Lab : 7 : Loan Approval Classification using SVM

Name : Murali kumar R

Roll no : 225229120

Step 1 : [Understand Data]

In [35]:
```
import pandas as pd
```

In [36]:
```
loan=pd.read_csv('train_loan.csv')
loan.head()
```

Out[36]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
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	

In [37]:
```
loan.shape
```

Out[37]:
```
(614, 13)
```

In [38]:
```
loan.columns
```

Out[38]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [39]:
```
type(loan)
```

Out[39]:
```
pandas.core.frame.DataFrame
```

In [40]:
```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               601 non-null object
Married              611 non-null object
Dependents           599 non-null object
Education            614 non-null object
Self_Employed        582 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           592 non-null float64
Loan_Amount_Term     600 non-null float64
Credit_History       564 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

In [41]:
```
loan.count()
```

Out[41]:
```
Loan_ID              614
Gender               601
Married              611
Dependents           599
Education            614
Self_Employed        582
ApplicantIncome      614
CoapplicantIncome    614
LoanAmount           592
Loan_Amount_Term     600
Credit_History       564
Property_Area        614
Loan_Status          614
dtype: int64
```

Step 2 : [Data Cleaning]

In [42]:
```
# Replace numbers as string by integer
```

In [43]:
```
def string(x):
    if x == '0':
        return 'bad'
    elif x == '1':
        return 'average'
    elif x == '2':
        return 'good'
    else:
        return 'excellent'
```

In [58]:
```
loan['Dependents'] = loan['Dependents'].apply(string)
```

In [45]:
```
loan.head()
```

Out[45]:

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

In [46]:
```
# Missing Categorical Columns
```

In [61]:
```
cat_cols=['Gender','Married','Dependents','Education','Credit_History']
cont_cols=['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term',

loan[cat_cols]=loan[cat_cols].fillna(loan.mode().iloc[0])
loan[cont_cols]=loan[cont_cols].fillna(loan.median().iloc[0])
```

In [63]:
```
loan['LoanAmount'].fillna(loan['LoanAmount'].mean(), inplace=True)
```

In [55]:
```
data=loan.drop(['Loan_ID'],axis=1)
data.head()
```

Out[55]:

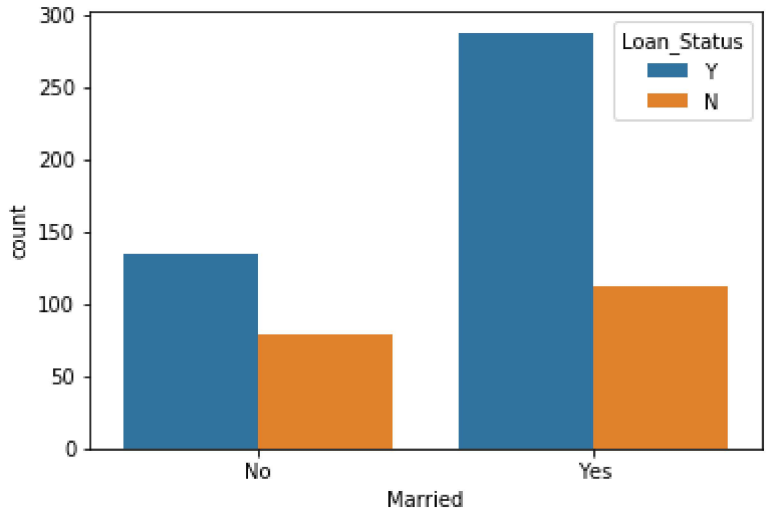
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	
0	Male	No	bad	Graduate	No	5849	0.0	
1	Male	Yes	average	Graduate	No	4583	1508.0	
2	Male	Yes	bad	Graduate	Yes	3000	0.0	
3	Male	Yes	bad	Not Graduate	No	2583	2358.0	
4	Male	No	bad	Graduate	No	6000	0.0	

Step 3 : [OPTIONAL : Exploratory Data Analysis - Who got their loan approved]

In [66]:
```
import seaborn as sns
```

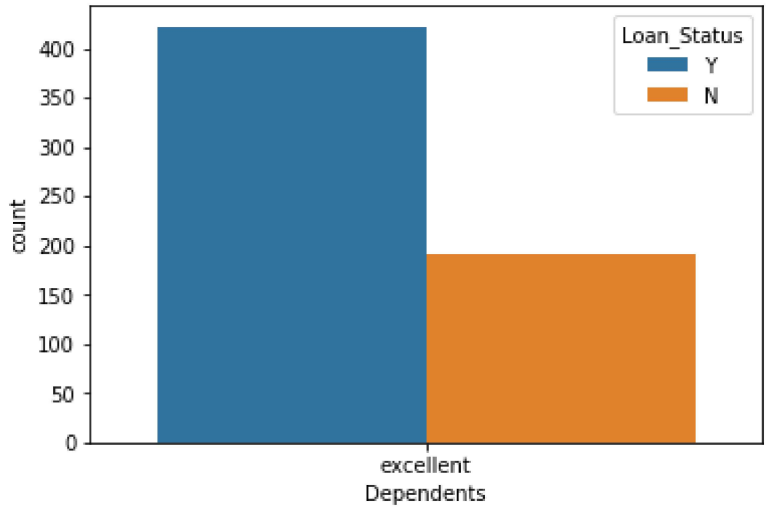
In [73]:
```
sns.countplot(x='Married',hue='Loan_Status',data=loan)
print
```

Out[73]: <function print>



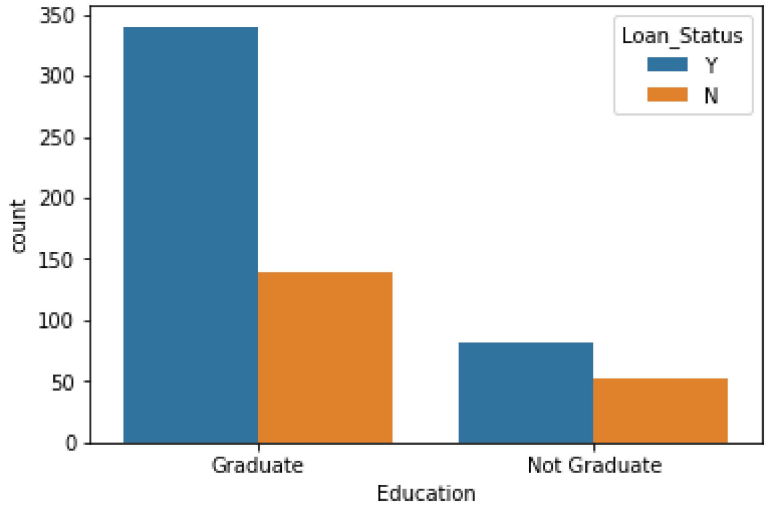
In [74]:
```
sns.countplot(x='Dependents',hue='Loan_Status',data=loan)
print
```

Out[74]: <function print>



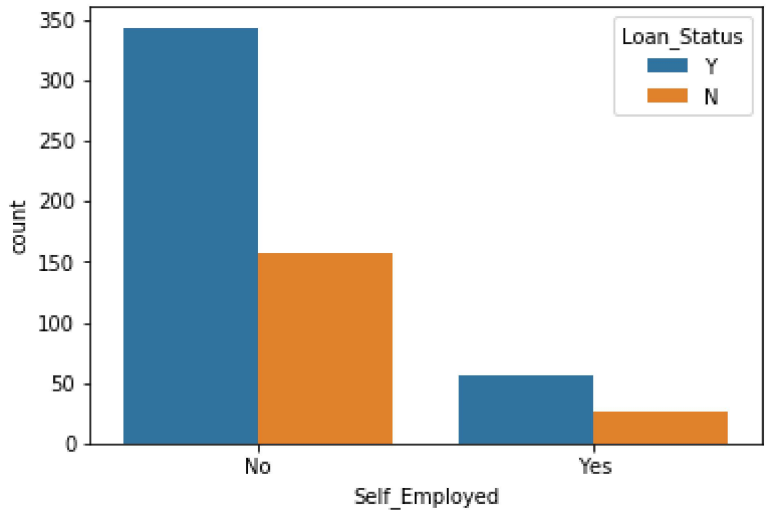
In [77]:
```
sns.countplot(x='Education',hue='Loan_Status',data=loan)
print
```

Out[77]: <function print>



In [79]:
```
sns.countplot(x='Self_Employed',hue='Loan_Status',data=loan)
print
```

Out[79]: <function print>



Step 4 : [Extract X and y]

In [81]:
```
X = loan.drop(['Loan_Status'],axis=1)
y = loan.Loan_Status
```

In [82]:
```
X.head()
```

Out[82]:

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

In [83]:
```
y.head()
```

Out[83]:
```
0    Y
1    N
2    Y
3    Y
4    Y
Name: Loan_Status, dtype: object
```

Step 5 : [One Hot Encoding]

In [84]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [87]:
```
X=pd.get_dummies(X)
X.head()
```

Out[87]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_ID_
0	5849	0.0	3812.5	360.0	1.0	
1	4583	1508.0	128.0	360.0	1.0	
2	3000	0.0	66.0	360.0	1.0	
3	2583	2358.0	120.0	360.0	1.0	
4	6000	0.0	141.0	360.0	1.0	

5 rows × 631 columns

Step 6 : [Model Building]

In [88]:
```
# Spilt X and y for Training and testing
```

In [89]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_stat
```

In [90]:
```
# Using StandardScaler, fit_transform:
```

In [92]:
```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
scale
```

Out[92]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [99]:
```
ss=scale.fit_transform(X_train)
ss1=scale.transform(X_test)

ss
ss1
```

Out[99]:
```
array([[ 0.21857952, -0.54770577, -0.29127367, ..., -0.63761852,
         1.29663025, -0.71453718],
       [-0.20156346, -0.54770577, -0.25238992, ..., -0.63761852,
         1.29663025, -0.71453718],
       [ 0.07848281,  0.79027582,  0.04201562, ..., -0.63761852,
        -0.77122989,  1.3995073 ],
       ...,
       [-0.35814214, -0.54770577, -0.35793153, ..., -0.63761852,
         1.29663025, -0.71453718],
       [-0.34578066, -0.54770577, -0.17184501, ...,  1.56833588,
        -0.77122989, -0.71453718],
       [-0.10370178, -0.54770577, -0.26627697, ..., -0.63761852,
         1.29663025, -0.71453718]])
```

In [93]:
```
# Linear SVC model:
```

In [103]:
```
from sklearn.svm import LinearSVC
model = LinearSVC()
model.fit(ss,y_train)
```

Out[103]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

In [104]:
```
Lsvc_y_pred = model.predict(ss1)
Lsvc_y_pred
```

Out[104]:
```
array(['Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y'], dtype=object)
```

In [102]:
```
# Print Accuracy value:
```

In [105]:
```
from sklearn.metrics import accuracy_score,confusion_matrix, classification_repor
accuracy_score(y_test,Lsvc_y_pred)
```

Out[105]:
```
0.8324324324324325
```

In [106]:
```
# Print Confusion matrix:
```

In [109]:
```
confusion_matrix(y_test,Lsvc_y_pred)
```

Out[109]:
```
array([[ 22,  29],
       [  2, 132]], dtype=int64)
```

In [107]:
```
#print Classification_report:
```

In [108]:
```
print(classification_report(y_test,Lsvc_y_pred))
```

```
             precision    recall  f1-score   support

          N       0.92      0.43      0.59        51
          Y       0.82      0.99      0.89       134

avg / total       0.85      0.83      0.81       185
```

Step 7 : [Performance Comparisons]

In [110]:
```
# Compare the performance of LinearSVC against LogisticRegression:
```

In [111]:
```
from sklearn.linear_model import LogisticRegression
lgr= LogisticRegression()
lgr.fit(ss,y_train)
lr_y_pred = lgr.predict(ss1)
from sklearn.svm import LinearSVC
l_svc = LinearSVC()
l_svc.fit(ss,y_train)
lsvc_y_pred = l_svc.predict(ss1)
print("LogisticRegression:",accuracy_score(y_test,lr_y_pred))
print("LinearSVC :",accuracy_score(y_test,lsvc_y_pred))
```

```
LogisticRegression: 0.8324324324324325
LinearSVC : 0.8324324324324325
```

In [112]:
```
# Compare the performance of LinearSVC against SGDClassifier:
```

In [113]:
```
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd.fit(ss,y_train)
sgdc_y_pred = sgd.predict(ss1)
from sklearn.svm import LinearSVC
l_svc = LinearSVC()
l_svc.fit(ss,y_train)
lsvc_y_pred = l_svc.predict(ss1)
print("SGDClassifier:", accuracy_score(y_test,sgdc_y_pred))
print("LinearSVC :",accuracy_score(y_test,lsvc_y_pred))
```

```
SGDClassifier: 0.8324324324324325
LinearSVC : 0.8324324324324325
```

In [119]:
```

```

In [124]:
```
from sklearn.svm import SVC

l_svc = LinearSVC()
l_svc.fit(ss,y_train)
lsvc_y_pred = l_svc.predict(ss1)

poly_svc = SVC(kernel='poly', C = 1.0)
poly_svc.fit(ss,y_train)
psvc_y_pred=poly_svc.predict(ss1)

rbf_svc = SVC(kernel='rbf', C = 1.0)
rbf_svc.fit(ss,y_train)
rbfsvc_y_pred=rbf_svc.predict(ss1)

sigmoid_svc = SVC(kernel='sigmoid', C = 1.0)
sigmoid_svc.fit(ss,y_train)
sigsvc_y_pred=sigmoid_svc.predict(ss1)

print("LinearSVC :",accuracy_score(y_test,lsvc_y_pred))
print("poly SVC :",accuracy_score(y_test,psvc_y_pred))
print("rbf SVC :",accuracy_score(y_test,rbfsvc_y_pred))
print("Sigmoid SVC :",accuracy_score(y_test,sigsvc_y_pred))
```

```
LinearSVC : 0.8324324324324325
poly SVC : 0.7243243243243244
rbf SVC : 0.7243243243243244
Sigmoid SVC : 0.7621621621621621
```

In [125]:
```
# Interpret the results
```

In [127]:
```
import numpy as np
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import auc
MLA = [model,lgr,sgd,poly_svc,rbf_svc,sigmoid_svc]
MLA_columns = []
MLA_compare = pd.DataFrame(columns = MLA_columns)
row_index = 0
for alg in MLA:
    predicted = alg.fit(ss, y_train).predict(ss1)
    predicted=np.where(predicted=='Y',1,0)
    y_testb=np.where(y_test=='Y',1,0)
    fp, tp, th = roc_curve(y_testb, predicted)
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index,'MLA used'] = MLA_name
    MLA_compare.loc[row_index, 'Train Accuracy'] = round(alg.score(ss,y_train), 4
    MLA_compare.loc[row_index, 'Test Accuracy'] = round(alg.score(ss1,y_test), 4)
    MLA_compare.loc[row_index, 'Precission'] = precision_score(y_testb, predicted
    MLA_compare.loc[row_index, 'Recall'] = recall_score(y_testb, predicted)
    MLA_compare.loc[row_index, 'AUC'] = auc(fp, tp)
row_index+=1
MLA_compare
```

Out[127]:

	MLA used	Train Accuracy	Test Accuracy	Precission	Recall	AUC
0	SVC	0.8415	0.7622	0.755682	0.992537	0.5747

In [ ]: