In [145]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import warnings

from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

In [146]:
```
ın2011 = pd.read_csv('Downloads/LendingClub/LoanStats2011.csv',  header=1, c
ın2013 = pd.read_csv('Downloads/LendingClub/LoanStats2013.csv',  header=1, c
ın2014 = pd.read_csv('Downloads/LendingClub/LoanStats2014.csv',  header=1, c
ın2015 = pd.read_csv('Downloads/LendingClub/LoanStats2015.csv', header=1, ch

_2016 = pd.read_csv('Downloads/LendingClub/LoanStats_2016Q1.csv', header=1,
_2016 = pd.read_csv('Downloads/LendingClub/LoanStats_2016Q2.csv', header=1,
_2016 = pd.read_csv('Downloads/LendingClub/LoanStats_2016Q3.csv', header=1,
_2016 = pd.read_csv('Downloads/LendingClub/LoanStats_2016Q4.csv', header=1,

_2017 = pd.read_csv('Downloads/LendingClub/LoanStats_2017Q1.csv', header=1,
_2017 = pd.read_csv('Downloads/LendingClub/LoanStats_2017Q2.csv', header=1,
_2017 = pd.read_csv('Downloads/LendingClub/LoanStats_2017Q3.csv', header=1,
_2017 = pd.read_csv('Downloads/LendingClub/LoanStats_2017Q4.csv', header=1,

_2018 = pd.read_csv('Downloads/LendingClub/LoanStats_2018Q1.csv', header=1,
_2018 = pd.read_csv('Downloads/LendingClub/LoanStats_2018Q2.csv', header=1,
_2018 = pd.read_csv('Downloads/LendingClub/LoanStats_2018Q3.csv', header=1,
_2018 = pd.read_csv('Downloads/LendingClub/LoanStats_2018Q4.csv', header=1,

_2019 = pd.read_csv('Downloads/LendingClub/LoanStats_2019Q1.csv', header=1,
```

In [147]:
```
chunk_list = []
files = [loan2011, loan2013, loan2014, loan2015,
         Q1_2016, Q2_2016, Q3_2016, Q3_2016,
         Q1_2017, Q2_2017, Q3_2017, Q3_2017,
         Q1_2018, Q2_2018, Q3_2018, Q4_2018, Q1_2019]

for file in files:
    for chunk in file:
        chunk_list.append(chunk)

df_concat = pd.concat(chunk_list)
```

In [149]:
```
column_names = ['loan_amnt', 'funded_amnt', 'term', 'int_rate', 'installmen
                'emp_length', 'annual_inc', 'issue_d', 'loan_status', 'addr_
                'delinq_2yrs', 'open_acc', 'pub_rec', 'total_acc', 'total_py
                 'tot_cur_bal', 'total_bal_il', 'revol_bal']

data = df_concat.iloc[:, 2:][column_names]
```

In [150]:
```
data.head(5)
```

Out[150]:

	loan_amnt	funded_amnt	term	int_rate	installment	grade	emp_length	annual_inc	issue_d
0	5000.0	5000.0	36 months	10.65%	162.87	B	10+ years	24000.0	Dec-2011
1	2500.0	2500.0	60 months	15.27%	59.83	C	< 1 year	30000.0	Dec-2011
2	2400.0	2400.0	36 months	15.96%	84.33	C	10+ years	12252.0	Dec-2011
3	10000.0	10000.0	36 months	13.49%	339.31	C	10+ years	49200.0	Dec-2011
4	3000.0	3000.0	60 months	12.69%	67.79	B	1 year	80000.0	Dec-2011

In [151]:
```
data.isnull().sum()
```

Out[151]:
```
loan_amnt            31
funded_amnt          31
term                 31
int_rate             31
installment          31
grade                31
emp_length       141440
annual_inc           35
issue_d              31
loan_status          31
addr_state           31
dti                1749
delinq_2yrs          60
open_acc             60
pub_rec              60
total_acc            60
total_pymnt          31
tot_cur_bal       70307
total_bal_il     866160
revol_bal            31
dtype: int64
```

In [152]:
```
data = data.dropna(subset=['loan_amnt', 'funded_amnt', 'dti',
                           'annual_inc', 'emp_length'])
```

In [153]:
```
data['issue_y'] = data['issue_d'].apply(lambda x: str(x)[-4:])
```

Loan Volume

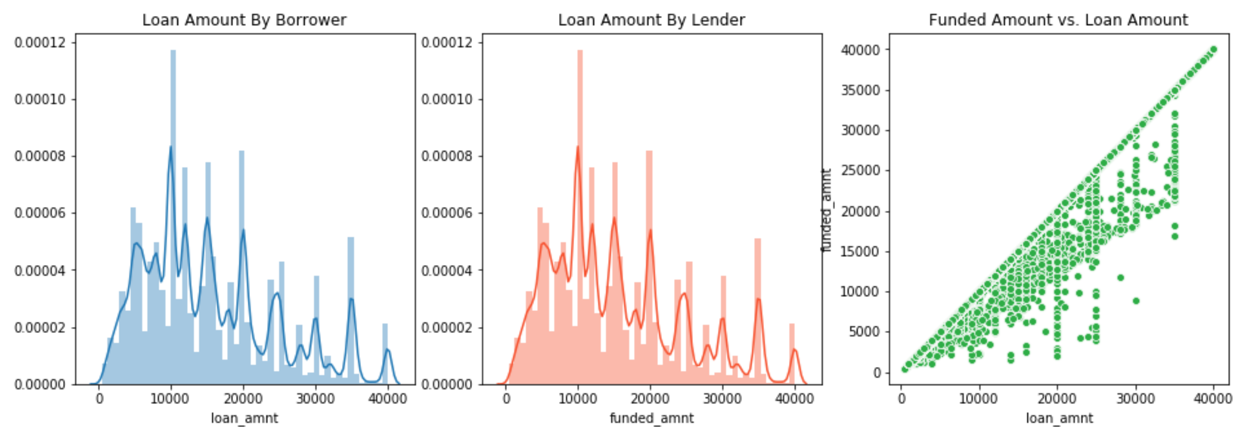
In [154]:
```
fig, ax = plt.subplots(3, 1, figsize = (14, 15))

g = data.groupby(['issue_y'])['loan_amnt'].count().reset_index()
sns.barplot('issue_y', 'loan_amnt', data=g, ax=ax[0])
ax[0].set_title('Total Loan Volume (Count)')

g = data.groupby(['issue_y'])['loan_amnt'].sum().reset_index()
sns.barplot('issue_y', 'loan_amnt', data=g, ax= ax[1])
ax[1].set_title('Total Loan Volume ($) by Year')

g = data.groupby(['issue_y'])['loan_amnt'].mean().reset_index()
sns.barplot('issue_y', 'loan_amnt', data=g, ax=ax[2])
ax[2].set_title('Average Loan Amnt Issued')
```

2018 has the highest loan volume in terms of count and $.

The avarege loan amount has an increasing trend overall.

In [155]:
```
, ax = plt.subplots(1, 3, figsize = (16, 5))
ns.distplot(data['loan_amnt'], ax= ax[0])
x[0].set_title('Loan Amount By Borrower')
ns.distplot(data['funded_amnt'], ax=ax[1], color="#F7522F")
x[1].set_title('Loan Amount By Lender')
ns.scatterplot(x = 'loan_amnt', y = 'funded_amnt', data = data, ax = ax[2],
x[2].set_title('Funded Amount vs. Loan Amount')

rint('Stats for funded amount:\n{}'.format( data['funded_amnt'].describe())
```

```
Stats for funded amount:
count    2.012615e+06
mean     1.536087e+04
std      9.242651e+03
min      5.000000e+02
25%      8.000000e+03
50%      1.350000e+04
75%      2.000000e+04
max      4.000000e+04
Name: funded_amnt, dtype: float64
```



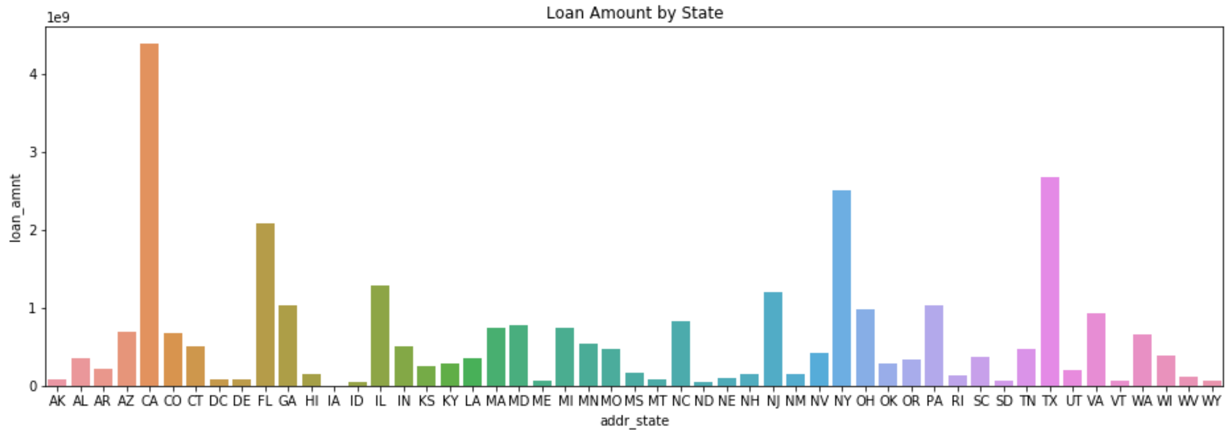
The amount asked for by borrower and issued by lender have similar distribution. It means that once the loan is approved, the borrowers are likely to get the full amount they had applied for.

The median loan applied is around 13,000 USD. Most loans issued range from 10,000 to 20,000 USD.

In [157]:
```
print('Top 4 Loan Amount{}'.format(data.groupby('addr_state')['loan_amnt'].
fig, ax = plt.subplots(1, 1, figsize = (16,5))

g = data.groupby('addr_state')['loan_amnt'].sum().reset_index()
sns.barplot(x = 'addr_state', y = 'loan_amnt', data = g, ax = ax)
ax.set_title('Loan Amount by State')
```

```
Top 4 Loan Amountaddr_state
CA    4.393444e+09
TX    2.683873e+09
NY    2.515649e+09
FL    2.078645e+09
Name: loan_amnt, dtype: float64
```

Out[157]:
```
Text(0.5, 1.0, 'Loan Amount by State')
```



California, Texas and Florida have the highest loan volume.

Loan quality

In [158]:
```
data['loan_status'].value_counts()
```

Out[158]:
```
Fully Paid                                             1030776
Current                                                 709099
Charged Off                                             243244
Late (31-120 days)                                       16267
In Grace Period                                           7560
Late (16-30 days)                                         2940
Does not meet the credit policy. Status:Fully Paid        1965
Does not meet the credit policy. Status:Charged Off        746
Default                                                     18
Name: loan_status, dtype: int64
```

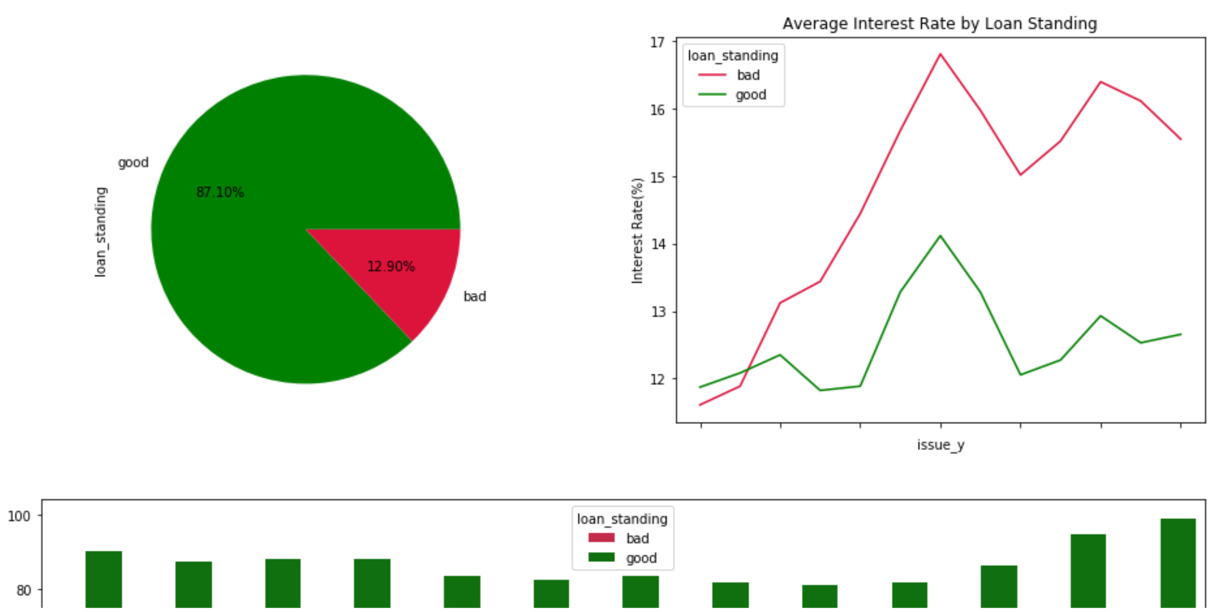
In [159]:
```
bad_status = ['Charged Off', 'Late (31-120 days)', 'Late (16-30 days)'
        'Does not meet the credit policy. Status:Charged Off', 'Default']

data['loan_standing'] = np.nan

def loan_standing(status):
    if status in bad_status:
        return 'bad'
    else:
        return 'good'

data['loan_standing'] = data['loan_status'].apply(loan_standing)
```

In [160]:
```
data['loan_target'] = np.where(data['loan_standing'] == 'good', 0, 1)
```

In [161]:
```
data['int_rate'] = data['int_rate'].apply(lambda x: float(x.rstrip("%")))
```

In [162]:
```
f = plt.figure( figsize=(16,12))

ax0 = f.add_subplot(221)
ax1 = f.add_subplot(222)
ax2 = f.add_subplot(212)

colors = ['crimson', 'g']
labels ="Good Loans", "Bad Loans"

data['loan_standing'].value_counts().plot.pie(autopct='%1.2f%%', ax=ax0, cc

data.groupby(['issue_y', 'loan_standing'])['int_rate'].mean().unstack().plc
ax1.set_title('Average Interest Rate by Loan Standing')
ax1.set_ylabel('Interest Rate(%)')

g = data.groupby(['issue_y', 'loan_standing'])['funded_amnt'].sum()
g = g.groupby(level = 0).apply(lambda x: x/x.sum() * 100).reset_index()
sns.barplot(x = 'issue_y', y = 'funded_amnt', hue = 'loan_standing', data=g
ax2.set(ylabel='% of Total Loans')
```

Out[162]: [Text(0, 0.5, '% of Total Loans')]



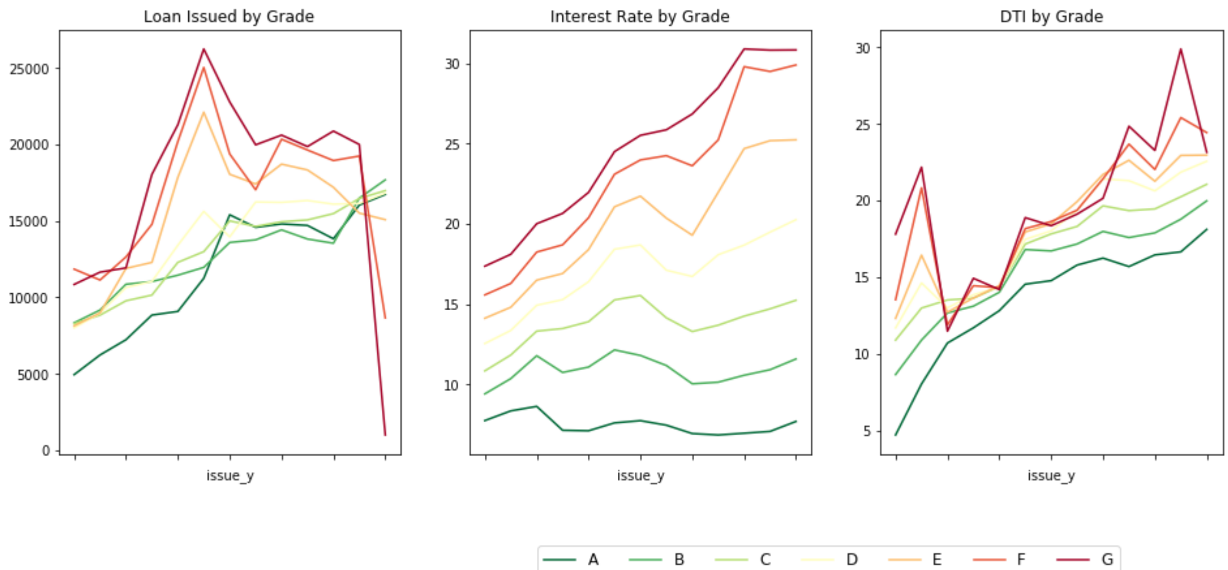
Bad loans account for 13% of the loans.

Not surprisingly, bad loans have much higher interest rate than good loans. The median interest rate for bad loans and good laons are ~15% and ~12% respectively.

Loans issued in 2015 and 2016 have the highest portions (~18%) of bad loans. We should keep in mind that current loans have risking of becoming bad loans. (For example, loans issued in 2019 do not have time to default yet since they were issued recently.)

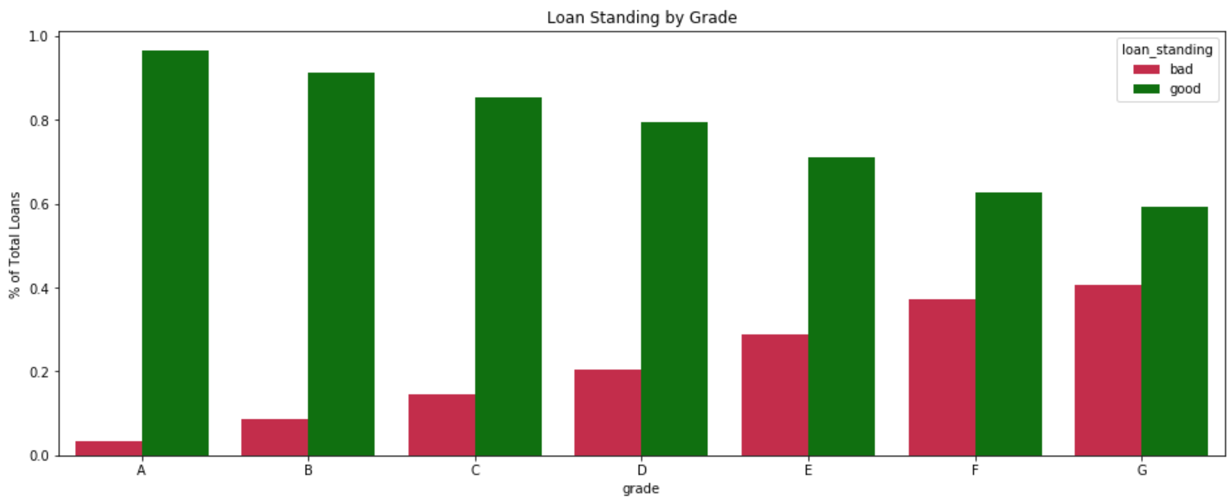
In [163]:
```
ax = plt.subplots(1, 3, figsize = (16, 6))
ıp = 'RdYlGn_r'

:a.groupby(['issue_y', 'grade'])['loan_amnt'].mean().unstack().plot(ax = ax[
0].set_title('Loan Issued by Grade')

:a.groupby(['issue_y', 'grade'])['int_rate'].mean().unstack().plot(ax = ax[1
1].set_title('Interest Rate by Grade')

:a.groupby(['issue_y', 'grade'])['dti'].mean().unstack().plot(ax = ax[2], co
2].set_title('DTI by Grade')

2].legend( bbox_to_anchor=(-1.0, -0.3, 1.7, 0.1), loc=5, prop={'size':12},
        ncol=7, mode="expand", borderaxespad=0.)
```

Out[163]: <matplotlib.legend.Legend at 0x1b7672a470>



In [164]:
```
f, ax = plt.subplots(1, 1, figsize = (16, 6))

g = data.groupby(['grade', 'loan_standing'])['loan_amnt'].count()
g = g.groupby(level = 0).apply(lambda x: x/x.sum()).reset_index()

sns.barplot(x = 'grade', y = 'loan_amnt', hue = 'loan_standing', data=g, pa
ax.set_title('Loan Standing by Grade')
ax.set_ylabel('% of Total Loans')
```

Out[164]:
```
Text(0, 0.5, '% of Total Loans')
```



There are more loans with bad credit ratings than good credit ratings.

Not surprisingly, the lower the loan grade, the higher the interest rate. Lending Club needs to charge the lender higher interest rate to mitigate the risk.

Loans with bad ratings have a higher chance of default, late payments and charge offs.

In [20]:
```
data['emp_length'].value_counts()
```

Out[20]:
```
10+ years    708401
2 years      193147
< 1 year     184370
3 years      171964
1 year       142056
5 years      133590
4 years      129886
6 years       97390
7 years       88921
8 years       88362
9 years       74532
Name: emp_length, dtype: int64
```

In [21]:
```
data = data.dropna(subset = ['annual_inc', 'emp_length'])
```

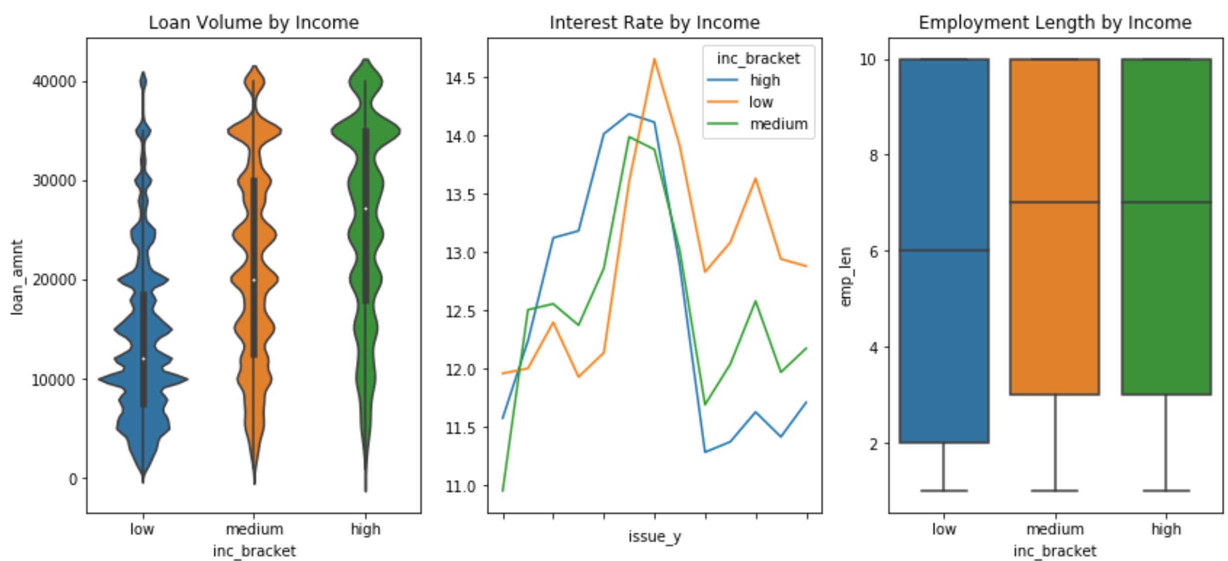
In [174]:
```
data['emp_len'] = data['emp_length'].str.extract('(\d+)').astype(int)
```

In [175]:
```
def income_bracket(income):
    if income <= 100000:
        return 'low'
    elif (income > 100000) & (income < 200000):
        return 'medium'
    elif income >= 200000:
        return 'high'

data['inc_bracket'] = data['annual_inc'].apply(income_bracket)
```

In [176]:
```
data['inc_bracket'] = data['inc_bracket'].astype('category')
```

In [177]:
```
fig, ax = plt.subplots(1, 3, figsize = (14, 6))

sns.violinplot(x='inc_bracket', y='loan_amnt', data=data, order = ['low', '
ax[0].set_title('Loan Volume by Income')

data.groupby(['issue_y', 'inc_bracket'])['int_rate'].mean().unstack().plot(
ax[1].set_title('Interest Rate by Income')

sns.boxplot(x='inc_bracket', y="emp_len", data=data, order = ['low', 'mediu
ax[2].set_title('Employment Length by Income')
```

Out[177]: Text(0.5, 1.0, 'Employment Length by Income')



Compared to low and medium income borrowers, high income borrowers take higher loan amounts.

In general, borrowers bear lower chances of defaulting and therefore get better rates than the ones with lower income.

In [178]:
```
data['term'].value_counts()
```

Out[178]:
```
 36 months    1415071
 60 months     597544
Name: term, dtype: int64
```

In [179]:
```
data['term_int'] = data['term'].str.extract('(\d+)').astype(int)
```

In [180]:
```
data['term_36'] = np.where(data['term_int'] == 36, 1, 0)
```

In [181]:
```
data['grade'] = data['grade'].astype('category')
grade = np.array(data['grade']).reshape(-1, 1)
enc = OneHotEncoder(sparse = False)
grade_binary = enc.fit_transform(grade)
```

In [182]:
```
grade_encode = grade_binary[:, :-1]
```

In [183]:
```
Frame( grade_encode, columns = ['A', 'B', 'C', 'D', 'E', 'F'], dtype = int)
```

In [184]:
```
columns = ['term_36', 'int_rate', 'emp_len', 'dti',
           'funded_amnt', 'total_pymnt', 'loan_target']

new_data = pd.concat([data[columns].reset_index(drop=True), grade_df.reset_
```

In [185]:
```
new_data.head(5)
```

Out[185]:

	term_36	int_rate	emp_len	dti	funded_amnt	total_pymnt	loan_target	A	B	C	D	E	F
0	1	10.65	10	27.65	5000.0	5863.155187	0	0	1	0	0	0	0
1	0	15.27	1	1.00	2500.0	1014.530000	1	0	0	1	0	0	0
2	1	15.96	10	8.72	2400.0	3005.666844	0	0	0	1	0	0	0
3	1	13.49	10	20.00	10000.0	12231.890000	0	0	0	1	0	0	0
4	0	12.69	1	17.94	3000.0	4066.908161	0	0	1	0	0	0	0

In [186]:
```
#dti, grade, inc_bracket, terms, interest rate
X = new_data.loc[:, new_data.columns != 'loan_target']
y = new_data['loan_target']
```

In [187]:
```
X.head(5)
```

Out[187]:

	term_36	int_rate	emp_len	dti	funded_amnt	total_pymnt	A	B	C	D	E	F
0	1	10.65	10	27.65	5000.0	5863.155187	0	1	0	0	0	0
1	0	15.27	1	1.00	2500.0	1014.530000	0	0	1	0	0	0
2	1	15.96	10	8.72	2400.0	3005.666844	0	0	1	0	0	0
3	1	13.49	10	20.00	10000.0	12231.890000	0	0	1	0	0	0
4	0	12.69	1	17.94	3000.0	4066.908161	0	1	0	0	0	0

In [188]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [189]:
```
#logistic regression
clf = LogisticRegression(random_state=1)
clf.fit(X_train, y_train)
print('Accuracy:{}'.format( clf.score(X_test, y_test)))

y_scores = clf.predict_proba(X_test)[:,1]
print('AUC:{}'.format( roc_auc_score(y_test, y_scores)))
```

```
Accuracy:0.8696273256435036
AUC:0.6482021994562321
```

Logistic regression acheives ~86% accuracy and ~64% roc-auc score. Using acccuracy is inappropriate since we have quite an imbalanced class here. Logistic regression is easy to understand and can give good reason why certain loans are classified as bad. ROC-AUC score is however low so we want to explore other techniques.

In [190]:
```
rf = RandomForestClassifier(random_state=1)
rf.fit(X_train, y_train)

y_scores = rf.predict_proba(X_test)[:,1]
print('AUC:{}'.format( roc_auc_score(y_test, y_scores)))
```

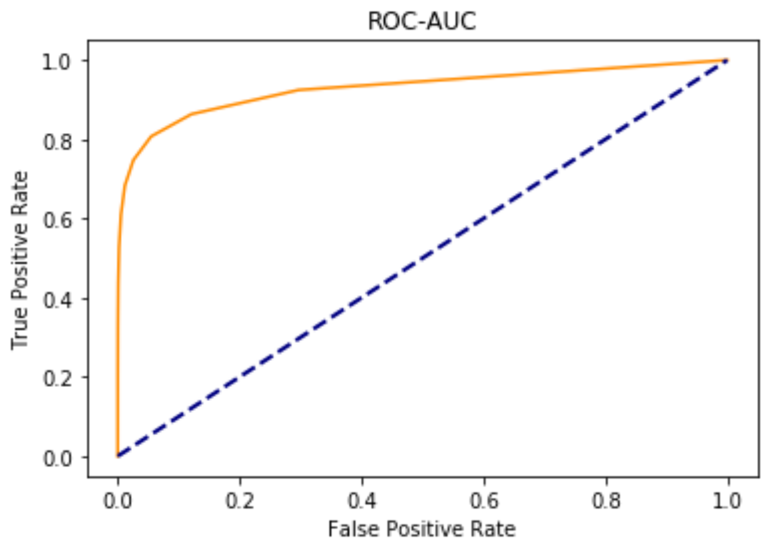
```
AUC:0.9286320673150786
```

Random Forest acheives ~92.8% roc-auc score. Random forest performs much better than logistic regression but we lost transparent as users do not know how it works.

Let's use grid search and cross validation to tune hyper-parameters and see if we can improve the model.

In [192]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
plt.plot(fpr, tpr, color = 'darkorange')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.title('ROC-AUC')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

Out[192]:
```
Text(0, 0.5, 'True Positive Rate')
```



In [ ]:
```
param_grid = { 'n_estimators': np.arange(0, 150, 50) ,
               'min_samples_leaf': [100, 300] }

gs_est = GridSearchCV( estimator = rf , param_grid = param_grid , cv=3, n_j
gs_est.fit(X_train, y_train)
print('Random Forest:{}'.format( gs_http://localhost:8888/notebooks/Lending
```

In [ ]:
```
gs_est.best_params_
```

Grid search used on number of estimators and minimal number of sample in the end node dp not seem to improve the performance of our original model. Therefore, we are going to stick with our default hyper

Final remarks:

The final model has a auc of 92.8% which is pretty good. However,there are lots of work we can do to improve the model. We could use calculation or imputation for other fields to fill in null values. We could also explore more variables, such as, recent inquiries, loan purpose, to be included in the model. In additional, we can tune some of the hyper-parameters to avoid overfitting.

In [ ]: