METRO COLLEGE OF TECHNOLOGY

Project For Predicting Borrowers Paying Back Fully using Logistic Regression Machine Learning Model in Python Programming

---

Submitted by:

Mohammad Monjur-E-Elahi

Course: Data Mining- Advanced Statistical Modeling [DSA09]

Program: Data Science and Application - Advanced Diploma [6060]

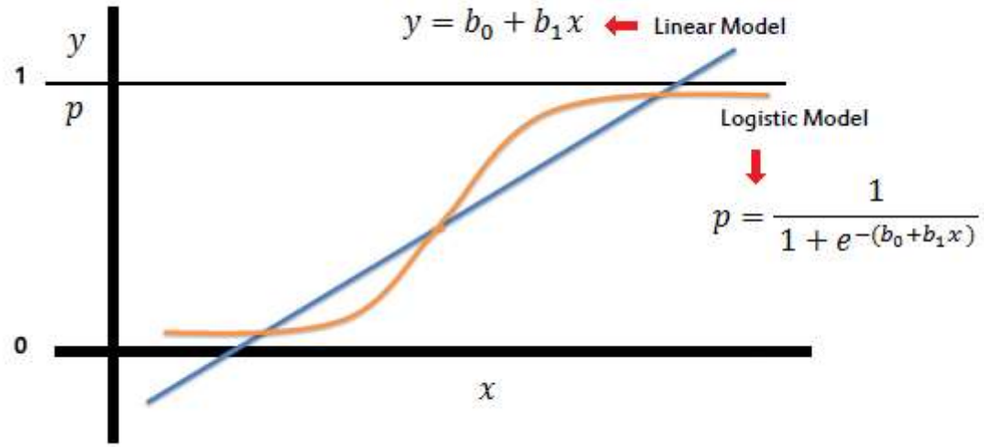
Metro College of Technology

Date: 30 April, 2021

Introduction:

In this project we will be exploring publicly available data from LendingClub.com. Lending Club connects people who need money (borrowers) with people who have money (investors). As an investor anyone would want to invest in people who showed a profile of having a high probability of paying him back. We will try to create a model that will help predict this.

We will be performing EDA, Feature Engineering and Visualization while and finally we will develop and logistic regression Machine Learning model along with performing the evaluation of the model.



We will start with importing the the requied libraries and modules.

In [2]:
```
import os
import numpy as np
import pandas as pd
import missingno as msno
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 600
plt.rcParams['savefig.dpi'] = 600
import seaborn as sns
sns.set(rc={"figure.dpi":600, 'savefig.dpi':600})
sns.set_context('notebook')
sns.set_style("ticks")
%matplotlib inline
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

Then we need to set the working directory for our project.

In [3]:
```
os.chdir(r'C:\\Users\\ruzdomain\\Desktop\\DMASM\\PROJECT\\LogisticRegression')
```

In [4]:
```
os.getcwd()
```

Out[4]:
```
'C:\\Users\\ruzdomain\\Desktop\\DMASM\\PROJECT\\LogisticRegression'
```

Getting the Data

Let us read in the Loan_data.csv file and set it to a panada data frame called loans.

In [5]:
```
loans = pd.read_csv('loan_data.csv',na_values  = 'NA')
```

We had to familarize ourselves with the different columns and what they meant.

In [6]:
```
loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9620 entries, 0 to 9619
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit.policy      9596 non-null   float64
 1   purpose            9591 non-null   object 
 2   int.rate           9596 non-null   float64
 3   installment        9596 non-null   float64
 4   log.annual.inc     9596 non-null   float64
 5   dti                9596 non-null   float64
 6   fico               9596 non-null   float64
 7   days.with.cr.line  9596 non-null   float64
 8   revol.bal          9596 non-null   float64
 9   revol.util         9596 non-null   float64
 10  inq.last.6mths     9596 non-null   float64
 11  delinq.2yrs        9596 non-null   float64
 12  pub.rec            9596 non-null   float64
 13  not.fully.paid     9588 non-null   float64
dtypes: float64(13), object(1)
memory usage: 1.0+ MB
```

Here are what the columns represent:

credit.policy: 1 if the customer meets the credit underwriting criteria of LendingClub.com, and 0 otherwise.

purpose: The purpose of the loan (takes values "credit_card", "debt_consolidation", "educational", "major_purchase", "small_business", and "all_other").

int.rate: The interest rate of the loan, as a proportion (a rate of 11% would be stored as 0.11). Borrowers judged by LendingClub.com to be more risky are assigned higher interest rates.

installment: The monthly installments owed by the borrower if the loan is funded.

log.annual.inc: The natural log of the self-reported annual income of the borrower.

dti: The debt-to-income ratio of the borrower (amount of debt divided by annual income).

fico: The FICO(Fair Isaac Corporation) credit score of the borrower. The base FICO® Scores range from 300 to 850.

days.with.cr.line: The number of days the borrower has had a credit line.

revol.bal: The borrower's revolving balance (amount unpaid at the end of the credit card billing cycle).

revol.util: The borrower's revolving line utilization rate (the amount of the credit line used relative to total credit available).

inq.last.6mths: The borrower's number of inquiries by creditors in the last 6 months.

delinq.2yrs: The number of times the borrower had been 30+ days past due on a payment in the past 2 years.

pub.rec: The borrower's number of derogatory public records (bankruptcy filings, tax liens, or judgments).

not.fully.paid: It is 1 if the borrower failed to payback fully and 0 if they are in good terms.

Did we explore various parameters of the dataset?

In [7]:
```
type(loans)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
loans.describe()
```

Out[8]:

	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
count	9596.000000	9596.000000	9596.000000	9596.000000	9596.000000	9596.000000	9596.000000	9.596000e+03	9596.000000	9596.000000	9596.000000	9596.000000	9588.000000
mean	0.805336	0.122632	319.092934	10.932314	12.608509	710.861088	4560.837840	1.690692e+04	46.796496	1.577011	0.163401	0.062109	0.160096
std	0.395963	0.026842	207.034780	0.614649	6.881133	37.959914	2497.099720	3.372999e+04	29.011032	2.198822	0.545748	0.262068	0.366714
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.103900	163.770000	10.558414	7.220000	682.000000	2820.000000	3.190750e+03	22.600000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.122100	268.950000	10.928884	12.670000	707.000000	4139.500000	8.593000e+03	46.250000	1.000000	0.000000	0.000000	0.000000
75%	1.000000	0.140700	432.487500	11.294769	17.950000	737.000000	5730.000000	1.824450e+04	70.900000	2.000000	0.000000	0.000000	0.000000
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	119.000000	33.000000	13.000000	5.000000	1.000000

In [9]:
```
loans.head(10)
```

Out[9]:

	credit.policy	purpose	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
0	1.0	debt_consolidation	0.1189	829.10	11.350407	19.48	737.0	5639.958333	28854.0	52.1	0.0	0.0	0.0	0.0
1	1.0	credit_card	0.1071	228.22	11.082143	14.29	707.0	2760.000000	33623.0	76.7	0.0	0.0	0.0	0.0
2	1.0	debt_consolidation	0.1357	366.86	10.373491	11.63	682.0	4710.000000	3511.0	25.6	1.0	0.0	0.0	0.0
3	1.0	debt_consolidation	0.1008	162.34	11.350407	8.10	712.0	2699.958333	33667.0	73.2	1.0	0.0	0.0	0.0
4	1.0	credit_card	0.1426	102.92	11.299732	14.97	667.0	4066.000000	4740.0	39.5	0.0	1.0	0.0	0.0
5	1.0	credit_card	0.0788	125.13	11.904968	16.98	727.0	6120.041667	50807.0	51.0	0.0	0.0	0.0	0.0
6	1.0	debt_consolidation	0.1496	194.02	10.714418	4.00	667.0	3180.041667	3839.0	76.8	0.0	0.0	1.0	1.0
7	1.0	all_other	0.1114	131.22	11.002100	11.08	722.0	5116.000000	24220.0	68.6	0.0	0.0	0.0	1.0
8	1.0	home_improvement	0.1134	87.19	11.407565	17.25	682.0	3989.000000	69909.0	51.1	1.0	0.0	0.0	0.0
9	1.0	debt_consolidation	0.1221	84.12	10.203592	10.00	707.0	2730.041667	5630.0	23.0	1.0	0.0	0.0	0.0

In [10]:
```
loans.tail(10)
```

Out[10]:

	credit.policy	purpose	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
9610	0.0	all_other	0.1979	37.06	10.645425	22.17	667.0	5916.000000	28854.0	59.8	6.0	0.0	1.0	0.0
9611	0.0	home_improvement	0.1426	823.34	12.429216	3.62	722.0	3239.958333	33575.0	83.9	5.0	0.0	0.0	1.0
9612	0.0	all_other	0.1671	113.63	10.645425	28.06	672.0	3210.041667	25759.0	63.8	5.0	0.0	0.0	1.0
9613	0.0	all_other	0.1568	161.01	11.225243	8.00	677.0	7230.000000	6909.0	29.2	4.0	0.0	1.0	1.0
9614	0.0	debt_consolidation	0.1565	69.98	10.110472	7.02	662.0	8190.041667	2999.0	39.5	6.0	0.0	0.0	1.0
9615	0.0	all_other	0.1461	344.76	12.180755	10.39	672.0	10474.000000	215372.0	82.1	2.0	0.0	0.0	1.0
9616	0.0	all_other	0.1253	257.70	11.141862	0.21	722.0	4380.000000	184.0	1.1	5.0	0.0	0.0	1.0
9617	0.0	debt_consolidation	0.1071	97.81	10.596635	13.09	687.0	3450.041667	10036.0	82.9	8.0	0.0	0.0	1.0
9618	0.0	home_improvement	0.1600	351.58	10.819778	19.18	692.0	1800.000000	0.0	3.2	5.0	0.0	0.0	1.0
9619	0.0	debt_consolidation	0.1392	853.43	11.264464	16.28	732.0	4740.000000	37879.0	57.0	6.0	0.0	0.0	1.0

In [11]:
```
loans.shape
```

Out[11]:
```
(9620, 14)
```

In [12]:
```
loans.columns
```

Out[12]:
```
Index(['credit.policy', 'purpose', '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'],
      dtype='object')
```

In [13]:
```
loans.dtypes
```

Out[13]:
```
credit.policy        float64
purpose               object
int.rate             float64
installment          float64
log.annual.inc       float64
dti                  float64
fico                 float64
days.with.cr.line    float64
revol.bal            float64
revol.util           float64
inq.last.6mths       float64
delinq.2yrs          float64
pub.rec              float64
not.fully.paid       float64
dtype: object
```

Did we opt to change the column names?

In [14]:
```
loans.columns
```

Out[14]:
```
Index(['credit.policy', 'purpose', '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'],
      dtype='object')
```

In [15]:
```
loans=loans.rename(columns={'credit.policy':"CreditPolicy",'purpose':'Purpose', 'int.rate':'InterestRate',
                            'installment':'Installment', 'log.annual.inc':'AnnualIncomeLn', 'dti':'DebtToIncome',
                            'fico':'FICO.Score', 'days.with.cr.line':'DaysSinceSubscription', 'revol.bal': 'RevolvingBalance',
                            'revol.util':"DebtToLimitRatio", 'inq.last.6mths':'InquirySixMonths',
                            'delinq.2yrs':'DelayedPaymentsTwoYears', 'pub.rec':'DerogatoryPublicRecords',
                            'not.fully.paid':'NotFullyPaid'})
```

In [16]:
```
loans.columns
```

Out[16]:
```
Index(['CreditPolicy', 'Purpose', 'InterestRate', 'Installment',
       'AnnualIncomeLn', 'DebtToIncome', 'FICO.Score', 'DaysSinceSubscription',
       'RevolvingBalance', 'DebtToLimitRatio', 'InquirySixMonths',
       'DelayedPaymentsTwoYears', 'DerogatoryPublicRecords', 'NotFullyPaid'],
      dtype='object')
```

In [17]:
```
loans.dtypes
```

Out[17]:
```
CreditPolicy              float64
Purpose                    object
InterestRate              float64
Installment               float64
AnnualIncomeLn            float64
DebtToIncome              float64
FICO.Score                float64
DaysSinceSubscription     float64
RevolvingBalance          float64
DebtToLimitRatio          float64
InquirySixMonths          float64
DelayedPaymentsTwoYears   float64
DerogatoryPublicRecords   float64
NotFullyPaid              float64
dtype: object
```

Did we deal with null and missing values?

In [18]:
```
loans.apply(lambda x: sum(x.isnull()))
```

Out[18]:
```
CreditPolicy              24
Purpose                   29
InterestRate              24
Installment               24
AnnualIncomeLn            24
DebtToIncome              24
FICO.Score                24
DaysSinceSubscription     24
RevolvingBalance          24
DebtToLimitRatio          24
InquirySixMonths          24
DelayedPaymentsTwoYears   24
DerogatoryPublicRecords   24
NotFullyPaid              32
dtype: int64
```

In [19]:
```
def percentage_of_miss(df):
  df1=df[df.columns[df.isnull().sum()>=1]]
  total_miss = df1.isnull().sum().sort_values(ascending=False)
  percent_miss = (df1.isnull().sum()/df1.isnull().count()).sort_values(ascending=False)
  missing_data = pd.concat([total_miss, percent_miss], axis=1, keys=['Number of Missing', 'Percentage'])
  return(missing_data)
```

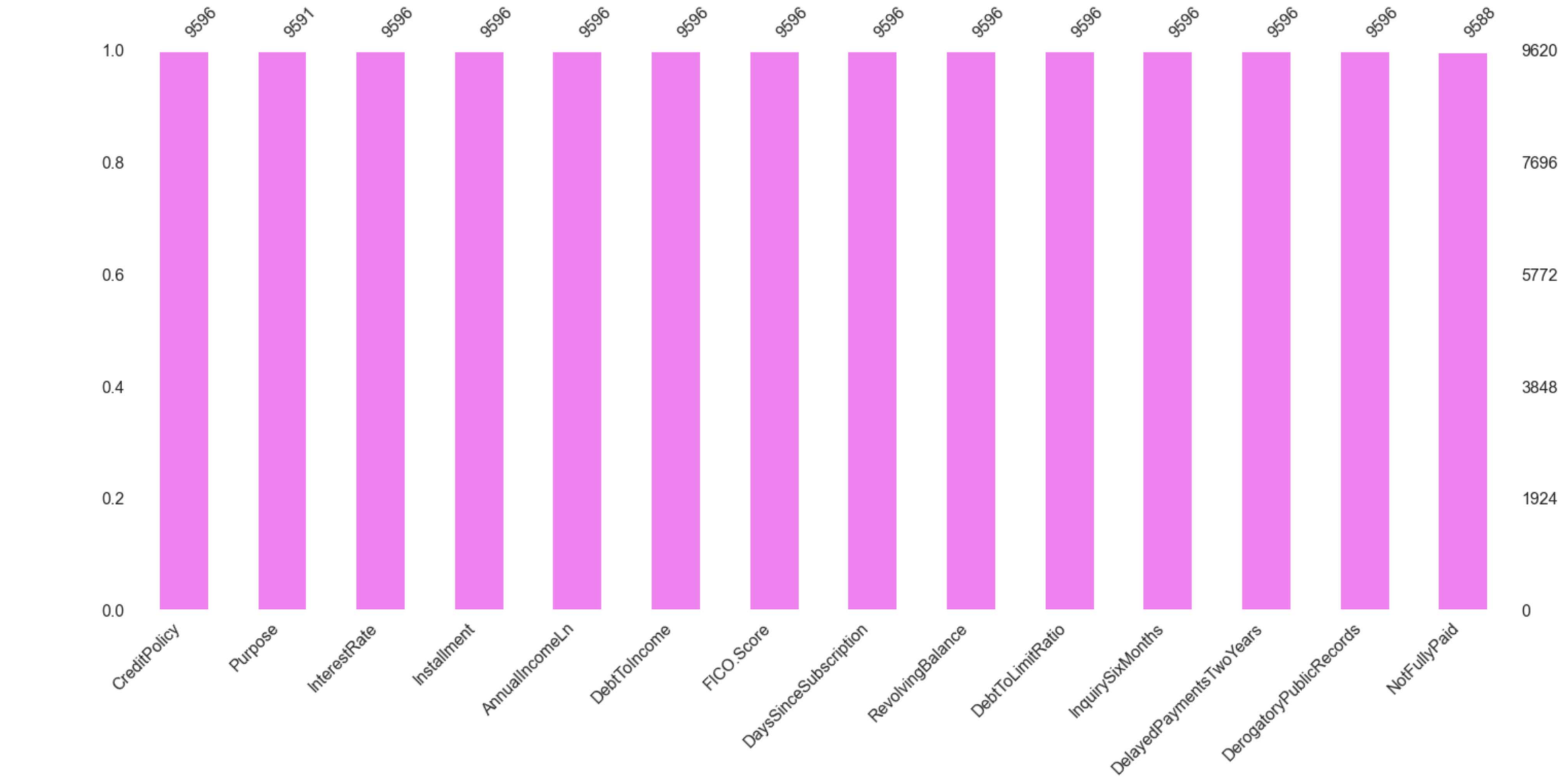
In [20]:
```
percentage_of_miss(loans)
```

Out[20]:

	Number of Missing	Percentage
NotFullyPaid	32	0.003326
Purpose	29	0.003015
DerogatoryPublicRecords	24	0.002495
DelayedPaymentsTwoYears	24	0.002495
InquirySixMonths	24	0.002495
DebtToLimitRatio	24	0.002495
RevolvingBalance	24	0.002495
DaysSinceSubscription	24	0.002495
FICO.Score	24	0.002495
DebtToIncome	24	0.002495
AnnualIncomeLn	24	0.002495
Installment	24	0.002495
InterestRate	24	0.002495
CreditPolicy	24	0.002495

In [21]:
```
msno.bar(loans, color ='violet')
```

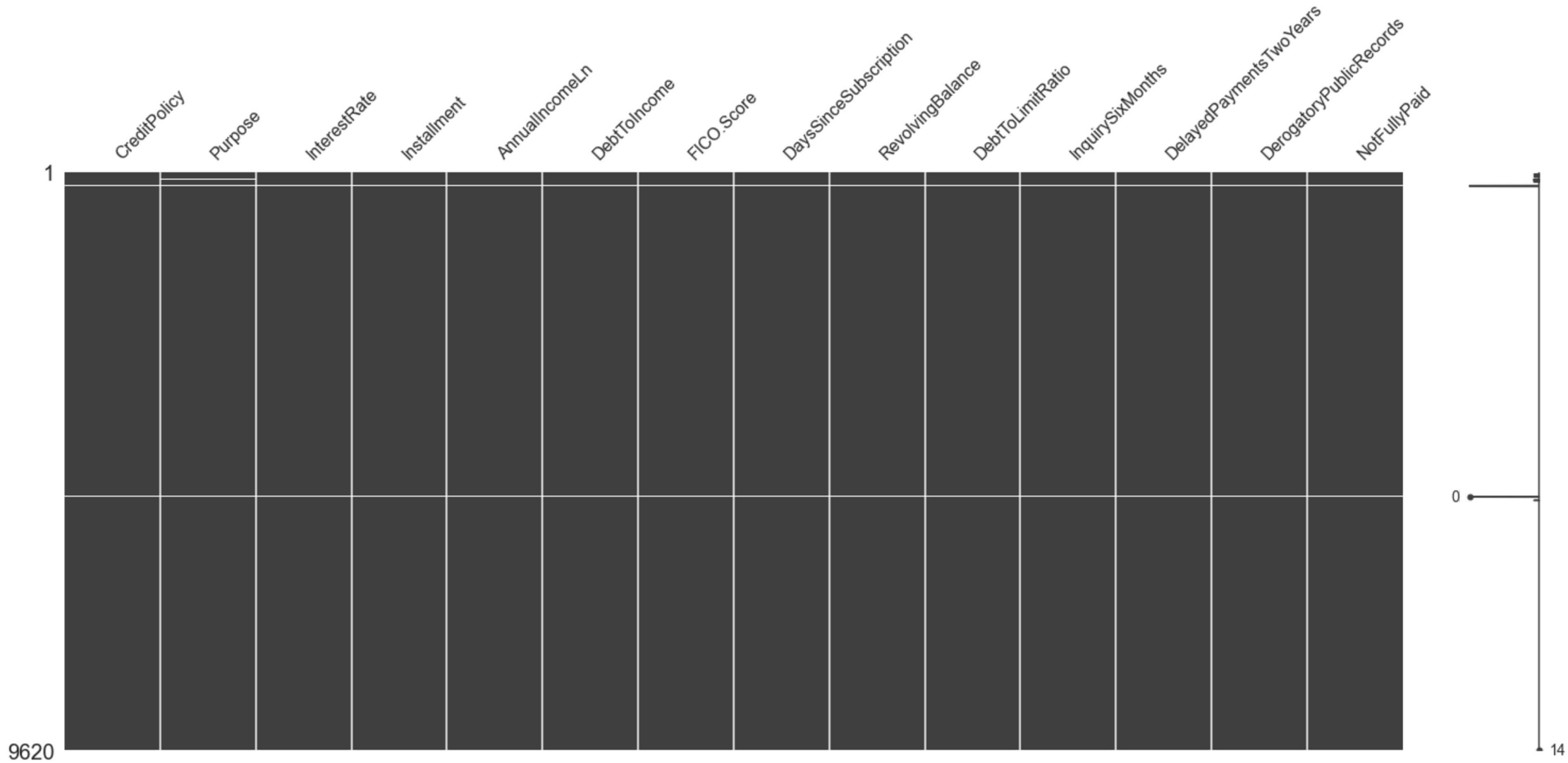
Out[21]: <AxesSubplot:>



Let us have alook at the missing values visually with the help of missingno package. Here, the horizontal line will depict the missing values.

In [22]:
```
msno.matrix(loans)
```

Out[22]: <AxesSubplot:>



Let us drop the observation that contains missing value for all the columns.

In [23]:
```
loans = loans.dropna( how='all')
```

In [24]:
```
percentage_of_miss(loans)
```

Out[24]:

	Number of Missing	Percentage
NotFullyPaid	8	0.000834
Purpose	5	0.000521

Let us now delete all the rows where the NotFullyPaid columns have missing values since this is our target column for training and testing the model.

In [25]:
```
loans = loans.dropna( how='any',
                      subset=['NotFullyPaid'])
```

In [26]:
```
percentage_of_miss(loans)
```

Out[26]:

	Number of Missing	Percentage
Purpose	5	0.000521

Let us look into the categories in "Purpose" column.

In [27]:
```
loans["Purpose"].value_counts()
```

Out[27]:
```
debt_consolidation    3959
all_other             2328
credit_card           1265
home_improvement       629
small_business         620
major_purchase         438
educational            344
Name: Purpose, dtype: int64
```

Let us check whether thre are duplicated rows and drop those.

In [28]:
```
duplicateRowsLoans = loans[loans.duplicated()]
duplicateRowsLoans
```

Out[28]:

	CreditPolicy	Purpose	InterestRate	Installment	AnnualIncomeLn	DebtToIncome	FICO.Score	DaysSinceSubscription	RevolvingBalance	DebtToLimitRatio	InquirySixMonths	DelayedP
4879	1.0	debt_consolidation	0.1218	93.24	11.461632	16.96	697.0	8130.000000	40868.0	86.6	3.0	
4880	1.0	credit_card	0.1218	549.45	11.225243	16.06	717.0	5039.958333	43373.0	66.9	0.0	
4881	1.0	debt_consolidation	0.1218	499.50	10.819778	21.24	722.0	3300.000000	7322.0	31.8	0.0	
4882	1.0	educational	0.1183	410.87	11.944708	17.38	717.0	7169.958333	452.0	8.4	1.0	
4883	1.0	all_other	0.1148	69.24	10.714418	18.24	717.0	1830.000000	6554.0	55.1	2.0	
4884	1.0	all_other	0.1218	366.30	11.695247	9.38	707.0	11460.000000	9772.0	41.8	1.0	
4885	1.0	credit_card	0.0859	202.31	10.734220	15.24	747.0	8400.000000	13010.0	36.8	0.0	
4886	1.0	major_purchase	0.1148	659.37	10.534334	10.19	767.0	3959.958333	5110.0	36.5	0.0	
4887	1.0	credit_card	0.0859	271.85	11.461632	16.16	752.0	4050.000000	30330.0	30.9	2.0	
4888	1.0	small_business	0.1774	720.42	11.243319	7.04	702.0	5100.000000	11829.0	74.4	3.0	

In [29]:
```
loans.drop_duplicates(keep=False,inplace=True)
```

In [30]:
```
duplicateRowsLoans = loans[loans.duplicated()]
duplicateRowsLoans
```

Out[30]:

CreditPolicy	Purpose	InterestRate	Installment	AnnualIncomeLn	DebtToIncome	FICO.Score	DaysSinceSubscription	RevolvingBalance	DebtToLimitRatio	InquirySixMonths	DelayedPaymentsTwoY

Let us replace the missing values in "Purpose" column with the most frequent category, ie, mode of the column.

In [31]:
```
loans["Purpose"].value_counts()
```

Out[31]:
```
debt_consolidation    3955
all_other             2324
credit_card           1259
home_improvement       629
small_business         618
major_purchase         436
educational            342
Name: Purpose, dtype: int64
```

In [32]:
```
percentage_of_miss(loans)
```

Out[32]:

	Number of Missing	Percentage
Purpose	5	0.000523

In [33]:
```
def impute_nan_most_frequent_category(DataFrame,ColName):
    most_frequent_category=DataFrame[ColName].mode()[0]
    DataFrame[ColName].fillna(most_frequent_category,inplace=True)
```

In [34]:
```
impute_nan_most_frequent_category(loans, 'Purpose')
```

In [35]:
```
percentage_of_miss(loans)
```

Out[35]:

	Number of Missing	Percentage

In [36]:
```
loans["Purpose"].value_counts()
```

Out[36]:
```
debt_consolidation    3960
all_other             2324
credit_card           1259
home_improvement       629
small_business         618
major_purchase         436
educational            342
Name: Purpose, dtype: int64
```

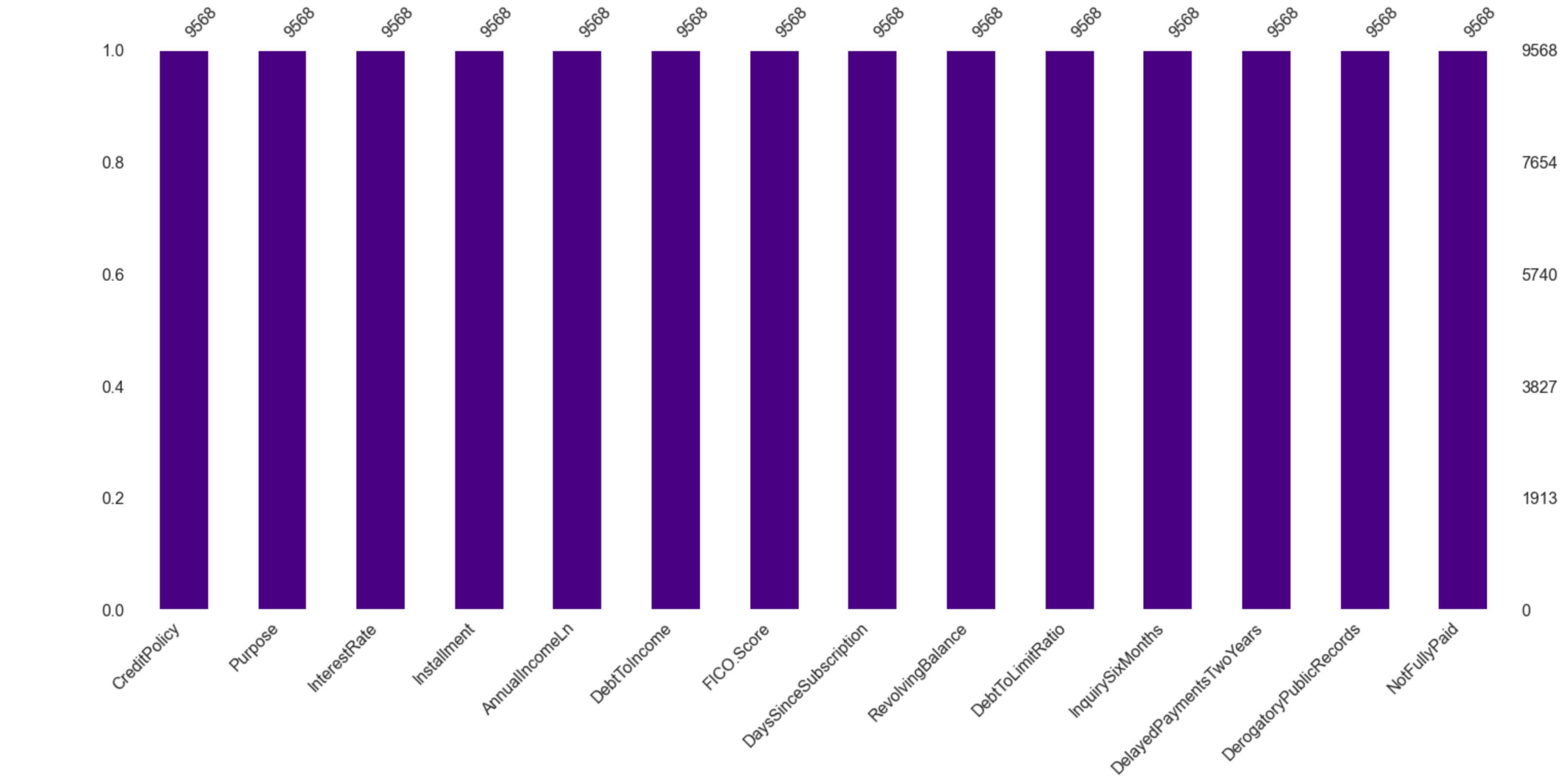
In [37]:
```
loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9568 entries, 0 to 9619
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   CreditPolicy              9568 non-null   float64
 1   Purpose                   9568 non-null   object 
 2   InterestRate              9568 non-null   float64
 3   Installment               9568 non-null   float64
 4   AnnualIncomeLn            9568 non-null   float64
 5   DebtToIncome              9568 non-null   float64
 6   FICO.Score                9568 non-null   float64
 7   DaysSinceSubscription     9568 non-null   float64
 8   RevolvingBalance          9568 non-null   float64
 9   DebtToLimitRatio          9568 non-null   float64
 10  InquirySixMonths          9568 non-null   float64
 11  DelayedPaymentsTwoYears   9568 non-null   float64
 12  DerogatoryPublicRecords   9568 non-null   float64
 13  NotFullyPaid              9568 non-null   float64
dtypes: float64(13), object(1)
memory usage: 1.1+ MB
```

Let us now visualize the missing values and we find that all the missing vlaues have been handled.

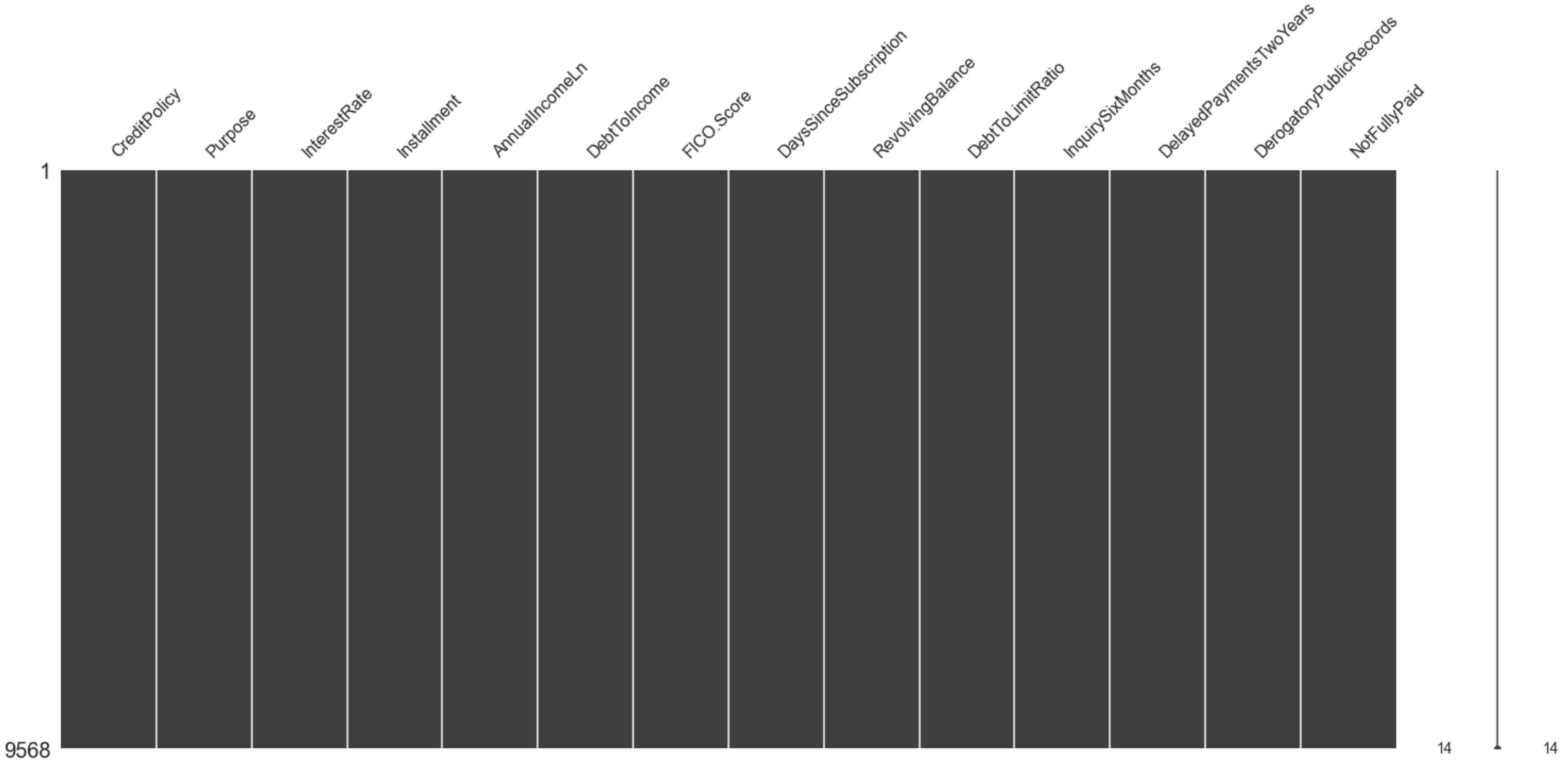
In [38]:
```
msno.bar(loans, color ='indigo')
```

Out[38]: <AxesSubplot:>



In [39]:
```
msno.matrix(loans)
```

Out[39]: <AxesSubplot:>



In [40]:
```
duplicateRowsLoans = loans[loans.duplicated()]
duplicateRowsLoans
```

Out[40]:

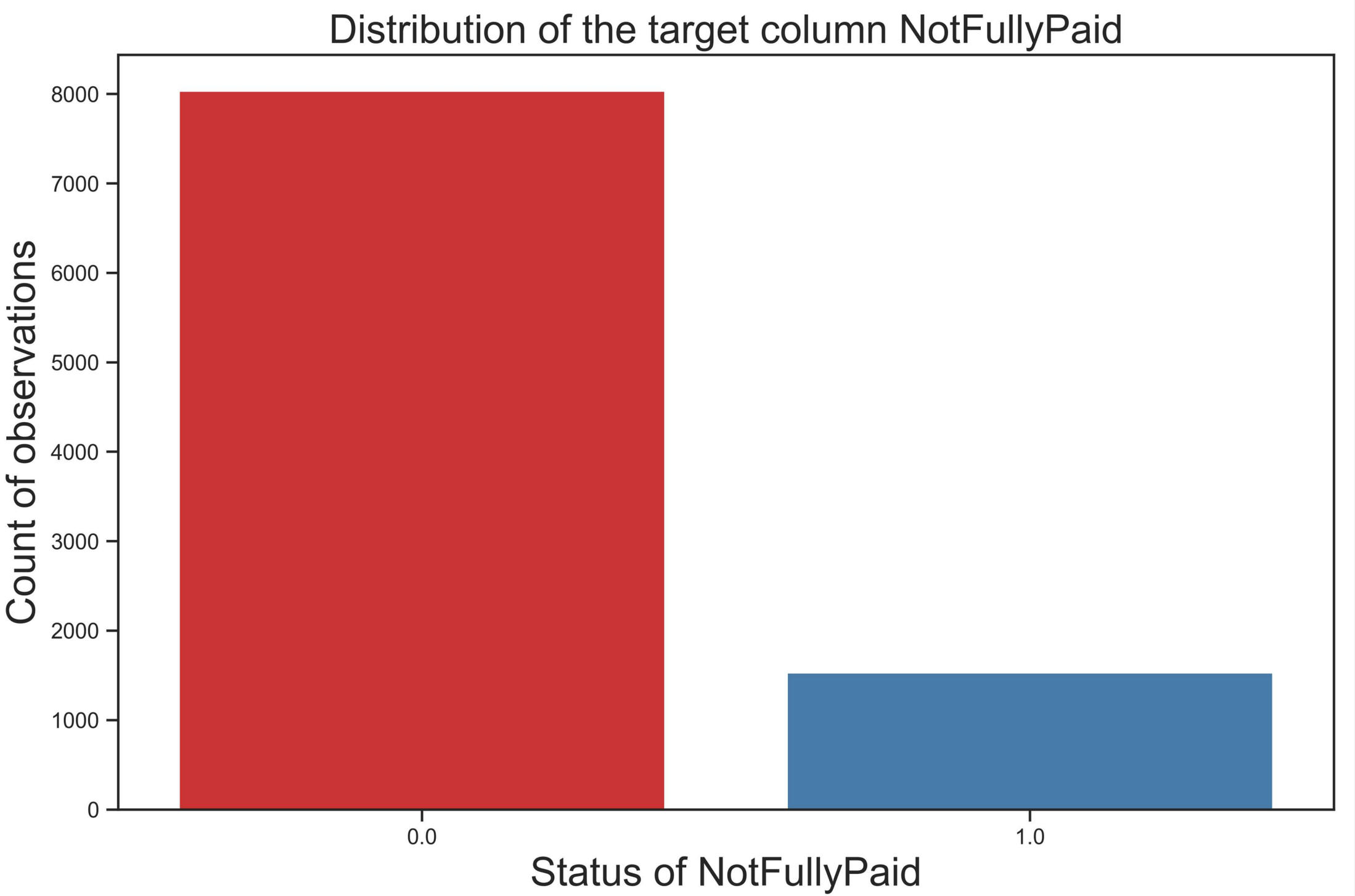
CreditPolicy	Purpose	InterestRate	Installment	AnnualIncomeLn	DebtToIncome	FICO.Score	DaysSinceSubscription	RevolvingBalance	DebtToLimitRatio	InquirySixMonths	DelayedPaymentsTw

Did we analyze the target column which was the NotFullyPaid columns in our dataframe?

In [41]:
```
NFP=loans['NotFullyPaid'].value_counts()
NFP
```

Out[41]:
```
0.0    8037
1.0    1531
Name: NotFullyPaid, dtype: int64
```

In [42]:
```
plt.figure(figsize=(11,7), dpi=600)
sns.countplot(x='NotFullyPaid',data=loans,palette='Set1')
plt.title("Distribution of the target column NotFullyPaid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Status of NotFullyPaid", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofthetargetcolumnNotFullyPaid.png",dpi=600)
```


Distribution of the target column NotFullyPaid
Count of observations
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
1.0
Status of NotFullyPaid

As we can see from the figure above we were dealing with unbalance data.

Did we analyze the CreditPolicy column of the dataframe?

In [43]:
```
CP=loans['CreditPolicy'].value_counts()
CP
```

Out[43]:
```
1.0    7700
0.0    1868
Name: CreditPolicy, dtype: int64
```

In [44]:
```
from scipy.stats import chi2_contingency

table1 = pd.crosstab(loans['CreditPolicy'],loans['NotFullyPaid'], margins=True)
table1
```

Out[44]:

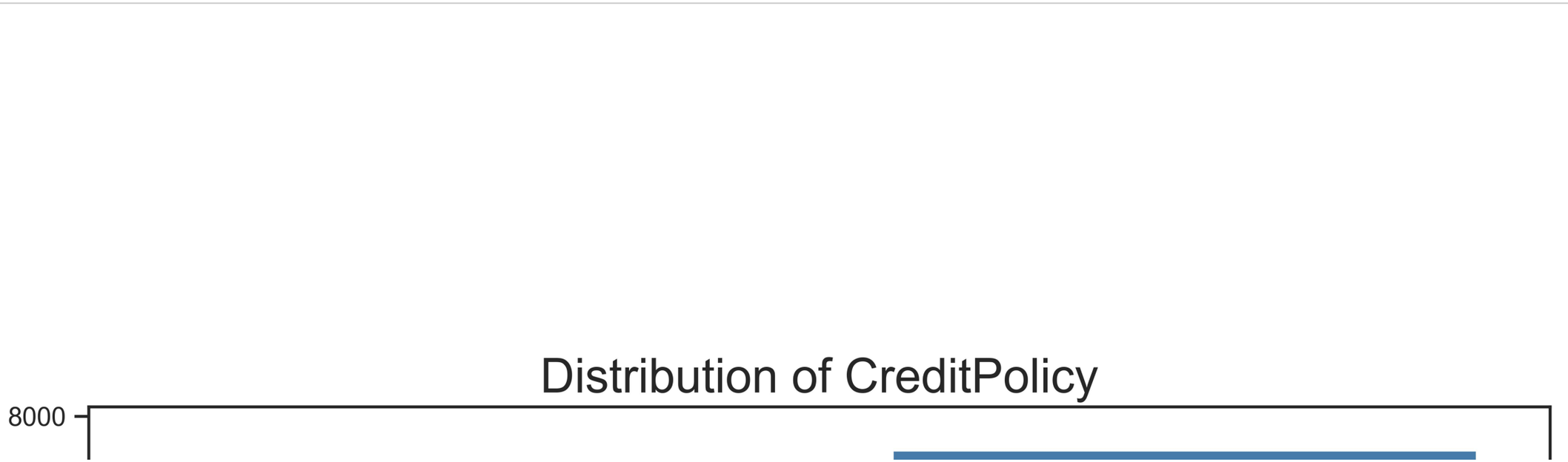
NotFullyPaid	0.0	1.0	All
CreditPolicy			
0.0	1349	519	1868
1.0	6688	1012	7700
All	8037	1531	9568

In [45]:
```
def chi_square(c1,c2):
    chi_2, p_val, dof, exp_val = chi2_contingency(pd.crosstab(loans[c1],loans[c2],margins = False))

    print(exp_val)
    print('\nChi-square is : %f'%chi_2, '\n\np_value is : %f'%p_val, '\n\ndegree of freedom is : %i'%dof)

    if p_val < 0.05:# consider significan level is 5%
        print("\nThere is some correlation between the two variables at significance level 0.05")
    else:
        print("\nThere is no correlation between the two variables")
```

In [46]:
```
chi_square('CreditPolicy','NotFullyPaid')
```

```
[[1569.09657191  298.90342809]
 [6467.90342809 1232.09657191]]

Chi-square is : 238.659022

p_value is : 0.000000

degree of freedom is : 1

There is some correlation between the two variables at significance level 0.05
```

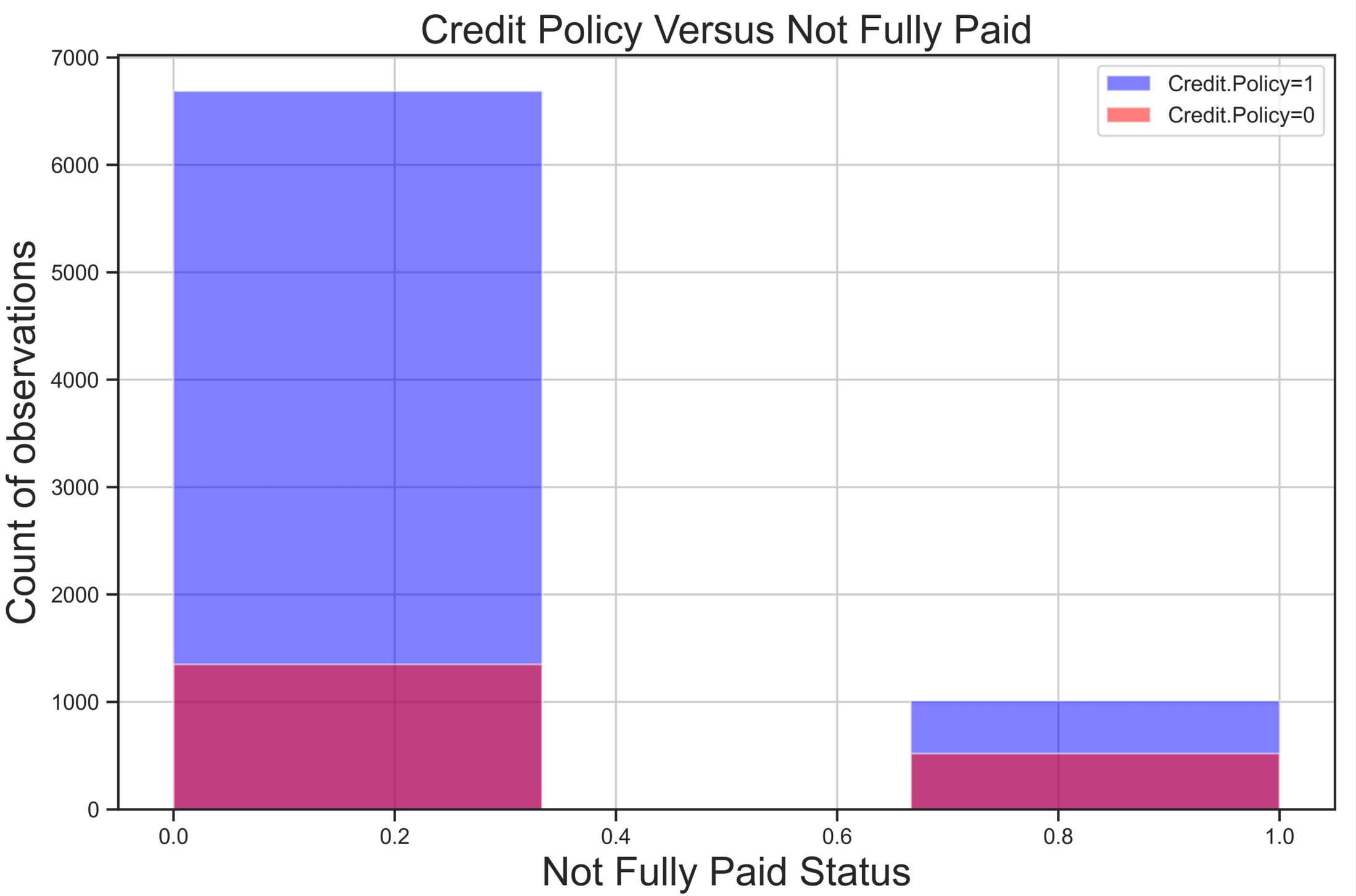
In [47]:
```
plt.figure(figsize=(11,7), dpi=600)
sns.countplot(x='CreditPolicy',data=loans,palette='Set1')
plt.title("Distribution of CreditPolicy", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Status of Credit Policy", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofCreditPolicy.png",dpi=600)
```



We can see that most of the borrowers meet the credit underwriting criteria of LendingClub.com.

In [48]:

```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['CreditPolicy']==1]['NotFullyPaid'].hist(alpha=0.5,color='blue',
                                              bins=3,label='Credit.Policy=1')
loans[loans['CreditPolicy']==0]['NotFullyPaid'].hist(alpha=0.5,color='red',
                                              bins=3,label='Credit.Policy=0')
plt.legend()
plt.title("Credit Policy Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Not Fully Paid Status", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofCreditPolicyVNotFullyPaid.png",dpi=600)
```


Credit Policy Versus Not Fully Paid
Credit.Policy=1
Credit.Policy=0
Count of observations
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
Not Fully Paid Status

Did we analyze the Purpose column of the dataframe?

In [49]:
```
PP=loans['Purpose'].value_counts()
PP
```

Out[49]:
```
debt_consolidation    3960
all_other             2324
credit_card           1259
home_improvement       629
small_business         618
major_purchase         436
educational            342
Name: Purpose, dtype: int64
```

In [50]:
```
table2 = pd.crosstab(loans['Purpose'],loans['NotFullyPaid'], margins=True)
table2
```

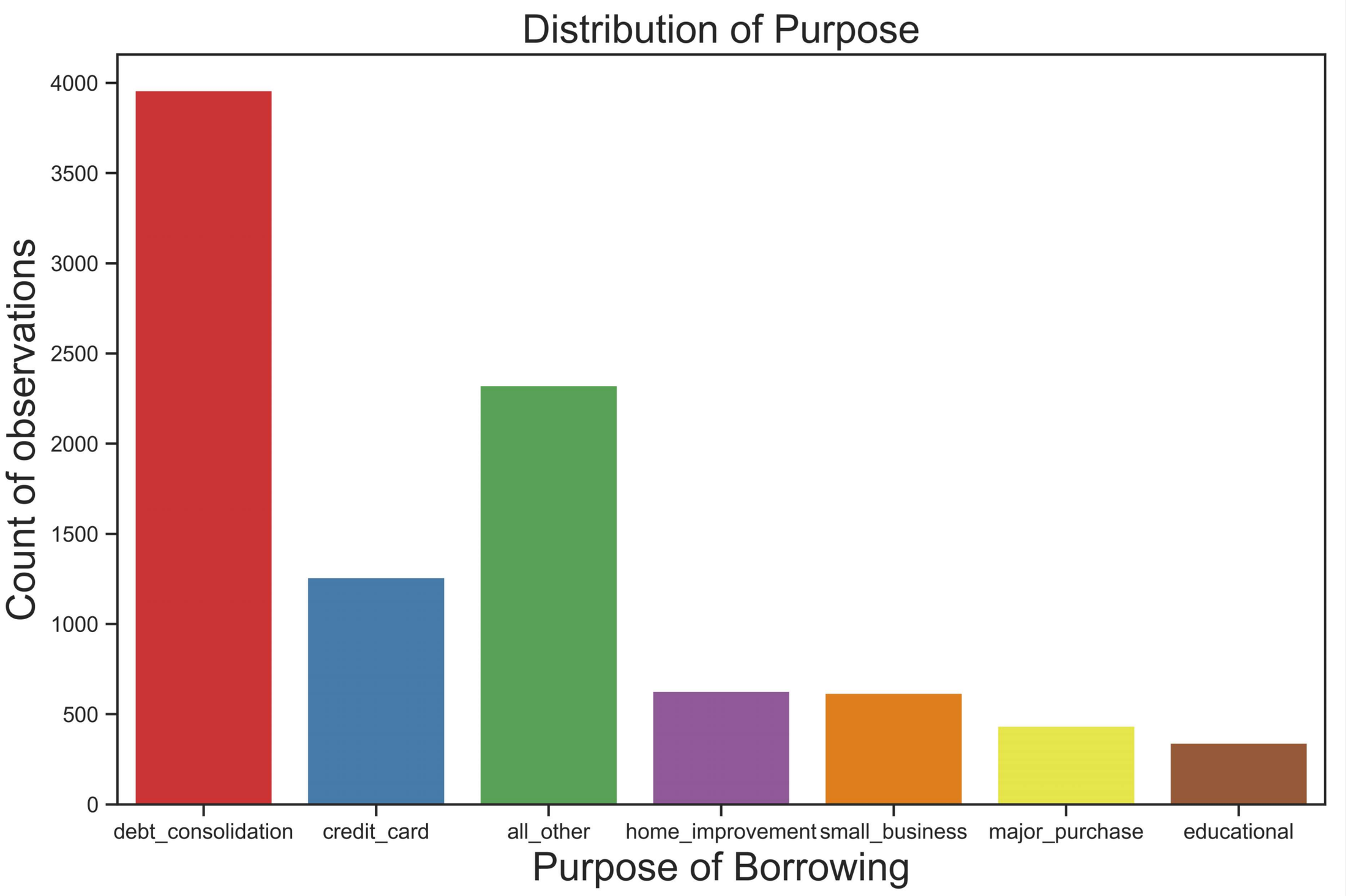
Out[50]:

NotFullyPaid	0.0	1.0	All
Purpose			
all_other	1938	386	2324
credit_card	1113	146	1259
debt_consolidation	3358	602	3960
educational	273	69	342
home_improvement	522	107	629
major_purchase	387	49	436
small_business	446	172	618
All	8037	1531	9568

In [51]:
```
chi_square('Purpose','NotFullyPaid')
```

```
[[1952.13085284  371.86914716]
 [1057.54420987  201.45579013]
 [3326.35033445  633.64966555]
 [ 287.2757107    54.7242893 ]
 [ 528.3521112   100.6478888 ]
 [ 366.23453177   69.76546823]
 [ 519.11224916   98.88775084]]

Chi-square is : 97.316184

p_value is : 0.000000

degree of freedom is : 6

There is some correlation between the two variables at significance level 0.05
```

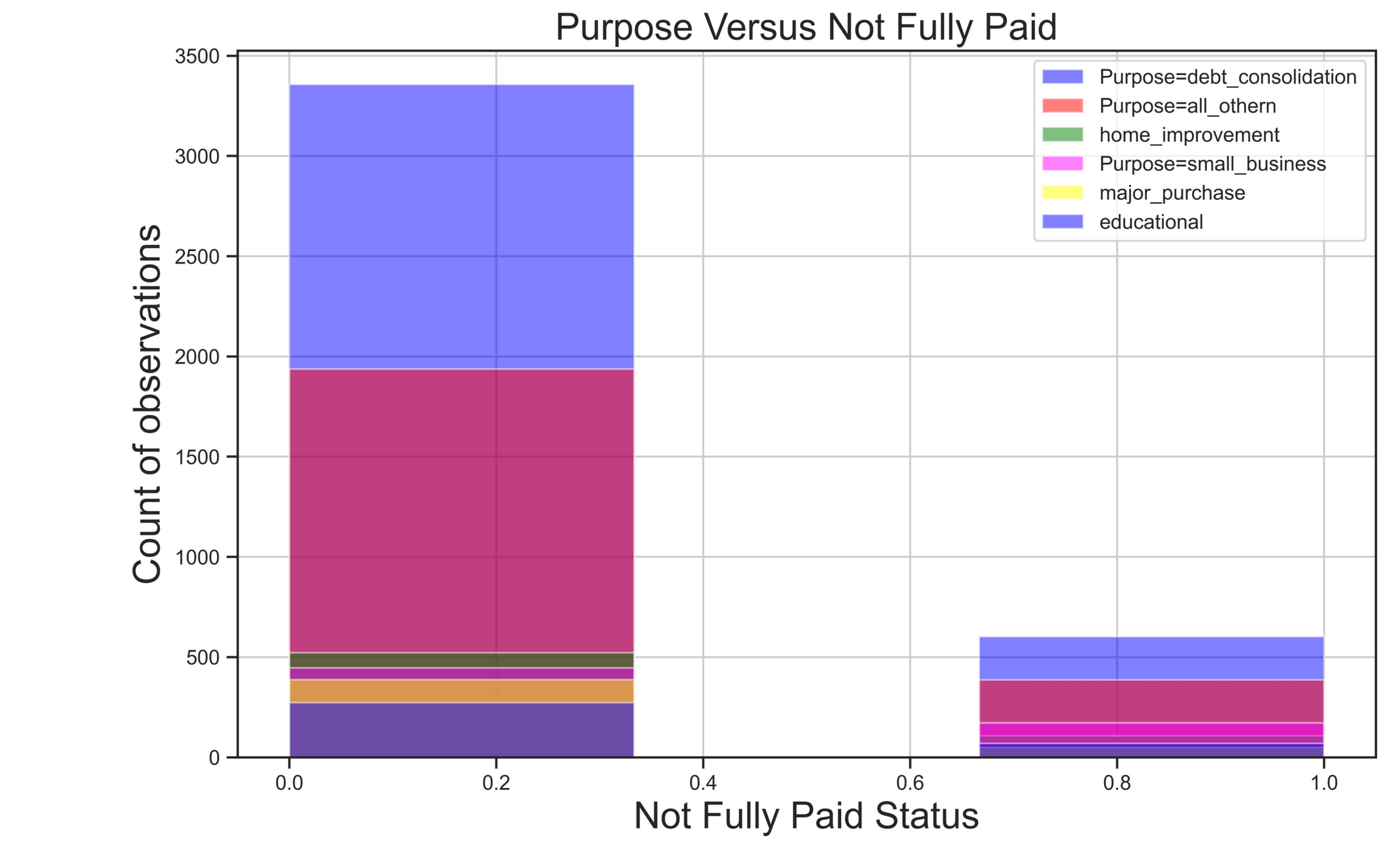
In [52]:
```
plt.figure(figsize=(11,7), dpi=600)
sns.countplot(x='Purpose',data=loans,palette='Set1')
plt.title("Distribution of Purpose", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Purpose of Borrowing", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofStatusofPurpose.png",dpi=600)
```


Distribution of Purpose
Count of observations
4000
3500
3000
2500
2000
1500
1000
500
0
debt_consolidation
credit_card
all_other
home_improvement
small_business
major_purchase
educational
Purpose of Borrowing

It is clear that debt consolidation was by far the main reason of borrowing.

In [53]:

```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['Purpose']=='debt_consolidation']['NotFullyPaid'].hist(alpha=0.5,color='blue',
                                            bins=3,label='Purpose=debt_consolidation')
loans[loans['Purpose']=='all_other']['NotFullyPaid'].hist(alpha=0.5,color='red',
                                            bins=3,label='Purpose=all_othern')
loans[loans['Purpose']=='home_improvement']['NotFullyPaid'].hist(alpha=0.5,color='green',
                                            bins=3,label='home_improvement')
loans[loans['Purpose']=='small_business']['NotFullyPaid'].hist(alpha=0.5,color='magenta',
                                            bins=3,label='Purpose=small_business')
loans[loans['Purpose']=='major_purchase']['NotFullyPaid'].hist(alpha=0.5,color='yellow',
                                            bins=3,label='major_purchase')
loans[loans['Purpose']=='educational']['NotFullyPaid'].hist(alpha=0.5,color='blue',
                                            bins=3,label='educational')
plt.legend()
plt.title("Purpose Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Not Fully Paid Status", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofPurposeVNotFullyPaid.png",dpi=600)
```


Purpose Versus Not Fully Paid
Count of observations
Not Fully Paid Status
Purpose=debt_consolidation
Purpose=all_othern
home_improvement
Purpose=small_business
major_purchase
educational
3500
3000
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0

Did we analyze the InterestRate column of the dataframe?

In [54]:
```
IR=loans['InterestRate'].describe()
IR
```

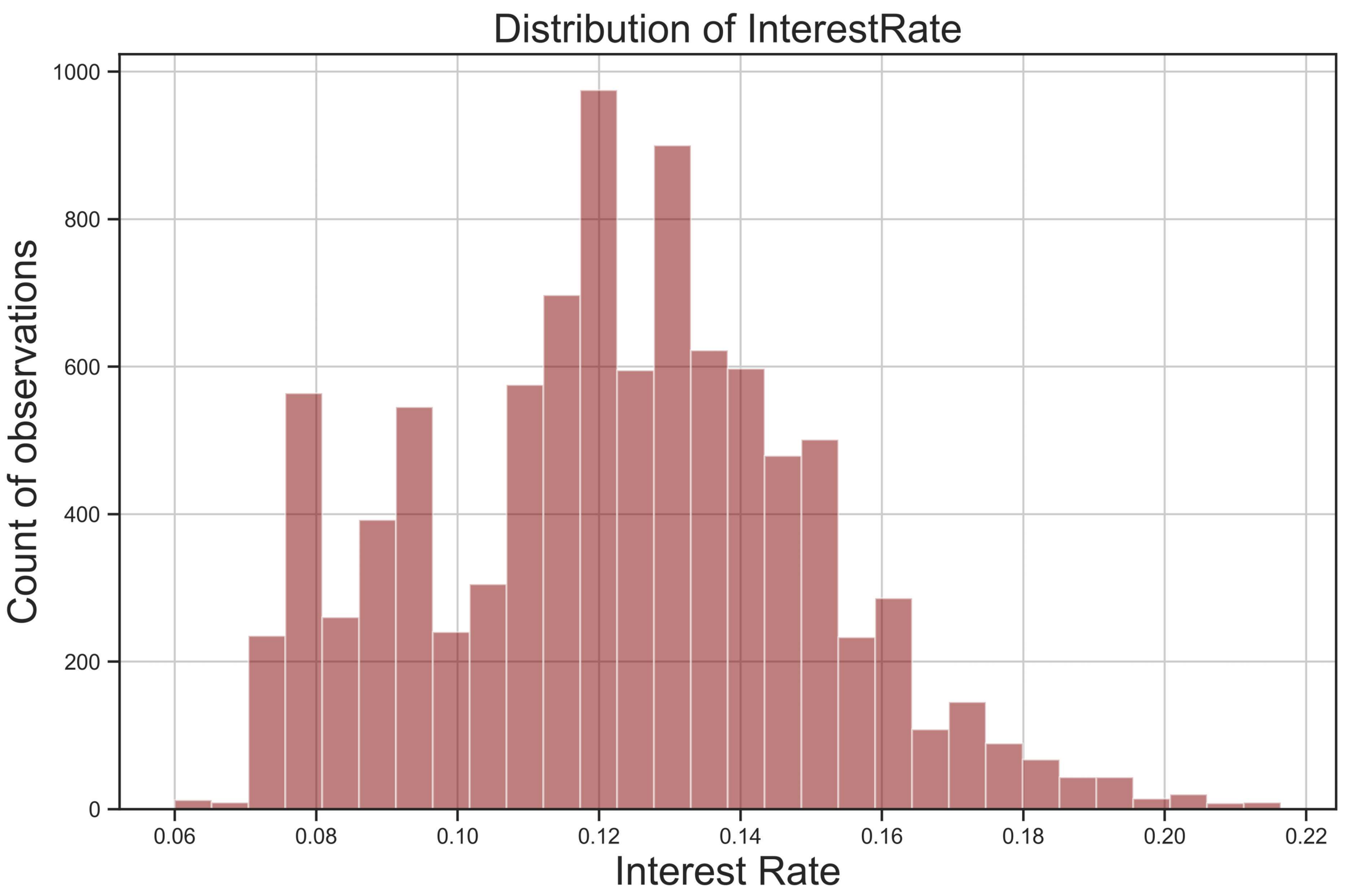
Out[54]:
```
count    9568.000000
mean        0.122644
std         0.026850
min         0.060000
25%         0.103900
50%         0.122100
75%         0.140700
max         0.216400
Name: InterestRate, dtype: float64
```

In [55]:
```
from scipy.stats import ttest_ind
ttest_ind(loans['InterestRate'],loans['NotFullyPaid'])
```

Out[55]:
```
Ttest_indResult(statistic=-9.94291736213115, pvalue=3.083166954029472e-23)
```

There is some correlation between the two variables at significance level 0.05

In [56]:
```
plt.figure(figsize=(11,7), dpi=600)
loans['InterestRate'].hist(alpha=0.5,color='maroon',
                                            bins=30)
plt.title("Distribution of InterestRate", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Interest Rate", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofInterestRate.png",dpi=600)
```


Distribution of InterestRate
Count of observations
1000
800
600
400
200
0
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22
Interest Rate

In [57]:

```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['InterestRate'].hist(alpha=0.5,color='blue',
                                            bins=15,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['InterestRate'].hist(alpha=0.5,color='red',
                                            bins=15,label='NotFullyPaid = 0')
plt.legend()
plt.title("Interest Rate Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Interest Rate", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofInterestVNotFullyPaid.png",dpi=600)
```


Interest Rate Versus Not Fully Paid
NotFullyPaid = 1
NotFullyPaid = 0
Count of observations
0
200
400
600
800
1000
1200
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
0.22
Interest Rate

Did we analyze the Installment column of the dataframe?

In [58]:
```
IST=loans['Installment'].describe()
IST
```

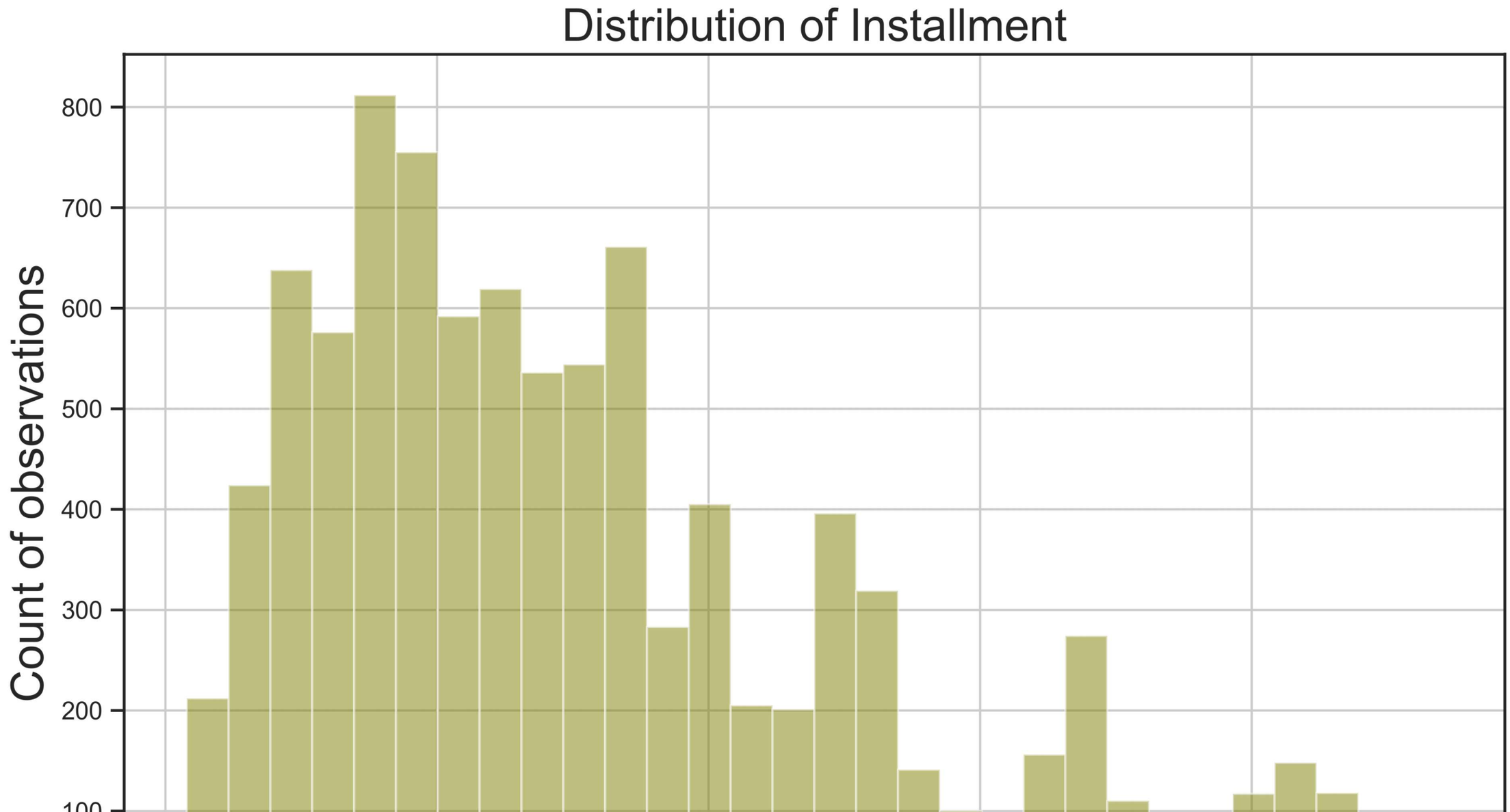
Out[58]:
```
count    9568.000000
mean      319.021305
std       207.052703
min        15.670000
25%       163.770000
50%       268.950000
75%       432.282500
max       940.140000
Name: Installment, dtype: float64
```

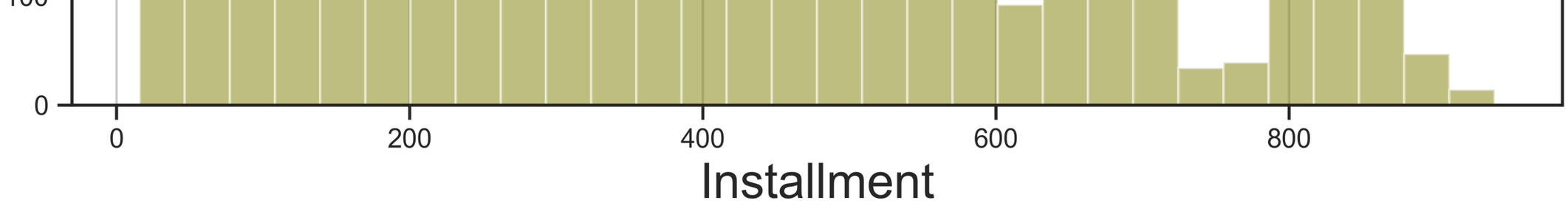
In [59]:
```
ttest_ind(loans['Installment'],loans['NotFullyPaid'])
```

Out[59]:
```
Ttest_indResult(statistic=150.63670239322266, pvalue=0.0)
```

There is some correlation between the two variables at significance level 0.05

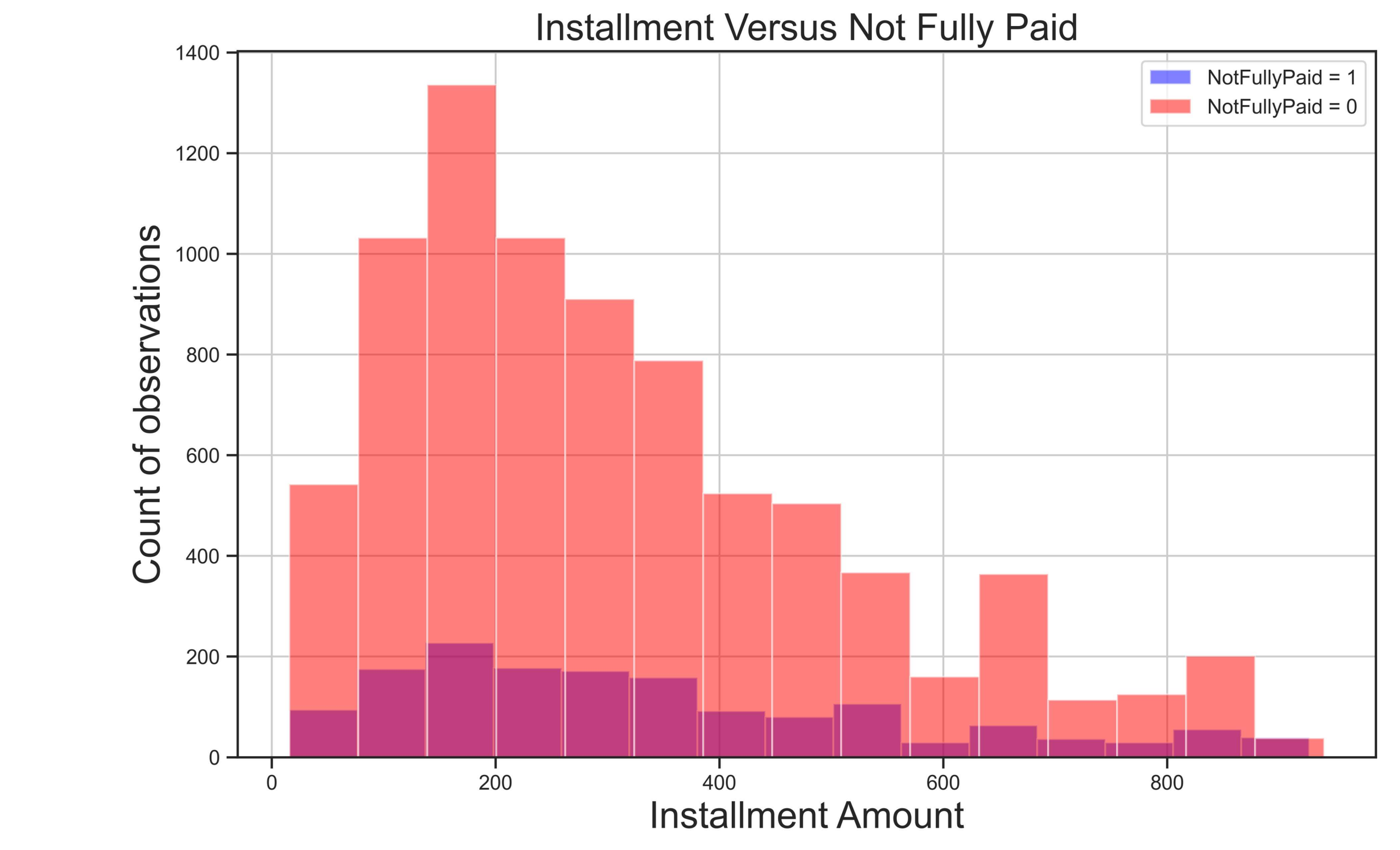
In [60]:
```
plt.figure(figsize=(11,7), dpi=600)
loans['Installment'].hist(alpha=0.5,color='olive',
                                          bins=30)
plt.title("Distribution of Installment", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Installment", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofInstallment.png",dpi=600)
```




100
0
0
200
400
600
800
Installment

In [61]:

```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['Installment'].hist(alpha=0.5,color='blue',
                                              bins=15,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['Installment'].hist(alpha=0.5,color='red',
                                              bins=15,label='NotFullyPaid = 0')
plt.legend()
plt.title("Installment Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Installment Amount", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofInstallmentVNotFullyPaid.png",dpi=600)
```

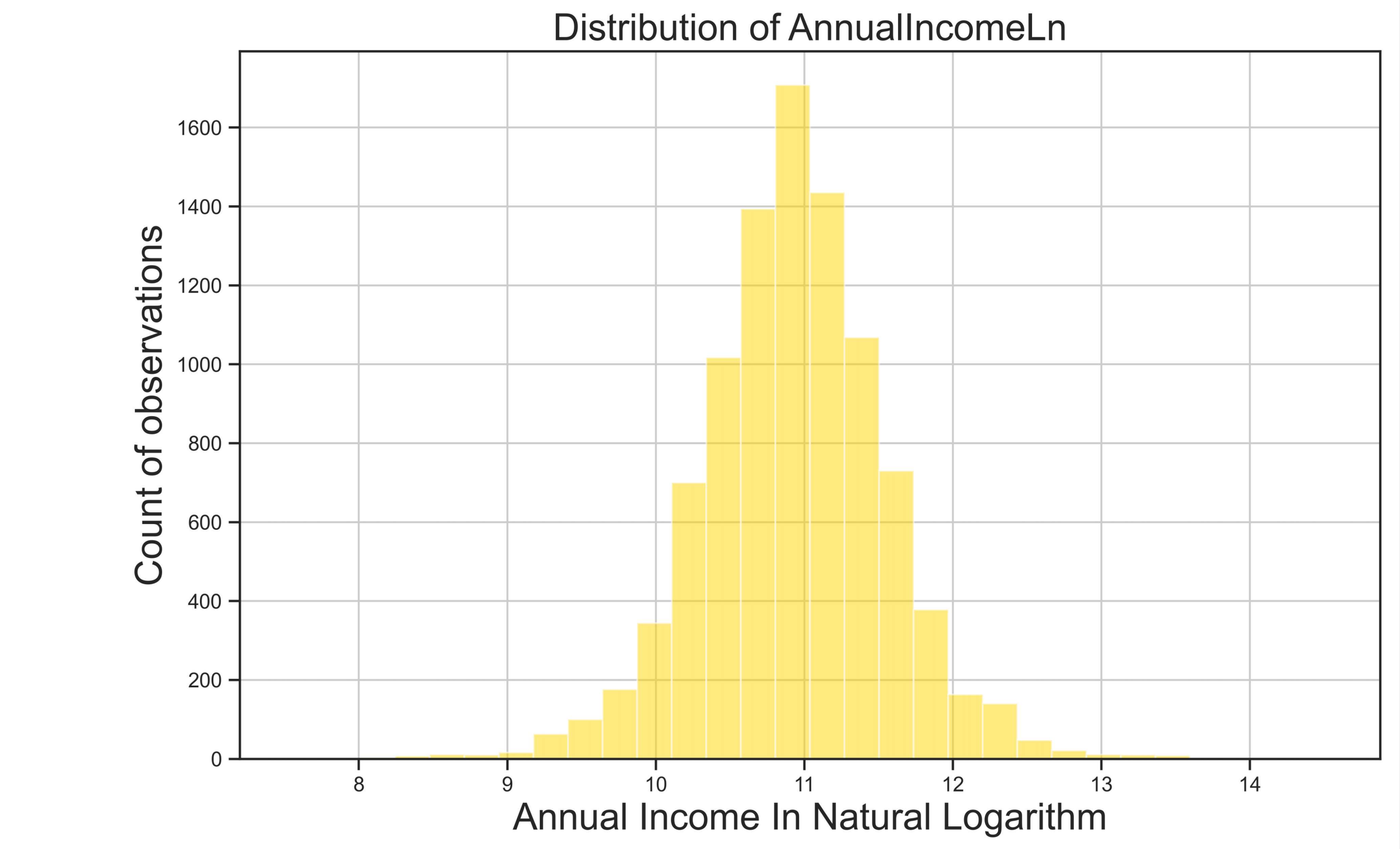

Installment Versus Not Fully Paid
NotFullyPaid = 1
NotFullyPaid = 0
Count of observations
Installment Amount
0
200
400
600
800
1000
1200
1400

Did we analyze AnnualIncomeLn column of the dataframe?

In [62]:
```
AIL=loans['AnnualIncomeLn'].describe()
AIL
```

Out[62]:
```
count    9568.000000
mean       10.931854
std         0.614912
min         7.547502
25%        10.558414
50%        10.927987
75%        11.289832
max        14.528354
Name: AnnualIncomeLn, dtype: float64
```

In [63]:
```
plt.figure(figsize=(11,7), dpi=600)
loans['AnnualIncomeLn'].hist(alpha=0.5,color='gold',
                                          bins=30)
plt.title("Distribution of AnnualIncomeLn", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Annual Income In Natural Logarithm", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofAnnualIncomeLn.png",dpi=600)
```


Distribution of AnnualIncomeLn
Count of observations
0
200
400
600
800
1000
1200
1400
1600
8
9
10
11
12
13
14
Annual Income In Natural Logarithm

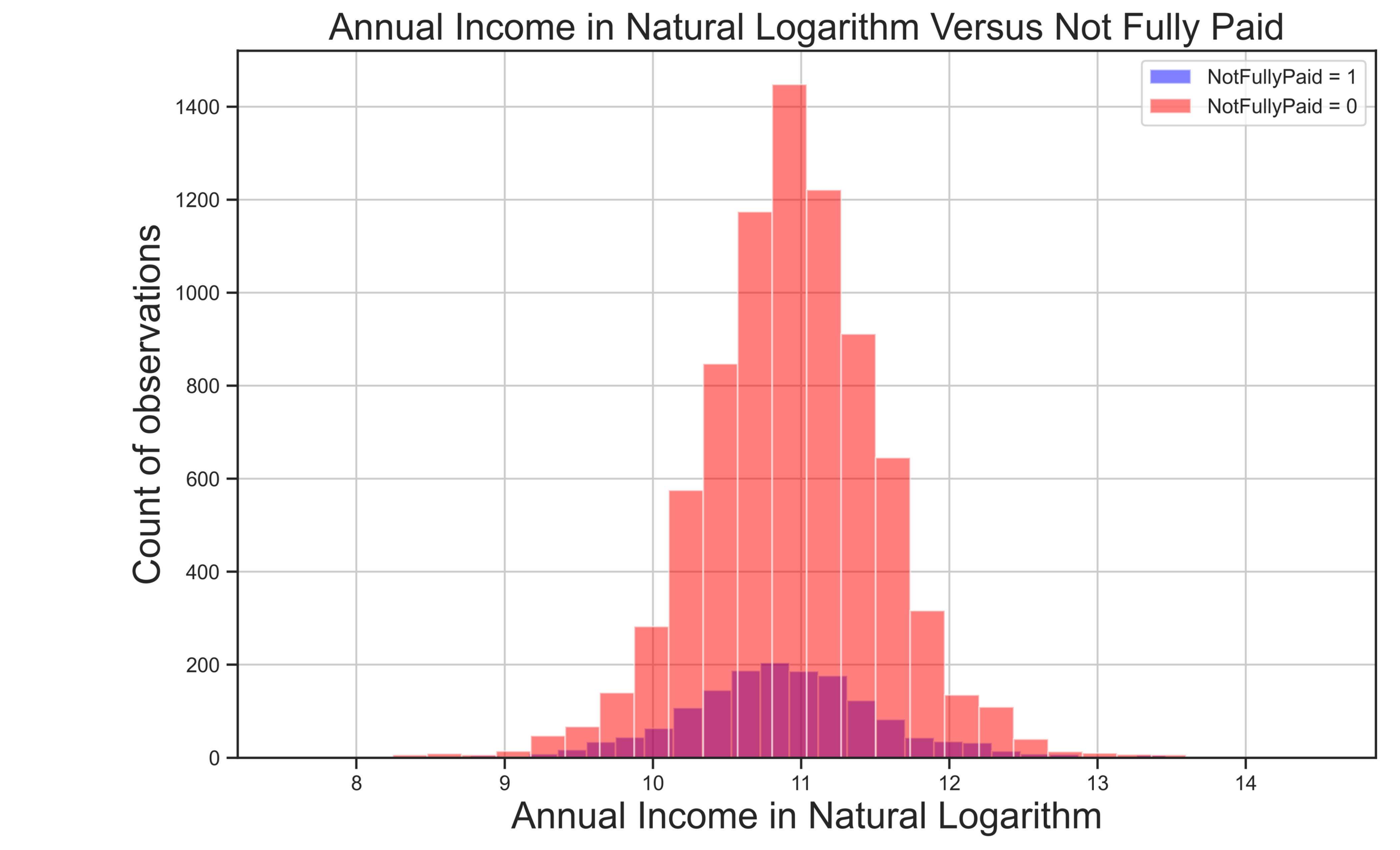
In [64]:
```
ttest_ind(loans['AnnualIncomeLn'],loans['NotFullyPaid'])
```

Out[64]:
```
Ttest_indResult(statistic=1471.7593957259485, pvalue=0.0)
```

There is some correlation between the two variables at significance level 0.05

In [65]:

```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['AnnualIncomeLn'].hist(alpha=0.5,color='blue',
                                            bins=30,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['AnnualIncomeLn'].hist(alpha=0.5,color='red',
                                            bins=30,label='NotFullyPaid = 0')
plt.legend()
plt.title("Annual Income in Natural Logarithm Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Annual Income in Natural Logarithm", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofAnnualIncomeLnVNotFullyPaid.png",dpi=600)
```


Annual Income in Natural Logarithm Versus Not Fully Paid
NotFullyPaid = 1
NotFullyPaid = 0
Count of observations
0
200
400
600
800
1000
1200
1400
8
9
10
11
12
13
14
Annual Income in Natural Logarithm

Did we analyze DebtToIncome column of the dataframe?

In [66]:
```
DTI=loans['DebtToIncome'].describe()
DTI
```

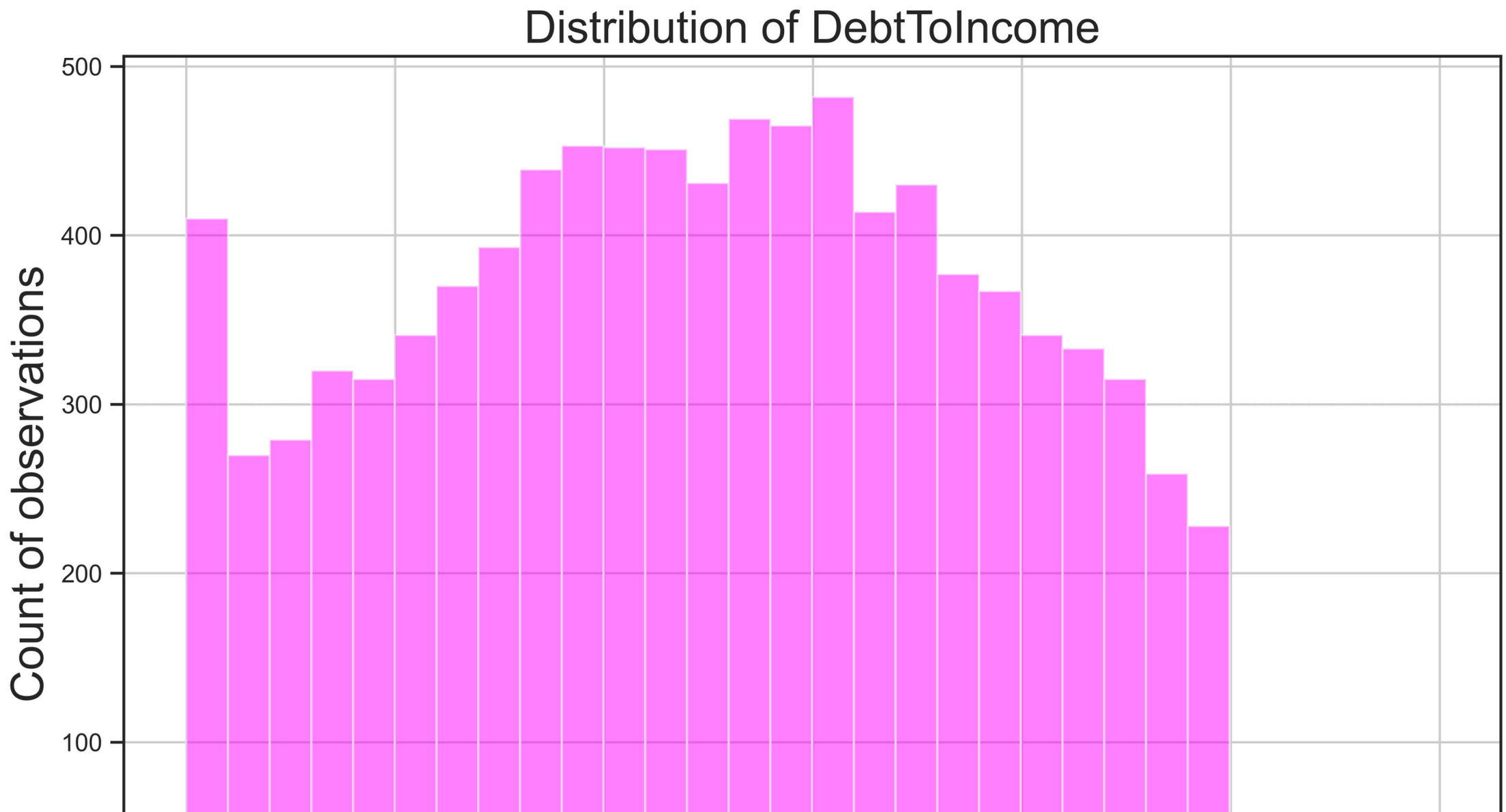
Out[66]:
```
count    9568.000000
mean       12.604398
std         6.885845
min         0.000000
25%         7.207500
50%        12.660000
75%        17.950000
max        29.960000
Name: DebtToIncome, dtype: float64
```

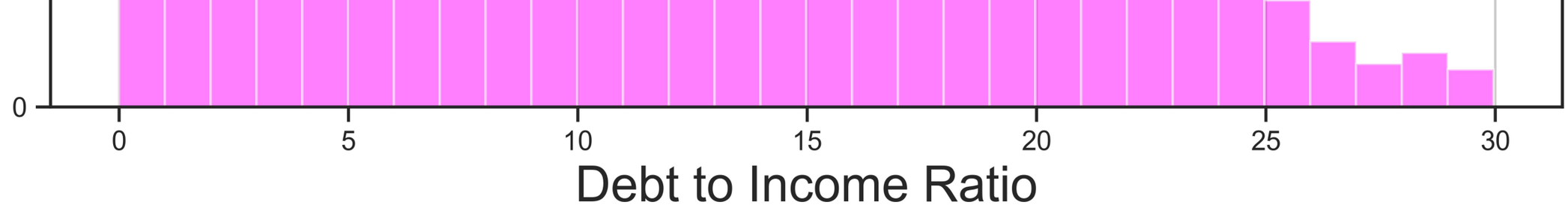
In [67]:
```
ttest_ind(loans['DebtToIncome'],loans['NotFullyPaid'])
```

Out[67]:
```
Ttest_indResult(statistic=176.5273597101466, pvalue=0.0)
```

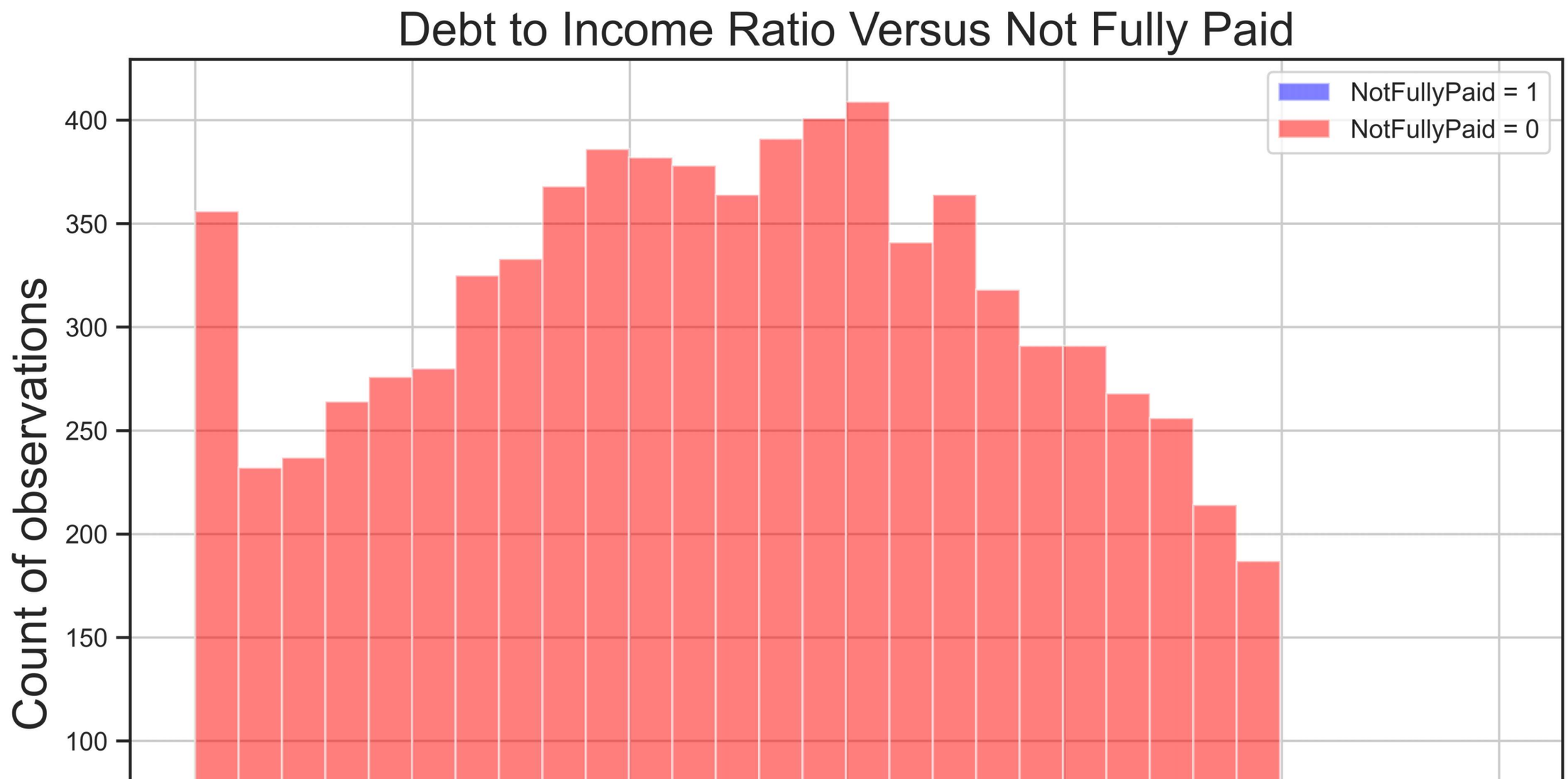
There is some correlation between the two variables at significance level 0.05

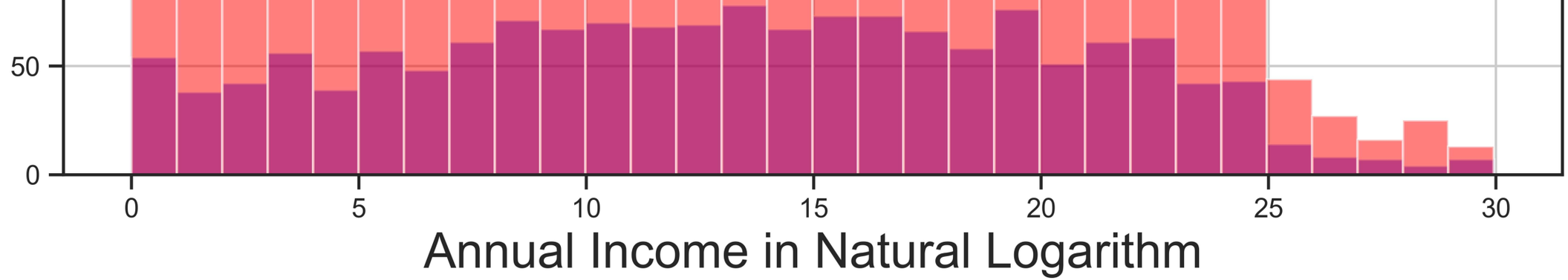
In [69]:
```
plt.figure(figsize=(11,7), dpi=600)
loans['DebtToIncome'].hist(alpha=0.5,color='magenta',
                                        bins=30)
plt.title("Distribution of DebtToIncome", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Debt to Income Ratio", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDebtToIncome.png",dpi=600)
```




0
0
5
10
15
20
25
30
Debt to Income Ratio

In [70]:
```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['DebtToIncome'].hist(alpha=0.5,color='blue',
                                            bins=30,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['DebtToIncome'].hist(alpha=0.5,color='red',
                                            bins=30,label='NotFullyPaid = 0')
plt.legend()
plt.title("Debt to Income Ratio Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Annual Income in Natural Logarithm", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDebtToIncomeVNotFullyPaid.png",dpi=600)
```





Did we analyze FICO.Score column of the dataframe?

In [71]:
```
FCS=loans['FICO.Score'].describe()
FCS
```

Out[71]:
```
count    9568.000000
mean      710.832044
std        37.981188
min       612.000000
25%       682.000000
50%       707.000000
75%       737.000000
max       827.000000
Name: FICO.Score, dtype: float64
```

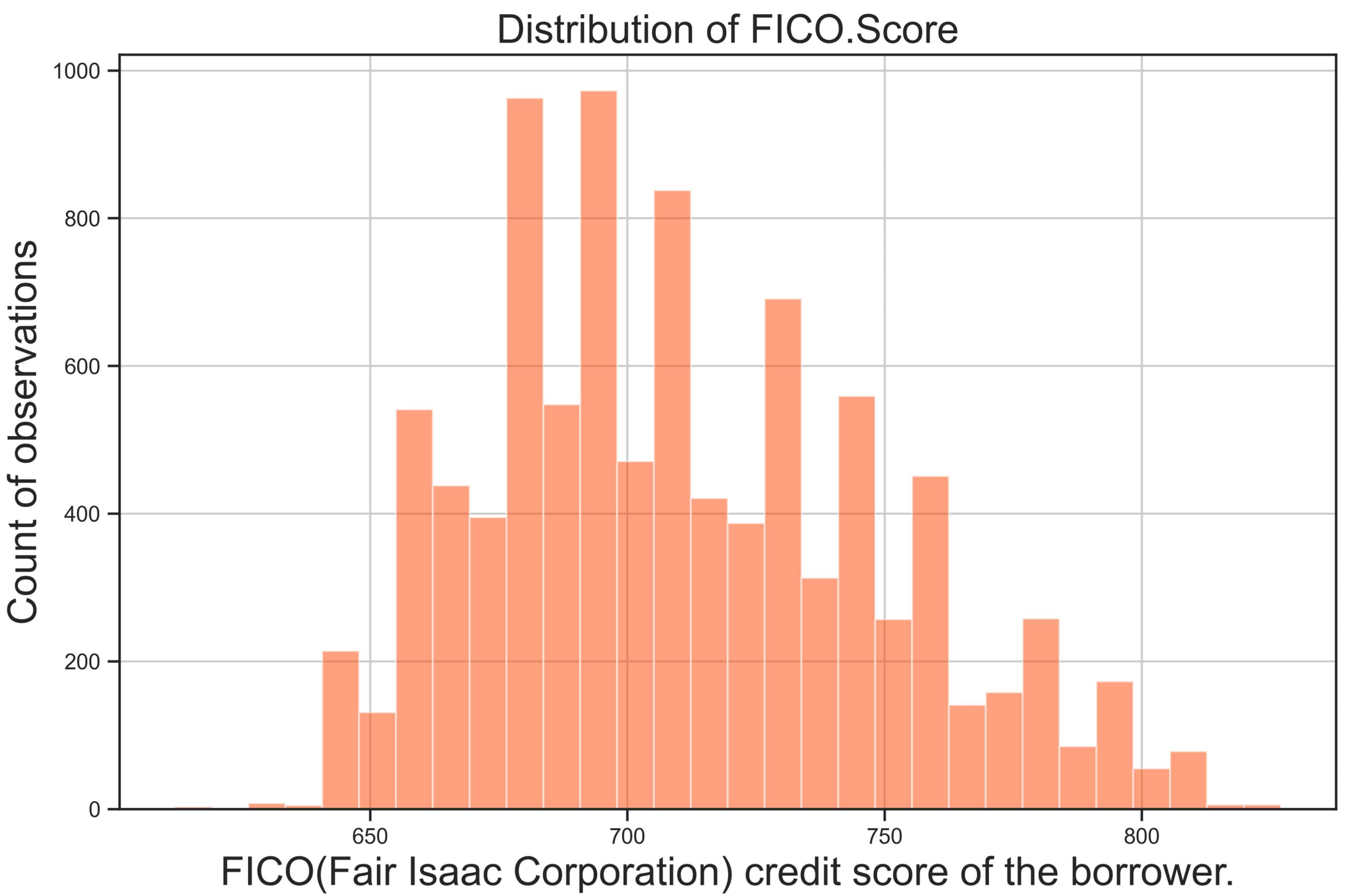
In [72]:
```
ttest_ind(loans['FICO.Score'],loans['NotFullyPaid'])
```

Out[72]:
```
Ttest_indResult(statistic=1830.1683185318614, pvalue=0.0)
```

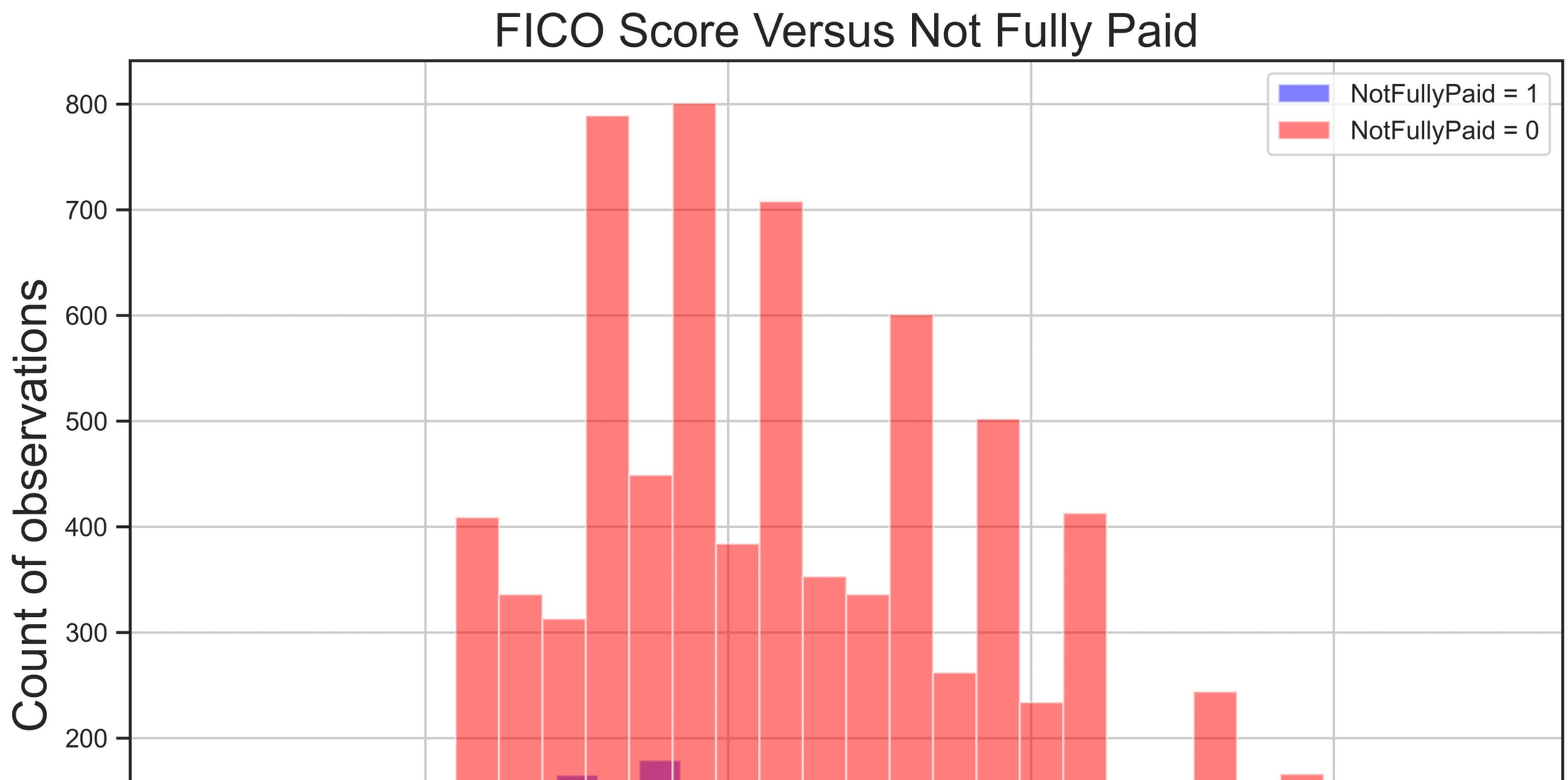
There is some correlation between the two variables at significance level 0.05

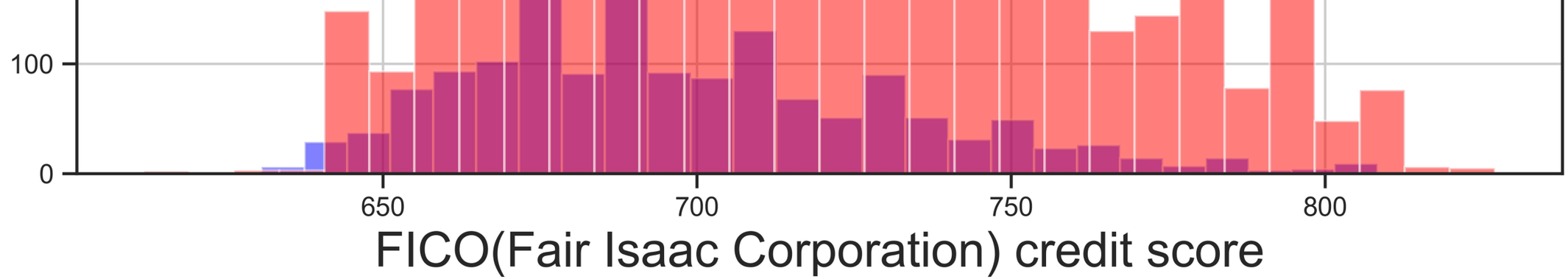
In [73]:

```
plt.figure(figsize=(11,7), dpi=600)
loans['FICO.Score'].hist(alpha=0.5,color='#FF4500',
                                        bins=30)
plt.title("Distribution of FICO.Score", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("FICO(Fair Isaac Corporation) credit score of the borrower.", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofFICOSCORE.png",dpi=600)
```


Distribution of FICO.Score
Count of observations
1000
800
600
400
200
0
650
700
750
800
FICO(Fair Isaac Corporation) credit score of the borrower.

In [74]:
```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['FICO.Score'].hist(alpha=0.5,color='blue',
                                              bins=30,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['FICO.Score'].hist(alpha=0.5,color='red',
                                              bins=30,label='NotFullyPaid = 0')
plt.legend()
plt.title("FICO Score Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("FICO(Fair Isaac Corporation) credit score", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofFICOScoreVNotFullyPaid.png",dpi=600)
```





Did we analyze DaysSinceSubscription column of the dataframe?

In [75]:
```
DSS=loans['DaysSinceSubscription'].describe()
DSS
```

Out[75]:
```
count     9568.000000
mean      4559.426038
std       2496.307691
min        178.958333
25%       2819.958333
50%       4139.958333
75%       5730.000000
max      17639.958330
Name: DaysSinceSubscription, dtype: float64
```

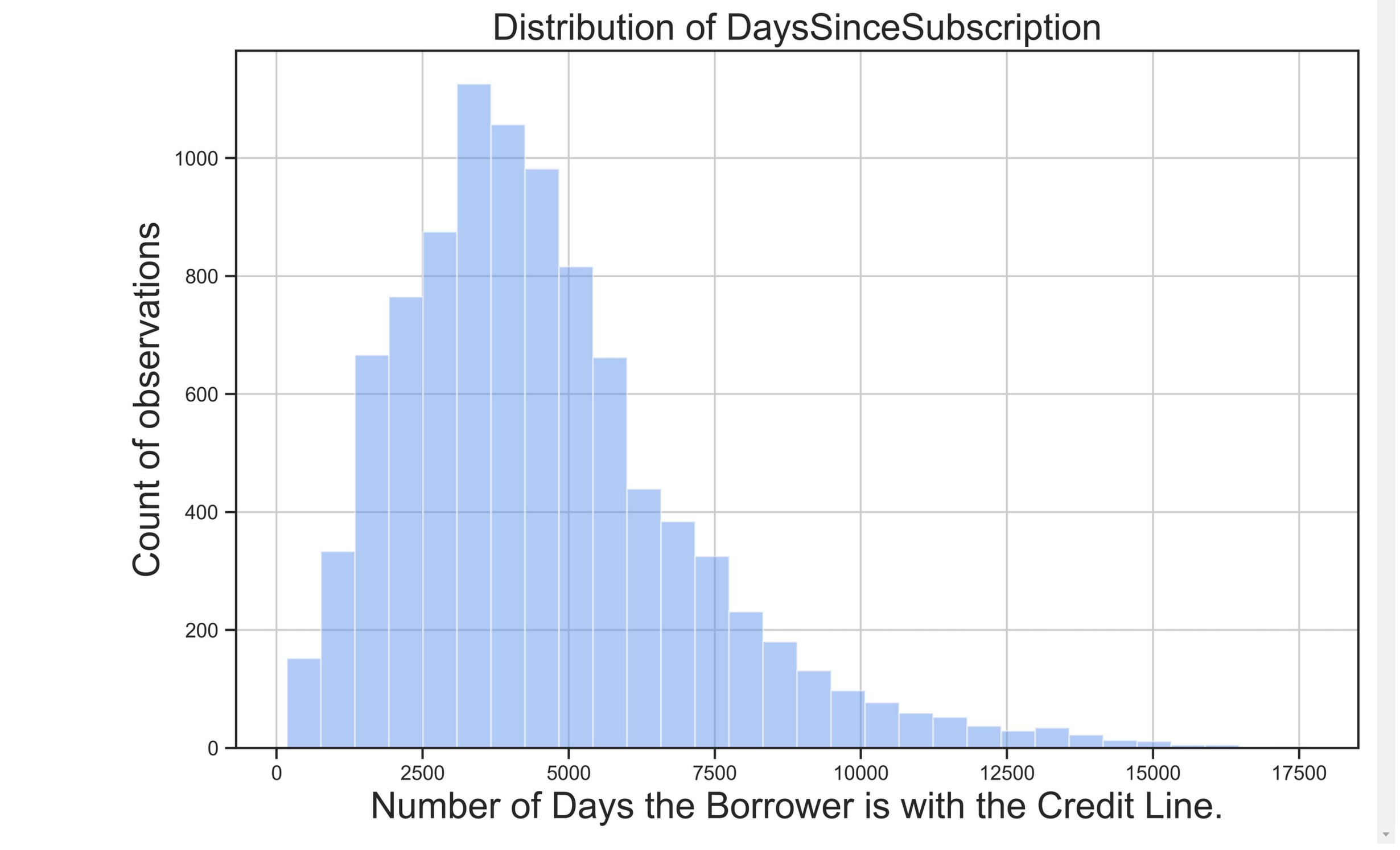
In [76]:
```
ttest_ind(loans['DaysSinceSubscription'],loans['NotFullyPaid'])
```

Out[76]:
```
Ttest_indResult(statistic=178.65180036867943, pvalue=0.0)
```

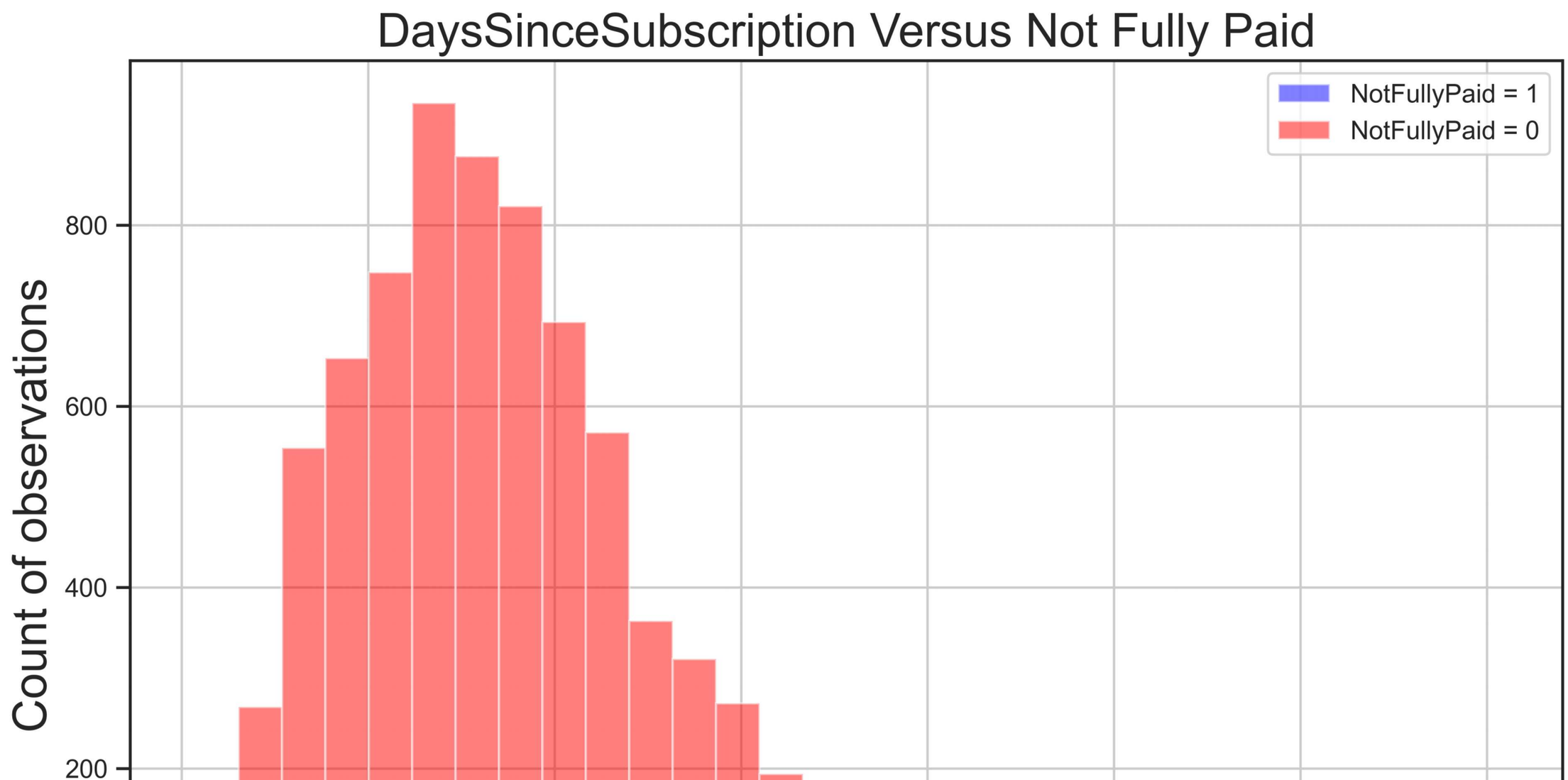
There is some correlation between the two variables at significance level 0.05

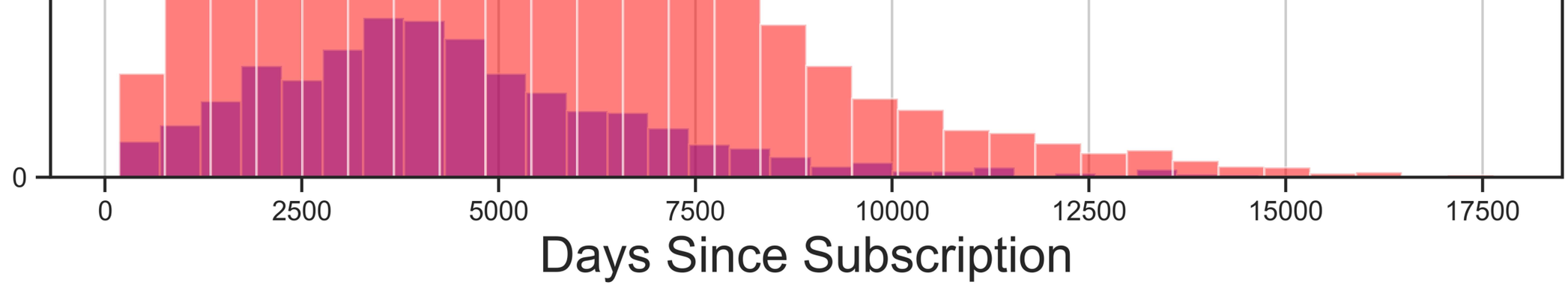
In [77]:

```
plt.figure(figsize=(11,7), dpi=600)
loans['DaysSinceSubscription'].hist(alpha=0.5,color='#6495ED',
                                    bins=30)
plt.title("Distribution of DaysSinceSubscription", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Number of Days the Borrower is with the Credit Line.", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDaysSinceSubscription.png",dpi=600)
```


Distribution of DaysSinceSubscription
Count of observations
0
200
400
600
800
1000
0
2500
5000
7500
10000
12500
15000
17500
Number of Days the Borrower is with the Credit Line.

In [78]:
```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['DaysSinceSubscription'].hist(alpha=0.5,color='blue',
                                            bins=30,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['DaysSinceSubscription'].hist(alpha=0.5,color='red',
                                            bins=30,label='NotFullyPaid = 0')
plt.legend()
plt.title("DaysSinceSubscription Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Days Since Subscription", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDaysSinceSubscriptionVNotFullyPaid.png",dpi=600)
```





Did we analyze RevolvingBalance column of the dataframe?

In [79]:
```
RVB=loans['RevolvingBalance'].describe()
RVB
```

Out[79]:
```
count    9.568000e+03
mean     1.691402e+04
std      3.377049e+04
min      0.000000e+00
25%      3.184750e+03
50%      8.593000e+03
75%      1.824450e+04
max      1.207359e+06
Name: RevolvingBalance, dtype: float64
```

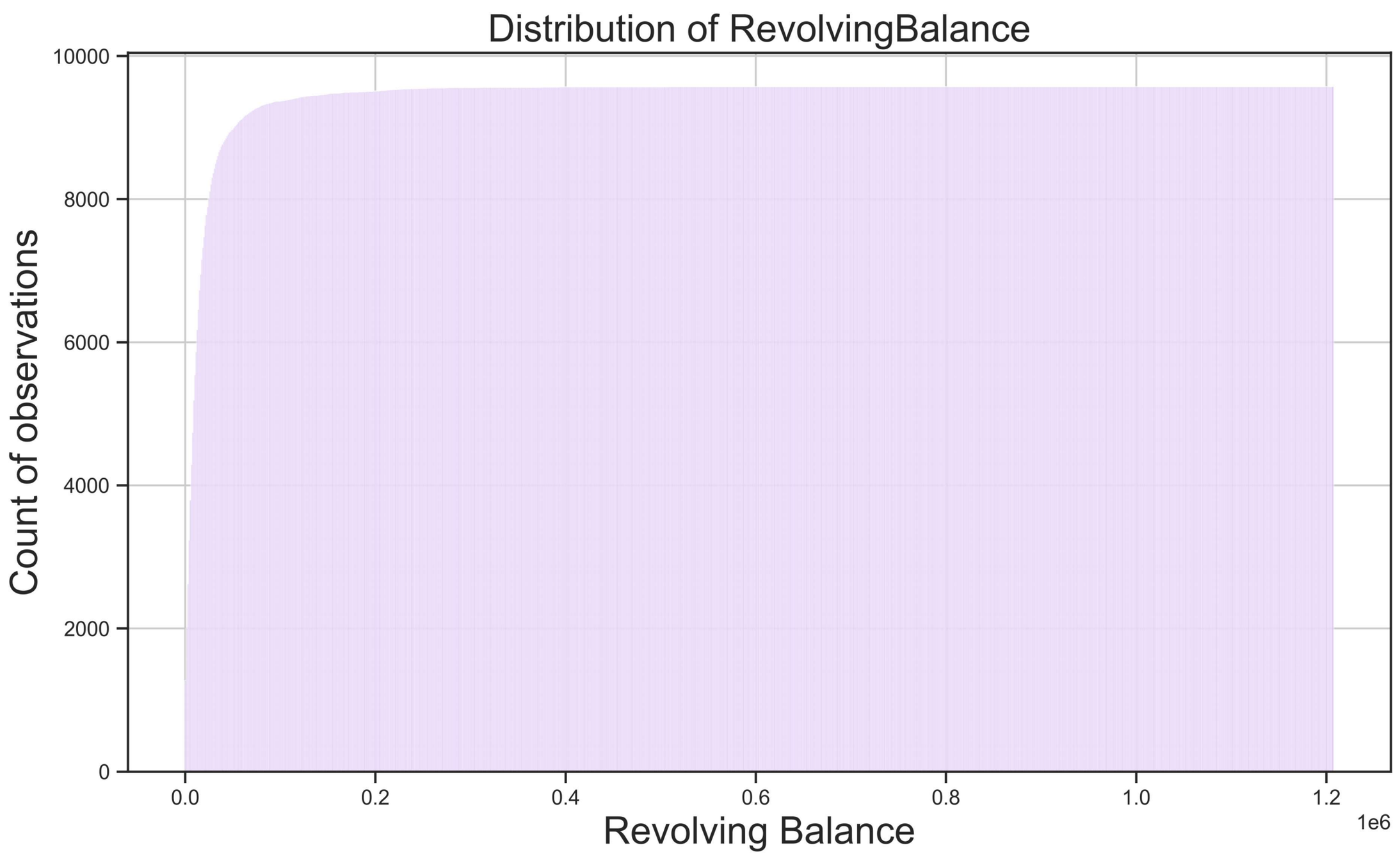
In [80]:
```
ttest_ind(loans['RevolvingBalance'],loans['NotFullyPaid'])
```

Out[80]:
```
Ttest_indResult(statistic=48.99095349989636, pvalue=0.0)
```

There is some correlation between the two variables at significance level 0.05

In [81]:

```
plt.figure(figsize=(12,7), dpi=600)
loans['RevolvingBalance'].hist(alpha=0.5,color='#8A2BE2',
                                        bins=1000, cumulative=True )
plt.title("Distribution of RevolvingBalance", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Revolving Balance", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofRevolvingBalance.png",dpi=600)
```



Did we analyze DebtToLimitRatio column of the dataframe?

In [82]:
```
DTL=loans['DebtToLimitRatio'].describe()
DTL
```

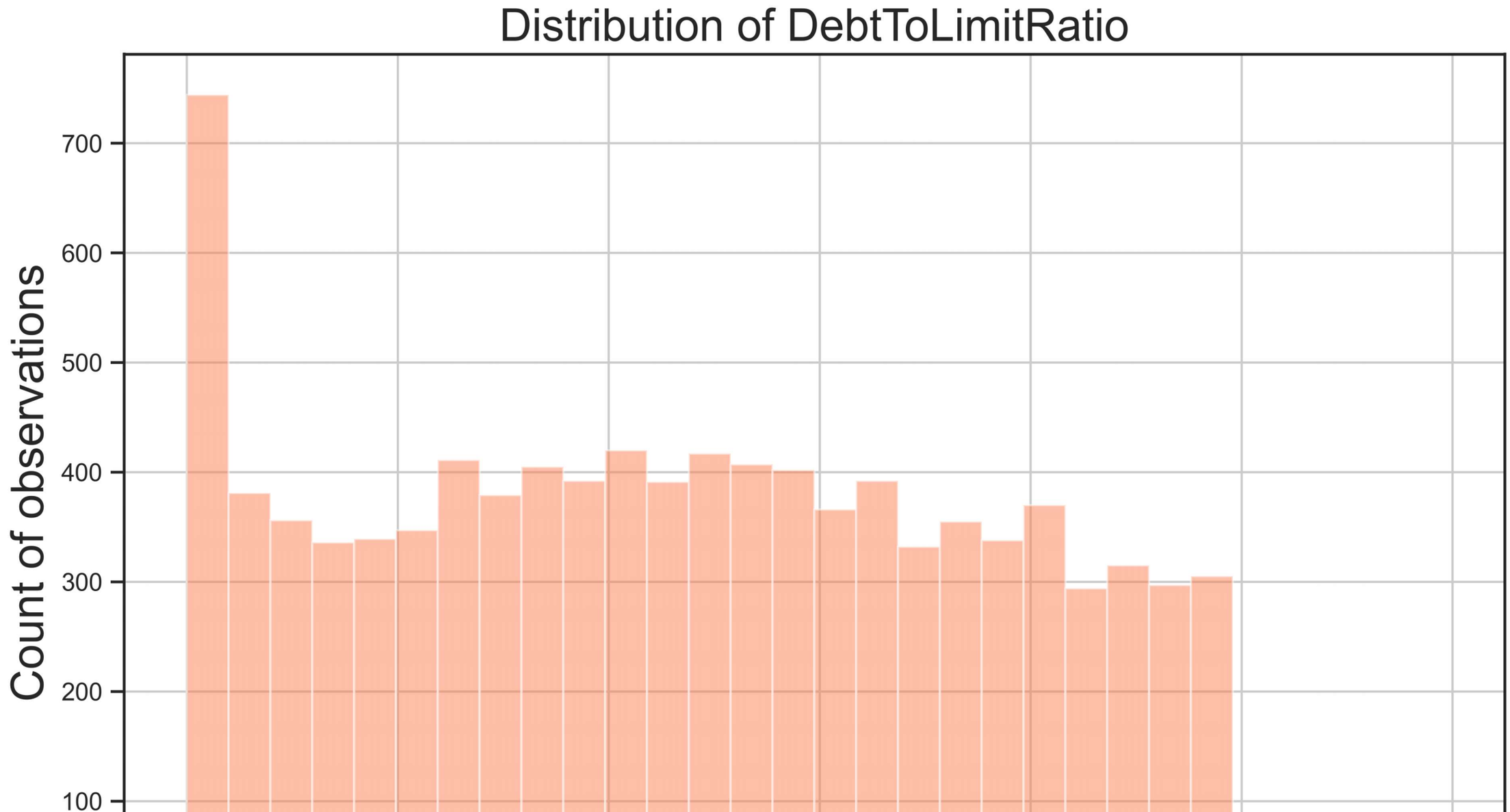
Out[82]:
```
count    9568.00000
mean       46.79911
std        29.02057
min         0.00000
25%        22.60000
50%        46.30000
75%        70.90000
max       119.00000
Name: DebtToLimitRatio, dtype: float64
```

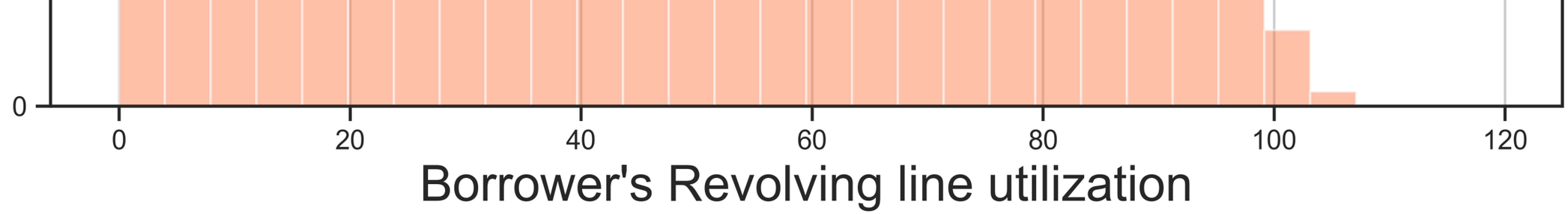
In [83]:
```
ttest_ind(loans['DebtToLimitRatio'],loans['NotFullyPaid'])
```

Out[83]:
```
Ttest_indResult(statistic=157.18826651260008, pvalue=0.0)
```

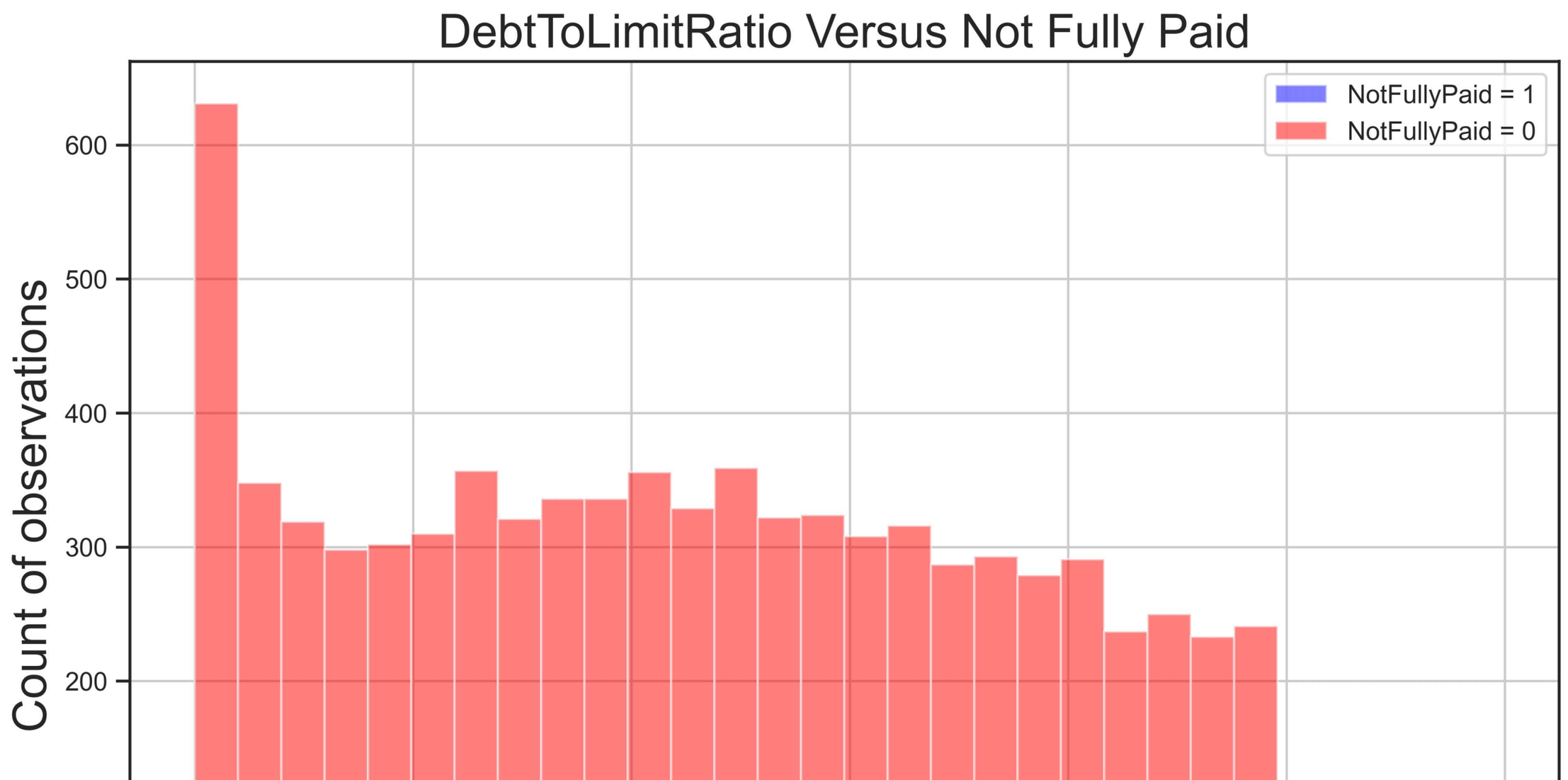
There is some correlation between the two variables at significance level 0.05

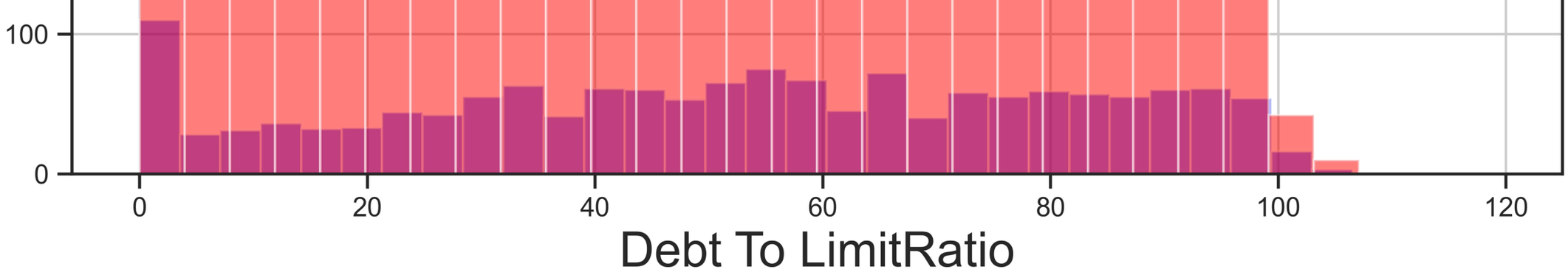
In [84]:
```
plt.figure(figsize=(11,7), dpi=600)
loans['DebtToLimitRatio'].hist(alpha=0.5,color='coral',
                                        bins=30)
plt.title("Distribution of DebtToLimitRatio", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Borrower's Revolving line utilization", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDebtToLimitRatio.png",dpi=600)
```




0
0
20
40
60
80
100
120
Borrower's Revolving line utilization

In [85]:
```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['DebtToLimitRatio'].hist(alpha=0.5,color='blue',
                                            bins=30,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['DebtToLimitRatio'].hist(alpha=0.5,color='red',
                                            bins=30,label='NotFullyPaid = 0')
plt.legend()
plt.title("DebtToLimitRatio Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Debt To LimitRatio", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDaysDebtToLimitRatioVNotFullyPaid.png",dpi=600)
```





Did we analyze InquirySixMonths column of the dataframe?

In [86]:
```
ISM=loans['InquirySixMonths'].describe()
ISM
```

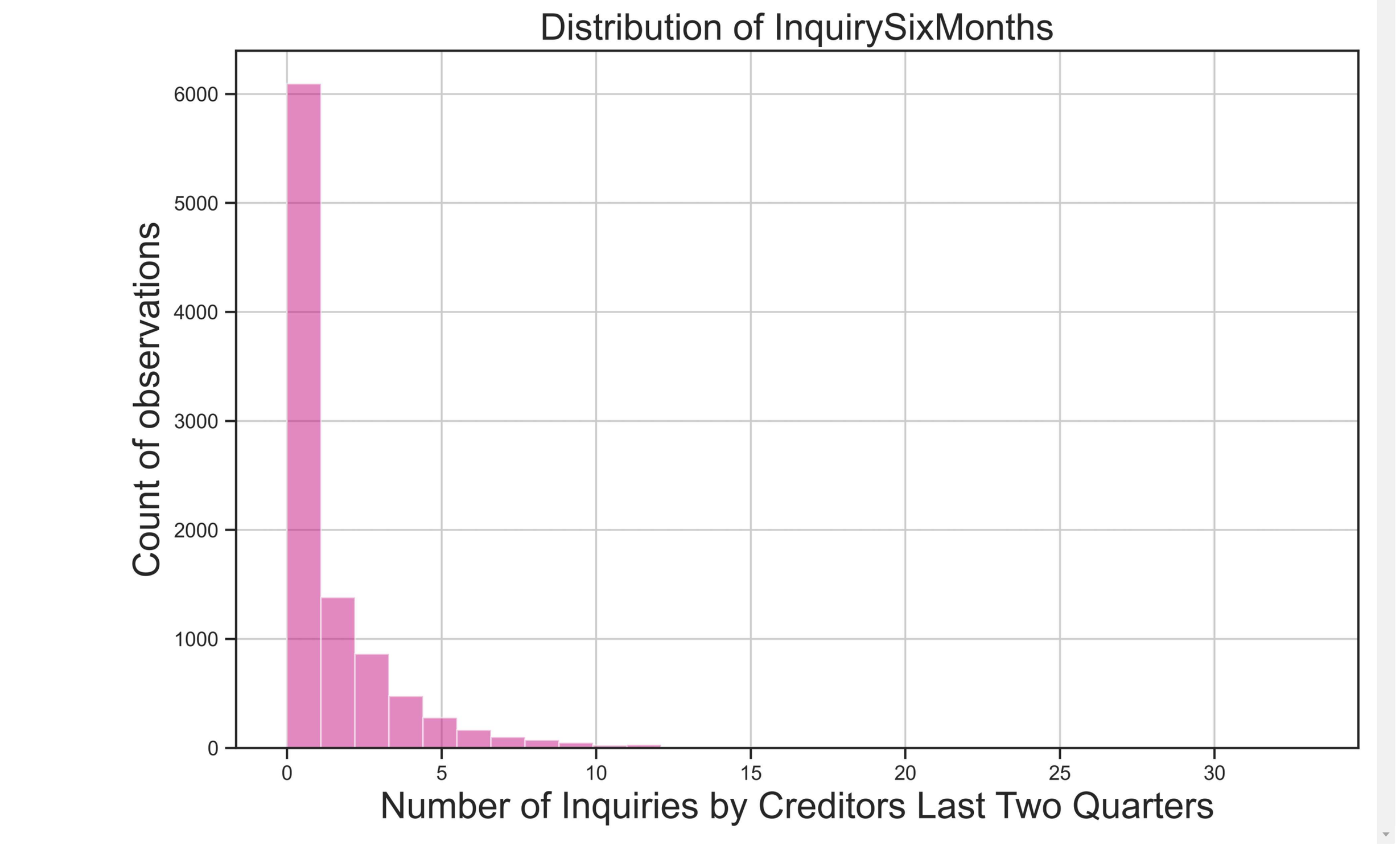
Out[86]:
```
count    9568.000000
mean        1.577864
std         2.201038
min         0.000000
25%         0.000000
50%         1.000000
75%         2.000000
max        33.000000
Name: InquirySixMonths, dtype: float64
```

In [87]:
```
ttest_ind(loans['InquirySixMonths'],loans['NotFullyPaid'])
```

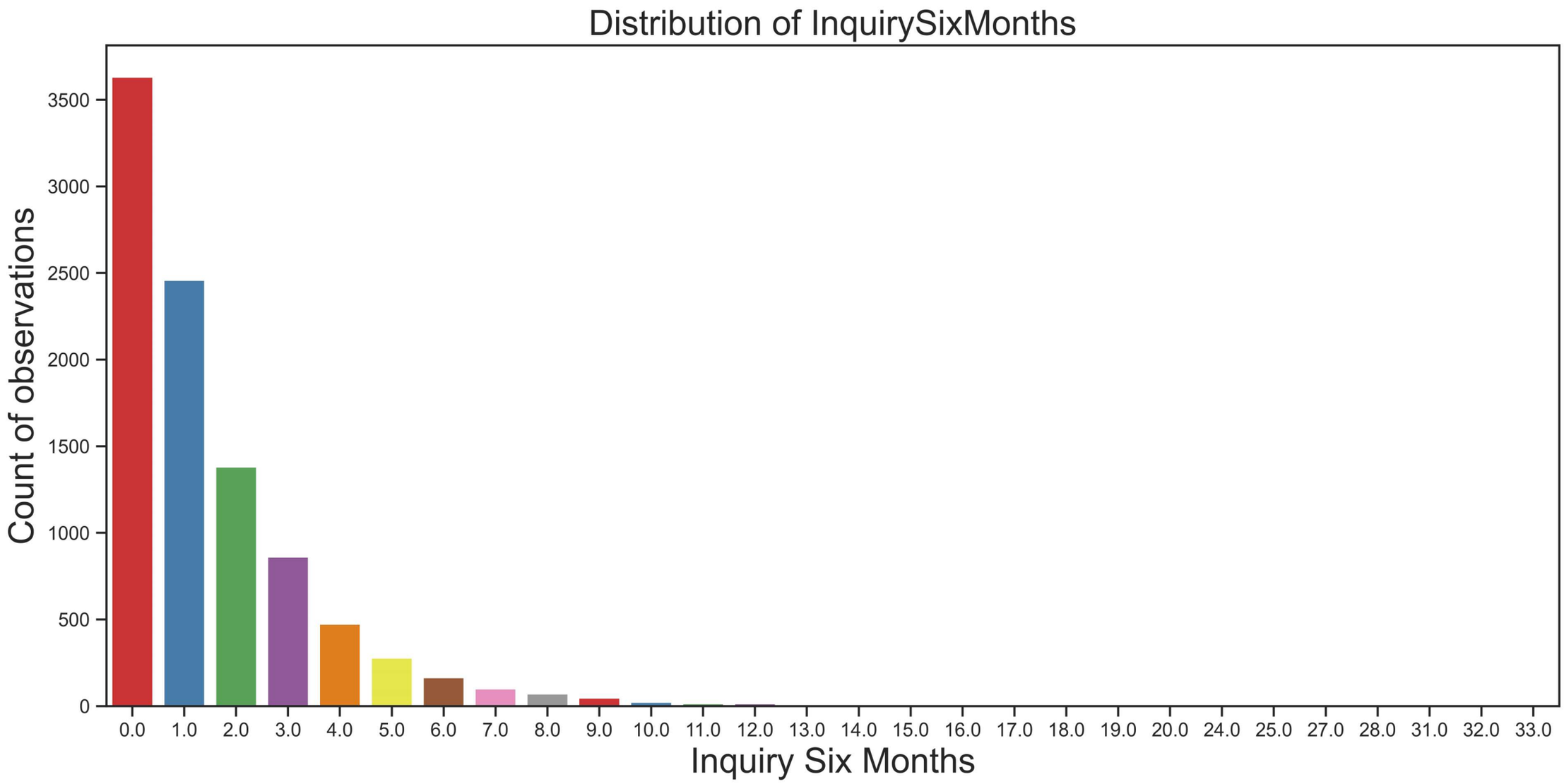
Out[87]:
```
Ttest_indResult(statistic=62.154207595689705, pvalue=0.0)
```

There is some correlation between the two variables at significance level 0.05

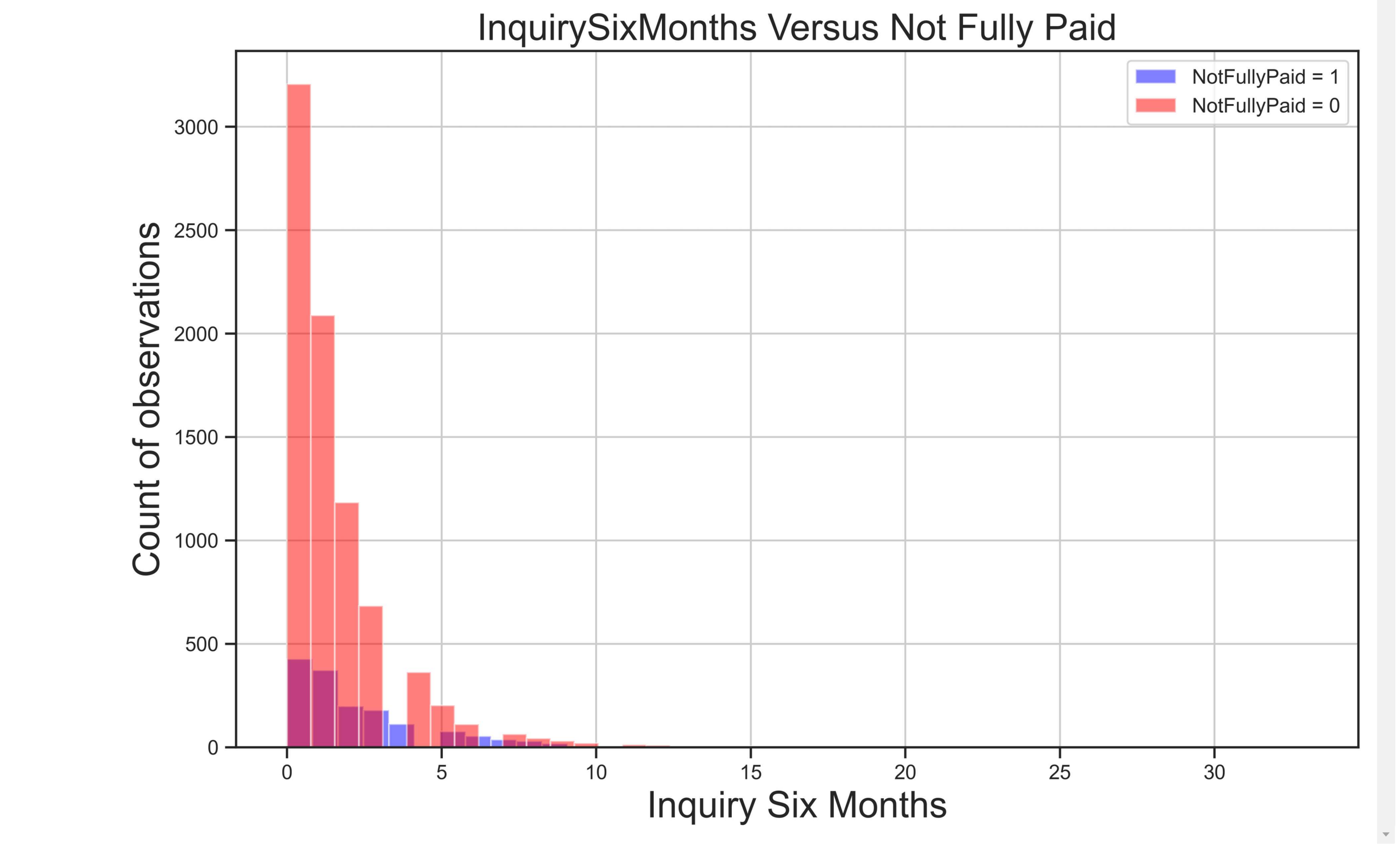
In [88]:
```
plt.figure(figsize=(11,7), dpi=600)
loans['InquirySixMonths'].hist(alpha=0.5,color='#C71585',
                                          bins=30)
plt.title("Distribution of InquirySixMonths", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Number of Inquiries by Creditors Last Two Quarters", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofInquirySixMonths.png",dpi=600)
```


Distribution of InquirySixMonths
Count of observations
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30
Number of Inquiries by Creditors Last Two Quarters

In [89]:
```
plt.figure(figsize=(15,7), dpi=600)
sns.countplot(x='InquirySixMonths',data=loans,palette='Set1')
plt.title("Distribution of InquirySixMonths", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Inquiry Six Months", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofInquirySixMonthsBar.png",dpi=600)
```



In [90]:
```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['InquirySixMonths'].hist(alpha=0.5,color='blue',
                                            bins=40,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['InquirySixMonths'].hist(alpha=0.5,color='red',
                                            bins=40,label='NotFullyPaid = 0')
plt.legend()
plt.title("InquirySixMonths Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Inquiry Six Months", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofInquirySixMonthsVNotFullyPaid.png",dpi=600)
```


InquirySixMonths Versus Not Fully Paid
NotFullyPaid = 1
NotFullyPaid = 0
Count of observations
3000
2500
2000
1500
1000
500
0
0
5
10
15
20
25
30
Inquiry Six Months

Did we analyze DelayedPaymentsTwoYears column of the dataframe?

In [91]:
```
DPTY=loans['DelayedPaymentsTwoYears'].value_counts()
DPTY
```

Out[91]:
```
0.0     8448
1.0      832
2.0      192
3.0       65
4.0       19
5.0        6
6.0        2
11.0       1
13.0       1
8.0        1
7.0        1
Name: DelayedPaymentsTwoYears, dtype: int64
```

In [92]:
```
table3 = pd.crosstab(loans['DelayedPaymentsTwoYears'],loans['NotFullyPaid'], margins=True)
table3
```

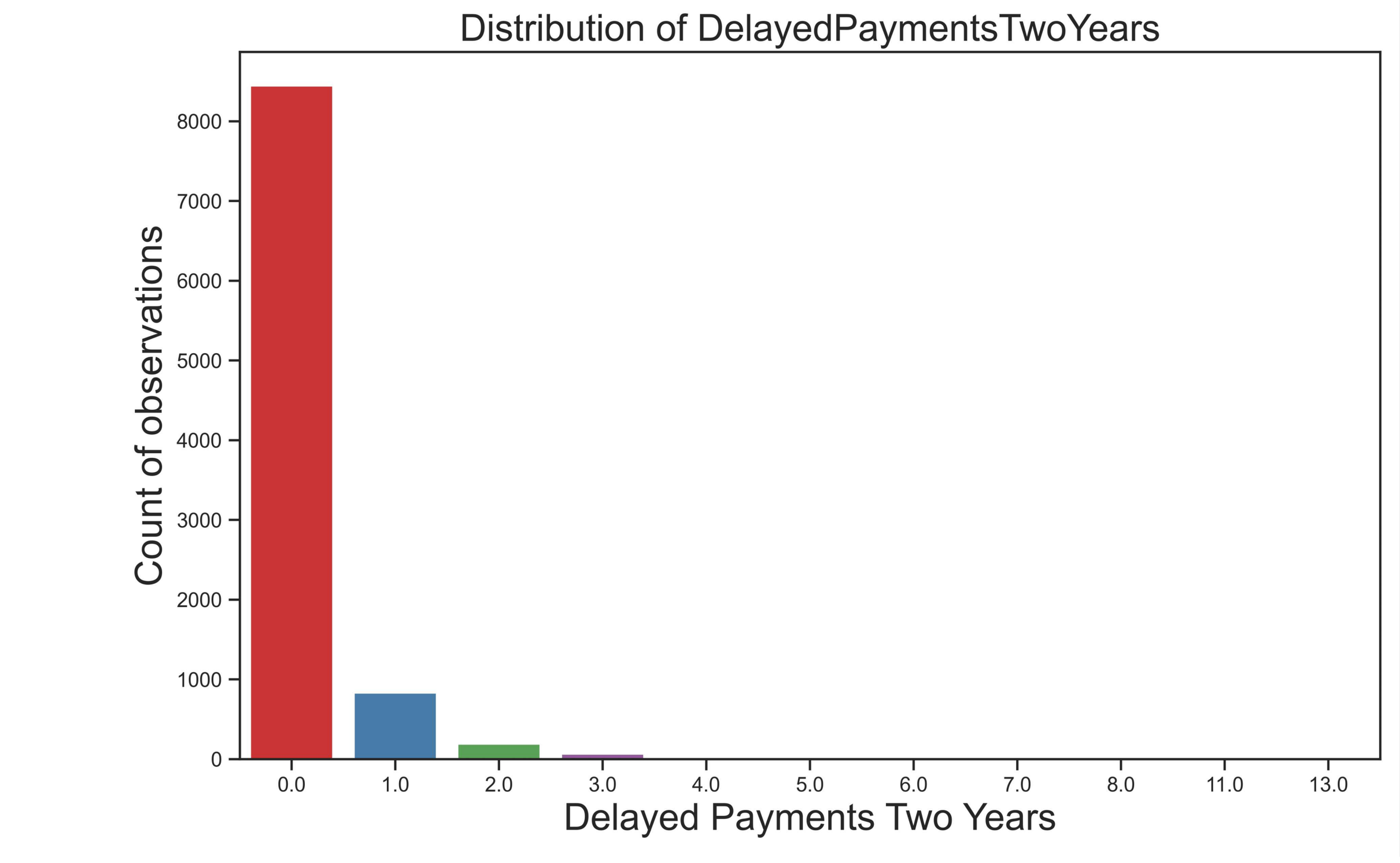
Out[92]:

NotFullyPaid	0.0	1.0	All
DelayedPaymentsTwoYears			
0.0	7113	1335	8448
1.0	687	145	832
2.0	158	34	192
3.0	52	13	65
4.0	15	4	19
5.0	6	0	6
6.0	2	0	2
7.0	1	0	1
8.0	1	0	1
11.0	1	0	1
13.0	1	0	1
All	8037	1531	9568

In [93]:
```
chi_square('DelayedPaymentsTwoYears','NotFullyPaid')
```

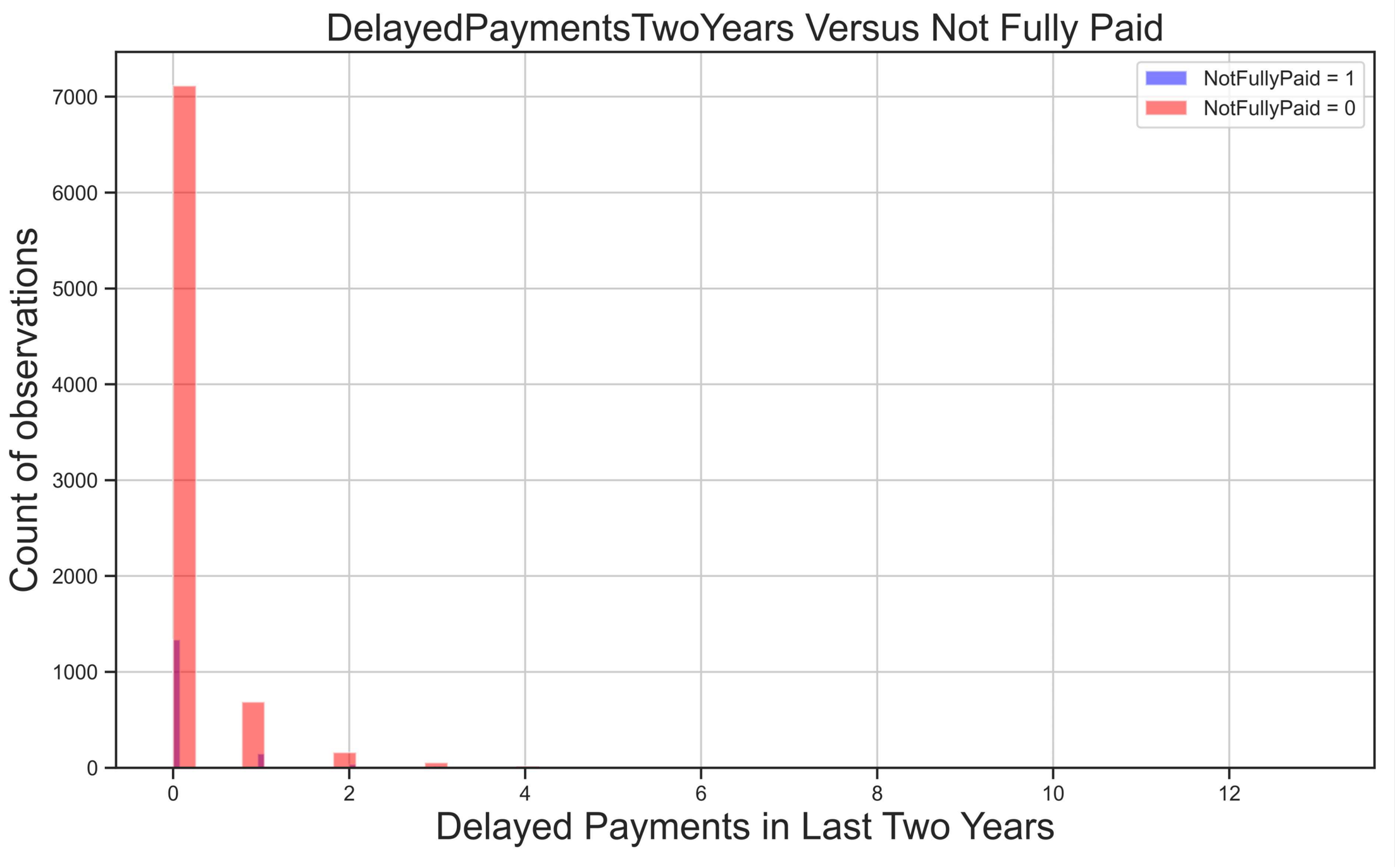
```
[[7.09621405e+03 1.35178595e+03]
 [6.98869565e+02 1.33130435e+02]
 [1.61277592e+02 3.07224080e+01]
 [5.45991848e+01 1.04008152e+01]
 [1.59597617e+01 3.04023829e+00]
 [5.03992475e+00 9.60075251e-01]
 [1.67997492e+00 3.20025084e-01]
 [8.39987458e-01 1.60012542e-01]
 [8.39987458e-01 1.60012542e-01]
 [8.39987458e-01 1.60012542e-01]
 [8.39987458e-01 1.60012542e-01]]

Chi-square is : 5.344180

p_value is : 0.867033

degree of freedom is : 10

There is no correlation between the two variables
```

In [94]:
```
plt.figure(figsize=(11,7), dpi=600)
sns.countplot(x='DelayedPaymentsTwoYears',data=loans,palette='Set1')
plt.title("Distribution of DelayedPaymentsTwoYears", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Delayed Payments Two Years", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDelayedPaymentsTwoYears.png",dpi=600)
```


Distribution of DelayedPaymentsTwoYears
Count of observations
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
11.0
13.0
Delayed Payments Two Years

In [95]:

```
plt.figure(figsize=(12,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['DelayedPaymentsTwoYears'].hist(alpha=0.5,color='blue',
                                            bins=50,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['DelayedPaymentsTwoYears'].hist(alpha=0.5,color='red',
                                            bins=50,label='NotFullyPaid = 0')
plt.legend()
plt.title("DelayedPaymentsTwoYears Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Delayed Payments in Last Two Years", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDelayedPaymentsTwoYearsVNotFullyPaid.png",dpi=600)
```



Did we analyze DerogatoryPublicRecords column of the dataframe?

In [96]:
```
DPR=loans['DerogatoryPublicRecords'].value_counts()
DPR
```

Out[96]:
```
0.0    9010
1.0     532
2.0      19
3.0       5
5.0       1
4.0       1
Name: DerogatoryPublicRecords, dtype: int64
```

In [97]:
```
table4 = pd.crosstab(loans['DerogatoryPublicRecords'],loans['NotFullyPaid'], margins=True)
table4
```

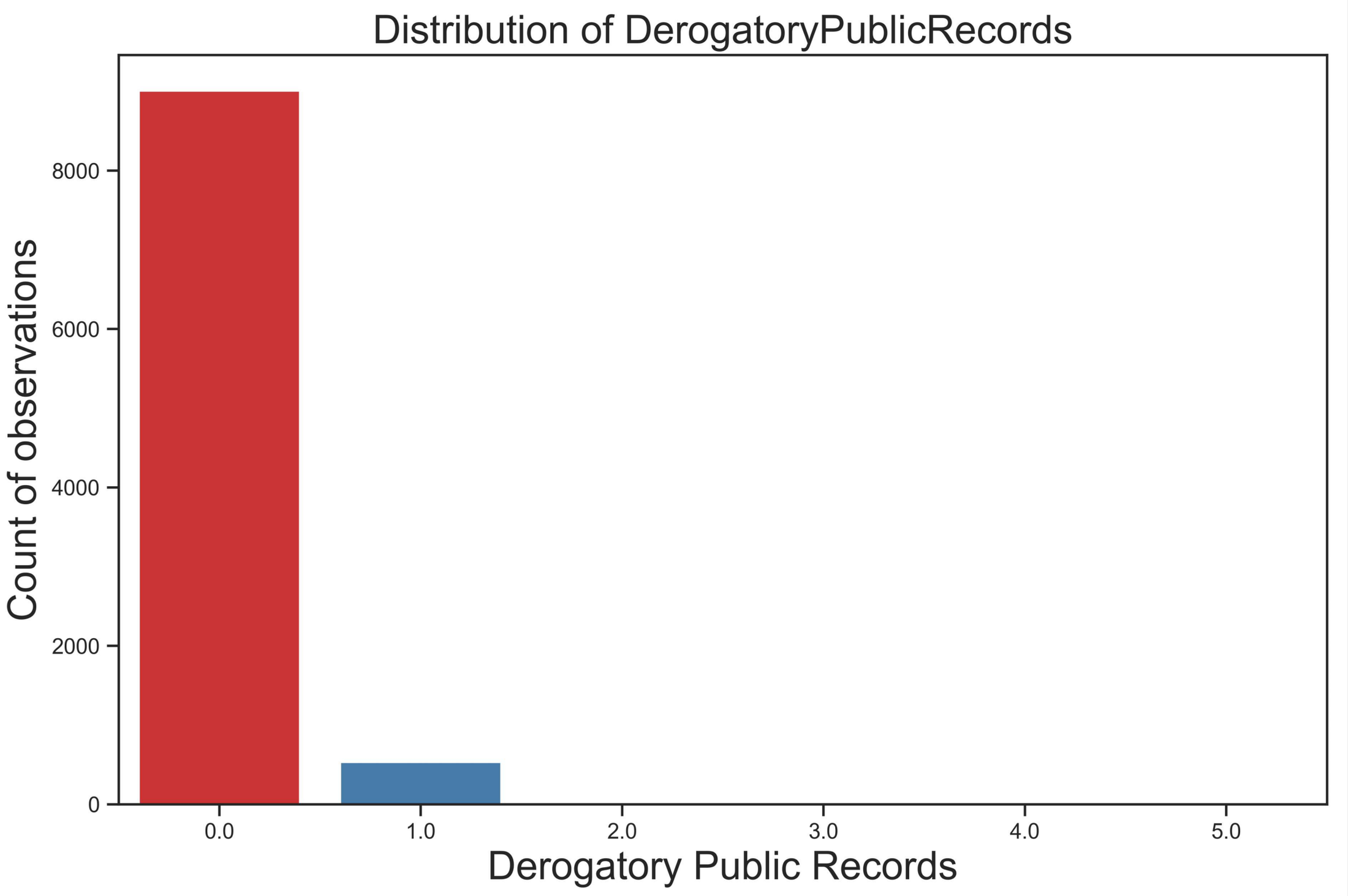
Out[97]:

NotFullyPaid	0.0	1.0	All
DerogatoryPublicRecords			
0.0	7617	1393	9010
1.0	396	136	532
2.0	17	2	19
3.0	5	0	5
4.0	1	0	1
5.0	1	0	1
All	8037	1531	9568

In [98]:
```
chi_square('DerogatoryPublicRecords','NotFullyPaid')
```

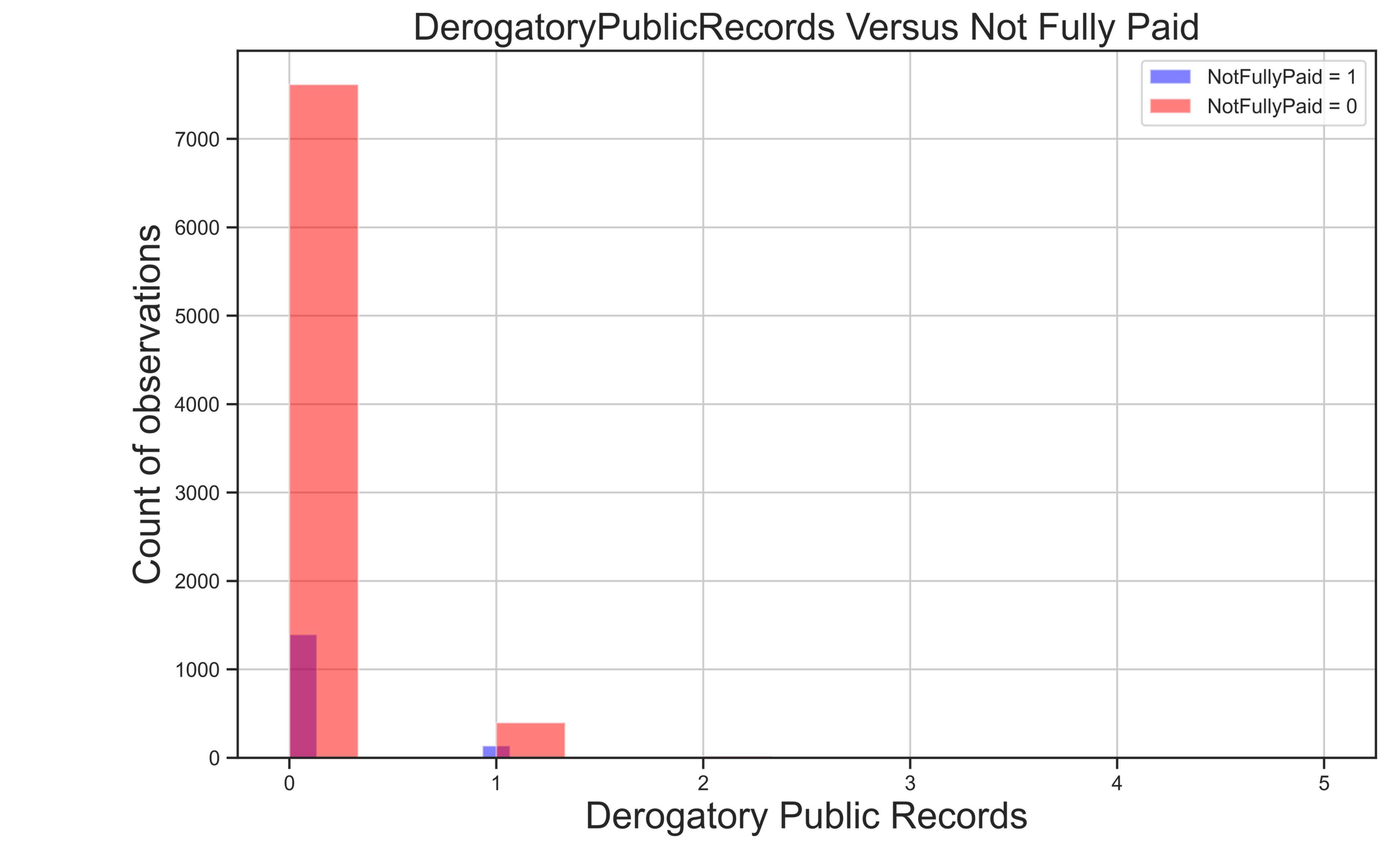
```
[[7.56828700e+03 1.44171300e+03]
 [4.46873328e+02 8.51266722e+01]
 [1.59597617e+01 3.04023829e+00]
 [4.19993729e+00 8.00062709e-01]
 [8.39987458e-01 1.60012542e-01]
 [8.39987458e-01 1.60012542e-01]]

Chi-square is : 39.911090

p_value is : 0.000000

degree of freedom is : 5

There is some correlation between the two variables at significance level 0.05
```

In [99]:
```
plt.figure(figsize=(11,7), dpi=600)
sns.countplot(x='DerogatoryPublicRecords',data=loans,palette='Set1')
plt.title("Distribution of DerogatoryPublicRecords", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Derogatory Public Records", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDerogatoryPublicRecords.png",dpi=600)
```


Distribution of DerogatoryPublicRecords
Count of observations
8000
6000
4000
2000
0
0.0
1.0
2.0
3.0
4.0
5.0
Derogatory Public Records

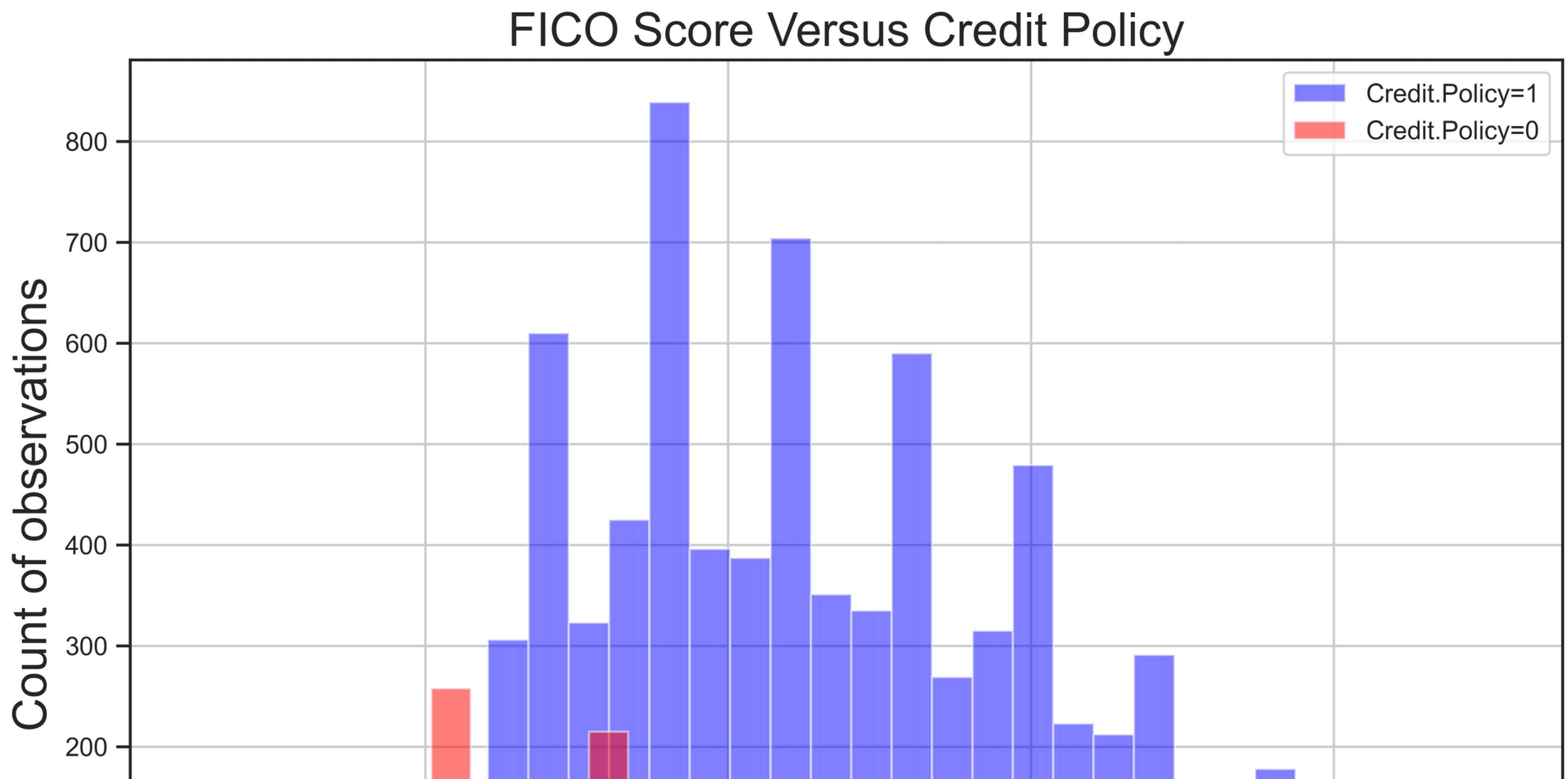
In [100]:

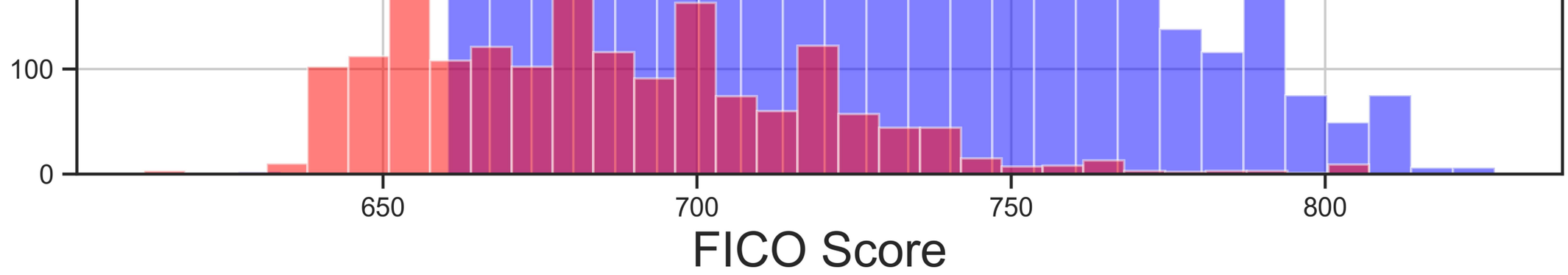
```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['NotFullyPaid']== 1 ]['DerogatoryPublicRecords'].hist(alpha=0.5,color='blue',
                                            bins=15,label='NotFullyPaid = 1')
loans[loans['NotFullyPaid']== 0 ]['DerogatoryPublicRecords'].hist(alpha=0.5,color='red',
                                            bins=15,label='NotFullyPaid = 0')
plt.legend()
plt.title("DerogatoryPublicRecords Versus Not Fully Paid", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("Derogatory Public Records", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofDerogatoryPublicRecordsVNotFullyPaid.png",dpi=600)
```


DerogatoryPublicRecords Versus Not Fully Paid
NotFullyPaid = 1
NotFullyPaid = 0
Count of observations
0
1000
2000
3000
4000
5000
6000
7000
0
1
2
3
4
5
Derogatory Public Records

Did we analyze CreditScore and FICO Score columns of the dataframe?

In [101]:
```
plt.figure(figsize=(11,7), dpi=600)
loans[loans['CreditPolicy']==1]['FICO.Score'].hist(alpha=0.5,color='blue',
                                              bins=30,label='Credit.Policy=1')
loans[loans['CreditPolicy']==0]['FICO.Score'].hist(alpha=0.5,color='red',
                                              bins=30,label='Credit.Policy=0')
plt.legend()
plt.title("FICO Score Versus Credit Policy", fontsize=20)
plt.xticks(rotation=0, horizontalalignment="center")
plt.xlabel("FICO Score", fontsize=20)
plt.ylabel("Count of observations", fontsize=20)
plt.savefig("DistributionofFICOScoreVCreditPolicy.png",dpi=600)
```





Did we changed the categorical features in to numeric ones?

Categorical Features

We notice that the Purpose column is categorical one.

That means we need to transform this column using dummy variables so that sklearn will be able to understand them. We performed this step using pd.get_dummies.

In [102]:
```
final_data = pd.get_dummies(loans,columns=['Purpose'],drop_first=True)
```

In [103]:
```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9568 entries, 0 to 9619
Data columns (total 19 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   CreditPolicy               9568 non-null   float64
 1   InterestRate               9568 non-null   float64
 2   Installment                9568 non-null   float64
 3   AnnualIncomeLn             9568 non-null   float64
 4   DebtToIncome               9568 non-null   float64
 5   FICO.Score                 9568 non-null   float64
 6   DaysSinceSubscription      9568 non-null   float64
 7   RevolvingBalance           9568 non-null   float64
 8   DebtToLimitRatio           9568 non-null   float64
 9   InquirySixMonths           9568 non-null   float64
 10  DelayedPaymentsTwoYears    9568 non-null   float64
 11  DerogatoryPublicRecords    9568 non-null   float64
 12  NotFullyPaid               9568 non-null   float64
 13  Purpose_credit_card        9568 non-null   uint8
 14  Purpose_debt_consolidation 9568 non-null   uint8
 15  Purpose_educational        9568 non-null   uint8
 16  Purpose_home_improvement   9568 non-null   uint8
 17  Purpose_major_purchase     9568 non-null   uint8
 18  Purpose_small_business     9568 non-null   uint8
dtypes: float64(13), uint8(6)
memory usage: 1.4 MB
```

In [104]:
```
final_data.head(5)
```

Out[104]:

›Years	DerogatoryPublicRecords	NotFullyPaid	Purpose_credit_card	Purpose_debt_consolidation	Purpose_educational	Purpose_home_improvement	Purpose_major_purchase	Purpose_small_business
0.0	0.0	0.0	0	1	0	0	0	0
0.0	0.0	0.0	1	0	0	0	0	0
0.0	0.0	0.0	0	1	0	0	0	0
0.0	0.0	0.0	0	1	0	0	0	0
1.0	0.0	0.0	1	0	0	0	0	0

Let us shuffle the dataset for proper sampling.

In [105]:
```
final_data = final_data.sample(frac = 1, random_state=1)
final_data.head(5)
```

Out[105]:

	CreditPolicy	InterestRate	Installment	AnnualIncomeLn	DebtToIncome	FICO.Score	DaysSinceSubscription	RevolvingBalance	DebtToLimitRatio	InquirySixMonths	DelayedPaymentsTwoYears
5053	1.0	0.1565	195.93	11.775290	6.94	667.0	7229.958333	9044.0	93.2	0.0	0.0
6999	1.0	0.1253	261.04	10.645425	19.00	702.0	4320.000000	0.0	0.0	2.0	0.0
9282	0.0	0.1392	409.65	10.505068	18.74	682.0	3932.958333	7780.0	41.6	4.0	2.0
4309	1.0	0.1253	401.60	11.211820	8.29	707.0	8969.958333	4358.0	46.4	2.0	0.0
6529	1.0	0.1600	527.36	11.512925	18.91	667.0	6420.041667	30523.0	66.2	0.0	0.0

In [106]:
```
final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9568 entries, 5053 to 254
Data columns (total 19 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   CreditPolicy               9568 non-null   float64
 1   InterestRate               9568 non-null   float64
 2   Installment                9568 non-null   float64
 3   AnnualIncomeLn             9568 non-null   float64
 4   DebtToIncome               9568 non-null   float64
 5   FICO.Score                 9568 non-null   float64
 6   DaysSinceSubscription      9568 non-null   float64
 7   RevolvingBalance           9568 non-null   float64
 8   DebtToLimitRatio           9568 non-null   float64
 9   InquirySixMonths           9568 non-null   float64
 10  DelayedPaymentsTwoYears    9568 non-null   float64
 11  DerogatoryPublicRecords    9568 non-null   float64
 12  NotFullyPaid               9568 non-null   float64
 13  Purpose_credit_card        9568 non-null   uint8
 14  Purpose_debt_consolidation 9568 non-null   uint8
 15  Purpose_educational        9568 non-null   uint8
 16  Purpose_home_improvement   9568 non-null   uint8
 17  Purpose_major_purchase     9568 non-null   uint8
 18  Purpose_small_business     9568 non-null   uint8
dtypes: float64(13), uint8(6)
memory usage: 1.1 MB
```

Let us extract dependent and independent .

In [107]:
```
X = final_data.drop(['NotFullyPaid'],axis=1)
y = final_data.NotFullyPaid
```

In [108]:
```
X.head(10)
```

Out[108]:

	CreditPolicy	InterestRate	Installment	AnnualIncomeLn	DebtToIncome	FICO.Score	DaysSinceSubscription	RevolvingBalance	DebtToLimitRatio	InquirySixMonths	DelayedPaymentsTwoYears
5053	1.0	0.1565	195.93	11.775290	6.94	667.0	7229.958333	9044.0	93.2	0.0	0.0
6999	1.0	0.1253	261.04	10.645425	19.00	702.0	4320.000000	0.0	0.0	2.0	0.0
9282	0.0	0.1392	409.65	10.505068	18.74	682.0	3932.958333	7780.0	41.6	4.0	2.0
4309	1.0	0.1253	401.60	11.211820	8.29	707.0	8969.958333	4358.0	46.4	2.0	0.0
6529	1.0	0.1600	527.36	11.512925	18.91	667.0	6420.041667	30523.0	66.2	0.0	0.0
6378	1.0	0.0859	158.06	11.034890	16.30	732.0	4110.041667	7739.0	32.8	0.0	0.0
3243	1.0	0.1189	663.28	10.781037	10.55	792.0	5022.000000	53287.0	0.2	5.0	0.0
8752	0.0	0.1197	199.20	10.778956	15.38	677.0	4229.958333	4956.0	40.0	4.0	0.0
6577	1.0	0.0774	156.10	10.872902	23.10	757.0	3030.041667	4987.0	12.2	0.0	0.0
7310	1.0	0.0751	357.76	11.082143	9.82	787.0	5558.000000	1944.0	13.6	3.0	0.0

In [109]:
```
y.head(10)
```

Out[109]:
```
5053    0.0
6999    0.0
9282    0.0
4309    0.0
6529    0.0
6378    0.0
3243    1.0
8752    0.0
6577    0.0
7310    0.0
Name: NotFullyPaid, dtype: float64
```

Let us Ensure that all variables are numeric now.

In [110]:
```
X.describe().transpose()
```

Out[110]:

	count	mean	std	min	25%	50%	75%	max
CreditPolicy	9568.0	0.804766	0.396402	0.000000	1.000000	1.000000	1.000000	1.000000e+00
InterestRate	9568.0	0.122644	0.026850	0.060000	0.103900	0.122100	0.140700	2.164000e-01
Installment	9568.0	319.021305	207.052703	15.670000	163.770000	268.950000	432.282500	9.401400e+02
AnnualIncomeLn	9568.0	10.931854	0.614912	7.547502	10.558414	10.927987	11.289832	1.452835e+01
DebtToIncome	9568.0	12.604398	6.885845	0.000000	7.207500	12.660000	17.950000	2.996000e+01
FICO.Score	9568.0	710.832044	37.981188	612.000000	682.000000	707.000000	737.000000	8.270000e+02
DaysSinceSubscription	9568.0	4559.426038	2496.307691	178.958333	2819.958333	4139.958333	5730.000000	1.763996e+04
RevolvingBalance	9568.0	16914.018186	33770.491059	0.000000	3184.750000	8593.000000	18244.500000	1.207359e+06
DebtToLimitRatio	9568.0	46.799110	29.020570	0.000000	22.600000	46.300000	70.900000	1.190000e+02
InquirySixMonths	9568.0	1.577864	2.201038	0.000000	0.000000	1.000000	2.000000	3.300000e+01
DelayedPaymentsTwoYears	9568.0	0.163880	0.546475	0.000000	0.000000	0.000000	0.000000	1.300000e+01
DerogatoryPublicRecords	9568.0	0.062082	0.262081	0.000000	0.000000	0.000000	0.000000	5.000000e+00
Purpose_credit_card	9568.0	0.131584	0.338056	0.000000	0.000000	0.000000	0.000000	1.000000e+00
Purpose_debt_consolidation	9568.0	0.413880	0.492553	0.000000	0.000000	0.000000	1.000000	1.000000e+00
Purpose_educational	9568.0	0.035744	0.185661	0.000000	0.000000	0.000000	0.000000	1.000000e+00
Purpose_home_improvement	9568.0	0.065740	0.247840	0.000000	0.000000	0.000000	0.000000	1.000000e+00
Purpose_major_purchase	9568.0	0.045569	0.208558	0.000000	0.000000	0.000000	0.000000	1.000000e+00
Purpose_small_business	9568.0	0.064590	0.245814	0.000000	0.000000	0.000000	0.000000	1.000000e+00

Let us perform feature scaling as the range of values for different features vary a lot.

In [111]:
```
from sklearn.preprocessing import MinMaxScaler
sc= MinMaxScaler()
X_scaled = pd.DataFrame(sc.fit_transform(X),columns = X.columns)
X_scaled.head()
```

Out[111]:

	CreditPolicy	InterestRate	Installment	AnnualIncomeLn	DebtToIncome	FICO.Score	DaysSinceSubscription	RevolvingBalance	DebtToLimitRatio	InquirySixMonths	DelayedPaymentsTwoYears	De
0	1.0	0.617008	0.194987	0.605626	0.231642	0.255814	0.403814	0.007491	0.783193	0.000000	0.000000	
1	1.0	0.417519	0.265417	0.443774	0.634179	0.418605	0.237159	0.000000	0.000000	0.060606	0.000000	
2	0.0	0.506394	0.426169	0.423668	0.625501	0.325581	0.214993	0.006444	0.349580	0.121212	0.153846	
3	1.0	0.417519	0.417461	0.524910	0.276702	0.441860	0.503465	0.003610	0.389916	0.060606	0.000000	
4	1.0	0.639386	0.553496	0.568043	0.631175	0.255814	0.357430	0.025281	0.556303	0.000000	0.000000	

Let us do k-fold cross validation using Logistic Regression.

In [112]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

cross_val_score(LogisticRegression(),X_scaled,y,cv=3).mean()
```

Out[112]: 0.840196561259266

In [113]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

cross_val_score(LogisticRegression(),X_scaled,y,cv=4).mean()
```

Out[113]: 0.8401964882943144

In [114]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

cross_val_score(LogisticRegression(),X_scaled,y,cv=5).mean()
```

Out[114]: 0.8406144834250174

In [115]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

cross_val_score(LogisticRegression(),X_scaled,y,cv=6).mean()
```

Out[115]:
```
0.8403010767913636
```

Let's also look at the test scores of a simple split. We did split into training and test set for cross-validation

In [116]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_scaled,y,random_state=0,test_size=0.25)
X_test.head()
```

Out[116]:

	CreditPolicy	InterestRate	Installment	AnnualIncomeLn	DebtToIncome	FICO.Score	DaysSinceSubscription	RevolvingBalance	DebtToLimitRatio	InquirySixMonths	DelayedPaymentsTwoYears
4834	1.0	0.212276	0.065995	0.481095	0.156542	0.581395	0.311036	0.001809	0.605882	0.030303	0.000000
1768	0.0	0.503197	0.130659	0.474369	0.230307	0.162791	0.142606	0.004367	0.661345	0.000000	0.000000
2819	1.0	0.395141	0.703463	0.576695	0.461949	0.558140	0.526602	0.018415	0.644538	0.030303	0.000000
7779	0.0	0.558824	0.356637	0.494868	0.253672	0.325581	0.329940	0.026631	0.531933	0.151515	0.153846
7065	0.0	0.684783	0.105963	0.443774	0.936582	0.279070	0.173592	0.021335	0.536134	0.151515	0.000000

Let us train the model on the training set.

In [117]:
```
model = LogisticRegression()
model.fit(X_train,y_train)
```

Out[117]:
```
LogisticRegression()
```

Let us look for the Test accuracy.

In [118]:
```
model.score(X_test,y_test)
```

Out[118]:
```
0.8344481605351171
```

Let's look at some other scores as well.

In [121]:
```
from sklearn.metrics import classification_report, confusion_matrix
y_pred = model.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.84      1.00      0.91      1990
         1.0       0.65      0.03      0.06       402

    accuracy                           0.83      2392
   macro avg       0.74      0.51      0.49      2392
weighted avg       0.80      0.83      0.77      2392
```

In [136]:
```
from sklearn.metrics import confusion_matrix
cmatrix = confusion_matrix(y_test,y_pred)
print(f'\n{cmatrix}')
```

```
[[1983    7]
 [ 389   13]]
```

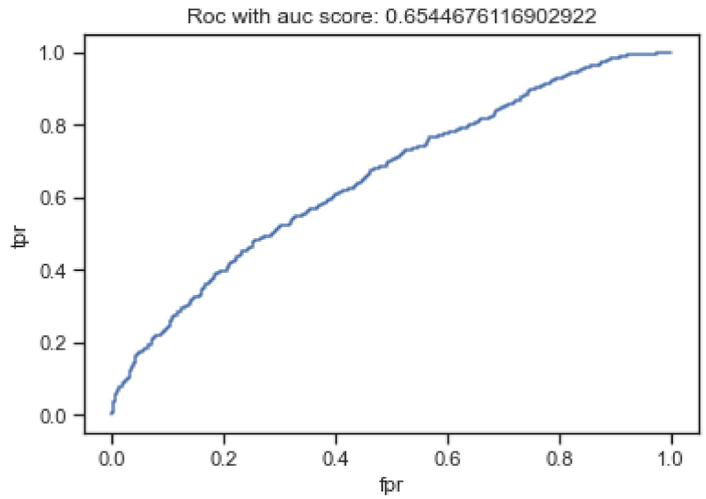
In [140]:
```
    TN = cmatrix[0][0]
    FP = cmatrix[0][1]
    FN = cmatrix[1][0]
    TP = cmatrix[1][1]
    print(f'\nTrue Positive (TP) = {TP}\n')
    print(f'Flase Positive (FP) = {FP}\n')
    print(f'False Negative (FN) = {FN}\n')
    print(f'True Negative (TN) = {TN}\n')
    print(f'Accuracy = {(TP+TN)/(TP+FN+FP+TN)}\n')
    print(f'Sensitivity (True Positive Rate, TPR) = {(TP)/(TP+FN)}\n')
    print(f'Specificity (True Negative Rate) = {(TN)/(FP+TN)}\n')
    print(f'False Positive Rate (FPR) = {(FP)/(FP+TN)}\n')
```

```
True Positive (TP) = 13

Flase Positive (FP) = 7

False Negative (FN) = 389

True Negative (TN) = 1983

Accuracy = 0.8344481605351171

Sensitivity (True Positive Rate, TPR) = 0.03233830845771144

Specificity (True Negative Rate) = 0.9964824120603015

False Positive Rate (FPR) = 0.0035175879396984926
```

Let us now Draw the ROC curve with ROC_AUC_SCORE.

In [ ]:
```
#ROC is the curve traced by the co-ordinates (False Positve Rate(FPR) at x-axis and True Positve Rate(TPR) in the y-axis
#for different probability threshold values
#AUC is the area under the ROC curve
#taking the second column for correct auc score
#second column represents the '1' class
#plot the curve with auc score
```

In [134]:
```
from sklearn.metrics import roc_auc_score, roc_curve

y_pred_prob = model.predict_proba(X_test)[:,1]


fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob, pos_label=1)


import  matplotlib.pyplot as plt
plt.plot(fpr,tpr)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Roc with auc score: {}'.format(roc_auc_score(y_test,y_pred_prob)))
plt.show()
```



Conclusion:

AUC is an excellent performance measure for Logistic Regression Model as it is robust against probability threshold values and truly depicts if the model is good or not for the data at hand. Here 0.65 means that Logistic Regression Model is probably not the best fit for the data.

True Positive Rate is very low and hence model will probably fail to predict positive class.

We could have tried to deal with the imbalance of the target column for better perfrmance of the model.

We can go ahead and check other classification methods like decision tree and random forest to find out whether these two perform better compared to the Logistic Regression targeting the parameter that we want to improve.