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

We load a dataset from voteview.com, which tabulates the roll-call vote of every session of the U.S. Congress. Here, we load all of the votes cast by senators between the 91st and 111th sessions of Congress.

In [2]:
```
senate = pd.read_csv("senate_votes.csv")
senate
```

Out[2]:

	congress	rollnumber	icpsr	cast_code
0	91	1	52	1
1	91	1	142	6
2	91	1	165	6
3	91	1	645	6
4	91	1	688	6
...	...	...	...	...
1703219	111	696	49901	1
1703220	111	696	49903	1
1703221	111	696	94659	1
1703222	111	696	94910	1
1703223	111	696	99911	1

1703224 rows × 4 columns

The `rollnumber` is a serial number of the bill or resolution that was voted on, and `icpsr` is a unique identifier for everyone who has ever served in Congress.

Without going into parliamentary details, there are three codes that mean a "yes" vote and three that mean "no." We will encode these as +1 and -1, respectively, and record anything else as a 0.

In [3]:
```
def vote_type(v):
    if v in [1,2,3]: return 1
    elif v in [4,5,6]: return -1
    else: return 0

senate["vote"] = senate["cast_code"].apply(vote_type)
senate.head(6)
```

Out[3]:

	congress	rollnumber	icpsr	cast_code	vote
0	91	1	52	1	1
1	91	1	142	6	-1
2	91	1	165	6	-1
3	91	1	645	6	-1
4	91	1	688	6	-1
5	91	1	823	1	1

In addition to the voting records, we will also access to the party affiliations of the individual senators. This information is loaded from a separate file here.

In [4]:
```
members = pd.read_csv("HSall_members.csv").set_index("icpsr")
members.tail(6)
```

Out[4]:

icpsr	congress	chamber	state_icpsr	district_code	state_abbrev	party_code	occupancy
20146	117	Senate	56	0.0	WV	200	NaN
40915	117	Senate	56	0.0	WV	100	NaN
29940	117	Senate	25	0.0	WI	100	NaN
41111	117	Senate	25	0.0	WI	200	NaN
20953	117	Senate	68	0.0	WY	200	NaN
40707	117	Senate	68	0.0	WY	200	NaN

6 rows × 21 columns

There have been over 50 political parties for senators over U.S. history, as encoded by `party_code` above. For modern times, we want to identify the two major parties and label all the others as independent.

In [5]:
```
# First, set all possible codes to "Ind"
party_name = pd.Series("Ind",index=members["party_code"].unique())
# Now set the two we really care about
party_name[100] = "Dem"
party_name[200] = "Rep"

members["party"] = members["party_code"].replace(party_name)

print(members["party"].value_counts())
```

```
Dem    23147
Rep    19596
Ind     7192
Name: party, dtype: int64
```

We now write a function that returns the data we need for a single session.

The first output is a table having the senator ID as the index, the individual roll calls as the columns, and values that are the individual votes. Any senator with a `NaN` in the record did not serve the full session, and their row is dropped. (Details: This requires a fancy reshaping operation called a `pivot`.)

The second output is the list of party affiliations for the senators in the table.

In [6]:
```
def session(number, senate, members):
    idx = senate["congress"] == number
    votes = senate.loc[idx].pivot(index="icpsr",columns="rollnumber",values=
    idx = members["congress"] == number
    party = members.loc[idx,"party"][votes.index]
    # If more than one party affiliation, use only the last one:
    party = party.loc[np.bitwise_not(party.index.duplicated(keep="last"))]
    return votes, party
```

The two ouputs from `session` are effectively the feature matrix and the party labels for the indicated session. So, for example, in the 100th Congress we have

In [7]:
```
X100, y100 = session(100, senate, members)
print("party counts:")
print(y100.value_counts())
print("\nfeature matrix:")
X100
```

```
party counts:
Dem    54
Rep    46
Name: party, dtype: int64

feature matrix:
```

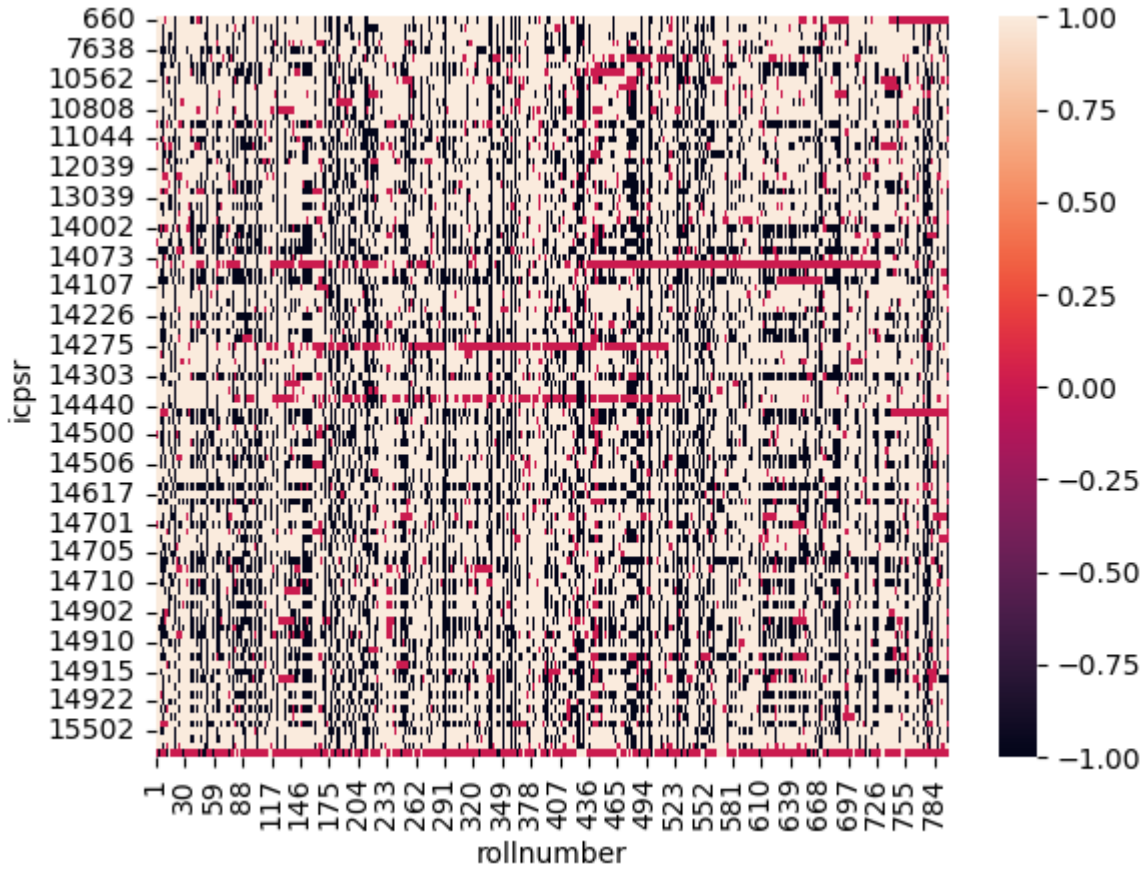
Out[7]:

rollnumber	1	2	3	4	5	6	7	8	9	10	...	790	791	792	793	794
icpsr																
660	1.0	1.0	-1.0	1.0	1.0	1.0	-1.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0
1252	1.0	1.0	-1.0	1.0	1.0	1.0	-1.0	-1.0	1.0	1.0	...	-1.0	1.0	1.0	1.0	1.0
1366	1.0	1.0	-1.0	1.0	1.0	1.0	-1.0	-1.0	1.0	1.0	...	1.0	1.0	1.0	1.0	1.0
4812	1.0	1.0	-1.0	1.0	1.0	1.0	-1.0	-1.0	1.0	1.0	...	-1.0	1.0	1.0	-1.0	1.0
7638	1.0	1.0	-1.0	1.0	1.0	-1.0	-1.0	-1.0	1.0	-1.0	...	-1.0	1.0	-1.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15501	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.0	1.0	1.0	1.0	1.0
15502	1.0	1.0	-1.0	1.0	1.0	-1.0	-1.0	-1.0	1.0	1.0	...	-1.0	1.0	1.0	1.0	1.0
15503	1.0	1.0	-1.0	1.0	1.0	1.0	-1.0	1.0	1.0	1.0	...	1.0	1.0	1.0	1.0	1.0
15504	1.0	1.0	-1.0	1.0	1.0	1.0	-1.0	-1.0	1.0	1.0	...	1.0	1.0	0.0	0.0	0.0
99907	0.0	0.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

100 rows × 799 columns

It's interesting to view this feature matrix as a heat map.

In [8]:
```
sns.heatmap(X100);
```



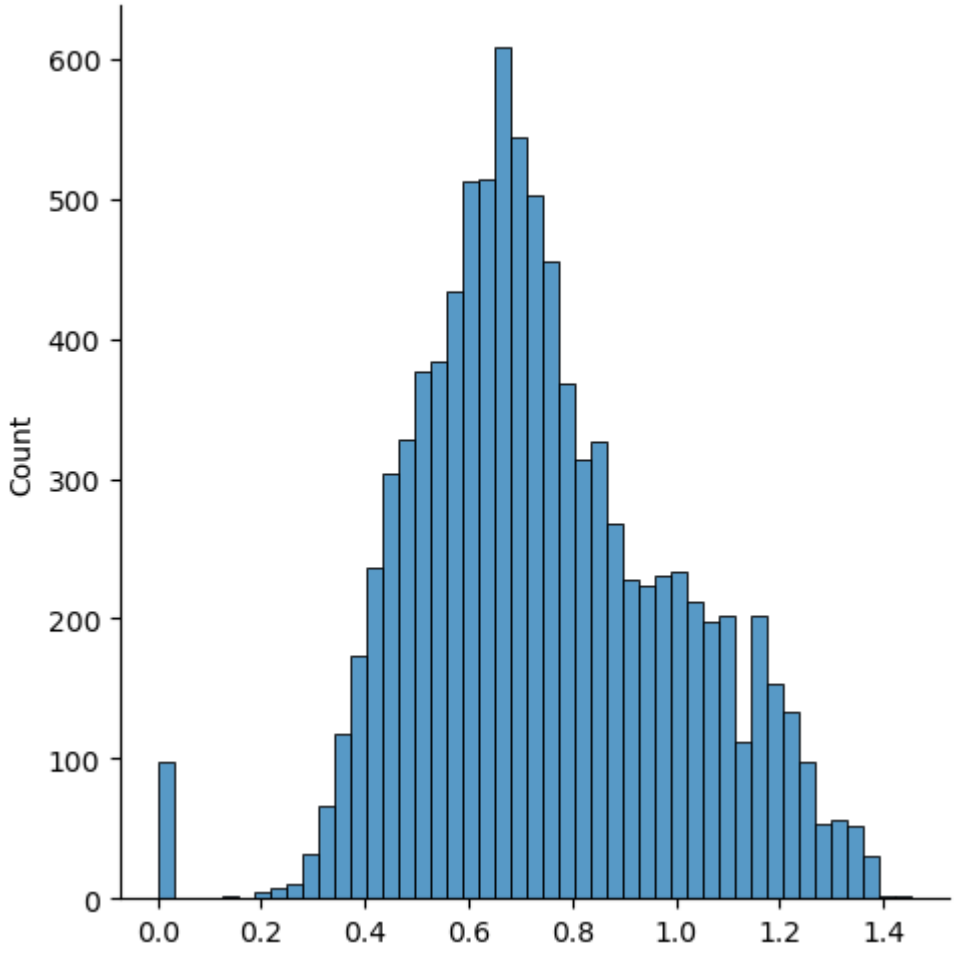
Each row is the voting record of a senator. Remember that the only values possible are -1, 0, and 1. You can definitely see some dominant motifs in the rows that imply party discipline. For the rest of the lab, you will investigate the extent of polarization in the Senate over time.

1 (6.1)

For the 95th Congress, compute the distance matrix using the cosine norm. Make a histogram showing the distribution of all the distances. (Hint: use `D.flatten()` to transform the distance matrix `D` into a vector for the histogram.)

In [9]:
```
# Result is a plot

### BEGIN SOLUTION
from sklearn.metrics import pairwise_distances
X95, y95 = session(95, senate, members)
dist95  = pairwise_distances(X95, metric="cosine")
sns.displot(dist95.flatten());
### END SOLUTION
```



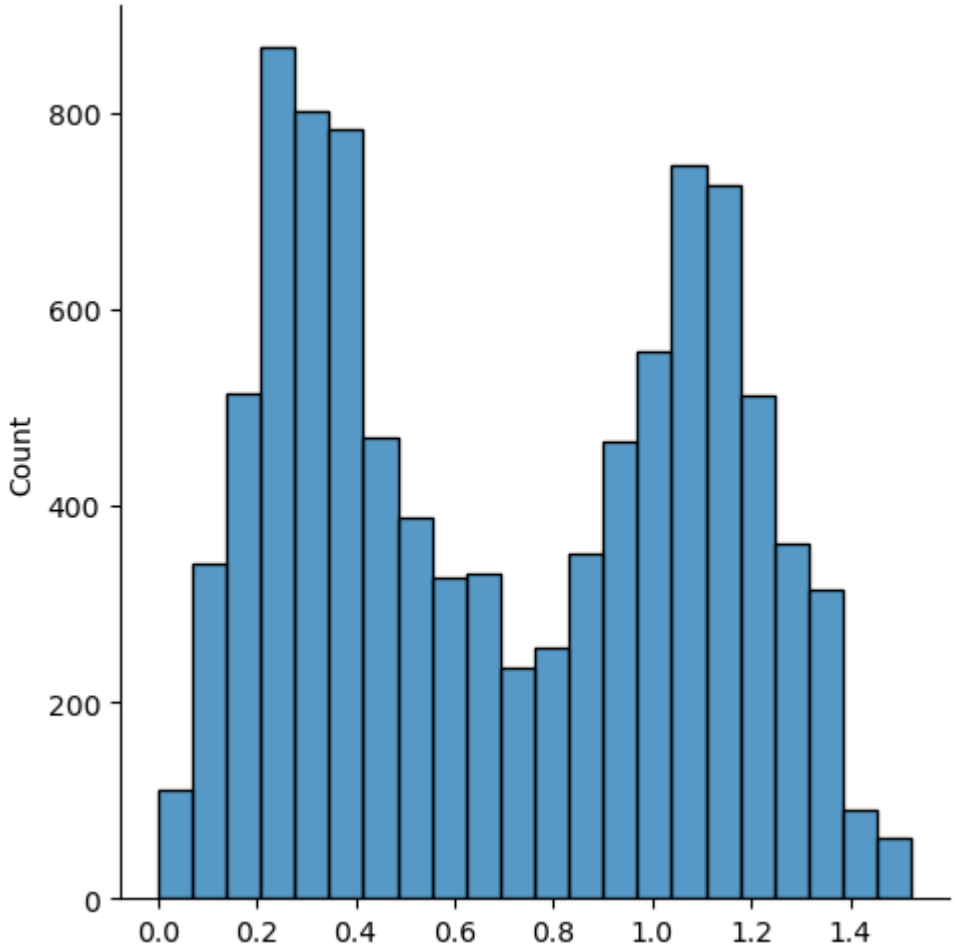
(Your histogram should have a single peak at a distance value a little less than 0.7. There are also 100 instances of distance zero that come from the diagonal of the matrix.)

2 (6.1)

Repeat step 1 for the 110th session of Congress.

In [10]:
```
# Result is a plot

### BEGIN SOLUTION
from sklearn.metrics import pairwise_distances
X110, y110 = session(110, senate, members)
D110  = pairwise_distances(X110, metric="cosine")
sns.displot(D110.flatten());
### END SOLUTION
```



(You should see two distinct peaks in this distribution, which is suggestive of increased polarization.)

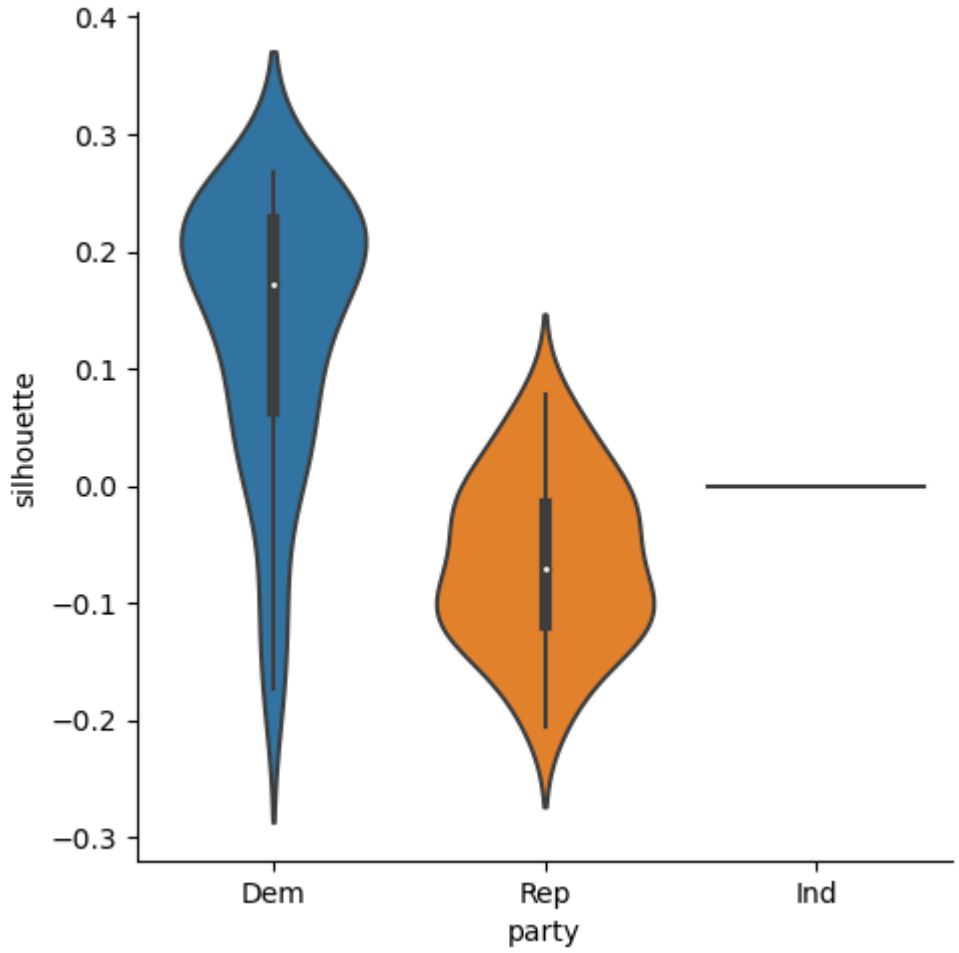
3 (6.2)

Party affiliations effectively define a reference clustering for each session.

For the 95th Congress, make a violin plot showing the distribution of silhouette values for each cluster as defined by party affiliation. (You want one violin per party, even though there is only one *Ind* senator.)

In [11]:
```
# Result is a plot

### BEGIN SOLUTION
from sklearn.metrics import silhouette_samples
sil95 = silhouette_samples(X95,y95)
cluster95 = pd.DataFrame({"silhouette":sil95, "party":y95})
sns.catplot(data=cluster95, x="party", y="silhouette", kind="violin");
### END SOLUTION
```



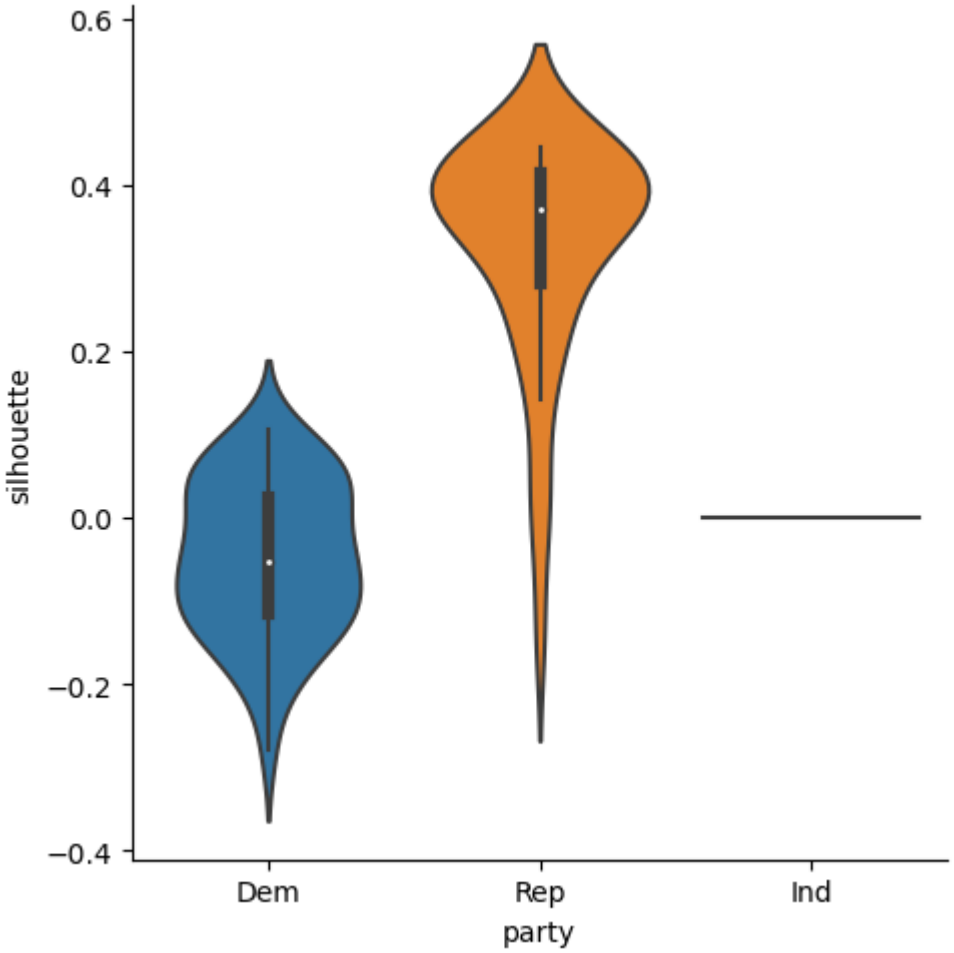
4 (6.2)

Repeat step 3 for the 110th Congress.

In [12]:
```
# Result is a plot

### BEGIN SOLUTION
```

```
sil110 = silhouette_samples(X110, y110)
cluster110 = pd.DataFrame({"silhouette":sil110, "party":y110})
sns.catplot(data=cluster110, x="party", y="silhouette", kind="violin");
### END SOLUTION
```



(Both violin plots show one major party that is well-clustered and another that is not. Thus, we should probably track the parties separately rather than averaging them together. Also, due to some long, asymmetric tails in the value distributions, we will prefer medians to means when scoring the silhouettes.)

5 (6.4)

Using agglomerative clustering with average linkage and 2 clusters, fit the voting records from the 95th Congress. Compute the adjusted Rand index with respect to the reference clustering implied by the party affiliations.

In [13]:
```
ARI = None
agg = None  # should be of type AgglomerativeClustering

### BEGIN SOLUTION
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score
```

```
agg = AgglomerativeClustering(n_clusters=2, linkage="average")
X, y = session(95, senate, members)
agg.fit(X)
y_hat = agg.labels_
ARI = adjusted_rand_score(y, y_hat)
### END SOLUTION

print(f"Adjusted Rand index for 95th session is {ARI:.4f}")
```

```
Adjusted Rand index for 95th session is 0.1985
```

In [14]:
```
# TESTS
assert str(type(agg)) == "<class 'sklearn.cluster._agglomerative.Agglomerati
assert agg.linkage == "average"
assert 0.17 < ARI < 0.20
### BEGIN HIDDEN TESTS
assert np.isclose( ARI, 0.198544703147 )
assert np.all( agg.labels_[:10] == np.array([1, 1, 0, 1, 0, 0, 1, 1, 1, 1])
### END HIDDEN TESTS
print("OK")
```

```
OK
```

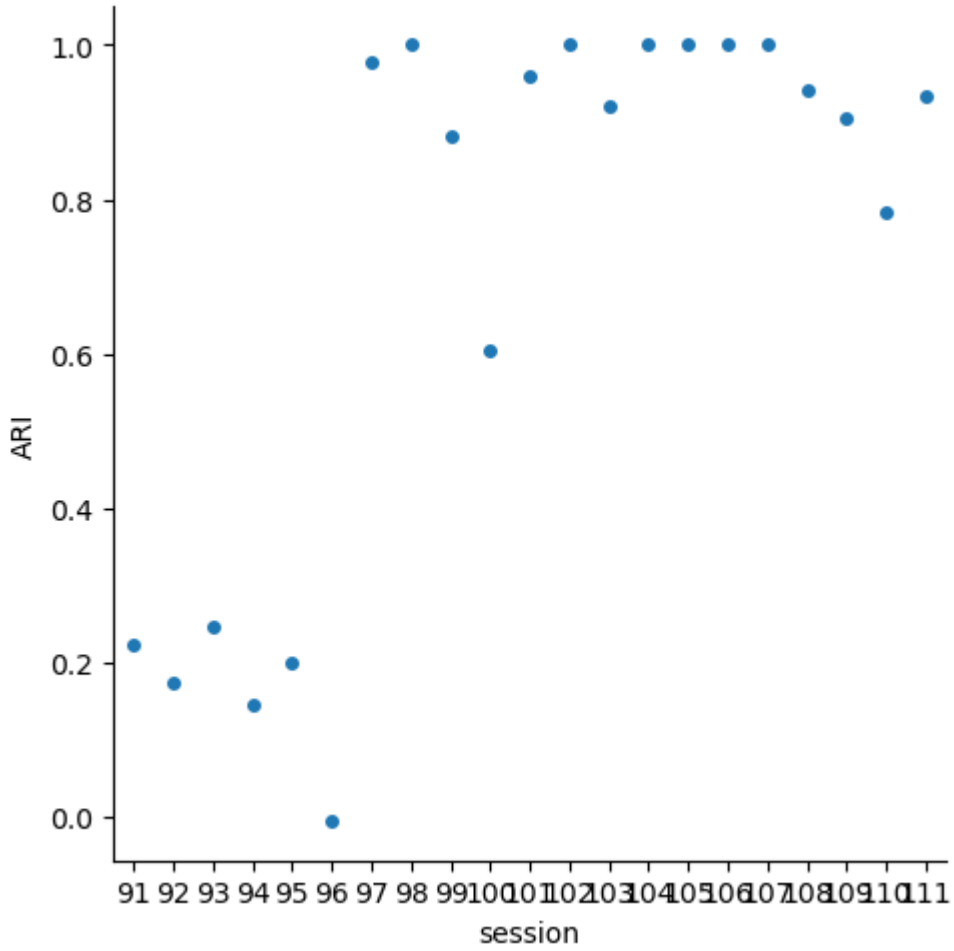
6 (6.4)

Repeating the methodology of step 5, calculate the ARI for all sessions from the 91st through the 111th. Make a scatter plot with session number on the x axis and ARI on the y axis.

In [15]:
```
# Result is a plot

### BEGIN SOLUTION
sessions = range(91,112)
results = []
agg = AgglomerativeClustering(n_clusters=2, linkage="average")

for sesh in sessions:
    X, y = session(sesh, senate, members)
    agg.fit(X)
    y_hat = agg.labels_
    results.append( adjusted_rand_score(y, y_hat) )
sns.catplot(
    data=pd.DataFrame({"session": sessions, "ARI": results}),
    x="session",
    y="ARI"
);
### END SOLUTION
```



(The plot is strong evidence that the agreement between voting records and party affiliation made a huge jump, and that the agreement at the end of the period is very strong.)

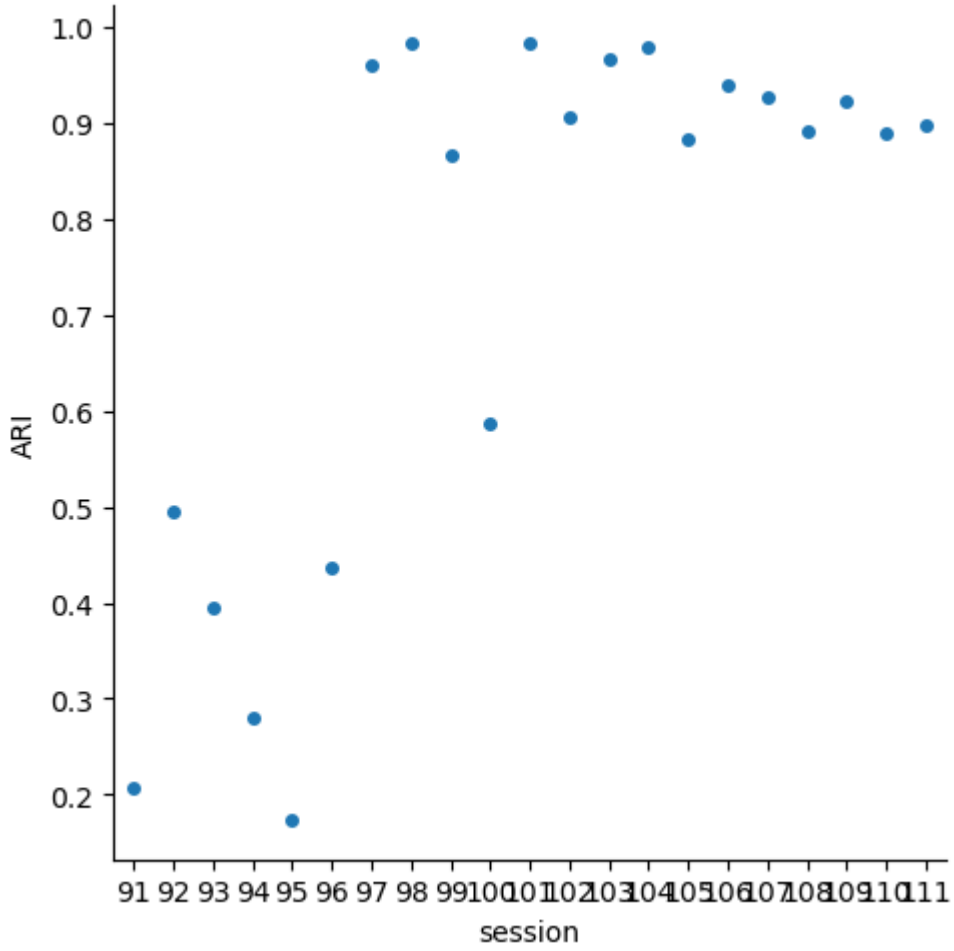
7 (6.4)

Repeat step 6, but with the number of clusters equal to 3.

In [16]:
```
# Result is a plot

### BEGIN SOLUTION
sessions = range(91,112)
results = []
agg = AgglomerativeClustering(n_clusters=3, linkage="average")
for sesh in sessions:
    X, y = session(sesh, senate, members)
    agg.fit(X)
    y_hat = agg.labels_
    results.append( adjusted_rand_score(y, y_hat) )
sns.catplot(
    data=pd.DataFrame({"session": sessions, "ARI": results}),
    x="session",
    y="ARI"
```

```
);
### END SOLUTION
```



(This plot, compared to the previous one, suggests that during the early part of the period, party affiliation often was at odds with voting record similarity if you allow more than 2 voting blocs.)