Pandas Test - Real life exercise

In [1]:
```
import pandas as pd
```

For visualizations:

In [2]:
```
import cufflinks as cf; cf.go_offline()
import plotly_express as px
```

About the data

The data you are about to load is coming from an Ad recommentation system of the company InBrain. InBrain is a adTech company and specalize in next-level ad recommendation.
The company has two major products - one giving high quality recommendation while the other is giving an highly yielding recommendation. Inbrain custormers are sending requests, asking for either of the products.

Once a week, the company is doing an internal quality assements, and sends a sample of its traffic requests to external annotation process, in which the annotators are trained to manually decided whether the recommended Ad is a succes or not. The data contains information about the ad (**ad_id**), the sampled week (**week_id**), the requester company (**requester**), the region of operation (**region**), the recommendation type (Quality/Yield, **recommendation_type**) and the recommendation provider algorithm (**rec_provider**). The annotators are marking whether the recommendation is meeting the Quality/Yield standards (**is_success**) and whether or not the recommendation is severly defected (**is_sever**)

See a sample below:

In [3]:
```
ad_recs_annotated = pd.read_csv('data/ad_annotations.csv')
ad_recs_annotated.sample(5)
```

Out[3]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_pr
354004	2022-week_06	AD008HZFAS	Fancy	US	yes	NaN	Qality	Ma
24256	2021-week_34	AD07G89L8	Search Million Culture	JP	yes	NaN	Qality	
251604	2021-week_50	AD078Z8M8K	Puentes Company	IN	yes	NaN	Yield	Ma
162450	2021-week_44	AD06XJ7X8L	Search Million Culture	GB	yes	NaN	Qality	Use
85763	2021-week_38	AD089RCT8N	Fancy	US	yes	NaN	Qality	

Your job, as the new and only data scientist of the company, is to get familiar with the the data, show week-over-week trends and produce insightfull graphs as a preparation to a full blown BI dashboard.

Questions

Data Modifications

- Add a column with the sample year
- Add a column with the sample quarter (google how to)
- Add a new success column with values 1 and 0 for yes and no
- Are there duplicated ads? To compac the data, remove duplicated ads and instead add an ad_count column **(pay attention, this is not trivial at all)**
- Are there any NaNs in the is_sever column? Count how many and create a new column with NaNs filled as False. Check that indeed the new column contaion no NaNs.
- Capitlize (first letter only) the is_success column

Subset selection

for each question, sub-select the data by using the conditional selection (`[]`) and the `.query` API. Use `.shape` on the subselection to obtain the result.
For example: `df.query('some_condition').shape`

1. How many requests are there in US region?
2. How many **successful** requests are there in US region?
3. How many **successful** requests are there in US region, on week 52?
4. How many requests where delivered by DDNQ, RRNY and UserPopQ together?

5. How many requests where delivered by rule based providers?
6. Select only the requests from 2021 - How many are there?
7. Select the reqeusts from week 30 to week 40 - How many are there?
8. Select all the data that comes from the most popular Ad
9. Select all the data that comes from the least popular provider
10. Select the data in which is_sever is None. How big is it?
11. Select the data in which the requester is a 3 word company
12. Select the data in which the requester is a 3 word company, and the ad_id has the letter 6 in it
13. Select the data in which the requester is a 3 word company, and the multiplication of all the numbers in the ad_id is bigger than 30

Analysis

1. How many weeks available in the data? Are there any missing weeks?
2. How many regions available in the data? Are there any missing weeks per region?
3. How many ads are being sent to annotation in each region per week?
4. How many None values are there in is_sever column?
5. Are ads being sent more than one time in a given week? what is the ad_id that was sent the most in a given week? (e.g. ad X where sent 50 times in week Y)
6. Are there ads that are being sent in more than one week? groupby ad_id week idmax
    A. Which is the ad that was sent in most amount of weeks (e.g. ad X where sent in Y different weeks)
    B. What are the weeks that the ad from above was sent in?
    C. Is there an Ad that was succsefull in one week, but not succesfful in a different week? Show one.
7. When is_sever is None, what is the number of successful requests? What is the number of non-successful requests? What do you learn from it about the reason for Nones in the first place?
8. Per each region, What is the Quality/Yield traffic proportion WoW? week over week, over time Proportion between quality and yield. needs to be sum = 1, per week
9. How many different reqeusters are there?
10. Which are the top 5 requesters per region?
11. Which are the different rec_proivders?
12. Are there different rec providers per region?
13. Are there different rec providers per rec type?
14. What are the notation rules distinguishing between quality vs yielding providers? meafien a shmot shel a provider, mistaim v "q" = quality kol ma sh magia mi provider q u mi quiality

1. Which is the most successful region of operation?
2. Which is the most successful rec provider per each region?
3. Present a table with a success rate, sever defects rate and CI (for each metric) per region

4. Present a table with a success rate, sever defects rate and CI (for each metric) per rec provider
5. Present a table with a success rate, sever defects rate and CI (for each metric) per region and rec provider combinations
6. Present a table with a success rate, sever defects rate and CI (for each metric) per rec type (Q/Y)
7. Present a table with a success rate, sever defects rate and CI (for each metric) per rec type and region
8. Present WoW table/graph with success rate and CI (see image below) per region
9. Present WoW table/graph with success rate and CI (see image below) per region and rec type
10. Present WoW table/graph with success rate and CI (see image below) per region and rec provider
11. Which are the requester to which we succeed the most?
12. Which are the requester to which we produce the most critical defects?
13. What is the overall success rate trend over time?
14. What is the overall sever defect rate trend over time?
15. Preset a WoW graph showing the number of requests per customer in each region (hint: use stacked bars), from it:
    A. Identify major traffic shape changes (addition/removal) of major clients
16. Preset a WoW graph showing the **requests proportion** per customer in each region (hint: use stacked bars), from it:
    A. Identify major traffic shape changes (addition/removal) of major clients

Analysis Bonus questions:

1. Compute the per region success rate and CI in trailing weeks, Say, 4 weeks instead of 1 week - to allow for smoother estimations and smaller CIs

Merges and joins

The Wow samples and annotations task were sent and recieve seperatly, per each week. The dataset you were working on was constructed from those files. You can see the files under the `data/weekly` folder, here are the first 10 files:

In [ ]:
```
import os
weekly_files = os.listdir('data/weekly/')
sorted(weekly_files)[:10]
```

Your task is to reconstruct the dataset above.

Visualizations

Produce the following success rate graph per region:

In [ ]:

Produce the following requestser proportion graph:

In [ ]:

Appendix

Code to create weekly files:

In [ ]:
```
# for l in ad_recs_annotated.region.unique():
#     for w in ad_recs_annotated.week_id.unique():
#         w_id = w.split('_')[1]
#         y = w.split('-')[0]
#         query = f'week_id == "{w}" and region == "{l}"'
#         sample_f_name = f'{y}_{w_id}_{l}__Sample.csv'
#         ad_recs_annotated.query(query)[['week_id','ad_id','requester','region','recc
#         for s in ad_recs_annotated.recommendation_type.unique():
#             ann_f_name = f'{y}_{w_id}_{l}_{s}_annotation_result.csv'
#             query = f'week_id == "{w}" and region == "{l}" and recommendation_type =
#             sss = ad_recs_annotated.query(query)
#             if sss.empty:
#                 continue
#             sss[['region','ad_id','is_success','is_sever']].drop_duplicates(subset=[
```

Answers

Import libraries

In [1]:
```
import pandas as pd
```

In [2]:
```
import cufflinks as cf; cf.go_offline()
import plotly_express as px
import ipywidgets as widgets
from ipywidgets import interact
```

Import database

In [3]:
```
ad_recs_annotated = pd.read_csv('data/ad_annotations.csv')
ad_recs_annotated.sample(5)
```

Out[3]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_p
253108	2021-week_50	AD06Y8PF78	RelayFoods	IN	yes	NaN	Qality	U
138528	2021-week_42	AD078F99RP	RelayFoods	JP	no	True	Qality	
180237	2021-week_45	AD08THQVTF	Puentes Company	IN	no	True	Yield	U
288776	2022-week_02	AD00Q8IG0	Bizanga	GB	yes	NaN	Yield	
201560	2021-week_47	AD08TMM69Z	Fry Multimedia	DE	yes	NaN	Qality	

Data Modifications

A1

In [4]:
```
#Add year column
ad_recs_annotated['year'] = ad_recs_annotated.week_id.apply(lambda x: x[:4])
ad_recs_annotated['year'] = ad_recs_annotated['year'].astype('int')
ad_recs_annotated.head(1)
```

Out[4]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_provider
0	2021-week_33	AD0088VOS	Search Million Culture	DE	yes	NaN	Qality	DNNQ

In [5]:
```
#Add week column for future use
ad_recs_annotated['week'] = ad_recs_annotated['week_id'].str[-2:]
ad_recs_annotated['week'] = ad_recs_annotated['week'].astype('int')
ad_recs_annotated.head(1)
```

Out[5]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_provider
0	2021-week_33	AD0088VOS	Search Million Culture	DE	yes	NaN	Qality	DNNQ

A2

In [6]:
```
wk = ad_recs_annotated['week'].astype('int')
ad_recs_annotated['quarter'] = ((wk-1) // 13) + 1
ad_recs_annotated.head(1)
```

Out[6]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_provider
0	2021-week_33	AD0088VOS	Search Million Culture	DE	yes	NaN	Qality	DNNQ

A3

In [7]:
```
ad_recs_annotated['new_success'] = ad_recs_annotated['is_success'].apply(lambda x: 1 i
ad_recs_annotated.sample(5)
```

Out[7]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_pr
316136	2022-week_04	AD088J8KZ8	MoJoe Brewing Company	DE	yes	NaN	Yield	Ma
352519	2022-week_06	AD08JGFRND	Search Million Culture	JP	no	True	Qality	Use
360686	2022-week_52	AD098PKPT8	MoJoe Brewing Company	GB	yes	NaN	Yield	Use
285077	2022-week_01	AD078K8F7Z	Fancy	US	yes	NaN	Qality	
268355	2021-week_51	AD08L8Y6M9	Extreme DA	US	yes	NaN	Yield	RuleE

A4

First we will check if there any duplicated ad.

In [8]:
```
ad_recs_annotated.duplicated(subset='ad_id').sum()
```

Out[8]:
```
73061
```

Conclusion: We can see that there are duplicates in ad_id column.

In [9]:
```
ad_recs_annotated['ad_count'] = 1
ad_recs_annotated.groupby(['ad_id']).ad_count.count().reset_index()
```

Out[9]:

	ad_id	ad_count
0	AD00000088	1
1	AD000000WF	1
2	AD00000876	1
3	AD00000888	2
4	AD0000088C	1
...	...	...
300722	AD98980890	1
300723	AD98988688	1
300724	AD98988898	2
300725	AD99798888	1
300726	ADOO8X898E	2

300727 rows × 2 columns

In [10]:
```
#Check the results with an alternative code
no_duplicates = ad_recs_annotated.pivot_table(columns=['ad_id'], aggfunc='size',margir
print(no_duplicates)
```

```
ad_id
AD00000088    1
AD000000WF    1
AD00000876    1
AD00000888    2
AD0000088C    1
             ..
AD98980890    1
AD98988688    1
AD98988898    2
AD99798888    1
ADOO8X898E    2
Length: 300727, dtype: int64
```

Finally we will check if our first code works well

In [11]:
```
no_rows = ad_recs_annotated['ad_id'].shape
sum_ad_count = ad_recs_annotated['ad_count'].sum()
check_dup = no_rows - sum_ad_count
check_dup
```

Out[11]:
```
array([0])
```

A5

In [12]:
```
ad_recs_annotated.is_sever.isnull().sum()
```

Out[12]:
```
304195
```

Conclusion: Yes, there are NaN values in column is_server.

In [13]:
```
ad_recs_annotated['new_is_sever'] = ad_recs_annotated['is_sever'].fillna(False)
```

Conclusion: Yes, there are NaN values in column is_server.ad_recs_annotated.sample(5)

In [14]:
```
ad_recs_annotated.new_is_sever.isnull().sum()
```

Out[14]:
```
0
```

As we can see above, the code did its jobs and there ins't any NaN in the new is_sever column.

A6

In [15]:
```
ad_recs_annotated['is_success'] = ad_recs_annotated['is_success'].str.capitalize()
ad_recs_annotated.sample(5)
```

Out[15]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_p
205145	2021-week_47	AD08IUM7O8	RelayFoods	GB	Yes	NaN	Qality	
49008	2021-week_36	AD088DTJ88	LocalVox Media	IN	Yes	NaN	Yield	U
131121	2021-week_42	AD00NWG8SX	RelayFoods	DE	No	False	Qality	U
88007	2021-week_39	AD08MZALRD	Tab Solutions	DE	No	True	Qality	
289827	2022-week_02	AD08R7LMP8	RelayFoods	GB	Yes	NaN	Qality	

Subset selection

A1

In [16]:
```
ad_recs_annotated.query('region == "US"').shape[0]
```

Out[16]:
```
103846
```

In [17]:
```
ad_recs_annotated.query('region == "US"').shape[0]
```

Out[17]:
```
103846
```

A2

In [18]:
```
ad_recs_annotated.loc[((ad_recs_annotated['region'] == 'US') &
                       (ad_recs_annotated['new_success'] == 1))] .shape[0]
```

Out[18]:
```
88918
```

In [19]:
```
ad_recs_annotated.query('region == "US" & new_success == 1').shape[0]
```

Out[19]:
```
88918
```

A3

In [20]:
```
ad_recs_annotated.loc[((ad_recs_annotated['region'] == 'US') & (ad_recs_annotated['new
                       (ad_recs_annotated['week'] == 52))] .shape[0]
```

Out[20]:
```
3342
```

In [21]:
```
ad_recs_annotated.query('region == "US" & new_success == 1 & week == 52').shape[0]
```

Out[21]:
```
3342
```

A4

In [22]:
```
ad_recs_annotated.loc[((ad_recs_annotated['rec_provider'] == 'DDNQ') | (ad_recs_annota
                       (ad_recs_annotated['rec_provider'] == 'UserPopQ'))].shape[0]
```

Out[22]:
```
69937
```

In [23]:
```
ad_recs_annotated.query('rec_provider == "DDNQ" | rec_provider == "RRNY" | rec_provide
```

Out[23]:
```
69937
```

A5

In the code below we can see that there are 2 providers that answer the definition: **RuleBased** and **RuleBasedY**:

In [24]:
```
ad_recs_annotated.groupby(['rec_provider']).count()
```

Out[24]:

rec_provider	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type
BooksQ	1720	1720	1720	1720	1720	281	1720
BooksY	4150	4150	4150	4150	4150	84	4150
DNNQ	117424	117424	117424	117424	117424	16632	117424
DNNY	45116	45116	45116	45116	45116	5110	45116
ManualQ	13844	13844	13844	13844	13844	650	13844
ManualY	1982	1982	1982	1982	1982	187	1982
RNNQ	20983	20983	20983	20983	20983	2045	20983
RNNY	12732	12732	12732	12732	12732	1024	12732
RuleBased	182	182	182	182	182	6	182
RuleBasedY	28154	28154	28154	28154	28154	11977	28154
UserPopQ	69937	69937	69937	69937	69937	17656	69937
UserPopSelectionQ	2417	2417	2417	2417	2417	447	2417
UserPopSelectionY	21	21	21	21	21	3	21
UserPopY	38600	38600	38600	38600	38600	10549	38600
XGBQ	12250	12250	12250	12250	12250	2388	12250
XGBY	4276	4276	4276	4276	4276	554	4276

In [25]:
```
ad_recs_annotated.loc[((ad_recs_annotated['rec_provider'] == 'RuleBasedY' ) | (ad_recs
```

Out[25]:
```
28336
```

In [26]:
```
ad_recs_annotated.query('rec_provider == "RuleBasedY" | rec_provider == "RuleBased"').
```

Out[26]:
```
28336
```

Alternative code:

In [27]:
```
ad_recs_annotated[ad_recs_annotated['rec_provider'].str.contains('RuleBased')].shape[0
```

Out[27]:
```
28336
```

A6

Note: It was not clear from the question if we were asked to query all the records from 2021 onwards or only those that from 2021 only. I chose the first option: records from 2021 onwards.

In [28]:
```
ad_recs_annotated.loc[((ad_recs_annotated['year'] >= 2021))].shape[0]
```

Out[28]:
```
373788
```

In [29]:
```
ad_recs_annotated.query('year >= 2021').shape[0]
```

Out[29]:
```
373788
```

A7

In [30]:
```
ad_recs_annotated.loc[((ad_recs_annotated['week'] >= 30) &
                       (ad_recs_annotated['week'] <= 40))].shape[0]
```

Out[30]:
```
115051
```

In [31]:
```
ad_recs_annotated.query('week >= 30 & week <= 40').shape[0]
```

Out[31]:
```
115051
```

A8

In [32]:
```
most_popular_ad = ad_recs_annotated['ad_id'].value_counts().idxmax() # I've assumed th
print(most_popular_ad)
```

```
AD07PFFMP9
```

In [33]:
```
print(ad_recs_annotated['ad_id'].value_counts()) # In order to prove that I get the sa
```

```
AD07PFFMP9    247
AD098SWYF6    239
AD08C8RR8J    138
AD0886VY87     99
AD08888888     96
             ...
AD07Y8JSHJ      1
AD0888J7AO      1
AD087G7SHR      1
AD086QCZGJ      1
AD08FWNFDO      1
Name: ad_id, Length: 300727, dtype: int64
```

In [34]:
```
ad_recs_annotated.query('ad_id == @most_popular_ad').shape[0]
```

Out[34]:
```
247
```

In [35]:
```
ad_recs_annotated.loc[ad_recs_annotated.ad_id == most_popular_ad].shape[0]
```

Out[35]:
```
247
```

A9

In [36]:
```
least_popular_provider = ad_recs_annotated['rec_provider'].value_counts().index[-1]
print(least_popular_provider)
```

```
UserPopSelectionY
```

In [37]:
```
print(ad_recs_annotated['rec_provider'].value_counts()) #in order to prove that i get
```

```
DNNQ                117424
UserPopQ             69937
DNNY                 45116
UserPopY             38600
RuleBasedY           28154
RNNQ                 20983
ManualQ              13844
RNNY                 12732
XGBQ                 12250
XGBY                  4276
BooksY                4150
UserPopSelectionQ     2417
ManualY               1982
BooksQ                1720
RuleBased              182
UserPopSelectionY       21
Name: rec_provider, dtype: int64
```

In [38]:
```
ad_recs_annotated.query('rec_provider == @least_popular_provider').shape[0]
```

Out[38]:
```
21
```

In [39]:
```
ad_recs_annotated.loc[ad_recs_annotated.rec_provider == least_popular_provider].shape[
```

Out[39]:
```
21
```

A10

In [40]:
```
ad_recs_annotated['is_sever'].isna().sum()
```

Out[40]:
```
304195
```

A11

In [41]:
```
ad_recs_annotated.loc[(ad_recs_annotated['requester'].str.count(' ') + 1) == 3].shape[
```

Out[41]:
```
118141
```

In [42]:
```
ad_recs_annotated.loc[(ad_recs_annotated['requester'].str.count(' ') + 1) == 3].sample
```

Out[42]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_pr
232147	2021-week_49	AD0088TU6F	Search Million Culture	DE	Yes	NaN	Qality	I
247659	2021-week_50	AD0088D878	Search Million Culture	GB	Yes	NaN	Qality	I
72593	2021-week_38	AD0788MRYS	Search Million Culture	DE	Yes	NaN	Qality	I
364798	2022-week_52	AD07F88KJX	MoJoe Brewing Company	IN	Yes	NaN	Yield	Use
224243	2021-week_48	AD08J887M6	MoJoe Brewing Company	IN	Yes	NaN	Yield	RuleE

A12

In [43]:
```
query12 = ad_recs_annotated.loc[( (ad_recs_annotated['requester'].str.count(' ') + 1)
                        (ad_recs_annotated['ad_id'].str.contains('6'))]
query12.shape[0]
```

Out[43]:
```
24197
```

In [44]:
```
query12.sample(5) #checking some examples
```

Out[44]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_pr
174705	2021-week_45	AD88888688	MoJoe Brewing Company	DE	Yes	NaN	Yield	E
307143	2022-week_03	AD09G8K868	Search Million Culture	IN	Yes	NaN	Yield	RuleE
246484	2021-week_50	AD098M68ZV	MoJoe Brewing Company	DE	Yes	NaN	Yield	
288719	2022-week_02	AD06Y8Z7HR	Search Million Culture	GB	Yes	NaN	Qality	
334290	2022-week_05	AD08MTXR6T	Search Million Culture	GB	Yes	NaN	Qality	

A13

Assumption for solving this question - All the digits that appear together were considered as a number. For example "AD07KYS8JM" the numbers that appear in this ad_id are: 07 and 8 (ie 7 and 8).

In [45]:
```
import re
def find_number(text):
    num = re.findall(r'[0-9]+',text)
    return " ".join(num)

number=ad_recs_annotated['ad_id'].apply(lambda x: find_number(x))
number.str.strip()
```

Out[45]:
```
0             0088
1             07 8
2           08 6 9
3         89608808
4             07 6
            ...
373783        08 8
373784        07 8
373785      096 88
373786          08
373787        00 8
Name: ad_id, Length: 373788, dtype: object
```

In [46]:
```
def remove(string):
    return string.replace(" ", "")
number_no_spaces = number.apply(lambda x: remove(x))
```

In [47]:
```
number_no_spaces = number_no_spaces.astype('int')
```

In [48]:
```
def no_mult(x):
    sum = 1

    while x > 0:
        d = x%10
        x = x//10
        sum *= d
    return sum

numb_mult=number_no_spaces.apply(lambda x: no_mult(x))
```

In [49]:
```
query13 = ad_recs_annotated.loc[( (ad_recs_annotated['requester'].str.count(' ') + 1)
                        (numb_mult > 30)]
query13.shape[0]
```

Out[49]:
```
95585
```

In [50]:
```
query13.sample(5) #checking some examples
```

Out[50]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_pr
306246	2022-week_03	AD08HWH8P6	Search Million Culture	IN	Yes	NaN	Qality	Use
280927	2022-week_01	AD098Y8VLD	MoJoe Brewing Company	JP	Yes	NaN	Yield	
66536	2021-week_37	AD07TXCN87	MoJoe Brewing Company	JP	No	True	Yield	Us
65821	2021-week_37	AD008QUZ8Z	Search Million Culture	IN	Yes	NaN	Qality	Use
109963	2021-week_40	AD098GZL7F	Search Million Culture	JP	Yes	NaN	Qality	

Analysis

A1

In [51]:
```
ad_recs_annotated.value_counts('week').count()
```

Out[51]:
```
26
```

In [52]:
```
ad_recs_annotated.week.unique()
```

Out[52]:
```
array([33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
       50, 51,  1,  2,  3,  4,  5,  6, 52])
```

In [53]:
```
print(ad_recs_annotated[ad_recs_annotated['week'].isnull()])
```

```
Empty DataFrame
Columns: [week_id, ad_id, requester, region, is_success, is_sever, recommendation_typ
e, rec_provider, year, week, quarter, new_success, ad_count, new_is_sever]
Index: []
```

Conclusion: We don't see any missing week.

A2

In [54]:
```
ad_recs_annotated.value_counts('region').count()
```

Out[54]:
```
5
```

In [55]:
```
ad_recs_annotated.region.unique()
```

Out[55]:
```
array(['DE', 'GB', 'IN', 'JP', 'US'], dtype=object)
```

In [56]:
```
print(ad_recs_annotated[ad_recs_annotated['region'].isnull()])
```

```
Empty DataFrame
Columns: [week_id, ad_id, requester, region, is_success, is_sever, recommendation_typ
e, rec_provider, year, week, quarter, new_success, ad_count, new_is_sever]
Index: []
```

Conclusion: We don't see any missing region.

A3

In [57]:
```
count_uniques = ad_recs_annotated.groupby('region').nunique().reset_index()
count_uniques
```

Out[57]:

	region	week_id	ad_id	requester	is_success	is_sever	recommendation_type	rec_provider	year
0	DE	26	54650	19	2	2	2	9	2
1	GB	26	64135	23	2	2	2	12	2
2	IN	26	61297	17	2	2	2	6	2
3	JP	25	38048	17	2	2	2	9	2
4	US	26	87323	34	2	2	2	14	2

In [58]:
```
query3 = count_uniques[['region','week','ad_id']]
query3
```

Out[58]:

	region	week	ad_id
0	DE	26	54650
1	GB	26	64135
2	IN	26	61297
3	JP	25	38048
4	US	26	87323

In [59]:
```
query3['ads_per_week'] = round(query3['ad_id']/query3['week'],2)
query3
```

Out[59]:

	region	week	ad_id	ads_per_week
0	DE	26	54650	2101.92
1	GB	26	64135	2466.73
2	IN	26	61297	2357.58
3	JP	25	38048	1521.92
4	US	26	87323	3358.58

A4

In [60]:
```
ad_recs_annotated['is_sever'].isna().sum()
```

Out[60]:
```
304195
```

A5

In [61]:
```
ad_recs_annotated[["ad_id","week"]].value_counts() > 1
```

Out[61]:
```
ad_id       week
AD098SWYF6  40       True
AD08C8RR8J  47       True
AD07QS8VCL  47       True
AD00IGCC8G  47       True
AD098C6SNV  47       True
                    ...
AD07GRVV8L  47      False
AD07GRVT8R  37      False
AD07GRTNSP  49      False
AD07GRTNPJ  37      False
ADOO8X898E  50      False
Length: 358763, dtype: bool
```

In [62]:
```
ad_recs_annotated[["ad_id","week"]].value_counts().idxmax()
```

Out[62]:
```
('AD098SWYF6', 40)
```

A6

In [63]:
```
sol6_df = ad_recs_annotated.groupby(['ad_id','week'])['week'].count()
sol6_df
```

Out[63]:
```
ad_id       week
AD00000088  37      1
AD000000WF  3       1
AD00000876  34      1
AD00000888  2       1
            43      1
                   ..
AD98988898  5       1
            43      1
AD99798888  42      1
ADOO8X898E  41      1
            50      1
Name: week, Length: 358763, dtype: int64
```

In [64]:
```
sol6_df_sorted = sol6_df.groupby(level=[0, 0]).transform("count").reset_index(name ='1
sol6_df_sorted
```

Out[64]:

	ad_id	week	Total Amount
0	AD00000088	37	1
226830	AD089GRKZT	34	1
226829	AD089GRKDJ	49	1
226826	AD089GRF68	35	1
226825	AD089GR8YY	34	1
...	...	...	...
104890	AD079VP6DH	48	26
104891	AD079VP6DH	49	26
104892	AD079VP6DH	50	26
104882	AD079VP6DH	40	26
338118	AD098ZJV6Z	6	26

358763 rows × 3 columns

Conclusion: We can see above that there are ads that were sent in more than 1 week.

In [65]:
```
ad_most_weeks = sol6_df_sorted[['ad_id','Total Amount']].value_counts().idxmax()
# In order to calculate the ad that was sent in the most number of weeks
ad_most_weeks
```

Out[65]:
```
('AD07PFFMP9', 26)
```

In [66]:
```
most_sold_add = ad_most_weeks[0]
print(most_sold_add) # Ad that was sent in the most number of weeks
```

```
AD07PFFMP9
```

In [67]:
```
most_sold_df = sol6_df_sorted.query('ad_id == @most_sold_add')
list(most_sold_df['week']) # List of the weeks from the most sold ad
```

Out[67]:
```
[5,
 2,
 3,
 51,
 50,
 49,
 48,
 47,
 46,
 45,
 44,
 52,
 43,
 41,
 40,
 39,
 38,
 37,
 36,
 35,
 34,
 33,
 4,
 42,
 1,
 6]
```

In [68]:
```
ad_recs_annotated.groupby(['ad_id','week'])['week'].count()
```

Out[68]:
```
ad_id       week
AD00000088  37      1
AD000000WF  3       1
AD00000876  34      1
AD00000888  2       1
            43      1
                   ..
AD98988898  5       1
            43      1
AD99798888  42      1
ADOO8X898E  41      1
            50      1
Name: week, Length: 358763, dtype: int64
```

In [69]:
```
df_new = ad_recs_annotated[['ad_id','week','new_success']]
df_new
```

Out[69]:

	ad_id	week	new_success
0	AD0088VOS	33	1
1	AD07KYS8JM	33	1
2	AD08PDP6Y9	33	1
3	AD89608808	33	1
4	AD07CMVHP6	33	1
...	...	...	...
373783	AD08KQ8GDG	52	1
373784	AD07C8XSMN	52	1
373785	AD096LRR88	52	1
373786	AD08FWNFDO	52	1
373787	AD00NATC8M	52	1

373788 rows × 3 columns

In [70]:
```
res = df_new.groupby(['ad_id','week'])['new_success'].agg(['count','sum'])
res
```

Out[70]:

		count	sum
ad_id	**week**		
AD00000088	**37**	1	1
AD000000WF	**3**	1	1
AD00000876	**34**	1	1
AD00000888	**2**	1	1
	43	1	1
...	**...**	...	...
AD98988898	**5**	1	1
	43	1	1
AD99798888	**42**	1	0
ADOO8X898E	**41**	1	1
	50	1	1

358763 rows × 2 columns

In [71]:
```
res2= res.groupby('ad_id')[['count','sum']].sum()
res2
```

Out[71]:

ad_id	count	sum
AD00000088	1	1
AD000000WF	1	1
AD00000876	1	1
AD00000888	2	2
AD0000088C	1	1
...	...	...
AD98980890	1	1
AD98988688	1	1
AD98988898	2	2
AD99798888	1	0
ADOO8X898E	2	2

300727 rows × 2 columns

In [72]:
```
res2['check'] = res2['count'] - res2['sum']
```

In [73]:
```
res2.loc[((res2['check'] != 0) & (res2['count'] > 1) & (res2['sum'] > 0))]
```

Out[73]:

ad_id	count	sum	check
AD00006IDK	2	1	1
AD00006IEI	4	3	1
AD00007F8Q	4	3	1
AD00007K6N	2	1	1
AD00008808	5	4	1
...	...	...	...
AD98888878	2	1	1
AD98888888	8	7	1
AD98889888	3	1	2
AD98898880	7	6	1
AD98908988	3	2	1

5418 rows × 3 columns

In [74]:
```
ad_recs_annotated.loc[ad_recs_annotated.ad_id == 'AD98888878'] # Example taken from th
```

Out[74]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_pro
21446	2021-week_34	AD98888878	Puentes Company	IN	Yes	NaN	Qality	User
165804	2021-week_44	AD98888878	Search Million Culture	IN	No	False	Qality	User

A7

In [75]:
```
ad_recs_annotated.loc[((ad_recs_annotated['is_sever'].isnull()) & (ad_recs_annotated['
```

Out[75]:
```
304189
```

In [76]:
```
ad_recs_annotated.loc[((ad_recs_annotated['is_sever'].isnull()) & (ad_recs_annotated['
```

Out[76]:
```
6
```

Conclusion: 'None" is_sever values are in fact 'False' entries, ie the recommendation is not severly defected.

A8

In [77]:
```
ad_recs_annotated['recommendation_type'].unique() # in order to know the values of the
```

Out[77]:
```
array(['Qality', 'Yield'], dtype=object)
```

We will build an indicator of quality/yield in order to facilitate further calculations.

In [78]:
```
ad_recs_annotated['rec_quality'] = ad_recs_annotated['recommendation_type'].apply(lamb
```

In [79]:
```
ad_recs_annotated['rec_yield'] = ad_recs_annotated['recommendation_type'].apply(lambda
```

In [80]:
```
query8 = ad_recs_annotated.groupby(['region','week'])[['rec_quality','rec_yield']].sum
query8
```

Out[80]:

	region	week	rec_quality	rec_yield
0	DE	1	2026	376
1	DE	2	2027	367
2	DE	3	1881	497
3	DE	4	2038	327
4	DE	5	1959	423
...	...	...	...	...
124	US	48	1237	2773
125	US	49	2047	1947
126	US	50	1865	2126
127	US	51	1208	2788
128	US	52	1299	2699

129 rows × 4 columns

In [81]:
```
query8['Q/Y prop']= round(query8['rec_quality'] / query8['rec_yield'] , 2)
query8.head(5)
```

Out[81]:

	region	week	rec_quality	rec_yield	Q/Y prop
0	DE	1	2026	376	5.39
1	DE	2	2027	367	5.52
2	DE	3	1881	497	3.78
3	DE	4	2038	327	6.23
4	DE	5	1959	423	4.63

A9

In [82]:
```
ad_recs_annotated.requester.nunique()
```

Out[82]: 36

A10

In [83]:
```
count_per_reg = ad_recs_annotated.groupby(['region','requester'])['requester'].count()
count_per_reg
```

Out[83]:

	region	requester	Total
0	DE	Allthetopbananas.com	51
1	DE	Bizanga	107
2	DE	Crescendo Networks	573
3	DE	Cuiker	6
4	DE	Extreme DA	115
...	...	...	...
105	US	Sensor Tower	3503
106	US	Tab Solutions	9790
107	US	VarVee	119
108	US	aPriori Technologies	57
109	US	iDreamsky Technology	1137

110 rows × 3 columns

In [84]:
```
top5 = count_per_reg.sort_values(by= ['region','Total'], ascending = [False,False]).gr
top5
```

Out[84]:

	region	requester	Total
85	US	Extreme DA	57215
86	US	Fancy	18668
106	US	Tab Solutions	9790
104	US	Search Million Culture	5610
105	US	Sensor Tower	3503
70	JP	RelayFoods	24574
71	JP	Search Million Culture	7906
66	JP	MoJoe Brewing Company	3549
65	JP	LocalVox Media	1724
72	JP	Tab Solutions	1366
53	IN	Puentes Company	29427
55	IN	Search Million Culture	21365
54	IN	RelayFoods	17352
49	IN	MoJoe Brewing Company	7191
58	IN	iDreamsky Technology	1995
37	GB	Search Million Culture	28002
35	GB	RelayFoods	15920
25	GB	Fancy	13678
31	GB	MoJoe Brewing Company	9530
41	GB	iDreamsky Technology	3919
14	DE	Search Million Culture	23657
12	DE	RelayFoods	17313
8	DE	MoJoe Brewing Company	8616
18	DE	iDreamsky Technology	3542
13	DE	SOLOMO365	2435

A11

In [85]:
```
print(ad_recs_annotated.groupby('rec_provider').groups.keys())
```

```
dict_keys(['BooksQ', 'BooksY', 'DNNQ', 'DNNY', 'ManualQ', 'ManualY', 'RNNQ', 'RNNY',
'RuleBased', 'RuleBasedY', 'UserPopQ', 'UserPopSelectionQ', 'UserPopSelectionY', 'Use
rPopY', 'XGBQ', 'XGBY'])
```

A12

In [86]:
```
ad_recs_annotated.groupby('region')['rec_provider'].unique().reset_index()
```

Out[86]:

	region	rec_provider
0	DE	[DNNQ, DNNY, BooksQ, UserPopQ, UserPopY, RuleB...
1	GB	[DNNY, XGBQ, DNNQ, XGBY, RNNQ, UserPopQ, Manua...
2	IN	[UserPopY, RuleBasedY, UserPopQ, ManualY, Manu...
3	JP	[ManualQ, UserPopQ, RuleBasedY, DNNQ, ManualY,...
4	US	[DNNY, ManualQ, DNNQ, UserPopY, XGBY, RNNY, Ma...

From the above table we can see that there is at least more than one provider in each region.

A13

In [87]:
```
ad_recs_annotated.groupby('recommendation_type')['rec_provider'].unique().reset_index(
```

Out[87]:

	recommendation_type	rec_provider
0	Qality	[DNNQ, BooksQ, UserPopQ, ManualY, ManualQ, XGB...
1	Yield	[DNNY, UserPopY, RuleBasedY, BooksY, ManualY, ...

From the above table we can see that there is at least more than one provider for each recommendation type.

A14

It seems that the last letter of the provider indicates the recommendation type. We are going to provide this in the following steps.

First we are going to pull the first letter for the recommendation type: Y or Q.

In [88]:
```
step1 = ad_recs_annotated['recommendation_type'].str[0:1]
step1.str.upper()
```

Out[88]:
```
0         Q
1         Y
2         Q
3         Q
4         Q
         ..
373783    Y
373784    Q
373785    Y
373786    Q
373787    Q
Name: recommendation_type, Length: 373788, dtype: object
```

Then we are going to pull the last letter from the rec provider (*hypothesis: Y or Q*).

In [89]:
```
step2 = ad_recs_annotated['rec_provider'].str[-1:]
step2.str.upper()
```

Out[89]:
```
0         Q
1         Y
2         Q
3         Q
4         Q
         ..
373783    Y
373784    Q
373785    Y
373786    Q
373787    Q
Name: rec_provider, Length: 373788, dtype: object
```

Afterwards we need to compare the first letter of the recommendation type with the last letter of the rec provider:

In [90]:
```
step3 = (step1 == step2)
step3
```

Out[90]:
```
0         True
1         True
2         True
3         True
4         True
          ...
373783    True
373784    True
373785    True
373786    True
373787    True
Length: 373788, dtype: bool
```

Here we add our control in the dataframe:

In [91]:
```
ad_recs_annotated['check_notation'] = step3
ad_recs_annotated.sample(2)
```

Out[91]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_pr
124250	2021-week_41	AD07878GFK	Search Million Culture	JP	Yes	NaN	Qality	
137872	2021-week_42	AD07TVJQL8	RelayFoods	IN	No	False	Qality	Use

Finally if we were right we need to see that all the values of our control are indeed **"True"**. If there are False values we are going to query them in the second code.

In [92]:
```
ad_recs_annotated.groupby('check_notation')['check_notation'].count().reset_index(name
```

Out[92]:

	check_notation	Total
0	False	5245
1	True	368543

In [93]:
```
ad_recs_annotated.loc[ad_recs_annotated.check_notation == False].head(5)
```

Out[93]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_prov
24	2021-week_33	AD07SNPKX8	Search Million Culture	DE	Yes	NaN	Qality	Man
313	2021-week_33	AD07PHPXHQ	iDreamsky Technology	DE	Yes	NaN	Qality	Man
314	2021-week_33	AD07PHPXHQ	Search Million Culture	DE	Yes	NaN	Qality	Man
315	2021-week_33	AD07PHPXHQ	iDreamsky Technology	DE	Yes	NaN	Qality	Man
316	2021-week_33	AD07PHPXHQ	Search Million Culture	DE	Yes	NaN	Qality	Man

Conclusion: In the same query we could see that there is a provider called "RuleBased" which doesn't have an indication of the recommendation type (Qality/Yield). Also the above query shoes that there are 5245 entries where the recommendation type doesn't seem to match re indication in rec_provider name. For example index 24, the recommendation_type is "Qality" but the last letter of rec_provider, ManualY, is indeed "Y". This should be further investigated in order to check if the data is wrong.

A15

In [94]:
```
success_set = ad_recs_annotated.query('new_success == 1')
success_set[["region","new_success"]].value_counts().idxmax()
```

Out[94]:
```
('US', 1)
```

In [95]:
```
query15 = success_set.groupby(['region'])['new_success'].count().reset_index() # in or
query15.tail(1)
```

Out[95]:

	region	new_success
4	US	88918

A16

In [96]:
```
success_set_region = success_set.groupby(['region','rec_provider'])['new_success'].cou
success_set_region.head(1)
```

Out[96]:

	region	rec_provider	new_success
0	DE	BooksQ	1082

In [97]:
```
def func(group):
    return group.loc[group['new_success'] == group['new_success'].max()]

success_set_region.groupby('region', as_index=False).apply(func).reset_index(drop=True
```

Out[97]:

	region	rec_provider	new_success
0	DE	DNNQ	27215
1	GB	DNNQ	34681
2	IN	UserPopQ	30407
3	JP	DNNQ	20113
4	US	DNNY	25705

A17

First we will add a severe indicator in order to make the calculation easier in the following codes.

In [98]:
```
ad_recs_annotated['sever_ind'] = ad_recs_annotated['new_is_sever'].apply(lambda x: 1 i
ad_recs_annotated.sample(2)
```

Out[98]:

	week_id	ad_id	requester	region	is_success	is_sever	recommendation_type	rec_pr
312398	2022-week_03	AD00MQG8Y8	Fancy	US	Yes	NaN	Qality	
213102	2021-week_47	AD08IS888S	Extreme DA	US	No	True	Yield	Rule

Then we will calculate the successs and severe defect rates. We add some extra information as n, success number of events (sum_succ) and severe defect numbers (sum_sev) in order to be sure that the calculations are ok.

In [99]:
```
d = {'region': ['count'],'new_success': ['sum','mean', 'std'], 'sever_ind': ['sum','me
new = ad_recs_annotated.groupby('region').agg(d).reset_index()
new.columns = ['region', 'n','sum_succ','succ_mean', 'succ_std', 'sum_sev','sev_mean',
new
```

Out[99]:

	region	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
0	DE	62177	50624	0.814192	0.388955	7063	0.113595	0.317321
1	GB	82984	69509	0.837619	0.368802	9929	0.119650	0.324553
2	IN	83154	63355	0.761900	0.425923	11736	0.141136	0.348164
3	JP	41627	31800	0.763927	0.424673	6457	0.155116	0.362019
4	US	103846	88918	0.856249	0.350839	9504	0.091520	0.288349

In [100…
```
from scipy import stats
import math
alpha = 0.05
n_sided = 2
```

In [101…
```
def ci(new):
    # confidence interval for sever indicator
    mean_sev = new['sev_mean']
    std_sev = new['sev_std']
    num = new['n']
    c = math.sqrt(num)
    z_crit = stats.norm.ppf(1-alpha/n_sided)
    ci_sev = (round(mean_sev - z_crit * std_sev/c,4), round(mean_sev + z_crit * std_se
    ci_sev_range = round(mean_sev + z_crit * std_sev/c,4) - round(mean_sev - z_crit *

    # confidence interval for success indicator
    mean_succ = new['succ_mean']
    std_succ = new['succ_std']
    num = new['n']
    c = math.sqrt(num)
    z_crit = stats.norm.ppf(1-alpha/n_sided)
    ci_succ = (round(mean_succ - z_crit * std_succ/c,4), round(mean_succ + z_crit * st
    ci_succ_range = round(mean_succ + z_crit * std_succ/c,4) - round(mean_succ - z_cri

    return pd.Series({'ci sever ind': ci_sev, 'ci success ind':ci_succ, 'ci_sev_range'
```

In [102…
```
res17 = new.groupby(['region','succ_mean','sev_mean']).apply(ci).reset_index()
res17
```

Out[102]:

	region	succ_mean	sev_mean	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	0.814192	0.113595	([0.1111], [0.1161])	([0.8111], [0.8172])	0 0.005 dtype: float64	0 0.0061 dtype: float64
1	GB	0.837619	0.119650	([0.1174], [0.1219])	([0.8351], [0.8401])	1 0.0045 dtype: float64	1 0.005 dtype: float64
2	IN	0.761900	0.141136	([0.1388], [0.1435])	([0.759], [0.7648])	2 0.0047 dtype: float64	2 0.0058 dtype: float64
3	JP	0.763927	0.155116	([0.1516], [0.1586])	([0.7598], [0.768])	3 0.007 dtype: float64	3 0.0082 dtype: float64
4	US	0.856249	0.091520	([0.0898], [0.0933])	([0.8541], [0.8584])	4 0.0035 dtype: float64	4 0.0043 dtype: float64

In [103…
```
res17['ci_succ_range'] = res17['ci_succ_range'].astype('float32')
```

```
res17['ci_sev_range'] = res17['ci_sev_range'].astype('float32')
res17
```

Out[103]:

	region	succ_mean	sev_mean	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	0.814192	0.113595	([0.1111], [0.1161])	([0.8111], [0.8172])	0.0050	0.0061
1	GB	0.837619	0.119650	([0.1174], [0.1219])	([0.8351], [0.8401])	0.0045	0.0050
2	IN	0.761900	0.141136	([0.1388], [0.1435])	([0.759], [0.7648])	0.0047	0.0058
3	JP	0.763927	0.155116	([0.1516], [0.1586])	([0.7598], [0.768])	0.0070	0.0082
4	US	0.856249	0.091520	([0.0898], [0.0933])	([0.8541], [0.8584])	0.0035	0.0043

A18

In [104…

```
d = {'rec_provider': ['count'],'new_success': ['sum','mean', 'std'], 'sever_ind': ['su
new = ad_recs_annotated.groupby('rec_provider').agg(d).reset_index()
new.columns = ['rec_provider', 'n','sum_succ','succ_mean', 'succ_std', 'sum_sev','sev_
new
```

Out[104]:

	rec_provider	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
0	BooksQ	1720	1439	0.836628	0.369812	28	0.016279	0.126583
1	BooksY	4150	4066	0.979759	0.140841	22	0.005301	0.072625
2	DNNQ	117424	100794	0.858376	0.348665	13980	0.119056	0.323855
3	DNNY	45116	40005	0.886714	0.316945	3866	0.085690	0.279909
4	ManualQ	13844	13196	0.953193	0.211233	342	0.024704	0.155227
5	ManualY	1982	1795	0.905651	0.292388	99	0.049950	0.217896
6	RNNQ	20983	18938	0.902540	0.296590	1545	0.073631	0.261176
7	RNNY	12732	11709	0.919651	0.271843	636	0.049953	0.217856
8	RuleBased	182	176	0.967033	0.179043	6	0.032967	0.179043
9	RuleBasedY	28154	16176	0.574554	0.494419	6303	0.223876	0.416847
10	UserPopQ	69937	52283	0.747573	0.434408	8388	0.119937	0.324890
11	UserPopSelectionQ	2417	1970	0.815060	0.388329	269	0.111295	0.314562
12	UserPopSelectionY	21	18	0.857143	0.358569	1	0.047619	0.218218
13	UserPopY	38600	28056	0.726839	0.445588	7512	0.194611	0.395906
14	XGBQ	12250	9863	0.805143	0.396107	1297	0.105878	0.307693
15	XGBY	4276	3722	0.870440	0.335858	395	0.092376	0.289590

In [105…

```
res = new.groupby('rec_provider').apply(ci).reset_index()
res
```

Out[105]:

	rec_provider	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	BooksQ	([0.0103], [0.0223])	([0.8192], [0.8541])	0 0.012 dtype: float64	0 0.0349 dtype: float64
1	BooksY	([0.0031], [0.0075])	([0.9755], [0.984])	1 0.0044 dtype: float64	1 0.0085 dtype: float64
2	DNNQ	([0.1172], [0.1209])	([0.8564], [0.8604])	2 0.0037 dtype: float64	2 0.004 dtype: float64
3	DNNY	([0.0831], [0.0883])	([0.8838], [0.8896])	3 0.0052 dtype: float64	3 0.0058 dtype: float64
4	ManualQ	([0.0221], [0.0273])	([0.9497], [0.9567])	4 0.0052 dtype: float64	4 0.007 dtype: float64
5	ManualY	([0.0404], [0.0595])	([0.8928], [0.9185])	5 0.0191 dtype: float64	5 0.0257 dtype: float64
6	RNNQ	([0.0701], [0.0772])	([0.8985], [0.9066])	6 0.0071 dtype: float64	6 0.0081 dtype: float64
7	RNNY	([0.0462], [0.0537])	([0.9149], [0.9244])	7 0.0075 dtype: float64	7 0.0095 dtype: float64
8	RuleBased	([0.007], [0.059])	([0.941], [0.993])	8 0.052 dtype: float64	8 0.052 dtype: float64
9	RuleBasedY	([0.219], [0.2287])	([0.5688], [0.5803])	9 0.0097 dtype: float64	9 0.0115 dtype: float64
10	UserPopQ	([0.1175], [0.1223])	([0.7444], [0.7508])	10 0.0048 dtype: float64	10 0.0064 dtype: float64
11	UserPopSelectionQ	([0.0988], [0.1238])	([0.7996], [0.8305])	11 0.025 dtype: float64	11 0.0309 dtype: float64
12	UserPopSelectionY	([-0.0457], [0.141])	([0.7038], [1.0105])	12 0.1867 dtype: float64	12 0.3067 dtype: float64
13	UserPopY	([0.1907], [0.1986])	([0.7224], [0.7313])	13 0.0079 dtype: float64	13 0.0089 dtype: float64
14	XGBQ	([0.1004], [0.1113])	([0.7981], [0.8122])	14 0.0109 dtype: float64	14 0.0141 dtype: float64
15	XGBY	([0.0837], [0.1011])	([0.8604], [0.8805])	15 0.0174 dtype: float64	15 0.0201 dtype: float64

A19

In [106…

```
d = {'region': ['count'],'new_success': ['sum','mean', 'std'], 'sever_ind': ['sum','me
new = ad_recs_annotated.groupby(['region','rec_provider']).agg(d).reset_index()
new.columns = ['region','rec_provider', 'n','sum_succ','succ_mean', 'succ_std', 'sum_s
new
```

Out[106]:

	region	rec_provider	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
0	DE	BooksQ	1353	1082	0.799704	0.400370	26	0.019217	0.137336
1	DE	BooksY	1257	1215	0.966587	0.179784	10	0.007955	0.088873
2	DE	DNNQ	32433	27215	0.839114	0.367431	3985	0.122869	0.328291
3	DE	DNNY	5417	4969	0.917297	0.275458	270	0.049843	0.217641
4	DE	ManualQ	968	830	0.857438	0.349806	47	0.048554	0.215044
5	DE	ManualY	605	581	0.960331	0.195343	24	0.039669	0.195343
6	DE	RuleBasedY	1614	1022	0.633209	0.482078	173	0.107187	0.309447
7	DE	UserPopQ	15179	11098	0.731142	0.443381	2034	0.134001	0.340665
8	DE	UserPopY	3351	2612	0.779469	0.414667	494	0.147419	0.354576
9	GB	DNNQ	39675	34681	0.874127	0.331710	4339	0.109364	0.312099
10	GB	DNNY	7273	6240	0.857968	0.349107	810	0.111371	0.314612
11	GB	ManualQ	2154	2082	0.966574	0.179788	47	0.021820	0.146129
12	GB	ManualY	553	483	0.873418	0.332805	46	0.083183	0.276408
13	GB	RNNQ	11369	10147	0.892515	0.309743	926	0.081450	0.273536
14	GB	RNNY	1548	1350	0.872093	0.334094	133	0.085917	0.280332
15	GB	RuleBased	40	39	0.975000	0.158114	1	0.025000	0.158114
16	GB	RuleBasedY	1849	697	0.376961	0.484756	942	0.509465	0.500046
17	GB	UserPopQ	6781	4561	0.672615	0.469294	1178	0.173721	0.378897
18	GB	UserPopY	1642	1082	0.658952	0.474206	426	0.259440	0.438461
19	GB	XGBQ	9136	7316	0.800788	0.399430	985	0.107815	0.310164
20	GB	XGBY	964	831	0.862033	0.345044	96	0.099585	0.299601
21	IN	ManualQ	5293	5041	0.952390	0.212960	124	0.023427	0.151270
22	IN	ManualY	225	212	0.942222	0.233843	0	0.000000	0.000000
23	IN	RuleBased	7	6	0.857143	0.377964	1	0.142857	0.377964
24	IN	RuleBasedY	14107	8859	0.627986	0.483359	3196	0.226554	0.418617
25	IN	UserPopQ	38225	30407	0.795474	0.403360	3590	0.093918	0.291718
26	IN	UserPopY	25297	18830	0.744357	0.436231	4825	0.190734	0.392888
27	JP	DNNQ	24574	20113	0.818467	0.385467	4017	0.163465	0.369797
28	JP	DNNY	3859	3091	0.800985	0.399311	537	0.139155	0.346153
29	JP	ManualQ	697	651	0.934003	0.248455	25	0.035868	0.186095
30	JP	ManualY	125	68	0.544000	0.500065	13	0.104000	0.306489
31	JP	RuleBasedY	627	203	0.323764	0.468285	138	0.220096	0.414642
32	JP	UserPopQ	7930	4990	0.629256	0.483034	1244	0.156873	0.363704

	region	rec_provider	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
33	JP	UserPopSelectionQ	2417	1970	0.815060	0.388329	269	0.111295	0.314562
34	JP	UserPopSelectionY	21	18	0.857143	0.358569	1	0.047619	0.218218
35	JP	UserPopY	1377	696	0.505447	0.500152	213	0.154684	0.361735
36	US	BooksQ	367	357	0.972752	0.163027	2	0.005450	0.073720
37	US	BooksY	2893	2851	0.985482	0.119633	12	0.004148	0.064282
38	US	DNNQ	20742	18785	0.905650	0.292322	1639	0.079018	0.269774
39	US	DNNY	28567	25705	0.899814	0.300252	2249	0.078727	0.269317
40	US	ManualQ	4732	4592	0.970414	0.169460	99	0.020921	0.143136
41	US	ManualY	474	451	0.951477	0.215096	16	0.033755	0.180789
42	US	RNNQ	9614	8791	0.914396	0.279793	619	0.064385	0.245451
43	US	RNNY	11184	10359	0.926234	0.261402	503	0.044975	0.207258
44	US	RuleBased	135	131	0.970370	0.170195	4	0.029630	0.170195
45	US	RuleBasedY	9957	5395	0.541830	0.498272	1854	0.186201	0.389288
46	US	UserPopQ	1822	1227	0.673436	0.469085	342	0.187706	0.390584
47	US	UserPopY	6933	4836	0.697534	0.459359	1554	0.224145	0.417048
48	US	XGBQ	3114	2547	0.817919	0.385973	312	0.100193	0.300305
49	US	XGBY	3312	2891	0.872886	0.333150	299	0.090278	0.286623

In [107…

```
res = new.groupby(['region','rec_provider']).apply(ci).reset_index()
res
```

Out[107]:

	region	rec_provider	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	BooksQ	([0.0119], [0.0265])	([0.7784], [0.821])	0 0.0146 dtype: float64	0 0.0426 dtype: float64
1	DE	BooksY	([0.003], [0.0129])	([0.9566], [0.9765])	1 0.0099 dtype: float64	1 0.0199 dtype: float64
2	DE	DNNQ	([0.1193], [0.1264])	([0.8351], [0.8431])	2 0.0071 dtype: float64	2 0.008 dtype: float64
3	DE	DNNY	([0.044], [0.0556])	([0.91], [0.9246])	3 0.0116 dtype: float64	3 0.0146 dtype: float64
4	DE	ManualQ	([0.035], [0.0621])	([0.8354], [0.8795])	4 0.0271 dtype: float64	4 0.0441 dtype: float64
5	DE	ManualY	([0.0241], [0.0552])	([0.9448], [0.9759])	5 0.0311 dtype: float64	5 0.0311 dtype: float64
6	DE	RuleBasedY	([0.0921], [0.1223])	([0.6097], [0.6567])	6 0.0302 dtype: float64	6 0.047 dtype: float64
7	DE	UserPopQ	([0.1286], [0.1394])	([0.7241], [0.7382])	7 0.0108 dtype: float64	7 0.0141 dtype: float64
8	DE	UserPopY	([0.1354], [0.1594])	([0.7654], [0.7935])	8 0.024 dtype: float64	8 0.0281 dtype: float64
9	GB	DNNQ	([0.1063], [0.1124])	([0.8709], [0.8774])	9 0.0061 dtype: float64	9 0.0065 dtype: float64
10	GB	DNNY	([0.1041], [0.1186])	([0.8499], [0.866])	10 0.0145 dtype: float64	10 0.0161 dtype: float64
11	GB	ManualQ	([0.0156], [0.028])	([0.959], [0.9742])	11 0.0124 dtype: float64	11 0.0152 dtype: float64
12	GB	ManualY	([0.0601], [0.1062])	([0.8457], [0.9012])	12 0.0461 dtype: float64	12 0.0555 dtype: float64
13	GB	RNNQ	([0.0764], [0.0865])	([0.8868], [0.8982])	13 0.0101 dtype: float64	13 0.0114 dtype: float64
14	GB	RNNY	([0.072], [0.0999])	([0.8555], [0.8887])	14 0.0279 dtype: float64	14 0.0332 dtype: float64
15	GB	RuleBased	([-0.024], [0.074])	([0.926], [1.024])	15 0.098 dtype: float64	15 0.098 dtype: float64
16	GB	RuleBasedY	([0.4867], [0.5323])	([0.3549], [0.3991])	16 0.0456 dtype: float64	16 0.0442 dtype: float64
17	GB	UserPopQ	([0.1647], [0.1827])	([0.6614], [0.6838])	17 0.018 dtype: float64	17 0.0224 dtype: float64
18	GB	UserPopY	([0.2382], [0.2806])	([0.636], [0.6819])	18 0.0424 dtype: float64	18 0.0459 dtype: float64
19	GB	XGBQ	([0.1015], [0.1142])	([0.7926], [0.809])	19 0.0127 dtype: float64	19 0.0164 dtype: float64
20	GB	XGBY	([0.0807], [0.1185])	([0.8403], [0.8838])	20 0.0378 dtype: float64	20 0.0435 dtype: float64

	region	rec_provider	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
21	IN	ManualQ	([0.0194], [0.0275])	([0.9467], [0.9581])	21 0.0081 dtype: float64	21 0.0114 dtype: float64
22	IN	ManualY	([0.0], [0.0])	([0.9117], [0.9728])	22 0.0 dtype: float64	22 0.0611 dtype: float64
23	IN	RuleBased	([-0.1371], [0.4229])	([0.5771], [1.1371])	23 0.56 dtype: float64	23 0.56 dtype: float64
24	IN	RuleBasedY	([0.2196], [0.2335])	([0.62], [0.636])	24 0.0139 dtype: float64	24 0.016 dtype: float64
25	IN	UserPopQ	([0.091], [0.0968])	([0.7914], [0.7995])	25 0.0058 dtype: float64	25 0.0081 dtype: float64
26	IN	UserPopY	([0.1859], [0.1956])	([0.739], [0.7497])	26 0.0097 dtype: float64	26 0.0107 dtype: float64
27	JP	DNNQ	([0.1588], [0.1681])	([0.8136], [0.8233])	27 0.0093 dtype: float64	27 0.0097 dtype: float64
28	JP	DNNY	([0.1282], [0.1501])	([0.7884], [0.8136])	28 0.0219 dtype: float64	28 0.0252 dtype: float64
29	JP	ManualQ	([0.0221], [0.0497])	([0.9156], [0.9524])	29 0.0276 dtype: float64	29 0.0368 dtype: float64
30	JP	ManualY	([0.0503], [0.1577])	([0.4563], [0.6317])	30 0.1074 dtype: float64	30 0.1754 dtype: float64
31	JP	RuleBasedY	([0.1876], [0.2526])	([0.2871], [0.3604])	31 0.065 dtype: float64	31 0.0733 dtype: float64
32	JP	UserPopQ	([0.1489], [0.1649])	([0.6186], [0.6399])	32 0.016 dtype: float64	32 0.0213 dtype: float64
33	JP	UserPopSelectionQ	([0.0988], [0.1238])	([0.7996], [0.8305])	33 0.025 dtype: float64	33 0.0309 dtype: float64
34	JP	UserPopSelectionY	([-0.0457], [0.141])	([0.7038], [1.0105])	34 0.1867 dtype: float64	34 0.3067 dtype: float64
35	JP	UserPopY	([0.1356], [0.1738])	([0.479], [0.5319])	35 0.0382 dtype: float64	35 0.0529 dtype: float64
36	US	BooksQ	([-0.0021], [0.013])	([0.9561], [0.9894])	36 0.0151 dtype: float64	36 0.0333 dtype: float64
37	US	BooksY	([0.0018], [0.0065])	([0.9811], [0.9898])	37 0.0047 dtype: float64	37 0.0087 dtype: float64
38	US	DNNQ	([0.0753], [0.0827])	([0.9017], [0.9096])	38 0.0074 dtype: float64	38 0.0079 dtype: float64
39	US	DNNY	([0.0756], [0.0819])	([0.8963], [0.9033])	39 0.0063 dtype: float64	39 0.007 dtype: float64
40	US	ManualQ	([0.0168], [0.025])	([0.9656], [0.9752])	40 0.0082 dtype: float64	40 0.0096 dtype: float64
41	US	ManualY	([0.0175], [0.05])	([0.9321], [0.9708])	41 0.0325 dtype: float64	41 0.0387 dtype: float64

	region	rec_provider	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
42	US	RNNQ	([0.0595], [0.0693])	([0.9088], [0.92])	42 0.0098 dtype: float64	42 0.0112 dtype: float64
43	US	RNNY	([0.0411], [0.0488])	([0.9214], [0.9311])	43 0.0077 dtype: float64	43 0.0097 dtype: float64
44	US	RuleBased	([0.0009], [0.0583])	([0.9417], [0.9991])	44 0.0574 dtype: float64	44 0.0574 dtype: float64
45	US	RuleBasedY	([0.1786], [0.1938])	([0.532], [0.5516])	45 0.0152 dtype: float64	45 0.0196 dtype: float64
46	US	UserPopQ	([0.1698], [0.2056])	([0.6519], [0.695])	46 0.0358 dtype: float64	46 0.0431 dtype: float64
47	US	UserPopY	([0.2143], [0.234])	([0.6867], [0.7083])	47 0.0197 dtype: float64	47 0.0216 dtype: float64
48	US	XGBQ	([0.0896], [0.1107])	([0.8044], [0.8315])	48 0.0211 dtype: float64	48 0.0271 dtype: float64
49	US	XGBY	([0.0805], [0.1])	([0.8615], [0.8842])	49 0.0195 dtype: float64	49 0.0227 dtype: float64

A20

In [108…
```
d = {'recommendation_type': ['count'],'new_success': ['sum','mean', 'std'], 'sever_inc
new = ad_recs_annotated.groupby('recommendation_type').agg(d).reset_index()
new.columns = ['rec_type', 'n','sum_succ','succ_mean', 'succ_std', 'sum_sev','sev_mean
new
```

Out[108]:

	rec_type	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
0	Qality	236572	196570	0.830910	0.374832	25836	0.109210	0.311903
1	Yield	137216	107636	0.784427	0.411220	18853	0.137397	0.344267

In [109…
```
res = new.groupby('rec_type').apply(ci).reset_index()
res
```

Out[109]:

	rec_type	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	Qality	([0.108], [0.1105])	([0.8294], [0.8324])	0 0.0025 dtype: float64	0 0.003 dtype: float64
1	Yield	([0.1356], [0.1392])	([0.7823], [0.7866])	1 0.0036 dtype: float64	1 0.0043 dtype: float64

A21

In [110…
```
d = {'region': ['count'],'new_success': ['sum','mean', 'std'], 'sever_ind': ['sum','me
new = ad_recs_annotated.groupby(['region','recommendation_type']).agg(d).reset_index()
new.columns = ['region','rec_type', 'n','sum_succ','succ_mean', 'succ_std', 'sum_sev',
new
```

Out[110]:

	region	rec_type	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
0	DE	Qality	50295	40572	0.806681	0.394905	6111	0.121503	0.326714
1	DE	Yield	11882	10052	0.845986	0.360978	952	0.080121	0.271492
2	GB	Qality	69403	59018	0.850367	0.356714	7516	0.108295	0.310755
3	GB	Yield	13581	10491	0.772476	0.419249	2413	0.177675	0.382253
4	IN	Qality	40852	32932	0.806129	0.395334	3635	0.088980	0.284718
5	IN	Yield	42302	30423	0.719186	0.449402	8101	0.191504	0.393489
6	JP	Qality	35469	27588	0.777806	0.415727	5554	0.156587	0.363416
7	JP	Yield	6158	4212	0.683988	0.464955	903	0.146639	0.353774
8	US	Qality	40553	36460	0.899070	0.301239	3020	0.074470	0.262538
9	US	Yield	63293	52458	0.828812	0.376676	6484	0.102444	0.303234

In [111…

```
res = new.groupby(['region','rec_type']).apply(ci).reset_index()
res
```

Out[111]:

	region	rec_type	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	Qality	([0.1186], [0.1244])	([0.8032], [0.8101])	0 0.0058 dtype: float64	0 0.0069 dtype: float64
1	DE	Yield	([0.0752], [0.085])	([0.8395], [0.8525])	1 0.0098 dtype: float64	1 0.013 dtype: float64
2	GB	Qality	([0.106], [0.1106])	([0.8477], [0.853])	2 0.0046 dtype: float64	2 0.0053 dtype: float64
3	GB	Yield	([0.1712], [0.1841])	([0.7654], [0.7795])	3 0.0129 dtype: float64	3 0.0141 dtype: float64
4	IN	Qality	([0.0862], [0.0917])	([0.8023], [0.81])	4 0.0055 dtype: float64	4 0.0077 dtype: float64
5	IN	Yield	([0.1878], [0.1953])	([0.7149], [0.7235])	5 0.0075 dtype: float64	5 0.0086 dtype: float64
6	JP	Qality	([0.1528], [0.1604])	([0.7735], [0.7821])	6 0.0076 dtype: float64	6 0.0086 dtype: float64
7	JP	Yield	([0.1378], [0.1555])	([0.6724], [0.6956])	7 0.0177 dtype: float64	7 0.0232 dtype: float64
8	US	Qality	([0.0719], [0.077])	([0.8961], [0.902])	8 0.0051 dtype: float64	8 0.0059 dtype: float64
9	US	Yield	([0.1001], [0.1048])	([0.8259], [0.8317])	9 0.0047 dtype: float64	9 0.0058 dtype: float64

A22

In [112…

```
d22 = {'region': ['count'],'new_success': ['sum','mean', 'std'], 'sever_ind': ['sum','
new22 = ad_recs_annotated.groupby(['region','week_id']).agg(d22).reset_index()
```

```
new22.columns = ['region', 'week_id', 'n','sum_succ','succ_mean', 'succ_std', 'sum_sev
new22
```

Out[112]:

	region	week_id	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
0	DE	2021-week_33	2395	1930	0.805846	0.395631	277	0.115658	0.319881
1	DE	2021-week_34	2360	1883	0.797881	0.401665	310	0.131356	0.337861
2	DE	2021-week_35	2393	1931	0.806937	0.394785	278	0.116172	0.320498
3	DE	2021-week_36	2401	1909	0.795085	0.403723	280	0.116618	0.321032
4	DE	2021-week_37	2404	1938	0.806156	0.395390	270	0.112313	0.315817
...	...	...	...	...	...	...	...	...	...
124	US	2022-week_03	3979	3313	0.832621	0.373361	390	0.098015	0.297372
125	US	2022-week_04	3999	3321	0.830458	0.375277	437	0.109277	0.312026
126	US	2022-week_05	3998	3342	0.835918	0.370396	408	0.102051	0.302753
127	US	2022-week_06	3992	3354	0.840180	0.366485	416	0.104208	0.305569
128	US	2022-week_52	3998	3342	0.835918	0.370396	437	0.109305	0.312060

129 rows × 9 columns

In [113…

```
res22 = new22.groupby(['region','week_id','succ_mean','sev_mean']).apply(ci).reset_inc
res22.head(1)
```

Out[113]:

	region	week_id	succ_mean	sev_mean	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	2021-week_33	0.805846	0.115658	([0.1028], [0.1285])	([0.79], [0.8217])	0 0.0257 dtype: float64	0 0.0317 dtype: float64

In [114…

```
res22['ci_succ_range'] = res22['ci_succ_range'].astype('float32')
res22['ci_sev_range'] = res22['ci_sev_range'].astype('float32')
res22
```

Out[114]:

	region	week_id	succ_mean	sev_mean	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	2021-week_33	0.805846	0.115658	([0.1028], [0.1285])	([0.79], [0.8217])	0.0257	0.0317
1	DE	2021-week_34	0.797881	0.131356	([0.1177], [0.145])	([0.7817], [0.8141])	0.0273	0.0324
2	DE	2021-week_35	0.806937	0.116172	([0.1033], [0.129])	([0.7911], [0.8228])	0.0257	0.0317
3	DE	2021-week_36	0.795085	0.116618	([0.1038], [0.1295])	([0.7789], [0.8112])	0.0257	0.0323
4	DE	2021-week_37	0.806156	0.112313	([0.0997], [0.1249])	([0.7904], [0.822])	0.0252	0.0316
...	...	...	...	...	...	...	...	...
124	US	2022-week_03	0.832621	0.098015	([0.0888], [0.1073])	([0.821], [0.8442])	0.0185	0.0232
125	US	2022-week_04	0.830458	0.109277	([0.0996], [0.1189])	([0.8188], [0.8421])	0.0193	0.0233
126	US	2022-week_05	0.835918	0.102051	([0.0927], [0.1114])	([0.8244], [0.8474])	0.0187	0.0230
127	US	2022-week_06	0.840180	0.104208	([0.0947], [0.1137])	([0.8288], [0.8515])	0.0190	0.0227
128	US	2022-week_52	0.835918	0.109305	([0.0996], [0.119])	([0.8244], [0.8474])	0.0194	0.0230

129 rows × 8 columns

In [115…

```
fig = px.line(res22, x='week_id',y='succ_mean', error_y='ci_succ_range', color='regior
       title='Weekly success rate per region')
fig.update_layout(xaxis_title='week_id', yaxis_title='Success Rate')
fig.show()
```

A23

In [116…
```
d23 = {'recommendation_type': ['count'],'new_success': ['sum','mean', 'std'], 'sever_i
new23 = ad_recs_annotated.groupby(['region','recommendation_type','week_id']).agg(d23)
new23.columns = ['region','rec_type','week_id', 'n','sum_succ','succ_mean', 'succ_std'
new23.head(1)
```

Out[116]:

	region	rec_type	week_id	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
0	DE	Qality	2021-week_33	1908	1537	0.805556	0.395876	223	0.116876	0.321357

In [117…
```
res23 = new23.groupby(['region','rec_type','week_id','succ_mean','sev_mean']).apply(ci
res23.head(1)
```

Out[117]:

	region	rec_type	week_id	succ_mean	sev_mean	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	Qality	2021-week_33	0.805556	0.116876	([0.1025], [0.1313])	([0.7878], [0.8233])	0 0.0288 dtype: float64	0 0.0355 dtype: float64

In [118…
```
res23['ci_succ_range'] = res23['ci_succ_range'].astype('float32')
res23['ci_sev_range'] = res23['ci_sev_range'].astype('float32')
res23.head(1)
```

Out[118]:

	region	rec_type	week_id	succ_mean	sev_mean	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	Qality	2021-week_33	0.805556	0.116876	([0.1025], [0.1313])	([0.7878], [0.8233])	0.0288	0.0355

In [119…
```
fig = px.line(res23, x='week_id',y='succ_mean', error_y='ci_succ_range', color='regior
       title='Weekly success rate per region and recommendation type')
fig.update_layout(xaxis_title='week_id', yaxis_title='Success Rate')
fig.show()
```

A24

In [120…
```
d24 = {'recommendation_type': ['count'],'new_success': ['sum','mean', 'std'], 'sever_i
new24 = ad_recs_annotated.groupby(['region','rec_provider','week_id']).agg(d24).reset_
new24.columns = ['region','rec_provider','week_id', 'n','sum_succ','succ_mean', 'succ_
new24.head(1)
```

Out[120]:

	region	rec_provider	week_id	n	sum_succ	succ_mean	succ_std	sum_sev	sev_mean	sev_std
0	DE	BooksQ	2021-week_33	68	57	0.838235	0.370973	0	0.0	0.0

In [121…
```
res24 = new24.groupby(['region','rec_provider','week_id','succ_mean','sev_mean']).appl
res24.head(1)
```

Out[121]:

	region	rec_provider	week_id	succ_mean	sev_mean	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	BooksQ	2021-week_33	0.838235	0.0	([0.0], [0.0])	([0.7501], [0.9264])	0 0.0 dtype: float64	0 0.1763 dtype: float64

In [122…
```
res24['ci_succ_range'] = res24['ci_succ_range'].astype('float32')
res24['ci_sev_range'] = res24['ci_sev_range'].astype('float32')
res24.head(1)
```

Out[122]:

	region	rec_provider	week_id	succ_mean	sev_mean	ci sever ind	ci success ind	ci_sev_range	ci_succ_range
0	DE	BooksQ	2021-week_33	0.838235	0.0	([0.0], [0.0])	([0.7501], [0.9264])	0.0	0.1763

In [123…
```
provider_widget = widgets.Dropdown(options=res24['rec_provider'].unique(), descriptior
```

In [124…
```
@interact
def plot(rec_provider = provider_widget):
    selection = res24.query('rec_provider == @rec_provider')
    fig = px.line(selection, x='week_id',y='succ_mean', error_y='ci_succ_range', color
            title='Weekly success rate per region and rec provider')
    fig.update_layout(xaxis_title='week_id', yaxis_title='Success Rate')
    fig.show()
```

```
interactive(children=(Dropdown(description='Select Provider', options=('BooksQ', 'Boo
ksY', 'DNNQ', 'DNNY', 'Ma…
```

A25

In [125…
```
d25 = {'requester': ['count'],'new_success': ['mean']}
max_req = ad_recs_annotated.groupby('requester').agg(d25).reset_index()
max_req.columns = ['requester','count','success_pct']
max_req.sort_values(['success_pct'], ascending=False)
```

Out[125]:

	requester	count	success_pct
18	Marketo Japan	1	1.000000
11	FlagTap	2	1.000000
4	Cue	4	1.000000
6	Derceto	1	1.000000
1	Altammune	5	1.000000
19	Metranome	34	0.970588
24	PageBites	135	0.911111
7	Doctorfun Entertainment, Ltd	204	0.892157
32	Tab Solutions	13294	0.889800
20	Mission Street Manufacturing	186	0.881720
10	Fancy	32346	0.881686
0	Allthetopbananas.com	216	0.865741
31	Sensor Tower	3503	0.862404
25	Pole Star	234	0.854701
35	iDreamsky Technology	11568	0.840249
3	Crescendo Networks	1933	0.838076
34	aPriori Technologies	1192	0.833893
29	SOLOMO365	5895	0.827820
9	Extreme DA	57450	0.823603
14	Glory Medical	125	0.816000
21	MoJoe Brewing Company	31210	0.814995
30	Search Million Culture	86540	0.812699
12	Fry Multimedia	5161	0.810696
13	G-mode	1273	0.809898
28	RelayFoods	78502	0.806196
2	Bizanga	2120	0.800943
33	VarVee	1064	0.793233
5	Cuiker	22	0.772727
15	Joules Clothing	25	0.760000
23	OpenDesks, Inc.	548	0.751825
16	Jun Group	4	0.750000
8	Earth Networks	4	0.750000
17	LocalVox Media	5811	0.739976

	requester	count	success_pct
26	Puentes Company	29427	0.732898
22	Modanisa	3748	0.578975
27	PureSafe water systems	1	0.000000

Conclusion: In the table above we can see that there are requesters with 100% success rate but with very low amount of sales. Therefore the success rate doesn't say much by itself in means of successful pct.

A26

In [126…

```
d26 = {'requester': ['count'],'sever_ind': ['mean']}
max_req_sev = ad_recs_annotated.groupby('requester').agg(d26).reset_index()
max_req_sev.columns = ['requester','count','sever_pct']
max_req_sev.sort_values(['sever_pct'], ascending=False)
```

Out[126]:

	requester	count	sever_pct
27	PureSafe water systems	1	1.000000
22	Modanisa	3748	0.378869
16	Jun Group	4	0.250000
15	Joules Clothing	25	0.240000
14	Glory Medical	125	0.184000
5	Cuiker	22	0.181818
26	Puentes Company	29427	0.175349
17	LocalVox Media	5811	0.160213
23	OpenDesks, Inc.	548	0.153285
2	Bizanga	2120	0.137264
28	RelayFoods	78502	0.126888
29	SOLOMO365	5895	0.122307
21	MoJoe Brewing Company	31210	0.116277
12	Fry Multimedia	5161	0.115869
13	G-mode	1273	0.114690
33	VarVee	1064	0.113722
25	Pole Star	234	0.111111
35	iDreamsky Technology	11568	0.110045
30	Search Million Culture	86540	0.108331
9	Extreme DA	57450	0.104247
34	aPriori Technologies	1192	0.104027
3	Crescendo Networks	1933	0.103983
10	Fancy	32346	0.096086
31	Sensor Tower	3503	0.090779
0	Allthetopbananas.com	216	0.087963
32	Tab Solutions	13294	0.084625
20	Mission Street Manufacturing	186	0.069892
24	PageBites	135	0.059259
7	Doctorfun Entertainment, Ltd	204	0.058824
19	Metranome	34	0.029412
1	Altammune	5	0.000000
11	FlagTap	2	0.000000
8	Earth Networks	4	0.000000

	requester	count	sever_pct
6	Derceto	1	0.000000
4	Cue	4	0.000000
18	Marketo Japan	1	0.000000

Conclusion: In the table above we can see the requester with 100% critical severity rate but with very low amount of sales. Therefore the severity rate doesn't say much by itself.

A27

In [127…

```
d27 = {'new_success': ['mean']}
succ_wow = ad_recs_annotated.groupby('week_id').agg(d27).reset_index()
succ_wow.columns = ['week_id','success_pct']
succ_wow
```

Out[127]:

	week_id	success_pct
0	2021-week_33	0.809000
1	2021-week_34	0.808294
2	2021-week_35	0.816044
3	2021-week_36	0.814107
4	2021-week_37	0.813230
5	2021-week_38	0.805255
6	2021-week_39	0.807228
7	2021-week_40	0.806400
8	2021-week_41	0.816273
9	2021-week_42	0.812396
10	2021-week_43	0.804134
11	2021-week_44	0.806350
12	2021-week_45	0.831163
13	2021-week_46	0.830871
14	2021-week_47	0.827941
15	2021-week_48	0.817204
16	2021-week_49	0.821162
17	2021-week_50	0.819425
18	2021-week_51	0.801562
19	2022-week_01	0.809663
20	2022-week_02	0.812868
21	2022-week_03	0.812321
22	2022-week_04	0.814088
23	2022-week_05	0.814162
24	2022-week_06	0.817152
25	2022-week_52	0.810609

In [128…

```
fig27 = px.line(succ_wow, x='week_id', y=['success_pct'],title = 'Success rate over ti
fig27.update_layout(xaxis_title='week_id', yaxis_title='Success Rate')
fig27.show()
```

A28

In [129…

```
d28 = {'sever_ind': ['mean']}
sev_wow = ad_recs_annotated.groupby(['week_id']).agg(d28).reset_index()
sev_wow.columns = ['week_id','sever_pct']
sev_wow
```

Out[129]:

	week_id	sever_pct
0	2021-week_33	0.124617
1	2021-week_34	0.123791
2	2021-week_35	0.114451
3	2021-week_36	0.116261
4	2021-week_37	0.120345
5	2021-week_38	0.124277
6	2021-week_39	0.128815
7	2021-week_40	0.128002
8	2021-week_41	0.122601
9	2021-week_42	0.123122
10	2021-week_43	0.129533
11	2021-week_44	0.125813
12	2021-week_45	0.110457
13	2021-week_46	0.112125
14	2021-week_47	0.107277
15	2021-week_48	0.118925
16	2021-week_49	0.111669
17	2021-week_50	0.119828
18	2021-week_51	0.122765
19	2022-week_01	0.119708
20	2022-week_02	0.116065
21	2022-week_03	0.114409
22	2022-week_04	0.117770
23	2022-week_05	0.117881
24	2022-week_06	0.116825
25	2022-week_52	0.121343

In [130…

```
fig28 = px.line(sev_wow, x='week_id', y='sever_pct', title = 'Sever Rate rate over tim
fig28.update_layout(xaxis_title='week_id', yaxis_title='Sever Rate')
fig28.show()
```

A29

In [131…

```
d29 = {'requester': ['count']}
req_no = ad_recs_annotated.groupby(['region','week_id']).agg(d29).reset_index()
req_no.columns = ['region','week_id','req_no']
req_no
```

Out[131]:

	region	week_id	req_no
0	DE	2021-week_33	2395
1	DE	2021-week_34	2360
2	DE	2021-week_35	2393
3	DE	2021-week_36	2401
4	DE	2021-week_37	2404
...	...	...	...
124	US	2022-week_03	3979
125	US	2022-week_04	3999
126	US	2022-week_05	3998
127	US	2022-week_06	3992
128	US	2022-week_52	3998

129 rows × 3 columns

In [132…

```
req_no_plot = px.bar(req_no, x="week_id", y="req_no", color='region')
req_no_plot.show()
```

A31

In [133…
```
d31 = {'week_id': ['count']}
req_prop = ad_recs_annotated.groupby(['week_id','requester']).agg(d31).reset_index()
req_prop.columns = ['week_id','requester','req_no']
req_prop
```

Out[133]:

	week_id	requester	req_no
0	2021-week_33	Allthetopbananas.com	7
1	2021-week_33	Bizanga	103
2	2021-week_33	Crescendo Networks	44
3	2021-week_33	Cuiker	1
4	2021-week_33	Extreme DA	505
…	…	…	…
671	2022-week_52	Sensor Tower	67
672	2022-week_52	Tab Solutions	223
673	2022-week_52	VarVee	6
674	2022-week_52	aPriori Technologies	61
675	2022-week_52	iDreamsky Technology	377

676 rows × 3 columns

In [134…
```
req_prop['req_prop'] =  req_prop['req_no'] / req_prop.groupby('week_id')['req_no'].tra
req_prop
```

Out[134]:

	week_id	requester	req_no	req_prop
0	2021-week_33	Allthetopbananas.com	7	0.000488
1	2021-week_33	Bizanga	103	0.007175
2	2021-week_33	Crescendo Networks	44	0.003065
3	2021-week_33	Cuiker	1	0.000070
4	2021-week_33	Extreme DA	505	0.035177
…	…	…	…	…
671	2022-week_52	Sensor Tower	67	0.004206
672	2022-week_52	Tab Solutions	223	0.013999
673	2022-week_52	VarVee	6	0.000377
674	2022-week_52	aPriori Technologies	61	0.003829
675	2022-week_52	iDreamsky Technology	377	0.023666

676 rows × 4 columns

In [135…
```
req_pro_plot = px.bar(req_prop, x="week_id", y="req_prop", color='requester')
req_pro_plot.show()
```

Merges and joins

A1

In [136…
```
import os
weekly_files = os.listdir('data/weekly/')
sorted(weekly_files)[:10]
```

Out[136]:
```
['.ipynb_checkpoints',
 '2021_33_DE_Qality_annotation_result.csv',
 '2021_33_DE_Yield_annotation_result.csv',
 '2021_33_DE__Sample.csv',
 '2021_33_GB_Qality_annotation_result.csv',
 '2021_33_GB_Yield_annotation_result.csv',
 '2021_33_GB__Sample.csv',
 '2021_33_IN_Qality_annotation_result.csv',
 '2021_33_IN_Yield_annotation_result.csv',
 '2021_33_IN__Sample.csv']
```

In [137…
```
len(weekly_files)
```

Out[137]:
```
389
```

In [138…
```
files_df['key'] = files_df.file_name.apply(lambda x: x[:11])
files_df.head(1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [138], in <cell line: 1>()
----> 1 files_df['key'] = files_df.file_name.apply(lambda x: x[:11])
      2 files_df.head(1)

NameError: name 'files_df' is not defined
```

In [139…
```
list_key = files_df['key'].tolist()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [139], in <cell line: 1>()
----> 1 list_key = files_df['key'].tolist()

NameError: name 'files_df' is not defined
```

In [141…
```
temp = []

# removing duplicates
for element in list_key:
    if(element not in temp):
        temp.append(element)

# Assigning the temporary list to the main list
list_key = temp
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [141], in <cell line: 4>()
      1 temp = []
      3 # removing duplicates
----> 4 for element in list_key:
      5     if(element not in temp):
      6         temp.append(element)

NameError: name 'list_key' is not defined
```

NOTE - The following code works well when all the set of files exists (sample/quality/yield). **I** received the following error:

FileNotFoundError: [Errno 2] No such file or directory: 'data/weekly/2021_51_JP_Qality_annotation_result.csv'

In [140…
```
final_file = pd.DataFrame([])
for i in list_key:
    sample_file = pd.read_csv('data/weekly/' + i + '_Sample.csv')
    quality_file = pd.read_csv('data/weekly/' + i + 'Qality_annotation_result.csv')
    quality_file['recommendation_type'] = 'Qality'
    yield_file = pd.read_csv('data/weekly/' + i + 'Yield_annotation_result.csv')
    yield_file['recommendation_type'] = 'Yield'
    rec_type_df = quality_file.append(yield_file)
    final_file = pd.merge(sample_file, rec_type_df, on=['region','ad_id','recommendati
return final_file
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [140], in <cell line: 2>()
      1 final_file = pd.DataFrame([])
----> 2 for i in list_key:
      3     sample_file = pd.read_csv('data/weekly/' + i + '_Sample.csv')
      4     quality_file = pd.read_csv('data/weekly/' + i + 'Qality_annotation_resul
t.csv')

NameError: name 'list_key' is not defined
```

Visualizations

A1

See answer 22 in chapter Analysis.

A2

See answer 31 in chapter Analysis.