use-case1

November 9, 2024

1 Supervised learning 1

1.1 Use-Case 1:

- Fit a model using binary classification using logistic regression.
- Identify correlated variables and form a less complex model.

[ ]:
```
# import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.linear_model import LogisticRegression

# For Evaluation
import sklearn.metrics
```

[49]:
```
# Reading the dataset using pandas
data=pd.read_csv('voice.csv')
data.info()
data.head()
```

```
<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
```

[49]:

	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]

[51]:
```
data.isnull().sum()
```

[51]:
```
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
```

[52]:
```
# Label Encosing
le = LabelEncoder()
data['label']=le.fit_transform(data['label'])
print(data.head())
```

```
   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      1
1  0.250000  0.009014  0.007812  0.054688  0.046875  0.052632      1
2  0.271186  0.007990  0.007812  0.015625  0.007812  0.046512      1
3  0.250000  0.201497  0.007812  0.562500  0.554688  0.247119      1
4  0.266667  0.712812  0.007812  5.484375  5.476562  0.208274      1

[5 rows x 21 columns]
```

[53]:
```
data.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   int64
dtypes: float64(20), int64(1)
memory usage: 519.9 KB
```

[54]:
```
# #Divide the dataset into independent and dependent variables
x=data.drop('label',axis=1)
y=data['label']
print(x)
print(y)
```

```
      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
…          …         …         …         …         …         …          …
3163  0.131884  0.084734  0.153707  0.049285  0.201144  0.151859   1.762129
3164  0.116221  0.089221  0.076758  0.042718  0.204911  0.162193   0.693730
3165  0.142056  0.095798  0.183731  0.033424  0.224360  0.190936   1.876502
3166  0.143659  0.090628  0.184976  0.043508  0.219943  0.176435   1.591065
3167  0.165509  0.092884  0.183044  0.070072  0.250827  0.180756   1.705029

             kurt    sp.ent       sfm      mode  centroid   meanfun    minfun  \
0      274.402906  0.893369  0.491918  0.000000  0.059781  0.084279  0.015702
1      634.613855  0.892193  0.513724  0.000000  0.066009  0.107937  0.015826
2     1024.927705  0.846389  0.478905  0.000000  0.077316  0.098706  0.015656
```

```
3        4.177296  0.963322  0.727232  0.083878  0.151228  0.088965  0.017798
4        4.333713  0.971955  0.783568  0.104261  0.135120  0.106398  0.016931
…           …        …         …         …         …         …         …
3163     6.630383  0.962934  0.763182  0.200836  0.131884  0.182790  0.083770
3164     2.503954  0.960716  0.709570  0.013683  0.116221  0.188980  0.034409
3165     6.604509  0.946854  0.654196  0.008006  0.142056  0.209918  0.039506
3166     5.388298  0.950436  0.675470  0.212202  0.143659  0.172375  0.034483
3167     5.769115  0.938829  0.601529  0.267702  0.165509  0.185607  0.062257

        maxfun   meandom    mindom    maxdom   dfrange   modindx
0     0.275862  0.007812  0.007812  0.007812  0.000000  0.000000
1     0.250000  0.009014  0.007812  0.054688  0.046875  0.052632
2     0.271186  0.007990  0.007812  0.015625  0.007812  0.046512
3     0.250000  0.201497  0.007812  0.562500  0.554688  0.247119
4     0.266667  0.712812  0.007812  5.484375  5.476562  0.208274
…        …         …         …         …         …         …
3163  0.262295  0.832899  0.007812  4.210938  4.203125  0.161929
3164  0.275862  0.909856  0.039062  3.679688  3.640625  0.277897
3165  0.275862  0.494271  0.007812  2.937500  2.929688  0.194759
3166  0.250000  0.791360  0.007812  3.593750  3.585938  0.311002
3167  0.271186  0.227022  0.007812  0.554688  0.546875  0.350000

[3168 rows x 20 columns]
0       1
1       1
2       1
3       1
4       1
       ..
3163    0
3164    0
3165    0
3166    0
3167    0
Name: label, Length: 3168, dtype: int64
```

[55]:
```
# Train Test Split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
print(x_train.shape,x_test.shape)
print(y_train.shape,y_test.shape)
print(x_train.head())
print(x_test.head())
```

```
(2534, 20) (634, 20)
(2534,) (634,)
      meanfreq        sd    median       Q25       Q75       IQR      skew  \
644   0.091436  0.077062  0.070372  0.023457  0.153963  0.130505  2.354569
52    0.141461  0.072861  0.124879  0.090724  0.201194  0.110470  2.264115
```

```
2795  0.188687  0.042357  0.179200  0.162193  0.215289  0.053096  2.640277
2059  0.157375  0.059417  0.168703  0.139805  0.188033  0.048229  2.722492
2574  0.234202  0.027835  0.236000  0.226000  0.248500  0.022500  2.673033

          kurt    sp.ent       sfm      mode  centroid   meanfun    minfun  \
644    9.180173  0.956468  0.731009  0.013590  0.091436  0.161466  0.016529
52    13.082297  0.964873  0.734290  0.060127  0.141461  0.101395  0.021798
2795  11.259316  0.873810  0.296574  0.168000  0.188687  0.156115  0.047151
2059  11.598569  0.932947  0.610590  0.169874  0.157375  0.173366  0.045326
2574  10.482536  0.795870  0.127670  0.229500  0.234202  0.196528  0.048632

        maxfun   meandom    mindom    maxdom   dfrange   modindx
644   0.246154  0.776278  0.007812  6.125000  6.117188  0.121208
52    0.235294  0.896282  0.007812  5.156250  5.148438  0.239139
2795  0.279070  1.645312  0.281250  9.398438  9.117188  0.107519
2059  0.262295  0.175272  0.007812  0.351562  0.343750  0.183983
2574  0.275862  1.497533  0.023438  8.929688  8.906250  0.115710
      meanfreq        sd    median       Q25       Q75       IQR       skew  \
2955  0.134838  0.100239  0.202387  0.009745  0.215438  0.205693  29.969000
1130  0.202333  0.063001  0.221946  0.137544  0.264817  0.127273   2.000371
2609  0.213689  0.032540  0.212500  0.197500  0.231000  0.033500   4.443849
3031  0.178194  0.042880  0.180882  0.167769  0.191102  0.023333   3.700943
1522  0.156615  0.081866  0.135940  0.111843  0.238354  0.126511   3.783645

            kurt    sp.ent       sfm      mode  centroid   meanfun    minfun  \
2955  989.215323  0.785192  0.360357  0.000000  0.134838  0.162348  0.017957
1130    6.681799  0.873847  0.261759  0.272855  0.202333  0.123610  0.047291
2609   29.437982  0.806468  0.174356  0.197500  0.213689  0.194279  0.048485
3031   18.070029  0.855257  0.336562  0.167190  0.178194  0.170643  0.042373
1522   36.992437  0.926943  0.558977  0.000000  0.156615  0.134451  0.020833

        maxfun   meandom    mindom    maxdom   dfrange   modindx
2955  0.271186  0.007812  0.007812  0.007812  0.000000  0.000000
1130  0.269663  1.190168  0.023438  7.429688  7.406250  0.093438
2609  0.277457  0.725977  0.023438  3.984375  3.960938  0.096154
3031  0.192308  0.327930  0.170898  0.742188  0.571289  0.466168
1522  0.262295  0.100497  0.007812  0.539062  0.531250  0.244281
```

[56]:
```
# Scaling the Features: Logistic regression performs better when the features␣
 ↪are on a similar scale.
# Standardize the features using StandardScaler before training:
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

[57]:
```
# Train Logistic regression model
log_reg = LogisticRegression()
```

```
log_reg.fit(x_train,y_train)
```

[57]:
```
LogisticRegression()
```

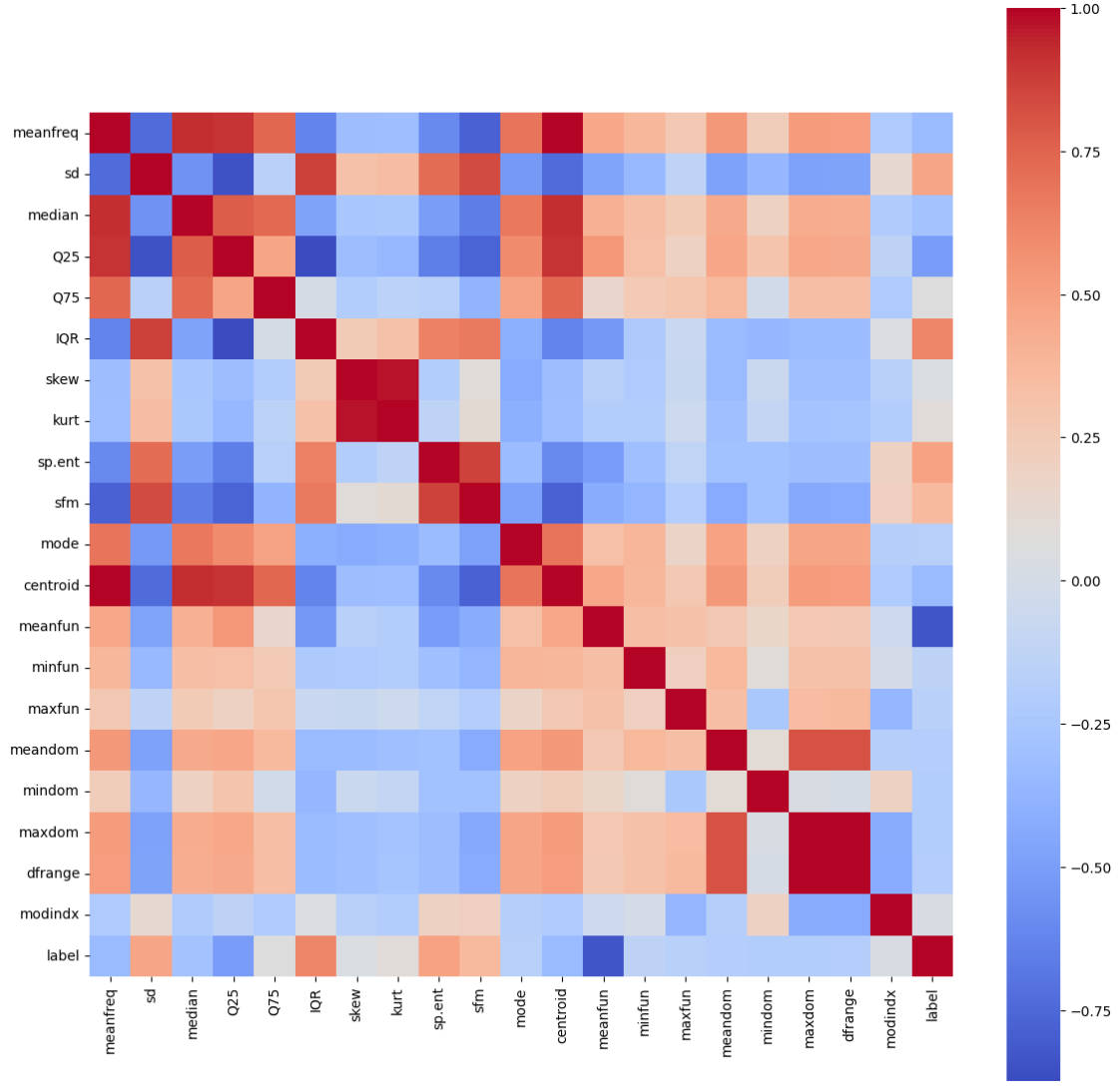
[58]:
```
# Checking prediction accuracy (Known data)
print(log_reg)
y_pred=log_reg.predict(x_train)
print(y_pred)
print("Train accuracy: ", sklearn.metrics.accuracy_score(y_train,y_pred))
```

```
LogisticRegression()
[0 1 0 … 0 1 0]
Train accuracy:  0.9751381215469613
```

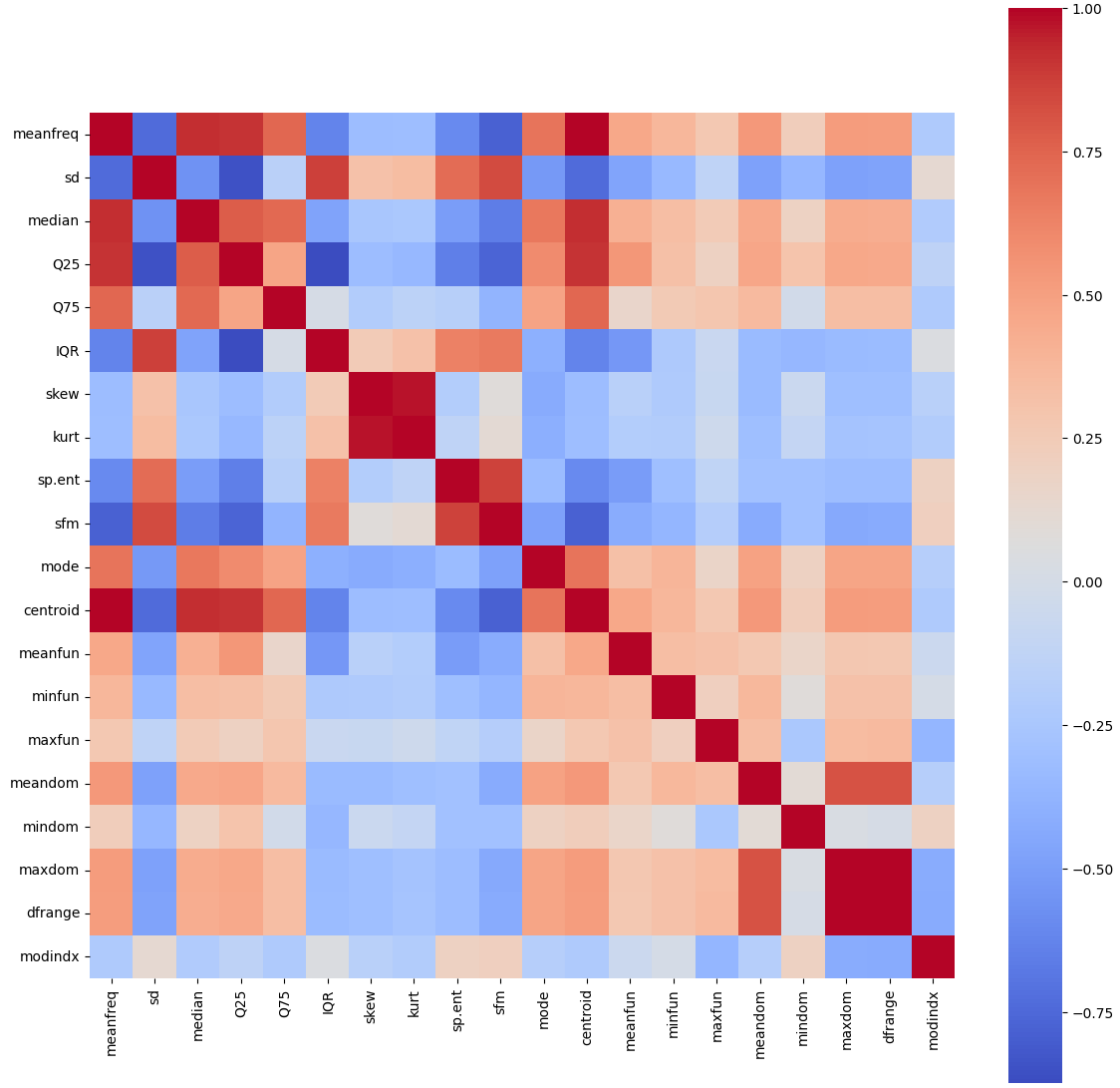
[59]:
```
# Checking prediction accuracy (UnKnown data)
y_pred=log_reg.predict(x_test)
print(y_pred)
print("Test accuracy: ", sklearn.metrics.accuracy_score(y_test,y_pred))
```

```
[0 1 0 0 1 1 1 1 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 0 0 0 1 0 1 0 1 1
 0 0 0 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1
 1 0 1 0 1 0 0 1 0 0 1 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 1 0 1 0 1 0 1 1 0
 1 0 1 0 1 1 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0 0 1 0 1 1 1 1 1 0 1 0 0 0 1 1 0
 0 1 1 1 1 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 0 1 0
 1 0 1 1 0 1 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 1
 1 1 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 1
 1 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 0 0 1 1 1 1 0 1 0 0 1 1 1 0 0
 1 1 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 1 0 0 1 0 1 1 0 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 0 0 1 1 1
 0 1 0 1 0 1 0 1 0 1 1 1 0 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 1 0
 0 1 0 0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 0 1 1 1
 0 0 0 0 1 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1
 0 1 0 1 0 0 1 1 0 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 1
 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 1
 1 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 0
 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0
 0 1 1 0 0]
Test accuracy:  0.9747634069400631
```

[72]:
```
corr = data.corr()
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True,  square = True,
            cmap= 'coolwarm')
plt.show()
```



[73]:
```
corr = x.corr()
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar = True,  square = True,
            cmap= 'coolwarm')
plt.show()
```



[83]:
```
# Features for the model ( remove collinearity)
# Removing multicollinearity helps achieve more stable, interpretable, and␣
 ↪reliable coefficients.
# When highly correlated features are present in a logistic regression model,␣
 ↪it can cause instability in the model's coefficients.
# Instability increases the variance of the model, making it sensitive to small␣
 ↪changes in the data and leading to a less generalizable model
# Consistent Feature Selection: Automate this feature selection by dropping one␣
 ↪variable from each highly correlated pair.
high_corr = corr[corr.abs() > 0.8]  # Using 0.8 as threshold for high␣
 ↪correlation
correlated_features = set()
for i in range(len(high_corr.columns)):
```

```
    for j in range(i):
        if abs(high_corr.iloc[i, j]) > 0.8:  # Identify pairs above threshold
            colname = high_corr.columns[i]
            correlated_features.add(colname)

print("Correlated features: ", correlated_features)
x_reduced = x.drop(labels=correlated_features, axis=1)
print("Remaining features in x_reduced:", x_reduced.columns.tolist())

# Removing highly correlated features simplifies the model, stabilizes the␣
 ↪logistic regression coefficients, and enhances generalization.
# This approach ensures that the model is interpreting each feature␣
 ↪independently, providing a clearer and more robust relationship with the␣
 ↪target variable.
```

```
Correlated features:  {'kurt', 'maxdom', 'sfm', 'dfrange', 'Q25', 'IQR',
'median', 'centroid'}
Remaining features in x_reduced: ['meanfreq', 'sd', 'Q75', 'skew', 'sp.ent',
'mode', 'meanfun', 'minfun', 'maxfun', 'meandom', 'mindom', 'modindx']
```

[75]:
```
# Train Test Split
x_train,x_test,y_train,y_test=train_test_split(x_reduced,y,test_size=0.2)
print(x_train.shape,x_test.shape)
print(y_train.shape,y_test.shape)
```

```
(2534, 12) (634, 12)
(2534,) (634,)
```

[76]:
```
# Scaling the Features: Logistic regression performs better when the features␣
 ↪are on a similar scale.
# Standardize the features using StandardScaler before training:
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

[77]:
```
# Train Logistic regression model
log_reg = LogisticRegression()
log_reg.fit(x_train,y_train)
```

[77]:
```
LogisticRegression()
```

[78]:
```
# Checking prediction accuracy (Known data)
print(log_reg)
y_pred=log_reg.predict(x_train)
print(y_pred)
print("Reduced Model Train accuracy: ", sklearn.metrics.
 ↪accuracy_score(y_train,y_pred))
```

```
LogisticRegression()
```

```
[1 0 1 … 0 1 0]
Train accuracy:  0.9688239936858721
```

[79]:
```
# Checking prediction accuracy (UnKnown data)
y_pred=log_reg.predict(x_test)
print(y_pred)
print("Reduced Model Test accuracy: ", sklearn.metrics.
 ↪accuracy_score(y_test,y_pred))
```

```
[1 1 1 0 1 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 1
 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 1 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0
 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1
 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 1 0 1 1 1 1 0 0 1 1 0 0 1 0 1 0 1 1 1 0 1 1
 1 0 1 1 0 1 0 0 1 1 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0
 0 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 0 0 1 1 0
 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0
 1 1 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 1 1 0
 1 0 1 0 1 0 1 1 1 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0 0 1 1 0 1 1 0 1 0 1 1 1 1
 1 0 1 1 1 0 1 0 1 1 0 1 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 0 0 0 0 1
 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 1 1 0 1 1 0
 0 0 1 1 1 0 0 1 1 0 1 0 1 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1
 0 0 0 1 0 1 0 0 1 1 1 0 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 0 0 1 0 0 0 0 1
 1 1 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 0 1 1 0 0 0 1 0 0 1 1 1 1 1
 0 1 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 1 1 1 0 1 0 0 0 0 1 1 1 0 0 1
 1 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0 1 1 0 1 1 0
 1 1 1 0 0]
Test accuracy:  0.9794952681388013
```

[85]:
```
# Adding Cross-Validation: To further validate the model, you might consider␣
 ↪using cross-validation to ensure that the model's performance is consistent.
# Use cross_val_score from sklearn.model_selection:
from sklearn.model_selection import cross_val_score
scores = cross_val_score(log_reg, x_reduced, y, cv=5)
print("Cross-validated scores on reduced model:", scores)
print("Mean cross-validation score:", scores.mean())
```

```
Cross-validated scores on reduced model: [0.70031546 0.8533123  0.9384858
0.87045814 0.84992101]
Mean cross-validation score: 0.8424985423176402
```

1.1.1 Cross-validation scores

The cross-validation scores obtained represent the accuracy of the logistic regression model on the reduced feature set across each fold in a 5-fold cross-validation. * Cross-validation helps ensure that the model's performance is consistent across different subsets of the data, reducing the chance of overfitting or underfitting. * It provides a more robust measure of model accuracy than a single train-test split, especially for smaller datasets or when evaluating model stability.In your case, an average score of approximately 84.25% suggests that the model is fairly accurate with the

reduced feature set, though the score may be slightly lower than when using the full feature set. * Variations between the scores (e.g., 0.7003 in one fold versus 0.9385 in another) may indicate that model performance varies depending on the data split. Consistency across scores usually indicates more stable performance. * The mean cross-validation score, 0.8425 (or about 84.25%), is the average accuracy across all five folds. * This value gives a good estimate of how well the model is expected to perform on unseen data, providing a more reliable measure than a single train-test split.

[ ]: