project goal:

The goal of this project is differentiate between people who would default the loan or not based on data provided. It help to give the chance to people who would complete the payment.

Dataset description:

Train Dataset:

#	Feature	Data Type	Description
0	Loan_ID	Text	A unique loan ID
1	Gender	categorical - text	Male / Female
2	Married	categorical - text	Married (Yes) / Not married (No)
3	Dependents	categorical - text	Number of people depending on the client (Applicant)
4	Education	categorical - text	Graduate / Ungraduate
5	Self_Employed	categorical - text	Yes / No
6	ApplicantIncome	Number (integer)	Income of client (Applicant)
7	CoapplicantIncome	Number (float)	Income of Co-applicant (additional person involved in the loan application process.)
8	LoanAmount	Number (float)	Amount of loan in thousands
9	Loan_Amount_Term	Number (float)	record of a borrower's responsible repayment of debts
10	Credit_History	Number (float)	Credit history (record of a borrower's responsible repayment of debts) that meets guidelines
11	Property_Area	categorical - text	Urban / Semi / Rural
12	Loan_Status	categorical - text	Approved (yes) / Not Approves (NO)

Size:

614 entries

13 columns

Test Dataset:

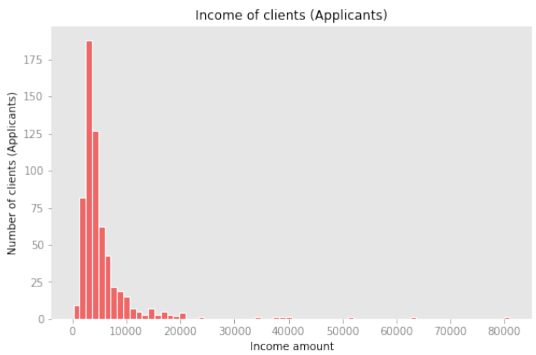
Same as train data set but without Loan_Status

Size:

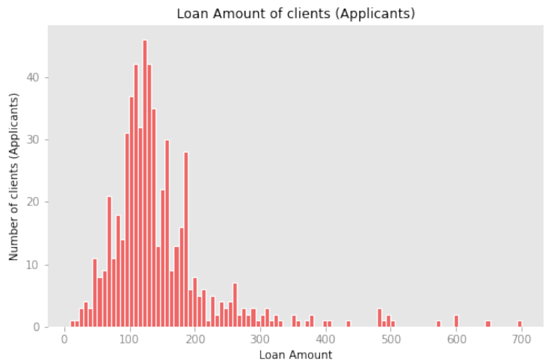
367 entries

12 columns

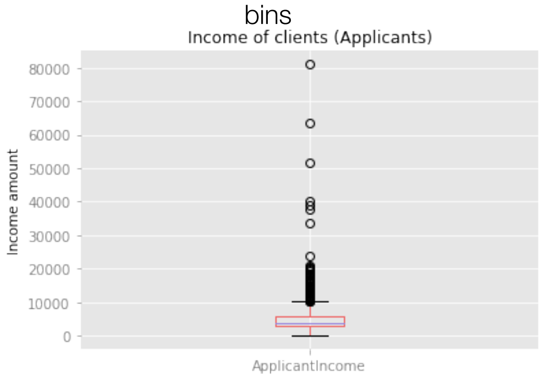
statistical and graphical presentation of the dataset:



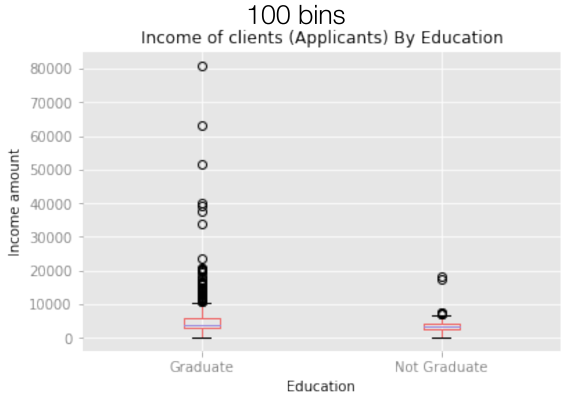
Histogram that shows incomes divided in 70 bins



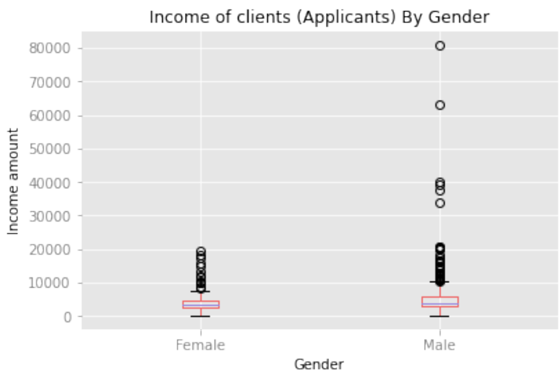
Histogram that shows loan amount divided in 100 bins



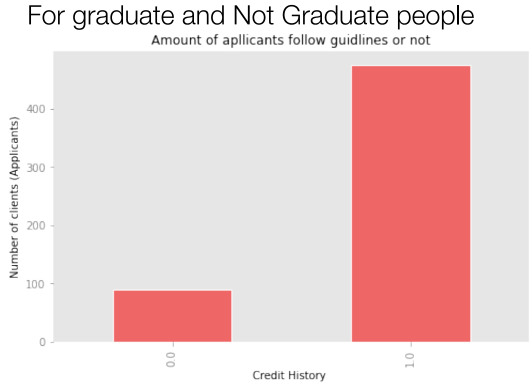
Box Blot shows the real mean of of incomes



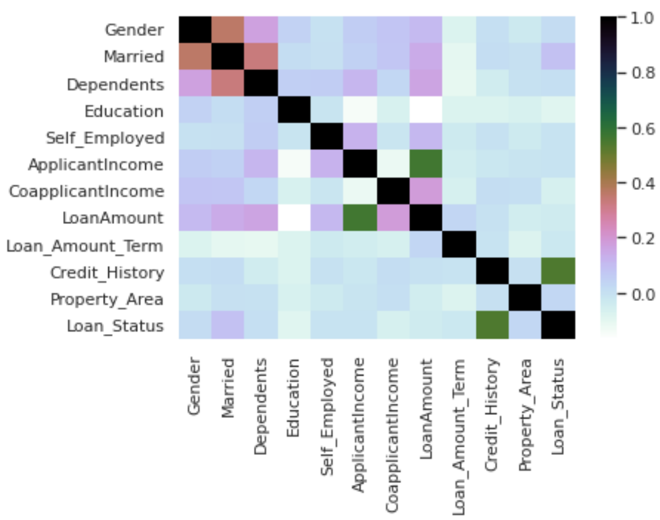
Box Blot shows the real mean of of incomes, For graduate and Not Graduate people



Box Blot shows the real mean of of incomes, For Female and Male



Bar chart for credit history shows the amount that meet the guidelines and the one is not.



Correlation matrix

Dataset preprocessing:

- Handle missing values
- Label Encoding
- Drop unaffected feature

Machine Learning Algorithms:

- **Decision Tree**: flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.[1]
- **Logistic regression**: supervised learning classification algorithm used to predict the probability of a target variable.[2]
- **Random Forest**: a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.[3]

Results:

- **Decision Tree**:

	precision	recall	f1-score	support
0	0.44	0.58	0.50	45
1	0.85	0.76	0.80	140
accuracy			0.72	185
macro avg	0.64	0.67	0.65	185
weighted avg	0.75	0.72	0.73	185

71.89% Accurate

- **Logistic regression**:

	precision	recall	f1-score	support
0	0.85	0.49	0.62	45
1	0.86	0.97	0.91	140
accuracy			0.85	185
macro avg	0.85	0.73	0.76	185
weighted avg	0.85	0.85	0.84	185

85.41% Accurate

- **Random Forest:**

```
              precision    recall  f1-score   support

           0       0.66      0.51      0.57        45
           1       0.85      0.91      0.88       140

    accuracy                           0.82       185
   macro avg       0.76      0.71      0.73       185
weighted avg       0.81      0.82      0.81       185

81.62% Accurate
```

Comparison with previous studies:

Study1: An Approach for Prediction of Loan Approval using Machine Learning Algorithm

Dataset description	dataset from Kaggle: The train dataset contains approximately 600+ rows and 13+ columns whereas the test dataset contains 300+ rows and 12+ columns, the test dataset does not contain the target variable.
ML method	Logistic Regression
Performance measure	Accuracy = 0.811

Study 2: Design and Simulation of Loan Approval Prediction Model using AWS Platform

Dataset description	dataset containing 4520 records and 17 properties.
ML method	decision tree logistic regression
Performance measure	Accuracy = 0.82

Study 3: Loan Default Prediction with Machine Learning Techniques

Dataset description	Xiamen International Bank
ML method	- XGBoost - Random Forest (RF) - AdaBoost - K nearest neighborhood (KNN) - Multilayer perceptrons (MLP)
Performance measure	**AUC:** - XGBoost = 0.7166 - RF = 0.501 - AdaBoost = 1 - KNN = 0.5036 - MLP = 0.5

Study 4: Predictions of Loan Defaulter - A Data Science Perspective

Dataset description	lending club loan dataset from Kaggle: The dataset was composed of 1.6 million records and 150 features

ML method	- Logistic Regression - RF - KNN
Performance measure	**Accuracy:** - Logistic Regression = 0.80 - RF = 0.79 - KNN = 0.78

Study 5: Swindle: Predicting the Probability of Loan Defaults using CatBoost Algorithm

Dataset description	standard Indian loan default dataset from Kaggle: containing 181398 records and 41 properties.
ML method	- CatBoost
Performance measure	Not mentioned

Study 6: Loan Prediction Using Ensemble Technique

Dataset description	data set include 13 attributes such as Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. The data sets contain 615 records.
ML method	Ensemble learning which combines: - SVM Model - Random Forest Network - Tree Model for Genetic Algorithm
Performance measure	Accuracy = 79.86

Best achieved Accuracy result from my models where 85.41 for **Logistic regression**

References:

[1] Decision Tree

[2] Machine Learning - Logistic Regression

[3] A Complete Guide to the Random Forest Algorithm

Studies:

1: An Approach for Prediction of Loan Approval using Machine Learning Algorithm
2: Design and Simulation of Loan Approval Prediction Model using AWS Platform
3: Loan Default Prediction with Machine Learning Techniques
4: Predictions of Loan Defaulter - A Data Science Perspective
5: Swindle: Predicting the Probability of Loan Defaults using CatBoost Algorithm
6: Loan Prediction Using Ensemble Technique

Appendix

```
              precision    recall  f1-score   support

           0       0.92      0.48      0.63        48
           1       0.81      0.98      0.89       106

    accuracy                           0.82       154
   macro avg       0.86      0.73      0.76       154
weighted avg       0.84      0.82      0.81       154
```

Study 2: Where 0 is non-default and 1 as default

Model	**Performance Metrics**		
	Accuracy	***Precision***	***Recall***
Logistic Regression	0.80	0.81	0.97
Random Forest	0.79	0.81	097
KNN	0.78	0.81	0.97

Study 4

Models	**Accuracy**	**H**	**Gini**	**AUC**	**AUCH**	**KS**	**MER**	**MWL**	**ROC**
Decision Tree	78.47	0.26	0.52	0.76	0.76	0.52	0.22	0.17	0.76
Linear Model	79.86	0.30	0.60	0.80	0.80	0.60	0.18	0.12	0.80
Neural Network	79.86	0.30	0.60	0.80	0.80	0.60	0.18	0.12	0.80
Random Forest	**80.56**	**0.32**	**0.60**	**0.80**	**0.80**	**0.60**	**0.19**	**0.13**	**0.80**
SVM	**80.56**	**0.32**	**0.60**	**0.80**	**0.80**	**0.60**	**0.19**	**0.13**	**0.82**
Bagged Cart	78.47	0.26	0.52	0.76	0.76	0.52	0.22	0.17	0.76
Tree model for genetic algorithm	**81.25**	**0.35**	**0.68**	**0.84**	**0.84**	**0.68**	**0.17**	**0.09**	**0.84**
model tree	79.86	0.30	0.59	0.79	0.79	0.59	0.19	0.13	0.79
Extreme learning machine	68.75	0.27	0.49	0.66	0.59	0.48	0.16	0.11	0.64
Multivariate Adaptive Regression Spline	79.86	0.30	0.60	0.80	0.80	0.60	0.18	0.12	0.80
BGLM	79.86	0.30	0.60	0.80	0.80	0.60	0.18	0.12	0.80
ENSEMBLED MODEL (SVM + RF + TMGA)	**79.86**	**0.31**	**0.63**	**0.78**	**0.78**	**0.63**	**0.20**	**0.14**	**0.79**

Study 6

Original Code

RNSS / RNSS Public

Code Pull requests Actions Projects Security Insights

main RNSS / loan_prediction_dataset.ipynb Go to file ...

RNSS Created using Colaborat... Latest commit c00e7ce 5 minutes ago History

1 contributor

1060 lines (1060 sloc) | 74.2 KB Raw Blame

Open in Colab
(https://colab.research.google.com/github/RNSS/RNSS/blob/main/loan_prediction_dataset.ipynb)

In [ ]:
```
# This Python 3 environment comes with many helpful analy
tics libraries installed
# It is defined by the kaggle/python docker image: https:
//github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)

# Input data files are available in the "../input/" direc
tory.
# For example, running this (by clicking run or pressing
Shift+Enter) will list the files in the input directory

import os
print(os.listdir("../input"))

# Any results you write to the current directory are save
d as output.
```

```
['test.csv', 'train.csv']
```

In [ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Loading and Summarizing Data

In [ ]:
```
train_data = pd.read_csv("../input/train.csv")
train_data.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	A
0	LP001002	Male	No	0	Graduate	No	5
1	LP001003	Male	Yes	1	Graduate	No	4
2	LP001005	Male	Yes	0	Graduate	Yes	3
3	LP001006	Male	Yes	0	Not Graduate	No	2
4	LP001008	Male	No	0	Graduate	No	6

In [ ]:
```
train_data.describe()
```

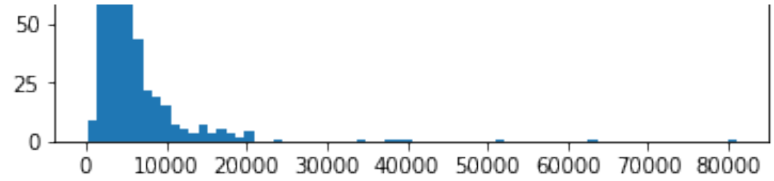
Out[ ]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
count	614.000000	614.000000	592.000000	600.00000
mean	5403.459283	1621.245798	146.412162	342.00000
std	6109.041673	2926.248369	85.587325	65.12041
min	150.000000	0.000000	9.000000	12.00000
25%	2877.500000	0.000000	100.000000	360.00000
50%	3812.500000	1188.500000	128.000000	360.00000
75%	5795.000000	2297.250000	168.000000	360.00000
max	81000.000000	41667.000000	700.000000	480.00000

Distribution Analysis

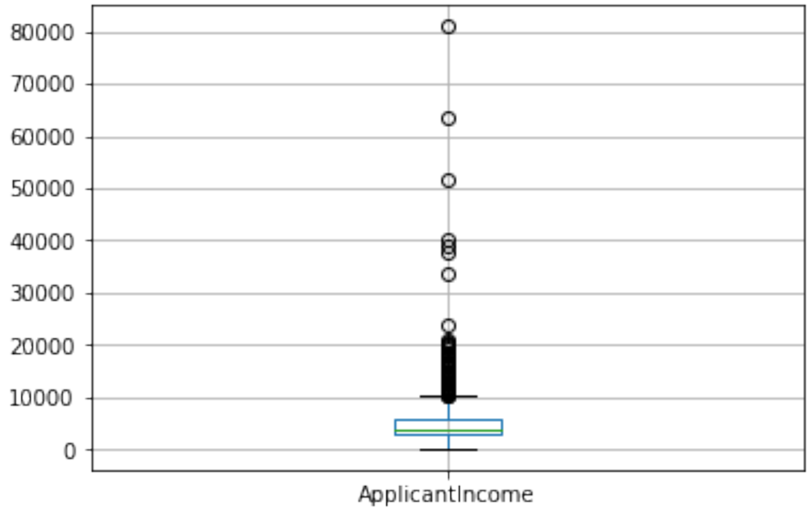
In [ ]:
```
train_data['ApplicantIncome'].hist(bins=70,grid=False)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe3f1129048>



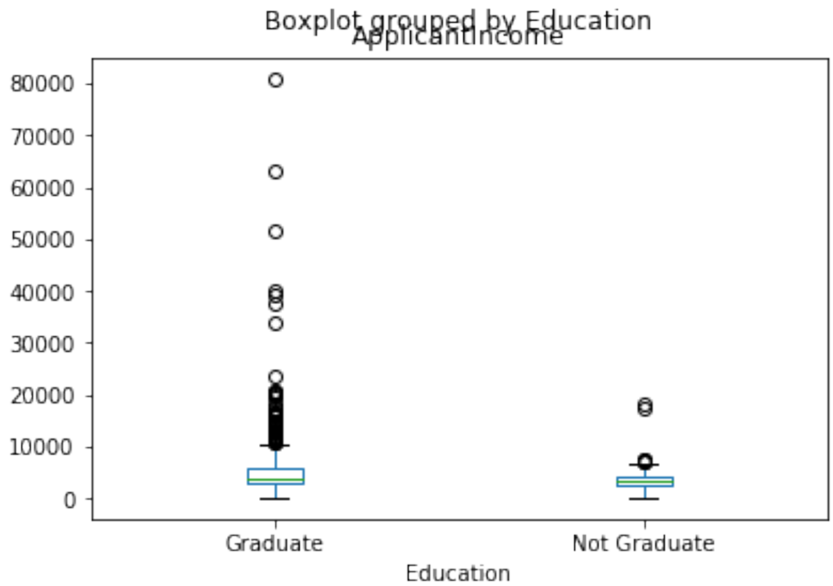
In [ ]:
```
train_data.boxplot(column = 'ApplicantIncome')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe3f105c518>



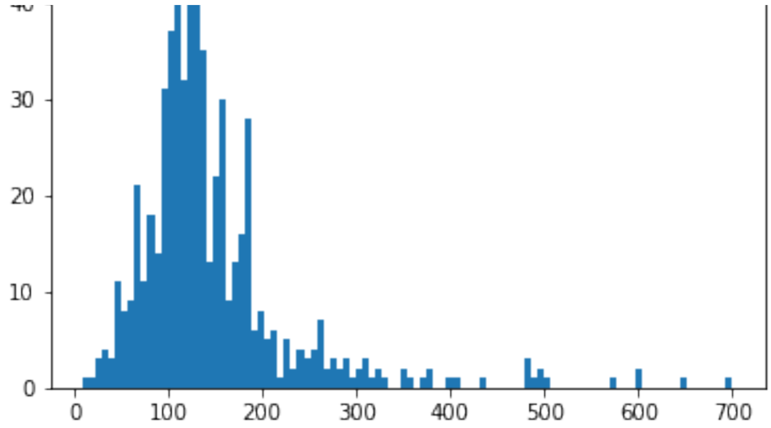
In [ ]:
```
train_data.boxplot(column = 'ApplicantIncome', grid =False, by = 'Education')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe3ed75f390>



In [ ]:
```
train_data['LoanAmount'].hist(bins=100,grid = False)
```

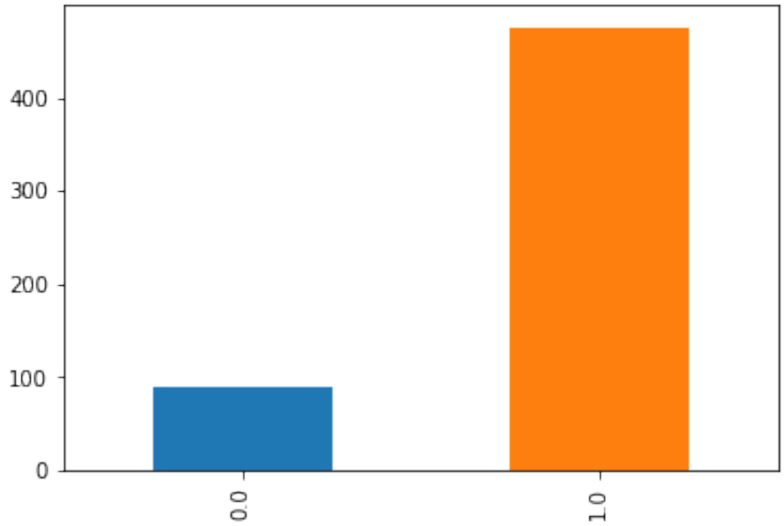
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe3ed6e67f0>



Categorical Value Analysis

In [ ]:
```
temp = train_data['Credit_History'].value_counts(ascendin
g = True)
temp.plot(kind = 'bar')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe3ed5e8da0>



Data Munging

In [ ]:
```
train_data.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[ ]:
```
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
```

In [ ]:
```
train_data['LoanAmount'].fillna(train_data['LoanAmount'].
mean(),inplace=True)
```

In [ ]:
```
train_data['Self_Employed'].fillna('No',inplace=True)
```

In [ ]:
```
train_data['Gender'].fillna(train_data['Gender'].mode()[0
], inplace=True)
train_data['Married'].fillna(train_data['Married'].mode()
[0], inplace=True)
train_data['Dependents'].fillna(train_data['Dependents'].
mode()[0], inplace=True)
train_data['Loan_Amount_Term'].fillna(train_data['Loan_Am
ount_Term'].mode()[0], inplace=True)
train_data['Credit_History'].fillna(train_data['Credit_Hi
story'].mode()[0], inplace=True)
```

In [ ]:
```
train_data.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	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

In [ ]:
```
train_data.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[ ]:
```
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 [ ]:
```
from sklearn.preprocessing import LabelEncoder
```

```
var_mod = ['Gender','Married','Dependents','Education','S
elf_Employed','Property_Area','Loan_Status']
le = LabelEncoder()
for i in var_mod:
    train_data[i] = le.fit_transform(train_data[i])
train_data.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed
0	LP001002	1	0	0	0	0
1	LP001003	1	1	1	0	0
2	LP001005	1	1	0	0	1
3	LP001006	1	1	0	1	0
4	LP001008	1	0	0	0	0

Training Model

modified Code

RNSS / RNSS Public

Code Pull requests Actions Projects Security Insights

main RNSS / IT351_Project.ipynb Go to file ···

RNSS Created using Colabora... Latest commit 8ce4e5e 24 minutes ago History

1 contributor

2488 lines (2488 sloc) | 204 KB Raw Blame

Open in Colab
(https://colab.research.google.com/github/RNSS/RNSS/blob/main/IT351_Project.ipynb)

1. Import Libraries

In [ ]:
```
#visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
#default theme
from matplotlib import cycler
colors = cycler('color',
                ['#EE6666', '#3388BB', '#9988DD',
                 '#EECC55', '#88BB44', '#FFBBBB'])
plt.rc('axes', facecolor='#E6E6E6', edgecolor='none',
       axisbelow=True, grid=True, prop_cycle=colors)
plt.rc('grid', color='w', linestyle='solid')
plt.rc('xtick', direction='out', color='gray')
plt.rc('ytick', direction='out', color='gray')
plt.rc('patch', edgecolor='#E6E6E6')
plt.rc('lines', linewidth=2)
```

In [ ]:
```
#Label encoding
from sklearn.preprocessing import LabelEncoder
#spliting data into train and test
from sklearn.model_selection import train_test_split
#Modeling
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
#Evaluation
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

In [ ]:

```
#data wrangling
import numpy as np
import pandas as pd
```

2. Data Acquisition

In [ ]:

```
#Upload from Colab
from google.colab import files
uploaded = files.upload()
```

```
Saving train.csv to train.csv
Saving test.csv to test.csv
```

Train Data set

In [ ]:

```
#train_data = pd.read_csv("/Users/rynadalswyd/Documents/LEARN/Level 9/IT351/project/train.csv")
train_data = pd.read_csv("train.csv")
train_data.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	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

Test Data set

In [ ]:

```
#test_data = pd.read_csv("/Users/rynadalswyd/Documents/LEARN/Level 9/IT351/project/test.csv")
test_data = pd.read_csv("test.csv")
test_data.head()
```

	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

3. Exploratory Analysis

Train Data set

In [ ]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [ ]:
```
train_data.describe()
```

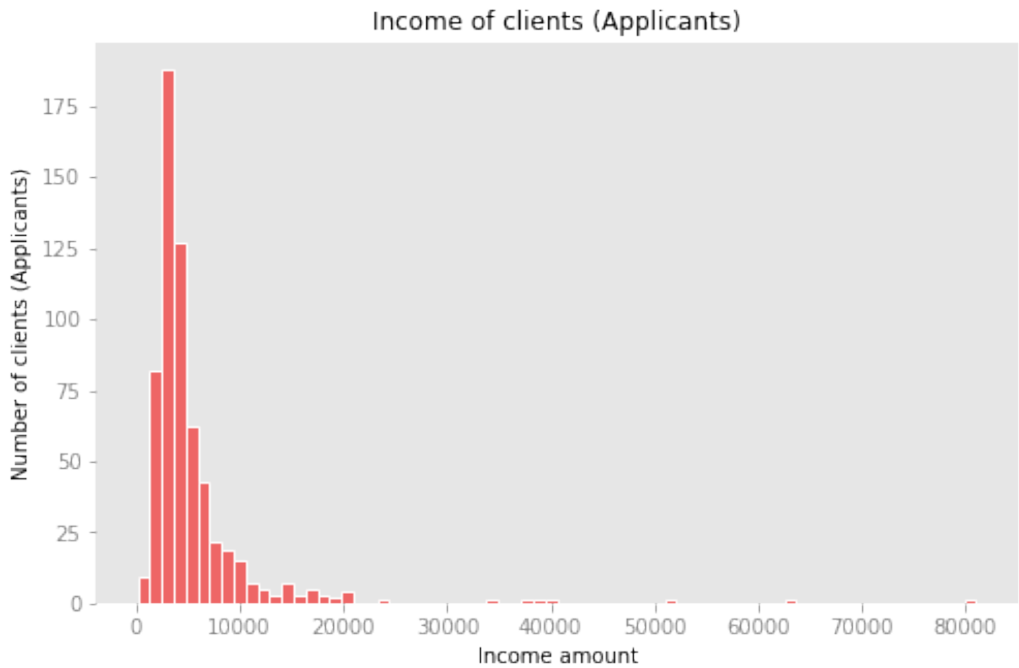
Out[ ]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
count	614.000000	614.000000	592.000000	600.00000
mean	5403.459283	1621.245798	146.412162	342.00000
std	6109.041673	2926.248369	85.587325	65.12041
min	150.000000	0.000000	9.000000	12.00000

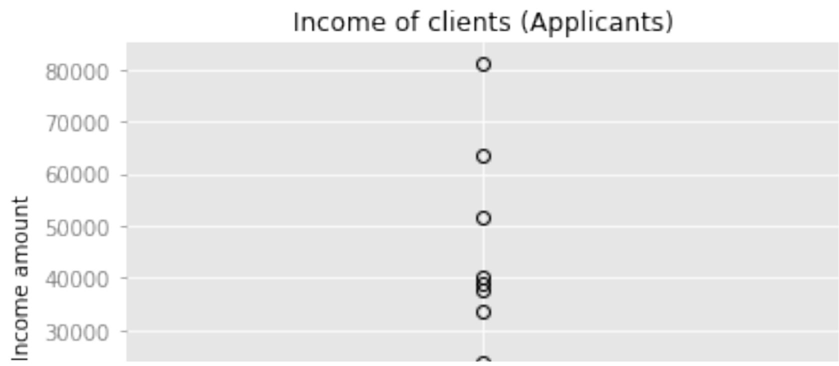
25%	2877.500000	0.000000	100.000000	360.00000
50%	3812.500000	1188.500000	128.000000	360.00000
75%	5795.000000	2297.250000	168.000000	360.00000
max	81000.000000	41667.000000	700.000000	480.00000

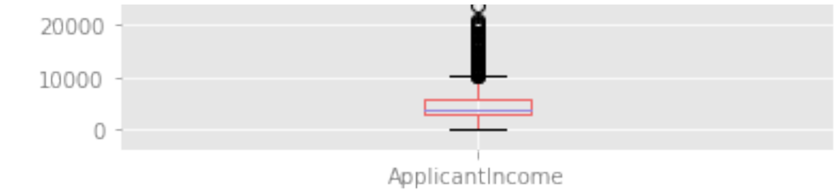
Graphical Techniques

In [ ]:
```
plt.subplots(figsize=(8,5))
train_data['ApplicantIncome'].hist(bins=70,grid=False,edgecolor='white')
plt.xlabel('Income amount')
plt.ylabel('Number of clients (Applicants)')
plt.title('Income of clients (Applicants)')
plt.show()
```



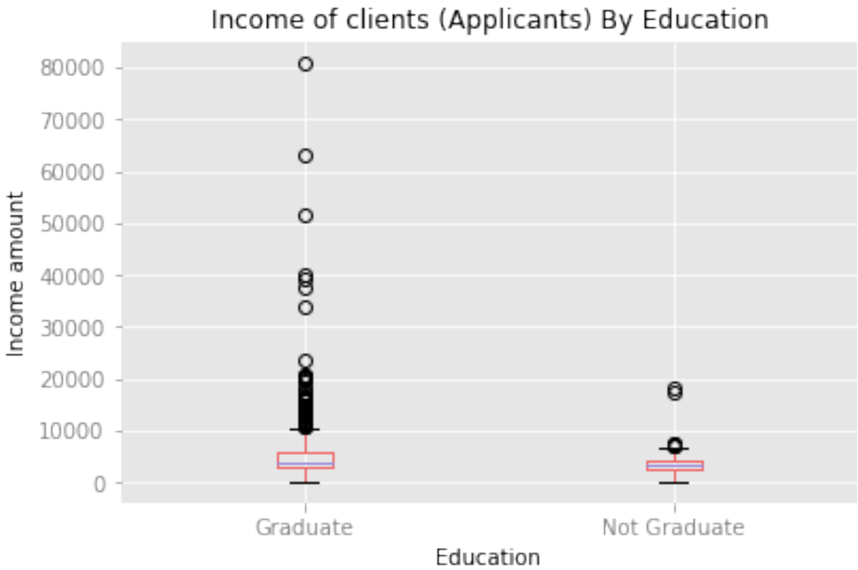
In [ ]:
```
train_data.boxplot(column = 'ApplicantIncome')
plt.ylabel('Income amount')
plt.title('Income of clients (Applicants)')
plt.show()
```





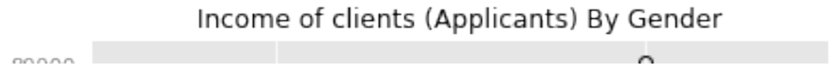
In [ ]:
```
train_data.boxplot(column = 'ApplicantIncome',  by = 'Edu
cation')
plt.ylabel('Income amount')
plt.title('Income of clients (Applicants) By Education')
plt.suptitle('')
plt.show()
```

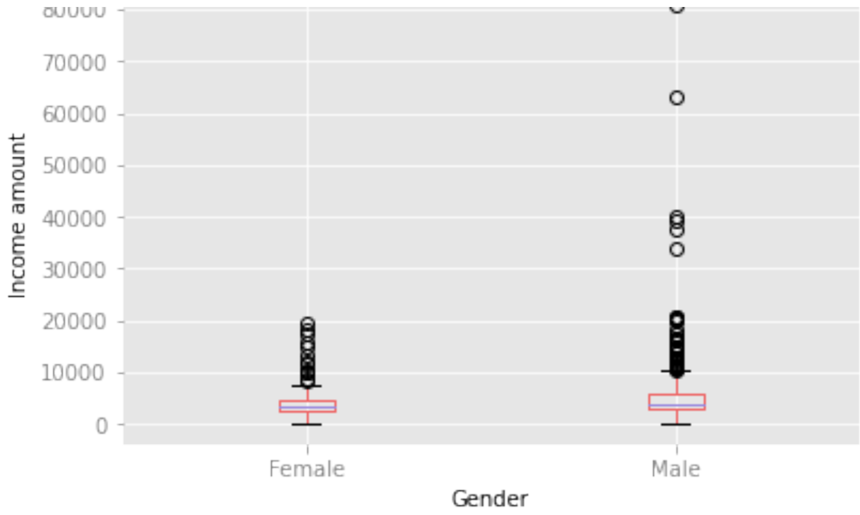
```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarra
y.py:83: VisibleDeprecationWarning: Creating an ndarray f
rom ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or sha
pes) is deprecated. If you meant to do this, you must spe
cify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```



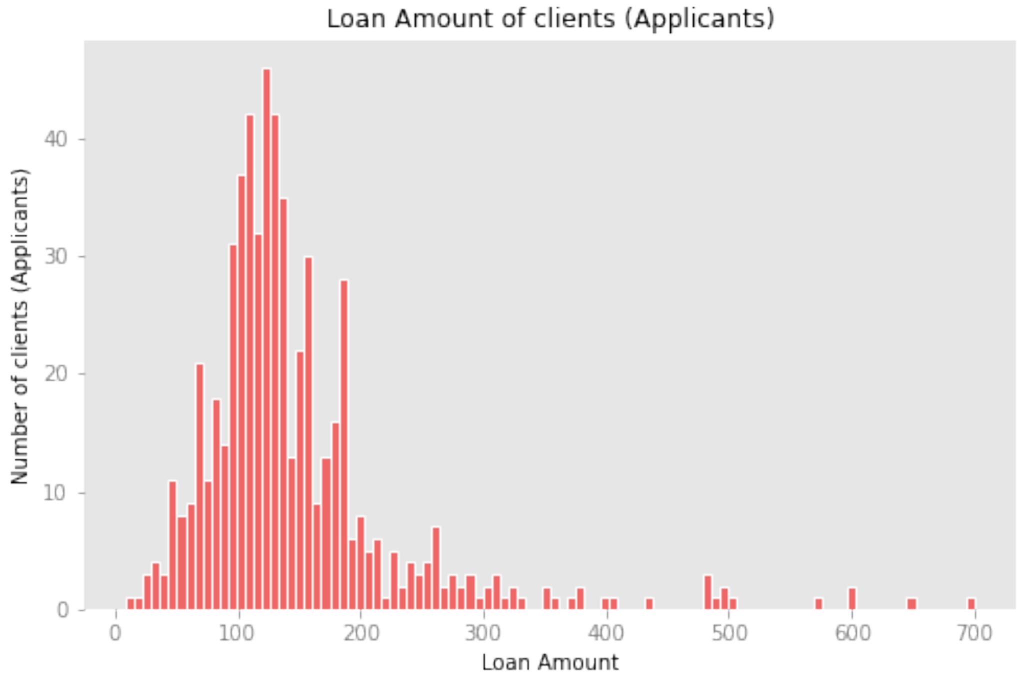
In [ ]:
```
train_data.boxplot(column = 'ApplicantIncome', by = 'Gend
er')
plt.ylabel('Income amount')
plt.title('Income of clients (Applicants) By Gender')
plt.suptitle('')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarra
y.py:83: VisibleDeprecationWarning: Creating an ndarray f
rom ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or sha
pes) is deprecated. If you meant to do this, you must spe
cify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```

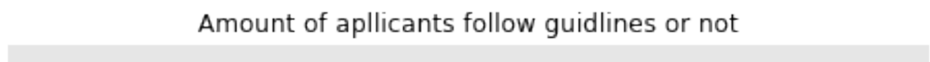


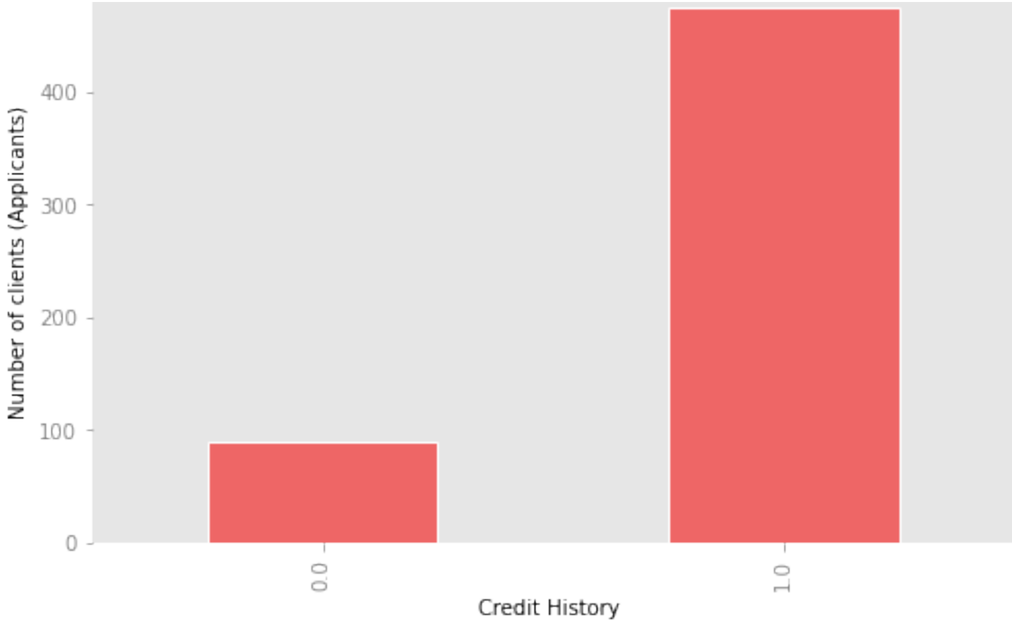


In [ ]:
```
plt.subplots(figsize=(8,5))
train_data['LoanAmount'].hist(bins=100,grid = False,edgecolor='white')
plt.xlabel('Loan Amount')
plt.ylabel('Number of clients (Applicants)')
plt.title('Loan Amount of clients (Applicants)')
plt.show()
```



In [ ]:
```
plt.subplots(figsize=(8,5))
temp = train_data['Credit_History'].value_counts(ascending = True)
temp.plot(kind = 'bar',grid = False,edgecolor='white')
plt.xlabel('Credit History ')
plt.ylabel('Number of clients (Applicants)')
plt.title('Amount of apllicants follow guidlines or not')
plt.show()
```





Test Data set

In [ ]:
```
test_data.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 [ ]:
```
test_data.describe()
```

Out[ ]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
count	367.000000	367.000000	362.000000	361.000000
mean	4805.599455	1569.577657	136.132597	342.537396
std	4910.685399	2334.232099	61.366652	65.156643
min	0.000000	0.000000	28.000000	6.000000

25%	2864.000000	0.000000	100.250000	360.000000
50%	3786.000000	1025.000000	125.000000	360.000000
75%	5060.000000	2430.500000	158.000000	360.000000
max	72529.000000	24000.000000	550.000000	480.000000

4. Process/Clean Data

Train Data set

1: Handle missing values

In [ ]:
```
train_data.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[ ]:
```
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
```

In [ ]:
```
train_data['LoanAmount'].fillna(train_data['LoanAmount'].
mean(),inplace=True)
```

In [ ]:
```
train_data['Self_Employed'].fillna('No',inplace=True)
```

In [ ]:
```
train_data['Gender'].fillna(train_data['Gender'].mode()[0
], inplace=True)
train_data['Married'].fillna(train_data['Married'].mode()
[0], inplace=True)
train_data['Dependents'].fillna(train_data['Dependents'].
mode()[0], inplace=True)
train_data['Loan_Amount_Term'].fillna(train_data['Loan_Am
ount_Term'].mode()[0], inplace=True)
train_data['Credit_History'].fillna(train_data['Credit_Hi
story'].mode()[0], inplace=True)
```

In [ ]:
```
train_data.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[ ]:
```
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
```

2: Lable Encoding

In [ ]:
```
var_mod = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area','Loan_Status']
le = LabelEncoder()
for i in var_mod:
    train_data[i] = le.fit_transform(train_data[i])
    print(i,"  :  ", le.classes_)
train_data.head(20)
```

```
Gender   :   ['Female' 'Male']
Married   :   ['No' 'Yes']
Dependents   :   ['0' '1' '2' '3+']
Education   :   ['Graduate' 'Not Graduate']
Self_Employed   :   ['No' 'Yes']
Property_Area   :   ['Rural' 'Semiurban' 'Urban']
Loan_Status   :   ['N' 'Y']
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed
0	LP001002	1	0	0	0	0
1	LP001003	1	1	1	0	0
2	LP001005	1	1	0	0	1
3	LP001006	1	1	0	1	0
4	LP001008	1	0	0	0	0
5	LP001011	1	1	2	0	1
6	LP001013	1	1	0	1	0
7	LP001014	1	1	3	0	0
8	LP001018	1	1	2	0	0
9	LP001020	1	1	1	0	0

10	LP001024	1	1	2	0	0
11	LP001027	1	1	2	0	0
12	LP001028	1	1	2	0	0
13	LP001029	1	0	0	0	0
14	LP001030	1	1	2	0	0
15	LP001032	1	0	0	0	0
16	LP001034	1	0	1	1	0
17	LP001036	0	0	0	0	0
18	LP001038	1	1	0	1	0
19	LP001041	1	1	0	0	0

In [ ]:
```
print(le.classes_)
```

```
['N' 'Y']
```

3: Drop unaffected label

In [ ]:
```
train_data.drop('Loan_ID',axis=1,inplace=True)
```

Test Data set

1: Handle missing values

In [ ]:
```
test_data.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[ ]:
```
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
```

In [ ]:
```
test_data['LoanAmount'].fillna(train_data['LoanAmount'].m
ean(),inplace=True)
```

In [ ]:
```
test_data['Self_Employed'].fillna('No',inplace=True)
```

In [ ]:
```
test_data['Gender'].fillna(test_data['Gender'].mode()[0], inplace=True)
test_data['Married'].fillna(test_data['Married'].mode()[0], inplace=True)
test_data['Dependents'].fillna(test_data['Dependents'].mode()[0], inplace=True)
test_data['Loan_Amount_Term'].fillna(test_data['Loan_Amount_Term'].mode()[0], inplace=True)
test_data['Credit_History'].fillna(test_data['Credit_History'].mode()[0], inplace=True)
```

In [ ]:
```
test_data.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[ ]:
```
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
```

2: Lable Encoding

In [ ]:
```
var_mod_test = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area']
for i in var_mod_test:
    test_data[i] = le.fit_transform(test_data[i])
    print(i,"  :  ", le.classes_)
test_data.head()
```

```
Gender   :   ['Female' 'Male']
Married   :   ['No' 'Yes']
Dependents   :   ['0' '1' '2' '3+']
Education   :   ['Graduate' 'Not Graduate']
Self_Employed   :   ['No' 'Yes']
Property_Area   :   ['Rural' 'Semiurban' 'Urban']
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed
0	LP001015	1	1	0	0	0
1	LP001022	1	1	1	0	0
2	LP001031	1	1	2	0	0

3	LP001035	1	1	2	0	0	2
4	LP001051	1	0	0	1	0	3

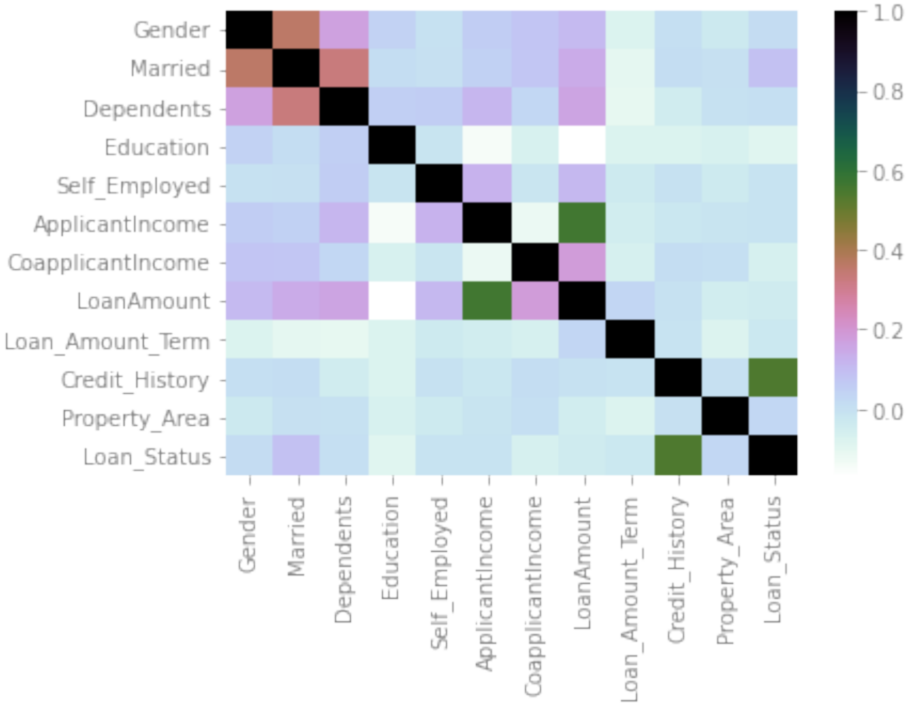
3: Drop unaffected label

In [ ]:
```
test_data.drop('Loan_ID',axis=1,inplace=True)
```

Corrlation Matrix

In [ ]:
```
sns.heatmap(train_data.corr() ,cmap='cubehelix_r')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa7931b4310>



5. Model Generation & Evaluation

In [ ]:
```
Y = train_data['Loan_Status']
X = train_data.drop('Loan_Status', axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size = 0.3, random_state = 3)
print("X_train = ",len(X_train),"\nX_test=",len(X_test))
```

```
X_train =  429
X_test= 185
```

Decision Tree

In [ ]:
```
DT = DecisionTreeClassifier()
DT.fit(X_train,y_train)
predict_DT = DT.predict(X_test)
```

In [ ]:
```
DT_SC = accuracy_score(predict_DT,y_test)
print(classification_report(y_test, predict_DT))
print(f"{round(DT_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.40      0.60      0.48        45
           1       0.85      0.71      0.78       140

    accuracy                           0.69       185
   macro avg       0.63      0.66      0.63       185
weighted avg       0.74      0.69      0.70       185

68.65% Accurate
```

Logistic Regression

In [ ]:
```
LR = LogisticRegression()
LR.fit(X_train, y_train)
predict_LR = LR.predict(X_test)
```

In [ ]:
```
LR_SC = accuracy_score(predict_LR,y_test)
print(classification_report(y_test, predict_LR))
print(f"{round(LR_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.85      0.49      0.62        45
           1       0.86      0.97      0.91       140

    accuracy                           0.85       185
   macro avg       0.85      0.73      0.76       185
weighted avg       0.85      0.85      0.84       185

85.41% Accurate
```

Random Forest