# Part 1: Modeling

# 1. Let's import necessary dependencies for our project

In [383]:
```
import pandas as pd
import numpy as np #array manipulations
import seaborn as sns # visualization
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.cross_validation import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,confusion_matrix,accuracy_sc
ore
from sklearn.svm import SVC
from sklearn import metrics, preprocessing
from IPython.display import Image, display
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.cross_validation import cross_val_score, KFold
%matplotlib inline
```

# 2. Import our dataset into pandas dataframe object

For this task we have been provided with two files. The first file contains measurements of Breast Cancer Cells and the second file contains the header for this file

In [298]:
```
header = open("C:/Users/subra/Desktop/field_names.txt","r")
#read the lines from the text file and split each one in a new line
header_line = header.read().split()
header_line
```

Out[298]:
```
['ID',
 'diagnosis',
 'radius_mean',
 'radius_sd_error',
 'radius_worst',
 'texture_mean',
 'texture_sd_error',
 'texture_worst',
 'perimeter_mean',
 'perimeter_sd_error',
 'perimeter_worst',
 'area_mean',
 'area_sd_error',
 'area_worst',
 'smoothness_mean',
 'smoothness_sd_error',
 'smoothness_worst',
 'compactness_mean',
 'compactness_sd_error',
 'compactness_worst',
 'concavity_mean',
 'concavity_sd_error',
 'concavity_worst',
 'concave_points_mean',
 'concave_points_sd_error',
 'concave_points_worst',
 'symmetry_mean',
 'symmetry_sd_error',
 'symmetry_worst',
 'fractal_dimension_mean',
 'fractal_dimension_sd_error',
 'fractal_dimension_worst']
```

In [299]:
```
###load the breast cancer measurement csv file and set the column names to the
 filed names stored above(header_line)
df=pd.read_csv("C:/Users/subra/Desktop/breast-cancer.csv",names=header_line)
df.head()
```

Out[299]:

| | ID | diagnosis | radius_mean | radius_sd_error | radius_worst | texture_mean | text |
|---|---|---|---|---|---|---|---|
| **0** | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11 |
| **1** | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08 |
| **2** | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10 |
| **3** | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14 |
| **4** | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10 |

5 rows × 32 columns

# 3. Data Pre-processing

In [300]:
```
#### we don't need the ID column, so let's drop that column
df.drop('ID', axis=1, inplace=True)
df.head()
```

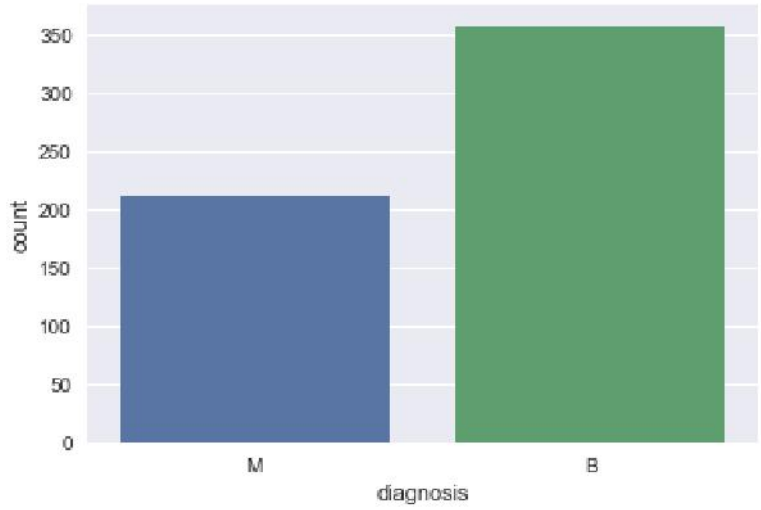
Out[300]:

| | diagnosis | radius_mean | radius_sd_error | radius_worst | texture_mean | texture_sd_err |
|---|---|---|---|---|---|---|
| **0** | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 |
| **1** | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 |
| **2** | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 |
| **3** | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 |
| **4** | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 |

5 rows × 31 columns

In [301]:
```
#separting the data by categories for visuliazation puprose
data_M = df[df['diagnosis']=='M']
data_B = df[df['diagnosis']=='B']
#Histogram of Class frequencies of Breast Cancer Diagnosis.
import seaborn as sns
sns.countplot('diagnosis',data=df)
```

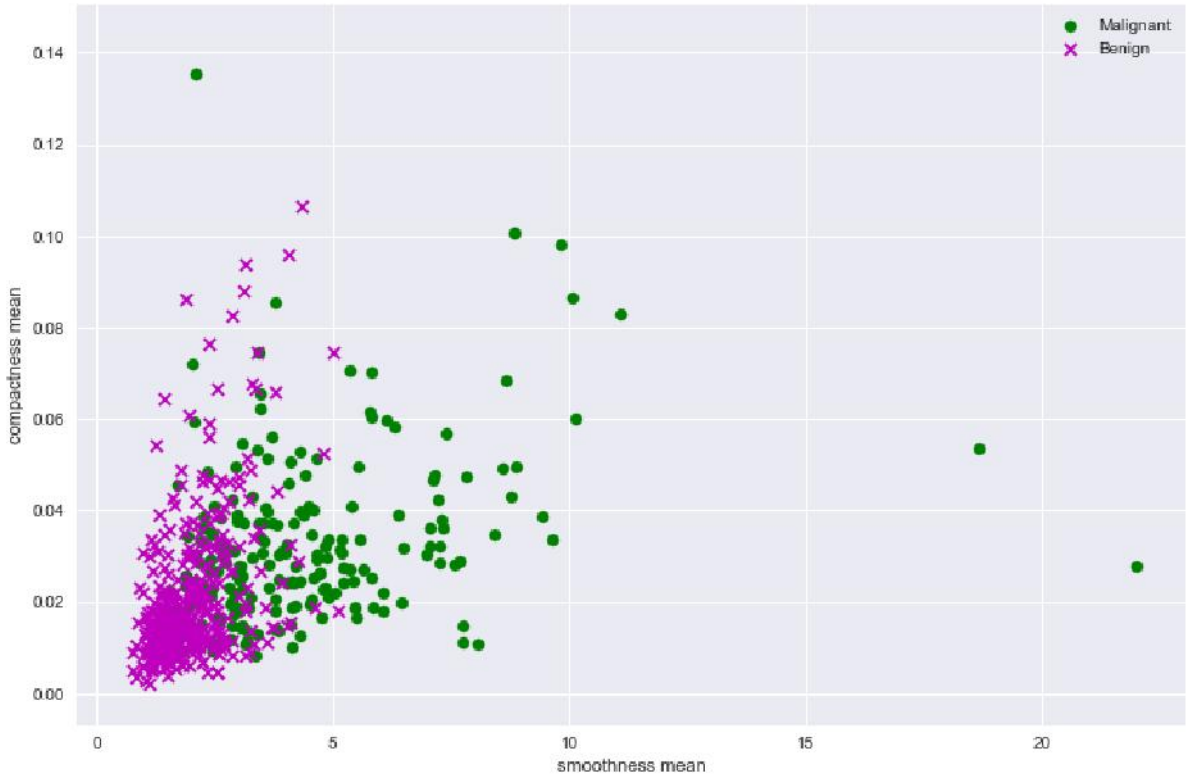
Out[301]: <matplotlib.axes._subplots.AxesSubplot at 0x2640bb9ee10>




The prior of classes(categories) play an important role on the performance of a classifier

In [362]:
```
#randomly choosing columns smoothness mean and compactness_mean to plot on 2-D
 axis for visual exploration
fig, (ax) = plt.subplots(figsize=(12,8))
ax.scatter(data_M['smoothness_mean'], data_M['compactness_mean'], s=50, c='g',
 marker='o', label='Malignant')
ax.scatter(data_B['smoothness_mean'], data_B['compactness_mean'], s=50, c='m',
 marker='x', label='Benign')
ax.legend()
ax.set_xlabel('smoothness mean')
ax.set_ylabel('compactness mean')
```

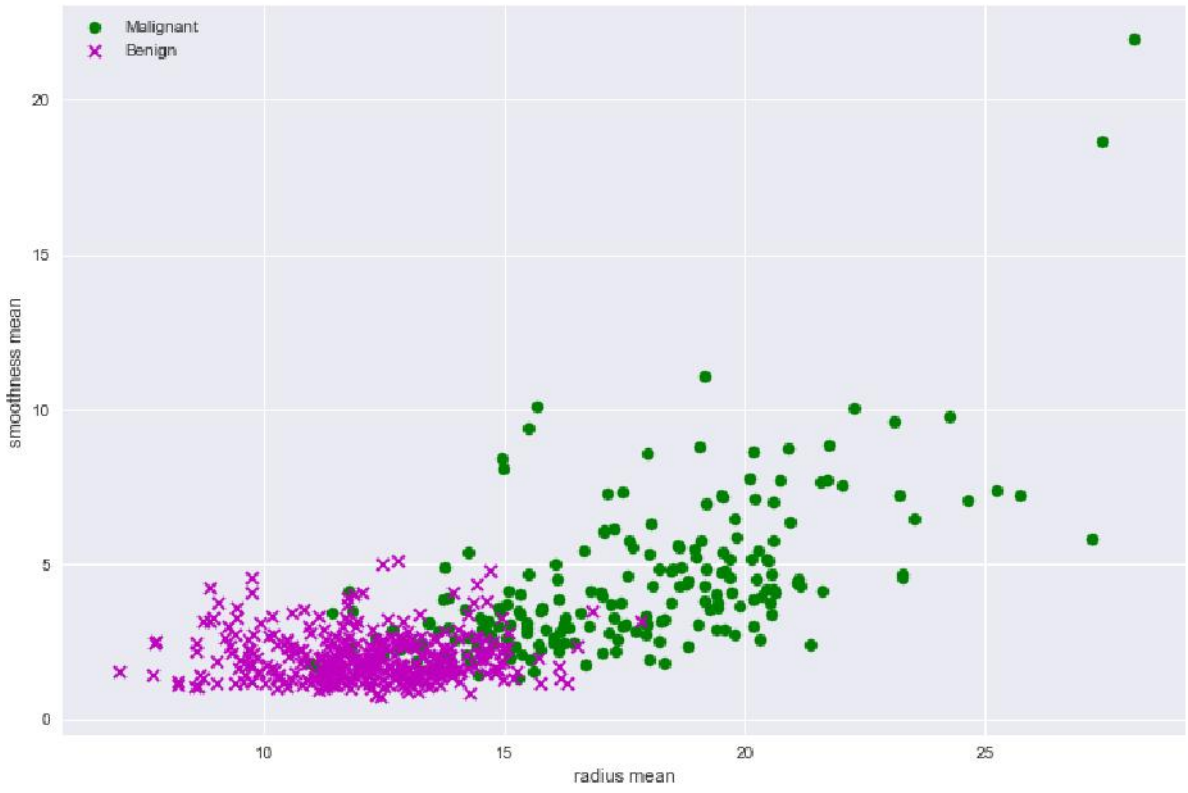
Out[362]: <matplotlib.text.Text at 0x26414d622e8>



In [364]:
```
#randomly choosing columns radius mean and smoothness mean to plot on 2-D axis
 for visual exploration

fig, ax = plt.subplots(figsize=(12,8))
ax.scatter(data_M['radius_mean'], data_M['smoothness_mean'], s=50, c='g', mark
er='o', label='Malignant')
ax.scatter(data_B['radius_mean'], data_B['smoothness_mean'], s=50, c='m', mark
er='x', label='Benign')
ax.legend()
ax.set_xlabel('radius mean')
ax.set_ylabel('smoothness mean')
```

Out[364]: <matplotlib.text.Text at 0x26414e5d4e0>




From the above plots it is quite evident that compactness mean and smoothness mean can be used to chart a separable boundary between Malignant and Benign type of Cancer.

# Missing Values

In [331]:
```
df.isnull().any()
```

Out[331]:
```
diagnosis                  False
radius_mean                False
radius_sd_error            False
radius_worst               False
texture_mean               False
texture_sd_error           False
texture_worst              False
perimeter_mean             False
perimeter_sd_error         False
perimeter_worst            False
area_mean                  False
area_sd_error              False
area_worst                 False
smoothness_mean            False
smoothness_sd_error        False
smoothness_worst           False
compactness_mean           False
compactness_sd_error       False
compactness_worst          False
concavity_mean             False
concavity_sd_error         False
concavity_worst            False
concave_points_mean        False
concave_points_sd_error    False
concave_points_worst       False
symmetry_mean              False
symmetry_sd_error          False
symmetry_worst             False
fractal_dimension_mean     False
fractal_dimension_sd_error False
fractal_dimension_worst    False
dtype: bool
```

# Descriptive statistics

In [332]:
```
df.shape
```

Out[332]:
```
(569, 31)
```

In [333]:
```
df.get_dtype_counts()
```

Out[333]:
```
float64    30
object      1
dtype: int64
```

In [334]:
```
df.describe()
```

Out[334]:

| | radius_mean | radius_sd_error | radius_worst | texture_mean | texture_sd_error | tex |
|---|---|---|---|---|---|---|
| **count** | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569 |
| **mean** | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 | 0.1 |
| **std** | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 | 0.0 |
| **min** | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 | 0.0 |
| **25%** | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 | 0.0 |
| **50%** | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 | 0.0 |
| **75%** | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 | 0.1 |
| **max** | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163400 | 0.3 |

8 rows × 30 columns

In [308]:
```
avg=pd.DataFrame(df.mean())
avg.head()
```

Out[308]:

| | 0 |
|---|---|
| **radius_mean** | 14.127292 |
| **radius_sd_error** | 19.289649 |
| **radius_worst** | 91.969033 |
| **texture_mean** | 654.889104 |
| **texture_sd_error** | 0.096360 |

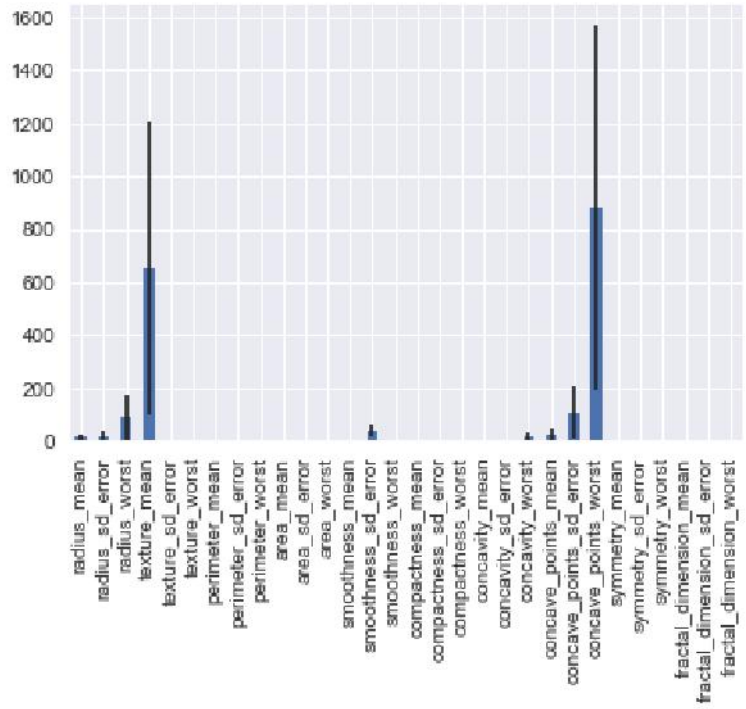
In [309]:
```
median=pd.DataFrame(df.median())
median.head()
```

Out[309]:

| | 0 |
|---|---|
| **radius_mean** | 13.37000 |
| **radius_sd_error** | 18.84000 |
| **radius_worst** | 86.24000 |
| **texture_mean** | 551.10000 |
| **texture_sd_error** | 0.09587 |

In [378]:
```
avg.plot(yerr=median,kind='bar',legend=False)
```

Out[378]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26419dfecf8>
```




In [311]:
```
fea_avg=pd.concat([avg,median],axis=1)
fea_avg.columns=['Average','median']
fea_avg.head()
```

Out[311]:

| | Average | median |
|---|---|---|
| **radius_mean** | 14.127292 | 13.37000 |
| **radius_sd_error** | 19.289649 | 18.84000 |
| **radius_worst** | 91.969033 | 86.24000 |
| **texture_mean** | 654.889104 | 551.10000 |
| **texture_sd_error** | 0.096360 | 0.09587 |

In [312]:
```
x=df.iloc[:,1:31].values
y=df.iloc[:,0:1].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=
0)
```

# Feature Selection

In this method we need to choose how many features we will use. For example, will k (number of features) be 5 or 10 or 15? The answer is only trying or intuitively. I do not try all combinations but I only choose k = 10 and find best 10 features.

In [313]:
```
#Ranking of features by Chi2 method
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# find best scored 5 features
select_feature = SelectKBest(chi2, k=5).fit(x_train, y_train)
print('Score list:', (select_feature.scores_))
```

```
Score list: [  2.10952274e+02   6.86835459e+01   1.58542291e+03   4.35882472e
+04
   1.16602030e-01   4.19953838e+00   1.49994945e+01   8.35975574e+00
   2.06246071e-01   2.15128072e-04   2.88901517e+01   3.42617835e-02
   2.02743541e+02   7.36233168e+03   4.41907010e-03   4.20685619e-01
   5.77229893e-01   1.97996479e-01   3.08390029e-04   3.17821831e-03
   3.96202415e+02   1.35206566e+02   2.92889254e+03   9.29175115e+04
   3.10137553e-01   1.51417061e+01   2.90940473e+01   1.06556712e+01
   1.08184602e+00   1.74011001e-01]
```

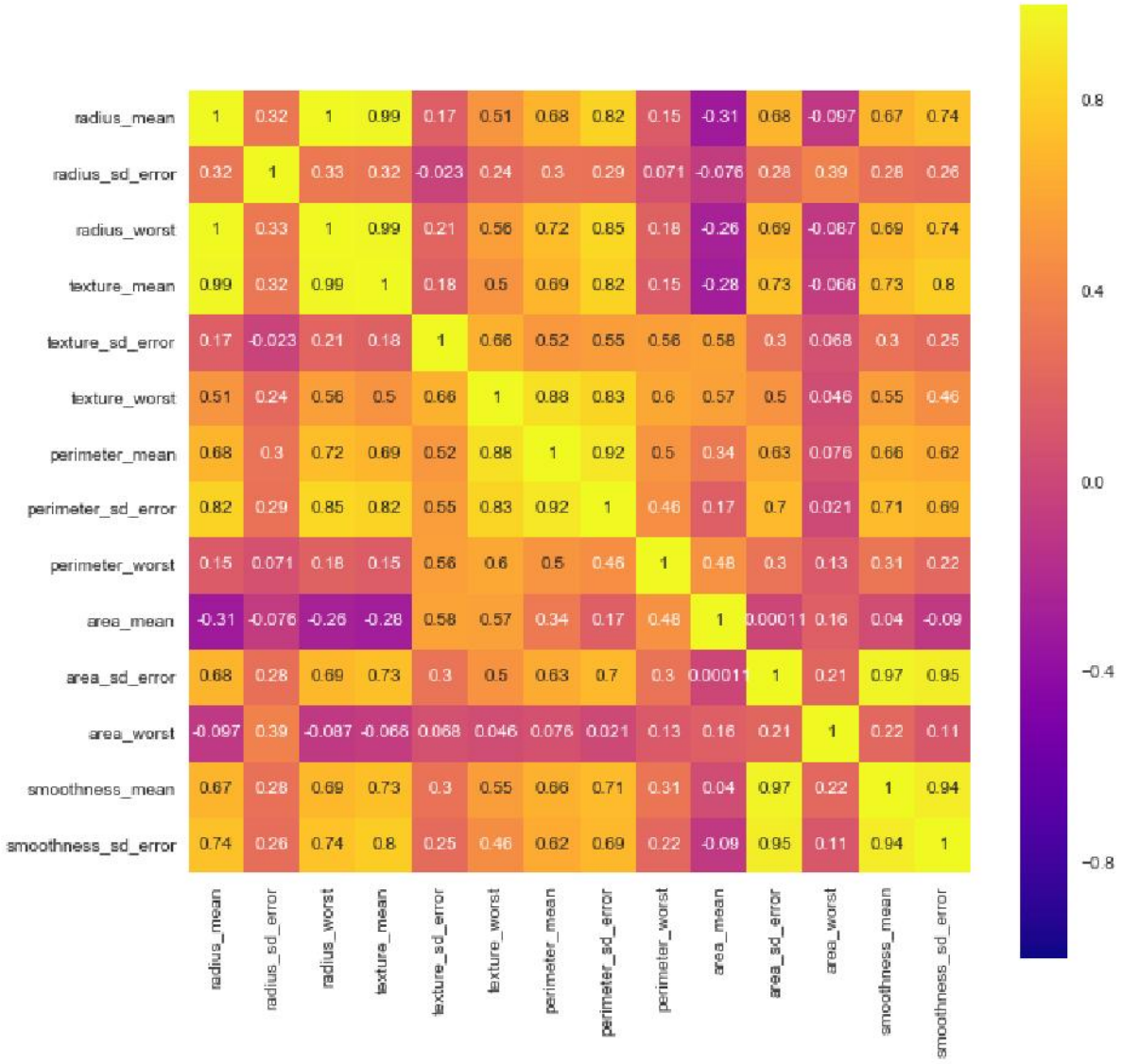
# What if we want to observe all correlation between features?

# HeatMap for finding the Correlation between the features

In [314]:
```
features_mean1= list(df.columns[1:15]) ##First 15 features
features_mean2= list(df.columns[15:30]) ##Next 15 features
```

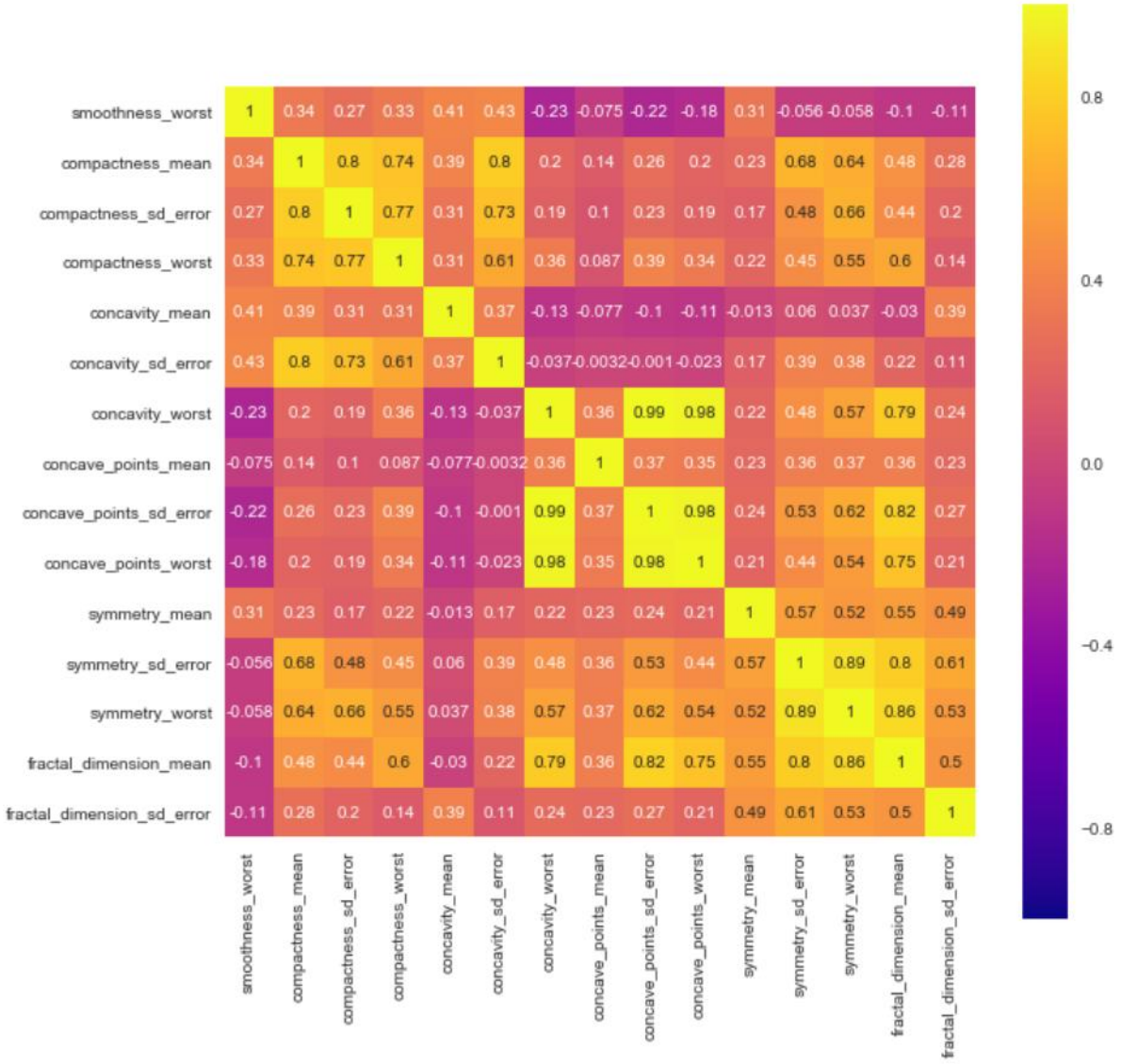
In [373]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df[features_mean1].corr(), annot=True, square=True, cmap='plasma')
#plt.show()
```

Out[373]: <matplotlib.axes._subplots.AxesSubplot at 0x26418b9bb38>



In [374]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df[features_mean2].corr(), annot=True, square=True, cmap='plasma')
#plt.show()
```

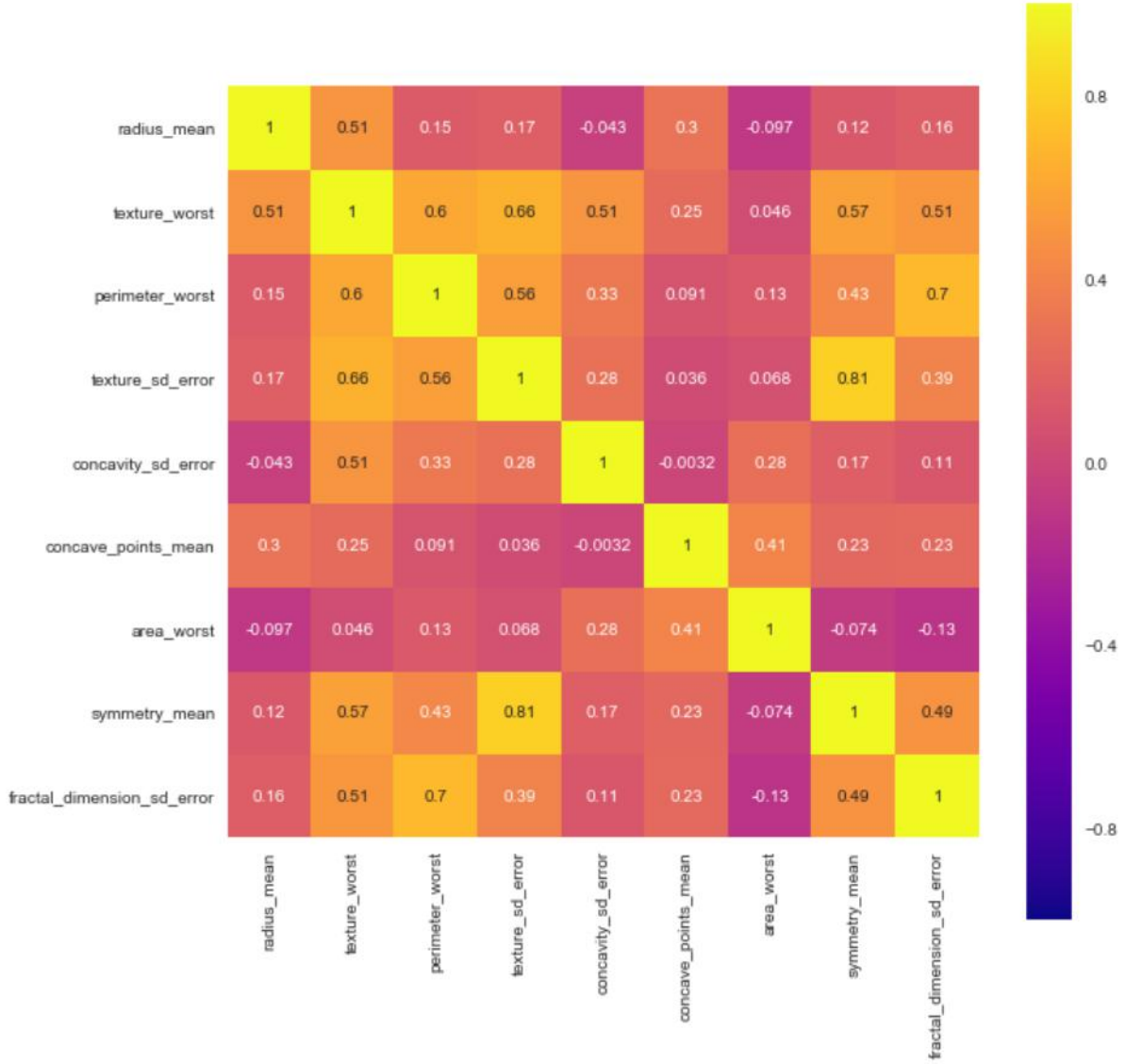
Out[374]: <matplotlib.axes._subplots.AxesSubplot at 0x26419a42780>




As it can be seen in map heat figure radius_mean, texture_mean and radius_worst are correlated with each other so we will use only radius_mean. If you ask how i choose area_mean as a feature to use, well actually there is no correct answer, I just look at swarm plots and area_mean looks like clear for me but we cannot make exact separation among other correlated features without trying.

In [375]:
```
features=list(['radius_mean','texture_worst','perimeter_worst','texture_sd_err
or','concavity_sd_error','concave_points_mean','area_worst','symmetry_mean','f
ractal_dimension_sd_error'])
```

In [376]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df[features].corr(), annot=True, square=True, cmap='plasma')
plt.show()
```

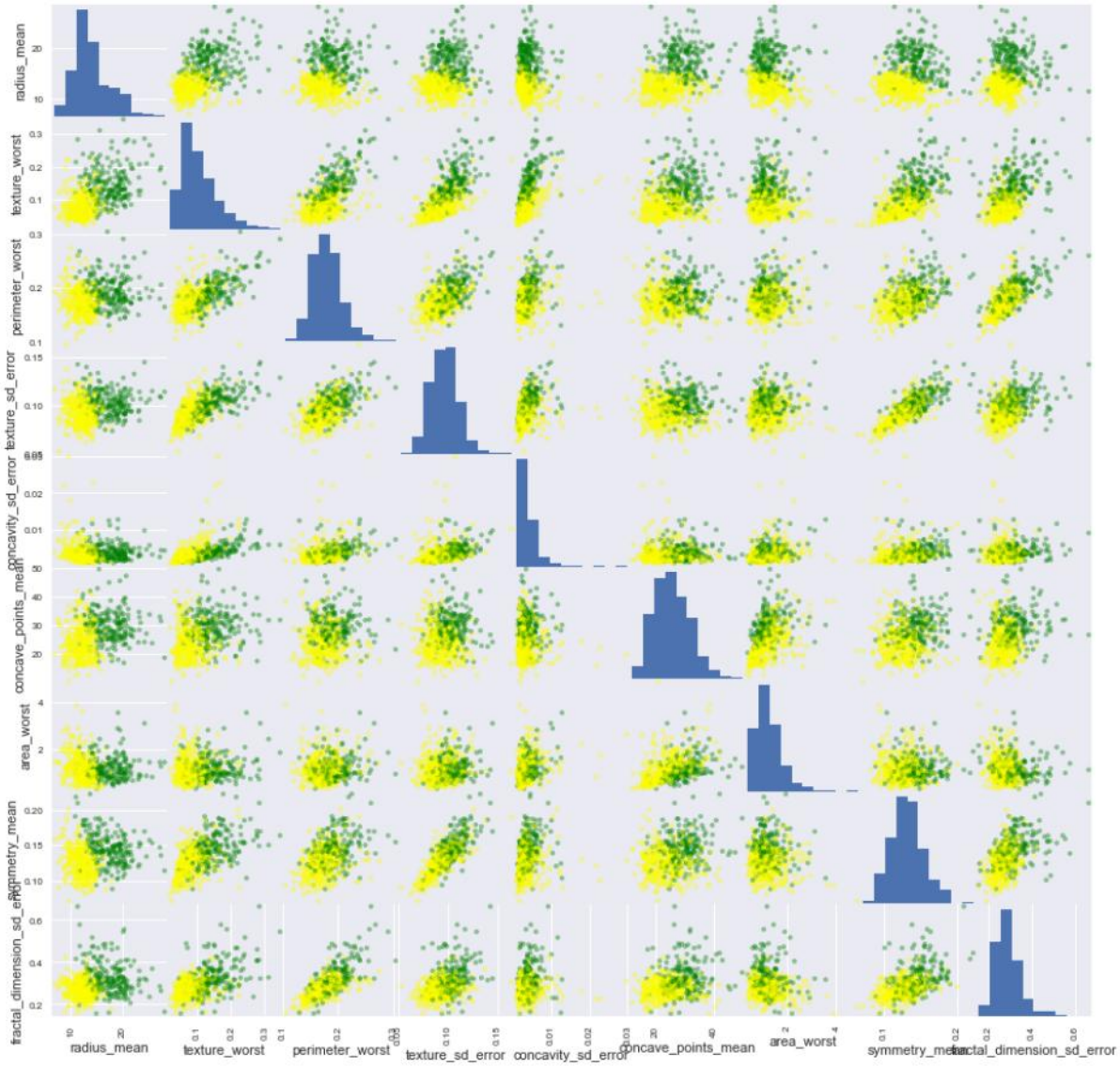



After drop correlated features, as it can be seen in below correlation matrix, there are no more correlated features.

In [370]:
```
color_dic = {'M':'green', 'B':'yellow'}
colors = df['diagnosis'].map(lambda x: color_dic.get(x))

sm = pd.scatter_matrix(df[features], c=colors, alpha=0.4, figsize=((15,15)));

plt.show()
```




** 1. Radius, area and perimeter have a strong linear relationship as expected. This graph shows the features like as texture_mean, smoothness_mean, symmetry_mean and fractal_dimension_mean cant be used for classify two category because both category are mixed there is no separable plane

Best 10 features to classify is that radius_mean, radius_worst, radius_sd_error, texture_worst,perimeter_worst,perimeter_mean,texture_sd_error,concavity_sd_error,concave_points_mean and symmetry_mean. So lets se what happens if we use only these best scored 10 feature.

In [320]:
```
x1=df[['radius_mean','radius_worst','radius_sd_error','texture_worst','perimet
er_worst','perimeter_mean','texture_sd_error','concavity_sd_error','concave_po
ints_mean','area_worst','symmetry_mean']]
x=df.iloc[:,1:31].values
y=df.iloc[:,0:1].values
x1
```

Out[320]:

| | radius_mean | radius_worst | radius_sd_error | texture_worst | perimeter_worst | perim |
|---|---|---|---|---|---|---|
| **0** | 17.990 | 122.80 | 10.38 | 0.27760 | 0.2419 | 0.300 |
| **1** | 20.570 | 132.90 | 17.77 | 0.07864 | 0.1812 | 0.086 |
| **2** | 19.690 | 130.00 | 21.25 | 0.15990 | 0.2069 | 0.197 |
| **3** | 11.420 | 77.58 | 20.38 | 0.28390 | 0.2597 | 0.241 |
| **4** | 20.290 | 135.10 | 14.34 | 0.13280 | 0.1809 | 0.198 |
| **5** | 12.450 | 82.57 | 15.70 | 0.17000 | 0.2087 | 0.157 |
| **6** | 18.250 | 119.60 | 19.98 | 0.10900 | 0.1794 | 0.112 |
| **7** | 13.710 | 90.20 | 20.83 | 0.16450 | 0.2196 | 0.093 |
| **8** | 13.000 | 87.50 | 21.82 | 0.19320 | 0.2350 | 0.185 |
| **9** | 12.460 | 83.97 | 24.04 | 0.23960 | 0.2030 | 0.227 |
| **10** | 16.020 | 102.70 | 23.24 | 0.06669 | 0.1528 | 0.032 |
| **11** | 15.780 | 103.60 | 17.89 | 0.12920 | 0.1842 | 0.099 |
| **12** | 19.170 | 132.40 | 24.80 | 0.24580 | 0.2397 | 0.206 |
| **13** | 15.850 | 103.70 | 23.95 | 0.10020 | 0.1847 | 0.099 |
| **14** | 13.730 | 93.60 | 22.61 | 0.22930 | 0.2069 | 0.212 |
| **15** | 14.540 | 96.73 | 27.54 | 0.15950 | 0.2303 | 0.163 |
| **16** | 14.680 | 94.74 | 20.13 | 0.07200 | 0.1586 | 0.073 |
| **17** | 16.130 | 108.10 | 20.68 | 0.20220 | 0.2164 | 0.172 |
| **18** | 19.810 | 130.00 | 22.15 | 0.10270 | 0.1582 | 0.147 |
| **19** | 13.540 | 87.46 | 14.36 | 0.08129 | 0.1885 | 0.066 |
| **20** | 13.080 | 85.63 | 15.71 | 0.12700 | 0.1967 | 0.045 |
| **21** | 9.504 | 60.34 | 12.44 | 0.06492 | 0.1815 | 0.029 |
| **22** | 15.340 | 102.50 | 14.26 | 0.21350 | 0.2521 | 0.207 |
| **23** | 21.160 | 137.20 | 23.04 | 0.10220 | 0.1769 | 0.109 |
| **24** | 16.650 | 110.00 | 21.38 | 0.14570 | 0.1995 | 0.152 |
| **25** | 17.140 | 116.00 | 16.40 | 0.22760 | 0.3040 | 0.222 |
| **26** | 14.580 | 97.41 | 21.53 | 0.18680 | 0.2252 | 0.142 |
| **27** | 18.610 | 122.10 | 20.25 | 0.10660 | 0.1697 | 0.149 |
| **28** | 15.300 | 102.40 | 25.27 | 0.16970 | 0.1926 | 0.168 |
| **29** | 17.570 | 115.00 | 15.05 | 0.11570 | 0.1739 | 0.098 |
| **...** | ... | ... | ... | ... | ... | ... |
| **539** | 7.691 | 48.34 | 25.44 | 0.11990 | 0.2037 | 0.092 |

| | radius_mean | radius_worst | radius_sd_error | texture_worst | perimeter_worst | perim |
|---|---|---|---|---|---|---|
| **540** | 11.540 | 74.65 | 14.44 | 0.11200 | 0.1818 | 0.0673 |
| **541** | 14.470 | 95.81 | 24.99 | 0.12300 | 0.1872 | 0.1009 |
| **542** | 14.740 | 94.70 | 25.42 | 0.07214 | 0.1840 | 0.0410 |
| **543** | 13.210 | 84.88 | 28.06 | 0.06877 | 0.1628 | 0.0298 |
| **544** | 13.870 | 89.77 | 20.70 | 0.10180 | 0.1620 | 0.0368 |
| **545** | 13.620 | 87.19 | 23.23 | 0.06747 | 0.1664 | 0.0297 |
| **546** | 10.320 | 65.31 | 16.35 | 0.04994 | 0.1885 | 0.0101 |
| **547** | 10.260 | 65.85 | 16.58 | 0.08066 | 0.1669 | 0.0435 |
| **548** | 9.683 | 61.05 | 19.34 | 0.05030 | 0.1580 | 0.0233 |
| **549** | 10.820 | 68.89 | 24.21 | 0.06602 | 0.1976 | 0.0154 |
| **550** | 10.860 | 68.51 | 21.48 | 0.04227 | 0.1661 | 0.0000 |
| **551** | 11.130 | 71.49 | 22.44 | 0.08194 | 0.2030 | 0.0482 |
| **552** | 12.770 | 81.35 | 29.43 | 0.04234 | 0.1539 | 0.0199 |
| **553** | 9.333 | 59.01 | 21.94 | 0.05605 | 0.1692 | 0.0399 |
| **554** | 12.880 | 82.50 | 28.92 | 0.05824 | 0.1566 | 0.0619 |
| **555** | 10.290 | 65.67 | 27.61 | 0.07658 | 0.1593 | 0.0599 |
| **556** | 10.160 | 64.73 | 19.59 | 0.07504 | 0.1791 | 0.0050 |
| **557** | 9.423 | 59.26 | 27.88 | 0.04971 | 0.1742 | 0.0000 |
| **558** | 14.590 | 96.39 | 22.68 | 0.13300 | 0.1454 | 0.1029 |
| **559** | 11.510 | 74.52 | 23.93 | 0.10210 | 0.1388 | 0.1112 |
| **560** | 14.050 | 91.38 | 27.15 | 0.11260 | 0.1537 | 0.0446 |
| **561** | 11.200 | 70.67 | 29.37 | 0.03558 | 0.1060 | 0.0000 |
| **562** | 15.220 | 103.40 | 30.62 | 0.20870 | 0.2128 | 0.2550 |
| **563** | 20.920 | 143.00 | 25.09 | 0.22360 | 0.2149 | 0.3174 |
| **564** | 21.560 | 142.00 | 22.39 | 0.11590 | 0.1726 | 0.2439 |
| **565** | 20.130 | 131.20 | 28.25 | 0.10340 | 0.1752 | 0.1440 |
| **566** | 16.600 | 108.30 | 28.08 | 0.10230 | 0.1590 | 0.0925 |
| **567** | 20.600 | 140.10 | 29.33 | 0.27700 | 0.2397 | 0.3514 |
| **568** | 7.760 | 47.92 | 24.54 | 0.04362 | 0.1587 | 0.0000 |

569 rows × 11 columns

In [361]:
```
#Encoding Y the categorical variable to 1 & 0
LabelEncoder_X=LabelEncoder()
y=LabelEncoder_X.fit_transform(y)
y
```

Out[361]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0], dtype=int64)
```

# Do we need Feature Scaling?

We should know something like variance, standart deviation, number of sample (count) or max min values. These type of information helps to understand about what is going on data. For example , the question is appeared in my mind the radius_worst feature's max value is 188.50 and concavity_sd_error features' max 0.029840. So defenitly we need Feature Scaling

In [340]:
```
x1.describe()
```

Out[340]:

| | radius_mean | radius_worst | radius_sd_error | texture_worst | perimeter_worst | peri |
|---|---|---|---|---|---|---|
| **count** | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569 |
| **mean** | 14.127292 | 91.969033 | 19.289649 | 0.104341 | 0.181162 | 0.08 |
| **std** | 3.524049 | 24.298981 | 4.301036 | 0.052813 | 0.027414 | 0.07 |
| **min** | 6.981000 | 43.790000 | 9.710000 | 0.019380 | 0.106000 | 0.00 |
| **25%** | 11.700000 | 75.170000 | 16.170000 | 0.064920 | 0.161900 | 0.02 |
| **50%** | 13.370000 | 86.240000 | 18.840000 | 0.092630 | 0.179200 | 0.06 |
| **75%** | 15.780000 | 104.100000 | 21.800000 | 0.130400 | 0.195700 | 0.13 |
| **max** | 28.110000 | 188.500000 | 39.280000 | 0.345400 | 0.304000 | 0.42 |

In [341]:
```
x_train,x_test,y_train,y_test=train_test_split(x1,y,test_size=0.2,random_state
=0)
```

In [342]:
```
sc_x=StandardScaler()
x_train=sc_x.fit_transform(x_train)
x_test=sc_x.transform(x_test)
x_train
```

Out[342]:
```
array([[-1.15036482, -1.12855021, -0.39064196, ...,  0.21353282,
         1.48720153,  0.34249851],
       [-0.93798972, -0.94820146,  0.68051405, ...,  1.06684183,
         0.75941203, -0.09553745],
       [ 0.574121  ,  0.51394098, -1.03333557, ..., -0.97781783,
        -1.05784511, -0.52472419],
       ...,
       [-1.32422924, -1.31754581, -0.20048168, ..., -0.08512533,
         0.06856616,  0.03720072],
       [-1.24380987, -1.28007609, -0.2245526 , ..., -0.77269547,
         6.78861237, -2.71180676],
       [-0.73694129, -0.71226578,  1.14989702, ...,  1.90702301,
         3.12934586, -0.11766047]])
```

# Model Classification

In [387]:
```
classifier=SVC()
classifier.fit(x_train,y_train)
classifier_score = classifier.score(x_test, y_test)
classifier_score
```

Out[387]:
```
0.94736842105263153
```

The SVM is working fine with good parameter it shows us what is the use of running of parameters In the first by using default I was getting only 80 % accuracy. The results show a similar tight distribution for all classifiers except SVM which is encouraging, suggesting low variance. The poor results for SVM are surprising. But with tuned parameter it is 95 %.

In [389]:
```
model=RandomForestClassifier(n_estimators=100)
model.fit(x_train,y_train)
prediction = model.predict(x_test)
metrics.accuracy_score(prediction,y_test)
```

Out[389]:
```
0.96491228070175439
```

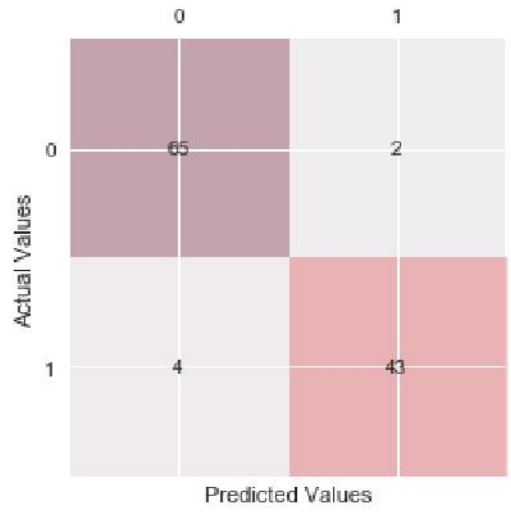
We get better accuracy when we apply Random Forest classifier. An advantage with Random Forest is that it returns a feature importance matrix which can be used to select features. So lets select the top 10 features and use them as predictors.

# Evaluation

In [355]:
```
y_pred = classifier.fit(x_train, y_train).predict(x_test)
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

Out[355]:
```
array([[65,  2],
       [ 4, 43]])
```

In [356]:
```
fig, ax = plt.subplots(figsize=(4,4))
ax.matshow(cm, cmap=plt.cm.Reds, alpha=0.3)
for i in range(cm.shape[0]):
     for j in range(cm.shape[1]):
         ax.text(x=j, y=i,
                 s=cm[i, j],
                 va='center', ha='center')
plt.xlabel('Predicted Values', )
plt.ylabel('Actual Values')
plt.show()
print(classification_report(y_test, y_pred ))
```




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.94 | 0.97 | 0.96 | 67 |
| 1 | 0.96 | 0.91 | 0.93 | 47 |
| avg / total | 0.95 | 0.95 | 0.95 | 114 |

Accuracy is almost 95% and as it can be seen in confusion matrix, we make few wrong prediction. What we did up to now is that we choose features according to correlation matrix

# Comparing with other Models

In [390]:
```
models = []
models.append(( 'LR' , LogisticRegression()))
models.append(( 'KNN' , KNeighborsClassifier()))
models.append(( 'CART' , DecisionTreeClassifier()))
models.append(( 'SVM' , SVC()))
```

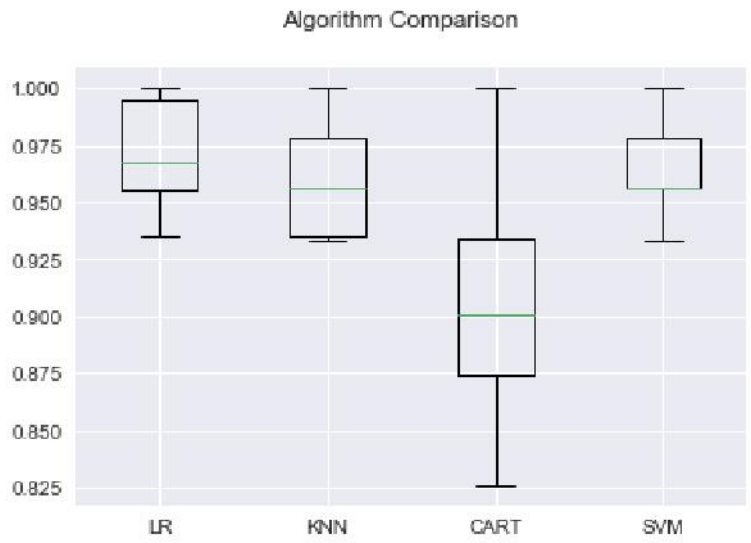
# Test options and evaluation metric

In [392]:
```
num_folds = 10
num_instances = len(x_train)
seed = 7
scoring =  'accuracy'

num_folds = 10
num_instances = len(x_train)
seed = 7
scoring =  'accuracy'
results = []
names = []
for name, model in models:
 kfold = KFold(n=num_instances, n_folds=num_folds, random_state=seed)
 cv_results = cross_val_score(model, x_train, y_train, cv=kfold, scoring=scori
ng)
 results.append(cv_results)
 names.append(name)
 msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
 print(msg)
print('-> 10-Fold cross-validation accurcay score for the training data for si
x classifiers')
```

```
LR: 0.969372 (0.024315)
KNN: 0.958261 (0.022975)
CART: 0.905652 (0.045863)
SVM: 0.964928 (0.022383)
-> 10-Fold cross-validation accurcay score for the training data for six clas
sifiers
```

In [393]:
```
fig = plt.figure()
fig.suptitle( 'Algorithm Comparison' )
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



# Conclusion

Shortly, I tried to show importance of feature selection and data visualization. Default data includes 33 feature but after feature selection we drop this number from 33 to 10 with accuracy 95%. In this kernel we just tried basic things, I am sure with these data visualization and feature selection methods, you can easily ecxeed the % 95 accuracy.

# Part 2: Feedback

In [ ]:

```
#!/usr/bin/env python

import pandas as pd
import numpy as np
from sklearn import LinearRegression

##It will show an error because it should be sklearn.linear_model

from sklearn.cross_validation import cross_val_score

# Load data
#Variable should have proper naming eg: "d" -> dataset
d = pd.read_csv('../data/train.csv')

#### You may try some EDA before you start building your data, this helps unde
rstand your data better.
####1) Finding the Missing Values
####2) Predict the variables that are highly correlated
####3) Normalize the data using Feature Scaling
####4) If you find any catogerical data encode those datas.
##eg: 2 catogerical data:Label encoder
##    more than 2 catogerical data:One hot encoder

# Setup data for prediction
x1 = data.SalaryNormalized
x2 = pd.get_dummies(data.ContractType)

# Setup model
model = LinearRegression()
### you need to fit your model before evaluation\

# Evaluate model
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import train_test_split

##Always keep your imports at the top of the script so it can lead to confusio
n.
###the cross validation package was imported twice in this script

scores = cross_val_score(model, x2, x1, cv=1, scoring='mean_absolute_error')
### We are not splitting datas into training and test set
### K-fold cross validation of 1 is not very useful

print(scores.mean())
###There are a few mistakes here which would be best solved through practice .


###This script shows a general understanding of the tools necessary to build a
 linear regression analysis.
###There are a few mistakes here which would be best solved through practice o
f best practices of organization.
```

In [ ]:

```
#!/usr/bin/env python

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.cross_validation import cross_val_score
### Well formatted document

# Load data
data = pd.read_csv('../data/train.csv')
#### You may try some EDA before you start building your data, this helps unde
rstand your data better.
#### 1) Finding the Missing Values
#### 2) Predict the variables that are highly correlated
#### 3) Normalize the data using Feature Scaling
#### 4) If you find any catogerical data encode those datas.

# Setup data for prediction
y = data.SalaryNormalized
X = pd.get_dummies(data.ContractType)

# Setup model
model = LinearRegression()
### you need to fit your model before evaluation
### lables are clearly defined

# Evaluate model
scores = cross_val_score(model, X, y, cv=5, scoring='mean_absolute_error')
print(scores.mean())

### The workflow and code is well formatted
### K-fold of 5 will give reliable test results.
```