Exploratory data analysis

Loading data

import the data.

In [228…
```
import pandas as pd

client_data = pd.read_csv("./data/client_data.csv")
price_data = pd.read_csv("./data/price_data.csv")

pd.set_option('display.max_columns', None)
```

In [229…
```
client_data.head()
```

Out[229]:

	id	channel_sales	cons_12m	cons_gas_12m
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	0
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	0

In [230…
```
price_data.head()
```

Out[230]:

	id	price_date	price_off_peak_var	price_peak_var	price_mid_pea
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0	
1	038af19179925da21a25619c5a24b745	2015-02-01	0.151367	0.0	
2	038af19179925da21a25619c5a24b745	2015-03-01	0.151367	0.0	
3	038af19179925da21a25619c5a24b745	2015-04-01	0.149626	0.0	
4	038af19179925da21a25619c5a24b745	2015-05-01	0.149626	0.0	

Description of data

client_data.csv

- id = client company identifier
- activity_new = category of the company's activity
- channel_sales = code of the sales channel
- cons_12m = electricity consumption of the past 12 months
- cons_gas_12m = gas consumption of the past 12 months
- cons_last_month = electricity consumption of the last month
- date_activ = date of activation of the contract
- date_end = registered date of the end of the contract
- date_modif_prod = date of the last modification of the product
- date_renewal = date of the next contract renewal
- forecast_cons_12m = forecasted electricity consumption for next 12 months
- forecast_cons_year = forecasted electricity consumption for the next calendar year
- forecast_discount_energy = forecasted value of current discount
- forecast_meter_rent_12m = forecasted bill of meter rental for the next 2 months
- forecast_price_energy_off_peak = forecasted energy price for 1st period (off peak)
- forecast_price_energy_peak = forecasted energy price for 2nd period (peak)
- forecast_price_pow_off_peak = forecasted power price for 1st period (off peak)
- has_gas = indicated if client is also a gas client
- imp_cons = current paid consumption
- margin_gross_pow_ele = gross margin on power subscription
- margin_net_pow_ele = net margin on power subscription
- nb_prod_act = number of active products and services
- net_margin = total net margin
- num_years_antig = antiquity of the client (in number of years)
- origin_up = code of the electricity campaign the customer first subscribed to

- pow_max = subscribed power
- churn = has the client churned over the next 3 months

price_data.csv

- id = client company identifier
- price_date = reference date
- price_off_peak_var = price of energy for the 1st period (off peak)
- price_peak_var = price of energy for the 2nd period (peak)
- price_mid_peak_var = price of energy for the 3rd period (mid peak)
- price_off_peak_fix = price of power for the 1st period (off peak)
- price_peak_fix = price of power for the 2nd period (peak)
- price_mid_peak_fix = price of power for the 3rd period (mid peak)

Data types

In [231…

```
client_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14606 entries, 0 to 14605
Data columns (total 26 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   id                               14606 non-null  object 
 1   channel_sales                    14606 non-null  object 
 2   cons_12m                         14606 non-null  int64  
 3   cons_gas_12m                     14606 non-null  int64  
 4   cons_last_month                  14606 non-null  int64  
 5   date_activ                       14606 non-null  object 
 6   date_end                         14606 non-null  object 
 7   date_modif_prod                  14606 non-null  object 
 8   date_renewal                     14606 non-null  object 
 9   forecast_cons_12m                14606 non-null  float64
 10  forecast_cons_year               14606 non-null  int64  
 11  forecast_discount_energy         14606 non-null  float64
 12  forecast_meter_rent_12m          14606 non-null  float64
 13  forecast_price_energy_off_peak   14606 non-null  float64
 14  forecast_price_energy_peak       14606 non-null  float64
 15  forecast_price_pow_off_peak      14606 non-null  float64
 16  has_gas                          14606 non-null  object 
 17  imp_cons                         14606 non-null  float64
 18  margin_gross_pow_ele             14606 non-null  float64
 19  margin_net_pow_ele               14606 non-null  float64
 20  nb_prod_act                      14606 non-null  int64  
 21  net_margin                       14606 non-null  float64
 22  num_years_antig                  14606 non-null  int64  
 23  origin_up                        14606 non-null  object 
 24  pow_max                          14606 non-null  float64
 25  churn                            14606 non-null  int64  
dtypes: float64(11), int64(7), object(8)
memory usage: 2.9+ MB
```

Changing the type of the columns containing dates to date type.

In [232…
```
def change_to_date_type(data, columns, current_format):
    data = data.copy()
    for column in columns:
        data[column] = pd.to_datetime(data[column], format=current_format)
    return data
```

In [233…
```
client_data = change_to_date_type(client_data, ["date_activ", "date_end", "date_mod
```

In [234…
```
client_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14606 entries, 0 to 14605
Data columns (total 26 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              14606 non-null  object
 1   channel_sales                   14606 non-null  object
 2   cons_12m                        14606 non-null  int64
 3   cons_gas_12m                    14606 non-null  int64
 4   cons_last_month                 14606 non-null  int64
 5   date_activ                      14606 non-null  datetime64[ns]
 6   date_end                        14606 non-null  datetime64[ns]
 7   date_modif_prod                 14606 non-null  datetime64[ns]
 8   date_renewal                    14606 non-null  datetime64[ns]
 9   forecast_cons_12m               14606 non-null  float64
 10  forecast_cons_year              14606 non-null  int64
 11  forecast_discount_energy        14606 non-null  float64
 12  forecast_meter_rent_12m         14606 non-null  float64
 13  forecast_price_energy_off_peak  14606 non-null  float64
 14  forecast_price_energy_peak      14606 non-null  float64
 15  forecast_price_pow_off_peak     14606 non-null  float64
 16  has_gas                         14606 non-null  object
 17  imp_cons                        14606 non-null  float64
 18  margin_gross_pow_ele            14606 non-null  float64
 19  margin_net_pow_ele              14606 non-null  float64
 20  nb_prod_act                     14606 non-null  int64
 21  net_margin                      14606 non-null  float64
 22  num_years_antig                 14606 non-null  int64
 23  origin_up                       14606 non-null  object
 24  pow_max                         14606 non-null  float64
 25  churn                           14606 non-null  int64
dtypes: datetime64[ns](4), float64(11), int64(7), object(4)
memory usage: 2.9+ MB
```

In [235…
```
price_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Data columns (total 8 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   id                  193002 non-null  object 
 1   price_date          193002 non-null  object 
 2   price_off_peak_var  193002 non-null  float64
 3   price_peak_var      193002 non-null  float64
 4   price_mid_peak_var  193002 non-null  float64
 5   price_off_peak_fix  193002 non-null  float64
 6   price_peak_fix      193002 non-null  float64
 7   price_mid_peak_fix  193002 non-null  float64
dtypes: float64(6), object(2)
memory usage: 11.8+ MB
```

In [236…

```
price_data = change_to_date_type(price_data, ["price_date"], "%Y-%m-%d")
price_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193002 entries, 0 to 193001
Data columns (total 8 columns):
 #   Column              Non-Null Count   Dtype         
---  ------              --------------   -----         
 0   id                  193002 non-null  object        
 1   price_date          193002 non-null  datetime64[ns]
 2   price_off_peak_var  193002 non-null  float64       
 3   price_peak_var      193002 non-null  float64       
 4   price_mid_peak_var  193002 non-null  float64       
 5   price_off_peak_fix  193002 non-null  float64       
 6   price_peak_fix      193002 non-null  float64       
 7   price_mid_peak_fix  193002 non-null  float64       
dtypes: datetime64[ns](1), float64(6), object(1)
memory usage: 11.8+ MB
```

Statistics

In [237…

```
client_data.describe()
```

Out[237]:

	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	dat
count	1.460600e+04	1.460600e+04	14606.000000	14606	14606	
mean	1.592203e+05	2.809238e+04	16090.269752	2011-01-28 07:54:18.879912448	2016-07-27 20:48:26.422018560	12:29:
min	0.000000e+00	0.000000e+00	0.000000	2003-05-09 00:00:00	2016-01-28 00:00:00	
25%	5.674750e+03	0.000000e+00	0.000000	2010-01-15 00:00:00	2016-04-27 06:00:00	
50%	1.411550e+04	0.000000e+00	792.500000	2011-03-04 00:00:00	2016-08-01 00:00:00	
75%	4.076375e+04	0.000000e+00	3383.000000	2012-04-19 00:00:00	2016-10-31 00:00:00	
max	6.207104e+06	4.154590e+06	771203.000000	2014-09-01 00:00:00	2017-06-13 00:00:00	
std	5.734653e+05	1.629731e+05	64364.196422	NaN	NaN	

In [238…

```
price_data.describe()
```

Out[238]:

	price_date	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix
count	193002	193002.000000	193002.000000	193002.000000	193002.000000
mean	2015-06-16 12:50:49.933161216	0.141027	0.054630	0.030496	43.334477
min	2015-01-01 00:00:00	0.000000	0.000000	0.000000	0.000000
25%	2015-04-01 00:00:00	0.125976	0.000000	0.000000	40.728885
50%	2015-07-01 00:00:00	0.146033	0.085483	0.000000	44.266930
75%	2015-10-01 00:00:00	0.151635	0.101673	0.072558	44.444710
max	2015-12-01 00:00:00	0.280700	0.229788	0.114102	59.444710
std	NaN	0.025032	0.049924	0.036298	5.410297

Data expolration

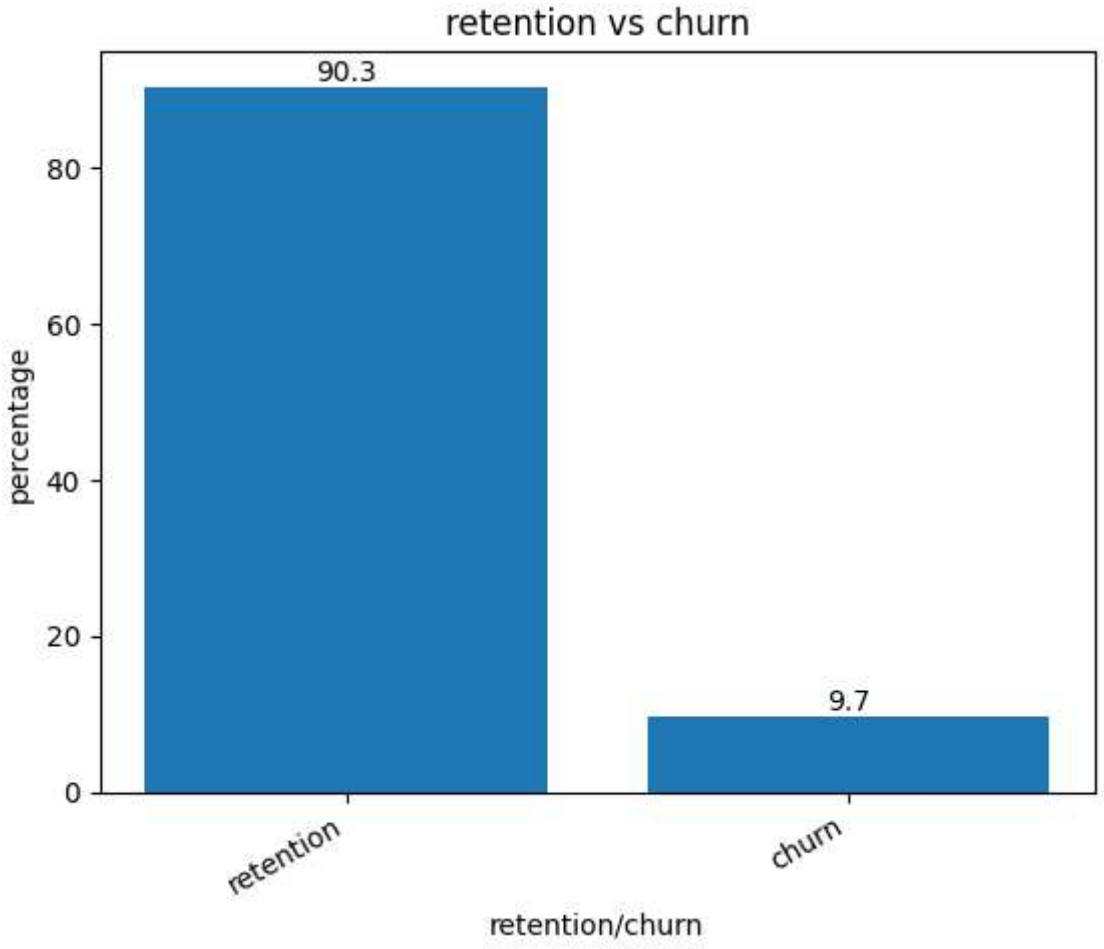
Retention vs churn

In [239…
```
import matplotlib.pyplot as plt


def plot_bar(x, y, xlabel, ylabel, title):
    fig, ax = plt.subplots()
    bar = ax.bar(x, y)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    # for i in range(len(x)):
    #     ax.annotate(str(y[i]), xy=(x[i], y[i]+5), horizontalalignment="center", v
    ax.bar_label(container=bar)

    ax.set_xticks(x)
    ax.set_xticklabels(x, rotation=30, ha='right')
```

In [240…
```
df_churn = client_data.groupby("churn").size()
df_churn = df_churn.apply(lambda x: round((x/len(client_data))*100, 1))
df_churn.index = ["retention", "churn"]
df_churn
```

Out[240]:
```
retention    90.3
churn         9.7
dtype: float64
```

In [241…
```
plot_bar(list(df_churn.index), df_churn, "retention/churn", "percentage", "retentio
```



So the company lost around 10% of customers.

Let's check the available sales channel.

In [242…
```
client_data["channel_sales"].value_counts()
```

Out[242]:
```
channel_sales
foosdfpfkusacimwkcsosbicdxkicaua    6754
MISSING                             3725
lmkebamcaaclubfxadlmueccxoimlema    1843
usilxuppasemubllopkaafesmlibmsdf    1375
ewpakwlliwisiwduibdlfmalxowmwpci     893
sddiedcslfslkckwlfkdpoeeailfpeds      11
epumfxlbckeskwekxbiuasklxalciiuu       3
fixdbufsefwooaasfcxdxadsiekoceaa       2
Name: count, dtype: int64
```

The channel sales codes are very difficulat to remember and visualize. So let's replace with simple codes.

In [243…
```
channel_code_dict = {}
for i, code in enumerate(client_data["channel_sales"].value_counts().index):
    channel_code_dict[code] = "channel_" + str(i)
channel_code_dict
```

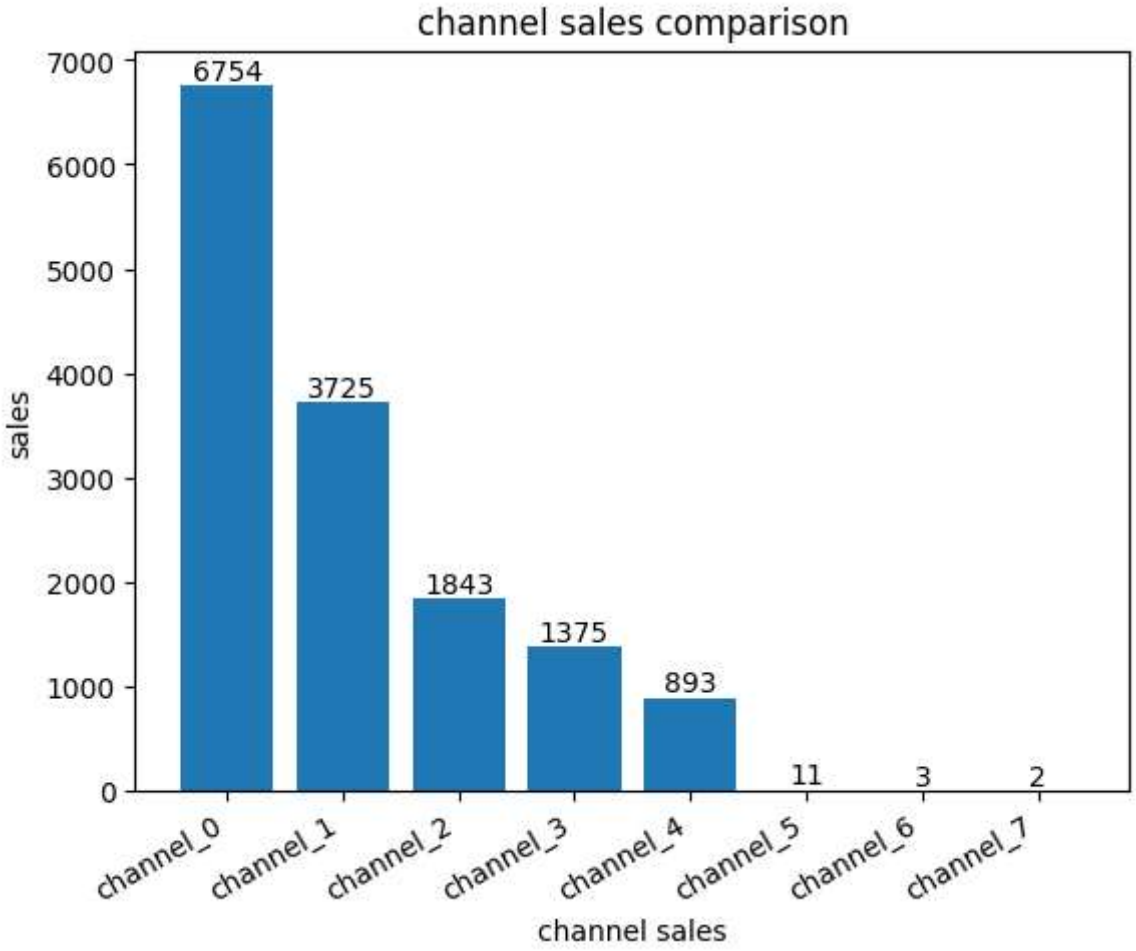
Out[243]:
```
{'foosdfpfkusacimwkcsosbicdxkicaua': 'channel_0',
 'MISSING': 'channel_1',
 'lmkebamcaaclubfxadlmueccxoimlema': 'channel_2',
 'usilxuppasemubllopkaafesmlibmsdf': 'channel_3',
 'ewpakwlliwisiwduibdlfmalxowmwpci': 'channel_4',
 'sddiedcslfslkckwlfkdpoeeailfpeds': 'channel_5',
 'epumfxlbckeskwekxbiuasklxalciiuu': 'channel_6',
 'fixdbufsefwooaasfcxdxadsiekoceaa': 'channel_7'}
```

Let's replace the old channel codes with new codes.

In [244…
```
client_data["channel_sales"] = client_data["channel_sales"].replace(channel_code_di
channel_sales = client_data["channel_sales"].value_counts()
channel_sales
```

Out[244]:
```
channel_sales
channel_0    6754
channel_1    3725
channel_2    1843
channel_3    1375
channel_4     893
channel_5      11
channel_6       3
channel_7       2
Name: count, dtype: int64
```

In [245…
```
plot_bar(x=channel_sales.index, y=channel_sales, xlabel="channel sales", ylabel="sa
```



In [246…
```
client_data["origin_up"].value_counts()
```

Out[246]:
```
origin_up
lxidpiddsbxsbosboudacockeimpuepw    7097
kamkkxfxxuwbdslkwifmmcsiusiuosws    4294
ldkssxwpmemidmecebumciepifcamkci    3148
MISSING                               64
usapbepcfoloekilkwsdiboslwaxobdp       2
ewxeelcelemmiwuafmddpobolfuxioce       1
Name: count, dtype: int64
```

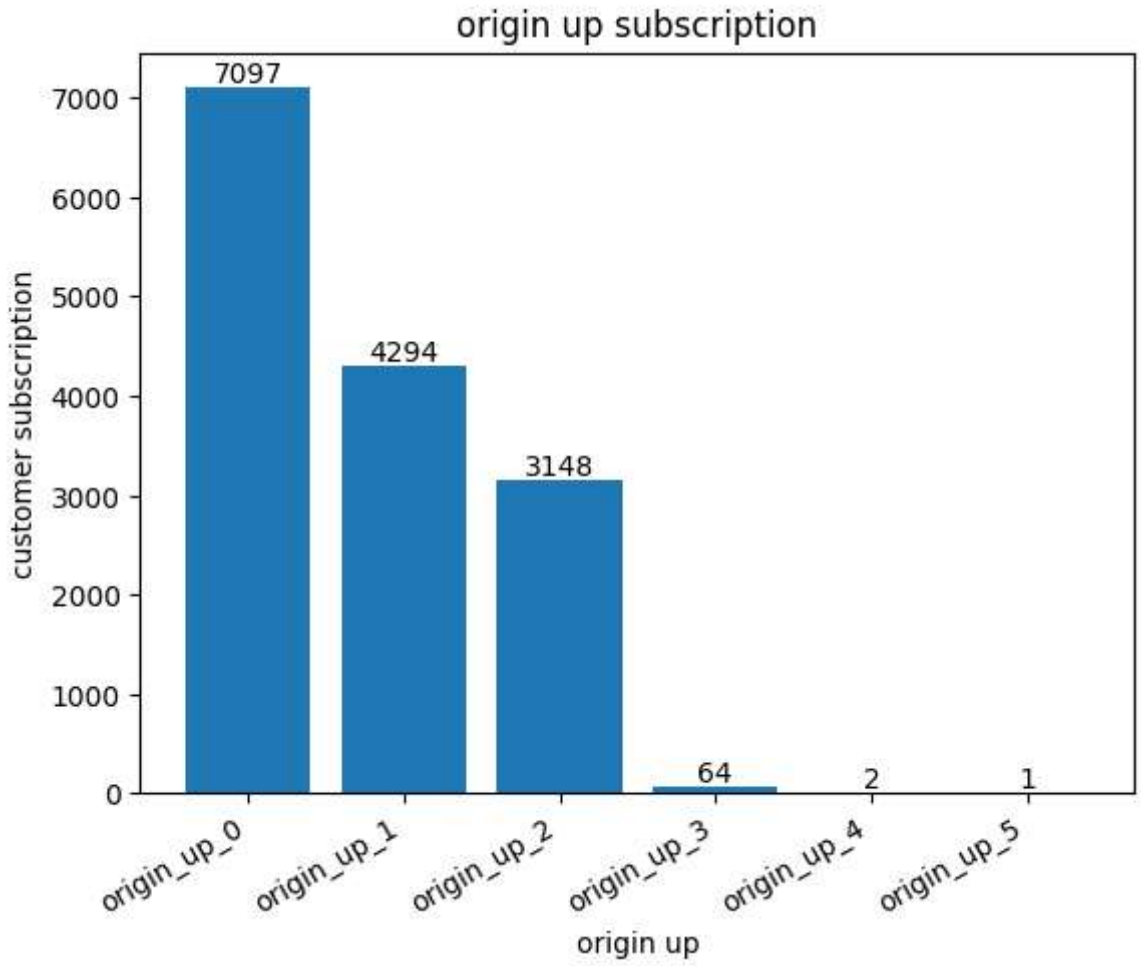
In [247…
```
origin_up_code_dict = {}
for i, code in enumerate(client_data["origin_up"].value_counts().index):
    origin_up_code_dict[code] = "origin_up_" + str(i)
origin_up_code_dict
```

Out[247]:
```
{'lxidpiddsbxsbosboudacockeimpuepw': 'origin_up_0',
 'kamkkxfxxuwbdslkwifmmcsiusiuosws': 'origin_up_1',
 'ldkssxwpmemidmecebumciepifcamkci': 'origin_up_2',
 'MISSING': 'origin_up_3',
 'usapbepcfoloekilkwsdiboslwaxobdp': 'origin_up_4',
 'ewxeelcelemmiwuafmddpobolfuxioce': 'origin_up_5'}
```

In [248…
```
client_data["origin_up"] = client_data["origin_up"].replace(origin_up_code_dict)
origin_up = client_data["origin_up"].value_counts()
origin_up
```

Out[248]:
```
origin_up
origin_up_0    7097
origin_up_1    4294
origin_up_2    3148
origin_up_3      64
origin_up_4       2
origin_up_5       1
Name: count, dtype: int64
```

In [249…
```
plot_bar(x=origin_up.index, y=origin_up, xlabel="origin up", ylabel="customer subsc
```



churn by sales channels

In [250…
```
def plot_stacked_bar(data, x_cols, y_cols, xlabel, ylabel, title):
    ax = data.plot.bar(x=x_cols, y=y_cols, stacked=True, figsize=(8, 4))
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    for rect in ax.patches:
        height = rect.get_height()
        width = rect.get_width()
        x = rect.get_x()
        y = rect.get_y()
        # The width of the bar is also not pixels, it's the
```

```
        # number of animals. So we can use it as the label!
        label_text = round(height, 1)

        # ax.text(x, y, text)
        label_x = x + width / 2
        label_y = y + height / 2
        if height != 0:
            ax.text(label_x, label_y, label_text, ha='center', va='center')
    ax.set_xticklabels(data[x_cols], rotation=30, ha='right')
```

In [251…

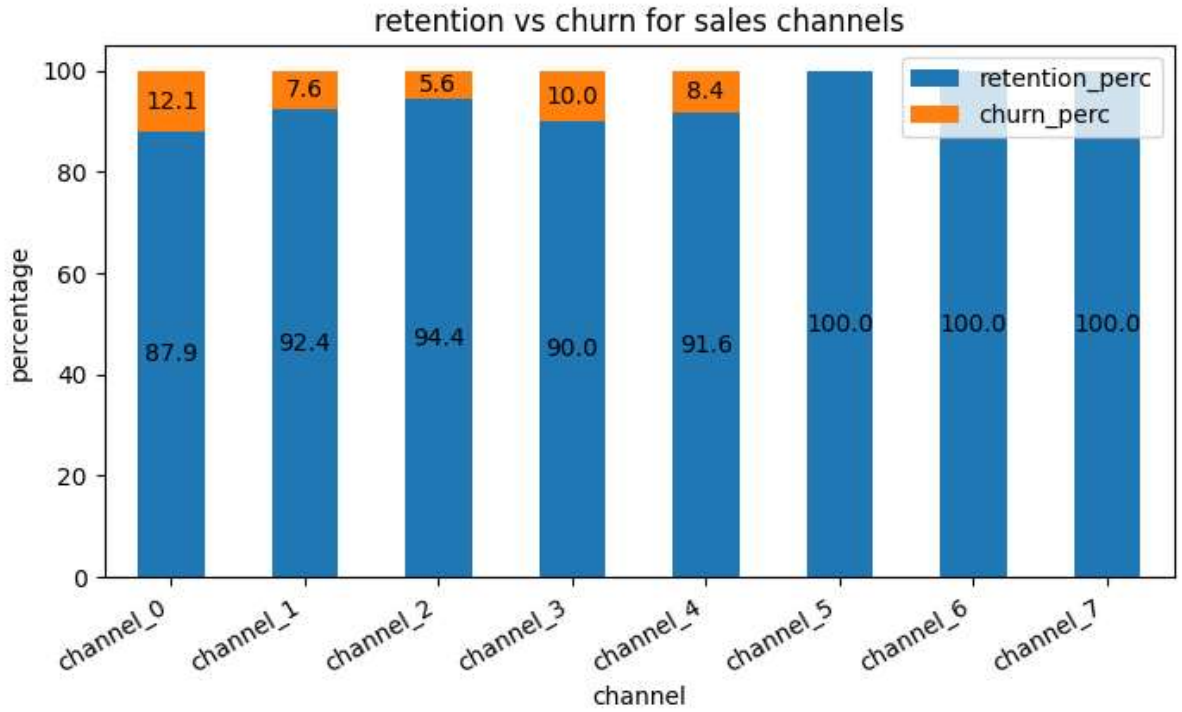
```
channel_sales_churn = pd.DataFrame()
channel_sales_churn["channel"] = channel_sales.index
channel_sales_churn["num_of_customers"] = channel_sales.values
channel_sales_churn["churn_count"] = client_data.groupby("channel_sales")["churn"].
channel_sales_churn["retention_count"] = channel_sales_churn["num_of_customers"].va
channel_sales_churn["churn_perc"] = (channel_sales_churn["churn_count"].values/chan
channel_sales_churn["retention_perc"] = (channel_sales_churn["retention_count"].val
channel_sales_churn["churn_perc"] = channel_sales_churn["churn_perc"].apply(lambda
channel_sales_churn["retention_perc"] = channel_sales_churn["retention_perc"].apply
channel_sales_churn
```

Out[251]:

	channel	num_of_customers	churn_count	retention_count	churn_perc	retention_perc
0	channel_0	6754	820	5934	12.1	87.9
1	channel_1	3725	283	3442	7.6	92.4
2	channel_2	1843	103	1740	5.6	94.4
3	channel_3	1375	138	1237	10.0	90.0
4	channel_4	893	75	818	8.4	91.6
5	channel_5	11	0	11	0.0	100.0
6	channel_6	3	0	3	0.0	100.0
7	channel_7	2	0	2	0.0	100.0

In [252…

```
plot_stacked_bar(data=channel_sales_churn, x_cols="channel", y_cols=["retention_per
```



`channel_0`, `channel_3` have high customer churn.

Let's check the churn for origin up.

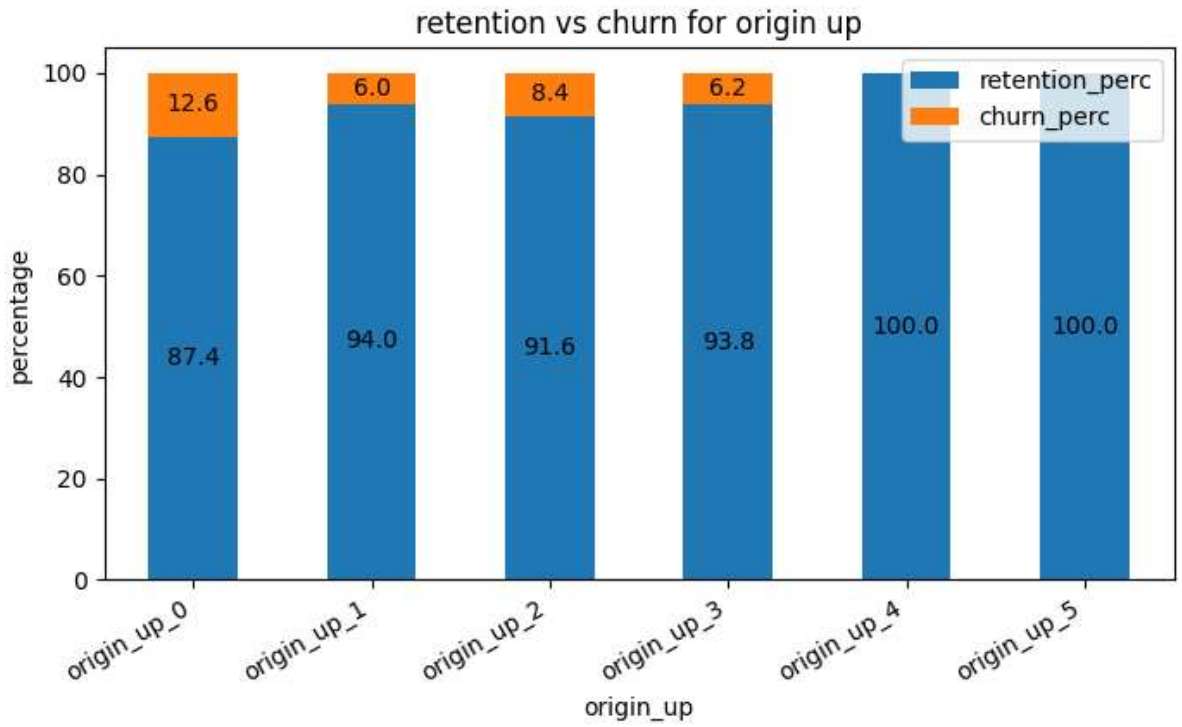
In [253…
```
origin_up_churn = pd.DataFrame()
origin_up_churn["origin_up"] = origin_up.index
origin_up_churn["num_of_customers"] = origin_up.values
origin_up_churn["churn_count"] = client_data.groupby("origin_up")["churn"].sum().va
origin_up_churn["retention_count"] = origin_up_churn["num_of_customers"].values-ori
origin_up_churn["churn_perc"] = (origin_up_churn["churn_count"].values/origin_up_ch
origin_up_churn["retention_perc"] = (origin_up_churn["retention_count"].values/orig
origin_up_churn["churn_perc"] = origin_up_churn["churn_perc"].apply(lambda x: round
origin_up_churn["retention_perc"] = origin_up_churn["retention_perc"].apply(lambda
origin_up_churn
```

Out[253]:

	origin_up	num_of_customers	churn_count	retention_count	churn_perc	retention_perc
0	origin_up_0	7097	893	6204	12.6	87.4
1	origin_up_1	4294	258	4036	6.0	94.0
2	origin_up_2	3148	264	2884	8.4	91.6
3	origin_up_3	64	4	60	6.2	93.8
4	origin_up_4	2	0	2	0.0	100.0
5	origin_up_5	1	0	1	0.0	100.0

In [254…
```
plot_stacked_bar(
    data=origin_up_churn,
    x_cols="origin_up",
    y_cols=["retention_perc", "churn_perc"],
    xlabel="origin_up", ylabel="percentage",
```

```
    title="retention vs churn for origin up"
)
```



`origin_up_0` has highest customer churn.

Price senstivity

In [255…
```
price_data["id"].value_counts().value_counts()
```

Out[255]:
```
count
12    15990
11       83
10       11
9         6
8         3
7         3
Name: count, dtype: int64
```

So every client has atleast 7 months price data.

In [256…
```
import numpy as np
def get_price_diff_mean(arr):
    diff_arr = np.diff(arr)
    return diff_arr.mean()
```

In [257…
```
price_off_peak_diff = price_data.groupby("id")["price_off_peak_var"].agg(get_price_
price_off_peak_diff
```

Out[257]:
```
id
0002203ffbb812588b632b9e628cc38d   -0.000563
0004351ebdd665e6ee664792efc4fd13   -0.000373
0010bcc39e42b3c2131ed2ce55246e3c    0.004586
0010ee3855fdea87602a5b7aba8e42de   -0.000911
00114d74e963e47177db89bc70108537   -0.000363
                                      ...
ffef185810e44254c3a4c6395e6b4d8a   -0.004567
fffac626da707b1b5ab11e8431a4d0a2   -0.000343
fffc0cacd305dd51f316424bbb08d1bd   -0.000160
fffe4f5646aa39c7f97f95ae2679ce64   -0.000854
ffff7fa066f1fb305ae285bb03bf325a   -0.000866
Name: price_off_peak_var, Length: 16096, dtype: float64
```

In [258…
```
price_peak_diff = price_data.groupby("id")["price_peak_var"].agg(get_price_diff_mea
price_peak_diff
```

Out[258]:
```
id
0002203ffbb812588b632b9e628cc38d   -0.000209
0004351ebdd665e6ee664792efc4fd13    0.000000
0010bcc39e42b3c2131ed2ce55246e3c    0.000000
0010ee3855fdea87602a5b7aba8e42de   -0.000465
00114d74e963e47177db89bc70108537    0.000000
                                      ...
ffef185810e44254c3a4c6395e6b4d8a   -0.003526
fffac626da707b1b5ab11e8431a4d0a2    0.000000
fffc0cacd305dd51f316424bbb08d1bd   -0.000337
fffe4f5646aa39c7f97f95ae2679ce64   -0.000449
ffff7fa066f1fb305ae285bb03bf325a   -0.000421
Name: price_peak_var, Length: 16096, dtype: float64
```

In [259…
```
price_mid_peak_diff = price_data.groupby("id")["price_mid_peak_var"].agg(get_price_
price_mid_peak_diff
```

Out[259]:
```
id
0002203ffbb812588b632b9e628cc38d    0.000317
0004351ebdd665e6ee664792efc4fd13    0.000000
0010bcc39e42b3c2131ed2ce55246e3c    0.000000
0010ee3855fdea87602a5b7aba8e42de    0.000069
00114d74e963e47177db89bc70108537    0.000000
                                      ...
ffef185810e44254c3a4c6395e6b4d8a   -0.002067
fffac626da707b1b5ab11e8431a4d0a2    0.000000
fffc0cacd305dd51f316424bbb08d1bd   -0.000666
fffe4f5646aa39c7f97f95ae2679ce64    0.000094
ffff7fa066f1fb305ae285bb03bf325a    0.000114
Name: price_mid_peak_var, Length: 16096, dtype: float64
```

In [260…
```
price_off_peak_fix_diff = price_data.groupby("id")["price_off_peak_fix"].agg(get_pr
price_off_peak_fix_diff
```

Out[260]:
```
id
0002203ffbb812588b632b9e628cc38d    1.481051e-02
0004351ebdd665e6ee664792efc4fd13    1.616171e-02
0010bcc39e42b3c2131ed2ce55246e3c    1.363636e-01
0010ee3855fdea87602a5b7aba8e42de    1.481051e-02
00114d74e963e47177db89bc70108537   -1.090909e-07
                                        ...
ffef185810e44254c3a4c6395e6b4d8a   -3.046226e-02
fffac626da707b1b5ab11e8431a4d0a2    1.616171e-02
fffc0cacd305dd51f316424bbb08d1bd    1.499232e-02
fffe4f5646aa39c7f97f95ae2679ce64    1.481051e-02
ffff7fa066f1fb305ae285bb03bf325a    1.481051e-02
Name: price_off_peak_fix, Length: 16096, dtype: float64
```

In [261…
```
price_peak_fix_diff = price_data.groupby("id")["price_peak_fix"].agg(get_price_diff
price_peak_fix_diff
```

Out[261]:
```
id
0002203ffbb812588b632b9e628cc38d    0.008886
0004351ebdd665e6ee664792efc4fd13    0.000000
0010bcc39e42b3c2131ed2ce55246e3c    0.000000
0010ee3855fdea87602a5b7aba8e42de    0.008886
00114d74e963e47177db89bc70108537    0.000000
                                      ...
ffef185810e44254c3a4c6395e6b4d8a   -0.036386
fffac626da707b1b5ab11e8431a4d0a2    0.000000
fffc0cacd305dd51f316424bbb08d1bd    0.009068
fffe4f5646aa39c7f97f95ae2679ce64    0.008886
ffff7fa066f1fb305ae285bb03bf325a    0.008886
Name: price_peak_fix, Length: 16096, dtype: float64
```

In [262…
```
price_mid_peak_fix_diff = price_data.groupby("id")["price_mid_peak_fix"].agg(get_pr
price_mid_peak_fix_diff
```

Out[262]:
```
id
0002203ffbb812588b632b9e628cc38d    0.005924
0004351ebdd665e6ee664792efc4fd13    0.000000
0010bcc39e42b3c2131ed2ce55246e3c    0.000000
0010ee3855fdea87602a5b7aba8e42de    0.005924
00114d74e963e47177db89bc70108537    0.000000
                                      ...
ffef185810e44254c3a4c6395e6b4d8a   -0.039349
fffac626da707b1b5ab11e8431a4d0a2    0.000000
fffc0cacd305dd51f316424bbb08d1bd    0.006106
fffe4f5646aa39c7f97f95ae2679ce64    0.005924
ffff7fa066f1fb305ae285bb03bf325a    0.005924
Name: price_mid_peak_fix, Length: 16096, dtype: float64
```

In [263…
```
price_diff = pd.DataFrame()
price_diff["id"] = price_off_peak_diff.index
price_diff["price_off_peak_diff"] = price_off_peak_diff.values
price_diff["price_peak_diff"] = price_peak_diff.values
price_diff["price_mid_peak_diff"] = price_mid_peak_diff.values
price_diff["price_off_peak_fix_diff"] = price_off_peak_fix_diff.values
price_diff["price_peak_fix_diff"] = price_peak_fix_diff.values
```

```
price_diff["price_mid_peak_fix_diff"] = price_mid_peak_fix_diff.values
price_diff.head()
```

Out[263]:

	id	price_off_peak_diff	price_peak_diff	price_mid_peak_diff	pri
0	0002203ffbb812588b632b9e628cc38d	-0.000563	-0.000209	0.000317	
1	0004351ebdd665e6ee664792efc4fd13	-0.000373	0.000000	0.000000	
2	0010bcc39e42b3c2131ed2ce55246e3c	0.004586	0.000000	0.000000	
3	0010ee3855fdea87602a5b7aba8e42de	-0.000911	-0.000465	0.000069	
4	00114d74e963e47177db89bc70108537	-0.000363	0.000000	0.000000	

In [264…

```
client_price_diff = pd.merge(client_data[["id", "churn"]], price_diff, on="id")
client_price_diff.head()
```

Out[264]:

	id	churn	price_off_peak_diff	price_peak_diff	price_mid_peak_d
0	24011ae4ebbe3035111d65fa7c15bc57	1	0.001823	-0.001628	-0.0065
1	d29c2c54acc38ff3c0614d0a653813dd	0	-0.000342	0.000000	0.0000
2	764c75f661154dac3a6c254cd082ea7d	0	-0.000425	0.000048	0.0000
3	bba03439a292a1e166f80264c16191cb	0	-0.000413	0.000000	0.0000
4	149d57cf92fc41cf94415803a877cb4b	0	-0.000563	-0.000209	0.0003

In [265…

```
client_price_diff["price_off_peak_diff_mean"] = (client_price_diff["price_off_peak_
client_price_diff["price_peak_diff_mean"] = (client_price_diff["price_peak_diff"]+c
client_price_diff["price_mid_peak_diff_mean"] = (client_price_diff["price_mid_peak_
client_price_diff_mean = client_price_diff.drop(["price_off_peak_diff", "price_off_
client_price_diff_mean.head()
```

Out[265]:

	id	churn	price_off_peak_diff_mean	price_peak_diff_mean	price
0	24011ae4ebbe3035111d65fa7c15bc57	1	0.169137	-1.107159	
1	d29c2c54acc38ff3c0614d0a653813dd	0	0.007910	0.000000	
2	764c75f661154dac3a6c254cd082ea7d	0	0.007869	0.000024	
3	bba03439a292a1e166f80264c16191cb	0	0.007874	0.000000	
4	149d57cf92fc41cf94415803a877cb4b	0	0.007124	0.004339	

In [266…

```
price_retention_churn = pd.DataFrame()
price_retention_churn["category"] = [
    "price_off_peak_inc",
    "price_off_peak_con_dec",
    "price_peak_inc",
    "price_peak_con_dec",
    "price_mid_peak_inc",
```

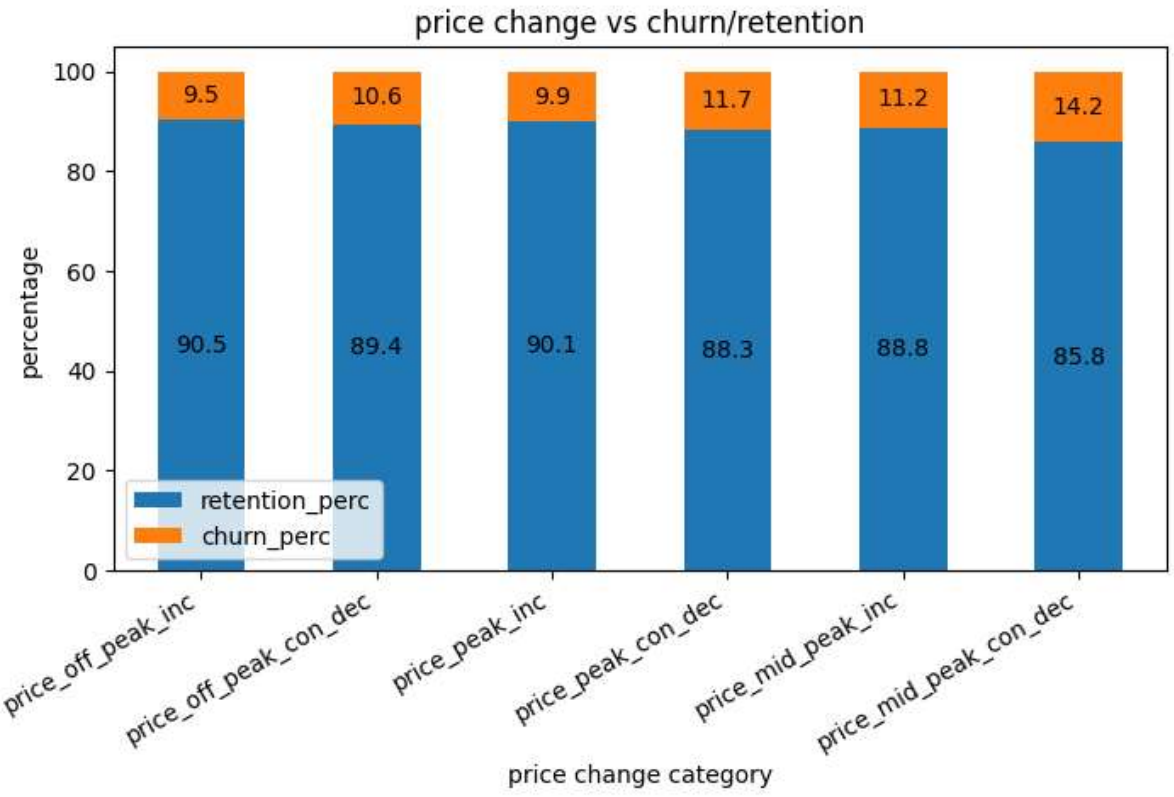
```
    "price_mid_peak_con_dec"
]

churn_list = []
retention_list = []

for column in ["price_off_peak_diff_mean", "price_peak_diff_mean", "price_mid_peak_
    churn_list.append(len(client_price_diff[(client_price_diff[column] > 0) & (clie
    retention_list.append(len(client_price_diff[(client_price_diff[column] > 0) & (
    churn_list.append(len(client_price_diff[(client_price_diff[column] < 0) & (clie
    retention_list.append(len(client_price_diff[(client_price_diff[column] < 0) & (

price_retention_churn["retention"] = retention_list
price_retention_churn["churn"] = churn_list
price_retention_churn["retention_perc"] = (price_retention_churn["retention"]/(pric
price_retention_churn["churn_perc"] = (price_retention_churn["churn"]/(price_retent

price_retention_churn.head(10)
```

Out[266]:

	category	retention	churn	retention_perc	churn_perc
0	price_off_peak_inc	9615	1009	90.502636	9.497364
1	price_off_peak_con_dec	3472	410	89.438434	10.561566
2	price_peak_inc	5004	551	90.081008	9.918992
3	price_peak_con_dec	1894	250	88.339552	11.660448
4	price_mid_peak_inc	4407	555	88.814994	11.185006
5	price_mid_peak_con_dec	536	89	85.760000	14.240000

In [267…

```
plot_stacked_bar(
    data=price_retention_churn,
    x_cols="category",
    y_cols=["retention_perc", "churn_perc"],
    xlabel="price change category",
    ylabel="percentage",
    title="price change vs churn/retention"
)
```



As we can see from the plots price increase doen't increase customer churn. Customers churned even when the price was nearly constant or even decreased.

Key findings and suggestions

Here is the summary of key findings and suggestions that can help in futher analysis

key findings

- Customer churn rate is almost 10%
- Price increase doesn't affect customer churn
- Customers churned even when the price was nearly constant or even decreased
- Sales channel coded as "foosdfpfkusacimwkcsosbicdxkicaua" has highest churn rate
- Origin up coded as "lxidpiddsbxsbosboudacockeimpuepw" has highest churn rate

suggestions

- Compare the pricing with competitors
- New competitors might be giving heavy discounts/more services to build their customer base
- Additional data on the average price/discount of all providers can be helpful

In [ ]: