HouseLoanAnalysis

January 4, 2021

[2]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[3]:
```python
data_set=pd.read_csv(r"loan_data.csv")
data_set.head()
```

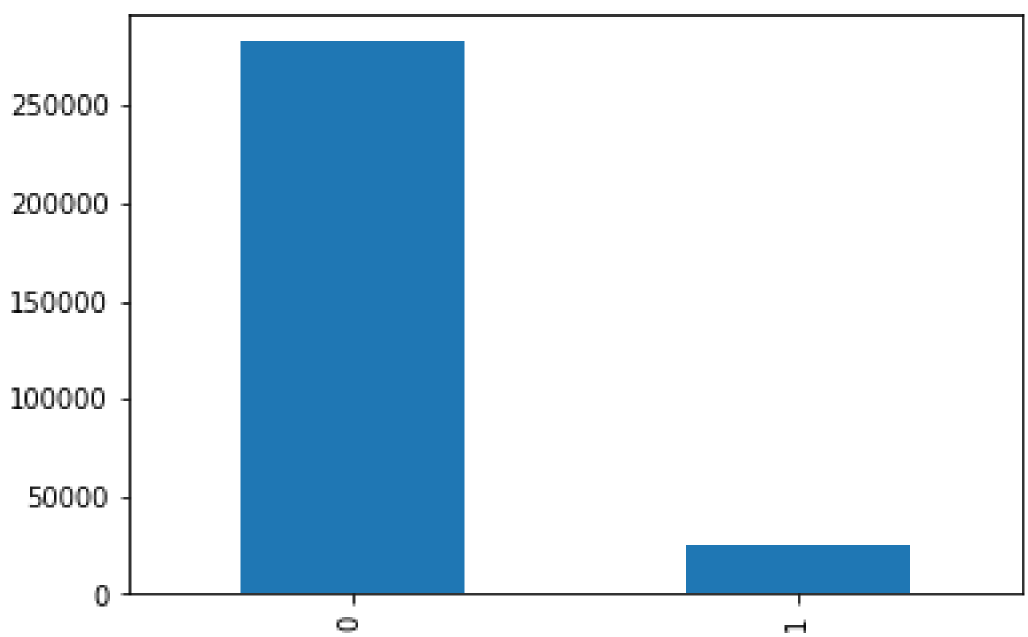
[3]:
```
   SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR  \
0      100002       1         Cash loans           M            N
1      100003       0         Cash loans           F            N
2      100004       0    Revolving loans           M            Y
3      100006       0         Cash loans           F            N
4      100007       0         Cash loans           M            N

  FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  \
0               Y             0          202500.0    406597.5      24700.5
1               N             0          270000.0   1293502.5      35698.5
2               Y             0           67500.0    135000.0       6750.0
3               Y             0          135000.0    312682.5      29686.5
4               Y             0          121500.0    513000.0      21865.5

   …  FLAG_DOCUMENT_18 FLAG_DOCUMENT_19 FLAG_DOCUMENT_20 FLAG_DOCUMENT_21  \
0  …                 0                0                0                0
1  …                 0                0                0                0
2  …                 0                0                0                0
3  …                 0                0                0                0
4  …                 0                0                0                0

  AMT_REQ_CREDIT_BUREAU_HOUR AMT_REQ_CREDIT_BUREAU_DAY  \
0                        0.0                       0.0
1                        0.0                       0.0
2                        0.0                       0.0
3                        NaN                       NaN
4                        0.0                       0.0
```

```
   AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
0                         0.0                        0.0
1                         0.0                        0.0
2                         0.0                        0.0
3                         NaN                        NaN
4                         0.0                        0.0

   AMT_REQ_CREDIT_BUREAU_QRT  AMT_REQ_CREDIT_BUREAU_YEAR
0                        0.0                         1.0
1                        0.0                         0.0
2                        0.0                         0.0
3                        NaN                         NaN
4                        0.0                         0.0

[5 rows x 122 columns]
```

[14]:

```
#checking for percentage of defaulters and non-defaulters
defaulters=data_set.TARGET.value_counts()[0]
non_defaulters=data_set.TARGET.value_counts()[1]
fraction_of_defaulters=defaulters/(defaulters + non_defaulters)
percentage_of_defaulters = round((fraction_of_defaulters * 100),2)
percentage_of_defaulters
data_set.TARGET.value_counts().plot.bar()
```

[14]: <AxesSubplot:>



1 Exploratory Data Analysis

[18]:
```
df=data_set
target=df["TARGET"]
data=df.drop(columns=["TARGET","SK_ID_CURR"])
print(data.shape,target.shape)
```

```
(307511, 120) (307511,)
```

[19]:
```
def check_missing_data(df):
    mis_col=[]
    for i in df.columns:
        percent=df[i].isnull().mean()
        if percent !=0:
            mis_col.append(i)
    if len(mis_col)==0:
        print('no columns with missing value')
    else:
        print('number of columns with missing value : ',len(mis_col))
        return mis_col
```

[20]:
```
#checking for percentage of missing values
check_missing_data(data_set)
```

```
number of columns with missing value :  67
```

[20]:
```
['AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'OWN_CAR_AGE',
 'OCCUPATION_TYPE',
 'CNT_FAM_MEMBERS',
 'EXT_SOURCE_1',
 'EXT_SOURCE_2',
 'EXT_SOURCE_3',
 'APARTMENTS_AVG',
 'BASEMENTAREA_AVG',
 'YEARS_BEGINEXPLUATATION_AVG',
 'YEARS_BUILD_AVG',
 'COMMONAREA_AVG',
 'ELEVATORS_AVG',
 'ENTRANCES_AVG',
 'FLOORSMAX_AVG',
 'FLOORSMIN_AVG',
 'LANDAREA_AVG',
 'LIVINGAPARTMENTS_AVG',
 'LIVINGAREA_AVG',
```

```
'NONLIVINGAPARTMENTS_AVG',
'NONLIVINGAREA_AVG',
'APARTMENTS_MODE',
'BASEMENTAREA_MODE',
'YEARS_BEGINEXPLUATATION_MODE',
'YEARS_BUILD_MODE',
'COMMONAREA_MODE',
'ELEVATORS_MODE',
'ENTRANCES_MODE',
'FLOORSMAX_MODE',
'FLOORSMIN_MODE',
'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE',
'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE',
'NONLIVINGAREA_MODE',
'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI',
'COMMONAREA_MEDI',
'ELEVATORS_MEDI',
'ENTRANCES_MEDI',
'FLOORSMAX_MEDI',
'FLOORSMIN_MEDI',
'LANDAREA_MEDI',
'LIVINGAPARTMENTS_MEDI',
'LIVINGAREA_MEDI',
'NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAREA_MEDI',
'FONDKAPREMONT_MODE',
'HOUSETYPE_MODE',
'TOTALAREA_MODE',
'WALLSMATERIAL_MODE',
'EMERGENCYSTATE_MODE',
'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE',
'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE',
'DAYS_LAST_PHONE_CHANGE',
'AMT_REQ_CREDIT_BUREAU_HOUR',
'AMT_REQ_CREDIT_BUREAU_DAY',
'AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_MON',
'AMT_REQ_CREDIT_BUREAU_QRT',
'AMT_REQ_CREDIT_BUREAU_YEAR']
```

[21]:
```
#seperating categorical and non-categorical columns
categ_col=[x for x in data if data[x].dtype=='O']
non_categorical=data.drop(columns=categ_col)
categorical=data[categ_col]
```

[29]:
```
#handling missing values
#non_categorical=non_categorical.interpolate(method='linear')
def input_missing_value(df):
    try:
        for column in df.columns:
            i=df.columns.get_loc(column)
            df.iloc[:,i].fillna(df[column].mode()[0],inplace=True)
    except:
            pass
input_missing_value(categorical)
input_missing_value(non_categorical)
```

[30]:
```
check_missing_data(non_categorical)
check_missing_data(categorical)
```

```
no columns with missing value
no columns with missing value
```

[35]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in categorical.columns:
    categorical[i]=le.fit_transform(categorical[i])
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

[36]:
```
features=pd.concat([non_categorical,categorical],axis=1)
print(non_categorical.shape,categorical.shape)
features.shape
```

```
(307511, 104) (307511, 16)
```

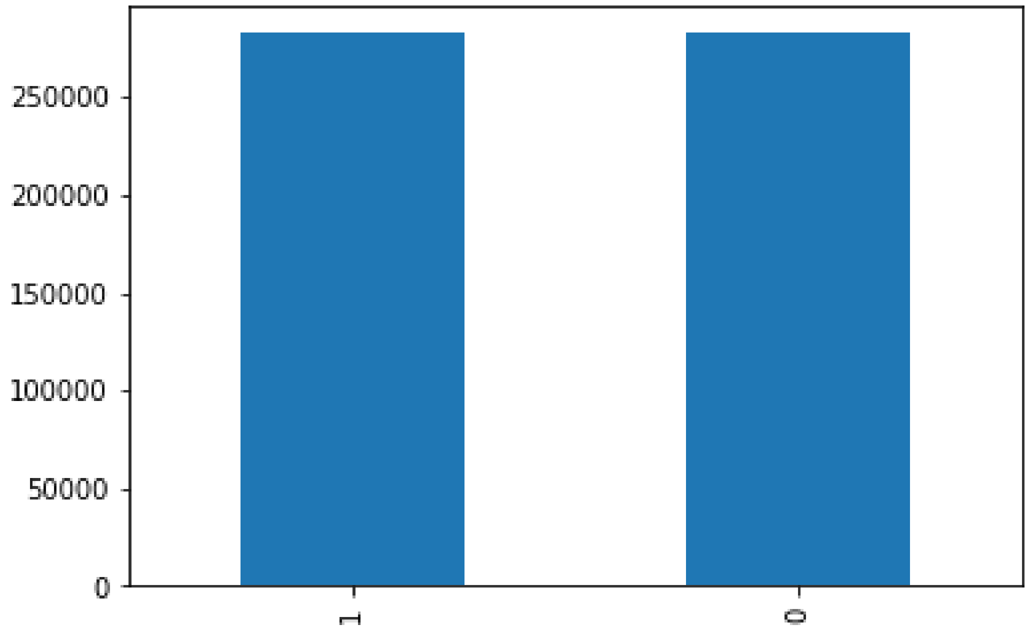
[36]: (307511, 120)

[37]:
```
#upscaling the data to make the dataset balanced
from imblearn.over_sampling import SMOTE
smk=SMOTE()
```

```
X_set,Y_set=smk.fit_sample(features,target)
print(X_set.shape,Y_set.shape)
Y_set.value_counts().plot.bar()
```

```
Using TensorFlow backend.

(565372, 120) (565372,)
```

[37]: <AxesSubplot:>



[39]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X_set,Y_set,test_size=.3)
print(X_train.shape,X_test.shape)
print(Y_train.shape,Y_test.shape)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

print(X_train.shape,X_test.shape)
```

```
(395760, 120) (169612, 120)
(395760,) (169612,)
(395760, 120) (169612, 120)
```

2 Model architecting

[40]:
```
import keras
from keras.losses import binary_crossentropy
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense,Dropout
#from keras.wrappers.scikit_learn import KerasClassifier
#from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
```

[ ]:
```
#function to architect the model for hyper-tuning
def create_model(layers,activation,optimizer):
    model=Sequential()
    for i,nodes in enumerate(layers):
        if i==0:
            model.add(Dense(nodes,input_dim=X_train.shape[1]))
            model.add(Activation(activation))
        else:
            model.add(Dense(nodes))
            model.add(Activation(activation))
        model.add(Dense(1))
        model.
 ↪compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy'])
        print('compiled')
    return model
layers=(64,32,16)
activations=['sigmoid','relu']
optimizers=['adam','sgd','rmsprop']
```

[42]:
```
#manually architecting the model
model=Sequential()
model.add(Dense(64,activation='relu',input_dim=X_train.
 ↪shape[1],kernel_initializer='uniform'))
model.add(Dropout(0.3))

model.add(Dense(32,activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(16,activation='relu'))
model.add(Dropout(0.3))

model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

model.summary()
```

```python
model_train=model.
↪fit(X_train,Y_train,batch_size=64,epochs=100,verbose=1,validation_split=0.2)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 64)                7744
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_7 (Dense)              (None, 16)                528
_________________________________________________________________
dropout_6 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 17
=================================================================
Total params: 10,369
Trainable params: 10,369
Non-trainable params: 0
_________________________________________________________________
Train on 316608 samples, validate on 79152 samples
Epoch 1/100
316608/316608 [==============================] - 10s 32us/step - loss: 0.3839 -
accuracy: 0.8379 - val_loss: 0.3253 - val_accuracy: 0.8623
Epoch 2/100
316608/316608 [==============================] - 10s 32us/step - loss: 0.3387 -
accuracy: 0.8602 - val_loss: 0.2997 - val_accuracy: 0.8739
Epoch 3/100
316608/316608 [==============================] - 10s 31us/step - loss: 0.3209 -
accuracy: 0.8678 - val_loss: 0.2880 - val_accuracy: 0.8801
Epoch 4/100
316608/316608 [==============================] - 10s 30us/step - loss: 0.3116 -
accuracy: 0.8719 - val_loss: 0.2800 - val_accuracy: 0.8841
Epoch 5/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.3052 -
accuracy: 0.8752 - val_loss: 0.2746 - val_accuracy: 0.8866
Epoch 6/100
316608/316608 [==============================] - 11s 33us/step - loss: 0.3007 -
accuracy: 0.8768 - val_loss: 0.2707 - val_accuracy: 0.8892
Epoch 7/100
316608/316608 [==============================] - 11s 35us/step - loss: 0.2983 -
accuracy: 0.8792 - val_loss: 0.2748 - val_accuracy: 0.8878
```

```
Epoch 8/100
316608/316608 [==============================] - 12s 39us/step - loss: 0.2954 -
accuracy: 0.8801 - val_loss: 0.2677 - val_accuracy: 0.8904
Epoch 9/100
316608/316608 [==============================] - 11s 34us/step - loss: 0.2922 -
accuracy: 0.8811 - val_loss: 0.2664 - val_accuracy: 0.8929
Epoch 10/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2902 -
accuracy: 0.8818 - val_loss: 0.2621 - val_accuracy: 0.8939
Epoch 11/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2893 -
accuracy: 0.8828 - val_loss: 0.2604 - val_accuracy: 0.8943
Epoch 12/100
316608/316608 [==============================] - 10s 32us/step - loss: 0.2868 -
accuracy: 0.8834 - val_loss: 0.2607 - val_accuracy: 0.8950
Epoch 13/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2856 -
accuracy: 0.8851 - val_loss: 0.2576 - val_accuracy: 0.8960
Epoch 14/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2845 -
accuracy: 0.8844 - val_loss: 0.2554 - val_accuracy: 0.8969
Epoch 15/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2825 -
accuracy: 0.8861 - val_loss: 0.2601 - val_accuracy: 0.8961
Epoch 16/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2813 -
accuracy: 0.8867 - val_loss: 0.2565 - val_accuracy: 0.8977
Epoch 17/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2807 -
accuracy: 0.8869 - val_loss: 0.2535 - val_accuracy: 0.8985
Epoch 18/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2801 -
accuracy: 0.8876 - val_loss: 0.2530 - val_accuracy: 0.8982
Epoch 19/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2788 -
accuracy: 0.8880 - val_loss: 0.2527 - val_accuracy: 0.8994
Epoch 20/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2783 -
accuracy: 0.8884 - val_loss: 0.2530 - val_accuracy: 0.8983
Epoch 21/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2767 -
accuracy: 0.8889 - val_loss: 0.2534 - val_accuracy: 0.8988
Epoch 22/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2761 -
accuracy: 0.8892 - val_loss: 0.2515 - val_accuracy: 0.8983
Epoch 23/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2743 -
accuracy: 0.8898 - val_loss: 0.2486 - val_accuracy: 0.9012
```

```
Epoch 24/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2751 -
accuracy: 0.8899 - val_loss: 0.2502 - val_accuracy: 0.9002
Epoch 25/100
316608/316608 [==============================] - 10s 31us/step - loss: 0.2737 -
accuracy: 0.8906 - val_loss: 0.2510 - val_accuracy: 0.9009
Epoch 26/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2736 -
accuracy: 0.8900 - val_loss: 0.2461 - val_accuracy: 0.9021
Epoch 27/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2716 -
accuracy: 0.8915 - val_loss: 0.2482 - val_accuracy: 0.9016
Epoch 28/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2717 -
accuracy: 0.8916 - val_loss: 0.2473 - val_accuracy: 0.9020
Epoch 29/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2711 -
accuracy: 0.8912 - val_loss: 0.2449 - val_accuracy: 0.9025
Epoch 30/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2708 -
accuracy: 0.8918 - val_loss: 0.2455 - val_accuracy: 0.9034
Epoch 31/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2702 -
accuracy: 0.8918 - val_loss: 0.2450 - val_accuracy: 0.9030
Epoch 32/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2698 -
accuracy: 0.8922 - val_loss: 0.2426 - val_accuracy: 0.9042
Epoch 33/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2691 -
accuracy: 0.8924 - val_loss: 0.2442 - val_accuracy: 0.9035
Epoch 34/100
316608/316608 [==============================] - 9s 30us/step - loss: 0.2688 -
accuracy: 0.8928 - val_loss: 0.2461 - val_accuracy: 0.9025
Epoch 35/100
316608/316608 [==============================] - 10s 30us/step - loss: 0.2678 -
accuracy: 0.8927 - val_loss: 0.2486 - val_accuracy: 0.9032
Epoch 36/100
316608/316608 [==============================] - 11s 36us/step - loss: 0.2676 -
accuracy: 0.8933 - val_loss: 0.2433 - val_accuracy: 0.9046
Epoch 37/100
316608/316608 [==============================] - 12s 37us/step - loss: 0.2669 -
accuracy: 0.8937 - val_loss: 0.2436 - val_accuracy: 0.9046
Epoch 38/100
316608/316608 [==============================] - 11s 36us/step - loss: 0.2678 -
accuracy: 0.8931 - val_loss: 0.2437 - val_accuracy: 0.9034
Epoch 39/100
316608/316608 [==============================] - 11s 36us/step - loss: 0.2666 -
accuracy: 0.8940 - val_loss: 0.2420 - val_accuracy: 0.9046
```

```
Epoch 40/100
316608/316608 [==============================] - 11s 36us/step - loss: 0.2663 -
accuracy: 0.8940 - val_loss: 0.2416 - val_accuracy: 0.9058
Epoch 41/100
316608/316608 [==============================] - 12s 37us/step - loss: 0.2659 -
accuracy: 0.8941 - val_loss: 0.2404 - val_accuracy: 0.9056
Epoch 42/100
316608/316608 [==============================] - 11s 36us/step - loss: 0.2653 -
accuracy: 0.8943 - val_loss: 0.2427 - val_accuracy: 0.9058
Epoch 43/100
316608/316608 [==============================] - 11s 36us/step - loss: 0.2650 -
accuracy: 0.8944 - val_loss: 0.2407 - val_accuracy: 0.9054
Epoch 44/100
316608/316608 [==============================] - 12s 37us/step - loss: 0.2664 -
accuracy: 0.8938 - val_loss: 0.2372 - val_accuracy: 0.9069
Epoch 45/100
316608/316608 [==============================] - 12s 38us/step - loss: 0.2644 -
accuracy: 0.8947 - val_loss: 0.2388 - val_accuracy: 0.9062
Epoch 46/100
316608/316608 [==============================] - 11s 36us/step - loss: 0.2648 -
accuracy: 0.8944 - val_loss: 0.2386 - val_accuracy: 0.9056
Epoch 47/100
316608/316608 [==============================] - 13s 40us/step - loss: 0.2645 -
accuracy: 0.8948 - val_loss: 0.2398 - val_accuracy: 0.9051
Epoch 48/100
316608/316608 [==============================] - 12s 37us/step - loss: 0.2620 -
accuracy: 0.8958 - val_loss: 0.2388 - val_accuracy: 0.9056
Epoch 49/100
316608/316608 [==============================] - 12s 37us/step - loss: 0.2676 -
accuracy: 0.8953 - val_loss: 0.2377 - val_accuracy: 0.9061
Epoch 50/100
316608/316608 [==============================] - 12s 37us/step - loss: 0.2626 -
accuracy: 0.8955 - val_loss: 0.2380 - val_accuracy: 0.9064
Epoch 51/100
316608/316608 [==============================] - 11s 36us/step - loss: 0.2624 -
accuracy: 0.8955 - val_loss: 0.2367 - val_accuracy: 0.9070
Epoch 52/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2628 -
accuracy: 0.8954 - val_loss: 0.2378 - val_accuracy: 0.9068
Epoch 53/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2617 -
accuracy: 0.8961 - val_loss: 0.2359 - val_accuracy: 0.9075
Epoch 54/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2628 -
accuracy: 0.8954 - val_loss: 0.2361 - val_accuracy: 0.9076
Epoch 55/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2620 -
accuracy: 0.8959 - val_loss: 0.2384 - val_accuracy: 0.9072
```

```
Epoch 56/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2619 -
accuracy: 0.8957 - val_loss: 0.2374 - val_accuracy: 0.9071
Epoch 57/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2611 -
accuracy: 0.8957 - val_loss: 0.2354 - val_accuracy: 0.9072
Epoch 58/100
316608/316608 [==============================] - 9s 30us/step - loss: 0.2608 -
accuracy: 0.8962 - val_loss: 0.2375 - val_accuracy: 0.9070
Epoch 59/100
316608/316608 [==============================] - 10s 33us/step - loss: 0.2621 -
accuracy: 0.8966 - val_loss: 0.2376 - val_accuracy: 0.9068
Epoch 60/100
316608/316608 [==============================] - 11s 34us/step - loss: 0.2643 -
accuracy: 0.8969 - val_loss: 0.2361 - val_accuracy: 0.9076
Epoch 61/100
316608/316608 [==============================] - 11s 35us/step - loss: 0.2613 -
accuracy: 0.8967 - val_loss: 0.2373 - val_accuracy: 0.9068
Epoch 62/100
316608/316608 [==============================] - 11s 35us/step - loss: 0.2604 -
accuracy: 0.8968 - val_loss: 0.2364 - val_accuracy: 0.9076
Epoch 63/100
316608/316608 [==============================] - 11s 34us/step - loss: 0.2607 -
accuracy: 0.8965 - val_loss: 0.2367 - val_accuracy: 0.9071
Epoch 64/100
316608/316608 [==============================] - 11s 34us/step - loss: 0.2604 -
accuracy: 0.8966 - val_loss: 0.2374 - val_accuracy: 0.9075
Epoch 65/100
316608/316608 [==============================] - 11s 33us/step - loss: 0.2605 -
accuracy: 0.8967 - val_loss: 0.2346 - val_accuracy: 0.9081
Epoch 66/100
316608/316608 [==============================] - 11s 34us/step - loss: 0.2591 -
accuracy: 0.8972 - val_loss: 0.2344 - val_accuracy: 0.9083
Epoch 67/100
316608/316608 [==============================] - 10s 33us/step - loss: 0.2597 -
accuracy: 0.8976 - val_loss: 0.2355 - val_accuracy: 0.9075
Epoch 68/100
316608/316608 [==============================] - 10s 32us/step - loss: 0.2590 -
accuracy: 0.8972 - val_loss: 0.2350 - val_accuracy: 0.9082
Epoch 69/100
316608/316608 [==============================] - 10s 31us/step - loss: 0.2573 -
accuracy: 0.8981 - val_loss: 0.2332 - val_accuracy: 0.9093
Epoch 70/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2591 -
accuracy: 0.8973 - val_loss: 0.2350 - val_accuracy: 0.9087
Epoch 71/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2588 -
accuracy: 0.8974 - val_loss: 0.2337 - val_accuracy: 0.9088
```

```
Epoch 72/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2585 -
accuracy: 0.8975 - val_loss: 0.2356 - val_accuracy: 0.9083
Epoch 73/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2583 -
accuracy: 0.8980 - val_loss: 0.2352 - val_accuracy: 0.9087
Epoch 74/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2580 -
accuracy: 0.8977 - val_loss: 0.2359 - val_accuracy: 0.9086
Epoch 75/100
316608/316608 [==============================] - 9s 29us/step - loss: 0.2576 -
accuracy: 0.8978 - val_loss: 0.2331 - val_accuracy: 0.9090
Epoch 76/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2584 -
accuracy: 0.8981 - val_loss: 0.2332 - val_accuracy: 0.9088
Epoch 77/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2577 -
accuracy: 0.8978 - val_loss: 0.2347 - val_accuracy: 0.9086
Epoch 78/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2576 -
accuracy: 0.8979 - val_loss: 0.2351 - val_accuracy: 0.9083
Epoch 79/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2587 -
accuracy: 0.8982 - val_loss: 0.2335 - val_accuracy: 0.9083
Epoch 80/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2589 -
accuracy: 0.8982 - val_loss: 0.2327 - val_accuracy: 0.9091
Epoch 81/100
316608/316608 [==============================] - 9s 27us/step - loss: 0.2572 -
accuracy: 0.8982 - val_loss: 0.2358 - val_accuracy: 0.9081
Epoch 82/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2585 -
accuracy: 0.8977 - val_loss: 0.2354 - val_accuracy: 0.9089
Epoch 83/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2578 -
accuracy: 0.8981 - val_loss: 0.2347 - val_accuracy: 0.9088
Epoch 84/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2569 -
accuracy: 0.8985 - val_loss: 0.2312 - val_accuracy: 0.9093
Epoch 85/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2569 -
accuracy: 0.8990 - val_loss: 0.2320 - val_accuracy: 0.9098
Epoch 86/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2563 -
accuracy: 0.8993 - val_loss: 0.2364 - val_accuracy: 0.9083
Epoch 87/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2600 -
accuracy: 0.8980 - val_loss: 0.2337 - val_accuracy: 0.9094
```

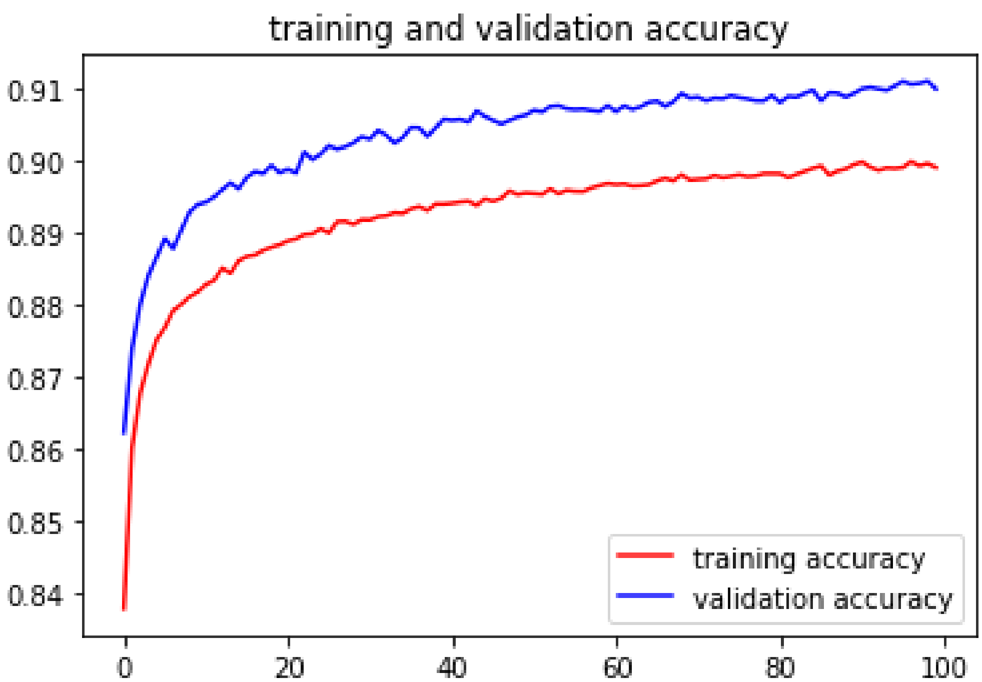
```
Epoch 88/100
316608/316608 [==============================] - 9s 27us/step - loss: 0.2572 -
accuracy: 0.8986 - val_loss: 0.2330 - val_accuracy: 0.9094
Epoch 89/100
316608/316608 [==============================] - 9s 27us/step - loss: 0.2567 -
accuracy: 0.8989 - val_loss: 0.2319 - val_accuracy: 0.9088
Epoch 90/100
316608/316608 [==============================] - 9s 27us/step - loss: 0.2562 -
accuracy: 0.8995 - val_loss: 0.2311 - val_accuracy: 0.9093
Epoch 91/100
316608/316608 [==============================] - 9s 27us/step - loss: 0.2555 -
accuracy: 0.8998 - val_loss: 0.2312 - val_accuracy: 0.9100
Epoch 92/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2562 -
accuracy: 0.8991 - val_loss: 0.2318 - val_accuracy: 0.9102
Epoch 93/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2563 -
accuracy: 0.8987 - val_loss: 0.2349 - val_accuracy: 0.9100
Epoch 94/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2567 -
accuracy: 0.8990 - val_loss: 0.2330 - val_accuracy: 0.9098
Epoch 95/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2559 -
accuracy: 0.8989 - val_loss: 0.2305 - val_accuracy: 0.9104
Epoch 96/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2554 -
accuracy: 0.8991 - val_loss: 0.2320 - val_accuracy: 0.9110
Epoch 97/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2550 -
accuracy: 0.8999 - val_loss: 0.2305 - val_accuracy: 0.9106
Epoch 98/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2569 -
accuracy: 0.8993 - val_loss: 0.2318 - val_accuracy: 0.9107
Epoch 99/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2554 -
accuracy: 0.8996 - val_loss: 0.2299 - val_accuracy: 0.9111
Epoch 100/100
316608/316608 [==============================] - 9s 28us/step - loss: 0.2559 -
accuracy: 0.8991 - val_loss: 0.2319 - val_accuracy: 0.9100
```

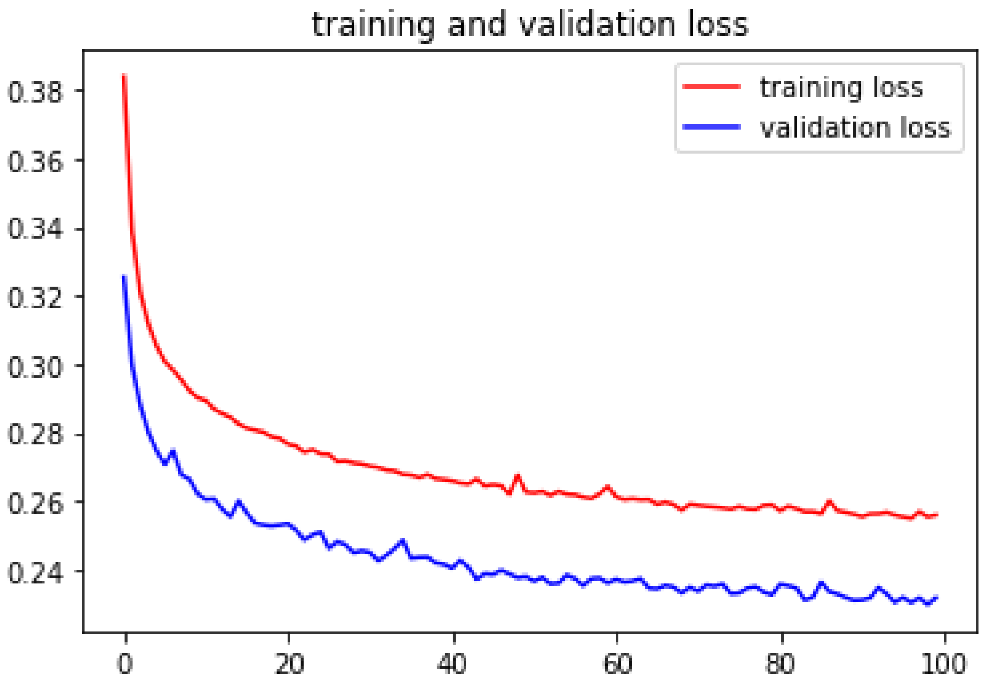
[43]:
```
#evaluating the model using plots
test_eval=model.evaluate(X_test,Y_test,verbose=1)
print("test loss : ",test_eval[0])
print("test accuracy : ",test_eval[1])
accuracy=model_train.history['accuracy']
val_accuracy=model_train.history['val_accuracy']
loss=model_train.history['loss']
```

```
val_loss=model_train.history['val_loss']
epochs=range(len(accuracy))
plt.plot(epochs,accuracy,'r',label='training accuracy')
plt.plot(epochs,val_accuracy,'b',label='validation accuracy')
plt.title('training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs,loss,'r',label='training loss')
plt.plot(epochs,val_loss,'b',label='validation loss')
plt.title('training and validation loss')
plt.legend()
plt.figure()
```

```
169612/169612 [==============================] - 3s 16us/step
test loss :  0.23198463619064588
test accuracy :  0.9088095426559448
```

[43]: <Figure size 432x288 with 0 Axes>



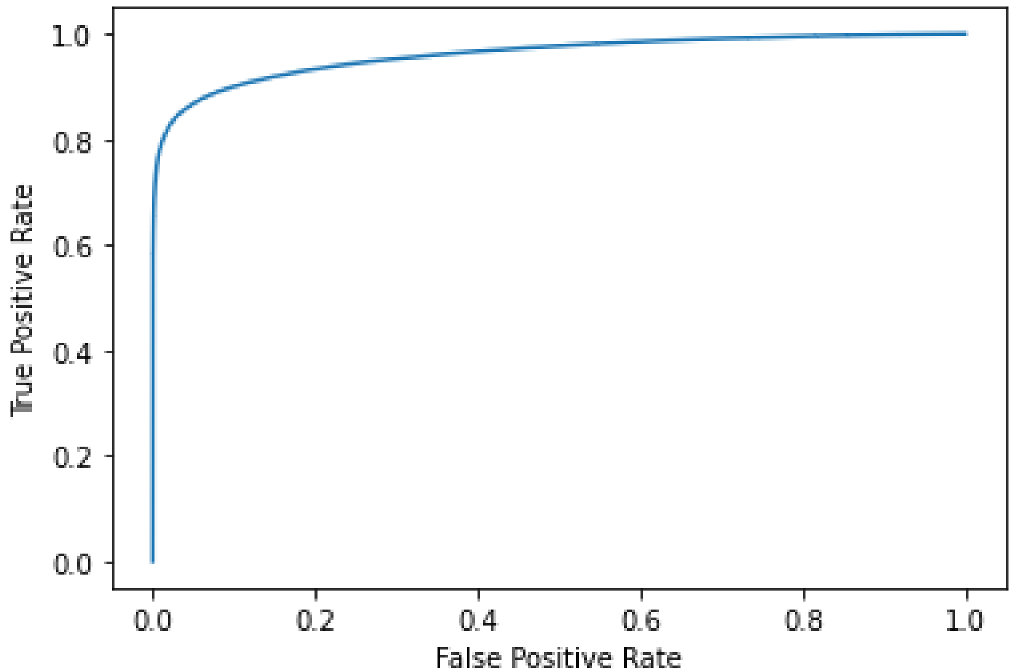



```
<Figure size 432x288 with 0 Axes>
```

[44]:
```python
from sklearn.metrics import roc_curve,roc_auc_score
Y_pred=model.predict(X_test)
auc_score=roc_auc_score(Y_test,Y_pred)
print(auc_score)
fpr,tpr,_=roc_curve(Y_test,Y_pred)
plt.plot(fpr,tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
0.9604345218764301
```

[44]:
```
Text(0, 0.5, 'True Positive Rate')
```


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

[ ]: