MA 324 - SIMA

Phonetic Analysis and Clustering of Vowel Sounds: Exploring Patterns in American English Vowels

Team Members:

Adapa Jashkayth - 210123005

Ankeshwar Suketh - 210123007

Koppula Hrushikesh -210123034

Lalhriemsang Fahriem - 210123036

Mude Srikanth - 210123040

TEAM NO:07

ABSTRACT

This project merges datasets from two pivotal studies in linguistics: the Peterson and Barney (1952) dataset and the Hillenbrand et al. (1995) dataset, focusing on American English vowel acoustics. While the Peterson and Barney dataset offers foundational insights, potential imperfections exist due to data preservation issues. Conversely, the Hillenbrand dataset provides cleaner data but may deviate slightly from the former's findings.

Combining both datasets, the project conducts comprehensive analyses. It begins with preprocessing steps, including handling missing data and mapping vowel characteristics. Classification tasks follow, utilizing machine learning algorithms to distinguish vowels based on acoustic measurements. Gender-specific analyses reveal subtle variations in vowel characteristics.

Additionally, the project employs visualization techniques such as Gaussian modeling and kernel density estimation to illustrate formant distributions. These insights shed light on underlying patterns and variations in American English vowel acoustics. By blending historical knowledge with modern computational methods, this project offers a nuanced understanding of vowel acoustics in American English.

HILLENBRANDETAL.1995DATASETBREAKDOWN

1. Index: This column is just a numerical index for referencing each row in the dataset.
2. ID: This column represents an identifier for each sample. It consists of multiple parts:
    - The first part indicates the sex and age group:
        - 'm' stands for male.
        - 'f' stands for female.
        - 'b' stands for boy.
        - 'g' stands for girl.
    - The next two digits are some kind of identity number specific to the individual within their sex and age group.
    - The last two characters represent the vowel being pronounced.
3. Duration: The duration of the vowel sound in milliseconds. F0-F4: These columns represent the formant frequencies F0 to F4 of the vowel sound. Formant frequencies are the resonant frequencies of the vocal tract.
4. F1_20, F2_20, F3_20: These columns represent the first three formant frequencies (F1, F2, F3) at 20% of the vowel duration.
5. F1_50, F2_50, F3_50: These columns represent the first three formant frequencies (F1, F2, F3) at 50% of the vowel duration.
6. F1_80, F2_80, F3_80: These columns represent the first three formant frequencies (F1, F2, F3) at 80% of the vowel duration

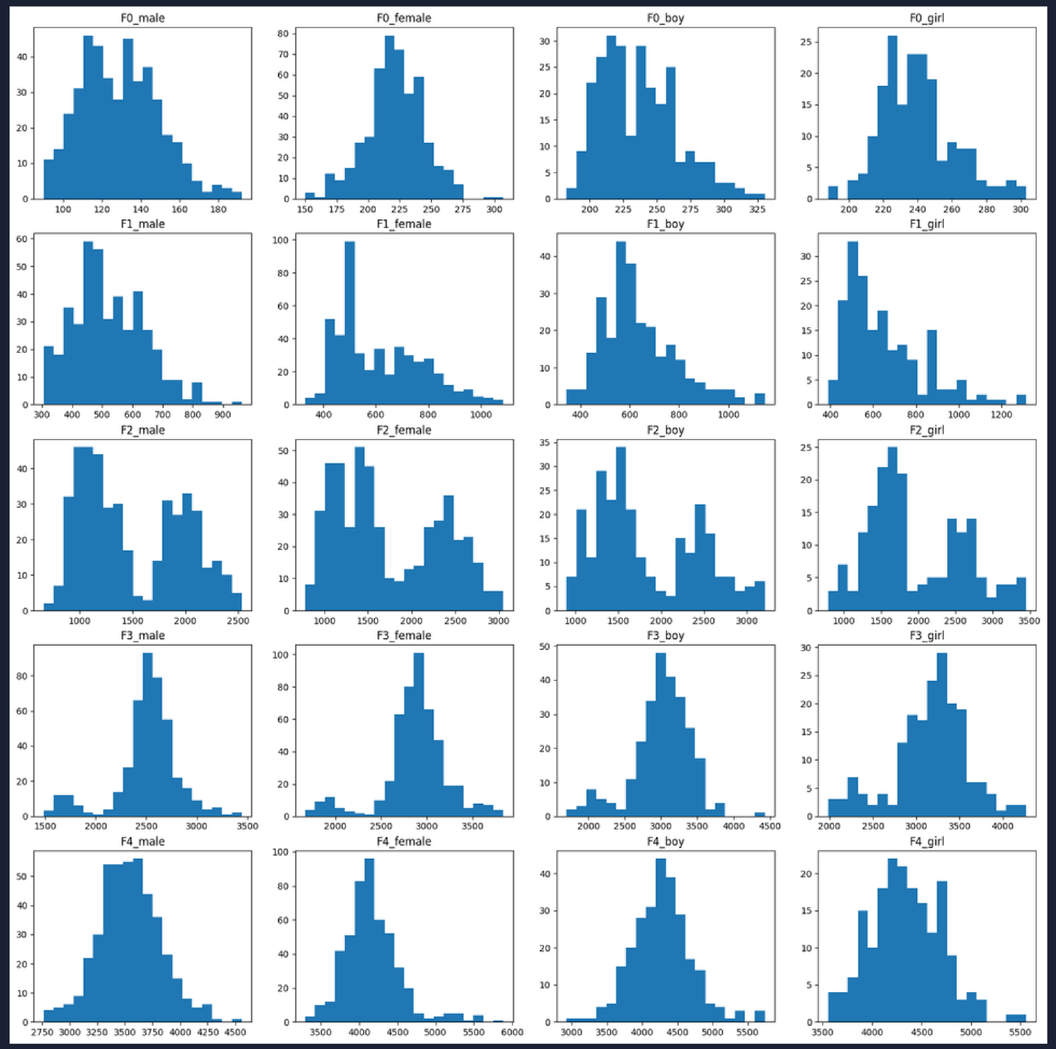
D A T A C L E A N I N G & M O D I F I C A T I O N

```
#Code for dropping columns with an
df = df.replace(0, np.nan)
df = df.dropna()
df = df.reset_index()
df = df.drop('Index', axis=1)
df = df.drop('index', axis=1)
df.head()
```

This part of code removes rows with missing values and resets the index for data consistency.

```
#Adding colum Loading... by Sex
for x in range(df.shape[0]):
    if df.loc[x, 'ID'].startswith('m'):
        df.loc[x, 'Sex'] = 'male'
    elif df.loc[x, 'ID'].startswith('w'):
        df.loc[x, 'Sex'] = 'female'
    elif df.loc[x, 'ID'].startswith('b'):
        df.loc[x, 'Sex'] = 'boy'
    elif df.loc[x, 'ID'].startswith('g'):
        df.loc[x, 'Sex'] = 'girl'
```

This code assigns the 'Sex' column in the DataFrame based on the prefix of the 'ID' column, categorizing individuals as male, female, boy, or girl.



Histogram Plots

- Histogram closely resembles that of normal and mixture of normal distributions.

EM-ALGORITHM:

1. Initialization:
    - The algorithm starts with initial guesses for the parameters of the normal distribution, which typically include the mean (μ) and standard deviation (σ).
2. Expectation (E-step):
    - In this step, for each data point, the algorithm computes the probability that the data point belongs to the normal distribution given the current parameter estimates. This is typically done using the probability density function (PDF) of the normal distribution.
    - The probabilities are referred to as "responsibilities" and represent the likelihood that each data point is generated by the current estimated distribution.
3. Maximization (M-step):
    - In this step, the algorithm updates the parameter estimates (μ and σ) based on the responsibilities computed in the E-step.
    - The new parameter estimates are computed to maximize the likelihood of the observed data under the current model. This often involves taking the weighted average of the data points, where the weights are the responsibilities computed in the E-step.
4. Repeat step 2 and 3 until convergence is achieved.

Estimating Mean and variance using EM-algorithm for single normal distibutions

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


def compute_responsibilities_single(data, mean, variance):
    return np.ones_like(data)

def update_parameters_single(data, responsibility):
    mean = np.sum(responsibility * data) / np.sum(responsibility)
    variance = np.sum(responsibility * (data - mean)**2) / np.sum(responsibility)
    return mean, variance


def em_algorithm_single(data, initial_mean, initial_variance, max_iter=100, tolerance=1e-6):
    mean = initial_mean
    variance = initial_variance

    for i in range(max_iter):
        # M-step
        responsibility = compute_responsibilities_single(data, mean, variance)
        new_mean, new_variance = update_parameters_single(data, responsibility)

        # Check convergence
        if np.abs(new_mean - mean) < tolerance and np.abs(new_variance - variance) < tolerance:
            break

        mean, variance = new_mean, new_variance

    return mean, variance
```

EM Algorithm Functions:

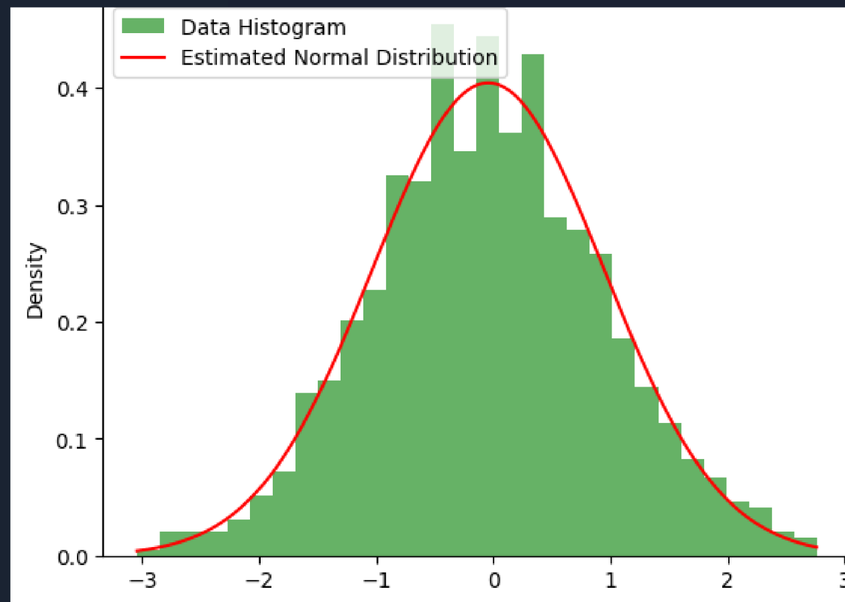
- compute_responsibilities_single(data, mean, variance): This function computes the responsibilities for each data point given the current estimates of the mean and variance. In this example, it returns a constant array of ones since it assumes a single normal distribution.
- update_parameters_single(data, responsibility): This function updates the parameters (mean and variance) based on the responsibilities computed in the E-step. It calculates the new mean and variance using weighted averages based on the responsibilities.
- em_algorithm_single(data, initial_mean, initial_variance, max_iter, tolerance): This function runs the EM algorithm. It initializes the mean and variance, then iterates between the E-step and M-step until convergence or until reaching the maximum number of iterations (max_iter). Convergence is checked based on the change in mean and variance compared to a predefined tolerance (tolerance).

Plotting:

- A histogram of the synthetic data is plotted to visualize its distribution.
- The estimated normal distribution obtained from the EM algorithm is plotted over the histogram for comparison.

Executing the EM-Algorithm

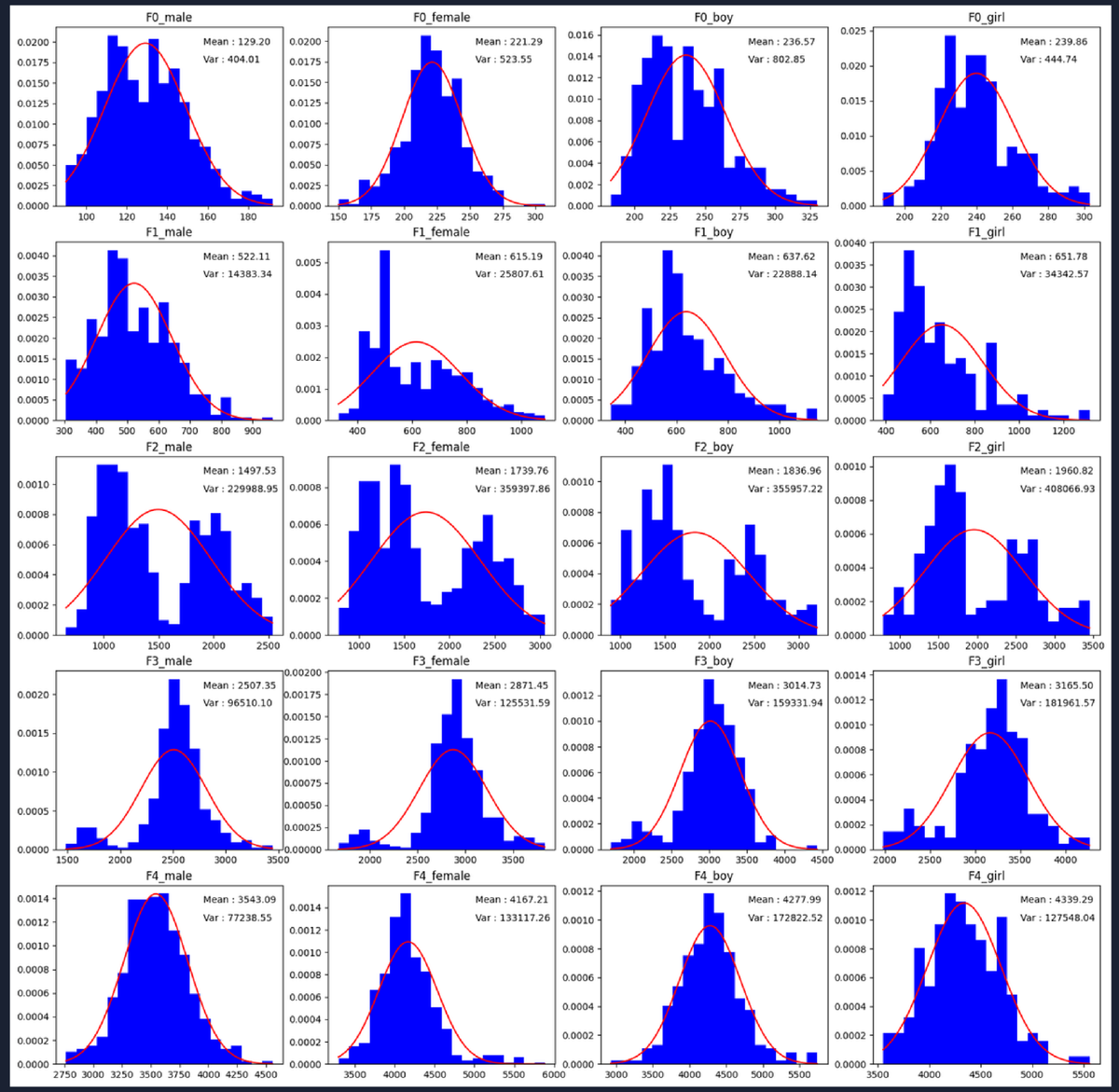
```
np.random.seed(0)
n_samples = 1000
mean_true = 0
variance_true = 1
data = np.random.normal(mean_true, np.sqrt(variance_true), n_samples)

# Initialize parameters
initial_mean = 0
initial_variance = 1

# Run EM algorithm for single component
mean_em, variance_em = em_algorithm_single(data, initial_mean, initial_variance)

# Plot histogram of data
plt.hist(data, bins=30, density=True, alpha=0.6, color='g', label='Data Histogram')

# Plot estimated normal distribution
x = np.linspace(min(data), max(data), 1000)
pdf = norm.pdf(x, mean_em, np.sqrt(variance_em))
plt.plot(x, pdf, 'r-', label='Estimated Normal Distribution')

plt.xlabel('Data')
plt.ylabel('Density')
plt.legend()
plt.show()

# Print estimates
print("EM Estimates (Single Component):")
print("mean:", mean_em)
print("variance:", variance_em)
```



Visualizing results for F0,F1,F2,F3,F4 and all males females boy and girl assuming single normal distribution

```
plots = 5
types = df['Sex'].nunique()
typelist = df['Sex'].unique()
fig,ax = plt.subplots(nrows=plots,ncols=types,figsize=[20,20])
for i in range(plots):
    for j,instance in enumerate(typelist):
        subset = df[df['Sex']==instance]
        data = subset['F'+str(i)].values
        initial_mean = 0
        initial_variance = 1
        mean_em, variance_em = em_algorithm_single(data, initial_mean, initial_variance)
        x = np.linspace(min(data), max(data), 1000)
        pdf = norm.pdf(x, mean_em, np.sqrt(variance_em))
        ax[i][j].hist(data, bins=20, density=True,  color='b', label='Data Histogram')
        ax[i][j].plot(x, pdf, 'r-', label='Estimated Normal Distribution')
        ax[i][j].set_title('F'+str(i)+'_'+instance)
        ax[i][j].text(0.65,0.9,f"Mean : {mean_em:.2f}",transform=ax[i][j].transAxes)
        ax[i][j].text(0.65,0.8,f"Var : {variance_em:.2f}",transform=ax[i][j].transAxes)

plt.show()
```

Plots:



Observations:

- F0, F1, F4 follow a single normal distributions but we can observe that F2, F3 don’t follow a single normal distributions.
- F2, F3 are comprised of a mixture of two normal distributions.
- This suggests that the data might be more accurately represented using a mixture model rather than a single normal distribution.

EM Algorithm for mixture of two normal distributions

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

# E-step: Compute responsibilities
def compute_responsibilities(data, p, mean1, variance1, mean2, variance2):
    likelihood1 = p * norm.pdf(data, mean1, np.sqrt(variance1)+1e-10)
    likelihood2 = (1 - p) * norm.pdf(data, mean2, np.sqrt(variance2)+1e-10)
    total_likelihood = likelihood1 + likelihood2
    total_likelihood[total_likelihood == 0] = np.finfo(float).eps  # Avoid division by zero
    responsibility1 = likelihood1 / total_likelihood
    responsibility2 = likelihood2 / total_likelihood
    return responsibility1, responsibility2

# M-step: Update parameters
# M-step: Update parameters
def update_parameters(data, responsibility1, responsibility2):
    p = np.mean(responsibility1)

    # Handle the case where responsibility is very small or zero
    sum_responsibility1 = np.sum(responsibility1)
    mean1 = np.sum(responsibility1 * data) / (sum_responsibility1 + 1e-10)
    variance1 = np.sum(responsibi (parameter) responsibility2: Any .responsibility1 + 1e-10)

    sum_responsibility2 = np.sum(responsibility2)
    mean2 = np.sum(responsibility2 * data) / (sum_responsibility2 + 1e-10)
    variance2 = np.sum(responsibility2 * (data - mean2)**2) / (sum_responsibility2 + 1e-10)

    return p, mean1, variance1, mean2, variance2
```

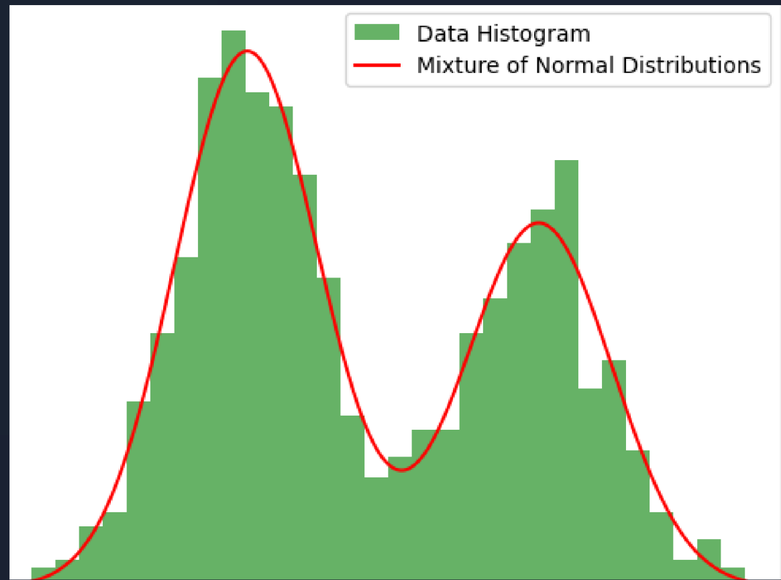
```python
# EM algorithm for mixture of normal distributions
def em_algorithm(data, initial_p, initial_mean1, initial_variance1, initial_mean2, initial_variance2,
max_iter=100, tolerance=1e-6):
    p = initial_p
    mean1 = initial_mean1
    variance1 = initial_variance1
    mean2 = initial_mean2
    variance2 = initial_variance2


    for i in range(max_iter):
        # E-step
        responsibility1, responsibility2 = compute_responsibilities(data, p, mean1, variance1, mean2,
        variance2)
        # M-step
        new_p, new_mean1, new_variance1, new_mean2, new_variance2 = update_parameters(data,
        responsibility1, responsibility2)

        # Check convergence
        if np.abs(new_p - p) < tolerance and np.abs(new_mean1 - mean1) < tolerance and np.abs
        (new_variance1 - variance1) < tolerance \
            and np.abs(new_mean2 - mean2) < tolerance and np.abs(new_variance2 - variance2) <
            tolerance:
            break

        p, mean1, variance1, mean2, variance2 = new_p, new_mean1, new_variance1, new_mean2,
        new_variance2

    return p, mean1, variance1, mean2, variance2
```

EM Algorithm for two normal distibutions

1. Compute Responsibilities Function (compute_responsibilities):
   - This function computes the responsibilities of each data point for the two normal distributions in the mixture model.
   - It calculates the likelihood of each data point belonging to each distribution based on the current parameter estimates (p, mean1, variance1, mean2, variance2) and normalizes them to sum to 1.
2. Update Parameters Function (update_parameters):
   - This function updates the parameters of the mixture model based on the responsibilities computed in the E-step.
   - It calculates the new values for the mixing coefficient (p), means (mean1, mean2), and variances (variance1, variance2) using weighted averages based on the responsibilities.
3. EM Algorithm Function (em_algorithm):
   - This function implements the EM algorithm for estimating the parameters of the mixture model.
   - It iteratively alternates between the E-step (computing responsibilities) and the M-step (updating parameters) until convergence or reaching a maximum number of iterations (max_iter).
   - Convergence is determined by checking if the changes in the parameters are below a predefined tolerance (tolerance).

Executing two normal distibutions EM Algorithm

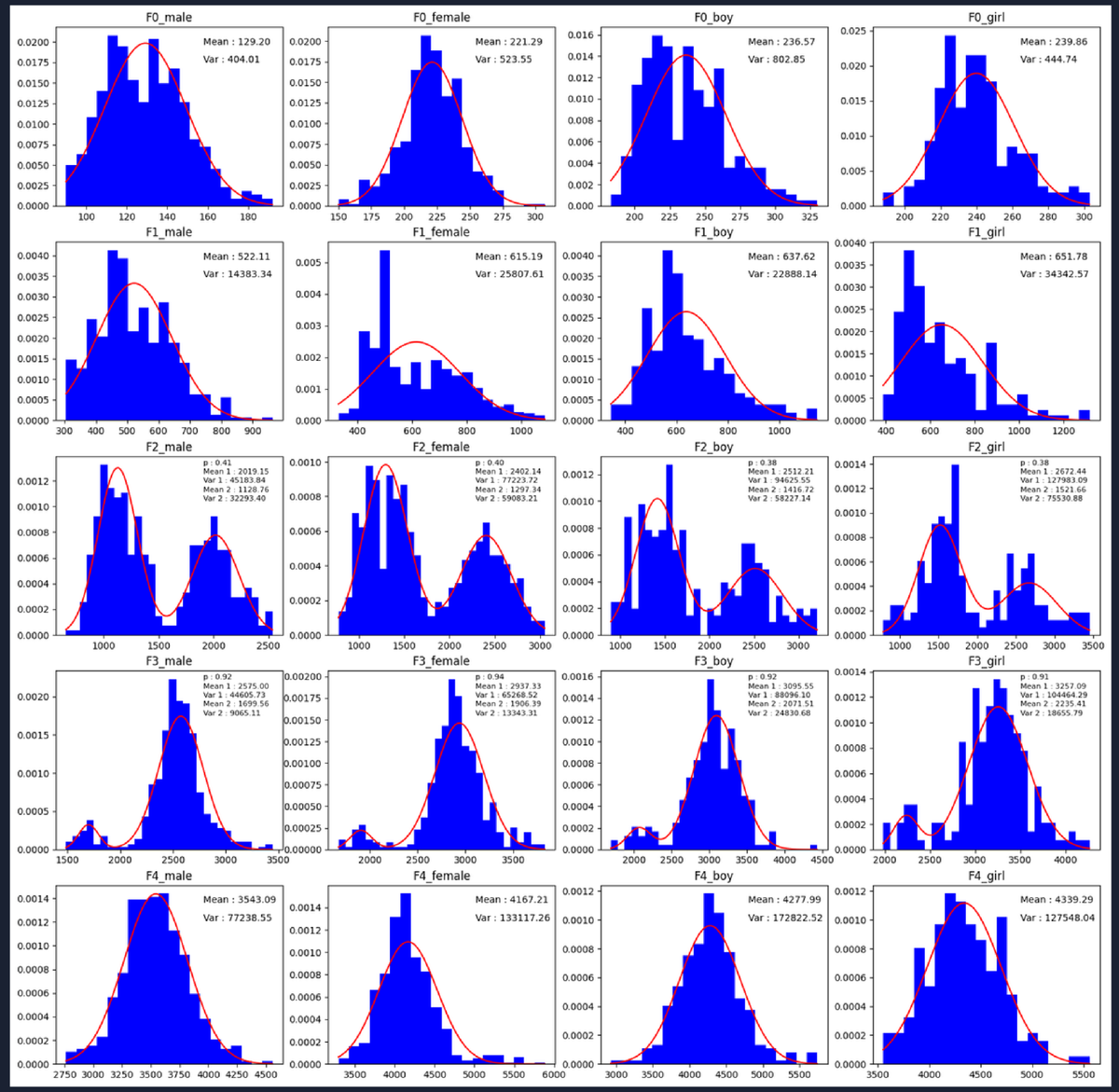
```
# Generate synthetic data following a mixture of two normal distributions
np.random.seed(0)
n_samples = 1000
p_true = 0.6
mean1_true = 0
variance1_true = 1
mean2_true = 4
variance2_true = 1
data = np.concatenate([np.random.normal(mean1_true, np.sqrt(variance1_true), int(n_samples * p_true)),
                       np.random.normal(mean2_true, np.sqrt(variance2_true), int(n_samples * (1 -
                       p_true)))])

# Initialize parameters
initial_p = 0.5
initial_mean1 = 0
initial_variance1 = 1
initial_mean2 = 1
initial_variance2 = 1

# Run EM algorithm for mixture of normal distributions
p_em, mean1_em, variance1_em, mean2_em, variance2_em = em_algorithm(data, initial_p, initial_mean1,
initial_variance1, initial_mean2, initial_variance2)
print(data.shape)
# Plot histogram of data
plt.hist(data, bins=30, density=True, alpha=0.6, color='g', label='Data Histogram')

# Plot estimated normal distributions
x = np.linspace(min(data), max(data), 1000)
pdf1 = norm.pdf(x, mean1_em, np.sqrt(variance1_em))
pdf2 = norm.pdf(x, mean2_em, np.sqrt(variance2_em))
mixture_pdf = p_em * pdf1 + (1 - p_em) * pdf2
plt.plot(x, mixture_pdf, 'r-', label='Mixture of Normal Distributions')

plt.xlabel('Data')
plt.ylabel('Density')
```



```
plots = 5
types = df['Sex'].nunique()
typelist = df['Sex'].unique()
fig,ax = plt.subplots(nrows=plots,ncols=types,figsize=[20,20])
for i in range(plots):
    for j,instance in enumerate(typelist):
        if i==0 or i == 1 or i==4:
            subset = df[df['Sex']==instance]
            data = subset['F'+str(i)].values
            initial_mean = 0
            initial_variance = 1
            mean_em, variance_em = em_algorithm_single(data, initial_mean, initial_variance)
            x = np.linspace(min(data), max(data), 1000)
            pdf = norm.pdf(x, mean_em, np.sqrt(variance_em))
            ax[i][j].hist(data, bins=20, density=True,  color='b', label='Data Histogram')
            ax[i][j].plot(x, pdf, 'r-', label='Estimated Normal Distribution')
            ax[i][j].set_title('F'+str(i)+'_'+instance)
            ax[i][j].text(0.65,0.9,f"Mean : {mean_em:.2f}",transform=ax[i][j].transAxes)
            ax[i][j].text(0.65,0.8,f"Var : {variance_em:.2f}",transform=ax[i][j].transAxes)
        elif i==2 or i==3:
            subset = df[df['Sex']==instance]
            data = subset['F'+str(i)].values
             (variable) initial_variance1: floating[Any]
            initial_variance1 = np.var(data)
            initial_mean2 = np.mean(data)/2
            initial_variance2 = np.var(data)/2
            p_em, mean1_em, variance1_em, mean2_em, variance2_em = em_algorithm(data, initial_p,
            initial_mean1, initial_variance1,initial_mean2, initial_variance2)
            x = np.linspace(min(data), max(data), 1000)
            pdf1 = norm.pdf(x, mean1_em, np.sqrt(variance1_em))
            pdf2 = norm.pdf(x, mean2_em, np.sqrt(variance2_em))
            mixture_pdf = p_em * pdf1 + (1 - p_em) * pdf2
            ax[i][j].hist(data, bins=30, density=True, alpha=1, color='b', label='Data Histogram')
            ax[i][j].plot(x, mixture_pdf, 'r-', label='Mixture of Normal Distributions')
            ax[i][j].set_title('F'+str(i)+'_'+instance)
            ax[i][j].text(0.65,0.95,f"p : {p_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.9,f"Mean 1 : {mean1_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.85,f"Var 1 : {variance1_em:.2f}",transform=ax[i][j].transAxes,
            fontsize=8)
            ax[i][j].text(0.65,0.8,f"Mean 2 : {mean2_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.75,f"Var 2 : {variance2_em:.2f}",transform=ax[i][j].transAxes,
            fontsize=8)
plt.show()
```

- **Visualizing data for F0, F1, F4 as single normal distributions and F2, F3 as mixture of two normal distributions**

Plots:



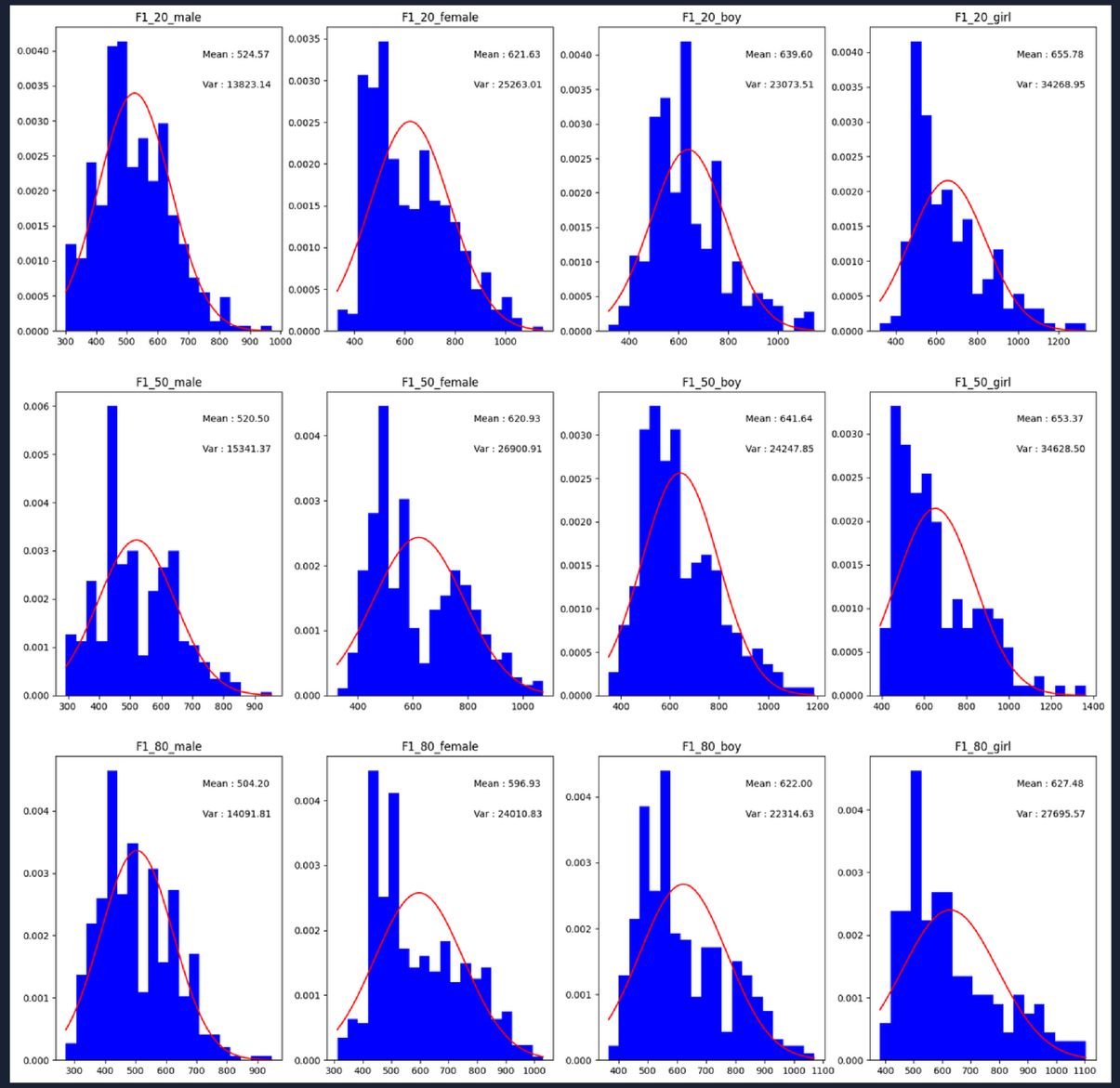
Observations:

- Upon visualizing the results using a mixture of two normal distributions, it's evident that they align more closely with the observed data.
- The likelihood of the mixture model capturing the underlying distribution appears to be higher.

- **Similarly visualizing F1_20,F1_50,F1_80 assuming single normal distibutions initially**

```
plots = 3
types = df["Sex"].nunique()
typelist = df["Sex"].unique()
fig, ax = plt.subplots(nrows=plots, ncols=types, figsize=[20, 20])
for i in range(plots):
    for j, instance in enumerate(typelist):
        subset = df[df["Sex"] == instance]
        data = subset["F1_" + str(20+i*30)].values
        initial_mean = 0
        initial_variance = 1
        mean_em, variance_em = em_algorithm_single(data, initial_mean, initial_variance)
        x = np.linspace(min(data), max(data), 1000)
        pdf = norm.pdf(x, mean_em, np.sqrt(variance_em))
        ax[i][j].hist(data, bins=20, density=True, color="b", label="Data Histogram")
        ax[i][j].plot(x, pdf, "r-", label="Estimated Normal Distribution")
        ax[i][j].set_title("F1_" + str(20+i*30) + "_" + instance)
        ax[i][j].text(0.65, 0.9, f"Mean : {mean_em:.2f}", transform=ax[i][j].transAxes)
        ax[i][j].text(
            0.65, 0.8, f"Var : {variance_em:.2f}", transform=ax[i][j].transAxes
        )

plt.show()
```

Plots:



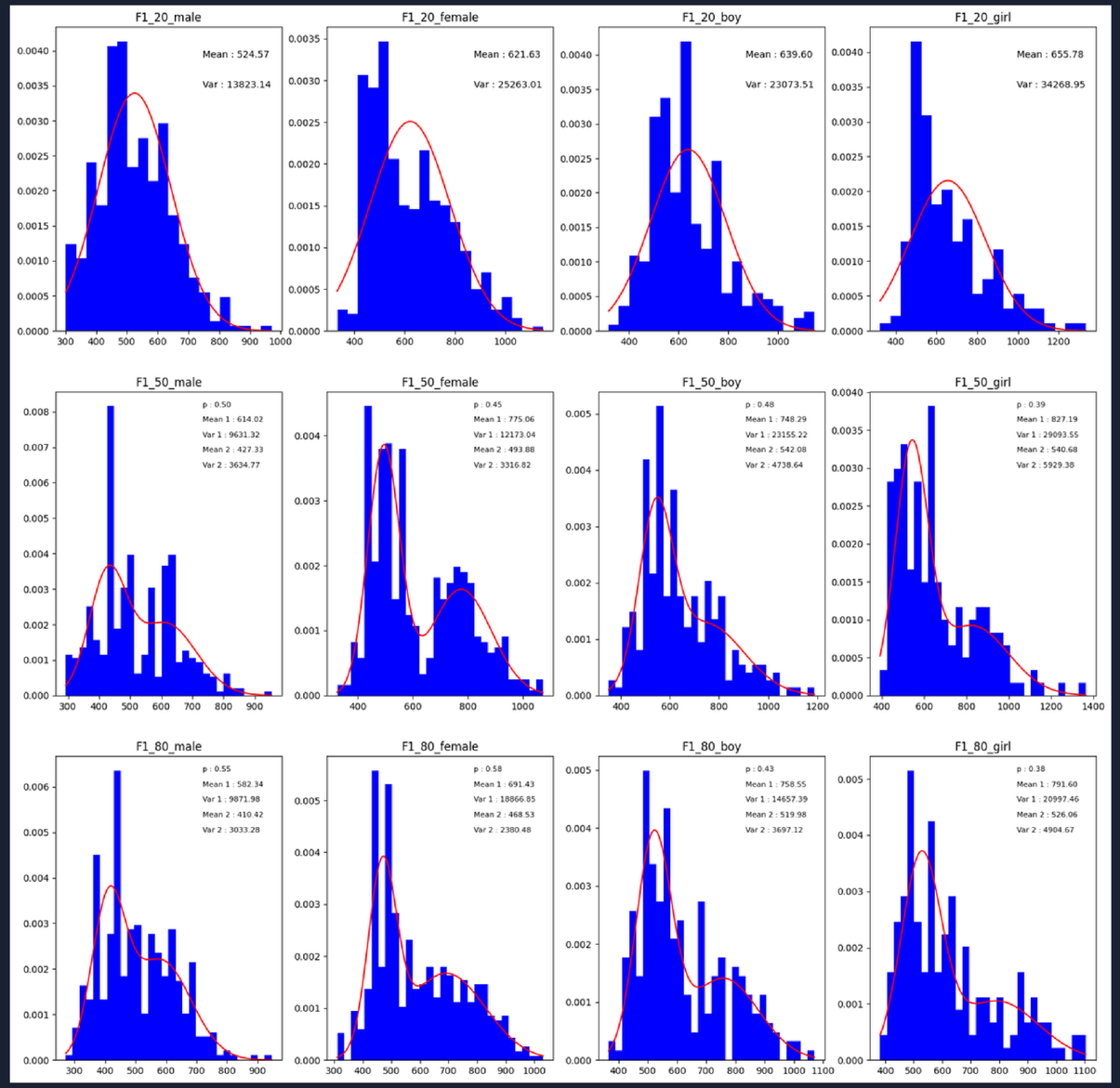
Observations:

- F1_20 follow a single normal distributions but we can observe that F1_50, F1_80 don’t follow a single normal distributions.
- F1_50, F1_80 are comprised of a mixture of two normal distributions.
- This suggests that the data might be more accurately represented using a mixture model rather than a single normal distribution.

```
plots = 3
types = df['Sex'].nunique()
typelist = df['Sex'].unique()
fig,ax = plt.subplots(nrows=plots,ncols=types,figsize=[20,20])
for i in range(plots):
    for j,instance in enumerate(typelist):
        if i==0:
            subset = df[df['Sex']==instance]
            data = subset['F1_'+str(20+i*30)].values
            initial_mean = 0
            initial_variance = 1
            mean_em, variance_em = em_algorithm_single(data, initial_mean, initial_variance)
            x = np.linspace(min(data), max(data), 1000)
            pdf = norm.pdf(x, mean_em, np.sqrt(variance_em))
            ax[i][j].hist(data, bins=20, density=True,  color='b', label='Data Histogram')
            ax[i][j].plot(x, pdf, 'r-', label='Estimated Normal Distribution')
            ax[i][j].set_title('F1_'+str(20+i*30)+'_'+instance)
            ax[i][j].text(0.65,0.9,f"Mean : {mean_em:.2f}",transform=ax[i][j].transAxes)
            ax[i][j].text(0.65,0.8,f"Var : {variance_em:.2f}",transform=ax[i][j].transAxes)
        elif i==1 or i==2:
            subset = df[df['Sex']==instance]
            data = subset['F1_'+str(20+i*30)].values
            initial_p = 0.3
            initial_mean1 = np.mean(data)
            initial_variance1 = np.var(data)
            initial_mean2 = np.mean(data)/2
            initial_variance2 = np.var(data)/2
            p_em, mean1_em, variance1_em, mean2_em, variance2_em = em_algorithm(data, initial_p,
            initial_mean1, initial_variance1,initial_mean2, initial_variance2)
            x = np.linspace(min(data), max(data), 1000)
            pdf1 = norm.pdf(x, mean1_em, np.sqrt(variance1_em))
            pdf2 = norm.pdf(x, mean2_em, np.sqrt(variance2_em))
            mixture_pdf = p_em * (function) bins: Any df2
            ax[i][j].hist(data, bins=30, density=True, alpha=1, color='b', label='Data Histogram')
            ax[i][j].plot(x, mixture_pdf, 'r-', label='Mixture of Normal Distributions')
            ax[i][j].set_title('F1_'+str(20+i*30)+'_'+instance)
            ax[i][j].text(0.65,0.95,f"p : {p_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.9,f"Mean 1 : {mean1_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.85,f"Var 1 : {variance1_em:.2f}",transform=ax[i][j].transAxes,
            fontsize=8)
            ax[i][j].text(0.65,0.8,f"Mean 2 : {mean2_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.75,f"Var 2 : {variance2_em:.2f}",transform=ax[i][j].transAxes,
            fontsize=8)
plt.show()
```

- **Visualizing data for F1_20 as single normal distribution and F1_50, F1_80 as mixture of two normal distributions.**

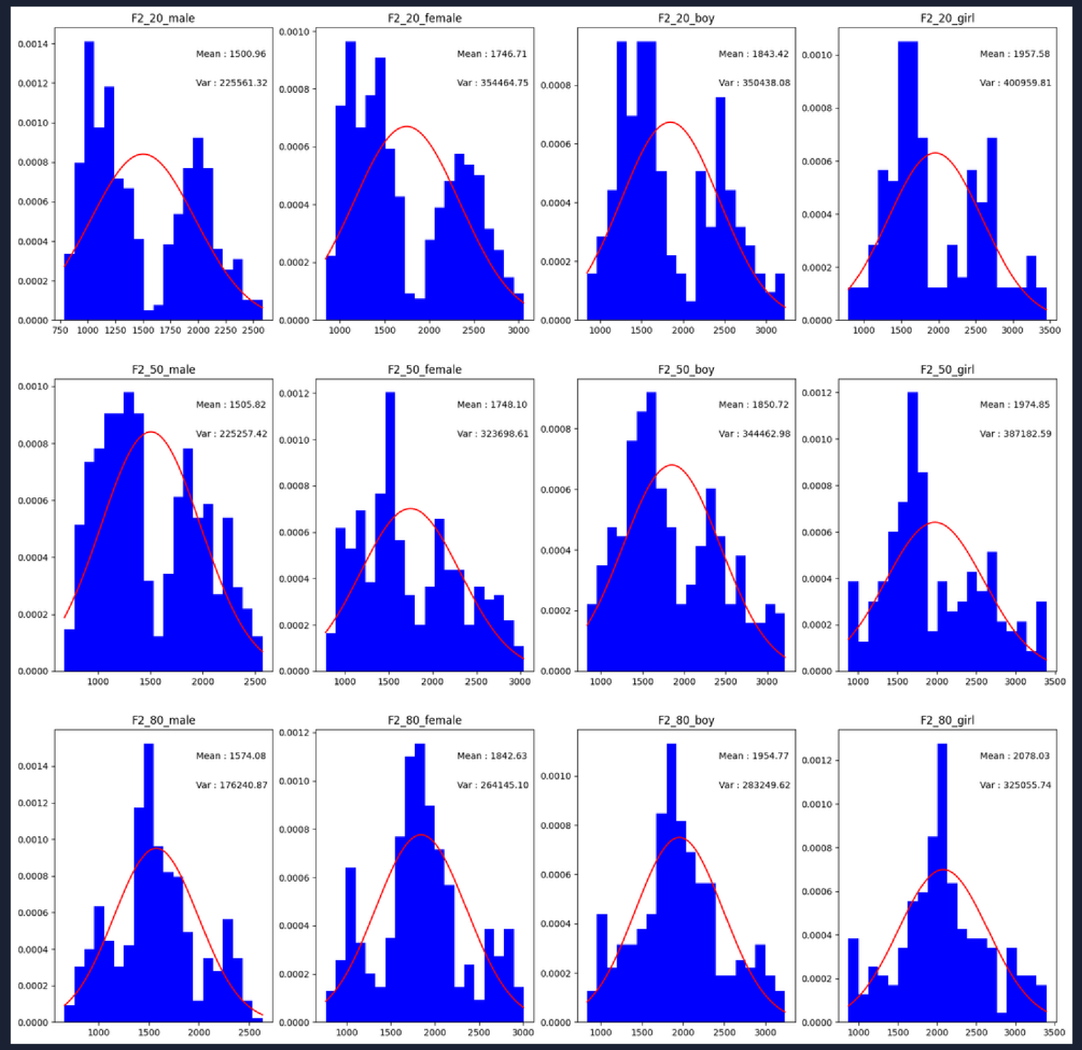
Plots:

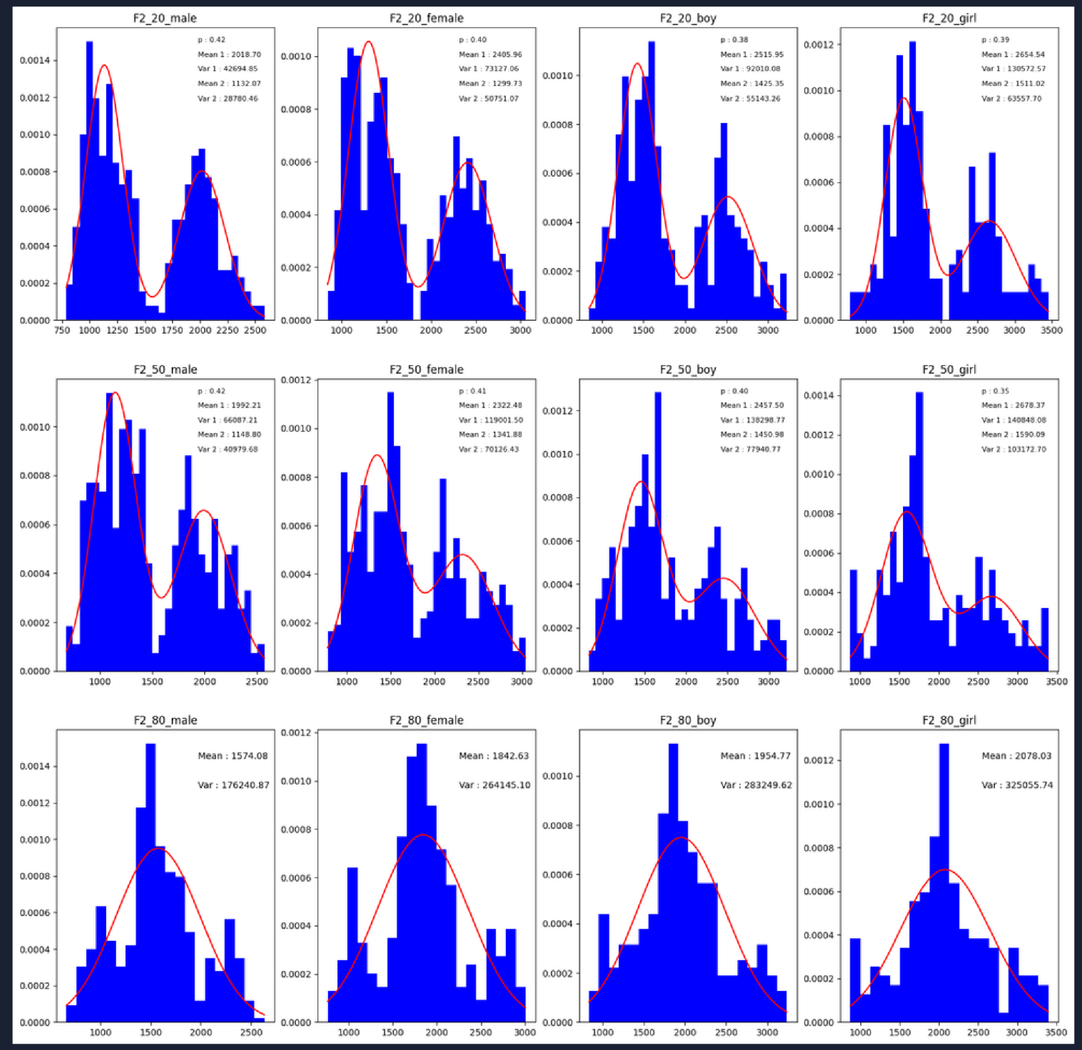


Observations:

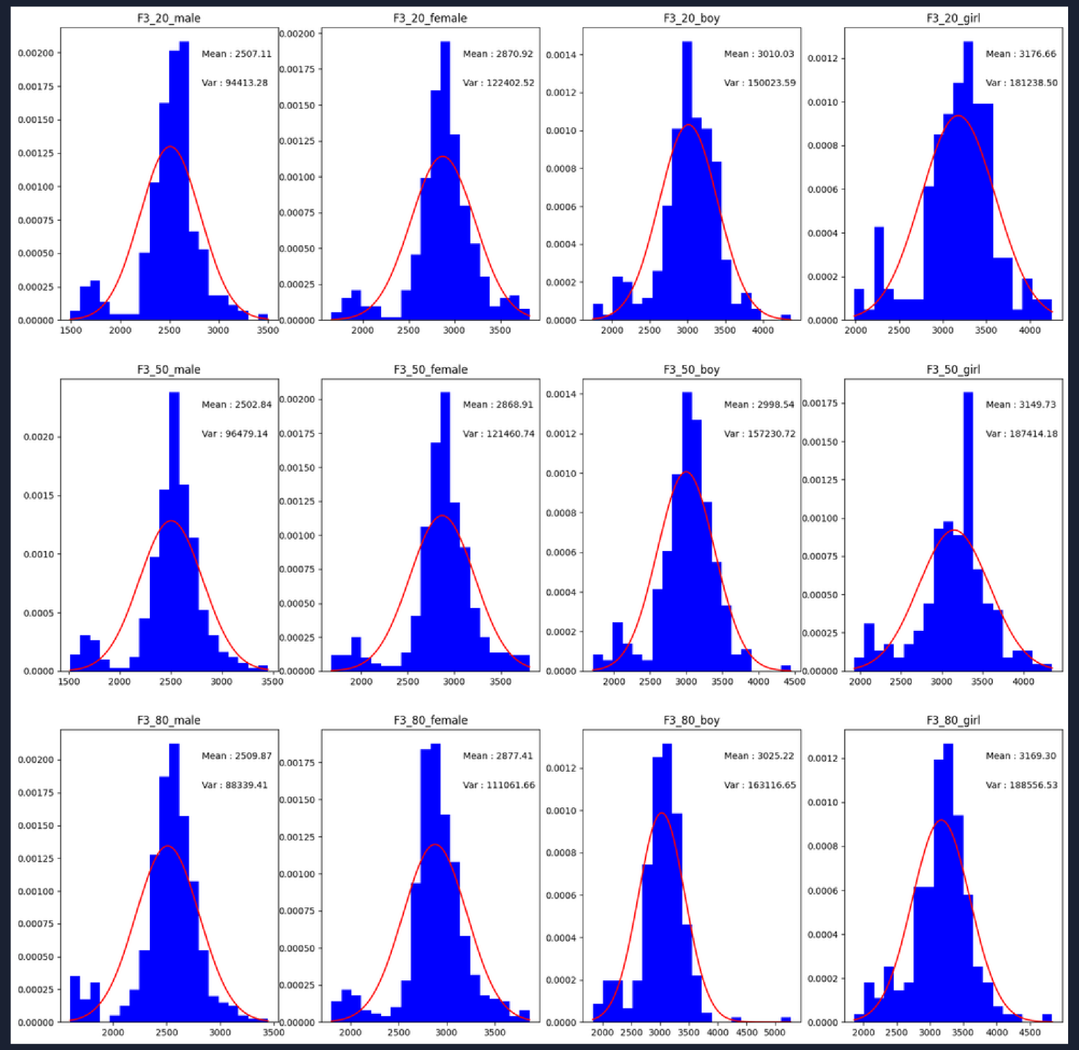
- Upon visualizing the results using a mixture of two normal distributions, it's evident that they align more closely with the observed data.
- The likelihood of the mixture model capturing the underlying distribution appears to be higher.

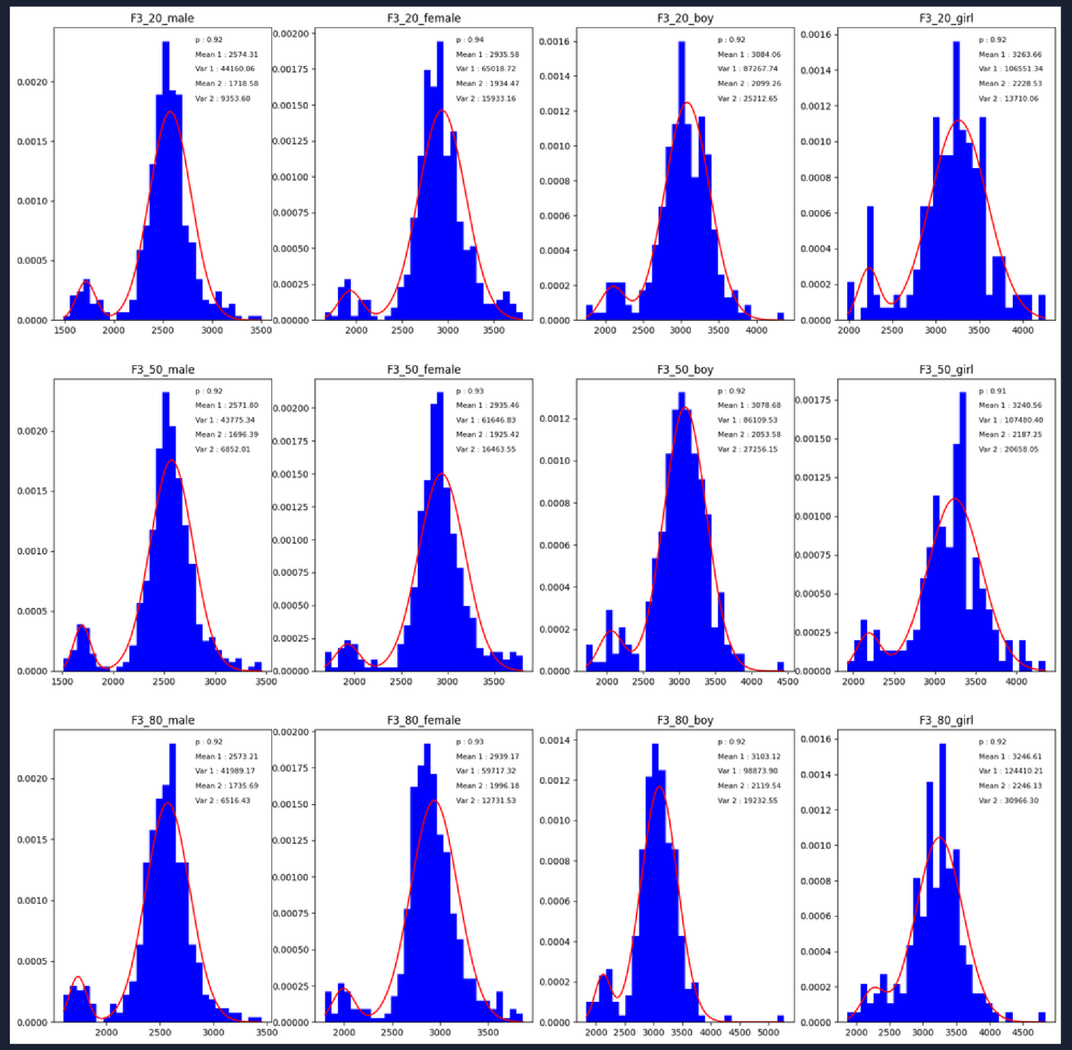
- Similarly for F2: F2_20 and F2_50 are visualized using mixture of 2 normal distributions and F2_80 as single normal distribution





- Similarly for F3: F3_20 F3_50 and F2_80 are visualized using mixture of 2 normal distributions





PETERSONBARNEY DATASETBREAKDOWN

1. Index: This column indicates the sex and age group:
    - '1' stands for male.
    - '2' stands for female.
    - '3' stands for children.
2. Speaker_Id: This represents ID of Speaker
3. Vowel_Id: This represents ID of Vowel
4. Vowel: Contains the Vowel with each having a repetition of 2
5. F0-F4: These columns represent the formant frequencies F0 to F4 of the vowel sound. Formant frequencies are the resonant frequencies of the vocal tract.

```
df = pd.read_csv("./peterson_barney_data.csv", sep=';')
```

- reading data

```
df = df.rename(columns={"gender": "Sex", "spkid": "ID", "f0":"F0"})
```

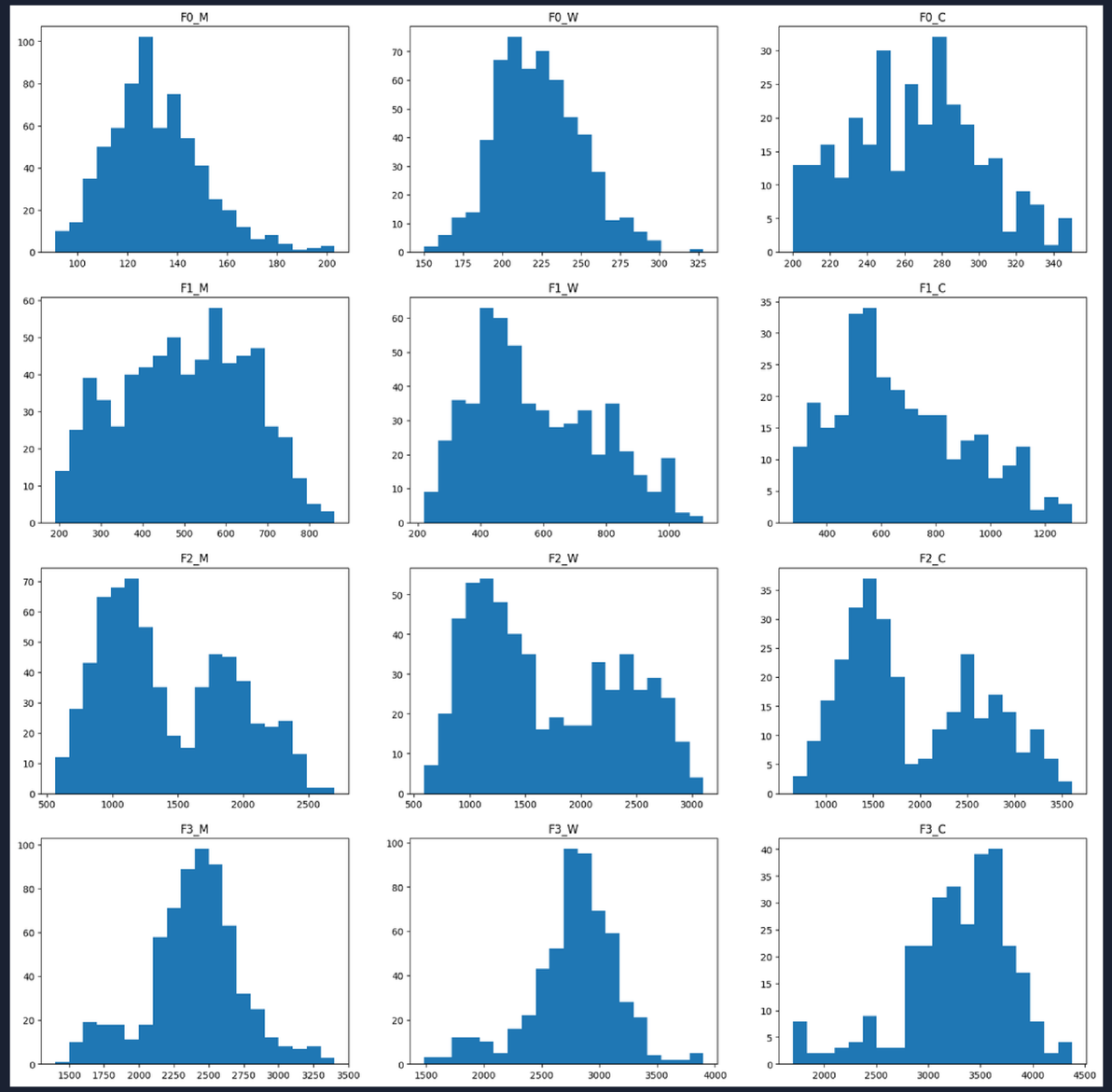
- renaming columns

```
# Code for dropping columns with any unfilled entry
df = df.replace(0, np.nan)
df = df.dropna()
df = df.reset_index()
df = df.drop("index", axis=1)
df.head()
```

- dropping columns with empty entries

```
plots = 4
types = df['Sex'].nunique()
typelist = df['Sex'].unique()
fig,ax = plt.subplots(nrows=plots,ncols=types,figsize=[20,20])
for i in range(plots):
    for j,instance in enumerate(typelist):
        subset = df[df['Sex']==instance]
        ax[i][j].hist(subset['F'+str(i)],bins=20)
        ax[i][j].set_title('F'+str(i)+'_'+instance)
plt.show()
```

- code to visualize data by plotting histograms



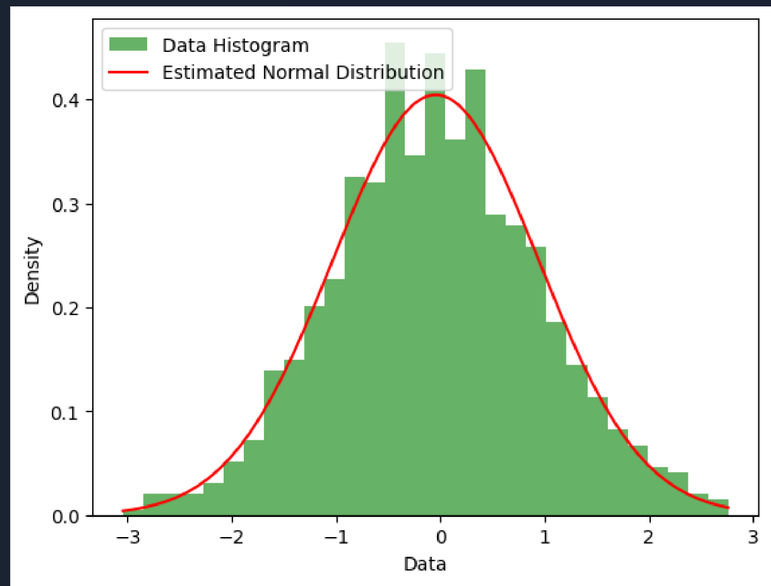
- Again, Histogram plots closely resembles that of normal and mixture of normal distributions.

Estimating Mean and variance using EM-algorithm for single normal distibutions

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


def compute_responsibilities_single(data, mean, variance):
    return np.ones_like(data)

def update_parameters_single(data, responsibility):
    mean = np.sum(responsibility * data) / np.sum(responsibility)
    variance = np.sum(responsibility * (data - mean)**2) / np.sum(responsibility)
    return mean, variance


def em_algorithm_single(data, initial_mean, initial_variance, max_iter=100, tolerance=1e-6):
    mean = initial_mean
    variance = initial_variance

    for i in range(max_iter):
        # M-step
        responsibility = compute_responsibilities_single(data, mean, variance)
        new_mean, new_variance = update_parameters_single(data, responsibility)

        # Check convergence
        if np.abs(new_mean - mean) < tolerance and np.abs(new_variance - variance) < tolerance:
            break

        mean, variance = new_mean, new_variance

    return mean, variance
```

Executing EM-algorithm and visualizing results

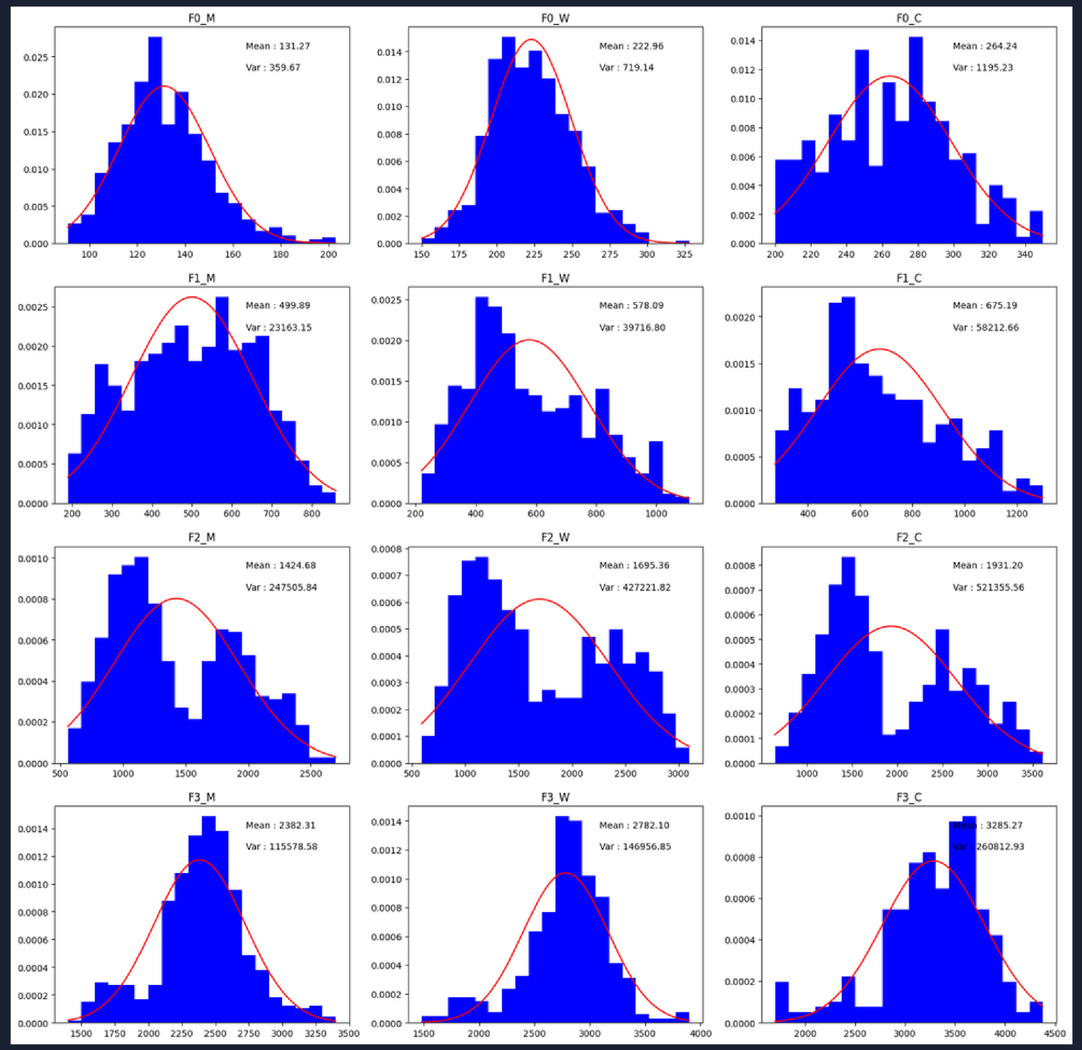
```python
np.random.seed(0)
n_samples = 1000
mean_true = 0
variance_true = 1
data = np.random.normal(mean_true, np.sqrt(variance_true), n_samples)

# Initialize parameters
initial_mean = 0
initial_variance = 1

# Run EM algorithm for single component
mean_em, variance_em = em_algorithm_single(data, initial_mean, initial_variance)

# Plot histogram of data
plt.hist(data, bins=30, density=True, alpha=0.6, color='g', label='Data Histogram')

# Plot estimated normal distribution
x = np.linspace(min(data), max(data), 1000)
pdf = norm.pdf(x, mean_em, np.sqrt(variance_em))
plt.plot(x, pdf, 'r-', label='Estimated Normal Distribution')

plt.xlabel('Data')
plt.ylabel('Density')
plt.legend()
plt.show()

# Print estimates
print("EM Estimates (Single Component):")
print("mean:", mean_em)
print("variance:", variance_em)
```



- **Visualizing results for F0,F1,F2,F3,F4 and all males females boy and girl assuming single normal distribution**

```
plots = 4
types = df['Sex'].nunique()
typelist = df['Sex'].unique()
fig,ax = plt.subplots(nrows=plots,ncols=types,figsize=[20,20])
for i in range(plots):
    for j,instance in enumerate(typelist):
        subset = df[df['Sex']==instance]
        data = subset['F'+str(i)].values
        initial_mean = 0
        initial_variance = 1
        mean_em, variance_em = em_algorithm_single(data, initial_mean, initial_variance)
        x = np.linspace(min(data), max(data), 1000)
        pdf = norm.pdf(x, mean_em, np.sqrt(variance_ (function) color: Any
        ax[i][j].hist(data, bins=20, density=True,  color='b', label='Data Histogram')
        ax[i][j].plot(x, pdf, 'r-', label='Estimated Normal Distribution')
        ax[i][j].set_title('F'+str(i)+'_'+instance)
        ax[i][j].text(0.65,0.9,f"Mean : {mean_em:.2f}",transform=ax[i][j].transAxes)
        ax[i][j].text(0.65,0.8,f"Var : {variance_em:.2f}",transform=ax[i][j].transAxes)

plt.show()
```

Plots:



Observations:

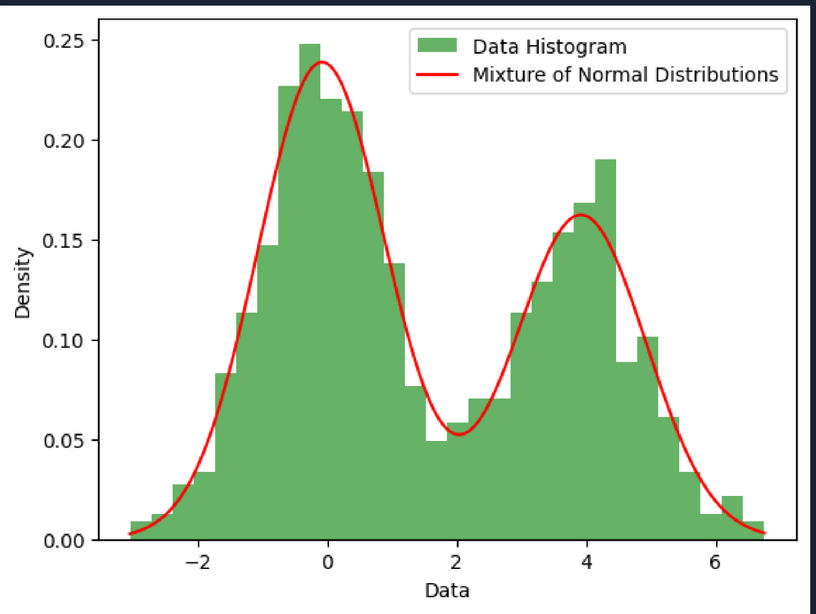
- F0, F1 follow a single normal distributions but we can observe that F2, F3 don’t follow a single normal distributions.
- F2, F3 are comprised of a mixture of two normal distributions.
- This suggests that the data might be more accurately represented using a mixture model rather than a single normal distribution.

EM Algorithm for mixture of two normal distributions

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

# E-step: Compute responsibilities
def compute_responsibilities(data, p, mean1, variance1, mean2, variance2):
    likelihood1 = p * norm.pdf(data, mean1, np.sqrt(variance1)+1e-10)
    likelihood2 = (1 - p) * norm.pdf(data, mean2, np.sqrt(variance2)+1e-10)
    total_likelihood = likelihood1 + likelihood2
    total_likelihood[total_likelihood == 0] = np.finfo(float).eps  # Avoid division by zero
    responsibility1 = likelihood1 / total_likelihood
    responsibility2 = likelihood2 / total_likelihood
    return responsibility1, responsibility2

# M-step: Update parameters
# M-step: Update parameters
def update_parameters(data, responsibility1, responsibility2):
    p = np.mean(responsibility1)

    # Handle the case where responsibility is very small or zero
    sum_responsibility1 = np.sum(responsibility1)
    mean1 = np.sum(responsibility1 * data) / (sum_responsibility1 + 1e-10)
    variance1 = np.sum(responsibility1 * (data - mean1)**2) / (sum_responsibility1 + 1e-10)

    sum_responsibility2 = np.sum(responsibility2)
    mean2 = np.sum(responsibility2 * data) / (sum_responsibility2 + 1e-10)
    variance2 = np.sum(responsibility2 * (data - mean2)**2) / (sum_responsibility2 + 1e-10)

    return p, mean1, variance1, mean2, variance2

# EM algorithm for mixture of normal distributions
def em_algorithm(data, initial_p, initial_mean1, initial_variance1, initial_mean2, initial_variance2,
max_iter=100, tolerance=1e-6):
    p = initial_p
    mean1 = initial_mean1
    variance1 = initial_variance1
    mean2 = initial_mean2
    variance2 = initial_variance2


    for i in range(max_iter):
        # E-step
        responsibility1, responsibility2 = compute_responsibilities(data, p, mean1, variance1, mean2,
        variance2)
        # M-step
        new_p, new_mean1, new_variance1, new_mean2, new_variance2 = update_parameters(data,
        responsibility1, responsibility2)

        # Check convergence
        if np.abs(new_p - p) < tolerance and np.abs(new_mean1 - mean1) < tolerance and np.abs
        (new_variance1 - variance1) < tolerance \
            and np.abs(new_mean2 - mean2) < tolerance and np.abs(new_variance2 - variance2) <
            tolerance:
            break

        p, mean1, variance1, mean2, variance2 = new_p, new_mean1, new_variance1, new_mean2,
        new_variance2

    return p, mean1, variance1, mean2, variance2
```

Executing two normal distibutions EM Algorithm

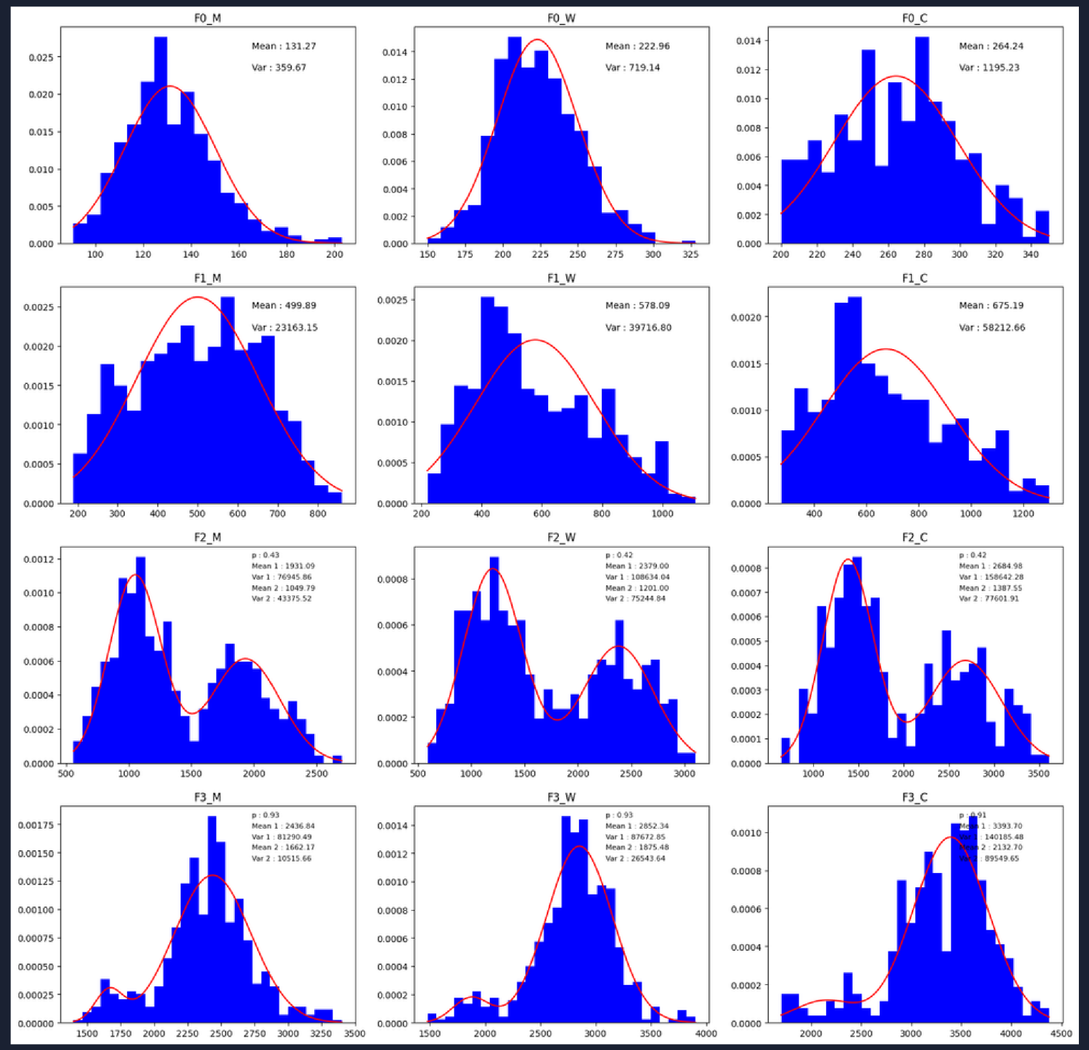
```python
# Generate synthetic data following a mixture of two normal distributions
np.random.seed(0)
n_samples = 1000
p_true = 0.6
mean1_true = 0
variance1_true = 1
mean2_true = 4
variance2_true = 1
data = np.concatenate([np.random.normal(mean1_true, np.sqrt(variance1_true), int(n_samples * p_true)),
                       np.random.normal(mean2_true, np.sqrt(variance2_true), int(n_samples * (1 -
                       p_true)))])

# Initialize parameters
initial_p = 0.5
initial_mean1 = 0
initial_variance1 = 1
initial_mean2 = 1
initial_variance2 = 1

# Run EM algorithm for mixture of normal distributions
p_em, mean1_em, variance1_em, mean2_em, variance2_em = em_algorithm(data, initial_p, initial_mean1,
initial_variance1, initial_mean2, initial_variance2)
print(data.shape)
# Plot histogram of data
plt.hist(data, bins=30, density=True, alpha=0.6, color='g', label='Data Histogram')

# Plot estimated normal distributions
x = np.linspace(min(data), max(data), 1000)
pdf1 = norm.pdf(x, mean1_em, np.s (variable) p_em: float | Any
pdf2 = norm.pdf(x, mean2_em, np.s
mixture_pdf = p_em * pdf1 + (1 - p_em) * pdf2
plt.plot(x, mixture_pdf, 'r-', label='Mixture of Normal Distributions')

plt.xlabel('Data')
plt.ylabel('Density')
plt.legend()
plt.show()

# Print MLE estimates
print("EM Estimates (Mixture of Normal Distributions):")
print("p:", p_em)
print("mean1:", mean1_em)
print("variance1:", variance1_em)
print("mean2:", mean2_em)
print("variance2:", variance2_em)
```



```
plots = 4
types = df['Sex'].nunique()
typelist = df['Sex'].unique()
fig,ax = plt.subplots(nrows=plots,ncols=types,figsize=[20,20])
for i in range(plots):
    for j,instance in enumerate(typelist):
        if i==0 or i==1:
            subset = df[df['Sex']==instance]
            data = subset['F'+str(i)].values
            initial_mean = 0
            initial_variance = 1
            mean_em, variance_em = em_algorithm_single(data, initial_mean, initial_variance)
            x = np.linspace(min(data), max(data), 1000)
            pdf = norm.pdf(x, mean_em, np.sqrt(variance_em))
            ax[i][j].hist(data, bins=20, density=True,  color='b', label='Data Histogram')
            ax[i][j].plot(x, pdf, 'r-', label='Estimated Normal Distribution')
            ax[i][j].set_title('F'+str(i)+'_'+instance)
            ax[i][j].text(0.65,0.9,f"Mean : {mean_em:.2f}",transform=ax[i][j].transAxes)
            ax[i][j].text(0.65,0.8,f"Var : {variance_em:.2f}",transform=ax[i][j].transAxes)
        elif i==2 or i==3:
            subset = df[df['Sex']==instance]
            data = subset['F'+str(i)].values
            initial_p = 0.3
            initial_mean1 = np.mean(data)
            initial_variance1 = np.var(data)
            initial_mean2 = np.mean(data)/2
            initial_variance2 = np.var(data)/2
            p_em, mean1_em, variance1_em, mean2_em, variance2_em = em_algorithm(data, initial_p,
            initial_mean1, initial_variance1,initial_mean2, initial_variance2)
            x = np.linspace(min(data), max(data), 1000)
            pdf1 = norm.pdf(x, mean1_em, np.sqrt(variance1_em))
            pdf2 = norm.pdf(x, mean2_em, np.sqrt(variance2_em))
            mixture_pdf = p_em * pdf1 + (1 - p_em) * pdf2
            ax[i][j].hist(data, bins=30, density=True, alpha=1, color='b', label='Data Histogram')
            ax[i][j].plot(x, mixture_pdf, 'r-', label='Mixture of Normal Distributions')
            ax[i][j].set_title('F'+str(i)+'_'+instance)
            ax[i][j].text(0.65,0.95,f"p : {p_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.9,f"Mean 1 : {mean1_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.85,f"Var 1 : {variance1_em:.2f}",transform=ax[i][j].transAxes,
            fontsize=8)
            ax[i][j].text(0.65,0.8,f"Mean 2 : {mean2_em:.2f}",transform=ax[i][j].transAxes,fontsize=8)
            ax[i][j].text(0.65,0.75,f"Var 2 : {variance2_em:.2f}",transform=ax[i][j].transAxes,
            fontsize=8)
plt.show()
```

- **Visualizing data for F0, F1 as single normal distributions and F2, F3 as mixture of two normal distributions**

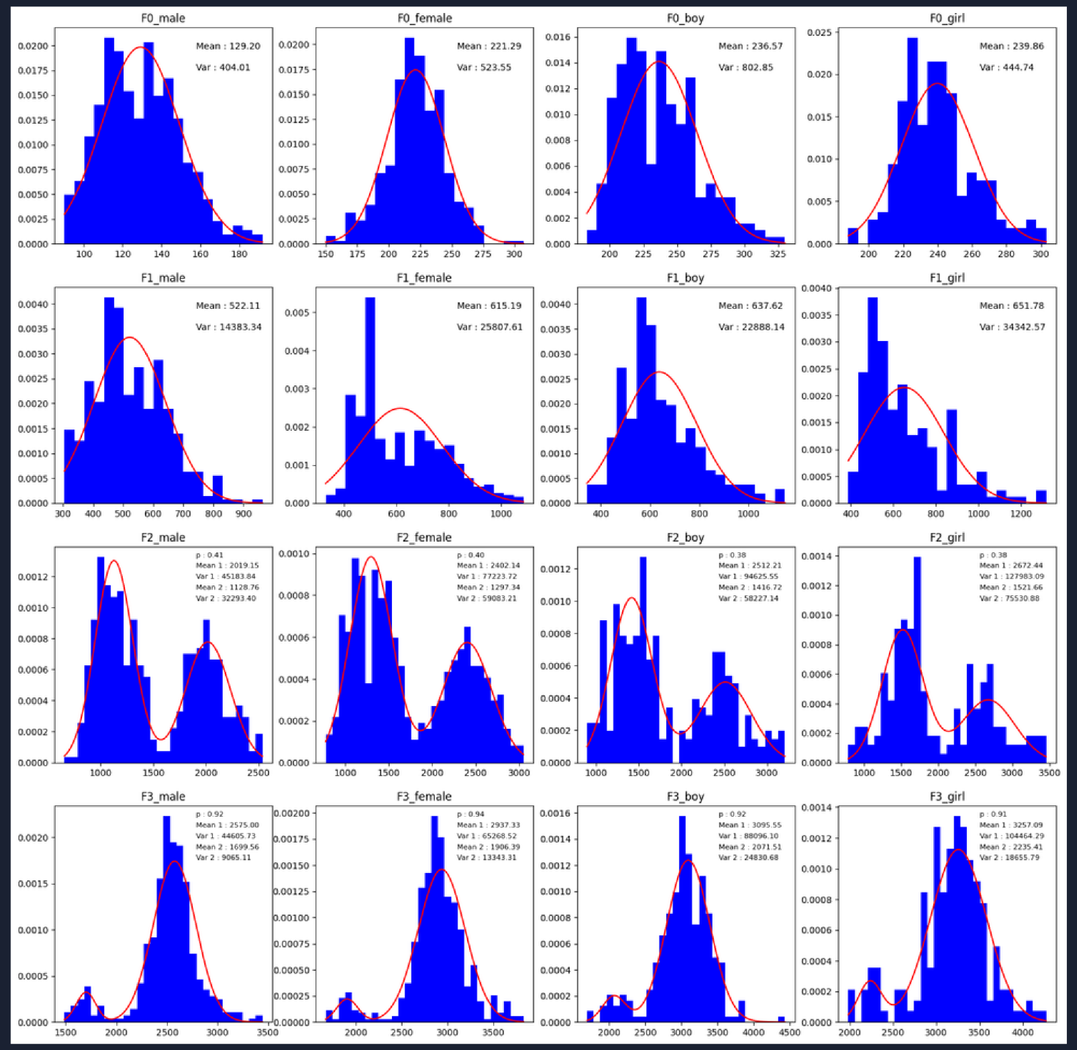
Plots:



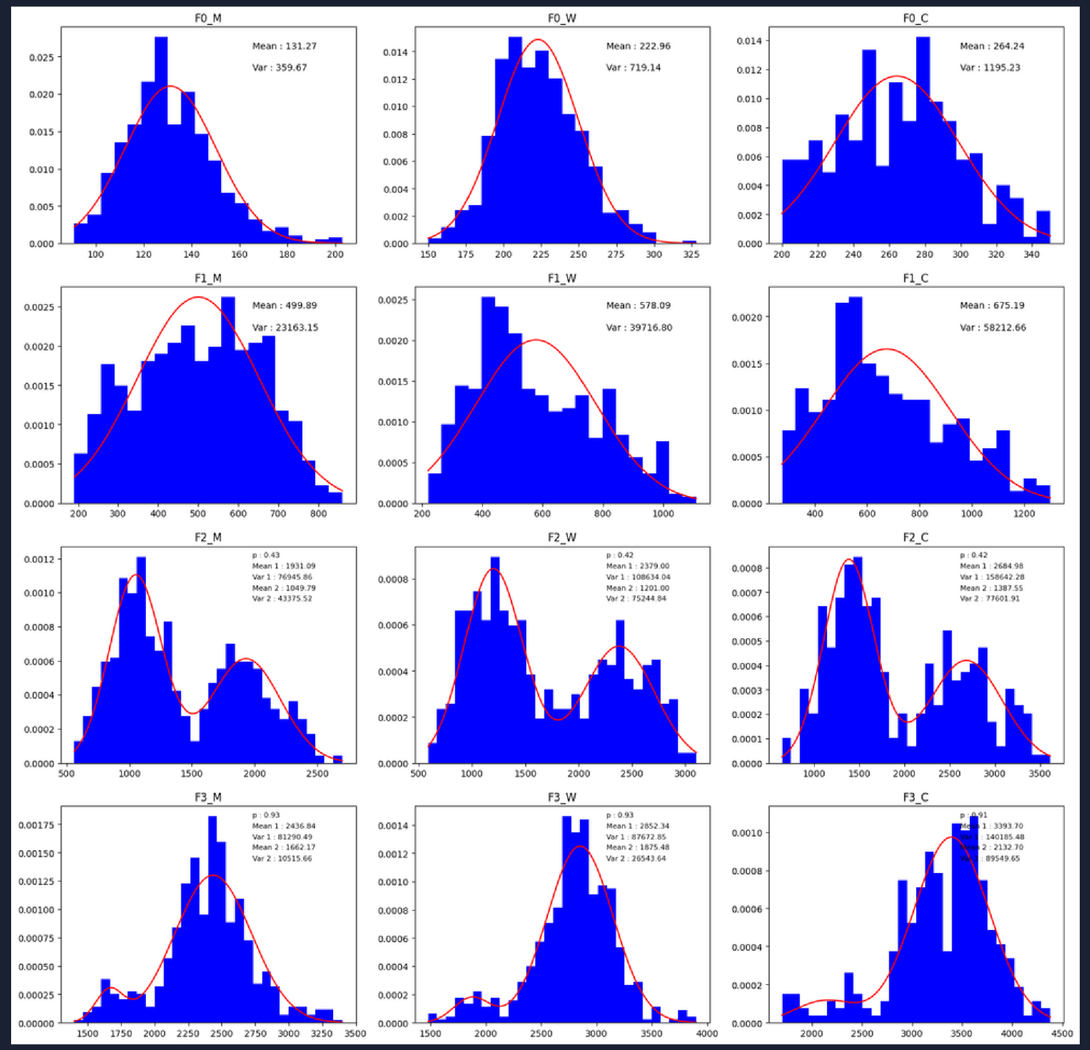
Observations:

- Upon visualizing the results using a mixture of two normal distributions, it's evident that they align more closely with the observed data.
- The likelihood of the mixture model capturing the underlying distribution appears to be higher.

Comparing Hillenbrand results and Peterson Barney results:



Hillenbrand



Peterson-Barney

Summary:

- After comparing the results from the Hillenbrand and Peterson-Barney datasets, it's notable that their 99% confidence intervals for F0, F1, F2, and F3 are similar, (their means, variances, and p-values are similar).
- It’s apparent that boys and girls exhibit similar data patterns, as indicated by the likelihood analysis, where the means and variances are comparable.
- The 99% confidence intervals almost overlap, so boys and girls cannot be differentiated or estimated based solely on given frequencies.
- However, given the data, it is possible to estimate differences between men and women.