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

In [2]:
```
df = pd.read_csv(r"C:\Users\Praveen Dev\Downloads\archive\data.csv")
```

In [3]:
```
df
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_m
0	842302	M	17.99	10.38	122.80	10
1	842517	M	20.57	17.77	132.90	13
2	84300903	M	19.69	21.25	130.00	12
3	84348301	M	11.42	20.38	77.58	3
4	84358402	M	20.29	14.34	135.10	12
...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	14
565	926682	M	20.13	28.25	131.20	12
566	926954	M	16.60	28.08	108.30	8
567	927241	M	20.60	29.33	140.10	12
568	92751	B	7.76	24.54	47.92	1

569 rows × 33 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

In [5]:
```
df.drop(['id','Unnamed: 32'],axis = 1,inplace = True)
```

In [6]:
```
df.head()
```

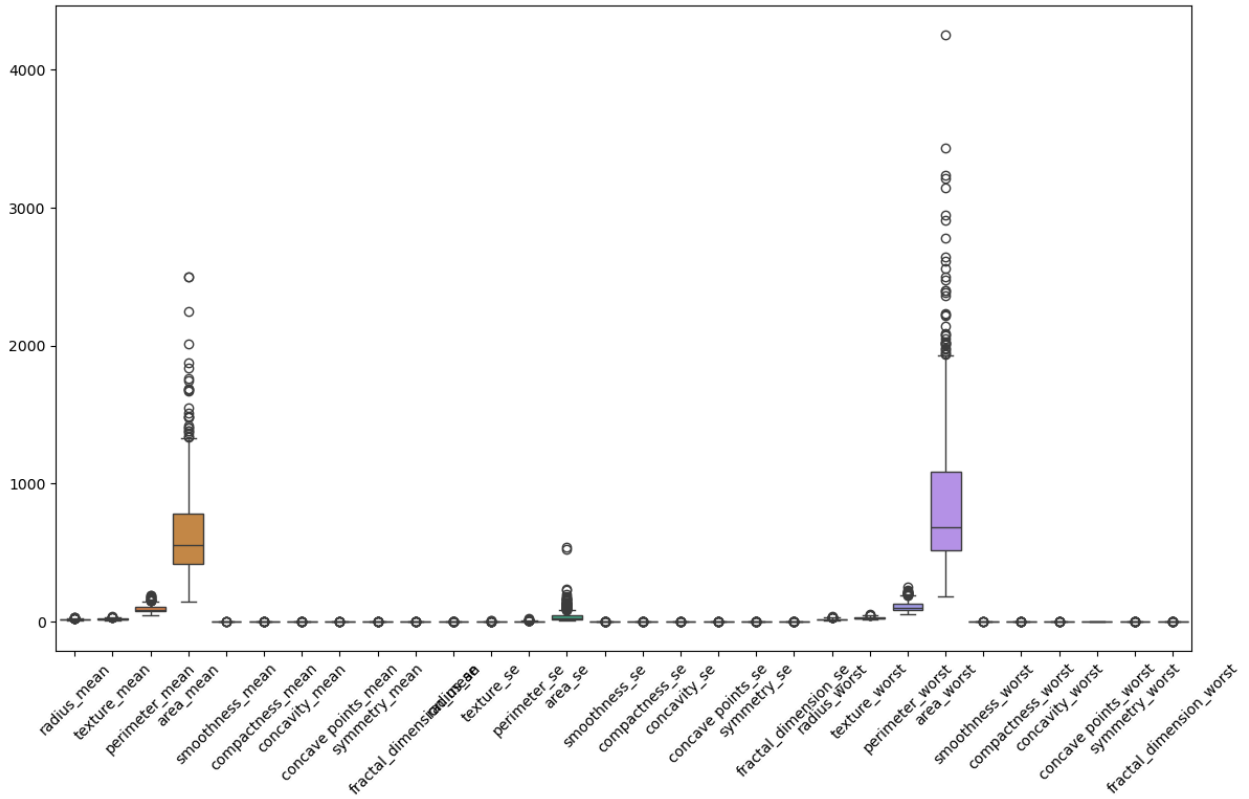
Out[6]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
0	M	17.99	10.38	122.80	1001.0	
1	M	20.57	17.77	132.90	1326.0	
2	M	19.69	21.25	130.00	1203.0	
3	M	11.42	20.38	77.58	386.1	
4	M	20.29	14.34	135.10	1297.0	

5 rows × 31 columns

Error Detection and Rectify

In [8]:
```
plt.figure(figsize = (14,8))
sns.boxplot(df)
plt.xticks(rotation = 45)
plt.show()
```



In [9]:
```
df.columns
```

Out[9]:
```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

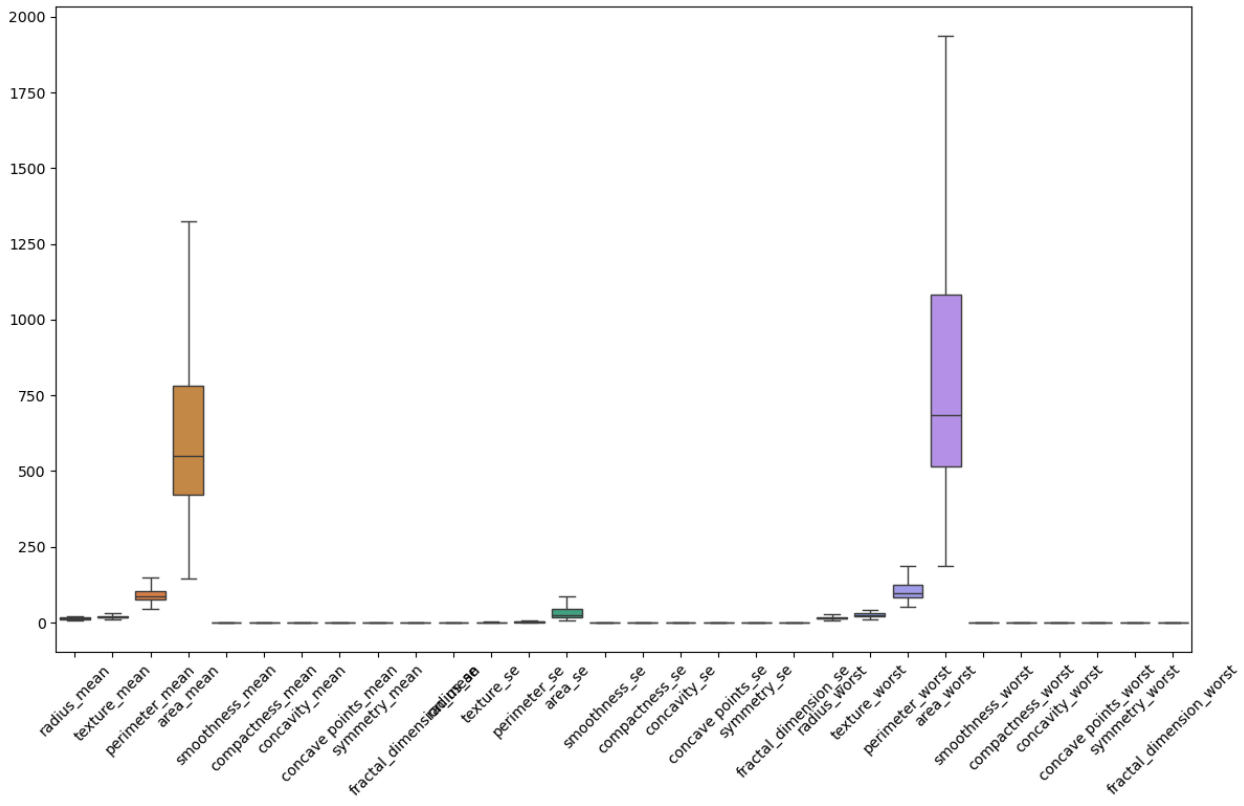
In [10]:
```
cols = [ 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']
```

In [11]:
```
cols
```

Out[11]:
```
['radius_mean',
 'texture_mean',
 'perimeter_mean',
 'area_mean',
 'smoothness_mean',
 'compactness_mean',
 'concavity_mean',
 'concave points_mean',
 'symmetry_mean',
 'fractal_dimension_mean',
 'radius_se',
 'texture_se',
 'perimeter_se',
 'area_se',
 'smoothness_se',
 'compactness_se',
 'concavity_se',
 'concave points_se',
 'symmetry_se',
 'fractal_dimension_se',
 'radius_worst',
 'texture_worst',
 'perimeter_worst',
 'area_worst',
 'smoothness_worst',
 'compactness_worst',
 'concavity_worst',
 'concave points_worst',
 'symmetry_worst',
 'fractal_dimension_worst']
```

In [12]:
```
for c in cols:
    q1 = df[c].quantile(0.25)
    q3 = df[c].quantile(0.75)
    iqr = q3-q1
    lower = q1-1.5*iqr
    upper = q3+1.5*iqr
    df[c] = df[c].map(lambda x: lower if x < lower else upper if x > upper els
```

In [13]:
```
plt.figure(figsize = (14,8))
sns.boxplot(df)
plt.xticks(rotation = 45)
plt.show()
```



Encoding

In [15]:
```
df.head()
```

Out[15]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
0	M	17.99	10.38	122.80	1001.0	
1	M	20.57	17.77	132.90	1326.0	
2	M	19.69	21.25	130.00	1203.0	
3	M	11.42	20.38	77.58	386.1	
4	M	20.29	14.34	135.10	1297.0	

5 rows × 31 columns

In [16]:
```
pd.get_dummies(df)
```

Out[16]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	17.99	10.38	122.80	1001.0	0.1184(
1	20.57	17.77	132.90	1326.0	0.0847
2	19.69	21.25	130.00	1203.0	0.1096(
3	11.42	20.38	77.58	386.1	0.1336
4	20.29	14.34	135.10	1297.0	0.1003(
...	...	...	...	...	
564	21.56	22.39	142.00	1326.3	0.1110(
565	20.13	28.25	131.20	1261.0	0.0978(
566	16.60	28.08	108.30	858.1	0.0845
567	20.60	29.33	140.10	1265.0	0.1178(
568	7.76	24.54	47.92	181.0	0.0579

569 rows × 32 columns

In [17]:
```
pd.get_dummies(df,drop_first = True)
```

Out[17]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	17.99	10.38	122.80	1001.0	0.1184(
1	20.57	17.77	132.90	1326.0	0.0847
2	19.69	21.25	130.00	1203.0	0.1096(
3	11.42	20.38	77.58	386.1	0.1336
4	20.29	14.34	135.10	1297.0	0.1003(
...	...	...	...	...	
564	21.56	22.39	142.00	1326.3	0.1110(
565	20.13	28.25	131.20	1261.0	0.0978(
566	16.60	28.08	108.30	858.1	0.0845
567	20.60	29.33	140.10	1265.0	0.1178(
568	7.76	24.54	47.92	181.0	0.0579

569 rows × 31 columns

In [21]:
```
pd.get_dummies(df,drop_first = True).replace([True,False],[1,0])
```

```
C:\Users\Praveen Dev\AppData\Local\Temp\ipykernel_15808\3313052313.py:1: Future
Warning: Downcasting behavior in `replace` is deprecated and will be removed in
a future version. To retain the old behavior, explicitly call `result.infer_obj
ects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('futur
e.no_silent_downcasting', True)`
  pd.get_dummies(df,drop_first = True).replace([True,False],[1,0])
```

Out[21]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	17.99	10.38	122.80	1001.0	0.1184(
1	20.57	17.77	132.90	1326.0	0.0847
2	19.69	21.25	130.00	1203.0	0.1096(
3	11.42	20.38	77.58	386.1	0.1336
4	20.29	14.34	135.10	1297.0	0.1003(
...	...	...	...	...	
564	21.56	22.39	142.00	1326.3	0.1110(
565	20.13	28.25	131.20	1261.0	0.0978(
566	16.60	28.08	108.30	858.1	0.0845
567	20.60	29.33	140.10	1265.0	0.1178(
568	7.76	24.54	47.92	181.0	0.0579

569 rows × 31 columns

In [23]:
```
df = pd.get_dummies(df,drop_first = True).replace([True,False],[1,0])
```

```
C:\Users\Praveen Dev\AppData\Local\Temp\ipykernel_15808\1066288145.py:1: Future
Warning: Downcasting behavior in `replace` is deprecated and will be removed in
a future version. To retain the old behavior, explicitly call `result.infer_obj
ects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('futur
e.no_silent_downcasting', True)`
  df = pd.get_dummies(df,drop_first = True).replace([True,False],[1,0])
```

In [24]:
```
df.head()
```

Out[24]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	17.99	10.38	122.80	1001.0	0.118400
1	20.57	17.77	132.90	1326.0	0.084740
2	19.69	21.25	130.00	1203.0	0.109600
3	11.42	20.38	77.58	386.1	0.133695
4	20.29	14.34	135.10	1297.0	0.100300

5 rows × 31 columns

Data separation

In [25]:
```
df.dtypes
```

Out[25]:
```
radius_mean               float64
texture_mean              float64
perimeter_mean            float64
area_mean                 float64
smoothness_mean           float64
compactness_mean          float64
concavity_mean            float64
concave points_mean       float64
symmetry_mean             float64
fractal_dimension_mean    float64
radius_se                 float64
texture_se                float64
perimeter_se              float64
area_se                   float64
smoothness_se             float64
compactness_se            float64
concavity_se              float64
concave points_se         float64
symmetry_se               float64
fractal_dimension_se      float64
radius_worst              float64
texture_worst             float64
perimeter_worst           float64
area_worst                float64
smoothness_worst          float64
compactness_worst         float64
concavity_worst           float64
concave points_worst      float64
symmetry_worst            float64
fractal_dimension_worst   float64
diagnosis_M                 int64
dtype: object
```

In [26]:
```
X = df.drop("diagnosis_M",axis = 1)
```

In [27]:
```
X
```

Out[27]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	17.99	10.38	122.80	1001.0	0.1184(
1	20.57	17.77	132.90	1326.0	0.0847
2	19.69	21.25	130.00	1203.0	0.1096(
3	11.42	20.38	77.58	386.1	0.1336
4	20.29	14.34	135.10	1297.0	0.1003(
...	...	...	...	...	
564	21.56	22.39	142.00	1326.3	0.1110(
565	20.13	28.25	131.20	1261.0	0.0978(
566	16.60	28.08	108.30	858.1	0.0845
567	20.60	29.33	140.10	1265.0	0.1178(
568	7.76	24.54	47.92	181.0	0.0579

569 rows × 30 columns

In [28]:
```
y = df["diagnosis_M"]
```

In [29]:
```
y
```

Out[29]:
```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis_M, Length: 569, dtype: int64
```

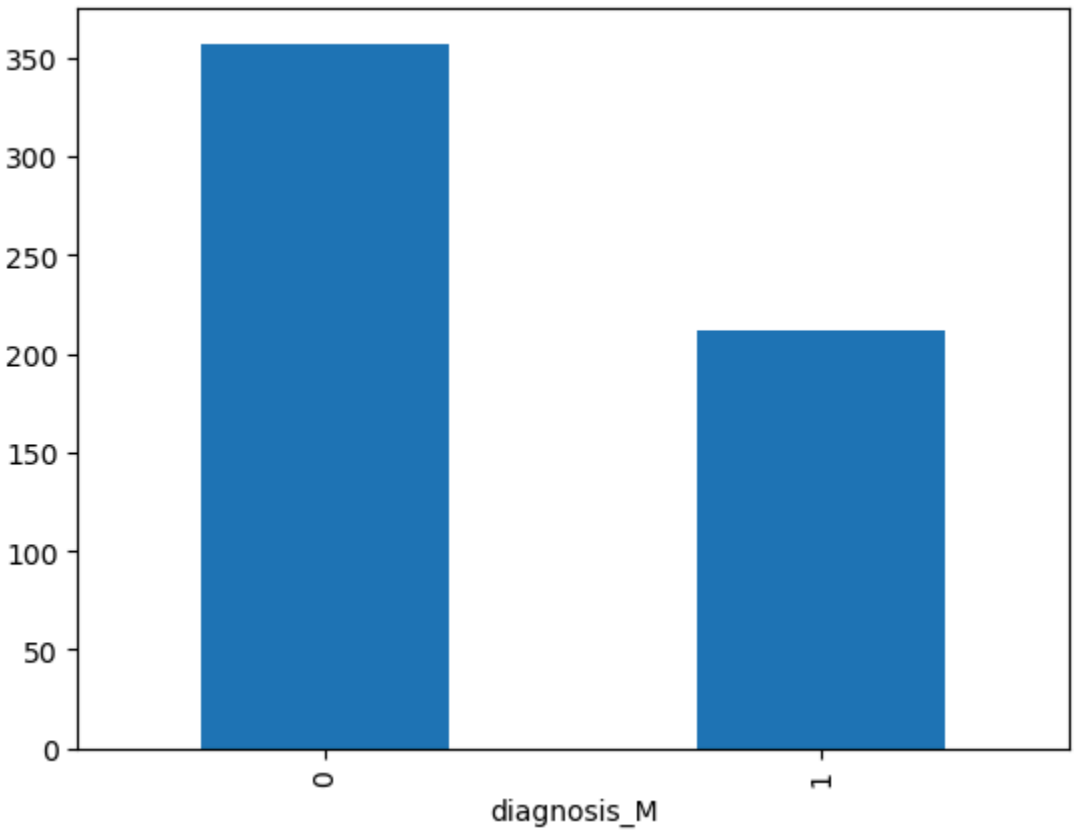
Imbalance

In [30]:
```
y.value_counts()
```

Out[30]:
```
diagnosis_M
0    357
1    212
Name: count, dtype: int64
```

In [32]:
```
y.value_counts().plot(kind = "bar")
```

Out[32]: <Axes: xlabel='diagnosis_M'>



In [33]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [34]:
```
ros = RandomOverSampler()
```

In [35]:
```
ros
```

Out[35]:
```
RandomOverSampler
RandomOverSampler()
```

In [36]:
```
X_ros, y_ros = ros.fit_resample(X,y)
```

In [37]:
```
X.shape
```

Out[37]: (569, 30)

In [38]:
```
X_ros.shape
```

Out[38]: (714, 30)

In [39]:
```
X_ros.value_counts
```

Out[39]:
```
<bound method DataFrame.value_counts of      radius_mean  texture_mean  perim
eter_mean  area_mean  smoothness_mean  \
0          17.99        10.380          122.80     1001.0         0.118400
1          20.57        17.770          132.90     1326.0         0.084740
2          19.69        21.250          130.00     1203.0         0.109600
3          11.42        20.380           77.58      386.1         0.133695
4          20.29        14.340          135.10     1297.0         0.100300
..           ...           ...             ...        ...              ...
709        14.58        21.530           97.41      644.8         0.105400
710        15.46        23.950          103.80      731.3         0.118300
711        20.48        21.460          132.50     1306.0         0.083550
712        13.77        22.290           90.63      588.9         0.120000
713        15.22        30.245          103.40      716.9         0.104800

     compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
0             0.22862         0.28241              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.22862         0.24140              0.10520         0.2464
4             0.13280         0.19800              0.10430         0.1809
..                ...             ...                  ...            ...
709           0.18680         0.14250              0.08783         0.2252
710           0.18700         0.20300              0.08520         0.1807
711           0.08348         0.09042              0.06022         0.1467
712           0.12670         0.13850              0.06526         0.1834
713           0.20870         0.25500              0.09429         0.2128

     fractal_dimension_mean  ...  radius_worst  texture_worst  \
0                   0.07871  ...         25.38          17.33
1                   0.05667  ...         24.99          23.41
2                   0.05999  ...         23.57          25.53
3                   0.07875  ...         14.91          26.50
4                   0.05883  ...         22.54          16.67
..                      ...  ...           ...            ...
709                 0.06924  ...         17.62          33.21
710                 0.07083  ...         17.11          36.33
711                 0.05177  ...         24.22          26.17
712                 0.06877  ...         16.39          34.01
713                 0.07152  ...         17.52          42.68

     perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
0             184.60     1937.05            0.1622            0.62695
1             158.80     1937.05            0.1238            0.18660
2             152.50     1709.00            0.1444            0.42450
3              98.87      567.70            0.1901            0.62695
4             152.20     1575.00            0.1374            0.20500
..               ...         ...               ...                ...
709           122.40      896.90            0.1525            0.62695
710           117.70      909.40            0.1732            0.49670
711           161.70     1750.00            0.1228            0.23110
712           111.60      806.90            0.1737            0.31220
713           128.70      915.00            0.1417            0.62695

     concavity_worst  concave points_worst  symmetry_worst  \
```

```
0             0.7119                0.2654          0.41915
1             0.2416                0.1860          0.27500
2             0.4504                0.2430          0.36130
3             0.6869                0.2575          0.41915
4             0.4000                0.1625          0.23640
..               ...                   ...              ...
709           0.5539                0.2701          0.41915
710           0.5911                0.2163          0.30130
711           0.3158                0.1445          0.22380
712           0.3809                0.1673          0.30800
713           0.7855                0.2356          0.40890

     fractal_dimension_worst
0                    0.11890
1                    0.08902
2                    0.08758
3                    0.12301
4                    0.07678
..                       ...
709                  0.12301
710                  0.10670
711                  0.07127
712                  0.09333
713                  0.12301

[714 rows x 30 columns]>
```

In [40]:
```
y.shape
```

Out[40]:
```
(569,)
```

In [41]:
```
y_ros.shape
```

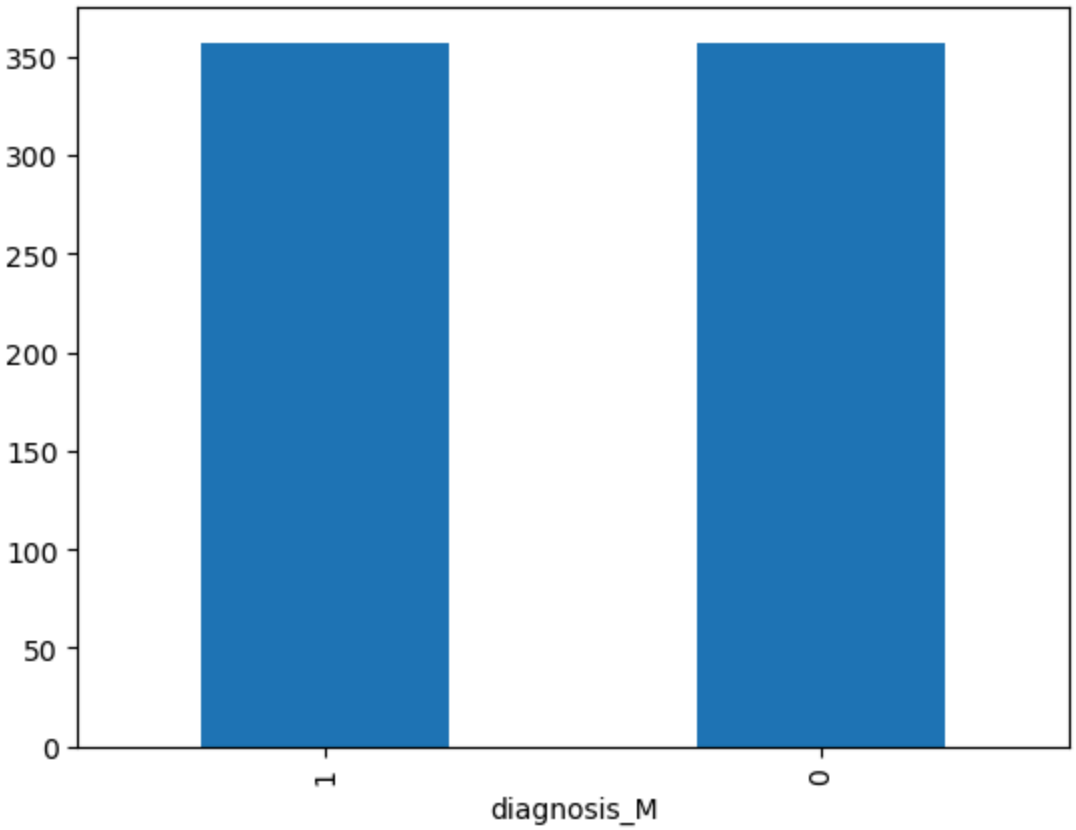
Out[41]:
```
(714,)
```

In [43]:
```
y_ros.value_counts()
```

Out[43]:
```
diagnosis_M
1    357
0    357
Name: count, dtype: int64
```

In [45]:
```
y_ros.value_counts().plot(kind = "bar")
```

Out[45]:
```
<Axes: xlabel='diagnosis_M'>
```



Data splitting

In [46]:
```
from sklearn.model_selection import train_test_split
```

In [129…
```
X_train, X_test, y_train, y_test = train_test_split(X_ros, y_ros, test_size =
```

In [130…
```
X.shape
```

Out[130… (569, 30)

In [131…
```
X_train.shape
```

Out[131… (571, 30)

In [132…
```
X_test.shape
```

Out[132… (143, 30)

In [133…
```
X_ros.shape
```

Out[133… (714, 30)

In [134…
```
y.shape
```

Out[134… (569,)

In [135…
```
y_train.shape
```

Out[135… (571,)

In [136…
```
y_test.shape
```

Out[136… (143,)

In [137…
```
y_ros.shape
```

Out[137… (714,)

Model Building

In [138…
```
from sklearn.tree import DecisionTreeClassifier
```

In [139…
```
dt_model = DecisionTreeClassifier()
```

In [140…
```
dt_model
```

Out[140…
```
DecisionTreeClassifier()
```

In [141…
```
dt_model.fit(X_train, y_train)
```

Out[141…
```
DecisionTreeClassifier()
```

In [142…
```
dt_model.score(X_test, y_test)
```

Out[142… 0.993006993006993

Accuracy score

In [143…
```
from sklearn.metrics import accuracy_score
```

In [144…
```
dt_model.predict(X_test)
```

Out[144…
```
array([1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1])
```

In [145…
```
y_pred = dt_model.predict(X_test)
```

In [146…
```
y_pred
```

Out[146…
```
array([1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1])
```

In [147…
```
accuracy_score(y_test, y_pred)
```

Out[147…
```
0.993006993006993
```

Auto Random State

In [94]:
```
sc_list = []
for i in range(1000):
    Xtrain,Xtest,ytrain,ytest = train_test_split(X_ros, y_ros, test_size = 0.2
    dt_model = DecisionTreeClassifier()
    dt_model.fit(Xtrain, ytrain)
    sc = dt_model.score(Xtest, ytest)
    sc_list.append(sc)
m = max(sc_list)
rs = sc_list.index(m)
print("Random State",rs)
print("Score",m)
```

```
Random State 153
Score 1.0
```

In [95]:
```
sc_list = []
for i in range(1000):
    Xtrain,Xtest,ytrain,ytest = train_test_split(X_train, y_train, test_size =
    dt_model = DecisionTreeClassifier()
    dt_model.fit(Xtrain, ytrain)
    sc = dt_model.score(Xtest, ytest)
    sc_list.append(sc)
m = max(sc_list)
rs = sc_list.index(m)
print("Random State",rs)
print("Score",m)
```

```
Random State 466
Score 1.0
```

Cross Validation

In [148…
```
from sklearn.model_selection import cross_val_score
```

In [149…
```
dt_model1 = DecisionTreeClassifier()
```

In [150…
```
dt_model1
```

Out[150…
```
DecisionTreeClassifier()
```

Normal cross validation

In [151…
```
score = cross_val_score(dt_model1, X_train, y_train, cv = 20, n_jobs = -1)
```

In [152…
```
score.mean()
```

Out[152…
```
np.float64(0.9457512315270936)
```

In [153…
```
score = cross_val_score(dt_model1, X, y, cv = 20, n_jobs = -1)
```

In [154…
```
score.mean()
```

Out[154…
```
np.float64(0.926293103448276)
```

In [155…
```
score = cross_val_score(dt_model1, X_ros, y_ros, cv = 20, n_jobs = -1)
```

In [156…
```
score.mean()
```

Out[156…
```
np.float64(0.9677380952380952)
```

KFold cross validation

In [157…
```
from sklearn.model_selection import KFold,StratifiedKFold
```

In [158…
```
kf = KFold(n_splits = 30)
```

In [160…
```
score1 = cross_val_score(dt_model1, X_train, y_train, cv = kf, n_jobs = -1)
```

In [161…
```
score1.mean()
```

Out[161…
```
np.float64(0.9457894736842103)
```

In [162…
```
score1 = cross_val_score(dt_model1, X, y, cv = kf, n_jobs = -1)
```

In [163…
```
score1.mean()
```

Out[163…
```
np.float64(0.9244639376218322)
```

In [164…
```
score1 = cross_val_score(dt_model1, X_ros, y_ros, cv = kf, n_jobs = -1)
```

In [165…
```
score1.mean()
```

Out[165…
```
np.float64(0.9708333333333333)
```

Stratified KFold cross validation

In [166…
```
skf = StratifiedKFold(n_splits = 30)
```

In [167…
```
score2 = cross_val_score(dt_model1, X_train, y_train, cv = skf, n_jobs = -1)
```

In [168…
```
score2.mean()
```

Out[168…
```
np.float64(0.940438596491228)
```

In [169…
```
score2 = cross_val_score(dt_model1, X, y, cv = skf, n_jobs = -1)
```

In [170…
```
score2.mean()
```

Out[170…
```
np.float64(0.9332358674463935)
```

In [171…
```
score2 = cross_val_score(dt_model1, X_ros, y_ros, cv = skf, n_jobs = -1)
```

In [172…
```
score2.mean()
```

Out[172…
```
np.float64(0.9719806763285025)
```

Auto ML

In [188…
```
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,Gradier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
```

In [189…
```
rfc = RandomForestClassifier()
abc = AdaBoostClassifier()
gbc = GradientBoostingClassifier()
dtc = DecisionTreeClassifier()
svm = SVC()
```

```
knn =  KNeighborsClassifier()
lr = LinearRegression()
lg = LogisticRegression()
```

In [190…
```
models = [rfc, abc, gbc, dtc, svm, knn, lr, lg]
```

In [191…
```
models
```

Out[191…
```
[RandomForestClassifier(),
 AdaBoostClassifier(),
 GradientBoostingClassifier(),
 DecisionTreeClassifier(),
 SVC(),
 KNeighborsClassifier(),
 LinearRegression(),
 LogisticRegression()]
```

Auto ML with Normal cross validation

In [192…
```
for m in models:
    scores = cross_val_score(m, X_ros, y_ros, cv = 20, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9763492063492063
AdaBoostClassifier() ===> 0.9776984126984127
GradientBoostingClassifier() ===> 0.9805158730158731
DecisionTreeClassifier() ===> 0.9691269841269842
SVC() ===> 0.9018253968253969
KNeighborsClassifier() ===> 0.921468253968254
LinearRegression() ===> 0.5332073717552147
LogisticRegression() ===> 0.9453174603174602
```

In [193…
```
for m in models:
    scores = cross_val_score(m, X_train, y_train, cv = 20, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9736453201970443
AdaBoostClassifier() ===> 0.9754310344827587
GradientBoostingClassifier() ===> 0.9789408866995075
DecisionTreeClassifier() ===> 0.945566502463054
SVC() ===> 0.9021551724137928
KNeighborsClassifier() ===> 0.9192733990147783
LinearRegression() ===> 0.7370185737351832
LogisticRegression() ===> 0.9474137931034482
```

In [194…
```
for m in models:
    scores = cross_val_score(m, X, y, cv = 20, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9615763546798028
AdaBoostClassifier() ===> 0.9754310344827587
GradientBoostingClassifier() ===> 0.9616379310344827
DecisionTreeClassifier() ===> 0.9088054187192117
SVC() ===> 0.9194581280788178
KNeighborsClassifier() ===> 0.9299261083743844
LinearRegression() ===> 0.661952402827339
LogisticRegression() ===> 0.9385467980295568
```

Auto ML with KFold cross validation

In [196…

```
for m in models:
    kf = KFold(n_splits = 30)
    scores = cross_val_score(m, X_ros, y_ros, cv = kf, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9722222222222221
AdaBoostClassifier() ===> 0.9777777777777779
GradientBoostingClassifier() ===> 0.9805555555555556
DecisionTreeClassifier() ===> 0.9694444444444446
SVC() ===> 0.9003019323671498
KNeighborsClassifier() ===> 0.9257850241545894
LinearRegression() ===> 0.5131432407226977
LogisticRegression() ===> 0.9452898550724638
```

In [197…

```
for m in models:
    kf = KFold(n_splits = 30)
    scores = cross_val_score(m, X_train, y_train, cv = kf, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9720175438596489
AdaBoostClassifier() ===> 0.9685087719298244
GradientBoostingClassifier() ===> 0.9737719298245612
DecisionTreeClassifier() ===> 0.9544736842105261
SVC() ===> 0.8985087719298244
KNeighborsClassifier() ===> 0.9159649122807018
LinearRegression() ===> 0.7338963804786997
LogisticRegression() ===> 0.9352631578947367
```

In [198…

```
for m in models:
    kf = KFold(n_splits = 30)
    scores = cross_val_score(m, X, y, cv = kf, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9596491228070175
AdaBoostClassifier() ===> 0.9701754385964912
GradientBoostingClassifier() ===> 0.963157894736842
DecisionTreeClassifier() ===> 0.9313840155945419
SVC() ===> 0.9175438596491226
KNeighborsClassifier() ===> 0.9280701754385965
LinearRegression() ===> 0.6722821128979797
LogisticRegression() ===> 0.9368421052631579
```

Auto ML with StratifiedKFold cross validation

In [202…

```
for m in models:
    skf = StratifiedKFold(n_splits = 30)
    scores = cross_val_score(m, X_ros, y_ros, cv = skf, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9762681159420289
AdaBoostClassifier() ===> 0.9804347826086955
GradientBoostingClassifier() ===> 0.9777173913043478
DecisionTreeClassifier() ===> 0.9803743961352658
SVC() ===> 0.9003623188405798
KNeighborsClassifier() ===> 0.9229468599033818
LinearRegression() ===> 0.7489527509373491
LogisticRegression() ===> 0.9455314009661836
```

In [203…

```
for m in models:
    skf = StratifiedKFold(n_splits = 30)
    scores = cross_val_score(m, X_train, y_train, cv = skf, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9720175438596488
AdaBoostClassifier() ===> 0.9720175438596491
GradientBoostingClassifier() ===> 0.9772807017543859
DecisionTreeClassifier() ===> 0.9405263157894735
SVC() ===> 0.9002631578947367
KNeighborsClassifier() ===> 0.9194736842105262
LinearRegression() ===> 0.7479656219633654
LogisticRegression() ===> 0.9441228070175437
```

In [204…

```
for m in models:
    skf = StratifiedKFold(n_splits = 30)
    scores = cross_val_score(m, X, y, cv = skf, n_jobs = -1)
    scores.mean()
    print(m,"===>",scores.mean())
```

```
RandomForestClassifier() ===> 0.9684210526315788
AdaBoostClassifier() ===> 0.9700779727095515
GradientBoostingClassifier() ===> 0.9649122807017544
DecisionTreeClassifier() ===> 0.9349902534113058
SVC() ===> 0.9174463937621831
KNeighborsClassifier() ===> 0.929727095516569
LinearRegression() ===> 0.7439425653612746
LogisticRegression() ===> 0.9437621832358672
```

Hyperparameter Tuning

GridsearchCV

In [205…
```
from sklearn.model_selection import GridSearchCV
```

In [206…
```
dt_model2 = DecisionTreeClassifier()
```

In [207…
```
dt_model2
```

Out[207…
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [208…
```
param = {
            "criterion" : ["gini", "entropy", "log_loss"],
            "splitter" : ["best", "random"],
            "max_depth": [2,3,6,7],
            "min_samples_split":[3,4,5]
        }
```

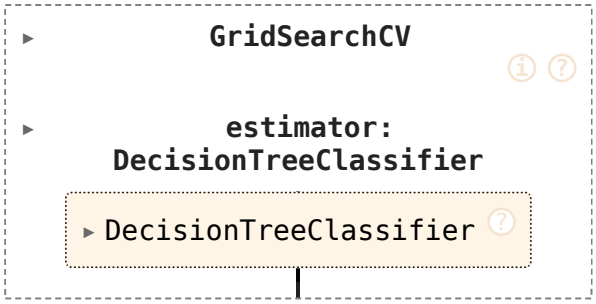
In [209…
```
param
```

Out[209…
```
{'criterion': ['gini', 'entropy', 'log_loss'],
 'splitter': ['best', 'random'],
 'max_depth': [2, 3, 6, 7],
 'min_samples_split': [3, 4, 5]}
```

In [211…
```
gridcv = GridSearchCV(estimator = dt_model2, param_grid = param, cv = 20, n_jc
```

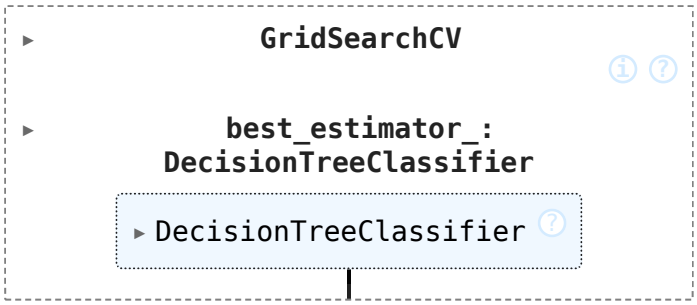
In [212…
```
gridcv
```

Out[212…



In [213…
```
gridcv.fit(X_ros, y_ros)
```

Out[213…



In [214…
```
gridcv.best_score_
```

Out[214…
```
np.float64(0.9721825396825396)
```

In [215…
```
gridcv.best_params_
```

Out[215…
```
{'criterion': 'log_loss',
 'max_depth': 7,
 'min_samples_split': 3,
 'splitter': 'random'}
```

In [216…
```
gridcv.best_estimator_
```

Out[216…
```
DecisionTreeClassifier(criterion='log_loss', max_depth=7, min_samples_split=3,
                       splitter='random')
```

In [217…
```
gridcv.best_index_
```

Out[217…
```
np.int64(67)
```

In [218…
```
gridcv.cv_results_
```

Out[218… {'mean_fit_time': array([0.06036973, 0.02846246, 0.04243176, 0.01052163, 0.07
467874,
        0.03691343, 0.0486125 , 0.02411234, 0.05056713, 0.02232847,
        0.04598544, 0.02386981, 0.06293334, 0.02409006, 0.06961389,
        0.03828204, 0.12114849, 0.03731862, 0.06637474, 0.02575175,
        0.06427271, 0.02098439, 0.06050185, 0.02505506, 0.0481939 ,
        0.02339276, 0.09462315, 0.03055859, 0.04875273, 0.01971964,
        0.0622254 , 0.02333337, 0.06372426, 0.02451555, 0.06340023,
        0.02445394, 0.08888628, 0.04936635, 0.08592076, 0.02330061,
        0.07224568, 0.02555938, 0.07492771, 0.02423429, 0.07922421,
        0.02371305, 0.07708691, 0.04342207, 0.07002262, 0.02341399,
        0.05169966, 0.02153895, 0.04899163, 0.02462616, 0.06322519,
        0.02345639, 0.06425418, 0.02214134, 0.08322382, 0.03827938,
        0.08175311, 0.02590169, 0.07644235, 0.02603408, 0.07691089,
        0.02549948, 0.07683738, 0.02666823, 0.10653536, 0.04398998,
        0.07798668, 0.02084184]),
 'std_fit_time': array([0.00629391, 0.00392568, 0.00694125, 0.00287634, 0.031
44215,
        0.01353805, 0.00400416, 0.00534303, 0.00611044, 0.0044558 ,
        0.00548543, 0.00581477, 0.00607223, 0.00442425, 0.00543223,
        0.01504598, 0.02128597, 0.01655615, 0.00452123, 0.00551383,
        0.00537431, 0.00353327, 0.00559054, 0.00417483, 0.00674955,
        0.00641116, 0.01804659, 0.01119026, 0.00533652, 0.00532111,
        0.00576454, 0.00351186, 0.00410148, 0.00486616, 0.00566994,
        0.00337937, 0.02800679, 0.018302 , 0.01863563, 0.00380685,
        0.00559281, 0.00472055, 0.00473245, 0.00373649, 0.00595446,
        0.00464753, 0.00908897, 0.01478407, 0.01900592, 0.00499645,
        0.00553864, 0.00420883, 0.00422725, 0.00266905, 0.0047585 ,
        0.00457685, 0.0047407 , 0.00236244, 0.03012143, 0.01144341,
        0.01098598, 0.00321189, 0.00495945, 0.00357947, 0.00567728,
        0.00377345, 0.00559118, 0.00427926, 0.04227406, 0.01620556,
        0.00620935, 0.00650915]),
 'mean_score_time': array([0.01896262, 0.01585283, 0.01300445, 0.00626137,
0.02270553,
        0.02107185, 0.01351775, 0.01308924, 0.01292465, 0.01243827,
        0.01256173, 0.01181934, 0.0127236 , 0.01528898, 0.01375798,
        0.02443874, 0.02484945, 0.01516478, 0.0130805 , 0.01185824,
        0.01242975, 0.0110647 , 0.01347048, 0.01282629, 0.01211632,
        0.01594288, 0.02495654, 0.01458483, 0.01176351, 0.01196586,
        0.01352642, 0.01299686, 0.01321989, 0.01401502, 0.01388533,
        0.0131961 , 0.01902515, 0.02389491, 0.01295035, 0.01219711,
        0.01304209, 0.01308669, 0.01259511, 0.01325972, 0.01205143,
        0.01287775, 0.01917762, 0.02527766, 0.01379532, 0.01431872,
        0.01206377, 0.01277742, 0.01329018, 0.01312101, 0.01322373,
        0.01354675, 0.01239405, 0.01193385, 0.02270269, 0.02202519,
        0.01315484, 0.01324427, 0.01226553, 0.01244681, 0.01305028,
        0.01385963, 0.01214449, 0.01360185, 0.02095015, 0.01778666,
        0.01205035, 0.00967163]),
 'std_score_time': array([0.00309453, 0.00326517, 0.00376791, 0.00186063, 0.0
1247786,
        0.00929454, 0.00264853, 0.00395278, 0.00315639, 0.00228521,
        0.00394211, 0.00330107, 0.00236074, 0.0048176 , 0.00323527,
        0.01425611, 0.0150242 , 0.00721589, 0.00306433, 0.00427754,
        0.00282861, 0.00410807, 0.00370616, 0.00315189, 0.00315381,

```
        0.00590006, 0.00788222, 0.00567935, 0.00369506, 0.00403262,
        0.00296395, 0.00248734, 0.00212059, 0.00590337, 0.00275272,
        0.00269171, 0.01001499, 0.00922146, 0.00173121, 0.00221727,
        0.00269136, 0.00292705, 0.00181897, 0.00431961, 0.00280193,
        0.00434469, 0.01122707, 0.0084267 , 0.00403249, 0.00303331,
        0.00312312, 0.00264579, 0.0018027 , 0.00309645, 0.00286396,
        0.00334714, 0.00186699, 0.00199899, 0.01245327, 0.00891122,
        0.00196551, 0.00215042, 0.00229516, 0.00410137, 0.002634  ,
        0.00173703, 0.00216887, 0.00275387, 0.00799387, 0.00723964,
        0.00232617, 0.00344004]),
 'param_criterion': masked_array(data=['gini', 'gini', 'gini', 'gini', 'gin
i', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'gini', 'gini', 'gini', 'gini',
                    'gini', 'gini', 'gini', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'entropy', 'entropy', 'entropy',
                    'entropy', 'entropy', 'log_loss', 'log_loss',
                    'log_loss', 'log_loss', 'log_loss', 'log_loss',
                    'log_loss', 'log_loss', 'log_loss', 'log_loss',
                    'log_loss', 'log_loss', 'log_loss', 'log_loss',
                    'log_loss', 'log_loss', 'log_loss', 'log_loss',
                    'log_loss', 'log_loss', 'log_loss', 'log_loss',
                    'log_loss', 'log_loss'],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value=np.str_('?'),
             dtype=object),
 'param_max_depth': masked_array(data=[2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 6,
6, 6, 6, 6, 6,
                    7, 7, 7, 7, 7, 7, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3,
                    6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 2, 2, 2, 2, 2, 2,
                    3, 3, 3, 3, 3, 3, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value=999999),
 'param_min_samples_split': masked_array(data=[3, 3, 4, 4, 5, 5, 3, 3, 4, 4,
5, 5, 3, 3, 4, 4, 5, 5,
```

```
                   3, 3, 4, 4, 5, 5, 3, 3, 4, 4, 5, 5, 3, 3, 4, 4, 5, 5,
                   3, 3, 4, 4, 5, 5, 3, 3, 4, 4, 5, 5, 3, 3, 4, 4, 5, 5,
                   3, 3, 4, 4, 5, 5, 3, 3, 4, 4, 5, 5, 3, 3, 4, 4, 5, 5],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value=999999),
 'param_splitter': masked_array(data=['best', 'random', 'best', 'random', 'be
st', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random',
                   'best', 'random', 'best', 'random', 'best', 'random'],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value=np.str_('?'),
            dtype=object),
 'params': [{'criterion': 'gini',
   'max_depth': 2,
   'min_samples_split': 3,
   'splitter': 'best'},
  {'criterion': 'gini',
   'max_depth': 2,
   'min_samples_split': 3,
   'splitter': 'random'},
  {'criterion': 'gini',
   'max_depth': 2,
   'min_samples_split': 4,
   'splitter': 'best'},
  {'criterion': 'gini',
   'max_depth': 2,
   'min_samples_split': 4,
   'splitter': 'random'},
  {'criterion': 'gini',
```

```
 'max_depth': 2,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 2,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_split': 5,
```

```
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'gini',
 'max_depth': 7,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 2,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 2,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 2,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 2,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 2,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 2,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 3,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'entropy',
```

```
 'max_depth': 3,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 3,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 3,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 3,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 3,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 6,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 6,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 6,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 6,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 6,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 6,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 7,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 7,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 7,
 'min_samples_split': 4,
```

```
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 7,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'entropy',
 'max_depth': 7,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'entropy',
 'max_depth': 7,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 2,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 2,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 2,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 2,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 2,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 2,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 3,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 3,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 3,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 3,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'log_loss',
```

```
 'max_depth': 3,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 3,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 6,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 6,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 6,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 6,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 6,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 6,
 'min_samples_split': 5,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 7,
 'min_samples_split': 3,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 7,
 'min_samples_split': 3,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 7,
 'min_samples_split': 4,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 7,
 'min_samples_split': 4,
 'splitter': 'random'},
{'criterion': 'log_loss',
 'max_depth': 7,
 'min_samples_split': 5,
 'splitter': 'best'},
{'criterion': 'log_loss',
 'max_depth': 7,
 'min_samples_split': 5,
```

```
   'splitter': 'random'}],
 'split0_test_score': array([0.97222222, 1.        , 0.97222222, 0.88888889,
0.97222222,
        1.        , 1.        , 0.91666667, 1.        , 1.        ,
        1.        , 0.97222222, 0.97222222, 0.97222222, 0.97222222,
        1.        , 0.97222222, 0.94444444, 0.97222222, 0.97222222,
        0.97222222, 0.97222222, 0.97222222, 1.        , 1.        ,
        0.80555556, 1.        , 0.86111111, 1.        , 0.88888889,
        0.94444444, 0.97222222, 0.94444444, 0.91666667, 0.94444444,
        1.        , 0.97222222, 1.        , 0.97222222, 1.        ,
        0.97222222, 0.97222222, 1.        , 0.97222222, 1.        ,
        1.        , 1.        , 0.97222222, 1.        , 1.        ,
        1.        , 0.91666667, 1.        , 0.94444444, 0.94444444,
        0.94444444, 0.94444444, 0.97222222, 0.94444444, 0.97222222,
        0.97222222, 0.88888889, 1.        , 1.        , 0.97222222,
        1.        , 1.        , 0.94444444, 0.97222222, 1.        ,
        0.97222222, 1.        ]),
 'split1_test_score': array([0.88888889, 0.83333333, 0.88888889, 0.91666667,
0.88888889,
        0.80555556, 0.88888889, 0.94444444, 0.88888889, 0.88888889,
        0.88888889, 0.80555556, 0.94444444, 0.91666667, 0.94444444,
        0.94444444, 0.94444444, 0.91666667, 0.91666667, 0.94444444,
        0.91666667, 0.97222222, 0.88888889, 1.        , 0.88888889,
        0.91666667, 0.88888889, 0.86111111, 0.88888889, 0.91666667,
        0.91666667, 0.97222222, 0.91666667, 0.86111111, 0.91666667,
        0.91666667, 0.88888889, 0.91666667, 0.88888889, 0.97222222,
        0.94444444, 0.97222222, 0.94444444, 0.88888889, 0.91666667,
        0.91666667, 0.91666667, 1.        , 0.88888889, 0.80555556,
        0.88888889, 0.94444444, 0.88888889, 0.88888889, 0.91666667,
        0.97222222, 0.91666667, 0.88888889, 0.91666667, 0.91666667,
        0.91666667, 0.88888889, 0.94444444, 0.91666667, 0.83333333,
        0.94444444, 0.88888889, 0.97222222, 0.88888889, 0.94444444,
        0.86111111, 0.94444444]),
 'split2_test_score': array([0.86111111, 0.86111111, 0.86111111, 0.86111111,
0.86111111,
        0.83333333, 0.88888889, 0.86111111, 0.88888889, 0.86111111,
        0.88888889, 0.80555556, 0.88888889, 0.91666667, 0.88888889,
        0.91666667, 0.88888889, 0.94444444, 0.88888889, 0.91666667,
        0.91666667, 0.91666667, 0.88888889, 0.88888889, 0.91666667,
        0.75      , 0.91666667, 0.86111111, 0.91666667, 0.83333333,
        0.83333333, 0.83333333, 0.83333333, 0.88888889, 0.83333333,
        0.88888889, 0.88888889, 0.86111111, 0.88888889, 0.86111111,
        0.88888889, 0.91666667, 0.88888889, 0.94444444, 0.88888889,
        0.88888889, 0.88888889, 0.88888889, 0.91666667, 0.80555556,
        0.91666667, 0.86111111, 0.91666667, 0.77777778, 0.83333333,
        0.91666667, 0.83333333, 0.83333333, 0.83333333, 0.88888889,
        0.88888889, 0.88888889, 0.88888889, 0.94444444, 0.88888889,
        0.88888889, 0.88888889, 0.88888889, 0.88888889, 0.88888889,
        0.88888889, 0.88888889]),
 'split3_test_score': array([0.91666667, 0.97222222, 0.91666667, 0.94444444,
0.91666667,
        0.86111111, 0.91666667, 0.88888889, 0.91666667, 0.97222222,
        0.91666667, 0.86111111, 0.97222222, 0.97222222, 0.97222222,
        0.94444444, 0.97222222, 0.94444444, 0.97222222, 0.91666667,
```

```
        0.94444444, 0.97222222, 0.97222222, 0.91666667, 0.88888889,
        0.88888889, 0.88888889, 0.88888889, 0.88888889, 0.91666667,
        0.94444444, 0.91666667, 0.94444444, 0.91666667, 0.94444444,
        0.86111111, 0.94444444, 1.        , 0.94444444, 0.97222222,
        0.94444444, 0.97222222, 0.94444444, 0.94444444, 0.94444444,
        0.94444444, 0.94444444, 0.94444444, 0.88888889, 0.83333333,
        0.88888889, 0.88888889, 0.88888889, 0.86111111, 0.94444444,
        0.86111111, 0.94444444, 0.88888889, 0.94444444, 0.91666667,
        0.94444444, 0.86111111, 0.94444444, 0.88888889, 0.94444444,
        0.91666667, 0.94444444, 0.94444444, 0.94444444, 0.88888889,
        0.97222222, 0.88888889]),
 'split4_test_score': array([0.91666667, 0.91666667, 0.91666667, 0.94444444,
0.91666667,
        0.86111111, 0.91666667, 0.91666667, 0.91666667, 0.86111111,
        0.91666667, 0.86111111, 0.94444444, 0.91666667, 0.97222222,
        0.91666667, 0.94444444, 0.91666667, 0.94444444, 0.97222222,
        0.94444444, 0.88888889, 0.97222222, 0.88888889, 0.94444444,
        0.94444444, 0.94444444, 0.83333333, 0.94444444, 0.86111111,
        0.91666667, 0.88888889, 0.91666667, 0.94444444, 0.91666667,
        0.91666667, 0.97222222, 0.91666667, 0.94444444, 0.97222222,
        0.94444444, 0.91666667, 0.94444444, 0.88888889, 0.97222222,
        0.97222222, 0.97222222, 0.91666667, 0.94444444, 0.88888889,
        0.94444444, 0.86111111, 0.94444444, 0.88888889, 0.91666667,
        0.91666667, 0.91666667, 0.91666667, 0.91666667, 0.80555556,
        0.97222222, 0.91666667, 0.94444444, 0.88888889, 0.97222222,
        0.94444444, 0.97222222, 0.97222222, 0.94444444, 0.94444444,
        0.94444444, 0.97222222]),
 'split5_test_score': array([0.91666667, 0.91666667, 0.91666667, 0.77777778,
0.91666667,
        0.94444444, 0.91666667, 0.88888889, 0.91666667, 0.91666667,
        0.91666667, 0.91666667, 0.91666667, 0.86111111, 0.94444444,
        0.88888889, 0.91666667, 0.94444444, 0.91666667, 0.88888889,
        0.91666667, 0.91666667, 0.94444444, 0.94444444, 0.94444444,
        0.88888889, 0.94444444, 0.88888889, 0.94444444, 0.94444444,
        0.91666667, 0.91666667, 0.91666667, 0.94444444, 0.91666667,
        0.94444444, 0.91666667, 0.91666667, 0.91666667, 0.91666667,
        0.91666667, 0.97222222, 0.91666667, 0.94444444, 0.91666667,
        1.        , 0.91666667, 0.97222222, 0.94444444, 0.91666667,
        0.94444444, 0.80555556, 0.94444444, 0.86111111, 0.91666667,
        0.91666667, 0.91666667, 0.86111111, 0.91666667, 0.88888889,
        0.91666667, 0.94444444, 0.91666667, 0.88888889, 0.91666667,
        0.97222222, 0.91666667, 0.94444444, 0.91666667, 0.88888889,
        0.91666667, 0.94444444]),
 'split6_test_score': array([0.94444444, 0.97222222, 0.94444444, 0.97222222,
0.94444444,
        0.88888889, 1.        , 0.97222222, 1.        , 0.97222222,
        1.        , 0.94444444, 1.        , 0.97222222, 1.        ,
        0.97222222, 1.        , 0.91666667, 1.        , 0.97222222,
        1.        , 0.91666667, 1.        , 1.        , 0.91666667,
        0.80555556, 0.91666667, 0.88888889, 0.91666667, 0.88888889,
        0.94444444, 0.88888889, 0.94444444, 0.91666667, 0.94444444,
        0.91666667, 0.97222222, 0.94444444, 0.97222222, 0.94444444,
        0.97222222, 0.94444444, 0.97222222, 0.94444444, 0.97222222,
        0.97222222, 0.97222222, 0.97222222, 0.91666667, 0.88888889,
```

```
        0.91666667, 0.86111111, 0.91666667, 0.88888889, 0.94444444,
        0.97222222, 0.94444444, 0.94444444, 0.94444444, 0.94444444,
        0.97222222, 0.94444444, 0.97222222, 0.94444444, 0.97222222,
        0.94444444, 0.97222222, 0.97222222, 0.97222222, 1.        ,
        0.97222222, 0.97222222]),
 'split7_test_score': array([0.91666667, 1.        , 0.91666667, 0.94444444,
0.91666667,
        0.83333333, 1.        , 1.        , 1.        , 0.94444444,
        1.        , 0.97222222, 0.97222222, 0.97222222, 0.97222222,
        0.91666667, 1.        , 0.97222222, 0.97222222, 0.88888889,
        1.        , 0.94444444, 1.        , 0.91666667, 1.        ,
        0.97222222, 1.        , 1.        , 1.        , 0.94444444,
        0.97222222, 0.97222222, 0.97222222, 0.94444444, 0.97222222,
        1.        , 0.91666667, 0.94444444, 0.97222222, 0.91666667,
        0.97222222, 0.94444444, 0.91666667, 0.97222222, 0.97222222,
        0.94444444, 0.94444444, 0.94444444, 1.        , 0.97222222,
        1.        , 0.88888889, 1.        , 1.        , 0.97222222,
        0.97222222, 0.97222222, 0.91666667, 0.97222222, 0.94444444,
        0.94444444, 0.97222222, 0.91666667, 0.91666667, 0.97222222,
        0.91666667, 0.91666667, 0.94444444, 0.94444444, 1.        ,
        0.94444444, 1.        ]),
 'split8_test_score': array([0.97222222, 0.91666667, 0.97222222, 0.94444444,
0.97222222,
        0.94444444, 0.97222222, 0.91666667, 0.97222222, 0.97222222,
        0.97222222, 0.94444444, 0.97222222, 0.97222222, 0.97222222,
        0.94444444, 0.97222222, 0.94444444, 0.97222222, 0.97222222,
        0.97222222, 0.91666667, 0.94444444, 0.97222222, 0.94444444,
        0.97222222, 0.94444444, 0.88888889, 0.94444444, 0.91666667,
        0.94444444, 0.91666667, 0.94444444, 0.97222222, 0.94444444,
        0.97222222, 0.97222222, 0.97222222, 0.97222222, 0.94444444,
        0.97222222, 0.97222222, 0.97222222, 1.        , 0.97222222,
        0.97222222, 0.97222222, 0.94444444, 0.94444444, 0.91666667,
        0.94444444, 0.91666667, 0.94444444, 0.97222222, 0.94444444,
        0.91666667, 0.94444444, 0.94444444, 0.94444444, 0.94444444,
        0.94444444, 0.97222222, 0.97222222, 0.97222222, 0.97222222,
        1.        , 0.97222222, 0.97222222, 0.97222222, 0.97222222,
        0.97222222, 0.94444444]),
 'split9_test_score': array([0.94444444, 0.94444444, 0.94444444, 0.91666667,
0.94444444,
        0.97222222, 0.97222222, 0.94444444, 0.97222222, 0.94444444,
        0.97222222, 0.94444444, 0.97222222, 1.        , 0.97222222,
        0.97222222, 0.97222222, 0.94444444, 0.97222222, 0.91666667,
        0.97222222, 0.91666667, 0.97222222, 0.97222222, 0.97222222,
        0.94444444, 0.97222222, 0.97222222, 0.97222222, 0.94444444,
        0.94444444, 0.97222222, 0.94444444, 0.94444444, 0.94444444,
        0.94444444, 0.97222222, 1.        , 0.97222222, 0.94444444,
        0.94444444, 0.94444444, 0.97222222, 0.94444444, 0.97222222,
        0.94444444, 0.97222222, 0.94444444, 0.97222222, 0.94444444,
        0.97222222, 0.91666667, 0.97222222, 0.88888889, 0.94444444,
        0.94444444, 0.94444444, 0.91666667, 0.94444444, 0.97222222,
        0.94444444, 0.94444444, 0.97222222, 0.97222222, 0.97222222,
        0.97222222, 0.97222222, 0.97222222, 0.97222222, 0.94444444,
        0.97222222, 0.94444444]),
 'split10_test_score': array([0.97222222, 0.88888889, 0.97222222, 0.94444444,
```

```
0.97222222,
        0.91666667, 0.97222222, 0.94444444, 1.        , 0.94444444,
        1.        , 0.94444444, 1.        , 0.97222222, 1.        ,
        0.97222222, 1.        , 1.        , 1.        , 0.97222222,
        1.        , 0.97222222, 1.        , 1.        , 0.94444444,
        0.88888889, 0.94444444, 0.91666667, 0.94444444, 0.94444444,
        0.97222222, 0.94444444, 0.97222222, 0.97222222, 0.97222222,
        0.91666667, 0.97222222, 1.        , 0.97222222, 0.97222222,
        1.        , 1.        , 0.97222222, 1.        , 0.97222222,
        0.97222222, 0.97222222, 1.        , 0.94444444, 0.94444444,
        0.94444444, 0.94444444, 0.94444444, 0.97222222, 0.97222222,
        0.91666667, 0.97222222, 0.94444444, 0.97222222, 0.97222222,
        1.        , 0.94444444, 0.97222222, 0.97222222, 0.97222222,
        0.97222222, 0.97222222, 0.97222222, 1.        , 1.        ,
        0.97222222, 0.97222222]),
 'split11_test_score': array([0.91666667, 0.88888889, 0.91666667, 0.86111111,
0.91666667,
        0.91666667, 0.94444444, 0.88888889, 0.94444444, 0.88888889,
        0.91666667, 0.88888889, 0.97222222, 0.94444444, 0.97222222,
        0.88888889, 0.97222222, 0.88888889, 1.        , 0.94444444,
        1.        , 0.97222222, 0.97222222, 0.97222222, 0.91666667,
        0.94444444, 0.91666667, 0.88888889, 0.91666667, 0.86111111,
        0.91666667, 0.91666667, 0.91666667, 0.97222222, 0.91666667,
        0.97222222, 0.94444444, 0.97222222, 0.94444444, 0.94444444,
        0.94444444, 0.97222222, 0.97222222, 0.94444444, 0.94444444,
        0.97222222, 0.94444444, 0.97222222, 0.91666667, 0.86111111,
        0.91666667, 0.86111111, 0.91666667, 0.91666667, 0.91666667,
        0.97222222, 0.91666667, 0.94444444, 0.91666667, 0.97222222,
        0.91666667, 0.94444444, 0.94444444, 1.        , 0.91666667,
        0.97222222, 0.97222222, 0.97222222, 0.91666667, 0.91666667,
        0.94444444, 0.97222222]),
 'split12_test_score': array([0.97222222, 0.97222222, 0.97222222, 0.91666667,
0.97222222,
        0.94444444, 0.97222222, 0.88888889, 0.97222222, 0.91666667,
        0.97222222, 0.97222222, 1.        , 0.97222222, 0.97222222,
        0.88888889, 0.97222222, 0.94444444, 1.        , 0.97222222,
        1.        , 1.        , 0.97222222, 0.97222222, 0.94444444,
        0.86111111, 0.94444444, 0.83333333, 0.94444444, 0.94444444,
        0.97222222, 0.94444444, 0.97222222, 0.94444444, 0.97222222,
        0.94444444, 0.97222222, 0.94444444, 0.97222222, 1.        ,
        0.97222222, 1.        , 0.97222222, 1.        , 0.97222222,
        0.94444444, 0.97222222, 1.        , 0.94444444, 0.91666667,
        0.94444444, 0.91666667, 0.94444444, 0.94444444, 0.97222222,
        0.94444444, 0.97222222, 0.91666667, 0.97222222, 1.        ,
        0.97222222, 0.97222222, 0.97222222, 1.        , 0.97222222,
        1.        , 0.97222222, 1.        , 0.97222222, 1.        ,
        0.97222222, 0.94444444]),
 'split13_test_score': array([0.94444444, 0.97222222, 0.94444444, 0.97222222,
0.94444444,
        0.94444444, 0.97222222, 0.97222222, 0.97222222, 0.91666667,
        0.97222222, 0.94444444, 1.        , 0.94444444, 0.97222222,
        0.94444444, 1.        , 0.97222222, 1.        , 1.        ,
        1.        , 0.97222222, 1.        , 0.97222222, 0.91666667,
        0.88888889, 0.91666667, 0.88888889, 0.91666667, 0.94444444,
```

```
        0.91666667, 0.91666667, 0.94444444, 0.94444444, 0.94444444,
        0.94444444, 0.97222222, 0.97222222, 1.        , 0.91666667,
        1.        , 0.94444444, 1.        , 0.97222222, 0.97222222,
        0.94444444, 1.        , 0.97222222, 0.91666667, 0.88888889,
        0.91666667, 0.91666667, 0.91666667, 0.91666667, 0.94444444,
        0.94444444, 0.91666667, 0.91666667, 0.91666667, 0.88888889,
        0.97222222, 0.97222222, 0.97222222, 0.97222222, 0.94444444,
        0.94444444, 0.97222222, 1.        , 0.97222222, 0.97222222,
        1.        , 0.86111111]),
 'split14_test_score': array([0.94285714, 0.88571429, 0.94285714, 0.71428571,
0.94285714,
        0.94285714, 0.91428571, 0.94285714, 0.91428571, 0.91428571,
        0.91428571, 0.94285714, 0.97142857, 1.        , 0.91428571,
        1.        , 0.91428571, 0.94285714, 0.97142857, 0.94285714,
        0.94285714, 0.94285714, 0.94285714, 0.91428571, 0.88571429,
        0.91428571, 0.88571429, 0.82857143, 0.88571429, 0.91428571,
        0.91428571, 0.94285714, 0.91428571, 0.91428571, 0.91428571,
        0.82857143, 0.97142857, 0.94285714, 0.94285714, 0.94285714,
        0.94285714, 0.91428571, 0.97142857, 0.94285714, 0.97142857,
        0.91428571, 0.94285714, 0.94285714, 0.88571429, 0.88571429,
        0.88571429, 0.91428571, 0.88571429, 0.91428571, 0.91428571,
        0.8       , 0.91428571, 0.85714286, 0.91428571, 0.94285714,
        0.97142857, 0.97142857, 1.        , 0.91428571, 0.97142857,
        0.94285714, 0.94285714, 0.97142857, 0.97142857, 0.97142857,
        0.94285714, 0.88571429]),
 'split15_test_score': array([0.94285714, 0.94285714, 0.94285714, 0.94285714,
0.94285714,
        0.94285714, 0.97142857, 0.97142857, 0.97142857, 0.94285714,
        0.97142857, 1.        , 1.        , 1.        , 1.        ,
        0.97142857, 1.        , 0.88571429, 0.97142857, 1.        ,
        1.        , 1.        , 1.        , 0.97142857, 0.91428571,
        0.82857143, 0.91428571, 0.94285714, 0.91428571, 0.97142857,
        0.91428571, 0.77142857, 0.91428571, 0.94285714, 0.91428571,
        0.94285714, 0.91428571, 1.        , 0.94285714, 0.91428571,
        0.91428571, 1.        , 0.94285714, 1.        , 0.91428571,
        0.97142857, 0.94285714, 0.97142857, 0.91428571, 0.97142857,
        0.91428571, 0.94285714, 0.91428571, 0.97142857, 0.91428571,
        0.94285714, 0.91428571, 0.97142857, 0.91428571, 1.        ,
        0.94285714, 0.94285714, 0.94285714, 1.        , 0.91428571,
        1.        , 0.94285714, 1.        , 0.91428571, 1.        ,
        0.94285714, 1.        ]),
 'split16_test_score': array([0.88571429, 0.91428571, 0.88571429, 0.91428571,
0.88571429,
        0.85714286, 0.91428571, 0.88571429, 0.91428571, 0.8       ,
        0.91428571, 0.82857143, 1.        , 0.91428571, 1.        ,
        0.91428571, 1.        , 0.94285714, 1.        , 0.94285714,
        1.        , 0.91428571, 1.        , 0.91428571, 0.85714286,
        0.85714286, 0.85714286, 0.88571429, 0.85714286, 0.82857143,
        0.91428571, 0.85714286, 0.91428571, 0.91428571, 0.91428571,
        0.85714286, 0.94285714, 0.91428571, 0.94285714, 0.94285714,
        0.94285714, 0.91428571, 0.94285714, 0.94285714, 0.94285714,
        0.94285714, 0.94285714, 0.94285714, 0.85714286, 0.85714286,
        0.85714286, 0.88571429, 0.85714286, 0.88571429, 0.91428571,
        0.82857143, 0.91428571, 0.88571429, 0.91428571, 0.82857143,
```

```
        0.94285714, 0.91428571, 0.94285714, 0.97142857, 0.94285714,
        0.94285714, 0.94285714, 1.        , 0.94285714, 0.88571429,
        0.94285714, 0.94285714]),
 'split17_test_score': array([0.91428571, 0.94285714, 0.91428571, 0.97142857,
0.91428571,
        0.91428571, 0.85714286, 0.91428571, 0.85714286, 0.97142857,
        0.85714286, 0.85714286, 0.88571429, 0.91428571, 0.88571429,
        0.97142857, 0.88571429, 1.        , 0.88571429, 1.        ,
        0.88571429, 0.94285714, 0.88571429, 1.        , 0.88571429,
        0.8       , 0.88571429, 0.77142857, 0.88571429, 0.85714286,
        0.94285714, 0.91428571, 0.94285714, 0.97142857, 0.94285714,
        0.97142857, 1.        , 0.97142857, 1.        , 0.97142857,
        1.        , 1.        , 1.        , 0.97142857, 1.        ,
        1.        , 1.        , 0.94285714, 0.88571429, 0.94285714,
        0.88571429, 0.8       , 0.88571429, 0.91428571, 0.94285714,
        0.91428571, 0.94285714, 0.88571429, 0.94285714, 0.94285714,
        1.        , 0.94285714, 1.        , 0.97142857, 1.        ,
        0.94285714, 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        ]),
 'split18_test_score': array([0.94285714, 0.91428571, 0.94285714, 0.88571429,
0.94285714,
        0.85714286, 0.97142857, 0.94285714, 0.97142857, 0.82857143,
        0.97142857, 0.88571429, 0.94285714, 0.94285714, 0.97142857,
        0.94285714, 0.97142857, 0.94285714, 0.94285714, 0.94285714,
        0.97142857, 0.94285714, 0.97142857, 0.91428571, 0.88571429,
        0.88571429, 0.88571429, 0.88571429, 0.88571429, 0.85714286,
        0.88571429, 0.91428571, 0.88571429, 0.91428571, 0.88571429,
        0.91428571, 0.91428571, 0.94285714, 0.97142857, 0.94285714,
        0.97142857, 0.85714286, 0.97142857, 0.91428571, 0.94285714,
        0.97142857, 0.97142857, 1.        , 0.88571429, 0.94285714,
        0.88571429, 0.8       , 0.88571429, 0.88571429, 0.88571429,
        0.88571429, 0.88571429, 0.85714286, 0.88571429, 0.88571429,
        0.94285714, 0.91428571, 0.94285714, 0.91428571, 0.94285714,
        0.91428571, 0.94285714, 1.        , 0.97142857, 0.88571429,
        0.94285714, 0.91428571]),
 'split19_test_score': array([0.97142857, 0.97142857, 0.97142857, 0.97142857,
0.97142857,
        1.        , 1.        , 1.        , 1.        , 0.94285714,
        1.        , 0.94285714, 0.94285714, 0.97142857, 0.94285714,
        0.97142857, 0.94285714, 1.        , 0.94285714, 0.97142857,
        0.94285714, 1.        , 0.94285714, 0.97142857, 1.        ,
        0.97142857, 1.        , 1.        , 1.        , 0.97142857,
        1.        , 1.        , 1.        , 0.97142857, 1.        ,
        1.        , 1.        , 1.        , 1.        , 0.94285714,
        1.        , 0.97142857, 1.        , 1.        , 0.91428571,
        0.97142857, 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 0.94285714, 1.        ,
        0.97142857, 1.        , 0.91428571, 1.        , 0.97142857,
        1.        , 0.97142857, 1.        , 1.        , 0.91428571,
        1.        , 1.        , 1.        , 1.        , 0.94285714,
        1.        , 0.97142857]),
 'mean_test_score': array([0.93277778, 0.93134921, 0.93277778, 0.91027778,
0.93277778,
        0.90904762, 0.94809524, 0.93007937, 0.94948413, 0.92      ,
```

```
        0.94809524, 0.91174603, 0.96353175, 0.95242063, 0.96210317,
        0.94412698, 0.96210317, 0.94543651, 0.96210317, 0.9525    ,
        0.96492063, 0.94964286, 0.96214286, 0.95511905, 0.92837302,
        0.88646825, 0.92837302, 0.8893254 , 0.92837302, 0.9075    ,
        0.93134921, 0.91861111, 0.9327381 , 0.93559524, 0.9327381 ,
        0.93265873, 0.95103175, 0.95662698, 0.95666667, 0.94674603,
        0.95801587, 0.95507937, 0.96087302, 0.95662698, 0.95373016,
        0.95801587, 0.95944444, 0.96222222, 0.92837302, 0.90916667,
        0.92837302, 0.89214286, 0.92837302, 0.9118254 , 0.9327381 ,
        0.92269841, 0.93134921, 0.90884921, 0.93134921, 0.92996032,
        0.95388889, 0.93563492, 0.95809524, 0.95246032, 0.94539683,
        0.9565873 , 0.9552381 , 0.97218254, 0.95388889, 0.95234127,
        0.95384921, 0.94821429]),
 'std_test_score': array([0.03110271, 0.04450216, 0.03110271, 0.06482843, 0.0
3110271,
        0.05501975, 0.04202484, 0.03891391, 0.0432407 , 0.05087271,
        0.04382246, 0.05694167, 0.03457028, 0.03549445, 0.03258019,
        0.03402662, 0.03595761, 0.03133753, 0.03583303, 0.03302605,
        0.03543862, 0.0325653 , 0.03690442, 0.03802115, 0.04062211,
        0.06235716, 0.04062211, 0.05534397, 0.04062211, 0.04385197,
        0.03489673, 0.05223975, 0.03483745, 0.02939539, 0.03483745,
        0.04681347, 0.03297244, 0.03792561, 0.03110706, 0.03167045,
        0.03001732, 0.03634921, 0.0297223 , 0.03361116, 0.02983546,
        0.02862528, 0.0298669 , 0.02963348, 0.04062211, 0.05625942,
        0.04062211, 0.05077684, 0.04062211, 0.04916979, 0.03483745,
        0.04680347, 0.03489673, 0.03777   , 0.03489673, 0.05136391,
        0.02955526, 0.03300757, 0.02990453, 0.03956721, 0.03792671,
        0.03249329, 0.03227086, 0.02777832, 0.03330866, 0.04447524,
        0.03442677, 0.04083785]),
 'rank_test_score': array([41, 48, 41, 65, 41, 67, 33, 52, 31, 61, 33, 64,
3, 27,  7, 38,  7,
        36,  6, 25,  2, 30,  5, 19, 54, 72, 54, 71, 54, 69, 48, 62, 44, 40,
        44, 47, 29, 15, 14, 35, 12, 20,  9, 15, 24, 12, 10,  4, 54, 66, 54,
        70, 54, 63, 44, 60, 48, 68, 48, 53, 21, 39, 11, 26, 37, 17, 18,  1,
        21, 28, 23, 32], dtype=int32)}
```

In [219…

```
pd.DataFrame(gridcv.cv_results_)
```

Out[219…

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criteric
0	0.060370	0.006294	0.018963	0.003095	gi
1	0.028462	0.003926	0.015853	0.003265	gi
2	0.042432	0.006941	0.013004	0.003768	gi
3	0.010522	0.002876	0.006261	0.001861	gi
4	0.074679	0.031442	0.022706	0.012478	gi
...	...	...	...	...	
67	0.026668	0.004279	0.013602	0.002754	log_lo
68	0.106535	0.042274	0.020950	0.007994	log_lo
69	0.043990	0.016206	0.017787	0.007240	log_lo
70	0.077987	0.006209	0.012050	0.002326	log_lo
71	0.020842	0.006509	0.009672	0.003440	log_lo

72 rows × 32 columns

In [ ]: