Data Science Concepts and Analysis

Week 5: Principal components

(Exploratory data analysis II)

- Covariation between variables
- Eigendecomposition
- Principal components analysis

This week: principal components

Objective: introduce an exploratory technique for exploring covariation among several variables.

- **Covariation between variables**
    - What is covariation and how is it measured quantitatively?
    - Covariance and correlation
    - Correlation matrix
    - Heatmap visualization
- **Eigendecomposition**
    - The eigenvalue problem
    - Geometric interpretation
    - Computations
- **Principal components analysis**
    - What is PCA exactly?
    - PCA in the low-dimensional setting
    - Variation capture and loss
    - Interpreting principal components
    - Example application

Covariation between variables

- What is covariation and how is it measured quantitatively?
- Covariance and correlation matrices
- Visualizations

City sustainability data

Throughout this week's lecture we'll use the following dataset on the sustainability of U.S. cities:

In [2]:

```
city_sust.head(4)
```

Out[2]:

	GEOID_MSA	Name	Econ_Domain	Social_Domain	Env_Domain	Sustain_Index
0	310M300US10100	Aberdeen, SD Micro Area	0.565264	0.591259	0.444472	1.600995
1	310M300US10140	Aberdeen, WA Micro Area	0.427671	0.520744	0.429274	1.377689
2	310M300US10180	Abilene, TX Metro Area	0.481092	0.496874	0.454192	1.432157
3	310M300US10220	Ada, OK Micro Area	0.466566	0.526206	0.425964	1.418737

For each **M**etropolitan **S**tatistical **A**rea (MSA), a sustainability index is calculated based on economic, social, and environmental indicators (also indices).

$$\text{sustainability index} = \text{economic} + \text{social} + \text{environmental}$$

The domain indices are computed from a large number of development indicator variables. To keep the application simple, we'll work with this derived data instead of the underlying variables.

If you're interested, you can dig deeper on the **Sustainable Development Report website**, which provides detailed data reports related to the U.N.'s 2030 sustainable development goals.

What is covariation?

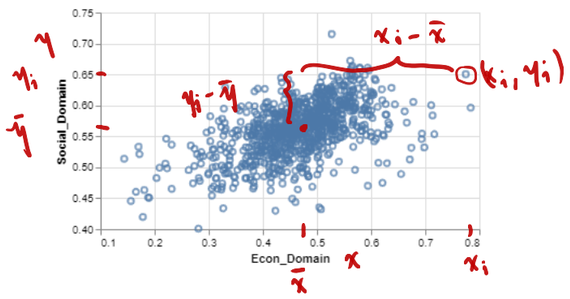
Last week we talked about exploring variation among individual variables: the tendency of values to change between observations.

Covariation refers to ***the tendency of two variables to change together between observations***. Covariation is about relationships.

In [4]:

```
econ_social
```

Out[4]:



The social and economic indices do seem to *vary together*: higher values of the economic index coincide with higher values of the social index.

That's all there is to it. ***You've already thought about covariation, even if you didn't know it!***

How is covariation measured?

Let $(x_1, y_1) \ldots, (x_n, y_n)$ denote n values of two variables, X and Y.

If X and Y tend to vary together, then whenever X is far from its mean, so is Y.

- In other words, *their deviations from typical values coincide*.

This coincidence (or lack thereof) can be captured quantiatively by the (sample) **covariance**:

$$\text{cov}(\text{x}, \text{y}) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

deviation of y from its mean

deviation of x from its mean

Sum over observations

"centering": $x_i - \bar{x}$

In [5]:

```
xy = city_sust.iloc[:, 2:4] # x = econ, y = social
xy_ctr = xy - xy.mean() # center by subtracting column means
xy_ctr.head(3)
```

Out[5]:

	$x_i - \bar{x}$ Econ_Domain	$y_i - \bar{y}$ Social_Domain
0	0.098892	0.029696
1	-0.038701	-0.040819
2	0.014720	-0.064690

$$= \begin{bmatrix} \text{Econ} & \text{Social} \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \vdots & \vdots \end{bmatrix} - \begin{bmatrix} \text{Econ} & \text{Social} \\ \bar{x} & \bar{y} \\ \bar{x} & \bar{y} \\ \bar{x} & \bar{y} \\ \vdots & \vdots \end{bmatrix}$$

Original

Column means

sum of product of deviations

In [6]:

```
xy_cov = xy_ctr.prod(axis = 1).sum()/(len(xy) - 1) # cov(x, y)
xy_cov
```

Out[6]:

0.0019850231634313846

Econ x Social

0.0988 x 0.0297

-0.0387 x -0.0408

⋮

Correlation: standardized covariance

Covariance is a little tricky to interpret. Is 0.00199 large or small?

It is a little more useful to compute the (sample) **correlation**:

$$\mathrm{corr}(\mathrm{x}, \mathrm{y}) = \frac{\mathrm{cov}(\mathrm{x}, \mathrm{y})}{S_x S_y}$$

This is simply ***covariance standardized to [-1, 1]***.

Properties:

- $\mathrm{corr}(\mathrm{x}, \mathrm{x}) = 1$, since any variable's deviations coincide *perfectly* with themselves
- the sign indicates whether X and Y vary together or in opposition
- $\mathrm{corr}(\mathrm{x}, \mathrm{y}) = 1, -1$ are the *strongest possible* correlations
- $\mathrm{corr}(\mathrm{x}, \mathrm{y}) = 0$ is the weakest possible correlation
- moderate to large correlations indicate that X and Y covary
- small correlations *do not indicate* that X and Y don't covary

Correlation of social and economic indices

Standardizing the covariance makes it more interpretable:

In [7]:

```
xy_cov/xy.std().prod() # cov(x, y)/(sx*sy)
```

Out[7]:

```
0.5314906371658936
```

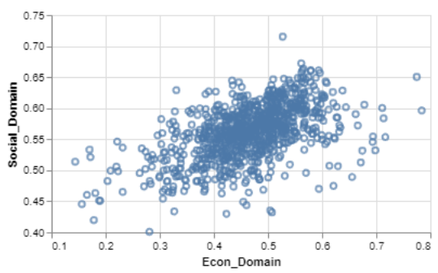
The correlation indicates that the social and economic indices vary together (positive) weakly (moderate magnitude on [0, 1] scale).

This is just a number that quantifies what you already knew from the graphic: there is a positive relationship.

In [8]:

```
econ_social
```

Out[8]:



What is the use of a quantitative measure of covariation?

It helps to have a number so that we can talk precisely about the *strength* of a relationship, and compare relationships quantitatvely.
Consider looking at all pairwise relationships for patterns of covariation:

In [10]:

```
scatter_panel
```

Out[10]:

Is the economic index *more* related to the environmental index or the social index?

Correlation matrix (example)

The pairwise correlations among all three variables can be represented in a simple square matrix:

In [11]:

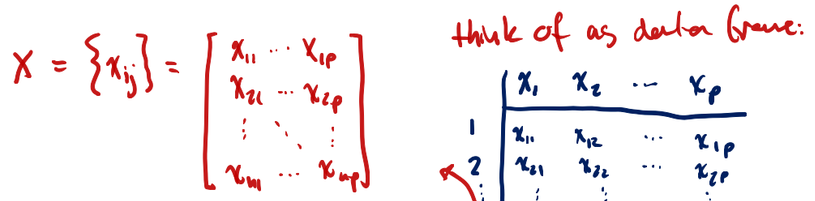
```
x_mx.corr() # correlation matrix in pandas
```

Out[11]:

	Econ_Domain	Social_Domain	Env_Domain
Econ_Domain	1.000000	0.531491	0.011206
Social_Domain	0.531491	1.000000	-0.138674
Env_Domain	0.011206	-0.138674	1.000000

The strongest relationship is between social and economic indices; the weakest is between environmental and economic indices.
Notice the correspondence between this matrix and the scatterplot panel -- there is one number per plot that describes the strenth of the relationship shown visually.

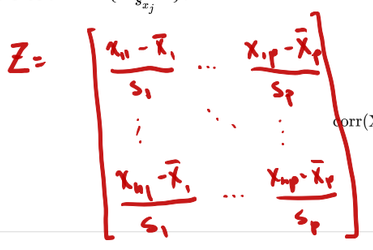
Correlation matrix (definition)



The (sample) correlation matrix can be expressed as a simple matrix product; let $\mathrm{X} \in \mathrm{R}^{n \times p}$ denote the data values for n observations of p variables.

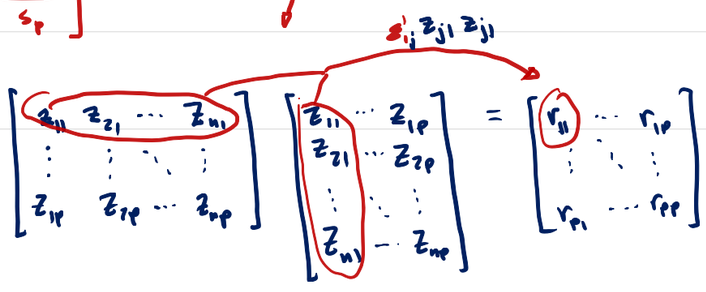
Consider the centered and scaled version of the data: $\mathrm{Z} = \{\frac{x_{ij} - \bar{x}_j}{s_{x_j}}\}$.

The (sample) **correlation matrix** is defined as:



$$\mathrm{corr}(\mathrm{X}) = \frac{1}{n-1}\mathrm{Z}'\mathrm{Z}$$

call this R



In [12]:

```
# correlation matrix 'by hand'
n = len(x_mx) # sample size
z_mx = (x_mx - x_mx.mean())/x_mx.std() # (xi - xbar)/sx
z_mx.transpose().dot(z_mx)/(n - 1) # Z'Z/(n - 1)
```

Out[12]:

	Econ_Domain	Social_Domain	Env_Domain
Econ_Domain	1.000000	0.531491	0.011206
Social_Domain	0.531491	1.000000	-0.138674
Env_Domain	0.011206	-0.138674	1.000000

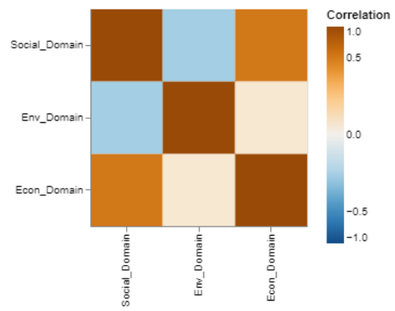
Heatmap visualization

Often it's useful to visualize the correlation matrix as a heatmap.

In [14]:

```
heatmap
```

Out[14]:



- Each cell corresponds to a pair of variables
- Cells are colored acccording to the magnitude of correlation between the pair
- Rows and columns are sorted in order of correlation strength

Notice again the correspondence with the scatterplot panel.

Eigendecomposition

- The eigenvalue problem
- Geometric interpretation
- Computations

Why are we talking about eigendecomposition?

We are going to factor the correlation matrix into components. This is, in essence, PCA.

This is useful for two reasons:

- identifying which variables drive variation and covariation
- reducing correlation structure among many variables to simpler components;
- visualizing multivariate data.

The eigendecomposition provides a means of factoring the matrix.

The eigenvalue problem

Let A be a square $(n \times n)$ matrix.

The **eigenvalue problem** refers to finding nonzero λ and x that satisfy the equation:

$$\mathrm{Ax} = \lambda \mathrm{x}$$

For any such solutions:

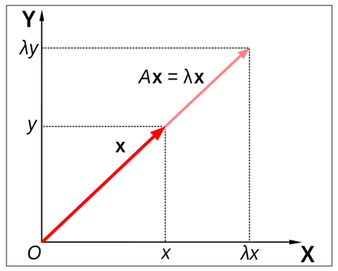
- λ is an **eigenvalue** of A;
- x is an **eigenvector** of A.

Geometry

For a simple example, suppose $n = 2$ and x is an eigenvalue of A. Then:

$$\mathrm{Ax} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{bmatrix} = \begin{bmatrix} \lambda x_1 \\ \lambda x_2 \end{bmatrix} = \lambda \mathrm{x}$$

So the eigenvalue problem equation says that the *linear transformation of* x *by* A is simply a rescaling of x by a factor of λ.



The decomposition

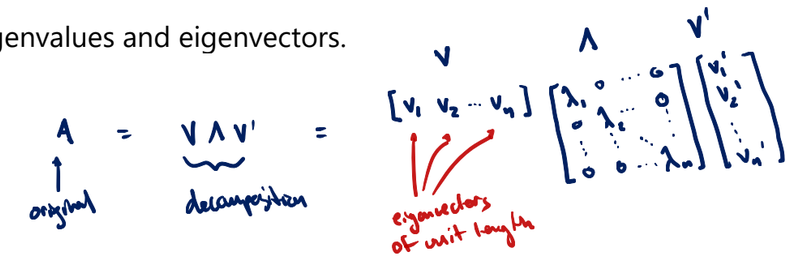
If λ, x are an eigenvalue and eigenvector of A, then so are $c\lambda, c\mathrm{x}$ for any constant c; so usually attention is constrained to solutions such that $\|\mathrm{x}\| = 1$; these are unique to within the sign of x.

The **eigendecomposition** of a square matrix consists in ***finding its unique nonzero eigenvalues and (unit-length) eigenvectors***.

This task is considered a 'decomposition' because the eigenvectors V and eigenvalues Λ satisfy

$$\underset{\text{original matrix}}{\mathrm{A}} = \underset{\text{eigendecomposition}}{\mathrm{V}\Lambda\mathrm{V}'}$$

So the original matrix can be *reconstructed* from the eigenvalues and eigenvectors.



Special case

We will be applying eigendecomposition to correlation matrices. These are of the form $\mathrm{A} = \mathrm{Z}'\mathrm{Z}$.

Such matrices have some special properties, with the consequences:

- $\mathrm{V}'\mathrm{V} = \mathrm{I}$, in other words, the eigenvectors are an orthonormal basis;
- $\lambda_i \geq 0$, in other words, all eigenvalues are nonnegative.

Computations

The eigendecomposition is computed numerically using iterative methods. Luckily, these are very easy to implement:

In [15]:

```
# compute decomposition
decomp = linalg.eig(corr_mx)
decomp
```

Out[15]:

```
(array([0.44788559+0.j, 1.54664121+0.j, 1.0054732 +0.j]),
 array([[ 0.68276586, -0.68571102,  0.2522522 ],
        [-0.70523279, -0.7087523 , -0.01780118],
        [-0.19099079,  0.16574249,  0.96749778]]))
```

In [16]:

```
# eigenvalues
decomp[0]
```

Out[16]:

```
array([0.44788559+0.j, 1.54664121+0.j, 1.0054732 +0.j])
```

In [17]:

```
# eigenvectors
decomp[1]
```

Out[17]:

```
array([[ 0.68276586, -0.68571102,  0.2522522 ],
       [-0.70523279, -0.7087523 , -0.01780118],
       [-0.19099079,  0.16574249,  0.96749778]])
```

Does the decomposition really work?

Let's check that in fact $A = V\Lambda V'$:

In [19]:

```
eigenvecs.dot(eigenvals).dot(eigenvecs.transpose()) # V Lambda V'
```

Out[19]:

	Econ_Domain	Social_Domain	Env_Domain
Econ_Domain	1.000000	0.531491	0.011206
Social_Domain	0.531491	1.000000	-0.138674
Env_Domain	0.011206	-0.138674	1.000000

In [20]:

```
corr_mx # correlation matrix (our 'A')
```

Out[20]:

	Econ_Domain	Social_Domain	Env_Domain
Econ_Domain	1.000000	0.531491	0.011206
Social_Domain	0.531491	1.000000	-0.138674
Env_Domain	0.011206	-0.138674	1.000000

Orthogonality

Let's check also that in fact $\mathrm{V}'\mathrm{V} = \mathrm{I}$:

In [21]:

```
eigenvecs.transpose().dot(eigenvecs) # V'V
```

Out[21]:

	v1	v2	v3
v1	1.000000e+00	-1.510283e-16	1.241330e-17
v2	-1.510283e-16	1.000000e+00	3.849861e-17
v3	1.241330e-17	3.849861e-17	1.000000e+00

The significance of this property is that $\mathrm{v}_1, \mathrm{v}_2, \mathrm{v}_3$ form an *orthonormal basis*.

What's a basis?

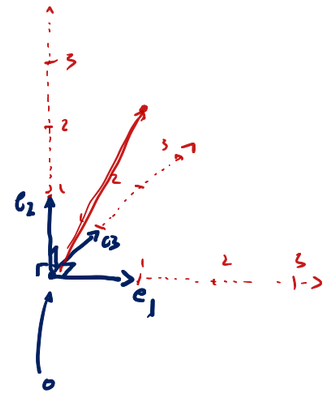
A **basis** in linear algebra is a ***set of vectors that span a linear space***. Think of a basis as a set of axes.

The usual basis for $\mathbb{R}^3$ are the unit vectors:

$$e_1 = (1,0,0) \quad e_2 = (0,1,0) \quad e_3 = (0,0,1)$$

Coordinates in $\mathbb{R}^3$ are usually represented as multiples of these basis vectors:

$$(1,2,1) = 1e_1 + 2e_2 + 1e_3$$



Terminology:

- A basis is *orthogonal* if *all basis vectors are at right angles to one another*.
- A basis is *orthonormal* if *it is orthogonal and basis vectors are of unit length*.

$$e_i^T e_j = 0 \quad \forall j \neq i$$

$$\|e_1\| = \sqrt{1^2 + 0^2 + 0^2} = \sqrt{1^2} = \sqrt{1} = 1$$

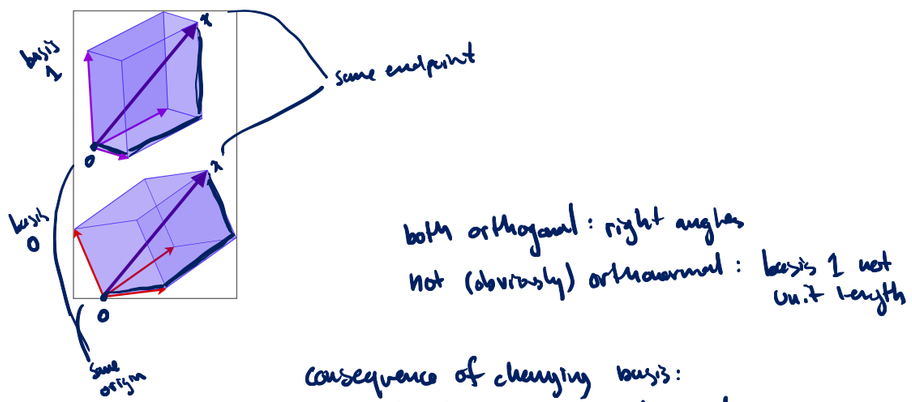
$$\|e_2\| = 1$$

$$\|e_3\| = 1$$

Nonstandard bases

To an extent the usual choice of $\mathrm{e}_1, \mathrm{e}_2, \mathrm{e}_3$ is arbitrary.

It's possible to choose *any* system based on three directions to represent the same point relative to the origin.



Interpreting a change of basis

Different bases are simply different coordinate systems.

It's akin to the possibility of locating the distance to the corner store by different units and relative to different directions. For example:

- half a block to the right from the front door; or
- sixty steps diagonally in a straight line; or
- fifteen steps north, fifty steps east.

Eigendecomposition of the correlation matrix

So, decomposing the correlation matrix yields a basis.

It turns out that when data are represented on that basis, patterns of covariation vanish.

> *Let* $\mathrm{Y} = \mathrm{XV}$*; this is the representation of* X *on the basis given by* V*. Then*
>
> $$\mathrm{Y'Y} = \mathrm{V'X'XV} = \mathrm{V'V\Lambda V'V} = \Lambda$$
>
> *which implies that* $\mathrm{cov}(\mathrm{y}_j, \mathrm{y}_k) = 0$ *since* Λ *is diagonal.*

So the change of basis to V in a sense 'captures' the covariation in the data.

Notice that the change of basis is a linear transformation XV.

This is the essence of PCA: using eigendecomposition to obtain a linear transformation that captures covariation.

Principal components analysis

- What is PCA exactly?
- PCA in the low-dimensional setting
- Variation capture and loss
- Interpreting principal components
- Example application

Principal components analysis

Principal components analysis (PCA) consists in ***finding a linear transformation of one's data that captures covariation***.

Applications are varied, and so are descriptions of just what the method is supposed to do, but the core technique is simple:

1. Compute the eigendecomposition of the correlation matrix.
2. Choose a subset of eigenvectors.
3. Then do other stuff (the 'analysis' part).

PCA terminology:

- Eigenvectors are called 'principal component directions'
- The values of eigenvectors are called 'loadings'
- The projected data consist of the 'principal components'

PCA in the low-dimensional setting

Let's consider what happens if we follow the steps of PCA outlined above with just two variables, say the social index and the economic index, which were the most correlated pair in the example data.

Denote the observations on these two variables by

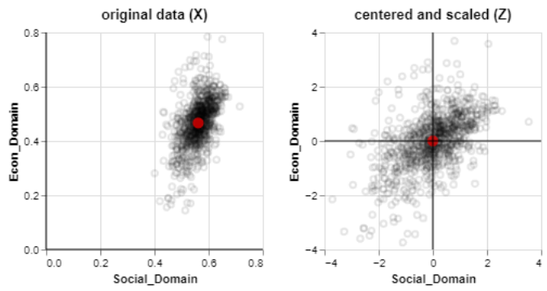
$$\mathrm{X} = [\mathrm{x}_1\ \mathrm{x}_2] \qquad \text{where} \qquad \mathrm{x}_1 = \text{social index and } \mathrm{x}_2 = \text{economic index}$$

To get the correlation matrix, first compute $\mathrm{Z} = \{\frac{x_i - \bar{x}}{s_x}\}$.

In [23]:

```
plot.configure_axis(domain = False)
```

Out[23]:



The red dots indicate the means.

PCA in the low-dimensional setting

Now we'll compute the eigendecomposition. This is pretty much all there is to it.

In [25]:

```
# compute correlation mx
r_mx = z_mx.transpose().dot(z_mx)/(len(z_mx) - 1)
r_mx
```

Out[25]:

	Econ_Domain	Social_Domain
Econ_Domain	1.000000	0.531491
Social_Domain	0.531491	1.000000

In [26]:

```
# decomposition
r_eig = linalg.eig(r_mx.values)

# show PC directions
directions = pd.DataFrame(r_eig[1], columns = ['PC1_Direction', 'PC2_Direction'], index = r_mx.columns)
directions
```

Out[26]:

	PC1_Direction	PC2_Direction
Econ_Domain	0.707107	-0.707107
Social_Domain	0.707107	0.707107

Remember that the principal component directions are just the eigenvectors.

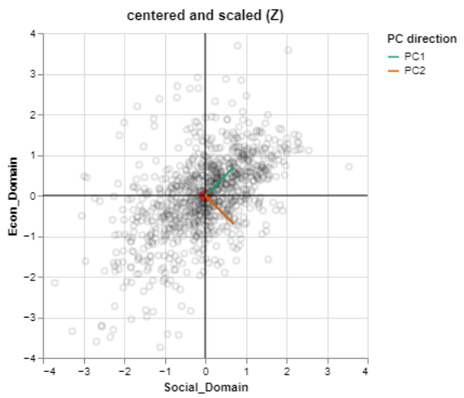
Geometry of PCA

What did we just do? Let's plot the principal component directions on the centered and scaled data.

In [28]:

```
centered_plot + eigenbasis
```

Out[28]:



Do you notice any relationship between the directions and the scatter?

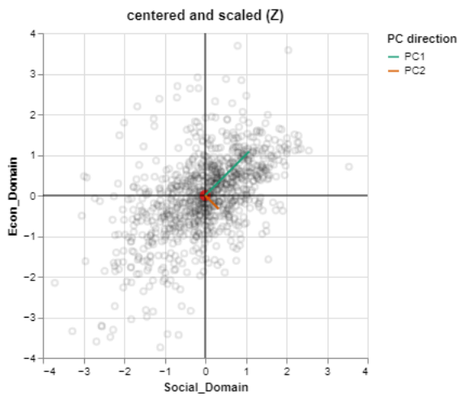
Geometry of PCA

How about now?

In [30]:

```
centered_plot + eigenbasis
```

Out[30]:



The difference here comes from scaling the directions by the corresponding eigenvalues (plotting $\lambda_1 v_1$ and $\lambda_2 v_2$).

The principal component directions are the axes along which the data vary most, and the eigenvalues give the magnitude of that variation.

Geometry of PCA: projection

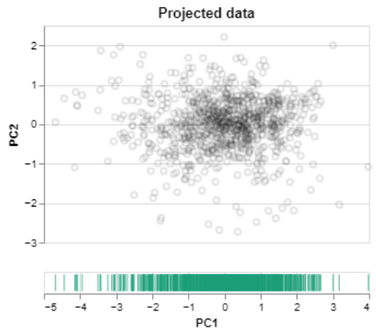
So if we wanted to look at just *one* quantity that retains variation and covariation in the original data, we could:

- project the data onto the new basis
- and look at the values on the axis of the first PC direction (the one with the most variation, *i.e.*, largest λ_i).

In [32]:

```
projection & pc1
```

Out[32]:



If we select just the first principal component to 'do other stuff' with, we lose the variation in PC2. Is this a problem?

Quantifying variation capture/loss

To figure out how much variation/covariation is captured and lost, we need to know how much is available in the first place.

The **total variation in** X is given in terms of the correlation matrix by:

$$\text{total variation} = \det\left(\mathrm{R}\right) = \sum_{i=1}^{p} \lambda_i$$

Now let $\mathrm{y}_j = \mathrm{Z}\mathrm{v}_j$ be the jth principal component. Its variance is:

$$\frac{1}{n-1}\mathrm{y}_j'\mathrm{y} = \frac{1}{n-1}\mathrm{v}_j'\mathrm{Z}'\mathrm{Z}\mathrm{v}_j = \mathrm{v}_j'\mathrm{V}'\Lambda\mathrm{V}\mathrm{v}_j = \mathrm{e}_j'\Lambda\mathrm{e}_j = \lambda_j$$

Quantifying variation capture/loss

So the total variation is the sum of the eigenvalues, and the variance of each PC is the corresponding eigenvalue.

We can therefore define the **proportion of total variation explained** by the jth principal component as:

$$\frac{\lambda_j}{\sum_{j=1}^{p} \lambda_j}$$

This is sometimes also called the *variance ratio* for the jth principal component.

So in our example:

In [33]:

```
eigenvalues = r_eig[0].real # store eigenvalues as real array
eigenvalues[0]/eigenvalues.sum() # variance ratio
```

Out[33]:

```
0.765745318582947
```

The first principal component captures 77% of total variation.

Interpreting principal components

So we have obtained a single derived variable that captures 3/4 of all variation and covariation in both variables. But what is this?

The values of the first principal component (green ticks) are given by:

$$\mathrm{PC1}_i = \mathrm{x}_i' \mathrm{v}_1 = 0.7071 \times \mathrm{economic}_i + 0.7071 \times \mathrm{social}_i$$

So the principal component is a linear combination of the underlying variables.

Those coefficients are known as **loadings**: they determine the ***relative weights*** by which the variables contribute to the principal component.

In this case, the PC1 loadings are equal; so this principal component is simply the average of the social and economic indices.

Example application

Now that we've reviewed the basic technique, we can consider cases with a larger number of variables.

This is really where PCA is most useful.

- It can help answer the question: which variables are driving variation in the data?
- It can help reduce data dimensions by finding combinations of variables that preserve variation.
- It can provide a means of visualizing high-dimensional data.

So, let's look at a different example: world development indicators.

In [34]:

```
wdi = pd.read_csv('data/wdi-data.csv').iloc[:, 2:].set_index('country')
wdi.head(3)
```

Out[34]:

country	inequality_adjusted_life	inequality_adjusted_education	maternal_mortality	parliament_pct_women	labor_participation_women	labor_participation_men	total_pop	urban_pct_pop	pop_under5	pop_15to64	...	total_unemployment	youth_unemployment	prison_5yr	trade	foreign_investment	net_migration	imm
Norway	0.931	0.908	2	40.8	60.4	67.2	5.4	82.6	0.055556	0.648148	...	3.3	9.3	80	72.1	0.4	5.3	
Ireland	0.926	0.892	5	24.3	56.0	68.4	4.9	63.4	0.061224	0.653061	...	4.9	13.1	82	239.2	-20.4	4.9	
Switzerland	0.947	0.883	5	38.6	62.9	73.8	8.6	73.8	0.046512	0.662791	...	4.6	7.4	77	119.4	-2.6	6.1	

3 rows × 31 columns

Computation via sklearn.decomposition.PCA

Luckily, `sklearn` contains an easy-to-use implementation that saves the trouble of going through all the calculations we did before 'by hand'.

In [35]:

```
from sklearn.decomposition import PCA
```

The implementation simply requires centering and scaling the data first.

In [36]:

```
# center and scale
wdi_ctr = (wdi - wdi.mean())/wdi.std()

# compute principal components
pca = PCA(31)
pca.fit(wdi_ctr)
```

Out[36]:

```
PCA(n_components=31)
```

Variance ratios

Selecting the number of principal components to use is somewhat subjective, but always based on the variance ratios.

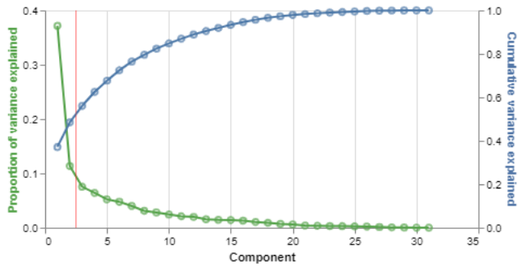
- how many PCs capture substantial variation individually before the variance ratio 'tails off'?
- how many PCs are needed to capture a certain proportion of the total variation?

This can be assessed by plotting the variance ratios and their cumulative sum:

In [38]:

```
variance_plot.properties(height = 200, width = 400)
```

Out[38]:



In this case, the variance ratios drop off after 2 components. These first two capture about half of total variation in the data.

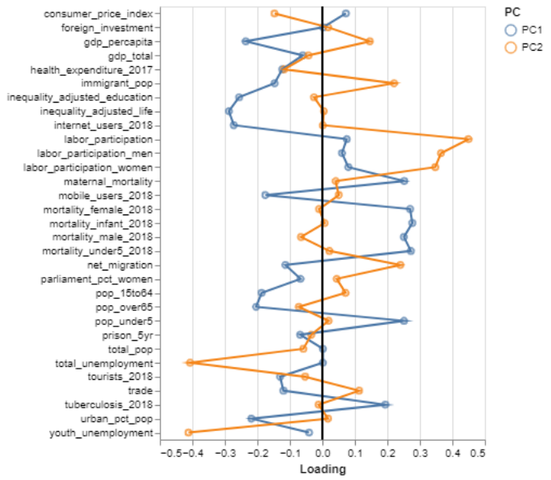
Loading plots

Examining the loadings can help address the question: *which variables drive variation in the data?*

In [40]:

```
loading_plot.properties(width = 300, height = 400)
```

Out[40]:



Visualization

Finally, we might want to use the first two principal components to visualize the data. Often it's helpful to merge the principal components with the original data and apply visualization techniques you already know to search for interesting patterns.

In [42]:

```
scatter + text
```

Out[42]: