Machine Learning Lab 5 - Predicting Breast Cancer - II

Submitted By
Name: Harsha KG
Register Number: 19112005
Class: **5 BSc Data Science**

Lab Overview

Objectives

Compare and understand the difference between decision tree and random forest classification result with respect to Breast cancer Dataset

Problem Definition

Compare and contrast the different evaluation metrices, effect of classification with respect to change in training-test split, random state, pre-processing labels, hyper parameters,alogrithm parameter of decision tree and random forest and interpret the change in result using visualization.

Approach

Imported the Dataset using required libraries from Kaggle(https://www.kaggle.com/uciml/breast-cancer-wisconsin-data (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data)) to python.Did some pre-processing technique and then build the model using Decision tree and Random forest and compare the difference in the classification metrics and other parameters and after than did hyperparameter tuning for checking the model which gives the highest accuaracy with all the parameter. At the end did some visualization on the results.

Sections

1. Lab Overview
2. Imported the Required Libraraies
3. Load the Dataset
4. Basic Inference from Data (Data Wrangling)
   A. Finding the dimension of dataset
   B. Getting the Concise summary of Dataframe
   C. Checking the no of null values
   D. Finding the unique values of Dependent varaiable and counting them
   E. Drop unnamed columns
   F. Description of Datset.
   G. Getting to view the correlation on our data set
   H. Dividing the Columns into Dependent(Y) and Independent One(X)
   I. Plotting target column
   J. Pairplot
   K.
5. Train-Test Split
6. Using DecisionTreeClassifier of tree class to use Decision Tree Algorithm
   A. Classification Report
   B. Confusion Matrix
   C. Finding the Accuracy by changing the Parameter
   D. Histogram of Accuracy
   E. HyperParameter Tuning
7. Random Forest
   A. Classification Report
   B. Confusion Matrix
   C. Finding the Accuracy by changing the Parameter
   D. Histogram of Accuracy
   E. HyperParameter Tuning

1. Comparing models before and after Parameter Tuning
2. Conclusion

References

1. https://www.kaggle.com/uciml/breast-cancer-wisconsin-data (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data)
2. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)
3. https://scikit-learn.org/stable/modules/tree.html (https://scikit-learn.org/stable/modules/tree.html)

About The Datset

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

diagnosis-The diagnosis of breast tissues (M = malignant, B = benign)

radius_mean-mean of distances from center to points on the perimeter

texture_mean-standard deviation of gray-scale values

perimeter_mean-mean size of the core tumor

area_mean

smoothness_mean-mean of local variation in radius lengths

compactness_mean-mean of perimeter^2 / area - 1.0

concavity_mean-mean of severity of concave portions of the contour

concave points_mean-mean for number of concave portions of the contour

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features.

Importing Required Libraries

In [1]:
```
# Importing the Libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
import hvplot.pandas

from sklearn.model_selection import train_test_split

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

Loading the Dataset

In [2]:
```
BC = pd.read_csv("Breast_Cancer.csv")
```

In [3]:
```
BC.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280

5 rows × 33 columns

Basic Inference from Data (Data Wrangling)

Dimension of Dataset

In [4]:
```
BC.shape
```

Out[4]: (569, 33)

Getting the Concise summary of Dataframe

In [5]:
```
BC.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

From this we can understand that total of 31 columns are of float type and remaining one are in int64 type.

Checking the no of null values

In [6]:
```
BC.isnull().sum()
BC.isna().sum()
```

Out[6]:
```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

Finding the unique values of Dependent varaiable and counting them

In [7]:
```
classes = pd.unique(BC['diagnosis'])
print(classes)
```

```
['M' 'B']
```

In [8]:
```
BC['diagnosis'].value_counts()
```

Out[8]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

Drop unnamed column

In [9]:
```

BC.drop(['Unnamed: 32'],axis=1,inplace=True)
```

Description of Datset

In [10]:
```
#Getting a statistical decription of our data
BC.describe()
```

Out[10]:

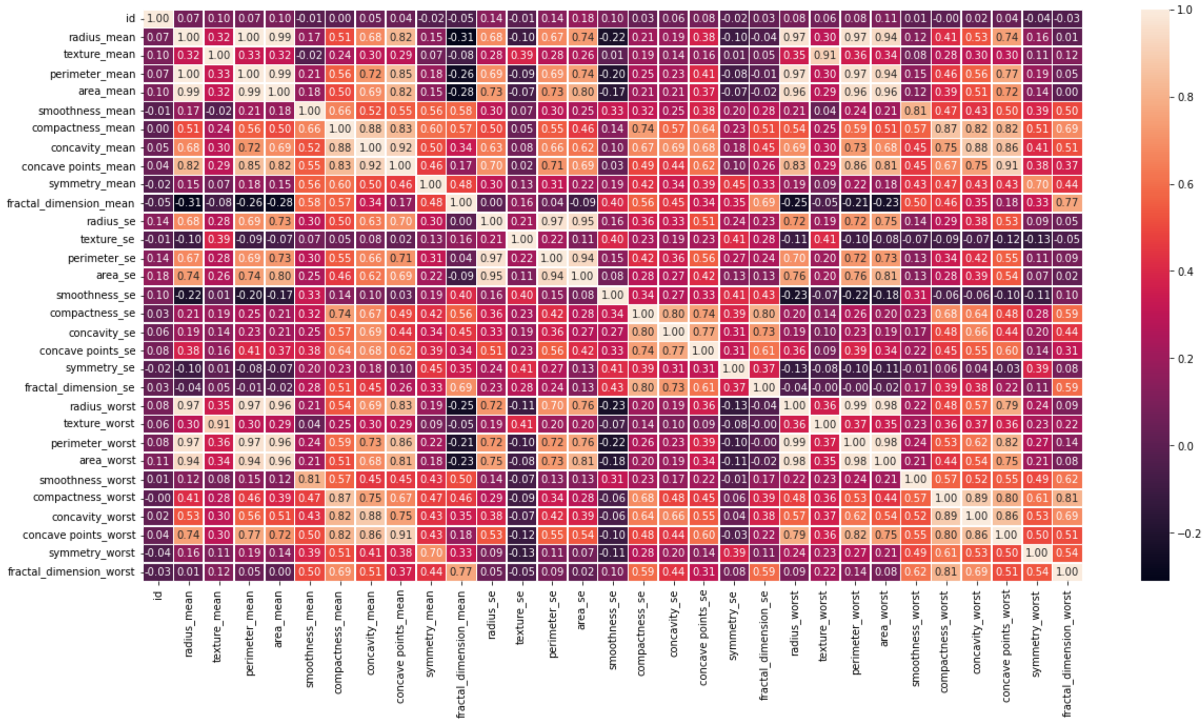
	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	c
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	

8 rows × 31 columns

Getting to view the correlation on our data set

In [11]:
```
plt.figure(figsize=(20,10))
sns.heatmap(BC.corr(),annot=True, fmt=".2f",annot_kws={"size":10},linewidths=.7)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x223ff3a7b80>



From our correlation matrix where there is a large positive near to 1.0 it indicates a strong positive correlation.

Where there is a negative value near to -1.0 it indicates a strong negative correlation that means the value of one variable will decreases with the increasing/decreasing of the other.

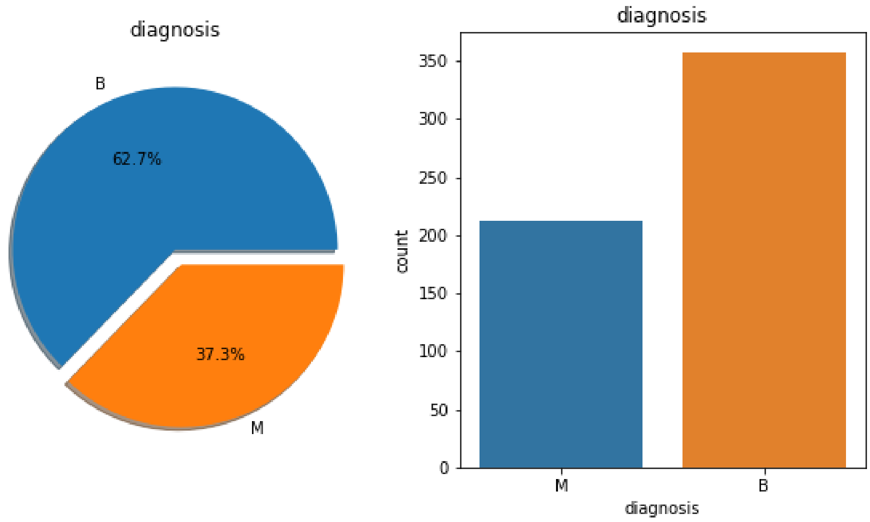
While a value near to 0 whether positive or negative indicates the absence of any correlation between the two variables they are independent of each other.

Dividing the Columns into Dependent(Y) and Independent One(X)

In [12]:
```
Y = BC['diagnosis']
X = BC.drop(['diagnosis'], axis=1)
```

Plotting target column

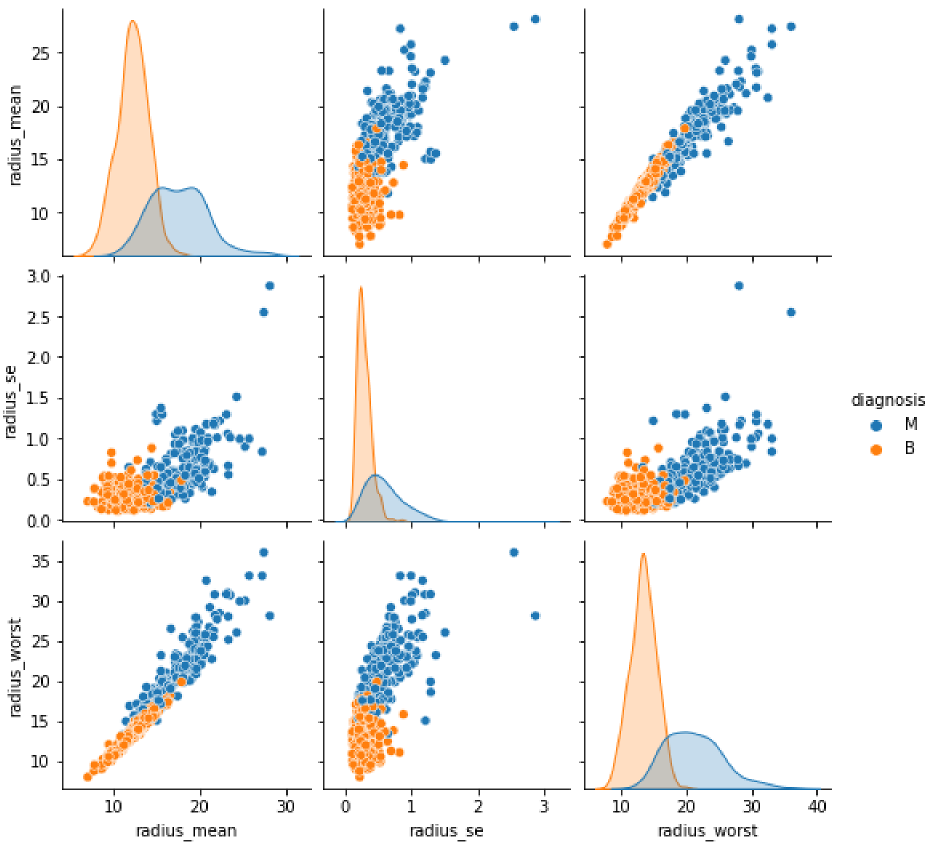
In [13]:
```
f,ax=plt.subplots(1,2,figsize=(10,5))
BC['diagnosis'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('diagnosis')
ax[0].set_ylabel('')
sns.countplot('diagnosis',data=BC,ax=ax[1])
ax[1].set_title('diagnosis')
plt.show()
```



Pairplot

In [14]:
```
sns.pairplot(BC[['radius_mean','radius_se','radius_worst','diagnosis']],hue='diagnosis')
```

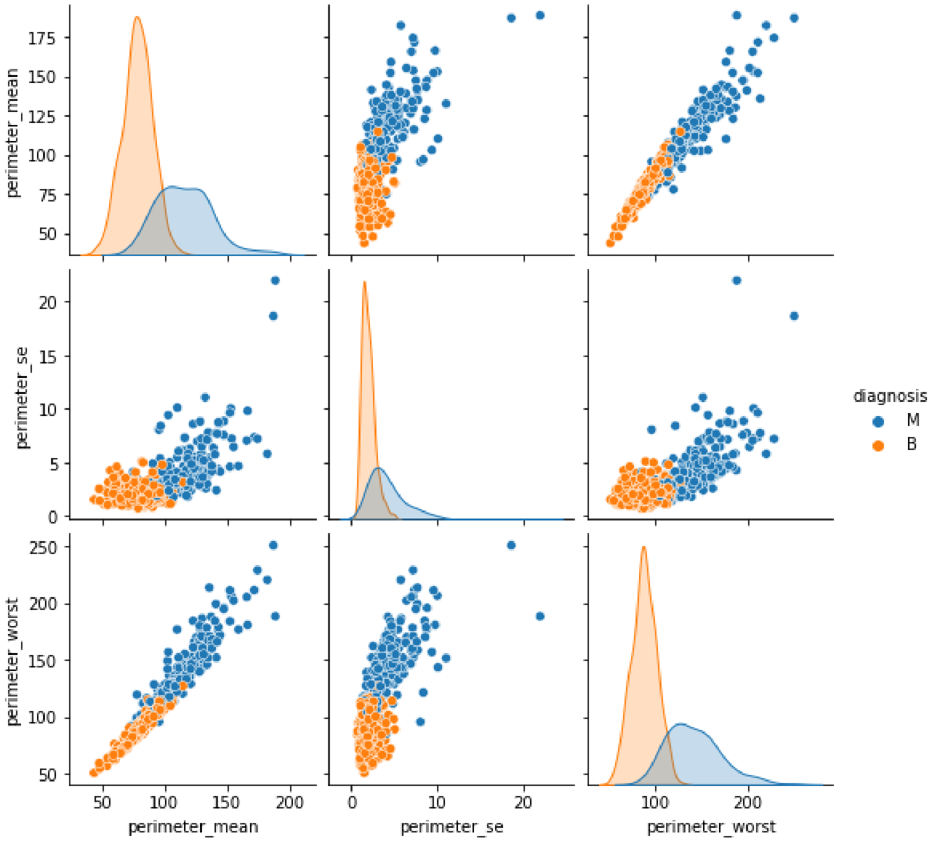
Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x223d6f729a0>
```



Features related to perimeter in relation to the diagnosis type

In [15]:
```
sns.pairplot(BC[['perimeter_mean','perimeter_se','perimeter_worst','diagnosis']],hue='diagnosis')
```

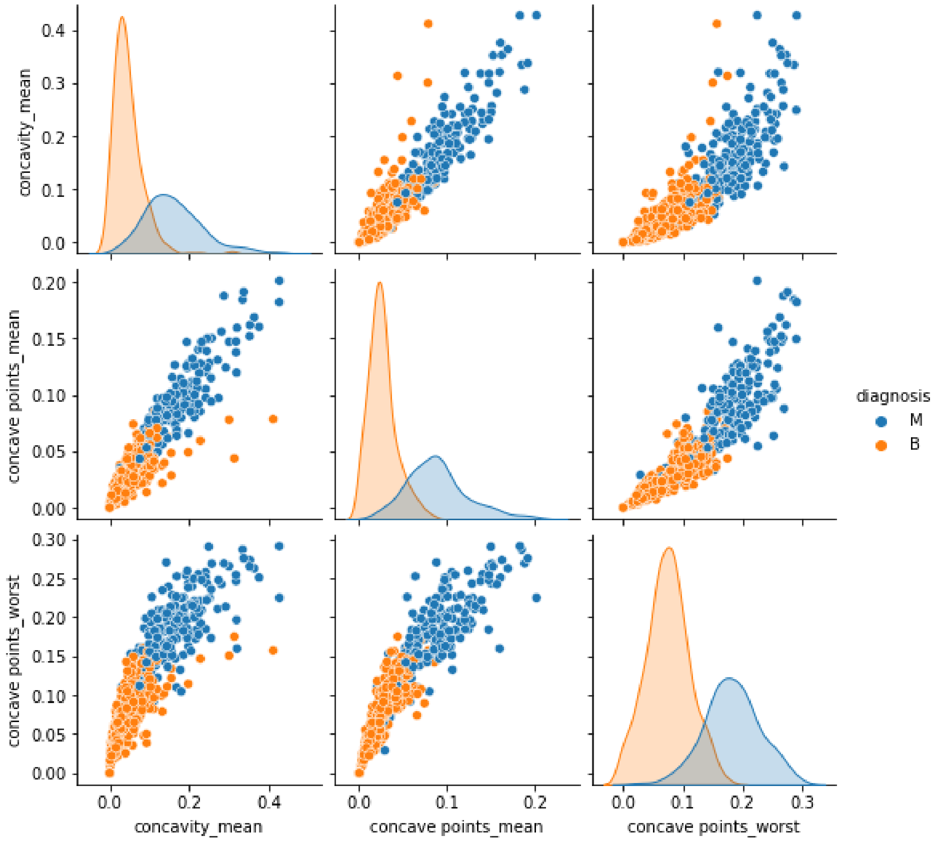
Out[15]: <seaborn.axisgrid.PairGrid at 0x223d955a5b0>



Features related to concave and concativity

In [16]:
```
sns.pairplot(BC[['concavity_mean','concave points_mean','concave points_worst','diagnosis']],hue=
'diagnosis')
```

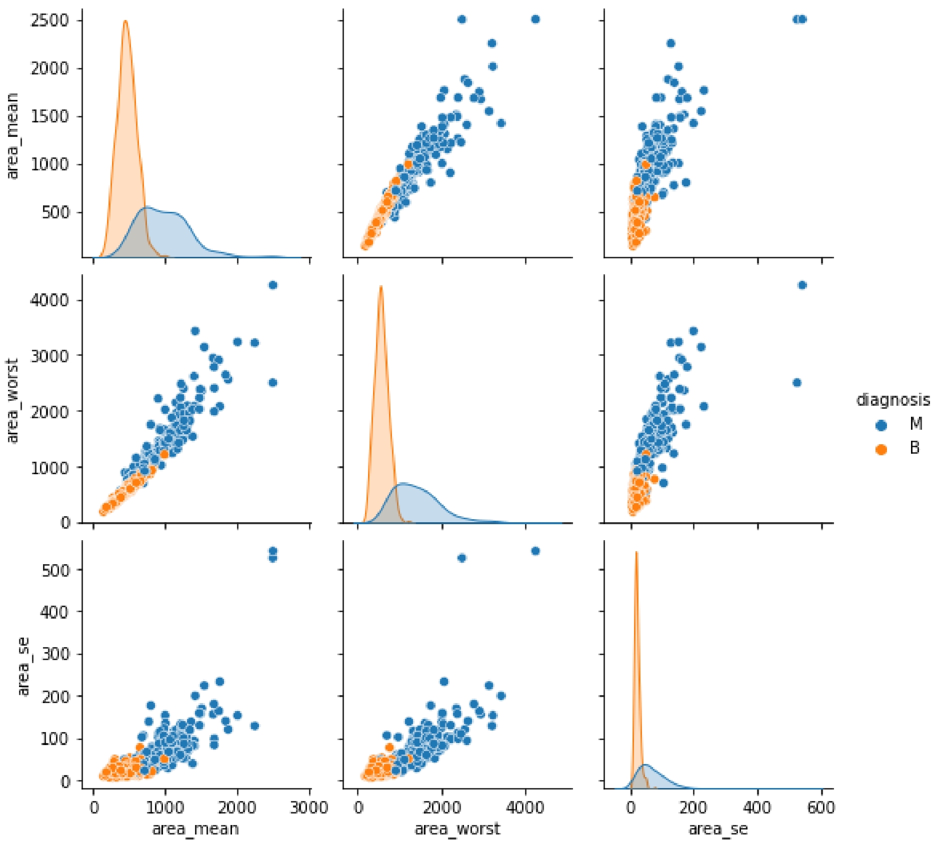
Out[16]: <seaborn.axisgrid.PairGrid at 0x223d9ad8b20>



Features related to area in relation to the diagnosis type

In [17]:
```
sns.pairplot(BC[['area_mean','area_worst','area_se','diagnosis']],hue='diagnosis')
```

Out[17]: <seaborn.axisgrid.PairGrid at 0x223d950c190>



Train-Test Split

In [21]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = .30, random_state = 0)
```

In [22]:
```
# print(X_train.shape);
print(Y_train.shape);
print("\r\n");
print(X_test.shape);
print(Y_test.shape);
```

```
(398,)


(171, 31)
(171,)
```

Using DecisionTreeClassifier of tree class to use Decision Tree Algorithm

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, Y_train)
```

Out[23]: DecisionTreeClassifier()

In [24]:
```
y_pred = classifier.predict(X_test)
```

In [25]:
```
y_pred[0:5]
```

Out[25]: array(['M', 'B', 'B', 'B', 'B'], dtype=object)

In [26]:
```
y_prob = classifier.predict_proba(X_test)
```

In [27]:
```
y_prob[0:5]
```

Out[27]:
```
array([[0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.]])
```

In [28]:
```
from sklearn.metrics import accuracy_score
acc_score1 = accuracy_score(Y_test, y_pred)
print(acc_score1)
```

```
0.9298245614035088
```

Classification Report

In [29]:
```
from sklearn.metrics import classification_report
class_report = classification_report(Y_test, y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

           B       0.96      0.93      0.94       108
           M       0.88      0.94      0.91        63

    accuracy                           0.93       171
   macro avg       0.92      0.93      0.93       171
weighted avg       0.93      0.93      0.93       171
```

Confusion Matrix

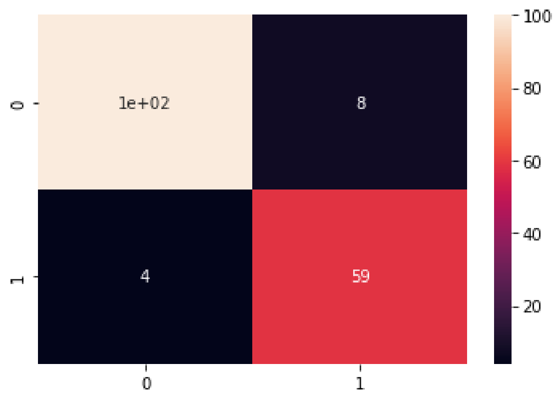
In [30]:
```
from sklearn.metrics import confusion_matrix
Cm = confusion_matrix(Y_test, y_pred)
```

In [31]:
```
Cm
```

Out[31]:
```
array([[100,   8],
       [  4,  59]], dtype=int64)
```

In [32]:
```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(Y_test, y_pred)
dataframe_conf_matrix = conf_matrix
sns.heatmap(dataframe_conf_matrix, annot=True)
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x223dbb41fd0>
```



In [33]:
```
Accuracy = (Cm[0][0] + Cm[1][1]) / (Cm[0][0] + Cm[1][1] + Cm[0][1] + Cm[1][0])
print("Accuracy",Accuracy)
Error_rate = (Cm[0][1] + Cm[1][0]) / (Cm[0][0] + Cm[1][1] + Cm[0][1] + Cm[1][0])
print("Error_rate",Error_rate)
Sensitivity = Cm[0][0]/(Cm[0][0] + Cm[1][0])
print("Sensitivity",Sensitivity)
Specificity = Cm[1][1]/(Cm[1][1] + Cm[0][1])
print("Specificity",Specificity)
Recall = Cm[0][0]/(Cm[0][0] + Cm[1][0])
print("Recall",Recall)
Precision = Cm[0][0]/(Cm[0][0] + Cm[0][1])
print("Precision",Precision)
F1Score = (2*(Precision*Recall))/(Precision + Recall)
print("F1Score",F1Score)
```

```
Accuracy 0.9298245614035088
Error_rate 0.07017543859649122
Sensitivity 0.9615384615384616
Specificity 0.8805970149253731
Recall 0.9615384615384616
Precision 0.9259259259259259
F1Score 0.9433962264150944
```

Finding the Accuracy by changing the Parameter

In [34]:
```
def doDC(X, y, test_size = 0.20, randomstate = 8,c='gini',mf='auto',mdps=3,mfn=2,mss=3):
    X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = test_size, random_state
=randomstate)
    cls2=DecisionTreeClassifier(criterion=c,max_features=mf,max_depth=mdps,max_leaf_nodes=mfn,min_
samples_split=mss)
    cls2.fit(X_train,Y_train)
    pred2=cls2.predict(X_test)
    acc_score2 = accuracy_score(pred2,Y_test)
    return acc_score2
```

In [35]:
```
test_size = [0.30, 0.25, 0.20,0.10]
random_states = [8, 27, 42]
criterions=['gini', 'entropy']
maxfeatures=['auto', 'sqrt', 'log2']
max_depths=[3,4,5,6]
max_leaf_nodes=[2,4,6,10,15]
min_samples_split=[2, 3, 4]
```

In [36]:
```
df = pd.DataFrame(columns = ['Test Size', 'Random States','Criterions','Max features','Max_depth',
'Max_leaf_node','Min_samples_splits','Decision Tree Accuracy'])
```

In [37]:
```
for t_size in test_size:
    for r_state in random_states:
        for crs in criterions:
            for mfs in maxfeatures:
                for mdps in max_depths:
                    for mfn in max_leaf_nodes:
                        for mss in min_samples_split:
                            a2 = doDC(X, Y, t_size, r_state,crs,mfs,mdps,mfn,mss)
                            I2 = {}
                            I2['Test Size'] = t_size
                            I2['Random States'] = r_state
                            I2['Criterions'] = crs
                            I2['Max features'] = mfs
                            I2['Max_depth'] = mdps
                            I2['Max_leaf_node'] = mfn
                            I2['Min_samples_splits'] = mss
                            I2['Decision Tree Accuracy'] = a2
                            df = df.append(I2, ignore_index = True)
```

In [38]:
```
df
```

Out[38]:

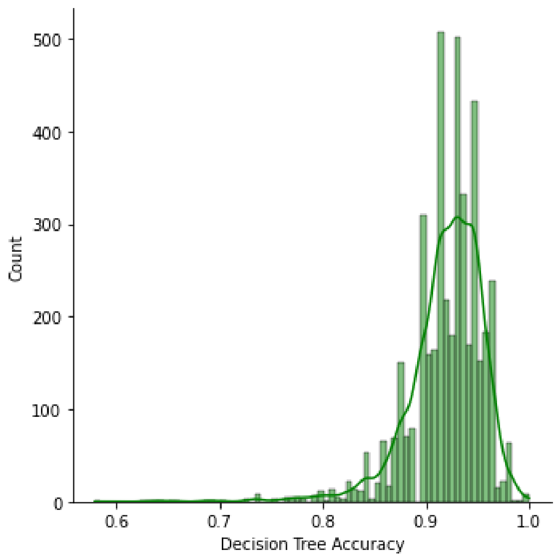
	Test Size	Random States	Criterions	Max features	Max_depth	Max_leaf_node	Min_samples_splits	Decision Tree Accuracy
0	0.3	8	gini	auto	3	2	2	0.865497
1	0.3	8	gini	auto	3	2	3	0.906433
2	0.3	8	gini	auto	3	2	4	0.865497
3	0.3	8	gini	auto	3	4	2	0.959064
4	0.3	8	gini	auto	3	4	3	0.947368
...	...	...	...	...	...	...	...	...
4315	0.1	42	entropy	log2	6	10	3	0.929825
4316	0.1	42	entropy	log2	6	10	4	0.929825
4317	0.1	42	entropy	log2	6	15	2	0.894737
4318	0.1	42	entropy	log2	6	15	3	0.912281
4319	0.1	42	entropy	log2	6	15	4	0.894737

4320 rows × 8 columns

Histogram Plot of Accuracy

In [39]:
```
sns.displot(x = 'Decision Tree Accuracy',kde = True, color='g', data = df)
```

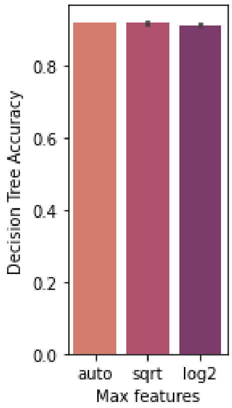
Out[39]: <seaborn.axisgrid.FacetGrid at 0x223dbbe5940>



Observation: Accuracy score range between 90 t0 98 mainly

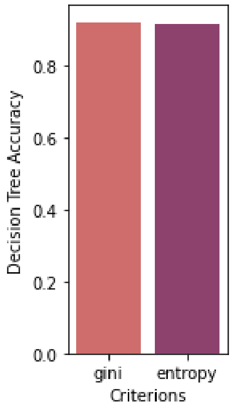
In [86]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'Max features', y = 'Decision Tree Accuracy',palette = "flare", data = df)
```

Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x223de85b910>



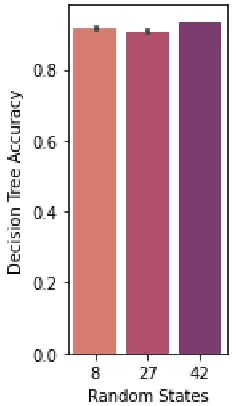
In [87]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'Criterions', y = 'Decision Tree Accuracy',palette = "flare", data = df)
```

Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x223d9c291c0>



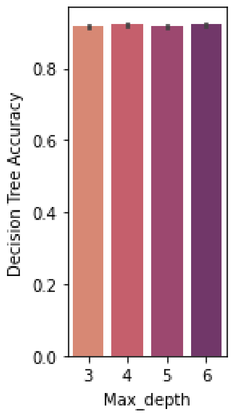
In [88]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'Random States', y = 'Decision Tree Accuracy',palette = "flare", data = df)
```

Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x223deaa3400>



In [89]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'Max_depth', y = 'Decision Tree Accuracy',palette = "flare", data = df)
```

Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x223de76a8e0>



Hyper Parameter Tuning

In [40]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
oHe = OneHotEncoder()
```

In [41]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'criterion':['gini','entropy'],'max_depth':[3,4,5,6,7,8,9,10,11,12,15,20,30,40,50,7
0,90,120,150],
               'max_leaf_nodes': [2,4,6,10,15,30,40,50,100], 'min_samples_split': [2, 3, 4]}]
oHe = OneHotEncoder()
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
grid_search.fit(X_train, Y_train)
best_accuracy_dtc = grid_search.best_score_
best_parameters = grid_search.best_params_
print(best_accuracy_dtc)
print(best_parameters)
```

```
0.9322435897435897
{'criterion': 'gini', 'max_depth': 10, 'max_leaf_nodes': 10, 'min_samples_split': 4}
```

Random Forest Classifier

In [42]:
```
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier()
classifier.fit(X_train, Y_train)
```

Out[42]: RandomForestClassifier()

In [ ]:
```
```

In [43]:
```
#Predicting the test set result
y_pred= classifier.predict(X_test)
```

In [44]:
```
y_pred
```

Out[44]:
```
array(['M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'M', 'M', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'M', 'B',
       'B', 'M', 'B', 'M', 'B', 'M', 'B', 'M', 'B', 'M', 'B', 'M', 'B',
       'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'M', 'M',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'M',
       'B', 'M', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'M', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'M', 'M', 'B',
       'M', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'M',
       'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M',
       'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'B',
       'B', 'B'], dtype=object)
```

In [62]:
```
from sklearn.metrics import accuracy_score
acc_score2 = accuracy_score(Y_test, y_pred)
print(acc_score2)
```

```
0.9590643274853801
```

In [45]:
```
y_prob = classifier.predict_proba(X_test)
```

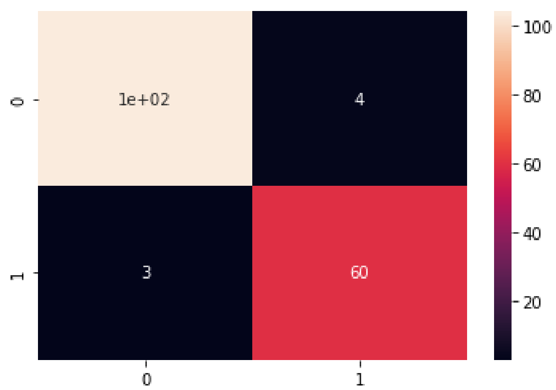
In [46]:
```
y_prob[0:5]
```

Out[46]:
```
array([[0.03, 0.97],
       [0.96, 0.04],
       [0.99, 0.01],
       [0.9 , 0.1 ],
       [0.95, 0.05]])
```

Confusion Matrix

In [47]:
```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(Y_test, y_pred)
dataframe_conf_matrix = conf_matrix
sns.heatmap(dataframe_conf_matrix, annot=True)
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x223dc0a5520>



In [48]:
```
Accuracy = (Cm[0][0] + Cm[1][1]) / (Cm[0][0] + Cm[1][1] + Cm[0][1] + Cm[1][0])
print("Accuracy",Accuracy)
Error_rate = (Cm[0][1] + Cm[1][0]) / (Cm[0][0] + Cm[1][1] + Cm[0][1] + Cm[1][0])
print("Error_rate",Error_rate)
Sensitivity = Cm[0][0]/(Cm[0][0] + Cm[1][0])
print("Sensitivity",Sensitivity)
Specificity = Cm[1][1]/(Cm[1][1] + Cm[0][1])
print("Specificity",Specificity)
Recall = Cm[0][0]/(Cm[0][0] + Cm[1][0])
print("Recall",Recall)
Precision = Cm[0][0]/(Cm[0][0] + Cm[0][1])
print("Precision",Precision)
F1Score = (2*(Precision*Recall))/(Precision + Recall)
print("F1Score",F1Score)
```

```
Accuracy 0.9298245614035088
Error_rate 0.07017543859649122
Sensitivity 0.9615384615384616
Specificity 0.8805970149253731
Recall 0.9615384615384616
Precision 0.9259259259259259
F1Score 0.9433962264150944
```

Finding the Accuracy by changing the Parameter

In [49]:
```
def doRF(X, y, test_size = 0.20, randomstate = 8,c='gini',mf='auto',mdps=3,mfn=2,mss=3,nes=10):
    X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = test_size, random_state
=randomstate)
    cls2=RandomForestClassifier(criterion=c,max_features=mf,max_depth=mdps,max_leaf_nodes=mfn,min_
samples_split=mss,n_estimators=nes)
    cls2.fit(X_train,Y_train)
    pred2=cls2.predict(X_test)
    acc_score3 = accuracy_score(pred2,Y_test)
    return acc_score3
```

In [50]:
```
test_size = [0.30, 0.25, 0.20,0.10]
random_states = [8, 27, 42]
criterions=['gini', 'entropy']
maxfeatures=['auto', 'sqrt', 'log2']
max_depths=[3,4,5,6]
max_leaf_nodes=[2,4,6,10,15]
min_samples_split=[2, 3, 4]
n_estimators=[10,20,40]
```

In [51]:
```
df2 = pd.DataFrame(columns = ['Test Size', 'Random States','Criterions','Max features','Max_depth'
,'Max_leaf_node','Min_samples_splits','n_estimators','Accuracy_RF'])
```

In [52]:
```
for t_size in test_size:
    for r_state in random_states:
        for crs in criterions:
            for mfs in maxfeatures:
                for mdps in max_depths:
                    for mfn in max_leaf_nodes:
                        for mss in min_samples_split:
                            for nes in n_estimators:
                                a3 = doRF(X, Y, t_size, r_state,crs,mfs,mdps,mfn,mss,nes)
                                RF = {}
                                RF['Test Size'] = t_size
                                RF['Random States'] = r_state
                                RF['Criterions'] = crs
                                RF['Max features'] = mfs
                                RF['Max_depth'] = mdps
                                RF['Max_leaf_node'] = mfn
                                RF['Min_samples_splits'] = mss
                                RF['n_estimators'] = nes
                                RF['Accuracy_RF'] = a3
                                df2 = df2.append(RF, ignore_index = True)
df2
```

Out[52]:

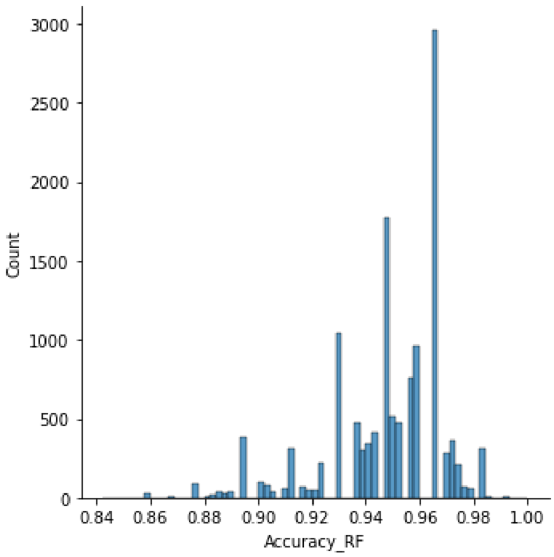
	Test Size	Random States	Criterions	Max features	Max_depth	Max_leaf_node	Min_samples_splits	n_estimators	Accuracy_RF
0	0.3	8	gini	auto	3	2	2	10	0.929825
1	0.3	8	gini	auto	3	2	2	20	0.923977
2	0.3	8	gini	auto	3	2	2	40	0.935673
3	0.3	8	gini	auto	3	2	3	10	0.929825
4	0.3	8	gini	auto	3	2	3	20	0.929825
...	...	...	...	...	...	...	...	...	...
12955	0.1	42	entropy	log2	6	15	3	20	0.964912
12956	0.1	42	entropy	log2	6	15	3	40	0.964912
12957	0.1	42	entropy	log2	6	15	4	10	0.964912
12958	0.1	42	entropy	log2	6	15	4	20	0.964912
12959	0.1	42	entropy	log2	6	15	4	40	0.964912

12960 rows × 9 columns

Histogram Plot of Accuracy

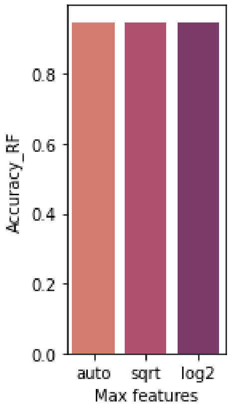
In [54]:
```
sns.displot(x = 'Accuracy_RF', data = df2)
```

Out[54]: <seaborn.axisgrid.FacetGrid at 0x223dbbc6b50>



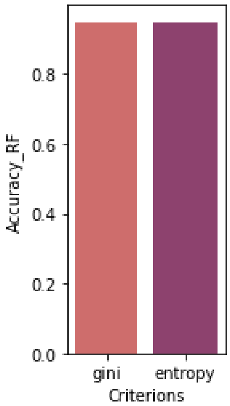
In [73]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'Max features', y = 'Accuracy_RF',palette = "flare", data = df2)
```

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x223d9c55970>



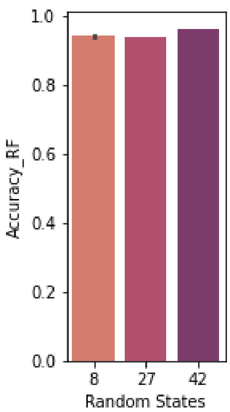
In [74]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'Criterions', y = 'Accuracy_RF',palette = "flare", data = df2)
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x223de6a8a90>



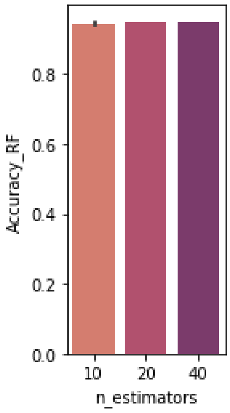
In [75]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'Random States', y = 'Accuracy_RF',palette = "flare", data = df2)
```

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x223dbd82ac0>



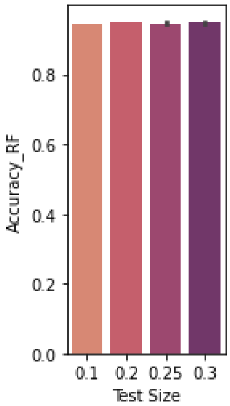
In [78]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'n_estimators', y = 'Accuracy_RF',palette = "flare", data = df2)
```

Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x223dea72610>



In [79]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'Test Size', y = 'Accuracy_RF',palette = "flare", data = df2)
```

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x223de860be0>



HyperParamter Tuning

In [55]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
oHe = OneHotEncoder()
```

In [60]:
```
from sklearn.model_selection import GridSearchCV
parameters = [{'criterion':['gini','entropy'],'max_depth':[3,4,5,6,7,8,9,10,11,12,15,20,30,40,50,7
0,90,120,150],
                'max_leaf_nodes': [2,4,6,10,15,30,40,50,100], 'min_samples_split': [2, 3, 4],'n_es
timators':[10,20,40,60,100]}]
oHe = OneHotEncoder()
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
grid_search.fit(X_train, Y_train)
best_accuracy_rf = grid_search.best_score_
best_parameters = grid_search.best_params_
print(best_accuracy_rf)
print(best_parameters)
```

```
0.97
{'criterion': 'gini', 'max_depth': 150, 'max_leaf_nodes': 50, 'min_samples_split': 3, 'n_estimator
s': 20}
```

Comparing wiith Test size and Random state

In [65]:
```
df3 = pd.DataFrame(columns = ['Test Size', 'Random States'])
```

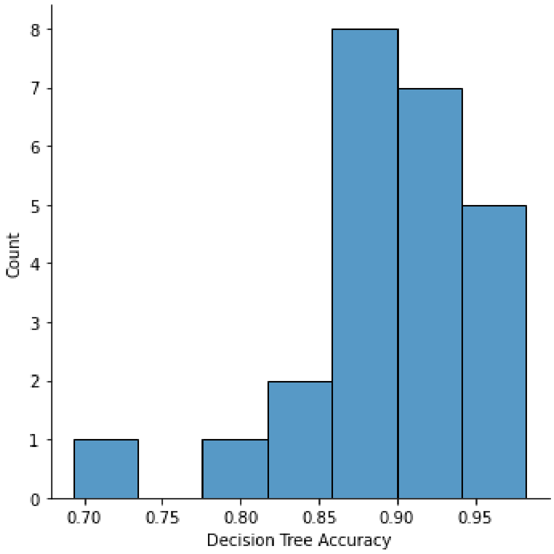
In [68]:
```
for t_size in test_size:
    for r_state in random_states:
        a2 = doDC(X, Y, t_size, r_state)
        a3 = doRF(X, Y, t_size, r_state)

        F = {}
        F['Test Size'] = t_size
        F['Random States'] = r_state


        F['Decision Tree Accuracy'] = a2
        F['Random Forest Accuracy'] = a3
        df3 = df3.append(F, ignore_index = True)

df3
```

Out[68]:

	Test Size	Random States	Decision Tree Accuracy	Random Forest Accuracy
0	0.30	8.0	0.941520	0.923977
1	0.30	27.0	0.871345	0.894737
2	0.30	42.0	0.842105	0.947368
3	0.25	8.0	0.916084	0.916084
4	0.25	27.0	0.881119	0.937063
5	0.25	42.0	0.951049	0.958042
6	0.20	8.0	0.947368	0.921053
7	0.20	27.0	0.894737	0.877193
8	0.20	42.0	0.938596	0.956140
9	0.10	8.0	0.789474	0.877193
10	0.10	27.0	0.842105	0.912281
11	0.10	42.0	0.982456	0.982456
12	0.30	8.0	0.865497	0.912281
13	0.30	27.0	0.871345	0.923977
14	0.30	42.0	0.894737	0.941520
15	0.25	8.0	0.923077	0.902098
16	0.25	27.0	0.909091	0.902098
17	0.25	42.0	0.895105	0.958042
18	0.20	8.0	0.692982	0.929825
19	0.20	27.0	0.894737	0.877193
20	0.20	42.0	0.947368	0.938596
21	0.10	8.0	0.912281	0.894737
22	0.10	27.0	0.912281	0.912281
23	0.10	42.0	0.929825	0.982456

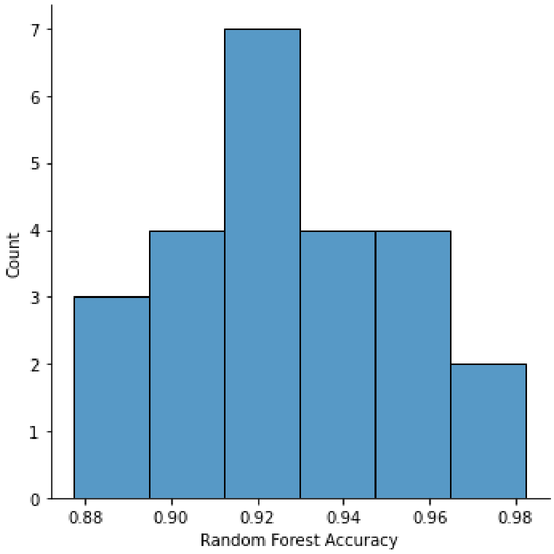
In [80]:
```
sns.displot(x = 'Decision Tree Accuracy', data = df3)
```

Out[80]: <seaborn.axisgrid.FacetGrid at 0x223debd4970>



In [81]:
```
sns.displot(x = 'Random Forest Accuracy', data = df3)
```

Out[81]: <seaborn.axisgrid.FacetGrid at 0x223dd36c760>



Comparing models before and after Parameter Tuning¶

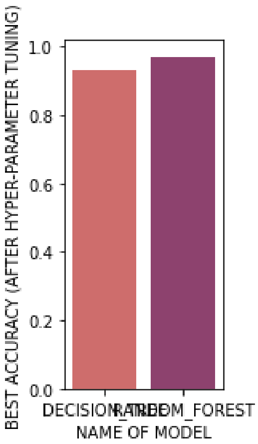
In [64]:
```
rediction_columns = ["NAME OF MODEL", "ACCURACY SCORE", "BEST ACCURACY (AFTER HYPER-PARAMETER TUNI
NG)"]
df_pred = {"NAME OF MODEL" : ["DECISION_TREE", "RANDOM_FOREST"],
           "ACCURACY SCORE " : [acc_score1, acc_score2],
           "BEST ACCURACY (AFTER HYPER-PARAMETER TUNING)" : [best_accuracy_dtc, best_accuracy_rf]}
df_predictions = pd.DataFrame (df_pred)
df_predictions
```

Out[64]:

	NAME OF MODEL	ACCURACY SCORE	BEST ACCURACY (AFTER HYPER-PARAMETER TUNING)
0	DECISION_TREE	0.929825	0.932244
1	RANDOM_FOREST	0.959064	0.970000

In [71]:
```
plt.subplot(1,3,3)
sns.barplot(x = 'NAME OF MODEL', y = 'BEST ACCURACY (AFTER HYPER-PARAMETER TUNING)',palette = "flare", data = df_predictions)
```

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x223deaa6c40>



Conclusion

In this Lab,, it is fairly evident that it is theRandom Forest model that has come out triumphant with the highest accuracies both before and after the hyper-parameter tuning. It ended up with an accuracy of 95% before hyper-parameter tuning and 97% after and is hence, the best suited model out of the rest for the given dataset.

In [ ]: