In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv("C:\\Users\\lenovo\\Desktop\\Data Science course\\loan dataset Analytics vidya\\train_ctrUa4K.csv")
```

In [3]:
```
df
```

Out[3]:

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

614 rows × 13 columns

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

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	

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

Out[5]:
```
(614, 13)
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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 [7]:
```
df.columns
```

Out[7]:
```
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.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
```

In [9]:
```
df.drop('Loan_ID',axis=1,inplace=True)
```

In [10]:
```
df
```

Out[10]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hist
0	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	
1	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	
4	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	
...	...	...	...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	
610	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	
611	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	
612	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	
613	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	

614 rows × 12 columns

In [11]:
```
df['Dependents'].unique()
```

Out[11]:
```
array(['0', '1', '2', '3+', nan], dtype=object)
```

In [12]:
```
df['Dependents']=df['Dependents'].replace('3+','3')
```

In [13]:
```
df['Dependents'].unique()
```

Out[13]:
```
array(['0', '1', '2', '3', nan], dtype=object)
```

In [14]:
```
df.columns
```

Out[14]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [15]:
```
df.dtypes
```

Out[15]:
```
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

In [16]:
```
dtypes=dict(df.dtypes)
cat_vars=[i for i in dtypes if dtypes[i]=='object']
num_vars=[i for i in dtypes if dtypes[i]!='object']
```

In [17]:
```
print("No of Categorical vaiables:",len(cat_vars))
cat_vars
```

```
No of Categorical vaiables: 7
```

Out[17]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

In [18]:
```
print("No of Numerical vaiables:",len(num_vars))
num_vars
```

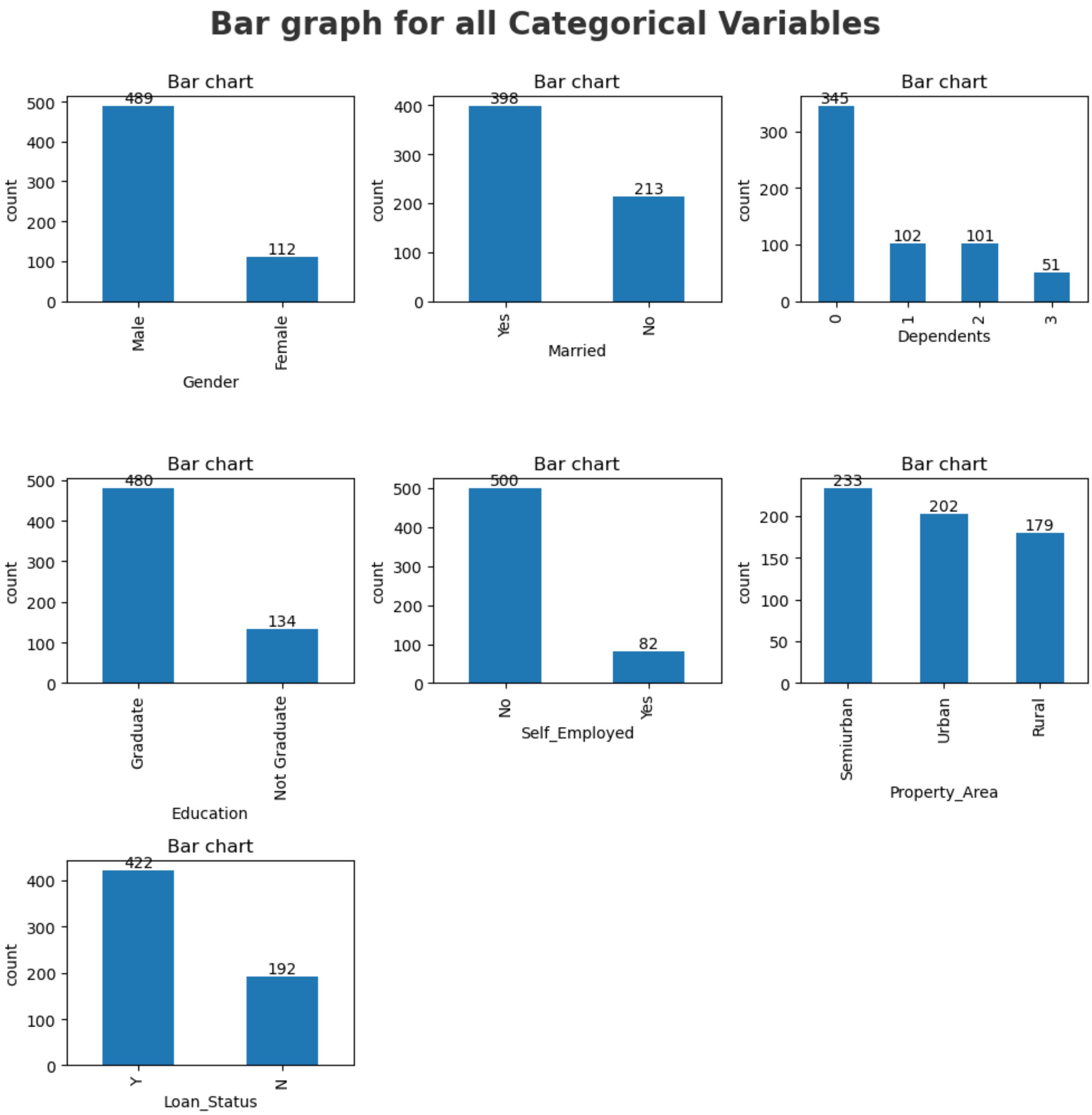
```
No of Numerical vaiables: 5
```

Out[18]:
```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

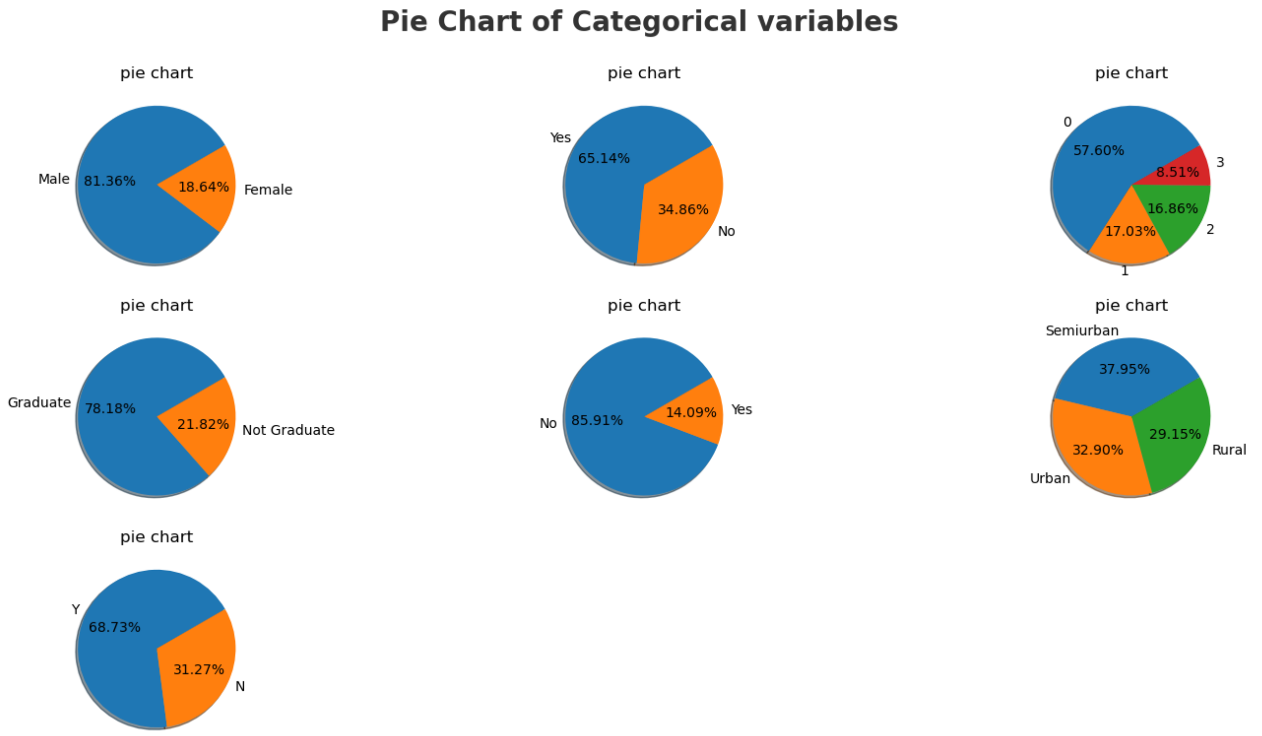
In [19]:
```
for i in cat_vars:
    print(i,df[i].unique(),df[i].nunique())
```

```
Gender ['Male' 'Female' nan] 2
Married ['No' 'Yes' nan] 2
Dependents ['0' '1' '2' '3' nan] 4
Education ['Graduate' 'Not Graduate'] 2
Self_Employed ['No' 'Yes' nan] 2
Property_Area ['Urban' 'Rural' 'Semiurban'] 3
Loan_Status ['Y' 'N'] 2
```

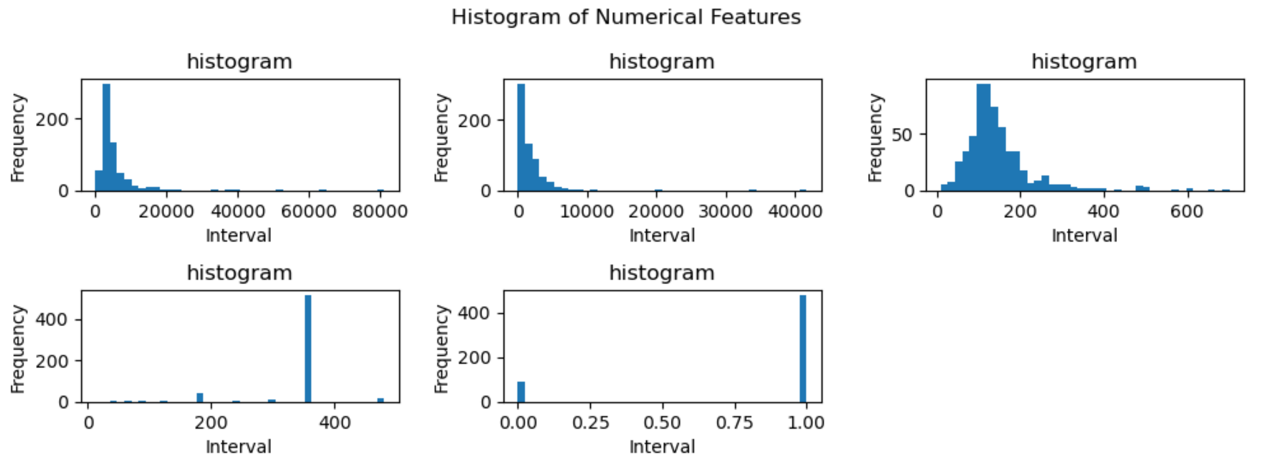
In [20]:
```
plt.figure(figsize=(10,10))
plt.suptitle("Bar graph for all Categorical Variables",fontsize=20, fontweight='bold', alpha=0.8, y=1)
for i in range(len(cat_vars)):
    plt.subplot(3,3,i+1)
    value=df[cat_vars[i]].value_counts()
    ax=value.plot(kind='bar')
    ax.bar_label(ax.containers[0])
    plt.title('Bar chart')
    plt.xlabel(cat_vars[i])
    plt.ylabel('count')
    plt.tight_layout()
```



In [21]:
```
plt.figure(figsize=(15,10))
plt.suptitle("Pie Chart of Categorical variables ",fontsize=20, fontweight='bold', alpha=0.8, y=1)
for i in range(len(cat_vars)):
    plt.subplot(4,3,i+1)
    values=df[cat_vars[i]].value_counts().values
    names=df[cat_vars[i]].value_counts().keys()
    plt.pie(x=values,labels=names,autopct="%0.2f%%",shadow=True,startangle=30)
    plt.title('pie chart')
    plt.tight_layout()
```



In [22]:
```
plt.figure(figsize=(10,5))
plt.suptitle("Histogram of Numerical Features")
for i in range(len(num_vars)):
    plt.subplot(3,3,i+1)
    data=df[num_vars[i]]
    plt.hist(data,bins=40)
    plt.title("histogram")
    plt.xlabel("Interval")
    plt.ylabel('Frequency')
    plt.tight_layout()
```



In [23]:
```
cat_vars
```

Out[23]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

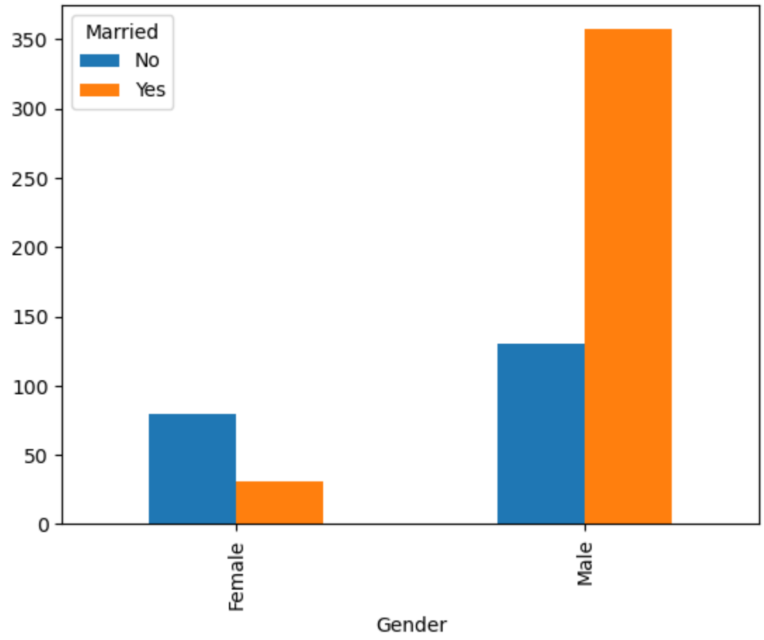
In [24]:
```
col1=df['Gender']
col2=df['Married']
result=pd.crosstab(col1,col2)
result
```

Out[24]:

Married	No	Yes
Gender		
Female	80	31
Male	130	357

In [25]:
```
result.plot(kind='bar')
```

Out[25]: <AxesSubplot:xlabel='Gender'>



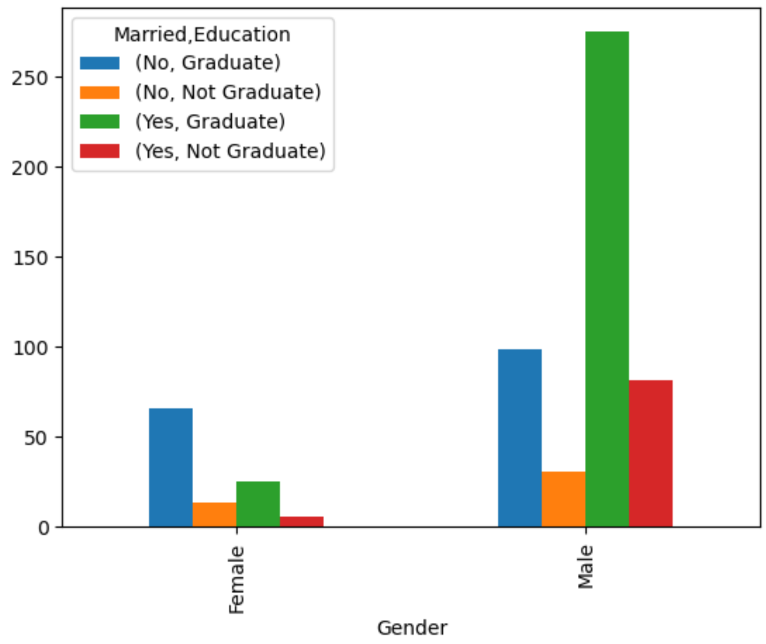
In [26]:
```
col1=df['Gender']
col2=df['Married']
col3=df['Education']
col=[col2,col3]
result1=pd.crosstab(col1,col)
result1
```

Out[26]:

Married	No		Yes	
Education	Graduate	Not Graduate	Graduate	Not Graduate
Gender				
Female	66	14	25	6
Male	99	31	275	82

In [27]:
```
result1.plot(kind='bar')
```

Out[27]: <AxesSubplot:xlabel='Gender'>

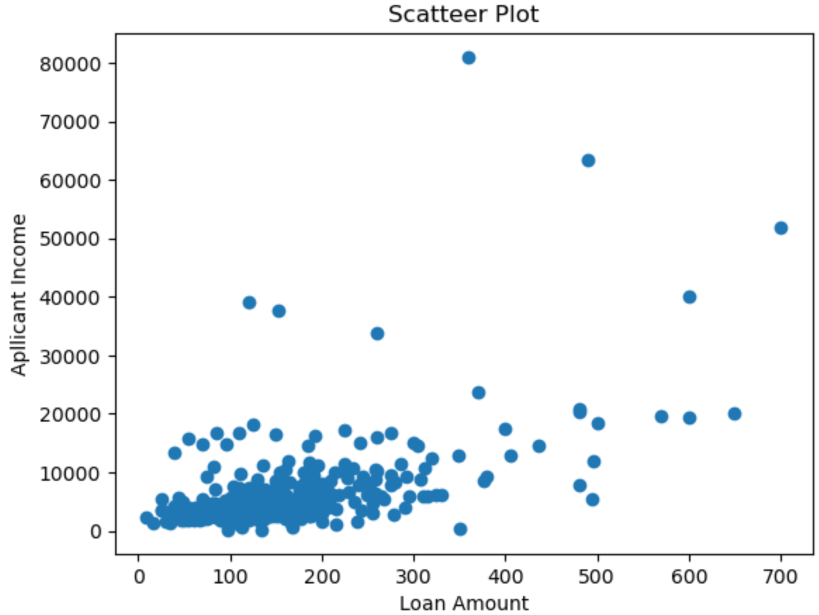


In [28]:
```
num_vars
```

Out[28]:
```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

In [29]:
```
col1=df['LoanAmount']
col2=df['ApplicantIncome']
plt.scatter(col1,col2)
plt.title("Scatteer Plot")
plt.xlabel("Loan Amount")
plt.ylabel("Apllicant Income")
```

Out[29]: Text(0, 0.5, 'Apllicant Income')



In [30]:
```
df.describe()
```

Out[30]:

	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 [31]:
```
df.isnull().sum()
```

Out[31]:
```
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 [32]:
```
check_missing=df.isnull().sum()*100/len(df)
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[32]:
```
Credit_History      8.143322
Self_Employed       5.211726
LoanAmount          3.583062
Dependents          2.442997
Loan_Amount_Term    2.280130
Gender              2.117264
Married             0.488599
dtype: float64
```

In [33]:
```
columns=['LoanAmount','Dependents','Loan_Amount_Term','Gender','Married']
```

In [34]:
```
df.dropna(subset=columns,inplace=True)
```

In [35]:
```
df.isnull().sum()
```

Out[35]:
```
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed        30
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term      0
Credit_History       48
Property_Area         0
Loan_Status           0
dtype: int64
```

In [36]:
```
df['Self_Employed'].unique()
```

Out[36]:
```
array(['No', 'Yes', nan], dtype=object)
```

In [37]:
```
df['Self_Employed'].mode()
```

Out[37]:
```
0    No
Name: Self_Employed, dtype: object
```

In [38]:
```
df['Self_Employed']=df['Self_Employed'].fillna('No')
```

In [39]:
```
df['Self_Employed'].unique()
```

Out[39]:
```
array(['No', 'Yes'], dtype=object)
```

In [40]:
```
df.isnull().sum()
```

Out[40]:
```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History      48
Property_Area        0
Loan_Status          0
dtype: int64
```

In [41]:
```
df['Credit_History'].unique()
```

Out[41]:
```
array([ 1.,  0., nan])
```

In [42]:
```
df['Credit_History'].mode()
```

Out[42]:
```
0    1.0
Name: Credit_History, dtype: float64
```

In [43]:
```
df['Credit_History']=df['Credit_History'].fillna(1.0)
```

In [44]:
```
df['Credit_History'].nunique()
```

Out[44]:
```
2
```

In [ ]:

In [45]:
```
df.isnull().sum()
```

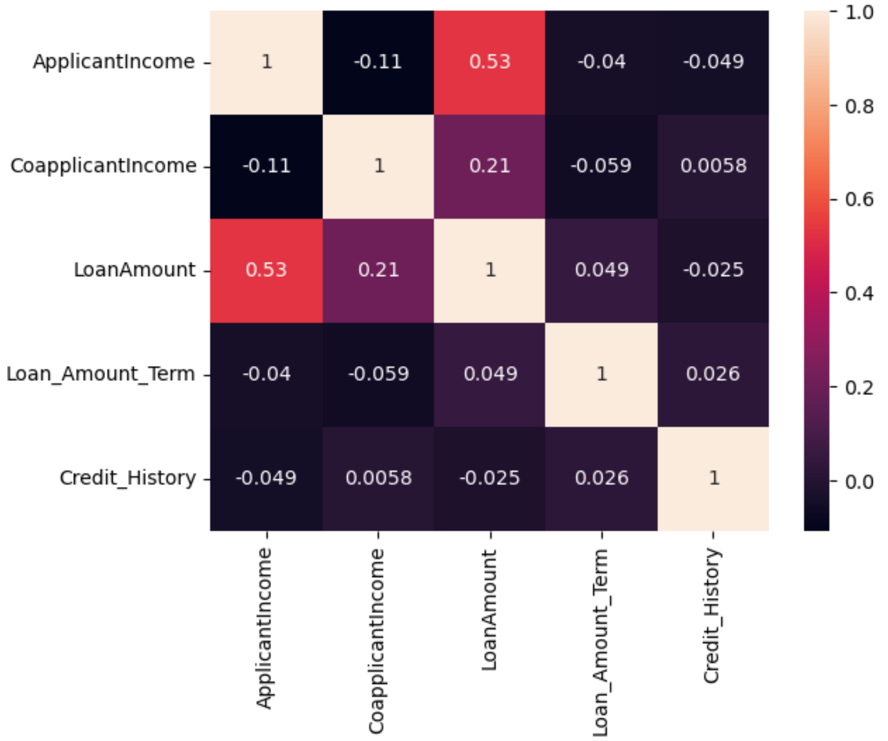
Out[45]:
```
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [46]:
```
df.corr()
```

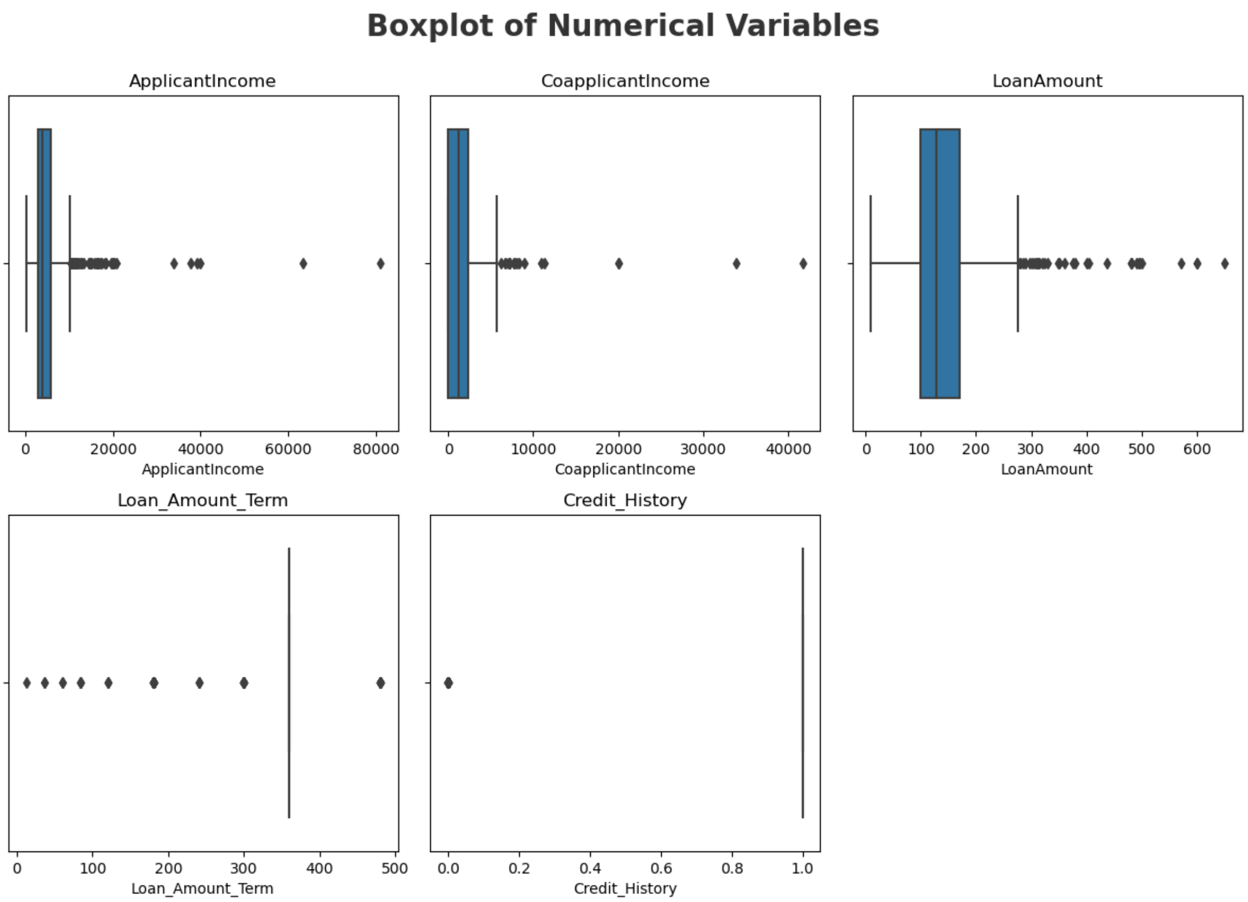
Out[46]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
ApplicantIncome	1.000000	-0.107597	0.529728	-0.040014	-0.049439
CoapplicantIncome	-0.107597	1.000000	0.205801	-0.059338	0.005814
LoanAmount	0.529728	0.205801	1.000000	0.049339	-0.025290
Loan_Amount_Term	-0.040014	-0.059338	0.049339	1.000000	0.026256
Credit_History	-0.049439	0.005814	-0.025290	0.026256	1.000000

In [47]:
```
sns.heatmap(df.corr(),annot=True)
```
Out[47]: <AxesSubplot:>



In [48]:
```
plt.figure(figsize=(12,12))
plt.suptitle("Boxplot of Numerical Variables",fontsize=20, fontweight='bold', alpha=0.8, y=1)
for i in range(len(num_vars)):
    plt.subplot(3,3,i+1)
    sns.boxplot(x=df[num_vars[i]])
    plt.title(num_vars[i])
    plt.tight_layout()
```



In [49]:
```
for i in range(len(num_vars)):
    q1 = np.percentile(df[num_vars[i]],25)
    q2 = np.percentile(df[num_vars[i]],50)
    q3 = np.percentile(df[num_vars[i]],75)
    iqr= q3-q1
    ub= q3+1.5*iqr
    lb= q1-1.5*iqr
    con1=df[num_vars[i]]>ub
    con2=df[num_vars[i]]<lb
    outliers=df[num_vars[i]][con1|con2].values
    print(num_vars[i],len(outliers))
```

```
ApplicantIncome 45
CoapplicantIncome 18
LoanAmount 33
Loan_Amount_Term 80
Credit_History 71
```

In [50]:
```
cat_vars
```

Out[50]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

In [51]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in cat_vars:
    df[i]=le.fit_transform(df[i])
```

In [52]:
```
df
```

Out[52]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hist
1	1	1	1	0	0	4583	1508.0	128.0	360.0	
2	1	1	0	0	1	3000	0.0	66.0	360.0	
3	1	1	0	1	0	2583	2358.0	120.0	360.0	
4	1	0	0	0	0	6000	0.0	141.0	360.0	
5	1	1	2	0	1	5417	4196.0	267.0	360.0	
...	...	...	...	...	...	...	...	...	...	
609	0	0	0	0	0	2900	0.0	71.0	360.0	
610	1	1	3	0	0	4106	0.0	40.0	180.0	
611	1	1	1	0	0	8072	240.0	253.0	360.0	
612	1	1	2	0	0	7583	0.0	187.0	360.0	
613	0	0	0	0	1	4583	0.0	133.0	360.0	

553 rows × 12 columns

In [53]:
```
df
```

Out[53]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hist
1	1	1	1	0	0	4583	1508.0	128.0	360.0	
2	1	1	0	0	1	3000	0.0	66.0	360.0	
3	1	1	0	1	0	2583	2358.0	120.0	360.0	
4	1	0	0	0	0	6000	0.0	141.0	360.0	
5	1	1	2	0	1	5417	4196.0	267.0	360.0	
...	...	...	...	...	...	...	...	...	...	
609	0	0	0	0	0	2900	0.0	71.0	360.0	
610	1	1	3	0	0	4106	0.0	40.0	180.0	
611	1	1	1	0	0	8072	240.0	253.0	360.0	
612	1	1	2	0	0	7583	0.0	187.0	360.0	
613	0	0	0	0	1	4583	0.0	133.0	360.0	

553 rows × 12 columns

In [54]:
```
X=df.drop('Loan_Status',axis=1)
y=df['Loan_Status']
```

In [55]:
```
df
```

Out[55]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hist
1	1	1	1	0	0	4583	1508.0	128.0	360.0	
2	1	1	0	0	1	3000	0.0	66.0	360.0	
3	1	1	0	1	0	2583	2358.0	120.0	360.0	
4	1	0	0	0	0	6000	0.0	141.0	360.0	
5	1	1	2	0	1	5417	4196.0	267.0	360.0	
...	...	...	...	...	...	...	...	...	...	
609	0	0	0	0	0	2900	0.0	71.0	360.0	
610	1	1	3	0	0	4106	0.0	40.0	180.0	
611	1	1	1	0	0	8072	240.0	253.0	360.0	
612	1	1	2	0	0	7583	0.0	187.0	360.0	
613	0	0	0	0	1	4583	0.0	133.0	360.0	

553 rows × 12 columns

In [56]:
```
X.columns
```

Out[56]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
```

In [57]:
```
cols = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term']
```

In [58]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
X[cols]=ss.fit_transform(X[cols])
```

In [59]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
for i in X.columns:
    df[i]=ss.fit_transform(df[[i]])
```

In [60]:
```
df
```

Out[60]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_
1	0.481275	0.735112	0.233308	-0.515215	-0.386896	-0.128694	-0.049699	-0.214368	0.279961	0
2	0.481275	0.735112	-0.759148	-0.515215	2.584677	-0.394296	-0.545638	-0.952675	0.279961	0
3	0.481275	0.735112	-0.759148	1.940938	-0.386896	-0.464262	0.229842	-0.309634	0.279961	0
4	0.481275	-1.360337	-0.759148	-0.515215	-0.386896	0.109057	-0.545638	-0.059562	0.279961	0
5	0.481275	0.735112	1.225764	-0.515215	2.584677	0.011239	0.834309	1.440866	0.279961	0
...	...	...	...	...	...	...	...	...	...	
609	-2.077813	-1.360337	-0.759148	-0.515215	-0.386896	-0.411075	-0.545638	-0.893134	0.279961	0
610	0.481275	0.735112	2.218219	-0.515215	-0.386896	-0.208727	-0.545638	-1.262287	-2.468292	0
611	0.481275	0.735112	0.233308	-0.515215	-0.386896	0.456706	-0.466709	1.274152	0.279961	0
612	0.481275	0.735112	1.225764	-0.515215	-0.386896	0.374659	-0.545638	0.488213	0.279961	0
613	-2.077813	-1.360337	-0.759148	-0.515215	2.584677	-0.128694	-0.545638	-0.154828	0.279961	-2

553 rows × 12 columns

In [61]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1234)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(442, 11)
(111, 11)
(442,)
(111,)
```

- Without Hyper parameter tuning

Decision Tree Classifier

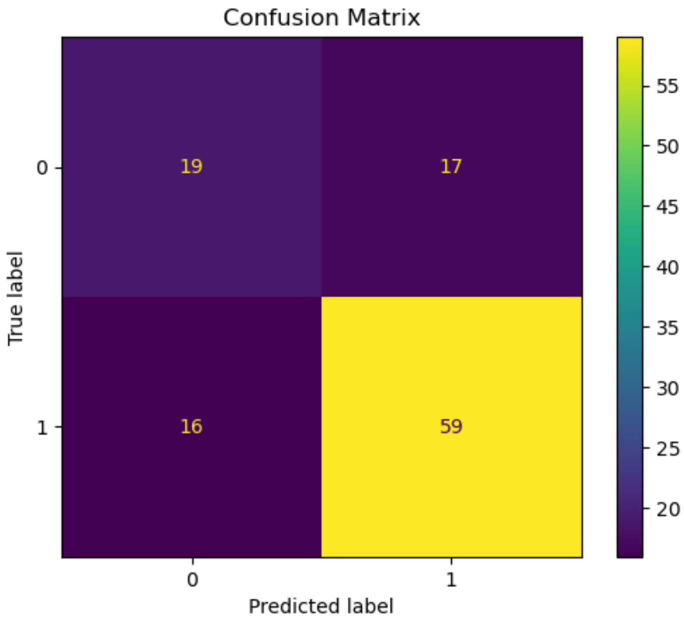
In [62]:
```
from sklearn.tree import DecisionTreeClassifier
dtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

Out[62]:
```
DecisionTreeClassifier()
```

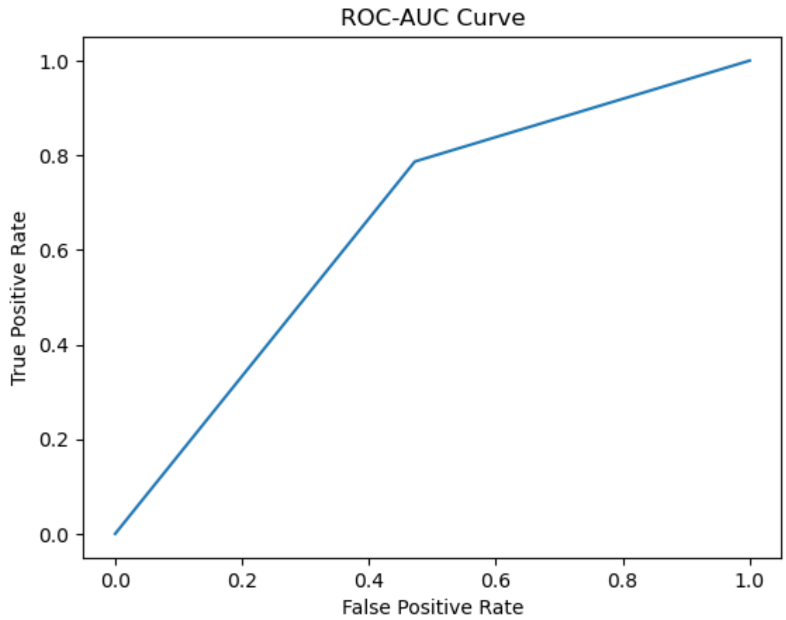
In [63]:
```
y_pred_dt=dtree.predict(X_test)
```

In [64]:
```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
acc_dt=round(accuracy_score(y_test,y_pred_dt)*100,2)
precision_dt=round(precision_score(y_test,y_pred_dt),2)
recall_dt=round(recall_score(y_test,y_pred_dt),2)
f1_dt=round(f1_score(y_test,y_pred_dt),2)
print("Accuracy",acc_dt)
print("precision",precision_dt)
print("Recall",recall_dt)
print("F1 Score",f1_dt)

from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cmt=confusion_matrix(y_test,y_pred_dt)
print(cmt)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title("Confusion Matrix")
plt.show()
tn, fp, fn, tp = confusion_matrix(y_test,y_pred_dt).ravel()
print("True Negative",tn)
print("False Positive",fp)
print("False Negative",fn)
print("True Positive",tp)

from sklearn.metrics import roc_curve,roc_auc_score
y_pred_proba_dt=dtree.predict_proba(X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,y_pred_proba_dt)
plt.plot(fpr,tpr)
plt.title("ROC-AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
Accuracy 70.27
precision 0.78
Recall 0.79
F1 Score 0.78
[[19 17]
 [16 59]]
```




```
True Negative 19
False Positive 17
False Negative 16
True Positive 59
```



Logistic Regression

In [65]:
```
from sklearn.linear_model import LogisticRegression
log_reg=LogisticRegression()
log_reg.fit(X_train,y_train)
```

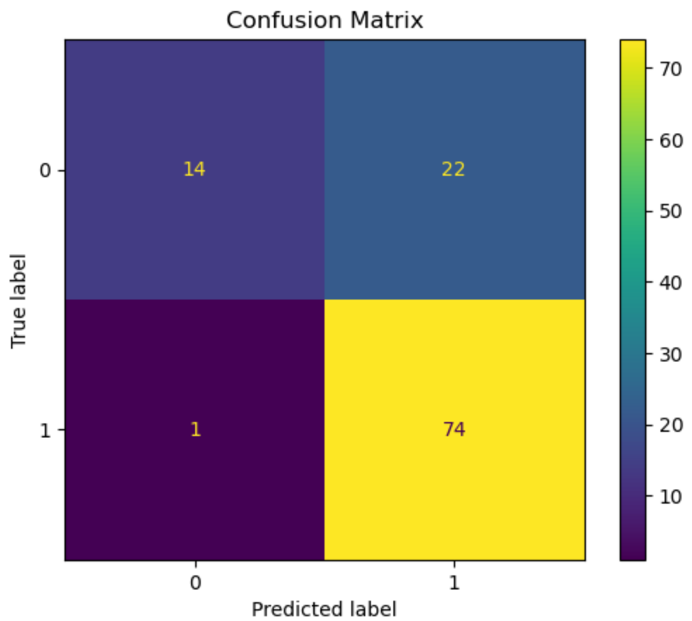
Out[65]:
```
LogisticRegression()
```

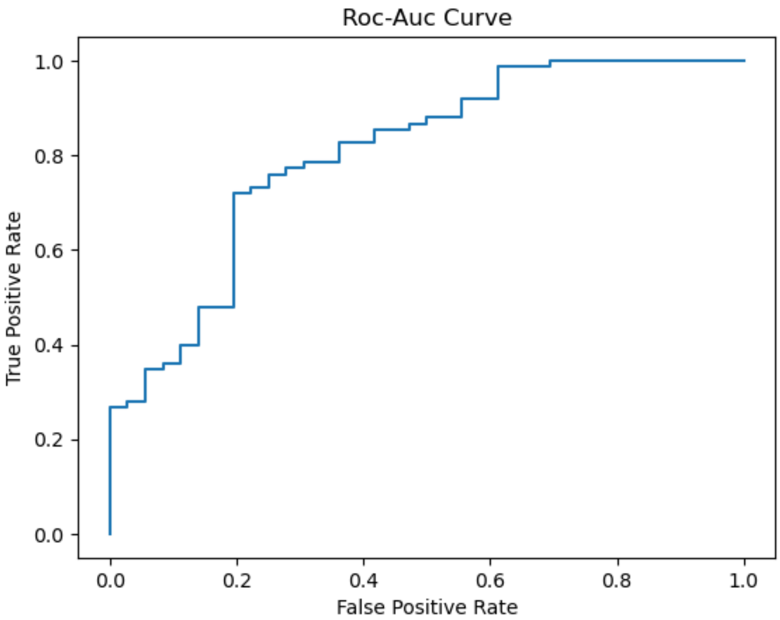
In [66]:
```
#predictions
y_pred_log=log_reg.predict(X_test)
y_pred_log
```

Out[66]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
       1])
```

In [67]:
```
#metrics
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,classification_report
acc_log=round(accuracy_score(y_test,y_pred_log)*100,2)
recall_log=round(recall_score(y_test,y_pred_log),2)
precision_log=round(precision_score(y_test,y_pred_log),2)
f1_log=round(f1_score(y_test,y_pred_log),2)
classification_report_log=classification_report(y_test,y_pred_log)


print("Accuaracy score",acc_log)
print("Precision Score",precision_log)
print("Recall Score",recall_log)
print("F1 Score",f1_log)
print(classification_report_log)

from sklearn.metrics import ConfusionMatrixDisplay,confusion_matrix
cmt=confusion_matrix(y_test,y_pred_log)
tn, fp, fn, tp=confusion_matrix(y_test,y_pred_log).ravel()
print("True Negative:",tn)
print("False Positive:",fp)
print("False Negative:",fn)
print("True Positive:",tp)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title("Confusion Matrix")
plt.show()

y_log_pred_prob=log_reg.predict_proba(X_test)[:,1]
y_log_pred_prob
fpr,tpr,threshold=roc_curve(y_test,y_log_pred_prob)
plt.plot(fpr,tpr)
plt.title("Roc-Auc Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
Accuaracy score 79.28
Precision Score 0.77
Recall Score 0.99
F1 Score 0.87
              precision    recall  f1-score   support

           0       0.93      0.39      0.55        36
           1       0.77      0.99      0.87        75

    accuracy                           0.79       111
   macro avg       0.85      0.69      0.71       111
weighted avg       0.82      0.79      0.76       111

True Negative: 14
False Positive: 22
False Negative: 1
True Positive: 74
```





Naive Bayes

In [68]:
```
from sklearn.naive_bayes import GaussianNB
NB=GaussianNB()
NB.fit(X_train,y_train)
```

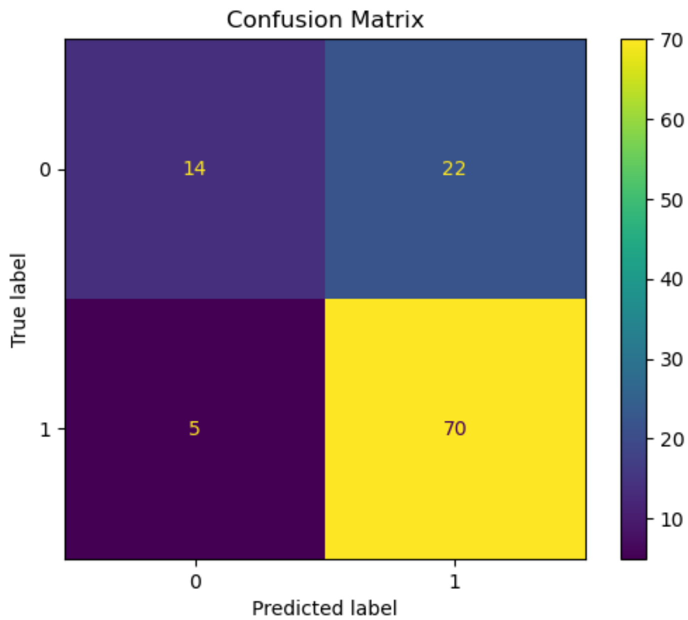
Out[68]:
```
GaussianNB()
```

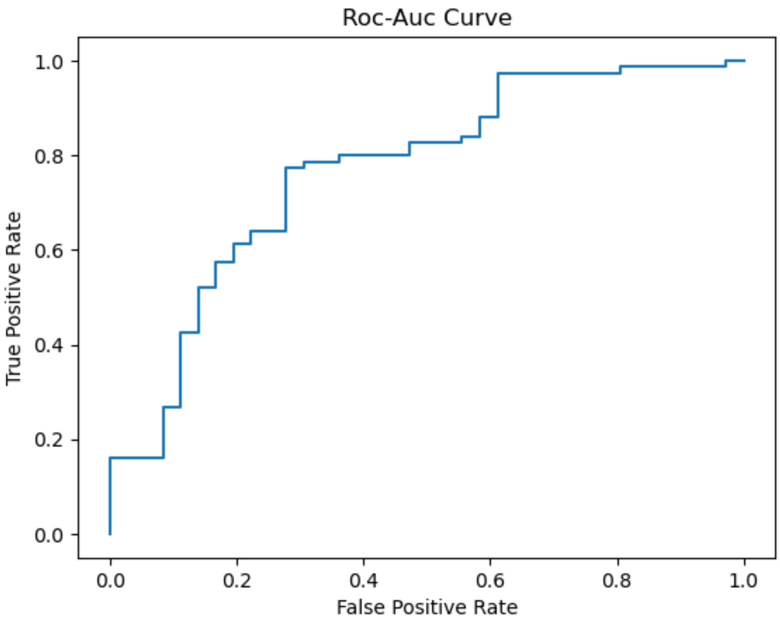
In [69]:
```
y_pred_NB=NB.predict(X_test)
y_pred_NB
```

Out[69]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       1])
```

In [70]:
```
#Metrics
from sklearn.metrics import accuracy_score,precision_score,f1_score,recall_score,classification_report
acc_NB=round(accuracy_score(y_test,y_pred_NB)*100,2)
precision_NB=round(precision_score(y_test,y_pred_NB),2)
recall_NB=round(recall_score(y_test,y_pred_NB),2)
f1_NB=round(f1_score(y_test,y_pred_NB),2)
classification_report_NB=classification_report(y_test,y_pred_NB)
print("'Metrics on Naive Bayes'")
print("Accuaracy score",acc_NB)
print("Precision Score",precision_NB)
print("Recall Score",recall_NB)
print("F1 Score",f1_NB)
print(classification_report_NB)
#######################################
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cmt=confusion_matrix(y_test,y_pred_NB)
tn, fp, fn, tp = confusion_matrix(y_test,y_pred_NB).ravel()
print("True Negative:",tn)
print("False Positive:",fp)
print("False Negative:",fn)
print("True Positive:",tp)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title("Confusion Matrix")
plt.show()
####################roc curve

y_pred_proba_NB=NB.predict_proba(X_test)[:,1]
y_pred_proba_NB
fpr,tpr,threshold=roc_curve(y_test,y_pred_proba_NB)
plt.plot(fpr,tpr)
plt.title("Roc-Auc Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
'Metrics on Naive Bayes'
Accuaracy score 75.68
Precision Score 0.76
Recall Score 0.93
F1 Score 0.84
              precision    recall  f1-score   support

           0       0.74      0.39      0.51        36
           1       0.76      0.93      0.84        75

    accuracy                           0.76       111
   macro avg       0.75      0.66      0.67       111
weighted avg       0.75      0.76      0.73       111

True Negative: 14
False Positive: 22
False Negative: 5
True Positive: 70
```





K-Nearesh Neighbor

In [71]:
```
from sklearn.neighbors import KNeighborsClassifier
KNN=KNeighborsClassifier()
KNN.fit(X_train,y_train)
```

Out[71]:
```
KNeighborsClassifier()
```

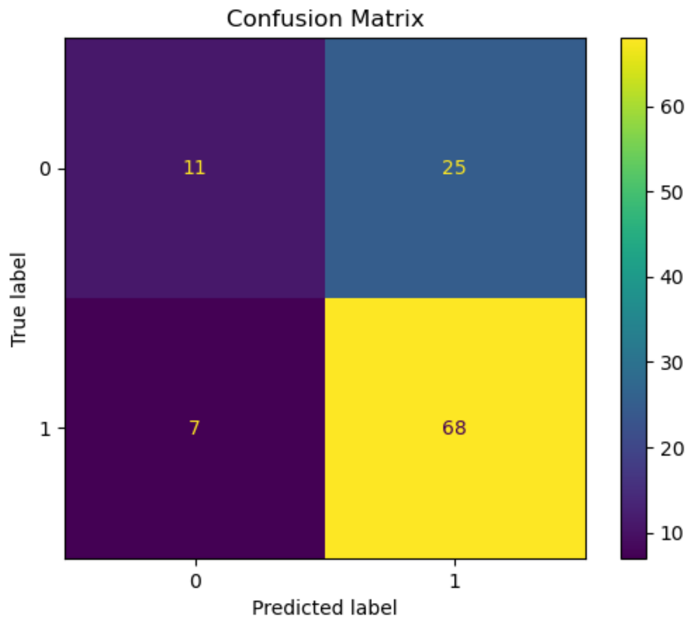
In [72]:
```
y_pred_knn=KNN.predict(X_test)
y_pred_knn
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other re
duction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts alo
ng. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over w
hich the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to Tru
e or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

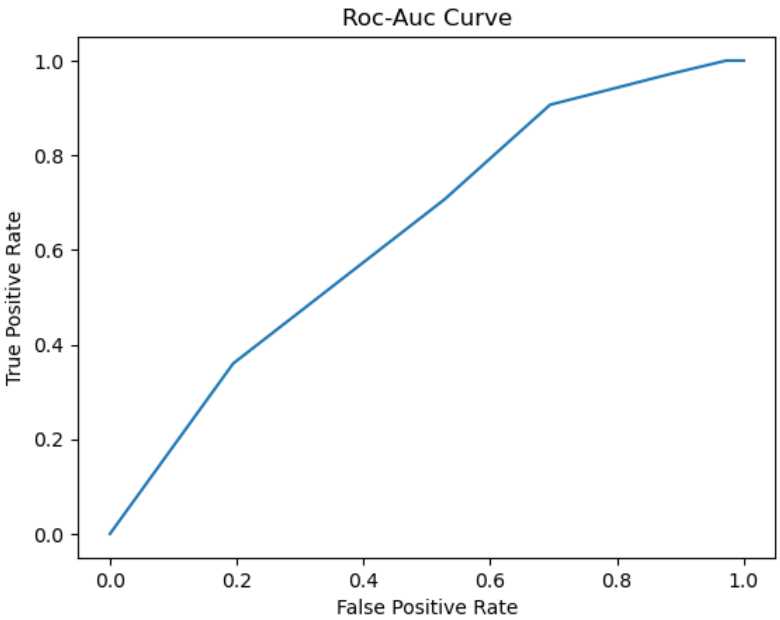
Out[72]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       1])
```

In [73]:
```
#metrics
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
acc_knn=round(accuracy_score(y_test,y_pred_knn)*100,2)
precision_knn=round(precision_score(y_test,y_pred_knn),2)
recall_knn=round(recall_score(y_test,y_pred_knn),2)
f1_knn=round(f1_score(y_test,y_pred_knn),2)
print("Accuracy:",acc_knn)
print("Precision Score:",precision_knn)
print("Recall Score:",recall_knn)
print("f1 Score:",f1_knn)
#consufion Matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cmt=confusion_matrix(y_test,y_pred_knn)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title('Confusion Matrix')
plt.show()
tn, fp, fn, tp = confusion_matrix(y_test,y_pred_knn).ravel()

print("True Negative:",tn)
print("False Positive:",fp)
print("False Negative:",fn)
print("True Positive:",tp)
#Roc Curve
y_pred_proba_knn=KNN.predict_proba(X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,y_pred_proba_knn)
plt.plot(fpr,tpr)
plt.title("Roc-Auc Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
Accuracy: 71.17
Precision Score: 0.73
Recall Score: 0.91
f1 Score: 0.81
```




```
True Negative: 11
False Positive: 25
False Negative: 7
True Positive: 68
```



Random Forest Classifier

In [74]:
```
from sklearn.ensemble import RandomForestClassifier
RFC=RandomForestClassifier()
RFC.fit(X_train,y_train)
```

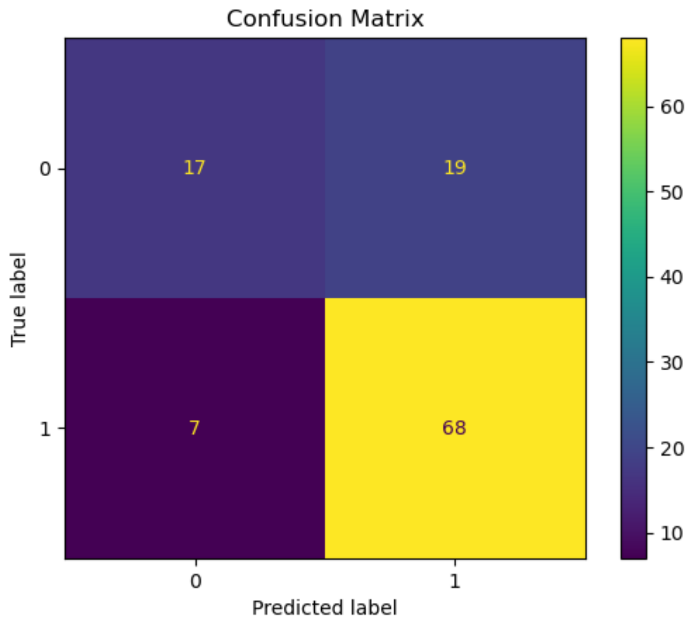
Out[74]:
```
RandomForestClassifier()
```

In [75]:
```
y_predict_rfc=RFC.predict(X_test)
y_predict_rfc
```

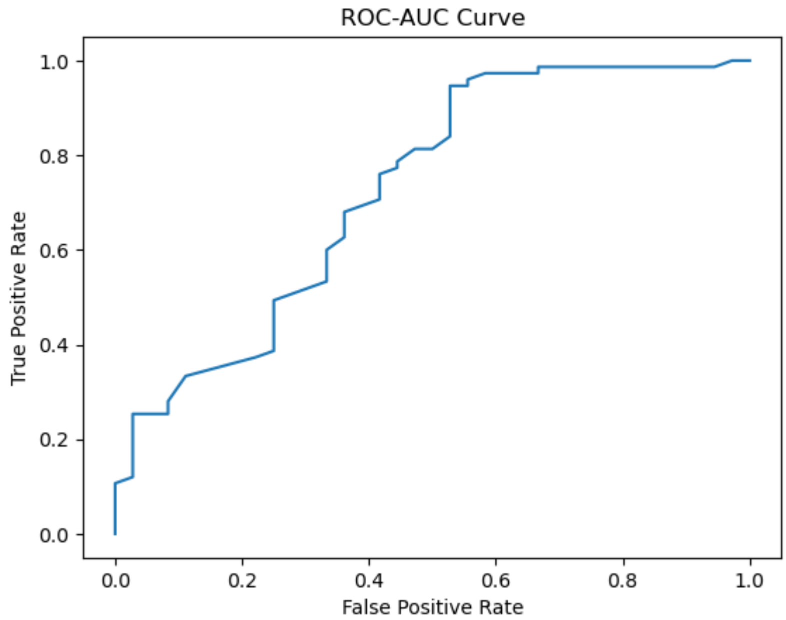
Out[75]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       1])
```

In [76]:
```
#Random Forest Classifier Metrics
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,classification_report
acc_rfc=round(accuracy_score(y_test,y_predict_rfc)*100,2)
precision_rfc=round(precision_score(y_test,y_predict_rfc),2)
recall_rfc=round(recall_score(y_test,y_predict_rfc),2)
f1_rfc=round(f1_score(y_test,y_predict_rfc),2)
print("Accuracy:",acc_rfc)
print("Precision Score:",precision_rfc)
print("Recall Score:",recall_rfc)
print("f1 Score:",f1_rfc)
##################
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cmt=confusion_matrix(y_test,y_predict_rfc)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title("Confusion Matrix")
plt.show()
tn, fp, fn, tp = confusion_matrix(y_test,y_pred_knn).ravel()

print("True Negative:",tn)
print("False Positive:",fp)
print("False Negative:",fn)
print("True Positive:",tp)

y_pred_proba_rfc=RFC.predict_proba(X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,y_pred_proba_rfc)
plt.plot(fpr,tpr)
plt.title("ROC-AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
Accuracy: 76.58
Precision Score: 0.78
Recall Score: 0.91
f1 Score: 0.84
```




```
True Negative: 11
False Positive: 25
False Negative: 7
True Positive: 68
```



In [77]:
```
print("Loan dataset Model Performances")
dict1={'Accuarcy':[acc_dt,acc_log,acc_NB,acc_knn,acc_rfc],
    'Precision Score':[precision_dt,precision_log,precision_NB,precision_knn,precision_rfc],
    'Recall Score':[recall_dt,recall_log,recall_NB,recall_knn,recall_rfc],
    'F1 Score':[f1_dt,f1_log,f1_NB,f1_knn,f1_rfc]}
pd.DataFrame(dict1,index=['Decision Tree','Logisitic Regression','Naive Bayes','K-Nearest Neighbors','Random Forest'])
```

Loan dataset Model Performances

Out[77]:

	Accuarcy	Precision Score	Recall Score	F1 Score
Decision Tree	70.27	0.78	0.79	0.78
Logisitic Regression	79.28	0.77	0.99	0.87
Naive Bayes	75.68	0.76	0.93	0.84
K-Nearest Neighbors	71.17	0.73	0.91	0.81
Random Forest	76.58	0.78	0.91	0.84

- With Hyper Parameter Tuning

Decision Tree

In [78]:
```
from sklearn.model_selection import GridSearchCV,cross_val_score
from sklearn.tree import DecisionTreeClassifier
grid_tree=DecisionTreeClassifier()#Base model
grid_tree
```

Out[78]: DecisionTreeClassifier()

In [79]:
```
grid_tree.get_params()
```

Out[79]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

In [80]:
```
param_grid = {
             'criterion':['gini','entropy'],
             'max_depth':[3,4,5,6,7,8],
             'min_samples_split':[2,3,4],
             'min_samples_leaf':[1,2,3,4],
             'random_state':[42,1234]
             }
#2*6*3*4*2=288
```

In [81]:
```
import time
start=time.time()
grid_search=GridSearchCV(grid_tree,param_grid,scoring='accuracy',cv=5,verbose=True)

end=time.time()
print("Total time taken is:",(end-start))
```

```
Total time taken is: 0.0
```

In [82]:
```
grid_search
```

Out[82]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 4, 5, 6, 7, 8],
                         'min_samples_leaf': [1, 2, 3, 4],
                         'min_samples_split': [2, 3, 4],
                         'random_state': [42, 1234]},
             scoring='accuracy', verbose=True)
```

In [83]:
```
import time
start=time.time()
grid_search.fit(X_train,y_train)
end=time.time()
print("Total time taken is:",(end-start))
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
Total time taken is: 21.368001222610474
```

In [84]:
```
grid_search.best_estimator_
```

Out[84]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
```

In [85]:
```
grid_search.best_params_
```

Out[85]:
```
{'criterion': 'entropy',
 'max_depth': 3,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'random_state': 42}
```

In [86]:
```
grid_search.best_score_
```

Out[86]:
```
0.8123340143003064
```

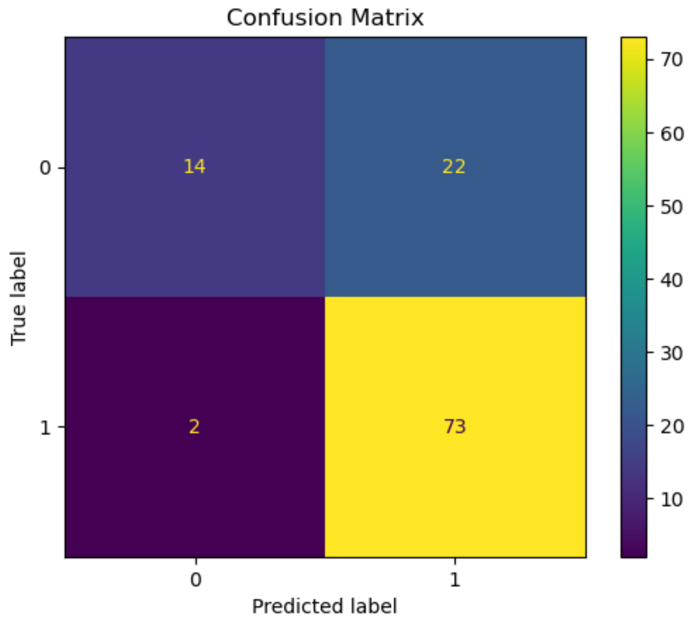
In [100]:
```
from sklearn.tree import DecisionTreeClassifier
dtree1=DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=1,min_samples_split=2,random_state=4
dtree1.fit(X_train,y_train)
```

Out[100]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
```

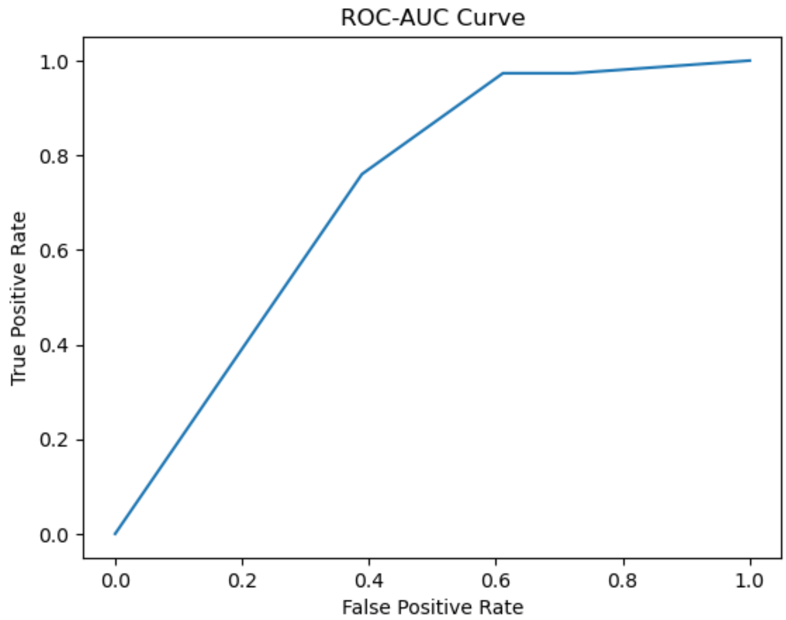
In [88]:
```
y_pred_dt1=dtree1.predict(X_test)
```

In [89]:
```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
acc_dt1=round(accuracy_score(y_test,y_pred_dt1)*100,2)
precision_dt1=round(precision_score(y_test,y_pred_dt1),2)
recall_dt1=round(recall_score(y_test,y_pred_dt1),2)
f1_dt1=round(f1_score(y_test,y_pred_dt1),2)
print("Accuracy",acc_dt1)
print("precision",precision_dt1)
print("Recall",recall_dt1)
print("F1 Score",f1_dt1)

from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cmt=confusion_matrix(y_test,y_pred_dt1)
print(cmt)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title("Confusion Matrix")
plt.show()
tn, fp, fn, tp = confusion_matrix(y_test,y_pred_dt1).ravel()
print("True Negative",tn)
print("False Positive",fp)
print("False Negative",fn)
print("True Positive",tp)

from sklearn.metrics import roc_curve,roc_auc_score
y_pred_proba_dt=dtree1.predict_proba(X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,y_pred_proba_dt)
plt.plot(fpr,tpr)
plt.title("ROC-AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
Accuracy 78.38
precision 0.77
Recall 0.97
F1 Score 0.86
[[14 22]
 [ 2 73]]
```




```
True Negative 14
False Positive 22
False Negative 2
True Positive 73
```



In [90]:
```
dtree1.feature_importances_
```
Out[90]:
```
array([0.        , 0.        , 0.        , 0.        , 0.0284887 ,
       0.        , 0.06999259, 0.066493  , 0.        , 0.77739157,
       0.05763415])
```

Random Forest Classifier

In [91]:
```
from sklearn.ensemble import RandomForestClassifier
grid_RFC=RandomForestClassifier()
grid_RFC
```
Out[91]:
```
RandomForestClassifier()
```

In [92]:
```
grid_RFC.get_params()
```
Out[92]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [93]:
```
param_grid={'n_estimators':[100,200],
           'criterion':['gini','entropy'],
           'max_depth':[3,5,10],
           'max_features':['sqrt','log2'],
           'random_state':[0,42]}
```

In [94]:
```
from sklearn.model_selection import GridSearchCV,cross_val_score
import time
start=time.time()
grid_search=GridSearchCV(grid_RFC,param_grid,scoring='accuracy',cv=5,verbose=True)

end=time.time()
print("Total time taken is:",(end-start))
```
```
Total time taken is: 0.0010025501251220703
```

In [95]:
```
grid_search
```

Out[95]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 10],
                         'max_features': ['sqrt', 'log2'],
                         'n_estimators': [100, 200], 'random_state': [0, 42]},
             scoring='accuracy', verbose=True)
```

In [96]:
```
import time
start=time.time()
grid_search.fit(X_train,y_train)
end=time.time()
print("Total time taken is:",(end-start))
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Total time taken is: 160.29201412200928
```

In [97]:
```
grid_search.best_estimator_
```

Out[97]:
```
RandomForestClassifier(criterion='entropy', max_depth=10, max_features='sqrt',
                       random_state=0)
```

In [98]:
```
grid_search.best_params_
```

Out[98]:
```
{'criterion': 'entropy',
 'max_depth': 10,
 'max_features': 'sqrt',
 'n_estimators': 100,
 'random_state': 0}
```

In [101]:
```
from sklearn.ensemble import RandomForestClassifier
RFC1=RandomForestClassifier(criterion='entropy',max_depth=10,max_features='sqrt',n_estimators=100,random_state=0)
RFC1.fit(X_train,y_train)
```

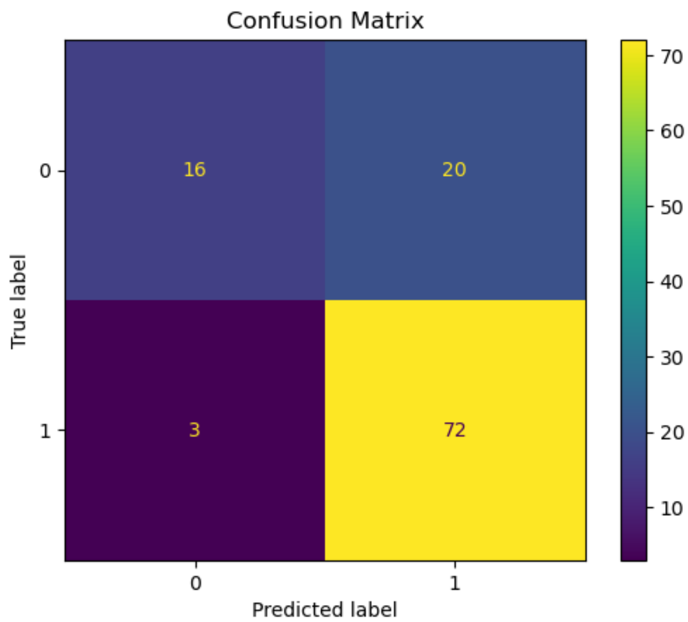
Out[101]:
```
RandomForestClassifier(criterion='entropy', max_depth=10, max_features='sqrt',
                       random_state=0)
```

In [102]:
```
y_predict_rfc1=RFC1.predict(X_test)
y_predict_rfc1
```

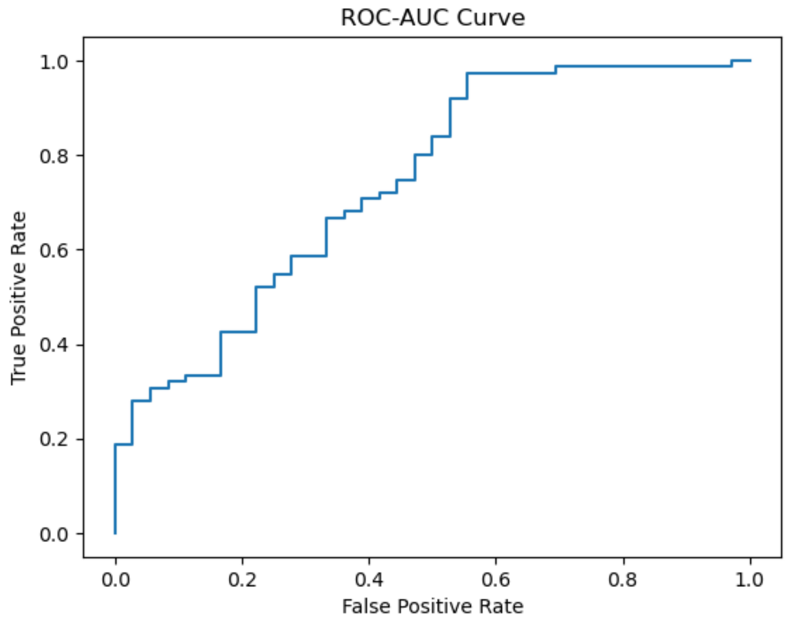
Out[102]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       1])
```

In [104]:
```
#Random Forest Classifier Metrics
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,classification_report
acc_rfc1=round(accuracy_score(y_test,y_predict_rfc1)*100,2)
precision_rfc1=round(precision_score(y_test,y_predict_rfc1),2)
recall_rfc1=round(recall_score(y_test,y_predict_rfc1),2)
f1_rfc1=round(f1_score(y_test,y_predict_rfc1),2)
print("Accuracy:",acc_rfc1)
print("Precision Score:",precision_rfc1)
print("Recall Score:",recall_rfc1)
print("f1 Score:",f1_rfc1)
##################
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cmt=confusion_matrix(y_test,y_predict_rfc1)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title("Confusion Matrix")
plt.show()
tn, fp, fn, tp = confusion_matrix(y_test,y_predict_rfc1).ravel()

print("True Negative:",tn)
print("False Positive:",fp)
print("False Negative:",fn)
print("True Positive:",tp)

y_pred_proba_rfc=RFC1.predict_proba(X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,y_pred_proba_rfc)
plt.plot(fpr,tpr)
plt.title("ROC-AUC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
Accuracy: 79.28
Precision Score: 0.78
Recall Score: 0.96
f1 Score: 0.86
```




```
True Negative: 16
False Positive: 20
False Negative: 3
True Positive: 72
```



Logistic Regression

In [105]:
```
from sklearn.linear_model import LogisticRegression
grid_log=LogisticRegression()
grid_log
```
Out[105]:
```
LogisticRegression()
```

In [107]:
```
grid_log.get_params()
```
Out[107]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [ ]:
```
param_grid={'n_estimators':[100,200],
           'criterion':['gini','entropy'],
           'max_depth':[3,5,10],
           'max_features':['sqrt','log2'],
           'random_state':[0,42]}
```

In [ ]:
```

solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}
```

In [111]:
```
param_grid={'n_jobs':[5,10],
           'max_iter':[100,200],
           'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
           'random_state':[0,42]}
```

In [112]:
```
from sklearn.model_selection import GridSearchCV,cross_val_score
import time
start=time.time()
grid_search=GridSearchCV(grid_log,param_grid,scoring='accuracy',cv=5,verbose=True)

end=time.time()
print("Total time taken is:",(end-start))
```
```
Total time taken is: 0.0
```

In [113]:
```python
import time
start=time.time()
grid_search.fit(X_train,y_train)
end=time.time()
print("Total time taken is:",(end-start))
```

```
  warnings.warn(
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1523: UserWarning: 'n_jobs' > 1 does
not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 10.
  warnings.warn(
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1523: UserWarning: 'n_jobs' > 1 does
not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 10.
  warnings.warn(
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1523: UserWarning: 'n_jobs' > 1 does
not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 10.
  warnings.warn(
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1523: UserWarning: 'n_jobs' > 1 does
not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 10.
  warnings.warn(
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1523: UserWarning: 'n_jobs' > 1 does
not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 10.
  warnings.warn(
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:1523: UserWarning: 'n_jobs' > 1 does
not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 10.
  warnings.warn(

Total time taken is: 56.407296805080835
```

In [115]:
```python
grid_search.best_params_
```

Out[115]:
```
{'max_iter': 100, 'n_jobs': 5, 'random_state': 0, 'solver': 'newton-cg'}
```

In [116]:
```python
from sklearn.linear_model import LogisticRegression
log_reg1=LogisticRegression(max_iter=100,n_jobs=5,random_state=0,solver='newton-cg')
log_reg1.fit(X_train,y_train)
```

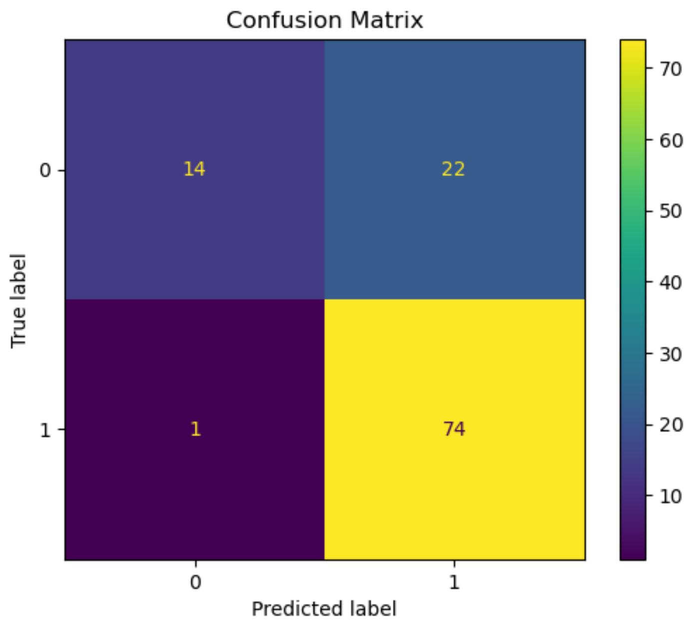
Out[116]:
```
LogisticRegression(n_jobs=5, random_state=0, solver='newton-cg')
```

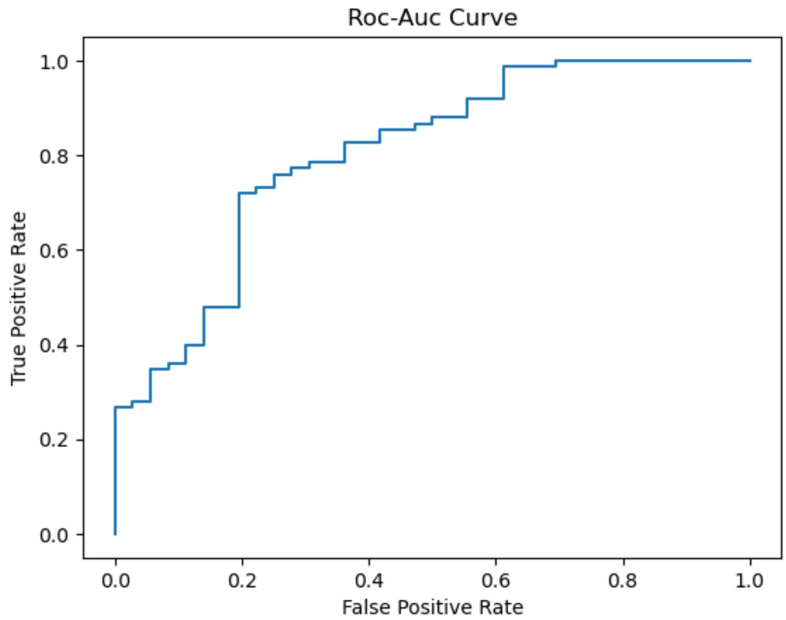
In [118]:
```python
#predictions
y_pred_log1=log_reg1.predict(X_test)
y_pred_log1
```

Out[118]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
       1])
```

In [119]:
```
#metrics
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,classification_report
acc_log1=round(accuracy_score(y_test,y_pred_log1)*100,2)
recall_log1=round(recall_score(y_test,y_pred_log1),2)
precision_log1=round(precision_score(y_test,y_pred_log1),2)
f1_log1=round(f1_score(y_test,y_pred_log1),2)
classification_report_log1=classification_report(y_test,y_pred_log1)


print("Accuaracy score",acc_log1)
print("Precision Score",precision_log1)
print("Recall Score",recall_log1)
print("F1 Score",f1_log1)
print(classification_report_log1)

from sklearn.metrics import ConfusionMatrixDisplay,confusion_matrix
cmt=confusion_matrix(y_test,y_pred_log1)
tn, fp, fn, tp=confusion_matrix(y_test,y_pred_log1).ravel()
print("True Negative:",tn)
print("False Positive:",fp)
print("False Negative:",fn)
print("True Positive:",tp)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title("Confusion Matrix")
plt.show()

y_log_pred_prob=log_reg1.predict_proba(X_test)[:,1]
y_log_pred_prob
fpr,tpr,threshold=roc_curve(y_test,y_log_pred_prob)
plt.plot(fpr,tpr)
plt.title("Roc-Auc Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
Accuaracy score 79.28
Precision Score 0.77
Recall Score 0.99
F1 Score 0.87
              precision    recall  f1-score   support

           0       0.93      0.39      0.55        36
           1       0.77      0.99      0.87        75

    accuracy                           0.79       111
   macro avg       0.85      0.69      0.71       111
weighted avg       0.82      0.79      0.76       111

True Negative: 14
False Positive: 22
False Negative: 1
True Positive: 74
```





Naive Bayes

In [120]:
```
from sklearn.naive_bayes import GaussianNB
grid_NB=GaussianNB()
grid_NB
```
Out[120]:
```
GaussianNB()
```

In [121]:
```
grid_NB.get_params()
```
Out[121]:
```
{'priors': None, 'var_smoothing': 1e-09}
```

In [122]:
```
param_grid={'var_smoothing': np.logspace(0,-9, num=100)}
```

In [123]:
```
from sklearn.model_selection import GridSearchCV,cross_val_score
import time
start=time.time()
grid_search=GridSearchCV(grid_NB,param_grid,scoring='accuracy',cv=5,verbose=True)

end=time.time()
print("Total time taken is:",(end-start))
```
```
Total time taken is: 0.000999450683593 75
```

In [124]:
```
import time
start=time.time()
grid_search.fit(X_train,y_train)
end=time.time()
print("Total time taken is:",(end-start))
```
```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Total time taken is: 5.081666946411133
```

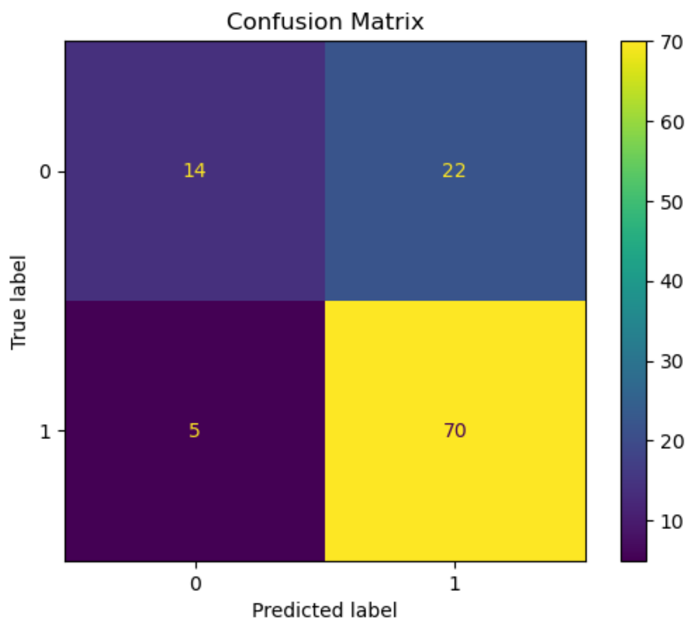
In [125]:
```
grid_search.best_params_
```
Out[125]:
```
{'var_smoothing': 0.08111308307896872}
```

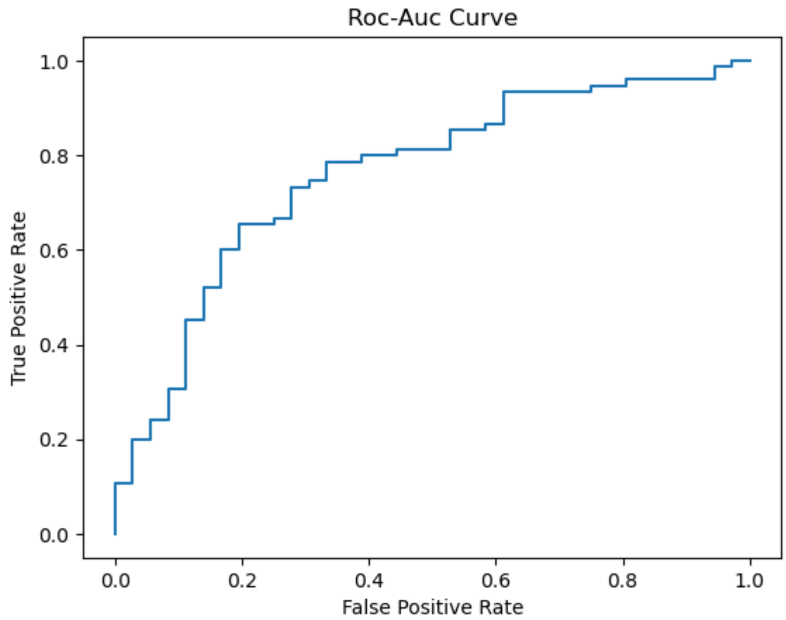
In [126]:
```
from sklearn.naive_bayes import GaussianNB
NB1=GaussianNB(var_smoothing=0.08111308307896872)
NB1.fit(X_train,y_train)
```
Out[126]:
```
GaussianNB(var_smoothing=0.08111308307896872)
```

In [144]:
```
y_pred_NB1=NB1.predict(X_test)
y_pred_NB1
```
Out[144]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       1])
```

In [145]:
```
#Metrics
from sklearn.metrics import accuracy_score,precision_score,f1_score,recall_score,classification_report
acc_NB1=round(accuracy_score(y_test,y_pred_NB1)*100,2)
precision_NB1=round(precision_score(y_test,y_pred_NB1),2)
recall_NB1=round(recall_score(y_test,y_pred_NB1),2)
f1_NB1=round(f1_score(y_test,y_pred_NB1),2)
classification_report_NB1=classification_report(y_test,y_pred_NB1)
print("'Metrics on Naive Bayes'")
print("Accuaracy score",acc_NB1)
print("Precision Score",precision_NB1)
print("Recall Score",recall_NB1)
print("F1 Score",f1_NB1)
print(classification_report_NB1)
#######################################
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cmt=confusion_matrix(y_test,y_pred_NB1)
tn, fp, fn, tp = confusion_matrix(y_test,y_pred_NB1).ravel()
print("True Negative:",tn)
print("False Positive:",fp)
print("False Negative:",fn)
print("True Positive:",tp)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title("Confusion Matrix")
plt.show()
####################roc curve

y_pred_proba_NB=NB1.predict_proba(X_test)[:,1]
y_pred_proba_NB
fpr,tpr,threshold=roc_curve(y_test,y_pred_proba_NB)
plt.plot(fpr,tpr)
plt.title("Roc-Auc Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
'Metrics on Naive Bayes'
Accuaracy score 75.68
Precision Score 0.76
Recall Score 0.93
F1 Score 0.84
              precision    recall  f1-score   support

           0       0.74      0.39      0.51        36
           1       0.76      0.93      0.84        75

    accuracy                           0.76       111
   macro avg       0.75      0.66      0.67       111
weighted avg       0.75      0.76      0.73       111

True Negative: 14
False Positive: 22
False Negative: 5
True Positive: 70
```





K-Nearesh Neighbor

In [129]:
```
from sklearn.neighbors import KNeighborsClassifier
grid_knn=KNeighborsClassifier()
grid_knn
```

Out[129]:
```
KNeighborsClassifier()
```

In [130]:
```
grid_knn.get_params()
```

Out[130]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

In [132]:
```
param_grid={'weights':['uniform', 'distance'],
           'algorithm':['auto', 'ball_tree', 'kd_tree', 'brute'],
           'p':[1,2,3],
           'n_neighbors':[4,5,6],
           'n_jobs':[5,10],
           'leaf_size':[20,30]}
```

In [133]:
```
from sklearn.model_selection import GridSearchCV,cross_val_score
import time
start=time.time()
grid_search=GridSearchCV(grid_knn,param_grid,scoring='accuracy',cv=5,verbose=True)

end=time.time()
print("Total time taken is:",(end-start))
```

```
Total time taken is: 0.0
```

In [134]:
```
import time
start=time.time()
grid_search.fit(X_train,y_train)
end=time.time()
print("Total time taken is:",(end-start))
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts
along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` o
ver which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims`
to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts
along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` o
ver which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims`
to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts
along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` o
ver which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims`
to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other
```

In [137]:
```
grid_search.best_params_
```

Out[137]:
```
{'algorithm': 'auto',
 'leaf_size': 20,
 'n_jobs': 5,
 'n_neighbors': 6,
 'p': 1,
 'weights': 'distance'}
```

In [139]:
```
from sklearn.neighbors import KNeighborsClassifier
KNN1=KNeighborsClassifier(algorithm='auto',leaf_size=20,n_jobs=5,n_neighbors=6,p=1,weights='distance')
KNN1.fit(X_train,y_train)
```

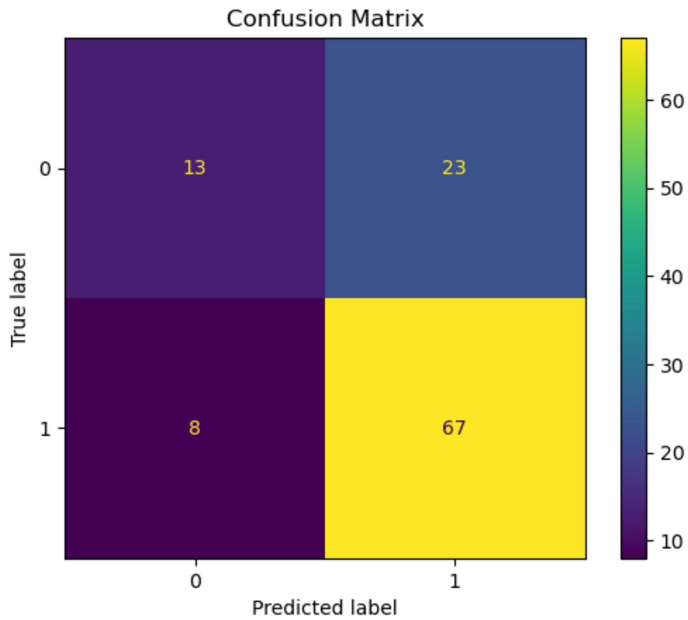
Out[139]:
```
KNeighborsClassifier(leaf_size=20, n_jobs=5, n_neighbors=6, p=1,
                     weights='distance')
```

In [140]:
```
y_pred_knn1=KNN1.predict(X_test)
y_pred_knn1
```

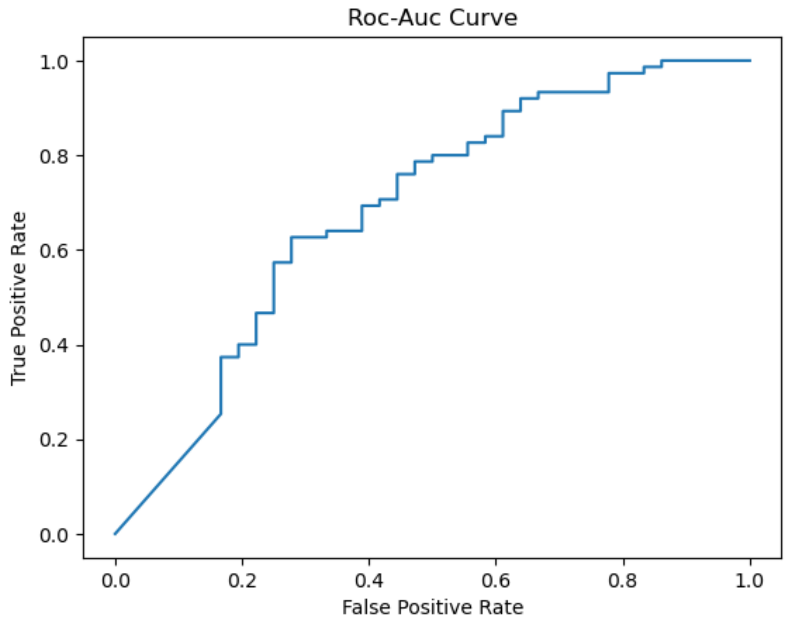
Out[140]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1,
       1])
```

In [141]:
```
#metrics
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
acc_knn1=round(accuracy_score(y_test,y_pred_knn1)*100,2)
precision_knn1=round(precision_score(y_test,y_pred_knn1),2)
recall_knn1=round(recall_score(y_test,y_pred_knn1),2)
f1_knn1=round(f1_score(y_test,y_pred_knn1),2)
print("Accuracy:",acc_knn1)
print("Precision Score:",precision_knn1)
print("Recall Score:",recall_knn1)
print("f1 Score:",f1_knn1)
#consufion Matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cmt=confusion_matrix(y_test,y_pred_knn1)
ConfusionMatrixDisplay(cmt).plot()
plt.grid(False)
plt.title('Confusion Matrix')
plt.show()
tn, fp, fn, tp = confusion_matrix(y_test,y_pred_knn).ravel()

print("True Negative:",tn)
print("False Positive:",fp)
print("False Negative:",fn)
print("True Positive:",tp)
#Roc Curve
y_pred_proba_knn=KNN1.predict_proba(X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,y_pred_proba_knn)
plt.plot(fpr,tpr)
plt.title("Roc-Auc Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
Accuracy: 72.07
Precision Score: 0.74
Recall Score: 0.89
f1 Score: 0.81
```




```
True Negative: 11
False Positive: 25
False Negative: 7
True Positive: 68
```



In [142]:
```
print("Loan dataset Model Performances")
dict1={'Accuarcy':[acc_dt,acc_log,acc_NB,acc_knn,acc_rfc],
    'Precision Score':[precision_dt,precision_log,precision_NB,precision_knn,precision_rfc],
    'Recall Score':[recall_dt,recall_log,recall_NB,recall_knn,recall_rfc],
    'F1 Score':[f1_dt,f1_log,f1_NB,f1_knn,f1_rfc]}
pd.DataFrame(dict1,index=['Decision Tree','Logisitic Regression','Naive Bayes','K-Nearest Neighbors','Random Forest']
```

```
Loan dataset Model Performances
```

Out[142]:

	Accuarcy	Precision Score	Recall Score	F1 Score
Decision Tree	70.27	0.78	0.79	0.78
Logisitic Regression	79.28	0.77	0.99	0.87
Naive Bayes	75.68	0.76	0.93	0.84
K-Nearest Neighbors	71.17	0.73	0.91	0.81
Random Forest	76.58	0.78	0.91	0.84

In [146]:
```
print("Loan dataset Model Performances with Hyper parameter tuning")
dict1={'Accuarcy':[acc_dt1,acc_log1,acc_NB1,acc_knn1,acc_rfc1],
    'Precision Score':[precision_dt1,precision_log1,precision_NB1,precision_knn1,precision_rfc1],
    'Recall Score':[recall_dt1,recall_log1,recall_NB1,recall_knn1,recall_rfc1],
    'F1 Score':[f1_dt1,f1_log1,f1_NB1,f1_knn1,f1_rfc1]}
pd.DataFrame(dict1,index=['Decision Tree','Logisitic Regression','Naive Bayes','K-Nearest Neighbors','Random Forest']
```

```
Loan dataset Model Performances with Hyper parameter tuning
```

Out[146]:

	Accuarcy	Precision Score	Recall Score	F1 Score
Decision Tree	78.38	0.77	0.97	0.86
Logisitic Regression	79.28	0.77	0.99	0.87
Naive Bayes	75.68	0.76	0.93	0.84
K-Nearest Neighbors	72.07	0.74	0.89	0.81
Random Forest	79.28	0.78	0.96	0.86

In [147]:
```
RFC1
```

Out[147]:
```
RandomForestClassifier(criterion='entropy', max_depth=10, max_features='sqrt',
                       random_state=0)
```

In [149]:
```
import pickle
```

In [150]:
```
pickle.dump(RFC,open('loan_classifier_model.pkl','wb'))
```

In [ ]: