In [4]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
df=pd.read_csv(r"C:\Users\chand\Desktop\DEEP LEARNING PROJECT MATERIAL\1585898503_datas
df.head()
```

Out[6]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	rev
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	2
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	3
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	3
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit.policy      9578 non-null   int64  
 1   purpose            9578 non-null   object 
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64  
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64  
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64  
 11  delinq.2yrs        9578 non-null   int64  
 12  pub.rec            9578 non-null   int64  
 13  not.fully.paid     9578 non-null   int64  
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

In [8]:
```
Data.isna().sum()
```

Out[8]:
```
credit.policy        0
purpose              0
int.rate             0
installment          0
log.annual.inc       0
dti                  0
fico                 0
days.with.cr.line    0
```

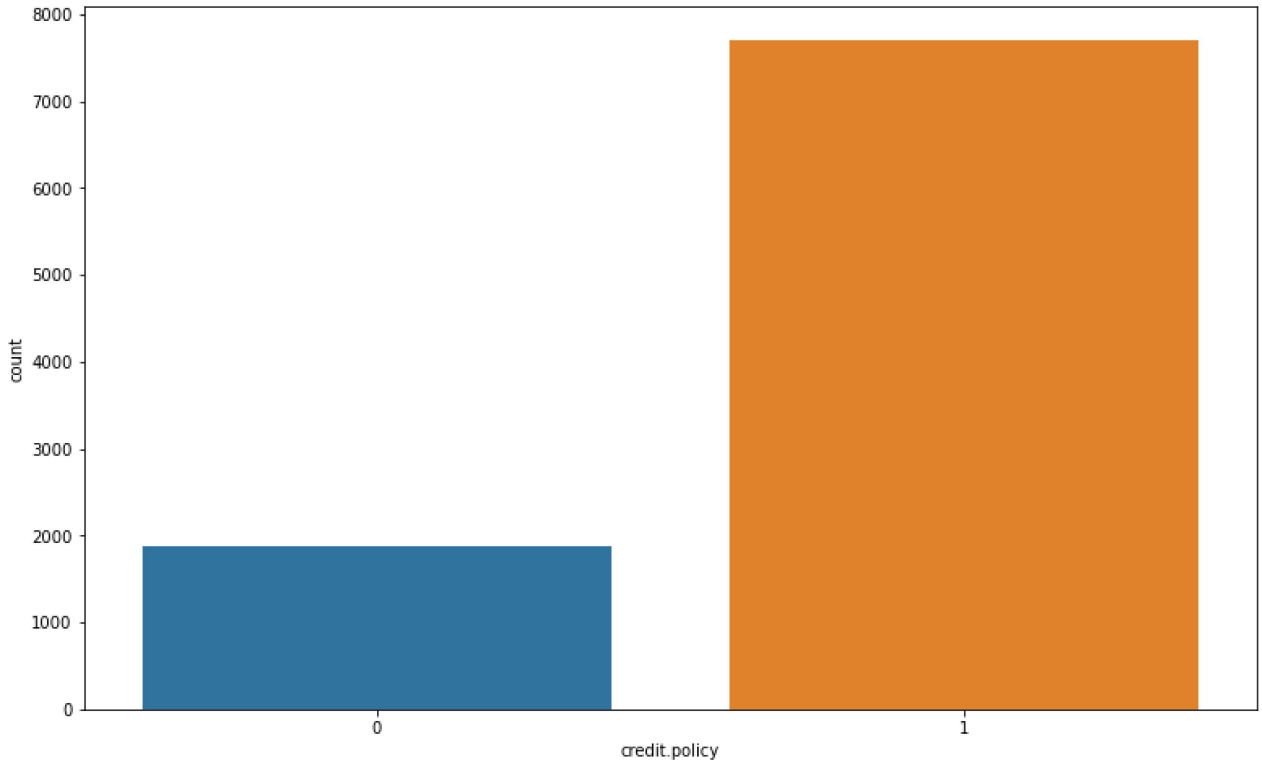
```
revol.bal           0
revol.util          0
inq.last.6mths      0
delinq.2yrs         0
pub.rec             0
not.fully.paid      0
dtype: int64
```

In [9]:
```
df.corr()
```

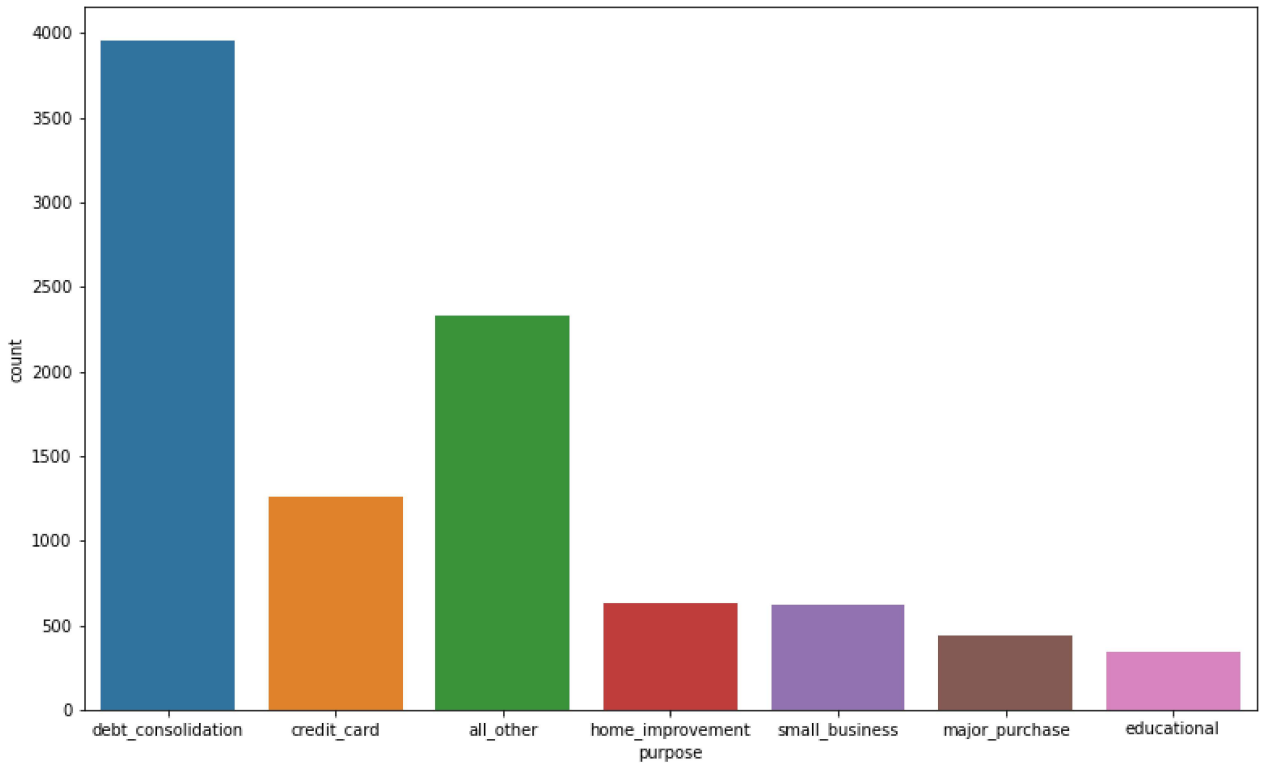
Out[9]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.lin
credit.policy	1.000000	-0.294089	0.058770	0.034906	-0.090901	0.348319	0.09902
int.rate	-0.294089	1.000000	0.276140	0.056383	0.220006	-0.714821	-0.12402
installment	0.058770	0.276140	1.000000	0.448102	0.050202	0.086039	0.18329
log.annual.inc	0.034906	0.056383	0.448102	1.000000	-0.054065	0.114576	0.33689
dti	-0.090901	0.220006	0.050202	-0.054065	1.000000	-0.241191	0.06010
fico	0.348319	-0.714821	0.086039	0.114576	-0.241191	1.000000	0.26388
days.with.cr.line	0.099026	-0.124022	0.183297	0.336896	0.060101	0.263880	1.00000
revol.bal	-0.187518	0.092527	0.233625	0.372140	0.188748	-0.015553	0.22934
revol.util	-0.104095	0.464837	0.081356	0.054881	0.337109	-0.541289	-0.02423
inq.last.6mths	-0.535511	0.202780	-0.010419	0.029171	0.029189	-0.185293	-0.04173
delinq.2yrs	-0.076318	0.156079	-0.004368	0.029203	-0.021792	-0.216340	0.08137
pub.rec	-0.054243	0.098162	-0.032760	0.016506	0.006209	-0.147592	0.07182
not.fully.paid	-0.158119	0.159552	0.049955	-0.033439	0.037362	-0.149666	-0.02923

In [10]:
```
plt.figure(figsize=(13,8))
sns.countplot(Data['credit.policy'])
plt.show()
```



In [11]:
```
plt.figure(figsize=(13,8))
sns.countplot(Data['purpose'])
plt.show()
```



In [12]:
```
dummy_purpose=pd.get_dummies(Data['purpose'])
dummy_purpose.head()
```

Out[12]:

	all_other	credit_card	debt_consolidation	educational	home_improvement	major_purchase	small_bus
0	0	0	1	0	0	0	
1	0	1	0	0	0	0	
2	0	0	1	0	0	0	
3	0	0	1	0	0	0	
4	0	1	0	0	0	0	

In [13]:

```
New_Data=pd.concat(([Data.iloc[:,0],dummy_purpose,Data.iloc[:,2:]]),axis=1)
New_Data.head()
```

Out[13]:

	credit.policy	all_other	credit_card	debt_consolidation	educational	home_improvement	major_purch
0	1	0	0	1	0	0	
1	1	0	1	0	0	0	
2	1	0	0	1	0	0	
3	1	0	0	1	0	0	
4	1	0	1	0	0	0	

In [14]:

```
import statsmodels.api as sm

print(sm.OLS(endog=New_Data.iloc[:,0],exog=New_Data.iloc[:,1:]).fit().summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          credit.policy   R-squared:                       0.401
Model:                            OLS   Adj. R-squared:                  0.400
Method:                 Least Squares   F-statistic:                     355.4
Date:                Sun, 28 May 2023   Prob (F-statistic):               0.00
Time:                        08:10:31   Log-Likelihood:                -2269.7
No. Observations:                9578   AIC:                             4577.
Df Residuals:                    9559   BIC:                             4714.
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
all_other          -1.1430      0.142     -8.071      0.000      -1.421      -0.865
credit_card        -1.1387      0.142     -8.043      0.000      -1.416      -0.861
debt_consolidation -1.1176      0.141     -7.924      0.000      -1.394      -0.841
educational        -1.1381      0.141     -8.048      0.000      -1.415      -0.861
home_improvement   -1.1250      0.143     -7.871      0.000      -1.405      -0.845
major_purchase     -1.1168      0.143     -7.834      0.000      -1.396      -0.837
small_business     -1.0872      0.144     -7.536      0.000      -1.370      -0.804
int.rate           -1.0911      0.201     -5.440      0.000      -1.484      -0.698
installment         0.0001   1.97e-05      5.739      0.000    7.44e-05       0.000
log.annual.inc      0.0441      0.006      6.900      0.000       0.032       0.057
dti                 0.0004      0.001      0.882      0.378      -0.001       0.001
fico                0.0023      0.000     15.342      0.000       0.002       0.003
days.with.cr.line 4.796e-06   1.44e-06      3.340      0.001    1.98e-06    7.61e-06
```

```
revol.bal          -2.717e-06   1.05e-07    -25.764      0.000   -2.92e-06   -2.51e-06
revol.util             0.0011      0.000      7.726      0.000       0.001       0.001
inq.last.6mths        -0.0843      0.001    -56.251      0.000      -0.087      -0.081
delinq.2yrs           -0.0104      0.006     -1.688      0.091      -0.022       0.002
pub.rec                0.0112      0.012      0.912      0.362      -0.013       0.035
not.fully.paid        -0.0427      0.009     -4.837      0.000      -0.060      -0.025
==============================================================================
Omnibus:                     1443.832   Durbin-Watson:                   0.795
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3297.711
Skew:                          -0.879   Prob(JB):                         0.00
Kurtosis:                       5.275   Cond. No.                     4.84e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 4.84e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [15]:

```
Data.describe().transpose()
```

Out[15]:

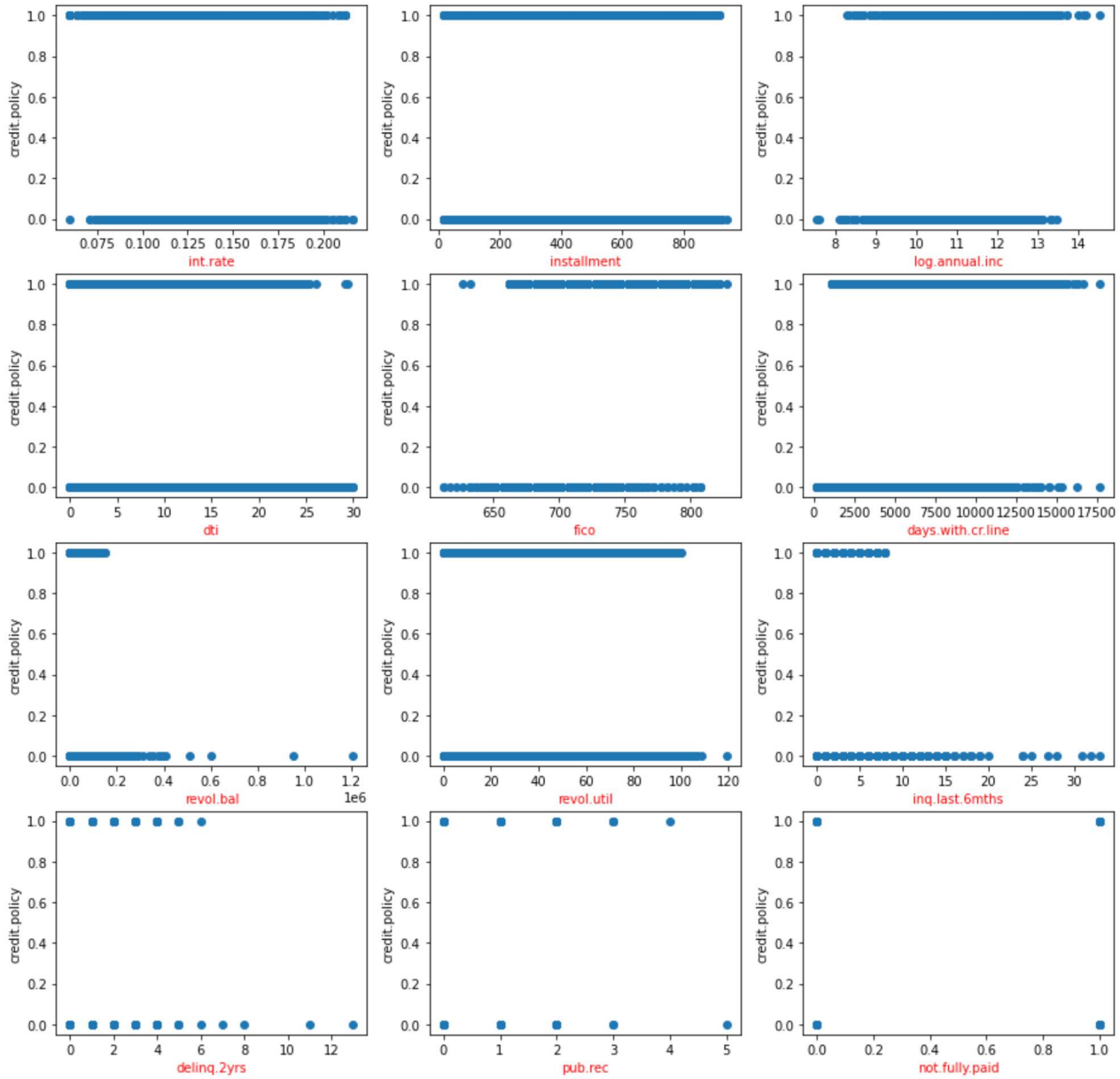
	count	mean	std	min	25%	50%	75%
credit.policy	9578.0	0.804970	0.396245	0.000000	1.000000	1.000000	1.000000
int.rate	9578.0	0.122640	0.026847	0.060000	0.103900	0.122100	0.140700
installment	9578.0	319.089413	207.071301	15.670000	163.770000	268.950000	432.762500
log.annual.inc	9578.0	10.932117	0.614813	7.547502	10.558414	10.928884	11.291293
dti	9578.0	12.606679	6.883970	0.000000	7.212500	12.665000	17.950000
fico	9578.0	710.846314	37.970537	612.000000	682.000000	707.000000	737.000000
days.with.cr.line	9578.0	4560.767197	2496.930377	178.958333	2820.000000	4139.958333	5730.000000
revol.bal	9578.0	16913.963876	33756.189557	0.000000	3187.000000	8596.000000	18249.500000
revol.util	9578.0	46.799236	29.014417	0.000000	22.600000	46.300000	70.900000
inq.last.6mths	9578.0	1.577469	2.200245	0.000000	0.000000	1.000000	2.000000
delinq.2yrs	9578.0	0.163708	0.546215	0.000000	0.000000	0.000000	0.000000
pub.rec	9578.0	0.062122	0.262126	0.000000	0.000000	0.000000	0.000000
not.fully.paid	9578.0	0.160054	0.366676	0.000000	0.000000	0.000000	0.000000

In [16]:

```
def show_data(data_name):
  fig,axs = plt.subplots(nrows=4,ncols=3,figsize=(15,15))
  b=0
  c=0
  a=3
  for i in data_name:
    axs[b,c].scatter(y=Data['credit.policy'],x=Data[f'{i}'])
    axs[b,c].set_ylabel('credit.policy')
    axs[b,c].set_xlabel(f'{i}',color='red')
    c+=1
    if c==a:
      b+=1
```

```
        c=0
      elif b==4:
        break
    plt.show()
```

In [17]:
```
show_data(Data.columns[2:])
```



In [18]:
```
#Train, Validation, Test
x=New_Data.copy().drop(columns='credit.policy')
y=New_Data.copy()['credit.policy']
```

In [19]:
```
x.head()
```

Out[19]:

	all_other	credit_card	debt_consolidation	educational	home_improvement	major_purchase	small_bus
0	0	0	1	0	0	0	
1	0	1	0	0	0	0	

	all_other	credit_card	debt_consolidation	educational	home_improvement	major_purchase	small_bus
2	0	0	1	0	0	0	
3	0	0	1	0	0	0	
4	0	1	0	0	0	0	

In [20]:
```
y.head()
```

Out[20]:
```
0    1
1    1
2    1
3    1
4    1
Name: credit.policy, dtype: int64
```

In [21]:
```
x_array=x.values
y_array=y.values
```

In [22]:
```
x.shape
```

Out[22]: (9578, 19)

In [23]:
```
y.shape
```

Out[23]: (9578,)

In [24]:
```
from sklearn.model_selection import train_test_split

x_train,x_val,y_train,y_val=train_test_split(x_array,y_array,test_size=0.05,random_stat
x_train,x_test,y_train,y_test=train_test_split(x_train,y_train,test_size=0.20,random_st
```

In [25]:
```
x_train.shape
```

Out[25]: (7279, 19)

In [26]:
```
y_train.shape
```

Out[26]: (7279,)

In [27]:
```
x_val.shape
```

Out[27]: (479, 19)

In [28]:
```
y_val.shape
```

Out[28]: (479,)

In [29]:
```
x_test.shape
```

Out[29]: (1820, 19)

In [30]:
```
y_test.shape
```

Out[30]: (1820,)

In [31]:
```
from sklearn.preprocessing import StandardScaler

obje_ss=StandardScaler()

x_train_ss=obje_ss.fit_transform(x_train)
x_val_ss=obje_ss.fit_transform(x_val)
x_test_ss=obje_ss.fit_transform(x_test)
```

In [32]:
```
x_train
```

Out[32]:
```
array([[0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.]])
```

In [33]:
```
x_train_ss
```

Out[33]:
```
array([[-0.56449927, -0.39117462,  1.18788184, ..., -0.29521897,
        -0.23860477, -0.44230165],
       [-0.56449927, -0.39117462,  1.18788184, ..., -0.29521897,
        -0.23860477, -0.44230165],
       [-0.56449927, -0.39117462,  1.18788184, ..., -0.29521897,
        -0.23860477, -0.44230165],
       ...,
       [-0.56449927, -0.39117462,  1.18788184, ..., -0.29521897,
        -0.23860477, -0.44230165],
       [-0.56449927, -0.39117462, -0.84183457, ..., -0.29521897,
        -0.23860477, -0.44230165],
       [-0.56449927, -0.39117462,  1.18788184, ..., -0.29521897,
        -0.23860477, -0.44230165]])
```

In [34]:
```
#Model
from sklearn.metrics import r2_score,classification_report,accuracy_score,confusion_mat
```

In [35]:
```
#Logistic Regression
from sklearn.linear_model import LogisticRegression

model_log=LogisticRegression(solver='lbfgs',max_iter=200,random_state=42).fit(x_train,y
y_pred=model_log.predict(x_val)
print(model_log)
```

```
LogisticRegression(max_iter=200, random_state=42)
```

In [36]:
```
print(confusion_matrix(y_val,y_pred))
```

```
[[ 61  40]
 [  5 373]]
```

In [37]:
```
print('Train success rate: %',model_log.score(x_train,y_train)*100)
print('Test success rate: %',accuracy_score(y_val,y_pred)*100)
```

```
Train success rate: % 89.82003022393185
Test success rate: % 90.6054279749478
```

In [38]:
```
print(classification_report(y_val,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.60      0.73       101
           1       0.90      0.99      0.94       378

    accuracy                           0.91       479
   macro avg       0.91      0.80      0.84       479
weighted avg       0.91      0.91      0.90       479
```

In [39]:
```
#SVC
from sklearn.svm import SVC

model_svc=SVC(C=1,kernel='rbf',degree=3,random_state=42).fit(x_train_ss,y_train)
y_pred=model_svc.predict(x_val_ss)
print(model_svc)
```

```
SVC(C=1, random_state=42)
```

In [40]:
```
print(confusion_matrix(y_val,y_pred))
```

```
[[ 72  29]
 [  8 370]]
```

In [41]:
```
print('Train success rate: %',model_svc.score(x_train_ss,y_train)*100)
print('Test success rate: %',accuracy_score(y_val,y_pred)*100)
```

```
Train success rate: % 93.85904657233137
Test success rate: % 92.27557411273486
```

In [42]:
```
print(classification_report(y_val,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.71      0.80       101
           1       0.93      0.98      0.95       378

    accuracy                           0.92       479
   macro avg       0.91      0.85      0.87       479
weighted avg       0.92      0.92      0.92       479
```

In [43]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier

model_knn=KNeighborsClassifier(n_neighbors=5,metric='minkowski').fit(x_train,y_train)
y_pred=model_knn.predict(x_val)
print(model_knn)
```

```
KNeighborsClassifier()
```

In [44]:
```
print(confusion_matrix(y_val,y_pred))
```

```
[[ 32  69]
 [  7 371]]
```

In [45]:
```
print('Train success rate: %',model_knn.score(x_train,y_train)*100)
print('Test success rate: %',accuracy_score(y_val,y_pred)*100)
```

```
Train success rate: % 85.79475202637725
Test success rate: % 84.13361169102296
```

In [46]:
```
print(classification_report(y_val,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.32      0.46       101
           1       0.84      0.98      0.91       378

    accuracy                           0.84       479
   macro avg       0.83      0.65      0.68       479
weighted avg       0.84      0.84      0.81       479
```

In [47]:
```
#Navie Bayes
from sklearn.naive_bayes import  BernoulliNB

model_bnb=BernoulliNB().fit(x_train,y_train)
y_pred=model_bnb.predict(x_val)
print(model_bnb)
```

```
BernoulliNB()
```

In [48]:
```
print(confusion_matrix(y_val,y_pred))
```

```
[[  7  94]
 [  8 370]]
```

In [49]:
```
print('Train success rate: %',model_bnb.score(x_train,y_train)*100)
print('Test success rate: %',accuracy_score(y_val,y_pred)*100)
```

```
Train success rate: % 80.38192059348812
Test success rate: % 78.70563674321504
```

In [50]:
```
print(classification_report(y_val,y_pred))
```

```
              precision    recall  f1-score   support
```

```
              0       0.47      0.07      0.12       101
              1       0.80      0.98      0.88       378

    accuracy                            0.79       479
   macro avg       0.63      0.52      0.50       479
weighted avg       0.73      0.79      0.72       479
```

In [51]:
```
#Tree Models
from sklearn.ensemble import RandomForestClassifier

model_rfr=RandomForestClassifier(n_estimators=100,criterion='gini',random_state=42,n_jo
y_pred=model_rfr.predict(x_val)
print(model_rfr)
```

```
RandomForestClassifier(n_jobs=-1, random_state=42)
```

In [52]:
```
print(confusion_matrix(y_val,y_pred))
```

```
[[ 95   6]
 [  1 377]]
```

In [53]:
```
print('Train success rate: %',model_rfr.score(x_train,y_train)*100)
print('Test success rate: %',accuracy_score(y_val,y_pred)*100)
```

```
Train success rate: % 100.0
Test success rate: % 98.53862212943632
```

In [54]:
```
print(classification_report(y_val,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.94      0.96       101
           1       0.98      1.00      0.99       378

    accuracy                           0.99       479
   macro avg       0.99      0.97      0.98       479
weighted avg       0.99      0.99      0.99       479
```

In [55]:
```
#Boosting
from xgboost import XGBClassifier

model_xgb=XGBClassifier(max_depth=4,learning_rate=0.1,n_estimators=100,objective='binar
y_pred=model_xgb.predict(x_val)
print(model_xgb)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=-1, num_parallel_tree=None,
              predictor=None, random_state=42, ...)
```

In [56]:
```
print(confusion_matrix(y_val,y_pred))
```

```
[[ 97   4]
 [  1 377]]
```

In [57]:
```
print('Train success rate: %',model_xgb.score(x_train,y_train)*100)
print('Test success rate: %',accuracy_score(y_val,y_pred)*100)
```

```
Train success rate: % 99.67028437972249
Test success rate: % 98.95615866388309
```

In [58]:
```
print(classification_report(y_val,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.96      0.97       101
           1       0.99      1.00      0.99       378

    accuracy                           0.99       479
   macro avg       0.99      0.98      0.98       479
weighted avg       0.99      0.99      0.99       479
```

In [59]:
```
#Test
def test_score(model_name):

  for i in model_name:
    print(f'{i.__class__} \n{classification_report(y_test,i.predict(x_test))}')
```

In [60]:
```
model_names=[model_log,model_knn,model_bnb,model_rfr,model_xgb]
test_score(model_names)
```

```
<class 'sklearn.linear_model._logistic.LogisticRegression'>
              precision    recall  f1-score   support

           0       0.85      0.57      0.68       363
           1       0.90      0.97      0.94      1457

    accuracy                           0.89      1820
   macro avg       0.88      0.77      0.81      1820
weighted avg       0.89      0.89      0.89      1820

<class 'sklearn.neighbors._classification.KNeighborsClassifier'>
              precision    recall  f1-score   support

           0       0.65      0.25      0.36       363
           1       0.84      0.97      0.90      1457

    accuracy                           0.82      1820
   macro avg       0.74      0.61      0.63      1820
weighted avg       0.80      0.82      0.79      1820

<class 'sklearn.naive_bayes.BernoulliNB'>
              precision    recall  f1-score   support

           0       0.47      0.05      0.09       363
           1       0.81      0.99      0.89      1457
```

```
    accuracy                           0.80      1820
   macro avg       0.64      0.52      0.49      1820
weighted avg       0.74      0.80      0.73      1820

<class 'sklearn.ensemble._forest.RandomForestClassifier'>
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       363
           1       0.99      1.00      0.99      1457

    accuracy                           0.99      1820
   macro avg       0.99      0.98      0.98      1820
weighted avg       0.99      0.99      0.99      1820

<class 'xgboost.sklearn.XGBClassifier'>
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       363
           1       0.99      1.00      1.00      1457

    accuracy                           0.99      1820
   macro avg       0.99      0.99      0.99      1820
weighted avg       0.99      0.99      0.99      1820
```

In [61]:

```
#Cross Validation
from sklearn.model_selection import cross_val_score
def cross_score(model_name):

  for i in model_name:
    print(f'{i.__class__}  | Cross val score: %{cross_val_score(i,X=x_train,y=y_train,c
cross_score(model_names)
```

```
<class 'sklearn.linear_model._logistic.LogisticRegression'>  | Cross val score: %89.7513
5946527699
<class 'sklearn.neighbors._classification.KNeighborsClassifier'>  | Cross val score: %8
2.86855670103093
<class 'sklearn.naive_bayes.BernoulliNB'>  | Cross val score: %80.32706468788943
<class 'sklearn.ensemble._forest.RandomForestClassifier'>  | Cross val score: %98.763604
093501
<class 'xgboost.sklearn.XGBClassifier'>  | Cross val score: %98.96967637173823
```