4. Project YP

February 3, 2022

1 Project 4

2 Determination of a prospective tariff for a telecom company

Project description

Clients are offered two tariff plans: “Smart” and “Ultra”. To adjust the advertising budget, the commercial department wants to understand which tariff brings in more money.

Task

It is necessary to analyze the behavior of customers and draw a conclusion - which tariff is better.

Data descripcion

Table users (information about users):

```
user_id - unique user ID
first_name - user first name
last_name - user last name
age - user age (years)
reg_date - tariff activation date (day, month, year)
churn_date - date of termination of using the tariff (if the value is omitted, then the tariff
city - user's city of residence
tarif - tariff plan name
```

The calls table (information about calls):

```
id - unique call ID
call_date - call date
duration - call duration in minutes
user_id - ID of the user who made the call
```

Messages table (message information):

```
id - unique message ID
message_date - message date
user_id - ID of the user who sent the message
```

internet table (information about internet sessions)

```
id - unique session ID
mb_used - the amount of Internet traffic spent per session (in megabytes)
session_date - internet session date
```

user_id - unique user ID

Tariffs table (tariff information):

tariff_name - tariff name
rub_monthly_fee - monthly subscription fee in rubles
minutes_included - the number of minutes of conversation per month included in the subscriptio
messages_included - number of messages per month included in the subscription fee
mb_per_month_included - the amount of Internet traffic included in the subscription fee (in me
rub_per_minute - the cost of a minute of conversation in excess of the tariff package (for exa
rub_per_message - the cost of sending a message in excess of the tariff package
rub_per_gb - the cost of an additional gigabyte of Internet traffic in excess of the tariff pa

2.1 Let's see the data from the file

Import the necessary libraries and see the tables

[2]:
```
from scipy import stats as st
import numpy as np
import pandas as pd
import math
import seaborn as sns
import matplotlib.pyplot as plt


df_calls = pd.read_csv('./calls.csv')
df_internet = pd.read_csv('./internet.csv')
df_messages = pd.read_csv('./messages.csv')
df_tariffs = pd.read_csv('./tariffs.csv')
df_users = pd.read_csv('./users.csv')

df_all = [df_calls, df_internet, df_messages, df_tariffs, df_users]

for i in df_all:
    print(i.info())
    display(i.head())
    print('*' * 100)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202607 entries, 0 to 202606
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   id         202607 non-null  object
 1   call_date  202607 non-null  object
 2   duration   202607 non-null  float64
 3   user_id    202607 non-null  int64
dtypes: float64(1), int64(1), object(2)
```

```
memory usage: 6.2+ MB
None
```

	id	call_date	duration	user_id
0	1000_0	2018-07-25	0.00	1000
1	1000_1	2018-08-17	0.00	1000
2	1000_2	2018-06-11	2.85	1000
3	1000_3	2018-09-21	13.80	1000
4	1000_4	2018-12-15	5.18	1000

```
**************************************************************************************
********************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149396 entries, 0 to 149395
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Unnamed: 0    149396 non-null  int64
 1   id            149396 non-null  object
 2   mb_used       149396 non-null  float64
 3   session_date  149396 non-null  object
 4   user_id       149396 non-null  int64
dtypes: float64(1), int64(2), object(2)
memory usage: 5.7+ MB
None
```

	Unnamed: 0	id	mb_used	session_date	user_id
0	0	1000_0	112.95	2018-11-25	1000
1	1	1000_1	1052.81	2018-09-07	1000
2	2	1000_2	1197.26	2018-06-25	1000
3	3	1000_3	550.27	2018-08-22	1000
4	4	1000_4	302.56	2018-09-24	1000

```
**************************************************************************************
********************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 123036 entries, 0 to 123035
Data columns (total 3 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            123036 non-null  object
 1   message_date  123036 non-null  object
 2   user_id       123036 non-null  int64
dtypes: int64(1), object(2)
memory usage: 2.8+ MB
None
```

	id	message_date	user_id
0	1000_0	2018-06-27	1000
1	1000_1	2018-10-08	1000

```
2  1000_2   2018-08-04     1000
3  1000_3   2018-06-16     1000
4  1000_4   2018-12-05     1000

********************************************************************************
********************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   messages_included      2 non-null      int64
 1   mb_per_month_included  2 non-null      int64
 2   minutes_included       2 non-null      int64
 3   rub_monthly_fee        2 non-null      int64
 4   rub_per_gb             2 non-null      int64
 5   rub_per_message        2 non-null      int64
 6   rub_per_minute         2 non-null      int64
 7   tariff_name            2 non-null      object
dtypes: int64(7), object(1)
memory usage: 256.0+ bytes
None

   messages_included  mb_per_month_included  minutes_included  \
0                 50                  15360               500
1               1000                  30720              3000

   rub_monthly_fee  rub_per_gb  rub_per_message  rub_per_minute tariff_name
0              550         200                3               3       smart
1             1950         150                1               1       ultra

********************************************************************************
********************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user_id     500 non-null    int64
 1   age         500 non-null    int64
 2   churn_date  38 non-null     object
 3   city        500 non-null    object
 4   first_name  500 non-null    object
 5   last_name   500 non-null    object
 6   reg_date    500 non-null    object
 7   tariff      500 non-null    object
dtypes: int64(2), object(6)
memory usage: 31.4+ KB
None
```

```
   user_id  age churn_date         city first_name  last_name    reg_date  \
0     1000   52        NaN                          2018-05-25
1     1001   41        NaN                               2018-11-01
2     1002   59        NaN                         2018-06-17
3     1003   23        NaN                             2018-08-17
4     1004   68        NaN                          2018-05-14

  tariff
0  ultra
1  smart
2  smart
3  ultra
4  ultra
```

```
********************************************************************************
********************
```

Let's see if there are empty values in the tables

[3]:
```
for i in df_all:
    print(i.isna().mean())
    print('*' * 100)
```

```
id           0.0
call_date    0.0
duration     0.0
user_id      0.0
dtype: float64
********************************************************************************
********************
Unnamed: 0      0.0
id              0.0
mb_used         0.0
session_date    0.0
user_id         0.0
dtype: float64
********************************************************************************
********************
id              0.0
message_date    0.0
user_id         0.0
dtype: float64
********************************************************************************
********************
messages_included        0.0
mb_per_month_included    0.0
minutes_included         0.0
rub_monthly_fee          0.0
rub_per_gb               0.0
```

```
rub_per_message          0.0
rub_per_minute           0.0
tariff_name              0.0
dtype: float64
********************************************************************************
********************
user_id       0.000
age           0.000
churn_date    0.924
city          0.000
first_name    0.000
last_name     0.000
reg_date      0.000
tariff        0.000
dtype: float64
********************************************************************************
********************
```

Empty values only in the users table, in the churn_date column. This means that subscribers are still active. Until we change this column

Let's see if there are duplicates

[4]:
```
for i in df_all:
    print(i.duplicated().sum())
```

```
0
0
0
0
0
```

There aren't any duplicates

2.1.1 Conclusion

So we have 5 tables. No empty values and no duplicates. Next, we move on to data preprocessing.

There are null values in the df_calls and df_internet tables. This is not a mistake: zeros indicate missed calls and unsuccessful internet connection, so they do not need to be deleted.

2.2 Data preprocessing

Change the column name in the df_tariffs table

[5]:
```
df_tariffs = df_tariffs.rename(columns={'tariff_name': 'tariff'})
```

Because operator always rounds each call to minutes (upwards), then we will reflect this in df_calls

[6]:
```
df_calls['duration'] = (df_calls['duration']).apply(math.ceil)
```

Let's convert the data to the required types

[7]:
```
#1. df_calls
df_calls['call_date'] = pd.to_datetime(df_calls['call_date'], format='%Y-%m-%d')
df_calls['duration'] = df_calls['duration'].astype(int)

#2. df_internet
df_internet['session_date'] = pd.to_datetime(df_internet['session_date'],␣
↪format='%Y-%m-%d')

#3. df_messages
df_messages['message_date'] = pd.to_datetime(df_messages['message_date'],␣
↪format='%Y-%m-%d')

#5. df_users
df_users['reg_date'] = pd.to_datetime(df_users['reg_date'], format='%Y-%m-%d')
```

Find the number of calls made and spent minutes of conversation by month

First, let's add a new month column to the df_calls table.

[8]:
```
df_calls['month'] = df_calls['call_date'].dt.month
```

Add a column with a tariff to the table with calls

[9]:
```
df_calls = df_calls.merge(df_users[['user_id', 'tariff']], on='user_id')
```

Let's make a summary table by the number and duration of calls

[10]:
```
df_calls_income = df_calls.pivot_table(index=['tariff', 'user_id', 'month'],␣
↪values='duration', aggfunc=['count', 'sum']).reset_index()
df_calls_income.head()
```

[10]:

	tariff	user_id	month	count	sum
				duration	duration
0	smart	1001	11	59	430
1	smart	1001	12	63	414
2	smart	1002	6	15	117
3	smart	1002	7	26	214
4	smart	1002	8	42	289

Change column names

[11]:
```
df_calls_income.columns = ['tariff', 'user_id', 'month', 'duration_count',␣
↪'duration_sum']
```

Find the number of sent messages by month

First, let's add a new month column to the df_messages table.

[12]:
```
df_messages['month'] = df_messages['message_date'].dt.month
```

Add a column with a tariff to the table with calls

[13]:
```
df_messages = df_messages.merge(df_users[['user_id', 'tariff']], on='user_id')
```

Let's make a pivot table by the number of SMS

[14]:
```
df_messages_income = df_messages.pivot_table(index=['tariff', 'user_id',␣
↪'month'], values='message_date', aggfunc='count').reset_index()
df_messages_income.head()
```

[14]:

	tariff	user_id	month	message_date
0	smart	1002	6	4
1	smart	1002	7	11
2	smart	1002	8	13
3	smart	1002	9	4
4	smart	1002	10	10

Find the amount of Internet traffic used by months

First, let's add a new month column to the df_internet table.

[15]:
```
df_internet['month'] = df_internet['session_date'].dt.month
```

Add a column with a tariff to the table with calls

[16]:
```
df_internet = df_internet.merge(df_users[['user_id', 'tariff']], on='user_id')
```

Let's make a pivot table on the amount of traffic

[17]:
```
df_internet_income = df_internet.pivot_table(index=['tariff', 'user_id',␣
↪'month'], values='mb_used', aggfunc='sum').reset_index()
df_internet_income.head()
```

[17]:

	tariff	user_id	month	mb_used
0	smart	1001	11	18429.34
1	smart	1001	12	14036.66
2	smart	1002	6	10856.82
3	smart	1002	7	17580.10
4	smart	1002	8	20319.26

Combine the pivots into one table. Let's take a table with calls as a basis

[18]:
```
df_income = df_calls_income
df_income = df_income.merge(df_messages_income, on=['tariff', 'user_id',␣
↪'month'], how='outer')
df_income = df_income.merge(df_internet_income, on=['tariff', 'user_id',␣
↪'month'], how='outer')
df_income.head()
```

[18]:

	tariff	user_id	month	duration_count	duration_sum	message_date	mb_used
0	smart	1001	11	59.0	430.0	NaN	18429.34
1	smart	1001	12	63.0	414.0	NaN	14036.66
2	smart	1002	6	15.0	117.0	4.0	10856.82
3	smart	1002	7	26.0	214.0	11.0	17580.10
4	smart	1002	8	42.0	289.0	13.0	20319.26

Let's check the empty values

[19]:
```
df_income.isna().sum()
```

[19]:
```
tariff              0
user_id             0
month               0
duration_count     40
duration_sum       40
message_date      497
mb_used            11
dtype: int64
```

Rename the message_date column to message_count and change all dates to 1 and empty values to 0

[20]:
```
df_income = df_income.rename(columns={'message_date': 'message_count'})

df_income['message_count'] = df_income['message_count'].fillna(0)

df_income.loc[(df_income['message_count'] != 0), 'message_count'] = 1
df_income['message_count'] = df_income['message_count'].astype(int)
df_income.head()
```

[20]:

	tariff	user_id	month	duration_count	duration_sum	message_count	\
0	smart	1001	11	59.0	430.0	0	
1	smart	1001	12	63.0	414.0	0	
2	smart	1002	6	15.0	117.0	1	
3	smart	1002	7	26.0	214.0	1	
4	smart	1002	8	42.0	289.0	1	

	mb_used
0	18429.34
1	14036.66
2	10856.82
3	17580.10
4	20319.26

Change gaps to 0 for columns duration_count, duration_sum, mb_used

[21]:
```
df_income['duration_count'] = df_income['duration_count'].fillna(0)
df_income['duration_sum'] = df_income['duration_sum'].fillna(0)
df_income['mb_used'] = df_income['mb_used'].fillna(0)
```

Let's check the empty values

[22]:
```
df_income.isna().sum()
```

[22]:
```
tariff            0
user_id           0
month             0
duration_count    0
duration_sum      0
message_count     0
mb_used           0
dtype: int64
```

Based on the tariff, add the necessary info from the df_tariffs table

[23]:
```
df_income = df_income.merge(df_tariffs, on='tariff')
df_income.head()
```

[23]:

	tariff	user_id	month	duration_count	duration_sum	message_count	\
0	smart	1001	11	59.0	430.0	0	
1	smart	1001	12	63.0	414.0	0	
2	smart	1002	6	15.0	117.0	1	
3	smart	1002	7	26.0	214.0	1	
4	smart	1002	8	42.0	289.0	1	

	mb_used	messages_included	mb_per_month_included	minutes_included	\
0	18429.34	50	15360	500	
1	14036.66	50	15360	500	
2	10856.82	50	15360	500	
3	17580.10	50	15360	500	
4	20319.26	50	15360	500	

	rub_monthly_fee	rub_per_gb	rub_per_message	rub_per_minute
0	550	200	3	3
1	550	200	3	3
2	550	200	3	3
3	550	200	3	3
4	550	200	3	3

Find the monthly revenue from each user

First, find the number of calls above the norm and add them to a new column

[24]:
```
df_income['duration_more_tarrif'] = df_income['duration_sum'] -␣
↪df_income['minutes_included']
df_income.loc[(df_income['duration_more_tarrif'] < 0), 'duration_more_tarrif']␣
↪= 0
```

Find the number of SMS over the norm and add them to a new column

[25]:
```
df_income['messages_more_tarrif'] = df_income['message_count'] -␣
↪df_income['messages_included']
df_income.loc[(df_income['messages_more_tarrif'] < 0), 'messages_more_tarrif']␣
↪= 0
```

Now with traffic and also add it to a new column

[26]:
```
df_income['internet_more_tarrif'] = df_income['mb_used'] -␣
↪df_income['mb_per_month_included']
df_income.loc[(df_income['internet_more_tarrif'] < 0), 'internet_more_tarrif']␣
↪= 0
```

[27]:
```
df_income.head()
```

[27]:

	tariff	user_id	month	duration_count	duration_sum	message_count	\
0	smart	1001	11	59.0	430.0	0	
1	smart	1001	12	63.0	414.0	0	
2	smart	1002	6	15.0	117.0	1	
3	smart	1002	7	26.0	214.0	1	
4	smart	1002	8	42.0	289.0	1	

	mb_used	messages_included	mb_per_month_included	minutes_included	\
0	18429.34	50	15360	500	
1	14036.66	50	15360	500	
2	10856.82	50	15360	500	
3	17580.10	50	15360	500	
4	20319.26	50	15360	500	

	rub_monthly_fee	rub_per_gb	rub_per_message	rub_per_minute	\
0	550	200	3	3	
1	550	200	3	3	
2	550	200	3	3	
3	550	200	3	3	
4	550	200	3	3	

	duration_more_tarrif	messages_more_tarrif	internet_more_tarrif
0	0.0	0	3069.34
1	0.0	0	0.00
2	0.0	0	0.00
3	0.0	0	2220.10
4	0.0	0	4959.26

Let's calculate the revenue. Let's multiply minutes, sms and excess traffic by the tariff. Let's convert the traffic from mb to GB and round up to a higher value. Put everything together and add the subscription fee. The result will be the total revenue in the new column

[28]:
```
df_income['income_total'] = (df_income['duration_more_tarrif'] *␣
↪df_income['rub_per_minute']) + (df_income['messages_more_tarrif'] *␣
↪df_income['rub_per_message']) + (((df_income['internet_more_tarrif'] / 1024).
↪apply(math.ceil)) * df_income['rub_per_gb']) + df_income['rub_monthly_fee']
df_income.head()
```

[28]:

	tariff	user_id	month	duration_count	duration_sum	message_count	\
0	smart	1001	11	59.0	430.0	0	
1	smart	1001	12	63.0	414.0	0	
2	smart	1002	6	15.0	117.0	1	
3	smart	1002	7	26.0	214.0	1	
4	smart	1002	8	42.0	289.0	1	

	mb_used	messages_included	mb_per_month_included	minutes_included	\
0	18429.34	50	15360	500	
1	14036.66	50	15360	500	
2	10856.82	50	15360	500	
3	17580.10	50	15360	500	
4	20319.26	50	15360	500	

	rub_monthly_fee	rub_per_gb	rub_per_message	rub_per_minute	\
0	550	200	3	3	
1	550	200	3	3	
2	550	200	3	3	
3	550	200	3	3	
4	550	200	3	3	

	duration_more_tarrif	messages_more_tarrif	internet_more_tarrif	\
0	0.0	0	3069.34	
1	0.0	0	0.00	
2	0.0	0	0.00	
3	0.0	0	2220.10	
4	0.0	0	4959.26	

	income_total
0	1150.0
1	550.0
2	550.0
3	1150.0
4	1550.0

2.2.1 Conclusion

Changed the column name and data type. We made several summary tables for each direction of the tariff and also made a general table with all the necessary data. Added the final cutting to the table. Moving on to data analysis

2.3 Data analysis

Let's look at the description of the table

[29]:
```
df_income.describe()
```

[29]:

	user_id	month	duration_count	duration_sum	message_count
count	3214.000000	3214.000000	3214.000000	3214.000000	3214.000000
mean	1251.590230	8.317362	63.038892	451.244866	0.845364
std	144.659172	2.905413	33.236368	241.909978	0.361614
min	1000.000000	1.000000	0.000000	0.000000	0.000000
25%	1125.000000	6.000000	40.000000	282.000000	1.000000
50%	1253.000000	9.000000	62.000000	443.000000	1.000000
75%	1378.750000	11.000000	82.000000	589.000000	1.000000
max	1499.000000	12.000000	244.000000	1673.000000	1.000000

	mb_used	messages_included	mb_per_month_included
count	3214.000000	3214.000000	3214.000000
mean	17207.612859	341.148102	20067.405103
std	7570.958771	438.044726	7082.491569
min	0.000000	50.000000	15360.000000
25%	12491.890000	50.000000	15360.000000
50%	16943.175000	50.000000	15360.000000
75%	21424.625000	1000.000000	30720.000000
max	49745.690000	1000.000000	30720.000000

	minutes_included	rub_monthly_fee	rub_per_gb	rub_per_message
count	3214.000000	3214.000000	3214.000000	3214.000000
mean	1266.179216	979.060361	184.676416	2.387057
std	1152.749279	645.539596	23.054986	0.922199
min	500.000000	550.000000	150.000000	1.000000
25%	500.000000	550.000000	150.000000	1.000000
50%	500.000000	550.000000	200.000000	3.000000
75%	3000.000000	1950.000000	200.000000	3.000000
max	3000.000000	1950.000000	200.000000	3.000000

	rub_per_minute	duration_more_tarrif	messages_more_tarrif
count	3214.000000	3214.000000	3214.0
mean	2.387057	28.854698	0.0
std	0.922199	73.077172	0.0
min	1.000000	0.000000	0.0
25%	1.000000	0.000000	0.0

```
50%          3.000000              0.000000                    0.0
75%          3.000000              0.000000                    0.0
max          3.000000            935.000000                    0.0

       internet_more_tarrif  income_total
count           3214.000000   3214.000000
mean            2141.032184   1517.009023
std             3407.192129    798.489284
min                0.000000    550.000000
25%                0.000000    750.000000
50%                0.000000   1619.500000
75%             3550.732500   1950.000000
max            23192.450000   6671.000000
```

The table shows that the majority of subscribers do not exceed the limits for calls and SMS. Half does not exceed the internet limit

Let's see how calls, SMS, Internet and revenue affect each other

Tariff Smart

[30]:
```
df_income_smart = df_income.query('tariff == "smart"')
print(df_income_smart[['duration_sum', 'message_count', 'mb_used',␣
 ↪'income_total']].corr())
```

```
                duration_sum  message_count   mb_used  income_total
duration_sum        1.000000       0.020943  0.340519      0.411425
message_count       0.020943       1.000000  0.011577     -0.010840
mb_used             0.340519       0.011577  1.000000      0.847751
income_total        0.411425      -0.010840  0.847751      1.000000
```

Traffic 0.85 affects revenue the most. The more traffic, the more revenue. Logical Less impacted by calls Almost no effect

Tariff Ultra

[31]:
```
df_income_ultra = df_income.query('tariff == "ultra"')
print(df_income_ultra[['duration_sum', 'message_count', 'mb_used',␣
 ↪'income_total']].corr())
```

```
                duration_sum  message_count   mb_used  income_total
duration_sum        1.000000       0.018719  0.177775      0.090650
message_count       0.018719       1.000000 -0.018834     -0.121952
mb_used             0.177775      -0.018834  1.000000      0.620143
income_total        0.090650      -0.121952  0.620143      1.000000
```

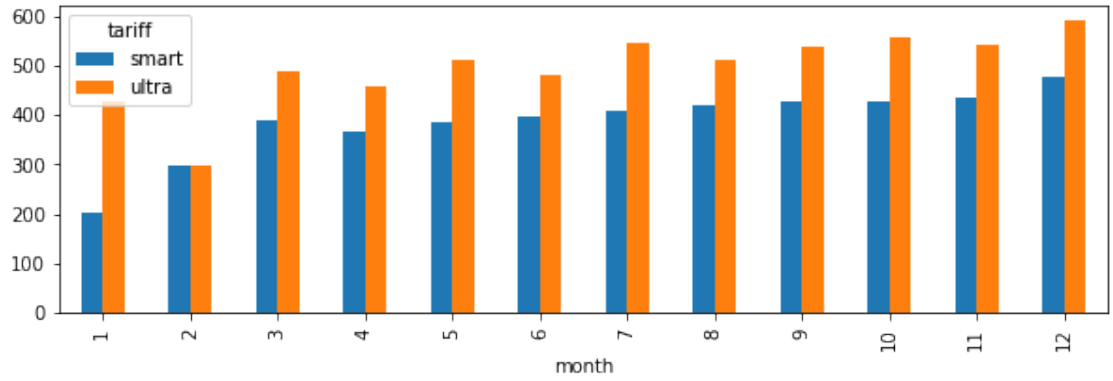
The Internet has a weaker effect on revenue than on the smart tariff, but nevertheless quite strongly Calls and SMS have almost no effect on revenue

For calls, SMS, Internet and revenue, we will build graphs and find the variance and standard deviation

1. Calls

[32]:
```
df_calls_mean = df_income.pivot_table(index='month', columns='tariff',␣
↪values='duration_sum', aggfunc='mean').reset_index()
df_calls_mean.plot(kind='bar', x='month', figsize=(10, 3))
```

[32]: <AxesSubplot:xlabel='month'>



[33]:
```
variance_calls_smart = np.var(df_income_smart['duration_sum'])
variance_calls_ultra = np.var(df_income_ultra['duration_sum'])

std_calls_smart = np.std(df_income_smart['duration_sum'])
std_calls_ultra = np.std(df_income_ultra['duration_sum'])

print('smart call duration variance', variance_calls_smart)
print('Standard deviation of call duration smart', std_calls_smart)
print()
print('ultra call duration variance', variance_calls_ultra)
print('Standard deviation of call duration Ultra', std_calls_ultra)
```

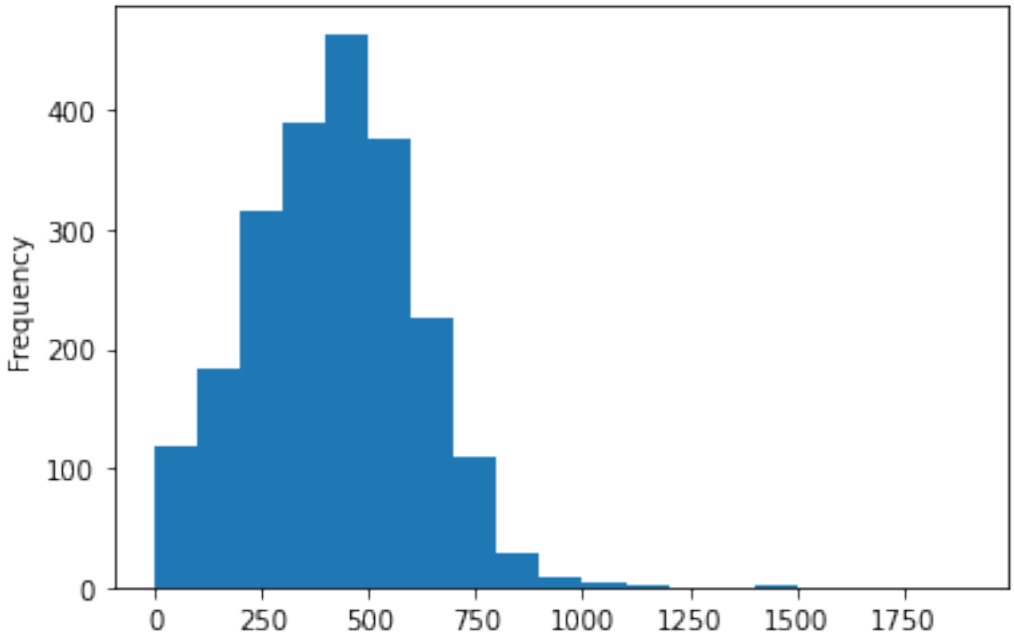
```
smart call duration variance 36203.06665209469
Standard deviation of call duration smart 190.27103471651876

ultra call duration variance 100771.2236172022
Standard deviation of call duration Ultra 317.44483554974113
```

Let's build a histogram of the distribution of call duration for the smart tariff

[34]:
```
df_income_smart['duration_sum'].plot(kind='hist', bins=range(0, 2000, 100))
```

[34]: <AxesSubplot:ylabel='Frequency'>

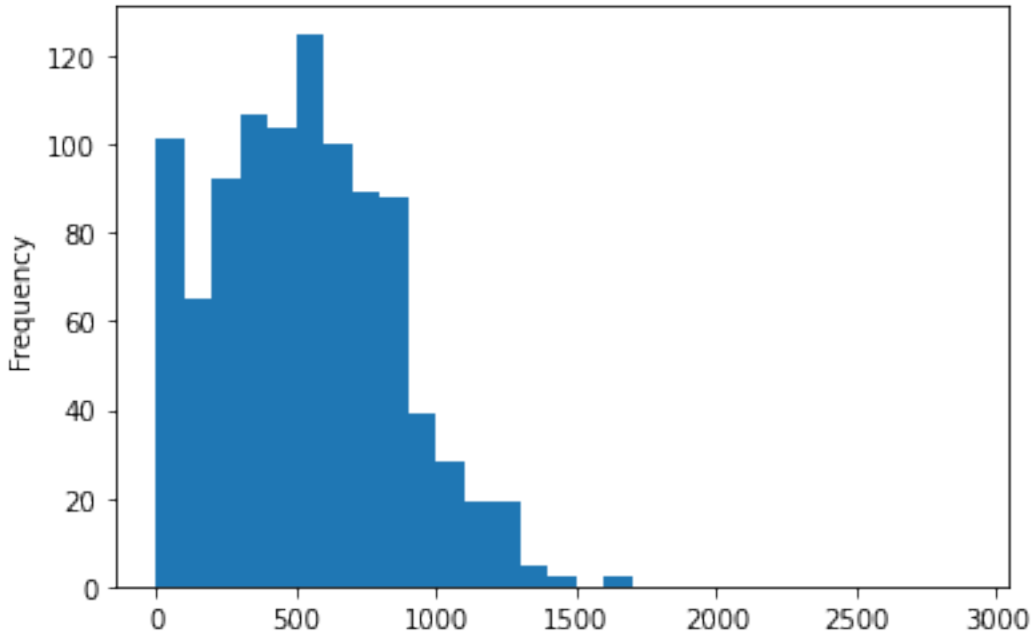


The histogram is normally distributed, with a slight deviation to the right. A small part of calls exceeds 1000 minutes per month.

Let's build a histogram of the distribution of call duration for the ultra tariff

[35]:
```
df_income_ultra['duration_sum'].plot(kind='hist', bins=range(0, 3000, 100))
```

[35]:
```
<AxesSubplot:ylabel='Frequency'>
```

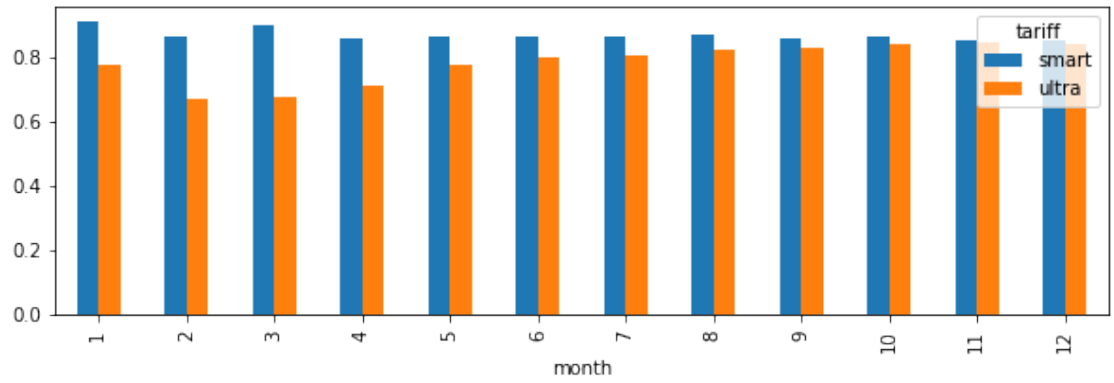


The histogram shows that all users fit into the conversation rate (3000 minutes according to the tariff). And almost no one talks for more than 1,500 minutes a month.

2. SMS

[36]:
```
df_messages_mean = df_income.pivot_table(index='month', columns='tariff',␣
↪values='message_count', aggfunc='mean').reset_index()
df_messages_mean.plot(kind='bar', x='month', figsize=(10, 3))
```

[36]: <AxesSubplot:xlabel='month'>



[37]:
```
variance_messages_smart = np.var(df_income_smart['message_count'])
variance_messages_ultra = np.var(df_income_ultra['message_count'])

std_messages_smart = np.std(df_income_smart['message_count'])
std_messages_ultra = np.std(df_income_ultra['message_count'])

print('variance of the number of sms smart', variance_messages_smart)
print('Standard deviation of call duration smart', std_messages_smart)
print()
print('variance of the number of sms ultra', variance_messages_ultra)
print('Standard deviation of call duration smart', std_messages_ultra)
```

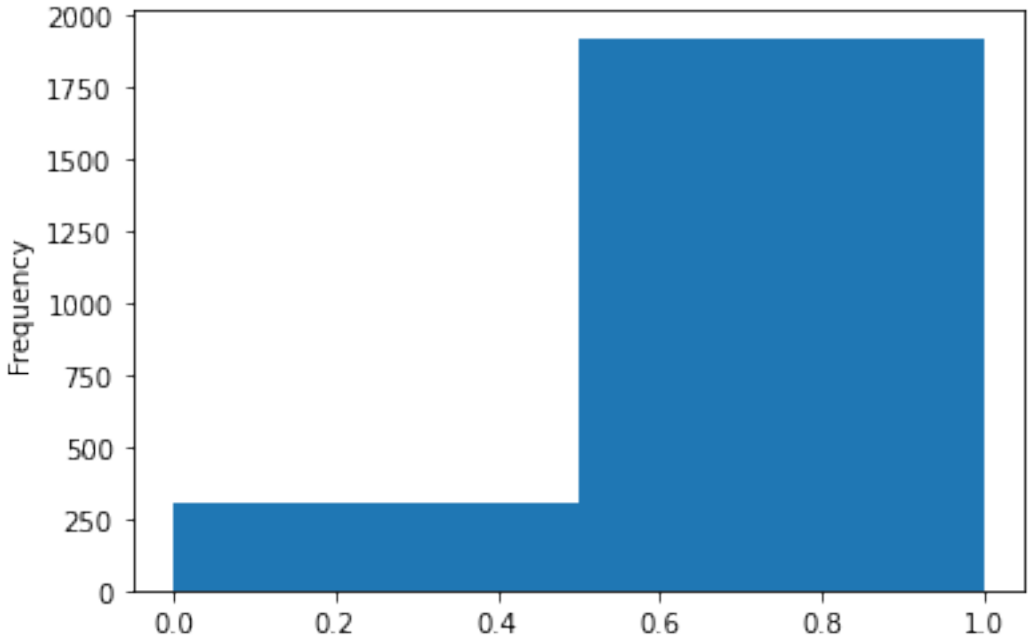
```
variance of the number of sms smart 0.11876039184122342
Standard deviation of call duration smart 0.34461629654040365

variance of the number of sms ultra 0.1556855368600067
Standard deviation of call duration smart 0.394570065843833
```

Let's build a histogram of SMS distribution for the smart tariff

[38]:
```
df_income_smart['message_count'].plot(kind='hist', bins=2)
```

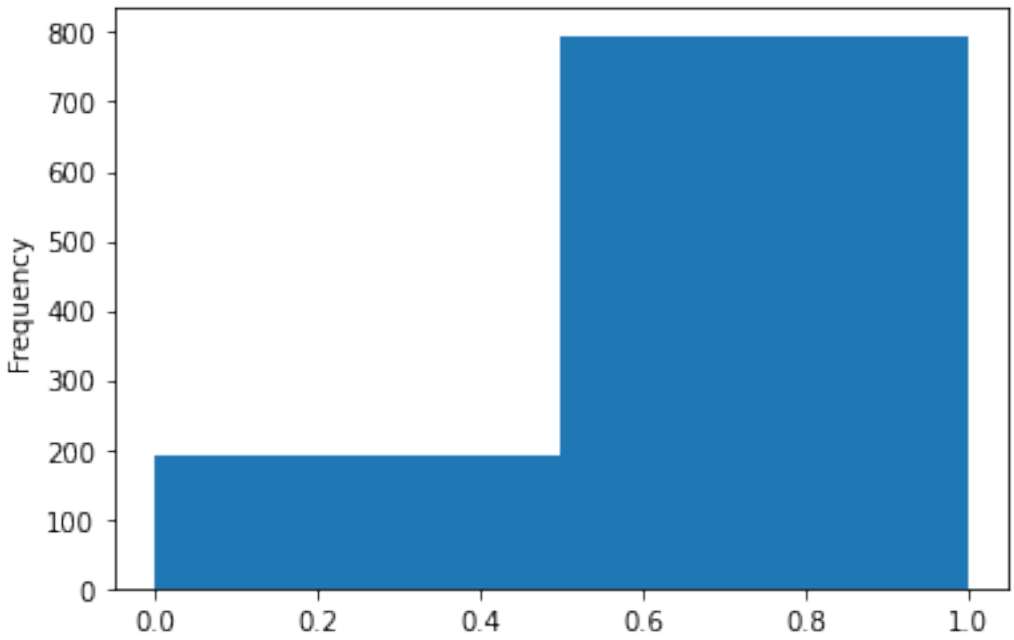
[38]: <AxesSubplot:ylabel='Frequency'>



Let's build a histogram of the distribution of call duration for the ultra tariff

[39]:
```
df_income_ultra['message_count'].plot(kind='hist', bins=2)
```

[39]: <AxesSubplot:ylabel='Frequency'>

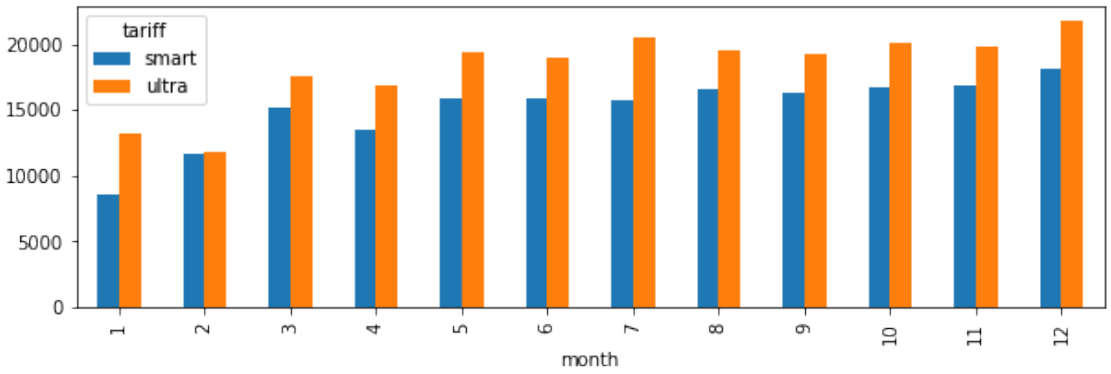


Two histograms show that there are subscribers who do not send SMS at all. The share of subscribers who do not send SMS anymore with the ultra tariff

3. Internet

[40]:
```
df_internet_mean = df_income.pivot_table(index='month', columns='tariff',␣
↪values='mb_used', aggfunc='mean').reset_index()
df_internet_mean.plot(kind='bar', x='month', figsize=(10, 3))
```

[40]: <AxesSubplot:xlabel='month'>



[41]:
```
variance_internet_smart = np.var(df_income_smart['mb_used'])
variance_internet_ultra = np.var(df_income_ultra['mb_used'])

std_internet_smart = np.std(df_income_smart['mb_used'])
std_internet_ultra = np.std(df_income_ultra['mb_used'])

print('Internet varience smart', variance_internet_smart)
print('Internet smart standard deviation', std_internet_smart)
print()
print('Internet varience ultra', variance_internet_ultra)
print('Internet ultra standard deviation ', std_internet_ultra)
```

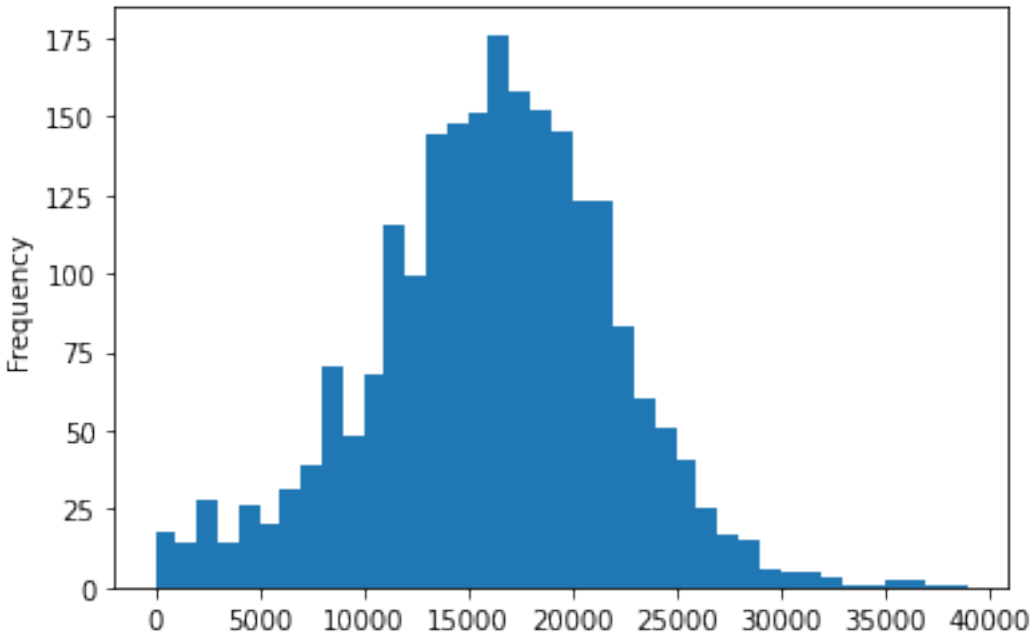
```
Internet varience smart 34447035.49528493
Internet smart standard deviation 5869.159692433401

Internet varience ultra 101647713.26174639
Internet ultra standard deviation  10082.049060669482
```

Let's build a histogram of Internet traffic distribution for the smart tariff

[42]:
```
df_income_smart['mb_used'].plot(kind='hist', bins=range(0, 40000, 1000))
```

[42]: <AxesSubplot:ylabel='Frequency'>

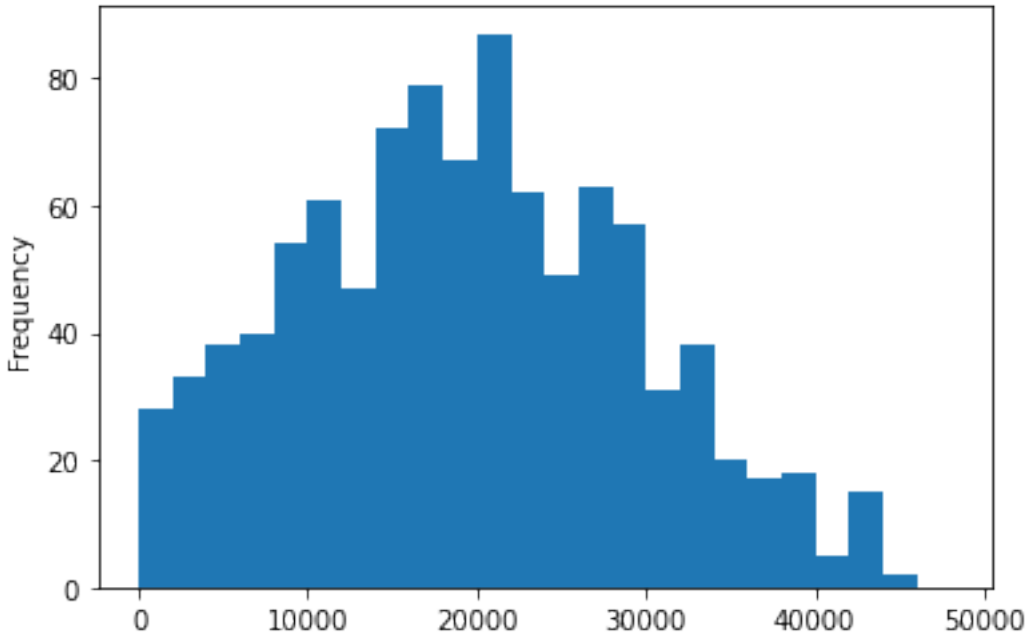


The histogram is normally distributed. But most of the subscribers exceed the traffic limit of 15GB.

Let's build a histogram of Internet traffic distribution for the ultra tariff

[43]:
```
df_income_ultra['mb_used'].plot(kind='hist', bins=range(0, 50000, 2000))
```

[43]: <AxesSubplot:ylabel='Frequency'>

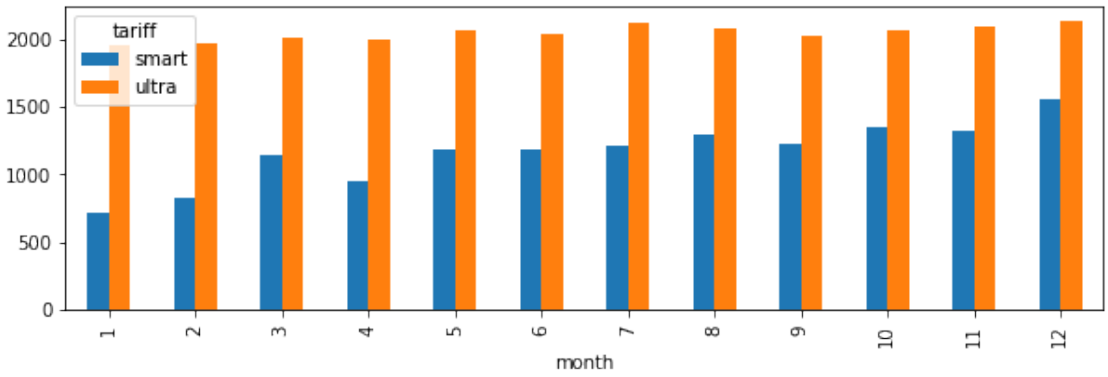


Subscribers of the ultra tariff are more likely to fit into the 30GB limit, but there are also subscribers who exceed the norm.

4. Income

[44]:
```
df_income_mean = df_income.pivot_table(index='month', columns='tariff',␣
↪values='income_total', aggfunc='mean').reset_index()
df_income_mean.plot(kind='bar', x='month', figsize=(10, 3))
```

[44]: <AxesSubplot:xlabel='month'>



[45]:
```
variance_income_smart = np.var(df_income_smart['income_total'])
variance_income_ultra = np.var(df_income_ultra['income_total'])

std_income_smart = np.std(df_income_smart['income_total'])
std_income_ultra = np.std(df_income_ultra['income_total'])

print('smart revenue variance', variance_income_smart)
print('smart revenue standard deviation', std_income_smart)
print()
print('ultra revenue variance', variance_income_ultra)
print('ultra revenue standard deviation', std_income_ultra)
```

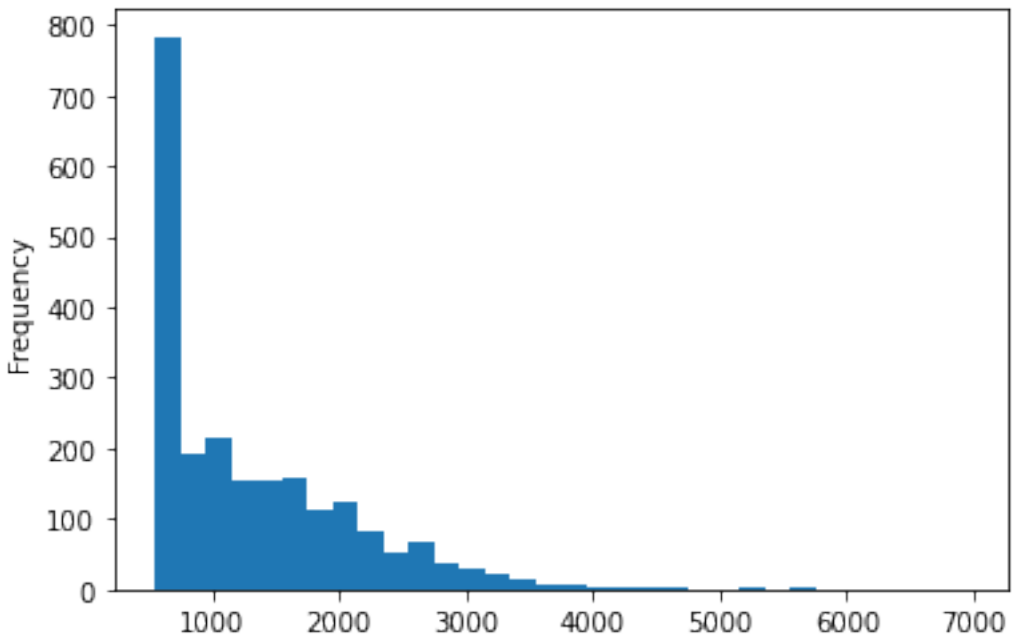
```
smart revenue variance 661620.0229295266
smart revenue standard deviation 813.4002845644491

ultra revenue variance 141373.07325620347
ultra revenue standard deviation 375.9961080333192
```

Let's build a histogram of the distribution of the final revenue for the smart tariff

[46]:
```
df_income_smart['income_total'].plot(kind='hist', bins=range(550, 7000, 200))
```

[46]:
```
<AxesSubplot:ylabel='Frequency'>
```

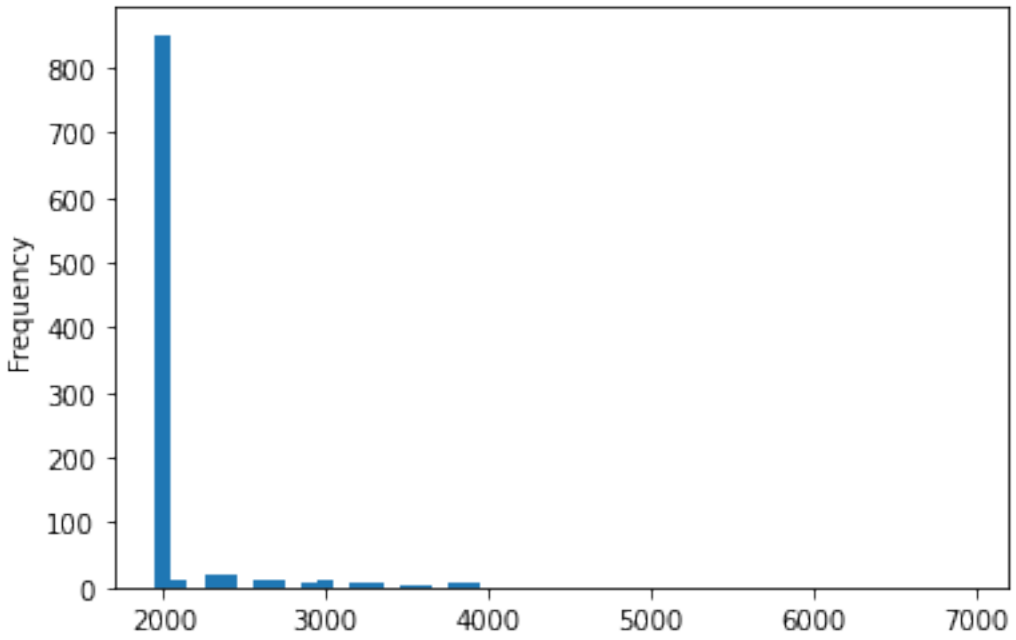


Many smart tariff subscribers pay more than 550 rubles/month. Moreover, there are also such subscribers who pay more than the cost of the ultra tariff. They clearly should change their tariff to save money

Let's build a histogram of the distribution of the final revenue for the ultra tariff

[47]:
```
df_income_ultra['income_total'].plot(kind='hist', bins=range(1950, 7000, 100))
```

[47]:
```
<AxesSubplot:ylabel='Frequency'>
```



Almost all subscribers of the ultra tariff pay only a monthly fee, without additional services. There is a small part of subscribers who exceed the limits, but the share of such subscribers is much less than that of the smart tariff

Find the number of subscribers for each tariff and the total revenue by tariffs

[48]:
```
print('Total income by tariffs')
print(df_income.pivot_table(index='tariff', values='income_total',␣
 ↪aggfunc='sum'))
print()
print('Total number of subscribers')
print(df_users.pivot_table(index='tariff', values='user_id', aggfunc='count'))
```

```
Total income by tariffs
        income_total
tariff
smart      2836567.0
ultra      2039100.0

Total number of subscribers
        user_id
```

```
tariff
smart       351
ultra       149
```

2.3.1 Conclusion

The graphs show that the average value for all parameters is higher for the ultra tariff, except for the number of SMS. In terms of the average number of SMS, the smart tariff is the leader The dispersion for calls, SMS and traffic is greater for the ultra tariff. The revenue dispersion of the smart tariff is greater The revenue of the smart tariff is strongly influenced by traffic. I think that 15GB is not enough for many subscribers and they pay separately for additional traffic. Calls are less affected The revenue of the ultra tariff is mainly influenced by the Internet. But here more subscribers fit into the tariff norms than on smart The smart tariff is used by more people than the ultra tariff. And the total revenue of the smart tariff is higher. We built distribution histograms for calls, sms, internet and revenue

2.4 Hypothesis testing

Hypothesis 1 H0 - Average revenue of Ultra tariff users = average revenue of Smart tariff users» H1 - Average revenue of Ultra tariff users average revenue of Smart tariff users»

We already have two ready tables

- df_income_smart
- df_income_ultra

Let's test the hypothesis

[49]:
```
alpha = .05

results = st.ttest_ind(
    df_income_smart['income_total'],
    df_income_ultra['income_total'],
    equal_var = False)

print('p-value: ', results.pvalue)

if results.pvalue < alpha:
    print("Rejecting the null hypothesis")
else:
    print("Failed to reject the null hypothesis")
```

```
p-value:  3.798003235009034e-261
Rejecting the null hypothesis
```

Because we don't have much data, then we can calculate more accurately using the mean() function

[50]:
```
print(df_income_smart['income_total'].mean())
print(df_income_ultra['income_total'].mean())
```

```
1272.5737999102737
2070.1522842639592
```

The hypothesis was not confirmed. The average revenue of the smart tariff is almost two times less than that of the ultra tariff

Hypothesis 2 H0 - Average revenue of users from Moscow = average revenue of users from other cities H1 - Average revenue of users from Moscow average revenue of users from other cities

Well, let's check

First, let's add the city to the table with revenue

[51]:
```
df_income = df_income.merge(df_users[['user_id', 'city']], on='user_id')
```

Now let's make two separate tables: 1. Only Moscow 2. Other cities

[52]:
```
df_income_moscow_city = df_income.query('city == "    "')
df_income_others_city = df_income.query('city != "    "')
```

Testing the hypothesis

[53]:
```
alpha = .05

results = st.ttest_ind(
    df_income_moscow_city['income_total'],
    df_income_others_city['income_total'],
    equal_var = False)

print('p-value: ', results.pvalue)

if results.pvalue < alpha:
    print("Rejecting the null hypothesis")
else:
    print("Failed to reject the null hypothesis")
```

```
p-value:  0.425625168367989
Failed to reject the null hypothesis
```

The hypothesis was confirmed, because pvalue turned out to be 42% The revenue of Moscow and other cities is approximately the same

Let's double-check

[54]:
```
print(df_income_moscow_city['income_total'].mean())
print(df_income_others_city['income_total'].mean())
```

```
1539.1767594108019
1511.8056089127929
```

Difference less than 100p

2.4.1 Conclusion

Hypothesis 1 was confirmed. Revenue between the two tariffs differs almost twice Hypothesis 2 was not confirmed. Revenue between Moscow and other cities is not much different.

For two hypotheses, I used a method to test the hypothesis that the mean of two populations is equal based on samples taken from them: scipy.stats.ttest_ind (array1, array2,equal_var)

We complete the project

2.5 General conclusion

After analyzing the data on 500 Megaline subscribers, we draw the following conclusion.

- The smart tariff brings the greatest revenue for the company. Also, the smart tariff has more number of people.
- The revenue of the ultra tariff is more stable than that of the smart tariff. I assume that this is due to the fact that many smart subscribers buy additional traffic.
- The dispersion for calls, SMS and traffic is greater for the ultra tariff. The revenue dispersion of the smart tariff is larger. This means that smart tariff subscribers more often exceed the limits set in their tariff and, accordingly, overpay more often.
- The hypothesis that the average revenue of users of the Ultra and Smart tariffs are equal has not been confirmed. On average, subscribers pay almost twice as much for the ultra tariff
- The hypothesis that the average revenue of users from Moscow is equal to the revenue of users from other regions has been confirmed. The difference in revenue between Moscow and other cities is no more than 100 rubles.

After analyzing this sample, we conclude the following:

The smart tariff is better than the ultra tariff. more revenue from this takrif. Smart tariff revenue - 2.8 million rubles, ultra tariff - 2 million rubles. Also, the smart tariff is used by more subscribers than the ultra tariff. 351 and 149 respectively.