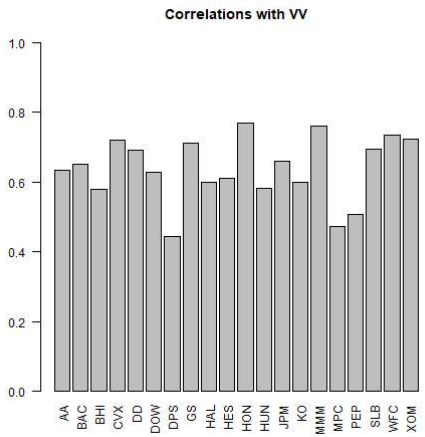
In [26]:
```
import rpy2
import rpy2.rinterface
# import rpy2.robjects as robjects
%load_ext rpy2.ipython

# Note:
# This cell creates magic commands and allows me to write in both R and Python in this notebook.
# I am much more comfortable in Python and will sometimes switch if I see it necessary to replace R's lack of functionality.
# However, most of this notebook is in R.
```

Task 1

In [3]:
```
%%R
my.data <- read.csv('stockdata.csv',sep=',', header=TRUE)
names(my.data)[1] <- "Date"
my.data$Date <- as.Date(my.data$Date, format = ('%d-%B-%y'))
sorted.df <- my.data[order(my.data$Date),]
AA <- log(sorted.df$AA[-1]/sorted.df$AA[-dim(sorted.df)[1]]);
# Manually check the first entry: log(9.45/9.23)
# Type cast the array as a data frame;
returns.df <- as.data.frame(AA);
returns.df$BAC <- log(sorted.df$BAC[-1]/sorted.df$BAC[-dim(sorted.df)[1]]);
returns.df$BHI <- log(sorted.df$BHI[-1]/sorted.df$BHI[-dim(sorted.df)[1]]);
returns.df$CVX <- log(sorted.df$CVX[-1]/sorted.df$CVX[-dim(sorted.df)[1]]);
returns.df$DD  <- log(sorted.df$DD[-1]/sorted.df$DD[-dim(sorted.df)[1]]);
returns.df$DOW <- log(sorted.df$DOW[-1]/sorted.df$DOW[-dim(sorted.df)[1]]);
returns.df$DPS <- log(sorted.df$DPS[-1]/sorted.df$DPS[-dim(sorted.df)[1]]);
returns.df$GS  <- log(sorted.df$GS[-1]/sorted.df$GS[-dim(sorted.df)[1]]);
returns.df$HAL <- log(sorted.df$HAL[-1]/sorted.df$HAL[-dim(sorted.df)[1]]);
returns.df$HES <- log(sorted.df$HES[-1]/sorted.df$HES[-dim(sorted.df)[1]]);
returns.df$HON <- log(sorted.df$HON[-1]/sorted.df$HON[-dim(sorted.df)[1]]);
returns.df$HUN <- log(sorted.df$HUN[-1]/sorted.df$HUN[-dim(sorted.df)[1]]);
returns.df$JPM <- log(sorted.df$JPM[-1]/sorted.df$JPM[-dim(sorted.df)[1]]);
returns.df$KO  <- log(sorted.df$KO[-1]/sorted.df$KO[-dim(sorted.df)[1]]);
returns.df$MMM <- log(sorted.df$MMM[-1]/sorted.df$MMM[-dim(sorted.df)[1]]);
returns.df$MPC <- log(sorted.df$MPC[-1]/sorted.df$MPC[-dim(sorted.df)[1]]);
returns.df$PEP <- log(sorted.df$PEP[-1]/sorted.df$PEP[-dim(sorted.df)[1]]);
returns.df$SLB <- log(sorted.df$SLB[-1]/sorted.df$SLB[-dim(sorted.df)[1]]);
returns.df$WFC <- log(sorted.df$WFC[-1]/sorted.df$WFC[-dim(sorted.df)[1]]);
returns.df$XOM <- log(sorted.df$XOM[-1]/sorted.df$XOM[-dim(sorted.df)[1]]);
returns.df$VV  <- log(sorted.df$VV[-1]/sorted.df$VV[-dim(sorted.df)[1]]);
```

Task 2

In [4]:
```
%%R
returns.cor <- cor(returns.df)
returns.cor[,c('VV')]
barplot(returns.cor[1:20,c('VV')],las=2,ylim=c(0,1.0))
title('Correlations with VV')
```



Task 2 Answers:

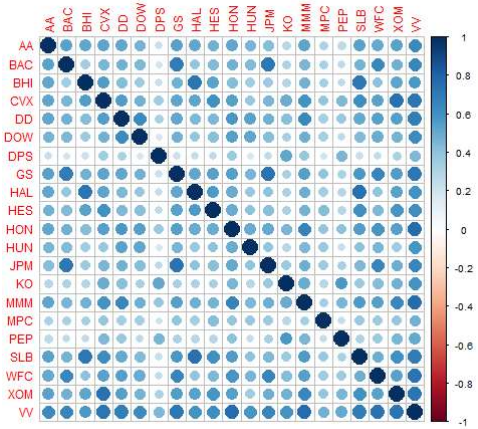
Which row or column of the correlation matrix are we most interested in?

I would say that we would be most interested in securities with high correlations of VV(So we would be looking at column VV for all rows not equal to VV).

Task 3

In [5]:
```
%%R
library(corrplot)
corrplot(returns.cor)
library(car)

# Fit some model
model.1 <- lm(VV ~ GS+DD+DOW+HON+HUN+JPM+KO+MMM+XOM, data=returns.df)
summary(model.1)
vif(model.1)

# Fit the full model
model.2 <- lm(VV ~ AA+BAC+GS+JPM+WFC+BHI+CVX+DD+DOW+DPS+HAL+HES+HON+HUN+KO+MMM+MPC+PEP+SLB+XOM,data=returns.df)
summary(model.2)
vif(model.2)
```

```
R[write to console]: corrplot 0.90 loaded

R[write to console]: Loading required package: carData

      AA      BAC       GS      JPM      WFC      BHI      CVX       DD
2.026270 2.686535 3.197811 2.875959 2.532626 2.651368 2.920187 2.441486
     DOW      DPS      HAL      HES      HON      HUN       KO      MMM
1.965886 1.525629 2.919924 2.096060 2.455876 1.743913 1.981647 2.684942
     MPC      PEP      SLB      XOM
1.376706 1.720663 3.258982 2.949826
```



Task 3 Answers:

Is the corrplot more useful or insightful than our simple barplot?

The barplot is more singular while the corrplot allows you to see all correlations between all variable within the scatter matrix.I personally use corrplot when doing correlation analysis and have never found myself using a barplot.

What is the difference between a 'statistical graphic' and a 'data visualization'?

So the corrplot is a statistical graphic while a graphing showing stock prices would be a data visualization. Statistical graphics show statistical trends within the data while data vizualizations are graphs, maps, etc. of the raw data.

With respect to the concept of multicollinearity, look at the corrplot and identify three stocks that should have low VIF values?

1. MPC - DPS
2. DPS - WFC
3. KO - MMM

Similarly, pick three stocks that should have high VIF values?

1. VV - CVX
2. BAC - GS
3. HAL - SLB

Task 4

In [6]:
```
%%R
library(car)

# Fit some model
model.1 <- lm(VV ~ GS+DD+DOW+HON+HUN+JPM+KO+MMM+XOM, data=returns.df)
summary(model.1)
vif(model.1)

# Fit the full model
model.2 <- lm(VV ~ AA+BAC+GS+JPM+WFC+BHI+CVX+DD+DOW+DPS+HAL+HES+HON+HUN+KO+MMM+MPC+PEP+SLB+XOM,data=returns.df)
summary(model.2)
vif(model.2)
```

```
      AA       BAC        GS       JPM       WFC       BHI       CVX        DD
2.026270  2.686535  3.197811  2.875959  2.532626  2.651368  2.920187  2.441486
```

```
2.026270 2.686535 3.197811 2.875959 2.532626 2.651368 2.920187 2.441486
     DOW      DPS      HAL      HES      HON      HUN       KO      MMM
1.965886 1.525629 2.919924 2.096060 2.455876 1.743913 1.981647 2.684942
     MPC      PEP      SLB      XOM
1.376706 1.720663 3.258982 2.949826
```

In [7]:
```
%%R
vif(model.1)
```

```
      GS       DD      DOW      HON      HUN      JPM       KO      MMM
2.705795 2.368257 1.919773 2.261397 1.633336 2.324600 1.473202 2.590177
     XOM
2.073721
```

Task 4 Answers:

Is multicollinearity a concern for either of these models?

Yes, there are a number of variables with a VIF score above 2.5. Overall model 2 seems to be more of a problem.

What value of VIF should make you concerned about multicollinearity?

I am concerned about anything over 2.5.

Task 5

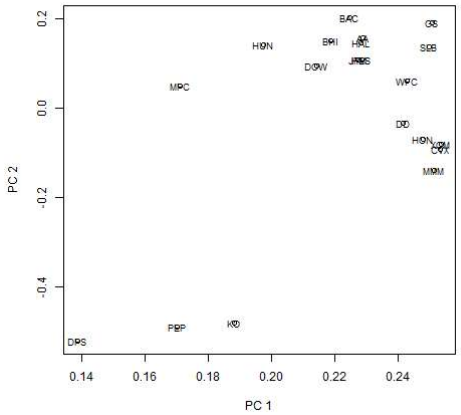
In [8]:
```
%%R
returns.pca <- princomp(x=returns.df[,-21],cor=TRUE)
names(returns.pca)
pc.1 <- returns.pca$loadings[,1];
pc.2 <- returns.pca$loadings[,2];
names(pc.1)
```

```
 [1] "AA"  "BAC" "BHI" "CVX" "DD"  "DOW" "DPS" "GS"  "HAL" "HES" "HON" "HUN"
[13] "JPM" "KO"  "MMM" "MPC" "PEP" "SLB" "WFC" "XOM"
```

In [9]:
```
%%R
names(pc.1)
```

```
 [1] "AA"  "BAC" "BHI" "CVX" "DD"  "DOW" "DPS" "GS"  "HAL" "HES" "HON" "HUN"
[13] "JPM" "KO"  "MMM" "MPC" "PEP" "SLB" "WFC" "XOM"
```

In [10]:
```
%%R
plot(x=pc.1,y=pc.2,type='p',xlab='PC 1',ylab='PC 2')
text(pc.1,pc.2,labels=names(pc.1),cex=0.75)
```



Task 5 Answers:

What are the loadings?

What groupings (or clusters) do you see in the plot of the first two principal components?

The graph above shows 3 primary clusters. The first one being in the lower left quadrant. The second in the upper middle-right quadrant. And the final in the middle right quadrant. I would performa a k-means clustering analysis at this point.

Any surprises?

These securities seem to have a high degree of multicollinearity with a few outliers.

Note: Supplemental graph & code for Task 5 is below.

In [11]:
```
%%R
# Task 5 Supplemental Graph
industry=read.csv('industry.csv',sep=',', header=TRUE)
```

```
names(industry)[1] <- "Ticker"
pc.1 <- as.data.frame(pc.1)
pc.2 <- as.data.frame(pc.2)
`row.names<-`(industry[-1], industry$Ticker)
```

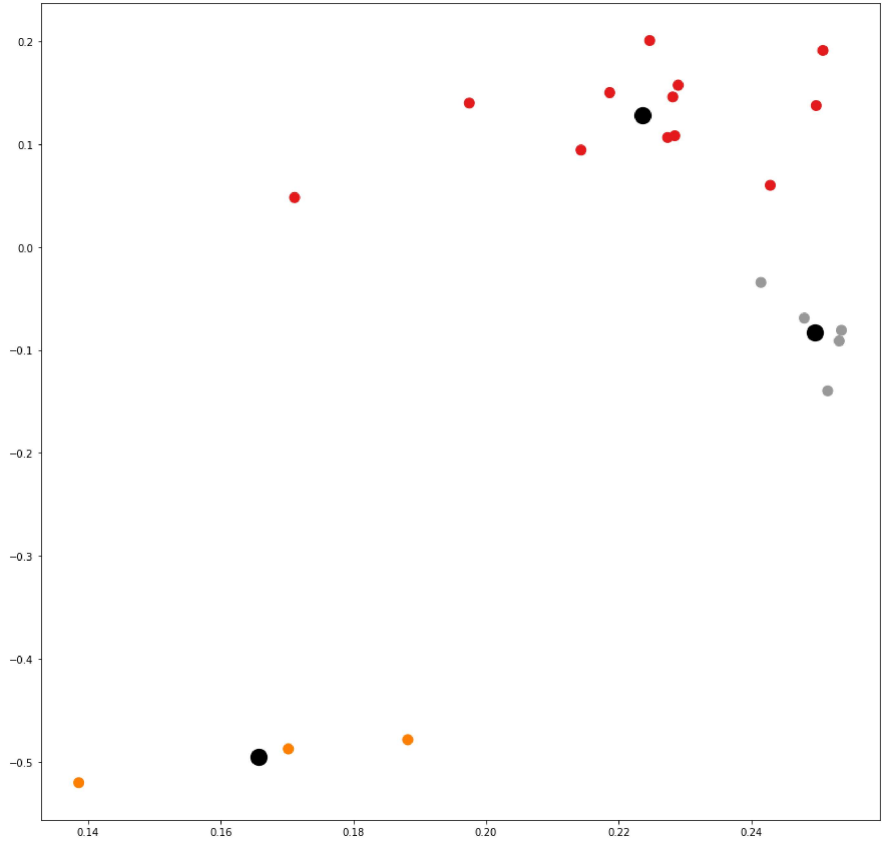
	Name	Industry	Industry_Code	color_code
AA	Alcoa Aluminum	Industrial - Metals	7	gold
BAC	Bank of America	Banking	2	orange
BHI	Baker Hughes Incorprated	Oil Field Services	3	green
CVX	Chevron	Oil Refining	4	blue
DD	Dupont	Industrial - Chemical	5	purple
DOW	Dow Chemical	Industrial - Chemical	5	purple
DPS	DrPepper Snapple	Soft Drinks	6	black
GS	Goldman Sachs	Banking	2	orange
HAL	Halliburton	Oil Field Services	3	green
HES	Hess Energy	Oil Refining	4	blue
HON	Honeywell International	Manufacturing	1	red
HUN	Huntsman Corporation	Industrial - Chemical	5	purple
JPM	JPMorgan Chase	Banking	2	orange
KO	The Coca-Cola Company	Soft Drinks	6	black
MMM	3M Company	Manufacturing	1	red
MPC	Marathon Petroleum Corp	Oil Refining	4	blue
PEP	Pepsi Company	Soft Drinks	6	black
SLB	Schlumberger	Oil Field Services	3	green
WFC	Wells Fargo	Banking	2	orange
XOM	Exxon-Mobile	Oil Refining	4	blue
VV	Vanguard Large Cap Index	Market Index	9	deeppink

In [12]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
industry= %Rget industry
pc1= %Rget pc.1
pc2= %Rget pc.2
industry=industry.set_index('Ticker')
industry=pd.merge(industry, pc1, left_index=True, right_index=True)
industry=pd.merge(industry, pc2, left_index=True, right_index=True)

fig, ax = plt.subplots(figsize=(15,15))
ax.scatter(industry['pc.1'], industry['pc.2'], facecolors='none', edgecolors='white', color=industry['color_code'])
plt.xlabel("pc.1")
plt.ylabel("pc.2")
for x,y,l,c in zip(industry['pc.1'], industry['pc.2'], industry.index.to_list(), industry['color_code']):
    labels="{}".format(l)
    plt.annotate(labels, (x,y), textcoords="offset points", xytext=(0,0), ha='center', color=c)
```

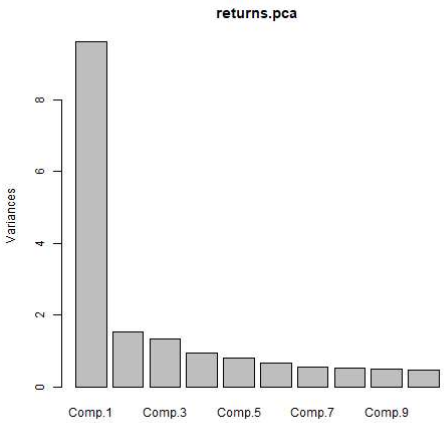
In [13]:

```
### This shows the clusters and their divisions
from sklearn.cluster import KMeans
from scipy.spatial import Voronoi, voronoi_plot_2d
import numpy as np
from numpy import array
from matplotlib.pyplot import figure
X = np.array(list(zip(industry['pc.1'].to_list(), industry['pc.2'].to_list())))
model = KMeans(n_clusters=3)
model.fit(X)
figure(figsize=(15,15))
y_kmeans=model.predict(X)
for centroid in model.cluster_centers_:
    plt.scatter(centroid[0], centroid[1],marker="o", color="k", linewidths=5, s=150)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=100, cmap='Set1')

plt.show()
```

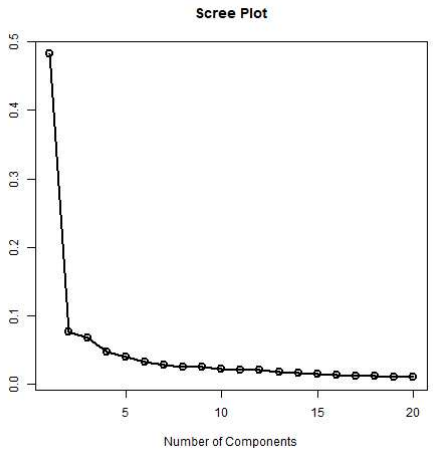


Task 6

In [14]:
```
%%R
plot(returns.pca)
```



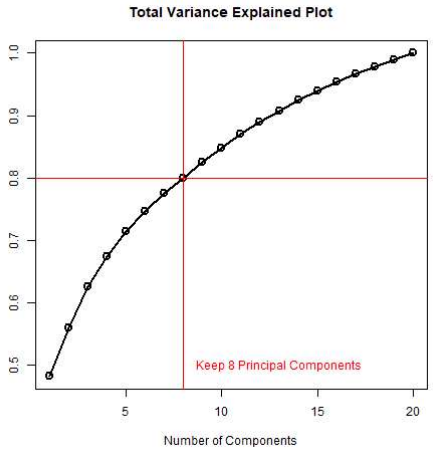
In [15]:
```
%%R
scree.values <- (returns.pca$sdev^2)/sum(returns.pca$sdev^2);

plot(scree.values,xlab='Number of Components',ylab='',type='l',lwd=2)
points(scree.values,lwd=2,cex=1.5)
title('Scree Plot')
```



In [16]:

```
%%R
variance.values <- cumsum(returns.pca$sdev^2)/sum(returns.pca$sdev^2);
print(variance.values)
plot(variance.values,xlab='Number of Components',ylab='',type='l',lwd=2)
points(variance.values,lwd=2,cex=1.5)
abline(h=0.8,lwd=1.5,col='red')
abline(v=8,lwd=1.5,col='red')
text(13,0.5,'Keep 8 Principal Components',col='red')
title('Total Variance Explained Plot')
```

```
   Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7    Comp.8
0.4818225 0.5587490 0.6261209 0.6735048 0.7132802 0.7466005 0.7745216 0.8001590
   Comp.9   Comp.10   Comp.11   Comp.12   Comp.13   Comp.14   Comp.15   Comp.16
0.8250011 0.8482177 0.8698897 0.8902775 0.9078318 0.9245871 0.9394356 0.9532500
  Comp.17   Comp.18   Comp.19   Comp.20
0.9662180 0.9783209 0.9894138 1.0000000
```



Task 6 Answers:

How many principal components do you think that we should keep? Why? (Hint: What decision rules should we use to determine the number of principal components to keep?)

I would say that we should keep the minimum number of components that meet our cutoff for percentage of explained variance, which is .8 and corresponds to 8 components in this situation.

Later in the assignment we will use the first eight principal components. Why eight?

It meets the 80% explained variance cutoff.

Task 7

In [17]:

```
%%R
# Create the data frame of PCA predictor variables;
return.scores <- as.data.frame(returns.pca$scores);
return.scores$VV <- returns.df$VV;
return.scores$u <- runif(n=dim(return.scores)[1],min=0,max=1);
head(return.scores)
```

```
# Split the data set into train and test data sets;
train.scores <- subset(return.scores,u<0.70);
test.scores <- subset(return.scores,u>=0.70);
dim(train.scores)
dim(test.scores)
dim(train.scores)+dim(test.scores)
dim(return.scores)

# Fit a linear regression model using the first 8 principal components;
pca1.lm <- lm(VV ~ Comp.1+Comp.2+Comp.3+Comp.4+Comp.5+Comp.6+Comp.7+Comp.8, data=train.scores);
summary(pca1.lm)

# Compute the Mean Absolute Error on the training sample;
pca1.mae.train <- mean(abs(train.scores$VV-pca1.lm$fitted.values));
vif(pca1.lm)

# Score the model out-of-sample and compute MAE;
pca1.test <- predict(pca1.lm,newdata=test.scores);
pca1.mae.test <- mean(abs(test.scores$VV-pca1.test));
```

Task 7 Answers:

The predictor variables for our predictive model will be the PCA scores. (What are the scores?)

The score is the distance from the orgin of the axis for the given observation.

The VIF values associated with every predictor variable in any principal components regression model should all be one. Why?

This finds collinearality and if not done can lead to overweighting the model towards two values that are very similar.

Task 8

In [18]:
```
%%R
# Let's compare the PCA regression model with a 'raw' regression model;
# Create a train/test split of the returns data set to match the scores data set;
returns.df$u <- return.scores$u;
train.returns <- subset(returns.df,u<0.70);
test.returns <- subset(returns.df,u>=0.70);
dim(train.returns)
dim(test.returns)
dim(train.returns)+dim(test.returns)
dim(returns.df)


# Fit model.1 on train data set and score on test data;
model.1 <- lm(VV ~ GS+DD+DOW+HON+HUN+JPM+KO+MMM+XOM, data=train.returns)
model1.mae.train <- mean(abs(train.returns$VV-model.1$fitted.values));
model1.test <- predict(model.1,newdata=test.returns);
model1.mae.test <- mean(abs(test.returns$VV-model1.test));


# Fit model.2 on train data set and score on test data;
model.2 <- lm(VV ~ AA+BAC+GS+JPM+WFC+BHI+CVX+DD+DOW+DPS+HAL+HES+HON+HUN+KO+MMM+MPC+PEP+SLB+XOM, data=train.returns)
model2.mae.train <- mean(abs(train.returns$VV-model.2$fitted.values));
model2.test <- predict(model.2,newdata=test.returns);
model2.mae.test <- mean(abs(test.returns$VV-model2.test));
```

In [19]:
```
%%R
specify_decimal <- function(x, k) trimws(format(round(x, k), nsmall=k))
print('model 1 mae test & train')
print(specify_decimal(model1.mae.test, 8))
print(specify_decimal(model1.mae.train,8))
```

```
[1] "model 1 mae test & train"
[1] "0.00219909"
[1] "0.00216549"
```

In [20]:
```
%%R
print('model 2 mae test & train')
print(specify_decimal(model2.mae.test, 8))
print(specify_decimal(model2.mae.train, 8))
```

```
[1] "model 2 mae test & train"
[1] "0.00218892"
[1] "0.00186935"
```

Task 8 Answers:

Is our principal components regression model a 'better' model or the 'best' model?

I would say that model1 is better as it uses only the principal components and it has a good mae. The fact that model2 has a lower mae could be luck and it may fail in a prod enviornment.

What does it mean for Model A to be better than Model B? How about in predictive modeling?

Model A is correctly weighted due to the pca analysis and is the only option of the two for predictive modeling.

Present the MAE values from models pca1.lm, model.1, and model.2 in a table so that they are easily compared and discussed in your paper. Discuss these results. Which model is the best model? Why?

The best model as stated prior is model 1 because it is not overweighted and will be less likely to fail if implemented in a prod enviornment.

The requiested table is below*

In [21]:
```
#model comparison
mod1_test_mae = %Rget model1.mae.test
mod1_train_mae = %Rget model1.mae.train
mod2_test_mae = %Rget model2.mae.test
mod2_train_mae = %Rget model2.mae.train
model_board=pd.DataFrame({
    'Train_Test': ['Train', 'Test'],
    'Model_1': [mod1_train_mae[0], mod1_test_mae[0]],
    'Model_2': [mod2_train_mae[0], mod2_test_mae[0]]
}).set_index('Train_Test')
model_board
```

Out[21]:

Train_Test	Model_1	Model_2
Train	0.002165	0.001869
Test	0.002199	0.002189

Task 9

In [22]:
```
%%R
full.lm <- lm(VV ~ ., data=train.scores);
summary(full.lm)
library(MASS)
backward.lm <- stepAIC(full.lm,direction=c('backward'))
summary(backward.lm)
backward.mae.train <- mean(abs(train.scores$VV-backward.lm$fitted.values));
vif(backward.lm)

backward.test <- predict(backward.lm,newdata=test.scores);
backward.mae.test <- mean(abs(test.scores$VV-backward.test));
```

```
Start:  AIC=-4338.48
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 + Comp.7 +
    Comp.8 + Comp.9 + Comp.10 + Comp.11 + Comp.12 + Comp.13 +
    Comp.14 + Comp.15 + Comp.16 + Comp.17 + Comp.18 + Comp.19 +
    Comp.20 + u

          Df Sum of Sq       RSS     AIC
- Comp.15  1 0.0000000 0.0021493 -4340.5
- u        1 0.0000001 0.0021494 -4340.5
- Comp.20  1 0.0000004 0.0021497 -4340.4
- Comp.12  1 0.0000010 0.0021502 -4340.3
- Comp.17  1 0.0000011 0.0021504 -4340.3
- Comp.7   1 0.0000046 0.0021539 -4339.7
- Comp.6   1 0.0000065 0.0021558 -4339.4
- Comp.16  1 0.0000085 0.0021578 -4339.0
- Comp.19  1 0.0000108 0.0021600 -4338.7
- Comp.18  1 0.0000118 0.0021611 -4338.5
<none>                 0.0021493 -4338.5
- Comp.5   1 0.0000157 0.0021649 -4337.8
- Comp.13  1 0.0000232 0.0021725 -4336.6
- Comp.8   1 0.0000267 0.0021760 -4336.0
- Comp.4   1 0.0000275 0.0021767 -4335.9
- Comp.14  1 0.0000320 0.0021813 -4335.1
- Comp.9   1 0.0000377 0.0021869 -4334.2
- Comp.10  1 0.0000385 0.0021878 -4334.0
- Comp.11  1 0.0000623 0.0022115 -4330.1
- Comp.3   1 0.0001300 0.0022793 -4319.1
- Comp.2   1 0.0001447 0.0022940 -4316.8
- Comp.1   1 0.0172714 0.0194206 -3539.2

Step:  AIC=-4340.48
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 + Comp.7 +
    Comp.8 + Comp.9 + Comp.10 + Comp.11 + Comp.12 + Comp.13 +
    Comp.14 + Comp.16 + Comp.17 + Comp.18 + Comp.19 + Comp.20 +
    u

          Df Sum of Sq       RSS     AIC
- u        1 0.0000001 0.0021494 -4342.5
- Comp.20  1 0.0000004 0.0021497 -4342.4
- Comp.12  1 0.0000010 0.0021503 -4342.3
- Comp.17  1 0.0000011 0.0021504 -4342.3
- Comp.7   1 0.0000046 0.0021539 -4341.7
- Comp.6   1 0.0000065 0.0021558 -4341.4
- Comp.16  1 0.0000085 0.0021578 -4341.0
- Comp.19  1 0.0000110 0.0021603 -4340.6
- Comp.18  1 0.0000118 0.0021611 -4340.5
<none>                 0.0021493 -4340.5
- Comp.5   1 0.0000157 0.0021650 -4339.8
- Comp.13  1 0.0000232 0.0021725 -4338.6
- Comp.8   1 0.0000267 0.0021760 -4338.0
- Comp.4   1 0.0000275 0.0021768 -4337.8
- Comp.14  1 0.0000320 0.0021813 -4337.1
- Comp.9   1 0.0000377 0.0021870 -4336.1
- Comp.10  1 0.0000385 0.0021878 -4336.0
- Comp.11  1 0.0000624 0.0022117 -4332.1
- Comp.3   1 0.0001300 0.0022793 -4321.1
- Comp.2   1 0.0001447 0.0022940 -4318.8
- Comp.1   1 0.0172736 0.0194229 -3541.2

Step:  AIC=-4342.46
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 + Comp.7 +
    Comp.8 + Comp.9 + Comp.10 + Comp.11 + Comp.12 + Comp.13 +
    Comp.14 + Comp.16 + Comp.17 + Comp.18 + Comp.19 + Comp.20

          Df Sum of Sq       RSS     AIC
- Comp.20  1 0.0000004 0.0021498 -4344.4
- Comp.12  1 0.0000010 0.0021504 -4344.3
```

```
- Comp.17  1 0.0000011 0.0021505 -4344.3
- Comp.7   1 0.0000046 0.0021540 -4343.7
- Comp.6   1 0.0000065 0.0021559 -4343.4
- Comp.16  1 0.0000086 0.0021580 -4343.0
- Comp.19  1 0.0000112 0.0021606 -4342.6
- Comp.18  1 0.0000117 0.0021611 -4342.5
<none>                0.0021494 -4342.5
- Comp.5   1 0.0000156 0.0021650 -4341.8
- Comp.13  1 0.0000231 0.0021725 -4340.6
- Comp.8   1 0.0000270 0.0021764 -4339.9
- Comp.4   1 0.0000274 0.0021769 -4339.8
- Comp.14  1 0.0000319 0.0021813 -4339.1
- Comp.9   1 0.0000377 0.0021871 -4338.1
- Comp.10  1 0.0000385 0.0021879 -4338.0
- Comp.11  1 0.0000631 0.0022125 -4333.9
- Comp.3   1 0.0001318 0.0022812 -4322.8
- Comp.2   1 0.0001447 0.0022941 -4320.7
- Comp.1   1 0.0172736 0.0194230 -3543.2

Step:  AIC=-4344.39
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 + Comp.7 +
    Comp.8 + Comp.9 + Comp.10 + Comp.11 + Comp.12 + Comp.13 +
    Comp.14 + Comp.16 + Comp.17 + Comp.18 + Comp.19

          Df Sum of Sq       RSS     AIC
- Comp.12  1 0.0000010 0.0021508 -4346.2
- Comp.17  1 0.0000011 0.0021509 -4346.2
- Comp.7   1 0.0000045 0.0021543 -4345.6
- Comp.6   1 0.0000066 0.0021564 -4345.3
- Comp.16  1 0.0000086 0.0021584 -4344.9
- Comp.19  1 0.0000112 0.0021610 -4344.5
- Comp.18  1 0.0000116 0.0021614 -4344.4
<none>                0.0021498 -4344.4
- Comp.5   1 0.0000155 0.0021653 -4343.8
- Comp.13  1 0.0000231 0.0021729 -4342.5
- Comp.8   1 0.0000269 0.0021767 -4341.9
- Comp.4   1 0.0000273 0.0021771 -4341.8
- Comp.14  1 0.0000321 0.0021819 -4341.0
- Comp.9   1 0.0000379 0.0021877 -4340.0
- Comp.10  1 0.0000385 0.0021883 -4339.9
- Comp.11  1 0.0000635 0.0022133 -4335.8
- Comp.3   1 0.0001315 0.0022813 -4324.8
- Comp.2   1 0.0001446 0.0022944 -4322.7
- Comp.1   1 0.0172807 0.0194305 -3545.1

Step:  AIC=-4346.22
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 + Comp.7 +
    Comp.8 + Comp.9 + Comp.10 + Comp.11 + Comp.13 + Comp.14 +
    Comp.16 + Comp.17 + Comp.18 + Comp.19

          Df Sum of Sq       RSS     AIC
- Comp.17  1 0.0000011 0.0021519 -4348.0
- Comp.7   1 0.0000043 0.0021552 -4347.5
- Comp.6   1 0.0000067 0.0021575 -4347.1
- Comp.16  1 0.0000087 0.0021595 -4346.8
- Comp.19  1 0.0000110 0.0021618 -4346.4
<none>                0.0021508 -4346.2
- Comp.18  1 0.0000119 0.0021627 -4346.2
- Comp.5   1 0.0000158 0.0021666 -4345.6
- Comp.13  1 0.0000234 0.0021742 -4344.3
- Comp.4   1 0.0000276 0.0021784 -4343.6
- Comp.8   1 0.0000279 0.0021787 -4343.5
- Comp.14  1 0.0000316 0.0021824 -4342.9
- Comp.9   1 0.0000375 0.0021883 -4341.9
- Comp.10  1 0.0000386 0.0021894 -4341.7
- Comp.11  1 0.0000638 0.0022146 -4337.6
- Comp.3   1 0.0001316 0.0022824 -4326.6
- Comp.2   1 0.0001449 0.0022957 -4324.5
- Comp.1   1 0.0173206 0.0194714 -3546.3

Step:  AIC=-4348.03
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 + Comp.7 +
    Comp.8 + Comp.9 + Comp.10 + Comp.11 + Comp.13 + Comp.14 +
    Comp.16 + Comp.18 + Comp.19

          Df Sum of Sq       RSS     AIC
- Comp.7   1 0.0000042 0.0021561 -4349.3
- Comp.6   1 0.0000067 0.0021586 -4348.9
- Comp.16  1 0.0000088 0.0021608 -4348.5
- Comp.19  1 0.0000108 0.0021628 -4348.2
<none>                0.0021519 -4348.0
- Comp.18  1 0.0000119 0.0021639 -4348.0
- Comp.5   1 0.0000159 0.0021678 -4347.4
- Comp.13  1 0.0000230 0.0021749 -4346.2
- Comp.4   1 0.0000273 0.0021793 -4345.4
- Comp.8   1 0.0000278 0.0021797 -4345.4
- Comp.14  1 0.0000313 0.0021833 -4344.8
- Comp.9   1 0.0000372 0.0021892 -4343.8
- Comp.10  1 0.0000387 0.0021906 -4343.5
- Comp.11  1 0.0000640 0.0022159 -4339.4
- Comp.3   1 0.0001339 0.0022859 -4328.1
- Comp.2   1 0.0001463 0.0022983 -4326.1
- Comp.1   1 0.0173460 0.0194979 -3547.8

Step:  AIC=-4349.32
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 + Comp.8 +
    Comp.9 + Comp.10 + Comp.11 + Comp.13 + Comp.14 + Comp.16 +
    Comp.18 + Comp.19

          Df Sum of Sq       RSS     AIC
- Comp.6   1 0.0000066 0.0021628 -4350.2
- Comp.16  1 0.0000080 0.0021641 -4350.0
- Comp.19  1 0.0000106 0.0021667 -4349.5
- Comp.18  1 0.0000117 0.0021678 -4349.4
<none>                0.0021561 -4349.3
- Comp.5   1 0.0000159 0.0021721 -4348.6
- Comp.13  1 0.0000224 0.0021785 -4347.6
- Comp.4   1 0.0000273 0.0021834 -4346.8
```

```
- Comp.8   1 0.0000277 0.0021838 -4346.7
- Comp.14  1 0.0000306 0.0021867 -4346.2
- Comp.9   1 0.0000364 0.0021925 -4345.2
- Comp.10  1 0.0000387 0.0021948 -4344.9
- Comp.11  1 0.0000660 0.0022221 -4340.3
- Comp.3   1 0.0001354 0.0022915 -4329.2
- Comp.2   1 0.0001473 0.0023034 -4327.3
- Comp.1   1 0.0173964 0.0195525 -3548.8

Step:  AIC=-4350.2
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.8 + Comp.9 +
    Comp.10 + Comp.11 + Comp.13 + Comp.14 + Comp.16 + Comp.18 +
    Comp.19

          Df Sum of Sq       RSS     AIC
- Comp.16  1 0.0000073 0.0021701 -4351.0
- Comp.19  1 0.0000106 0.0021733 -4350.4
<none>                0.0021628 -4350.2
- Comp.18  1 0.0000119 0.0021747 -4350.2
- Comp.5   1 0.0000157 0.0021784 -4349.6
- Comp.13  1 0.0000217 0.0021845 -4348.6
- Comp.8   1 0.0000259 0.0021887 -4347.9
- Comp.4   1 0.0000274 0.0021901 -4347.6
- Comp.14  1 0.0000296 0.0021923 -4347.3
- Comp.9   1 0.0000359 0.0021987 -4346.2
- Comp.10  1 0.0000369 0.0021997 -4346.0
- Comp.11  1 0.0000663 0.0022290 -4341.2
- Comp.3   1 0.0001375 0.0023002 -4329.8
- Comp.2   1 0.0001473 0.0023101 -4328.2
- Comp.1   1 0.0174395 0.0196023 -3549.9

Step:  AIC=-4350.97
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.8 + Comp.9 +
    Comp.10 + Comp.11 + Comp.13 + Comp.14 + Comp.18 + Comp.19

          Df Sum of Sq       RSS     AIC
- Comp.19  1 0.0000108 0.0021809 -4351.2
- Comp.18  1 0.0000111 0.0021812 -4351.1
<none>                0.0021701 -4351.0
- Comp.5   1 0.0000164 0.0021865 -4350.2
- Comp.13  1 0.0000218 0.0021918 -4349.3
- Comp.8   1 0.0000250 0.0021951 -4348.8
- Comp.4   1 0.0000281 0.0021981 -4348.3
- Comp.14  1 0.0000293 0.0021994 -4348.1
- Comp.10  1 0.0000365 0.0022066 -4346.9
- Comp.9   1 0.0000365 0.0022066 -4346.9
- Comp.11  1 0.0000656 0.0022356 -4342.1
- Comp.3   1 0.0001350 0.0023051 -4331.0
- Comp.2   1 0.0001473 0.0023174 -4329.1
- Comp.1   1 0.0175165 0.0196866 -3550.3

Step:  AIC=-4351.17
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.8 + Comp.9 +
    Comp.10 + Comp.11 + Comp.13 + Comp.14 + Comp.18

          Df Sum of Sq       RSS     AIC
- Comp.18  1 0.0000111 0.0021920 -4351.3
<none>                0.0021809 -4351.2
- Comp.5   1 0.0000176 0.0021984 -4350.2
- Comp.13  1 0.0000201 0.0022010 -4349.8
- Comp.8   1 0.0000253 0.0022061 -4349.0
- Comp.4   1 0.0000272 0.0022081 -4348.7
- Comp.14  1 0.0000293 0.0022102 -4348.3
- Comp.10  1 0.0000354 0.0022162 -4347.3
- Comp.9   1 0.0000354 0.0022163 -4347.3
- Comp.11  1 0.0000659 0.0022468 -4342.3
- Comp.3   1 0.0001375 0.0023184 -4330.9
- Comp.2   1 0.0001514 0.0023323 -4328.7
- Comp.1   1 0.0175297 0.0197106 -3551.8

Step:  AIC=-4351.31
VV ~ Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.8 + Comp.9 +
    Comp.10 + Comp.11 + Comp.13 + Comp.14

          Df Sum of Sq       RSS     AIC
<none>                0.0021920 -4351.3
- Comp.5   1 0.0000174 0.0022094 -4350.4
- Comp.13  1 0.0000202 0.0022122 -4350.0
- Comp.8   1 0.0000255 0.0022175 -4349.1
- Comp.4   1 0.0000291 0.0022211 -4348.5
- Comp.14  1 0.0000296 0.0022216 -4348.4
- Comp.10  1 0.0000338 0.0022258 -4347.7
- Comp.9   1 0.0000357 0.0022277 -4347.4
- Comp.11  1 0.0000649 0.0022569 -4342.7
- Comp.3   1 0.0001342 0.0023262 -4331.7
- Comp.2   1 0.0001544 0.0023464 -4328.5
- Comp.1   1 0.0175318 0.0197238 -3553.6
```

Task 9 Answers

Does our unsupervised decision rule translate to producing the best predictive model?

In this case I would say that we have a the first itertation of a production level model. However, models can always be improved utalizing differing methods.

More directly, is the eight principal components model the best predictive model?

In this case it is. However different models will utalize different numbers of principal components.

How else might we select the 'best' number of principal components to keep in our predictive modeling problem?

You want to keep shooting for that .8 benchmark of explained variance.

Why don't we consider a supervised approach of using variable selection to select the number of principal components to keep, and which principal components to keep?

That can be done, hoever with this high degree of collinearity an accurate model would be hard to produce using current methods.

How many principal components does the variable selection approach suggest that we keep? Which ones?

It suggests we keep 8 Principal components. They are 16, 6, 11, 9, 10, 8, 2, & 3

How does this differ from our previous discussions about the number of principal components to keep?

The previous process also suggested 8 components but it was based on reaching 80% explained variance.

Create a new table where you add these new MAE results to the previous MAE results.

Table is below.

How does our backward.lm model compare to all of the other models that we have fit in this assignment? Which model is best and why?

The backward.lm is the best model for dimensionality reduction as it eliminates colinear components and hasa the best mae score.

In [23]:
```
backward_mae_test = %Rget backward.mae.test
backward_mae_train = %Rget backward.mae.train
model_board['backward.lm']=[backward_mae_train[0], backward_mae_test[0]]
model_board
```

Out[23]:

Train_Test	Model_1	Model_2	backward.lm
Train	0.002165	0.001869	0.001899
Test	0.002199	0.002189	0.002140

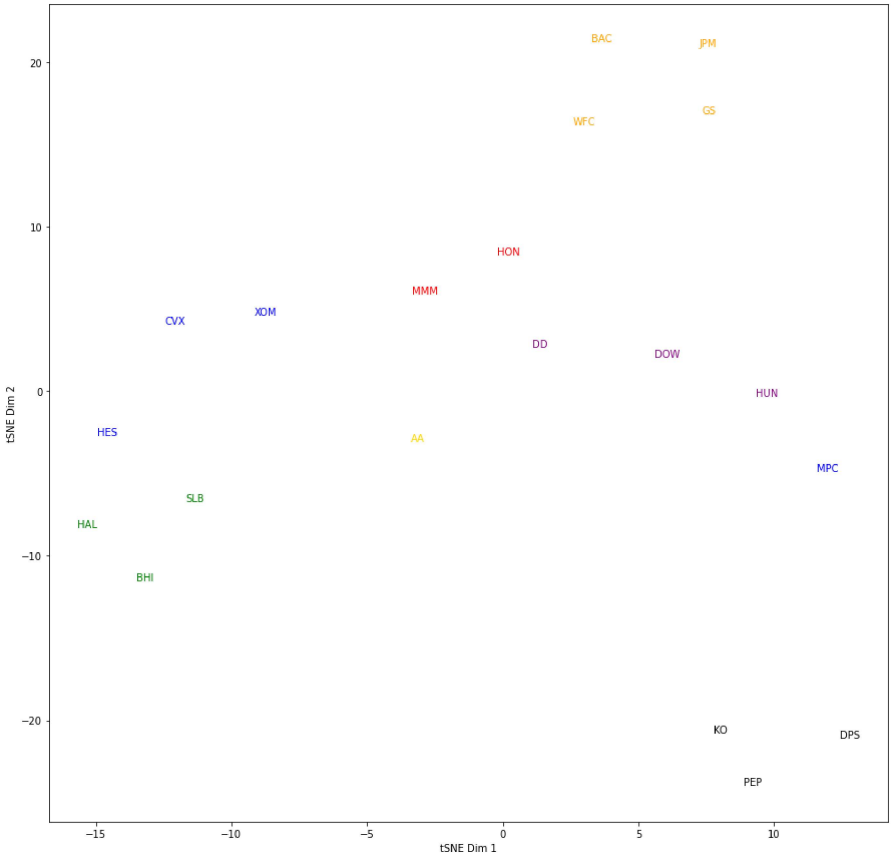
Task 10

In [24]:
```
%%R
# Let's go back to the beginning with this data set, and use the t-SNE approach to reduce and visualize this data.
# Conduct a t-SNE analysis on the Stock Portfolio data.
library(Rtsne)
tsne.cor <- returns.cor[-21,-21]
tsne <- Rtsne(tsne.cor, dims = 2, perplexity=6, verbose =TRUE, max_iter = 8000, learning = 50, n_components=8)
tnse_fin <- tsne$Y
```

```
Performing PCA
Read the 20 x 20 data matrix successfully!
OpenMP is working. 1 threads.
Using no_dims = 2, perplexity = 6.000000, and theta = 0.500000
Computing input similarities...
Building tree...
Done in 0.00 seconds (sparsity = 0.945000)!
Learning embedding...
Iteration 50: error is 49.913877 (50 iterations in 0.00 seconds)
Iteration 100: error is 58.445057 (50 iterations in 0.00 seconds)
Iteration 150: error is 52.603346 (50 iterations in 0.00 seconds)
Iteration 200: error is 54.457247 (50 iterations in 0.00 seconds)
Iteration 250: error is 55.282040 (50 iterations in 0.00 seconds)
Iteration 300: error is 0.957341 (50 iterations in 0.00 seconds)
Iteration 350: error is 0.566883 (50 iterations in 0.00 seconds)
Iteration 400: error is 0.316644 (50 iterations in 0.00 seconds)
Iteration 450: error is 0.105555 (50 iterations in 0.00 seconds)
Iteration 500: error is 0.106823 (50 iterations in 0.00 seconds)
Iteration 550: error is 0.104495 (50 iterations in 0.00 seconds)
Iteration 600: error is 0.113991 (50 iterations in 0.00 seconds)
Iteration 650: error is 0.112041 (50 iterations in 0.00 seconds)
Iteration 700: error is 0.110040 (50 iterations in 0.00 seconds)
Iteration 750: error is 0.109353 (50 iterations in 0.00 seconds)
Iteration 800: error is 0.102703 (50 iterations in 0.00 seconds)
Iteration 850: error is 0.069872 (50 iterations in 0.00 seconds)
Iteration 900: error is 0.104790 (50 iterations in 0.00 seconds)
Iteration 950: error is 0.103752 (50 iterations in 0.00 seconds)
Iteration 1000: error is 0.106659 (50 iterations in 0.00 seconds)
Iteration 1050: error is 0.104967 (50 iterations in 0.00 seconds)
Iteration 1100: error is 0.103822 (50 iterations in 0.00 seconds)
Iteration 1150: error is 0.101972 (50 iterations in 0.00 seconds)
Iteration 1200: error is 0.102327 (50 iterations in 0.00 seconds)
Iteration 1250: error is 0.102685 (50 iterations in 0.00 seconds)
Iteration 1300: error is 0.104557 (50 iterations in 0.00 seconds)
Iteration 1350: error is 0.103178 (50 iterations in 0.00 seconds)
Iteration 1400: error is 0.102762 (50 iterations in 0.00 seconds)
Iteration 1450: error is 0.103869 (50 iterations in 0.00 seconds)
Iteration 1500: error is 0.094682 (50 iterations in 0.00 seconds)
Iteration 1550: error is 0.099907 (50 iterations in 0.00 seconds)
Iteration 1600: error is 0.102531 (50 iterations in 0.00 seconds)
Iteration 1650: error is 0.102687 (50 iterations in 0.00 seconds)
Iteration 1700: error is 0.096922 (50 iterations in 0.00 seconds)
Iteration 1750: error is 0.098207 (50 iterations in 0.00 seconds)
Iteration 1800: error is 0.099598 (50 iterations in 0.00 seconds)
Iteration 1850: error is 0.102649 (50 iterations in 0.00 seconds)
Iteration 1900: error is 0.104662 (50 iterations in 0.00 seconds)
Iteration 1950: error is 0.104393 (50 iterations in 0.00 seconds)
Iteration 2000: error is 0.103097 (50 iterations in 0.00 seconds)
Iteration 2050: error is 0.101186 (50 iterations in 0.00 seconds)
Iteration 2100: error is 0.418069 (50 iterations in 0.00 seconds)
Iteration 2150: error is 0.099450 (50 iterations in 0.00 seconds)
Iteration 2200: error is 0.095993 (50 iterations in 0.00 seconds)
Iteration 2250: error is 0.097506 (50 iterations in 0.00 seconds)
Iteration 2300: error is 0.092182 (50 iterations in 0.00 seconds)
Iteration 2350: error is 0.099429 (50 iterations in 0.00 seconds)
Iteration 2400: error is 0.093727 (50 iterations in 0.00 seconds)
Iteration 2450: error is 0.095669 (50 iterations in 0.00 seconds)
Iteration 2500: error is 0.094412 (50 iterations in 0.00 seconds)
Iteration 2550: error is 0.087508 (50 iterations in 0.00 seconds)
Iteration 2600: error is 0.213760 (50 iterations in 0.00 seconds)
Iteration 2650: error is 1.503633 (50 iterations in 0.00 seconds)
Iteration 2700: error is 0.832632 (50 iterations in 0.00 seconds)
```

```
Iteration 2750: error is 0.475916 (50 iterations in 0.00 seconds)
Iteration 2800: error is 0.236902 (50 iterations in 0.00 seconds)
Iteration 2850: error is 0.203109 (50 iterations in 0.00 seconds)
Iteration 2900: error is 0.164727 (50 iterations in 0.00 seconds)
Iteration 2950: error is 0.146802 (50 iterations in 0.00 seconds)
Iteration 3000: error is 0.114745 (50 iterations in 0.00 seconds)
Iteration 3050: error is 0.112610 (50 iterations in 0.00 seconds)
Iteration 3100: error is 0.109363 (50 iterations in 0.00 seconds)
Iteration 3150: error is 0.113525 (50 iterations in 0.00 seconds)
Iteration 3200: error is 0.102344 (50 iterations in 0.00 seconds)
Iteration 3250: error is 0.102008 (50 iterations in 0.00 seconds)
Iteration 3300: error is 0.110683 (50 iterations in 0.00 seconds)
Iteration 3350: error is 0.112820 (50 iterations in 0.00 seconds)
Iteration 3400: error is 0.102522 (50 iterations in 0.00 seconds)
Iteration 3450: error is 0.097619 (50 iterations in 0.00 seconds)
Iteration 3500: error is 0.085036 (50 iterations in 0.00 seconds)
Iteration 3550: error is 0.123314 (50 iterations in 0.00 seconds)
Iteration 3600: error is 0.092881 (50 iterations in 0.00 seconds)
Iteration 3650: error is 0.097055 (50 iterations in 0.00 seconds)
Iteration 3700: error is 0.094270 (50 iterations in 0.00 seconds)
Iteration 3750: error is 0.099151 (50 iterations in 0.00 seconds)
Iteration 3800: error is 0.092467 (50 iterations in 0.00 seconds)
Iteration 3850: error is 0.090913 (50 iterations in 0.00 seconds)
Iteration 3900: error is 0.089688 (50 iterations in 0.00 seconds)
Iteration 3950: error is 0.072340 (50 iterations in 0.00 seconds)
Iteration 4000: error is 0.075117 (50 iterations in 0.00 seconds)
Iteration 4050: error is 0.072499 (50 iterations in 0.00 seconds)
Iteration 4100: error is 0.077130 (50 iterations in 0.00 seconds)
Iteration 4150: error is 0.072790 (50 iterations in 0.00 seconds)
Iteration 4200: error is 0.076540 (50 iterations in 0.00 seconds)
Iteration 4250: error is 0.078979 (50 iterations in 0.00 seconds)
Iteration 4300: error is 0.079280 (50 iterations in 0.00 seconds)
Iteration 4350: error is 0.077649 (50 iterations in 0.00 seconds)
Iteration 4400: error is 0.074922 (50 iterations in 0.00 seconds)
Iteration 4450: error is 0.077584 (50 iterations in 0.00 seconds)
Iteration 4500: error is 0.077835 (50 iterations in 0.00 seconds)
Iteration 4550: error is 0.082992 (50 iterations in 0.00 seconds)
Iteration 4600: error is 0.069315 (50 iterations in 0.00 seconds)
Iteration 4650: error is 0.078071 (50 iterations in 0.00 seconds)
Iteration 4700: error is 0.079208 (50 iterations in 0.00 seconds)
Iteration 4750: error is 0.082757 (50 iterations in 0.00 seconds)
Iteration 4800: error is 0.069013 (50 iterations in 0.00 seconds)
Iteration 4850: error is 0.084763 (50 iterations in 0.00 seconds)
Iteration 4900: error is 0.068206 (50 iterations in 0.00 seconds)
Iteration 4950: error is 0.046319 (50 iterations in 0.00 seconds)
Iteration 5000: error is 0.074590 (50 iterations in 0.00 seconds)
Iteration 5050: error is 0.069546 (50 iterations in 0.00 seconds)
Iteration 5100: error is 0.078571 (50 iterations in 0.00 seconds)
Iteration 5150: error is 0.075238 (50 iterations in 0.00 seconds)
Iteration 5200: error is 0.084787 (50 iterations in 0.00 seconds)
Iteration 5250: error is 0.119435 (50 iterations in 0.00 seconds)
Iteration 5300: error is 0.080592 (50 iterations in 0.00 seconds)
Iteration 5350: error is 0.079422 (50 iterations in 0.00 seconds)
Iteration 5400: error is 0.082877 (50 iterations in 0.00 seconds)
Iteration 5450: error is 0.073487 (50 iterations in 0.00 seconds)
Iteration 5500: error is 0.107865 (50 iterations in 0.00 seconds)
Iteration 5550: error is 0.083006 (50 iterations in 0.00 seconds)
Iteration 5600: error is 0.092667 (50 iterations in 0.00 seconds)
Iteration 5650: error is 0.080451 (50 iterations in 0.00 seconds)
Iteration 5700: error is 0.100606 (50 iterations in 0.00 seconds)
Iteration 5750: error is 0.081958 (50 iterations in 0.00 seconds)
Iteration 5800: error is 0.108012 (50 iterations in 0.00 seconds)
Iteration 5850: error is 0.083559 (50 iterations in 0.00 seconds)
Iteration 5900: error is 0.084692 (50 iterations in 0.00 seconds)
Iteration 5950: error is 0.084791 (50 iterations in 0.00 seconds)
Iteration 6000: error is 0.084854 (50 iterations in 0.00 seconds)
Iteration 6050: error is 0.091578 (50 iterations in 0.00 seconds)
Iteration 6100: error is 0.115456 (50 iterations in 0.00 seconds)
Iteration 6150: error is 0.140617 (50 iterations in 0.00 seconds)
Iteration 6200: error is 0.079791 (50 iterations in 0.00 seconds)
Iteration 6250: error is 0.081101 (50 iterations in 0.00 seconds)
Iteration 6300: error is 0.079201 (50 iterations in 0.00 seconds)
Iteration 6350: error is 0.079851 (50 iterations in 0.00 seconds)
Iteration 6400: error is 0.078173 (50 iterations in 0.00 seconds)
Iteration 6450: error is 0.074968 (50 iterations in 0.00 seconds)
Iteration 6500: error is 0.072843 (50 iterations in 0.00 seconds)
Iteration 6550: error is 0.071865 (50 iterations in 0.00 seconds)
Iteration 6600: error is 0.076938 (50 iterations in 0.00 seconds)
Iteration 6650: error is 0.077347 (50 iterations in 0.00 seconds)
Iteration 6700: error is 0.327339 (50 iterations in 0.00 seconds)
Iteration 6750: error is 0.352559 (50 iterations in 0.00 seconds)
Iteration 6800: error is 0.071215 (50 iterations in 0.00 seconds)
Iteration 6850: error is 0.076026 (50 iterations in 0.00 seconds)
Iteration 6900: error is 0.075052 (50 iterations in 0.00 seconds)
Iteration 6950: error is 0.074953 (50 iterations in 0.00 seconds)
Iteration 7000: error is 0.072530 (50 iterations in 0.00 seconds)
Iteration 7050: error is 0.074043 (50 iterations in 0.00 seconds)
Iteration 7100: error is 0.074178 (50 iterations in 0.00 seconds)
Iteration 7150: error is 0.074295 (50 iterations in 0.00 seconds)
Iteration 7200: error is 0.073301 (50 iterations in 0.00 seconds)
Iteration 7250: error is 0.066125 (50 iterations in 0.00 seconds)
Iteration 7300: error is 0.068788 (50 iterations in 0.00 seconds)
Iteration 7350: error is 0.071436 (50 iterations in 0.00 seconds)
Iteration 7400: error is 0.068047 (50 iterations in 0.00 seconds)
Iteration 7450: error is 0.069083 (50 iterations in 0.00 seconds)
Iteration 7500: error is 0.083876 (50 iterations in 0.00 seconds)
Iteration 7550: error is 0.075231 (50 iterations in 0.00 seconds)
Iteration 7600: error is 0.076269 (50 iterations in 0.00 seconds)
Iteration 7650: error is 0.081962 (50 iterations in 0.00 seconds)
Iteration 7700: error is 0.078448 (50 iterations in 0.00 seconds)
Iteration 7750: error is 0.071012 (50 iterations in 0.00 seconds)
Iteration 7800: error is 0.076294 (50 iterations in 0.00 seconds)
Iteration 7850: error is 0.090645 (50 iterations in 0.00 seconds)
Iteration 7900: error is 0.080905 (50 iterations in 0.00 seconds)
Iteration 7950: error is 0.072982 (50 iterations in 0.00 seconds)
Iteration 8000: error is 0.074126 (50 iterations in 0.00 seconds)
Fitting performed in 0.50 seconds.
```

In [25]:
```
industry= %Rget industry
industry=industry.set_index('Ticker')
industry=pd.merge(industry, pc1, left_index=True, right_index=True)
industry=pd.merge(industry, pc2, left_index=True, right_index=True)
rtnse = %Rget tnse_fin
industry['tnse_x']=rtnse[:,0]
industry['tnse_y']=rtnse[:,1]
fig, ax = plt.subplots(figsize=(15,15))
plt.xlabel("tSNE Dim 1")
plt.ylabel("tSNE Dim 2")
ax.scatter(industry['tnse_x'], industry['tnse_y'], facecolors='none', edgecolors='white', color=industry['color_code'])
for x,y,l,c in zip(industry['tnse_x'], industry['tnse_y'], industry.index.to_list(), industry['color_code']):
    labels="{}".format(l)
    plt.annotate(labels, (x,y), textcoords="offset points", xytext=(0,0), ha='center', color=c)
```



Task 10 Answers

Interpret the results, then compare and contrast the t-SNE results with that of the Principal Components analysis you've conducted previously.

The t-SNE analysis correctly identified the clusters utalizing raw input data which is pretty good. This clustering could largely reduce the number of components needed when conducting future analysis.

Task 11 Answers

Please write a reflection on your modeling experiences during this first assignment.

I enjoyed this assignment quite a bit. It was a very comprehensive dive into dimensionality reduction and the various benefits of varying methods. I also enjoy working with time series data as it fits into my telematics background.

From my point of view this data was a very simplified version of what we normally deal with and all of the modeling methods worked quite well. The PCA analysis easily identified 8 principal components as did the tSNE analysis, them lining up was suprising.

Going forward with this type of data I would focus on clustering analysis for building models as I demonstrated above it will yield the best results.

I also enjoyed getting to know R a little better as I am not as familiar with it as I am with Python.