Lab 3 - Linear Methods in Classification_Problem_Sai

March 5, 2022

1 Two Problems for Lab 3:

2 Problem 1: Gender Recognition by Voice

From the description file at https://data.world/ml-research/gender-recognition-by-voice:

In order to analyze gender by voice and speech, a training database was required. A database was built using thousands of samples of male and female voices, each labeled by their gender of male or female. Voice samples were collected from the following resources:

- The Harvard-Haskins Database of Regularly-Timed Speech
- Telecommunications & Signal Processing Laboratory (TSP) Speech Database at McGill University
- VoxForge Speech Corpus
- Festvox CMU_ARCTIC Speech Database at Carnegie Mellon University

Each voice sample is stored as a .WAV file, which is then pre-processed for acoustic analysis using the specan function from the WarbleR R package. Specan measures 22 acoustic parameters on acoustic signals for which the start and end times are provided.

The output from the pre-processed WAV files were saved into a CSV file, containing 3168 rows and 21 columns (20 columns for each feature and one label column for the classification of male or female). You can download the pre-processed dataset in CSV format, using the link above Acoustic Properties Measured

The following acoustic properties of each voice are measured:

- **duration:** length of signal
- **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

The gender of the speaker is given in the **label** column.

Note, the features for duration and peak frequency (peakf) were removed from training. Duration refers to the length of the recording, which for training, is cut off at 20 seconds. Peakf was omitted from calculation due to time and CPU constraints in calculating the value. In this case, all records will have the same value for duration (20) and peak frequency (0).

Load file using the code below.

Question 1: Which two features are most indicative of gendered voice?

Question 2: Preform Linear Regression, Logistic Regression, and Quadratic Discriminant Analysis on the features, graphing the resulting fits. How does the two feature fit compare to the fit on all features?

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import io
import requests

data = pd.read_csv("/Users/sakshisuman12/Documents/NU_Coursework/MATH7243/labs/
↪3/Archive/voice.csv")

data.head()
```

[2]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	\
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	

	kurt	sp.ent	sfm	…	centroid	meanfun	minfun	\
0	274.402906	0.893369	0.491918	…	0.059781	0.084279	0.015702	
1	634.613855	0.892193	0.513724	…	0.066009	0.107937	0.015826	
2	1024.927705	0.846389	0.478905	…	0.077316	0.098706	0.015656	

```
3     4.177296  0.963322  0.727232  …  0.151228  0.088965  0.017798
4     4.333713  0.971955  0.783568  …  0.135120  0.106398  0.016931

     maxfun   meandom    mindom    maxdom   dfrange   modindx  label
0  0.275862  0.007812  0.007812  0.007812  0.000000  0.000000   male
1  0.250000  0.009014  0.007812  0.054688  0.046875  0.052632   male
2  0.271186  0.007990  0.007812  0.015625  0.007812  0.046512   male
3  0.250000  0.201497  0.007812  0.562500  0.554688  0.247119   male
4  0.266667  0.712812  0.007812  5.484375  5.476562  0.208274   male

[5 rows x 21 columns]
```

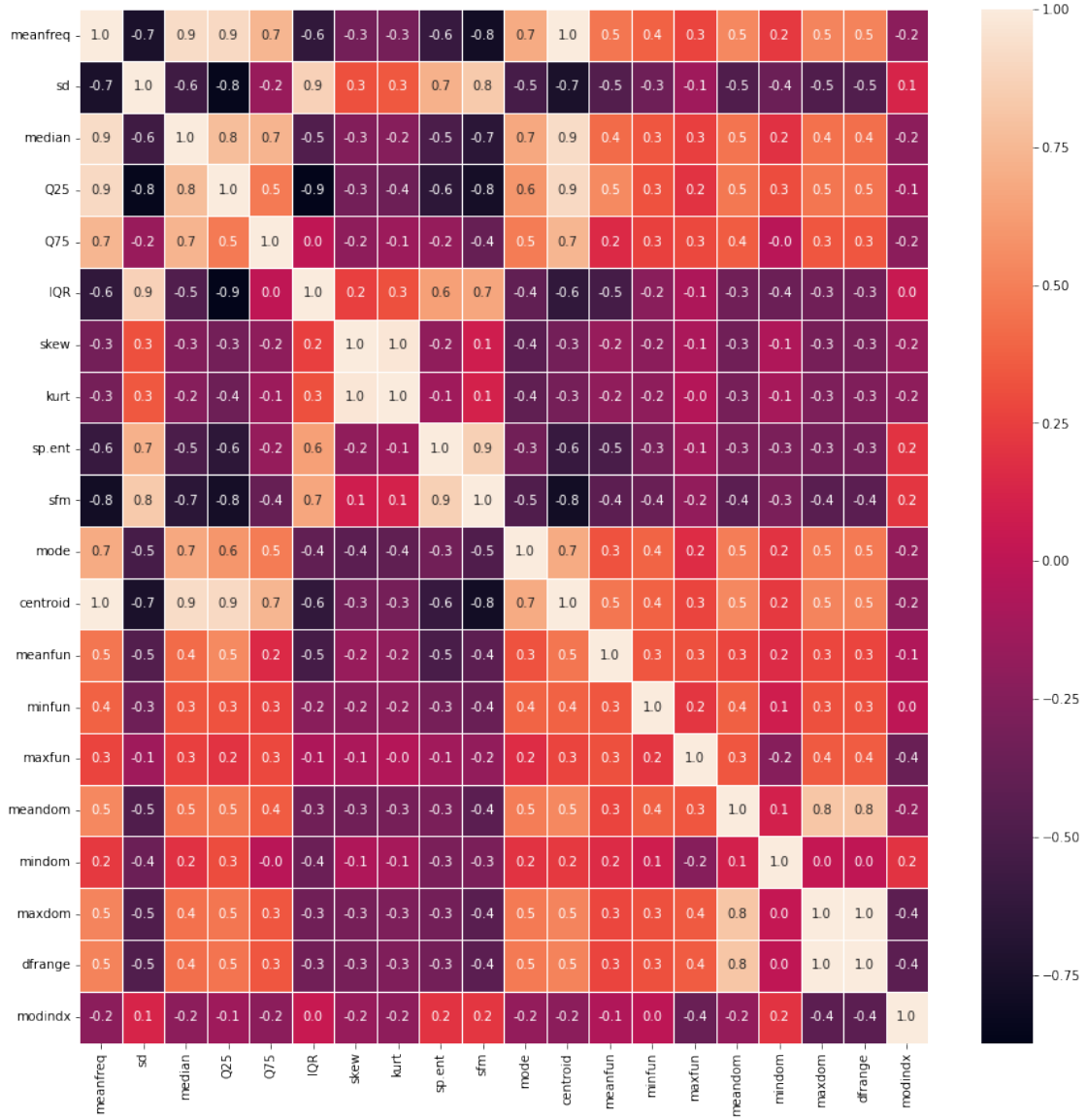
[3]:
```
## Drop Feature Columns X
X = data.drop(columns=["label"])

## Set Up Target Variables y
y = data["label"]

## Normalize feature data by centering on the mean and dividing by std
X = (X - X.mean())/X.std()
```

[4]:
```
sex = data['label'].replace(['male','female'],[0,1])
y = sex
print(y)
```

```
0       0
1       0
2       0
3       0
4       0
       ..
3163    1
3164    1
3165    1
3166    1
3167    1
Name: label, Length: 3168, dtype: int64
```

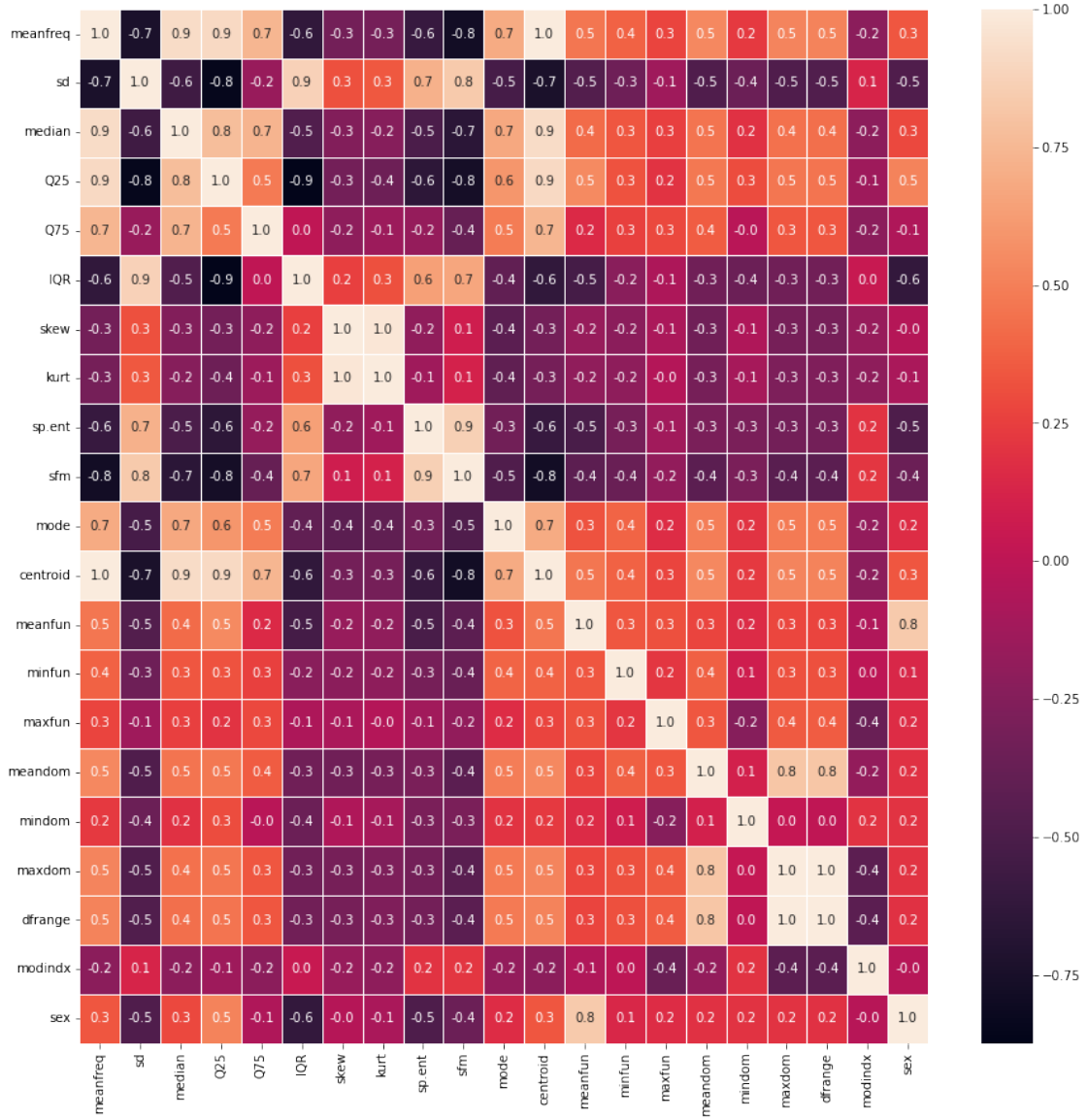
[5]:
```
f,ax = plt.subplots(figsize=(15, 15))
sns.heatmap(data.corr(),annot=True,linewidth=.5, fmt='.1f')
```

[5]:
```
<AxesSubplot:>
```



[6]:
```
data_y = data.assign(sex=y)
f,ax = plt.subplots(figsize=(15, 15))
sns.heatmap(data_y.corr(),annot=True,linewidth=.5, fmt='.1f')
```

[6]: <AxesSubplot:>

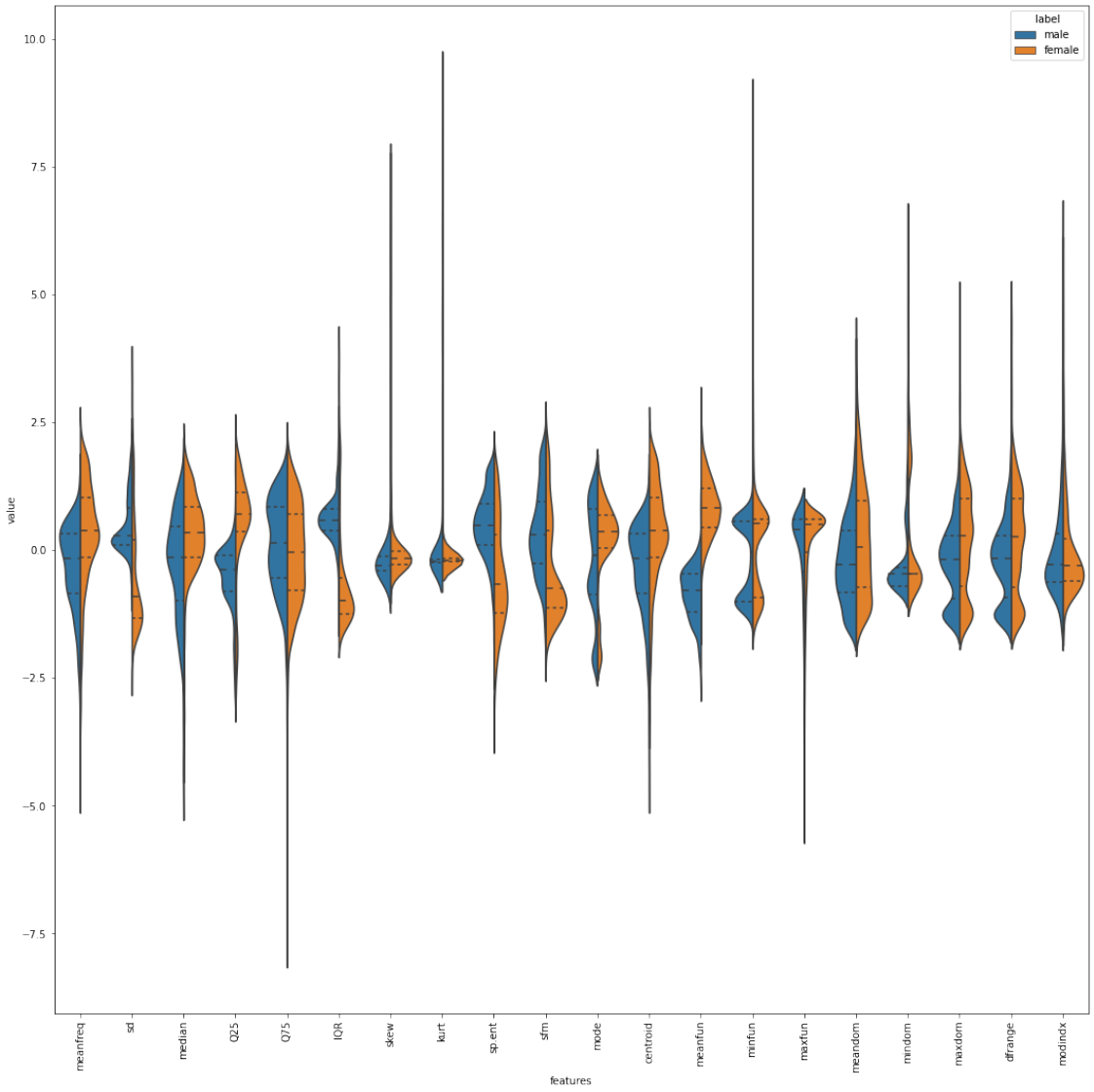


Here are a few plots that can be used for data analysis

[41]:
```
# plots
plt.figure(figsize=(18,18))

vio = pd.concat([y,X],axis=1)
vio = pd.melt(vio,id_vars="label",
                    var_name="features",
                    value_name='value')

sns.violinplot(x="features", y="value", hue="label", data=vio,split=True,␣
↪inner="quart")
```

```
plt.xticks(rotation=90)
```

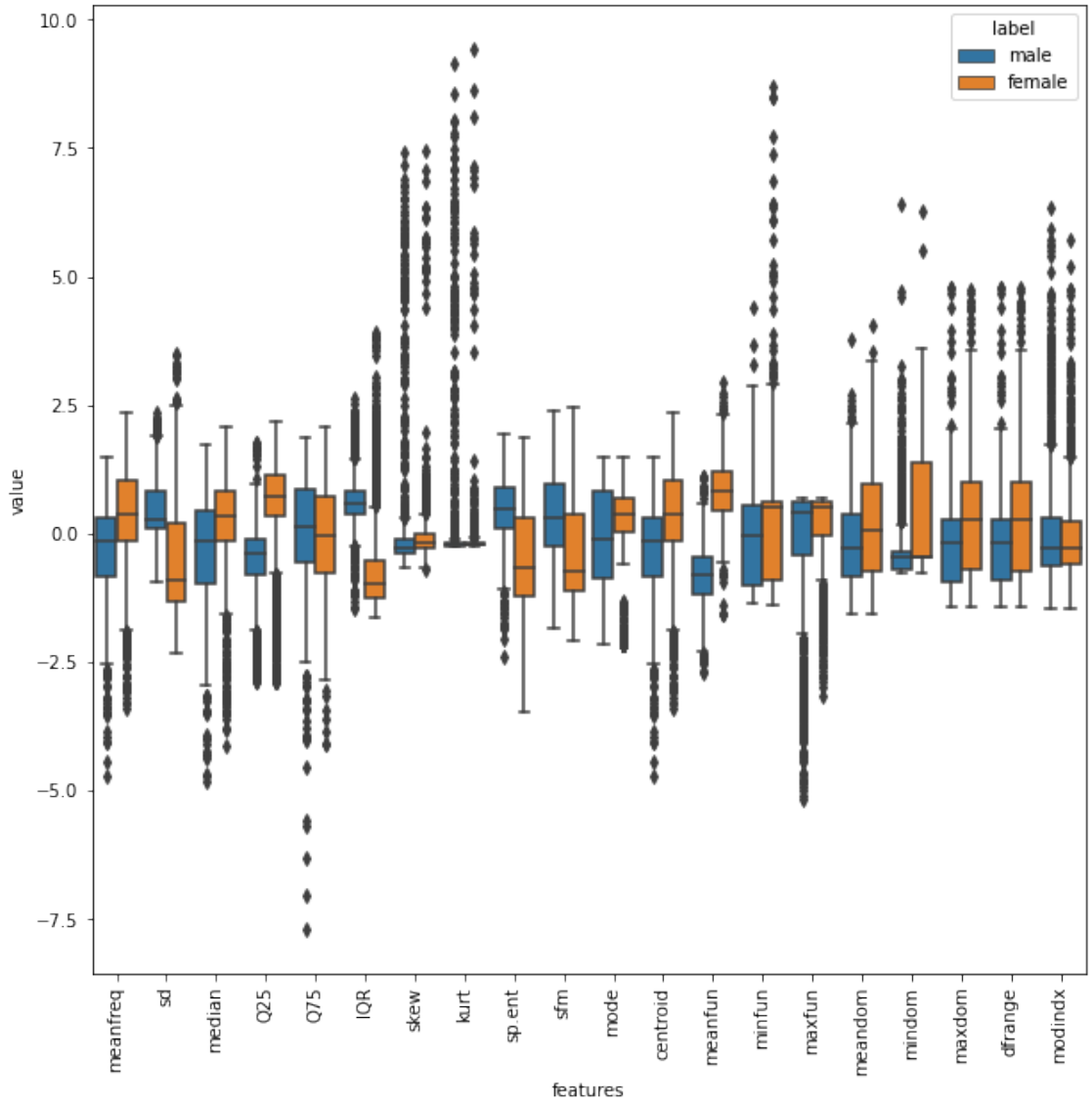
[41]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19]),
 [Text(0, 0, 'meanfreq'),
  Text(1, 0, 'sd'),
  Text(2, 0, 'median'),
  Text(3, 0, 'Q25'),
  Text(4, 0, 'Q75'),
  Text(5, 0, 'IQR'),
  Text(6, 0, 'skew'),
  Text(7, 0, 'kurt'),
  Text(8, 0, 'sp.ent'),
  Text(9, 0, 'sfm'),
  Text(10, 0, 'mode'),
  Text(11, 0, 'centroid'),
  Text(12, 0, 'meanfun'),
  Text(13, 0, 'minfun'),
  Text(14, 0, 'maxfun'),
  Text(15, 0, 'meandom'),
  Text(16, 0, 'mindom'),
  Text(17, 0, 'maxdom'),
  Text(18, 0, 'dfrange'),
  Text(19, 0, 'modindx')])
```



[42]:
```
plt.figure(figsize=(10,10))

sns.boxplot(x="features", y="value", hue="label", data=vio)

plt.xticks(rotation=90)
```

[42]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19]),
 [Text(0, 0, 'meanfreq'),
  Text(1, 0, 'sd'),
  Text(2, 0, 'median'),
  Text(3, 0, 'Q25'),
  Text(4, 0, 'Q75'),
```

```
Text(5, 0, 'IQR'),
Text(6, 0, 'skew'),
Text(7, 0, 'kurt'),
Text(8, 0, 'sp.ent'),
Text(9, 0, 'sfm'),
Text(10, 0, 'mode'),
Text(11, 0, 'centroid'),
Text(12, 0, 'meanfun'),
Text(13, 0, 'minfun'),
Text(14, 0, 'maxfun'),
Text(15, 0, 'meandom'),
Text(16, 0, 'mindom'),
Text(17, 0, 'maxdom'),
Text(18, 0, 'dfrange'),
Text(19, 0, 'modindx')])
```

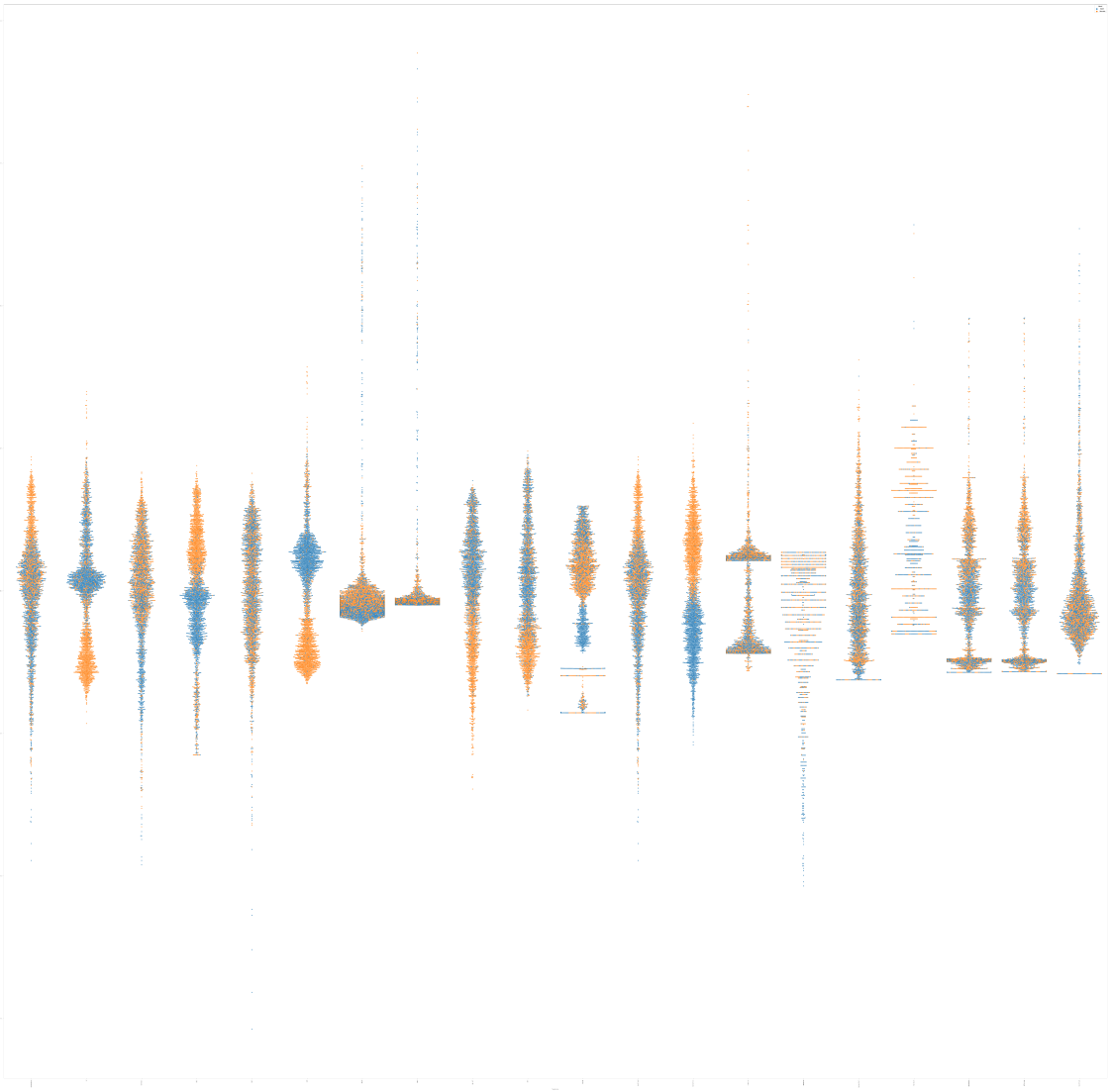


[43]:
```
plt.figure(figsize=(120,120))
sns.swarmplot(x="features", y="value", hue="label", data=vio)
plt.xticks(rotation=90)
```

```
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/seaborn/categorical.py:1296: UserWarning: 21.1% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/seaborn/categorical.py:1296: UserWarning: 69.8% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/seaborn/categorical.py:1296: UserWarning: 6.6% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/seaborn/categorical.py:1296: UserWarning: 30.1% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/seaborn/categorical.py:1296: UserWarning: 55.3% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/seaborn/categorical.py:1296: UserWarning: 64.3% of the points cannot be
placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

[43]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19]),
 [Text(0, 0, 'meanfreq'),
  Text(1, 0, 'sd'),
  Text(2, 0, 'median'),
  Text(3, 0, 'Q25'),
  Text(4, 0, 'Q75'),
  Text(5, 0, 'IQR'),
  Text(6, 0, 'skew'),
  Text(7, 0, 'kurt'),
  Text(8, 0, 'sp.ent'),
  Text(9, 0, 'sfm'),
  Text(10, 0, 'mode'),
  Text(11, 0, 'centroid'),
  Text(12, 0, 'meanfun'),
  Text(13, 0, 'minfun'),
```

```
Text(14, 0, 'maxfun'),
Text(15, 0, 'meandom'),
Text(16, 0, 'mindom'),
Text(17, 0, 'maxdom'),
Text(18, 0, 'dfrange'),
Text(19, 0, 'modindx')])
```

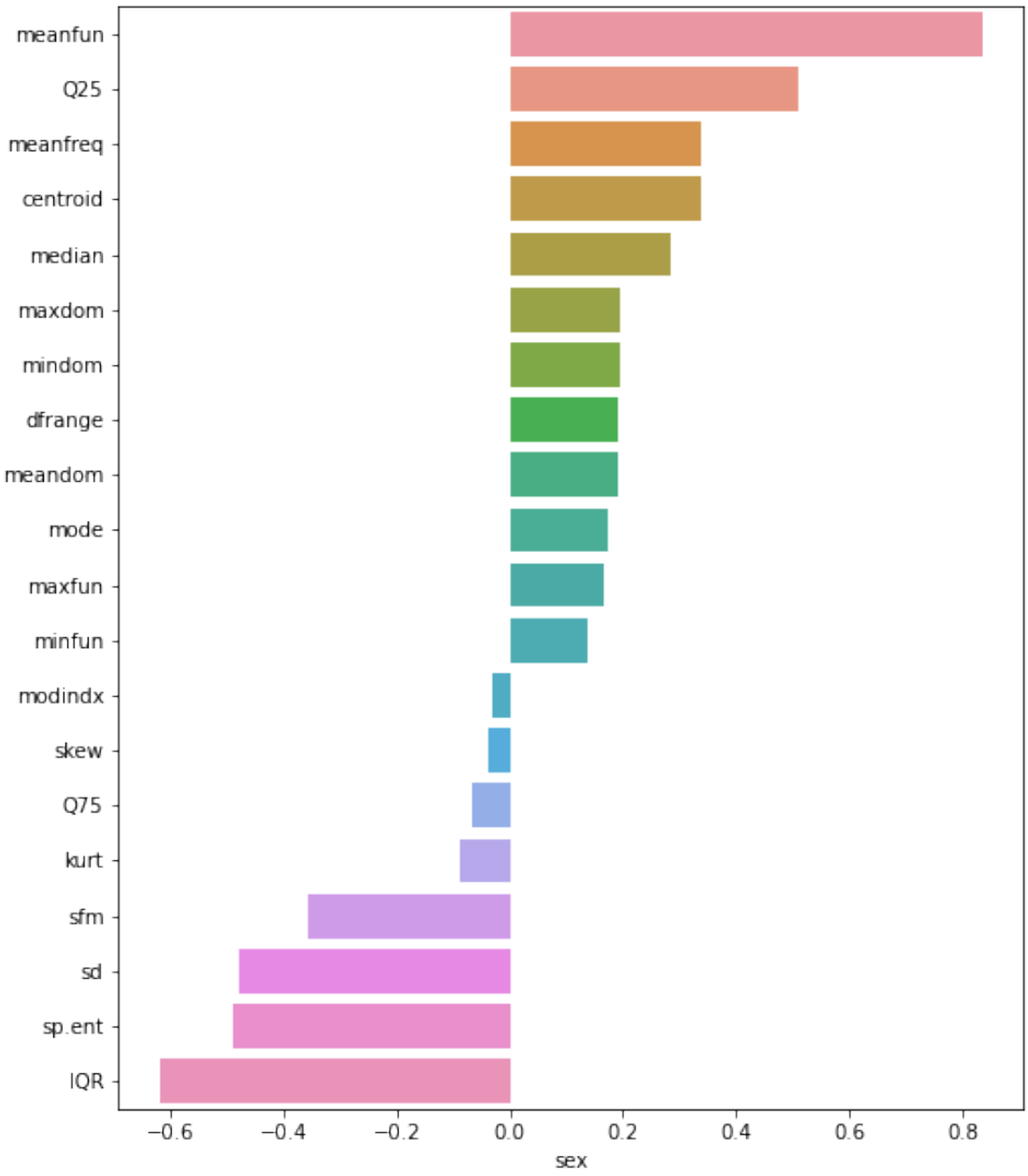


[8]:
```python
def get_feature_groups():
    """ Returns a list of numerical and categorical features,
    excluding sex. """

    # Numerical Features
```

```
    nums = data_y.select_dtypes(include=['int64','float64']).columns
    nums = nums.drop(['sex']) # drop sex

    return list(nums)

nums = get_feature_groups()
```

[9]:
```
corr = data_y[['sex'] + nums].corr()

corr = corr.sort_values('sex', ascending=False)
plt.figure(figsize=(8,10))
sns.barplot( corr.sex[1:], corr.index[1:], orient='h')
plt.show()
```

```
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables
as keyword args: x, y. From version 0.12, the only valid positional argument
will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```



[11]:
```
corr.sex
```

[11]:
```
sex         1.000000
meanfun     0.833921
Q25         0.511455
meanfreq    0.337415
centroid    0.337415
median      0.283919
maxdom      0.195657
```

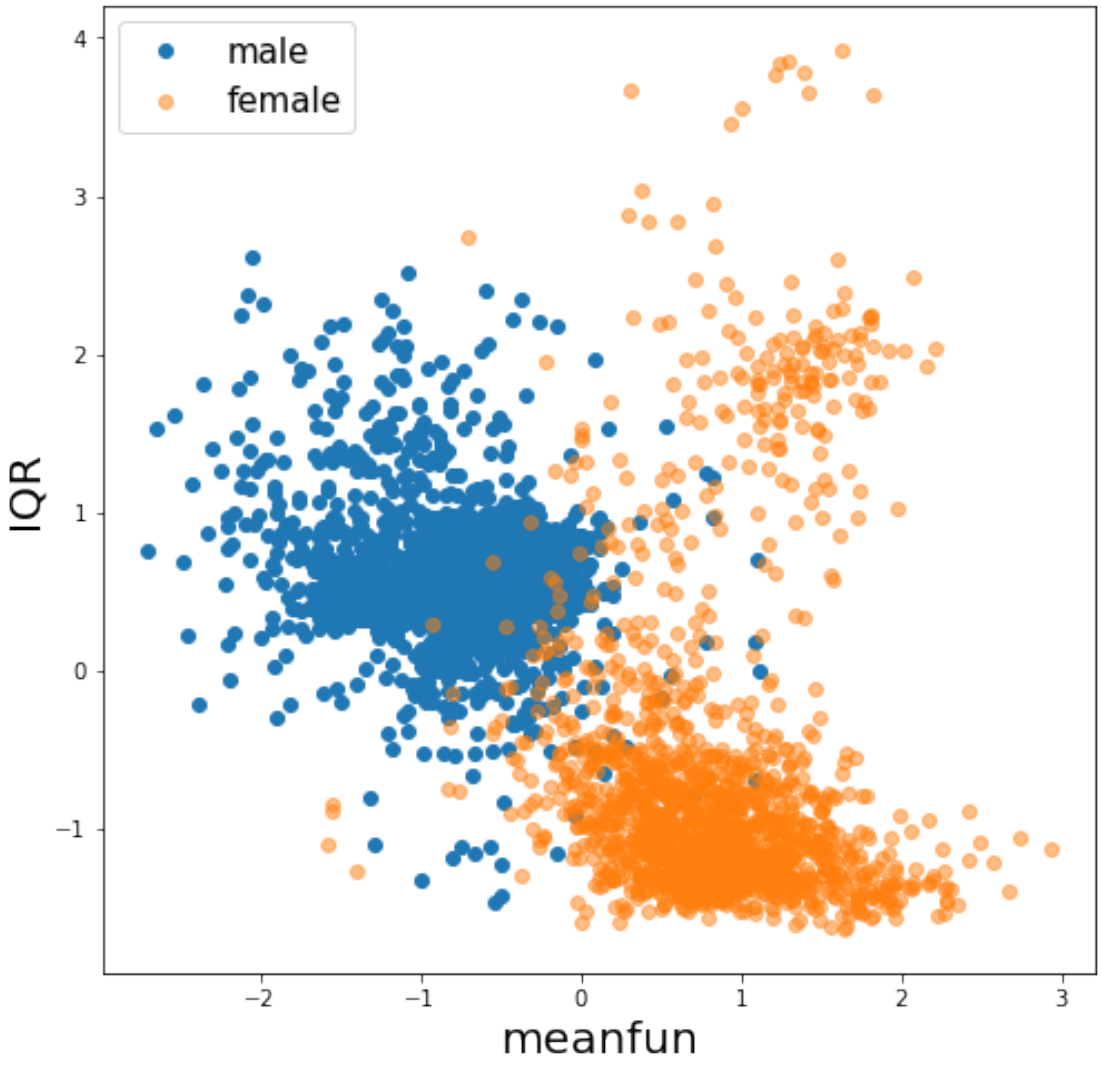
```
mindom      0.194974
dfrange     0.192213
meandom     0.191067
mode        0.171775
maxfun      0.166461
minfun      0.136692
modindx    -0.030801
skew       -0.036627
Q75        -0.066906
kurt       -0.087195
sfm        -0.357499
sd         -0.479539
sp.ent     -0.490552
IQR        -0.618916
Name: sex, dtype: float64
```

2.0.1 Two features that have the highest correlation w.r.t sex are "meanfun" & "IQR"

[12]:
```
from sklearn.linear_model import LinearRegression
# Applying Linear Regression, Logistic Regression and QDA
# Two features with highest correlation

## Drop Feature Columns X
X = data.drop(columns=["label"])

## Set Up Target Variables y
y = data["label"]

## Normalize feature data by centering on the mean and dividing by std
X = (X - X.mean())/X.std()
```

[13]:
```
f, ax = plt.subplots(figsize=(8,8))

I_m = y=="male"
I_f = y=="female"

plt.plot(X["meanfun"][I_m],X["IQR"][I_m],'o',label="male")

## We set alpha=.5 to try to avoid masking, but some points still will be␣
 ↪buried.
plt.plot(X["meanfun"][I_f],X["IQR"][I_f],'o',label="female",alpha=.5)

plt.xlabel("meanfun",fontsize=20)
plt.ylabel("IQR",fontsize=20)
plt.legend(fontsize=15)
```

[13]:
```
<matplotlib.legend.Legend at 0x7fc150e96df0>
```



[14]:
```
y_train = pd.get_dummies(y)
display(y_train)
```

	female	male
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1
…	…	…
3163	1	0
3164	1	0
3165	1	0
3166	1	0

```
3167       1     0

[3168 rows x 2 columns]
```

[15]:
```
# Linear Regression

X_train = X[['meanfun',"IQR"]]

lr = LinearRegression()
lr.fit(X_train,y_train)

print("The r^2 score on the training data is %.3f"%(lr.score(X_train,y_train),))
```

```
The r^2 score on the training data is 0.737
```

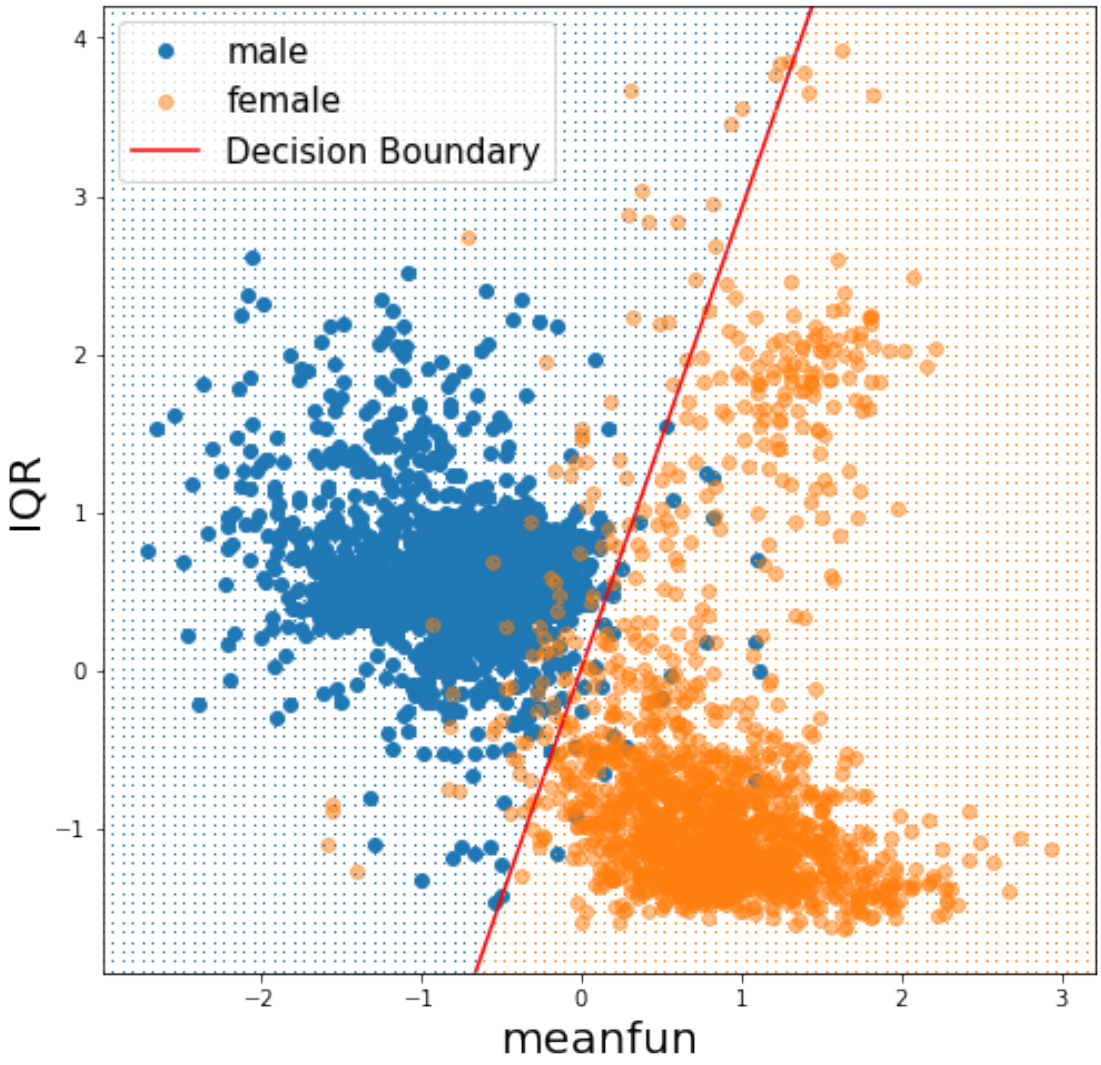
[16]:
```
beta_0 = lr.intercept_
beta_c = lr.coef_

print("The Linear Coefficients:\n", beta_c)
print("The Intercept:", beta_0)
```

```
The Linear Coefficients:
 [[ 0.35221767 -0.12125969]
 [-0.35221767  0.12125969]]
The Intercept: [0.5 0.5]
```

[17]:
```
f, ax = plt.subplots(figsize=(8,8))

X1 = X["meanfun"]
X2 = X["IQR"]

plt.plot(X1[I_m],X2[I_m],'o',label="male")
plt.plot(X1[I_f],X2[I_f],'o',label="female",alpha=.5)


## We want to make a nice clean line directly across the graph as it was before
## The best way to do this is to find the limits of the graph and plot using␣
 ↪them

xm,xM = plt.xlim()
ym,yM = plt.ylim()

u = np.linspace(xm,xM, 2)
v = (u*(beta_c[0,0]-beta_c[1,0]) + beta_0[0]-beta_0[1])/
 ↪(beta_c[1,1]-beta_c[0,1])
plt.plot(u,v,label="Decision Boundary",color="red")
```

```
## We also may also want to color in the side of the decicion boundry we're
## Labeling each point. One way to do this is using a mesh grid, and then using
## an indexon the equation from before

XX, YY = np.meshgrid(np.linspace(xm,xM, 100),np.linspace(ym,yM, 100))

z1 = YY >(XX*(beta_c[0,0]-beta_c[1,0]) + beta_0[0]-beta_0[1])/
 ↪(beta_c[1,1]-beta_c[0,1])
z2 = YY <(XX*(beta_c[0,0]-beta_c[1,0]) + beta_0[0]-beta_0[1])/
 ↪(beta_c[1,1]-beta_c[0,1])

plt.plot(XX[z1],YY[z1],',',color="C0")
plt.plot(XX[z2],YY[z2],',',color="C1")

## We now reset the x and y limits to make sure our view is centered tightly
## around the data.

plt.xlabel("meanfun",fontsize=20)
plt.ylabel("IQR",fontsize=20)
plt.legend(fontsize=15)

ax.set_xlim([xm, xM])
ax.set_ylim([ym, yM])
```

[17]: (-1.9081155923982005, 4.2025798818885)



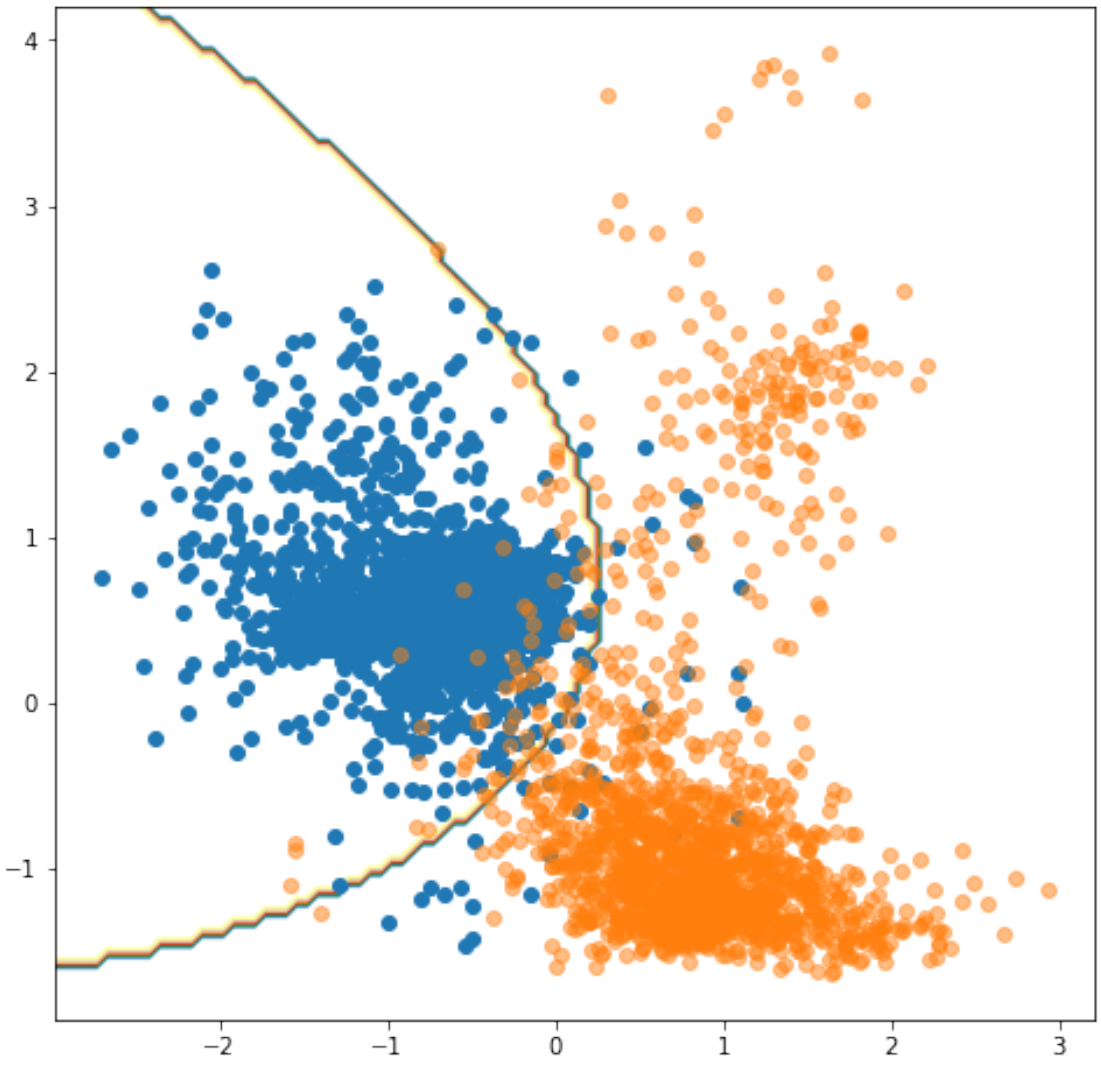
[18]:
```
# QDA

from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

qda = QuadraticDiscriminantAnalysis(store_covariance=True)
qda.fit(X_train, y)
print("Score: %.3f"%qda.score(X_train,y))
```

```
Score: 0.966
```

[27]:
```
f, ax = plt.subplots(figsize=(8,8))

X1 = X["meanfun"]
X2 = X["IQR"]
```

```
plt.plot(X1[I_m],X2[I_m],'o',label="male")
plt.plot(X1[I_f],X2[I_f],'o',label="female",alpha=.5)

## As before we generate a meshgrid, but now we use qda.predict to guess at the␣
 ↪label.

xm,xM = plt.xlim()
ym,yM = plt.ylim()

XX, YY = np.meshgrid(np.linspace(xm,xM, 100),np.linspace(ym,yM, 100))

## We now form a 10000x2 array of the (x,y) coordiantes for each point by␣
 ↪reshaping
## the XX and YY matricies and pasting them together. We need to feed a Nx2␣
 ↪vector
## into the qda.predict function, otherwise it will think we have too many␣
 ↪features.
## We can reshape it later to get our grid back

grid=np.concatenate([XX.reshape(-1,1),YY.reshape(-1,1)],axis=1)

ZZ = qda.predict(grid).reshape(XX.shape)  ## We predict, and reshape back to␣
 ↪the origional grid

z1 = ZZ == 'male'
z2 = ZZ == 'female'

plt.contour(XX, YY, z1, cmap=plt.cm.Paired)
```

[27]: <matplotlib.contour.QuadContourSet at 0x7fc1e205afd0>



[28]:
```
# Logistic Regression

from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X_train,y)

print("Score: %.3f"%clf.score(X_train,y))
```

```
Score: 0.966
```

[29]:
```
X_train = X[['meanfun',"IQR"]]

lg = LogisticRegression()
```

```
lg.fit(X_train,y)

print("The r^2 score on the training data is %.3f"%(lg.score(X_train,y),))
```

```
The r^2 score on the training data is 0.966
```

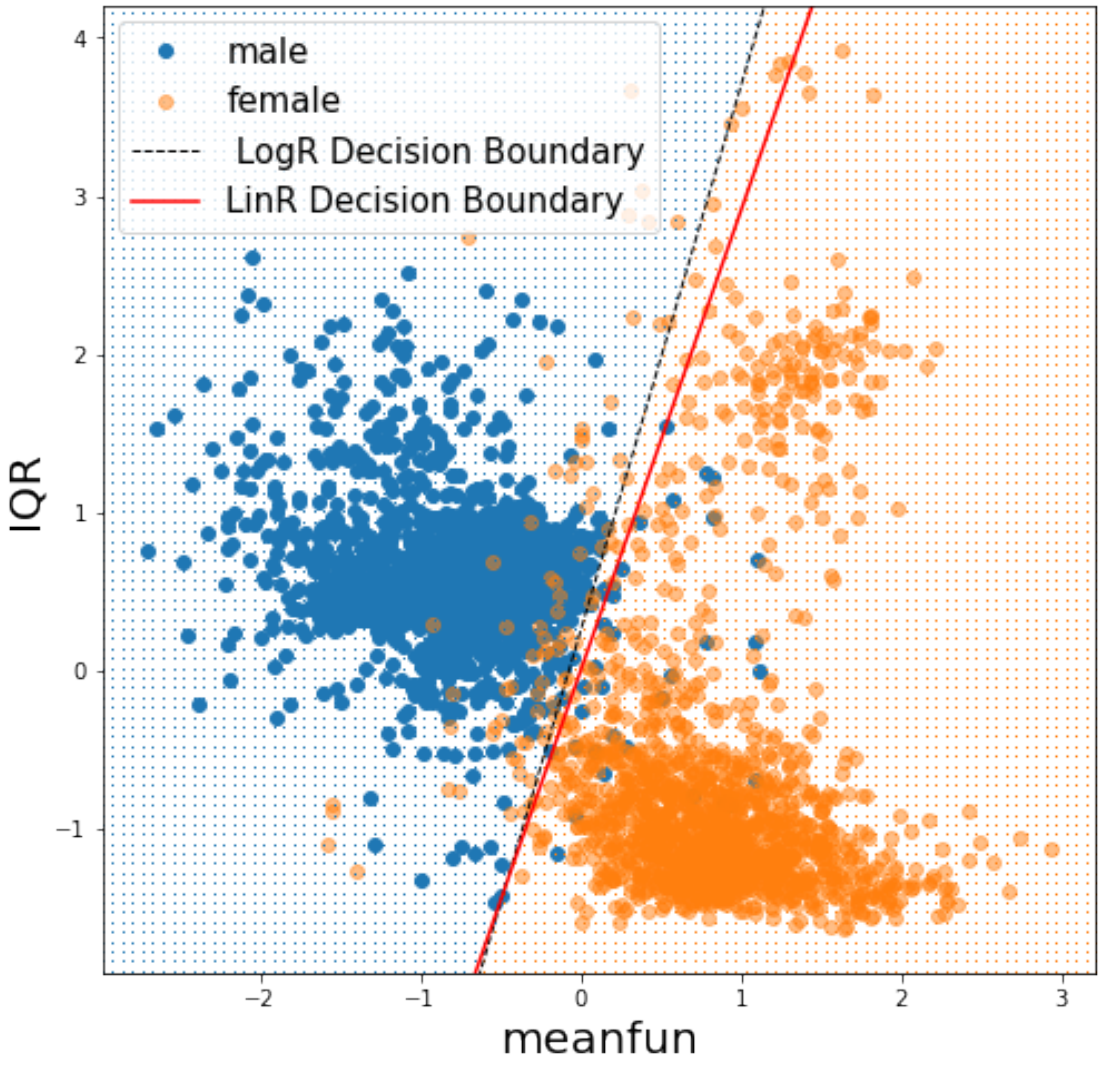
[30]:
```
beta = lg.intercept_
beta_1 = lg.coef_.T

print("The Coefficients:\n", beta_c)
print("The Intercept:", beta)
```

```
The Coefficients:
 [[ 0.35221767 -0.12125969]
 [-0.35221767  0.12125969]]
The Intercept: [-0.37726753]
```

[31]:
```
f, ax = plt.subplots(figsize=(8,8))

X1 = X["meanfun"]
X2 = X["IQR"]

plt.plot(X1[I_m],X2[I_m],'o',label="male")
plt.plot(X1[I_f],X2[I_f],'o',label="female",alpha=.5)

## As before we generate a meshgrid, but now we use qda.predict to guess at the␣
 ↪label.

xp,xP = plt.xlim()
yp,yP = plt.ylim()

c = -beta/beta_1[1]
m = -beta_1[0]/beta_1[1]

XX, YY = np.meshgrid(np.linspace(xp,xP, 100),np.linspace(yp,yP, 100))

## We now form a 10000x2 array of the (x,y) coordiantes for each point by␣
 ↪reshaping
## the XX and YY matricies and pasting them together. We need to feed a Nx2␣
 ↪vector
## into the qda.predict function, otherwise it will think we have too many␣
 ↪features.
## We can reshape it later to get our grid back

grid=np.concatenate([XX.reshape(-1,1),YY.reshape(-1,1)],axis=1)
```

```
ZZ = clf.predict(grid).reshape(XX.shape)  ## We predict, and reshape back to␣
 ↪the origional grid

z1 = ZZ == 'male'
z2 = ZZ == 'female'

plt.plot(XX[z1],YY[z1],',',color="C0")
plt.plot(XX[z2],YY[z2],',',color="C1")

xd = np.array([xp, xP])
yd = m*xd + c
plt.plot(xd, yd, 'k', lw=1, ls='--', label=" LogR Decision Boundary")

plt.plot(u,v,label="LinR Decision Boundary",color="red")

## We now reset the x and y limits to make sure our view is centered tightly
## around the data.

plt.xlabel("meanfun",fontsize=20)
plt.ylabel("IQR",fontsize=20)
plt.legend(fontsize=15)

ax.set_xlim([xp, xP])
ax.set_ylim([yp, yP])
```

[31]: (-1.9081155923982005, 4.2025798818885)



[40]:
```
#Cross Validation

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X[["IQR","meanfun"]], y,␣
↪test_size=0.2)

## Linear Regression Analysis

lr = LinearRegression()
lr.fit(X_train, pd.get_dummies(y_train))
y_hat = np.argmax(lr.predict(X_test), axis=1)
y_true = np.argmax(np.matrix(pd.get_dummies(y_test)), axis=1)
```

```
lr_2 = accuracy_score(y_hat,y_true)

## Quadratic Discriminant Analysis

qda = QuadraticDiscriminantAnalysis(store_covariance=True)
qda.fit(X_train, y_train)
qda_2 = qda.score(X_test,y_test)

## Logisitic Regression

clf = LogisticRegression()
clf.fit(X_train,y_train)
clf_2 = clf.score(X_test,y_test)


#For All Features

X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.4)

## Linear Regression Analysis

lr2 = LinearRegression()
lr2.fit(X_train2, pd.get_dummies(y_train2))
y_hat = np.argmax(lr2.predict(X_test2), axis=1)
y_true = np.argmax(np.matrix(pd.get_dummies(y_test2)), axis=1)
lr_all = accuracy_score(y_hat,y_true)

## Quadratic Discriminant Analysis

qda2 = QuadraticDiscriminantAnalysis(store_covariance=True)
qda2.fit(X_train2, y_train2)
qda_all = qda2.score(X_test2,y_test2)

## Logisitic Regression

clf2 = LogisticRegression()
clf2.fit(X_train2,y_train2)
clf_all = clf2.score(X_test2,y_test2)

print("Model Name \t All features \t Two features")
print("Linear Reg: \t {0:.3f} \t\t {1:.3f}".format(lr_all,lr_2))
print("Logistic Reg: \t {0:.3f} \t\t {1:.3f}".format(clf_all,clf_2))
print("QDA Score: \t {0:.3f} \t\t {1:.3f}".format(qda_all,qda_2))
```

```
Model Name      All features    Two features
Linear Reg:     0.970           0.965
Logistic Reg:   0.975           0.967
```

```
QDA Score:       0.964           0.965
```

```
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/sklearn/discriminant_analysis.py:808: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

2.0.2 Conclusion : After performing Linear Regression, Logistic Regression and QDA, we can see that we get higher scores with all features included rather than just fitting "meanfun" & "IQR" in all techniques

3 Problem 2: MRI Data

The dementia level for the Oasis 1 MRI dataset is based on a patient assessment. As a result, it is not clear whether the levels of 0, .5, 1 and 2 should actually be understood as meaningfully numeric, or if they in fact are categorical labels.

In this problem we want to treat them as categorical. However, we would also like to construct a slightly larger dataset, as we have seen that for images our 700 may not be sufficient. To construct a larger dataset we will again down sample the images, however this time we will use the down sampling to expand the dataset instead of throwing data away. After fixing a down sample rate D, we will construct one image out of the pixels nD, for $n = 1,2,..., $. We will also construct $nD + i$, for $i = 1, \ldots, D$. This way, by down sampling with a rate D, we construct D more pictures.

Note: It is very import that we perform the train test split *before* we expand the dataset through down sampling. If not, we are effectively training on the test data.

[44]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

file_dir = '/Users/sakshisuman12/Documents/NU_Coursework/MATH7243/labs/2/
 ↪MRI_Images/'
labels = pd.read_csv(file_dir + 'labels.csv')
display(labels)
y_new = labels.CDR
```

```
     Unnamed: 0                Filename             ID M/F Hand  Age  Educ  \
0             0   OAS1_0001_MR1_55.png  OAS1_0001_MR1   F    R   74     2
1             1  OAS1_0001_MR1_120.png  OAS1_0001_MR1   F    R   74     2
2             2  OAS1_0001_MR1_180.png  OAS1_0001_MR1   F    R   74     2
3             3   OAS1_0002_MR1_55.png  OAS1_0002_MR1   F    R   55     4
4             4  OAS1_0002_MR1_120.png  OAS1_0002_MR1   F    R   55     4
..          ...                    ...            ...  ..  ...  ...    ...
604         604  OAS1_0449_MR1_120.png  OAS1_0449_MR1   F    R   71     3
605         605  OAS1_0449_MR1_180.png  OAS1_0449_MR1   F    R   71     3
606         606   OAS1_0456_MR1_55.png  OAS1_0456_MR1   M    R   61     5
607         607  OAS1_0456_MR1_120.png  OAS1_0456_MR1   M    R   61     5
608         608  OAS1_0456_MR1_180.png  OAS1_0456_MR1   M    R   61     5
```

```
     SES  MMSE  CDR  eTIV   nWBV    ASF  Delay  Slice
0    3.0    29  0.0  1344  0.743  1.306    NaN     55
1    3.0    29  0.0  1344  0.743  1.306    NaN    120
2    3.0    29  0.0  1344  0.743  1.306    NaN    180
3    1.0    29  0.0  1147  0.810  1.531    NaN     55
4    1.0    29  0.0  1147  0.810  1.531    NaN    120
..   …   …  …    …     …    …     …    …
604  4.0    29  0.0  1264  0.818  1.388    NaN    120
605  4.0    29  0.0  1264  0.818  1.388    NaN    180
606  2.0    30  0.0  1637  0.780  1.072    NaN     55
607  2.0    30  0.0  1637  0.780  1.072    NaN    120
608  2.0    30  0.0  1637  0.780  1.072    NaN    180

[609 rows x 15 columns]
```

[45]:
```
labels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 609 entries, 0 to 608
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  609 non-null    int64
 1   Filename    609 non-null    object
 2   ID          609 non-null    object
 3   M/F         609 non-null    object
 4   Hand        609 non-null    object
 5   Age         609 non-null    int64
 6   Educ        609 non-null    int64
 7   SES         561 non-null    float64
 8   MMSE        609 non-null    int64
 9   CDR         609 non-null    float64
 10  eTIV        609 non-null    int64
 11  nWBV        609 non-null    float64
 12  ASF         609 non-null    float64
 13  Delay       0 non-null      float64
 14  Slice       609 non-null    int64
dtypes: float64(5), int64(6), object(4)
memory usage: 71.5+ KB
```

[46]:
```
y_new.max()
```

[46]: 2.0

[47]:
```
labels.size
```

[47]: 9135

[48]:
```
data_1 = np.zeros([609, 30976])

for n, file_name in enumerate(labels.Filename):
    data_1[n,:] = np.mean(matplotlib.image.imread(file_dir + file_name),axis=2).
 ↪reshape(-1)
```

[51]:
```
from sklearn.model_selection import train_test_split

X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(data_1,␣
 ↪y_new, test_size=0.2, random_state=0)
print(y_train_new.shape, y_test_new.shape)
```

```
(487,) (122,)
```

We want to sample the data array using the `data[start:stop:step]` slice paradigm. This means we are taking elements of the array `data` starting at `start`, ending at `stop` with step `step`. This is why previously `data[::DS]` down sampled at a rate of DS. For example,

```
lst = list(range(165)); lst[6::10]
```

returns

```
[6, 16, 26, 36, 46, 56, 66, 76, 86, 96, 106, 116, 126, 136, 146, 156]
```

We need to create two new arrays, one of shape $[561 \times DS, 30976/DS]$ containing the down sampled data, and one of shape $[561 \times DS]$ containing the labels. The for each of the N_{train} images in the training array, we need to create DS new down sampled images, with the downsample starting from i:

```
Xds_train[n+i, :] = X_train[i::DS]
```

This will split our images into DS down sampled images. We then need to be sure to save out the appropriate label:

[52]:
```
DS = 8              # Downsample rate, must be a multiple of 30976

N_train = y_train_new.shape[0]  # The length of the training data

if 30976/DS % 1 > 0:
    print("Downsample rate is not a multiple of 30976")
    DS = 1
    im_size = 30976
else:
    im_size = int(30976/DS)

Xds_train = np.zeros([N_train*DS, im_size])
yds_train = np.zeros(N_train*DS)

for n in range(N_train):
    for i in range(DS):
        Xds_train[n+i,:] = X_train_new[n,i::DS]
```

```
        yds_train[n+i] = y_new[n]

print(Xds_train.shape)
print(yds_train.shape)
print(N_train)
```

```
(3896, 3872)
(3896,)
487
```

[53]:
```
N_train = y_train_new.shape[0]
```

[54]:
```
N_train*DS
```

[54]: 3896

[55]:
```
N_train
```

[55]: 487

3.0.1 Question 1:

Based on the code above, downsample the test data in the same way. ()

[56]:
```
  DS = 8             # Downsample rate, must be a multiple of 30976

N_test= y_test_new.shape[0]  # The length of the testing data data

if 30976/DS % 1 > 0:
    print("Downsample rate is not a multiple of 30976")
    DS = 1
    im_size = 30976
else:
    im_size = int(30976/DS)
```

[57]:
```
Xds_test = np.zeros([N_test*DS, im_size])
yds_test = np.zeros(N_test*DS)
```

[58]:
```
N_test
```

[58]: 122

[60]:
```
Xds_test.shape
```

[60]: (976, 3872)

[59]:
```
im_size
```

[59]: 3872

[61]:
```
for n in range(N_test):
    for i in range(DS):
        Xds_test[n+i,:] = X_test_new[n,i::DS]
        yds_test[n+i] = y_new[n]

print(Xds_test.shape)
print(yds_test.shape)
```

```
(976, 3872)
(976,)
```

[62]:
```
yds_test
```

[62]:
```
array([0. , 0. , 0. , 0. , 0. , 0. , 0.5, 0.5, 0.5, 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.5, 0.5,
       0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 1. , 1. , 1. , 0. , 0. , 0. ,
       1. , 1. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 1. , 1. , 0.5,
       0.5, 0.5, 0.5, 0.5, 0.5, 0. , 0. , 0. , 0.5, 0.5, 0.5, 1. , 1. ,
       1. , 1. , 1. , 1. , 1. , 1. , 1. , 0. , 0. , 0. , 0.5, 0.5, 0.5,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.5, 0.5, 0.5, 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 1. , 1. , 1. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0.5, 0.5, 0.5, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
```

```
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
0. ])
```

3.0.2 Question 2:

Perform LDA, QDA, Logistic Regression and Categorical Linear Regression on the down sampled Oasis 1 dataset. How do these compare to linear regression?

[63]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(store_covariance=True)
lda.fit(Xds_train, yds_train.astype('str'))
print("LDA Score: %.3f"%lda.score(Xds_test,yds_test.astype('str')))
```

```
LDA Score: 0.926
```

[64]:
```
qda = QuadraticDiscriminantAnalysis(store_covariance=True)
qda.fit(Xds_train, yds_train.astype('str'))
print("QDA Score: %.3f"%qda.score(Xds_test,yds_test.astype('str')))
```

```
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/sklearn/discriminant_analysis.py:808: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

```
QDA Score: 0.879
```

[65]:
```
clf = LogisticRegression()
clf.fit(Xds_train, yds_train.astype('str'))

print("Logistic Regression Score: %.3f"%clf.score(Xds_test,yds_test.
 ↪astype('str')))
```

```
Logistic Regression Score: 0.939
```

```
/Users/sakshisuman12/opt/anaconda3/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[66]:
```
pd.get_dummies(yds_train)
```

[66]:

	0.0	0.5	1.0	2.0
0	1	0	0	0
1	1	0	0	0
2	1	0	0	0

```
3       1    0    0    0
4       1    0    0    0
…    …  …  …  …
3891    1    0    0    0
3892    1    0    0    0
3893    1    0    0    0
3894    1    0    0    0
3895    1    0    0    0

[3896 rows x 4 columns]
```

[67]:
```
len(Xds_train)
```

[67]:
```
3896
```

[69]:
```
from sklearn.metrics import accuracy_score
lr = LinearRegression()
lr.fit(Xds_train, pd.get_dummies(yds_train))
y_hat = np.argmax(lr.predict(Xds_test), axis=1)
y_true = np.argmax(np.matrix(pd.get_dummies(yds_test)), axis=1)
lr_2 = accuracy_score(y_hat,y_true)
```

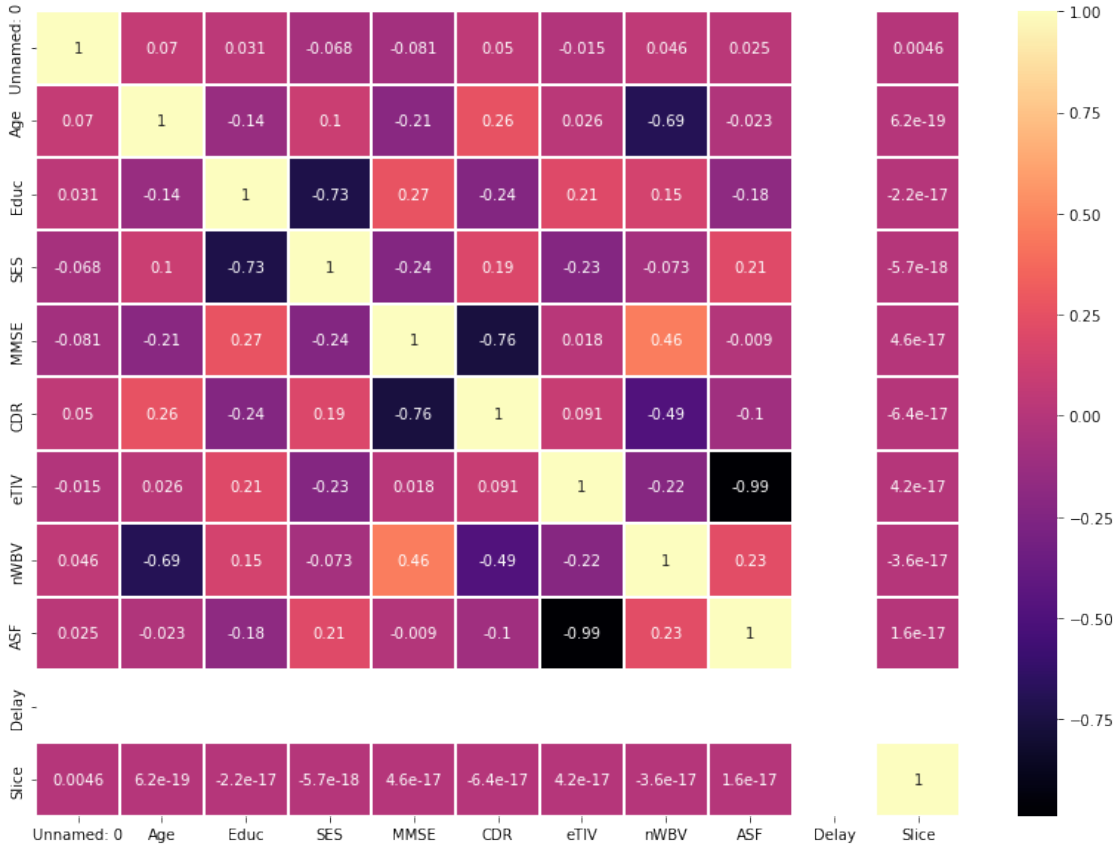
[70]:
```
lr_2
```

[70]:
```
0.9262295081967213
```

[71]:
```
from sklearn.metrics import confusion_matrix
conf_mx = confusion_matrix(y_true, y_hat)
conf_mx
```

[71]:
```
array([[891,  28,   6,   0],
       [ 16,  11,   3,   0],
       [ 12,   6,   2,   1],
       [  0,   0,   0,   0]])
```

[72]:
```
fig,ax = plt.subplots(figsize=(14, 10))

sns.heatmap(labels.corr(), ax=ax, annot = True, linewidths=0.05,cmap="magma")
plt.show()
```


Unnamed: 0
Age
Educ
SES
MMSE
CDR
eTIV
nWBV
ASF
Delay
Slice
1 0.07 0.031 -0.068 -0.081 0.05 -0.015 0.046 0.025 0.0046
0.07 1 -0.14 0.1 -0.21 0.26 0.026 -0.69 -0.023 6.2e-19
0.031 -0.14 1 -0.73 0.27 -0.24 0.21 0.15 -0.18 -2.2e-17
-0.068 0.1 -0.73 1 -0.24 0.19 -0.23 -0.073 0.21 -5.7e-18
-0.081 -0.21 0.27 -0.24 1 -0.76 0.018 0.46 -0.009 4.6e-17
0.05 0.26 -0.24 0.19 -0.76 1 0.091 -0.49 -0.1 -6.4e-17
-0.015 0.026 0.21 -0.23 0.018 0.091 1 -0.22 -0.99 4.2e-17
0.046 -0.69 0.15 -0.073 0.46 -0.49 -0.22 1 0.23 -3.6e-17
0.025 -0.023 -0.18 0.21 -0.009 -0.1 -0.99 0.23 1 1.6e-17
0.0046 6.2e-19 -2.2e-17 -5.7e-18 4.6e-17 -6.4e-17 4.2e-17 -3.6e-17 1.6e-17 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75