Pattern_HW2_student_2026

January 28, 2026

1 Employee Attrition Prediction

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

1.0.1 read CSV

[2]:
```
df = pd.read_csv('hr-employee-attrition-with-null.csv')
```

1.0.2 Dataset statistic

[3]:
```
df.describe()
```

[3]:
```
       Unnamed: 0          Age    DailyRate  DistanceFromHome    Education  \
count  1470.000000  1176.000000  1176.000000        1176.00000  1176.000000
mean    734.500000    37.134354   798.875850           9.37500     2.920918
std     424.496761     9.190317   406.957684           8.23049     1.028796
min       0.000000    18.000000   102.000000           1.00000     1.000000
25%     367.250000    30.000000   457.750000           2.00000     2.000000
50%     734.500000    36.000000   798.500000           7.00000     3.000000
75%    1101.750000    43.000000  1168.250000          15.00000     4.000000
max    1469.000000    60.000000  1499.000000          29.00000     5.000000

       EmployeeCount  EmployeeNumber  EnvironmentSatisfaction   HourlyRate  \
count         1176.0     1176.000000              1176.000000  1176.000000
mean             1.0     1031.399660                 2.733844    65.821429
std              0.0      601.188955                 1.092992    20.317323
min              1.0        1.000000                 1.000000    30.000000
25%              1.0      494.750000                 2.000000    48.000000
50%              1.0     1027.500000                 3.000000    66.000000
75%              1.0     1562.250000                 4.000000    84.000000
max              1.0     2068.000000                 4.000000   100.000000

       JobInvolvement  …  RelationshipSatisfaction  StandardHours  \
count     1176.000000  …               1176.000000         1176.0
mean         2.728741  …                  2.694728           80.0
std          0.705280  …                  1.093660            0.0
```

```
min          1.000000  ...                 1.000000            80.0
25%          2.000000  ...                 2.000000            80.0
50%          3.000000  ...                 3.000000            80.0
75%          3.000000  ...                 4.000000            80.0
max          4.000000  ...                 4.000000            80.0

       StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
count       1176.000000        1176.000000            1176.000000
mean           0.752551          11.295068               2.787415
std            0.822550           7.783376               1.290507
min            0.000000           0.000000               0.000000
25%            0.000000           6.000000               2.000000
50%            1.000000          10.000000               3.000000
75%            1.000000          15.000000               3.000000
max            3.000000          40.000000               6.000000

       WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
count      1176.000000     1176.000000         1176.000000
mean          2.770408        7.067177            4.290816
std           0.705004        6.127836            3.630901
min           1.000000        0.000000            0.000000
25%           2.000000        3.000000            2.000000
50%           3.000000        5.000000            3.000000
75%           3.000000       10.000000            7.000000
max           4.000000       37.000000           18.000000

       YearsSinceLastPromotion  YearsWithCurrManager
count              1176.000000           1176.000000
mean                  2.159014              4.096939
std                   3.163524              3.537393
min                   0.000000              0.000000
25%                   0.000000              2.000000
50%                   1.000000              3.000000
75%                   2.250000              7.000000
max                  15.000000             17.000000

[8 rows x 27 columns]
```

[4]:
```
df.head()
```

[4]:
```
   Unnamed: 0   Age Attrition     BusinessTravel  DailyRate  \
0           0  41.0       Yes      Travel_Rarely        NaN
1           1   NaN        No                NaN      279.0
2           2  37.0       Yes                NaN     1373.0
3           3   NaN        No  Travel_Frequently     1392.0
4           4  27.0        No      Travel_Rarely      591.0
```

```
            Department  DistanceFromHome  Education EducationField  \
0                  NaN               1.0        NaN  Life Sciences
1  Research & Development               NaN        NaN  Life Sciences
2                  NaN               2.0        2.0            NaN
3  Research & Development               3.0        4.0  Life Sciences
4  Research & Development               2.0        1.0        Medical

   EmployeeCount  …  RelationshipSatisfaction  StandardHours  \
0            1.0  …                       1.0           80.0
1            1.0  …                       4.0            NaN
2            1.0  …                       NaN           80.0
3            NaN  …                       3.0            NaN
4            1.0  …                       4.0           80.0

  StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  WorkLifeBalance  \
0              0.0                8.0                    0.0              NaN
1              1.0               10.0                    NaN              3.0
2              0.0                7.0                    3.0              NaN
3              NaN                8.0                    3.0              NaN
4              1.0                6.0                    NaN              3.0

  YearsAtCompany  YearsInCurrentRole YearsSinceLastPromotion  \
0            6.0                 NaN                     0.0
1           10.0                 NaN                     NaN
2            NaN                 0.0                     NaN
3            8.0                 NaN                     3.0
4            2.0                 2.0                     2.0

   YearsWithCurrManager
0                   NaN
1                   7.0
2                   0.0
3                   0.0
4                   NaN

[5 rows x 36 columns]
```

1.0.3 Feature transformation

[5]:
```
df.loc[df["Attrition"] == "no", "Attrition"] = 0.0
df.loc[df["Attrition"] == "yes", "Attrition"] = 1.0
string_categorical_col = ['Department', 'Attrition', 'BusinessTravel',␣
 ↪'EducationField', 'Gender', 'JobRole',
                              'MaritalStatus', 'Over18', 'OverTime']

# ENCODE STRING COLUMNS TO CATEGORICAL COLUMNS
for col in string_categorical_col:
```

```
    # INSERT CODE HERE
    df[col] = pd.Categorical(df[col]).codes

# HANDLE NULL NUMBERS
# I don't thinkwe need to handle null?
# INSERT CODE HERE

df = df.loc[:, ~df.columns.isin(['EmployeeNumber', 'Unnamed: 0',␣
 ↪'EmployeeCount', 'StandardHours', 'Over18'])]
```

1.0.4 Spliting data into train and test

[6]:
```
from sklearn.model_selection import train_test_split
```

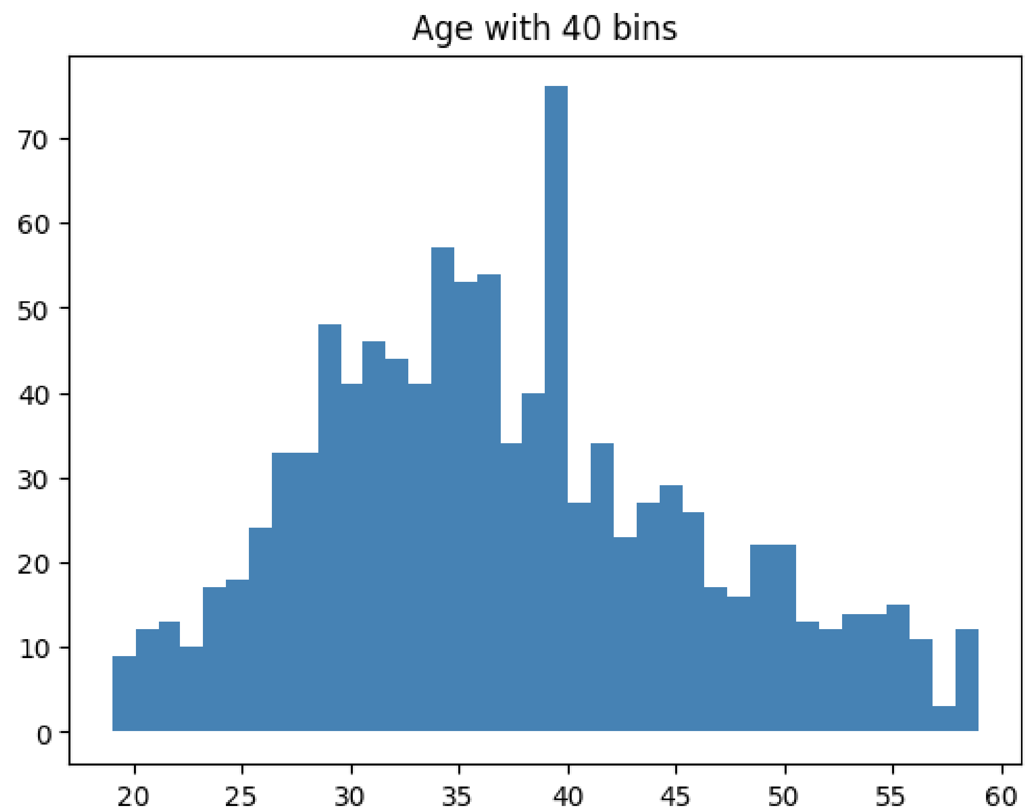
[7]:
```
X = df.drop(["Attrition"], axis=1)
Y = df["Attrition"]

x_train, x_test, y_train, y_test = train_test_split(X, Y, stratify=Y,␣
 ↪test_size=0.1, random_state=12345)
```

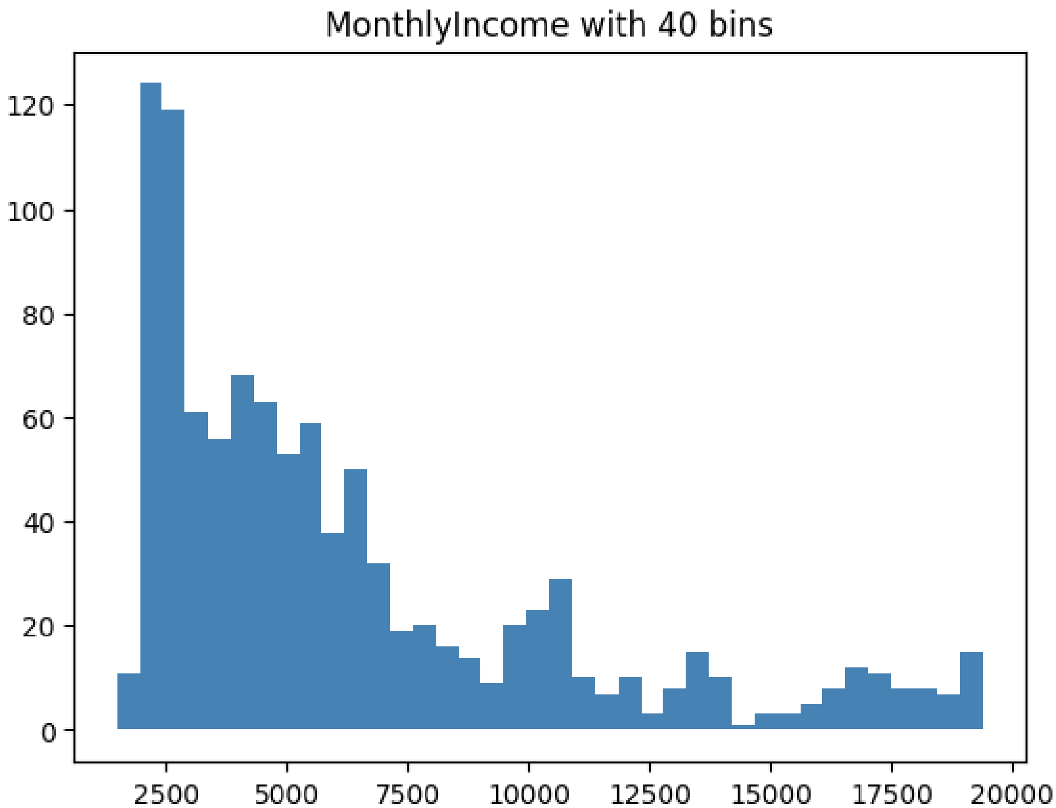
1.0.5 Display histogram of each feature

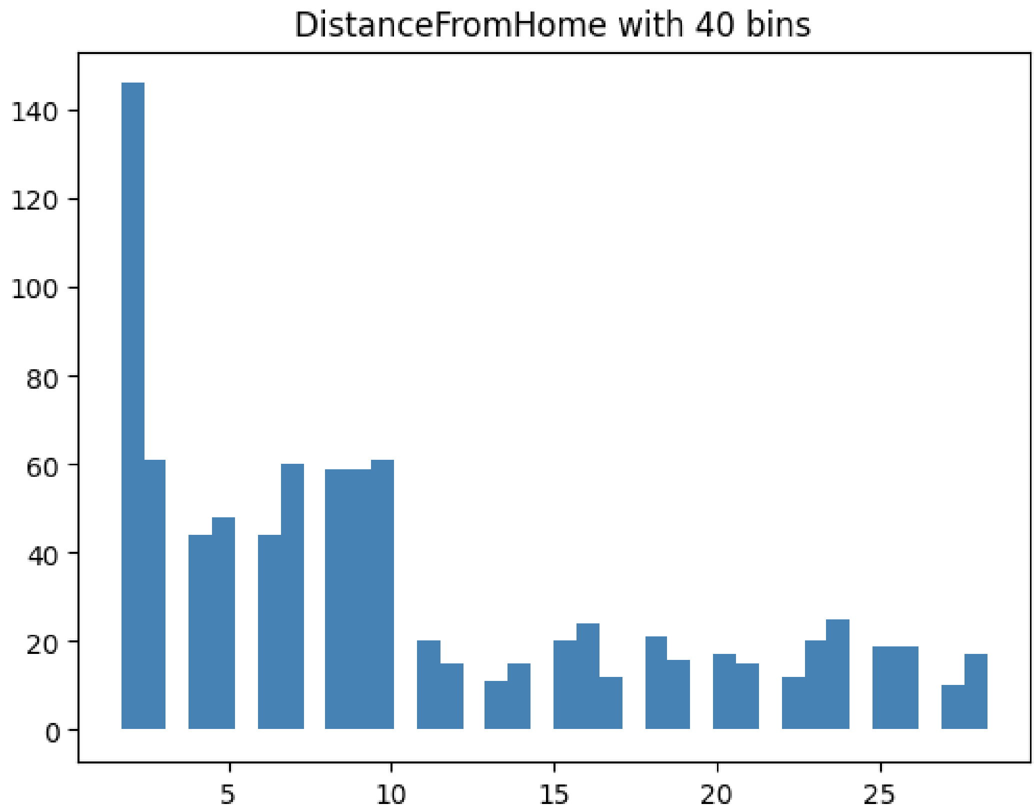
[8]:
```
def display_histogram(df, col_name, n_bin = 40):
    # INSERT CODE HERE
    col_nonan = df[col_name][~np.isnan(df[col_name])]

    # col = np.array(df[col_name])
    # col_nonan = np.array( col[~ np.isnan(col)] )

    # hist is the count for each bin
    # bin_edge is the edge values of the bins
    hist, bin_edge = np.histogram(col_nonan, n_bin)
    bin_edge[0] = -np.inf
    bin_edge[-1] = np.inf

    # plot the histogram
    plt.fill_between(bin_edge.repeat(2)[1:-1], hist.repeat(2),␣
 ↪facecolor='steelblue')
    plt.title(col_name + " with " + str(n_bin) + " bins")
    plt.show()
```

1.0.6 T4. Observe the histogram for Age, MonthlyIncome and DistanceFromHome. How many bins have zero counts? Do you think this is a good discretization? Why?

[9]:
```
display_histogram(x_train, "Age")
display_histogram(x_train, "MonthlyIncome")
display_histogram(x_train, "DistanceFromHome")
```




MonthlyIncome with 40 bins
0
20
40
60
80
100
120
2500
5000
7500
10000
12500
15000
17500
20000

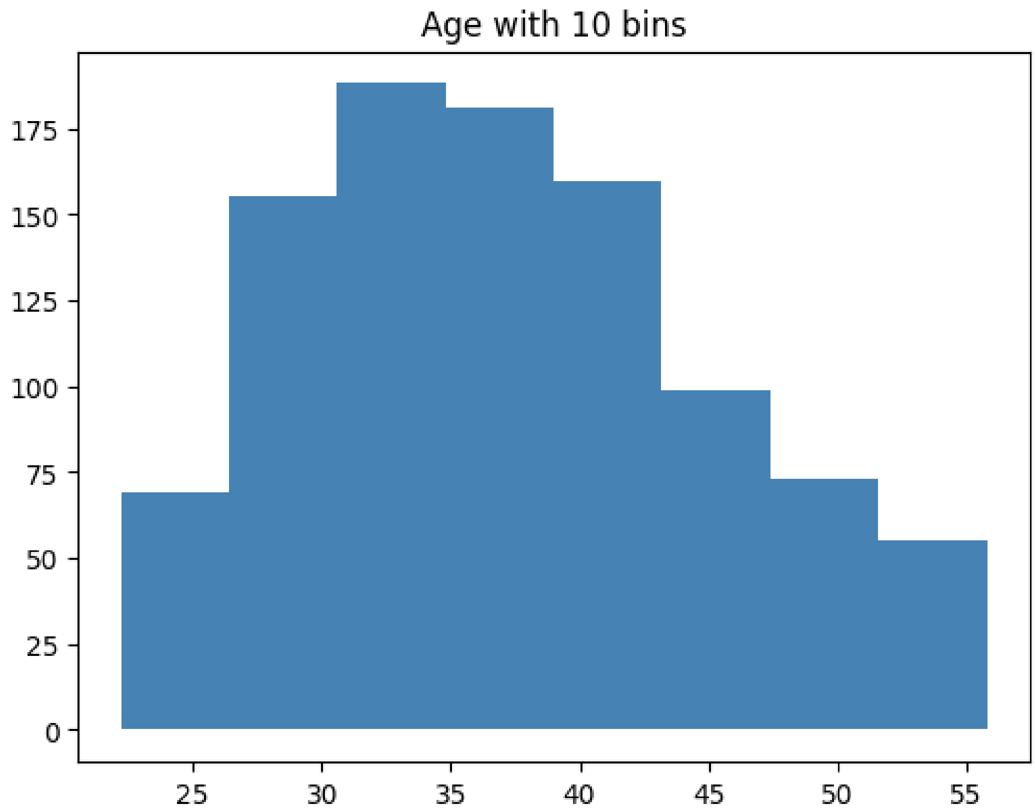


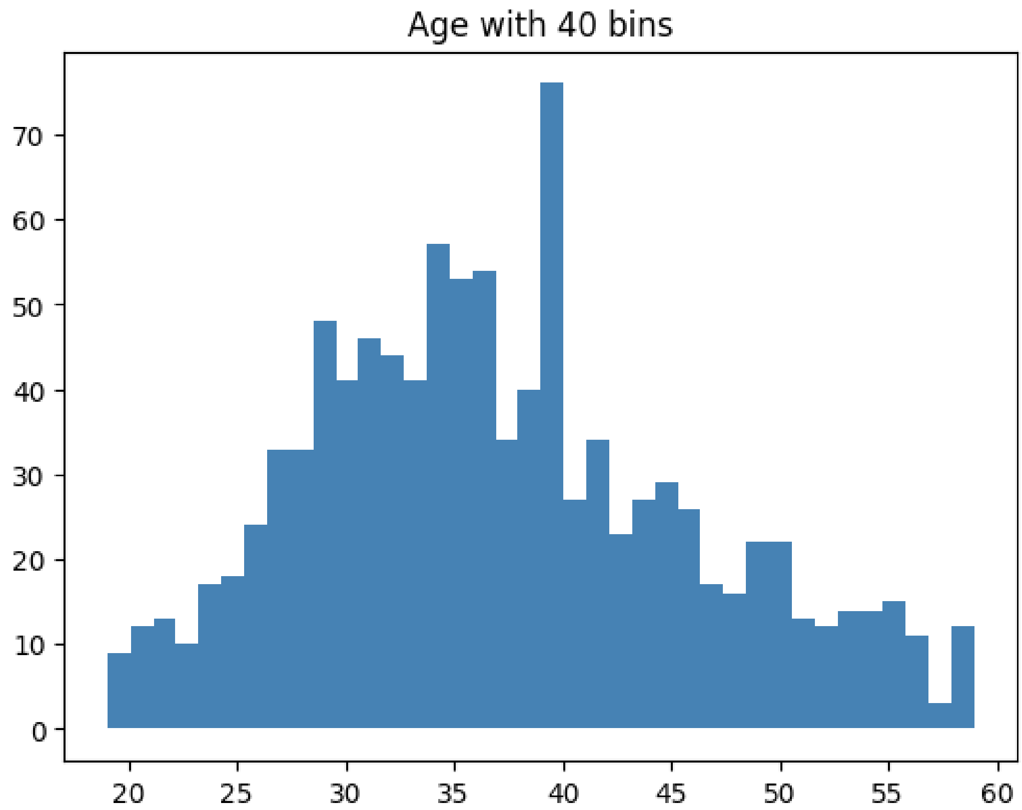
1.0.7 T5. Can we use a Gaussian to estimate this histogram? Why? What about a Gaussian Mixture Model (GMM)?

```
As 40 bins, Age and MonthlyIncome has no zero counts bin,
but DistanceFromHome have 11 bins with zero counts. I don't
think this is a good discretization because there are zero
counts bin which lead to probability of 0 due to less data.
```

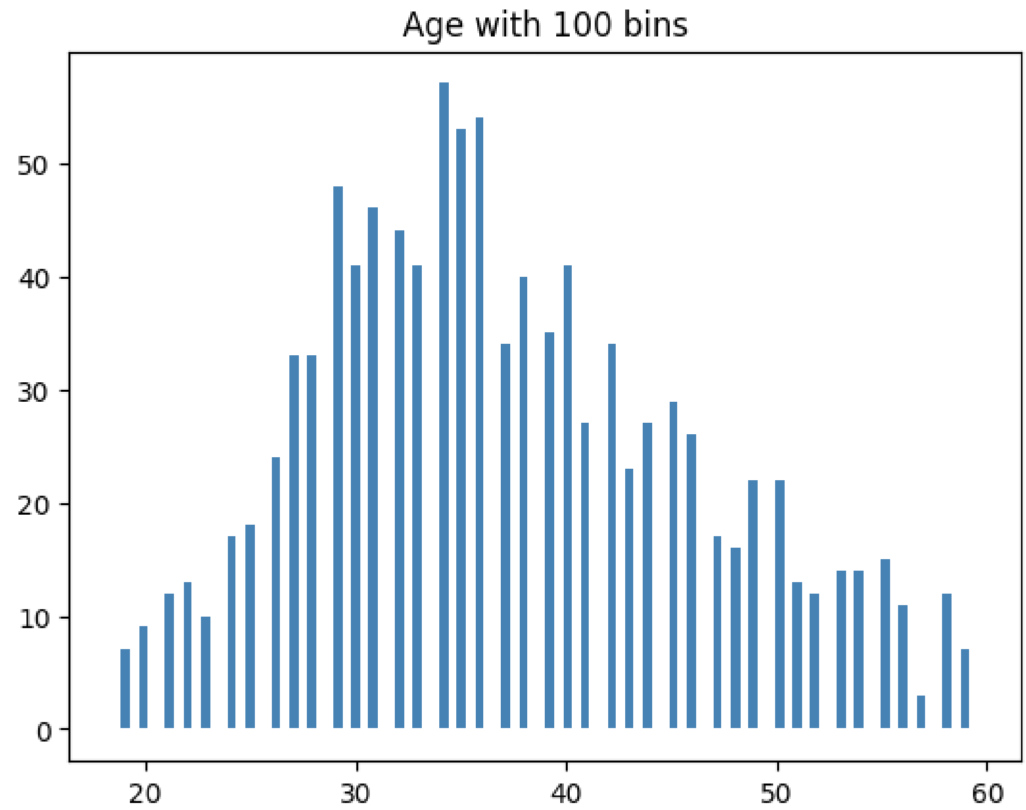
1.0.8 T6. Now plot the histogram according to the method described above (with 10, 40, and 100 bins) and show 3 plots each for Age, MonthlyIncome, and DistanceFromHome. Which bin size is most sensible for each features? Why?

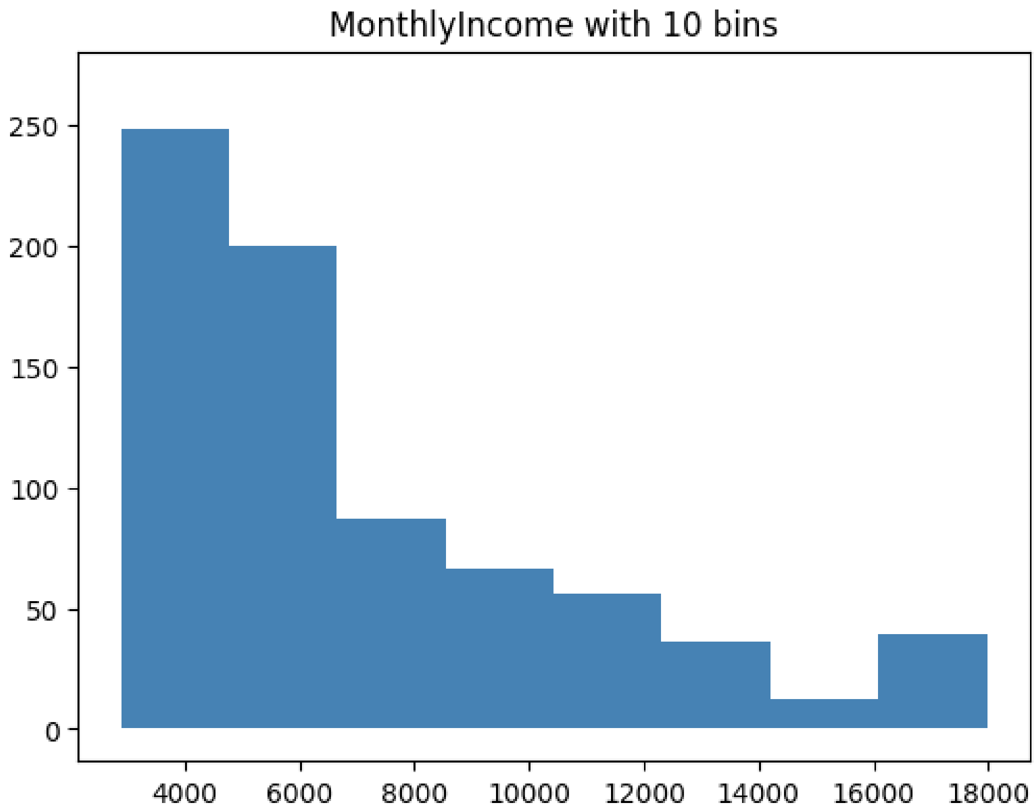
[11]:
```
columns = ["Age", "MonthlyIncome", "DistanceFromHome"]
num_bins = [10, 40, 100]

for col in columns:
    for num in num_bins:
        display_histogram(x_train, col, num)
```

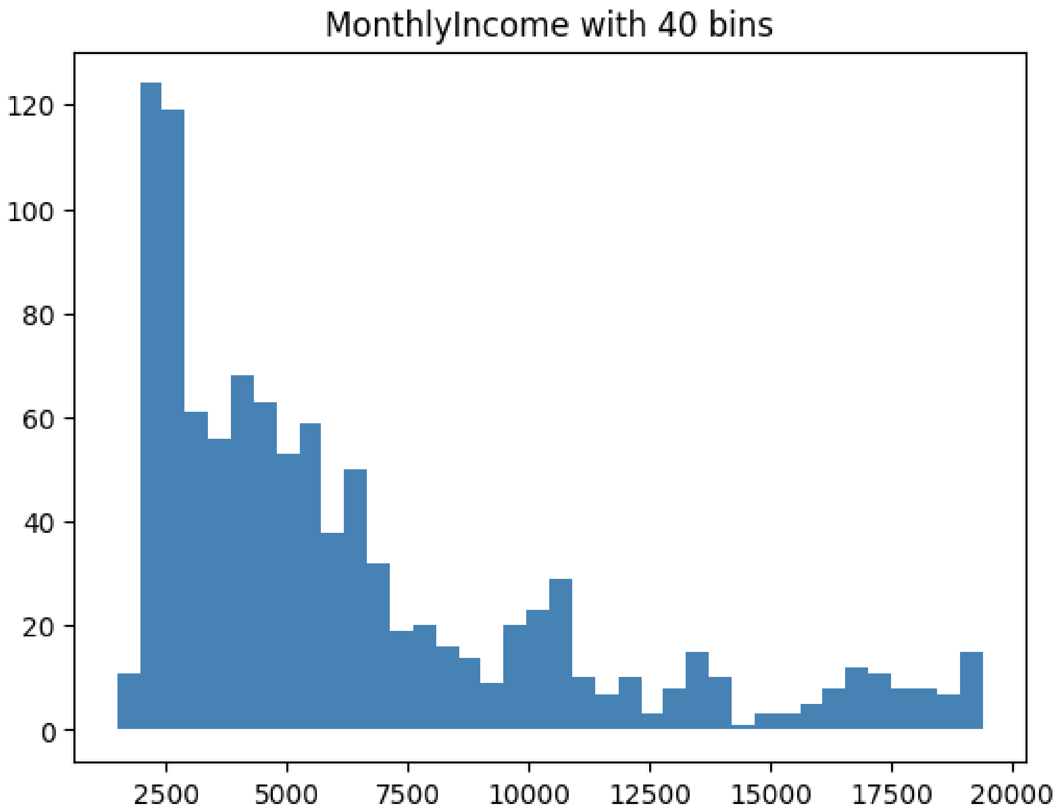

Age with 10 bins
175
150
125
100
75
50
25
0
25
30
35
40
45
50
55



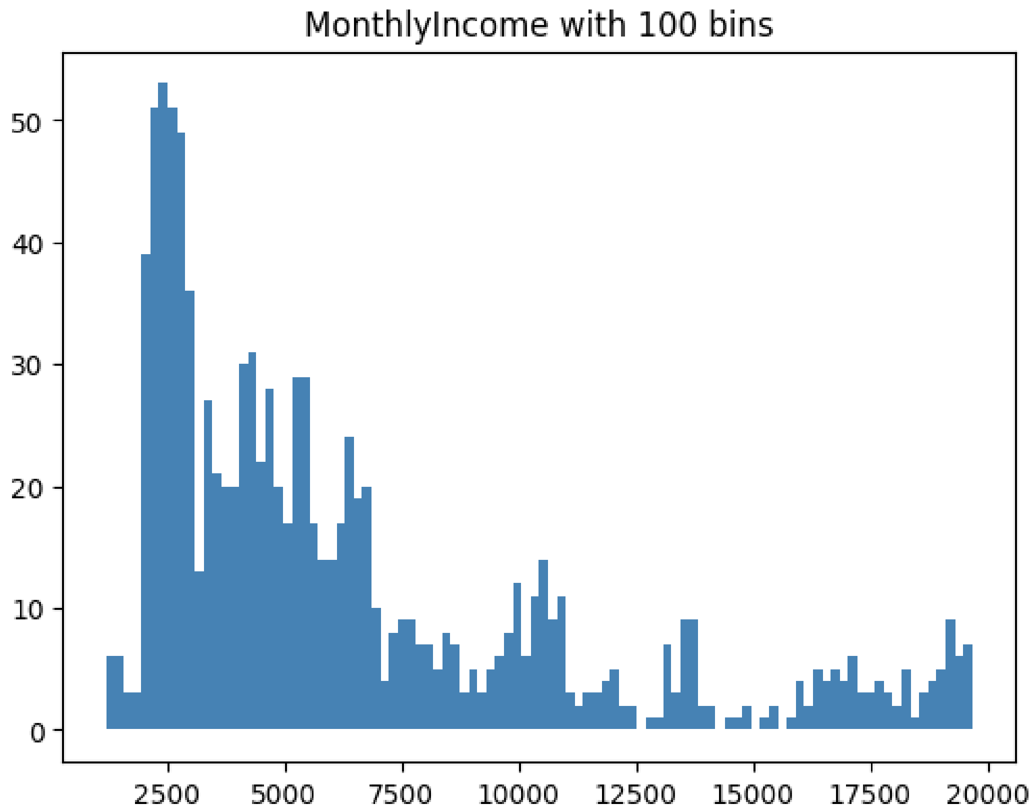
Age with 40 bins
0
10
20
30
40
50
60
70
20
25
30
35
40
45
50
55
60

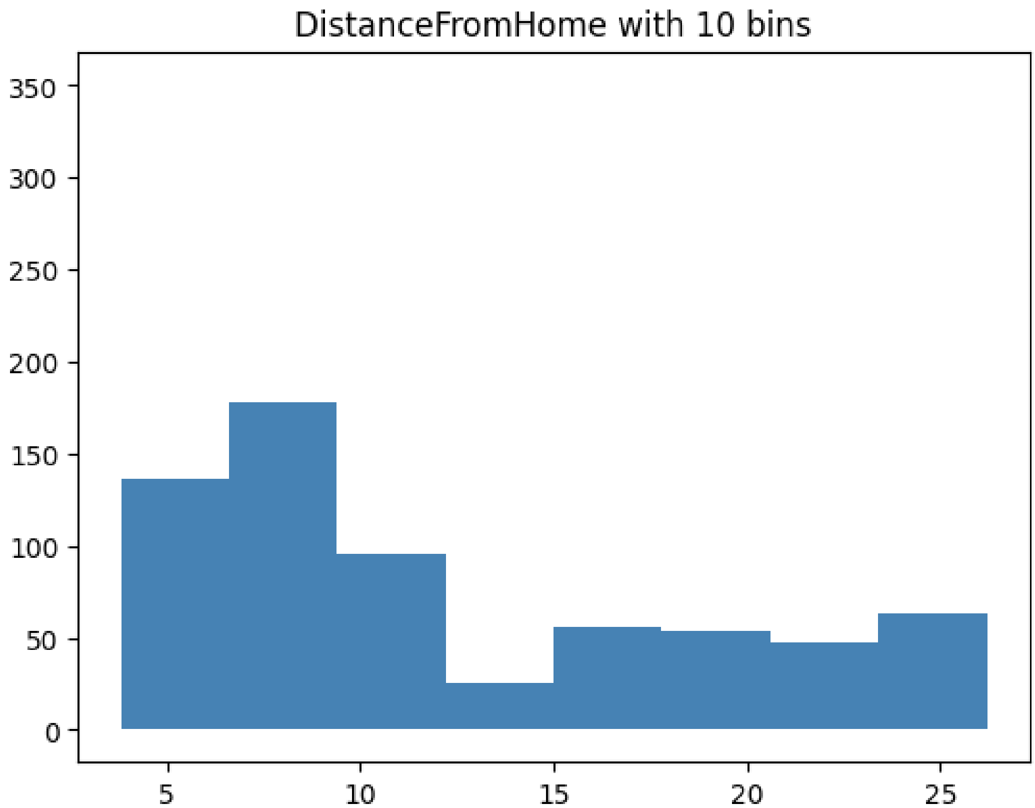

Age with 100 bins
0
10
20
30
40
50
20
30
40
50
60

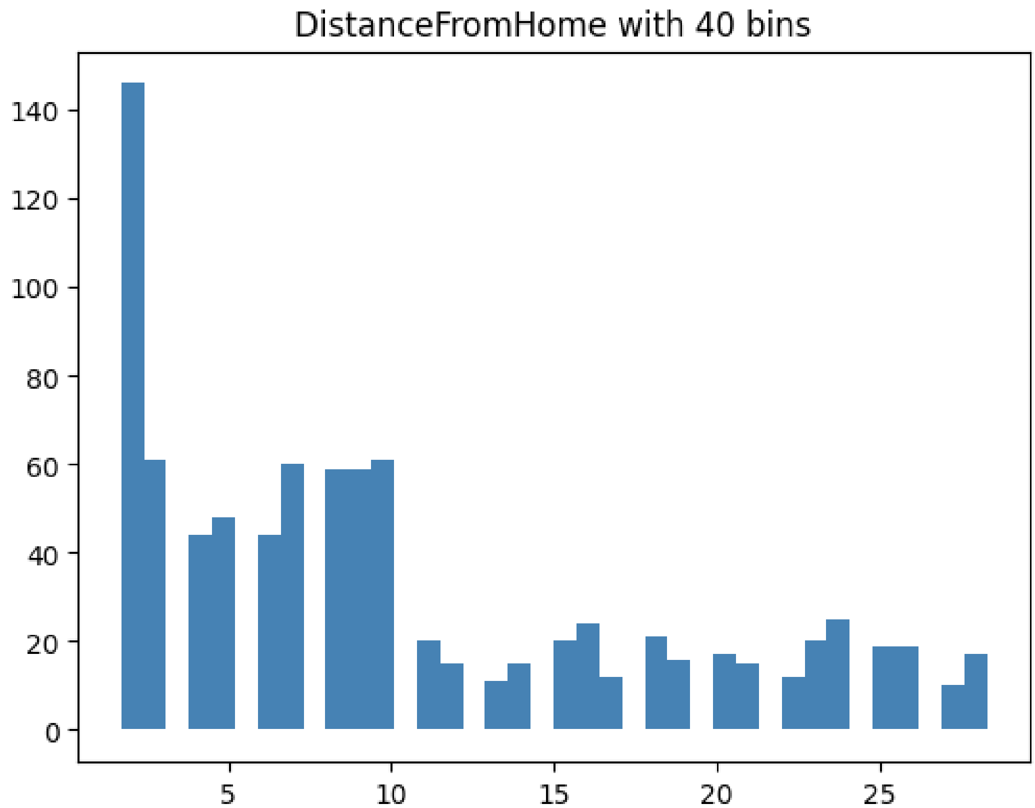

MonthlyIncome with 10 bins
250
200
150
100
50
0
4000
6000
8000
10000
12000
14000
16000
18000

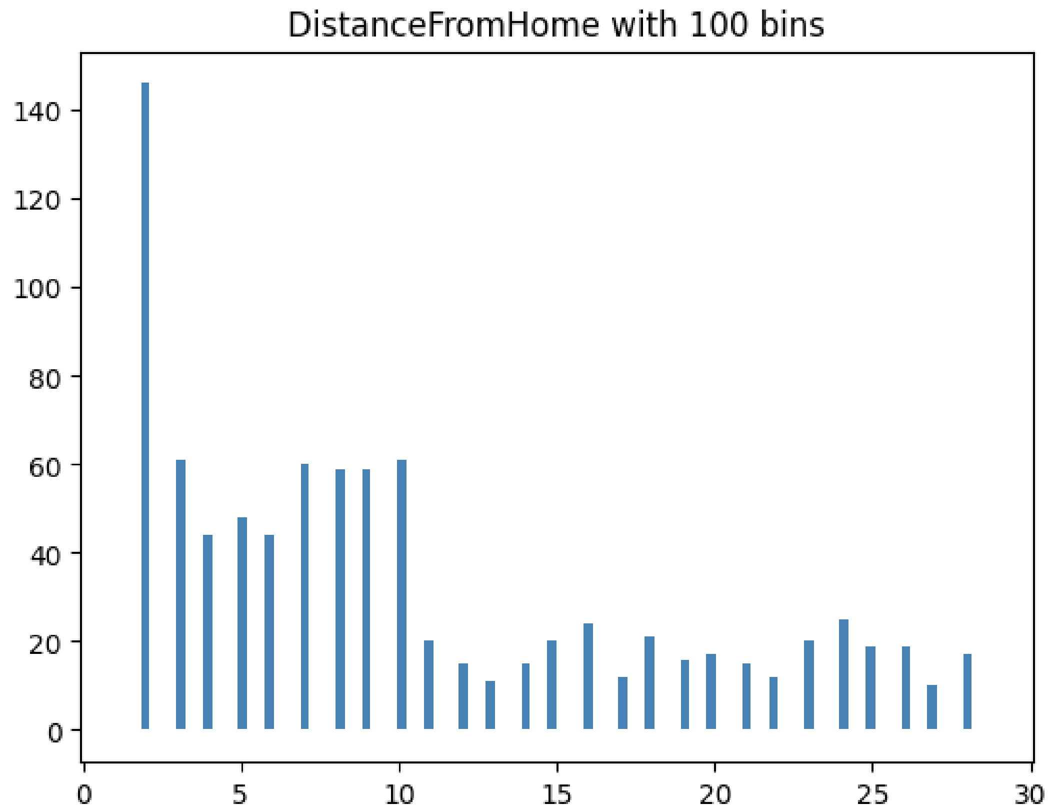


MonthlyIncome with 40 bins
120
100
80
60
40
20
0
2500
5000
7500
10000
12500
15000
17500
20000


MonthlyIncome with 100 bins
0
10
20
30
40
50
2500
5000
7500
10000
12500
15000
17500
20000


DistanceFromHome with 10 bins
350
300
250
200
150
100
50
0
5
10
15
20
25


DistanceFromHome with 40 bins
140
120
100
80
60
40
20
0
5
10
15
20
25



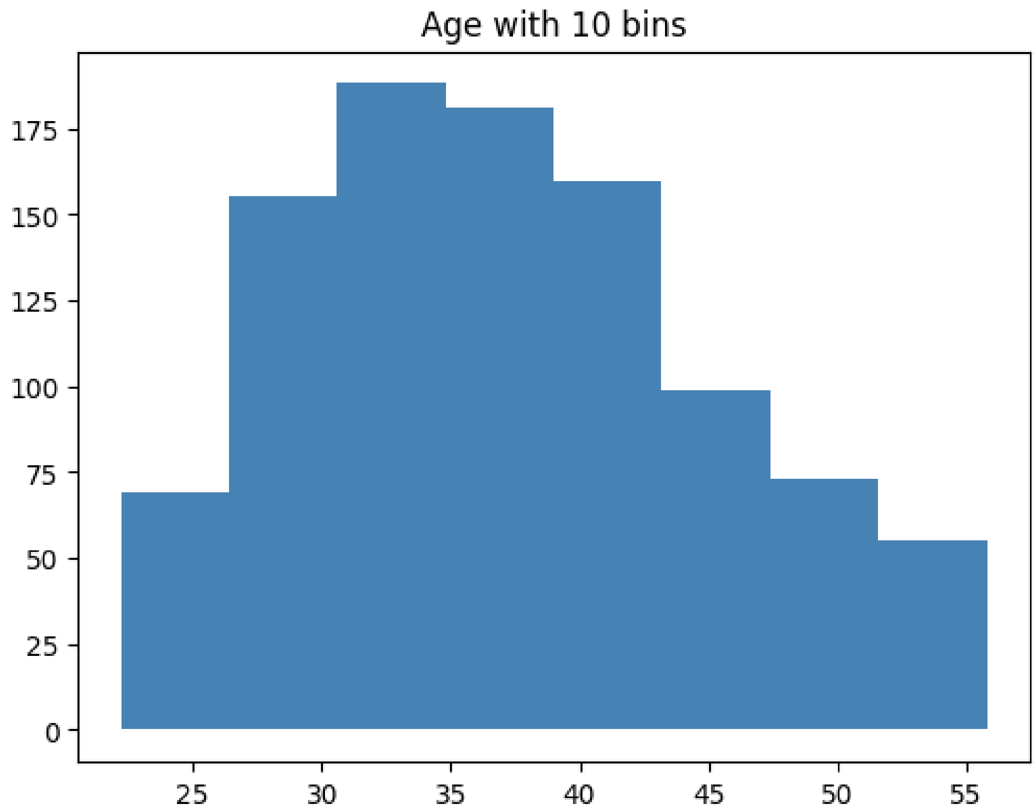

```
For Age and MonthlyIncome:
I think 40 bins is the best because there are no zero counts.
Well, 10 bins also doesn't have zero counts too, but you can
elaborate more details in the 40 bins. Anyway, the 100 bins
got some zero counts, so it isn't a good discretization.

For DistanceFromHome:
40 bins and 100 bins will both have a zero counts bin.
So that, the only good number of bins in these three are 10.
```

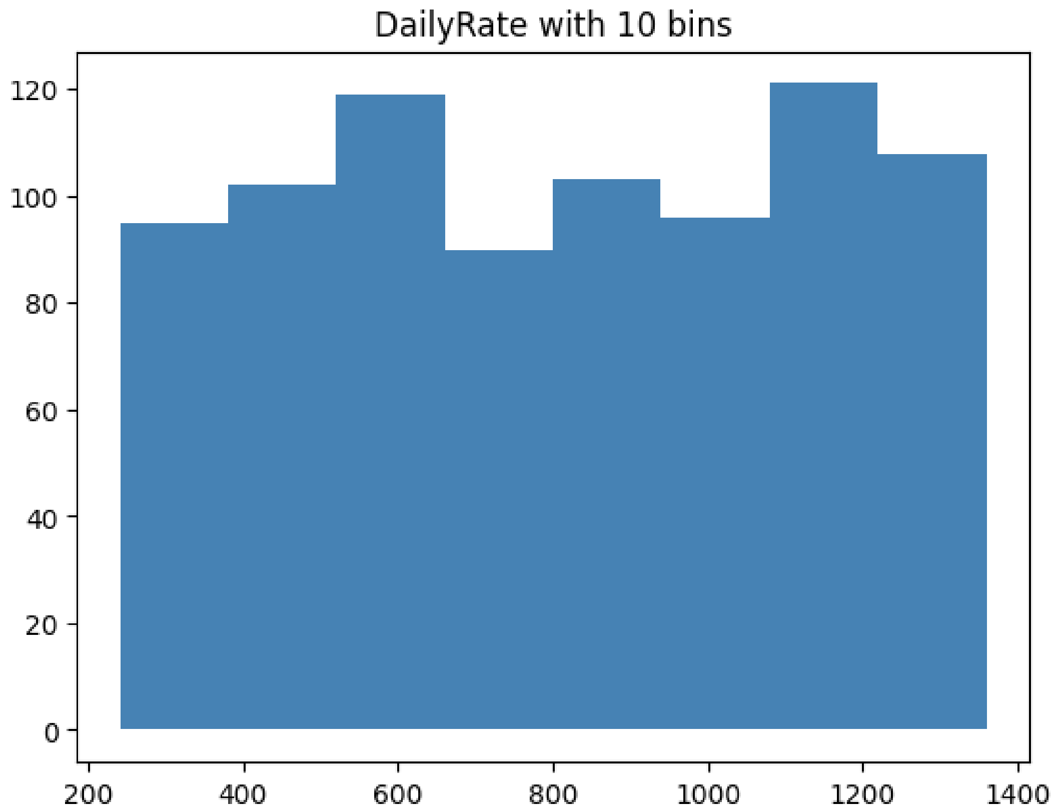
1.0.9 T7. For the rest of the features, which one should be discretized in order to be modeled by histograms? What are the criteria for choosing whether we should discretize a feature or not? Answer this and discretize those features into 10 bins each. In other words, figure out the bin edge for each feature, then use digitize() to convert the features to discrete values

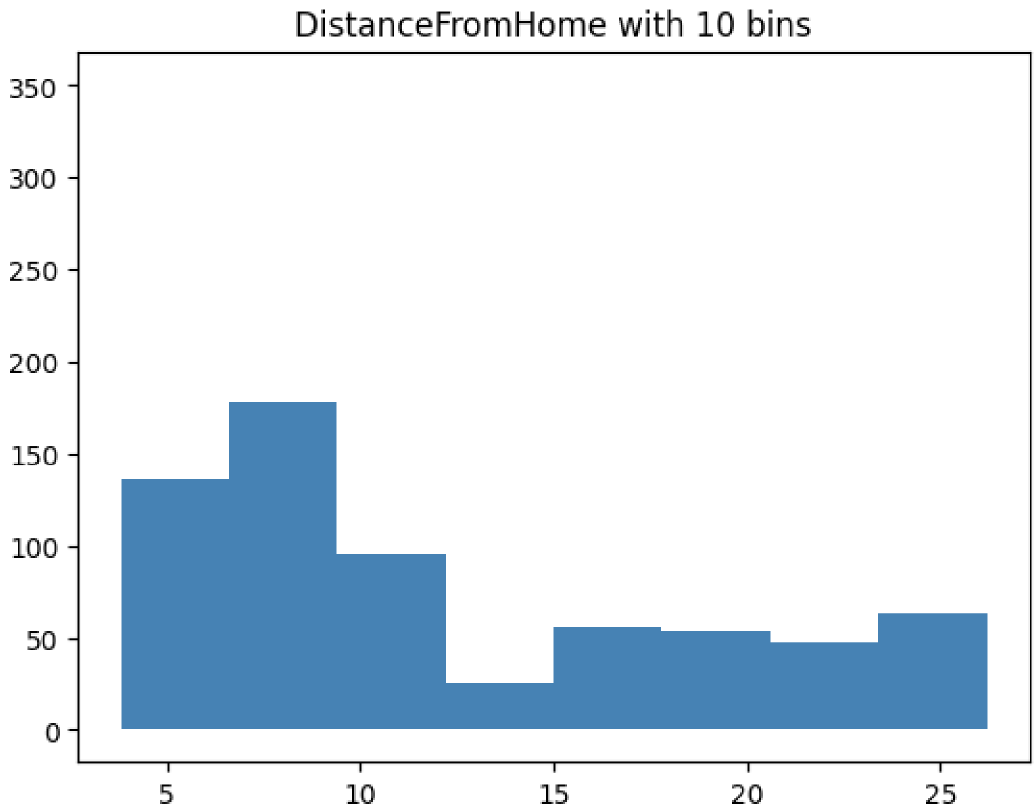
```
First, the encoded from categorical values shouldn't be discretized
because the value is already discretized via the encoded function.
For me, what should be discretized is the one with continuous range
value or have many unique values. The threshold for considering if
that features should be discretized or not is > 40 (or any suit number)
unique values.
```

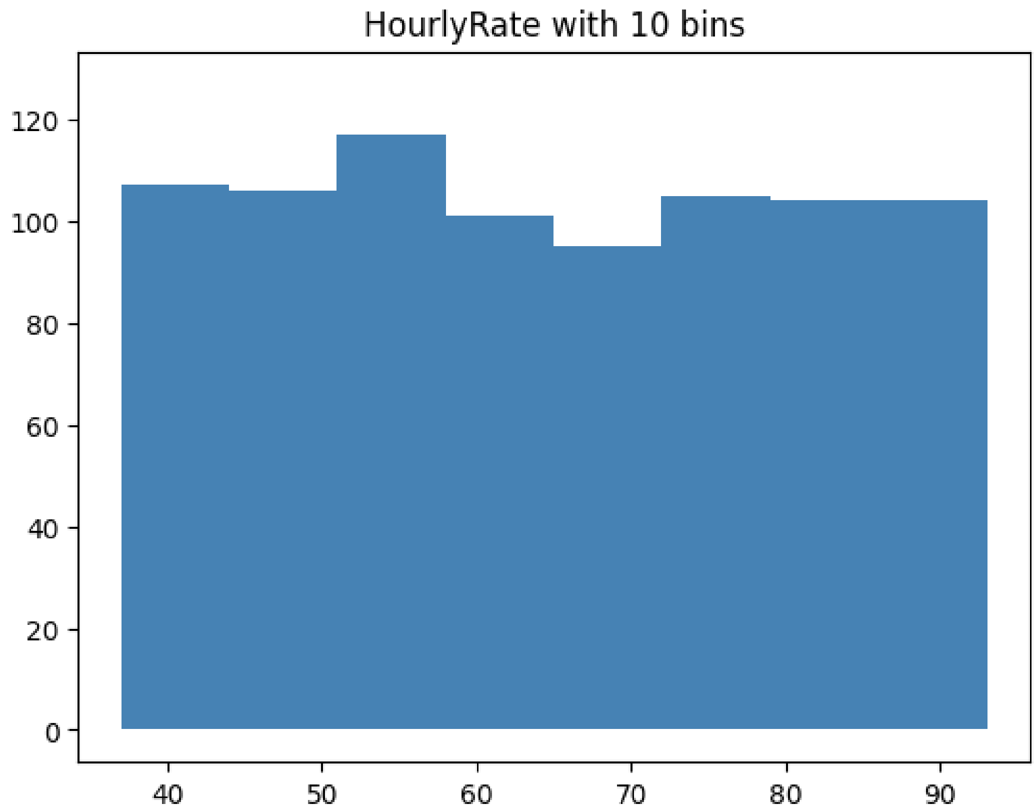
[12]:

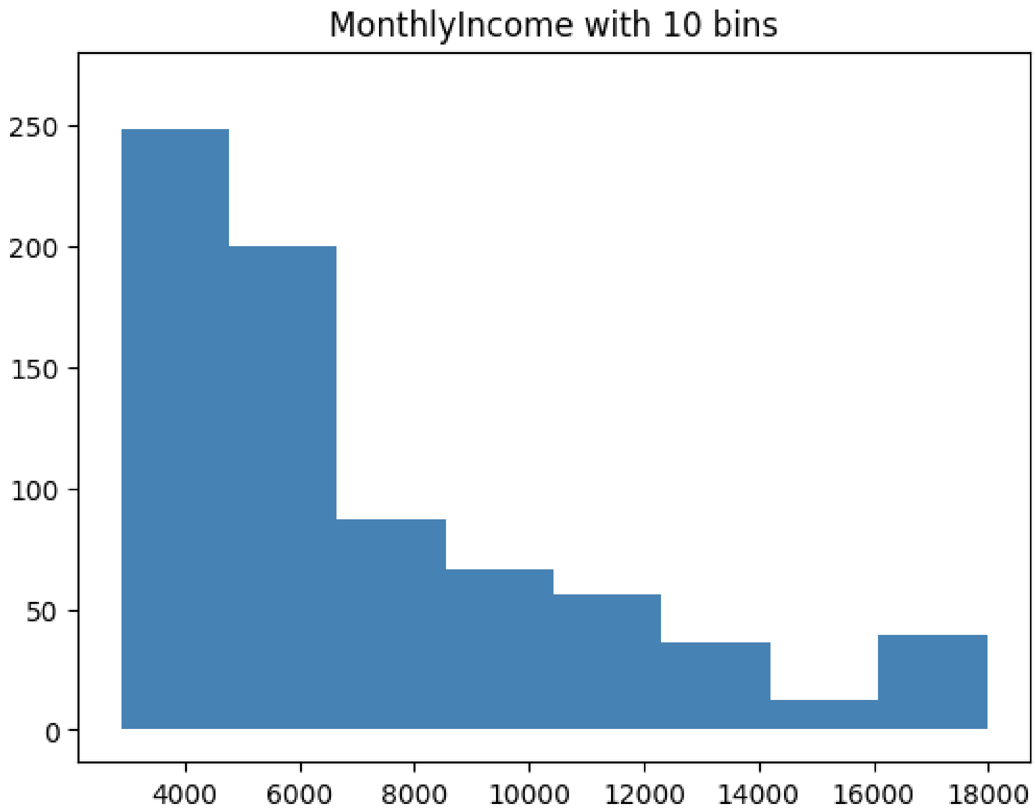
```
def hist(array, col_name, n_bin=10):
    nonan = array[~np.isnan(array)]

    # hist is the count for each bin
    # bin_edge is the edge values of the bins
    hist, bin_edges = np.histogram(nonan, n_bin)
    bin_edges[0] = -np.inf
    bin_edges[-1] = np.inf

    bin_indices = np.full_like(array, np.nan, dtype=float)
    bin_indices[~np.isnan(array)] = np.digitize(nonan, bin_edges)

    # plot the histogram
    plt.fill_between(bin_edges.repeat(2)[1:-1], hist.repeat(2),␣
 ↪facecolor='steelblue')
    plt.title(col_name + " with " + str(n_bin) + " bins")
    plt.show()

    return bin_indices, bin_edges

discretize = []

for col in x_train.columns:
    if (x_train[col].nunique() > 10):
        x_train[col], _ = hist(x_train[col], col)
        discretize.append(col)

print(discretize)
```

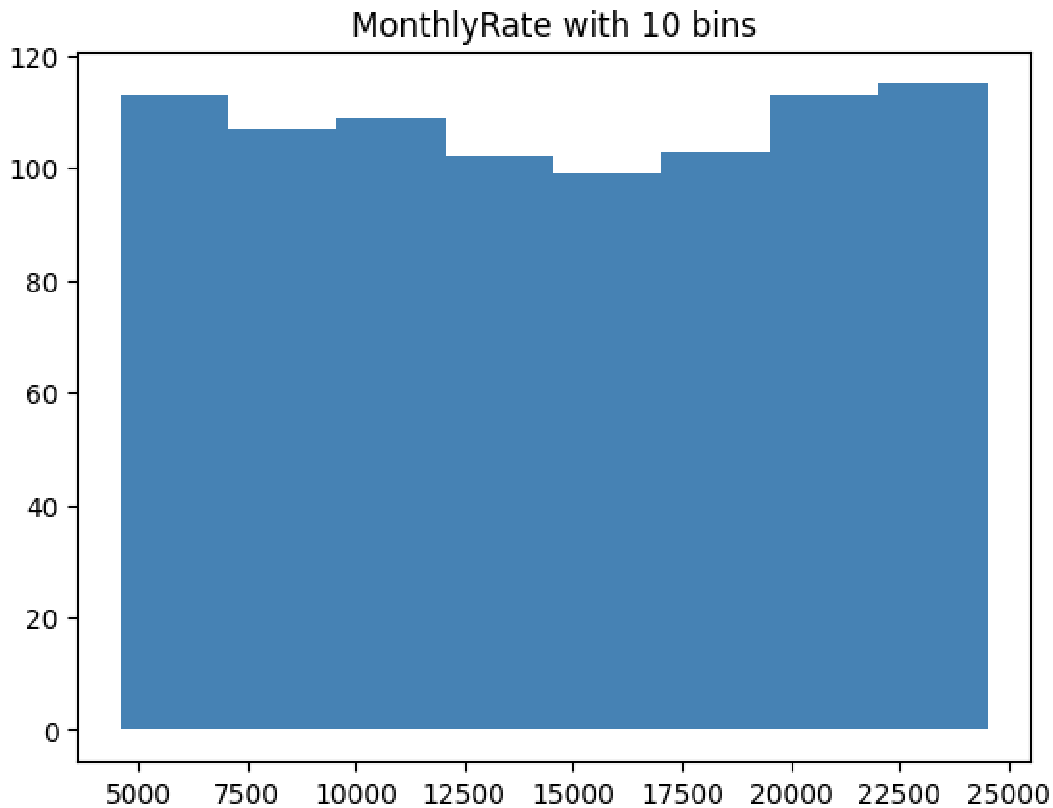

Age with 10 bins
175
150
125
100
75
50
25
0
25
30
35
40
45
50
55


DailyRate with 10 bins
0
20
40
60
80
100
120
200
400
600
800
1000
1200
1400

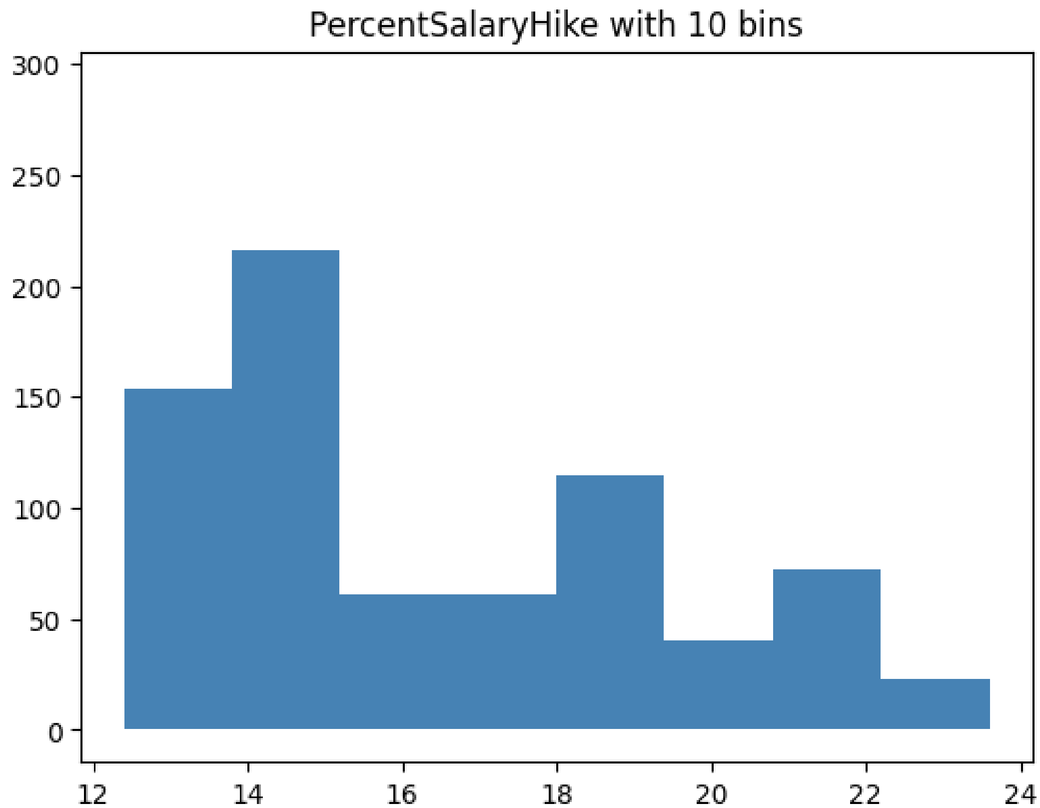

DistanceFromHome with 10 bins
350
300
250
200
150
100
50
0
5
10
15
20
25


HourlyRate with 10 bins
120
100
80
60
40
20
0
40
50
60
70
80
90

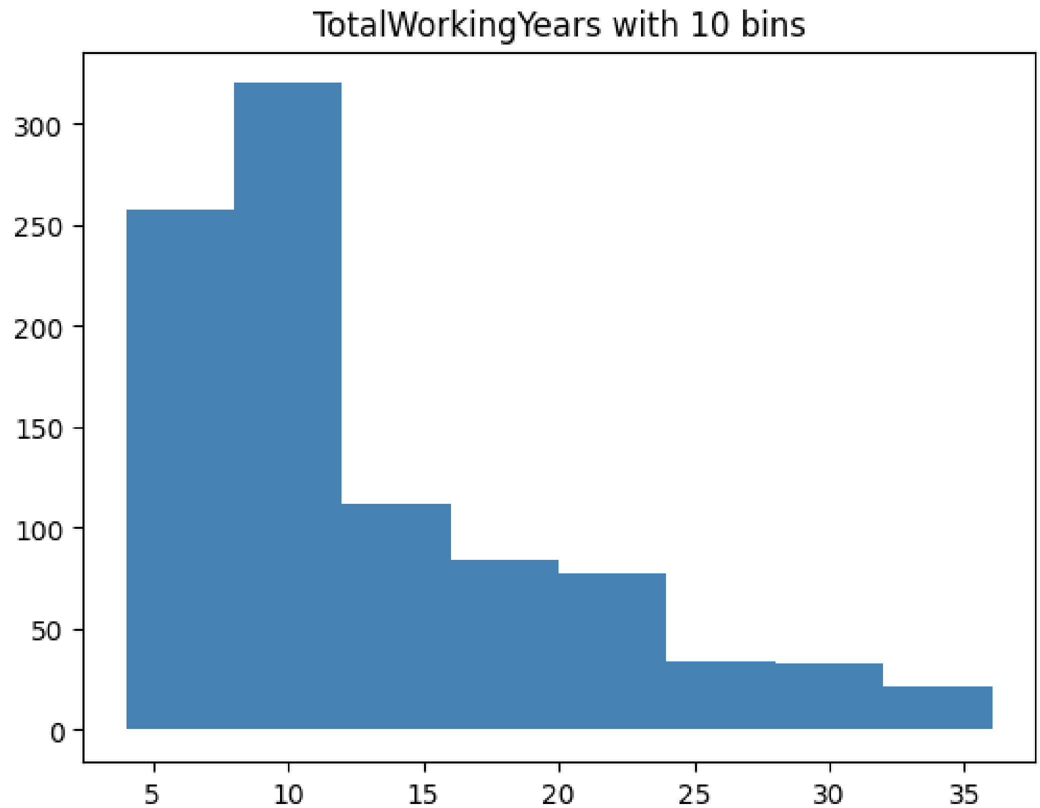

MonthlyIncome with 10 bins
250
200
150
100
50
0
4000
6000
8000
10000
12000
14000
16000
18000

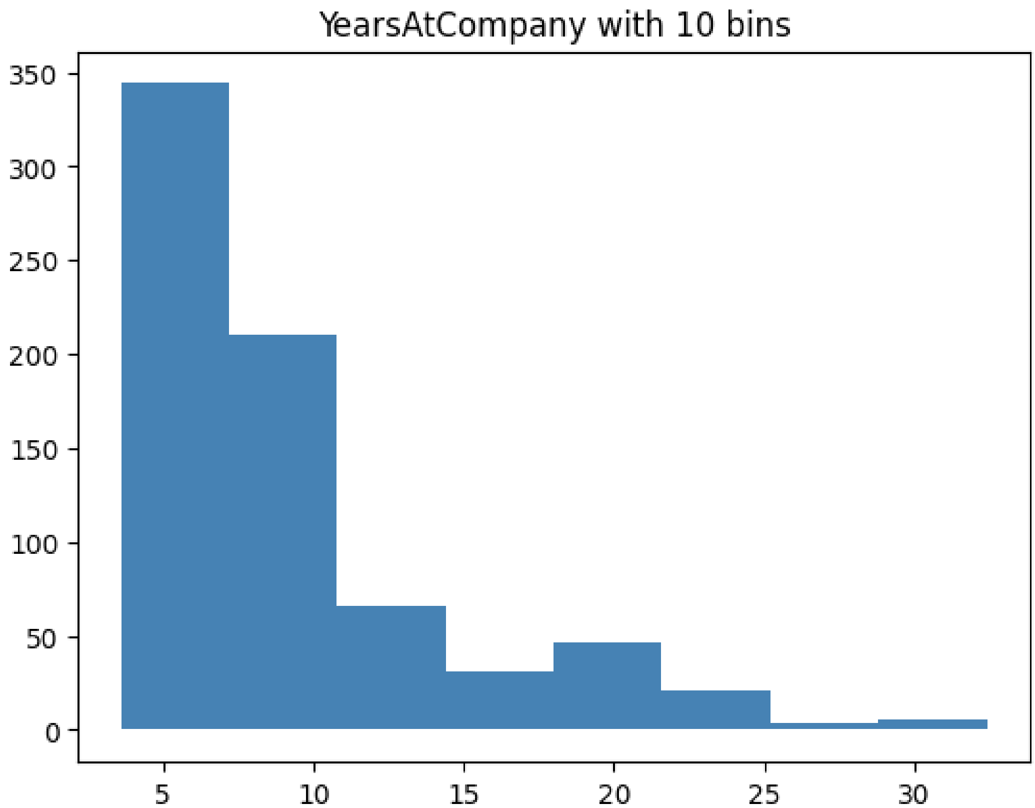


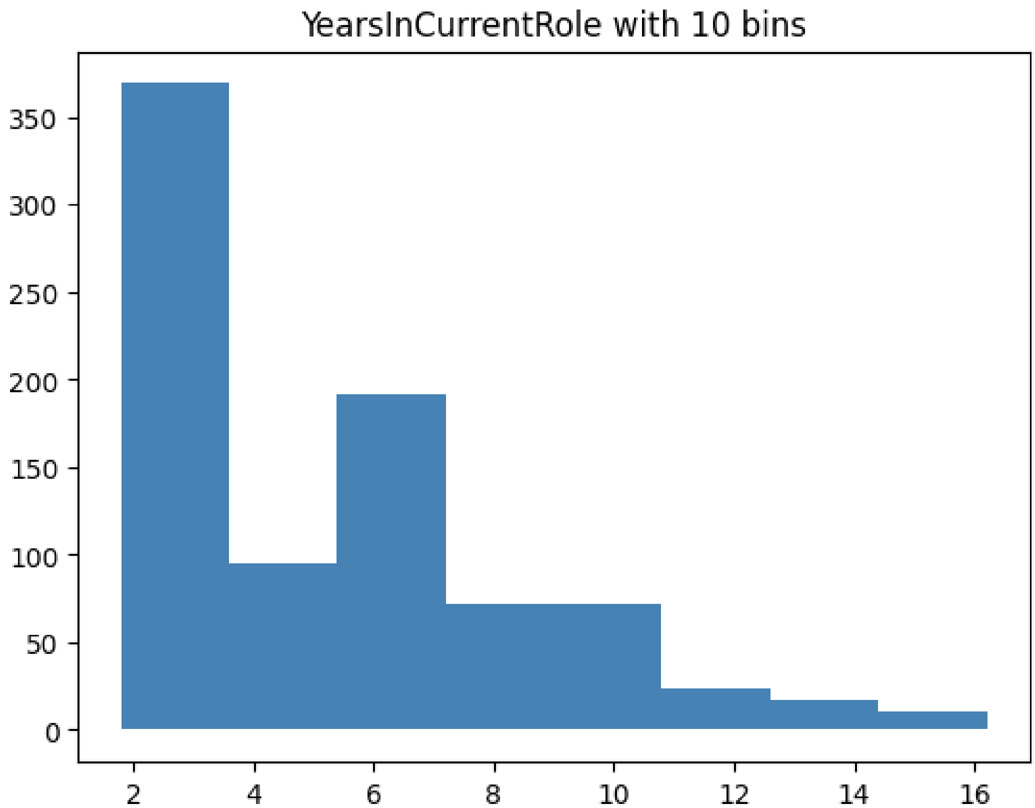
MonthlyRate with 10 bins
120
100
80
60
40
20
0
5000
7500
10000
12500
15000
17500
20000
22500
25000

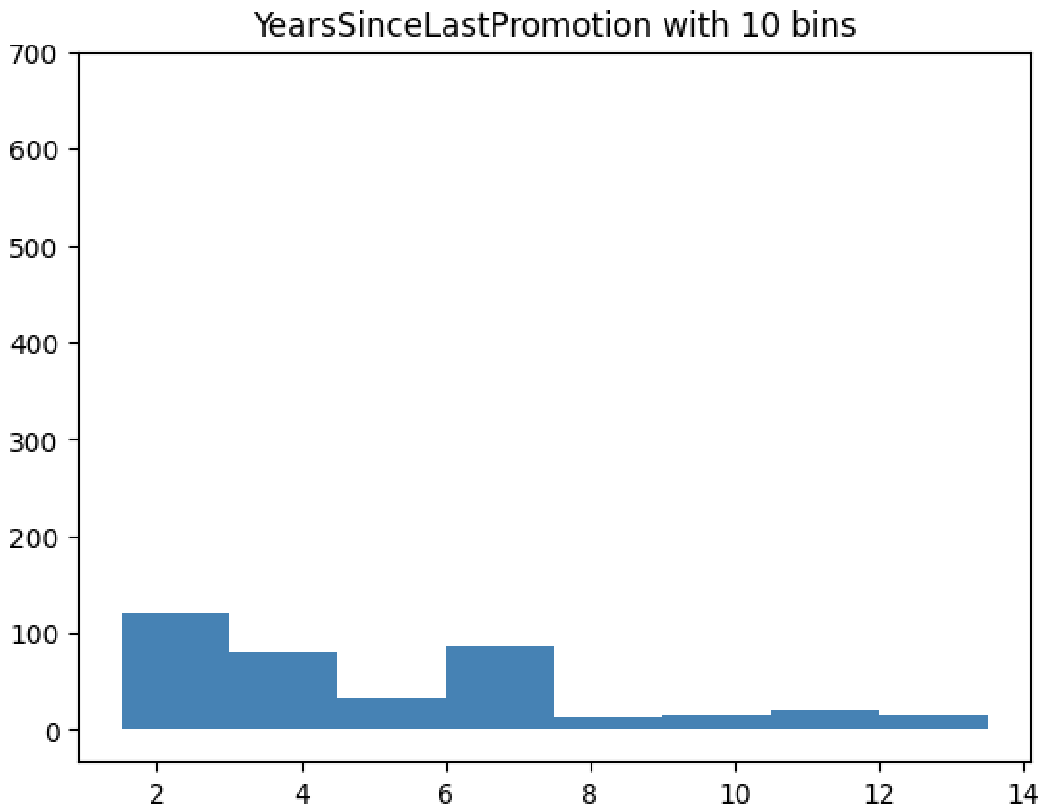


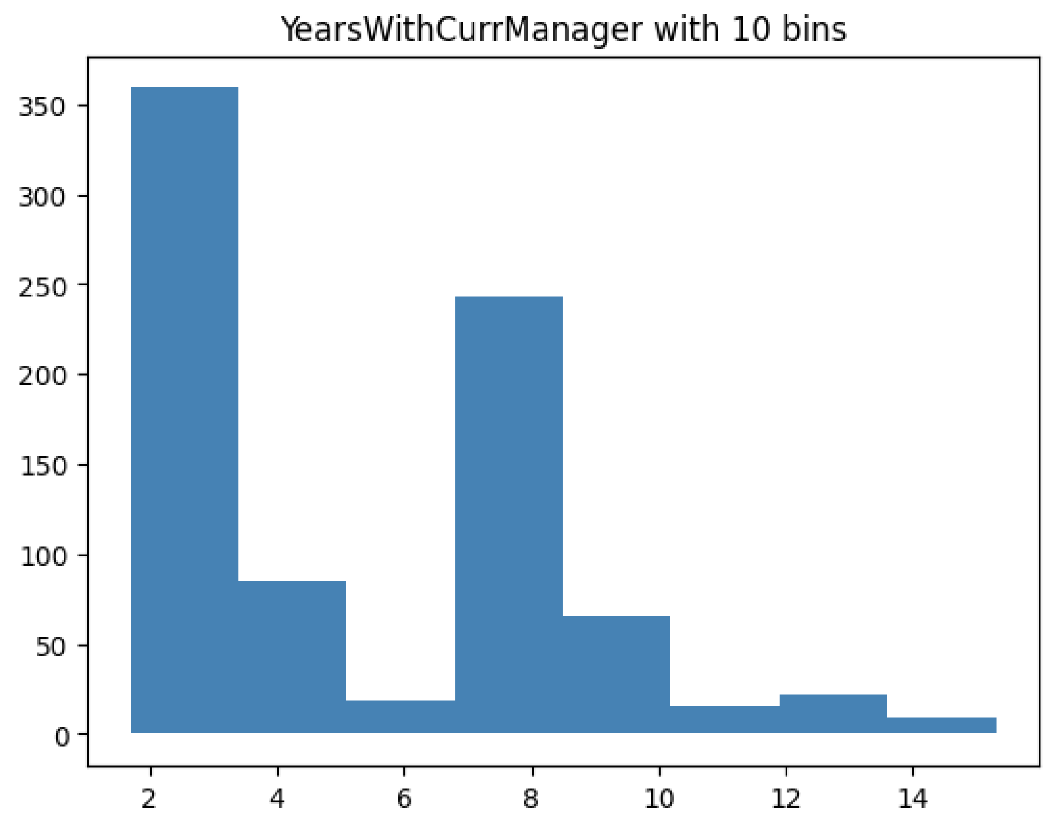
PercentSalaryHike with 10 bins
300
250
200
150
100
50
0
12
14
16
18
20
22
24


TotalWorkingYears with 10 bins
300
250
200
150
100
50
0
5
10
15
20
25
30
35


YearsAtCompany with 10 bins
350
300
250
200
150
100
50
0
5
10
15
20
25
30


YearsInCurrentRole with 10 bins
350
300
250
200
150
100
50
0
2
4
6
8
10
12
14
16


YearsSinceLastPromotion with 10 bins
700
600
500
400
300
200
100
0
2
4
6
8
10
12
14



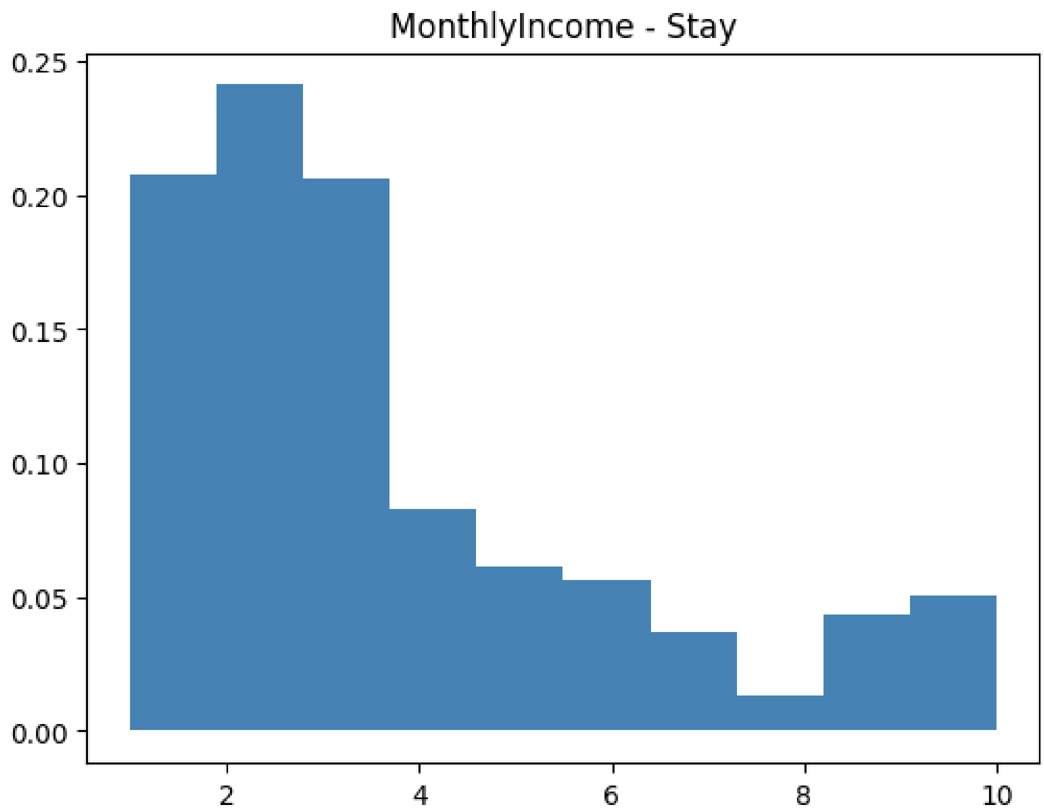
['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate', 'MonthlyIncome',
'MonthlyRate', 'PercentSalaryHike', 'TotalWorkingYears', 'YearsAtCompany',
'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']

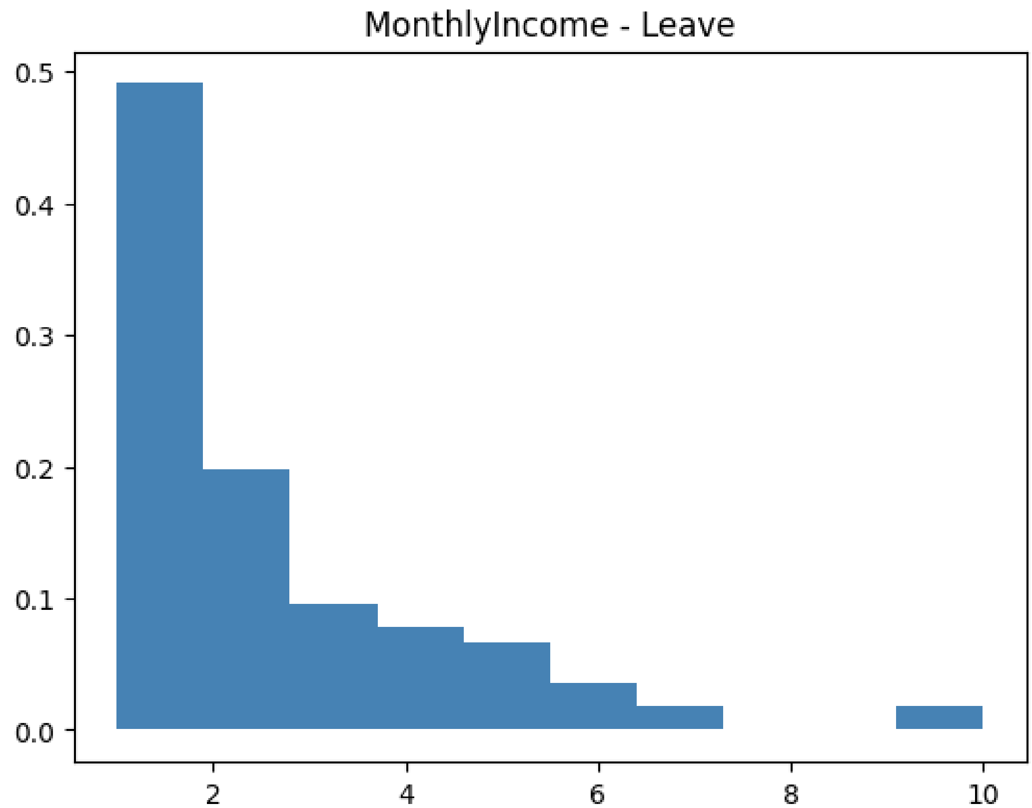
1.0.10 T8. What kind of distribution should we use to model histograms? (Answer a distribution name) What is the MLE for the likelihood distribution? (Describe how to do the MLE). Plot the likelihood distributions of MonthlyIncome, JobRole, HourlyRate, and MaritalStatus for different Attrition values.

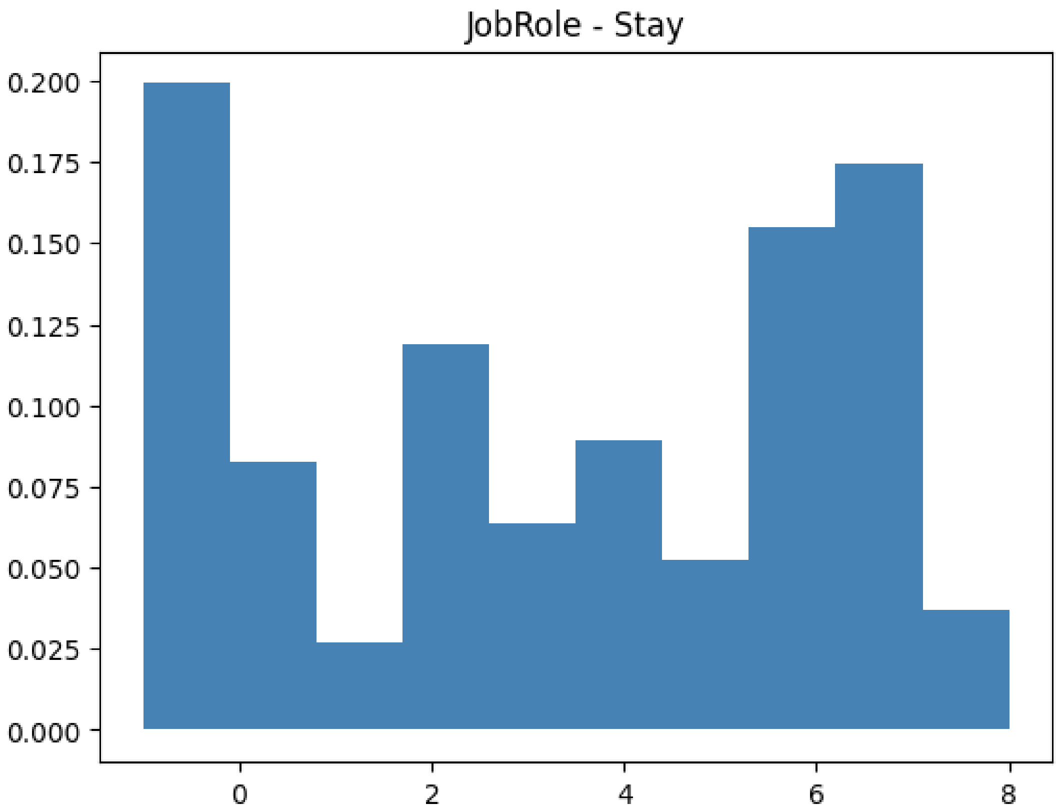
```
Multinomial distribution. As we mapped the value into different bins,
each bin can be treated as one category. The MLE for the probability
of each bin = number of data in that bin / number of all data
```

[13]:
```
def plot_likelihood(x_train, y_train, col, n_bin=10):
    nonan_stay = x_train[y_train == 0][col].dropna()
    nonan_leave = x_train[y_train == 1][col].dropna()

    hist, bin_edges = np.histogram(nonan_stay)
    hist = hist / nonan_stay.shape[0]
    # plot the histogram
```

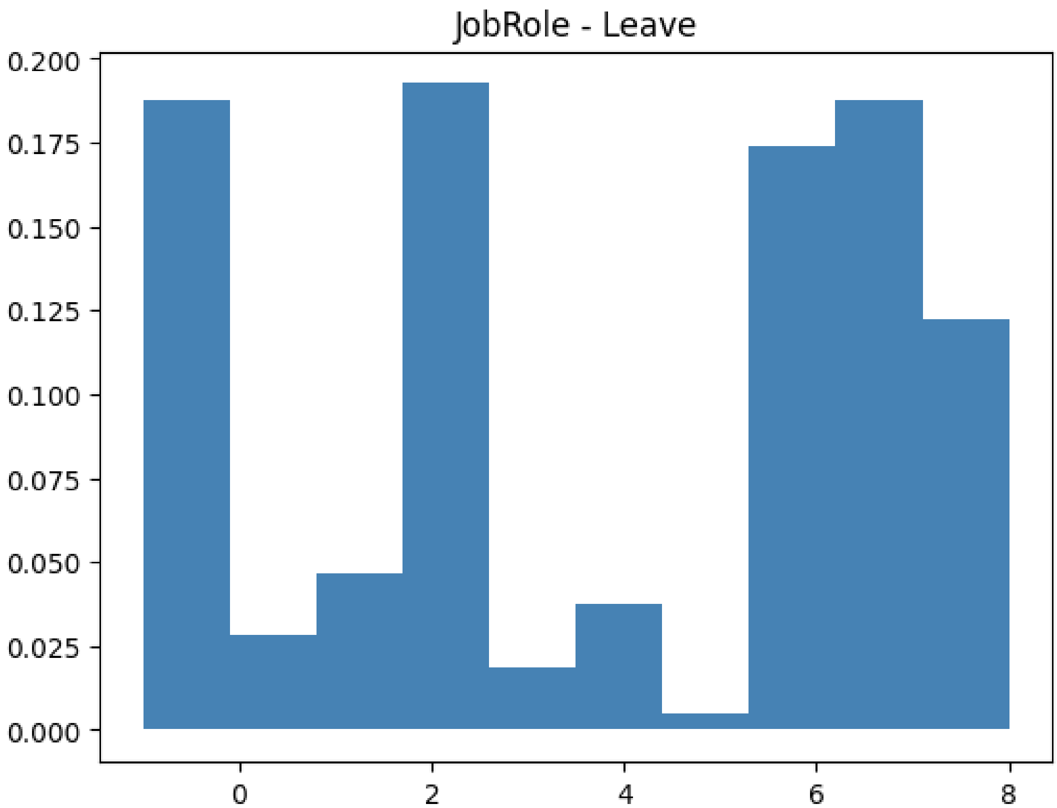
```python
    plt.fill_between(bin_edges.repeat(2)[1:-1], hist.repeat(2),␣
↪facecolor='steelblue')
    plt.title(f"{col} - Stay")
    plt.show()

    hist, bin_edges = np.histogram(nonan_leave)
    hist = hist / nonan_leave.shape[0]
    # plot the histogram
    plt.fill_between(bin_edges.repeat(2)[1:-1], hist.repeat(2),␣
↪facecolor='steelblue')
    plt.title(f"{col} - Leave")
    plt.show()

cols = ["MonthlyIncome", "JobRole", "HourlyRate", "MaritalStatus"]

for col in cols:
    plot_likelihood(x_train, y_train, col)
```

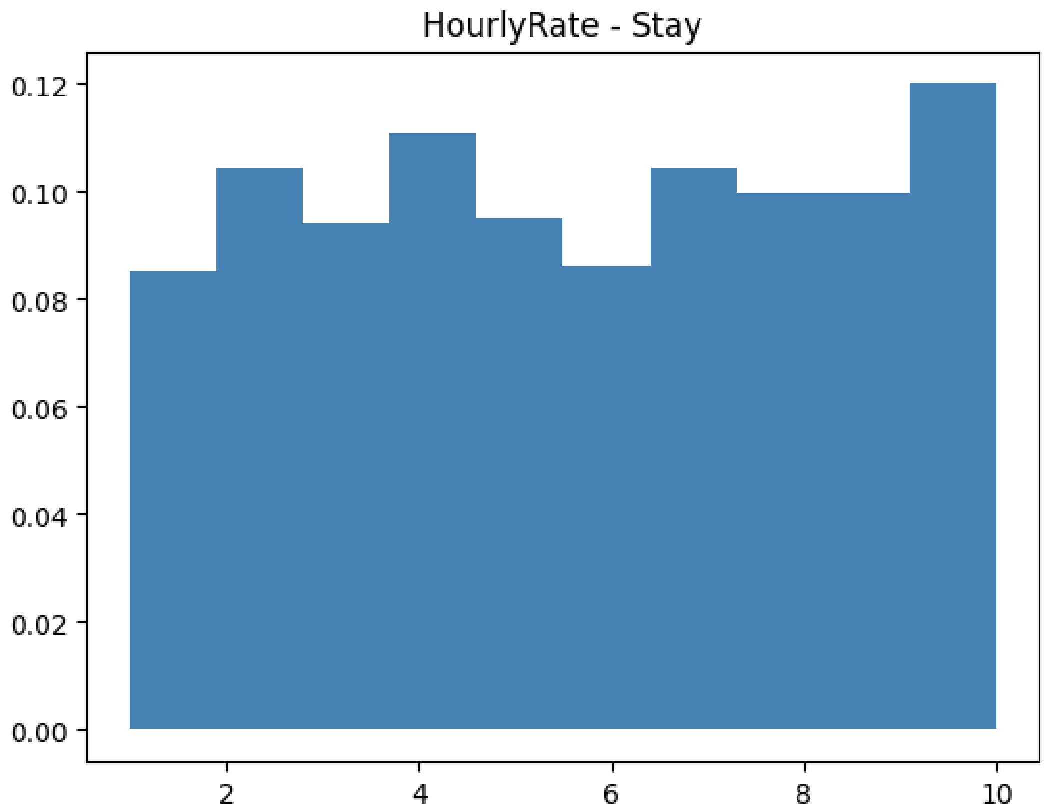


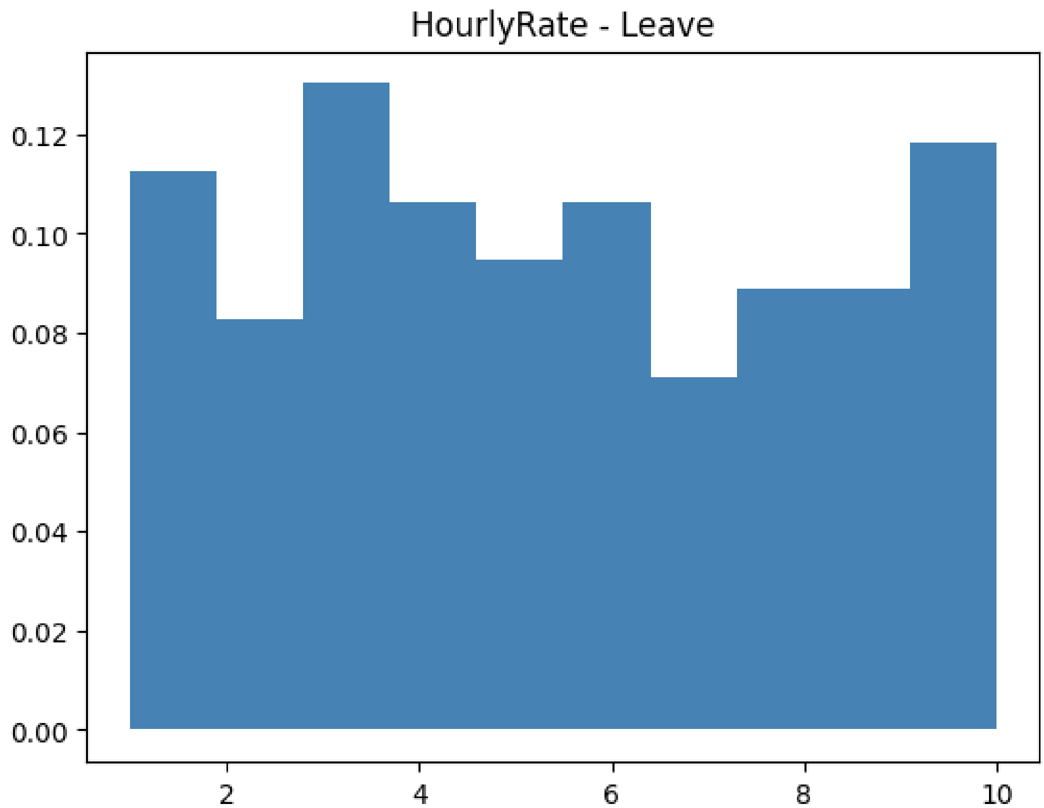

MonthlyIncome - Leave
0.5
0.4
0.3
0.2
0.1
0.0
2
4
6
8
10

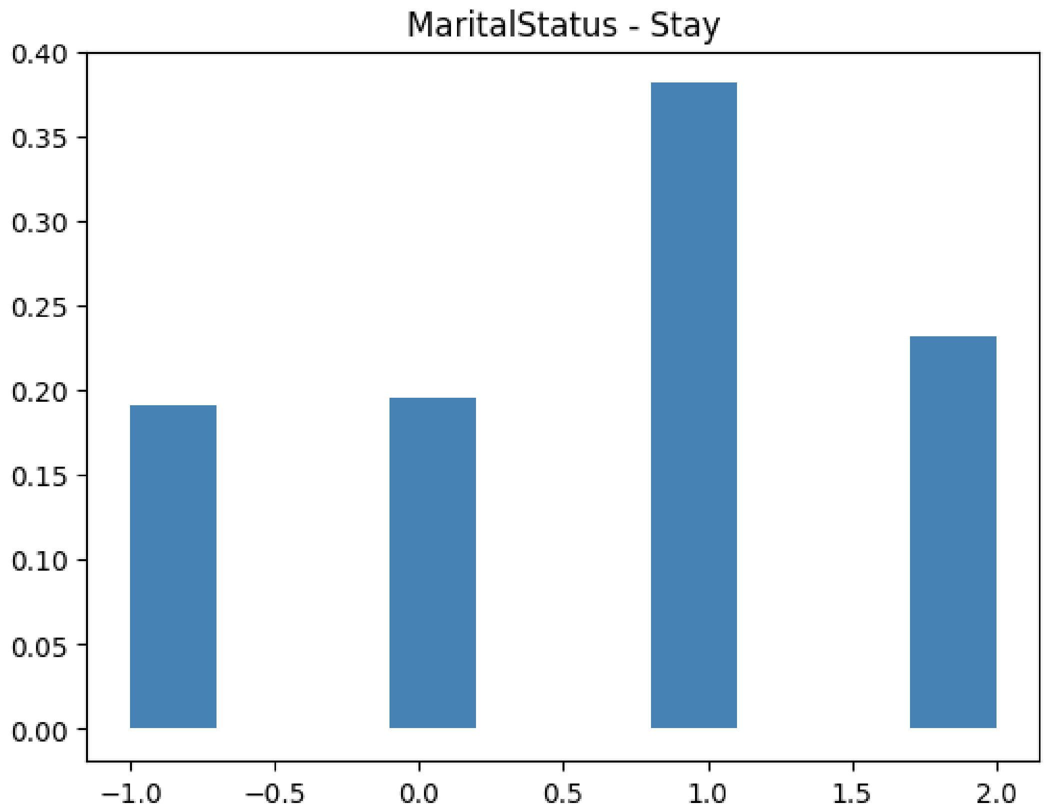


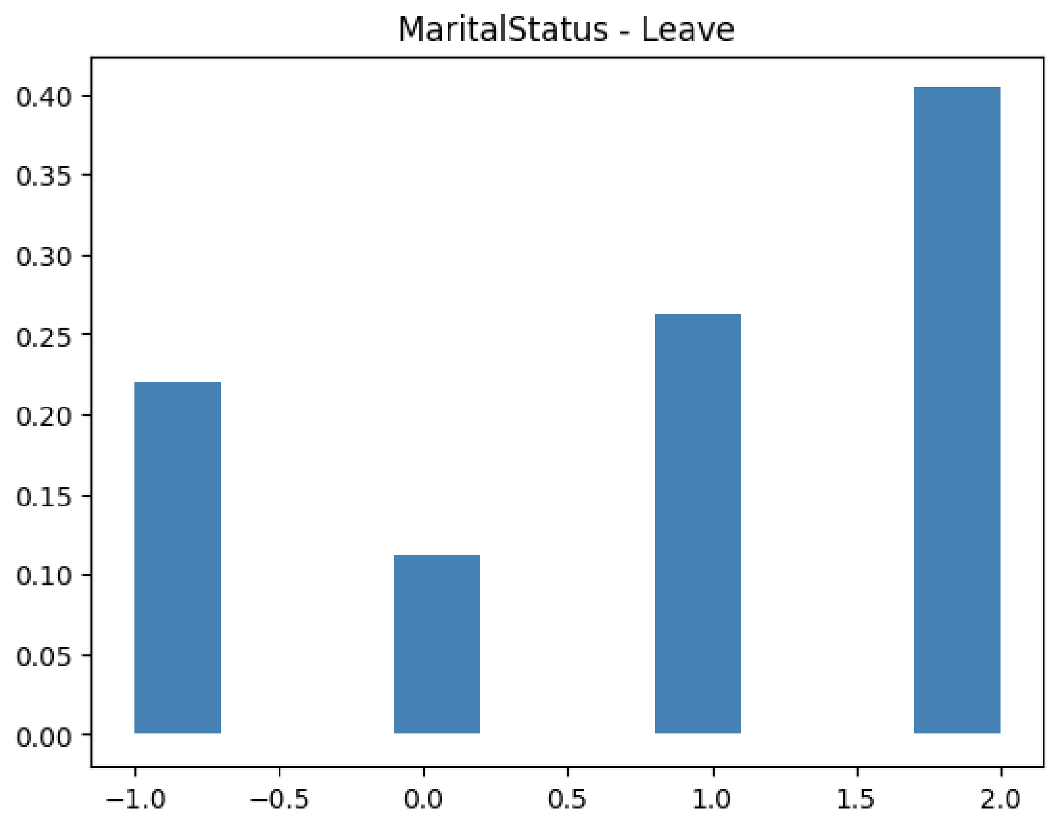
JobRole - Stay
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
2
4
6
8


JobRole - Leave
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
2
4
6
8


HourlyRate - Stay
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2
4
6
8
10


HourlyRate - Leave
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2
4
6
8
10


MaritalStatus - Stay
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0



1.0.11 T9. What is the prior distribution of the two classes?

[14]:
```
stay = np.sum(y_train == 0)  / y_train.shape[0]
leave = np.sum(y_train == 1)  / y_train.shape[0]

print("Stay:", stay)
print("Leave:", leave)
```

```
Stay: 0.8390022675736961
Leave: 0.16099773242630386
```

1.0.12 T10. If we use the current Naive Bayes with our current Maximum Likelihood Estimates, we will find that some P (x i |attrition) will be zero and will result in the entire product term to be zero. Propose a method to fix this problem.

```
Inserting some small values (epsilon) would help preventing the probability to be zero.
```

1.0.13 T11. Implement your Naive Bayes classifier. Use the learned distributions to classify the test set. Don't forget to allow your classifier to handle missing values in the test set. Report the overall Accuracy. Then, report the Precision, Recall, and F score for detecting attrition. See Lecture 1 for the definitions of each metric.

[20]:
```
import importlib, SimpleBayesClassifier
importlib.reload(SimpleBayesClassifier)
from SimpleBayesClassifier import SimpleBayesClassifier
```

[21]:
```
model = SimpleBayesClassifier(n_pos = np.sum(y_train == 1), n_neg = np.
↪sum(y_train == 0))
```

[23]:
```
def check_prior():
    """
    This function designed to test the implementation of the prior probability␣
↪calculation in a Naive Bayes classifier.
    Specifically, it checks if the classifier correctly computes the prior␣
↪probabilities for the
    negative and positive classes based on given input counts.
    """

    # prior_neg = 5/(5 + 5) = 0.5 and # prior_pos = 5/(5 + 5) = 0.5
    assert (SimpleBayesClassifier(5, 5).prior_pos, SimpleBayesClassifier(5, 5).
↪prior_neg) == (0.5, 0.5)

    assert (SimpleBayesClassifier(3, 5).prior_pos, SimpleBayesClassifier(3, 5).
↪prior_neg) == (0.375, 0.625)
    assert (SimpleBayesClassifier(0, 1).prior_pos, SimpleBayesClassifier(0, 1).
↪prior_neg) == (0, 1)
    assert (SimpleBayesClassifier(1, 0).prior_pos, SimpleBayesClassifier(1, 0).
↪prior_neg) == (1, 0)

check_prior()
```

[25]:
```
model.fit_params(np.array(x_train), np.array(y_train))
```

[25]:
```
([(array([0.02581369, 0.05499439, 0.14253648, 0.17171717, 0.18181818,
          0.16498316, 0.09876543, 0.07182941, 0.05387205, 0.03367003]),
   array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
  (array([0.2027027 , 0.        , 0.        , 0.08558559, 0.        ,
          0.        , 0.13153153, 0.        , 0.        , 0.58018018]),
   array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
  (array([0.10961969, 0.08053691, 0.098434  , 0.11297539, 0.08501119,
          0.09619687, 0.0950783 , 0.11409396, 0.10738255, 0.10067114]),
   array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
  (array([0.19099099, 0.        , 0.        , 0.03603604, 0.        ,
```

```
        0.        , 0.54594595, 0.        , 0.        , 0.22702703]),
 array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
(array([0.34698521, 0.13083049, 0.17406143, 0.08987486, 0.02275313,
        0.04778157, 0.05119454, 0.04095563, 0.05005688, 0.04550626]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.10946408, 0.        , 0.19156214, 0.        , 0.        ,
        0.38312429, 0.        , 0.28164196, 0.        , 0.03420753]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.19459459, 0.01531532, 0.        , 0.33693694, 0.        ,
        0.07567568, 0.26666667, 0.        , 0.04234234, 0.06846847]),
 array([-inf, -0.4,  0.2,  0.8,  1.4,  2. ,  2.6,  3.2,  3.8,  4.4,  inf])),
(array([0.18423973, 0.        , 0.        , 0.18201998, 0.        ,
        0.        , 0.32741398, 0.        , 0.        , 0.3063263 ]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.2009009 , 0.        , 0.        , 0.        , 0.        ,
        0.33873874, 0.        , 0.        , 0.        , 0.46036036]),
 array([-inf, -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  inf])),
(array([0.08520179, 0.10426009, 0.0941704 , 0.11098655, 0.09529148,
        0.08632287, 0.10426009, 0.09977578, 0.09977578, 0.11995516]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.0407701 , 0.        , 0.        , 0.26274066, 0.        ,
        0.        , 0.59116648, 0.        , 0.        , 0.10532276]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.31513083, 0.        , 0.37997725, 0.        , 0.        ,
        0.16382253, 0.        , 0.09215017, 0.        , 0.04891923]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.1990991 , 0.08288288, 0.02702703, 0.11891892, 0.06396396,
        0.08918919, 0.05225225, 0.15495495, 0.17477477, 0.03693694]),
 array([-inf, -0.1,  0.8,  1.7,  2.6,  3.5,  4.4,  5.3,  6.2,  7.1,  inf])),
(array([0.18459796, 0.        , 0.        , 0.19705549, 0.        ,
        0.        , 0.29331823, 0.        , 0.        , 0.32502831]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.19099099, 0.        , 0.        , 0.1954955 , 0.        ,
        0.        , 0.38108108, 0.        , 0.        , 0.23243243]),
 array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
(array([0.2073991 , 0.24103139, 0.20627803, 0.08295964, 0.06165919,
        0.05605381, 0.03699552, 0.01345291, 0.04372197, 0.05044843]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.09775281, 0.11460674, 0.10449438, 0.09662921, 0.0988764 ,
        0.09438202, 0.09438202, 0.10674157, 0.11348315, 0.07865169]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.13718821, 0.33446712, 0.10544218, 0.12131519, 0.1031746 ,
        0.03741497, 0.04421769, 0.04875283, 0.03061224, 0.03741497]),
 array([-inf,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,  8.1,  inf])),
(array([0.19189189, 0.        , 0.        , 0.        , 0.        ,
        0.62162162, 0.        , 0.        , 0.        , 0.18648649]),
 array([-inf, -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  inf])),
```

```
 (array([0.26131222, 0.14253394, 0.21266968, 0.05769231, 0.05656109,
        0.11312217, 0.03846154, 0.07126697, 0.02036199, 0.0260181 ]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
 (array([0.85310734, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.14689266]),
  array([-inf,  3.1,  3.2,  3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  inf])),
 (array([0.1868743 , 0.        , 0.        , 0.20244716, 0.        ,
        0.        , 0.30700779, 0.        , 0.        , 0.30367075]),
  array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
 (array([0.39595051, 0.        , 0.        , 0.44769404, 0.        ,
        0.        , 0.10686164, 0.        , 0.        , 0.04949381]),
  array([-inf,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8,  2.1,  2.4,  2.7,  inf])),
 (array([0.08017817, 0.23273942, 0.31514477, 0.11024499, 0.08240535,
        0.08017817, 0.03452116, 0.03452116, 0.02227171, 0.0077951 ]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
 (array([0.03703704, 0.05274972, 0.        , 0.36026936, 0.        ,
        0.332211  , 0.07856341, 0.        , 0.09539843, 0.04377104]),
  array([-inf,  0.6,  1.2,  1.8,  2.4,  3. ,  3.6,  4.2,  4.8,  5.4,  inf])),
 (array([0.04519774, 0.        , 0.        , 0.23050847, 0.        ,
        0.        , 0.62146893, 0.        , 0.        , 0.10282486]),
  array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
 (array([0.27160494, 0.34006734, 0.20089787, 0.06734007, 0.03142536,
        0.04938272, 0.02132435, 0.00448934, 0.00561167, 0.00785634]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
 (array([0.17502787, 0.34225195, 0.09587514, 0.18617614, 0.07246377,
        0.07134894, 0.02564103, 0.01672241, 0.01003344, 0.00445931]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
 (array([0.62107623, 0.11210762, 0.07959641, 0.03363229, 0.07847534,
        0.01457399, 0.01345291, 0.02017937, 0.01345291, 0.01345291]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
 (array([0.18459796, 0.34994337, 0.08720272, 0.01812005, 0.2400906 ,
        0.06568516, 0.01585504, 0.02491506, 0.00792752, 0.00566251]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf]))],
[(array([0.13294798, 0.11560694, 0.16184971, 0.20231214, 0.10982659,
        0.07514451, 0.06358382, 0.05202312, 0.04046243, 0.04624277]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
 (array([0.19248826, 0.        , 0.        , 0.03286385, 0.        ,
        0.        , 0.23004695, 0.        , 0.        , 0.54460094]),
  array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
 (array([0.08917197, 0.14649682, 0.08917197, 0.11464968, 0.08917197,
        0.10828025, 0.07006369, 0.12101911, 0.07643312, 0.0955414 ]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
 (array([0.22535211, 0.        , 0.        , 0.05164319, 0.        ,
        0.        , 0.41314554, 0.        , 0.        , 0.30985915]),
  array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
 (array([0.25988701, 0.11864407, 0.14124294, 0.0960452 , 0.03389831,
        0.07909605, 0.05084746, 0.06214689, 0.10734463, 0.05084746]),
```

```
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.13068182, 0.        , 0.19886364, 0.        , 0.        ,
        0.41477273, 0.        , 0.22727273, 0.        , 0.02840909]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.18309859, 0.02816901, 0.        , 0.29577465, 0.        ,
        0.12676056, 0.22535211, 0.        , 0.03755869, 0.10328638]),
 array([-inf, -0.4,  0.2,  0.8,  1.4,  2. ,  2.6,  3.2,  3.8,  4.4,  inf])),
(array([0.26875, 0.     , 0.     , 0.2    , 0.     , 0.     , 0.2375 ,
        0.     , 0.     , 0.29375]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.19248826, 0.        , 0.        , 0.        , 0.        ,
        0.29577465, 0.        , 0.        , 0.        , 0.51173709]),
 array([-inf, -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  inf])),
(array([0.11242604, 0.08284024, 0.13017751, 0.10650888, 0.09467456,
        0.10650888, 0.07100592, 0.0887574 , 0.0887574 , 0.1183432 ]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.1091954 , 0.        , 0.        , 0.31034483, 0.        ,
        0.        , 0.52873563, 0.        , 0.        , 0.05172414]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.60795455, 0.        , 0.23295455, 0.        , 0.        ,
        0.11931818, 0.        , 0.01704545, 0.        , 0.02272727]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.18779343, 0.02816901, 0.04694836, 0.19248826, 0.01877934,
        0.03755869, 0.00469484, 0.17370892, 0.18779343, 0.12206573]),
 array([-inf, -0.1,  0.8,  1.7,  2.6,  3.5,  4.4,  5.3,  6.2,  7.1,  inf])),
(array([0.24      , 0.        , 0.        , 0.2       , 0.        ,
        0.        , 0.30857143, 0.        , 0.        , 0.25142857]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.22065728, 0.        , 0.        , 0.11267606, 0.        ,
        0.        , 0.2629108 , 0.        , 0.        , 0.40375587]),
 array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
(array([0.49101796, 0.19760479, 0.09580838, 0.07784431, 0.06586826,
        0.03592814, 0.01796407, 0.        , 0.        , 0.01796407]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.13333333, 0.06666667, 0.08484848, 0.13939394, 0.08484848,
        0.09090909, 0.11515152, 0.10909091, 0.08484848, 0.09090909]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.0960452 , 0.38983051, 0.07909605, 0.06779661, 0.08474576,
        0.06779661, 0.05649718, 0.06779661, 0.02824859, 0.06214689]),
 array([-inf,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,  8.1,  inf])),
(array([0.23474178, 0.        , 0.        , 0.        , 0.        ,
        0.36150235, 0.        , 0.        , 0.        , 0.40375587]),
 array([-inf, -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  inf])),
(array([0.33519553, 0.15642458, 0.15642458, 0.05586592, 0.06145251,
        0.08379888, 0.03351955, 0.05027933, 0.02793296, 0.03910615]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.83529412, 0.        , 0.        , 0.        , 0.        ,
```

```
        0.        , 0.        , 0.        , 0.        , 0.16470588]),
 array([-inf,  3.1,  3.2,  3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  inf])),
(array([0.26219512, 0.        , 0.        , 0.20121951, 0.        ,
        0.        , 0.2804878 , 0.        , 0.        , 0.25609756]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.65116279, 0.        , 0.        , 0.22674419, 0.        ,
        0.        , 0.06395349, 0.        , 0.        , 0.05813953]),
 array([-inf,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8,  2.1,  2.4,  2.7,  inf])),
(array([0.25454545, 0.2969697 , 0.22424242, 0.07878788, 0.06060606,
        0.03636364, 0.01818182, 0.01212121, 0.00606061, 0.01212121]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.05202312, 0.05202312, 0.        , 0.42196532, 0.        ,
        0.28901734, 0.10404624, 0.        , 0.06358382, 0.01734104]),
 array([-inf,  0.6,  1.2,  1.8,  2.4,  3. ,  3.6,  4.2,  4.8,  5.4,  inf])),
(array([0.10588235, 0.        , 0.        , 0.23529412, 0.        ,
        0.        , 0.52352941, 0.        , 0.        , 0.13529412]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.48837209, 0.23837209, 0.18023256, 0.03488372, 0.01744186,
        0.01744186, 0.01162791, 0.        , 0.00581395, 0.00581395]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.31137725, 0.37125749, 0.05389222, 0.1497006 , 0.        ,
        0.04191617, 0.04790419, 0.00598802, 0.01197605, 0.00598802]),
 array([-inf,  1.8,  2.6,  3.4,  4.2,  5. ,  5.8,  6.6,  7.4,  8.2,  inf])),
(array([0.66470588, 0.11764706, 0.05882353, 0.01176471, 0.09411765,
        0.        , 0.01764706, 0.01176471, 0.01176471, 0.01176471]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.40462428, 0.28901734, 0.04624277, 0.01734104, 0.        ,
        0.17919075, 0.04624277, 0.00578035, 0.        , 0.01156069]),
 array([-inf,  1.8,  2.6,  3.4,  4.2,  5. ,  5.8,  6.6,  7.4,  8.2,  inf]))])
```

[26]:

```
def check_fit_params():

    """
    This function is designed to test the fit_params method of a␣
↪SimpleBayesClassifier.
    This method is presumably responsible for computing parameters for a Naive␣
↪Bayes classifier
    based on the provided training data. The parameters in this context is bins␣
↪and edges from each histogram.
    """

    T = SimpleBayesClassifier(2, 2)
    X_TRAIN_CASE_1 = np.array([
        [0, 1, 2, 3],
        [1, 2, 3, 4],
        [2, 3, 4, 5],
        [3, 4, 5, 6]
```

```python
    ])
    Y_TRAIN_CASE_1 = np.array([0, 1, 0, 1])
    STAY_PARAMS_1, LEAVE_PARAMS_1 = T.fit_params(X_TRAIN_CASE_1, Y_TRAIN_CASE_1)

    print("STAY PARAMETERS")
    for f_idx in range(len(STAY_PARAMS_1)):
        print(f"Feature : {f_idx}")
        print(f"BINS : {STAY_PARAMS_1[f_idx][0]}")
        print(f"EDGES : {STAY_PARAMS_1[f_idx][1]}")
    print("")
    print("LEAVE PARAMETERS")
    for f_idx in range(len(STAY_PARAMS_1)):
        print(f"Feature : {f_idx}")
        print(f"BINS : {LEAVE_PARAMS_1[f_idx][0]}")
        print(f"EDGES : {LEAVE_PARAMS_1[f_idx][1]}")

check_fit_params()
```

```
STAY PARAMETERS
Feature : 0
BINS : [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
EDGES : [-inf  0.2  0.4  0.6  0.8  1.   1.2  1.4  1.6  1.8  inf]
Feature : 1
BINS : [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
EDGES : [-inf  1.2  1.4  1.6  1.8  2.   2.2  2.4  2.6  2.8  inf]
Feature : 2
BINS : [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
EDGES : [-inf  2.2  2.4  2.6  2.8  3.   3.2  3.4  3.6  3.8  inf]
Feature : 3
BINS : [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
EDGES : [-inf  3.2  3.4  3.6  3.8  4.   4.2  4.4  4.6  4.8  inf]

LEAVE PARAMETERS
Feature : 0
BINS : [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
EDGES : [-inf  1.2  1.4  1.6  1.8  2.   2.2  2.4  2.6  2.8  inf]
Feature : 1
BINS : [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
EDGES : [-inf  2.2  2.4  2.6  2.8  3.   3.2  3.4  3.6  3.8  inf]
Feature : 2
BINS : [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
EDGES : [-inf  3.2  3.4  3.6  3.8  4.   4.2  4.4  4.6  4.8  inf]
Feature : 3
BINS : [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
EDGES : [-inf  4.2  4.4  4.6  4.8  5.   5.2  5.4  5.6  5.8  inf]
```

[27]:
```
x_test_np = np.array(x_test)
y_pred = np.array(model.predict(x = x_test_np))
```

[28]:
```
y_pred.shape
```

[28]: (147,)

[29]:
```
def evaluate(y_true, y_pred, show_result = True):
    if (y_true.shape[0] != y_pred.shape[0]):
        return -1, -1, -1, -1

    tp = np.sum((y_pred == 1) & (y_true == 1))
    fn = np.sum((y_pred == 0) & (y_true == 1))
    fp = np.sum((y_pred == 1) & (y_true == 0))
    tn = np.sum((y_pred == 0) & (y_true == 0))

    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = (tp) / (tp + fp)
    recall = (tp) / (tp + fn)
    F1 = 2 / (1 / precision + 1 / recall)
    fpr = (fp) / (fp + tn)

    if (show_result):
        print(f"Accuracy: {accuracy * 100}%\nPrecision: {precision}\nRecall: 
 ↪{recall}\nF1: {F1}\nFPR: {fpr}\n")

    return accuracy, precision, recall, F1, fpr
```

[30]:
```
evaluate(np.array(y_test), y_pred)
```

```
Accuracy: 80.95238095238095%
Precision: 0.375
Recall: 0.25
F1: 0.30000000000000004
FPR: 0.08130081300813008
```

[30]:
```
(np.float64(0.8095238095238095),
 np.float64(0.375),
 np.float64(0.25),
 np.float64(0.30000000000000004),
 np.float64(0.08130081300813008))
```

1.0.14 T12. Use the learned distributions to classify the test set. Report the results using the same metric as the previous question.

[32]:
```
df = pd.read_csv('hr-employee-attrition-with-null.csv')

df.loc[df["Attrition"] == "no", "Attrition"] = 0.0
df.loc[df["Attrition"] == "yes", "Attrition"] = 1.0
string_categorical_col = ['Department', 'Attrition', 'BusinessTravel',␣
↪'EducationField', 'Gender', 'JobRole',
                          'MaritalStatus', 'Over18', 'OverTime']

# ENCODE STRING COLUMNS TO CATEGORICAL COLUMNS
for col in string_categorical_col:
    # INSERT CODE HERE
    df[col] = pd.Categorical(df[col]).codes
# HANDLE NULL NUMBERS
# I don't think we need to handle null?

# INSERT CODE HERE
df = df.loc[:, ~df.columns.isin(['EmployeeNumber', 'Unnamed: 0',␣
↪'EmployeeCount', 'StandardHours', 'Over18'])] # drop these columns

X = df.drop(["Attrition"], axis=1)
Y = df["Attrition"]

x_train, x_test, y_train, y_test = train_test_split(X, Y, stratify=Y,␣
↪test_size=0.1, random_state=12345)
```

[34]:
```
model.fit_gaussian_params(np.array(x_train), np.array(y_train))
```

[34]:
```
([(np.float64(37.809203142536475), np.float64(8.896433952739484)),
  (np.float64(1.0891891891891892), np.float64(1.2118670987811953)),
  (np.float64(808.728187919463), np.float64(407.2251752337319)),
  (np.float64(0.809009009009009), np.float64(0.9952635170583247)),
  (np.float64(9.0773606370876), np.float64(8.112178479155704)),
  (np.float64(2.9395667046750287), np.float64(1.0218798132530393)),
  (np.float64(1.6054054054054054), np.float64(1.7453071098573252)),
  (np.float64(2.755826859045505), np.float64(1.0796687888712027)),
  (np.float64(0.2594594594594595), np.float64(0.7706763588940992)),
  (np.float64(65.53699551569507), np.float64(20.404589827789692)),
  (np.float64(2.7610419026047563), np.float64(0.6885077018805345)),
  (np.float64(2.179749715585893), np.float64(1.120498601502818)),
  (np.float64(3.324324324324324), np.float64(3.076684076050504)),
  (np.float64(2.7587768969422424), np.float64(1.0964885160093172)),
  (np.float64(0.6549549549549549), np.float64(1.0357778761789396)),
  (np.float64(6847.662556053811), np.float64(4798.092360725341)),
  (np.float64(14266.125842696629), np.float64(7124.572193733051)),
```

```
 (np.float64(2.697278911564626), np.float64(2.475772279106189)),
 (np.float64(-0.005405405405405406), np.float64(0.6151009347828872)),
 (np.float64(15.266968325791856), np.float64(3.6403649988412696)),
 (np.float64(3.146892655367232), np.float64(0.35399887452701784)),
 (np.float64(2.727474972191324), np.float64(1.0859766282649876)),
 (np.float64(0.8098987626546682), np.float64(0.8152600328328238)),
 (np.float64(11.946547884187082), np.float64(7.805074751024999)),
 (np.float64(2.823793490460157), np.float64(1.314400760052736)),
 (np.float64(2.781920903954802), np.float64(0.6830562214440479)),
 (np.float64(7.47250280583614), np.float64(6.18731880041615)),
 (np.float64(4.498327759197324), np.float64(3.630701412165684)),
 (np.float64(2.280269058295964), np.float64(3.22057788303618)),
 (np.float64(4.394110985277464), np.float64(3.552047784335249))],
[(np.float64(33.947976878612714), np.float64(10.265417501739252)),
 (np.float64(1.1267605633802817), np.float64(1.1538412368896775)),
 (np.float64(769.3248407643312), np.float64(399.78765409130403)),
 (np.float64(0.8075117370892019), np.float64(1.1071851998485391)),
 (np.float64(11.344632768361581), np.float64(8.707415308937323)),
 (np.float64(2.8238636363636362), np.float64(1.0156160600369408)),
 (np.float64(1.7089201877934272), np.float64(1.8101864241960348)),
 (np.float64(2.55625), np.float64(1.171254002127634)),
 (np.float64(0.3192488262910798), np.float64(0.7760834613779912)),
 (np.float64(63.875739644970416), np.float64(20.622901977931885)),
 (np.float64(2.5229885057471266), np.float64(0.755850917220641)),
 (np.float64(1.6136363636363635), np.float64(0.9223465403501375)),
 (np.float64(3.807511737089202), np.float64(3.2653451249559375)),
 (np.float64(2.5714285714285716), np.float64(1.1080411102665897)),
 (np.float64(0.8497652582159625), np.float64(1.1732393426854273)),
 (np.float64(4590.413173652694), np.float64(3511.341093996613)),
 (np.float64(14242.339393939394), np.float64(7085.770015252696)),
 (np.float64(3.0282485875706215), np.float64(2.7193165029633763)),
 (np.float64(0.16901408450704225), np.float64(0.7809813645794668)),
 (np.float64(14.966480446927374), np.float64(3.8168978034257677)),
 (np.float64(3.164705882352941), np.float64(0.3709148887161046)),
 (np.float64(2.5304878048780486), np.float64(1.133867655243493)),
 (np.float64(0.5290697674418605), np.float64(0.8519971447530724)),
 (np.float64(8.587878787878788), np.float64(7.430476047221869)),
 (np.float64(2.601156069364162), np.float64(1.2056700551978288)),
 (np.float64(2.6882352941176473), np.float64(0.8348175939838075)),
 (np.float64(5.325581395348837), np.float64(5.541475457256379)),
 (np.float64(3.2095808383233533), np.float64(3.2991240632602064)),
 (np.float64(1.9529411764705882), np.float64(3.084709471887354)),
 (np.float64(2.9653179190751446), np.float64(3.2769927595779498))])
```

[35]:

```
def check_fit_gaussian_params():

    """
```

```
    This function is designed to test the fit_gaussian_params method of a␣
↪SimpleBayesClassifier.
    This method is presumably responsible for computing parameters for a Naive␣
↪Bayes classifier
    based on the provided training data. The parameters in this context is mean␣
↪and STD.
    """

    T = SimpleBayesClassifier(2, 2)
    X_TRAIN_CASE_1 = np.array([
        [0, 1, 2, 3],
        [1, 2, 3, 4],
        [2, 3, 4, 5],
        [3, 4, 5, 6]
    ])
    Y_TRAIN_CASE_1 = np.array([0, 1, 0, 1])
    STAY_PARAMS_1, LEAVE_PARAMS_1 = T.fit_gaussian_params(X_TRAIN_CASE_1,␣
↪Y_TRAIN_CASE_1)

    print("STAY PARAMETERS")
    for f_idx in range(len(STAY_PARAMS_1)):
        print(f"Feature : {f_idx}")
        print(f"Mean : {STAY_PARAMS_1[f_idx][0]}")
        print(f"STD. : {STAY_PARAMS_1[f_idx][1]}")
    print("")
    print("LEAVE PARAMETERS")
    for f_idx in range(len(STAY_PARAMS_1)):
        print(f"Feature : {f_idx}")
        print(f"Mean : {LEAVE_PARAMS_1[f_idx][0]}")
        print(f"STD. : {LEAVE_PARAMS_1[f_idx][1]}")

check_fit_gaussian_params()
```

```
STAY PARAMETERS
Feature : 0
Mean : 1.0
STD. : 1.0
Feature : 1
Mean : 2.0
STD. : 1.0
Feature : 2
Mean : 3.0
STD. : 1.0
Feature : 3
Mean : 4.0
STD. : 1.0
```

```
LEAVE PARAMETERS
Feature : 0
Mean : 2.0
STD. : 1.0
Feature : 1
Mean : 3.0
STD. : 1.0
Feature : 2
Mean : 4.0
STD. : 1.0
Feature : 3
Mean : 5.0
STD. : 1.0
```

[36]:
```
y_pred = model.gaussian_predict(np.array(x_test))
```

[37]:
```
evaluate(y_test, y_pred)
```

```
Accuracy: 81.63265306122449%
Precision: 0.45161290322580644
Recall: 0.5833333333333334
F1: 0.509090909090909
FPR: 0.13821138211382114
```

[37]:
```
(np.float64(0.8163265306122449),
 np.float64(0.45161290322580644),
 np.float64(0.5833333333333334),
 np.float64(0.509090909090909),
 np.float64(0.13821138211382114))
```

1.0.15 T13 : The random choice baseline is the accuracy if you make a random guess for each test sample. Give random guess (50% leaving, and 50% staying) to the test samples. Report the overall Accuracy. Then, report the Precision, Recall, and F score for attrition prediction using the random choice baseline.

[38]:
```
y_random_pred = np.random.default_rng(seed=12345).random(y_test.shape)
```

[39]:
```
y_random_pred
```

[39]:
```
array([0.22733602, 0.31675834, 0.79736546, 0.67625467, 0.39110955,
       0.33281393, 0.59830875, 0.18673419, 0.67275604, 0.94180287,
       0.24824571, 0.94888115, 0.66723745, 0.09589794, 0.44183967,
       0.88647992, 0.6974535 , 0.32647286, 0.73392816, 0.22013496,
       0.08159457, 0.1598956 , 0.34010018, 0.46519315, 0.26642103,
       0.8157764 , 0.19329439, 0.12946908, 0.09166475, 0.59856801,
       0.8547419 , 0.60162124, 0.93198836, 0.72478136, 0.86055132,
       0.9293378 , 0.54618601, 0.93767296, 0.49498794, 0.27377318,
```

```
0.45177871, 0.66503892, 0.33089093, 0.90345401, 0.25707418,
0.33982834, 0.2588534 , 0.35544648, 0.00502233, 0.62860454,
0.28238271, 0.06808769, 0.61682898, 0.17632632, 0.30438839,
0.44088681, 0.15020234, 0.21792886, 0.47433312, 0.47636886,
0.25523235, 0.29756527, 0.27906712, 0.26057921, 0.48276159,
0.21197904, 0.4956306 , 0.24626133, 0.83848265, 0.18013059,
0.86215629, 0.17829944, 0.75053133, 0.6111204 , 0.20915503,
0.75987242, 0.24926057, 0.08557173, 0.61805672, 0.53696833,
0.63452671, 0.17437411, 0.24816449, 0.68482298, 0.08087165,
0.8750736 , 0.42869438, 0.6183942 , 0.3131055 , 0.17896286,
0.00971213, 0.21004296, 0.87000068, 0.9728298 , 0.44179234,
0.37874949, 0.27594708, 0.96610411, 0.05820261, 0.4087339 ,
0.16862884, 0.24014406, 0.78000786, 0.2037676 , 0.55205095,
0.36699414, 0.50728172, 0.3334378 , 0.28272167, 0.2818303 ,
0.08538129, 0.48181366, 0.88334289, 0.94722777, 0.02738372,
0.91775224, 0.12152453, 0.74784776, 0.89652074, 0.1679298 ,
0.33146322, 0.37815663, 0.34684896, 0.5162557 , 0.00899403,
0.4226782 , 0.87765773, 0.08740515, 0.48408482, 0.48122773,
0.78257149, 0.96455958, 0.70709644, 0.27373672, 0.6701133 ,
0.3475348 , 0.76812784, 0.67577142, 0.97753203, 0.86670979,
0.04610801, 0.29032371, 0.8623911 , 0.60084783, 0.34425796,
0.05560258, 0.76287267])
```

[40]:
```
y_random_pred[y_random_pred >= 0.5] = 1
y_random_pred[y_random_pred < 0.5] = 0
```

[41]:
```
y_random_pred
```

[41]:
```
array([0., 0., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 0., 0., 1., 1.,
       0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 1., 0., 1., 1., 0., 1., 0., 0., 1., 1., 1., 0., 0., 1., 0.,
       1., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0.,
       1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 1., 0., 1., 1.,
       0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 1., 1., 0., 1., 0.,
       1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1.])
```

[42]:
```
evaluate(y_test, y_random_pred)
```

```
Accuracy: 54.421768707483%
Precision: 0.13559322033898305
Recall: 0.3333333333333333
F1: 0.1927710843373494
FPR: 0.4146341463414634
```

[42]:
```
(np.float64(0.54421768707483),
 np.float64(0.13559322033898305),
 np.float64(0.3333333333333333),
 np.float64(0.1927710843373494),
 np.float64(0.4146341463414634))
```

1.0.16 T14. The majority rule is the accuracy if you use the most frequent class from the training set as the classification decision. Report the overall Accuracy. Then, report the Precision, Recall, and F score for attrition prediction using the majority rule baseline.

[43]:
```python
print("Leave:", np.sum(y_train == 1))
print("Stay:", np.sum(y_train == 0))
print("Stay (0) is the majority class")
y_major_pred = np.zeros(y_pred.shape)
```

```
Leave: 213
Stay: 1110
Stay (0) is the majority class
```

[44]:
```python
y_major_pred
```

[44]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[45]:
```python
evaluate(y_test, y_major_pred)
```

```
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0


C:\Users\chyut\AppData\Local\Temp\ipykernel_24836\297337688.py:11:
RuntimeWarning: invalid value encountered in scalar divide
  precision = (tp) / (tp + fp)
C:\Users\chyut\AppData\Local\Temp\ipykernel_24836\297337688.py:13:
RuntimeWarning: divide by zero encountered in scalar divide
  F1 = 2 / (1 / precision + 1 / recall)
```

[45]:
```
(np.float64(0.8367346938775511),
 np.float64(nan),
 np.float64(0.0),
 np.float64(nan),
 np.float64(0.0))
```

1.0.17 T15. Compare the two baselines with your Naive Bayes classifier.

```
Mine's accuracy is more than the random baseline, but a little bit less than the stupid baseli
But, the stupid baseline precision is 0 which mine's is better.
```

1.0.18 T16. Use the following threshold values

$ t = np.arange(-5,5,0.05) $ ### find the best accuracy, and F score (and the corresponding thresholds)

[46]:
```
t = np.arange(-5, 5, 0.05)
maxaccuracy = -1
maxf1 = -1
maxthresholdacc = -10
maxthresholdf1 = -10

history = {}
history["Accuracy"] = []
history["Precision"] = []
history["Recall"] = []
history["F1"] = []
history["FPR"] = []

for each in t:
    print("--- Threshold =", each, "---")
    y_pred = model.gaussian_predict(np.array(x_test), thresh=each)

    accuracy, precision, recall, f1, fpr = evaluate(y_test, y_pred)

    history["Accuracy"].append(accuracy)
    history["Precision"].append(precision)
    history["Recall"].append(recall)
    history["F1"].append(f1)
    history["FPR"].append(fpr)

    if (accuracy > maxaccuracy):
        maxaccuracy = accuracy
        maxthresholdacc = each
    if (f1 > maxf1):
        maxf1 = f1
        maxthresholdf1 = each
```

```python
print("Best Accuracy:", maxaccuracy, "with responding threshold:",␣
 ↪maxthresholdacc)
print("Best F1:", maxf1, "with responding threshold:", maxthresholdf1)
```

```
--- Threshold = -5.0 ---
Accuracy: 27.2108843537415%
Precision: 0.1732283464566929
Recall: 0.9166666666666666
F1: 0.2913907284768212
FPR: 0.8536585365853658

--- Threshold = -4.95 ---
Accuracy: 27.2108843537415%
Precision: 0.1732283464566929
Recall: 0.9166666666666666
F1: 0.2913907284768212
FPR: 0.8536585365853658

--- Threshold = -4.9 ---
Accuracy: 27.2108843537415%
Precision: 0.1732283464566929
Recall: 0.9166666666666666
F1: 0.2913907284768212
FPR: 0.8536585365853658

--- Threshold = -4.8500000000000005 ---
Accuracy: 27.2108843537415%
Precision: 0.1732283464566929
Recall: 0.9166666666666666
F1: 0.2913907284768212
FPR: 0.8536585365853658

--- Threshold = -4.800000000000001 ---
Accuracy: 27.2108843537415%
Precision: 0.1732283464566929
Recall: 0.9166666666666666
F1: 0.2913907284768212
FPR: 0.8536585365853658

--- Threshold = -4.750000000000001 ---
Accuracy: 27.2108843537415%
Precision: 0.1732283464566929
Recall: 0.9166666666666666
F1: 0.2913907284768212
FPR: 0.8536585365853658

--- Threshold = -4.700000000000001 ---
Accuracy: 27.2108843537415%
```

```
Precision: 0.1732283464566929
Recall: 0.9166666666666666
F1: 0.2913907284768212
FPR: 0.8536585365853658

--- Threshold = -4.650000000000001 ---
Accuracy: 27.2108843537415%
Precision: 0.1732283464566929
Recall: 0.9166666666666666
F1: 0.2913907284768212
FPR: 0.8536585365853658

--- Threshold = -4.600000000000001 ---
Accuracy: 28.57142857142857%
Precision: 0.176
Recall: 0.9166666666666666
F1: 0.29530201342281875
FPR: 0.8373983739837398

--- Threshold = -4.550000000000002 ---
Accuracy: 28.57142857142857%
Precision: 0.176
Recall: 0.9166666666666666
F1: 0.29530201342281875
FPR: 0.8373983739837398

--- Threshold = -4.500000000000002 ---
Accuracy: 29.25170068027211%
Precision: 0.1774193548387097
Recall: 0.9166666666666666
F1: 0.2972972972972973
FPR: 0.8292682926829268

--- Threshold = -4.450000000000002 ---
Accuracy: 29.931972789115648%
Precision: 0.17886178861788618
Recall: 0.9166666666666666
F1: 0.29931972789115646
FPR: 0.8211382113821138

--- Threshold = -4.400000000000002 ---
Accuracy: 29.931972789115648%
Precision: 0.17886178861788618
Recall: 0.9166666666666666
F1: 0.29931972789115646
FPR: 0.8211382113821138

--- Threshold = -4.350000000000002 ---
```

```
Accuracy: 29.931972789115648%
Precision: 0.17886178861788618
Recall: 0.9166666666666666
F1: 0.29931972789115646
FPR: 0.8211382113821138

--- Threshold = -4.3000000000000025 ---
Accuracy: 29.931972789115648%
Precision: 0.17886178861788618
Recall: 0.9166666666666666
F1: 0.29931972789115646
FPR: 0.8211382113821138

--- Threshold = -4.250000000000003 ---
Accuracy: 30.612244897959183%
Precision: 0.18032786885245902
Recall: 0.9166666666666666
F1: 0.30136986301369867
FPR: 0.8130081300813008

--- Threshold = -4.200000000000003 ---
Accuracy: 30.612244897959183%
Precision: 0.18032786885245902
Recall: 0.9166666666666666
F1: 0.30136986301369867
FPR: 0.8130081300813008

--- Threshold = -4.150000000000003 ---
Accuracy: 30.612244897959183%
Precision: 0.18032786885245902
Recall: 0.9166666666666666
F1: 0.30136986301369867
FPR: 0.8130081300813008

--- Threshold = -4.100000000000003 ---
Accuracy: 30.612244897959183%
Precision: 0.18032786885245902
Recall: 0.9166666666666666
F1: 0.30136986301369867
FPR: 0.8130081300813008

--- Threshold = -4.050000000000003 ---
Accuracy: 30.612244897959183%
Precision: 0.18032786885245902
Recall: 0.9166666666666666
F1: 0.30136986301369867
FPR: 0.8130081300813008
```

```
--- Threshold = -4.0000000000000036 ---
Accuracy: 31.292517006802722%
Precision: 0.18181818181818182
Recall: 0.9166666666666666
F1: 0.30344827586206896
FPR: 0.8048780487804879

--- Threshold = -3.9500000000000037 ---
Accuracy: 31.292517006802722%
Precision: 0.18181818181818182
Recall: 0.9166666666666666
F1: 0.30344827586206896
FPR: 0.8048780487804879

--- Threshold = -3.900000000000004 ---
Accuracy: 31.292517006802722%
Precision: 0.18181818181818182
Recall: 0.9166666666666666
F1: 0.30344827586206896
FPR: 0.8048780487804879

--- Threshold = -3.850000000000004 ---
Accuracy: 31.292517006802722%
Precision: 0.18181818181818182
Recall: 0.9166666666666666
F1: 0.30344827586206896
FPR: 0.8048780487804879

--- Threshold = -3.8000000000000043 ---
Accuracy: 31.97278911564626%
Precision: 0.18333333333333332
Recall: 0.9166666666666666
F1: 0.3055555555555555
FPR: 0.7967479674796748

--- Threshold = -3.7500000000000044 ---
Accuracy: 31.97278911564626%
Precision: 0.18333333333333332
Recall: 0.9166666666666666
F1: 0.3055555555555555
FPR: 0.7967479674796748

--- Threshold = -3.7000000000000046 ---
Accuracy: 32.6530612244898%
Precision: 0.18487394957983194
Recall: 0.9166666666666666
F1: 0.3076923076923077
FPR: 0.7886178861788617
```

```
--- Threshold = -3.650000000000005 ---
Accuracy: 32.6530612244898%
Precision: 0.18487394957983194
Recall: 0.9166666666666666
F1: 0.3076923076923077
FPR: 0.7886178861788617

--- Threshold = -3.600000000000005 ---
Accuracy: 34.01360544217687%
Precision: 0.18803418803418803
Recall: 0.9166666666666666
F1: 0.3120567375886525
FPR: 0.7723577235772358

--- Threshold = -3.550000000000005 ---
Accuracy: 34.01360544217687%
Precision: 0.18803418803418803
Recall: 0.9166666666666666
F1: 0.3120567375886525
FPR: 0.7723577235772358

--- Threshold = -3.5000000000000053 ---
Accuracy: 34.69387755102041%
Precision: 0.1896551724137931
Recall: 0.9166666666666666
F1: 0.3142857142857143
FPR: 0.7642276422764228

--- Threshold = -3.4500000000000055 ---
Accuracy: 34.01360544217687%
Precision: 0.1826086956521739
Recall: 0.875
F1: 0.302158273381295
FPR: 0.7642276422764228

--- Threshold = -3.4000000000000057 ---
Accuracy: 34.69387755102041%
Precision: 0.18421052631578946
Recall: 0.875
F1: 0.30434782608695654
FPR: 0.7560975609756098

--- Threshold = -3.350000000000006 ---
Accuracy: 34.69387755102041%
Precision: 0.18421052631578946
Recall: 0.875
F1: 0.30434782608695654
```

```
FPR: 0.7560975609756098

--- Threshold = -3.300000000000006 ---
Accuracy: 34.69387755102041%
Precision: 0.18421052631578946
Recall: 0.875
F1: 0.30434782608695654
FPR: 0.7560975609756098

--- Threshold = -3.250000000000006 ---
Accuracy: 34.69387755102041%
Precision: 0.18421052631578946
Recall: 0.875
F1: 0.30434782608695654
FPR: 0.7560975609756098

--- Threshold = -3.2000000000000064 ---
Accuracy: 34.69387755102041%
Precision: 0.18421052631578946
Recall: 0.875
F1: 0.30434782608695654
FPR: 0.7560975609756098

--- Threshold = -3.1500000000000066 ---
Accuracy: 35.374149659863946%
Precision: 0.18584070796460178
Recall: 0.875
F1: 0.30656934306569344
FPR: 0.7479674796747967

--- Threshold = -3.1000000000000068 ---
Accuracy: 36.054421768707485%
Precision: 0.1875
Recall: 0.875
F1: 0.3088235294117647
FPR: 0.7398373983739838

--- Threshold = -3.050000000000007 ---
Accuracy: 36.054421768707485%
Precision: 0.1875
Recall: 0.875
F1: 0.3088235294117647
FPR: 0.7398373983739838

--- Threshold = -3.000000000000007 ---
Accuracy: 37.41496598639456%
Precision: 0.19090909090909092
Recall: 0.875
```

```
F1: 0.3134328358208955
FPR: 0.7235772357723578

--- Threshold = -2.9500000000000073 ---
Accuracy: 37.41496598639456%
Precision: 0.19090909090909092
Recall: 0.875
F1: 0.3134328358208955
FPR: 0.7235772357723578

--- Threshold = -2.9000000000000075 ---
Accuracy: 37.41496598639456%
Precision: 0.19090909090909092
Recall: 0.875
F1: 0.3134328358208955
FPR: 0.7235772357723578

--- Threshold = -2.8500000000000076 ---
Accuracy: 37.41496598639456%
Precision: 0.19090909090909092
Recall: 0.875
F1: 0.3134328358208955
FPR: 0.7235772357723578

--- Threshold = -2.800000000000008 ---
Accuracy: 38.095238095238095%
Precision: 0.1926605504587156
Recall: 0.875
F1: 0.31578947368421056
FPR: 0.7154471544715447

--- Threshold = -2.750000000000008 ---
Accuracy: 40.136054421768705%
Precision: 0.19811320754716982
Recall: 0.875
F1: 0.3230769230769231
FPR: 0.6910569105691057

--- Threshold = -2.700000000000008 ---
Accuracy: 40.816326530612244%
Precision: 0.2
Recall: 0.875
F1: 0.32558139534883723
FPR: 0.6829268292682927

--- Threshold = -2.6500000000000083 ---
Accuracy: 41.49659863945578%
Precision: 0.20192307692307693
```

```
Recall: 0.875
F1: 0.328125
FPR: 0.6747967479674797

--- Threshold = -2.6000000000000085 ---
Accuracy: 41.49659863945578%
Precision: 0.20192307692307693
Recall: 0.875
F1: 0.328125
FPR: 0.6747967479674797

--- Threshold = -2.5500000000000087 ---
Accuracy: 41.49659863945578%
Precision: 0.20192307692307693
Recall: 0.875
F1: 0.328125
FPR: 0.6747967479674797

--- Threshold = -2.500000000000009 ---
Accuracy: 42.857142857142854%
Precision: 0.20588235294117646
Recall: 0.875
F1: 0.3333333333333333
FPR: 0.6585365853658537

--- Threshold = -2.450000000000009 ---
Accuracy: 44.89795918367347%
Precision: 0.21212121212121213
Recall: 0.875
F1: 0.3414634146341463
FPR: 0.6341463414634146

--- Threshold = -2.4000000000000092 ---
Accuracy: 44.89795918367347%
Precision: 0.21212121212121213
Recall: 0.875
F1: 0.3414634146341463
FPR: 0.6341463414634146

--- Threshold = -2.3500000000000094 ---
Accuracy: 45.57823129251701%
Precision: 0.21428571428571427
Recall: 0.875
F1: 0.34426229508196715
FPR: 0.6260162601626016

--- Threshold = -2.3000000000000096 ---
Accuracy: 47.61904761904761%
```

```
Precision: 0.22105263157894736
Recall: 0.875
F1: 0.35294117647058826
FPR: 0.6016260162601627

--- Threshold = -2.2500000000000098 ---
Accuracy: 48.29931972789115%
Precision: 0.22340425531914893
Recall: 0.875
F1: 0.35593220338983056
FPR: 0.5934959349593496

--- Threshold = -2.20000000000001 ---
Accuracy: 48.97959183673469%
Precision: 0.22580645161290322
Recall: 0.875
F1: 0.358974358974359
FPR: 0.5853658536585366

--- Threshold = -2.15000000000001 ---
Accuracy: 49.65986394557823%
Precision: 0.22826086956521738
Recall: 0.875
F1: 0.3620689655172414
FPR: 0.5772357723577236

--- Threshold = -2.1000000000000103 ---
Accuracy: 51.02040816326531%
Precision: 0.23333333333333334
Recall: 0.875
F1: 0.3684210526315789
FPR: 0.5609756097560976

--- Threshold = -2.0500000000000105 ---
Accuracy: 53.06122448979592%
Precision: 0.2413793103448276
Recall: 0.875
F1: 0.37837837837837845
FPR: 0.5365853658536586

--- Threshold = -2.0000000000000107 ---
Accuracy: 53.74149659863946%
Precision: 0.2441860465116279
Recall: 0.875
F1: 0.3818181818181817
FPR: 0.5284552845528455

--- Threshold = -1.9500000000000108 ---
```

```
Accuracy: 55.78231292517006%
Precision: 0.25301204819277107
Recall: 0.875
F1: 0.39252336448598135
FPR: 0.5040650406504065

--- Threshold = -1.900000000000011 ---
Accuracy: 55.78231292517006%
Precision: 0.25301204819277107
Recall: 0.875
F1: 0.39252336448598135
FPR: 0.5040650406504065

--- Threshold = -1.8500000000000112 ---
Accuracy: 55.78231292517006%
Precision: 0.25301204819277107
Recall: 0.875
F1: 0.39252336448598135
FPR: 0.5040650406504065

--- Threshold = -1.8000000000000114 ---
Accuracy: 56.4625850340136%
Precision: 0.25609756097560976
Recall: 0.875
F1: 0.39622641509433965
FPR: 0.4959349593495935

--- Threshold = -1.7500000000000115 ---
Accuracy: 57.14285714285714%
Precision: 0.25925925925925924
Recall: 0.875
F1: 0.4
FPR: 0.4878048780487805

--- Threshold = -1.7000000000000117 ---
Accuracy: 57.82312925170068%
Precision: 0.2625
Recall: 0.875
F1: 0.40384615384615385
FPR: 0.4796747967479675

--- Threshold = -1.650000000000012 ---
Accuracy: 57.14285714285714%
Precision: 0.25316455696202533
Recall: 0.8333333333333334
F1: 0.38834951456310685
FPR: 0.4796747967479675
```

```
--- Threshold = -1.600000000000012 ---
Accuracy: 58.50340136054422%
Precision: 0.2597402597402597
Recall: 0.8333333333333334
F1: 0.396039603960396
FPR: 0.4634146341463415

--- Threshold = -1.5500000000000123 ---
Accuracy: 58.50340136054422%
Precision: 0.2597402597402597
Recall: 0.8333333333333334
F1: 0.396039603960396
FPR: 0.4634146341463415

--- Threshold = -1.5000000000000124 ---
Accuracy: 59.863945578231295%
Precision: 0.26666666666666666
Recall: 0.8333333333333334
F1: 0.40404040404040403
FPR: 0.44715447154471544

--- Threshold = -1.4500000000000126 ---
Accuracy: 59.863945578231295%
Precision: 0.26666666666666666
Recall: 0.8333333333333334
F1: 0.40404040404040403
FPR: 0.44715447154471544

--- Threshold = -1.4000000000000128 ---
Accuracy: 61.224489795918366%
Precision: 0.273972602739726
Recall: 0.8333333333333334
F1: 0.41237113402061853
FPR: 0.43089430894308944

--- Threshold = -1.350000000000013 ---
Accuracy: 61.904761904761905%
Precision: 0.2777777777777778
Recall: 0.8333333333333334
F1: 0.4166666666666667
FPR: 0.42276422764227645

--- Threshold = -1.3000000000000131 ---
Accuracy: 61.904761904761905%
Precision: 0.2777777777777778
Recall: 0.8333333333333334
F1: 0.4166666666666667
FPR: 0.42276422764227645
```

```
--- Threshold = -1.2500000000000133 ---
Accuracy: 63.94557823129252%
Precision: 0.2898550724637681
Recall: 0.8333333333333334
F1: 0.4301075268817205
FPR: 0.3983739837398374

--- Threshold = -1.2000000000000135 ---
Accuracy: 65.3061224489796%
Precision: 0.29850746268656714
Recall: 0.8333333333333334
F1: 0.43956043956043955
FPR: 0.3821138211382114

--- Threshold = -1.1500000000000137 ---
Accuracy: 66.66666666666666%
Precision: 0.3076923076923077
Recall: 0.8333333333333334
F1: 0.449438202247191
FPR: 0.36585365853658536

--- Threshold = -1.1000000000000139 ---
Accuracy: 68.02721088435374%
Precision: 0.31746031746031744
Recall: 0.8333333333333334
F1: 0.4597701149425287
FPR: 0.34959349593495936

--- Threshold = -1.050000000000014 ---
Accuracy: 68.70748299319727%
Precision: 0.3225806451612903
Recall: 0.8333333333333334
F1: 0.46511627906976744
FPR: 0.34146341463414637

--- Threshold = -1.0000000000000142 ---
Accuracy: 68.70748299319727%
Precision: 0.3225806451612903
Recall: 0.8333333333333334
F1: 0.46511627906976744
FPR: 0.34146341463414637

--- Threshold = -0.9500000000000144 ---
Accuracy: 69.38775510204081%
Precision: 0.32786885245901637
Recall: 0.8333333333333334
F1: 0.47058823529411764
```

```
FPR: 0.3333333333333333

--- Threshold = -0.9000000000000146 ---
Accuracy: 70.74829931972789%
Precision: 0.3389830508474576
Recall: 0.8333333333333334
F1: 0.48192771084337344
FPR: 0.3170731707317073

--- Threshold = -0.8500000000000147 ---
Accuracy: 72.10884353741497%
Precision: 0.3508771929824561
Recall: 0.8333333333333334
F1: 0.4938271604938272
FPR: 0.3008130081300813

--- Threshold = -0.8000000000000149 ---
Accuracy: 74.82993197278913%
Precision: 0.37735849056603776
Recall: 0.8333333333333334
F1: 0.5194805194805195
FPR: 0.2682926829268293

--- Threshold = -0.7500000000000151 ---
Accuracy: 76.19047619047619%
Precision: 0.39215686274509803
Recall: 0.8333333333333334
F1: 0.5333333333333333
FPR: 0.25203252032520324

--- Threshold = -0.7000000000000153 ---
Accuracy: 76.19047619047619%
Precision: 0.39215686274509803
Recall: 0.8333333333333334
F1: 0.5333333333333333
FPR: 0.25203252032520324

--- Threshold = -0.6500000000000155 ---
Accuracy: 77.55102040816327%
Precision: 0.40425531914893614
Recall: 0.7916666666666666
F1: 0.5352112676056338
FPR: 0.22764227642276422

--- Threshold = -0.6000000000000156 ---
Accuracy: 78.2312925170068%
Precision: 0.41304347826086957
Recall: 0.7916666666666666
```

```
F1: 0.5428571428571428
FPR: 0.21951219512195122

--- Threshold = -0.5500000000000158 ---
Accuracy: 79.59183673469387%
Precision: 0.4318181818181818
Recall: 0.7916666666666666
F1: 0.5588235294117646
FPR: 0.2032520325203252

--- Threshold = -0.500000000000016 ---
Accuracy: 79.59183673469387%
Precision: 0.42857142857142855
Recall: 0.75
F1: 0.5454545454545454
FPR: 0.1951219512195122

--- Threshold = -0.45000000000001616 ---
Accuracy: 80.27210884353741%
Precision: 0.43902439024390244
Recall: 0.75
F1: 0.5538461538461539
FPR: 0.18699186991869918

--- Threshold = -0.40000000000001634 ---
Accuracy: 80.27210884353741%
Precision: 0.43902439024390244
Recall: 0.75
F1: 0.5538461538461539
FPR: 0.18699186991869918

--- Threshold = -0.3500000000000165 ---
Accuracy: 80.95238095238095%
Precision: 0.45
Recall: 0.75
F1: 0.5625
FPR: 0.17886178861788618

--- Threshold = -0.3000000000000167 ---
Accuracy: 80.27210884353741%
Precision: 0.4358974358974359
Recall: 0.7083333333333334
F1: 0.5396825396825398
FPR: 0.17886178861788618

--- Threshold = -0.2500000000000169 ---
Accuracy: 79.59183673469387%
Precision: 0.42105263157894735
```

```
Recall: 0.6666666666666666
F1: 0.5161290322580645
FPR: 0.17886178861788618

--- Threshold = -0.20000000000001705 ---
Accuracy: 80.95238095238095%
Precision: 0.4444444444444444
Recall: 0.6666666666666666
F1: 0.5333333333333333
FPR: 0.16260162601626016

--- Threshold = -0.15000000000001723 ---
Accuracy: 81.63265306122449%
Precision: 0.45714285714285713
Recall: 0.6666666666666666
F1: 0.5423728813559322
FPR: 0.15447154471544716

--- Threshold = -0.10000000000001741 ---
Accuracy: 82.31292517006803%
Precision: 0.46875
Recall: 0.625
F1: 0.5357142857142857
FPR: 0.13821138211382114

--- Threshold = -0.050000000000017586 ---
Accuracy: 82.31292517006803%
Precision: 0.46875
Recall: 0.625
F1: 0.5357142857142857
FPR: 0.13821138211382114

--- Threshold = -1.7763568394002505e-14 ---
Accuracy: 81.63265306122449%
Precision: 0.45161290322580644
Recall: 0.5833333333333334
F1: 0.509090909090909
FPR: 0.13821138211382114

--- Threshold = 0.04999999999998206 ---
Accuracy: 81.63265306122449%
Precision: 0.45161290322580644
Recall: 0.5833333333333334
F1: 0.509090909090909
FPR: 0.13821138211382114

--- Threshold = 0.09999999999998188 ---
Accuracy: 81.63265306122449%
```

```
Precision: 0.4482758620689655
Recall: 0.5416666666666666
F1: 0.4905660377358491
FPR: 0.13008130081300814

--- Threshold = 0.1499999999999817 ---
Accuracy: 82.31292517006803%
Precision: 0.4642857142857143
Recall: 0.5416666666666666
F1: 0.5
FPR: 0.12195121951219512

--- Threshold = 0.19999999999998153 ---
Accuracy: 82.99319727891157%
Precision: 0.48148148148148145
Recall: 0.5416666666666666
F1: 0.5098039215686274
FPR: 0.11382113821138211

--- Threshold = 0.24999999999998135 ---
Accuracy: 82.31292517006803%
Precision: 0.46153846153846156
Recall: 0.5
F1: 0.4800000000000001
FPR: 0.11382113821138211

--- Threshold = 0.29999999999998117 ---
Accuracy: 82.31292517006803%
Precision: 0.46153846153846156
Recall: 0.5
F1: 0.4800000000000001
FPR: 0.11382113821138211

--- Threshold = 0.349999999999981 ---
Accuracy: 82.31292517006803%
Precision: 0.45454545454545453
Recall: 0.4166666666666667
F1: 0.4347826086956522
FPR: 0.0975609756097561

--- Threshold = 0.3999999999999808 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.4166666666666667
F1: 0.45454545454545453
FPR: 0.08130081300813008

--- Threshold = 0.44999999999998064 ---
```

```
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.4166666666666667
F1: 0.45454545454545453
FPR: 0.08130081300813008

--- Threshold = 0.49999999999998046 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.375
F1: 0.4285714285714286
FPR: 0.07317073170731707

--- Threshold = 0.5499999999999803 ---
Accuracy: 82.31292517006803%
Precision: 0.4375
Recall: 0.2916666666666667
F1: 0.35000000000000003
FPR: 0.07317073170731707

--- Threshold = 0.5999999999999801 ---
Accuracy: 82.31292517006803%
Precision: 0.4375
Recall: 0.2916666666666667
F1: 0.35000000000000003
FPR: 0.07317073170731707

--- Threshold = 0.6499999999999799 ---
Accuracy: 82.99319727891157%
Precision: 0.4666666666666667
Recall: 0.2916666666666667
F1: 0.358974358974359
FPR: 0.06504065040650407

--- Threshold = 0.6999999999999797 ---
Accuracy: 82.99319727891157%
Precision: 0.46153846153846156
Recall: 0.25
F1: 0.32432432432432434
FPR: 0.056910569105691054

--- Threshold = 0.7499999999999796 ---
Accuracy: 82.99319727891157%
Precision: 0.46153846153846156
Recall: 0.25
F1: 0.32432432432432434
FPR: 0.056910569105691054
```

```
--- Threshold = 0.7999999999999794 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.25
F1: 0.3333333333333333
FPR: 0.04878048780487805

--- Threshold = 0.8499999999999792 ---
Accuracy: 85.03401360544217%
Precision: 0.6
Recall: 0.25
F1: 0.3529411764705882
FPR: 0.032520325203252036

--- Threshold = 0.899999999999979 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.16666666666666666
F1: 0.25
FPR: 0.032520325203252036

--- Threshold = 0.9499999999999789 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.16666666666666666
F1: 0.25
FPR: 0.032520325203252036

--- Threshold = 0.9999999999999787 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025

--- Threshold = 1.0499999999999785 ---
Accuracy: 85.03401360544217%
Precision: 0.6666666666666666
Recall: 0.16666666666666666
F1: 0.26666666666666666
FPR: 0.016260162601626018

--- Threshold = 1.0999999999999783 ---
Accuracy: 84.35374149659864%
Precision: 0.6
Recall: 0.125
F1: 0.20689655172413796
FPR: 0.016260162601626018
```

```
--- Threshold = 1.1499999999999782 ---
Accuracy: 84.35374149659864%
Precision: 0.6
Recall: 0.125
F1: 0.20689655172413796
FPR: 0.016260162601626018

--- Threshold = 1.199999999999978 ---
Accuracy: 84.35374149659864%
Precision: 0.6
Recall: 0.125
F1: 0.20689655172413796
FPR: 0.016260162601626018

--- Threshold = 1.2499999999999778 ---
Accuracy: 84.35374149659864%
Precision: 0.6
Recall: 0.125
F1: 0.20689655172413796
FPR: 0.016260162601626018

--- Threshold = 1.2999999999999776 ---
Accuracy: 84.35374149659864%
Precision: 0.6
Recall: 0.125
F1: 0.20689655172413796
FPR: 0.016260162601626018

--- Threshold = 1.3499999999999774 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.08333333333333333
F1: 0.14285714285714285
FPR: 0.016260162601626018

--- Threshold = 1.3999999999999773 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.08333333333333333
F1: 0.14285714285714285
FPR: 0.016260162601626018

--- Threshold = 1.449999999999977 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.08333333333333333
F1: 0.14285714285714285
```

```
FPR: 0.016260162601626018

--- Threshold = 1.499999999999977 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.08333333333333333
F1: 0.14285714285714285
FPR: 0.016260162601626018

--- Threshold = 1.5499999999999767 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.08333333333333333
F1: 0.14285714285714285
FPR: 0.016260162601626018

--- Threshold = 1.5999999999999766 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.08333333333333333
F1: 0.14285714285714285
FPR: 0.016260162601626018

--- Threshold = 1.6499999999999764 ---
Accuracy: 82.99319727891157%
Precision: 0.3333333333333333
Recall: 0.041666666666666664
F1: 0.07407407407407407
FPR: 0.016260162601626018

--- Threshold = 1.6999999999999762 ---
Accuracy: 82.99319727891157%
Precision: 0.3333333333333333
Recall: 0.041666666666666664
F1: 0.07407407407407407
FPR: 0.016260162601626018

--- Threshold = 1.749999999999976 ---
Accuracy: 82.99319727891157%
Precision: 0.3333333333333333
Recall: 0.041666666666666664
F1: 0.07407407407407407
FPR: 0.016260162601626018

--- Threshold = 1.7999999999999758 ---
Accuracy: 82.99319727891157%
Precision: 0.3333333333333333
Recall: 0.041666666666666664
```

```
F1: 0.07407407407407407
FPR: 0.016260162601626018

--- Threshold = 1.8499999999999757 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 1.8999999999999755 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 1.9499999999999753 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 1.9999999999999751 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 2.049999999999975 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 2.0999999999999748 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 2.1499999999999746 ---
Accuracy: 83.6734693877551%
Precision: 0.5
```

```
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 2.1999999999999744 ---
```

```
C:\Users\chyut\AppData\Local\Temp\ipykernel_24836\297337688.py:13:
RuntimeWarning: divide by zero encountered in scalar divide
  F1 = 2 / (1 / precision + 1 / recall)
```

```
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 2.2499999999999742 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 2.299999999999974 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 2.349999999999974 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 2.3999999999999737 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 2.4499999999999735 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
```

```
FPR: 0.008130081300813009

--- Threshold = 2.4999999999999734 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 2.549999999999973 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 2.599999999999973 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 2.649999999999973 ---
```

```
C:\Users\chyut\AppData\Local\Temp\ipykernel_24836\297337688.py:11:
RuntimeWarning: invalid value encountered in scalar divide
  precision = (tp) / (tp + fp)
```

```
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.6999999999999726 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.7499999999999725 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0
```

```
--- Threshold = 2.7999999999999723 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.849999999999972 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.899999999999972 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.9499999999999718 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.9999999999999716 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.0499999999999723 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.0999999999999712 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0
```

```
--- Threshold = 3.14999999999997 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.199999999999971 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.2499999999999716 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.2999999999999705 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.3499999999999694 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.39999999999997 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.449999999999971 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
```

```
FPR: 0.0

--- Threshold = 3.49999999999997 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.5499999999999687 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.5999999999999694 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.64999999999997 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.699999999999969 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.749999999999968 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.7999999999999687 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
```

```
F1: nan
FPR: 0.0

--- Threshold = 3.8499999999999694 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.8999999999999684 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.9499999999999673 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.999999999999968 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.049999999999969 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.099999999999968 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.149999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
```

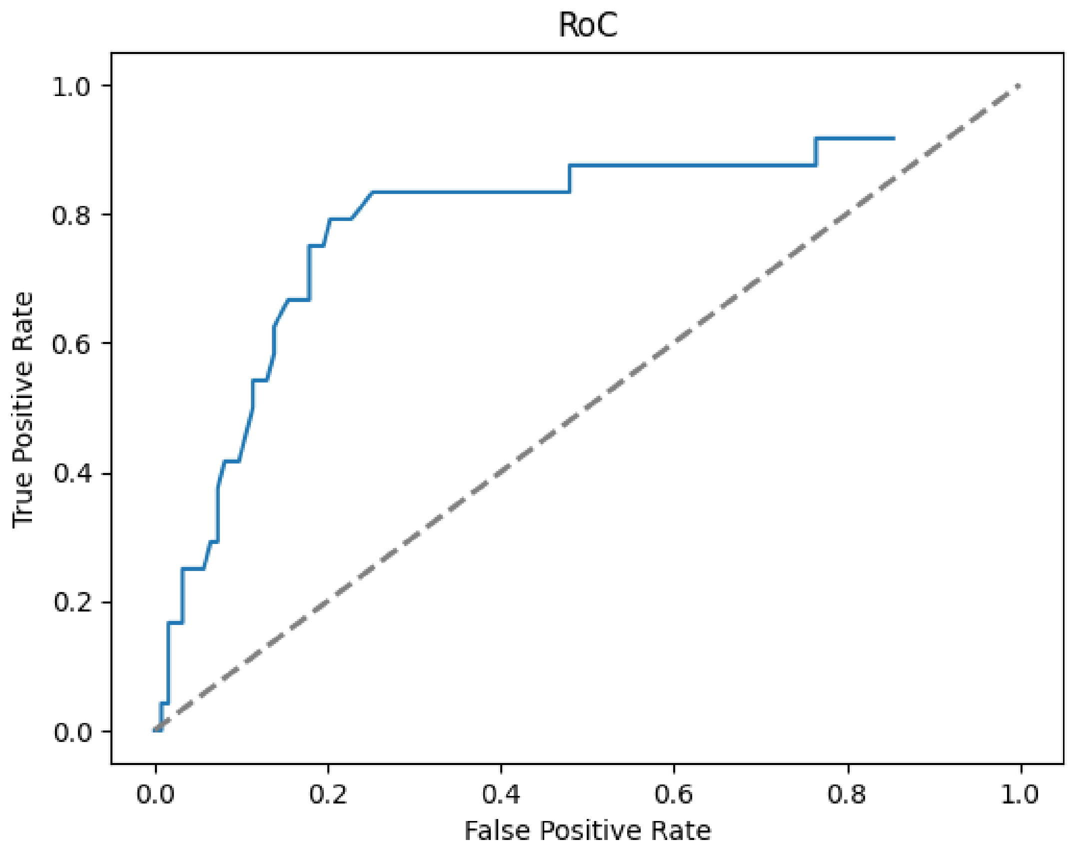
```
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.199999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.249999999999968 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.299999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.349999999999966 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.399999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.449999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.499999999999966 ---
Accuracy: 83.6734693877551%
```

```
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.549999999999965 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.599999999999966 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.649999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.6999999999999655 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.7499999999999645 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.799999999999965 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.849999999999966 ---
```

```
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.899999999999965 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.949999999999964 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

Best Accuracy: 0.8503401360544217 with responding threshold: 0.8499999999999792
Best F1: 0.5625 with responding threshold: -0.3500000000000165
```

1.0.19 T17. Plot the RoC of your classifier.

[47]:
```
plt.plot(history["FPR"], history["Recall"])
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', linewidth=2)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("RoC")
plt.show()
```



1.0.20 T18. Change the number of discretization bins to 5. What happens to the RoC curve? Which discretization is better? The number of discretization bins can be considered as a hyperparameter, and must be chosen by comparing the final performance.

[48]:

```
df = pd.read_csv('hr-employee-attrition-with-null.csv')
df.loc[df["Attrition"] == "no", "Attrition"] = 0.0
df.loc[df["Attrition"] == "yes", "Attrition"] = 1.0
string_categorical_col = ['Department', 'Attrition', 'BusinessTravel',␣
 ↪'EducationField', 'Gender', 'JobRole',
                              'MaritalStatus', 'Over18', 'OverTime']

# ENCODE STRING COLUMNS TO CATEGORICAL COLUMNS
for col in string_categorical_col:
    # INSERT CODE HERE
    df[col] = pd.Categorical(df[col]).codes
# HANDLE NULL NUMBERS
# I don't think we need to handle null?
```

```
# INSERT CODE HERE
df = df.loc[:, ~df.columns.isin(['EmployeeNumber', 'Unnamed: 0',␣
↪'EmployeeCount', 'StandardHours', 'Over18'])] # drop these columns
X = df.drop(["Attrition"], axis=1)
Y = df["Attrition"]

x_train, x_test, y_train, y_test = train_test_split(X, Y, stratify=Y,␣
↪test_size=0.1, random_state=12345)
```

[49]:
```
def hist(array, col_name, n_bin=10):
    nonan = array[~np.isnan(array)]

    # hist is the count for each bin
    # bin_edge is the edge values of the bins
    hist, bin_edges = np.histogram(nonan, n_bin)
    bin_edges[0] = -np.inf
    bin_edges[-1] = np.inf

    bin_indices = np.full_like(array, np.nan, dtype=float)
    bin_indices[~np.isnan(array)] = np.digitize(nonan, bin_edges)

    return bin_indices, bin_edges

discretize = []

for col in x_train.columns:
    if (x_train[col].nunique() > 10):
        x_train[col], _ = hist(x_train[col], col, 5)
        discretize.append(col)

print(discretize)
```

```
['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate', 'MonthlyIncome',
'MonthlyRate', 'PercentSalaryHike', 'TotalWorkingYears', 'YearsAtCompany',
'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']
```

[50]:
```
import importlib, SimpleBayesClassifier
importlib.reload(SimpleBayesClassifier)
from SimpleBayesClassifier import SimpleBayesClassifier

model = SimpleBayesClassifier(n_pos = np.sum(y_train == 1), n_neg = np.
↪sum(y_train == 0))

model.fit_params(np.array(x_train), np.array(y_train))
```

[50]:
```
([(array([0.08080808, 0.        , 0.31425365, 0.        , 0.        ,
          0.34680135, 0.        , 0.17059484, 0.        , 0.08754209]),
```

```
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.2027027 , 0.        , 0.        , 0.08558559, 0.        ,
        0.        , 0.13153153, 0.        , 0.        , 0.58018018]),
 array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
(array([0.1901566 , 0.        , 0.2114094 , 0.        , 0.        ,
        0.18120805, 0.        , 0.20917226, 0.        , 0.20805369]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.19099099, 0.        , 0.        , 0.03603604, 0.        ,
        0.        , 0.54594595, 0.        , 0.        , 0.22702703]),
 array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
(array([0.4778157 , 0.        , 0.26393629, 0.        , 0.        ,
        0.0705347 , 0.        , 0.09215017, 0.        , 0.09556314]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.10946408, 0.        , 0.19156214, 0.        , 0.        ,
        0.38312429, 0.        , 0.28164196, 0.        , 0.03420753]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.19459459, 0.01531532, 0.        , 0.33693694, 0.        ,
        0.07567568, 0.26666667, 0.        , 0.04234234, 0.06846847]),
 array([-inf, -0.4,  0.2,  0.8,  1.4,  2. ,  2.6,  3.2,  3.8,  4.4,  inf])),
(array([0.18423973, 0.        , 0.        , 0.18201998, 0.        ,
        0.        , 0.32741398, 0.        , 0.        , 0.3063263 ]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.2009009 , 0.        , 0.        , 0.        , 0.        ,
        0.33873874, 0.        , 0.        , 0.        , 0.46036036]),
 array([-inf, -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  inf])),
(array([0.18946188, 0.        , 0.20515695, 0.        , 0.        ,
        0.18161435, 0.        , 0.20403587, 0.        , 0.21973094]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.0407701 , 0.        , 0.        , 0.26274066, 0.        ,
        0.        , 0.59116648, 0.        , 0.        , 0.10532276]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.31513083, 0.        , 0.37997725, 0.        , 0.        ,
        0.16382253, 0.        , 0.09215017, 0.        , 0.04891923]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.1990991 , 0.08288288, 0.02702703, 0.11891892, 0.06396396,
        0.08918919, 0.05225225, 0.15495495, 0.17477477, 0.03693694]),
 array([-inf, -0.1,  0.8,  1.7,  2.6,  3.5,  4.4,  5.3,  6.2,  7.1,  inf])),
(array([0.18459796, 0.        , 0.        , 0.19705549, 0.        ,
        0.        , 0.29331823, 0.        , 0.        , 0.32502831]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.19099099, 0.        , 0.        , 0.1954955 , 0.        ,
        0.        , 0.38108108, 0.        , 0.        , 0.23243243]),
 array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
(array([0.44843049, 0.        , 0.28923767, 0.        , 0.        ,
        0.117713  , 0.        , 0.05044843, 0.        , 0.0941704 ]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.21235955, 0.        , 0.2011236 , 0.        , 0.        ,
```

```
        0.19325843, 0.        , 0.2011236 , 0.        , 0.19213483]),
  array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
 (array([0.13718821, 0.33446712, 0.10544218, 0.12131519, 0.1031746 ,
        0.03741497, 0.04421769, 0.04875283, 0.03061224, 0.03741497]),
  array([-inf,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,  8.1,  inf])),
 (array([0.19189189, 0.        , 0.        , 0.        , 0.        ,
        0.62162162, 0.        , 0.        , 0.        , 0.18648649]),
  array([-inf, -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  inf])),
 (array([0.40384615, 0.        , 0.27036199, 0.        , 0.        ,
        0.16968326, 0.        , 0.10972851, 0.        , 0.04638009]),
  array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
 (array([0.85310734, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.14689266]),
  array([-inf,  3.1,  3.2,  3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  inf])),
 (array([0.1868743 , 0.        , 0.        , 0.20244716, 0.        ,
        0.        , 0.30700779, 0.        , 0.        , 0.30367075]),
  array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
 (array([0.39595051, 0.        , 0.        , 0.44769404, 0.        ,
        0.        , 0.10686164, 0.        , 0.        , 0.04949381]),
  array([-inf,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8,  2.1,  2.4,  2.7,  inf])),
 (array([0.31291759, 0.        , 0.42538976, 0.        , 0.        ,
        0.16258352, 0.        , 0.06904232, 0.        , 0.03006682]),
  array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
 (array([0.03703704, 0.05274972, 0.        , 0.36026936, 0.        ,
        0.332211  , 0.07856341, 0.        , 0.09539843, 0.04377104]),
  array([-inf,  0.6,  1.2,  1.8,  2.4,  3. ,  3.6,  4.2,  4.8,  5.4,  inf])),
 (array([0.04519774, 0.        , 0.        , 0.23050847, 0.        ,
        0.        , 0.62146893, 0.        , 0.        , 0.10282486]),
  array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
 (array([0.61167228, 0.        , 0.26823793, 0.        , 0.        ,
        0.08080808, 0.        , 0.02581369, 0.        , 0.01346801]),
  array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
 (array([0.51727982, 0.        , 0.28205128, 0.        , 0.        ,
        0.14381271, 0.        , 0.04236343, 0.        , 0.01449275]),
  array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
 (array([0.73318386, 0.        , 0.1132287 , 0.        , 0.        ,
        0.09304933, 0.        , 0.03363229, 0.        , 0.02690583]),
  array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
 (array([0.53454134, 0.        , 0.10532276, 0.        , 0.        ,
        0.30577576, 0.        , 0.0407701 , 0.        , 0.01359003]),
  array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf]))],
[(array([0.24855491, 0.        , 0.36416185, 0.        , 0.        ,
        0.1849711 , 0.        , 0.11560694, 0.        , 0.0867052 ]),
  array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
 (array([0.19248826, 0.        , 0.        , 0.03286385, 0.        ,
        0.        , 0.23004695, 0.        , 0.        , 0.54460094]),
  array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
```

```
(array([0.23566879, 0.        , 0.20382166, 0.        , 0.        ,
        0.19745223, 0.        , 0.1910828 , 0.        , 0.17197452]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.22535211, 0.        , 0.        , 0.05164319, 0.        ,
        0.        , 0.41314554, 0.        , 0.        , 0.30985915]),
 array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
(array([0.37853107, 0.        , 0.23728814, 0.        , 0.        ,
        0.11299435, 0.        , 0.11299435, 0.        , 0.15819209]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.13068182, 0.        , 0.19886364, 0.        , 0.        ,
        0.41477273, 0.        , 0.22727273, 0.        , 0.02840909]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.18309859, 0.02816901, 0.        , 0.29577465, 0.        ,
        0.12676056, 0.22535211, 0.        , 0.03755869, 0.10328638]),
 array([-inf, -0.4,  0.2,  0.8,  1.4,  2. ,  2.6,  3.2,  3.8,  4.4,  inf])),
(array([0.26875, 0.     , 0.     , 0.2    , 0.     , 0.     , 0.2375 ,
        0.     , 0.     , 0.29375]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.19248826, 0.        , 0.        , 0.        , 0.        ,
        0.29577465, 0.        , 0.        , 0.        , 0.51173709]),
 array([-inf, -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  inf])),
(array([0.19526627, 0.        , 0.23668639, 0.        , 0.        ,
        0.20118343, 0.        , 0.15976331, 0.        , 0.20710059]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.1091954 , 0.        , 0.        , 0.31034483, 0.        ,
        0.        , 0.52873563, 0.        , 0.        , 0.05172414]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.60795455, 0.        , 0.23295455, 0.        , 0.        ,
        0.11931818, 0.        , 0.01704545, 0.        , 0.02272727]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.18779343, 0.02816901, 0.04694836, 0.19248826, 0.01877934,
        0.03755869, 0.00469484, 0.17370892, 0.18779343, 0.12206573]),
 array([-inf, -0.1,  0.8,  1.7,  2.6,  3.5,  4.4,  5.3,  6.2,  7.1,  inf])),
(array([0.24      , 0.        , 0.        , 0.2       , 0.        ,
        0.        , 0.30857143, 0.        , 0.        , 0.25142857]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.22065728, 0.        , 0.        , 0.11267606, 0.        ,
        0.        , 0.2629108 , 0.        , 0.        , 0.40375587]),
 array([-inf, -0.7, -0.4, -0.1,  0.2,  0.5,  0.8,  1.1,  1.4,  1.7,  inf])),
(array([0.68862275, 0.        , 0.17365269, 0.        , 0.        ,
        0.10179641, 0.        , 0.01796407, 0.        , 0.01796407]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.2       , 0.        , 0.22424242, 0.        , 0.        ,
        0.17575758, 0.        , 0.22424242, 0.        , 0.17575758]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.0960452 , 0.38983051, 0.07909605, 0.06779661, 0.08474576,
        0.06779661, 0.05649718, 0.06779661, 0.02824859, 0.06214689]),
```

```
 array([-inf,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,  8.1,  inf])),
(array([0.23474178, 0.        , 0.        , 0.        , 0.        ,
        0.36150235, 0.        , 0.        , 0.        , 0.40375587]),
 array([-inf, -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  inf])),
(array([0.49162011, 0.        , 0.2122905 , 0.        , 0.        ,
        0.1452514 , 0.        , 0.08379888, 0.        , 0.06703911]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.83529412, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.16470588]),
 array([-inf,  3.1,  3.2,  3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  inf])),
(array([0.26219512, 0.        , 0.        , 0.20121951, 0.        ,
        0.        , 0.2804878 , 0.        , 0.        , 0.25609756]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.65116279, 0.        , 0.        , 0.22674419, 0.        ,
        0.        , 0.06395349, 0.        , 0.        , 0.05813953]),
 array([-inf,  0.3,  0.6,  0.9,  1.2,  1.5,  1.8,  2.1,  2.4,  2.7,  inf])),
(array([0.55151515, 0.        , 0.3030303 , 0.        , 0.        ,
        0.0969697 , 0.        , 0.03030303, 0.        , 0.01818182]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.05202312, 0.05202312, 0.        , 0.42196532, 0.        ,
        0.28901734, 0.10404624, 0.        , 0.06358382, 0.01734104]),
 array([-inf,  0.6,  1.2,  1.8,  2.4,  3. ,  3.6,  4.2,  4.8,  5.4,  inf])),
(array([0.10588235, 0.        , 0.        , 0.23529412, 0.        ,
        0.        , 0.52352941, 0.        , 0.        , 0.13529412]),
 array([-inf,  1.3,  1.6,  1.9,  2.2,  2.5,  2.8,  3.1,  3.4,  3.7,  inf])),
(array([0.72674419, 0.        , 0.21511628, 0.        , 0.        ,
        0.03488372, 0.        , 0.01162791, 0.        , 0.01162791]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.68263473, 0.        , 0.20359281, 0.        , 0.        ,
        0.08982036, 0.        , 0.01796407, 0.        , 0.00598802]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.78235294, 0.        , 0.07058824, 0.        , 0.        ,
        0.09411765, 0.        , 0.02941176, 0.        , 0.02352941]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf])),
(array([0.69364162, 0.        , 0.06358382, 0.        , 0.        ,
        0.22543353, 0.        , 0.00578035, 0.        , 0.01156069]),
 array([-inf,  1.4,  1.8,  2.2,  2.6,  3. ,  3.4,  3.8,  4.2,  4.6,  inf]))])
```

[51]:
```python
hbin5 = {}
hbin5["Accuracy"] = []
hbin5["Precision"] = []
hbin5["Recall"] = []
hbin5["F1"] = []
hbin5["FPR"] = []

for each in t:
    print("--- Threshold =", each, "---")
```

```
    y_pred = np.array(model.predict(np.array(x_test), thresh=each))

    accuracy, precision, recall, f1, fpr = evaluate(y_test, y_pred)

    hbin5["Accuracy"].append(accuracy)
    hbin5["Precision"].append(precision)
    hbin5["Recall"].append(recall)
    hbin5["F1"].append(f1)
    hbin5["FPR"].append(fpr)

plt.plot(hbin5["FPR"], hbin5["Recall"])
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', linewidth=2)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("RoC with 5 bin")
plt.show()
```

```
--- Threshold = -5.0 ---
Accuracy: 50.34013605442177%
Precision: 0.24210526315789474
Recall: 0.9583333333333334
F1: 0.3865546218487395
FPR: 0.5853658536585366

--- Threshold = -4.95 ---
Accuracy: 50.34013605442177%
Precision: 0.24210526315789474
Recall: 0.9583333333333334
F1: 0.3865546218487395
FPR: 0.5853658536585366

--- Threshold = -4.9 ---
Accuracy: 51.70068027210885%
Precision: 0.24731182795698925
Recall: 0.9583333333333334
F1: 0.39316239316239315
FPR: 0.5691056910569106

--- Threshold = -4.8500000000000005 ---
Accuracy: 51.70068027210885%
Precision: 0.24731182795698925
Recall: 0.9583333333333334
F1: 0.39316239316239315
FPR: 0.5691056910569106

--- Threshold = -4.800000000000001 ---
Accuracy: 52.38095238095239%
Precision: 0.25
```

```
Recall: 0.9583333333333334
F1: 0.396551724137931
FPR: 0.5609756097560976

--- Threshold = -4.750000000000001 ---
Accuracy: 54.421768707483%
Precision: 0.25842696629213485
Recall: 0.9583333333333334
F1: 0.40707964601769914
FPR: 0.5365853658536586

--- Threshold = -4.700000000000001 ---
Accuracy: 55.78231292517006%
Precision: 0.26436781609195403
Recall: 0.9583333333333334
F1: 0.4144144144144144
FPR: 0.5203252032520326

--- Threshold = -4.650000000000001 ---
Accuracy: 56.4625850340136%
Precision: 0.26744186046511625
Recall: 0.9583333333333334
F1: 0.4181818181818182
FPR: 0.5121951219512195

--- Threshold = -4.600000000000001 ---
Accuracy: 58.50340136054422%
Precision: 0.27710843373493976
Recall: 0.9583333333333334
F1: 0.42990654205607476
FPR: 0.4878048780487805

--- Threshold = -4.550000000000002 ---
Accuracy: 59.183673469387756%
Precision: 0.2804878048780488
Recall: 0.9583333333333334
F1: 0.4339622641509434
FPR: 0.4796747967479675

--- Threshold = -4.500000000000002 ---
Accuracy: 59.183673469387756%
Precision: 0.2804878048780488
Recall: 0.9583333333333334
F1: 0.4339622641509434
FPR: 0.4796747967479675

--- Threshold = -4.450000000000002 ---
Accuracy: 59.863945578231295%
```

```
Precision: 0.2839506172839506
Recall: 0.9583333333333334
F1: 0.4380952380952381
FPR: 0.4715447154471545

--- Threshold = -4.400000000000002 ---
Accuracy: 59.863945578231295%
Precision: 0.2839506172839506
Recall: 0.9583333333333334
F1: 0.4380952380952381
FPR: 0.4715447154471545

--- Threshold = -4.350000000000002 ---
Accuracy: 60.544217687074834%
Precision: 0.2875
Recall: 0.9583333333333334
F1: 0.44230769230769224
FPR: 0.4634146341463415

--- Threshold = -4.3000000000000025 ---
Accuracy: 61.224489795918366%
Precision: 0.2911392405063291
Recall: 0.9583333333333334
F1: 0.4466019417475728
FPR: 0.45528455284552843

--- Threshold = -4.250000000000003 ---
Accuracy: 61.224489795918366%
Precision: 0.2857142857142857
Recall: 0.9166666666666666
F1: 0.43564356435643564
FPR: 0.44715447154471544

--- Threshold = -4.200000000000003 ---
Accuracy: 60.544217687074834%
Precision: 0.27631578947368424
Recall: 0.875
F1: 0.4200000000000001
FPR: 0.44715447154471544

--- Threshold = -4.150000000000003 ---
Accuracy: 61.224489795918366%
Precision: 0.28
Recall: 0.875
F1: 0.4242424242424243
FPR: 0.43902439024390244

--- Threshold = -4.100000000000003 ---
```

```
Accuracy: 62.585034013605444%
Precision: 0.2876712328767123
Recall: 0.875
F1: 0.4329896907216495
FPR: 0.42276422764227645

--- Threshold = -4.050000000000003 ---
Accuracy: 62.585034013605444%
Precision: 0.28169014084507044
Recall: 0.8333333333333334
F1: 0.42105263157894735
FPR: 0.4146341463414634

--- Threshold = -4.0000000000000036 ---
Accuracy: 63.94557823129252%
Precision: 0.2898550724637681
Recall: 0.8333333333333334
F1: 0.4301075268817205
FPR: 0.3983739837398374

--- Threshold = -3.9500000000000037 ---
Accuracy: 63.94557823129252%
Precision: 0.2898550724637681
Recall: 0.8333333333333334
F1: 0.4301075268817205
FPR: 0.3983739837398374

--- Threshold = -3.900000000000004 ---
Accuracy: 64.62585034013605%
Precision: 0.29411764705882354
Recall: 0.8333333333333334
F1: 0.4347826086956522
FPR: 0.3902439024390244

--- Threshold = -3.850000000000004 ---
Accuracy: 65.3061224489796%
Precision: 0.29850746268656714
Recall: 0.8333333333333334
F1: 0.43956043956043955
FPR: 0.3821138211382114

--- Threshold = -3.8000000000000043 ---
Accuracy: 65.3061224489796%
Precision: 0.2857142857142857
Recall: 0.75
F1: 0.4137931034482759
FPR: 0.36585365853658536
```

```
--- Threshold = -3.7500000000000044 ---
Accuracy: 65.98639455782312%
Precision: 0.2903225806451613
Recall: 0.75
F1: 0.4186046511627907
FPR: 0.35772357723577236

--- Threshold = -3.7000000000000046 ---
Accuracy: 68.02721088435374%
Precision: 0.3050847457627119
Recall: 0.75
F1: 0.4337349397590362
FPR: 0.3333333333333333

--- Threshold = -3.650000000000005 ---
Accuracy: 68.70748299319727%
Precision: 0.3103448275862069
Recall: 0.75
F1: 0.43902439024390244
FPR: 0.3252032520325203

--- Threshold = -3.600000000000005 ---
Accuracy: 68.70748299319727%
Precision: 0.2962962962962963
Recall: 0.6666666666666666
F1: 0.41025641025641024
FPR: 0.3089430894308943

--- Threshold = -3.550000000000005 ---
Accuracy: 68.02721088435374%
Precision: 0.2830188679245283
Recall: 0.625
F1: 0.38961038961038963
FPR: 0.3089430894308943

--- Threshold = -3.5000000000000053 ---
Accuracy: 68.02721088435374%
Precision: 0.27450980392156865
Recall: 0.5833333333333334
F1: 0.37333333333333335
FPR: 0.3008130081300813

--- Threshold = -3.4500000000000055 ---
Accuracy: 70.06802721088435%
Precision: 0.2916666666666667
Recall: 0.5833333333333334
F1: 0.38888888888888895
FPR: 0.2764227642276423
```

```
--- Threshold = -3.4000000000000057 ---
Accuracy: 70.06802721088435%
Precision: 0.2916666666666667
Recall: 0.5833333333333334
F1: 0.38888888888888895
FPR: 0.2764227642276423

--- Threshold = -3.350000000000006 ---
Accuracy: 70.06802721088435%
Precision: 0.2916666666666667
Recall: 0.5833333333333334
F1: 0.38888888888888895
FPR: 0.2764227642276423

--- Threshold = -3.300000000000006 ---
Accuracy: 71.42857142857143%
Precision: 0.30434782608695654
Recall: 0.5833333333333334
F1: 0.4
FPR: 0.2601626016260163

--- Threshold = -3.250000000000006 ---
Accuracy: 72.10884353741497%
Precision: 0.3111111111111111
Recall: 0.5833333333333334
F1: 0.40579710144927533
FPR: 0.25203252032520324

--- Threshold = -3.2000000000000064 ---
Accuracy: 72.78911564625851%
Precision: 0.3181818181818182
Recall: 0.5833333333333334
F1: 0.411764705882353
FPR: 0.24390243902439024

--- Threshold = -3.1500000000000066 ---
Accuracy: 72.10884353741497%
Precision: 0.3023255813953488
Recall: 0.5416666666666666
F1: 0.38805970149253727
FPR: 0.24390243902439024

--- Threshold = -3.1000000000000068 ---
Accuracy: 72.78911564625851%
Precision: 0.30952380952380953
Recall: 0.5416666666666666
F1: 0.393939393939394
```

```
FPR: 0.23577235772357724

--- Threshold = -3.050000000000007 ---
Accuracy: 72.10884353741497%
Precision: 0.2926829268292683
Recall: 0.5
F1: 0.3692307692307692
FPR: 0.23577235772357724

--- Threshold = -3.000000000000007 ---
Accuracy: 72.10884353741497%
Precision: 0.2926829268292683
Recall: 0.5
F1: 0.3692307692307692
FPR: 0.23577235772357724

--- Threshold = -2.9500000000000073 ---
Accuracy: 72.10884353741497%
Precision: 0.2926829268292683
Recall: 0.5
F1: 0.3692307692307692
FPR: 0.23577235772357724

--- Threshold = -2.9000000000000075 ---
Accuracy: 72.10884353741497%
Precision: 0.28205128205128205
Recall: 0.4583333333333333
F1: 0.3492063492063492
FPR: 0.22764227642276422

--- Threshold = -2.8500000000000076 ---
Accuracy: 72.10884353741497%
Precision: 0.28205128205128205
Recall: 0.4583333333333333
F1: 0.3492063492063492
FPR: 0.22764227642276422

--- Threshold = -2.800000000000008 ---
Accuracy: 72.10884353741497%
Precision: 0.28205128205128205
Recall: 0.4583333333333333
F1: 0.3492063492063492
FPR: 0.22764227642276422

--- Threshold = -2.750000000000008 ---
Accuracy: 73.46938775510205%
Precision: 0.2972972972972973
Recall: 0.4583333333333333
```

```
F1: 0.36065573770491804
FPR: 0.21138211382113822

--- Threshold = -2.700000000000008 ---
Accuracy: 74.14965986394559%
Precision: 0.28125
Recall: 0.375
F1: 0.32142857142857145
FPR: 0.18699186991869918

--- Threshold = -2.6500000000000083 ---
Accuracy: 74.14965986394559%
Precision: 0.28125
Recall: 0.375
F1: 0.32142857142857145
FPR: 0.18699186991869918

--- Threshold = -2.6000000000000085 ---
Accuracy: 74.82993197278913%
Precision: 0.2903225806451613
Recall: 0.375
F1: 0.32727272727272727
FPR: 0.17886178861788618

--- Threshold = -2.5500000000000087 ---
Accuracy: 74.14965986394559%
Precision: 0.26666666666666666
Recall: 0.3333333333333333
F1: 0.2962962962962963
FPR: 0.17886178861788618

--- Threshold = -2.500000000000009 ---
Accuracy: 75.51020408163265%
Precision: 0.2857142857142857
Recall: 0.3333333333333333
F1: 0.3076923076923077
FPR: 0.16260162601626016

--- Threshold = -2.450000000000009 ---
Accuracy: 76.87074829931973%
Precision: 0.3076923076923077
Recall: 0.3333333333333333
F1: 0.32
FPR: 0.14634146341463414

--- Threshold = -2.4000000000000092 ---
Accuracy: 76.87074829931973%
Precision: 0.3076923076923077
```

```
Recall: 0.3333333333333333
F1: 0.32
FPR: 0.14634146341463414

--- Threshold = -2.3500000000000094 ---
Accuracy: 77.55102040816327%
Precision: 0.32
Recall: 0.3333333333333333
F1: 0.32653061224489793
FPR: 0.13821138211382114

--- Threshold = -2.3000000000000096 ---
Accuracy: 77.55102040816327%
Precision: 0.32
Recall: 0.3333333333333333
F1: 0.32653061224489793
FPR: 0.13821138211382114

--- Threshold = -2.2500000000000098 ---
Accuracy: 77.55102040816327%
Precision: 0.32
Recall: 0.3333333333333333
F1: 0.32653061224489793
FPR: 0.13821138211382114

--- Threshold = -2.20000000000001 ---
Accuracy: 78.2312925170068%
Precision: 0.3333333333333333
Recall: 0.3333333333333333
F1: 0.3333333333333333
FPR: 0.13008130081300814

--- Threshold = -2.15000000000001 ---
Accuracy: 78.91156462585033%
Precision: 0.34782608695652173
Recall: 0.3333333333333333
F1: 0.3404255319148936
FPR: 0.12195121951219512

--- Threshold = -2.1000000000000103 ---
Accuracy: 78.91156462585033%
Precision: 0.34782608695652173
Recall: 0.3333333333333333
F1: 0.3404255319148936
FPR: 0.12195121951219512

--- Threshold = -2.0500000000000105 ---
Accuracy: 78.91156462585033%
```

```
Precision: 0.34782608695652173
Recall: 0.3333333333333333
F1: 0.3404255319148936
FPR: 0.12195121951219512

--- Threshold = -2.0000000000000107 ---
Accuracy: 79.59183673469387%
Precision: 0.36363636363636365
Recall: 0.3333333333333333
F1: 0.34782608695652173
FPR: 0.11382113821138211

--- Threshold = -1.9500000000000108 ---
Accuracy: 80.95238095238095%
Precision: 0.4
Recall: 0.3333333333333333
F1: 0.36363636363636365
FPR: 0.0975609756097561

--- Threshold = -1.900000000000011 ---
Accuracy: 81.63265306122449%
Precision: 0.42105263157894735
Recall: 0.3333333333333333
F1: 0.37209302325581395
FPR: 0.08943089430894309

--- Threshold = -1.8500000000000112 ---
Accuracy: 82.31292517006803%
Precision: 0.4444444444444444
Recall: 0.3333333333333333
F1: 0.38095238095238093
FPR: 0.08130081300813008

--- Threshold = -1.8000000000000114 ---
Accuracy: 82.99319727891157%
Precision: 0.47058823529411764
Recall: 0.3333333333333333
F1: 0.3902439024390244
FPR: 0.07317073170731707

--- Threshold = -1.7500000000000115 ---
Accuracy: 82.99319727891157%
Precision: 0.47058823529411764
Recall: 0.3333333333333333
F1: 0.3902439024390244
FPR: 0.07317073170731707

--- Threshold = -1.7000000000000117 ---
```

```
Accuracy: 82.99319727891157%
Precision: 0.47058823529411764
Recall: 0.3333333333333333
F1: 0.3902439024390244
FPR: 0.07317073170731707

--- Threshold = -1.650000000000012 ---
Accuracy: 84.35374149659864%
Precision: 0.5333333333333333
Recall: 0.3333333333333333
F1: 0.41025641025641024
FPR: 0.056910569105691054

--- Threshold = -1.600000000000012 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.2916666666666667
F1: 0.3684210526315789
FPR: 0.056910569105691054

--- Threshold = -1.5500000000000123 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.2916666666666667
F1: 0.3684210526315789
FPR: 0.056910569105691054

--- Threshold = -1.5000000000000124 ---
Accuracy: 84.35374149659864%
Precision: 0.5384615384615384
Recall: 0.2916666666666667
F1: 0.3783783783783784
FPR: 0.04878048780487805

--- Threshold = -1.4500000000000126 ---
Accuracy: 84.35374149659864%
Precision: 0.5384615384615384
Recall: 0.2916666666666667
F1: 0.3783783783783784
FPR: 0.04878048780487805

--- Threshold = -1.4000000000000128 ---
Accuracy: 84.35374149659864%
Precision: 0.5384615384615384
Recall: 0.2916666666666667
F1: 0.3783783783783784
FPR: 0.04878048780487805
```

```
--- Threshold = -1.350000000000013 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.25
F1: 0.3333333333333333
FPR: 0.04878048780487805

--- Threshold = -1.3000000000000131 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.25
F1: 0.3333333333333333
FPR: 0.04878048780487805

--- Threshold = -1.2500000000000133 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.25
F1: 0.3333333333333333
FPR: 0.04878048780487805

--- Threshold = -1.2000000000000135 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.25
F1: 0.3333333333333333
FPR: 0.04878048780487805

--- Threshold = -1.1500000000000137 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.20833333333333334
F1: 0.29411764705882354
FPR: 0.04065040650406504

--- Threshold = -1.1000000000000139 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.20833333333333334
F1: 0.29411764705882354
FPR: 0.04065040650406504

--- Threshold = -1.050000000000014 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025
```

```
--- Threshold = -1.0000000000000142 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025

--- Threshold = -0.9500000000000144 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025

--- Threshold = -0.9000000000000146 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025

--- Threshold = -0.8500000000000147 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025

--- Threshold = -0.8000000000000149 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025

--- Threshold = -0.7500000000000151 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025

--- Threshold = -0.7000000000000153 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
```

```
FPR: 0.024390243902439025

--- Threshold = -0.6500000000000155 ---
Accuracy: 84.35374149659864%
Precision: 0.5714285714285714
Recall: 0.16666666666666666
F1: 0.25806451612903225
FPR: 0.024390243902439025

--- Threshold = -0.6000000000000156 ---
Accuracy: 85.03401360544217%
Precision: 0.6666666666666666
Recall: 0.16666666666666666
F1: 0.26666666666666666
FPR: 0.016260162601626018

--- Threshold = -0.5500000000000158 ---
Accuracy: 85.03401360544217%
Precision: 0.6666666666666666
Recall: 0.16666666666666666
F1: 0.26666666666666666
FPR: 0.016260162601626018

--- Threshold = -0.500000000000016 ---
Accuracy: 85.03401360544217%
Precision: 0.6666666666666666
Recall: 0.16666666666666666
F1: 0.26666666666666666
FPR: 0.016260162601626018

--- Threshold = -0.45000000000001616 ---
Accuracy: 85.03401360544217%
Precision: 0.6666666666666666
Recall: 0.16666666666666666
F1: 0.26666666666666666
FPR: 0.016260162601626018

--- Threshold = -0.40000000000001634 ---
Accuracy: 85.03401360544217%
Precision: 0.6666666666666666
Recall: 0.16666666666666666
F1: 0.26666666666666666
FPR: 0.016260162601626018

--- Threshold = -0.3500000000000165 ---
Accuracy: 85.03401360544217%
Precision: 0.6666666666666666
Recall: 0.16666666666666666
```

```
F1: 0.26666666666666666
FPR: 0.016260162601626018

--- Threshold = -0.3000000000000167 ---
Accuracy: 85.03401360544217%
Precision: 0.6666666666666666
Recall: 0.16666666666666666
F1: 0.26666666666666666
FPR: 0.016260162601626018

--- Threshold = -0.2500000000000169 ---
Accuracy: 84.35374149659864%
Precision: 0.6
Recall: 0.125
F1: 0.20689655172413796
FPR: 0.016260162601626018

--- Threshold = -0.20000000000001705 ---
Accuracy: 84.35374149659864%
Precision: 0.6
Recall: 0.125
F1: 0.20689655172413796
FPR: 0.016260162601626018

--- Threshold = -0.15000000000001723 ---
Accuracy: 85.03401360544217%
Precision: 0.75
Recall: 0.125
F1: 0.21428571428571427
FPR: 0.008130081300813009

--- Threshold = -0.10000000000001741 ---
Accuracy: 84.35374149659864%
Precision: 0.6666666666666666
Recall: 0.08333333333333333
F1: 0.14814814814814814
FPR: 0.008130081300813009

--- Threshold = -0.050000000000017586 ---
Accuracy: 84.35374149659864%
Precision: 0.6666666666666666
Recall: 0.08333333333333333
F1: 0.14814814814814814
FPR: 0.008130081300813009

--- Threshold = -1.7763568394002505e-14 ---
Accuracy: 84.35374149659864%
Precision: 0.6666666666666666
```

```
Recall: 0.08333333333333333
F1: 0.14814814814814814
FPR: 0.008130081300813009

--- Threshold = 0.04999999999998206 ---
Accuracy: 84.35374149659864%
Precision: 0.6666666666666666
Recall: 0.08333333333333333
F1: 0.14814814814814814
FPR: 0.008130081300813009

--- Threshold = 0.09999999999998188 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 0.1499999999999817 ---
Accuracy: 83.6734693877551%
Precision: 0.5
Recall: 0.041666666666666664
F1: 0.07692307692307693
FPR: 0.008130081300813009

--- Threshold = 0.19999999999998153 ---
```

```
C:\Users\chyut\AppData\Local\Temp\ipykernel_24836\297337688.py:13:
RuntimeWarning: divide by zero encountered in scalar divide
  F1 = 2 / (1 / precision + 1 / recall)
```

```
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 0.24999999999998135 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 0.29999999999998117 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
```

```
FPR: 0.008130081300813009

--- Threshold = 0.3499999999999981 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 0.3999999999999808 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 0.44999999999998064 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 0.49999999999998046 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 0.5499999999999803 ---
Accuracy: 82.99319727891157%
Precision: 0.0
Recall: 0.0
F1: 0.0
FPR: 0.008130081300813009

--- Threshold = 0.5999999999999801 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 0.6499999999999799 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
```

```
F1: nan
FPR: 0.0

--- Threshold = 0.6999999999999797 ---
```

```
C:\Users\chyut\AppData\Local\Temp\ipykernel_24836\297337688.py:11:
RuntimeWarning: invalid value encountered in scalar divide
  precision = (tp) / (tp + fp)
```

```
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 0.7499999999999796 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 0.7999999999999794 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 0.8499999999999792 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 0.899999999999979 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 0.9499999999999789 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0
```

```
--- Threshold = 0.9999999999999787 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.0499999999999785 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.0999999999999783 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.1499999999999782 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.199999999999978 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.2499999999999778 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.2999999999999776 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
```

```
FPR: 0.0

--- Threshold = 1.3499999999999774 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.3999999999999773 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.449999999999977 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.499999999999977 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.5499999999999767 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.5999999999999766 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.6499999999999764 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
```

```
F1: nan
FPR: 0.0

--- Threshold = 1.6999999999999762 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.749999999999976 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.7999999999999758 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.8499999999999757 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.8999999999999755 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.9499999999999753 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 1.9999999999999751 ---
Accuracy: 83.6734693877551%
Precision: nan
```

```
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.049999999999975 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.0999999999999748 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.1499999999999746 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.1999999999999744 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.2499999999999742 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.299999999999974 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.349999999999974 ---
Accuracy: 83.6734693877551%
```

```
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.3999999999999737 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.4499999999999735 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.4999999999999734 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.549999999999973 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.599999999999973 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.649999999999973 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.6999999999999726 ---
```

```
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.7499999999999725 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.7999999999999723 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.849999999999972 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.899999999999972 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.9499999999999718 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 2.9999999999999716 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0
```

```
--- Threshold = 3.0499999999999723 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.0999999999999712 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.14999999999997 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.199999999999971 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.2499999999999716 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.2999999999999705 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.3499999999999694 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0
```

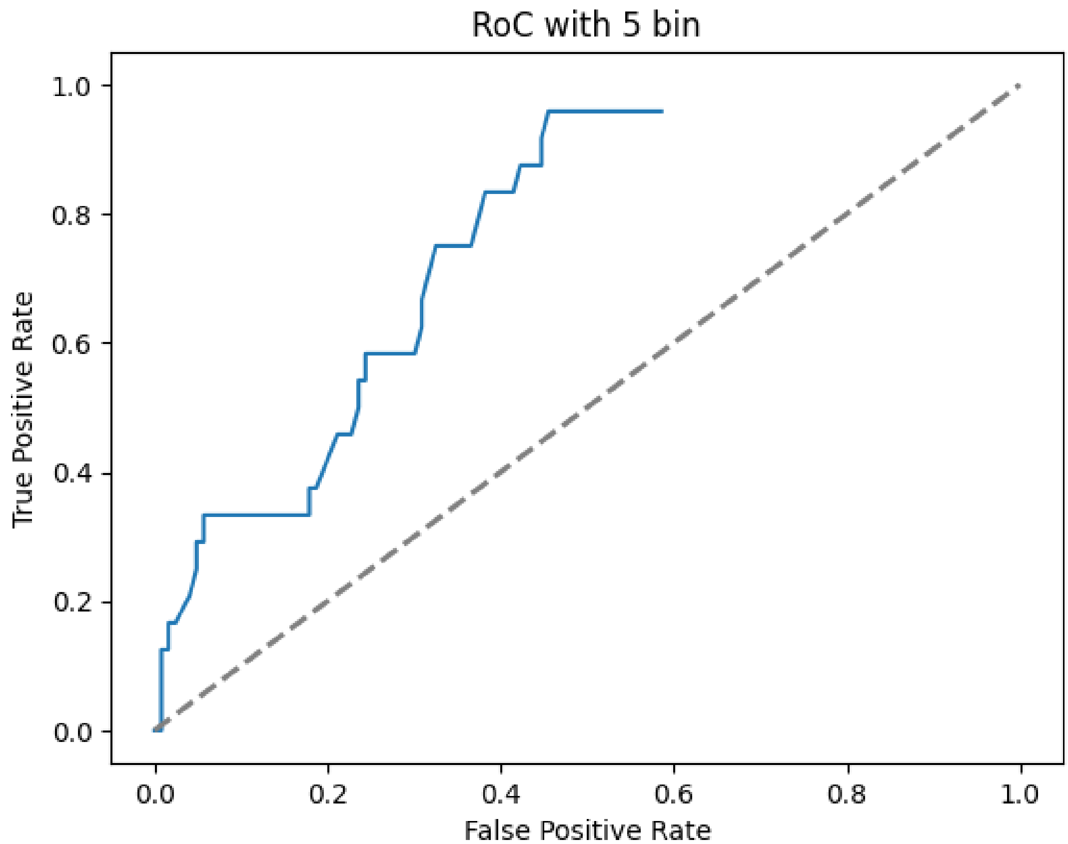
```
--- Threshold = 3.39999999999997 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.449999999999971 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.49999999999997 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.5499999999999687 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.5999999999999694 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.64999999999997 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.699999999999969 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
```

```
FPR: 0.0

--- Threshold = 3.749999999999968 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.7999999999999687 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.8499999999999694 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.8999999999999684 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.9499999999999673 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 3.999999999999968 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.049999999999969 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
```

```
F1: nan
FPR: 0.0

--- Threshold = 4.099999999999968 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.149999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.199999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.249999999999968 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.299999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.349999999999966 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.399999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
```

```
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.449999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.499999999999966 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.549999999999965 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.599999999999966 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.649999999999967 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.6999999999999655 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.7499999999999645 ---
Accuracy: 83.6734693877551%
```

```
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.799999999999965 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.849999999999966 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.899999999999965 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0

--- Threshold = 4.949999999999964 ---
Accuracy: 83.6734693877551%
Precision: nan
Recall: 0.0
F1: nan
FPR: 0.0
```



1.0.21 OT4

[52]:
```
df = pd.read_csv('hr-employee-attrition-with-null.csv')
df.loc[df["Attrition"] == "no", "Attrition"] = 0.0
df.loc[df["Attrition"] == "yes", "Attrition"] = 1.0
string_categorical_col = ['Department', 'Attrition', 'BusinessTravel',␣
 ↪'EducationField', 'Gender', 'JobRole',
                          'MaritalStatus', 'Over18', 'OverTime']

for col in string_categorical_col:
    df[col] = pd.Categorical(df[col]).codes

df = df.loc[:, ~df.columns.isin(['EmployeeNumber', 'Unnamed: 0',␣
 ↪'EmployeeCount', 'StandardHours', 'Over18'])] # drop these columns
X = df.drop(["Attrition"], axis=1)
Y = df["Attrition"]

accuracies = []
```

```python
for i in range(10):
    x_train, x_test, y_train, y_test = train_test_split(X, Y, stratify=Y,␣
 ↪test_size=0.1)

    def hist(array, col_name, n_bin=10):
        nonan = array[~np.isnan(array)]

        # hist is the count for each bin
        # bin_edge is the edge values of the bins
        hist, bin_edges = np.histogram(nonan, n_bin)
        bin_edges[0] = -np.inf
        bin_edges[-1] = np.inf

        bin_indices = np.full_like(array, np.nan, dtype=float)
        bin_indices[~np.isnan(array)] = np.digitize(nonan, bin_edges)

        return bin_indices, bin_edges

    discretize = []

    for col in x_train.columns:
        if (x_train[col].nunique() > 10):
            x_train[col], _ = hist(x_train[col], col, 5)
            discretize.append(col)

    import importlib, SimpleBayesClassifier
    importlib.reload(SimpleBayesClassifier)
    from SimpleBayesClassifier import SimpleBayesClassifier

    model = SimpleBayesClassifier(n_pos = np.sum(y_train == 1), n_neg = np.
 ↪sum(y_train == 0))

    model.fit_gaussian_params(np.array(x_train), np.array(y_train))
    y_pred = model.gaussian_predict(np.array(x_test))

    accuracy, _, _, _, _ = evaluate(np.array(y_test), y_pred, False)

    accuracies.append(accuracy)

accuracies = np.array(accuracies)
print("Mean:", np.mean(accuracies))
print("Variance:", np.std(accuracies) ** 2)
```

```
Mean: 0.8251700680272108
Variance: 0.00042621130084686886
```