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


# 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.csv
Saving train1.csv to train1.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
```

```
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
```

```
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

```
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

Data cleaning

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

```
Loan_ID                0
Gender                24
Married                3
Dependents            25
Education              0
Self_Employed         55
ApplicantIncome        0
CoapplicantIncome      0
LoanAmount            27
Loan_Amount_Term      20
Credit_History        79
Property_Area          0
Loan_Status          367
dtype: int64
```

```
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
```

Converting the string values to 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)
```

```
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    float64
 2   Married            978 non-null    float64
 3   Dependents         956 non-null    object
 4   Education          981 non-null    int64
 5   Self_Employed      926 non-null    float64
 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    int64
 12  Loan_Status        614 non-null    float64
dtypes: float64(8), int64(3), object(2)
memory usage: 107.3+ KB
```

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

```
Loan_ID                0
Gender                24
Married                3
Dependents            25
Education              0
Self_Employed         55
ApplicantIncome        0
CoapplicantIncome      0
LoanAmount            27
Loan_Amount_Term      20
Credit_History        79
Property_Area          0
Loan_Status          367
dtype: int64
```

Data analysis:

```
df.describe()
```

	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIn
count	957.000000	978.000000	981.000000	926.000000	981.000000	981.00
mean	1.809822	1.354806	1.222222	1.871490	5179.795107	1601.91
std	0.392646	0.478699	0.415952	0.334837	5695.104533	2718.77
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.00
25%	2.000000	1.000000	1.000000	2.000000	2875.000000	0.00
50%	2.000000	1.000000	1.000000	2.000000	3800.000000	1110.00
75%	2.000000	2.000000	1.000000	2.000000	5516.000000	2365.00
max	2.000000	2.000000	2.000000	2.000000	81000.000000	41667.00

Here we can see the shape of our data with the `.shape`. Here we see (614, 13) this means that we have a 614 rows and 13 columns

```
df.shape
```

```
(981, 13)
```

Here we can see the shape of our test data with the .shape. Here we see (367, 12) this means that

To view what data that is stored we can use .columns. This will return the colums of our 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')
```

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

```
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Co
0	LP001015	2.0	1.0	0	1	2.0	5720	
1	LP001022	2.0	1.0	1	1	2.0	3076	
2	LP001031	2.0	1.0	2	1	2.0	5000	
3	LP001035	2.0	1.0	2	1	2.0	2340	
4	LP001051	2.0	2.0	0	2	2.0	3276	

```
df.tail()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
609	LP002978	1.0	2.0	0	1	2.0	2900
610	LP002979	2.0	1.0	3+	1	2.0	4106
611	LP002983	2.0	1.0	1	1	2.0	8072
612	LP002984	2.0	1.0	2	1	2.0	7583
613	LP002990	1.0	2.0	0	1	1.0	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    float64
  2   Married            978 non-null    float64
  3   Dependents         956 non-null    object
  4   Education          981 non-null    int64
  5   Self_Employed      926 non-null    float64
  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    int64
  12  Loan_Status        614 non-null    float64
 dtypes: float64(8), int64(3), object(2)
 memory usage: 107.3+ KB
```

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

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	2.0	2.0	0	1	2.0	5849
380	LP002226	2.0	1.0	0	1	NaN	3333
379	LP002225	2.0	1.0	2	1	2.0	5391
377	LP002223	2.0	1.0	0	1	2.0	4310
376	LP002219	2.0	1.0	3+	1	2.0	8750
375	LP002211	2.0	1.0	0	1	2.0	4817
374	LP002209	1.0	2.0	0	1	NaN	2764
372	LP002201	2.0	1.0	2	1	1.0	9323
371	LP002197	2.0	1.0	2	1	2.0	5185
370	LP002194	1.0	2.0	0	1	1.0	15759
368	LP002190	2.0	1.0	1	1	2.0	6325

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

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001015	2.0	1.0	0	1	2.0	5720
234	LP001778	2.0	1.0	1	1	2.0	3155
235	LP001784	2.0	1.0	1	1	2.0	5500
236	LP001786	2.0	1.0	0	1	NaN	5746
237	LP001788	1.0	2.0	0	1	1.0	3463
238	LP001790	1.0	2.0	1	1	2.0	3812
239	LP001792	2.0	1.0	1	1	2.0	3315
240	LP001798	2.0	1.0	2	1	2.0	5819
242	LP001806	2.0	2.0	0	1	2.0	2965
243	LP001807	2.0	1.0	2	1	1.0	6250
245	LP001813	2.0	2.0	0	1	1.0	6050
233	LP001776	1.0	2.0	0	1	2.0	8333
246	LP001814	2.0	1.0	2	1	2.0	9703
249	LP001825	2.0	1.0	0	1	2.0	1809
251	LP001836	1.0	2.0	2	1	2.0	3427
254	LP001844	2.0	2.0	0	1	1.0	16250
255	LP001846	1.0	2.0	3+	1	2.0	3083
257	LP001854	2.0	1.0	3+	1	2.0	5250
258	LP001859	2.0	1.0	0	1	2.0	14683
260	LP001865	2.0	1.0	1	1	2.0	6083
261	LP001868	2.0	2.0	0	1	2.0	2060
262	LP001870	1.0	2.0	1	1	2.0	3481
263	LP001871	1.0	2.0	0	1	2.0	7200
248	LP001824	2.0	1.0	1	1	2.0	2882
231	LP001768	2.0	1.0	0	1	NaN	3716
230	LP001765	2.0	1.0	1	1	2.0	2491
229	LP001761	2.0	2.0	0	1	1.0	6400
196	LP001666	2.0	2.0	0	1	2.0	8333
198	LP001671	1.0	1.0	0	1	2.0	3416
199	LP001673	2.0	2.0	0	1	1.0	11000

201	LP001677	2.0	2.0	2	1	2.0	4923
206	LP001693	1.0	2.0	0	1	2.0	3244

Here we can see one row (one person)

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

```
Loan_ID              LP001015
Gender                      2
Married                     1
Dependents                  0
Education                   1
Self_Employed               2
ApplicantIncome          5720
CoapplicantIncome           0
LoanAmount                110
Loan_Amount_Term          360
Credit_History              1
Property_Area               2
Loan_Status               NaN
Name: 0, dtype: object
```

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

224	LP001750	2.0	1.0	0	1	2.0	6250

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

```
1.0    422
2.0    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()
```

```
2.0    775
1.0    182
```

```
Name: Gender, dtype: int64
```

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

```
1.0    631
2.0    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
1     160
2     160
3+     91
Name: Dependents, dtype: int64
```

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

```
1    763
2    218
Name: Education, dtype: int64
```

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

```
2.0    807
1.0    119
Name: Self_Employed, dtype: int64
```

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

```
array([nan,  1.,  2.])
```

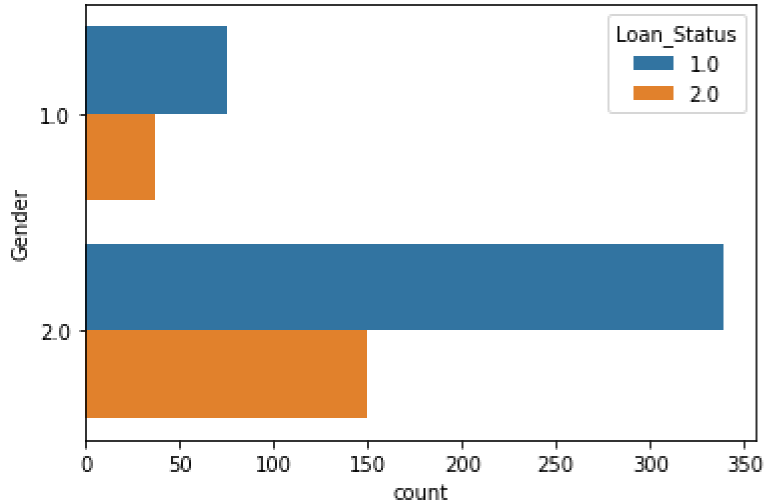
```
df['ApplicantIncome'].unique()
```

```
array([ 5720,  3076,  5000,  2340,  3276,  2165,  2226,  3881, 13633,
```

```
  2400,  3091,  2185,  4166, 12173,  4666,  5667,  4583,  3786,
  9226,  1300,  1888,  2083,  3909,  3765,  5400,     0,  4363,
  7500,  3772,  2942,  2478,  6250,  3268,  2783,  2740,  3150,
  7350,  2267,  5833,  3643,  5629,  3644,  1750,  6500,  3666,
  4260,  4163,  2356,  6792,  8000,  2419,  3500,  4116,  5293,
  2750,  4402,  3613,  2779,  4720,  2415,  7016,  4968,  2101,
  4490,  2917,  4700,  3445,  7666,  2458,  3250,  4463,  4083,
  3900,  4750,  3583,  3189,  6356,  3413,  7950,  3829, 72529,
  4136,  8449,  4456,  4635,  3571,  3066,  3235,  5058,  3188,
 13518,  4364,  4766,  4609,  6260,  3333,  9719,  6835,  4452,
  2262,  3901,  2687,  2243,  3417,  1596,  4513,  4500,  4523,
  4742,  4082,  2922,  4167,  4243,  1516,  1762,   724,  3125,
  2333,  3350,  2500, 12500,  4667,  3073,  3321,  3391,  3343,
  3620,  4000,  4258,  2014,  4727,  3089,  6794, 32000, 10890,
 12941,  8703,  5900,  3071,  2463,  4855,  1599,  4246,  4333,
  5823,  7895,  4150,  2964,  5583,  2708,  3180,  2268,  1141,
  3042,  3564,  3958,  4483,  5225,  3017,  2431,  4912,  2918,
  5128, 15312,  4334,  4358, 10166,  4521,  9167, 13083,  7874,
  3785,  2654, 10000,  4796,  2000,  2540,  1900,  8706,  2855,
  3016,  3159,  1937,  2613,  4960,  3074,  4213,  2038,  2362,
  5333,  5384,  5708,  3754,  2914,  2747,  7830,  3507,  3747,
  2166,  2896,  5062,  5184,  2545,  2553,  3436,  2412,  5180,
 14911,  2860,  1173,  7600,  2157,  2231,  2274,  6166,  2513,
  3844,  3887,  3510,  2539,  2107,  3186,  3943,  2925,  3242,
  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,
```

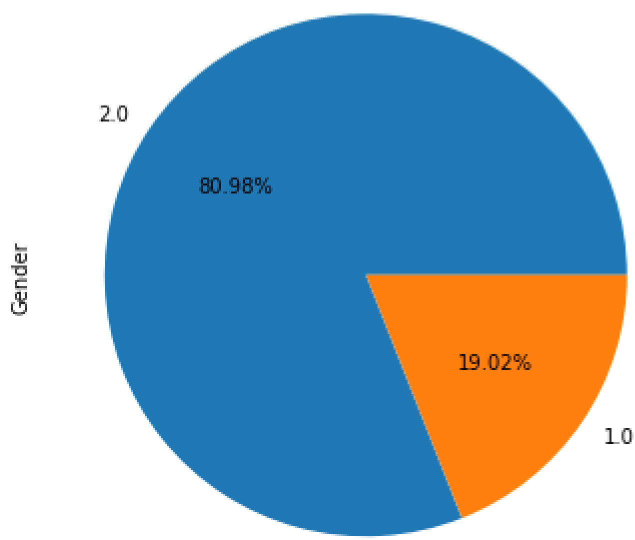
```
sns.countplot(y = 'Gender', hue = 'Loan_Status', data = df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f99cc032c90>
```




```
df['Gender'].value_counts().plot(kind='pie', autopct='%1.2f%%', figsize=(6, 6))
```

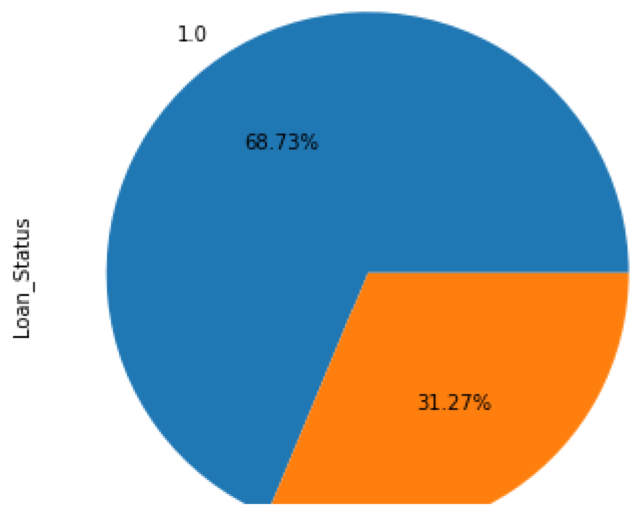
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f99e0c59f50>
```



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

```
df['Loan_Status'].value_counts().plot(kind='pie', autopct='%1.2f%%', figsize=(6, 6))
```

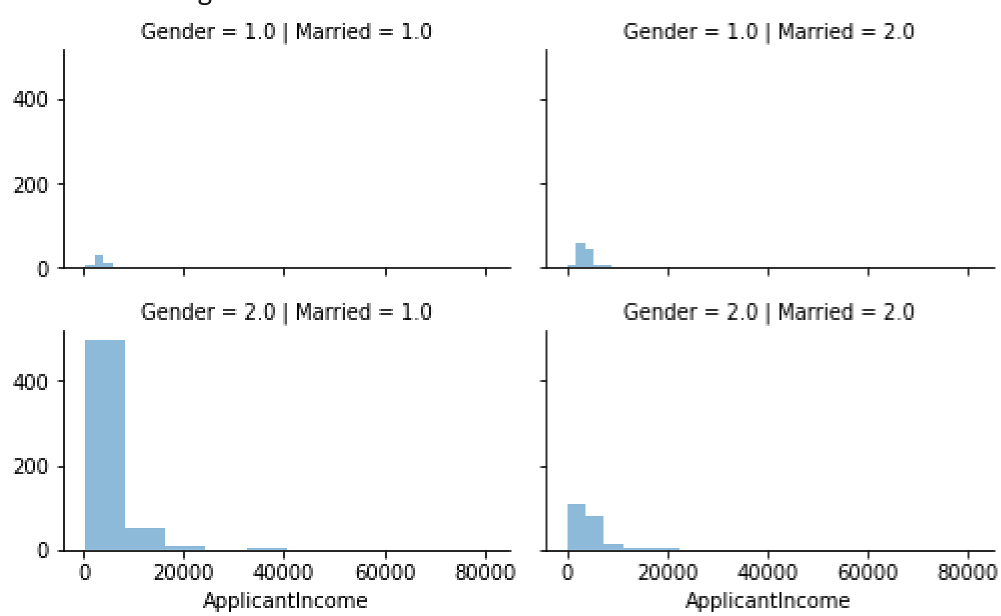
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f99cd11d410>
```



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

```
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 0x7f99cb69e8d0>
```

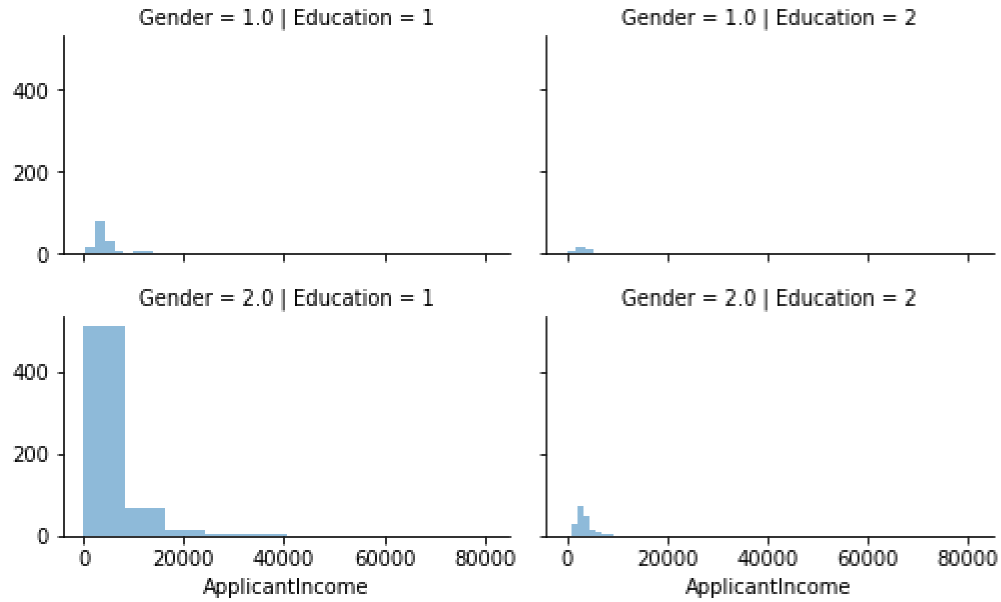


> Males have the highest income according to the data. Males that are married have greater income that unmarried male. And the same goes for female.

```
grid=sns.FacetGrid(df, row='Gender', col='Education', size=2.2, aspect=1.6)
```

```
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()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:316: UserWarning: The `size`
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.FacetGrid at 0x7f99cb69e3d0>
```



> 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)
```

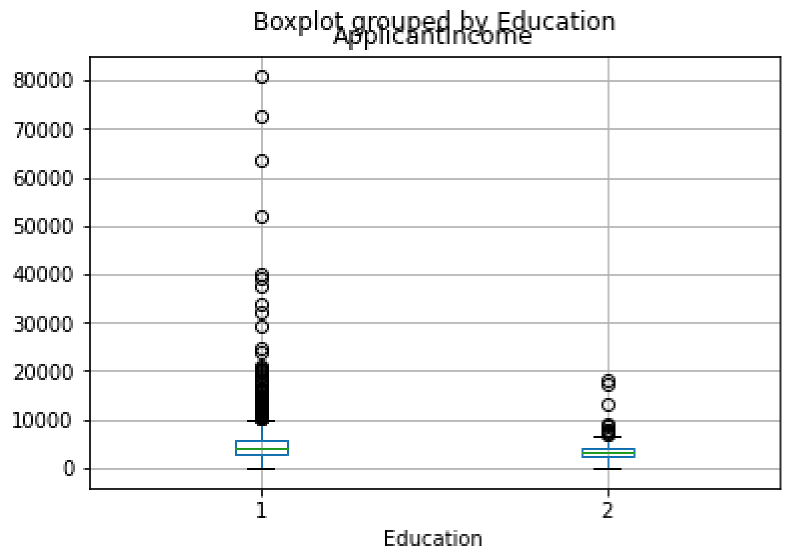
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f99c262d090>
```

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

150

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

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarr
  return array(a, dtype, copy=False, order=order)
<matplotlib.axes._subplots.AxesSubplot at 0x7f99c2587fd0>
```



> We can conclude that there is no substantial different 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.

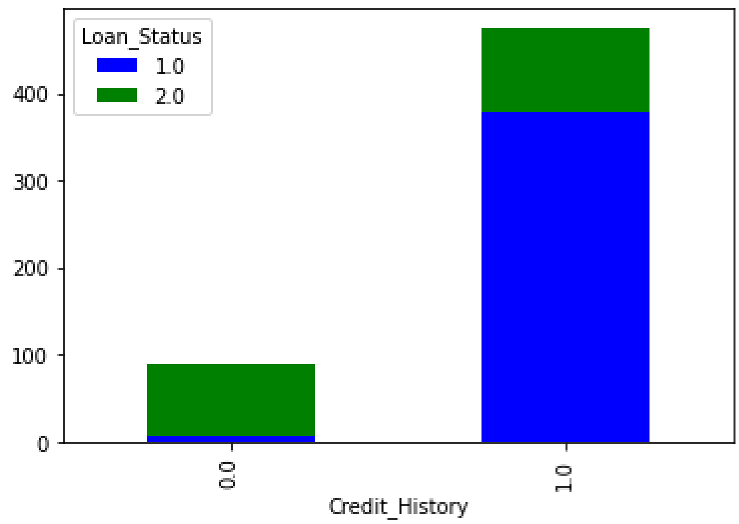
```
sns.boxplot(x='Education',y='ApplicantIncome',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f99c24be3d0>
```

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


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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f99c23e5490>
```



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

Data processing:

Checking if there are certain missing values that need to be fixed.

```
total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

	Total	Percent
Loan_Status	367	0.374108
Credit_History	79	0.080530
Self_Employed	55	0.056065
LoanAmount	27	0.027523
Dependents	25	0.025484
Gender	24	0.024465
Loan_Amount_Term	20	0.020387
Married	3	0.003058
Property_Area	0	0.000000
CoapplicantIncome	0	0.000000
ApplicantIncome	0	0.000000

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['Dependents'] = df['Dependents'].fillna(
df['Dependents'].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['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(
df['Loan_Amount_Term'].dropna().mode().values[0] )
df['Credit_History'] = df['Credit_History'].fillna(
df['Credit_History'].dropna().mode().values[0] )
```

Checking if there any empty values.

```
df.isnull().all()
```

```
Loan_ID              False
Gender               False
Married              False
Dependents           False
Education            False
```

```
Self_Employed        False
ApplicantIncome      False
CoapplicantIncome    False
LoanAmount           False
Loan_Amount_Term     False
Credit_History       False
Property_Area        False
Loan_Status          False
dtype: bool
```

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

```
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          367
dtype: int64
```

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'])
```

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

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f99c2305950>
```

Modeling:

Encoding to numeric data in order to start the training of the models.

```
#drop the uniques loan id
df.drop('Loan_ID', axis = 1, inplace = True)
```

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

```
2.0    799
1.0    182
Name: Gender, dtype: int64
```

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

```
0     570
1     160
2     160
3+     91
Name: Dependents, dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 981 entries, 0 to 613
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             981 non-null    float64
 1   Married            981 non-null    float64
 2   Dependents         981 non-null    object
 3   Education          981 non-null    int64
 4   Self_Employed      981 non-null    float64
 5   ApplicantIncome    981 non-null    int64
 6   CoapplicantIncome  981 non-null    float64
 7   LoanAmount         981 non-null    float64
 8   Loan_Amount_Term   981 non-null    float64
 9   Credit_History     981 non-null    float64
 10  Property_Area      981 non-null    int64
 11  Loan_Status        614 non-null    float64
 12  LoanAmount_log     981 non-null    float64
 13  TotalIncome        981 non-null    float64
 14  TotalIncome_log    981 non-null    float64
dtypes: float64(11), int64(3), object(1)
memory usage: 142.6+ KB
```

Need to covnvert the object values to numeric ones - `Dependents` needs to become an int.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 981 entries, 0 to 613
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             981 non-null    float64
 1   Married            981 non-null    float64
 2   Dependents         981 non-null    object 
 3   Education          981 non-null    int64  
 4   Self_Employed      981 non-null    float64
 5   ApplicantIncome    981 non-null    int64  
 6   CoapplicantIncome  981 non-null    float64
 7   LoanAmount         981 non-null    float64
 8   Loan_Amount_Term   981 non-null    float64
 9   Credit_History     981 non-null    float64
 10  Property_Area      981 non-null    int64  
 11  Loan_Status        614 non-null    float64
 12  LoanAmount_log     981 non-null    float64
 13  TotalIncome        981 non-null    float64
 14  TotalIncome_log    981 non-null    float64
dtypes: float64(11), int64(3), object(1)
memory usage: 142.6+ KB
```

Heatmaps are very useful to find relations between two variables in a dataset and this way the user gets a visualisation of the numeric data. No correlations are extremely high. Each square shows the correlation between the variables on each axis.

- The correlations between `LoanAmount` and `ApplicantIncome` can be explained:

> The close to 1 the correlation is the more positively correlated they are; that is as one increases so does the other and the closer to 1 the stronger this relationship is. It is noticable that the correlation between the `ApplicantIncome` and `LoanAmount` is 0.57, which mean that they have a positive correlation, but not strong.

```
sns.heatmap(df.corr(), annot = True, cmap = 'magma')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f99c2277e10>
```

Importing sklearn libraries

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
```

Splitting into train and test set after choosing the right features X and labels y

```
y = df['Loan_Status']
X = df.drop('Loan_Status', axis = 1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
```

```
numeric_features = df.select_dtypes(include=[np.number])
```

```
numeric_features.columns
```

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

```
# use only those input features with numeric data type
df = df.select_dtypes(include=["int64","float64"])
```

```
# set the target and predictors
y = df.Loan_Status  # target

# use only those input features with numeric data type
df_temp = df.select_dtypes(include=["int64","float64"])
```

```
X = df_temp.drop(["Loan_Status"],axis=1)  # predictors
```

Logistic Regression

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_reg=model.predict(X_test)
evaluation = f1_score(y_test, y_reg)
evaluation
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-129-ae0c7bcd60e1> in <module>()
      1 model = LogisticRegression()
----> 2 model.fit(X_train, y_train)
      3 y_reg=model.predict(X_test)
      4 evaluation = f1_score(y_test, y_reg)
      5 evaluation

5 frames

/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py in asarray(a, dtype,
order)
     81
     82     """
---> 83     return array(a, dtype, copy=False, order=order)
     84
     85

ValueError: could not convert string to float: '3+'
```

SEARCH STACK OVERFLOW

Decision tree:

1. Creating classifier
2. Fitting classifier with train data

```
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Do predictions on a test set. **Testing** the model by testing the test data.

```
y_tree=tree.predict(X_test)
print(y_tree)
```

```
[0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1
 1 1 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 0 0 1 1 1 0 1 1 1 1 0
 1 1 0 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0 1 0 1 1 1 1 1 1 0 0
 1 1 1 0 0 0 1 1 0 1 0 0]
```

Evaluate classsifier, measure accuracy, which is 0.76

```
evaluation = f1_score(y_test, y_tree)
evaluation
```

```
0.7619047619047619
```

Random forests

```
forest = RandomForestClassifier()
forest.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Testing the model by testing the test data.

```
y_forest=forest.predict(X_test)
print(y_forest)
```

```
[1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1
 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1
 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 0 0 1 1 1 1 1 0 1]
```

Result of the **accuracy**.

```
evaluation_f= f1_score(y_test, y_forest)
evaluation_f
```

```
0.8795811518324607
```

Conclusion

From the Exploratory Data Analysis, it can be concluded:

1. There amount of male applicants seems to be greater than the female ones and they tend to live in the semisuburbian areas.
2. There are more positive than negative loan statuses - more approvals.
3. The distributions shows that the graduates have more outliers which means that the people with huge income are most likely to be educated.
4. Males have the highest income according to the data. Males that are married have greater income that unmarried male. And the same goes for female. Therefore, there is a greater chance for educated and married people to receive a loan than applicant who are not.

From the Modelling, it can be concluded:

1. After the exploring of different types of modelling, that the more accurate model is Random forest than Decision tree.
2. From the evaluation of the three models, it can be noticed that the Logistic Regression performed better than others