House Loan Data Analysis Project

Importing required libraries

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
from subprocess import check_output
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

In [ ]:
```
loan_ds = pd.read_csv('loan_data.csv')
loan_ds.head()
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_R
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	

5 rows × 122 columns

In [ ]:
```
target=loan_ds["TARGET"]
data=loan_ds.drop(columns=["TARGET","SK_ID_CURR"])
print(data.shape,target.shape)
```

```
(13574, 120) (13574,)
```

Check for null values in the dataset

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

In [ ]:
```
check_missing_data(loan_ds)
```

```
number of columns with missing value :  110
```

Out[ ]:
```
['AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'OWN_CAR_AGE',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'OCCUPATION_TYPE',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'WEEKDAY_APPR_PROCESS_START',
 'HOUR_APPR_PROCESS_START',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'LIVE_CITY_NOT_WORK_CITY',
 'ORGANIZATION_TYPE',
 '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',
 'FLAG_DOCUMENT_2',
 'FLAG_DOCUMENT_3',
 'FLAG_DOCUMENT_4',
 'FLAG_DOCUMENT_5',
 'FLAG_DOCUMENT_6',
 'FLAG_DOCUMENT_7',
 'FLAG_DOCUMENT_8',
 'FLAG_DOCUMENT_9',
 'FLAG_DOCUMENT_10',
 'FLAG_DOCUMENT_11',
 'FLAG_DOCUMENT_12',
 'FLAG_DOCUMENT_13',
 'FLAG_DOCUMENT_14',
 'FLAG_DOCUMENT_15',
 'FLAG_DOCUMENT_16',
 'FLAG_DOCUMENT_17',
 'FLAG_DOCUMENT_18',
 'FLAG_DOCUMENT_19',
 'FLAG_DOCUMENT_20',
 'FLAG_DOCUMENT_21',
 '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']
```

In [ ]:

```
#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]
```

In [ ]:

```
categorical
```

Out[ ]:

	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_TYPI
0	Cash loans	M	N	Y	Unaccor
1	Cash loans	F	N	N	
2	Revolving loans	M	Y	Y	Unaccor
3	Cash loans	F	N	Y	Unaccor
4	Cash loans	M	N	Y	Unaccor
...	...	...	...	...	
13569	Cash loans	M	Y	Y	Unaccor
13570	Cash loans	F	Y	Y	
13571	Cash loans	M	Y	N	Unaccor
13572	Cash loans	M	Y	Y	Unaccor
13573	Cash loans	M	N	N	Unaccor

13574 rows × 16 columns

In [ ]:

```
#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)
```

In [ ]:

```
check_missing_data(non_categorical)
check_missing_data(categorical)
```

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

Print the percentage of default to payer of the dataset for the target column

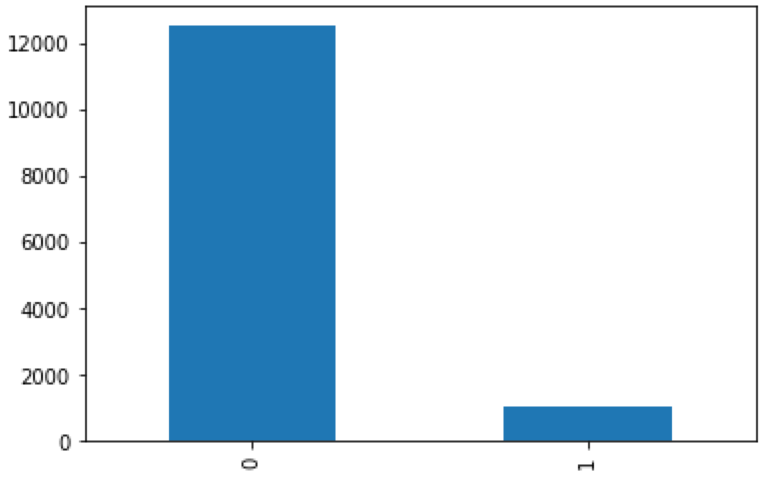
In [ ]:

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

```
Percentage of defaulters  92.29
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff31f70f290>
```



In [ ]:

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

```
NAME_CONTRACT_TYPE
CODE_GENDER
FLAG_OWN_CAR
FLAG_OWN_REALTY
NAME_TYPE_SUITE
NAME_INCOME_TYPE
NAME_EDUCATION_TYPE
NAME_FAMILY_STATUS
NAME_HOUSING_TYPE
OCCUPATION_TYPE
WEEKDAY_APPR_PROCESS_START
ORGANIZATION_TYPE
FONDKAPREMONT_MODE
HOUSETYPE_MODE
WALLSMATERIAL_MODE
EMERGENCYSTATE_MODE
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWar
ning:
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/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
```

In [ ]:

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

```
(13574, 104) (13574, 16)
```

Out[ ]:

```
(13574, 120)
```

Handling imbalances in the dataset

Using SMOTE method

In [ ]:

```python
from imblearn.over_sampling import SMOTE
```

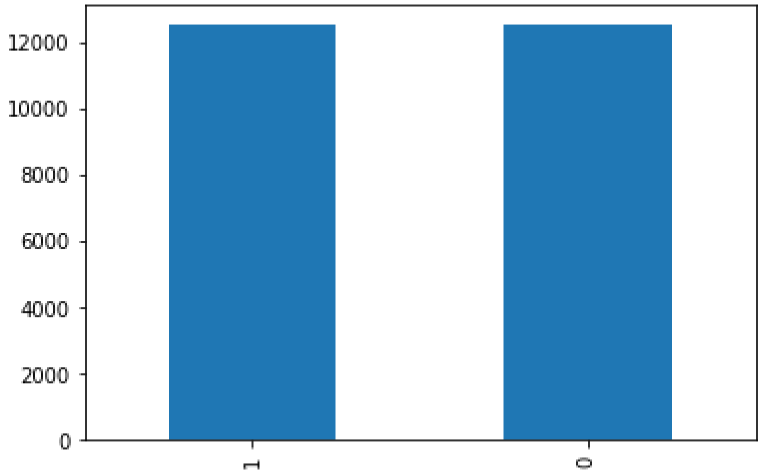
```
sm = SMOTE()
```

In [ ]:
```
X, y = sm.fit_resample(features, target)
```

In [ ]:
```
print(X.shape, y.shape)
y.value_counts().plot.bar()
```

```
(25056, 120) (25056,)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff2a2288a50>
```



In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
print('Train Sizes: {}, {}'.format(X_train.shape, y_train.shape))
print('Test Sizes: {}, {}'.format(X_test.shape, y_test.shape))
```

```
Train Sizes: (18792, 120), (18792,)
Test Sizes: (6264, 120), (6264,)
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)

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

```
(18792, 120) (6264, 120)
```

Building a neural network

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

In [ ]:
```
X_train.shape[1]
```

Out[ ]:
```
120
```

In [ ]:
```
model = Sequential([
    Dense(64,input_dim=X_train.shape[1], activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [ ]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 64)                7744      
                                                                 
 dropout (Dropout)           (None, 64)                0         
                                                                 
 dense_1 (Dense)             (None, 32)                2080      
                                                                 
 dropout_1 (Dropout)         (None, 32)                0         
                                                                 
 dense_2 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 9,857
Trainable params: 9,857
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
train_history= model.fit(X_train, y_train, batch_size=64, epochs=100, validation_s
```

```
Epoch 1/100
235/235 [==============================] - 5s 6ms/step - loss: 0.6209 - accuracy: 0.6682 - val_loss: 0.5103 - val_accuracy: 0.7664
Epoch 2/100
235/235 [==============================] - 1s 4ms/step - loss: 0.5046 - accuracy: 0.7637 - val_loss: 0.4716 - val_accuracy: 0.7840
Epoch 3/100
235/235 [==============================] - 1s 4ms/step - loss: 0.4719 - accuracy: 0.7817 - val_loss: 0.4461 - val_accuracy: 0.7962
Epoch 4/100
235/235 [==============================] - 1s 4ms/step - loss: 0.4462 - accuracy: 0.7988 - val_loss: 0.4240 - val_accuracy: 0.8175
Epoch 5/100
235/235 [==============================] - 1s 4ms/step - loss: 0.4217 - accuracy: 0.8115 - val_loss: 0.4038 - val_accuracy: 0.8255
Epoch 6/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3995 - accuracy: 0.8219 - val_loss: 0.3840 - val_accuracy: 0.8343
Epoch 7/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3829 - accuracy: 0.8345 - val_loss: 0.3646 - val_accuracy: 0.8399
Epoch 8/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3626 - accuracy: 0.8397 - val_loss: 0.3502 - val_accuracy: 0.8500
Epoch 9/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3579 - accuracy: 0.8453 - val_loss: 0.3401 - val_accuracy: 0.8547
Epoch 10/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3446 - accuracy: 0.8547 - val_loss: 0.3268 - val_accuracy: 0.8649
Epoch 11/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3287 - accuracy: 0.8605 - val_loss: 0.3153 - val_accuracy: 0.8686
Epoch 12/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3208 - accuracy: 0.8659 - val_loss: 0.3076 - val_accuracy: 0.8734
Epoch 13/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3130 - accuracy: 0.8692 - val_loss: 0.2976 - val_accuracy: 0.8720
Epoch 14/100
235/235 [==============================] - 1s 4ms/step - loss: 0.3020 - accuracy: 0.8737 - val_loss: 0.2918 - val_accuracy: 0.8760
Epoch 15/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2977 - accuracy: 0.8764 - val_loss: 0.2864 - val_accuracy: 0.8840
Epoch 16/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2860 - accuracy: 0.8799 - val_loss: 0.2847 - val_accuracy: 0.8819
Epoch 17/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2839 - accuracy: 0.8803 - val_loss: 0.2801 - val_accuracy: 0.8875
Epoch 18/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2776 - accuracy: 0.8855 - val_loss: 0.2736 - val_accuracy: 0.8888
Epoch 19/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2698 - accuracy: 0.8883 - val_loss: 0.2728 - val_accuracy: 0.8901
Epoch 20/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2661 - accuracy: 0.8900 - val_loss: 0.2658 - val_accuracy: 0.8896
Epoch 21/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2643 - accuracy: 0.8906 - val_loss: 0.2663 - val_accuracy: 0.8907
Epoch 22/100
```

```
235/235 [==============================] - 1s 4ms/step - loss: 0.2570 - accuracy:
0.8948 - val_loss: 0.2598 - val_accuracy: 0.8957
Epoch 23/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2515 - accuracy:
0.8980 - val_loss: 0.2571 - val_accuracy: 0.8947
Epoch 24/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2564 - accuracy:
0.8927 - val_loss: 0.2528 - val_accuracy: 0.8965
Epoch 25/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2408 - accuracy:
0.9023 - val_loss: 0.2527 - val_accuracy: 0.8960
Epoch 26/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2386 - accuracy:
0.9024 - val_loss: 0.2503 - val_accuracy: 0.8981
Epoch 27/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2388 - accuracy:
0.9018 - val_loss: 0.2458 - val_accuracy: 0.9018
Epoch 28/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2291 - accuracy:
0.9043 - val_loss: 0.2423 - val_accuracy: 0.9042
Epoch 29/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2326 - accuracy:
0.9061 - val_loss: 0.2430 - val_accuracy: 0.9037
Epoch 30/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2293 - accuracy:
0.9077 - val_loss: 0.2398 - val_accuracy: 0.9032
Epoch 31/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2205 - accuracy:
0.9113 - val_loss: 0.2372 - val_accuracy: 0.9056
Epoch 32/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2190 - accuracy:
0.9109 - val_loss: 0.2355 - val_accuracy: 0.9056
Epoch 33/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2253 - accuracy:
0.9099 - val_loss: 0.2284 - val_accuracy: 0.9133
Epoch 34/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2134 - accuracy:
0.9173 - val_loss: 0.2336 - val_accuracy: 0.9061
Epoch 35/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2101 - accuracy:
0.9151 - val_loss: 0.2302 - val_accuracy: 0.9074
Epoch 36/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2089 - accuracy:
0.9169 - val_loss: 0.2236 - val_accuracy: 0.9130
Epoch 37/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2112 - accuracy:
0.9157 - val_loss: 0.2248 - val_accuracy: 0.9101
Epoch 38/100
235/235 [==============================] - 1s 4ms/step - loss: 0.2085 - accuracy:
0.9168 - val_loss: 0.2229 - val_accuracy: 0.9141
Epoch 39/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1988 - accuracy:
0.9184 - val_loss: 0.2231 - val_accuracy: 0.9119
Epoch 40/100
235/235 [==============================] - 2s 8ms/step - loss: 0.1963 - accuracy:
0.9214 - val_loss: 0.2240 - val_accuracy: 0.9114
Epoch 41/100
235/235 [==============================] - 2s 7ms/step - loss: 0.1936 - accuracy:
0.9229 - val_loss: 0.2194 - val_accuracy: 0.9138
Epoch 42/100
235/235 [==============================] - 2s 8ms/step - loss: 0.1949 - accuracy:
0.9253 - val_loss: 0.2149 - val_accuracy: 0.9170
Epoch 43/100
235/235 [==============================] - 2s 9ms/step - loss: 0.1940 - accuracy:
```

```
0.9243 - val_loss: 0.2221 - val_accuracy: 0.9146
Epoch 44/100
235/235 [==============================] - 2s 8ms/step - loss: 0.1888 - accuracy:
0.9267 - val_loss: 0.2114 - val_accuracy: 0.9149
Epoch 45/100
235/235 [==============================] - 2s 8ms/step - loss: 0.1929 - accuracy:
0.9252 - val_loss: 0.2113 - val_accuracy: 0.9199
Epoch 46/100
235/235 [==============================] - 2s 7ms/step - loss: 0.1878 - accuracy:
0.9266 - val_loss: 0.2171 - val_accuracy: 0.9194
Epoch 47/100
235/235 [==============================] - 2s 7ms/step - loss: 0.1866 - accuracy:
0.9268 - val_loss: 0.2080 - val_accuracy: 0.9191
Epoch 48/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1824 - accuracy:
0.9287 - val_loss: 0.2085 - val_accuracy: 0.9199
Epoch 49/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1846 - accuracy:
0.9295 - val_loss: 0.2074 - val_accuracy: 0.9255
Epoch 50/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1855 - accuracy:
0.9278 - val_loss: 0.2111 - val_accuracy: 0.9205
Epoch 51/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1809 - accuracy:
0.9276 - val_loss: 0.2105 - val_accuracy: 0.9199
Epoch 52/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1814 - accuracy:
0.9258 - val_loss: 0.2097 - val_accuracy: 0.9213
Epoch 53/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1781 - accuracy:
0.9295 - val_loss: 0.2057 - val_accuracy: 0.9186
Epoch 54/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1767 - accuracy:
0.9298 - val_loss: 0.2063 - val_accuracy: 0.9223
Epoch 55/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1718 - accuracy:
0.9329 - val_loss: 0.2054 - val_accuracy: 0.9226
Epoch 56/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1732 - accuracy:
0.9337 - val_loss: 0.2020 - val_accuracy: 0.9236
Epoch 57/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1715 - accuracy:
0.9345 - val_loss: 0.2078 - val_accuracy: 0.9231
Epoch 58/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1720 - accuracy:
0.9339 - val_loss: 0.2048 - val_accuracy: 0.9197
Epoch 59/100
235/235 [==============================] - 1s 5ms/step - loss: 0.1733 - accuracy:
0.9324 - val_loss: 0.2019 - val_accuracy: 0.9234
Epoch 60/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1712 - accuracy:
0.9339 - val_loss: 0.1984 - val_accuracy: 0.9231
Epoch 61/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1632 - accuracy:
0.9345 - val_loss: 0.2011 - val_accuracy: 0.9229
Epoch 62/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1665 - accuracy:
0.9345 - val_loss: 0.1954 - val_accuracy: 0.9260
Epoch 63/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1685 - accuracy:
0.9348 - val_loss: 0.1986 - val_accuracy: 0.9252
Epoch 64/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1607 - accuracy:
0.9375 - val_loss: 0.1948 - val_accuracy: 0.9242
```

```
Epoch 65/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1619 - accuracy:
0.9365 - val_loss: 0.2058 - val_accuracy: 0.9210
Epoch 66/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1682 - accuracy:
0.9346 - val_loss: 0.1974 - val_accuracy: 0.9236
Epoch 67/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1569 - accuracy:
0.9377 - val_loss: 0.2002 - val_accuracy: 0.9250
Epoch 68/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1596 - accuracy:
0.9375 - val_loss: 0.1971 - val_accuracy: 0.9250
Epoch 69/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1603 - accuracy:
0.9361 - val_loss: 0.1944 - val_accuracy: 0.9276
Epoch 70/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1560 - accuracy:
0.9393 - val_loss: 0.1890 - val_accuracy: 0.9306
Epoch 71/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1584 - accuracy:
0.9375 - val_loss: 0.1924 - val_accuracy: 0.9290
Epoch 72/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1566 - accuracy:
0.9388 - val_loss: 0.1900 - val_accuracy: 0.9295
Epoch 73/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1589 - accuracy:
0.9371 - val_loss: 0.1901 - val_accuracy: 0.9314
Epoch 74/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1517 - accuracy:
0.9429 - val_loss: 0.1996 - val_accuracy: 0.9263
Epoch 75/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1523 - accuracy:
0.9403 - val_loss: 0.2032 - val_accuracy: 0.9255
Epoch 76/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1570 - accuracy:
0.9387 - val_loss: 0.1963 - val_accuracy: 0.9284
Epoch 77/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1512 - accuracy:
0.9401 - val_loss: 0.1923 - val_accuracy: 0.9279
Epoch 78/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1500 - accuracy:
0.9401 - val_loss: 0.1937 - val_accuracy: 0.9298
Epoch 79/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1533 - accuracy:
0.9401 - val_loss: 0.2010 - val_accuracy: 0.9239
Epoch 80/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1488 - accuracy:
0.9429 - val_loss: 0.1856 - val_accuracy: 0.9300
Epoch 81/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1500 - accuracy:
0.9429 - val_loss: 0.1901 - val_accuracy: 0.9287
Epoch 82/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1496 - accuracy:
0.9399 - val_loss: 0.1923 - val_accuracy: 0.9242
Epoch 83/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1443 - accuracy:
0.9436 - val_loss: 0.1931 - val_accuracy: 0.9274
Epoch 84/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1420 - accuracy:
0.9453 - val_loss: 0.1969 - val_accuracy: 0.9263
Epoch 85/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1462 - accuracy:
0.9438 - val_loss: 0.1886 - val_accuracy: 0.9274
Epoch 86/100
```

```
235/235 [==============================] - 2s 7ms/step - loss: 0.1414 - accuracy: 0.9454 - val_loss: 0.1953 - val_accuracy: 0.9274
Epoch 87/100
235/235 [==============================] - 2s 7ms/step - loss: 0.1451 - accuracy: 0.9439 - val_loss: 0.1957 - val_accuracy: 0.9279
Epoch 88/100
235/235 [==============================] - 2s 7ms/step - loss: 0.1428 - accuracy: 0.9455 - val_loss: 0.1975 - val_accuracy: 0.9263
Epoch 89/100
235/235 [==============================] - 2s 7ms/step - loss: 0.1408 - accuracy: 0.9475 - val_loss: 0.2035 - val_accuracy: 0.9282
Epoch 90/100
235/235 [==============================] - 2s 6ms/step - loss: 0.1370 - accuracy: 0.9472 - val_loss: 0.1990 - val_accuracy: 0.9268
Epoch 91/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1412 - accuracy: 0.9466 - val_loss: 0.2005 - val_accuracy: 0.9292
Epoch 92/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1494 - accuracy: 0.9441 - val_loss: 0.1978 - val_accuracy: 0.9308
Epoch 93/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1390 - accuracy: 0.9476 - val_loss: 0.1979 - val_accuracy: 0.9300
Epoch 94/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1423 - accuracy: 0.9449 - val_loss: 0.2008 - val_accuracy: 0.9258
Epoch 95/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1439 - accuracy: 0.9442 - val_loss: 0.1982 - val_accuracy: 0.9303
Epoch 96/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1395 - accuracy: 0.9455 - val_loss: 0.2019 - val_accuracy: 0.9298
Epoch 97/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1422 - accuracy: 0.9433 - val_loss: 0.1993 - val_accuracy: 0.9295
Epoch 98/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1401 - accuracy: 0.9469 - val_loss: 0.2022 - val_accuracy: 0.9290
Epoch 99/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1366 - accuracy: 0.9467 - val_loss: 0.1964 - val_accuracy: 0.9295
Epoch 100/100
235/235 [==============================] - 1s 4ms/step - loss: 0.1379 - accuracy: 0.9443 - val_loss: 0.1963 - val_accuracy: 0.9290
```

In [ ]:

```python
eval_test_data = model.evaluate(X_test, y_test)
print("Test Loss: ", eval_test_data[0])
print("Test Accuracy: ", eval_test_data[1])
accuracy = train_history.history['accuracy']
val_accuracy = train_history.history['val_accuracy']
loss = train_history.history['loss']
```

```
196/196 [==============================] - 1s 3ms/step - loss: 0.1977 - accuracy: 0.9250
Test Loss:  0.1976928859949112
Test Accuracy:  0.9249680638313293
```

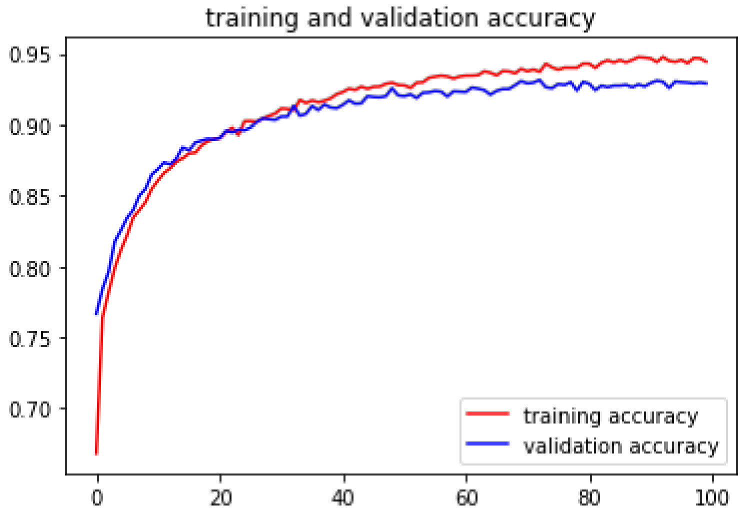
In [ ]:

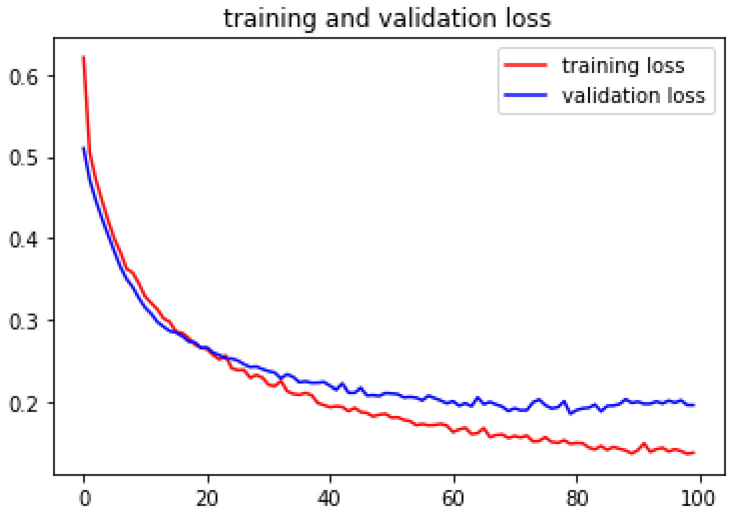
```python
val_loss=train_history.history['val_loss']
epochs=range(len(accuracy))
plt.plot(epochs,accuracy,'r',label='training accuracy')
plt.plot(epochs,val_accuracy,'b',label='validation accuracy')
plt.title('training and validation accuracy')
plt.legend()
plt.figure()
```

```
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()
```

Out[ ]:

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






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

Calculating the area under receiver operating characteristics curve

In [ ]:

```
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("Area under the ROC Curve: ", auc_score)
```

```
Area under the ROC Curve:  0.9779623995711011
```