In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv('data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	id	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epithelial_size
0	1000025	5	1	1	1	2
1	1002945	5	4	4	5	7
2	1015425	3	1	1	1	2
3	1016277	6	8	8	1	3
4	1017023	4	1	1	3	2

In [4]:
```
# drop unknown values and id classification
df = df[df['bare_nucleoli'] != '?']
df = df.drop(columns = 'id')
df = df.apply(pd.to_numeric)
```

In [5]:
```
df.describe()
```

Out[5]:

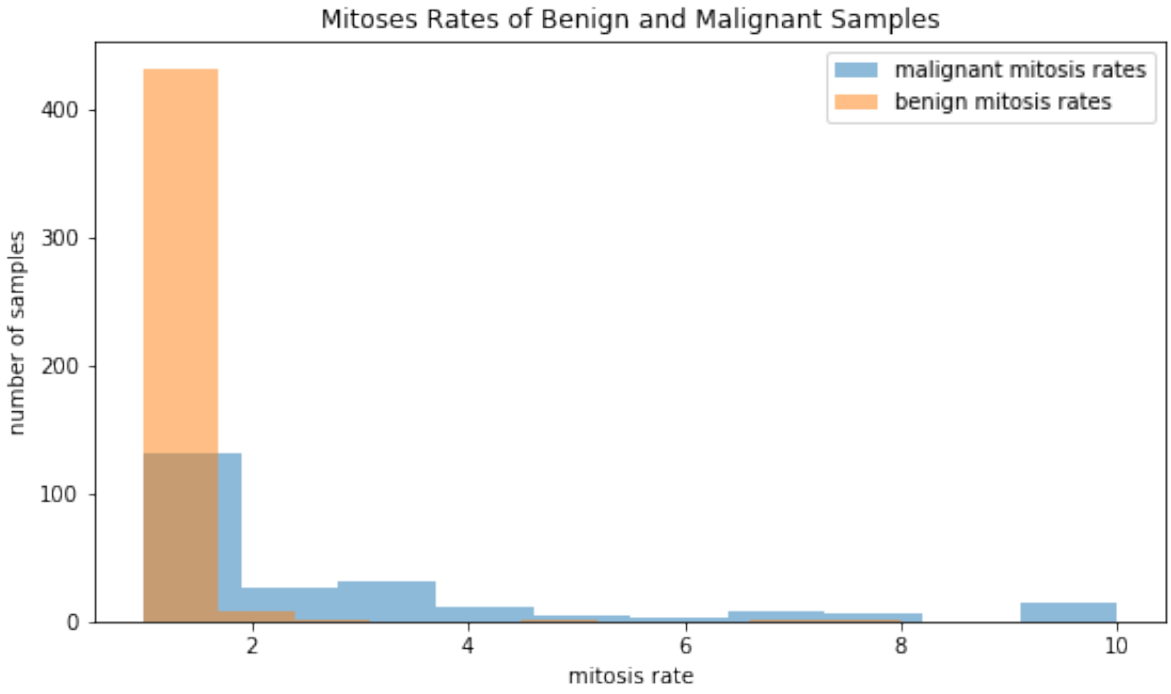
	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epithelial_size	bar
count	683.000000	683.000000	683.000000	683.000000	683.000000	6
mean	4.442167	3.150805	3.215227	2.830161	3.234261	
std	2.820761	3.065145	2.988581	2.864562	2.223085	
min	1.000000	1.000000	1.000000	1.000000	1.000000	
25%	2.000000	1.000000	1.000000	1.000000	2.000000	
50%	4.000000	1.000000	1.000000	1.000000	2.000000	
75%	6.000000	5.000000	5.000000	4.000000	4.000000	
max	10.000000	10.000000	10.000000	10.000000	10.000000	

In [6]:
```
# prelimary graph: histogram seperated by 'class'.
# bar plot of mean values, plus standard errors
# overlayed, maybe one combining both classes (all samples)

# something regression of mitoses rate
```

In [7]:
```
# seperate the mitosis rates by class
benign_mitosis = df.loc[df['class'] == 2, ['mitoses']]
malignant_mitosis = df.loc[df['class'] == 4, ['mitoses']]
```

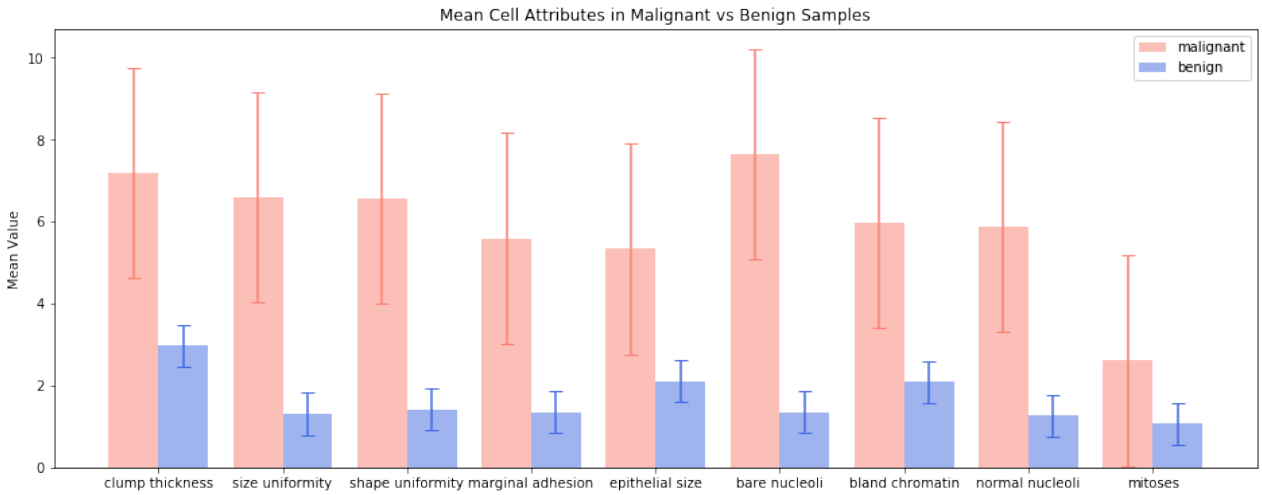
In [8]:
```
# graphs the mitosis rates histogram of benign and malignant celss
fig = plt.figure(figsize=(9, 5))
ax = fig.add_subplot(1,1,1)
for i in range(2):
    if i==1:
        ax.hist(benign_mitosis['mitoses'], alpha = 0.5, label='benign mi
    else:
        ax.hist(malignant_mitosis['mitoses'], alpha = 0.5, label='malign
plt.legend()
plt.title('Mitoses Rates of Benign and Malignant Samples')
plt.xlabel('mitosis rate')
plt.ylabel('number of samples')
```

Out[8]:
```
Text(0, 0.5, 'number of samples')
```



In [9]:
```
# https://stackoverflow.com/questions/33742588/pandas-split-dataframe-by
# seperates dataframe into malignant and benign
malignant, benign=[x for _, x in df.groupby(df['class'] < 3)]
```

In [10]:
```
cell_means = np.zeros(9)
cell_stds = np.zeros(9)
codes = np.arange(0, 9)

fig = plt.figure(figsize = (16,6))
ax1 = fig.add_subplot(1,1,1)

# for loop to compute mean values for all columns in malignant samples
for i in range(9):
    cell_means[i] = malignant.iloc[:, i].mean()
    cell_stds = malignant.iloc[:, i].std()
ax1.bar(codes-0.2, cell_means, align='center', width=0.4, alpha=0.5, lab
ax1.errorbar(codes-0.2, cell_means, yerr=cell_stds,linestyle='none', cap

# for loop to compute mean values for benign samples
cell_means = np.zeros(9)
cell_stds = np.zeros(9)
for i in range(9):
    cell_means[i] = benign.iloc[:, i].mean()
    cell_stds = benign.iloc[:, i].std()
ax1.bar(codes+0.2, cell_means, align='center', width=0.4, alpha=0.5, lab
ax1.errorbar(codes+0.2, cell_means, yerr=cell_stds,linestyle='none', cap

classifier_names = ('clump thickness', 'size uniformity', 'shape uniform
                    'bare nucleoli', 'bland chromatin', 'normal nucleoli'
plt.xticks(codes, classifier_names)
plt.legend()
plt.title('Mean Cell Attributes in Malignant vs Benign Samples')
plt.ylabel('Mean Value')

plt.savefig("Mean Cell Attributes.png")
```



In [11]:
```
df.head()
```

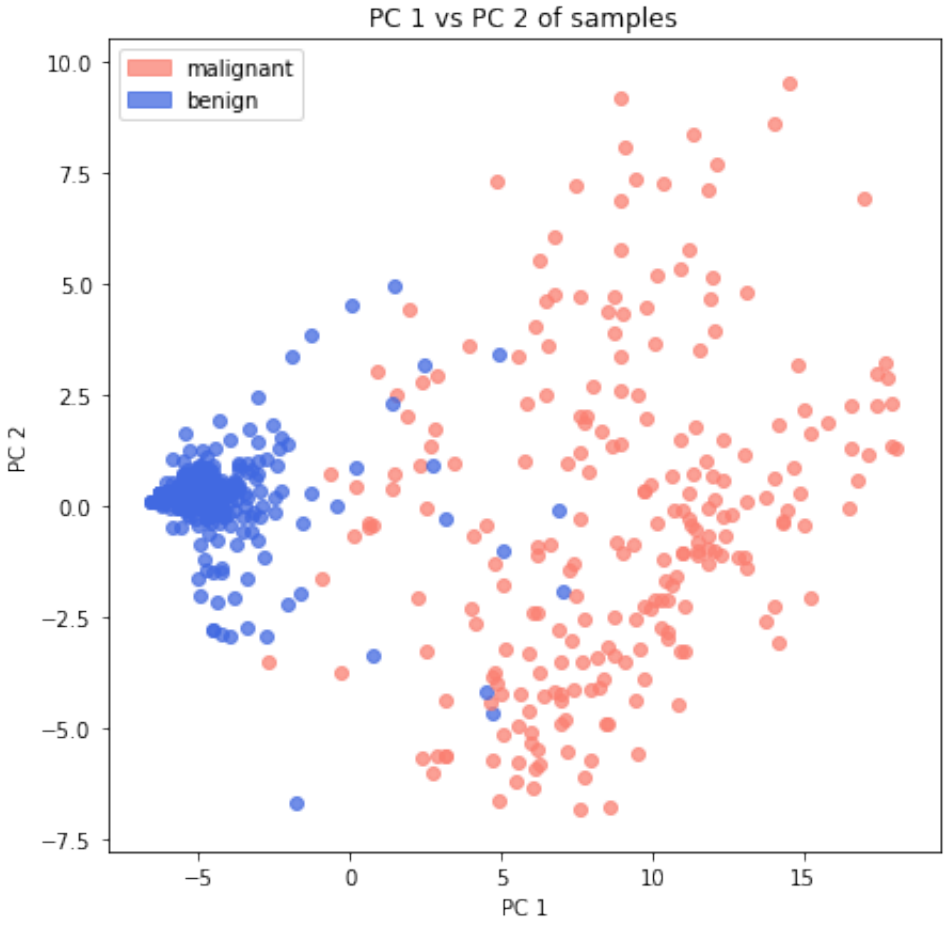
Out[11]:

	clump_thickness	size_uniformity	shape_uniformity	marginal_adhesion	epithelial_size	bare_nu
0	5	1	1	1	2	
1	5	4	4	5	7	
2	3	1	1	1	2	
3	6	8	8	1	3	
4	4	1	1	3	2	

In [12]:
```
# Making a PCA analysis for the Data

# creating seperate numpy array of labels and values
df_labels = df.iloc[:, -1].values #contains class (either 2 or 4)
df_values = df.iloc[:, 0:10].values

# importing PCA
from sklearn.decomposition import PCA

# running and fitting PCA on values of samples
pca = PCA(n_components = 2)
pca.fit(df_values)
pcs = pca.transform(df_values)

fig, ax = plt.subplots(figsize=(7,7))

# manually making legend for cancer type information. Got code from:
# https://stackoverflow.com/questions/39500265/manually-add-legend-items
import matplotlib.patches as mpatches
red_patch = mpatches.Patch(color='salmon', label='malignant',
                           alpha=0.75)
blue_patch = mpatches.Patch(color='royalblue', label='benign',
                            alpha=0.75)
plt.legend(handles=[red_patch, blue_patch])


# plotting the each point with associated colors
for i in range(len(df_labels)):
    # if its associated with label 4: then its malignant
    if df_labels[i] == 4:
        ax.scatter(pcs[i,0], pcs[i,1], color='salmon', alpha=0.75)
    # else if its associated with label 2: then its benign
    elif df_labels[i] == 2:
        ax.scatter(pcs[i,0], pcs[i,1], color='royalblue',alpha=0.75)
```

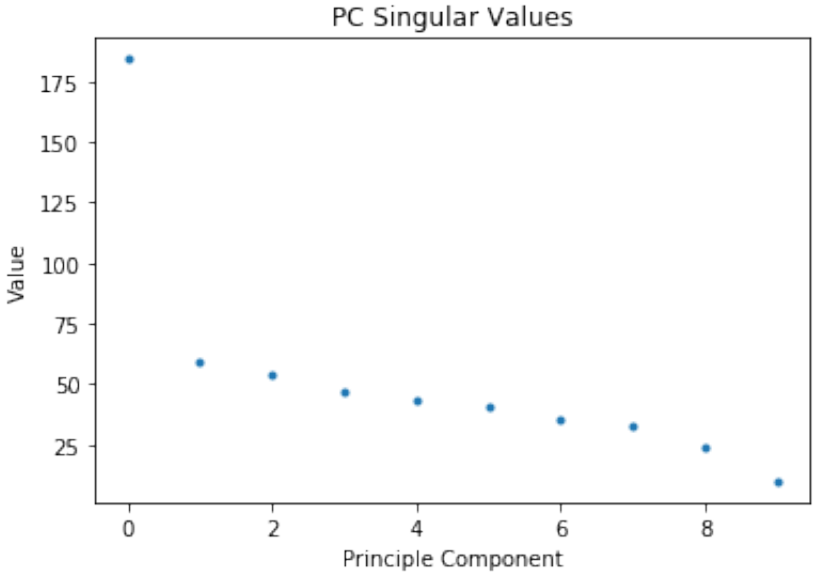
```
plt.title('PC 1 vs PC 2 of samples')
plt.xlabel('PC 1')
plt.ylabel('PC 2')

#plt.savefig("Data PC1 vs PC2.png")
```

Out[12]: Text(0, 0.5, 'PC 2')



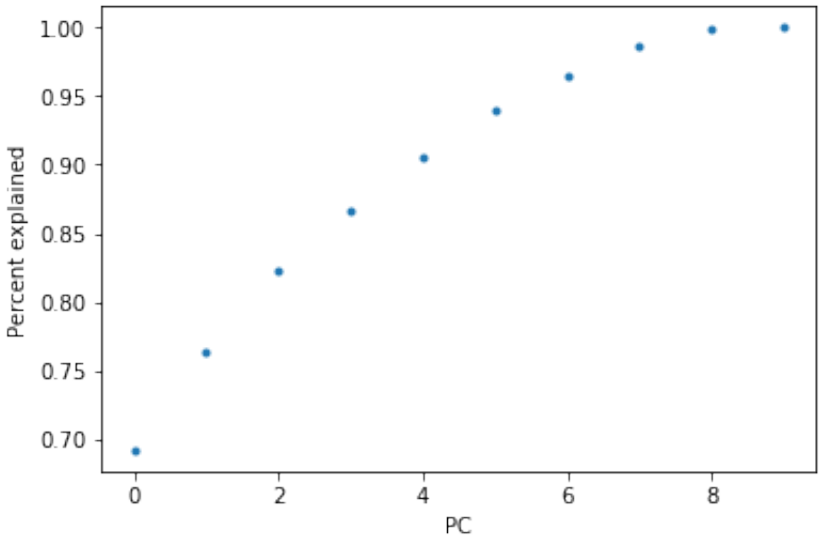
In [13]:
```
# plotting pca singular values
pca = PCA()
pca.fit(df_values)

plt.plot(pca.singular_values_, '.')
plt.title('PC Singular Values')
plt.xlabel('Principle Component')
plt.ylabel('Value')
```

Out[13]: Text(0, 0.5, 'Value')



In [14]:
```
plt.plot(pca.explained_variance_ratio_.cumsum(), '.')
plt.xlabel('PC')
plt.ylabel('Percent explained')
```

Out[14]: Text(0, 0.5, 'Percent explained')



In [15]:
```
# Making Initial Nearest Neighbor Classifier
from sklearn.neighbors import KNeighborsClassifier
```

In [16]:
```
# Function that splits training and testing data randomly
def split(df, frac_p):
    train_data = df.sample(frac = frac_p)
    test_data = df.drop(train_data.index)
    return(train_data, test_data)
```

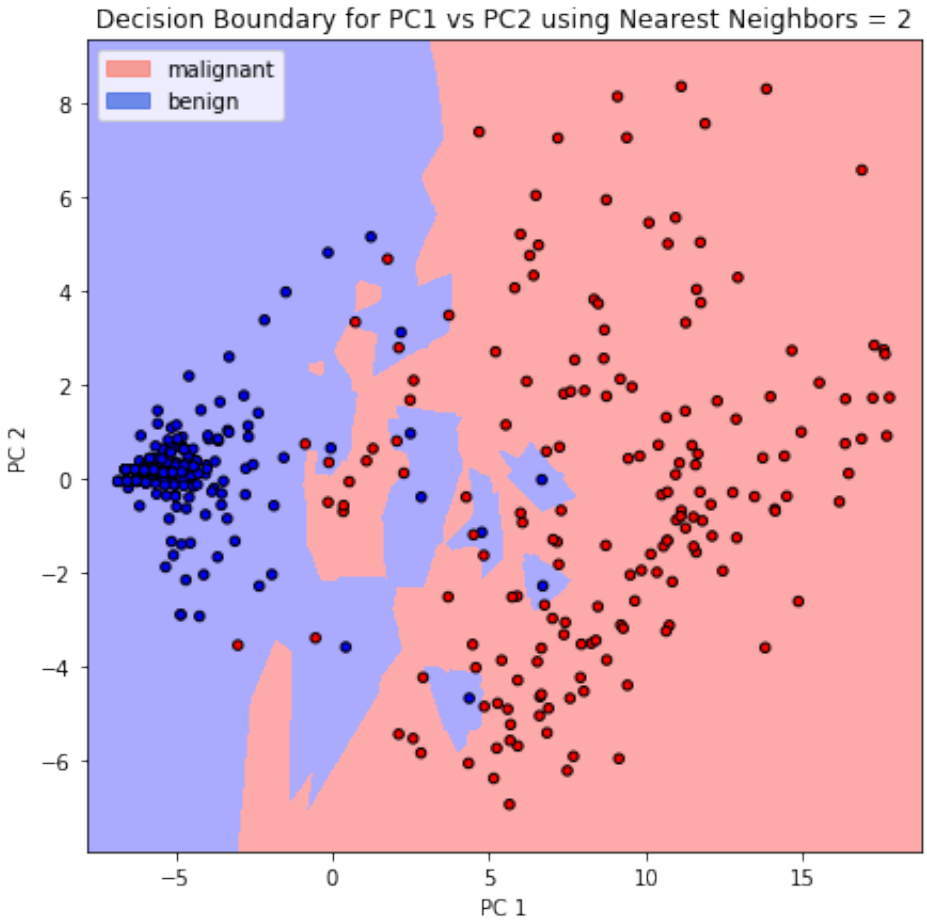
In [17]:
```
# splitting training and testing data
train_data, test_data = split(df, 0.7)
train_values = train_data.iloc[:, 0:10].values
train_labels  = train_data.iloc[:, -1].values
train_labels = train_labels.reshape(478, 1)

test_values = test_data.iloc[:, 0:10].values
test_labels  = test_data.iloc[:, -1].values
```

In [18]:
```
pca = PCA(n_components = 2)
pca.fit(train_values)
pcs = pca.transform(train_values)
```

In [64]:
```
# Displaying the Decision Boundary when doing Nearest Neighbors
# https://scikit-learn.org/stable/auto_examples/neighbors/plot_classific
```

```
# http://scikit-learn.org/stable/auto_examples/neighbors/plot_classific

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 2

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = pcs
y = train_labels

h = .02  # step size in the mesh

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

for weights in ['uniform']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y.ravel())

    # Plot the decision boundary. For that, we will assign a color to ea
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(figsize=(7,7))
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y.ravel(), cmap=cmap_bold,
                edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())

red_patch = mpatches.Patch(color='salmon', label='malignant',
                           alpha=0.75)
blue_patch = mpatches.Patch(color='royalblue', label='benign',
                            alpha=0.75)
plt.legend(handles=[red_patch, blue_patch])

plt.xlabel('PC 1')
```

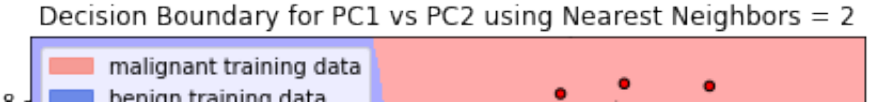
```
plt.ylabel('PC 2')
plt.title("Decision Boundary for PC1 vs PC2 using Nearest Neighbors = 2"
#plt.savefig("KNN1.png")
plt.show()
```

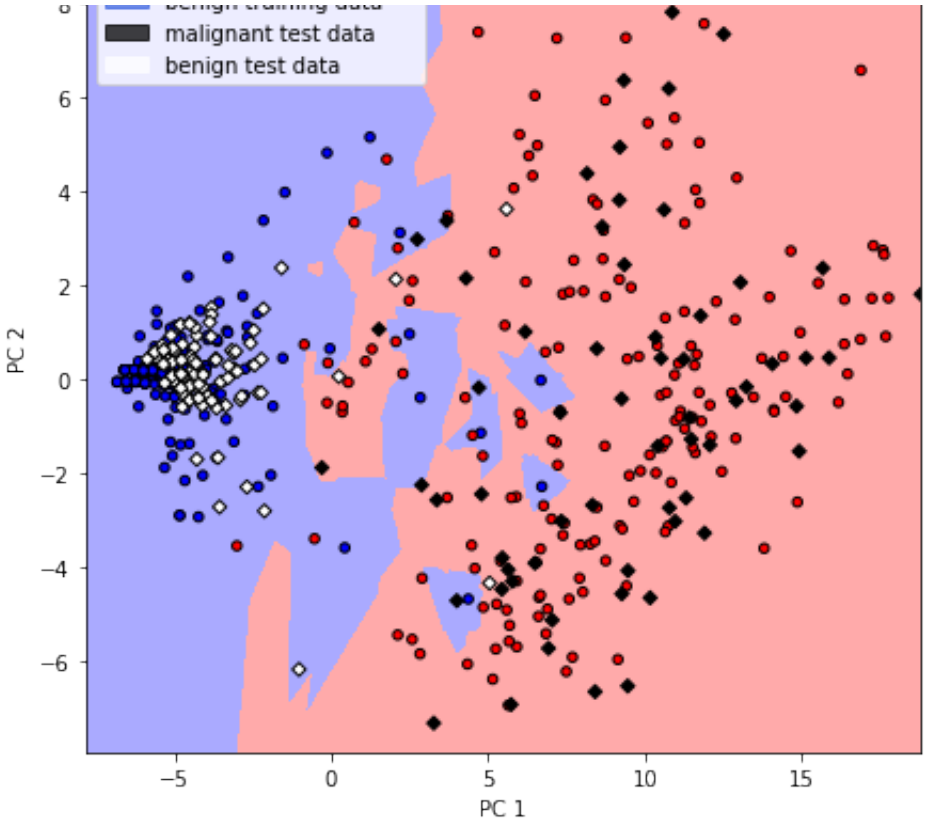


In [65]:

```
# Displaying the Decision Boundary when doing Nearest Neighbors
# https://scikit-learn.org/stable/auto_examples/neighbors/plot_classific

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 2

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = pcs
y = train_labels

h = .02  # step size in the mesh

# Create color maps
```

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

for weights in ['uniform']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y.ravel())

    # Plot the decision boundary. For that, we will assign a color to ea
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(figsize=(7,7))
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y.ravel(), cmap=cmap_bold,
                edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())

#plotting test points
cmap_test = ListedColormap(['white','black'])
pca = PCA(n_components = 2)
pca.fit(test_values)
test_pcs = pca.transform(test_values)
w = test_labels
plt.scatter(test_pcs[:, 0], test_pcs[:, 1], c=w.ravel(), marker = "D",
            edgecolor='k', s=20)

red_patch = mpatches.Patch(color='salmon', label='malignant training dat
blue_patch = mpatches.Patch(color='royalblue', label='benign training da
black_patch = mpatches.Patch(color='black', label='malignant test data',
white_patch = mpatches.Patch(color='white', label='benign test data',alp
plt.legend(handles=[red_patch, blue_patch, black_patch, white_patch])

plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.title("Decision Boundary for PC1 vs PC2 using Nearest Neighbors = 2"
#plt.savefig("KNN2.png")
plt.show()
```




malignant test data
benign test data
PC 2
PC 1
6
4
2
0
−2
−4
−6
−5
0
5
10
15

In [20]:
```
# function adapted from HW5 to take in classifier and give % accuracy in
def class_accuracy(model, X, r, test_frac, reps):
    total = np.zeros(reps)
    for i in range(reps):
        # split into random test and train arrays
        train_data, test_data = split(X, test_frac)

        # seperating train values/labels and test values/labels
        train_values = train_data.iloc[:, 0:10].values
        train_labels  = train_data.iloc[:, -1].values
        test_values = test_data.iloc[:, 0:10].values
        test_labels  = test_data.iloc[:, -1].values

        # making PCA
        pca = PCA(n_components = r)
        pca.fit(train_values)
        Xtrain_trans = pca.transform(train_values)
        test_trans = pca.transform(test_values)

        classifier = mymodel
        classifier.fit(Xtrain_trans, train_labels)

        score = classifier.score(test_trans, test_labels)
        total[i] = score
    #print("mean accuracy for classifier is", total / reps)
    cv_acc = np.mean(total) * 100
    cv_std = np.std(total) * 100
    return(cv_acc, cv_std)
```

In [21]:
```
# cross_val_class_accuracy(model, df, #components, test_frac = 0.3, #of

# KNN 2 neighbors, r=2 components
mymodel=KNeighborsClassifier(n_neighbors=2)
knn_acc_1, knn_std = class_accuracy(mymodel, df, 2, 0.3, 100)
print(knn_acc_1, knn_std)
```

```
94.7008368200837 1.2646283451890843
```

In [22]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [23]:
```
#LDA, r=2 components
mymodel=LinearDiscriminantAnalysis()
lda_acc_1, lda_std = class_accuracy(mymodel, df, 2, 0.3, 100)
print(lda_acc_1, lda_std)
```

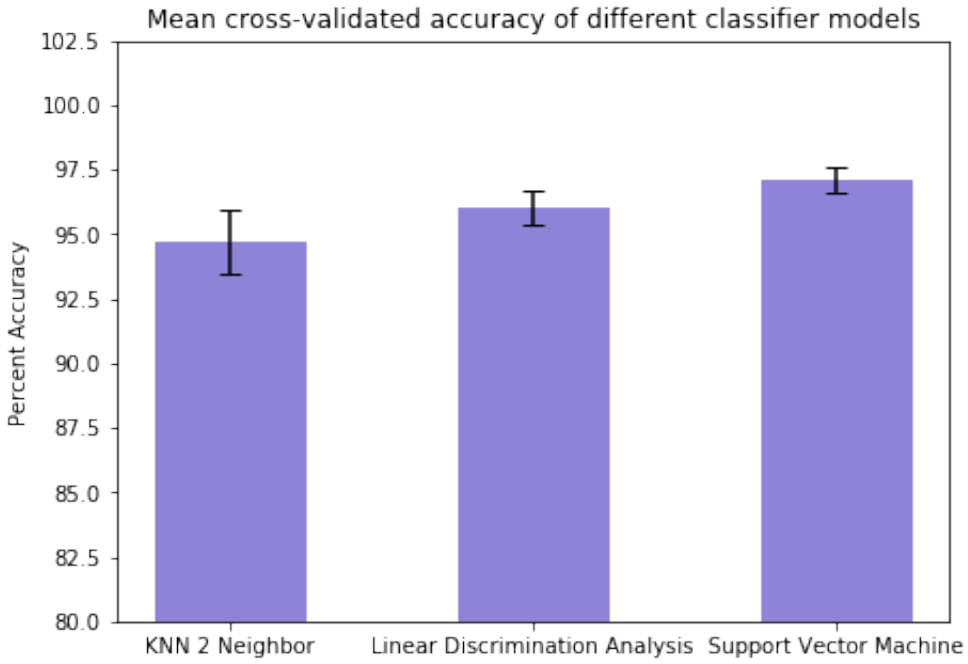
```
96.01882845188285 0.6230341889794202
```

In [24]:
```
# https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vect
from sklearn import svm
```

In [25]:
```
# support vector machine, linear kernel, r = 2
mymodel = svm.SVC(kernel='linear')
svm_acc, svm_std = class_accuracy(mymodel, df, 2, 0.3, 100)
print(svm_acc, svm_std)
```

```
97.0836820083682 0.5045442211176163
```

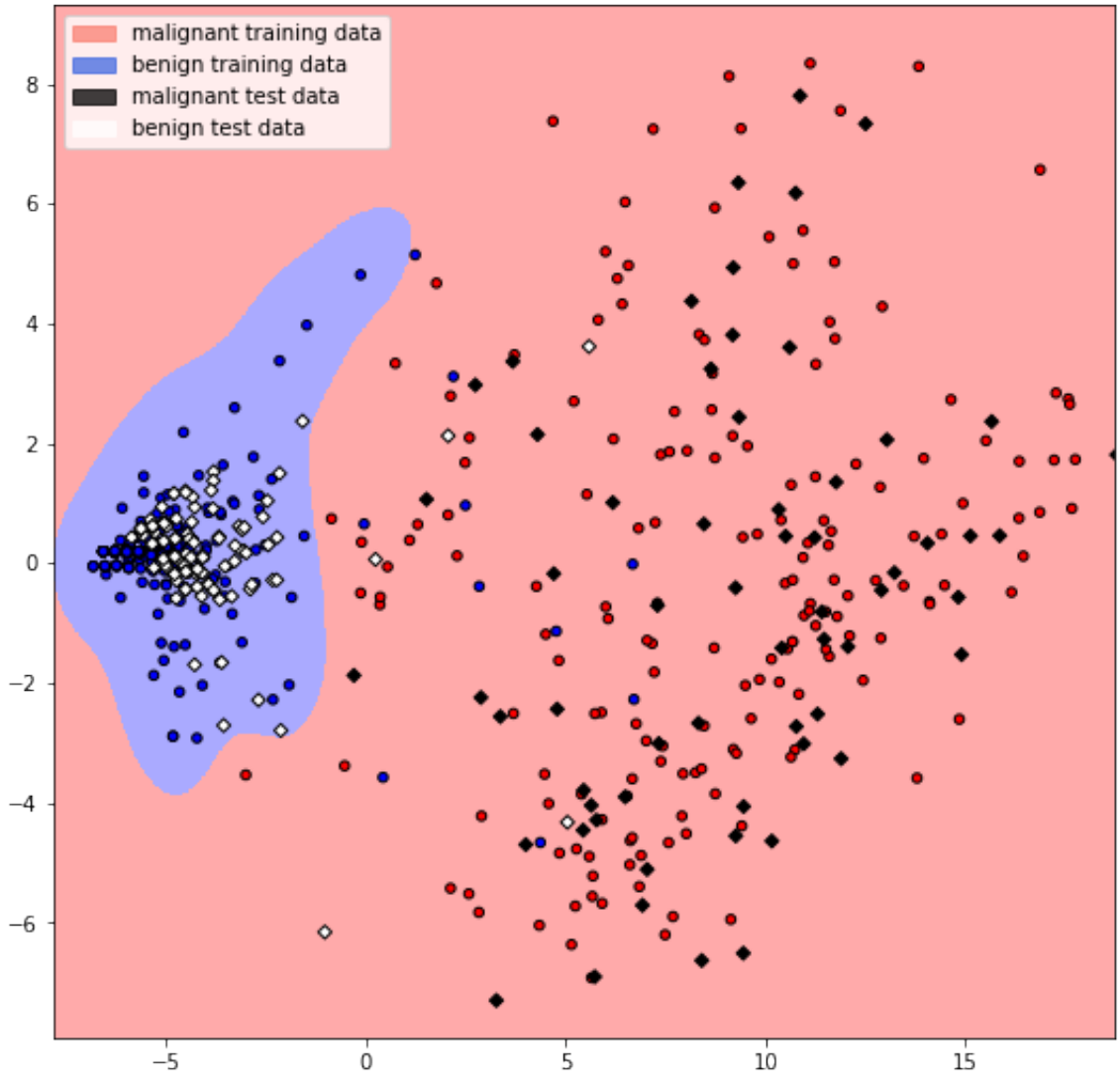
In [66]:
```
classifier_names = ('KNN 2 Neighbor', 'Linear Discrimination Analysis',
cross_acc = np.array([knn_acc_1,lda_acc_1, svm_acc])
cross_std = np.array([knn_std, lda_std, svm_std])
fig = plt.figure(figsize = (7,5))
ax1 = fig.add_subplot(1,1,1)

ax1.bar(np.arange(0, 3), cross_acc, align = 'center', color = 'slateblue
ax1.errorbar(np.arange(0, 3), cross_acc, yerr=cross_std, linestyle='none

plt.ylim(bottom = 80)
plt.xticks( np.arange(0, 3), classifier_names)
plt.ylabel('Percent Accuracy')
plt.title('Mean cross-validated accuracy of different classifier models'

#plt.savefig("means.png")
```



In [27]:
```
def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

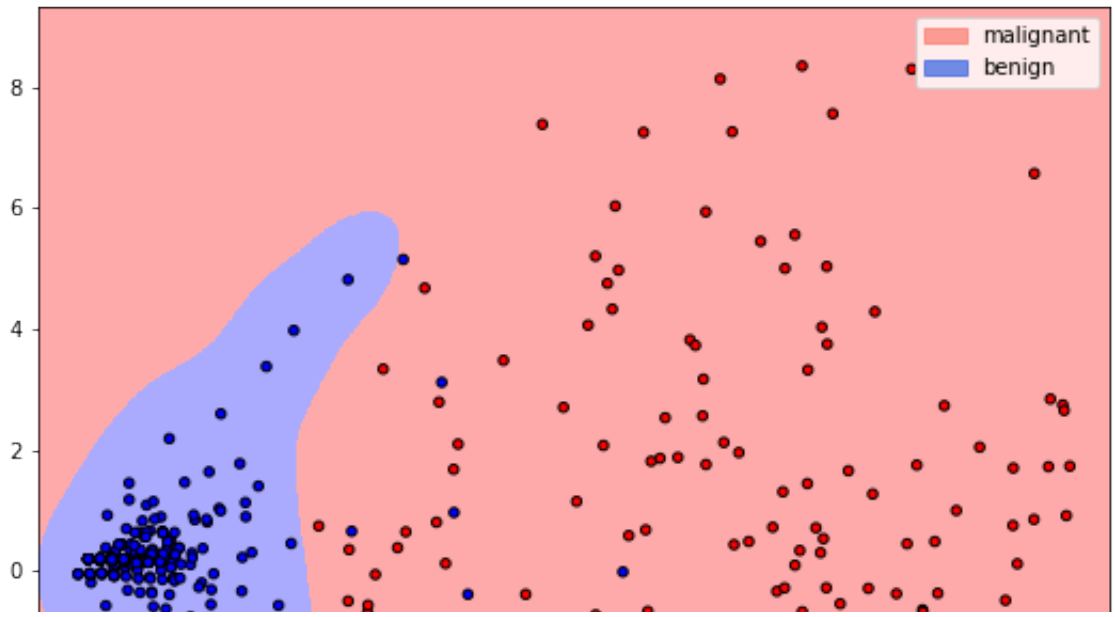
In [58]:
```
# Plotting SVM
# https://scikit-learn.org/stable/auto_examples/svm/plot_iris.html
from sklearn import svm, datasets
# Take the first two features. We could avoid this by using a two-dim da
X = pcs
y = train_labels

# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
clf = svm.SVC(kernel='rbf', gamma=0.7,  C=C)
clf.fit(X, y.ravel())

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
#cmap_light = ListedColormap(['#DBDBFF', '#AAFFAA', '#FFD3D3'])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(9,9))
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y.ravel(), cmap=cmap_bold,
            edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

plot_contours(ax, clf, xx, yy,
              cmap=cmap_light, alpha=0.8)

# Plot Test Points
cmap_test = ListedColormap(['white','black'])
pca = PCA(n_components = 2)
pca.fit(test_values)
test_pcs = pca.transform(test_values)
w = test_labels
plt.scatter(test_pcs[:, 0], test_pcs[:, 1], c=w.ravel(), marker = "D",
            edgecolor='k', s=20)
```

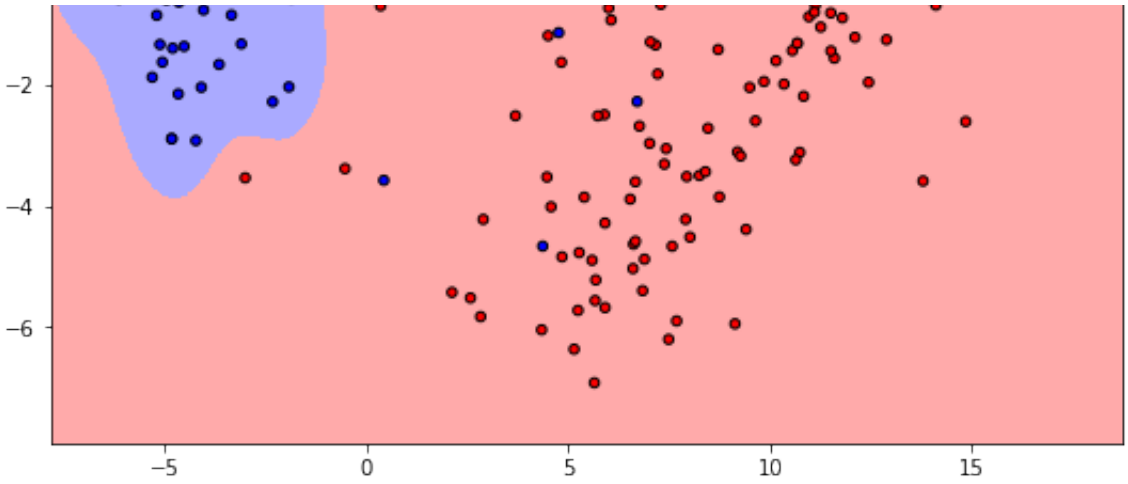
```
red_patch = mpatches.Patch(color='salmon', label='malignant training dat
                           alpha=0.75)
blue_patch = mpatches.Patch(color='royalblue', label='benign training da
                            alpha=0.75)
black_patch = mpatches.Patch(color='black', label='malignant test data',
                            alpha=0.75)
white_patch = mpatches.Patch(color='white', label='benign test data',
                            alpha=0.75)
plt.legend(handles=[red_patch, blue_patch, black_patch, white_patch])
plt.show()
```



In [59]:
```
# Plotting SVM
# https://scikit-learn.org/stable/auto_examples/svm/plot_iris.html
from sklearn import svm, datasets
# Take the first two features. We could avoid this by using a two-dim da
X = pcs
y = train_labels

# we create an instance of SVM and fit out data. We do not scale our
```

```
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
clf = svm.SVC(kernel='rbf', gamma=0.7,  C=C)
clf.fit(X, y.ravel())

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
#cmap_light = ListedColormap(['#DBDBFF', '#AAFFAA', '#FFD3D3'])
Z = Z.reshape(xx.shape)
plt.figure(figsize=(9,9))
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y.ravel(), cmap=cmap_bold,
            edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

plot_contours(ax, clf, xx, yy,
              cmap=cmap_light, alpha=0.8)

red_patch = mpatches.Patch(color='salmon', label='malignant',
                           alpha=0.75)
blue_patch = mpatches.Patch(color='royalblue', label='benign',
                            alpha=0.75)
plt.legend(handles=[red_patch, blue_patch])
plt.show()
```





In [ ]: