progect description

This project is dedicated to **hypotheses prioritizing and A B testing**. I need to:

- prioritize hypotheses that may help boost revenue,
- launch an A/B test,
- analyze the results

We have 3 datasets: 1 for hypotheses Prioritization and 2 for A B testing:

Hypotheses Prioritization:

hypotheses dataset contains a number of hypotheses with brief descriptions and scaled (0-10) qualities like:

- Reach
- Impact
- Confidence
- Effort they are to be compared with the help of ICE/RICE Prioritization Types (STEP 2)

A B testing:

orders dataset contains information on every orders and on users who made it:

- transactionId — order identifier
- visitorId — identifier of the user who placed the order
- date — of the order
- revenue — from the order
- group — the A/B test group that the user belongs to

visits dataset contains information on the number of visits on the date specified in the A/B test group specified:

- date — of the order
- group — A/B test group
- visits — number of visits

Table of Contents

Step 1. Downloading the data and its preprocessing

We will use 8 libraries:

- pandas: for data processing
- numpy, math: for calculations
- matplotlib, seaborn, plotly express: for data visualisation
- datetime: for working with data
- scipy for hypotheses testing
- sys, warnings: for not showing the warnings

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from datetime import timedelta
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import math as mth
from scipy import stats as st
import sys
import warnings
if not sys.warnoptions:
       warnings.simplefilter("ignore")

pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)
```

In [2]:
```
try:
    hypotheses = pd.read_csv('https://code.s3.yandex.net//datasets/hypotheses_us.csv', sep=';',
                             dtype={'Hypothesis': 'str', 'Reach': 'uint8',
                                    'Impact': 'uint8', 'Confidence': 'uint8', 'Effort': 'uint8',
})
    orders = pd.read_csv('https://code.s3.yandex.net//datasets/orders_us.csv',
                         parse_dates=['date'])
    visits = pd.read_csv('https://code.s3.yandex.net//datasets/visits_us.csv',
                         parse_dates=['date'])
    print('Data extracted successfully')
except FileNotFoundError:
    print('Ooops, the dateset not found.')

except pd.errors.EmptyDataError:
    print('Ooops, the dataset is empty.')
```

```
Data extracted successfully
```

In [3]:
```
datasets = ['hypotheses', 'orders', 'visits']

for dataset in datasets:

    print('The ' + dataset + ' dataset:')
    print(vars()[dataset].info(memory_usage='deep'))
    print(vars()[dataset].describe())
    if 'date' in (vars()[dataset].columns):
        print(
            f'The dataset "{dataset}" contains information on the period from {vars()[dataset].d
ate.min()} to {vars()[dataset].date.max()}: {int((orders.date.max() - orders.date.min()) /  np.t
imedelta64(1, "D")) + 1} days')
    print('Number of duplicated entities: ',
          vars()[dataset].duplicated().sum())
    vars()[dataset].columns = vars()[dataset].columns.str.lower()

    print('\n\n')
```

```
The hypotheses dataset:
```

```
The hypotheses dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Hypothesis  9 non-null      object
 1   Reach       9 non-null      uint8 
 2   Impact      9 non-null      uint8 
 3   Confidence  9 non-null      uint8 
 4   Effort      9 non-null      uint8 
dtypes: object(1), uint8(4)
memory usage: 1.4 KB
None
           Reach     Impact  Confidence     Effort
count   9.000000   9.000000    9.000000   9.000000
mean    4.777778   4.777778    5.555556   4.888889
std     3.153481   3.192874    3.045944   2.803767
min     1.000000   1.000000    1.000000   1.000000
25%     3.000000   3.000000    3.000000   3.000000
50%     3.000000   3.000000    7.000000   5.000000
75%     8.000000   7.000000    8.000000   6.000000
max    10.000000  10.000000    9.000000  10.000000
Number of duplicated entities:  0



The orders dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1197 entries, 0 to 1196
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   transactionId  1197 non-null   int64         
 1   visitorId      1197 non-null   int64         
 2   date           1197 non-null   datetime64[ns]
 3   revenue        1197 non-null   float64       
 4   group          1197 non-null   object        
dtypes: datetime64[ns](1), float64(1), int64(2), object(1)
memory usage: 105.3 KB
None
       transactionId     visitorId       revenue
count   1.197000e+03  1.197000e+03   1197.000000
mean    2.155621e+09  2.165960e+09    131.491646
std     1.229085e+09  1.236014e+09    603.004729
min     1.062393e+06  5.114589e+06      5.000000
25%     1.166776e+09  1.111826e+09     20.800000
50%     2.145194e+09  2.217985e+09     50.200000
75%     3.237740e+09  3.177606e+09    130.400000
max     4.293856e+09  4.283872e+09  19920.400000
The dataset "orders" contains information on the period from 2019-08-01 00:00:00 to 2019-08-31 0
0:00:00: 31 days
Number of duplicated entities:  0



The visits dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62 entries, 0 to 61
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   date    62 non-null     datetime64[ns]
 1   group   62 non-null     object        
 2   visits  62 non-null     int64         
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 4.6 KB
None
           visits
count   62.000000
mean   607.290323
std    114.400560
min    361.000000
25%    534.000000
50%    624.500000
75%    710.500000
max    770.000000
The dataset "visits" contains information on the period from 2019-08-01 00:00:00 to 2019-08-31 0
0:00:00: 31 days
Number of duplicated entities:  0
```

downcasting

some downcasting to save space

In [4]:
```
orders['transactionid'] = pd.to_numeric(orders['transactionid'], downcast='unsigned')
orders['visitorid'] = pd.to_numeric(orders['visitorid'], downcast='unsigned')
orders['revenue'] = pd.to_numeric(orders['revenue'], downcast='float')

visits['visits']= pd.to_numeric(visits['visits'], downcast='unsigned')
```

comparing

number of orders made

In [5]:
```
successes = orders.group.value_counts()
successes
```

Out[5]:
```
B    640
A    557
Name: group, dtype: int64
```

number of visits made

In [6]:
```
trials = visits.groupby('group')['visits'].sum()
trials
```

Out[6]:
```
group
A    18736
B    18916
Name: visits, dtype: uint16
```

difference

In [7]:
```
(successes/trials)[1] - (successes/trials)[0]
```

Out[7]:
```
0.004104927280643319
```

Double-group participants

Let's find whether we have users who belong to A and B at the same time and get rid of them

In [8]:
```
orders_count = orders.groupby(
    'visitorid')['group'].value_counts().unstack().fillna(0)
spolied_uses_groupping = orders_count[np.logical_not(
    np.logical_or(orders_count['A'] == 0.0, orders_count['B'] == 0.0))]
```

We found users who belong both to A and B and we better get rid of them for keeping the data clean

In [9]:
```
bad_orders = orders[orders.visitorid.isin(spolied_uses_groupping.index)] # orders of spoiled use
rs
orders = orders[np.logical_not(orders.visitorid.isin(spolied_uses_groupping.index))] # clear ord
ers data
```

In [10]:
```
bad_orders_dates = bad_orders.groupby(['group', 'date']).visitorid.nunique().reset_index()
bad_orders_dates.columns=['group','date','bad_orders_amount']
```

Lets also substract the amount of spoiled orders from the visits table

In [11]:
```
visits = visits.merge(bad_orders_dates, on =['group', 'date'], how='left').fillna(0)
visits['visits'] = visits['visits'] - visits['bad_orders_amount']
visits = visits.drop('bad_orders_amount', axis=1)

orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1016 entries, 0 to 1196
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   transactionid  1016 non-null   uint32        
 1   visitorid      1016 non-null   uint32        
 2   date           1016 non-null   datetime64[ns]
 3   revenue        1016 non-null   float32
```

```
 4   group           1016 non-null   object
dtypes: datetime64[ns](1), float32(1), object(1), uint32(2)
memory usage: 35.7+ KB
```

In [62]:
```
print(f'We threw away {len(spolied_uses_groupping)} users that belong both to A and B made {len(
bad_orders)} orders')
```

```
We threw away 58 users that belong both to A and B made 181 orders
```

Conclusion

The datasets contain information on monthly visits ond orders on the site.
There are users who have been mistakenly assigned to both groups A and B groups. I got rid of them losing 181 rows of orders data from 58 users. Also I substracted those wrong users from the visits table. Now the data is clear, no duplicates found.

hypotheses:
We have 9 hypotheses to test with nice and understandable ratings.
A B:
Orders and visits datasets are prepared for A/B test analyse due to source separations they already have. It's just A/B test, not muliple test. There are both a little more visitors and a little more orders from group B. From the first glance, ratio orders per visits for B is ~0.4% bigger, that's not much, certainly not for this volumes of sales. Further information is yet to discover.

Step 2. Part 1.Prioritizing Hypotheses

ICE framework

In [13]:
```
hypotheses['ICE'] = round(hypotheses['impact'] *
                          hypotheses['confidence'] / hypotheses['effort'], 2)
hypotheses['ICE_share'] = round(
    hypotheses['ICE'] / hypotheses['ICE'].max() * 100, 2)
ICE_hypotheses = hypotheses[['hypothesis', 'ICE', 'ICE_share']].sort_values(
    by='ICE', ascending=False)
ICE_hypotheses
```

Out[13]:

	hypothesis	ICE	ICE_share
8	Launch a promotion that gives users discounts ...	16.20	100.00
0	Add two new channels for attracting traffic. T...	13.33	82.28
7	Add a subscription form to all the main pages....	11.20	69.14
6	Show banners with current offers and sales on ...	8.00	49.38
2	Add product recommendation blocks to the store...	7.00	43.21
1	Launch your own delivery service. This will sh...	2.00	12.35
5	Add a customer review page. This will increase...	1.33	8.21
3	Change the category structure. This will incre...	1.12	6.91
4	Change the background color on the main page. ...	1.00	6.17

RICE framework

In [14]:
```
hypotheses['RICE'] = round(hypotheses['reach'] * hypotheses['impact']
                           * hypotheses['confidence'] / hypotheses['effort'], 2)
hypotheses['RICE_share'] = round(
    hypotheses['RICE'] / hypotheses['RICE'].max() * 100, 2)
RICE_hypotheses = hypotheses[['hypothesis', 'RICE', 'RICE_share']].sort_values(
    by='RICE', ascending=False)
RICE_hypotheses
```

Out[14]:

	hypothesis	RICE	RICE_share
2	Add product recommendation blocks to the store...	56.0	100.00
0	Add two new channels for attracting traffic. T...	40.0	71.43
6	Show banners with current offers and sales on ...	40.0	71.43
8	Launch a promotion that gives users discounts ...	16.2	28.93
7	Add a subscription form to all the main pages....	9.6	17.14
3	Change the category structure. This will incre...	9.0	16.07
1	Launch your own delivery service. This will sh...	4.0	7.14

5	Add a customer review page. This will increase...	4.0	7.14
4	Change the background color on the main page. ...	3.0	5.36

Prioritization changes

Lets see whether 5 lead hypotheses for ICE and RICE are the same:

In [15]:
```
top_5 = RICE_hypotheses[['hypothesis','RICE']][:5].merge(ICE_hypotheses[
                                    'ICE'][:5], how='inner', left_index=True, right_index=True)
top_5
```

Out[15]:

	hypothesis	RICE	ICE
2	Add product recommendation blocks to the store...	56.0	7.00
0	Add two new channels for attracting traffic. T...	40.0	13.33
6	Show banners with current offers and sales on ...	40.0	8.00
8	Launch a promotion that gives users discounts ...	16.2	16.20
7	Add a subscription form to all the main pages....	9.6	11.20

They are. How about top 3 leaders?

In [16]:
```
top_3 = RICE_hypotheses[['hypothesis','RICE']][:3].merge(ICE_hypotheses[
                                    'ICE'][:3], how='inner', left_index=True, right_index=True)
top_3
```

Out[16]:

	hypothesis	RICE	ICE
0	Add two new channels for attracting traffic. T...	40.0	13.33

In [17]:
```
top_3.hypothesis[0]
```

Out[17]:
```
'Add two new channels for attracting traffic. This will bring 30% more users'
```

Conclusion:

In [18]:
```
print(f'We see here that ICE and RICE approach bring us same top 5 leaders, #{list(top_5.index)}
, but they differ in the most promising hypoteses. For ICE it is: \n"{ICE_hypotheses.reset_index
().hypothesis[0]}", \nand for RICE: \n"{RICE_hypotheses.reset_index().hypothesis[0]}". \nNever t
he less both of them consider hypotesis "{top_3.hypothesis[0]}" to be among top 3, but it is sti
ll not the favorite one. \nImportant is that we can see a bigger difference in share: RICE leade
r stands {round(RICE_hypotheses.reset_index().RICE_share[0]-RICE_hypotheses.reset_index().RICE_s
hare[1],2)}% aside from the closes hypotheis, meanwhile ICE have smaller difference of {round(IC
E_hypotheses.reset_index().ICE_share[0]-ICE_hypotheses.reset_index().ICE_share[1],2)}%. Since we
really should pay attention to the amount of users our hypothesis can affect and since the perce
ntage margin for RICE is bigger, I would go with its leader hypothesis: \n"{RICE_hypotheses.rese
t_index().hypothesis[0]}"')
```

```
We see here that ICE and RICE approach bring us same top 5 leaders, #[2, 0, 6, 8, 7], but they d
iffer in the most promising hypoteses. For ICE it is:
"Launch a promotion that gives users discounts on their birthdays",
and for RICE:
"Add product recommendation blocks to the store's site. This will increase conversion and averag
e purchase size".
Never the less both of them consider hypotesis "Add two new channels for attracting traffic. Thi
s will bring 30% more users" to be among top 3, but it is still not the favorite one.
Important is that we can see a bigger difference in share: RICE leader stands 28.57% aside from
the closes hypotheis, meanwhile ICE have smaller difference of 17.72%. Since we really should pa
y attention to the amount of users our hypothesis can affect and since the percentage margin for
RICE is bigger, I would go with its leader hypothesis:
"Add product recommendation blocks to the store's site. This will increase conversion and averag
e purchase size"
```

ICE/RICE comparison plot

In [19]:
```
ICE_RICE_merged = ICE_hypotheses[['hypothesis', 'ICE']].merge(
    RICE_hypotheses['RICE'], left_index=True, right_index=True)
```

In [20]:
```
fig = go.Figure(data=[
    go.Bar(name='ICE', x=ICE_RICE_merged.index, y=ICE_RICE_merged.ICE),
    go.Bar(name='RICE', x=ICE_RICE_merged.index, y=ICE_RICE_merged.RICE)
])
fig.update_layout(title_text='ICE/RICE ratio')

fig.show()
```

```
plt.show()
```

Step 3. Part 2. A/B Test Analysis

Cumulative revenue by group

Let's compare the revenue, one of the most speaking for itself indicator of success

In [21]:
```
datesGroups = orders[['date', 'group']].drop_duplicates()
```

In [22]:
```
ordersAggregated = datesGroups.apply(lambda x: orders[np.logical_and(orders['date'] <= x['date'
], orders['group'] == x['group'])].agg(
    {'date': 'max', 'group': 'max', 'transactionid': pd.Series.nunique, 'visitorid': pd.Series.n
unique, 'revenue': 'sum'}), axis=1).sort_values(by=['date', 'group'])
```

In [23]:
```
visitorsAggregated = datesGroups.apply(lambda x: visits[np.logical_and(visits['date'] <= x['dat
e'], visits['group'] == x['group'])].agg({'date' : 'max', 'group' : 'max', 'visits' : 'sum'}), a
xis=1).sort_values(by=['date','group'])
```

In [24]:
```
cumulativeData = ordersAggregated.merge(visitorsAggregated, left_on=[
                                        'date', 'group'], right_on=['date', 'group'])
cumulativeData.columns = ['date', 'group',
                          'orders', 'buyers', 'revenue', 'visitors']
```

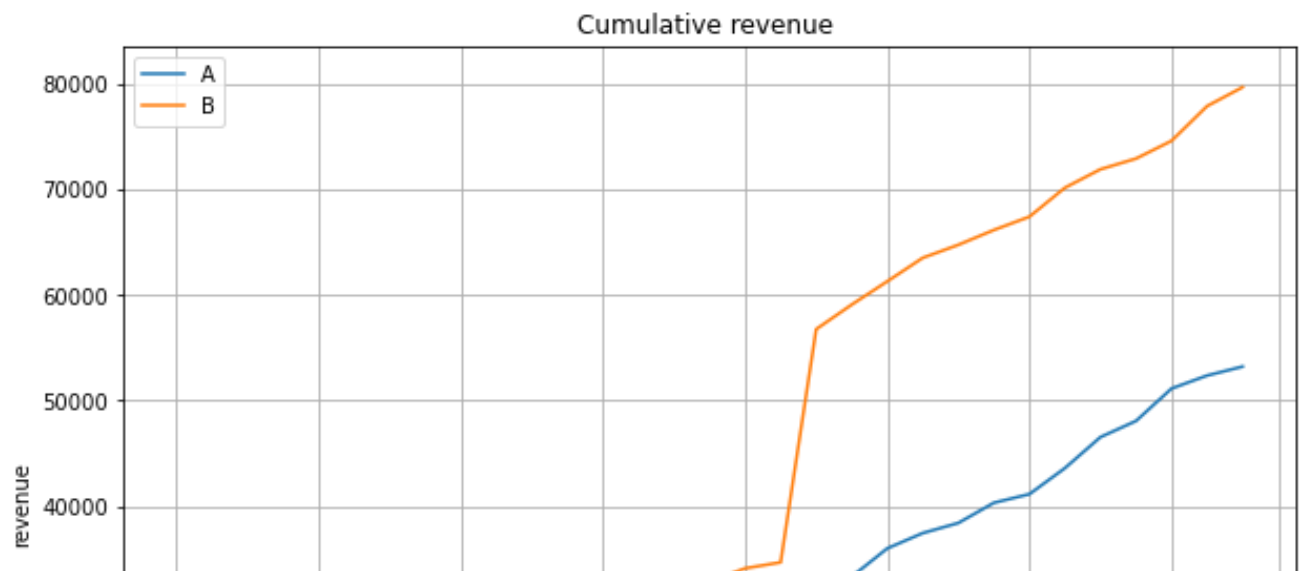
In [25]:
```
cumulativeData['average_order'] = cumulativeData['revenue'] / \
    cumulativeData['orders']
cumulativeData.head(10)
```

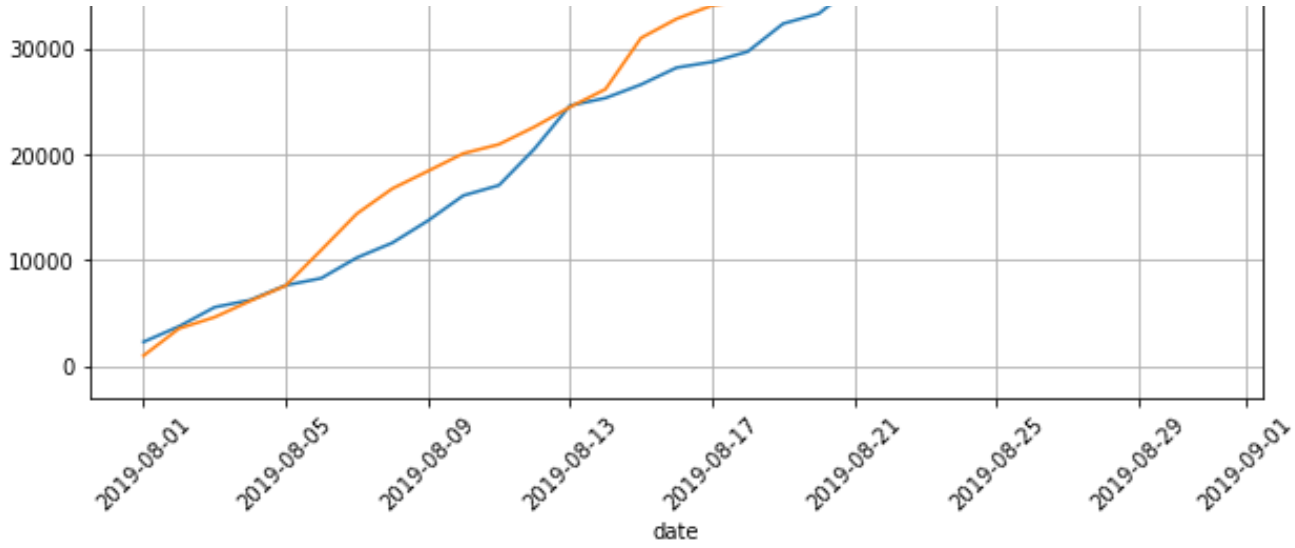
Out[25]:

	date	group	orders	buyers	revenue	visitors	average_order
0	2019-08-01	A	23	19	2266.599854	718.0	98.547820
1	2019-08-01	B	17	17	967.199951	710.0	56.894115
2	2019-08-02	A	42	36	3734.899902	1336.0	88.926188
3	2019-08-02	B	40	39	3535.300293	1290.0	88.382507
4	2019-08-03	A	66	60	5550.100098	1843.0	84.092426
5	2019-08-03	B	54	53	4606.899902	1797.0	85.312961
6	2019-08-04	A	77	71	6225.600098	2556.0	80.851949
7	2019-08-04	B	68	66	6138.500000	2564.0	90.272059
8	2019-08-05	A	99	92	7623.600098	3309.0	77.006062
9	2019-08-05	B	89	87	7587.800293	3269.0	85.256183

In [26]:
```
cumulativeRevenueA = cumulativeData[cumulativeData['group'] == 'A'][[
    'date', 'revenue', 'average_order']]
cumulativeRevenueB = cumulativeData[cumulativeData['group'] == 'B'][[
    'date', 'revenue', 'average_order']]
```

In [27]:
```
plt.figure(figsize=(10, 8))
plt.title('Cumulative revenue ')
plt.grid()
plt.plot(cumulativeRevenueA['date'], cumulativeRevenueA['revenue'], label='A')
plt.plot(cumulativeRevenueB['date'], cumulativeRevenueB['revenue'], label='B')
plt.xticks(rotation=45)
plt.xlabel('date')
plt.ylabel('revenue')
plt.legend();
```





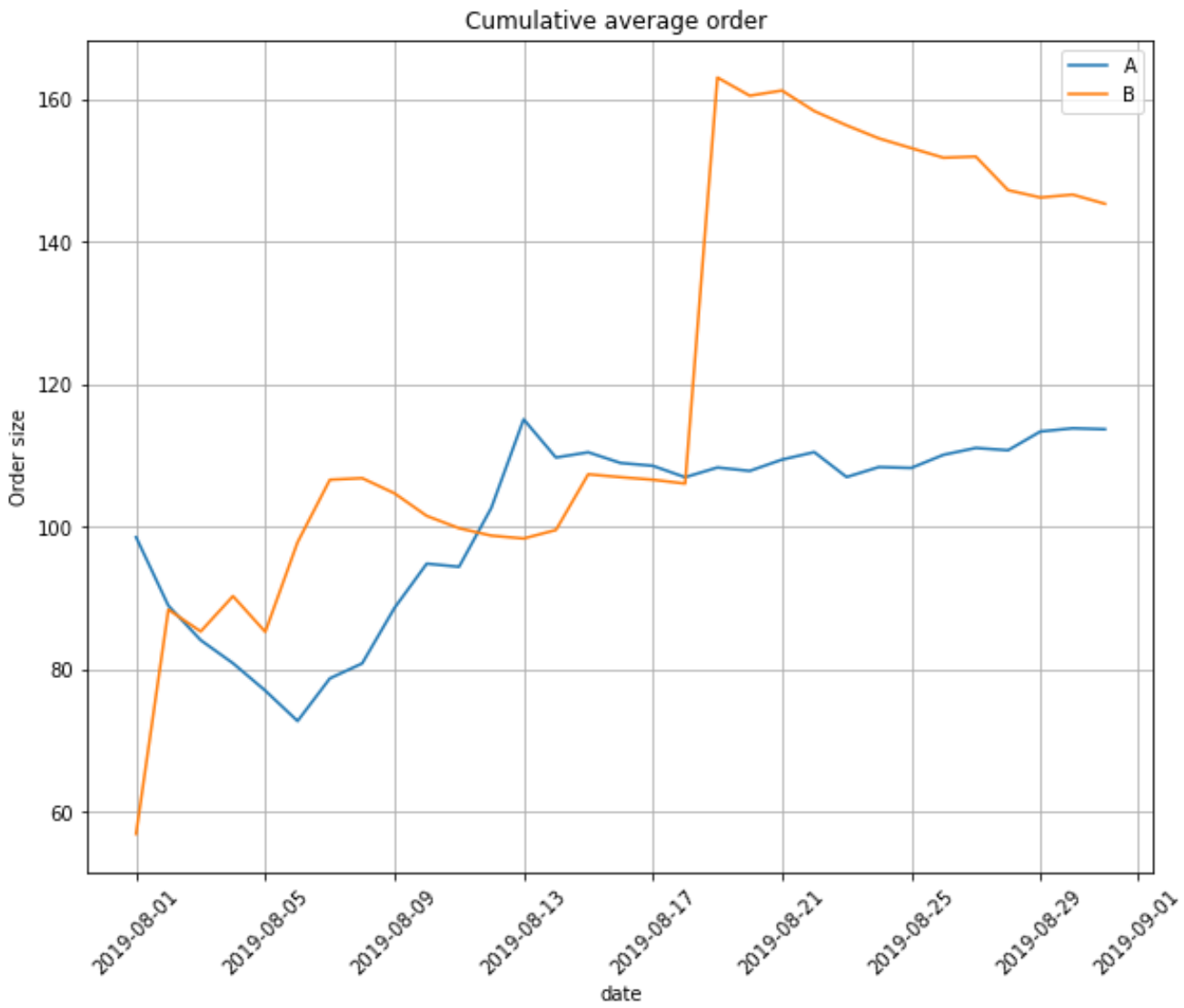
Conclusions and conjectures

The B group brought 26439 more revenue than A group (A: 53212, B: 79651), leading from the second day of test till the end, with a great surge on the 2019-08-18, while group A kept the same pace, stably gaining revenue. Probably the B group introduced a different marketing methods or announced a very popular item that caught users attention or a nice discount on that day and that caused huge order flow, but that was a 1-day event, because later on B continued with the same pace. Would be interesting to find out what happened there. Shortly: the revenue margin happened not die to "bad" A group visitors flow, but due to something great in B group on the 2019-08-18 and general stronger B pace.

Cumulative average order size by group

Let's compare the average order situration

In [28]:
```
plt.figure(figsize=(10, 8))
plt.grid()
plt.plot(cumulativeRevenueA['date'],
         cumulativeRevenueA['average_order'], label='A')
plt.plot(cumulativeRevenueB['date'],
         cumulativeRevenueB['average_order'], label='B')
plt.xticks(rotation=45)
plt.title('Cumulative average order ')
plt.xlabel('date')
plt.ylabel('Order size')
plt.legend();
```



Conclusions and conjectures¶

Once again we see that group B leads. A and B met on the 2019-08-13 but due to the mentioned above B orders surge, groups separated and A ended up with 113 cumulative order while B with 145: the difference is 32. Looks like a D is definite leader, but maybe this happened due to huge 2019-08-18 outlier

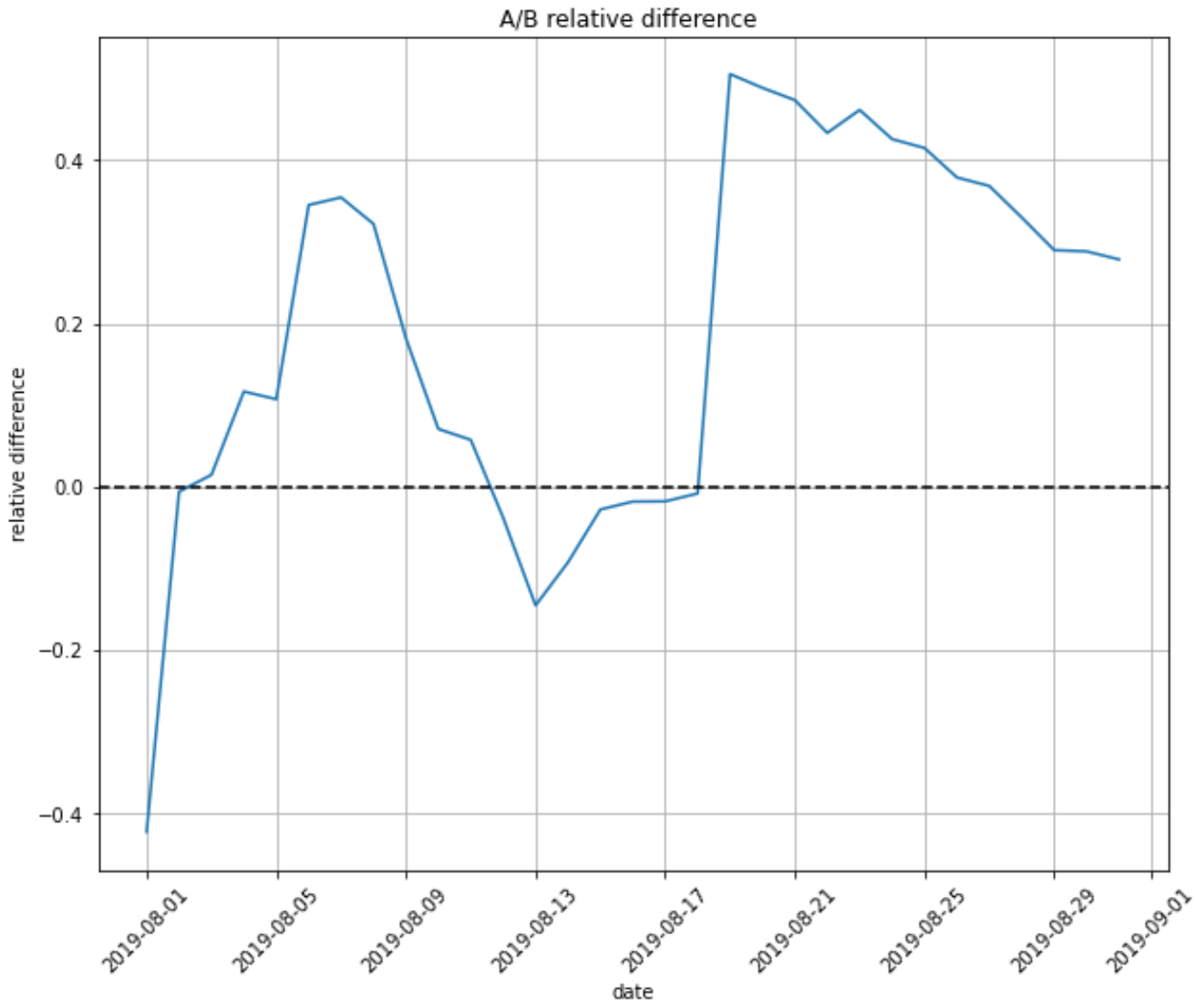
huge 2019-08-18 outlier.

Relative difference graph for the average purchase sizes

In [29]:
```
mergedCumulativeRevenue = cumulativeRevenueA.merge(
    cumulativeRevenueB, left_on='date', right_on='date', how='left', suffixes=['A', 'B'])
plt.figure(figsize=(10, 8))
plt.grid()

plt.plot(mergedCumulativeRevenue['date'], (
    mergedCumulativeRevenue['average_orderB']/mergedCumulativeRevenue['average_orderA'])-1)

plt.xticks(rotation=45)
plt.title('A/B relative difference ')
plt.xlabel('date')
plt.ylabel('relative difference')
plt.axhline(y=0, color='black', linestyle='--');
```



Conclusions and conjectures¶

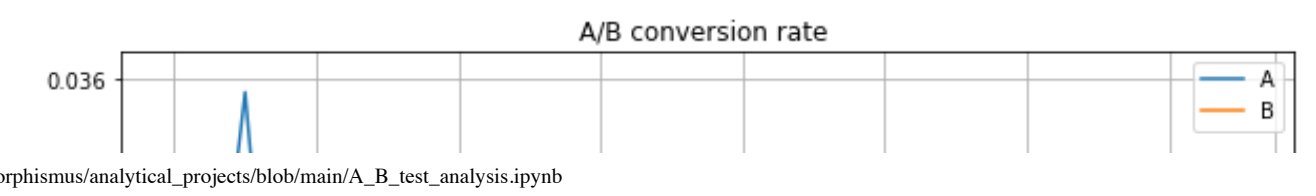
We can see that, starting low and finishing top, B group have interesting fluctuations. It has clear risings: on 2019-08-01, on 2019-08-05, on 2019-08-18, the date, that stands out again. It falls on 2019-08-08 and from and from 2019-08-11 to 2019-08-13, and from 2019-08-19. We see that B is nearly 27% more "successful" than A: their difference grow to 50%, but falls to 27% at the end. Some of the risings seem natural, but the 2019-08-17 looks like it is due to the outliers.

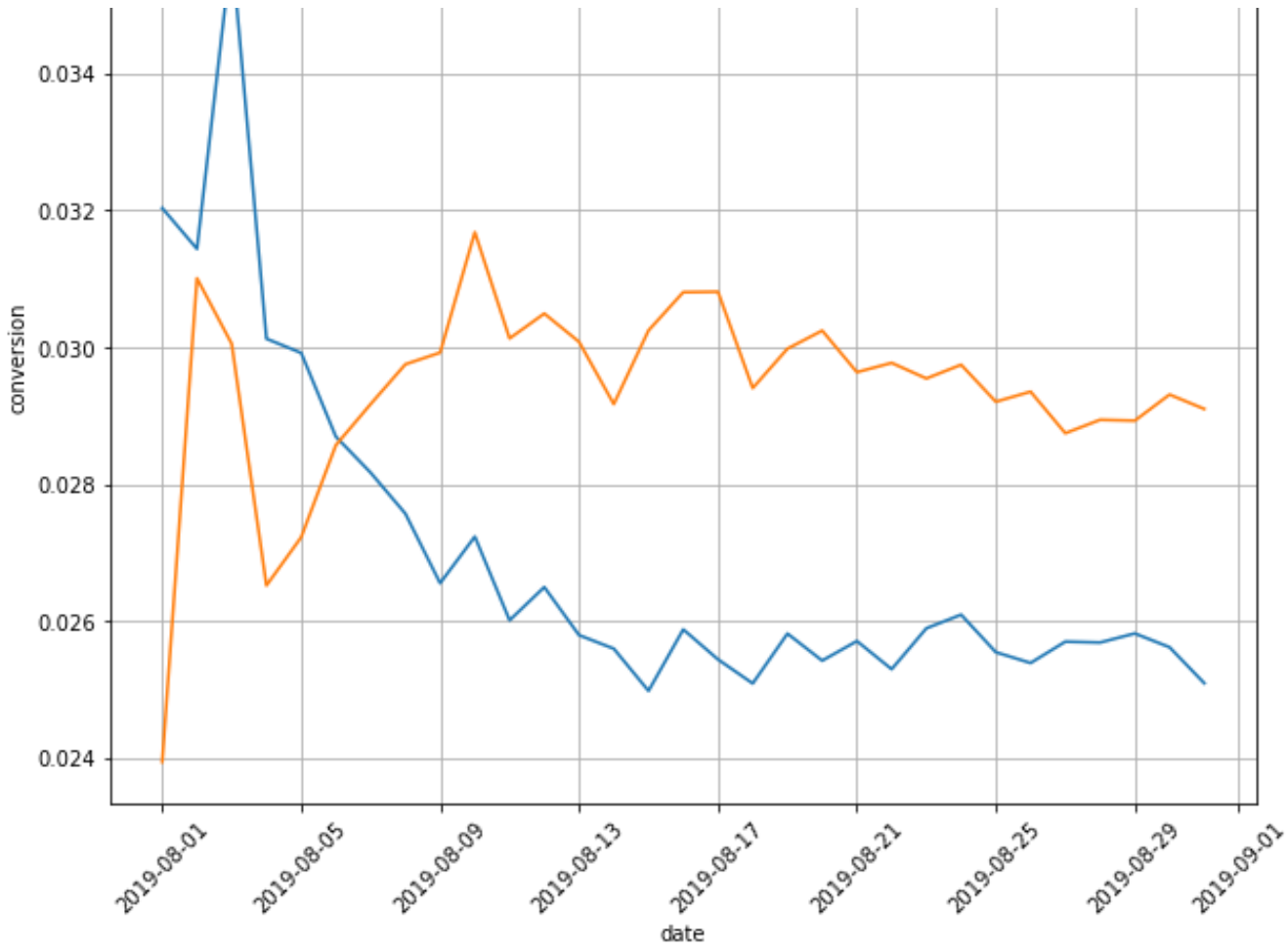
Each group's conversion rate daily

In [30]:
```
cumulativeData['conversion'] = cumulativeData['orders'] / \
    cumulativeData['visitors']
plt.figure(figsize=(10, 8))
plt.grid()

cumulativeDataA = cumulativeData[cumulativeData['group'] == 'A']
cumulativeDataB = cumulativeData[cumulativeData['group'] == 'B']

plt.plot(cumulativeDataA['date'], cumulativeDataA['conversion'], label='A')
plt.plot(cumulativeDataB['date'], cumulativeDataB['conversion'], label='B')
plt.xticks(rotation=45)
plt.title('A/B conversion rate ')
plt.xlabel('date')
plt.ylabel('conversion')
plt.legend();
```





In [31]:
```
B_conversion = pd.Series(cumulativeDataB.conversion).reset_index()
A_conversion = pd.Series(cumulativeDataA.conversion).reset_index()

print('mean conversion difference: ', (B_conversion - A_conversion).mean()[1])
print('median conversion difference: ',(B_conversion - A_conversion).median()[1])
```

```
mean conversion difference:  0.0024366555642437756
median conversion difference:  0.0036586135848206275
```

Conclusions and conjectures¶

Both mean and median difference between A and B conversion are less than 0.1% meaning that none of them can be called a leader. The graph is symmetrical after A and B meeting on 2019-08-06. Except for the spikes at the first day (which still bring no difference to mean and median). So even though A conversion from 2019-08-06 till the end was lower the B's, A can't be considered as looser.
Worth mentioning, that all the fluctuations are very little and happened on the distance from 2.4 to 3.3 for B and from 2.5 to 3.5 for A: so, the fluctuations are less than 1%. The difference between final conversion rate is just 0.4%: 2.9% for B and 2.5% for A

Scatter chart of the number of orders per user

In [32]:
```
ordersByUsers = orders.drop(['group', 'revenue', 'date'], axis=1).groupby('visitorid', as_index=
False).agg({'transactionid' : pd.Series.nunique})
ordersByUsers.columns = ['userId', 'orders']
```

In [33]:
```
x_values = pd.Series(range(0, len(ordersByUsers)))
fig = px.scatter(ordersByUsers, x=x_values, y=ordersByUsers['orders'])
fig.update_layout(title_text="Number or orders per user",
                  title_font_size=30, xaxis_title="number of users", yaxis_title="number of orde
rs",
)

fig.show()
```

In [34]:
```
print(f'most frequent amount of orders is {int(ordersByUsers.orders.mode())},\nmiddle amount of
 orders is {int(ordersByUsers.orders.median())},\nmean amount of orders is also {round(ordersByU
sers.orders.mean(),2)},\n')
```

```
most frequent amount of orders is 1,
middle amount of orders is 1,
mean amount of orders is also 1.04,
```

Conclusions and conjectures

In general amount of purchases is 1, but we have 2 and 3 orders from 1 user. I would go for calling 3 an outlier: having 2 orders is rare, but it is a normal customer behaviour, we can't just cut them

95th and 99th percentiles for the number of orders per user

In [35]:
```
ordersByUsers['orders'].max()
```

Out[35]: 3

In [36]:
```
np.percentile(ordersByUsers['orders'], [90, 95, 99])
```

Out[36]: array([1., 1., 2.])

Conclusions and conjectures

Just 5% of users purchased more than 1 and only 1% bought more than 2. We can set 3 orders per user as an outlier (yes, 2 orders and above is the choise for 1% of users only, but I don't think that 2 orders is too much per user)

Scatter chart of order prices

In [37]:
```
x_values = pd.Series(range(0, len(orders)))
fig = px.scatter(ordersByUsers, x=x_values, y=orders['revenue'])
fig.update_layout(title_text="Price per order",
                  title_font_size=30, xaxis_title="index number of order", yaxis_title="price",
)

fig.show()
```

A closer look to the values

In [38]:
```
x_values = pd.Series(range(0, len(orders)))
fig = px.scatter(ordersByUsers, x=x_values, y=orders['revenue'])
fig.update_layout(title_text="Price per order",
                  title_font_size=30, xaxis_title="index number of order", yaxis_title="price",
)
fig.update_layout(yaxis_range=[0,1500])

fig.show()
```

In [39]:
```
orders.sort_values(by='revenue', ascending=False).head(10)
```

Out[39]:

	transactionid	visitorid	date	revenue	group
425	590470918	1920142716	2019-08-19	19920.400391	B
1196	3936777065	2108080724	2019-08-15	3120.100098	B
1136	666610489	1307669133	2019-08-13	1425.800049	A
744	3668308183	888512513	2019-08-27	1335.599976	B
743	3603576309	4133034833	2019-08-09	1050.000000	A
1103	1348774318	1164614297	2019-08-12	1025.800049	A
1099	316924019	148427295	2019-08-12	1015.900024	A
949	1347999392	887908475	2019-08-21	930.000000	A
940	2420050534	4003628586	2019-08-08	905.799988	B
613	4071177889	3931967268	2019-08-07	830.299988	B

Conclusions and conjectures

We have 1 enormous outlier of nearly 19920 per order and some other outliers that lay closer to average values. These must be one of the reasons for the data to be skewed. Also can see that 2 orders with highest price came from the B group and I would say, they boosted the revenue that happened from 2019-08-18 to 2019-08-19.
From the first glance, I would say that a price for on order can be considered an outlier if it is above 600, but with respect to the percentile (that can be found below), I would choose 450.

Percentile of order prices

In [40]:
```
np.percentile(orders['revenue'], [90, 95, 99]).round(2)
```

Out[40]: array([280.8 , 414.28, 830.3])

Conclusions and conjectures

Only 5% of orders cost more than 435.5 and only 1% more than 900. Let's consider 450 a base value for finding outliers.

Statistical significance of the difference in conversion

hypotheses description

Here we are to trying to establish whether there is statistically significant difference in **conversion** between the groups or not. We work with the **row data**. Lets formulate our hypotheses:

- Null hypotheses: there's no statistically significant difference in conversion between the groups A anb B
- Alternative hypotheses: there is statistically significant difference in conversion between the groups A anb B

In [41]:
```
ordersByUsersA = orders[orders['group'] == 'A'].groupby(
    'visitorid', as_index=False).agg({'transactionid' : pd.Series.nunique})
ordersByUsersA.columns = ['userId', 'orders']

ordersByUsersB = orders[orders['group'] == 'B'].groupby(
    'visitorid', as_index=False).agg({'transactionid' : pd.Series.nunique})
ordersByUsersB.columns = ['userId', 'orders']
```

In [42]:
```
sampleA = pd.concat([ordersByUsersA['orders'], pd.Series(0, index=np.arange(
    visits[visits['group'] == 'A']['visits'].sum() - len(ordersByUsersA['orders'])), name='order
s')], axis=0)
sampleB = pd.concat([ordersByUsersB['orders'], pd.Series(0, index=np.arange(
    visits[visits['group'] == 'B']['visits'].sum() - len(ordersByUsersB['orders'])), name='order
s')], axis=0)
alpha = .05
results = st.mannwhitneyu(sampleA, sampleB, True, 'less')[1]

if results < alpha:
    print('Null hypothesis rejected. There is a statistically significant difference between con
version from A anb B')
else:
    print('Null hypothesis can not be rejected, therefore no statistically significant differenc
e found')
```

```
Null hypothesis rejected. There is statistically significant difference between conversion from
A anb B
```

In [43]:
```
results #p-value
```

Out[43]:
```
0.005544298301416285
```

In [44]:
```
sampleB.mean()/sampleA.mean()-1
```

Out[44]:
```
0.159623735558315
```

Conclusions and conjectures

Using raw data we found statistically significant difference between conversion from A anb B and understood that A is less than B. Relative conversion gain of B is 15%, but lets see whether outliers have something to do with it or not.

Statistical significance of the difference in average order size

hypotheses description

Here we are to trying to establish whether there is statistically significant difference in the **average order size** per user to purchase between the groups. We work with the **row data**. Lets formulate our hypotheses:

- Null hypotheses: there's no statistically significant difference in the average order size between the groups A anb B
- Alternative hypotheses: there is statistically significant difference in the average order size between the groups A anb B

In [45]:
```
reslut_revenue = st.mannwhitneyu(
    orders[orders['group'] == 'A']['revenue'], orders[orders['group'] == 'B']['revenue'], True,
'less')[1]
alpha = .05
if reslut_revenue < alpha:
    print('Null hypothesis rejected. There is statistically significant difference in average or
der size for A anb B')
else:
    print('Null hypothesis can not be rejected, no statistically significant difference in avera
ge order size found')
```