```
from google.colab import userdata
import json
import time
from datetime import datetime, timedelta, timezone
from pathlib import Path
import requests
import re
import pandas as pd

```

Download data

↳ 4 cells hidden

Which cards have the highest winrate?

```
df2["totalCards"] = (
    df2[[c for c in df2.columns if "powerCardsOwned" in c]]
    .fillna("")
    .astype(str)
    .agg(",".join, axis=1)   # ← THIS IS THE FIX
)

df2['totalCards'][0]
cards = (
    df2["totalCards"]
    .fillna("")
    .astype(str)
    .str.replace(r"\.\.\.", "", regex=True)      # drop ...
    .str.replace(r"[\[\]']", "", regex=True)     # drop [ ] '
    .str.replace(r"\s+", "", regex=True)         # remove ALL whitespace
    .str.split(",")
    .apply(lambda xs: [x for x in xs if x])      # drop empty strings
    .apply(lambda xs: [x.replace("_", "") for x in xs])

)

df2["cards_list"] = cards
df2["size"] = df2["cards_list"].str.len()
from sklearn.preprocessing import MultiLabelBinarizer

# cards_list must be list-of-strings per row
# example: ["CardA", "CardB"]
cards = df2["cards_list"].apply(lambda xs: xs if isinstance(xs, list) els

mlb = MultiLabelBinarizer(sparse_output=True)
X = mlb.fit_transform(cards)  # scipy sparse matrix (memory efficient)

# If you need it as a pandas DataFrame, keep it SPARSE (do NOT densify)
dummies = (
    pd.DataFrame.sparse.from_spmatrix(
        X,
        index=df2.index,
        columns=[f"cards__{c}" for c in mlb.classes_],
    )
)

df2 = df2.join(dummies)
df2
```

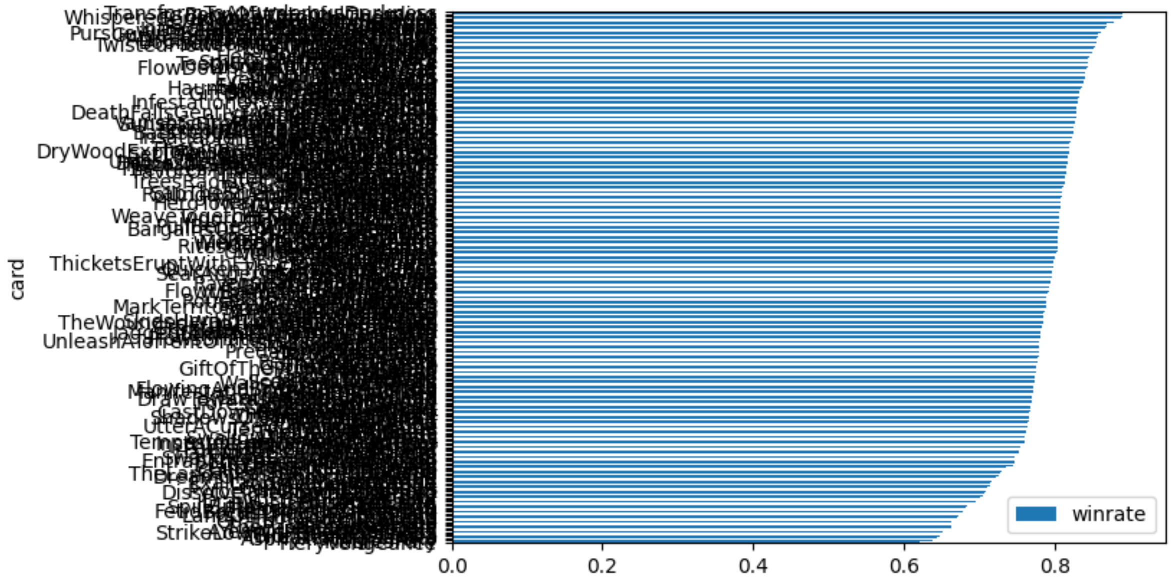
	Unnamed: 0.1	objectId	spirits	boards	powe
0	0	W1LHHUuDSB	['FathomlessMudOfTheSwamp']	['E']	
1	1	3hkL6pzv04	['Thunderspeaker']	['D']	
2	2	F2nkABojOG	['SerpentSlumberingBeneathTheIsland']	['A']	
3	3	ceephrxuSm	['LureOfTheDeepWilderness']	['A']	
4	4	yQnauB7H7t	['LightningsSwiftStrike', 'VitalStrengthOfTheE...	['B', 'H']	
...	...	...	...	...	
17404	8111	oiABuW0AIs	['SerpentSlumberingBeneathTheIsland']	['D']	
17405	8112	GCkcyNGEUY	['SerpentSlumberingBeneathTheIsland']	['D']	
17406	8113	PJjs54RycW	['VitalStrengthOfTheEarth']	['D']	['V
17407	8114	IENBxLzHdn	['VolcanoLoomingHigh', 'LureOfTheDeepWilderness']	['A', 'C']	
17408	8115	8Q0BACF3NI	['SharpFangsBehindTheLeaves']	['A']	

15057 rows × 696 columns

```
df2['won'] = df2.apply(lambda x: x['endingResult'].startswith('Win'), ax
volc = df2[df2['cards__GiftOfConstancy']==1]

volc.groupby('won').size()
volc.groupby('won').size()[True]
all_cards = [column for column in df2.columns if 'cards__' in column]
```

```
# all_cards_winrate = {card: df2[card]gb[True] / [True] + [False] for ca
gb = []
for card in all_cards:
    vc = df2.loc[df2[card] == 1, "won"].value_counts()

    result = {
        "card": card[7:],
        "wins": int(vc.get(True, 0)),
        "losses": int(vc.get(False, 0)),
    }
    gb.append(result)
df_winrate = pd.DataFrame(gb)
df_winrate['winrate'] = df_winrate['wins'] / (df_winrate['wins'] + df_wi
df_winrate.sort_values('winrate').plot.barh(x='card', y='winrate')
```

```
<Axes: ylabel='card'>
```



```
df_winrate.sort_values('winrate', ascending=True).head(10)
```

	card	wins	losses	winrate
73	FieryVengeance	184	112	0.621622
76	FlamesFury	416	236	0.638037
8	AsphyxiatingSmoke	405	223	0.644904
226	ThreateningFlames	403	219	0.647910

```
 15        BlurTheArcOfYears  220  119  0.648968
199  StrikeLowWithSuddenFevers  244  130  0.652406
  1   AYearOfPerfectStillness  622  319  0.660999
142              Plaguebearers  224  114  0.662722
221       ThePastReturnsAgain  171   87  0.662791
  2            AbsoluteStasis  230  117  0.662824
```

```
df_winrate.sort_values('winrate', ascending=False).head(10)
```

```
                                card  wins  losses   winrate
233   TransformToAMurderousDarkness    46       5  0.901961
 19           BoonOfWatchfulGuarding   921     113  0.890716
259  WhisperedGuidanceThroughTheNight  838     103  0.890542
 56         EerieNoisesAndMovingTrees  853     106  0.889468
133             MysteriousAbductions   804     101  0.888398
135            OpenShiftingWaterways   606      83  0.879536
 85            FoulVaporsAndFetidMuck  554      83  0.869702
 67        ExaltationOfTangledGrowth   507      76  0.869640
118      IntractableThicketsAndThorns  647      98  0.868456
 64                     EntwinedPower  210      32  0.867769
```

My notes: TransformToAMurderousDarkness has the highest winrate, I suspect this is because this is the kind of card you only take when you've already won the game.

Notable here is that 3/4 of the top winrate cards are all starting powers for eyes watch from the trees, that's surprising, but that spirit is quite good, even for new players that spirit is harder to lose as compared to thers.

Which adversaries have the lowest winrate?

```
# lets do single adversaries first
single_adversaries = df2[~df2['adversary'].str.contains(',', na=False)]
```

```
single_adversaries['adversary'].unique()
```

```
array(['TheKingdomOfSweden', 'No Adversary', 'TheKingdomOfEngland',
       'TheKingdomOfFrance', 'TheTsardomOfRussia',
       'TheKingdomOfBrandenburgPrussia', 'TheHabsburgMonarchy'],
      dtype=object)
```

```python
# single_adversaries.groupby('adversary').value_counts()
single_adversaries['adversary'] = single_adversaries['adversary'].fillna
single_adversaries.groupby('adversary').size()
```

```
                                   0
                       adversary
                  No Adversary  6634
           TheHabsburgMonarchy   644
TheKingdomOfBrandenburgPrussia  2630
           TheKingdomOfEngland  1864
            TheKingdomOfFrance  1208
            TheKingdomOfSweden  1308
            TheTsardomOfRussia   769

dtype: int64
```

```python
single_adversaries.groupby('adversary')['won'].value_counts().unstack(fi
```

```
                             won  False  True
                       adversary
                  No Adversary   1100  5534
           TheHabsburgMonarchy    238   406
TheKingdomOfBrandenburgPrussia    563  2067
           TheKingdomOfEngland    625  1239
            TheKingdomOfFrance    491   717
            TheKingdomOfSweden    338   970
            TheTsardomOfRussia    296   473
```

```python
summary = single_adversaries.groupby('adversary').agg(
    games=('won', 'count'),
    wins=('won', 'sum')
)
```

```

summary['win_rate'] = summary['wins'] / summary['games']
summary.sort_values('win_rate', ascending=False)
# france, russia, habsburg have the worst
```

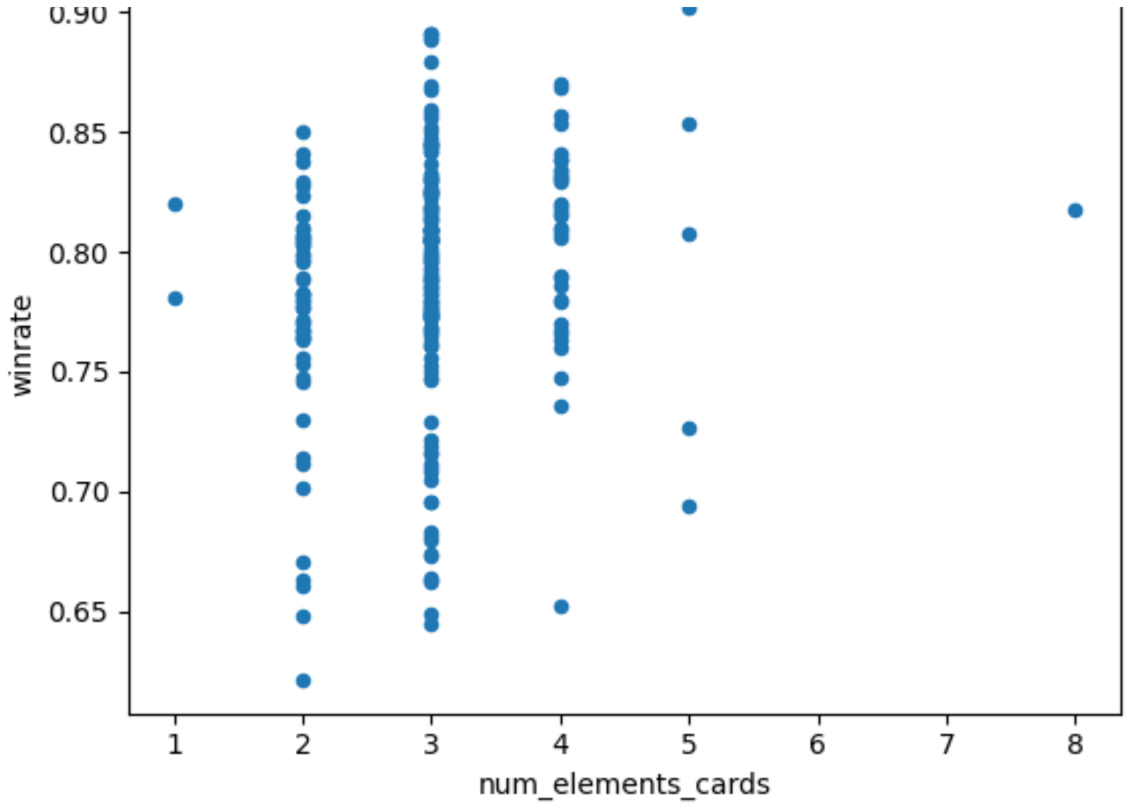
	games	wins	win_rate
adversary			
No Adversary	6634	5534	0.834188
TheKingdomOfBrandenburgPrussia	2630	2067	0.785932
TheKingdomOfSweden	1308	970	0.741590
TheKingdomOfEngland	1864	1239	0.664700
TheHabsburgMonarchy	644	406	0.630435
TheTsardomOfRussia	769	473	0.615085
TheKingdomOfFrance	1208	717	0.593543

france, russia, habsburg have the worst winrate, no adversary, prussia and sweden have the highest winrate.

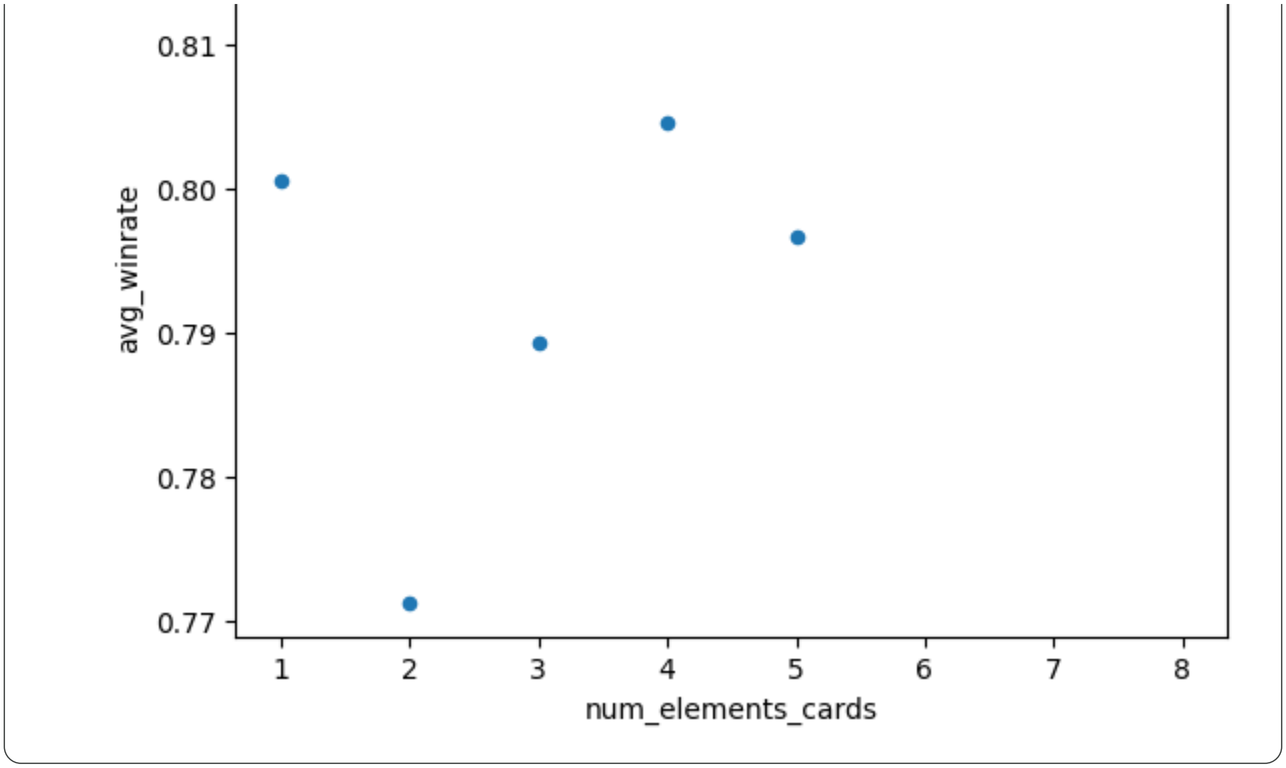
Is the number of elements on a card a good predictor for its winrate

```
df_cards = pd.read_csv('cards.csv')
df_cards
df_cards['num_elements'] = df_cards['elements'].str.count(',') + 1
df_cards[df_cards['elements'] == ""]
# df_cards['num_elements'].hist()
df_cards['name_no_space'] = df_cards['name'].str.replace(" ", "")
# df_winrate['num_elements'] = df_winrate['card'].map(df_cards.set_index
# df_winrate.drop('num_elements')
# df_cards = df_cards.reset_index()
df_cards['name_no_space'] = df_cards['name_no_space'].str.lower()
df_winrate['card'] = df_winrate['card'].str.lower()
wr_card_df = df_winrate.join(df_cards.set_index('name_no_space'), on='ca
wr_card_df
wr_card_df.plot.scatter(x='num_elements_cards', y='winrate')

```

```
<Axes: xlabel='num_elements_cards', ylabel='winrate'>
```



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

avg_df = (
    wr_card_df[['num_elements_cards', 'winrate']]
        .assign(
            num_elements_cards=lambda d: pd.to_numeric(d['num_elements_c
            winrate=lambda d: pd.to_numeric(d['winrate'], errors='coerce
        )
        .replace([np.inf, -np.inf], np.nan)
        .dropna()
        .groupby('num_elements_cards', as_index=False)
        .agg(
            avg_winrate=('winrate', 'mean'),
            games=('winrate', 'count')
        )
)
ax = avg_df.plot.scatter(
    x='num_elements_cards',
    y='avg_winrate'
)
# looks like 2, 3 costs have lower winrate than 1, 4, 5, 8
```



Not really, winrate for 2, 3 elements is lower than 1, and 5 is lower than 4. I didn't have 0

other questions

↳ 24 cells hidden