Mid-Term Examination Project Code

1. Identify the problem

Breast cancer is the most common malignancy among women, accounting for nearly 1 in 3 cancers diagnosed among women in the United States, and it is the second leading cause of cancer death among women. Breast Cancer occurs as a results of abnormal growth of cells in the breast tissue, commonly referred to as a Tumor. A tumor does not mean cancer - tumors can be benign (not cancerous), pre-malignant (pre-cancerous), or malignant (cancerous). Tests such as MRI, mammogram, ultrasound and biopsy are commonly used to diagnose breast cancer performed.

1.1 Expected outcome

Given breast cancer results from breast fine needle aspiration (FNA) test (is a quick and simple procedure to perform, which removes some fluid or cells from a breast lesion or cyst (a lump, sore or swelling) with a fine needle similar to a blood sample needle). Since this build a model that can classify a breast cancer tumor using two training classification:

- 1= Malignant (Cancerous) - Present
- 0= Benign (Not Cancerous) -Absent

Getting Started: Load libraries and set options

In [1]:
```
#load libraries
import numpy as np          # linear algebra
import pandas as pd         # data processing, CSV file I/O (e.g. pd
.read_csv)

# Read the file "data.csv" and print the contents.
#!cat data/data.csv
data = pd.read_csv('data/data.csv', index_col=False,)
```

Load Dataset

First, load the supplied CSV file using additional options in the Pandas read_csv function.

Inspecting the data

The first step is to visually inspect the new data set. There are multiple ways to achieve this:

- The easiest being to request the first few records using the DataFrame data.head()* method. By default, “data.head()” returns the first 5 rows from the DataFrame object df (excluding the header row).
- Alternatively, one can also use “df.tail()” to return the five rows of the data frame.
- For both head and tail methods, there is an option to specify the number of records by including the required number in between the parentheses when calling either method.Inspecting the data

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

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.8	1001.0	
1	842517	M	20.57	17.77	132.9	1326.0	

2 rows × 32 columns

You can check the number of cases, as well as the number of fields, using the shape method, as shown below.

In [3]:
```
# Id column is redundant and not useful, we want to drop it
data.drop('id', axis =1, inplace=True)
#data.drop('Unnamed: 0', axis=1, inplace=True)
data.head(2)
```

Out[3]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	M	17.99	10.38	122.8	1001.0	0.11840
1	M	20.57	17.77	132.9	1326.0	0.08474

2 rows × 31 columns

In [4]:
```
data.shape
```

Out[4]:
```
(569, 31)
```

In the result displayed, we can see the data has 569 records, each with 32 columns.

The **"info()"** method provides a concise summary of the data; from the output, it provides the type of data in each column, the number of non-null values in each column, and how much memory the data frame is using.

The method **get_dtype_counts()** will return the number of columns of each type in a DataFrame:

In [5]:
```
# Review data types with "info()".
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                569 non-null    object 
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

From the above results, from the 32, variables,column id number 1 is an integer, diagnosis 569 non-null object. and rest are float.

In [7]:
```
#check for missing variables
#data.isnull().any()
```

In [9]:
```
data.diagnosis.unique()
```

Out[9]:
```
array(['M', 'B'], dtype=object)
```

From the results above, diagnosis is a categorical variable, because it represents a fix number of possible values (i.e, Malignant, of Benign. The machine learning algorithms wants numbers, and not strings, as their inputs so we need some method of coding to convert them.

In [10]:
```
#save the cleaner version of dataframe for future analyis
data.to_csv('data/clean-data.csv')
```

2.0 Notebook 2: Exploratory Data Analysis

This step involves taking a closer look at attributes and data values. In this section, we are getting familiar with the data, which will provide useful knowledge for data pre-processing.

2.2 Descriptive statistics

Summary statistics are measurements meant to describe data.

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt

#Load libraries for data processing
import pandas as pd #data processing, CSV file I/O (e.g. pd.read_cs
v)
import numpy as np
from scipy.stats import norm
import seaborn as sns # visualization


plt.rcParams['figure.figsize'] = (15,8)
plt.rcParams['axes.titlesize'] = 'large'
```

In [2]:
```
data = pd.read_csv('data/clean-data.csv', index_col=False)
data.drop('Unnamed: 0',axis=1, inplace=True)
#data.head(2)
```

In [3]:
```
#basic descriptive statistics
data.describe()
```

Out[3]:

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

8 rows × 30 columns

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

Out[4]:
```
radius_mean                0.942380
texture_mean               0.650450
perimeter_mean             0.990650
area_mean                  1.645732
smoothness_mean            0.456324
compactness_mean           1.190123
concavity_mean             1.401180
concave points_mean        1.171180
symmetry_mean              0.725609
fractal_dimension_mean     1.304489
radius_se                  3.088612
texture_se                 1.646444
perimeter_se               3.443615
area_se                    5.447186
smoothness_se              2.314450
compactness_se             1.902221
concavity_se               5.110463
concave points_se          1.444678
symmetry_se                2.195133
fractal_dimension_se       3.923969
radius_worst               1.103115
texture_worst              0.498321
perimeter_worst            1.128164
area_worst                 1.859373
smoothness_worst           0.415426
compactness_worst          1.473555
concavity_worst            1.150237
concave points_worst       0.492616
symmetry_worst             1.433928
fractal_dimension_worst    1.662579
dtype: float64
```

The skew result show a positive (right) or negative (left) skew. Values closer to zero show less skew. From the graphs, we can see that **radius_mean**, **perimeter_mean**, **area_mean**, **concavity_mean** and **concave_points_mean** are useful in predicting cancer type due to the distinct grouping between malignant and benign cancer types in these features. We can also see that area_worst and perimeter_worst are also quite useful.

In [5]:
```
#data.diagnosis.unique()
```

In [6]:
```
# Group by diagnosis and review the output.
#diag_gr = data.groupby('diagnosis', axis=0)
#pd.DataFrame(diag_gr.size(), columns=['# of observations'])
```

Check binary encoding from NB1 to confirm the coversion of the diagnosis categorical data into numeric, where

- Malignant = 1 (indicates prescence of cancer cells)
- Benign = 0 (indicates abscence)

Observation

> *357 observations indicating the absence of cancer cells and 212 show absence of cancer cell*

Lets confirm this, by ploting the histogram

2.3 Unimodal Data Visualizations

One of the main goals of visualizing the data here is to observe which features are most helpful in predicting malignant or benign cancer. The other is to see general trends that may aid us in model selection and hyper parameter selection.

Apply 3 techniques that you can use to understand each attribute of your dataset independently.

- Histograms.
- Density Plots.
- Box and Whisker Plots.

In [7]:
```
#lets get the frequency of cancer diagnosis
sns.set_style("white")
sns.set_context({"figure.figsize": (10, 8)})
#sns.countplot(data['diagnosis'],label='Count',palette="Set3")
```

2.3.1 Visualise distribution of data via histograms

Histograms are commonly used to visualize numerical variables. A histogram is similar to a bar graph after the values of the variable are grouped (binned) into a finite number of intervals (bins).

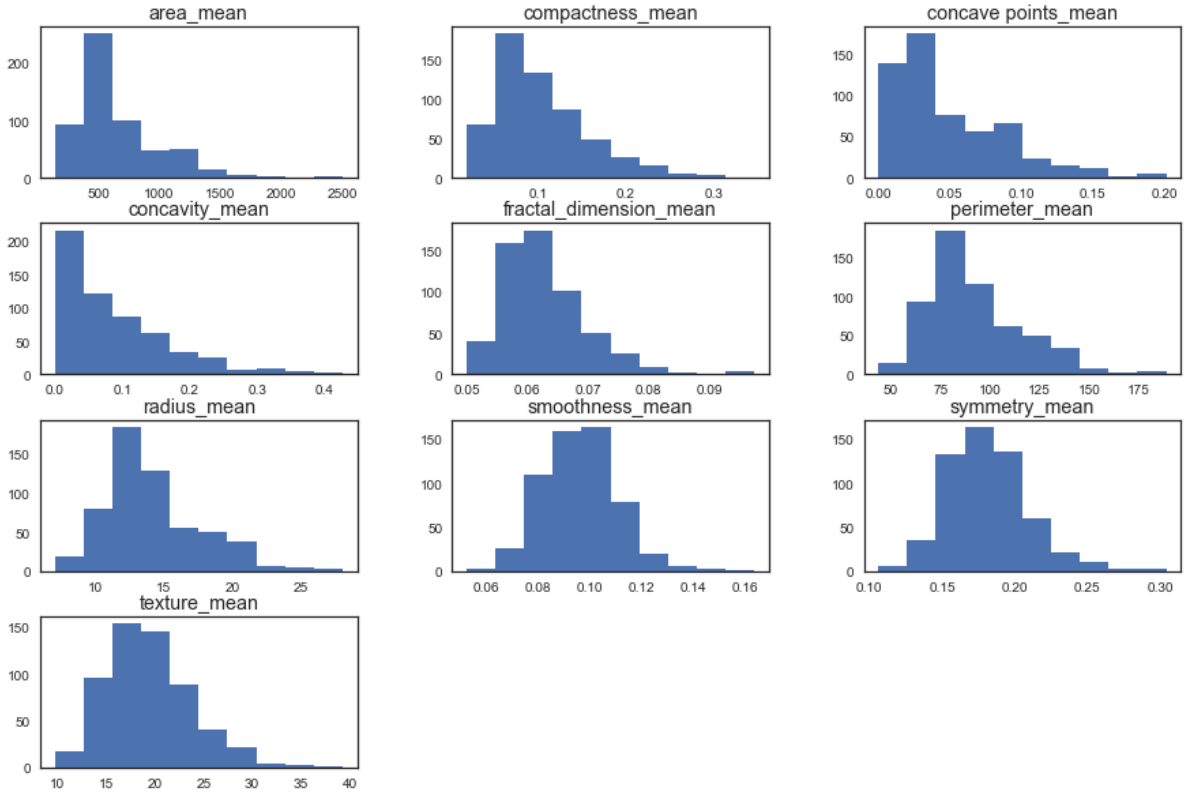
Histograms group data into bins and provide you a count of the number of observations in each bin. From the shape of the bins you can quickly get a feeling for whether an attribute is Gaussian, skewed or even has an exponential distribution. It can also help you see possible outliers.

Separate columns into smaller dataframes to perform visualization

In [8]:
```
#Break up columns into groups, according to their suffix designatio
n
#(_mean, _se,
# and __worst) to perform visualisation plots off.
#Join the 'ID' and 'Diagnosis' back on
data_id_diag=data.loc[:,["id","diagnosis"]]
data_diag=data.loc[:,["diagnosis"]]

#For a merge + slice:
data_mean=data.ix[:,1:11]
#data_se=data.ix[:,11:22]
#data_worst=data.ix[:,23:]

#print(df_id_diag.columns)
#print(data_mean.columns)
#print(data_se.columns)
#print(data_worst.columns)
```

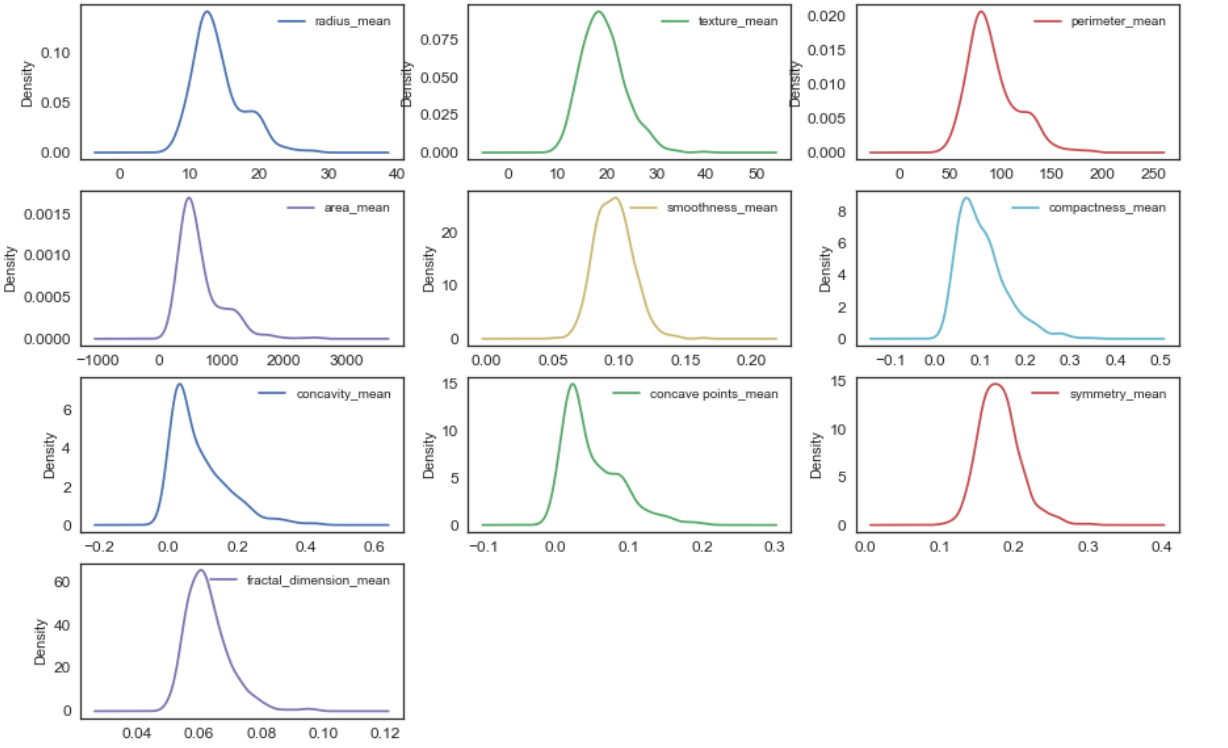
Histogram the "_mean" suffix designition

In [9]:
```
#Plot histograms of CUT1 variables
hist_mean=data_mean.hist(bins=10, figsize=(15, 10),grid=False,)

#Any individual histograms, use this:
#df_cut['radius_worst'].hist(bins=100)
```



2.3.2 Visualize distribution of data via density plots

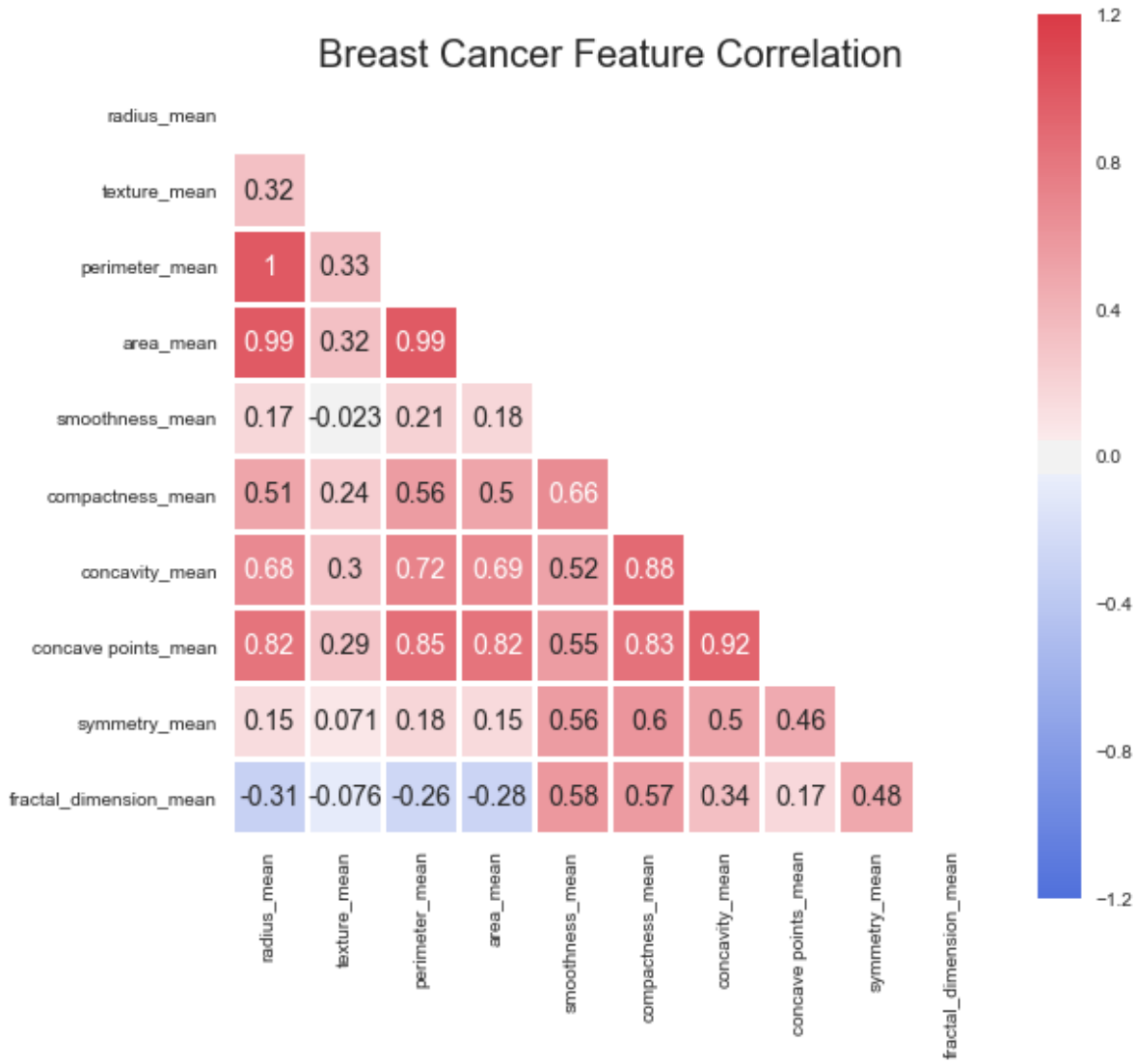
In [12]:
```
#Density Plots
plt = data_mean.plot(kind= 'density', subplots=True, layout=(4,3),
sharex=False,
                     sharey=False,fontsize=12, figsize=(15,10))
```



2.4 Multimodal Data Visualizations

- Scatter plots
- Correlation matrix

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x114932630>

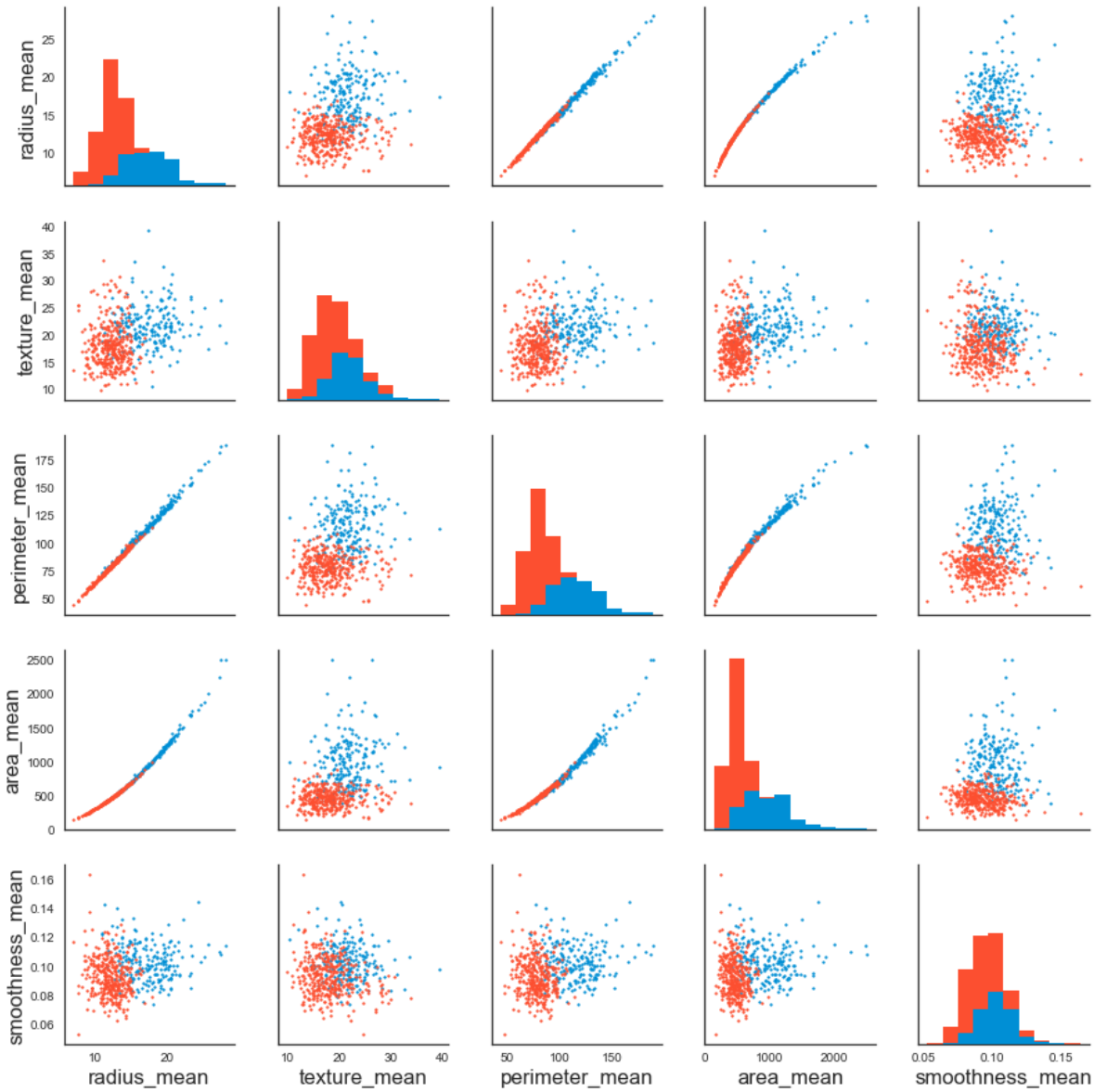


Observation:

We can see strong positive relationship exists with mean values paramaters between 1-0.75;.

- The mean area of the tissue nucleus has a strong positive correlation with mean values of radius and parameter;
- Some paramters are moderately positive corrlated (r between 0.5-0.75)are concavity and area, concavity and perimeter etc
- Likewise, we see some strong negative correlation between fractal_dimension with radius, texture, parameter mean values.

In [19]:
```
plt.style.use('fivethirtyeight')
sns.set_style("white")

data = pd.read_csv('data/clean-data.csv', index_col=False)
g = sns.PairGrid(data[[data.columns[1],data.columns[2],data.columns
[3],
                       data.columns[4], data.columns[5],data.columns[
6]]],hue='diagnosis' )
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter, s = 3)
```



Summary

- 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.
- 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.

Notebook 3: Pre-Processing the data

Goal:

Find the most predictive features of the data and filter it so it will enhance the predictive power of the analytics model.

Load data and essential libraries

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt

#Load libraries for data processing
import pandas as pd #data processing, CSV file I/O (e.g. pd.read_cs
v)
import numpy as np
from scipy.stats import norm

# visualization
import seaborn as sns
plt.style.use('fivethirtyeight')
sns.set_style("white")


plt.rcParams['figure.figsize'] = (8,4)
#plt.rcParams['axes.titlesize'] = 'large'

data = pd.read_csv('data/clean-data.csv', index_col=False)
data.drop('Unnamed: 0',axis=1, inplace=True)
#data.head()
```

Label encoding

Here, I assign the 30 features to a NumPy array X, and transform the class labels from their original string representation (M and B) into integers

In [2]:
```
#Assign predictors to a variable of ndarray (matrix) type
array = data.values
X = array[:,1:31]
y = array[:,0]
```

In [3]:
```
#transform the class labels from their original string representati
on (M and B) into integers
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

#Call the transform method of LabelEncorder on two dummy variables
#le.transform (['M', 'B'])
```

Split data into training and test sets

In [4]:
```
from sklearn.model_selection import train_test_split

##Split data set in train 70% and test 30%
X_train, X_test, y_train, y_test = train_test_split( X, y, test_siz
e=0.25, random_state=7)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[4]:
```
((426, 30), (426,), (143, 30), (143,))
```

Feature Standardization

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

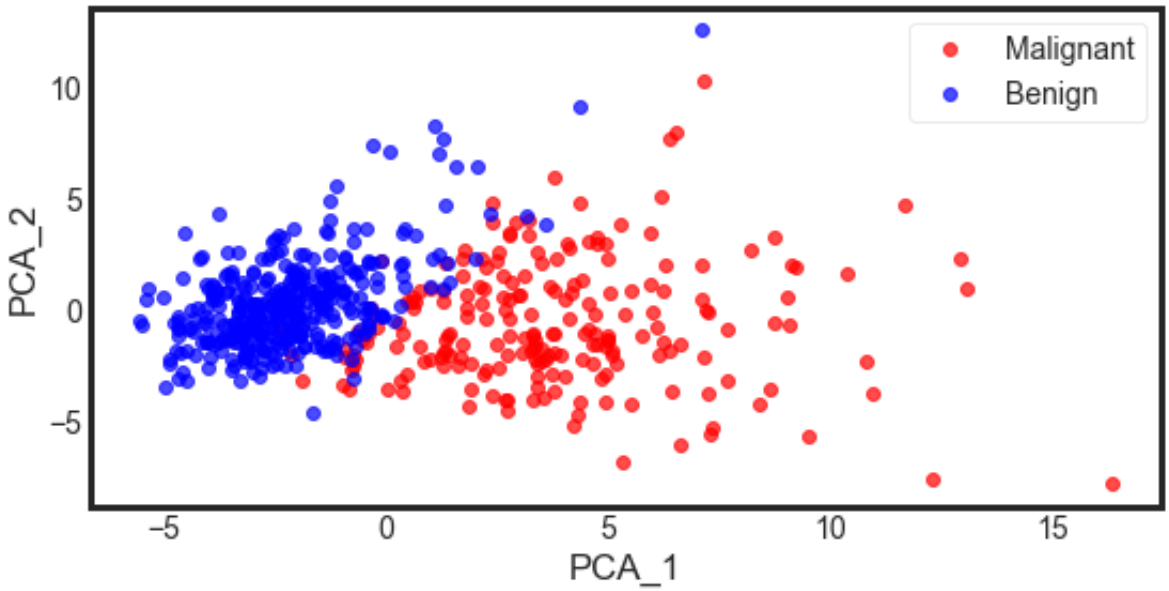
In [5]:
```
from sklearn.preprocessing import StandardScaler

# Normalize the  data (center around 0 and scale to remove the vari
ance).
scaler =StandardScaler()
Xs = scaler.fit_transform(X)
```

Feature decomposition using Principal Component Analysis(PCA)

From the pair plot earlier, lot of feature pairs divide nicely the data to a similar extent, therefore, it makes sense to use one of the dimensionality reduction methods to try to use as many features as possible and maintian as much information as possible when working with only 2 dimensions. I will use PCA

In [6]:
```
from sklearn.decomposition import PCA
# feature extraction
pca = PCA(n_components=10)
fit = pca.fit(Xs)

# summarize components
#print("Explained Variance: %s") % fit.explained_variance_ratio_
#print(fit.components_)
```

In [7]:
```
X_pca = pca.transform(Xs)

PCA_df = pd.DataFrame()

PCA_df['PCA_1'] = X_pca[:,0]
PCA_df['PCA_2'] = X_pca[:,1]

plt.plot(PCA_df['PCA_1'][data.diagnosis == 'M'],PCA_df['PCA_2'][dat
a.diagnosis == 'M'],'o', alpha = 0.7, color = 'r')
plt.plot(PCA_df['PCA_1'][data.diagnosis == 'B'],PCA_df['PCA_2'][dat
a.diagnosis == 'B'],'o', alpha = 0.7, color = 'b')

plt.xlabel('PCA_1')
plt.ylabel('PCA_2')
plt.legend(['Malignant','Benign'])
plt.show()
```



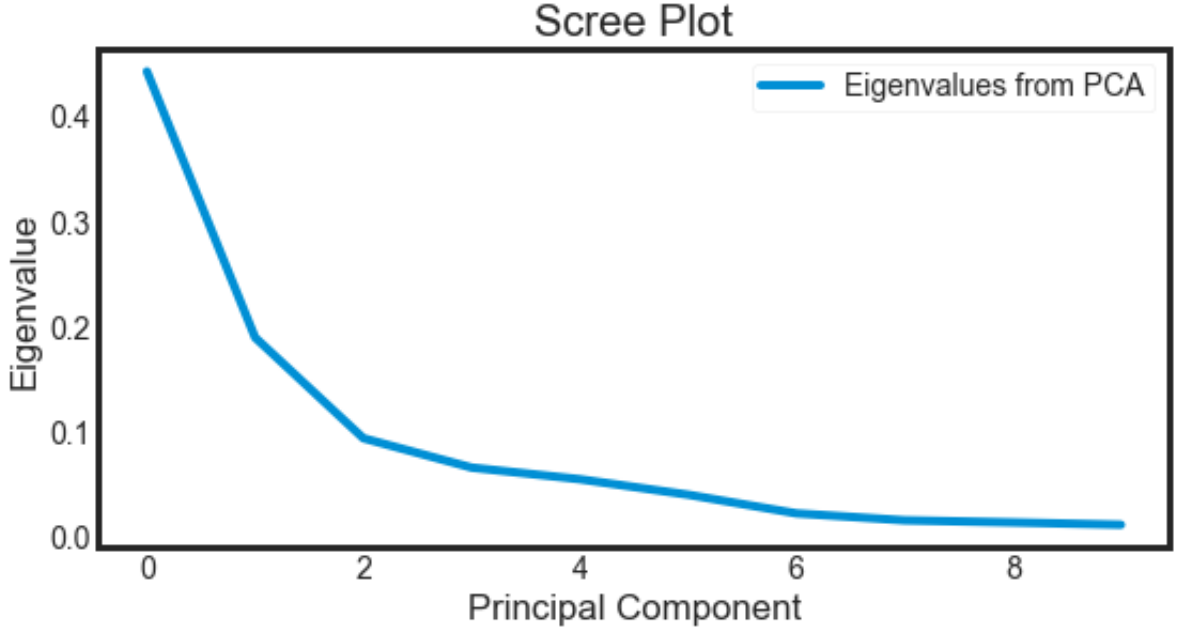
Now, what we got after applying the linear PCA transformation is a lower dimensional subspace (from 3D to 2D in this case), where the samples are “most spread” along the new feature axes.

In [8]:
```
#The amount of variance that each PC explains
var= pca.explained_variance_ratio_
#Cumulative Variance explains
#var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)
*100)
#print(var1)
```

Deciding How Many Principal Components to Retain

In order to decide how many principal components should be retained, it is common to summarise the results of a principal components analysis by making a scree plot. More about scree plot can be found here (http://python-for-multivariate-analysis.readthedocs.io/a_little_book_of_python_for_multivariate_analysis.html), and hear (https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/)

In [10]:
```
#The amount of variance that each PC explains
var= pca.explained_variance_ratio_
#Cumulative Variance explains
#var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)
*100)
#print(var1)

plt.plot(var)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')

leg = plt.legend(['Eigenvalues from PCA'], loc='best', borderpad=0.
3,shadow=False,markerscale=0.4)
leg.get_frame().set_alpha(0.4)
#leg.draggable(state=True)
plt.show()
```



A Summary of the Data Preprocing Approach used here:

1. assign features to a NumPy array X, and transform the class labels from their original string representation (M and B) into integers
2. Split data into training and test sets
3. Standardize the data.
4. Obtain the Eigenvectors and Eigenvalues from the covariance matrix or correlation matrix
5. Sort eigenvalues in descending order and choose the kk eigenvectors that correspond to the kk largest eigenvalues where k is the number of dimensions of the new feature subspace (k≤dk≤d).
6. Construct the projection matrix W from the selected k eigenvectors.
7. Transform the original dataset X via W to obtain a k-dimensional feature subspace Y.

Predictive model using Support Vector Machine (SVM)

Support vector machines (SVMs) learning algorithm will be used to build the predictive model. SVMs are one of the most popular classification algorithms, and have an elegant way of transforming nonlinear data so that one can use a linear algorithm to fit a linear model to the data (Cortes and Vapnik 1995)

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt

#Load libraries for data processing
import pandas as pd #data processing, CSV file I/O (e.g. pd.read_cs
v)
import numpy as np
from scipy.stats import norm

## Supervised learning.
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix
from sklearn import metrics, preprocessing
from sklearn.metrics import classification_report

# visualization
import seaborn as sns
plt.style.use('fivethirtyeight')
sns.set_style("white")

plt.rcParams['figure.figsize'] = (8,4)
#plt.rcParams['axes.titlesize'] = 'large'
```

In [2]:
```
data = pd.read_csv('data/clean-data.csv', index_col=False)
data.drop('Unnamed: 0',axis=1, inplace=True)
#data.head()
```

In [3]:
```
#Assign predictors to a variable of ndarray (matrix) type
array = data.values
X = array[:,1:31] # features
y = array[:,0]

#transform the class labels from their original string representati
on (M and B) into integers
le = LabelEncoder()
y = le.fit_transform(y)

# Normalize the  data (center around 0 and scale to remove the vari
ance).
scaler =StandardScaler()
Xs = scaler.fit_transform(X)
```

Classification with cross-validation

As discussed earlier, splitting the data into test and training sets is crucial to avoid overfitting. This allows generalization of real, previously-unseen data. Cross-validation extends this idea further. Instead of having a single train/test split, we specify **so-called folds** so that the data is divided into similarly-sized folds.

In [4]:
```
# 5. Divide records in training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(Xs, y, test_siz
e=0.3, random_state=2, stratify=y)

# 6. Create an SVM classifier and train it on 70% of the data set.
clf = SVC(probability=True)
clf.fit(X_train, y_train)

 #7. Analyze accuracy of predictions on 30% of the holdout test sam
ple.
classifier_score = clf.score(X_test, y_test)
print ('\nThe classifier accuracy score is {:03.2f}\n'.format(class
ifier_score))
```

```
The classifier accuracy score is 0.95
```

To get a better measure of prediction accuracy, we can successively split the data into folds that you will use for training and testing:

In [5]:
```
# Get average of 3-fold cross-validation score using an SVC estimat
or.
n_folds = 3
cv_error = np.average(cross_val_score(SVC(), Xs, y, cv=n_folds))
print ('\nThe {}-fold cross-validation accuracy score for this clas
sifier is {:.2f}\n'.format(n_folds, cv_error))
```

```
The 3-fold cross-validation accuracy score for this classifier is
0.97
```

The above evaluations were based on using the entire set of features. You will now employ the correlation-based feature selection strategy to assess the effect of using 3 features which have the best correlation with the class labels.

In [6]:
```
from sklearn.feature_selection import SelectKBest, f_regression
clf2 = make_pipeline(SelectKBest(f_regression, k=3),SVC(probability
=True))

scores = cross_val_score(clf2, Xs, y, cv=3)

# Get average of 3-fold cross-validation score using an SVC estimat
or.
n_folds = 3
cv_error = np.average(cross_val_score(SVC(), Xs, y, cv=n_folds))
print ('\nThe {}-fold cross-validation accuracy score for this clas
sifier is {:.2f}\n'.format(n_folds, cv_error))
```

```
The 3-fold cross-validation accuracy score for this classifier is
0.97
```

In [7]:
```
print(scores)
avg = (100*np.mean(scores), 100*np.std(scores)/np.sqrt(scores.shape
[0]))
print ("Average score and uncertainty: (%.2f +- %.3f)%%"%avg)
```

```
[0.93157895 0.95263158 0.94179894]
Average score and uncertainty: (94.20 +- 0.496)%
```

From the above results, you can see that only a fraction of the features are required to build a model that performs similarly to models based on using the entire set of features. Feature selection is an important part of the model-building process that you must always pay particular attention to. The details are beyond the scope of this notebook. In the rest of the analysis, you will continue using the entire set of features.

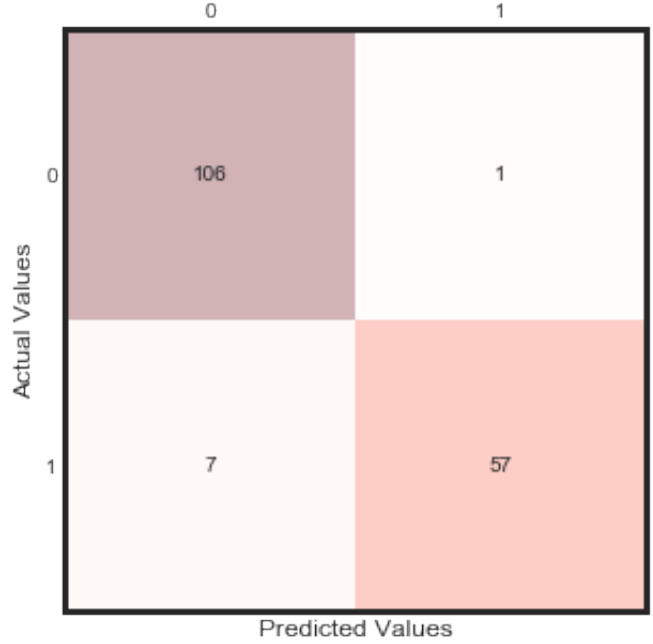
Model Accuracy: Receiver Operating Characteristic (ROC) curve

In statistical modeling and machine learning, a commonly-reported performance measure of model accuracy for binary classification problems is Area Under the Curve (AUC).

> (More information in the project report).

In [8]:
```
# The confusion matrix helps visualize the performance of the algor
ithm.
y_pred = clf.fit(X_train, y_train).predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
#print(cm)
```

In [9]:
```
%matplotlib inline
import matplotlib.pyplot as plt

from IPython.display import Image, display

fig, ax = plt.subplots(figsize=(5, 5))
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	0.99	0.96	107
1	0.98	0.89	0.93	64
accuracy			0.95	171
macro avg	0.96	0.94	0.95	171
weighted avg	0.95	0.95	0.95	171

Observation

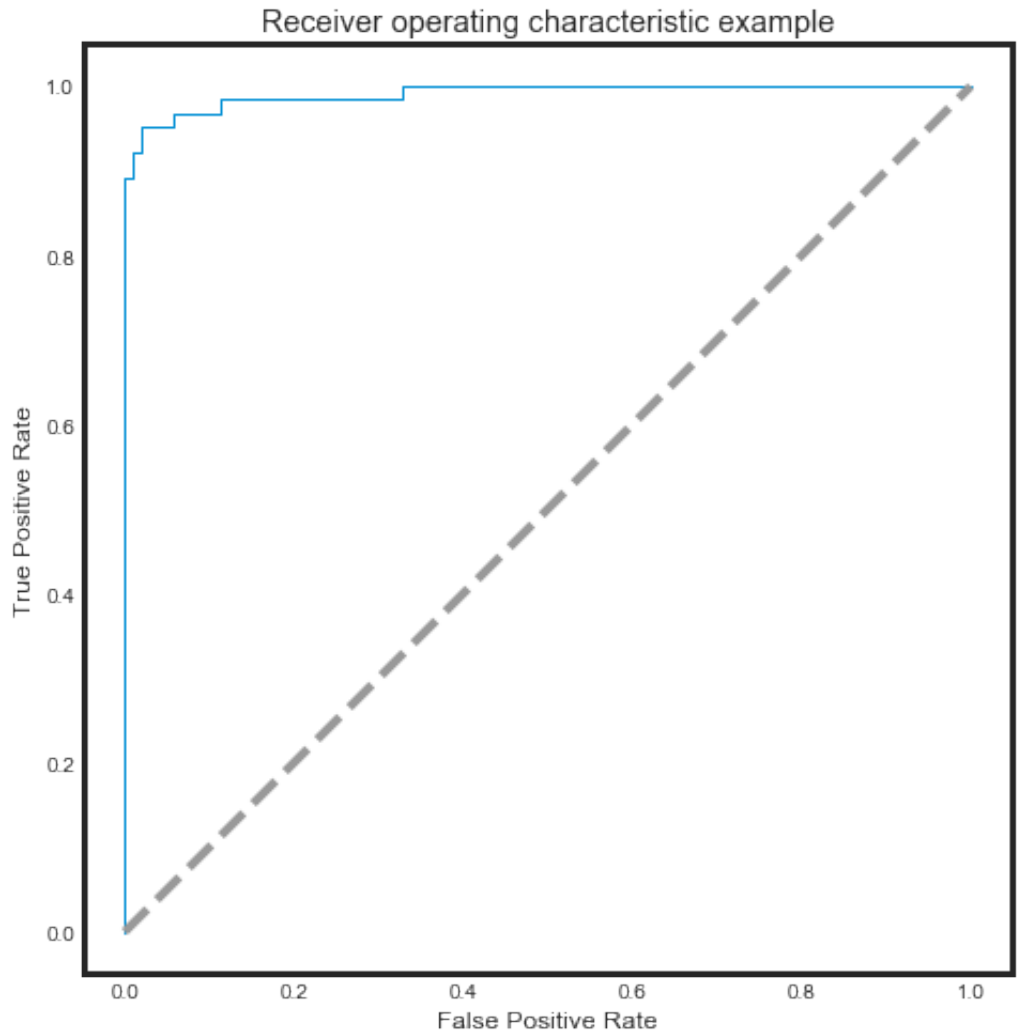
There are two possible predicted classes: "1" and "0". Malignant = 1 (indicates prescence of cancer cells) and Benign = 0 (indicates abscence).

- The classifier made a total of 174 predictions (i.e 174 patients were being tested for the presence breast cancer).
- Out of those 174 cases, the classifier predicted "yes" 58 times, and "no" 113 times.
- In reality, 64 patients in the sample have the disease, and 107 patients do not.

Rates as computed from the confusion matrix

1. **Accuracy**: Overall, how often is the classifier correct?
   - (TP+TN)/total = (57+106)/171 = 0.95
2. **Misclassification Rate**: Overall, how often is it wrong?
   - (FP+FN)/total = (1+7)/171 = 0.05 equivalent to 1 minus Accuracy also known as ***"Error Rate"***
3. **True Positive Rate:** When it's actually yes, how often does it predict 1?
   - TP/actual yes = 57/64 = 0.89 also known as "Sensitivity" or ***"Recall"***
4. **False Positive Rate**: When it's actually 0, how often does it predict 1?
   - FP/actual no = 1/107 = 0.01
5. **Specificity**: When it's actually 0, how often does it predict 0? also know as **true positive rate**
   - TN/actual no = 106/107 = 0.99 equivalent to 1 minus False Positive Rate
6. **Precision**: When it predicts 1, how often is it correct?
   - TP/predicted yes = 57/58 = 0.98
7. **Prevalence**: How often does the yes condition actually occur in our sample?
   - actual yes/total = 64/171 = 0.34

In [10]:
```
from sklearn.metrics import roc_curve, auc
# Plot the receiver operating characteristic curve (ROC).
plt.figure(figsize=(10,8))
probas_ = clf.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, lw=1, label='ROC fold (area = %0.2f)' % (roc_auc
))
plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Random
')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.axes().set_aspect(1)
```



- To interpret the ROC correctly, consider what the points that lie along the diagonal represent. For these situations, there is an equal chance of "+" and "-" happening. Therefore, this is not that different from making a prediction by tossing of an unbiased coin. Put simply, the classification model is random.
- For the points above the diagonal, tpr > fpr, and the model says that you are in a zone where you are performing better than random. For example, assume tpr = 0.99 and fpr = 0.01, Then, the probability of being in the true positive group is $(0.99/(0.99 + 0.01)) = 99\%$. Furthermore, holding fpr constant, it is easy to see that the more vertically above the diagonal you are positioned, the better the classification model.

Optimizing the SVM Classifier

Machine learning models are parameterized so that their behavior can be tuned for a given problem. Models can have many parameters and finding the best combination of parameters can be treated as a search problem. In this notebook, I aim to tune parameters of the SVM Classification model using scikit-learn.

Load Libraries and Data

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt

#Load libraries for data processing
import pandas as pd #data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np
from scipy.stats import norm

## Supervised learning.
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.metrics import confusion_matrix
from sklearn import metrics, preprocessing
from sklearn.metrics import classification_report
from sklearn.feature_selection import SelectKBest, f_regression

# visualization
import seaborn as sns
plt.style.use('fivethirtyeight')
sns.set_style("white")

plt.rcParams['figure.figsize'] = (8,4)
#plt.rcParams['axes.titlesize'] = 'large'
```

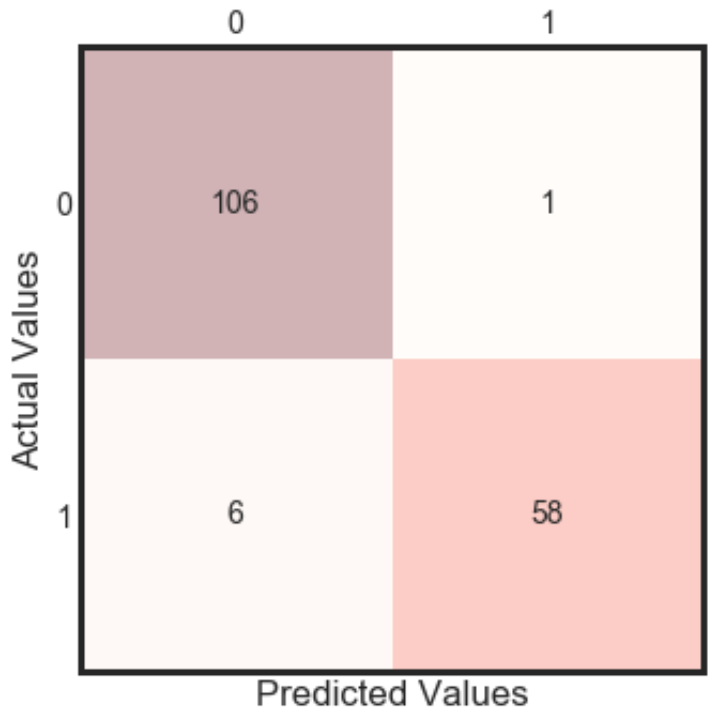
Build a predictive model and evaluate with 5-cross validation using support vector classifies (ref NB4) for details

In [4]:
```
data = pd.read_csv('data/clean-data.csv', index_col=False)
data.drop('Unnamed: 0',axis=1, inplace=True)


#Assign predictors to a variable of ndarray (matrix) type
array = data.values
X = array[:,1:31]
```

```
y = array[:,0]

#transform the class labels from their original string representati
on (M and B) into integers
le = LabelEncoder()
y = le.fit_transform(y)

# Normalize the  data (center around 0 and scale to remove the vari
ance).
scaler =StandardScaler()
Xs = scaler.fit_transform(X)

from sklearn.decomposition import PCA
# feature extraction
pca = PCA(n_components=10)
fit = pca.fit(Xs)
X_pca = pca.transform(Xs)

# 5. Divide records in training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_
size=0.3, random_state=2, stratify=y)

# 6. Create an SVM classifier and train it on 70% of the data set.
clf = SVC(probability=True)
clf.fit(X_train, y_train)

 #7. Analyze accuracy of predictions on 30% of the holdout test sam
ple.
classifier_score = clf.score(X_test, y_test)
print ('\nThe classifier accuracy score is {:03.2f}\n'.format(class
ifier_score))

clf2 = make_pipeline(SelectKBest(f_regression, k=3),SVC(probability
=True))
scores = cross_val_score(clf2, X_pca, y, cv=3)

# Get average of 5-fold cross-validation score using an SVC estimat
or.
n_folds = 5
cv_error = np.average(cross_val_score(SVC(), X_pca, y, cv=n_folds))
print ('\nThe {}-fold cross-validation accuracy score for this clas
sifier is {:.2f}\n'.format(n_folds, cv_error)
)
y_pred = clf.fit(X_train, y_train).predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)

print(classification_report(y_test, y_pred ))

fig, ax = plt.subplots(figsize=(5, 5))
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()
```

```
The classifier accuracy score is 0.96


The 5-fold cross-validation accuracy score for this classifier is
0.98

              precision    recall  f1-score   support

           0       0.95      0.99      0.97       107
           1       0.98      0.91      0.94        64

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```



Optimizing classifiers

Python scikit-learn provides two simple methods for algorithm parameter tuning:

- Grid Search Parameter Tuning.
- Random Search Parameter Tuning.

In [5]:
```
# Train classifiers.
kernel_values = [ 'linear' , 'poly' , 'rbf' , 'sigmoid' ]
param_grid = {'C': np.logspace(-3, 2, 6), 'gamma': np.logspace(-3,
2, 6),'kernel': kernel_values}

grid = GridSearchCV(SVC(), param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

Out[5]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200
,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3
,
                           gamma='scale', kernel='rbf', max_iter=-
1,
                           probability=False, random_state=None, s
hrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+0
0, 1.e+01, 1.e+02]),
                         'gamma': array([1.e-03, 1.e-02, 1.e-01, 1
.e+00, 1.e+01, 1.e+02]),
                         'kernel': ['linear', 'poly', 'rbf', 'sigm
oid']},
             pre_dispatch='2*n_jobs', refit=True, return_train_sco
re=False,
             scoring=None, verbose=0)
```

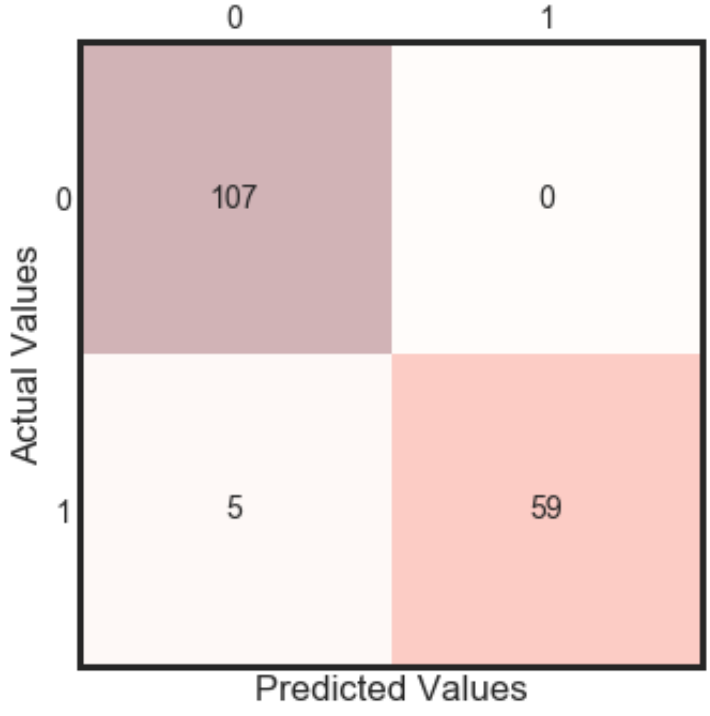
In [6]:
```
print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'C': 0.1, 'gamma': 0.001, 'kernel': 'line
ar'} with a score of 0.98
```

In [7]:
```
grid.best_estimator_.probability = True
clf = grid.best_estimator_
```

In [8]:
```
y_pred = clf.fit(X_train, y_train).predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
#print(cm)
print(classification_report(y_test, y_pred ))

fig, ax = plt.subplots(figsize=(5, 5))
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()
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       107
           1       1.00      0.92      0.96        64

    accuracy                           0.97       171
   macro avg       0.98      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```



Decision boundaries of different classifiers

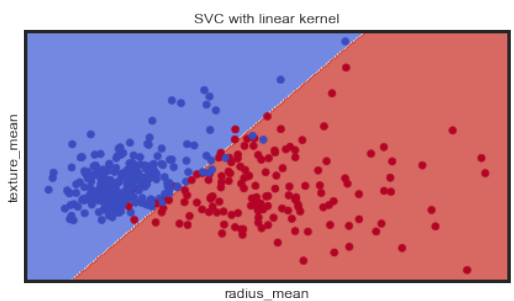
Let's see the decision boundaries produced by the linear, Gaussian and polynomial classifiers.

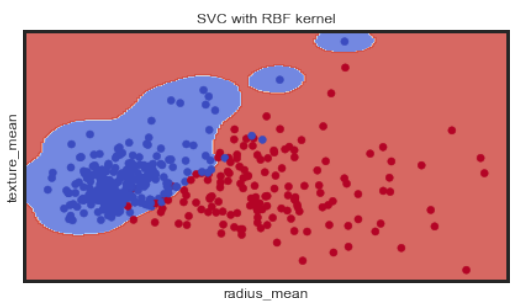
In [9]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import svm, datasets

def decision_plot(X_train, y_train, n_neighbors, weights):
      h = .02  # step size in the mesh

Xtrain = X_train[:, :2] # we only take the first two features.

#====================================================================
# Create color maps
#====================================================================
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])


#====================================================================
# we create an instance of SVM and fit out data.
# We do not scale ourdata since we want to plot the support vectors
#====================================================================

C = 1.0  # SVM regularization parameter

svm = SVC(kernel='linear', random_state=0, gamma=0.1, C=C).fit(Xtra
in, y_train)
rbf_svc = SVC(kernel='rbf', gamma=0.7, C=C).fit(Xtrain, y_train)
poly_svc = SVC(kernel='poly', degree=3, C=C).fit(Xtrain, y_train)
```

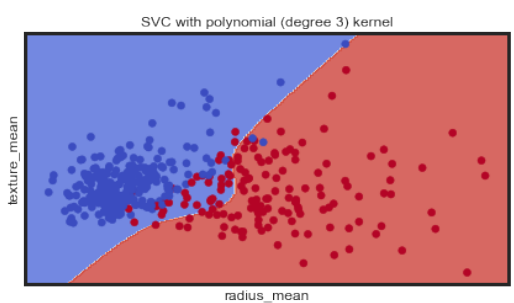
In [10]:
```
%matplotlib inline
plt.rcParams['figure.figsize'] = (15, 9)
plt.rcParams['axes.titlesize'] = 'large'

    # create a mesh to plot in
x_min, x_max = Xtrain[:, 0].min() - 1, Xtrain[:, 0].max() + 1
y_min, y_max = Xtrain[:, 1].min() - 1, Xtrain[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# title for the plots
titles = ['SVC with linear kernel',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel']
```

In [11]:

```
for i, clf in enumerate((svm, rbf_svc, poly_svc)):
    # Plot the decision boundary. For that, we will assign a color
to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(2, 2, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    plt.scatter(Xtrain[:, 0], Xtrain[:, 1], c=y_train, cmap=plt.cm.
coolwarm)
    plt.xlabel('radius_mean')
    plt.ylabel('texture_mean')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])

plt.show()
```







Next Task:

1. Summary and conclusion of findings
2. Compare with other classification methods
    - Decision trees with tree.DecisionTreeClassifier();
    - K-nearest neighbors with neighbors.KNeighborsClassifier();
    - Random forests with ensemble.RandomForestClassifier();

Automate the ML process using pipelines

There are standard workflows in a machine learning project that can be automated. In Python scikit-learn, Pipelines help to clearly define and automate these workflows.

- Pipelines help overcome common problems like data leakage in your test harness.
- Python scikit-learn provides a Pipeline utility to help automate machine learning workflows.
- Pipelines work by allowing for a linear sequence of data transforms to be chained together culminating in a modeling process that can be evaluated.

Data Preparation and Modeling Pipeline

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt

# Create a pipeline that standardizes the data then creates a model
#Load libraries for data processing
import pandas as pd #data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np
from scipy.stats import norm

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

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.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# visualization
import seaborn as sns
plt.style.use('fivethirtyeight')
sns.set_style("white")

plt.rcParams['figure.figsize'] = (8,4)
#plt.rcParams['axes.titlesize'] = 'large'
```

Evaluate Some Algorithms

Now it is time to create some models of the data and estimate their accuracy on unseen data. Here is what we are going to cover in this step:

1. Separate out a validation dataset.
2. Setup the test harness to use 10-fold cross validation.
3. Build 5 different models
4. Select the best model

1.0 Validation Dataset

In [2]:
```
#load data
data = pd.read_csv('data/clean-data.csv', index_col=False)
data.drop('Unnamed: 0',axis=1, inplace=True)

# Split-out validation dataset
array = data.values
X = array[:,1:31]
y = array[:,0]

# Divide records in training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.3, random_state=7)

#transform the class labels from their original string representati
on (M and B) into integers
le = LabelEncoder()
y = le.fit_transform(y)
```

2.0 Evaluate Algorithms: Baseline

In [3]:
```
# Spot-Check Algorithms
models = []
models.append(( 'LR' , LogisticRegression()))
models.append(( 'LDA' , LinearDiscriminantAnalysis()))
models.append(( 'KNN' , KNeighborsClassifier()))
models.append(( 'CART' , DecisionTreeClassifier()))
models.append(( 'NB' , GaussianNB()))
models.append(( 'SVM' , SVC()))

# Test options and evaluation metric
num_folds = 10
num_instances = len(X_train)
seed = 7
scoring =  'accuracy'

# Test options and evaluation metric
num_folds = 10
num_instances = len(X_train)
seed = 7
scoring =  'accuracy'
results = []
names = []
for name, model in models:
 kfold = KFold()
 cv_results = cross_val_score(model, X_train, y_train, cv=kfold, sc
oring=scoring)
 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 six classifiers')
```

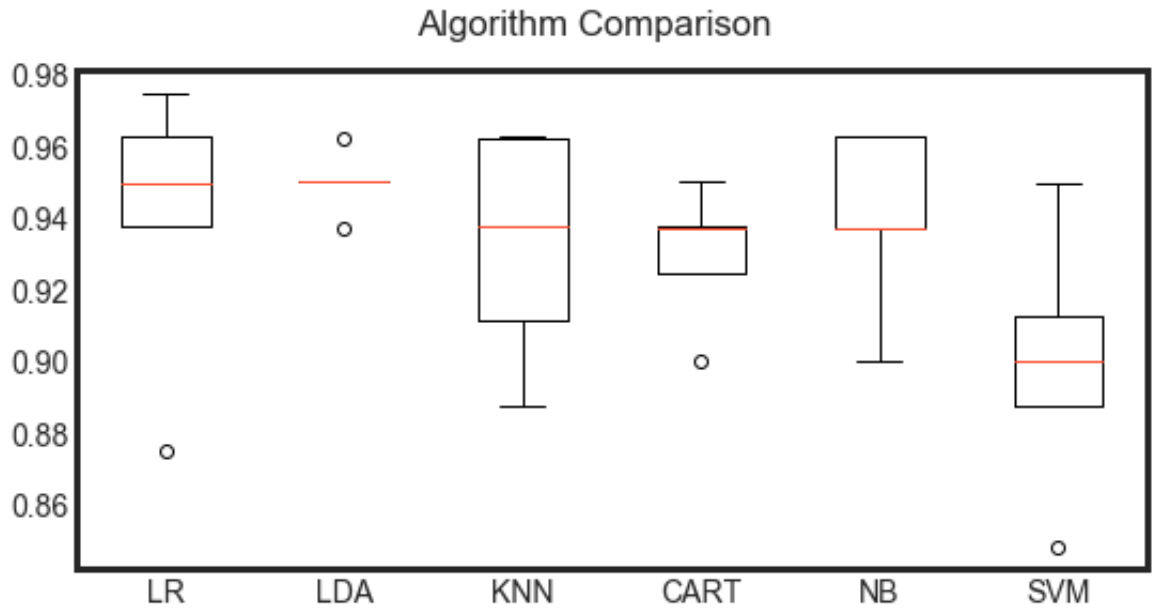
```
LR: 0.939810 (0.034722)
LDA: 0.949747 (0.008012)
KNN: 0.932184 (0.029212)
CART: 0.929652 (0.016947)
NB: 0.939684 (0.022951)
SVM: 0.899494 (0.032990)
-> 10-Fold cross-validation accurcay score for the training data f
or six classifiers

/Users/aseemsangalay/anaconda3/lib/python3.7/site-packages/sklearn
/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opti
ons:
    https://scikit-learn.org/stable/modules/linear_model.html#logi
stic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/aseemsangalay/anaconda3/lib/python3.7/site-packages/sklearn
/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed t
```

```
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opti
ons:
    https://scikit-learn.org/stable/modules/linear_model.html#logi
stic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/aseemsangalay/anaconda3/lib/python3.7/site-packages/sklearn
/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opti
ons:
    https://scikit-learn.org/stable/modules/linear_model.html#logi
stic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/aseemsangalay/anaconda3/lib/python3.7/site-packages/sklearn
/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opti
ons:
    https://scikit-learn.org/stable/modules/linear_model.html#logi
stic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/aseemsangalay/anaconda3/lib/python3.7/site-packages/sklearn
/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed t
o converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opti
ons:
    https://scikit-learn.org/stable/modules/linear_model.html#logi
stic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Observation

> The results suggest That both Logistic Regression and LDA may be worth further study. These are just mean accuracy values. It is always wise to look at the distribution of accuracy values calculated across cross validation folds. We can do that graphically using box and whisker plots.

In [4]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle( 'Algorithm Comparison' )
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Observation

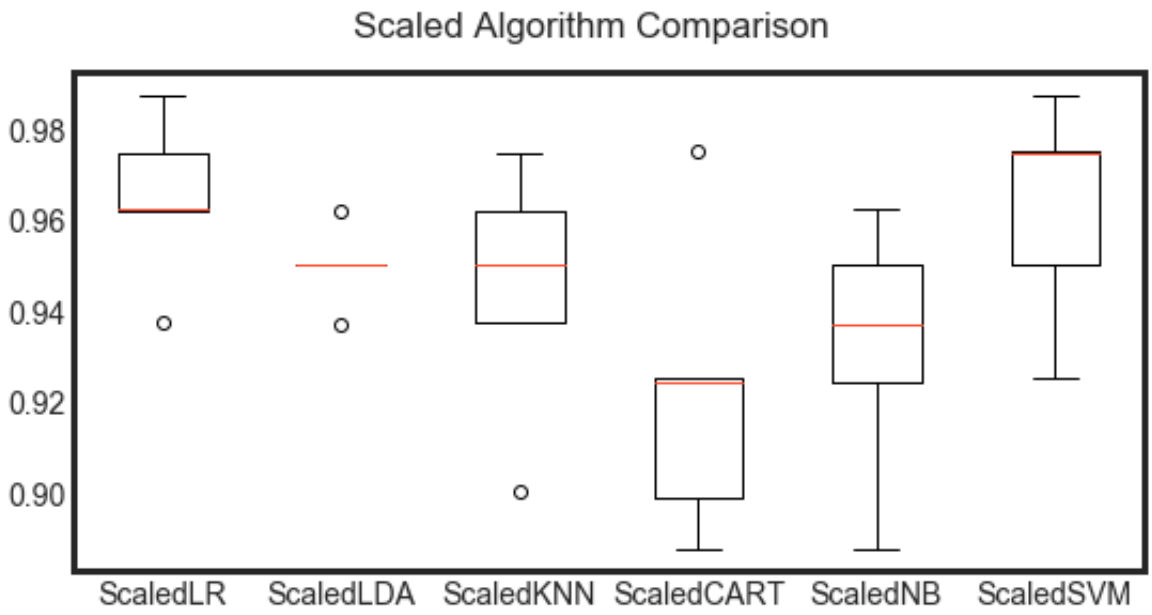
> The results show a similar tight distribution for all classifiers except SVM which is encouraging, suggesting low variance. The poor results for SVM are surprising.
>
> It is possible the varied distribution of the attributes may have an effect on the accuracy of algorithms such as SVM. In the next section we will repeat this spot-check with a standardized copy of the training dataset.

2.1 Evaluate Algorithms: Standardize Data

In [5]:
```
# Standardize the dataset
pipelines = []
pipelines.append(( 'ScaledLR' , Pipeline([( 'Scaler' , StandardScal
er()),( 'LR' ,
    LogisticRegression())])))
pipelines.append(( 'ScaledLDA' , Pipeline([( 'Scaler' , StandardSca
ler()),( 'LDA' ,
    LinearDiscriminantAnalysis())])))
pipelines.append(( 'ScaledKNN' , Pipeline([( 'Scaler' , StandardSca
ler()),( 'KNN' ,
    KNeighborsClassifier())])))
pipelines.append(( 'ScaledCART' , Pipeline([( 'Scaler' , StandardSc
aler()),( 'CART' ,
    DecisionTreeClassifier())])))
pipelines.append(( 'ScaledNB' , Pipeline([( 'Scaler' , StandardScal
er()),( 'NB' ,
    GaussianNB())])))
pipelines.append(( 'ScaledSVM' , Pipeline([( 'Scaler' , StandardSca
ler()),( 'SVM' , SVC())])))

results = []
names = []
for name, model in pipelines:
  kfold = KFold()
  cv_results = cross_val_score(model, X_train, y_train, cv=kfold,
      scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

```
ScaledLR: 0.964842 (0.016560)
ScaledLDA: 0.949747 (0.008012)
ScaledKNN: 0.944842 (0.025601)
ScaledCART: 0.922057 (0.030179)
ScaledNB: 0.932152 (0.025767)
ScaledSVM: 0.962405 (0.022290)
```

In [6]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle( 'Scaled Algorithm Comparison' )
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Observations

> The results show that standardization of the data has lifted the skill of SVM to be the most accurate algorithm tested so far.

The results suggest digging deeper into the SVM and LDA and LR algorithms. It is very likely that configuration beyond the default may yield even more accurate models.

3.0 Algorithm Tuning

In this section we investigate tuning the parameters for three algorithms that show promise from the spot-checking in the previous section: LR, LDA and SVM.

Tuning hyper-parameters - SVC estimator

In [7]:
```
#Make Support Vector Classifier Pipeline
pipe_svc = Pipeline([('scl', StandardScaler()),
                     ('pca', PCA(n_components=2)),
                     ('clf', SVC(probability=True, verbose=False))]
)

#Fit Pipeline to training Data
pipe_svc.fit(X_train, y_train)

#print('--> Fitted Pipeline to training Data')

scores = cross_val_score(estimator=pipe_svc, X=X_train, y=y_train,
cv=10, n_jobs=1, verbose=0)
print('--> Model Training Accuracy: %.3f +/- %.3f' %(np.mean(scores
), np.std(scores)))

#Tune Hyperparameters
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [{'clf__C': param_range,'clf__kernel': ['linear']},
              {'clf__C': param_range,'clf__gamma': param_range,
               'clf__kernel': ['rbf']}]
gs_svc = GridSearchCV(estimator=pipe_svc,
                      param_grid=param_grid,
                      scoring='accuracy',
                      cv=10,
                      n_jobs=1)
gs_svc = gs_svc.fit(X_train, y_train)
print('--> Tuned Parameters Best Score: ',gs_svc.best_score_)
print('--> Best Parameters: \n',gs_svc.best_params_)
```

```
--> Model Training Accuracy: 0.940 +/- 0.034
--> Tuned Parameters Best Score:  0.9446794871794871
--> Best Parameters:
 {'clf__C': 1.0, 'clf__kernel': 'linear'}
```

Tuning the hyper-parameters - k-NN hyperparameters

For your standard k-NN implementation, there are two primary hyperparameters that you'll want to tune:

- The number of neighbors k.
- The distance metric/similarity function.

Both of these values can dramatically affect the accuracy of your k-NN classifier. Grid object is ready to do 10-fold cross validation on a KNN model using classification accuracy as the evaluation metric In addition, there is a parameter grid to repeat the 10-fold cross validation process 30 times Each time, the n_neighbors parameter should be given a different value from the list We can't give GridSearchCV just a list We've to specify n_neighbors should take on 1 through 30 You can set n_jobs = -1 to run computations in parallel (if supported by your computer and OS)

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier as KNN

pipe_knn = Pipeline([('scl', StandardScaler()),
                     ('pca', PCA(n_components=2)),
                     ('clf', KNeighborsClassifier())])

#Fit Pipeline to training Data
pipe_knn.fit(X_train, y_train)

scores = cross_val_score(estimator=pipe_knn,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=1)
print('--> Model Training Accuracy: %.3f +/- %.3f' %(np.mean(scores
), np.std(scores)))

#Tune Hyperparameters
param_range = range(1, 31)
param_grid = [{'clf__n_neighbors': param_range}]
# instantiate the grid
grid = GridSearchCV(estimator=pipe_knn,
                    param_grid=param_grid,
                    cv=10,
                    scoring='accuracy')
gs_knn = grid.fit(X_train, y_train)
print('--> Tuned Parameters Best Score: ',gs_knn.best_score_)
print('--> Best Parameters: \n',gs_knn.best_params_)
```

```
--> Model Training Accuracy: 0.927 +/- 0.044
--> Tuned Parameters Best Score:  0.9396153846153847
--> Best Parameters:
 {'clf__n_neighbors': 19}
```

Finalize Model

In [9]:
```
#Use best parameters
clf_svc = gs_svc.best_estimator_

#Get Final Scores
clf_svc.fit(X_train, y_train)
scores = cross_val_score(estimator=clf_svc,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=1)
print('--> Final Model Training Accuracy: %.3f +/- %.3f' %(np.mean(
scores), np.std(scores)))

print('--> Final Accuracy on Test set: %.5f' % clf_svc.score(X_test
,y_test))
```

```
--> Final Model Training Accuracy: 0.945 +/- 0.041
--> Final Accuracy on Test set: 0.97076
```

In [10]:
```
clf_svc.fit(X_train, y_train)
y_pred = clf_svc.predict(X_test)

print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9707602339181286
[[114   2]
 [  3  52]]
              precision    recall  f1-score   support

           B       0.97      0.98      0.98       116
           M       0.96      0.95      0.95        55

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```

Summary

Worked through a classification predictive modeling machine learning problem from end-to-end using Python. Specifically, the steps covered were:

1. Problem Definition (Breast Cancer data).
2. Loading the Dataset.
3. Analyze Data (same scale but di↵erent distributions of data).
   - Evaluate Algorithms (KNN looked good).
   - Evaluate Algorithms with Standardization (KNN and SVM looked good).
4. Algorithm Tuning (K=19 for KNN was good, SVM with an RBF kernel and C=100 was best)..
5. Finalize Model (use all training data and confirm using validation dataset)