T1.

Likelihood ($L(\alpha)$)

find: MLE: $\underset{\alpha}{\arg\max}\ \overbrace{p(Y \mid \alpha)}^{\text{Likelihood } (L(\alpha))}$

$\cap$ intersect

$= \underset{\alpha}{\arg\max}\ p(y_2, y_1, y_0 \mid \alpha)$

probabilities density f^n

continuous random value.

} Conditional Probabilities

$$p(\alpha \cap \beta) = p(\alpha)\, p(\beta \mid \alpha)$$

in continuous RV. , in discrete RV. $P(X \in C \mid Y \in D) = \dfrac{P(X \in C, Y \in D)}{P(Y \in D)}$

$$p(\alpha \cap \beta \cap \gamma) = p(\alpha \cap \beta)\, p(\gamma \mid \alpha \cap \beta)$$

$$\vdots$$

Chain Rule of Condition Probabilities

$$p(\alpha_1 \cap \dots \cap \alpha_n) = p(\alpha_1)\, p(\alpha_2 \mid \alpha_1) \dots p(\alpha_n \mid \alpha_1 \cap \dots \cap \alpha_{n-1})$$

$$= \prod_{k=1}^{n} p\left(\alpha_k \,\middle|\, \bigcap_{j=1}^{k-1} a_j\right)$$

$$= \underset{\alpha}{\arg\max}\ p(y_2 \mid y_1 \cap y_0 ; \alpha)\ p(y_1 \mid y_0 ; \alpha)\ p(y_0 ; \alpha)$$

} from Q. : y_0 is independent ($\perp$) of the noise sequence (w_0, w_1)

$$\therefore\ y_0 \perp w_0, w_1$$

from $y_1 = \alpha y_0 + w_0$ implie $y_1 \perp y_2 ; y_0$

$y_2 = \alpha y_1 + w_1$ " $\therefore\ y_2 \perp y_0 ; y_1$

} from $y_2 \perp y_0 ; y_1$

, Markov process : $p(y_2 \mid y_1, y_0) = p(y_2 \mid y_1)$

$$= \underset{\alpha}{\arg\max}\ p(y_2 \mid y_1 ; \alpha)\ p(y_1 \mid y_0 ; \alpha)\ p(y_0 ; \alpha)$$

$$= \arg\max_{\alpha} \; p(y_2 \mid y_1 ; \alpha) \; p(y_1 \mid y_0 ; \alpha) \; p(y_0 ; \alpha)$$

$$L(\alpha) = p(y_2 \mid y_1 ; \alpha) \; p(y_1 \mid y_0 ; \alpha) \; p(y_0 ; \alpha)$$

$$\ln L(\alpha) = \ln p(y_2 \mid y_1 ; \alpha) + \ln p(y_1 \mid y_0 ; \alpha) + \ln p(y_0 ; \alpha)$$

} in Normal Dist., probability density f^n:

- $p(x) = \dfrac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

$$\ln p(x) = \ln e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} - \ln (2\pi\sigma^2)^{\frac{1}{2}}$$
$$= -\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2 - \frac{1}{2} \ln (2\pi\sigma^2)$$

$$\ln p(x) = -\frac{1}{2} \frac{(x-\mu)^2}{\sigma^2} - \frac{1}{2} \ln (2\pi\sigma^2)$$

} from $w_0, w_1 \sim \mathcal{N}(0, \sigma^2)$

$y_0 \sim \mathcal{N}(0, \lambda)$

and $y_1 = \underbrace{\alpha \, y_0}_{\text{const.}} + \underbrace{w_0}_{\text{variable}}$

const. const.

} from Linear combination of normal random variables

$$\therefore \; y_1 \mid y_0 \sim \mathcal{N}(\alpha y_0, \sigma^2)$$

and $y_2 = \alpha y_1 + w_1$

$$\therefore \; y_2 \mid y_1 \sim \mathcal{N}(\alpha y_1, \sigma^2)$$

$$\ln L(\alpha) = \ln p(y_2 \mid y_1 ; \alpha) + \ln p(y_1 \mid y_0 ; \alpha) + \ln p(y_0 ; \alpha)$$

$$\ln L(\alpha) = -\frac{1}{2} \frac{(y_2 - \alpha y_1)^2}{\sigma^2} - \frac{1}{2} \ln (2\pi\sigma^2) + -\frac{1}{2} \frac{(y_1 - \alpha y_0)^2}{\sigma^2} - \frac{1}{2} \ln(2\pi\sigma^2) + -\frac{1}{2} \frac{y_0^2}{\lambda} - \frac{1}{2} \ln (2\pi\lambda)$$

$$\ln L(\alpha) = -\cancel{\frac{1}{2}}\frac{(y_2-\alpha y_1)^2}{\cancel{\sigma^2}} \cancel{-\frac{1}{2}\ln(2\pi\sigma^2)} + -\cancel{\frac{1}{2}}\frac{(y_1-\alpha y_0)^2}{\cancel{\sigma^2}} \cancel{-\frac{1}{2}\ln(2\pi\sigma^2)} + \cancel{-\frac{1}{2}\frac{y_0^2}{\lambda} - \frac{1}{2}\ln(2\pi\lambda)}$$

$$\frac{\partial}{\partial\alpha} L(\alpha) \equiv 0 = \cancel{2}(y_2-\alpha y_1)(-y_1) + \cancel{2}(y_1-\alpha y_0)(-y_0)$$

$$0 = -y_1 y_2 + \alpha y_1^2 - y_0 y_1 + \alpha y_0^2$$

$$\therefore \alpha = \frac{y_2 y_1 + y_1 y_0}{y_1^2 + y_0^2}$$

T1.

OT1.

$$\frac{\partial}{\partial\alpha} L(\alpha) \equiv 0 = \sum_{i=0}^{n} (y_{i+1} - \alpha y_i)(-y_i)$$

; General Case

$y_{n+1} = \alpha y_n + w_n$; $n = 0, 1, 2, \ldots$

$$= \sum_{i=0}^{n} \left(\alpha y_i^2 - y_{i+1} y_i\right)$$

$$0 = \sum \alpha y_i^2 - \sum y_{i+1} y_i$$

$$\sum_{i=0}^{n} \alpha y_i^2 = \sum_{i=0}^{n} y_{i+1} y_i$$

$$\therefore \alpha = \frac{\sum_{i=0}^{n} y_{i+1} y_i}{\sum_{i=0}^{n} y_i^2}$$

OT. 1.

HW2_SimpleBayesClassifier

February 2, 2024

Pupipat Singkhorn

1 Simple Bayes Classifier

A student in Pattern Recognition course had finally built the ultimate classifier for cat emotions. He used one input features: the amount of food the cat ate that day, x (Being a good student he already normalized x to standard Normal). He proposed the following likelihood probabilities for class 1 (happy cat) and 2 (sad cat)

$P(x|w1) = N(4, 2)$
$P(x|w2) = N(0, 2)$

Normal Distribution: $\mathcal{N}(\mu, \sigma^2)$

$$P(w_i|x) = \frac{P(x|w_i)P(w_i)}{P(x)}$$

$$Posterior = \frac{likelihood * prior}{evidence}$$

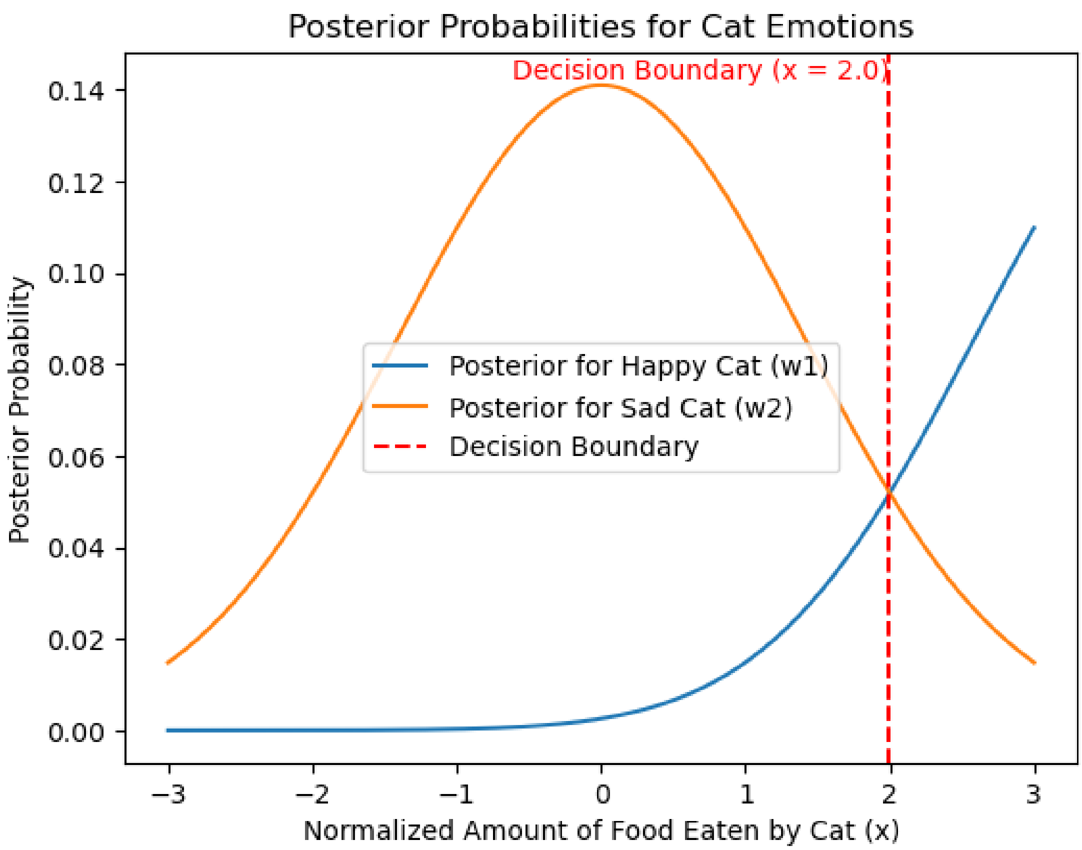
1.1 T2.

Plot the posteriors values of the two classes on the same axis. Using the likelihood ratio test, what is the decision boundary for this classifier? Assume equal prior probabilities.

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

class SimpleBayesClassifier:
    def __init__(self, mu1, var1, mu2, var2, prior_w1=0.5, prior_w2=0.5):
        self.mu1 = mu1
        self.var1 = var1
        self.sd1 = var1**0.5
        self.mu2 = mu2
        self.var2 = var2
        self.sd2 = var2**0.5
        self.prior_w1 = prior_w1
        self.prior_w2 = prior_w2
```

```python
        self.decision_boundary = None

    def calculate_posteriors(self, x_values):
        likelihood_w1 = norm.pdf(x_values, loc=self.mu1, scale=self.sd1)
        likelihood_w2 = norm.pdf(x_values, loc=self.mu2, scale=self.sd2)

        posterior_w1 = likelihood_w1 * self.prior_w1
        posterior_w2 = likelihood_w2 * self.prior_w2

        return posterior_w1, posterior_w2

    def find_decision_boundary(self, x_values):
        self.decision_boundary = x_values[np.argmin(np.abs(self.calculate_posteriors(x_values)[0] - self.calculate_posteriors(x_values)[1]))]

    def plot_posteriors(self, x_values):
        if self.decision_boundary is None:
            self.find_decision_boundary(x_values)

        posterior_w1, posterior_w2 = self.calculate_posteriors(x_values)

        # Plot posteriors
        plt.plot(x_values, posterior_w1, label='Posterior for Happy Cat (w1)')
        plt.plot(x_values, posterior_w2, label='Posterior for Sad Cat (w2)')

        # Plot decision boundary
        plt.axvline(x=self.decision_boundary, color='r', linestyle='--', label='Decision Boundary')

        # Add text annotation for decision boundary value
        plt.text(self.decision_boundary, max(max(posterior_w1), max(posterior_w2)),
                 f'Decision Boundary (x = {round(self.decision_boundary, 2)})',
                 verticalalignment='bottom', horizontalalignment='right', color='r')

        # Add labels and legend
        plt.xlabel('Normalized Amount of Food Eaten by Cat (x)')
        plt.ylabel('Posterior Probability')
        plt.title('Posterior Probabilities for Cat Emotions')
        plt.legend()
        plt.show()

if __name__ == "__main__":
    # Generate standard normalized x values
    x_values = np.linspace(-3, 3, 1000)

    # Create the classifier instance
```

```
classifier = SimpleBayesClassifier(mu1=4, var1=2, mu2=0, var2=2)

# Plot posteriors graph
classifier.plot_posteriors(x_values)

print(f'Decision Boundary: x = {classifier.decision_boundary}')
```



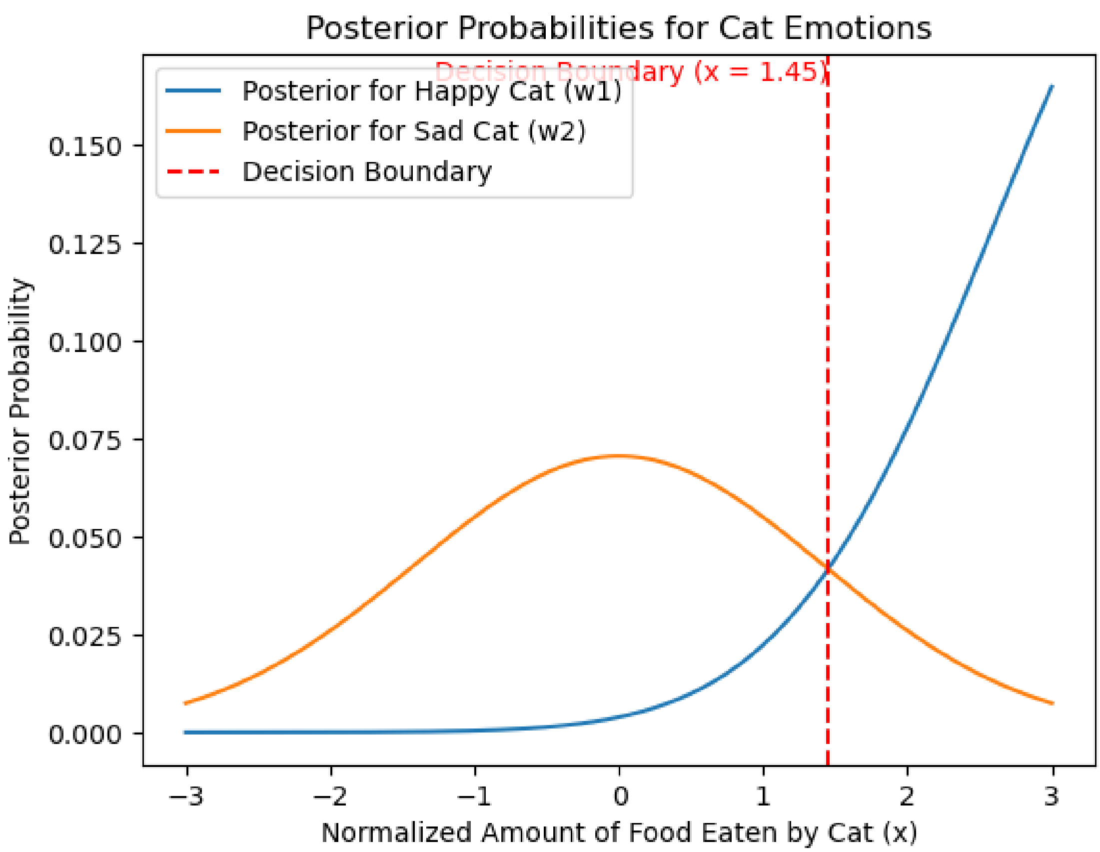

```
Decision Boundary: x = 1.9969969969969972
```

1.2 T3.

What happen to the decision boundary if the cat is happy with a prior of 0.75?

```
[ ]: # Generate standard normalized x values
x_values = np.linspace(-3, 3, 1000)

# Create the classifier instance with a different prior for the happy cat
classifier = SimpleBayesClassifier(mu1=4, var1=2, mu2=0, var2=2, prior_w1=0.75,␣
  ↪prior_w2=0.25)
```

```python
# Plot posteriors graph
classifier.plot_posteriors(x_values)

print(f'Decision Boundary: x = {classifier.decision_boundary}')
```




```
Decision Boundary: x = 1.4504504504504503
```

1.3 OT2.

$$P(x|w_1) = \mathcal{N}(\mu_1, \sigma^2), \quad P(x|w_2) = \mathcal{N}(\mu_2, \sigma^2), \quad p(w_1) = p(w_2) = 0.5$$

The decision boundary is where the posterior is the same

$$P(w_1|x) = P(w_2|x)$$

$$\frac{P(x|w_1)P(w_1)}{P(x)} = \frac{P(x|w_2)P(w_2)}{P(x)}$$

$$P(x|w_1) = P(x|w_2)$$

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

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

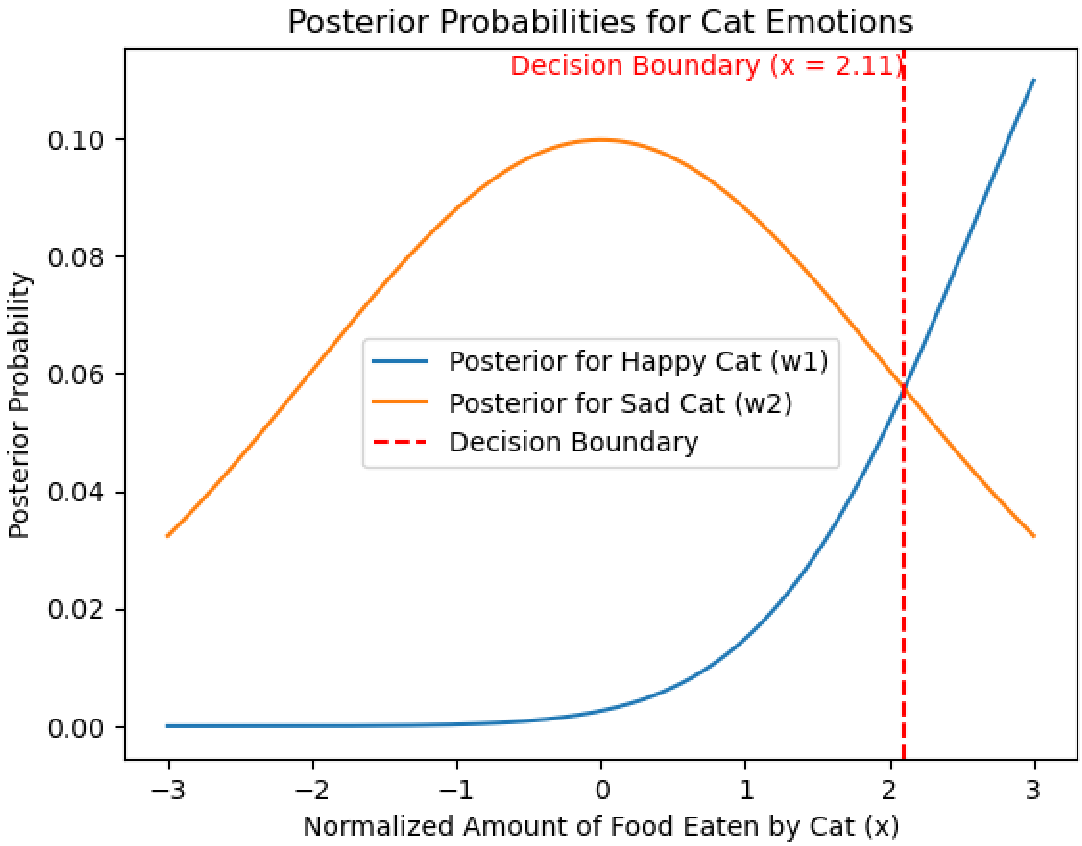
$$x^2 - 2\mu_1 x + \mu_1^2 = x^2 - 2\mu_2 x + \mu_2^2$$

$$\therefore x = \frac{\mu_1 + \mu_2}{2}$$

1.4 OT3.

If the student changed his model to P(x|w1) = N(4,2) P(x|w2) = N(0,4) Plot the posteriors values of the two classes on the same axis. What is the decision boundary for this classifier? Assume equal prior probabilities.

[ ]:
```
# Generate standard normalized x values
x_values = np.linspace(-3, 3, 1000)

# Create the classifier instance with a different prior for the happy cat
classifier = SimpleBayesClassifier(mu1=4, var1=2, mu2=0, var2=4, prior_w1=0.5,␣
 ↪prior_w2=0.5)

# Plot posteriors graph
classifier.plot_posteriors(x_values)

print(f'Decision Boundary: x = {classifier.decision_boundary}')
```




```
Decision Boundary: x = 2.105105105105105
```

HW2_EmployeeAttritionPrediction

February 7, 2024

1 Employee Attrition Prediction

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

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:
```

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

# HANDLE NULL NUMBERS
# INSERT CODE HERE

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

[ ]:
```
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]
```

1.0.4 Spliting data into train and test

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

df_train, df_test = train_test_split(df,
                                     test_size=0.1,
                                     stratify=df["Attrition"],
                                     random_state=7)
```

[ ]:
```python
df_train
```

[ ]:
```
       Age  Attrition  BusinessTravel  DailyRate  Department  \
1024  47.0          0               2        NaN           1
93    46.0          0               1      638.0           1
525   24.0          1               2      693.0           2
1450  35.0          0               2     1146.0           0
922   44.0          0               2     1199.0           1
…     …           …               …         …          …
216    NaN          1               1        NaN          -1
1336  55.0          0               2      836.0           1
880   32.0          0               1        NaN          -1
237   52.0          0               0      771.0           2
345   23.0          0               2        NaN           1

      DistanceFromHome  Education  EducationField  EnvironmentSatisfaction  \
1024               2.0        4.0              -1                      NaN
93                 1.0        3.0              -1                      3.0
525                3.0        2.0              -1                      1.0
1450               NaN        NaN              -1                      3.0
922                4.0        2.0               1                      3.0
…                  …         …              …                      …
216               26.0        4.0              -1                      3.0
1336               2.0        NaN               5                      2.0
880                NaN        3.0               4                      3.0
237                2.0        NaN              -1                      1.0
345               26.0        NaN               1                      3.0

      Gender  …  PerformanceRating  RelationshipSatisfaction  \
1024       0  …                3.0                       2.0
93         1  …                3.0                       3.0
525        0  …                3.0                       1.0
1450       0  …                NaN                       NaN
922        1  …                3.0                       4.0
…        …  …                 …                       …
216        0  …                3.0                       3.0
1336      -1  …                4.0                       2.0
880        0  …                4.0                       3.0
```

```
237        1  …                 NaN                        NaN
345        1  …                 3.0                        3.0

      StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
1024               2.0               26.0                    2.0
93                 1.0               21.0                    5.0
525                NaN                4.0                    3.0
1450               0.0                9.0                    2.0
922                2.0               26.0                    4.0
…                  …                 …                    …
216                0.0                9.0                    5.0
1336               1.0               19.0                    2.0
880                1.0                2.0                    2.0
237                0.0               33.0                    NaN
345                2.0                NaN                    2.0

      WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
1024              4.0             NaN                 NaN
93                2.0            10.0                 9.0
525               NaN             NaN                 2.0
1450              3.0             9.0                 0.0
922               2.0             NaN                 NaN
…                 …              …                  …
216               2.0             6.0                 3.0
1336              4.0             5.0                 NaN
880               3.0             NaN                 2.0
237               4.0            33.0                 7.0
345               2.0             4.0                 2.0

      YearsSinceLastPromotion  YearsWithCurrManager
1024                      5.0                   6.0
93                        NaN                   NaN
525                       2.0                   0.0
1450                      1.0                   7.0
922                       NaN                  13.0
…                         …                    …
216                       0.0                   1.0
1336                      NaN                   NaN
880                       2.0                   NaN
237                      15.0                  12.0
345                       0.0                   NaN

[1323 rows x 31 columns]
```

[ ]:
```
df_test
```

[ ]:
```
       Age  Attrition  BusinessTravel  DailyRate  Department  \
1239  31.0          0               1      163.0           1
1014   NaN          0               2        NaN           1
259   31.0          1               1      307.0          -1
759   45.0          0               2        NaN           0
1443  42.0          0               2      300.0          -1
…      …          …               …        …          …
583   34.0          0               2     1111.0           2
476    NaN          0               2      823.0           1
219   54.0          0               2     1147.0          -1
466   41.0          0               2     1276.0           2
1308  38.0          0               2      723.0           2

      DistanceFromHome  Education  EducationField  EnvironmentSatisfaction  \
1239              24.0        1.0               5                      NaN
1014               8.0        5.0               1                      NaN
259               29.0        2.0               3                      NaN
759               24.0        4.0               3                      2.0
1443               2.0        3.0               1                      NaN
…                  …        …             …                      …
583                8.0        2.0               1                      3.0
476               17.0        2.0               4                      4.0
219                3.0        3.0              -1                      4.0
466                2.0        5.0              -1                      2.0
1308               NaN        4.0               2                      2.0

      Gender  …  PerformanceRating  RelationshipSatisfaction  \
1239       0  …                4.0                       NaN
1014       0  …                3.0                       3.0
259        1  …                NaN                       2.0
759       -1  …                3.0                       1.0
1443       1  …                NaN                       1.0
…        …  …                …                       …
583       -1  …                3.0                       2.0
476        1  …                4.0                       4.0
219        0  …                3.0                       4.0
466        0  …                NaN                       2.0
1308      -1  …                4.0                       1.0

      StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
1239               0.0                9.0                    3.0
1014               0.0                9.0                    3.0
259                NaN                6.0                    2.0
759                0.0                6.0                    3.0
1443               0.0               24.0                    2.0
…                  …                …                    …
583                1.0                6.0                    1.0
```

```
476                NaN                1.0                    2.0
219                1.0                NaN                    4.0
466                1.0               22.0                    NaN
1308               2.0               20.0                    4.0

      WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
1239              2.0             5.0                 4.0
1014              4.0             3.0                 2.0
259               4.0             5.0                 4.0
759               3.0             6.0                 NaN
1443              2.0            22.0                 6.0
…                 …              …                  …
583               3.0             3.0                 2.0
476               3.0             NaN                 0.0
219               3.0             NaN                 NaN
466               3.0            18.0                16.0
1308              2.0             4.0                 2.0

      YearsSinceLastPromotion  YearsWithCurrManager
1239                      NaN                   4.0
1014                      NaN                   0.0
259                       1.0                   4.0
759                       0.0                   4.0
1443                      NaN                   NaN
…                       …                    …
583                       NaN                   NaN
476                       0.0                   NaN
219                       NaN                   NaN
466                       NaN                   NaN
1308                      0.0                   3.0

[147 rows x 31 columns]
```

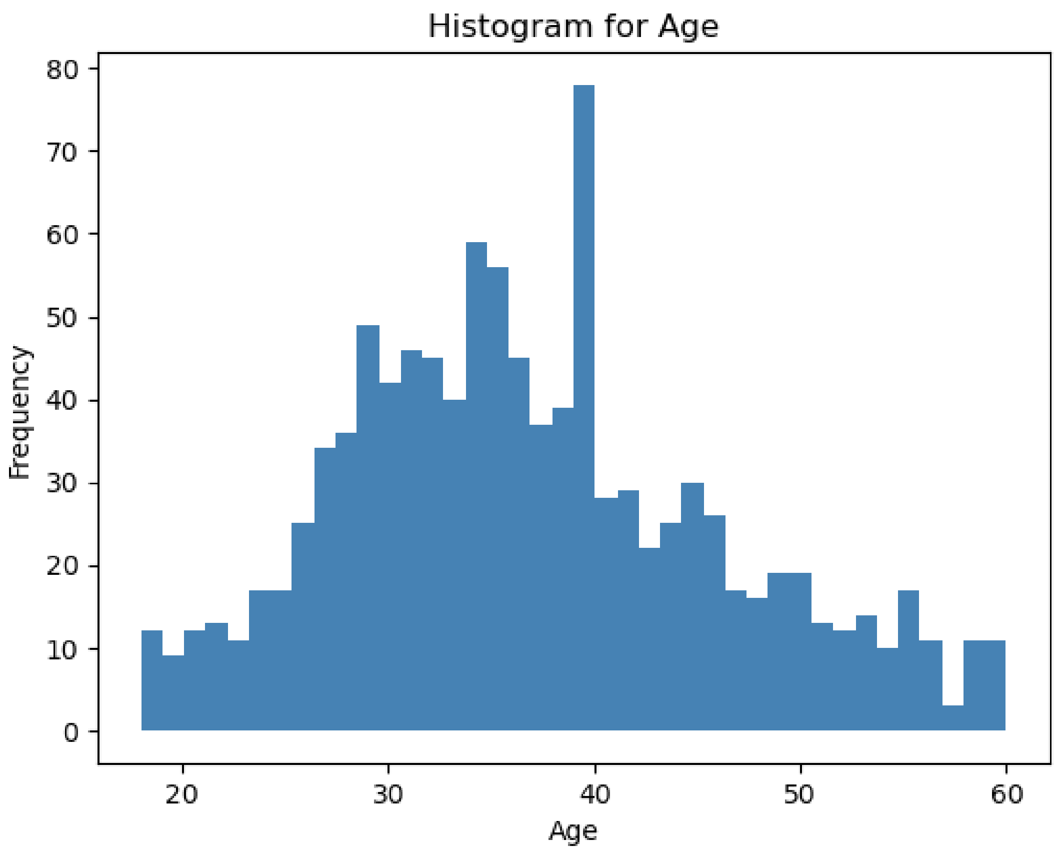
1.0.5 Display histogram of each feature

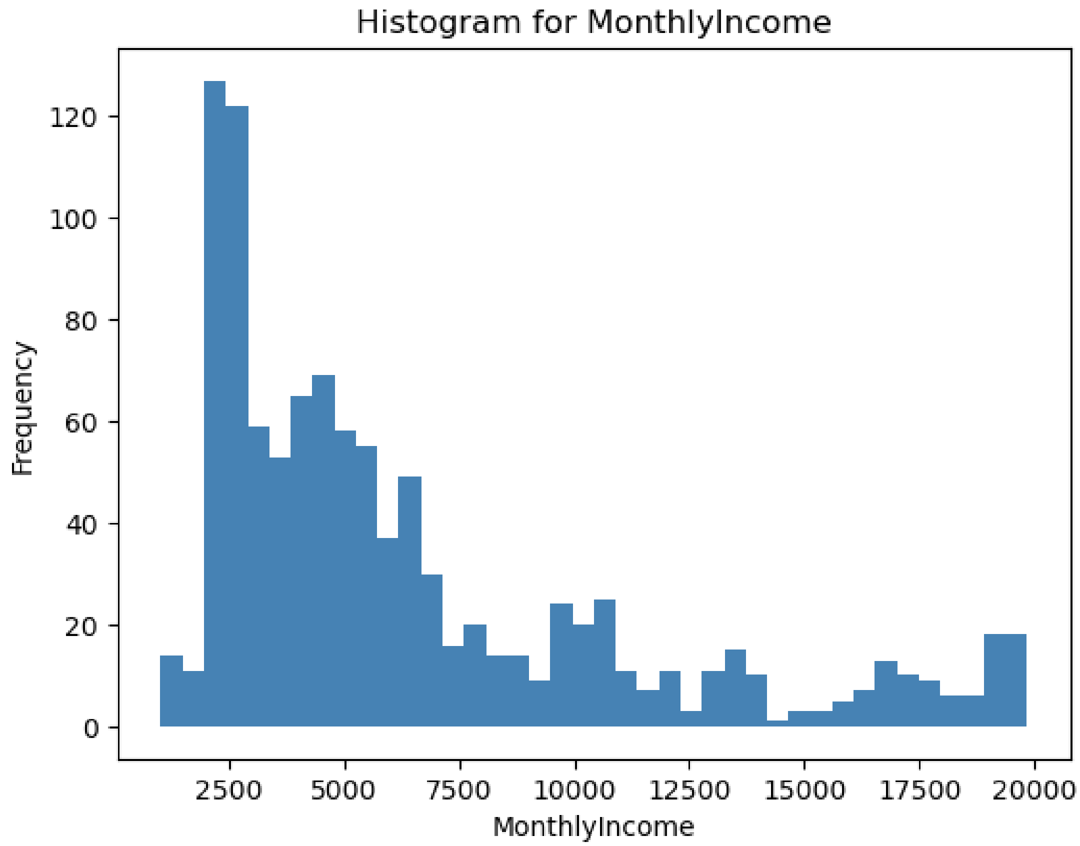
[ ]:
```
def display_histogram(df, col_name, n_bin=40):
    # Filter the DataFrame for the specified column, dropping NaN values
    col_no_nan = df[col_name].dropna()

    # Bin the data into equally spaced bins
    hist, bin_edge = np.histogram(col_no_nan, bins=n_bin)

    # Plot the histogram
    plt.fill_between(bin_edge.repeat(2)[1:-1], hist.repeat(2),␣
 ↪facecolor='steelblue')
    plt.title(f"Histogram for {col_name}")
    plt.xlabel(col_name)
```

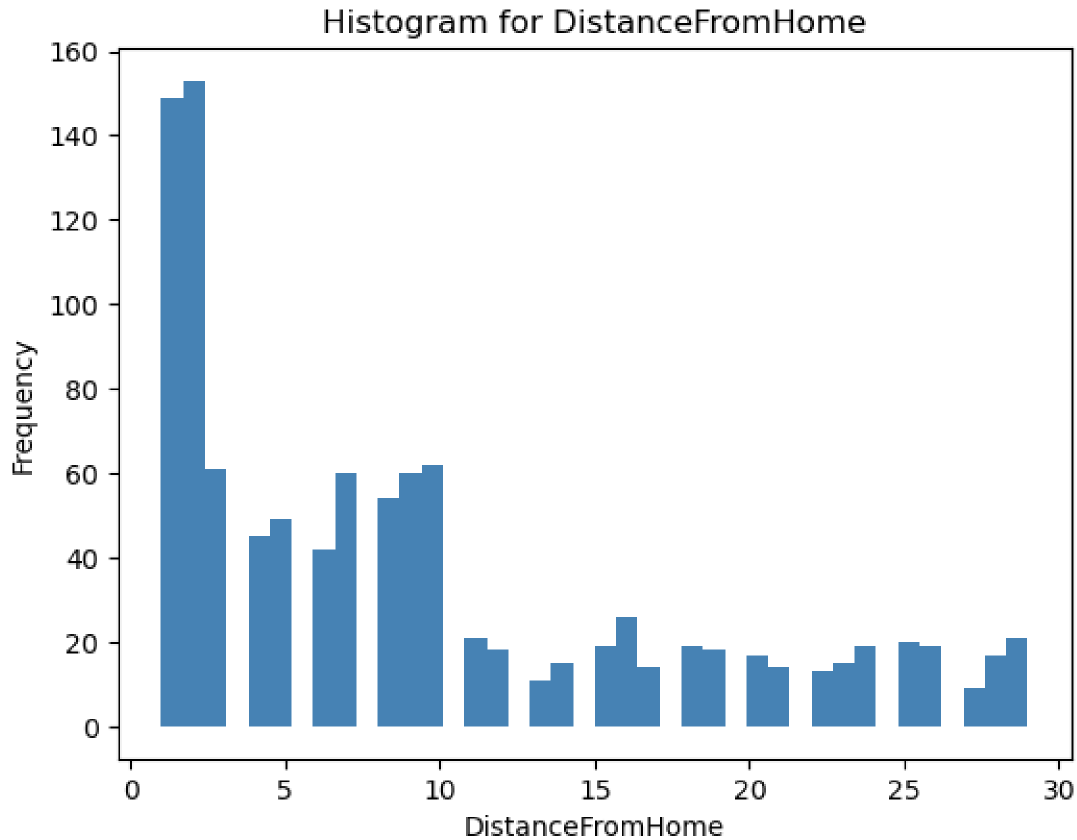
```
    plt.ylabel("Frequency")
    plt.show()
```

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?

[ ]:
```
# by Feature
display_histogram(df_train, "Age")
display_histogram(df_train, "MonthlyIncome")
display_histogram(df_train, "DistanceFromHome")
```




Histogram for MonthlyIncome
Frequency
0
20
40
60
80
100
120
2500
5000
7500
10000
12500
15000
17500
20000
MonthlyIncome



How many bins have zero counts?

```
[ ]: def count_zero_bins(df, col_name, n_bin=40):
         # Filter the DataFrame for the specified column, dropping NaN values
         col_no_nan = df[col_name].dropna()

         # Bin the data into equally spaced bins
         hist, bin_edge = np.histogram(col_no_nan, bins=n_bin)

         # Count the number of bins with zero counts
         zero_bins_count = np.count_nonzero(hist == 0)

         return zero_bins_count

     # Calculate
     zero_bins_age = count_zero_bins(df_train, "Age")
     zero_bins_income = count_zero_bins(df_train, "MonthlyIncome")
     zero_bins_distance = count_zero_bins(df_train, "DistanceFromHome")

     # Display results
```

```
print("Number of zero bins for Age:", zero_bins_age)
print("Number of zero bins for MonthlyIncome:", zero_bins_income)
print("Number of zero bins for DistanceFromHome:", zero_bins_distance)

print("Total number of zero bins:", zero_bins_age + zero_bins_income +␣
 ↪zero_bins_distance)
```

```
Number of zero bins for Age: 0
Number of zero bins for MonthlyIncome: 0
Number of zero bins for DistanceFromHome: 11
Total number of zero bins: 11
```

Do you think this is a good discretization? Why?

'Age' and 'MonthlyIncome' are okay, but 'DistanceFromHome' is not good because it has 11 zero bins.

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

Can we use a Gaussian to estimate this histogram? Why?

Can use a Gaussian distribution for 'Age,'
but for 'MonthlyIncome' and 'DistanceFromHome,' it might not be suitable;
a beta distribution is likely more appropriate.

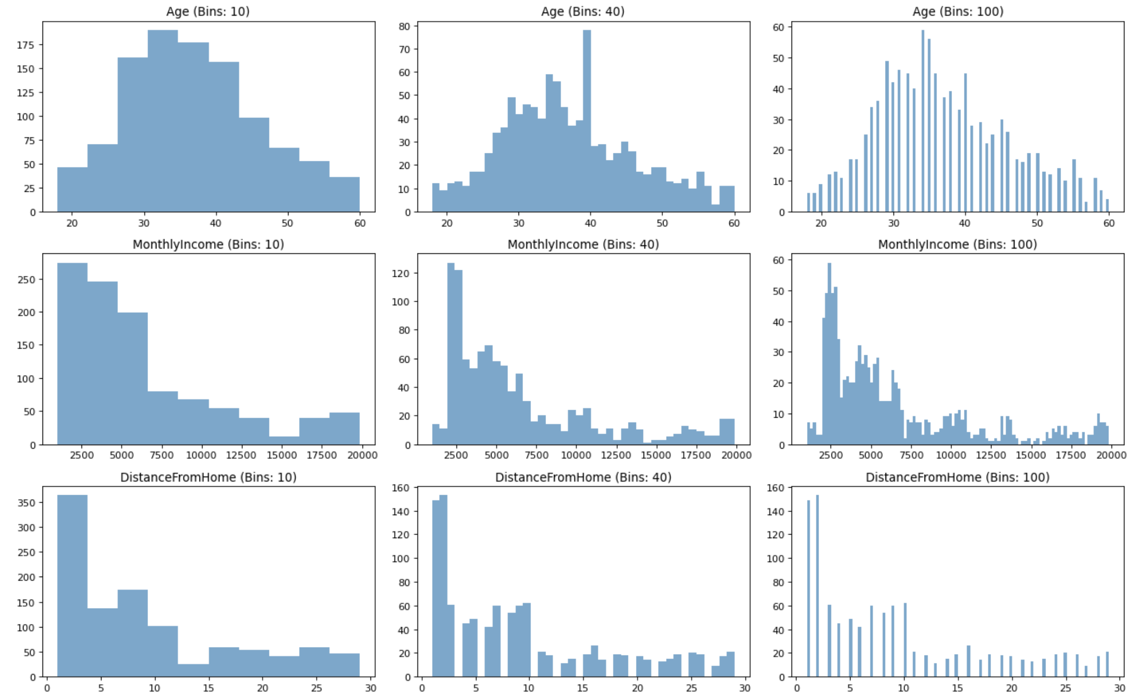
What about a Gaussian Mixture Model (GMM)?

Using Gaussian Mixture Model (GMM) may or may not be appropriate because, from the histogram, it seems there could be either one or multiple hidden Gaussians

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 display_histogram_subplot(df, col_names, bins_list):
    fig, axs = plt.subplots(ncols=len(bins_list),
                            nrows=len(col_names),
                            figsize=(16, 10),
                            dpi=80)

    for i, bins in enumerate(bins_list):
        for j, col in enumerate(col_names):
            axs[j][i].set_title(f"{col} (Bins: {bins})")
            axs[j][i].hist(df[col], bins=bins, color='steelblue', alpha=0.7)

    plt.tight_layout()
    plt.show()

# Example usage:
```

```
bins = [10, 40, 100]
col_names = ["Age", "MonthlyIncome", "DistanceFromHome"]
display_histogram_subplot(df_train, col_names, bins)
```



Which bin size is most sensible for each features? Why?

Age: Divided into 10 bins, appears to follow a normal distribution.
MonthlyIncome: Divided into 40 bins, others are either too coarse or too fine.
DistanceFromHome: Divided into 10 bins, others have zero bins.

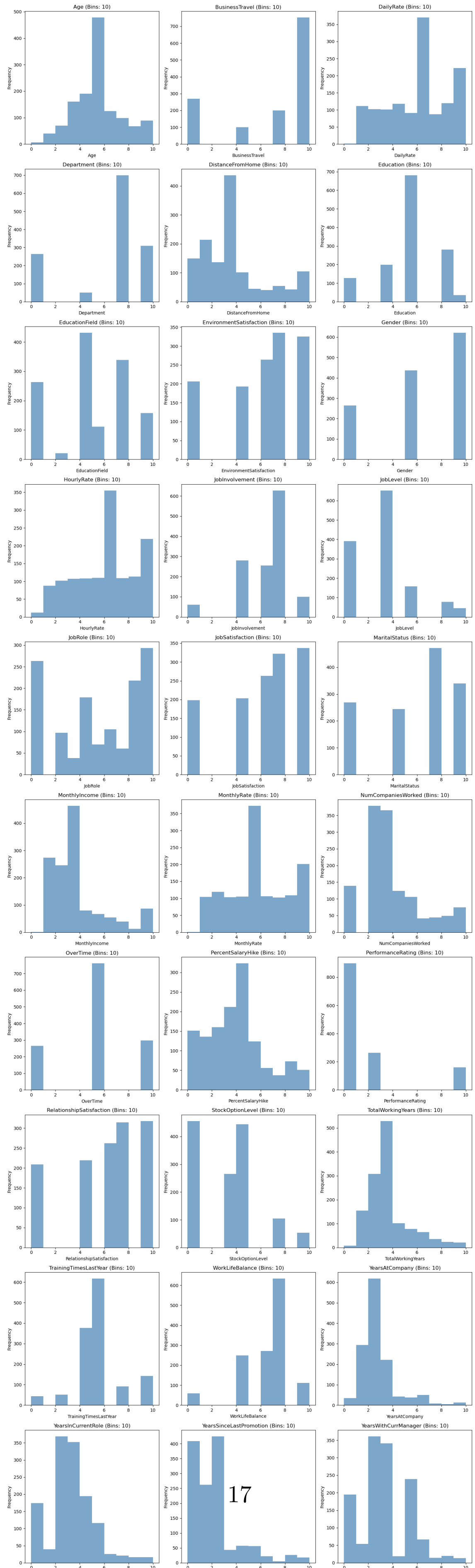
1.0.9 T7. For the rest of the features, which one should be discretized in order to be modeled by histograms? What are the criteria for choosing whether we should discretize a feature or not? Answer this and discretize those features into 10 bins each. In other words, figure out the bin edge for each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

Using all features in df_train that has alraedy drop unnesscary features and encoded, except 'Attrition'

[ ]:
```
def discretized_histograms(df, num_bins=10, show=True):
    """
    Plots and discretized histograms for each continuous feature in the␣
 ↪DataFrame.
    ! Data transformation is applied

    Parameters:
```

```python
    - df (pd.DataFrame): The DataFrame containing the features.
    - num_bins (int): Number of bins to use for histogram calculation.
    - show (bool): Whether to display the plot. If False, returns the figure.

    Returns:
    None or plt.Figure: If show is False, returns the matplotlib figure.
    """

    # Identify features
    features = df.drop("Attrition", axis=1).columns

    # Calculate the number of rows needed based on the number of features and 3␣
↪columns
    num_rows = (len(features) + 2) // 3  # Adding 2 and using integer division

    # Create subplots
    if show:
        fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5 *␣
↪num_rows))

        # Flatten the axs array for easier indexing
        axs = axs.flatten()

    # Discretize each continuous feature and plot
    for i, feature in enumerate(features):
        # Check for NaN values and replace them with the mean
        df[feature].fillna(df[feature].mean(), inplace=True)

        # Compute bin edges
        bin_edges = np.histogram_bin_edges(df[feature], bins=num_bins)

        # Convert to discrete values using digitize
        df[feature] = np.digitize(df[feature], bin_edges, right=True)

        if show:
            # Plot the histogram
            axs[i].hist(df[feature], bins=num_bins, color='steelblue', alpha=0.
↪7)
            axs[i].set_title(f"{feature} (Bins: {num_bins})")
            axs[i].set_xlabel(feature)
            axs[i].set_ylabel("Frequency")

    # Remove empty subplots
    if show:
        for i in range(len(features), len(axs)):
            fig.delaxes(axs[i])
```

```
        plt.tight_layout()
        plt.show()


# Example usage:
discretized_histograms(df_train)
discretized_histograms(df_test, show=False)
```


Age (Bins: 10)
BusinessTravel (Bins: 10)
DailyRate (Bins: 10)
Department (Bins: 10)
DistanceFromHome (Bins: 10)
Education (Bins: 10)
EducationField (Bins: 10)
EnvironmentSatisfaction (Bins: 10)
Gender (Bins: 10)
HourlyRate (Bins: 10)
JobInvolvement (Bins: 10)
JobLevel (Bins: 10)
JobRole (Bins: 10)
JobSatisfaction (Bins: 10)
MaritalStatus (Bins: 10)
MonthlyIncome (Bins: 10)
MonthlyRate (Bins: 10)
NumCompaniesWorked (Bins: 10)
OverTime (Bins: 10)
PercentSalaryHike (Bins: 10)
PerformanceRating (Bins: 10)
RelationshipSatisfaction (Bins: 10)
StockOptionLevel (Bins: 10)
TotalWorkingYears (Bins: 10)
TrainingTimesLastYear (Bins: 10)
WorkLifeBalance (Bins: 10)
YearsAtCompany (Bins: 10)
YearsInCurrentRole (Bins: 10)
YearsSinceLastPromotion (Bins: 10)
YearsWithCurrManager (Bins: 10)
Frequency

1.1 The MLE for the likelihood distribution of discretized histograms

1.1.1 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.

What kind of distribution should we use to model histograms? (Answer a distribution name)

Multinomial Distribution, multiple discrete outcomes, each with its own probability.

What is the MLE for the likelihood distribution? (Describe how to do the MLE)

$$p_j = P(x_j) = \frac{x_j}{n}$$
$$f(x_1, \dots, x_n \mid p_1, \dots, p_m) = \frac{n!}{\prod_{j=1}^{m} x_i!} \prod_{j=1}^{m} p_j^{x_j}$$

Log-Likelihood function of Multinomial

$$\begin{aligned} loglik(p_1, \dots, p_m) &= log[f(x_1, \dots, x_m \mid p_1, \dots, p_m)] \\ &= log(n!) - \sum_{j=1}^{m} log(x_j!) + \sum_{j=1}^{m} x_j log(p_j) \end{aligned}$$

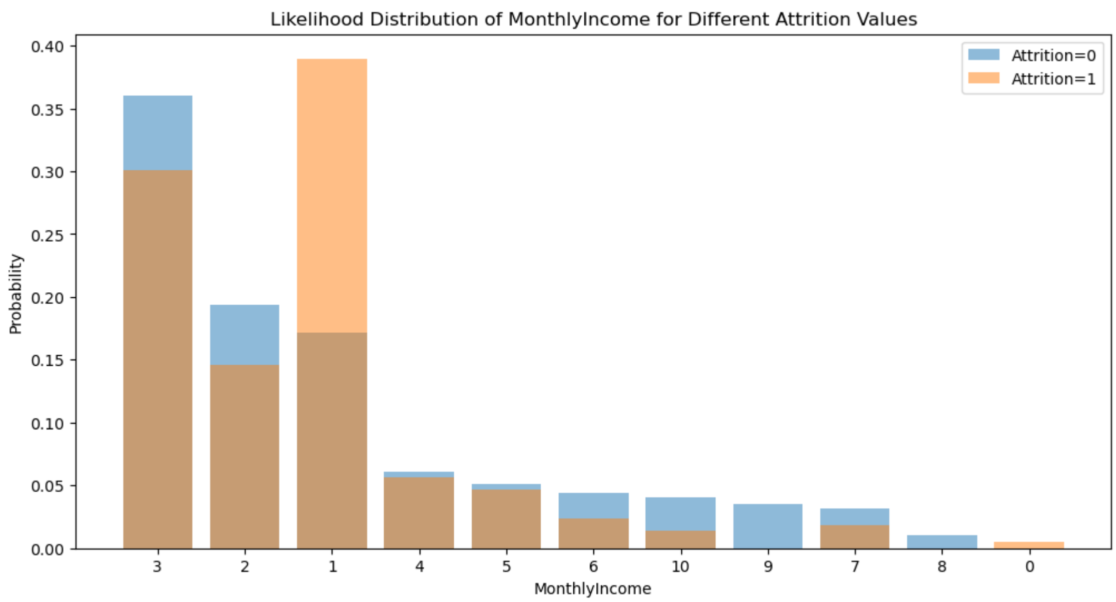
- Maximum achieved when differential is zero
- Constraint: $\sum_{j=1}^{m} p_j = 1$
- Apply method of Lagrange multipliers

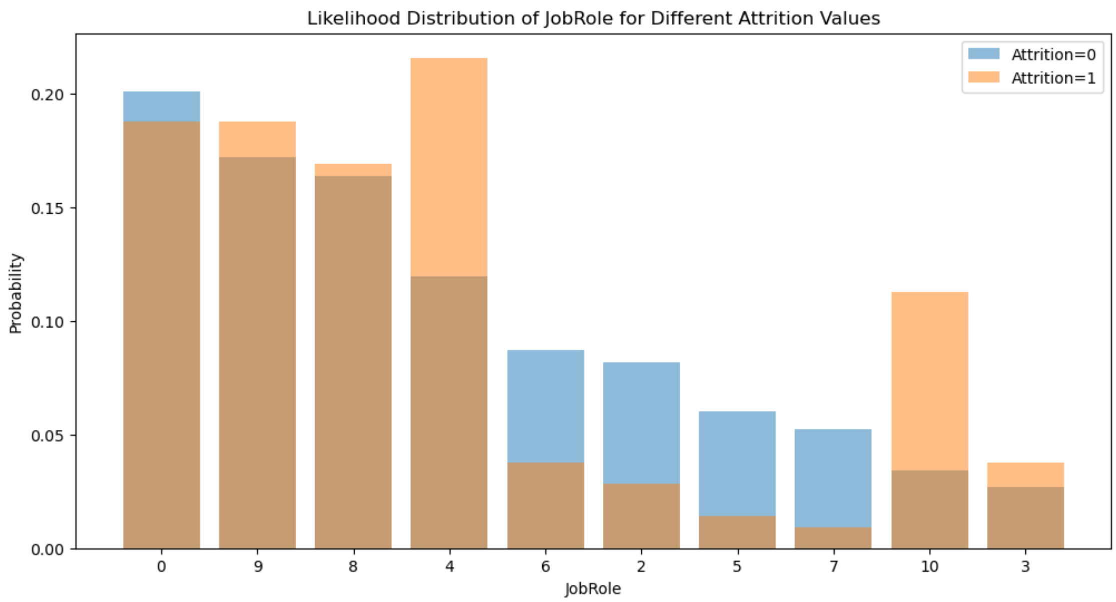
$$\therefore \hat{p}_j = \frac{x_j}{n} \quad ; \; j = 1, 2, \dots, m$$

Plot the likelihood distributions of MonthlyIncome, JobRole, HourlyRate, and MaritalStatus for different Attrition values.

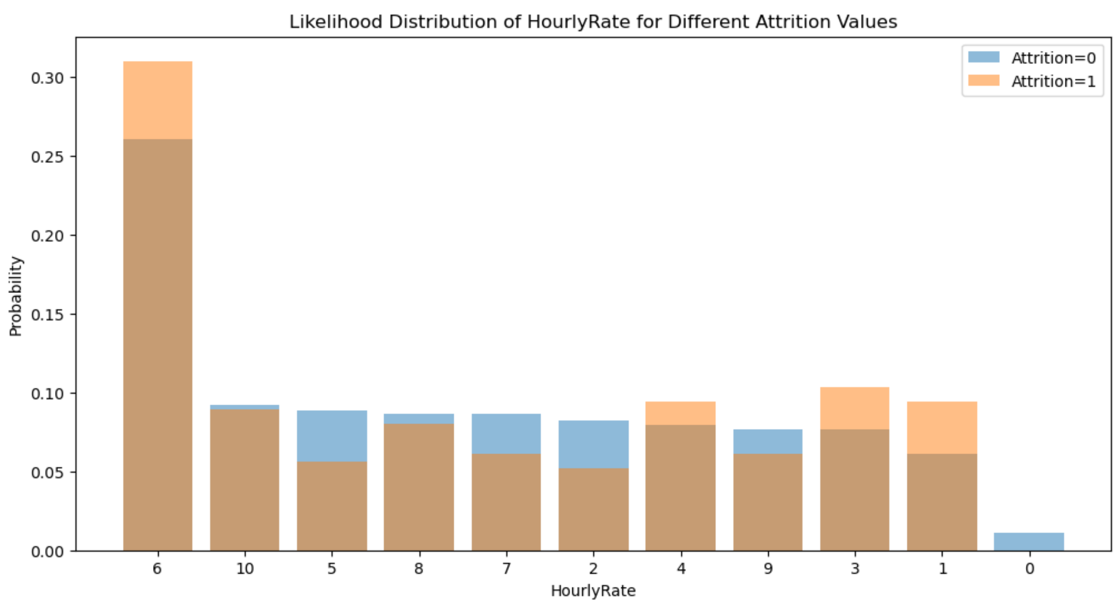
```
[ ]: # Features to plot
features_to_plot = ["MonthlyIncome", "JobRole", "HourlyRate", "MaritalStatus"]

# Loop through each feature and plot likelihood distributions
for feature in features_to_plot:
    plt.figure(figsize=(12, 6))

    for attrition_value in df_train["Attrition"].unique():
        subset_data = df_train[df_train["Attrition"] ==␣
 ↪attrition_value][feature]
        category_counts = subset_data.value_counts()
```

```
        # Calculate MLE probabilities for each category
        mle_probabilities = category_counts / category_counts.sum()

        # Plot the likelihood distribution
        plt.bar(category_counts.index.astype(str), mle_probabilities,␣
↪label=f'Attrition={attrition_value}', alpha=0.5)

    plt.title(f"Likelihood Distribution of {feature} for Different Attrition␣
↪Values")
    plt.xlabel(feature)
    plt.ylabel("Probability")
    plt.legend()
    plt.show()
```

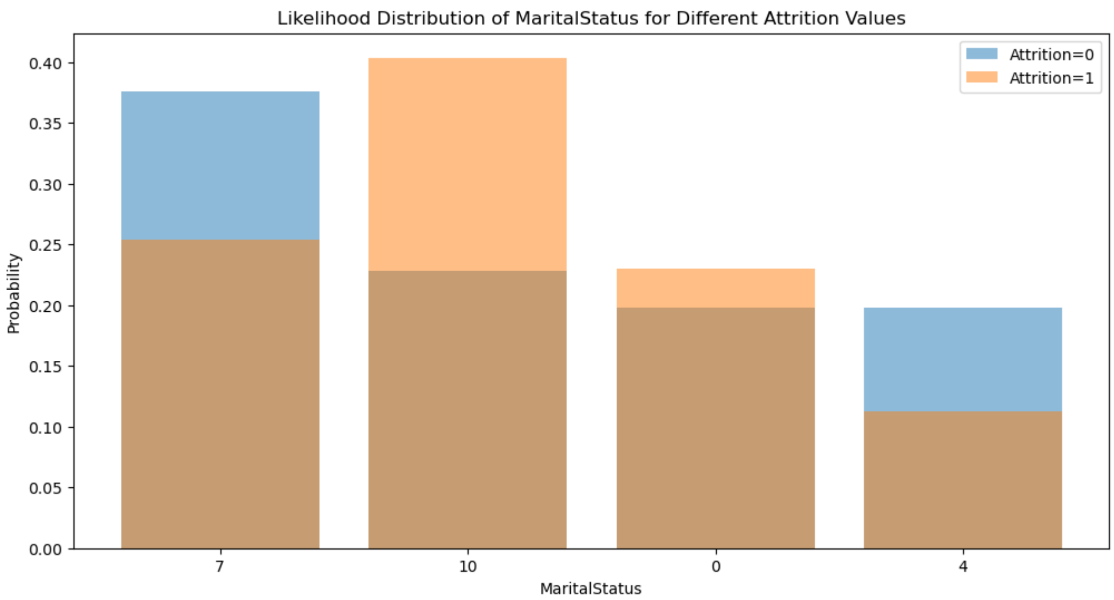




Likelihood Distribution of JobRole for Different Attrition Values
Attrition=0
Attrition=1
Probability
0.00
0.05
0.10
0.15
0.20
0
9
8
4
6
2
5
7
10
3
JobRole



Likelihood Distribution of HourlyRate for Different Attrition Values
Attrition=0
Attrition=1
Probability
0.00
0.05
0.10
0.15
0.20
0.25
0.30
6
10
5
8
7
2
4
9
3
1
0
HourlyRate



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

```
[ ]: # Calculate prior
def calculate_prior(df, cls):
  return df.loc[df["Attrition"] == cls, "Attrition"].count() / df.shape[0]

priorClass0 = calculate_prior(df_train, 0)
priorClass1 = calculate_prior(df_train, 1)
print(f"Prior class 0: {priorClass0}")
print(f"Prior class 1: {priorClass1}")
```

```
Prior class 0: 0.8390022675736961
Prior class 1: 0.16099773242630386
```

1.2 Naive Bayes classification

1.2.1 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.

3 solutions 1. Use a very small value instead of zero (flooring) 2. Smooth the values using counts from other observations (smoothing) 3. Use priors (MAP adaptation)

```
[ ]: def apply_flooring_np(arr, epsilon=1e-10):
    """
    Apply flooring to replace zero values in a NumPy array with a small epsilon␣
  ↪value.
```

```
    Parameters:
    - arr (np.ndarray): The NumPy array containing probabilities.
    - epsilon (float): The small value to replace zero.

    Returns:
    - arr_floored (np.ndarray): The NumPy array with zero values replaced by␣
↪epsilon.
    """
    return np.where(arr == 0, epsilon, arr)
```

1.2.2 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.

[ ]:
```
from SimpleBayesClassifier import SimpleBayesClassifier
```

[ ]:
```
selected_columns = ['MonthlyIncome', 'JobRole', 'HourlyRate', 'MaritalStatus']

x_train = df_train.drop(columns='Attrition').to_numpy() # all features
# x_train = df_train[selected_columns].to_numpy()
x_train = apply_flooring_np(x_train)
y_train = df_train["Attrition"].to_numpy()

x_test = df_test.drop(columns='Attrition').to_numpy() # all features
# x_test = df_test[selected_columns].to_numpy()
x_test = apply_flooring_np(x_test)
y_test = df_test["Attrition"].to_numpy()
```

[ ]:
```
n_pos = np.count_nonzero(y_train == 1)
n_neg = np.count_nonzero(y_train == 0)

model = SimpleBayesClassifier(n_pos=n_pos, n_neg=n_neg)
```

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

check_prior()
```

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

[ ]:
```
([(array([0.02162162, 0.04414414, 0.11711712, 0.14324324, 0.36846847,
          0.1045045 , 0.07927928, 0.05315315, 0.04324324, 0.02522523]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.20540541, 0.        , 0.        , 0.08378378, 0.        ,
          0.        , 0.13513514, 0.        , 0.        , 0.57567568]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.0960961 , 0.07807808, 0.08708709, 0.1001001 , 0.07707708,
          0.3033033 , 0.07707708, 0.1021021 , 0.1031031 , 0.08708709]),
   array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
  (array([0.1963964 , 0.        , 0.        , 0.03693694, 0.        ,
          0.        , 0.54504505, 0.        , 0.        , 0.22162162]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.28558559, 0.10540541, 0.33693694, 0.07477477, 0.03063063,
          0.02702703, 0.04054054, 0.02882883, 0.03603604, 0.03423423]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.09369369, 0.        , 0.14774775, 0.        , 0.51351351,
          0.        , 0.        , 0.21621622, 0.        , 0.02882883]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.1963964 , 0.01441441, 0.        , 0.33333333, 0.07657658,
          0.        , 0.26666667, 0.        , 0.04504505, 0.06756757]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.14594595, 0.        , 0.        , 0.14414414, 0.        ,
          0.19279279, 0.26486486, 0.        , 0.        , 0.25225225]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.2009009 , 0.        , 0.        , 0.        , 0.33333333,
          0.        , 0.        , 0.        , 0.        , 0.46576577]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.07207207, 0.08198198, 0.07657658, 0.07927928, 0.08828829,
          0.26036036, 0.08648649, 0.08648649, 0.07657658, 0.09189189]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.03513514, 0.        , 0.        , 0.20810811, 0.        ,
          0.1954955 , 0.47927928, 0.        , 0.        , 0.08198198]),
   array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
  (array([0.25765766, 0.        , 0.51351351, 0.        , 0.12612613,
          0.        , 0.        , 0.06576577, 0.        , 0.03693694]),
```

```
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.2009009 , 0.08198198, 0.02702703, 0.11981982, 0.06036036,
        0.08738739, 0.05225225, 0.16396396, 0.17207207, 0.03423423]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.14504505, 0.        , 0.        , 0.15315315, 0.        ,
        0.1972973 , 0.23873874, 0.        , 0.        , 0.26576577]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.1981982 , 0.        , 0.        , 0.1981982 , 0.        ,
        0.        , 0.37567568, 0.        , 0.        , 0.22792793]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.19019019, 0.21521522, 0.4004004 , 0.06806807, 0.05705706,
        0.04904905, 0.03503504, 0.01201201, 0.03903904, 0.04504505]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.07567568, 0.0954955 , 0.08108108, 0.07297297, 0.28018018,
        0.08198198, 0.07747748, 0.08198198, 0.09099099, 0.06216216]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.10990991, 0.28018018, 0.28468468, 0.1       , 0.08198198,
        0.02612613, 0.03063063, 0.03333333, 0.02612613, 0.02702703]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.1972973 , 0.        , 0.        , 0.        , 0.61531532,
        0.        , 0.        , 0.        , 0.        , 0.18738739]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.20720721, 0.12072072, 0.16306306, 0.25135135, 0.09459459,
        0.04414414, 0.02702703, 0.05675676, 0.01711712, 0.01801802]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.68108108, 0.2       , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.11891892]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.14954955, 0.        , 0.        , 0.16846847, 0.        ,
        0.19189189, 0.24504505, 0.        , 0.        , 0.24504505]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.30720721, 0.        , 0.2027027 , 0.36486486, 0.        ,
        0.        , 0.08468468, 0.        , 0.        , 0.04054054]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.0990991 , 0.22612613, 0.41351351, 0.08198198, 0.06126126,
        0.05225225, 0.02882883, 0.02072072, 0.01351351, 0.0027027 ]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.02972973, 0.03873874, 0.        , 0.28018018, 0.47297297,
        0.        , 0.06306306, 0.        , 0.08198198, 0.03333333]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.03603604, 0.        , 0.        , 0.18648649, 0.        ,
        0.2045045 , 0.49189189, 0.        , 0.        , 0.08108108]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.21981982, 0.47927928, 0.17387387, 0.03513514, 0.03063063,
        0.04234234, 0.00540541, 0.0045045 , 0.00540541, 0.0036036 ]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.14414414, 0.27747748, 0.26486486, 0.15315315, 0.0954955 ,
```

```
        0.02162162, 0.01801802, 0.01261261, 0.00900901, 0.0036036 ]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.5036036 , 0.31891892, 0.03513514, 0.04504505, 0.04144144,
        0.01891892, 0.0036036 , 0.02162162, 0.0009009 , 0.01081081]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.15945946, 0.28108108, 0.26306306, 0.01441441, 0.18918919,
        0.05315315, 0.01171171, 0.01801802, 0.00630631, 0.0036036 ]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf]))],
[(array([0.10328638, 0.09859155, 0.14553991, 0.14553991, 0.32394366,
        0.03755869, 0.04694836, 0.03755869, 0.02347418, 0.03755869]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.1971831 , 0.        , 0.        , 0.03755869, 0.        ,
        0.        , 0.23474178, 0.        , 0.        , 0.53051643]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.07511737, 0.11267606, 0.0657277 , 0.08450704, 0.0657277 ,
        0.31455399, 0.04694836, 0.08450704, 0.06103286, 0.08920188]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.21126761, 0.        , 0.        , 0.04225352, 0.        ,
        0.        , 0.44600939, 0.        , 0.        , 0.30046948]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.21596244, 0.08920188, 0.29577465, 0.08450704, 0.05164319,
        0.04694836, 0.04225352, 0.04694836, 0.08450704, 0.04225352]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.10798122, 0.        , 0.15962441, 0.        , 0.52112676,
        0.        , 0.        , 0.19248826, 0.        , 0.01877934]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.21126761, 0.02347418, 0.        , 0.28638498, 0.12206573,
        0.        , 0.20187793, 0.        , 0.04225352, 0.11267606]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.20657277, 0.        , 0.        , 0.15492958, 0.        ,
        0.23474178, 0.19248826, 0.        , 0.        , 0.21126761]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.1971831 , 0.        , 0.        , 0.        , 0.31455399,
        0.        , 0.        , 0.        , 0.        , 0.48826291]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.10432968, 0.05738132, 0.11476265, 0.10432968, 0.06259781,
        0.34428795, 0.06781429, 0.08868023, 0.06781429, 0.0991132 ]),
  array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
 (array([0.09859155, 0.        , 0.        , 0.23474178, 0.        ,
        0.17840376, 0.44600939, 0.        , 0.        , 0.04225352]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.49295775, 0.        , 0.38497653, 0.        , 0.08450704,
        0.        , 0.        , 0.01877934, 0.        , 0.01877934]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
 (array([0.18779343, 0.02816901, 0.03755869, 0.21596244, 0.01408451,
        0.03755869, 0.00938967, 0.16901408, 0.18779343, 0.11267606]),
  array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
```

```
(array([0.17370892, 0.        , 0.        , 0.15492958, 0.        ,
        0.20657277, 0.26760563, 0.        , 0.        , 0.1971831 ]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.23004695, 0.        , 0.        , 0.11267606, 0.        ,
        0.        , 0.25352113, 0.        , 0.        , 0.40375587]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.3943662 , 0.14553991, 0.30046948, 0.05633803, 0.04694836,
        0.02347418, 0.01877934, 0.        , 0.        , 0.01408451]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.10954617, 0.06781429, 0.06781429, 0.12519562, 0.32342201,
        0.07824726, 0.08346375, 0.09389671, 0.07303078, 0.08868023]),
 array([-inf,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1,  inf])),
(array([0.07981221, 0.31924883, 0.23474178, 0.06103286, 0.07042254,
        0.05633803, 0.04694836, 0.05633803, 0.02347418, 0.05164319]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.21596244, 0.        , 0.        , 0.        , 0.36619718,
        0.        , 0.        , 0.        , 0.        , 0.41784038]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.26760563, 0.12206573, 0.14553991, 0.20657277, 0.08920188,
        0.03286385, 0.03286385, 0.04694836, 0.02347418, 0.03286385]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.66666667, 0.1971831 , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.13615023]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.20187793, 0.        , 0.        , 0.15023474, 0.        ,
        0.23004695, 0.20187793, 0.        , 0.        , 0.21596244]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.5399061 , 0.        , 0.18779343, 0.18309859, 0.        ,
        0.        , 0.05164319, 0.        , 0.        , 0.03755869]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.24413146, 0.26760563, 0.3286385 , 0.04694836, 0.04694836,
        0.03286385, 0.01408451, 0.00469484, 0.00469484, 0.00938967]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.05164319, 0.04225352, 0.        , 0.30985915, 0.43192488,
        0.        , 0.09859155, 0.        , 0.05164319, 0.01408451]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.08920188, 0.        , 0.        , 0.1971831 , 0.        ,
        0.20657277, 0.4084507 , 0.        , 0.        , 0.09859155]),
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.06259781, 0.37558685, 0.44861763, 0.15127804, 0.01564945,
        0.02086594, 0.01564945, 0.01043297, 0.        , 0.01043297]),
 array([-inf,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,  8.1,  inf])),
(array([0.24517475, 0.0312989 , 0.31820553, 0.30255608, 0.12519562,
        0.05216484, 0.01043297, 0.00521648, 0.01043297, 0.01043297]),
 array([-inf,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,  8.1,  inf])),
(array([0.5258216 , 0.33333333, 0.01877934, 0.03286385, 0.04694836,
        0.00469484, 0.00469484, 0.00938967, 0.00938967, 0.01408451]),
```

```
 array([-inf,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  inf])),
(array([0.33385498, 0.04173187, 0.25560772, 0.25560772, 0.01564945,
        0.15127804, 0.04173187, 0.00521648, 0.        , 0.01043297]),
 array([-inf,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,  8.1,  inf]))])
```

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

```
STAY PARAMETERS
Feature : 0
BINS : [2.5 0.  0.  0.  0.  0.  0.  0.  0.  2.5]
EDGES : [-inf  0.2  0.4  0.6  0.8  1.   1.2  1.4  1.6  1.8  inf]
Feature : 1
BINS : [2.5 0.  0.  0.  0.  0.  0.  0.  0.  2.5]
EDGES : [-inf  1.2  1.4  1.6  1.8  2.   2.2  2.4  2.6  2.8  inf]
```

```
Feature : 2
BINS : [2.5 0.  0.  0.  0.  0.  0.  0.  0.  2.5]
EDGES : [-inf  2.2  2.4  2.6  2.8  3.   3.2  3.4  3.6  3.8  inf]
Feature : 3
BINS : [2.5 0.  0.  0.  0.  0.  0.  0.  0.  2.5]
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 : [2.5 0.  0.  0.  0.  0.  0.  0.  0.  2.5]
EDGES : [-inf  1.2  1.4  1.6  1.8  2.   2.2  2.4  2.6  2.8  inf]
Feature : 1
BINS : [2.5 0.  0.  0.  0.  0.  0.  0.  0.  2.5]
EDGES : [-inf  2.2  2.4  2.6  2.8  3.   3.2  3.4  3.6  3.8  inf]
Feature : 2
BINS : [2.5 0.  0.  0.  0.  0.  0.  0.  0.  2.5]
EDGES : [-inf  3.2  3.4  3.6  3.8  4.   4.2  4.4  4.6  4.8  inf]
Feature : 3
BINS : [2.5 0.  0.  0.  0.  0.  0.  0.  0.  2.5]
EDGES : [-inf  4.2  4.4  4.6  4.8  5.   5.2  5.4  5.6  5.8  inf]
```

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

[ ]:
```
def evaluate(y_test, y_pred, show_result=True):
    y_test = np.array(y_test)
    y_pred = np.array(y_pred)

    # Mask NaN values
    nan_mask = ~np.isnan(y_test) & ~np.isnan(y_pred)
    y_test = y_test[nan_mask]
    y_pred = y_pred[nan_mask]

    # Calculate True Positives, True Negatives, False Positives, False Negatives
    tp = sum((y_test == 1) & (y_pred == 1)) # Use 1(leave) as Positive(+) class
    tn = sum((y_test == 0) & (y_pred == 0)) # Use 0(stay) as Negative(-) class
    fp = sum((y_test == 0) & (y_pred == 1))
    fn = sum((y_test == 1) & (y_pred == 0))

    # Calculate evaluation metrics
    accuracy = (tp + tn) / (tp + tn + fp + fn) if (tp + tn + fp + fn)!=0 else
 ↪(tp + tn)/1e-10
    precision = tp / (tp + fp) if (tp + fp) != 0 else tp / 1e-10
    recall = tp / (tp + fn) if (tp + fn) != 0 else tp / 1e-10
    f1 = 2 * (precision * recall) / (precision + recall) if (precision +
 ↪recall) != 0 else (2 * precision * recall) / 1e-10
    false_positive_rate = fp / (fp + tn) if (fp + tn) != 0 else fp / 1e-10
```

```
    if show_result:
        print("Accuracy:", accuracy)
        print("Precision:", precision)
        print("Recall:", recall)
        print("F1 Score:", f1)
        print("False Positive Rate:", false_positive_rate)

    return accuracy, precision, recall, f1, false_positive_rate
```

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

```
Accuracy: 0.7687074829931972
Precision: 0.32142857142857145
Recall: 0.375
F1 Score: 0.3461538461538462
False Positive Rate: 0.15447154471544716
```

[ ]:
```
(0.7687074829931972,
 0.32142857142857145,
 0.375,
 0.3461538461538462,
 0.15447154471544716)
```

1.2.3 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)
```

[ ]:
```
([(5.1216216216219825, 1.896079663690412),
  (7.037837837858379, 3.9998210332136908),
  (5.646846846846847, 2.612093145001865),
  (6.179279279298919, 3.3581048150388924),
  (3.18828828829973, 2.5630161967738347),
  (5.028828828838198, 2.366256304489039),
  (4.692792792812433, 2.958914206462426),
  (6.109909909924505, 3.1811085085323296),
  (6.324324324344415, 3.861713576952009),
  (5.681081081082162, 2.6109786751450543),
  (6.180180180183694, 1.9636250637316426),
  (3.0666666666924325, 2.5046023702126856),
  (5.118918918939009, 3.3697518691866386),
  (6.1252252252397295, 3.210847842551065),
  (5.701801801821622, 3.444056951447781),
  (3.4351351351351354, 2.3192942639172363),
  (5.3324324324325225, 2.5504517233583175),
  (3.3675675675785586, 2.3206927422472705),
  (4.95045045047018, 3.100751837709882),
  (3.54864864865964, 2.361262636140796),
```

```
 (1.5891891892572971, 3.1884744961563714),
 (5.990990991005946, 3.197396168916343),
 (3.065765765796486, 2.5429994170341543),
 (3.252252252252613, 1.7240663668395595),
 (5.0756756756786485, 1.9400109694149206),
 (6.227027027030631, 1.9444513779950432),
 (2.3864864864884687, 1.4799553491215574),
 (2.9432432432546847, 1.8047545624974786),
 (1.763063063092973, 1.959657440525224),
 (3.0342342342460364, 1.9490582385179713)],
[(4.370892018780282, 2.198697363511647),
 (7.098591549315493, 3.8425157896164146),
 (5.492957746479343, 2.6138305283765684),
 (6.295774647908452, 3.597316551470779),
 (3.9107981220760566, 2.883745253461901),
 (4.8122065727807515, 2.3461968655404117),
 (4.723004694856808, 3.162765531507217),
 (5.48826291081878, 3.3715327571490534),
 (6.455399061052582, 3.875301769879587),
 (5.530516431924883, 2.5829448229976144),
 (5.553990610338498, 2.3262425201842385),
 (1.9154929577957742, 2.2334197375191165),
 (5.56338028170892, 3.523492163208394),
 (5.704225352130048, 3.195041784602555),
 (6.262910798145071, 3.9213374378665793),
 (2.4553990610333334, 1.7017338452838642),
 (5.323943661971831, 2.5606702901494747),
 (3.784037558693427, 2.6091800086957915),
 (6.0093896713831, 3.8504808678429208),
 (3.4272300469619714, 2.589362552937142),
 (1.755868544667605, 3.3646220474804394),
 (5.553990610348826, 3.3655456051500576),
 (2.0328638498192486, 2.6026078948783353),
 (2.6431924882643187, 1.6399968926834116),
 (4.77934272300986, 1.872244902887518),
 (5.873239436628638, 2.443327086663691),
 (1.9389671361558685, 1.3569618314616148),
 (2.3521126760784035, 1.7314971530314949),
 (1.6150234742145537, 1.981643369867945),
 (2.2676056338328636, 1.9641702292340792)])
```

[ ]:

```python
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
```

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

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

```
Accuracy: 0.8095238095238095
Precision: 0.4
Recall: 0.3333333333333333
F1 Score: 0.3636363636363636
False Positive Rate: 0.0975609756097561
```

[ ]:
```
(0.8095238095238095,
 0.4,
 0.3333333333333333,
 0.3636363636363636,
 0.0975609756097561)
```

1.3 Baseline comparison

1.3.1 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.

[ ]:
```
def random_choice_baseline(y_test):
    # Generate random predictions
    y_random_pred = np.random.randint(2, size=len(y_test))

    # Evaluate the random predictions
    evaluate(y_test, y_random_pred)

# Assuming y_test contains the true labels for the test samples
random_choice_baseline(y_test)
```

```
Accuracy: 0.48299319727891155
Precision: 0.18292682926829268
Recall: 0.625
F1 Score: 0.2830188679245283
```

```
False Positive Rate: 0.5447154471544715
```

1.3.2 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.

[ ]:
```
def majority_rule_baseline(y_train, y_test):
    # Determine the most frequent class in the training set
    majority_class = np.argmax(np.bincount(y_train))

    # Generate predictions using the majority class
    y_pred = np.full_like(y_test, fill_value=majority_class)

    # Evaluate the predictions
    evaluate(y_test, y_pred)

# Assuming y_train contains the true labels for the training set
# and y_test contains the true labels for the test set
majority_rule_baseline(y_train, y_test)
```

```
Accuracy: 0.8367346938775511
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
False Positive Rate: 0.0
```

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

[ ]:
```
# Make predictions using your Naive Bayes classifier
y_pred_nonparametric = model.predict(x_test)
y_pred_parametric = model.gaussian_predict(x_test)

# Evaluate the predictions from your Naive Bayes classifier
print("Non Parametric:")
evaluate(y_test, y_pred_nonparametric)

# Evaluate the predictions from your Naive Bayes classifier
print("\nParametric:")
evaluate(y_test, y_pred_parametric)

# Evaluate the random choice baseline
print("\nRandom Choice Baseline:")
random_choice_baseline(y_test)

# Evaluate the majority rule baseline
print("\nMajority Rule Baseline:")
majority_rule_baseline(y_train, y_test)
```

```
Non Parametric:
Accuracy: 0.7687074829931972
Precision: 0.32142857142857145
Recall: 0.375
F1 Score: 0.3461538461538462
False Positive Rate: 0.15447154471544716

Parametric:
Accuracy: 0.8095238095238095
Precision: 0.4
Recall: 0.3333333333333333
F1 Score: 0.3636363636363636
False Positive Rate: 0.0975609756097561

Random Choice Baseline:
Accuracy: 0.5102040816326531
Precision: 0.125
Recall: 0.3333333333333333
F1 Score: 0.18181818181818182
False Positive Rate: 0.45528455284552843

Majority Rule Baseline:
Accuracy: 0.8367346938775511
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
False Positive Rate: 0.0
```

1.4 Threshold finding

1.4.1 T16. Use the following threshold values

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

[ ]:
```python
# Define the range of threshold values
thresholds = np.arange(-5, 5, 0.05)

best_accuracy = 0
best_accuracy_threshold = None

best_f_score = 0
best_f_score_threshold = None

# Iterate over the threshold values
for threshold in thresholds:
    # Make predictions using the non-parametric model
    y_pred_nonparametric = model.predict(x_test, thresh=threshold)
```

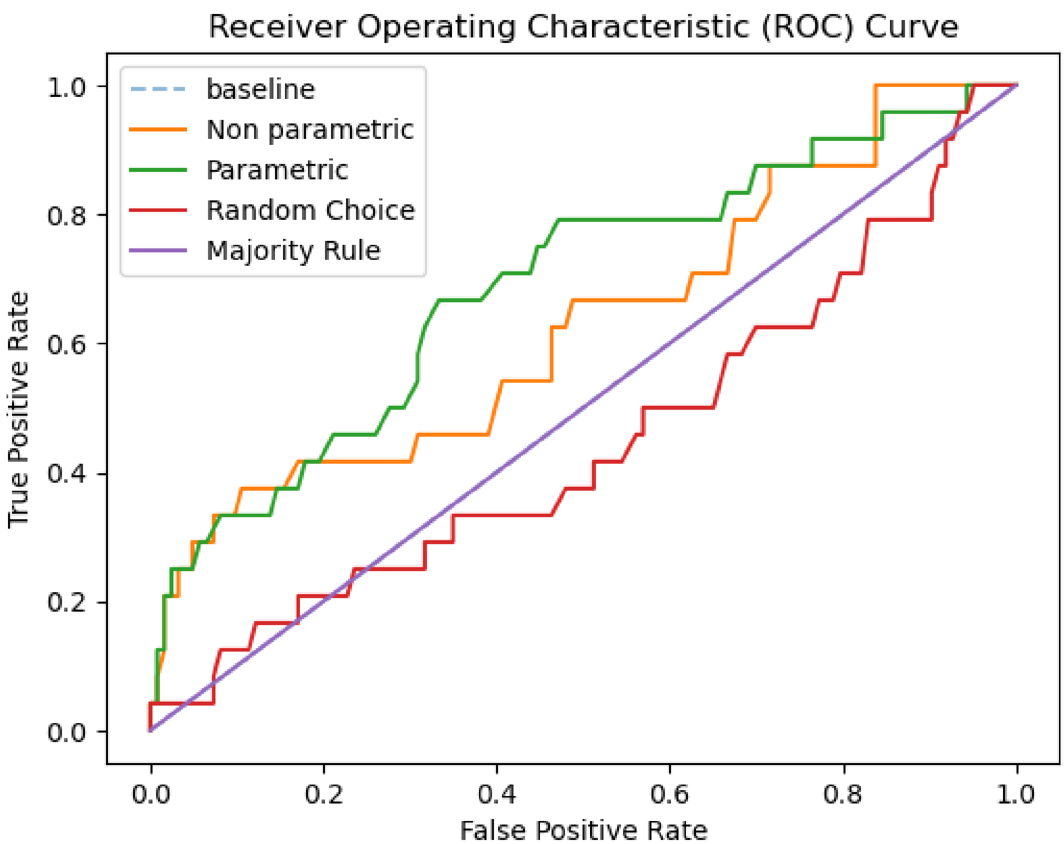
```
    # Make predictions using the parametric model
    y_pred_parametric = model.gaussian_predict(x_test, thresh=threshold)

    # Evaluate the predictions from the non-parametric model
    accuracy_nonparametric, _, _, _, f_score_nonparametric = evaluate(y_test,␣
↪y_pred_nonparametric, show_result=False)

    # Evaluate the predictions from the parametric model
    accuracy_parametric, _, _, _, f_score_parametric = evaluate(y_test,␣
↪y_pred_parametric, show_result=False)

    # Update the best accuracy and corresponding threshold if applicable
    if accuracy_nonparametric > best_accuracy:
        best_accuracy = accuracy_nonparametric
        best_accuracy_threshold = threshold

    # Update the best F-score and corresponding threshold if applicable
    if f_score_nonparametric > best_f_score:
        best_f_score = f_score_nonparametric
        best_f_score_threshold = threshold

# Print the best accuracy and corresponding threshold
print("Best Accuracy (Non-parametric):", best_accuracy)
print("Corresponding Threshold (Non-parametric):", best_accuracy_threshold)

# Print the best F-score and corresponding threshold
print("\nBest F-score (Non-parametric):", best_f_score)
print("Corresponding Threshold (Non-parametric):", best_f_score_threshold)
```

```
Best Accuracy (Non-parametric): 0.8571428571428571
Corresponding Threshold (Non-parametric): 2.849999999999972

Best F-score (Non-parametric): 0.8373983739837398
Corresponding Threshold (Non-parametric): -5.0
```

1.4.2 T17. Plot the RoC of your classifier.

[ ]:
```
pred_proba_nonparametric = model.predict_proba(x_test)
pred_proba_nonparametric = np.array(pred_proba_nonparametric)

pred_proba_parametric = model.gaussian_predict_proba(x_test)
pred_proba_parametric = np.array(pred_proba_parametric)

baseline_random_choice = np.random.rand(y_test.shape[0]) * 10 - 5
baseline_majority_rule = np.array([0] * y_test.shape[0])
```

```python
def plot_roc(y_pred_nonparametric, y_pred_parametric, y_test):
    thresholds = np.arange(-100, 100, 0.05)
    plt.plot([0, 1], [0, 1], "--", label="baseline", alpha=0.5)
    datas = [
        pred_proba_nonparametric,
        pred_proba_parametric,
        baseline_random_choice,
        baseline_majority_rule
    ]
    labels = ["Non parametric", "Parametric", "Random Choice", "Majority Rule"]
    for i, data in enumerate(datas):
        sensitivities = []
        specificities = []
        for t in thresholds:
            # Convert scores to binary predictions based on threshold
            y_pred = (data >= t).astype(int)

            # Calculate True Positives (TP), False Positives (FP), True␣
 ↪Negatives (TN), False Negatives (FN)
            tp = np.sum((y_test == 1) & (y_pred == 1))
            fp = np.sum((y_test == 0) & (y_pred == 1))
            tn = np.sum((y_test == 0) & (y_pred == 0))
            fn = np.sum((y_test == 1) & (y_pred == 0))

            # Calculate True Positive Rate (TPR) and False Positive Rate (FPR)
            tpr = tp / (tp + fn)
            fpr = fp / (fp + tn)

            # Append TPR and FPR to lists
            sensitivities.append(tpr)
            specificities.append(fpr)
        plt.plot(specificities, sensitivities, label=labels[i])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.show()

# Call the function with predictions from both models and the true labels
plot_roc(y_pred_nonparametric, y_pred_parametric, y_test)
```

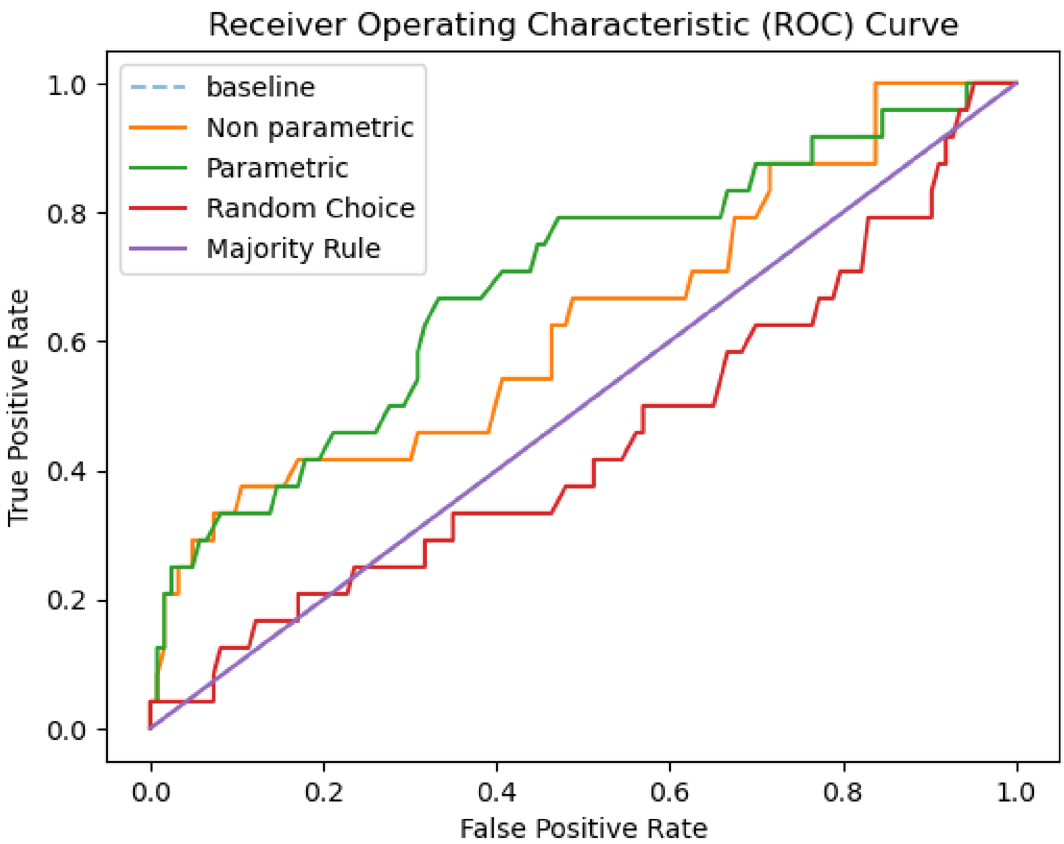


1.4.3 T18. Change the number of discretization bins to 5. What happens to the RoC curve? Which discretization is better? The number of discretization bins can be considered as a hyperparameter, and must be chosen by comparing the final performance.

[ ]:
```
# Data Preprocessing

df_train, df_test = train_test_split(df,
                                     test_size=0.1,
                                     stratify=df["Attrition"],
                                     random_state=7)

discretized_histograms(df_train, num_bins=5, show=False)
discretized_histograms(df_test, num_bins=5, show=False)

# selected_columns = ['MonthlyIncome', 'JobRole', 'HourlyRate', 'MaritalStatus']

x_train = df_train.drop(columns='Attrition').to_numpy() # all features
# x_train = df_train[selected_columns].to_numpy() # select features
x_train = apply_flooring_np(x_train)
```

```python
y_train = df_train["Attrition"].to_numpy()

x_test = df_test.drop(columns='Attrition').to_numpy() # all features
# x_test = df_test[selected_columns].to_numpy() # select features
x_test = apply_flooring_np(x_test)
y_test = df_test["Attrition"].to_numpy()
```

[ ]:
```python
# Modelling
n_pos = np.count_nonzero(y_train == 1)
n_neg = np.count_nonzero(y_train == 0)

model_T18 = SimpleBayesClassifier(n_pos=n_pos, n_neg=n_neg)

model_T18.fit_params(x_train, y_train)
y_pred_nonparametric = model_T18.predict(x_test)

model_T18.fit_gaussian_params(x_train, y_train)
y_pred_parametric = model_T18.gaussian_predict(x_test)

plot_roc(y_pred_nonparametric, y_pred_parametric, y_test)
```



1.4.4 OT4.

Shuffle the database, and create new test and train sets. Redo the entire training and evaluation process 10 times (each time with a new training and test set). Calculate the mean and variance of the accuracy rate.

```
[ ]: round = 10
accuracyLst = []

for i in range(round):

    # Data Preprocessing
    df_train, df_test = train_test_split(df,
                                         test_size=0.1,
                                         stratify=df["Attrition"],
                                         shuffle=True)

    discretized_histograms(df_train, num_bins=5, show=False)
    discretized_histograms(df_test, num_bins=5, show=False)

    x_train = df_train.drop(columns='Attrition').to_numpy() # all features
    x_train = apply_flooring_np(x_train)
    y_train = df_train["Attrition"].to_numpy()

    x_test = df_test.drop(columns='Attrition').to_numpy() # all features
    x_test = apply_flooring_np(x_test)
    y_test = df_test["Attrition"].to_numpy()

    # Modelling
    n_pos = np.count_nonzero(y_train == 1)
    n_neg = np.count_nonzero(y_train == 0)

    model_T18 = SimpleBayesClassifier(n_pos=n_pos, n_neg=n_neg)

    model_T18.fit_params(x_train, y_train)
    y_pred_nonparametric = model_T18.predict(x_test)
    accuracy, precision, recall, f1, false_positive_rate = evaluate(y_test,␣
↪y_pred_nonparametric, show_result=False)
    accuracyLst.append(accuracy)

    model_T18.fit_gaussian_params(x_train, y_train)
    y_pred_parametric = model_T18.gaussian_predict(x_test)
    accuracy, precision, recall, f1, false_positive_rate = evaluate(y_test,␣
↪y_pred_nonparametric, show_result=False)
    accuracyLst.append(accuracy)


print(f'Mean: {np.mean(accuracyLst)}')
```

```python
print(f'Variance: {np.var(accuracyLst)}')
```

```
Mean: 0.6074829931972788
Variance: 0.002767828219723263
```