Decompositions (Matrix Factorizations)

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.decomposition (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.decomposition)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA, SparsePCA, NMF
from sklearn.metrics import explained_variance_score
from IPython.core.display import display, HTML
```

Imagine the following bike counters, where everybody is biking to work, from different start places:

```
 >-----A----B-
              \
              D----->WORK
             /
 >--------C--
```

In [2]:
```
plt.rcParams["font.size"] = 14
```

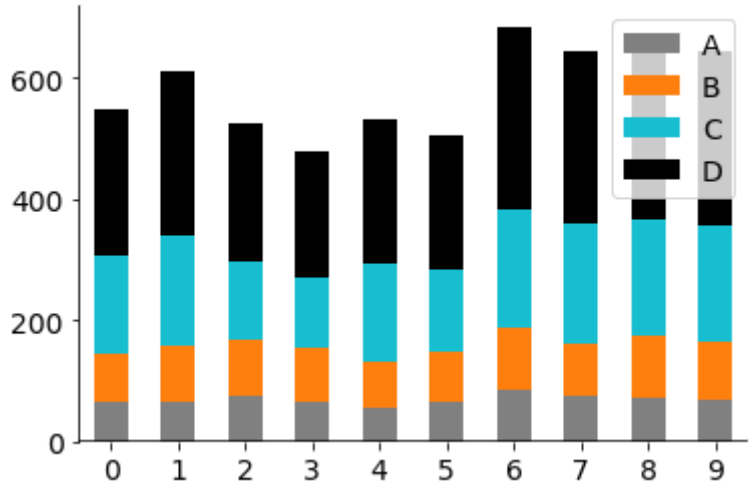
In [3]:
```
df = pd.DataFrame()
df["A"] = np.random.randint(50,100,size=40)
df["B"] = df["A"] + np.random.randint(10,30,size=40)
df["C"] = np.random.randint(100, 200,size=40)
df["D"] = df["B"] + df["C"] + np.random.randint(0,5,size=40)
df.head()
```

Out[3]:

	A	B	C	D
0	67	77	162	243
1	66	91	182	273
2	75	92	131	226
3	67	88	116	207
4	56	77	160	239

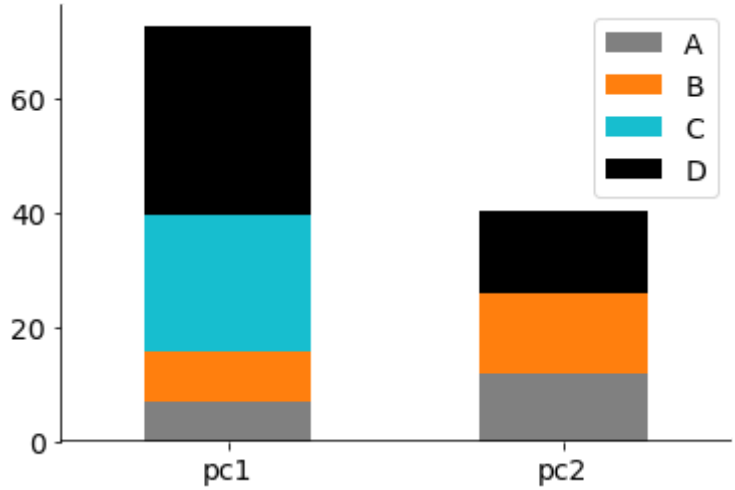
In [4]:
```
def stack(df):
    pos = df.copy()
    neg = df.copy()
    pos[pos < 0] = 0
    neg[neg > 0] = 0
    colors = ["0.5", "tab:orange", "tab:cyan", "black"]
    ax = pos.plot.bar(stacked=True, color=colors)
    neg.plot.bar(stacked=True, ax=ax, legend=False, color=colors)
    ax.plot(ax.get_xlim(), (0,0), "k")
    for s in ["top", "right", "bottom"]:
        ax.spines[s].set_visible(False)
    plt.xticks(rotation=0)
    return ax.get_figure()

stack(df.iloc[:10])
```

```
None
```



In [7]:
```
def try_decomp(f):
    display(HTML("<h1>" + type(f).__name__ + "</h1>"))

    # step 1: compute the principal components
    f.fit(df)
    pc = [f"pc{i+1}" for i in range(len(f.components_))]
    components = pd.DataFrame(f.components_, columns=df.columns, index=p
c)
    display(stack(components))
    print("\ncomponents:")
    display(components)

    # step 2: compute the compressed data
    data = f.transform(df)
    data = pd.DataFrame(data, columns=pc)
    print("\ndata:")
    display(data.head())

    # step 3: try to reconstruct the original data -- how close is it?
    orig_data = data.values @ f.components_
    if hasattr(f, "mean_"):
        orig_data += f.mean_
        mean = pd.DataFrame(f.mean_.reshape(1,-1), columns=df.columns)
        print("\nmean:")
        display(mean)
    orig_data = pd.DataFrame(orig_data, columns=df.columns)
    print("\nreconstructed:")
    display(orig_data.head())
    print("\nSCORE: " + str(explained_variance_score(df, orig_data)))
    plt.close()

ncomps = 2
try_decomp(NMF(ncomps))
try_decomp(PCA(ncomps))
try_decomp(SparsePCA(ncomps))
```

NMF



components:

	A	B	C	D
pc1	6.816393	8.969201	23.904579	33.002298
pc2	11.836349	14.007520	0.000000	14.329780

data:

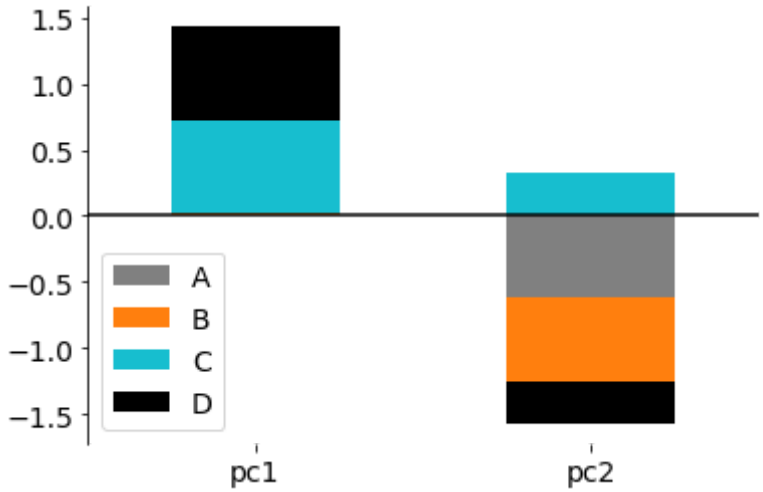
	pc1	pc2
0	6.760446	1.406496
1	7.633300	1.445804
2	5.489163	3.113671
3	4.906090	3.063926
4	6.739412	1.086675

reconstructed:

	A	B	C	D
0	62.729639	80.337320	161.605616	243.265036
1	69.144612	88.716723	182.470822	272.634490
2	74.270785	92.848206	131.216123	225.773199
3	69.707537	86.921711	117.278009	205.817624
4	58.800747	75.668759	161.102805	237.987896

SCORE: 0.9810398928768552

PCA



components:

	A	B	C	D
pc1	0.012027	0.006087	0.706794	0.707291
pc2	-0.613439	-0.641638	0.333564	-0.317377

data:

	pc1	pc2
0	-4.400707	28.002407
1	31.027086	16.782888
2	-38.147753	8.525246
3	-62.308756	17.026013
4	-8.775757	35.352618

mean:

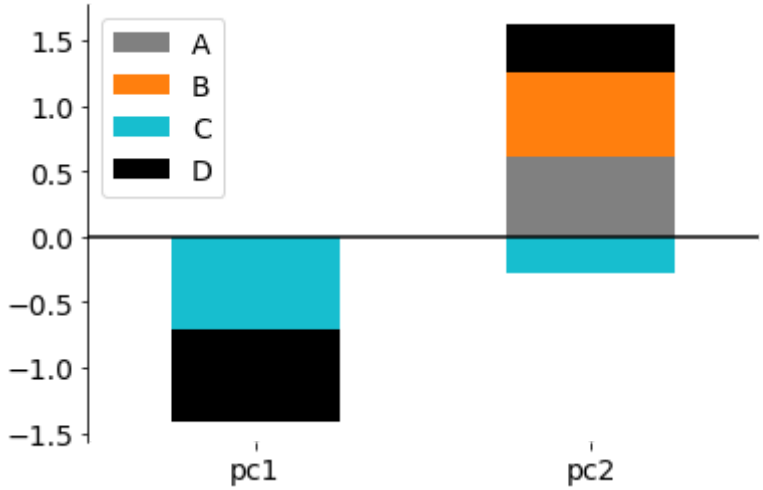
	A	B	C	D
0	78.775	99.275	154.875	255.95

reconstructed:

	A	B	C	D
0	61.544297	81.280807	161.105201	243.950105
1	68.852888	88.695315	182.402936	272.568668
2	73.086467	93.572684	130.756100	226.262734
3	67.581174	87.971209	116.514790	206.475936
4	56.982769	76.538003	160.464705	238.522885

SCORE: 0.9841137136409097

SparsePCA



components:

	A	B	C	D
pc1	0.000000	0.000000	-0.711674	-0.702510
pc2	0.614094	0.642218	-0.279799	0.363534

data:

	pc1	pc2
0	2.437168	-27.822356
1	-31.873621	-16.182114
2	37.152533	-9.017576
3	60.447670	-17.810827
4	6.218487	-35.185558

mean:

	A	B	C	D
0	78.775	99.275	154.875	255.95

reconstructed:

	A	B	C	D
0	61.689453	81.406986	160.925213	244.123499
1	68.837658	88.882557	182.086362	272.458801
2	73.237359	93.483751	130.957635	226.571771
3	67.837475	87.836569	116.839448	207.010055
4	57.167753	76.678206	160.294367	238.790311

```
SCORE: 0.9839917630581557
```

In [ ]: