Gender Recognition by Voice

Бинарная классификация
на примере логистической регрессии

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
```

1. Get dataset

This database was created to identify a voice as male or female, based upon acoustic properties of the voice and speech. The dataset consists of 3,168 recorded voice samples, collected from male and female speakers. The voice samples are pre-processed by acoustic analysis in R using the seewave and tuneR packages, with an analyzed frequency range of 0hz-280hz (human vocal range)

The following acoustic properties of each voice are measured and included within the CSV:

·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
·label: male or female

Out[3]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	ma
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	...	0.059781	0.084279	0.015702	0.275
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	...	0.066009	0.107937	0.015826	0.250
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	...	0.077316	0.098706	0.015656	0.271
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0.727232	...	0.151228	0.088965	0.017798	0.250
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0.783568	...	0.135120	0.106398	0.016931	0.266

5 rows × 21 columns

2. Analyse dataset

In [4]:
```
print('\u2022 Размерность массива входных данных: \n\n', df.shape)
print('\n\u2022 Список названий столбцов: \n\n', df.columns)
print('\n\u2022 Тип переменных в столбцах:\n\n', df.dtypes)
print('\n\u2022 Проверяем, есть ли незаполненные данные:\n\n', df.isna().any())
```

• Размерность массива входных данных:

```
(3168, 21)
```

• Список названий столбцов:

```
Index(['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
       'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx', 'label'],
      dtype='object')
```

• Тип переменных в столбцах:

```
meanfreq    float64
sd          float64
median      float64
Q25         float64
Q75         float64
IQR         float64
skew        float64
kurt        float64
sp.ent      float64
sfm         float64
mode        float64
centroid    float64
meanfun     float64
minfun      float64
maxfun      float64
meandom     float64
mindom      float64
maxdom      float64
dfrange     float64
modindx     float64
label        object
```

• Проверяем, есть ли незаполненные данные:

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

In [5]:
```
# Кол-во м/ж записей (значения распределились поровну)
df['label'].value_counts()
```

Out[5]:
```
male      1584
female    1584
Name: label, dtype: int64
```

In [6]:
```
# Кол-во уникальных значений для м/ж записей
# Практически все значения уникальный, кроме: minfun, maxfun, mindow, maxdow, dfrange
df.groupby(['label']).nunique()
```

Out[6]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	maxfun	meandom	mindom	maxdo
label																		
female	1583	1583	1547	1552	1565	1545	1583	1583	1583	1583	...	1583	1583	676	64	1543	67	771
male	1583	1583	1562	1557	1532	1535	1583	1583	1583	1583	...	1583	1583	573	116	1503	55	743

2 rows × 21 columns

In [7]:
```
# Все признаки в выборке носят количественный характер, кроме label - категориальный, поэтому далее преобразуем его в бинарный
```

2.1. Preprocessing

In [8]:
```
# Кодируем пол бинарными значениями, проверяем на транспонированном векторе
df_label_int = pd.DataFrame(data=pd.factorize(df['label'])[0], columns=['label'])
df_label_int.T
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
label	0	0	0	0	0	0	0	0	0	0	...	1	1	1	1	1	1	1	1	1	1

1 rows × 3168 columns

In [9]:
```
# Удаляем последний столбец, содержащий словесное описание пола
df_except_label = df.drop('label', axis=1)
df_except_label.head()
```

Out[9]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	mode	centroid	meanfun	minfu
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	0.000000	0.059781	0.084279	0.01570
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	0.000000	0.066009	0.107937	0.01582
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	0.000000	0.077316	0.098706	0.01565
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0.727232	0.083878	0.151228	0.088965	0.01779
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0.783568	0.104261	0.135120	0.106398	0.01693

In [10]:
```
# Соединяем наши массивы обратно в один
df_handle = pd.concat([df_except_label, df_label_int], axis=1)
df_handle.head()
```

Out[10]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	ma
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	...	0.059781	0.084279	0.015702	0.27
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	...	0.066009	0.107937	0.015826	0.25
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	...	0.077316	0.098706	0.015656	0.27
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0.727232	...	0.151228	0.088965	0.017798	0.25
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0.783568	...	0.135120	0.106398	0.016931	0.26

5 rows × 21 columns

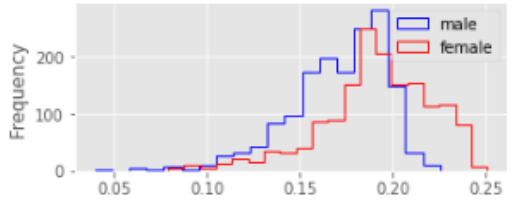
2.2. Visualization dataset

In [12]:
```
# print(plt.style.available)
plt.style.use('ggplot')
```

In [13]:
```
# Делим выборку на м/ж
mask = df_handle['label'] == 0
male = df_handle[mask]
female = df_handle[~mask]
```

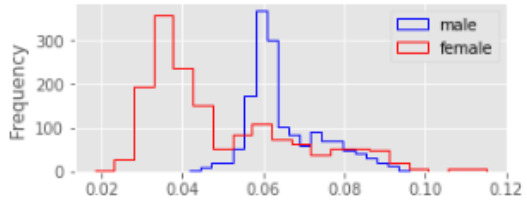
In [14]:
```
# Посмотрим на признаки и убедимся, что пропусков ни в одном из них нет – везде по 1584 записи
#male.info()
#print('\n')
#female.info()
```

In [15]:
```
# Большинство признаков выборки имеют слаборазличимую частотность по м/ж как признак meanfreq:
male['meanfreq'].plot.hist(bins=20, label='male', color='blue', histtype='step', linewidth=1, figsize=(5,2))
female['meanfreq'].plot.hist(bins=20, label='female', color='red', histtype='step', linewidth=1, figsize=(5,2))
plt.legend(); plt.draw()
```

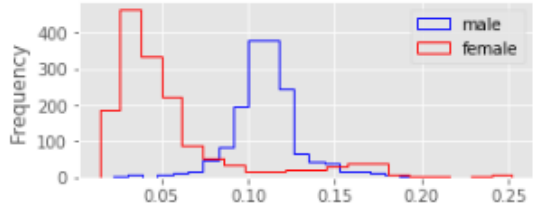


Наиболее различимые по м/ж признаки ниже:

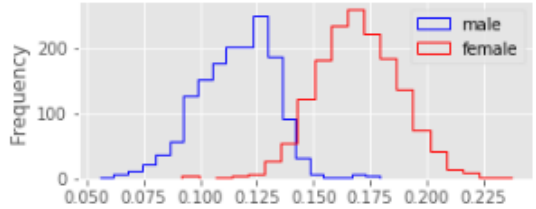
In [16]:
```
# 1. sd - standart deviation of frequency
male['sd'].plot.hist(bins=20, label='male', color='blue', histtype='step', linewidth=1, figsize=(5,2))
female['sd'].plot.hist(bins=20, label='female', color='red', histtype='step', linewidth=1, figsize=(5,2))
plt.legend(); plt.draw()
```



In [17]:
```
# 2. IQR - interquantile range
male['IQR'].plot.hist(bins=20, label='male', color='blue', histtype='step', linewidth=1, figsize=(5,2))
female['IQR'].plot.hist(bins=20, label='female', color='red', histtype='step', linewidth=1, figsize=(5,2))
plt.legend(); plt.draw()
```

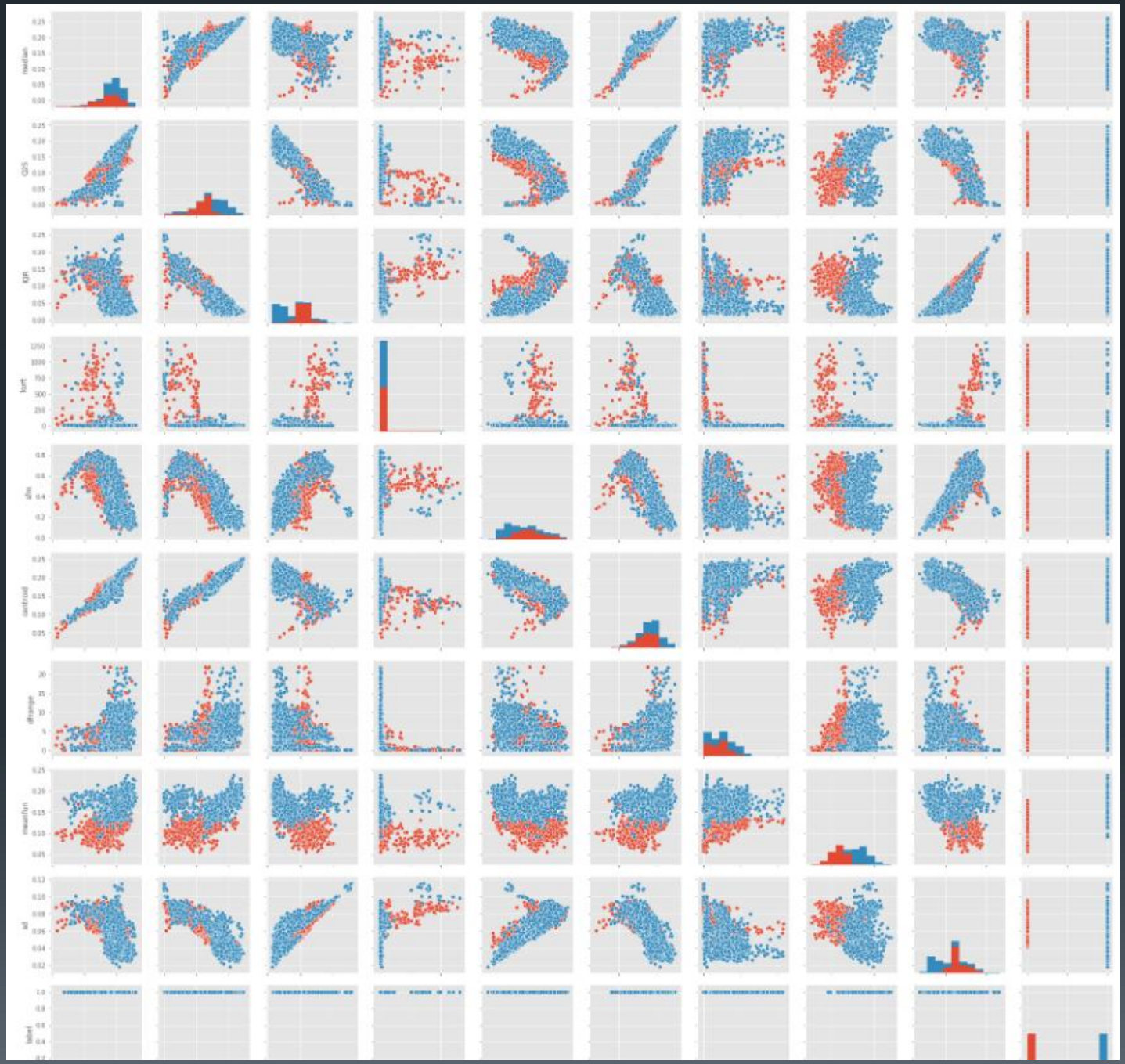


In [18]:
```
# 3. meanfun - average of fundamental frequency measured across accoustic signal
male['meanfun'].plot.hist(bins=20, label='male', color='blue', histtype='step', linewidth=1, figsize=(5,2))
female['meanfun'].plot.hist(bins=20, label='female', color='red', histtype='step', linewidth=1, figsize=(5,2))
plt.legend(); plt.draw()
```



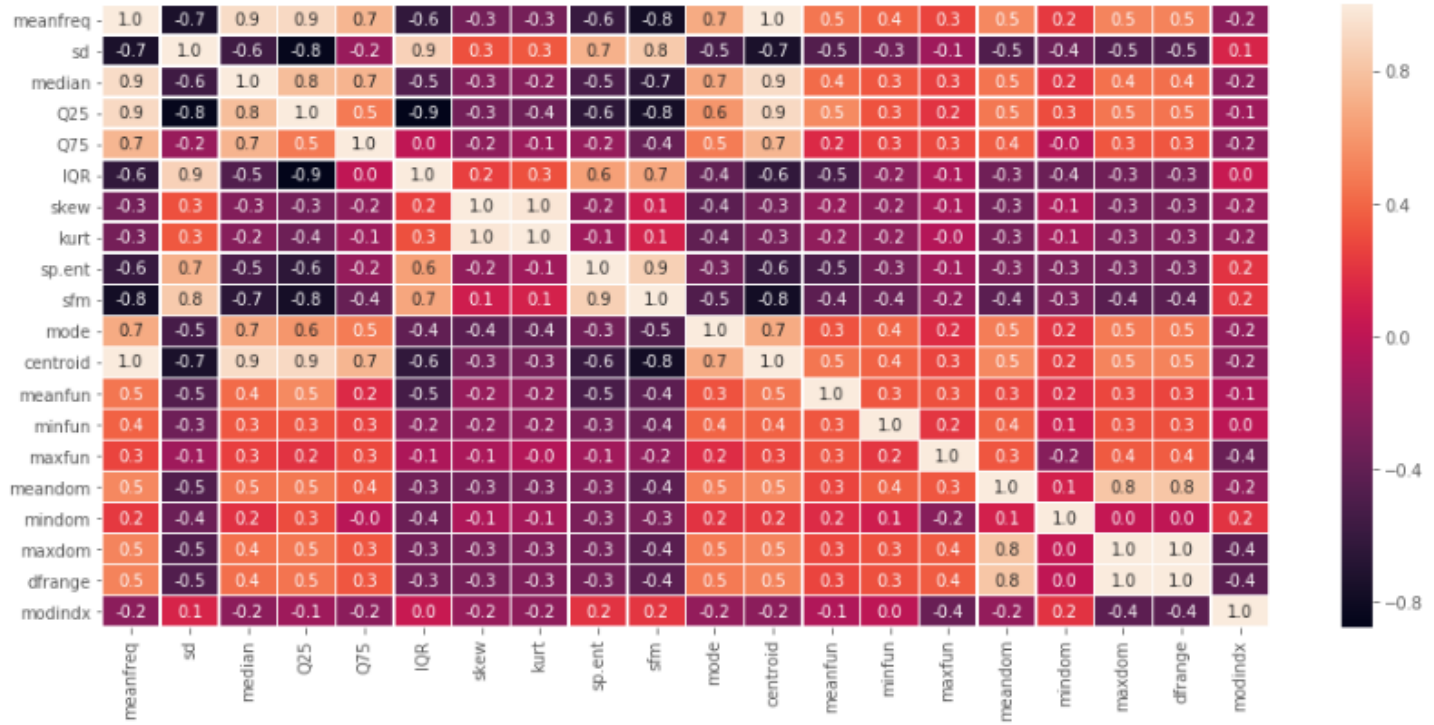
In [19]:
```
# Визуализация scatter plot matrix поможет посмотреть на одной картинке, как связаны между собой различные признаки.
# На диагонали матрицы графиков расположены гистограммы распределений признака (21 x 21)
# Остальные графики – это обычные scatter plots для соответствующих пар признаков
#sns.pairplot(df_handle, hue='label')
#plt.draw()
```

In [20]:
```
# Более компактно (по наиболее коррелируемым величинам, см. heatmap ниже):
sns.pairplot(df_handle[['median', 'Q25', 'IQR', 'kurt', 'sfm', 'centroid', 'dfrange', 'meanfun', 'sd', 'label']], hue='label')
plt.draw()
```

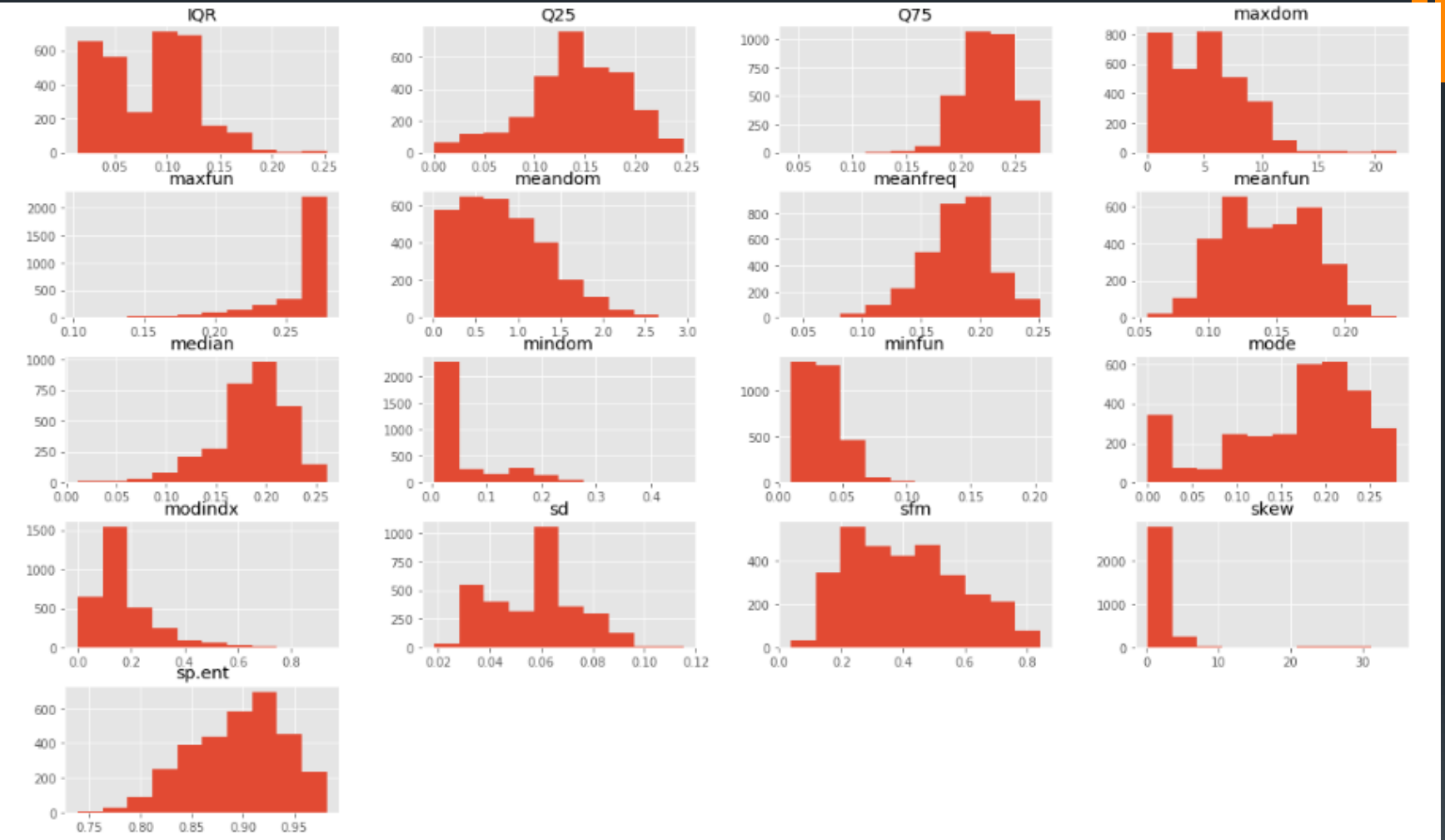

median
Q25
IQR
kurt
shn
centroid
dfrange
meanfun
sd
label

Посчитаем корреляцию количественных признаков

In [21]:
```
# Чем светлее (ближе к 1) коэффициент корреляции, тем больше зависимость между велечинами
# Коэффициент корреляции между количественными признаками по полу практически не различим (жен. чуть сильнее)
corr_matrix = df_handle.drop(['label'], axis=1).corr()
plt.figure(figsize = (16,7))
sns.heatmap(corr_matrix, annot=True, fmt=".1f", linewidths=.5); plt.draw()
```



In [22]:
```
# Наиболее сильно коррелируют между собой: skew/kurt; centroid/meanfreq; dfrange/maxdom
# Поэтому можно оставить для дальнейшего анализа только 3 признака из перечисленных выше с учетом бинарного признака label:
cor_features = list(set(df_handle.columns) - set(['kurt', 'centroid', 'dfrange', 'label']))
df_handle[cor_features].hist(figsize=(20, 12))
plt.draw()
```



[23]:
```
# Как было видно ранее из графиков, признаки maxfun, mindom, minfun, skew можно не рассматривать из-за их "выбросов"
# По нормальному закону распределены: meanfreq, meanfun, median, Q25, Q75, sp.ent. Остальные - ближе к Пуассоновскому.
# Т.к. по половому признаку более различны записи по признакам: meanfun/sd/IQR, - и IQR коррелирует с sd,
# а закон распределения больше похож на нормальный у признака meanfun, то ключевыми признаками будут: meanfun и sd.
# + kurt/skew, т.к. по нему более четко прослеживается граница (см. график sparse_matrix на seaborn, но неоднозначно)
```

3. Build model

Logistic regression

Перед нами данные, соответствующие задаче бинарной классификации. В этом случае прогноз получают с помощью формулы:

$$y = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b > 0$$

В качестве прогнозируемого значения задаем порог, равный нулю. Если значение функции меньше нуля, то мы прогнозируем класс -1, и, наоборот. Бинарнй линейный классификатор - это классификатор, который разделяет два класса с помощью линии, плоскости или гиперплоскости.

In [24]:
```
X = df_except_label

df.label=[1 if each =="female" else 0 for each in df.label]
y = df.label.values
```

In [25]:
```
# Посмотрим на минимальные и максимальные значения признаков
print("Min of features:\n", np.min(X))
print("\nMax of features:\n", np.max(X))
```

```
Min of features:
 meanfreq    0.039363
sd           0.018363
median       0.010975
Q25          0.000229
Q75          0.042946
IQR          0.014558
skew         0.141735
kurt         2.068455
sp.ent       0.738651
sfm          0.036876
mode         0.000000
centroid     0.039363
meanfun      0.055565
minfun       0.009775
maxfun       0.103093
meandom      0.007812
mindom       0.004883
maxdom       0.007812
dfrange      0.000000
modindx      0.000000
dtype: float64

Max of features:
 meanfreq       0.251124
sd             0.115273
median         0.261224
Q25            0.247347
Q75            0.273469
IQR            0.252225
skew          34.725453
kurt        1309.612887
sp.ent         0.981997
sfm            0.842936
```

In [26]:
```
# Из-за большого разброса по некоторым признакам, нормализуем признаки
X_norm = (X - np.min(X)) / (np.max(X) - np.min(X)).values
```

In [27]:
```
X_norm.shape
```

Out[27]:
```
(3168, 20)
```

In [28]:
```
# Разобьем матрицу признаков и целевой вектор на обучающую и тестовую выборки
x_train, x_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.15, random_state=42)
```

In [29]:
```
# Транспонируем матрицы/вектора
x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T
```

In [30]:
```
print("x_train shape:",x_train.shape)
print("x_test shape:",x_test.shape)
print("y_train shape:",y_train.shape)
print("y_test shape:",y_test.shape)
```

```
x_train shape: (20, 2692)
x_test shape: (20, 476)
y_train shape: (2692,)
y_test shape: (476,)
```

In [31]:
```
# Инициализируем параметры функции, описанной выше и зададим сигмоиную функцию для сглаживания прогнозируемой величины
# (логистическое уравнение или уравнение Ферхюльста)
def initialize_weights_and_bias(dimension):
    w = np.full((dimension, 1), 0.01)
    b = 0.0
    return w, b
def sigmoid(z):
    y_head = 1/(1 + np.exp(-z))
    return y_head
```

In [32]:
```
# Создадим методы прямого/обратного распространения ошибки (функцию потерь)
def forward_backward_propagation(w, b, x_train, y_train):
    # forward propagation
    z = np.dot(w.T, x_train) + b
    y_head = sigmoid(z)
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    cost = (np.sum(loss)) / x_train.shape[1]
    # x_train.shape[1]  is for scaling

    # backward propagation
    derivative_weight = (np.dot(x_train,((y_head - y_train).T))) / x_train.shape[1]
    derivative_bias = np.sum(y_head - y_train) / x_train.shape[1]
    gradients = {"derivative_weight": derivative_weight, "derivative_bias": derivative_bias}

    return cost, gradients
```

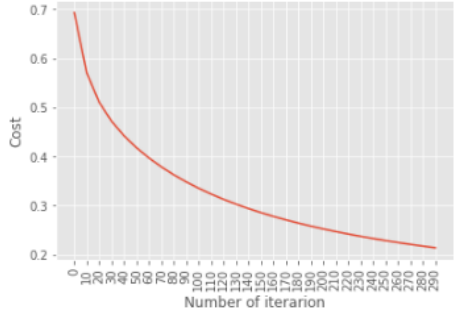
In [33]:
```
# Обновим обучающие параметры модели
def update(w, b, x_train, y_train, learning_rate, number_of_iterarion):
    cost_list = []
    cost_list2 = []
    index = []

    for i in range(number_of_iterarion):
        # Проводим расчет forward-backward propagation и находим cost и gradients
        cost, gradients = forward_backward_propagation(w, b, x_train, y_train)
        cost_list.append(cost)
        # Обновялем веса (w) и свободный коэффициент (b)
        w = w - learning_rate * gradients["derivative_weight"]
        b = b - learning_rate * gradients["derivative_bias"]
        if i % 10 == 0:
            cost_list2.append(cost)
            index.append(i)
            print("Cost after iteration %i: %f" % (i, cost))

    # Обновляем параметры модели (w, b)
    parameters = {"weight": w, "bias": b}
    plt.plot(index, cost_list2)
    plt.xticks(index, rotation='vertical')
    plt.xlabel("Number of iterarion")
    plt.ylabel("Cost")
    plt.show()
    return parameters, gradients, cost_list
```

In [34]:
```
# Проставляем метки класса, в зависимости от принимаемого значения сигмоидной функции
def predict(w, b, x_test):
    z = sigmoid(np.dot(w.T, x_test) + b)
    Y_prediction = np.zeros((1, x_test.shape[1]))
    # если z больше 0.5, то прогнозируем класс 1 (y_head = 1), иначе ноль (y_head = 0)
    for i in range(z.shape[1]):
        if z[0, i]<= 0.5:
            Y_prediction[0, i] = 0
        else:
            Y_prediction[0, i] = 1

    return Y_prediction
```

In [35]:
```
# Определим функцию логистической регрессии и проверим её на learning_rate = 1, num_iterations = 300
def logistic_regression(x_train, y_train, x_test, y_test, learning_rate, num_iterations):
    dimension =  x_train.shape[0]  # равно 20, согласно размерности вектора признаков
    w, b = initialize_weights_and_bias(dimension)
    parameters, gradients, cost_list = update(w, b, x_train, y_train, learning_rate, num_iterations)

    y_prediction_test = predict(parameters["weight"], parameters["bias"], x_test)

    print("test accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_test - y_test)) * 100))

logistic_regression(x_train, y_train, x_test, y_test, learning_rate = 1, num_iterations = 300)
```

```
logistic_regression(x_train, y_train, x_test, y_test, learning_rate = 1, num_iterations = 300)
```

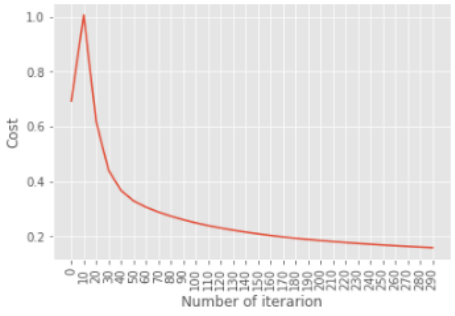
```
Cost after iteration 0: 0.692736
Cost after iteration 10: 0.570231
Cost after iteration 20: 0.510444
Cost after iteration 30: 0.471619
Cost after iteration 40: 0.442082
Cost after iteration 50: 0.417778
Cost after iteration 60: 0.396971
Cost after iteration 70: 0.378769
Cost after iteration 80: 0.362633
Cost after iteration 90: 0.348199
Cost after iteration 100: 0.335201
Cost after iteration 110: 0.323428
Cost after iteration 120: 0.312715
Cost after iteration 130: 0.302923
Cost after iteration 140: 0.293940
Cost after iteration 150: 0.285669
Cost after iteration 160: 0.278028
Cost after iteration 170: 0.270949
Cost after iteration 180: 0.264371
Cost after iteration 190: 0.258244
Cost after iteration 200: 0.252523
Cost after iteration 210: 0.247169
Cost after iteration 220: 0.242147
Cost after iteration 230: 0.237429
Cost after iteration 240: 0.232986
Cost after iteration 250: 0.228797
Cost after iteration 260: 0.224839
Cost after iteration 270: 0.221095
Cost after iteration 280: 0.217548
Cost after iteration 290: 0.214182
```




```
test accuracy: 96.84873949579831 %
```

```
In [37]: logistic_regression(x_train, y_train, x_test, y_test, learning_rate = 2, num_iterations = 300)
```

```
Cost after iteration 0: 0.692736
Cost after iteration 10: 1.005434
Cost after iteration 20: 0.616500
Cost after iteration 30: 0.439932
Cost after iteration 40: 0.366578
Cost after iteration 50: 0.329188
Cost after iteration 60: 0.306164
Cost after iteration 70: 0.288343
Cost after iteration 80: 0.273280
Cost after iteration 90: 0.260302
Cost after iteration 100: 0.248999
Cost after iteration 110: 0.239065
Cost after iteration 120: 0.230266
Cost after iteration 130: 0.222421
Cost after iteration 140: 0.215383
Cost after iteration 150: 0.209034
Cost after iteration 160: 0.203280
Cost after iteration 170: 0.198041
Cost after iteration 180: 0.193252
Cost after iteration 190: 0.188857
Cost after iteration 200: 0.184811
Cost after iteration 210: 0.181073
Cost after iteration 220: 0.177610
Cost after iteration 230: 0.174393
Cost after iteration 240: 0.171397
Cost after iteration 250: 0.168600
Cost after iteration 260: 0.165982
Cost after iteration 270: 0.163528
Cost after iteration 280: 0.161222
Cost after iteration 290: 0.159052
```




```
test accuracy: 97.6890756302521 %
```

Варьируя различными параметрами скорости обучения (learning_rate) и количеством итераций расчетов (num_iterations), можно заключчить, что оптимальная предсказательная точность модели (accuracy), равная 97,6890756302521 % достигается при learning_rate = 2, num_iterations = 300

In [38]:
```
# Попробуем решить задачу с помощью sklearn:
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train.T, y_train.T)
print("Test Accuracy : {}".format(lr.score(x_test.T, y_test.T)))
```

```
Test Accuracy : 0.976890756302521
```

Решение классификатора LogisticRegression с дефолтными настройками дает тот же результат, что и в предыдущем ручном расчете