Dimension Reduction

Data Mining for Business Analytics in Python

Shmueli, Bruce, Gedeck & Patel

Dimension Reduction

- The dimension of a dataset, which is the number of variables, must be reduced for the data mining algorithms to operate efficiently. This process is part of the pilot/prototype phase of data mining and is done before deploying a model.
- Several dimension reduction approaches:

1) Incorporating domain knowledge to remove or combine categories
2) Using data summaries to detect information overlap between variables (and remove or combine redundant variables or categories)
3) Using data conversion techniques such as converting categorical variables into numerical variables
4) Employing automated reduction techniques, such as Principal Components Analysis (PCA), where a new set of variables (which are weighted averages of the original variables) is created. These new variables are uncorrelated and a small subset of them usually contains most of their combined information (hence, we can reduce dimension by using only a subset of the new variables).

- Introducing data mining methods such as “regression models and classification” and “regression trees”, which can be used for removing redundant variables and for combining “similar” categories of categorical variables.
- In Artificial Intelligence literature, dimension reduction is often referred to as *factor selection* or *feature extraction.*

Exploring the data

Statistical summary of data: common metrics

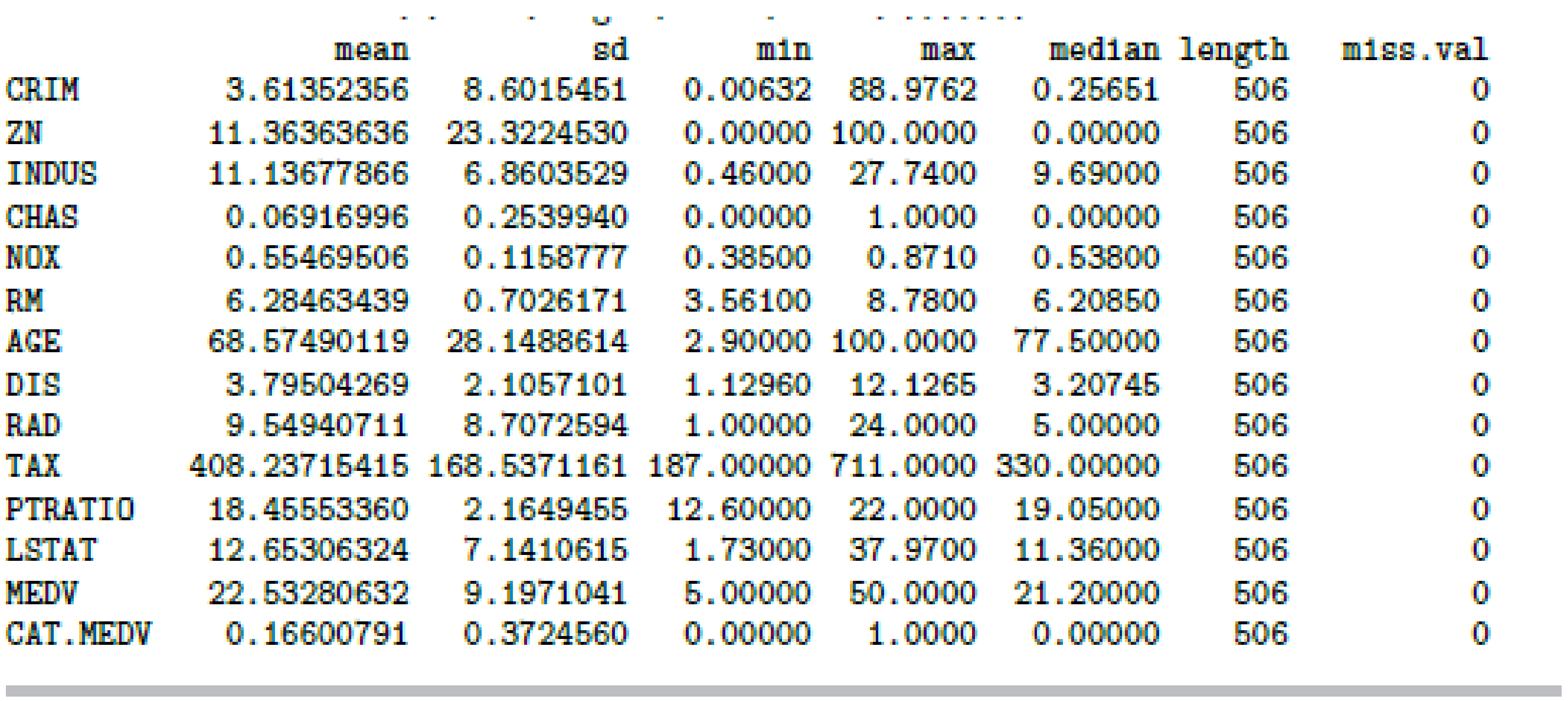
- Average
- Median
- Minimum
- Maximum
- Standard deviation
- Counts & percentages

Exploring the data

```
bostonHousing_df = pd.read_csv('BostonHousing.csv')
bostonHousing_df = bostonHousing_df.rename(columns={'CAT.
    MEDV': 'CAT_MEDV'})

# Compute mean, standard dev., min, max, median, length,
   and missing values for all variables
pd.DataFrame({'mean': bostonHousing_df.mean(),
'sd': bostonHousing_df.std(),
'min': bostonHousing_df.min(),
'max': bostonHousing_df.max(),
'median': bostonHousing_df.median(),
'length': len(bostonHousing_df),
'miss.val': bostonHousing_df.isnull().sum(),
})
```

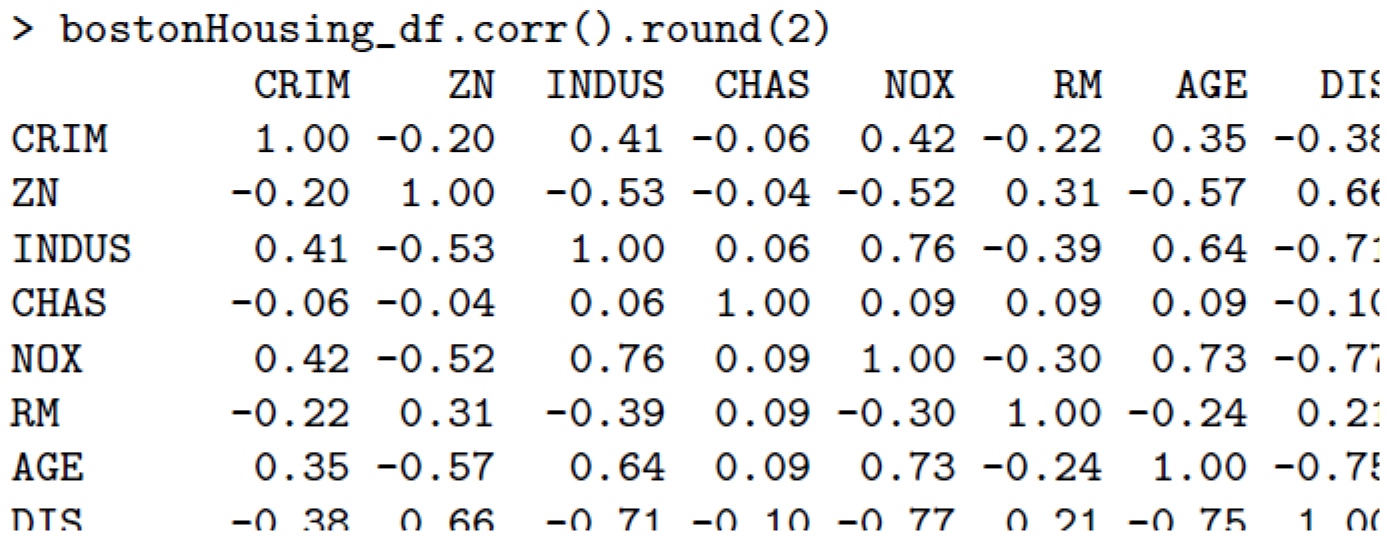
Summary Statistics for Boston Housing Data



```
                 mean          sd       min      max    median length  miss.val
CRIM       3.61352356   8.6015451   0.00632  88.9762   0.25651    506         0
ZN        11.36363636  23.3224530   0.00000 100.0000   0.00000    506         0
INDUS     11.13677866   6.8603529   0.46000  27.7400   9.69000    506         0
CHAS       0.06916996   0.2539940   0.00000   1.0000   0.00000    506         0
NOX        0.55469506   0.1158777   0.38500   0.8710   0.53800    506         0
RM         6.28463439   0.7026171   3.56100   8.7800   6.20850    506         0
AGE       68.57490119  28.1488614   2.90000 100.0000  77.50000    506         0
DIS        3.79504269   2.1057101   1.12960  12.1265   3.20745    506         0
RAD        9.54940711   8.7072594   1.00000  24.0000   5.00000    506         0
TAX      408.23715415 168.5371161 187.00000 711.0000 330.00000    506         0
PTRATIO   18.45553360   2.1649455  12.60000  22.0000  19.05000    506         0
LSTAT     12.65306324   7.1410615   1.73000  37.9700  11.36000    506         0
MEDV      22.53280632   9.1971041   5.00000  50.0000  21.20000    506         0
CAT.MEDV   0.16600791   0.3724560   0.00000   1.0000   0.00000    506         0
```

Correlation Matrix for Boston Housing Data

portion of the correlation table shown, using `corr` from `pandas`



```
> bostonHousing_df.corr().round(2)
          CRIM    ZN  INDUS  CHAS   NOX    RM   AGE   DI
CRIM      1.00 -0.20   0.41 -0.06  0.42 -0.22  0.35 -0.3
ZN       -0.20  1.00  -0.53 -0.04 -0.52  0.31 -0.57  0.6
INDUS     0.41 -0.53   1.00  0.06  0.76 -0.39  0.64 -0.7
CHAS     -0.06 -0.04   0.06  1.00  0.09  0.09  0.09 -0.1
NOX       0.42 -0.52   0.76  0.09  1.00 -0.30  0.73 -0.7
RM       -0.22  0.31  -0.39  0.09 -0.30  1.00 -0.24  0.2
AGE       0.35 -0.57   0.64  0.09  0.73 -0.24  1.00 -0.7
DIS      -0.38  0.66  -0.71 -0.10 -0.77  0.21 -0.75  1.0
```

Using `value_counts()`, in `pandas`, to tabulate counts

```
> bostonHousing_df.CHAS.value_counts()


0    1
471  35

Name: CHAS, dtype: int64
```

35 neighborhoods have a CHAS value of “1,” i.e. they border the Charles River

Using `groupby()` to tabulate counts using multiple variables

```
# Create bins of size 1 for variable using the method pd.cut. Default
# creates a categorical variable, e.g. (6,7]. labels=False determines
# integers instead, e.g. 6.
bostonHousing_df['RM_bin’] = pd.cut(bostonHousing_df.RM, range(0, 10), labels=False)

# Compute average of MEDV by (binned) RM and CHAS. First group the data frame
# using the groupby method, then restrict the analysis to MEDV and determine the
# mean for each group.
bostonHousing_df.groupby(['RM_bin', 'CHAS'])['MEDV'].mean()
```

```
        RM_bin CHAS
        3         0    25.300000
        4         0    15.407143
        5         0    17.200000
                  1    22.218182
        6         0    21.769170
                  1    25.918750
        7         0    35.964444
                  1    44.066667
        8         0    45.700000
                  1    35.950000
```

In neighborhoods where houses averaged 3 rooms and did not border the Charles, median value was 25.3 ($000)

RM = Average number of rooms per dwelling

- pd.cut(bostonHousing_df.RM, range(0, 10), labels=False): This part of the code applies the pd.cut function to the 'RM' column. The pd.cut function is used to create discrete bins for the values in the specified column.
- bostonHousing_df.RM: This is the Series containing the values to be binned.
- range(0, 10): This specifies the bin edges. It creates bins from 0 (inclusive) to 10 (exclusive). So, you'll have bins like [0, 1), [1, 2), [2, 3), and so on.
- labels=False: This parameter specifies that you want the output to be numeric bin labels rather than categorical labels.

Use `pivot_table()` in `pandas` for pivot tables

```
# use pivot_table() to reshape data and generate pivot table
pd.pivot_table(bostonHousing_df, values='MEDV', index=['RM_bin'],
  columns=['CHAS'], aggfunc=np.mean, margins=True)
```

```
CHAS          0          1           All
RM_bin
3       25.300000   NaN          25.300000
4       15.407143   NaN          15.407143
5       17.200000   22.218182    17.551592
6       21.769170   25.918750    22.015985
7       35.964444   44.066667    36.917647
8       45.700000   35.950000    44.200000
All     22.093843   28.440000    22.532806
```

neighborhoods not bordering Charles River, with 8 rooms, have MEDV = 45.7

- aggfunc=np.mean: This specifies that the aggregation function to be used is np.mean, which calculates the mean (average) of values in each group of the pivot table. We do not need to insert aggfunc=np.mean as "avg" is the default parameter for aggfunc.
- margins=True: This includes the margins (totals) in the pivot table, giving you the mean values for 'MEDV' across all 'RM_bin' and 'CHAS' groups.
- **margins** *bool, default False* If margins=True, special All columns and rows will be added with partial group aggregates across the categories on the rows and columns.

Reducing Categories

- A single categorical variable with *m* categories is typically transformed into *m* or *m-1* dummy variables
- Each dummy variable takes the values 0 or 1

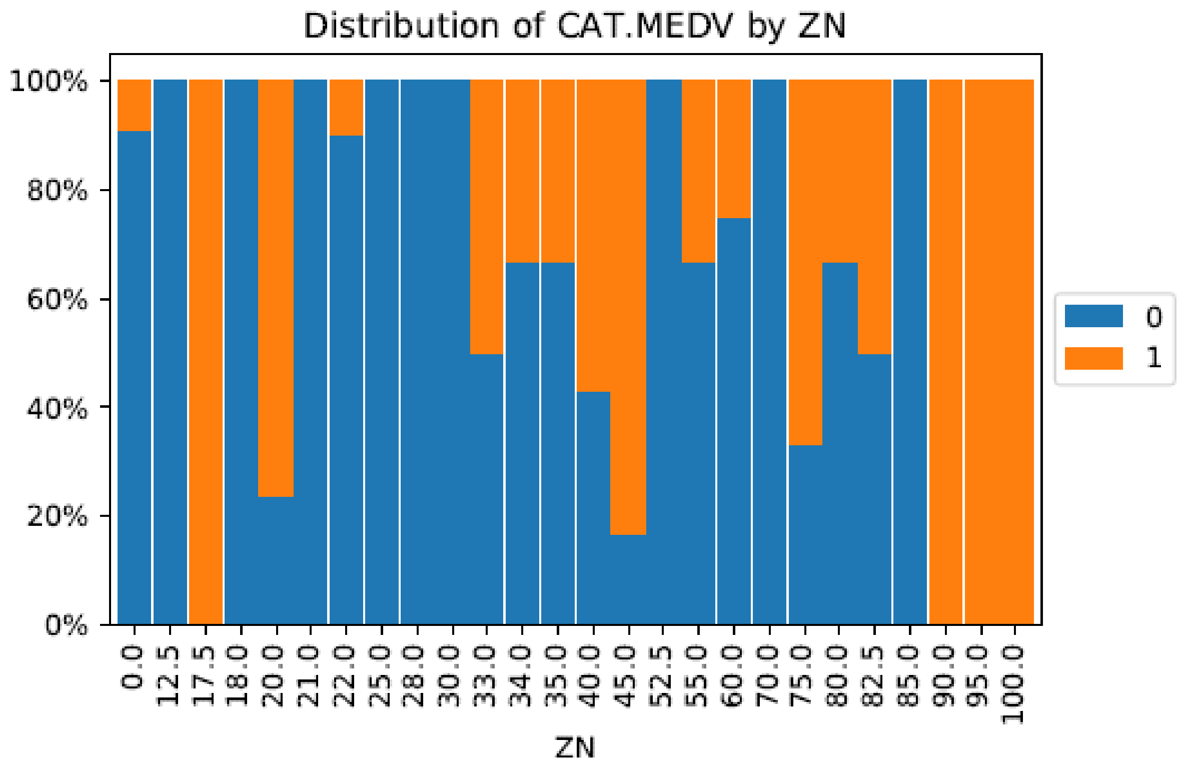
  0 = “no”

  1 = “yes”
- Problem: Can end up with too many variables
- Solution: Reduce by combining categories that are close to each other
- Use pivot tables to assess outcome variable sensitivity to the dummies
- Exception: Naïve Bayes can handle categorical variables without transforming them into dummies

Combining Categories

Stacked bar chart:

- Many zoning categories are the same or similar with respect to CATMEDV



ZN = Percentage of residential land zoned for lots over 25,000 ft^2

CAT.MEDV = Is median value of owner-occupied homes in tract above $30,000 (CAT.MEDV = 1) or not (CAT.MEDV = 0)

Example: For a lot with 33 percent of residence, 50% of homes belong to category 0 and 50% of home belong to category 1

CODE FOR STACKED BAR CHART

```
# use method crosstab to create a cross-tabulation of two
    variables
tbl = pd.crosstab(bostonHousing_df.CAT_MEDV, bostonHousing_df.ZN)
# convert numbers to ratios
propTbl = tbl / tbl.sum()
propTbl.round(2)
# plot the ratios in a stacked bar chart
ax = propTbl.transpose().plot(kind='bar', stacked=True)
ax.set_yticklabels(['{:,.0%}'.format(x) for x in
ax.get_yticks()])
plt.title('Distribution of CAT.MEDV by ZN')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

- The format string '{:,.0%}' is used to format a numeric value as a percentage with specific formatting:
- : Marks the beginning of the format specification.
- ,: Specifies that a comma should be used as a thousand separator for large numbers. For example, the number 1000 would be formatted as "1,000."
- .0: Specifies that there should be no decimal places in the formatted percentage.
- %: Specifies that the value should be presented as a percentage, and the % symbol will be appended to the end. So, it will display percentages like "50%" or "75%" with a comma for thousands if needed.
- The legend appears to the right of the plot because bbox_to_anchor=(1, 0.5) places the legend's anchor point at the right edge of the axes. The loc='center left' specifies that the center of the left side of the legend box aligns with that anchor point, thus moving the legend outside the plot area on the right.

Principal Components Analysis

Goal: Reduce a set of **numerical** variables.

The idea: Remove the overlap of information between these variable. [“Information” is measured by the sum of the variances of the variables.]

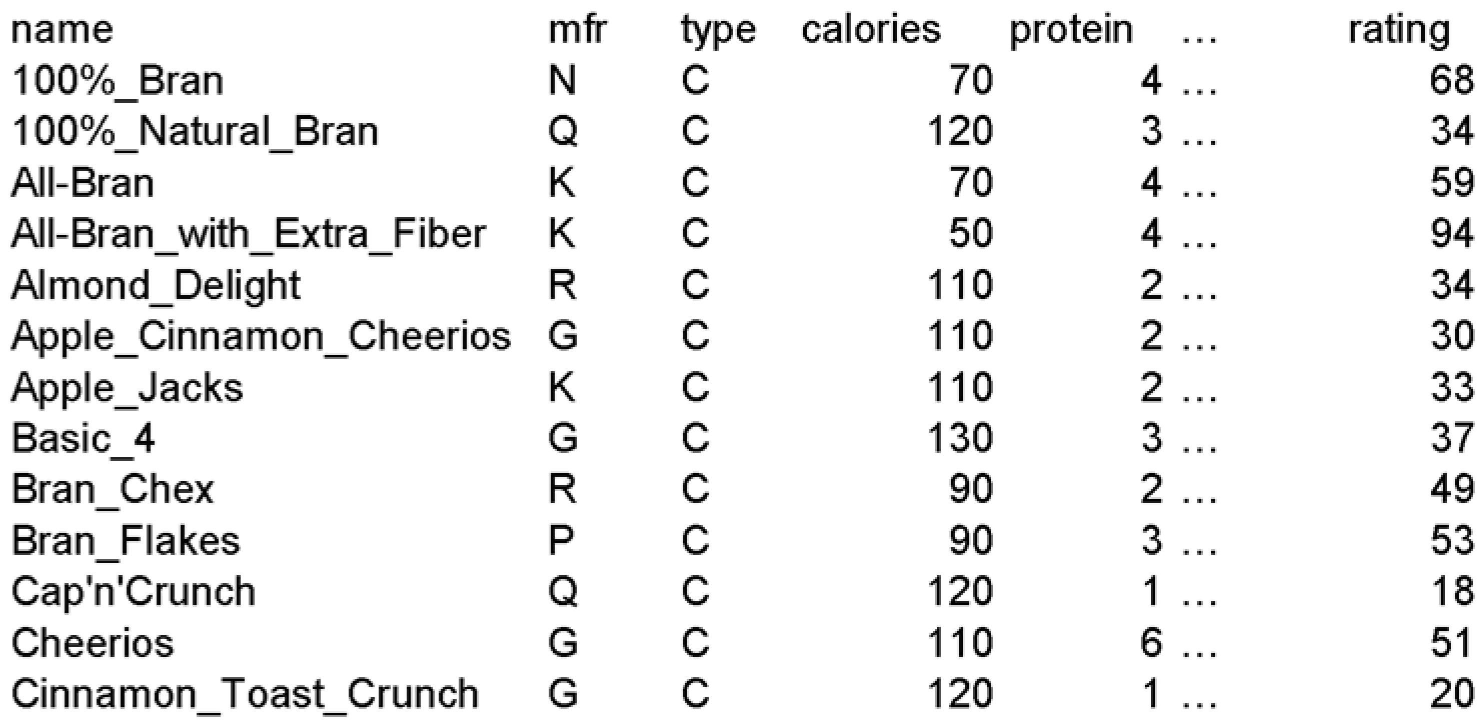
Final product: A smaller number of numerical variables that contain most of the information

Principal Components Analysis

How does PCA do this?

- Creates new variables that are linear combinations of the original variables (i.e., they are weighted averages of the original variables).
- These linear combinations are uncorrelated (no information overlap), and only a few of them contain most of the original information.
- The new variables are called *principal components*.

Example – Breakfast Cereals (excerpt)



name	mfr	type	calories	protein	…	rating
100%_Bran	N	C	70	4	…	68
100%_Natural_Bran	Q	C	120	3	…	34
All-Bran	K	C	70	4	…	59
All-Bran_with_Extra_Fiber	K	C	50	4	…	94
Almond_Delight	R	C	110	2	…	34
Apple_Cinnamon_Cheerios	G	C	110	2	…	30
Apple_Jacks	K	C	110	2	…	33
Basic_4	G	C	130	3	…	37
Bran_Chex	R	C	90	2	…	49
Bran_Flakes	P	C	90	3	…	53
Cap'n'Crunch	Q	C	120	1	…	18
Cheerios	G	C	110	6	…	51
Cinnamon_Toast_Crunch	G	C	120	1	…	20

Description of Variables

name: name of cereal

mfr: manufacturer

type: cold or hot

calories: calories per serving

protein: grams

fat: grams

sodium: mg

fiber: grams

carbo: grams complex carbohydrates

sugars: grams

potass: mg

vitamins: % FDA rec

shelf: display shelf

weight: oz. 1 serving

cups: # in one serving

rating: consumer reports

Consider calories & ratings covariance matrix

Covariance Matrix Formula: $$\begin{bmatrix} Var(X_1) & \dots & Cov(X_n, X_1) \\ \vdots & \ddots & \vdots \\ Cov(X_n, X_1) & \dots & Var(X_n) \end{bmatrix}$$

```
cereal_df = pd.read_csv('Cereals.csv')
cereal_df[['calories', 'rating']].cov()

cereal_df.calories.var()
cereal_df.rating.var()
```

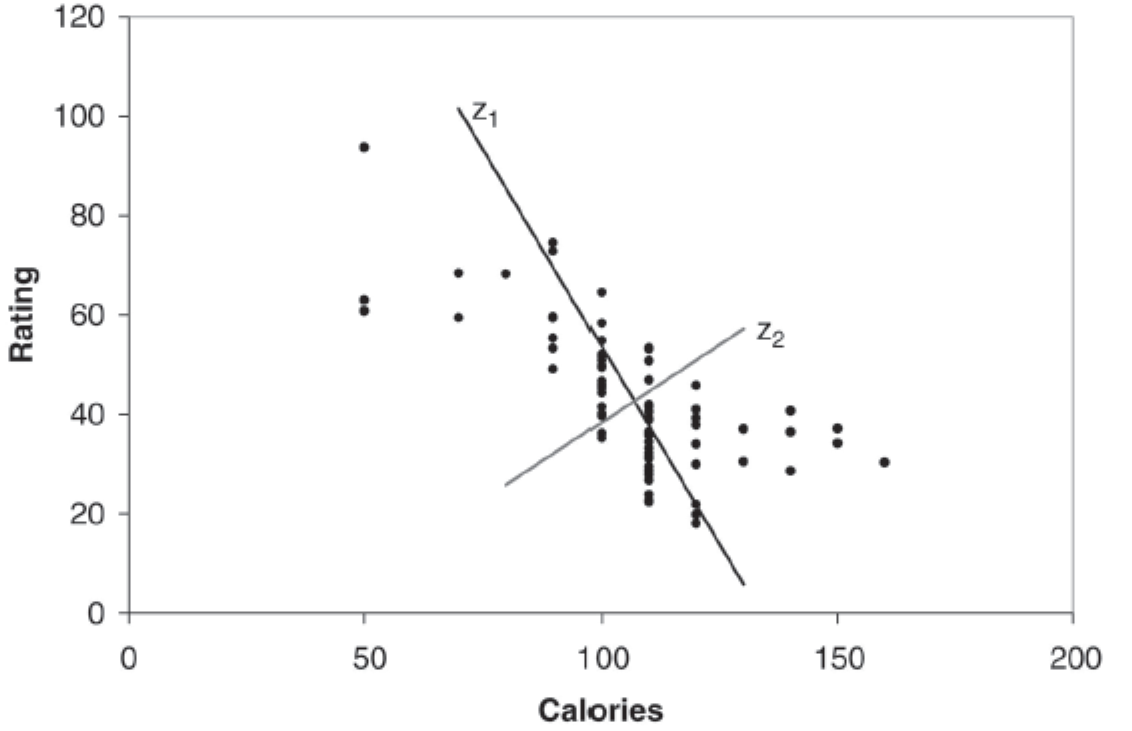
	calories	ratings
calories	379.63	-189.68
ratings	-189.68	197.32

- Total variance (=“information”) is sum of individual variances: 379.63 + 197.32 = 577
- Calories accounts for 379.63/577 = 66%
- If we want to make decision with just calories, we lose 34% of the variation
- The main purpose of a covariance matrix is to show how pairs of variables in a dataset covary or move together. A positive covariance indicates that when one variable increases, the other tends to increase as well, while a negative covariance indicates that one variable tends to decrease when the other increases.
- The magnitude of the covariances can indicate the degree of dependence or independence between variables. If the covariances are close to zero, the variables are relatively independent. Larger positive or negative covariances indicate stronger relationships.
- While the covariance provides information about the direction and strength of the linear relationship between variables, it does not offer a standardized measure like correlation. Correlation, which is derived from the covariance matrix, provides a normalized measure that ranges between -1 and 1, making it easier to interpret.

Using linear combinations to redistribute the variability in a more polarized way

Z_1 and Z_2 are two linear combinations.

- Z_1 has the highest variation (spread of values)
- Z_2 has the lowest variation



PCA output for these two variables

```
from sklearn.decomposition import PCA
pcs = PCA(n_components=2)
pcs.fit(cereals_df[['calories', 'rating']])
```

Weights to project original data onto Z_1 & Z_2, e.g. (-0.847, 0.532) are weights for Z_1

```
pcs.explained_variance_ratio_
pcs_df = pd.DataFrame(pcs.components_, columns=['PC1', 'PC2'])
```

```
                           PC1        PC2
 cereals.df.calories  -0.8470535  0.5315077
 cereals.df.rating     0.5315077  0.8470535
```

```
pcsSummary = pd.DataFrame({'Standard deviation': np.sqrt(pcs.explained_variance_),
                           'Proportion of variance': pcs.explained_variance_ratio_,
                           'Cumulative proportion’: np.cumsum(pcs.explained_variance_ratio_)})
pcsSummary = pcsSummary.transpose()
```

```
 Importance of components:
                          PC1     PC2
 Standard deviation     22.3165  8.8844
 Proportion of Variance  0.8632  0.1368
 Cumulative Proportion   0.8632  1.0000
```

86% of the total variance is accounted for by component 1

Principal Component Scores for the First Five Records

Score_PC1 = Weight_PC1 * (Value_PC1 – Mean of Column/Variable in PC1)

Score_PC2 = Weight_PC2 * (Value_PC2 – Mean of Column/Variable in PC2)

SCORE = Score_PC1 + Score_PC2

```
pd.DataFrame(pcs.transform(cereal_df[['calories', 'rating’]]),
                                                columns=[PC1', ‘PC2'])
```

```
              PC1        PC2
[1,]    44.921528   2.1971833
[2,]   -15.725265  -0.3824165
[3,]    40.149935  -5.4072123
[4,]    75.310772  12.9991256
[5,]    -7.041508  -5.3576857
```

PCA for the 13 Numerical Variables in the Cereals Data

```
pcs = PCA()
pcs.fit(cereals_df.iloc[:, 3:].dropna())
pcsSummary_df = pd.DataFrame({'Standard deviation':
   np.sqrt(pcs.explained_variance_), 'Proportion of variance':
   pcs.explained_variance_ratio_,'Cumulative proportion':
   np.cumsum(pcs.explained_variance_ratio_)})
pcsSummary_df = pcsSummary_df.transpose()
pcsSummary_df.columns = ['PC{}'.format(i) for i in range(1,
   len(pcsSummary_df.columns) + 1)]
pcsSummary_df.round(4)
```

```
                            PC1      PC2      PC3      PC4     PC5     PC6
Standard deviation      83.7641  70.9143  22.6437  19.1815  8.4232  2.0917
Proportion of variance   0.5395   0.3867   0.0394   0.0283  0.0055  0.0003
Cumulative proportion    0.5395   0.9262   0.9656   0.9939  0.9993  0.9997

                           PC7     PC8     PC9    PC10    PC11   PC12   PC13
Standard deviation      1.6994  0.7796  0.6578  0.3704  0.1864  0.063   0.0
Proportion of variance  0.0002  0.0000  0.0000  0.0000  0.0000  0.000   0.0
Cumulative proportion   0.9999  1.0000  1.0000  1.0000  1.0000  1.000 1.0
```

The first two components account for 93% of the total variance, so using 2-3 components in further modeling would probably be sufficient

The Weightings for the First Five Components

```
pd.DataFrame(pcs.components_, columns=pcsSummary_df.columns,
                              index=cereal_df.iloc[:,3:].columns)
pcsComponents_df.iloc[:, :5]
```

```
                 PC1       PC2       PC3       PC4       PC5
calories   -0.077984 -0.009312  0.629206 -0.601021  0.454959
protein     0.000757  0.008801  0.001026  0.003200  0.056176
fat         0.000102  0.002699  0.016196 -0.025262 -0.016098
sodium     -0.980215  0.140896 -0.135902 -0.000968  0.013948
fiber       0.005413  0.030681 -0.018191  0.020472  0.013605
carbo      -0.017246 -0.016783  0.017370  0.025948  0.349267
sugars     -0.002989 -0.000253  0.097705 -0.115481 -0.299066
potass      0.134900  0.986562  0.036782 -0.042176 -0.047151
vitamins   -0.094293  0.016729  0.691978  0.714118 -0.037009
shelf       0.001541  0.004360  0.012489  0.005647 -0.007876
weight     -0.000512  0.000999  0.003806 -0.002546  0.003022
cups       -0.000510 -0.001591  0.000694  0.000985  0.002148
rating      0.075296  0.071742 -0.307947  0.334534  0.757708
```

Generalization

X_1, X_2, X_3, … X_p, original *p* variables

Z_1, Z_2, Z_3, … Z_p, weighted averages of original variables

All pairs of Z variables have 0 correlation

Order Z's by variance (Z_1 largest, Z_p smallest)

Usually the first few Z variables contain most of the information, and so the rest can be dropped.

Normalizing data

- In these results, sodium dominates first PC
- Just because of the way it is measured (mg), its scale is greater than almost all other variables
- Hence its variance will be a dominant component of the total variance
- Normalize each variable to remove scale effect
    - Divide by std. deviation (may subtract mean first)
- Normalization (= standardization) is usually performed in PCA; otherwise measurement units affect results

```
from sklearn.preprocessing import scale
pcs = PCA()
pcs.fit(preprocessing.scale(cereals_df.iloc[:, 3:].dropna()))
```

Normalize the variables

PCA Output Using all 13 ***Normalized*** Numerical Variables

```
                         PC1    PC2    PC3    PC4    PC5    PC6
Standard deviation     1.9192 1.7864 1.3912 1.0166 1.0015 0.8555
Proportion of variance 0.2795 0.2422 0.1469 0.0784 0.0761 0.0555
Cumulative proportion  0.2795 0.5217 0.6685 0.7470 0.8231 0.8786

                         PC7    PC8    PC9    PC10    PC11   PC12   PC13
Standard deviation     0.8251 0.6496 0.5658 0.3051 0.2537 0.1399   0.0
Proportion of variance 0.0517 0.0320 0.0243 0.0071 0.0049 0.0015   0.0
Cumulative proportion  0.9303 0.9623 0.9866 0.9936 0.9985 1.0000   1.0
```

Weightings for the First Five ***Normalized*** Components

	PC1	PC2	PC3	PC4	PC5
calories	-0.299542	-0.393148	0.114857	-0.204359	0.203899
protein	0.307356	-0.165323	0.277282	-0.300743	0.319749
fat	-0.039915	-0.345724	-0.204890	-0.186833	0.586893
sodium	-0.183397	-0.137221	0.389431	-0.120337	-0.338364
fiber	0.453490	-0.179812	0.069766	-0.039174	-0.255119
carbo	-0.192449	0.149448	0.562452	-0.087835	0.182743
sugars	-0.228068	-0.351434	-0.355405	0.022707	-0.314872
potass	0.401964	-0.300544	0.067620	-0.090878	-0.148360
vitamins	-0.115980	-0.172909	0.387859	0.604111	-0.049287
shelf	0.171263	-0.265050	-0.001531	0.638879	0.329101
weight	-0.050299	-0.450309	0.247138	-0.153429	-0.221283
cups	-0.294636	0.212248	0.140000	-0.047489	0.120816
rating	0.438378	0.251539	0.181842	-0.038316	0.057584

- To convert the initial variables of a new observation to the PCs, you first normalize the new observation using the same normalization method applied to the training data. Then, you apply the PCA transformation learned from the training set (using the eigenvectors of the covariance matrix) to the normalized new data to obtain the PCs.
- Video

PCA in Classification/Prediction

- Apply PCA to training data
- Decide how many PC's to use
- Use variable weights in those PC's with validation/new data
- This creates a new reduced set of predictors in validation/new data

Regression-Based Dimension Reduction

- Multiple Linear Regression or Logistic Regression
- Use subset selection
- Algorithm chooses a subset of variables
- This procedure is integrated directly into the predictive task