Sergey Ivanov
Problem 1:

(a) Using 50 as the number of observations in a leaf and 7 as the maximum number of levels to be displayed, we have the following rules:

I denote class 0 as non-competetive auction, and class 1 as competitve auction

If (open price <= 2.449) and (close price <=2.025) then class=0
If (open price <= 2.449) and (close price > 2.025) then class=1
If (4.919>= open price > 2.449) and (close price <=4.195) then class=0
If (4.919>= open price > 2.449) and (4.195< close price <=10.50) then class=1
If (open price > 4.919) and (close price <=10.50) then class=0
If (11.069>= open price > 2.449) and (close price > 10) then class=1
If (open price > 11.0668) and (close price > 10.05) and (seller rating > 553) then class=0
If (open price > 11.0668) and (close price > 10.05) and (seller rating <= 553) then class=1

(b) We can say that it's not practical because it doesn't output the closed price.

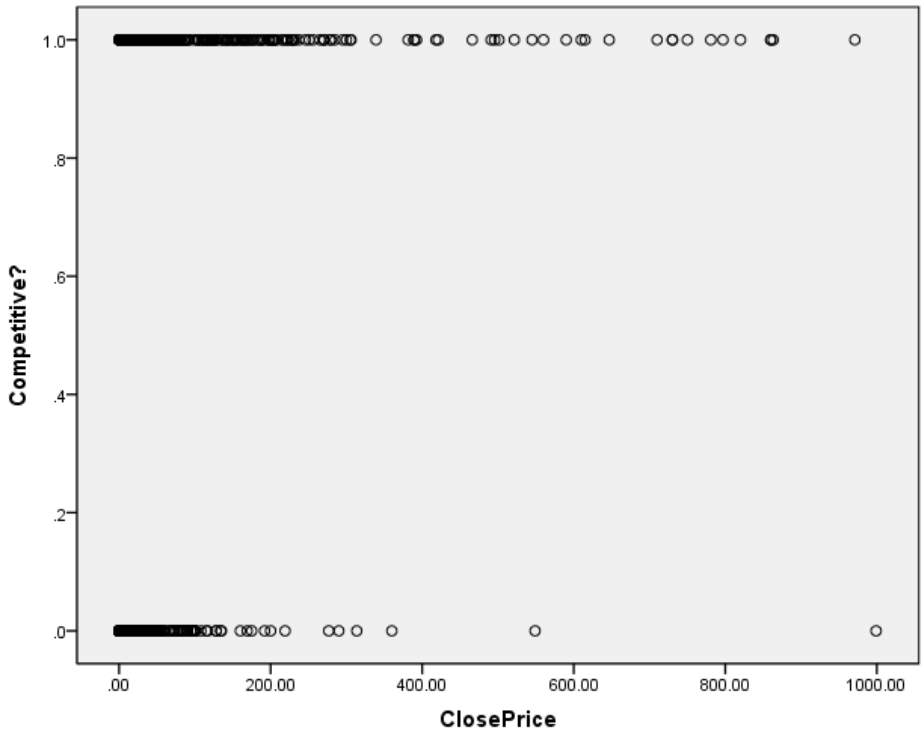
(c) Interesting is that high open and closed prices are associated with low seller ratings meaning that competitve auction are generated by amateur sellers with high open and close prices.

(d) Using the predictors that can predict outcome of a new auction

If (open price <= 1.23) then class=1
If (1.23< open price <= 1.7373) and then class=0
If (1.7373< open price <= 2.4498) then class=1
If (2.4498< open price >= 9.97) and (seller rating < 557) then class=0
If (open price > 9.97) and (seller rating <= 557) then class=1
If (open price > 2.4498) and (557< seller rating <= 2295) then class=0
If (open price > 2.4498) and (2295< seller rating <= 2370) then class=1
If (open price > 2.4498) and (2370< seller rating <= 3350) then class=0
If (open price > 2.4498) and (seller rating > 3350) then class=0

(e)

Sergey Ivanov



(e)
The accuracy for validation data set is (279+299)*100/789=73.26%. For competitive auctions the error is 64%, while for non-competitive auctions the error is 16%.

(g) Opening price is the most important factor. The lower opening price, the more bidders there are, hence the more chance auction to competitive. Based on results if opening price is less than 1.23 then the auction is predicted to be a competitive.

Code:

```
# Classification Tree with rpart
library(rpart)

attach
# grow tree
fit <- rpart(competitve~ . ,
        method="class", data=table)

prune(fit) # prune fit
printcp(fit) # display the results
plotcp(fit) # visualize cross-validation results
```

Sergey Ivanov

```
summary(fit) # detailed summary of splits

# plot tree
plot(fit, uniform=TRUE,
     main="Classification Tree for Kyphosis")
text(fit, use.n=TRUE, all=TRUE, cex=.8)
```

Sergey Ivanov

Problem 2:

A test set of size 1565 was randomly chosen, leaving 3036 observations in the training set.

Additive model:

pred	nonspam	spam
nonspam	58.3	2.5
spam	3.0	36.3

Total error: 5.5

SVM:

pred	nonspam	spam
nonspam	58.01	3.7
spam	2.17	36.1

Total error: 5.87

Interpretability of SVM:

Parameters:

SVM-Type: C-classification

SVM-Kernel: radial

cost: 100

gamma: 0.001

Number of Support Vectors: 596

(300 296)

Number of Classes: 2

Levels:

nonspam spam

10-fold cross-validation on training data:

Total Accuracy: 93.84058

Single Accuracies:

95.0495 93.09211 95.0495 93.09211 92.10526 96.36964 95.06579 93.06931 91.77632 93.75

Neural Net:

pred	nonspam	spam
nonspam	57.12	2.7
spam	3.6	36.4

Total error: 6.3

Sergey Ivanov
Interpretability of NNet:
a 57-10-1 network with 591 weights

In general NNet is significantly harder to interpret than SVM or additive model.

For significance of each predictor we can leave it out and run the model again calculating total error: the discrepancy of a predictor will be an indicator of significance.

Additional information can be found using summary for models.

Code:

```
data(spam)

spm_size <- floor(.66*nrow(spam))
train_ind <-sample(seq_len(nrow(spam)), size = spm_size)

train <- spam[train_ind,]
test <- spam[-train_ind,]

# svm
require(kernlab)
tobj <- tune.svm(type ~ ., data = train[1:300,], gamma=10^(-6:-3), cost=10^(1:2))
Gamma <- tobj$best.parameters[[1]]
C <- tobj$best.parameters[[2]]

SVM_model <- svm(type ~ ., data=train, cost=C, gamma=Gamma, cross=10)
summary(SVM_model)

pred <- predict(SVM_model, test)
acc <- table(pred, test$type)
acc/sum(acc)*100

# NNet
require(nnet)
NN_model <- nnet(type~., data=train, size=10)
pred <- predict(NN_model, test[-58], type="class")
acc <- table(pred, test$type)
acc/sum(acc)*100
```

Sergey Ivanov
Problem 3:

(a) We have a customer with the following characteristics: Age=40, Experience=10, Income=84, Family=2, CCAvg=2, Education=2, Mortgage=0, Securities Account=0, CD Account=0, Online=1 CreditCard=1. Dropping ID and ZipCode features and using k=1, we have predicted class equals one (loan acceptance).

(b)
For first 20 values of k we have the following table:
[1] 1.00 4.05
[1] 2.00 4.65
[1] 3.0 3.9
[1] 4.00 4.25
[1] 5.00 4.55
[1] 6.00 4.75
[1] 7.00 5.25
[1] 8.00 5.55
[1] 9.0 5.4
[1] 10.0 5.4
[1] 11.0 5.7
[1] 12.00 6.05
[1] 13.0 6.1
[1] 14.00 6.15
[1] 15.00 6.55
[1] 16.00 6.75
[1] 17.00 6.75
[1] 18.00 6.95
[1] 19.00 7.05
[1] 20.00 7.15

Thus, k = 3 gives the best accuracy and it balances between overfitting and ignoring the predictor information.

(c)

The accuracy rate is the following:
knn_model 0 1
0 90.20% 3.45%
1 0.45% 5.90%

(d) For the same customer as in (a) we have a class 1. For all other customers we run code.

(e) k=3 gives the best accuracy in validation set.
Classification matrices:

```
train_pred     0     1
         0 89.84  2.36
         1  0.28  7.52
Total error: 2.64
```

```
val_pred          0          1
       0 90.46              4.06
       1  0.26               5.2
Total error: 4.32
```

```
test_pred    0    1
        0      89.9  3.8
        1       0.7  5.6
Total error: 4.5
```

On training set the error is close to zero (in fact, for k = 1, it will be 0). For validation set, when we select a model the error is expectedly higher. For new data we have slightly higher error, 4.5.

Code:

```
require(class)
banks <- read.table("banks.csv", header=T, sep=",")

# convert Education categorical variable
banks$Education1 <- rep(0, dim(banks)[1])
banks$Education2 <- rep(0, dim(banks)[1])
banks$Education3 <- rep(0, dim(banks)[1])
for (i in 1:dim(banks)[1]) {
  if (banks[i,8] == 1) {banks$Education1[i] <- 1}
  else if (banks[i,8] == 2) {banks$Education2[i] <- 1}
  else if (banks[i,8] == 3) {banks$Education3[i] <- 1}
}
banks <- banks[,-8] # remove Education variable

# drop variables
drops <- c("ID", "ZIP.Code")
```

```
banks_s <- banks[,!(names(banks) %in% drops)]
# append this customer to the end
customer <- data.frame(Age=40, Experience=10, Income=84, Family=2, CCAvg=2,
Mortgage=0, Securities.Account=0, CD.Account=0, Online=1, CreditCard=1, Education1=0,
Education2=1, Education=0)
banks_s[dim(banks_s)[1] + 1,] = customer
# normalize
normalize <- function (x) {
  return ( (x - min(x))/(max(x) - min(x)))
}
banks_n <- as.data.frame(lapply(banks_s[,-7], normalize))


spm_size <- floor(.6*nrow(banks))
train_ind <-sample(seq_len(nrow(banks)), size = spm_size)

train <- banks_n[train_ind,]
test <- banks_n[-train_ind,]
test <- test[-2001,] # drop this customer

# a
knn_model <- knn(train, banks_n[5001, ], banks[train_ind, 9], k=1)

# b
min_error = 100
min_k = -1
for (i in 1:20) {
  knn_model <- knn(train, test, banks[train_ind, 9], k=i)
  pred <- table(knn_model, banks[-train_ind, 9])
  error <- (pred[2,1] + pred[1,2])/sum(pred)*100
  print(c(i, error))
  if (error < min_error) {
    min_error <- error
    min_k <- i
  }
}

# c
knn_model <- knn(train, test, banks[train_ind, 9], k=min_k)
pred <- table(knn_model, banks[-train_ind, 9])
```

```
# d
customer_class <- knn(train, banks_n[5001, ], banks[train_ind, 9], k=min_k)
knn_model <- knn(train, banks_n, banks[train_ind, 9], k=min_k)
pred <- table(knn_model, banks_n[, 9])

# e
# found at http://stackoverflow.com/a/20056357/2069858
split_data <- function(dat, props = c(.5, .3, .2), which.adjust = 1){
  stopifnot(all(props >= 0), which.adjust <= length(props))
  props <- props/sum(props)
  n <- nrow(dat)
  ns <- round(n * props)
  ns[which.adjust] <- n - sum(ns[-which.adjust])
  ids <- rep(1:length(props), ns)
  which.group <- sample(ids)
  split(dat, which.group)
}
banks_normed <- as.data.frame(lapply(banks_s, normalize))
d <- split_data(banks_normed)
new_train <- d$`1`
new_validate <- d$`2`
new_test <- d$`3`

for (i in 1:20) {
val_pred <- knn(new_train[,-7], new_validate[,-7], new_train[,7],k=i)
class_mtx <- table(val_pred, new_validate[,7])
print(c(i, (class_mtx[1,2] + class_mtx[2,1])/sum(class_mtx)*100))
}
# k=3 gives the best accuracy in validation set
train_pred <- knn(new_train[,-7], new_train[,-7], new_train[,7],k=3)
class_mtx <- table(train_pred, new_train[,7])
class_mtx/sum(class_mtx)*100

val_pred <- knn(new_train[,-7], new_validate[,-7], new_train[,7],k=3)
class_mtx <- table(val_pred, new_validate[,7])
class_mtx/sum(class_mtx)*100

test_pred <- knn(new_train[,-7], new_test[,-7], new_train[,7],k=3)
class_mtx <- table(test_pred, new_test[,7])
class_mtx/sum(class_mtx)*100
```

Sergey Ivanov
Problem 4:

(a) I use two different algorithms for clustering: k-means and hierarchical agglomerative clustering (HAC).
For HAC I use two different methods, single and complete, for clustering data. In order to evaluate the number of classes I use elbow method that is I compare the “jump” of percentage of variance than it was before.
For k-means I calculate classes for k=1..9

Depending on the number of classes I have the following partitions:
Hierarchical clustering (single)
2 [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
6 [1, 6, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 4, 1, 3, 1, 5, 2, 1, 1, 1]
Hierarchical clustering (complete)
2 [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2]
3 [2, 3, 3, 2, 2, 3, 2, 3, 3, 2, 1, 3, 1, 3, 1, 2, 1, 3, 2, 3, 2]

K-Means
2 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2, 2, 1, 1, 1, 1]
3 [1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 3, 2, 3, 2, 3, 1, 3, 1, 2, 2, 1]
4 [1, 4, 4, 1, 1, 4, 1, 4, 4, 1, 3, 4, 2, 4, 3, 1, 2, 1, 4, 4, 1]
5 [5, 4, 1, 5, 5, 1, 5, 1, 1, 5, 2, 1, 3, 1, 2, 2, 3, 5, 5, 1, 5]
6 [1, 2, 4, 1, 5, 4, 1, 4, 4, 1, 6, 4, 3, 4, 6, 1, 3, 2, 5, 4, 1]
7 [5, 3, 7, 5, 5, 4, 5, 7, 1, 5, 6, 7, 2, 1, 6, 6, 2, 5, 5, 1, 5]
8 [4, 2, 5, 4, 7, 1, 7, 5, 5, 4, 8, 5, 6, 5, 8, 3, 6, 4, 7, 5, 7]
9 [1, 4, 7, 1, 9, 7, 3, 7, 8, 1, 2, 7, 2, 6, 2, 5, 2, 9, 3, 6, 3]

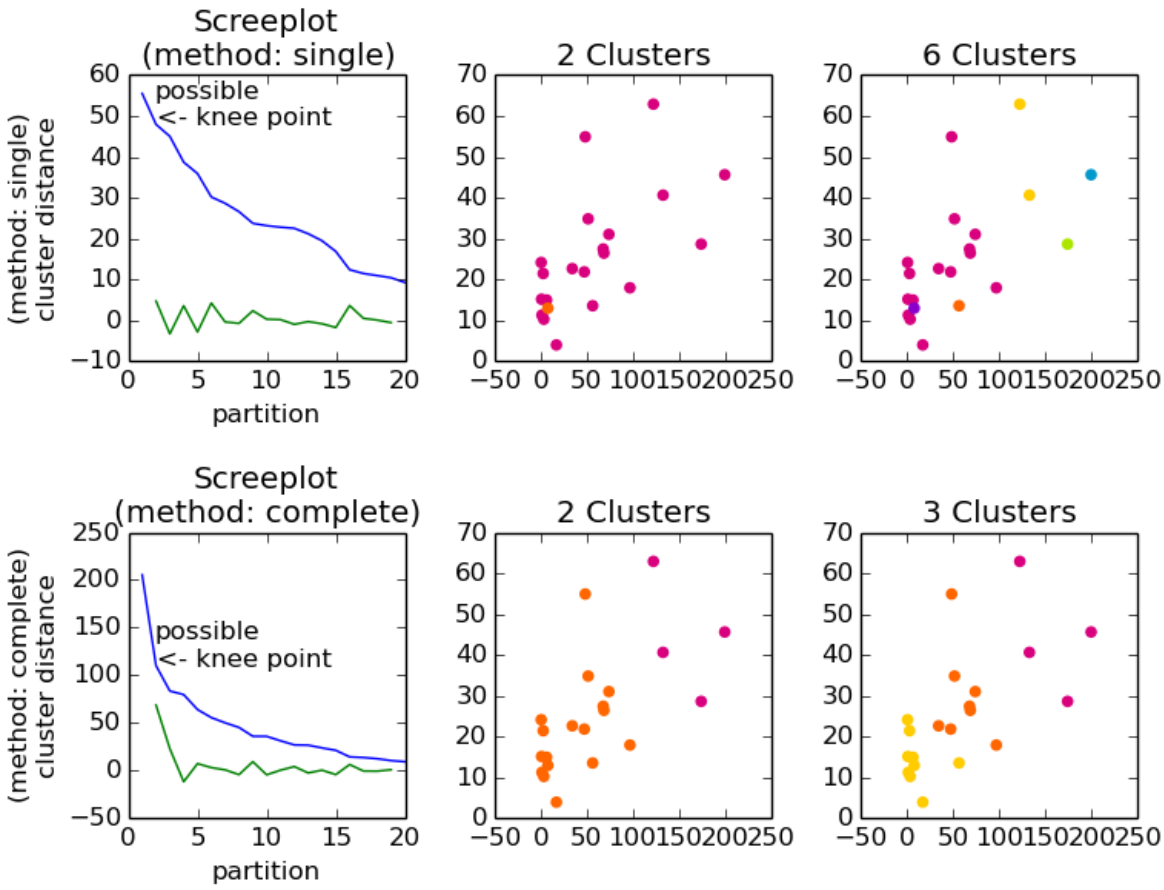
The first column is k, the number of clusters, which followed by the assignment of partitions of 21 firms.

We can see that HAC(complete) performs similar (but not the same) as k-means.

The number of clusters around 2-3 should be a reasonable choice.

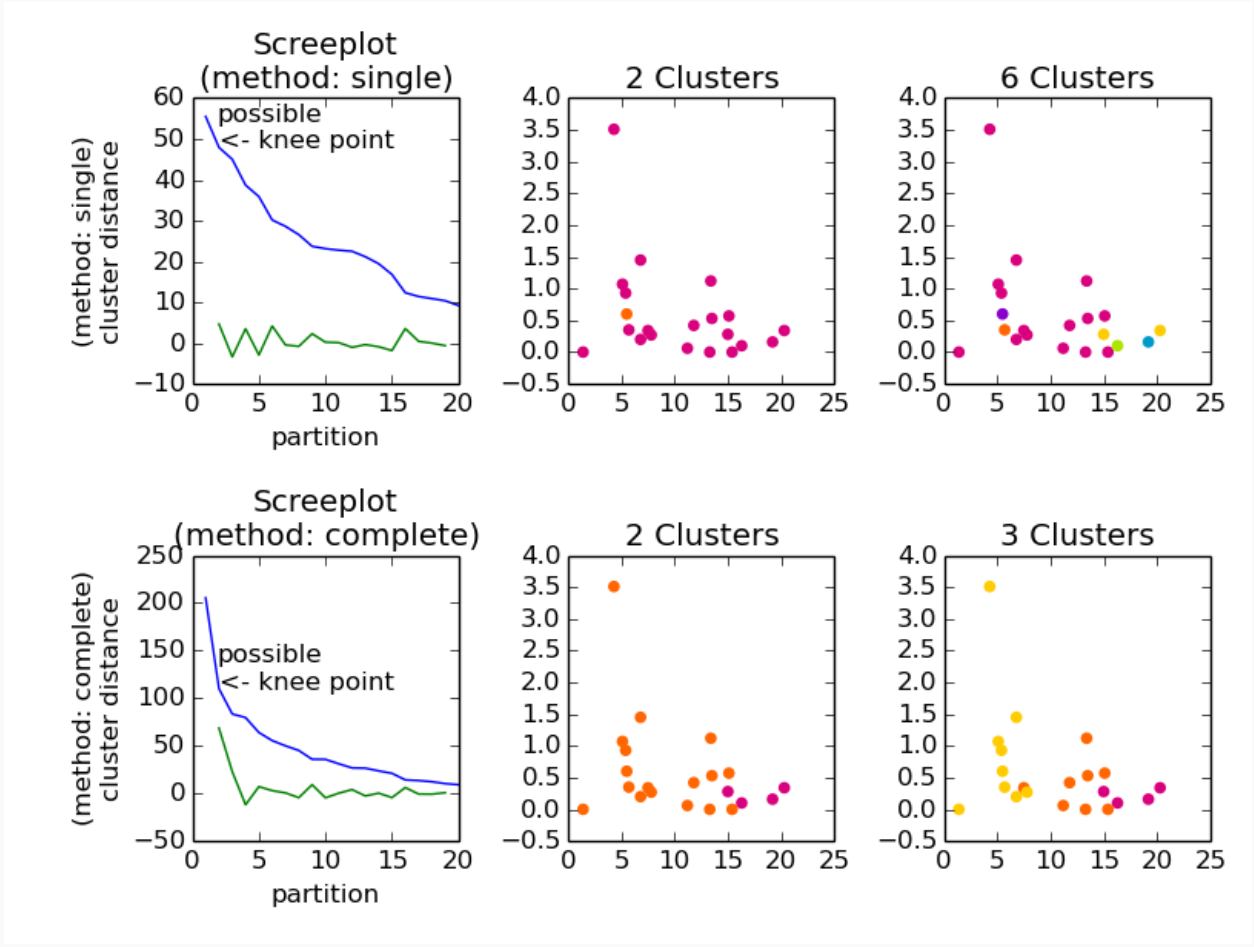
(b)

We have the following plots for market cap vs beta values.



It means that we will cluster nodes with low market cap most likely to the same cluster regardless of the value of beta.

For values of ROE vs Leverage we have the following picture:



As we see HAC tends to cluster almost all nodes to the same cluster for these features.

(c)
It look that location and exchange features do not play significant role in clusterization. However, firms with median_recommendation that equal to Hold or Moderate Buy tend to be in the same group based on k-means and HAC.

(d)

We have the following clusters:

Abbott Laboratories 2
AstraZeneca PLC 2
Aventis 2
Bristol-Myers Squibb Company 2
Eli Lilly and Company 2
Novartis AG 2

Schering-Plough Corporation 2
Wyeth 2

Allergan, Inc. 3
Amersham plc 3
Bayer AG 3
Chattem, Inc 3
Elan Corporation, plc 3
Medicis Pharmaceutical Corporation 3
IVAX Corporation 3
Pharmacia Corporation 3
Watson Pharmaceuticals, Inc. 3

GlaxoSmithKline plc 1
Johnson & Johnson 1
Merck & Co., Inc. 1
Pfizer Inc 1

So cluster 1 can be named as firms with higher capitalization (>100).
Cluster 2 is the one with medium capitalization (50-100)
Cluster 3 with low market capitalization (<50)

Code:

```
from __future__ import division
import numpy as np
import scipy.cluster.hierarchy as hac
import matplotlib.pyplot as plt
import pandas as pd
from scipy.cluster.vq import kmeans, vq

data = pd.read_csv("pharma.csv")
variables = data.ix[:, 2:11].as_matrix()

# hierarchical clustering
print 'Hierarchical clustering'
fig, axes23 = plt.subplots(2, 3)
for method, axes in zip(['single', 'complete'], axes23):
  print method
  z = hac.linkage(variables, method=method)

  # Plotting
  axes[0].plot(range(1, len(z)+1), z[::-1, 2])
  knee = np.diff(z[::-1, 2], 2)
  axes[0].plot(range(2, len(z)), knee)

  num_clust1 = knee.argmax() + 2
  knee[knee.argmax()] = 0
  num_clust2 = knee.argmax() + 2

  axes[0].text(num_clust1, z[::-1, 2][num_clust1-1], 'possible\n<- knee point')

  part1 = hac.fcluster(z, num_clust1, 'maxclust')
  print num_clust1, list(part1)
  part2 = hac.fcluster(z, num_clust2, 'maxclust')
  print num_clust2, list(part2)
  for i in range(len(part2)):
    print data.ix[i, 1], part2[i]

  clr = ['#2200CC' ,'#D9007E' ,'#FF6600' ,'#FFCC00' ,'#ACE600' ,'#0099CC' ,
  '#8900CC' ,'#FF0000' ,'#FF9900' ,'#FFFF00' ,'#00CC01' ,'#0055CC']
```

```
    for part, ax in zip([part1, part2], axes[1:]):
        for cluster in set(part):
            ax.scatter(variables[part == cluster, 4], variables[part == cluster, 6],
                       color=clr[cluster])

    m = '\n(method: {})'.format(method)
    plt.setp(axes[0], title='Screeplot{}'.format(m), xlabel='partition',
             ylabel='{}\ncluster distance'.format(m))
    plt.setp(axes[1], title='{} Clusters'.format(num_clust1))
    plt.setp(axes[2], title='{} Clusters'.format(num_clust2))

plt.tight_layout()
plt.show()

# kmeans
print 'kmeans'
for i in range(2,10):
    centroids,_ = kmeans(variables,i)
    idx,_ = vq(variables, centroids)
    print i, [el if el else i for el in idx]
    print
```