In [1]:

```
import pyodbc
```

In [2]:

```
conn = pyodbc.connect(r'DSN=tekkredi;UID=yavuzs;PWD=18651438-155E-4450-859D-803181407D18')
```

In [3]:

```
import pandas as pd
import numpy as np
```

In [4]:

```
#database connection and import table
df = pd.read_sql("select * from dbo.IndusTrain_Model", conn)
df_test = pd.read_sql("select * from dbo.IndusTest_Model", conn)
```

In [5]:

```
#make binary target integer and check it
df['target_var']=df['target_var'].astype(int)
df_test['target_var']=df_test['target_var'].astype(int)
```

In [6]:

```
df_model=df[['VAR_ca_score', 'VAR_ctb_average_months_on_time_x_creditcard_loan_last_3months',

'VAR_Inverse_of_ratio_avg_months_30day_delinquent_last_6months_to_avg_months_30day_delinquent_last_
nths',
           'VAR_Inverse_of_ctb_avg_months_90day_delinquent_last_18months',
           'VAR_Log_of_ctb_avg_months_60day_delinquent_x_open_loan_x_personal_loan_last_6months',
           'VAR_Log_of_avg_ratio_totaldebt_to_creditlimit_last_3months',
           'VAR_Inverse_of_ctb_avg_months_90day_delinquent_x_overdraft_acct_last_6months','target_v
ar']]
```

In [7]:

```
df_test_model=df_test[['VAR_ca_score',
'VAR_ctb_average_months_on_time_x_creditcard_loan_last_3months',

'VAR_Inverse_of_ratio_avg_months_30day_delinquent_last_6months_to_avg_months_30day_delinquent_last_
nths',
               'VAR_Inverse_of_ctb_avg_months_90day_delinquent_last_18months',

'VAR_Log_of_ctb_avg_months_60day_delinquent_x_open_loan_x_personal_loan_last_6months',
               'VAR_Log_of_avg_ratio_totaldebt_to_creditlimit_last_3months',
               'VAR_Inverse_of_ctb_avg_months_90day_delinquent_x_overdraft_acct_last_6months','targ
et_var']]
```

In [8]:

```
df_vars=df_model.columns.values.tolist()
df_test_vars=df_test_model.columns.values.tolist()
```

In [9]:

```
y=df_model[['target_var']]
x=df_model[[i for i in df_vars if i not in y]]
y_test=df_test_model[['target_var']]
x_test=df_test_model[[i for i in df_test_vars if i not in y_test]]
```

In [13]:

```
#standardized xtrain
x_std = x.std(0)
x_mean = x.mean(0)
x_edit=(x-x_mean)/x_std
```

In [14]:

```
#standardized xtest
x_test_std = x_test.std(0)
x_test_mean = x_test.mean(0)
x_test_edit=(x_test-x_test_mean)/x_test_std
```

In [15]:

```
# Parameters initialization
weights = np.random.normal(0, 0.1, 7)
bias = np.random.normal(0, 0.1)
lr = 0.05
n = x_edit.shape[0]
```

In [16]:

```
for epoch in range(300):

    # Logistic function
    Z = np.dot(x_edit, weights) + bias
    A = 1 / (1 + np.exp(-Z))

    # Negative log likelihood -- loss function
    J = np.sum(-(y['target_var'] * np.log(A) + (1 - y['target_var'])* np.log(1 - A))) / n

    # Gradient computation
    dZ = A - y['target_var']
    dw = np.dot(dZ, x_edit) / n
    db = np.sum(dZ) / n

    # Update weights
    weights = weights - lr * dw
    bias = bias - lr * db

    if epoch % 10 == 0:
        print("epoch %s - loss %s" % (epoch, J))
```

```
epoch 0 - loss 0.677481894908509
epoch 10 - loss 0.61883121768083
epoch 20 - loss 0.5807421654855339
epoch 30 - loss 0.5545727294082987
epoch 40 - loss 0.5357011156944247
epoch 50 - loss 0.5215634418698314
epoch 60 - loss 0.5106565389076163
epoch 70 - loss 0.5020490933055944
epoch 80 - loss 0.49513482682896776
epoch 90 - loss 0.48950184376528716
epoch 100 - loss 0.4848599848289227
epoch 110 - loss 0.4809985245090218
epoch 120 - loss 0.47776047799866556
epoch 130 - loss 0.4750263895003382
epoch 140 - loss 0.4727037438290159
epoch 150 - loss 0.4707198315571611
epoch 160 - loss 0.46901680418688557
epoch 170 - loss 0.46754815977694103
epoch 180 - loss 0.4662761887967125
epoch 190 - loss 0.46517008102393415
epoch 200 - loss 0.4642044981681205
epoch 210 - loss 0.4633584815762525
epoch 220 - loss 0.4626146056169394
epoch 230 - loss 0.46195831423942074
epoch 240 - loss 0.4613773961366458
epoch 250 - loss 0.4608615661484515
epoch 260 - loss 0.4604021290164696
epoch 270 - loss 0.45999170759833685
epoch 280 - loss 0.4596240219649304
epoch 290 - loss 0.4592937089612868
```

In [17]:

```
#logreg prediction score for test dataset
pred_score = 1 / (1 + np.exp(-(np.dot(x_test_edit, weights) + bias)))
```

In [18]:

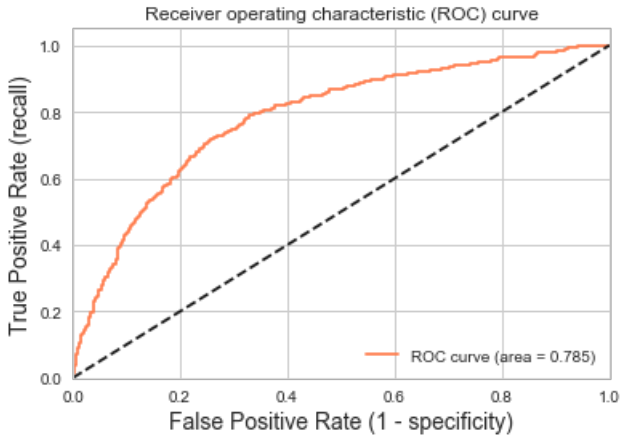
```
from sklearn.metrics import roc_curve, auc, log_loss, accuracy_score, confusion_matrix
[fpr, tpr, thr] = roc_curve(y_test, pred_score)
```

In [19]:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="white") #white background style for seaborn plots
sns.set(style="whitegrid", color_codes=True)
```

In [20]:

```
#draw the roc curve
plt.figure()
plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, tpr))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - specificity)', fontsize=14)
plt.ylabel('True Positive Rate (recall)', fontsize=14)
plt.title('Receiver operating characteristic (ROC) curve')
plt.legend(loc="lower right")
plt.show()
```



In [21]:

```
#calculate gini
2*0.785-1
```

Out[21]:

```
0.5700000000000001
```

In [25]:

```
list(x_edit)
```

Out[25]:

```
['VAR_ca_score',
 'VAR_ctb_average_months_on_time_x_creditcard_loan_last_3months',

'VAR_Inverse_of_ratio_avg_months_30day_delinquent_last_6months_to_avg_months_30day_delinquent_last_
nths',
 'VAR_Inverse_of_ctb_avg_months_90day_delinquent_last_18months',
 'VAR_Log_of_ctb_avg_months_60day_delinquent_x_open_loan_x_personal_loan_last_6months',
```

```
 'VAR_Log_of_avg_ratio_totaldebt_to_creditlimit_last_3months',
 'VAR_Inverse_of_ctb_avg_months_90day_delinquent_x_overdraft_acct_last_6months']
```

In [35]:

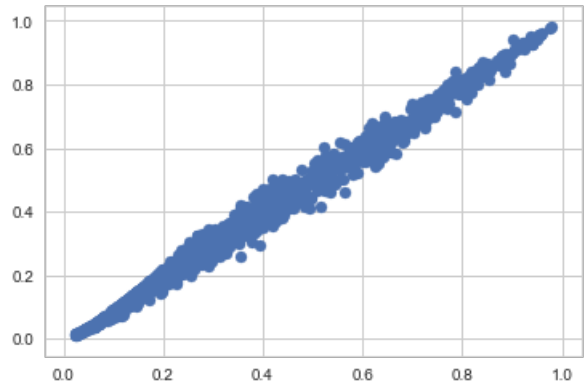
```
coef = np.array([-0.00108833861134955, -0.259787884876201, -0.0695878790121725,
                 -0.031520831720243, 0.345284032993245, 2.47679123771833, -0.0183380190178917])
intercept = 2.07915111384338
```

In [36]:

```
#Indus prediction score for test dataset
Indus_pred_score = 1 / (1 + np.exp(-(np.dot(x_test, coef) + intercept)))
```

In [61]:

```
plt.scatter(pred_score, Indus_pred_score )
plt.show()
```



In [42]:

```
#adding predicted probability results to test data
df_test_result=df_test_model
df_test_result = df_test_result.assign(pred_prob=pred_score)
df_test_result = df_test_result.assign(Indus_pred_prob=Indus_pred_score)
```

In [46]:

```
#creating deciles for ranking
df_test_result['decile_pred'] = pd.qcut(df_test_result['pred_prob'], 10, labels=np.arange(10, 0, -1
))
df_test_result['decile_Indus'] = pd.qcut(df_test_result['Indus_pred_prob'], 10, labels=np.arange(10
, 0, -1))
df_test_result['decile_Bureau'] = pd.qcut(df_test_result['VAR_ca_score'], 10, labels=False)+1

#making deciles integers
df_test_result['decile_pred']=df_test_result['decile_pred'].astype(int)
df_test_result['decile_Indus']=df_test_result['decile_Indus'].astype(int)
df_test_result['decile_Bureau']=df_test_result['decile_Bureau'].astype(int)
```

In [56]:

```
#creating table for deciles vs. bad rate
badrate_vs_decile_pred = pd.pivot_table(df_test_result, values='pred_prob', index=['decile_pred'],
                     columns=['target_var'], aggfunc='count')
badrate_vs_decile_Indus = pd.pivot_table(df_test_result, values='Indus_pred_prob', index=['decile_I
ndus'],
                     columns=['target_var'], aggfunc='count')
badrate_vs_decile_Bureau = pd.pivot_table(df_test_result, values='VAR_ca_score', index=['decile_Bur
eau'],
                     columns=['target_var'], aggfunc='count')

rank_pred = pd.DataFrame(badrate_vs_decile_pred.to_records())
rank_Indus = pd.DataFrame(badrate_vs_decile_Indus.to_records())
rank_Bureau = pd.DataFrame(badrate_vs_decile_Bureau.to_records())
```

```
rank_pred['decile'] = rank_pred['decile_pred']
rank_Indus['decile'] = rank_Indus['decile_Indus']
rank_Bureau['decile'] = rank_Bureau['decile_Bureau']

rank_pred['bad_rate']=rank_pred['1']/(rank_pred['0']+rank_pred['1'])
rank_Indus['bad_rate']=rank_Indus['1']/(rank_Indus['0']+rank_Indus['1'])
rank_Bureau['bad_rate']=rank_Bureau['1']/(rank_Bureau['0']+rank_Bureau['1'])

rank_merge=rank_pred.merge(rank_Indus, left_on='decile', right_on='decile', how='left')
rank_merge=rank_merge.merge(rank_Bureau, left_on='decile', right_on='decile', how='left')
```

In [57]:

```
rank_merge
```

Out[57]:

	decile_pred	0_x	1_x	decile	bad_rate_x	decile_Indus	0_y	1_y	bad_rate_y	decile_Bureau	0	1	bad_rate
0	1	60	139	1	0.698492	1	58	141	0.708543	1	90	110	0.550000
1	2	82	117	2	0.587940	2	88	111	0.557789	2	95	103	0.520202
2	3	110	89	3	0.447236	3	99	100	0.502513	3	93	107	0.535000
3	4	118	81	4	0.407035	4	127	72	0.361809	4	106	93	0.467337
4	5	149	50	5	0.251256	5	146	53	0.266332	5	154	44	0.222222
5	6	164	35	6	0.175879	6	170	29	0.145729	6	150	49	0.246231
6	7	174	25	7	0.125628	7	168	31	0.155779	7	175	27	0.133663
7	8	180	19	8	0.095477	8	181	18	0.090452	8	168	29	0.147208
8	9	178	21	9	0.105528	9	178	21	0.105528	9	185	14	0.070352
9	10	186	13	10	0.065327	10	186	13	0.065327	10	185	13	0.065657

In [62]:

```
plt.plot(rank_merge['decile'], rank_merge['bad_rate_x'], color='blue')
plt.plot(rank_merge['decile'], rank_merge['bad_rate_y'], color='brown')
plt.plot(rank_merge['decile'], rank_merge['bad_rate'], color='red')
plt.xlim((1, 10))
plt.ylim((0.0, 1.0))
plt.xticks(np.arange(1, 11, 1))
plt.show()
```