A Practical Guide to Dimensionality Reduction

Vishal Patel

October 8, 2016

WhoAmI

- Vishal Patel
- Data Science Consultant
- Founder of DERIVE, LLC
  - Fully Automated Advanced Analytics products
- Have been mining data for about a decade and a half
- MS in Computer Science, and MS in Decision Sciences (emphasis on Statistics)
- Richmond, VA

www.derive.io

Agenda

Dimensionality Reduction

What	When	Why	How	Q&A
5-7 minutes			25 minutes	5-10 minutes

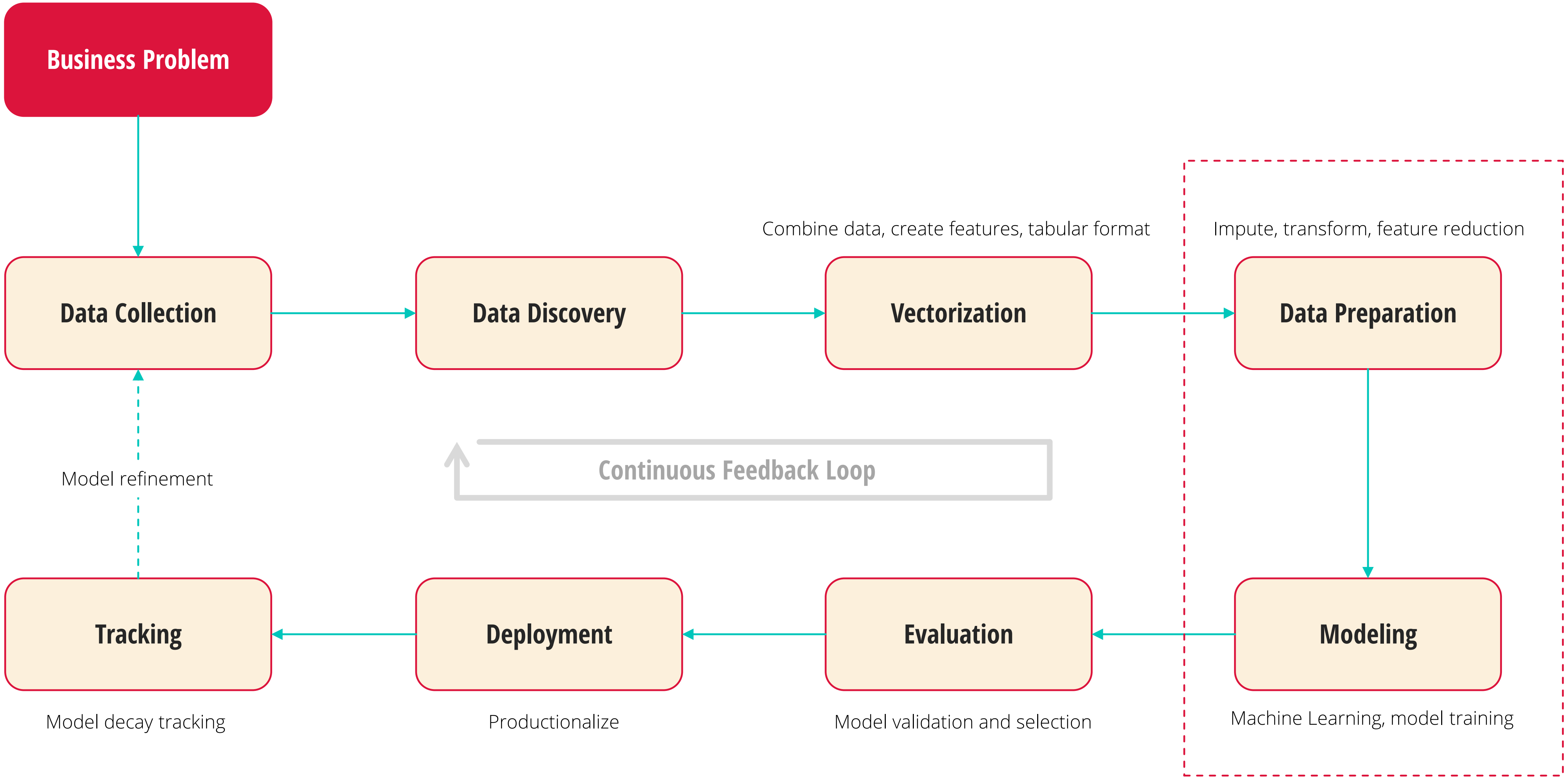
What

- Dimensionality Reduction: The process of selecting a subset of features for use in model construction
  - Pre-processing of data in machine learning
  - Can be useful for both supervised and unsupervised learning problems, but we will focus on the former

“Art is the elimination of the unnecessary.“

Pablo Picasso

When (Process Flow)



Dataset

Springleaf
Lending made personal

Completed • $100,000 • 2,226 teams

Springleaf Marketing Response

Fri 14 Aug 2015 – Mon 19 Oct 2015 (11 months ago)

Anonymized features → Predict which customers will respond to a DM offer

Train	Test	
1,933	1,933	features
145,231	145,232	records

Challenge: To construct new meta-variables and employ feature-selection methods to approach this dauntingly wide dataset.

https://www.kaggle.com/c/springleaf-marketing-response/data

Why Not This

```python
from sklearn.linear_model import LogisticRegression
import pandas as pd

train = pd.read_csv(r'C:\train.csv')

y_train = train['target']
train.pop('target')

modelFit = LogisticRegression().fit(train, y_train)
```

The kitchen sink approach

Because...

- True dimensionality <<< Observed dimensionality
  - The abundance of redundant and irrelevant features
- Curse of dimensionality
  - With a fixed number of training samples, the predictive power reduces as the dimensionality increases. [Hughes phenomenon]
  - With d binary variables, the number of possible combinations is $O(2^d)$.
- Value of Analytics
  - Descriptive → Diagnostic → Predictive → Prescriptive

    Hindsight **Insight** **Foresight**
- Law of Parsimony [Occam's Razor]
  - Other things being equal, simpler explanations are generally better than complex ones.
- Overfitting
- Execution time (Algorithm and data)

How

Dimensionality Reduction Techniques

1. Percent missing values
2. Amount of variation
3. Pairwise correlation
4. Multicolinearity
5. Principal Component Analysis (PCA)
6. Cluster analysis
7. Correlation (with the target)
8. Forward selection
9. Backward elimination
10. Stepwise selection
11. LASSO
12. Tree-based selection

Information

Redundancy

Predictive Power

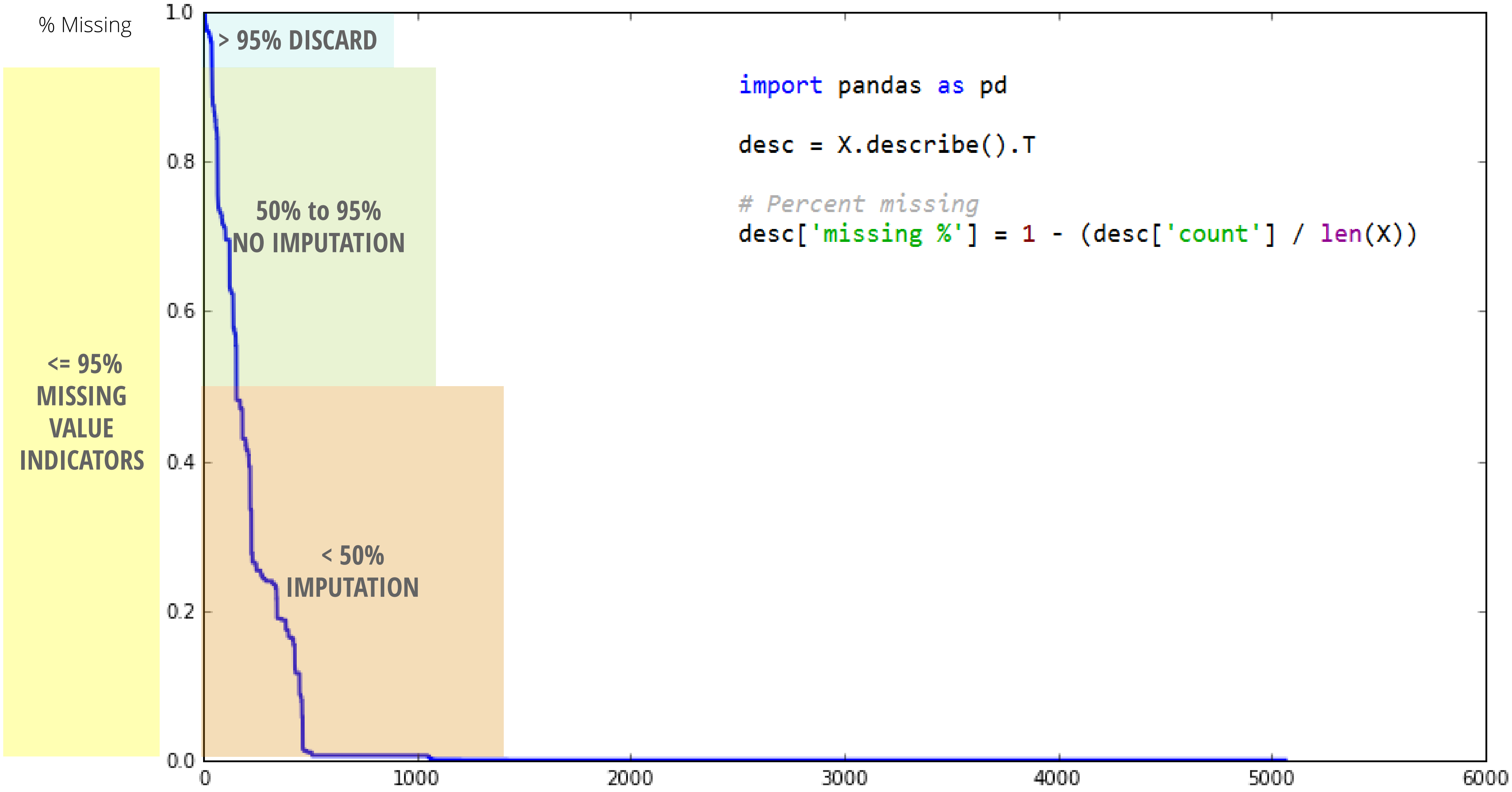
Greedy Selection

Embedded

1. Percent Missing Values

- Drop variables that have a very high % of missing values
  - # of records with missing values / # of total records
- Create binary indicators to denote missing (or non-missing) values
- Review or visualize variables with high % of missing values

1. Percent Missing Values

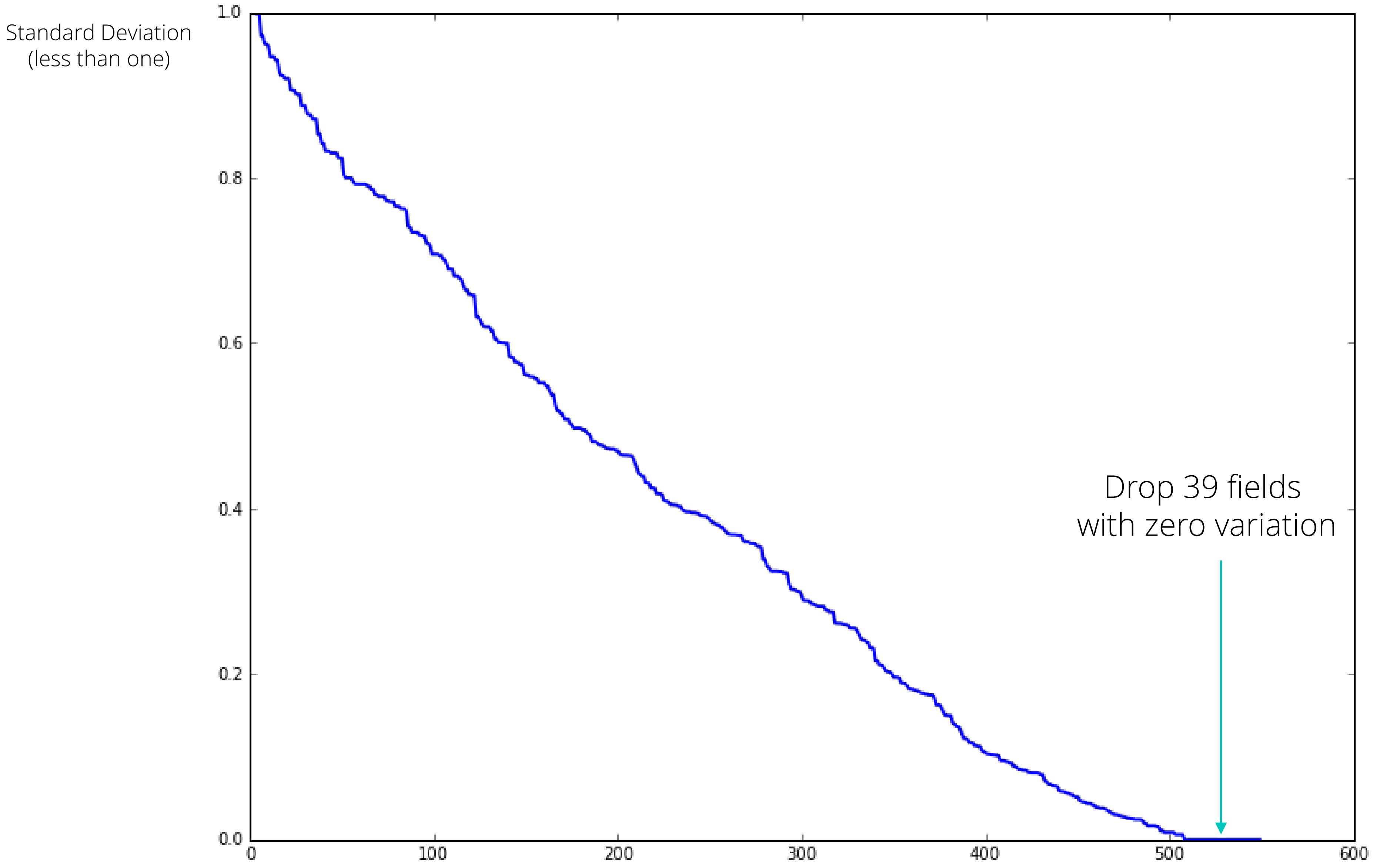



```python
import pandas as pd

desc = X.describe().T

# Percent missing
desc['missing %'] = 1 - (desc['count'] / len(X))
```

2. Amount of Variation

- Drop or review variables that have a very low variation
  - $\boldsymbol{VAR(x) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$
  - Either standardize all variables, or use standard deviation $\boldsymbol{\sigma}$ to account for variables with difference scales
  - Drop variables with zero variation (unary)

2. Amount of Variation



3. Pairwise Correlations

- Many variables are often correlated with each other, and hence are redundant.
- If two variables are highly correlated, keeping only one will help reduce dimensionality without much loss of information.
  - Which variable to keep? The one that has a higher correlation coefficient with the target.

3. Pairwise Correlations

1. Identify pairs of highly correlated variables

	x_1	x_2	x_3	x_4
x_1	1.00	0.15	$\boxed{0.85}$	0.01
x_2		1.00	0.45	0.60
x_3			1.00	0.17
x_4				1.00

Correlation tolerance ≈ 0.65

```
# Correlation matrix for all independent vars
corrMatrix = X.corr()
```

2. Discard variable with weaker correlation with the target

	y
x_1	$\boxed{0.67}$
x_2	0.82
x_3	0.58
x_4	0.25

Keep x_1, Discard x_3

```
absCorrWithDep = []

for var in allVars:
    absCorrWithDep.append(abs(y.corr(X[var])))
```

3. Pairwise Correlations

```
# For each column in the corr matrix
for col in corrMatrix:
    if col in corrMatrix.keys():
        thisCol = []
        thisVars = []

        # Store the corr with the dep var for fields that are highly correlated with each other
        for i in range(len(corrMatrix)):

            if abs(corrMatrix[col][i]) == 1.0 and col <> corrMatrix.keys()[i]:
                thisCorr = 0
            else:
                thisCorr = (1 if abs(corrMatrix[col][i]) > corrTol else -1) * abs(temp[corrMatrix.keys()[i]])
            thisCol.append(thisCorr)
            thisVars.append(corrMatrix.keys()[i])

        mask = np.ones(len(thisCol), dtype = bool) # Initialize the mask

        ctDelCol= 0     # To keep track of the number of columns deleted

        for n, j in enumerate(thisCol):
            # Delete if (a) a var is correlated with others and do not have the best corr with dep,
            # or (b) completely corr with the 'col'
            mask[n] = not (j != max(thisCol) and j >= 0)

            if j != max(thisCol) and j >= 0:
                # Delete the column from corr matrix
                corrMatrix.pop('%s' %thisVars[n])
                temp.pop('%s' %thisVars[n])
                ctDelCol += 1

        # Delete the corresponding row(s) from the corr matrix
        corrMatrix = corrMatrix[mask]
```

4. Multicolinearity

- When two or more variables are highly correlated with each other.
- Dropping one or more variables should help reduce dimensionality without a substantial loss of information.
  - Which variable(s) to drop? Use Condition Index.

	x_1	x_2	x_3	x_4
x_1	1.00	0.15	0.85	0.01
x_2		1.00	0.45	0.60
x_3			1.00	0.17
x_4				1.00

Condition Index	x_1	x_2	x_3	x_4
$u_1 = 1.0$	0.01	0.05	0.00	0.16
$u_2 = 16.5$	0.03	0.12	0.01	0.19
$u_3 = 28.7$	0.05	0.02	0.13	0.25
$\boxed{u_4 = 97.1}$	0.93	0.91	$\boxed{0.98}$	0.11

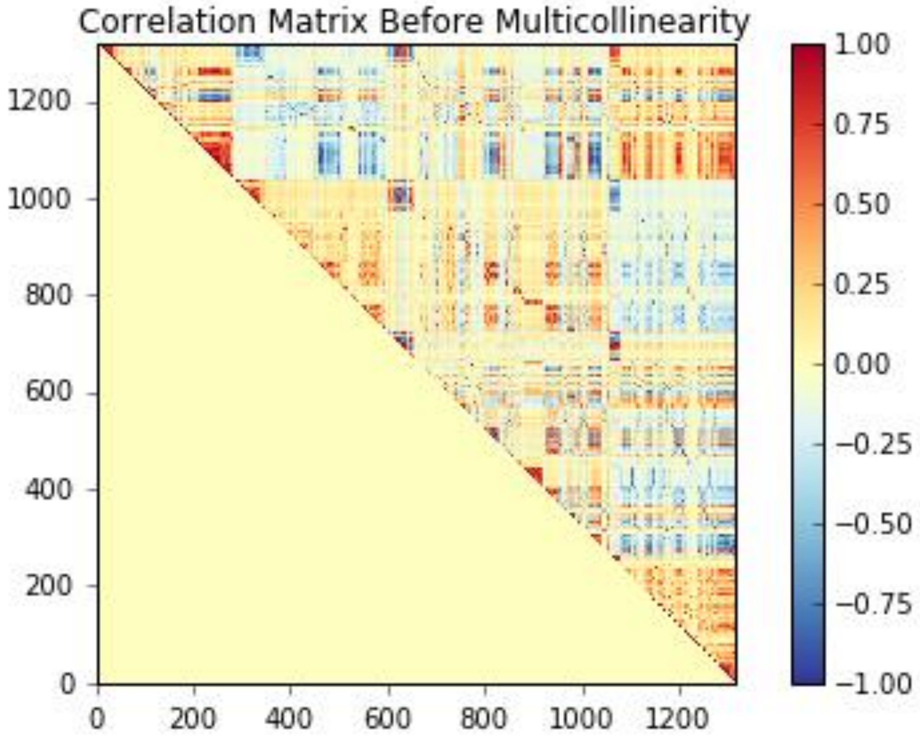
Discard x_3, Iterate

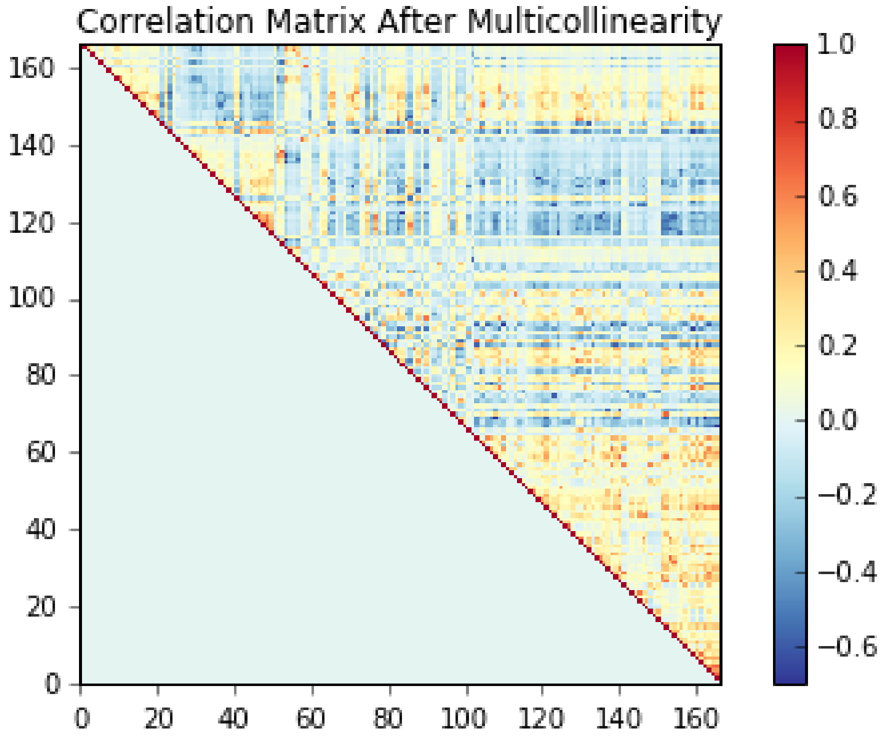
Condition Index tolerance ≈ 0.30

4. Multicolinearity

```
ct = len(corrMatrix)

if ct > minVarsKeep:

    print '\n' + 'Performing multicollinearity analysis'
    while True:
        ct -= 1
        cols = corrMatrix.keys()    # Update the list of columns
        w, v = np.linalg.eig(corrMatrix)    # Eigen values and vectors
        w1 = (max(w) / w) ** 0.5

        # If the condition index <= 30 then multicolinearity is not an issue
        if max(w1) <= condIndexTol or ct == minVarsKeep:
            break

        for i, val in enumerate(w):
            if val == min(w):   # Min value, close to zero
                for j, vec in enumerate(v[:, i]):   # Look into that vector
                    if abs(vec) == max(abs((v[:, i]))):  # Var that has the max weight
                        mask = np.ones(len(corrMatrix), dtype = bool) # Initialize
                        for n, col in enumerate(corrMatrix.keys()):
                            mask[n] = n != j
                        # Delete row
                        corrMatrix = corrMatrix[mask]
                        # Delete column
                        corrMatrix.pop(cols[j])
```

4. Multicolinearity

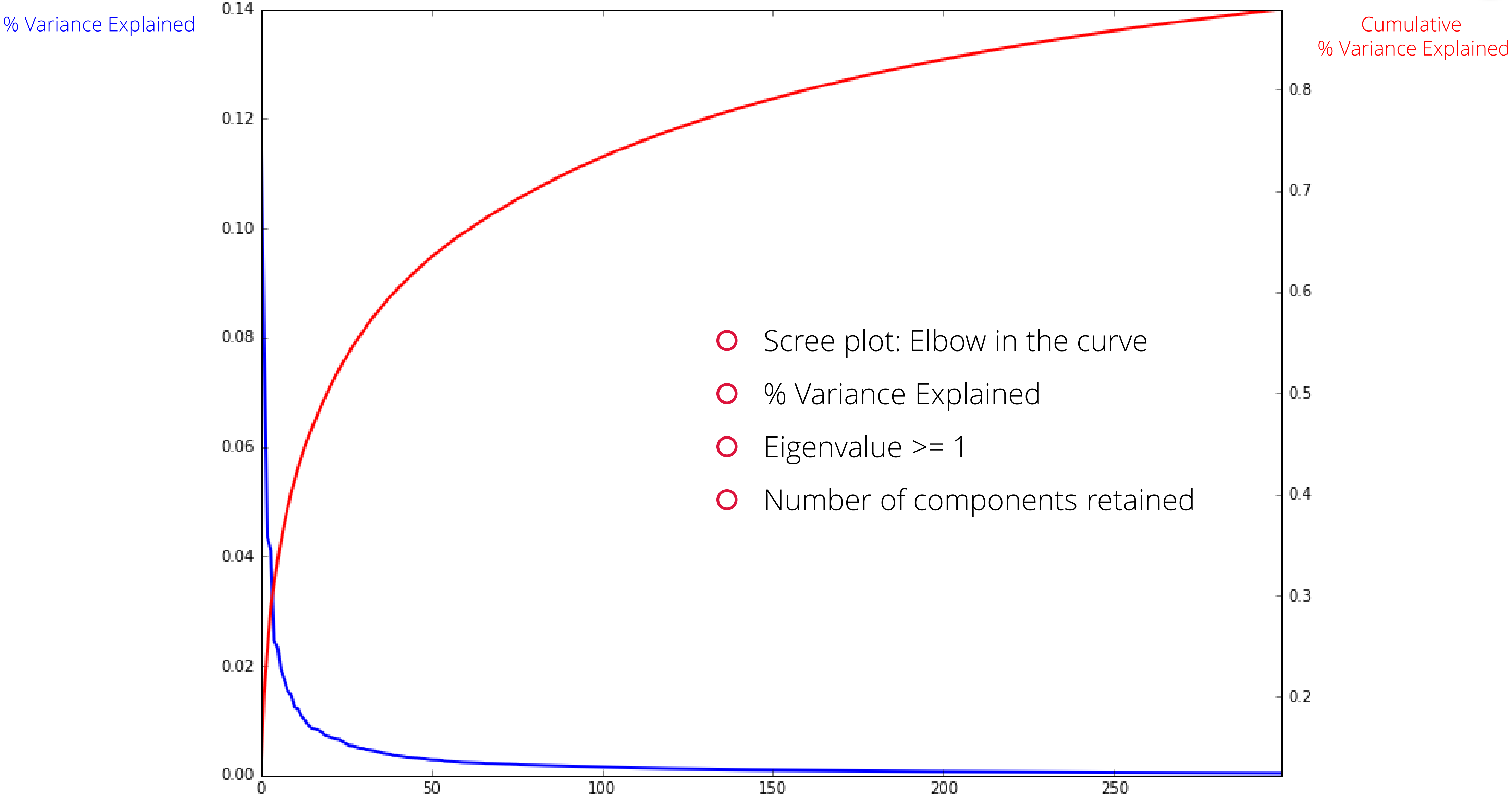




5. Principal Component Analysis (PCA)

- Dimensionality reduction technique which emphasizes variation.
- Eliminates multicollinearity – but explicability is compromised.
  - Uses orthogonal transformation
- When to use:
  - Excessive multicollinearity
  - Explanation of the predictors is not important.
  - A slight overhead in implementation is okay.

5. PCA



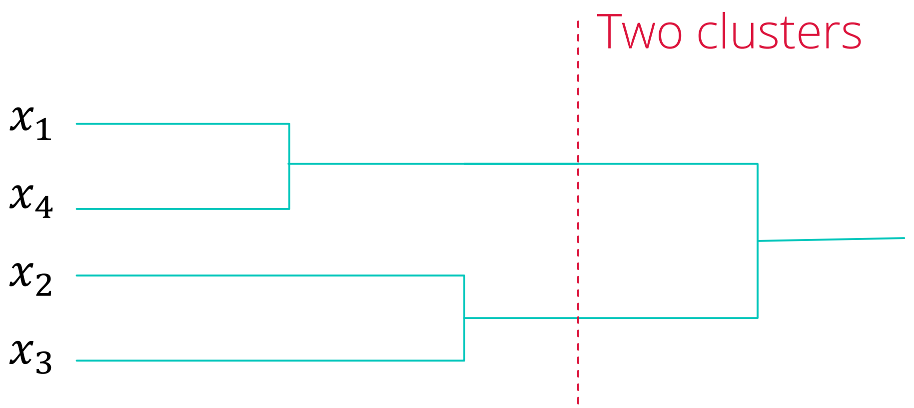
- Scree plot: Elbow in the curve
- % Variance Explained
- Eigenvalue >= 1
- Number of components retained

5. PCA

```python
from sklearn.decomposition import PCA
import pylab as pl

# standardize the input data
train_scaled = preprocessing.scale(train[allVars])

# perform PCA
pca = PCA().fit(train_scaled)

# plot results
pl.figure(figsize = (12, 9))
ax1 = plt.subplot(111)
ax2 = ax1.twinx()
ax1.plot(pca.explained_variance_ratio_[:300], linewidth = 2)
ax2.plot(pca.explained_variance_ratio_.cumsum()[:300], linewidth = 2, color = 'r')
pl.xlabel('Principal Components')

# select the number of components to keep and transform data
reduced_data = PCA(n_components = 250).fit_transform(train_scaled)
```

6. Cluster Analysis

- Dimensionality reduction technique which emphasizes correlation/similarity.
  - Identify groups of variables that are as correlated as possible among themselves and as uncorrelated as possible with variables in other clusters.
- Reduces multicollinearity – and explicability is not (always) compromised.
- When to use:
  - Excessive multicollinearity.
  - Explanation of the predictors is important.



6. Cluster Analysis

```
from sklearn.cluster import FeatureAgglomeration

varClus = FeatureAgglomeration(n_clusters = 10)

varClus.fit(train_scaled)

train_varclus = varClus.transform(train_scaled)
```

- This returns a pooled value for each cluster (i.e., the centroids), which can then be used to build a supervised learning model.
- In SAS, using PROC VARCLUS you can choose one variable from each cluster that is the most representative of that cluster.
  - Use $1 - R^2$ ratio to select a variable from each cluster.

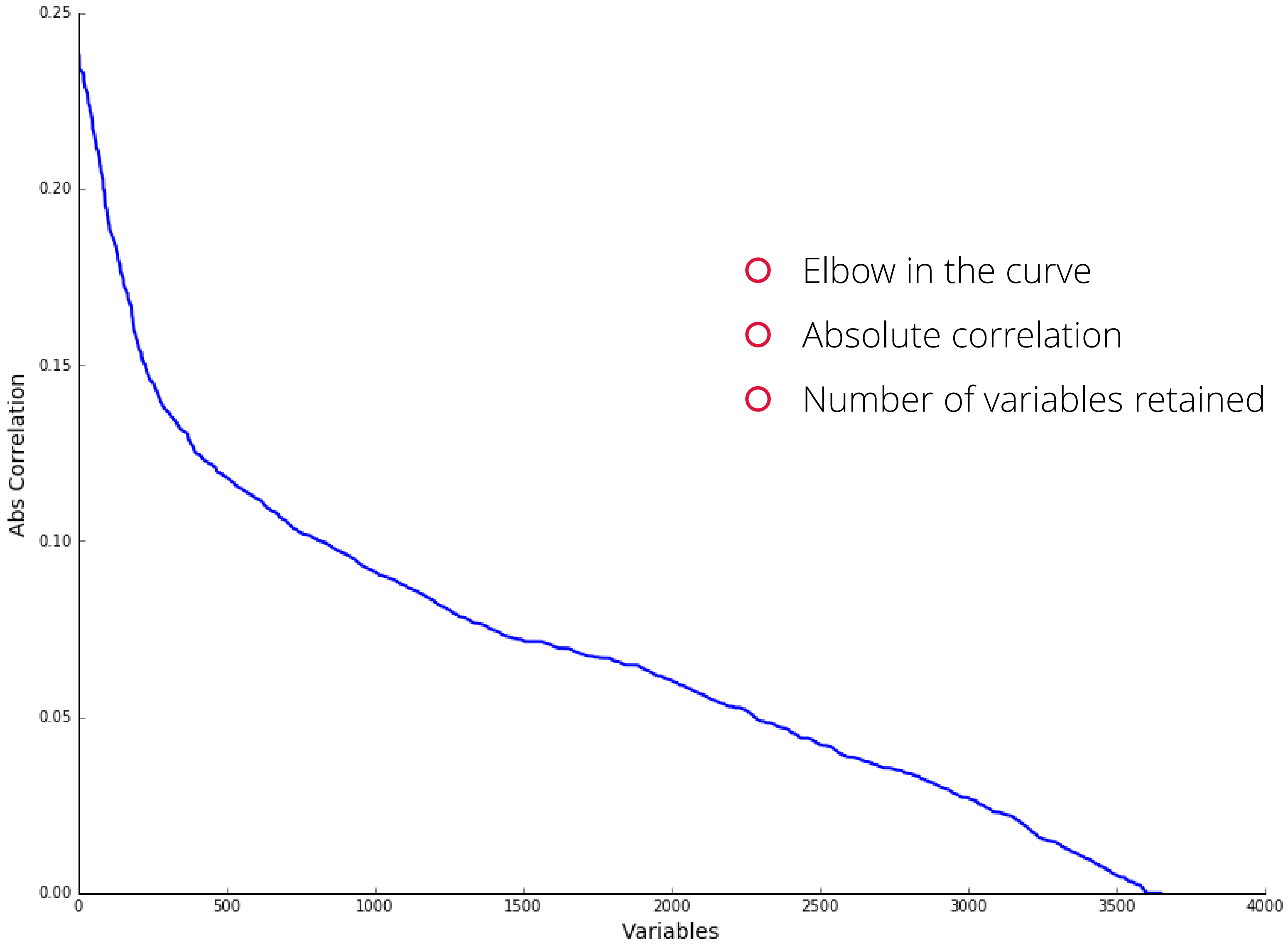
http://scikit-learn.org/stable/modules/generated/sklearn.cluster.FeatureAgglomeration.html#sklearn.cluster.FeatureAgglomeration

7. Correlation with the Target

- Drop variables that have a very low correlation with the target.
  - If a variable has a very low correction with the target, it’s not going to useful for the model (prediction).

```
absCorrWithDep = []

for var in allVars:
    absCorrWithDep.append(abs(y.corr(X[var])))
```

7. Correlation with the Target



- Elbow in the curve
- Absolute correlation
- Number of variables retained

8-10. Forward/Backward/Stepwise Selection

- Forward Selection
  1. Identify the best variable (e.g., based on model accuracy)
  2. Add the next best variable into the model
  3. And so on until some predefined criteria is satisfied.
- Backward Elimination
  1. Start with all variables included in the model.
  2. Drop the least useful variable (e.g., based on the smallest drop in model accuracy)
  3. And so on until some predefined criteria is satisfied.
- Stepwise Selection
  - Similar to forward selection process, but a variable can also be dropped if it's deemed as not useful any more after a certain number of steps.

10. Stepwise Selection

```
for j, var in enumerate(sortedVars):
    okayToAdd = 1
    modelFit = model.fit(X_train[:][sortedVars[:j-dropCt+1]], y_train)

    # make sure the coefficient signs are not reserved
    for k, modelVar in enumerate(sortedVars[:j-dropCt+1]):
        for varName, corr in corrWithDep.iteritems():
            if varName == modelVar:
                if np.transpose(modelFit.coef_)[k] * corr < 0:
                    sortedVars.pop(j-dropCt)
                    dropCt += 1
                    okayToAdd = 0

    if okayToAdd == 1:

        trainPreds = modelFit.predict(X_train[:][sortedVars[:j-dropCt+1]])
        testPreds = modelFit.predict(X_test[:][sortedVars[:j-dropCt+1]])

        # generate evaluation metrics
        trainAccuracy.append(metrics.accuracy_score(y_train, trainPreds))
        testAccuracy.append(metrics.accuracy_score(y_test, testPreds))

        trainROC.append(metrics.roc_auc_score(y_train, trainPreds))
        testROC.append(metrics.roc_auc_score(y_test, testPreds))

        selectedVars.append(sortedVars[j-dropCt])

    if j-dropCt == maxSteps:
        break
```

11. LASSO

- Least Absolute Shrinkage and Selection Operator
- Two birds, one stone: Variable Selection + Regularization

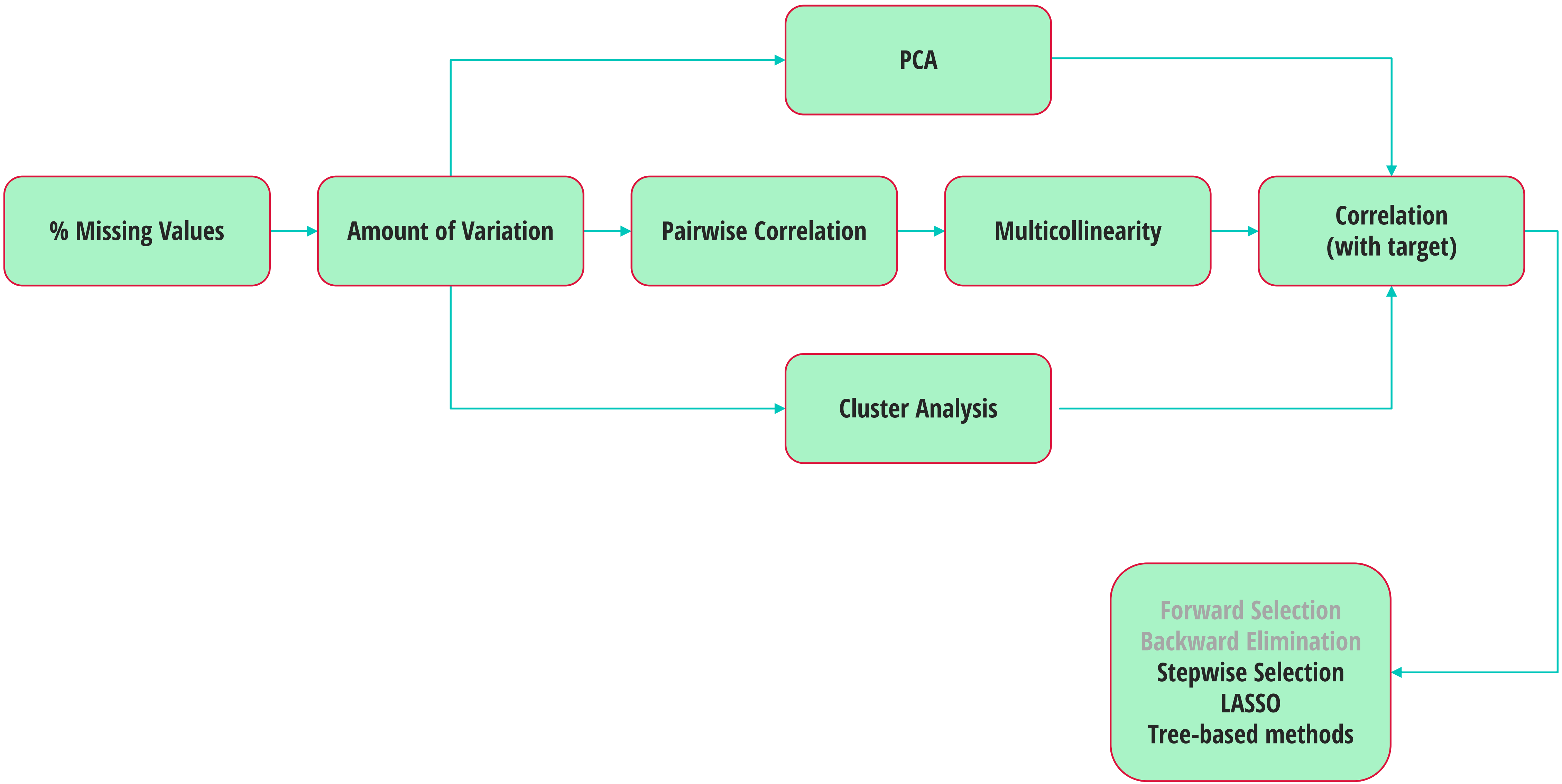
```
while jLasso <= maxPreds:

    thisLASSOFit = LogisticRegression(C = thisC,
                                      penalty = 'l1',
                                      random_state = 314).fit(train1_scaled, y_train)

    modelCoeff = np.transpose(thisLASSOFit.coef_)[np.where(thisLASSOFit.coef_ <> 0)[1]]
    thisLASSOPreds = train.keys()[np.where(thisLASSOFit.coef_ <> 0)[1]]
    thisParamsList = pd.DataFrame(zip(thisLASSOPreds, modelCoeff))
```

12. Tree-based

- Forests of trees to evaluate the importance of features
- Fit a number of randomized decision trees on various sub-samples of the dataset and use averaging to rank order features

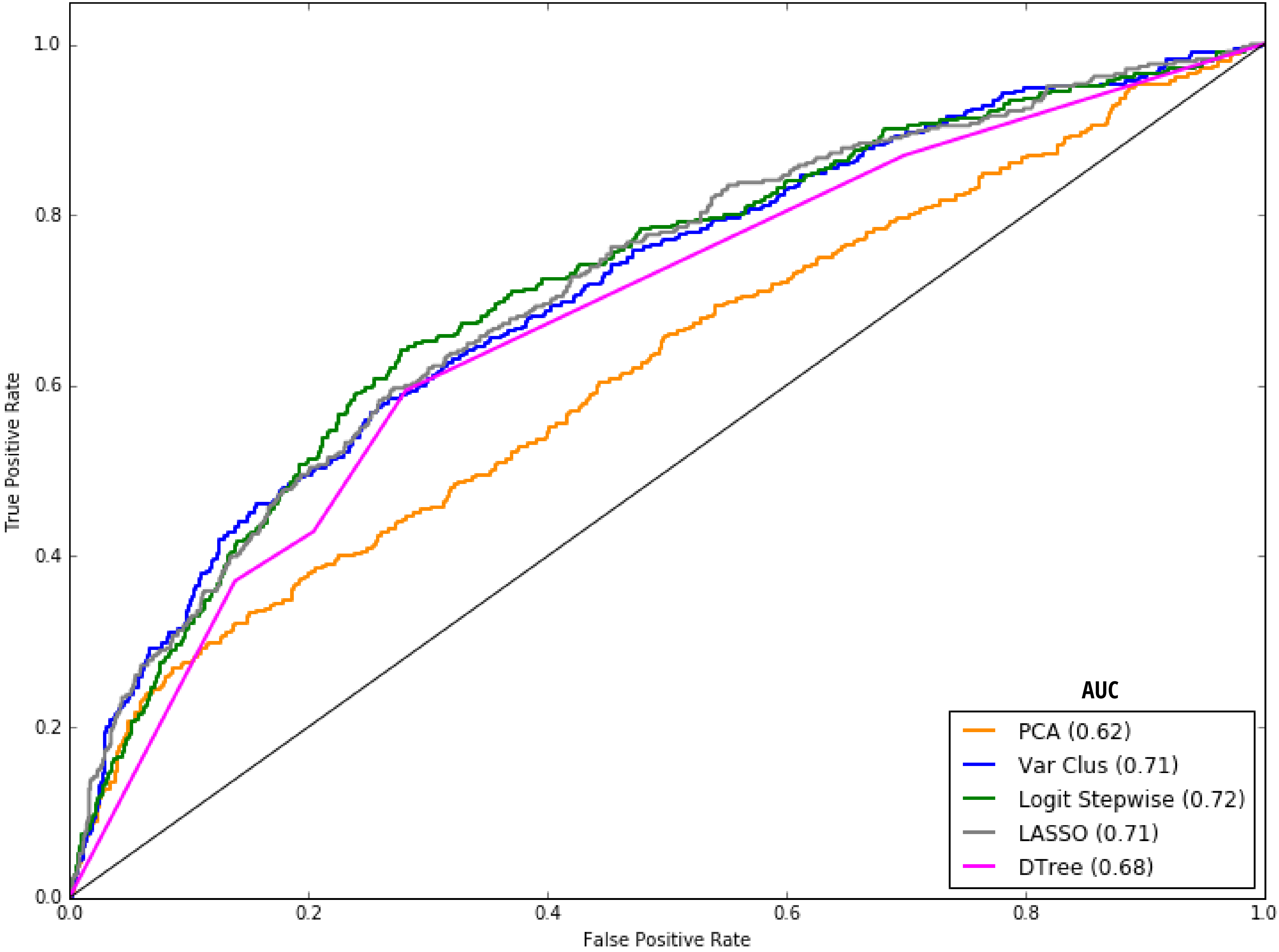
```
orderedParams['DTree'] = {}
orderedImportances = {}

minSplitNum = int(minSplit * len(train))
minLeafNum = int(minLeaf * len(train))

selForestFit = ExtraTreesClassifier(n_estimators = 100,
                                    min_samples_split = minSplitNum,
                                    min_samples_leaf = minLeafNum).fit(train, y_train)

importances = selForestFit.feature_importances_
selForestRanks = np.argsort(importances)[::-1]

# Ordered list of best predictors
for rank, ind in enumerate(selForestRanks):
    var = train.keys()[ind]
    orderedParams['DTree'][rank] = var
    orderedImportances[var] = importances[ind]
```

Workflow

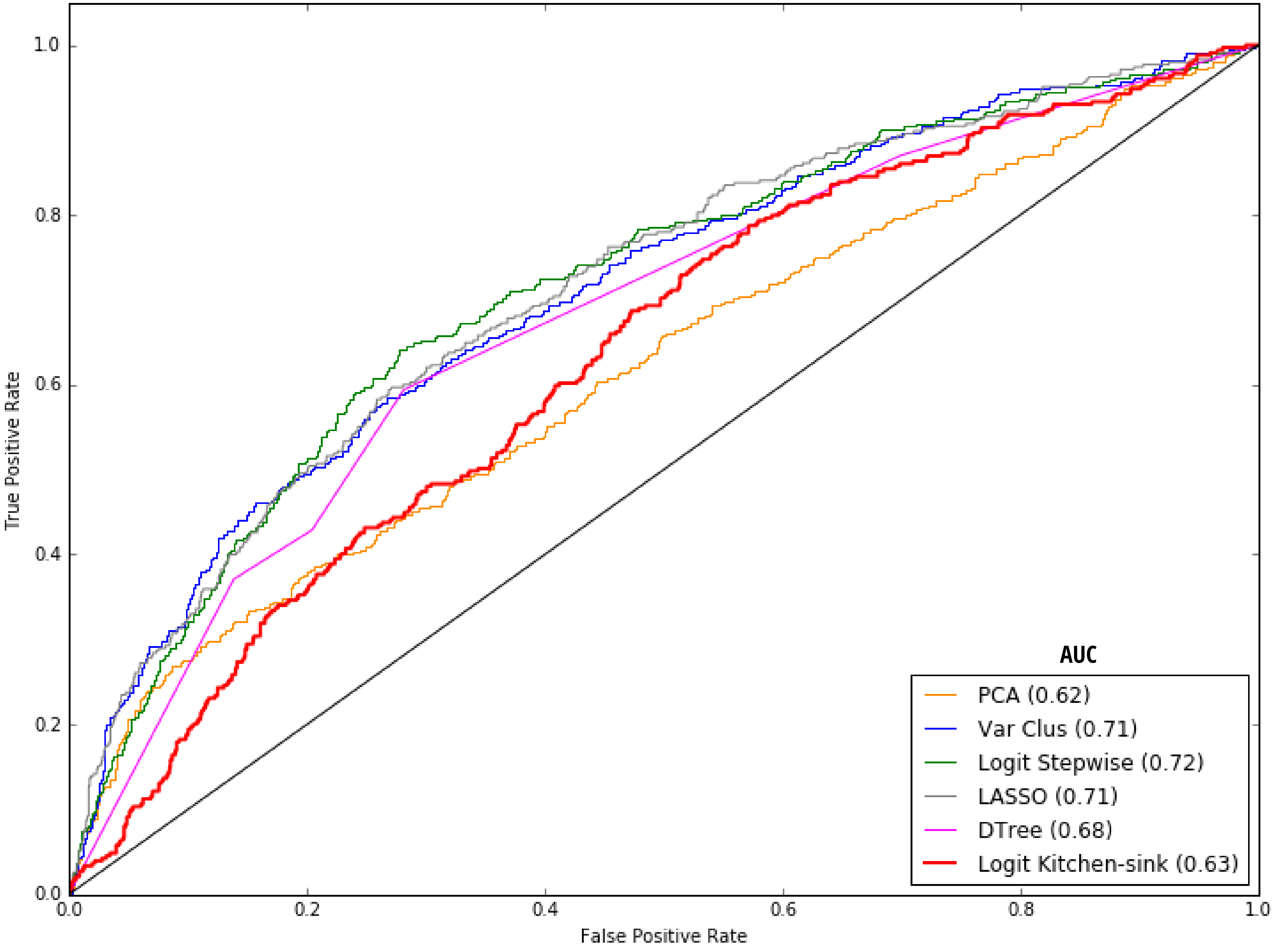


PCA
% Missing Values
Amount of Variation
Pairwise Correlation
Multicollinearity
Correlation
(with target)
Cluster Analysis
Forward Selection
Backward Elimination
Stepwise Selection
LASSO
Tree-based methods

Comparisons



The Kitchen Sink Approach



Dimensionality Reduction

All you can eat

Eat healthy

THANK YOU!

QUESTIONS?

vishal@derive.io

www.linkedIn.com/in/VishalJP