project

December 27, 2025

1 About the Project

1.1 Overview

The project involves the statistical analysis of the official results of the Intra-School Council Elections using libraries like Pandas, Matplotlib & Seaborn and SQL.

The student council elections are held every year to choose representatives for different leadership roles in the school and the election process is managed entirely by student-led tech teams.

1.2 Elections

The elections were held in school using a custom made *Election Software* (1), a fullstack application that I designed and developed featuring:

- A frontend built with HTML, CSS, and TypeScript
- A REST API using FastAPI (Python)
- Votes securely stored in a MongoDB database

1.3 Report Generation

This report is a single Jupyter notebook exported via a custom script to HTML and styled with CSS (and then printed).
Source available at *GitHub Repo*(2).

1.4 References

1. https://github.com/d1vij/electionsoftware

2. https://github.com/d1vij/ip-proj

1.5 Libraries Used

1. *Pymongo* → for querying vote documents from MongoDB server

2. *sqlite3* → for querying local SQL database

3. *pandas* → for data manipulation and analysis
4. *seaborn* → graphing
5. *matplotlib* → graphing

1.5.1 Importing Stuff

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme()
```

1.5.2 Defining Constants

1.6 Structure of Data

Each vote session is stored as a single MongoDB document and the documents are seggegated into collections based on the class of the student which casted the vote

A vote session represents all votes cast by a single client in one voting attempt.

Feilds 1. *client* - Unique Identifier of the device that vote was casted on

2. *token*
   - UUID to identify the vote session. Each *valid* vote has a corresponding unique token
3. *vote_data*
   - Array of vote objects
   - Each vote object contains the name of candidate and the post the vote is casted for

```
// example single vote document
{
    "_id": {
        "$oid": "68a1819ceff178ec25b66fbb"           // internal mongodb document id
    },
    "token": "b489737f-7997-430c-950f-b8c1b22f68c3", // A unique uuid4 token identifying the v
    "client": "29",                                  // Computer on which the vote was casted
    "vote_data": [                                   // Candidates voted by the voter
        {
            "name": "Abhichandra Charke",
            "post": "Captain Boy"
        },
        {
            "name": "Gauravi Zade",
            "post": "Captain Girl"
        },
        {
            "name": "Kausar Chandra",
```

```
            "post": "Vice Captain Boy"
        },
        {
            "name": "Ketaki Phalle",
            "post": "Vice Captain Girl"
        }
    ]
}
```

[2]:
```python
# Fetches all the documents of all classes from the mongodb cluster
# Returns an array of dictionaries containing the feilds for name of class and␣
 ↪array of vote sessions
def get_classwise_documents(
    connection_url: str, database_name: str, classes: list[str]
) -> list[dict]:
    import pymongo

    conn = pymongo.MongoClient(connection_url)
    database = conn.get_database(database_name)
    all_documents: list[dict] = []
    vote_document: dict
    for class_name in classes:
        class_documents: list[dict] = []
        collection = database.get_collection(class_name)

        for vote_document in collection.find({}):
            class_documents.append(vote_document["vote_data"])  # type: ignore

        all_documents.append({"name": class_name, "votes": class_documents})

    return all_documents
```

```
[
  {
    "name": "10A",
    "votes": [
      [
        { "name": "Aadityaraje Desai", "post": "Captain Boy" },
        { "name": "Tvisha Shah", "post": "Captain Girl" },
        { "name": "Kausar Chandra", "post": "Vice Captain Boy" },
        { "name": "Ketaki Phalle", "post": "Vice Captain Girl" }
      ],
      [
        { "name": "Abhichandra Charke", "post": "Captain Boy" },
        { "name": "Tvisha Shah", "post": "Captain Girl" },
        { "name": "Sagnik Ghosh", "post": "Vice Captain Boy" },
        { "name": "Ketaki Phalle", "post": "Vice Captain Girl" }
      ],
```

```
    [
      { "name": "Aadityaraje Desai", "post": "Captain Boy" },
      { "name": "Tvisha Shah", "post": "Captain Girl" },
      { "name": "Avaneesh Mahalle", "post": "Vice Captain Boy" },
      { "name": "Trisha Kandpal", "post": "Vice Captain Girl" }
    ],
    [
      { "name": "Aadityaraje Desai", "post": "Captain Boy" },
      { "name": "Gauravi Zade", "post": "Captain Girl" },
      { "name": "Kausar Chandra", "post": "Vice Captain Boy" },
      { "name": "Riya Shirode", "post": "Vice Captain Girl" }
...
```

[3]:
```
CONNECTION_URL = "mongodb+srv://vermadivij:elections@cluster1.######.mongodb.
 ↪net/"
CLASSES = ['10A', '10B', '10C', '10D', '10E', '10F', '10G', '10H', '10I',␣
 ↪'10J', '11A', '11B', '11C', '11D', '11E', '12A', '12B', '12C', '12D', '9A',␣
 ↪'9B', '9C', '9D', '9E', '9F', '9G', '9H', '9I', '9J', 'absentees',␣
 ↪'candidates']  # fmt: off
CLUSTER="votes"

def download_results():
    import json
    documents = get_classwise_documents(CONNECTION_URL, "votes", CLASSES)
    with open("votes.json", "w+") as file:
        file.write(json.dumps(documents))
```

1.7 Compiling Data

[4]:
```
# fmt: off
candidate_data = {
    "Captain Boy": [ "Aadityaraje Desai", "Abhichandra Charke", "Praneel␣
 ↪Deshmukh", "Rachit Srivastava", ],
    "Captain Girl": [ "Tvisha Shah", "Gauravi Zade", "Kirthika Jayachander",␣
 ↪"Naisha Rastogi", ],
    "Vice Captain Boy": [ "Kausar Chandra", "Sagnik Ghosh", "Avaneesh Mahalle",␣
 ↪"Krishna Yadav", "Viren Jadhav", ],
    "Vice Captain Girl": [ "Ketaki Phalle", "Trisha Kandpal", "Riya Shirode",␣
 ↪"Kavya Mehta", "Sumedha Vaidya", ],
}
# fmt: on
import json
import pandas as pd

def get_classwise_documents_from_local(votes_json: str) -> list[dict]:
    with open(votes_json) as file:
        return json.loads(file.read())
```

```
def calculate_votes(votes_json: str) -> pd.DataFrame:
    classwise_votes = get_classwise_documents_from_local(votes_json)
    votes_df = pd.DataFrame(
        [
            {
                "class": _class["name"],
                "candidate_name": candidate["name"],
                "post": candidate["post"],
            }
            for _class in classwise_votes
            for votes in _class["votes"]
            for candidate in votes
        ],
        columns=["class", "candidate_name", "post"],  # type: ignore
    )
    return votes_df


votes_df = calculate_votes("votes.json")
print(votes_df.head(10))
```

```
  class      candidate_name              post
0   10A   Aadityaraje Desai       Captain Boy
1   10A         Tvisha Shah      Captain Girl
2   10A      Kausar Chandra  Vice Captain Boy
3   10A       Ketaki Phalle Vice Captain Girl
4   10A  Abhichandra Charke       Captain Boy
5   10A         Tvisha Shah      Captain Girl
6   10A        Sagnik Ghosh  Vice Captain Boy
7   10A       Ketaki Phalle Vice Captain Girl
8   10A   Aadityaraje Desai       Captain Boy
9   10A         Tvisha Shah      Captain Girl
```

[5]:
```
# Classwise dataframes
classwise_grouped = votes_df.groupby("post")

cb = votes_df.loc[classwise_grouped.groups["Captain Boy"]].drop("post", axis=1).
 ↪reset_index(drop=True)
cg = votes_df.loc[classwise_grouped.groups["Captain Girl"]].drop( "post",␣
 ↪axis=1 ).reset_index(drop=True)
vcb = votes_df.loc[classwise_grouped.groups["Vice Captain Boy"]].drop( "post",␣
 ↪axis=1 ).reset_index(drop=True)
vcg = votes_df.loc[classwise_grouped.groups["Vice Captain Girl"]].drop( "post",␣
 ↪axis=1 ).reset_index(drop=True)

postwise_votes_df = {
```

```
    "captain_boy": cb,
    "captain_girl": cg,
    "vice_captain_boy": vcb,
    "vice_captain_girl": vcg,
}
```

1.8 SQL Querying

SqliteDatabase is a wrapper around the `sqlite3` library which allows for quick and hasslefree SQL querying of the data.

[6]:
```
import sqlite3
from sqlite3 import Connection, Cursor
from typing import Any, Literal


class SqliteDatabase:
    def __init__(self, database: str):
        self.database = database
        self.conn: Connection | None = None
        self.cursor: Cursor | None = None

    def __enter__(self):
        try:
            self.conn = sqlite3.connect(self.database)
            self.cursor = self.conn.cursor()
        except Exception as e:
            print(f"Error occured in connecting to the database {self.database}.
 ↪ Error Details: {e}")

        return self.query

    def __exit__(self, exc_type, exc, tb):
        assert self.conn is not None
        assert self.cursor is not None

        self.cursor.close()
        self.conn.close()

        return False  # dont suppress the error

    def query(
        self,
        query: str,
        *,
        is_updation=False,  # is the current query contains some kind of␣
 ↪updation ?? Doesnt return anything if true
```

```
        return_rows: None | Literal["str"] | Literal["tuple"] = None,
        table_heading: str | None = None,  # Title printed before printing␣
↪output
    ) -> None | tuple[tuple[str, ...], ...]:
        assert self.conn is not None
        assert self.cursor is not None

        try:
            results = self.cursor.execute(query)
            self.conn.commit()
        except Exception as err:
            print("** Row / Column names with spaces should be enlcosed within␣
↪quotes **")
            raise err

        if is_updation:
            return

        rows: list[Any] = results.fetchall()
        columns_headers: tuple[str, ...] = tuple(str(col[0]) for col in results.
↪description)

        lines: tuple[tuple[str, ...], ...] = tuple((columns_headers, *rows))

        if return_rows == "tuple":
            return lines
        elif return_rows is None:
            # printing table header if provided
            if table_heading is not None:
                print(table_heading)

            # Finding max column width
            column_widths: list[int] = []

            for col_idx in range(len(lines[0])):
                widths = []
                for row_idx in range(len(lines)):
                    widths.append(len(str(lines[row_idx][col_idx])))
                column_widths.append(max(widths))

            # Printing column headers
            border_top_bottom = "+" + "-" * (sum(column_widths) + 3 *␣
↪len(column_widths) - 1) + "+"
            print(border_top_bottom)
            print("| ", end="")
            for idx, col_label in enumerate(lines[0]):
                print(str(col_label).ljust(column_widths[idx]), end=" | ")
```

```
            print()
            print(border_top_bottom)


            for row in rows:
                print("| ", end="")
                for idx, col_val in enumerate(row):
                    print(str(col_val).ljust(column_widths[idx]), end=" | ")
                print()

            print(border_top_bottom)
            return None
```

1.9 Creating post dataframes and saving them to SQLite database

[7]:
```
with SqliteDatabase("votes.db") as query:
    for post_name, post_df in postwise_votes_df.items():
        query(f"drop table if exists {post_name};", is_updation=True)
        query(
            f"create table {post_name} (class varchar(255), candidate_name␣
 ↪varchar(255));",
            is_updation=True,
        )

        rows = []
        for (idx, row) in post_df.iterrows():
            (class_name, candidate_name) = (row["class"], row["candidate_name"])
            rows.append(f"('{class_name}', '{candidate_name}')")

        query(f"insert into {post_name} values" + ",".join(rows),␣
 ↪is_updation=True)
        query(f"select * from {post_name} limit 10", table_heading=post_name)
```

```
captain_boy
+---------------------------+
| class | candidate_name    |
+---------------------------+
| 10A   | Adityaraje Desai  |
| 10A   | Abhichandra Charke |
| 10A   | Adityaraje Desai  |
| 10A   | Adityaraje Desai  |
| 10A   | Abhichandra Charke |
| 10A   | Abhichandra Charke |
| 10A   | Praneel Deshmukh  |
| 10A   | Adityaraje Desai  |
| 10A   | Praneel Deshmukh  |
| 10A   | Adityaraje Desai  |
```

```
+---------------------------+
captain_girl
+-------------------------------+
| class | candidate_name        |
+-------------------------------+
| 10A   | Tvisha Shah           |
| 10A   | Tvisha Shah           |
| 10A   | Tvisha Shah           |
| 10A   | Gauravi Zade          |
| 10A   | Gauravi Zade          |
| 10A   | Gauravi Zade          |
| 10A   | Gauravi Zade          |
| 10A   | Kirthika Jayachander  |
| 10A   | Gauravi Zade          |
| 10A   | Naisha Rastogi        |
+-------------------------------+
vice_captain_boy
+--------------------------+
| class | candidate_name   |
+--------------------------+
| 10A   | Kausar Chandra   |
| 10A   | Sagnik Ghosh     |
| 10A   | Avaneesh Mahalle |
| 10A   | Kausar Chandra   |
| 10A   | Krishna Yadav    |
| 10A   | Kausar Chandra   |
| 10A   | Krishna Yadav    |
| 10A   | Avaneesh Mahalle |
| 10A   | Avaneesh Mahalle |
| 10A   | Krishna Yadav    |
+--------------------------+
vice_captain_girl
+------------------------+
| class | candidate_name |
+------------------------+
| 10A   | Ketaki Phalle  |
| 10A   | Ketaki Phalle  |
| 10A   | Trisha Kandpal |
| 10A   | Riya Shirode   |
| 10A   | Ketaki Phalle  |
| 10A   | Kavya Mehta    |
| 10A   | Ketaki Phalle  |
| 10A   | Trisha Kandpal |
| 10A   | Trisha Kandpal |
| 10A   | Trisha Kandpal |
+------------------------+
```

1.10 Statistical Analysis

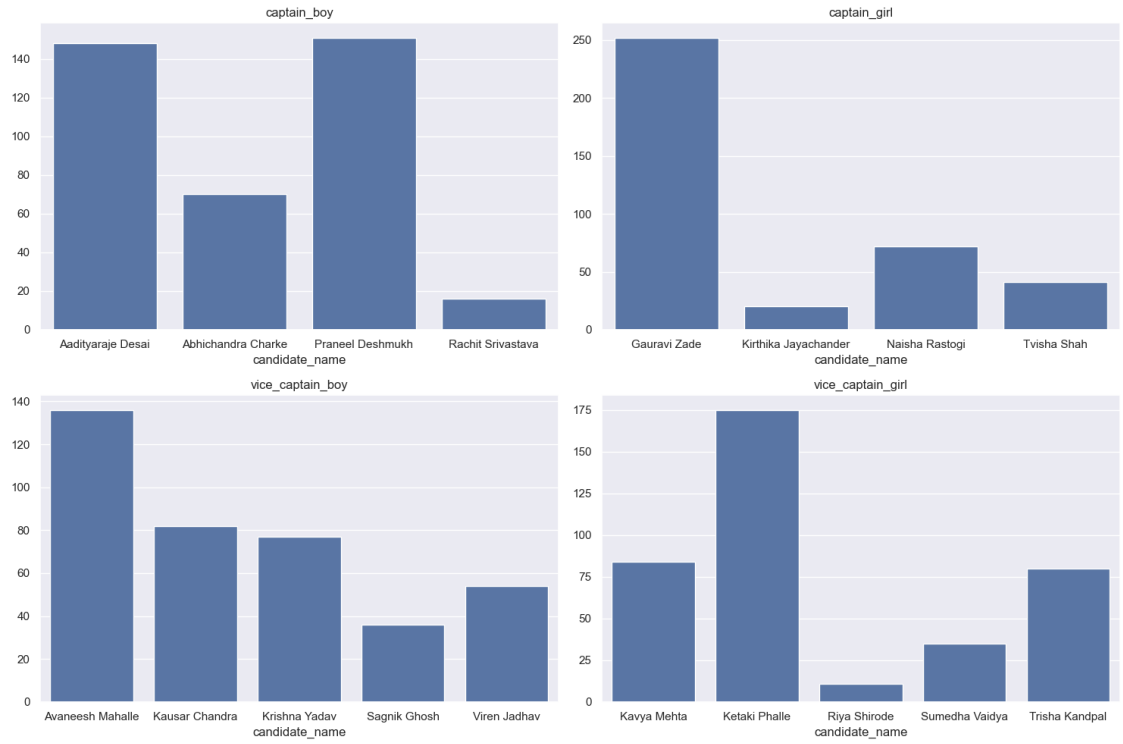
1.10.1 Total Votes across all classes

[8]:
```
fig, axes = plt.subplots(2,2, figsize=(15,10))
positions = [(0,0), (0,1), (1,0), (1,1)]

for idx, (post_name, post_df) in enumerate(postwise_votes_df.items()):
    with SqliteDatabase("votes.db") as query:
        query(
            f"""
            select candidate_name as Name, count(*) as Votes
            from {post_name}
            group by candidate_name
            order by Votes desc;
            """,
            table_heading="Total Votes for - " + post_name
        )
    print()
    votes = post_df.groupby("candidate_name").size()
    sns.barplot(votes, ax=axes[positions[idx]]) #type: ignore
    axes[positions[idx]].set_title(post_name)
plt.tight_layout()
plt.show()
print()
```

```
Total Votes for - captain_boy
+---------------------------+
| Name               | Votes |
+---------------------------+
| Praneel Deshmukh   | 151   |
| Aadityaraje Desai  | 148   |
| Abhichandra Charke | 70    |
| Rachit Srivastava  | 16    |
+---------------------------+

Total Votes for - captain_girl
+-----------------------------+
| Name                 | Votes |
+-----------------------------+
| Gauravi Zade         | 252   |
| Naisha Rastogi       | 72    |
| Tvisha Shah          | 41    |
| Kirthika Jayachander | 20    |
+-----------------------------+

Total Votes for - vice_captain_boy
+--------------------------+
| Name             | Votes |
```

```
+--------------------------+
| Avaneesh Mahalle | 136   |
| Kausar Chandra   | 82    |
| Krishna Yadav    | 77    |
| Viren Jadhav     | 54    |
| Sagnik Ghosh     | 36    |
+--------------------------+
```

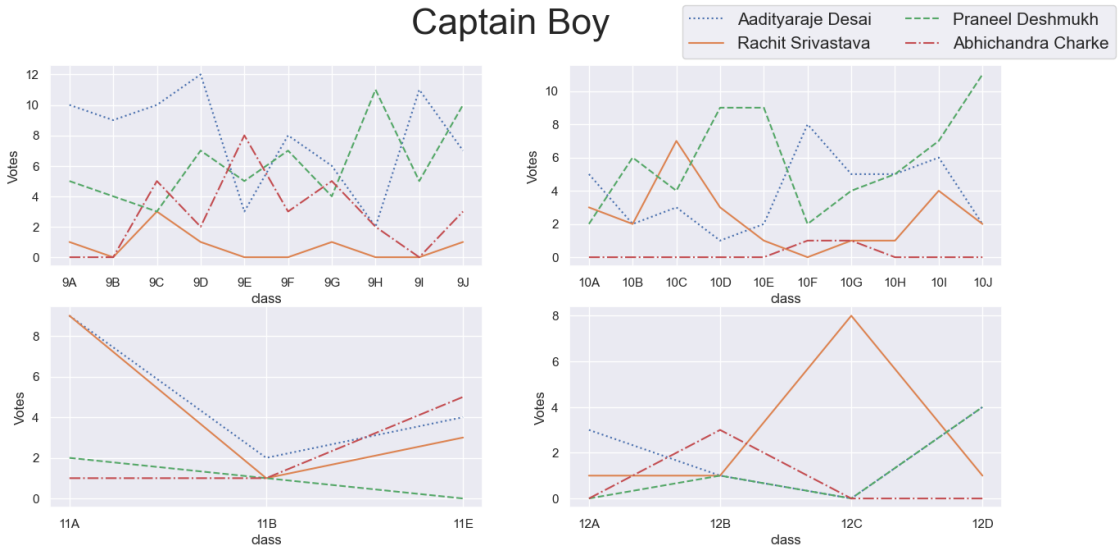
```
Total Votes for - vice_captain_girl
+------------------------+
| Name           | Votes |
+------------------------+
| Ketaki Phalle  | 175   |
| Kavya Mehta    | 84    |
| Trisha Kandpal | 80    |
| Sumedha Vaidya | 35    |
| Riya Shirode   | 11    |
+------------------------+
```

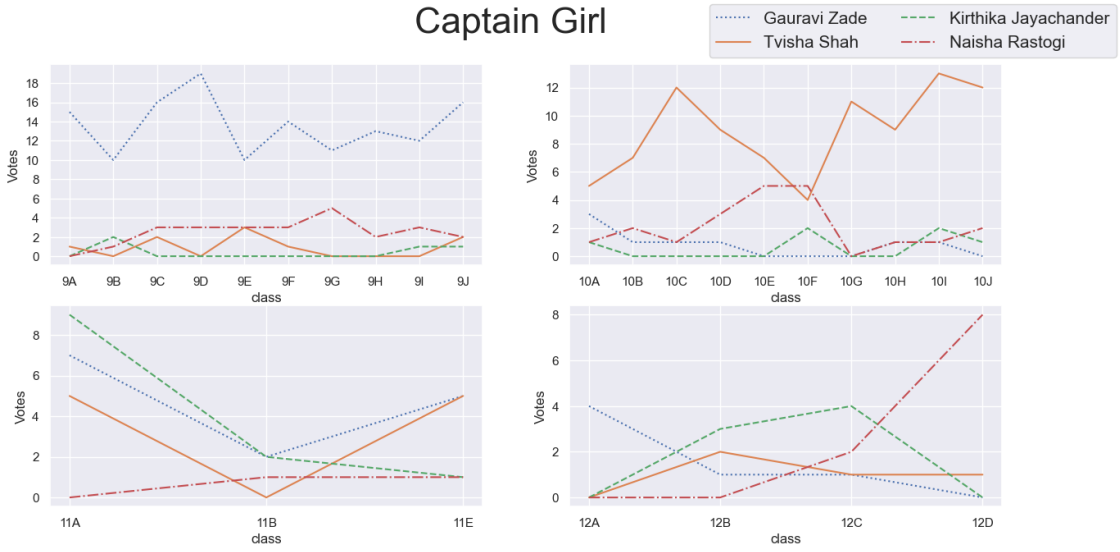


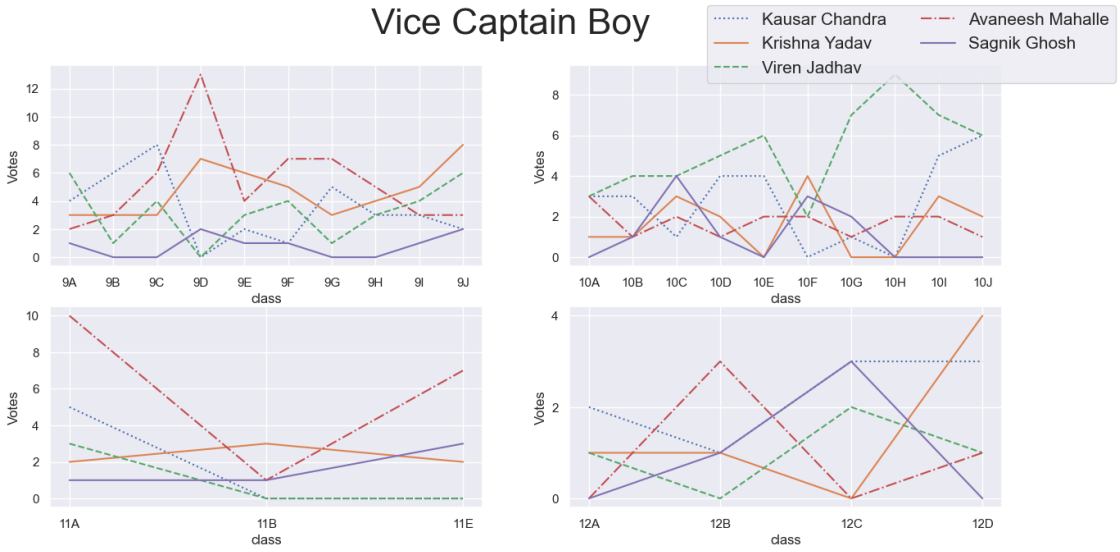
1.11 Candidate Popularity trends

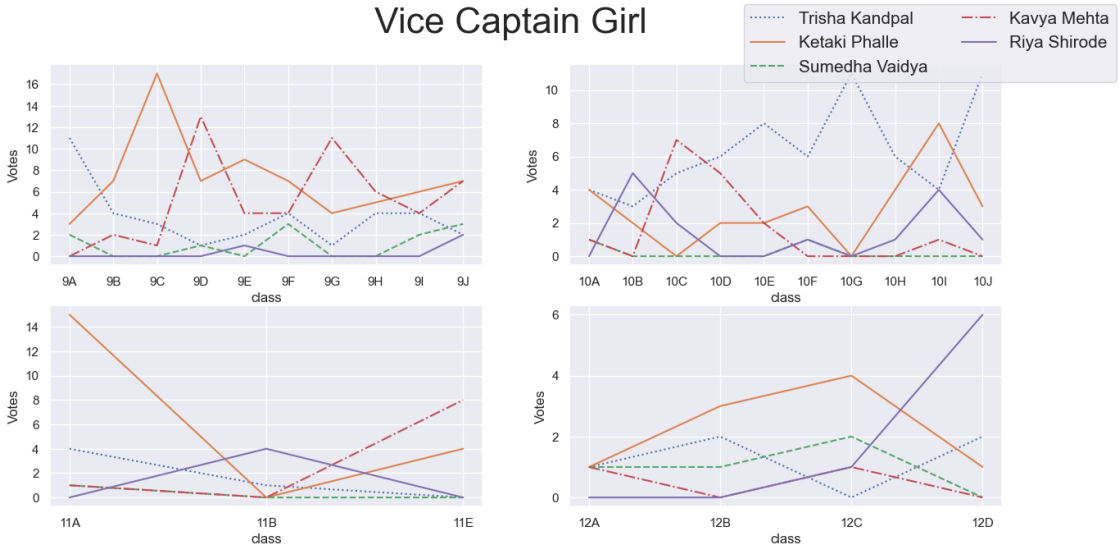
[9]:
```
# candidate popularity trends - comparing candidate performances across classes

from matplotlib.ticker import MultipleLocator


def plot_popularity_trends(post_name: str, post_df: pd.DataFrame):

    # extracting rows belonging to a particular class from the post's dataframe␣
 ↪using regular expressions
    class_wise_dataframes = [
        post_df.loc[post_df["class"].str.contains(_re)]
        for _re in [r"9\w", r"10\w", r"11\w", r"12\w"]  # <--- regex btw
    ]

    # dividing the plot into 4 subplots
    fig, axes = plt.subplots(2, 2, figsize=(15, 7))

    subplot_positions = [
        (0, 0),
        (0, 1),
        (1, 0),
        (1, 1),
    ]  # since there are only 4 classes / subplots
    linestyles = [":", "-", "--", "-.", "solid"]

    for idx in range(4):
        pos = subplot_positions[idx]

        class_df = class_wise_dataframes[idx]

        # series of all sections
        sections = class_df["class"].unique()
        candidates = class_df["candidate_name"].unique()

        # Group all the votes by class, then candidate
        # Create a new column with the total size of each group (ie votes in␣
 ↪that class) and name it votes
        classwise_votes = class_df.groupby(["class", "candidate_name"]).size().
 ↪reset_index(name="votes")
                                    # ^^^ Groupby forms a series with a␣
 ↪multi-index
        for idx, candidate_name in enumerate(candidates):
            candidate_votes = (
                classwise_votes
                    # Select all the votes for a given candidate
```

```
                    .loc[classwise_votes["candidate_name"] == candidate_name]
                    # Set the "class" column as the index and take only the␣
 ↪votes column
                    .set_index("class")["votes"]
                    # Ensure that the candidate has entries for each section,
                    # fill the missing ones with 0
                    .reindex(sections, fill_value=0)
            )
            # plotting a subplot for each class
            axes[pos].plot(
                sections,
                candidate_votes,
                label=candidate_name,
                linestyle=linestyles[idx],
            )

        axes[pos].set_xlabel("class")
        axes[pos].set_ylabel("Votes")

        # axes[pos].set_ylim(0, post_df.max().max() + 1)

        # values on y-axis would have a difference of 2
        axes[pos].yaxis.set_major_locator(MultipleLocator(2))

    fig.suptitle(post_name, fontsize=32)

    # setting a common legend for the whole plot
    handles, labels = axes[0, 0].get_legend_handles_labels()
    fig.legend(handles, labels, loc="upper right", ncols=2, fontsize=15)

    plt.show()


plot_popularity_trends("Captain Boy", cb)
plot_popularity_trends("Captain Girl", cg)
plot_popularity_trends("Vice Captain Boy", vcb)
plot_popularity_trends("Vice Captain Girl", vcg)
```


Captain Boy
Aadityaraje Desai
Rachit Srivastava
Praneel Deshmukh
Abhichandra Charke
Votes
class
9A 9B 9C 9D 9E 9F 9G 9H 9I 9J
10A 10B 10C 10D 10E 10F 10G 10H 10I 10J
11A 11B 11E
12A 12B 12C 12D


Captain Girl
Gauravi Zade
Tvisha Shah
Kirthika Jayachander
Naisha Rastogi
Votes
class
9A 9B 9C 9D 9E 9F 9G 9H 9I 9J
10A 10B 10C 10D 10E 10F 10G 10H 10I 10J
11A 11B 11E
12A 12B 12C 12D



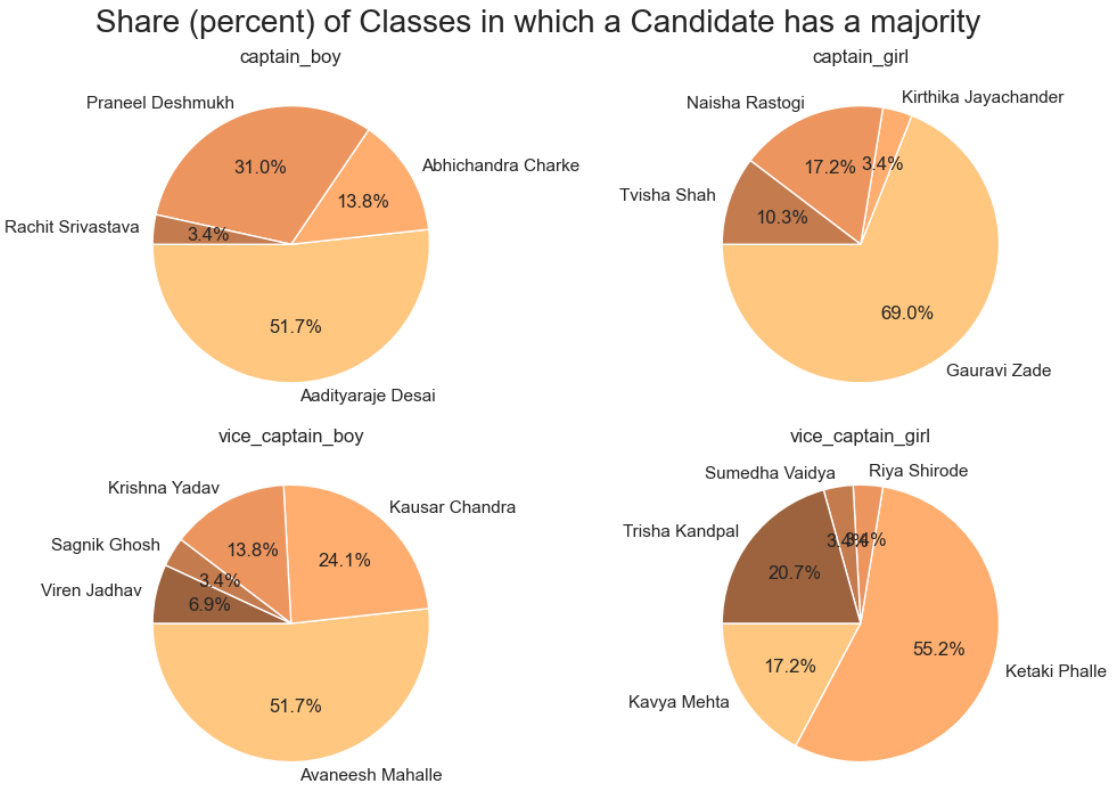


1.12 Majority Share

The share in percent of classes in which a candidate has majority

[10]:
```
total_classes = len(cb["class"].unique())

fig, axes = plt.subplots(2, 2, figsize=(10, 7), constrained_layout=True)
fig.suptitle("Share (percent) of Classes in which a Candidate has a majority
↪", fontsize=20)
```

```
subplot_positions = [
        (0, 0),
        (0, 1),
        (1, 0),
        (1, 1),
    ]

colors = plt.cm.copper_r(np.linspace(0,0.50,5)) # type: ignore


for idx, (post_name, post_df) in enumerate(postwise_votes_df.items()):
    pos = subplot_positions[idx]
    # kitne classes me each candidate has the maximum votes

    votes = post_df.value_counts(["class", "candidate_name"]).
 ↪reset_index(name="votes")
    winners = votes.loc[votes.groupby("class")["votes"].idxmax()]
    count_series = winners.groupby("candidate_name").size()
    percents_series = count_series / total_classes

    max_val = percents_series.max()

    axes[pos].set_title(post_name)
    axes[pos].pie(
        percents_series,
        labels=percents_series.index.map(
            lambda name: name.replace("_", " ")
        ),
        autopct="%1.1f%%",
        startangle=180,
        colors=colors,
    )
    axes[pos].set(aspect='equal')


plt.show()
```



Analyzes whether voters who supported one candidate also tended to support another.

Working:

1. Builds a `vote session` dataframe recording which candidates were chosen in each voting session.

```
# Example

Captain Boy          Captain Girl     Vice Captain Boy     Vice Captain Girl     class

25  Praneel Deshmukh  Gauravi Zade      Viren Jadhav        Kavya Mehta           10C
26  Aadityaraje Desai Gauravi Zade      Avaneesh Mahalle    Kavya Mehta           10C
27  Abhichandra Charke Gauravi Zade     Avaneesh Mahalle    Ketaki Phalle         10C
28  Praneel Deshmukh  Gauravi Zade      Sagnik Ghosh        Sumedha Vaidya         10C
29  Abhichandra Charke Gauravi Zade     Sagnik Ghosh        Sumedha Vaidya         10C
30  Abhichandra Charke Gauravi Zade     Avaneesh Mahalle    Ketaki Phalle         10C
```

2. Constructs a `co-occurrence matrix` showing how often Candidate B was voted when Candidate A was voted.

```
# Example
                    Gauravi Zade  Kirthika Jayachander  Naisha Rastogi
Avaneesh Mahalle              94                     4              26
```

Krishna Yadav	54	3	14
Viren Jadhav	33	2	11
Ketaki Phalle	116	5	38
Trisha Kandpal	52	5	15
Riya Shirode	5	3	0
Kavya Mehta	52	3	18
Sumedha Vaidya	27	4	1

3. Normalizes it into a `probability matrix` to estimate the likelihood of co-support between candidates. Each row of the co-occurrence matrix is divided by the total votes in that row. This converts raw counts into conditional probabilities, i.e., the chance of Candidate B being voted given that Candidate A was voted.

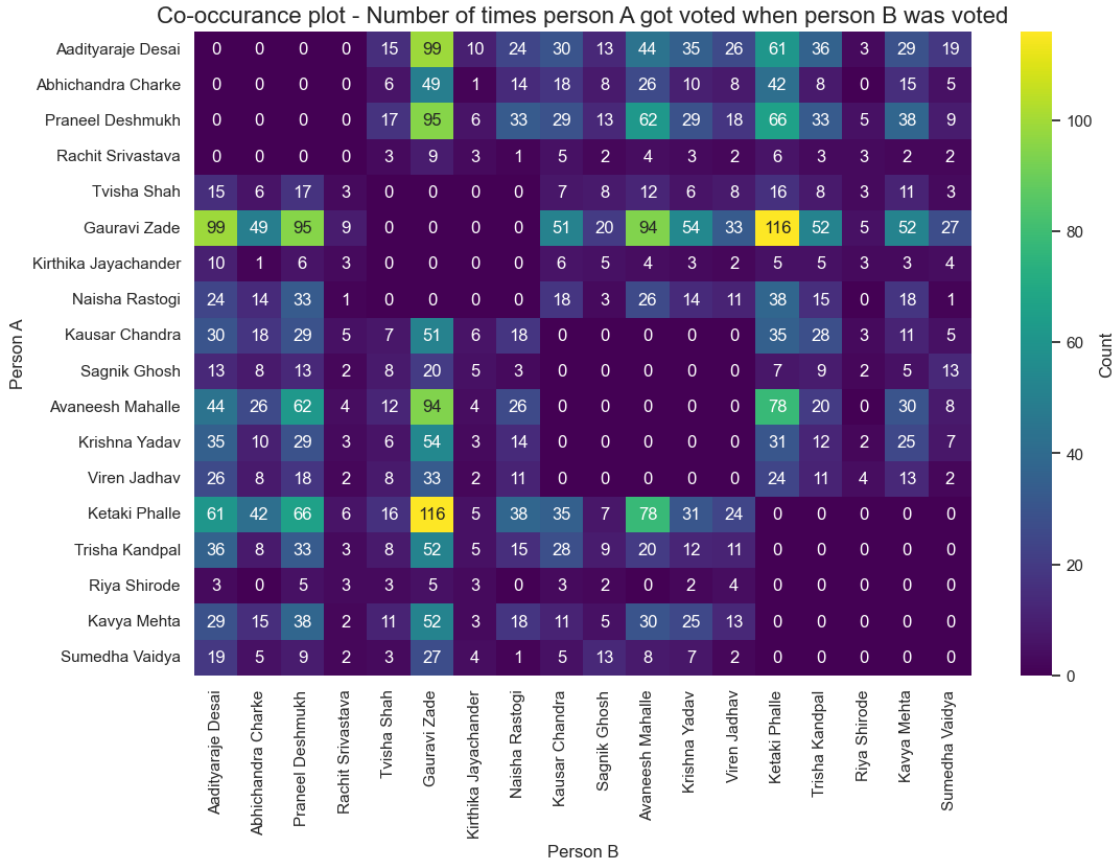
4. Visualizes both matrices using heatmaps — one for raw counts, the other for probabilities.

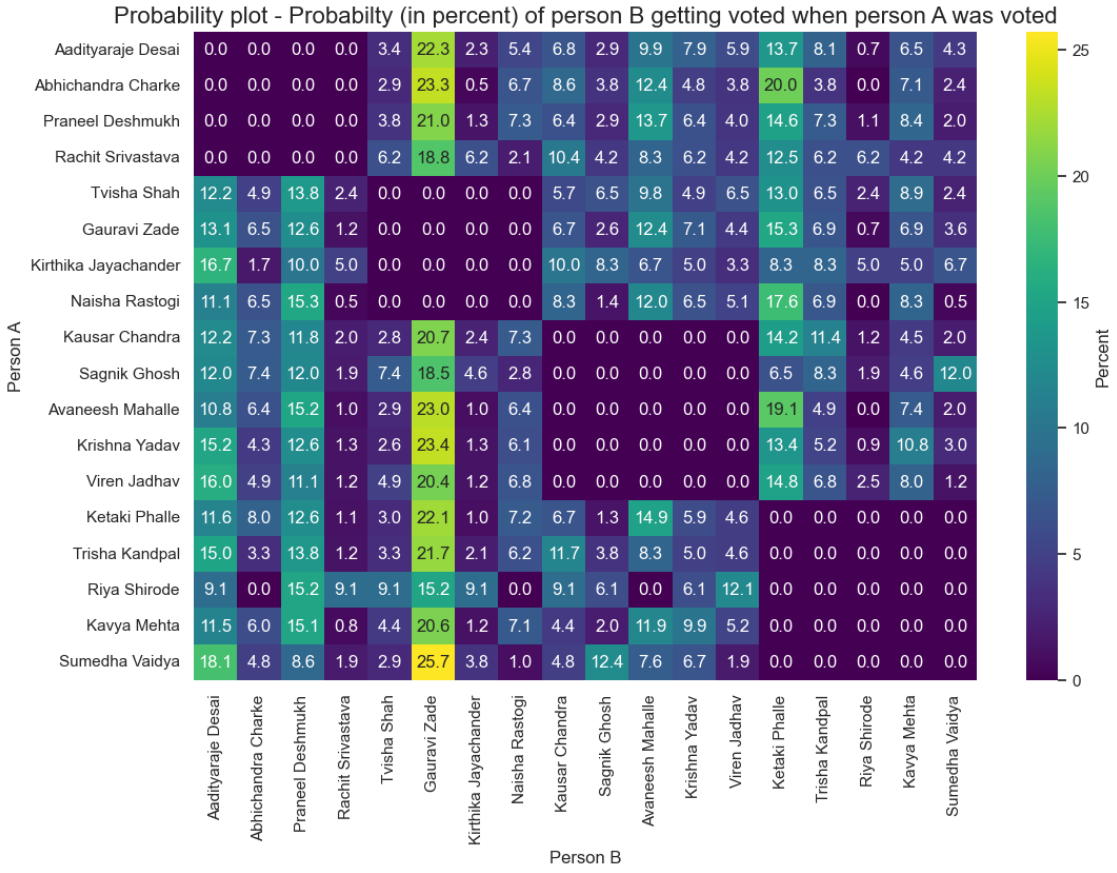
[11]:
```
from itertools import combinations

all_candidates = [
    candidate
    for post_candidates in candidate_data.values()
    for candidate in post_candidates
]

classwise_votes = get_classwise_documents_from_local("votes.json")

# Which candidates were voted toghether in a particular session
vote_sessions = pd.DataFrame(columns=candidate_data.keys())

for _class in classwise_votes:
    for session in _class["votes"]:
        voted = {}
        for vote in session:
            voted[vote["post"]] = vote["name"]
        vote_sessions.loc[len(vote_sessions)] = voted

print(vote_sessions.head())

# co-occurance matrix is the matrix showing how many times candidate B was␣
 ↪voted when candidate A was voted
# co-occurance matrix would be N * N where N are the total number of candidates␣
 ↪across all posts (18 * 18 for this case)
co_occurance_matrix = pd.DataFrame(0, index=all_candidates,␣
 ↪columns=all_candidates)

# updating co-occurance matrix
for (_, session) in vote_sessions.iterrows():
    for (candidate_A, candidate_B) in combinations(session.to_list(), 2):
        co_occurance_matrix.at[candidate_A, candidate_B] += 1
```

```
        co_occurance_matrix.at[candidate_B, candidate_A] += 1

# creating conditional probability matrix
# conditional probabilty matrix is created by normalizing columns of␣
 ↪co-occurance matrix
# normalizing means dividing each row of co-occurance matrix by the total votes␣
 ↪in that row
# the matrix gives the probabilty of person b (x axis) being voted when person␣
 ↪A (y axis) was voted
probability_matrix = co_occurance_matrix.div(co_occurance_matrix.sum(axis=1),␣
 ↪axis=0)

# ---- first plot ----
fig1, ax1 = plt.subplots(figsize=(12, 8))
ax1.set_title(
    "Co-occurance plot - Number of times person A got voted when person B was␣
 ↪voted",
    size=16,
)
sns.heatmap(
    co_occurance_matrix,
    cmap="viridis",
    vmin=0,
    annot=True,
    ax=ax1,
    fmt=".0f",
    cbar_kws={"label": "Count"},
)
ax1.set_ylabel("Person A", size=12)
ax1.set_xlabel("Person B", size=12)
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90)
ax1.set_yticklabels(ax1.get_yticklabels(), rotation=0)
plt.show()

# ---- second plot ----
fig2, ax2 = plt.subplots(figsize=(12, 8))
ax2.set_title(
    "Probability plot - Probabilty (in percent) of person B getting voted when␣
 ↪person A was voted",
    size=16,
)
sns.heatmap(
    probability_matrix * 100,
    cmap="viridis",
    vmin=0,
    annot=True,
```

```
    ax=ax2,
    fmt=".1f",
    cbar_kws={"label": "Percent"},
)
ax2.set_ylabel("Person A", size=12)
ax2.set_xlabel("Person B", size=12)
ax2.set_xticklabels(ax2.get_xticklabels(), rotation=90)
ax2.set_yticklabels(ax2.get_yticklabels(), rotation=0)
plt.show()

# the percents here for a column dont add up to 100 cuz they are mutually␣
 ↪exclusive events
```

	Captain Boy	Captain Girl	Vice Captain Boy	Vice Captain Girl
0	Aadityaraje Desai	Tvisha Shah	Kausar Chandra	Ketaki Phalle
1	Abhichandra Charke	Tvisha Shah	Sagnik Ghosh	Ketaki Phalle
2	Aadityaraje Desai	Tvisha Shah	Avaneesh Mahalle	Trisha Kandpal
3	Aadityaraje Desai	Gauravi Zade	Kausar Chandra	Riya Shirode
4	Abhichandra Charke	Gauravi Zade	Krishna Yadav	Ketaki Phalle





1.13 Strongest && Weakest Allies

Which candidates appeared most often (or least often) alongside another candidate during the elections.

For each candidate, the candidate with most and least co-occurance probability is the strongest and weakest ally respectively.

[12]:
```
# Use row indexes for all comparisions

strongest_ally_series = pd.Series(name="Strongest Ally")
weakest_ally_series = pd.Series(name="Weakest Ally")

for post_name, same_post_candidates in candidate_data.items():
    for name in same_post_candidates:
        # extracting the row which gives co-occurance probabilty for a candidate
        cps = probability_matrix.loc[name]

        # removing values of all the candidates in the same post
```

```
        candidate_probability_series = cps[~cps.index.
 ↪isin(same_post_candidates)]

        strongest_ally_series[name] = candidate_probability_series.idxmax()
        weakest_ally_series[name] = candidate_probability_series.idxmin()

# concat based on similar rows
summary = pd.concat([strongest_ally_series, weakest_ally_series], axis=1)
print(summary.sort_values(by=list(summary.columns)))

# strongest ally is the candidate who is most likely to be voted when a␣
 ↪candidate is voted
# weakest ally is the candidate who is least likely to be voted when a␣
 ↪candidate is voted
```

```
                         Strongest Ally          Weakest Ally
Kirthika Jayachander  Aadityaraje Desai    Abhichandra Charke
Ketaki Phalle              Gauravi Zade  Kirthika Jayachander
Rachit Srivastava          Gauravi Zade        Naisha Rastogi
Sumedha Vaidya             Gauravi Zade        Naisha Rastogi
Sagnik Ghosh               Gauravi Zade     Rachit Srivastava
Viren Jadhav               Gauravi Zade     Rachit Srivastava
Trisha Kandpal             Gauravi Zade     Rachit Srivastava
Kavya Mehta                Gauravi Zade     Rachit Srivastava
Aadityaraje Desai          Gauravi Zade          Riya Shirode
Abhichandra Charke         Gauravi Zade          Riya Shirode
Praneel Deshmukh           Gauravi Zade          Riya Shirode
Kausar Chandra             Gauravi Zade          Riya Shirode
Avaneesh Mahalle           Gauravi Zade          Riya Shirode
Krishna Yadav              Gauravi Zade          Riya Shirode
Gauravi Zade              Ketaki Phalle          Riya Shirode
Naisha Rastogi            Ketaki Phalle          Riya Shirode
Riya Shirode           Praneel Deshmukh    Abhichandra Charke
Tvisha Shah            Praneel Deshmukh     Rachit Srivastava
```

[13]:
```
# mean co-support - mean of all conditional probabilties across all candidates
probability_matrix.mean() * 100
```

[13]:
```
Aadityaraje Desai       10.255480
Abhichandra Charke       3.995533
Praneel Deshmukh         9.970875
Rachit Srivastava        1.704038
Tvisha Shah              3.314564
Gauravi Zade            16.480611
Kirthika Jayachander     2.111771
Naisha Rastogi           4.018980
Kausar Chandra           5.526015
```

```
Sagnik Ghosh             3.230816
Avaneesh Mahalle         7.106593
Krishna Yadav            4.575089
Viren Jadhav             3.635561
Ketaki Phalle           10.174875
Trisha Kandpal           5.039716
Riya Shirode             1.252007
Kavya Mehta              5.038090
Sumedha Vaidya           2.569386
dtype: float64
```

1.14 Popular choice groups

A "Choice group" refers to the candidates voted in a single session.

[14]:
```
choice_groups = vote_sessions.value_counts()
```

1.14.1 Most popular choice groups

[15]:
```
print(f"Most popular choice groups\n\n{choice_groups.head(5)}")
```

```
Most popular choice groups

Captain Boy         Captain Girl  Vice Captain Boy  Vice Captain Girl
Praneel Deshmukh    Gauravi Zade  Avaneesh Mahalle  Ketaki Phalle        24
Aadityaraje Desai   Gauravi Zade  Avaneesh Mahalle  Ketaki Phalle        18
Abhichandra Charke  Gauravi Zade  Avaneesh Mahalle  Ketaki Phalle        14
Aadityaraje Desai   Gauravi Zade  Krishna Yadav     Ketaki Phalle        10
                                  Kausar Chandra    Ketaki Phalle         9
Name: count, dtype: int64
```

1.15 Least popular choice groups

[16]:
```
print(f"Least popular choice groups \n\n{choice_groups.tail(5)}")
```

```
Least popular choice groups

Captain Boy        Captain Girl          Vice Captain Boy  Vice Captain Girl
Rachit Srivastava  Kirthika Jayachander  Krishna Yadav     Kavya Mehta
1
                   Naisha Rastogi        Kausar Chandra    Ketaki Phalle
1
                   Tvisha Shah           Avaneesh Mahalle  Ketaki Phalle
1
                                         Kausar Chandra    Riya Shirode
1
                                         Viren Jadhav      Ketaki Phalle
```

```
1
Name: count, dtype: int64
```

1.16 Average occurance of a choice group

Average count where all 4 voted candidate were same.

[17]:
```
print(f"Average occurance of a choice group {choice_groups.mean():.2f}")
```

```
Average occurance of a choice group 2.81
```

[20]:
```
# Exporting
# !uv run jupyter nbconvert --to html project.ipynb --output index.html
!uv run jupyter nbconvert --to pdf project.ipynb --output index.pdf
```

```
[NbConvertApp] Converting notebook project.ipynb to pdf
[NbConvertApp] Support files will be in index_files/
[NbConvertApp] Making directory ./index_files
[NbConvertApp] Writing 109285 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 1199586 bytes to index.pdf
```