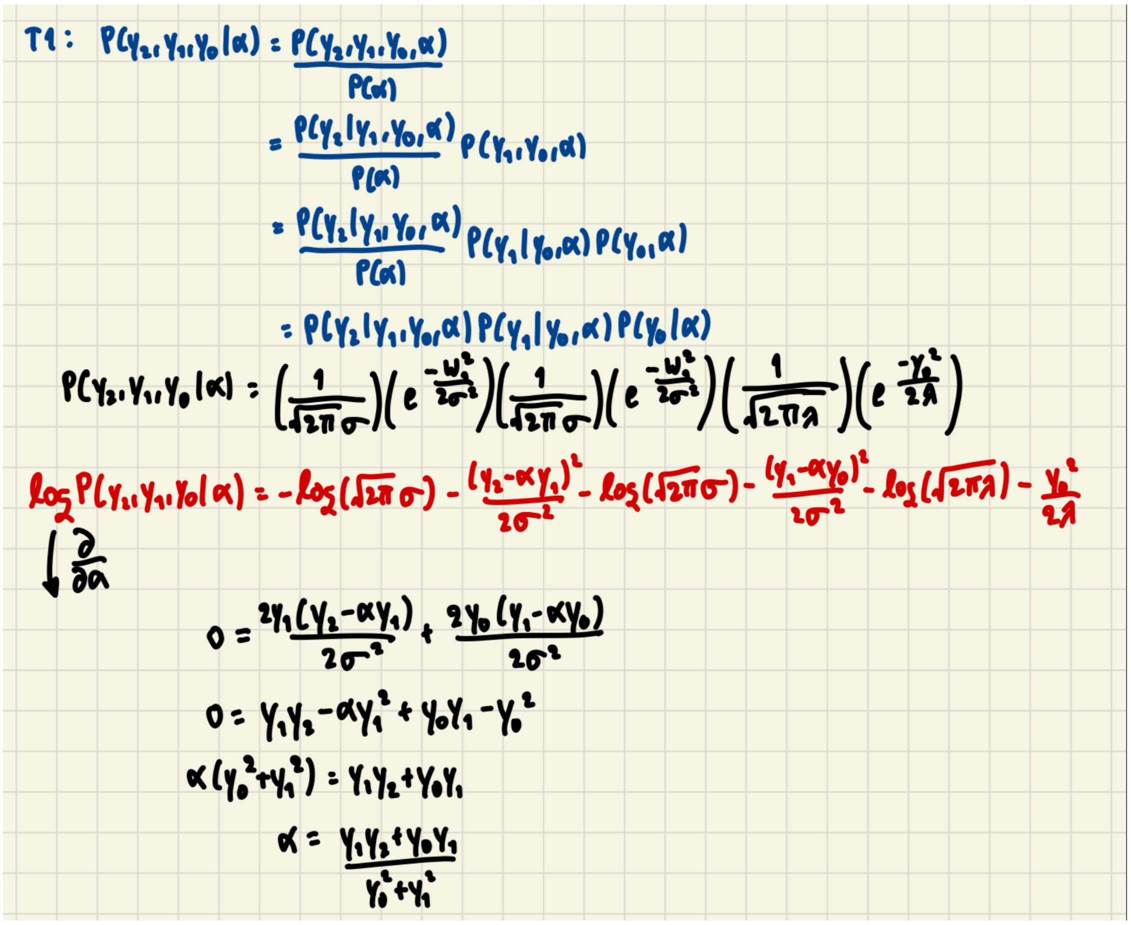
Pattern_HW2_student_2024

February 3, 2024

0.1 For interactive notebook please see on this link

0.2 T1



T1: $P(y_2,y_1,y_0|\alpha) = \dfrac{P(y_2,y_1,y_0,\alpha)}{P(\alpha)}$

$$= \frac{P(y_2|y_1,y_0,\alpha)}{P(\alpha)} P(y_1,y_0,\alpha)$$

$$= \frac{P(y_2|y_1,y_0,\alpha)}{P(\alpha)} P(y_1|y_0,\alpha) P(y_0,\alpha)$$

$$= P(y_2|y_1,y_0,\alpha) P(y_1|y_0,\alpha) P(y_0|\alpha)$$

$$P(y_2,y_1,y_0|\alpha) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)\left(e^{\frac{-W_2^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma}\right)\left(e^{\frac{-W_1^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi\lambda}}\right)\left(e^{\frac{-y_0^2}{2\lambda}}\right)$$

$$\log_2 P(y_2,y_1,y_0|\alpha) = -\log_2(\sqrt{2\pi}\sigma) - \frac{(y_2-\alpha y_1)^2}{2\sigma^2} - \log_2(\sqrt{2\pi}\sigma) - \frac{(y_1-\alpha y_0)^2}{2\sigma^2} - \log_2(\sqrt{2\pi\lambda}) - \frac{y_0^2}{2\lambda}$$

$\downarrow \frac{\partial}{\partial \alpha}$

$$0 = \frac{2y_1(y_2-\alpha y_1)}{2\sigma^2} + \frac{2y_0(y_1-\alpha y_0)}{2\sigma^2}$$

$$0 = y_1y_2 - \alpha y_1^2 + y_0y_1 - y_0^2$$

$$\alpha(y_0^2+y_1^2) = y_1y_2 + y_0y_1$$

$$\alpha = \frac{y_1y_2+y_0y_1}{y_0^2+y_1^2}$$

0.2.1 OT1

OT1: $\log(P_{N+1}, P_N, P_{N-1} \ldots \mid \alpha)$

$$= \left[\prod_{i=1}^{N}\left(\frac{1}{\sqrt{2\pi}\sigma}\right)e^{\frac{-u_i^2}{2\sigma^2}}\right]\left(\frac{1}{\sqrt{2\pi\lambda}}\right)\left(e^{\frac{-y_0^2}{2\lambda}}\right)$$

$\log \downarrow$

$$= \sum_{i=1}^{N}\left(-\log(\sqrt{2\pi}\sigma) - \frac{(y_{i+1}-\alpha y_i)^2}{2\sigma^2}\right) - \log(\sqrt{2\pi\lambda}) - \frac{y_0^2}{2\lambda}$$

$\frac{\partial}{\partial\alpha} \downarrow$

$$= \sum_{i=1}^{N}\left(\frac{2y_i(y_{i+1}-\alpha y_i)}{2\sigma^2}\right)$$

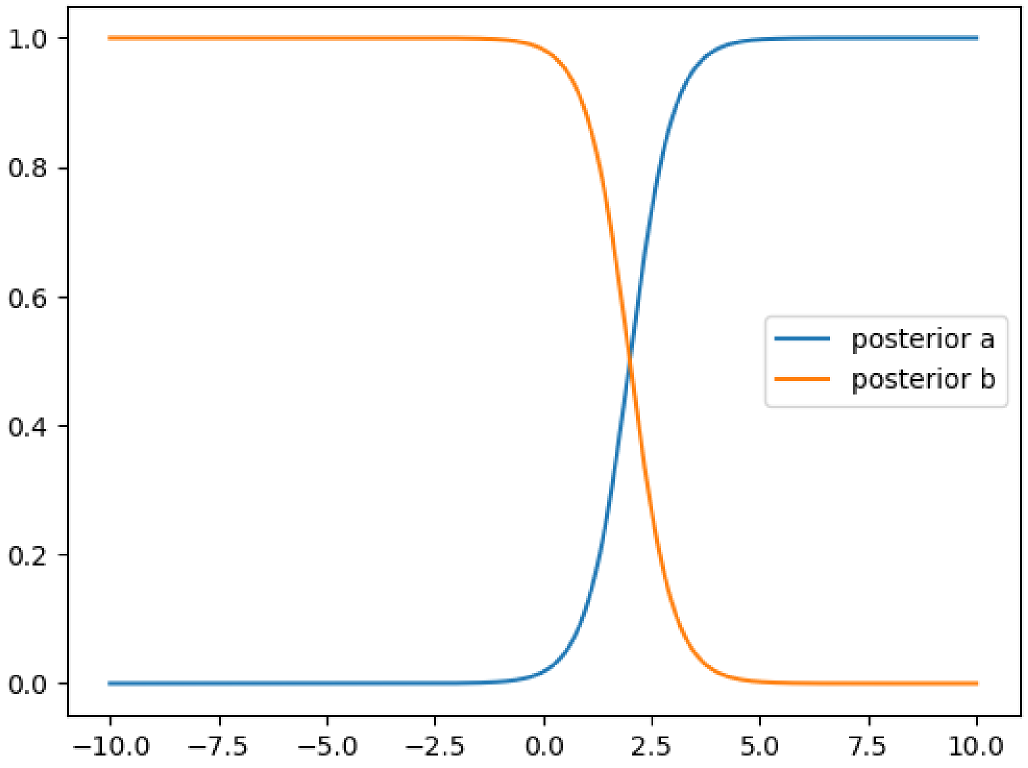
$$\alpha\sum_{i=1}^{N} y_i^2 = \sum_{i=1}^{N} y_i y_{i+1} \Rightarrow \alpha = \frac{\sum_{i=1}^{N} y_i y_{i+1}}{\sum_{i=1}^{N} y_i^2} \quad \#$$

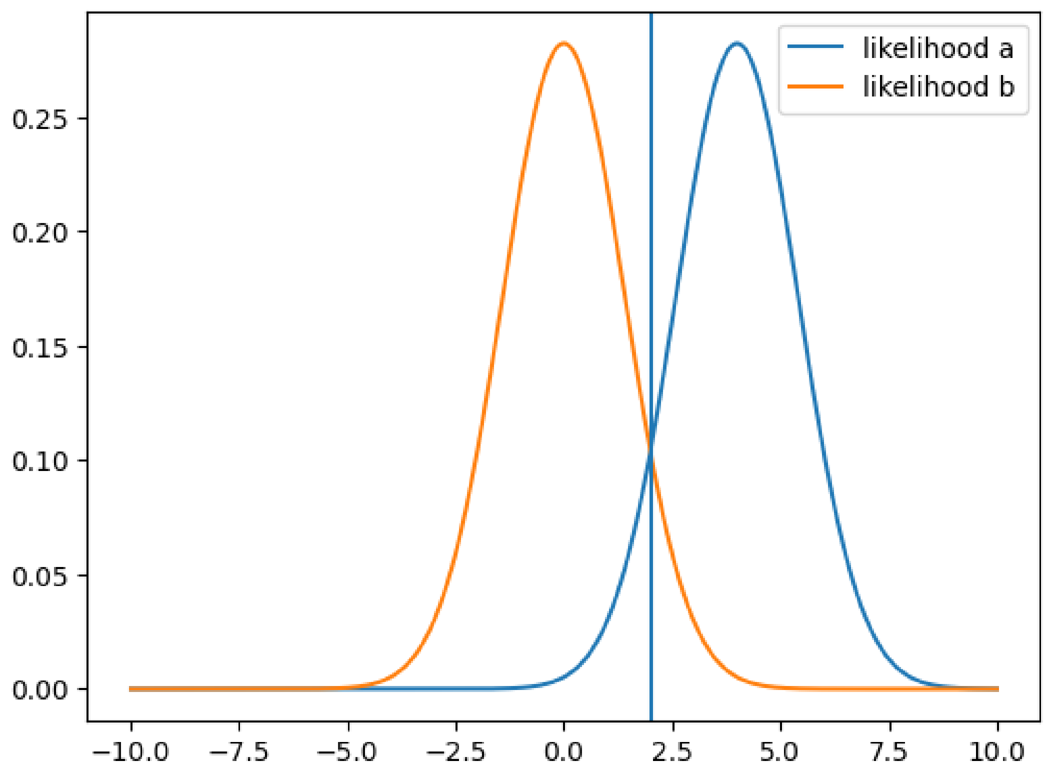
[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

[ ]:
```python
def find_decision_boundary(mean_a, sd_a, mean_b, sd_b, prior_a, prior_b):
    x_axis = np.arange(-10, 10, 0.0001)

    likelihood_a = norm.pdf(x_axis, loc=mean_a, scale=sd_a)
    likelihood_b = norm.pdf(x_axis, loc=mean_b, scale=sd_b)

    evidence = likelihood_a * prior_a + likelihood_b * prior_b

    plt.plot(x_axis, (likelihood_a / evidence) * prior_a, label="posterior a")
    plt.plot(x_axis, (likelihood_b / evidence) * prior_b, label="posterior b")
    plt.legend()
    plt.show()

    likelihood_ratio = likelihood_a / likelihood_b
    priors_ratio = prior_b / prior_a

    boundary_position = x_axis[likelihood_ratio >= priors_ratio][0]
    decision_boundary = likelihood_b[x_axis == boundary_position]

    plt.plot(x_axis, likelihood_a, label="likelihood a")
    plt.plot(x_axis, likelihood_b, label="likelihood b")
    plt.axvline(x=boundary_position)
    plt.legend()
    plt.show()
    return (boundary_position, decision_boundary[0])
```

0.3 T2

```
[ ]: print("Decision Boundary is at", find_decision_boundary(4, np.sqrt(2), 0, np.
     ↪sqrt(2), 0.5, 0.5))
```

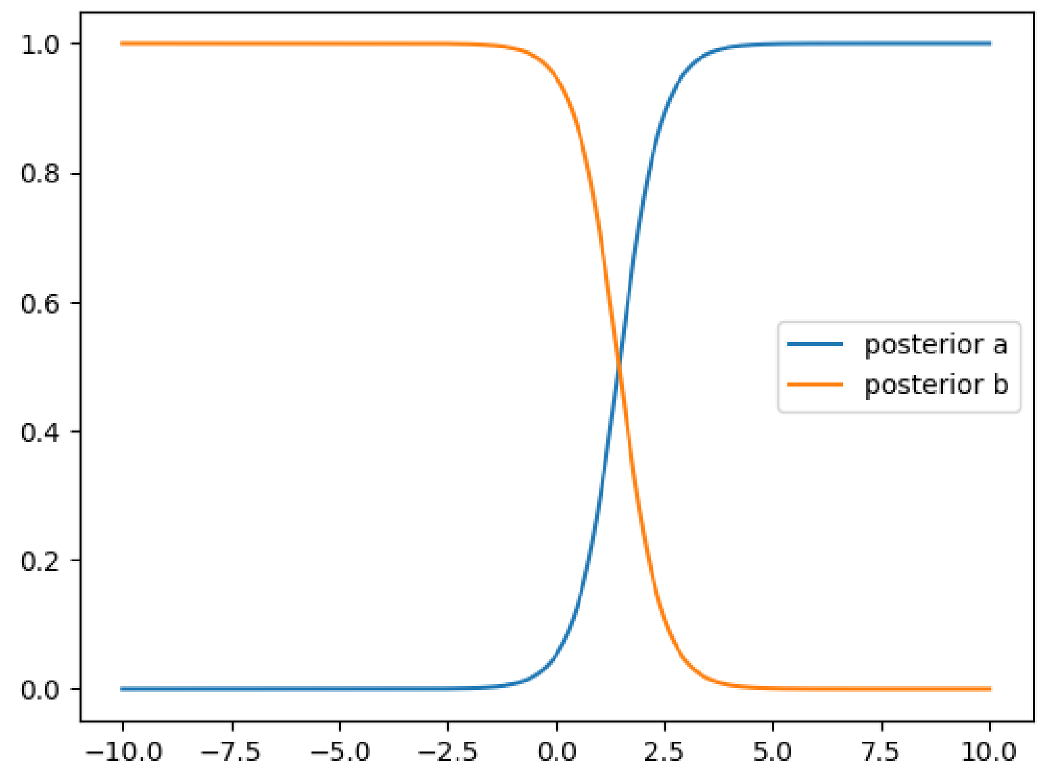




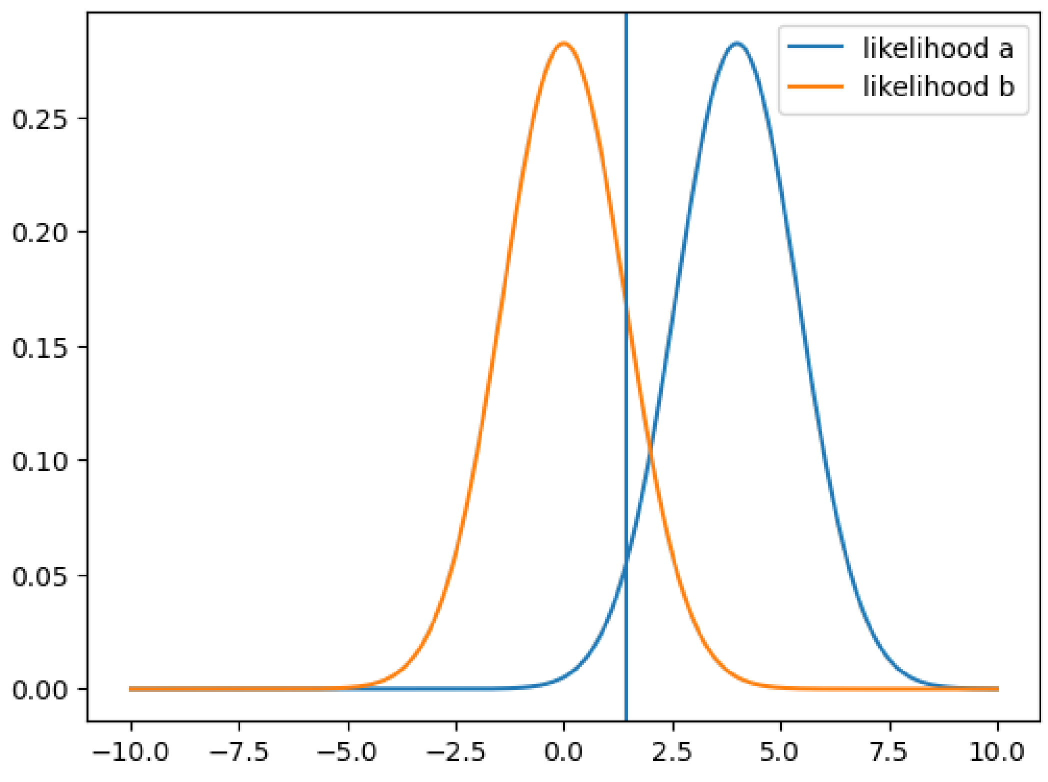
```
Decision Boundary is at (2.000099999972033, 0.10376649693006622)
```

0.4 T3

```
[ ]: print("Decision Boundary is at", find_decision_boundary(4, np.sqrt(2), 0, np.
      ↪sqrt(2), 0.75, 0.25))
```



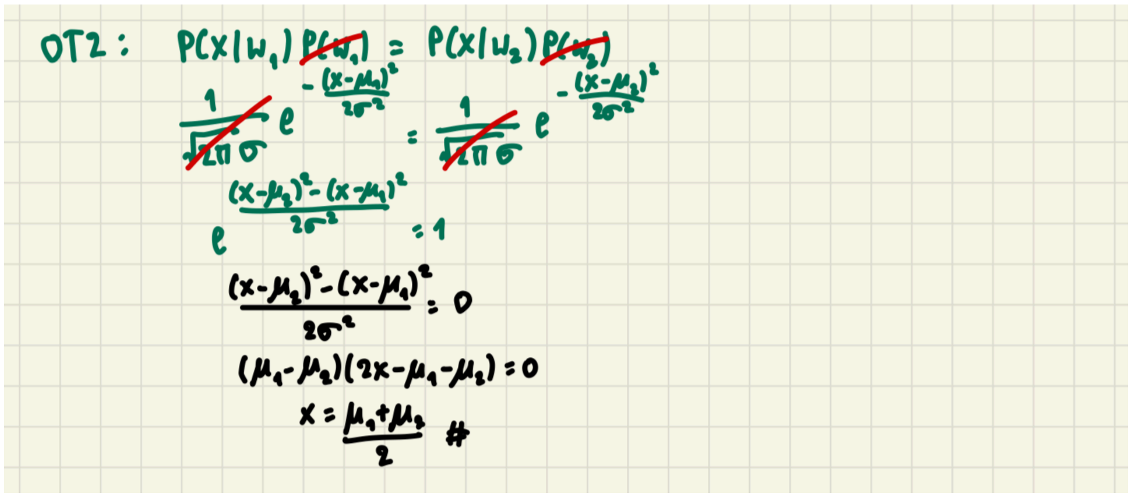
posterior a
posterior b



likelihood a
likelihood b

```
Decision Boundary is at (1.4506999999733132, 0.1666857883072887)
```

0.5 OT2



OT2 : $P(x|w_1)\cancel{P(w_1)} = P(x|w_2)\cancel{P(w_2)}$

$$\cancel{\frac{1}{\sqrt{2\pi}\sigma}} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}} = \cancel{\frac{1}{\sqrt{2\pi}\sigma}} e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}$$

$$e^{\frac{(x-\mu_2)^2-(x-\mu_1)^2}{2\sigma^2}} = 1$$

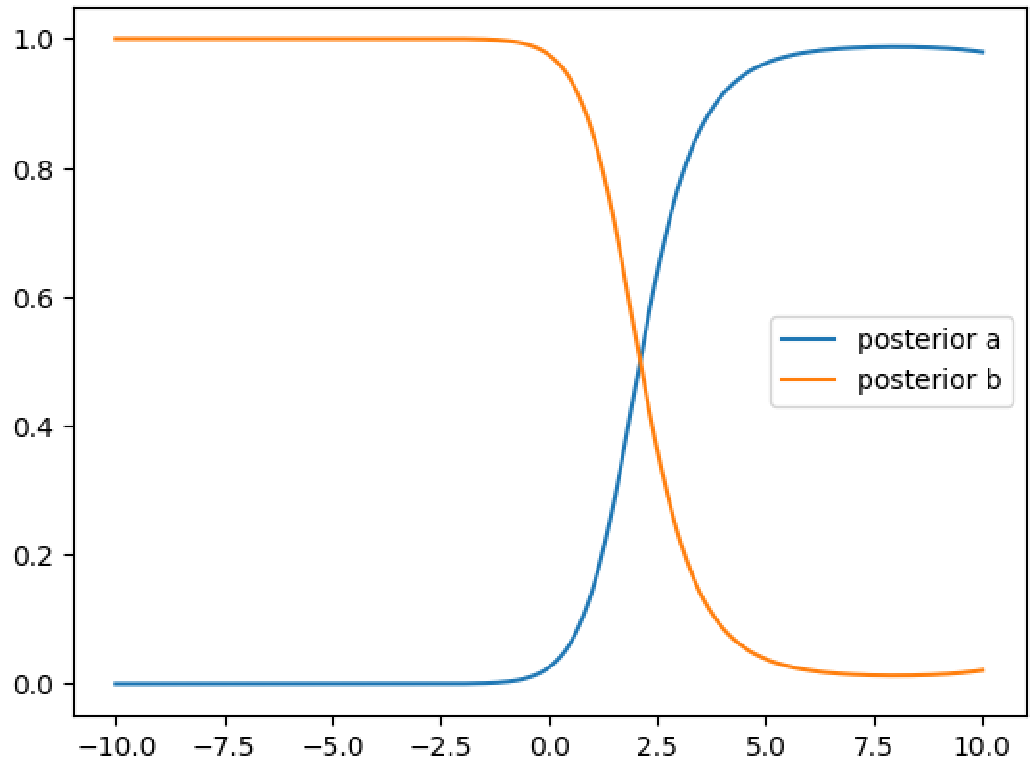
$$\frac{(x-\mu_2)^2-(x-\mu_1)^2}{2\sigma^2} = 0$$

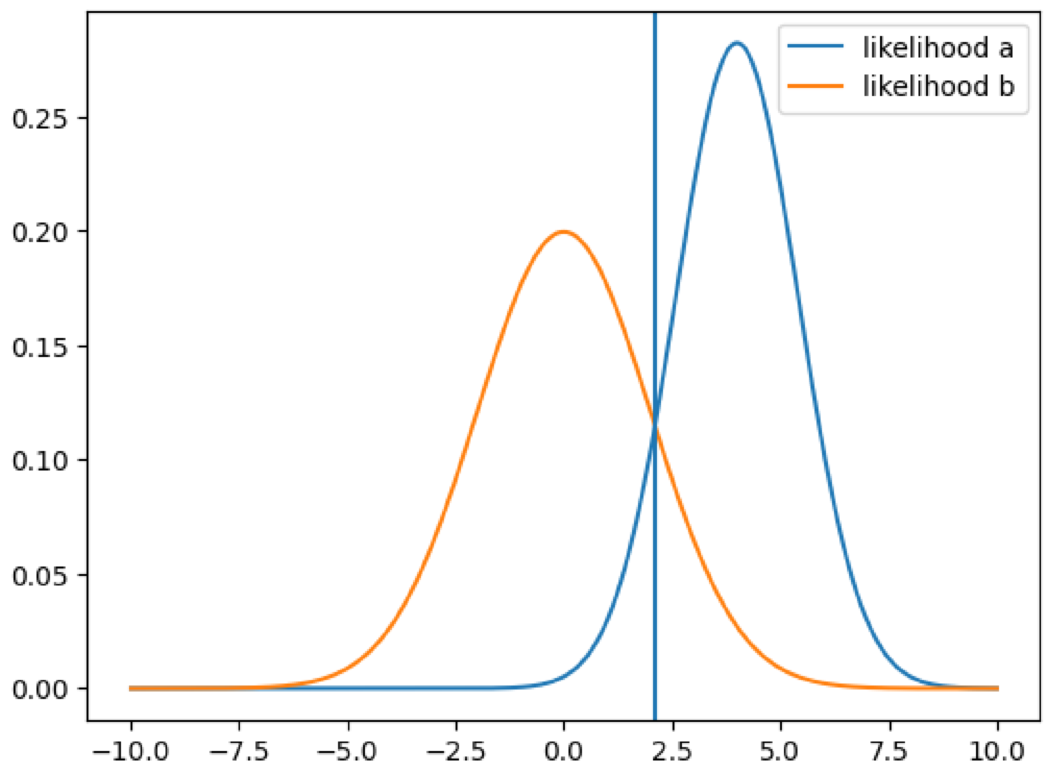
$$(\mu_1-\mu_2)(2x-\mu_1-\mu_2) = 0$$

$$x = \frac{\mu_1+\mu_2}{2} \quad \#$$

0.6 OT3

```
[ ]: print("Decision Boundary:", find_decision_boundary(4, np.sqrt(2), 0, 2, 0.5, 0.
     ↪5))
```





```
Decision Boundary: (2.1031999999717925, 0.11474798458217128)
```

1 Employee Attrition Prediction

1.0.1 read CSV

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

1.0.2 Dataset statistic

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

```
[ ]:        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]
```

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

[ ]:
```
   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

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

     # HANDLE NULL NUMBERS
     # df[np.isnan(df)] = -1
     # INSERT CODE HERE

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

1.0.4 Spliting data into train and test

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

```
[ ]: df_train, df_test = train_test_split(df, test_size=0.1, train_size=0.9,␣
      ↪stratify=df["Attrition"])
```

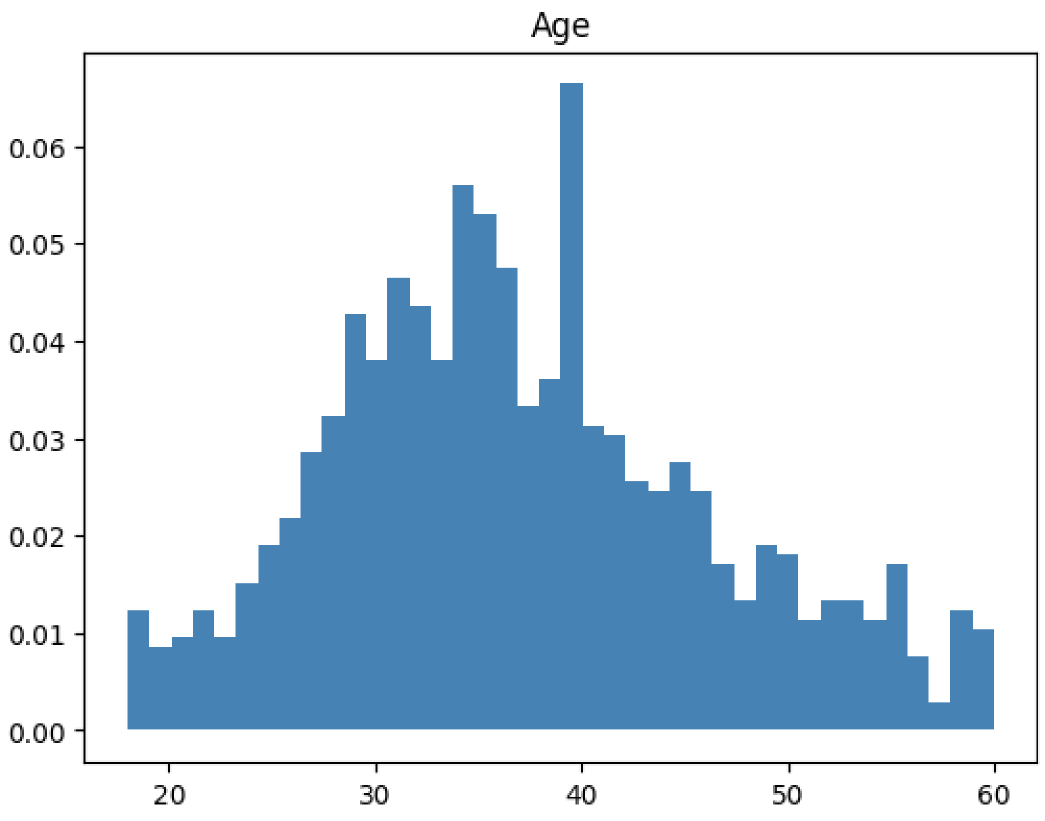
1.0.5 Display histogram of each feature

```
[ ]: def display_histogram(df, col_name, cls, n_bin = 40):
         train_col_no_nan = df[col_name][~(np.isnan(df[col_name]))]
         hist, bin_edge = np.histogram(train_col_no_nan, n_bin)

         hist = np.array(hist, dtype=float) / np.sum(hist)

         plt.fill_between(bin_edge.repeat(2)[1:-1], hist.repeat(2),␣
      ↪facecolor="steelblue")
         plt.title(cls)
         plt.show()

         print("Zero bins:", np.sum(hist == 0))
```

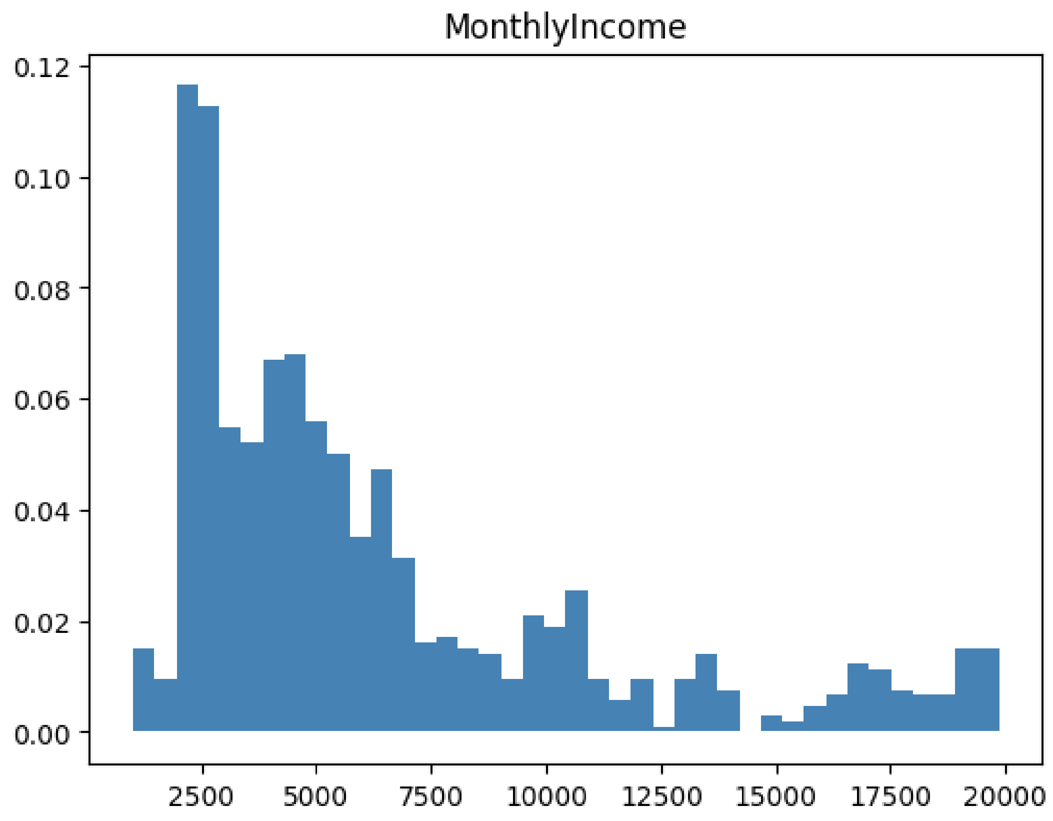
```
    # INSERT CODE HERE
```

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?

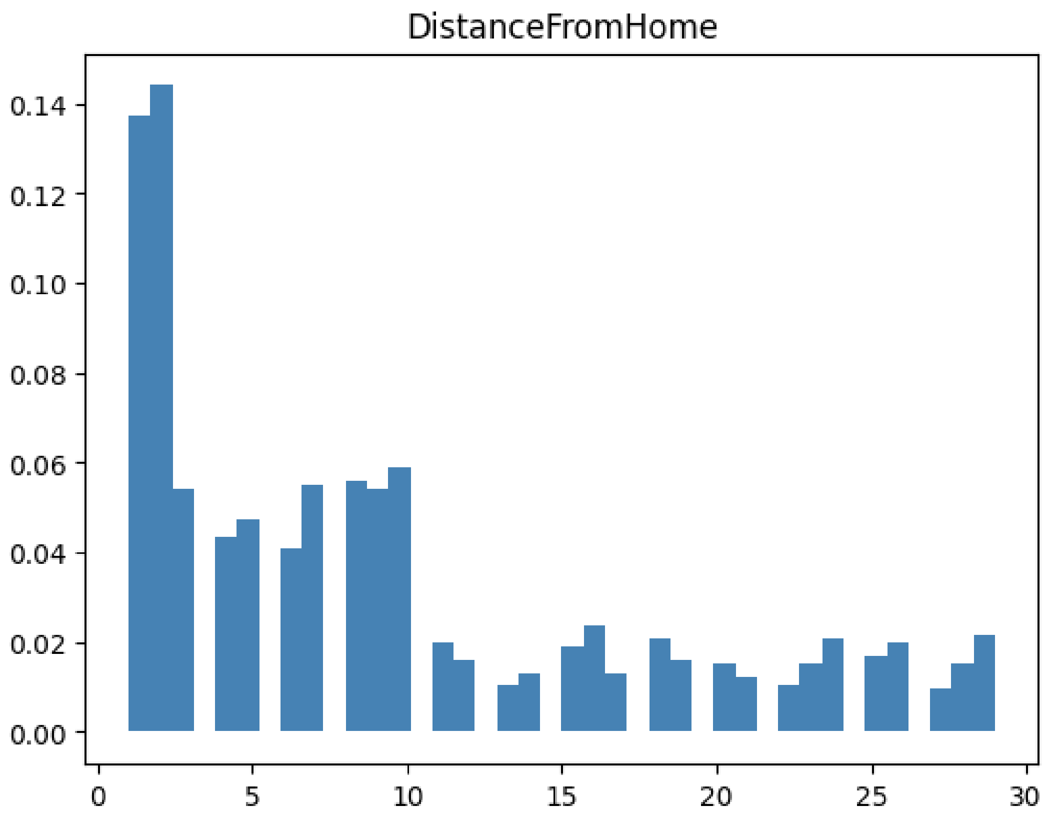
[ ]:
```
display_histogram(df_train, "Age", "Age", 40)
display_histogram(df_train, "MonthlyIncome", "MonthlyIncome", 40)
display_histogram(df_train, "DistanceFromHome", "DistanceFromHome", 40)
```



```
Zero bins: 0
```


MonthlyIncome
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2500
5000
7500
10000
12500
15000
17500
20000

```
Zero bins: 1
```



```
Zero bins: 11
```

- `Age` has 0 zero bin
- `MonthlyIncome` has 0 zero bin
- `DistanceFromHome` has 11 zero bins

Feature `DistanceFromHome` does not get a good discretization with bin size 40 because it appears zero bins in the histogram even though `Age` and `MonthlyIncome` gets a smooth histogram

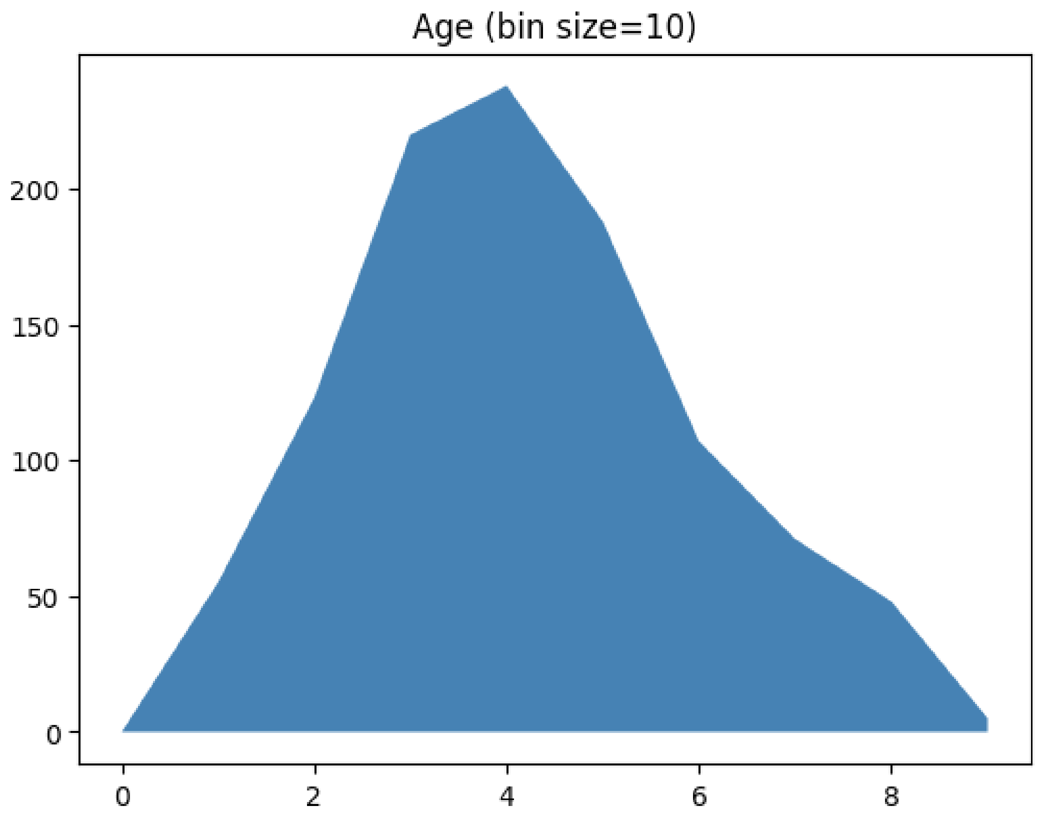
1.0.7 T5. Can we use a Gaussian to estimate this histogram? Why? What about a Gaussian Mixture Model (GMM)?

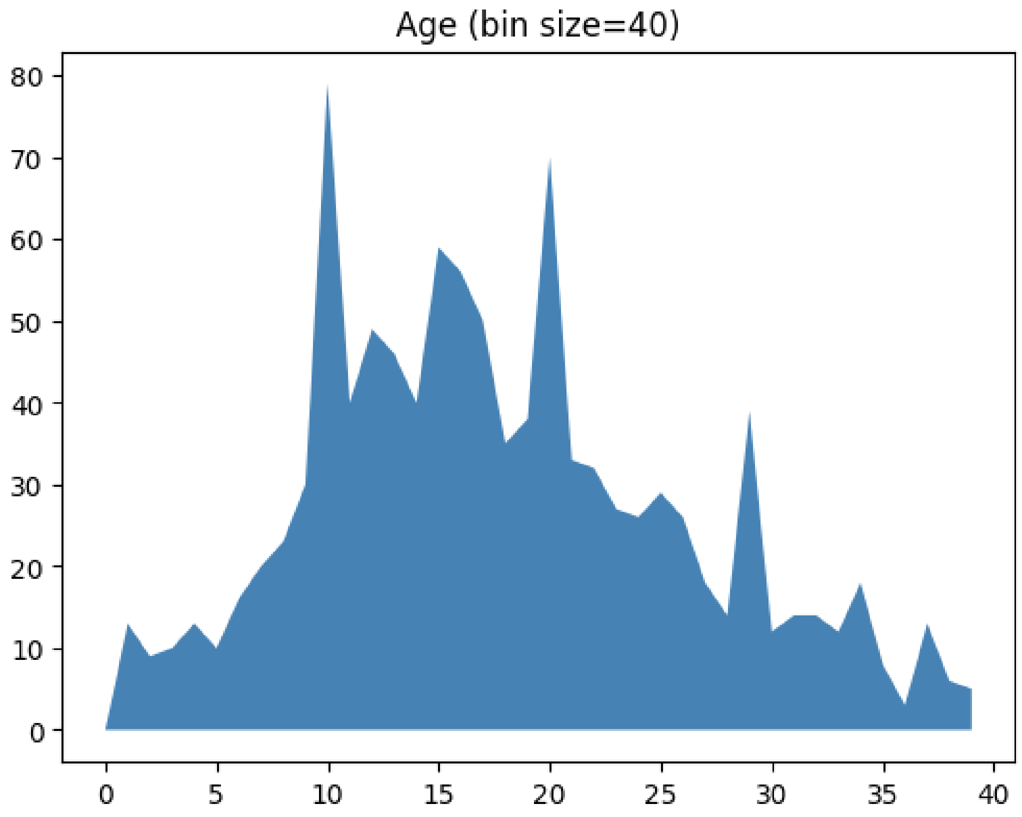
We are likely able to use Gaussian to estimate `Age` because the shape does not look like a normal gaussian distribution but for `MonthlyIncome`, it looks like two gaussian distribution in histogram which can be formulated as Gaussian Mixture Model but it's not likely to be used on `DistanceFromHome` because its shape look like logarithm graph instead

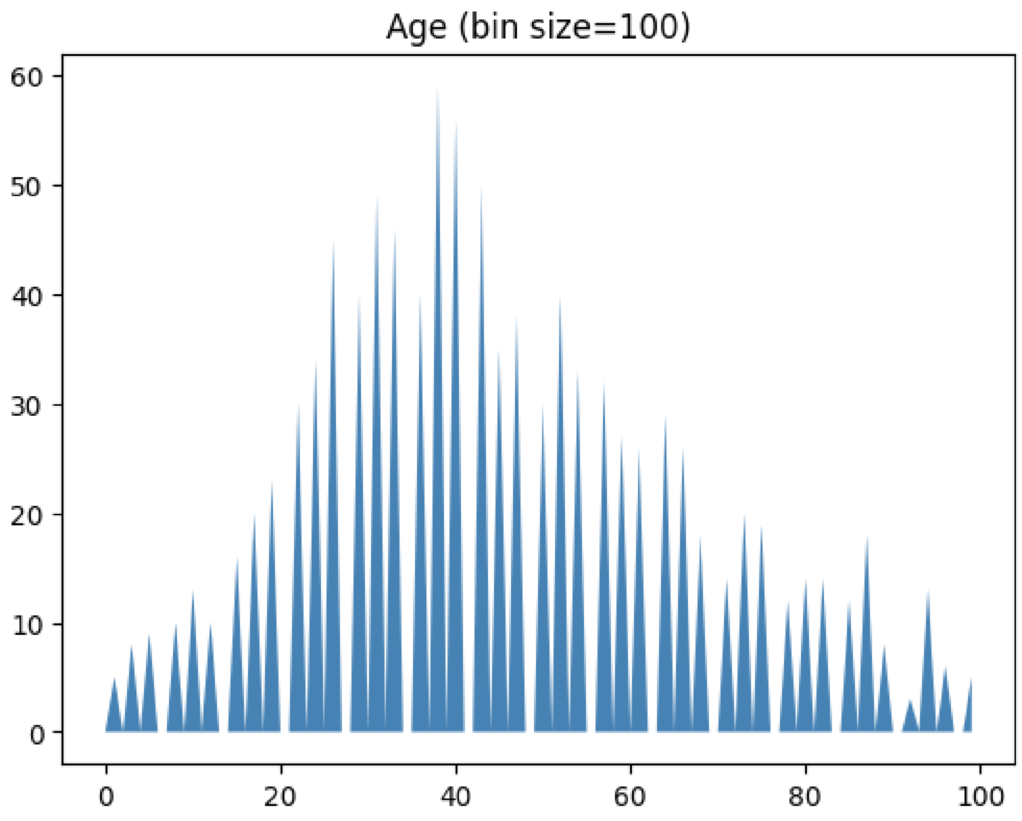
1.0.8 T6. Now plot the histogram according to the method described above (with 10, 40, and 100 bins) and show 3 plots each for Age, MonthlyIncome, and DistanceFromHome. Which bin size is most sensible for each features? Why?

[ ]:
```
def calculate_bins_and_edges(arr, n_bin = 10):
    min_bin = -1e9
    max_bin = 1e9

    max_train = np.max(arr)
    min_train = np.min(arr)

    edges = np.linspace(start=min_train, stop=max_train, num=n_bin - 1)
    edges = np.insert(edges, 0, min_bin)
    edges = np.append(edges, max_bin)

    digitized_df = np.digitize(arr, edges)
    bins = np.bincount(digitized_df)[1:]

    return bins, edges

def plot_histogram_v2(df, col_name, n_bin = 10):
    train_col_no_nan = df[col_name][~(np.isnan(df[col_name]))]
    bins, _ = calculate_bins_and_edges(train_col_no_nan, n_bin)

    plt.fill_between(np.arange(0, n_bin), bins, facecolor="steelblue")
    plt.title(f"{col_name} (bin size={n_bin})")
    plt.show()
```

[ ]:
```
plot_histogram_v2(df_train, "Age", 10)
plot_histogram_v2(df_train, "Age", 40)
plot_histogram_v2(df_train, "Age", 100)
```

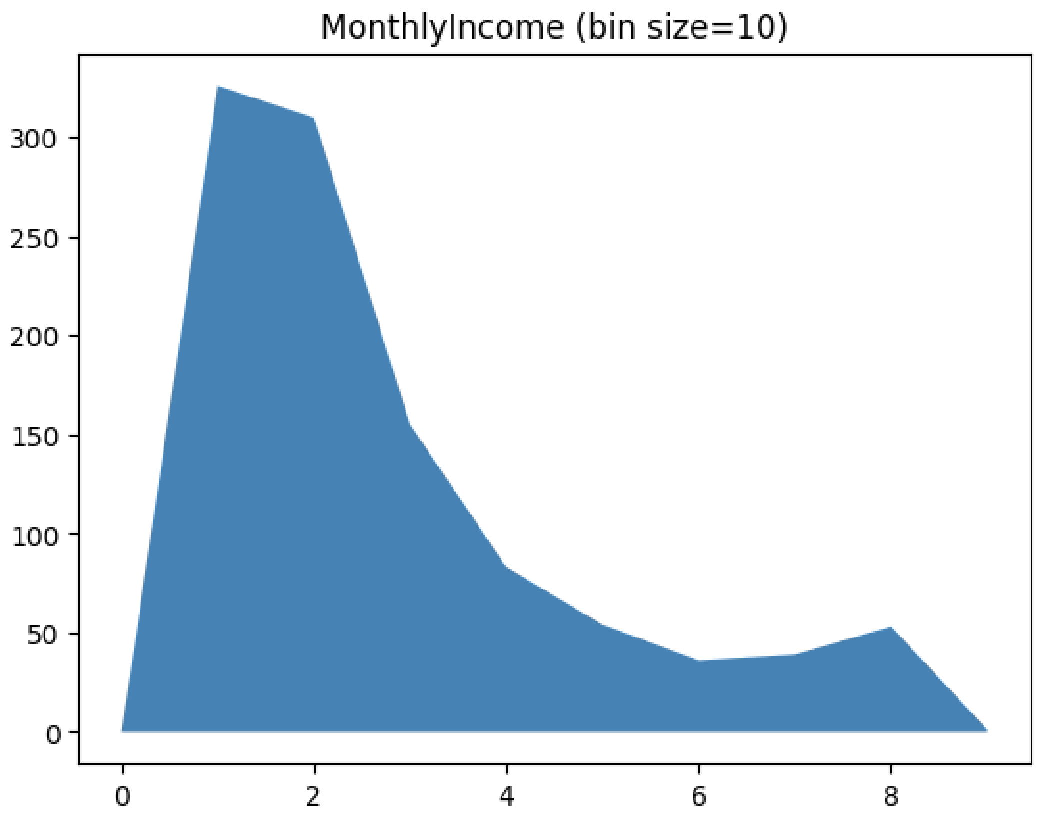

Age (bin size=10)
200
150
100
50
0
0
2
4
6
8

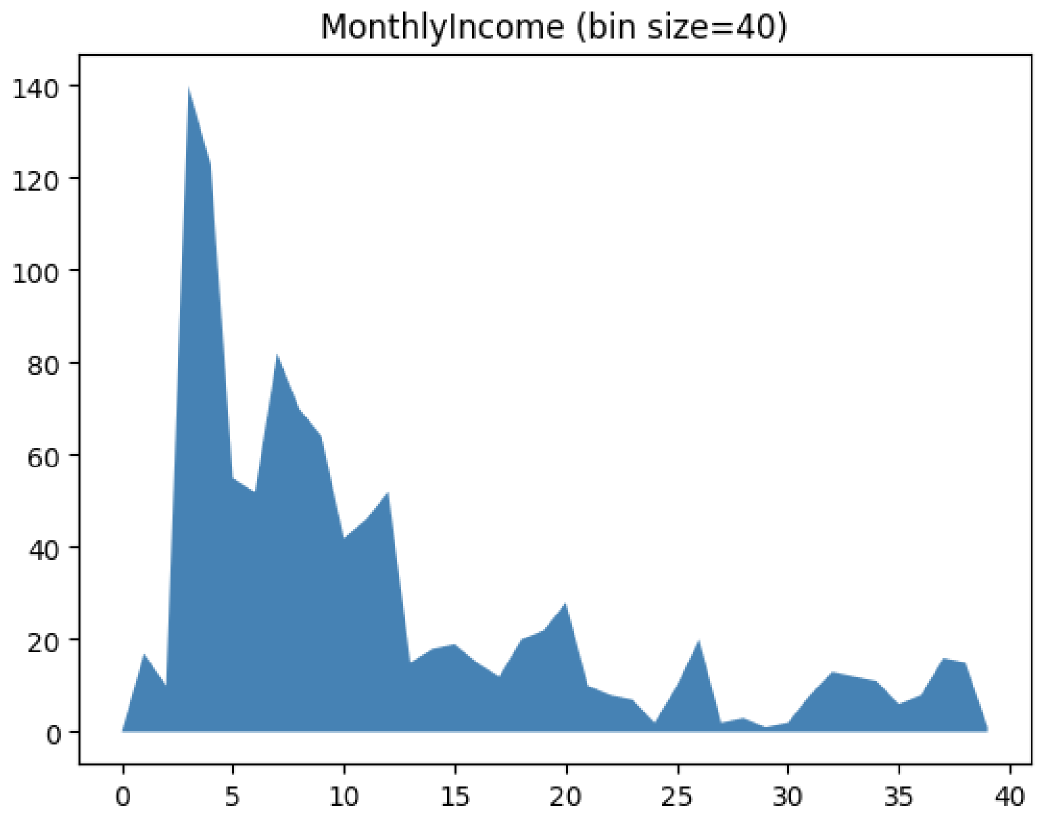

Age (bin size=40)
80
70
60
50
40
30
20
10
0
0
5
10
15
20
25
30
35
40

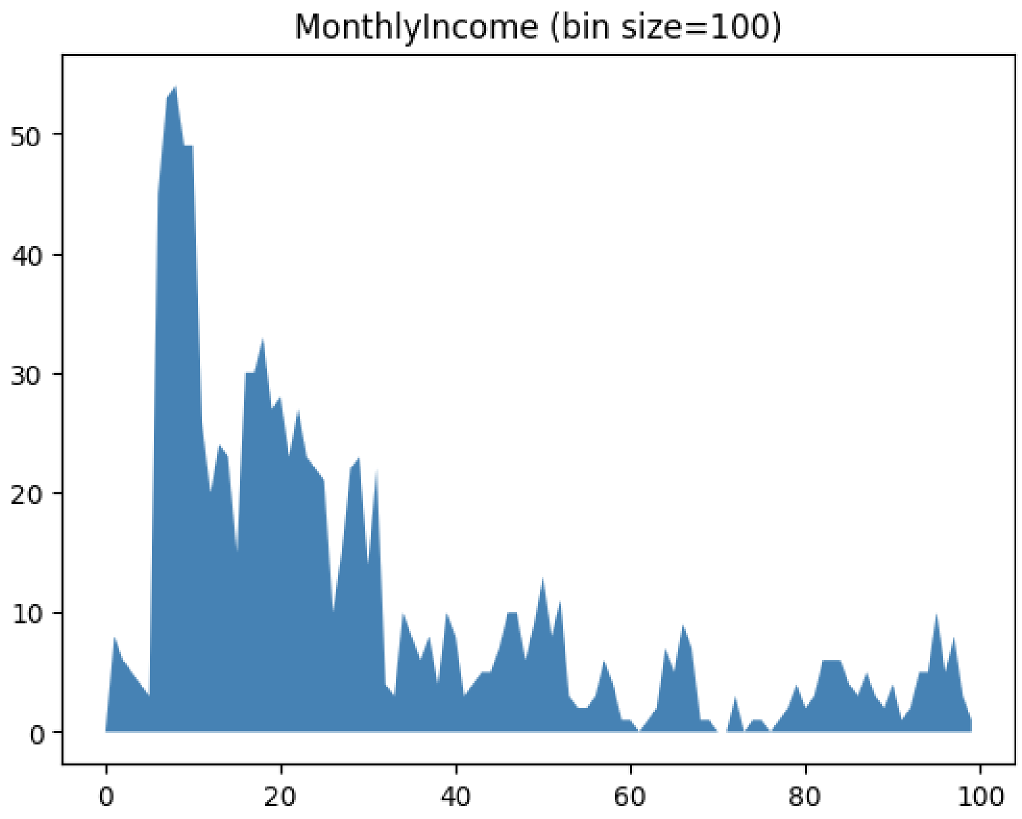


For age feature, we will see that when bin size is 100 there start to have zero bins appearing in histogram but it look like a normal distribution curve also

```
[ ]: plot_histogram_v2(df_train, "MonthlyIncome", 10)
     plot_histogram_v2(df_train, "MonthlyIncome", 40)
     plot_histogram_v2(df_train, "MonthlyIncome", 100)
```

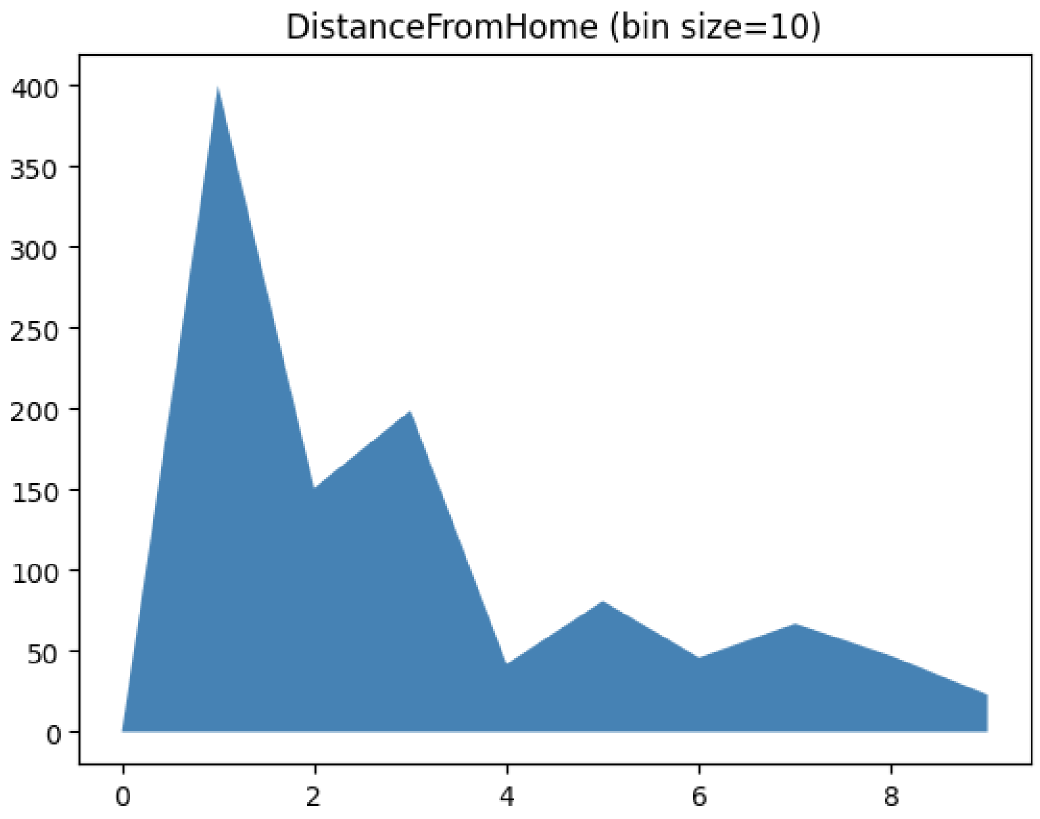

MonthlyIncome (bin size=10)
300
250
200
150
100
50
0
0
2
4
6
8

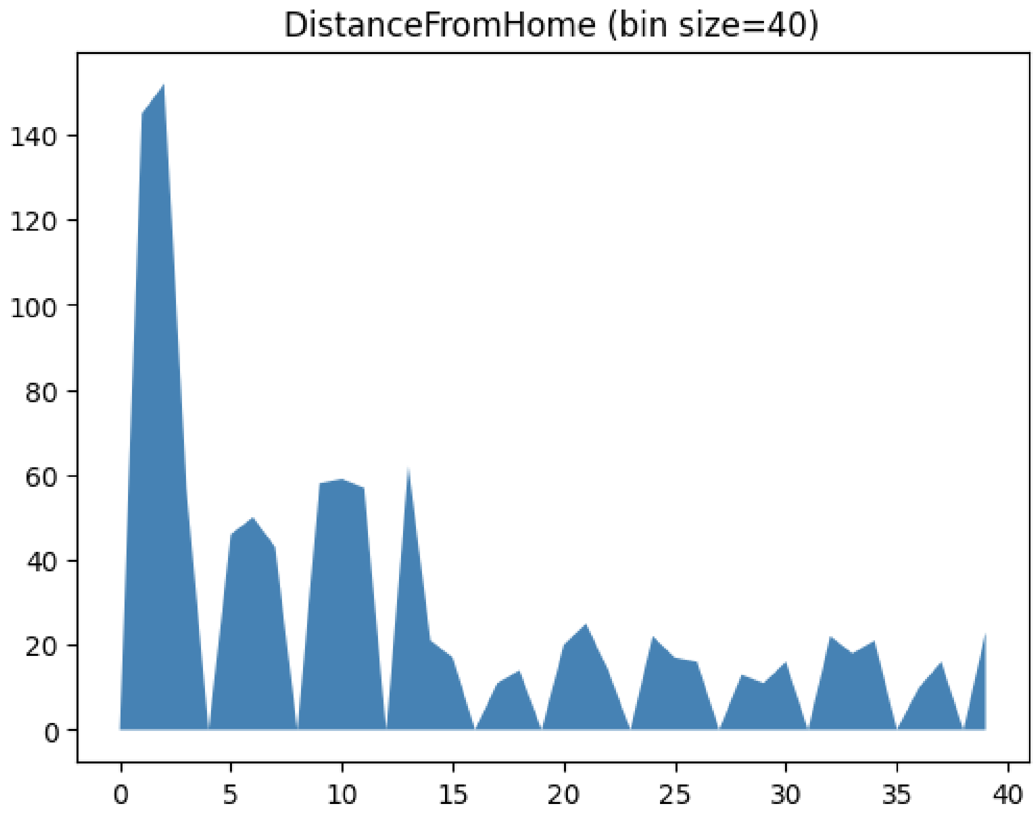

MonthlyIncome (bin size=40)
140
120
100
80
60
40
20
0
0
5
10
15
20
25
30
35
40

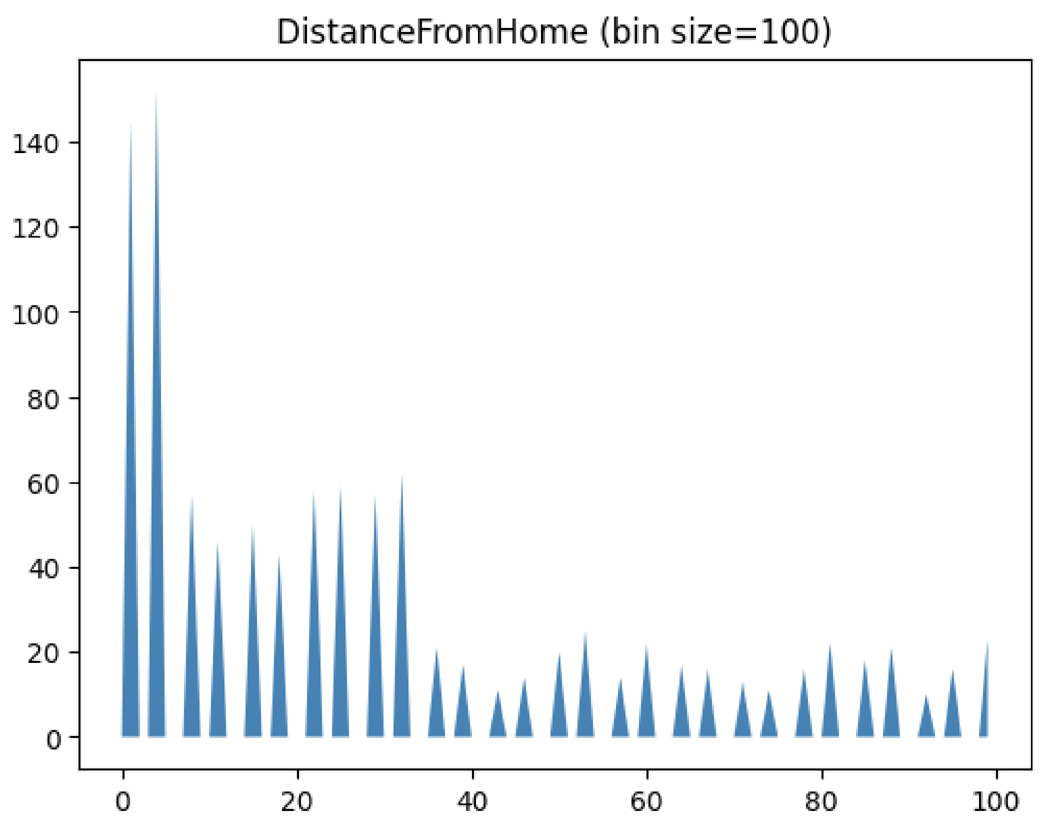


For MonthlyIncome feature, we will see that the sensible point start at bin size 100 which start to see the zero bins that should not appear in probabilistic estimation

[ ]:
```
plot_histogram_v2(df_train, "DistanceFromHome", 10)
plot_histogram_v2(df_train, "DistanceFromHome", 40)
plot_histogram_v2(df_train, "DistanceFromHome", 100)
```


DistanceFromHome (bin size=10)
400
350
300
250
200
150
100
50
0
0
2
4
6
8


DistanceFromHome (bin size=40)
0
20
40
60
80
100
120
140
0
5
10
15
20
25
30
35
40



For DistanceFromHome feature, we will see that at least bin size 40 it seems to appear the zero bins in histogram

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

[ ]:
```
df
```

[ ]:
```
       Age  Attrition  BusinessTravel  DailyRate  Department  \
0     41.0          1               2        NaN          -1
1      NaN          0              -1      279.0           1
2     37.0          1              -1     1373.0          -1
3      NaN          0               1     1392.0           1
4     27.0          0               2      591.0           1
…      …          …               …          …          …
1465  36.0          0               1      884.0           1
1466  39.0          0               2      613.0          -1
1467  27.0          0              -1      155.0           1
```

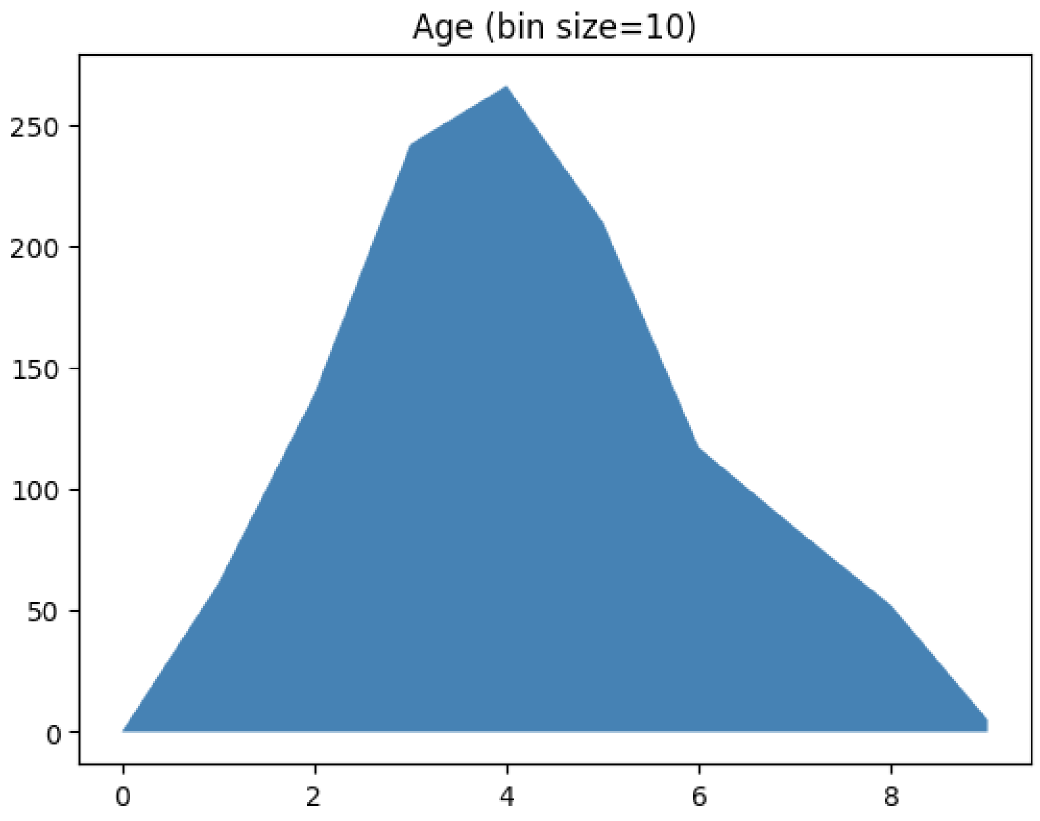
```
1468  49.0           0              1     1023.0           2
1469  34.0           0              2      628.0           1

      DistanceFromHome  Education  EducationField  EnvironmentSatisfaction  \
0                  1.0        NaN               1                      2.0
1                  NaN        NaN               1                      3.0
2                  2.0        2.0              -1                      NaN
3                  3.0        4.0               1                      NaN
4                  2.0        1.0               3                      1.0
…                 …        …              …                      …
1465               NaN        NaN               3                      3.0
1466               6.0        NaN               3                      4.0
1467               4.0        3.0               1                      2.0
1468               2.0        3.0              -1                      4.0
1469               NaN        NaN              -1                      2.0

      Gender  …  PerformanceRating  RelationshipSatisfaction  \
0          0  …                NaN                       1.0
1          1  …                NaN                       4.0
2          1  …                3.0                       NaN
3          0  …                3.0                       3.0
4          1  …                3.0                       4.0
…       …  …               …                        …
1465       1  …                3.0                       3.0
1466      -1  …                3.0                       NaN
1467       1  …                NaN                       2.0
1468       1  …                3.0                       4.0
1469       1  …                3.0                       NaN

      StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
0                  0.0                8.0                    0.0
1                  1.0               10.0                    NaN
2                  0.0                7.0                    3.0
3                  NaN                8.0                    3.0
4                  1.0                6.0                    NaN
…                 …                 …                    …
1465               1.0               17.0                    3.0
1466               NaN                9.0                    5.0
1467               1.0                6.0                    0.0
1468               0.0               17.0                    NaN
1469               0.0                6.0                    3.0

      WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
0                 NaN             6.0                 NaN
1                 3.0            10.0                 NaN
2                 NaN             NaN                 0.0
3                 NaN             8.0                 NaN
```

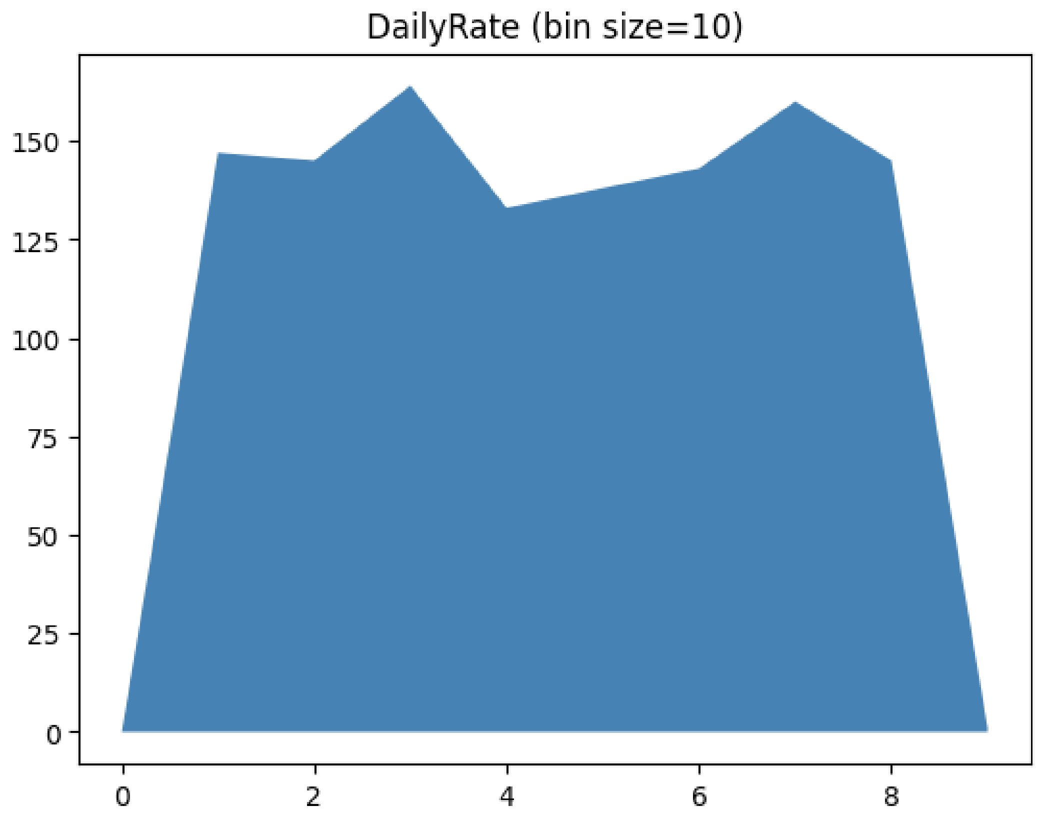
```
4                 3.0             2.0                 2.0
…               …             …                 …
1465              3.0             5.0                 2.0
1466              3.0             7.0                 7.0
1467              3.0             6.0                 NaN
1468              2.0             9.0                 6.0
1469              4.0             4.0                 NaN

      YearsSinceLastPromotion  YearsWithCurrManager
0                         0.0                   NaN
1                         NaN                   7.0
2                         NaN                   0.0
3                         3.0                   0.0
4                         2.0                   NaN
…                       …                   …
1465                      0.0                   3.0
1466                      1.0                   7.0
1467                      0.0                   3.0
1468                      0.0                   8.0
1469                      1.0                   2.0

[1470 rows x 31 columns]
```

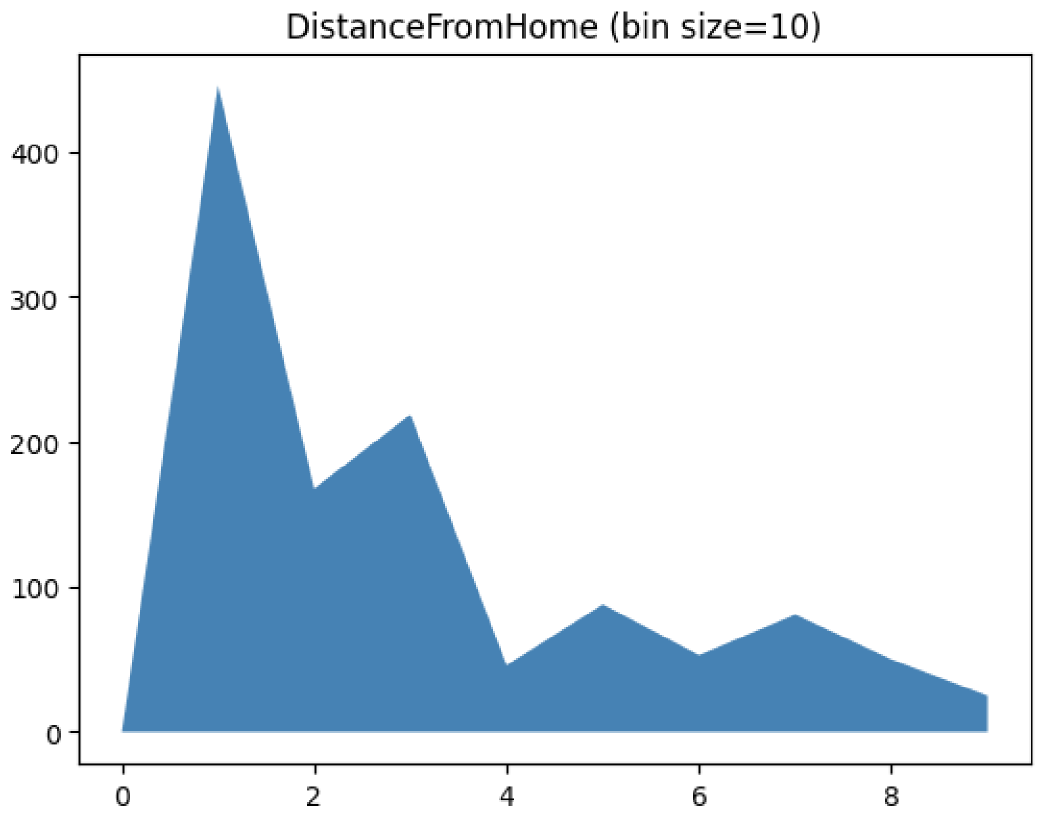
I will use the continuity of histogram as a criteria for considering which features should be discretized

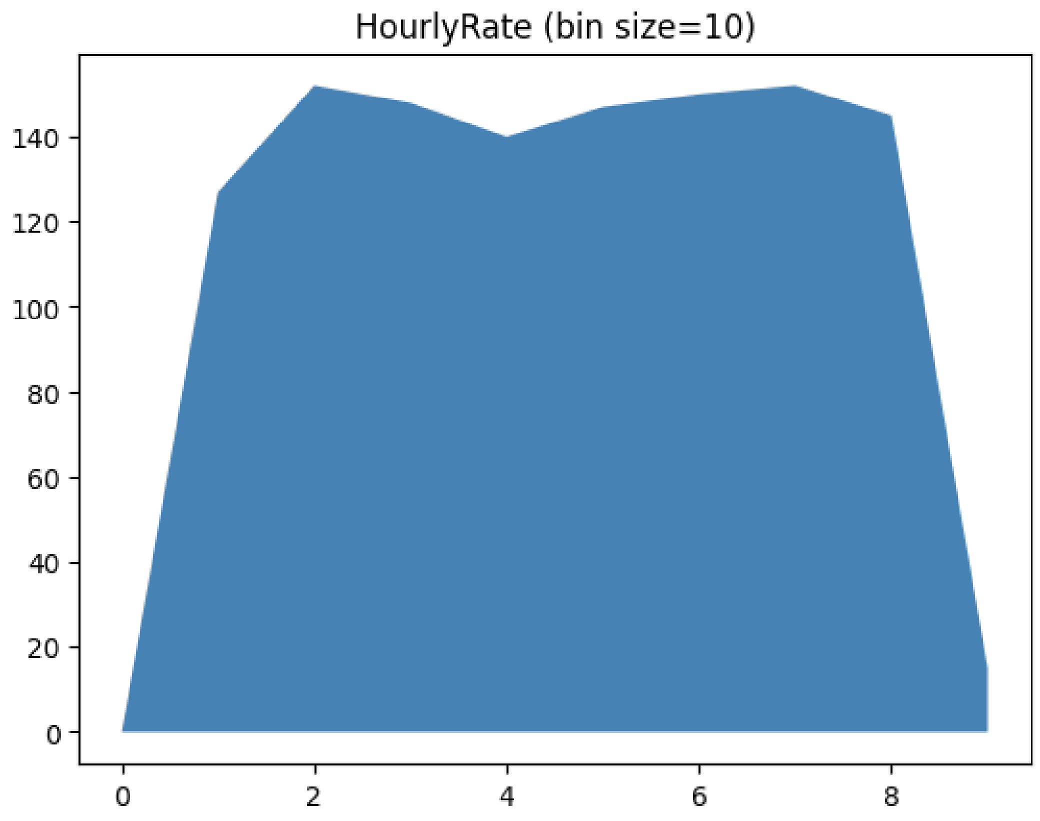
- Age
- DailyRate
- DistanceFromHome
- HourlyRate
- JobRole
- MonthlyIncome
- MonthlyRate
- NumCompaniesWorked
- PercentSalaryHike
- TotalWorkingYears
- TrainingTimesLastYear
- YearsAtCompany
- YearsInCurrentRole
- YearsSinceLastPromotion
- YearsWithCurrManager

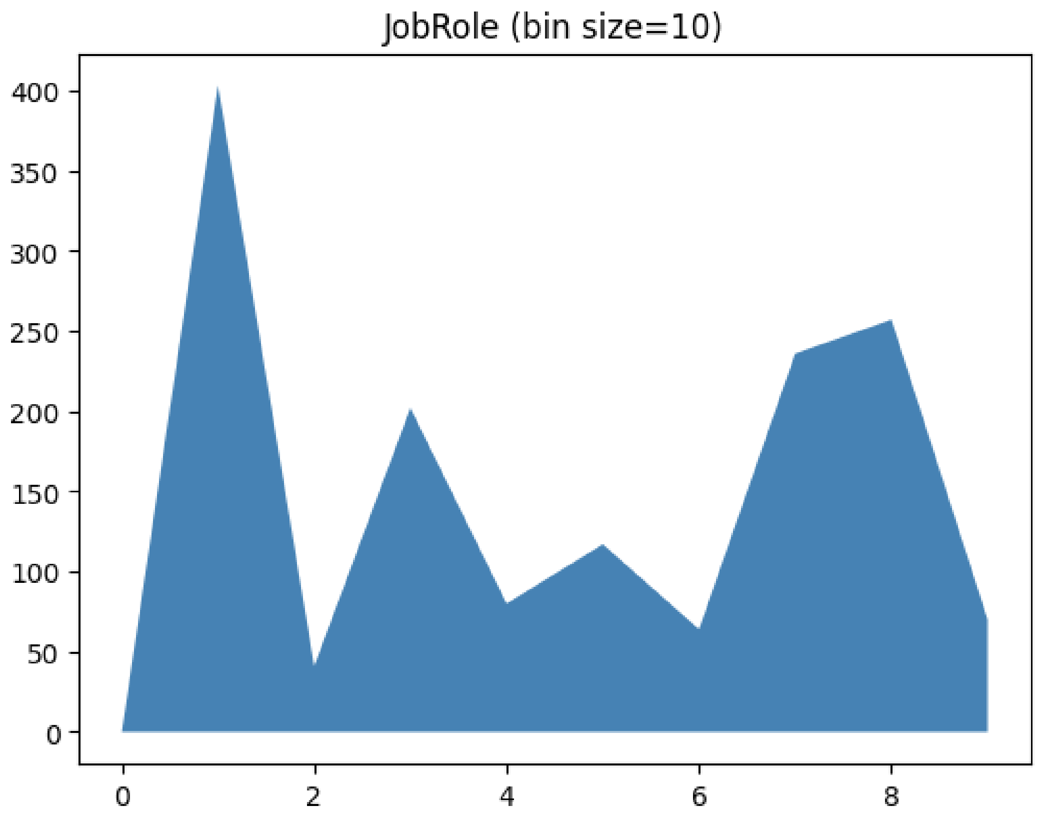
```
[ ]: column_name = ["Age", "DailyRate", "DistanceFromHome", "HourlyRate", "JobRole",␣
      ↪"MonthlyIncome", "MonthlyRate", "NumCompaniesWorked", "PercentSalaryHike",␣
      ↪"TotalWorkingYears", "TrainingTimesLastYear", "YearsAtCompany",␣
      ↪"YearsInCurrentRole", "YearsSinceLastPromotion", "YearsWithCurrManager"]

     for col in column_name:
         plot_histogram_v2(df, col, 10)
```

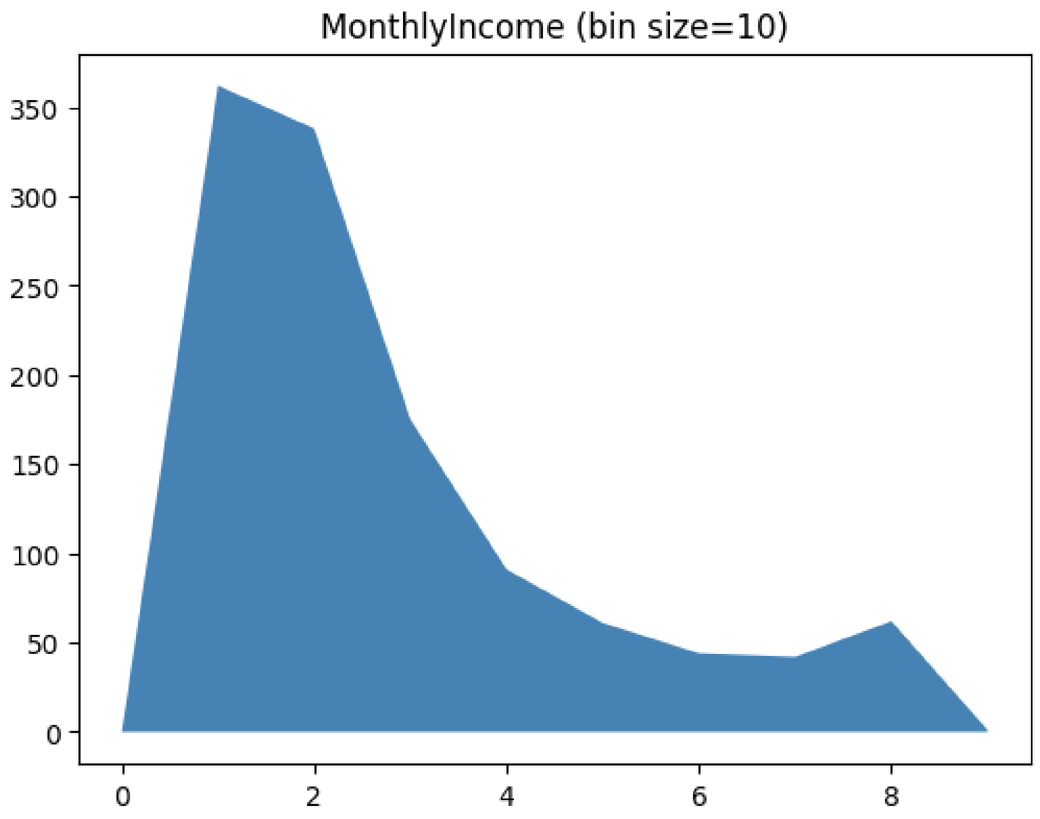

Age (bin size=10)
250
200
150
100
50
0
0
2
4
6
8

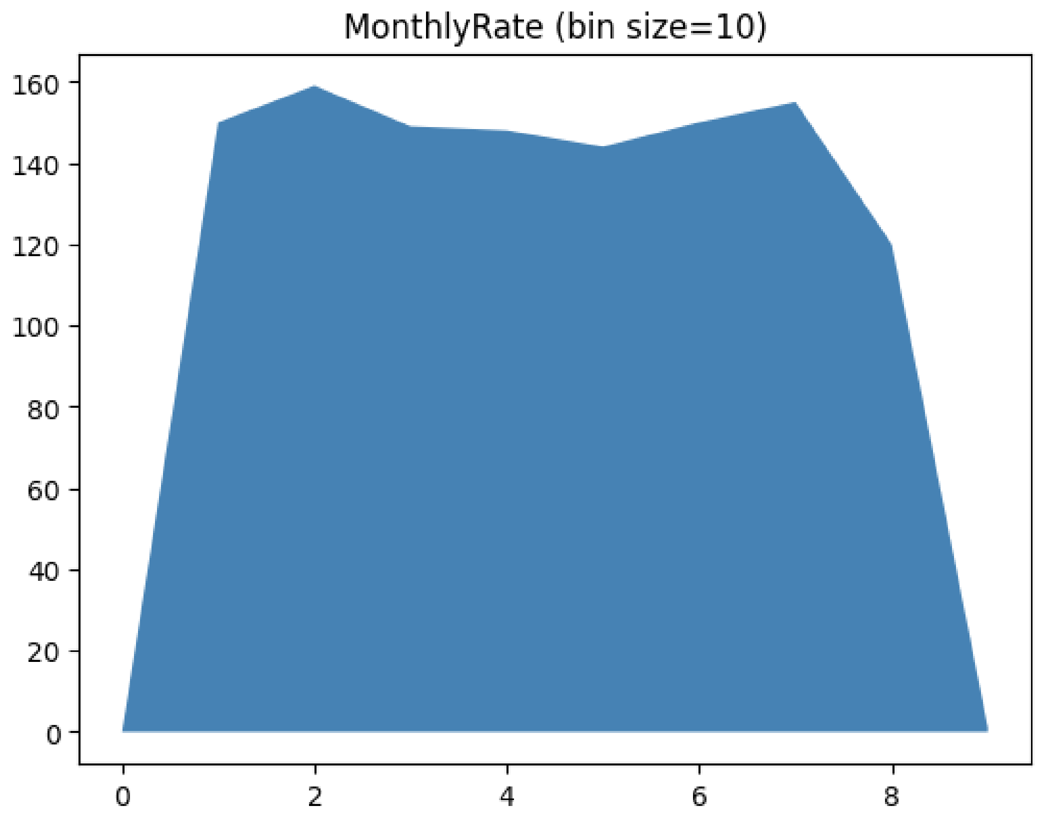

DailyRate (bin size=10)
150
125
100
75
50
25
0
0
2
4
6
8

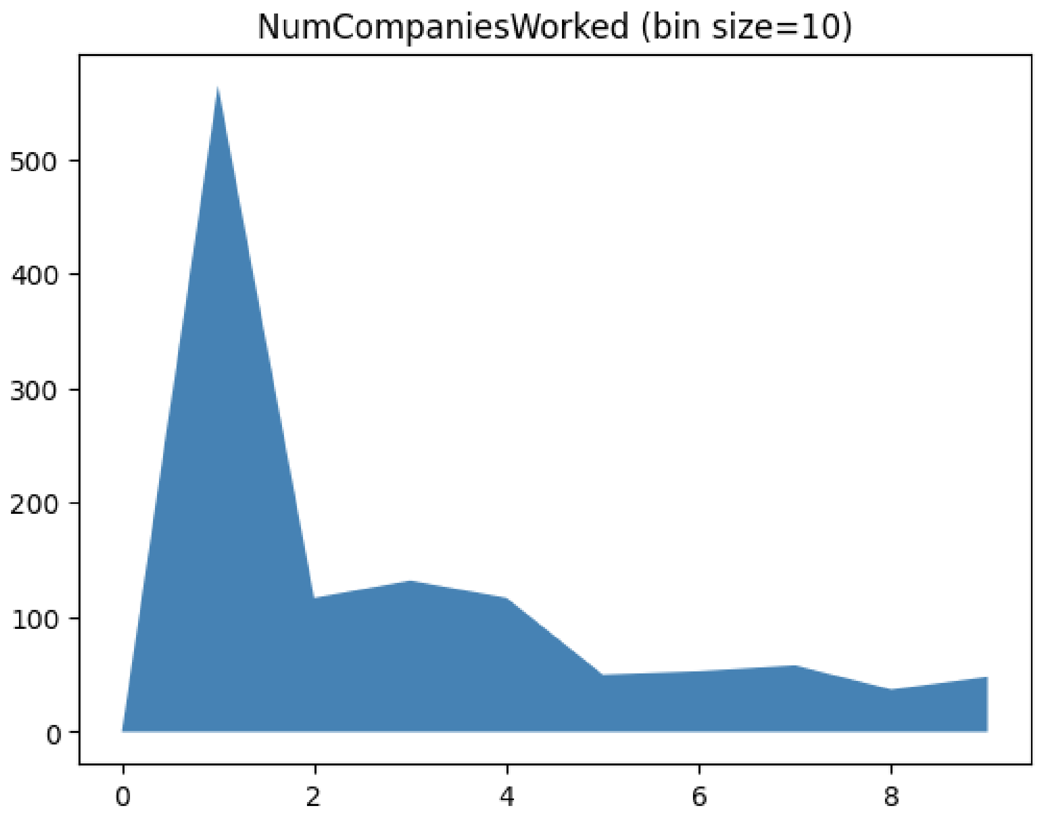

DistanceFromHome (bin size=10)
0
100
200
300
400
0
2
4
6
8

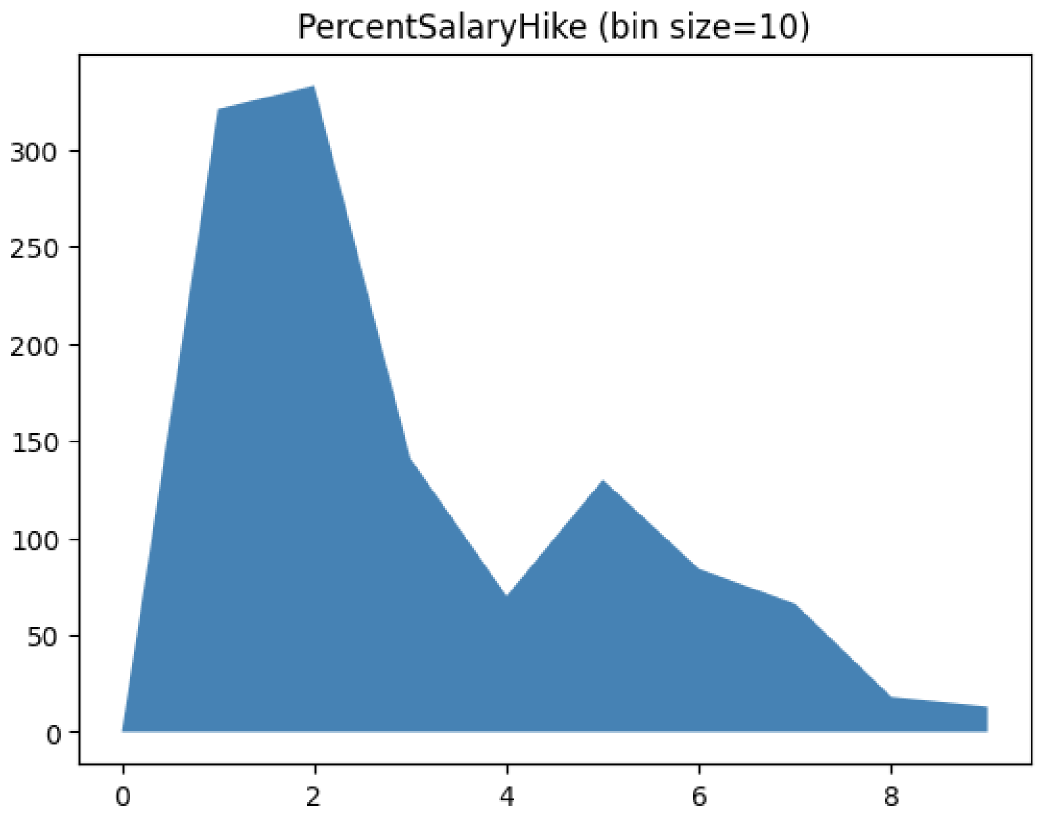

HourlyRate (bin size=10)
0
20
40
60
80
100
120
140
0
2
4
6
8

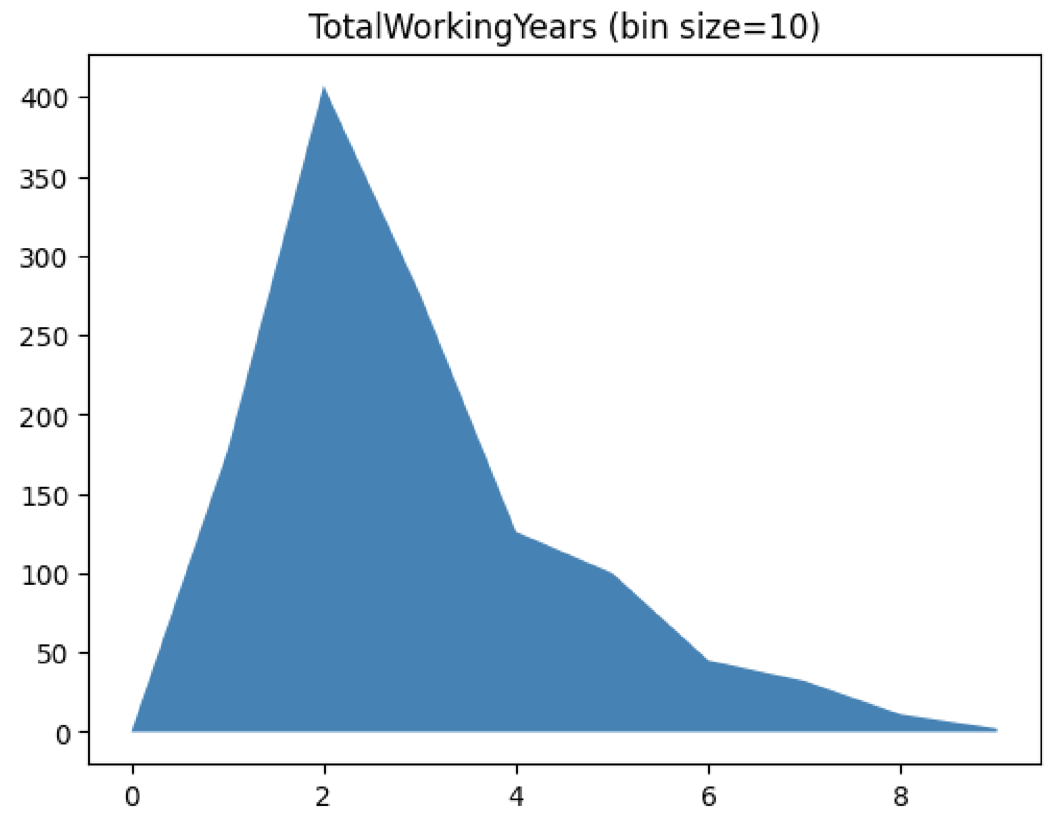

JobRole (bin size=10)
0
50
100
150
200
250
300
350
400
0
2
4
6
8


MonthlyIncome (bin size=10)
0
50
100
150
200
250
300
350
0
2
4
6
8

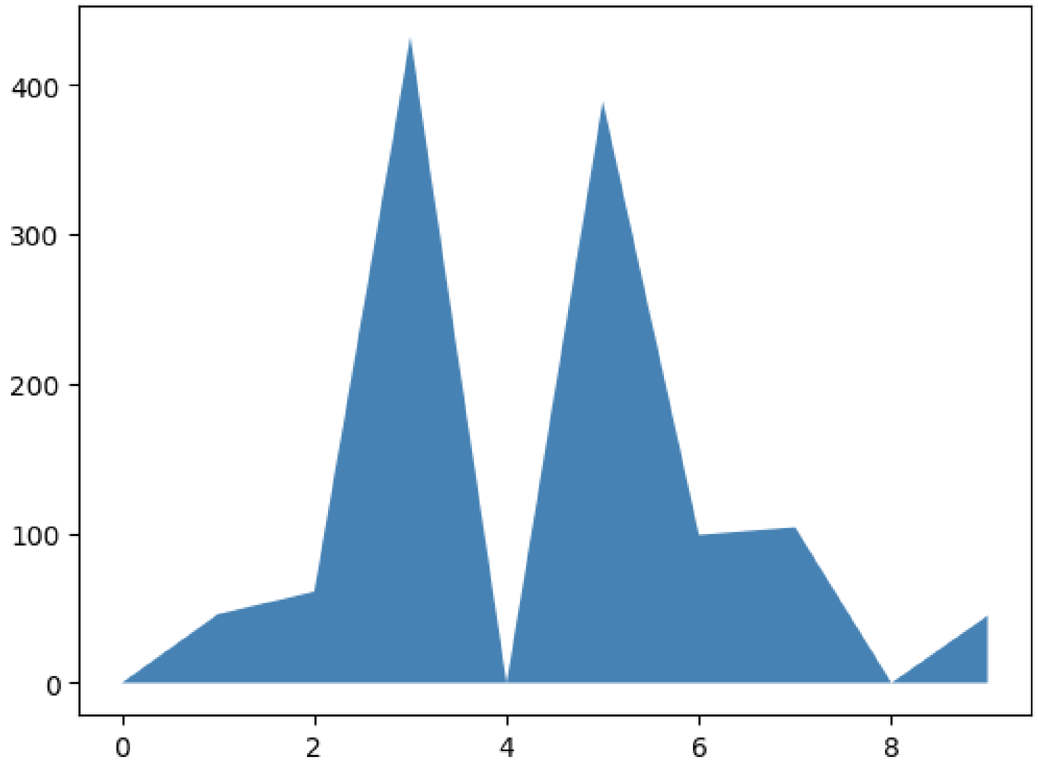

MonthlyRate (bin size=10)
0
20
40
60
80
100
120
140
160
0
2
4
6
8

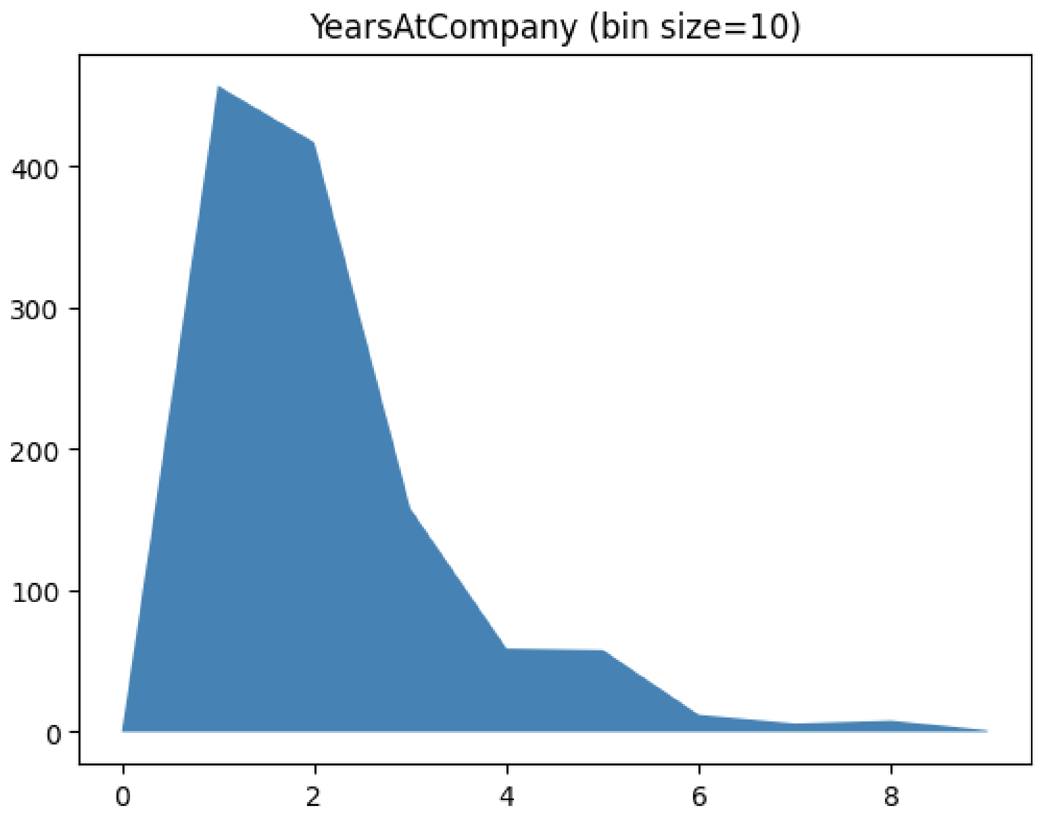

NumCompaniesWorked (bin size=10)
0
100
200
300
400
500
0
2
4
6
8

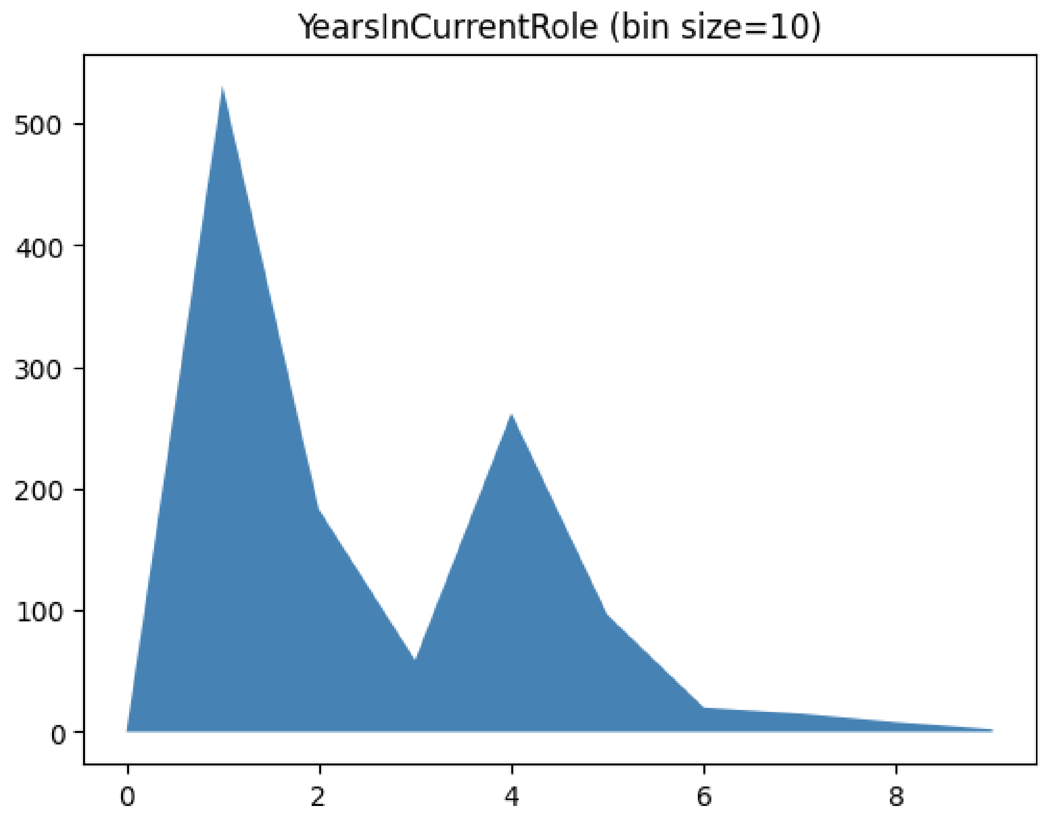

PercentSalaryHike (bin size=10)
0
50
100
150
200
250
300
0
2
4
6
8

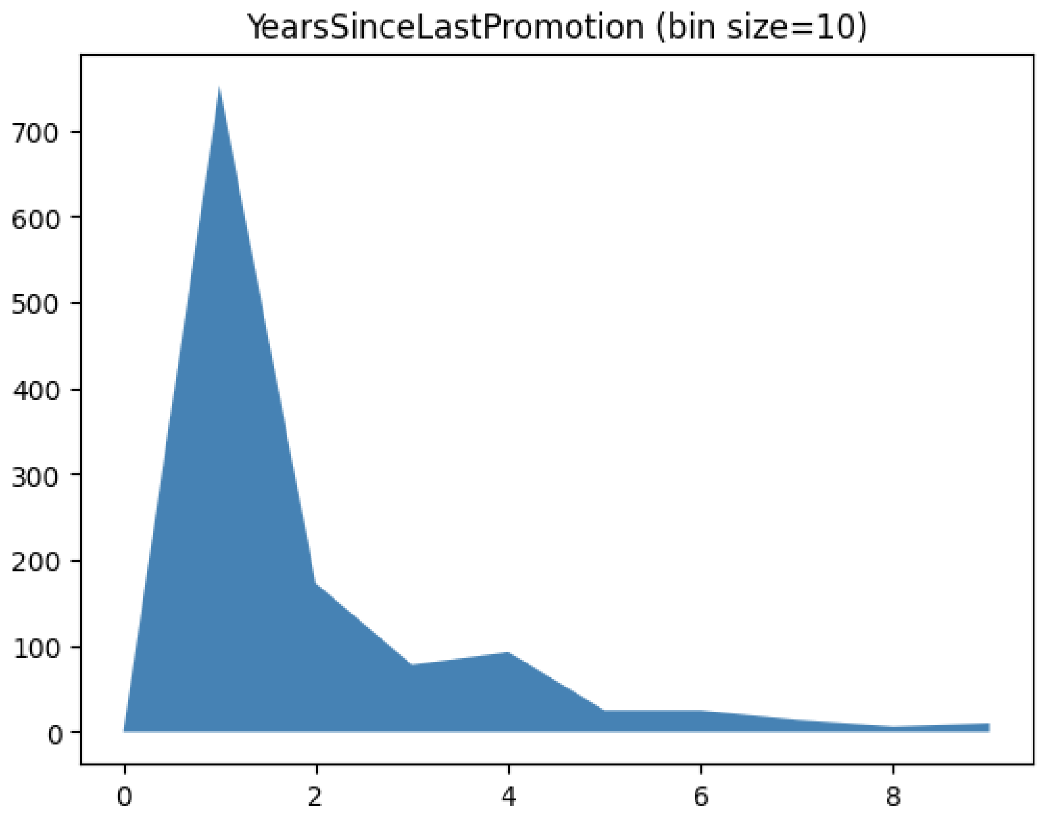

TotalWorkingYears (bin size=10)
400
350
300
250
200
150
100
50
0
0
2
4
6
8

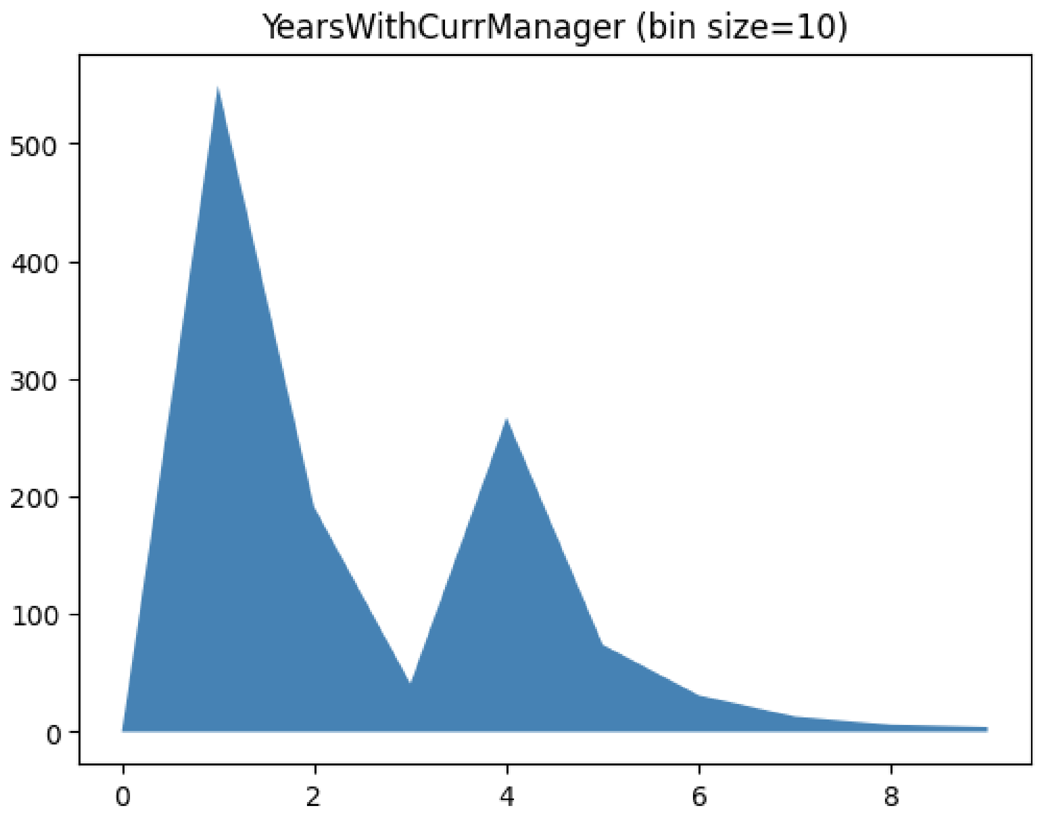
TrainingTimesLastYear (bin size=10)




YearsAtCompany (bin size=10)
0
100
200
300
400
0
2
4
6
8


YearsInCurrentRole (bin size=10)
500
400
300
200
100
0
0
2
4
6
8


YearsSinceLastPromotion (bin size=10)
700
600
500
400
300
200
100
0
0
2
4
6
8



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.

The distribution which we will use for describing histograms is categorical distribution which describe each bins as parameter and x_i as frequency in each bins

$P(X = x) = p_1^{x_1} p_2^{x_2} ... p_k^{x_k}$

We will find p_i which maximize $P(X = x)$ by using lagrange multiplier

T8 : $L(\theta) = \prod_{i=1}^{N} \theta_i^{x_i}$

$$\log L(\theta) = \sum_{i=1}^{N} x_i \log \theta_i$$

$$f(\theta_1, \theta_2, \dots \theta_n) = \log L(\theta) \rightarrow \nabla f(\theta_1, \theta_2, \theta_3, \dots, \theta_n) = \left(\frac{x_1}{\theta_1}, \frac{x_2}{\theta_2}, \dots \frac{x_n}{\theta_n}\right)$$

$$g(\theta_1, \theta_2, \dots, \theta_n) = \sum_{i=1}^{N} \theta_i \longrightarrow \nabla g(\theta_1, \theta_2, \theta_3, \dots, \theta_n) = (1, 1, 1, \dots, 1)$$

Solve for $\nabla f(\theta_1, \theta_2, \dots, \theta_n) = \lambda \nabla g(\theta_1, \theta_2 \dots, \theta_n)$

$$\frac{x_1}{\theta_1} = \lambda, \frac{x_2}{\theta_2} = \lambda \dots \frac{x_n}{\theta_n} = \lambda$$

$$\therefore x_1 = \lambda\theta_1, x_2 = \lambda\theta_2, \dots, x_n = \lambda\theta_n$$

$$g(\theta_1, \theta_2, \dots, \theta_n) = \sum_{i=1}^{N} \theta_i = \frac{x_1}{\lambda} + \frac{x_2}{\lambda} + \frac{x_3}{\lambda} + \dots + \frac{x_n}{\lambda} = 1$$

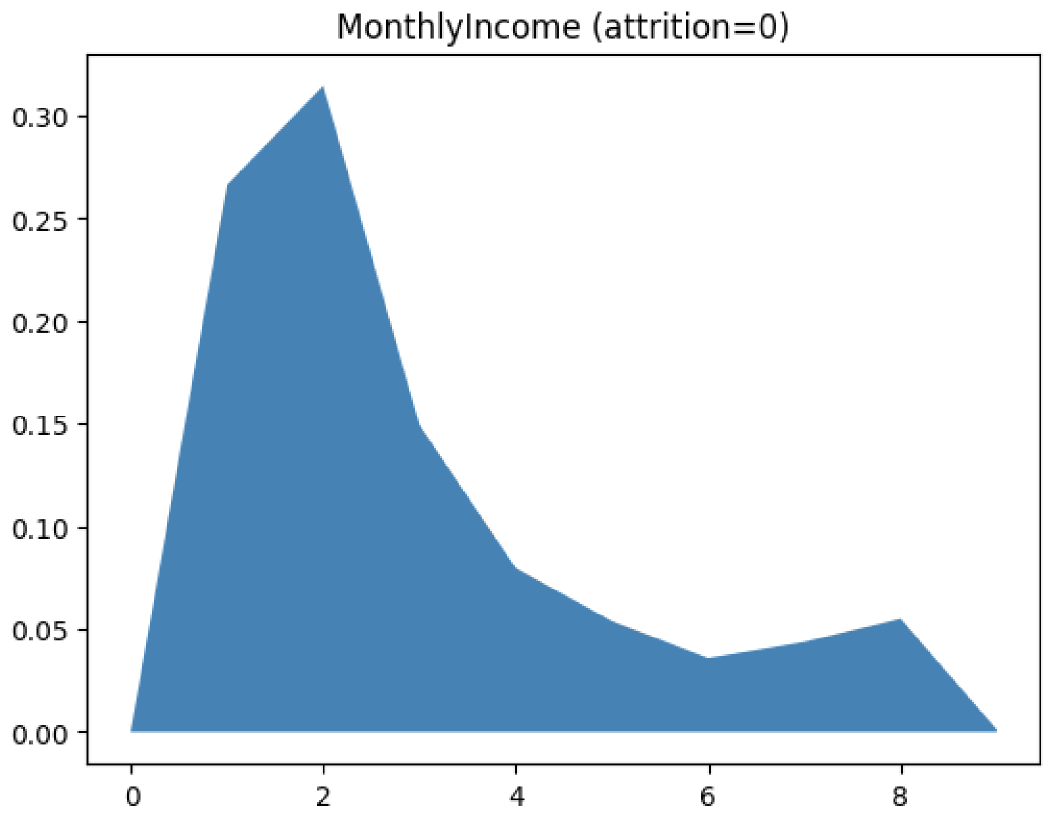
$$\therefore \lambda = x_1 + x_2 + x_3, \dots x_n$$

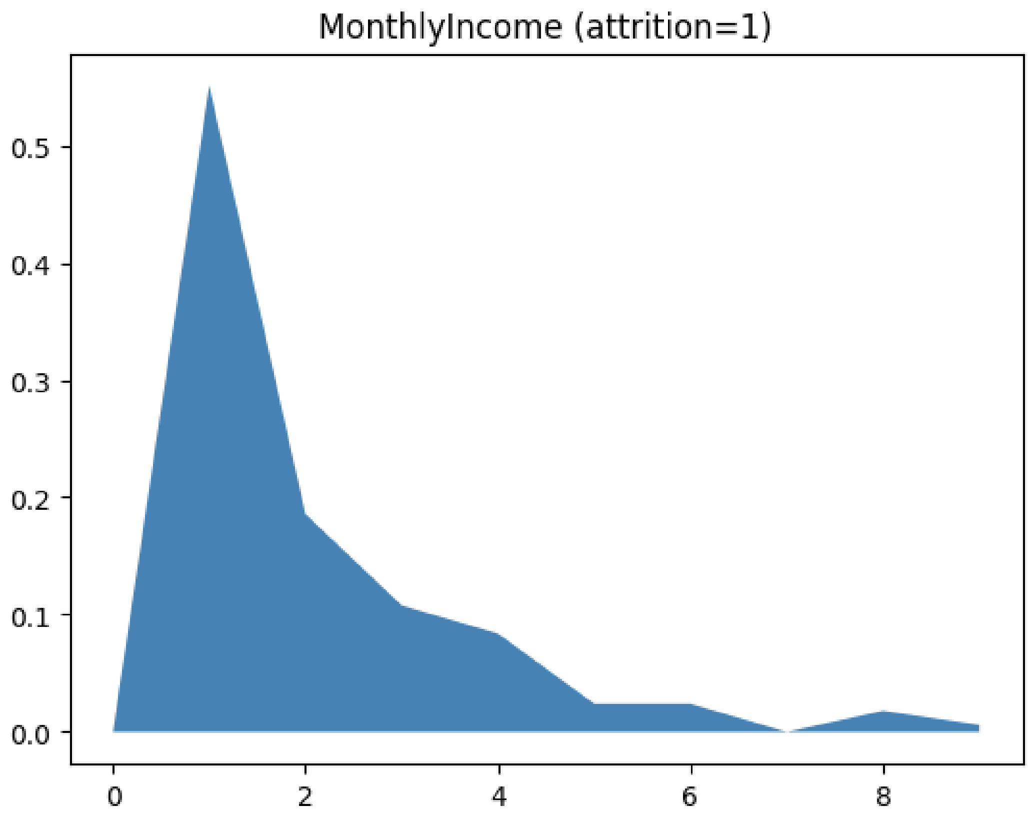
$$\therefore \theta_1 = \frac{x_1}{\sum x_i}, \theta_2 = \frac{x_2}{\sum x}, \dots, \theta_n = \frac{x_n}{\sum x_i}$$

[ ]:
```python
def plot_likelihood_attrition(df, y, attrition, col_name, n_bin=10):
    features = df[col_name][~(np.isnan(df[col_name]))]
    features = features[y == attrition]

    bins, edges = calculate_bins_and_edges(features, n_bin)
    bins = bins / np.sum(bins)

    plt.fill_between(np.arange(0, n_bin), bins, facecolor="steelblue")
    plt.title(f"{col_name} (attrition={attrition})")
    plt.show()
```

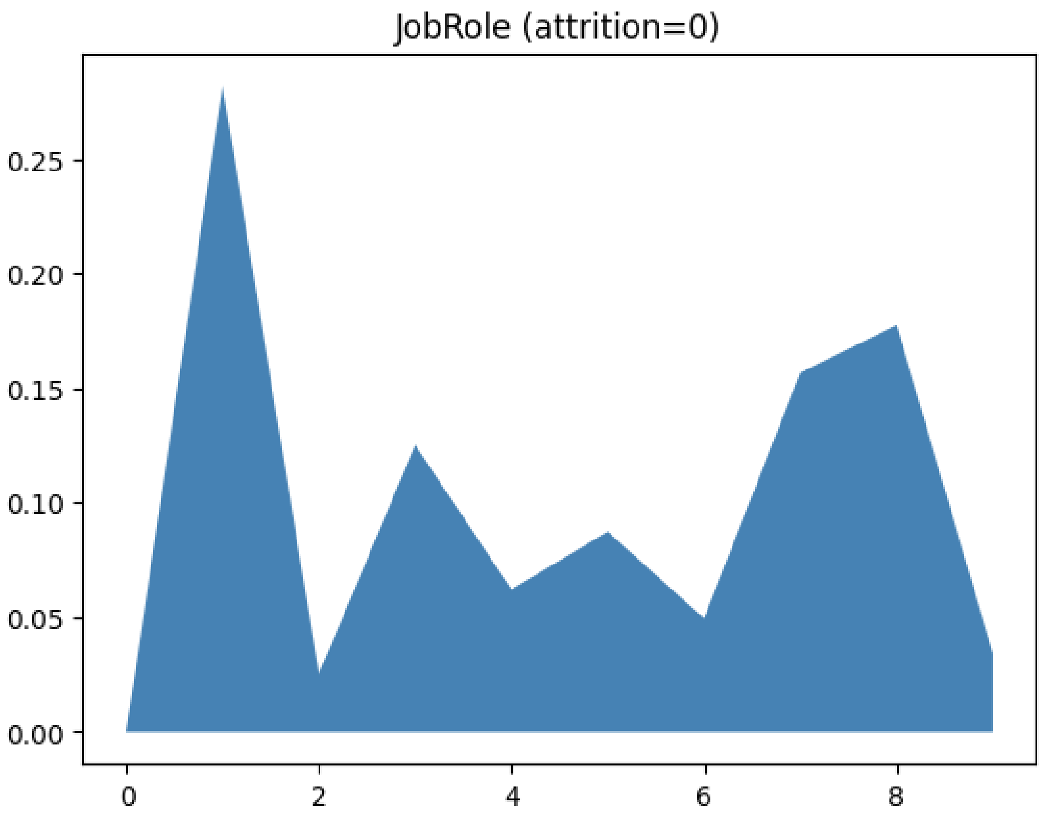
[ ]:
```python
y_train = df_train["Attrition"]
```

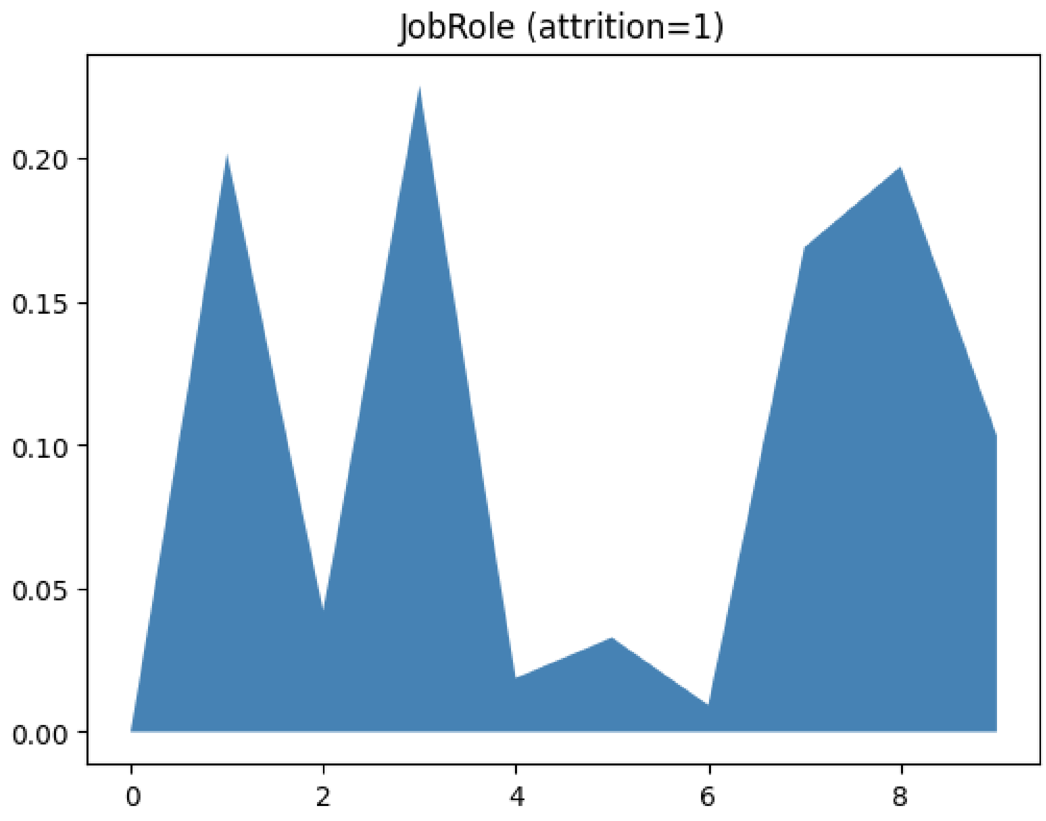
[ ]:
```python
plot_likelihood_attrition(df_train, y_train, 0, "MonthlyIncome")
plot_likelihood_attrition(df_train, y_train, 1, "MonthlyIncome")
```


MonthlyIncome (attrition=0)
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8

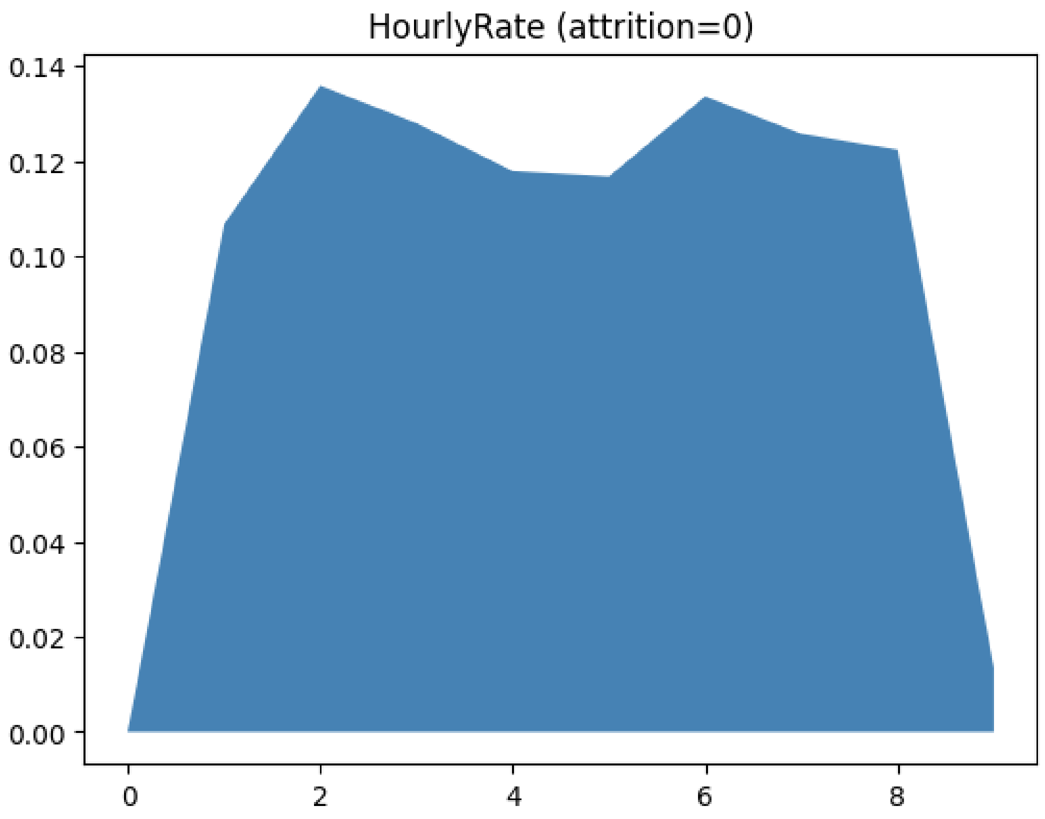


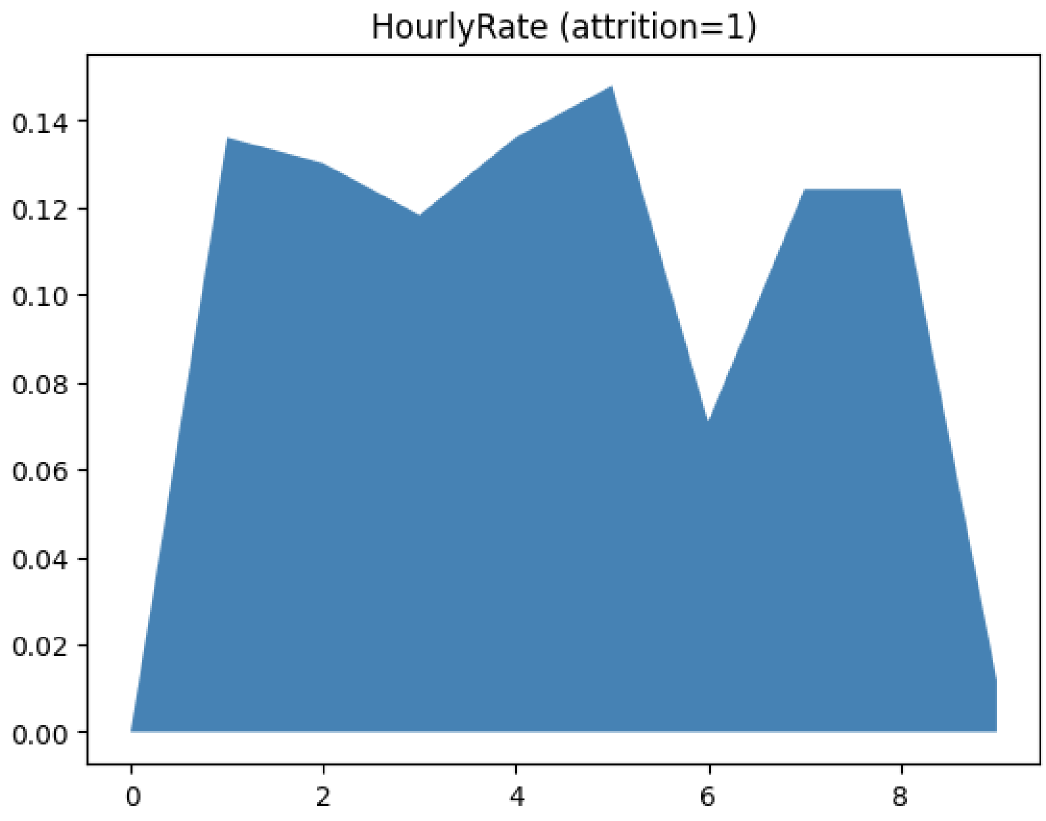
```
[ ]: plot_likelihood_attrition(df_train, y_train, 0, "JobRole")
     plot_likelihood_attrition(df_train, y_train, 1, "JobRole")
```


JobRole (attrition=0)
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8

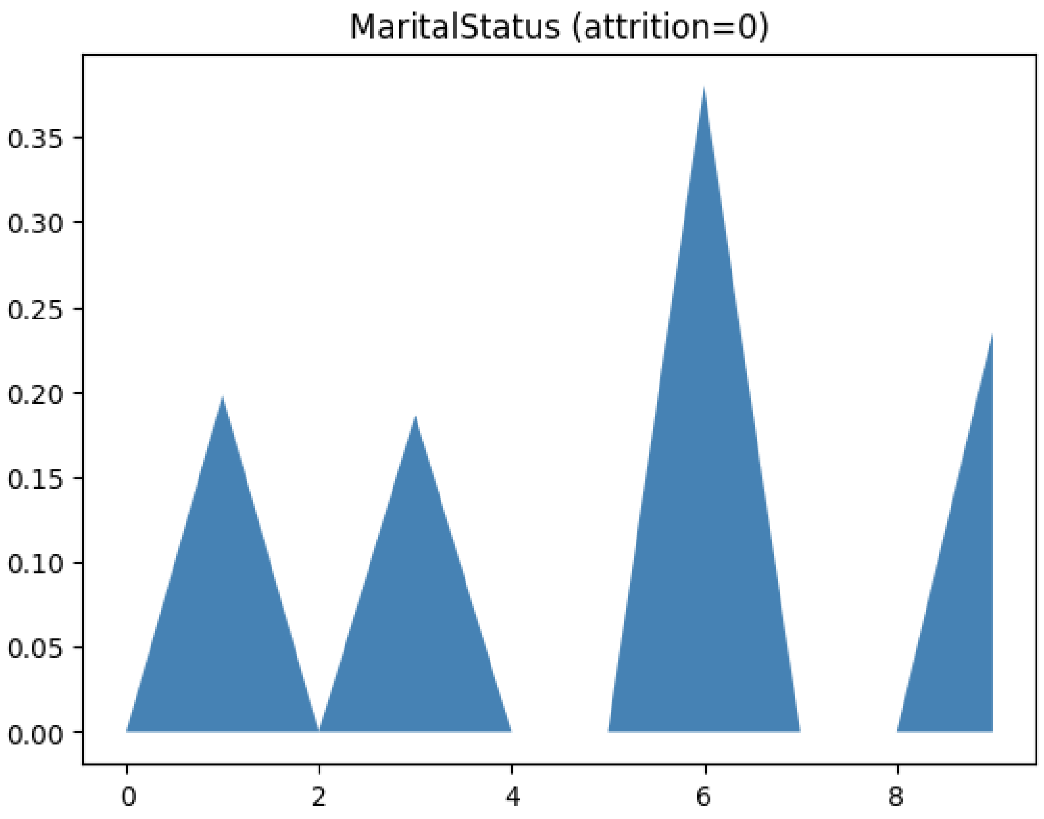


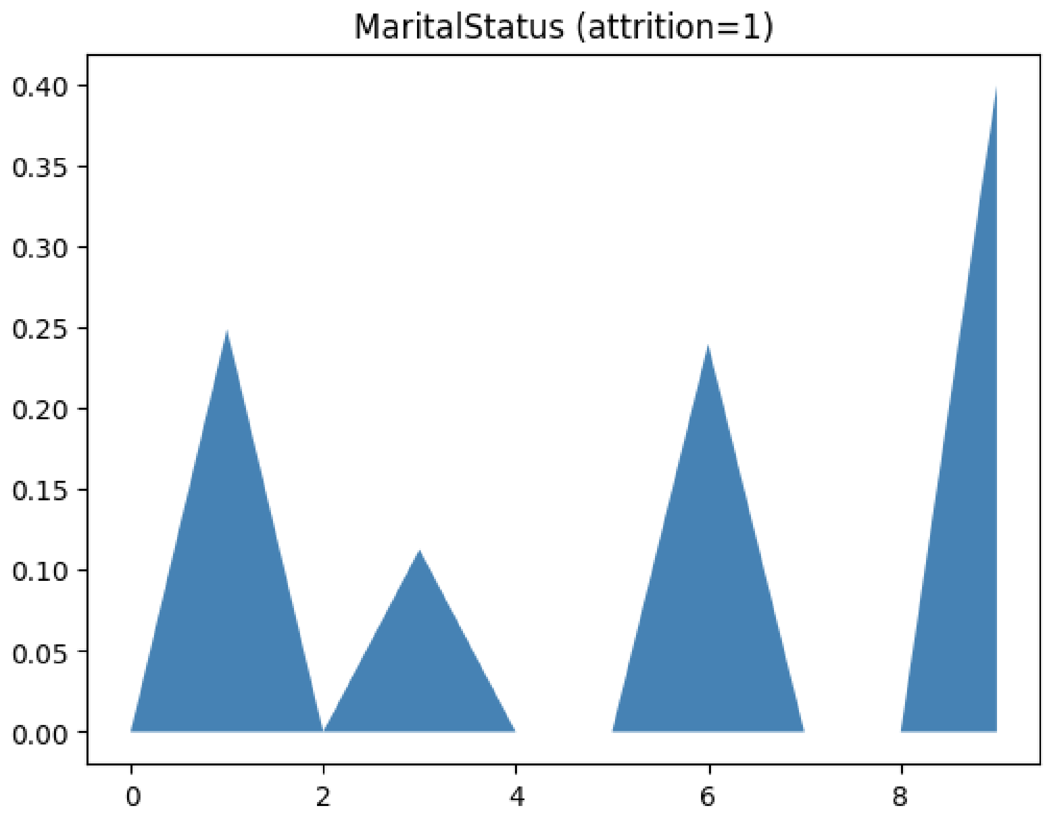
```
[ ]: plot_likelihood_attrition(df_train, y_train, 0, "HourlyRate")
     plot_likelihood_attrition(df_train, y_train, 1, "HourlyRate")
```


HourlyRate (attrition=0)
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
2
4
6
8



```
[ ]: plot_likelihood_attrition(df_train, y_train, 0, "MaritalStatus")
     plot_likelihood_attrition(df_train, y_train, 1, "MaritalStatus")
```


MaritalStatus (attrition=0)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8



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

$P(\theta) = (0.16)^{\theta} * (0.84)^{1-\theta}\ ; \theta = 0, 1$

```
[ ]: n_stay = np.sum(y_train == 1)
     n_leave = np.sum(y_train == 0)

     prior_stay = n_stay / (n_stay + n_leave)
     prior_leave = n_leave / (n_stay + n_leave)

     print(f"Prior class stay: {prior_stay}")
     print(f"Prior class leave: {prior_leave}")
```

```
Prior class stay: 0.16099773242630386
Prior class leave: 0.8390022675736961
```

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.

I think one of the method to prevent zero probability is to distribute a small value from non-zero term to add to its, this way will not affect to overall probability much and still preserve zero probability

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.

```
[ ]: import random as rnd
     import pandas as pd
     import matplotlib.pyplot as plt
     import numpy as np
     from scipy import stats

     class SimpleBayesClassifier:

         def __init__(self, n_pos, n_neg):

             """
             Initializes the SimpleBayesClassifier with prior probabilities.

             Parameters:
             n_pos (int): The number of positive samples.
             n_neg (int): The number of negative samples.

             Returns:
             None: This method does not return anything as it is a constructor.
             """

             self.n_pos = n_pos
             self.n_neg = n_neg
             self.prior_pos = (n_pos) / (n_pos + n_neg)
             self.prior_neg = (n_neg) / (n_pos + n_neg)

         def fit_params(self, x, y, n_bins = 10):

             """
             Computes histogram-based parameters for each feature in the dataset.

             Parameters:
             x (np.ndarray): The feature matrix, where rows are samples and columns␣
      ↪are features.
             y (np.ndarray): The target array, where each element corresponds to the␣
      ↪label of a sample.
             n_bins (int): Number of bins to use for histogram calculation.

             Returns:
```

```
        (stay_params, leave_params): A tuple containing two lists of tuples,
        one for 'stay' parameters and one for 'leave' parameters.
        Each tuple in the list contains the bins and edges of the histogram for␣
↪a feature.
        """

        self.stay_params = [(None, None) for _ in range(x.shape[1])]
        self.leave_params = [(None, None) for _ in range(x.shape[1])]

        stay_df = x[y == 1]
        leave_df = x[y == 0]

        column_width = x.shape[1]

        for i in range(column_width):
            feature_stay_df = stay_df[:, i]
            feature_stay_df = feature_stay_df[~(np.isnan(feature_stay_df))]

            bins, edges = calculate_bins_and_edges(feature_stay_df, n_bins)
            bins = bins / np.sum(bins)

            x = np.sum(bins == 0)

            bins[bins == 0] = 1e-6
            bins[bins.argmax()] -= 1e-6 * x

            self.stay_params[i] = (bins, edges)

        for i in range(column_width):
            feature_leave_df = leave_df[:, i]
            feature_leave_df = feature_leave_df[~(np.isnan(feature_leave_df))]

            bins, edges = calculate_bins_and_edges(feature_leave_df, n_bins)
            bins = bins / np.sum(bins)

            x = np.sum(bins == 0)

            bins[bins == 0] = 1e-6
            bins[bins.argmax()] -= 1e-6 * x

            self.leave_params[i] = (bins, edges)

        return self.stay_params, self.leave_params

    def predict(self, x, thresh = 0):

        """
        Predicts the class labels for the given samples using the␣
↪non-parametric model.

        Parameters:
        x (np.ndarray): The feature matrix for which predictions are to be made.
        thresh (float): The threshold for log probability to decide between␣
↪classes.

        Returns:
        result (list): A list of predicted class labels (0 or 1) for each␣
↪sample in the feature matrix.
        """

        y_pred = []
        classifier_arr = []

        for data in x:
            classifier = np.log(self.prior_neg) - np.log(self.prior_pos)
            for (idx, feature) in enumerate(data):
                if np.isnan(feature):
                    continue

                bins_pos, edges_pos = self.stay_params[idx]

                digitalized_feature_pos = np.digitize(feature, edges_pos) - 1
                log_prob_pos = np.log(bins_pos[digitalized_feature_pos])

                bins_neg, edges_neg = self.leave_params[idx]

                digitalized_feature_neg = np.digitize(feature, edges_neg) - 1
                log_prob_neg = np.log(bins_neg[digitalized_feature_neg])

                classifier += log_prob_neg - log_prob_pos

            classifier_arr.append(classifier)
            y_pred.append(1 if classifier <= thresh else 0)

        return np.array(y_pred), np.array(classifier_arr)

    def fit_gaussian_params(self, x: np.ndarray, y: np.ndarray):

        """
        Computes mean and standard deviation for each feature in the dataset.

        Parameters:
        x (np.ndarray): The feature matrix, where rows are samples and columns␣
↪are features.
```

```
        y (np.ndarray): The target array, where each element corresponds to the␣
↪label of a sample.

        Returns:
        (gaussian_stay_params, gaussian_leave_params): A tuple containing two␣
↪lists of tuples,
        one for 'stay' parameters and one for 'leave' parameters.
        Each tuple in the list contains the mean and standard deviation for a␣
↪feature.
        """

        self.gaussian_stay_params = [(0, 0) for _ in range(x.shape[1])]
        self.gaussian_leave_params = [(0, 0) for _ in range(x.shape[1])]

        stay_df = x[y == 1]
        leave_df = x[y == 0]

        column_width = x.shape[1]

        for i in range(column_width):
            feature_stay_df = stay_df[:, i]
            feature_stay_df = feature_stay_df[~(np.isnan(feature_stay_df))]

            self.gaussian_stay_params[i] = (np.mean(feature_stay_df), np.
↪std(feature_stay_df))

            feature_leave_df = leave_df[:, i]
            feature_leave_df = feature_leave_df[~(np.isnan(feature_leave_df))]

            self.gaussian_leave_params[i] = (np.mean(feature_leave_df), np.
↪std(feature_leave_df))

        return self.gaussian_stay_params, self.gaussian_leave_params

    def gaussian_predict(self, x:np.ndarray, thresh = 0):
        """
        Predicts the class labels for the given samples using the parametric␣
↪model.

        Parameters:
        x (np.ndarray): The feature matrix for which predictions are to be made.
        thresh (float): The threshold for log probability to decide between␣
↪classes.

        Returns:
```

```
        result (list): A list of predicted class labels (0 or 1) for each␣
↪sample in the feature matrix.
        """

        y_pred = []
        classifier_arr = []

        # INSERT CODE HERE
        for data in x:
            classifier = np.log(self.prior_neg) - np.log(self.prior_pos)
            for (idx, feature) in enumerate(data):
                if np.isnan(feature):
                    continue

                log_prob_pos = stats.norm.logpdf(feature, self.
↪gaussian_stay_params[idx][0], self.gaussian_stay_params[idx][1])
                log_prob_neg = stats.norm.logpdf(feature, self.
↪gaussian_leave_params[idx][0], self.gaussian_leave_params[idx][1])

                classifier += log_prob_neg - log_prob_pos

            classifier_arr.append(classifier)
            y_pred.append(1 if classifier <= thresh else 0)

        return np.array(y_pred), np.array(classifier_arr)
```

[ ]:
```
data_train = df_train.to_numpy()
data_test = df_test.to_numpy()

df
```

[ ]:
```
       Age  Attrition  BusinessTravel  DailyRate  Department  \
0     41.0          1               2        NaN          -1
1      NaN          0              -1      279.0           1
2     37.0          1              -1     1373.0          -1
3      NaN          0               1     1392.0           1
4     27.0          0               2      591.0           1
…      …          …               …         …           …
1465  36.0          0               1      884.0           1
1466  39.0          0               2      613.0          -1
1467  27.0          0              -1      155.0           1
1468  49.0          0               1     1023.0           2
1469  34.0          0               2      628.0           1

      DistanceFromHome  Education  EducationField  EnvironmentSatisfaction  \
0                  1.0        NaN               1                      2.0
1                  NaN        NaN               1                      3.0
```

```
2                  2.0        2.0              -1                     NaN
3                  3.0        4.0               1                     NaN
4                  2.0        1.0               3                     1.0
…                …        …             …                     …
1465               NaN        NaN               3                     3.0
1466               6.0        NaN               3                     4.0
1467               4.0        3.0               1                     2.0
1468               2.0        3.0              -1                     4.0
1469               NaN        NaN              -1                     2.0

      Gender  …  PerformanceRating  RelationshipSatisfaction  \
0          0  …                NaN                       1.0
1          1  …                NaN                       4.0
2          1  …                3.0                       NaN
3          0  …                3.0                       3.0
4          1  …                3.0                       4.0
…        …  …              …                       …
1465       1  …                3.0                       3.0
1466      -1  …                3.0                       NaN
1467       1  …                NaN                       2.0
1468       1  …                3.0                       4.0
1469       1  …                3.0                       NaN

      StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
0                  0.0                8.0                    0.0
1                  1.0               10.0                    NaN
2                  0.0                7.0                    3.0
3                  NaN                8.0                    3.0
4                  1.0                6.0                    NaN
…                …                …                    …
1465               1.0               17.0                    3.0
1466               NaN                9.0                    5.0
1467               1.0                6.0                    0.0
1468               0.0               17.0                    NaN
1469               0.0                6.0                    3.0

      WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
0                 NaN             6.0                 NaN
1                 3.0            10.0                 NaN
2                 NaN             NaN                 0.0
3                 NaN             8.0                 NaN
4                 3.0             2.0                 2.0
…               …             …                 …
1465              3.0             5.0                 2.0
1466              3.0             7.0                 7.0
1467              3.0             6.0                 NaN
1468              2.0             9.0                 6.0
```

```
1469               4.0             4.0                   NaN

      YearsSinceLastPromotion  YearsWithCurrManager
0                         0.0                   NaN
1                         NaN                   7.0
2                         NaN                   0.0
3                         3.0                   0.0
4                         2.0                   NaN
…                       …                     …
1465                      0.0                   3.0
1466                      1.0                   7.0
1467                      0.0                   3.0
1468                      0.0                   8.0
1469                      1.0                   2.0

[1470 rows x 31 columns]
```

```
[ ]: x_train = np.delete(data_train, 1, axis=1)
     y_train = data_train[:,1]

     x_test = np.delete(data_test, 1, axis=1)
     y_test = data_test[:,1]
```

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

```
[ ]: 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) == (0.0, 1.0)
         assert (SimpleBayesClassifier(1, 0).prior_pos, SimpleBayesClassifier(1, 0).
      ↪prior_neg) == (1.0, 0.0)

     check_prior()
```

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

[ ]:
```
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]
    ])
    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()
```

[ ]:
```
y_pred, _ = model.predict(x = x_test)
```

[ ]:
```
def evaluate(y_true, y_pred, cls = 0):
  y_pred_arr = np.array(y_pred)
  y_true_arr = np.array(y_true)

  result = y_true_arr == y_pred_arr

  accuracy = np.sum(result) / len(y_pred)
  recall = np.sum(result[y_pred_arr == cls]) / np.sum(y_true_arr == cls)
  precision = np.sum(result[y_pred_arr == cls]) / np.sum(y_pred_arr == cls)
  F1 = 2 * (recall * precision) / (recall + precision)

  fpr = np.sum(np.logical_and(y_true == 0, y_pred == 1)) / np.sum(y_true_arr ==␣
↪0)

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

[ ]:
```
accuracy, precision, recall, F1, fpr = evaluate(y_test, y_pred, cls=1)

print(f"accuracy: {accuracy}")
print(f"precision: {precision}")
print(f"recall: {recall}")
print(f"F1 score: {F1}")
```

```
accuracy: 0.8707482993197279
precision: 0.6923076923076923
recall: 0.375
F1 score: 0.48648648648648646
```

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

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

[ ]:
```
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 : 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

LEAVE 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
```

[ ]:
```
y_pred, _ = model.gaussian_predict(x_test)
```

[ ]:
```
accuracy, precision, recall, F1, fpr = evaluate(y_test, y_pred, cls=0)

print(f"accuracy: {accuracy}")
print(f"precision: {precision}")
print(f"recall: {recall}")
print(f"F1 score: {F1}")
```

```
accuracy: 0.8367346938775511
precision: 0.889763779527559
recall: 0.9186991869918699
F1 score: 0.904
```

[ ]:
```
accuracy, precision, recall, F1, fpr = evaluate(y_test, y_pred, cls=1)

print(f"accuracy: {accuracy}")
print(f"precision: {precision}")
print(f"recall: {recall}")
print(f"F1 score: {F1}")
```

```
accuracy: 0.8367346938775511
precision: 0.5
recall: 0.4166666666666667
F1 score: 0.45454545454545453
```

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.

[ ]:
```
test_size = y_test.shape[0]

y_random_result = np.random.choice([0, 1], test_size)

y_random_result
```

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

[ ]:
```
accuracy, precision, recall, F1, fpr = evaluate(y_test, y_random_result, cls=1)

print(f"accuracy: {accuracy}")
print(f"precision: {precision}")
print(f"recall: {recall}")
print(f"F1 score: {F1}")
```

```
accuracy: 0.4897959183673469
precision: 0.17721518987341772
```

```
recall: 0.5833333333333334
F1 score: 0.27184466019417475
```

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.

[ ]:
```
test_size = y_test.shape[0]

most_appear_class = np.argmax(np.bincount(np.array(y_train, dtype=int)))

y_majority_result = np.array([most_appear_class] * test_size)
```

[ ]:
```
accuracy, precision, recall, F1, fpr = evaluate(y_test, y_majority_result,␣
 ↪cls=1)

print(f"accuracy: {accuracy}")
print(f"precision: {precision}")
print(f"recall: {recall}")
print(f"F1 score: {F1}")
```

```
accuracy: 0.8367346938775511
precision: nan
recall: 0.0
F1 score: nan
```

```
<ipython-input-376-48a16e509fc3>:9: RuntimeWarning: invalid value encountered in
long_scalars
  precision = np.sum(result[y_pred_arr == cls]) / np.sum(y_pred_arr == cls)
```

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

- Naive Bayes Classifier which calculate probability from histogram has better accuracy than Random classifier baseline and Majority rule baseline but has poor recall compared to random classifier baseline
- Naive Bayes Classifier which calculate probabilty from gaussian distribution also has better accuracy than random and little difference from majority rule baseline but it has quite better precision than random baseline

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)

[ ]:
```
t = np.arange(-5, 5, 0.05)
```

```
[ ]: y_predicted, classifier_arr = model.predict(x_test, 0)

     F1 = (0.0, 0.0)
     accuracy = (0.0, 0.0)

     for threshold in t:
         result = classifier_arr <= threshold

         y_predicted[result == True] = 1
         y_predicted[result == False] = 0

         y_pred_arr = np.array(y_predicted)
         y_true_arr = np.array(y_test)

         fpr = y_true_arr == y_pred_arr

         _accuracy, _, _, _F1, fpr = evaluate(y_test, y_predicted, cls=1)

         if accuracy[0] < _accuracy:
             accuracy = (_accuracy, threshold)

         if F1[0] < _F1:
             F1 = (_F1, threshold)

     print(f"Best Accuracy: {accuracy[0]} at threshold {accuracy[1]}")
     print(f"Best F1 Score: {F1[0]} at threshold {F1[1]}")
```

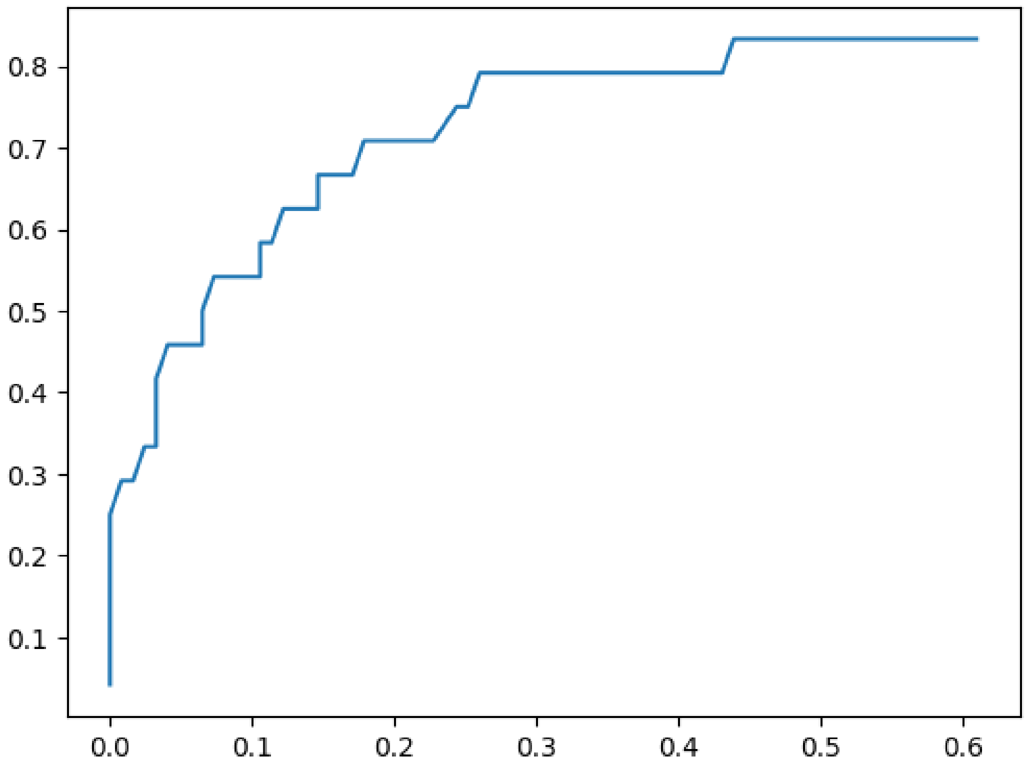
```
Best Accuracy: 0.8775510204081632 at threshold -1.7500000000000115
Best F1 Score: 0.5652173913043478 at threshold 0.5499999999999803
```

1.0.19 T17. Plot the RoC of your classifier.

```
[ ]: y_predicted, classifier_arr = model.predict(x_test, 0)

     roc_curve_tpr = []
     roc_curve_fpr = []

     for threshold in t:
       result = classifier_arr <= threshold

       y_predicted[result == True] = 1
       y_predicted[result == False] = 0

       y_pred_arr = np.array(y_predicted)
       y_true_arr = np.array(y_test)

       _, _, tpr, _, fpr = evaluate(y_test, y_predicted, cls=1)
```

```
    roc_curve_tpr.append(tpr)
    roc_curve_fpr.append(fpr)
```

[ ]:
```
plt.plot(roc_curve_fpr, roc_curve_tpr)
plt.show()
```



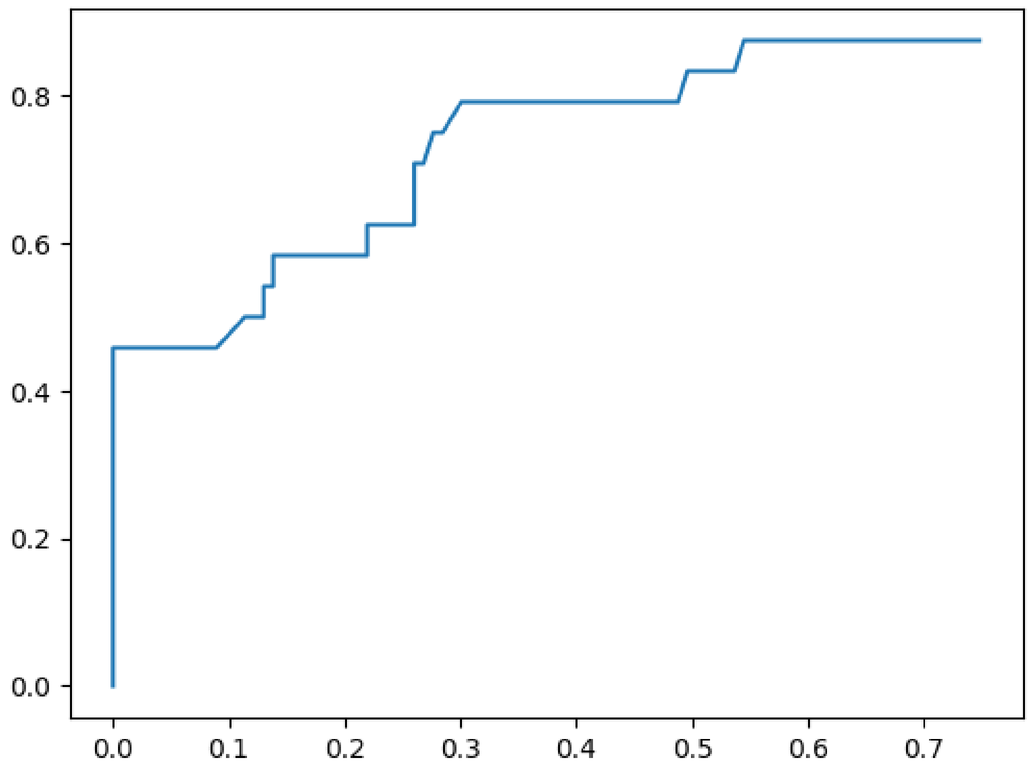
1.0.20 T18. Change the number of discretization bins to 5. What happens to the RoC curve? Which discretization is better? The number of discretization bins can be considered as a hyperparameter, and must be chosen by comparing the final performance.

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

[ ]:
```
new_model.fit_params(x_train, y_train, 5)
```

[ ]:
```
new_y_predicted, new_classifier_arr = new_model.predict(x_test, 0)

new_roc_curve_tpr = []
new_roc_curve_fpr = []

for threshold in t:
  result = new_classifier_arr <= threshold

  new_y_predicted[result == True] = 1
  new_y_predicted[result == False] = 0

  y_pred_arr = np.array(new_y_predicted)
  y_true_arr = np.array(y_test)

  _, _, tpr, _, fpr = evaluate(y_test, new_y_predicted, cls=1)

  new_roc_curve_tpr.append(tpr)
  new_roc_curve_fpr.append(fpr)
```

[ ]:
```
plt.plot(new_roc_curve_fpr, new_roc_curve_tpr)
plt.show()
```



We see that true positive rate and false positive rate is significantly increased from bin size 10 and overall trend look like that ROC curve has increased faster than ROC curve when bin size is 10. I think bin size 10 has better discretization than bin size 5 because at the same true positive rate value, bin size 10 tends to give less false positive rate than bin size 5 if we compare in ROC curve