Traffic Sources Analysis

Annotation

1. Telegram segment has better performance in terms of conversion rate (0.06% against 0.04% for posts) as well as better involvements (median is 23 clicks against 13 for posts).
2. At the same time Telegram has lower lead's quantity (61K against 157k), lower client's quantity (4000 against 6000) and lower average deposit (385 USD against 560 USD).
3. Around 2/3 of traffic and money came from posts.
4. There are 3 leading channels: Facebook, SMM, direct.
5. And two leading countries: Spain (ES) and Germany (DE).
6. My recommendation is to make a couple more studies to see telegram trend and to check France performing.
7. Till that time, I can recommend to follow the exists strategy.
8. In case we need to choose channels it's better to focus on Facebook, SMM, direct.
   For countries preferable choices are Spain (ES) and Germany (DE).

General information

- **Beneficiary:** marketing department.
- **Research goal:** to optimize marketing activity.
- **Research tasks:**
  - provide dataset study;
  - find out best sources and channels;
  - make recommendations on marketing activity.
- **Research steps:**
  - data preparation: cleaning gaps, duplicates and deviation study;
  - EDA and data visualization.
  - conclusion.

Data description:

1. file "synthetic_data":
   a. depo — amount of deposit, USD;
   b. segment/source - source of traffic acquisition, there are two possible sources (posts and telegram channel):
      i. "postid" — the lead came from article. id of post doesn't matter,
      ii. "telegram" — the lead came from telegram;
   c. channel — channel of traffic, for example, user can come from 'telegram' source and from 'affiliate' channel;
   d. clicks — amount of clicks user made during first day after registration;
   e. latency — time of application loading in milliseconds;
   f. client_id — it's assigned during registration and isn't changed anymore;
2. file "country":
   a. country of lead/client (iso2);
   b. client_id — it's assigned during registration and isn't changed anymore.

Data preparation

```
!pip install -U pandas
!pip install -U plotly
```

```
Requirement already satisfied: pandas in c:\users\acer\anaconda3\lib\site-packages (1.4.2)
Requirement already satisfied: numpy>=1.18.5 in c:\users\acer\anaconda3\lib\site-packages (from
pandas) (1.20.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\acer\anaconda3\lib\site-packages
(from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\acer\anaconda3\lib\site-packages (from
pandas) (2021.3)
Requirement already satisfied: six>=1.5 in c:\users\acer\anaconda3\lib\site-packages (from python-
dateutil>=2.8.1->pandas) (1.16.0)
Requirement already satisfied: plotly in c:\users\acer\anaconda3\lib\site-packages (5.8.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\acer\anaconda3\lib\site-packages (from
plotly) (8.0.1)
```

```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```
# saving file's names to variables
a = 'synthetic_data'
b = 'countries'
```

```
# reading files
data = pd.read_csv(a + '.csv')
countries = pd.read_csv(b + '.csv')
```

File 'synthetic_data'. Study

As the first step let's check general information about the dataset.

```
# using method info()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 219314 entries, 0 to 219313
Data columns (total 7 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   Unnamed: 0 219314 non-null int64
 1   depo     219314 non-null int64
 2   segment    219314 non-null object
 3   channel    217142 non-null object
 4   clicks    219314 non-null float64
 5   latency    219314 non-null float64
 6   client_id  219314 non-null int64
dtypes: float64(2), int64(3), object(2)
memory usage: 11.7+ MB
```

```python
# checking NA
print('NA amount:')
print(data.isnull().sum())
```

```
NA amount:
Unnamed: 0    0
depo         0
segment      0
channel   2172
clicks       0
latency      0
client_id    0
dtype: int64
```

```python
# and NA share
print('NA share: {:.2%}'.format(data['channel'].isnull().sum() / len(data)))
```

```
NA share: 0.99%
```

Dataset contains 219314 lines, only one column — channel — contains NA values and the share of these values is less than 1%. This information can't be restored from the other columns. Column 'segment' is filled for these lines so we can still use lines with NA to research segments performance, that's why we are not going to drop them.

There is a redundant column 'Unnamed', which is better to drop. Also, the dtype of column 'clicks' doesn't fit with the content. It is been expected that clicks must be integer. Let's try to change the dtype.

```python
# cleaning the extra column
data_cl = data.drop(columns=['Unnamed: 0'], axis = 1)
# changing dtype
data_cl['clicks'] = data_cl['clicks'].astype('int64')
# checking the result
data_cl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 219314 entries, 0 to 219313
Data columns (total 6 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   depo     219314 non-null int64
 1   segment   219314 non-null object
 2   channel   217142 non-null object
 3   clicks    219314 non-null int64
 4   latency   219314 non-null float64
 5   client_id 219314 non-null int64
dtypes: float64(1), int64(3), object(2)
memory usage: 10.0+ MB
```

Let's look closely to dataset: if it has duplicates and ask to show five first rows to see data examples.

```python
# declare a function to data check: duplicates, duplicates share, first 5 lines to visual check
def check(data):
 display(data.head())
 duplicates = data.duplicated().sum()
 duplicates_part = duplicates / len(data)
 print('Duplicated lines, amount:', duplicates)
 print('Duplicated lines, share: {:.2%}'.format(duplicates_part))
```

```
# let's check the first dataset
check(data_cl)
```

	depo	segment	channel	clicks	latency	client_id
0	0	postid_4057	smm	1	2.649725	1442498
1	0	telegram	affiliate	10	2.610846	7865631
2	0	postid_8542	facebook	13	3.001162	8165584
3	0	telegram	direct	0	1.788369	5893056
4	0	telegram	smm	0	1.932069	3780924

```
Duplicated lines, amount: 0
Duplicated lines, share: 0.00%
```

There are no duplicates in dataset. Column's types are corresponding to the content. All other parameters seem fine. Let's clear the segment column: according to description it has only 2 different sources — telegram and post (all the variety is about post's id).

```
# splitting the target column, drop extradata and rename
m = data_cl['segment'].str.split('_',expand=True)
m.columns=['segment','for_dropping']
m = m.drop(columns=['for_dropping'], axis = 1)
# droping the original column from the dataset
data_up = data_cl.drop(columns=['segment'], axis = 1)
# combining datasets
data_up = pd.concat([data_up, m], axis=1)
# checking if there are only two options for the source
data_up.groupby('segment')['segment'].count()
```

```
segment
postid    157560
telegram    61754
Name: segment, dtype: int64
```

First file is ready. Let's check the second one.

File 'countries'. Study

```
# checking general information about dataset
countries.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 219314 entries, 0 to 219313
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   country   219314 non-null  object
 1   client_id 219314 non-null  int64
dtypes: int64(1), object(1)
memory usage: 3.3+ MB
```

There is no missing data in this file. With the function which has been declared before let's check the second dataset for duplicates.

```
# using declared function
check(countries)
```

	country	client_id
0	IN	6348826
1	FR	6751691
2	DE	8638448
3	LT	4722696
4	ES	2411132

```
Duplicated lines, amount: 61754
Duplicated lines, share: 28.16%
```

There are almost 30% of duplicates in this dataset. Total number of lines is equal for both files so it might be several lines for some unique client's ids in 'synthetic_data', which represents visits from different sources. Let's check duplicates for client_id (previously we checked the whole line for duplicates).

```
# counting unique ids in the dataset
data_up['client_id'].nunique()
```

```
219314
```

The assumption we made before was incorrect: file 'synthetic_data' contains only unique client ids. That's mean for some client we might not have information about the country. For not to overestimate the number of users per country lets drop duplicates for countries dataset.

```
# using method drop_duplicates() to clean the data
countries_cl = countries.drop_duplicates()
countries_cl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 157560 entries, 0 to 219313
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   country   157560 non-null object
 1   client_id 157560 non-null int64 
dtypes: int64(1), object(1)
memory usage: 3.6+ MB
```

Let's unite two datasets to have full information in one table. Supposed NA we are going to replace with 'nd' = no data.

```
# using merge() to unite data
df = data_up.merge(countries_cl, on='client_id', how='left').fillna('nd')
df.sample(10)
```

	depo	channel	clicks	latency	client_id	segment	country
167023	0	smm	0	3.709683	8177718	telegram	nd
209575	0	social media	1	4.045738	5746787	telegram	nd
5964	0	direct	27	3.164631	7257175	telegram	IS
169211	0	direct	9	2.887288	8362207	postid	nd
182651	0	social media	31	3.259292	3477615	postid	nd
46135	0	facebook	0	2.215026	8647832	telegram	US
141455	0	smm	6	2.672162	3472960	postid	IS
65873	0	social media	0	2.556720	3228783	postid	LT
68923	0	smm	2	3.503075	4537987	postid	US
49609	0	social media	0	2.982904	5768490	postid	IN

```
# checking is everything worked
print('NA in the column "country":', df.query('country == "nd"')['country'].count())
```

```
NA in the column "country": 61754
```

The NA number is totally similar for the amount of the lines with telegram source. Check if all the telegram segment doesn't have country.

```
# country NA by source
df.query('country == "nd"').groupby('segment')['segment'].count()
```

```
segment
postid    44401
telegram  17353
Name: segment, dtype: int64
```

As it was expected 61754 lines do not have information about the country (these lines belong to different sources). For now, let's just have it in mind moving to the next EDA step.

EDA

Firstly, let's check numerical parameters.

```
# using describe() method
df[['depo', 'clicks', 'latency']].describe()
```

	depo	clicks	latency
count	219314.000000	219314.000000	219314.000000
mean	22.361217	11.430114	3.021579
std	397.835611	12.628842	1.048472
min	-164.000000	0.000000	0.000071
25%	0.000000	0.000000	2.320726
50%	0.000000	8.000000	3.001301
75%	0.000000	19.000000	3.693649
max	31675.000000	50.000000	11.016521

From the first look the column 'latency' is perfectly fine with mean = 3 and std = 1 (standard deviation): all values are not so far away from the mean value. The situation a bit more complicated for 'clicks' column: std = 12.6 and mean = 11.4 which can be explained with huge number of zero-value lines: even 25% percentile is equal zero. So, quarter or so users do not make any clicks in the first day after registration. Clicks_max doesn't rise any questions, numbers look true-to-life. Depo columns on the other hand has some unexpected data which are negative deposits. Plus 75% percentile for the column is still zero. Let's check how big is this problem.

```
# selecting negative lines only
negative_depo = df.query('depo < 0')

# counting number and share of the negative lines
print('Negative depo, amount:', negative_depo['depo'].count())
print('Negative depo, share: {:.2%}'.format(negative_depo['depo'].count() / len(df)))
```

```
Negative depo, amount: 108
Negative depo, share: 0.05%
```

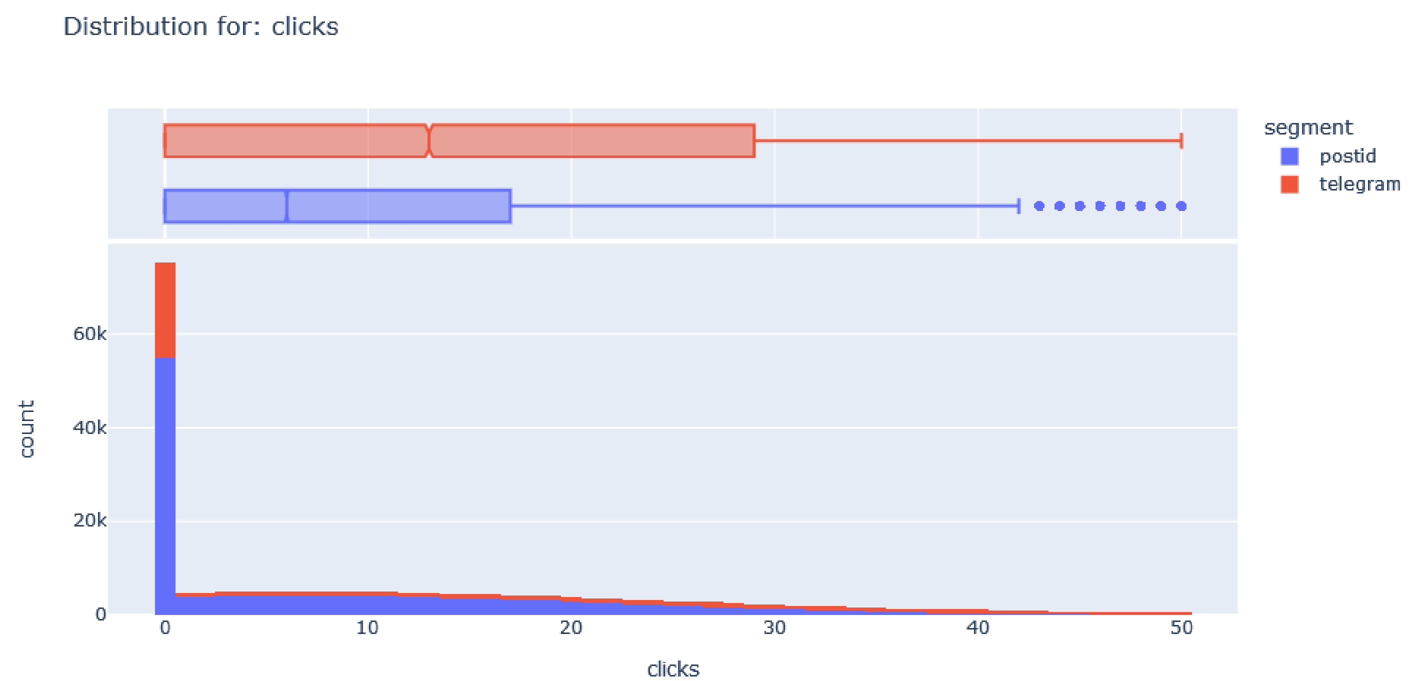
Negative deposits are only 0.05% from all of the lines, so we are going to drop them. There is no suggestion of the origin of this problem so it's better to place ticket for tech team to check (all available information for checking is in the dataframe 'negative_depo').

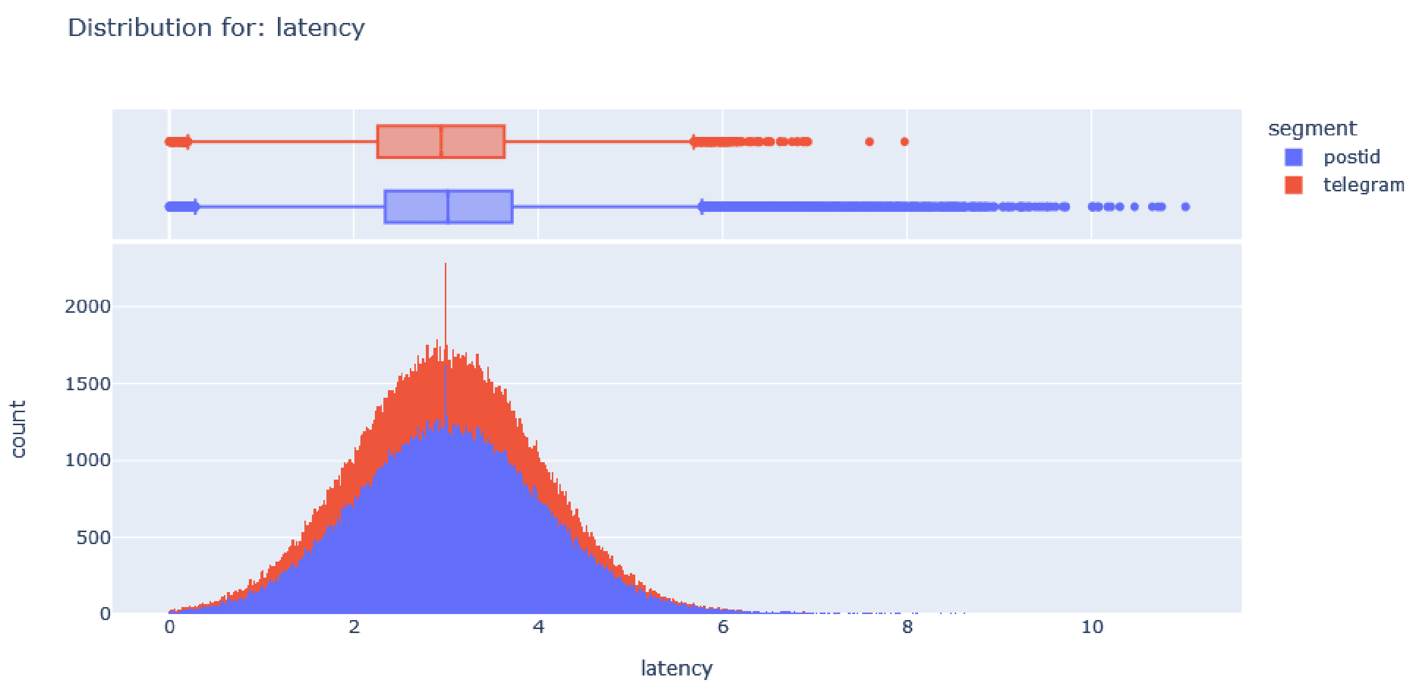
```
# dataset without negative depo
df_up = df.query('depo >= 0')
df_up.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 219206 entries, 0 to 219313
Data columns (total 7 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   depo     219206 non-null int64
 1   channel   219206 non-null object
 2   clicks    219206 non-null int64
 3   latency   219206 non-null float64
 4   client_id 219206 non-null int64
 5   segment   219206 non-null object
 6   country   219206 non-null object
dtypes: float64(1), int64(3), object(3)
memory usage: 13.4+ MB
```

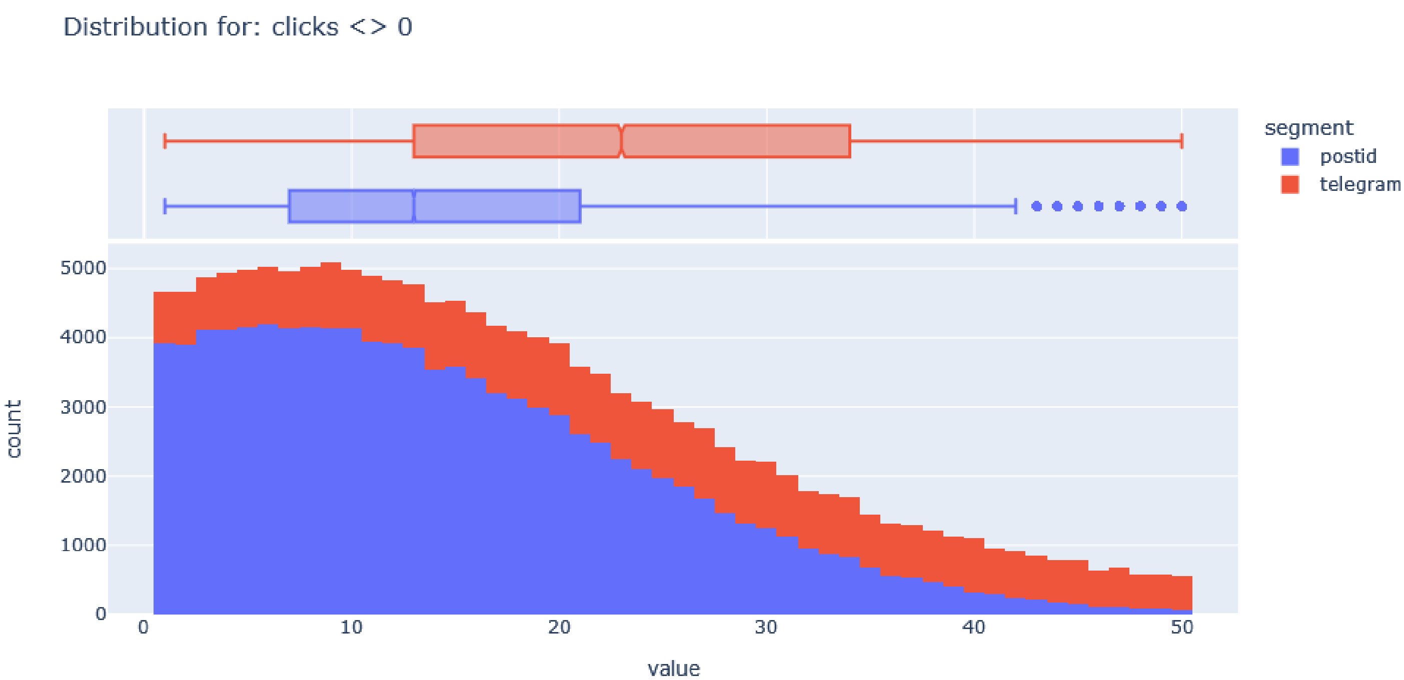
New dataset contains 219206 lines. Let's make graphs of distribution for numeric parameters except 'depo' because of high level of zero values.

```
# making dataset without client_id
for_graph = df_up[['clicks', 'latency', 'segment']]

# making graph for each column of dataset
for col in for_graph.drop(columns=['segment']).columns:
  fig = px.histogram(for_graph, x = col, marginal = 'box', color = 'segment', title = 'Distribution for: '+col)
  fig.show()
```





```python
# checking clicks distribution without 0
fig = px.histogram(df_up.query('clicks != 0'), x = ['clicks'], marginal = 'box', color = 'segment',
        color_discrete_map={
       'telegram': '#EF553B',
       'postid': '#636EFA'},
        title = 'Distribution for: clicks <> 0')
fig.show()
```



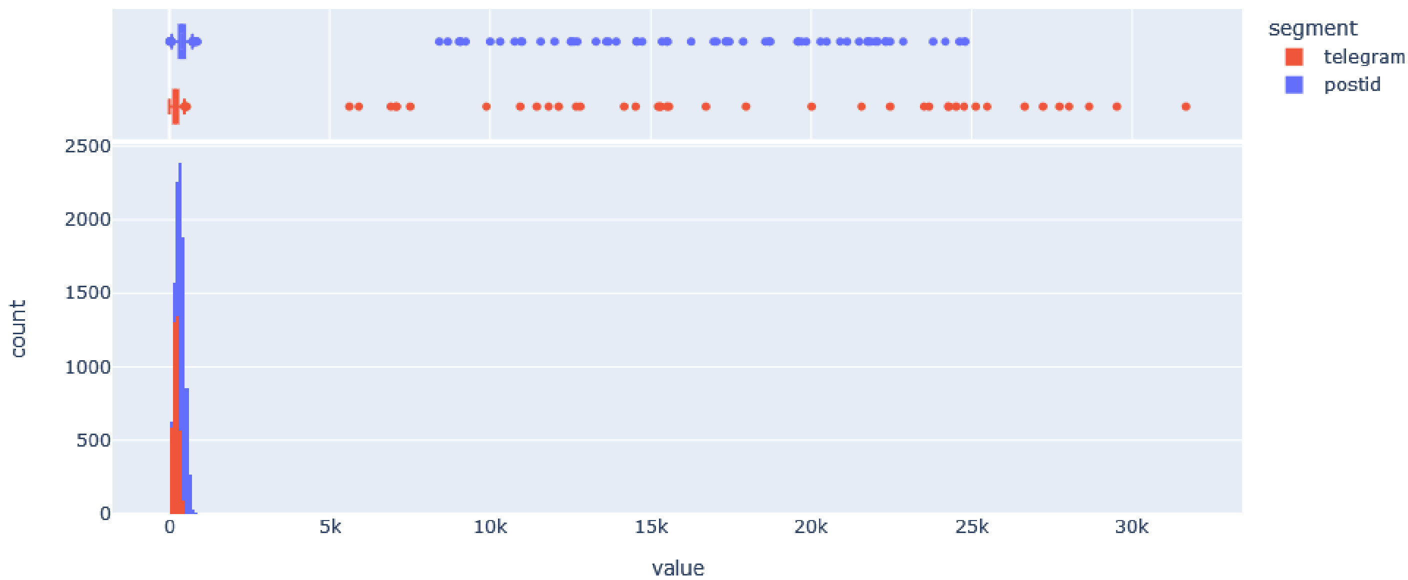
Distributions of 'clicks' and 'latency' parameters match the expectation. While examine graphs we can see:

- clicks, all data: median for telegram (13) is bigger than for posts (6);
- clicks, all data: 75% percentile for telegram is bigger as well (29 against 17);
- latency is slightly better for telegram too: 2.95 against 3.02;
- telegram 2.5 times yields in total user quantity: 61754 against 157560 for posts.

Latency has outliers for both telegram and post around 3 seconds. It is close to median so it seems like the result of previous data processing. Let's make a graph for 'depo' column without zero values.

```python
# making graph with the same color code
fig = px.histogram(df_up.query('depo != 0'), x = ['depo'], marginal = 'box', color = 'segment',
        color_discrete_map={
       'telegram': '#EF553B',
       'postid': '#636EFA'},
        title = 'Distribution for: deposits <> 0')
fig.show()
```

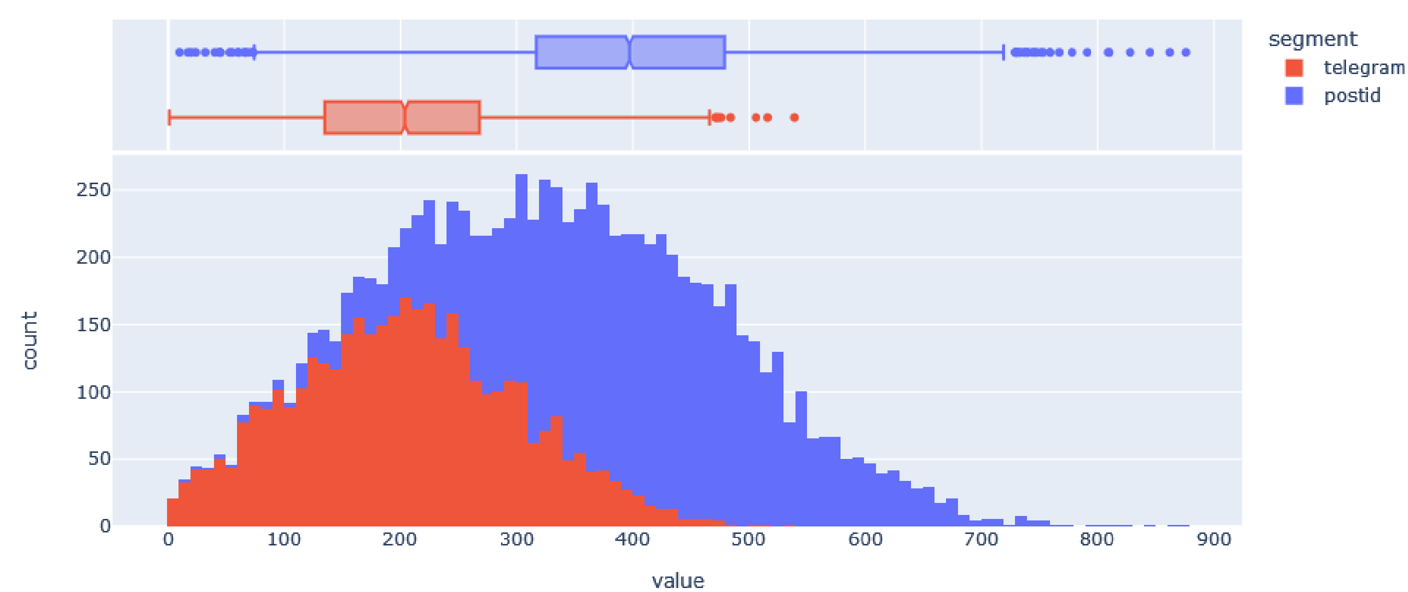
Distribution for: deposits <> 0



The graph is still not very demonstrative: we can see all the values focused around zero and the maximum (without outliers) is about 700 USD. I can't make significant assumptions about outliers — deposits with 10K or 30k might be common for the business. I don't think I should correct them. But let's check distribution for the main data. For this purpose, we are going to cut the data again.

```
# making graph with the same color code
fig = px.histogram(df_up.query('depo != 0 and depo < 1000'), x = ['depo'], marginal = 'box', color = 'segment',
        color_discrete_map={
      'telegram': '#EF553B',
      'postid': '#636EFA'},
        title = 'Distribution for: deposits less than 1000 by segment')
fig.show()
```

Distribution for: deposits less than 1000 by segment



Depo-distribution looks as expected.

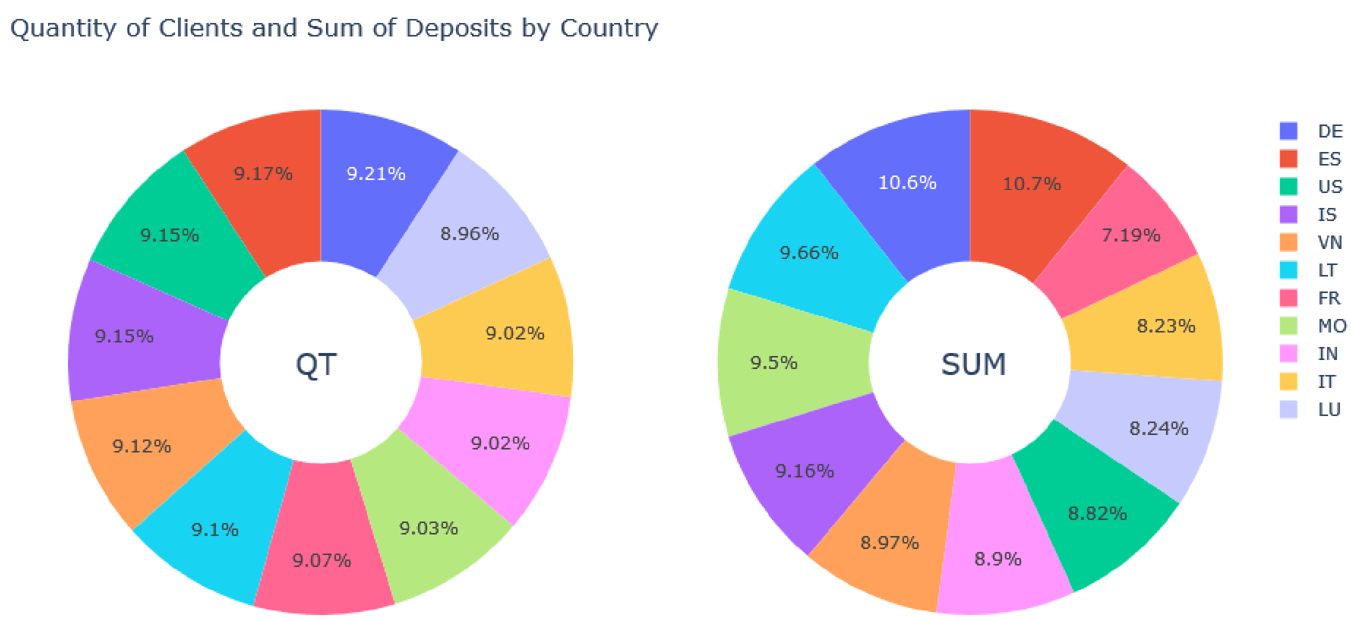
- posts have greater median (379 USD against 204 USD for telegram);
- posts contribution to deposits in general seems prevailing.

Let's see how users and deposits are distributed over countries.

```
# declaring a function to make pie charts
def pie_maker(data, col_1, col_2, col_3, title):
 labels = col_1
 fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])
 fig.add_trace(go.Pie(labels=labels, values=col_2, name='clients'),
    1, 1)
 fig.add_trace(go.Pie(labels=labels, values=col_3, name='deposit'),
    1, 2)
 fig.update_traces(hole=.4, hoverinfo='label+percent+name')
 fig.update_layout(
  title_text=title,
  annotations=[dict(text='QT', x=0.20, y=0.5, font_size=20, showarrow=False),
        dict(text='SUMM', x=0.81, y=0.5, font_size=20, showarrow=False)])
 fig.show()
```

```
# creating dataset without NA in countries
pie = df_up.query('country != "nd"').groupby('country').agg({'depo': 'sum', 'client_id':'count'})\
   .reset_index().rename(columns={'depo': 'sum', 'client_id':'quantity'})
```

```
pie_maker(pie, pie['country'], pie['quantity'], pie['sum'], 'Quantity of Clients and Sum of Deposits by Country')
```



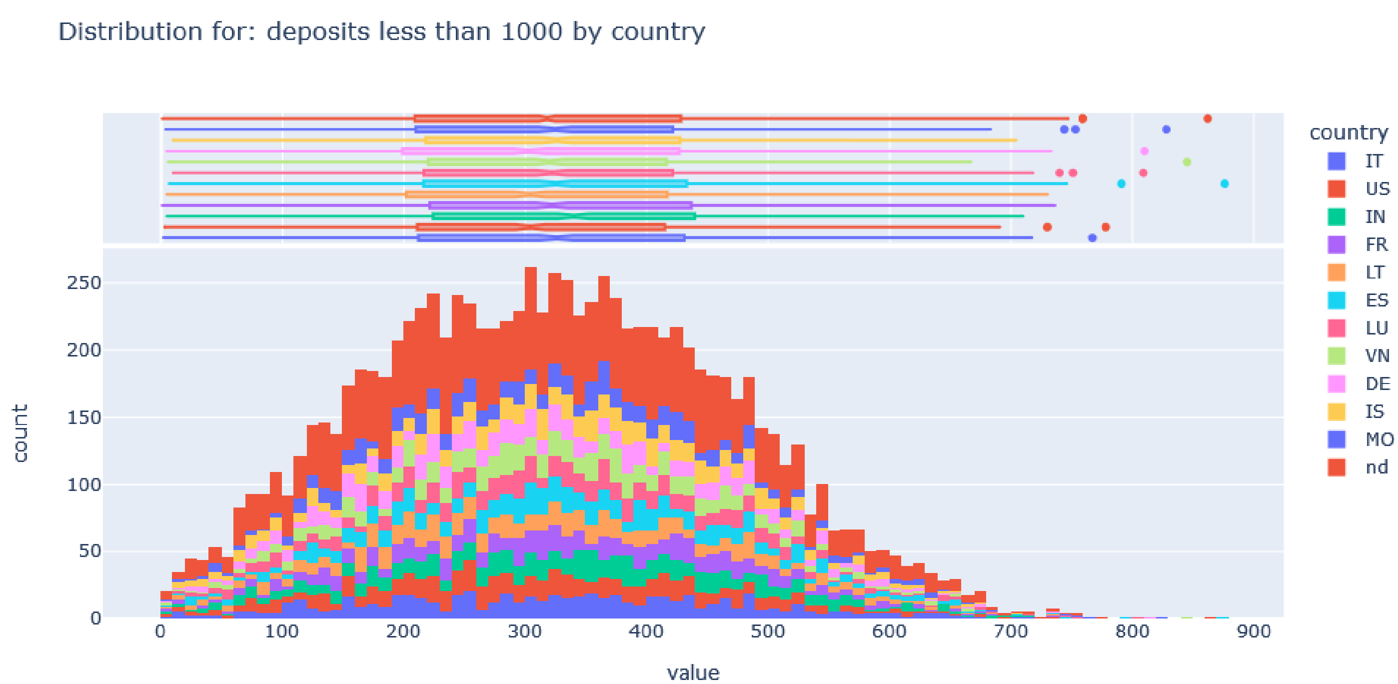
According to the graph clients evenly distributed over countries: minimum share is 8.96 and maximum is 9.21. Deposits sum is not that flat: from 7.19 to 10.7. Let's check average deposit sum for country.

```
# making a table from the existed data frame
pie['avg_depo'] = pie['sum'] / pie['quantity']
pie['avg_depo'] = pie['avg_depo'].apply(lambda x: '{:.2f}'.format(x))
pie.sort_values(by='avg_depo', ascending=False)
```

	country	summ	quantity	avg_depo
1	ES	376270	14441	26.06
0	DE	370878	14501	25.58
6	LT	338112	14332	23.59
8	MO	332463	14220	23.38
4	IS	320635	14403	22.26
3	IN	311522	14209	21.92
10	VN	313932	14367	21.85
9	US	308791	14417	21.42
7	LU	288583	14111	20.45
5	IT	288078	14201	20.29
2	FR	251849	14288	17.63

There is almost 50% difference between country with the highest average deposit (ES=26) and country with the lowest (FR=17.6). Check if the reason is a few extra big deposits.

```
# declaring a function for hist making
def hist_maker(data, col_1, title):
  fig = px.histogram(data, x = ['depo'], marginal = 'box', color = col_1, title = title)
  fig.show()

# making proper dataset
d = df_up.query('depo != 0 and depo < 1000')
hist_maker(d, d['country'], 'Distribution for: deposits less than 1000 by country')
```



All the distributions look alike and normal.

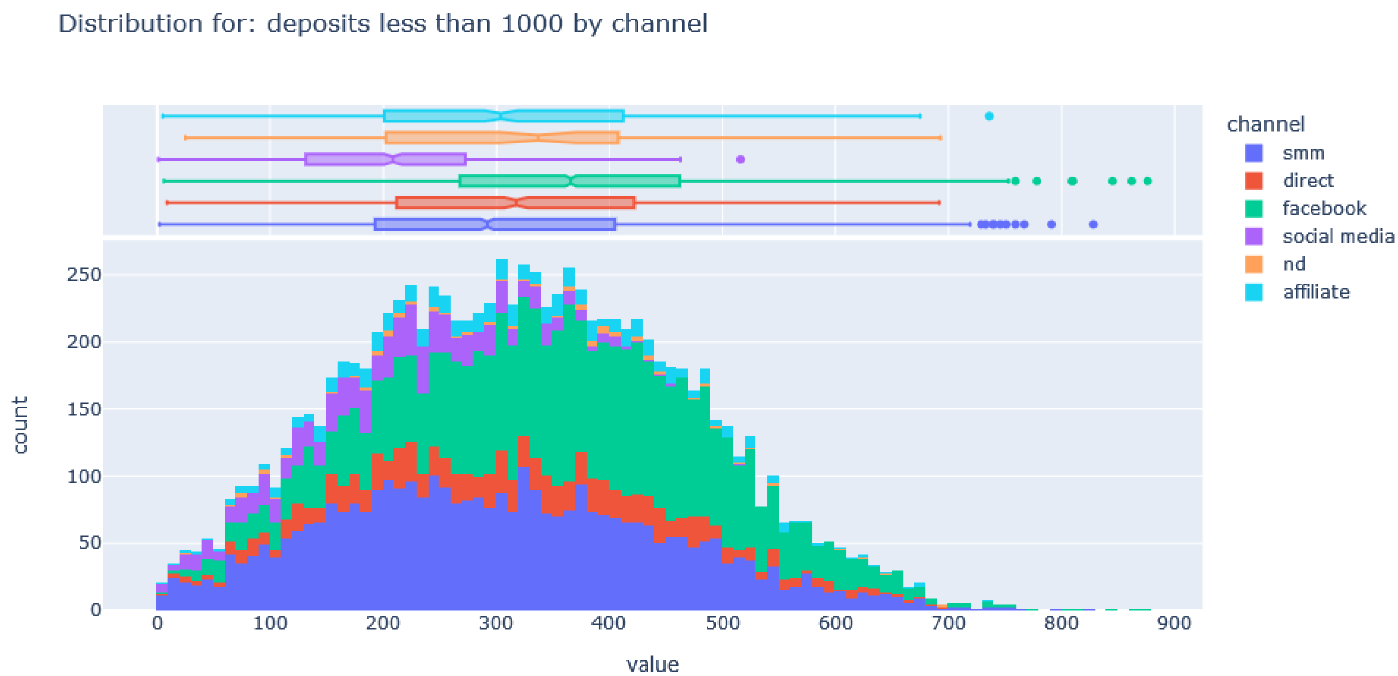
```
# making table with remaining data
d_1000 = df_up.query('depo > 1000 and country !="nd"').groupby('country')['client_id'].count().reset_index()\
                .sort_values(by='client_id', ascending=False).rename(columns={'client_id':'more_than_1000'})
d_10000 = df_up.query('depo > 10000 and country !="nd"').groupby('country')['client_id'].count().reset_index()\
                .sort_values(by='client_id', ascending=False).rename(columns={'client_id':'more_than_10000'})
d_20000 = df_up.query('depo > 20000 and country !="nd"').groupby('country')['client_id'].count().reset_index()\
                .sort_values(by='client_id', ascending=False).rename(columns={'client_id':'more_than_20000'})

from functools import reduce
data_frames = [d_1000, d_10000, d_20000]
data_merged = reduce(lambda left,right: pd.merge(left,right,on=['country'], how='outer'), data_frames).fillna(0)
data_merged
```

	country	more_than_1000	more_than_10000	more_than_20000
0	DE	10	8	3.0
1	ES	8	8	3.0
2	LT	8	7	2.0
3	MO	7	6	4.0
4	IS	6	5	2.0
5	US	6	6	2.0
6	VN	6	6	2.0
7	IN	5	5	3.0
8	LU	5	4	2.0
9	FR	4	3	0.0
10	IT	4	4	2.0

Seems like DE and ES are the leading countries with client's quantity and deposit's sum (in total and by segments). But this conclusion could be done based on graphs. FR in fact has a smaller number of expensive deposits (3 for more than 10000 and 0 for more than 20000). This point needs to be researched separately.

Let's check distribution by channel (the only one left aside).

```
# using declared function
list_maker(d, d['channel'], 'Distribution for: deposits less than 1000 by channel')
```



There are two main channels in terms of user quantity: Facebook and SMM. Moreover, Facebook has the greatest median (366 USD).
The rest looks this way:

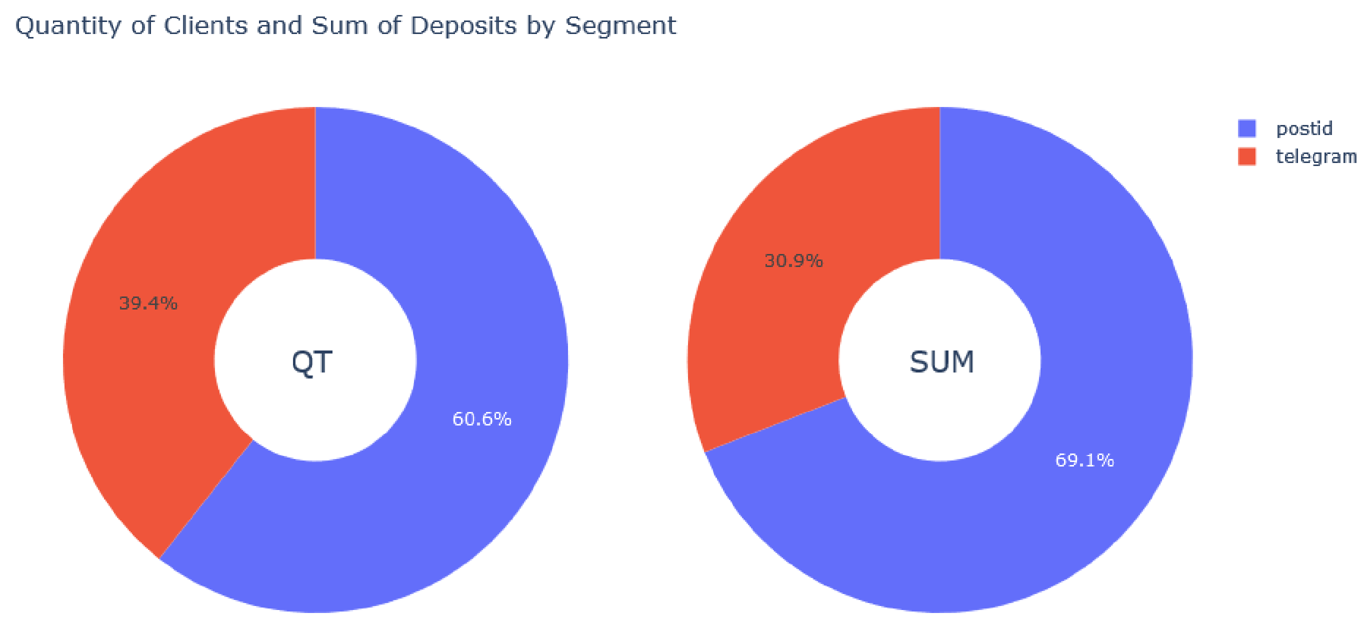
2. direct = 317;
3. affiliate = 304;
4. SMM = 292;
5. social media = 208.5.

```
# declaring a function to make tables
def avg_table(data, col_1):
 d = data.groupby(col_1).agg({'depo': 'sum', 'client_id':'count'})\
    .reset_index().rename(columns={'depo': 'sum', 'client_id':'quantity'})
 d['avg_depo'] = d['sum'] / d['quantity']
 d['avg_depo'] = d['avg_depo'].apply(lambda x:'{:.2f}'.format(x))
 d = d.sort_values(by='avg_depo', ascending=False)
 return d

# making table
avg_table(df_up, df_up['channel'])
```

	channel	sum	quantity	avg_depo
5	social media	354997	42220	8.41
3	nd	60753	2170	28.00
2	facebook	2104138	76497	27.51
4	smm	1701455	65764	25.87
1	direct	515143	21704	23.73
0	affiliate	170996	10851	15.76

Average deposit is bigger for Facebook as well. The second channel on this parameter is SMM, third one is direct. They are 3 channels with both greater client's quantity and greater average deposit. Let's take a look on a conversion rate by segments. First will visualize data with pie charts.

```
# making dataset
seg_1 = df_up.groupby('segment').agg({'depo': 'sum', 'client_id': 'count'})\
    .reset_index().rename(columns={'depo': 'sum', 'client_id': 'leads'})
seg_2 = df_up.query('depo != 0').groupby('segment').agg({'client_id': 'count'})\
   .reset_index().rename(columns={'client_id': 'clients'})
seg = seg_1.merge(seg_2, on='segment')

# using declared function
pie_maker(seg, seg['segment'], seg['clients'], seg['sum'], 'Quantity of Clients and Sum of Deposits by Segment')
```



Posts generate about 2/3 of traffic and money. Telegram share in quantity is more than telegram share in sum so average deposit must be lower. Let's check numbers at the table.

```
# adding to the table a couple new columns
seg['avg_depo'] = seg['sum'] / seg['clients']
seg['avg_depo'] = seg['avg_depo'].apply(lambda x: '{:.2f}'.format(x))
seg['conversion_rate'] = seg['clients'] / seg['leads']
seg['conversion_rate'] = seg['conversion_rate'].apply(lambda x: '{:.2f}%'.format(x))
seg
```

	segment	sum	leads	clients	avg_depo	conversion_rate
0	postid	3390223	157557	6057	559.72	0.04%
1	telegram	1517259	61649	3933	385.78	0.06%

Telegram average deposit is 30% lower than the posts. But conversion rate is 50% higher: more clients who came via telegram making a deposit. Yet in absolute figures posts still give more clients (6000 against 4000).

Conclusion

1. Dataset has negative values in column 'depo'. Total amount is 0.05% which is not critical for the report but it might need an investigation (dataset for downloading = negative_depo)
2. Telegram segment has better performance in terms of conversion rate (0.06% against 0.04% for posts) as well as better involvements (median is 23 clicks against 13 for posts) but I can't recommend to put in more money into the source because of lower lead's quantity (61K against 157k), lower client's quantity (4000 against 6000) and lower average deposit (385 USD against 560 USD).
3. Around 2/3 of traffic and money came from posts.
4. Speaking of channels: there are 3 leading channels
   - Facebook (deposit median = 366 USD, average deposit 27.5 USD);
   - SMM (median = 292 USD, avg deposit 25.8 USD);
   - Direct (median = 317 USD, avg deposit 23.7 USD).
5. And there are 2 leading countries: Spain (ES) and Germany (DE), though the difference between counties is not highly expressed.

Recommendations

1. Make separate study about telegram dynamics: it seems like a preferable source but to put in more money we need to be sure in trend.
2. Make separate study for deposit outliers: what are the people who make deposits with 15k-30k USD, how to attract more of them.
3. Make separate study for country France (FR): for this region there are no big clients. May be there are some problems with sources or messages. Might be some sort of national characteristics.
4. Till that time, I can recommend to follow the exists strategy. In case we need to choose channels it's better to focus on Facebook, SMM, direct. For countries preferable choices are Spain (ES) and Germany (DE).