In [1]:
```
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
from scipy import stats
import numpy as np
import seaborn as sb
from matplotlib import pyplot
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
import sklearn.metrics as sm
from sklearn.preprocessing import PolynomialFeatures
```

In [2]:
```
dataset = pd.read_csv("loans_full_schema.csv", low_memory=False)
```

In [3]:
```
dataset.head()
```

Out[3]:

	emp_title	emp_length	state	homeownership	annual_income	verified_income	debt_to_inc
0	global config engineer	3.0	NJ	MORTGAGE	90000.0	Verified	
1	warehouse office clerk	10.0	HI	RENT	40000.0	Not Verified	
2	assembly	3.0	WI	RENT	40000.0	Source Verified	
3	customer service	1.0	PA	RENT	30000.0	Not Verified	
4	security supervisor	10.0	CA	RENT	35000.0	Verified	

5 rows × 55 columns

In [4]:
```
dataset.info()
```

```
public_record_bankrupt                10000 non-null int64
loan_purpose                          10000 non-null object
application_type                      10000 non-null object
loan_amount                           10000 non-null int64
term                                  10000 non-null int64
interest_rate                         10000 non-null float64
installment                           10000 non-null float64
grade                                 10000 non-null object
sub_grade                             10000 non-null object
issue_month                           10000 non-null object
loan_status                           10000 non-null object
initial_listing_status                10000 non-null object
disbursement_method                   10000 non-null object
balance                               10000 non-null float64
paid_total                            10000 non-null float64
paid_principal                        10000 non-null float64
paid_interest                         10000 non-null float64
paid_late_fees                        10000 non-null float64
dtypes: float64(17), int64(25), object(13)
memory usage: 4.2+ MB
```

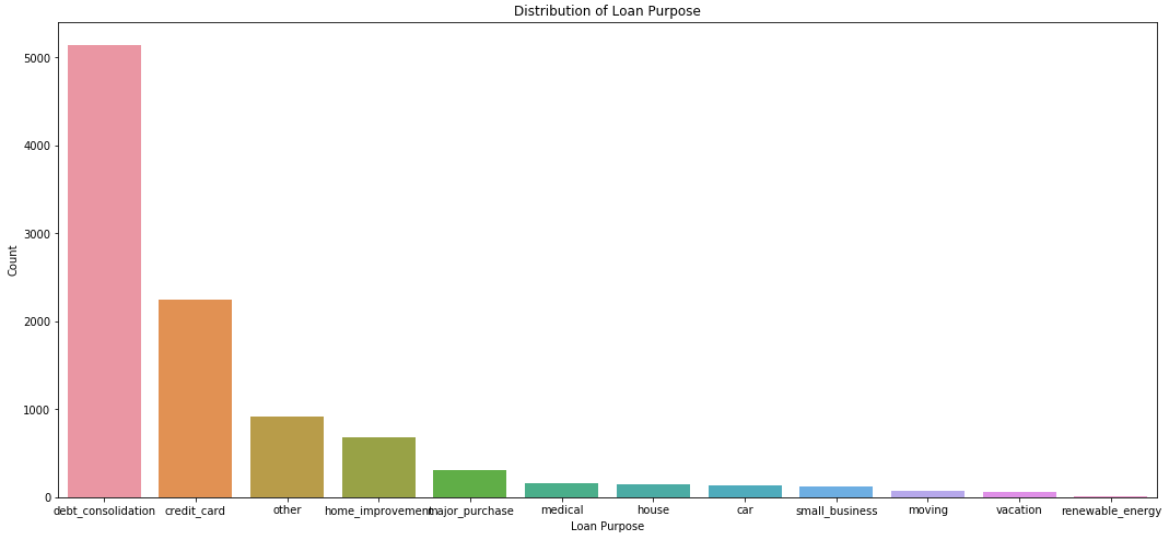
In [5]:

```
### In my visual first screening i will remove the following columns as thes
### all these columns are irrelevent for model unless there is any special c
### dropping column emp_title , emp_length, state, application_type, loan_st

dataset.drop(['emp_title' , 'emp_length', 'state', 'application_type', 'loar

## I am adding column annual income (which is primary applicant income) with
## the total income for the household. I will create a new column named totc
### and annual_income_joint

##first we will replace the nan values in the column annual_income_joint wit
dataset['annual_income_joint'].fillna(0, inplace=True)
dataset['total_annual_income'] = dataset['annual_income'] + dataset['annual_

dataset.drop(['annual_income','annual_income_joint'], axis=1, inplace=True)

## As the column verification income joint have only 1495 rows and 90% of th
## income verification status i am dropping this column 'verification income

dataset.drop(['verification_income_joint'], axis=1, inplace=True)

## information of column debt to income is important but from the data it dc
### percentage e..g if debt to income has value 18.01 then it says 18% is de

## I am going to replace the NaNs in the column debt_to_income_joint with ze
## as theses are not missing values.

dataset['debt_to_income_joint'].fillna(0, inplace=True)
dataset['debt_to_income'].fillna(0, inplace=True)

## the column earliest_credit_line has year values. I am going to substract
## so we would get how much old the earliest credit line was

dataset['earliest_credit_line_age'] = 2018 - dataset['earliest_credit_line']
dataset.drop(['earliest_credit_line'], axis=1, inplace=True)

### I am dropping the column total credit lines as the column open(active) c

dataset.drop(['total_credit_lines'], axis=1, inplace=True)

## I am going to calculate the percentage credit utilized and create a new c
## then i will drop columns total_credit_limit, total_credit_utilized

dataset['percentage_credit_utilized'] = (dataset['total_credit_utilized'] /
dataset.drop(['total_credit_utilized','total_credit_limit'], axis=1, inplace

## i will change the months to year for column 'months_since_90d_late' as al

dataset['months_since_90d_late'].fillna(0, inplace=True)
dataset['year_since_90dlate'] = (dataset['months_since_90d_late'] / 12).roun
dataset.drop(['months_since_90d_late'], axis=1, inplace=True)

## I am counting the zeros in the column num_accounts_120d_past_due,

a =(dataset['num_accounts_120d_past_due'] == 0).sum()
print('the zeros in column num_accounts_120d_past_due are:', a)
```

```
b = (dataset['num_accounts_30d_past_due'] == 0).sum()
print('the zeros in column num_accounts_30d_past_due:', b)

## The zeros in above columns are valid zeros. A customer can have 0 or no r

dataset.drop(['num_accounts_120d_past_due','num_accounts_30d_past_due'], axi

## i am dropping column num_active_debit_accounts as this column will not af
## rather the column total debit limit is what is important for our model as

dataset.drop(['num_active_debit_accounts'], axis=1, inplace=True)

### the column current_accounts_delinq have 9999 zero values hence dropping
dataset.drop(['current_accounts_delinq'], axis=1, inplace=True)

## i will calculate the percent number cc carrying balance accounts by divid
## cc account and multiplying by 100
## as percentage of number of cc carrying balance is important rather than c

dataset['percent_num_cc_carrying_balance'] = (dataset['num_cc_carrying_balar

## dropping column num_cc_carrying_balance, num_total_cc_accounts, num_open_
dataset.drop(['num_cc_carrying_balance','num_total_cc_accounts','num_open_cc

## the columns stating balance, principal or interest paid are the columns w
## as we are developing a model that will predict the interest rate based or

dataset.drop(['balance','paid_total','paid_principal','paid_interest','paid_
dataset.drop(['installment','grade','sub_grade'], axis=1, inplace=True)

## i will change the months to year for column 'months_since_last_delinq' as

dataset['months_since_last_delinq'].fillna(0, inplace=True)
dataset['year_since_last_delinq'] = (dataset['months_since_last_delinq'] / 1
dataset.drop(['months_since_last_delinq'], axis=1, inplace=True)

## i will change the months to year for column 'months_since_last_credit_inq

dataset['months_since_last_credit_inquiry'].fillna(0, inplace=True)
dataset['year_since_last_credit_inquiry'] = (dataset['months_since_last_crec
dataset.drop(['months_since_last_credit_inquiry'], axis=1, inplace=True)

dataset['percentage_credit_utilized'].fillna(0, inplace=True)
```

```
the zeros in column num_accounts_120d_past_due are: 9682
the zeros in column num_accounts_30d_past_due: 9999
```

In [6]:
```
## the below bar plot helps to understand the loan purpose and number of loc
## it gave overall idea that debt_consolidation type of loan was taken more
viz_loan_purpose = dataset['loan_purpose'].value_counts().to_frame().reset_i
viz_loan_purpose.columns = ['Loan Purpose', 'Count']
plt.subplots(figsize=(18,8))
sb.barplot(y='Count', x='Loan Purpose', data=viz_loan_purpose)
plt.ylabel('Count')
plt.title('Distribution of Loan Purpose')
plt.show()
```



In [7]:
```
dataset.shape
```

Out[7]:
```
(10000, 29)
```

In [8]:
```
### we will now see our categorical variables
dataset['verified_income'].value_counts()
```

Out[8]:
```
Source Verified    4116
Not Verified       3594
Verified           2290
Name: verified_income, dtype: int64
```

In [9]:
```
## by using label encoder i will convert them to 0,1,2
label_encoder = LabelEncoder()
dataset['verified_income'] = label_encoder.fit_transform(dataset['verified_i
dataset['verified_income'].value_counts()
```

Out[9]:
```
1    4116
0    3594
2    2290
Name: verified_income, dtype: int64
```

In [10]:
```
label_encoder = LabelEncoder()
dataset['homeownership'] = label_encoder.fit_transform(dataset['homeownershi
dataset['homeownership'].value_counts()
```

Out[10]:
```
0    4789
2    3858
1    1353
Name: homeownership, dtype: int64
```

In [11]:
```
label_encoder = LabelEncoder()
dataset['loan_purpose'] = label_encoder.fit_transform(dataset['loan_purpose'
dataset['loan_purpose'].value_counts()
```

Out[11]:
```
2     5144
1     2249
8      914
3      680
5      303
6      162
4      151
0      131
10     125
7       69
11      62
9       10
Name: loan_purpose, dtype: int64
```

In [12]:
```
dataset.info()
```

```
num_satisfactory_accounts           10000 non-null int64
total_debit_limit                   10000 non-null int64
num_mort_accounts                   10000 non-null int64
account_never_delinq_percent        10000 non-null float64
tax_liens                           10000 non-null int64
public_record_bankrupt              10000 non-null int64
loan_purpose                        10000 non-null int32
loan_amount                         10000 non-null int64
term                                10000 non-null int64
interest_rate                       10000 non-null float64
total_annual_income                 10000 non-null float64
earliest_credit_line_age            10000 non-null int64
percentage_credit_utilized          10000 non-null float64
year_since_90dlate                  10000 non-null float64
percent_num_cc_carrying_balance     10000 non-null float64
year_since_last_delinq              10000 non-null float64
year_since_last_credit_inquiry      10000 non-null float64
dtypes: float64(10), int32(3), int64(16)
memory usage: 2.1 MB
```

In [13]:
```
## it a very useful data visualization tool tells us the minimum and maximum
dataset.describe()
```

Out[13]:

	homeownership	verified_income	debt_to_income	debt_to_income_joint	delinq_2y	ir
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	
mean	0.906900	0.869600	19.261852	2.986906	0.21600	
std	0.925266	0.755945	15.016585	7.775200	0.68366	
min	0.000000	0.000000	0.000000	0.000000	0.00000	
25%	0.000000	0.000000	10.997500	0.000000	0.00000	
50%	1.000000	1.000000	17.530000	0.000000	0.00000	
75%	2.000000	1.000000	24.990000	0.000000	0.00000	
max	2.000000	2.000000	469.090000	39.980000	13.00000	

8 rows × 29 columns

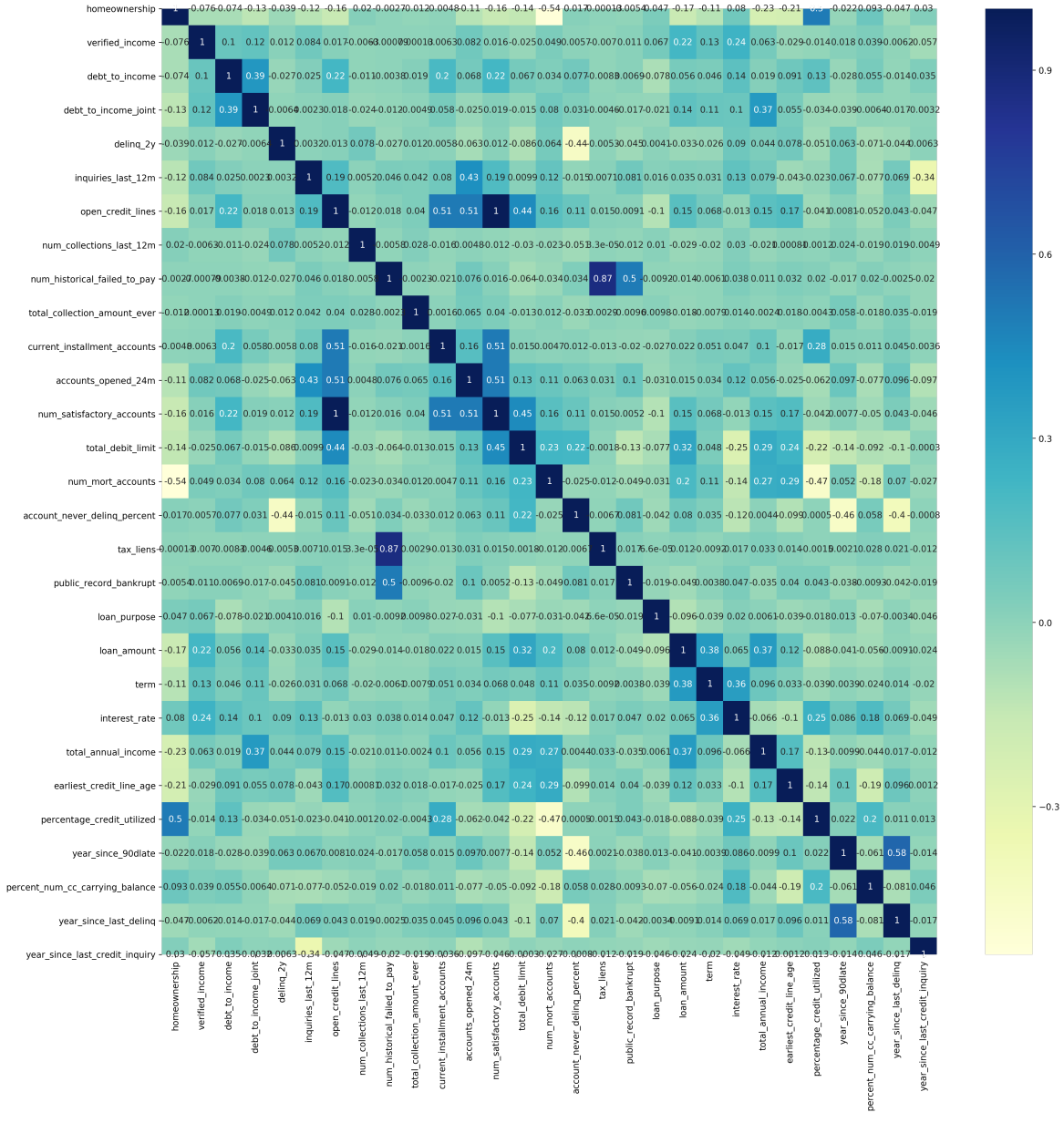
In [17]:

```
## the correlation matrix plot is very important to visualize the correlatic
## 1 tells us that there positive correlation and -1 negative correlation. i
## to avoid biased predictions.
## from the below correlation plot i  am going to remove the column num_stis

print("  Correlation matrix plot")
dataplot = sb.heatmap(dataset.corr(), cmap="YlGnBu", annot=True)
plt.show
```

```
  Correlation matrix plot
```

Out[17]:

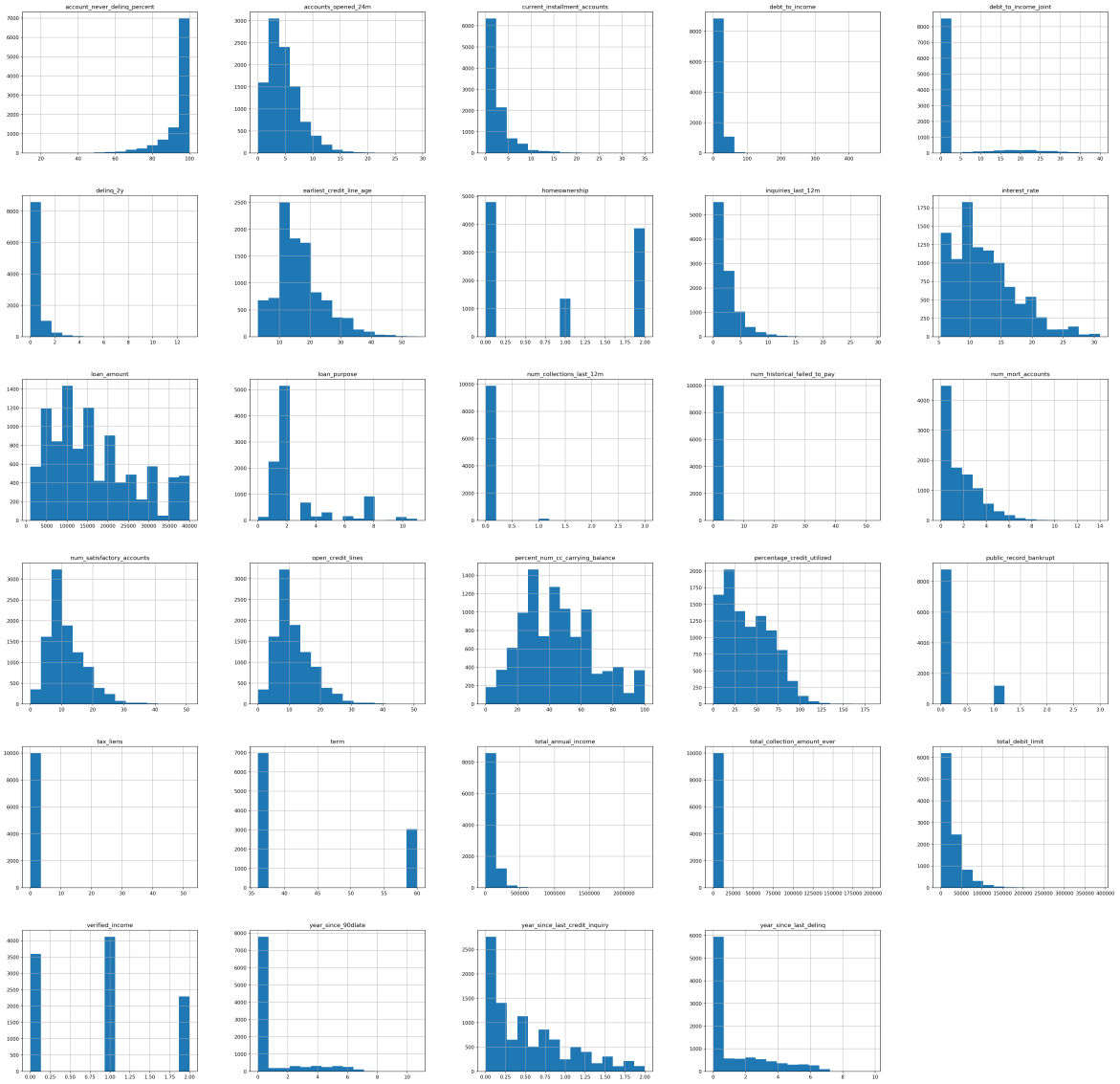
```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [18]:

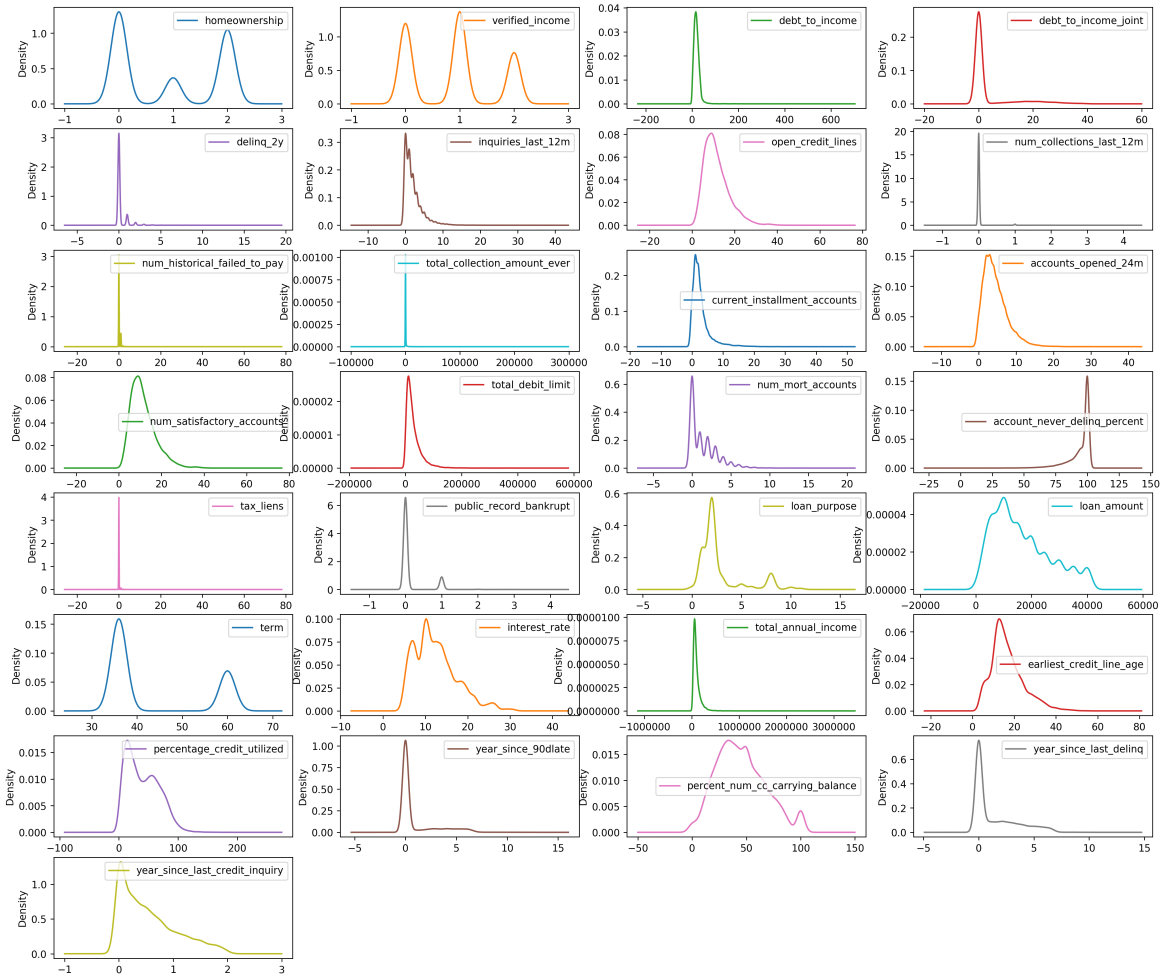
```
## histograms are frequency plots which shows data distribution.

print("  Histogram")
dataset.hist(figsize=(40,40), bins=15)
plt.show()
```

```
  Histogram
```



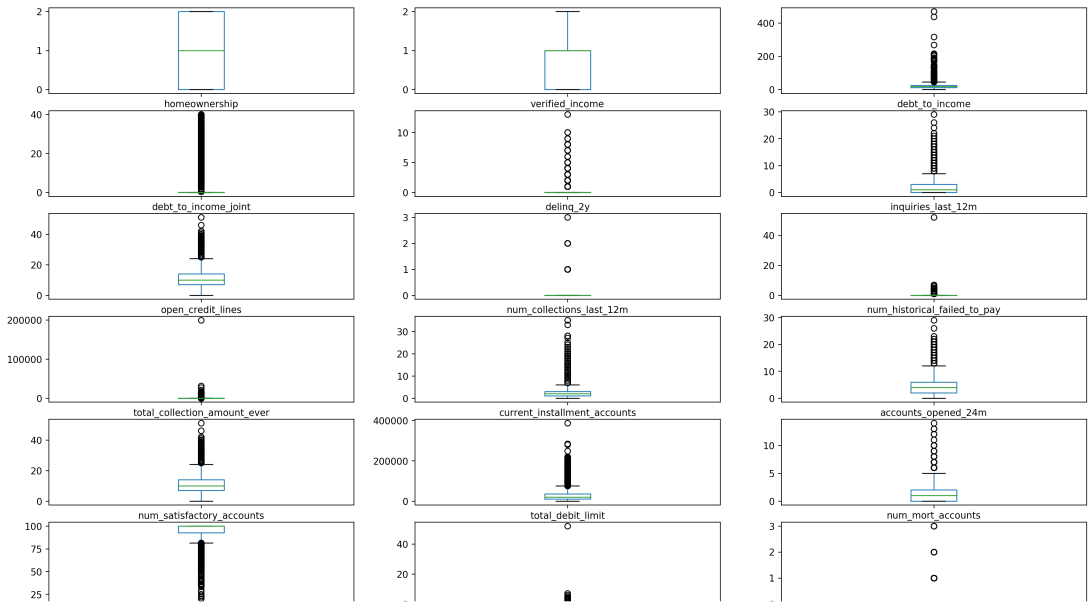
In [19]:
```
## density plots are effective against histogram when it comes to visualizir
print("  Density plot")
dataset.plot(kind='density', subplots=True, layout=(9,4), sharex=False, figs
pyplot.show()
```

```
  Density plot
```



In [20]:
```
print("  Box and whisker plot")
dataset.plot(kind = 'box', subplots = 'true', layout = (10,3))
plt.rcParams['figure.figsize']=[20,20]
plt.rcParams['figure.dpi']= 250
plt.show()
```

```
  Box and whisker plot
```



In [21]:
```
## dropping the highly correlated column
dataset.drop(['num_satisfactory_accounts'],axis=1, inplace=True)
```

In [22]:
```
dataset_X = dataset.drop(['interest_rate'], axis=1)
dataset_Y = dataset['interest_rate']
```

In [25]:
```
dataset_std = StandardScaler().fit_transform(dataset)
pca = PCA(n_components = 25)
pcs = pca.fit_transform(dataset_std)
print('the explained variance captured by dataset:',np.cumsum(pca.explained_
```

```
the explained variance captured by dataset: [0.10902508 0.18878943 0.267736
36 0.33928579 0.40197401 0.45619048
 0.50101678 0.54344547 0.58358572 0.6205607  0.65684396 0.6924088
 0.7265881  0.75881478 0.79026878 0.82011815 0.8480109  0.87110845
 0.89189089 0.9083333  0.92421301 0.93978626 0.95412263 0.96724501
 0.98035489]
```

In [27]:
```
columns = []
for i in range(25):
     columns.append(i+1)
df = pd.DataFrame(data=pcs, columns=columns)
df['ROI'] = dataset['interest_rate']
df.head()
```

Out[27]:

	1	2	3	4	5	6	7	8	
0	0.674424	1.931762	0.438492	-0.682014	0.823774	-1.870873	-1.018128	-0.473623	0.83
1	-0.811495	-0.595153	0.739046	2.360534	-1.825887	-0.545638	0.323601	-0.170547	-1.18
2	-1.764159	1.696450	1.151768	-0.508283	-1.630450	-1.429719	0.422013	-0.668450	0.90
3	-2.202997	-1.852907	-0.475481	2.078874	0.556598	1.065965	-0.897710	0.057125	0.00
4	0.542260	-0.684653	3.148243	-2.304930	1.311049	0.034270	2.466293	-1.480834	1.53

5 rows × 26 columns

In [28]:
```
df_X = df.drop(['ROI'], axis=1).values
df_Y = df['ROI'].values
```

In [29]:
```
df_X.shape
```

Out[29]: (10000, 25)

In [30]:
```
df_Y.shape
```

Out[30]: (10000,)

In [31]:
```
x_train, x_test, y_train, y_test = train_test_split(df_X, df_Y, test_size=0.
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
rf.fit(x_train, y_train)
predictions = rf.predict(x_test)
errors = abs(predictions - y_test)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
mape = 100 * (errors / y_test)
# Calculate and display accuracy. i tried to calculate overall how much accu
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

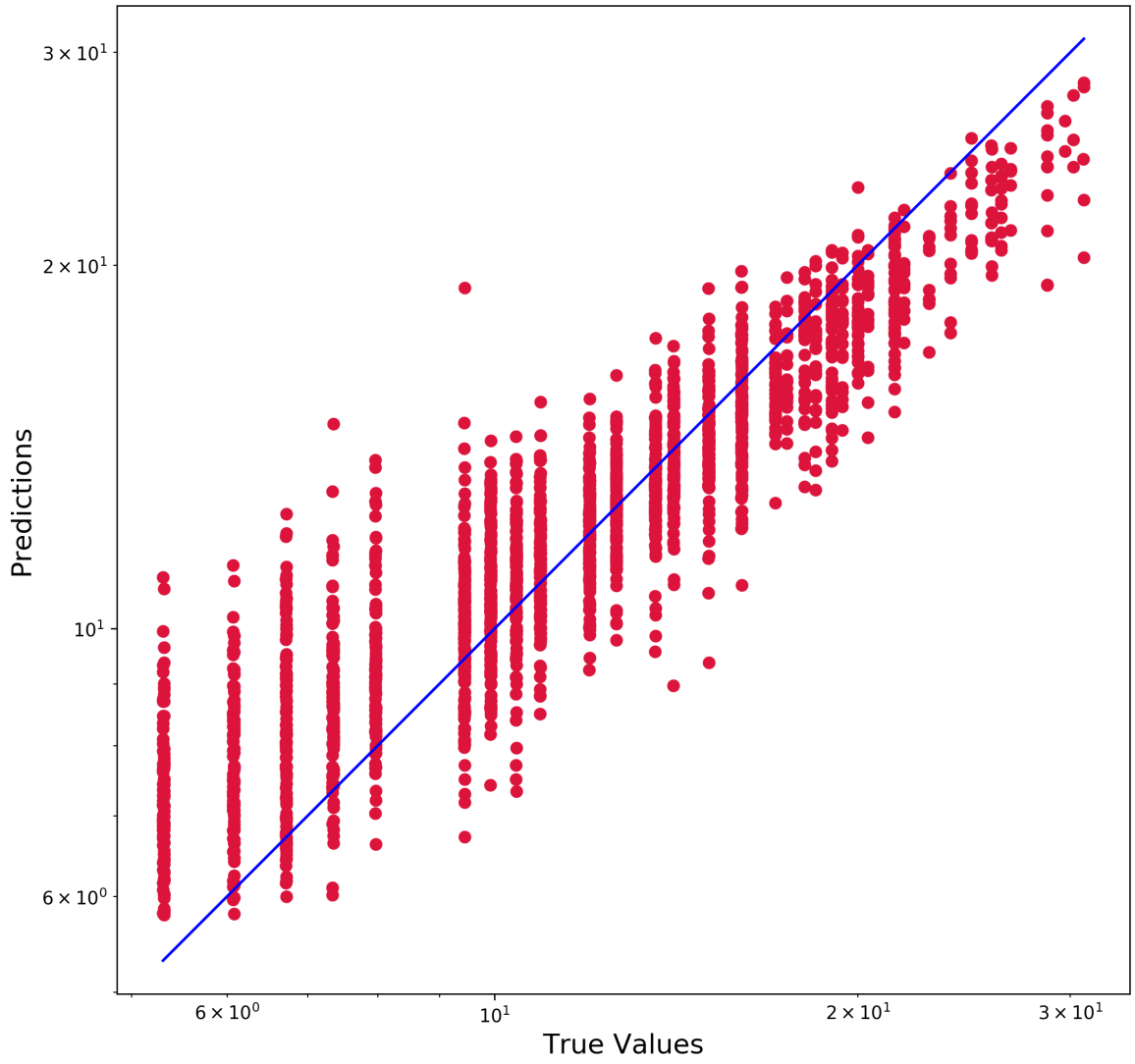
```
Mean Absolute Error: 1.54 degrees.
Accuracy: 85.67 %.
```

In [32]:
```
##the r2 is measure of correlation between two variables
## the result 0.84 shows that there is a good relationship, and we can use p
R_square = sm.r2_score(y_test,predictions)
```

In [33]:
```
R_square
```

Out[33]:
```
0.8464272143138855
```

In [34]:
```
plt.figure(figsize=(10,10))
plt.scatter(y_test, predictions, c='crimson')
plt.yscale('log')
plt.xscale('log')

p1 = max(max(predictions), max(y_test))
p2 = min(min(predictions), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```



In [ ]:
```

```