LOAN APPROVAL PREDICITION BY USING MACHINE LEARNING

A Mini Project Report

Submitted by:

Sainath Reddy

Project Abstract:

With the increase in banking sector many people are applying for loans in bank. All these loans are not approvable. The main income of bank assets comes from gain earned from loans. The main objective of banks is to invest their assets in safe customers. Today many banks approve loan after many process of verification and validation but still there is no surety that selected customer is safe or not. Therefore it is important to apply various techniques in banking sector for selecting a customer who pays loan on time.As this dataset goes for only the visualization part

DATA INTRODUCTION

Loan Distribution is the main business part of many banks. The main portion of banks income comes from the loan distributed to customers. These banks apply interest on loan which are distributed to customers.

The main objective of banks is to invest their assets in safe customers. Up to now many banks are processing loans after regress process of verification and validation. But till now no bank can give surety that the customer who is chosen for loan application is safe or not. So to avoid this situation we introduced a system for the approval of bank loans known as Loan Prediction System Using Python.

Loan Prediction System is a software which checks the eligibility of a particular customer who is capable of paying loan or not. This system checks various parameters such as customer's martial status, income, expenditure and various factors. This process is applied for many customers of trained data set. By considering these factors a required model is built. This model is applied on the test data set for getting required output. The output generated will be in the form of yes or no. Yes indicates that a particular customer is capable of paying loan and no indicates that the particular customer is not capable of paying loan. Based on these factors we can approve loans for customers.

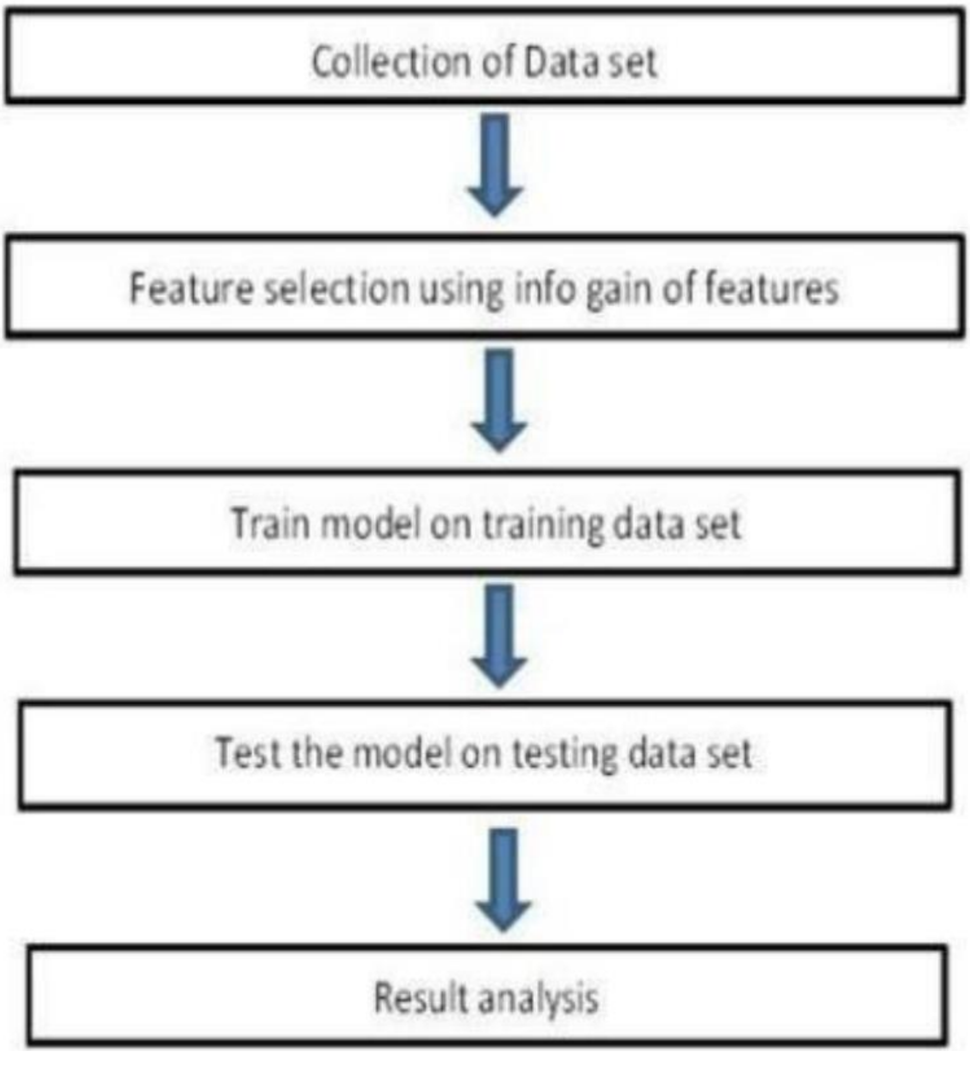
Till now loans are processed by various banks through pen and paperwork. When the large no of customers' apply for bank loan these bank take lot of time to approve their loan. After approval of loan by the banks, there is no surety that the chosen applicant is capable of paying loan or not. Many banks use their own software's for the loan approval. In existing system we use data mining algorithms for the loan approval; this is the old technique for the approval of loan. Mutiple data sets are combined and form a Generalised datasets, and different machine learning algorithms are applied to generate results. But these techniques are not up to the mark. Due to this huge banks are suffering from financial crises. To resolve this issue we introduce a new way for approval of loans.

This project covers the whole process from problem statement to model development and evaluation:

1. Implementation
2. Hypothesis Generation
3. Data Collection
4. Exploratory Data Analysis (EDA)
5. Data Pre-processing

IMPLEMENTATION

A method which consists of many no. of weak classifiers. We upload our training dataset in these classifiers to obtain result. While providing the output these classifiers considering various factors build a model. Each classifier may build a model according to given training dataset.



Flowchart Diagram of Loan Approval Prediction System

Hypothesis Generation

Hypothesis Generation is the process of listing out all the possible factors that can affect the outcome i.e. which of the features will have an impact on whether a loan will be approved or not. Some of the hypothesis are:

- Education - Applicants with higher education level i.e. graduate level should have higher chances of loan approval
- Income: Applicants with higher income should have more chances of loan approval
- Loan amount: If the loan amount is less, the chances of loan approval should be high
- Loan term: Loans with shorter time period should have higher chances of approval
- Previous credit history: Applicants who have repayed their previous debts should have higher chances of loan approval
- Monthly installment amount: If the monthly installment amount is low, the chances of loan approval should be high
- And so on

Some of the hypothesis seem intuitive while others may not. We will try to validate each of these hypothesis based on the dataset.

Data Collection

The data have already been provided by GitHub.

The training set will be used for training the model, i.e. our model will learn from this data. It contains all the independent variables and the target variable. The test set contains all the independent variables, but not the target variable. We will apply the model to predict the target variable for the test data. There are 13 columns of features and 614 rows of records in the training set and 12 columns of features and 367 rows of records in the test set. The dataset variables are summarized as below:

No	Variable	Type	Description
1	Loan_ID	Numerical - Discrete	Unique Loan ID
2	Gender	Categorical - Nominal	Male / Female
3	Married	Categorical - Nominal	Applicant married (Y/N)
4	Dependents	Categorical - Ordinal	Number of dependents (0, 1, 2, 3+)
5	Education	Categorical - Nominal	Applicant Education (Graduate / Under Graduate)
6	Self_Employed	Categorical - Nominal	Self employed (Y/N)
7	ApplicantIncome	Numerical - Continuous	Applicant income
8	CoapplicantIncome	Numerical - Continuous	Coapplicant income
9	LoanAmount	Numerical - Continuous	Loan amount in thousands
10	Loan_Amount_Term	Numerical - Discrete	Term of loan in months
11	Credit_History	Categorical - Nominal	credit history meets guidelines (0, 1)
12	Property_Area	Categorical - Ordinal	Urban / Semi Urban / Rural
13	Loan_Status	Categorical - Nominal	Loan approved (Y/N)

Exploratory Data Analysis(EDA)

In [2]:

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Test Dataset

In [22]:

```
#load the test dataset
a=pd.read_csv("test.csv")
a
```

Out[22]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001015	Male	Yes	0	Graduate	No	5720
1	LP001022	Male	Yes	1	Graduate	No	3076
2	LP001031	Male	Yes	2	Graduate	No	5000
3	LP001035	Male	Yes	2	Graduate	No	2340
4	LP001051	Male	No	0	Not Graduate	No	3276
...	...	...	...	...	...	...	...
362	LP002971	Male	Yes	3+	Not Graduate	Yes	4009
363	LP002975	Male	Yes	0	Graduate	No	4158
364	LP002980	Male	No	0	Graduate	No	3250
365	LP002986	Male	Yes	0	Graduate	No	5000
366	LP002989	Male	No	0	Graduate	Yes	9200

367 rows × 12 columns

In [ ]:

```
a.describe
```

Out[3]:

```
<bound method NDFrame.describe of       Loan_ID Gender Married Dependents
Education Self_Employed  \
0    LP001015   Male     Yes          0      Graduate            No
1    LP001022   Male     Yes          1      Graduate            No
2    LP001031   Male     Yes          2      Graduate            No
3    LP001035   Male     Yes          2      Graduate            No
4    LP001051   Male      No          0  Not Graduate            No
..        ...    ...     ...        ...           ...           ...
362  LP002971   Male     Yes         3+  Not Graduate           Yes
363  LP002975   Male     Yes          0      Graduate            No
364  LP002980   Male      No          0      Graduate            No
365  LP002986   Male     Yes          0      Graduate            No
366  LP002989   Male      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5720                  0       110.0             360.0
1               3076               1500       126.0             360.0
2               5000               1800       208.0             360.0
3               2340               2546       100.0             360.0
4               3276                  0        78.0             360.0
..               ...                ...         ...               ...
362             4009               1777       113.0             360.0
363             4158                709       115.0             360.0
364             3250               1993       126.0             360.0
365             5000               2393       158.0             360.0
366             9200                  0        98.0             180.0

     Credit_History Property_Area
0               1.0         Urban
1               1.0         Urban
2               1.0         Urban
3               NaN         Urban
4               1.0         Urban
..              ...           ...
362             1.0         Urban
363             1.0         Urban
364             NaN     Semiurban
365             1.0         Rural
366             1.0         Rural

[367 rows x 12 columns]>
```

In [ ]:

```
# show the shape of the dataset i.e. no of rows, no of columns
a.shape
```

Out[4]:

```
(367, 12)
```

In [ ]:

```
a_length=len(a)
a_length
```

Out[6]:

```
367
```

In [ ]:

```
# take a look at the features (i.e. independent variables) in the dataset
a.columns
```

Out[7]:

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmou
nt',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
```

In [ ]:

```
# show the data types for each column of the test set
a.dtypes
```

Out[9]:

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome      int64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
dtype: object
```

In [ ]:

```
# concise summary of the dataset, info about index dtype, column dtypes, non-null values
a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            367 non-null    object
 1   Gender             356 non-null    object
 2   Married            367 non-null    object
 3   Dependents         357 non-null    object
 4   Education          367 non-null    object
 5   Self_Employed      344 non-null    object
 6   ApplicantIncome    367 non-null    int64
 7   CoapplicantIncome  367 non-null    int64
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

In [35]:

```
a.mean()
```

```
C:\Users\meruv\AppData\Local\Temp\ipykernel_16028\1798845826.py:1: Future
Warning: The default value of numeric_only in DataFrame.mean is deprecate
d. In a future version, it will default to False. In addition, specifying
'numeric_only=None' is deprecated. Select only valid columns or specify t
he value of numeric_only to silence this warning.
  a.mean()
```

Out[35]:

```
ApplicantIncome      4805.599455
CoapplicantIncome    1569.577657
LoanAmount            136.132597
Loan_Amount_Term      342.537396
Credit_History          0.825444
dtype: float64
```

In [36]:

```
a.std()
```

```
C:\Users\meruv\AppData\Local\Temp\ipykernel_16028\2800935211.py:1: Future
Warning: The default value of numeric_only in DataFrame.std is deprecate
d. In a future version, it will default to False. In addition, specifying
'numeric_only=None' is deprecated. Select only valid columns or specify t
he value of numeric_only to silence this warning.
  a.std()
```

Out[36]:

```
ApplicantIncome      4910.685399
CoapplicantIncome    2334.232099
LoanAmount             61.366652
Loan_Amount_Term       65.156643
Credit_History          0.380150
dtype: float64
```

Independent Variable(Categorical)

frequency table of a variable will give us the count of each category in that variable

In [ ]:

```
a['Married'].value_counts()
```

Out[11]:

```
Yes    233
No     134
Name: Married, dtype: int64
```

In [ ]:

```
a['Gender'].value_counts()
```

Out[12]:

```
Male      286
Female     70
Name: Gender, dtype: int64
```

In [ ]:

```
a['Dependents'].value_counts()
```

Out[13]:

```
0     200
2      59
1      58
3+     40
Name: Dependents, dtype: int64
```

In [ ]:

```
a['Self_Employed'].value_counts()
```

Out[14]:

```
No     307
Yes     37
Name: Self_Employed, dtype: int64
```

In [ ]:

```
a['Loan_Amount_Term'].value_counts()
```

Out[15]:

```
360.0    311
180.0     22
480.0      8
300.0      7
240.0      4
84.0       3
60.0       1
12.0       1
350.0      1
36.0       1
120.0      1
6.0        1
Name: Loan_Amount_Term, dtype: int64
```

In [ ]:

```
a['Credit_History'].value_counts()
```

Out[16]:

```
1.0    279
0.0     59
Name: Credit_History, dtype: int64
```

Data Pre-processing

Missing value and outlier treatment

In [12]:

```
# check for missing values
a.apply(lambda x:sum(x.isnull()),axis=0)
```

Out[12]:

```
Loan_ID              0
Gender              11
Married              0
Dependents          10
Education            0
Self_Employed       23
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           5
Loan_Amount_Term     6
Credit_History      29
Property_Area        0
dtype: int64
```

There are missing values in Gender, Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_History features. We will treat the missing values in all the features one by one.

- For numerical variables: imputation using mean or median
- For categorical variables: imputation using mode

There are very less missing values in Gender, Married, Dependents, Credit_History and Self_Employed features so we can fill them using the mode of the features. If an independent variable in our dataset has huge amount of missing data e.g. 80% missing values in it, then we would drop the variable from the dataset.

In [ ]:

```
# replace missing values with the mean ,max and others
a['LoanAmount'].fillna(a['LoanAmount'].mean(), inplace=True)
a['Loan_Amount_Term'].fillna(a['Loan_Amount_Term'].mean(), inplace=True)
a['Credit_History'].fillna(a['Credit_History'].max(), inplace=True)
a['Gender'].fillna('Female',inplace=True)
a['Married'].fillna('Yes',inplace=True)
a['Dependents'].fillna('0',inplace=True)
a['Self_Employed'].fillna('No',inplace=True)
```

In [ ]:

```
a.head()
```

Out[18]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Co
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	

In [ ]:

```
# check whether all the missing values are filled in the test dataset
a.apply(lambda x:sum(x.isnull()),axis=0)
```

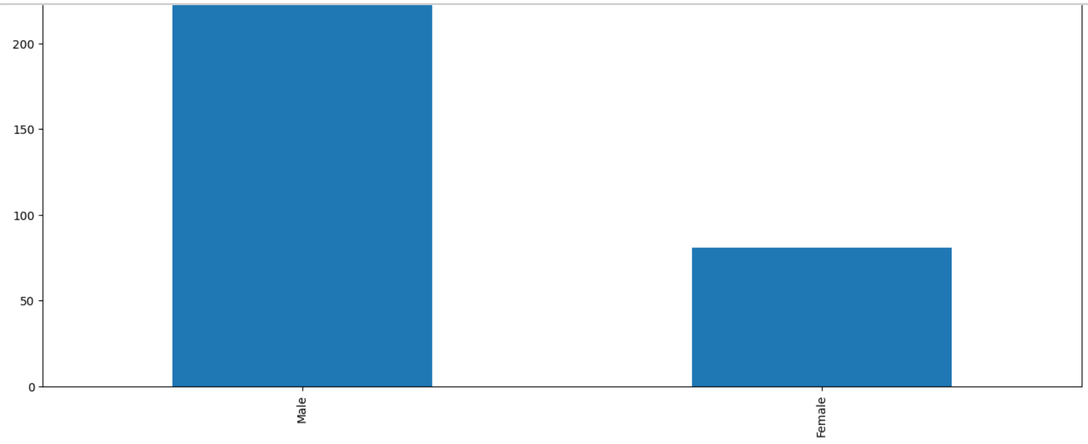
Out[19]:

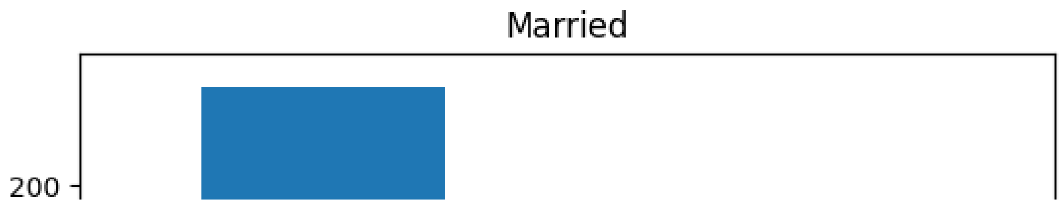
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
dtype: int64
```

Note:-We need to replace the missing values in Test set using the mode/median/mean of the Training set, not from the Test set. Likewise, if you remove values above some threshold in the test case, make sure that the threshold is derived from the training and not test set. Make sure to calculate the mean (or any other metrics) only on the train data to avoid data leakage to your test set

In [ ]:

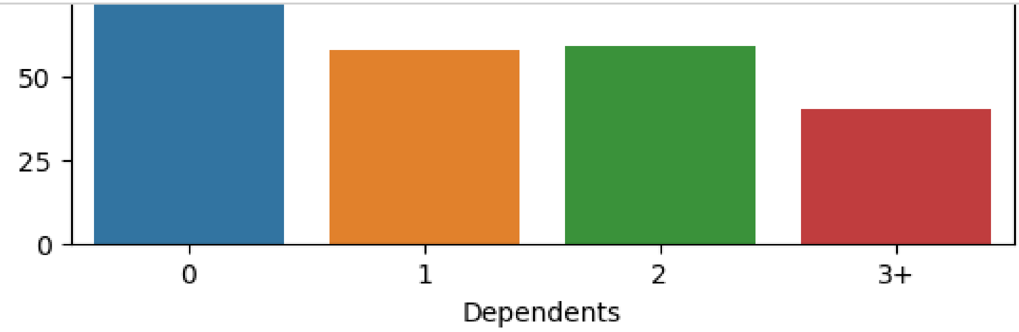
```
# Visualizing categorical features
# plt.figure(1)
a['Gender'].value_counts().plot.bar(figsize=(16,8),title="Gender")
plt.show()
a['Married'].value_counts().plot.bar(title="Married")
plt.show()
a['Self_Employed'].value_counts().plot.bar(title="Self_employed")
plt.show()
a['Credit_History'].value_counts().plot.bar(title="Credit_History")
plt.show()
```

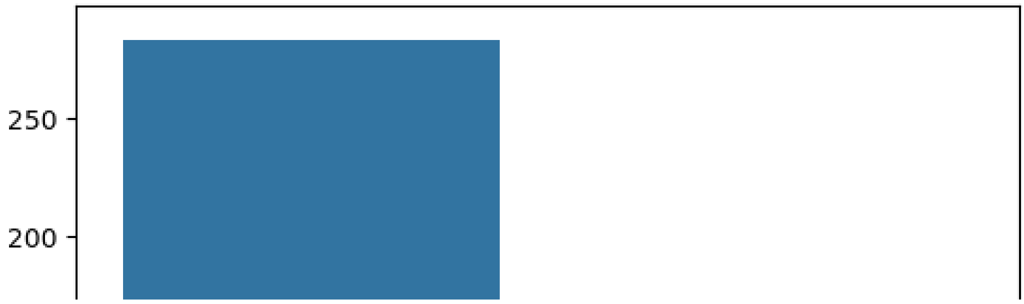




In [23]:

```
sns.countplot(x=a['Gender'])
plt.show()
sns.countplot(x=a['Dependents'])
plt.show()
sns.countplot(x=a['Education'])
plt.show()
sns.countplot(x=a['Credit_History'])
plt.show()
sns.countplot(x=a['Property_Area'])
```



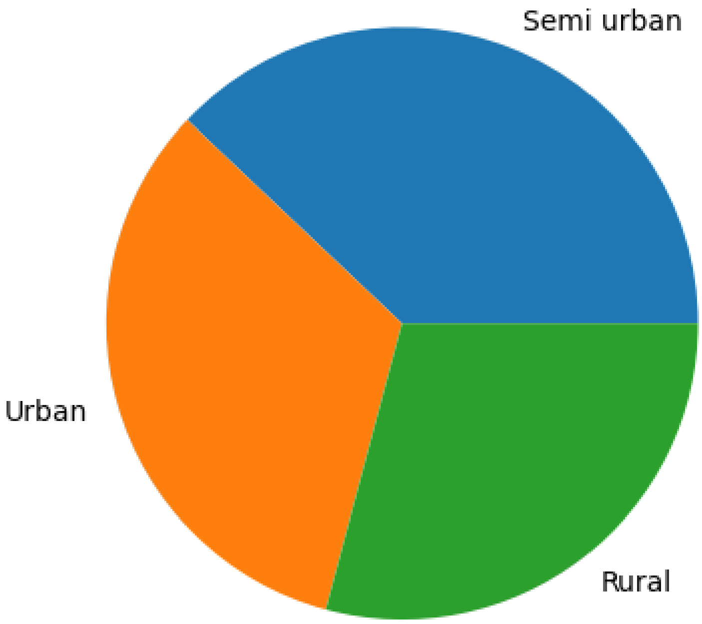


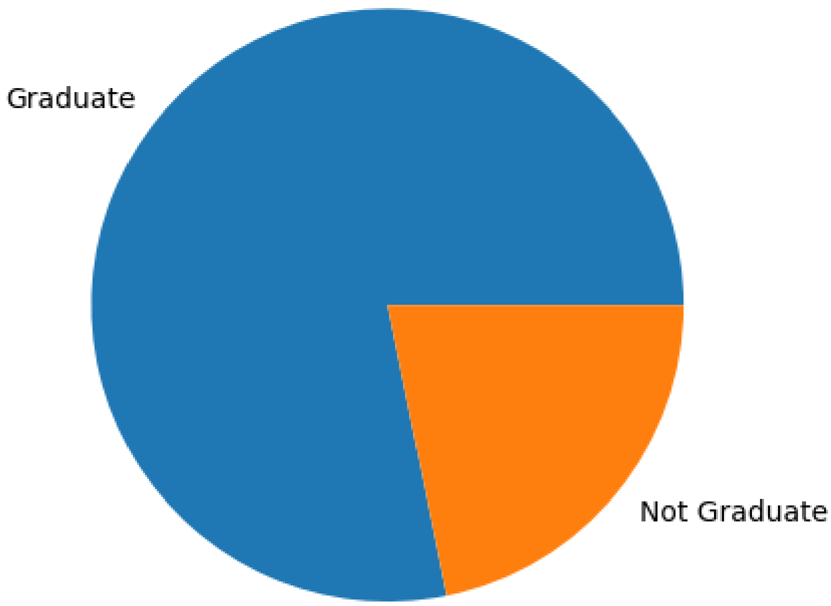
It can be inferred from the above bar plots that:

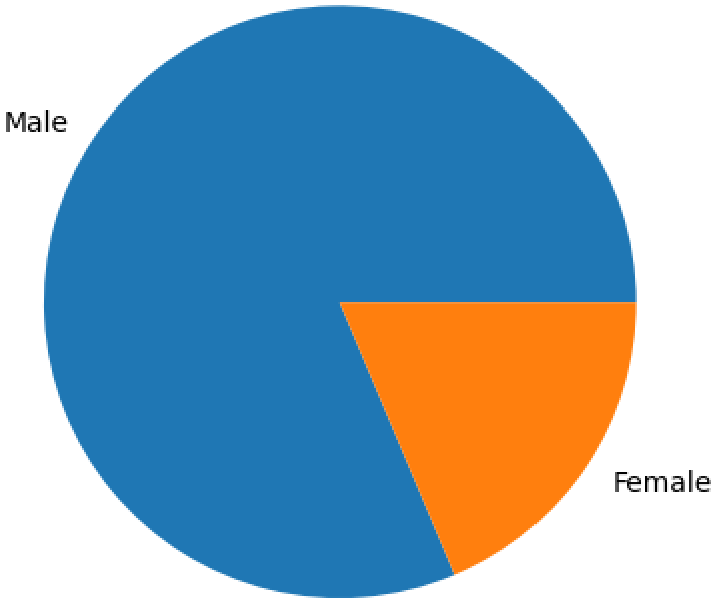
- 80% applicants in the dataset are male.
- Around 65% of the applicants in the dataset are married.
- Around 15% applicants in the dataset are self employed.
- Around 85% applicants have credit history (repaid their debts).
- Around 80% of the applicants are Graduate.

In [31]:

```
plt.pie(b.Property_Area.value_counts(),[0,0,0],labels=['Semi urban','Urban','Rural'])
plt.show()
plt.pie(b.Education.value_counts(),[0,0],labels=['Graduate','Not Graduate'])
plt.show()
plt.pie(b.Gender.value_counts(),[0,0],labels=['Male','Female'])
plt.show()
plt.pie(b.Self_Employed.value_counts(),[0,0],labels=['No','Yes'])
```

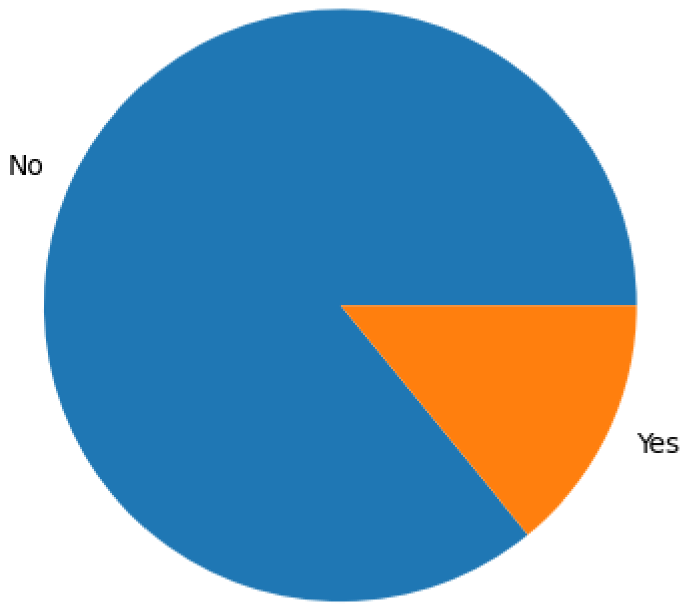






Out[31]:

```
([<matplotlib.patches.Wedge at 0x12e2f6a05e0>,
  <matplotlib.patches.Wedge at 0x12e2f6a04f0>],
 [Text(-0.9939912136472331, 0.47114909231802693, 'No'),
  Text(0.9939912357033128, -0.47114904578593964, 'Yes')])
```

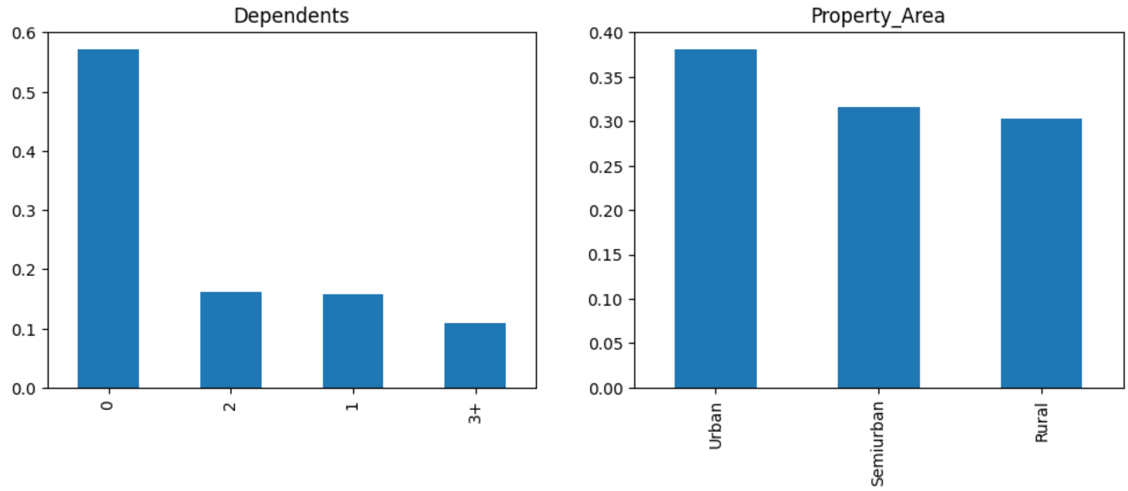


In [ ]:

```
plt.subplot(121)
a['Dependents'].value_counts(normalize=True).plot.bar(figsize=(12,4), title= 'Dependents

plt.subplot(122)
a['Property_Area'].value_counts(normalize=True).plot.bar(title= 'Property_Area')

plt.show()
```



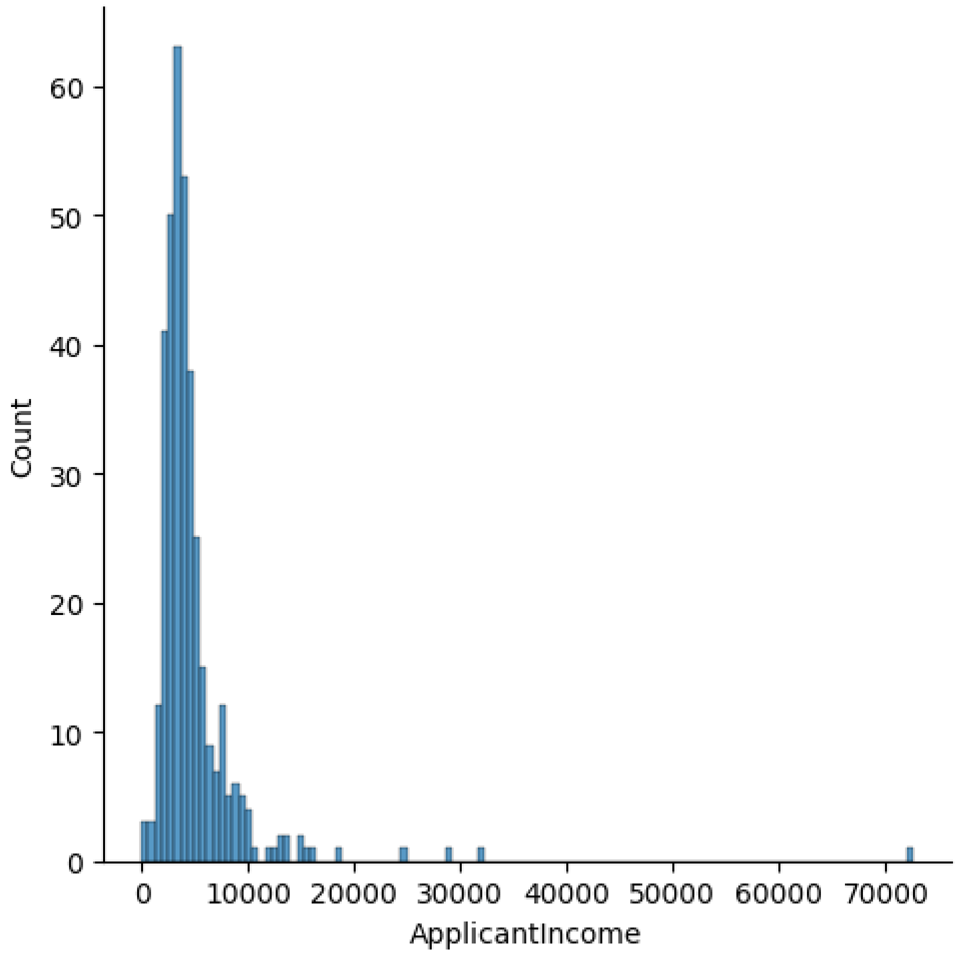
- More than half of the applicants don’t have any dependents.
- Most of the applicants are from Semiurban area.

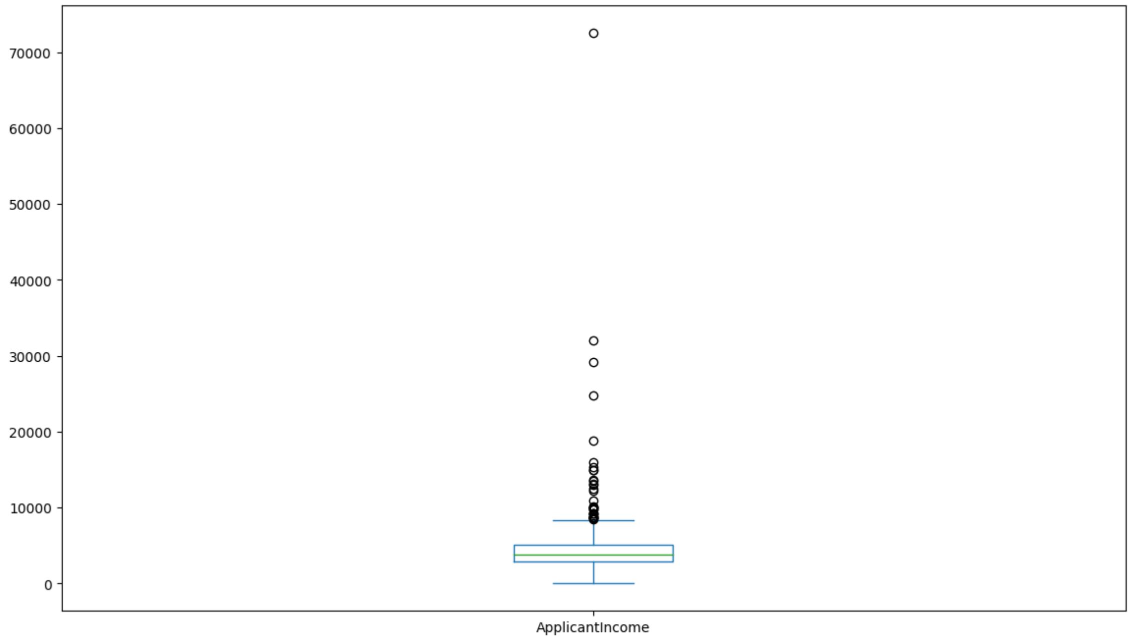
Independent Variable (Numerical)

There are 4 features that are Numerical: These features have numerical values (ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term)

In [ ]:

```
sns.displot(a['ApplicantIncome']);
plt.show()
a['ApplicantIncome'].plot.box(figsize=(14,8))
plt.show()
```





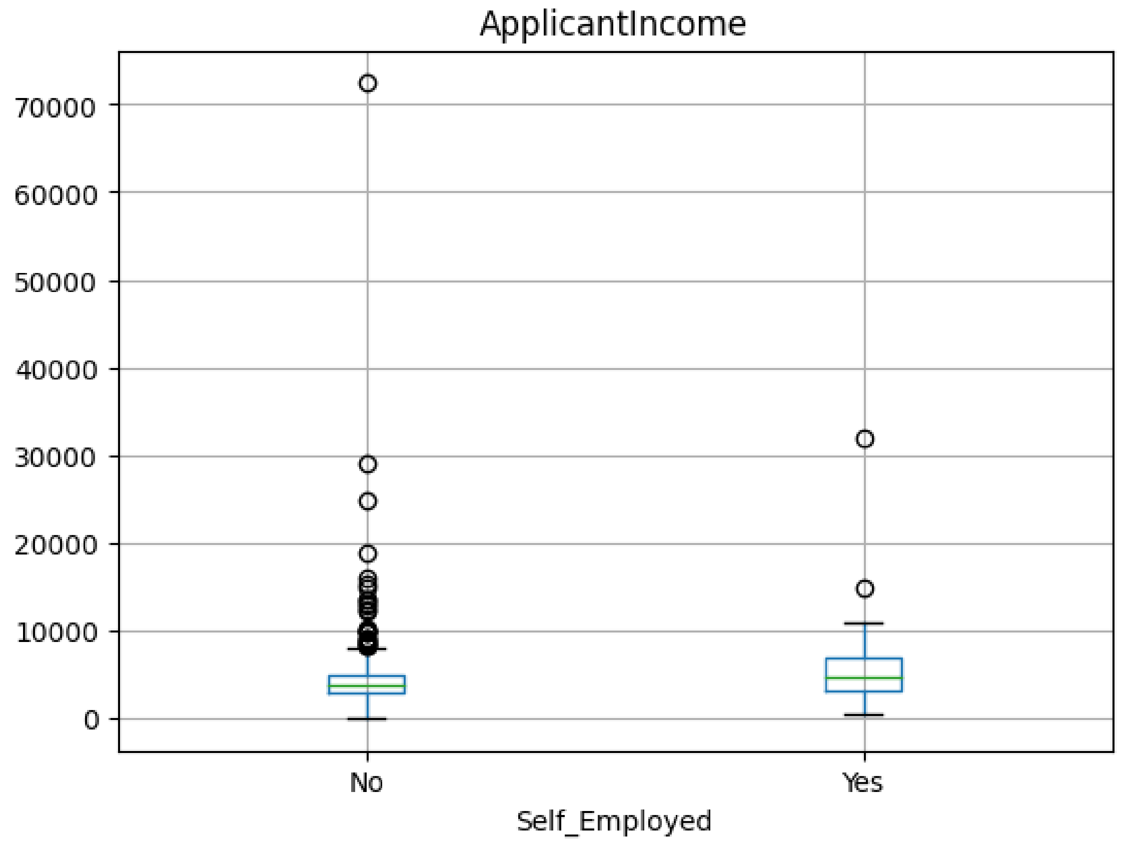
It can be inferred that most of the data in the distribution of applicant income is towards left which means it is not normally distributed. The distribution is right-skewed (positive skewness). We will try to make it normal in later sections as algorithms works better if the data is normally distributed.

In [ ]:

```
a.boxplot(column='ApplicantIncome',by='Self_Employed')
plt.suptitle("")
```

Out[28]:

```
Text(0.5, 0.98, '')
```

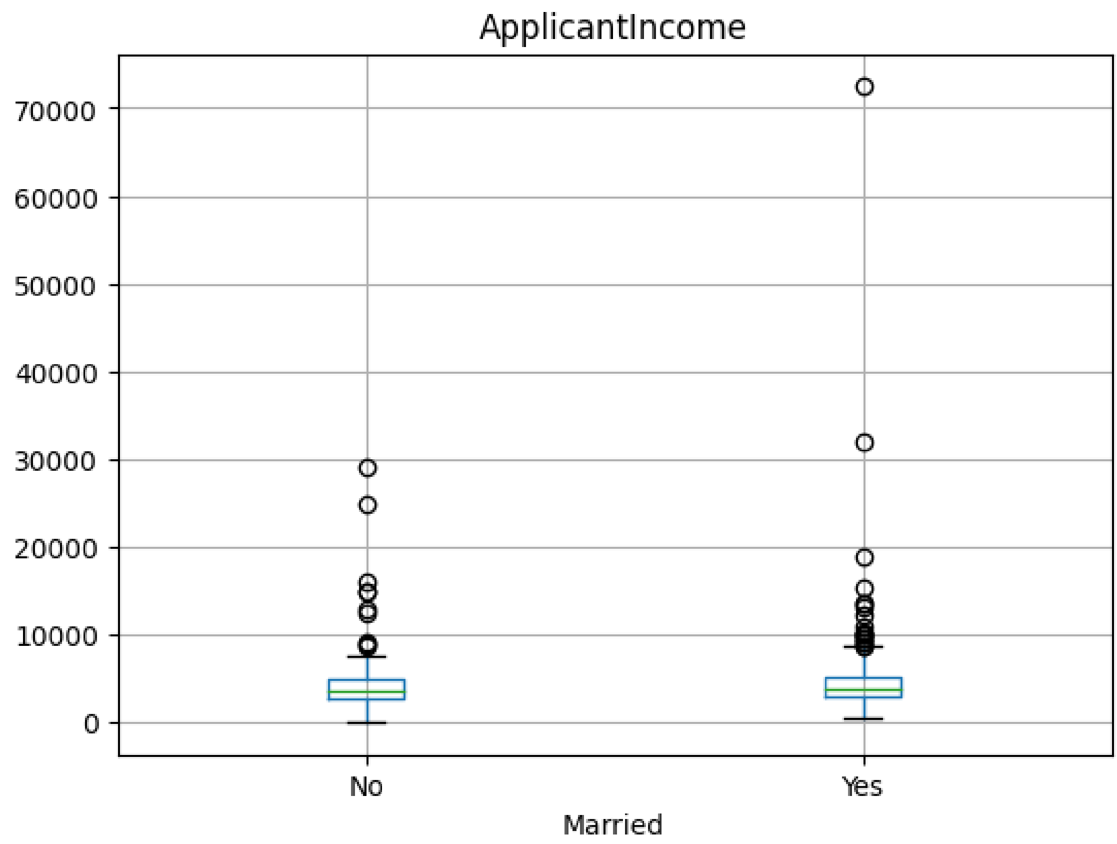


In [ ]:

```
a.boxplot(column='ApplicantIncome',by='Married')
plt.suptitle("")
```

Out[29]:

```
Text(0.5, 0.98, '')
```



The boxplot confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society. Part of this can be driven by the fact that we are looking at people with different education levels. Let us segregate them by Education and Married:

In [ ]:

```
plt.subplot(121)
sns.distplot(a['CoapplicantIncome']);

plt.subplot(122)
a['CoapplicantIncome'].plot.box(figsize=(16,5))

plt.show()
```

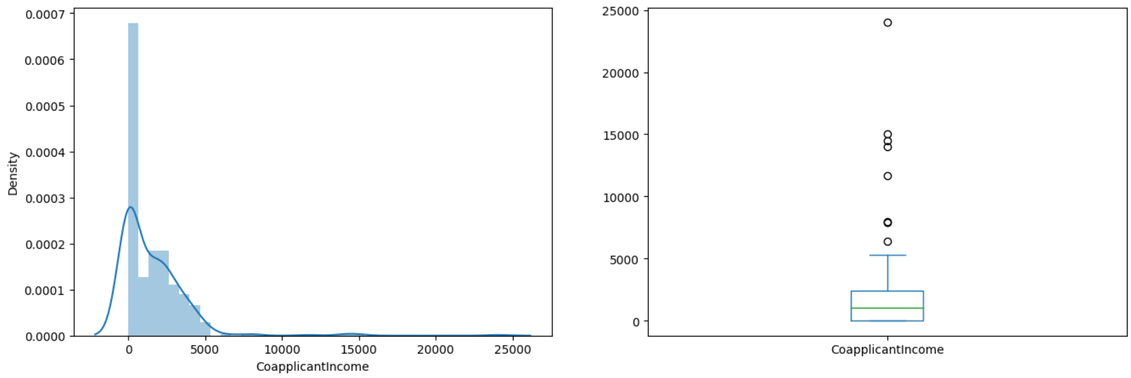
```
<ipython-input-30-150be3c244b2>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function w
ith
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (http
s://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(a['CoapplicantIncome']);
```

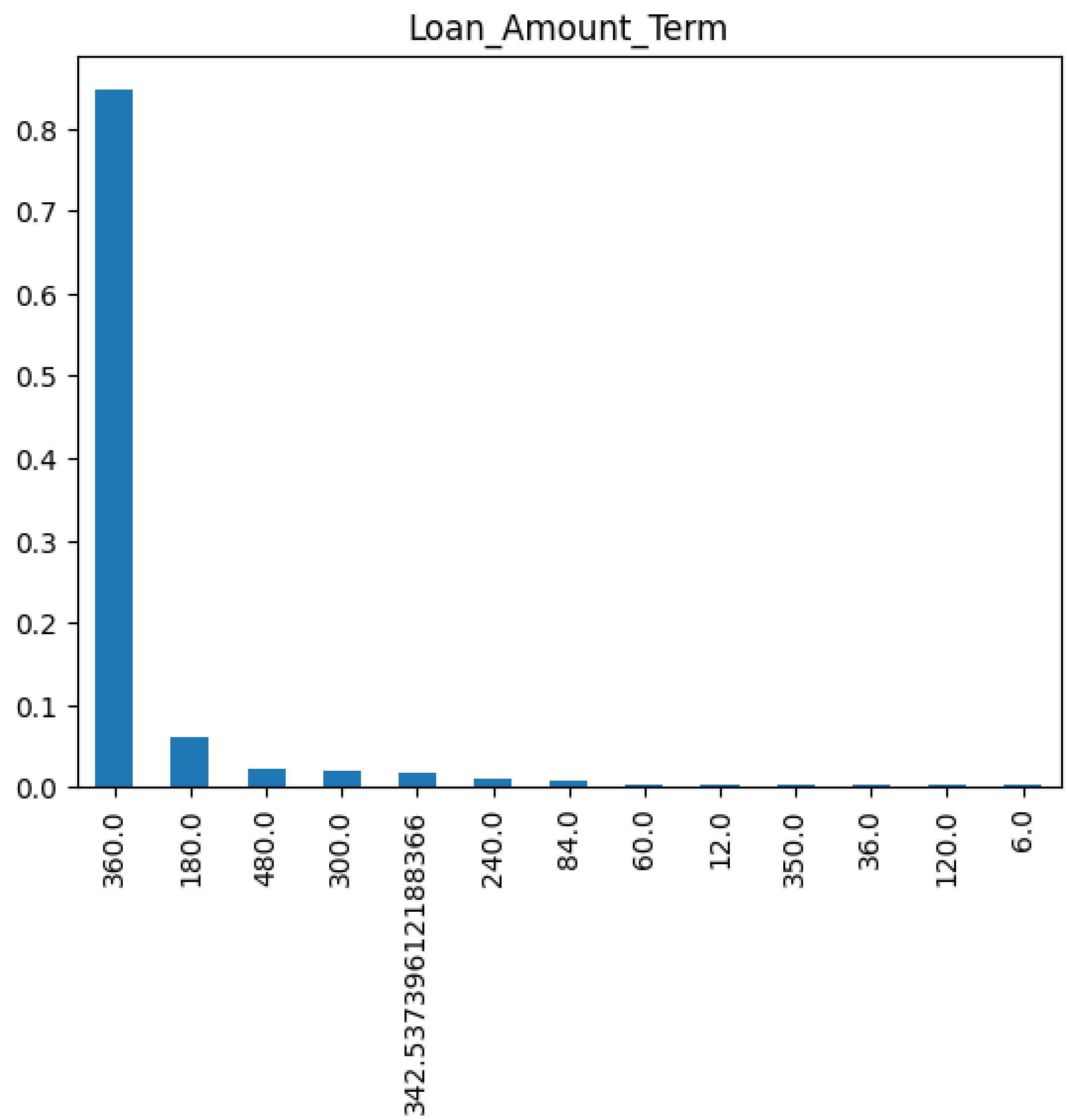


In [ ]:

```
a['Loan_Amount_Term'].value_counts(normalize=True).plot.bar(title= 'Loan_Amount_Term')
```

Out[34]:

```
<Axes: title={'center': 'Loan_Amount_Term'}>
```

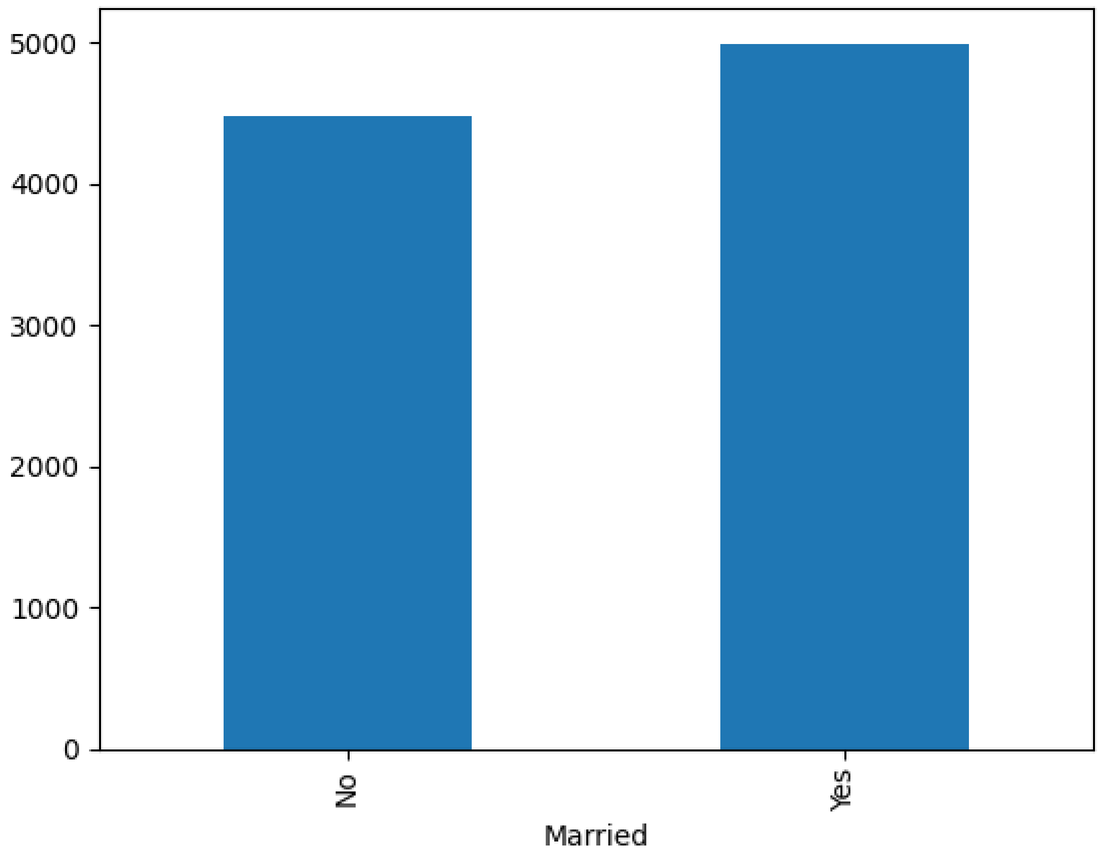


In [ ]:

```
a.groupby('Married')['ApplicantIncome'].mean().plot(kind='bar')
```

Out[46]:

```
<Axes: xlabel='Married'>
```

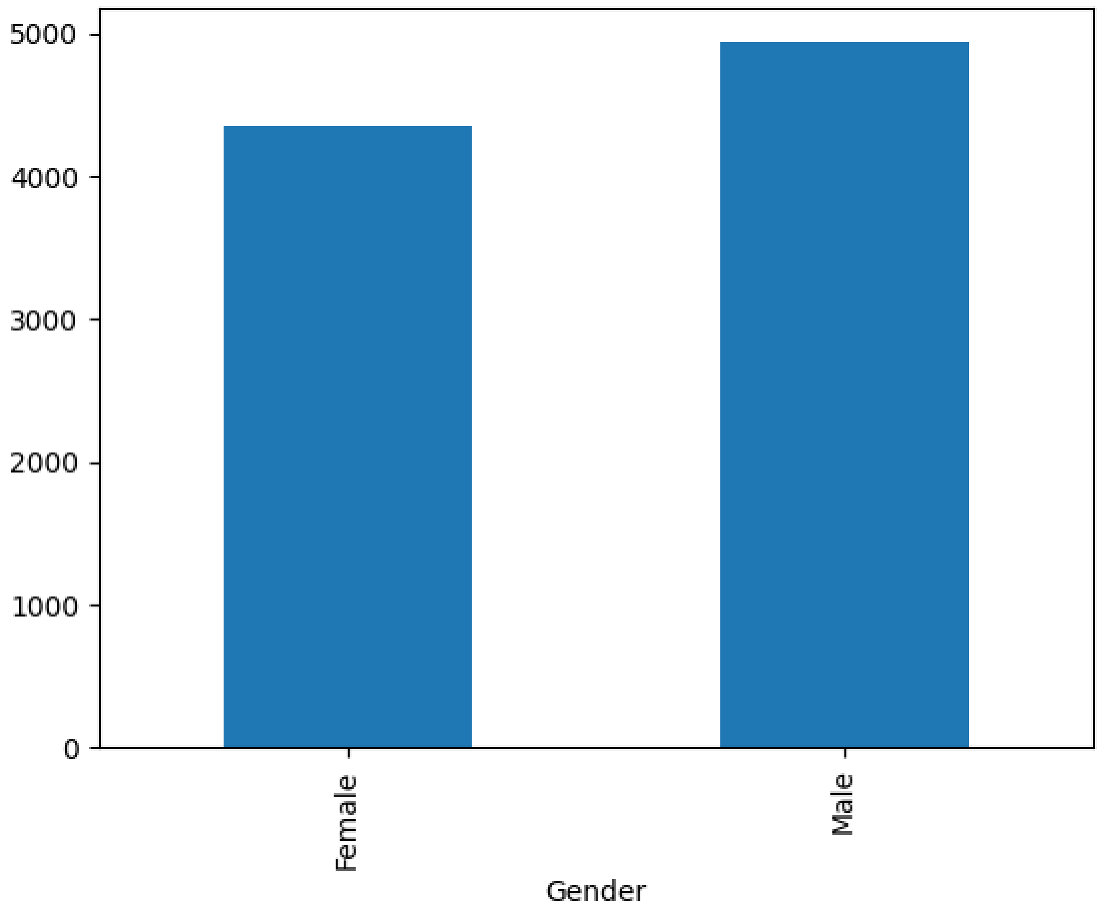


In [ ]:

```python
a.groupby('Gender')['ApplicantIncome'].mean().plot(kind='bar')
```

Out[47]:

```
<Axes: xlabel='Gender'>
```

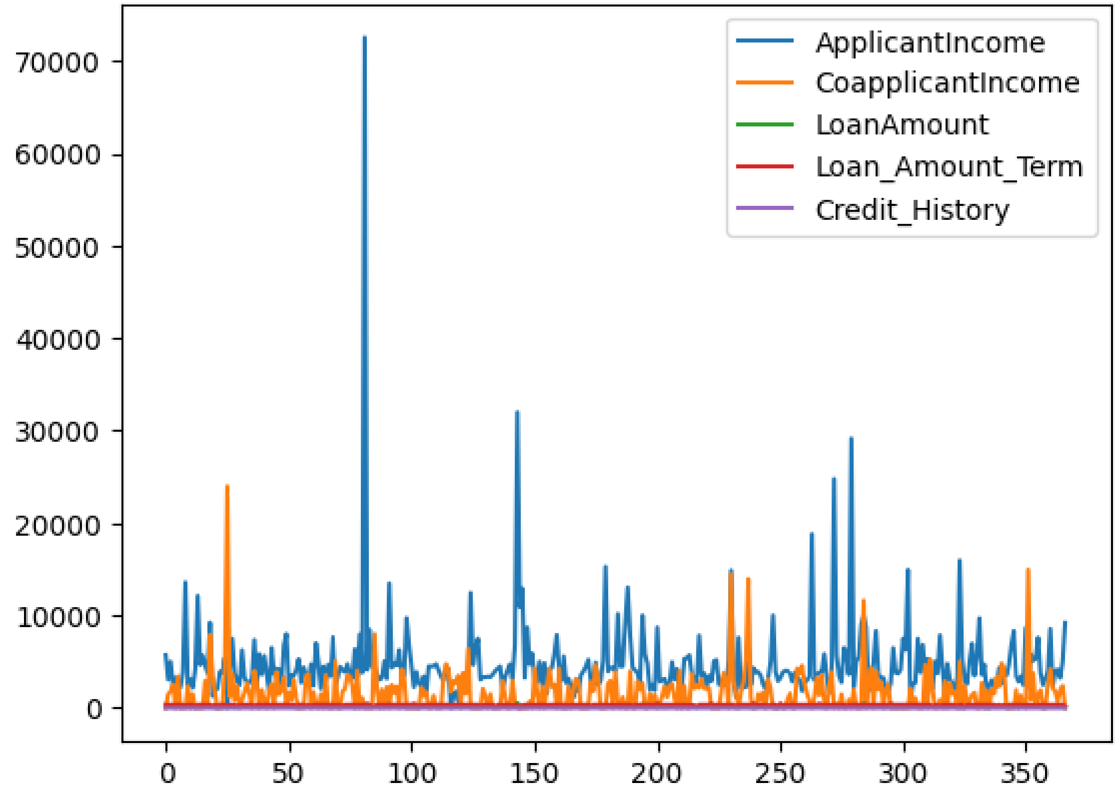


In [ ]:

```
a.plot()
```

Out[48]:

<Axes: >

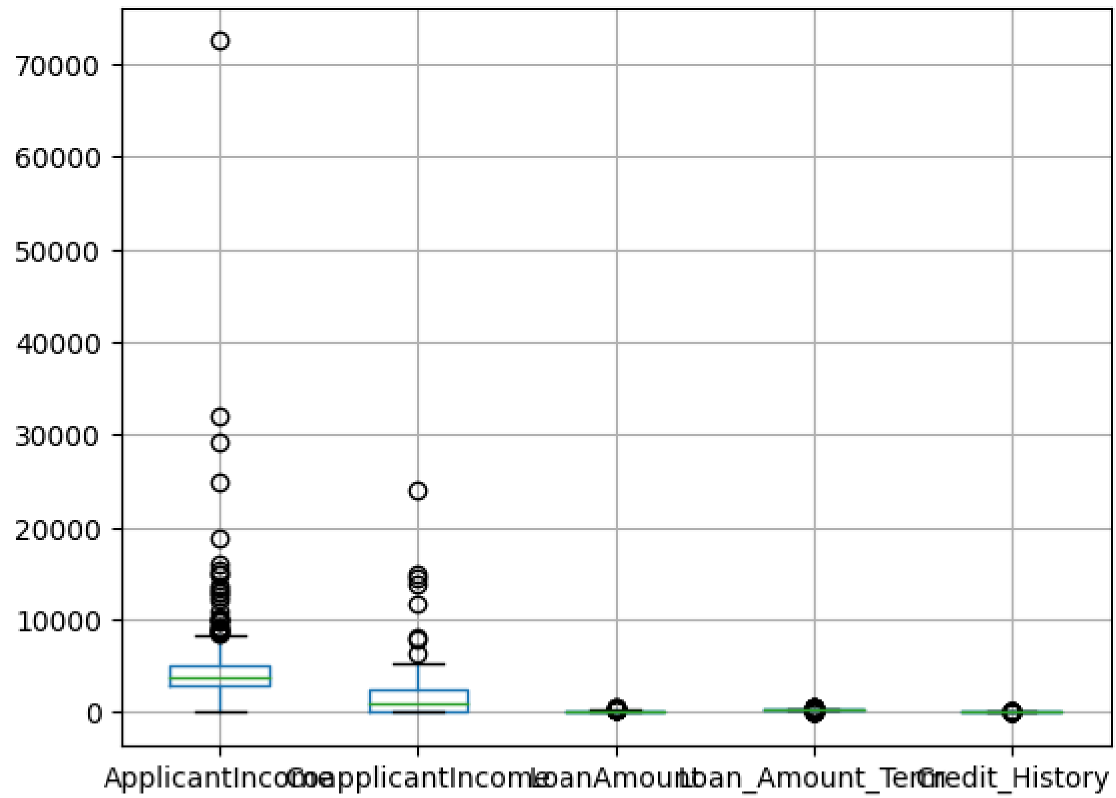


In [ ]:

```
a.boxplot()
```

Out[49]:

<Axes: >



In [ ]:

```
a.corr()
```

Out[50]:

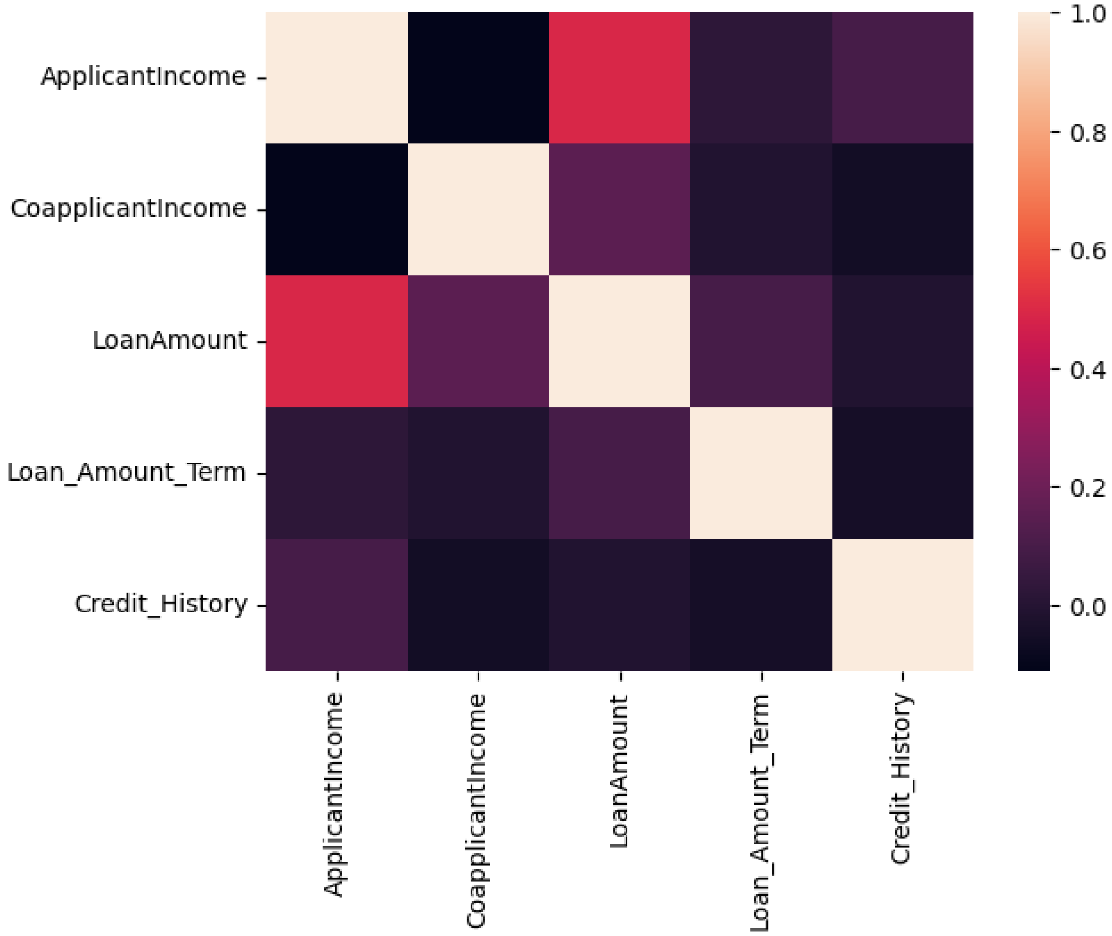
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
ApplicantIncome	1.000000	-0.110335	0.490174	0.023187
CoapplicantIncome	-0.110335	1.000000	0.150112	-0.010940
LoanAmount	0.490174	0.150112	1.000000	0.093856
Loan_Amount_Term	0.023187	-0.010940	0.093856	1.000000
Credit_History	0.094944	-0.058004	-0.013201	-0.048146

In [ ]:

```
sns.heatmap(a.corr())
```

Out[51]:

```
<Axes: >
```

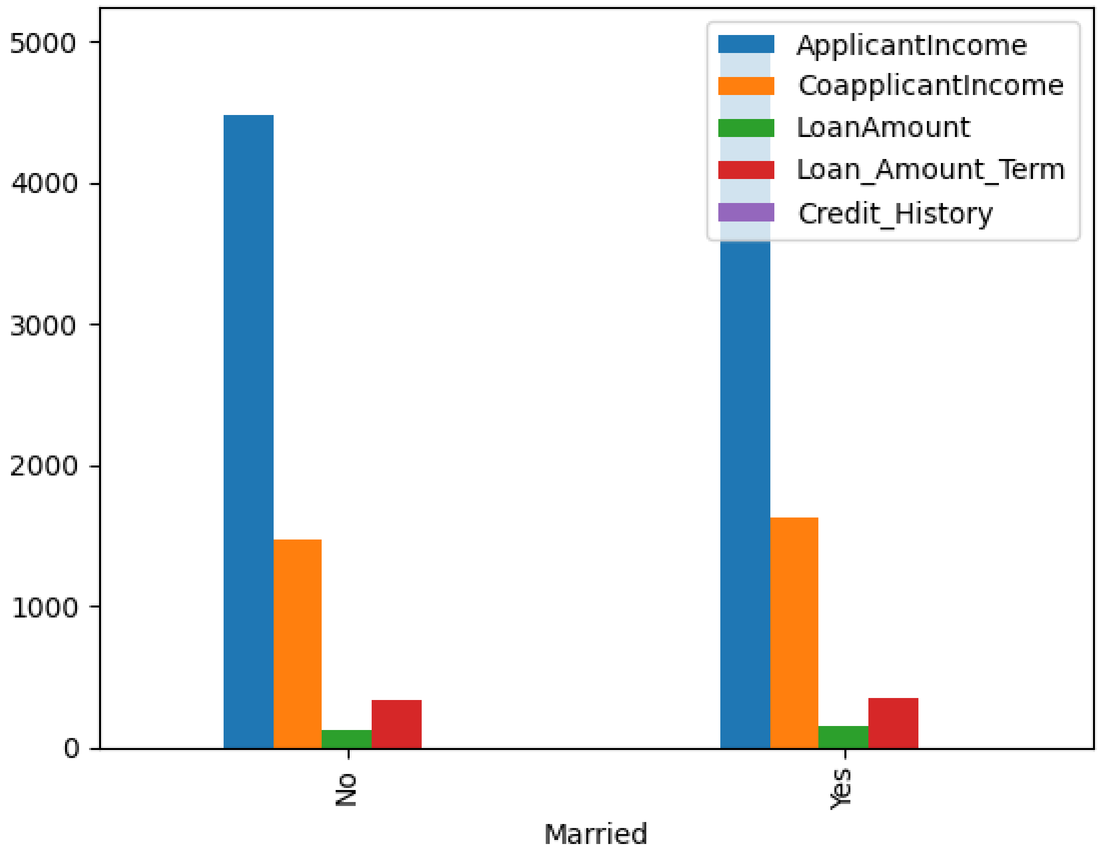


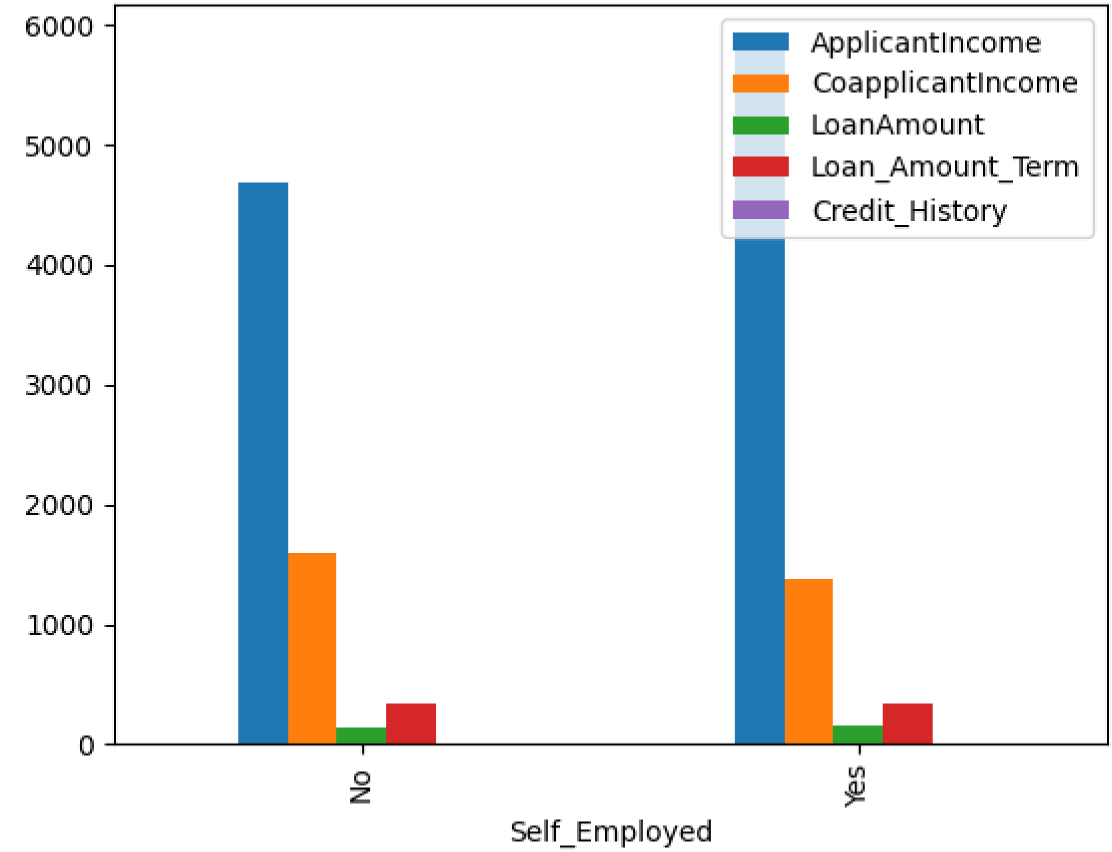
In [ ]:

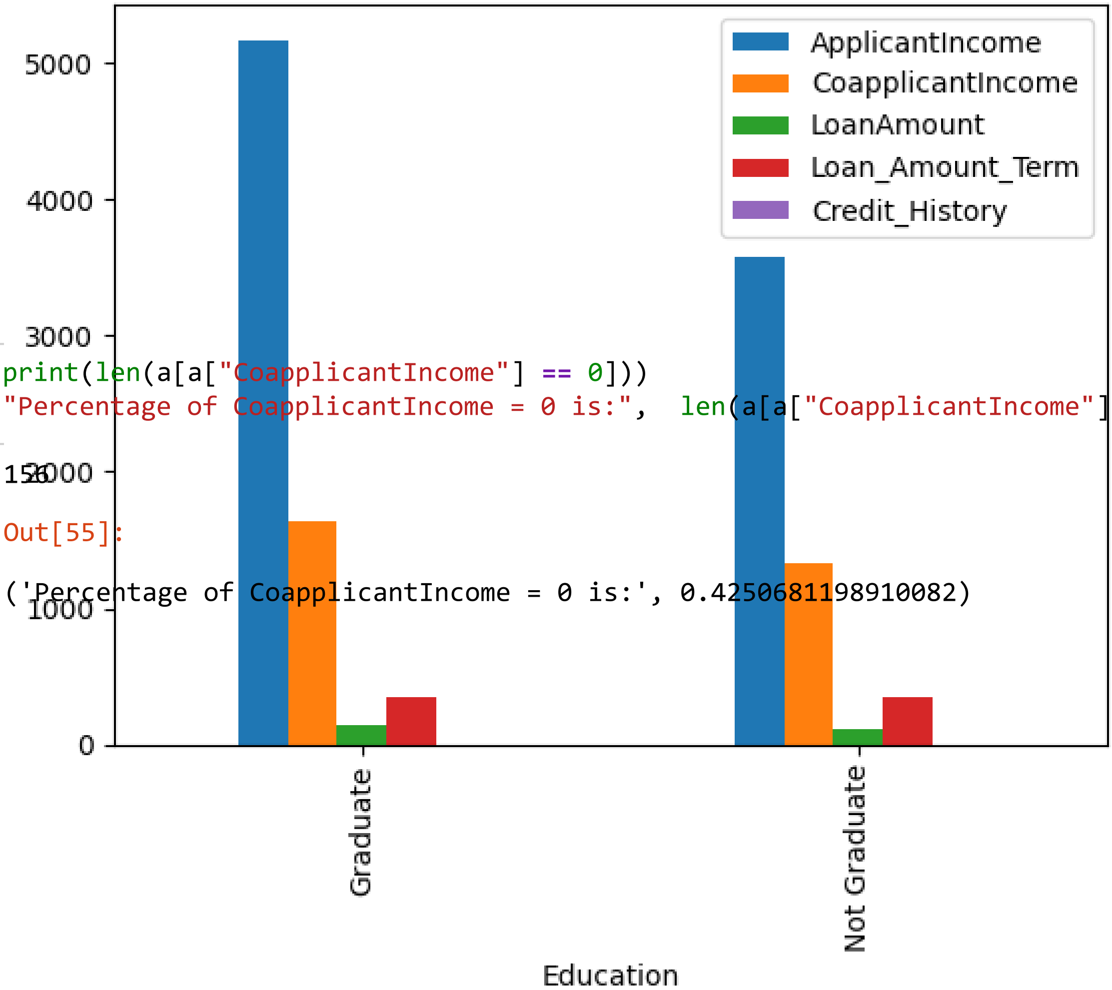
```
Marr=a.groupby(by='Married').mean()
Marr.plot(kind='bar')
Edu1=a.groupby(by='Self_Employed').mean()
Edu1.plot(kind='bar')
Edu=a.groupby(by='Education').mean()
Edu.plot(kind='bar')
```

Out[54]:

```
<Axes: xlabel='Education'>
```








```
print(len(a[a["CoapplicantIncome"] == 0]))
"Percentage of CoapplicantIncome = 0 is:",  len(a[a["CoapplicantIncome"] == 0])/len(a["C
```

156

Out[55]:

```
('Percentage of CoapplicantIncome = 0 is:', 0.4250681198910082)
```

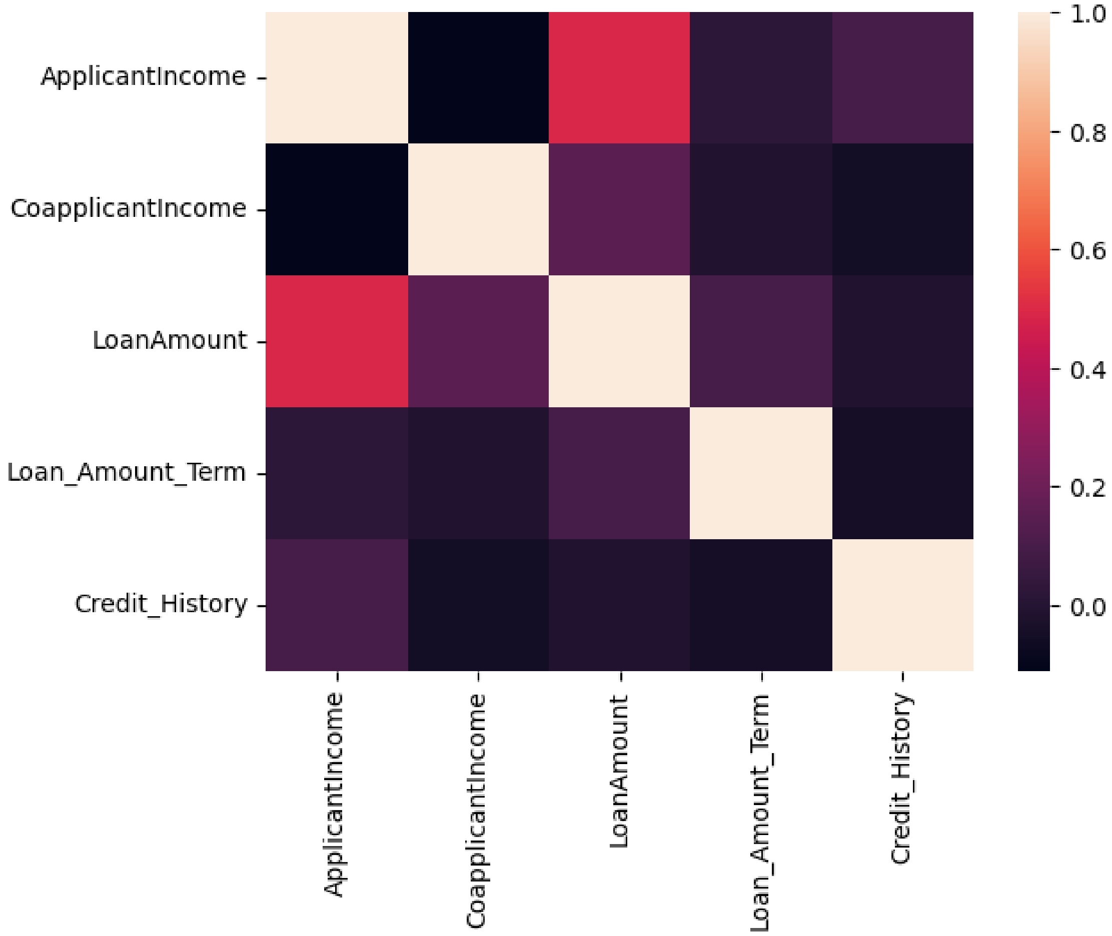
does not look
applicant, so the
on it. So we can
visualize the

In [ ]:

```
sns.heatmap(a.corr())
```

Out[60]:

```
<Axes: >
```



Now lets look at the correlation between all the numerical variables. We can use the corr() to compute pairwise correlation of columns, excluding NA/null values using pearson correlation coefficient. Then we will use the heat map to visualize the correlation. Heatmaps visualize data through variations in coloring. The variables with darker color means their correlation is more.

In [ ]:

```
matrix=a.corr()
f, ax=plt.subplots(figsize=(17,8))
sns.heatmap(matrix, vmax=1, square=True, cmap="BuPu", annot=True)

matrix
```

Out[57]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
ApplicantIncome	1.000000	-0.110335	0.490174	0.023187
CoapplicantIncome	-0.110335	1.000000	0.150112	-0.010940
LoanAmount	0.490174	0.150112	1.000000	0.093856
Loan_Amount_Term	0.023187	-0.010940	0.093856	1.000000
Credit_History	0.094944	-0.058004	-0.013201	-0.048146

Note: We see that the most correlated variables are

- (ApplicantIncome - LoanAmount) with correlation coefficient of 0.49
- (Credit_History - CoapplicantIncome) with correlation coefficient of -0.058
- LoanAmount is also correlated with CoapplicantIncome with correlation coefficient of 0.15.

Outlier Treatment

As we saw earlier in univariate analysis, LoanAmount contains outliers so we have to treat them as the presence of outliers affects the distribution of the data. Having outliers in the dataset often has a significant effect on the mean and standard deviation and hence affecting the distribution. We must take steps to remove outliers from our data sets.

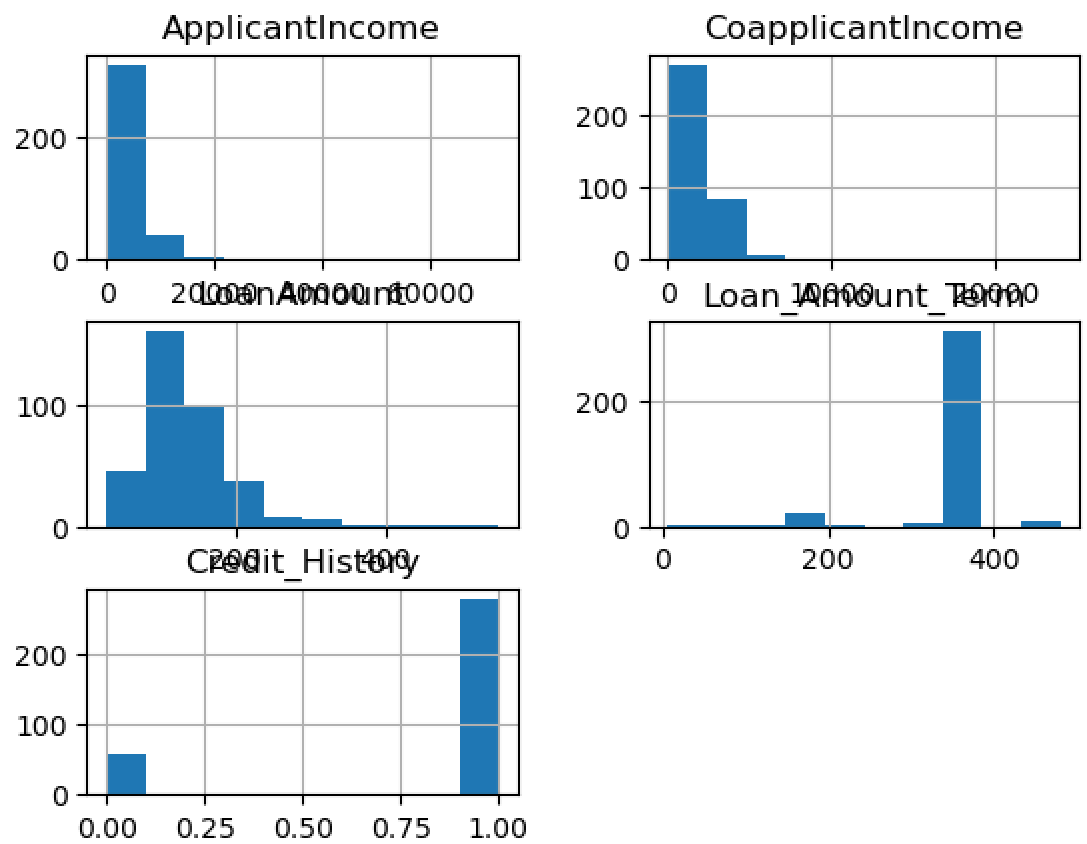
Due to these outliers bulk of the data in the loan amount is at the left and the right tail is longer. This is called right skewness (or positive skewness). One way to remove the skewness is by doing the log transformation. As we take the log transformation, it does not affect the smaller values much, but reduces the larger values. So, we get a distribution similar to normal distribution.

In [34]:

```
a.hist()
```

Out[34]:

```
array([[<Axes: title={'center': 'ApplicantIncome'}>,
        <Axes: title={'center': 'CoapplicantIncome'}>],
       [<Axes: title={'center': 'LoanAmount'}>,
        <Axes: title={'center': 'Loan_Amount_Term'}>],
       [<Axes: title={'center': 'Credit_History'}>, <Axes: >]],
      dtype=object)
```

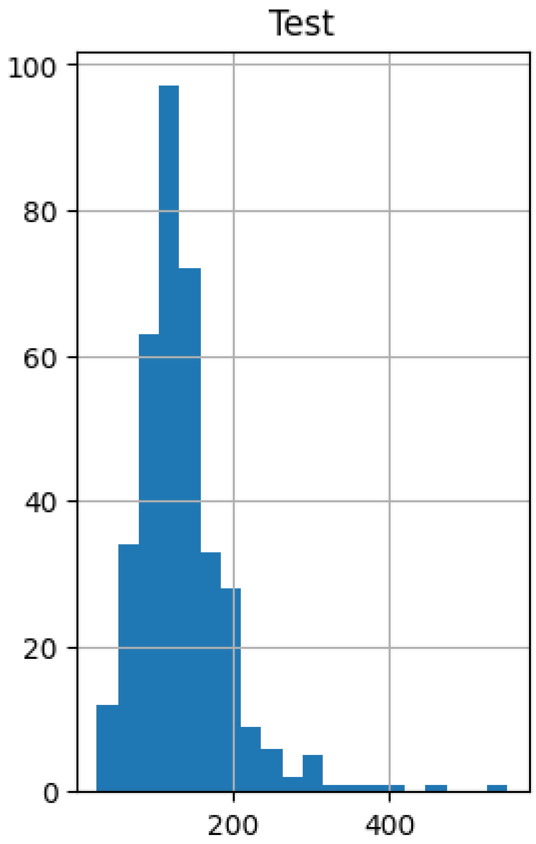


In [ ]:

```
# before log transformation
ax2 = plt.subplot(122)
a['LoanAmount'].hist(bins=20)
ax2.set_title("Test")
```

Out[58]:

```
Text(0.5, 1.0, 'Test')
```

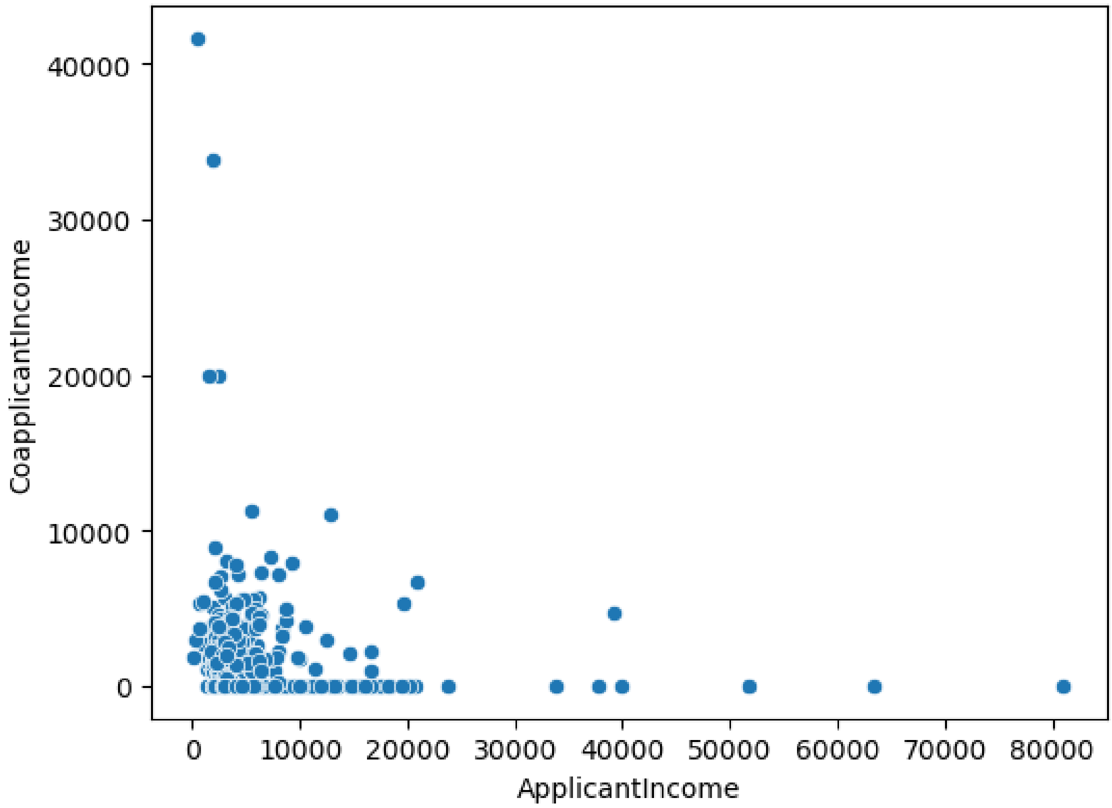


In [ ]:

```
sns.scatterplot(x='ApplicantIncome',y='CoapplicantIncome',data=b)
```

Out[7]:

```
<Axes: xlabel='ApplicantIncome', ylabel='CoapplicantIncome'>
```

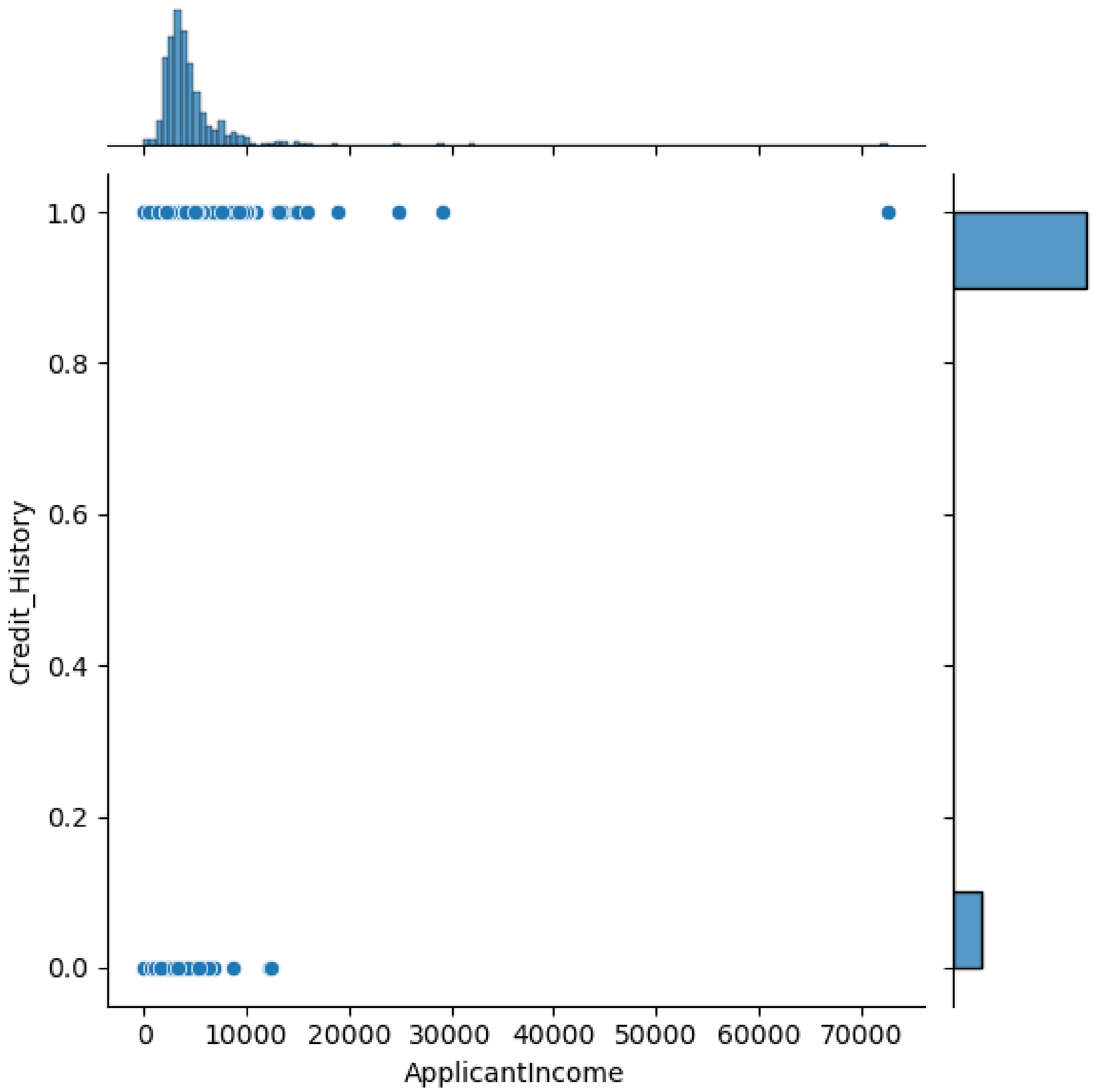


In [ ]:

```
sns.jointplot(x='ApplicantIncome',y='Credit_History',data=a)
```

Out[8]:

```
<seaborn.axisgrid.JointGrid at 0x20935ba5b10>
```

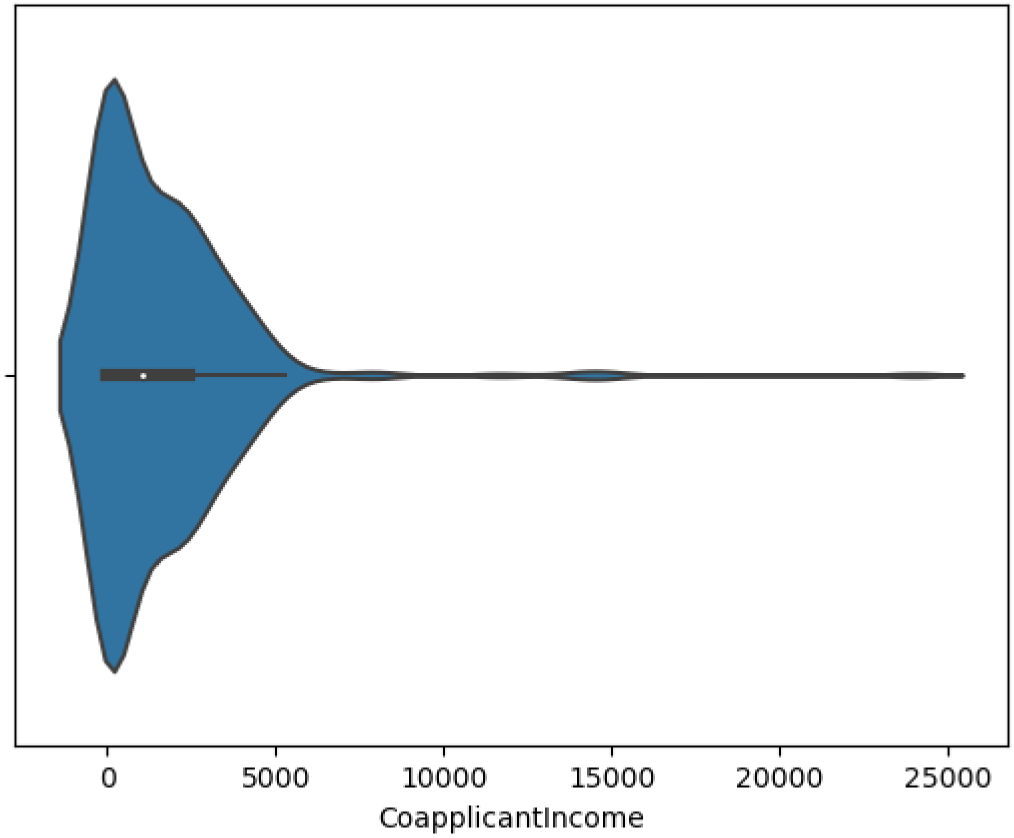


In [ ]:

```
sns.violinplot(x='CoapplicantIncome',data=a)
```

Out[11]:

```
<Axes: xlabel='CoapplicantIncome'>
```

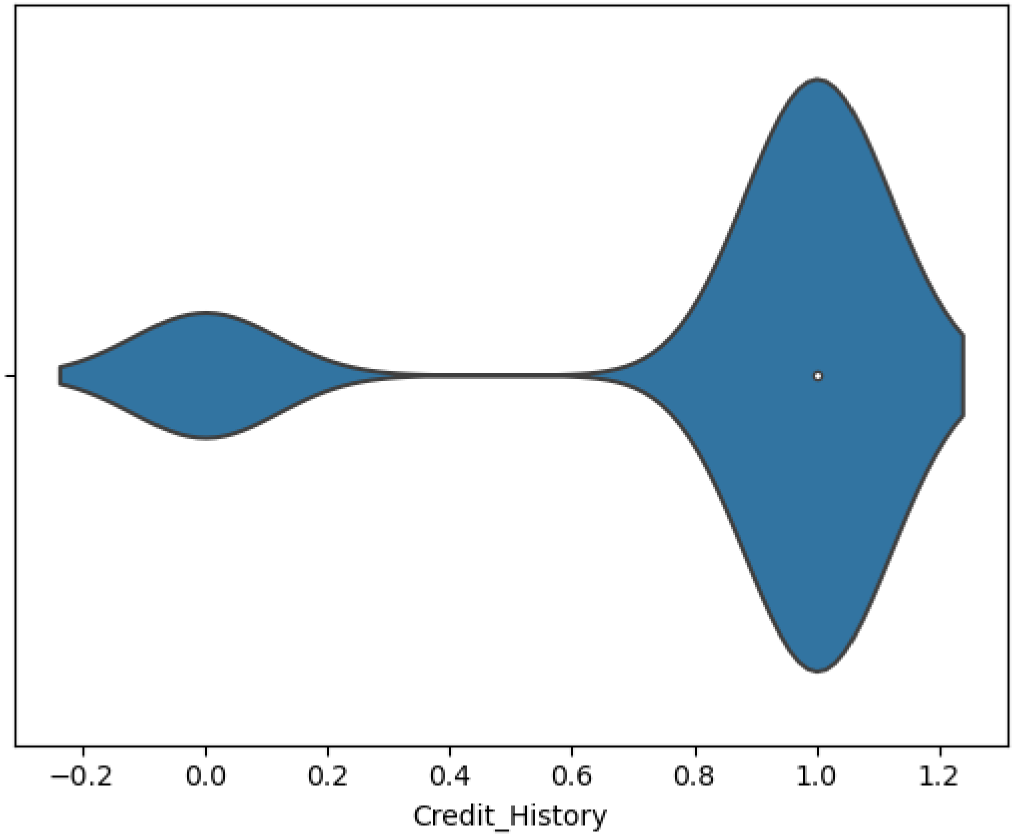


In [ ]:

```
sns.violinplot(x='Credit_History',data=a)
```

Out[12]:

```
<Axes: xlabel='Credit_History'>
```

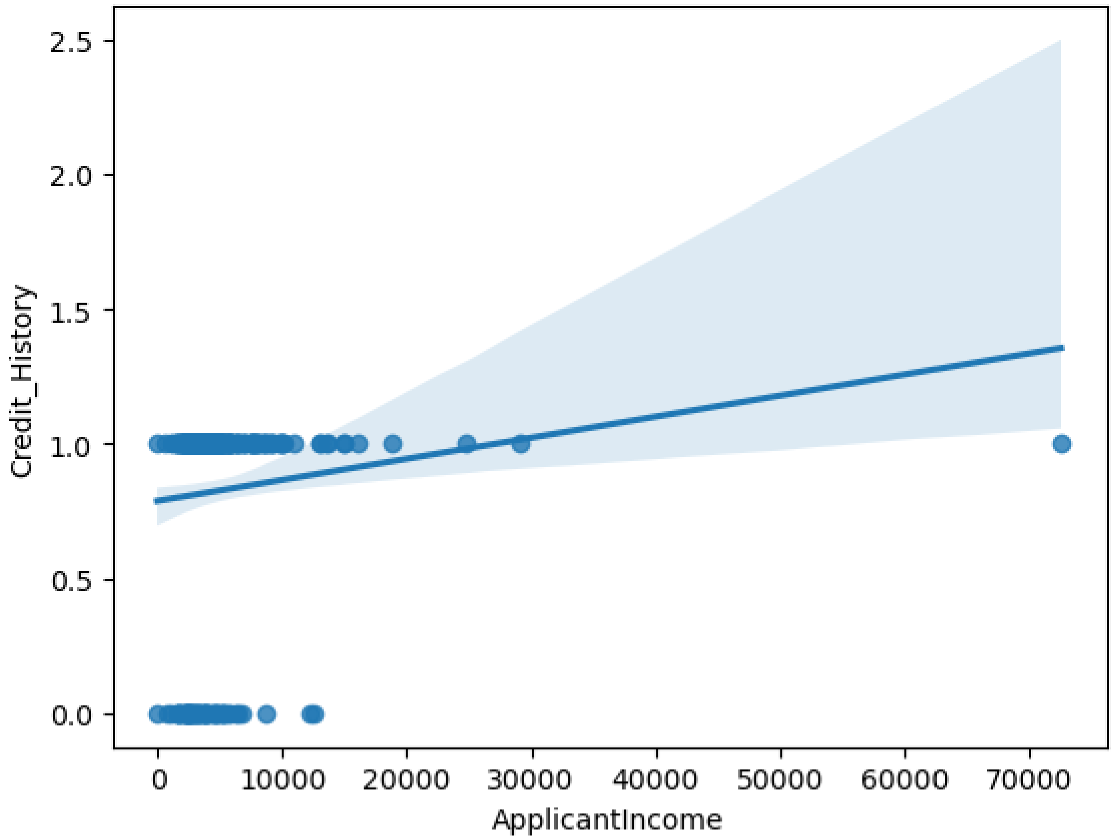


In [ ]:

```
sns.regplot(x='ApplicantIncome',y='Credit_History',data=a)
```

Out[18]:

```
<Axes: xlabel='ApplicantIncome', ylabel='Credit_History'>
```

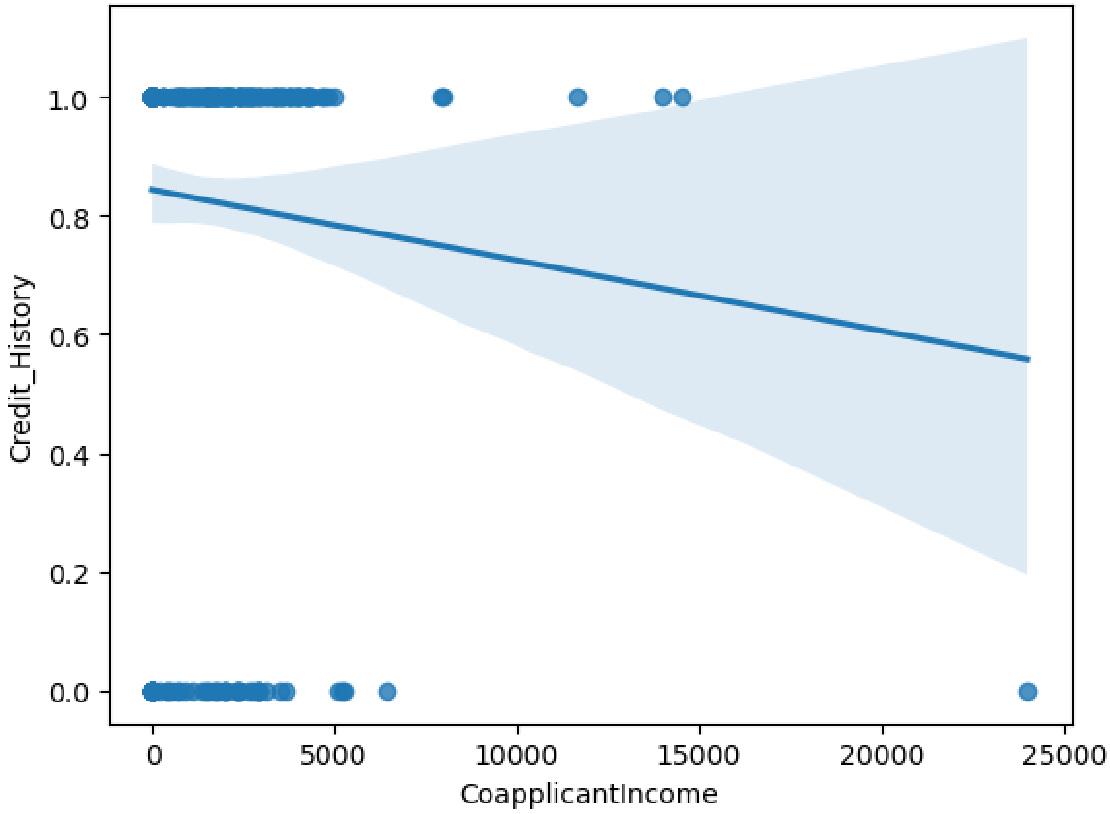


In [ ]:

```
sns.regplot(x='CoapplicantIncome',y='Credit_History',data=a)
```

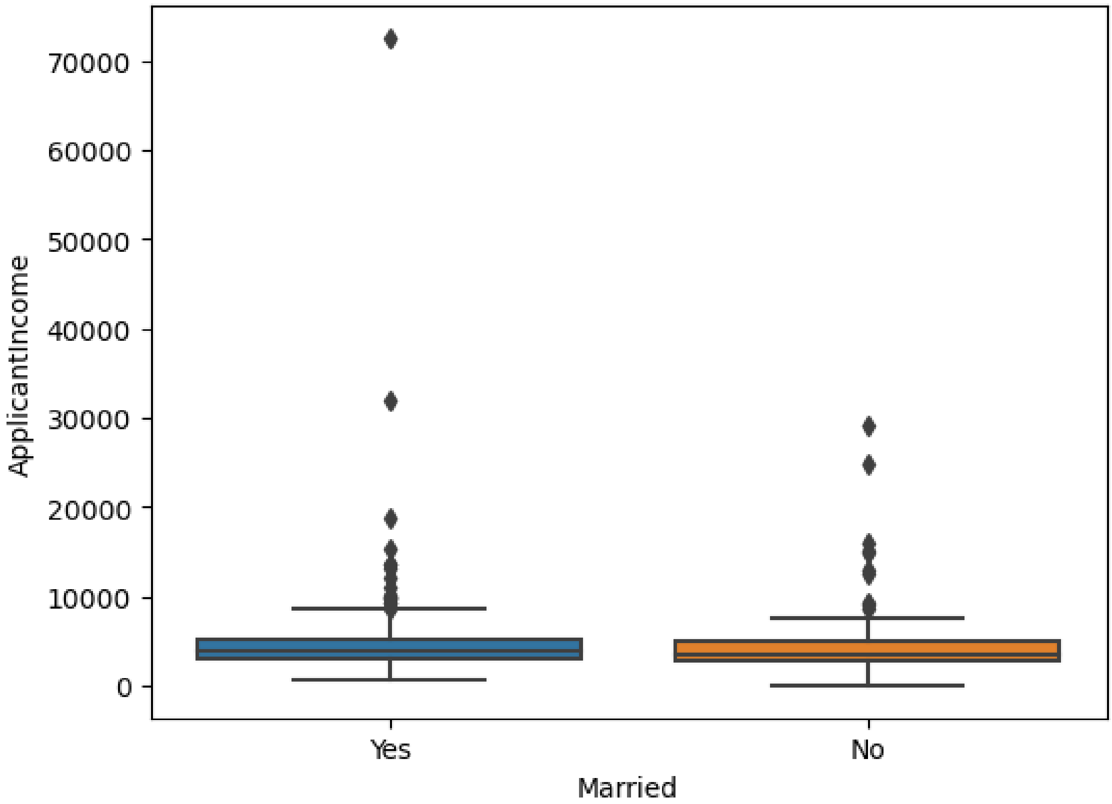
Out[19]:

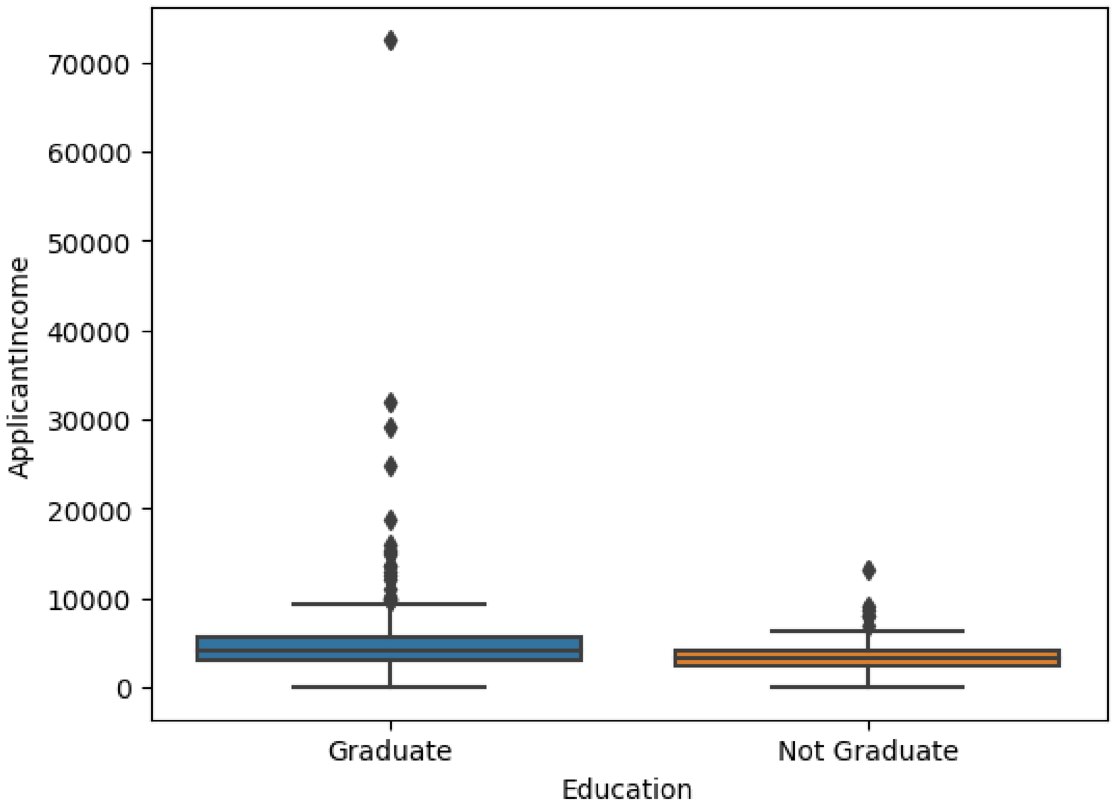
```
<Axes: xlabel='CoapplicantIncome', ylabel='Credit_History'>
```

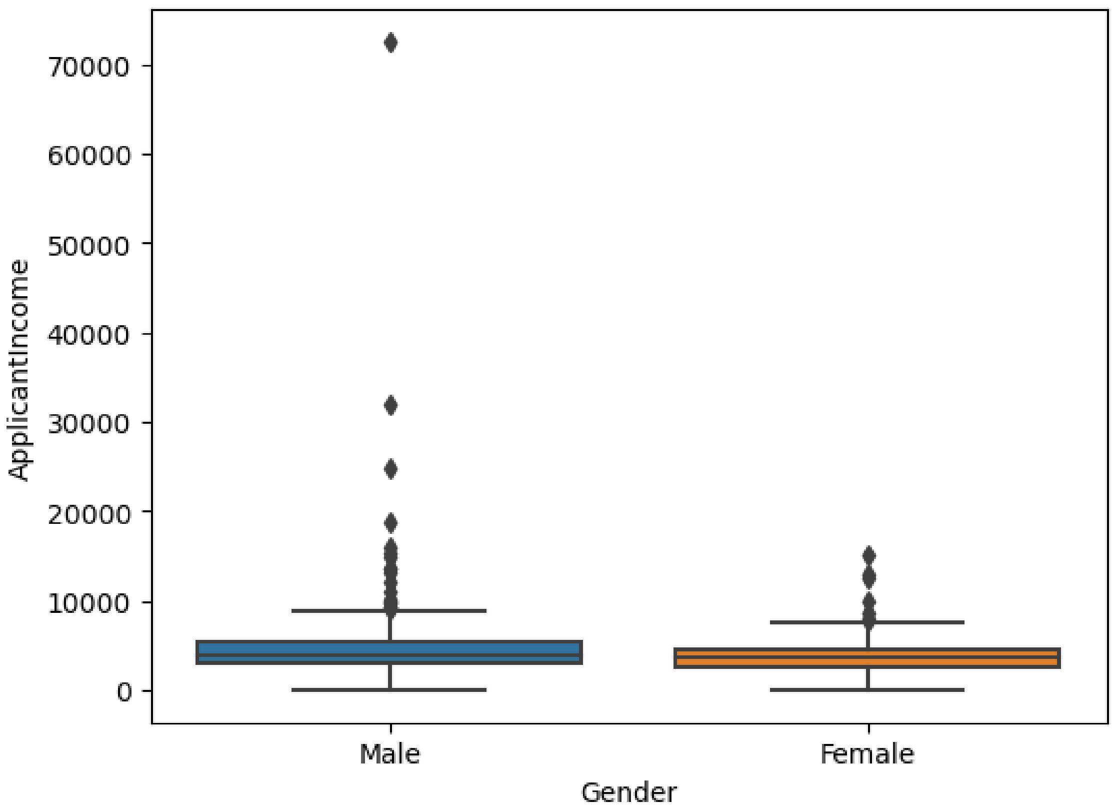


In [ ]:

```
sns.boxplot(x='Married',y='ApplicantIncome',data=a)
plt.show()
sns.boxplot(x='Education',y='ApplicantIncome',data=a)
plt.show()
sns.boxplot(x='Gender',y='ApplicantIncome',data=a);
plt.show()
```





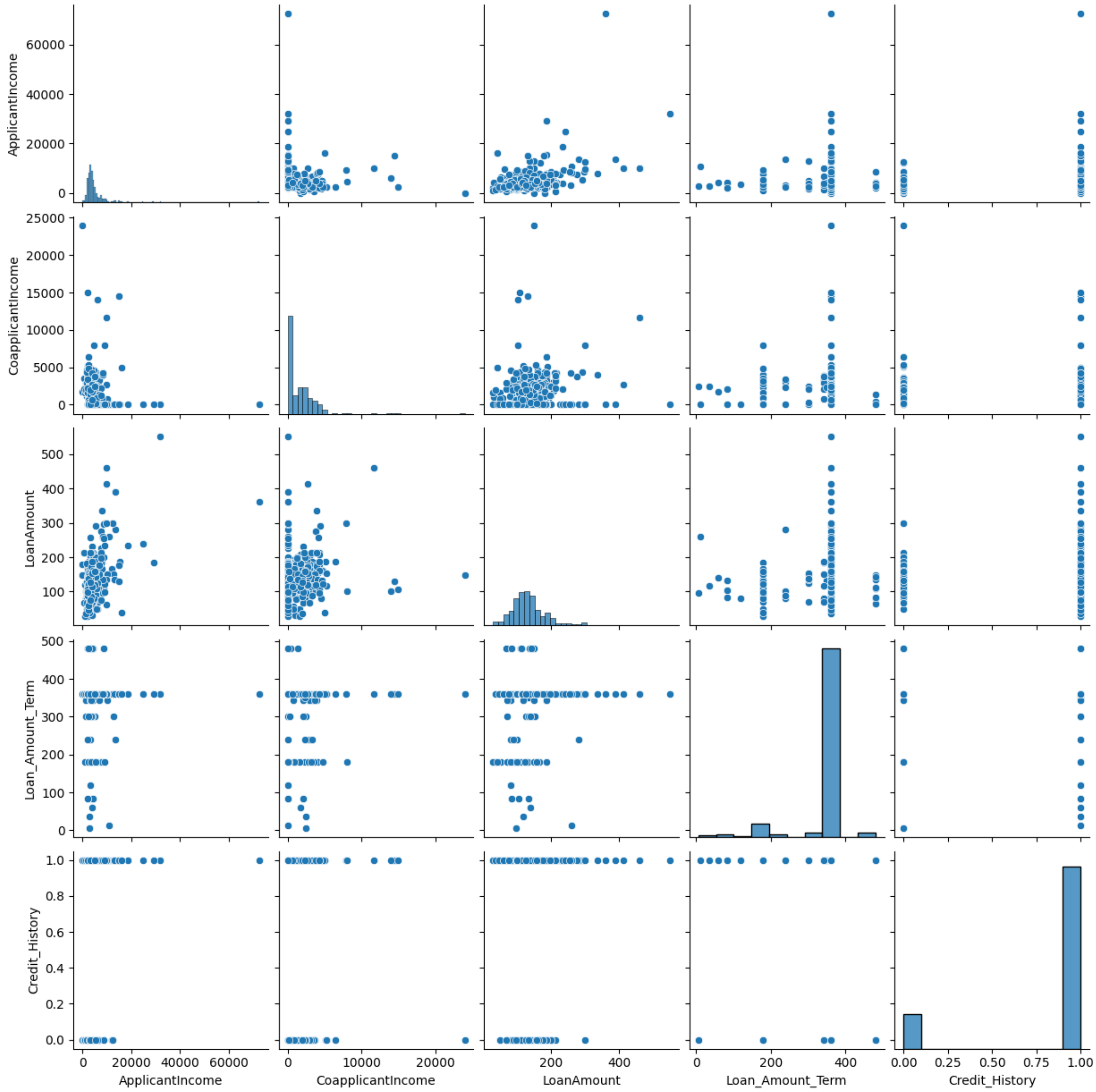

70000
60000
50000
40000
30000
20000
10000
0
ApplicantIncome
Male
Female
Gender

In [ ]:

```
sns.pairplot(a)
```

Out[71]:

<seaborn.axisgrid.PairGrid at 0x7fb24c967bb0>



Train dataset

In [3]:

```
#load the test dataset
b=pd.read_csv("train.csv")
b
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000
...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900
610	LP002979	Male	Yes	3+	Graduate	No	4106
611	LP002983	Male	Yes	1	Graduate	No	8072
612	LP002984	Male	Yes	2	Graduate	No	7583
613	LP002990	Female	No	0	Graduate	Yes	4583

614 rows × 13 columns

In [ ]:

```
b.describe
```

Out[36]:

```
<bound method NDFrame.describe of       Loan_ID  Gender Married Dependent
s     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
1               4583             1508.0       128.0             360.0
2               3000                0.0        66.0             360.0
3               2583             2358.0       120.0             360.0
4               6000                0.0       141.0             360.0
..               ...                ...         ...               ...
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
..              ...           ...         ...
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[614 rows x 13 columns]>
```

In [ ]:

```
# show the shape of the dataset i.e. no of rows, no of columns
b.shape
```

Out[38]:

```
(614, 13)
```

In [ ]:

```
b_length=len(b)
```

In [ ]:

```
#take a look at the features (i.e. independent variables) in the test dataset
b.columns
```

Out[40]:

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmou
nt',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Statu
s'],
      dtype='object')
```

In [ ]:

```
b.dtypes
```

Out[42]:

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

In [39]:

```
b.mean()
```

```
C:\Users\meruv\AppData\Local\Temp\ipykernel_16028\2233723229.py:1: Future
Warning: The default value of numeric_only in DataFrame.mean is deprecate
d. In a future version, it will default to False. In addition, specifying
'numeric_only=None' is deprecated. Select only valid columns or specify t
he value of numeric_only to silence this warning.
  b.mean()
```

Out[39]:

```
ApplicantIncome      5403.459283
CoapplicantIncome    1621.245798
LoanAmount            146.412162
Loan_Amount_Term      342.000000
Credit_History          0.842199
Loan_Status             0.687296
dtype: float64
```

In [40]:

```
b.std()
```

```
C:\Users\meruv\AppData\Local\Temp\ipykernel_16028\2079931243.py:1: Future
Warning: The default value of numeric_only in DataFrame.std is deprecate
d. In a future version, it will default to False. In addition, specifying
'numeric_only=None' is deprecated. Select only valid columns or specify t
he value of numeric_only to silence this warning.
  b.std()
```

Out[40]:

```
ApplicantIncome      6109.041673
CoapplicantIncome    2926.248369
LoanAmount             85.587325
Loan_Amount_Term       65.120410
Credit_History          0.364878
Loan_Status             0.463973
dtype: float64
```

Data Pre-Processing

Missing value imputation

In [ ]:

```
# check for missing values
b.apply(lambda x:sum(x.isnull()),axis=0)
```

Out[43]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

There are missing values in Gender, Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_History features. We will treat the missing values in all the features one by one.

- For numerical variables: imputation using mean or median
- For categorical variables: imputation using mode

There are very less missing values in Gender, Married, Dependents, Credit_History and Self_Employed features so we can fill them using the mode of the features. If an independent variable in our dataset has huge amount of missing data e.g. 80% missing values in it, then we would drop the variable from the dataset.

In [ ]:

```
# replace missing values with the mode
b['Gender'].fillna(b['Gender'].mode()[0], inplace=True)
b['Married'].fillna(b['Married'].mode()[0], inplace=True)
b['Dependents'].fillna(b['Dependents'].mode()[0], inplace=True)
b['Self_Employed'].fillna(b['Self_Employed'].mode()[0], inplace=True)
b['Credit_History'].fillna(b['Credit_History'].mode()[0], inplace=True)
```

In [ ]:

```
# check whether all the missing values are filled in the Train dataset
b.apply(lambda x:sum(x.isnull()),axis=0)
```

Out[62]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [ ]:

```
b.describe()
```

Out[63]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histor
count	614.000000	614.000000	614.000000	614.000000	614.00000
mean	5403.459283	1621.245798	146.043839	342.012253	0.85504
std	6109.041673	2926.248369	84.059220	64.372539	0.35233
min	150.000000	0.000000	9.000000	12.000000	0.00000
25%	2877.500000	0.000000	100.250000	360.000000	1.00000
50%	3812.500000	1188.500000	129.000000	360.000000	1.00000
75%	5795.000000	2297.250000	164.750000	360.000000	1.00000
max	81000.000000	41667.000000	700.000000	480.000000	1.00000

Target Variable (Categorical)

We will first look at the target variable, i.e., Loan_Status. As it is a categorical variable, let us look at its frequency table and bar plot.

In [ ]:

```
b['Credit_History'].value_counts()
```

Out[64]:

```
1.0    525
0.0     89
Name: Credit_History, dtype: int64
```

In [ ]:

```
b['Loan_Amount_Term'].value_counts()
```

Out[65]:

```
360.000000    512
180.000000     44
480.000000     15
342.537396     14
300.000000     13
240.000000      4
84.000000       4
120.000000      3
60.000000       2
36.000000       2
12.000000       1
Name: Loan_Amount_Term, dtype: int64
```

In [ ]:

```
b['Self_Employed'].value_counts()
```

Out[67]:

```
No     532
Yes     82
Name: Self_Employed, dtype: int64
```

In [ ]:

```
b['Dependents'].value_counts()
```

Out[68]:

```
0     360
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [ ]:

```
b['Gender'].value_counts()
```

Out[69]:

```
Male      489
Female    125
Name: Gender, dtype: int64
```

In [ ]:

```
b['Married'].value_counts()
```

Out[70]:

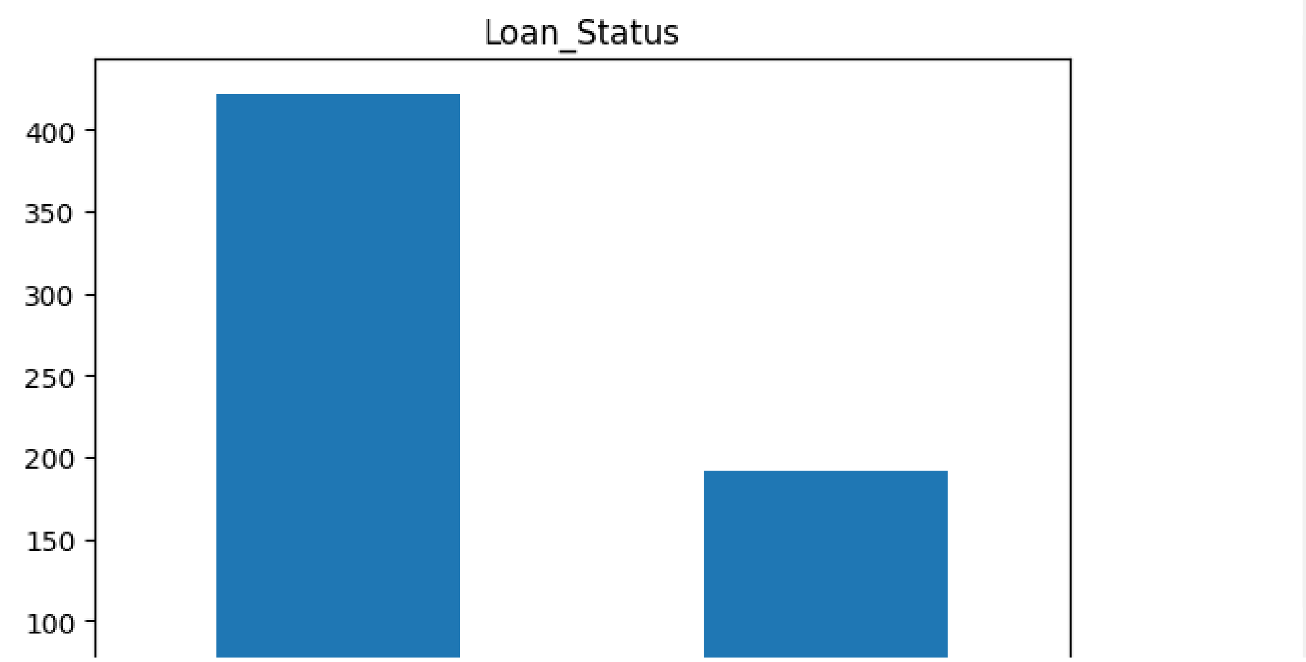
```
Yes    401
No     213
Name: Married, dtype: int64
```

In [ ]:

```
# replacing 3+ in Dependents variable with 3 for both train and test set
tra=b['Dependents'].replace('3+', 3, inplace=True)
```

In [ ]:

```
b['Loan_Status'].value_counts().plot.bar(title="Loan_Status")
plt.show()
b['Gender'].value_counts().plot.bar(title="Gender")
plt.show()
b['Married'].value_counts().plot.bar(title="Married")
plt.show()
b['Self_Employed'].value_counts().plot.bar(title="Self_employed")
plt.show()
b['Credit_History'].value_counts().plot.bar(title="Credit_History")
plt.show()
```



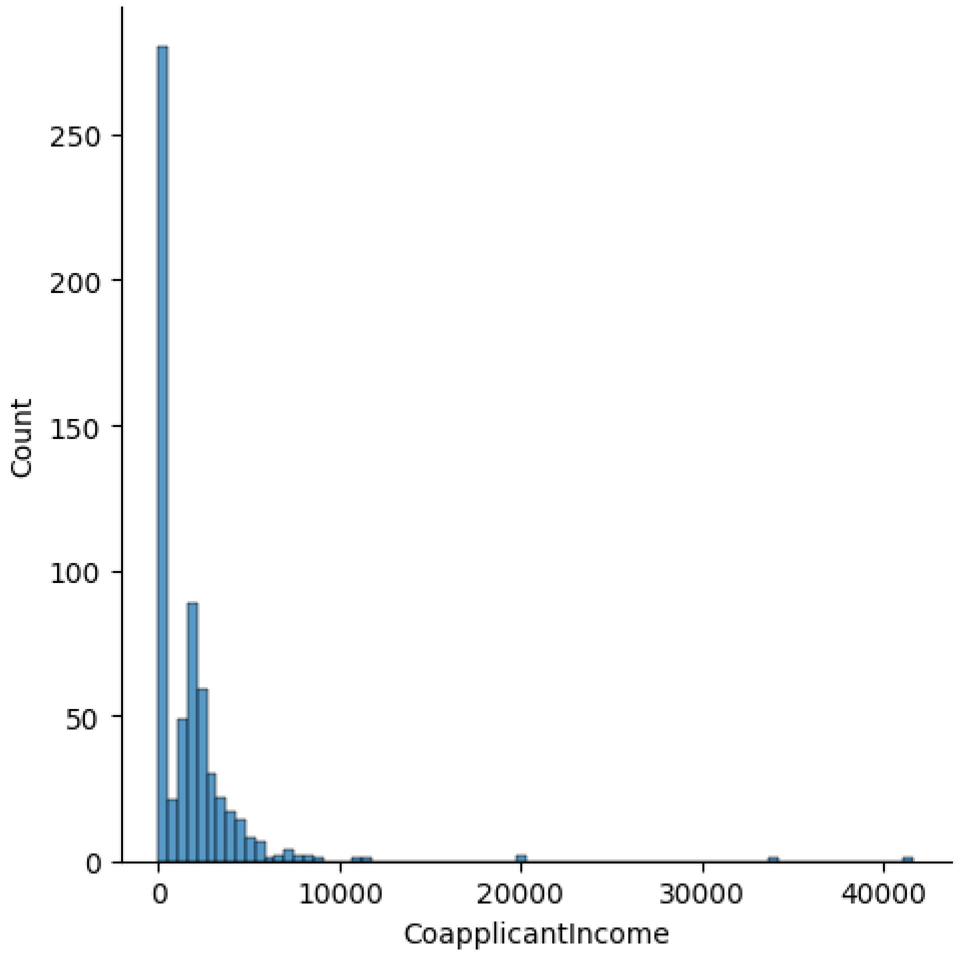
Independent Variable (Numerical)

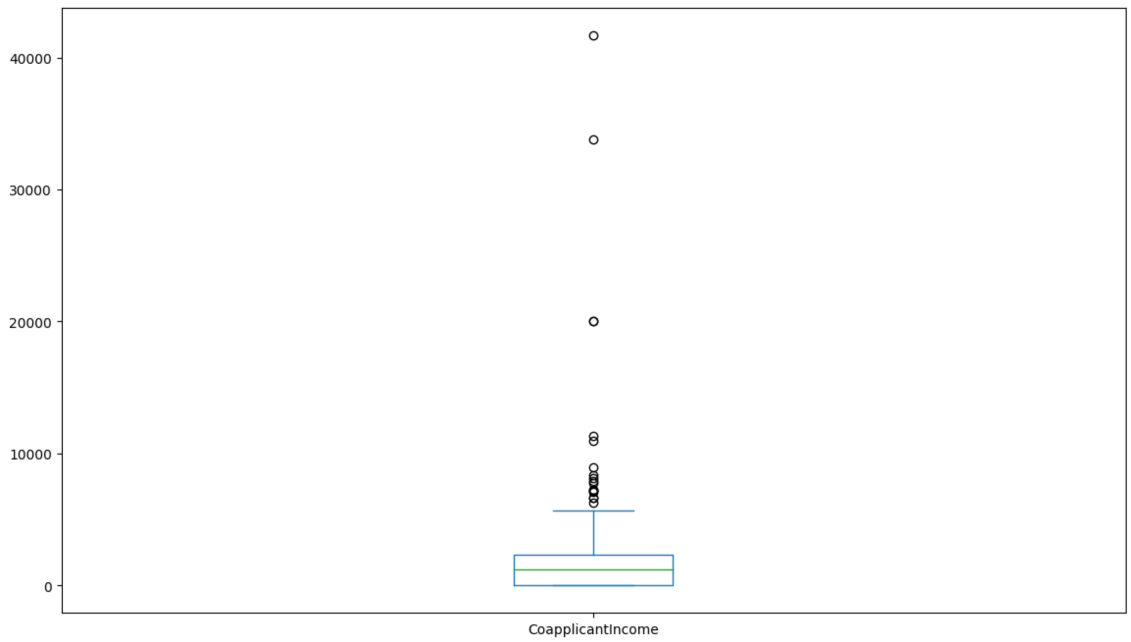
There are 4 features that are Numerical: These features have numerical values (ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term)

It can be inferred that most of the data in the distribution of applicant income is towards left which means it is not normally distributed. The distribution is right-skewed (positive skewness). We will try to make it normal in later sections as algorithms works better if the data is normally distributed.

In [14]:

```python
# Visualizing CoapplicantIncome
sns.displot(b['CoapplicantIncome'])
plt.show()
b['CoapplicantIncome'].plot.box(figsize=(14,8))
plt.show()
```



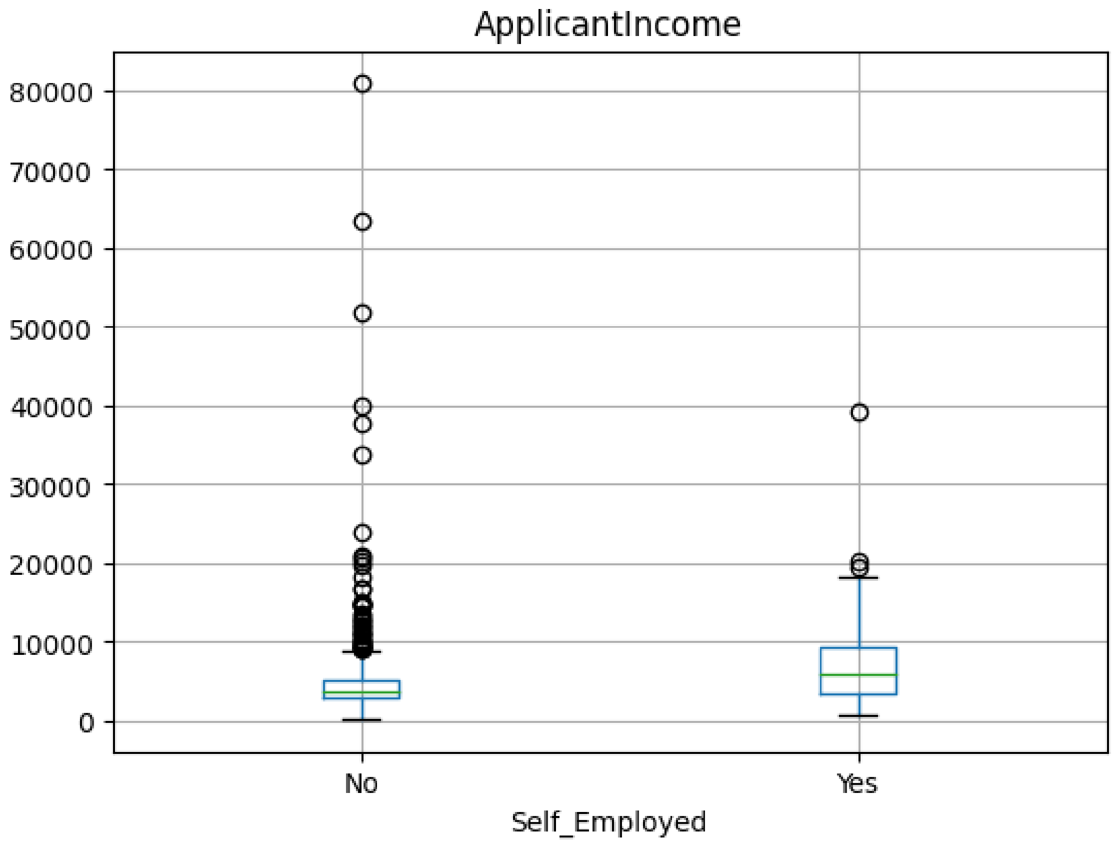


In [ ]:

```
b.boxplot(column='ApplicantIncome',by='Self_Employed')
plt.suptitle("")
```

Out[74]:

```
Text(0.5, 0.98, '')
```



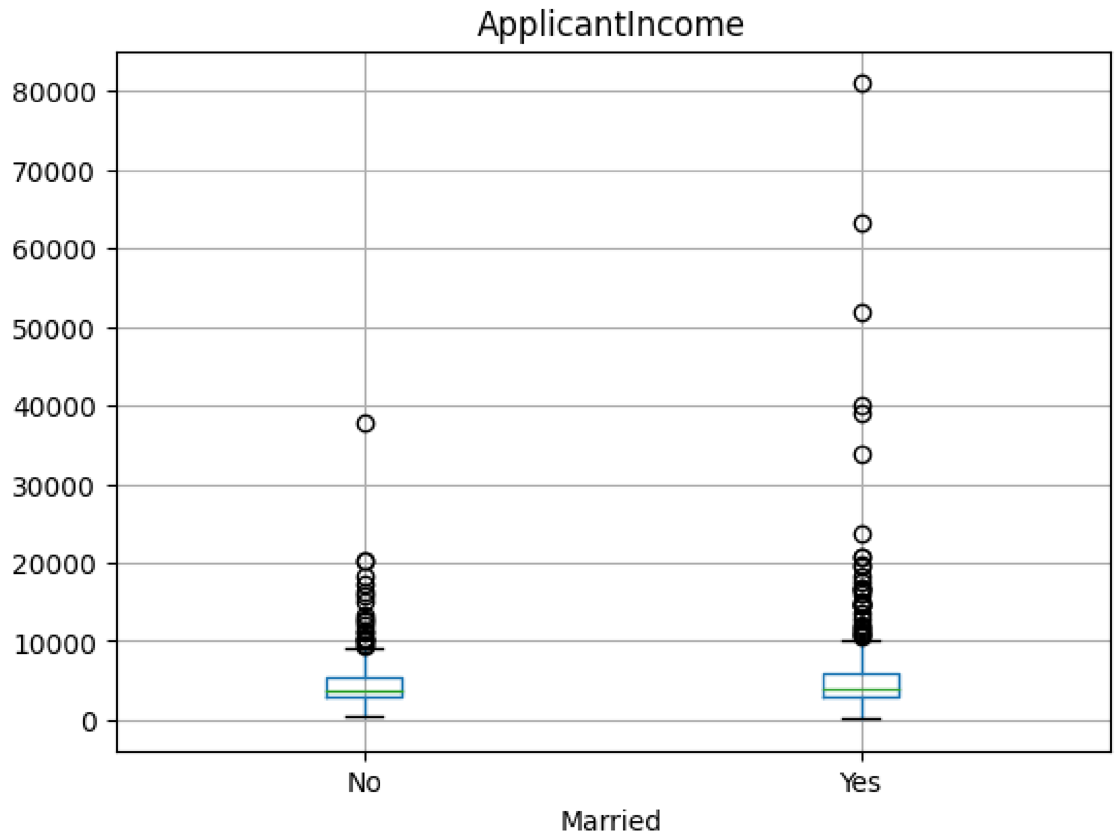
The boxplot confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society. Part of this can be driven by the fact that we are looking at people with different education levels. Let us segregate them by Education:

In [ ]:

```
b.boxplot(column='ApplicantIncome',by='Married')
plt.suptitle("")
```

Out[75]:

```
Text(0.5, 0.98, '')
```



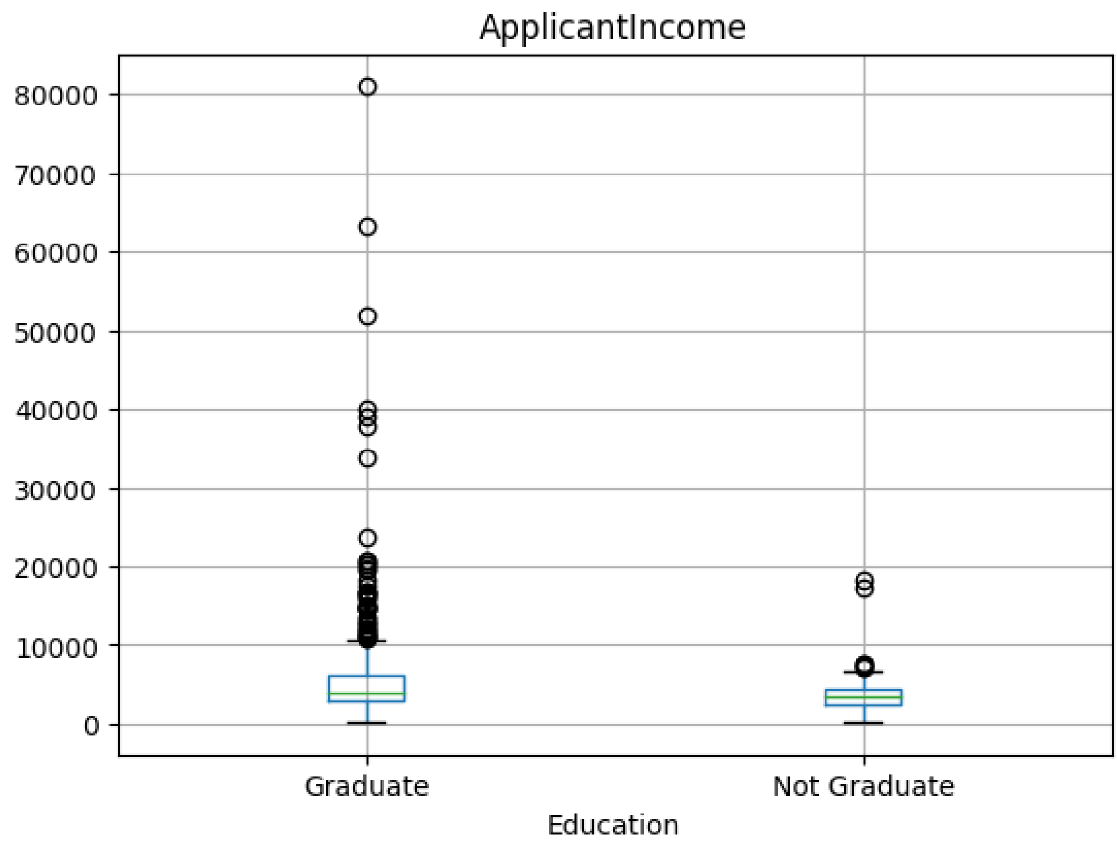
The boxplot confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society. Let us segregate them by Married:

In [ ]:

```
b.boxplot(column='ApplicantIncome',by='Education')
plt.suptitle("")
```

Out[76]:

```
Text(0.5, 0.98, '')
```

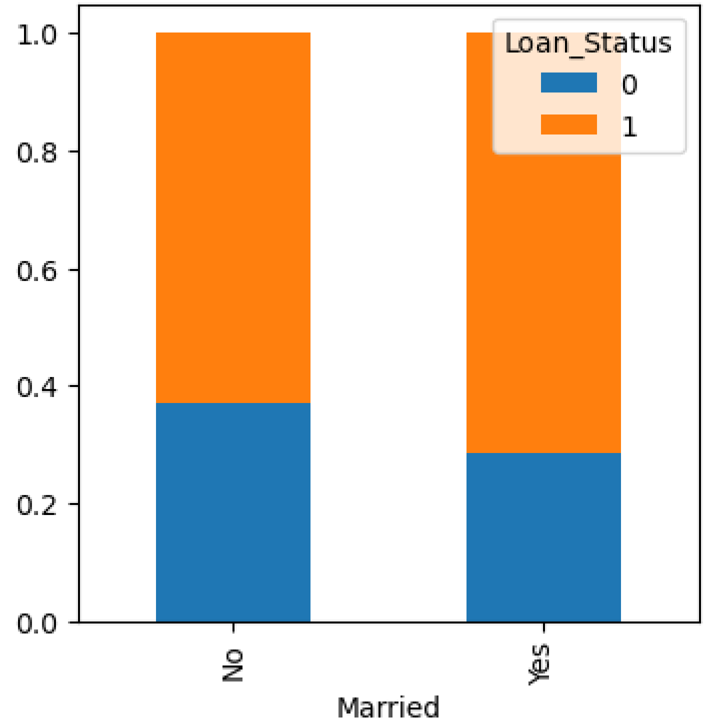


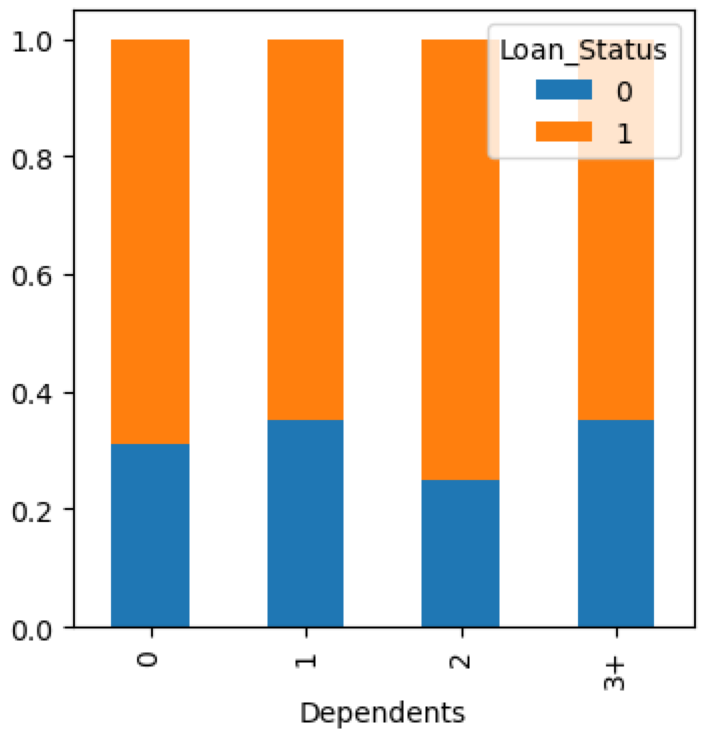
Categorical Independent Variable vs Target Variable

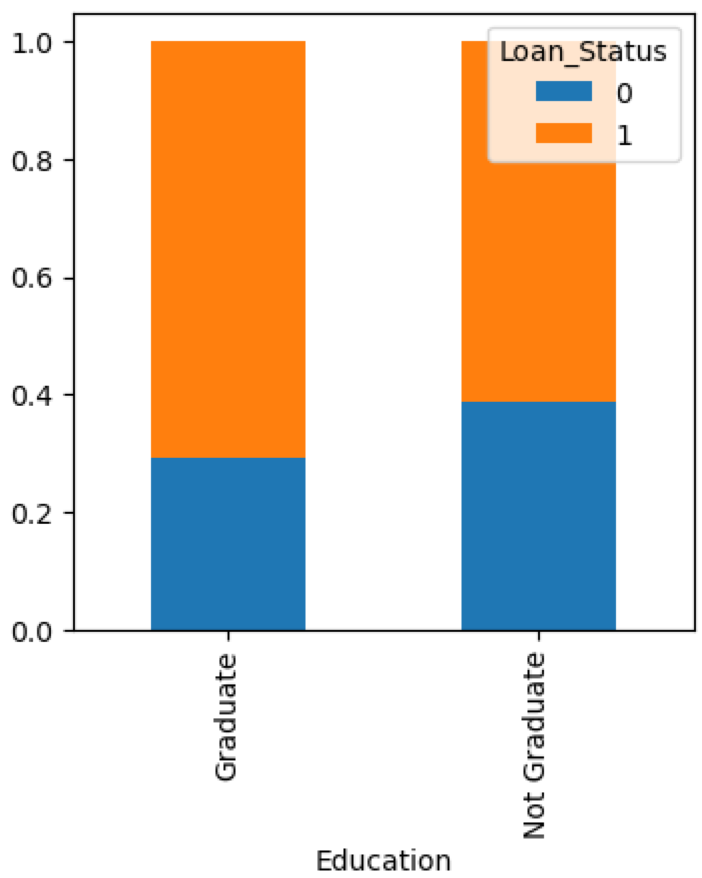
First of all we will find the relation between target variable and categorical independent variables. Let us look at the stacked bar plot now which will give us the proportion of approved and unapproved loans. For example, we want to see whether an applicant's gender will have any effect on approval chances.

In [24]:

```
Gender=pd.crosstab(b['Gender'],b['Loan_Status'])
Married=pd.crosstab(b['Married'],b['Loan_Status'])
Dependents=pd.crosstab(b['Dependents'],b['Loan_Status'])
Education=pd.crosstab(b['Education'],b['Loan_Status'])
Self_Employed=pd.crosstab(b['Self_Employed'],b['Loan_Status'])
Married.div(Married.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(
plt.show()
Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,fig
plt.show()
Education.div(Education.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsi
plt.show()
Self_Employed.div(Self_Employed.sum(1).astype(float), axis=0).plot(kind="bar",stacked=Tr
plt.show()
```



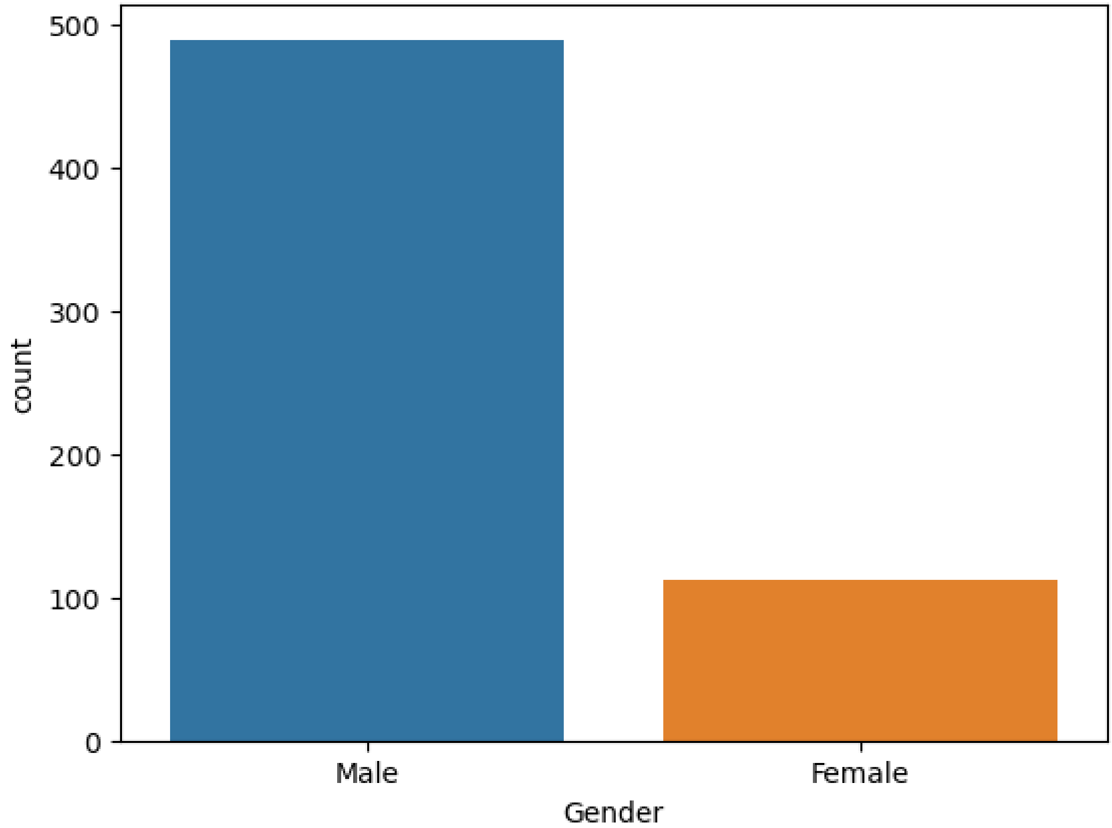




From the bar charts above, it can be inferred that:

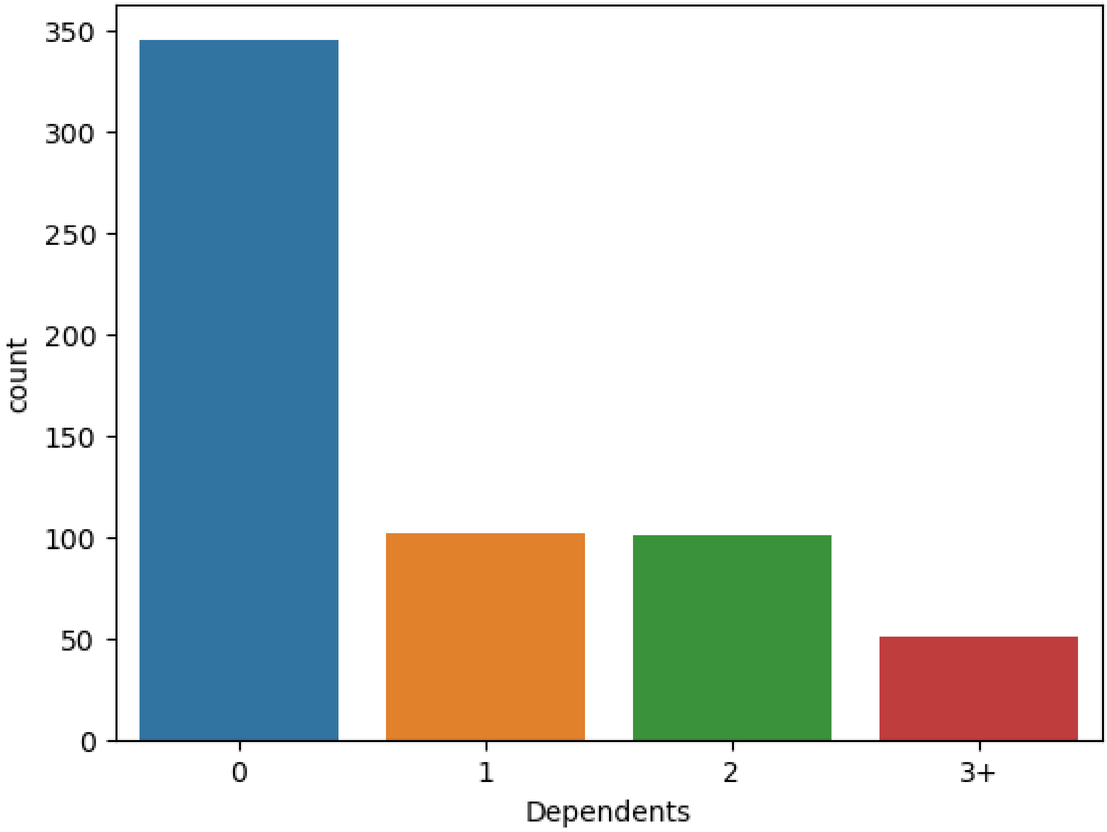
- proportion of male and female applicants is more or less same for both approved and unapproved loans
- proportion of married applicants is higher for the approved loans
- distribution of applicants with 1 or 3+ dependents is similar across both the categories of Loan_Status
- there is nothing significant we can infer from Self_Employed vs Loan_Status plot


```
sns.countplot(x=b['Gender'])
plt.show()
```



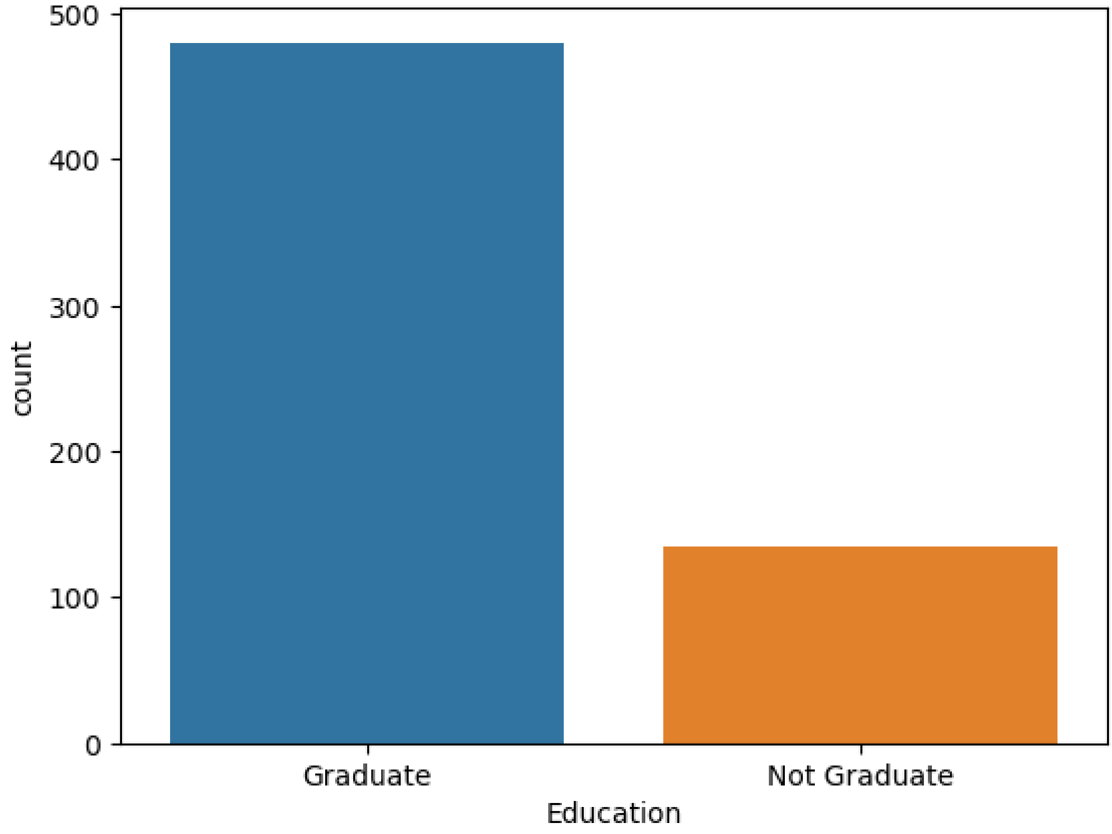
In [45]:

```
sns.countplot(x=b['Dependents'])
plt.show()
```



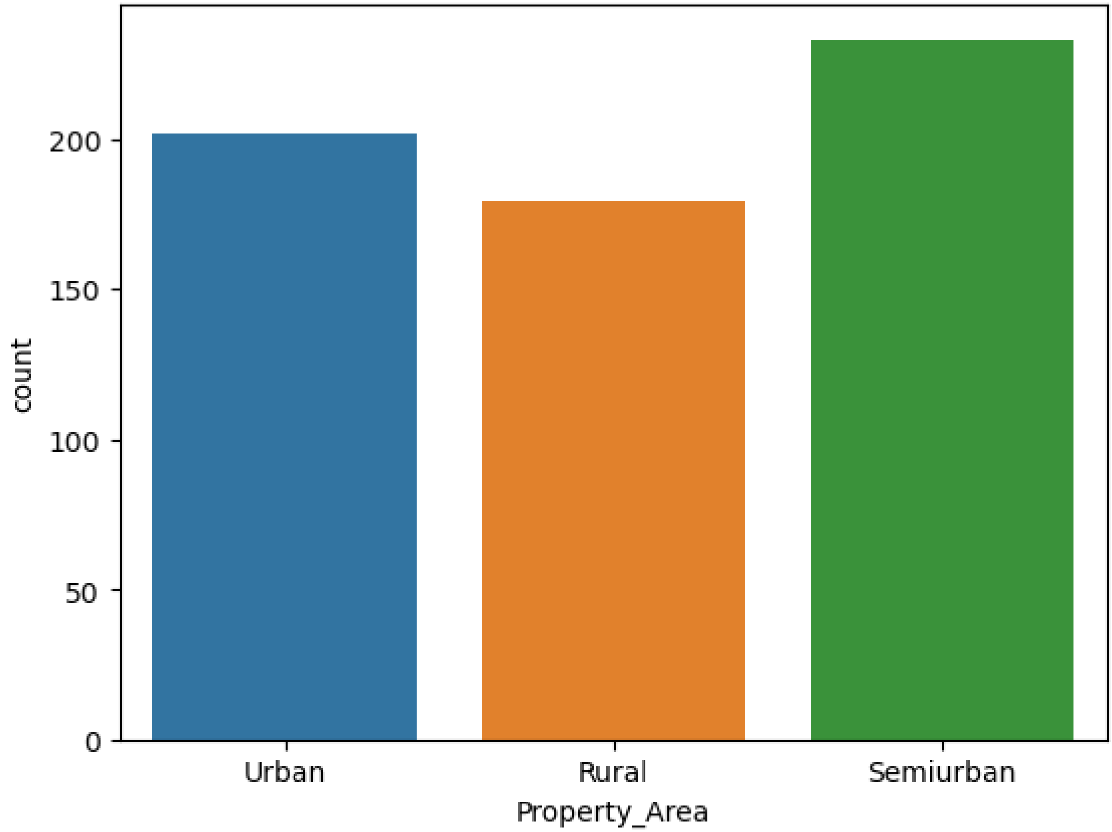
In [46]:

```
sns.countplot(x=b['Education'])
plt.show()
```



In [47]:

```
sns.countplot(x=b['Property_Area'])
plt.show()
```



Numerical Independent Variable vs Target Variable

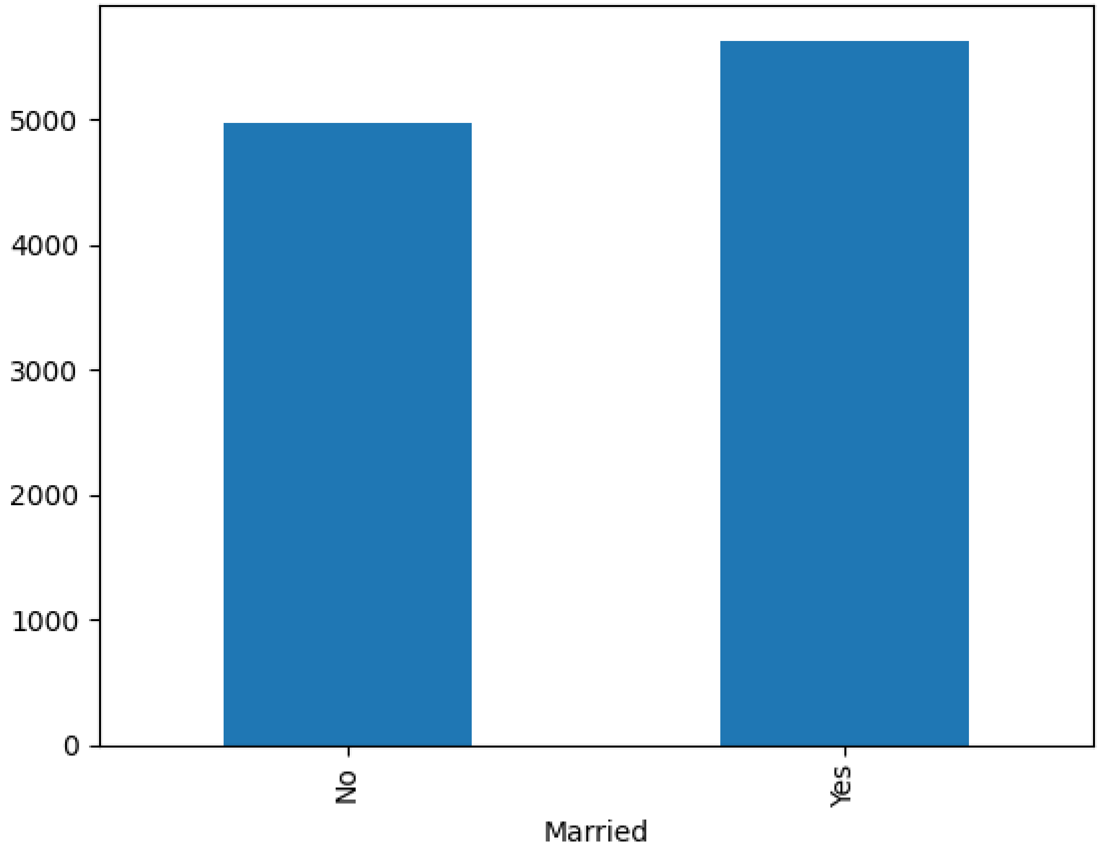
We will try to find the mean income of people for which the loan has been approved vs the mean income of people for which the loan has not been approved.

In [ ]:

```
b.groupby('Married')['ApplicantIncome'].mean().plot(kind='bar')
```

Out[78]:

```
<Axes: xlabel='Married'>
```

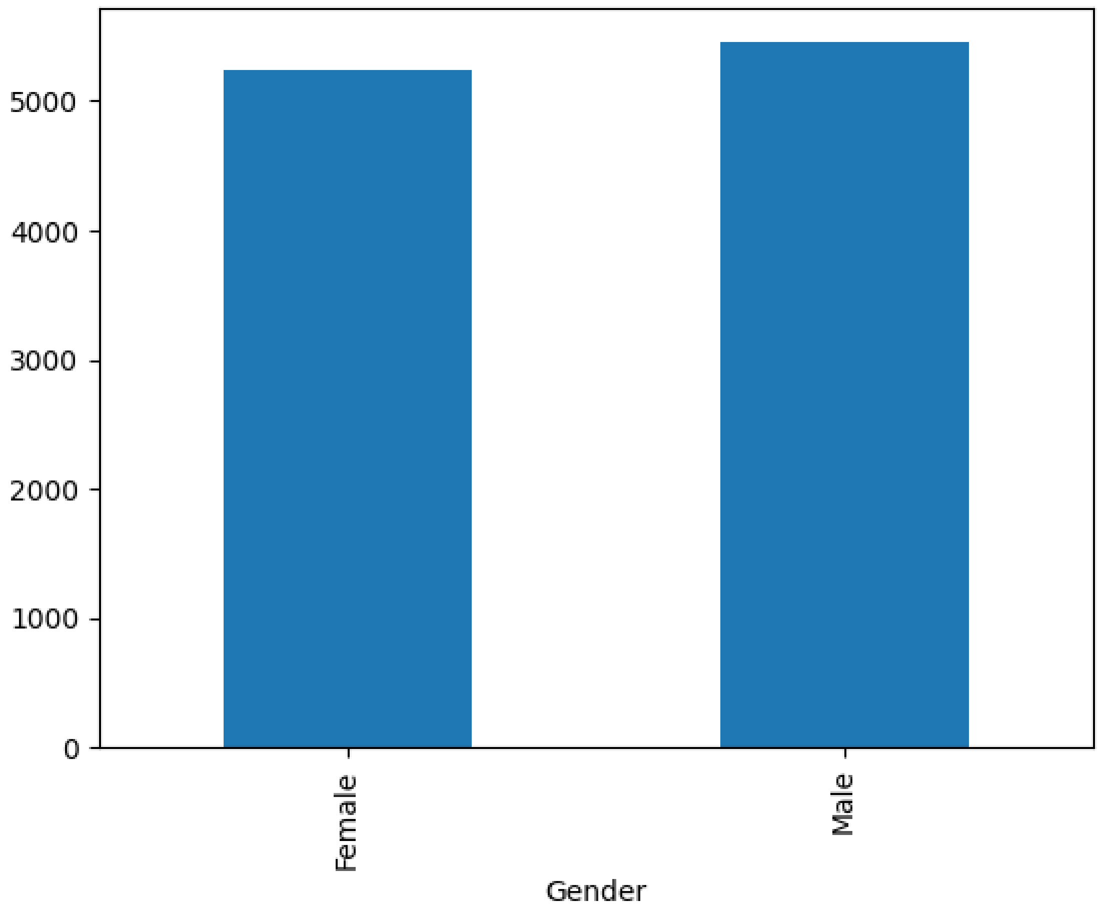


In [ ]:

```
b.groupby('Gender')['ApplicantIncome'].mean().plot(kind='bar')
```

Out[79]:

```
<Axes: xlabel='Gender'>
```

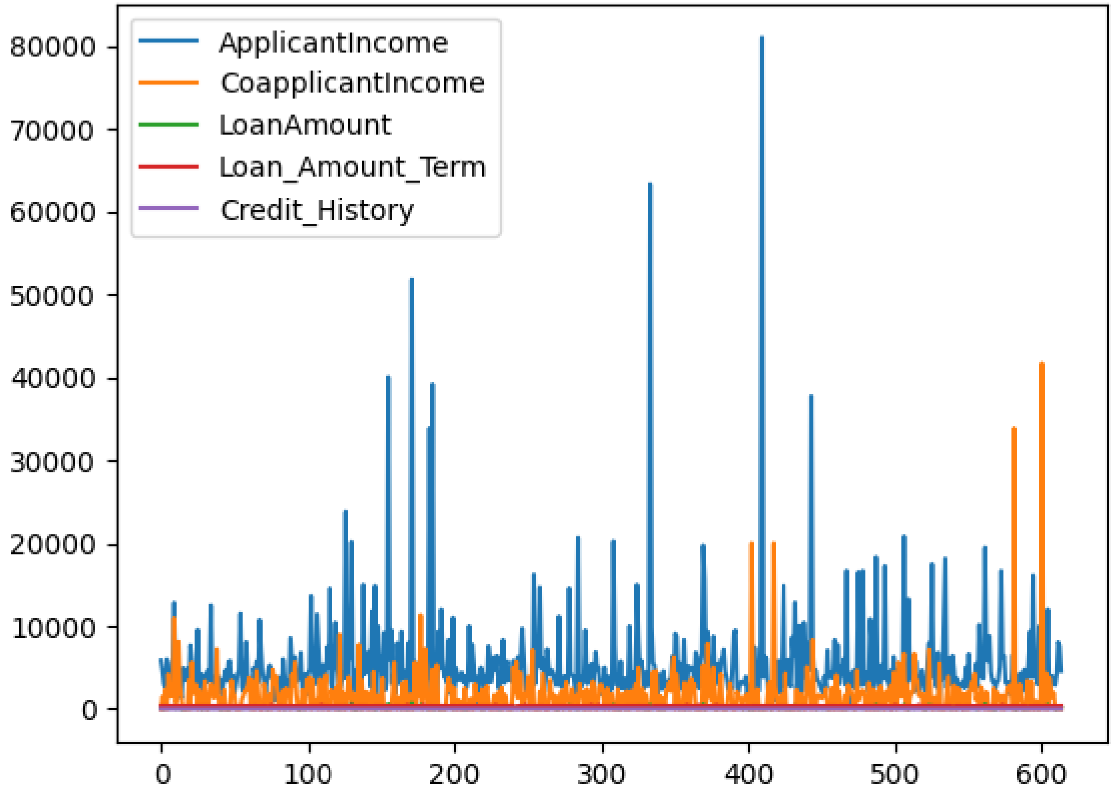


In [ ]:

```
b.plot()
```

Out[80]:

```
<Axes: >
```

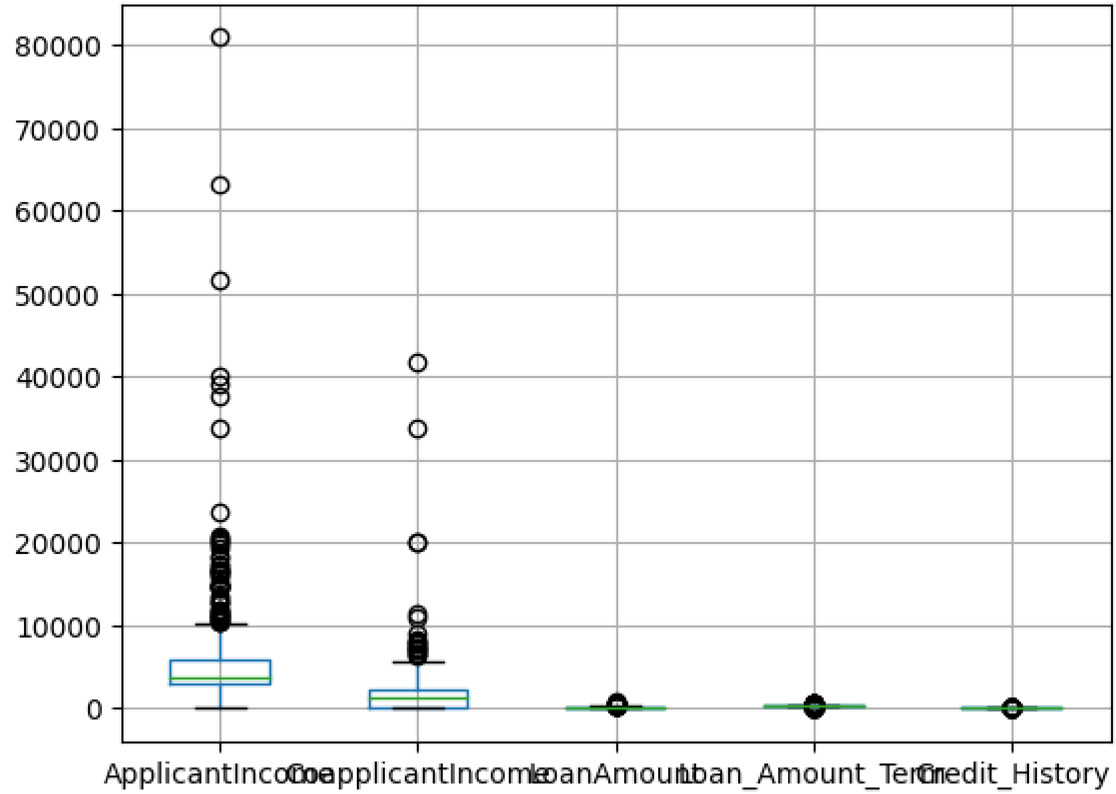


In [ ]:

```
b.boxplot()
```

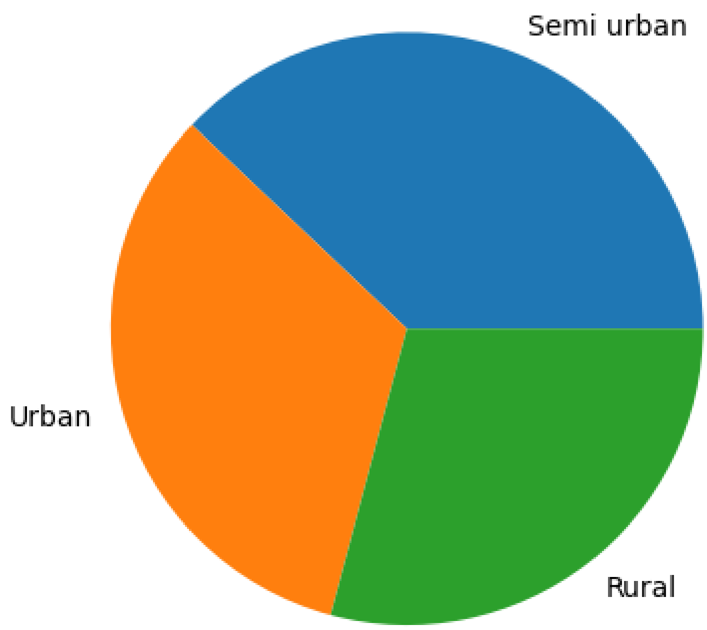
Out[81]:

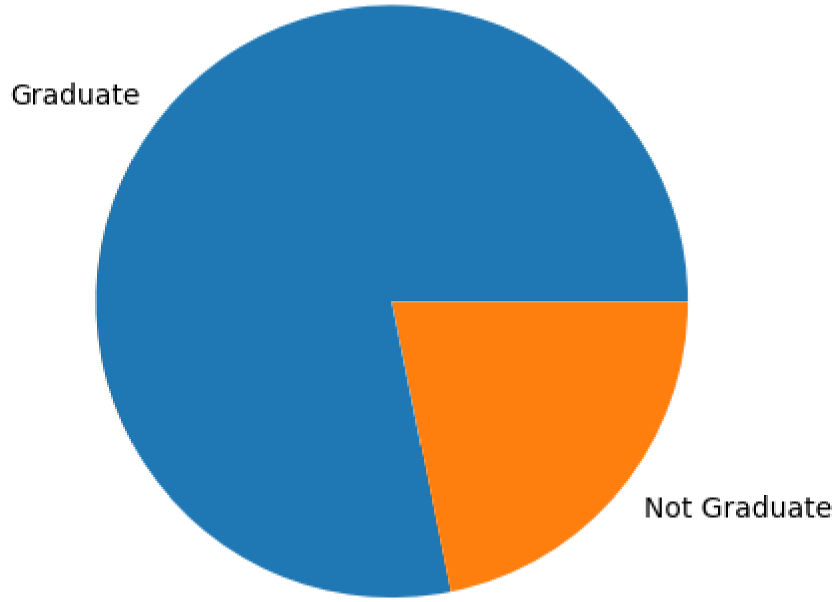
```
<Axes: >
```

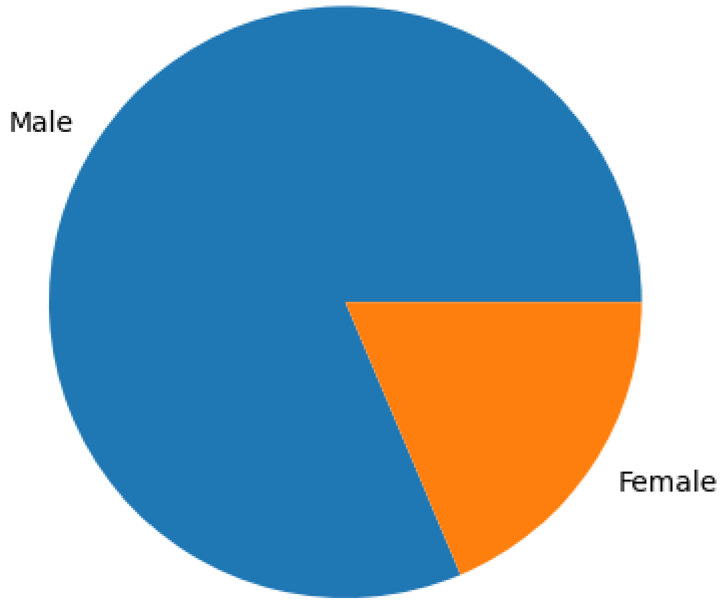


In [30]:

```
plt.pie(b.Property_Area.value_counts(),[0,0,0],labels=['Semi urban','Urban','Rural'])
plt.show()
plt.pie(b.Education.value_counts(),[0,0],labels=['Graduate','Not Graduate'])
plt.show()
plt.pie(b.Gender.value_counts(),[0,0],labels=['Male','Female'])
plt.show()
plt.pie(b.Self_Employed.value_counts(),[0,0],labels=['No','Yes'])
```

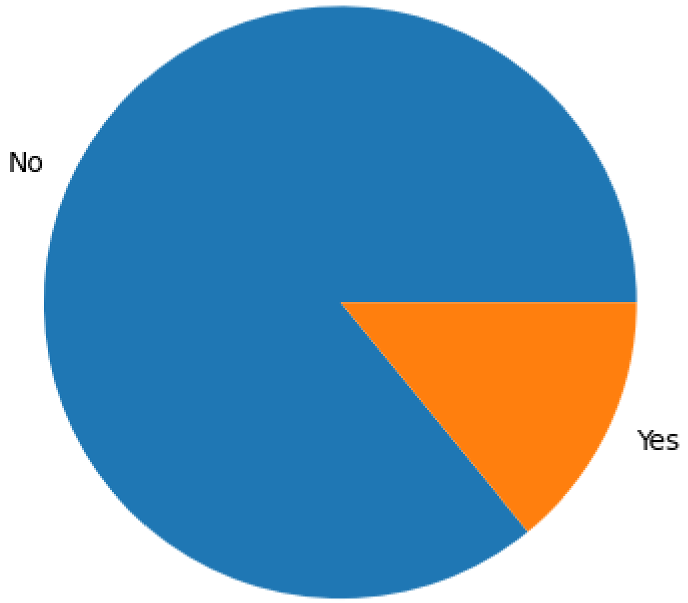






Out[30]:

```
([<matplotlib.patches.Wedge at 0x12e2f60b5b0>,
  <matplotlib.patches.Wedge at 0x12e2f60b4c0>],
 [Text(-0.9939912136472331, 0.47114909231802693, 'No'),
  Text(0.9939912357033128, -0.47114904578593964, 'Yes')])
```



In [ ]:

```
b.corr()
```

Out[82]:

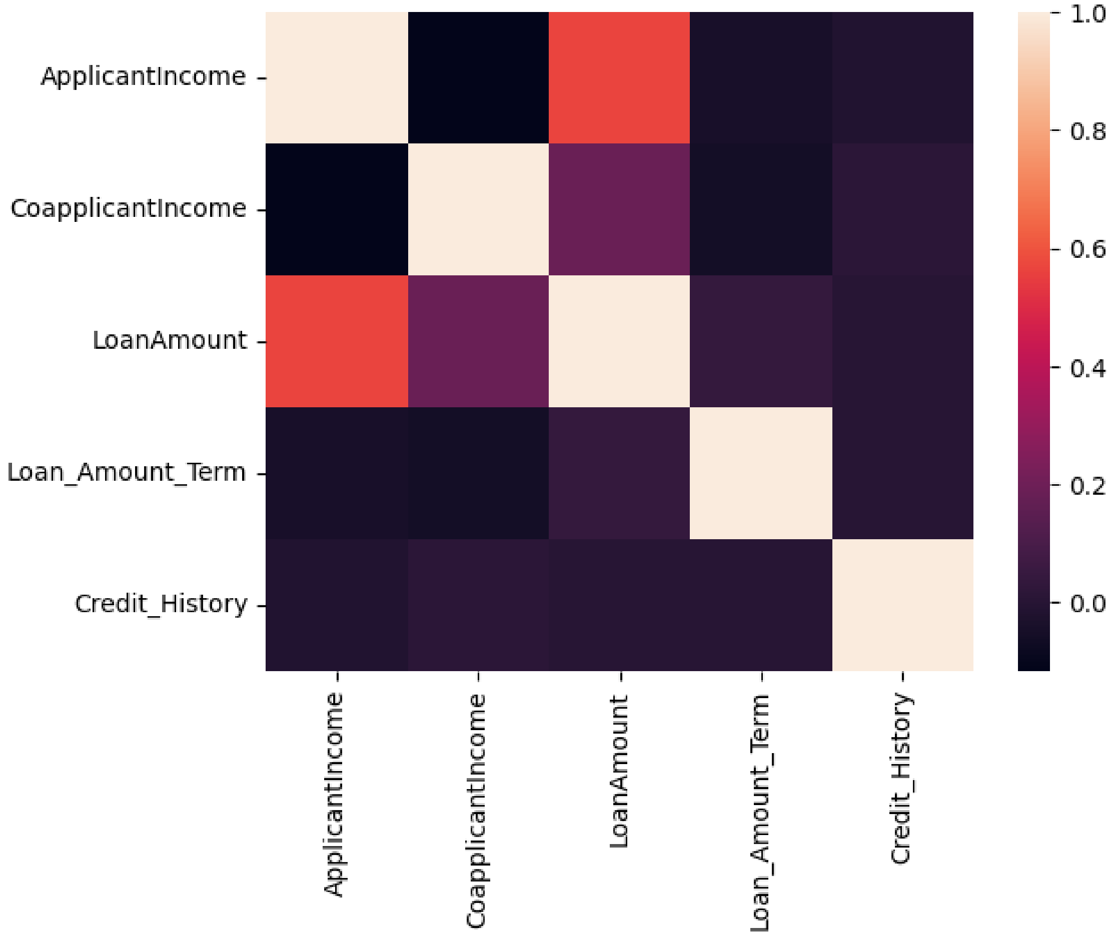
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
ApplicantIncome	1.000000	-0.116605	0.565490	-0.045281
CoapplicantIncome	-0.116605	1.000000	0.188643	-0.059668
LoanAmount	0.565490	0.188643	1.000000	0.038984
Loan_Amount_Term	-0.045281	-0.059668	0.038984	1.000000
Credit_History	-0.018615	0.011134	-0.000971	0.000278

In [ ]:

```
sns.heatmap(b.corr())
```

Out[83]:

<Axes: >



In [ ]:

```
print(len(b[b["ApplicantIncome"] == 0]))
"Percentage of ApplicantIncome = 0 is:",  len(b[b["ApplicantIncome"] == 0])/len(b["Appli
```

```
0
```

Out[86]:

```
('Percentage of ApplicantIncome = 0 is:', 0.0)
```

In [ ]:

```
print(len(b[b["CoapplicantIncome"] == 0]))
"Percentage of CoapplicantIncome = 0 is:",  len(b[b["CoapplicantIncome"] == 0])/len(b["C
```

```
273
```

Out[87]:

```
('Percentage of CoapplicantIncome = 0 is:', 0.44462540716612375)
```

In [ ]:

```
# calculate and visualize correlation matrix
matrix=b.corr()
f, ax=plt.subplots(figsize=(17,8))
sns.heatmap(matrix, vmax=1, square=True, cmap="BuPu", annot=True)

matrix
```

Out[88]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
ApplicantIncome	1.000000	-0.116605	0.565490	-0.045281
CoapplicantIncome	-0.116605	1.000000	0.188643	-0.059668
LoanAmount	0.565490	0.188643	1.000000	0.038984
Loan_Amount_Term	-0.045281	-0.059668	0.038984	1.000000
Credit_History	-0.018615	0.011134	-0.000971	0.000278

Note : We see that the most correlated variables are

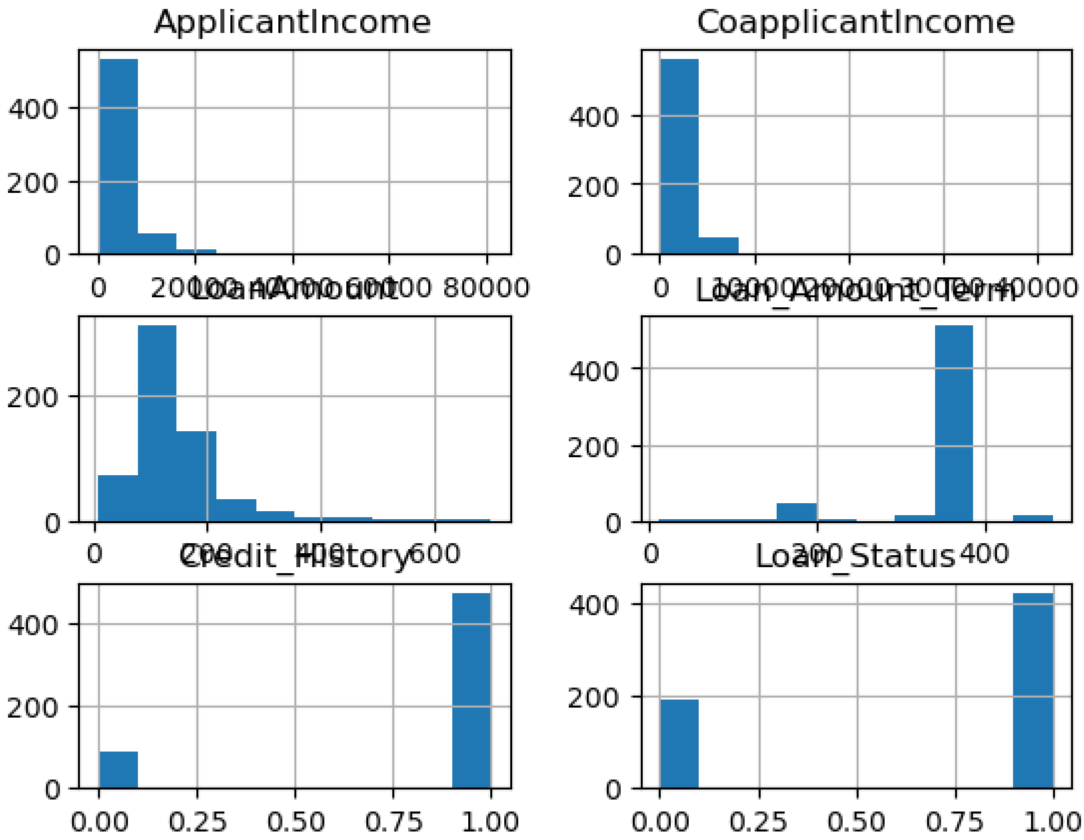
- (ApplicantIncome - LoanAmount) with correlation coefficient of 0.57
- (Credit_History - Loan_Status) with correlation coefficient of 0.56
- LoanAmount is also correlated with CoapplicantIncome with correlation coefficient of 0.19.

In [33]:

```
b.hist()
```

Out[33]:

```
array([[<Axes: title={'center': 'ApplicantIncome'}>,
        <Axes: title={'center': 'CoapplicantIncome'}>],
       [<Axes: title={'center': 'LoanAmount'}>,
        <Axes: title={'center': 'Loan_Amount_Term'}>],
       [<Axes: title={'center': 'Credit_History'}>,
        <Axes: title={'center': 'Loan_Status'}>]], dtype=object)
```

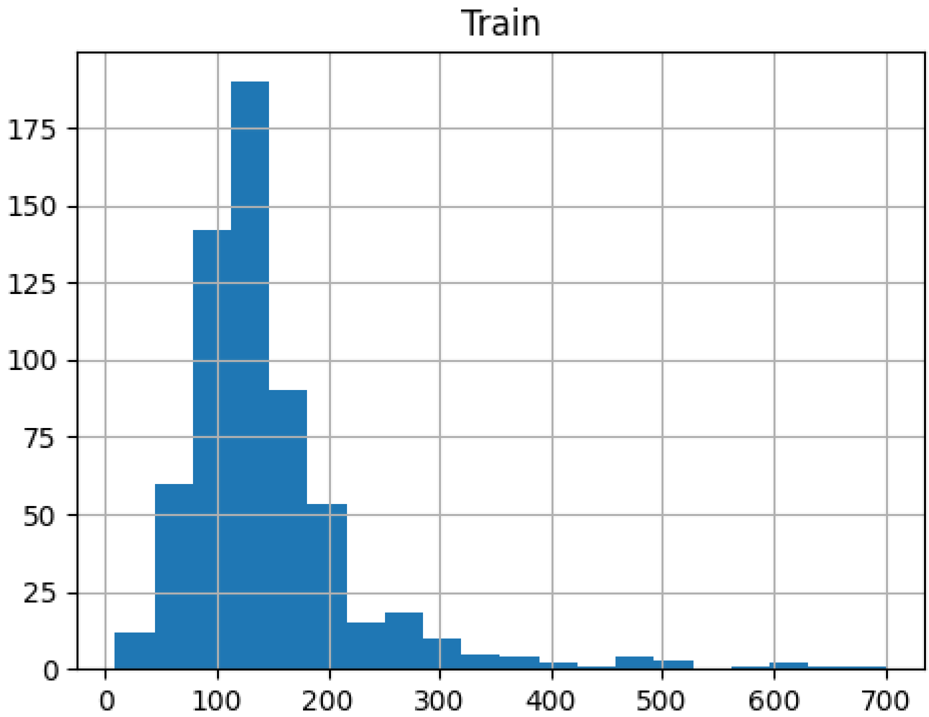


In [ ]:

```
ax1 = plt.subplot(121)
b['LoanAmount'].hist(bins=20, figsize=(12,4))
ax1.set_title("Train")
```

Out[89]:

```
Text(0.5, 1.0, 'Train')
```

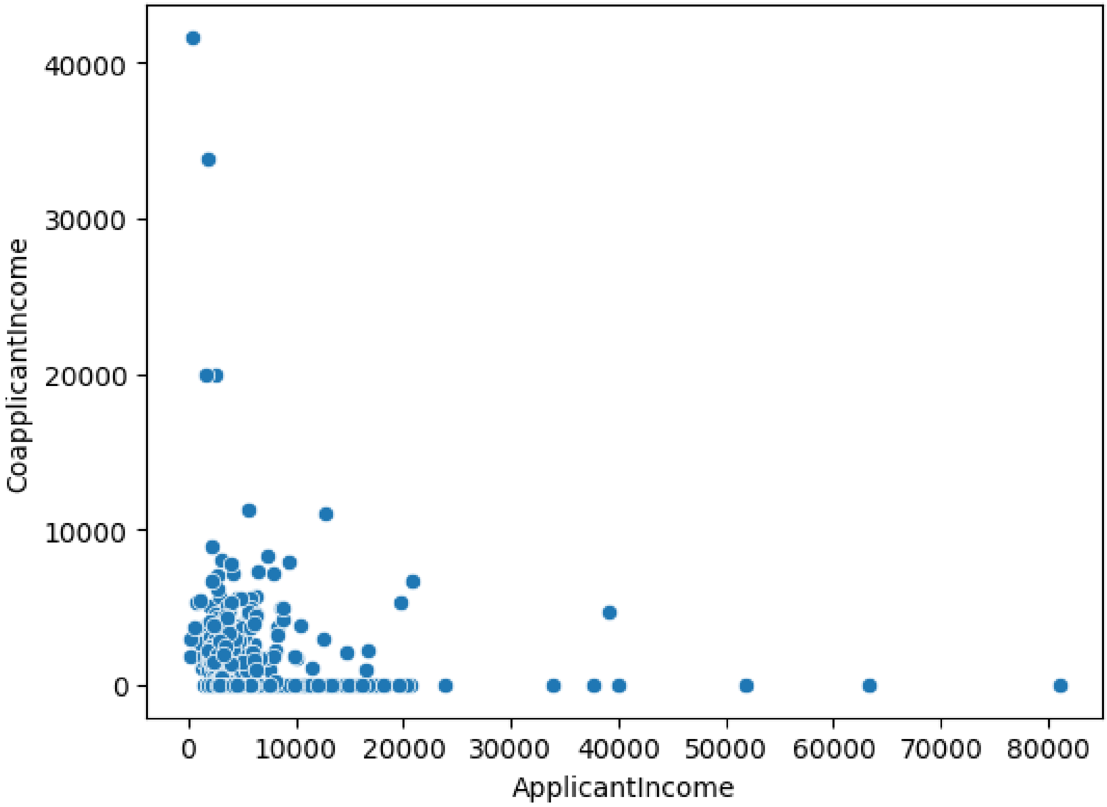


In [ ]:

```
sns.scatterplot(x='ApplicantIncome', y='CoapplicantIncome', data=b)
```

Out[91]:

```
<Axes: xlabel='ApplicantIncome', ylabel='CoapplicantIncome'>
```

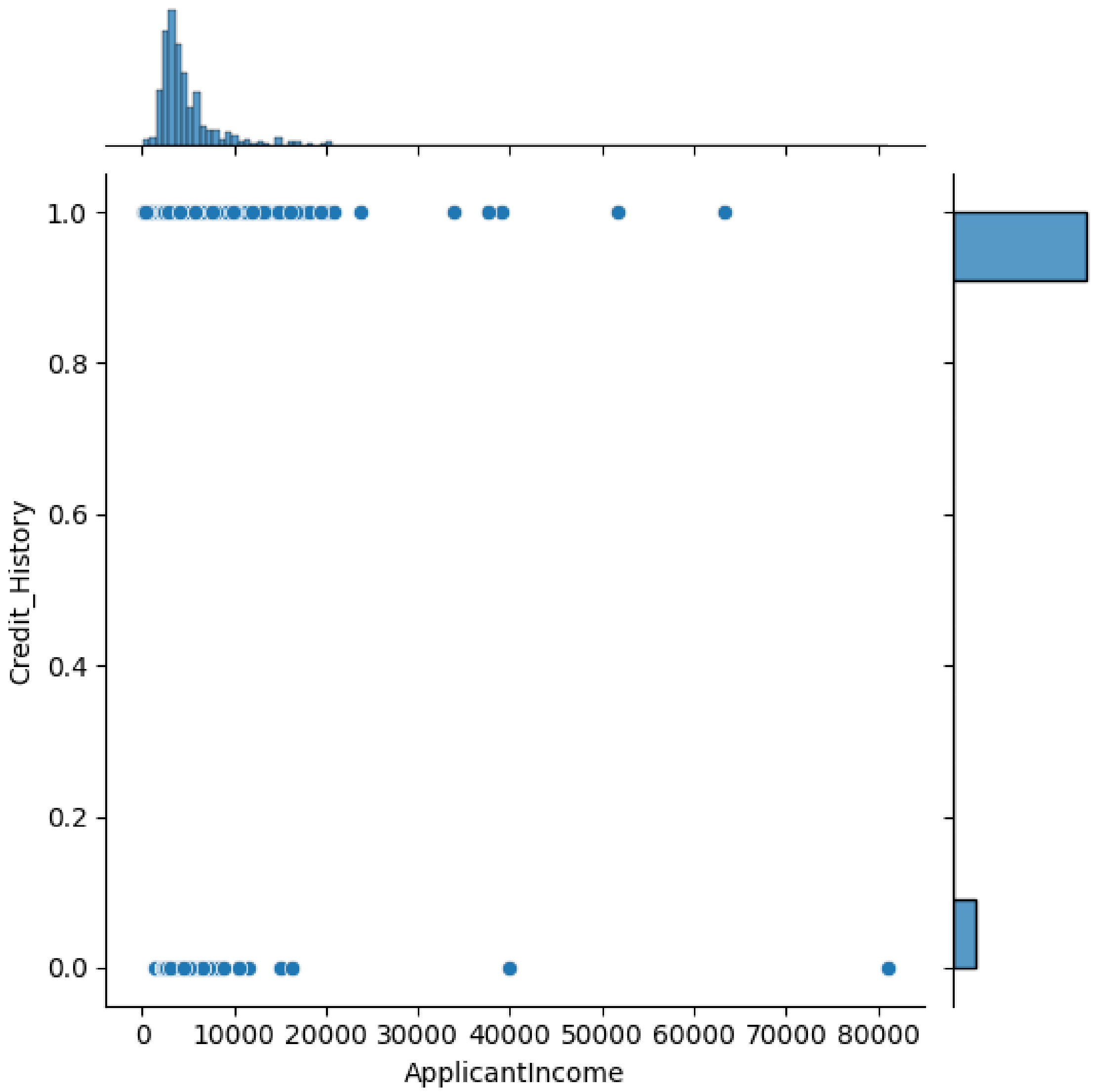


In [ ]:

```
sns.jointplot(x='ApplicantIncome', y='Credit_History', data=b)
```

Out[94]:

```
<seaborn.axisgrid.JointGrid at 0x7fb24a795cd0>
```

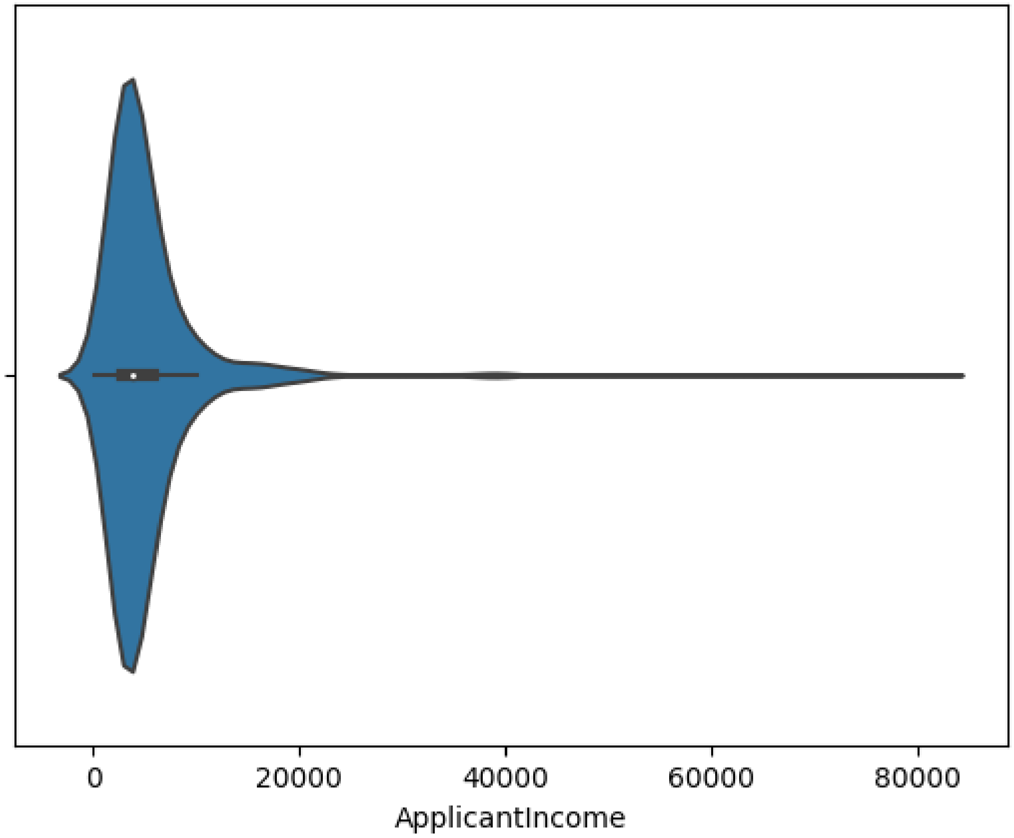


In [ ]:

```
sns.violinplot(x='ApplicantIncome', data=b)
```

Out[95]:

```
<Axes: xlabel='ApplicantIncome'>
```

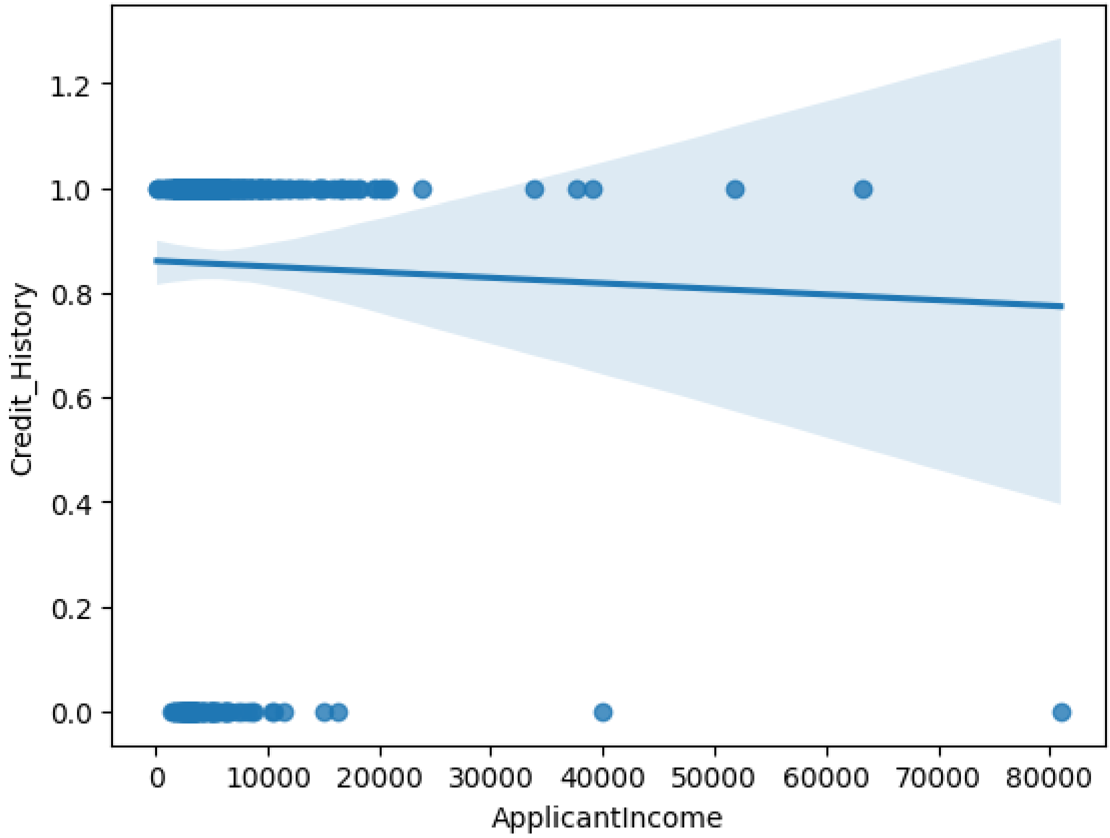


In [ ]:

```
sns.regplot(x='ApplicantIncome', y='Credit_History', data=b)
```

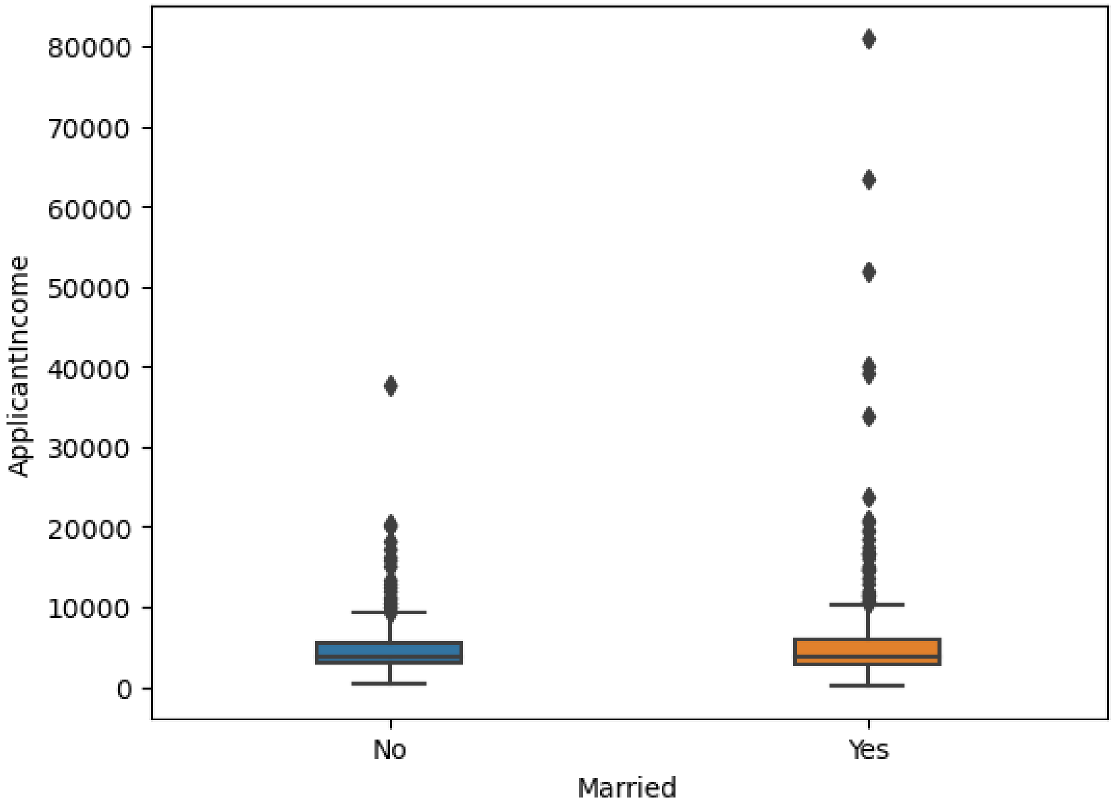
Out[96]:

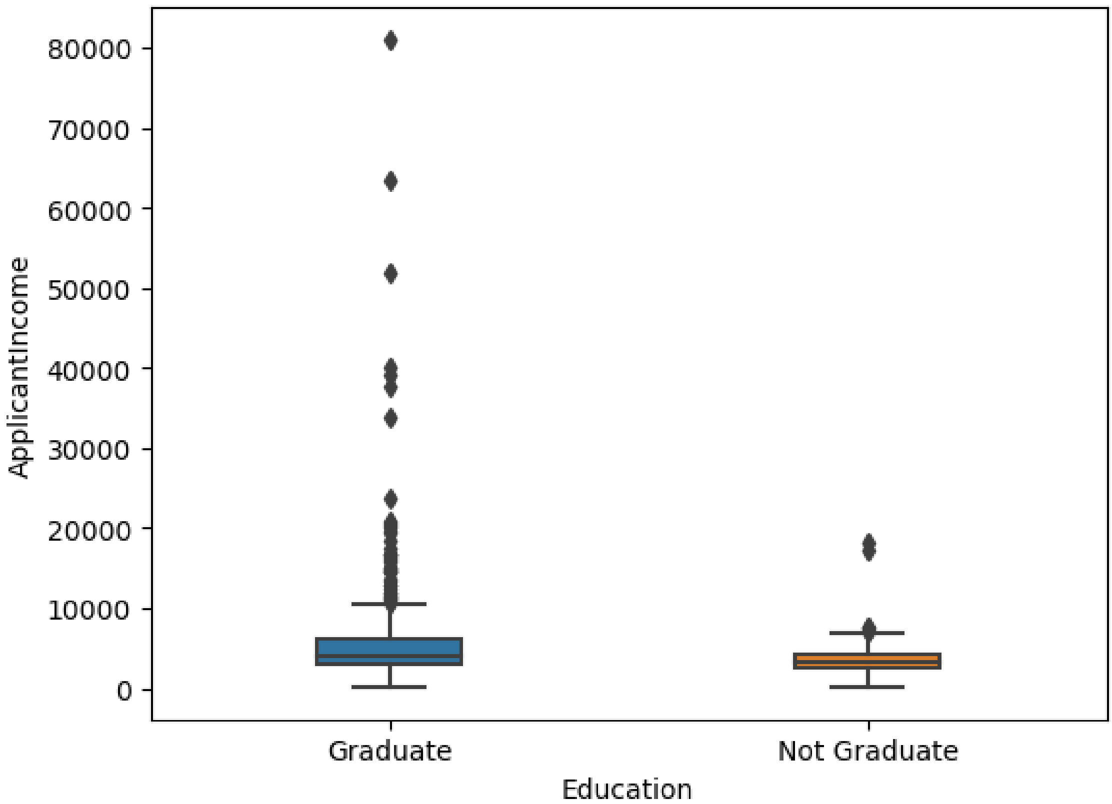
```
<Axes: xlabel='ApplicantIncome', ylabel='Credit_History'>
```

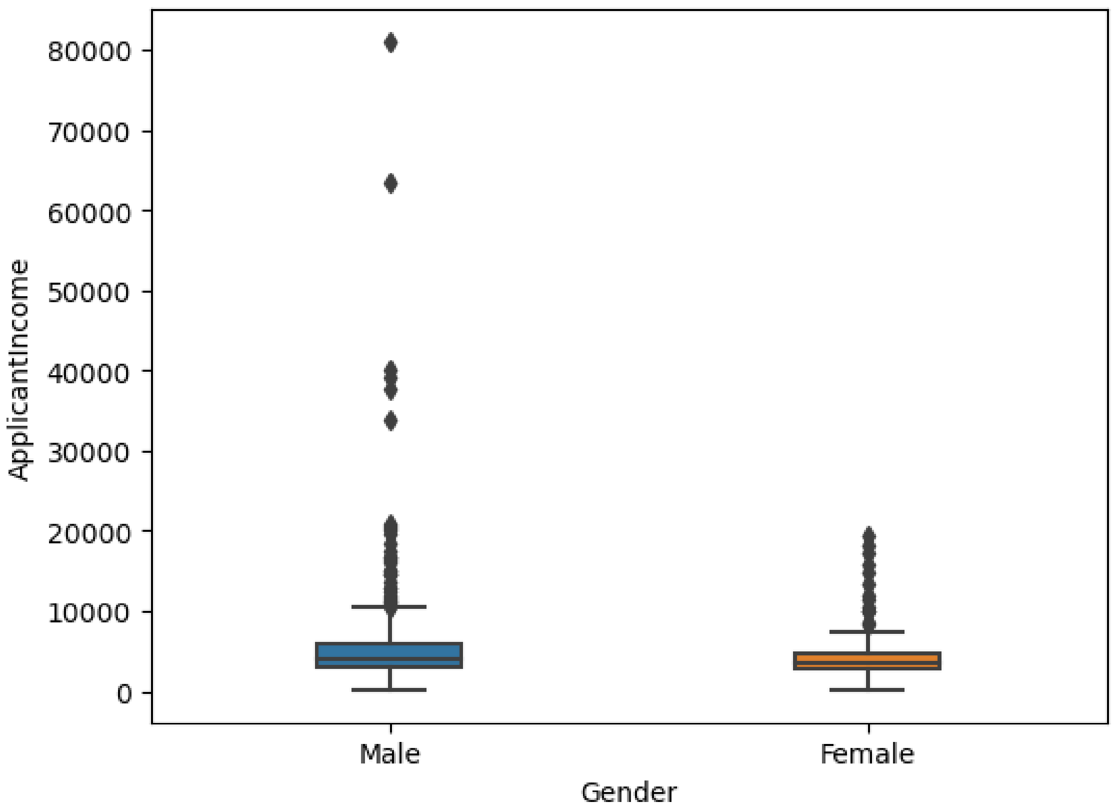


In [ ]:

```
sns.boxplot(x='Married',y='ApplicantIncome',data=b,width=0.3)
plt.show()
sns.boxplot(x='Education',y='ApplicantIncome',data=b,width=0.3)
plt.show()
sns.boxplot(x='Gender',y='ApplicantIncome',data=b,width=0.3);
plt.show()
```







80000
70000
60000
50000
40000
30000
20000
10000
0
ApplicantIncome
Male
Female
Gender

In [ ]:

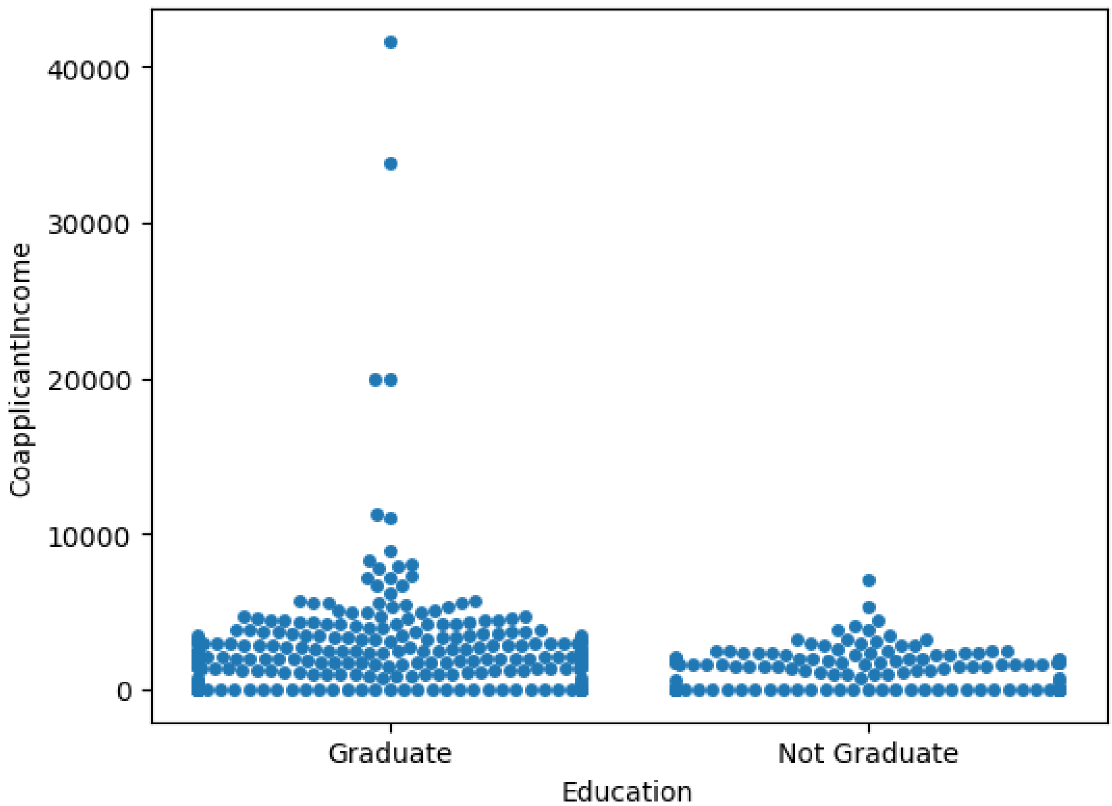
```
sns.swarmplot(x='Education', y='CoapplicantIncome', data=b)
```

```
C:\ProgramData\anaconda3\lib\site-packages\seaborn\categorical.py:3544: U
serWarning: 45.8% of the points cannot be placed; you may want to decreas
e the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\ProgramData\anaconda3\lib\site-packages\seaborn\categorical.py:3544: U
serWarning: 5.2% of the points cannot be placed; you may want to decrease
the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

Out[33]:

```
<Axes: xlabel='Education', ylabel='CoapplicantIncome'>
```

```
C:\ProgramData\anaconda3\lib\site-packages\seaborn\categorical.py:3544: U
serWarning: 61.3% of the points cannot be placed; you may want to decreas
e the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\ProgramData\anaconda3\lib\site-packages\seaborn\categorical.py:3544: U
serWarning: 28.4% of the points cannot be placed; you may want to decreas
e the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

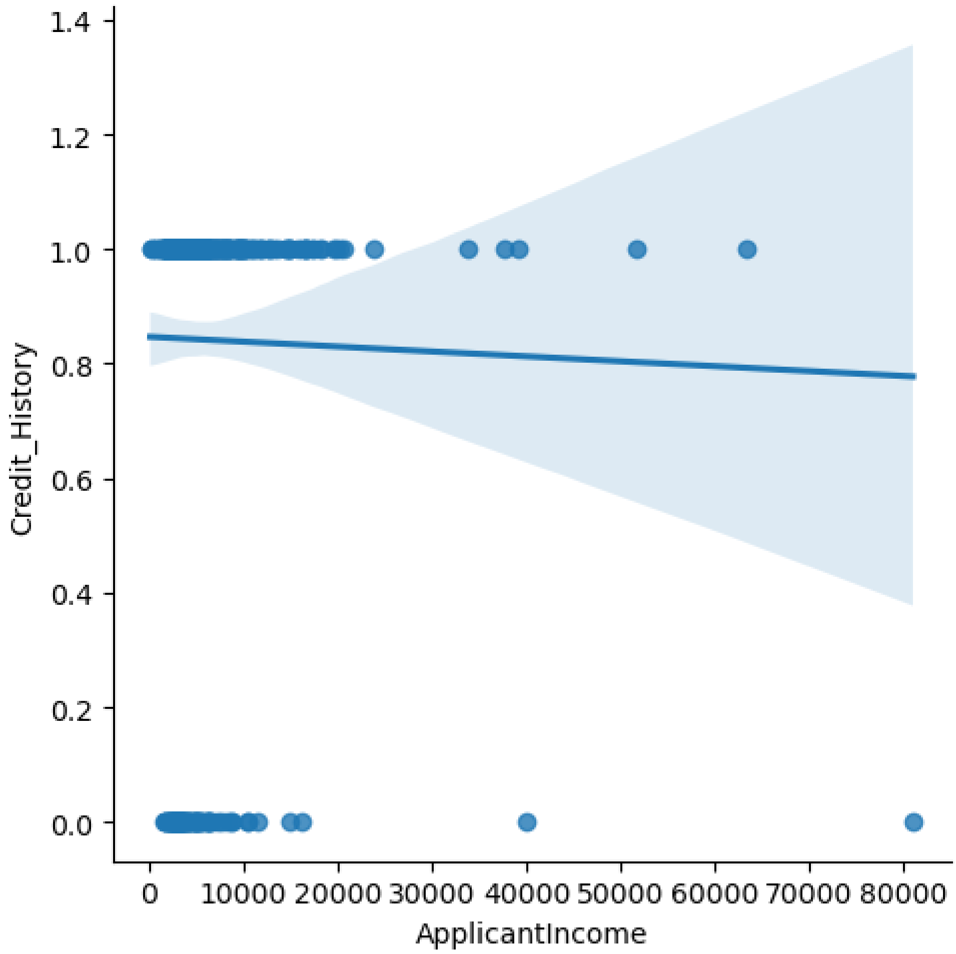


In [ ]:

```
sns.lmplot(x='ApplicantIncome',y='Credit_History',data=b)
```

Out[35]:

```
<seaborn.axisgrid.FacetGrid at 0x2093fb535e0>
```



Model Buliding : Part 1

Let us make our first model to predict the target variable. We will start with Logistic Regression which is used for predicting binary outcome.

- Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.
- Logistic regression is an estimation of Logit function. Logit function is simply a log of odds in favor of the event.
- This function creates a s-shaped curve with the probability estimate, which is very similar to the required step wise function

Predicting output Y column from Loan Prediction of Train Dataset

Lets drop the Loan_ID variable as it do not have any effect on the loan status. We will do the same changes to the test dataset which we did for the training dataset.

In [5]:

```
# replacing Y and N in Loan_Status variable with 1 and 0 respectively
b['Loan_Status'].replace('N', 0, inplace=True)
b['Loan_Status'].replace('Y', 1, inplace=True)
```

In [6]:

```
# drop Loan_ID
train = b.drop('Loan_ID', axis=1)
```

In [7]:

```
X = b.drop('Loan_Status', 1)
y = b.Loan_Status
```

```
C:\Users\meruv\AppData\Local\Temp\ipykernel_16028\1397215060.py:1: Future
Warning: In a future version of pandas all arguments of DataFrame.drop ex
cept for the argument 'labels' will be keyword-only.
  X = b.drop('Loan_Status', 1)
```

In [8]:

```
train.shape
```

Out[8]:

```
(614, 12)
```

Let us understand the process of dummies first:

- Consider the “Gender” variable. It has two classes, Male and Female.
- As logistic regression takes only the numerical values as input, we have to change male and female into numerical value.
- Once we apply dummies to this variable, it will convert the “Gender” variable into two variables(Gender_Male and Gender_Female), one for each class, i.e. Male and Female.
- Gender_Male will have a value of 0 if the gender is Female and a value of 1 if the gender is Male.

We can use pandas **get_dummies** function to convert categorical variable into dummy/indicator variables, it will only convert "object" type and will not affect numerical type.

In [9]:

```
# adding dummies to the dataset
X = pd.get_dummies(X)
train = pd.get_dummies(b)
```

In [10]:

```
X.head()
```

Out[10]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	L
0	5849	0.0	NaN	360.0	1.0	
1	4583	1508.0	128.0	360.0	1.0	
2	3000	0.0	66.0	360.0	1.0	
3	2583	2358.0	120.0	360.0	1.0	
4	6000	0.0	141.0	360.0	1.0	

5 rows × 634 columns

In [11]:

```
y
```

Out[11]:

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int64
```

Result:-

Predicting output Y column from Loan Prediction of Train Dataset.
Y will be Loan_Status

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
```

Conclusion:-

- We did Exploratory data analysis on the features of this dataset and saw how each feature is distributed.
- We did bivariate and multivariate analysis to see impact of one another on their features using charts.
- We cleaned the data and removed NA values.
- We can also make independent vs independent variable visualizations to discover some more patterns.
- We calculated correlation between independent variables and found that applicant income and loan amount have significant value.
- We created dummy variable for constructing model.

References:-

- https://towardsdatascience.com/data-types-in-statistics-347e152e8bee
- https://machinelearning-blog.com/2018/04/23/logistic-regression-101/
- https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/
- https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/