ML CAC

* Name: Shruti Mall

* Register Number: 23122032

* Class:MSc DS A

* Date :16/05/24

Dataset Descrtiption

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
year	assembly_	state_code	state_name	all_votes	yes_votes	no_votes	abstain	idealpoint_estimate	affinityscore_usa	affinityscore_russia	affinityscore_china	affinityscore_india	affinityscore_brazil	affinityscore_israel
1946	1	2	United States of America	42	25	15	2	1.7377	1	0.2143		0.4762	0.6429	
1947	2	2	United States of America	38	27	10	1	1.8417	1	0.2632		0.2973	0.8421	
1948	3	2	United States of America	103	46	54	3	1.9909	1	0.1275		0.37	0.7767	0.1667
1949	4	2	United States of America	63	17	33	13	1.9395	1	0.1111		0.3651	0.5397	0.5161
1950	5	2	United States of America	53	26	25	2	1.8651	1	0.1731		0.5094	0.8113	0.6042
1951	6	2	United States of America	25	10	11	4	1.8919	1	0.12		0.36	0.64	0.6522
1952	7	2	United States of America	49	25	19	5	1.9617	1	0.1429		0.3061	0.6531	0.6327
1953	8	2	United States of America	25	12	6	7	1.7707	1	0.2		0.3333	0.64	0.52
1954	9	2	United States of America	30	18	3	9	1.5565	1	0.2		0.3	0.6333	0.4333
1955	10	2	United States of America	27	13	8	6	1.8166	1	0.1481		0.1111	0.7778	0.4815
1956	11	2	United States of America	60	44	11	5	1.3449	1	0.2167		0.3667	0.9167	0.5833
1957	12	2	United States of America	34	22	7	5	1.3156	1	0.2353		0.4118	0.8235	0.6176
1958	13	2	United States of America	33	25	4	4	1.3081	1	0.303		0.4848	0.8485	0.6563
1959	14	2	United States of America	54	23	17	14	1.6179	1	0.1296		0.3396	0.7593	0.7059
1960	15	2	United States of America	103	53	36	14	1.574	1	0.233		0.2913	0.7184	0.6735
1961	16	2	United States of America	73	36	26	11	1.7276	1	0.1096		0.2466	0.6986	0.6164
1962	17	2	United States of America	46	28	13	5	1.9215	1	0.1739		0.4889	0.6087	0.5238
1963	18	2	United States of America	31	15	6	10	1.904	1	0.129		0.3871	0.5484	0.5
1965	20	2	United States of America	40	14	15	11	2.0057	1	0.225		0.35	0.575	0.5385
1966	21	2	United States of America	50	19	20	11	2.0598	1	0.16		0.22	0.6	0.6875
1967	22	2	United States of America	57	17	26	14	2.2585	1	0.193		0.2281	0.614	0.6182
1968	23	2	United States of America	51	18	22	11	2.209	1	0.1373		0.28	0.449	0.587
1969	24	2	United States of America	43	16	18	9	2.1056	1	0.2326		0.3488	0.4878	0.4419
1970	25	2	United States of America	67	31	23	13	2.1448	1	0.1343		0.2985	0.4923	0.5373
1971	26	2	United States of America	119	54	26	39	1.8363	1	0.2437	0.2553	0.3675	0.614	0.5804
1972	27	2	United States of America	100	41	21	38	2.067	1	0.303	0.2338	0.3333	0.47	0.5253
1973	28	2	United States of America	91	26	31	34	2.2431	1	0.2967	0.1972	0.3034	0.4205	0.6623

- **assembly_session**: This column represents the session number of the assembly. It likely serves as a unique identifier for each assembly session.
- **state_code**: This column contains numerical codes that represent different states. These codes are likely standardized identifiers for each state.
- **state_name**: This column contains the names of the states corresponding to the state codes. Each state name is associated with its respective state code.
- **all_votes**: This column represents the total number of votes cast in each assembly session. It provides an overall count of votes, including 'yes' votes, 'no' votes, and abstentions.
- **yes_votes**: This column represents the number of 'yes' votes cast in each assembly session. It indicates the count of votes in favor of a particular motion or proposal.
- **no_votes**: This column represents the number of 'no' votes cast in each assembly session. It indicates the count of votes against a particular motion or proposal.
- abstain: This column represents the number of abstentions in each assembly session. It indicates the count of members who chose not to vote either in favor or against a particular motion.
- **idealpoint_estimate**: This column contains numerical estimates representing the ideal point of the assembly for each session. It could be a measure of the assembly's preferred position on a particular issue or policy.

- **affinityscore_usa**: This column contains affinity scores representing the relationship or similarity between each state and the United States. A higher affinity score indicates a stronger perceived alignment or similarity with the United States.
- **affinityscore_russia**: This column contains affinity scores representing the relationship or similarity between each state and Russia. -Similar to affinityscore_usa, a higher score indicates a stronger perceived alignment or similarity with Russia.
- **affinityscore_china**: This column contains affinity scores representing the relationship or similarity between each state and China. Similar to the previous columns, a higher score indicates a stronger perceived alignment or similarity with China.
- **affinityscore_india**: This column contains affinity scores representing the relationship or similarity between each state and India. A higher score indicates a stronger perceived alignment or similarity with India.
- **affinityscore_brazil**: This column contains affinity scores representing the relationship or similarity between each state and Brazil. A higher score indicates a stronger perceived alignment or similarity with Brazil.
- **affinityscore_israel**: This column contains affinity scores representing the relationship or similarity between each state and Israel. A - -higher score indicates a stronger perceived alignment or similarity with Israel.

This dataset appears to capture voting patterns and relationships between different states and various countries based on affinity scores. It provides valuable insights into state-level dynamics and international relations.

Tools and Libraries

- **Pandas**: Used for importing the dataset, performing data manipulation, and conducting exploratory analysis.
- **Matplotlib**: Employed for creating various types of plots, including histograms, scatter plots, and regression visualizations.
- **Seaborn**: Enhances the visualization aesthetics and provides additional plotting functions for statistical analysis.
- **Scikit-learn**: Utilized for building regression models to predict median house values based on the dataset features.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
df=pd.read_csv("states.csv")
```

In [ ]:
```
df.describe(exclude='object')
```

Out[ ]:

	year	assembly_session	state_code	all_votes	yes_votes	9
count	9697.000000	9697.000000	9697.000000	9697.000000	9697.000000	
mean	1987.037125	42.037125	446.914613	75.246674	59.793235	
std	18.478671	18.478671	258.472010	33.043167	32.657278	
min	1946.000000	1.000000	2.000000	1.000000	0.000000	
25%	1973.000000	28.000000	220.000000	58.000000	38.000000	
50%	1989.000000	44.000000	437.000000	68.000000	57.000000	
75%	2003.000000	58.000000	660.000000	86.000000	71.000000	
max	2015.000000	70.000000	990.000000	158.000000	156.000000	

In [ ]:
```
df.describe(include='object')
```

Out[ ]:

	state_name
count	9697
unique	197
top	United States of America
freq	69

In [ ]:
```
import pandas as pd

# Check unique country names in the DataFrame
unique_countries = df['state_name'].unique()
print(unique_countries)
```

```
['United States of America' 'Canada' 'Bahamas' 'Cuba' 'Haiti'
 'Dominican Republic' 'Jamaica' 'Trinidad and Tobago' 'Barbados'
 'Dominica' 'Grenada' 'St. Lucia' 'St. Vincent and the Grenadines'
 'Antigua & Barbuda' 'St. Kitts and Nevis' 'Mexico' 'Belize' 'Guatemala'
 'Honduras' 'El Salvador' 'Nicaragua' 'Costa Rica' 'Panama' 'Colombia'
 'Venezuela' 'Guyana' 'Suriname' 'Ecuador' 'Peru' 'Brazil' 'Bolivia'
 'Paraguay' 'Chile' 'Argentina' 'Uruguay' 'United Kingdom' 'Ireland'
 'Netherlands' 'Belgium' 'Luxembourg' 'France' 'Monaco' 'Liechtenstein'
 'Switzerland' 'Spain' 'Andorra' 'Portugal' 'German Federal Republic'
 'German Democratic Republic' 'Poland' 'Austria' 'Hungary'
 'Czechoslovakia' 'Czech Republic' 'Slovakia' 'Italy' 'San Marino' 'Malta'
 'Albania' 'Montenegro' nan 'Macedonia' 'Croatia' 'Yugoslavia'
 'Bosnia and Herzegovina' 'Slovenia' 'Greece' 'Cyprus' 'Bulgaria'
 'Moldova' 'Romania' 'Russia' 'Estonia' 'Latvia' 'Lithuania' 'Ukraine'
 'Belarus' 'Armenia' 'Georgia' 'Azerbaijan' 'Finland' 'Sweden' 'Norway'
 'Denmark' 'Iceland' 'Cape Verde' 'Sao Tome and Principe' 'Guinea-Bissau'
 'Equatorial Guinea' 'Gambia' 'Mali' 'Senegal' 'Benin' 'Mauritania'
 'Niger' 'Ivory Coast' 'Guinea' 'Burkina Faso' 'Liberia' 'Sierra Leone'
 'Ghana' 'Togo' 'Cameroon' 'Nigeria' 'Gabon' 'Central African Republic'
 'Chad' 'Congo' 'Democratic Republic of the Congo' 'Uganda' 'Kenya'
 'Tanzania' 'Burundi' 'Rwanda' 'Somalia' 'Djibouti' 'Ethiopia' 'Eritrea'
 'Angola' 'Mozambique' 'Zambia' 'Zimbabwe' 'Malawi' 'South Africa'
 'Namibia' 'Lesotho' 'Botswana' 'Swaziland' 'Madagascar' 'Comoros'
 'Mauritius' 'Seychelles' 'Morocco' 'Algeria' 'Tunisia' 'Libya' 'Sudan'
 'South Sudan' 'Iran' 'Turkey' 'Iraq' 'Egypt' 'Syria' 'Lebanon' 'Jordan'
 'Israel' 'Saudi Arabia' 'Yemen Arab Republic' "Yemen People's Republic"
 'Kuwait' 'Bahrain' 'Qatar' 'United Arab Emirates' 'Oman' 'Afghanistan'
 'Turkmenistan' 'Tajikistan' 'Kyrgyzstan' 'Uzbekistan' 'Kazakhstan'
 'China' 'Mongolia' 'Taiwan' 'North Korea' 'South Korea' 'Japan' 'India'
 'Bhutan' 'Pakistan' 'Bangladesh' 'Myanmar' 'Sri Lanka' 'Maldives' 'Nepal'
 'Thailand' 'Cambodia' 'Laos' 'Vietnam' 'Malaysia' 'Singapore' 'Brunei'
 'Philippines' 'Indonesia' 'East Timor' 'Australia' 'Papua New Guinea'
 'New Zealand' 'Vanuatu' 'Solomon Islands' 'Kiribati' 'Tuvalu' 'Fiji'
 'Tonga' 'Nauru' 'Marshall Islands' 'Palau'
 'Federated States of Micronesia' 'Samoa']
```

In [ ]:
```
unique_countries_count = df['state_name'].nunique()
print(unique_countries_count)
```

```
197
```

In [ ]:
```
df.head(20)
```

Out[ ]:

	year	assembly_session	state_code	state_name	all_votes	yes_votes	no_v
0	1946.0	1.0	2	United States of America	42.0	25.0	
1	1947.0	2.0	2	United States of America	38.0	27.0	
2	1948.0	3.0	2	United States of America	103.0	46.0	
3	1949.0	4.0	2	United States of America	63.0	17.0	
4	1950.0	5.0	2	United States of America	53.0	26.0	
5	1951.0	6.0	2	United States of America	25.0	10.0	
6	1952.0	7.0	2	United States of America	49.0	25.0	
7	1953.0	8.0	2	United States of America	25.0	12.0	
8	1954.0	9.0	2	United States of America	30.0	18.0	
9	1955.0	10.0	2	United States of America	27.0	13.0	
10	1956.0	11.0	2	United States of America	60.0	44.0	
11	1957.0	12.0	2	United States of America	34.0	22.0	
12	1958.0	13.0	2	United States of America	33.0	25.0	
13	1959.0	14.0	2	United States of America	54.0	23.0	
14	1960.0	15.0	2	United States of America	103.0	53.0	
15	1961.0	16.0	2	United States of America	73.0	36.0	

	year	assembly_session	state_code	state_name	all_votes	yes_votes	no_v
16	1962.0	17.0	2	United States of America	46.0	28.0	
17	1963.0	18.0	2	United States of America	31.0	15.0	
18	1965.0	20.0	2	United States of America	40.0	14.0	
19	1966.0	21.0	2	United States of America	50.0	19.0	

In [ ]:
```
df.tail(20)
```

Out[ ]:

	year	assembly_session	state_code	state_name	all_votes	yes_votes	n
9685	1996.0	51.0	990	Samoa	74.0	64.0	
9686	1997.0	52.0	990	Samoa	66.0	56.0	
9687	1998.0	53.0	990	Samoa	57.0	49.0	
9688	1999.0	54.0	990	Samoa	62.0	50.0	
9689	2000.0	55.0	990	Samoa	62.0	52.0	
9690	2001.0	56.0	990	Samoa	37.0	30.0	
9691	2002.0	57.0	990	Samoa	62.0	52.0	
9692	2003.0	58.0	990	Samoa	67.0	55.0	
9693	2004.0	59.0	990	Samoa	65.0	51.0	
9694	2005.0	60.0	990	Samoa	71.0	58.0	
9695	2006.0	61.0	990	Samoa	80.0	59.0	
9696	2007.0	62.0	990	Samoa	66.0	55.0	
9697	2008.0	63.0	990	Samoa	69.0	58.0	
9698	2009.0	64.0	990	Samoa	64.0	50.0	
9699	2010.0	65.0	990	Samoa	65.0	53.0	
9700	2011.0	66.0	990	Samoa	59.0	48.0	
9701	2012.0	67.0	990	Samoa	68.0	56.0	
9702	2013.0	68.0	990	Samoa	62.0	51.0	
9703	2014.0	69.0	990	Samoa	75.0	65.0	
9704	2015.0	70.0	990	Samoa	67.0	59.0	

In [ ]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9705 entries, 0 to 9704
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   year                  9697 non-null   float64
 1   assembly_session      9697 non-null   float64
 2   state_code            9705 non-null   int64  
 3   state_name            9697 non-null   object 
 4   all_votes             9697 non-null   float64
 5   yes_votes             9697 non-null   float64
 6   no_votes              9697 non-null   float64
 7   abstain               9697 non-null   float64
 8   idealpoint_estimate   9697 non-null   float64
 9   affinityscore_usa     9696 non-null   float64
 10  affinityscore_russia  9692 non-null   float64
 11  affinityscore_china   7608 non-null   float64
 12  affinityscore_india   9696 non-null   float64
 13  affinityscore_brazil  9696 non-null   float64
 14  affinityscore_israel  9585 non-null   float64
dtypes: float64(13), int64(1), object(1)
memory usage: 1.1+ MB
None
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
year                    8
assembly_session        8
state_code              0
state_name              8
all_votes               8
yes_votes               8
no_votes                8
abstain                 8
idealpoint_estimate     8
affinityscore_usa       9
affinityscore_russia   13
affinityscore_china    2097
affinityscore_india     9
affinityscore_brazil    9
affinityscore_israel  120
dtype: int64
```

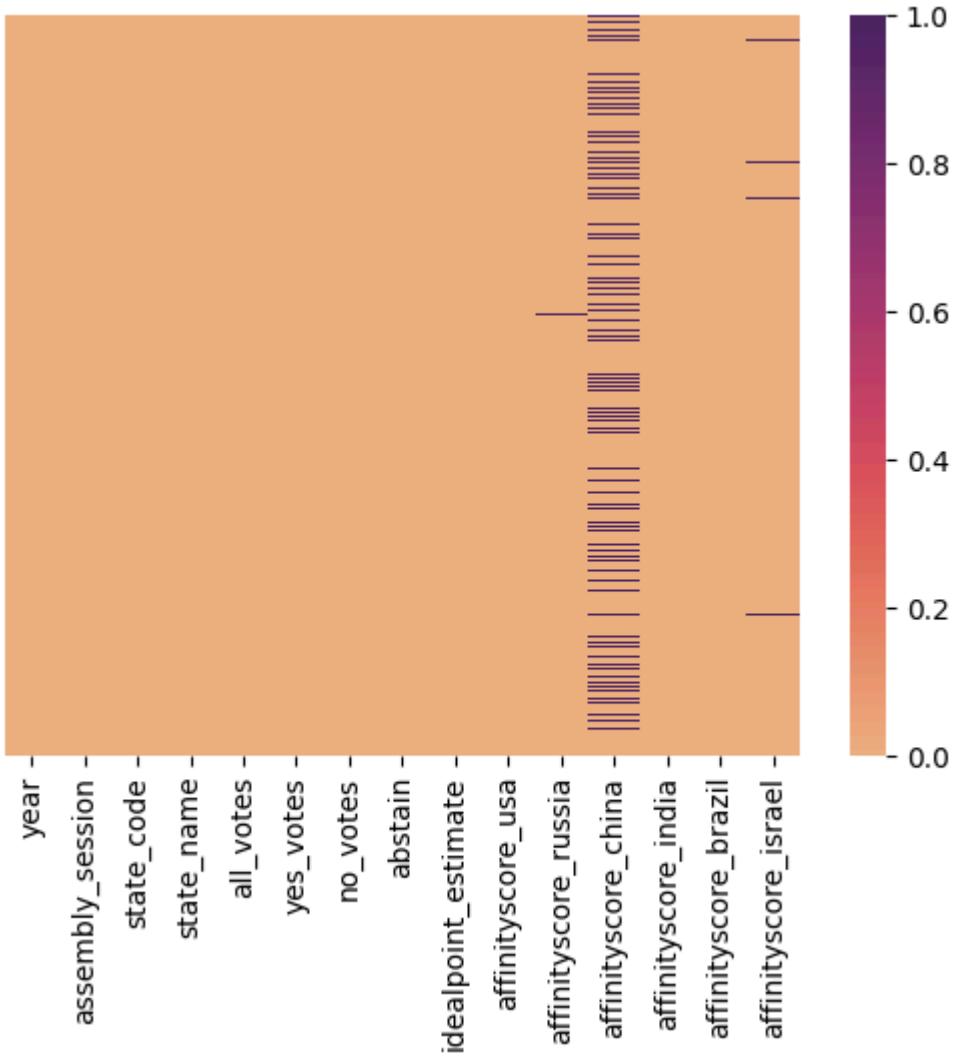
In [ ]:
```
# Drop rows with null values in the 'state_name' column
df = df.dropna(subset=['state_name'])
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
year                      0
assembly_session          0
state_code                0
state_name                0
all_votes                 0
yes_votes                 0
no_votes                  0
abstain                   0
idealpoint_estimate       0
affinityscore_usa         1
affinityscore_russia      5
affinityscore_china    2089
affinityscore_india       1
affinityscore_brazil      1
affinityscore_israel    112
dtype: int64
```

In [ ]:
```
sns.heatmap(df.isnull(),yticklabels=False,cmap='flare')
```

Out[ ]:
```
<Axes: >
```



In [ ]:
```
df.shape
```

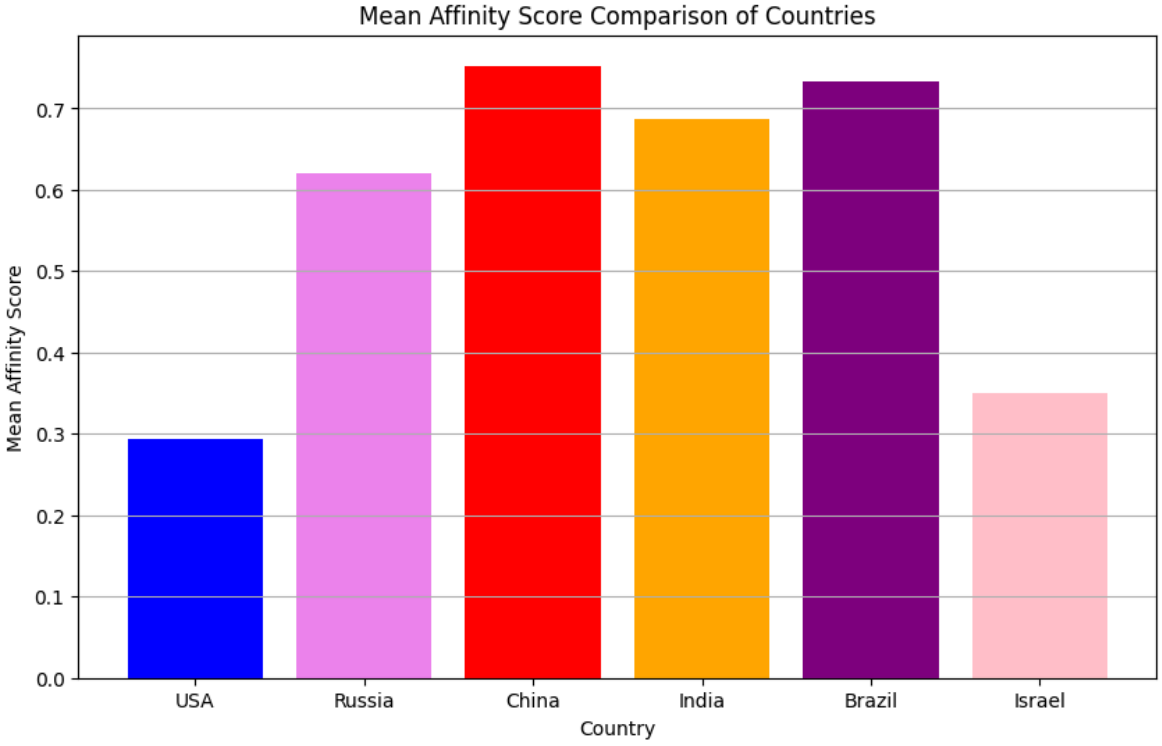
Out[ ]:
```
(9697, 15)
```

In [ ]:
```
# Calculate the mean of each column
mean_affinityscore_usa = df['affinityscore_usa'].mean()
```

```
mean_affinityscore_russia = df['affinityscore_russia'].mean()
mean_affinityscore_india = df['affinityscore_india'].mean()
mean_affinityscore_brazil = df['affinityscore_brazil'].mean()
mean_affinityscore_israel = df['affinityscore_israel'].mean()
mean_affinityscore_china = df['affinityscore_china'].mean()

# Replace null values with mean of each column using .loc accessor
df.loc[df['affinityscore_usa'].isnull(), 'affinityscore_usa'] = mean_affi
df.loc[df['affinityscore_russia'].isnull(), 'affinityscore_russia'] = mea
df.loc[df['affinityscore_india'].isnull(), 'affinityscore_india'] = mean_
df.loc[df['affinityscore_brazil'].isnull(), 'affinityscore_brazil'] = mea
df.loc[df['affinityscore_israel'].isnull(), 'affinityscore_israel'] = mea
df.loc[df['affinityscore_china'].isnull(), 'affinityscore_china'] = mean_
```

In [ ]:

```
import matplotlib.pyplot as plt

# Calculate the mean affinity score for each country
mean_affinity_usa = df['affinityscore_usa'].mean()
mean_affinity_russia = df['affinityscore_russia'].mean()
mean_affinity_china = df['affinityscore_china'].mean()
mean_affinity_india = df['affinityscore_india'].mean()
mean_affinity_brazil = df['affinityscore_brazil'].mean()
mean_affinity_israel = df['affinityscore_israel'].mean()

# Create lists of means and corresponding countries
countries = ['USA', 'Russia', 'China', 'India', 'Brazil', 'Israel']
means = [mean_affinity_usa, mean_affinity_russia, mean_affinity_china, me

# Plotting the mean affinity scores
plt.figure(figsize=(10, 6))
plt.bar(countries, means, color=['blue', 'violet', 'red', 'orange', 'purp
plt.xlabel('Country')
plt.ylabel('Mean Affinity Score')
plt.title('Mean Affinity Score Comparison of Countries')
plt.grid(axis='y')
plt.show()
```



- The graph compares the mean affinity score of six countries: USA, Russia, China, India, Brazil and Israel.
- The countries on the x-axis are listed alphabetically from Brazil to USA. The y-axis shows the mean affinity score. The scale goes from 0 to 0.7
- The countries with the highest mean affinity score are China and India (at about 0.65). The United States and Russia have the lowest mean affinity score (at about 0.15).

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
year                    0
assembly_session        0
state_code              0
state_name              0
all_votes               0
yes_votes               0
no_votes                0
abstain                 0
idealpoint_estimate     0
affinityscore_usa       0
affinityscore_russia    0
affinityscore_china     0
affinityscore_india     0
affinityscore_brazil    0
affinityscore_israel    0
dtype: int64
```

In [ ]:
```
df.describe()
```

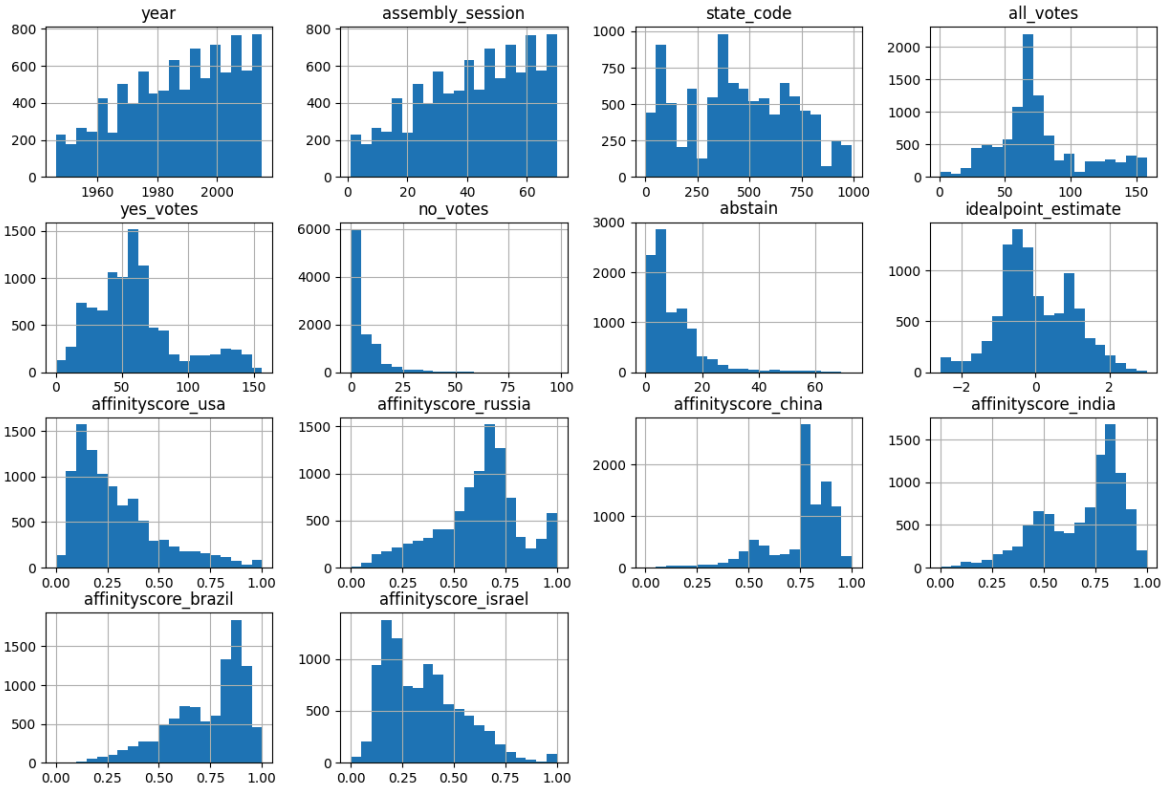
Out[ ]:

	year	assembly_session	state_code	all_votes	yes_votes	9
count	9697.000000	9697.000000	9697.000000	9697.000000	9697.000000	
mean	1987.037125	42.037125	446.914613	75.246674	59.793235	
std	18.478671	18.478671	258.472010	33.043167	32.657278	
min	1946.000000	1.000000	2.000000	1.000000	0.000000	
25%	1973.000000	28.000000	220.000000	58.000000	38.000000	
50%	1989.000000	44.000000	437.000000	68.000000	57.000000	
75%	2003.000000	58.000000	660.000000	86.000000	71.000000	
max	2015.000000	70.000000	990.000000	158.000000	156.000000	

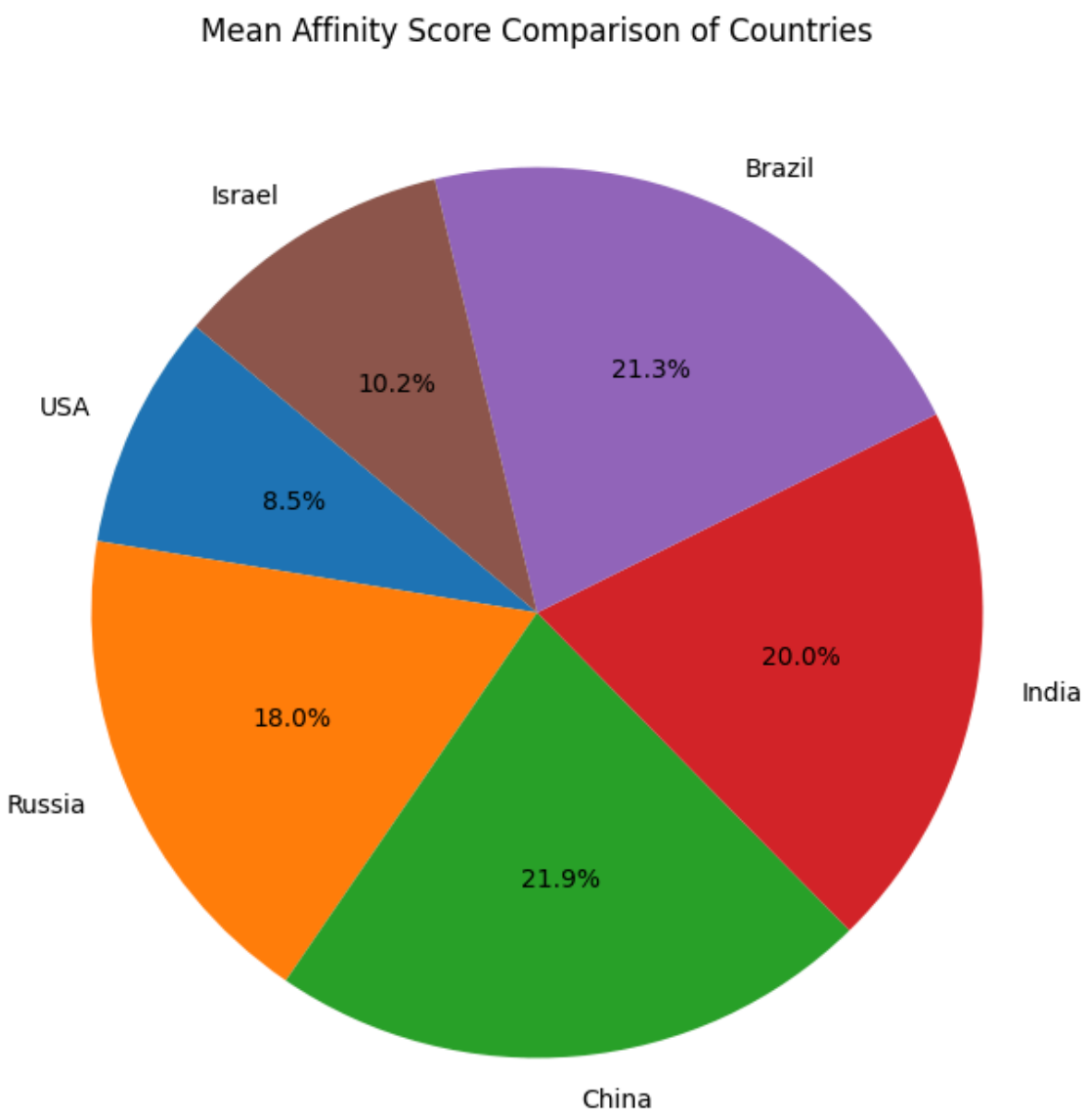
In [ ]:
```
df.nunique()
```

Out[ ]:
```
year                    69
assembly_session        69
state_code             198
state_name             197
all_votes              158
yes_votes              157
no_votes                70
abstain                 71
idealpoint_estimate    8377
affinityscore_usa      2257
affinityscore_russia   2411
affinityscore_china    1687
affinityscore_india    2276
affinityscore_brazil   2120
affinityscore_israel   2023
dtype: int64
```

In [ ]:
```
import matplotlib.pyplot as plt
# Histograms for each numerical feature
df.hist(bins=20, figsize=(15, 10))
plt.show()
```



In [ ]:
```
countries = ['USA', 'Russia', 'China', 'India', 'Brazil', 'Israel']
means = [mean_affinity_usa, mean_affinity_russia, mean_affinity_china, me

# Plotting the mean affinity scores as a pie chart
plt.figure(figsize=(8, 8))
plt.pie(means, labels=countries, autopct='%1.1f%%', startangle=140)
plt.title('Mean Affinity Score Comparison of Countries')
plt.show()
```



In [ ]:

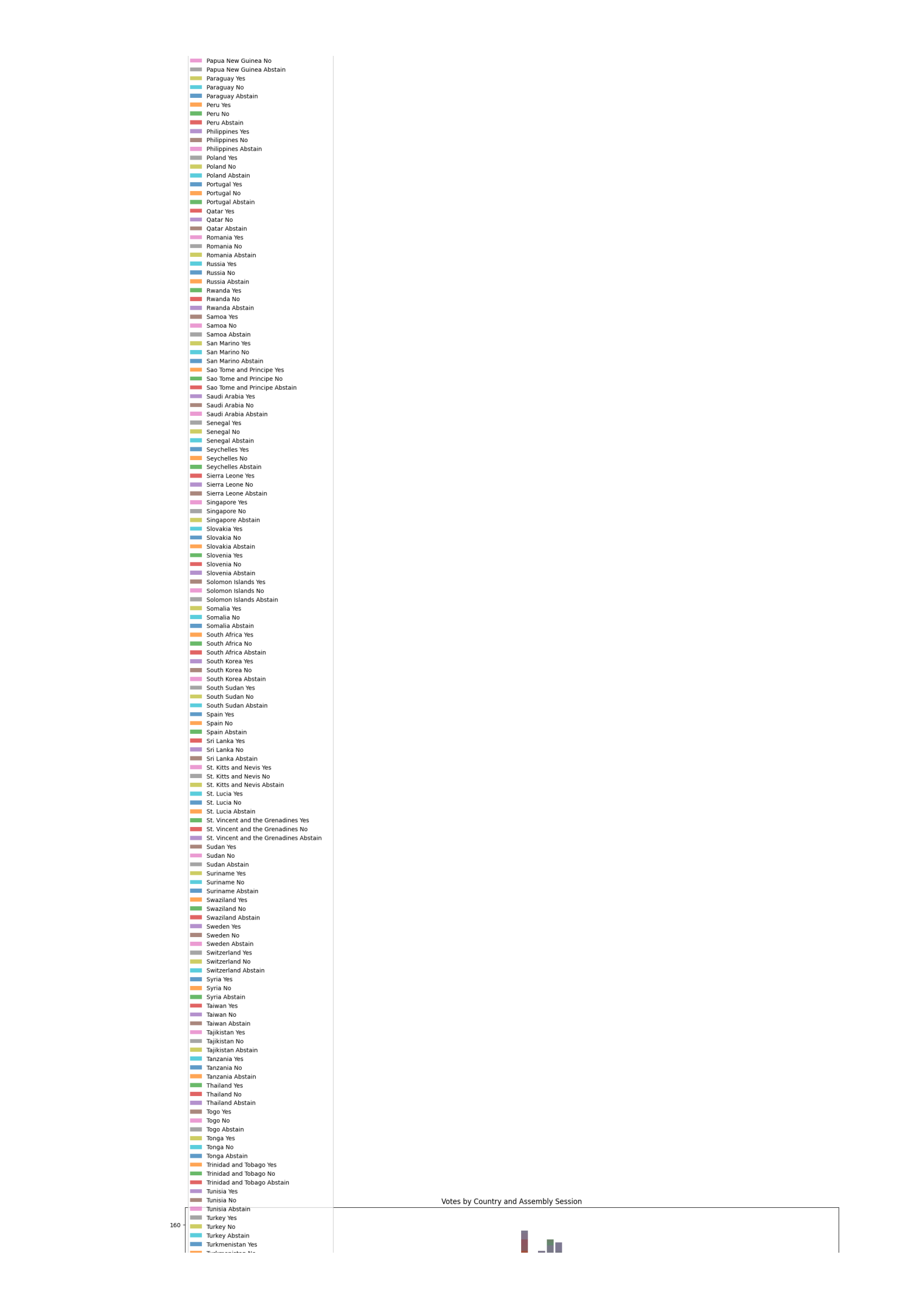
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming you have loaded your dataset into a DataFrame named df
# Let's group the DataFrame by 'state_name' and 'assembly_session' and ca
grouped_df = df.groupby(['state_name', 'assembly_session']).sum().reset_i

# Stacked Bar Plot for Yes, No, and Abstain votes
plt.figure(figsize=(20, 15))
for country, data in grouped_df.groupby('state_name'):
    plt.bar(data['assembly_session'], data['yes_votes'], label=f'{country
    plt.bar(data['assembly_session'], data['no_votes'], bottom=data['yes_
    plt.bar(data['assembly_session'], data['abstain'], bottom=data['yes_v

plt.xlabel('Assembly Session')
plt.ylabel('Votes')
plt.title('Votes by Country and Assembly Session')
plt.legend()
plt.show()
```

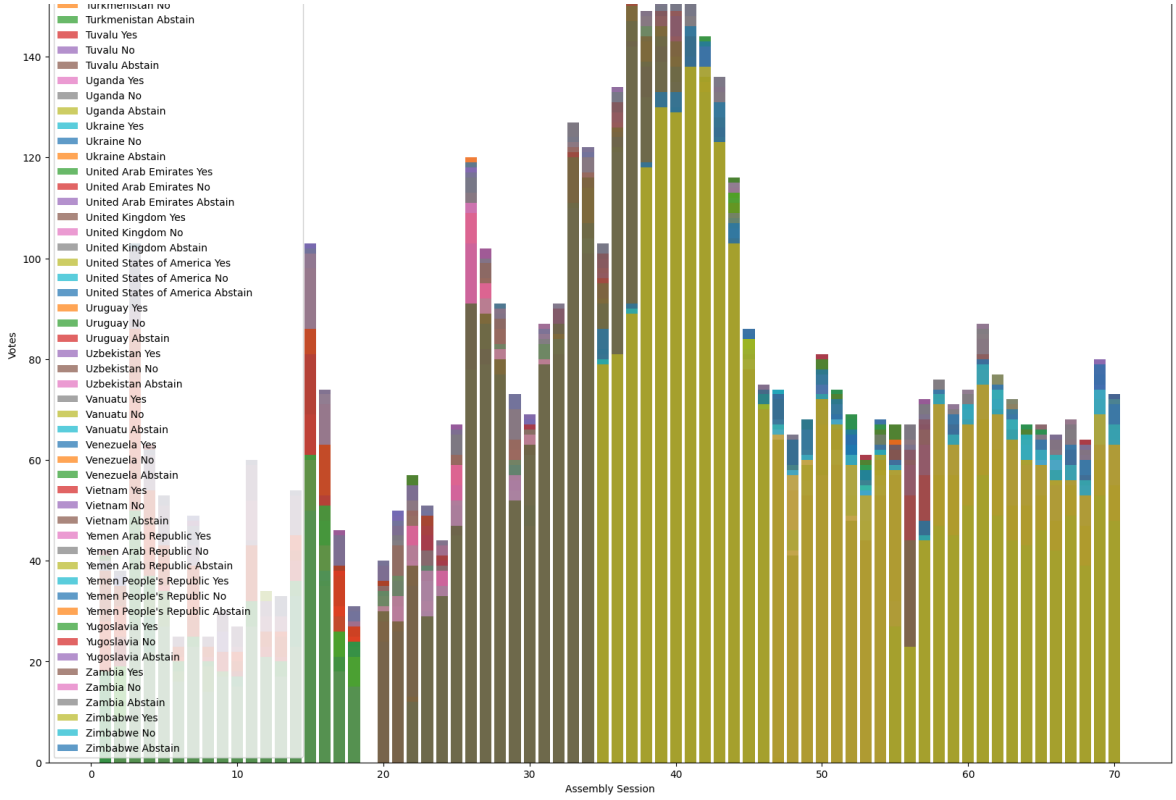
```
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-p
ackages/IPython/core/pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```


Afghanistan Yes
Afghanistan No
Afghanistan Abstain
Albania Yes
Albania No
Albania Abstain
Algeria Yes
Algeria No
Algeria Abstain
Andorra Yes
Andorra No
Andorra Abstain
Angola Yes
Angola No
Angola Abstain
Antigua & Barbuda Yes
Antigua & Barbuda No
Antigua & Barbuda Abstain
Argentina Yes
Argentina No
Argentina Abstain
Armenia Yes
Armenia No
Armenia Abstain
Australia Yes
Australia No
Australia Abstain
Austria Yes
Austria No
Austria Abstain
Azerbaijan Yes
Azerbaijan No
Azerbaijan Abstain
Bahamas Yes
Bahamas No
Bahamas Abstain
Bahrain Yes
Bahrain No
Bahrain Abstain
Bangladesh Yes
Bangladesh No
Bangladesh Abstain
Barbados Yes
Barbados No
Barbados Abstain
Belarus Yes
Belarus No
Belarus Abstain
Belgium Yes
Belgium No
Belgium Abstain
Belize Yes
Belize No
Belize Abstain
Benin Yes
Benin No
Benin Abstain
Bhutan Yes
Bhutan No
Bhutan Abstain
Bolivia Yes
Bolivia No
Bolivia Abstain
Bosnia and Herzegovina Yes
Bosnia and Herzegovina No
Bosnia and Herzegovina Abstain
Botswana Yes
Botswana No
Botswana Abstain
Brazil Yes
Brazil No
Brazil Abstain
Brunei Yes
Brunei No
Brunei Abstain
Bulgaria Yes
Bulgaria No
Bulgaria Abstain
Burkina Faso Yes
Burkina Faso No
Burkina Faso Abstain
Burundi Yes
Burundi No
Burundi Abstain
Cambodia Yes
Cambodia No
Cambodia Abstain
Cameroon Yes
Cameroon No
Cameroon Abstain
Canada Yes
Canada No
Canada Abstain
Cape Verde Yes
Cape Verde No
Cape Verde Abstain
Central African Republic Yes
Central African Republic No
Central African Republic Abstain
Chad Yes
Chad No
Chad Abstain
Chile Yes
Chile No
Chile Abstain
China Yes
China No
China Abstain
Colombia Yes
Colombia No
Colombia Abstain
Comoros Yes
Comoros No
Comoros Abstain
Congo Yes
Congo No
Congo Abstain
Costa Rica Yes
Costa Rica No
Costa Rica Abstain
Croatia Yes
Croatia No
Croatia Abstain
Cuba Yes
Cuba No
Cuba Abstain
Cyprus Yes
Cyprus No
Cyprus Abstain
Czech Republic Yes
Czech Republic No
Czech Republic Abstain
Czechoslovakia Yes
Czechoslovakia No
Czechoslovakia Abstain


Democratic Republic of the Congo Yes
Democratic Republic of the Congo No
Democratic Republic of the Congo Abstain
Denmark Yes
Denmark No
Denmark Abstain
Djibouti Yes
Djibouti No
Djibouti Abstain
Dominica Yes
Dominica No
Dominica Abstain
Dominican Republic Yes
Dominican Republic No
Dominican Republic Abstain
East Timor Yes
East Timor No
East Timor Abstain
Ecuador Yes
Ecuador No
Ecuador Abstain
Egypt Yes
Egypt No
Egypt Abstain
El Salvador Yes
El Salvador No
El Salvador Abstain
Equatorial Guinea Yes
Equatorial Guinea No
Equatorial Guinea Abstain
Eritrea Yes
Eritrea No
Eritrea Abstain
Estonia Yes
Estonia No
Estonia Abstain
Ethiopia Yes
Ethiopia No
Ethiopia Abstain
Federated States of Micronesia Yes
Federated States of Micronesia No
Federated States of Micronesia Abstain
Fiji Yes
Fiji No
Fiji Abstain
Finland Yes
Finland No
Finland Abstain
France Yes
France No
France Abstain
Gabon Yes
Gabon No
Gabon Abstain
Gambia Yes
Gambia No
Gambia Abstain
Georgia Yes
Georgia No
Georgia Abstain
German Democratic Republic Yes
German Democratic Republic No
German Democratic Republic Abstain
German Federal Republic Yes
German Federal Republic No
German Federal Republic Abstain
Ghana Yes
Ghana No
Ghana Abstain
Greece Yes
Greece No
Greece Abstain
Grenada Yes
Grenada No
Grenada Abstain
Guatemala Yes
Guatemala No
Guatemala Abstain
Guinea Yes
Guinea No
Guinea Abstain
Guinea-Bissau Yes
Guinea-Bissau No
Guinea-Bissau Abstain
Guyana Yes
Guyana No
Guyana Abstain
Haiti Yes
Haiti No
Haiti Abstain
Honduras Yes
Honduras No
Honduras Abstain
Hungary Yes
Hungary No
Hungary Abstain
Iceland Yes
Iceland No
Iceland Abstain
India Yes
India No
India Abstain
Indonesia Yes
Indonesia No
Indonesia Abstain
Iran Yes
Iran No
Iran Abstain
Iraq Yes
Iraq No
Iraq Abstain
Ireland Yes
Ireland No
Ireland Abstain
Israel Yes
Israel No
Israel Abstain
Italy Yes
Italy No
Italy Abstain
Ivory Coast Yes
Ivory Coast No
Ivory Coast Abstain
Jamaica Yes
Jamaica No
Jamaica Abstain
Japan Yes
Japan No
Japan Abstain
Jordan Yes
Jordan No
Jordan Abstain
Kazakhstan Yes
Kazakhstan No
Kazakhstan Abstain

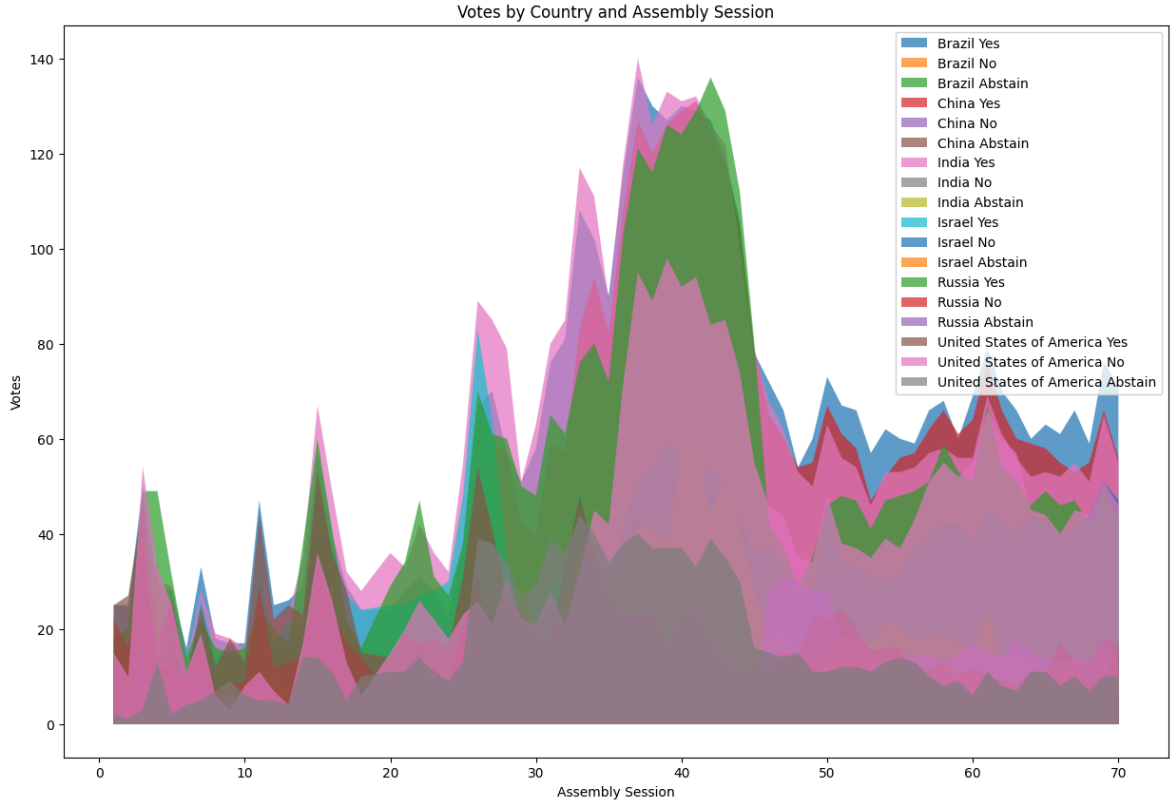

Kenya Yes
Kenya No
Kenya Abstain
Kiribati Yes
Kiribati No
Kiribati Abstain
Kuwait Yes
Kuwait No
Kuwait Abstain
Kyrgyzstan Yes
Kyrgyzstan No
Kyrgyzstan Abstain
Laos Yes
Laos No
Laos Abstain
Latvia Yes
Latvia No
Latvia Abstain
Lebanon Yes
Lebanon No
Lebanon Abstain
Lesotho Yes
Lesotho No
Lesotho Abstain
Liberia Yes
Liberia No
Liberia Abstain
Libya Yes
Libya No
Libya Abstain
Liechtenstein Yes
Liechtenstein No
Liechtenstein Abstain
Lithuania Yes
Lithuania No
Lithuania Abstain
Luxembourg Yes
Luxembourg No
Luxembourg Abstain
Macedonia Yes
Macedonia No
Macedonia Abstain
Madagascar Yes
Madagascar No
Madagascar Abstain
Malawi Yes
Malawi No
Malawi Abstain
Malaysia Yes
Malaysia No
Malaysia Abstain
Maldives Yes
Maldives No
Maldives Abstain
Mali Yes
Mali No
Mali Abstain
Malta Yes
Malta No
Malta Abstain
Marshall Islands Yes
Marshall Islands No
Marshall Islands Abstain
Mauritania Yes
Mauritania No
Mauritania Abstain
Mauritius Yes
Mauritius No
Mauritius Abstain
Mexico Yes
Mexico No
Mexico Abstain
Moldova Yes
Moldova No
Moldova Abstain
Monaco Yes
Monaco No
Monaco Abstain
Mongolia Yes
Mongolia No
Mongolia Abstain
Montenegro Yes
Montenegro No
Montenegro Abstain
Morocco Yes
Morocco No
Morocco Abstain
Mozambique Yes
Mozambique No
Mozambique Abstain
Myanmar Yes
Myanmar No
Myanmar Abstain
Namibia Yes
Namibia No
Namibia Abstain
Nauru Yes
Nauru No
Nauru Abstain
Nepal Yes
Nepal No
Nepal Abstain
Netherlands Yes
Netherlands No
Netherlands Abstain
New Zealand Yes
New Zealand No
New Zealand Abstain
Nicaragua Yes
Nicaragua No
Nicaragua Abstain
Niger Yes
Niger No
Niger Abstain
Nigeria Yes
Nigeria No
Nigeria Abstain
North Korea Yes
North Korea No
North Korea Abstain
Norway Yes
Norway No
Norway Abstain
Oman Yes
Oman No
Oman Abstain
Pakistan Yes
Pakistan No
Pakistan Abstain
Palau Yes
Palau No
Palau Abstain
Panama Yes
Panama No
Panama Abstain
Papua New Guinea Yes


Papua New Guinea No
Papua New Guinea Abstain
Paraguay Yes
Paraguay No
Paraguay Abstain
Peru Yes
Peru No
Peru Abstain
Philippines Yes
Philippines No
Philippines Abstain
Poland Yes
Poland No
Poland Abstain
Portugal Yes
Portugal No
Portugal Abstain
Qatar Yes
Qatar No
Qatar Abstain
Romania Yes
Romania No
Romania Abstain
Russia Yes
Russia No
Russia Abstain
Rwanda Yes
Rwanda No
Rwanda Abstain
Samoa Yes
Samoa No
Samoa Abstain
San Marino Yes
San Marino No
San Marino Abstain
Sao Tome and Principe Yes
Sao Tome and Principe No
Sao Tome and Principe Abstain
Saudi Arabia Yes
Saudi Arabia No
Saudi Arabia Abstain
Senegal Yes
Senegal No
Senegal Abstain
Seychelles Yes
Seychelles No
Seychelles Abstain
Sierra Leone Yes
Sierra Leone No
Sierra Leone Abstain
Singapore Yes
Singapore No
Singapore Abstain
Slovakia Yes
Slovakia No
Slovakia Abstain
Slovenia Yes
Slovenia No
Slovenia Abstain
Solomon Islands Yes
Solomon Islands No
Solomon Islands Abstain
Somalia Yes
Somalia No
Somalia Abstain
South Africa Yes
South Africa No
South Africa Abstain
South Korea Yes
South Korea No
South Korea Abstain
South Sudan Yes
South Sudan No
South Sudan Abstain
Spain Yes
Spain No
Spain Abstain
Sri Lanka Yes
Sri Lanka No
Sri Lanka Abstain
St. Kitts and Nevis Yes
St. Kitts and Nevis No
St. Kitts and Nevis Abstain
St. Lucia Yes
St. Lucia No
St. Lucia Abstain
St. Vincent and the Grenadines Yes
St. Vincent and the Grenadines No
St. Vincent and the Grenadines Abstain
Sudan Yes
Sudan No
Sudan Abstain
Suriname Yes
Suriname No
Suriname Abstain
Swaziland Yes
Swaziland No
Swaziland Abstain
Sweden Yes
Sweden No
Sweden Abstain
Switzerland Yes
Switzerland No
Switzerland Abstain
Syria Yes
Syria No
Syria Abstain
Taiwan Yes
Taiwan No
Taiwan Abstain
Tajikistan Yes
Tajikistan No
Tajikistan Abstain
Tanzania Yes
Tanzania No
Tanzania Abstain
Thailand Yes
Thailand No
Thailand Abstain
Togo Yes
Togo No
Togo Abstain
Tonga Yes
Tonga No
Tonga Abstain
Trinidad and Tobago Yes
Trinidad and Tobago No
Trinidad and Tobago Abstain
Tunisia Yes
Tunisia No
Tunisia Abstain
Turkey Yes
Turkey No
Turkey Abstain
Turkmenistan Yes
Votes by Country and Assembly Session
160

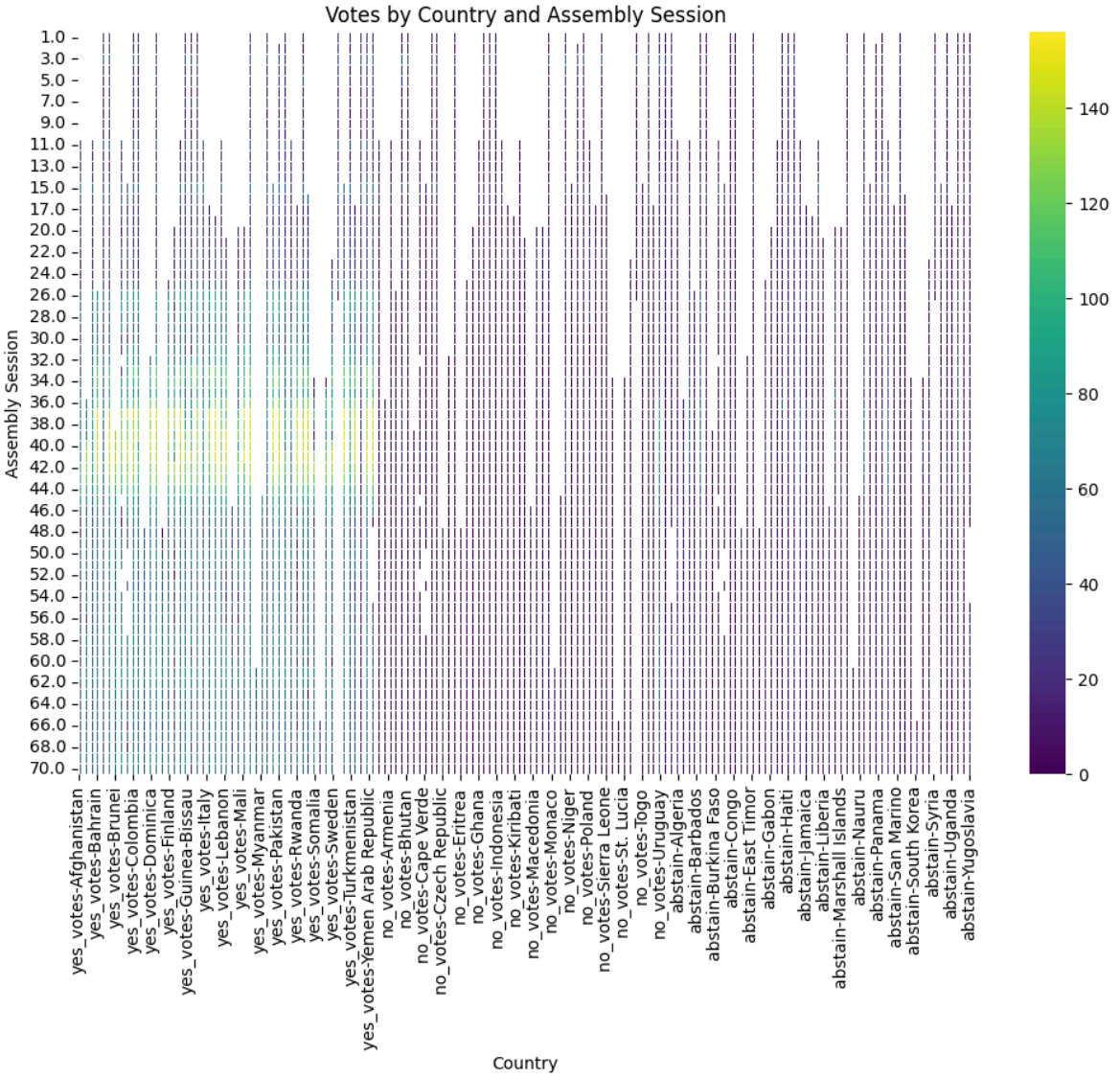


In [ ]:

In [ ]:
```
import matplotlib.pyplot as plt

# Assuming you have already defined grouped_df correctly
plt.figure(figsize=(15, 10))
for country, data in grouped_df.groupby('state_name'):
    plt.fill_between(data['assembly_session'], data['yes_votes'], label=f
    plt.fill_between(data['assembly_session'], data['no_votes'], label=f'
    plt.fill_between(data['assembly_session'], data['abstain'], label=f'{

plt.xlabel('Assembly Session')
plt.ylabel('Votes')
plt.title('Votes by Country and Assembly Session')
plt.legend()
plt.show()
```



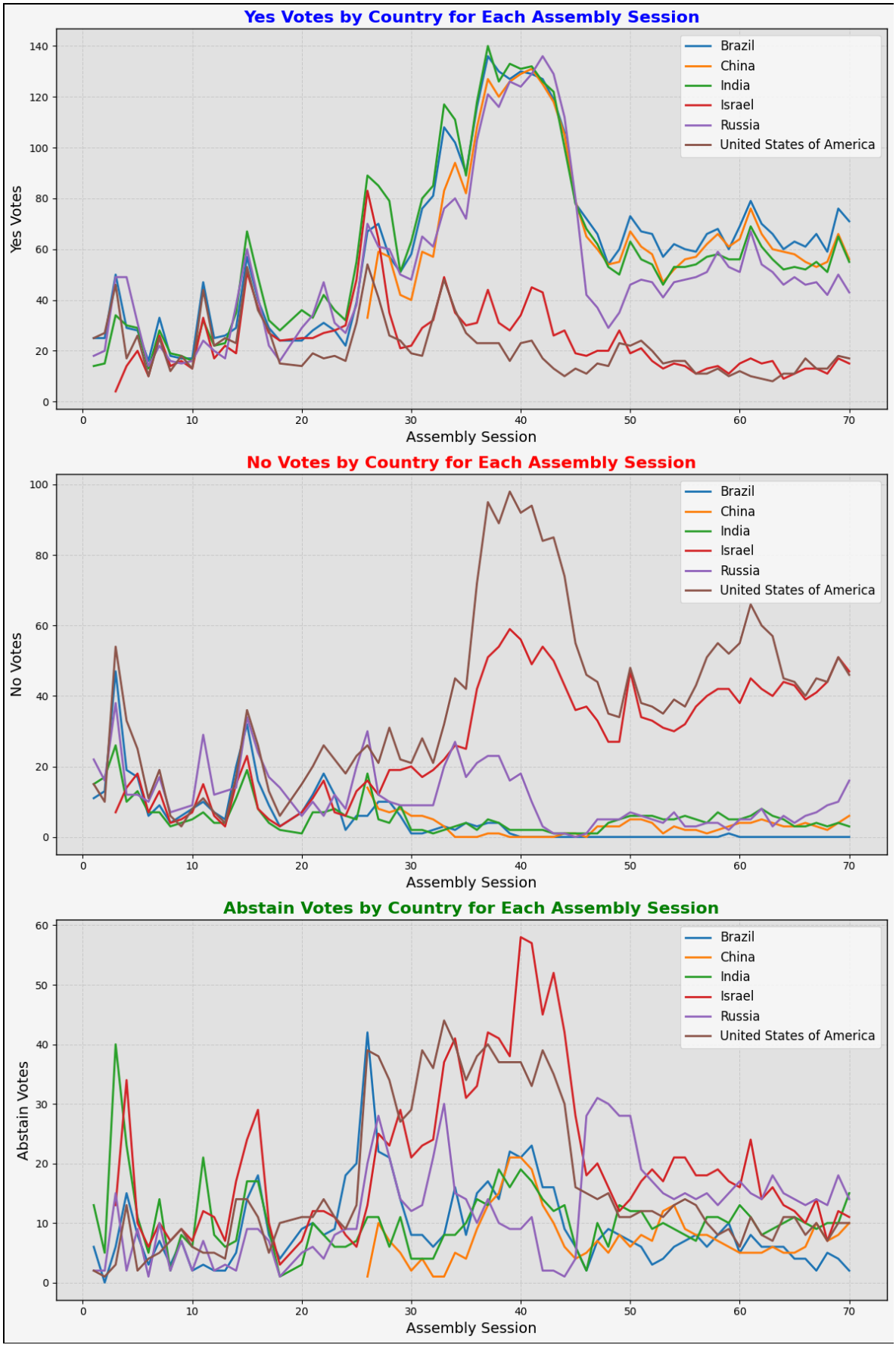
In [ ]:
```
# Create a pivot table for heatmap
heatmap_df = grouped_df.pivot(index='assembly_session', columns='state_na

# Heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_df, cmap='viridis', linewidths=0.5)
plt.title('Votes by Country and Assembly Session')
plt.xlabel('Country')
plt.ylabel('Assembly Session')
plt.show()
```



In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming your DataFrame is named df

# Filter data for the desired countries
desired_countries = ['Brazil', 'China', 'United States of America', 'Isra
filtered_df = df[df['state_name'].isin(desired_countries)]

# Group by assembly session and calculate sum of yes, no, and abstain vot
grouped_df = filtered_df.groupby(['assembly_session', 'state_name']).sum(

# Plotting
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 18))  # Adjusted
fig.patch.set_facecolor('#f5f5f5')  # Set background color for the figure
fig.patch.set_linewidth(1)  # Set border width for the figure
fig.patch.set_edgecolor('black')  # Set border color for the figure

# Set background color for the plot area
for ax in axes:
    ax.set_facecolor('#e5e5e5')  # Set background color for the plot area
    ax.spines['top'].set_visible(True)  # Show top spine/border
    ax.spines['right'].set_visible(True)  # Show right spine/border
    ax.spines['bottom'].set_linewidth(1)  # Set border width for the bott
    ax.spines['left'].set_linewidth(1)  # Set border width for the left s
```

```
    ax.spines['top'].set_linewidth(1)  # Set border width for the top spi
    ax.spines['right'].set_linewidth(1)  # Set border width for the right

# Plot for Yes votes
axes[0].set_title('Yes Votes by Country for Each Assembly Session', fonts
for country, data in grouped_df.groupby('state_name'):
    axes[0].plot(data['assembly_session'], data['yes_votes'], label=count
axes[0].set_xlabel('Assembly Session', fontsize=14)
axes[0].set_ylabel('Yes Votes', fontsize=14)
axes[0].legend(fontsize=12)
axes[0].grid(True, linestyle='--', alpha=0.5)  # Add grid lines

# Plot for No votes
axes[1].set_title('No Votes by Country for Each Assembly Session', fontsi
for country, data in grouped_df.groupby('state_name'):
    axes[1].plot(data['assembly_session'], data['no_votes'], label=countr
axes[1].set_xlabel('Assembly Session', fontsize=14)
axes[1].set_ylabel('No Votes', fontsize=14)
axes[1].legend(fontsize=12)
axes[1].grid(True, linestyle='--', alpha=0.5)  # Add grid lines

# Plot for Abstain votes
axes[2].set_title('Abstain Votes by Country for Each Assembly Session', f
for country, data in grouped_df.groupby('state_name'):
    axes[2].plot(data['assembly_session'], data['abstain'], label=country
axes[2].set_xlabel('Assembly Session', fontsize=14)
axes[2].set_ylabel('Abstain Votes', fontsize=14)
axes[2].legend(fontsize=12)
axes[2].grid(True, linestyle='--', alpha=0.5)  # Add grid lines

plt.tight_layout()
plt.show()
```



In [ ]:
```
# Calculate the mean affinity score for each country
mean_affinity_usa = df['affinityscore_usa'].mean()
mean_affinity_russia = df['affinityscore_russia'].mean()
mean_affinity_china = df['affinityscore_china'].mean()
mean_affinity_india = df['affinityscore_india'].mean()
mean_affinity_brazil = df['affinityscore_brazil'].mean()
mean_affinity_israel = df['affinityscore_israel'].mean()
```

```
# Create lists of means and corresponding countries
countries = ['USA', 'Russia', 'China', 'India', 'Brazil', 'Israel']
means = [mean_affinity_usa, mean_affinity_russia, mean_affinity_china, me

# Create the Highcharts HTML code
highcharts_html = """
<!DOCTYPE html>
<html>
<head>
  <title>Mean Affinity Score Comparison of Countries</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>

<div id="container" style="width: 600px; height: 400px; margin: 0 auto"><

<script>
Highcharts.chart('container', {{
  chart: {{
    type: 'bar'
  }},
  title: {{
    text: 'Mean Affinity Score Comparison of Countries'
  }},
  xAxis: {{
    categories: {categories}
  }},
  yAxis: {{
    title: {{
      text: 'Mean Affinity Score'
    }}
  }},
  series: [{{
    name: 'Mean Affinity Score',
    data: {means}
  }}]
}});
</script>

</body>
</html>
""".format(categories=countries, means=means)


# Save the HTML output to a file or display it
print(highcharts_html)
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Mean Affinity Score Comparison of Countries</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>

<div id="container" style="width: 600px; height: 400px; margin: 0 auto"></
div>

<script>
Highcharts.chart('container', {
  chart: {
    type: 'bar'
  },
  title: {
    text: 'Mean Affinity Score Comparison of Countries'
  },
  xAxis: {
    categories: ['USA', 'Russia', 'China', 'India', 'Brazil', 'Israel']
  },
  yAxis: {
    title: {
      text: 'Mean Affinity Score'
    }
  },
  series: [{
    name: 'Mean Affinity Score',
    data: [0.29358908828382835, 0.620303931077177, 0.7525578338590958, 0.6
873531043729373, 0.7338206373762376, 0.3505397496087637]
  }]
});
</script>

</body>
</html>
```

In [ ]:

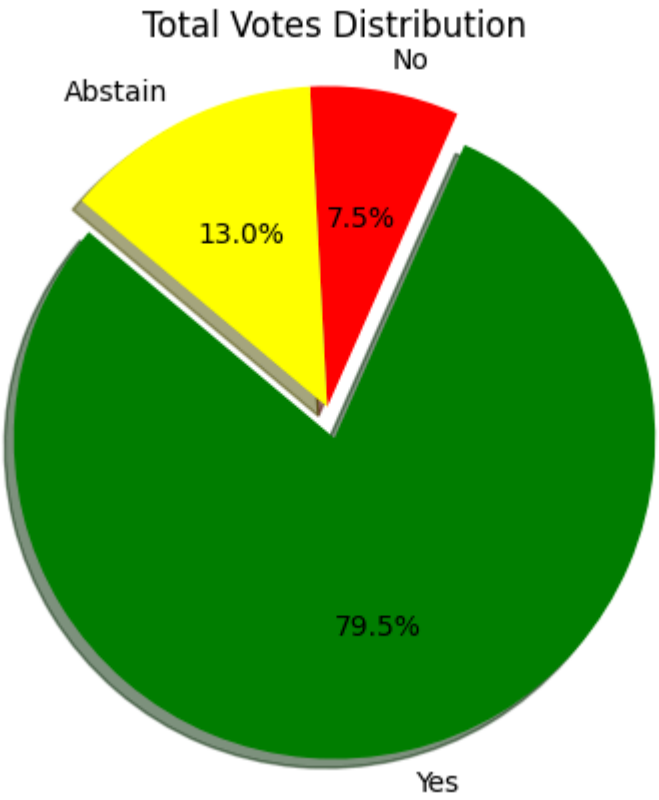
```
html_content = """
<!DOCTYPE html>
<html>
<head>
  <title>Mean Affinity Score Comparison of Countries</title>
  <script src="https://code.highcharts.com/highcharts.js"></script>
</head>
<body>

<div id="container" style="width: 600px; height: 400px; margin: 0 auto"><

<script>
Highcharts.chart('container', {
  chart: {
    type: 'bar'
  },
  title: {
    text: 'Mean Affinity Score Comparison of Countries'
  },
  xAxis: {
    categories: ['USA', 'Russia', 'China', 'India', 'Brazil', 'Israel']
```

```
  },
  yAxis: {
    title: {
      text: 'Mean Affinity Score'
    }
  },
  series: [{
    name: 'Mean Affinity Score',
    data: [0.29358908828382835, 0.620303931077177, 0.6659521023958656, 0.
  }]
});
</script>

</body>
</html>
"""

# Write the HTML content to a file
with open('affinity_score_comparison.html', 'w') as file:
    file.write(html_content)

print("HTML file saved successfully!")
```

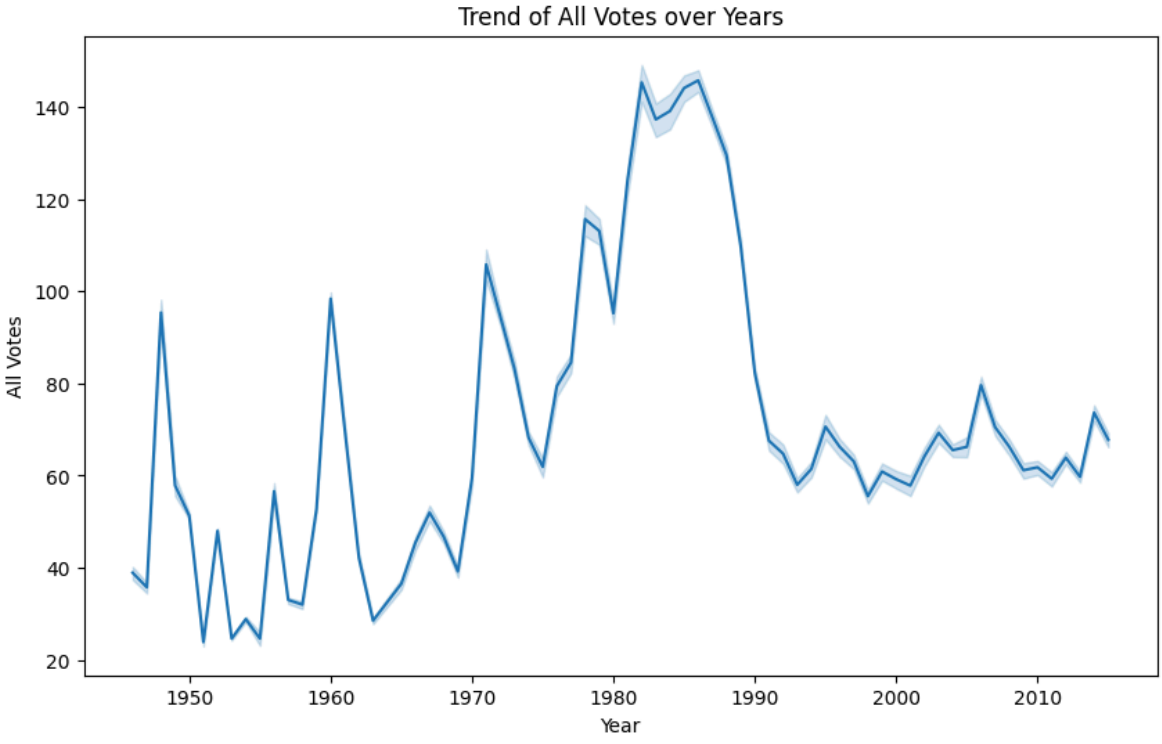
```
HTML file saved successfully!
```

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame with columns 'yes_votes', 'no_votes', a

# Step 1: Calculate the sum of each column
sum_yes_votes = df['yes_votes'].sum()
sum_no_votes = df['no_votes'].sum()
sum_abstain = df['abstain'].sum()

# Step 2: Add up the sums of the three columns to get the total
total_votes = sum_yes_votes + sum_no_votes + sum_abstain

# Step 3: Calculate the percentage of each sum
yes_percentage = (sum_yes_votes / total_votes) * 100
no_percentage = (sum_no_votes / total_votes) * 100
abstain_percentage = (sum_abstain / total_votes) * 100

# Step 4: Create a pie chart
labels = ['Yes', 'No', 'Abstain']
sizes = [yes_percentage, no_percentage, abstain_percentage]
colors = ['green', 'red', 'yellow']
explode = (0.1, 0, 0)  # explode 1st slice (optional)

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a ci
plt.title('Total Votes Distribution')
plt.show()
```

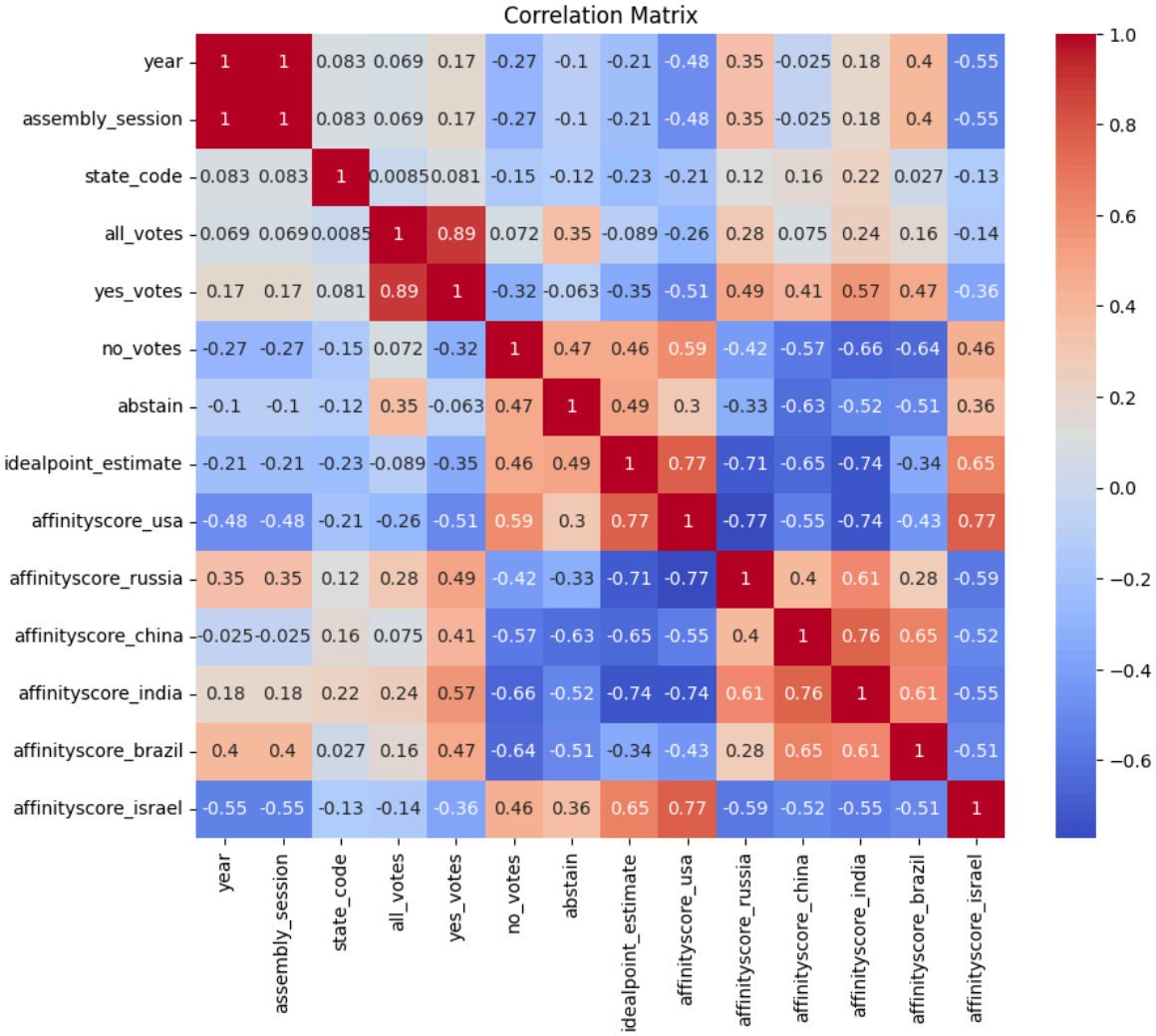


In [ ]:
```
# Time Series Analysis
plt.figure(figsize=(10, 6))
sns.lineplot(x='year', y='all_votes', data=df)
plt.title('Trend of All Votes over Years')
plt.xlabel('Year')
plt.ylabel('All Votes')
plt.show()
```

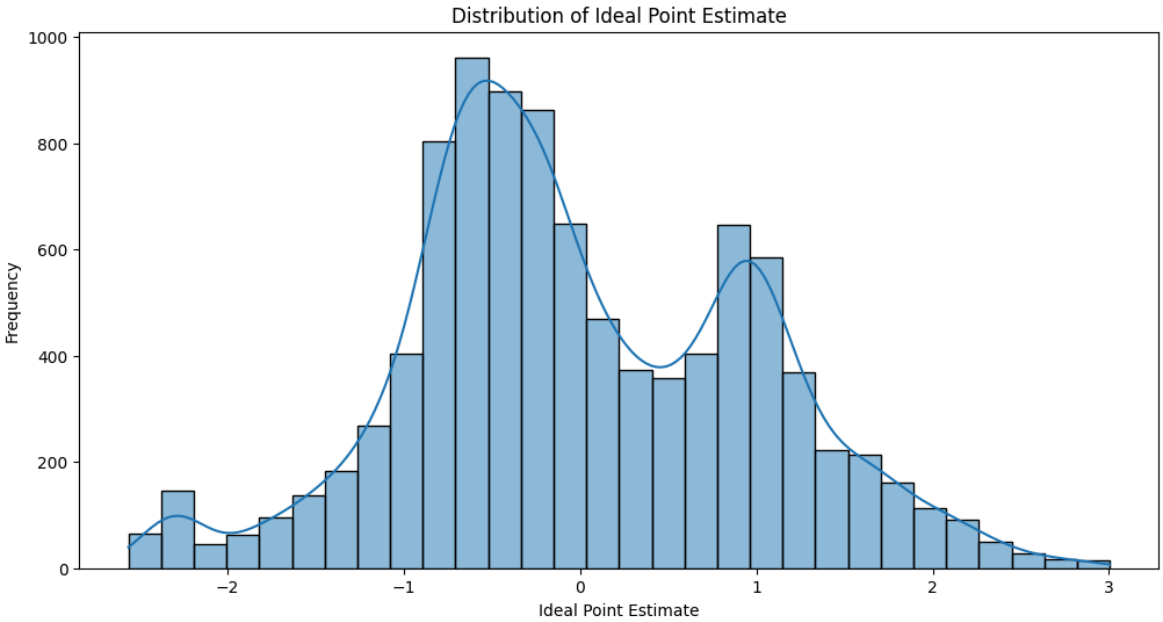


In [ ]:

In [ ]:
```
# Correlation Analysis
# correlation_matrix = df.corr()
# plt.figure(figsize=(10, 8))
# sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
# plt.title('Correlation Matrix')
# plt.show()
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
numeric_data = df[numeric_columns]

# Visualize correlation matrix
correlation_matrix = numeric_data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

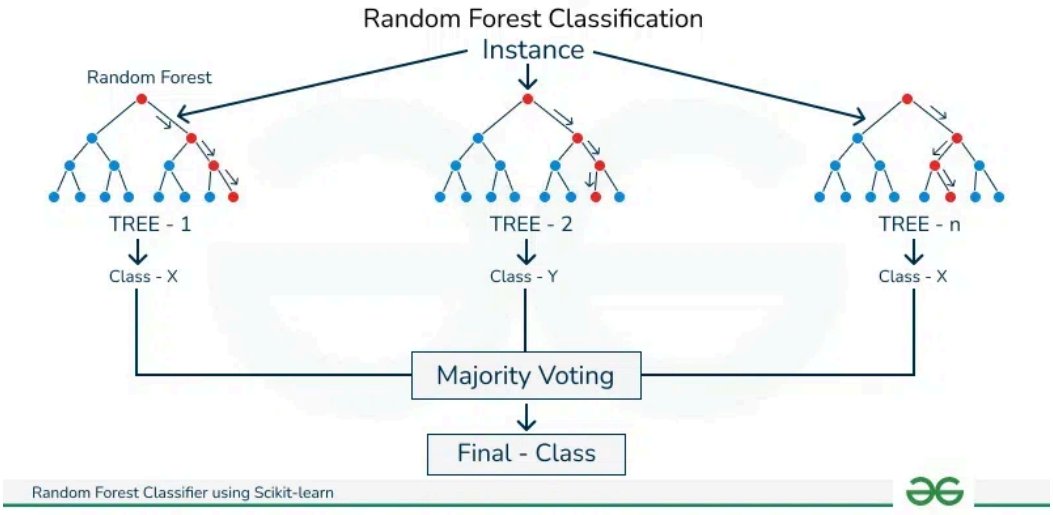


In [ ]:
```
# Distribution Analysis
plt.figure(figsize=(12, 6))
sns.histplot(df['idealpoint_estimate'], kde=True, bins=30)
plt.title('Distribution of Ideal Point Estimate')
plt.xlabel('Ideal Point Estimate')
plt.ylabel('Frequency')
plt.show()
```



Random Forest Classifier

- The Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is a set of decision trees (DT) from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction.



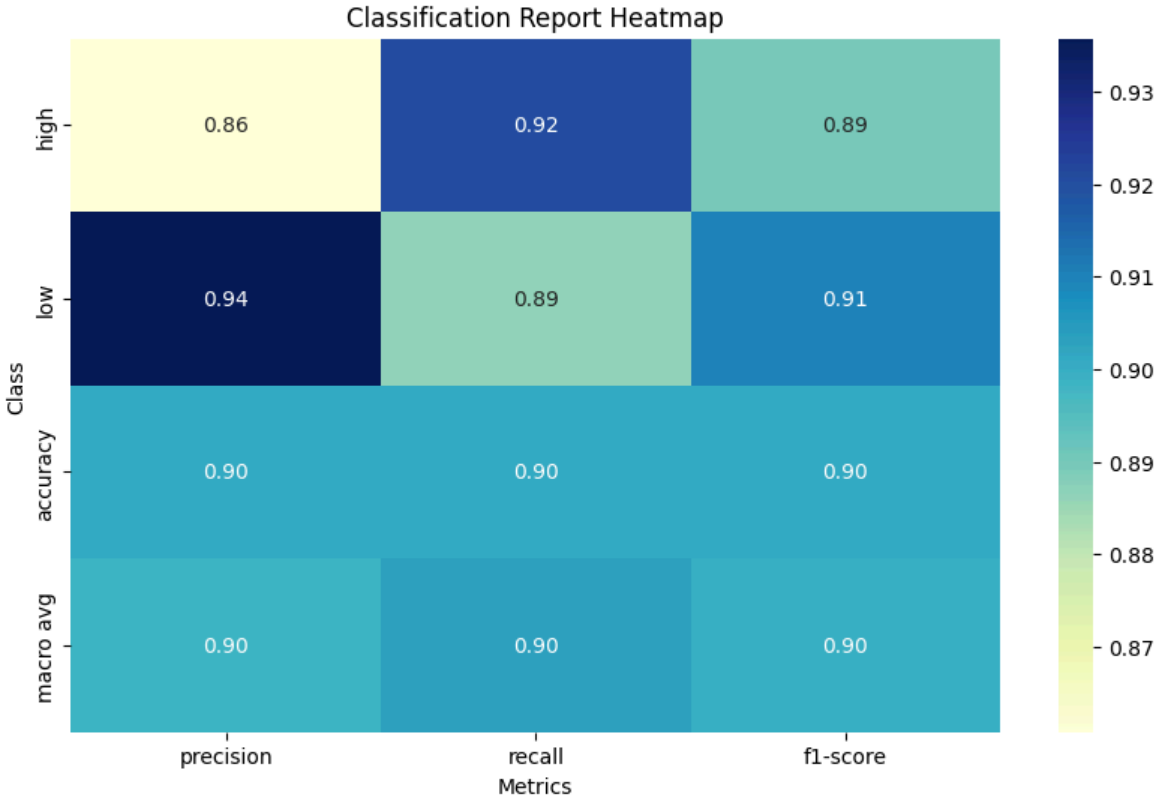
In [ ]:
```
# import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confus

# Load the data (replace 'data.csv' with your actual file path)
# data = pd.read_csv('data.csv')

# Prepare data
X = df[['affinityscore_usa', 'affinityscore_russia',
        'affinityscore_china', 'affinityscore_india',
        'affinityscore_brazil', 'affinityscore_israel']]
y = df['yes_votes'].apply(lambda x: 'high' if x > df['yes_votes'].mean()

# Split data into train and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Initialize and train the model
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)


# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9010309278350516
Classification Report:
              precision    recall  f1-score   support

        high       0.86      0.92      0.89       840
         low       0.94      0.89      0.91      1100

    accuracy                           0.90      1940
   macro avg       0.90      0.90      0.90      1940
weighted avg       0.90      0.90      0.90      1940

[[773  67]
 [125 975]]
```

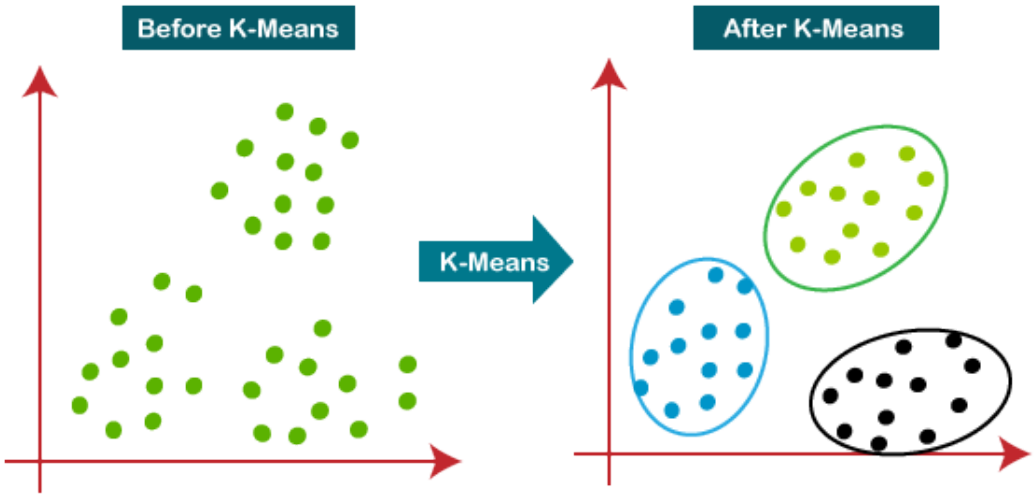
In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Generate classification report
report = classification_report(y_test, y_pred, output_dict=True)

# Convert the classification report to a DataFrame
report_df = pd.DataFrame(report).transpose()

# Plot the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(report_df.iloc[:-1, :-1], annot=True, cmap="YlGnBu", fmt=".2f
plt.title('Classification Report Heatmap')
plt.xlabel('Metrics')
plt.ylabel('Class')
plt.show()
```



K Means

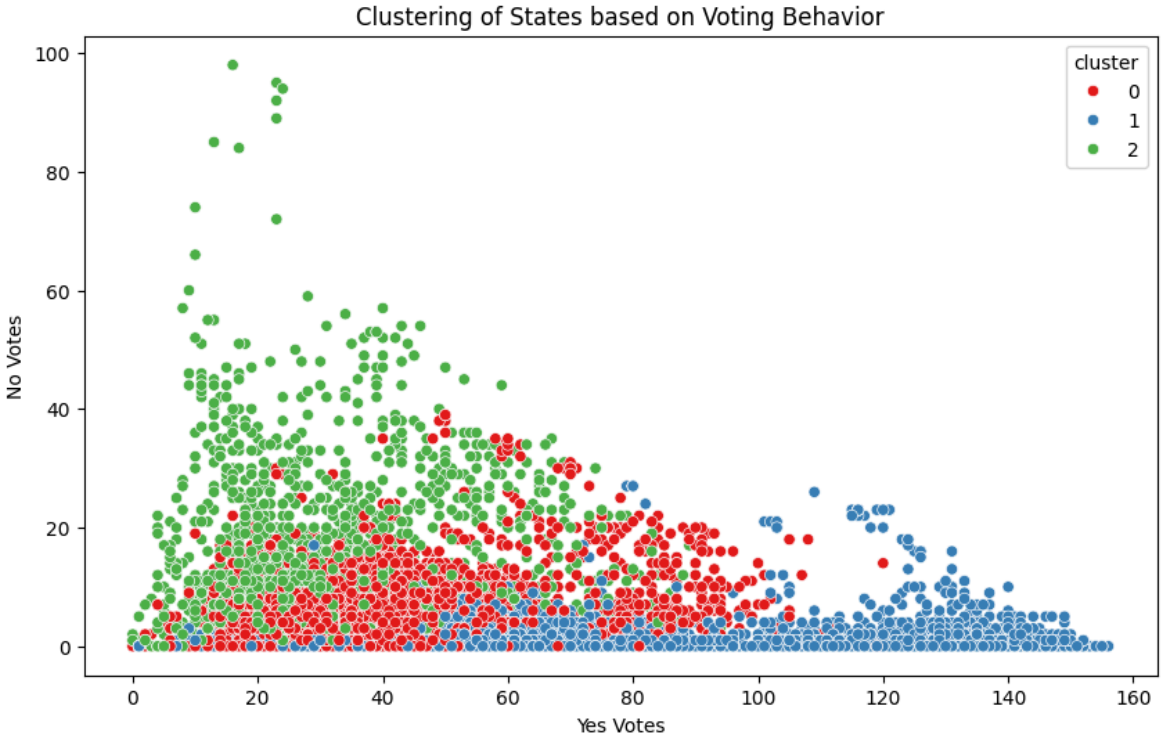
- Unsupervised Machine Learning is the process of teaching a computer to use unlabeled, unclassified data and enabling the algorithm to operate on that data without supervision. Without any previous data training, the machine's job in this case is to organize unsorted data according to parallels, patterns, and variations.



In [ ]:
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt



# Prepare data
X = df[['affinityscore_usa', 'affinityscore_russia',
        'affinityscore_china', 'affinityscore_india',
```

```
          'affinityscore_brazil', 'affinityscore_israel']]

# Initialize and fit K-means model
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Add cluster labels to the DataFrame
df['cluster'] = kmeans.labels_

# Visualize clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='yes_votes', y='no_votes', hue='cluster', data=df, pale
plt.title('Clustering of States based on Voting Behavior')
plt.xlabel('Yes Votes')
plt.ylabel('No Votes')
plt.show()
```

```
/var/folders/4y/p5wbs2392ng1qdh47g8kk6x00000gn/T/ipykernel_38761/293989921
3.py:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['cluster'] = kmeans.labels_
```



In [ ]:
```
from sklearn.metrics import silhouette_score

# Calculate silhouette score
silhouette_avg = silhouette_score(X, kmeans.labels_)
print("Silhouette Score:", silhouette_avg)
```

```
Silhouette Score: 0.40464240364072335
```

In [ ]:
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
```

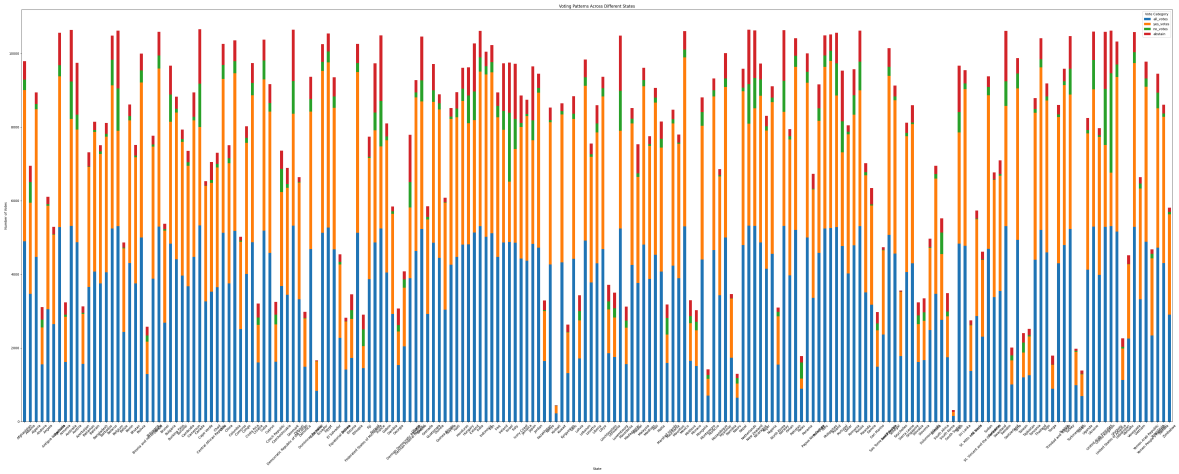
```
# Prepare data
X = df[['affinityscore_usa', 'affinityscore_russia',
          'affinityscore_china', 'affinityscore_india',
          'affinityscore_brazil', 'affinityscore_israel']]

# Initialize and fit K-means model
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Add cluster labels to the DataFrame
df['cluster'] = kmeans.labels_

# Visualize clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='affinityscore_india', y='affinityscore_china', hue='cl
plt.title('Clustering of States based on Affinity Scores')
plt.xlabel('Affinity Score - Inida')
plt.ylabel('Affinity Score - China')
plt.show()
```

```
/var/folders/4y/p5wbs2392ng1qdh47g8kk6x00000gn/T/ipykernel_38761/269634134
9.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['cluster'] = kmeans.labels_
```

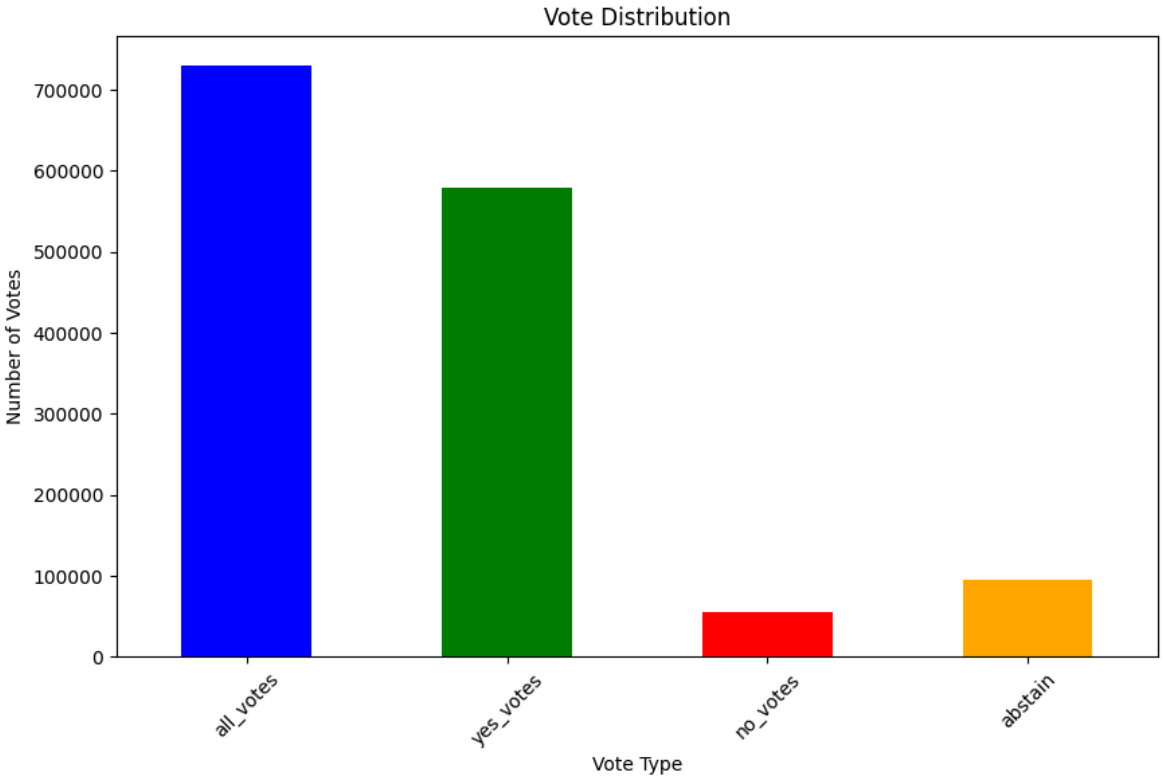
In [ ]:
```
from sklearn.metrics import silhouette_score

# Calculate silhouette score
silhouette_avg = silhouette_score(X, kmeans.labels_)
print("Silhouette Score:", silhouette_avg)
```

```
Silhouette Score: 0.40464240364072335
```

In [ ]:
```
# Group the data by state and calculate the sum of votes for each categor
state_votes = df.groupby('state_name')[['all_votes', 'yes_votes', 'no_vot

# Plotting bar charts for each voting category
state_votes.plot(kind='bar', stacked=True, figsize=(50, 20))
plt.title('Voting Patterns Across Different States')
plt.xlabel('State')
plt.ylabel('Number of Votes')
plt.xticks(rotation=45)  # Rotate state names for better readability
plt.legend(title='Vote Category')
plt.tight_layout()  # Corrected attribute name
plt.show()
```



In [ ]:
```
# Calculate the sum of votes for each category
vote_distribution = df[['all_votes', 'yes_votes', 'no_votes', 'abstain']]

# Plotting bar chart for vote distribution
plt.figure(figsize=(10, 6))
vote_distribution.plot(kind='bar', color=['blue', 'green', 'red', 'orange
plt.title('Vote Distribution')
plt.xlabel('Vote Type')
plt.ylabel('Number of Votes')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

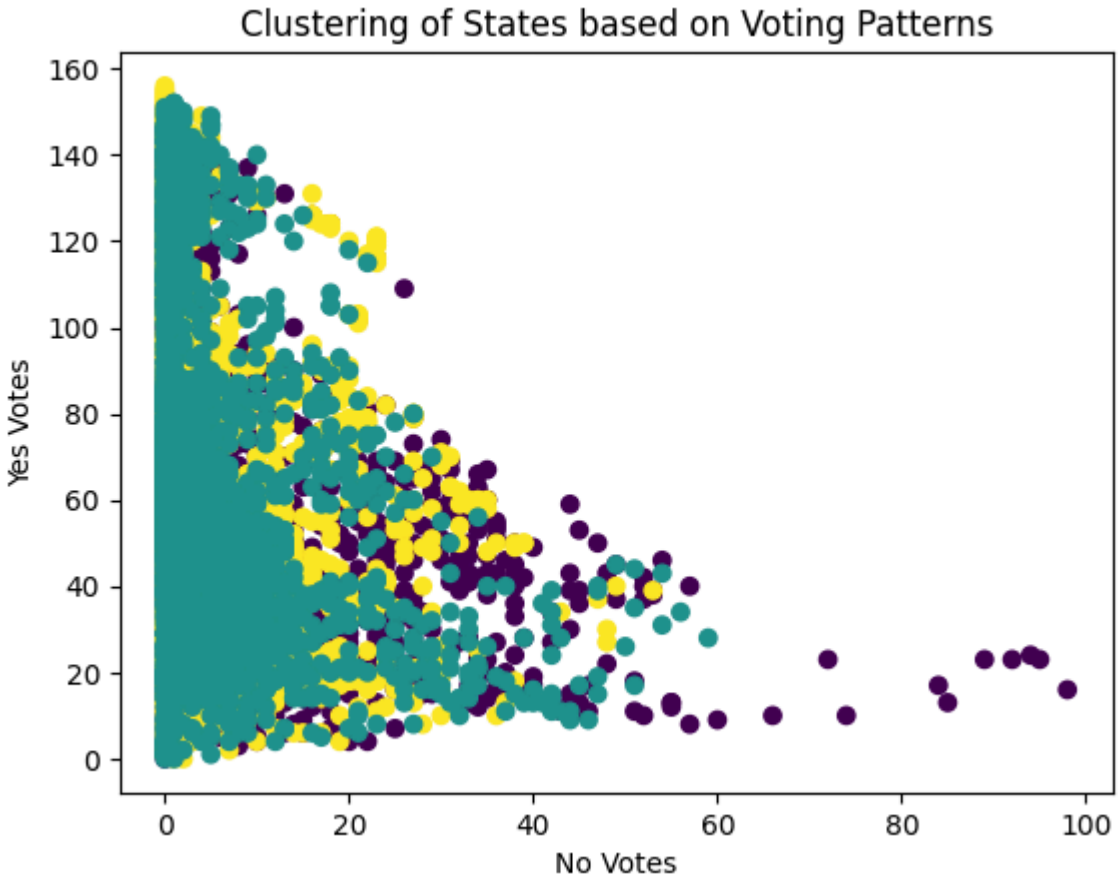


In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score



# Data preprocessing
# For simplicity, let's assume 'state_name' is the target variable and ot
X = df.drop(columns=['state_name'])  # Features
y = df['state_name']  # Target

# Encode categorical variables
le = LabelEncoder()
y = le.fit_transform(y)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Model training
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.5252577319587629
```

In [ ]:
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Data preprocessing
```

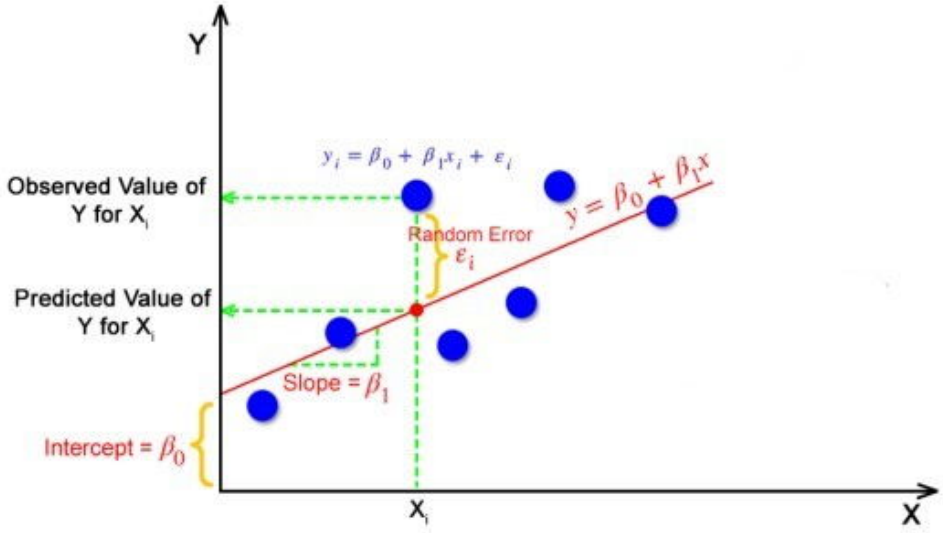
```
# For simplicity, let's assume 'state_name' is dropped as we are clusteri
X = df.drop(columns=['state_name'])

# Model training
kmeans = KMeans(n_clusters=3)  # Specify the number of clusters
kmeans.fit(X)

# Visualizing clusters
plt.scatter(X['no_votes'], X['yes_votes'], c=kmeans.labels_, cmap='viridi
plt.xlabel('No Votes')
plt.ylabel('Yes Votes')
plt.title('Clustering of States based on Voting Patterns')
plt.show()
```



Linear Regression

- Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's

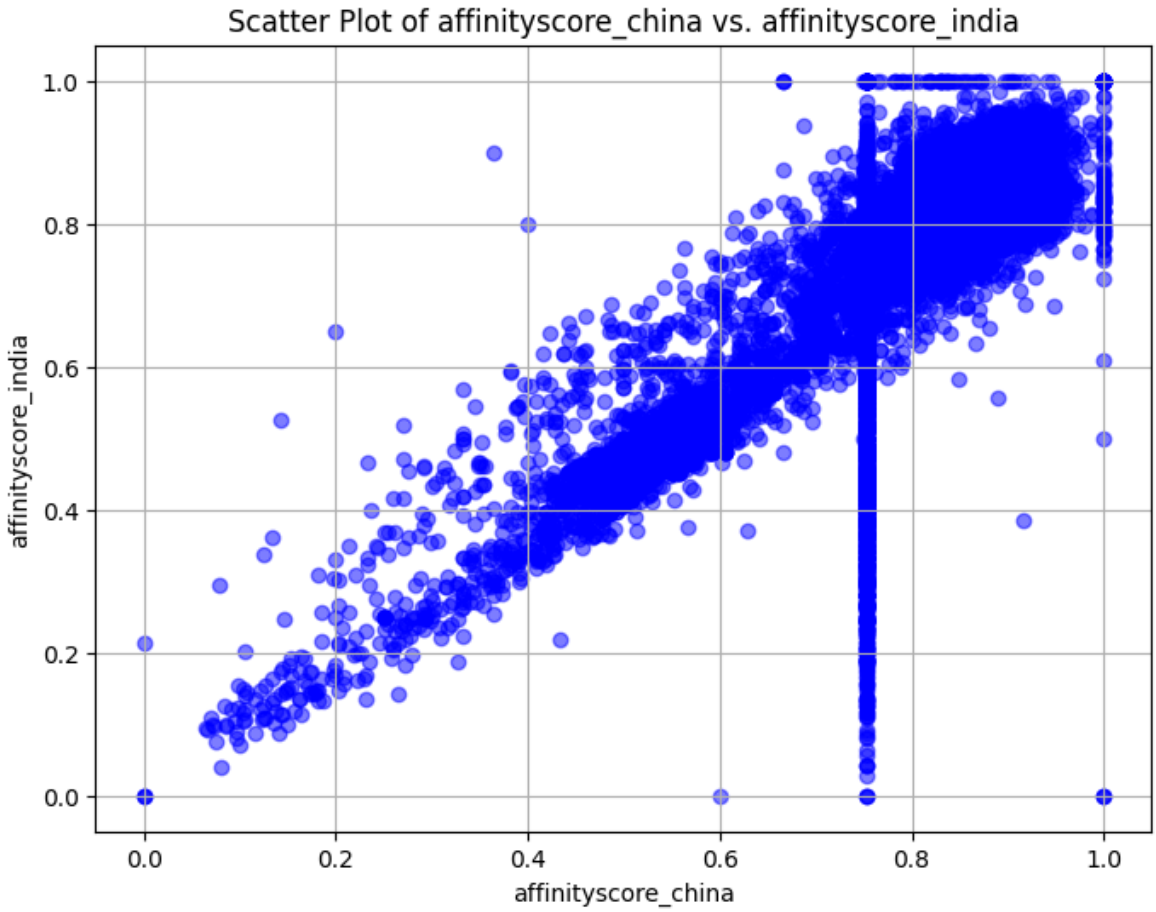
value is called the independent variable.



In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Assuming df is your DataFrame containing the data

# Data preprocessing
X = df['affinityscore_china']  # Features
y = df['affinityscore_india']  # Target

# Reshape X and y to be two-dimensional arrays
X = X.values.reshape(-1, 1)  # Reshape X to a column vector
y = y.values.reshape(-1, 1)  # Reshape y to a column vector

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Model training
model = LinearRegression()
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.015783713134011237
```

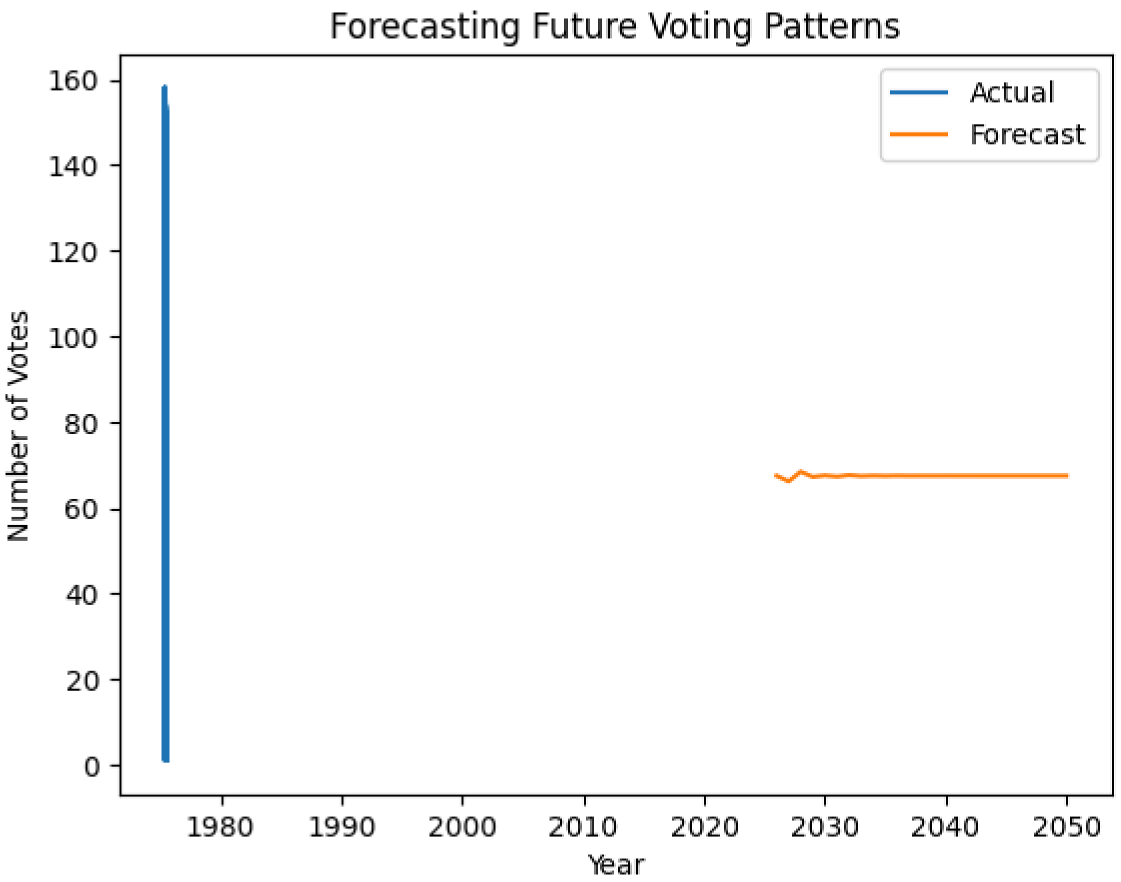
In [ ]:
```
import matplotlib.pyplot as plt

# Plotting the relationship between the independent variable (X) and the
plt.figure(figsize=(8, 6))
plt.scatter(X, y, color='blue', alpha=0.5)
plt.title('Scatter Plot of affinityscore_china vs. affinityscore_india')
plt.xlabel('affinityscore_china')
plt.ylabel('affinityscore_india')
plt.grid(True)
plt.show()
```



In [ ]:

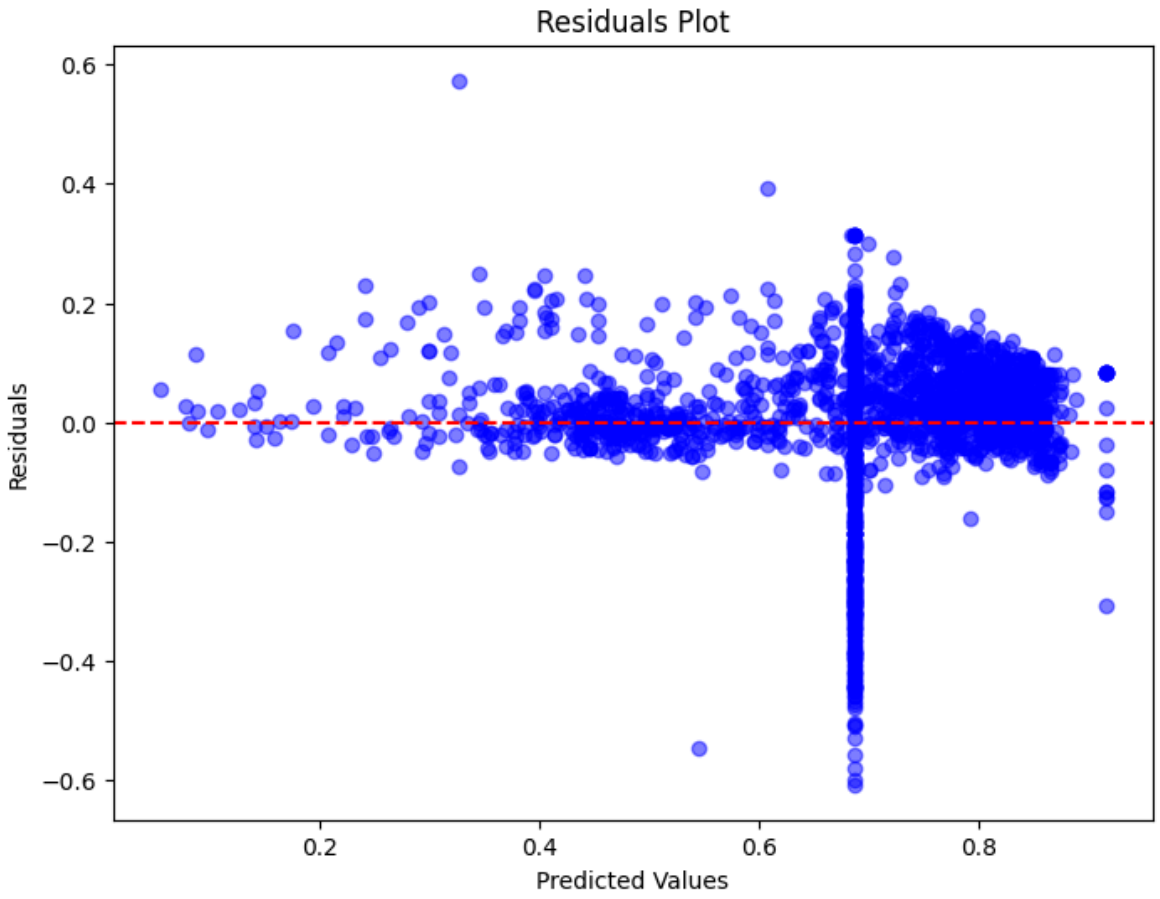
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Assuming 'year' is the time variable and 'all_votes' is the target vari
ts_data = df[['year', 'all_votes']]
ts_data.set_index('year', inplace=True)

# Model training
model = ARIMA(ts_data, order=(5,1,0))
fit_model = model.fit()

# Forecasting
forecast_index = pd.date_range(start='2025', end='2050', freq='Y')  # Gen
forecast = fit_model.forecast(steps=len(forecast_index))  # Forecasting u

# Plotting forecast
plt.plot(ts_data, label='Actual')
plt.plot(forecast_index, forecast, label='Forecast')
plt.title('Forecasting Future Voting Patterns')
plt.xlabel('Year')
plt.ylabel('Number of Votes')
plt.legend()
plt.show()
```

```
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-p
ackages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupporte
d index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-p
ackages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupporte
d index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-p
ackages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupporte
d index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/var/folders/4y/p5wbs2392ng1qdh47g8kk6x00000gn/T/ipykernel_38761/147541855
6.py:14: FutureWarning: 'Y' is deprecated and will be removed in a future
version, please use 'YE' instead.
  forecast_index = pd.date_range(start='2025', end='2050', freq='Y')  # Ge
nerating date range for forecasting
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-p
ackages/statsmodels/tsa/base/tsa_model.py:836: ValueWarning: No supported
index is available. Prediction results will be given with an integer index
beginning at `start`.
  return get_prediction_index(
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-p
ackages/statsmodels/tsa/base/tsa_model.py:836: FutureWarning: No supported
index is available. In the next version, calling this method in a model wi
thout a supported index will result in an exception.
  return get_prediction_index(
```



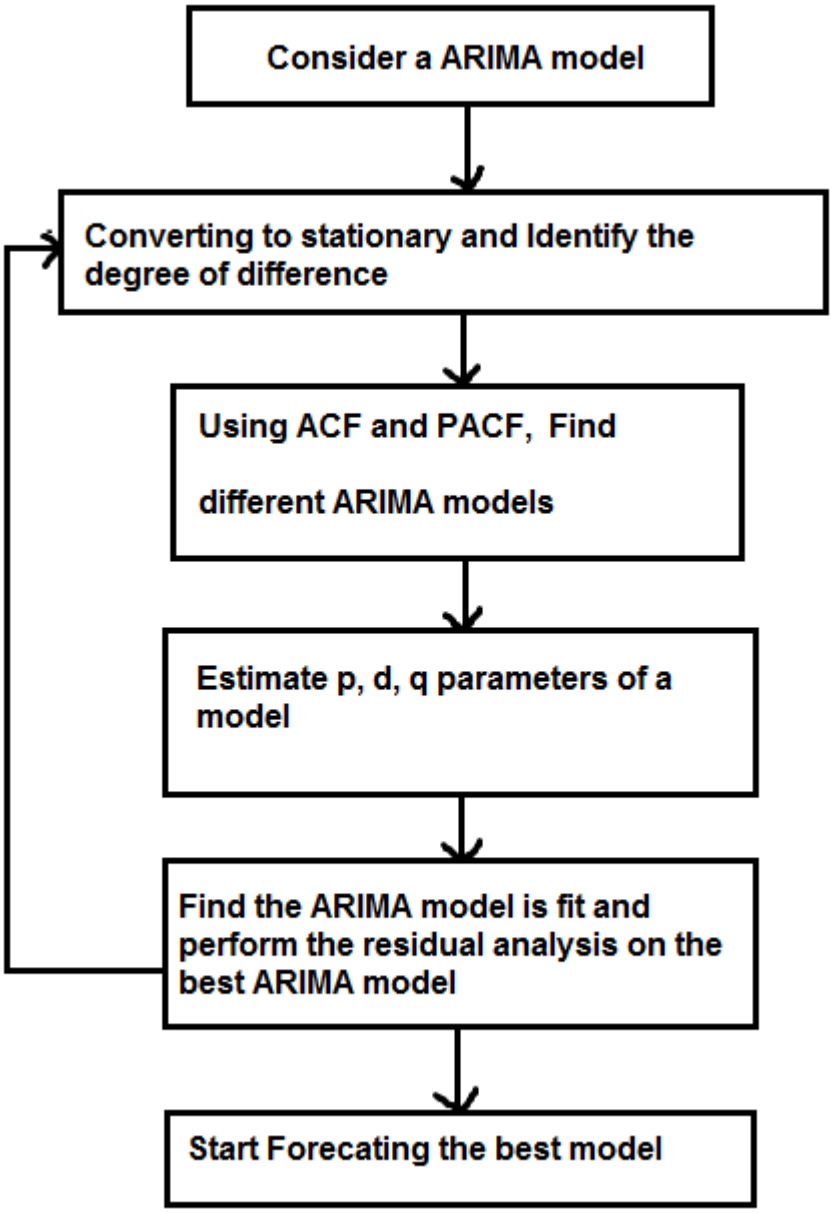
In [ ]:
```
# Calculate residuals
residuals = y_test - y_pred

# Plot residuals
plt.figure(figsize=(8, 6))
```

```
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residuals Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```



ARIMA(AutoRegressive Integrated Moving Average)

- An autoregressive integrated moving average, or ARIMA, is a statistical analysis model that uses time series data to either better understand the data set or to predict future trends.
- A statistical model is autoregressive if it predicts future values based on past values. For example, an ARIMA model might seek to predict a stock's future prices based on its past performance or forecast a company's earnings based on past periods.



In [ ]:

```
from sklearn.metrics import mean_squared_error
import numpy as np

# Splitting the dataset into training and testing sets
train_size = int(len(target_data) * 0.8)  # Using 80% of the data for tra
train_data, test_data = target_data.iloc[:train_size], target_data.iloc[t

# Model training
model = ARIMA(train_data, order=(5, 1, 0))
fit_model = model.fit()

# Forecasting
forecast = fit_model.forecast(steps=len(test_data))  # Forecasting on the

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(test_data, forecast)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
```

```
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
Mean Squared Error (MSE): 1145.1768315695354
Root Mean Squared Error (RMSE): 33.84046145621444
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:836: ValueWarning: No supported index is available. Prediction results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:836: FutureWarning: No supported index is available. In the next version, calling this method in a model without a supported index will result in an exception.
  return get_prediction_index(
```

In [ ]:

```
from sklearn.metrics import mean_squared_error
import numpy as np

# Splitting the dataset into training and testing sets
train_size = int(len(target_data) * 0.8)  # Using 80% of the data for tra
train_data, test_data = target_data.iloc[:train_size], target_data.iloc[t

# Model training
model = ARIMA(train_data, order=(5, 1, 0))
fit_model = model.fit()

# Forecasting
forecast = fit_model.forecast(steps=len(test_data))  # Forecasting on the

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(test_data, forecast)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)

# Print the forecast
print("Forecast:")
print(forecast)
```

```
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

```
Mean Squared Error (MSE): 1145.1768315695354
Root Mean Squared Error (RMSE): 33.84046145621444
Forecast:
7757    64.542714
7758    64.554631
7759    65.089764
7760    64.674080
7761    64.790490
          ...
9692    64.779412
9693    64.779412
9694    64.779412
9695    64.779412
9696    64.779412
Name: predicted_mean, Length: 1940, dtype: float64
```

```
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:836: ValueWarning: No supported index is available. Prediction results will be given with an integer index beginning at `start`.
  return get_prediction_index(
/Users/shrutimall/.local/pipx/.cache/5c9468f9a0a782a/lib/python3.12/site-packages/statsmodels/tsa/base/tsa_model.py:836: FutureWarning: No supported index is available. In the next version, calling this method in a model without a supported index will result in an exception.
  return get_prediction_index(
```

In [ ]: