Overview

In the lending industry, investors provide loans to borrowers in exchange for the promise of repayment with interest. If the borrower repays the loan, then the lender would make profit from the interest. However, if the borrower fails to repay the loan, then the lender loses money. Therefore, lenders face the problem of predicting the risk of a borrower being unable to repay a loan.

In [1]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

Data Loading

In [2]:

```
df=pd.read_csv('loan_data.csv')
df
```

Out[2]:

	credit_policy	purpose	int_rate	installment	log_annual_inc	dti	fico	day:
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	
...	...	...	...	...	...	...	...	
9573	0	all_other	0.1461	344.76	12.180755	10.39	672	
9574	0	all_other	0.1253	257.70	11.141862	0.21	722	
9575	0	debt_consolidation	0.1071	97.81	10.596635	13.09	687	
9576	0	home_improvement	0.1600	351.58	10.819778	19.18	692	
9577	0	debt_consolidation	0.1392	853.43	11.264464	16.28	732	

9578 rows × 14 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit_policy      9578 non-null   int64  
 1   purpose            9578 non-null   object 
 2   int_rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log_annual_inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64  
 7   days_with_cr_line  9578 non-null   float64
 8   revol_bal          9578 non-null   int64  
 9   revol_util         9578 non-null   float64
 10  inq_last_6mths     9578 non-null   int64  
 11  delinq_2yrs        9578 non-null   int64  
 12  pub_rec            9578 non-null   int64  
 13  not_fully_paid     9578 non-null   int64  
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

df.describe()

DATA CLEANNING AND EDA PROCESS

Using pandas Functions like .isnull().sum() , .value_counts() Find out missing or Null values. and replace them with mean, mode and median

In [4]:

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

Out[4]:

```
credit_policy        0
purpose              0
int_rate             0
installment          0
log_annual_inc       0
dti                  0
fico                 0
days_with_cr_line    0
revol_bal            0
revol_util           0
inq_last_6mths       0
delinq_2yrs          0
pub_rec              0
not_fully_paid       0
dtype: int64
```

In [5]:

```
df['credit_policy'].value_counts()
```

Out[5]:

```
1    7710
0    1868
Name: credit_policy, dtype: int64
```

In [6]:

```
df['purpose'].value_counts()
```

Out[6]:

```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

In [7]:

```
df['int_rate'].value_counts()
```

Out[7]:

```
0.1253    354
0.0894    299
0.1183    243
0.1218    215
0.0963    210
         ...
0.2016      1
0.1683      1
0.1778      1
0.1756      1
0.1867      1
Name: int_rate, Length: 249, dtype: int64
```

In [8]:

```
df['installment'].value_counts()
```

Out[8]:

```
317.72    41
316.11    34
319.47    29
381.26    27
662.68    27
          ..
97.53      1
76.26      1
150.84     1
158.99     1
853.43     1
Name: installment, Length: 4788, dtype: int64
```

In [9]:

```
df['log_annual_inc'].value_counts()
```

Out[9]:

```
11.002100    308
10.819778    248
10.308953    224
10.596635    224
10.714418    221
            ... 
11.217534      1
12.078239      1
10.068451      1
9.621788       1
10.110472      1
Name: log_annual_inc, Length: 1987, dtype: int64
```

In [10]:

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

Out[10]:

```
0
```

In [11]:

```
df.purpose.value_counts()
```

Out[11]:

```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

Our DataFrame contain Zero Null values.

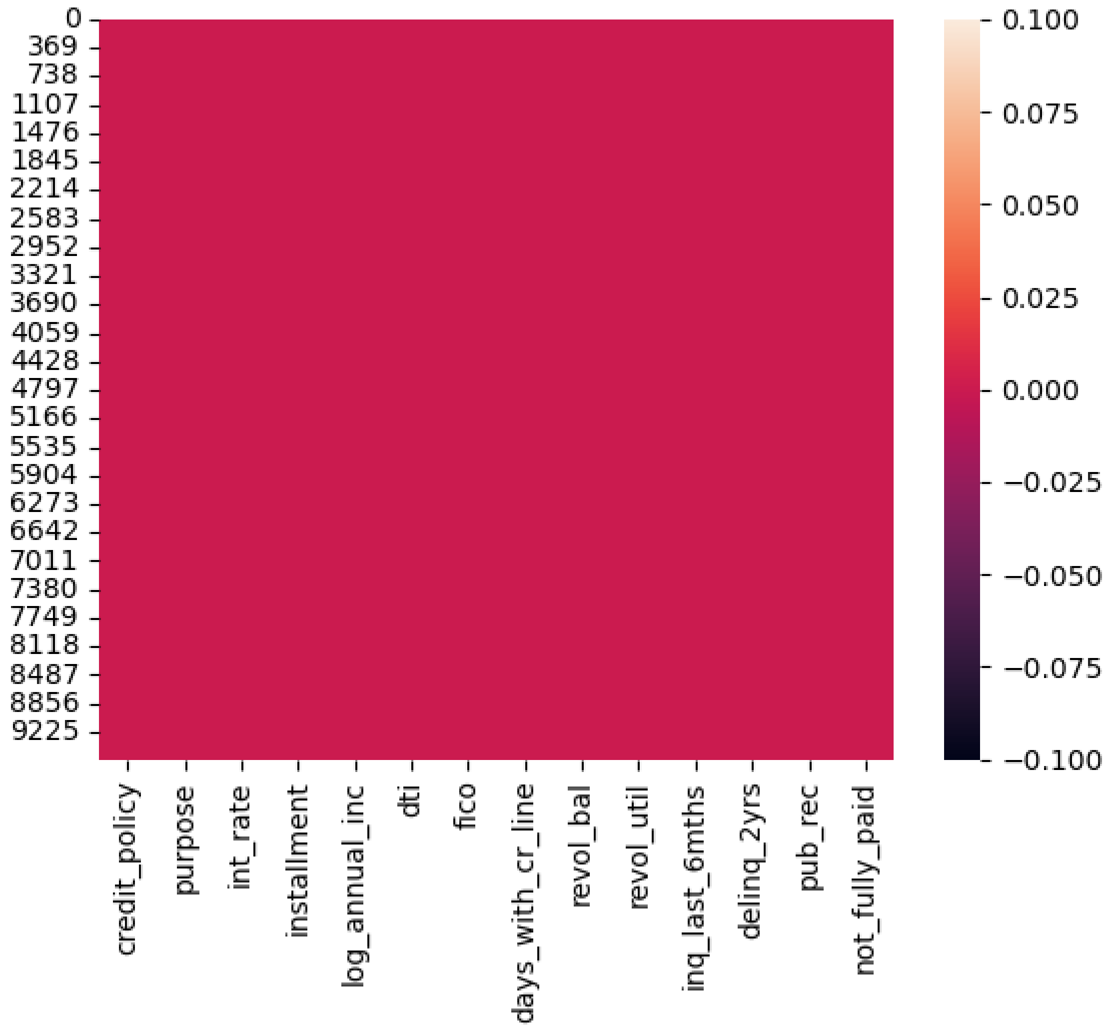
Now lets solve the problem with Purpose Attribute.

In [12]:

```
sns.heatmap(df.isnull())
```

Out[12]:

```
<AxesSubplot:>
```



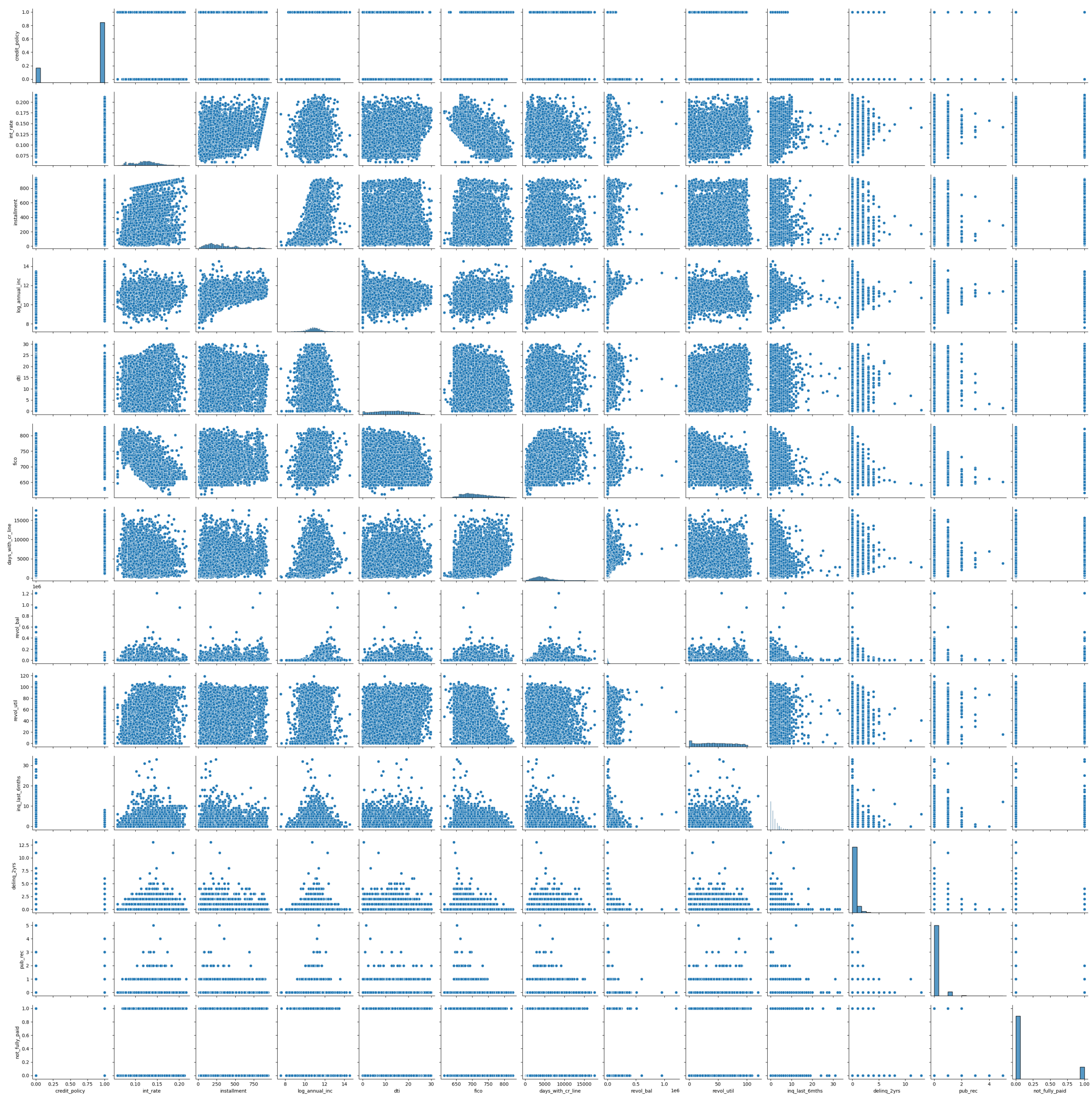
Data Visualization

In [13]:

```
sns.pairplot(df)
```

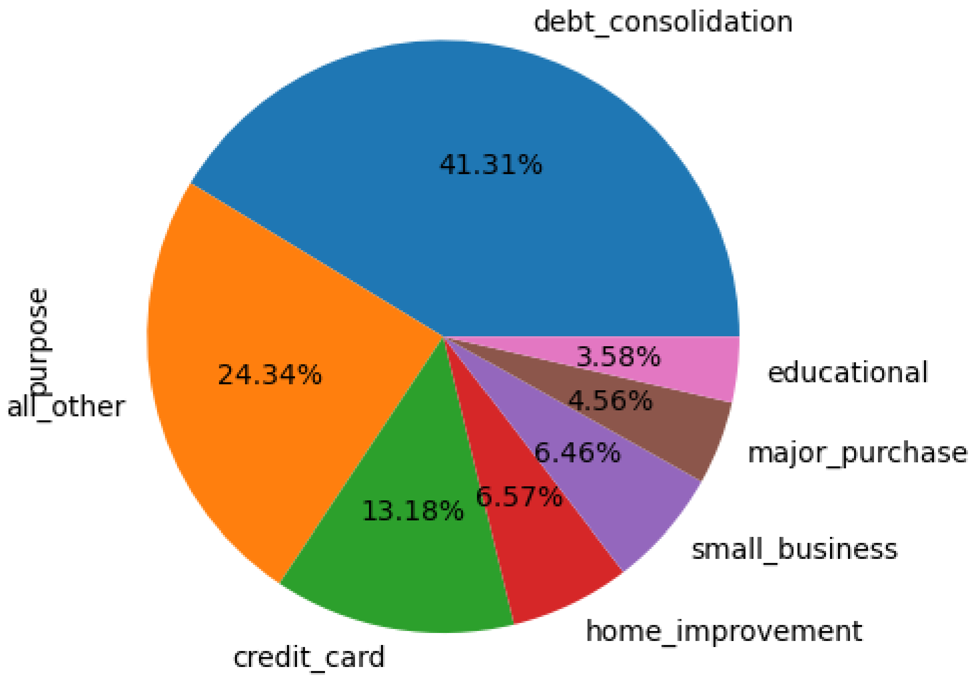
Out[13]:

<seaborn.axisgrid.PairGrid at 0x1ef29b13a60>



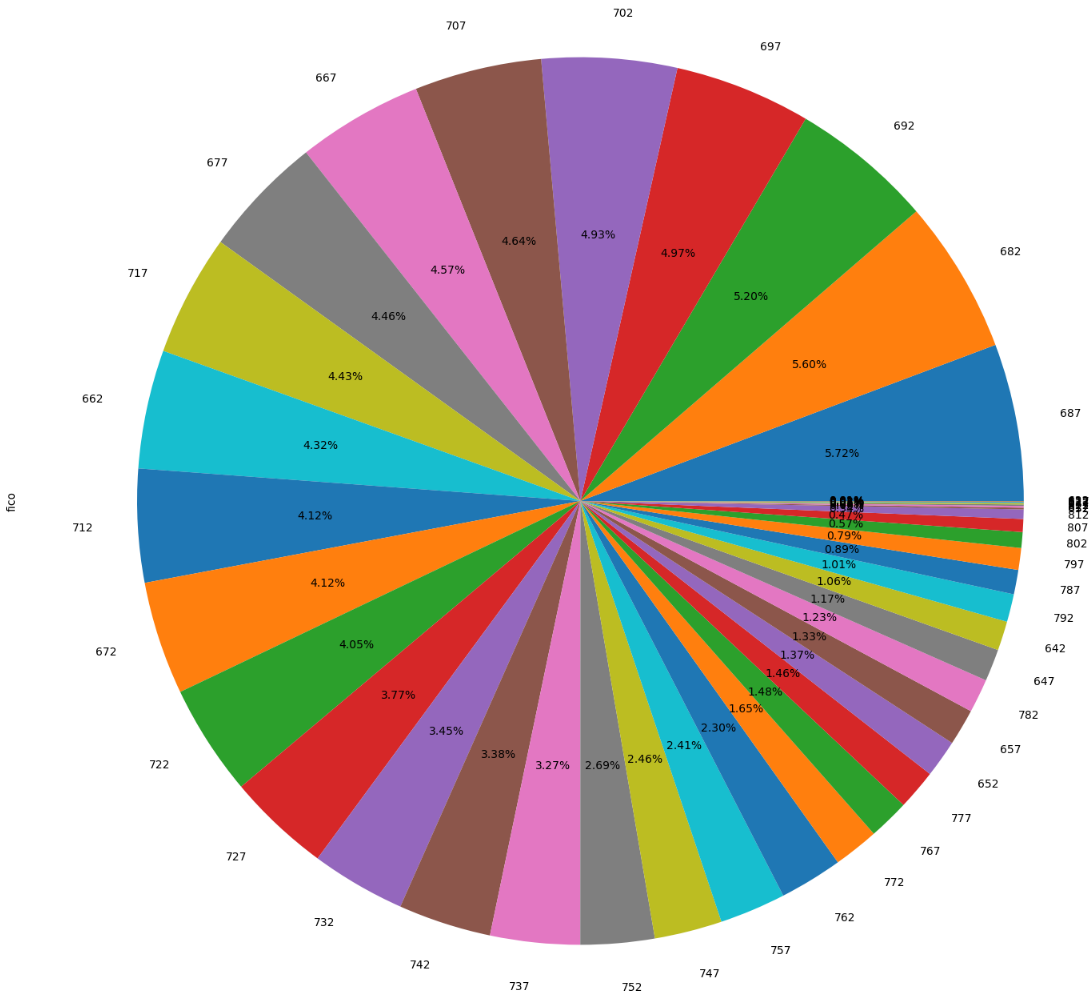
In [14]:

```
df.purpose.value_counts().plot(kind='pie',autopct='%1.2f%%')
plt.show()
```



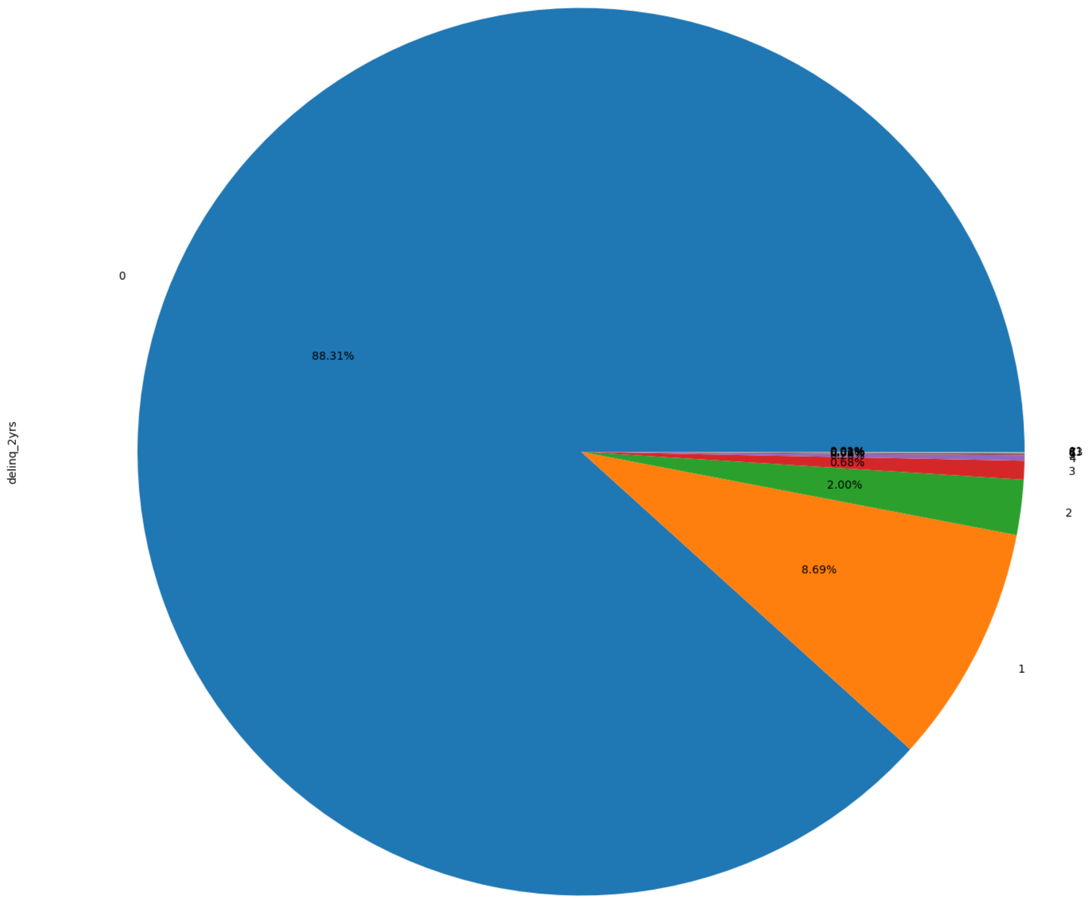
In [15]:

```
plt.figure(figsize=(18,18))
df.fico.value_counts().plot(kind='pie',autopct='%1.2f%%')
plt.show()
```



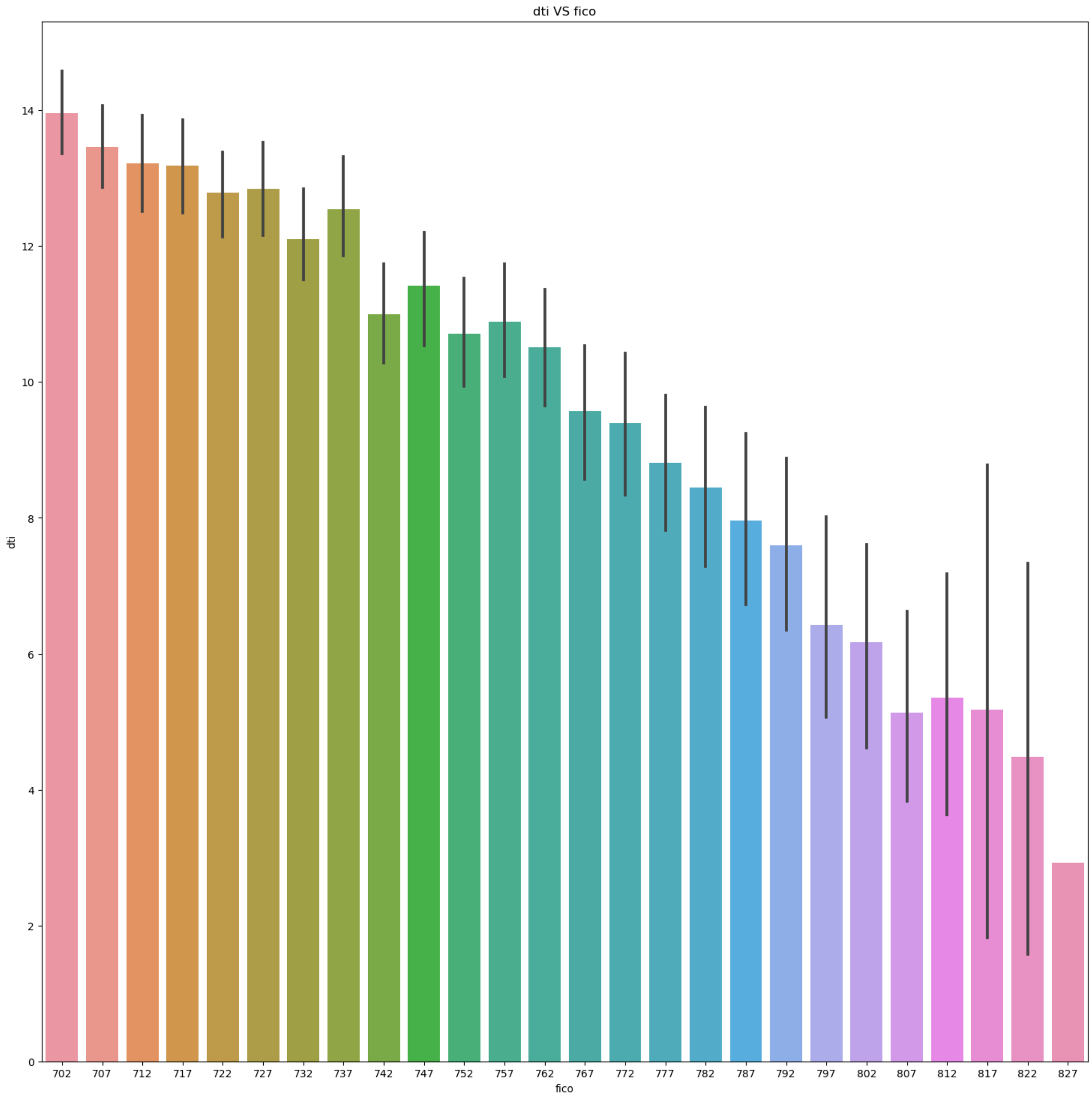
In [16]:

```
plt.figure(figsize=(18,18))
df.delinq_2yrs.value_counts().plot(kind='pie', autopct='%1.2f%%')
plt.show()
```



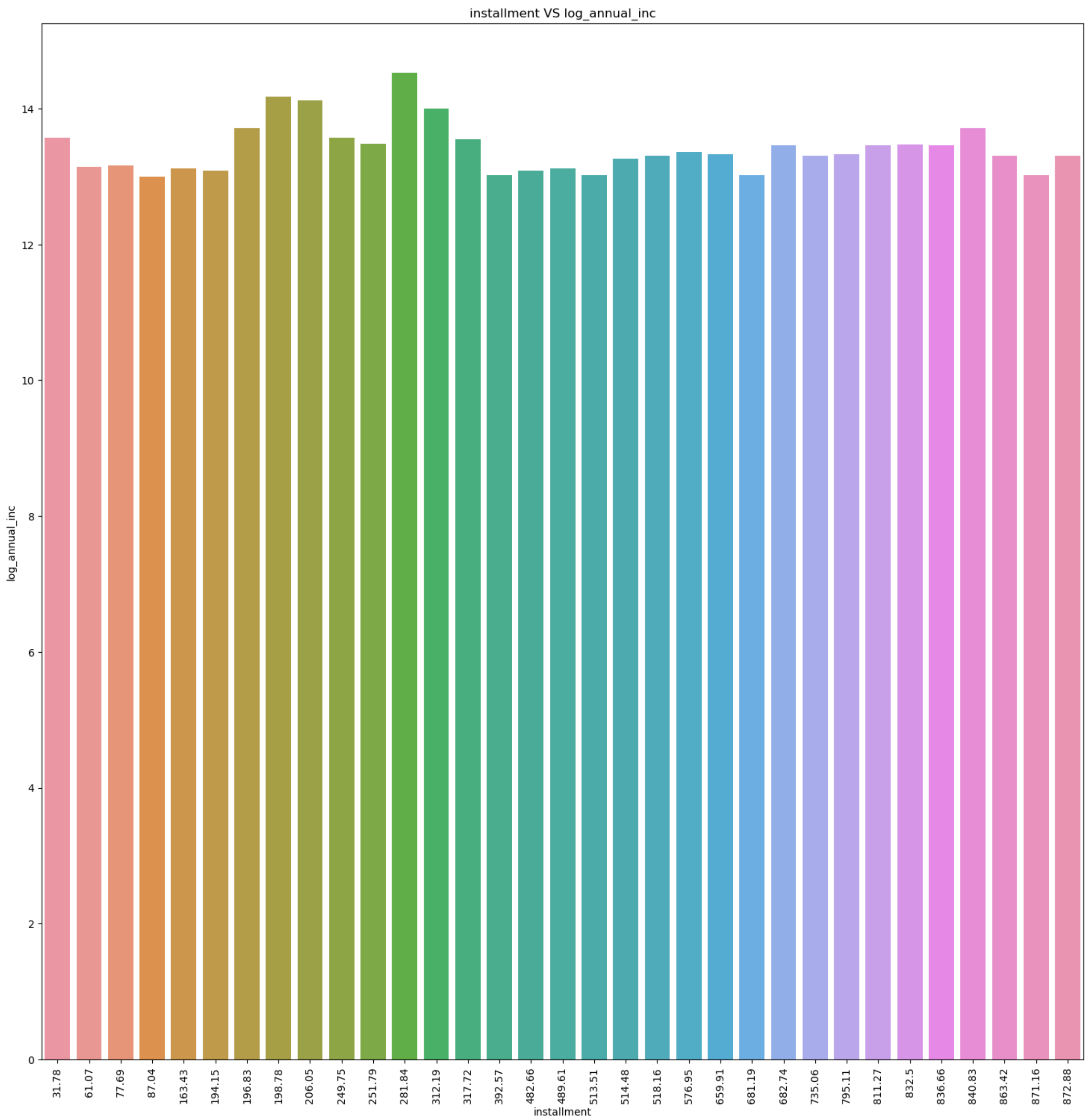
In [17]:

```
plt.figure(figsize=(18,18))
ss=df.loc[(df['fico']>700)]
sns.barplot(x ="fico", y= "dti", data = ss)
plt.title("dti VS fico")
plt.show()
```



In [18]:

```
plt.figure(figsize=(18,18))
ss=df.loc[(df['log_annual_inc']>13)]
sns.barplot(x ="installment", y= "log_annual_inc", data = ss)
plt.title("installment VS log_annual_inc")
plt.xticks(rotation=90)
plt.show()
```

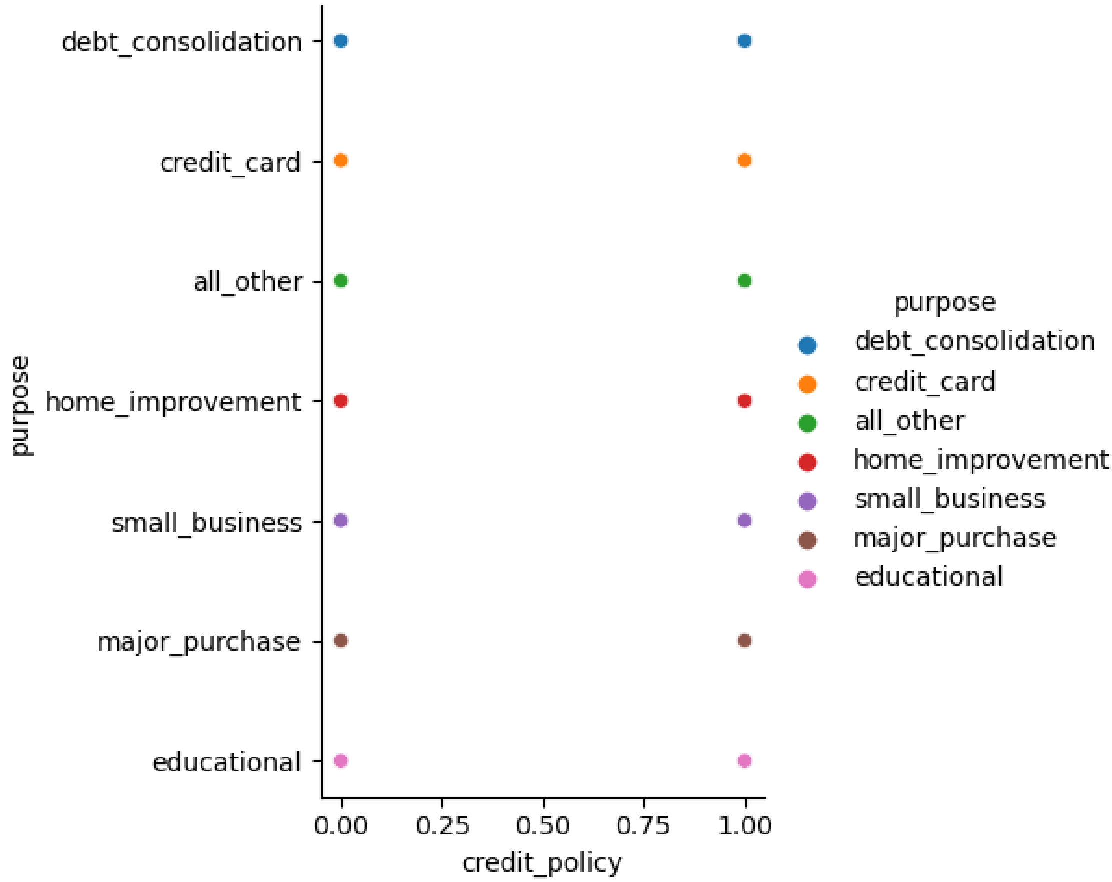


In [76]:

```
sns.relplot(x='credit_policy',y="purpose",color='red', hue='purpose',data=df)
#plt.xticks(rotation=90)
```

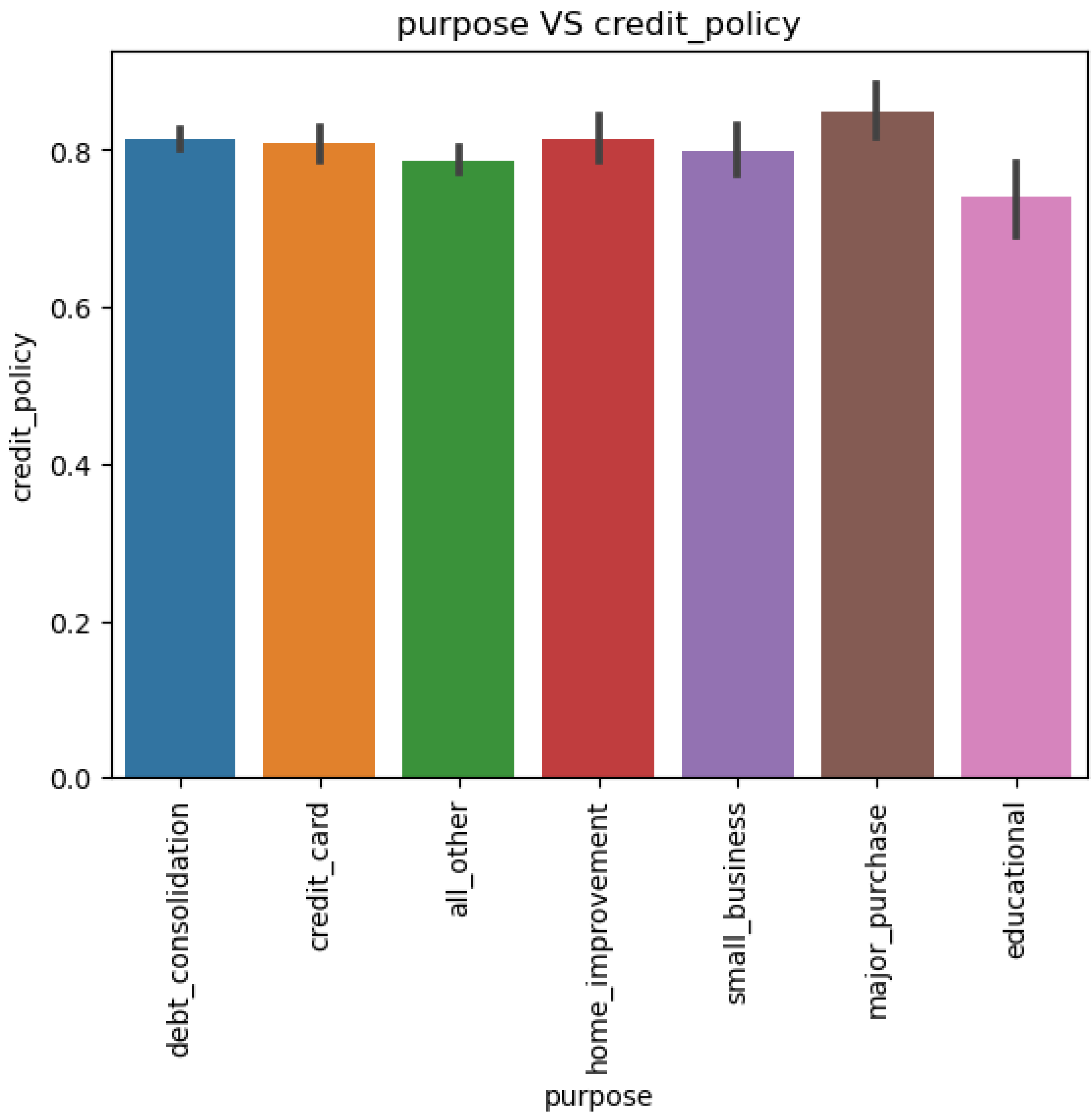
Out[76]:

```
<seaborn.axisgrid.FacetGrid at 0x1ef3bf0c4c0>
```



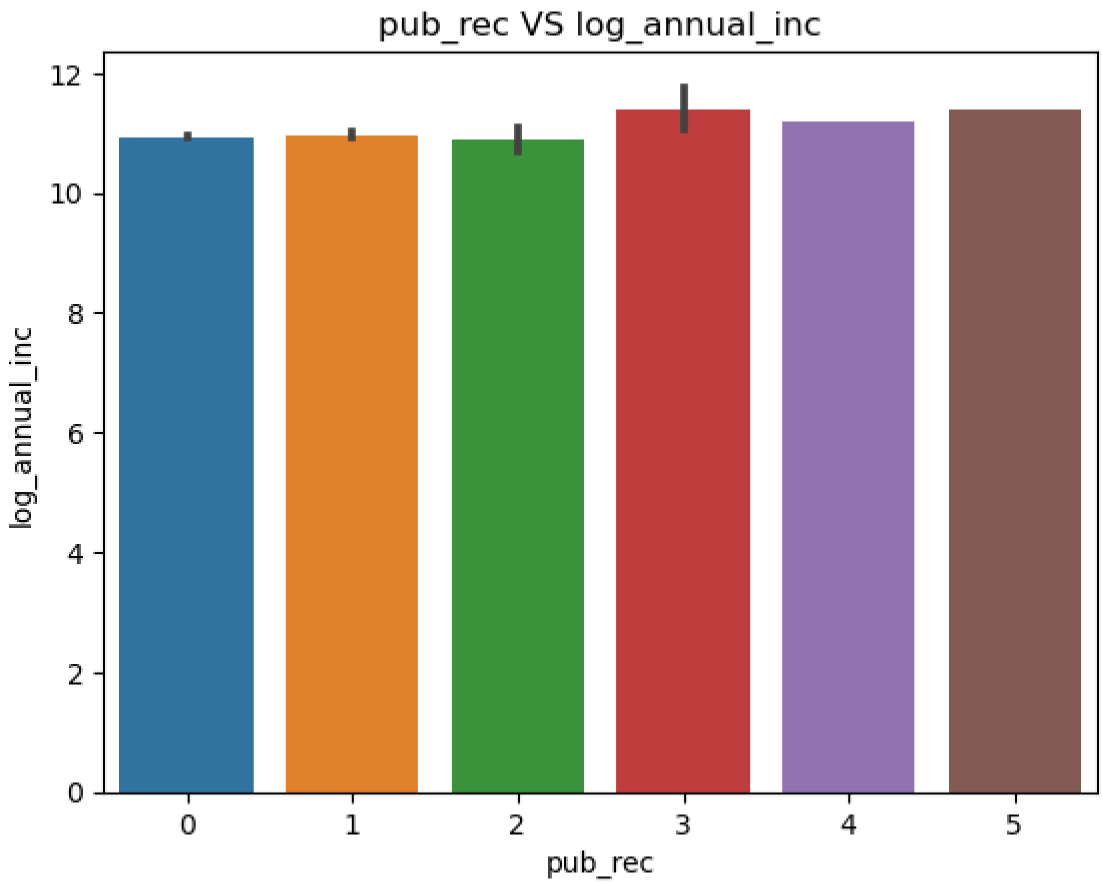
In [20]:

```
sns.barplot(x ="purpose", y= "credit_policy", data = df)
plt.title("purpose VS credit_policy")
plt.xticks(rotation = 90)
plt.show()
```



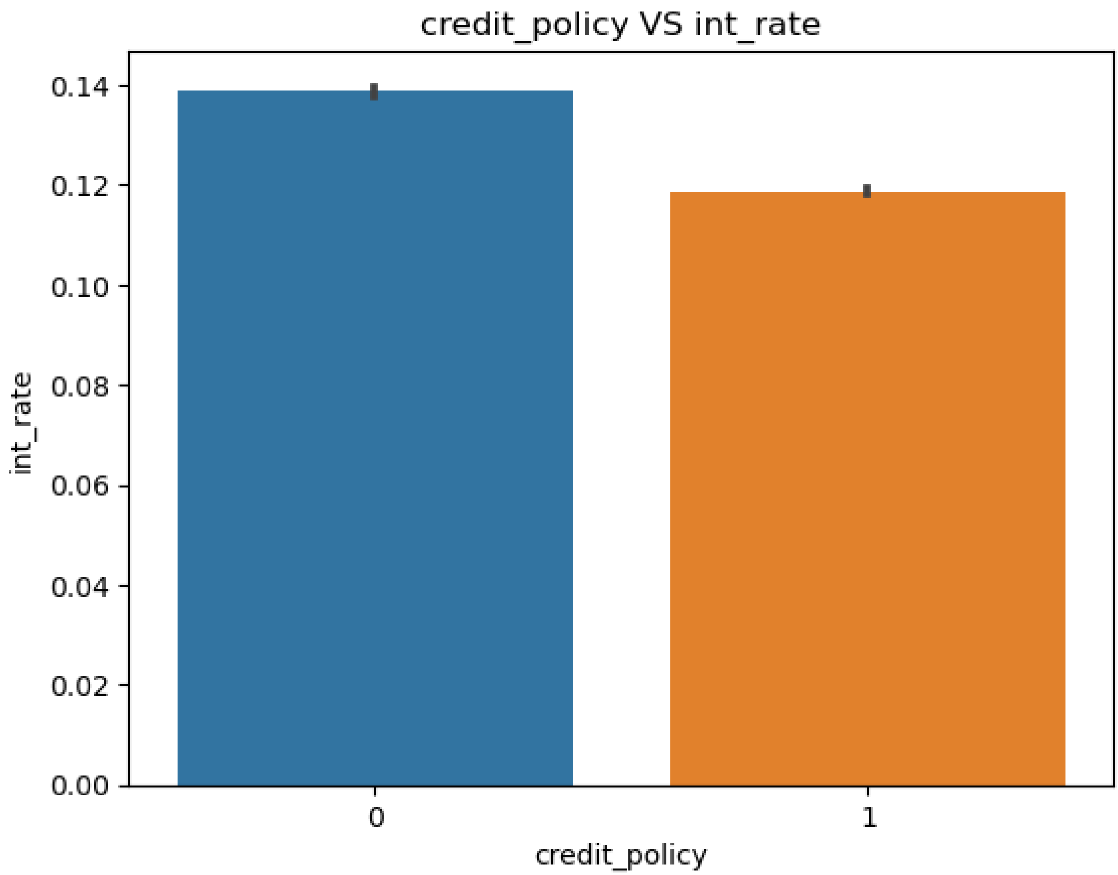
In [21]:

```
sns.barplot(x ="pub_rec", y= "log_annual_inc", data = df)
plt.title("pub_rec VS log_annual_inc")
plt.show()
```



In [22]:

```
sns.barplot(x ="credit_policy", y= "int_rate", data = df)
plt.title("credit_policy VS int_rate")
plt.show()
```



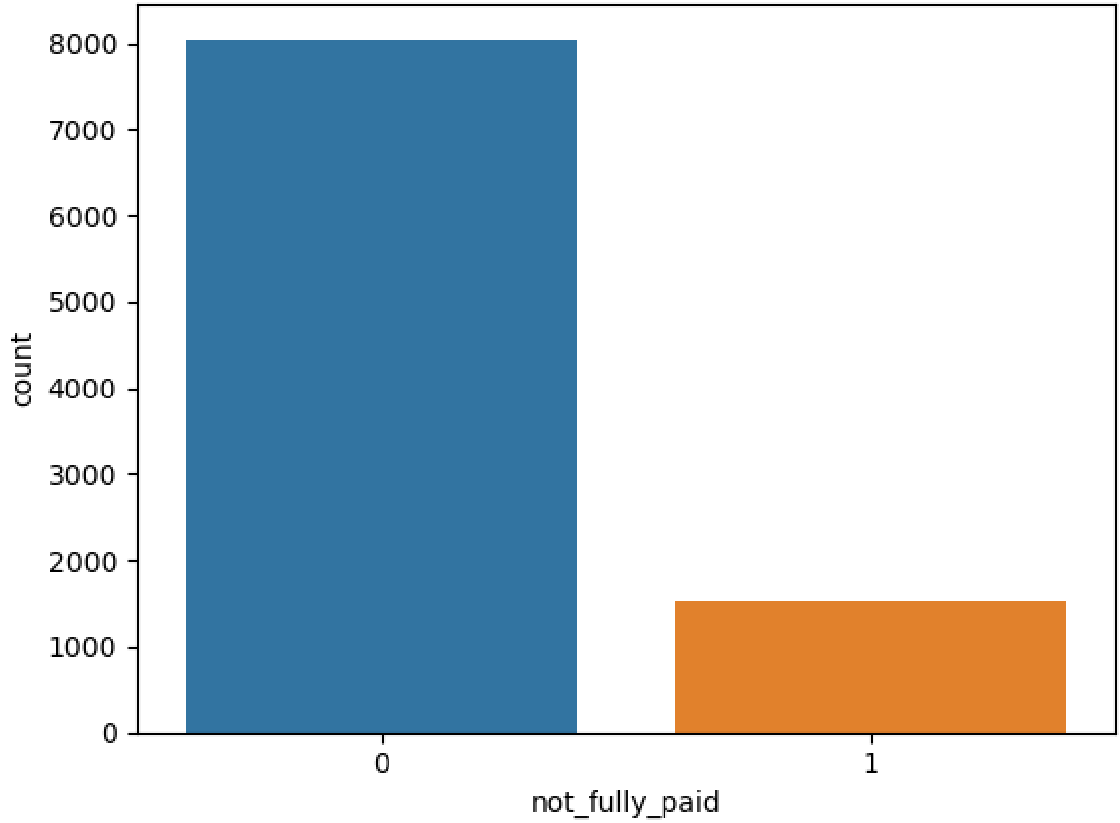
In [ ]:

```

```

In [23]:

```
sns.countplot(data=df,x="not_fully_paid")
plt.show()
```

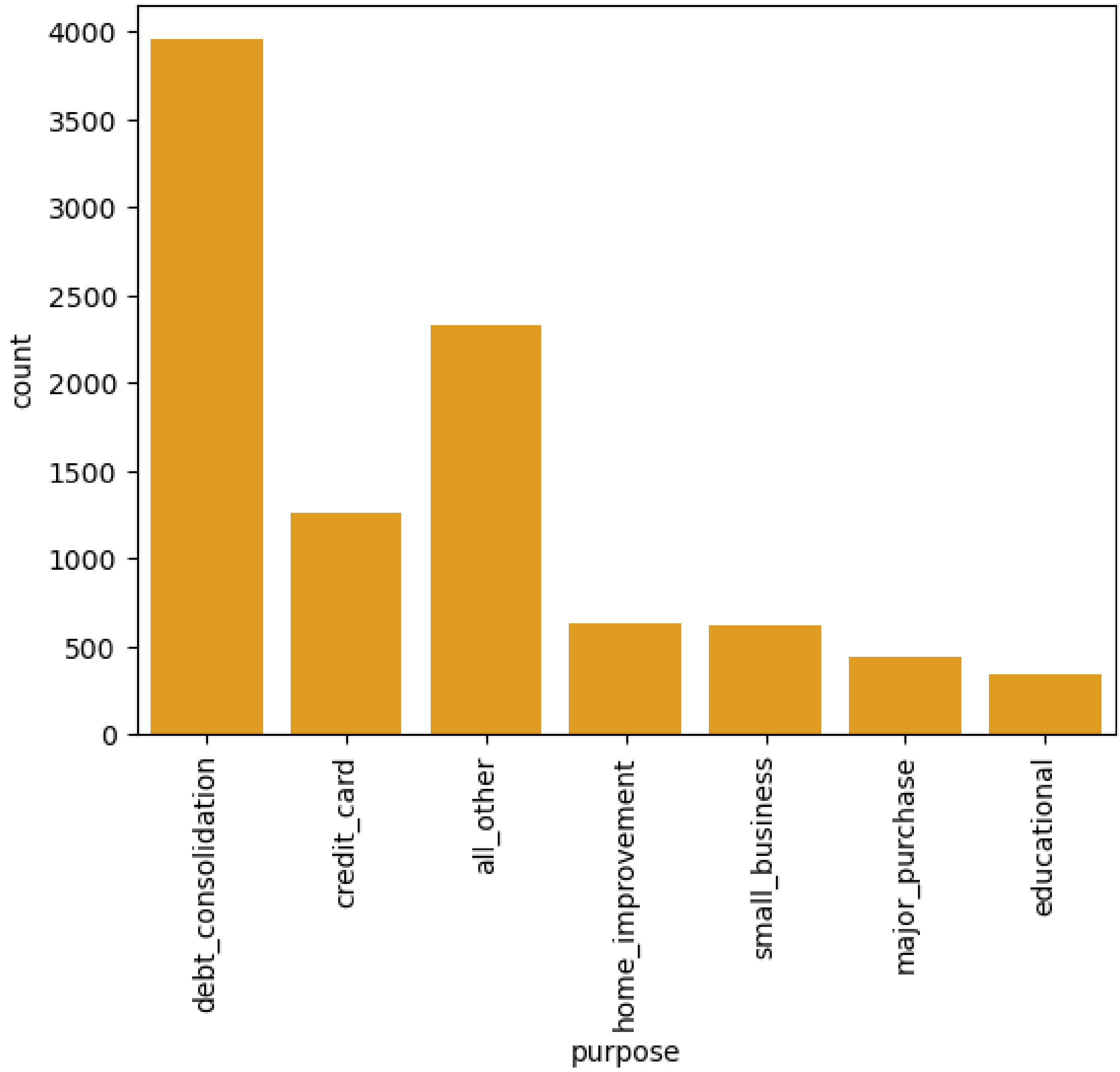


In [24]:

```
sns.countplot(x='purpose',color="orange",data=df)
plt.xticks(rotation = 90)
```

Out[24]:

```
(array([0, 1, 2, 3, 4, 5, 6]),
 [Text(0, 0, 'debt_consolidation'),
  Text(1, 0, 'credit_card'),
  Text(2, 0, 'all_other'),
  Text(3, 0, 'home_improvement'),
  Text(4, 0, 'small_business'),
  Text(5, 0, 'major_purchase'),
  Text(6, 0, 'educational')])
```

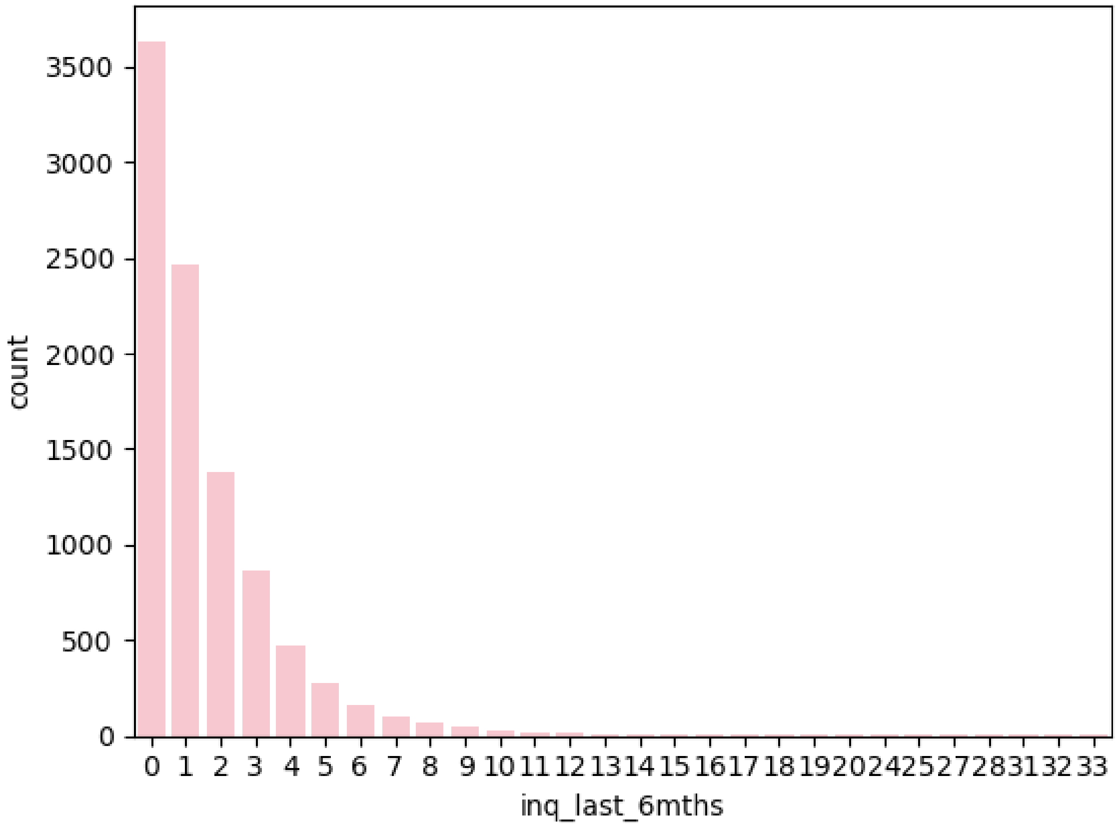


In [25]:

```
sns.countplot(x='inq_last_6mths',color="pink",data=df)
```

Out[25]:

```
<AxesSubplot:xlabel='inq_last_6mths', ylabel='count'>
```

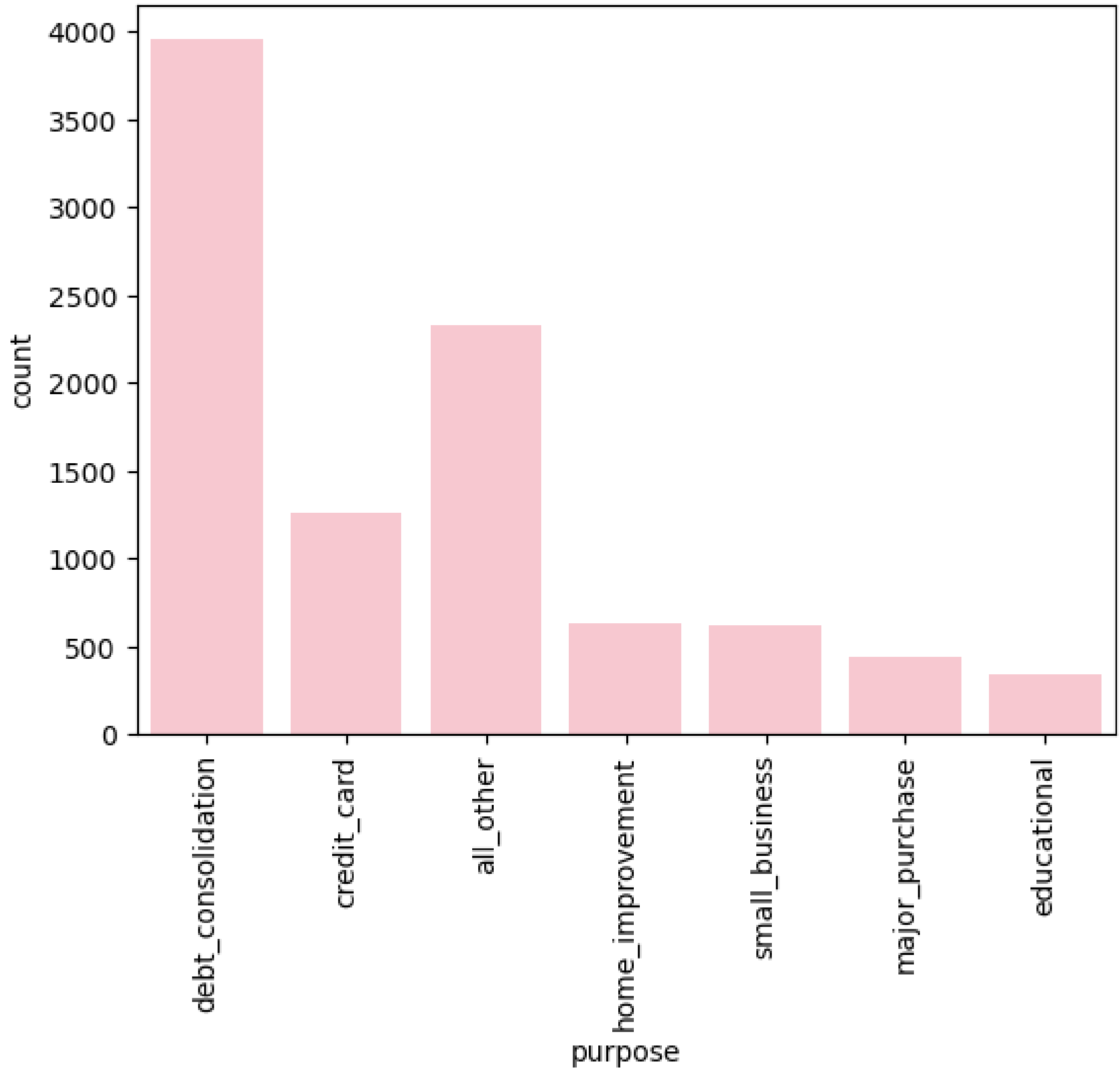


In [26]:

```
sns.countplot(x='purpose',color="pink",data=df)
plt.xticks(rotation=90)
```

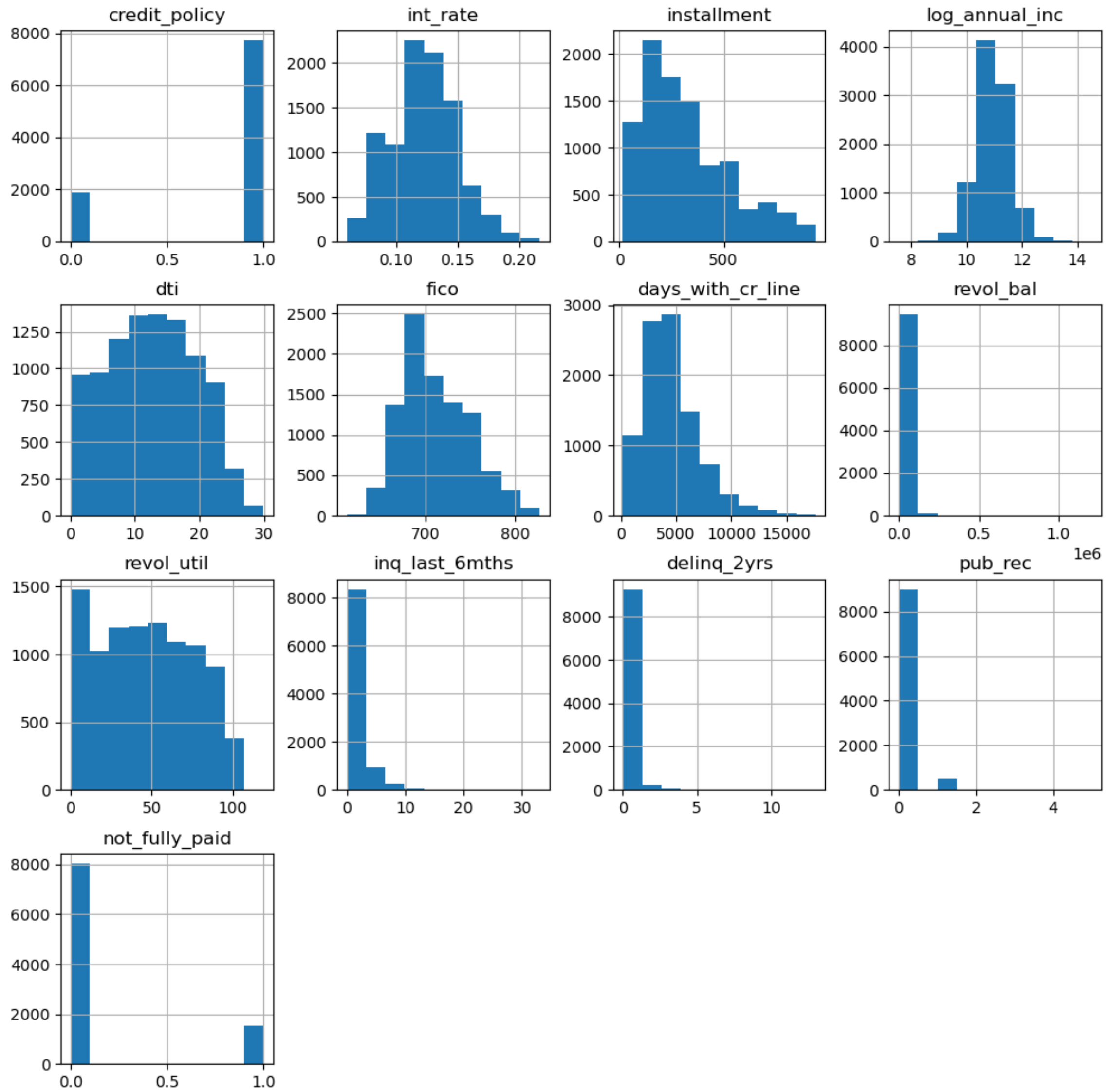
Out[26]:

```
(array([0, 1, 2, 3, 4, 5, 6]),
 [Text(0, 0, 'debt_consolidation'),
  Text(1, 0, 'credit_card'),
  Text(2, 0, 'all_other'),
  Text(3, 0, 'home_improvement'),
  Text(4, 0, 'small_business'),
  Text(5, 0, 'major_purchase'),
  Text(6, 0, 'educational')])
```



In [27]:

```
df.hist(figsize=(12,12))
plt.show()
```

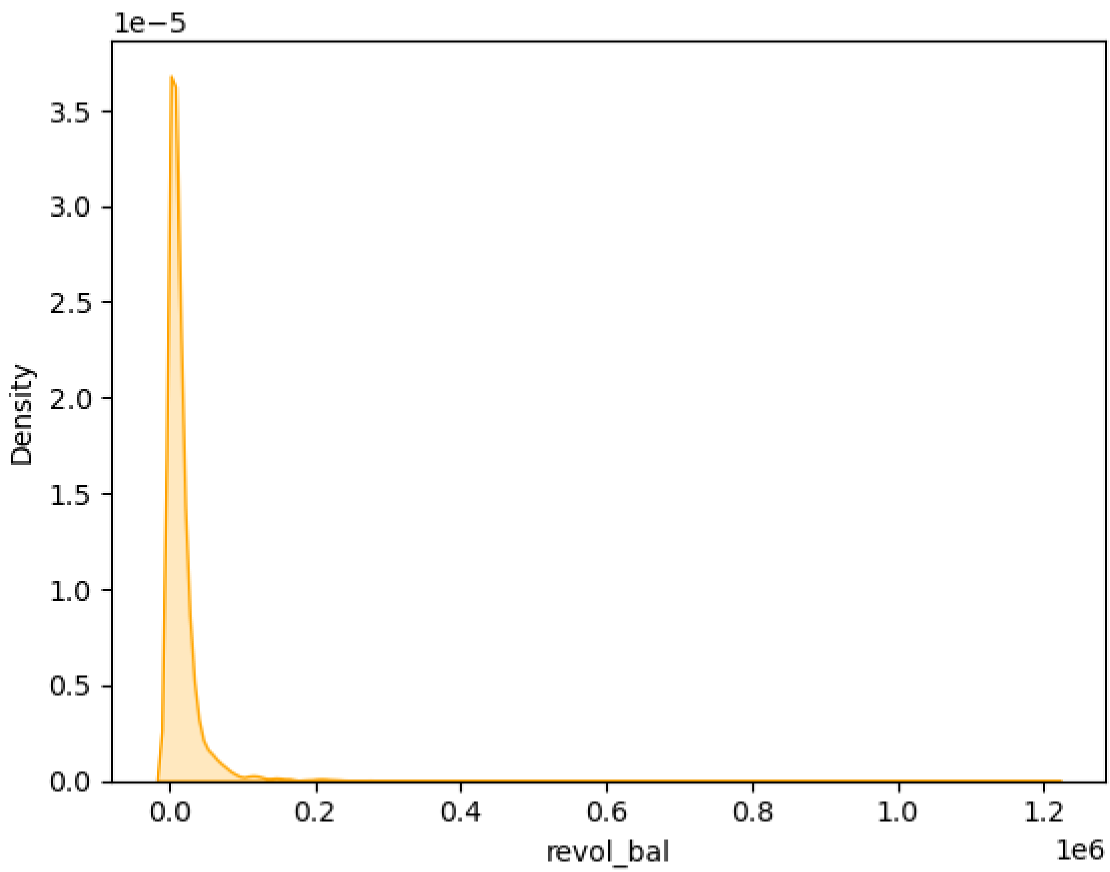


In [28]:

```
sns.kdeplot(df['revol_bal'],color='orange',shade=True)
```

Out[28]:

```
<AxesSubplot:xlabel='revol_bal', ylabel='Density'>
```



SPLIT DATA INTO Numerical column and Categorical Column

from the dataframe we split data in two categories first one numerical data and categorical data, so we can find out where data have outliers and skewness

In [29]:

```
catcol=[]
numcol=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        catcol.append(i)
    else:
        numcol.append(i)
```

In [30]:

```
catcol
```

Out[30]:

```
['purpose']
```

In [31]:

```
numcol
```

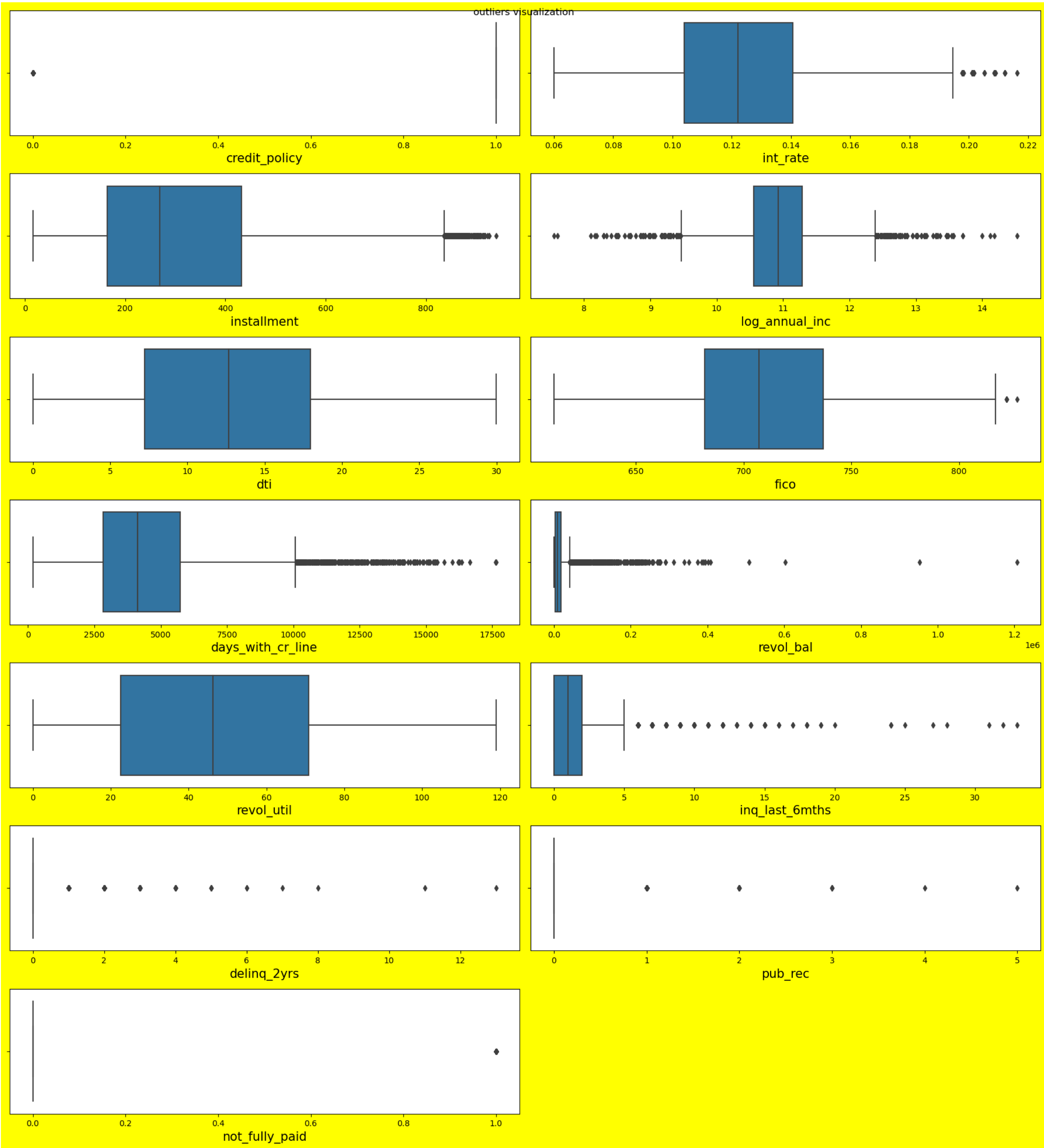
Out[31]:

```
['credit_policy',
 'int_rate',
 'installment',
 'log_annual_inc',
 'dti',
 'fico',
 'days_with_cr_line',
 'revol_bal',
 'revol_util',
 'inq_last_6mths',
 'delinq_2yrs',
 'pub_rec',
 'not_fully_paid']
```

PLOT BOXPLOT FOR NUMERIC VALUES TO FIND OUT OUTLIERS FROM COLUMN

In [32]:

```
plt.figure(figsize=(18,20),facecolor="yellow")
plt.suptitle('outliers visualization')
pltn=1
for i in numcol:
    if pltn<=13:
        ax=plt.subplot(7,2,pltn)
        sns.boxplot(df[i])
        plt.xlabel(i,fontsize=15)
    pltn=pltn+1
plt.tight_layout()
plt.show()
```



in column int_rate, installment, log_annual_inc, fico, days_with_cr_line, revol_bal, inq_last_6mths, delinq_2yrs, pub_rec, not_fully_paid we get outliers in these columns so we need to remove these outliers to regularize the data with the help of z-score

In [33]:

```
f=df[['credit_policy',
 'int_rate',
 'installment',
 'log_annual_inc',
 'dti',
 'fico',
 'days_with_cr_line',
 'revol_bal',
 'revol_util',
 'inq_last_6mths',
 'delinq_2yrs',
 'pub_rec',
 'not_fully_paid']]
```

In [34]:

```
from scipy.stats import zscore
z=abs(zscore(f))
```

In [35]:

```
z
```

Out[35]:

	credit_policy	int_rate	installment	log_annual_inc	dti	fico	days_with_cr_l
0	0.492222	0.139318	2.463099	0.680388	0.998505	0.688825	0.432
1	0.492222	0.578868	0.438854	0.244031	0.244540	0.101303	0.721
2	0.492222	0.486484	0.230708	0.908659	0.141885	0.759742	0.059
3	0.492222	0.813544	0.757022	0.680388	0.654697	0.030385	0.745
4	0.492222	0.743509	1.043992	0.597961	0.343326	1.154806	0.198
...	...	...	...	...	...	...	
9573	2.031603	0.873884	0.123976	2.031030	0.322023	1.023118	2.368
9574	2.031603	0.099083	0.296481	0.341170	1.800898	0.293761	0.072
9575	2.031603	0.578868	1.068670	0.545694	0.070213	0.628054	0.444
9576	2.031603	1.391660	0.156914	0.182730	0.954924	0.496366	1.105
9577	2.031603	0.616859	2.580601	0.540594	0.533633	0.557137	0.071

9578 rows × 13 columns

we remove the outliers with help of z-score now we introduce new dataframe 'newdf' where we have taken z-score less than 3

In [36]:

```
newdf=df[(z<3).all(axis=1)]
newdf
```

Out[36]:

	credit_policy	purpose	int_rate	installment	log_annual_inc	dti	fico	days_with_cr_line
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000
...	...	...	...	...	...	...	...	...
9572	0	debt_consolidation	0.1565	69.98	10.110472	7.02	662	8190.041667
9574	0	all_other	0.1253	257.70	11.141862	0.21	722	4380.000000
9575	0	debt_consolidation	0.1071	97.81	10.596635	13.09	687	3450.041667
9576	0	home_improvement	0.1600	351.58	10.819778	19.18	692	1800.000000

Encoding

we need to encode every column so we can correlate every parameter with each other

with the help of OrdinalEncoder we change the categorical data into numerical data

In [37]:

```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
newdf[catcol]=oe.fit_transform(newdf[catcol])
```

In [38]:

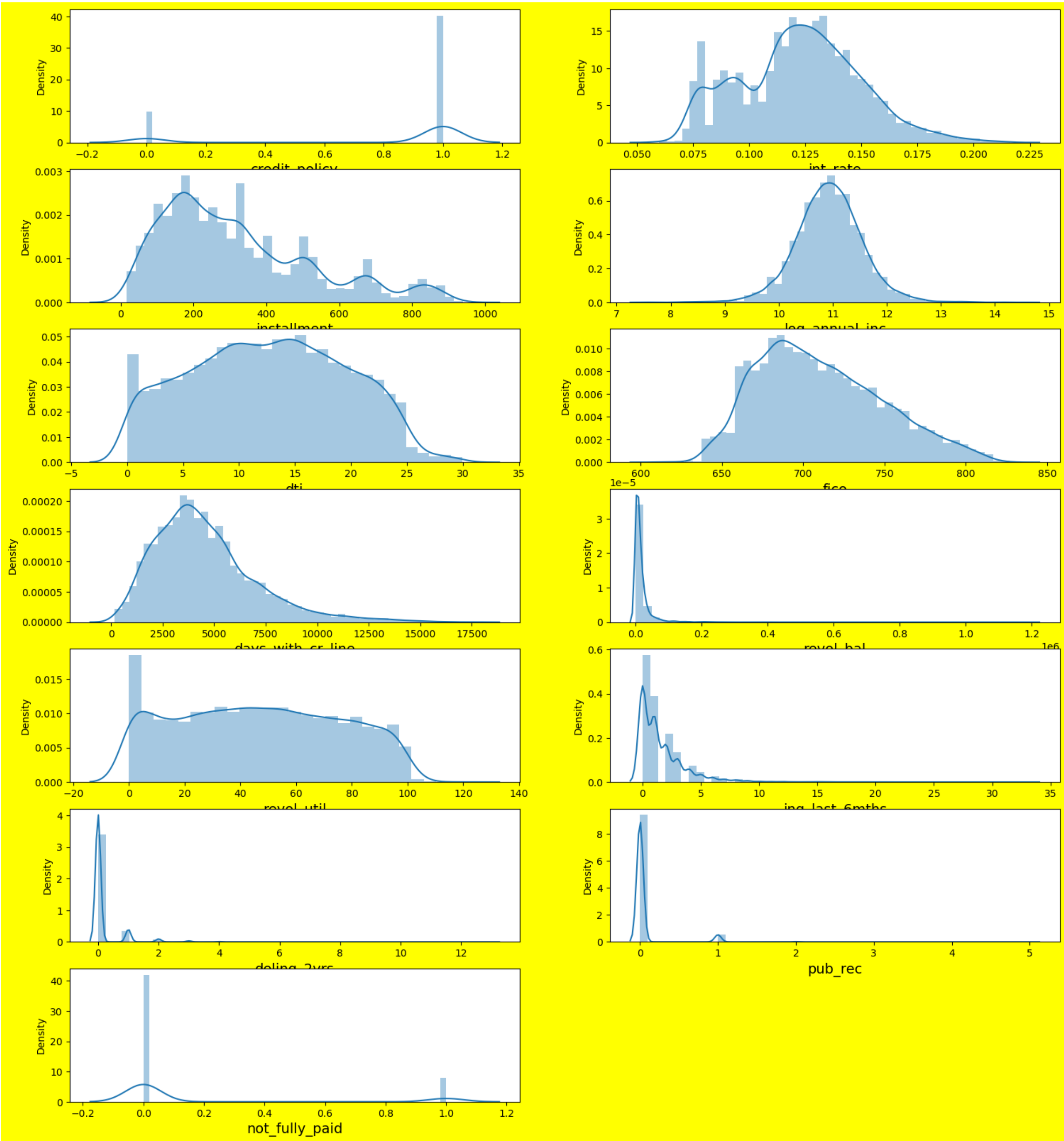
```
newdf.head()
```

Out[38]:

	credit_policy	purpose	int_rate	installment	log_annual_inc	dti	fico	days_with_cr_line
0	1	2.0	0.1189	829.10	11.350407	19.48	737	5639.95833
1	1	1.0	0.1071	228.22	11.082143	14.29	707	2760.00000
2	1	2.0	0.1357	366.86	10.373491	11.63	682	4710.00000
3	1	2.0	0.1008	162.34	11.350407	8.10	712	2699.95833
4	1	1.0	0.1426	102.92	11.299732	14.97	667	4066.00000

In [39]:

```
plt.figure(figsize=(18,20),facecolor='yellow')
plotn=1
for i in numcol:
    if plotn<=14:
        ax=plt.subplot(7,2,plotn)
        sns.distplot(df[i])
        plt.xlabel(i,fontsize=14)
        plotn=plotn+1
```



the above graph is skewness graph where shows that data is not symetric in column so with help of PowerTransformer(method='yoe-johnson') we can remove that skewed data and make data symmetric

FInd Out Skewness

In [40]:

```
newdf.skew()
```

Out[40]:

```
credit_policy       -1.829841
purpose              0.907314
int_rate             0.099503
installment          0.909451
log_annual_inc      -0.045202
dti                  0.016336
fico                 0.407359
days_with_cr_line    0.808196
revol_bal            2.552095
revol_util           0.067031
inq_last_6mths       1.535016
delinq_2yrs          2.877160
pub_rec              0.000000
not_fully_paid       1.965131
dtype: float64
```

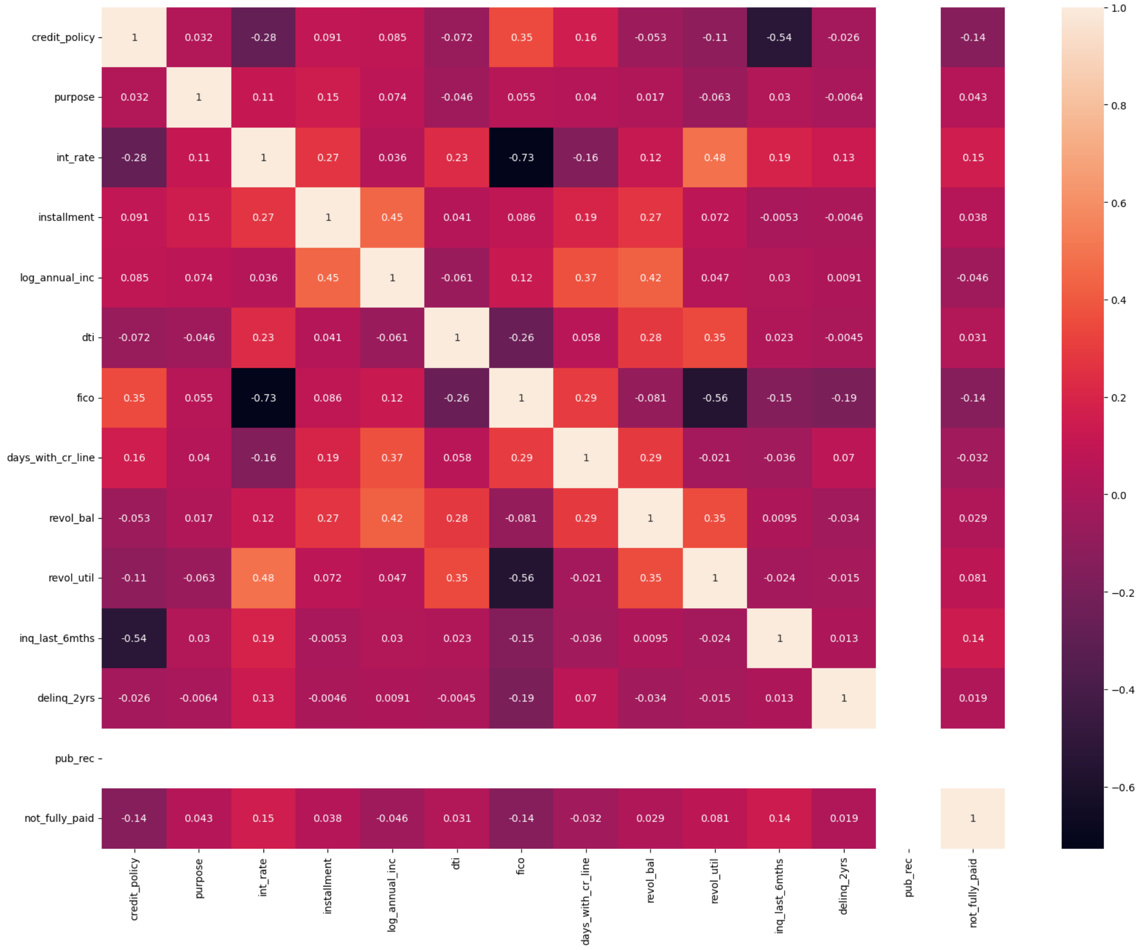
FIND OUT CORRELATION OF Features and Target using Heatmap

In [41]:

```
plt.figure(figsize=(20,15))
sns.heatmap(newdf.corr(),annot=True)
```

Out[41]:

<AxesSubplot:>



from dataset we conclude that we have skewness in Column credit.policy, int.rate, installment, log.annual.inc, dti, fico,days.with.cr.line, revol.bal, revol.util, inq.last.6mths, delinq.2yrs But purpose is stringly correlated with our target so dont remove skewness from them

In [42]:

```
s=['credit_policy', 'int_rate', 'installment', 'log_annual_inc', 'dti', 'fico','days_wit
from sklearn.preprocessing import PowerTransformer
scaler=PowerTransformer(method='yeo-johnson')
newdf[s]=scaler.fit_transform(newdf[s].values)
```

In [43]:

```
newdf.skew()
```

Out[43]:

```
credit_policy       -1.829841
purpose              0.907314
int_rate             0.003069
installment         -0.035951
log_annual_inc       0.003879
dti                 -0.193694
fico                 0.032343
days_with_cr_line   -0.005794
revol_bal           -0.060913
revol_util          -0.306038
inq_last_6mths       0.117221
delinq_2yrs          2.877160
pub_rec              0.000000
not_fully_paid       1.965131
dtype: float64
```

In [44]:

```
df.skew()
```

Out[44]:

```
credit_policy        -1.539621
int_rate              0.164420
installment           0.912522
log_annual_inc        0.028668
dti                   0.023941
fico                  0.471260
days_with_cr_line     1.155748
revol_bal            11.161058
revol_util            0.059985
inq_last_6mths        3.584151
delinq_2yrs           6.061793
pub_rec               5.126434
not_fully_paid        1.854592
dtype: float64
```

Split Data into Feature and Target

Here we seperate the target and features so we can keep the target as it is and we can scale or encode the categorical data in numeric format

In [45]:

```
x = newdf.drop('not_fully_paid',axis=1)
y = newdf['not_fully_paid']
```

In [46]:

```
x
```

Out[46]:

	credit_policy	purpose	int_rate	installment	log_annual_inc	dti	fico	da
0	0.440522	2.0	-0.060146	1.907813	0.779834	0.990123	0.694879	
1	0.440522	1.0	-0.515508	-0.247330	0.301546	0.295335	-0.075988	
2	0.440522	2.0	0.573498	0.454263	-0.945811	-0.079110	-0.815028	
3	0.440522	2.0	-0.762210	-0.694586	0.779834	-0.602574	0.060708	
4	0.440522	1.0	0.828896	-1.227711	0.689235	0.388835	-1.307606	
...	...	...	...	...	...	...	...	
9572	-2.270032	2.0	1.335083	-1.626501	-1.402685	-0.770586	-1.480859	
9574	-2.270032	0.0	0.183244	-0.076824	0.407734	-2.007047	0.323923	
9575	-2.270032	2.0	-0.515508	-1.282989	-0.555593	0.128218	-0.659394	
9576	-2.270032	4.0	1.460824	0.387386	-0.163013	0.951036	-0.507804	
9577	-2.270032	2.0	0.703395	1.965896	0.626249	0.566705	0.574276	

8282 rows × 13 columns

In [47]:

```
y
```

Out[47]:

```
0       0
1       0
2       0
3       0
4       0
       ..
9572    1
9574    1
9575    1
9576    1
9577    1
Name: not_fully_paid, Length: 8282, dtype: int64
```

In [48]:

```
newdf['credit_policy'].value_counts()
```

Out[48]:

```
 0.440522    6936
-2.270032    1346
Name: credit_policy, dtype: int64
```

In [49]:

```
newdf['purpose'].value_counts()
```

Out[49]:

```
2.0    3438
0.0    2037
1.0    1084
4.0     531
6.0     514
5.0     386
3.0     292
Name: purpose, dtype: int64
```

In [50]:

```
df['purpose'].value_counts()
```

Out[50]:

```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

In [51]:

```
newdf['pub_rec'].value_counts()
```

Out[51]:

```
0.0    8282
Name: pub_rec, dtype: int64
```

Split Data For Training AND Testing

from newdf we have give some data in training and testing to give data to test and train after applying Machine Learning algorithms

In [52]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=0)
```

In [53]:

```
from sklearn.metrics import classification_report, accuracy_score,confusion_matrix
```

In [54]:

```
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB,GaussianNB,BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
```

In [55]:

```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)

    train=model.score(xtrain,ytrain)
    test=model.score(xtest,ytest)

    print(f"Traning accuracy:{train}\n Testing accuracy:{test}\n\n")
    print(confusion_matrix(ytest,ypred))
    print(classification_report(ytest,ypred))
    print(f"Accuracy:{accuracy_score(ytest,ypred)}")
    return model
```

In [56]:

```
lr=mymodel(LogisticRegression())
```

```
Traning accuracy:0.8513023977919614
 Testing accuracy:0.8474849094567405


[[2106    0]
 [ 379    0]]
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2106
           1       0.00      0.00      0.00       379

    accuracy                           0.85      2485
   macro avg       0.42      0.50      0.46      2485
weighted avg       0.72      0.85      0.78      2485

Accuracy:0.8474849094567405
```

In [57]:

```
knn=mymodel(KNeighborsClassifier())
```

```
Traning accuracy:0.8621700879765396
 Testing accuracy:0.8338028169014085


[[2053   53]
 [ 360   19]]
              precision    recall  f1-score   support

           0       0.85      0.97      0.91      2106
           1       0.26      0.05      0.08       379

    accuracy                           0.83      2485
   macro avg       0.56      0.51      0.50      2485
weighted avg       0.76      0.83      0.78      2485

Accuracy:0.8338028169014085
```

In [58]:

```
bnb=mymodel(BernoulliNB())
```

```
Traning accuracy:0.8461273072278764
 Testing accuracy:0.8422535211267606


[[2081   25]
 [ 367   12]]
              precision    recall  f1-score   support

           0       0.85      0.99      0.91      2106
           1       0.32      0.03      0.06       379

    accuracy                           0.84      2485
   macro avg       0.59      0.51      0.49      2485
weighted avg       0.77      0.84      0.78      2485

Accuracy:0.8422535211267606
```

In [59]:

```
Gb=mymodel(GaussianNB())
```

```
Traning accuracy:0.790063826116957
 Testing accuracy:0.7879275653923541


[[1864  242]
 [ 285   94]]
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      2106
           1       0.28      0.25      0.26       379

    accuracy                           0.79      2485
   macro avg       0.57      0.57      0.57      2485
weighted avg       0.78      0.79      0.78      2485

Accuracy:0.7879275653923541
```

In [60]:

```
dc=mymodel(DecisionTreeClassifier())
```

```
Traning accuracy:1.0
 Testing accuracy:0.7553319919517103


[[1800  306]
 [ 302   77]]
              precision    recall  f1-score   support

           0       0.86      0.85      0.86      2106
           1       0.20      0.20      0.20       379

    accuracy                           0.76      2485
   macro avg       0.53      0.53      0.53      2485
weighted avg       0.76      0.76      0.76      2485

Accuracy:0.7553319919517103
```

In [61]:

```
XGB=mymodel(XGBClassifier(random_state=1,reg_alpha=1))
```

```
Traning accuracy:0.9843022252889425
 Testing accuracy:0.827364185110664


[[2036   70]
 [ 359   20]]
              precision    recall  f1-score   support

           0       0.85      0.97      0.90      2106
           1       0.22      0.05      0.09       379

    accuracy                           0.83      2485
   macro avg       0.54      0.51      0.49      2485
weighted avg       0.75      0.83      0.78      2485

Accuracy:0.827364185110664
```

In [62]:

```
Ada=mymodel(AdaBoostClassifier(random_state=1))
```

```
Traning accuracy:0.8540624460928066
 Testing accuracy:0.8426559356136821


[[2084   22]
 [ 369   10]]
              precision    recall  f1-score   support

           0       0.85      0.99      0.91      2106
           1       0.31      0.03      0.05       379

    accuracy                           0.84      2485
   macro avg       0.58      0.51      0.48      2485
weighted avg       0.77      0.84      0.78      2485

Accuracy:0.8426559356136821
```

In [63]:

```
lsvc=mymodel(LinearSVC(random_state=1))
```

```
Traning accuracy:0.851647403829567
 Testing accuracy:0.8474849094567405


[[2106    0]
 [ 379    0]]
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2106
           1       0.00      0.00      0.00       379

    accuracy                           0.85      2485
   macro avg       0.42      0.50      0.46      2485
weighted avg       0.72      0.85      0.78      2485

Accuracy:0.8474849094567405
```

In [64]:

```
lsvc=mymodel(LinearSVC(random_state=1,C=0.2))
```

```
Traning accuracy:0.851647403829567
 Testing accuracy:0.8474849094567405


[[2106    0]
 [ 379    0]]
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2106
           1       0.00      0.00      0.00       379

    accuracy                           0.85      2485
   macro avg       0.42      0.50      0.46      2485
weighted avg       0.72      0.85      0.78      2485

Accuracy:0.8474849094567405
```

In [65]:

```
Rc=mymodel(RandomForestClassifier())
```

```
Traning accuracy:0.9998274969811972
 Testing accuracy:0.847887323943662


[[2100    6]
 [ 372    7]]
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2106
           1       0.54      0.02      0.04       379

    accuracy                           0.85      2485
   macro avg       0.69      0.51      0.48      2485
weighted avg       0.80      0.85      0.78      2485

Accuracy:0.847887323943662
```

we have applied above classifier algorithms where we get a best accuracy with RandomForestClassifier which is 0.845

Hyperparameter Tunning for RandomForestClassifier

we have to improve above accuracy in RandomForestClassifier with the help of Hyperparameter Tunning and related Parameters

In [66]:

```
Rc=mymodel(RandomForestClassifier(min_samples_leaf=20,max_depth=15,n_estimators=30))
```

```
Traning accuracy:0.851647403829567
 Testing accuracy:0.8474849094567405


[[2106    0]
 [ 379    0]]
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2106
           1       0.00      0.00      0.00       379

    accuracy                           0.85      2485
   macro avg       0.42      0.50      0.46      2485
weighted avg       0.72      0.85      0.78      2485

Accuracy:0.8474849094567405
```

In [67]:

```
for i in range(1,50):
    dt1=RandomForestClassifier(max_depth=i)
    dt1.fit(xtrain,ytrain)
    ypred=dt1.predict(xtest)

    train=dt1.score(xtrain,ytrain)
    test=dt1.score(xtest,ytest)

    print(f"{i}    {train}    {test}")
```

```
1    0.851647403829567    0.8474849094567405
2    0.851647403829567    0.8474849094567405
3    0.851647403829567    0.8474849094567405
4    0.851647403829567    0.8474849094567405
5    0.8519924098671727    0.8474849094567405
6    0.8523374159047783    0.8474849094567405
7    0.8575125064688632    0.8474849094567405
8    0.8630326030705537    0.8470824949698189
9    0.8720027600483008    0.8466800804828973
10    0.8785578747628083    0.8466800804828973
11    0.8858030015525271    0.8482897384305835
12    0.8968431947559082    0.8470824949698189
13    0.9089184060721063    0.8470824949698189
14    0.9225461445575297    0.8474849094567405
15    0.9322063136104882    0.8462776659959759
16    0.9387614283249957    0.8458752515090543
17    0.953596687942039    0.8446680080482898
18    0.9618768328445748    0.8466800804828973
19    0.9708469898223219    0.847887323943662
20    0.9822321890633086    0.8458752515090543
21    0.9886148007590133    0.8442655935613682
22    0.9922373641538726    0.847887323943662
23    0.9953424184923236    0.8450704225352113
24    0.9984474728307745    0.8458752515090543
25    0.9993099879247886    0.8466800804828973
26    0.9996549939623943    0.8474849094567405
27    1.0    0.8438631790744466
28    1.0    0.8474849094567405
29    0.9998274969811972    0.8470824949698189
30    1.0    0.8466800804828973
31    0.9998274969811972    0.8474849094567405
32    1.0    0.8474849094567405
33    1.0    0.8454728370221328
34    1.0    0.8470824949698189
35    1.0    0.8458752515090543
36    1.0    0.8482897384305835
37    1.0    0.8462776659959759
38    1.0    0.8450704225352113
39    1.0    0.8454728370221328
40    1.0    0.8470824949698189
41    1.0    0.8470824949698189
42    1.0    0.8458752515090543
43    1.0    0.8470824949698189
44    1.0    0.8466800804828973
45    1.0    0.848692152917505
46    1.0    0.8462776659959759
47    1.0    0.8458752515090543
48    1.0    0.8458752515090543
49    1.0    0.8450704225352113
```

In [68]:

```
for i in range(1,50):
    dt1=RandomForestClassifier(min_samples_leaf=i)
    dt1.fit(xtrain,ytrain)
    ypred=dt1.predict(xtest)

    train=dt1.score(xtrain,ytrain)
    test=dt1.score(xtest,ytest)
    print(f"{i}    {train}    {test}")
```

```
1    1.0    0.8462776659959759
2    0.9651543902018286    0.848692152917505
3    0.9075383819216837    0.847887323943662
4    0.8792478868380197    0.8474849094567405
5    0.8692427117474556    0.8470824949698189
6    0.8637226151457651    0.847887323943662
7    0.8573400034500603    0.8474849094567405
8    0.8561324823184405    0.8474849094567405
9    0.855097464205236    0.8474849094567405
10    0.8531999309987924    0.8470824949698189
11    0.8521649128859755    0.8474849094567405
12    0.8519924098671727    0.8474849094567405
13    0.8518199068483698    0.8474849094567405
14    0.8518199068483698    0.8474849094567405
15    0.8518199068483698    0.8474849094567405
16    0.851647403829567    0.8474849094567405
17    0.8518199068483698    0.8474849094567405
18    0.851647403829567    0.8474849094567405
19    0.851647403829567    0.8474849094567405
20    0.851647403829567    0.8474849094567405
21    0.851647403829567    0.8474849094567405
22    0.851647403829567    0.8474849094567405
23    0.851647403829567    0.8474849094567405
24    0.851647403829567    0.8474849094567405
25    0.851647403829567    0.8474849094567405
26    0.851647403829567    0.8474849094567405
27    0.851647403829567    0.8474849094567405
28    0.851647403829567    0.8474849094567405
29    0.851647403829567    0.8474849094567405
30    0.851647403829567    0.8474849094567405
31    0.851647403829567    0.8474849094567405
32    0.851647403829567    0.8474849094567405
33    0.851647403829567    0.8474849094567405
34    0.851647403829567    0.8474849094567405
35    0.851647403829567    0.8474849094567405
36    0.851647403829567    0.8474849094567405
37    0.851647403829567    0.8474849094567405
38    0.851647403829567    0.8474849094567405
39    0.851647403829567    0.8474849094567405
40    0.851647403829567    0.8474849094567405
41    0.851647403829567    0.8474849094567405
42    0.851647403829567    0.8474849094567405
43    0.851647403829567    0.8474849094567405
44    0.851647403829567    0.8474849094567405
45    0.851647403829567    0.8474849094567405
46    0.851647403829567    0.8474849094567405
47    0.851647403829567    0.8474849094567405
48    0.851647403829567    0.8474849094567405
49    0.851647403829567    0.8474849094567405
```

In [69]:

```
dt3=mymodel(RandomForestClassifier(criterion="entropy",max_depth=1,min_samples_leaf=9))
```

```
Traning accuracy:0.851647403829567
 Testing accuracy:0.8474849094567405


[[2106    0]
 [ 379    0]]
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2106
           1       0.00      0.00      0.00       379

    accuracy                           0.85      2485
   macro avg       0.42      0.50      0.46      2485
weighted avg       0.72      0.85      0.78      2485

Accuracy:0.8474849094567405
```

from parameters of RandomForestClassifier we try to improve accuracy and we picked up the best parameter values as criterion="entropy",max_depth=1,min_samples_leaf=9.

In [70]:

```
parameters={"criterion":['gini','entropy'],
            "max_depth":list(range(1,50,5)),
            "min_samples_leaf":list(range(1,50,5))
}
```

In [71]:

```
from sklearn.model_selection import GridSearchCV
grid=GridSearchCV(DecisionTreeClassifier(),parameters)
grid.fit(xtrain,ytrain)
```

Out[71]:

```
GridSearchCV(estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 6, 11, 16, 21, 26, 31, 36, 41, 4
6],
                         'min_samples_leaf': [1, 6, 11, 16, 21, 26, 31, 3
6, 41,
                                              46]})
```

In [72]:

```
dt3=mymodel(RandomForestClassifier(criterion="entropy",max_depth=46,min_samples_leaf=46)
```

```
Traning accuracy:0.851647403829567
 Testing accuracy:0.8474849094567405


[[2106    0]
 [ 379    0]]
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2106
           1       0.00      0.00      0.00       379

    accuracy                           0.85      2485
   macro avg       0.42      0.50      0.46      2485
weighted avg       0.72      0.85      0.78      2485

Accuracy:0.8474849094567405
```

I have applied all type of algorithms and related parameters where we get best accuracy=0.847 with RandomForestClassifier

In [ ]:

```

```