In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

The following loads a dataset about customer credit card activity over a six-month period.

In [2]:
```
credit = pd.read_csv("credit_cards.csv").dropna()
credit
```

Out[2]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	I
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
4	C10005	817.714335	1.000000	16.00	16.00	
5	C10006	1809.828751	1.000000	1333.28	0.00	
...	...	...	...	...	...	
8943	C19184	5.871712	0.500000	20.90	20.90	
8945	C19186	28.493517	1.000000	291.12	0.00	
8947	C19188	23.398673	0.833333	144.40	0.00	
8948	C19189	13.457564	0.833333	0.00	0.00	
8949	C19190	372.708075	0.666667	1093.25	1093.25	

8636 rows × 18 columns

The *CUST_ID* column is artificial and of no help in clustering, so we drop it to get the feature matrix.

In [3]:
```
X = credit.drop("CUST_ID", axis=1)
```

1 (6.3)

Fit `X` to a pipeline called `km` with a standardization preprocessor and k-means with 3 clusters, 2 initializations, and random state equal to 302.

In [4]:
```
km = None   # should be a Pipeline

### BEGIN SOLUTION
from sklearn.cluster import KMeans
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

```
km = KMeans(n_clusters=3, n_init=2, random_state=302)
km = make_pipeline( StandardScaler(), km )
km.fit(X)
### END SOLUTION

print("Cluster centers found at:")
print(km[1].cluster_centers_)
```

```
Cluster centers found at:
[[ 1.14056882  0.31134389 -0.29762615 -0.21310416 -0.3121167   1.35759403
  -0.66043983 -0.31853207 -0.56615189  1.53721067  1.31942536 -0.37695292
   0.58969438  0.43652009  0.38854136 -0.42295051 -0.14206443]
 [ 0.31080745  0.42016266  1.51262607  1.26431863  1.25234464 -0.2473708
   1.13362075  1.53733018  0.95994641 -0.36184534 -0.25286111  1.6716924
   0.88991238  0.82619997  0.17044902  0.46577825  0.29528903]
 [-0.37104107 -0.17061724 -0.23270916 -0.20410109 -0.17499408 -0.31400108
  -0.05675817 -0.23219413 -0.04620849 -0.33864264 -0.30259972 -0.24426246
  -0.34261328 -0.2882407  -0.13974836  0.01745257 -0.02284524]]
```

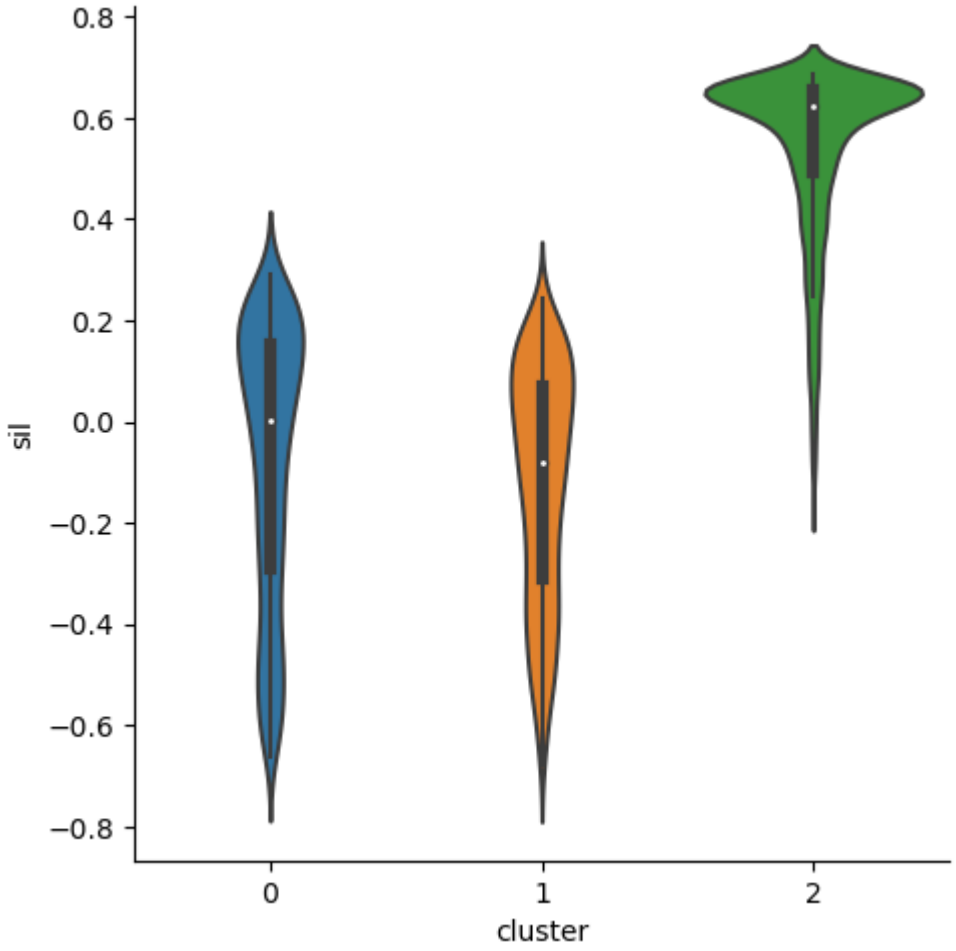
In [5]:
```
# TESTS
from sklearn.pipeline import Pipeline
assert type(km) == Pipeline
assert km[1].random_state == 302
assert np.all( np.isclose( km[1].cluster_centers_[:,0], [1.14056882, 0.31080
print("OK")
```

```
OK
```

2 (6.3)

Make a violin plot showing the silhouette values for `km` by cluster for all the samples. (You should see one decent cluster with mostly positive values and two fairly bad clusters.)

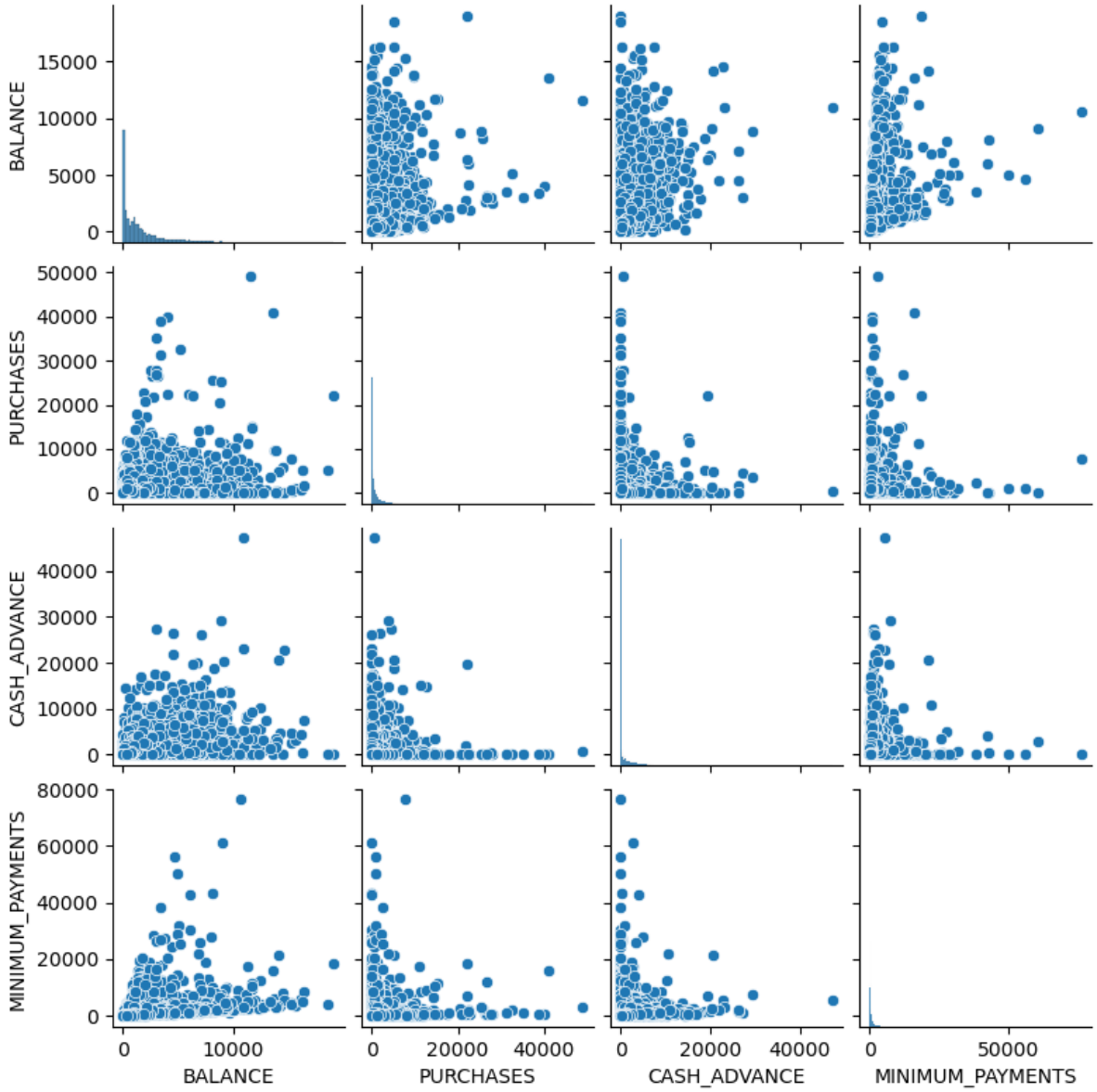
In [6]:
```
# Result is a plot

### BEGIN SOLUTION
from sklearn.metrics import silhouette_samples

y = km[1].labels_
credit["cluster"] = y
credit["sil"] = silhouette_samples(X, y)
sns.catplot(data=credit, x="cluster", y="sil", kind="violin");
### END SOLUTION
```



3

One reason for the poor clustering performance above is that many of the columns of `X` are extremely left-skewed, as you can see along the diagonal of a pairwise plot:

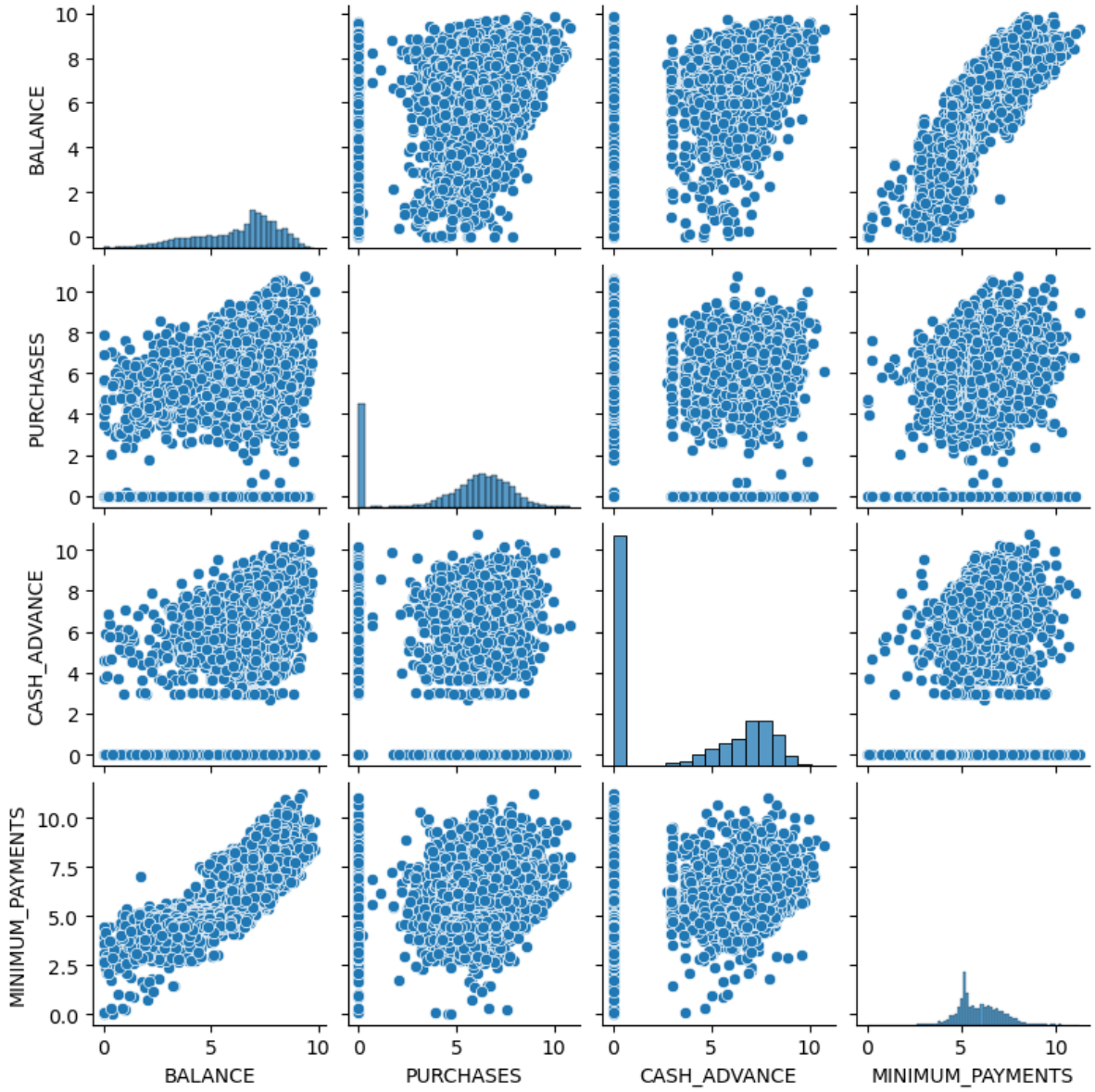
In [7]:
```
sns.pairplot(
    X[["BALANCE", "PURCHASES", "CASH_ADVANCE", "MINIMUM_PAYMENTS"]],
    height=2
);
```



Since the data values are nonnegative, we can combat the skewness by transforming the features according to $$ f(x) = \log(1 + x). $$ Let `XL` be a new frame defined by applying f to each column of `X`. (Use `np.log` for the log function.)

The new plot at the end of this cell should show more-symmetric distributions.

In [8]:
```
XL = None

### BEGIN SOLUTION
XL = X.transform(lambda x: np.log(x+1))
### END SOLUTION

sns.pairplot(
    XL[["BALANCE", "PURCHASES", "CASH_ADVANCE", "MINIMUM_PAYMENTS"]],
    height=2
);
```



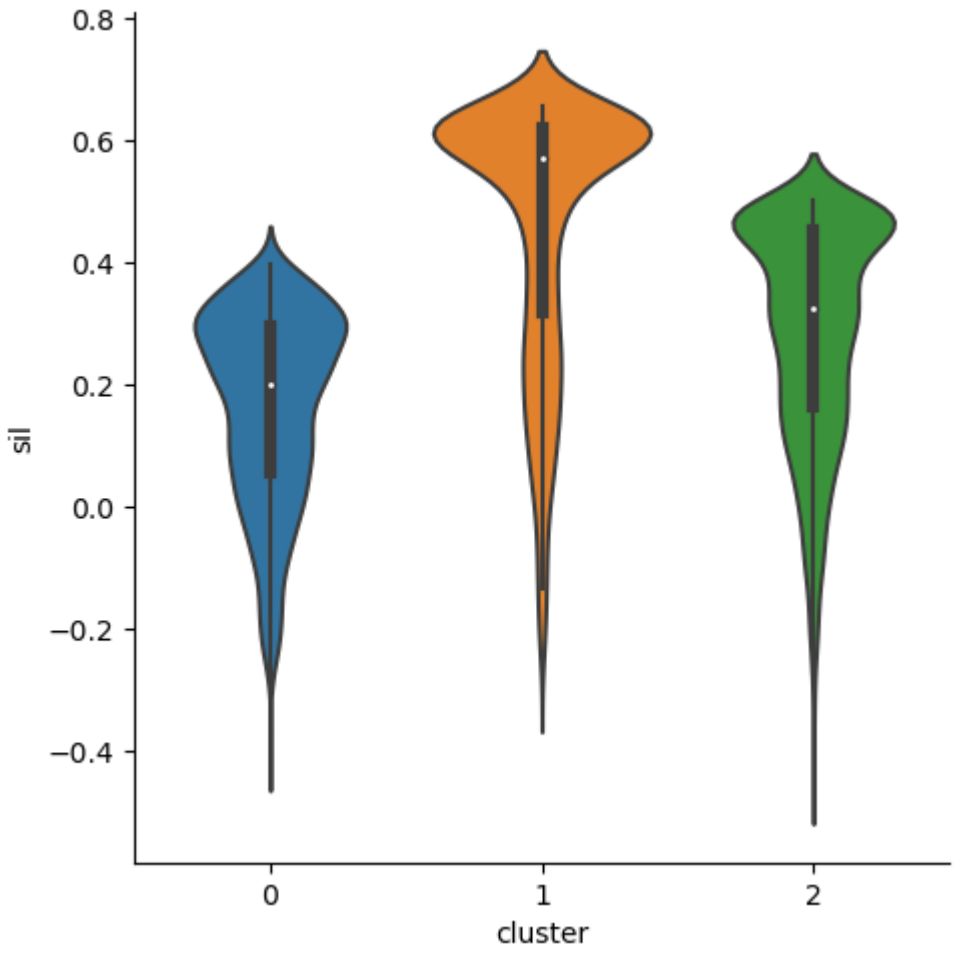
In [9]:
```
# TESTS
assert np.isclose(XL["BALANCE"].mean(), 6.265737)
assert np.isclose(XL["PURCHASES_FREQUENCY"].median(), 0.405465)
assert set(X.columns) == set(XL.columns)
### BEGIN HIDDEN TESTS
assert np.isclose(XL["CASH_ADVANCE"].mean(), 3.349135)
assert np.isclose(XL["TENURE"].mean(), 2.521363)
### END HIDDEN TESTS
print("OK")
```

```
OK
```

4 (6.3)

Repeat steps 1 and 2 for `XL` in place of `X`. (Your plot should show one good cluster and two that are just OK.)

In [10]:
```
# Result should be a plot
km = None   # should be a Pipeline

### BEGIN SOLUTION
km = KMeans(n_clusters=3, n_init=2, random_state=302)
km = make_pipeline( StandardScaler(), km )
km.fit(XL)
y = km[1].labels_
credit["cluster"] = y
credit["sil"] = silhouette_samples(XL, y)
sns.catplot(data=credit, x="cluster", y="sil", kind="violin");
### END SOLUTION
```



5 (6.3)

For each value $k=2,3,\dots,7$, fit `XL` to a pipeline with standardization and k-means clustering with 2 initializations and random state 19716. Compute and record the mean silhouette score for the fit.

Make a series indexed by k called *results* for the mean silhouette scores. (The results should indicate that there is no justification for going past $k=2$.)

In [11]:
```
results = None

### BEGIN SOLUTION
from sklearn.metrics import silhouette_score
results = []
kvals = range(2,8)
for k in kvals:
    km = make_pipeline(StandardScaler(),
                       KMeans(n_clusters=k, n_init=2, random_state=19716) )
    km.fit(XL)
    y = km.predict(XL)
    sil = silhouette_score(XL, y)
    results.append(sil)

results = pd.Series(results, index=kvals)
### END SOLUTION

print(results)
```

```
2    0.361875
3    0.297828
4    0.270107
5    0.304573
6    0.295862
7    0.303840
dtype: float64
```

In [12]:
```
# TESTS
assert type(results) == pd.Series
assert np.all( results[2] > results.loc[3:8] )
### BEGIN HIDDEN TESTS
assert np.isclose( results[2], 0.3618745672339 )
assert np.isclose( results[7], 0.303839852 )
### END HIDDEN TESTS
print("OK")
```

```
OK
```