In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
```

In [2]:
```python
#Load the dataset that is given to you
df = pd.read_csv('/content/loan_data (1).csv')
df
```

Out[2]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAM
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	351000.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	1129500.0	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	135000.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	297000.0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	513000.0	
...	...	...	...	...	...	...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	N	0	157500.0	254700.0	27558.0	225000.0	
307507	456252	0	Cash loans	F	N	Y	0	72000.0	269550.0	12001.5	225000.0	
307508	456253	0	Cash loans	F	N	Y	0	153000.0	677664.0	29979.0	585000.0	
307509	456254	1	Cash loans	F	N	Y	0	171000.0	370107.0	20205.0	319500.0	
307510	456255	0	Cash loans	F	N	N	0	157500.0	675000.0	49117.5	675000.0	

307511 rows × 122 columns

In [3]:
```python
df.shape
```

Out[3]:
```
(307511, 122)
```

In [4]:
```python
df.info
```

Out[4]:
```
<bound method DataFrame.info of         SK_ID_CURR  ...  AMT_REQ_CREDIT_BUREAU_YEAR
0           100002  ...                         1.0
1           100003  ...                         0.0
2           100004  ...                         0.0
3           100006  ...                         NaN
4           100007  ...                         0.0
...            ...  ...                         ...
307506      456251  ...                         NaN
307507      456252  ...                         NaN
307508      456253  ...                         1.0
307509      456254  ...                         0.0
307510      456255  ...                         1.0

[307511 rows x 122 columns]>
```

In [5]:
```python
df.columns
```

Out[5]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY',
       ...
       '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'],
      dtype='object', length=122)
```

In [6]:
```python
#Check for null values in the dataset
df.isnull().sum().sum()
for i in df.columns:
  df[i] = df[i].fillna(df[i].mean)
```

In [7]:
```python
df.isnull().sum().sum()
```

Out[7]:
```
0
```

In [8]:
```python
#Print percentage of default to payer of the dataset for the TARGET column
df['TARGET'].value_counts()
```

Out[8]:
```
0    282686
1     24825
Name: TARGET, dtype: int64
```

In [9]:
```python
#drop object data
for i in df.columns:
    if(df[i].dtype)=='object':
        df.drop(columns=i, inplace=True)
```

In [10]:
```python
df.dtypes.value_counts()
```

Out[10]:
```
int64      41
float64     4
dtype: int64
```

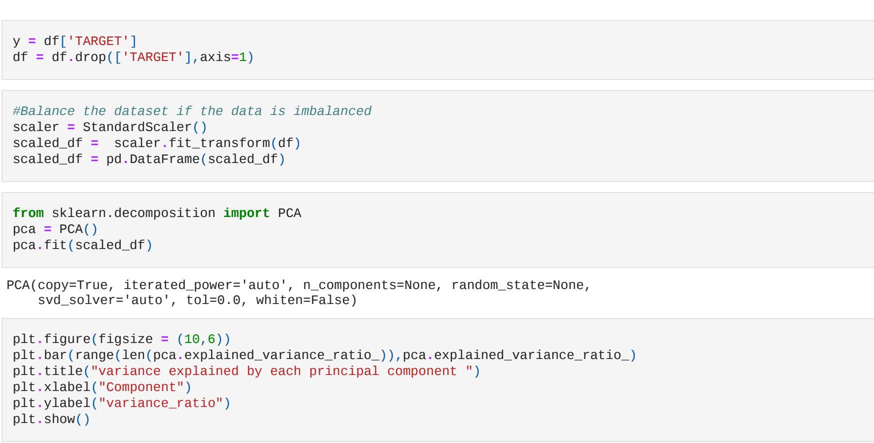
In [11]:
```python
df.corr()
```

Out[11]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_
SK_ID_CURR	1.000000	-0.002108	-0.001129	-0.001820	-0.000343	0.000849	-0.001500	0.001366	-0.000973	
TARGET	-0.002108	1.000000	0.019187	-0.003982	-0.030369	-0.037227	0.078239	-0.044932	0.041975	
CNT_CHILDREN	-0.001129	0.019187	1.000000	0.012882	0.002145	-0.025573	0.330938	-0.239818	0.183395	
AMT_INCOME_TOTAL	-0.001820	-0.003982	0.012882	1.000000	0.156870	0.074796	0.027261	-0.064223	0.027805	
AMT_CREDIT	-0.000343	-0.030369	0.002145	0.156870	1.000000	0.099738	-0.055436	-0.066838	0.009621	
REGION_POPULATION_RELATIVE	0.000849	-0.037227	-0.025573	0.074796	0.099738	1.000000	-0.029582	-0.003980	-0.053820	
DAYS_BIRTH	-0.001500	0.078239	0.330938	0.027261	-0.055436	-0.029582	1.000000	-0.615864	0.331912	
DAYS_EMPLOYED	0.001366	-0.044932	-0.239818	-0.064223	-0.066838	-0.003980	-0.615864	1.000000	-0.210242	
DAYS_REGISTRATION	-0.000973	0.041975	0.183395	0.027805	0.009621	-0.053820	0.331912	-0.210242	1.000000	
DAYS_ID_PUBLISH	-0.000384	0.051457	-0.028019	0.008506	-0.006575	-0.003993	0.272691	-0.272378	0.101896	
FLAG_MOBIL	0.002804	0.000534	0.001041	0.000325	0.001436	0.001333	-0.003084	0.000818	-0.000100	
FLAG_EMP_PHONE	-0.001337	0.045982	0.240714	0.063994	0.065519	0.004045	0.619888	-0.999755	0.212361	
FLAG_WORK_PHONE	-0.000415	0.028524	0.055630	-0.017193	-0.021085	-0.015628	0.172457	-0.233972	0.058283	
FLAG_CONT_MOBILE	0.002815	0.000370	-0.000794	-0.008290	0.023653	-0.012478	-0.014985	0.012745	-0.003848	
FLAG_PHONE	0.002753	-0.023806	-0.029906	0.000159	0.026213	0.090939	-0.042402	0.015291	-0.075188	
FLAG_EMAIL	0.000281	-0.001758	0.022619	0.038378	0.016632	0.040012	0.088208	-0.062112	0.034388	
REGION_RATING_CLIENT	-0.001075	0.058899	0.025423	-0.085465	-0.101776	-0.532877	0.009361	0.032750	0.080210	
REGION_RATING_CLIENT_W_CITY	-0.001138	0.060893	0.024781	-0.091735	-0.110915	-0.531535	0.008073	0.034624	0.074038	
HOUR_APPR_PROCESS_START	0.000350	-0.024166	-0.007292	0.036459	0.052738	0.171285	0.091064	-0.091138	-0.010908	
REG_REGION_NOT_LIVE_REGION	-0.000283	0.005576	-0.013319	0.031191	0.024010	0.002118	0.065486	-0.035803	0.028213	
REG_REGION_NOT_WORK_REGION	0.001097	0.006942	0.008185	0.062340	0.051929	0.056944	0.095819	-0.107150	0.036787	
LIVE_REGION_NOT_WORK_REGION	0.002903	0.002819	0.014835	0.058059	0.052609	0.081016	0.069567	-0.095539	0.027560	
REG_CITY_NOT_LIVE_CITY	-0.001885	0.044395	0.020072	0.003574	-0.026886	-0.050499	0.180382	-0.090516	0.064334	
REG_CITY_NOT_WORK_CITY	-0.001582	0.050994	0.070650	0.006431	-0.018856	-0.044057	0.242401	-0.254559	0.099874	
LIVE_CITY_NOT_WORK_CITY	0.000067	0.032518	0.069957	0.008285	0.000081	-0.015188	0.158882	-0.218101	0.072658	
FLAG_DOCUMENT_2	0.000700	0.005417	0.001786	-0.001000	0.008905	-0.003040	-0.001191	-0.003090	-0.004466	
FLAG_DOCUMENT_3	-0.003411	0.044346	0.056837	-0.016751	0.096365	-0.084644	0.109666	-0.249082	0.033740	
FLAG_DOCUMENT_4	-0.004139	-0.002672	-0.003709	0.000529	0.000630	0.008102	-0.002789	0.000437	-0.004221	
FLAG_DOCUMENT_5	-0.001097	-0.000316	-0.016737	0.001507	-0.011756	0.016032	0.016774	-0.020041	-0.001243	
FLAG_DOCUMENT_6	0.002121	-0.028602	-0.157024	-0.045878	-0.046717	0.000728	-0.407160	0.597484	-0.137769	
FLAG_DOCUMENT_7	-0.002694	-0.001520	-0.001498	0.003825	-0.004040	0.002436	0.001469	-0.002221	0.001366	
FLAG_DOCUMENT_8	0.001809	-0.008040	0.051697	0.072451	0.082819	0.088523	0.113243	-0.121571	0.057041	
FLAG_DOCUMENT_9	0.001505	-0.004352	-0.001997	0.018389	0.022602	0.038434	0.018213	-0.023834	0.009898	
FLAG_DOCUMENT_10	-0.000815	-0.001414	-0.002756	0.000290	-0.003100	0.001992	-0.000168	-0.000439	0.000292	
FLAG_DOCUMENT_11	-0.002012	-0.004229	-0.005318	0.002315	0.028986	0.024632	0.044716	-0.028915	0.014731	
FLAG_DOCUMENT_12	-0.001045	-0.000756	0.000293	0.002540	0.003857	-0.000404	0.000236	-0.001225	0.001682	
FLAG_DOCUMENT_13	0.000896	-0.011583	0.003945	0.022747	0.052429	0.031186	0.026333	-0.025778	0.015470	
FLAG_DOCUMENT_14	-0.001077	-0.009464	-0.005459	0.020708	0.048828	0.032202	0.030733	-0.023383	0.011636	
FLAG_DOCUMENT_15	0.002604	-0.006536	0.003609	0.010793	0.032252	0.009446	0.012953	-0.014285	0.008195	
FLAG_DOCUMENT_16	-0.000724	-0.011615	0.010662	0.007269	0.061925	0.006313	0.024044	-0.042905	0.025935	
FLAG_DOCUMENT_17	0.001450	-0.003378	0.000773	0.002230	0.011743	0.007414	0.007877	-0.007187	0.002102	
FLAG_DOCUMENT_18	0.000509	-0.007952	0.004031	0.003130	0.034329	0.013367	0.044498	-0.040396	0.017040	
FLAG_DOCUMENT_19	0.000167	-0.001358	0.000864	0.002408	0.021082	0.002364	0.003614	-0.010040	0.004764	
FLAG_DOCUMENT_20	0.001073	0.000215	0.000988	0.000242	0.031023	-0.000045	0.009015	-0.010135	0.006971	
FLAG_DOCUMENT_21	0.000282	0.003709	-0.002450	-0.000589	-0.016148	0.003437	0.027282	-0.008396	0.003859	

In [36]:
```python
#Plot the balanced data or imbalanced data
plt.figure(figsize=(15,7))
plt.plot(df.iloc[:10,:5])
plt.legend(df.columns)
```

Out[36]:
```
<matplotlib.legend.Legend at 0x7fc7292c96d0>
```



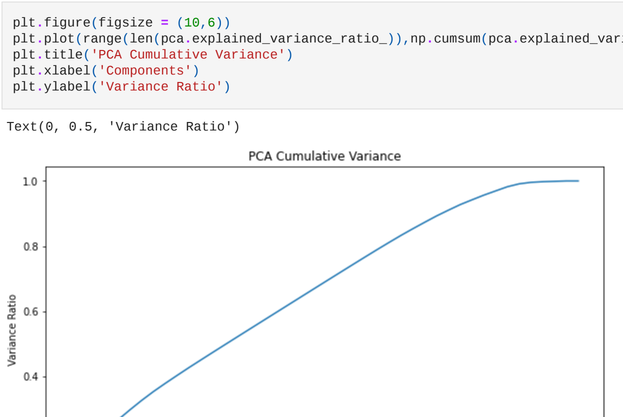
In [13]:
```python
y = df['TARGET']
df = df.drop(['TARGET'],axis=1)
```

In [14]:
```python
#Balance the dataset if the data is imbalanced
scaler = StandardScaler()
scaled_df =  scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df)
```

In [15]:
```python
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(scaled_df)
```

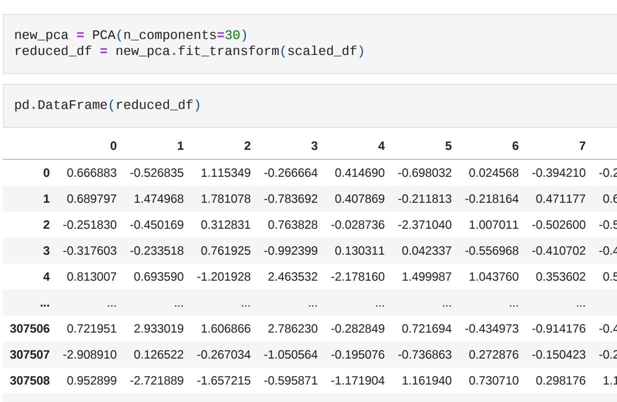
Out[15]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [16]:
```python
plt.figure(figsize = (10,6))
plt.bar(range(len(pca.explained_variance_ratio_)),pca.explained_variance_ratio_)
plt.title("variance explained by each principal component ")
plt.xlabel("Component")
plt.ylabel("variance_ratio")
plt.show()
```



In [17]:
```python
plt.figure(figsize = (10,6))
plt.plot(range(len(pca.explained_variance_ratio_)),np.cumsum(pca.explained_variance_ratio_))
plt.title('PCA Cumulative Variance')
plt.xlabel('Components')
plt.ylabel('Variance Ratio')
```

Out[17]:
```
Text(0, 0.5, 'Variance Ratio')
```



more than 80% of the data will be described with 30 components

In [18]:
```python
new_pca = PCA(n_components=30)
reduced_df = new_pca.fit_transform(scaled_df)
```

In [19]:
```python
pd.DataFrame(reduced_df)
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0.666883	-0.526835	1.115349	-0.266664	0.414690	-0.698032	0.024568	-0.394210	-0.246000	0.384523	0.022293	-0.048688	0.190382	-0.091637	1.032737	0.430735	-0.181029	-0.188488	0.046543
1	0.689797	1.474968	1.781078	-0.783692	0.407869	-0.211813	-0.218164	0.471177	0.607534	0.709000	0.492444	0.010305	0.165757	0.030448	1.143280	0.256139	-0.298867	-0.163691	0.037327
2	-0.251830	-0.450169	0.312831	0.763828	-0.028736	-2.371040	1.007011	-0.502600	-0.509895	0.350942	-0.111377	0.224132	-0.041664	0.228107	1.036290	0.563493	-0.296723	-0.140870	0.016468
3	-0.317603	-0.233518	0.761925	-0.992399	0.130311	0.042337	-0.556968	-0.410702	-0.448516	1.051800	0.613676	0.480019	-0.209297	-0.175080	1.089270	0.361748	-0.079407	-0.220630	0.164509
4	0.813007	0.693590	-1.201928	2.463532	-2.178160	1.499987	1.043760	0.353602	0.538255	0.701821	0.237900	0.855974	-0.279591	0.178014	1.162896	0.491029	-0.410951	-0.131698	0.053642
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
307506	0.721951	2.933019	1.606866	2.786230	-0.282849	0.721694	-0.434973	-0.914176	-0.497971	0.736294	0.749348	1.269338	-0.953643	0.088023	-1.116098	-0.186178	-0.047290	0.007021	-0.123985
307507	-2.908910	0.126522	-0.267034	-1.050564	-0.195076	-0.736863	0.272876	-0.150423	-0.266103	-0.430890	-0.589939	-0.111480	-0.079835	-0.000982	-1.227329	-0.189159	0.277571	0.107405	-0.129009
307508	0.952899	-2.721889	-1.657215	-0.595871	-1.171904	1.161940	0.730710	0.298176	1.125497	0.454582	-0.296271	-1.071784	1.417880	-0.775920	-1.622358	0.105249	0.379257	0.505491	-0.263079
307509	1.755051	-1.313699	-1.115022	0.096300	-0.768204	0.290846	-2.731108	0.257145	-0.123146	0.581427	0.489705	0.006109	-0.256361	-0.230523	-1.230789	-0.344947	0.197117	0.031174	-0.105666
307510	2.282829	3.202507	0.787527	-1.094166	-3.178154	-1.668947	0.300009	0.349727	-0.135153	0.491291	0.337506	0.262084	-0.497640	0.308634	-1.115869	-0.306477	0.044784	0.069932	-0.130653

307511 rows × 30 columns

Create The model

In [20]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df,y, train_size=0.8, random_state =0)
```

In [21]:
```python
model = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation = 'relu'),
        tf.keras.layers.Dense(64, activation = 'relu'),
        tf.keras.layers.Dense(10, activation = 'softmax')
])
```

In [22]:
```python
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
EPOCHS = 3

history = model.fit(df,y, epochs = EPOCHS)
```

```
Epoch 1/3
9610/9610 [==============================] - 22s 2ms/step - loss: 203.2691 - accuracy: 0.8550
Epoch 2/3
9610/9610 [==============================] - 18s 2ms/step - loss: 29.0451 - accuracy: 0.9028
Epoch 3/3
9610/9610 [==============================] - 19s 2ms/step - loss: 0.4061 - accuracy: 0.9193
```

In [23]:
```python
pred = model.predict(X_test)
```

In [38]:
```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               5760
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 14,666
Trainable params: 14,666
Non-trainable params: 0
_________________________________________________________________
```

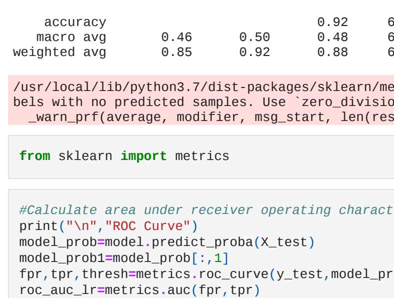
In [39]:
```python
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, y_train)
pre = model.predict(X_test)
pre
```

Out[39]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [41]:
```python
# Calculate Sensitivity as a metrice of Random Forest

cnf_matrix = metrics.confusion_matrix(y_test, pre)
sns.heatmap(pd.DataFrame(cnf_matrix),annot=True, cmap="YlGnBu" ,fmt='g')
plt.ylabel('Actual values')
plt.xlabel('Prediction')
```

Out[41]:
```
Text(0.5, 15.0, 'Prediction')
```



In [42]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, pre))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     56648
           1       0.00      0.00      0.00      4855

    accuracy                           0.92     61503
   macro avg       0.46      0.50      0.48     61503
weighted avg       0.85      0.92      0.88     61503
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in la
bels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [27]:
```python
from sklearn import metrics
```

In [43]:
```python
#Calculate area under receiver operating characteristics curve
print("\n","ROC Curve")
model_prob=model.predict_proba(X_test)
model_prob1=model_prob[:,1]
fpr,tpr,thresh=metrics.roc_curve(y_test,model_prob1)
roc_auc_lr=metrics.auc(fpr,tpr)
plt.figure(dpi=80)
plt.title("ROC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr,tpr,'b',label='AUC Score = %0.2f'%roc_auc_lr)
plt.plot(fpr,fpr,'r--',color='red')
plt.legend()
```

```
 ROC Curve
```

Out[43]:
```
<matplotlib.legend.Legend at 0x7fc72b6dc1d0>
```