ML-Model-Flask-Deployment using TMDb dataset

Table of Contents

Introduction

In this jupyter notebook, I attempted to use exploratory data analysis approaches to explore my dataset, then clean and prepare it so that I could develop three distinct models(Logistic regression, KNN, and Decision three) and analyze them to see which one performed the best.

Data Cleaning

General Properties

- Load data
- Get general info and overview
- Identify problems and actions to analyse research questions

In [1]:
```
# Use this cell to set up import statements for all of the packages that you plan to use.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import pickle
```

- **Read data from csv file**

In [2]:
```
df=pd.read_csv('customer.csv')
# Get an overview about my DataSet
df
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Statu
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	
...	...	...	...	...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	

614 rows × 13 columns

In [3]:
```
# Get general info
df.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
```

> **display missing values**

In [4]:
```
df.isnull().sum().sort_values(ascending=False)
```

Out[4]:
```
Credit_History       50
Self_Employed        32
LoanAmount           22
Dependents           15
Loan_Amount_Term     14
Gender               13
Married               3
Loan_ID               0
Education             0
ApplicantIncome       0
CoapplicantIncome     0
Property_Area         0
Loan_Status           0
dtype: int64
```

> **Get an overview about my Categorical data**

In [5]:
```
df.describe(include='O')
```

Out[5]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
count	614	601	611	599	614	582	614	614
unique	614	2	2	4	2	2	3	2
top	LP001002	Male	Yes	0	Graduate	No	Semiurban	Y
freq	1	489	398	345	480	500	233	422

> **Splitting our dataframe into two dataframes one for categorical columns and one for numerical columns for cleaning.**

In [6]:
```
cat_data=[]
num_data=[]
for i,c in enumerate(df.dtypes):
    if c==object:
        cat_data.append(df.iloc[:,i])#add column i in cat_data
    else:
        num_data.append(df.iloc[:,i])#add column i in num_data

cat_data=pd.DataFrame(cat_data).transpose()#convert cata_data to a dataframe
num_data=pd.DataFrame(num_data).transpose()#convert num_data to a dataframe
#Get an overview
cat_data
```

Out[6]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	Urban	Y
4	LP001008	Male	No	0	Graduate	No	Urban	Y
...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	Rural	Y
610	LP002979	Male	Yes	3+	Graduate	No	Rural	Y
611	LP002983	Male	Yes	1	Graduate	No	Urban	Y
612	LP002984	Male	Yes	2	Graduate	No	Urban	Y
613	LP002990	Female	No	0	Graduate	Yes	Semiurban	N

614 rows × 8 columns

- **For categorical data, Rows with missing values will be replaced with the value which are repeated the most**.

In [7]:
```
cat_data=cat_data.apply(lambda x:x.fillna(x.value_counts().index[0]))
#NaN VALUES REVIEW
cat_data.isnull().sum().any()
```

Out[7]:
```
False
```

- **For numerical data, Rows with missing values will be replaced with the value of the preceding row**.

In [8]:
```
num_data.fillna(method='bfill',inplace=True)
#NaN VALUES REVIEW
num_data.isnull().sum().any()
```

Out[8]:
```
False
```

In [9]:
```
#transoform the target column for prediction. 1 for yes 0 for no
target_value={'Y':1,'N':0}
target=cat_data['Loan_Status']
# drop loan_status colomun from cat_data
cat_data.drop('Loan_Status',axis=1,inplace=True)
target=target.map(target_value)
target
```

Out[9]:
```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int64
```

---

replace categorical values with numerical values using LabelEncoder()

In [10]:
```
le=LabelEncoder()
for i in cat_data:
    cat_data[i]=le.fit_transform(cat_data[i])
cat_data
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area
0	0	1	0	0	0	0	2
1	1	1	1	1	0	0	0
2	2	1	1	0	0	1	2
3	3	1	1	0	1	0	2
4	4	1	0	0	0	0	2
...	...	...	...	...	...	...	...
609	609	0	0	0	0	0	0
610	610	1	1	3	0	0	0
611	611	1	1	1	0	0	2
612	612	1	1	2	0	0	2
613	613	0	0	0	0	1	1

614 rows × 7 columns

- Drop laon_ID column.

In [11]:
```
cat_data.drop('Loan_ID',axis=1,inplace=True)
```

- Vertically concatenating our dataframes.

In [12]:
```
X=pd.concat([cat_data,num_data],axis=1)
y=target
```

In [13]:
```
X
```

Out[13]:

	Gender	Married	Dependents	Education	Self_Employed	Property_Area	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	1	0	0	0	0	2	5849.0	0.0	128.0	360.0	1.0
1	1	1	1	0	0	0	4583.0	1508.0	128.0	360.0	1.0
2	1	1	0	0	1	2	3000.0	0.0	66.0	360.0	1.0
3	1	1	0	1	0	2	2583.0	2358.0	120.0	360.0	1.0
4	1	0	0	0	0	2	6000.0	0.0	141.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
609	0	0	0	0	0	0	2900.0	0.0	71.0	360.0	1.0
610	1	1	3	0	0	0	4106.0	0.0	40.0	180.0	1.0
611	1	1	1	0	0	2	8072.0	240.0	253.0	360.0	1.0
612	1	1	2	0	0	2	7583.0	0.0	187.0	360.0	1.0
613	0	0	0	0	1	1	4583.0	0.0	133.0	360.0	0.0

614 rows × 11 columns

In [14]:
```
y
```

Out[14]:
```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int64
```

Exploratory Data Analysis

Explore Data

- Distribution of variables
- Descriptive statistics

---

after the modifications we will concatenate between the categorical data and the numerical data

In [15]:
```
df=pd.concat([cat_data,num_data,target],axis=1)
```

- We'll start with the target variable.

In [16]:
```
target.value_counts()
```

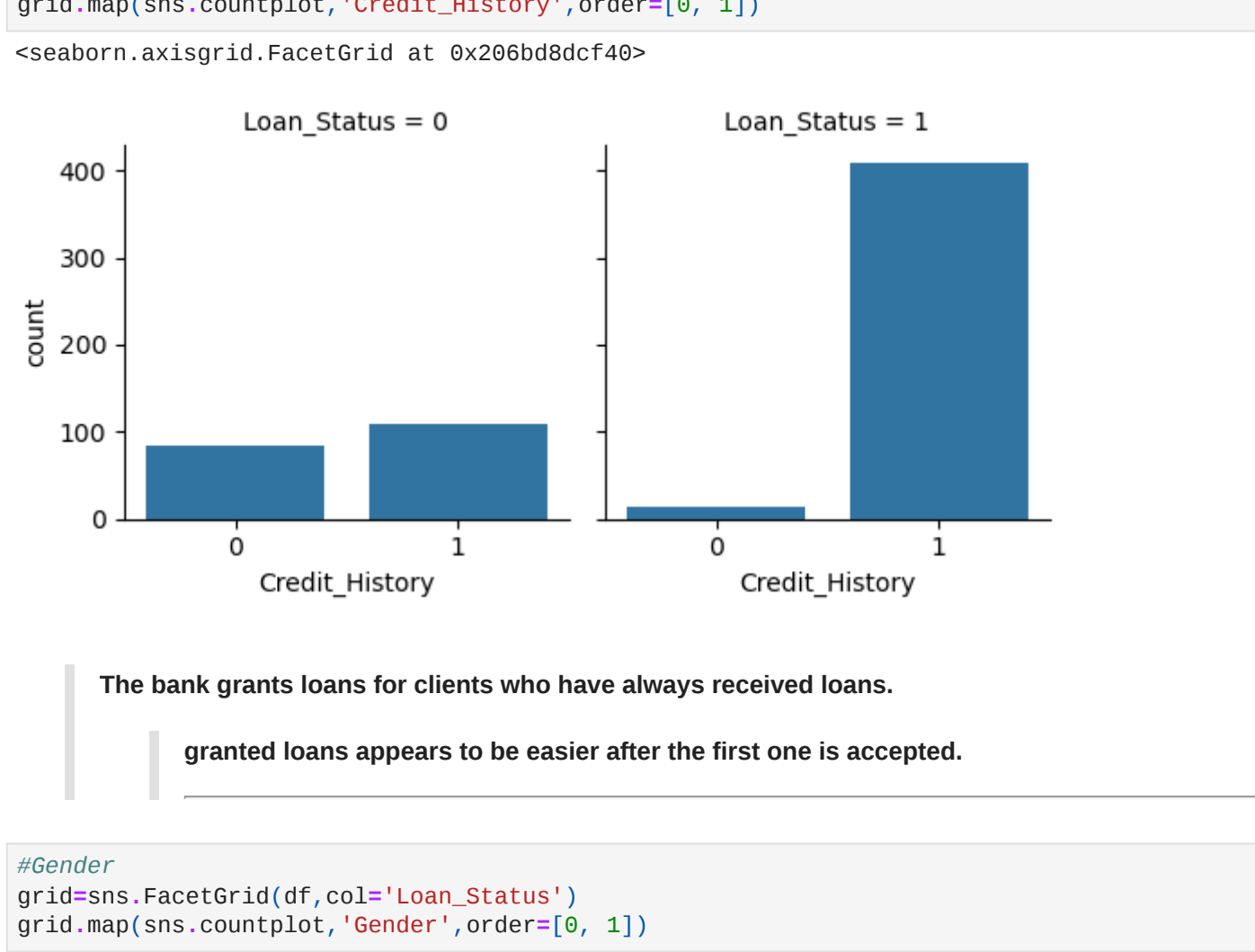
Out[16]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

Look at histograms to get idea of how variables are distrubuted (overall)

In [17]:
```
plt.figure(figsize=(8,5))#SIZE
sns.countplot(x=target,order=[1, 0])
yes=(target.value_counts()[1]/len(target))*100
no=(target.value_counts()[0]/len(target))*100
print(f'percentage of granted loans: {yes}%')
print(f'percentage of rejected loans: {no}%')
plt.title('Target values frequency')
```

```
percentage of granted loans: 68.72964169381108%
percentage of rejected loans: 31.27035830618892%
```

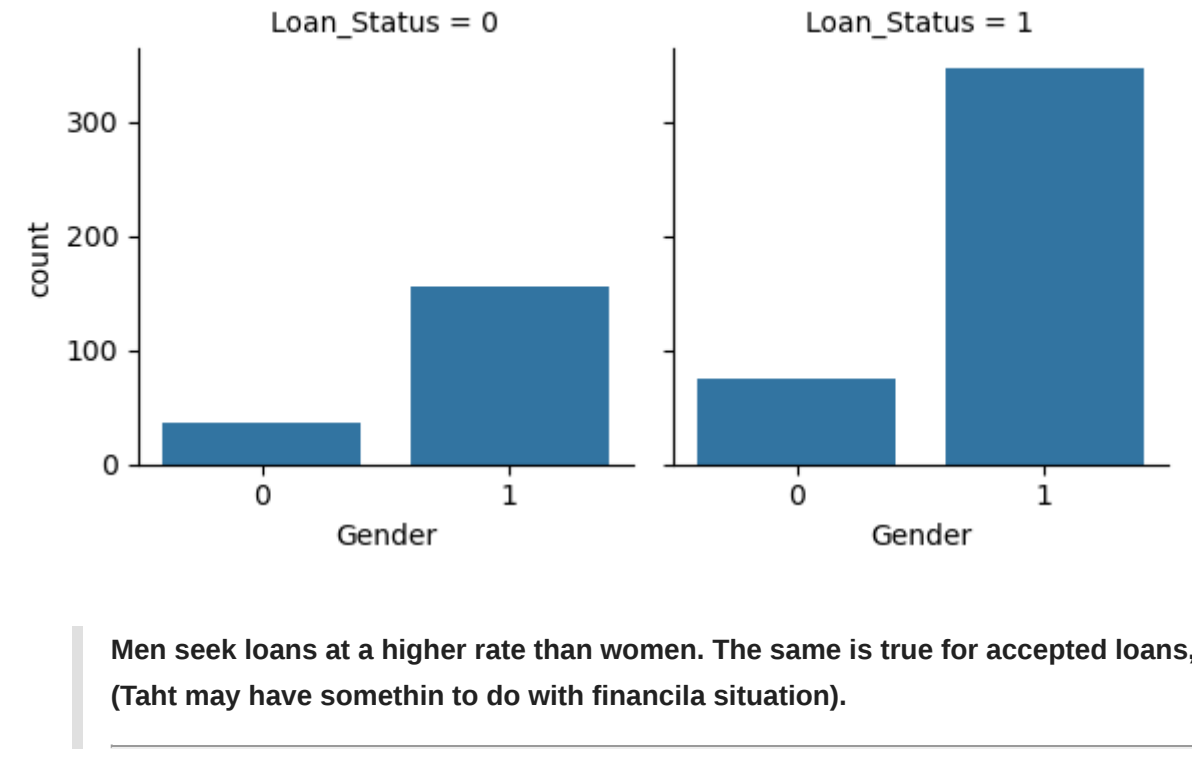
Out[17]:
```
Text(0.5, 1.0, 'Target values frequency')
```



Interpretation of the graph: It appears that the bank approved more than twice as many loans as were requested.

In [23]:
```
#credit history
grid=sns.FacetGrid(df,col='Loan_Status')
grid.map(sns.countplot,'Credit_History',order=[0, 1])
```

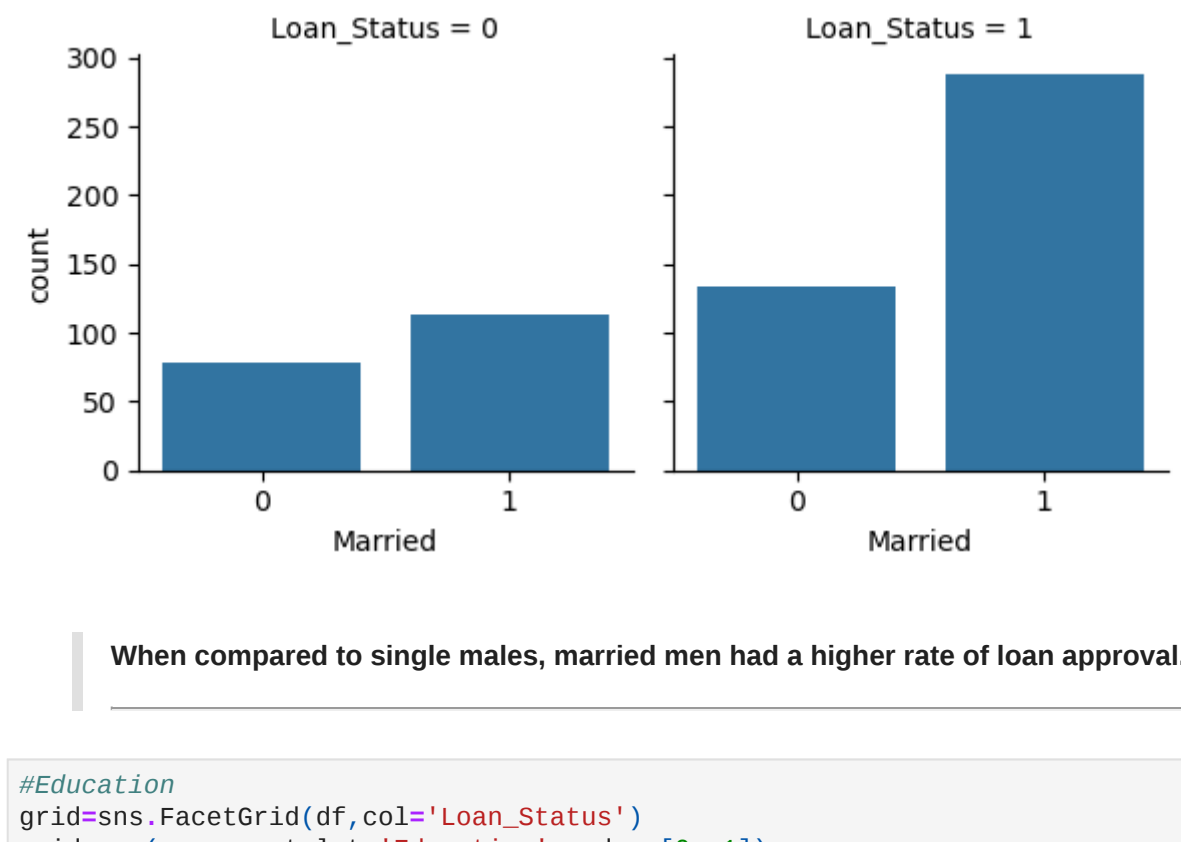
Out[23]:
```
<seaborn.axisgrid.FacetGrid at 0x206bd8dcf40>
```



> **The bank grants loans for clients who have always received loans.**
>
> > **granted loans appears to be easier after the first one is accepted.**

In [19]:
```
#Gender
grid=sns.FacetGrid(df,col='Loan_Status')
grid.map(sns.countplot,'Gender',order=[0, 1])
```

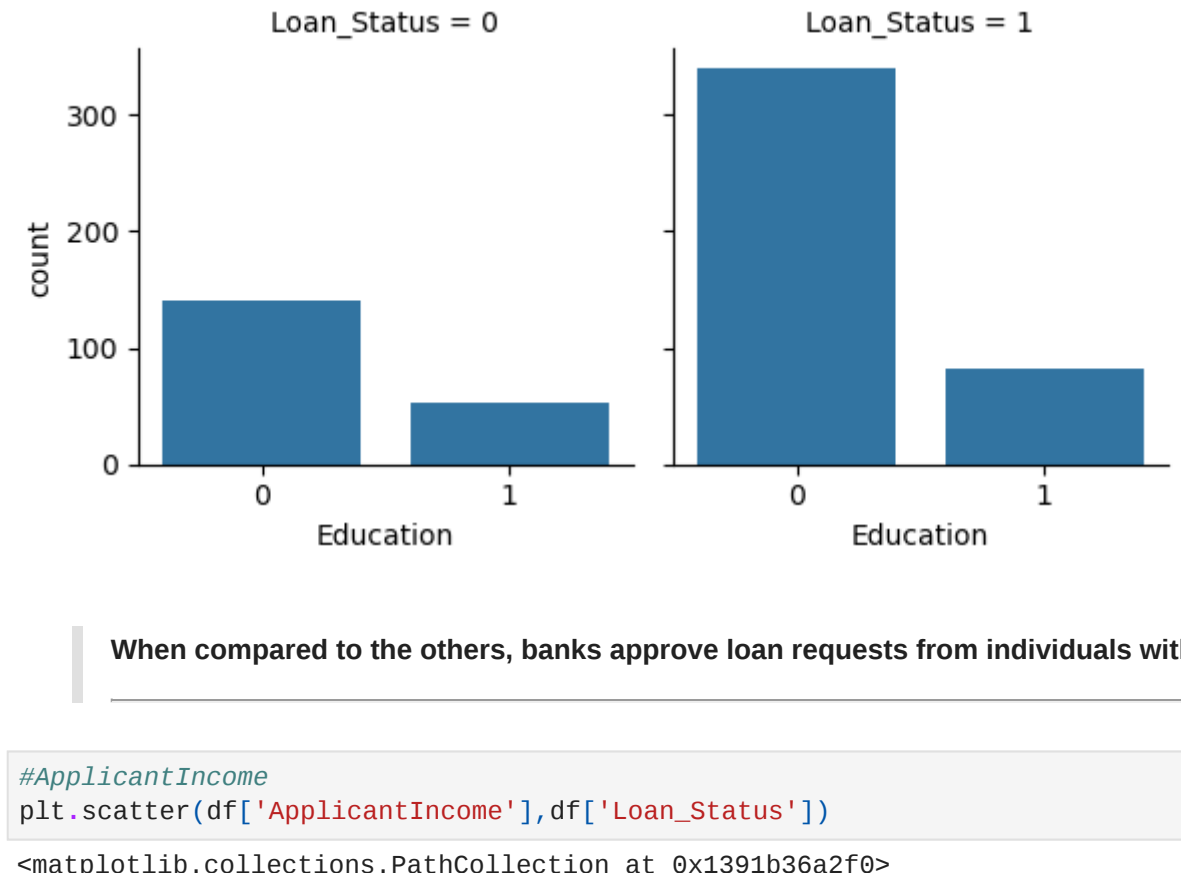
Out[19]:
```
<seaborn.axisgrid.FacetGrid at 0x1391a0c33d0>
```



> **Men seek loans at a higher rate than women. The same is true for accepted loans, since men requests are more likely to be approved by banks than women. (Taht may have somethin to do with financila situation).**

In [20]:
```
#marriage
grid=sns.FacetGrid(df,col='Loan_Status')
grid.map(sns.countplot,'Married',order=[0, 1])
```

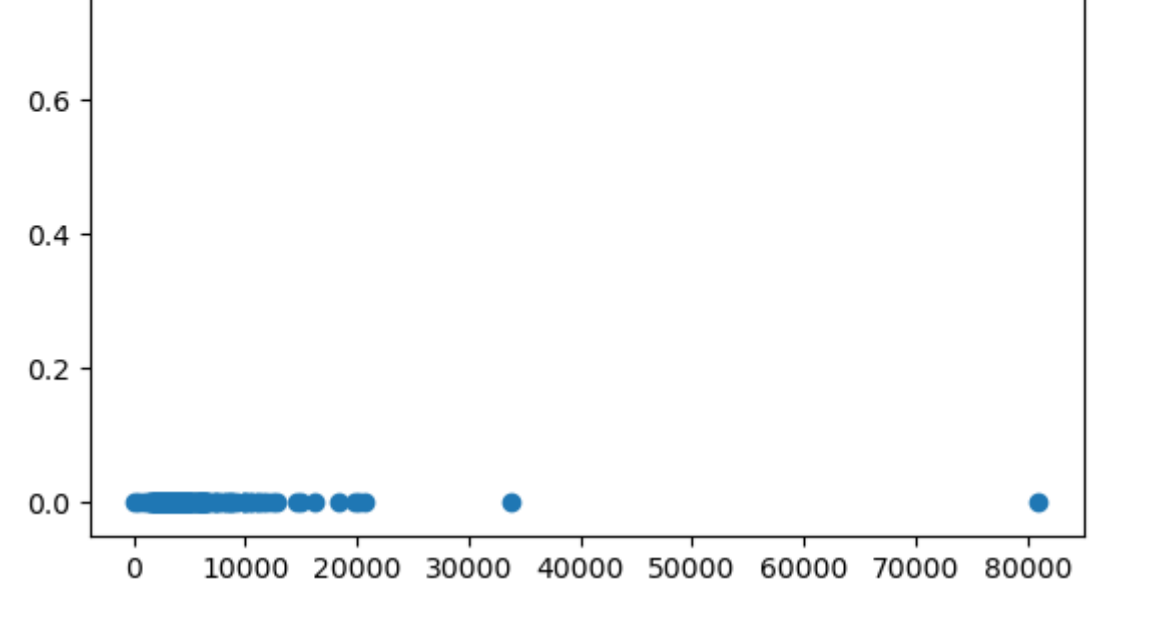
Out[20]:
```
<seaborn.axisgrid.FacetGrid at 0x1391a0c1e40>
```



> **When compared to single males, married men had a higher rate of loan approval.**

In [21]:
```
#Education
grid=sns.FacetGrid(df,col='Loan_Status')
grid.map(sns.countplot,'Education',order=[0, 1])
```

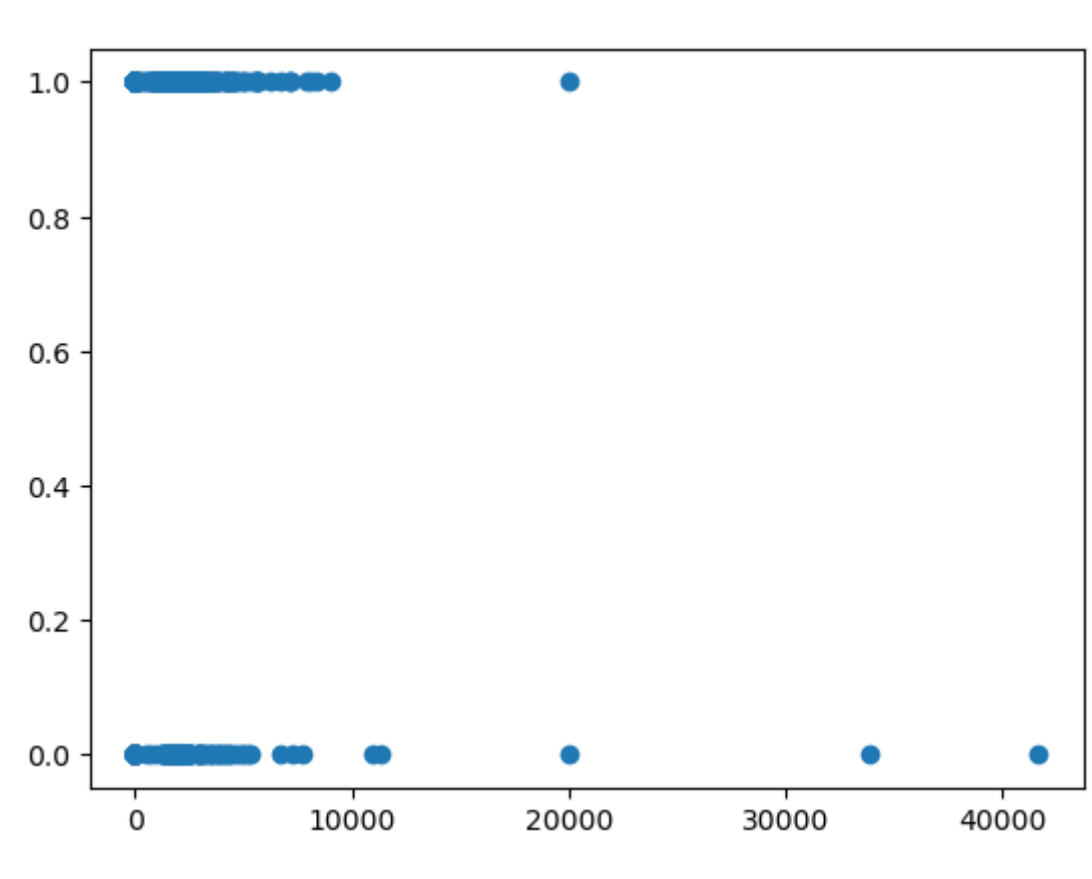
Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x1391b245d80>
```



> **When compared to the others, banks approve loan requests from individuals with an education background.**

In [22]:
```
#ApplicantIncome
plt.scatter(df['ApplicantIncome'],df['Loan_Status'])
```

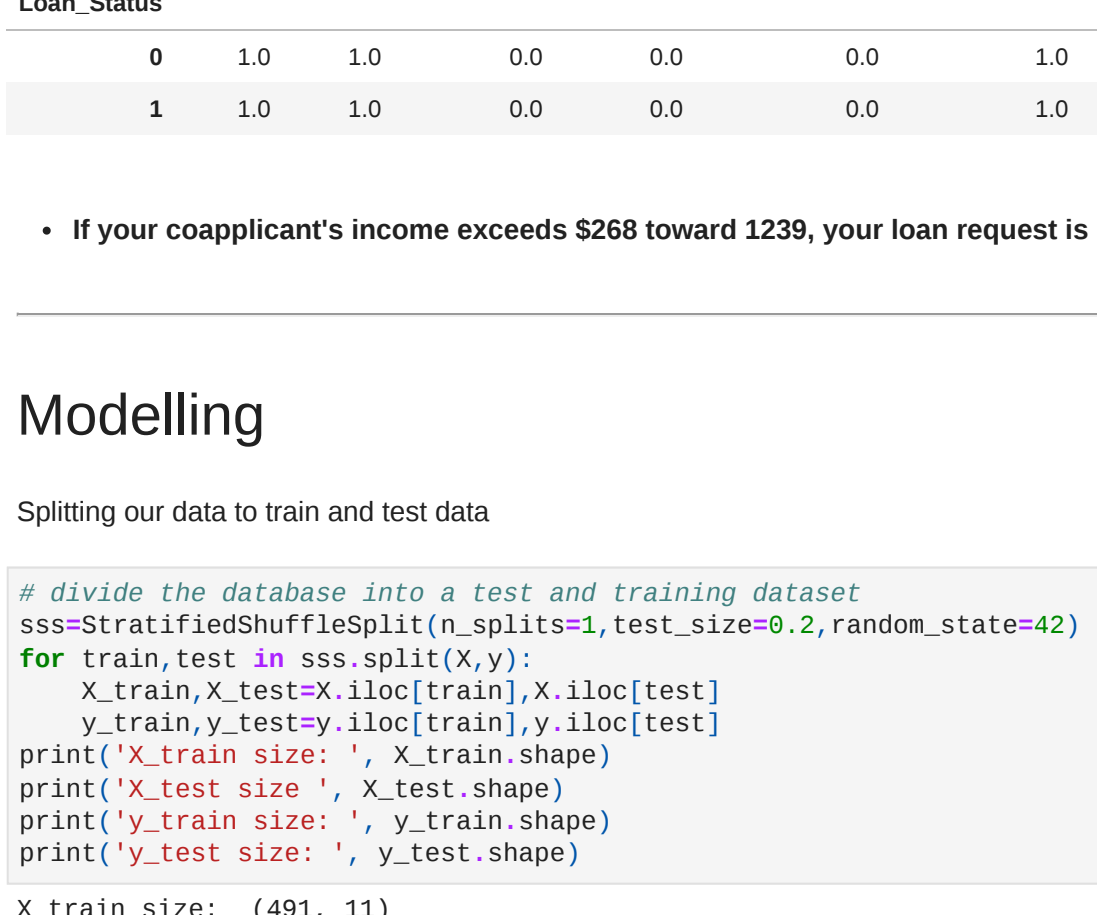
Out[22]:
```
<matplotlib.collections.PathCollection at 0x1391b36a2f0>
```



> **Applicant income has less of an impact on laan approval than we might anticipate.**

In [23]:
```
#revenu du demendeur
plt.scatter(df['CoapplicantIncome'],df['Loan_Status'])
```

Out[23]:
```
<matplotlib.collections.PathCollection at 0x1391b228eb0>
```



> **Coapplicant Income income has less of an impact on laan approval than we might anticipate. The sames goes for the coapplicant income.**

In [24]:
```
df.groupby('Loan_Status').median()
```

Out[24]:

Loan_Status	Gender	Married	Dependents	Education	Self_Employed	Property_Area	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	1.0	1.0	0.0	0.0	0.0	1.0	3833.5	268.0	132.5	360.0	1.0
1	1.0	1.0	0.0	0.0	0.0	1.0	3812.5	1239.5	127.5	360.0	1.0

- **If your coapplicant's income exceeds $268 toward 1239, your loan request is more likely to be granted.**

Modelling

Splitting our data to train and test data

In [18]:
```
# divide the database into a test and training dataset
sss=StratifiedShuffleSplit(n_splits=1,test_size=0.2,random_state=42)
for train,test in sss.split(X,y):
    X_train,X_test=X.iloc[train],X.iloc[test]
    y_train,y_test=y.iloc[train],y.iloc[test]
print('X_train size: ', X_train.shape)
print('X_test size ', X_test.shape)
print('y_train size: ', y_train.shape)
print('y_test size: ', y_test.shape)
```

```
X_train size:  (491, 11)
X_test size  (123, 11)
y_train size:  (491,)
y_test size:  (123,)
```

- modeling using three distinct algorithms **Logistitic Regression**, **Knn** ,**DecisionTree**

In [19]:
```
models={
    'LogisticRegression':LogisticRegression(random_state=42),
    'KNeighborsClassifier':KNeighborsClassifier(),
    'DecisionTreeClassifier':DecisionTreeClassifier(max_depth=1,random_state=42),
}
#The precision function
def accu(y_true,y_pred,retu=False):
    acc=accuracy_score(y_true,y_pred)
    if retu:
        return acc
    else:
        print(f'the precision of the model is:{acc}')
#this is the application function of the model
def train_test_eval(models,X_train,y_train,X_test,y_test):
    for name,model in models.items():
        print(name,':')
        model.fit(X_train,y_train)
        accu(y_test,model.predict(X_test))
        print('-'*30)
train_test_eval(models,X_train,y_train,X_test,y_test)
```

```
LogisticRegression :
the precision of the model is:0.8536585365853658
------------------------------
KNeighborsClassifier :
the precision of the model is:0.6504065040650406
------------------------------
DecisionTreeClassifier :
the precision of the model is:0.8455284552845529
------------------------------
```

```
D:\app\anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

- Logistic regression is the most precise model. **85.3%** .

> In this step, We will maintain the most influential columns on the loan status column.

In [20]:
```
X_2=X[['Credit_History','Married','CoapplicantIncome']]
```

> We'll repeat the splitting process to see if we can improve the precision of our model.

In [21]:
```
# diviser la base de données en une base de données test et d'entrainement
sss=StratifiedShuffleSplit(n_splits=1,test_size=0.2,random_state=42)
for train,test in sss.split(X_2,y):
    X_train,X_test=X_2.iloc[train],X_2.iloc[test]
    y_train,y_test=y.iloc[train],y.iloc[test]
print('X_train size: ', X_train.shape)
print('X_test size: ', X_test.shape)
print('y_train size: ', y_train.shape)
print('y_test size: ', y_test.shape)
```

```
X_train size:  (491, 3)
X_test size:  (123, 3)
y_train size:  (491,)
y_test size:  (123,)
```

In [23]:
```
train_test_eval(models,X_train,y_train,X_test,y_test)
```

```
LogisticRegression :
the precision of the model is:0.8536585365853658
------------------------------
KNeighborsClassifier :
the precision of the model is:0.6991869918699187
------------------------------
DecisionTreeClassifier :
the precision of the model is:0.8455284552845529
------------------------------
```

> Even after trying to optimize our model, we still have the same precision.

- **fitting data to our Logitistic regression model.**

In [22]:
```
Classifier=LogisticRegression()
Classifier.fit(X_2,y)
```

Out[22]:
```
▾ LogisticRegression
LogisticRegression()
```

- **Save model**

In [23]:
```
pickle.dump(Classifier,open('model.pkl','wb'))
```

Conclusions

This project has four major parts :

1. model.py - This contains code fot our Machine Learning model to predict customer loan status absed on trainign data in 'customer.csv' file.
2. app.py - This contains Flask APIs that receives customer details through GUI or API calls, computes the precited value based on our model and returns it.
3. request.py - This uses requests module to call APIs already defined in app.py and dispalys the returned value.
4. templates - This folder contains the HTML template to allow user to enter customer detail and displays the predicted loan status.

In [ ]: