Business Problem

Dream housing finance company deals in all kinds of home loans they have presence across all urban semi urban and rural areas custom of first applies for home loan and after that company validates the customer eligibility for loan. company wants to automate the loan eligibility process real time based on customer detail provided while feeling online application form this details education ,number of dependents, income, loan amount, credit history, and others to automatically process they have provided data set to identify the customer segments that available for loan amount so that can specifically target this customers

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')
```

In [3]:
```
df= pd.read_csv('Loan_Data.csv')
df.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Applicant
0	LP001002	Male	No	0	Graduate	No	
1	LP001003	Male	Yes	1	Graduate	No	
2	LP001005	Male	Yes	0	Graduate	Yes	
3	LP001006	Male	Yes	0	Not Graduate	No	
4	LP001008	Male	No	0	Graduate	No	

In [4]:
```
df.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
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
```

from above info we observed that 'dependents' and 'loan_amount_term' column is labelled as object as it is count_variable ------------> hence we need to change

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [8]:
```
df['Loan_ID'].nunique()
```

Out[8]:
```
614
```

There are unique loan id's hence theres no need of this variable

In [9]:
```
#Drop this column
df.drop(columns = ['Loan_ID'] , inplace = True)
```

In [10]:
```
df['Gender'].unique()
```

Out[10]:
```
array(['Male', 'Female', nan], dtype=object)
```

In [11]:
```
df['Gender'].value_counts()
```

Out[11]:
```
Gender
Male      489
Female    112
Name: count, dtype: int64
```

In [12]:
```
df['Married'].unique()
```

Out[12]: array(['No', 'Yes', nan], dtype=object)

In [13]:
```
df['Married'].value_counts()
```

Out[13]:
```
Married
Yes    398
No     213
Name: count, dtype: int64
```

In [14]:
```
df['Dependents'].unique()
```

Out[14]: array(['0', '1', '2', '3+', nan], dtype=object)

In [15]:
```
df['Dependents'].value_counts()
```

Out[15]:
```
Dependents
0     345
1     102
2     101
3+     51
Name: count, dtype: int64
```

In [16]:
```
df['Education'].unique()
```

Out[16]: array(['Graduate', 'Not Graduate'], dtype=object)

In [17]:
```
df['Education'].value_counts()
```

Out[17]:
```
Education
Graduate        480
Not Graduate    134
Name: count, dtype: int64
```

In [18]:
```
df['Self_Employed'].unique()
```

Out[18]: array(['No', 'Yes', nan], dtype=object)

In [19]:
```
df['Self_Employed'].value_counts()
```

Out[19]:
```
Self_Employed
No     500
Yes     82
Name: count, dtype: int64
```

In [20]:
```
# As per business problem we merge columnsof applicants income along with co-a
df['Income'] = df['ApplicantIncome']+df['CoapplicantIncome']
```

In [21]:
```
# as it was created as a seperate column we delected the combined two columns
df.drop(columns=['ApplicantIncome', 'CoapplicantIncome'] , inplace = True)
```

In [22]:
```
df['Loan_Amount_Term'].unique()
```

Out[22]: array([360., 120., 240., nan, 180., 60., 300., 480., 36., 84., 12.])

In [23]:
```
df['Loan_Amount_Term'].value_counts()
```

Out[23]:
```
Loan_Amount_Term
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: count, dtype: int64
```

In [24]:
```
df['Credit_History'].unique()
```

Out[24]: array([1., 0., nan])

In [25]:
```
df['Credit_History'].value_counts()
```

Out[25]:
```
Credit_History
1.0    475
0.0     89
Name: count, dtype: int64
```

In [26]:
```
df['Property_Area'].unique()
```

Out[26]: array(['Urban', 'Rural', 'Semiurban'], dtype=object)

In [27]:
```
df['Property_Area'].value_counts()
```

Out[27]:
```
Property_Area
Semiurban    233
Urban        202
Rural        179
Name: count, dtype: int64
```

In [28]:
```
df['Loan_Status'].unique()
```

Out[28]: array(['Y', 'N'], dtype=object)

In [29]:
```
df['Loan_Status'].value_counts()
```

Out[29]:
```
Loan_Status
Y    422
N    192
Name: count, dtype: int64
```

In [30]:
```
continuous=['Income' , 'LoanAmount']
```

```
discrete_categorical  = ['Gender', 'Married', 'Education',
       'Self_Employed', 'Credit_History', 'Property_Area', 'Loan_Status']
discrete_count = ['Dependents' , 'Loan_Amount_Term']
```

EDA

For continuous data we do describe , histplot , corr using heatmap , pairplot

In [31]:

```
df[continuous].describe()
```

Out[31]:

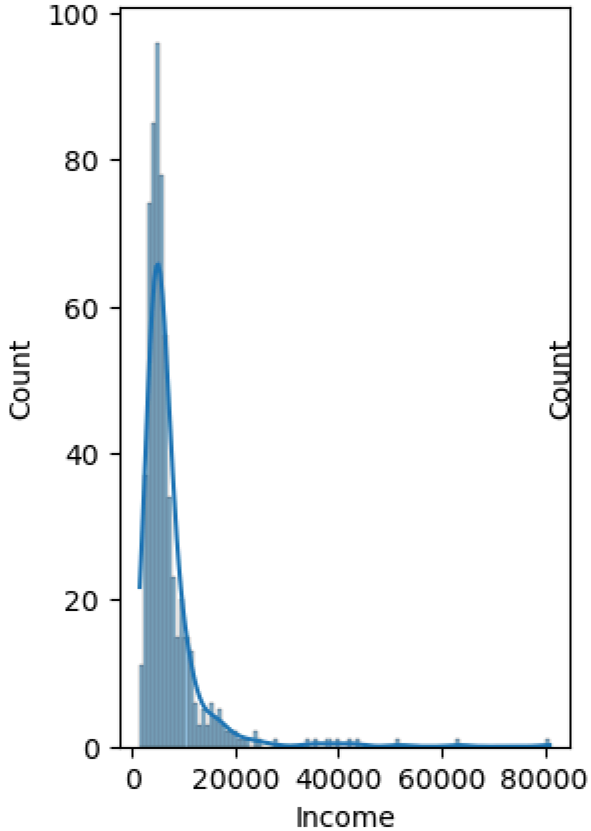
	Income	LoanAmount
count	614.000000	592.000000
mean	7024.705081	146.412162
std	6458.663872	85.587325
min	1442.000000	9.000000
25%	4166.000000	100.000000
50%	5416.500000	128.000000
75%	7521.750000	168.000000
max	81000.000000	700.000000

In [32]:

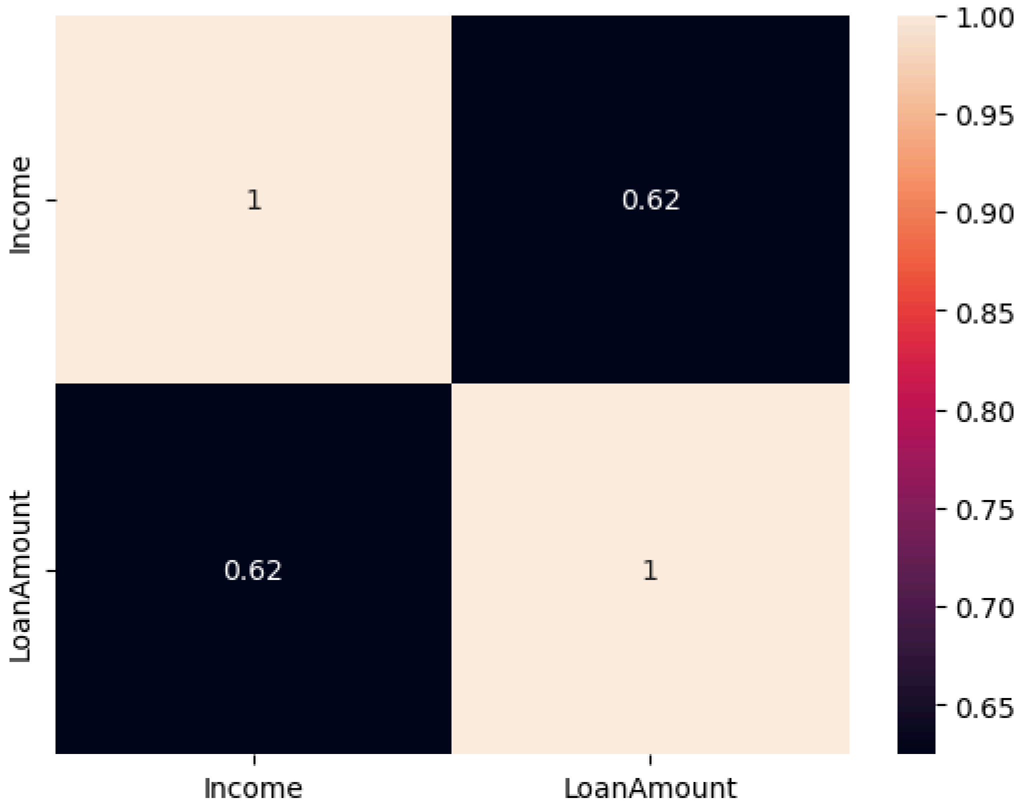
```
plt.subplot(1,2,1)
sns.histplot(df['Income'] , kde = True)

plt.subplot(1,2,2)
sns.histplot(df['LoanAmount'], kde = True)
```

Out[32]:

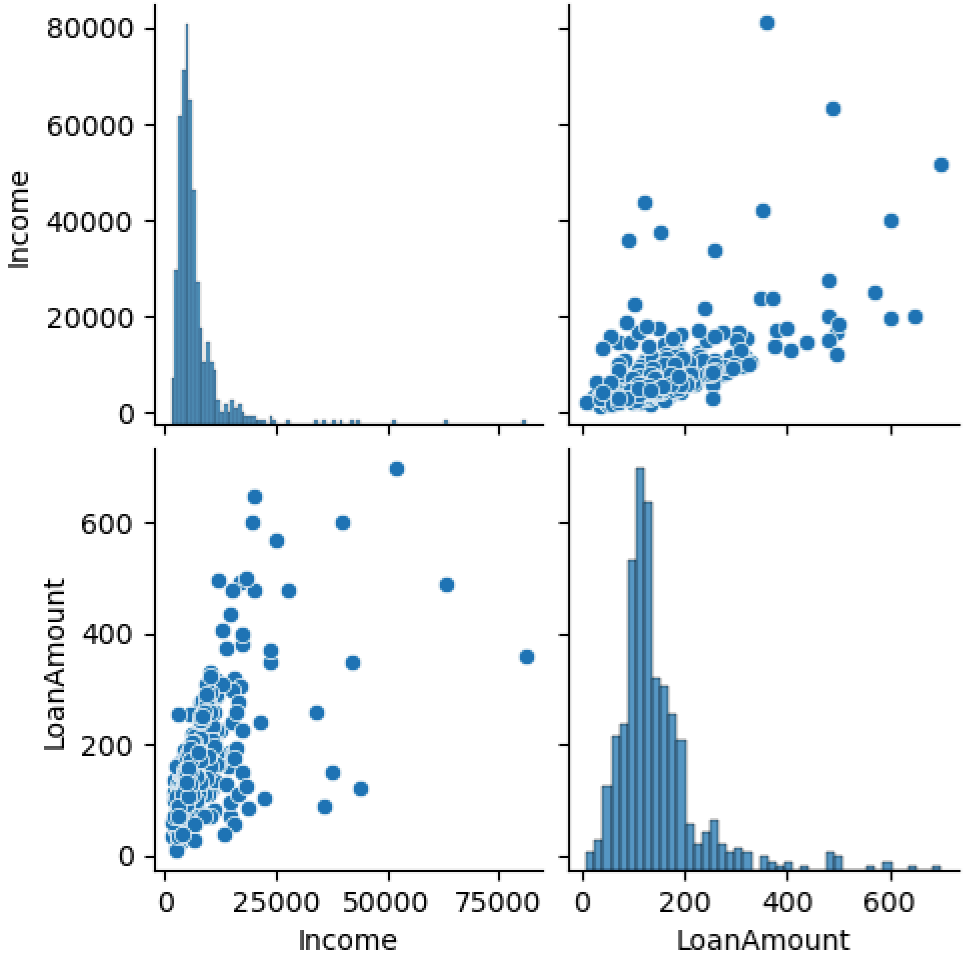
```
<Axes: xlabel='LoanAmount', ylabel='Count'>
```



In [33]:
```
sns.heatmap(df[continuous].corr() , annot = True)
plt.show()
```



In [34]:
```
sns.pairplot(df[continuous])
plt.show()
```



For Discrete_variables

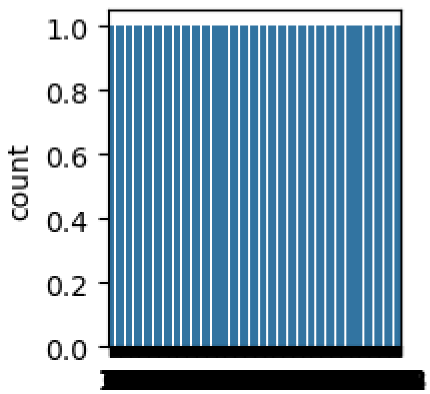
In [35]:
```
df[discrete_categorical].describe()
```

Out[35]:

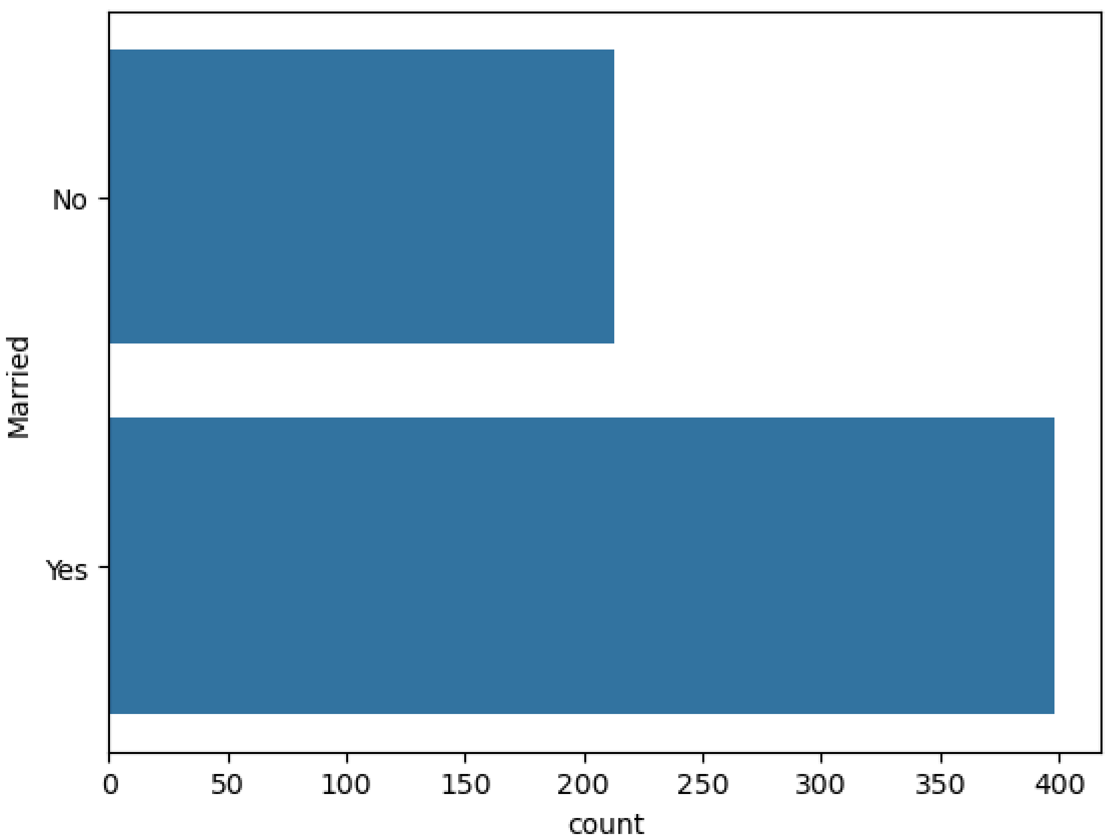
	Credit_History
count	564.000000
mean	0.842199
std	0.364878
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	1.000000

In [36]:
```python
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])
```

In [38]:
```python
df['Gender'] = df['Gender'].replace({'Male': 1 , 'Female' : 0})
plt.subplot(2,3,1)
sns.countplot(df['Gender']) # first change into count values
plt.show()

sns.countplot(df['Married'])
```



Out[38]:
```
<Axes: xlabel='count', ylabel='Married'>
```



In [39]:
```python
df['Gender'].value_counts()
df.columns
```

Out[39]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area',
       'Loan_Status', 'Income'],
      dtype='object')
```

In [40]:
```python
# Lets compare all categories
print("impact of marriage on loan status")
print(pd.crosstab(df['Loan_Status'], df['Married']))
print('\n')

print('impact of dependents on loan status')
print(pd.crosstab(df['Loan_Status'], df['Dependents']))
print('\n')

print('impact of education on loan status')
print(pd.crosstab(df['Loan_Status'] , ['Education']))
print('\n')

print('impact of self employed on loan status')
print(pd.crosstab(df['Loan_Status'] , ['Self_Employed']))
print('\n')

print('impacto of LoanAmount on loan status')
print(pd.crosstab(df['Loan_Status'] , ['LoanAmount']))
print('\n')

print('impacto of Loan_Amount_Term on loan status')
print(pd.crosstab(df['Loan_Status'] , ['Loan_Amount_Term']))
print('\n')

print('impacto of Credit_History on loan status')
print(pd.crosstab(df['Loan_Status'] , ['Credit_History']))
print('\n')

print('impacto of Property_Area on loan status')
print(pd.crosstab(df['Loan_Status'] , ['Property_Area']))
print('\n')

print('impacto of Income on loan status')
print(pd.crosstab(df['Loan_Status'] , ['Income']))
print('\n')
```

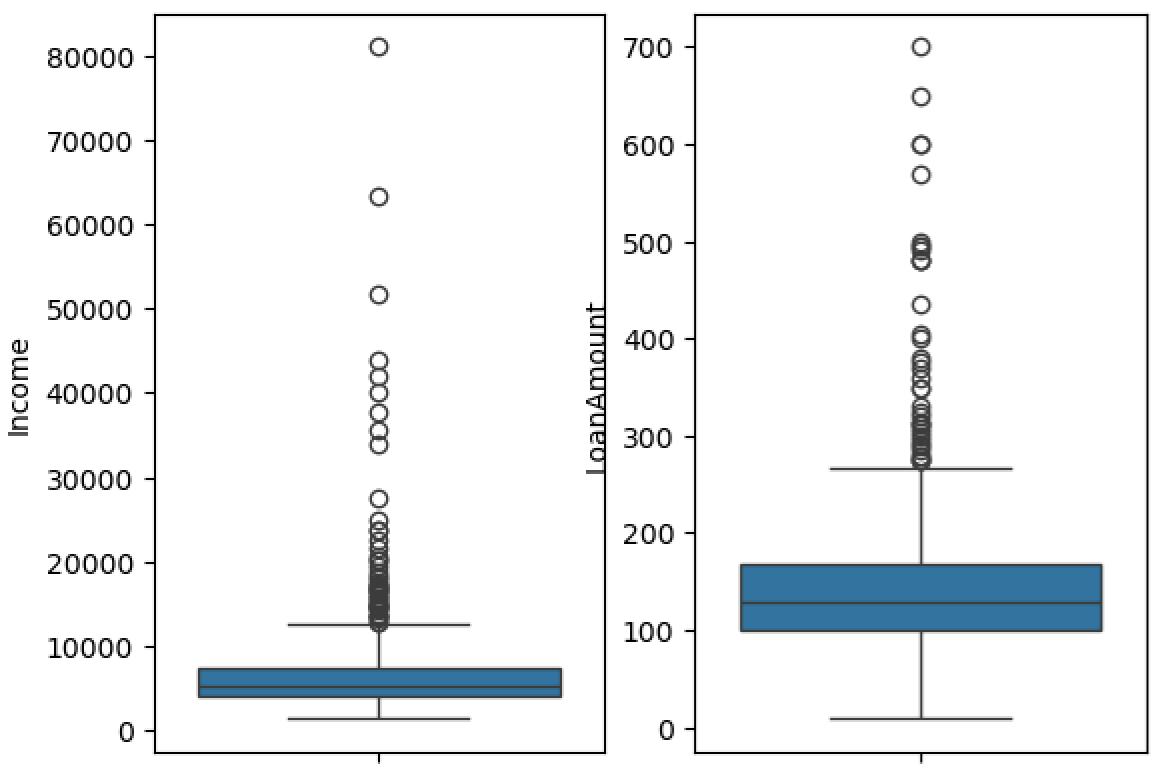
```
impact of marriage on loan status
Married       No  Yes
Loan_Status
N             79  113
Y            134  285


impact of dependents on loan status
Dependents     0   1   2  3+
Loan_Status
N            107  36  25  18
Y            238  66  76  33


impact of education on loan status
col_0        Education
Loan_Status
N                  192
Y                  422


impact of self employed on loan status
col_0        Self_Employed
Loan_Status
N                      192
Y                      422


impacto of LoanAmount on loan status
col_0        LoanAmount
Loan_Status
N                   192
Y                   422


impacto of Loan_Amount_Term on loan status
col_0        Loan_Amount_Term
Loan_Status
N                         192
Y                         422


impacto of Credit_History on loan status
col_0        Credit_History
Loan_Status
N                       192
Y                       422


impacto of Property_Area on loan status
col_0        Property_Area
Loan_Status
N                      192
Y                      422
```

```
impacto of Income on loan status
col_0          Income
Loan_Status
N                 192
Y                 422
```

In [41]:
```
# ckeck skewness
df[continuous].skew()


#here skewness is very high hence we reduce skewwness using boxcox
```

Out[41]:
```
Income        5.633449
LoanAmount    2.677552
dtype: float64
```

In [42]:
```
# check outliers
plt.subplot(1,2,1)
sns.boxplot(df['Income'])

plt.subplot(1,2,2)
sns.boxplot(df['LoanAmount'])
#plt.title('Outliers')
plt.show()
```



Data Preprocessing

Wrong data missimg values wrong data type duplicates outliers

2. Data wangling transformation scaling encoding

In [43]:
```
df.isnull().sum()
```

Out[43]:
```
Gender              0
Married             3
Dependents         15
Education           0
Self_Employed      32
LoanAmount         22
Loan_Amount_Term   14
Credit_History     50
Property_Area       0
Loan_Status         0
Income              0
dtype: int64
```

In [44]:
```
# wrong data treatment
df['Dependents'] = df['Dependents'].replace('3+'  ,3)
df['Dependents'].unique()
```

Out[44]:
```
array(['0', '1', '2', 3, nan], dtype=object)
```

In [48]:
```
df['Gender'].isnull().sum()
```

Out[48]:
```
0
```

In [272…
```
# filling missing values
```

In [49]:
```
df['Married']=df['Married'].fillna(df['Married'].mode()[0])
df['Dependents'] = df['Dependents'].fillna(0).astype(int)
df['Self_Employed']=df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])
df['LoanAmount']=df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Loan_Amount_Term']= df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].m
df['Credit_History']= df['Credit_History'].fillna(df['Credit_History'].mode()[
```

In [50]:
```
#converting wrong data type
df['Dependents'] = df['Dependents'].astype('int')
df['Loan_Amount_Term'].astype('int')
```

Out[50]:
```
0      360
1      360
2      360
3      360
4      360
      ...
609    360
610    180
611    360
612    360
613    360
Name: Loan_Amount_Term, Length: 614, dtype: int32
```

In [51]:
```
df.duplicated().sum()
```

Out[51]:
```
0
```

In [52]:
```
#X = df.drop('Loan_Status' ,axis = 1)
df.columns
```

Out[52]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area',
       'Loan_Status', 'Income'],
      dtype='object')
```

In [53]:
```
# Encoding
#pd.get_dummies(X, drop_first = True).astype(int)
df['Married'] = df['Married'].replace({"Yes":1 , 'No' : 0})
df['Education']  = df['Education'].replace({'Graduate' : 0 , 'Not Graduate' :
df['Self_Employed']= df['Self_Employed'].replace({'No':0, 'Yes':1})
df['Property_Area'] = df['Property_Area'].replace({'Urban' : 2, 'Rural' : 0, '
df['Loan_Status'] = df['Loan_Status'].replace({'Y' : 1 , 'N' : 0})
```

In [54]:
```
df['Loan_Status'].unique()
```

Out[54]:
```
array([1, 0], dtype=int64)
```

In [55]:
```
df['Loan_Amount_Term']=df['Loan_Amount_Term']/12
```

In [56]:
```
# Transformation (boz we have noticed some skew )
from scipy.stats import boxcox
df['Income'],a = boxcox(df['Income'])
df['LoanAmount'] , c = boxcox(df['LoanAmount'])
```

In [57]:
```
df[continuous].skew() #al gets normalized
```

Out[57]:
```
Income        -0.034662
LoanAmount     0.030458
dtype: float64
```

In [59]:
```
X = df.drop(columns=['Loan_Status'])
y = df['Loan_Status']
```

In [60]:
```
from sklearn.model_selection import train_test_split
X_train  ,X_test , y_train , y_test = train_test_split(X , y ,train_size = 0.8
```

In [64]:
```
scaling  = ['LoanAmount' ,'Loan_Amount_Term' , 'Income' ]
```

In [65]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[scaling] = sc.fit_transform(X_train[scaling])
X_test_scaled[scaling] = sc.transform(X_test[scaling])
```

In [66]:
```
 # Modelling and evalutation
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import plot_tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score
```

In [67]:
```
df.head()
```

Out[67]:

	Gender	Married	Dependents	Education	Self_Employed	LoanAmount	Loan_
0	1	0	0	0	0	6.034999	
1	1	1	1	0	0	5.841340	
2	1	1	0	0	1	4.914615	
3	1	1	0	1	0	5.749027	
4	1	0	0	0	0	5.980529	

In [68]:
```
#------Creating a DataFrame that stores all the metrics and performance of eac
algorithms = ['logistic_Model', 'knn_Model', 'svm_Model', 'dt_Model', 'rf_Mode
metrics = ['TrainAccuracy', 'TestAccuracy', 'TrainPrecision', 'TestPrecision',
           'TrainF1', 'TestF1', 'CV']

analysis_df = pd.DataFrame(index=algorithms, columns=metrics)
```

In [69]:
```
analysis_df
```

Out[69]:

	TrainAccuracy	TestAccuracy	TrainPrecision	TestPrecision	TrainI
logistic_Model	NaN	NaN	NaN	NaN	
knn_Model	NaN	NaN	NaN	NaN	
svm_Model	NaN	NaN	NaN	NaN	
dt_Model	NaN	NaN	NaN	NaN	
rf_Model	NaN	NaN	NaN	NaN	
ada_Model	NaN	NaN	NaN	NaN	
gb_Model	NaN	NaN	NaN	NaN	
xg_Model	NaN	NaN	NaN	NaN	

In [70]:

```
#---Function that calculates all the metrics and Classification report and upc
def model_performance(model_key, model_obj, X_train, y_train, X_test, y_test,
    y_train_pred = model_obj.predict(X_train)
    y_test_pred = model_obj.predict(X_test)


    analysis_df.loc[model_key, 'TrainAccuracy'] = accuracy_score(y_train, y_tr
    analysis_df.loc[model_key, 'TestAccuracy'] = accuracy_score(y_test, y_test
    analysis_df.loc[model_key, 'TrainPrecision'] = precision_score(y_train, y_
    analysis_df.loc[model_key, 'TestPrecision'] = precision_score(y_test, y_te
    analysis_df.loc[model_key, 'TrainRecall'] = recall_score(y_train, y_train_
    analysis_df.loc[model_key, 'TestRecall'] = recall_score(y_test, y_test_pre
    analysis_df.loc[model_key, 'TrainF1'] = f1_score(y_train, y_train_pred)
    analysis_df.loc[model_key, 'TestF1'] = f1_score(y_test, y_test_pred)

    cv_score = cross_val_score(model_obj, X_train, y_train, cv=5, scoring='acc
    analysis_df.loc[model_key, 'CV'] = cv_score

    print(f'◈ Classification Report — {model_key} (Train)')
    print(classification_report(y_train, y_train_pred))
    print(f'◈ Classification Report — {model_key} (Test)')
    print(classification_report(y_test, y_test_pred))

    # Confusion Matrix - Train
    cm_train = confusion_matrix(y_train, y_train_pred)
    disp_train = ConfusionMatrixDisplay(confusion_matrix=cm_train)
    disp_train.plot(cmap='Blues')
    plt.title(f'{model_key} — Confusion Matrix (Train)')
    plt.show()

    # Confusion Matrix - Test
    cm_test = confusion_matrix(y_test, y_test_pred)
    disp_test = ConfusionMatrixDisplay(confusion_matrix=cm_test)
    disp_test.plot(cmap='Oranges')
    plt.title(f'{model_key} — Confusion Matrix (Test)')
    plt.show()
```

```python
    return analysis_df
```

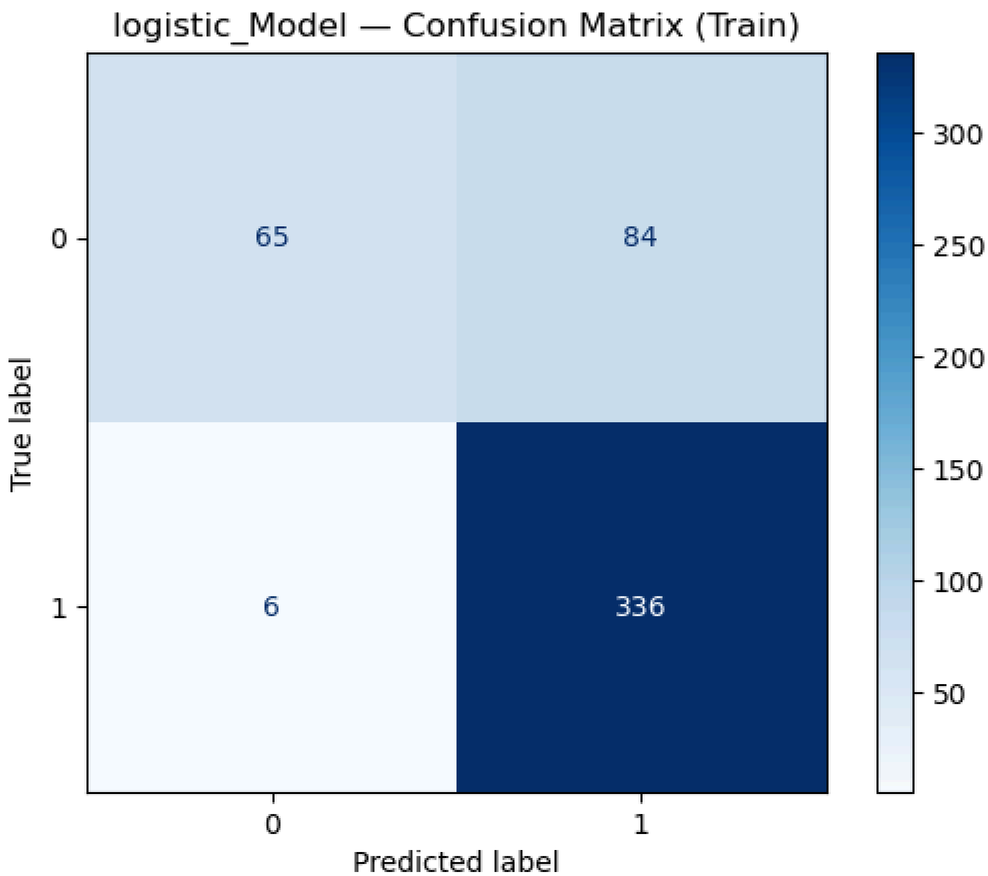
In [81]:
```python
from sklearn.metrics import precision_score, recall_score, f1_score, Confusion
```

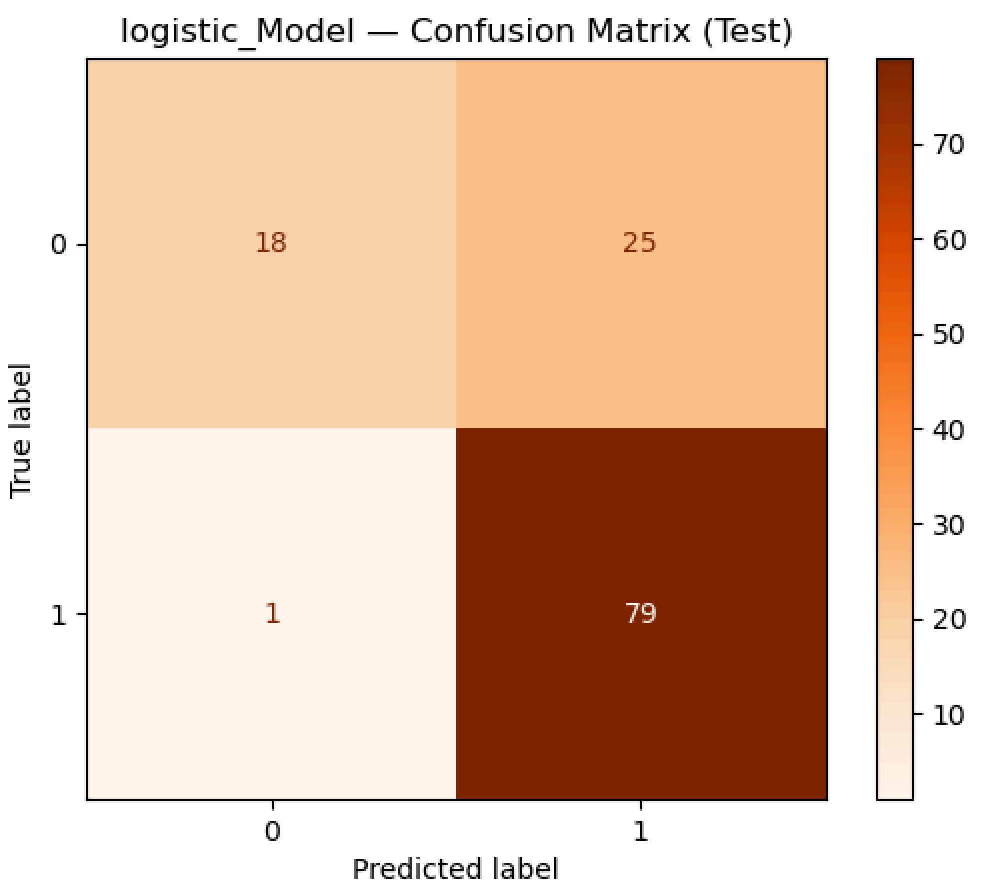
Logistic Regresion

In [72]:
```python
lr = LogisticRegression()
lr.fit(X_train_scaled , y_train) # Here you did mistake [Didn't give scaled In
ypred_train  = lr.predict(X_train_scaled)
ypred_test  = lr.predict(X_test_scaled)
```

In [82]:
```python
logistic_model_report = model_performance('logistic_Model', lr, X_train_scaled
```

```
◈ Classification Report — logistic_Model (Train)
              precision    recall  f1-score   support

           0       0.92      0.44      0.59       149
           1       0.80      0.98      0.88       342

    accuracy                           0.82       491
   macro avg       0.86      0.71      0.74       491
weighted avg       0.84      0.82      0.79       491

◈ Classification Report — logistic_Model (Test)
              precision    recall  f1-score   support

           0       0.95      0.42      0.58        43
           1       0.76      0.99      0.86        80

    accuracy                           0.79       123
   macro avg       0.85      0.70      0.72       123
weighted avg       0.83      0.79      0.76       123
```


logistic_Model — Confusion Matrix (Train)
True label
Predicted label
0
1
65
84
6
336
300
250
200
150
100
50


logistic_Model — Confusion Matrix (Test)
True label
Predicted label
0
1
18
25
1
79
70
60
50
40
30
20
10

In [83]:
```
# hyper parameter tuning
estimator = KNeighborsClassifier()
param_grid = {'n_neighbors': list(range(1,20))}
grid = GridSearchCV(estimator , param_grid , cv = 5 , scoring= 'recall')
grid.fit(X_train , y_train)
```

Out[83]:

```
▸ GridSearchCV
▸ best_estimator_: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

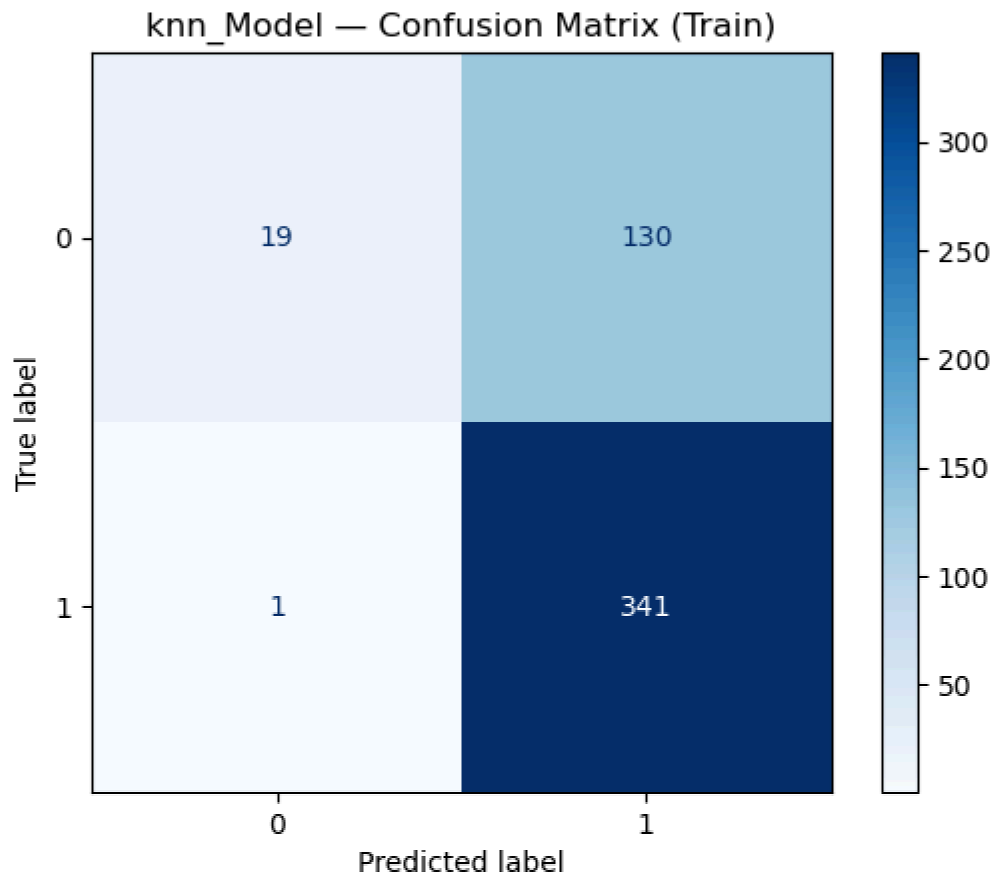
In [84]:
```
grid.best_params_
```

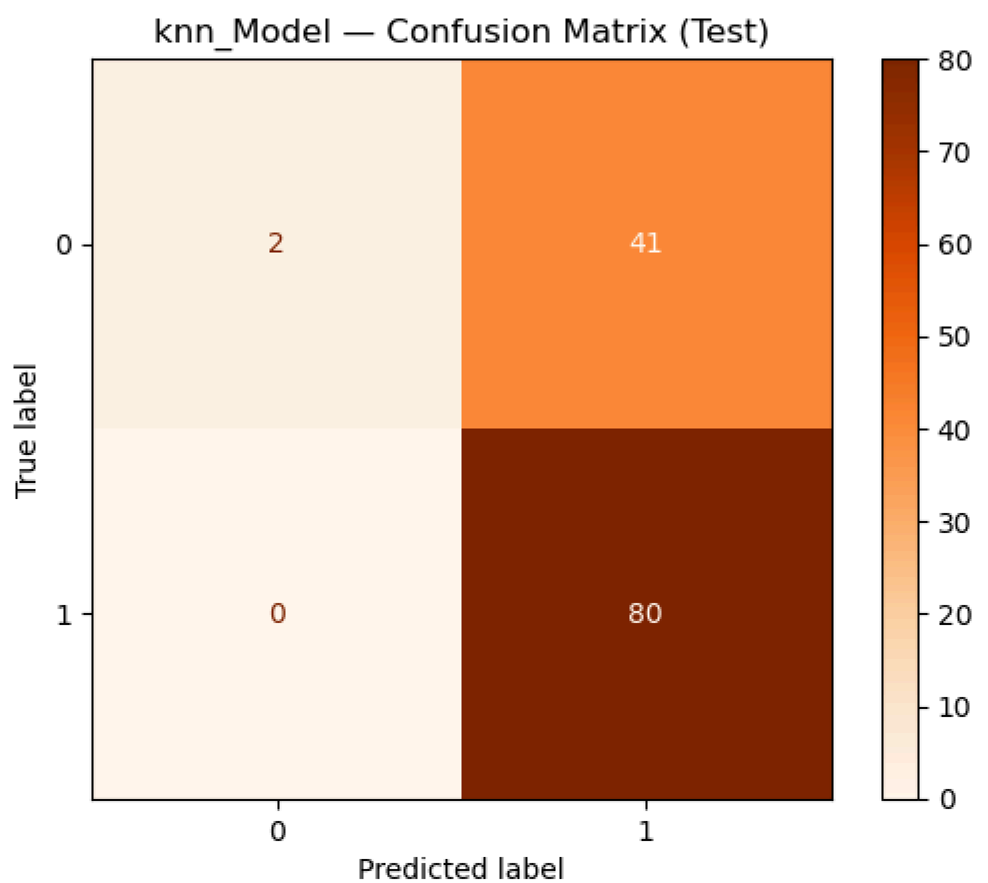
Out[84]:
```
{'n_neighbors': 19}
```

In [85]:
```
knn = KNeighborsClassifier(n_neighbors  = 19)
knn.fit(X_train_scaled , y_train)
ypred_train = knn.predict(X_train_scaled)
ypred_test = knn.predict(X_test_scaled)
```

In [86]:
```
logistic_model_report = model_performance('knn_Model', knn, X_train_scaled, y_
```

```
◈ Classification Report — knn_Model (Train)
              precision    recall  f1-score   support

           0       0.95      0.13      0.22       149
           1       0.72      1.00      0.84       342

    accuracy                           0.73       491
   macro avg       0.84      0.56      0.53       491
weighted avg       0.79      0.73      0.65       491

◈ Classification Report — knn_Model (Test)
              precision    recall  f1-score   support

           0       1.00      0.05      0.09        43
           1       0.66      1.00      0.80        80

    accuracy                           0.67       123
   macro avg       0.83      0.52      0.44       123
weighted avg       0.78      0.67      0.55       123
```


knn_Model — Confusion Matrix (Train)
True label
Predicted label
0
1
0
1
19
130
1
341
300
250
200
150
100
50


knn_Model — Confusion Matrix (Test)
True label
Predicted label
0
1
0
1
2
41
0
80
80
70
60
50
40
30
20
10
0

SVM

In [87]:
```
estimator = SVC()
param_grid = {'C' : [0.01 , 0.1 , 1] , 'kernel' : ['poly','rbf' , 'linear' , '

grid = GridSearchCV(estimator , param_grid , cv= 5 , scoring='recall')
grid.fit(X_train_scaled , y_train)
```

Out[87]:
```
▸ GridSearchCV
▸ best_estimator_: SVC
    ▸ SVC
```

In [88]:
```
grid.best_params_
```

Out[88]:
```
{'C': 0.01, 'kernel': 'poly'}
```

In [89]:
```
svm = SVC(
    C=0.01,
    kernel='poly'
)
```

In [90]:
```
svm.fit(X_train_scaled, y_train)
```

Out[90]:
```
▾ SVC
SVC(C=0.01, kernel='poly')
```

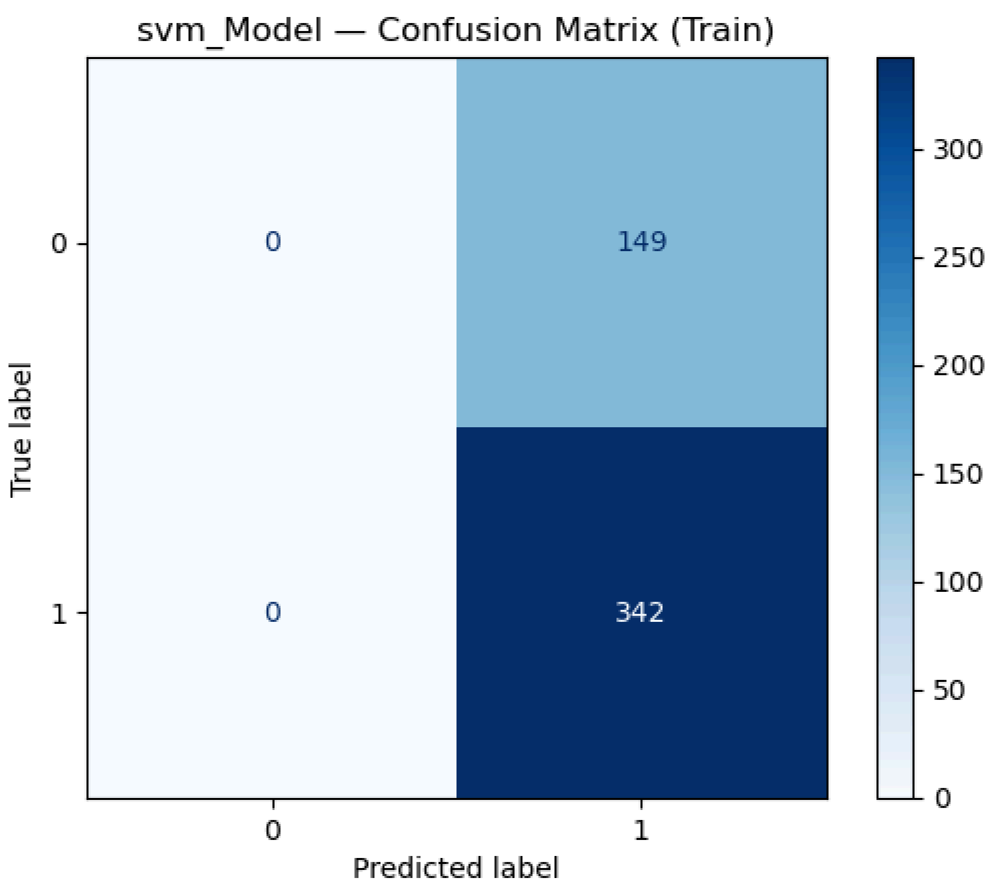
In [92]:
```
analysis_df
```

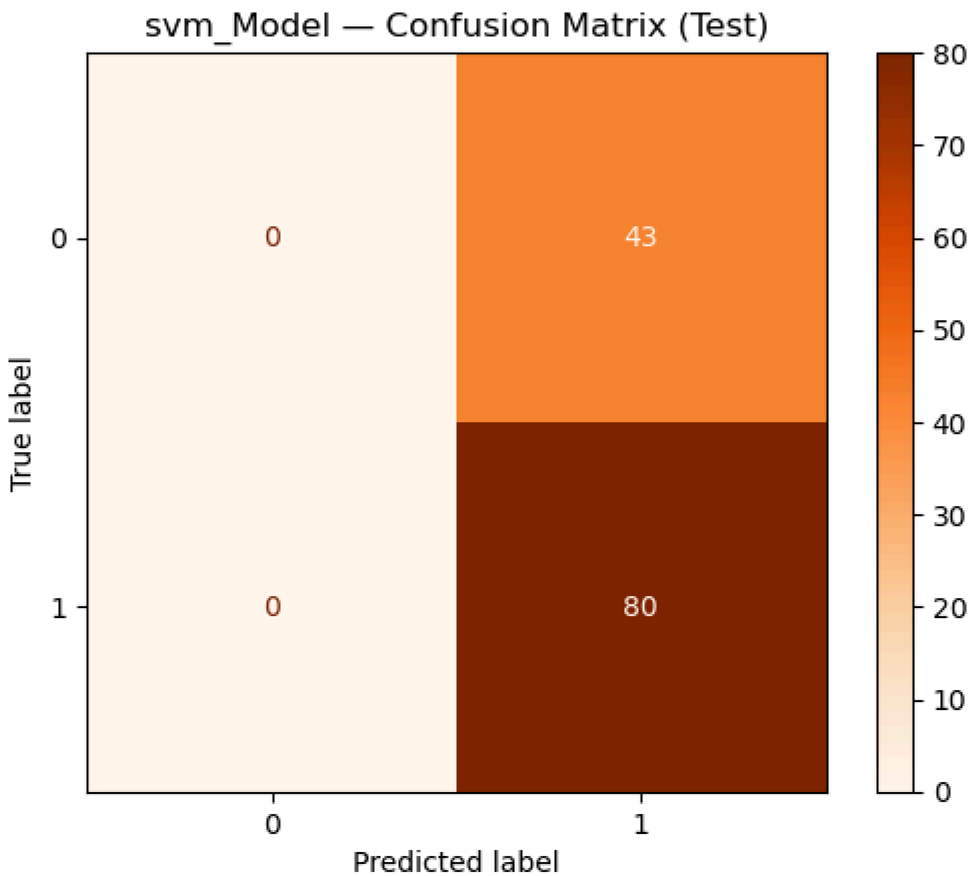
Out[92]:

	TrainAccuracy	TestAccuracy	TrainPrecision	TestPrecision	TrainI
logistic_Model	0.816701	0.788618	0.8	0.759615	0.9
knn_Model	0.733198	0.666667	0.723992	0.661157	0.9
svm_Model	NaN	NaN	NaN	NaN	
dt_Model	NaN	NaN	NaN	NaN	
rf_Model	NaN	NaN	NaN	NaN	
ada_Model	NaN	NaN	NaN	NaN	
gb_Model	NaN	NaN	NaN	NaN	
xg_Model	NaN	NaN	NaN	NaN	

In [93]:
```
svm_model_report = model_performance('svm_Model', svm, X_train_scaled, y_trair
```

```
◈ Classification Report — svm_Model (Train)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       149
           1       0.70      1.00      0.82       342

    accuracy                           0.70       491
   macro avg       0.35      0.50      0.41       491
weighted avg       0.49      0.70      0.57       491

◈ Classification Report — svm_Model (Test)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        43
           1       0.65      1.00      0.79        80

    accuracy                           0.65       123
   macro avg       0.33      0.50      0.39       123
weighted avg       0.42      0.65      0.51       123
```



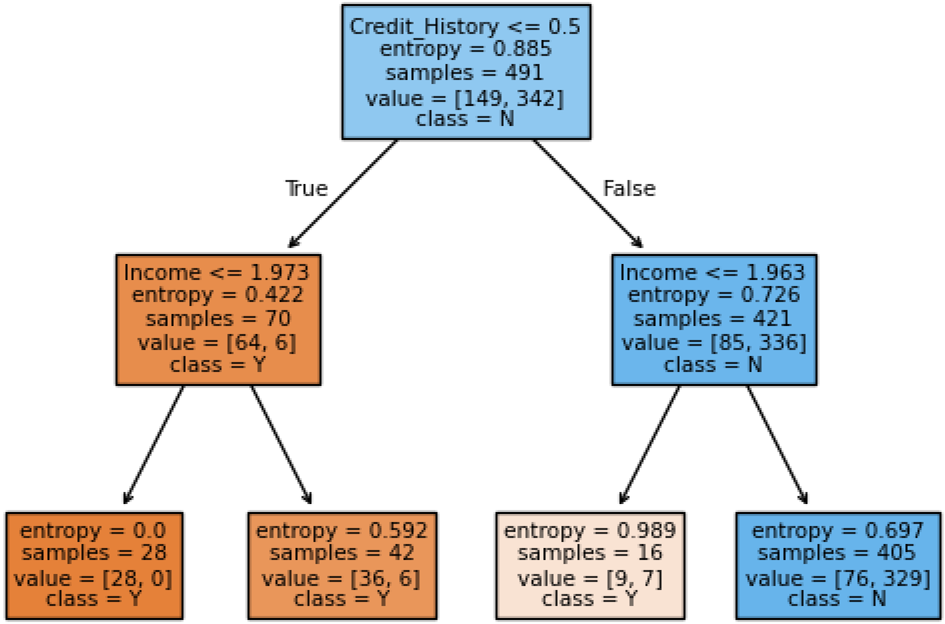


Decision Tree Classifier

In [95]:
```
estimator = DecisionTreeClassifier()
param_grid = {'criterion' : ['gini' , 'entropy'] , 'max_depth'  : list(range(1
grid = GridSearchCV(estimator , param_grid  , cv =5 , scoring = 'accuracy')
grid.fit(X_train , y_train)
grid.best_estimator_
```

Out[95]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

In [96]:
```
dt = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=2
)
dt.fit(X_train , y_train)
plot_tree(dt ,filled = True , feature_names=X_train.columns.to_list() , class_
plt.show()
```



In [97]:
```
s1 = pd.DataFrame(grid.best_estimator_.feature_importances_ , columns = ['imp'
s1
```

Out[97]:

	imp
Gender	0.000000
Married	0.000000
Dependents	0.000000
Education	0.000000
Self_Employed	0.000000
LoanAmount	0.000000
Loan_Amount_Term	0.000000
Credit_History	0.890206
Property_Area	0.000000
Income	0.109794

In [98]:
```
imp_features = s1[s1['imp']>0].index.to_list()
imp_features
```

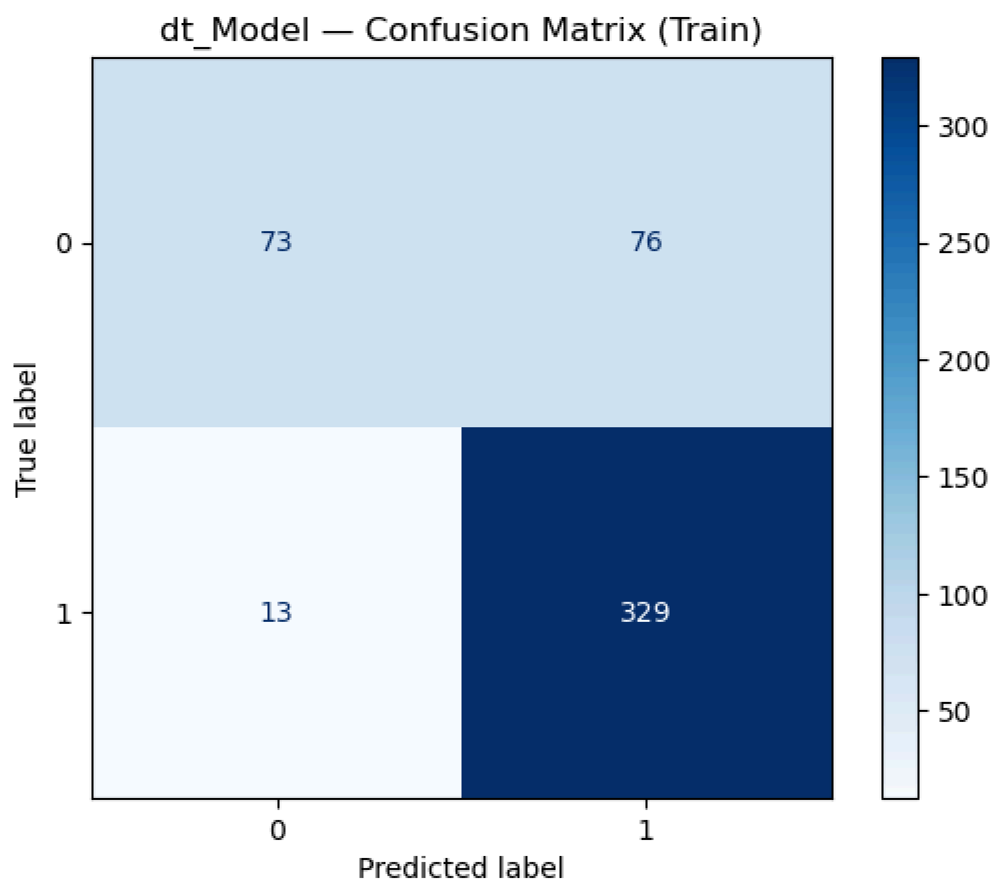
Out[98]: ['Credit_History', 'Income']

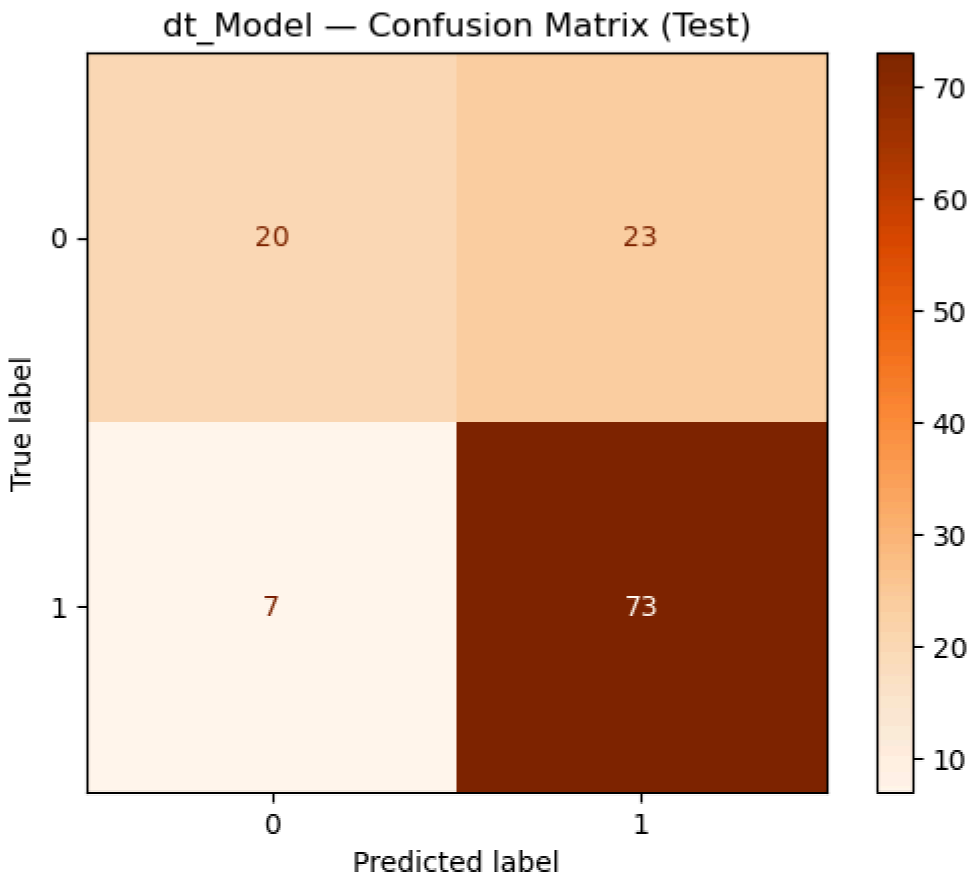
In [99]:
```
analysis_df.index
```

```
Out[99]: Index(['logistic_Model', 'knn_Model', 'svm_Model', 'dt_Model', 'rf_Model',
               'ada_Model', 'gb_Model', 'xg_Model'],
              dtype='object')
```

```
In [102…  decision_model_report = model_performance('dt_Model', dt, X_train, y_train, X_
```

```
◈ Classification Report — dt_Model (Train)
              precision    recall  f1-score   support

           0       0.85      0.49      0.62       149
           1       0.81      0.96      0.88       342

    accuracy                           0.82       491
   macro avg       0.83      0.73      0.75       491
weighted avg       0.82      0.82      0.80       491

◈ Classification Report — dt_Model (Test)
              precision    recall  f1-score   support

           0       0.74      0.47      0.57        43
           1       0.76      0.91      0.83        80

    accuracy                           0.76       123
   macro avg       0.75      0.69      0.70       123
weighted avg       0.75      0.76      0.74       123
```





RF

In [103…
```
estimator  = RandomForestClassifier()
param_grid = ({'n_estimators' : list(range(1,50))})
grid= GridSearchCV(estimator , param_grid , cv= 5 , scoring = 'accuracy')
grid.fit(X_train , y_train)
grid.best_estimator_
```

Out[103…

RandomForestClassifier

RandomForestClassifier(n_estimators=47)

In [105…
```
rf_Model = RandomForestClassifier(
    n_estimators=47
)

rf_Model.fit(X_train, y_train)
```

Out[105…

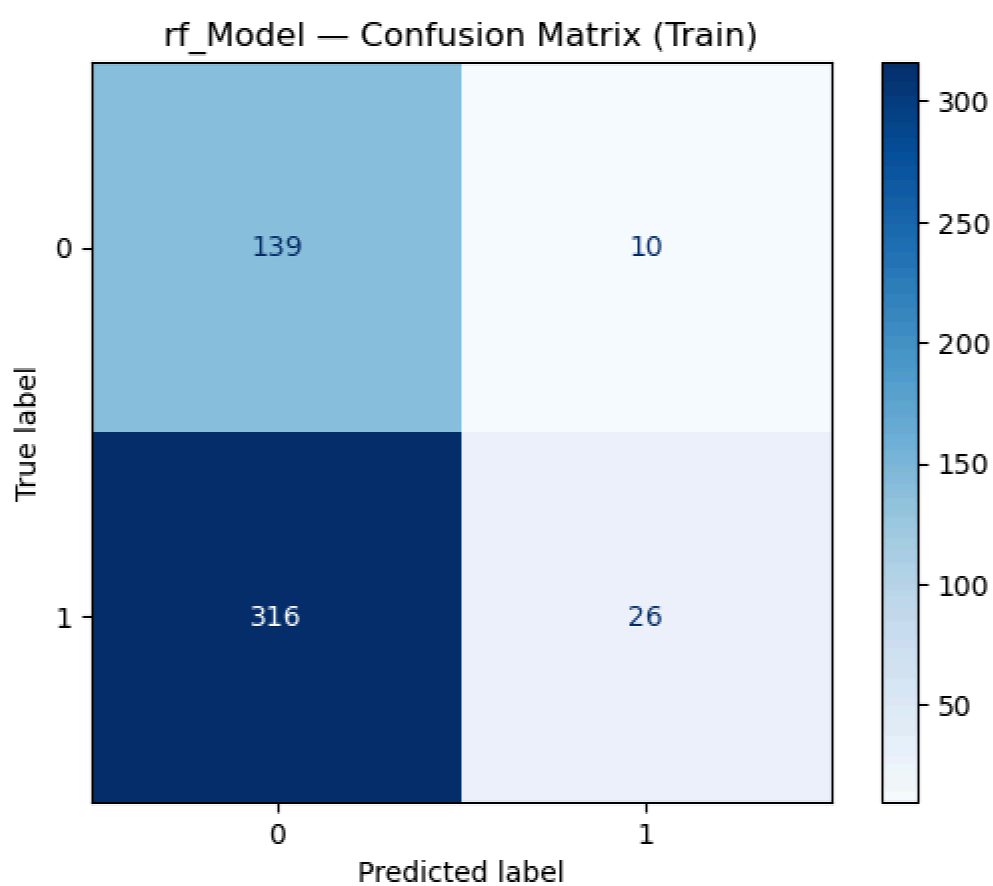
RandomForestClassifier

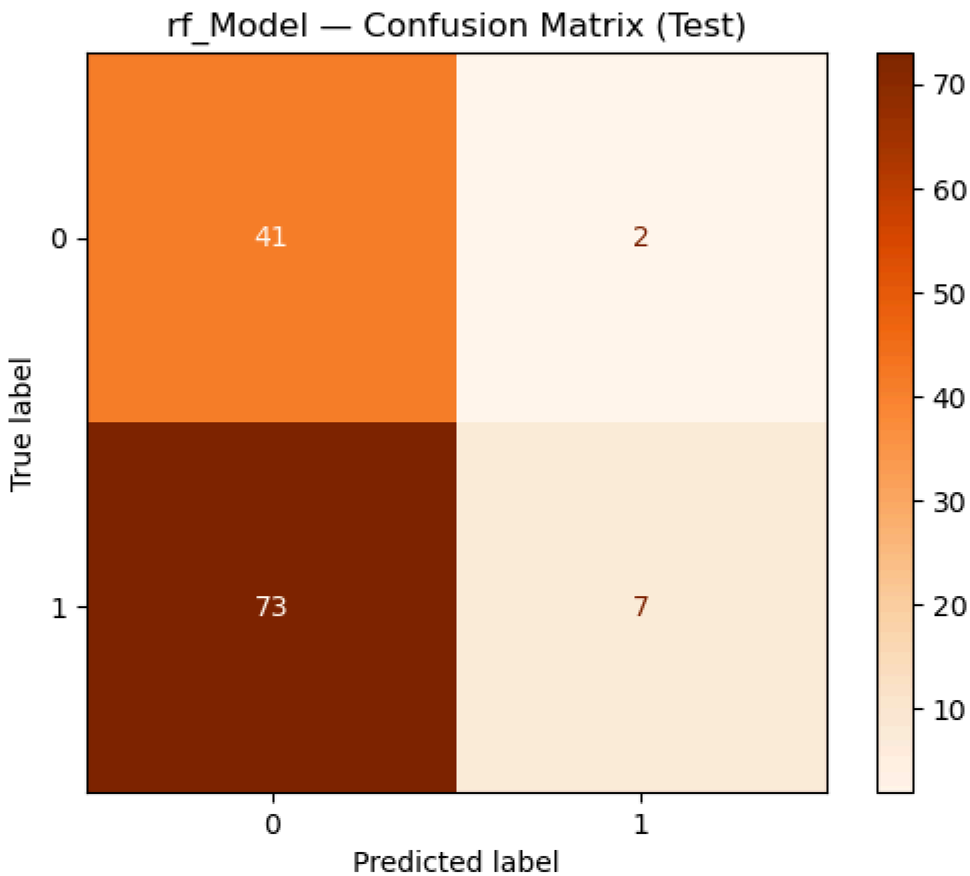
RandomForestClassifier(n_estimators=47)

In [106…

```
rf_model_report = model_performance('rf_Model', rf_Model, X_train_scaled, y_tr
```

```
◈ Classification Report — rf_Model (Train)
              precision    recall  f1-score   support

           0       0.31      0.93      0.46       149
           1       0.72      0.08      0.14       342

    accuracy                           0.34       491
   macro avg       0.51      0.50      0.30       491
weighted avg       0.60      0.34      0.24       491

◈ Classification Report — rf_Model (Test)
              precision    recall  f1-score   support

           0       0.36      0.95      0.52        43
           1       0.78      0.09      0.16        80

    accuracy                           0.39       123
   macro avg       0.57      0.52      0.34       123
weighted avg       0.63      0.39      0.28       123
```





ADA boost

In [107…

```
estimator = AdaBoostClassifier()
param_grid = {'n_estimators' : list(range(1,20))}
grid = GridSearchCV(estimator , param_grid , cv = 5 , scoring = 'accuracy')
grid.fit(X_train , y_train)
grid.best_estimator_
```

Out[107…

```
▾ AdaBoostClassifier
AdaBoostClassifier(n_estimators=11)
```

In [108…

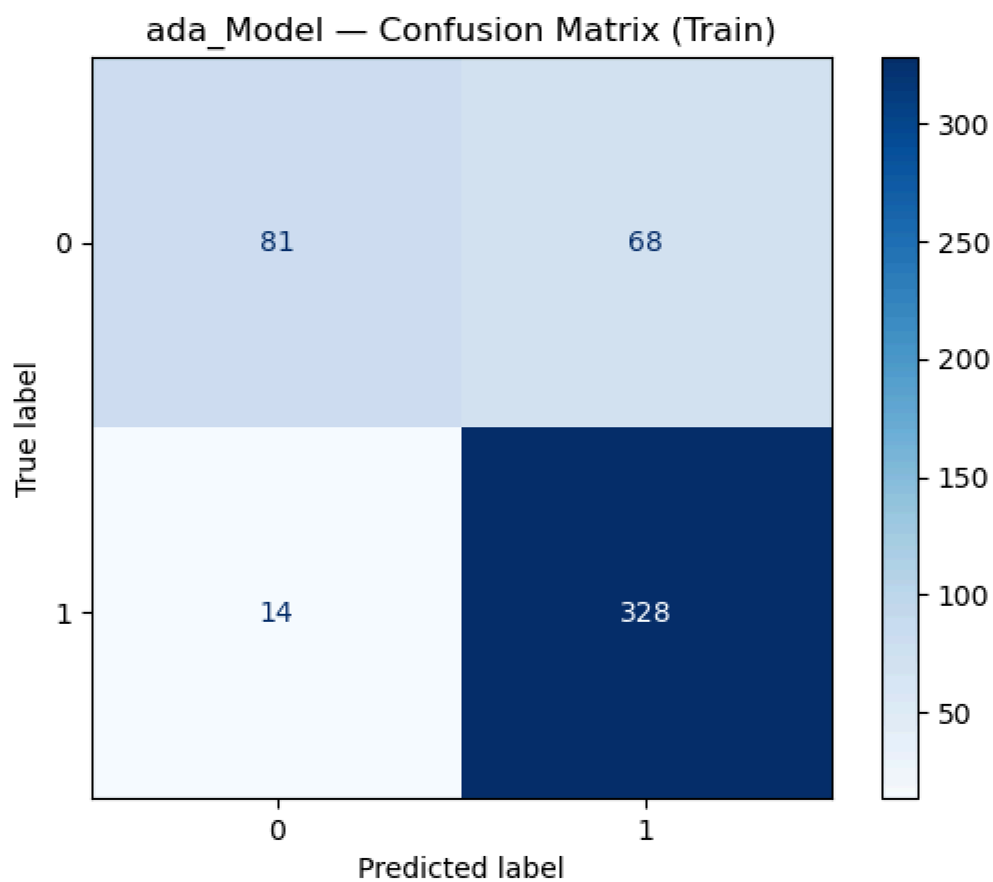
```
ada_Model = AdaBoostClassifier(
    n_estimators=11
)

ada_Model.fit(X_train, y_train)
```

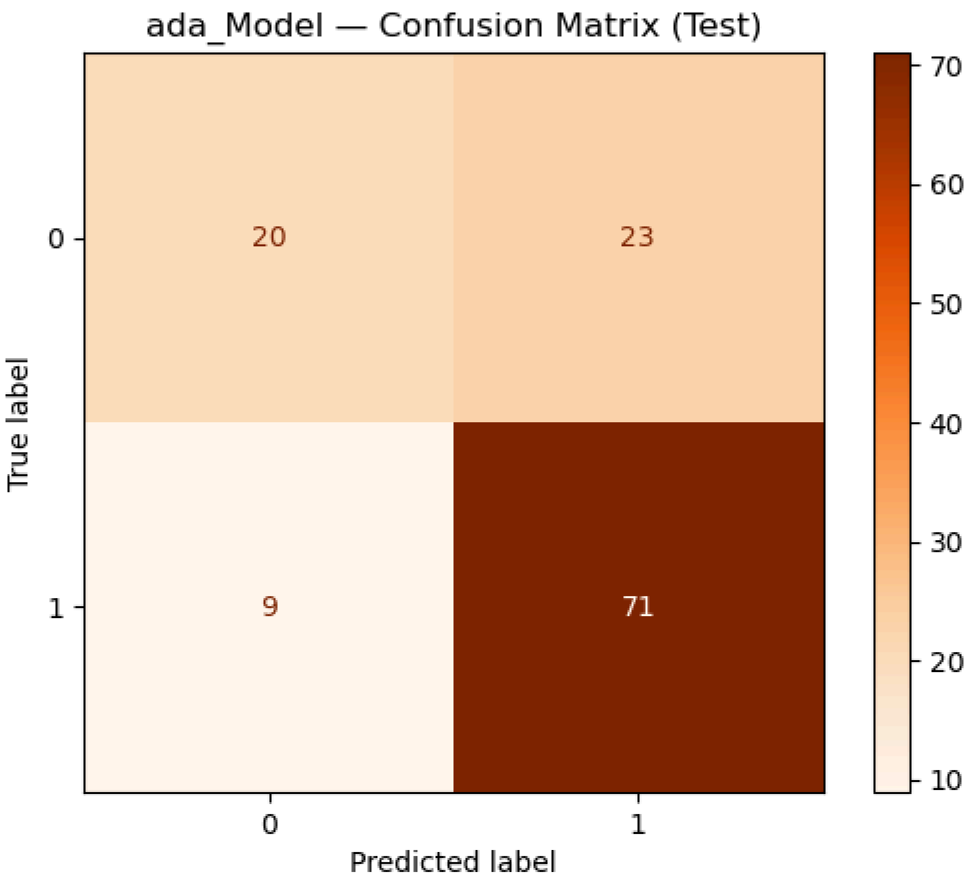
Out[108…

```
▾ AdaBoostClassifier
AdaBoostClassifier(n_estimators=11)
```

```
In [113… ada_model_report = model_performance('ada_Model', ada_Model, X_train, y_train,
```

```
🔶 Classification Report — ada_Model (Train)
              precision    recall  f1-score   support

           0       0.85      0.54      0.66       149
           1       0.83      0.96      0.89       342

    accuracy                           0.83       491
   macro avg       0.84      0.75      0.78       491
weighted avg       0.84      0.83      0.82       491

🔶 Classification Report — ada_Model (Test)
              precision    recall  f1-score   support

           0       0.69      0.47      0.56        43
           1       0.76      0.89      0.82        80

    accuracy                           0.74       123
   macro avg       0.72      0.68      0.69       123
weighted avg       0.73      0.74      0.73       123
```





GradientBoosting

In [110…

```
estimator = GradientBoostingClassifier()
param_grid = {'n_estimators' : list(range(1,20)), 'learning_rate':[0.1,0.2,0.3
grid = GridSearchCV(estimator , param_grid , cv = 5 , scoring = 'accuracy')
grid.fit(X_train , y_train)
grid.best_estimator_
```

Out[110…

```
GradientBoostingClassifier
GradientBoostingClassifier(n_estimators=5)
```

In [111…

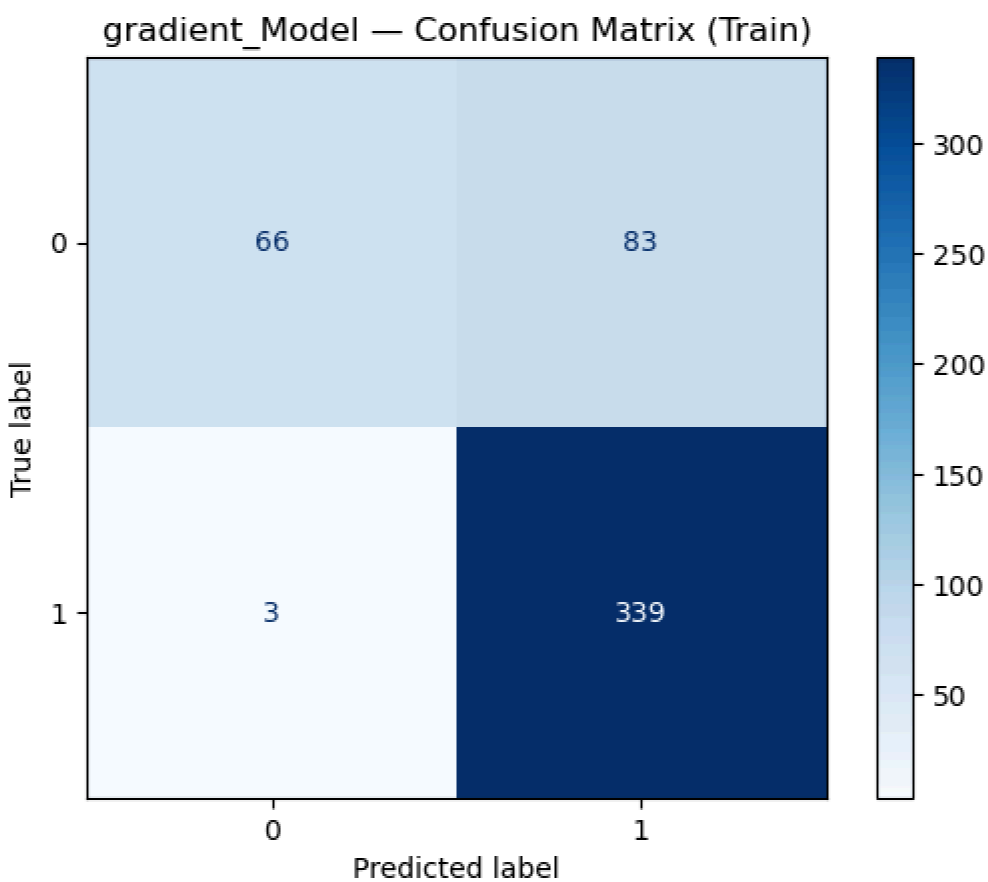
```
gradient_Model = GradientBoostingClassifier(
    n_estimators=5
)

gradient_Model.fit(X_train, y_train)
```

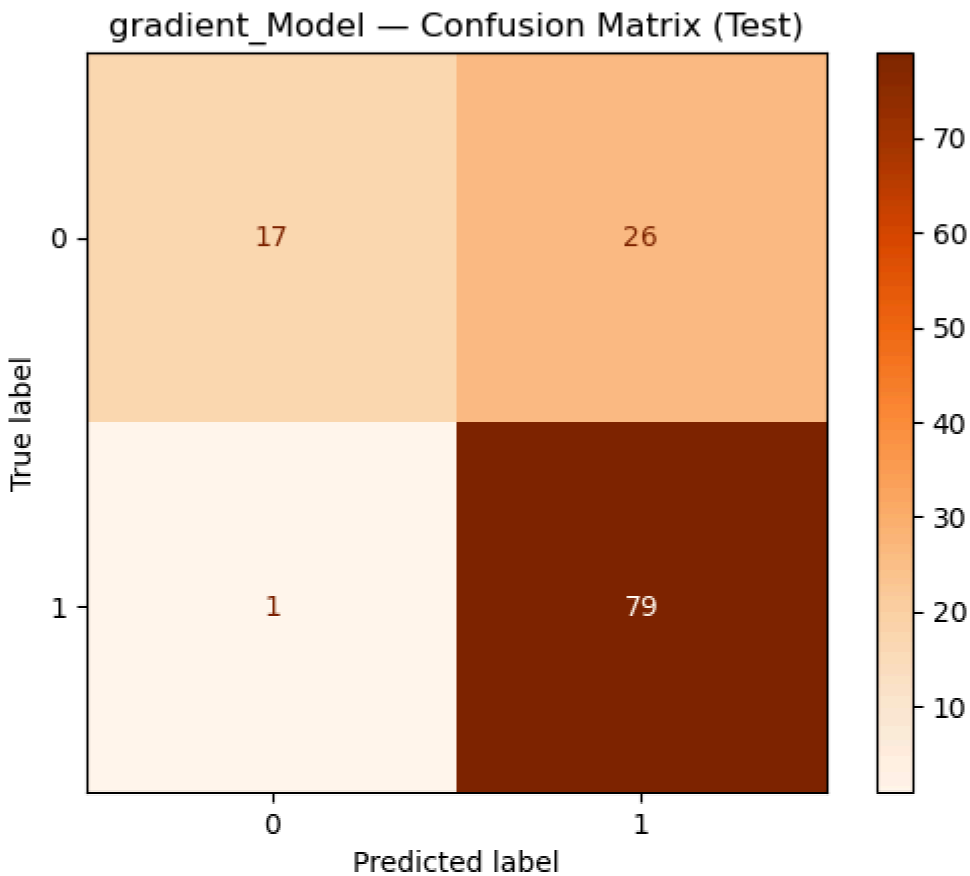
Out[111…

```
GradientBoostingClassifier
GradientBoostingClassifier(n_estimators=5)
```

```
In [112…  gradient_model_report = model_performance('gradient_Model', gradient_Model, X_
```

```
◈ Classification Report — gradient_Model (Train)
              precision    recall  f1-score   support

           0       0.96      0.44      0.61       149
           1       0.80      0.99      0.89       342

    accuracy                           0.82       491
   macro avg       0.88      0.72      0.75       491
weighted avg       0.85      0.82      0.80       491

◈ Classification Report — gradient_Model (Test)
              precision    recall  f1-score   support

           0       0.94      0.40      0.56        43
           1       0.75      0.99      0.85        80

    accuracy                           0.78       123
   macro avg       0.85      0.69      0.71       123
weighted avg       0.82      0.78      0.75       123
```





In [216…
```
imp_features = s1[s1['imp']>0].index.to_list()
imp_features
```

Out[216… ['Credit_History']

In [217…
```
X_imp= X[imp_feature]
X_train , X_test , y_train , y_test = train_test_split(X_imp , y, train_size =
gb=GradientBoostingClassifier(n_estimators = 4)
gb.fit(X_train, y_train)

print('accuracy_score' , accuracy_score(ypred_train , y_train))
print('cvs:'  , cross_val_score(gb , X_train , y_train , cv = 5).mean())
print('test_accuracy:' , accuracy_score(ypred_test , y_test))
```

```
accuracy_score 0.6496945010183299
cvs: 0.8227994227994226
test_accuracy: 0.6178861788617886
```

XG Boosting

In [218…
```
from xgboost import XGBClassifier
estimator = XGBClassifier()
param_grid = {'n_estimators' : [10,20,40,100], 'learning_rate':[0.1,0.05,0.5,1
grid = GridSearchCV(estimator , param_grid , cv = 5 , scoring = 'accuracy')
```

```
grid.fit(X_train , y_train)
grid.best_estimator_
```

Out[218…

```
▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_roun
ds=None,
              enable_categorical=False, eval_metric=None, feature_typ
es=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=Non
e,
```

In [ ]:

```
# Prediction Function
def user_input(model):
    Male = int(input('Enter 1 for Male and 0 for Female: '))
    Married = int(input('Enter 1 for Married and 0 for Unmarried: '))
    Dependents = int(input('Enter the Number of Dependents: '))
    Education = int(input('Enter 1 if you are educated else 0: '))
    Self_Employed = int(input('Enter 1 for Self Employed and 0 for Not: '))
    LoanAmount = float(input('Enter the Desired Loan Amount: '))
    Loan_Amount_Term = int(input('Enter the loan amount term: '))
    CreditHistory = int(input('Enter 1 if you have past credit history else 0:
    Area = int(input('Choose 1 for Rural, 2 for Semiurban and 3 for Urban: '))
    income = float(input('Enter your income: '))

    #Scaling of Continous Variables
    amount_income = np.array([[LoanAmount ,Loan_Amount_Term , income]])
    scale = sc.transform(amount_income)
    LoanAmount_Scaled = scale[0][0]
    Loan_Amount_Term_Scaled = scale[0][1]
    income_scaled = scale[0][2]

    # Create input array
    input_data = np.array([[Male, Married, Dependents, Self_Employed,Educatior

    prediction = model.predict(input_data)

    if prediction[0] == 1:
        print('� The User is Eligible for Loan')
    else:
        print('� The User is NOT Eligible for Loan')

# Run the function
user_input(lr)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:
```