ASSIGNMENT - 3

Attribute Information:

- 1) ID number
- 2) Diagnosis (M = malignant, B = benign)

DATASET:- BREAST CANCER

Load Libraries

In [33]:
```
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.model_selection import KFold   #For K-fold cross validation
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics
```

Load the data

In [2]:
```
df = pd.read_csv("../input/data.csv",header = 0)
df.head()
```

Out[2]:

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

5 rows × 33 columns

Clean and prepare data

In [3]:
```
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
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

Explore data

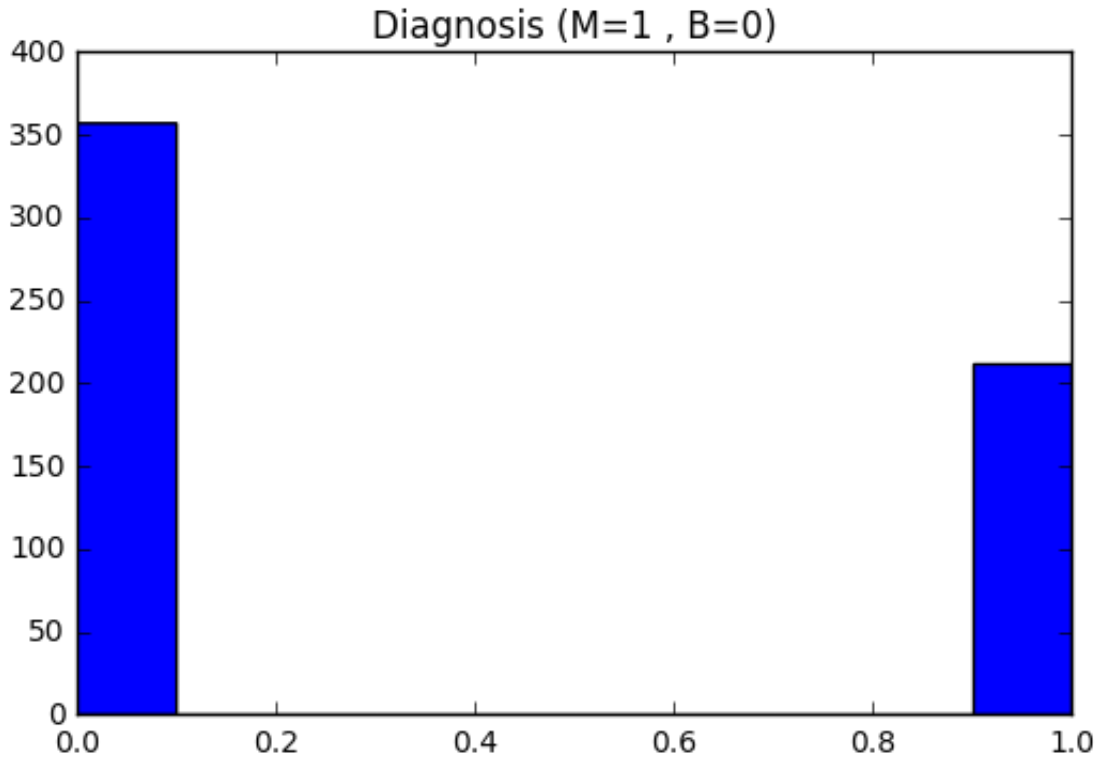
In [6]:
```
df.describe()
```

Out[6]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.0
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.0
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.0
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.0
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.0
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.

8 rows × 31 columns

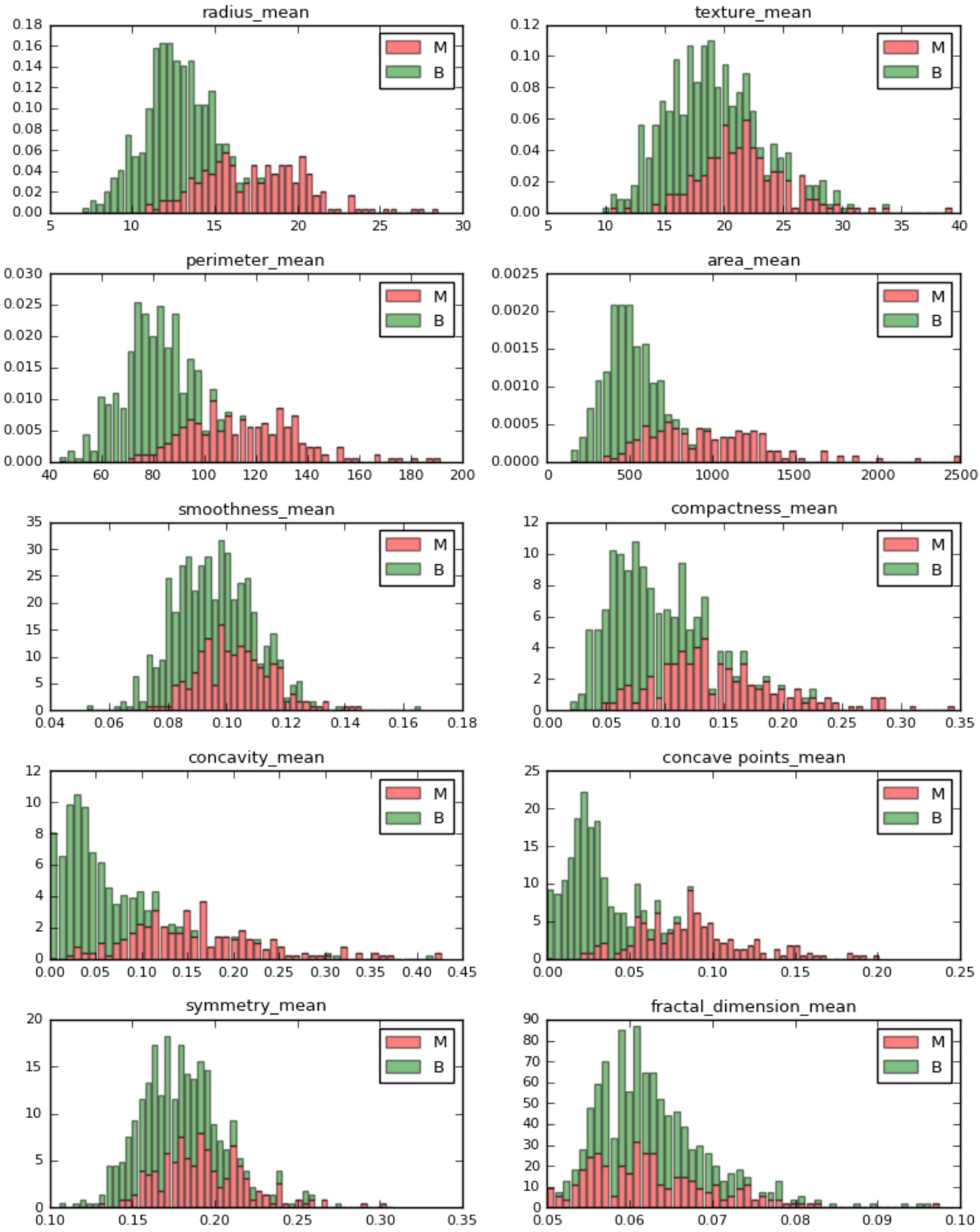
In [7]:
```
df.describe()
plt.hist(df['diagnosis'])
plt.title('Diagnosis (M=1 , B=0)')
plt.show()
```



nucleus features vs diagnosis

In [8]:
```
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]]))/
    ax.hist([dfM[features_mean[idx]],dfB[features_mean[idx]]], bins=np.ara
    ax.legend(loc='upper right')
    ax.set_title(features_mean[idx])
plt.tight_layout()
plt.show()
```


radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave points_mean
symmetry_mean
fractal_dimension_mean
M
B

Observations

1. mean values of cell radius, perimeter, area, compactness, concavity and concave points can be used in classification of the cancer. Larger values of these parameters tends to show a correlation with malignant tumors.
2. mean values of texture, smoothness, symmetry or fractual dimension does not show a particular preference of one diagnosis over the other. In any of the histograms there are no noticeable large outliers that warrants further cleanup.

Creating a test set and a training set

Since this data set is not ordered, I am going to do a simple 70:30 split to create a training data set and a test data set.

In [10]:
```
traindf, testdf = train_test_split(df, test_size = 0.3)
```

Model Classification

Here we are going to build a classification model and evaluate its performance using the training set.

In [11]:
```
#Generic function for making a classification model and accessing the perf
# From AnalyticsVidhya tutorial
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))

  #Perform k-fold cross-validation with 5 folds
  kf = KFold(data.shape[0], n_folds=5)
  error = []
  for train, test in kf:
    # Filter training data
    train_predictors = (data[predictors].iloc[train,:])

    # The target we're using to train the algorithm.
    train_target = data[outcome].iloc[train]

    # Training the algorithm using the predictors and target.
    model.fit(train_predictors, train_target)

    #Record error from each cross-validation run
    error.append(model.score(data[predictors].iloc[test,:], data[outcome].

    print("Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error))

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

Logistic Regression model

Logistic regression is widely used for classification of discrete data. In this case we will use it for binary (1,0) classification.

Based on the observations in the histogram plots, we can reasonably hypothesize that the cancer diagnosis depends on the mean cell radius, mean perimeter, mean area, mean compactness, mean concavity and mean concave points. We can then perform a logistic regression analysis using those features as follows:

In [12]:
```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_me
outcome_var='diagnosis'
model=LogisticRegression()
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 88.442%
Cross-Validation Score : 88.750%
Cross-Validation Score : 87.500%
Cross-Validation Score : 87.917%
Cross-Validation Score : 87.773%
Cross-Validation Score : 88.193%
```

The prediction accuracy is reasonable. What happens if we use just one predictor? Use the mean_radius:

In [13]:
```
predictor_var = ['radius_mean']
model=LogisticRegression()
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 87.940%
Cross-Validation Score : 90.000%
Cross-Validation Score : 88.750%
Cross-Validation Score : 88.333%
Cross-Validation Score : 88.718%
Cross-Validation Score : 87.684%
```

This gives a similar prediction accuracy and a cross-validation score.

The accuracy of the predictions are good but not great. The cross-validation scores are reasonable. Can we do better with another model?

Decision Tree Model

In [14]:
```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_me
model = DecisionTreeClassifier()
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
Cross-Validation Score : 87.500%
Cross-Validation Score : 87.500%
Cross-Validation Score : 85.000%
Cross-Validation Score : 84.636%
Cross-Validation Score : 84.924%
```

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 [15]:
```
predictor_var = ['radius_mean']
model = DecisionTreeClassifier()
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 97.236%
Cross-Validation Score : 85.000%
Cross-Validation Score : 82.500%
Cross-Validation Score : 83.750%
Cross-Validation Score : 84.015%
Cross-Validation Score : 83.921%
```

The accuracy of the prediction is much much better here. But does it depend on the predictor?

Using a single predictor gives a 97% prediction accuracy for this model but the cross-validation score is not that great.

Randome Forest

In [16]:
```
# Use all the features of the nucleus
predictor_var = features_mean
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_
classification_model(model, traindf,predictor_var,outcome_var)
```

```
Accuracy : 94.724%
Cross-Validation Score : 93.750%
Cross-Validation Score : 92.500%
Cross-Validation Score : 91.250%
Cross-Validation Score : 90.906%
Cross-Validation Score : 90.953%
```

Using all the features improves the prediction accuracy and the cross-validation score is great.

An advantage with Random Forest is that it returns a feature importance matrix which can be used to select features. So lets select the top 5 features and use them as predictors.

In [17]:
```
#Create a series with feature importances:
featimp = pd.Series(model.feature_importances_, index=predictor_var).sort_
print(featimp)
```

```
perimeter_mean            0.204561
concave points_mean       0.197067
concavity_mean            0.182823
area_mean                 0.173446
radius_mean               0.115543
compactness_mean          0.043632
texture_mean              0.038492
smoothness_mean           0.019023
symmetry_mean             0.016371
fractal_dimension_mean    0.009043
dtype: float64
```

In [18]:
```
# Using top 5 features
predictor_var = ['concave points_mean','area_mean','radius_mean','perimete
model = RandomForestClassifier(n_estimators=100, min_samples_split=25, max_
classification_model(model,traindf,predictor_var,outcome_var)
```

```
Accuracy : 94.221%
Cross-Validation Score : 92.500%
Cross-Validation Score : 91.875%
Cross-Validation Score : 90.833%
Cross-Validation Score : 90.277%
Cross-Validation Score : 90.449%
```

Using the top 5 features only changes the prediction accuracy a bit but I think we get a better result if we use all the predictors.

What happens if we use a single predictor as before? Just check.

In [19]:
```
predictor_var =  ['radius_mean']
model = RandomForestClassifier(n_estimators=100)
classification_model(model, traindf,predictor_var,outcome_var)
```

```
Accuracy : 97.236%
Cross-Validation Score : 85.000%
Cross-Validation Score : 81.875%
Cross-Validation Score : 83.333%
Cross-Validation Score : 83.703%
Cross-Validation Score : 83.924%
```

This gives a better prediction accuracy too but the cross-validation is not great.

Using on the test data set

In [20]:
```
# Use all the features of the nucleus
predictor_var = features_mean
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_
classification_model(model, testdf,predictor_var,outcome_var)
```

```
Accuracy : 96.491%
Cross-Validation Score : 97.143%
Cross-Validation Score : 95.630%
Cross-Validation Score : 96.106%
Cross-Validation Score : 94.139%
Cross-Validation Score : 93.546%
```

The prediction accuracy for the test data set using the above Random Forest model is 95%!

Conclusion

The best model to be used for diagnosing breast cancer as found in this analysis is the Random Forest model with the top 5 predictors, 'concave points_mean','area_mean','radius_mean','perimeter_mean','concavity_mean'. It gives a prediction accuracy of ~95% and a cross-validation score ~ 93% for the test data set.