In [1]:
```
import numpy as no
import matplotlib.pyplot as plt
import pandas as pd


data = pd.read_csv('/Users/carlosquispe/Downloads/datasets-180-408-data.
```

Exploration of the data

In [2]:
```
data.head(10)
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
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	843786	M	12.45	15.70	82.57	477.1	0.12
6	844359	M	18.25	19.98	119.60	1040.0	0.09
7	84458202	M	13.71	20.83	90.20	577.9	0.11
8	844981	M	13.00	21.82	87.50	519.8	0.12
9	84501001	M	12.46	24.04	83.97	475.9	0.11

10 rows × 33 columns

Rows and columns in the data

In [3]:
```
print(data.shape)
```

```
(569, 33)
```

Statistics

In [4]:
```
data.describe()
```

Out[4]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.00000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.09636
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.01406
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.05263
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.08637
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.09587
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.10530
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.16340

8 rows × 32 columns

The 'diagnosis' column contains values of M= malign and B= benign. We need to transform those values to 1 and 0 respectively

In [5]:
```
data['diagnosis'] = data['diagnosis'].apply(lambda x: '1' if x == 'M' el
data = data.set_index('id')
```

We do not need the last column 'Unnamed:32'

In [6]:
```
del data['Unnamed: 32']
```

In [7]:
```
data.head(5)
```

Out[7]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
842302	1	17.99	10.38	122.80	1001.0	0.11840
842517	1	20.57	17.77	132.90	1326.0	0.08474
84300903	1	19.69	21.25	130.00	1203.0	0.10960
84348301	1	11.42	20.38	77.58	386.1	0.14250
84358402	1	20.29	14.34	135.10	1297.0	0.10030

5 rows × 31 columns

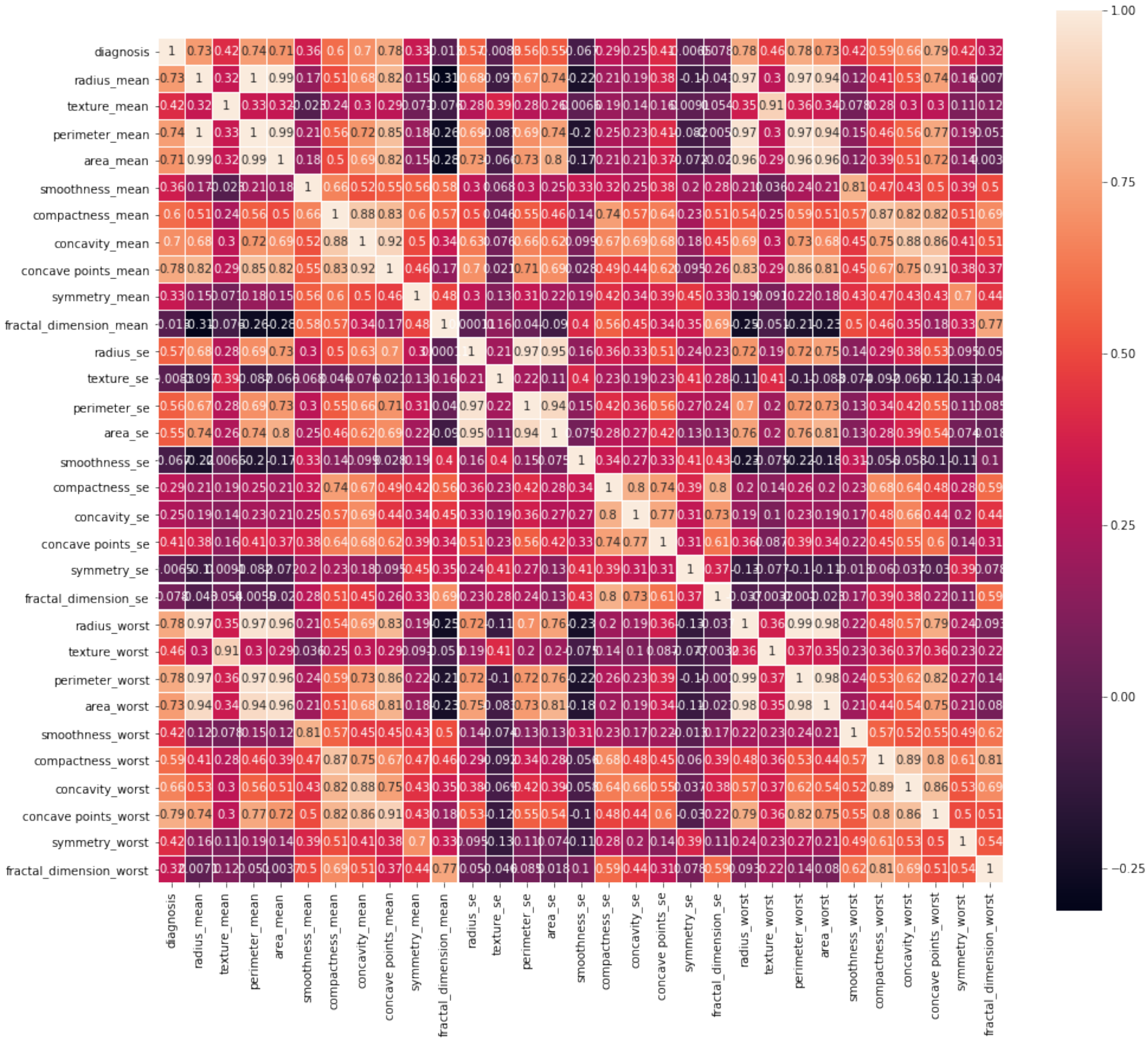
To know the amount fo M (1) and B (0) in the data

In [8]:
```
print (data.groupby('diagnosis').size())
```

```
diagnosis
0    357
1    212
dtype: int64
```

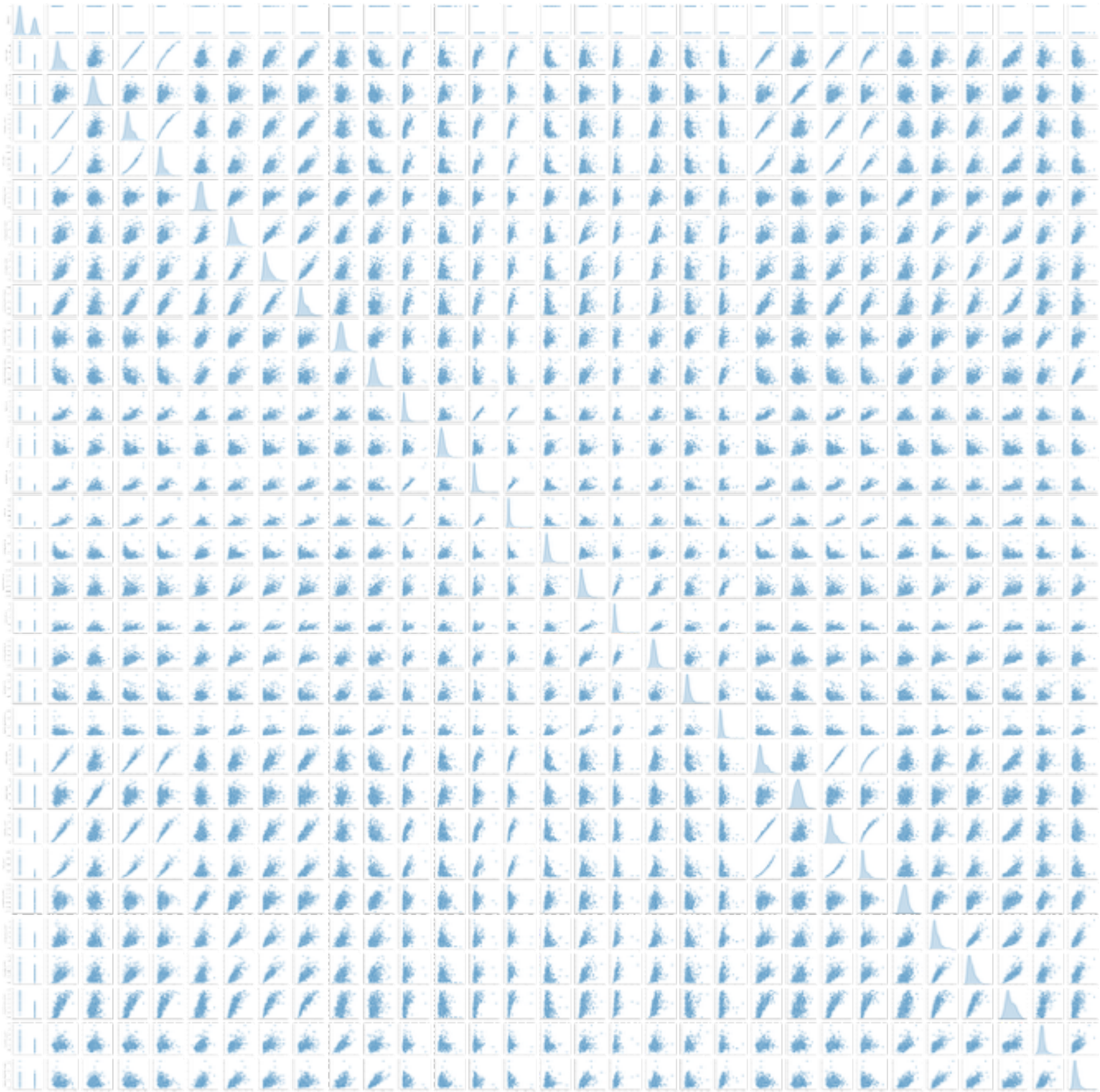
To know the attributes correlation

In [9]:
```
import seaborn as sns
plt.figure(figsize=(16,14))
sns.heatmap(data.astype(float).corr(), linewidths=0.1, square=True, anno
plt.show()
```



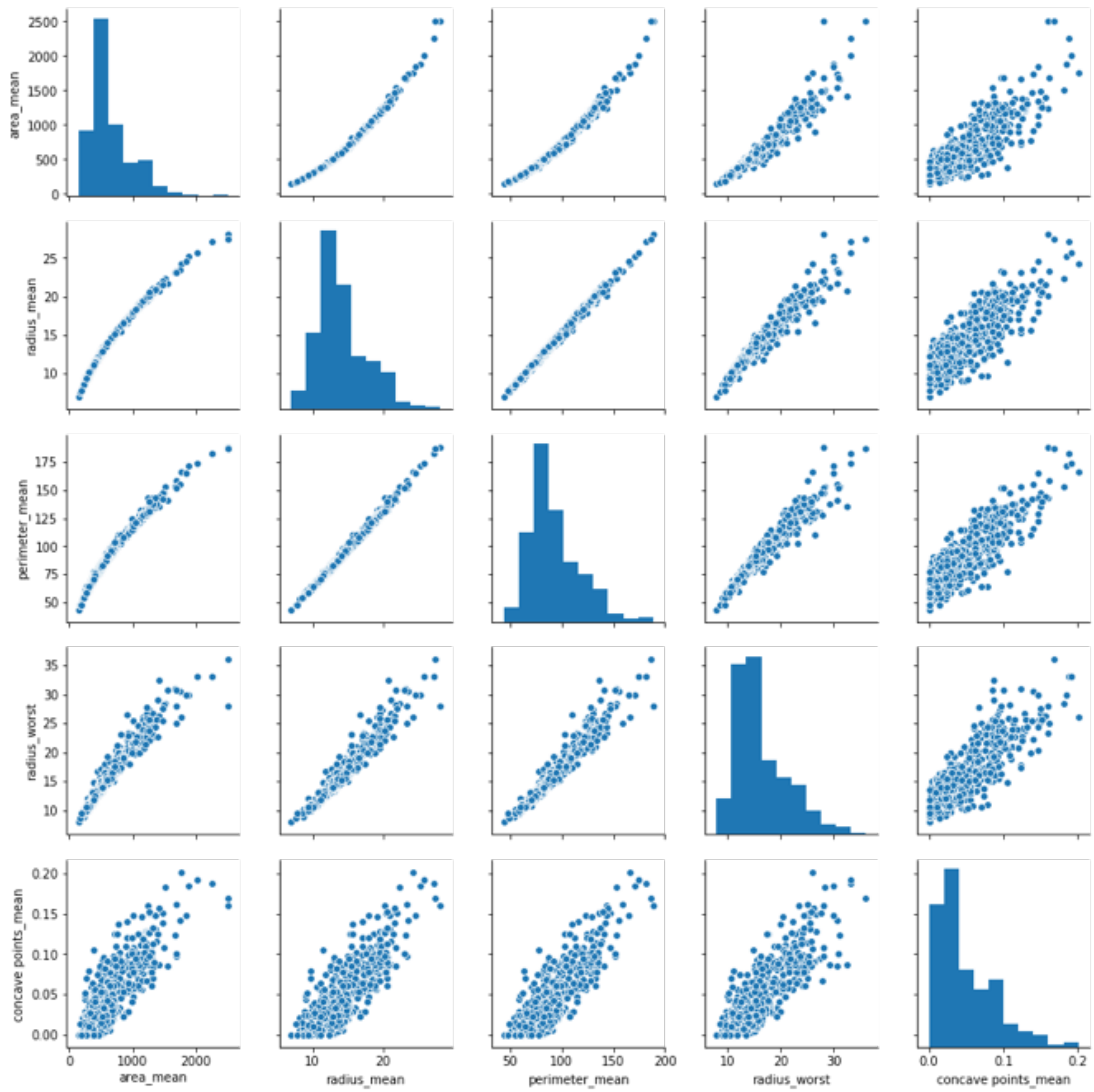
In [45]:
```
sns.pairplot(data, diag_kind='kde')
```

Out[45]: <seaborn.axisgrid.PairGrid at 0x13e88a3c8>



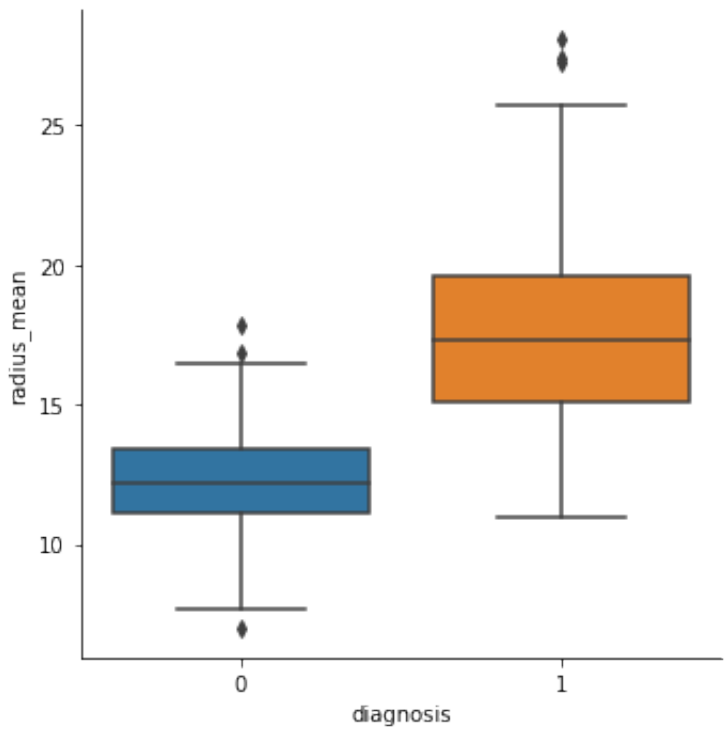
In [10]:
```
a, vars=["area_mean", 'radius_mean', 'perimeter_mean', 'radius_worst', 'c
```

Out[10]: <seaborn.axisgrid.PairGrid at 0x12ea7c5c0>



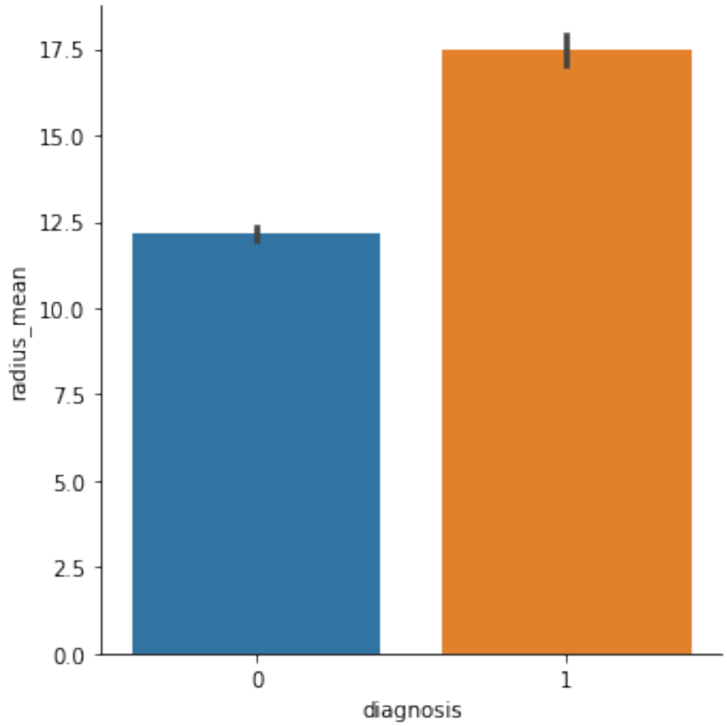
In [11]:
```
sns.catplot(x= 'diagnosis', y = 'radius_mean', data = data, kind = 'box'
```

Out[11]: <seaborn.axisgrid.FacetGrid at 0x12e69e940>



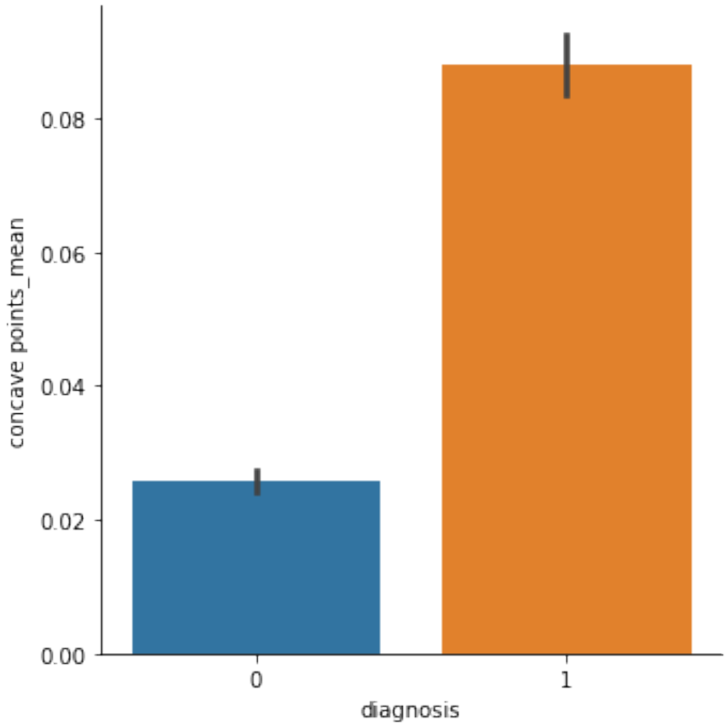
In [12]:
```
sns.catplot(x= 'diagnosis', y = 'radius_mean', data = data, kind = 'bar'
```

Out[12]: <seaborn.axisgrid.FacetGrid at 0x12e728e10>



In [13]:
```
sns.catplot(x= 'diagnosis', y = 'concave points_mean', data = data, kind
```

Out[13]: <seaborn.axisgrid.FacetGrid at 0x110304fd0>



In [16]:
```
print(data.shape)
```

(569, 31)

In [19]:
```
data.head(5)
```

Out[19]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
842302	1	17.99	10.38	122.80	1001.0	0.11840
842517	1	20.57	17.77	132.90	1326.0	0.08474
84300903	1	19.69	21.25	130.00	1203.0	0.10960
84348301	1	11.42	20.38	77.58	386.1	0.14250
84358402	1	20.29	14.34	135.10	1297.0	0.10030

5 rows × 31 columns

In [23]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
import time
```

In [24]:
```
X = data.drop('diagnosis', axis=1).values
Y = data['diagnosis'].values
```

Splitting the dataset intro the training set and Test set

In [25]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X,Y, test_size=0.20,
```

Algorithm Checking in training set

Building a model to predict if a given set of symptoms leat to breast cancer Try it with: Classification and Regression Trees (CART), Linear Support Vector Machines (SVM), Gaussian Naive (NB), and K-Nearest Neighbors (KNN)

In [26]:
```
models_list = []
models_list.append(('CART', DecisionTreeClassifier()))
models_list.append(('SVM', SVC()))
models_list.append(('NB', GaussianNB()))
models_list.append(('KNN', KNeighborsClassifier()))
```

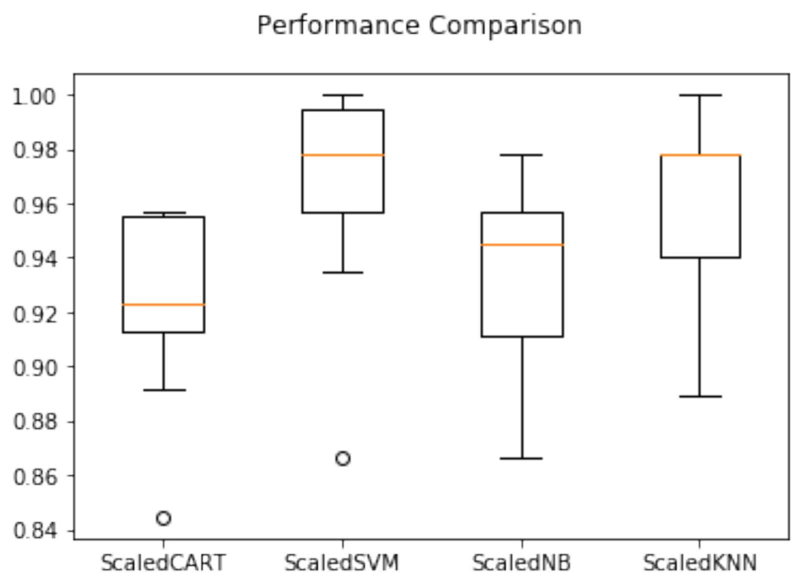
Standardizing the data using pipelines and applying the algorithms

In [29]:

```
import warnings

# Standardize the dataset
pipelines = []

pipelines.append(('ScaledCART', Pipeline([('Scaler', StandardScaler()),(

pipelines.append(('ScaledSVM', Pipeline([('Scaler', StandardScaler()),('
pipelines.append(('ScaledNB', Pipeline([('Scaler', StandardScaler()),('N
                                                                     Ga
pipelines.append(('ScaledKNN', Pipeline([('Scaler', StandardScaler()),('
                                                                      K

results = []
names = []
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    kfold = KFold(n_splits=num_folds, random_state=123)
    for name, model in pipelines:
        start = time.time()
        cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
        end = time.time()
        results.append(cv_results)
        names.append(name)
        print( "%s: %f (%f) (run time: %f)" % (name, cv_results.mean(),
```

```
ScaledCART: 0.923140 (0.034472) (run time: 0.256537)
ScaledSVM: 0.964879 (0.038621) (run time: 0.056662)
ScaledNB: 0.931932 (0.038625) (run time: 0.032737)
ScaledKNN: 0.958357 (0.038595) (run time: 0.062349)
```

In [30]:
```
fig = plt.figure()
fig.suptitle('Performance Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Best results with SVM (Support Vector Machine), tuning before applying to test set

c_values and kernel_values

In [31]:
```
scaler = StandardScaler().fit(X_train)
rescaledX = scaler.transform(X_train)
c_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
param_grid = dict(C=c_values, kernel=kernel_values)
model = SVC()
kfold = KFold(n_splits=num_folds, random_state=21)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='acc
grid_result = grid.fit(rescaledX, Y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_p
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

...

Applying tuned SVM on Test Set

In [32]:
```
# prepare the model
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
model = SVC(C=2.0, kernel='rbf')
start = time.time()
model.fit(X_train_scaled, Y_train)
end = time.time()
print( "Run Time: %f" % (end-start))
```

```
Run Time: 0.004648
```

In [33]:
```
# estimate accuracy on test dataset
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    X_test_scaled = scaler.transform(X_test)
predictions = model.predict(X_test_scaled)
```

In [34]:
```
print("Accuracy score %f" % accuracy_score(Y_test, predictions))
print(classification_report(Y_test, predictions))
```

```
Accuracy score 0.991228
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        75
           1       0.97      1.00      0.99        39

   micro avg       0.99      0.99      0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114
```

In [35]:
```
print(confusion_matrix(Y_test, predictions))
```

```
[[74  1]
 [ 0 39]]
```

Result in test set: Accuracy = 99.12%

From the confussion matrix: 1 case of mis-classification