Telecom company operators' effficiency analysis

Project description

The virtual telephony service CallMeMaybe is developing a new function that will give supervisors information on the least effective operators. An operator is considered ineffective if they have a large number of missed incoming calls (internal and external) and a long waiting time for incoming calls. Moreover, if an operator is supposed to make outgoing calls, a small number of them is also a sign of ineffectiveness.

Table of Contents

Data Preprocessing

In [1]:
```
#load libraries
!pip install seaborn --upgrade
!pip install plotly

import matplotlib.pyplot as plt
import matplotlib as mpl
import re
import numpy as np
import pandas as pd
import seaborn as sns
import math as mth
import warnings; warnings.simplefilter('ignore')
import plotly.express as px


from functools import reduce
from math import factorial
from scipy import stats as st
from statistics import mean
from IPython.display import display
from plotly import graph_objects as go
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import KMeans


pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 100)
```

```
Requirement already up-to-date: seaborn in c:\users\michael\anaconda3\lib\site-packages (0.11.1)
Requirement already satisfied, skipping upgrade: pandas>=0.23 in c:\users\michael\anaconda3\lib\site-packages (from seabo
rn) (1.1.3)
Requirement already satisfied, skipping upgrade: numpy>=1.15 in c:\users\michael\anaconda3\lib\site-packages (from seabor
n) (1.19.2)
```

```
Requirement already satisfied, skipping upgrade: scipy>=1.0 in c:\users\michael\anaconda3\lib\site-packages (from seabor
n) (1.5.2)
Requirement already satisfied, skipping upgrade: matplotlib>=2.2 in c:\users\michael\anaconda3\lib\site-packages (from se
aborn) (3.3.2)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.7.3 in c:\users\michael\anaconda3\lib\site-packages
(from pandas>=0.23->seaborn) (2.8.1)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in c:\users\michael\anaconda3\lib\site-packages (from panda
s>=0.23->seaborn) (2020.1)
Requirement already satisfied, skipping upgrade: pillow>=6.2.0 in c:\users\michael\anaconda3\lib\site-packages (from matp
lotlib>=2.2->seaborn) (8.0.1)
Requirement already satisfied, skipping upgrade: certifi>=2020.06.20 in c:\users\michael\anaconda3\lib\site-packages (fro
m matplotlib>=2.2->seaborn) (2020.6.20)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in c:\users\michael\anaconda3\lib\site-packages (from
matplotlib>=2.2->seaborn) (1.3.0)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\michael\anaconda3\l
ib\site-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied, skipping upgrade: cycler>=0.10 in c:\users\michael\anaconda3\lib\site-packages (from matpl
otlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied, skipping upgrade: six>=1.5 in c:\users\michael\anaconda3\lib\site-packages (from python-da
teutil>=2.7.3->pandas>=0.23->seaborn) (1.15.0)
Requirement already satisfied: plotly in c:\users\michael\anaconda3\lib\site-packages (4.14.1)
Requirement already satisfied: six in c:\users\michael\anaconda3\lib\site-packages (from plotly) (1.15.0)
Requirement already satisfied: retrying>=1.3.3 in c:\users\michael\anaconda3\lib\site-packages (from plotly) (1.3.3)
```

Read dataframes and have a general look at them.

In [2]:
```
try:
    clients = pd.read_csv('telecom_clients_us.csv')
    dataset = pd.read_csv('telecom_dataset_us.csv')
except:
    clients = pd.read_csv('/datasets/telecom_clients_us.csv')
    dataset = pd.read_csv('/datasets/telecom_dataset_us.csv')

display(clients.head())
display(clients.describe(include='all'))
display(clients.info())
display(dataset.head())
display(dataset.describe(include='all'))
display(dataset.info())
```

	user_id	tariff_plan	date_start
0	166713	A	2019-08-15
1	166901	A	2019-08-23
2	168527	A	2019-10-29

	user_id	tariff_plan	date_start
3	167097	A	2019-09-01
4	168193	A	2019-10-16

	user_id	tariff_plan	date_start
count	732.000000	732	732
unique	NaN	3	73
top	NaN	C	2019-09-24
freq	NaN	395	24
mean	167431.927596	NaN	NaN
std	633.810383	NaN	NaN
min	166373.000000	NaN	NaN
25%	166900.750000	NaN	NaN
50%	167432.000000	NaN	NaN
75%	167973.000000	NaN	NaN
max	168606.000000	NaN	NaN

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 732 entries, 0 to 731
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   user_id      732 non-null    int64 
 1   tariff_plan  732 non-null    object
 2   date_start   732 non-null    object
dtypes: int64(1), object(2)
memory usage: 17.3+ KB
None
```

	user_id	date	direction	internal	operator_id	is_missed_call	calls_count	call_duration	total_call_duration
0	166377	2019-08-04 00:00:00+03:00	in	False	NaN	True	2	0	4
1	166377	2019-08-05 00:00:00+03:00	out	True	880022.0	True	3	0	5
2	166377	2019-08-05 00:00:00+03:00	out	True	880020.0	True	1	0	1

	user_id	date	direction	internal	operator_id	is_missed_call	calls_count	call_duration	total_call_duration
3	166377	2019-08-05 00:00:00+03:00	out	True	880020.0	False	1	10	18
4	166377	2019-08-05 00:00:00+03:00	out	False	880022.0	True	3	0	25

	user_id	date	direction	internal	operator_id	is_missed_call	calls_count	call_duration	total_call_duration
count	53902.000000	53902	53902	53785	45730.000000	53902	53902.000000	53902.000000	53902.000000
unique	NaN	119	2	2	NaN	2	NaN	NaN	NaN
top	NaN	2019-11-25 00:00:00+03:00	out	False	NaN	False	NaN	NaN	NaN
freq	NaN	1220	31917	47621	NaN	30334	NaN	NaN	NaN
mean	167295.344477	NaN	NaN	NaN	916535.993002	NaN	16.451245	866.684427	1157.133297
std	598.883775	NaN	NaN	NaN	21254.123136	NaN	62.917170	3731.791202	4403.468763
min	166377.000000	NaN	NaN	NaN	879896.000000	NaN	1.000000	0.000000	0.000000
25%	166782.000000	NaN	NaN	NaN	900788.000000	NaN	1.000000	0.000000	47.000000
50%	167162.000000	NaN	NaN	NaN	913938.000000	NaN	4.000000	38.000000	210.000000
75%	167819.000000	NaN	NaN	NaN	937708.000000	NaN	12.000000	572.000000	902.000000
max	168606.000000	NaN	NaN	NaN	973286.000000	NaN	4817.000000	144395.000000	166155.000000

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53902 entries, 0 to 53901
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   user_id              53902 non-null  int64  
 1   date                 53902 non-null  object 
 2   direction            53902 non-null  object 
 3   internal             53785 non-null  object 
 4   operator_id          45730 non-null  float64
 5   is_missed_call       53902 non-null  bool   
 6   calls_count          53902 non-null  int64  
 7   call_duration        53902 non-null  int64  
 8   total_call_duration  53902 non-null  int64  
dtypes: bool(1), float64(1), int64(4), object(3)
memory usage: 3.3+ MB
None
```

In [3]:
```
dataset.operator_id.nunique()
```

Out[3]:
```
1092
```

So few conclusions based on data:

- there are 732 users of CallMeMaybe service that were taken into account;
- there are 3 different tariffs;
- there are 1092 operators;
- there are some strange outliers with too long calls in call_duration column;
- also there are some outliers in calls_count columns.

Convert datatypes to proper ones

In [4]:
```
clients['date_start'] = pd.to_datetime(clients['date_start'])
dataset['date'] = pd.to_datetime(dataset['date'])
```

Check data for missing values

In [5]:
```
for column in dataset.columns:
    print ('Number of missed enteries in', column,'column:', len(dataset[dataset[column].isna() == True]))
```

```
Number of missed enteries in user_id column: 0
Number of missed enteries in date column: 0
Number of missed enteries in direction column: 0
Number of missed enteries in internal column: 117
Number of missed enteries in operator_id column: 8172
Number of missed enteries in is_missed_call column: 0
Number of missed enteries in calls_count column: 0
Number of missed enteries in call_duration column: 0
Number of missed enteries in total_call_duration column: 0
```

There 2 columns that have some missing values and they should be approached differently.

In [6]:
```
dataset[dataset.internal.isna()].sample(10)
```

Out[6]:

	user_id	date	direction	internal	operator_id	is_missed_call	calls_count	call_duration	total_call_duration
39518	167747	2019-10-07 00:00:00+03:00	in	NaN	NaN	True	1	0	9
29887	167264	2019-11-15 00:00:00+03:00	in	NaN	919552.0	False	1	125	158
29989	167272	2019-10-09 00:00:00+03:00	in	NaN	912684.0	False	1	123	175
38069	167650	2019-10-14 00:00:00+03:00	in	NaN	921318.0	False	1	136	145

	user_id	date	direction	internal	operator_id	is_missed_call	calls_count	call_duration	total_call_duration
43860	168018	2019-11-28 00:00:00+03:00	in	NaN	NaN	True	1	0	2
7523	166604	2019-10-31 00:00:00+03:00	in	NaN	NaN	True	1	0	5
51021	168253	2019-11-15 00:00:00+03:00	in	NaN	952948.0	False	2	61	63
52042	168361	2019-10-28 00:00:00+03:00	in	NaN	NaN	True	3	0	15
16180	166916	2019-10-01 00:00:00+03:00	in	NaN	906396.0	False	1	100	117
41379	167852	2019-10-23 00:00:00+03:00	in	NaN	NaN	True	1	0	31

In [7]:
```
dataset.internal.value_counts()
```

Out[7]:
```
False    47621
True      6164
Name: internal, dtype: int64
```

Because most of made calls weren't internal we can fill missing values in *internal* columns with **False** value.

In [8]:
```
dataset.internal = dataset.internal.fillna('False').astype(bool)
```

let's see how many different operators can be working for one user and how many users can one operator process.

In [9]:
```
dataset.groupby('operator_id')['user_id'].nunique().value_counts()
```

Out[9]:
```
1    1092
Name: user_id, dtype: int64
```

In [10]:
```
dataset.groupby('user_id')['operator_id'].nunique().value_counts()
```

Out[10]:
```
1     107
2      63
3      34
4      26
0      17
5      17
6      10
7       7
8       5
15      3
9       2
11      2
16      2
27      2
```

```
48      1
10      1
12      1
14      1
17      1
18      1
21      1
28      1
30      1
50      1
Name: operator_id, dtype: int64
```

So each operator works only with one user. But each user can have from 1 to 50 operators working for them.

I don't see any good way to fill the missing values in this project, therefore I'll nuke these rows to have more clean data.

In [11]:
```
#drop empty operator_id values
dataset = dataset[~dataset['operator_id'].isna()]
```

Check data for mistakes

Let's see if there are any calls that are written to be missed but to have call_duration more than 0.

In [12]:
```
display(dataset.query('call_duration >0 and is_missed_call == True').shape)
dataset.query('call_duration >0 and is_missed_call == True').sample(5)
```

```
(325, 9)
```

Out[12]:

	user_id	date	direction	internal	operator_id	is_missed_call	calls_count	call_duration	total_call_duration
7645	166604	2019-11-28 00:00:00+03:00	in	False	893402.0	True	1	48	71
38610	167653	2019-11-12 00:00:00+03:00	in	False	939708.0	True	1	1	6
22757	167071	2019-10-15 00:00:00+03:00	in	False	913942.0	True	4	1	40
16193	166916	2019-10-01 00:00:00+03:00	in	False	906396.0	True	1	81	127
6288	166541	2019-10-14 00:00:00+03:00	in	True	908958.0	True	1	1	21

So looks like there are 325 rows that are noted as missed calls by mistake. I think that I'll change them to not missed calls instead.

In [13]:
```
dataset.loc[(dataset['is_missed_call'] == True) & (dataset['call_duration'] > 0), 'is_missed_call'] = False
```

In [14]:
```
#check if there are any left
dataset.query('call_duration >0 and is_missed_call == True')
```

Out[14]:

user_id	date	direction	internal	operator_id	is_missed_call	calls_count	call_duration	total_call_duration

Everything is clear no calls that are marked as missed by mistake.

Check if there are any strange operators that will affect analysis

In [15]:
```
df = (dataset.groupby('operator_id').agg(calls_count=('calls_count','sum'),
                                          missed_calls_count=('is_missed_call','sum'))
      .sort_values('calls_count', ascending=False).reset_index()
)
df.head()
```

Out[15]:

	operator_id	calls_count	missed_calls_count
0	885876.0	66049	129
1	885890.0	66016	104
2	929428.0	24572	35
3	925922.0	22210	28
4	908640.0	16699	24

There is definetely one strange operator. I think it's voicemail. I'll drop it so it won't affect any analysis to come.

In [16]:
```
dataset = dataset[dataset['operator_id']!=df.iloc[0]['operator_id']]
```

Calculate average call duration and average total call duration.

Here I will calculate what was average call for one operator for particular user in particular direction (one row call duration divided by one row call count).

In [17]:
```
dataset['avg_call_duration'] = dataset['call_duration'] / dataset['calls_count']
dataset['avg_total_call_duration'] = dataset['total_call_duration'] / dataset['calls_count']
```

Calculate waiting time

In [18]:
```
dataset['avg_waiting_time'] = dataset['avg_total_call_duration'] - dataset['avg_call_duration']
```

In [19]:
```
dataset['waiting_time'] = dataset['total_call_duration'] - dataset['call_duration']
```

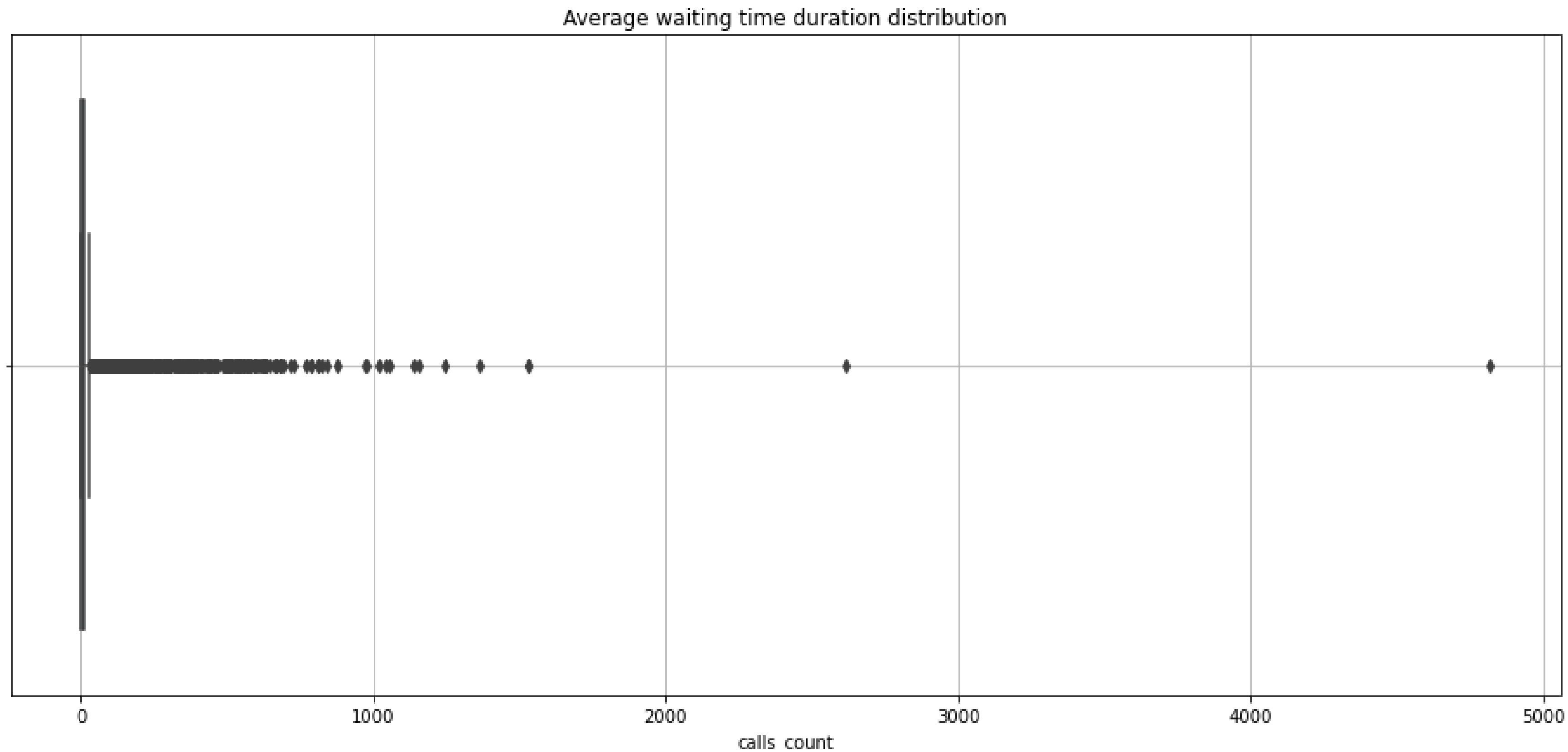
In [20]:
```
dataset.sample(10)
```

Out[20]:

	user_id	date	direction	internal	operator_id	is_missed_call	calls_count	call_duration	total_call_duration	avg_call_duration	a
13476	166782	2019-11-22 00:00:00+03:00	out	True	900180.0	True	2	0	0	0.000000	
13516	166782	2019-11-26 00:00:00+03:00	out	False	900180.0	False	19	1143	1454	60.157895	
35244	167521	2019-11-26 00:00:00+03:00	in	False	944228.0	False	3	609	828	203.000000	
44175	168021	2019-11-28 00:00:00+03:00	out	False	968150.0	True	4	0	70	0.000000	
865	166405	2019-09-04 00:00:00+03:00	in	False	882686.0	False	10	1827	1993	182.700000	
40187	167799	2019-11-13 00:00:00+03:00	out	False	925104.0	True	14	0	389	0.000000	
25939	167150	2019-09-27 00:00:00+03:00	out	True	905570.0	False	2	435	461	217.500000	
37429	167626	2019-10-13 00:00:00+03:00	out	False	919456.0	False	37	4115	4730	111.216216	
40016	167799	2019-10-01 00:00:00+03:00	out	False	925104.0	True	4	0	0	0.000000	
37824	167644	2019-10-25 00:00:00+03:00	out	False	924546.0	False	3	206	238	68.666667	

Check calls count column for outliers

In [21]:
```
fig, ax = plt.subplots(figsize=(16, 7))
ax.set_title('Average waiting time duration distribution')
sns.boxplot(data=dataset, x = 'calls_count', ax=ax)
plt.grid()
plt.show()
print ('99% of operators had less than {:.0f} calls in a given day'.format(
    dataset.calls_count.quantile(0.99)))
```




```
99% of operators had less than 139 calls in a given day
```

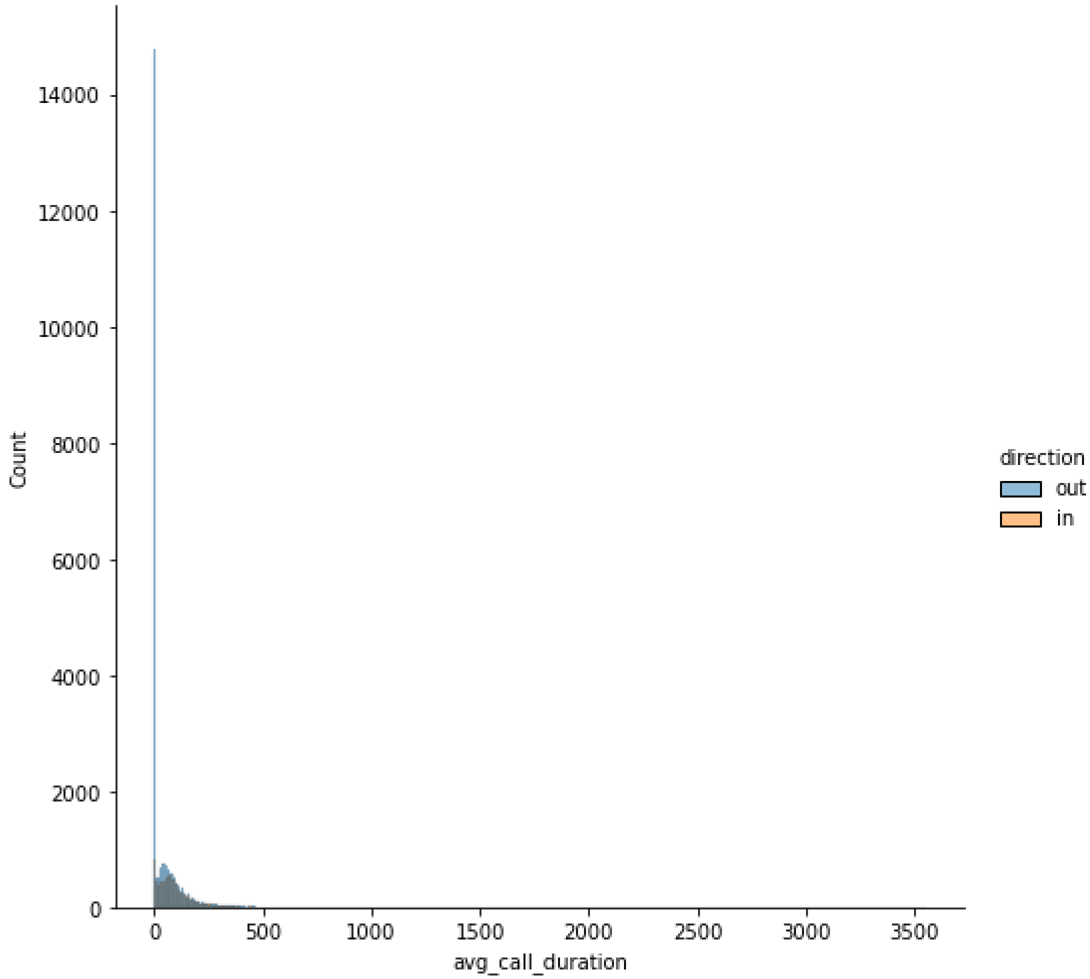
So I think that I'll drop all rows with more that 166 calls in a day, because that doesn't look real.

In [22]:
```
dataset = dataset.query('calls_count <= 166')
```

Exploratory Data Ananysis

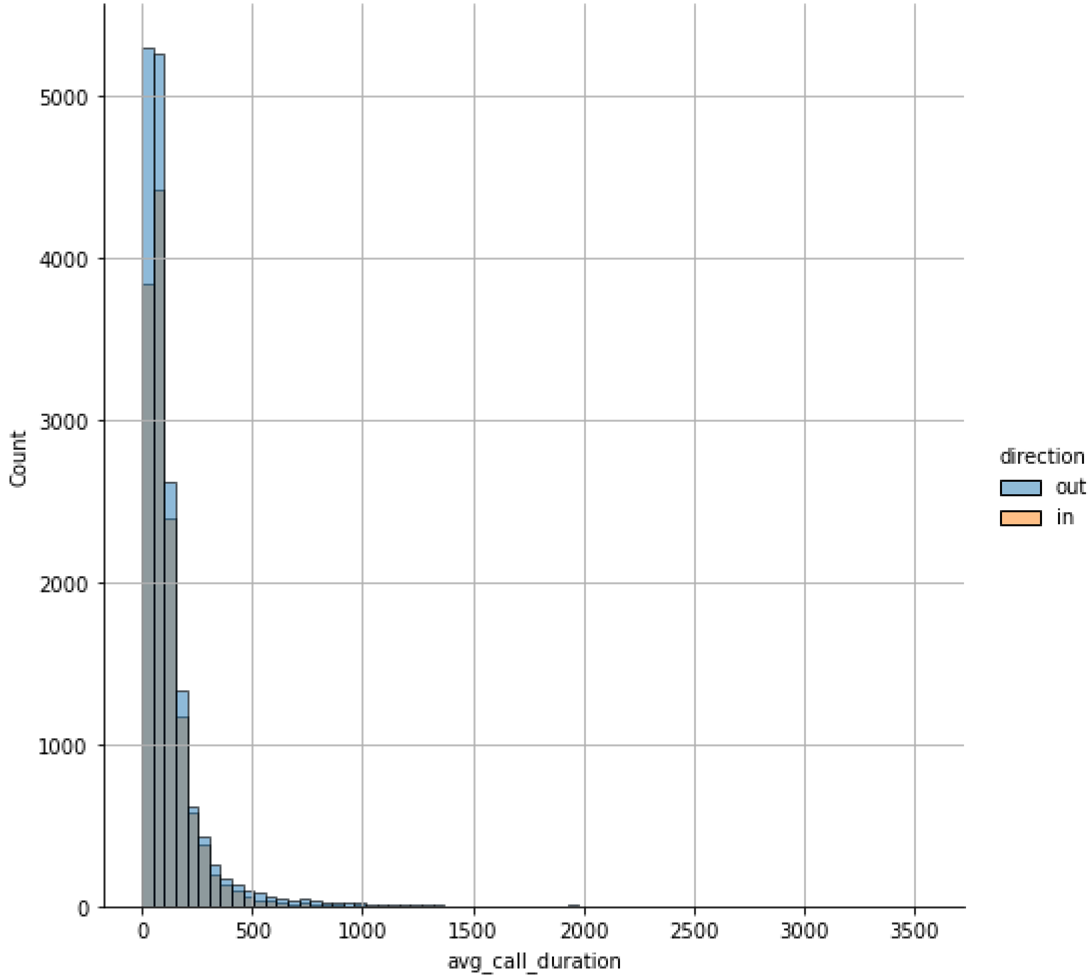
Have a look at call length distribution for incoming and outgoing calls

In [23]:
```
sns.displot(dataset, x='avg_call_duration', hue='direction', height=7)
plt.show()
```



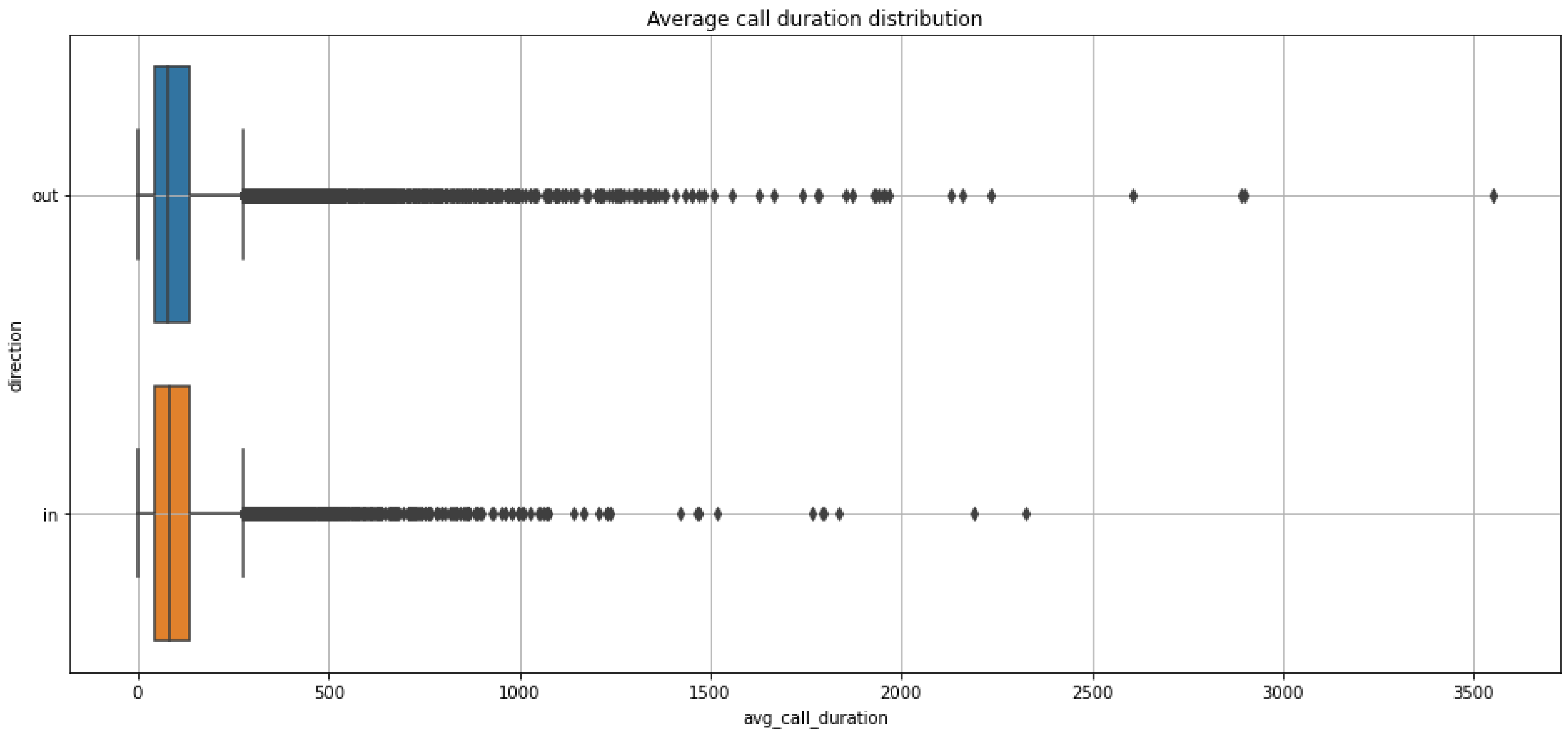
Nothing good here, let's look only at rows that aren't missed calls.

In [24]:
```
ax = sns.displot(dataset.query('is_missed_call == False'), x='avg_call_duration',
                 hue='direction', kind='hist',bins=70, height=7)
ax.set_titles('Average call duration histogram')
plt.grid()
plt.show()
```



I see here lots of outliers. Let's plot boxplot.

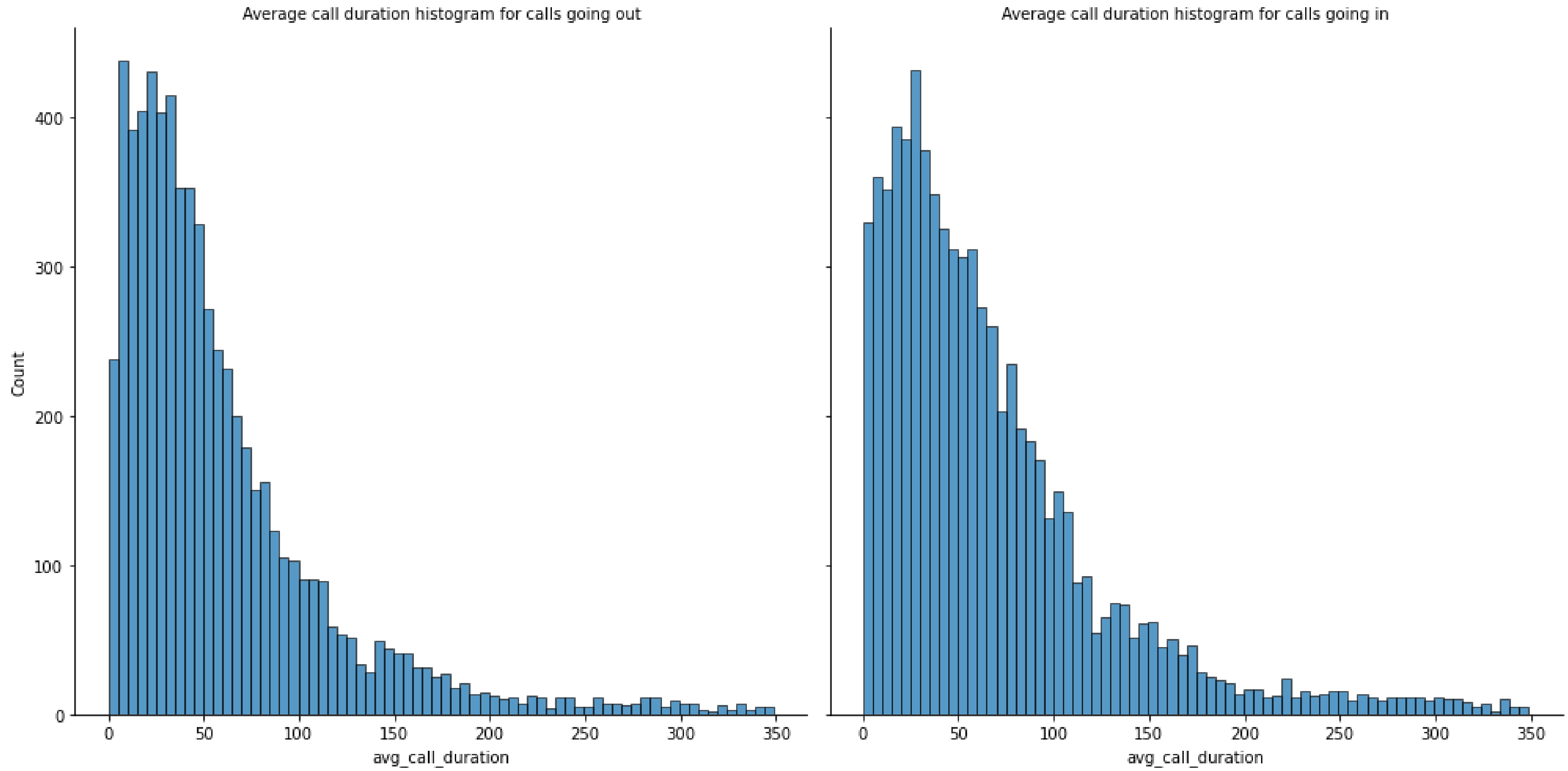
In [25]:
```
fig, ax = plt.subplots(figsize=(16, 7))
ax.set_title('Average call duration distribution')
sns.boxplot(data=dataset.query('is_missed_call == False'), x = 'avg_call_duration',y ='direction', ax=ax)
plt.grid()
plt.show()

print ('95% of calls were shorter than: {:.0f} seconds'.format(
    dataset.query('is_missed_call == False').avg_call_duration.quantile(0.95)))
print ('Average call duration for incoming calls: {:.0f} seconds'.format(
    dataset.query('is_missed_call == False and direction == "in"').avg_call_duration.mean()))
print ('Average call duration for outgoing calls: {:.0f} seconds'.format(
    dataset.query('is_missed_call == False and direction == "out"').avg_call_duration.mean()))
```




```
95% of calls were shorter than: 334 seconds
Average call duration for incoming calls: 111 seconds
Average call duration for outgoing calls: 121 seconds
```

There are many extreme outliers here. Let's take a look at distributions only for those call that had duration under 350 seconds.

In [26]:
```
ax = sns.displot(dataset.query('is_missed_call == False and call_duration <=350'),
                 x='avg_call_duration', col='direction', kind='hist',bins=70, height=7)
ax.set_titles('Average call duration histogram for calls going {col_name}')
plt.show()
```

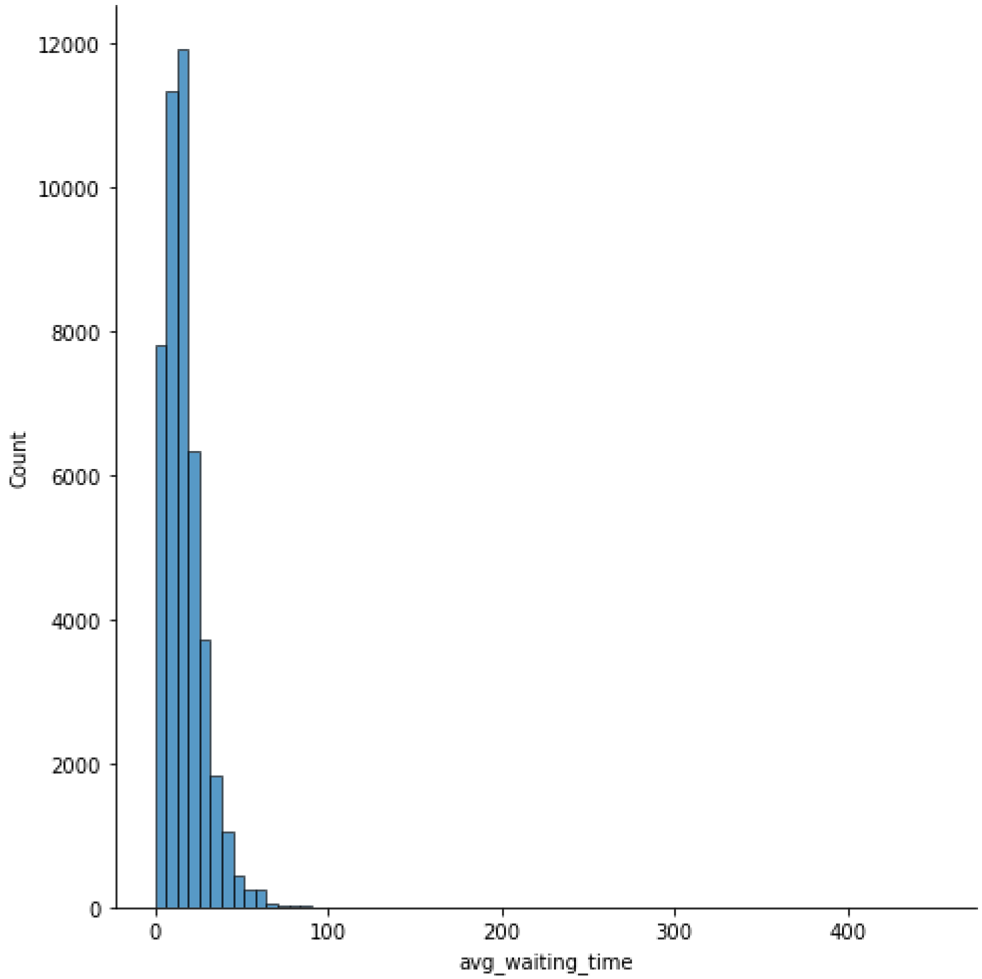


Few conclusions:

- call distribution has a positive skew;
- most of the calls are relatively short - under 2 minutes;
- distribution for incoming calls and outgoing is relatively simillar.

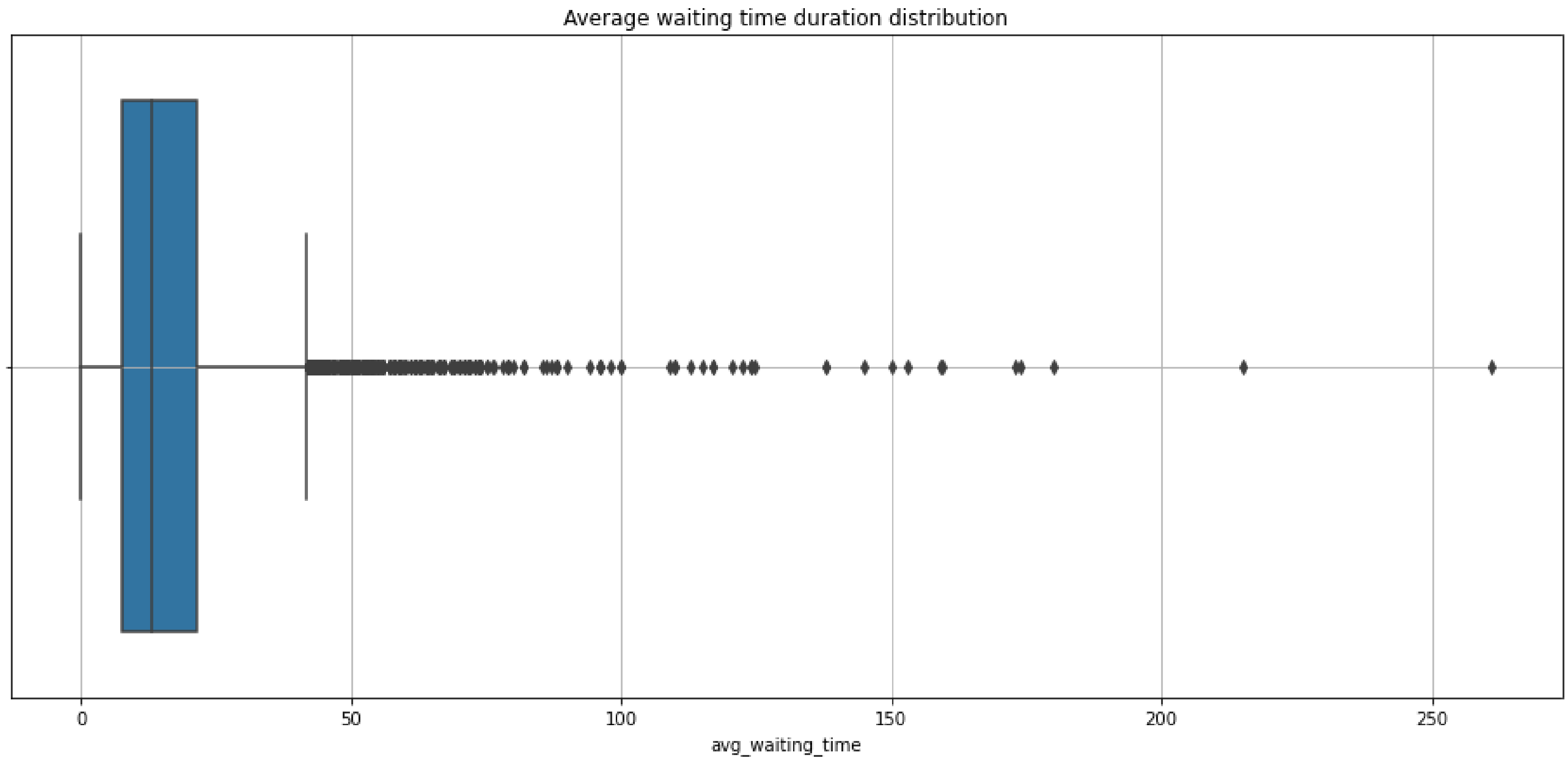
Now let's look at average waiting time distribution. We won't be interested in calls that are going out, because it long waiting time for calls going out doesn't characterize operator who made a call at all.

In [27]:
```
df = dataset.query('direction == "in" and is_missed_call == False')
sns.displot(dataset, x='avg_waiting_time', height=7, bins=70)
plt.show()
```



In [28]:

```
fig, ax = plt.subplots(figsize=(16, 7))
ax.set_title('Average waiting time duration distribution')
sns.boxplot(data=df, x = 'avg_waiting_time', ax=ax)
plt.grid()
plt.show()

print ('99% of calls had waiting time shorter than {:.0f} seconds'.format(
    dataset.query('is_missed_call == False').avg_waiting_time.quantile(0.99)))
print ('95% of calls had waiting time shorter than {:.0f} seconds'.format(
    dataset.query('is_missed_call == False').avg_waiting_time.quantile(0.95)))
print ('μ + 2σ is {:.1f} seconds'.format(np.mean(dataset.query('is_missed_call == False')['avg_waiting_time'])
    + 2*np.std(dataset.query('is_missed_call == False')['avg_waiting_time'])))
```




```
99% of calls had waiting time shorter than 50 seconds
95% of calls had waiting time shorter than 32 seconds
μ + 2σ is 35.4 seconds
```

So I suppose that we can consider operators that had incoming calls with average waiting time more than 50 seconds being extremelly ineffective, but concidiering the fact that one operator can receive more than one call from one user and here we have an average value, I'll use $\mu + 2\sigma$ as 35.4 seconds as a threshold for ineffectiveness of operator.

Have a look at number of missed calls calls with waiting time more than 35 seconds, grouped by operator

Create column for not missed incoming calls that had waiting time more than 35.4 seconds.

In [29]:
```
def f1(a,b,c):
    if a == False and b == 'in' and c >=35.4:
        return True
    else: return False
```

```
dataset['long_waiting'] = dataset.apply(lambda x: f1(x['is_missed_call'],x['direction'],x['avg_waiting_time']), axis=1)
dataset.long_waiting.value_counts()
```

Out[29]:

```
False    44093
True       978
Name: long_waiting, dtype: int64
```

There were 978 enteries with calls with avg waiting time more than 35.4 seconds.

In [30]:

```
dataset['is_outgoing'] = dataset['direction'] == 'out'
```

So let's use week as defying point for this analysis. I'll analyze how many calls does an operator miss per week, how many outgoing calls he makes per week and what is his average waiting time for incoming calls.

In [31]:

```
dataset['week'] = dataset.date.dt.week
```

In [32]:

```
#Group data by operator by week
df = dataset.groupby(['operator_id','week']).agg(missed_calls_count=('is_missed_call','sum'),
                                                   total_calls_count=('calls_count', 'sum'),
                                                   avg_waiting_time=('avg_waiting_time', 'mean'),
                                                   outgoing_count=('is_outgoing','sum')).reset_index()

df
```

Out[32]:

	operator_id	week	missed_calls_count	total_calls_count	avg_waiting_time	outgoing_count
0	879896.0	31	3	26	17.671875	7
1	879896.0	32	7	206	14.466142	17
2	879896.0	33	4	124	20.653935	10
3	879896.0	34	4	19	19.005556	7
4	879896.0	35	7	559	17.542951	13
...	...	...	...	...	...	...
5713	972410.0	48	2	77	18.882118	4
5714	972412.0	48	2	61	19.553322	4
5715	972460.0	48	3	70	10.134921	7
5716	973120.0	48	1	3	9.750000	2
5717	973286.0	48	0	2	44.000000	0

5718 rows × 6 columns

In [33]:
```
#now agroup data only by operator
operator_data = df.groupby('operator_id').agg(missed_calls_per_week=('missed_calls_count','mean'),
                                               total_calls_per_week=('total_calls_count', 'mean'),
                                               avg_waiting_time=('avg_waiting_time', 'mean'),
                                               outgoing_per_week=('outgoing_count','mean')).reset_index()
operator_data.head()
```

Out[33]:

	operator_id	missed_calls_per_week	total_calls_per_week	avg_waiting_time	outgoing_per_week
0	879896.0	3.333333	75.400000	14.433294	7.000000
1	879898.0	5.555556	443.000000	14.584750	10.388889
2	880020.0	1.166667	9.000000	7.579167	2.333333
3	880022.0	2.538462	16.846154	11.402814	5.230769
4	880026.0	5.529412	143.470588	11.763294	10.529412

In [34]:
```
print ('There are', operator_data.shape[0], 'operators.')
```

```
There are 1091 operators.
```

In [35]:
```
operator_data['operator_id'] = operator_data['operator_id'].astype(int)
operator_data.describe()
```

Out[35]:

	operator_id	missed_calls_per_week	total_calls_per_week	avg_waiting_time	outgoing_per_week
count	1091.000000	1091.000000	1091.000000	1091.000000	1091.000000
mean	925553.879010	2.133420	88.702376	16.507869	4.386216
std	22833.436523	2.071645	156.163005	8.156840	4.086124
min	879896.000000	0.000000	1.000000	0.000000	0.000000
25%	906395.000000	0.348485	3.801282	11.738556	1.000000
50%	925106.000000	1.500000	15.400000	15.600000	3.400000
75%	944213.000000	3.500000	94.950000	20.065912	7.309524
max	973286.000000	11.500000	1274.000000	62.000000	18.923077

It looks like there are also some outliers in the dataset. When 75% of users missed only 3.5 calls per week, there can't be users that miss 442 calls.

In [36]:
```
for col in operator_data.columns:
    print ('99% of values of column',col, 'are lower than: ',
           operator_data[col].quantile(0.98))
```

```
99% of values of column operator_id are lower than:  969289.2
99% of values of column missed_calls_per_week are lower than:  7.117460317460319
99% of values of column total_calls_per_week are lower than:  578.1833333333337
99% of values of column avg_waiting_time are lower than:  40.700000000000045
99% of values of column outgoing_per_week are lower than:  14.312820512820515
```

In [37]:
```
operator_data.sort_values('total_calls_per_week', ascending=False).head()
```

Out[37]:

	operator_id	missed_calls_per_week	total_calls_per_week	avg_waiting_time	outgoing_per_week
447	919364	5.750000	1274.000000	22.303542	10.750000
846	945286	5.500000	1132.000000	18.780367	11.000000
852	945302	6.166667	1061.000000	21.613826	13.166667
861	945322	5.500000	835.833333	21.256491	11.166667
461	919504	4.666667	738.000000	24.442664	9.000000

Looks okay to me, let's go on.

In [38]:
```
operator_data.head()
```

Out[38]:

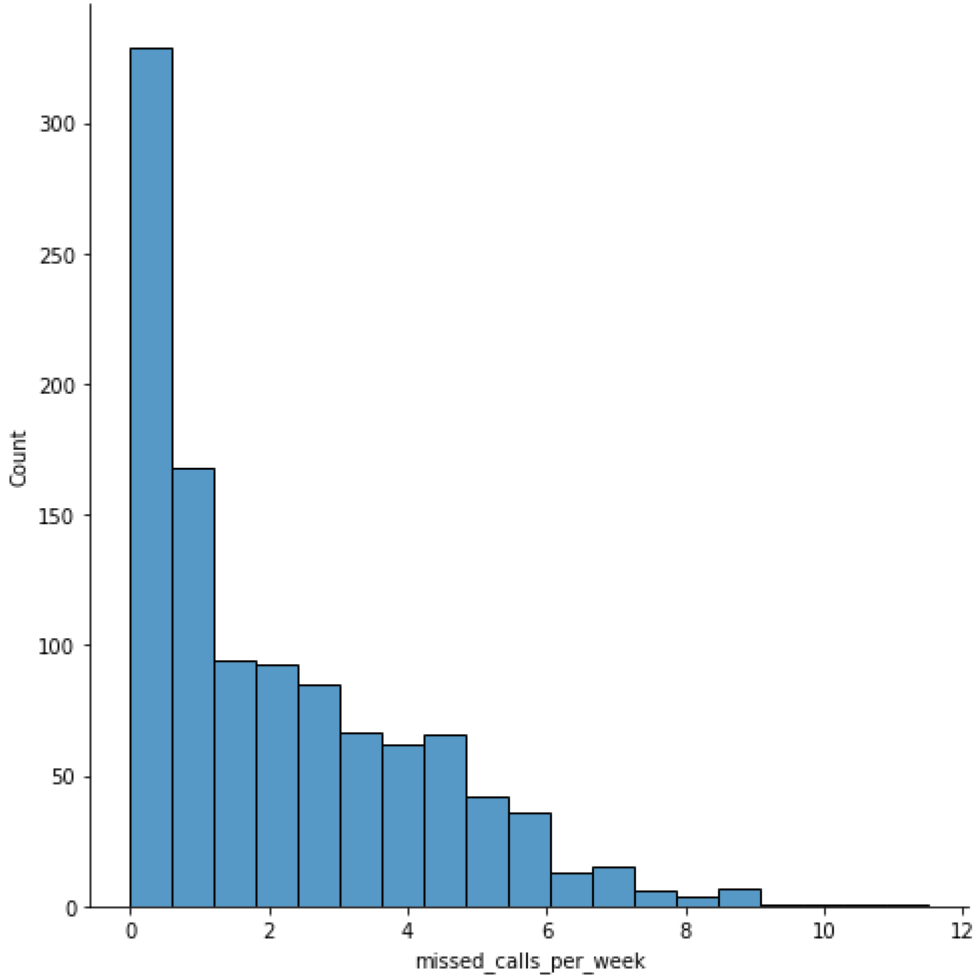
	operator_id	missed_calls_per_week	total_calls_per_week	avg_waiting_time	outgoing_per_week
0	879896	3.333333	75.400000	14.433294	7.000000
1	879898	5.555556	443.000000	14.584750	10.388889
2	880020	1.166667	9.000000	7.579167	2.333333
3	880022	2.538462	16.846154	11.402814	5.230769
4	880026	5.529412	143.470588	11.763294	10.529412

Let's also calculate ratio for missed to total amount of calls.

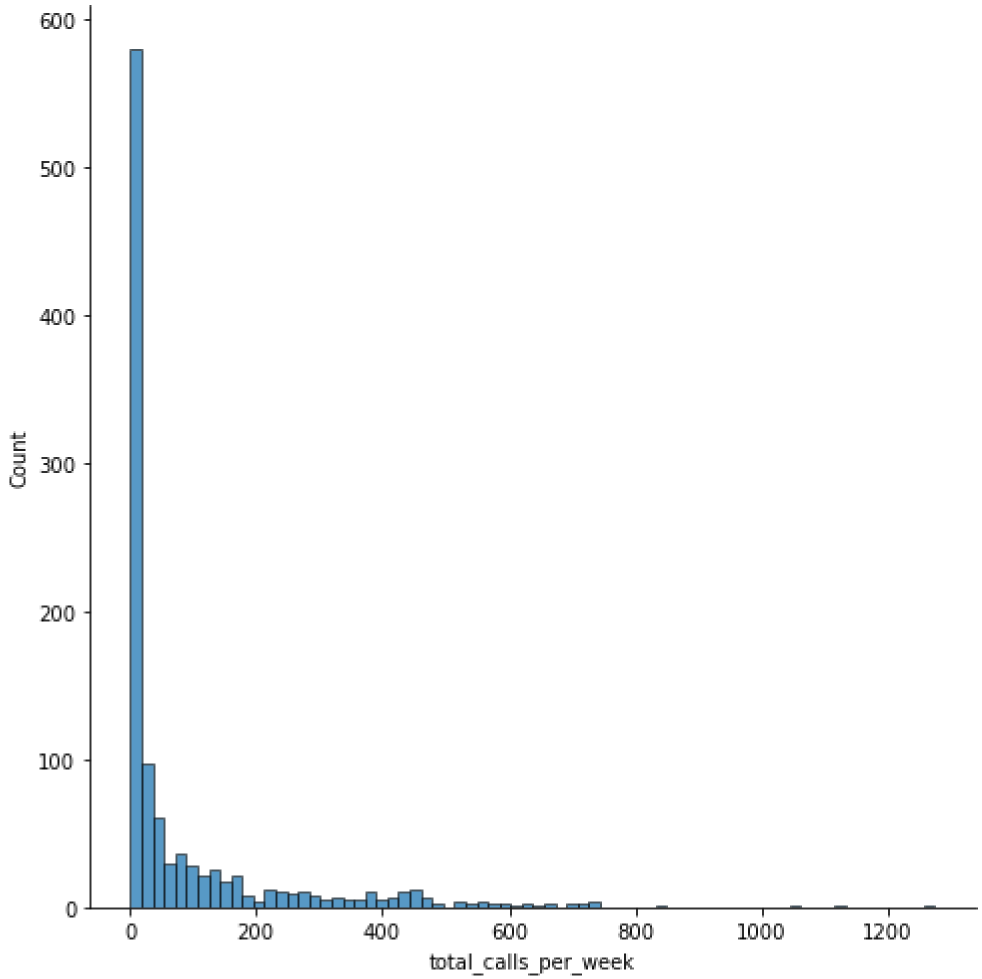
In [39]:
```
operator_data['missed_to_total'] = operator_data['missed_calls_per_week'] / operator_data['total_calls_per_week']
```

In [40]:

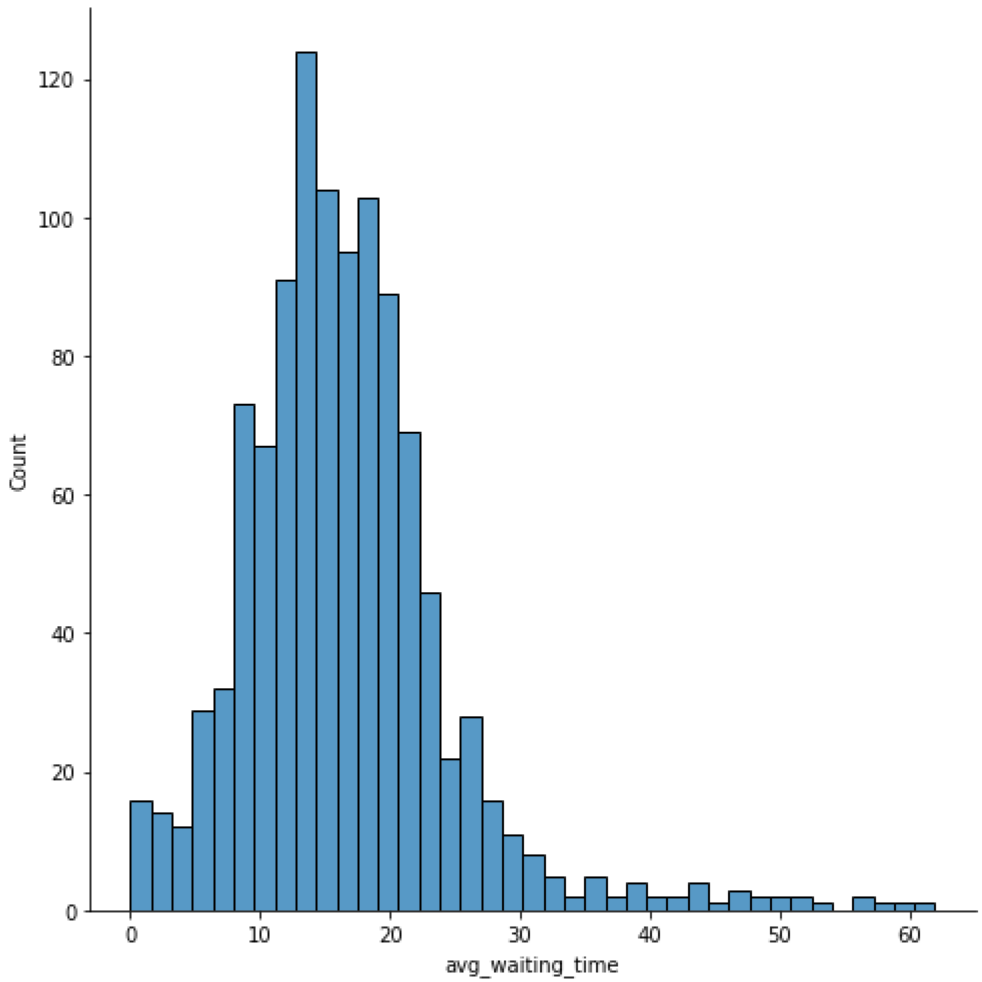
```
for col in operator_data.columns:
#     sns.boxplot(data=operator_data, x = col, ax=ax)
    if col == 'operator_id': continue
    else:
        g = sns.displot(data=operator_data, x=col, height=7)
        g.set_titles("{col_name} penguins")
        plt.show()
        if col == 'outgoing_per_week':
            print('Average value for column {}: {:.2f}'.format(col, operator_data
                                                                    .query('outgoing_per_week >=1')[col].mean()))
            print('Median value for column {}: {:.2f}'.format(col, operator_data
                                                                   .query('outgoing_per_week >=1')[col].median()))
            print('μ + 2σ value for column {}: {:.2f}'.format(col, np.mean(operator_data
                                                                             .query('outgoing_per_week >=1')[col])
                                                               + 2*np.std(operator_data.query('outgoing_per_week >=1')[col]
            print ('80% of values are higher than {:.3f}'.format(operator_data
                                                                      .query('outgoing_per_week >=1')[col].quantile(0.2)))
        elif col == 'total_calls_per_week':
            print('Average value for column {}: {:.2f}'.format(col, operator_data[col].mean()))
            print('Median value for column {}: {:.2f}'.format(col, operator_data[col].median()))
            print('μ + 2σ value for column {}: {:.2f}'.format(col, np.mean(operator_data[col])
                                                               + 2*np.std(operator_data[col])))
            print ('80% of values are higher than {:.3f}'.format(operator_data[col].quantile(0.2)))
        else:
            print('Average value for column {}: {:.2f}'.format(col, operator_data[col].mean()))
            print('Median value for column {}: {:.2f}'.format(col, operator_data[col].median()))
            print('μ + 2σ value for column {}: {:.2f}'.format(col, np.mean(operator_data[col])
                                                               + 2*np.std(operator_data[col])))
            print ('95% of values are smaller than {:.3f}'.format(operator_data[col].quantile(0.95)))
```



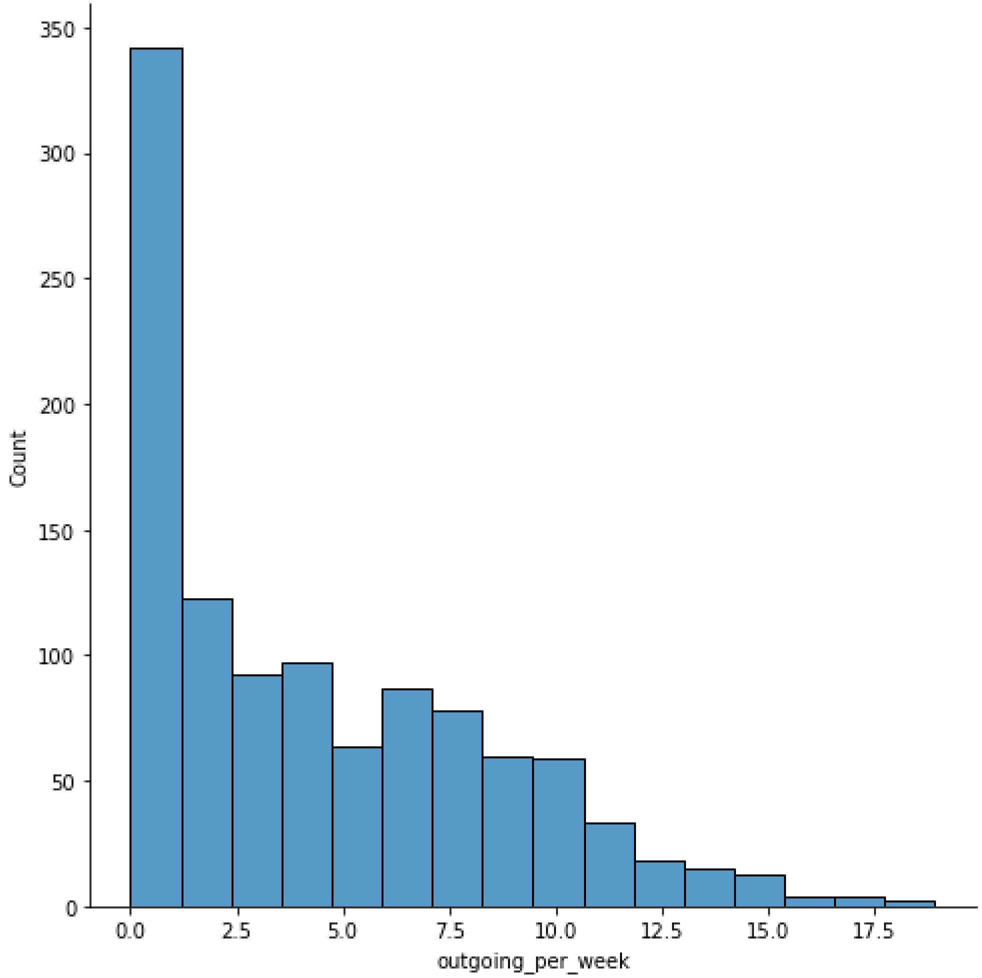

```
Average value for column missed_calls_per_week: 2.13
Median value for column missed_calls_per_week: 1.50
μ + 2σ value for column missed_calls_per_week: 6.27
95% of values are smaller than 6.000
```



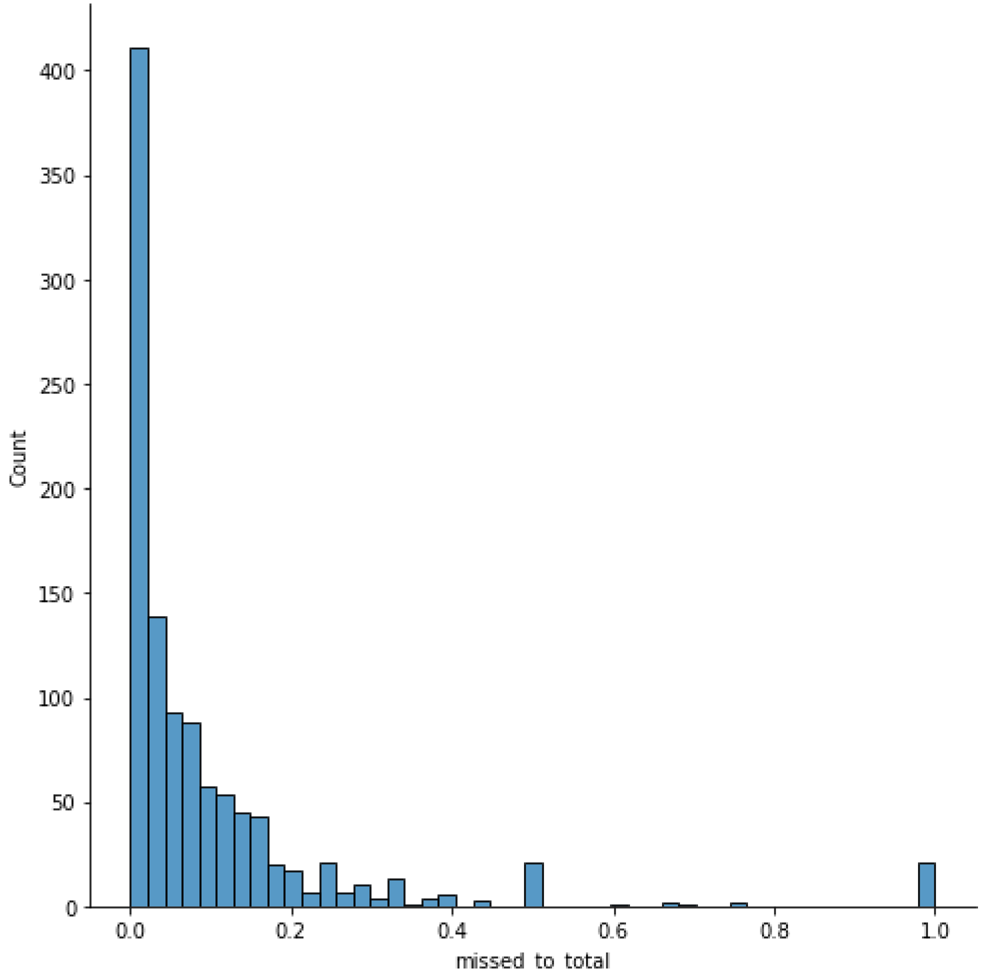

```
Average value for column total_calls_per_week: 88.70
Median value for column total_calls_per_week: 15.40
μ + 2σ value for column total_calls_per_week: 400.89
80% of values are higher than 3.000
```




```
Average value for column avg_waiting_time: 16.51
Median value for column avg_waiting_time: 15.60
μ + 2σ value for column avg_waiting_time: 32.81
95% of values are smaller than 29.716
```




```
Average value for column outgoing_per_week: 5.76
Median value for column outgoing_per_week: 5.00
μ + 2σ value for column outgoing_per_week: 13.31
80% of values are higher than 2.000
```




```
Average value for column missed_to_total: 0.10
Median value for column missed_to_total: 0.04
μ + 2σ value for column missed_to_total: 0.44
95% of values are smaller than 0.400
```

I will concider operators ineffective based on 2 sigma value. Except for outgoing calls. Here I will concider operators ineffeective only if they make not more than 2 calls in a week(are in lowest 20 percent). So operator is concidered ineffective if one of the folowing is true:

- Operator's waiting time for incoming calls is on average higher than 32 seconds;
- Operator has mised to total ratio higher than 0.44;
- Operator has missed 6 and more calls in a week;
- Operator has to make outgoing calls, but he/she made less than 2 calls in a week.

In [41]:
```
operator_data
```

Out[41]:

	operator_id	missed_calls_per_week	total_calls_per_week	avg_waiting_time	outgoing_per_week	missed_to_total
0	879896	3.333333	75.400000	14.433294	7.000000	0.044209
1	879898	5.555556	443.000000	14.584750	10.388889	0.012541
2	880020	1.166667	9.000000	7.579167	2.333333	0.129630
3	880022	2.538462	16.846154	11.402814	5.230769	0.150685
4	880026	5.529412	143.470588	11.763294	10.529412	0.038540
...	...	...	...	...	...	...
1086	972410	2.000000	77.000000	18.882118	4.000000	0.025974
1087	972412	2.000000	61.000000	19.553322	4.000000	0.032787
1088	972460	3.000000	70.000000	10.134921	7.000000	0.042857
1089	973120	1.000000	3.000000	9.750000	2.000000	0.333333
1090	973286	0.000000	2.000000	44.000000	0.000000	0.000000

1091 rows × 6 columns

In [42]:
```
filter_query = ("avg_waiting_time > 35 or missed_to_total > 0.44"
                "or missed_calls_per_week > 54 or outgoing_per_week >=1 and outgoing_per_week <=2")
ineffective_operators = (operator_data
                         .reset_index()
                         .query(filter_query)
                         .operator_id
                        )
ineffective_operators
```

Out[42]:
```
7       881278
9       882478
15      883018
16      883898
27      885682
         ...
1075    970244
1076    970250
1079    970258
1089    973120
```

```
1090    973286
Name: operator_id, Length: 218, dtype: int32
```

Conclusion

After performing analysis I have found out 218 ineffective operators who comply to these measures:

- Operator's waiting time for incoming calls is on average higher than 32 seconds;
- Operator has mised to total ratio higher than 0.44;
- Operator has missed 6 and more calls in a week;
- Operator has to make outgoing calls, but he/she made less than 2 calls in a week.

My cryteria were mostly based on value of 2 Sigma. I also think that these values should be rechecked once in a while because opearors' behavior may change.

Check statistical hypothesis

My goal here is to check if there is any difference in distribution of call duration, average waiting time and number of missed calls for users that are in different tariffs. I want to find out if operators may behave differently when they are working with users from different tariffs and if so, we should be using different treshold for operators of different tariffs.

Let's first group data for each user.

In [43]:
```
users_data = (dataset
              .groupby('user_id')[['avg_call_duration','avg_waiting_time','is_missed_call']]
              .agg(avg_call_duration=('avg_call_duration','mean'),
                   avg_waiting_time=('avg_waiting_time', 'mean'),
                                               missed_count=('is_missed_call','sum'))
              .reset_index()
             )
users_data
```

Out[43]:

	user_id	avg_call_duration	avg_waiting_time	missed_count
0	166377	57.073881	13.551207	232
1	166391	42.416667	24.416667	1
2	166392	175.572013	30.822642	0
3	166399	11.818182	15.409091	0

	user_id	avg_call_duration	avg_waiting_time	missed_count
4	166405	91.282783	19.412779	332
...	...	...	...	...
285	168583	41.313725	13.382353	0
286	168598	114.454545	8.194805	0
287	168601	61.828325	13.422988	21
288	168603	50.752381	20.714286	3
289	168606	241.275000	18.750000	3

290 rows × 4 columns

In [44]:
```
users_data_full = pd.merge(users_data, clients[['user_id','tariff_plan']],
                          how="left", on='user_id')
users_data_full
```

Out[44]:

	user_id	avg_call_duration	avg_waiting_time	missed_count	tariff_plan
0	166377	57.073881	13.551207	232	B
1	166391	42.416667	24.416667	1	C
2	166392	175.572013	30.822642	0	C
3	166399	11.818182	15.409091	0	C
4	166405	91.282783	19.412779	332	B
...	...	...	...	...	...
285	168583	41.313725	13.382353	0	B
286	168598	114.454545	8.194805	0	C
287	168601	61.828325	13.422988	21	C
288	168603	50.752381	20.714286	3	B
289	168606	241.275000	18.750000	3	C

290 rows × 5 columns

Let's see how tariffs are distributed.

In [45]:
```
#define color palette
colors = ['lightblue', 'mediumturquoise', 'orange', 'lightgreen', 'plum', 'bisque','lavender',
          'lightcyan','palevioletred']
df = users_data_full.tariff_plan.value_counts()
fig = go.Figure(data=[go.Pie(labels=df.index, values=df)],
                layout_title_text="Proportions of users with different tariffs")
fig.update_traces(textfont_size=15,
                  marker=dict(colors=colors, line=dict(color='#000000', width=1)),textinfo='value+percent')
fig.show()
```

It's important to notice that there aren't so many users in the dataset, therefore results can be not completely true.

There are much less users with tariff A, so we won't use them for testing. Also I'll even number of users, so there will be 101 users of tariff "B" and 101 users of tariff "C". Now let's see distributions of avg_call_duration, avg_waiting_time and number of missed calls among users.

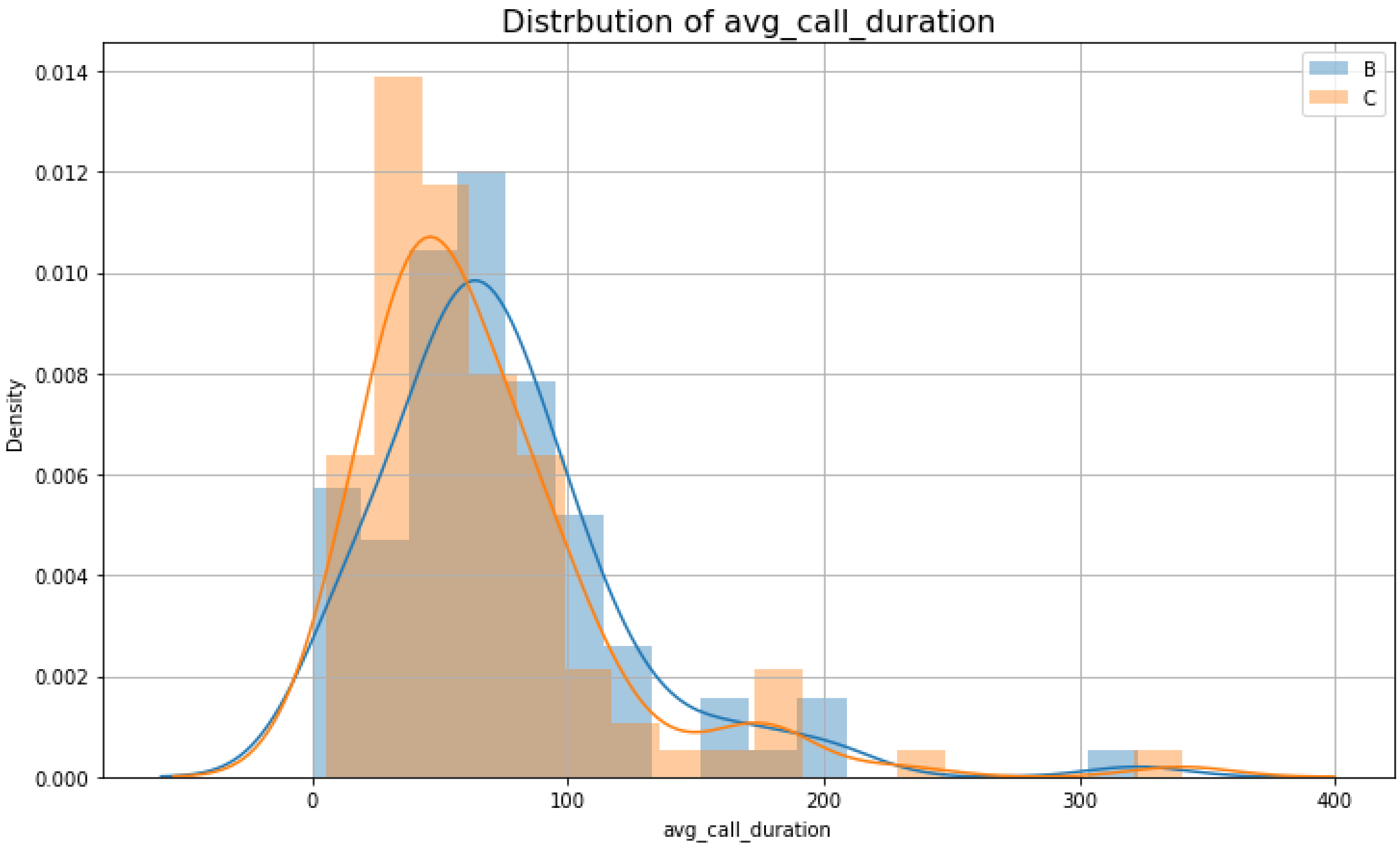
In [46]:
```
group_b = users_data_full.query('tariff_plan == "B"')[['avg_call_duration','avg_waiting_time','missed_count']]
group_c = (users_data_full.query('tariff_plan == "C"')[['avg_call_duration','avg_waiting_time','missed_count']]
           .sample(n=101, random_state=2))
display(group_b.describe())
display(group_c.describe())
```

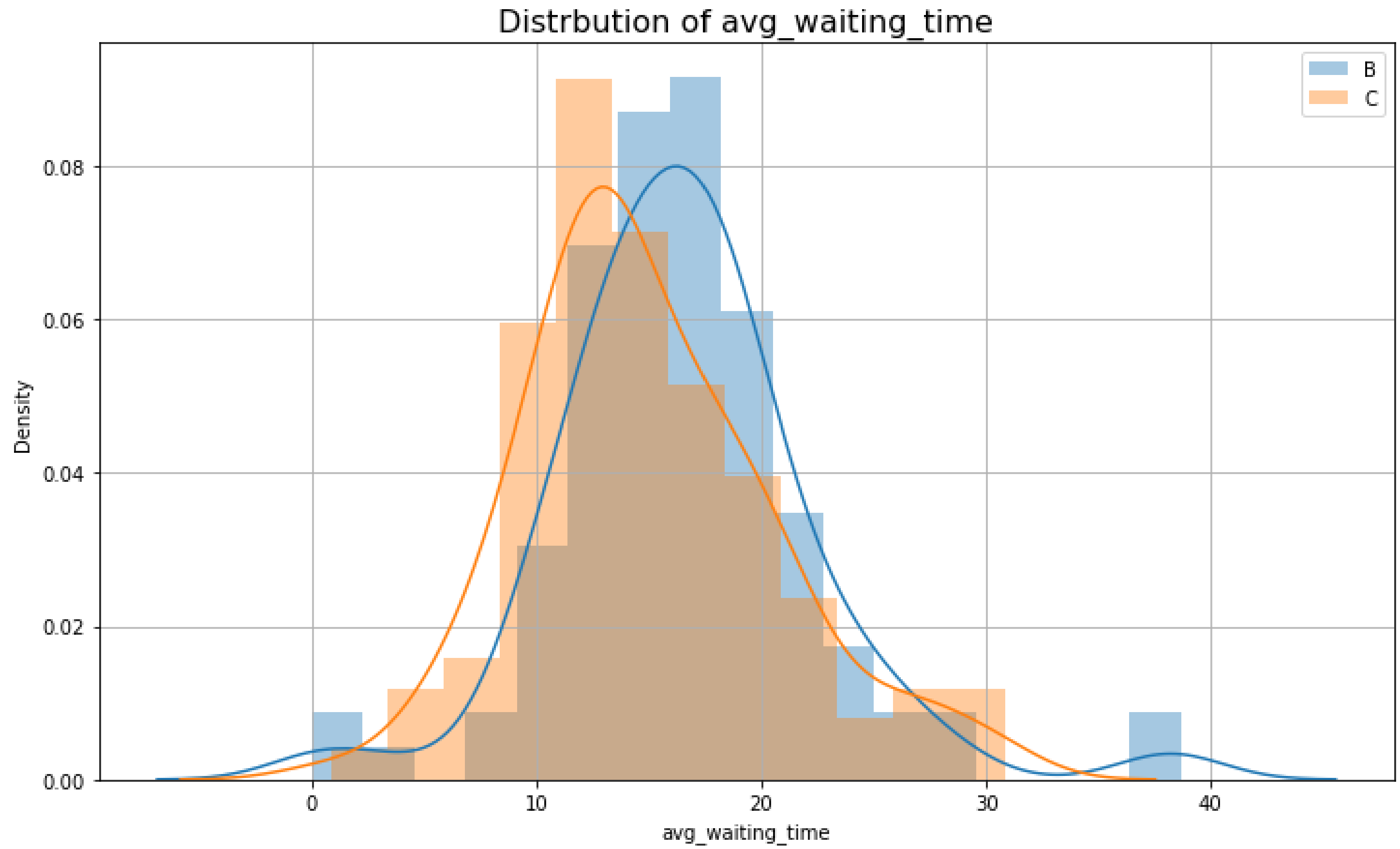
	avg_call_duration	avg_waiting_time	missed_count
count	101.000000	101.000000	101.000000
mean	73.656802	16.592659	52.980198
std	49.607202	5.776993	99.043120
min	0.000000	0.000000	0.000000
25%	45.722986	13.382353	4.000000
50%	65.911693	16.263836	20.000000
75%	91.282783	18.968128	57.000000
max	322.500000	38.666667	814.000000

	avg_call_duration	avg_waiting_time	missed_count
count	101.000000	101.000000	101.000000
mean	66.398527	14.953291	32.019802
std	50.285356	5.620194	60.946203
min	5.763543	0.857143	0.000000
25%	36.317797	11.686275	0.000000
50%	52.564103	13.550000	7.000000
75%	81.975274	17.949262	36.000000
max	339.857143	30.822642	350.000000

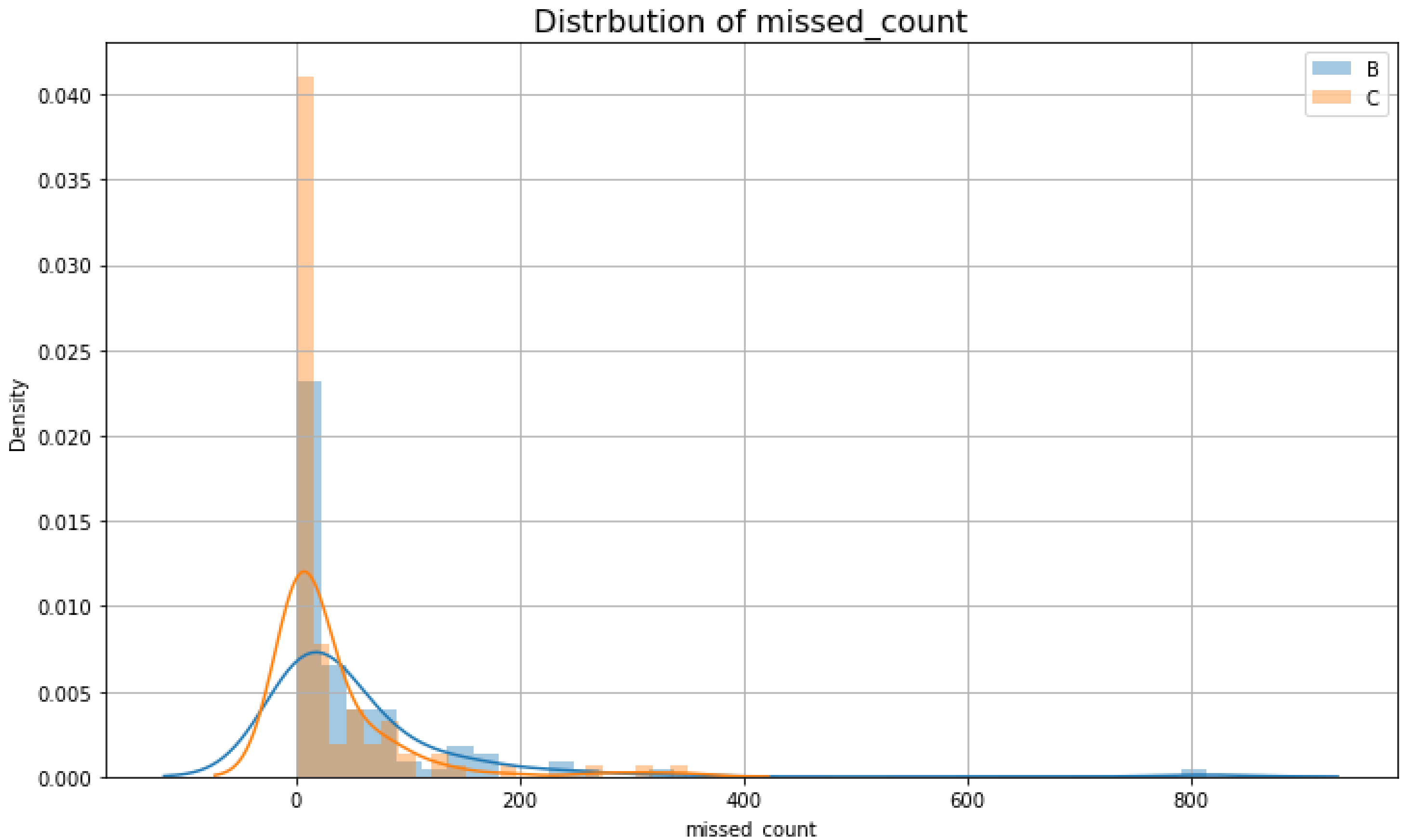
In [47]:

```
for col in group_b.columns:
    fig, ax = plt.subplots(figsize=(12, 7))
    ax.set_title(str('Distrbution of '+col),fontsize=16)
    sns.distplot(group_b[col], ax=ax, label="B", bins='auto')
    sns.distplot(group_c[col], ax=ax, label="C", bins='auto')
    plt.grid()
    plt.legend()
    plt.show()
```





Distrbution of avg_waiting_time
B
C
0.08
0.06
0.04
0.02
0.00
Density
0
10
20
30
40
avg_waiting_time



Here the data is not normally distributed, but populations have simillar shape. So I concider using Mann-Whitney test will give good results.

Null hypothesis: There is no statistically significant difference between average call duration, average waiting time and number of missed calls for users from group b and from group c.

Alternative hypothesis: There is statistically significant difference between average call duration, average waiting time and number of missed calls for users from group "B" and from group "C".

I'm going to use alpha = 0.05 as industry standart, but due to the fact that I'm going to perform 3 tests, I'm going to lower my alpha to avoid type 2 errors. Alpha will be 0.05/3 - 0.0167.

In [48]:
```
for col in group_b.columns:
    alpha = 0.05/3
    sample_a = group_b[col]
```

```
    sample_b = group_c[col]
    p_value = st.mannwhitneyu(sample_a, sample_b)[1]
    print("P-value for {} column: {:.10f}".format(col, p_value))
    print('Significance level:{:.3f}'.format(alpha))
    print('Mean value for {} column for group "B": {:.2f}'.format(col, sample_a.mean()))
    print('Mean value for {} column for group "C": {:.2f}'.format(col, sample_b.mean()))
    if (p_value < alpha):
        print("Reject H0 for",col, 'and tariffs "B" and "C"','\n')
    else:
        print("Fail to Reject H0 for", col,'and tariffs "B" and "C"','\n')
```

```
P-value for avg_call_duration column: 0.0394338618
Significance level:0.017
Mean value for avg_call_duration column for group "B": 73.66
Mean value for avg_call_duration column for group "C": 66.40
Fail to Reject H0 for avg_call_duration and tariffs "B" and "C"

P-value for avg_waiting_time column: 0.0054159550
Significance level:0.017
Mean value for avg_waiting_time column for group "B": 16.59
Mean value for avg_waiting_time column for group "C": 14.95
Reject H0 for avg_waiting_time and tariffs "B" and "C"

P-value for missed_count column: 0.0012159349
Significance level:0.017
Mean value for missed_count column for group "B": 52.98
Mean value for missed_count column for group "C": 32.02
Reject H0 for missed_count and tariffs "B" and "C"
```

Conclusion

There isn't any statistically significant difference in call duration for tariffs "B" and "C", but there is statistically is statisticalli significant difference in average waiting time and number of missed calls. Therefore we can't concider these populations definitely simillar.

Therefore I suggest to perform another analysis of operators perfomance based on tariffs of users, that they are working with. This may give us new results and different cryteria for selecting uneffective users with different tariffs.

General Conclusion

After analyzing data from call-center company I have discovered appropriate threshold that define if operator is effective or is not. Affective operators

- Don't have waiting time for incoming calls on average higher than 32 seconds;
- If operators have to make outgoing calls, they make more than 2 calls;
- Operators have missed to total calls ratio lower than 0.44;
- Don't miss more than 6 calls in a week.

It's important to notice that there were some parameters that weren't deeply looked into, like if does the fact that operator has to perform internal calls affect his effectiveness, or if operators that are only recieving calls on average more effective. One paramter that has been looked at is performance for users of different tariffs. And based on this analysis there is some statistically important difference in several parameters for users of different tariffs. So it may make sense to analyze operators that work with different types of clients separately. This may give us different result in the end.

Used literature and articles

1. Mann-Whitney tutorial - to determine if the data fits requirements for Mann-Whitney test.
2. How to Evaluate the Effectiveness of Your Call Center - I used this article to determine the best metrics to evaluate a call center.
3. Call Center Dashoboard - I used this article to determine what will be the best dashboard for call center performance analysis.
4. When not to use machine learning - I used this article to determine if I need to use ML algorithms in this project.
5. 5 Tips For Increasing Call Center Agent Productivity - I used this article to give better advices on how to increase operators efficiency.

Presentation link

Final Project Presentation

Dashboard link

Dashboard Link