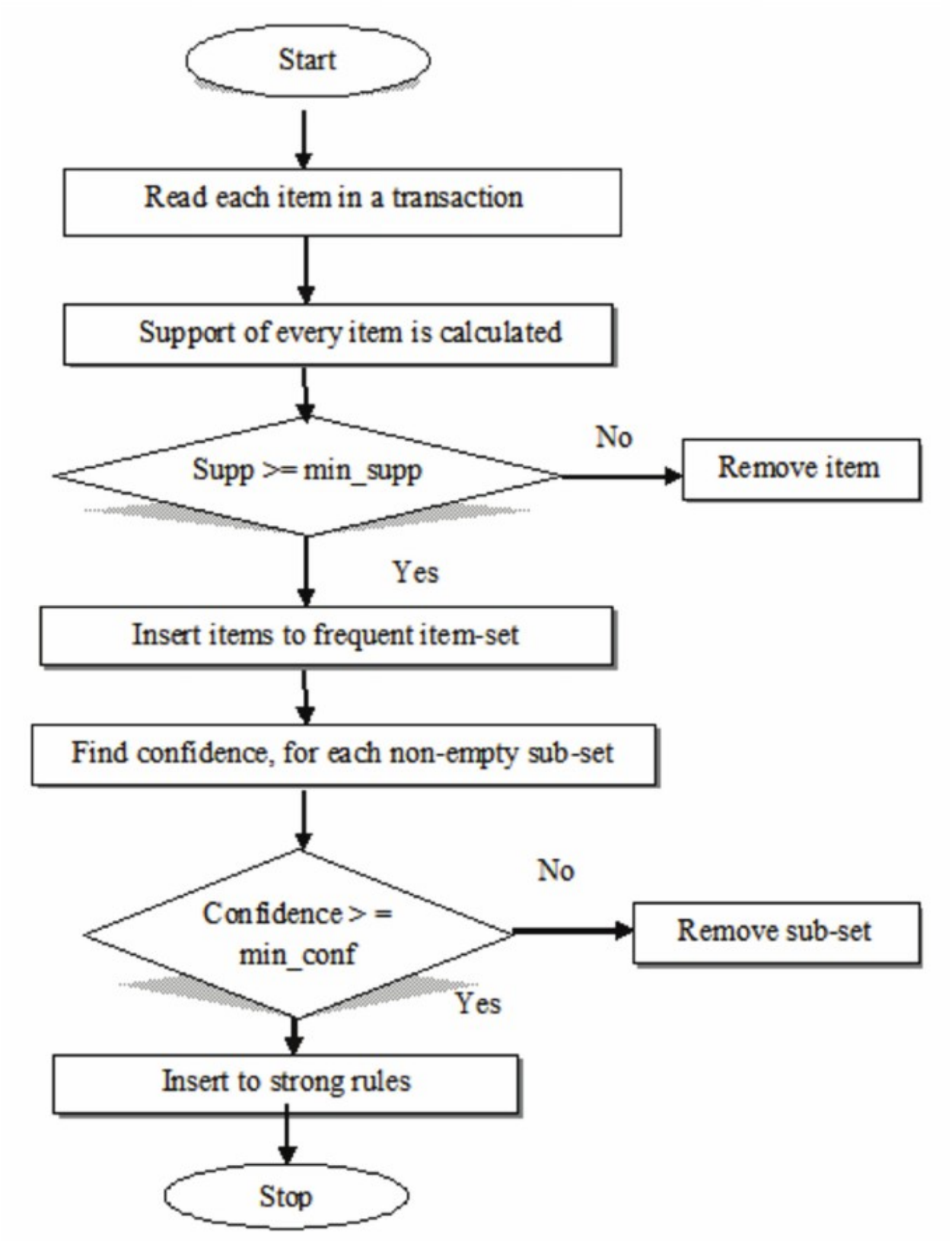
Apriori Algorithm

Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another. In other words, we can say that the apriori algorithm is an association rule leaning that analyzes that people who bought product A also bought product B.

The primary objective of the apriori algorithm is to create the association rule between different objects. The association rule describes how two or more objects are related to one another. Apriori algorithm is also called frequent pattern mining. Generally, you operate the Apriori algorithm on a database that consists of a huge number of transactions. Let's understand the apriori algorithm with the help of an example; suppose you go to Big Bazar and buy different products. It helps the customers buy their products with ease and increases the sales performance of the Big Bazar. In this tutorial, we will discuss the apriori algorithm with examples.


Start
Read each item in a transaction
Support of every item is calculated
Supp >= min_supp
No
Remove item
Yes
Insert items to frequent item-set
Find confidence, for each non-empty sub-set
Confidence > = min_conf
No
Remove sub-set
Yes
Insert to strong rules
Stop

Components of Apriori algorithm

The given three components comprise the apriori algorithm.

- Support
- Confidence
- Lift

Mlxtend

Mlxtend stands for Machine Learning Extensions. It is a third-party Python library which contains many utilities and tools for machine learning and Data Science tasks, including feature selection, ensemble methods, visualization, and model evaluation.

Overview on Mlxtend and Apriori

Apriori is a popular algorithm [1] for extracting frequent itemsets with applications in association rule learning. The apriori algorithm has been designed to operate on databases containing transactions, such as purchases by customers of a store. An itemset is considered as "frequent" if it meets a user-specified support threshold. For instance, if the support threshold is set to 0.5 (50%), a frequent itemset is defined as a set of items that occur together in at least 50% of all transactions in the database.

Comparison IGBR and Apriori algorithm

Run time

First, we compare our grouping with the grouping proposed in other articles in terms of time. We have used a Apriori. In the table below, the running time of this algorithm on a similar dataset is compared with IRGB.

Dataset volume: 100836

Number of group's members: 16

Algorithm	create gruope(s)	generate answer(s)	total(S)
Apriori	85.49	26.91	112.40
IRGB	19.41	5.48	24.89

As a result, in terms of time, IRGB algorithm has a better performance.

Accuracy

Now we compare two algorithms in terms of accuracy.

Algorithm	Accuracy	Precision	Recall	Balanced_Accuracy	TP	FP	TN	FN
Apriori	93.481	86.25	0.9025	50.44	69	11	108728	7576
FCM	81.893	85.33	2.6913	51.29	64	11	10451	2314

The superiority of the Apriori is clearly evident.

Implementation

In the following, we execute the code process and evaluate it according to the different and variable elements

import libs and classes

```
import matplotlib.pyplot as plt
from apriori import GROUP
from engine import Engine
```

Create objects

By calling the dataset, we group it to the desired number and call the main engine of the code.

```
G1 = GROUP('dataset/raiting.csv', 10)
Group = G1.set_matrix()
E1 = Engine(Group)

Group constructor is called to export 10 elements from
dataset/raiting.csv
Engine constructure is called!
```

matrix of group:

```
Group

             1     2     3     4     5     6     7     8     9     10
...  \
Unnamed: 0
...
3            0.0   0.0   0.0   0.0   0.0   0.5   0.0   0.0   0.0   3.5
...
79           0.0   0.0   2.5   4.0   0.0   3.0   0.0   0.5   0.0   0.0
...
161          2.0   0.0   3.5   0.0   3.5   0.0   3.0   0.0   0.0   0.0
...
199          1.5   2.0   1.0   0.0   0.0   0.0   3.0   2.0   0.0   0.0
...
272          2.0   3.5   0.0   0.0   0.0   0.0   1.0   3.5   0.0   0.0
...
```

```
670          3.5   3.5   3.5   4.0   4.0   3.5   4.0   3.5   3.5   3.0
...
702          0.0   0.0   3.5   0.0   3.5   0.0   3.0   4.0   3.0   0.0
...
969          0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
...
1039         0.0   3.5   0.0   0.0   3.5   0.0   0.0   3.0   3.0   0.0
...
1187         2.5   3.0   4.0   0.0   3.0   0.0   3.5   0.0   0.0   0.0
...

             2062  2063  2064  2065  2066  2067  2068  2069  2070  2071

Unnamed: 0

3               0     0     0     0     0     0     0     0     0     0

79              0     0     0     0     0     0     0     0     0     0

161             0     0     0     0     0     0     0     0     0     0

199             0     0     0     0     0     0     0     0     0     0

272             0     0     0     0     0     0     0     0     0     0

670             0     0     0     0     0     0     0     0     0     0

702             0     0     0     0     0     0     0     0     0     0

969             0     0     0     0     0     0     0     0     0     0

1039            0     0     0     0     0     0     0     0     0     0

1187            0     0     0     0     0     0     0     0     0     0


[10 rows x 2071 columns]
```

To run the code, we can use the E1.run function. The function input specifies whether or not to use the centrality criterion. We will explain it in the next sections.

```
E1.run(True)
```

```
Evaluation Results: {'Accuracy': 97.56272401433692, 'Precision': 16.0,
'Recall': 3.361344537815126, 'Balanced_Accuracy': 51.488399791339354,
'Confusion_counters': {'TP': 8, 'FP': 42, 'TN': 10880, 'FN': 230}}
```

We can get the accuracy from the function as follows.

```
run_values = E1.run_val(True)
run_values['Accuracy']
```

```
97.56272401433692
```

Comparison diferent datasets

```
G1 = GROUP('dataset/raiting.csv', 10)
G2 = GROUP('dataset/movielenz.csv', 10)
```

```
Group constructor is called to export 10 elements from
dataset/raiting.csv
Group constructor is called to export 10 elements from
dataset/movielenz.csv
```

```
Group1 = G1.set_matrix()
Group2 = G2.set_matrix()
E1 = Engine(Group1)
E2 = Engine(Group2)
```

```
Engine constructure is called!
Engine constructure is called!
```

```
E1.run(True)
```

```
Evaluation Results: {'Accuracy': 97.56272401433692, 'Precision': 16.0,
'Recall': 3.361344537815126, 'Balanced_Accuracy': 51.488399791339354,
'Confusion_counters': {'TP': 8, 'FP': 42, 'TN': 10880, 'FN': 230}}
```

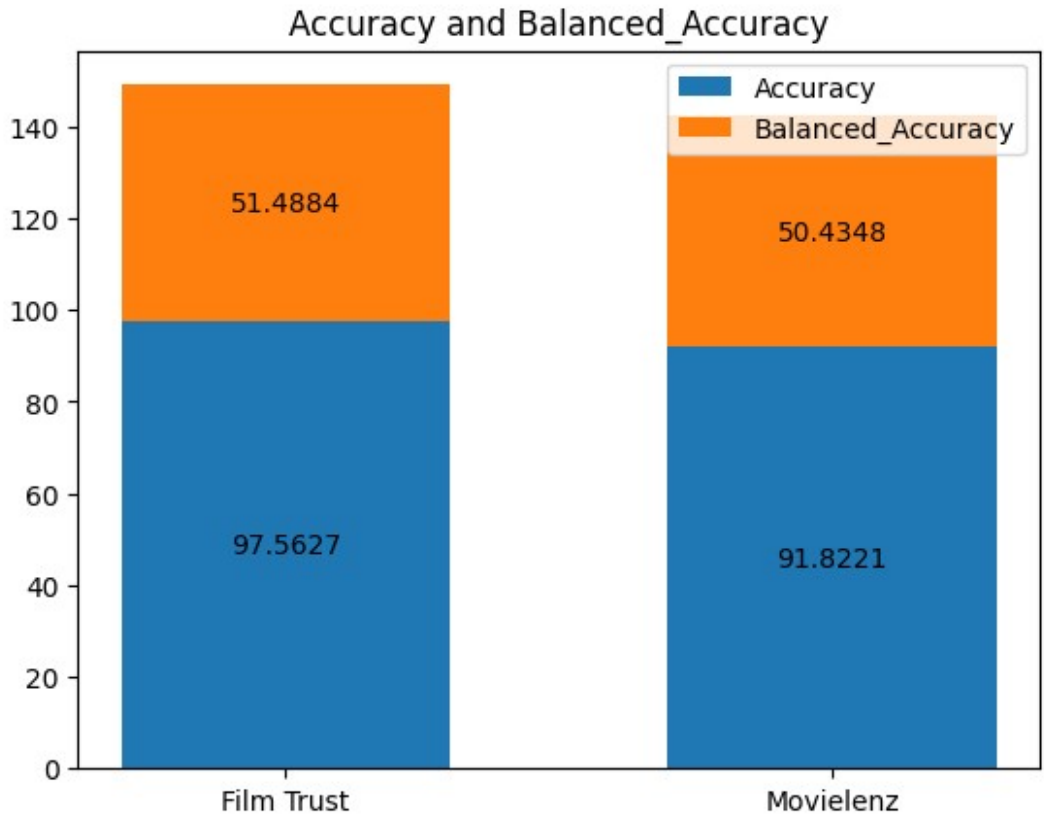
```
E2.run(True)
```

```
Evaluation Results: {'Accuracy': 91.82213557288541, 'Precision': 96.0,
'Recall': 0.872885979268958, 'Balanced_Accuracy': 50.43480849526531,
'Confusion_counters': {'TP': 48, 'FP': 2, 'TN': 61179, 'FN': 5451}}
```

```
run_values1 = E1.run_val(True)
run_values2 = E2.run_val(True)
```

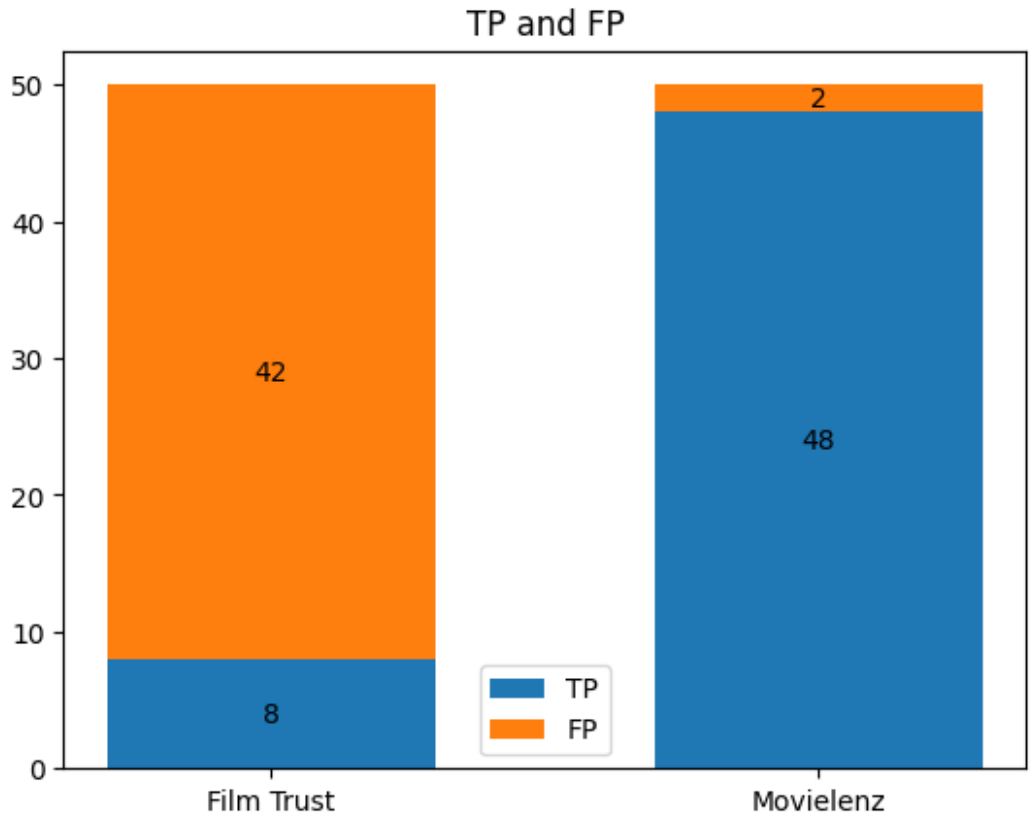
```
import numpy as np
species = ('Film Trust', 'Movielenz')
acc_count = {
    'Accuracy': np.array([run_values1['Accuracy'],
run_values2['Accuracy']]),
    'Balanced_Accuracy': np.array([run_values1['Balanced_Accuracy'],
run_values2['Balanced_Accuracy']]),
}
width = 0.6


fig, ax = plt.subplots()
bottom = np.zeros(2)
```

```
for a, a_count in acc_count.items():
    p = ax.bar(species, a_count, width, label=a, bottom=bottom)
    bottom += a_count

    ax.bar_label(p, label_type='center')

ax.set_title('Accuracy and Balanced_Accuracy')
ax.legend()

plt.show()
```



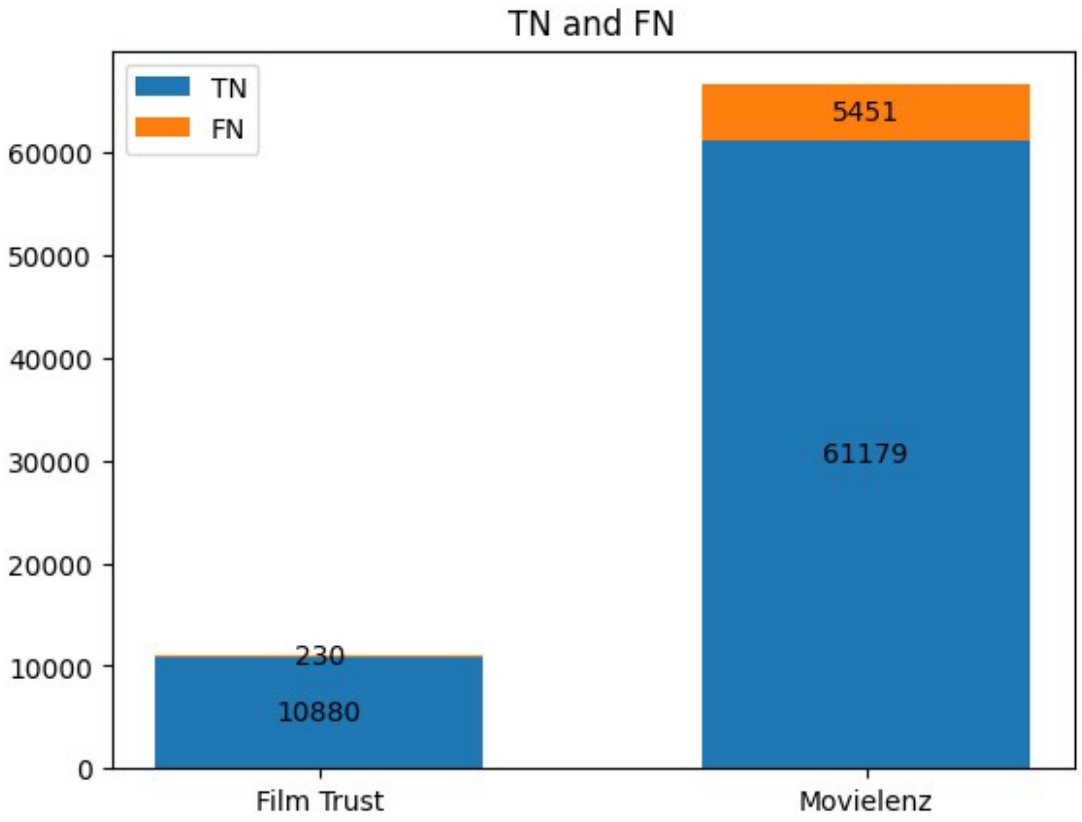

```
species = ('Film Trust', 'Movielenz')
acc_count = {
    'TP': np.array([run_values1['Confusion_counters']['TP'],
run_values2['Confusion_counters']['TP']]),
    'FP': np.array([run_values1['Confusion_counters']['FP'],
run_values2['Confusion_counters']['FP']]),
}
width = 0.6


fig, ax = plt.subplots()
bottom = np.zeros(2)
```

```
for a, a_count in acc_count.items():
    p = ax.bar(species, a_count, width, label=a, bottom=bottom)
    bottom += a_count

    ax.bar_label(p, label_type='center')

ax.set_title('TP and FP')
ax.legend()

plt.show()
```




```
species = ('Film Trust', 'Movielenz')
acc_count = {
    'TN': np.array([run_values1['Confusion_counters']['TN'],
run_values2['Confusion_counters']['TN']]),
    'FN': np.array([run_values1['Confusion_counters']['FN'],
run_values2['Confusion_counters']['FN']]),
}
width = 0.6


fig, ax = plt.subplots()
bottom = np.zeros(2)
```

```
for a, a_count in acc_count.items():
    p = ax.bar(species, a_count, width, label=a, bottom=bottom)
    bottom += a_count

    ax.bar_label(p, label_type='center')

ax.set_title('TN and FN')
ax.legend()

plt.show()
```




```
number_of_group_elements = [2, 4, 6, 8, 10]
result_list = []

for noge in number_of_group_elements:
    G1 = GROUP('dataset/raiting.csv', noge)
    Group = G1.set_matrix()
    E1 = Engine(Group)
    run_values = E1.run_val(True)
    result_list.append(run_values)
```

```
Group constructor is called to export 2 elements from
dataset/raiting.csv
Engine constructure is called!
```

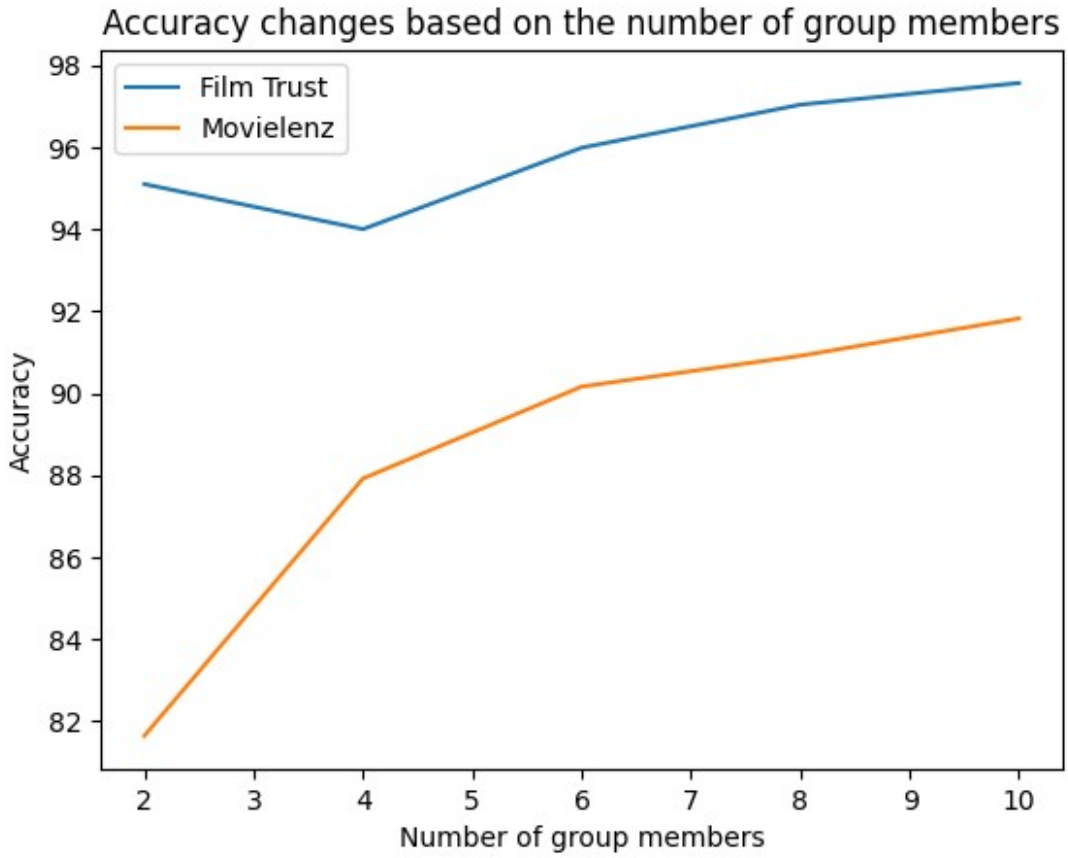
```
Group constructor is called to export 4 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 6 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 8 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 10 elements from
dataset/raiting.csv
Engine constructure is called!
```

```
number_of_group_elements = [2, 4, 6, 8, 10]
result_list2 = []

for noge in number_of_group_elements:
    G2 = GROUP('dataset/movielenz.csv', noge)
    Group2 = G2.set_matrix()
    E2 = Engine(Group2)
    run_values2 = E2.run_val(True)
    result_list2.append(run_values2)
```

```
Group constructor is called to export 2 elements from
dataset/movielenz.csv
Engine constructure is called!
Group constructor is called to export 4 elements from
dataset/movielenz.csv
Engine constructure is called!
Group constructor is called to export 6 elements from
dataset/movielenz.csv
Engine constructure is called!
Group constructor is called to export 8 elements from
dataset/movielenz.csv
Engine constructure is called!
Group constructor is called to export 10 elements from
dataset/movielenz.csv
Engine constructure is called!
```

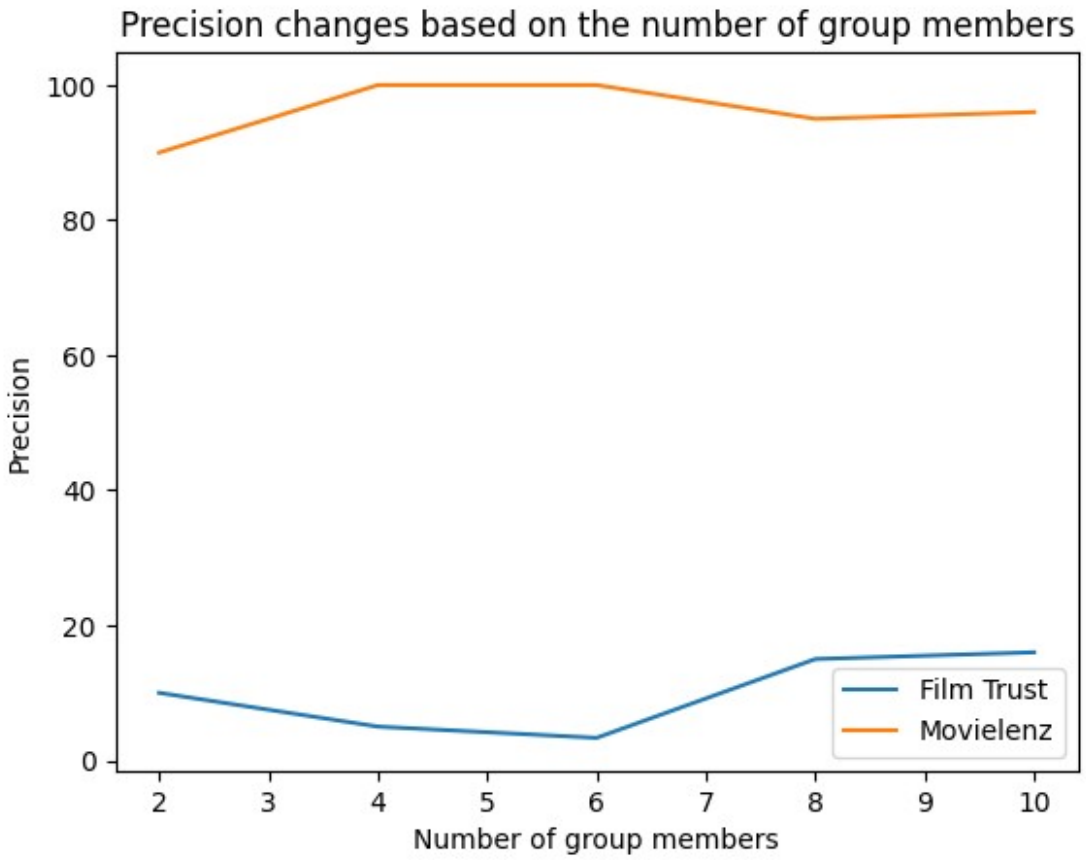
```
Accuracy_list = []
for rl in result_list:
    Accuracy_list.append(rl['Accuracy'])

Accuracy_list2 = []
for rl in result_list2:
    Accuracy_list2.append(rl['Accuracy'])

x = Accuracy_list
x2 = Accuracy_list2
y = number_of_group_elements
```

```
plt.plot(y, x, label = "Film Trust")
plt.plot(y, x2, label = "Movielenz")

plt.xlabel("Number of group members")
plt.ylabel('Accuracy')
plt.title("Accuracy changes based on the number of group members")

plt.legend()
plt.show()
```



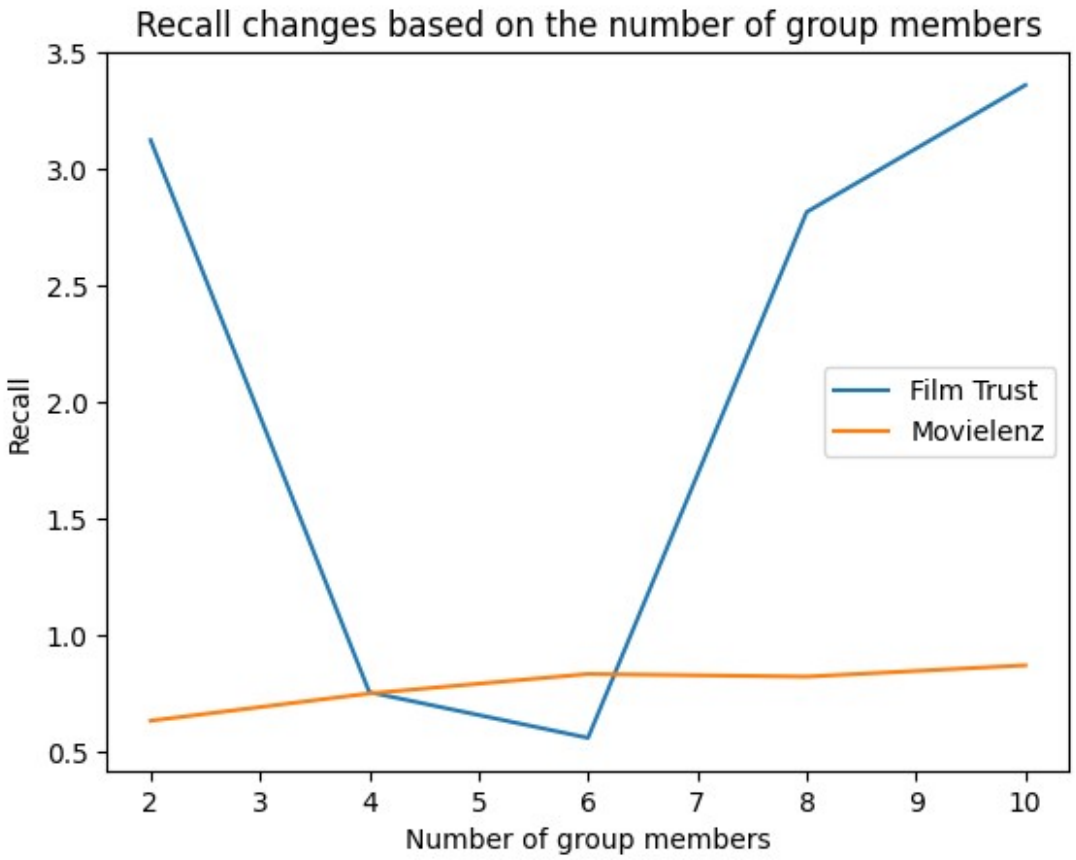

```
Precision_list = []
for rl in result_list:
    Precision_list.append(rl['Precision'])

Precision_list2 = []
for rl in result_list2:
    Precision_list2.append(rl['Precision'])

x = Precision_list
x2 = Precision_list2
y = number_of_group_elements
```

```
plt.plot(y, x, label = "Film Trust")
plt.plot(y, x2, label = "Movielenz")

plt.xlabel("Number of group members")
plt.ylabel('Precision')
plt.title("Precision changes based on the number of group members")

plt.legend()
plt.show()
```




```
Recall_list = []
for rl in result_list:
    Recall_list.append(rl['Recall'])

Recall_list2 = []
for rl in result_list2:
    Recall_list2.append(rl['Recall'])

x = Recall_list
x2 = Recall_list2
y = number_of_group_elements
```

```
plt.plot(y, x, label = "Film Trust")
plt.plot(y, x2, label = "Movielenz")

plt.xlabel("Number of group members")
plt.ylabel('Recall')
plt.title("Recall changes based on the number of group members")

plt.legend()
plt.show()
```



Run with diferent number of group's member

```
number_of_group_elements = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
result_list = []

for noge in number_of_group_elements:
    G1 = GROUP('dataset/raiting.csv', noge)
    Group = G1.set_matrix()
    E1 = Engine(Group)
    run_values = E1.run_val(True)
    result_list.append(run_values)
```

```
Group constructor is called to export 2 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 4 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 6 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 8 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 10 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 12 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 14 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 16 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 18 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 20 elements from
dataset/raiting.csv
Engine constructure is called!
```

The effect of the number of people in the group on accuracy

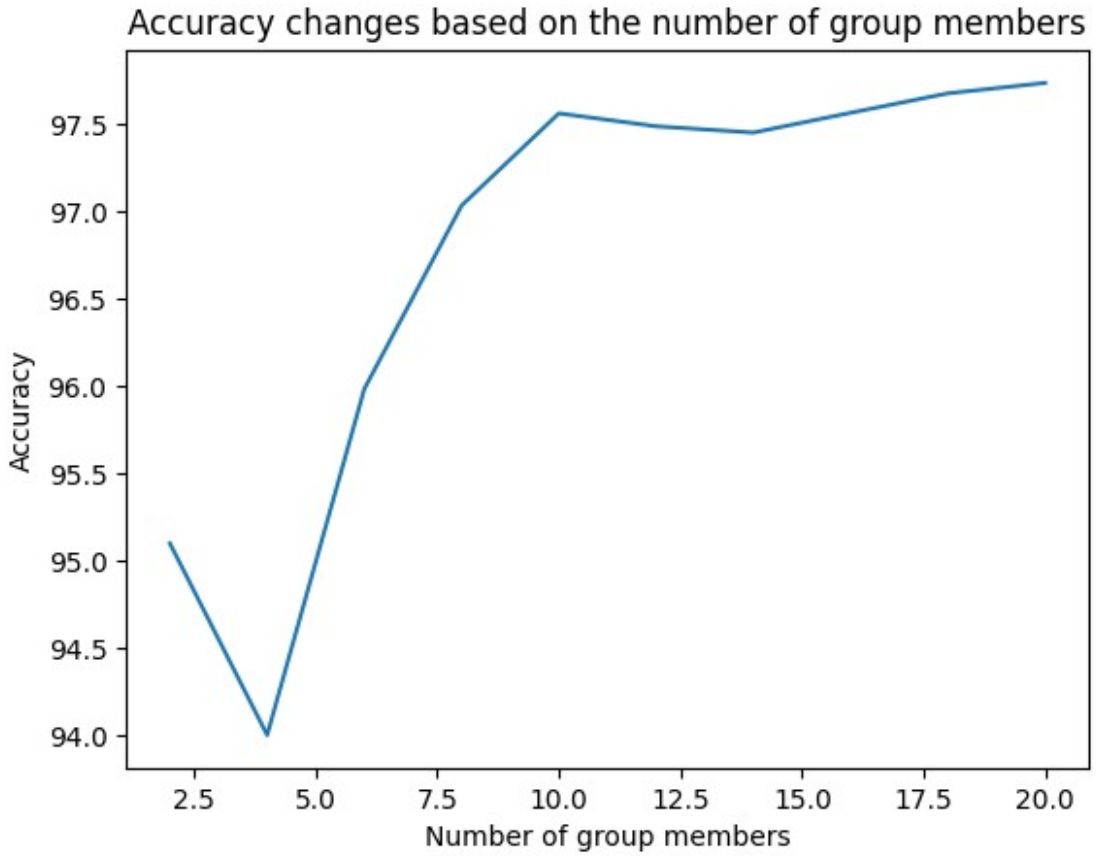
We run a list of the number of different people in one lap and get the necessary outputs.

```
Accuracy_list = []
Precision_list = []
Recall_list = []
Balanced_Accuracy_list = []
for rl in result_list:
    Accuracy_list.append(rl['Accuracy'])
    Precision_list.append(rl['Precision'])
    Recall_list.append(rl['Recall'])
    Balanced_Accuracy_list.append(rl['Balanced_Accuracy'])
```

Accuracy

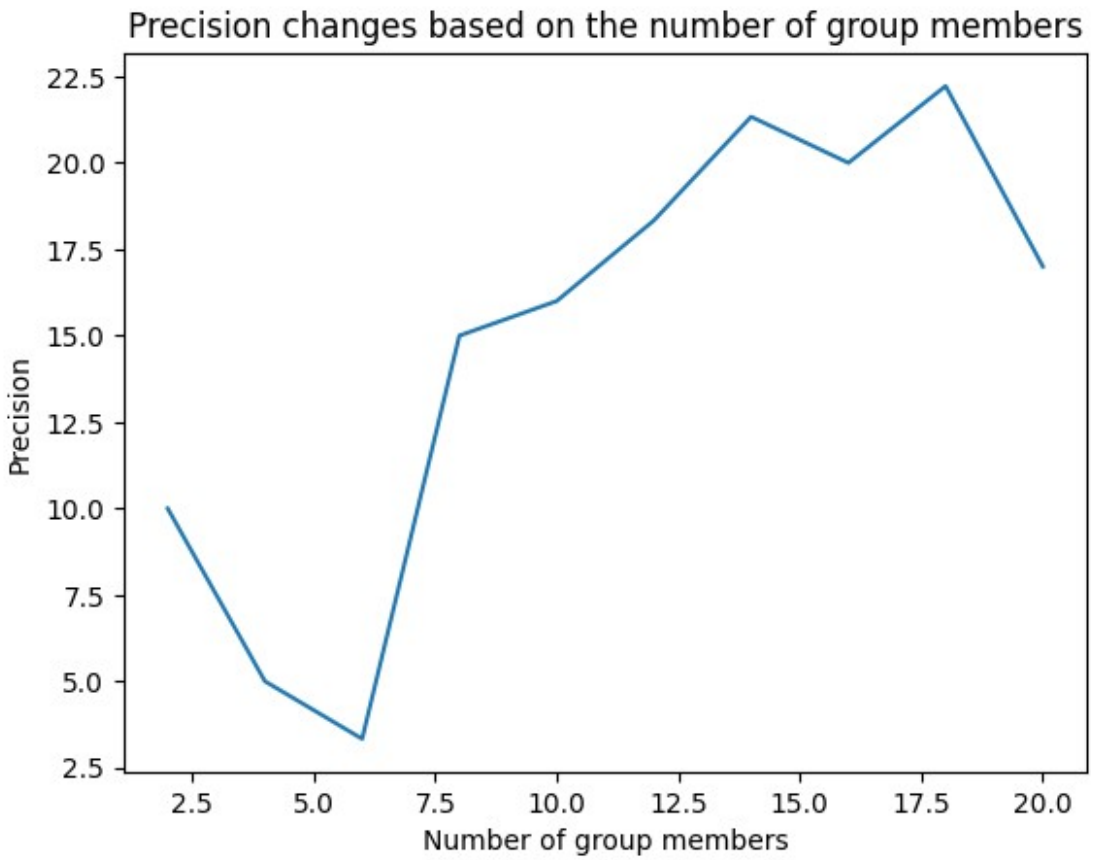
```
fig = plt.figure()
x = number_of_group_elements
```

```
y = Accuracy_list

plt.xlabel("Number of group members")
plt.ylabel('Accuracy')
plt.title("Accuracy changes based on the number of group members")

plt.plot(x, y)
plt.show()
```



Precision

```
fig = plt.figure()
x = number_of_group_elements
y = Precision_list

plt.xlabel("Number of group members")
plt.ylabel('Precision')
plt.title("Precision changes based on the number of group members")

plt.plot(x, y)
plt.show()
```

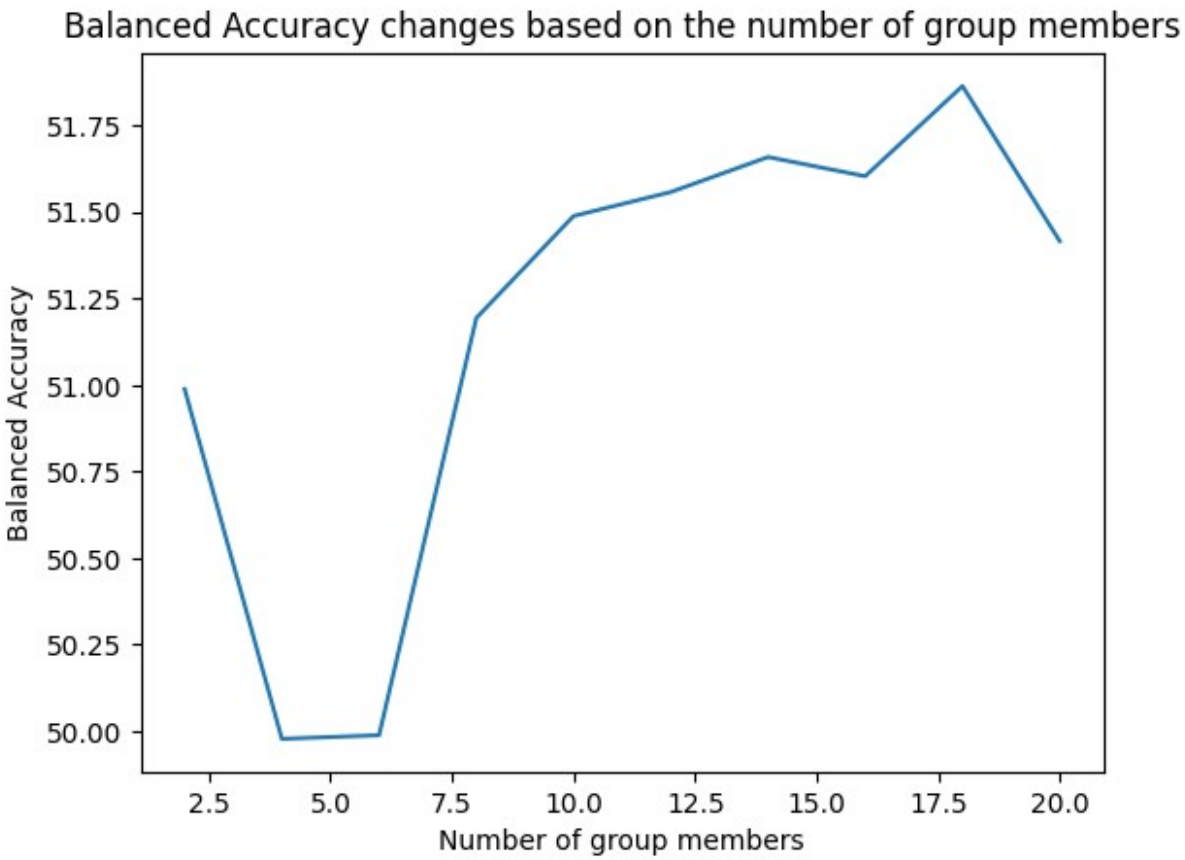


Recall

```
fig = plt.figure()
x = number_of_group_elements
y = Recall_list

plt.xlabel("Number of group members")
plt.ylabel('Recall')
plt.title("Recall changes based on the number of group members")

plt.plot(x, y)
plt.show()
```

Balanced_Accuracy

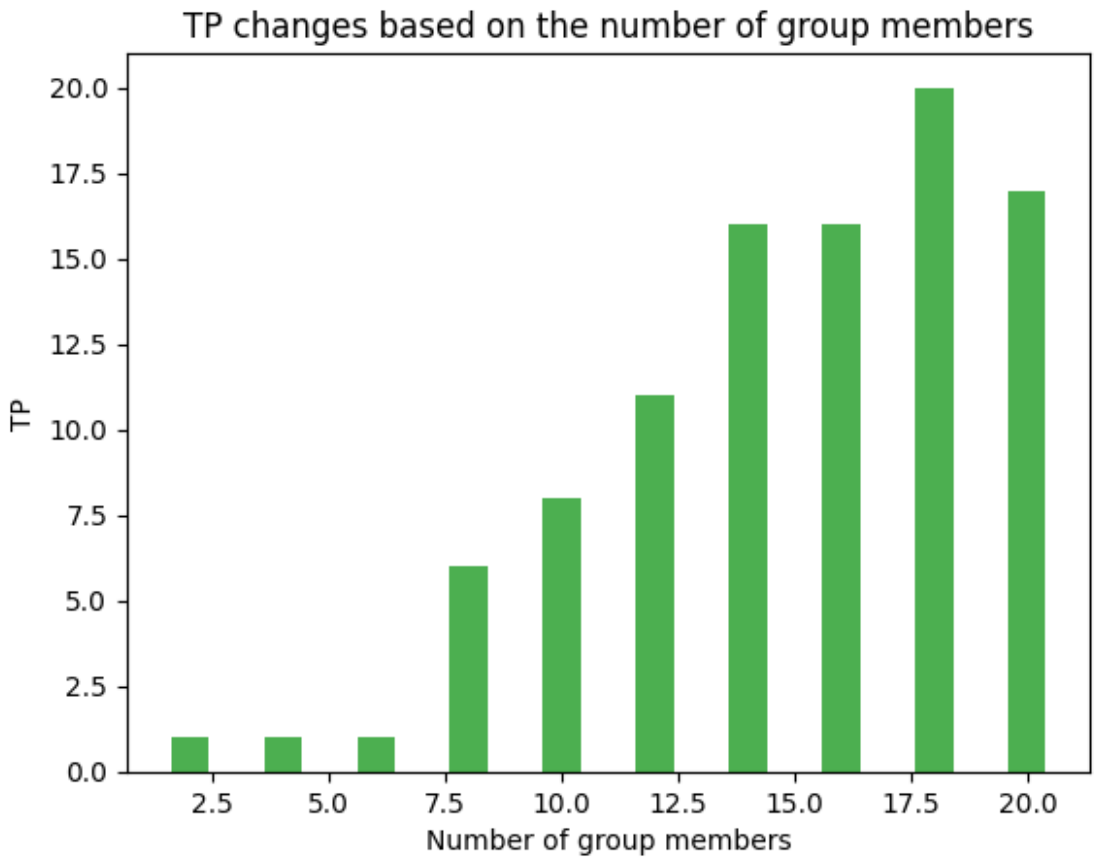
```
fig = plt.figure()
x = number_of_group_elements
y = Balanced_Accuracy_list

plt.xlabel("Number of group members")
plt.ylabel('Balanced Accuracy')
plt.title("Balanced Accuracy changes based on the number of group
members")

plt.plot(x, y)
plt.show()
```




```
TP = []
FP = []
TN = []
FN = []
Balanced_Accuracy_list = []
for rl in result_list:
    TP.append(rl['Confusion_counters']['TP'])
    FP.append(rl['Confusion_counters']['FP'])
    TN.append(rl['Confusion_counters']['TN'])
    FN.append(rl['Confusion_counters']['FN'])
```

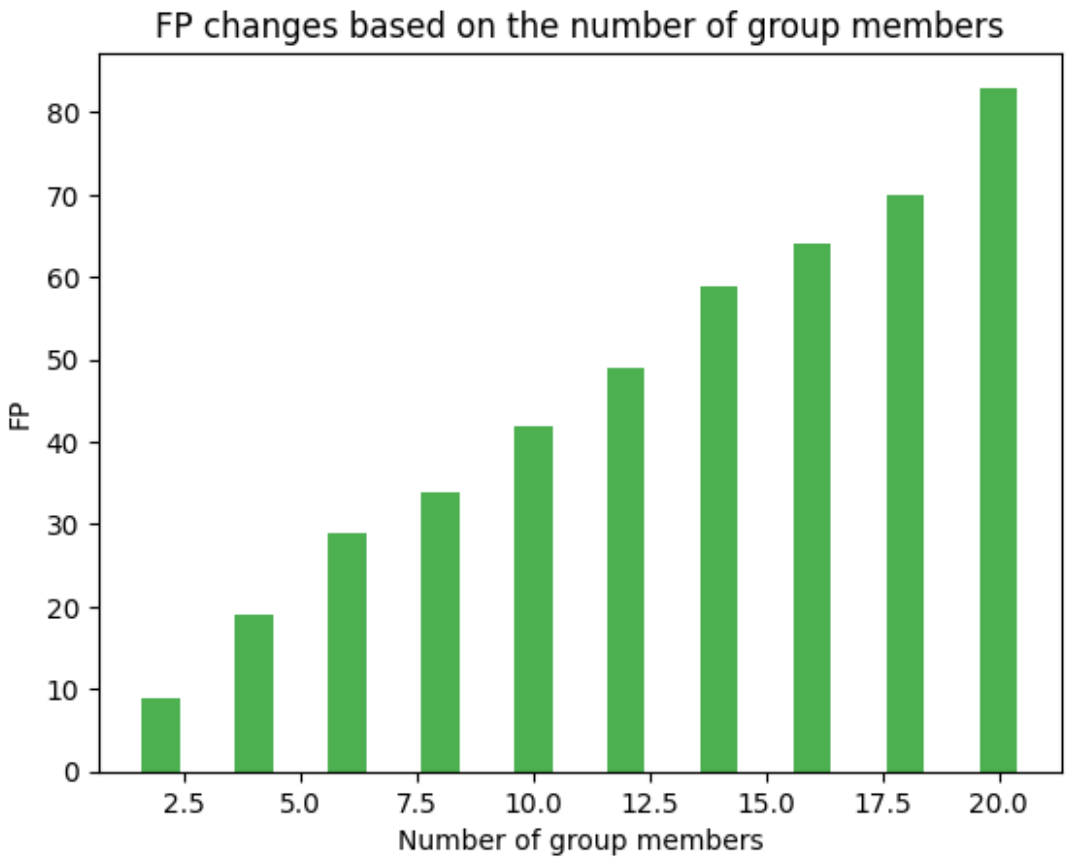
TP

```
x = number_of_group_elements
y = TP

plt.xlabel("Number of group members")
plt.ylabel('TP')
plt.title("TP changes based on the number of group members")

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



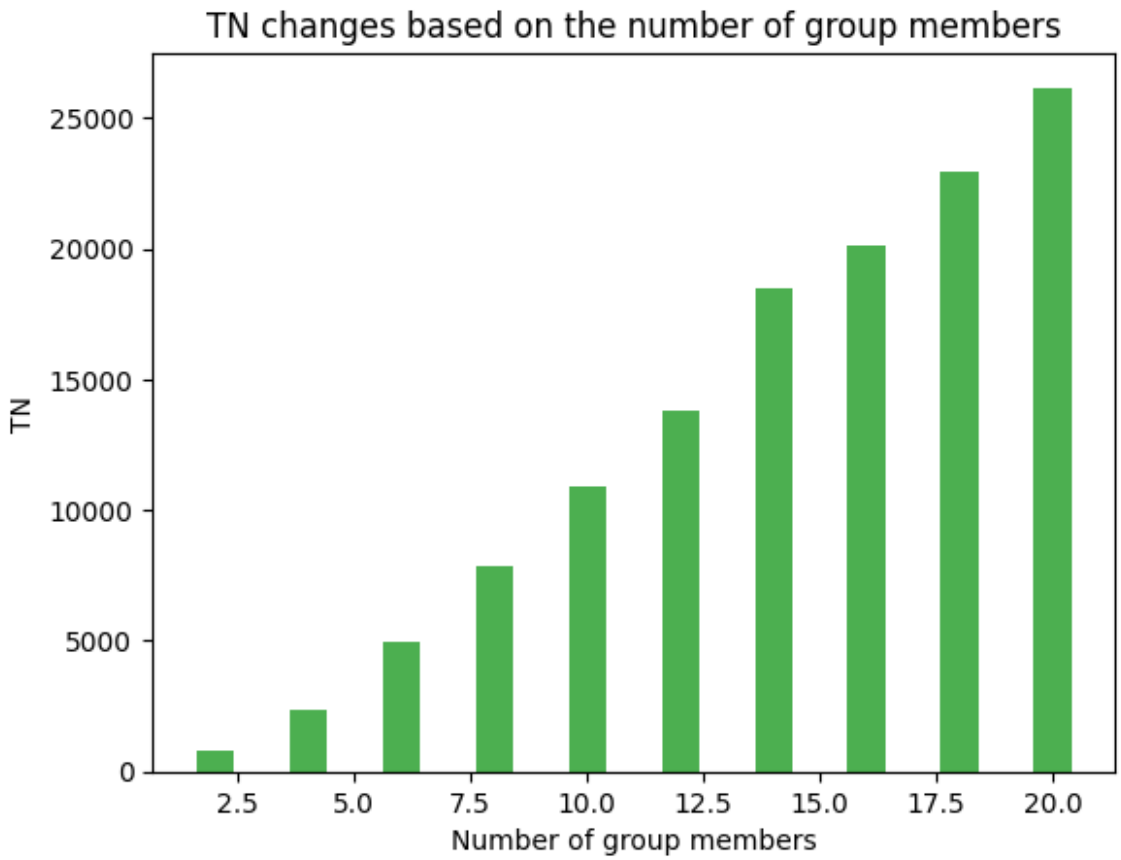
FP

```
x = number_of_group_elements
y = FP

plt.xlabel("Number of group members")
plt.ylabel('FP')
plt.title("FP changes based on the number of group members")

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



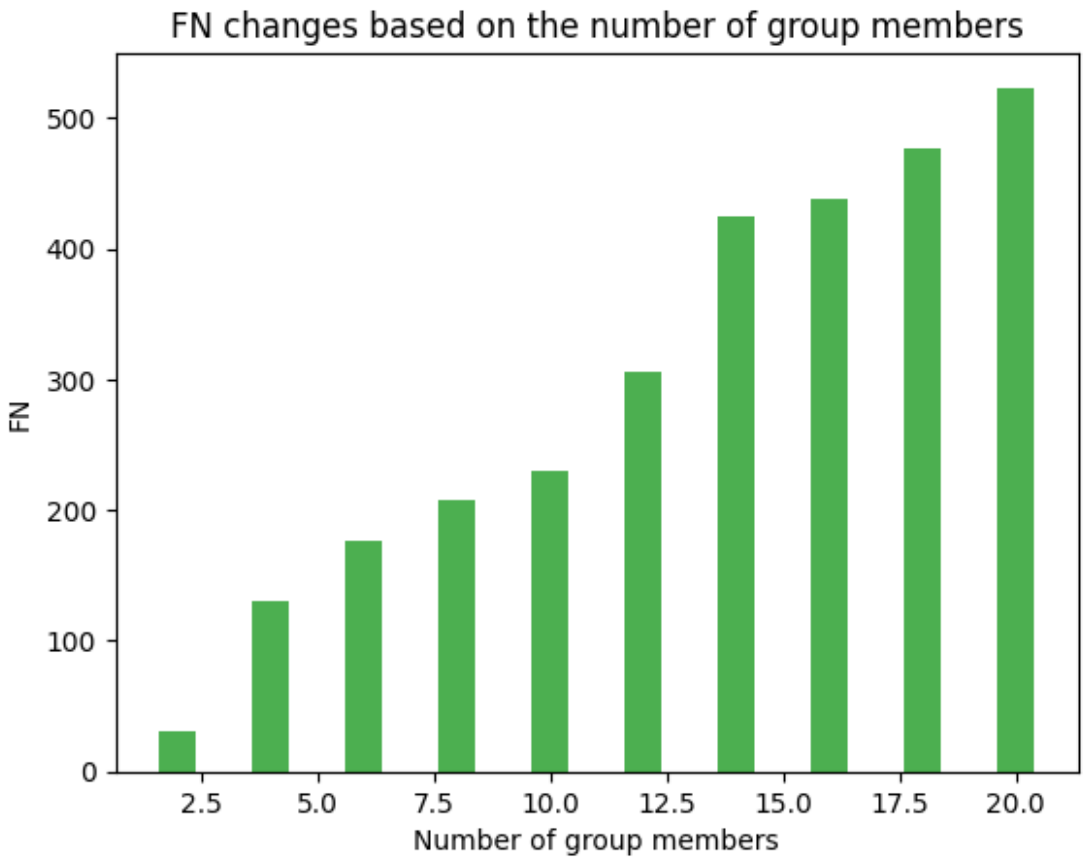
TN

```
x = number_of_group_elements
y = TN

plt.xlabel("Number of group members")
plt.ylabel('TN')
plt.title("TN changes based on the number of group members")

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



FN

```
x = number_of_group_elements
y = FN

plt.xlabel("Number of group members")
plt.ylabel('FN')
plt.title("FN changes based on the number of group members")

plt.bar(x, y, color = "#4CAF50")
plt.show()
```



Centrality

```
def calculate_centerality(self):
    members = self.group.index
    ratings = self.group.to_numpy()  # Convert DataFrame to a NumPy
array
    matrix = np.zeros((len(members), len(members)))
    avg = np.average(ratings)

    loop_counter = 0
    for r in ratings:
        matrix[loop_counter][loop_counter] = abs(avg - np.average(r))
        loop_counter += 1

    # Convert the matrix to a DataFrame with proper index and columns
    Cd = pd.DataFrame(matrix, index=members, columns=members)

    return Cd

result_list = []
result_list2 = []
number_of_group_elements = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for noge in number_of_group_elements:
    G1 = GROUP('dataset/raiting.csv', noge)
```

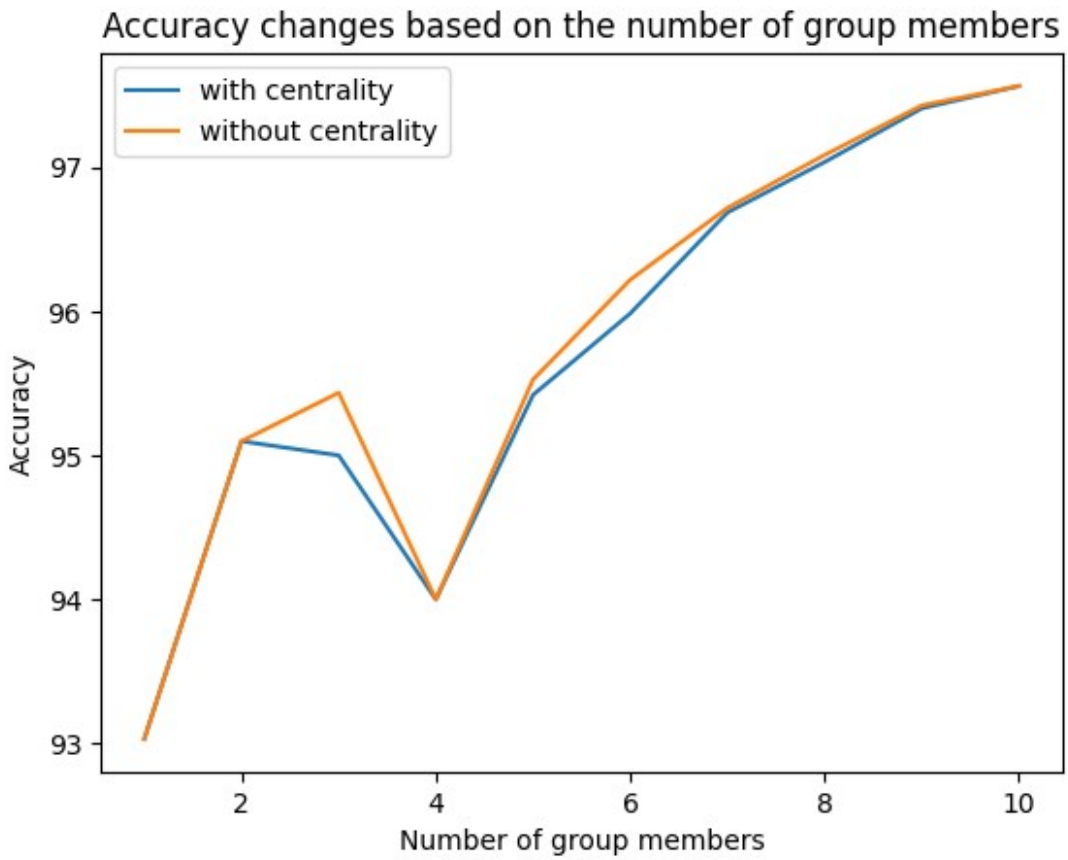
```
    Group = G1.set_matrix()
    E1 = Engine(Group)
    run_values = E1.run_val(True)#with centrality
    result_list.append(run_values)
    run_values2 = E1.run_val(False)#without centrality
    result_list2.append(run_values2)
```

```
Group constructor is called to export 1 elements from
dataset/raiting.csv
Engine constructure is called!

/home/sajjad/jupyter/Apriori/engine.py:93: RuntimeWarning: divide by
zero encountered in double_scalars
  LeaderImpact = ts_sumation[LeaderId] / (total_members - 1)
/home/sajjad/jupyter/Apriori/engine.py:108: RuntimeWarning: divide by
zero encountered in double_scalars
  LeaderImpact = ts_sumation[LeaderId] / (total_members - 1)

Group constructor is called to export 2 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 3 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 4 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 5 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 6 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 7 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 8 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 9 elements from
dataset/raiting.csv
Engine constructure is called!
Group constructor is called to export 10 elements from
dataset/raiting.csv
Engine constructure is called!

Accuracy_list = []
Precision_list = []
Recall_list = []
Balanced_Accuracy_list = []
```

```
for rl in result_list:
    Accuracy_list.append(rl['Accuracy'])
    Precision_list.append(rl['Precision'])
    Recall_list.append(rl['Recall'])
    Balanced_Accuracy_list.append(rl['Balanced_Accuracy'])


Accuracy_list2 = []
Precision_list2 = []
Recall_list2 = []
Balanced_Accuracy_list2 = []
for rl in result_list2:
    Accuracy_list2.append(rl['Accuracy'])
    Precision_list2.append(rl['Precision'])
    Recall_list2.append(rl['Recall'])
    Balanced_Accuracy_list2.append(rl['Balanced_Accuracy'])
```

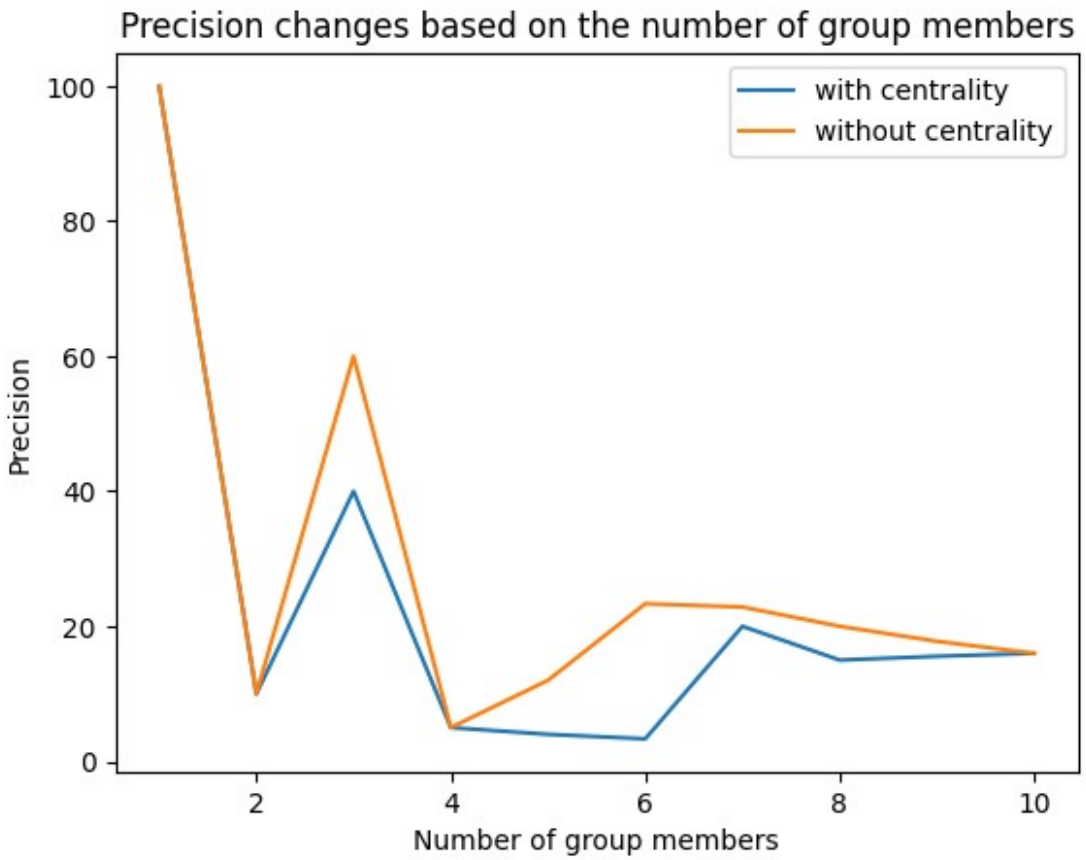
Accuracy

```
x = Accuracy_list
x2 = Accuracy_list2
y = number_of_group_elements

plt.plot(y, x, label = "with centrality")
plt.plot(y, x2, label = "without centrality")

plt.xlabel("Number of group members")
plt.ylabel('Accuracy')
plt.title("Accuracy changes based on the number of group members")

plt.legend()
plt.show()
```



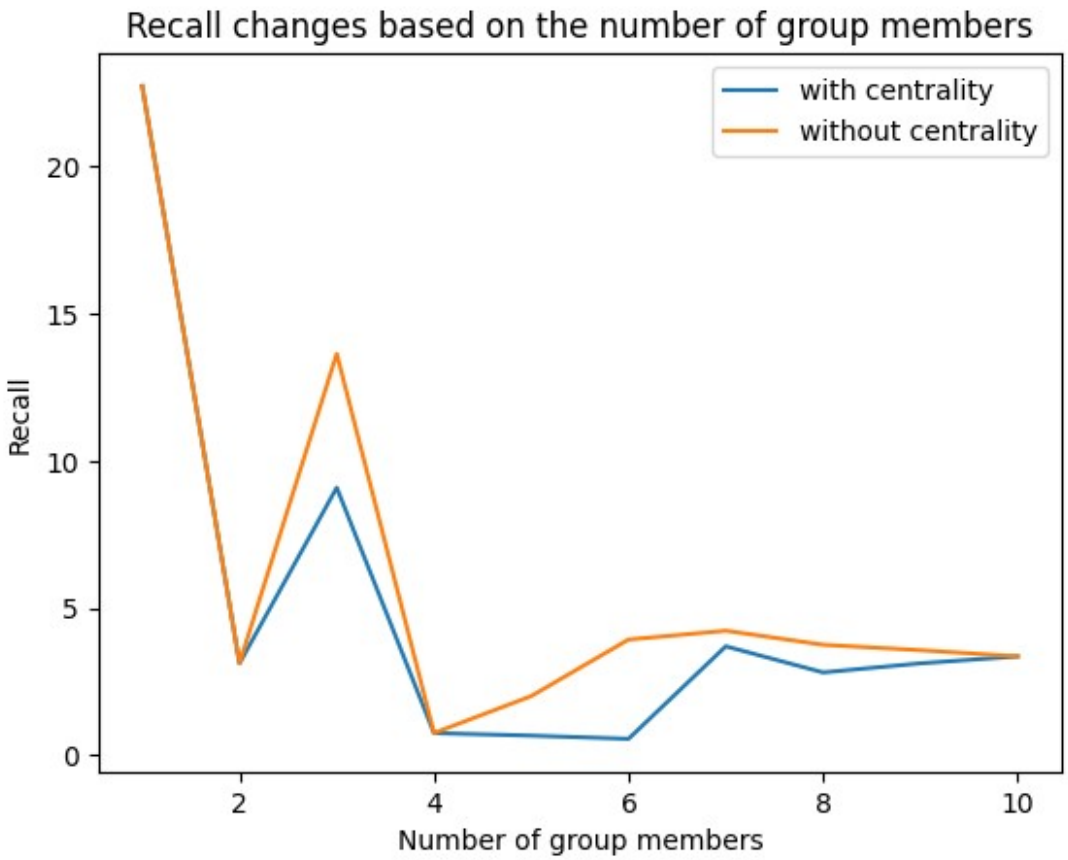
Precision

```
x = Precision_list
x2 = Precision_list2
y = number_of_group_elements

plt.plot(y, x, label = "with centrality")
plt.plot(y, x2, label = "without centrality")

plt.xlabel("Number of group members")
plt.ylabel('Precision')
plt.title("Precision changes based on the number of group members")

plt.legend()
plt.show()
```



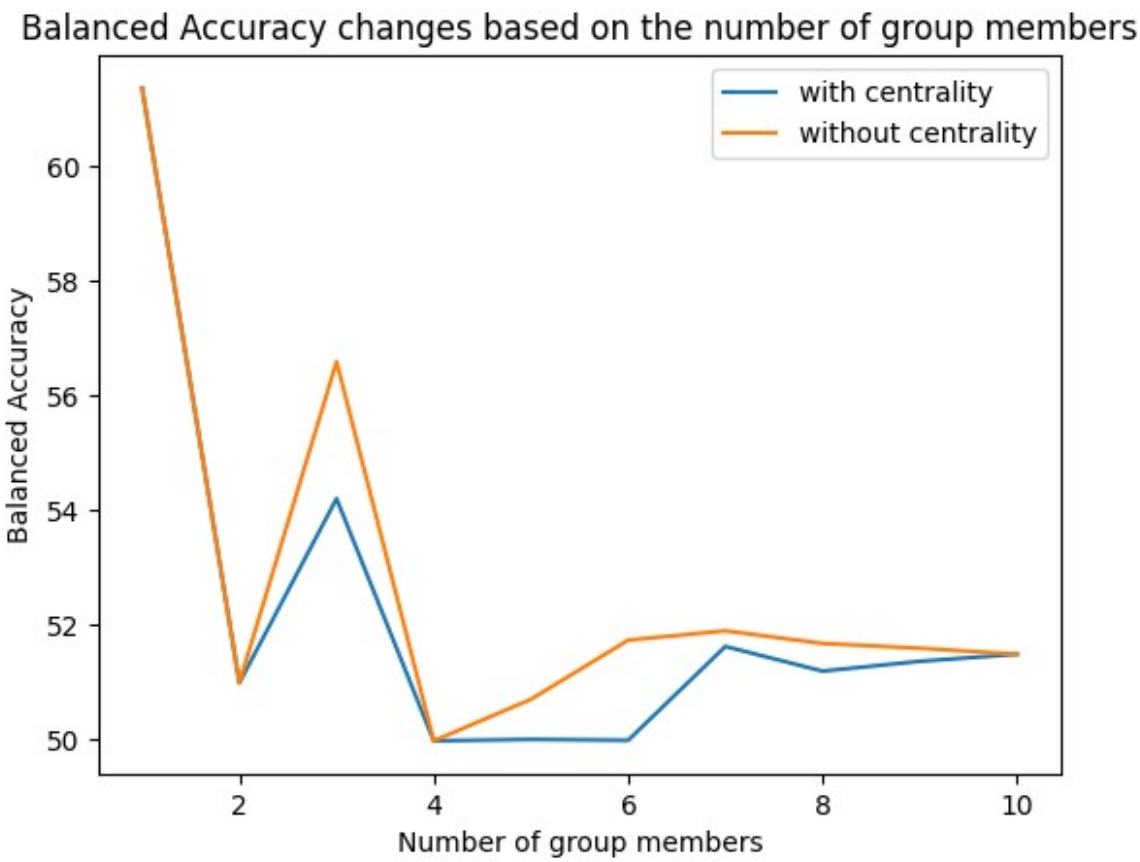
Recall

```
x = Recall_list
x2 = Recall_list2
y = number_of_group_elements

plt.plot(y, x, label = "with centrality")
plt.plot(y, x2, label = "without centrality")

plt.xlabel("Number of group members")
plt.ylabel('Recall')
plt.title("Recall changes based on the number of group members")

plt.legend()
plt.show()
```



Balanced Accuracy

```
x = Balanced_Accuracy_list
x2 = Balanced_Accuracy_list2
y = number_of_group_elements

plt.plot(y, x, label = "with centrality")
plt.plot(y, x2, label = "without centrality")

plt.xlabel("Number of group members")
plt.ylabel('Balanced Accuracy')
plt.title("Balanced Accuracy changes based on the number of group
members")

plt.legend()
plt.show()
```


Balanced Accuracy changes based on the number of group members
with centrality
without centrality
Balanced Accuracy
60
58
56
54
52
50
2
4
6
8
10
Number of group members