Consider the following very simple model for stock pricing. The price at the end of each day is the price of the previous day multiplied by a fixed, but unknown, rate of return, α, with some noise, w. For a two-day period, we can observe the following sequence

$y_2 = \alpha y_1 + w_1$

$y_1 = \alpha y_0 + w_0$

where the noises w_0, w_1 are iid with the distribution $N(0, \sigma^2)$, $y_0 \sim N(0, \lambda)$ is independent of the noise sequence. σ^2 and λ are known, while α is unknown.

T1. Find the MLE of the rate of return, α, given the observed price at the end of each day y_2, y_1, y_0. In other words, compute for the value of α that maximizes $p(y_2, y_1, y_0|\alpha)$

$$p(y_2, y_1, y_0 \mid \alpha) = p(y_2 \mid y_1)\, p(y_1 \mid y_0)\, p(y_0 \mid \alpha)$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(y_2 - \alpha y_1)^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(y_1 - \alpha y_0)^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi\lambda}} e^{-\frac{y_0^2}{2\lambda}}$$

Take logarithm

$$\ln\left(p(y_2, y_1, y_0 \mid \alpha)\right) = \ln\left[\frac{1}{2\sqrt{2}\,\pi^{3/2}\sqrt{\lambda}\,\sigma^2} e^{-\frac{(y_2 - \alpha y_1)^2}{2\sigma^2} - \frac{(y_1 - \alpha y_0)^2}{2\sigma^2} - \frac{y_0^2}{2\lambda}}\right]$$

$$= \ln\left(\frac{1}{2\sqrt{2}\,\pi^{3/2}\sqrt{\lambda}\,\sigma^2}\right) - \frac{(y_2 - \alpha y_1)^2}{2\sigma^2} - \frac{(y_1 - \alpha y_0)^2}{2\sigma^2} - \frac{y_0^2}{2\lambda}$$

Find MLE with derivative

$$\frac{d}{d\alpha} \ln\left(p(y_2, y_1, y_0 \mid \alpha)\right) = 0$$

$$\frac{d}{d\alpha}\left(\underbrace{\ln\left(\frac{1}{2\sqrt{2}\,\pi^{3/2}\sqrt{\lambda}\,\sigma^2}\right)}_{0} - \frac{(y_2 - \alpha y_1)^2}{2\sigma^2} - \frac{(y_1 - \alpha y_0)^2}{2\sigma^2} - \frac{y_0^2}{2\lambda}\right) = 0$$

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

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

OT1. Consider the general case, where
$y_{n+1} = \alpha y_n + w_n, n = 0, 1, 2, \ldots$
Find the MLE given the observed price $y_{N+1}, y_N, \ldots, y_0$

$$p(y_{n+1}, y_n, y_{n-1} \cdots, y_0 \mid \alpha) = p(y_{n+1} \mid y_n)\, p(y_n \mid y_{n-1}) \cdots p(y_0 \mid \alpha)$$

$$\cdot \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(y_{n+1} - \alpha y_n)^2}{2\sigma^2}} \quad \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(y_n - \alpha y_{n-1})^2}{2\sigma^2}} \quad \cdots \quad \frac{1}{\sqrt{2\pi}\,\sigma} e^{\frac{-y_0^2}{2\sigma}}$$

Take Logarithm

$$\ln\left(p(y_{n+1}, y_n, y_{n-1}, \ldots, y_0 \mid \alpha)\right) = \ln\left[\frac{1}{(\sqrt{2\pi})^{n+2}\,\sigma^{n+1}\sqrt{\sigma}} e^{\frac{-(y_{n+1} - \alpha y_n)^2}{2\sigma^2} - \frac{(y_n - \alpha y_{n-1})^2}{2\sigma^2} - \cdots - \frac{y_0^2}{2\sigma}}\right]$$

$$= \ln\left(\frac{1}{(\sqrt{2\pi})^{n+2}\,\sigma^{n+1}\sqrt{\sigma}}\right) - \frac{(y_{n+1} - \alpha y_n)^2}{2\sigma^2} - \frac{(y_n - \alpha y_{n-1})^2}{2\sigma^2} - \cdots - \frac{y_0^2}{2\sigma}$$

Find MLE with derivative

$$\frac{d}{d\alpha} \ln\left(p(y_2, y_1, y_0 \mid \alpha)\right) = 0$$

$$\frac{d}{d\alpha}\left(\ln\left(\frac{1}{2\sqrt{2}\,\pi^{3/2}\sqrt{\sigma}\,\sigma^2}\right) - \frac{(y_{n+1} - \alpha y_n)^2}{2\sigma^2} - \frac{(y_n - \alpha y_{n-1})}{2\sigma^2} - \cdots - \frac{y_0^2}{2\sigma}\right) = 0$$

$$\frac{-(y_{n+1} - \alpha y_n)(-y_n)}{\sigma^2} + \frac{-(y_n - \alpha y_{n-1})(-y_{n-1})}{\sigma^2} + \cdots = 0$$

$$\frac{(y_{n+1} y_n - \alpha y_n^2) + (y_n y_{n-1} - \alpha y_{n-1}^2) + \cdots}{\sigma^2} = 0$$

$$(y_{n+1} y_n + y_n y_{n-1} + \cdots) - \alpha(y_n^2 + y_{n-1}^2 + \cdots) = 0$$

$$\alpha = \frac{y_{n+1} y_n + y_n y_{n-1} + \cdots}{y_n^2 + y_{n-1}^2 + \cdots}$$

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

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)

$P(x|w_1) = N(4, 2)$
$P(x|w_2) = N(0, 2)$

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.

From Bayes' Theorem (posterior, likelihood, prior, evidence)

$$P(a|b) = \frac{P(b|a)\,P(a)}{P(b)}$$

We can write $p(w_i|x)$ as

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

Assume

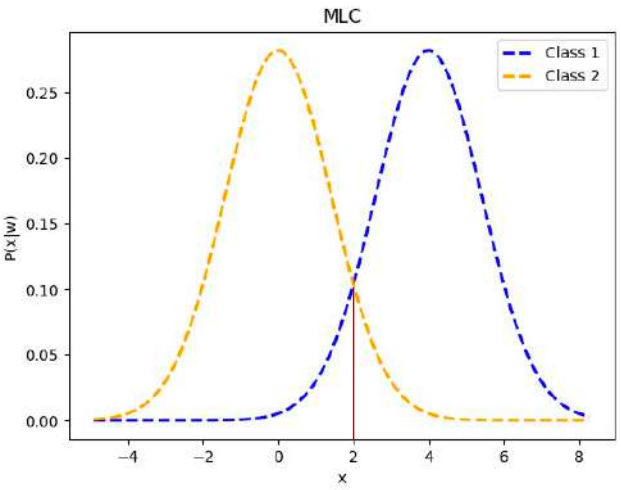
$$P(w_1|x) \;?\; P(w_2|x)$$

$$\frac{P(x|w_1)\,P(w_1)}{P(x)} \;?\; \frac{P(x|w_2)\,P(w_2)}{P(x)}$$

$$\frac{P(x|w_1)}{P(x|w_2)} \;?\; \frac{P(w_2)}{P(w_1)}$$ (ratio = 1)

※ Maximum likelihood criterion

```python
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Class 1
mean1, std_dev1 = 4, math.sqrt(2)
data1 = np.random.normal(mean1, std_dev1, 1000)

# Class 2
mean2, std_dev2 = 0, math.sqrt(2)
data2 = np.random.normal(mean2, std_dev2, 1000)

x_range = np.linspace(min(data1.min(), data2.min()), max(data1.max(), data2.max()), 100)
plt.plot(x_range, norm.pdf(x_range, mean1, std_dev1), color='blue', linestyle='dashed', linewidth=2)
plt.plot(x_range, norm.pdf(x_range, mean2, std_dev2), color='orange', linestyle='dashed', linewidth=2)

plt.title('MLC')
plt.xlabel('x')
plt.ylabel('P(x|w)')
plt.legend()
plt.show()
```



หา decision boundary

จาก MLC ได้ว่า boundary อยู่ที่

$$\frac{P(x|w_1)}{P(x|w_2)} = 1$$

$$\frac{\frac{1}{\sqrt{2\pi}\,\sigma_1} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}}{\frac{1}{\sqrt{2\pi}\,\sigma_2} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}} = 1$$

$$\frac{1}{\sqrt{2\pi}\sqrt{2}} e^{-\frac{(x-4)^2}{2(2)}} = \frac{1}{\sqrt{2\pi}\sqrt{2}} e^{-\frac{(x-0)^2}{2(2)}}$$

$$(x-4)^2 = (x-0)^2$$

$$x^2 - 8x + 16 = x^2$$

$$8x = 16$$

$$x = 2$$ ※ decision boundary

T3. What happen to the decision boundary if the cat is happy with a prior of 0.75?

w_1

$\Rightarrow P(w_1) = 0.75$

$P(w_2) = 0.25$

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

$$\frac{\frac{1}{\sqrt{2\pi}\sigma_1} e^{\frac{-(x-\mu_1)^2}{2\sigma_1^2}}}{\frac{1}{\sqrt{2\pi}\sigma_2} e^{\frac{-(x-\mu_2)^2}{2\sigma_2^2}}} = \frac{P(w_2)}{P(w_1)}$$

$$\frac{\frac{1}{\sqrt{2\pi}\sqrt{2}} e^{\frac{-(x-4)^2}{2(2)}}}{\frac{1}{\sqrt{2\pi}\sqrt{2}} e^{\frac{-(x-0)^2}{2(2)}}} = \frac{0.25}{0.75}$$

$$e^{\frac{-(x-4)^2}{2(2)} + \frac{x^2}{2(2)}} = \frac{1}{3}$$

$$\ln\left(e^{\frac{-(x-4)^2}{2(2)} + \frac{x^2}{2(2)}}\right) = \ln\left(\frac{1}{3}\right)$$

$$-\frac{(x-4)^2}{4} + \frac{x^2}{4} = -\ln(3)$$

$$-(x-4)^2 + x^2 = -4\ln(3)$$

$$-(x^2 - 8x + 16) + x^2 = -4\ln(3)$$

$$8x - 16 = -4\ln(3)$$

$$x = \frac{16 - 4\ln(3)}{8} = 2 - \frac{\ln(3)}{2} = 1.45$$

boundary เข้าใกล้ w_1 มากขึ้น

OT2. For the ordinary case of $P(x|w_1) = N(\mu_1, \sigma^2)$, $P(x|w_2) = N(\mu_2, \sigma^2)$, $p(w_1) = p(w_2) = 0.5$, prove that the decision boundary is at $x = \frac{\mu_1 + \mu_2}{2}$

ถ้า $p(w_1) = p(w_2) = 0.5$ แสดงว่า prior เท่ากัน

สามารถใช้ MLC ได้

$$\frac{P(x|w_1)}{P(x|w_2)} = 1$$

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

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

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

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

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

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

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

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

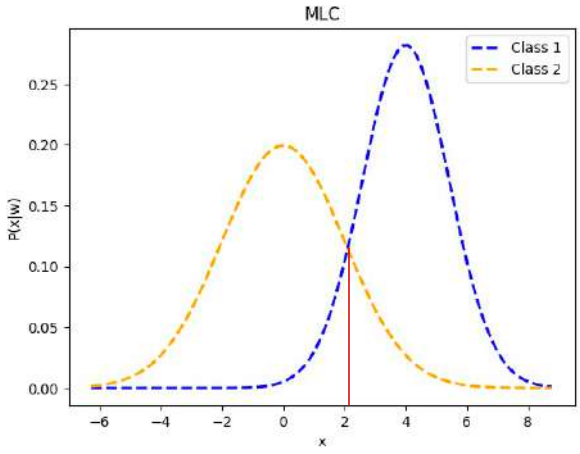
$$(\mu_1 - \mu_2)(\mu_1 + \mu_2) = 2x(\mu_1 - \mu_2)$$

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

\# $\mu_2 \neq \mu_2$

OT3. If the student changed his model to
$P(x|w_1) = N(4, 2)$
$P(x|w_2) = N(0, 4)$
Plot the posteriors values of the two classes on the same axis. What is the decision boundary for this classifier? Assume equal prior probabilities.

```python
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Class 1
mean1, std_dev1 = 4, math.sqrt(2)
data1 = np.random.normal(mean1, std_dev1, 1000)

# Class 2
mean2, std_dev2 = 0, math.sqrt(4)
data2 = np.random.normal(mean2, std_dev2, 1000)

x_range = np.linspace(min(data1.min(), data2.min()), max(data1.max(), data2.max()), 100)
plt.plot(x_range, norm.pdf(x_range, mean1, std_dev1), color='blue', linestyle='dashed', linewidth=2, label='Class 1')
plt.plot(x_range, norm.pdf(x_range, mean2, std_dev2), color='orange', linestyle='dashed', linewidth=2, label='Class 2')

plt.title('MLC')
plt.xlabel('x')
plt.ylabel('P(x|w)')
plt.legend()
plt.show()
```



จาก prior เท่ากัน ใช้ MLC

$$\frac{P(x|w_1)}{P(x|w_2)} = 1$$

$$\frac{\frac{1}{\sqrt{2\pi}\,\sigma_1} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}}{\frac{1}{\sqrt{2\pi}\,\sigma_2} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}} = 1$$

$$\frac{1}{\sqrt{2\pi}\,\sigma_1} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} = \frac{1}{\sqrt{2\pi}\,\sigma_2} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}$$

$$\frac{1}{\sqrt{2\pi}\,\sqrt{2}} e^{-\frac{(x-4)^2}{2(2)}} = \frac{1}{\sqrt{2\pi}\,\sqrt{4}} e^{-\frac{(x-0)^2}{2(4)}}$$

$$\frac{1}{\sqrt{2}} e^{-\frac{(x-4)^2}{4}} = \frac{1}{2} e^{-\frac{x^2}{8}}$$

$$e^{-\frac{(x-4)^2}{4} + \frac{x^2}{8}} = \frac{1}{\sqrt{2}}$$

$$\ln\left(e^{-\frac{(x-4)^2}{4} + \frac{x^2}{8}}\right) = \ln\left(\frac{1}{\sqrt{2}}\right)$$

$$-\frac{(x-4)^2}{4} + \frac{x^2}{8} = -\frac{1}{2}\ln(2)$$

$$-2(x-4)^2 + x^2 = -4\ln(2)$$

$$-2(x^2 - 8x + 16) + x^2 = -4\ln(2)$$

$$-2x^2 + 16x - 32 + x^2 + 4\ln(2) = 0$$

$$-x^2 + 16x - 32 + 4\ln(2) = 0$$

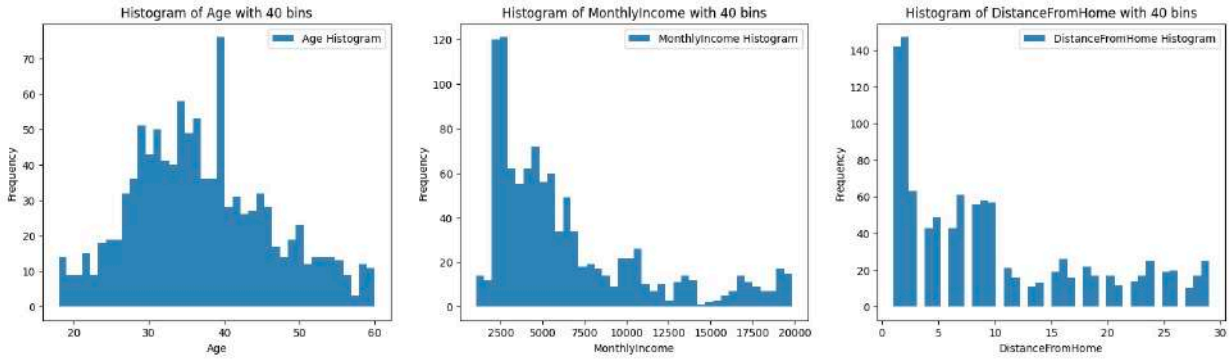
$$x = 8 \pm 2\sqrt{8 + \ln(2)}$$

$$X = 8 - 2\sqrt{8 + \ln(2)} \approx 2.10$$ ✱

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

```python
def display_histogram(df, col_name, n_bin = 40):
    # remove NaN values in specific column
    train_col_no_nan = df[col_name][~np.isnan(df[col_name])]

    # bin the data into 40 equally spaced bins
    # hist is the count for each bin
    # bin_edge is the edge values of the bins
    hist, bin_edge = np.histogram(train_col_no_nan, n_bin)

    # plot the histogram
    plt.fill_between(bin_edge.repeat(2)[1:-1],hist.repeat(2),facecolor='steelblue', label=f'{col_name} Histogram')

    plt.title(f'Histogram of {col_name} with {n_bin} bins')
    plt.xlabel(col_name)
    plt.ylabel('Frequency')
    plt.legend()
    # plt.show()

    # count zero bins
    zero_bins_count = np.sum(hist == 0)
    print(f"Number of bins with zero counts in {col_name}: {zero_bins_count}")
```

```python
plt.figure(figsize=(20, 5))
plt.subplot(1, 3, 1)
display_histogram(df_train, 'Age')
plt.subplot(1, 3, 2)
display_histogram(df_train, 'MonthlyIncome')
plt.subplot(1, 3, 3)
display_histogram(df_train, 'DistanceFromHome')
```




```
Number of bins with zero counts in Age: 0
Number of bins with zero counts in MonthlyIncome: 0
Number of bins with zero counts in DistanceFromHome: 11
```

For Age and MonthlyIncome, 40 bins is enough to discretize the data but 40 bins is not effectively capture the distribution of data and lead to 0 probability in PDF

So histogram with 40 bins is not a good discretization because it cause bin with zero count but it can fix by reducing the number is bin.

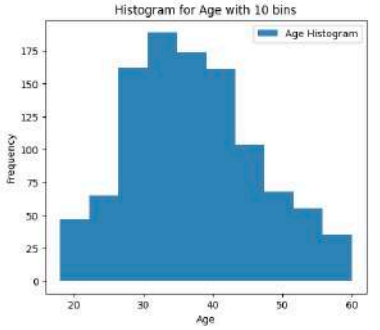
T5. Can we use a Gaussian to estimate this histogram? Why? What about a Gaussian Mixture Model (GMM)?

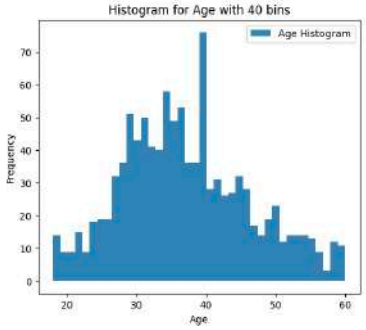
- Gaussian cannot estimate this histogram because the histogram has multi-modal
- GMM (Gaussian Mixture Model) can be estimate this histogram because the histogram look like many Gaussian combined together.

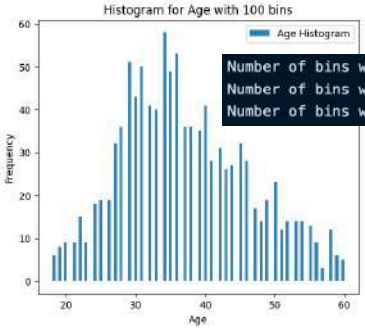
T6. Now plot the histogram according to the method described above (with 10, 40, and 100 bins) and show 3 plots each for Age, MonthlyIncome, and DistanceFromHome. Which bin size is most sensible for each features? Why?

```python
def display_histogram_inf(df, col_name, n_bin = 40):
    # remove NaN values in specific column
    train_col_no_nan = df[col_name][~np.isnan(df[col_name])]

    # create bin edges, including -inf and inf
    bin_edges = np.linspace(train_col_no_nan.min(), train_col_no_nan.max(), n_bin + 1)

    # add -inf as the first bin edge and inf as the last bin edge
    bin_edges_inf = bin_edges.copy()
    bin_edges_inf[0] = -np.inf
    bin_edges_inf[-1] = np.inf

    # create the histogram
    hist, _ = np.histogram(train_col_no_nan, bins=bin_edges_inf)

    # plot the histogram with label
    plt.fill_between(bin_edges.repeat(2)[1:-1], hist.repeat(2), facecolor='steelblue', label=f'{col_name} Histogram')
    plt.title(f'Histogram for {col_name} with {n_bin} bins')
    plt.xlabel(col_name)
    plt.ylabel('Frequency')
    plt.legend()
    # plt.show()

    # count zero bins
    zero_bins_count = np.sum(hist == 0)
    print(f"Number of bins with zero counts in {col_name} with {n_bin} bins: {zero_bins_count}")
```

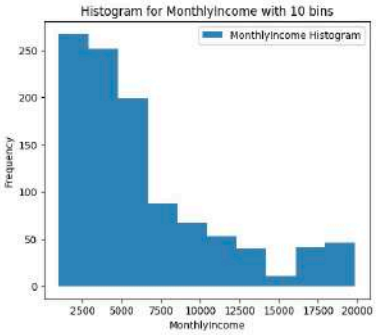
```python
nbins = [10, 40, 100]

for col in ['Age', 'MonthlyIncome', 'DistanceFromHome']:
  plt.figure(figsize=(20, 5))
  for i in range(3):
      plt.subplot(1, 3, i+1)
      display_histogram_inf(df_train, col, nbins[i])
  plt.show()
```

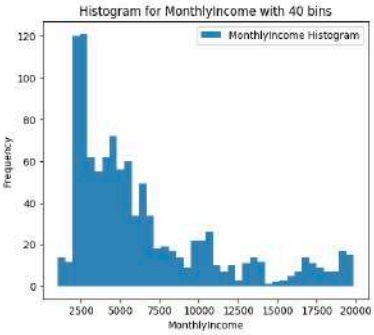


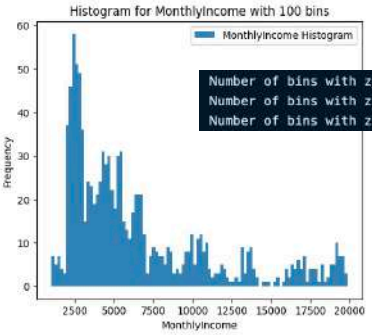




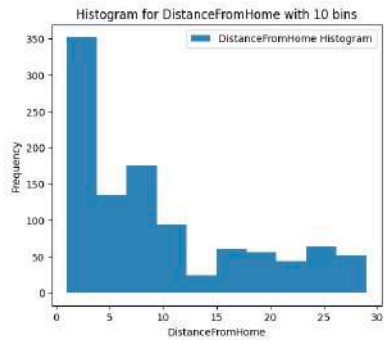

```
Number of bins with zero counts in Age with 10 bins: 0
Number of bins with zero counts in Age with 40 bins: 0
Number of bins with zero counts in Age with 100 bins: 57
```

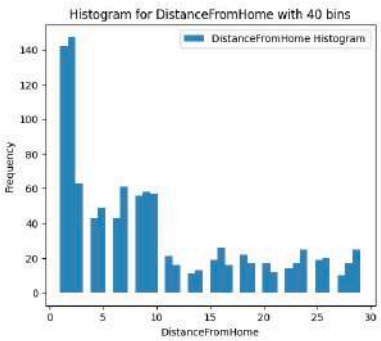


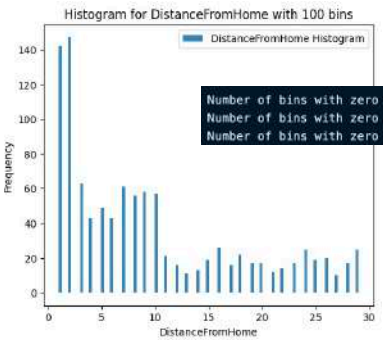





```
Number of bins with zero counts in MonthlyIncome with 10 bins: 0
Number of bins with zero counts in MonthlyIncome with 40 bins: 0
Number of bins with zero counts in MonthlyIncome with 100 bins: 3
```







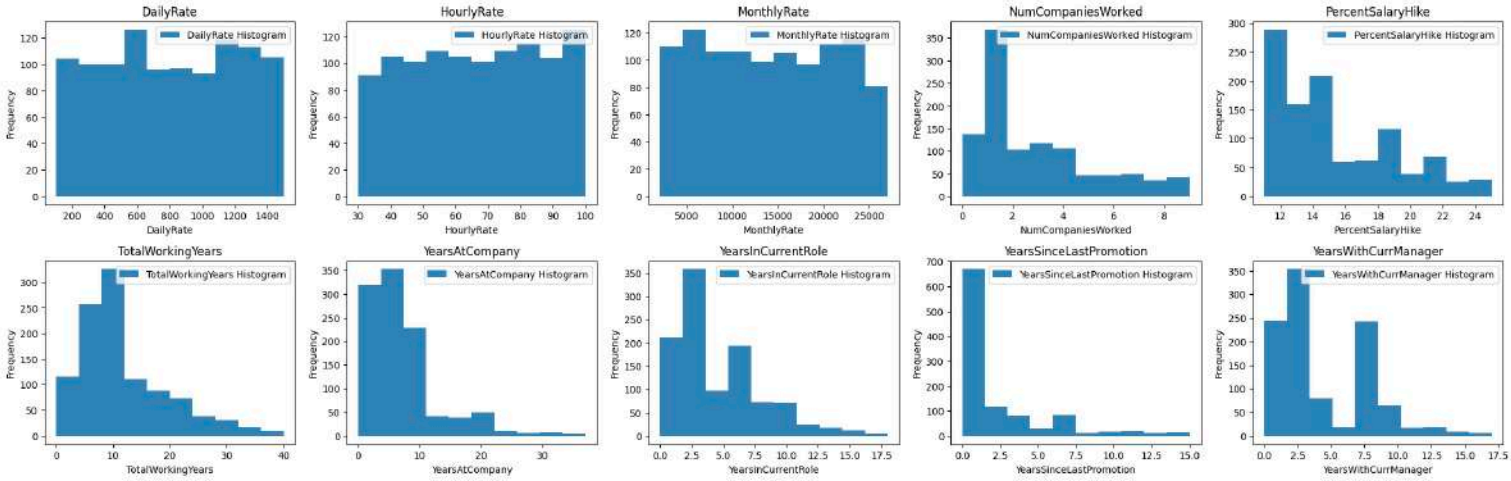

```
Number of bins with zero counts in DistanceFromHome with 10 bins: 0
Number of bins with zero counts in DistanceFromHome with 40 bins: 11
Number of bins with zero counts in DistanceFromHome with 100 bins: 71
```

Age : 40 bins
MonthlyIncome : 40 bins because it can discretize the data without having zero count in bins.
DistanceFromHome : 10 bins and optimal to provide enough granular view of the data.

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.

Features : DailyRate, HourlyRate, MonthlyRate, NumCompaniesWorked, PercentSalaryHike, TotalWorkingYears
YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager

Criterea :
1. Not a Categorical
2. Not that value (70% duplicate data in same column)
3. We are not chosen




```
idx = 1
plt.figure(figsize=(22, 10))
for col in ['DailyRate', 'HourlyRate', 'MonthlyRate', 'NumCompaniesWorked', 'PercentSalaryHike', 'TotalWo
    plt.subplot(3, 5, idx)
    display_histogram_inf(df_train, col, 10)
    plt.title(col)

    idx += 1

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

```
Number of bins with zero counts in DailyRate with 10 bins: 0
Number of bins with zero counts in HourlyRate with 10 bins: 0
Number of bins with zero counts in MonthlyRate with 10 bins: 0
Number of bins with zero counts in NumCompaniesWorked with 10 bins: 0
Number of bins with zero counts in PercentSalaryHike with 10 bins: 0
Number of bins with zero counts in TotalWorkingYears with 10 bins: 0
Number of bins with zero counts in YearsAtCompany with 10 bins: 0
Number of bins with zero counts in YearsInCurrentRole with 10 bins: 0
Number of bins with zero counts in YearsSinceLastPromotion with 10 bins: 0
Number of bins with zero counts in YearsWithCurrManager with 10 bins: 0
```

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

We can use Multinomial Distribution to model the histograms (finite choices)

Find MLE $P(X_1, X_2, \dots, X_k \mid p_1, p_2, \dots p_k) = \frac{n!}{\prod_{i=1}^{k} x_i!} \prod p_i^{x_i}$; $n = \sum_{i=1}^{k} x_i$ and $\sum_{i=1}^{k} p_i = 1$

$\log(L) = \sum_{i=1}^{k} x_i \log(p_i) + \log\left(\frac{n!}{\prod_{i=1}^{k} x_i!}\right)$

From Lagrange Multipliers

$\log(L) = \sum_{i=1}^{k} x_i \log(p_i) + \log\left(\frac{n!}{\prod_{i=1}^{k} x_i!}\right) + \lambda\left(\sum_{i=1}^{k} p_i - 1\right)$

Find MLE with derivative

$\frac{\partial}{\partial p_j} \log(L) = \frac{\partial}{\partial p_j}\left[\sum_{i=1}^{k} x_i \log(p_i) + \log\left(\frac{n!}{\prod_{i=1}^{k} x_i!}\right) + \lambda\left(\sum_{i=1}^{k} p_i - 1\right)\right]$

$\frac{\partial}{\partial p_j} \log(L) = \frac{x_j}{p_j} + \lambda$ for maximum $= 0$

$p_j = \frac{x_j}{-\lambda}$

From Lagrange Multiplier $\sum_{i=1}^{k} \frac{x_i}{-\lambda} = 1$

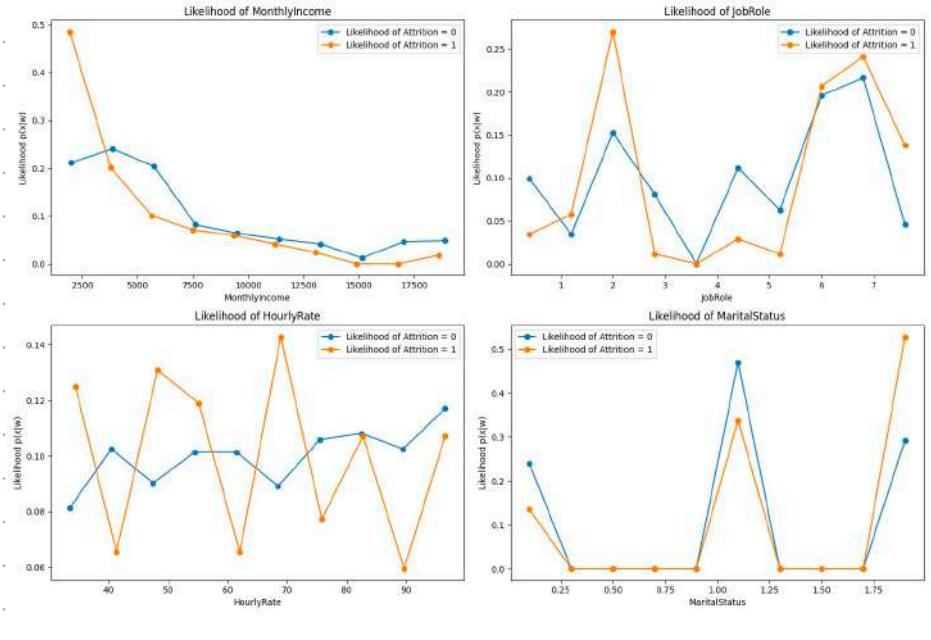
$-\lambda = \sum_{i=1}^{k} x_i$

$-\lambda = n$

$\therefore p_j = \frac{x_i}{n}$

```python
def display_likelihood(df, col_name, n_bin = 40):
    for attrition in [0, 1]:
        # filter DataFrame based on group
        group_df = df[df["Attrition"] == attrition]

        # remove NaN values in specific column
        train_col_no_nan = group_df[col_name][~np.isnan(group_df[col_name])]

        # create bin edges, including -inf and inf
        bin_edges = np.linspace(train_col_no_nan.min(), train_col_no_nan.max(), n_bin + 1)

        # add -inf as the first bin edge and inf as the last bin edge
        bin_edges_inf = bin_edges.copy()
        bin_edges_inf[0] = -np.inf
        bin_edges_inf[-1] = np.inf

        # create the histogram
        hist, _ = np.histogram(train_col_no_nan, bins=bin_edges_inf)

        # calculate the likelihood
        likelihood = hist / hist.sum()

        # calculate the bin centers
        bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

        # plot the likelihood
        plt.plot(bin_centers, likelihood, marker='o', label=f'Likelihood of Attrition = {attrition}')

    # set labels and title
    plt.xlabel(col_name)
    plt.ylabel('Likelihood p(x|w)')
    plt.title(f'Likelihood of {col_name}')

    # show legend
    plt.legend()
```

```python
idx = 1
plt.figure(figsize=(15, 10))

for col in ['MonthlyIncome', 'JobRole', 'HourlyRate', 'MaritalStatus']:
  plt.subplot(2, 2, idx)
  display_likelihood(df_train, col, 10)
  idx += 1

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



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

$P(\text{leave}) = \frac{\text{Number of leaving}}{\text{Total Data}}$

$P(\text{stay}) = \frac{\text{Number of staying}}{\text{Total Data}}$

```
pYes = len(df_train[df_train['Attrition'] == 1]) / len(df_train)
pNo = len(df_train[df_train['Attrition'] == 0]) / len(df_train)

print(f"P(Attrition = yes) = {pYes}")
print(f"P(Attrition = no) = {pNo}")
```

✓ 0.0s

```
P(Attrition = yes) = 0.16099773242630386
P(Attrition = no) = 0.8390022675736961
```

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.

- Use ε (10^{-9}) instead of 0 to make it non-zero without messing the data. (flooring)
- Smooth the values using counts from other observations (smoothing)
- Use Laplace smoothing

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.

```
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])]

    # INSERT CODE HERE
    for i in range(x.shape[1]):  # iterate over each feature
        # seperate data points
        stay_data = x[y == 0, i] # Attrition = 0
        leave_data = x[y == 1, i] # Attrition = 1

        # compute histogram for Attrition = 0
        stay_data_no_nan = stay_data[~np.isnan(stay_data)]
        stay_edges = np.linspace(stay_data_no_nan.min(), stay_data_no_nan.max(), n_bins + 1)
        stay_edges_inf = stay_edges.copy()
        stay_edges_inf[0] = -np.inf
        stay_edges_inf[-1] = np.inf
        stay_hist, _ = np.histogram(stay_data_no_nan, bins=stay_edges_inf)
        self.stay_params[i] = (stay_hist, stay_edges_inf)

        # compute histogram for Attrition = 1
        leave_data_no_nan = leave_data[~np.isnan(leave_data)]
        leave_edges = np.linspace(leave_data_no_nan.min(), leave_data_no_nan.max(), n_bins + 1)
        leave_edges_inf = leave_edges.copy()
        leave_edges_inf[0] = -np.inf
        leave_edges_inf[-1] = np.inf
        leave_hist, _ = np.histogram(leave_data_no_nan, bins=leave_edges_inf)
        self.leave_params[i] = (leave_hist, leave_edges_inf)

    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 = []

    # INSERT CODE HERE
    for sample in x:
        log_prob_leave = np.log(self.prior_pos) # P(leave) "Yes"
        log_prob_stay = np.log(self.prior_neg) # P(stay) "No"

        lH = log_prob_leave - log_prob_stay

        for i, feature in enumerate(sample):
            if np.isnan(feature):
                # skip in case of NaN
                continue

            # find the bin index for the feature value in the histogram bins
            leave_bin_idx = np.digitize(feature, bins=self.leave_params[i][1]) - 1 # we handle the -inf and inf
            stay_bin_idx = np.digitize(feature, bins=self.stay_params[i][1]) - 1 # we handle the -inf and inf

            # print(feature, leave_bin_idx, self.leave_params[i][1])
            # print(feature, stay_bin_idx, self.stay_params[i][1])

            # find log likelihoods of the feature value being in each bin
            log_likelihood_leave = np.log(self.leave_params[i][0][leave_bin_idx] / self.n_pos if self.leave_params[i][0][leave_bin_idx] != 0 else 1e-9)
            log_likelihood_stay = np.log(self.stay_params[i][0][stay_bin_idx] / self.n_neg if self.stay_params[i][0][leave_bin_idx] != 0 else 1e-9)

            lH += log_likelihood_leave - log_likelihood_stay

        # Compare log probabilities and assign the class label
        if lH > thresh:
            y_pred.append(1)  # 'Yes' 'leave'
        else:
            y_pred.append(0)  # 'No' 'stay'

    return np.array(y_pred).astype(np.float64)
```

```
def evaluate(y_true, y_pred, show_result = True):
    tp = np.sum((y_true == 1) & (y_pred == 1))
    tn = np.sum((y_true == 0) & (y_pred == 0))
    fp = np.sum((y_true == 0) & (y_pred == 1))
    fn = np.sum((y_true == 1) & (y_pred == 0))

    accuracy = (tp + tn) / (tp+tn+fp+fn + 1e-9)
    precision = tp / (tp + fp + 1e-9)
    recall = tp / (tp + fn + 1e-9)
    F1 = 1 / (1/(precision+1e-9) + 1/(recall+1e-9))
    fpr = fp / (fp + tn + 1e-9)

    if show_result:
        print(f"Accuracy: {accuracy:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall: {recall:.4f}")
        print(f"F1: {F1:.4f}")
        print(f"FPR: {fpr:.4f}")

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

```
Evaluate Histogram Naive Bayes
Accuracy: 0.8027
Precision: 0.3529
Recall: 0.2500
F1: 0.1463
FPR: 0.0894

(0.8027210884299135,
 0.352941176449827,
 0.24999999998958333,
 0.1463414639220702,
 0.089430894308216)
```

T12. Use the learned distributions to classify the `test_set`. Report the results using the same metric as the previous question.

```python
def fit_gaussian_params(self, x, y):

    """
    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])]

    # INSERT CODE HERE
    for i in range(x.shape[1]):
        stay = x[y == 0, i]
        stay = stay[~np.isnan(stay)]

        leave = x[y == 1, i]
        leave = leave[~np.isnan(leave)]

        self.gaussian_stay_params[i] = (np.mean(stay), np.std(stay))
        self.gaussian_leave_params[i] = (np.mean(leave), np.std(leave))

    return self.gaussian_stay_params, self.gaussian_leave_params
```

```
Evaluate Gaussian Naive Bayes
Accuracy: 0.7959
Precision: 0.3636
Recall: 0.3333
F1: 0.1739
FPR: 0.1138

(0.7959183673415244,
 0.3636363636198347,
 0.3333333333194444,
 0.17391304397164462,
 0.11382113821045674)
```

```python
def gaussian_predict(self, x, 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 = []

    # INSERT CODE HERE
    for i in range(x.shape[0]):
        log_prob_leave = np.log(self.prior_pos)
        log_prob_stay = np.log(self.prior_neg)

        lH = log_prob_leave - log_prob_stay

        for j in range(x.shape[1]):
            if np.isnan(x[i, j]):
                continue

            log_likelihood_leave = np.log(stats.norm.pdf(x[i, j], self.gaussian_leave_params[j][0], self.gaussian_leave_params[j][1]))
            log_likelihood_stay = np.log(stats.norm.pdf(x[i, j], self.gaussian_stay_params[j][0], self.gaussian_stay_params[j][1]))

            lH += log_likelihood_leave - log_likelihood_stay

        if lH > thresh:
            y_pred.append(1)
        else:
            y_pred.append(0)

    return np.array(y_pred).astype(np.float64)
```

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.

```
class BaselineModelA:

    def __init__(self):
        pass

    def predict(self, x):
        return np.random.choice([0, 1], size = x.shape[0], p = [0.5, 0.5])

baseline_model_a = BaselineModelA()
y_pred = baseline_model_a.predict(x = x_test)
evaluate(y_test, y_pred)
```

```
Accuracy: 0.5034
Precision: 0.1549
Recall: 0.4583
F1: 0.1158
FPR: 0.4878

(0.5034013605407932,
 0.1549295774626066,
 0.4583333333142361,
 0.11578947430415513,
 0.4878048780448146)
```

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.

```
class BaselineModelB:

    def __init__(self):
        pass

    def predict(self, x):
        # Major class is 0
        return np.array([0] * x.shape[0])

baseline_model_b = BaselineModelB()
y_pred = baseline_model_b.predict(x = x_test)
evaluate(y_test, y_pred)
```

```
Accuracy: 0.8367
Precision: 0.0000
Recall: 0.0000
F1: 0.0000
FPR: 0.0000

(0.8367346938718591, 0.0, 0.0, 5e-10, 0.0)
```

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

The accuracy of the first baseline model (50/50) is approximately 0.5

The accuracy of the second baseline model (majority vote rule) is 0.84 which is similar to major class prior

The accuracy of our Naive Bayes is around 0.80 (histogram) and 0.79 (gaussian)

The second baseline model has greater accuracy compare to Naive bayes but precision and recall are 0

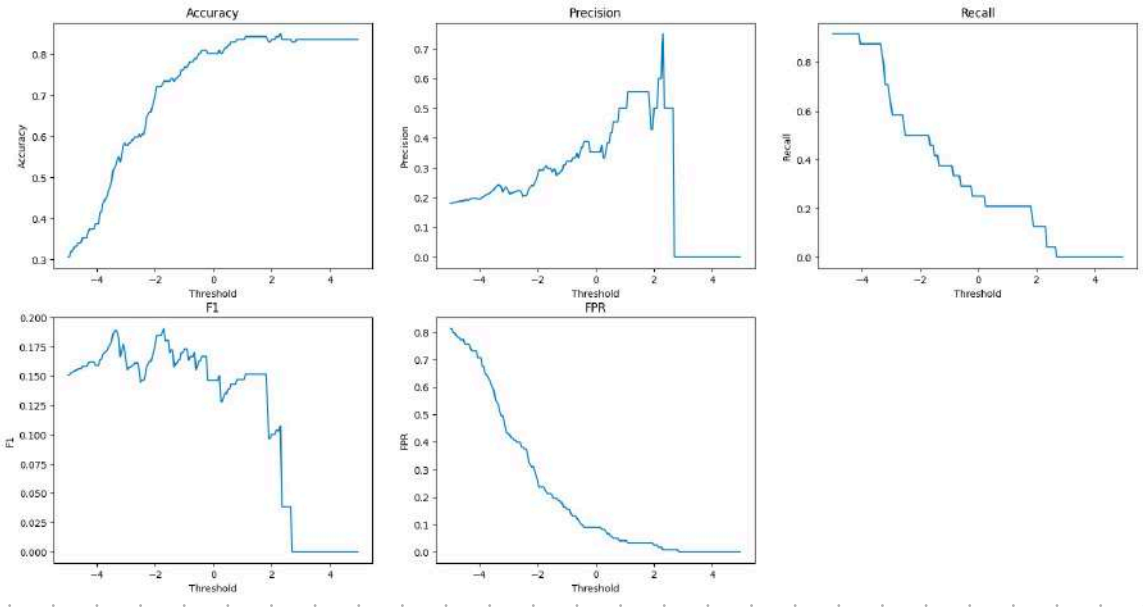
T16. Use the following threshold values

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

find the best accuracy, and F score (and the corresponding thresholds)

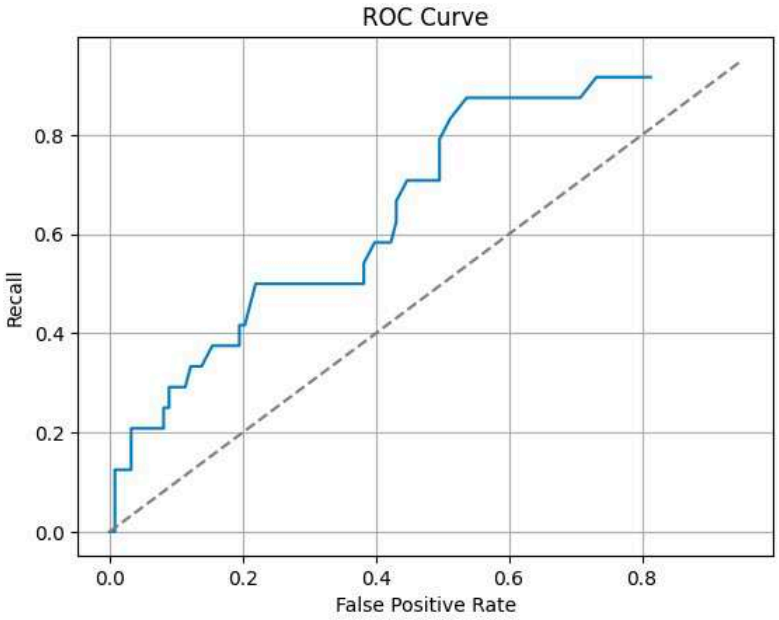
```python
best_acc = float('-inf')
best_acc_t = 0

best_f1 = float('-inf')
best_f1_t = 0

history = {
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1': [],
    'FPR': []
}

# Fit parameters again to ensure that the model is reset
model.fit_params(x_train, y_train)

for threshold in np.arange(-5, 5, 0.05):
  y_pred = model.predict(x_test, threshold)

  print(f"Threshold: {threshold:.2f}",end='\t| ')
  score = evaluate(y_test, y_pred, show_result=False)

  for idx, metric in enumerate(['Accuracy', 'Precision', 'Recall', 'F1', 'FPR']):
    print(f"{metric}: {score[idx]:.2f}", end='\t| ')
    history[metric].append(score[idx])

  print()

  if score[0] > best_acc:
    best_acc = score[0]
    best_acc_t = threshold

  if score[3] > best_f1:
    best_f1 = score[3]
    best_f1_t = threshold

print(f"Best threshold for accuracy: {best_acc_t:.2f} (Accuracy: {best_acc:.2f})")
print(f"Best threshold for F1: {best_f1_t:.2f} (F1: {best_f1:.2f})")
```

```
Best threshold for accuracy: 2.30 (Accuracy: 0.85)
Best threshold for F1: -1.70 (F1: 0.19)
```

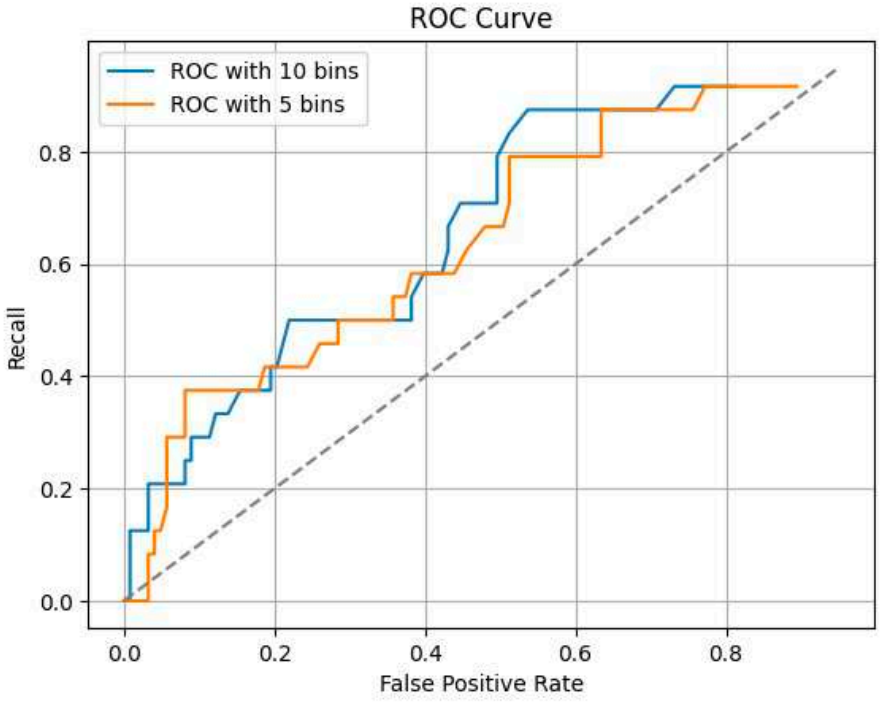


T17. Plot the RoC of your classifier.

```
plt.plot(history['FPR'], history['Recall'], label='ROC')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('Recall')

plt.plot(np.arange(0, 1, 0.05), np.arange(0, 1, 0.05), linestyle='dashed', color='gray', label='Random Guessing')
plt.grid()
plt.show()
```



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.



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.

```
accuracy_history = []

for shuffle in range (10):
  df_shuffled = df.sample(frac=1, random_state=shuffle).reset_index(drop=True)

  df_train, df_test = train_test_split(df_shuffled, test_size = 0.1, random_state = shuffle, stratify = df['Attrition'])

  # select_features = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
  # select_columns = ['Attrition'] + select_features

  data_train = df_train[select_columns].to_numpy().astype(np.float64)
  data_test = df_test[select_columns].to_numpy().astype(np.float64)

  x_train = data_train[:, 1:]
  y_train = data_train[:, 0]

  x_test = data_test[:, 1:]
  y_test = data_test[:, 0]

  model = SimpleBayesClassifier (n_pos = sum(y_train), n_neg = len(y_train) - sum(y_train))

  model.fit_params(x_train, y_train)

  y_pred = model.predict(x = x_test)

  score = evaluate(y_test, np.array(y_pred), show_result = False)

  accuracy_history.append(score[0])

meanAccuracy = sum(accuracy_history) / len(accuracy_history)
varAccuracy = sum([(x - meanAccuracy) ** 2) for x in accuracy_history]) / len(accuracy_history)
print(f'Average Accuracy: {meanAccuracy}')
print(f"Variance: {varAccuracy}")
```

```
Average Accuracy: 0.8122448979536582
Variance: 0.0004646212226323943
```