In [8]:

```
import numpy as np

# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *
from qiskit.providers.aer import QasmSimulator

# Loading your IBM Quantum account(s)
provider = IBMQ.load_account()
```

```
ibmqfactory.load_account:WARNING:2022-01-07 06:03:14,843: Credentials are
already in use. The existing account in the session will be replaced.
```

In [9]:

```
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 models from scikit learn module:
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

In [10]:

```
dataset = pd.read_csv("data.csv",header = 0) #read csv files
dataset.head() #read top 5 rows
```

Out[10]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
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

In [11]:

```
#dropping unnecessary data from dataset
dataset.drop('id',axis=1,inplace=True)   #id column removed
dataset.drop('Unnamed: 32',axis=1,inplace=True) #'Unnamed' column removed
# size of the dataframe
len(dataset)
```

Out[11]:

```
569
```

In [12]:

```
#finding unique values of 'diagnosis' column of dataset
dataset.diagnosis.unique()
```

Out[12]:

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

In [13]:

```
#mapping the unique string values to respecitve integer values of 'diagnosis' column
dataset['diagnosis'] = dataset['diagnosis'].map({'M':1,'B':0})     # M=malignant(cancer
ous growth), B = benign(non harmful tumour)
dataset.head()   #printing top 5 rows of dataset
```

Out[13]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	c
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 [14]:

```
dataset.describe() #getting an overview of dataset
```

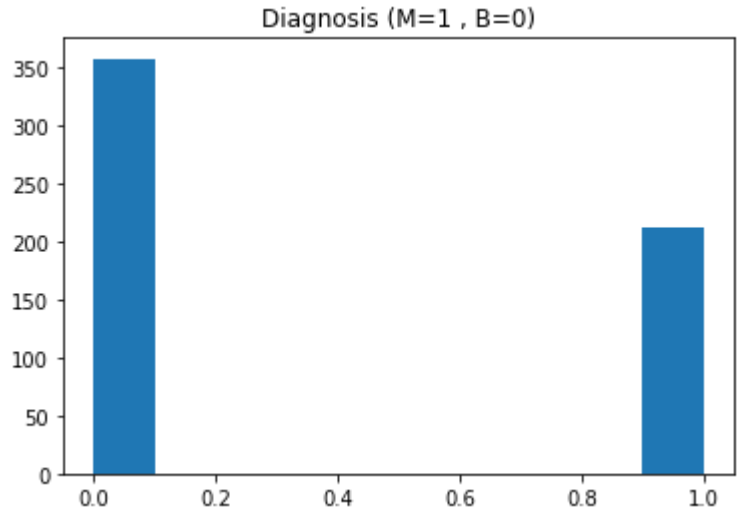
Out[14]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000
mean	0.372583	14.127292	19.289649	91.969033	654.889104	0.096
std	0.483918	3.524049	4.301036	24.298981	351.914129	0.014
min	0.000000	6.981000	9.710000	43.790000	143.500000	0.052
25%	0.000000	11.700000	16.170000	75.170000	420.300000	0.086
50%	0.000000	13.370000	18.840000	86.240000	551.100000	0.095
75%	1.000000	15.780000	21.800000	104.100000	782.700000	0.105
max	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163

8 rows × 31 columns

In [15]:

```
plt.hist(dataset['diagnosis'])
plt.title('Diagnosis (M=1 , B=0)')
plt.show() #plotting a histogram of M and B
```



In [16]:

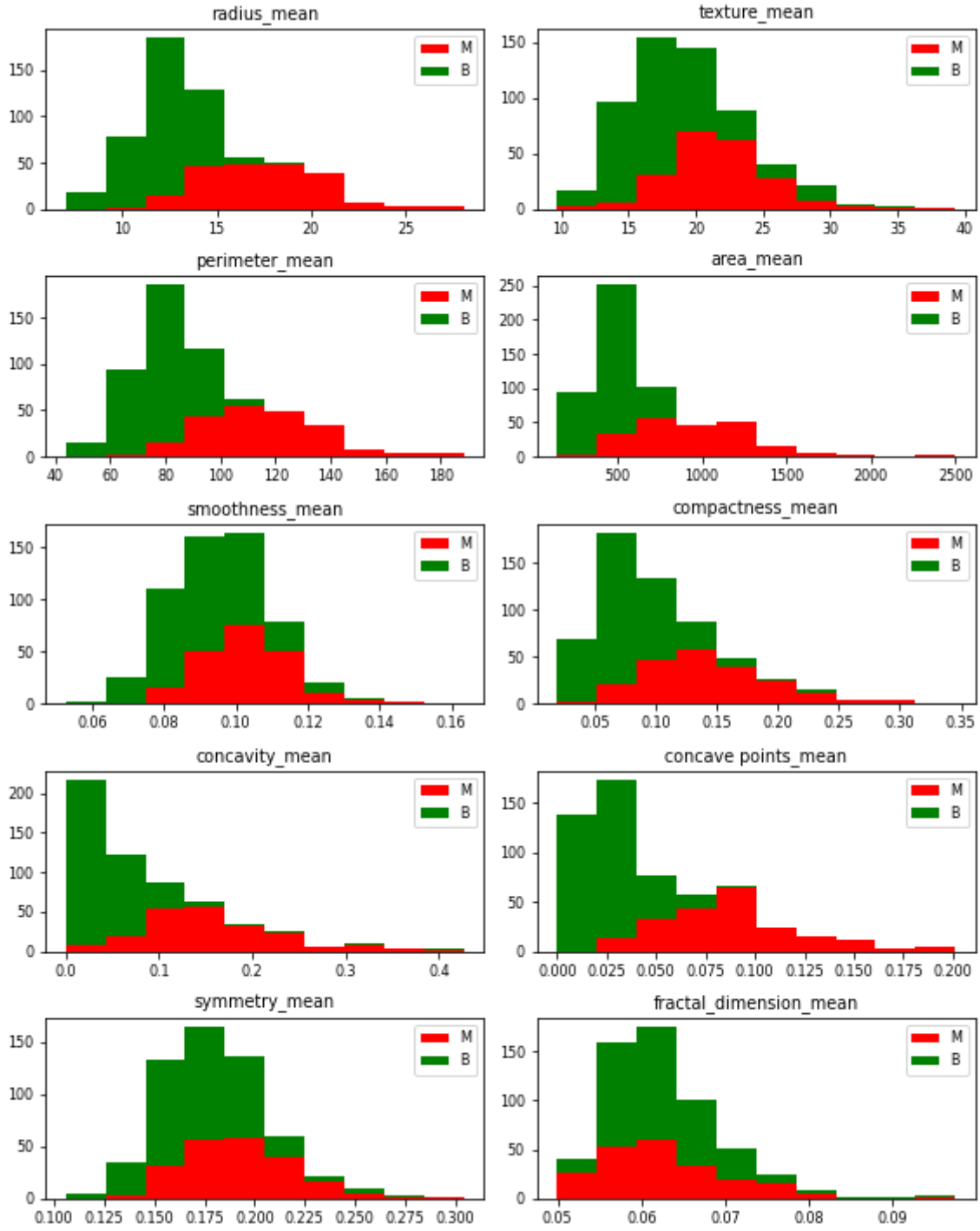
```
# features to be used for prediction
features_mean=list(dataset.columns[1:11])
datasetM=dataset[dataset['diagnosis'] ==1]
datasetB=dataset[dataset['diagnosis'] ==0]
features_mean
```

Out[16]:

```
['radius_mean',
 'texture_mean',
 'perimeter_mean',
 'area_mean',
 'smoothness_mean',
 'compactness_mean',
 'concavity_mean',
 'concave points_mean',
 'symmetry_mean',
 'fractal_dimension_mean']
```

In [17]:

```
#Plotting histogram for all features from both datasetM and datasetB
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):
    binwidth= (max(dataset[features_mean[idx]]) - min(dataset[features_mean[idx]]))/50
    ax.hist([datasetM[features_mean[idx]],datasetB[features_mean[idx]]],stacked=True,la
bel=['M','B'],color=['r','g'])
    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

In [18]:

```
#split data into two parts
traindf, testdf = train_test_split(dataset, test_size = 0.3)
```

In [19]:

```
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:.4%}".format(accuracy))
```

In [20]:

```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave
points_mean'] #attributes of important features
outcome_var='diagnosis'         #outcome variable
model=LogisticRegression()      #object of Logistic regression
classification_model(model,traindf,predictor_var,outcome_var) #using predict_var to pre
dict outcome_var using Logistic regression model
```

```
Accuracy : 88.9447%
```

In [21]:

```
import warnings
warnings.filterwarnings("ignore")  #to avoid printing warnings

#Taking data fromm user
Radius_mean=float(input("Enter Radius Mean: "))
Perimeter_mean=float(input("Enter Perimeter Mean: "))
Area_mean=float(input("Enter Area Mean"))
Compactness_mean=float(input("Enter Compactness Mean: "))
Concavepoints_mean=float(input("Enter concave points_mean  Mean: "))


input_data = (Radius_mean,Perimeter_mean,Area_mean,Compactness_mean,Concavepoints_mean)

# changing input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshaped) #using logistic regression
print(prediction)
#print('Diagnosis ', prediction[0])
if(prediction[0]==0):
  print("Non-cancerous tumour")
else:
  print("Cancerous tumour")
```

```
[1]
Cancerous tumour
```