In [1]:

```
# classification - decision tree implemetaiton
# aim : to predict the chances of having breast cancer using classification algorithm

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# keeps the plots in one place. calls image as static pngs
%matplotlib inline
import matplotlib.pyplot as plt # side-stepping mpl backend
import matplotlib.gridspec as gridspec # subplots
import mpld3 as mpl

#Import models from scikit learn module:
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics
```

In [2]:

```
# Load the Data
df = pd.read_csv("breast_cancer_data.csv",header = 0)
df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.1

5 rows × 33 columns

In [3]:

```
# Cleaning and Preparing the data
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
# size of the dataframe
len(df)
```

Out[3]:

```
569
```

In [4]:

```
df.diagnosis.unique()
```

Out[4]:

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

In [5]:

```
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
df.head()
```

Out[5]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	com
0	1	17.99	10.38	122.80	1001.0	0.11840	
1	1	20.57	17.77	132.90	1326.0	0.08474	
2	1	19.69	21.25	130.00	1203.0	0.10960	
3	1	11.42	20.38	77.58	386.1	0.14250	
4	1	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 31 columns

In [6]:

```
# Explore the data
df.describe()
```

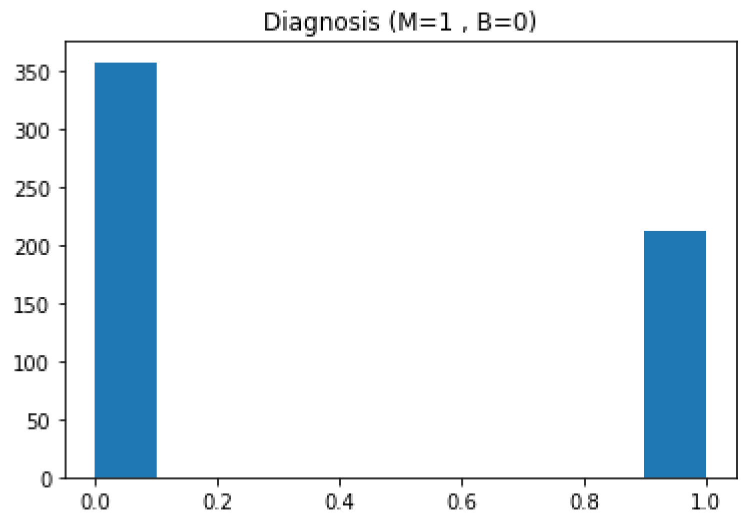
Out[6]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mear
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400

8 rows × 31 columns

In [7]:

```
df.describe()
plt.hist(df['diagnosis'])
plt.title('Diagnosis (M=1 , B=0)')
plt.show()
```



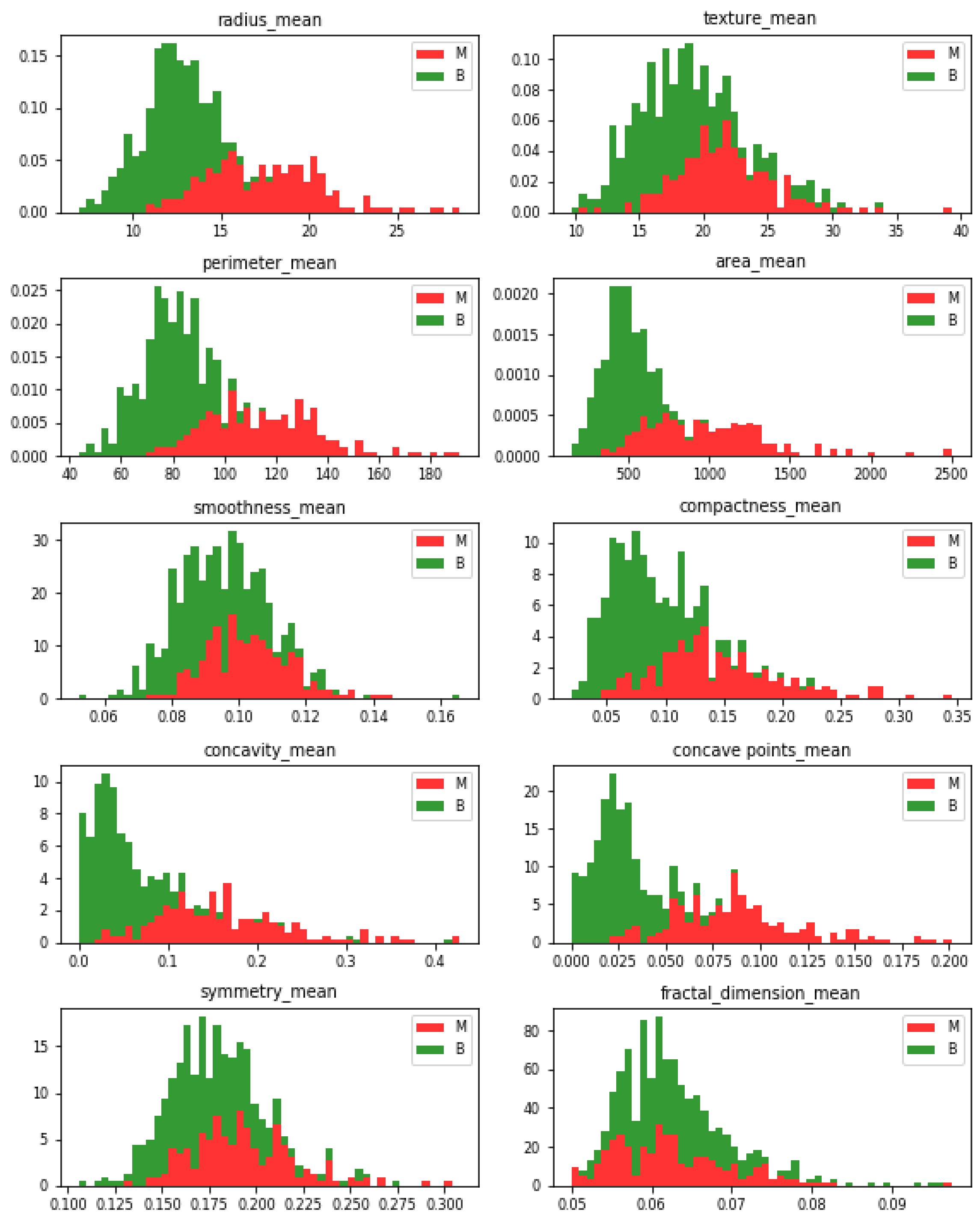
In [8]:

```
# nucleus features vs diagnosis
features_mean=list(df.columns[1:11])
# split dataframe into two based on diagnosis
dfM=df[df['diagnosis'] ==1]
dfB=df[df['diagnosis'] ==0]
```

In [9]:

```
#Stack the data
plt.rcParams.update({'font.size': 8})
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8,10))
axes = axes.ravel()
for idx,ax in enumerate(axes):
    ax.figure
    binwidth= (max(df[features_mean[idx]]) - min(df[features_mean[idx]]))/50
    ax.hist([dfM[features_mean[idx]],dfB[features_mean[idx]]], bins=np.arange(min(df[featur
    ax.legend(loc='upper right')
    ax.set_title(features_mean[idx])
plt.tight_layout()
plt.show()
```

```
D:\anaconda_folder\lib\site-packages\numpy\core\_asarray.py:83: VisibleDepre
cationWarning: Creating an ndarray from ragged nested sequences (which is a
list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shape
s) is deprecated. If you meant to do this, you must specify 'dtype=object' w
hen creating the ndarray
  return array(a, dtype, copy=False, order=order)
```



In [10]:

```
# Observations
# 1. mean values of cell radius, perimeter, area, compactness, concavity and concave points
#    of the cancer. Larger values of these parameters tends to show a correlation with malig
# 2. mean values of texture, smoothness, symmetry or fractal dimension does not show a par
#     over the other. In any of the histograms there are no noticeable large outliers that w
```

In [11]:

```
# Creating a test set and a training set
traindf, testdf = train_test_split(df, test_size = 0.3)
```

In [12]:

```
#Generic function for making a classification model and accessing the performance.

def classification_model(model, data, predictors, outcome):
  #Fit the model:
  model.fit(data[predictors],data[outcome])

  #Make predictions on training set:
  predictions = model.predict(data[predictors])

  #Print accuracy
  accuracy = metrics.accuracy_score(predictions,data[outcome])
  print("Accuracy : %s" % "{0:.3%}".format(accuracy))

  #Fit the model again so that it can be refered outside the function:
  model.fit(data[predictors],data[outcome])
```

In [13]:

```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave poi
outcome_var='diagnosis'
model=LogisticRegression()
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 90.955%

D:\anaconda_folder\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a s
olver to silence this warning.
  FutureWarning)
D:\anaconda_folder\lib\site-packages\sklearn\linear_model\logistic.py:432: F
utureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a s
olver to silence this warning.
  FutureWarning)
```

In [14]:

```
# Decision Tree Model

predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave poi
model = DecisionTreeClassifier()
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
```

In [15]:

```
# Here we are over-fitting the model probably due to the large number of predictors.
# Let use a single predictor, the obvious one is the radius of the cell.
```

In [16]:

```
predictor_var = ['radius_mean']
model = DecisionTreeClassifier()
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 97.487%
```

In [17]:

```
# The accuracy of the prediction is much much better here. But does it depend on the predic
# Using a single predictor gives a 97% prediction accuracy for this model.
```

In [18]:

```
# Randome Forest
# Use all the features of the nucleus
predictor_var = features_mean
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_depth=7, max_feat
classification_model(model, traindf,predictor_var,outcome_var)
```

```
Accuracy : 96.482%
```

In [19]:

```
#Create a series with feature importances:
featimp = pd.Series(model.feature_importances_, index=predictor_var).sort_values(ascending=
print(featimp)
```

```
perimeter_mean            0.216669
concave points_mean       0.194711
radius_mean               0.184149
area_mean                 0.158421
concavity_mean            0.092122
compactness_mean          0.067511
texture_mean              0.044243
smoothness_mean           0.018908
fractal_dimension_mean    0.014772
symmetry_mean             0.008495
dtype: float64
```

In [20]:

```
# Using top 5 features
predictor_var = ['concave points_mean','area_mean','radius_mean','perimeter_mean','concavit
model = RandomForestClassifier(n_estimators=100, min_samples_split=25, max_depth=7, max_fea
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 95.226%
```

In [21]:

```
predictor_var =  ['radius_mean']
model = RandomForestClassifier(n_estimators=100)
classification_model(model, traindf,predictor_var,outcome_var)
```

```
Accuracy : 97.487%
```

In [22]:

```
# Using on the test dataset
# Use all the features of the nucleus
predictor_var = features_mean
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_depth=7, max_feat
classification_model(model, testdf,predictor_var,outcome_var)
```

```
Accuracy : 95.322%
```