In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot  as plt
import seaborn as sns
import missingno as msno
import pandas_profiling as pfile
import datetime
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor
import warnings
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, log_loss
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns',None)
```

executed in 9ms, finished 19:50:16 2020-09-18

In [3]:

```
df=pd.read_csv(r'train.csv')
```

executed in 3.14s, finished 19:50:19 2020-09-18

In [4]:

```
1 df
```

executed in 91ms, finished 19:50:19 2020-09-18

Out[4]:

	id	loanAmnt	term	interestRate	installment	grade	subGrade	employment
0	0	35000.0	5	19.52	917.97	E	E2	3
1	1	18000.0	5	18.49	461.90	D	D2	2198
2	2	12000.0	5	16.99	298.17	D	D3	316
3	3	11000.0	3	7.26	340.96	A	A4	468
4	4	3000.0	3	12.99	101.07	C	C2	
...	...	...	...	...	...	...	...	
799995	799995	25000.0	3	14.49	860.41	C	C4	26
799996	799996	17000.0	3	7.90	531.94	A	A4	292
799997	799997	6000.0	3	13.33	203.12	C	C3	25
799998	799998	19200.0	3	6.92	592.14	A	A4	1
799999	799999	9000.0	3	11.06	294.91	B	B3	

800000 rows × 47 columns

In [5]:

```
1 numerical_fea = list(df.select_dtypes(exclude=['object']).columns)
2 category_fea = list(filter(lambda x: x not in numerical_fea,list(df.columns)))
```

executed in 151ms, finished 19:50:19 2020-09-18

In [6]:

```
1 category_fea
```

executed in 5ms, finished 19:50:19 2020-09-18

Out[6]:

['grade', 'subGrade', 'employmentLength', 'issueDate', 'earliesCreditLine']

In [7]:

```
1 df.info()
```

executed in 281ms, finished 19:50:19 2020-09-18

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800000 entries, 0 to 799999
Data columns (total 47 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   id                 800000 non-null  int64  
 1   loanAmnt           800000 non-null  float64
 2   term               800000 non-null  int64  
 3   interestRate       800000 non-null  float64
 4   installment        800000 non-null  float64
 5   grade              800000 non-null  object 
 6   subGrade           800000 non-null  object 
 7   employmentTitle    799999 non-null  float64
 8   employmentLength   753201 non-null  object 
 9   homeOwnership      800000 non-null  int64  
 10  annualIncome       800000 non-null  float64
 11  verificationStatus 800000 non-null  int64  
 12  issueDate          800000 non-null  object 
 13  isDefault          800000 non-null  int64  
 14  purpose            800000 non-null  int64  
 15  postCode           799999 non-null  float64
 16  regionCode         800000 non-null  int64  
 17  dti                799761 non-null  float64
 18  delinquency_2years 800000 non-null  float64
 19  ficoRangeLow       800000 non-null  float64
 20  ficoRangeHigh      800000 non-null  float64
 21  openAcc            800000 non-null  float64
 22  pubRec             800000 non-null  float64
 23  pubRecBankruptcies 799595 non-null  float64
 24  revolBal           800000 non-null  float64
 25  revolUtil          799469 non-null  float64
 26  totalAcc           800000 non-null  float64
 27  initialListStatus  800000 non-null  int64  
 28  applicationType    800000 non-null  int64  
 29  earliesCreditLine  800000 non-null  object 
 30  title              799999 non-null  float64
 31  policyCode         800000 non-null  float64
 32  n0                 759730 non-null  float64
 33  n1                 759730 non-null  float64
 34  n2                 759730 non-null  float64
 35  n2.1               759730 non-null  float64
 36  n4                 766761 non-null  float64
 37  n5                 759730 non-null  float64
 38  n6                 759730 non-null  float64
 39  n7                 759730 non-null  float64
 40  n8                 759729 non-null  float64
 41  n9                 759730 non-null  float64
 42  n10                766761 non-null  float64
 43  n11                730248 non-null  float64
 44  n12                759730 non-null  float64
 45  n13                759730 non-null  float64
 46  n14                759730 non-null  float64
dtypes: float64(33), int64(9), object(5)
memory usage: 286.9+ MB
```

In [8]:

```
1 df.isnull().sum()
```

executed in 208ms, finished 19:50:19 2020-09-18

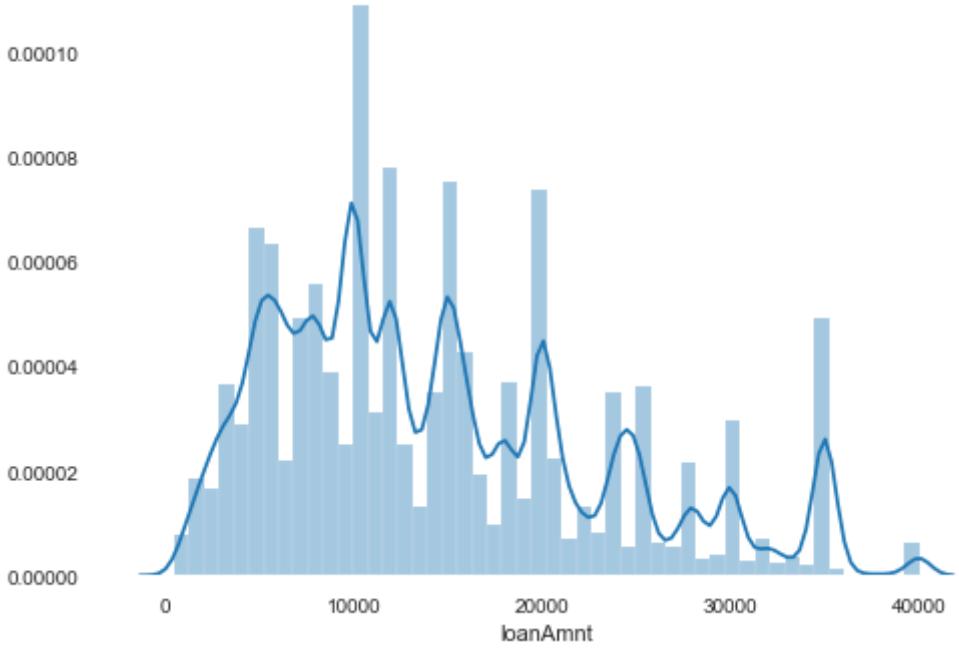
Out[8]:

```
id                     0
loanAmnt               0
term                   0
interestRate           0
installment            0
grade                  0
subGrade               0
employmentTitle        1
employmentLength   46799
homeOwnership          0
annualIncome           0
verificationStatus     0
issueDate              0
isDefault              0
purpose                0
postCode               1
regionCode             0
dti                  239
delinquency_2years     0
ficoRangeLow           0
ficoRangeHigh          0
openAcc                0
pubRec                 0
pubRecBankruptcies   405
revolBal               0
revolUtil            531
totalAcc               0
initialListStatus      0
applicationType        0
earliesCreditLine      0
title                  1
policyCode             0
n0                 40270
n1                 40270
n2                 40270
n2.1               40270
n4                 33239
n5                 40270
n6                 40270
n7                 40270
n8                 40271
n9                 40270
n10                33239
n11                69752
n12                40270
n13                40270
n14                40270
dtype: int64
```

In [9]:

```
1 #查看贷款金额数据分布
2 f,ax = plt.subplots()
3 sns.distplot(df['loanAmnt'])
4 plt.show()
```

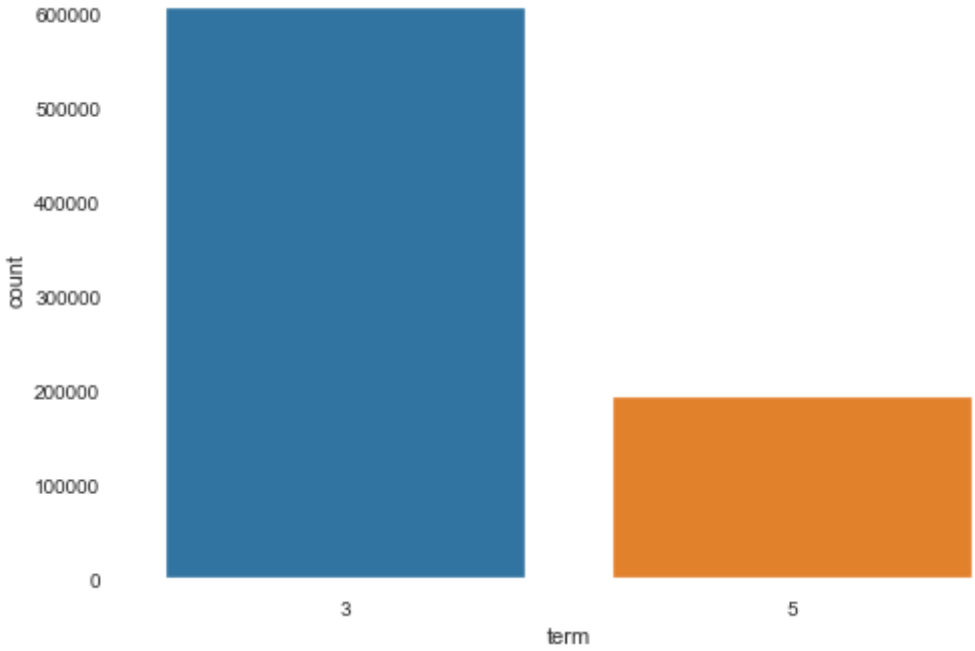
executed in 969ms, finished 19:50:20 2020-09-18



In [10]:

```
1 #查看贷款月份
2 f,ax = plt.subplots()
3 sns.countplot(df['term'])
4 plt.show()
```

executed in 168ms, finished 19:50:21 2020-09-18



In [11]:

```
1 df.term.value_counts()
```

executed in 14ms, finished 19:50:21 2020-09-18

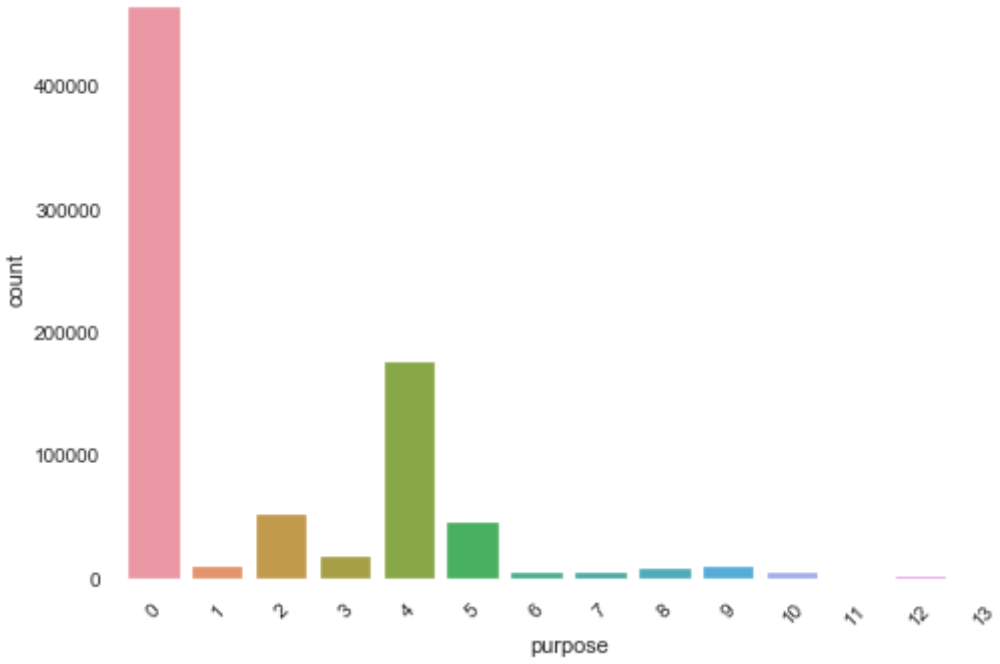
Out[11]:

```
3    606902
5    193098
Name: term, dtype: int64
```

In [12]:

```
#查看贷款目的 我猜0是个人周转还是消费贷?
f,ax = plt.subplots()
sns.countplot(df['purpose'])
plt.xticks(rotation = 45)
plt.show()
```

executed in 221ms, finished 19:50:21 2020-09-18

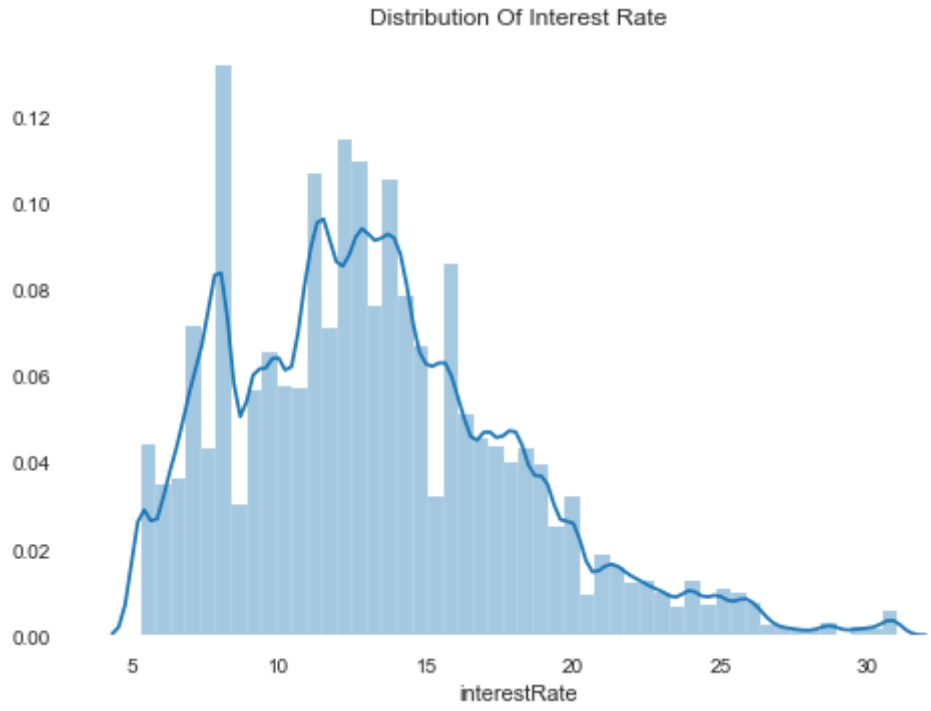


In [13]:

```
#贷款利率分布
f,ax = plt.subplots()
sns.distplot(df['interestRate'])
plt.title('Distribution Of Interest Rate')
plt.show()

print(df['interestRate'].describe())
print(df['interestRate'].median())
```

executed in 457ms, finished 19:50:21 2020-09-18

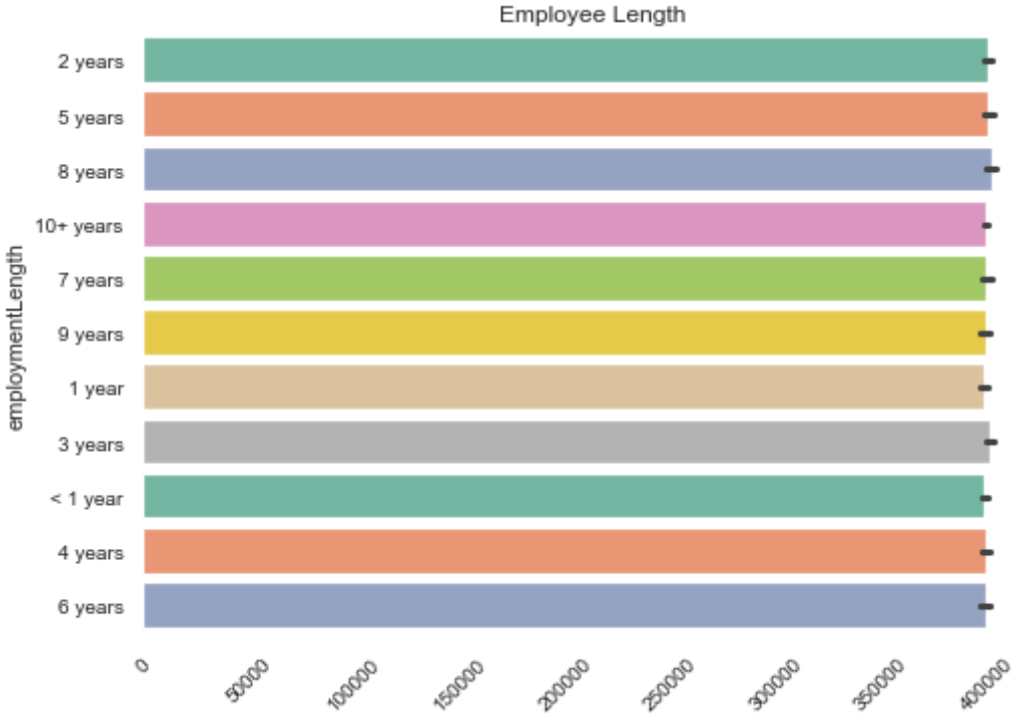



```
count    800000.000000
mean         13.238391
std           4.765757
min           5.310000
25%           9.750000
50%          12.740000
75%          15.990000
max          30.990000
Name: interestRate, dtype: float64
12.74
```

In [14]:

```
f,ax = plt.subplots()
sns.barplot(x = df.employmentLength.index,y = df.employmentLength,palette= 'Set2')
plt.xticks(rotation = 45)
plt.xlabel('')
plt.title('Employee Length')
plt.show()
```

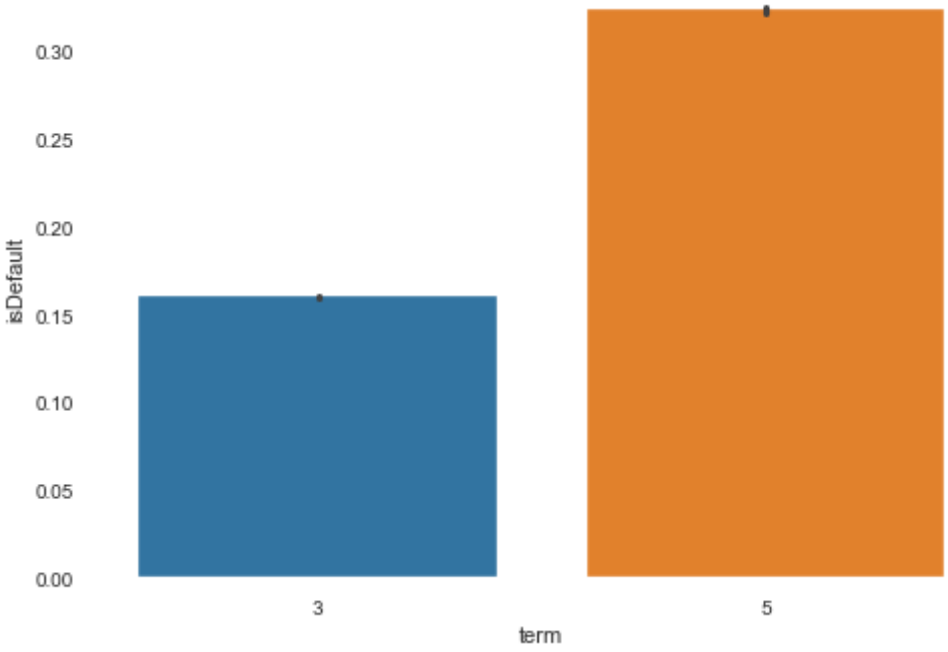
executed in 6.06s, finished 19:50:27 2020-09-18



In [15]:

```
#多变量分析
f,ax = plt.subplots()
sns.barplot(x = 'term',y = 'isDefault',data = df)
plt.show()
#借贷周期越长，违约的概率越高。
```

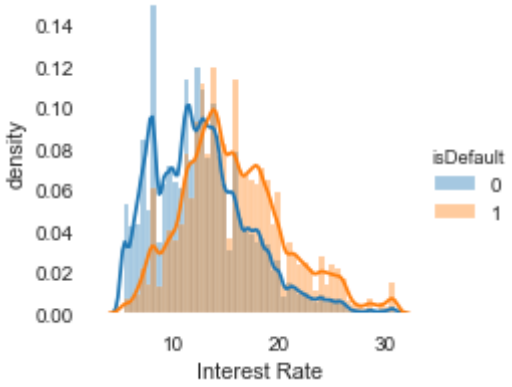
executed in 10.5s, finished 19:50:38 2020-09-18



In [41]:

```
#f,ax = plt.subplots()
g = sns.FacetGrid(data = df,hue = 'isDefault')
g.map(sns.distplot,'interestRate',norm_hist=True)
g.add_legend()
plt.ylabel('density')
plt.xlabel('Interest Rate')
plt.show()
#从不同贷款状态的利率密度分布图来看，当利率小于12%时，贷款状态为0的密度整体大于状态为1，当利率
```

executed in 5.22s, finished 19:55:39 2020-09-18

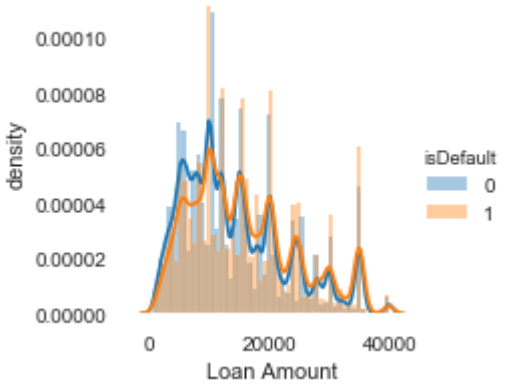


In [40]:

```
#f,ax = plt.subplots()
plt.figure(figsize=(10,5))
g = sns.FacetGrid(data = df,hue = 'isDefault')
g.map(sns.distplot,'loanAmnt',norm_hist=True)
g.add_legend()
plt.ylabel('density')
plt.xlabel('Loan Amount')
plt.show()
#不同贷款状态瞎的贷款金额密度没有明显的差别，可以看出贷款金额对于违约与否的影响并不大
```

executed in 4.92s, finished 19:55:14 2020-09-18

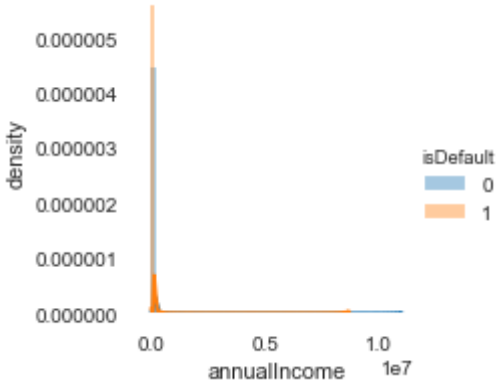
<Figure size 720x360 with 0 Axes>



In [42]:

```
#f,ax = plt.subplots()
g = sns.FacetGrid(data = df,hue = 'isDefault')
g.map(sns.distplot,'annualIncome',norm_hist=True)
g.add_legend()
plt.ylabel('density')
plt.xlabel('annualIncome')
plt.show()
```

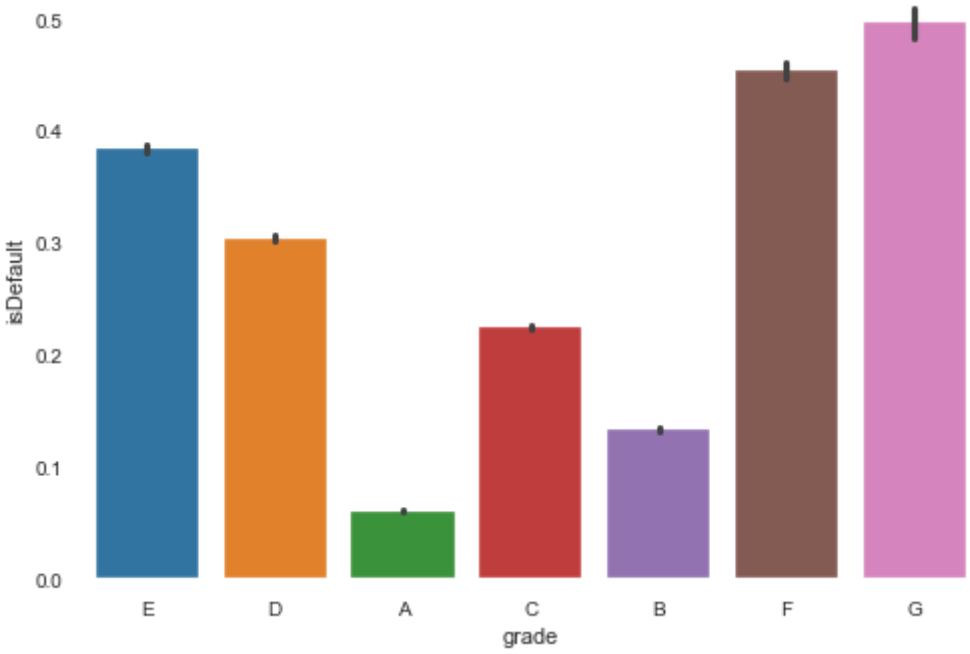
executed in 4.71s, finished 19:55:46 2020-09-18



In [19]:

```
f,ax = plt.subplots()
sns.barplot(x = 'grade',y = 'isDefault',data = df)
plt.show()
#图中，随着信用等级由A到G不断降低，贷款状态越加接近于1，违约的概率不断增加。由此可以看出，信用
```

executed in 8.39s, finished 19:50:53 2020-09-18



In [ ]:

```
1
```

In [20]:

```
1 #按照平均数填充数值型特征
2 df[numerical_fea] = df[numerical_fea].fillna(df[numerical_fea].median())
3 #按照众数填充类别型特征
4 df[category_fea] = df[category_fea].fillna(df[category_fea].mode())
```

executed in 1.67s, finished 19:50:55 2020-09-18

In [21]:

```
1 df.isnull().sum()
```

executed in 215ms, finished 19:50:55 2020-09-18

Out[21]:

```
id                     0
loanAmnt               0
term                   0
interestRate           0
installment            0
grade                  0
subGrade               0
employmentTitle        0
employmentLength   46799
homeOwnership          0
annualIncome           0
verificationStatus     0
issueDate              0
isDefault              0
purpose                0
postCode               0
regionCode             0
dti                    0
delinquency_2years     0
ficoRangeLow           0
ficoRangeHigh          0
openAcc                0
pubRec                 0
pubRecBankruptcies     0
revolBal               0
revolUtil              0
totalAcc               0
initialListStatus      0
applicationType        0
earliesCreditLine      0
title                  0
policyCode             0
n0                     0
n1                     0
n2                     0
n2.1                   0
n4                     0
n5                     0
n6                     0
n7                     0
n8                     0
n9                     0
n10                    0
n11                    0
n12                    0
n13                    0
n14                    0
dtype: int64
```

In [22]:

```
#转化成时间格式
for data in [df]:
    data['issueDate'] = pd.to_datetime(data['issueDate'],format='%Y-%m-%d')
    startdate = datetime.datetime.strptime('2007-06-01', '%Y-%m-%d')
    #构造时间特征
    data['issueDateDT'] = data['issueDate'].apply(lambda x: x-startdate).dt.days
```

executed in 14.8s, finished 19:51:10 2020-09-18

In [23]:

```
df['employmentLength'].value_counts(dropna=False).sort_index()
```

executed in 68ms, finished 19:51:10 2020-09-18

Out[23]:

```
1 year       52489
10+ years   262753
2 years      72358
3 years      64152
4 years      47985
5 years      50102
6 years      37254
7 years      35407
8 years      36192
9 years      30272
< 1 year     64237
NaN          46799
Name: employmentLength, dtype: int64
```

In [24]:

```
def employmentLength_to_int(s):
    if pd.isnull(s):
        return s
    else:
        return np.int8(s.split()[0])
for data in [df]:
    data['employmentLength'].replace(to_replace='10+ years', value='10 years', inplace=True)
    data['employmentLength'].replace('< 1 year', '0 years', inplace=True)
    data['employmentLength'] = data['employmentLength'].apply(employmentLength_to_int)
```

executed in 1.05s, finished 19:51:11 2020-09-18

In [25]:

```
1 df['employmentLength'].value_counts(dropna=False).sort_index()
```

executed in 19ms, finished 19:51:11 2020-09-18

Out[25]:

```
0.0     64237
1.0     52489
2.0     72358
3.0     64152
4.0     47985
5.0     50102
6.0     37254
7.0     35407
8.0     36192
9.0     30272
10.0   262753
NaN     46799
Name: employmentLength, dtype: int64
```

In [26]:

```
1 df['earliesCreditLine'].sample(5)
```

executed in 154ms, finished 19:51:11 2020-09-18

Out[26]:

```
624121    Nov-2007
793579    May-2000
4603      Jun-2000
726555    Mar-2003
439908    Oct-1993
Name: earliesCreditLine, dtype: object
```

In [27]:

```
1 for data in [df]:
2     data['earliesCreditLine'] = data['earliesCreditLine'].apply(lambda s: int(s[-4:]))
```

executed in 378ms, finished 19:51:12 2020-09-18

In [28]:

```
# 部分类别特征
cate_features = ['grade', 'subGrade', 'employmentTitle', 'homeOwnership', 'verificationStatus', 'purp
                 'applicationType', 'initialListStatus', 'title', 'policyCode']
for f in cate_features:
    print(f, '类型数：', df[f].nunique())
```

executed in 126ms, finished 19:51:12 2020-09-18

```
grade 类型数： 7
subGrade 类型数： 35
employmentTitle 类型数： 248683
homeOwnership 类型数： 6
verificationStatus 类型数： 3
purpose 类型数： 14
postCode 类型数： 932
regionCode 类型数： 51
applicationType 类型数： 2
initialListStatus 类型数： 2
title 类型数： 39644
policyCode 类型数： 1
```

In [29]:

```
for data in [df]:
    data['grade'] = data['grade'].map({'A':1,'B':2,'C':3,'D':4,'E':5,'F':6,'G':7})
```

executed in 48ms, finished 19:51:12 2020-09-18

In [30]:

```
# 类型数在2之上，又不是高维稀疏的,且纯分类特征
for data in [df]:
    data = pd.get_dummies(data, columns=['subGrade', 'homeOwnership', 'verificationStatus', 'purpo
```

executed in 877ms, finished 19:51:13 2020-09-18

In [31]:

```
def find_outliers_by_3segama(data,fea):
    data_std = np.std(data[fea])
    data_mean = np.mean(data[fea])
    outliers_cut_off = data_std * 3
    lower_rule = data_mean - outliers_cut_off
    upper_rule = data_mean + outliers_cut_off
    data[fea+'_outliers'] = data[fea].apply(lambda x:str('异常值') if x > upper_rule or x < lower_rule el
    return data
```

executed in 5ms, finished 19:51:13 2020-09-18

In [32]:

```
#df = df.copy()
#for fea in numerical_fea:
    #df = find_outliers_by_3segama(df,fea)
   # print(df[fea+'_outliers'].value_counts())
    #print('*'*10) 异常值的判断依据不信服

```

executed in 3ms, finished 19:51:13 2020-09-18

In [33]:

```
#删除异常值
#for fea in numerical_fea:
  #  data_train = data_train[data_train[fea+'_outliers']=='正常值']
    #data_train = data_train.reset_index(drop=True)
```

executed in 6ms, finished 19:51:13 2020-09-18

In [34]:

```
# 通过除法映射到间隔均匀的分箱中，每个分箱的取值范围都是loanAmnt/1000
data['loanAmnt_bin1'] = np.floor_divide(data['loanAmnt'], 1000)
## 通过对数函数映射到指数宽度分箱
data['loanAmnt_bin2'] = np.floor(np.log10(data['loanAmnt']))
data['loanAmnt_bin3'] = pd.qcut(data['loanAmnt'], 10, labels=False)
```

executed in 109ms, finished 19:51:13 2020-09-18

In [35]:

```
for col in ['grade', 'subGrade']:
    temp_dict = data_train.groupby([col])['isDefault'].agg(['mean']).reset_index().rename(columns={'m
    temp_dict.index = temp_dict[col].values
    temp_dict = temp_dict[col + '_target_mean'].to_dict()

    data_train[col + '_target_mean'] = data_train[col].map(temp_dict)
    data_test_a[col + '_target_mean'] = data_test_a[col].map(temp_dict)
```

executed in 359ms, finished 19:51:13 2020-09-18

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-35-e8b35c347f59> in <module>
      1 for col in ['grade', 'subGrade']:
----> 2     temp_dict = data_train.groupby([col])['isDefault'].agg(['mean']).reset_index(
).rename(columns={'mean': col + '_target_mean'})
      3     temp_dict.index = temp_dict[col].values
      4     temp_dict = temp_dict[col + '_target_mean'].to_dict()
      5

NameError: name 'data_train' is not defined
```

In [ ]:

```
# 其他衍生变量 mean 和 std
for df in [data_train, data_test_a]:
    for item in ['n0','n1','n2','n2.1','n4','n5','n6','n7','n8','n9','n10','n11','n12','n13','n14']:
        df['grade_to_mean_' + item] = df['grade'] / df.groupby([item])['grade'].transform('mean')
        df['grade_to_std_' + item] = df['grade'] / df.groupby([item])['grade'].transform('std')
```

executed in 1m 40.1s, finished 19:51:13 2020-09-18

In [ ]:

```
#label-encode:subGrade,postCode,title
# 高维类别特征需要进行转换
for col in tqdm(['employmentTitle', 'postCode', 'title','subGrade']):
    le = LabelEncoder()
    le.fit(list(data_train[col].astype(str).values) + list(data_test_a[col].astype(str).values))
    data_train[col] = le.transform(list(data_train[col].astype(str).values))
    data_test_a[col] = le.transform(list(data_test_a[col].astype(str).values))
print('Label Encoding 完成')
```

executed in 1m 40.1s, finished 19:51:13 2020-09-18

In [ ]:

```
# 删除不需要的数据
for data in [data_train, data_test_a]:
    data.drop(['issueDate','id'], axis=1,inplace=True)
```

executed in 1m 40.1s, finished 19:51:13 2020-09-18

In [ ]:

```
"纵向用缺失值上面的值替换缺失值"
data_train = data_train.fillna(axis=0,method='ffill')
```

executed in 1m 40.1s, finished 19:51:13 2020-09-18

In [ ]:

```
features = [f for f in data_train.columns if f not in ['id','issueDate','isDefault'] and '_outliers' not in f
x_train = data_train[features]
x_test = data_test_a[features]
y_train = data_train['isDefault']
```

executed in 1m 40.1s, finished 19:51:13 2020-09-18

In [ ]:

```
def cv_model(clf, train_x, train_y, test_x, clf_name):
    folds = 5
    seed = 2020
    kf = KFold(n_splits=folds, shuffle=True, random_state=seed)

    train = np.zeros(train_x.shape[0])
    test = np.zeros(test_x.shape[0])

    cv_scores = []

    for i, (train_index, valid_index) in enumerate(kf.split(train_x, train_y)):
        print('************************************ {} ************************************'.format(str(i+1)))
        trn_x, trn_y, val_x, val_y = train_x.iloc[train_index], train_y[train_index], train_x.iloc[valid_index], t

        if clf_name == "lgb":
            train_matrix = clf.Dataset(trn_x, label=trn_y)
            valid_matrix = clf.Dataset(val_x, label=val_y)

            params = {
                'boosting_type': 'gbdt',
                'objective': 'binary',
                'metric': 'auc',
                'min_child_weight': 5,
                'num_leaves': 2 ** 5,
                'lambda_l2': 10,
                'feature_fraction': 0.8,
                'bagging_fraction': 0.8,
                'bagging_freq': 4,
                'learning_rate': 0.1,
                'seed': 2020,
                'nthread': 28,
                'n_jobs':24,
                'silent': True,
                'verbose': -1,
            }

            model = clf.train(params, train_matrix, 50000, valid_sets=[train_matrix, valid_matrix], verbose
            val_pred = model.predict(val_x, num_iteration=model.best_iteration)
            test_pred = model.predict(test_x, num_iteration=model.best_iteration)

            # print(list(sorted(zip(features, model.feature_importance("gain")), key=lambda x: x[1], rever

        if clf_name == "xgb":
            train_matrix = clf.DMatrix(trn_x , label=trn_y)
            valid_matrix = clf.DMatrix(val_x , label=val_y)

            params = {'booster': 'gbtree',
                      'objective': 'binary:logistic',
                      'eval_metric': 'auc',
                      'gamma': 0.5,
                      'min_child_weight': 1.5,
                      'max_depth': 5,
                      'lambda':5,
                      'subsample': 0.7,
                      'colsample_bytree': 0.7,
                      'colsample_bylevel': 0.7,
                      'eta': 0.01,
                      'tree_method': 'exact',
                      'seed': 2020,
```

```
                'nthread': -1,
                "silent": True,
                }

        watchlist = [(train_matrix, 'train'),(valid_matrix, 'eval')]

        model = clf.train(params, train_matrix, num_boost_round=5000, evals=watchlist, verbose_ev
        val_pred  = model.predict(valid_matrix, ntree_limit=model.best_ntree_limit)
        test_pred = model.predict(test_x , ntree_limit=model.best_ntree_limit)

    if clf_name == "cat":
        params = {'learning_rate': 0.05, 'depth': 5, 'l2_leaf_reg': 10, 'bootstrap_type': 'Bernoulli',
                'od_type': 'Iter', 'od_wait': 50, 'random_seed': 11, 'allow_writing_files': False}

        model = clf(iterations=20000, **params)
        model.fit(trn_x, trn_y, eval_set=(val_x, val_y),
                cat_features=[], use_best_model=True, verbose=500)

        val_pred  = model.predict(val_x)
        test_pred = model.predict(test_x)

    train[valid_index] = val_pred
    test = test_pred / kf.n_splits
    cv_scores.append(roc_auc_score(val_y, val_pred))

    print(cv_scores)

print("%s_scotrainre_list:" % clf_name, cv_scores)
print("%s_score_mean:" % clf_name, np.mean(cv_scores))
print("%s_score_std:" % clf_name, np.std(cv_scores))
return train, test
```

executed in 1m 40.1s, finished 19:51:13 2020-09-18

In [ ]:

```
def lgb_model(x_train, y_train, x_test):
    lgb_train, lgb_test = cv_model(lgb, x_train, y_train, x_test, "lgb")
    return lgb_train, lgb_test

def xgb_model(x_train, y_train, x_test):
    xgb_train, xgb_test = cv_model(xgb, x_train, y_train, x_test, "xgb")
    return xgb_train, xgb_test

def cat_model(x_train, y_train, x_test):
    cat_train, cat_test = cv_model(CatBoostRegressor, x_train, y_train, x_test, "cat")
```

executed in 1m 40.1s, finished 19:51:13 2020-09-18

In [ ]:

```
xgb_train, xgb_test = xgb_model(x_train, y_train, x_test)
```

executed in 1m 40.1s, finished 19:51:13 2020-09-18

In [ ]:

```

```

executed in 51m 51s, finished 22:47:34 2020-09-14

In [ ]:

```
1
```