Loan Prediction

The aim of this project is to predict real-estate prices using the machine learning algorithms: Logistic Regression, Decision tree, Random Forest.

This file is the EDA and its purpose is to go through the sevarl steps of working with data - data gathering, data understanding, data preparation. Visualization of the information is made for better understanding.

Data storage solution:

An essential part of Machine Learning is the data storage solution for the selected data and machine learning model. In order to accomplish the most efficient manner of working with data during this project, the following tools were used:

- Git Version Control
- Data Version Control (DVC)

Git Version Control: Git has been a popular tool among programmers and it is so for a reason. It allows tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

Data Version Control (DVC): Data Version Control is a new type of data versioning, workflow, and experiment management software that builds upon Git (although it can work stand-alone). Using Git and DVC, machine learning teams can version experiments, manage large datasets, and make projects reproducible. By utilizing DVC data will be tracked and stored in an effective and efficient way because the data is accessible from everywhere via internet connection for every contributor.

Summary:

- DVC will create reference files to data versions
- Git will store the DVC files

At the current stage, the dataset has been loaded after integrating it. Changes were made to like cleaning and processing it so as to make it more suitable to work with and acceptable to store.

Loading the data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from google.colab import files
from datetime import datetime
import seaborn as sns
%matplotlib inline

import io
```

Using two files which later will be integrated into one dataframe.

```
# Load the data
local_file = files.upload()
train_data = io.BytesIO(local_file['train1.csv'])
train_data2 = io.BytesIO(local_file['train.csv'])
df1 = pd.read_csv(train_data)
df2 = pd.read_csv(train_data2)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving train.csv to train (2).csv
Saving train1.csv to train1 (2).csv
```

Data integration

```
df1.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
```

> It can be seen that there are some numeric values and also categorical ones. Later, the categorical will have to be converted into either float or int in order to be plotted and then used for the trainig of the models. There are also missing values in the dataset.

```
df2.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
```

> It can be seen that there are some numeric values and also categorical ones. Later, the categorical will have to be converted into either float or int in order to be plotted and then used for the trainig of the models. There are also missing values in the dataset.

Creating one dataframe out of the two dataframes from the csv files.

```
frames = [df1, df2]
```

```
df = pd.concat(frames)
df
```

	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
...	...	...	...	...	...	...	...
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

Data analysis:

Checking the dimension of the dataset and the features.

Take a look at the summary of the numerical fields.

```
df.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	981.000000	981.000000	954.000000	961.000000	902.000000
mean	5179.795107	1601.916330	142.511530	342.201873	0.835920
std	5695.104533	2718.772806	77.421743	65.100602	0.370553
min	0.000000	0.000000	9.000000	6.000000	0.000000
25%	2875.000000	0.000000	100.000000	360.000000	1.000000
50%	3800.000000	1110.000000	126.000000	360.000000	1.000000
75%	5516.000000	2365.000000	162.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

```
df.shape
```

```
(981, 13)
```

> Here it can be seen the shape of our data with the `.shape`. Here are the the rows and columns(981, 13).

To view what data that is stored the `columns` should be explored. This will return the colums of the data.

```
df.columns
```

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

After reviewing all the columns, in the next cell is outlined only the quantitive columns.

```
quantitative = [f for f in df.columns if df.dtypes[f] != 'object']
print(quantitative)
```

```
['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_Histo
```

To look at the data I'll use the .head() method from pandas. This will show the first 5 items in the dataframe.

```
#First 5 rows of our dataset
df.head()
```

	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	

To look at the data I'll use the .tail() method from pandas. This will show us the last 5 items in the

```
#Last 5 rows of our dataset
df.tail()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 981 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            981 non-null    object 
 1   Gender             957 non-null    object 
 2   Married            978 non-null    object 
 3   Dependents         956 non-null    object 
 4   Education          981 non-null    object 
 5   Self_Employed      926 non-null    object 
 6   ApplicantIncome    981 non-null    int64  
 7   CoapplicantIncome  981 non-null    float64
 8   LoanAmount         954 non-null    float64
 9   Loan_Amount_Term   961 non-null    float64
 10  Credit_History     902 non-null    float64
 11  Property_Area      981 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 107.3+ KB
```

> It can be seen that there are features that are numeric and also objects. Later, the ones that are not numeric will have to be converted into either float or int in order to be plotted and then used for the trainig of the models. There are also missing values in the dataset, which will be handled later.

Sorting the data by `Loan_Status` and showing 50 elements.

```
df.sort_values('Loan_Status', ascending = True)[:50]
```

226	LP001754	Male	Yes	NaN	Not Graduate	Yes	4735
459	LP002473	Male	Yes	0	Graduate	No	8334
457	LP002467	Male	Yes	0	Graduate	No	3708
236	LP001786	Male	Yes	0	Graduate	NaN	5746
452	LP002448	Male	Yes	0	Graduate	No	3948
183	LP001637	Male	Yes	1	Graduate	No	33846
450	LP002446	Male	Yes	2	Not Graduate	No	2309
181	LP001634	Male	No	0	Graduate	No	1916
179	LP001630	Male	No	0	Not Graduate	No	2333
524	LP002697	Male	No	0	Graduate	No	4680
135	LP001488	Male	Yes	3+	Graduate	No	4000
136	LP001489	Female	Yes	0	Graduate	No	4583
519	LP002684	Female	No	0	Not Graduate	No	3400
138	LP001492	Male	No	0	Graduate	No	14999
139	LP001493	Male	Yes	2	Not Graduate	No	4200
140	LP001497	Male	Yes	2	Graduate	No	5042
518	LP002683	Male	No	0	Graduate	No	4683
517	LP002682	Male	Yes	NaN	Not Graduate	No	3074
514	LP002652	Male	No	0	Graduate	No	5815
513	LP002648	Male	Yes	0	Graduate	No	2130
148	LP001519	Female	No	0	Graduate	No	10000
150	LP001528	Male	No	0	Graduate	No	6277
510	LP002637	Male	No	0	Not Graduate	No	3598
152	LP001531	Male	No	0	Graduate	No	9166
153	LP001532	Male	Yes	2	Not Graduate	No	2281
507	LP002625	NaN	No	0	Graduate	No	3583
503	LP002618	Male	Yes	1	Not Graduate	No	4050

Sorting the data by `Education` and showing 50 elements.

```
df.sort_values('Education')[:50]
```

258	LP001859	Male	Yes	0	Graduate	No	14683
260	LP001865	Male	Yes	1	Graduate	No	6083
261	LP001868	Male	No	0	Graduate	No	2060
262	LP001870	Female	No	1	Graduate	No	3481
263	LP001871	Female	No	0	Graduate	No	7200
248	LP001824	Male	Yes	1	Graduate	No	2882
231	LP001768	Male	Yes	0	Graduate	NaN	3716
230	LP001765	Male	Yes	1	Graduate	No	2491
229	LP001761	Male	No	0	Graduate	Yes	6400
196	LP001666	Male	No	0	Graduate	No	8333
198	LP001671	Female	Yes	0	Graduate	No	3416
199	LP001673	Male	No	0	Graduate	Yes	11000
201	LP001677	Male	No	2	Graduate	No	4923
206	LP001693	Female	No	0	Graduate	No	3244
208	LP001699	Male	No	0	Graduate	No	2479
209	LP001702	Male	No	0	Graduate	No	3418
210	LP001708	Female	No	0	Graduate	No	10000
211	LP001711	Male	Yes	3+	Graduate	No	3430
212	LP001713	Male	Yes	1	Graduate	Yes	7787
214	LP001716	Male	Yes	0	Graduate	No	3173
216	LP001722	Male	Yes	0	Graduate	No	150
217	LP001726	Male	Yes	0	Graduate	No	3727
218	LP001732	Male	Yes	2	Graduate	NaN	5000
219	LP001734	Female	Yes	2	Graduate	No	4283
220	LP001736	Male	Yes	0	Graduate	No	2221
221	LP001743	Male	Yes	2	Graduate	No	4009
222	LP001744	Male	No	0	Graduate	No	2971
223	LP001749	Male	Yes	0	Graduate	No	7578
224	LP001750	Male	Yes	0	Graduate	No	6250
225	LP001751	Male	Yes	0	Graduate	No	3250
227	LP001758	Male	Yes	2	Graduate	No	6250
228	LP001760	Male	NaN	NaN	Graduate	No	4758

This is a representation of one row content, which helps by showing what to look for and what to expect to be in each other row.

```
df.iloc[0]
```

```
Loan_ID              LP001015
Gender                   Male
Married                   Yes
Dependents                  0
Education            Graduate
Self_Employed              No
ApplicantIncome          5720
CoapplicantIncome           0
LoanAmount                110
Loan_Amount_Term          360
Credit_History              1
Property_Area           Urban
Loan_Status               NaN
Name: 0, dtype: object
```

Get the unique values and their frequency of variable. (Checking how many times the certain value occurs.)

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

```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

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

```
2500     13
5000     11
3333     10
3500      9
2600      8
         ..
5391      1
15000     1
14999     1
7830      1
1811      1
Name: ApplicantIncome, Length: 752, dtype: int64
```

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

```
Male      775
Female    182
Name: Gender, dtype: int64
```

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

```
Yes    631
No     347
Name: Married, dtype: int64
```

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

```
0.0       429
2500.0      6
1666.0      5
2000.0      5
2083.0      5
         ...
6250.0      1
1742.0      1
189.0       1
1868.0      1
4266.0      1
Name: CoapplicantIncome, Length: 437, dtype: int64
```

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

```
0     545
2     160
1     160
3+     91
Name: Dependents, dtype: int64
```

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

```
Graduate        763
Not Graduate    218
Name: Education, dtype: int64
```

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

```
No     807
Yes    119
Name: Self_Employed, dtype: int64
```

```
df['Loan_Status'].unique()
```

```
array([nan, 'Y', 'N'], dtype=object)
```

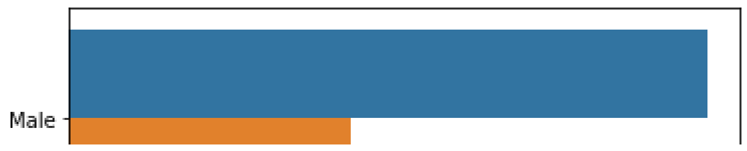
```
df['ApplicantIncome'].unique()
```

```
             3863,  4028,  4010,  3719,  2858,  3833,  3007,  1850,  2792,
             2982, 18840,  2995,  3579,  3835,  3854,  3508,  1635, 24797,
             2773,  5769,  3634, 29167,  5530,  9000,  8750,  1972,  4983,
             8333,  3667,  3166,  3271,  2241,  1792,  2666,  6478,  3808,
             3729,  4120,  6300, 14987,   570,  2600,  2733,  3859,  6825,
             3708,  5314,  2366,  2066,  3767,  7859,  4283,  1700,  4768,
             3083,  2667,  1647,  3400, 16000,  2875,  5041,  6958,  5509,
             9699,  3621,  4709,  3015,  2292,  2360,  2623,  3972,  3522,
             6858,  8334,  2868,  3418,  8667,  2283,  5817,  5119,  5316,
             7603,  3791,  3132,  8550,  2269,  4009,  4158,  9200,  5849,
             3000,  2583,  6000,  5417,  3036,  4006, 12841,  3200,  1853,
             1299,  4950,  3596,  4887,  7660,  5955,  3365,  3717,  9560,
             2799,  4226,  1442,  3750,  3167,  4692,  2275,  1828,  3748,
             3600,  1800,  3941,  4695,  3410,  5649,  5821,  2645,  1928,
             3086,  4230,  4616, 11500,  2132,  3366,  8080,  3357,  3029,
             2609,  4945,  5726, 10750,  7100,  4300,  3208,  1875,  4755,
             5266,  1000,  3846,  2395,  1378,  3988,  8566,  5695,  2958,
             3273,  4133,  6782,  2484,  1977,  4188,  1759,  4288,  4843,
            13650,  4652,  3816,  3052, 11417,  7333,  3800,  2071,  2929,
             3572,  7451,  5050, 14583,  2214,  5568, 10408,  2137,  2957,
             3692, 23803,  3865, 10513,  6080, 20166,  2718,  3459,  4895,
             3316, 14999,  4200,  5042,  6950,  2698, 11757,  2330, 14866,
             1538,  4860,  6277,  2577,  9166,  2281,  3254, 39999,  9538,
             2980,  1863,  7933,  9323,  3707,  2439,  2237,  1820, 51763,
             4344,  3497,  2045,  5516,  6400,  1916,  4600, 33846,  3625,
            39147,  2178,  2383,   674,  9328,  4885, 12000,  6033,  3858,
             4191,  1907,  3416, 11000,  4923,  3992,  3917,  4408,  3244,
             3975,  2479,  3430,  7787,  5703,  3173,  3850,   150,  3727,
             2221,  2971,  7578,  4735,  4758,  2491,  3716,  3155,  5500,
             5746,  3463,  3812,  3315,  5819,  2510,  2965,  3406,  6050,
             9703,  6608,  2882,  1809,  1668,  3427,  2661, 16250,  6045,
             5250, 14683,  4931,  6083,  2060,  3481,  7200,  5166,  4095,
             4708,  2876,  3237, 11146,  2833,  2620,  3993,  3103,  4100,
             4053,  3927,  2301,  1811, 20667,  3158,  3704,  4124,  9508,
             3075,  4400,  3153,  4416,  6875,  1625,  3762, 20233,  7667,
             2927,  2507,  2473,  3399,  2058,  3541,  4342,  3601, 15000,
             8666,  4917,  5818,  4384,  2935, 63337,  9833,  5503,  1830,
             4160,  2647,  2378,  4554,  2499,  3523,  6333,  2625,  9083,
             2423,  3813,  3875,  5167,  4723,  3013,  6822,  6216,  5124,
             6325, 19730, 15759,  5185,  3062,  2764,  4817,  4310,  3069,
             5391,  5941,  7167,  4566,  2346,  3010,  5488,  9504,  1993,
             3100,  3033,  3902,  1500,  2889,  2755,  1963,  7441,  4547,
             2167,  2213,  8300, 81000,  3867,  6256,  6096,  2253,  2149,
             1600,  1025,  3246,  5829,  2720,  7250, 14880,  4606,  5935,
             2920,  2717,  8624, 12876,  2425, 10047,  1926, 10416,  7142,
             3660,  7901,  4707, 37719,  3466,  3539,  3340,  2769,  2309,
             1958,  3948,  2483,  7085,  4301,  4354,  7740,  5191,  2947,
            16692,   210,  3450,  2653,  4691,  5532, 16525,  6700,  2873,
            16667,  4350,  3095, 10833,  3547, 18333,  2435,  2699,  3691,
            17263,  3597,  3326,  4625,  2895,  6283,   645,  4865,  4050,
             3814, 20833, 13262,  3598,  6065,  3283,  2130,  5815,  2031,
             4683,  2192,  5677,  7948,  4680, 17500,  3775,  5285,  2679,
             6783,  4281,  3588, 11250, 18165,  2550,  6133,  3617,  6417,
             4608,  2138,  3652,  2239,  2768,  3358,  2526,  2785,  6633,
             2492,  2454,  3593,  5468, 10139,  4180,  3675, 19484,  5923,
             5800,  8799,  4467,  5116, 16666,  6125,  6406,  3087,  3229,
             1782,  3182,  6540,  1836,  1880,  2787,  2297,  2726,  9357,

            16120,  6383,  2987,  9963,  5780,   416,  2894,  3676,  3987,
             3232,  2900,  4106,  8072,  7583])
```

Plotting the difference between men and women who applied for a loan and then the difference between the approvals and denials.

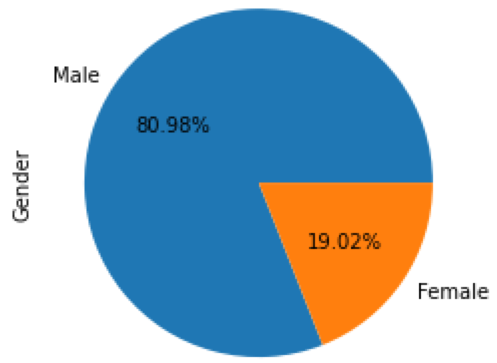
```
sns.countplot(y = 'Gender', hue = 'Loan_Status', data = df)
plt.figure(figsize=(16, 6))
plt.show()
```



> The diagram shows on one hand that there are more male applicants than female and on other hand, there are more approved loans than disapproved.

Plotting the difference between men and women applied for a loan in percentage.

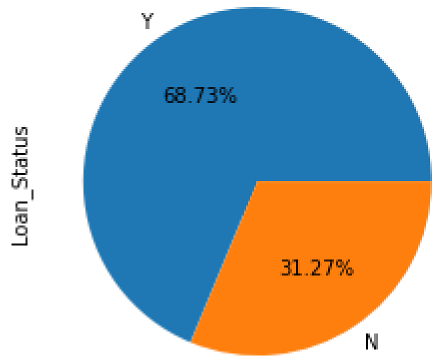
```
df['Gender'].value_counts().plot(kind='pie', autopct='%1.2f%%', figsize=(4, 4))
plt.show()
```



> The percentage of males who applied for a loan is greater than the one of females.

Plotting the difference between men and women applied for a loan in percentage.

```
df['Loan_Status'].value_counts().plot(kind='pie', autopct='%1.2f%%', figsize=(4,4))
plt.show()
```

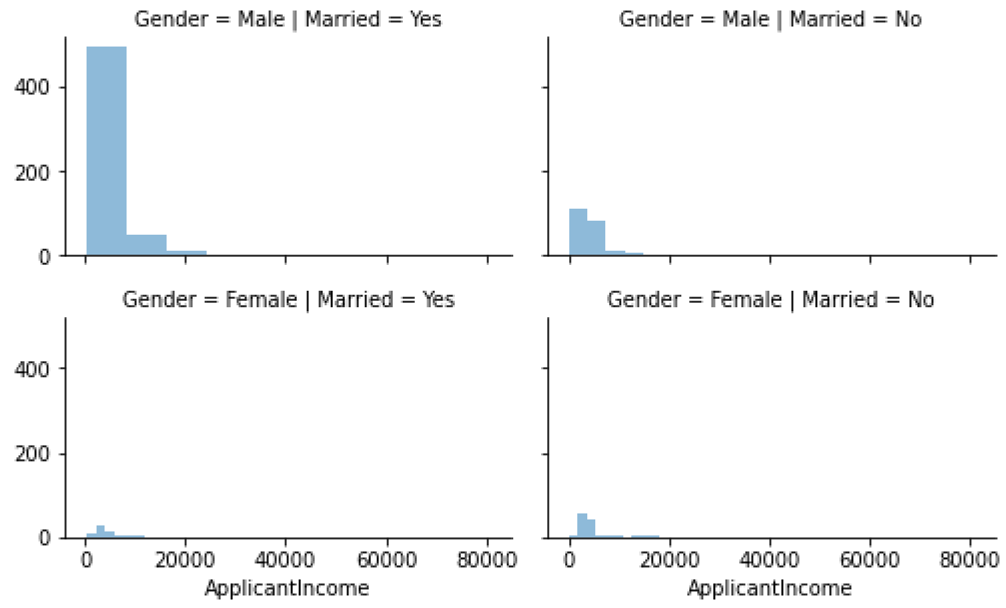


> According to the pie chart, there are more approved loans that disapproved.

Distribution Analysis

```
grid=sns.FacetGrid(df, row='Gender', col='Married', size=2.2, aspect=1.6)
grid.map(plt.hist, 'ApplicantIncome', alpha=.5, bins=10)
grid.add_legend()
```

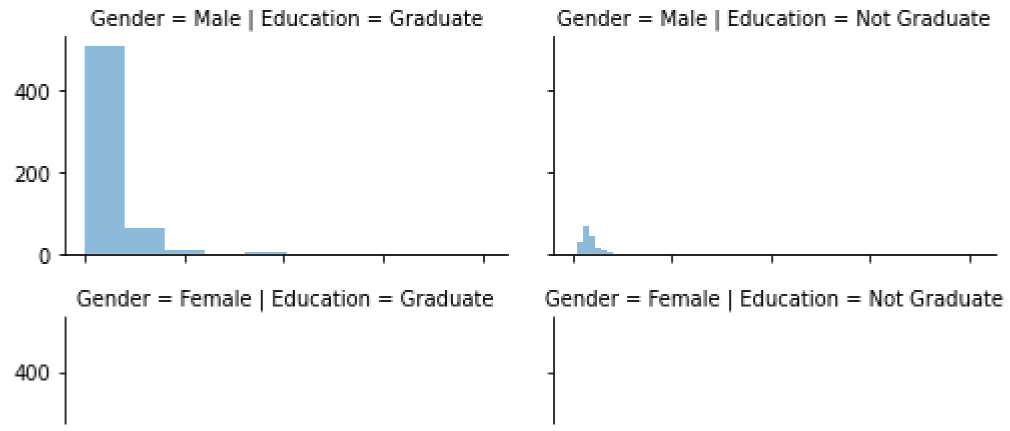
```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:316: UserWarning: The `size`
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.FacetGrid at 0x7fc3ebad3e50>
```



> Males have the highest income according to the data. Males that are married have greater income than unmarried male. And the same goes for females.

```
grid=sns.FacetGrid(df, row='Gender', col='Education', size=2.2, aspect=1.6)
grid.map(plt.hist, 'ApplicantIncome', alpha=.5, bins=10)
grid.add_legend()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:316: UserWarning: The `size`
  warnings.warn(msg, UserWarning)
```



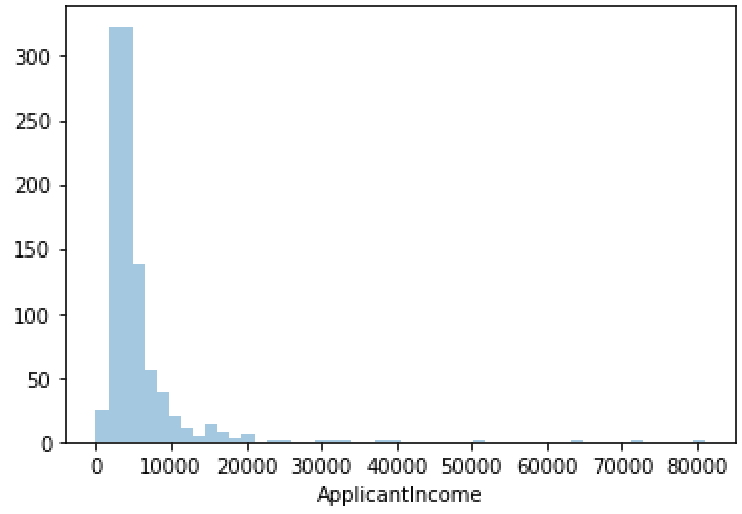
> A graduate who is a male has more income than a one whithout and the same goes for females.

Here I am exploring the distribution of the numerical variables mainly the Applicant income and the Loan amount.

What can be noticed are quite a few outliers.

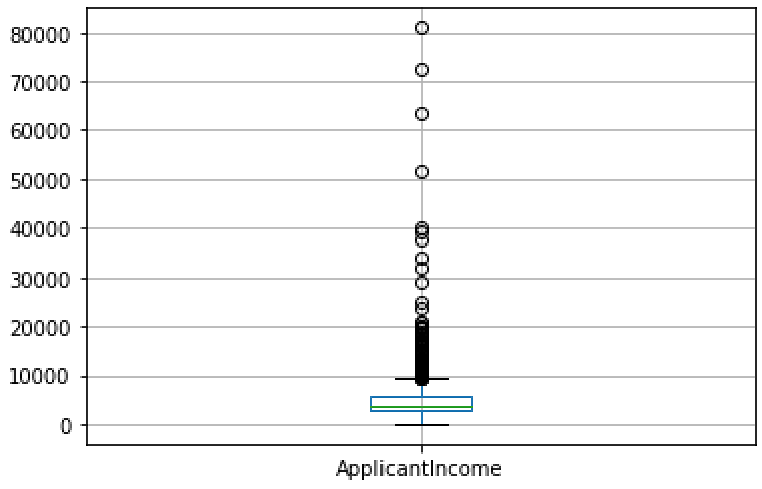
```
sns.distplot(df.ApplicantIncome,kde=False)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
```



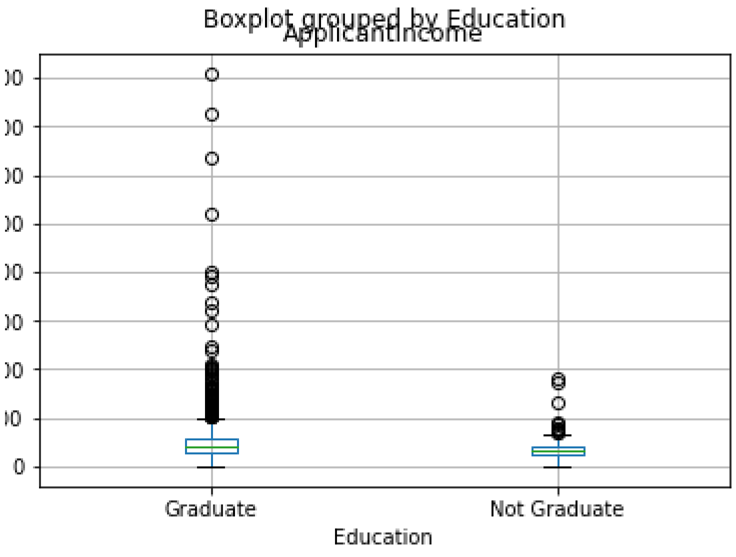
> People with better education should normally have a higher income, we can check that by plotting the education level against the income.

```
df.boxplot(column='ApplicantIncome')
plt.show()
```



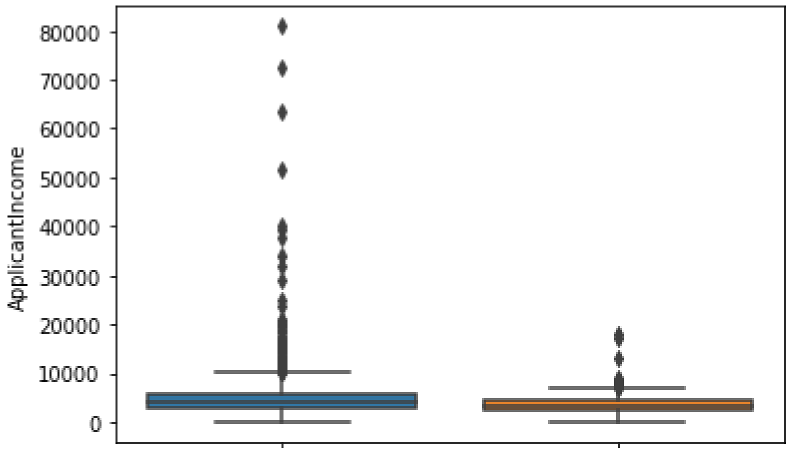
```
df.boxplot(column='ApplicantIncome', by = 'Education')
plt.show()
```

```
·/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarning:
:turn array(a, dtype, copy=False, order=order)
```



> We can conclude that there is no substantial difference between the mean income of graduate and non-graduates. However, there are a higher number of graduates with very high incomes, which are appearing to be the outliers.

```
sns.boxplot(x='Education',y='ApplicantIncome',data=df)
plt.show()
```

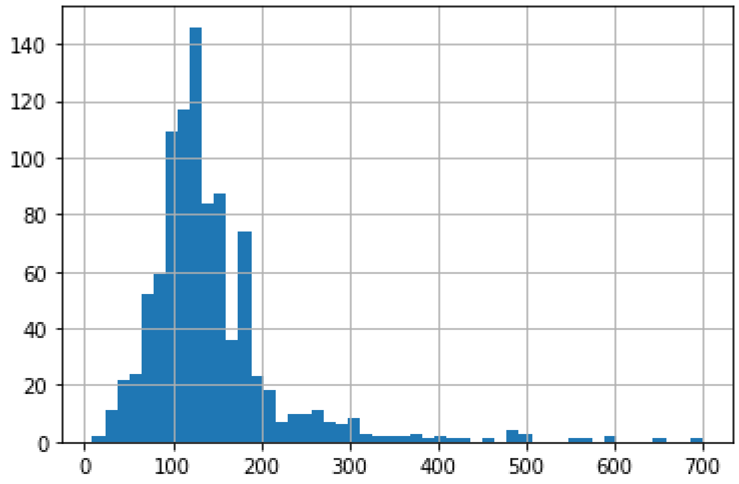


> The distributions shows that the graduates have more outliers which means that the people with huge income are most likely to be educated.

Examining the data distributions of the features. I will start with the target variable, Loan Amount, to make sure it's normally distributed.

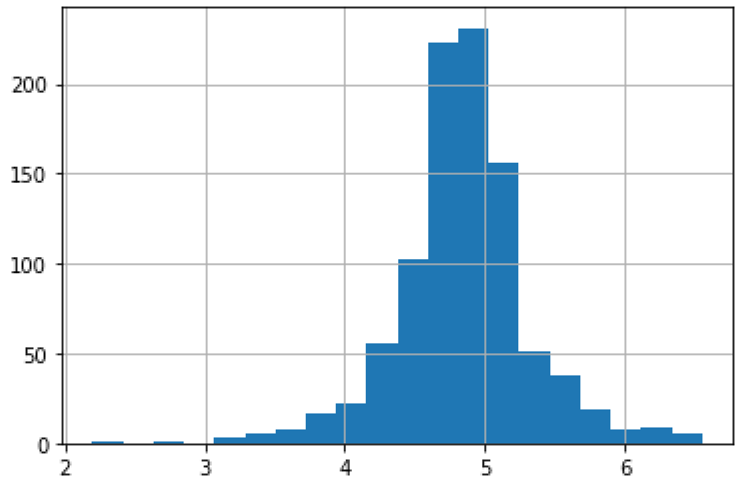
This is important because most machine learning algorithms make the assumption that the data is normally distributed. When data fits a normal distribution, statements about the price using analytical techniques will be made.

```
df['LoanAmount'].hist(bins=50)
plt.show()
```



Transforming the `Loan Amount` into normal distribution.

```
df['LoanAmount_log'] = np.log(df['LoanAmount'])
df['LoanAmount_log'].hist(bins=20)
plt.show()
```



It can be seen that the `Loan Amount` distribution is not skewed after the transformation, but normally distributed.

> The transformed data will be used in in the dataframe and remove the skewed distribution: **Normally distributed** means that the data is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean.

Sometimes loan will be also given based on the co applicant income. Some people might have a low income, but strong CoappliantIncome, so a good idea would be to combine them in a TotalIncome column.

```
df['LoanAmount_log']=np.log(df['LoanAmount'])
df['TotalIncome']= df['ApplicantIncome'] + df['CoapplicantIncome']
df['TotalIncome_log']=np.log(df['TotalIncome'])
plt.show()
```

```
sns.distplot(df.TotalIncome,kde=False)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `d
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fc3ec4cf310>
```

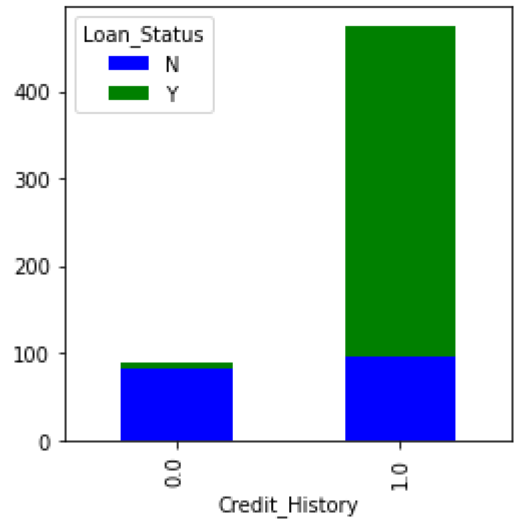
Probability of getting a loan

Checking the credit history from the dataset because it is a vital part of the decision whether a person is proper for a loan.

```
temp_credit = df['Credit_History'].value_counts(ascending=True)
print ('Frequency Table for Credit History:')
print (temp_credit)
```

```
Frequency Table for Credit History:
0.0    148
1.0    754
Name: Credit_History, dtype: int64
```

```
temp3 = pd.crosstab(df['Credit_History'], df['Loan_Status'])
temp3.plot(kind='bar', stacked=True, color=['blue','green'], grid=False, figsize=(4,4))
plt.show()
```



> This shows that the chances of getting a loan are higher if the applicant has a valid credit history.

Data cleaning & Data processing

Converting all categorical variables into numeric ones to use them in the training of the models.

One-Hot Encoding: This process takes categorical variables and converts them to a numerical representation without an arbitrary ordering. What computers know is numbers and for machine learning it is vital to accommodate the feautures into numeric values.

```
numeric_gender = {'Female': 1, 'Male': 2}
df ['Gender'] = df['Gender'].map(numeric_gender)
numeric_married = {'Yes': 1, 'No': 2}
df ['Married'] = df['Married'].map(numeric_married)
numeric_edu = {'Graduate': 1, 'Not Graduate': 2}
df ['Education'] = df['Education'].map(numeric_edu)
numeric_self = {'Yes': 1, 'No': 2}
df ['Self_Employed'] = df['Self_Employed'].map(numeric_self)
numeric_loan = {'Y': 1, 'N': 2}
df ['Loan_Status'] = df['Loan_Status'].map(numeric_loan)
numeric_property = {'Rural': 1, 'Urban': 2, 'Semiurban': 3}
df ['Property_Area'] = df['Property_Area'].map(numeric_property)
numeric_d = {'3+': 3}
df ['Dependents'] = df['Dependents'].map(numeric_d)
```

Checking if there are any null values and if so, which.

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

```
     Loan_ID                0
     Gender                24
     Married                3
     Dependents           890
     Education              0
     Self_Employed         55
     ApplicantIncome        0
     CoapplicantIncome      0
     LoanAmount            27
     Loan_Amount_Term      20
     Credit_History        79
     Property_Area          0
     Loan_Status          367
     LoanAmount_log        27
     TotalIncome            0
     TotalIncome_log        0
     dtype: int64
```

Checking if the percentage of missing values of each value and whic has to be dropped if any.

```
# Find columns with missing values and their percent missing
df.isnull().sum()
miss_val = df.isnull().sum().sort_values(ascending=False)
miss_val = pd.DataFrame(data=df.isnull().sum().sort_values(ascending=False), columns=['Missva

# Add a new column to the dataframe and fill it with the percentage of missing values
miss_val['Percent'] = miss_val.MissvalCount.apply(lambda x : '{:.2f}'.format(float(x)/df.shap
miss_val = miss_val[miss_val.MissvalCount > 0].style.background_gradient(cmap='Reds')
miss_val
```

	MissvalCount	Percent
Dependents	890	90.72
Loan_Status	367	37.41
Credit_History	79	8.05
Self_Employed	55	5.61
LoanAmount_log	27	2.75
LoanAmount	27	2.75
Gender	24	2.45
Loan_Amount_Term	20	2.04
Married	3	0.31

> The light red color shows the small amount of NaN values. If the features were with a high than 50% of missing values, they would have to be removed. Yet, in this case, they have relatively low percentage so they can be used in future. Then, the NaN values will be replaced. The only feature that will be dropped is the `Dependents`.

```
df.drop(['Dependents'], axis=1, inplace=True)
```

Filling up the null values in order to train the model.

Filling the missing values, for categorical we can fill them with the mode (the value with the highest frequency). The best practice is to use mode with data points such as salary field or any other kind of money.

```
df['Gender'] = df['Gender'].fillna(
df['Gender'].dropna().mode().values[0] )
df['Married'] = df['Married'].fillna(
df['Married'].dropna().mode().values[0] )
df['Self_Employed'] = df['Self_Employed'].fillna(
df['Self_Employed'].dropna().mode().values[0] )
df['LoanAmount'] = df['LoanAmount'].fillna(
df['LoanAmount'].dropna().median() )
```

```
df['LoanAmount'].dropna().median() )
df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(
df['Loan_Amount_Term'].dropna().mode().values[0] )
df['LoanAmount_log'] = df['LoanAmount_log'].fillna(
df['LoanAmount_log'].dropna().mode().values[0] )
df['Credit_History'] = df['Credit_History'].fillna(
df['Credit_History'].dropna().mode().values[0] )
df['Loan_Status'] = df['Loan_Status'].fillna(
df['Loan_Status'].dropna().mode().values[0] )
df['TotalIncome_log'] = df['TotalIncome_log'].fillna(
df['TotalIncome_log'].dropna().mode().values[0] )
```

Checking if there are still any empty values after the filling.

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

```
    Loan_ID              0
    Gender               0
    Married              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
    LoanAmount_log       0
    TotalIncome          0
    TotalIncome_log      0
    dtype: int64
```

Saving the results to a csv file so that the cleaned and processed data can be used in the modelling part of the project.

```
df.to_csv('results.csv')
```

Conclusion

From Data Analysis:

1. There are features that are numeric and also objects. Later, the ones that are not numeric were converted into either float or int in order to be plotted and then used for the trainig of the models.
2. There were also missing values in the dataset, which will be handled later.
3. The shape of the data with the `shape`. Here are the the rows and columns(981, 13).