Assignment 3: Logistic Regression

Problem statement

You are working as a Data Scientist at a Finance company which proposes home loans. When a customer applies for a home loan, the company studies his demand to decide whether he/she is eligible or not.

Your boss asks you to develop a decision-aid tool to automate the loan eligibility process. The tool is supposed to be given as input the information provided by the customer while filling his/her online loan application form. These information include customer's personal citeria such as Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History, etc... To develop the algorithm/model to be applied in the tool, you are given a data set containing historical information about applicants (**features**) as well as their loan status (**output** given as binary variable indicating whether or not a the loan was approved).

image.png

1- Load data

```
# imports
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# show plots in the notebook
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=False)

df = pd.read_csv('/content/drive/MyDrive/loan_prediction.csv')
df.shape
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r
(614, 13)
```

```
assert df.shape == (614, 13)
```

```
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000

2- Remove the "Loan_ID" column, and then transform the "Loan_Status" feature from categorical into numerical values (Y ==> 1, N ==> 0)

```
df.drop("Loan_ID",1, inplace=True)
df["Loan_Status"] = df["Loan_Status"].astype(str).astype("category").cat.codes
```

```
assert df.shape == (614, 12); assert df["Loan_Status"].dtype != 'O'
```

Data Exploration

3- Build a dataframe named `stats` displaying information about df columns. The index is the columns names and the columns are the following :

- `type` : type of the column
- `# null` : number of null values
- `# unique` : number of unique values
- `unique values` : unique values, concatenated as one text separated with `-`. If the number of unique values is greater than equal 10 (>= 10), print only the the first *20* characters followed by `...`

`stats` would look like this (only the last rows are displayed)

```
stats = pd.DataFrame(columns=["type","# null","# unique","unique values"])

stats["type"] = df.dtypes
stats["# null"] = df.isnull().sum()
stats["# unique"] = df.nunique()

for col in df:
    uniqueVal = df[col].unique()
    uniqueVal = "-".join(str(x) for x in uniqueVal)
    stats["unique values"][col] = uniqueVal

stats
```

	type	# null	# unique	unique
Gender	object	13	2	Male-Fema
Married	object	3	2	No-Y
Dependents	object	15	4	0-1-2-
Education	object	0	2	Graduate-Not Gr
Self_Employed	object	32	2	No-Y
ApplicantIncome	int64	0	505	5849-4583-3000-2583-6000-5417-2333-3036-4(
CoapplicantIncome	float64	0	287	0.0-1508.0-2358.0-4196.0-1516.0-2504.0-15
LoanAmount	float64	22	203	nan-128.0-66.0-120.0-141.0-267.0-95.0-158
Loan_Amount_Term	float64	14	10	360.0-120.0-240.0-nan-180.0-60.0-300.0-48
Credit_History	float64	50	2	1.0-(
Property_Area	object	0	3	Urban-Rural-Sen
Loan_Status	int8	0	2	

4- Fill the null values in the column **[Married]** by the most frequent one (which is "Yes"). Then, remove all the rows containing at least one NullValue.

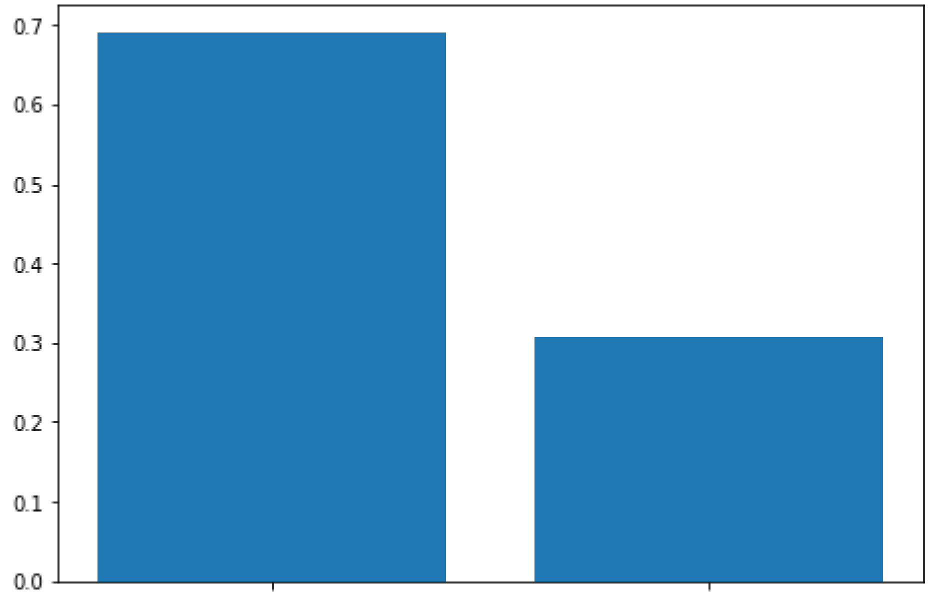
```
df = df.fillna(value= {"Married": "Yes"})
df.dropna(inplace=True)
print(df.shape)

    (480, 12)
```

```
assert df.isnull().sum().sum() == 0
```

Data Visualization

5-1- Plot a **bar plot** of the column **Loan_Status** (% of each class).

```
ax = plt.figure().add_axes([0,0,1,1])
ax.bar(['Yes', 'No'],df.Loan_Status.value_counts(normalize=True))
plt.show()
```



5-2- How many approved and non approved loans are there ? How much accuracy can you get if prediction is always equal to "*1*" ?

```
approuved = (df['Loan_Status'] == 1).sum()
print("Number of approved loans :")
print(approuved)

nonApproved = (df['Loan_Status'] ==0).sum()
print("Number of non approved loans :")
print(nonApproved)
```

```
Number of approved loans :
332
Number of non approved loans :
148
```

```
accuracy = approuved/(approuved + nonApproved)
print(accuracy)
```

```
0.6916666666666667
```

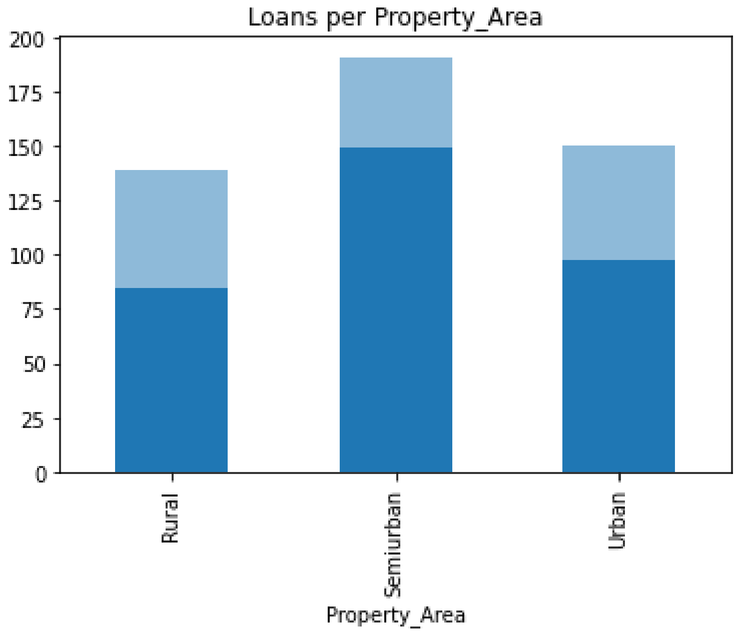
6-1- Create a function to be named **make_bar_plot**, which takes two parameters as input: a dataframe df (parameter 1) and a column name (parameter 2). The function must output a borplot of the approval rate `Loan_Status` by value of the column.

Calling `make_bar_plot` on `Property_Area` should look like this

```
def make_bar_plot(df, column):
    ploter = pd.DataFrame()
    ploter['total']=df.groupby(column).size()
    ploter['Given_Loan'] = df.groupby(column).sum()['Loan_Status']
    ploter['percentage'] = round(ploter['Given_Loan']/ploter['total']*100,2)
    print(ploter)
    ploter['Given_Loan'].plot(kind="bar")
    ploter['total'].plot(kind="bar",alpha=0.5,title="Loans per "+str(column))
    plt.show()
```

```
make_bar_plot(df, 'Property_Area')
```

```
               total  Given_Loan  percentage
Property_Area
Rural            139        85.0       61.15
Semiurban        191       149.0       78.01
Urban            150        98.0       65.33
```



6-2- Create a function to be named **make_box_plot**, which takes two parameters as input: a dataframe df (parameter 1) and a column name (parameter 2). The function must output a boxplot of the distribution of the column by `Loan_Status`.

Calling `make_box_plot` on `ApplicantIncome` should look like this

```
def make_box_plot(df, col):
    df.boxplot(by='Loan_Status',column=col,grid=False)
```

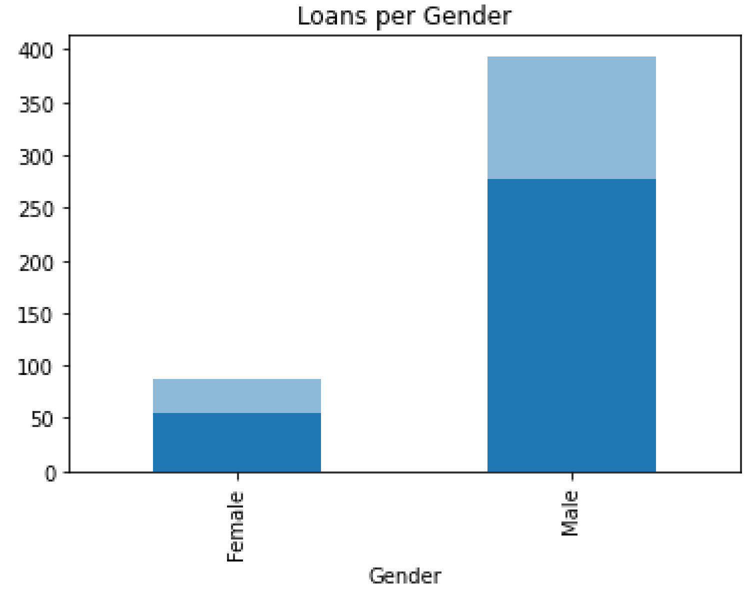
```
make_box_plot(df, 'ApplicantIncome')
```

6-3- Loop through df columns, except for the target and make a plot depending:

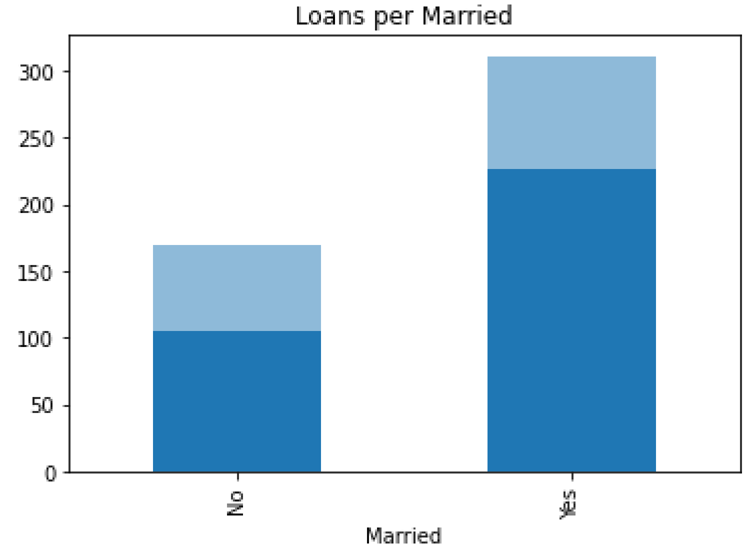
- if the column is numeric, call `make_box_plot`
- else, call `make_bar_plot`

```
from pandas.api.types import is_numeric_dtype

for col in df:
  if(col!="Loan_Status"):
    print(f'****************** {col} ******************')
    if is_numeric_dtype(df[col]):
      make_box_plot(df,col)
    else:
      make_bar_plot(df,col)
```

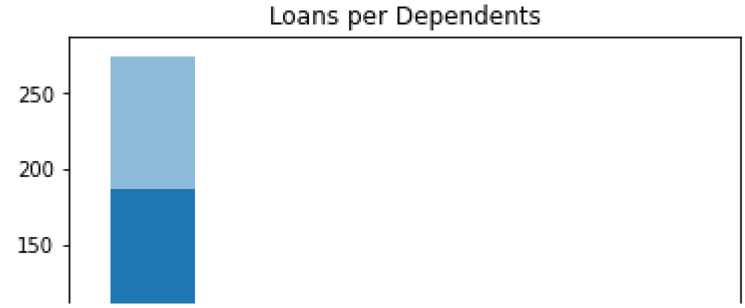
```
****************** Gender *******************
        total  Given_Loan  percentage
Gender
Female     86        54.0       62.79
Male      394       278.0       70.56
```

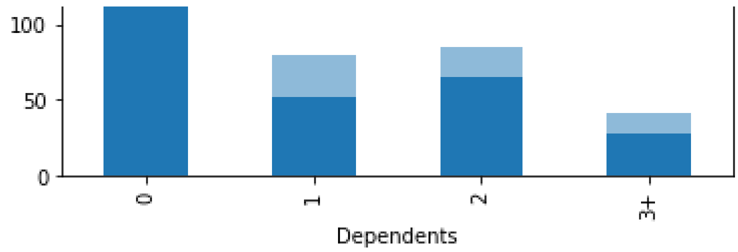


```
****************** Married *******************
         total  Given_Loan  percentage
Married
No         169       105.0       62.13
Yes        311       227.0       72.99
```

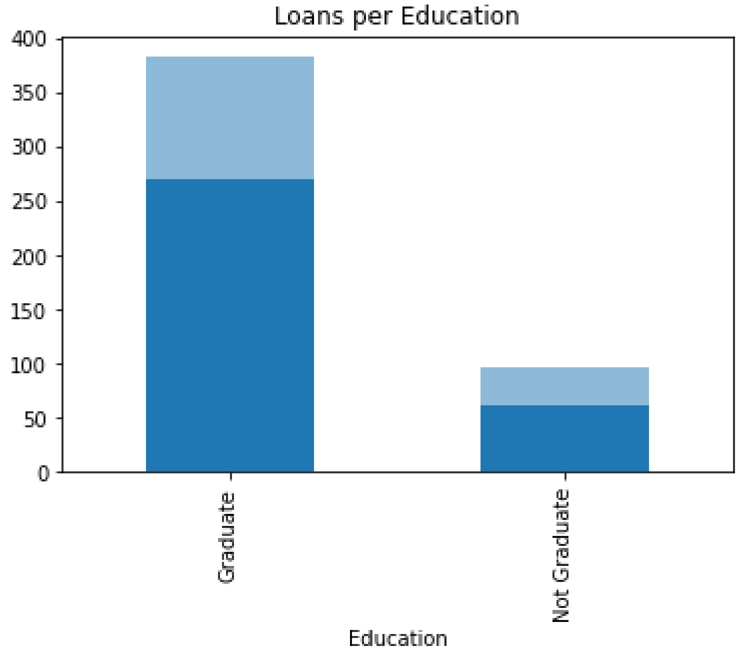


```
****************** Dependents *******************
            total  Given_Loan  percentage
Dependents
0             274       187.0       68.25
1              80        52.0       65.00
2              85        65.0       76.47
3+             41        28.0       68.29
```

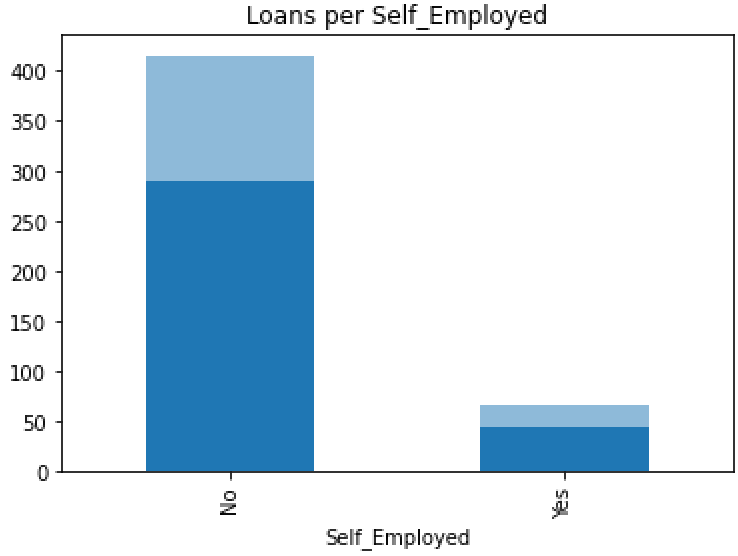




```
***************** Education ******************
              total  Given_Loan  percentage
Education
Graduate        383       271.0       70.76
Not Graduate     97        61.0       62.89
```

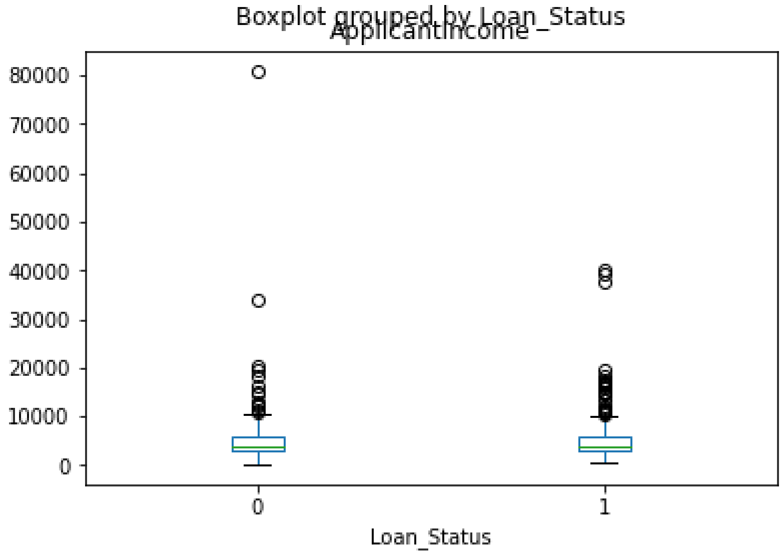


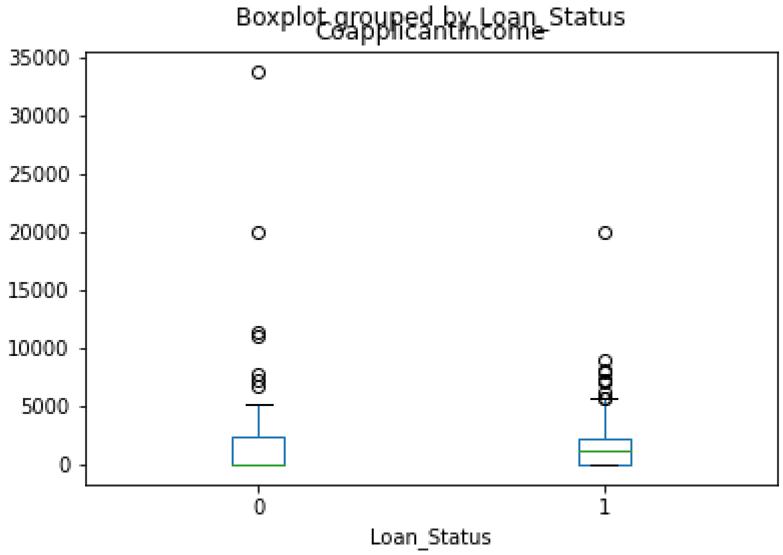
```
***************** Self_Employed ******************
               total  Given_Loan  percentage
Self_Employed
No               414       289.0       69.81
Yes               66        43.0       65.15
```

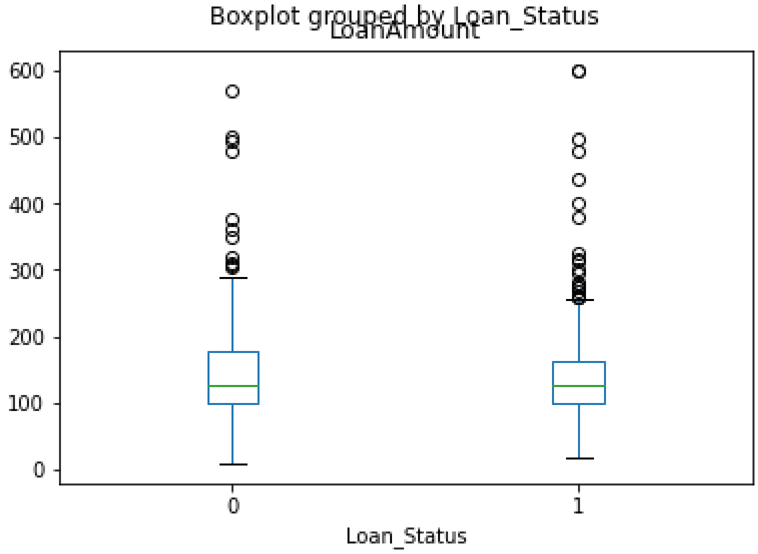


```
***************** ApplicantIncome ******************
***************** CoapplicantIncome ******************
***************** LoanAmount ******************
***************** Loan_Amount_Term ******************
***************** Credit_History ******************
***************** Property_Area ******************
               total  Given_Loan  percentage
```

```
Property_Area
Rural           139       85.0       61.15
Semiurban       191      149.0       78.01
Urban           150       98.0       65.33
```







Boxplot grouped by Loan_Status

Model Training & Evaluation

7-1- Use pandas.get_dummies to transform `Property_Area` and `Dependents` into dummy variables

```
df = pd.get_dummies(df,columns=["Property_Area","Dependents"])
```

print the head of df

```
df.head()
```

	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
1	Male	Yes	Graduate	No	4583	1508.0	
2	Male	Yes	Graduate	Yes	3000	0.0	
3	Male	Yes	Not Graduate	No	2583	2358.0	
4	Male	No	Graduate	No	6000	0.0	
5	Male	Yes	Graduate	Yes	5417	4196.0	

```
assert df.shape == (480, 17)
```

7-2- Use `pandas.Categorical` (as seen in *Titanic* notebook) to transform the remaining text columns of df into numerical ones

```
for col in df:
    if df[col].dtype == "object":
        df[col] = pd.Categorical(df[col]).codes
```

Print the head of df

```
df.head()
```