Support Vector Machine Model - Gender Recognition by Voice Dataset

Objective

- The objective is to identify a voice as male or female, based upon acoustic properties of the voice and speech.

Support Vector Machine

- Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression.They are extremely popular because of their ability to handle multiple continuous and categorical variables.
- The objective of the support vector machine algorithm is to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps −
- a. First, SVM will generate hyperplanes iteratively that segregates the classes in best way.
- b. Then, it will choose the hyperplane that separates the classes correctly.

SVM Terminology

- Hyperplane: Hyperplane is the decision boundary that is used to separate the data points of different classes in a feature space.
- Support Vectors: Support vectors are the closest data points to the hyperplane, which makes a critical role in deciding the hyperplane and margin.
- Margin: Margin is the distance between the support vector and hyperplane. The main objective of the support vector machine algorithm is to maximize the margin. The wider margin indicates better classification performance.
- Hard Margin: The maximum-margin hyperplane or the hard margin hyperplane is a hyperplane that properly separates the data points of different categories without any misclassifications.
- Soft Margin: When the data is not perfectly separable or contains outliers, SVM permits a soft margin technique. Each data point has a slack variable introduced by the soft-margin SVM formulation, which

softens the strict margin requirement and permits certain misclassifications or violations. It discovers a compromise between increasing the margin and reducing violations.

- C: Margin maximisation and misclassification fines are balanced by the regularisation parameter C in SVM. The penalty for going over the margin or misclassifying data items is decided by it. A stricter penalty is imposed with a greater value of C, which results in a smaller margin and perhaps fewer misclassifications.
- Hinge Loss: A typical loss function in SVMs is hinge loss. It punishes incorrect classifications or margin

Dataset source & brief

- The dataset Gender Recognition by Voice and Speech Analysis is sourced from Kaggle. It consists of 3,168 recorded voice samples, collected from male and female speakers.
- The acoustic properties of each voice are: meanfreq: mean frequency (in kHz), sd: standard deviation of frequency, median: median frequency (in kHz), Q25: first quantile (in kHz), Q75: third quantile (in kHz), IQR: interquantile range (in kHz), skew: skewness (see note in specprop description), kurt: kurtosis (see note in specprop description), sp.ent: spectral entropy, sfm: spectral flatness, mode: mode frequency, centroid: frequency centroid (see specprop), peakf: peak frequency (frequency with highest energy), meanfun: average of fundamental frequency measured across acoustic signal, minfun: minimum fundamental frequency measured across acoustic signal, maxfun: maximum fundamental frequency measured across acoustic signal, meandom: average of dominant frequency measured across acoustic signal, mindom: minimum of dominant frequency measured across acoustic signal, maxdom: maximum of dominant frequency measured across acoustic signal, dfrange: range of dominant frequency measured across acoustic signal, modindx: modulation index. Calculated as the accumulated absolute difference between adjacent measurements of fundamental frequencies divided by the frequency range and the target variable label: male or female

Import the required libraries

In [1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

Load & read the dataset

In [24]:

```
df=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stats and ML\Dataset\voice.csv")
df.head()
```

Out[24]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.89
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.89
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.84
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.96
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.97

5 rows × 21 columns

Check basic information

In [25]:

```
df.shape    # check shape
```

Out[25]:

```
(3168, 21)
```

In [26]:

```
df.info()   # check info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   meanfreq  3168 non-null   float64
 1   sd        3168 non-null   float64
 2   median    3168 non-null   float64
 3   Q25       3168 non-null   float64
 4   Q75       3168 non-null   float64
 5   IQR       3168 non-null   float64
 6   skew      3168 non-null   float64
 7   kurt      3168 non-null   float64
 8   sp.ent    3168 non-null   float64
 9   sfm       3168 non-null   float64
 10  mode      3168 non-null   float64
 11  centroid  3168 non-null   float64
 12  meanfun   3168 non-null   float64
 13  minfun    3168 non-null   float64
 14  maxfun    3168 non-null   float64
 15  meandom   3168 non-null   float64
 16  mindom    3168 non-null   float64
 17  maxdom    3168 non-null   float64
 18  dfrange   3168 non-null   float64
 19  modindx   3168 non-null   float64
 20  label     3168 non-null   object 
dtypes: float64(20), object(1)
memory usage: 519.9+ KB
```

In [27]:

```
df.describe().T  #statistical summary
```

Out[27]:

	count	mean	std	min	25%	50%	75%	m
meanfreq	3168.0	0.180907	0.029918	0.039363	0.163662	0.184838	0.199146	0.2511
sd	3168.0	0.057126	0.016652	0.018363	0.041954	0.059155	0.067020	0.1152
median	3168.0	0.185621	0.036360	0.010975	0.169593	0.190032	0.210618	0.2612
Q25	3168.0	0.140456	0.048680	0.000229	0.111087	0.140286	0.175939	0.2473
Q75	3168.0	0.224765	0.023639	0.042946	0.208747	0.225684	0.243660	0.2734
IQR	3168.0	0.084309	0.042783	0.014558	0.042560	0.094280	0.114175	0.2522
skew	3168.0	3.140168	4.240529	0.141735	1.649569	2.197101	2.931694	34.7254
kurt	3168.0	36.568461	134.928661	2.068455	5.669547	8.318463	13.648905	1309.6128
sp.ent	3168.0	0.895127	0.044980	0.738651	0.861811	0.901767	0.928713	0.9819
sfm	3168.0	0.408216	0.177521	0.036876	0.258041	0.396335	0.533676	0.8429
mode	3168.0	0.165282	0.077203	0.000000	0.118016	0.186599	0.221104	0.2800
centroid	3168.0	0.180907	0.029918	0.039363	0.163662	0.184838	0.199146	0.2511
meanfun	3168.0	0.142807	0.032304	0.055565	0.116998	0.140519	0.169581	0.2376
minfun	3168.0	0.036802	0.019220	0.009775	0.018223	0.046110	0.047904	0.2040
maxfun	3168.0	0.258842	0.030077	0.103093	0.253968	0.271186	0.277457	0.2791
meandom	3168.0	0.829211	0.525205	0.007812	0.419828	0.765795	1.177166	2.9576
mindom	3168.0	0.052647	0.063299	0.004883	0.007812	0.023438	0.070312	0.4589
maxdom	3168.0	5.047277	3.521157	0.007812	2.070312	4.992188	7.007812	21.8671
dfrange	3168.0	4.994630	3.520039	0.000000	2.044922	4.945312	6.992188	21.8437
modindx	3168.0	0.173752	0.119454	0.000000	0.099766	0.139357	0.209183	0.9323

In [28]:

```
df.corr()   #correlation
```

Out[28]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	ku
meanfreq	1.000000	-0.739039	0.925445	0.911416	0.740997	-0.627605	-0.322327	-0.3160
sd	-0.739039	1.000000	-0.562603	-0.846931	-0.161076	0.874660	0.314597	0.3462
median	0.925445	-0.562603	1.000000	0.774922	0.731849	-0.477352	-0.257407	-0.2433
Q25	0.911416	-0.846931	0.774922	1.000000	0.477140	-0.874189	-0.319475	-0.3501
Q75	0.740997	-0.161076	0.731849	0.477140	1.000000	0.009636	-0.206339	-0.1488
IQR	-0.627605	0.874660	-0.477352	-0.874189	0.009636	1.000000	0.249497	0.3161
skew	-0.322327	0.314597	-0.257407	-0.319475	-0.206339	0.249497	1.000000	0.9770
kurt	-0.316036	0.346241	-0.243382	-0.350182	-0.148881	0.316185	0.977020	1.0000
sp.ent	-0.601203	0.716620	-0.502005	-0.648126	-0.174905	0.640813	-0.195459	-0.1276
sfm	-0.784332	0.838086	-0.661690	-0.766875	-0.378198	0.663601	0.079694	0.1098
mode	0.687715	-0.529150	0.677433	0.591277	0.486857	-0.403764	-0.434859	-0.4067
centroid	1.000000	-0.739039	0.925445	0.911416	0.740997	-0.627605	-0.322327	-0.3160
meanfun	0.460844	-0.466281	0.414909	0.545035	0.155091	-0.534462	-0.167668	-0.1945
minfun	0.383937	-0.345609	0.337602	0.320994	0.258002	-0.222680	-0.216954	-0.2032
maxfun	0.274004	-0.129662	0.251328	0.199841	0.285584	-0.069588	-0.080861	-0.0456
meandom	0.536666	-0.482726	0.455943	0.467403	0.359181	-0.333362	-0.336848	-0.3032
mindom	0.229261	-0.357667	0.191169	0.302255	-0.023750	-0.357037	-0.061608	-0.1033
maxdom	0.519528	-0.482278	0.438919	0.459683	0.335114	-0.337877	-0.305651	-0.2745
dfrange	0.515570	-0.475999	0.435621	0.454394	0.335648	-0.331563	-0.304640	-0.2727
modindx	-0.216979	0.122660	-0.213298	-0.141377	-0.216475	0.041252	-0.169325	-0.2055

Data pre - processing

In [29]:

```
df.duplicated().sum()  #check duplicates
```

Out[29]:

2

In [30]:

```
df=df.drop_duplicates()  # remove duplicates
```

In [31]:

```
df.isnull().sum()   #check missing values
```

Out[31]:

```
meanfreq    0
sd          0
median      0
Q25         0
Q75         0
IQR         0
skew        0
kurt        0
sp.ent      0
sfm         0
mode        0
centroid    0
meanfun     0
minfun      0
maxfun      0
meandom     0
mindom      0
maxdom      0
dfrange     0
modindx     0
label       0
dtype: int64
```

In [32]:

```
df['label'].value_counts() # check balance of data
```

Out[32]:

```
male      1583
female    1583
Name: label, dtype: int64
```

In [33]:

```
# Lable encoder for Target variable
df['label']=df['label'].astype('category')
df['label']=df['label'].cat.codes
```

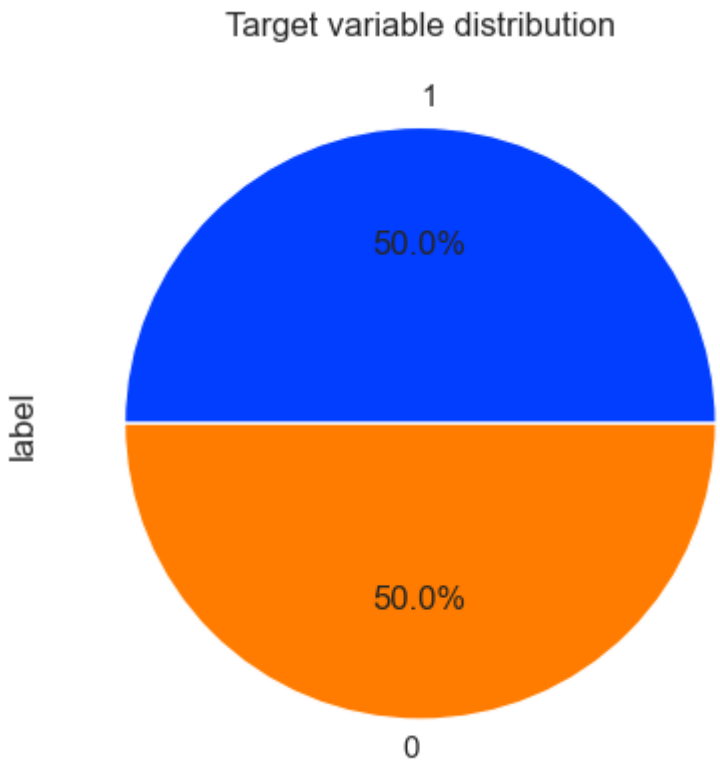
Exploratory Data Analysis

In [34]:

```
sns.set_theme(palette='bright',style='whitegrid')
```

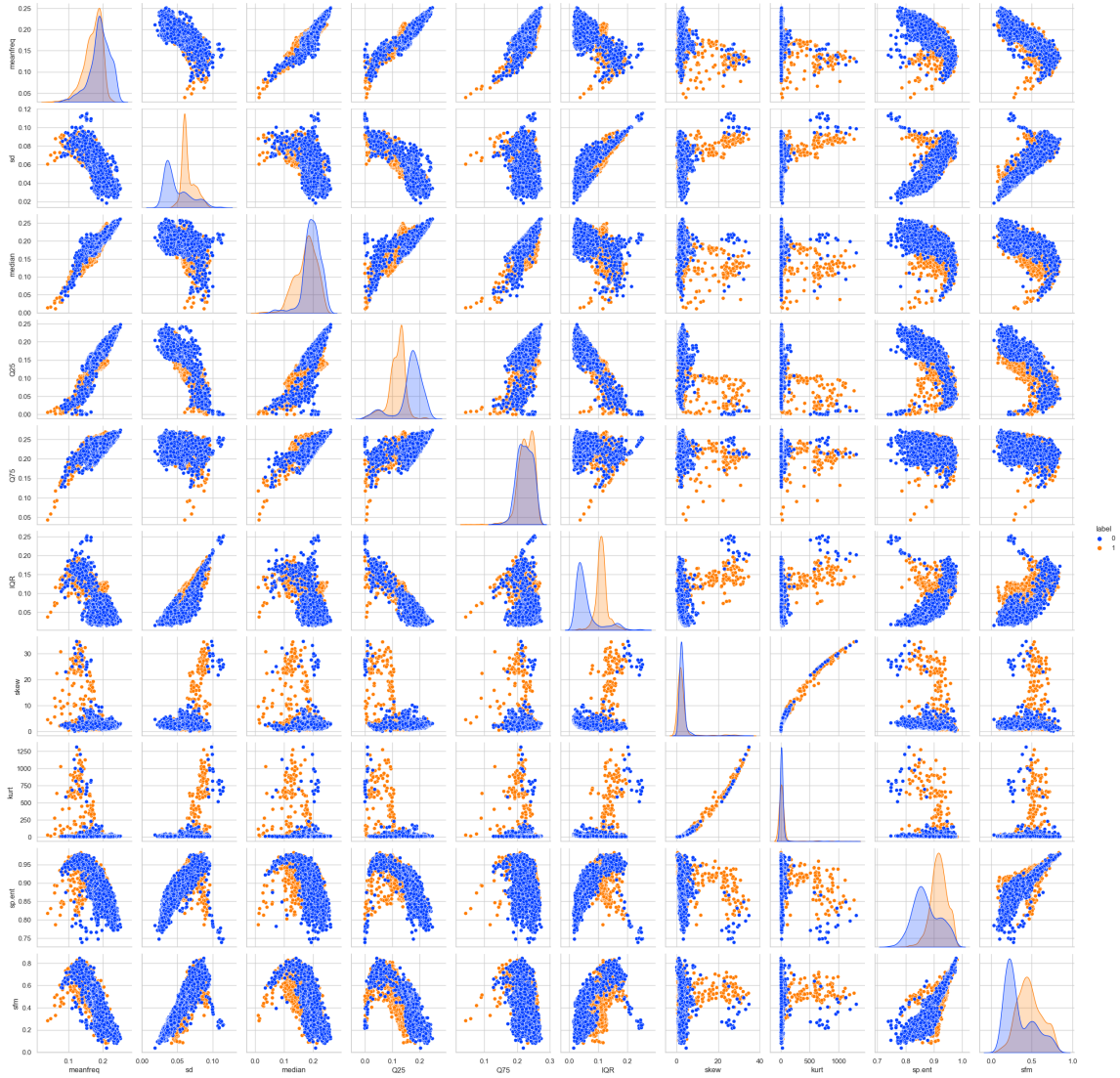
In [35]:

```
df['label'].value_counts().plot(kind='pie',autopct='%0.1f%%')
plt.title('Target variable distribution')
plt.show()
```



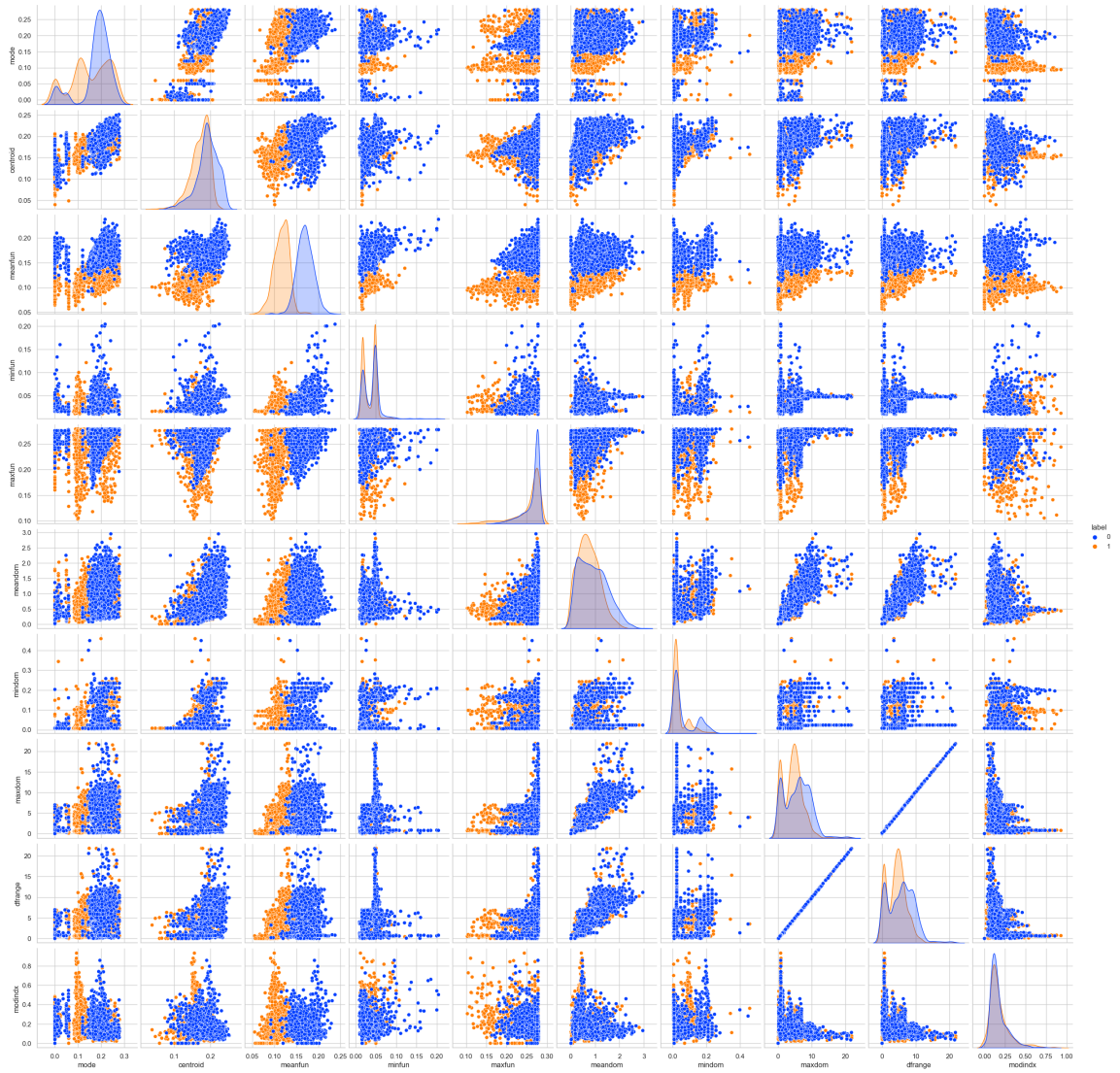
In [20]:

```
sns.pairplot(df[['meanfreq','sd','median','Q25','Q75','IQR','skew',
                 'kurt','sp.ent', 'sfm','label']], hue='label')
plt.show()
```



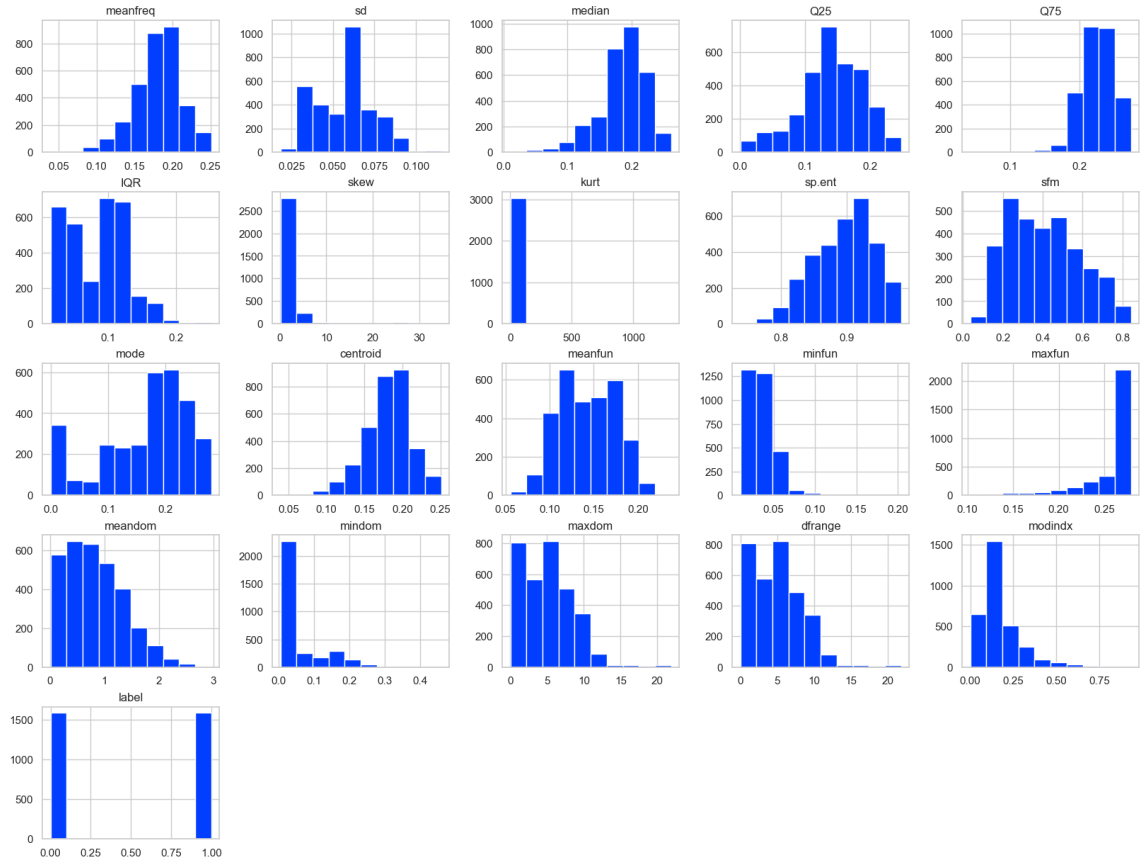
In [21]:

```
sns.pairplot(df[['mode','centroid','meanfun','minfun','maxfun','meandom',
                 'mindom','maxdom','dfrange','modindx', 'label']], hue='label')
plt.show()
```



In [13]:

```
df.hist(figsize=(20,15))
plt.show()
```



Data Splitting

In [36]:

```
# split the data into x and y
x = df.drop(['label',], axis=1)
y = df[['label']]
```

In [37]:

```
x.head(2)
```

Out[37]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892

In [38]:

```
y.head(2)
```

Out[38]:

	label
0	1
1	1

Feature Scaling

In [39]:

```
sc=StandardScaler()
x1=sc.fit_transform(x)
pd.DataFrame(x1).head()
```

Out[39]:

	0	1	2	3	4	5	6	7	
0	-4.048763	0.427014	-4.224894	-2.575370	-5.693565	-0.214711	2.292464	1.762278	-0.03
1	-3.840558	0.611291	-3.999247	-2.486167	-5.588933	-0.258411	4.546556	4.431537	-0.06
2	-3.462553	1.603266	-4.095821	-2.706234	-3.928446	0.909211	6.511582	7.323867	-1.08
3	-0.991527	0.899560	-0.758836	-0.900952	-0.710567	0.632620	-0.449899	-0.240166	1.51
4	-1.530036	1.322037	-1.676492	-1.267871	-0.791400	1.005458	-0.480942	-0.239006	1.70

In [40]:

```
# split the training data into train and test
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=1)
```

Kernel & Kernel Trick

- Kernel is the mathematical function. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate.
- Some of the common kernel functions are Radial basis function(RBF), Linear, Polynomial and Sigmoid.

Model building, Prediction & Evaluation

1. Radial Basis Function Kernel (RBF) - (Defaut SVM)

- It is used to solve the problem of classifying datasets that cannot be separated linearly. It is known to have good performance with certain parameters, and the results of the training have a small error value compared to other kernels.

In [41]:

```
#Model building
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(x_train, y_train)


#Predict
y_pred_train_rbf = svm_rbf.predict(x_train)
y_pred_test_rbf = svm_rbf.predict(x_test)

#Evaluate
print("Training Accuracy - Rbf :", accuracy_score(y_train, y_pred_train_rbf))
print('------------------------'*5)
print("Test Accuracy - Rbf :", accuracy_score(y_test, y_pred_test_rbf))
```

```
Training Accuracy - Rbf : 0.6725908372827805
--------------------------------------------------------------------------------
-------------------------------------------
Test Accuracy - Rbf : 0.7018927444794952
```

2. Linear Kernel

- It is used when the data is Linearly separable. It is mostly used when there are a Large number of Features in a particular Data Set. It is mostly preferred for text-classification problems.

In [42]:

```
#Model building
svm_linear = SVC(kernel='linear')
svm_linear.fit(x_train, y_train)

#Predict
y_pred_train_linear = svm_linear.predict(x_train)
y_pred_test_linear = svm_linear.predict(x_test)

#Evaluate
print("Training Accuracy - Linear :", accuracy_score(y_train, y_pred_train_linear))
print('------------------------'*5)
print("Test Accuracy - Linear :", accuracy_score(y_test, y_pred_test_linear))
```

```
Training Accuracy - Linear : 0.9206161137440758
--------------------------------------------------------------------------------
-------------------------------------------
Test Accuracy - Linear : 0.9148264984227129
```

3. Polynomial Kernel

- It represents the similarity of the training sample vectors in a feature space. Polynomial kernels are also suitable for solving classification problems on normalized training datasets.

In [43]:

```
#Model building
svm_poly = SVC(kernel='poly')
svm_poly.fit(x_train, y_train)


#Predict
y_pred_train_poly = svm_poly.predict(x_train)
y_pred_test_poly = svm_poly.predict(x_test)


#Evaluate
print("Training Accuracy - Polynomial :", accuracy_score(y_train, y_pred_train_poly))
print('------------------------'*5)
print("Test Accuracy - Polynomial :", accuracy_score(y_test, y_pred_test_poly))
```

```
Training Accuracy - Polynomial : 0.5233017377567141
--------------------------------------------------------------------------------
-------------------------------------------
Test Accuracy - Polynomial : 0.4889589905362776
```

4. Sigmoid Kernel

- It is mostly preferred for neural networks. This kernel function is similar to a two-layer perceptron model of the neural network, which works as an activation function for neurons.

In [44]:

```
# Model building
svm_sigmoid = SVC(kernel='sigmoid')
svm_sigmoid.fit(x_train, y_train)

# Predict
y_pred_train_sigmoid = svm_sigmoid.predict(x_train)
y_pred_test_sigmoid = svm_sigmoid.predict(x_test)

#Evaluate
print("Training Accuracy - Sigmoid :", accuracy_score(y_train, y_pred_train_sigmoid))
print('------------------------'*5)
print("Test Accuracy - Sigmoid :", accuracy_score(y_test, y_pred_test_sigmoid))
```

```
Training Accuracy - Sigmoid : 0.37203791469194314
--------------------------------------------------------------------------------
-------------------------------------------
Test Accuracy - Sigmoid : 0.3722397476340694
```

Kernel model conclusion

- Linear Kernel model is showing the best result with Training accuracy at 92% and Test accuracy at 91%.
- Rbf kernel (defalut kernel) model is showing Training accuracy at 67% and Test accuracy at 70% considered as poor accuracy.
- Both Polynomial & Sigmoid kernel models are showing very poor results.
- We will not be considering Rbf,Polynomial & Sigmoid kernel models for cross validation.

Classification report- Linear Kernel Model

In [45]:

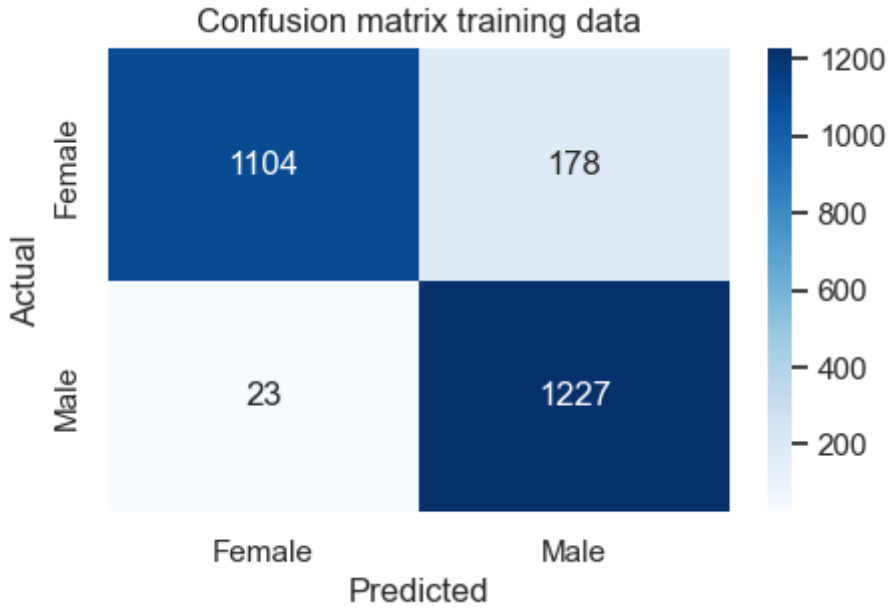
```
print("Training Accuracy - Linear :", classification_report(y_train, y_pred_train_linear
print('-----------------------'*5)
print("Test Accuracy - Linear :", classification_report(y_test, y_pred_test_linear))
```

```
Training Accuracy - Linear :              precision    recall  f1-score
support

           0       0.98      0.86      0.92      1282
           1       0.87      0.98      0.92      1250

    accuracy                           0.92      2532
   macro avg       0.93      0.92      0.92      2532
weighted avg       0.93      0.92      0.92      2532

-------------------------------------------------------------------------
-------------------------------------------
Test Accuracy - Linear :              precision    recall  f1-score   sup
port

           0       0.97      0.85      0.90       301
           1       0.88      0.98      0.92       333

    accuracy                           0.91       634
   macro avg       0.92      0.91      0.91       634
weighted avg       0.92      0.91      0.91       634
```

Confusion Matrix - Linear Kernel Model

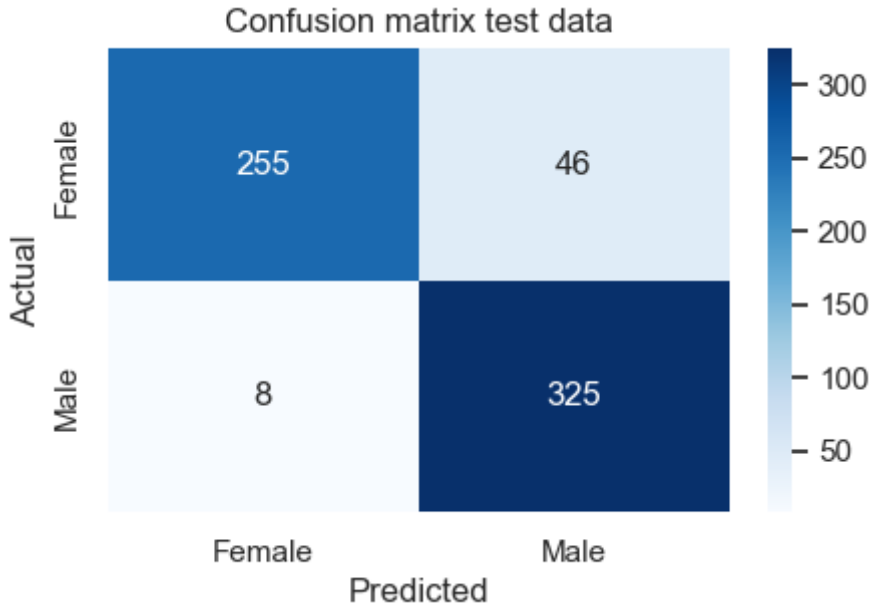
In [47]:

```
#plotting confusion matrix of Training data
Labels = ['Female', 'Male']
conf_matrix = confusion_matrix(y_train, y_pred_train_linear)
plt.figure(figsize=(5,3))
sns.heatmap(conf_matrix,xticklabels= Labels, yticklabels=Labels,cmap='Blues',
            annot=True, fmt='g')
plt.title("Confusion matrix training data")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [48]:

```
#plotting confusion matrix of test data
Labels = ['Female', 'Male']
conf_matrix = confusion_matrix(y_test, y_pred_test_linear)
plt.figure(figsize=(5,3))
sns.heatmap(conf_matrix,xticklabels= Labels, yticklabels=Labels,cmap='Blues',
            annot=True, fmt='g')
plt.title("Confusion matrix test data")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Cross Validation - Linear Kernel Model

In [49]:

```
train_accuracy = cross_val_score(svm_linear, x_train, y_train, cv=10)
test_accuracy = cross_val_score(svm_linear, x_test, y_test, cv=10)
print("Training - Mean Accuracy :", train_accuracy.mean())
print('------------------------'*5)
print("Training - Max Accuracy :", train_accuracy.max())
print('------------------------'*5)
print("Test - Mean Accuracy :", test_accuracy.mean())
print('------------------------'*5)
print("Test- Max Accuracy :", test_accuracy.max())
```

```
Training - Mean Accuracy : 0.9178565870965734
--------------------------------------------------------------------------------
----------------------------------------
Training - Max Accuracy : 0.9525691699604744
--------------------------------------------------------------------------------
----------------------------------------
Test - Mean Accuracy : 0.8766865079365079
--------------------------------------------------------------------------------
----------------------------------------
Test- Max Accuracy : 0.953125
```

Hyperparameters & Hyperparameter Tuning

- These are the parameters that are explicitly defined by the user to control the learning process.
- Hyperparameter tuning is basically referred to as tweaking the parameters of the model, which is basically a prolonged process.

GridSearchCV

- It takes a dictionary that describes the parameters that could be tried on a model to train it. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

In [50]:

```
param_grid = {'C':[0.1,1,10,100,1000],'gamma':[1,0.1,0.01,0.001,0.0001]}
grid = GridSearchCV(SVC(), param_grid, refit=True)

# model fitting
grid.fit(x_train, y_train)
grid_predict = grid.predict(x_test)
print(accuracy_score(y_test, grid_predict))
```

```
0.9384858044164038
```

Conclusion

- In this project, I tried to build Support Vector Machine model on Gender recognition by voice dataset.
- After preprocessing data, I used different Kernels for model building- Rbf (default), Linear, Polynomial & Sigmoid.
- Out of the 4 kernels Linear kernel had the highest accuracy- Training data accuracy at 92% and Test data accuracy at 91%.
- Rest 3 kernels gave poor results.
- Cross validation of Linear kernel yielded Training data accuracy of 91% and Test data accuracy of 87%.
- Next step was Hyperparameter tuning by using GridSearchCV, it gave accuracy score of 93%.
- The above points makes it a good model to recognise gender by voice which was the primary objective of this whole project.

In [ ]:

```

```