In [ ]:
```
## P.S.1 = build the predictive model which can predict if loan has to be approved o
# Target variable = loan status

# P.S.2 = Run a campign which can target the good customer & offer them some new loa
```

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

In [2]:
```
cr = pd.read_csv(r"D:\python data set lec\CreditRisk.csv")
```

In [3]:
```
cr.head(10)
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplican
0	LP001002	Male	No	0.0	Graduate	No	5849	
1	LP001003	Male	Yes	1.0	Graduate	No	4583	
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	
4	LP001008	Male	No	0.0	Graduate	No	6000	
5	LP001011	Male	Yes	2.0	Graduate	Yes	5417	
6	LP001013	Male	Yes	0.0	Not Graduate	No	2333	
7	LP001014	Male	Yes	4.0	Graduate	No	3036	
8	LP001018	Male	Yes	2.0	Graduate	No	4006	
9	LP001020	Male	Yes	1.0	Graduate	No	12841	

In [4]:
```
cr.shape
```

Out[4]:
```
(981, 13)
```

Find Null Value

In [5]:
```
cr.isnull().sum()
```

Out[5]:
```
Loan_ID              0
Gender              24
Married              3
Dependents          25
Education            0
Self_Employed       55
ApplicantIncome      0
CoapplicantIncome    0
```

```
LoanAmount           27
Loan_Amount_Term     20
Credit_History       79
Property_Area         0
Loan_Status           0
dtype: int64
```

In [6]:
```
cr.isnull().sum() [ cr.isnull().sum() * 100 / cr.shape[0] > 0 ]
```

Out[6]:
```
Gender              24
Married              3
Dependents          25
Self_Employed       55
LoanAmount          27
Loan_Amount_Term    20
Credit_History      79
dtype: int64
```

Fill null value

In [7]:
```
cr.Gender              = cr.Gender.fillna("Male")
cr.Married             = cr.Married.fillna("No")
cr.Dependents          = cr.Dependents.fillna(0)
cr.Self_Employed       = cr.Self_Employed.fillna("Yes")
cr.LoanAmount          = cr.LoanAmount.fillna(cr.LoanAmount.mean())
cr.Loan_Amount_Term    = cr.Loan_Amount_Term.fillna(cr.LoanAmount.mean())
cr.Credit_History      = cr.Credit_History.fillna(1)
```

In [8]:
```
cr.isnull().sum()
```

Out[8]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

labele encoder for object to numeric

In [9]:
```
### Labelencoder

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [10]:
```
cr.Loan_Status    = le.fit_transform(cr.Loan_Status)
cr.Gender         = le.fit_transform(cr.Gender)
cr.Married        = le.fit_transform(cr.Married)
cr.Education      = le.fit_transform(cr.Education)
cr.Self_Employed  = le.fit_transform(cr.Self_Employed)
cr.Property_Area  = le.fit_transform(cr.Property_Area)
```

In [11]:
```
cr.head(10)
```

Out[11]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplican
0	LP001002	1	0	0.0	0	0	5849	
1	LP001003	1	1	1.0	0	0	4583	
2	LP001005	1	1	0.0	0	1	3000	
3	LP001006	1	1	0.0	1	0	2583	
4	LP001008	1	0	0.0	0	0	6000	
5	LP001011	1	1	2.0	0	1	5417	
6	LP001013	1	1	0.0	1	0	2333	
7	LP001014	1	1	4.0	0	0	3036	
8	LP001018	1	1	2.0	0	0	4006	
9	LP001020	1	1	1.0	0	0	12841	

In [12]:
```
cr1=cr   ## duplicate
```

In [13]:
```
cr = cr.iloc[: , 1::]   ## remove loan id
```

In [14]:
```
cr.head(10)
```

Out[14]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	L
0	1	0	0.0	0	0	5849	0.0	
1	1	1	1.0	0	0	4583	1508.0	
2	1	1	0.0	0	1	3000	0.0	
3	1	1	0.0	1	0	2583	2358.0	
4	1	0	0.0	0	0	6000	0.0	
5	1	1	2.0	0	1	5417	4196.0	
6	1	1	0.0	1	0	2333	1516.0	
7	1	1	4.0	0	0	3036	2504.0	
8	1	1	2.0	0	0	4006	1526.0	
9	1	1	1.0	0	0	12841	10968.0	

Sampling

In [15]:
```
from sklearn.model_selection import train_test_split
```

In [16]:
```
cr_train , cr_test = train_test_split(cr, test_size = .2)
```

In [17]:
```
print(cr.shape)
print(cr_train.shape)
print(cr_test.shape)
```

```
(981, 12)
(784, 12)
(197, 12)
```

In [18]:
```
cr_train_x = cr_train.iloc[:, 0:-1]
cr_train_y = cr_train.iloc[:, -1]
```

In [19]:
```
cr_train_x.head(5)
```

Out[19]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
379	1	1	2.0	0	0	5391	0.0
559	0	1	0.0	0	0	4180	2306.0
15	1	0	0.0	0	0	4950	0.0
172	1	1	4.0	1	0	3522	0.0
195	1	1	1.0	0	0	3125	2583.0

In [20]:
```
cr_train_y.head(5)
```

Out[20]:
```
379    1
559    1
15     1
172    0
195    0
Name: Loan_Status, dtype: int32
```

In [21]:
```
cr_test_x = cr_test.iloc[:, 0:-1]
cr_test_y = cr_test.iloc[:, -1]
```

In [22]:
```
cr_test_x.head(5)
```

Out[22]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
874	1	1	4.0	1	0	2792	2619.0
10	1	1	2.0	0	0	3200	700.0
468	0	1	2.0	1	1	210	2917.0
883	1	0	0.0	0	0	3508	0.0
595	1	0	0.0	1	0	3833	0.0

In [23]:

```
cr_test_y.head(5)
```

Out[23]:
```
874    1
10     1
468    1
883    0
595    1
Name: Loan_Status, dtype: int32
```

In [24]:
```
print(cr_train_x.shape)
print(cr_train_y.shape)
print("----")
print(cr_test_x.shape)
print(cr_test_y.shape)
```

```
(784, 11)
(784,)
----
(197, 11)
(197,)
```

Logistic model building

In [25]:
```
from sklearn.linear_model import LogisticRegression
```

In [26]:
```
logreg_cr = LogisticRegression()
```

In [27]:
```
logreg_cr.fit(cr_train_x , cr_train_y)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

Out[27]:
```
LogisticRegression()
```

In [28]:
```
pred_cr = logreg_cr.predict(cr_test_x)    ## predicted values for test data
```

In [29]:
```
pred_cr
```

Out[29]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [30]:
```
len(pred_cr)   # same as test data
```

Out[30]:
```
197
```

In [31]:
```
cr_test_y   # actual values for test data
```

Out[31]:
```
874    1
10     1
468    1
883    0
595    1
      ..
976    1
587    1
678    1
372    1
352    1
Name: Loan_Status, Length: 197, dtype: int32
```

In [32]:
```
from sklearn.metrics import confusion_matrix
```

In [33]:
```
# tab_cr = confusion_matrix( predicted , actual )
## 1st = predicted & 2nd = matrix

tab_cr = confusion_matrix( pred_cr , cr_test_y )
```

In [34]:
```
tab_cr                      # 25 = TN / 3 = FN / 30 = FP / 139 = TP
```

Out[34]:
```
array([[ 28,   2],
       [ 24, 143]], dtype=int64)
```

In [35]:
```
tab_cr.diagonal()
```

Out[35]:
```
array([ 28, 143], dtype=int64)
```

In [36]:
```
tab_cr.diagonal().sum()
```

Out[36]:
```
171
```

In [37]:
```
tab_cr.sum()
```

Out[37]:
```
197
```

Find Accuracy

In [38]:
```
acc = tab_cr.diagonal().sum() *100 / tab_cr.sum()
acc
```

Out[38]:
```
86.80203045685279
```

In [39]:
```
logreg_cr.coef_     ## slope / coef for each x variable
```

Out[39]:
```
array([[ 2.07350406e-01,  4.59122813e-01, -9.94063801e-02,
        -5.51942791e-01,  1.00523358e-02,  1.70300110e-05,
```

```
       -5.03942673e-05, -4.45970537e-03, -3.60957915e-03,
        3.03687396e+00,  1.42917098e-01]])
```

In [40]:
```
logreg_cr.intercept_     ## intercept
```

Out[40]:
```
array([0.1133581])
```

In [41]:
```
# We did prediction in categories but we can do prediction in probabilities also
```

In [42]:
```
pred_prob_train = logreg_cr.predict_proba(cr_train_x)
pred_prob_test  = logreg_cr.predict_proba(cr_test_x)
```

In [43]:
```
pred_prob_test                          # for each records you get 2 probabilites
```

Out[43]:
```
array([[0.29659585, 0.70340415],
       [0.09018986, 0.90981014],
       [0.24385758, 0.75614242],
       [0.15757748, 0.84242252],
       [0.25333581, 0.74666419],
       [0.14190589, 0.85809411],
       [0.13361175, 0.86638825],
       [0.15046765, 0.84953235],
       [0.59915537, 0.40084463],
       [0.19417271, 0.80582729],
       [0.20063266, 0.79936734],
       [0.81890415, 0.18109585],
       [0.17493231, 0.82506769],
       [0.11780944, 0.88219056],
       [0.14543545, 0.85456455],
       [0.11309248, 0.88690752],
       [0.07685056, 0.92314944],
       [0.33967966, 0.66032034],
       [0.8101003 , 0.1898997 ],
       [0.16864592, 0.83135408],
       [0.35767958, 0.64232042],
       [0.1224648 , 0.8775352 ],
       [0.12413768, 0.87586232],
       [0.08645273, 0.91354727],
       [0.18493738, 0.81506262],
       [0.12618063, 0.87381937],
       [0.2050285 , 0.7949715 ],
       [0.78906251, 0.21093749],
       [0.16159004, 0.83840996],
       [0.12000522, 0.87999478],
       [0.12854912, 0.87145088],
       [0.49513542, 0.50486458],
       [0.18149931, 0.81850069],
       [0.2327771 , 0.7672229 ],
       [0.33152035, 0.66847965],
       [0.11021471, 0.88978529],
       [0.1917096 , 0.8082904 ],
       [0.11405506, 0.88594494],
       [0.12766979, 0.87233021],
       [0.10838873, 0.89161127],
       [0.07792529, 0.92207471],
       [0.41441017, 0.58558983],
       [0.13145099, 0.86854901],
       [0.23580013, 0.76419987],
       [0.05025431, 0.94974569],
       [0.16199807, 0.83800193],
       [0.21843955, 0.78156045],
       [0.753819  , 0.246181  ],
       [0.13246553, 0.86753447],
```

```
[0.04467329, 0.95532671],
[0.15539785, 0.84460215],
[0.21657395, 0.78342605],
[0.16611302, 0.83388698],
[0.11579075, 0.88420925],
[0.10838891, 0.89161109],
[0.21743739, 0.78256261],
[0.20970627, 0.79029373],
[0.10919909, 0.89080091],
[0.10966484, 0.89033516],
[0.09338797, 0.90661203],
[0.15411468, 0.84588532],
[0.9235393 , 0.0764607 ],
[0.13149858, 0.86850142],
[0.22441638, 0.77558362],
[0.18944682, 0.81055318],
[0.9160118 , 0.0839882 ],
[0.08803785, 0.91196215],
[0.09468018, 0.90531982],
[0.94522757, 0.05477243],
[0.17840349, 0.82159651],
[0.13451059, 0.86548941],
[0.85926027, 0.14073973],
[0.15172247, 0.84827753],
[0.24851907, 0.75148093],
[0.14010063, 0.85989937],
[0.07417392, 0.92582608],
[0.23888496, 0.76111504],
[0.84436982, 0.15563018],
[0.19859481, 0.80140519],
[0.1382951 , 0.8617049 ],
[0.29827711, 0.70172289],
[0.19103613, 0.80896387],
[0.89160986, 0.10839014],
[0.08657092, 0.91342908],
[0.14718536, 0.85281464],
[0.15081308, 0.84918692],
[0.21028134, 0.78971866],
[0.07650016, 0.92349984],
[0.09612478, 0.90387522],
[0.11149356, 0.88850644],
[0.20348697, 0.79651303],
[0.06450085, 0.93549915],
[0.23314065, 0.76685935],
[0.12417833, 0.87582167],
[0.11080225, 0.88919775],
[0.64848059, 0.35151941],
[0.84651823, 0.15348177],
[0.11844424, 0.88155576],
[0.17712403, 0.82287597],
[0.09336997, 0.90663003],
[0.14357518, 0.85642482],
[0.12345533, 0.87654467],
[0.1175811 , 0.8824189 ],
[0.76321181, 0.23678819],
[0.05839272, 0.94160728],
[0.10334831, 0.89665169],
[0.23698993, 0.76301007],
[0.26169645, 0.73830355],
[0.24288608, 0.75711392],
[0.17440709, 0.82559291],
[0.08997951, 0.91002049],
[0.21666004, 0.78333996],
[0.23857229, 0.76142771],
[0.06345576, 0.93654424],
[0.74667163, 0.25332837],
[0.11325126, 0.88674874],
[0.12326587, 0.87673413],
[0.25136795, 0.74863205],
```

```
[0.24777775, 0.75222225],
[0.06970169, 0.93029831],
[0.15389906, 0.84610094],
[0.22044327, 0.77955673],
[0.10236717, 0.89763283],
[0.2567137 , 0.7432863 ],
[0.6993631 , 0.3006369 ],
[0.31130075, 0.68869925],
[0.19028828, 0.80971172],
[0.87842669, 0.12157331],
[0.15116012, 0.84883988],
[0.13890261, 0.86109739],
[0.34882657, 0.65117343],
[0.21332873, 0.78667127],
[0.12585408, 0.87414592],
[0.1813428 , 0.8186572 ],
[0.15510307, 0.84489693],
[0.21938412, 0.78061588],
[0.14023956, 0.85976044],
[0.13590584, 0.86409416],
[0.80487994, 0.19512006],
[0.79239681, 0.20760319],
[0.16877387, 0.83122613],
[0.08618426, 0.91381574],
[0.49981369, 0.50018631],
[0.15706285, 0.84293715],
[0.13763342, 0.86236658],
[0.03636492, 0.96363508],
[0.1650978 , 0.8349022 ],
[0.25379351, 0.74620649],
[0.81281556, 0.18718444],
[0.80973265, 0.19026735],
[0.82465552, 0.17534448],
[0.186831  , 0.813169  ],
[0.16700276, 0.83299724],
[0.22377744, 0.77622256],
[0.12255001, 0.87744999],
[0.13050902, 0.86949098],
[0.12749676, 0.87250324],
[0.08963159, 0.91036841],
[0.65693061, 0.34306939],
[0.09542058, 0.90457942],
[0.15436724, 0.84563276],
[0.77308876, 0.22691124],
[0.74997243, 0.25002757],
[0.15858931, 0.84141069],
[0.88658764, 0.11341236],
[0.75828606, 0.24171394],
[0.15144305, 0.84855695],
[0.04508773, 0.95491227],
[0.18629104, 0.81370896],
[0.15999375, 0.84000625],
[0.17029591, 0.82970409],
[0.8870584 , 0.1129416 ],
[0.11418447, 0.88581553],
[0.19575635, 0.80424365],
[0.13017052, 0.86982948],
[0.12501082, 0.87498918],
[0.18712519, 0.81287481],
[0.12281237, 0.87718763],
[0.24747214, 0.75252786],
[0.12770878, 0.87229122],
[0.12118784, 0.87881216],
[0.12055604, 0.87944396],
[0.18805037, 0.81194963],
[0.11737701, 0.88262299],
[0.10428612, 0.89571388],
[0.82172248, 0.17827752],
[0.34897122, 0.65102878],
```

```
       [0.11908901, 0.88091099],
       [0.84917205, 0.15082795],
       [0.26507634, 0.73492366],
       [0.12958624, 0.87041376],
       [0.17869115, 0.82130885],
       [0.20778137, 0.79221863],
       [0.23749754, 0.76250246],
       [0.20221232, 0.79778768],
       [0.35151197, 0.64848803],
       [0.16295687, 0.83704313]])
```

In [44]:
```
len(pred_prob_test)
```

Out[44]: 197

In [45]:
```
type(pred_prob_test)
```

Out[45]: numpy.ndarray

In [46]:
```
pred_prob_test = pd.DataFrame(pred_prob_test)
```

In [47]:
```
pred_prob_test
```

Out[47]:

	0	1
0	0.296596	0.703404
1	0.090190	0.909810
2	0.243858	0.756142
3	0.157577	0.842423
4	0.253336	0.746664
...	...	...
192	0.207781	0.792219
193	0.237498	0.762502
194	0.202212	0.797788
195	0.351512	0.648488
196	0.162957	0.837043

197 rows × 2 columns

In [48]:
```
pred_prob_test.rename(columns ={pred_prob_test.columns[0]: "Pred_prob" ,
                                pred_prob_test.columns[1]: "Pred_prob1"} , inplace =
```

In [49]:
```
pred_prob_test
```

Out[49]:

	Pred_prob	Pred_prob1
0	0.296596	0.703404
1	0.090190	0.909810

	Pred_prob	Pred_prob1
2	0.243858	0.756142
3	0.157577	0.842423
4	0.253336	0.746664
...	...	...
192	0.207781	0.792219
193	0.237498	0.762502
194	0.202212	0.797788
195	0.351512	0.648488
196	0.162957	0.837043

197 rows × 2 columns

In [50]:
```
# AUROC curve
```

In [51]:
```
from sklearn.metrics import  roc_curve , roc_auc_score
```

In [52]:
```
fpr , tpr , thershold = roc_curve(cr_test_y , pred_prob_test.iloc[: , 1] )
```

In [53]:
```
fpr
```

Out[53]:
```
array([0.        , 0.        , 0.        , 0.01923077, 0.01923077,
       0.03846154, 0.03846154, 0.05769231, 0.05769231, 0.07692308,
       0.07692308, 0.09615385, 0.09615385, 0.11538462, 0.11538462,
       0.13461538, 0.13461538, 0.17307692, 0.17307692, 0.19230769,
       0.19230769, 0.21153846, 0.21153846, 0.23076923, 0.23076923,
       0.25      , 0.25      , 0.26923077, 0.26923077, 0.30769231,
       0.30769231, 0.32692308, 0.32692308, 0.34615385, 0.34615385,
       0.36538462, 0.36538462, 0.40384615, 0.40384615, 0.42307692,
       0.42307692, 0.44230769, 0.44230769, 0.82692308, 0.82692308,
       0.84615385, 0.84615385, 1.        ])
```

In [54]:
```
tpr
```

Out[54]:
```
array([0.        , 0.00689655, 0.07586207, 0.07586207, 0.13103448,
       0.13103448, 0.15172414, 0.15172414, 0.26206897, 0.26206897,
       0.30344828, 0.30344828, 0.34482759, 0.34482759, 0.42758621,
       0.42758621, 0.45517241, 0.45517241, 0.52413793, 0.52413793,
       0.53793103, 0.53793103, 0.54482759, 0.54482759, 0.55862069,
       0.55862069, 0.63448276, 0.63448276, 0.66206897, 0.66206897,
       0.72413793, 0.72413793, 0.83448276, 0.83448276, 0.85517241,
       0.85517241, 0.95862069, 0.95862069, 0.96551724, 0.96551724,
       0.97241379, 0.97241379, 0.9862069 , 0.9862069 , 0.99310345,
       0.99310345, 1.        , 1.        ])
```

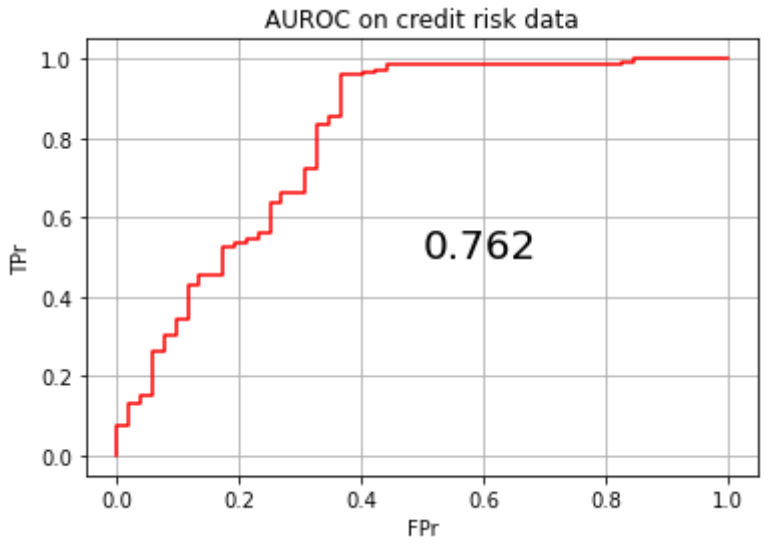
In [55]:
```
area_auroc = roc_auc_score(cr_test_y , pred_cr)
area_auroc
```

Out[55]:
```
0.7623342175066313
```

In [56]:
```
area_auroc = np.round(area_auroc , 3)
area_auroc
```

Out[56]: 0.762

In [57]:
```
plt.plot(fpr , tpr , color = "r")
plt.xlabel("FPr")
plt.ylabel("TPr")
plt.title("AUROC on credit risk data")
plt.text(x = 0.5 , y = 0.5, s = area_auroc , size = 20)
plt.grid()
```



In [58]:
```
from sklearn.metrics import accuracy_score , f1_score , precision_score , recall_sco
```

In [59]:
```
accuracy_score(pred_cr , cr_test_y)      ## prdicted then actual
```

Out[59]: 0.868020304568528

In [60]:
```
precision_score( cr_test_y , pred_cr)    ## actual then predicted
```

Out[60]: 0.8562874251497006

In [61]:
```
f1_score(cr_test_y , pred_cr)            ## actual then predicted
```

Out[61]: 0.9166666666666666

In [62]:
```
recall_score(cr_test_y , pred_cr)        ## actual then predicted
```

Out[62]: 0.9862068965517241

In [ ]:

campgin will run on entire data not on train or test data

In [63]:
```
pred_full_data = logreg_cr.predict(cr.iloc[: , 0: -1] )
```

In [64]:
```
pred_full_data
```

Out[64]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 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, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 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, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 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, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 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, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 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, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 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, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 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, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [65]:
```
pred_full_data_prob = logreg_cr.predict_proba(cr.iloc[: , 0: -1] )  ## only x variab
```

In [66]:
```
cr.iloc[: , 0: -1]
```

Out[66]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	1	0	0.0	0	0	5849	0.0
1	1	1	1.0	0	0	4583	1508.0
2	1	1	0.0	0	1	3000	0.0

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
3	1	1	0.0	1	0	2583	2358.0
4	1	0	0.0	0	0	6000	0.0
...	...	...	...	...	...	...	...
976	1	1	4.0	1	1	4009	1777.0
977	1	1	0.0	0	0	4158	709.0
978	1	0	0.0	0	0	3250	1993.0
979	1	1	0.0	0	0	5000	2393.0
980	1	0	0.0	0	1	9200	0.0

981 rows × 11 columns

In [67]:
```
pred_full_data_prob = pd.DataFrame(pred_full_data_prob)
pred_full_data_prob
```

Out[67]:

	0	1
0	0.140878	0.859122
1	0.135983	0.864017
2	0.071101	0.928899
3	0.162310	0.837690
4	0.139755	0.860245
...	...	...
976	0.207781	0.792219
977	0.089049	0.910951
978	0.168816	0.831184
979	0.144652	0.855348
980	0.080355	0.919645

981 rows × 2 columns

In [68]:
```
pred_full_data_prob.rename(columns ={pred_full_data_prob.columns[0]: "Pred_prob0"
                                     , pred_full_data_prob.columns[1]: "Pred_prob1"} , in
```

In [69]:
```
pred_full_data_prob
```

Out[69]:

	Pred_prob0	Pred_prob1
0	0.140878	0.859122
1	0.135983	0.864017
2	0.071101	0.928899

	Pred_prob0	Pred_prob1
3	0.162310	0.837690
4	0.139755	0.860245
...	...	...
976	0.207781	0.792219
977	0.089049	0.910951
978	0.168816	0.831184
979	0.144652	0.855348
980	0.080355	0.919645

981 rows × 2 columns

In [70]:
```
pred_full_data_prob.shape
```

Out[70]: (981, 2)

In [ ]:
```
## lets do two things (remove this columns "pred_prob0") & add loan_id column
```

In [71]:
```
pred_full_data_prob = pd.concat([pred_full_data_prob , cr1.Loan_ID] , axis=1)
pred_full_data_prob
```

Out[71]:

	Pred_prob0	Pred_prob1	Loan_ID
0	0.140878	0.859122	LP001002
1	0.135983	0.864017	LP001003
2	0.071101	0.928899	LP001005
3	0.162310	0.837690	LP001006
4	0.139755	0.860245	LP001008
...	...	...	...
976	0.207781	0.792219	LP002971
977	0.089049	0.910951	LP002975
978	0.168816	0.831184	LP002980
979	0.144652	0.855348	LP002986
980	0.080355	0.919645	LP002989

981 rows × 3 columns

In [72]:
```
pred_full_data_prob = pred_full_data_prob.iloc[:, [ 1 ,2]]
pred_full_data_prob
```

Out[72]:

	Pred_prob1	Loan_ID
0	0.859122	LP001002

	Pred_prob1	Loan_ID
1	0.864017	LP001003
2	0.928899	LP001005
3	0.837690	LP001006
4	0.860245	LP001008
...	...	...
976	0.792219	LP002971
977	0.910951	LP002975
978	0.831184	LP002980
979	0.855348	LP002986
980	0.919645	LP002989

981 rows × 2 columns

In [73]:
```
pred_full_data_prob.sort_values('Pred_prob1' ,ascending = False)
```

Out[73]:

	Pred_prob1	Loan_ID
497	0.970659	LP002588
133	0.970227	LP001482
14	0.966501	LP001030
164	0.963635	LP001572
686	0.959176	LP001375
...	...	...
325	0.083988	LP002067
338	0.079131	LP002113
925	0.076461	LP002747
639	0.054772	LP001153
177	0.030549	LP001610

981 rows × 2 columns

In [74]:
```
pred_full_data_prob.head(10)   ## target = will be which have higher probability i.e
```

Out[74]:

	Pred_prob1	Loan_ID
0	0.859122	LP001002
1	0.864017	LP001003
2	0.928899	LP001005
3	0.837690	LP001006
4	0.860245	LP001008

	Pred_prob1	Loan_ID
5	0.786572	LP001011
6	0.857026	LP001013
7	0.174590	LP001014
8	0.863700	LP001018
9	0.661568	LP001020

In [ ]:

over sampling

In [75]:
```
cr=pd.read_csv(r"D:\python data set lec\CreditRisk.csv")

cr.Credit_History = cr.Credit_History.fillna(1)
cr.Gender         = cr.Gender.fillna("Male")
cr.Married        = cr.Married.fillna("No")
cr.Dependents     = cr.Dependents.fillna(0)
cr.Self_Employed  = cr.Self_Employed.fillna("Yes")
cr.LoanAmount     = cr.LoanAmount.fillna(   cr.LoanAmount.mean()  )
cr.Loan_Amount_Term  = cr.Loan_Amount_Term.fillna(cr.Loan_Amount_Term.mean())

from sklearn.preprocessing  import LabelEncoder
le = LabelEncoder()

cr.Loan_Status  = le.fit_transform(cr.Loan_Status)
cr.Gender   = le.fit_transform(cr.Gender)
cr.Married  = le.fit_transform(cr.Married)
cr.Education = le.fit_transform(cr.Education)
cr.Self_Employed = le.fit_transform(cr.Self_Employed)
cr.Property_Area = le.fit_transform(cr.Property_Area)

cr1 = cr

from sklearn.model_selection import train_test_split

cr = cr.iloc[: , 1::]
```

In [59]:
```
cr_train , cr_test  = train_test_split(cr, test_size = .2)
```

In [60]:
```
# OS has to be done only on train
```

In [76]:
```
cr_train.shape
```

Out[76]:
```
(784, 12)
```

In [77]:
```
cr_train.Loan_Status.value_counts()
```

Out[77]:
```
1    565
0    219
Name: Loan_Status, dtype: int64
```

In [78]:
```
ab = cr_train[cr_train.Loan_Status == 0]
ab
```

Out[78]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
732	0	0	0.0	0	0	1762	2666.0
680	1	1	0.0	1	0	4700	0.0
373	1	0	1.0	0	0	3062	1987.0
457	1	1	0.0	0	0	3708	2569.0
911	1	0	0.0	1	1	3808	0.0
...	...	...	...	...	...	...	...
452	1	1	0.0	0	0	3948	1733.0
202	1	1	4.0	1	0	3992	0.0
307	0	0	0.0	0	0	2400	1863.0
550	1	1	2.0	0	1	6633	0.0
150	1	0	0.0	0	0	6277	0.0

219 rows × 12 columns

In [79]:
```
ab.Loan_Status.value_counts()
```

Out[79]:
```
0    219
Name: Loan_Status, dtype: int64
```

In [81]:
```
cr_train1 = pd.concat([cr_train , ab , ab , ab , ab , ab])
cr_train1
```

Out[81]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
468	0	1	2.0	1	1	210	2917.0
819	1	1	0.0	0	0	2613	2417.0
732	0	0	0.0	0	0	1762	2666.0
814	1	1	0.0	0	1	8706	0.0
158	1	0	0.0	0	1	2980	2083.0
...	...	...	...	...	...	...	...
452	1	1	0.0	0	0	3948	1733.0
202	1	1	4.0	1	0	3992	0.0
307	0	0	0.0	0	0	2400	1863.0
550	1	1	2.0	0	1	6633	0.0
150	1	0	0.0	0	0	6277	0.0

1879 rows × 12 columns

In [82]:
```
cr_train.shape
```

Out[82]:
```
(784, 12)
```

In [83]:
```
cr_train1.shape
```

Out[83]:
```
(1879, 12)
```

In [84]:
```
cr_train.Loan_Status.value_counts()
```

Out[84]:
```
1    565
0    219
Name: Loan_Status, dtype: int64
```

In [85]:
```
cr_train1.Loan_Status.value_counts()
```

Out[85]:
```
0    1314
1     565
Name: Loan_Status, dtype: int64
```

In [70]:
```
## now divide the train data in x & y
```

In [86]:
```
cr_train_x = cr_train1.iloc[: , 0:-1]
cr_train_y = cr_train1.iloc[: , -1]
```

In [87]:
```
cr_train_x
```

Out[87]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
468	0	1	2.0	1	1	210	2917.0
819	1	1	0.0	0	0	2613	2417.0
732	0	0	0.0	0	0	1762	2666.0
814	1	1	0.0	0	1	8706	0.0
158	1	0	0.0	0	1	2980	2083.0
...	...	...	...	...	...	...	...
452	1	1	0.0	0	0	3948	1733.0
202	1	1	4.0	1	0	3992	0.0
307	0	0	0.0	0	0	2400	1863.0
550	1	1	2.0	0	1	6633	0.0
150	1	0	0.0	0	0	6277	0.0

1879 rows × 11 columns

In [88]:
```
cr_train_y
```

Out[88]:
```
468    1
819    1
732    0
814    1
158    1
      ..
452    0
202    0
307    0
550    0
150    0
Name: Loan_Status, Length: 1879, dtype: int32
```

In [89]:
```
cr_test_x = cr_test.iloc[:, 0:-1]
cr_test_y = cr_test.iloc[:, -1]
```

In [90]:
```
from sklearn.linear_model import LogisticRegression
glm = LogisticRegression()
```

In [91]:
```
glm.fit(cr_train_x , cr_train_y)
```

Out[91]:
```
LogisticRegression()
```

In [92]:
```
pred_test_new = glm.predict(cr_test_x)
pred_test_new
```

Out[92]:
```
array([0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0])
```

In [93]:
```
from sklearn.metrics import confusion_matrix
```

In [94]:
```
tab2 = confusion_matrix(pred_test_new , cr_test_y)
tab2
```

Out[94]:
```
array([[35, 74],
       [15, 73]], dtype=int64)
```

In [95]:
```
fpr= 15 /(15+35)
fpr
```

Out[95]:
```
0.3
```

In [96]:
```
tpr = 73 /(73+74)
tpr
```

Out[96]:
```
0.4965986394557823
```

In [97]:
```
pre = 73 /(73+15)
pre
```

Out[97]:
```
0.8295454545454546
```

In [98]:
```
acc= (35+73) / (35+73+74+15)
acc
```

Out[98]:
```
0.5482233502538071
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: