Loan Application Status Prediction

Problem Statement:

This dataset includes details of applicants who have applied for loan. The dataset includes details like credit history, loan amount, their income, dependents etc.

Independent Variables:

- Loan_ID
- Gender
- Married
- Dependents
- Education
- Self_Employed
- ApplicantIncome
- CoapplicantIncome
- Loan_Amount
- Loan_Amount_Term
- Credit History
- Property_Area

Dependent Variable (Target Variable):

- Loan_Status

In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
loan_df=pd.read_csv('Loan_Application_Status_Prediction.csv')
loan_df.head(10)
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
6	LP001013	Male	Yes	0	Not Graduate	No	2333	
7	LP001014	Male	Yes	3+	Graduate	No	3036	
8	LP001018	Male	Yes	2	Graduate	No	4006	
9	LP001020	Male	Yes	1	Graduate	No	12841	

In [3]:
```
#number of rows and columns
loan_df.shape
```

Out[3]:
```
(614, 13)
```

In [4]:
```
#statical mmeasures
loan_df.describe()
```

Out[4]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [5]:
```
#number of missing values in each columns
loan_df.isnull().sum()
```

Out[5]:
```
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 [6]:
```
#dropping the missing values
loan_df=loan_df.dropna()
```

In [7]:
```
#Again check number of missing values in each column
loan_df.isnull().sum()
```

Out[7]:
```
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
```

Here no missing values found

In [8]:
```
# label encoding
loan_df.replace({"Loan_Status":{'N':0,'Y':1}},inplace=True)
```

In [9]:
```
loan_df.head()
```

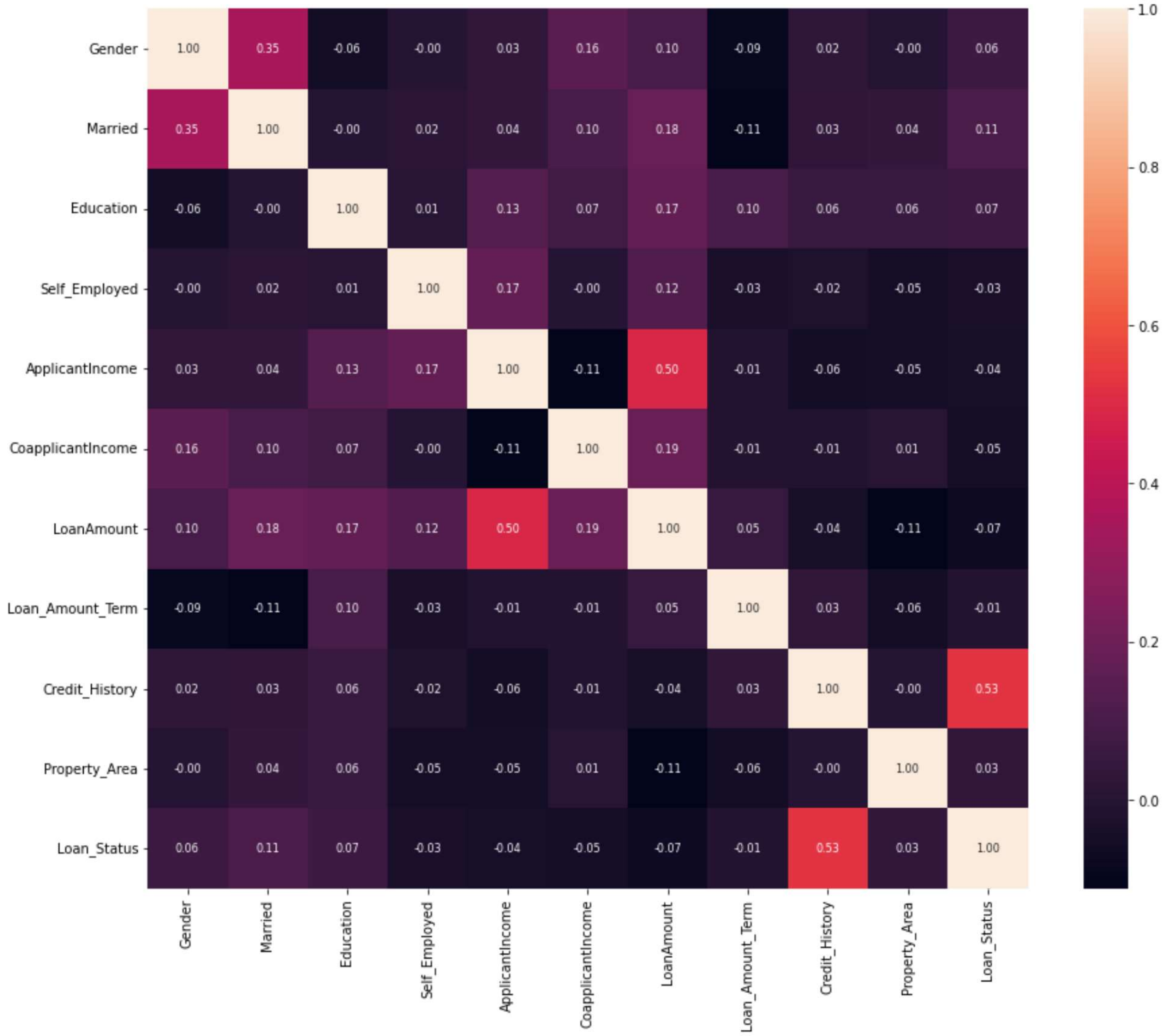
Out[9]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	

Correlation

In [33]:
```
plt.figure(figsize=(15,12))
sns.heatmap(loan_df.corr(),cbar=True,square=True,fmt='.2f',annot=True,annot_kws={'size
```

Out[33]: <AxesSubplot:>



In [10]:
```
#Dependent columns values
loan_df['Dependents'].value_counts()
```

Out[10]:
```
0     274
2      85
1      80
3+     41
Name: Dependents, dtype: int64
```

In [11]:
```
#replacing the value of 3+ to 4
loan_df=loan_df.replace(to_replace='3+',value=4)
```

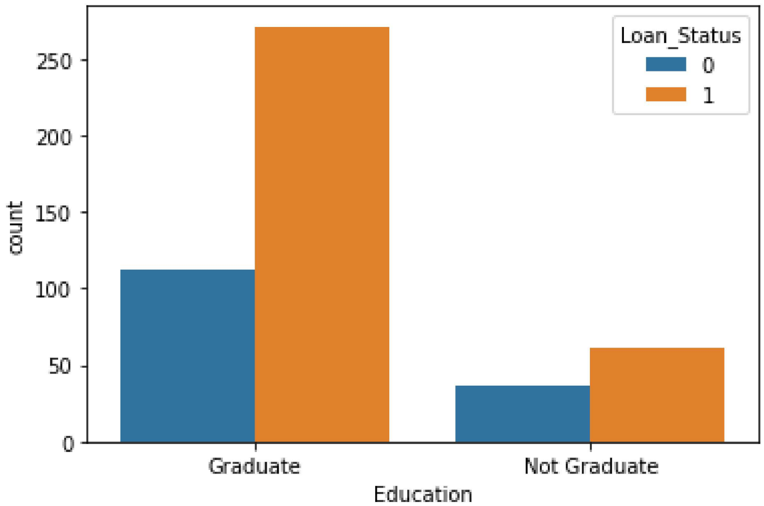
In [12]:
```
#Dependent values
loan_df['Dependents'].value_counts()
```

Out[12]:
```
0    274
2     85
1     80
4     41
Name: Dependents, dtype: int64
```

Data Visualization

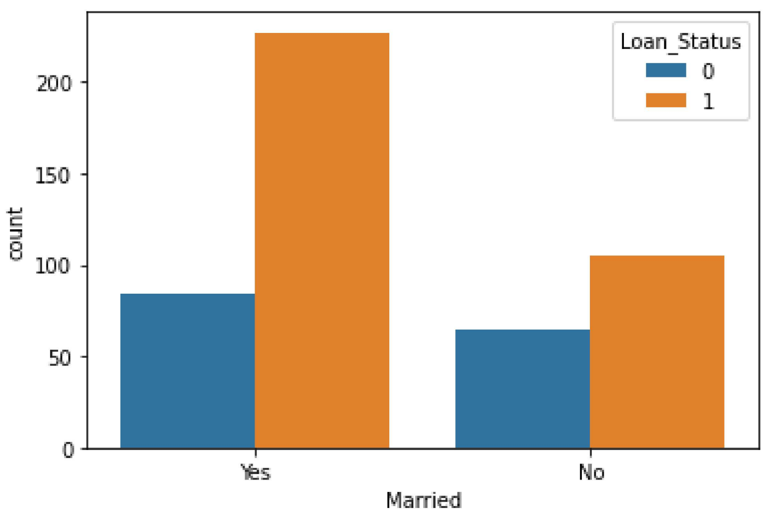
In [13]:
```
# Education & Loan Status
sns.countplot(x='Education',hue='Loan_Status',data=loan_df)
```

Out[13]: <AxesSubplot:xlabel='Education', ylabel='count'>



In [14]:
```
#Marital status & Loan Status
sns.countplot(x='Married',hue='Loan_Status',data=loan_df)
```

Out[14]: <AxesSubplot:xlabel='Married', ylabel='count'>



In [15]:
```
# convert categorical columns to numerical values
loan_df.replace({'Married':{'No':0,'Yes':1},'Gender':{'Male':1,'Female':0},'Self_Emplo
                      'Property_Area':{'Rural':0,'Semiurban':1,'Urban':2},'Education':
```

In [16]:
```
loan_df.head()
```

Out[16]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
1	LP001003	1	1	1	1	0	4583	
2	LP001005	1	1	0	1	1	3000	
3	LP001006	1	1	0	0	0	2583	
4	LP001008	1	0	0	1	0	6000	
5	LP001011	1	1	2	1	1	5417	

In [17]:
```
# separating the data and label
x = loan_df.drop(columns=['Loan_ID','Loan_Status'],axis=1)
y = loan_df['Loan_Status']
```

In [18]:
```
print(x.shape)
print(y.shape)
```

```
(480, 11)
(480,)
```

Train Test Split

In [19]:
```
x_train, x_test,y_train,y_test = train_test_split(x,y,test_size=0.1,random_state=2)
```

In [20]:
```
print(x.shape, x_train.shape, x_test.shape)
```

```
(480, 11) (432, 11) (48, 11)
```

Training the model:

Support Vector Machine Model

In [21]:
```
classifier = svm.SVC(kernel='linear')
#training the support Vector Macine model
classifier.fit(x_train,y_train)
```

Out[21]:

```
▾ SVC
SVC(kernel='linear')
```

Model Evaluation

In [22]:
```
# accuracy score on training data
x_train_prediction = classifier.predict(x_train)
training_data_accuray = accuracy_score(x_train_prediction,y_train)

print('Accuracy on training data : ', training_data_accuray)
```

```
Accuracy on training data :  0.7962962962962963
```

In [23]:
```
# accuracy score on training data
x_test_prediction = classifier.predict(x_test)
```

```
test_data_accuray = accuracy_score(x_test_prediction,y_test)

print('Accuracy on test data : ', test_data_accuray)
```

```
Accuracy on test data :  0.8125
```

Training Machine Learning Models

In [24]:
```
#Create a fuction within many machine learning Models
def models(x_train,y_train):

    #using Logistic Regression Algorithm to the Training set
    from sklearn.linear_model import LogisticRegression
    lr=LogisticRegression()
    lr.fit(x_train,y_train)

    #using KNeighborsClassifier Method of neighbors class to use Nearest Neighbor algo
    from sklearn.neighbors import KNeighborsClassifier
    knn=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
    knn.fit(x_train,y_train)

    #using SVC method of svm class use Support Vector Machine Algorithm
    from sklearn.svm import SVC
    svc=SVC(kernel='linear',random_state=0)
    svc.fit(x_train,y_train)

    #using GaussinNB method of navie_bayes class to use Naive Bayes Algorithm
    from sklearn.naive_bayes import GaussianNB
    gnb=GaussianNB()
    gnb.fit(x_train,y_train)

    #using DecisionTreeClassifier of tree class to use Decision Tree Classifier algori
    from sklearn.tree import DecisionTreeClassifier
    dtc=DecisionTreeClassifier()
    dtc.fit(x_train,y_train)

    #using RandomForestClassifier method of ensemble class to use Random Forest Classi
    from sklearn.ensemble import RandomForestClassifier
    rfc=RandomForestClassifier(n_estimators=11,criterion='entropy',random_state=0)
    rfc.fit(x_train,y_train)

    #print model accuracy on the training data.
    print('[0] Logistic Regression Training Accuracy : ',lr.score(x_train,y_train))
    print('[1] K Nearest Neighbor Training Accuracy : ',knn.score(x_train,y_train))
    print('[2] Support Vector Machine(Linear Classifier) Training Accuracy : ',svc.sco
    print('[3] Gussian Naive Bayes Training Accuracy : ',gnb.score(x_train,y_train))
    print('[4] Decision Tree Classifier Training Accuracy : ',dtc.score(x_train,y_trai
    print('[5] Random Forest Classifier Training Accuracy : ',rfc.score(x_train,y_trai

    return lr,knn,svc,gnb,dtc,rfc
```

Evaluating Performance on Training Sets

In [25]:
```
#Get and train all of the models
model=models(x_train,y_train)
```

```
[0] Logistic Regression Training Accuracy :  0.7986111111111112
[1] K Nearest Neighbor Training Accuracy :  0.7384259259259259
[2] Support Vector Machine(Linear Classifier) Training Accuracy :  0.7962962962962963
[3] Gussian Naive Bayes Training Accuracy :  0.7916666666666666
[4] Decision Tree Classifier Training Accuracy :  1.0
[5] Random Forest Classifier Training Accuracy :  0.9837962962962963
```

Evaluating Performance on Testing Sets

In [26]:

```
from sklearn.metrics import confusion_matrix
for i in range(len(model)):
    cm=confusion_matrix(y_test, model[i].predict(x_test))
    #extracting TN,FP,FN,TP
    TN,FP,FN,TP = confusion_matrix(y_test, model[i].predict(x_test)).ravel()
    print(cm)
    print('model[{}] Testing Accuracy="{} !"'.format(i,(TP + TN) / (TP + TN + FN + FP)
    print() #Print a new line
```

```
[[ 7  8]
 [ 0 33]]
model[0] Testing Accuracy="0.8333333333333334 !"

[[ 5 10]
 [ 4 29]]
model[1] Testing Accuracy="0.7083333333333334 !"

[[ 6  9]
 [ 0 33]]
model[2] Testing Accuracy="0.8125 !"

[[ 7  8]
 [ 1 32]]
model[3] Testing Accuracy="0.8125 !"

[[ 8  7]
 [ 5 28]]
model[4] Testing Accuracy="0.75 !"

[[ 8  7]
 [ 3 30]]
model[5] Testing Accuracy="0.7916666666666666 !"
```

Saving the model

In [27]:

```
import pickle
filename='Red Wine Quality Prediction.pkl'
pickle.dump(classifier, open(filename,'wb'))
```

In [29]:

```
a=np.array(y_test)
predicted=np.array(classifier.predict(x_test))
loan_df.com=pd.DataFrame({"original":a,"predicted":predicted},index=range(len(a)))
loan_df.com
```

Out[29]:

	original	predicted
0	1	1
1	0	0
2	1	1
3	1	1
4	1	1
5	0	1
6	0	0
7	1	1
8	1	1
9	0	1
10	1	1
11	1	1
12	1	1
13	1	1
14	0	0
15	1	1
16	1	1
17	1	1
18	1	1
19	0	0
20	1	1
21	0	0
22	1	1
23	1	1
24	1	1
25	1	1
26	0	1
27	1	1
28	1	1
29	0	1
30	1	1
31	1	1
32	0	1

	original	predicted
33	1	1
34	1	1
35	0	1
36	1	1
37	1	1
38	0	1
39	1	1
40	1	1
41	0	1
42	1	1
43	1	1
44	0	1
45	1	1
46	1	1
47	0	0

```
In [ ]:
```