Classification using Logistc Regression

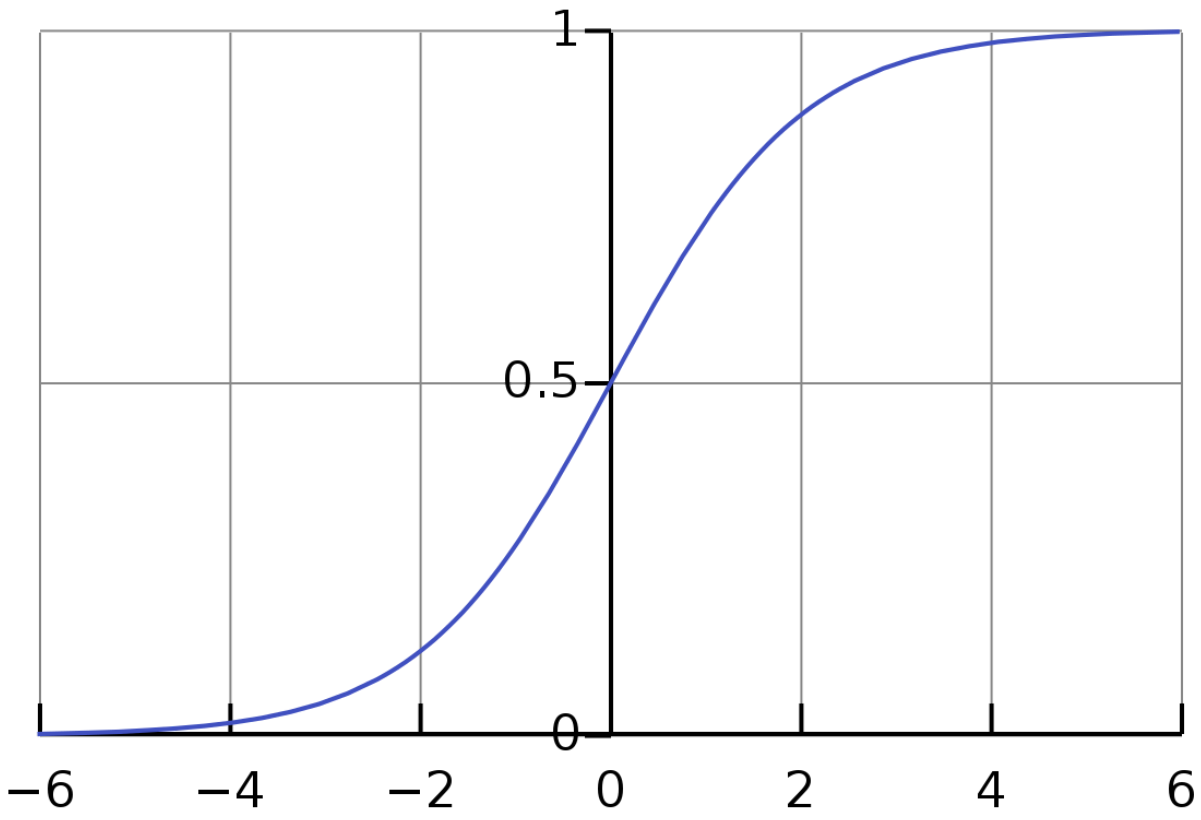
Let X1, X2, ..., Xn be n number of features.

Z = b0 + b1 X1 + ...+ bn Xn

f(Z) = e^Z/(1 + e^Z) ....(f(Z) always less than 1), (e^Z always greater than 0)

Therefore, f(Z) lies in between 0 and 1 for every value of Z

Such a function is called the logistic function or Sigmoid function.



f(Z) is the probability

If 0<f(Z)<0.5 then Z ==> belongs to the class 0

If 0.5<F(z)<1 then Z ==> belongs to the class 1

$f(Z) = \frac{e^Z}{1+e^Z}$

Logistic Regression for German Credit platform

Sourcing the data

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

In [2]:
```
# Dataset

gc = pd.read_csv("https://online.stat.psu.edu/onlinecourses/sites/stat508/files/german_credit.csv")
gc
```

Out[2]:

	Creditability	Account Balance	Duration of Credit (month)	Payment Status of Previous Credit	Purpose	Credit Amount	Value Savings/Stocks	Length of current employment	Instalment per cent	Sex & Marital Status	...	Duration in Current address	Most valuable available asset	Age (years)	Concurrent Credits
0	1	1	18	4	2	1049	1	2	4	2	...	4	2	21	3
1	1	1	9	4	0	2799	1	3	2	3	...	2	1	36	3
2	1	2	12	2	9	841	2	4	2	2	...	4	1	23	3
3	1	1	12	4	0	2122	1	3	3	3	...	2	1	39	3
4	1	1	12	4	0	2171	1	3	4	3	...	4	2	38	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0	1	24	2	3	1987	1	3	2	3	...	4	1	21	3
996	0	1	24	2	0	2303	1	5	4	3	...	1	1	45	3
997	0	4	21	4	0	12680	5	5	4	3	...	4	4	30	3
998	0	2	12	2	3	6468	5	1	2	3	...	1	4	52	3
999	0	1	30	2	2	6350	5	5	4	3	...	4	2	31	3

1000 rows × 21 columns

In [3]:
```
gc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Creditability                     1000 non-null   int64
 1   Account Balance                   1000 non-null   int64
 2   Duration of Credit (month)        1000 non-null   int64
 3   Payment Status of Previous Credit 1000 non-null   int64
 4   Purpose                           1000 non-null   int64
 5   Credit Amount                     1000 non-null   int64
 6   Value Savings/Stocks              1000 non-null   int64
 7   Length of current employment      1000 non-null   int64
 8   Instalment per cent               1000 non-null   int64
 9   Sex & Marital Status              1000 non-null   int64
 10  Guarantors                        1000 non-null   int64
 11  Duration in Current address       1000 non-null   int64
 12  Most valuable available asset     1000 non-null   int64
 13  Age (years)                       1000 non-null   int64
 14  Concurrent Credits                1000 non-null   int64
 15  Type of apartment                 1000 non-null   int64
 16  No of Credits at this Bank        1000 non-null   int64
 17  Occupation                        1000 non-null   int64
 18  No of dependents                  1000 non-null   int64
 19  Telephone                         1000 non-null   int64
 20  Foreign Worker                    1000 non-null   int64
dtypes: int64(21)
memory usage: 164.2 KB
```

Preprocessing of the data

In [4]:
```
# Target

y = gc['Creditability']
y
```

Out[4]:
```
0      1
1      1
2      1
3      1
4      1
      ..
995    0
996    0
997    0
998    0
999    0
Name: Creditability, Length: 1000, dtype: int64
```

In [5]:
```
y.unique() # no. of unique values
```

Out[5]:
```
array([1, 0], dtype=int64)
```

Only 2 classes ==> Binary Classification

In [6]:
```
# Features

X = gc.drop(['Creditability'], axis = 1)
X
```

Out[6]:

	Account Balance	Duration of Credit (month)	Payment Status of Previous Credit	Purpose	Credit Amount	Value Savings/Stocks	Length of current employment	Instalment per cent	Sex & Marital Status	Guarantors	Duration in Current address	Most valuable available asset	Age (years)	Concurrent Credits	ap
0	1	18	4	2	1049	1	2	4	2	1	4	2	21	3	
1	1	9	4	0	2799	1	3	2	3	1	2	1	36	3	
2	2	12	2	9	841	2	4	2	2	1	4	1	23	3	
3	1	12	4	0	2122	1	3	3	3	1	2	1	39	3	
4	1	12	4	0	2171	1	3	4	3	1	4	2	38	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	1	24	2	3	1987	1	3	2	3	1	4	1	21	3	
996	1	24	2	0	2303	1	5	4	3	2	1	1	45	3	
997	4	21	4	0	12680	5	5	4	3	1	4	4	30	3	
998	2	12	2	3	6468	5	1	2	3	1	1	4	52	3	
999	1	30	2	2	6350	5	5	4	3	1	4	2	31	3	

1000 rows × 20 columns

In [7]:
```
X.shape
```

Out[7]:
```
(1000, 20)
```

In [8]:
```
X.columns
```

Out[8]:
```
Index(['Account Balance', 'Duration of Credit (month)',
       'Payment Status of Previous Credit', 'Purpose', 'Credit Amount',
       'Value Savings/Stocks', 'Length of current employment',
       'Instalment per cent', 'Sex & Marital Status', 'Guarantors',
       'Duration in Current address', 'Most valuable available asset',
       'Age (years)', 'Concurrent Credits', 'Type of apartment',
       'No of Credits at this Bank', 'Occupation', 'No of dependents',
       'Telephone', 'Foreign Worker'],
      dtype='object')
```

In [9]:
```
category_cols = ['Account Balance','Payment Status of Previous Credit',
                 'Purpose', 'Value Savings/Stocks', 'Length of current employment',
                 'Instalment per cent', 'Sex & Marital Status', 'Guarantors',
                 'Duration in Current address', 'Most valuable available asset',
                 'Concurrent Credits', 'Type of apartment','No of Credits at this Bank',
                 'Occupation', 'No of dependents','Telephone', 'Foreign Worker']

#'Duration of Credit (month)','Credit Amount','Age (years)' are dropped since they are not categorical
```

In [10]:
```
len(category_cols)
```

Out[10]:
```
17
```

In [11]:
```
# Converting categorical features to numeric

X_1 = pd.get_dummies(X, columns = category_cols)
```

In [12]:
```
X_1.shape
```

Out[12]:
```
(1000, 71)
```

columns increased to 71

In [13]:
```
# Adding constant

import statsmodels.api as sm
X_1 = sm.add_constant(X_1)
X_1.shape
```

```
C:\Users\Urvi Sharma\anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: In a future version of pandas
all arguments of concat except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Out[13]: (1000, 72)

In [14]:
```
X_1
```

Out[14]:

	const	Duration of Credit (month)	Credit Amount	Age (years)	Account Balance_1	Account Balance_2	Account Balance_3	Account Balance_4	Payment Status of Previous Credit_0	Payment Status of Previous Credit_1	...	Occupation_1	Occupation_2	Occupation_3	Occu
0	1.0	18	1049	21	1	0	0	0	0	0	...	0	0	1	
1	1.0	9	2799	36	1	0	0	0	0	0	...	0	0	1	
2	1.0	12	841	23	0	1	0	0	0	0	...	0	1	0	
3	1.0	12	2122	39	1	0	0	0	0	0	...	0	1	0	
4	1.0	12	2171	38	1	0	0	0	0	0	...	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
995	1.0	24	1987	21	1	0	0	0	0	0	...	0	1	0	
996	1.0	24	2303	45	1	0	0	0	0	0	...	0	0	1	
997	1.0	21	12680	30	0	0	0	1	0	0	...	0	0	0	
998	1.0	12	6468	52	0	1	0	0	0	0	...	0	0	0	
999	1.0	30	6350	31	1	0	0	0	0	0	...	0	0	1	

1000 rows × 72 columns

In [15]:
```
X_1.iloc[0:10, 0:20]
```

Out[15]:

	const	Duration of Credit (month)	Credit Amount	Age (years)	Account Balance_1	Account Balance_2	Account Balance_3	Account Balance_4	Payment Status of Previous Credit_0	Payment Status of Previous Credit_1	Payment Status of Previous Credit_2	Payment Status of Previous Credit_3	Payment Status of Previous Credit_4	Purpose_0	Purpose_1	P
0	1.0	18	1049	21	1	0	0	0	0	0	0	0	1	0	0	
1	1.0	9	2799	36	1	0	0	0	0	0	0	0	1	1	0	
2	1.0	12	841	23	0	1	0	0	0	0	1	0	0	0	0	
3	1.0	12	2122	39	1	0	0	0	0	0	0	0	1	1	0	
4	1.0	12	2171	38	1	0	0	0	0	0	0	0	1	1	0	
5	1.0	10	2241	48	1	0	0	0	0	0	0	0	1	1	0	
6	1.0	8	3398	39	1	0	0	0	0	0	0	0	1	1	0	
7	1.0	6	1361	40	1	0	0	0	0	0	0	0	1	1	0	
8	1.0	18	1098	65	0	0	0	1	0	0	0	0	1	0	0	
9	1.0	24	3758	23	0	1	0	0	0	0	1	0	0	0	0	

In [16]:
```
X_1.iloc[0:10, 20:40]
```

Out[16]:

	Purpose_8	Purpose_9	Purpose_10	Value Savings/Stocks_1	Value Savings/Stocks_2	Value Savings/Stocks_3	Value Savings/Stocks_4	Value Savings/Stocks_5	Length of current employment_1	Length curre employment
0	0	0	0	1	0	0	0	0	0	
1	0	0	0	1	0	0	0	0	0	
2	0	1	0	0	1	0	0	0	0	
3	0	0	0	1	0	0	0	0	0	
4	0	0	0	1	0	0	0	0	0	
5	0	0	0	1	0	0	0	0	0	
6	0	0	0	1	0	0	0	0	0	
7	0	0	0	1	0	0	0	0	0	
8	0	0	0	1	0	0	0	0	1	
9	0	0	0	0	0	1	0	0	1	

In [17]:
```
X_1.iloc[0:10, 40:60]
```

Out[17]:

	Sex & Marital Status_4	Guarantors_1	Guarantors_2	Guarantors_3	Duration in Current address_1	Duration in Current address_2	Duration in Current address_3	Duration in Current address_4	Most valuable available asset_1	Most valuable available asset_2	Most valuable available asset_3	Most valuable available asset_4	Concurrent Credits_1	Cor Cr
0	0	1	0	0	0	0	0	1	0	1	0	0	0	
1	0	1	0	0	0	1	0	0	1	0	0	0	0	
2	0	1	0	0	0	0	0	1	1	0	0	0	0	
3	0	1	0	0	0	1	0	0	1	0	0	0	0	
4	0	1	0	0	0	0	0	1	0	1	0	0	1	
5	0	1	0	0	0	0	1	0	1	0	0	0	0	
6	0	1	0	0	0	0	0	1	1	0	0	0	0	
7	0	1	0	0	0	0	0	1	1	0	0	0	0	
8	0	1	0	0	0	0	0	1	0	0	1	0	0	
9	0	1	0	0	0	0	0	1	0	0	0	1	0	

In [18]:
```
X_1.iloc[0:10, 60:]
```

Out[18]:

	No of Credits at this Bank_3	No of Credits at this Bank_4	Occupation_1	Occupation_2	Occupation_3	Occupation_4	No of dependents_1	No of dependents_2	Telephone_1	Telephone_2	Foreign Worker_1	Foreign Worker_2
0	0	0	0	0	1	0	1	0	1	0	1	0
1	0	0	0	0	1	0	0	1	1	0	1	0
2	0	0	0	1	0	0	1	0	1	0	1	0
3	0	0	0	1	0	0	0	1	1	0	0	1
4	0	0	0	1	0	0	1	0	1	0	0	1
5	0	0	0	1	0	0	0	1	1	0	0	1
6	0	0	0	1	0	0	1	0	1	0	0	1
7	0	0	0	1	0	0	0	1	1	0	0	1
8	0	0	1	0	0	0	1	0	1	0	1	0
9	0	0	1	0	0	0	1	0	1	0	1	0

Splitting to training and testing

In [19]:
```
from sklearn.model_selection import train_test_split

X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_1, y, test_size = 0.2, random_state = 10)

X_train_1.shape, X_test_1.shape, y_train_1.shape, y_test_1.shape
```

Out[19]: ((800, 72), (200, 72), (800,), (200,))

In [20]:
```
X_train_1
```

Out[20]:

	const	Duration of Credit (month)	Credit Amount	Age (years)	Account Balance_1	Account Balance_2	Account Balance_3	Account Balance_4	Payment Status of Previous Credit_0	Payment Status of Previous Credit_1	...	Occupation_1	Occupation_2	Occupation_3	Occu
188	1.0	36	2273	32	0	1	0	0	0	0	...	0	0	1	
194	1.0	12	1262	25	0	0	0	1	0	0	...	0	0	1	
225	1.0	24	1516	43	0	0	0	1	0	0	...	0	1	0	
580	1.0	24	1546	24	1	0	0	0	0	1	...	0	1	0	
428	1.0	12	1200	23	1	0	0	0	0	0	...	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
320	1.0	10	1597	40	0	0	0	1	0	0	...	0	1	0	
527	1.0	12	2930	27	0	1	0	0	0	0	...	0	0	1	
996	1.0	24	2303	45	1	0	0	0	0	0	...	0	0	1	
125	1.0	24	5103	47	0	0	0	1	0	0	...	0	0	1	
265	1.0	22	2675	40	0	0	0	1	0	0	...	0	0	1	

800 rows × 72 columns

Building the model

In [21]:
```
logR_1 = sm.Logit(y_train_1, X_train_1) #Logit function is a part of statsmodels.api

# Fit the model
logR_1 = logR_1.fit()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.434640
         Iterations: 35

C:\Users\Urvi Sharma\anaconda3\lib\site-packages\statsmodels\base\model.py:566: ConvergenceWarning: Maximum Likelihood optimiza
tion failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Diagnosing the model

In [22]:
```
logR_1.summary2()
```

Out[22]:

Model:	Logit	Pseudo R-squared:	0.280
Dependent Variable:	Creditability	AIC:	805.4233
Date:	2022-10-17 12:46	BIC:	1063.0770
No. Observations:	800	Log-Likelihood:	-347.71
Df Model:	54	LL-Null:	-482.61
Df Residuals:	745	LLR p-value:	1.9042e-30
Converged:	0.0000	Scale:	1.0000
No. Iterations:	35.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	-20.3529	nan	nan	nan	nan	nan
Duration of Credit (month)	-0.0245	0.0108	-2.2707	0.0232	-0.0456	-0.0033
Credit Amount	-0.0001	0.0001	-2.7644	0.0057	-0.0002	-0.0000

variables with p value <0.05 are important features

In [23]:
```
imp_features = ['Duration of Credit (month)', 'Credit Amount', 'Age (years)']
```

In [24]:
```
imp_features
```

Out[24]:
```
['Duration of Credit (month)', 'Credit Amount', 'Age (years)']
```

In [25]:
```
# Update the feature set

X_2 = X_1[imp_features]
```

In [26]:
```
X_2
```

Out[26]:

	Duration of Credit (month)	Credit Amount	Age (years)
0	18	1049	21
1	9	2799	36
2	12	841	23
3	12	2122	39
4	12	2171	38
...	...	...	...
995	24	1987	21
996	24	2303	45
997	21	12680	30
998	12	6468	52
999	30	6350	31

1000 rows × 3 columns

Continue with revised feature set

In [27]:
```
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_2, y, test_size = 0.2, random_state = 10)

X_train_1.shape, X_test_2.shape, y_train_2.shape, y_test_2.shape
```

Out[27]:
```
((800, 72), (200, 3), (800,), (200,))
```

Building the model #2

In [28]:
```
logR_2 = sm.Logit(y_train_2, X_train_2)
logR_2 = logR_2.fit()
logR_2.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.574603
         Iterations 5
```

Out[28]:

Model:	Logit	Pseudo R-squared:	0.048
Dependent Variable:	Creditability	AIC:	925.3648
Date:	2022-10-17 12:46	BIC:	939.4186
No. Observations:	800	Log-Likelihood:	-459.68
Df Model:	2	LL-Null:	-482.61
Df Residuals:	797	LLR p-value:	1.0992e-10
Converged:	1.0000	Scale:	1.0000
No. Iterations:	5.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Duration of Credit (month)	-0.0226	0.0078	-2.8827	0.0039	-0.0379	-0.0072
Credit Amount	-0.0001	0.0000	-1.8939	0.0582	-0.0001	0.0000
Age (years)	0.0452	0.0044	10.2041	0.0000	0.0365	0.0539

In [29]:
```
# Significant features

imp_features_2 = ['Duration of Credit (month)', 'Age (years)'] # features with p value < 0.05

X_3 = X_2[imp_features_2]
```

In [30]:
```
X_3
```

Out[30]:

	Duration of Credit (month)	Age (years)
0	18	21
1	9	36
2	12	23
3	12	39
4	12	38
...	...	...
995	24	21
996	24	45
997	21	30
998	12	52
999	30	31

1000 rows × 2 columns

Continue with revised feature set

In [31]:
```
X_train_3, X_test_3, y_train_3, y_test_3 = train_test_split(X_3, y, test_size = 0.2, random_state = 10)

X_train_3.shape, X_test_3.shape, y_train_3.shape, y_test_3.shape
```

Out[31]:
```
((800, 2), (200, 2), (800,), (200,))
```

Building the model #3

In [32]:
```
logR_3 = sm.Logit(y_train_3, X_train_3)
logR_3 = logR_3.fit()
logR_3.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.576819
         Iterations 5
```

Out[32]:

Model:	Logit	Pseudo R-squared:	0.044
Dependent Variable:	Creditability	AIC:	926.9100
Date:	2022-10-17 12:46	BIC:	936.2792
No. Observations:	800	Log-Likelihood:	-461.45
Df Model:	1	LL-Null:	-482.61
Df Residuals:	798	LLR p-value:	7.7598e-11
Converged:	1.0000	Scale:	1.0000
No. Iterations:	5.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
Duration of Credit (month)	-0.0319	0.0061	-5.2237	0.0000	-0.0439	-0.0200
Age (years)	0.0444	0.0044	10.1043	0.0000	0.0358	0.0530

Model Building is complete (all p values are less than 0.05, 0 in the last iteration)

In [33]:
```
logR_3.params
```

Out[33]:
```
Duration of Credit (month)   -0.031934
Age (years)                   0.044396
dtype: float64
```

The final LR Model:

Creditability = Duration of (month) * -0.031934 + Age (years) * 0.044396

` $f(Z) = \frac{e^Z}{1+e^Z}$

Prediction using the model

In [34]:
```
y_pred = logR_3.predict(X_test_3)
y_pred
```

Out[34]:
```
841    0.690284
956    0.884680
544    0.850290
173    0.657965
759    0.512259
         ...
274    0.687278
192    0.385243
398    0.807991
450    0.927365
520    0.523344
Length: 200, dtype: float64
```

These are probabilities

Creditability of records with probability<0.5 will be 0, for others it will be 1

In [35]:
```
y_test_3
```

Out[35]:
```
841    0
956    0
544    1
173    1
759    0
      ..
274    1
192    1
398    1
450    0
520    0
Name: Creditability, Length: 200, dtype: int64
```

In [36]:
```
# Creating a dataframe

pred_df = pd.DataFrame({'Actual Class':y_test_3,'Predicted Probability':y_pred})
```

In [37]:
```
pred_df
```

Out[37]:

	Actual Class	Predicted Probability
841	0	0.690284
956	0	0.884680
544	1	0.850290
173	1	0.657965
759	0	0.512259
...	...	...
274	1	0.687278
192	1	0.385243
398	1	0.807991
450	0	0.927365
520	0	0.523344

200 rows × 2 columns

In [38]:
```
# Adding a column to the dataframe

pred_df['Predicted Class'] = pred_df['Predicted Probability'].map(lambda x: 1 if x>0.5 else 0)

pred_df
```

Out[38]:

	Actual Class	Predicted Probability	Predicted Class
841	0	0.690284	1
956	0	0.884680	1
544	1	0.850290	1
173	1	0.657965	1
759	0	0.512259	1
...	...	...	...
274	1	0.687278	1
192	1	0.385243	0
398	1	0.807991	1
450	0	0.927365	1
520	0	0.523344	1

200 rows × 3 columns

Performance measures

Sensitivity = P(Preditcted Class= +ve/Actual Class= +ve) = TP/(TP + FN)

Specificity = P(Preditcted Class= -ve/Actual Class= -ve) = TN/(FP + TN)

Precision = P(Actual Class= +ve/Predictive Class= +ve) = TP/(TP + FP)

F1 Score = HM(Recall, Precision)

Accuracy = (TP + TN)/N....(N -> Total number of data points)

Confusion matrix

In [39]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(pred_df['Actual Class'], pred_df['Predicted Class'])

print('The Confusion Matrix:\n', cm)
```

```
The Confusion Matrix:
 [[ 11  56]
 [  3 130]]
```

11 -> True Positive (TP)

56 -> False Negative (FN)

3 -> False Positive (FP)

130 -> True Negative (TN)

This the 2nd row of confusion matrix correctly classifies

Classification Report

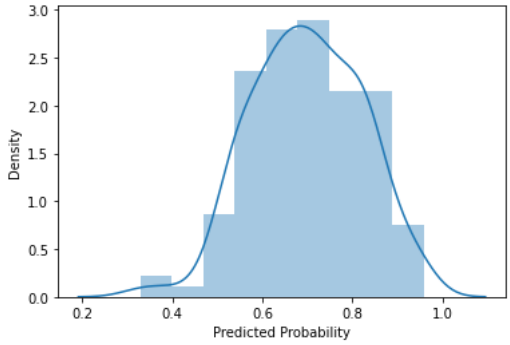
In [40]:
```
from sklearn.metrics import classification_report # performance measures not needed to be calculated manually by using the fomulc

report = classification_report(pred_df['Actual Class'], pred_df['Predicted Class'])

print('The classification report:\n', report)
```

```
The classification report:
              precision    recall  f1-score   support

           0       0.79      0.16      0.27        67
           1       0.70      0.98      0.82       133

    accuracy                           0.70       200
   macro avg       0.74      0.57      0.54       200
weighted avg       0.73      0.70      0.63       200
```

Plotting the distribution of predicted probabilities

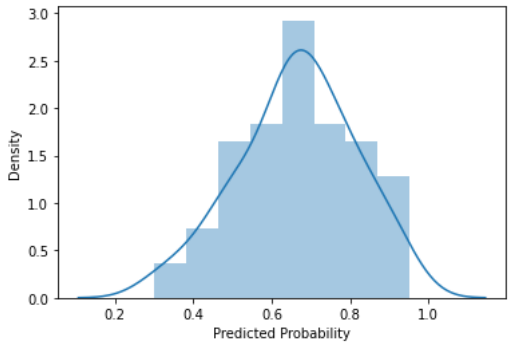
In [46]:
```
# Distribution of pred prob corresponding class = 1

sns.distplot(pred_df[pred_df['Actual Class']==1]['Predicted Probability']);
```

```
C:\Users\Urvi Sharma\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated funct
ion and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with simil
ar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [47]:
```
sns.distplot(pred_df[pred_df['Actual Class']==0]['Predicted Probability']);
```

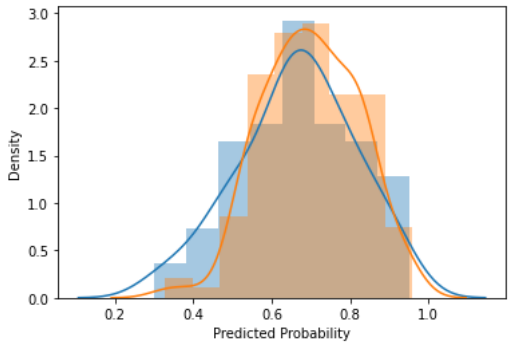
```
C:\Users\Urvi Sharma\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated funct
ion and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with simil
ar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [48]:
```
sns.distplot(pred_df[pred_df['Actual Class']==0]['Predicted Probability'])
sns.distplot(pred_df[pred_df['Actual Class']==1]['Predicted Probability'])
```

```
C:\Users\Urvi Sharma\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated funct
ion and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with simil
ar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Urvi Sharma\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated funct
ion and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with simil
ar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[48]: <AxesSubplot:xlabel='Predicted Probability', ylabel='Density'>



ROC Curve

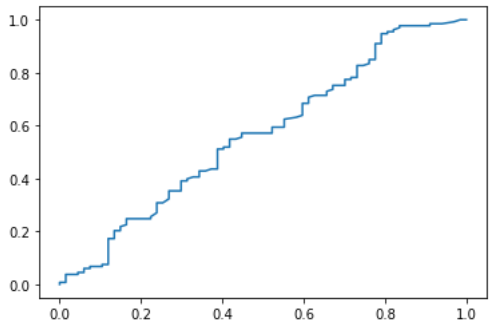
Receiver Operating Charateristics Curve

Plotted by taking:

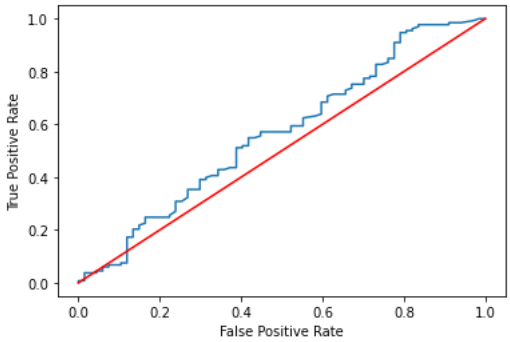
FPR(Sensitivity) ===> X-axis

TPR(1-Specificity) ===> Y-axis

In [49]:
```
from sklearn.metrics import roc_curve

fpr, tpr, threshold = roc_curve(pred_df['Actual Class'],
                                pred_df['Predicted Probability'])

plt.plot(fpr, tpr);
```



In [50]:
```
plt.plot(fpr,tpr)
plt.plot([0,1],[0,1],c='r')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show();
```



ROC-AUC Curve

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

score=roc_auc_score(pred_df['Actual Class'],
                    pred_df['Predicted Class'])

print('The ROC-AUC-Score:',score)
```

```
The ROC-AUC-Score: 0.5708113567500841
```

Finding the optimum classification cut-off

Many measures:

Youden's index : Max { Sensitivity + Specificity - 1}

: Max { Sensitivity - (1 - Specificity)}

: Max { FPR - TPR}, actual diff

In [53]:
```
# Creating a new DF

fpr_tpr = pd.DataFrame({'FPR': fpr,
                        'TPR': tpr,
                        'Threshold':threshold})
```

In [54]:
```
fpr_tpr
```

Out[54]:

	FPR	TPR	Threshold
0	0.000000	0.000000	1.958436
1	0.000000	0.007519	0.958436
2	0.014925	0.007519	0.952478
3	0.014925	0.037594	0.928304
4	0.044776	0.037594	0.920128
...	...	...	...
102	0.910448	0.984962	0.460934
103	0.940299	0.984962	0.421208
104	0.970149	0.992481	0.385243
105	0.985075	1.000000	0.327971
106	1.000000	1.000000	0.299315

107 rows × 3 columns

In [55]:
```
# creating the col of Diff

fpr_tpr['Diff']=fpr_tpr['FPR']-fpr_tpr['TPR']

fpr_tpr
```

Out[55]:

	FPR	TPR	Threshold	Diff
0	0.000000	0.000000	1.958436	0.000000
1	0.000000	0.007519	0.958436	-0.007519
2	0.014925	0.007519	0.952478	0.007407
3	0.014925	0.037594	0.928304	-0.022669
4	0.044776	0.037594	0.920128	0.007182
...	...	...	...	...
102	0.910448	0.984962	0.460934	-0.074515
103	0.940299	0.984962	0.421208	-0.044664
104	0.970149	0.992481	0.385243	-0.022332
105	0.985075	1.000000	0.327971	-0.014925
106	1.000000	1.000000	0.299315	0.000000

107 rows × 4 columns

In [56]:
```
#Sorting

fpr_tpr.sort_values('Diff',ascending=False)
```

Out[56]:

	FPR	TPR	Threshold	Diff
12	0.119403	0.075188	0.866104	0.044215
10	0.104478	0.067669	0.877585	0.036808
11	0.104478	0.075188	0.872735	0.029290
6	0.059701	0.045113	0.895420	0.014589
8	0.074627	0.060150	0.884680	0.014476
...	...	...	...	...
98	0.820896	0.962406	0.513231	-0.141510
100	0.835821	0.977444	0.504669	-0.141623
93	0.791045	0.939850	0.527421	-0.148805
96	0.805970	0.954887	0.521594	-0.148917
94	0.791045	0.947368	0.526841	-0.156324

107 rows × 4 columns

The threshold which provides the maximum DIFF is: 0.526841

In [59]:
```
# Use the new cut-off

pred_df['New Predicted Class'] = pred_df['Predicted Probability'].map(lambda x: 1 if x>0.526841 else 0)
```

In [60]:
```
pred_df
```

Out[60]:

	Actual Class	Predicted Probability	Predicted Class	New Predicted Class
841	0	0.690284	1	1
956	0	0.884680	1	1
544	1	0.850290	1	1
173	1	0.657965	1	1
759	0	0.512259	1	0
...	...	...	...	...
274	1	0.687278	1	1
192	1	0.385243	0	0
398	1	0.807991	1	1
450	0	0.927365	1	1
520	0	0.523344	1	0

200 rows × 4 columns

In [64]:
```
# New confusion matrix

new_cm = confusion_matrix(pred_df['Actual Class'],
                          pred_df['New Predicted Class'])
print("The new CM:\n", new_cm)
```

```
The new CM:
 [[ 14  53]
 [  8 125]]
```

In [66]:
```
new_score = roc_auc_score(pred_df['Actual Class'],
                  pred_df['New Predicted Class'])

print('The new score:', new_score)
```

```
The new score: 0.5744024239703737
```

The mode has slightly improved, not substantially

In [ ]: