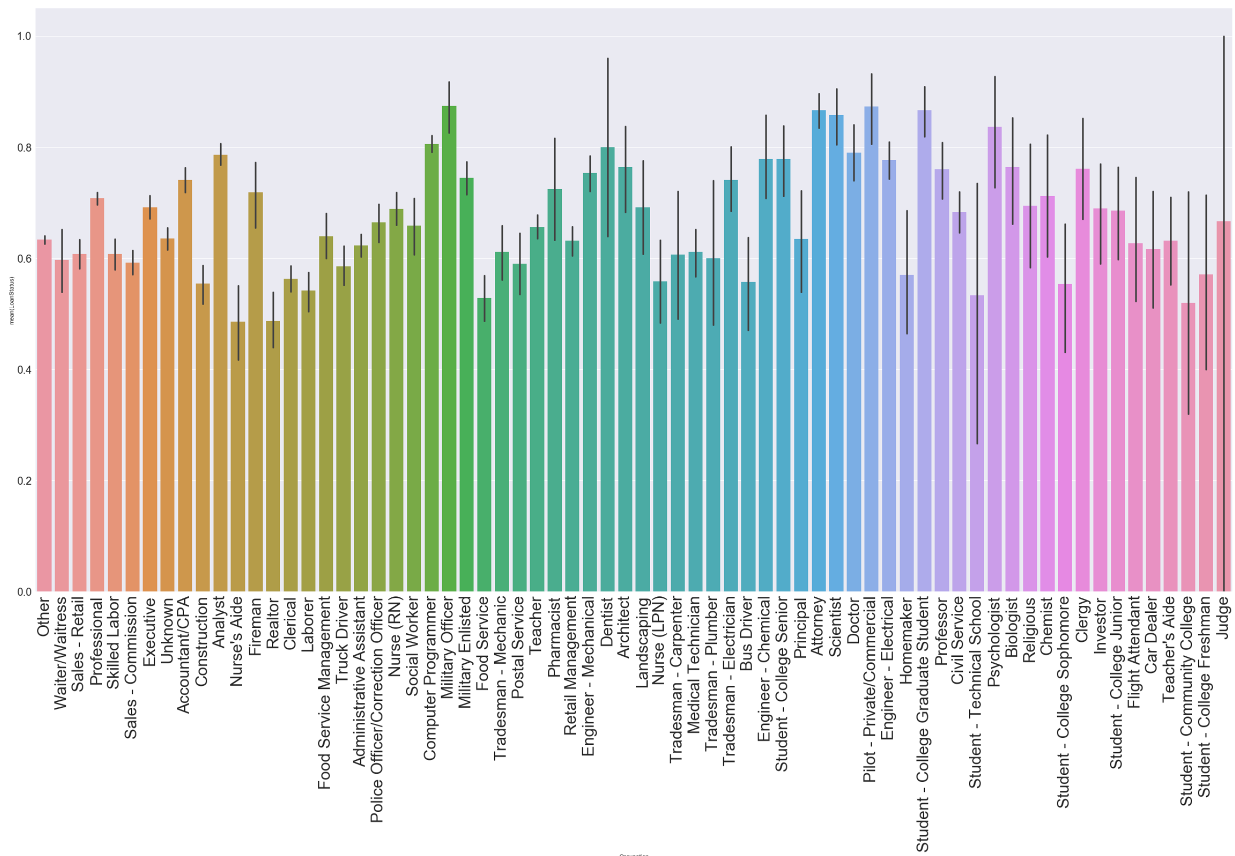


Fom the bar plot, we can see the non defaulted values of loan statuses of different states of USA. Alabama, Missouri and San Diego states 'Completed' loan average strikes between 50% and 60%. The largest non defaulted states are Nebraska and Washington DC followed by Arkansas, Wyoming and colorado ranging between 75% to 80% of their 'Completed loans'. This categorical data seems to give good insights.

Occupation

In [593]:
```
fig=plt.figure(figsize=(40,20))
sns.barplot(x='Occupation', y='LoanStatus', data=p_c)
plt.xticks(fontsize=30, rotation=90)
plt.yticks(fontsize=20)
```

Out[593]:
```
(array([ 0. ,  0.2,  0.4,  0.6,  0.8,  1. ,  1.2]),
 <a list of 7 Text yticklabel objects>)
```

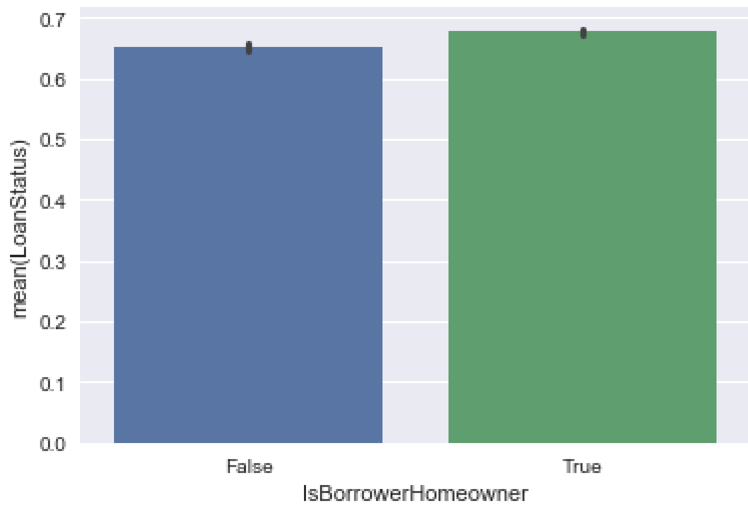


Setting a threshold of 80% and greater to be the top occupations that result in good return of the loans, some of these include 'Military Officer','Computer Programmer','Dentist','Attorney','Scientist','Pilot','Student-College Graduate Student' and 'Psychologist'. While occupations that have probability to complete the loans less than 50% include 'Nurse's Aide' and 'Realtor'.

Is Borrower Home Owner

In [594]:
```
sns.barplot(x='IsBorrowerHomeowner', y='LoanStatus', data=p_c)
```

Out[594]: <matplotlib.axes._subplots.AxesSubplot at 0x2429a6fd278>



In [595]:
```
import scipy
from scipy.stats import pearsonr
x=data.IsBorrowerHomeowner
y=data.LoanStatus
cor, p= pearsonr(x, y)
print("The correlation between IsBorrowerHomeowner and loan default is {}, with a
```

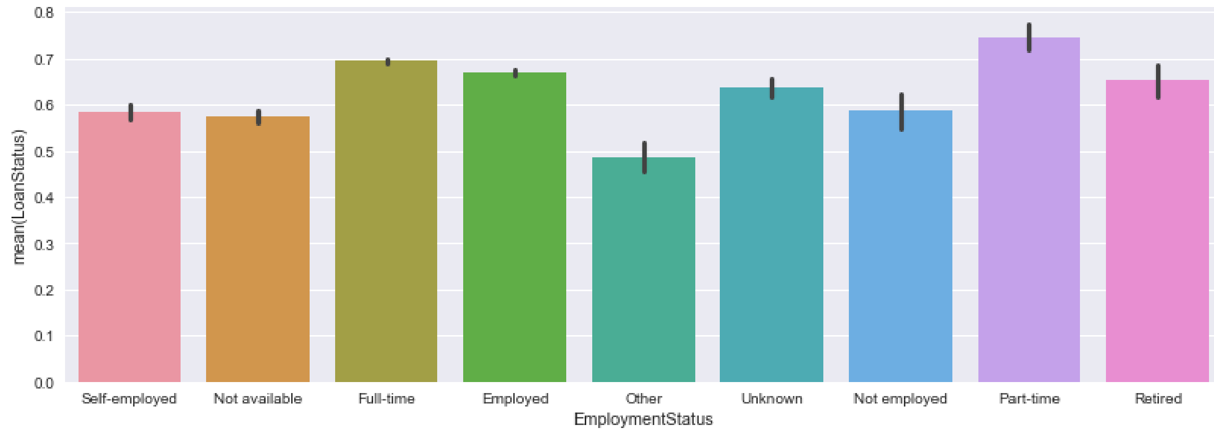
```
The correlation between IsBorrowerHomeowner and loan default is 0.0265008260759
58207, with a p-value of 2.1818058327590924e-10
```

Looking at this plot, this variable has almost equal distribution in predicting loan status, however home owners tend to default less. Also the correlation is not so strong.So having information whether borrower is a home owner or not, dosen't really help in prediciting the status of loan payment.

Employment Status

In [596]:
```
fig = plt.figure(figsize=(30, 10))

ax1 = fig.add_subplot(221)
sns.barplot(x="EmploymentStatus", y="LoanStatus", data=p_c)
```

Out[596]: <matplotlib.axes._subplots.AxesSubplot at 0x2429ac7e128>



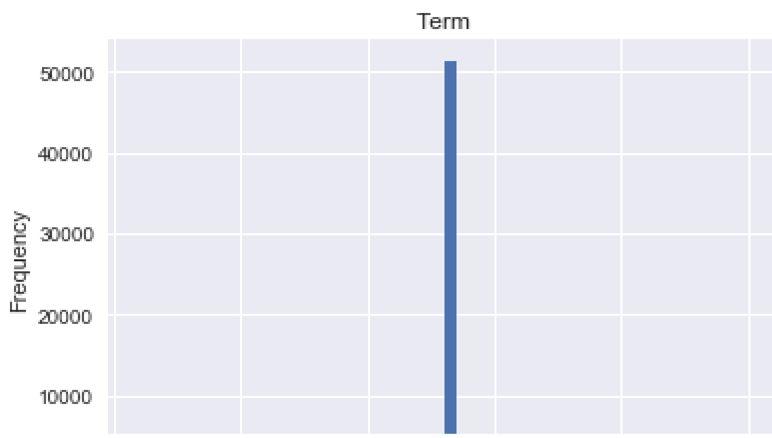
EmploymentStatus has a relationship with default. We see a interesting observation here as part-time workers defaulted less often than the full-time workers and other employement status. It is great to see Retired people having successful loan payment percentage close to 70%.Also, unfortunately the people who listed their employment status as Other defaulted even more often than those who were non employed.

Feature Engineering: Numeric

In [597]:

```
#NUMERICAL DATA

import matplotlib.pyplot as plt
%matplotlib inline

for i in p_nf.columns:
    fig, ax = plt.subplots()
    p_nf[i].hist(bins=50)
    ax.set_title(i)
    ax.set_xlabel(i)
    ax.set_ylabel('Frequency')
```

```
C:\Program Files\Anaconda3\lib\site-packages\matplotlib\pyplot.py:524: Runtim
eWarning: More than 20 figures have been opened. Figures created through the
pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly c
losed and may consume too much memory. (To control this warning, see the rcPa
ram `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```



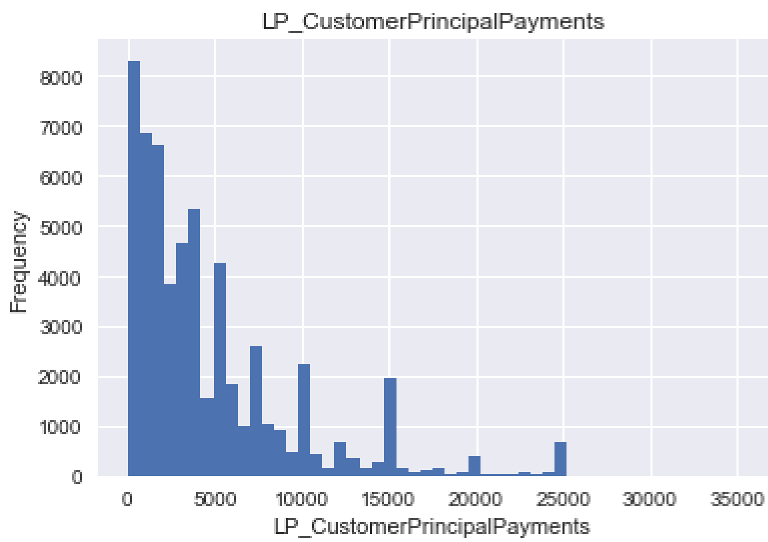
In [598]:

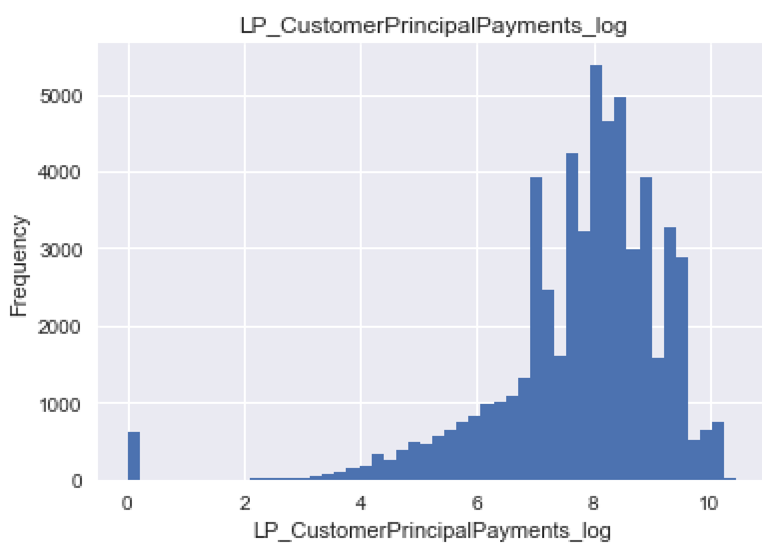
```
p_nf['LP_CustomerPrincipalPayments_log'] = np.log(1+(p_nf['LP_CustomerPrincipalPa
p_nf['LP_CustomerPayments_log'] = np.log((1+ p_nf['LP_CustomerPayments']))
p_nf['LoanOriginalAmount_log'] = np.log((1+ p_nf['LoanOriginalAmount']))
p_nf['Investors_log'] = np.log((1+ p_nf['Investors']))
p_nf['MonthlyLoanPayment_log'] =np.log((1+ p_nf['MonthlyLoanPayment']))
```

```
C:\Program Files\Anaconda3\lib\site-packages\ipykernel\__main__.py:2: RuntimeWa
rning: invalid value encountered in log
  from ipykernel import kernelapp as app
```

In [599]:
```
fig, ax = plt.subplots()
p_nf['LP_CustomerPrincipalPayments'].hist(bins=50)
ax.set_title('LP_CustomerPrincipalPayments')
ax.set_xlabel('LP_CustomerPrincipalPayments')
ax.set_ylabel('Frequency')

fig, ax = plt.subplots()
p_nf['LP_CustomerPrincipalPayments_log'].hist(bins=50)
ax.set_title('LP_CustomerPrincipalPayments_log')
ax.set_xlabel('LP_CustomerPrincipalPayments_log')
ax.set_ylabel('Frequency')
```

Out[599]: <matplotlib.text.Text at 0x2428f428320>

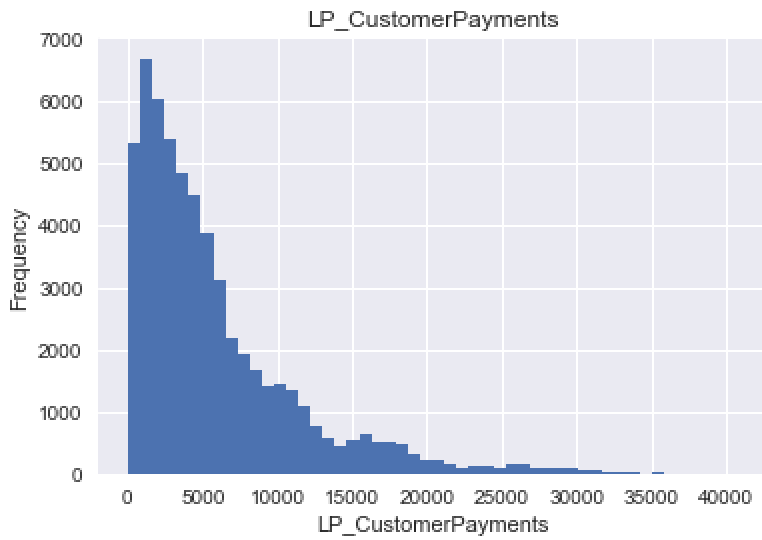


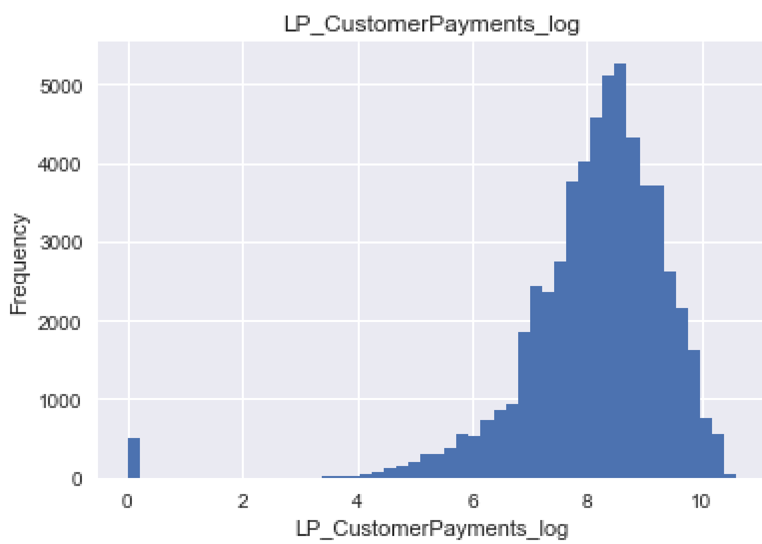


In [600]:

```
fig, ax = plt.subplots()
p_nf['LP_CustomerPayments'].hist(bins=50)
ax.set_title('LP_CustomerPayments')
ax.set_xlabel('LP_CustomerPayments')
ax.set_ylabel('Frequency')

fig, ax = plt.subplots()
p_nf['LP_CustomerPayments_log'].hist(bins=50)
ax.set_title('LP_CustomerPayments_log')
ax.set_xlabel('LP_CustomerPayments_log')
ax.set_ylabel('Frequency')
```

Out[600]: <matplotlib.text.Text at 0x24290968400>

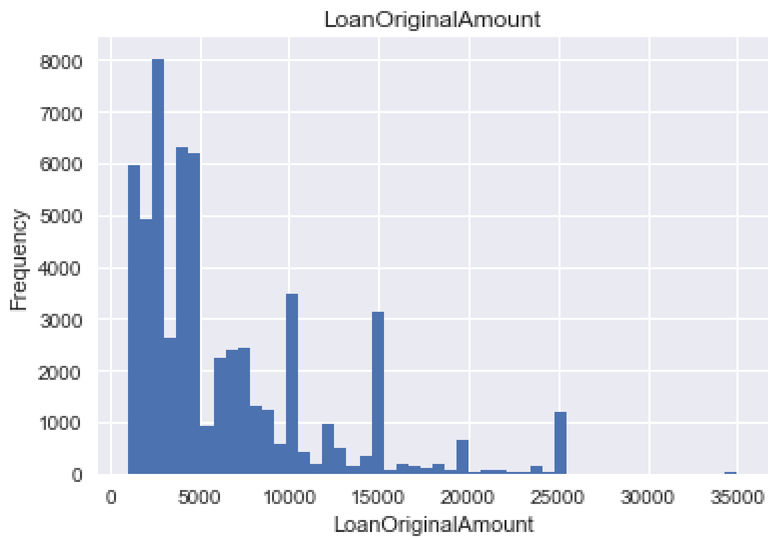


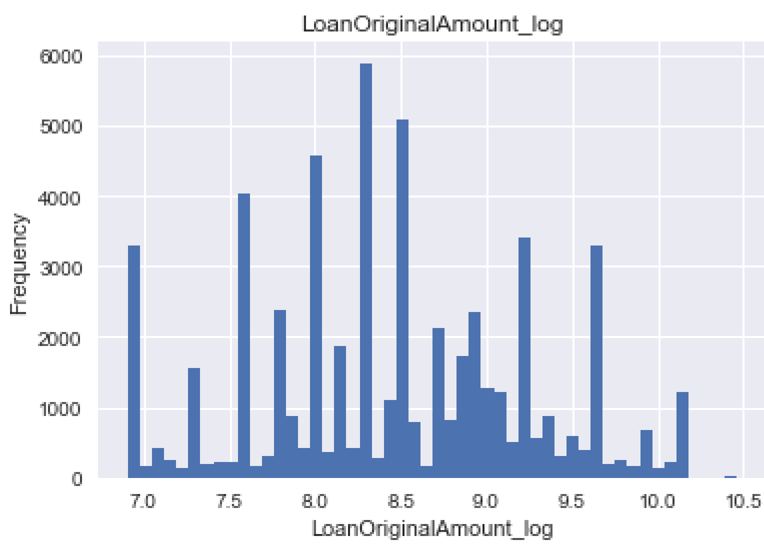


In [601]:

```
fig, ax = plt.subplots()
p_nf['LoanOriginalAmount'].hist(bins=50)
ax.set_title('LoanOriginalAmount')
ax.set_xlabel('LoanOriginalAmount')
ax.set_ylabel('Frequency')

fig, ax = plt.subplots()
p_nf['LoanOriginalAmount_log'].hist(bins=50)
ax.set_title('LoanOriginalAmount_log')
ax.set_xlabel('LoanOriginalAmount_log')
ax.set_ylabel('Frequency')
```

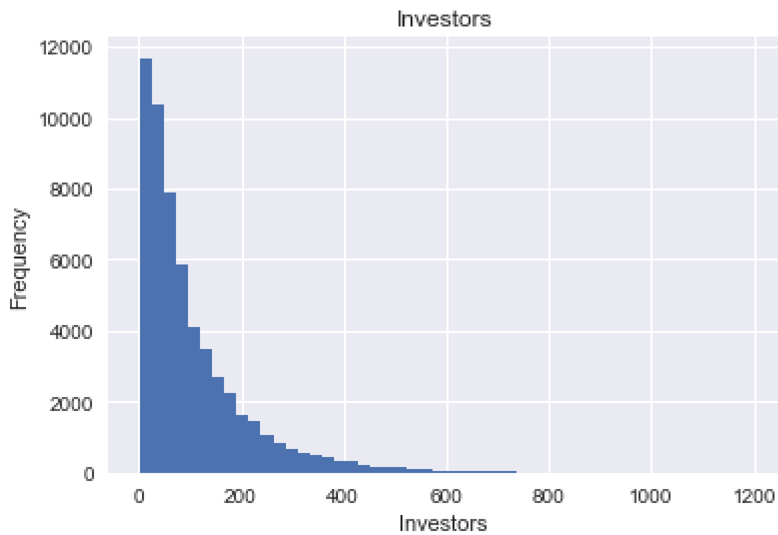
Out[601]: <matplotlib.text.Text at 0x24290c64cc0>

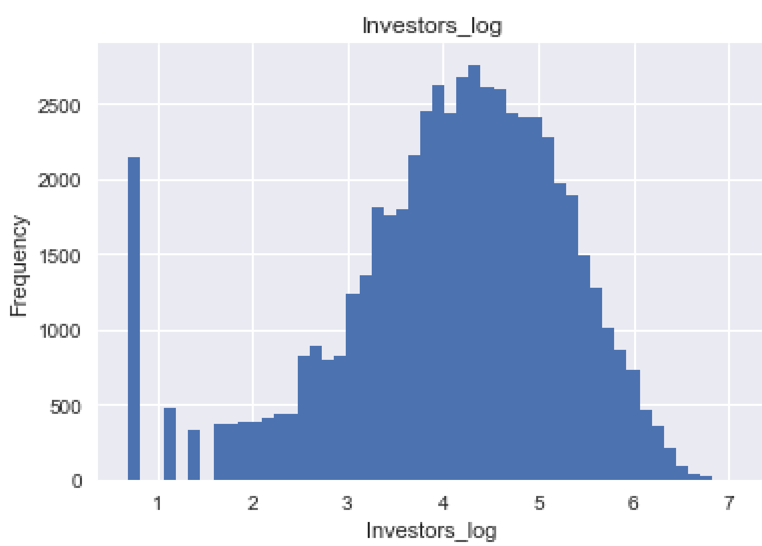




In [602]:
```
fig, ax = plt.subplots()
p_nf['Investors'].hist(bins=50)
ax.set_title('Investors')
ax.set_xlabel('Investors')
ax.set_ylabel('Frequency')

fig, ax = plt.subplots()
p_nf['Investors_log'].hist(bins=50)
ax.set_title('Investors_log')
ax.set_xlabel('Investors_log')
ax.set_ylabel('Frequency')
```

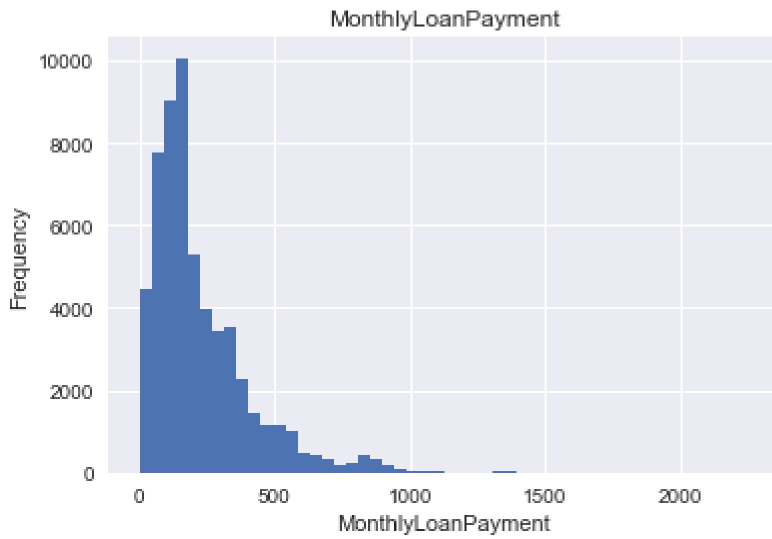
Out[602]: <matplotlib.text.Text at 0x2429166bda0>

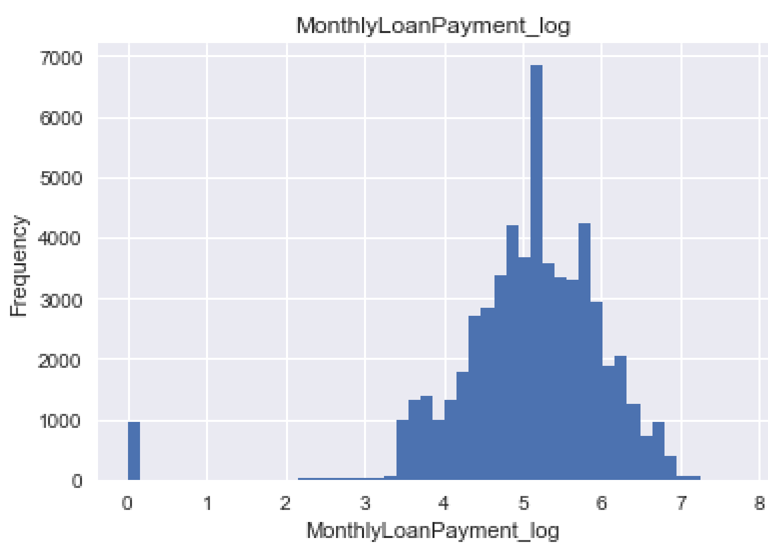




In [603]:
```
fig, ax = plt.subplots()
p_nf['MonthlyLoanPayment'].hist(bins=50)
ax.set_title('MonthlyLoanPayment')
ax.set_xlabel('MonthlyLoanPayment')
ax.set_ylabel('Frequency')

fig, ax = plt.subplots()
p_nf['MonthlyLoanPayment_log'].hist(bins=50)
ax.set_title('MonthlyLoanPayment_log')
ax.set_xlabel('MonthlyLoanPayment_log')
ax.set_ylabel('Frequency')
```

Out[603]: <matplotlib.text.Text at 0x24290abf390>





Dropping the original columns (untransformed variables)

In [604]:
```
to_drop=['LP_CustomerPayments','LP_CustomerPrincipalPayments','LoanOriginalAmount
        'MonthlyLoanPayment','Investors']
```

In [605]:
```
p_nf.drop(to_drop,inplace=True, axis=1)
```

In [606]:
```
p_nf[p_nf.isnull()] = 0
```

In [607]:
```
y1=(p_c['LoanStatus'])
```

In [608]:
```
y1.value_counts()
```

Out[608]:
```
1    38074
0    19287
Name: LoanStatus, dtype: int64
```

In [609]:
```
p_c.drop(['LoanStatus'], inplace=True,axis=1)
```

In [610]:
```
p_c.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 57361 entries, 0 to 113935
Data columns (total 10 columns):
CreditGrade                 57361 non-null object
BorrowerState               57361 non-null object
Occupation                  57361 non-null object
EmploymentStatus            57361 non-null object
IsBorrowerHomeowner         57361 non-null bool
CurrentlyInGroup            57361 non-null bool
FirstRecordedCreditLine     57361 non-null object
IncomeRange                 57361 non-null object
IncomeVerifiable            57361 non-null bool
LoanOriginationQuarter      57361 non-null object
dtypes: bool(3), object(7)
memory usage: 3.7+ MB
```

FEATURE ENGINEERING, CATEGORICAL DATA

For simplicity, replacing the 'Not employed' target variable type to 0

In [611]:
```
p_c.IncomeRange.replace('Not employed','$0', inplace=True)
```

In [612]:
```
p_c['IncomeRange'].value_counts()
```

Out[612]:
```
$25,000-49,999    17081
$50,000-74,999    13435
Not displayed      7741
$75,000-99,999     6777
$100,000+          6421
$1-24,999          4738
$0                 1168
Name: IncomeRange, dtype: int64
```

Mapping the ordinal data type

In [613]:
```
income_range_map={'Not displayed' :0,'$0':1,'$1-24,999' :2,'$25,000-49,999':3,
                  '$50,000-74,999':4,'$75,000-99,999':5,'$100,000+':6}
p_c['Income_Range_Label']=p_c.IncomeRange.map(income_range_map)
```

In [614]:
```
p_c['Income_Range_Label'].value_counts()
```

Out[614]:
```
3    17081
4    13435
0     7741
5     6777
6     6421
2     4738
1     1168
Name: Income_Range_Label, dtype: int64
```

In [615]:
```
Income_Range_dummy=pd.get_dummies(p_c.Income_Range_Label, drop_first=True)
```

In [616]:
```
Income_Range_dummy.head()
```

Out[616]:

	1	2	3	4	5	6
0	0	0	1	0	0	0
2	0	0	0	0	0	0
11	0	1	0	0	0	0
12	0	0	1	0	0	0
15	0	0	0	1	0	0

In [617]:
```
dummy=p_c[['CreditGrade','EmploymentStatus','BorrowerState','Occupation','LoanOri
dummies=pd.get_dummies(dummy, drop_first=True)
dummies.head()
```

Out[617]:

	CreditGrade_AA	CreditGrade_B	CreditGrade_C	CreditGrade_D	CreditGrade_E	Credit
0	0	0	1	0	0	0
2	0	0	0	0	0	1
11	0	0	1	0	0	0
12	0	0	0	0	0	0
15	0	0	0	0	0	0

5 rows × 166 columns

In [618]:
```
from datetime import timedelta
import datetime
p_c['FirstRecordedCreditLine'].replace('unknown',0,inplace=True)
p_c['FirstRecordedCreditLine'].replace('Unknown',0,inplace=True)
p_c['FirstRecordedCreditLine']=pd.to_datetime(p_c.FirstRecordedCreditLine)
p_c['FirstRecordedCreditLine']=p_c.FirstRecordedCreditLine.apply(lambda date: dat
```

In [619]:
```
index=p_c.index
```

In [620]:
```
## FEATURE HASHING
from sklearn.feature_extraction import FeatureHasher
fh1=FeatureHasher(n_features=150, input_type='string')

#hash1
hash_1=fh1.fit_transform(p_c['FirstRecordedCreditLine'])
hash_1=hash_1.toarray()
hash_1=pd.DataFrame(hash_1, index=index)
```

In [621]:
```
hash_1.head()
```

Out[621]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	2
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
11	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
15	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

5 rows × 150 columns

In [622]:
```
categorical_loan=pd.concat([hash_1,Income_Range_dummy,dummies],axis=1)
```

In [623]:
```
categorical_loan.head()
```

Out[623]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	2
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
11	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
15	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

5 rows × 322 columns

In [624]:
```
y_f=pd.DataFrame(y1)
```

In [625]:
```
y_f.head(10)
```

Out[625]:

	LoanStatus
0	1
2	1
11	1
12	0
15	0
17	0
21	1
23	0
26	1
27	1

In [626]:
```
final=pd.concat((p_nf,categorical_loan,y_f), axis=1, join='outer')
```

In [627]:
```
final.head()
```

Out[627]:

	Term	BorrowerAPR	BorrowerRate	LenderYield	EstimatedEffectiveYield	EstimatedLo
0	36.0	0.16516	0.1580	0.1380	0.168661	0.080306
2	36.0	0.28269	0.2750	0.2400	0.168661	0.080306
11	36.0	0.15033	0.1325	0.1225	0.168661	0.080306
12	36.0	0.17969	0.1435	0.1335	0.126400	0.052400
15	36.0	0.35797	0.3177	0.3077	0.289600	0.165000

5 rows × 376 columns

In [628]:
```
final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 57361 entries, 0 to 113935
Columns: 376 entries, Term to LoanStatus
dtypes: float64(203), int32(1), uint8(172)
memory usage: 98.9 MB
```

In [629]:
```
pd.set_option("display.max_columns", len(final.columns))
final.head(10)
```

Out[629]:

	Term	BorrowerAPR	BorrowerRate	LenderYield	EstimatedEffectiveYield	EstimatedLos
0	36.0	0.16516	0.1580	0.1380	0.168661	0.080306
2	36.0	0.28269	0.2750	0.2400	0.168661	0.080306
11	36.0	0.15033	0.1325	0.1225	0.168661	0.080306
12	36.0	0.17969	0.1435	0.1335	0.126400	0.052400
15	36.0	0.35797	0.3177	0.3077	0.289600	0.165000
17	36.0	0.13202	0.1250	0.1175	0.168661	0.080306
21	36.0	0.21488	0.2075	0.1975	0.168661	0.080306
23	36.0	0.28032	0.2419	0.2319	0.212600	0.107500
26	60.0	0.30748	0.2809	0.2709	0.247300	0.122500
27	36.0	0.11296	0.0920	0.0820	0.060800	0.021000

In [630]:
```
#Train test split
X=final.iloc[:,:-1]
y=final.iloc[:,-1]
```

In [631]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 57361 entries, 0 to 113935
Columns: 375 entries, Term to LoanOriginationQuarter_Q4 2013
dtypes: float64(203), uint8(172)
memory usage: 98.7 MB
```

In [632]:
```
X.head()
```

Out[632]:

	Term	BorrowerAPR	BorrowerRate	LenderYield	EstimatedEffectiveYield	EstimatedLos
0	36.0	0.16516	0.1580	0.1380	0.168661	0.080306
2	36.0	0.28269	0.2750	0.2400	0.168661	0.080306
11	36.0	0.15033	0.1325	0.1225	0.168661	0.080306
12	36.0	0.17969	0.1435	0.1335	0.126400	0.052400
15	36.0	0.35797	0.3177	0.3077	0.289600	0.165000

Scaling the variables before training the model

In [633]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

#Fit on training set only
scaler.fit(X)

#Apply transform on both training and test set
X = scaler.transform(X)
```

In [634]:
```
X=pd.DataFrame(X, index=final.index)

final_loan_normal = pd.concat((X,y), axis=1, join='outer')
```

In [635]:
```
final_loan_normal.to_csv('prosper_final_normal.csv')
```

In [636]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.20, random_state=
```

In [637]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45888 entries, 52775 to 27871
Columns: 375 entries, 0 to 374
dtypes: float64(375)
memory usage: 131.6 MB
```

In [638]:
```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11473 entries, 1872 to 3095
Columns: 375 entries, 0 to 374
dtypes: float64(375)
memory usage: 32.9 MB
```

CLASSIFIERS

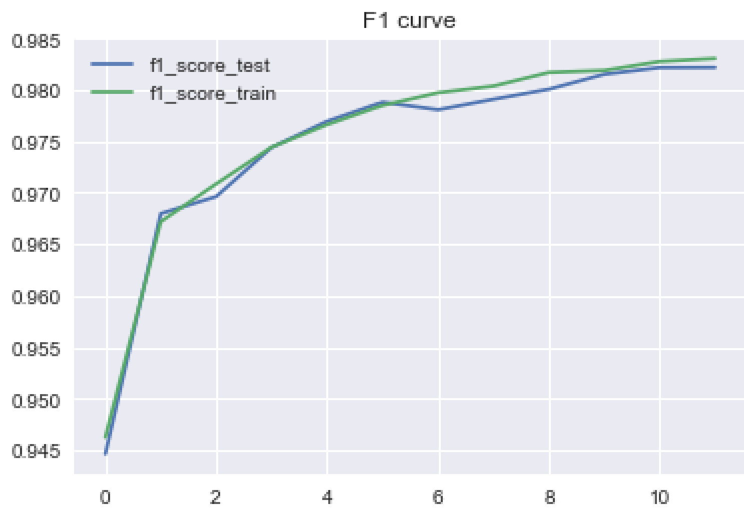
Model 1: SVM with L1(Binary)

In [639]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import confusion_matrix
from sklearn.metrics import average_precision_score
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import f1_score, recall_score

f1_score_train = []
f1_score_test = []
recall_train = []
recall_test = []


chunksize = 5000

estimator = SGDClassifier(loss='hinge', penalty='l1', l1_ratio=1)
for i,chunk in enumerate(pd.read_csv('prosper_final_normal.csv', chunksize=chunks
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    # Since our focus is on default loan class, the parameter pos_label is set to
    f_te = f1_score(y_test,y_pred_test,pos_label=0)
    f_ta = f1_score(y_train,y_pred_train, pos_label=0)

    rc_test=recall_score(y_test,y_pred_test, pos_label=0)
    rc_train=recall_score(y_train,y_pred_train, pos_label=0)

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)

    recall_train.append(rc_train)
    recall_test.append(rc_test)
```

In [640]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test

print('Average Recall score is : {0:0.2f}'.format(sum(recall_test)/len(recall_tes
```

```
Average f1 score :  0.97
Average Recall score is : 0.96
```

In [641]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()

plt.plot(recall_test)
plt.plot(recall_train)
plt.legend(('recall_test','recall_train'))
plt.title('Recall curve')
plt.show()
```



Model 2:Logistic Regression with L1(Binary)

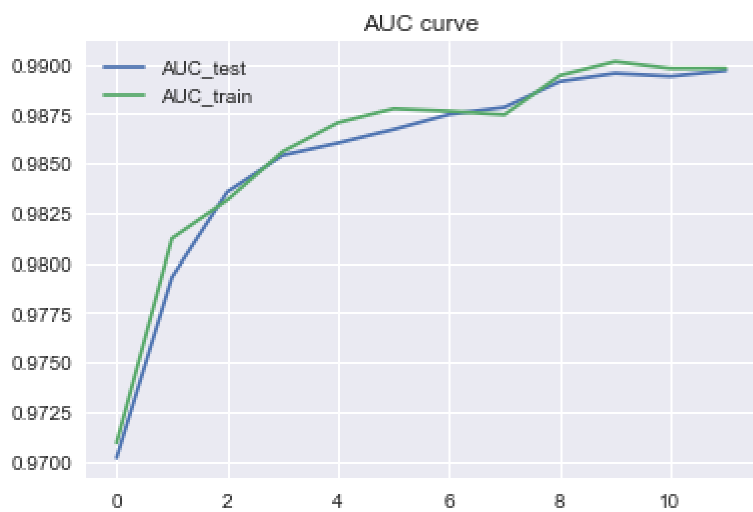
In [642]:
```
#Logistic Regression
from sklearn.metrics import roc_auc_score

auc_train = []
auc_test = []
chunksize = 5000

estimator = SGDClassifier(loss='log', penalty='l1', l1_ratio=1)
for i,chunk in enumerate(pd.read_csv('prosper_final_normal.csv', chunksize=chunks
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict_proba(X_test)[:,1]
    y_pred_train = estimator.predict_proba(X_train)[:,1]

    auc_tr=roc_auc_score(y_train,y_pred_train,average='micro')
    auc_te=roc_auc_score(y_test,y_pred_test, average='micro')

    auc_train.append(auc_tr)
    auc_test.append(auc_te)
```

```
C:\Program Files\Anaconda3\lib\site-packages\sklearn\linear_model\base.py:352:
RuntimeWarning: overflow encountered in exp
  np.exp(prob, prob)
```

In [643]:
```
print('Average AUC Score : {0:0.2f}' .format(sum(auc_test)/len(auc_test)))
```

```
Average AUC Score : 0.99
```

In [644]:
```
plt.plot(auc_train)
plt.plot(auc_test)
plt.legend(('AUC_test','AUC_train'))
plt.title('AUC curve')
plt.show()
```



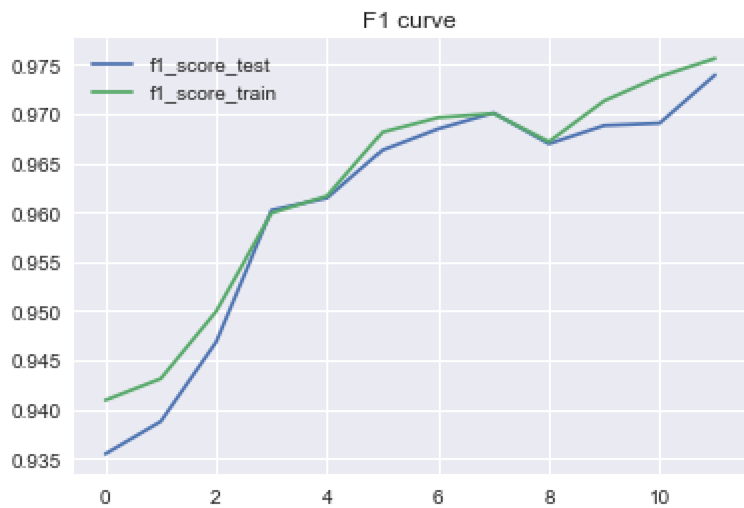
Model 3:SVM with L2(Binary)

In [645]:
```
f1_score_train = []
f1_score_test = []

recall_train = []
recall_test=[]

chunksize = 5000

estimator = SGDClassifier(loss='hinge', penalty='l2', l1_ratio=0)
for i,chunk in enumerate(pd.read_csv('prosper_final_normal.csv', chunksize=chunks
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test, pos_label=0)
    f_ta = f1_score(y_train,y_pred_train, pos_label=0)

    rc_test = recall_score(y_test,y_pred_test, pos_label=0)
    rc_train = recall_score(y_train, y_pred_train,pos_label=0)

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)

    recall_train.append(rc_train)
    recall_test.append(rc_test)
```

In [646]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test

print('Average Recall score is : {0:0.2f}'.format(sum(recall_test)/len(recall_tes
```

```
Average f1 score :  0.96
Average Recall score is : 0.96
```

In [647]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()

plt.plot(recall_test)
plt.plot(recall_train)
plt.legend(('recall_test','recall_train'))
plt.title('Recall curve')
plt.show()
```



Model 4:Logistic Regression with L2(Binary)

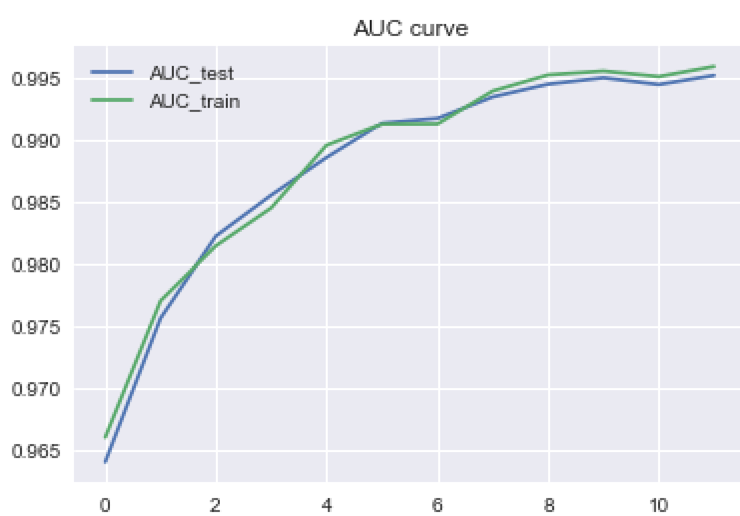
In [648]:
```
#Logistic Regression

auc_train = []
auc_test = []


chunksize = 5000

estimator = SGDClassifier(loss='log', penalty='l2', l1_ratio=0)
for i,chunk in enumerate(pd.read_csv('prosper_final_normal.csv', chunksize=chunks
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict_proba(X_test)[:,1]
    y_pred_train = estimator.predict_proba(X_train)[:,1]

    auc_tr=roc_auc_score(y_train,y_pred_train,average='micro')
    auc_te=roc_auc_score(y_test,y_pred_test, average='micro')

    auc_train.append(auc_tr)
    auc_test.append(auc_te)
```

```
C:\Program Files\Anaconda3\lib\site-packages\sklearn\linear_model\base.py:352:
RuntimeWarning: overflow encountered in exp
  np.exp(prob, prob)
```

In [649]:
```
print('Average AUC Score : {0:0.2f}' .format(sum(auc_test)/len(auc_test)))
```

```
Average AUC Score : 0.99
```

In [650]:
```
plt.plot(auc_train)
plt.plot(auc_test)
plt.legend(('AUC_test','AUC_train'))
plt.title('AUC curve')
plt.show()
```



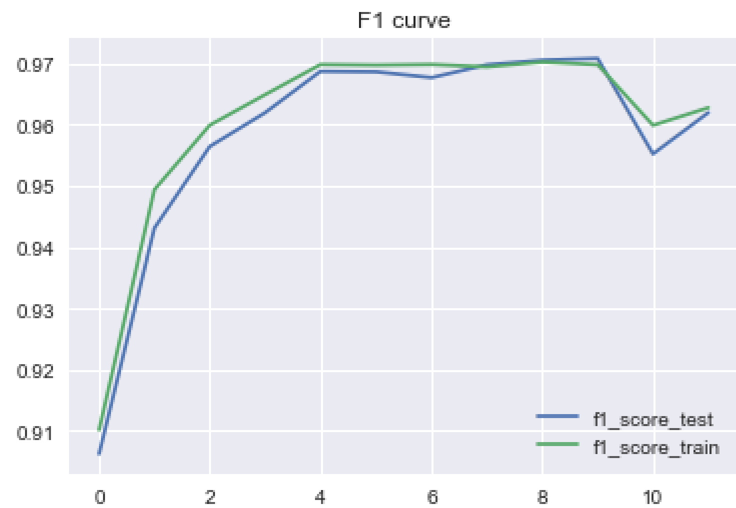
Model 5: Multi Layer Perceptron(Binary Classification)

In [651]:
```
from sklearn.neural_network import MLPClassifier

f1_score_train = []
f1_score_test = []

recall_train = []
recall_test = []

chunksize = 5000

# After tuning the parameter alpha, we get the best model for alpha = 1 and the h

estimator = MLPClassifier(hidden_layer_sizes=(2056,),alpha=1)
for i,chunk in enumerate(pd.read_csv('prosper_final_normal.csv', chunksize=chunks
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk,classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,pos_label=0)
    f_ta = f1_score(y_train,y_pred_train,pos_label=0)

    rc_train=recall_score(y_train, y_pred_train, pos_label=0)
    rc_test=recall_score(y_test, y_pred_test, pos_label=0)


    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)

    recall_train.append(rc_train)
    recall_test.append(rc_test)
```

In [652]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test

print('Average Recall score is : {0:0.2f}'.format(sum(recall_test)/len(recall_tes
```

```
Average f1 score :  0.96
Average Recall score is : 0.93
```

In [653]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()

plt.plot(recall_test)
plt.plot(recall_train)
plt.legend(('recall_test','recall_train'))
plt.title('Recall curve')
plt.show()
```



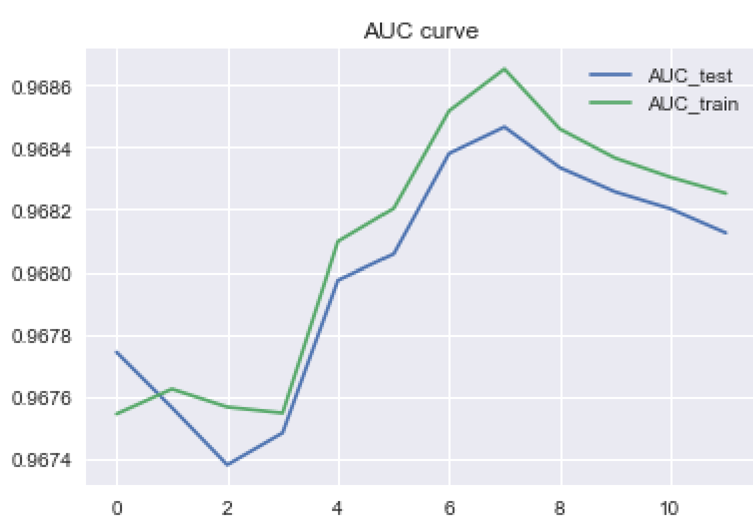
Model 6: BernoulliNB(Binary Classification)

In [654]:
```
from sklearn.naive_bayes import BernoulliNB

auc_train=[]
auc_test=[]

chunksize = 5000

estimator = BernoulliNB()
for i,chunk in enumerate(pd.read_csv('prosper_final_normal.csv', chunksize=chunks
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk,classes=np.unique(y_test))

    y_pred_test = estimator.predict_proba(X_test)[:,1]
    y_pred_train = estimator.predict_proba(X_train)[:,1]

    auc_tr=roc_auc_score(y_train,y_pred_train,average='micro')
    auc_te=roc_auc_score(y_test,y_pred_test, average='micro')

    auc_train.append(auc_tr)
    auc_test.append(auc_te)
```

In [655]:
```
print('Average AUC Score : {0:0.2f}' .format(sum(auc_test)/len(auc_test)))
```

```
Average AUC Score : 0.97
```

In [656]:
```
plt.plot(auc_train)
plt.plot(auc_test)
plt.legend(('AUC_test','AUC_train'))
plt.title('AUC curve')
plt.show()
```



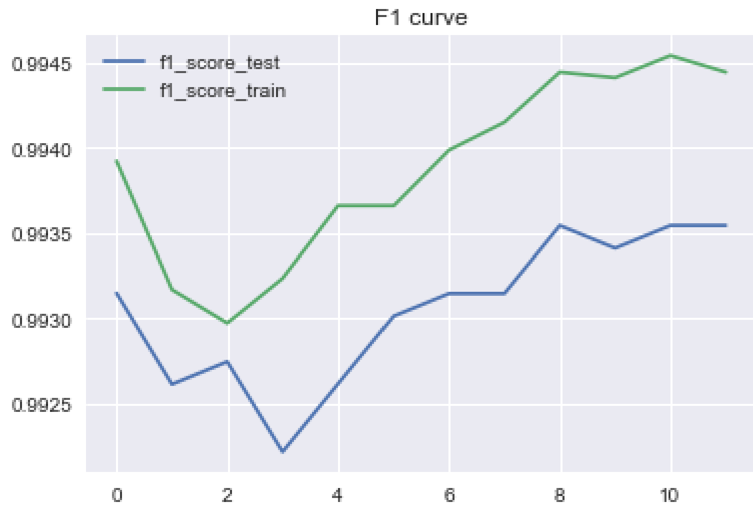
Model 7: Random Forest(Binary Classification)

In [657]:
```
from sklearn.ensemble import RandomForestClassifier

f1_score_train = []
f1_score_test = []

recall_train = []
recall_test = []

chunksize = 5000

estimator = RandomForestClassifier(n_estimators = 50, warm_start=True)
for i,chunk in enumerate(pd.read_csv('prosper_final_normal.csv', chunksize=chunks
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]

    estimator.fit(X_chunk,y_chunk)
    estimator.set_params(n_estimators = 100+50*i)

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test, pos_label=0)
    f_ta = f1_score(y_train,y_pred_train,pos_label=0)

    rc_train=recall_score(y_train,y_pred_train,pos_label=0)
    rc_test=recall_score(y_test,y_pred_test, pos_label=0)

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)

    recall_train.append(rc_train)
    recall_test.append(rc_test)
```

In [658]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test

print('Average Recall score is : {0:0.2f}'.format(sum(recall_test)/len(recall_tes
```

```
Average f1 score :  0.99
Average Recall score is : 0.99
```

In [659]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()

plt.plot(recall_test)
plt.plot(recall_train)
plt.legend(('recall_test','recall_train'))
plt.title('Recall curve')
plt.show()
```



MULTI - CLASS CLASSIFICATION

In [660]:
```
y_new=(pl['LoanStatus'])
```

In [661]:
```
y_new=y_new[y_new!='Current']
```

In [662]:
```
y_new.value_counts()
```

Out[662]:
```
Completed                 38074
Chargedoff                11992
Defaulted                  5018
Past Due (1-15 days)        806
Past Due (31-60 days)       363
Past Due (61-90 days)       313
Past Due (91-120 days)      304
Past Due (16-30 days)       265
FinalPaymentInProgress      205
Past Due (>120 days)         16
Cancelled                     5
Name: LoanStatus, dtype: int64
```

In [663]:
```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(y_new)
encoder_Y = encoder.transform(y_new)
```

In [664]:
```
unique, counts = np.unique(encoder_Y, return_counts=True)

print (np.asarray((unique, counts)))
```

```
[[    0     1     2     3     4     5     6     7     8     9    10]
 [    5 11992 38074  5018   205   806   265   363   313   304    16]]
```

In [665]:
```
y_m = pd.DataFrame(encoder_Y, index = final.index)
```

In [666]:
```
y_m.head()
```

Out[666]:

	0
0	2
2	2
11	2
12	5
15	3

In [667]:
```
final_loan_m_normal = pd.concat((X,y_m), axis=1, join='outer')
```

In [668]:
```
final_loan_m_normal.head()
```

Out[668]:

	0	1	2	3	4	5	6	7	8
0	-0.156766	-0.667834	-0.539023	-0.665467	-0.088074	-0.176553	-0.213943	0.166661	-(
2	-0.156766	0.669629	0.906298	0.603059	-0.088074	-0.176553	-0.213943	0.166661	-(
11	-0.156766	-0.836595	-0.854029	-0.858233	-0.088074	-0.176553	-0.213943	0.166661	-(
12	-0.156766	-0.502486	-0.718144	-0.721431	-0.832885	-0.897949	-1.033395	0.891813	-
15	-0.156766	1.526296	1.433779	1.445012	2.043330	2.012879	0.845512	-2.234661	-(

In [669]:
```
final_loan_m_normal.to_csv('prosper_final_m_normal.csv')
```

In [670]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y_m, test_size=0.20, random_stat
```

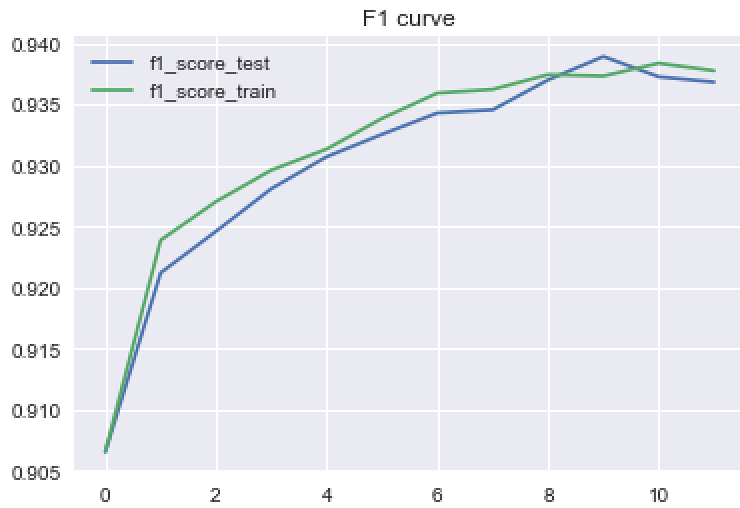
Model 8: SVM WITH L1(Multi Classification)

In [671]:
```
from sklearn.metrics import matthews_corrcoef

f1_score_train = []
f1_score_test = []

chunksize = 5000

estimator = SGDClassifier(loss='hinge', alpha=0.0001,penalty='l1', l1_ratio=1)
for i,chunk in enumerate(pd.read_csv('prosper_final_m_normal.csv', chunksize=chun
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,average='micro')
    f_ta = f1_score(y_train,y_pred_train,average='micro')

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)
```

In [672]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test
```

```
Average f1 score :  0.93
```

In [673]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()
```



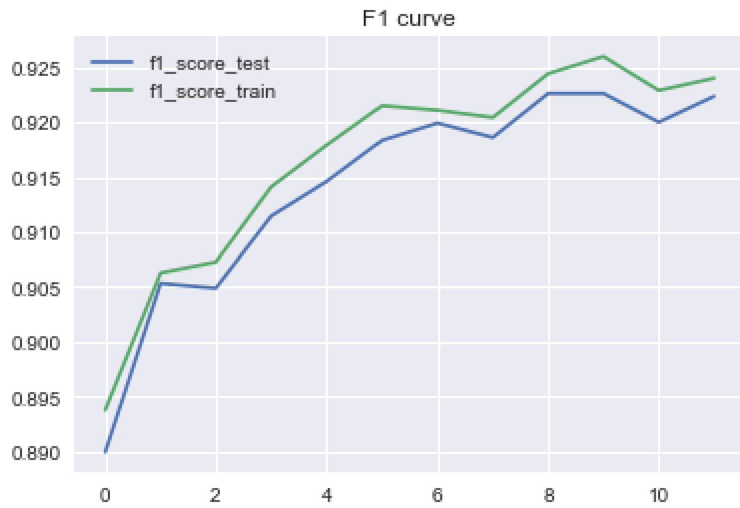
Model 9: SVM with L2(Multi Classification)

In [674]:
```
from sklearn.metrics import matthews_corrcoef
f1_score_train = []
f1_score_test = []

chunksize = 5000

estimator = SGDClassifier(loss='hinge', alpha=0.0001,penalty='l2', l1_ratio=0)
for i,chunk in enumerate(pd.read_csv('prosper_final_m_normal.csv', chunksize=chun
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,average='micro')
    f_ta = f1_score(y_train,y_pred_train,average='micro')

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)
```

In [675]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test
```

```
Average f1 score :  0.91
```

In [676]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()
```



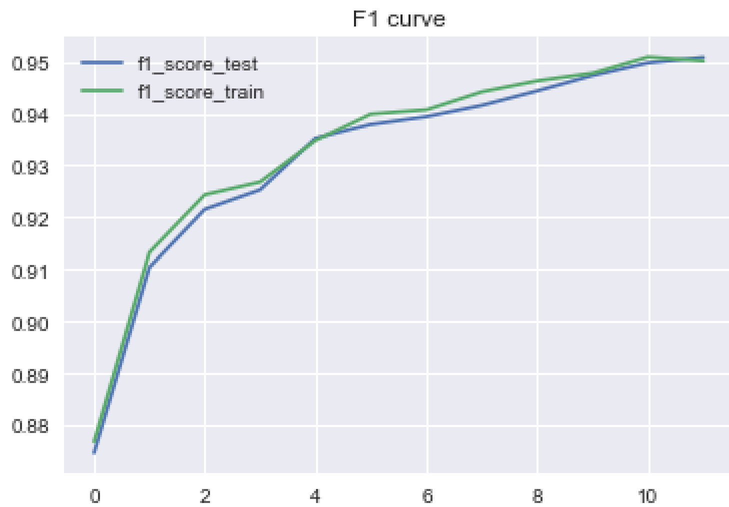
Model 10: Multi Layer Perceptron(Multi Classification)

In [677]:
```
from sklearn.neural_network import MLPClassifier

f1_score_train = []
f1_score_test = []


chunksize = 5000

estimator = MLPClassifier(hidden_layer_sizes=(2056,),alpha=0.1)
for i,chunk in enumerate(pd.read_csv('prosper_final_m_normal.csv', chunksize=chun
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk,np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,average='micro')
    f_ta = f1_score(y_train,y_pred_train,average='micro')

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)
```

In [678]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test
```

```
Average f1 score :  0.93
```

In [679]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()
```



PRINCIPAL COMPONENT ANALYSIS(PCA)

In [680]:
```
from sklearn.decomposition import PCA

pca = PCA(0.95)
pca.fit(X)
```

Out[680]:
```
PCA(copy=True, iterated_power='auto', n_components=0.95, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [681]:
```
pca.n_components_
```

Out[681]:
```
186
```

In [682]:
```
X = pca.transform(X)

X_f=pd.DataFrame(X, index=final.index)
```

In [683]:
```
final_loan = pd.concat((X_f,y), axis=1, join='outer')
```

In [684]:
```
final_loan.to_csv('prosper_final.csv')
```

In [685]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_f,y, test_size=0.20, random_stat
```

In [686]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45888 entries, 52775 to 27871
Columns: 186 entries, 0 to 185
dtypes: float64(186)
memory usage: 65.5 MB
```

In [687]:
```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11473 entries, 1872 to 3095
Columns: 186 entries, 0 to 185
dtypes: float64(186)
memory usage: 16.4 MB
```

BINARY CLASS CLASSIFICATION

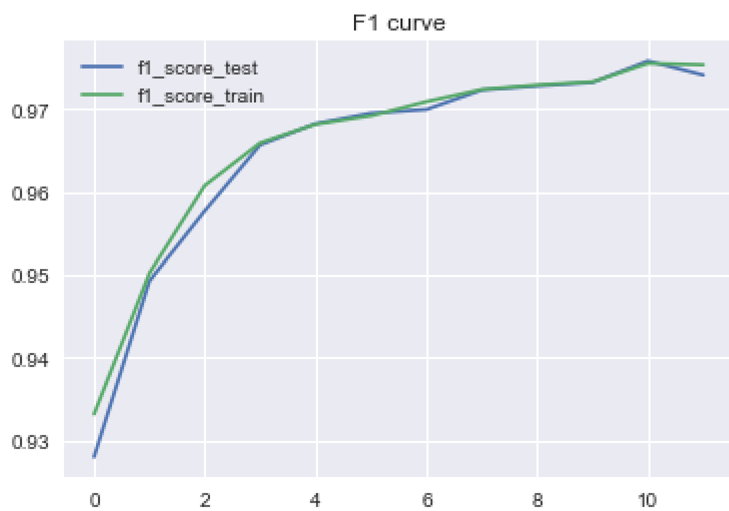
Model 11: SVM with L1(PCA/Binary Classification)

In [688]:
```

f1_score_train = []
f1_score_test = []

recall_train = []
recall_test = []

chunksize = 5000

estimator = SGDClassifier(loss='hinge', penalty='l1', l1_ratio=1)
for i,chunk in enumerate(pd.read_csv('prosper_final.csv', chunksize=chunksize)):
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,pos_label=0)
    f_ta = f1_score(y_train,y_pred_train, pos_label=0)

    rc_test=recall_score(y_test,y_pred_test, pos_label=0)
    rc_train=recall_score(y_train,y_pred_train, pos_label=0)

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)

    recall_train.append(rc_train)
    recall_test.append(rc_test)
```

In [689]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test

print('Average Recall score is : {0:0.2f}'.format(sum(recall_test)/len(recall_tes
```

```
Average f1 score :  0.96
Average Recall score is : 0.95
```

In [690]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()

plt.plot(recall_test)
plt.plot(recall_train)
plt.legend(('recall_test','recall_train'))
plt.title('Recall curve')
plt.show()
```



Model 12: Logistic Regression with L1(PCA/Binary Classification)

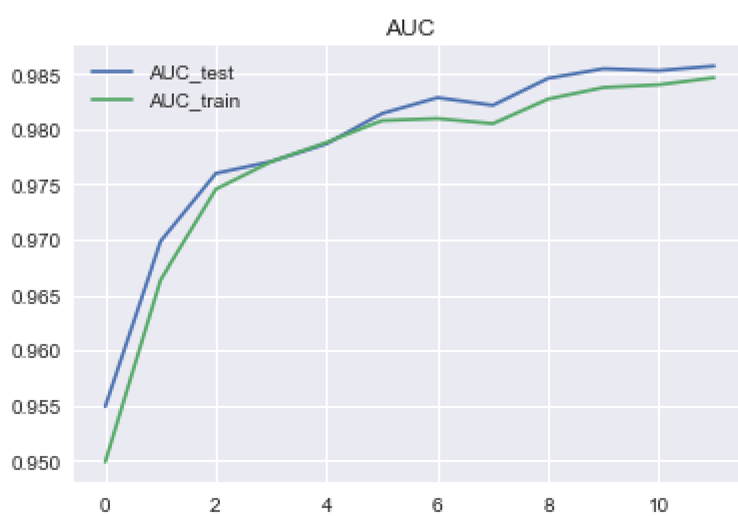
In [691]:
```
#Logistic Regression

auc_train = []
auc_test = []

chunksize = 5000

estimator = SGDClassifier(loss='log', penalty='l1', l1_ratio=1)
for i,chunk in enumerate(pd.read_csv('prosper_final.csv', chunksize=chunksize)):
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict_proba(X_test)[:,1]
    y_pred_train = estimator.predict_proba(X_train)[:,1]

    auc_tr=roc_auc_score(y_train,y_pred_train,average='micro')
    auc_te=roc_auc_score(y_test,y_pred_test, average='micro')

    auc_train.append(auc_tr)
    auc_test.append(auc_te)
```

```
C:\Program Files\Anaconda3\lib\site-packages\sklearn\linear_model\base.py:352:
RuntimeWarning: overflow encountered in exp
  np.exp(prob, prob)
```

In [692]:
```
print('Average AUC Score : {0:0.2f}' .format(sum(auc_test)/len(auc_test)))
```

```
Average AUC Score : 0.98
```

In [693]:
```
plt.plot(auc_train)
plt.plot(auc_test)
plt.legend(('AUC_test','AUC_train'))
plt.title('AUC')
plt.show()
```



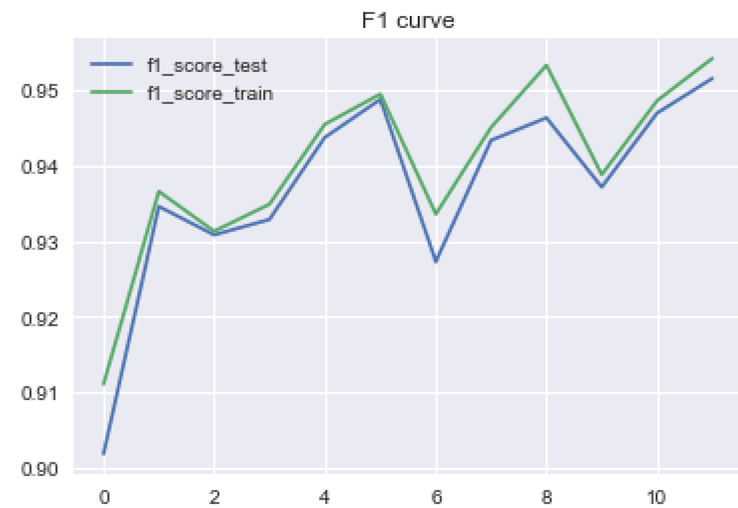
Model 13:SVM with L2(PCA/Binary Classification)

In [694]:
```
f1_score_train = []
f1_score_test = []

recall_train = []
recall_test = []

chunksize = 5000

estimator = SGDClassifier(loss='hinge', penalty='l2', l1_ratio=0)
for i,chunk in enumerate(pd.read_csv('prosper_final.csv', chunksize=chunksize)):
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test, pos_label=0)
    f_ta = f1_score(y_train,y_pred_train, pos_label=0)

    rc_test = recall_score(y_test,y_pred_test, pos_label=0)
    rc_train=recall_score(y_train, y_pred_train,pos_label=0)

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)

    recall_train.append(rc_train)
    recall_test.append(rc_test)
```

In [695]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test

print('Average Recall score is : {0:0.2f}'.format(sum(recall_test)/len(recall_tes
```

```
Average f1 score :  0.94
Average Recall score is : 0.93
```

In [696]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()

plt.plot(recall_test)
plt.plot(recall_train)
plt.legend(('recall_test','recall_train'))
plt.title('Recall curve')
plt.show()
```



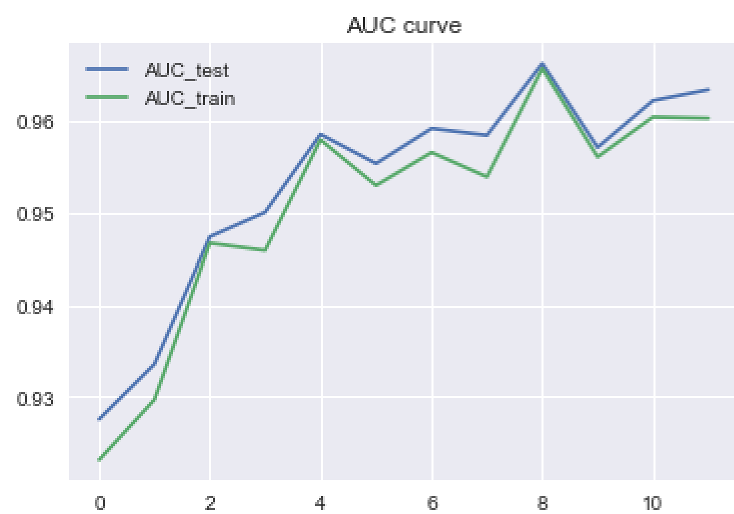
Model 14:Logistic Regression with L2 (PCA/Binary Classification)

In [697]:
```
auc_train = []
auc_test = []

chunksize = 5000

estimator = SGDClassifier(loss='log', penalty='l2', l1_ratio=0)
for i,chunk in enumerate(pd.read_csv('prosper_final.csv', chunksize=chunksize)):
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    auc_tr=roc_auc_score(y_train,y_pred_train,average='micro')
    auc_te=roc_auc_score(y_test,y_pred_test, average='micro')

    auc_train.append(auc_tr)
    auc_test.append(auc_te)
```

In [698]:
```
print('Average AUC Score : {0:0.2f}' .format(sum(auc_test)/len(auc_test)))
```

```
Average AUC Score : 0.95
```

In [699]:
```
plt.plot(auc_train)
plt.plot(auc_test)
plt.legend(('AUC_test','AUC_train'))
plt.title('AUC curve')
plt.show()
```



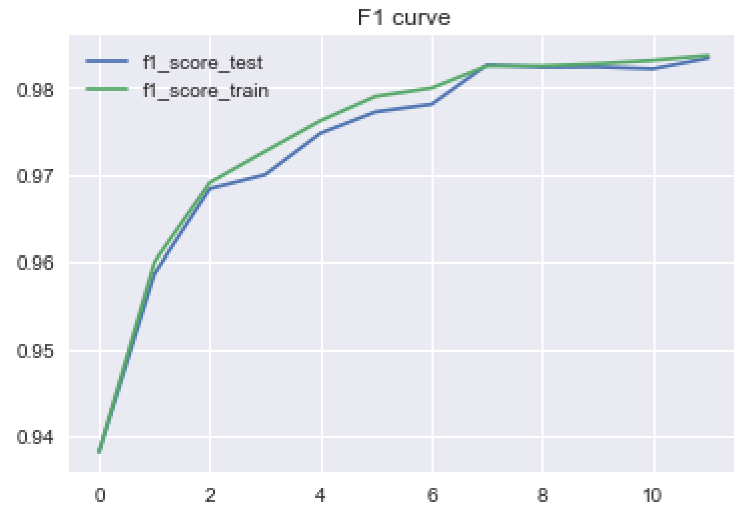
Model 15:Multi Layer Perceptron (PCA/Binary Classification)

In [700]:
```
from sklearn.neural_network import MLPClassifier

f1_score_train = []
f1_score_test = []

recall_test = []
recall_train = []

chunksize = 5000

estimator = MLPClassifier(hidden_layer_sizes=(2056,),alpha=0.1)
for i,chunk in enumerate(pd.read_csv('prosper_final.csv', chunksize=chunksize)):
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk,classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test)
    f_ta = f1_score(y_train,y_pred_train)

    rc_train=recall_score(y_train, y_pred_train, pos_label=0)
    rc_test=recall_score(y_test, y_pred_test, pos_label=0)

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)

    recall_train.append(rc_train)
    recall_test.append(rc_test)
```

In [701]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test

print('Average Recall score is : {0:0.2f}'.format(sum(recall_test)/len(recall_tes
```

```
Average f1 score :  0.97
Average Recall score is : 0.90
```

In [702]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()

plt.plot(recall_test)
plt.plot(recall_train)
plt.legend(('recall_test','recall_train'))
plt.title('Recall curve')
plt.show()
```



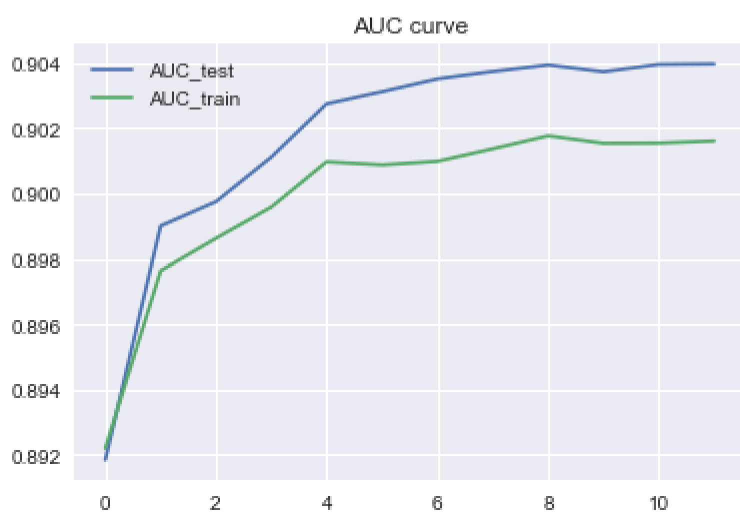
Model 16:BernoulliNB (PCA/Binary Classification)

In [703]:
```
from sklearn.naive_bayes import BernoulliNB

auc_train = []
auc_test = []

chunksize = 5000

estimator = BernoulliNB()
for i,chunk in enumerate(pd.read_csv('prosper_final.csv', chunksize=chunksize)):
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk,classes=np.unique(y_test))

    y_pred_test = estimator.predict_proba(X_test)[:,1]
    y_pred_train = estimator.predict_proba(X_train)[:,1]

    auc_tr=roc_auc_score(y_train,y_pred_train,average='micro')
    auc_te=roc_auc_score(y_test,y_pred_test, average='micro')

    auc_train.append(auc_tr)
    auc_test.append(auc_te)
```

In [704]:
```
print('Average AUC Score : {0:0.2f}' .format(sum(auc_test)/len(auc_test)))
```

```
Average AUC Score : 0.90
```

In [705]:
```
plt.plot(auc_train)
plt.plot(auc_test)
plt.legend(('AUC_test','AUC_train'))
plt.title('AUC curve')
plt.show()
```



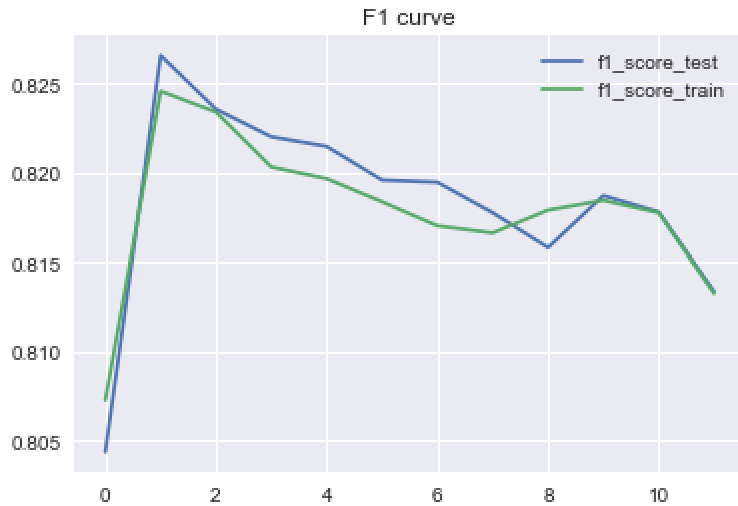
Model 17:Random Forest (PCA/Binary Classification)

In [706]:
```
from sklearn.ensemble import RandomForestClassifier

f1_score_train = []
f1_score_test = []

recall_train = []
recall_test = []

chunksize = 5000

estimator = RandomForestClassifier(n_estimators = 50, warm_start=True)
for i,chunk in enumerate(pd.read_csv('prosper_final.csv', chunksize=chunksize)):
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]

    estimator.fit(X_chunk,y_chunk)
    estimator.set_params(n_estimators = 100+50*i)

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test, pos_label=0)
    f_ta = f1_score(y_train,y_pred_train,pos_label=0)

    rc_train=recall_score(y_train,y_pred_train,pos_label=0)
    rc_test=recall_score(y_test,y_pred_test, pos_label=0)

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)

    recall_train.append(rc_train)
    recall_test.append(rc_test)
```

In [707]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test

print('Average Recall score is : {0:0.2f}'.format(sum(recall_test)/len(recall_tes
```

```
Average f1 score :  0.82
Average Recall score is : 0.70
```

In [708]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()

plt.plot(recall_test)
plt.plot(recall_train)
plt.legend(('recall_test','recall_train'))
plt.title('Recall curve')
plt.show()
```



MULTI - CLASS CLASSIFICATION

In [709]:
```
y_new=(pl['LoanStatus'])
```

In [710]:
```
y_new=y_new[y_new!='Current']
```

In [711]:
```
y_new.value_counts()
```

Out[711]:
```
Completed                 38074
Chargedoff                11992
Defaulted                  5018
Past Due (1-15 days)        806
Past Due (31-60 days)       363
Past Due (61-90 days)       313
Past Due (91-120 days)      304
Past Due (16-30 days)       265
FinalPaymentInProgress      205
Past Due (>120 days)         16
Cancelled                     5
Name: LoanStatus, dtype: int64
```

In [712]:
```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(y_new)
encoder_Y = encoder.transform(y_new)
```

In [713]:
```
unique, counts = np.unique(encoder_Y, return_counts=True)

print (np.asarray((unique, counts)))
```

```
[[    0     1     2     3     4     5     6     7     8     9    10]
 [    5 11992 38074  5018   205   806   265   363   313   304    16]]
```

In [714]:
```
y_m = pd.DataFrame(encoder_Y, index = final.index)
```

In [715]:
```
y_m.head()
```

Out[715]:

	0
0	2
2	2
11	2
12	5
15	3

In [716]:
```
final_loan_m = pd.concat((X_f,y_m), axis=1, join='outer')
```

In [717]:
```
final_loan_m.head()
```

Out[717]:

	0	1	2	3	4	5	6	7	8
0	0.470168	2.097374	-1.681590	3.217867	-1.205997	0.638174	0.819883	0.331106	1
2	-3.583566	2.698086	-0.023599	0.030426	-1.472195	-1.134283	1.841556	-4.181326	0
11	-2.818261	3.890026	-3.161130	0.300431	0.618288	-0.575934	1.212722	0.890014	-
12	-0.802027	-0.562779	-1.722157	-3.494290	1.625004	-0.898534	-1.920570	0.169104	0
15	-3.044296	-5.508623	-0.342920	-0.822520	0.091234	-0.891753	-1.734992	1.016163	0

In [718]:
```
final_loan_m.to_csv('prosper_final_m.csv')
```

In [719]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X_f,y_m, test_size=0.20, random_st
```

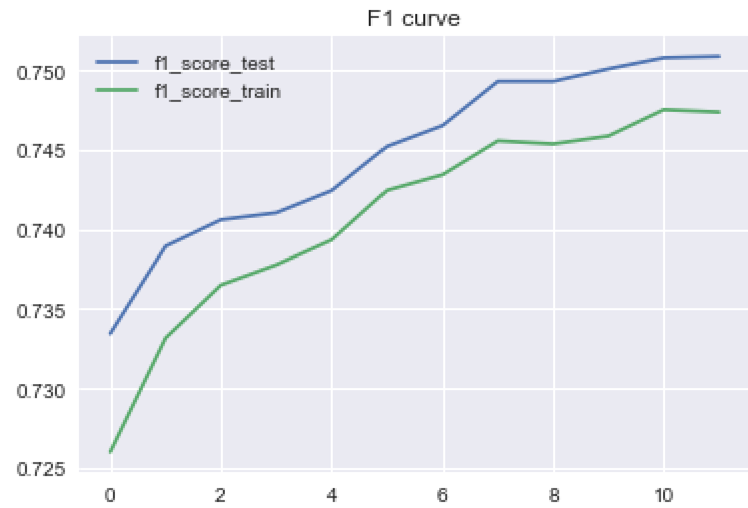
Model 18:SVM WITH L1 (PCA/Multi Classification)

In [720]:
```
f1_score_train = []
f1_score_test = []


chunksize = 5000

estimator = SGDClassifier(loss='hinge', penalty='l1', l1_ratio=1, alpha=0.1)
for i,chunk in enumerate(pd.read_csv('prosper_final_m.csv', chunksize=chunksize))
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,average='micro')
    f_ta = f1_score(y_train,y_pred_train,average='micro')

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)
```

In [721]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test
```

```
Average f1 score :  0.74
```

In [722]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()
```



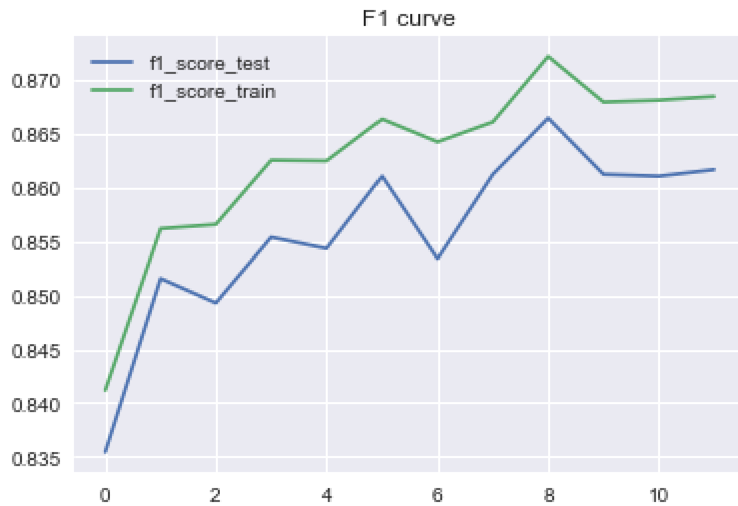
Model 19:SVM with L2 (PCA/Multi Classification)

In [723]:
```
f1_score_train = []
f1_score_test = []

chunksize = 5000

estimator = SGDClassifier(loss='hinge', penalty='l2', l1_ratio=0)
for i,chunk in enumerate(pd.read_csv('prosper_final_m.csv', chunksize=chunksize))
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,average='micro')
    f_ta = f1_score(y_train,y_pred_train,average='micro')

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)
```

In [724]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test
```

```
Average f1 score :  0.86
```

In [725]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()
```



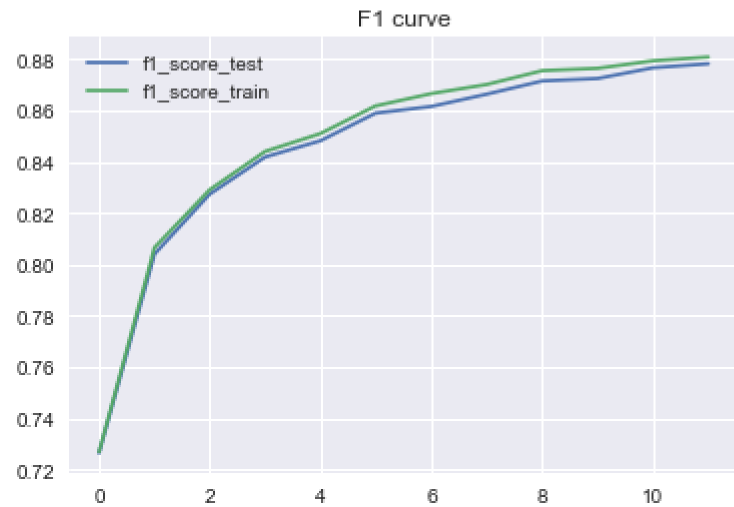
Model 20:Multi Layer Perceptron(PCA/Multi Classification)

In [726]:
```
from sklearn.neural_network import MLPClassifier

f1_score_train = []
f1_score_test = []

chunksize = 5000

estimator = MLPClassifier()
for i,chunk in enumerate(pd.read_csv('prosper_final_m.csv', chunksize=chunksize))
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk,classes=np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,average='micro')
    f_ta = f1_score(y_train,y_pred_train,average='micro')

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)
```

In [727]:
```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test
```

```
Average f1 score :  0.84
```

In [728]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()
```



RESULT AND CONCLUSION

BEST MODEL FOR BINARY CLASSIFICATION

Model 12: Logistic Regression with L1(PCA/Binary Classification)

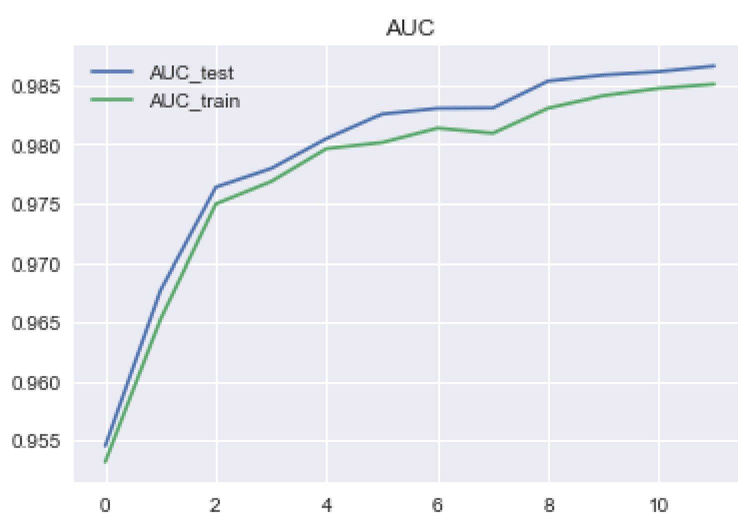
In [517]:
```
#Logistic Regression

auc_train = []
auc_test = []

chunksize = 5000

estimator = SGDClassifier(loss='log', penalty='l1', l1_ratio=1)
for i,chunk in enumerate(pd.read_csv('prosper_final.csv', chunksize=chunksize)):
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk = chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk, classes=np.unique(y_test))

    y_pred_test = estimator.predict_proba(X_test)[:,1]
    y_pred_train = estimator.predict_proba(X_train)[:,1]

    auc_tr=roc_auc_score(y_train,y_pred_train,average='micro')
    auc_te=roc_auc_score(y_test,y_pred_test, average='micro')

    auc_train.append(auc_tr)
    auc_test.append(auc_te)
```

```
C:\Program Files\Anaconda3\lib\site-packages\sklearn\linear_model\base.py:352:
RuntimeWarning: overflow encountered in exp
  np.exp(prob, prob)
```

In [518]:
```
print('Average AUC Score : {0:0.2f}' .format(sum(auc_test)/len(auc_test)))
```

```
Average AUC Score : 0.98
```

In [519]:
```
plt.plot(auc_train)
plt.plot(auc_test)
plt.legend(('AUC_test','AUC_train'))
plt.title('AUC')
plt.show()
```



Logistic regression model with L1 is the best model performer to predict the classes for the binary type classification. The average Area under the curve score indicates that the model correctly predicts the classes with a probability of around 0.98. The curve is gradually increasing with increase in number of iterations, which suggests that the model is a good performer in predicting the classes accurately. The auc score of 0.98 is same prior and after applying pca which suggests that although the score being the same, pca aids in extracting only the required features which best explains the model thereby reducing the complexity of the model.

Model 10: Multi Layer Perceptron(Multi Classification)

In [541]:

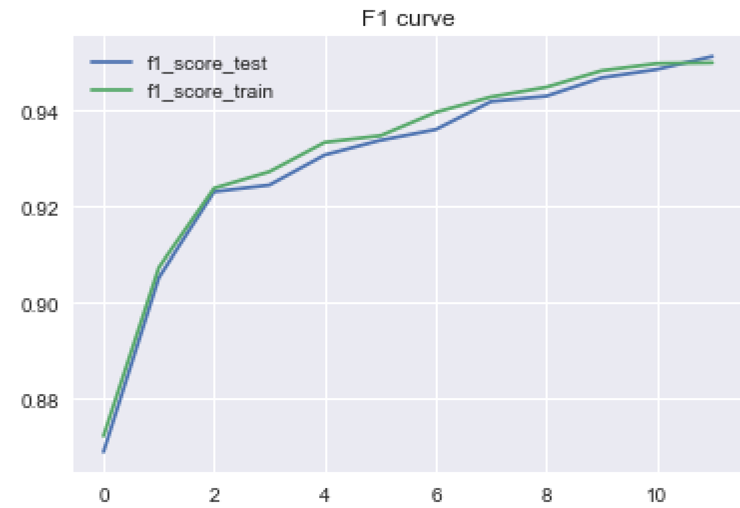
```
from sklearn.neural_network import MLPClassifier

f1_score_train = []
f1_score_test = []

chunksize = 5000

estimator = MLPClassifier(hidden_layer_sizes=(2056,),alpha=0.1)
for i,chunk in enumerate(pd.read_csv('prosper_final_m_normal.csv', chunksize=chun
    X_chunk = chunk.iloc[:,1:-1]
    y_chunk= chunk.iloc[:,-1]
    estimator.partial_fit(X_chunk,y_chunk,np.unique(y_test))

    y_pred_test = estimator.predict(X_test)
    y_pred_train = estimator.predict(X_train)

    f_te = f1_score(y_test,y_pred_test,average='micro')
    f_ta = f1_score(y_train,y_pred_train,average='micro')

    f1_score_train.append(f_ta)
    f1_score_test.append(f_te)
```

In [542]:

```
print('Average f1 score :  {0:0.2f}' .format(sum(f1_score_test)/len(f1_score_test
```

```
Average f1 score :  0.93
```

In [543]:
```
plt.plot(f1_score_test)
plt.plot(f1_score_train)
plt.legend(('f1_score_test','f1_score_train'))
plt.title('F1 curve')
plt.show()
```



For the multi-class classification, MLP Classifier has the highest f1 score among the classifiers. It is the weighted average of precision and recall score for each class in a multi class scenario. F1 score indicates to what extent the model correctly predicts all the classes (multi class case). Closer the score to 1, better is the model performance. Although SVM with L1 penalty has a similar score, comparing the graphs of both the classifiers over a number of iterations we can infer that the MLP has a comparatively smoother curve, which indicates its robustness towards partial fit.

It was noted that the f1 score for MLP Classifier reduced when the features where trained into the model after applying PCA. This might be for the reason that, since the target variable has 11 unbalanced classes, it probably requires initial number of features (prior application of pca) to have a good performance on the data.