Loan application

- Loan application series 1, series 2

```
!kaggle dataset
!ls -lrt loan-application*
!unzip loan-application*

import matplotlib
%matplotlib inline
import pandas as pd, numpy as np, seaborn as sns, sklearn, os, sys

from IPKM import *
from PyCommon/Utils import *
from PyCommon/Plots import *
```

```
!kaggle dataset -d ninzaami/loan-predication
!ls -lrt loan-predication*
!unzip loan-predication*
```

```
df = pd.read_csv('loan-train.csv')
df.shape, df.head(20)
```

```
(614, 13)
   Loan_ID Gender Married Dependents     Education Self_Employed  ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History Property_Area Loan_Status
0 LP001002   Male     No          0      Graduate            No             5849                0         NaN             360             1.0         Urban           Y
1 LP001003   Male    Yes          1      Graduate            No             4583             1508.0       128.0             360             1.0         Rural           N
2 LP001005   Male    Yes          0      Graduate           Yes             3000                0        66.0             360             1.0         Urban           Y
3 LP001006   Male    Yes          0 Not Graduate            No             2583             2358.0       120.0             360             1.0         Urban           Y
4 LP001008   Male     No          0      Graduate            No             6000                0       141.0             360             1.0         Urban           Y
5 LP001011   Male    Yes         2+      Graduate           Yes             5417             4196.0       267.0             360             1.0         Urban           Y
6 LP001013   Male    Yes          0 Not Graduate            No             2333             1516.0        95.0             360             1.0         Urban           Y
7 LP001014   Male    Yes         3+      Graduate            No             3036             2504.0       158.0             360             0.0         Semi-Urban      N
8 LP001018   Male    Yes          2      Graduate            No             4006             1526.0       168.0             360             1.0         Urban           Y
9 LP001020   Male    Yes         3+      Graduate            No            12841            10968.0       349.0             360             1.0         Semi-Urban      N
10 LP001024   Male    Yes          2      Graduate            No             3200             700.0        70.0             360             1.0         Urban           Y
11 LP001026   Male    Yes         3+      Graduate            No             2500             1840.0       109.0             360             1.0         Urban           Y
12 LP001028   Male    Yes         3+      Graduate            No             3073             8106.0       200.0             360             1.0         Urban           Y
13 LP001029   Male     No          1      Graduate            No             1853             2840.0       114.0             360             1.0         Rural           N
14 LP001030   Male    Yes         2+      Graduate            No             1299             1086.0        17.0             120             1.0         Urban           Y
15 LP001031   Male    Yes         2+      Graduate           Yes             5955                0        126.0             360             1.0         Urban           Y
16 LP001032   Male     No          0      Graduate            No             4283             3000.0       250.0             360             0.0         Semi-Urban      N
17 LP001035   Male    Yes         2+      Graduate            No             4051             2400.0       167.0             360             1.0         Urban           Y
18 LP001036 Female     No          0      Graduate            No             3000                0        90.0             360             0.0         Urban           N
19 LP001037   Male     No          2      Graduate            No             4887                0        207.0             360             1.0         Rural           Y
```

```
df.describe(), df.info()
```

```
<class'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Loan_ID            614 non-null    object
 #   Gender             601 non-null    object
 #   Married            611 non-null    object
 #   Dependents         599 non-null    object
 #   Education          614 non-null    object
 #   Self_Employed      582 non-null    object
 #   ApplicantIncome    614 non-null    int64
 #   CoapplicantIncome  614 non-null    int64
 #   LoanAmount         592 non-null    float64
 #   Loan_Amount_Term   600 non-null    float64
 #   Credit_History     564 non-null    float64
 #   Property_Area      614 non-null    object
 #   Loan_Status        614 non-null    object
dtype: int64, object: 8
memory usage: 62.4+ KB
```

```
df.isna.sum()
```

```
Loan_ID            0
Gender            13
Married            3
Dependents        15
Education          0
Self_Employed     32
ApplicantIncome    0
CoapplicantIncome  0
LoanAmount        22
Loan_Amount_Term  14
Credit_History    50
Property_Area      0
Loan_Status        0
dtype: int64
```

```
df.isna().sum()
```

Data clean-up & treating missing values

Fixing Gender

```
df.loc[df.Gender.isna(),'Gender'] = 'Male'
```

```
df.Gender.value_counts()
```

```
df.loc[df.Gender.isna(),'Gender'] = df.Gender.mode[0]
```

```
df.Gender.value_counts()
```

```
df.Gender.isna().sum()
```

```
0
```

```
df.Married.value_counts()
```

```
Married
Yes    398
No     213
Name: count, dtype: int64
```

```
df.loc[df.Married.isna(),'Married'] = df.Married.mode[0]
```

```
df.Married.isna().sum()
```

```
0
```

Fixing Loan amount

```
df.loc[df.LoanAmount.isna(),'LoanAmount'] = df.LoanAmount.median()
df.loc[df.Loan_Amount_Term.isna(),'Loan_Amount_Term'] = df.Loan_Amount_Term.median()
df.loc[df.Credit_History.isna(),'Credit_History'] = df.Credit_History.median()
```

```
df.isna().sum()
```

```
Loan_ID            0
Gender            0
Married           0
Dependents       15
Education         0
Self_Employed    32
ApplicantIncome   0
CoapplicantIncome 0
LoanAmount        0
Loan_Amount_Term  0
Credit_History    0
Property_Area     0
Loan_Status       0
dtype: int64
```

```
df.Dependents.value_counts()
```

```
Dependents
0    345
1    102
2    101
3+    51
Name: count, dtype: int64
```

```
df.loc[df.Dependents.isna(),'Dependents'] = df.Dependents.mode[0]
df.loc[df.Self_Employed.isna(),'Self_Employed'] = df.Self_Employed.mode[0]
```

```
df.isna().sum()
```

```
Loan_ID            0
Gender            0
Married           0
Dependents        0
Education         0
Self_Employed     0
ApplicantIncome   0
CoapplicantIncome 0
LoanAmount        0
Loan_Amount_Term  0
Credit_History    0
Property_Area     0
Loan_Status       0
dtype: int64
```

Exploratory Data Analysis

```
df.loc[df.ApplicantIncome>20000,'ApplicantIncome'] = df.loc[df.ApplicantIncome>20000,'ApplicantIncome'].median()
df.loc[df.CoapplicantIncome>10000,'CoapplicantIncome'] = df.loc[df.CoapplicantIncome>10000,'CoapplicantIncome'].median()
```

```
df.describe()
```

```
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History
count       614.000000         614.000000  614.000000        614.000000        614.000
mean       5403.459283        1621.245798  146.412162        342.410423          0.842
std        6109.041673        2926.248369   84.037468         65.412416          0.364
min           150.000           0.000000    9.000000         12.000000          0.000
25%          2877.500           0.000000  100.250000        360.000000          1.000
50%          3812.500        1188.500000  128.000000        360.000000          1.000
75%          5795.000        2297.250000  164.750000        360.000000          1.000
max         20000.000        5701.000000  700.000000        480.000000          1.000
```

```
df.Loan_Status.value_counts
```

```
Loan_Status
Y    422
N    192
Name: count, dtype: int64
```

Exploring & treating features

```
sns.boxplot(df.ApplicantIncome,df.Loan_Status,orient='h')
```

```
sns.boxplot(df.CoapplicantIncome,df.Loan_Status,orient='h')
```

```
sns.boxplot(df.LoanAmount,df.Loan_Status,orient='h')
```

```
sns.boxplot(df.Loan_Amount_Term,df.Loan_Status,orient='h')
```

```
sns.boxplot(df.Credit_History,df.Loan_Status,orient='h')
```

```
sns.boxplot(df.Dependents,df.Loan_Status,orient='h')
```

```
sns.boxplot(df.Gender,df.Loan_Status,orient='h')
```

```
sns.boxplot(df.Married,df.Loan_Status,orient='h')
```

```
sns.boxplot(df.Self_Employed,df.Loan_Status,orient='h')
```


Fixing Loan amount

```
df.loc[df.LoanAmount>500,'LoanAmount'] = df.LoanAmount.median()
df.loc[df.Loan_Amount_Term>400,'Loan_Amount_Term'] = df.Loan_Amount_Term.median()
df.loc[df.Loan_Amount_Term<100,'Loan_Amount_Term'] = df.Loan_Amount_Term.median()
df.loc[df.ApplicantIncome>20000,'ApplicantIncome'] = df.loc[df.ApplicantIncome>20000,'ApplicantIncome'].median()
df.loc[df.CoapplicantIncome>10000,'CoapplicantIncome'] = df.loc[df.CoapplicantIncome>10000,'CoapplicantIncome'].median()
```

```
df.describe()
```

```
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History
count       614.000000         614.000000  614.000000        614.000000        614.000
mean       5403.459283        1621.245798  146.412162        342.410423          0.842
std        6109.041673        2926.248369   84.037468         65.412416          0.364
min           150.000           0.000000    9.000000         12.000000          0.000
25%          2877.500           0.000000  100.250000        360.000000          1.000
50%          3812.500        1188.500000  128.000000        360.000000          1.000
75%          5795.000        2297.250000  164.750000        360.000000          1.000
max         20000.000        5701.000000  700.000000        480.000000          1.000
```

```
df.Loan_Status.value_counts
```

```
Loan_Status
Y    422
N    192
Name: count, dtype: int64
```

```
df.loc[df.Loan_Status=='Y','Loan_Status'] = 1
df.loc[df.Loan_Status=='N','Loan_Status'] = 0
```

```
df.Loan_Status.value_counts
```

```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

```
df.groupby('Property_Area').mean()
```

```
               ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  Loan_Status
Property_Area
Rural                 5433.718           1821.000  146.048077        342.000000        0.827778     0.614035
Semi-Urban            5012.223           1744.192  144.168831        342.000000        0.868571     0.768911
Urban                 5445.087           1443.757  145.020408        342.000000        0.836735     0.658163
```

```
df.groupby('Property_Area','Loan_Status').mean()
```

```
                                 ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History
Property_Area Loan_Status
Rural         0                        5384.068           1504.875  143.732870        342.000000        0.730556
              1                        5468.488           1877.807  147.828035        342.000000        0.887324
Semi-Urban    0                        5020.359           1821.718  145.415095        342.000000        0.579365
              1                        5011.094           1704.700  143.732549        342.000000        0.901639
Urban         0                        5446.078           1795.848  141.986111        342.000000        0.694444
              1                        5445.087           1443.757  145.020408        342.000000        0.836735
```

Fixing Property Area

```
df.loc[df.Property_Area=='Rural','Property_Area'] = 1
df.loc[df.Property_Area=='Semi-Urban','Property_Area'] = 2
df.loc[df.Property_Area=='Urban','Property_Area'] = 3
```

```
df.Property_Area.value_counts
```

```
Property_Area
2    233
3    202
1    179
Name: count, dtype: int64
```

```
df.groupby('Property_Area','Loan_Status').mean()
```

```
                                 ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History
Property_Area Loan_Status
1             0                        5384.068           1504.875  143.732870        342.000000        0.730556
              1                        5468.488           1877.807  147.828035        342.000000        0.887324
2             0                        5020.359           1821.718  145.415095        342.000000        0.579365
              1                        5011.094           1704.700  143.732549        342.000000        0.901639
3             0                        5446.078           1795.848  141.986111        342.000000        0.694444
              1                        5445.087           1443.757  145.020408        342.000000        0.836735
```

Fixing Dependents

```
df.loc[df.Dependents=='3+','Dependents'] = 3
df.Dependents.value_counts
```

```
Dependents
0    345
1    102
2    101
3     51
Name: count, dtype: int64
```

Fixing Education

```
df.loc[df.Education=='Graduate','Education'] = 1
df.loc[df.Education=='Not Graduate','Education'] = 2
```

```
df.Education.value_counts
```

```
Education
1    480
2    134
Name: count, dtype: int64
```

Tidy up & create dataset for Machine Learning

```
df.loc[df.Gender=='Male','Gender'] = 1
df.loc[df.Gender=='Female','Gender'] = 2
df.loc[df.Married=='Yes','Married'] = 1
df.loc[df.Married=='No','Married'] = 2
df.loc[df.Self_Employed=='Yes','Self_Employed'] = 1
df.loc[df.Self_Employed=='No','Self_Employed'] = 2
```

```
df[df.Loan_ID==1]
```

```
   Loan_ID Gender Married Dependents Education Self_Employed  ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History Property_Area Loan_Status
0 LP001002      1       2          0         1            2             5849                0         NaN             360             1.0         Urban           1
```

```
df.drop('Loan_ID',axis=1,inplace=True)
df.shape, df.head(1)
```

```
(614, 12)
   Gender Married Dependents Education Self_Employed  ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History Property_Area Loan_Status
0       1       2          0         1            2             5849                0         NaN             360             1.0         Urban           1
```

```
df.Dependents.value_counts
```

```
Dependents
0    345
1    102
2    101
3     51
Name: count, dtype: int64
```

Fitting Model

```
Accuracy = 0.8699187672790105 * 100%
[array([0.86666667, 0.86666667, 0.86666667, 0.86666667, 0.86666667])]
Model = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
Cross Validation Mean Score: 0.8666666666666667
Model score: 0.8699187672790105
Classification_report
              precision    recall  f1-score   support

           0       0.92       0.60       0.73       154
           1       0.86       0.98       0.91       422

    accuracy       0.87       0.87       0.87       576
   macro avg       0.89       0.79       0.82       576
weighted avg       0.87       0.87       0.86       576
confusion_matrix
[[ 92  62]
[  9 413]]
```

```
df[df.Loan_ID==1]
```

```
   Gender Married Dependents Education Self_Employed  ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History Property_Area Loan_Status
0       1       2          0         1            2             5849                0         NaN             360             1.0         Urban           1
```

```
df.Dependents.value_counts
```

```
Dependents
0    345
1    102
2    101
3     51
Name: count, dtype: int64
```

Exploring features importances

```
Feature importances
                   Feature  Importance
0           ApplicantIncome    0.166667
1         CoapplicantIncome    0.166667
2                Dependents    0.166667
3             Property_Area    0.166667
4                Education    0.166667
5                   Married    0.166667
6                    Gender    0.166667
7             Self_Employed    0.166667
8                Credit_History    0.166667
9                LoanAmount    0.166667
10         Loan_Amount_Term    0.166667
```

```
In [84]: df[cols_0].
```

	Code_Status	Pediatrician	Leadership	Low_Status
0	1.0	0000.0	100.0	0
1	1.0	0001.0	100.0	0
2	1.0	0002.0	100.0	0
3	1.0	0003.0	100.0	0
4	1.0	0004.0	100.0	0
5	1.0	0005.0	100.0	0
6	1.0	0006.0	100.0	0
7	1.0	0007.0	100.0	0
8	1.0	0008.0	100.0	0
9	1.0	0009.0	100.0	0
10	1.0	0010.0	100.0	0
11	1.0	0011.0	100.0	0
12	1.0	0012.0	100.0	0
13	1.0	0013.0	100.0	0
14	1.0	0014.0	100.0	0
15	1.0	0015.0	100.0	0
16	1.0	0016.0	100.0	0
17	1.0	0017.0	100.0	0
18	1.0	0018.0	100.0	0
19	1.0	0019.0	100.0	0
20	1.0	0020.0	100.0	0
21	1.0	0021.0	100.0	0
22	1.0	0022.0	100.0	0
23	1.0	0023.0	100.0	0
24	1.0	0024.0	100.0	0
25	1.0	0025.0	100.0	0
26	1.0	0026.0	100.0	0
27	1.0	0027.0	100.0	0
28	1.0	0028.0	100.0	0
29	1.0	0029.0	100.0	0
30	1.0	0030.0	100.0	0
31	1.0	0031.0	100.0	0
32	1.0	0032.0	100.0	0
33	1.0	0033.0	100.0	0
34	1.0	0034.0	100.0	0
35	1.0	0035.0	100.0	0
36	1.0	0036.0	100.0	0
37	1.0	0037.0	100.0	0
38	1.0	0038.0	100.0	0
39	1.0	0039.0	100.0	0
40	1.0	0040.0	100.0	0
41	1.0	0041.0	100.0	0
42	1.0	0042.0	100.0	0
43	1.0	0043.0	100.0	0
44	1.0	0044.0	100.0	0
45	1.0	0045.0	100.0	0
46	1.0	0046.0	100.0	0
47	1.0	0047.0	100.0	0
48	1.0	0048.0	100.0	0
49	1.0	0049.0	100.0	0
50	1.0	0050.0	100.0	0
51	1.0	0051.0	100.0	0
52	1.0	0052.0	100.0	0
53	1.0	0053.0	100.0	0
54	1.0	0054.0	100.0	0
55	1.0	0055.0	100.0	0
56	1.0	0056.0	100.0	0
57	1.0	0057.0	100.0	0
58	1.0	0058.0	100.0	0
59	1.0	0059.0	100.0	0
60	1.0	0060.0	100.0	0
61	1.0	0061.0	100.0	0
62	1.0	0062.0	100.0	0
63	1.0	0063.0	100.0	0
64	1.0	0064.0	100.0	0
65	1.0	0065.0	100.0	0
66	1.0	0066.0	100.0	0
67	1.0	0067.0	100.0	0
68	1.0	0068.0	100.0	0
69	1.0	0069.0	100.0	0
70	1.0	0070.0	100.0	0
71	1.0	0071.0	100.0	0
72	1.0	0072.0	100.0	0
73	1.0	0073.0	100.0	0
74	1.0	0074.0	100.0	0
75	1.0	0075.0	100.0	0
76	1.0	0076.0	100.0	0
77	1.0	0077.0	100.0	0
78	1.0	0078.0	100.0	0
79	1.0	0079.0	100.0	0
80	1.0	0080.0	100.0	0
81	1.0	0081.0	100.0	0
82	1.0	0082.0	100.0	0
83	1.0	0083.0	100.0	0
84	1.0	0084.0	100.0	0
85	1.0	0085.0	100.0	0
86	1.0	0086.0	100.0	0
87	1.0	0087.0	100.0	0
88	1.0	0088.0	100.0	0
89	1.0	0089.0	100.0	0
90	1.0	0090.0	100.0	0
91	1.0	0091.0	100.0	0
92	1.0	0092.0	100.0	0
93	1.0	0093.0	100.0	0
94	1.0	0094.0	100.0	0
95	1.0	0095.0	100.0	0
96	1.0	0096.0	100.0	0
97	1.0	0097.0	100.0	0
98	1.0	0098.0	100.0	0
99	1.0	0099.0	100.0	0
100	1.0	0100.0	100.0	0
101	1.0	0101.0	100.0	0
102	1.0	0102.0	100.0	0
103	1.0	0103.0	100.0	0
104	1.0	0104.0	100.0	0
105	1.0	0105.0	100.0	0
106	1.0	0106.0	100.0	0
107	1.0	0107.0	100.0	0
108	1.0	0108.0	100.0	0
109	1.0	0109.0	100.0	0
110	1.0	0110.0	100.0	0
111	1.0	0111.0	100.0	0
112	1.0	0112.0	100.0	0
113	1.0	0113.0	100.0	0
114	1.0	0114.0	100.0	0
115	1.0	0115.0	100.0	0
116	1.0	0116.0	100.0	0
117	1.0	0117.0	100.0	0
118	1.0	0118.0	100.0	0
119	1.0	0119.0	100.0	0
120	1.0	0120.0	100.0	0
121	1.0	0121.0	100.0	0
122	1.0	0122.0	100.0	0
123	1.0	0123.0	100.0	0
124	1.0	0124.0	100.0	0
125	1.0	0125.0	100.0	0
126	1.0	0126.0	100.0	0
127	1.0	0127.0	100.0	0
128	1.0	0128.0	100.0	0
129	1.0	0129.0	100.0	0
130	1.0	0130.0	100.0	0
131	1.0	0131.0	100.0	0
132	1.0	0132.0	100.0	0
133	1.0	0133.0	100.0	0
134	1.0	0134.0	100.0	0
135	1.0	0135.0	100.0	0
136	1.0	0136.0	100.0	0
137	1.0	0137.0	100.0	0
138	1.0	0138.0	100.0	0
139	1.0	0139.0	100.0	0
140	1.0	0140.0	100.0	0
141	1.0	0141.0	100.0	0
142	1.0	0142.0	100.0	0
143	1.0	0143.0	100.0	0
144	1.0	0144.0	100.0	0
145	1.0	0145.0	100.0	0
146	1.0	0146.0	100.0	0
147	1.0	0147.0	100.0	0
148	1.0	0148.0	100.0	0
149	1.0	0149.0	100.0	0
150	1.0	0150.0	100.0	0
151	1.0	0151.0	100.0	0
152	1.0	0152.0	100.0	0
153	1.0	0153.0	100.0	0
154	1.0	0154.0	100.0	0
155	1.0	0155.0	100.0	0
156	1.0	0156.0	100.0	0
157	1.0	0157.0	100.0	0
158	1.0	0158.0	100.0	0
159	1.0	0159.0	100.0	0
160	1.0	0160.0	100.0	0
161	1.0	0161.0	100.0	0
162	1.0	0162.0	100.0	0
163	1.0	0163.0	100.0	0
164	1.0	0164.0	100.0	0
165	1.0	0165.0	100.0	0
166	1.0	0166.0	100.0	0
167	1.0	0167.0	100.0	0
168	1.0	0168.0	100.0	0
169	1.0	0169.0	100.0	0
170	1.0	0170.0	100.0	0
171	1.0	0171.0	100.0	0
172	1.0	0172.0	100.0	0
173	1.0	0173.0	100.0	0
174	1.0	0174.0	100.0	0
175	1.0	0175.0	100.0	0
176	1.0	0176.0	100.0	0
177	1.0	0177.0	100.0	0
178	1.0	0178.0	100.0	0
179	1.0	0179.0	100.0	0
180	1.0	0180.0	100.0	0
181	1.0	0181.0	100.0	0
182	1.0	0182.0	100.0	0
183	1.0	0183.0	100.0	0
184	1.0	0184.0	100.0	0
185	1.0	0185.0	100.0	0
186	1.0	0186.0	100.0	0
187	1.0	0187.0	100.0	0
188	1.0	0188.0	100.0	0
189	1.0	0189.0	100.0	0
190	1.0	0190.0	100.0	0
191	1.0	0191.0	100.0	0
192	1.0	0192.0	100.0	0
193	1.0	0193.0	100.0	0
194	1.0	0194.0	100.0	0
195	1.0	0195.0	100.0	0
196	1.0	0196.0	100.0	0
197	1.0	0197.0	100.0	0
198	1.0	0198.0	100.0	0
199	1.0	0199.0	100.0	0
200	1.0	0200.0	100.0	0
201	1.0	0201.0	100.0	0
202	1.0	0202.0	100.0	0
203	1.0	0203.0	100.0	0
204	1.0	0204.0	100.0	0
205	1.0	0205.0	100.0	0

Split into Train & Test sets

```
In [85]: X_train, X_test, y_train, y_test = train_test_split(
         df[cols_0].drop(['Low_Status'], axis=1),
         df[cols_0]['Low_Status'],
         random_state=0)

         print("X_train shape:{}".
         format(X_train.shape))
         print("y_train shape:{}".
         format(y_train.shape))
```

```
X_train shape:(1,)
y_train shape:(1,)
```

Fit into Machine Learning

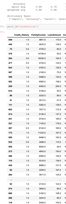
```
In [86]: clf = LogisticRegression(C=100, random_state=0).fit(X_train, y_train)
         plot_class_regions_for_classifier(clf, X_train, y_train,
                                           X_test, y_test,
                                           'Logistic Regression, Fixed Acidity, '+
                                           'Citric Acid, Chlorides, Sulphates, '+
                                           'Total Sulphur Dioxide, Density, '+
                                           'Pediatrician, Leadership, Low_Status');
```

```
In [87]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import roc_curve, auc
         from sklearn.model_selection import train_test_split
         from sklearn.datasets import load_breast_cancer
         from sklearn.preprocessing import StandardScaler

         X_train, X_test, y_train, y_test = train_test_split(df[cols_0].drop(['Low_Status'], axis=1),
                                             df[cols_0]['Low_Status'],
                                             random_state=0)
         X_train = X_train[:-1]
         y_train = y_train[:-1]
         X_test = X_test[:-1]
         y_test = y_test[:-1]
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)
         clf = LogisticRegression(C=100).fit(X_train_scaled, y_train)
         plot_class_regions_for_classifier_2d(clf, X_train_scaled, y_train,
                                              X_test_scaled, y_test,
                                              'Decision Boundary using Pediatrician/Boundaries,'+
                                              'Leadership,Low_Status,Pediatrician,Leadership,'+
                                              'Low_Status,Pediatrician,Leadership,Low_Status,'+
                                              'Pediatrician,Leadership,Low_Status,Pediatrician,'+
                                              'Leadership,Low_Status,Pediatrician,Leadership,'+
                                              'Low_Status,Pediatrician,Leadership,Low_Status,');
```

```
Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,
```

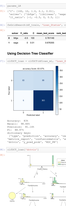
```
In [88]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import roc_curve, auc
         from sklearn.model_selection import train_test_split
         from sklearn.datasets import load_breast_cancer
         from sklearn.preprocessing import StandardScaler

         X_train, X_test, y_train, y_test = train_test_split(df[cols_0].drop(['Low_Status'], axis=1),
                                             df[cols_0]['Low_Status'],
                                             random_state=0)
         X_train = X_train[:-1]
         y_train = y_train[:-1]
         X_test = X_test[:-1]
         y_test = y_test[:-1]
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)
         clf = LogisticRegression(C=100).fit(X_train_scaled, y_train)
         plot_class_regions_for_classifier_2d(clf, X_train_scaled, y_train,
                                              X_test_scaled, y_test,
                                              'Decision Boundary,'+
                                              'Pediatrician,Leadership,'+
                                              'Low_Status,Pediatrician,Leadership,'+
                                              'Low_Status,Pediatrician,Leadership,'+
                                              'Low_Status,Pediatrician,Leadership,'+
                                              'Low_Status,Pediatrician,Leadership,'+
                                              'Low_Status,Pediatrician,Leadership,');
```

```
Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,Pediatrician,Leadership,Low_Status,
```



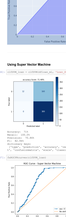
Adjust the ROC threshold

```
In [89]: plot_roc_curve(clf, X_test_scaled, y_test,
                        name='ROC Curve, Logistic Regression',
                        ax=ax);
```




```
In [90]: fig,(ax1,ax2,ax3)=plt.subplots(1,3,figsize=(15,5))
         plot_roc_curve(clf, X_train_scaled, y_train,
                        name='ROC Curve, Training Data',
                        ax=ax1);
         plot_roc_curve(clf, X_test_scaled, y_test,
                        name='ROC Curve, Test Data',
                        ax=ax2);
         plot_roc_curve(clf, X_test_scaled, y_test,
                        name='ROC Curve, Logistic Regression',
                        ax=ax3);
```

```
In [91]: plot_class_regions_for_classifier(clf, X_train_scaled, y_train,
                                           X_test_scaled, y_test,
                                           'Logistic Regression, Fixed Acidity, '+
                                           'Citric Acid, Chlorides, Sulphates, '+
                                           'Total Sulphur Dioxide, Density, '+
                                           'Pediatrician, Leadership, Low_Status');
```



Confusion for validation

```
In [92]: pred_LR = clf.predict(X_test_scaled)
         confusion = confusion_matrix(y_test, pred_LR)
         print("Confusion matrix:\n{}".
               format(confusion))
         print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
               .format(metrics.accuracy_score(y_test, pred_LR),
                       metrics.precision_score(y_test, pred_LR),
                       metrics.recall_score(y_test, pred_LR)))
         print("\n")
         print(metrics.classification_report(y_test, pred_LR,
                                             target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [93]: pred_LR = clf.predict(X_test_scaled)
         confusion = confusion_matrix(y_test, pred_LR)
         print("Confusion matrix:\n{}".
               format(confusion))
         print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
               .format(metrics.accuracy_score(y_test, pred_LR),
                       metrics.precision_score(y_test, pred_LR),
                       metrics.recall_score(y_test, pred_LR)))
         print("\n")
         print(metrics.classification_report(y_test, pred_LR,
                                             target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [94]: pred_LR = clf.predict(X_test_scaled)
         confusion = confusion_matrix(y_test, pred_LR)
         print("Confusion matrix:\n{}".
               format(confusion))
         print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
               .format(metrics.accuracy_score(y_test, pred_LR),
                       metrics.precision_score(y_test, pred_LR),
                       metrics.recall_score(y_test, pred_LR)))
         print("\n")
         print(metrics.classification_report(y_test, pred_LR,
                                             target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [95]: pred_LR = clf.predict(X_test_scaled)
         confusion = confusion_matrix(y_test, pred_LR)
         print("Confusion matrix:\n{}".
               format(confusion))
         print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
               .format(metrics.accuracy_score(y_test, pred_LR),
                       metrics.precision_score(y_test, pred_LR),
                       metrics.recall_score(y_test, pred_LR)))
         print("\n")
         print(metrics.classification_report(y_test, pred_LR,
                                             target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [96]: pred_LR = clf.predict(X_test_scaled)
         confusion = confusion_matrix(y_test, pred_LR)
         print("Confusion matrix:\n{}".
               format(confusion))
         print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
               .format(metrics.accuracy_score(y_test, pred_LR),
                       metrics.precision_score(y_test, pred_LR),
                       metrics.recall_score(y_test, pred_LR)))
         print("\n")
         print(metrics.classification_report(y_test, pred_LR,
                                             target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [97]: pred_LR = clf.predict(X_test_scaled)
         confusion = confusion_matrix(y_test, pred_LR)
         print("Confusion matrix:\n{}".
               format(confusion))
         print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
               .format(metrics.accuracy_score(y_test, pred_LR),
                       metrics.precision_score(y_test, pred_LR),
                       metrics.recall_score(y_test, pred_LR)))
         print("\n")
         print(metrics.classification_report(y_test, pred_LR,
                                             target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [98]: pred_LR = clf.predict(X_test_scaled)
         confusion = confusion_matrix(y_test, pred_LR)
         print("Confusion matrix:\n{}".
               format(confusion))
         print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
               .format(metrics.accuracy_score(y_test, pred_LR),
                       metrics.precision_score(y_test, pred_LR),
                       metrics.recall_score(y_test, pred_LR)))
         print("\n")
         print(metrics.classification_report(y_test, pred_LR,
                                             target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [99]: pred_LR = clf.predict(X_test_scaled)
         confusion = confusion_matrix(y_test, pred_LR)
         print("Confusion matrix:\n{}".
               format(confusion))
         print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
               .format(metrics.accuracy_score(y_test, pred_LR),
                       metrics.precision_score(y_test, pred_LR),
                       metrics.recall_score(y_test, pred_LR)))
         print("\n")
         print(metrics.classification_report(y_test, pred_LR,
                                             target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [100]: pred_LR = clf.predict(X_test_scaled)
          confusion = confusion_matrix(y_test, pred_LR)
          print("Confusion matrix:\n{}".
                format(confusion))
          print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
                .format(metrics.accuracy_score(y_test, pred_LR),
                        metrics.precision_score(y_test, pred_LR),
                        metrics.recall_score(y_test, pred_LR)))
          print("\n")
          print(metrics.classification_report(y_test, pred_LR,
                                              target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [101]: pred_LR = clf.predict(X_test_scaled)
          confusion = confusion_matrix(y_test, pred_LR)
          print("Confusion matrix:\n{}".
                format(confusion))
          print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
                .format(metrics.accuracy_score(y_test, pred_LR),
                        metrics.precision_score(y_test, pred_LR),
                        metrics.recall_score(y_test, pred_LR)))
          print("\n")
          print(metrics.classification_report(y_test, pred_LR,
                                              target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [102]: pred_LR = clf.predict(X_test_scaled)
          confusion = confusion_matrix(y_test, pred_LR)
          print("Confusion matrix:\n{}".
                format(confusion))
          print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
                .format(metrics.accuracy_score(y_test, pred_LR),
                        metrics.precision_score(y_test, pred_LR),
                        metrics.recall_score(y_test, pred_LR)))
          print("\n")
          print(metrics.classification_report(y_test, pred_LR,
                                              target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [103]: pred_LR = clf.predict(X_test_scaled)
          confusion = confusion_matrix(y_test, pred_LR)
          print("Confusion matrix:\n{}".
                format(confusion))
          print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
                .format(metrics.accuracy_score(y_test, pred_LR),
                        metrics.precision_score(y_test, pred_LR),
                        metrics.recall_score(y_test, pred_LR)))
          print("\n")
          print(metrics.classification_report(y_test, pred_LR,
                                              target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [104]: pred_LR = clf.predict(X_test_scaled)
          confusion = confusion_matrix(y_test, pred_LR)
          print("Confusion matrix:\n{}".
                format(confusion))
          print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
                .format(metrics.accuracy_score(y_test, pred_LR),
                        metrics.precision_score(y_test, pred_LR),
                        metrics.recall_score(y_test, pred_LR)))
          print("\n")
          print(metrics.classification_report(y_test, pred_LR,
                                              target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [105]: pred_LR = clf.predict(X_test_scaled)
          confusion = confusion_matrix(y_test, pred_LR)
          print("Confusion matrix:\n{}".
                format(confusion))
          print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
                .format(metrics.accuracy_score(y_test, pred_LR),
                        metrics.precision_score(y_test, pred_LR),
                        metrics.recall_score(y_test, pred_LR)))
          print("\n")
          print(metrics.classification_report(y_test, pred_LR,
                                              target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [106]: pred_LR = clf.predict(X_test_scaled)
          confusion = confusion_matrix(y_test, pred_LR)
          print("Confusion matrix:\n{}".
                format(confusion))
          print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
                .format(metrics.accuracy_score(y_test, pred_LR),
                        metrics.precision_score(y_test, pred_LR),
                        metrics.recall_score(y_test, pred_LR)))
          print("\n")
          print(metrics.classification_report(y_test, pred_LR,
                                              target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0.00      0.00       0.0
weighted avg       0.00      0.00      0.00       0.0
```

```
In [107]: pred_LR = clf.predict(X_test_scaled)
          confusion = confusion_matrix(y_test, pred_LR)
          print("Confusion matrix:\n{}".
                format(confusion))
          print("{:.2f} accuracy, {:.2f} precision, {:.2f} recall".
                .format(metrics.accuracy_score(y_test, pred_LR),
                        metrics.precision_score(y_test, pred_LR),
                        metrics.recall_score(y_test, pred_LR)))
          print("\n")
          print(metrics.classification_report(y_test, pred_LR,
                                              target_names=['Yes', 'Pediatrician', 'Leadership']))
```

```
Confusion matrix:
[[0]]
0.00 accuracy, 0.00 precision, 0.00 recall

              precision    recall  f1-score   support

         Yes       0.00      0.00      0.00       0.0
Pediatrician       0.00      0.00      0.00       0.0
  Leadership       0.00      0.00      0.00       0.0

   micro avg       0.00      0.00      0.00       0.0
   macro avg       0.00      0
```