Exploratory Data Analysis and Cleaning

By Shreyanth S

The Datasets

The dataset `ml_client_training_output.csv` named as `pco_output` contains:

> - `id` : contact id
> - `churn` : has the client churned over the next 3 months

The dataset `ml_price_training_hist_data.csv` named as `pco_hist` contains the history of energy and power consumption per client:

> - `id` : contact id
> - `price_date` : reference date
> - `price_off_peak_var` : price of energy for the 1st period
> - `price_peak_var` : price of energy for the 2nd period
> - `price_mid_peak_var` : price of energy for the 3rd period
> - `price_off_peak_fix` : price of power for the 1st period
> - `price_peak_fix` : price of power for the 2nd period
> - `price_mid_peak_fix` : price of power for the 3rd period

The dataset `ml_client_training_data.csv` contains:

> - `id` : contact id
> - `activity_new` : category of the company's activity. 419 unique values, remove `NaN`
> - `campaign_disc_elec` : code of the electricity campaign the customer last subscribed to. 0 non-null

- `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 consupmtion of the last month
- `date_activ` : date of activation of the contract
- `date_end` : registered date of the end of the contract
- `date_first_activ` : date of first contract of the client
- `date_modif_prod` : date of last modification of the product
- `date_renewal` : date of the next contract renewal
- `forecast_base_bill_ele` : forecasted electricity bill baseline for next month
- `forecast_base_bill_year` : forecasted electricity bill baseline for calendar year
- `forecast_bill_12m` : forecasted electricity bill baseline for 12 months
- `forecast_cons` : forecasted electricity consumption for next month
- `forecast_cons_12m` : forecasted electricity consumption for next 12 months
- `forecast_cons_year` : forecasted electricity consumption for next calendar year
- `forecast_discount_energy` : forecasted value of current discount
- `forecast_meter_rent_12m` : forecasted bill of meter rental for the next 12 months
- `forecast_price_energy_off_peak` : forecasted energy price for 1st period
- `forecast_price_energy_peak` : forecasted energy price for 2nd period
- `forecast_price_pow_off_peak` : forecasted power price for 1st period
- `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

Importing Libraries and Datasets

In this section we import the libraries of interest as well as the datasets.

```
# Import libraries
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import missingno as msno
from scipy.stats import zscore as zscore

# Read in dataset
url = 'https://raw.githubusercontent.com/ShreyanthBalasubramanian/Boston_Consulting_Group-
url2 = 'https://raw.githubusercontent.com/ShreyanthBalasubramanian/Boston_Consulting_Group
url3 = 'https://raw.githubusercontent.com/ShreyanthBalasubramanian/Boston_Consulting_Group

# list of dates
dt_lst = ['date_activ','date_end','date_first_activ','date_modif_prod','date_renewal']

# data importing
pco_main = pd.read_csv(url, parse_dates=dt_lst)#, index_col= 'date_activ')
pco_hist = pd.read_csv(url2, parse_dates=['price_date']) # Yearly history of consumption p
pco_output = pd.read_csv(url3)
pd.set_option('display.max_columns',None)
```

Data Exploration

The Client Output Dataset

From the output dataset we can derive a quick insight on customer retention.

```
# Replace the churn column with appropiate labels
pco_output['churn'] = pco_output['churn'].replace({0:'Stayed',1:'Churned'})

# Glimpse
pco_output.head()
```

	id	churn
0	24011ae4ebbe3035111d65fa7c15bc57	Churned
1	d29c2c54acc38ff3c0614d0a653813dd	Stayed
2	764c75f661154dac3a6c254cd082ea7d	Stayed
3	bba03439a292a1e166f80264c16191cb	Stayed
4	149d57cf92fc41cf94415803a877cb4b	Stayed

```
# What number of customers have churned in the last 3 months?
attrition_count = pco_output['churn'].value_counts()
print('Total Number of Churned Customers:\n', attrition_count)
```

```
Total Number of Churned Customers:
```

```
 Stayed     13187
Churned     1419
Name: churn, dtype: int64
```

```
# What is the proportion of customer attrition in the last 3 months?
attrition_rate = pco_output['churn'].value_counts() / pco_output.shape[0] * 100
print('Attrition rate: \n', attrition_rate)
```

```
Attrition rate:
 Stayed     90.284814
Churned     9.715186
Name: churn, dtype: float64
```

Facts

> - In the last 3 months 1,419 customers have churned
> - There are currently 13,187 active clients
> - Customer retention is 90% in the last 3 months
> - Customer attrition is 10% in the last 3 months

Observations

- Dataset has complete cases

The Price History Dataset

This dataset contains 1-year historical data for each client. It provides insights of the yearly activity of each client.

```
# Display the yearly consumption of energy and power of customers
pco_hist.head()
```

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

```
# Examing the structure of the dataframe
pco_hist.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  191643 non-null  float64
 3   price_peak_var      191643 non-null  float64
 4   price_mid_peak_var  191643 non-null  float64
 5   price_off_peak_fix  191643 non-null  float64
 6   price_peak_fix      191643 non-null  float64
 7   price_mid_peak_fix  191643 non-null  float64
dtypes: datetime64[ns](1), float64(6), object(1)
memory usage: 11.8+ MB
```

```
# Examine the descriptive statistics of the dataframe
pco_hist.describe()
```

	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix
count	191643.000000	191643.000000	191643.000000	191643.000000
mean	0.140991	0.054412	0.030712	43.325546
std	0.025117	0.050033	0.036335	5.437952
min	0.000000	0.000000	0.000000	-0.177779
25%	0.125976	0.000000	0.000000	40.728885
50%	0.146033	0.085483	0.000000	44.266930
75%	0.151635	0.101780	0.072558	44.444710
max	0.280700	0.229788	0.114102	59.444710

```
# Identify the nullity of the dataframe
missing_values_hist = pco_hist.isna().sum()
print('Total Missing Values:\n', missing_values_hist)
```

```
Total Missing Values:
 id                     0
price_date             0
price_off_peak_var    1359
price_peak_var        1359
price_mid_peak_var    1359
price_off_peak_fix    1359
price_peak_fix        1359
price_mid_peak_fix    1359
dtype: int64
```

```
# Identify the percentage of nullity in the dataframe for each collumn
missing_values_hist_perc = pco_hist.isnull().mean() * 100
print('Percentage of Missing Values:\n', missing_values_hist_perc)
```

```
Percentage of Missing Values:
 id                    0.000000
price_date            0.000000
price_off_peak_var    0.704138
price_peak_var        0.704138
price_mid_peak_var    0.704138
price_off_peak_fix    0.704138
price_peak_fix        0.704138
price_mid_peak_fix    0.704138
dtype: float64
```

Facts

- The average price of energy for the 1st period was: $0.14
- The average price of energy for the 2nd period was: $0.05
- The average price of energy for the 3rd period was: $0.03

The average price of energy was declining in the last year.

- The average power of power for the 1st period was: $43.32
- The average power of power for the 2nd period was: $10.69
- The average power of power for the 3rd period was: $6.45

The average price of power was declining in the last year.

Observations

- The columns `price_off_peak_fix`, `price_peak_fix`, and `price_mid_peak_fix` contain negative values. *These negative prices of power do not make sense.*

- The dataset `pco_hist` contains 1359 rows displaying `NaN` values on 6 variables except for `id` and `price_date`.

- The `price_..._var` and `price_..._fix` columns are missing **0.704%** of the data in each of them.

Note: `Pandas` *recognizes these* `NaN` *values and removes them when displaying descriptives statistics*.

Notice how the price of energy has a minimum value of zero. Perhaps some customers churned thereby making the consumption of energy zero.

The Main Dataset (Client)

This dataset contain more characteristics about each client's account and activity.

```
# Print header
pco_main.head()
```

	id	activity_new	campaign_disc_el
0	24011ae4ebbe3035111d65fa7c15bc57	esoiiifxdlbkcsluxmfuacbdckommixw	Nal
1	d29c2c54acc38ff3c0614d0a653813dd	NaN	Nal
2	764c75f661154dac3a6c254cd082ea7d	NaN	Nal
3	bba03439a292a1e166f80264c16191cb	NaN	Nal
4	149d57cf92fc41cf94415803a877cb4b	NaN	Nal

```
# printo info
pco_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16093 entries, 0 to 16092
Data columns (total 32 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              14606 non-null  object
 1   activity_new                    6551 non-null   object
 2   campaign_disc_ele               0 non-null      float64
 3   channel_sales                   10881 non-null  object
 4   cons_12m                        14606 non-null  float64
 5   cons_gas_12m                    14606 non-null  float64
 6   cons_last_month                 14606 non-null  float64
 7   date_activ                      14606 non-null  datetime64[ns]
 8   date_end                        14606 non-null  datetime64[ns]
 9   date_first_activ                3508 non-null   datetime64[ns]
 10  date_modif_prod                 14606 non-null  datetime64[ns]
 11  date_renewal                    14606 non-null  datetime64[ns]
 12  forecast_base_bill_ele          3508 non-null   float64
 13  forecast_base_bill_year         3508 non-null   float64
 14  forecast_bill_12m               3508 non-null   float64
 15  forecast_cons                   3508 non-null   float64
 16  forecast_cons_12m               14606 non-null  float64
 17  forecast_cons_year              14606 non-null  float64
 18  forecast_discount_energy        14606 non-null  float64
 19  forecast_meter_rent_12m         14606 non-null  float64
 20  forecast_price_energy_off_peak  14606 non-null  float64
 21  forecast_price_energy_peak      14606 non-null  float64
 22  forecast_price_pow_off_peak     14606 non-null  float64
 23  has_gas                         14606 non-null  object
```

```
 24  imp_cons                      14606 non-null  float64
 25  margin_gross_pow_ele          14606 non-null  float64
 26  margin_net_pow_ele            14606 non-null  float64
 27  nb_prod_act                   14606 non-null  float64
 28  net_margin                    14606 non-null  float64
 29  num_years_antig               14606 non-null  float64
 30  origin_up                     14606 non-null  object
 31  pow_max                       14606 non-null  float64
dtypes: datetime64[ns](5), float64(22), object(5)
memory usage: 3.9+ MB
```

```
# Identify the percentage of nullity in the dataframe for each collumn
missing_values_main_perc = pco_main.isnull().mean() * 100
print('Percentage of Missing Values:\n', missing_values_main_perc)
```

```
Percentage of Missing Values:
 id                               9.240042
activity_new                     59.292860
campaign_disc_ele               100.000000
channel_sales                    32.386752
cons_12m                          9.240042
cons_gas_12m                      9.240042
cons_last_month                   9.240042
date_activ                        9.240042
date_end                          9.240042
date_first_activ                 78.201703
date_modif_prod                   9.240042
date_renewal                      9.240042
forecast_base_bill_ele           78.201703
forecast_base_bill_year          78.201703
forecast_bill_12m                78.201703
forecast_cons                    78.201703
forecast_cons_12m                 9.240042
forecast_cons_year                9.240042
forecast_discount_energy          9.240042
forecast_meter_rent_12m           9.240042
forecast_price_energy_off_peak    9.240042
forecast_price_energy_peak        9.240042
forecast_price_pow_off_peak       9.240042
has_gas                           9.240042
imp_cons                          9.240042
margin_gross_pow_ele              9.240042
margin_net_pow_ele                9.240042
nb_prod_act                       9.240042
net_margin                        9.240042
num_years_antig                   9.240042
origin_up                         9.240042
pow_max                           9.240042
dtype: float64
```

```
# Examine the descriptive statistics of the main dataset
pco_main.describe()
```

	campaign_disc_ele	cons_12m	cons_gas_12m	cons_last_month	forecast_base
count	0.0	1.460600e+04	1.460600e+04	14606.000000	35
mean	NaN	1.592203e+05	2.809238e+04	16090.269752	3
std	NaN	5.734653e+05	1.629731e+05	64364.196422	6
min	NaN	0.000000e+00	0.000000e+00	0.000000	-3
25%	NaN	5.674750e+03	0.000000e+00	0.000000	
50%	NaN	1.411550e+04	0.000000e+00	792.500000	1
75%	NaN	4.076375e+04	0.000000e+00	3383.000000	3
max	NaN	6.207104e+06	4.154590e+06	771203.000000	125

Facts

- The average tenure of a client is 5 years
- The average net marging is $189

Observations

- The 14 columns contain negative minimum values
- The `activity_new` column is missing 59.3% of its data
- The `campaign_disc_ele` column is missing completely
- The `channel_sales` column is missing 32.3% of its data
- The `date_end` column is missing 9.2% of its data
- The `date_first_activ_` column is missing 78.2% of its data
- The `date_modif_prod` column is missing 9.2% of its data
- The `date_renewal` column is missing 9.2% of it data
- The `marging_gross_pow_ele` and `margin_net_pow_ele` columns are both missing 9.2% of its data
- The `net_margin` column is missing 9.2% of its data
- The `origin_up` column is missing 9.2% of its data
- The `pow_max` column is missing 9.2% of its data
- The `forecast_base_bill_ele`, `forecast_base_bill_year`, `forecast_bill_12m`, and `forecast_cons` columns are each missing 78.2% of its data

Data Cleaning and Imputation

Dealing with missing data

Workflow for treating missing values

1. Convert all missing values to null values
2. Analyze the amount and type of missingness in the data
3. Appropriately delete or impute missing values
4. Evaluate & compare the performance of the treated/imputed dataset

The `missingno` (imported as `msno`) package is great for visualizing missing data - we will be using:

- `msno.matrix()` visualizes a missingness matrix
- `msno.bar()` visualizes a missngness barplot
- `plt.show()` to show the plot

Is the data missing at random?

Types of missingness

1. Missing Completely at Random (MCAR)

> *Missingness has no relationship between any values, observed or missing*

2. Missing at Random (MAR)

> *There is a systematic relationship between missingness and other observed data, but not the missing data*

3. Missing Not at Random (MNAR)

> *There is a relationship between missingness and its values, missing or non-missing*

When and how to delete missing data?

Types of deletions

1. Pairwise deletion

> Pandas skips `NaN` whic is equivalent to pairwise deletion. Pairwise deletions minimize the amount of data loss and are hence preferred. However, it is also true that at several instances they might negatively affect our analysis.

2. Listwise deletion

> In listwise deletion the incomplete row is deleted, also called complete case analysis. The major disadvantage of listwise deletions is amount of data lost. Example: `df.dropna(subset=['column'], how='any',inplace=True)`

Note: *Both of these deletions are used only when the values are missing completely at random that is MCAR*

The Price History Dataset

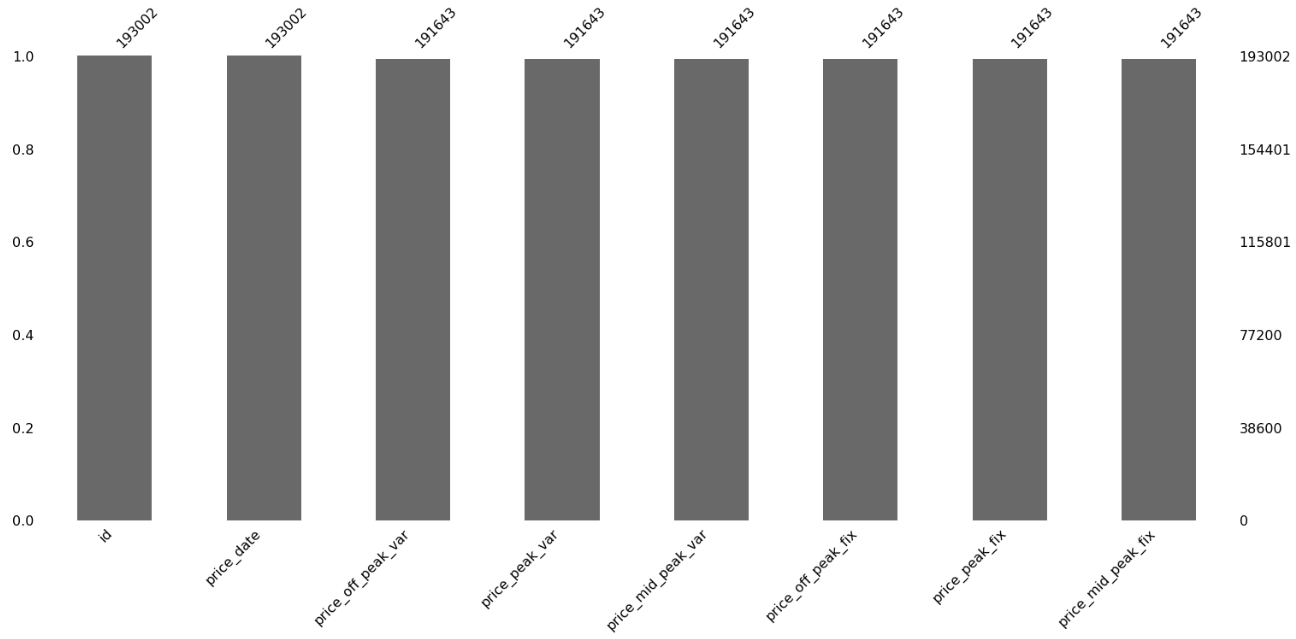
```
# Identify negative columns
negative_cols = ['price_off_peak_fix','price_peak_fix','price_mid_peak_fix']

# Convert to positive the negative columns in pco_hist
pco_hist[negative_cols] = pco_hist[negative_cols].apply(abs)

pco_hist.describe()
```

	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix
count	191643.000000	191643.000000	191643.000000	191643.000000
mean	0.140991	0.054412	0.030712	43.325563
std	0.025117	0.050033	0.036335	5.437816
min	0.000000	0.000000	0.000000	0.000000
25%	0.125976	0.000000	0.000000	40.728885
50%	0.146033	0.085483	0.000000	44.266930
75%	0.151635	0.101780	0.072558	44.444710
max	0.280700	0.229788	0.114102	59.444710

Visualizing the amount of missingness

```
# Visualize the completeness of the dataframe
msno.bar(pco_hist)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8676f74f90>
```



To the untrained eye, it might seem that there's no data missing. However, we estimated that 0.7% of the data in the price columns are missing. We can notice that the value counts at the top of each columns display a different amount.

```
# Visualize the locations of the missing values of the dataset
sorted = pco_hist.sort_values(by = ['id','price_date'])
msno.matrix(sorted)
```

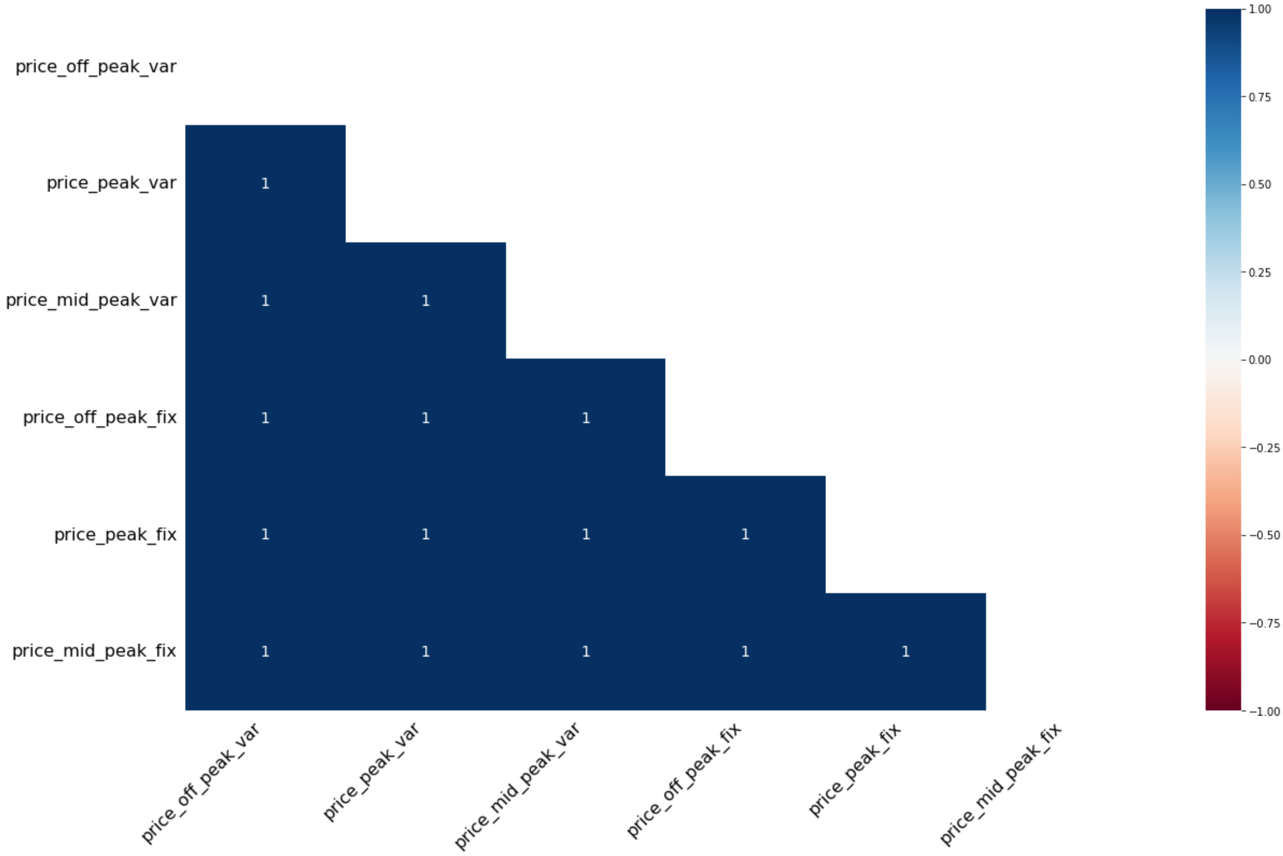
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f86745d1d50>
```

The nullity matrix describes the nullity of the dataset and appears blank wherever there are missing values.

The column on the very right summarizes the general shape of the data completeness and points out the row. Total count of columns at the bottom right.

```
# Visualize the correlation between the numeric variables of the dataframe
msno.heatmap(pco_hist)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f86745b0cd0>
```




```
# Identify the index of the IDs containing missing values.
hist_NAN_index = pco_hist[pco_hist.isnull().any(axis=1)].index.values.tolist()

# Obtain a dataframe with the missing values
```

```
pco_hist_missing = pco_hist.iloc[hist_NAN_index,:]

# Glimpse at the NaN cases of the pco_hist dataset
pco_hist_missing.head(10)
```

	id	price_date	price_off_peak_var	price_peak_va
75	ef716222bbd97a8bdfcbb831e3575560	2015-01-04	NaN	Na
221	0f5231100b2febab862f8dd8eaab3f43	2015-01-06	NaN	Na
377	2f93639de582fadfbe3e86ce1c8d8f35	2015-01-06	NaN	Na
413	f83c1ab1ca1d1802bb1df4d72820243c	2015-01-06	NaN	Na
461	3076c6d4a060e12a049d1700d9b09cf3	2015-01-06	NaN	Na
471	33bb3af90650ac2e9ecac6ff2c975a6b	2015-01-04	NaN	Na
472	33bb3af90650ac2e9ecac6ff2c975a6b	2015-01-05	NaN	Na
475	33bb3af90650ac2e9ecac6ff2c975a6b	2015-01-08	NaN	Na
476	33bb3af90650ac2e9ecac6ff2c975a6b	2015-01-09	NaN	Na
874	0e90101b08183cc9548e827e4b256f47	2015-01-12	NaN	Na

```
# extract the unique dates of missing data
date_lst = pco_hist_missing['price_date'].unique()
id_lst = pco_hist_missing['id'].unique()

# Create a time dataframe with the unique dates
time_df = pd.DataFrame(data=date_lst, columns=['price_date'] )

# Glimpse the time dataframe
time_df.sort_values(by=['price_date'])
```

	price_date
9	2015-01-01
11	2015-01-02
8	2015-01-03

Facts

- There is high correlation between the missingness in the numeric columns and is values, missing or non-missing
- There are 1359 clients who are missing price data at least in 1 month

3	2015-01-08

Observations

- After sorting the `pco_hist` dataset by `id` and `price_date`, we found that some columns are likely to be **MNAR**.
- The columns containing prices display strong positive correlation in the missingness suggests a case of **MNAR**.
- This event suggest that multicolinearity might be present in the dataset.

Imputations

Imputing time-series data requires a specialized treatment. Time-series data usually comes with special characteristics such trend, seasonality and cyclicality of which we can exploit when imputing missing values in the data.

In this particular dataset, there's not such thing as seasonality because it only has monthly data for one year.

Filling Time Series Data

```
# Make a copy of pco_hist dataset
pco_hist_ff = pco_hist.copy(deep=True)

# Print prior to imputing missing values
print(pco_hist_ff.iloc[hist_NAN_index,3:9].head())

# Fill NaNs using forward fill
pco_hist_ff.fillna(method = 'ffill', inplace=True)

print(pco_hist_ff.iloc[hist_NAN_index,3:9].head())
```

```
    price_peak_var  price_mid_peak_var  price_off_peak_fix  price_peak_fix  \
75             NaN                 NaN                 NaN             NaN
```

```
221            NaN                 NaN                 NaN              NaN
377            NaN                 NaN                 NaN              NaN
413            NaN                 NaN                 NaN              NaN
461            NaN                 NaN                 NaN              NaN

     price_mid_peak_fix
75                  NaN
221                 NaN
377                 NaN
413                 NaN
461                 NaN
     price_peak_var  price_mid_peak_var  price_off_peak_fix  price_peak_fix  \
75         0.000000            0.000000           44.266931        0.000000
221        0.000000            0.000000           44.266931        0.000000
377        0.087970            0.000000           44.266931        0.000000
413        0.102239            0.070381           40.565969       24.339581
461        0.000000            0.000000           44.266931        0.000000

     price_mid_peak_fix
75             0.000000
221            0.000000
377            0.000000
413           16.226389
461            0.000000
```

```
pco_hist_ff.describe()
```

	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix
count	193002.000000	193002.000000	193002.000000	193002.000000
mean	0.141006	0.054376	0.030689	43.326213
std	0.025091	0.050040	0.036333	5.431161
min	0.000000	0.000000	0.000000	0.000000
25%	0.125976	0.000000	0.000000	40.728885
50%	0.146033	0.085450	0.000000	44.266930
75%	0.151635	0.101780	0.072558	44.444710
max	0.280700	0.229788	0.114102	59.444710

```
# Merge output dataset with historical forward fill dataset
pco_hist_ff_merged = pco_hist_ff.merge(right=pco_output,on=['id'])
pco_hist_ff_merged.head()
```

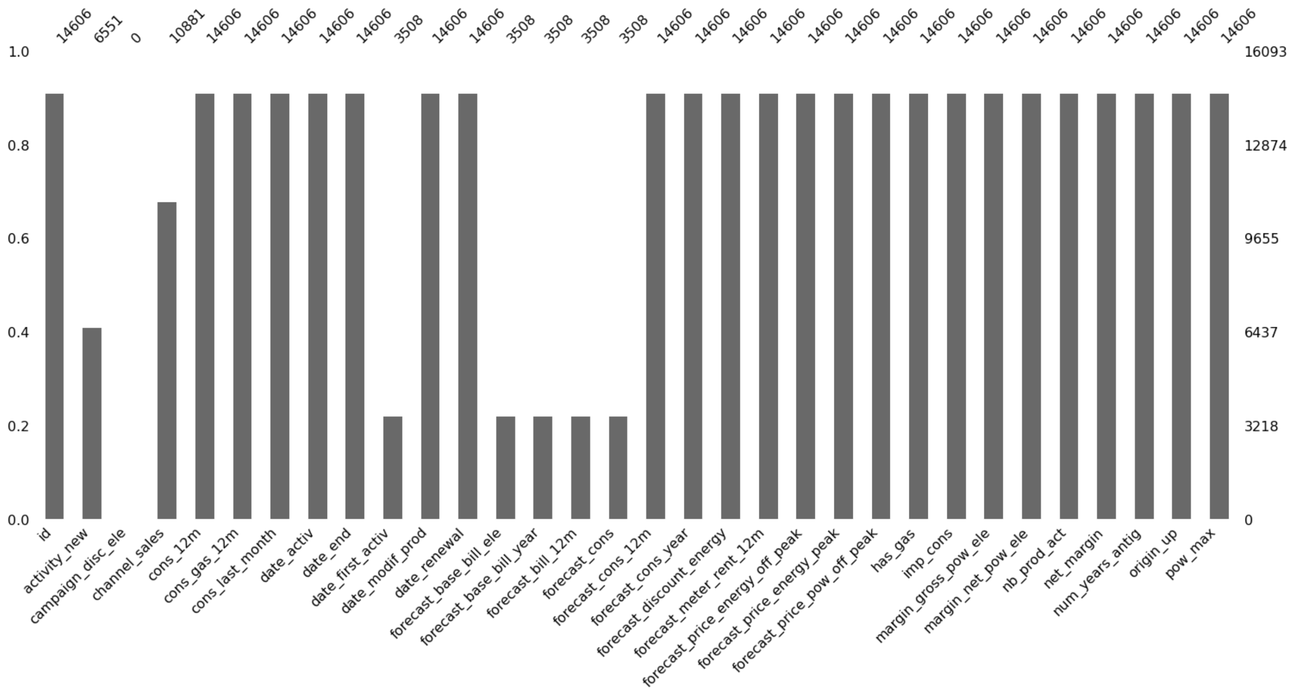
	id	price_date	price_off_peak_var	price_peak_var
0	038af19179925da21a25619c5a24b745	2015-01-01	0.151367	0.0
1	038af19179925da21a25619c5a24b745	2015-01-02	0.151367	0.0
2	038af19179925da21a25619c5a24b745	2015-01-03	0.151367	0.0
3	038af19179925da21a25619c5a24b745	2015-01-04	0.149626	0.0

The Main Dataset (Client)

Visualizing the amount of missingness

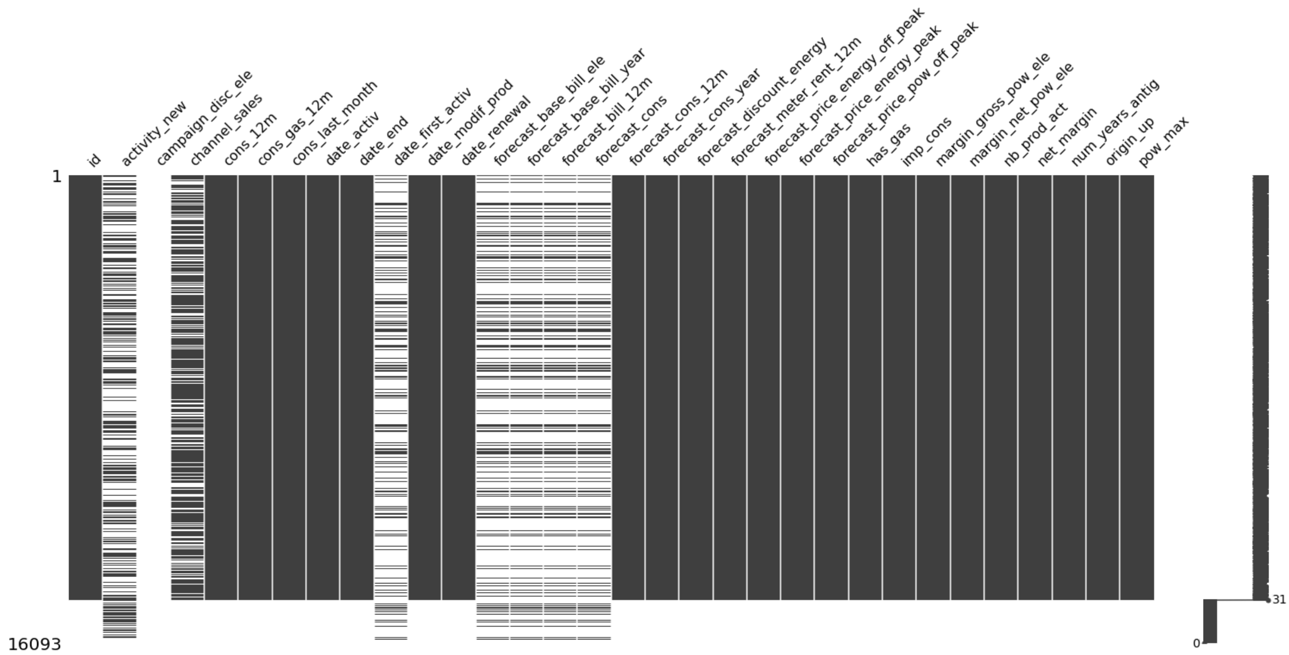
```
# Visualize the completeness of the dataframe
msno.bar(pco_main)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f86740f1590>
```



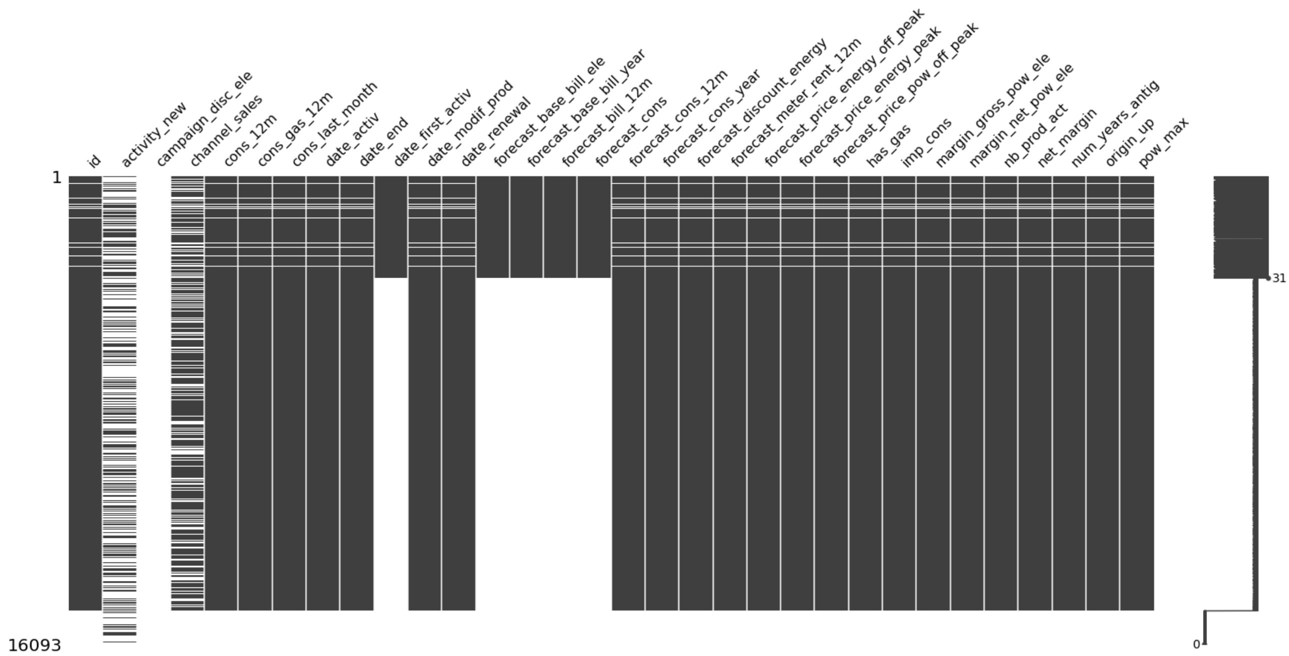

```
# Visualize the locations of the missing values of the dataset
msno.matrix(pco_main)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8673db1690>
```



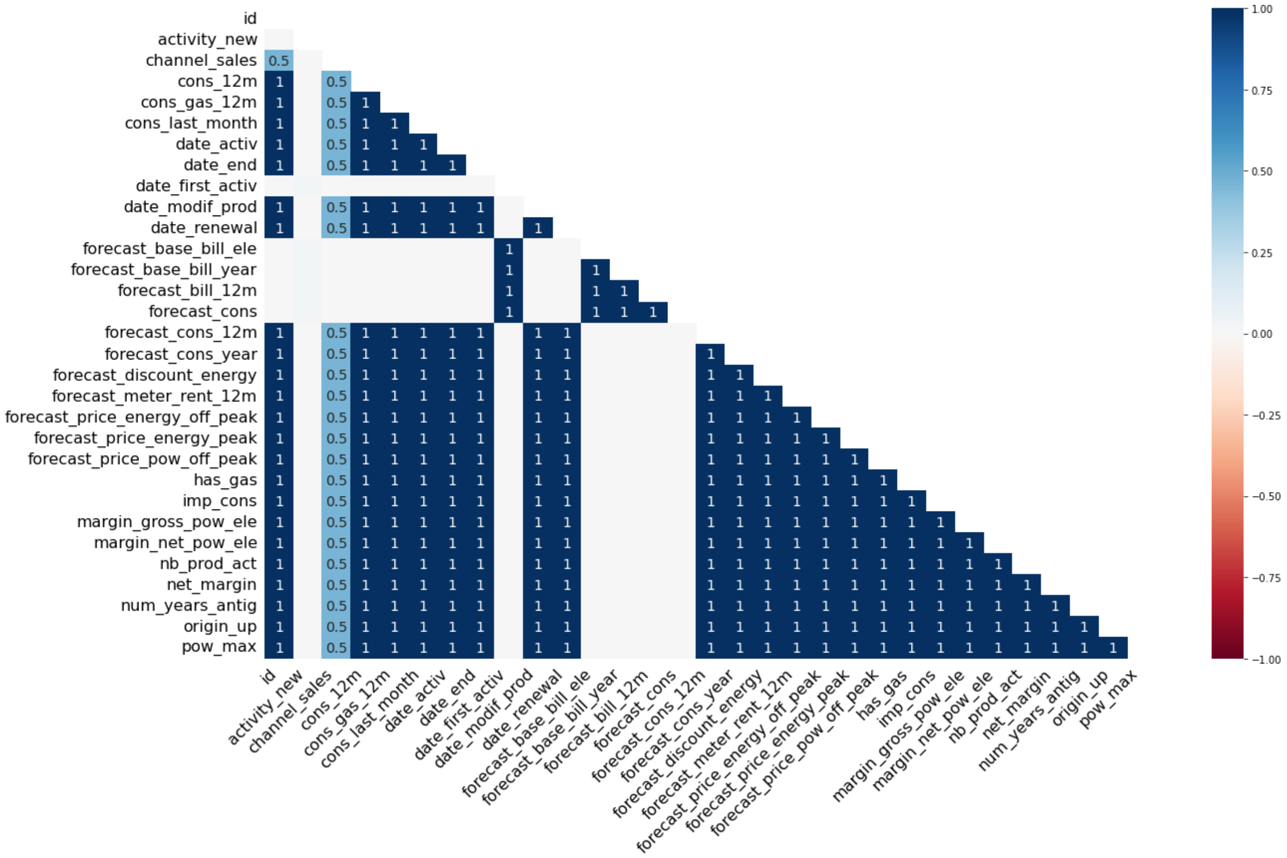

```
sorted_main = pco_main.sort_values('date_first_activ')
msno.matrix(sorted_main)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8673be2810>
```



```
msno.heatmap(pco_main)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f8673a66e10>

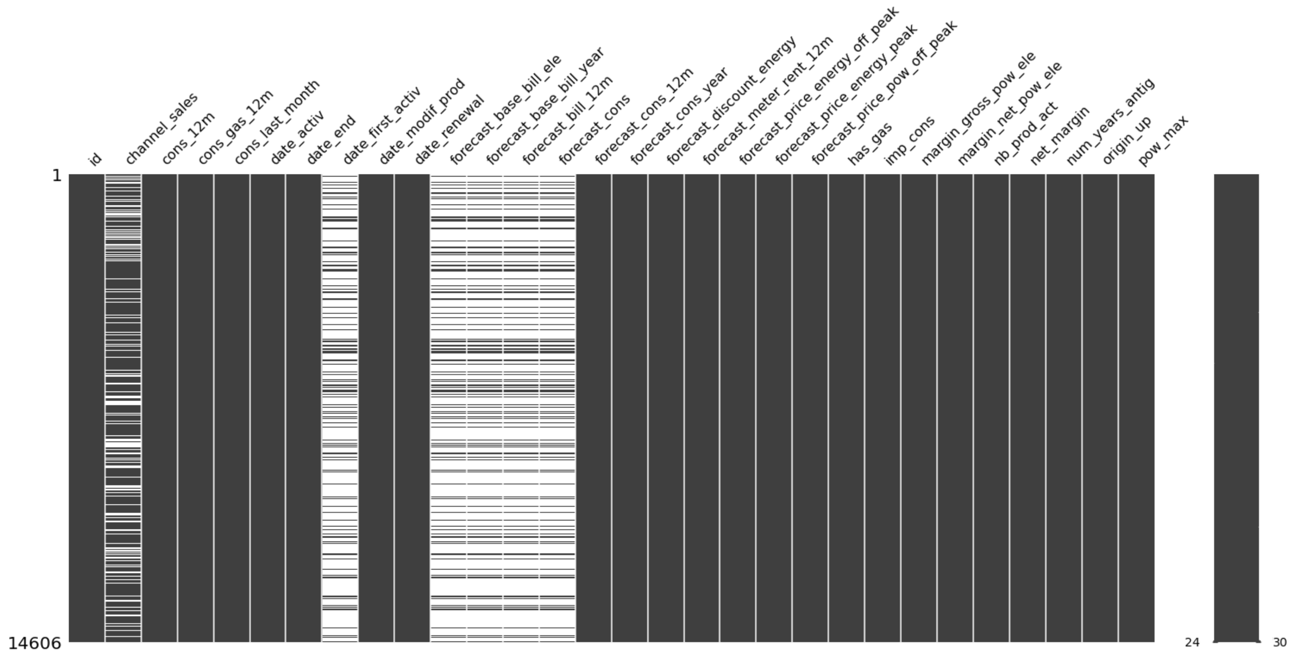


```
# Demonstrate why the date_activ column cannot replace completely date_first_activ
activity = ['date_activ','date_first_activ']

# Filter the columns of interest
pco_activity = pco_main[activity]

# Obtain only the complete cases
pco_activity_cc = pco_activity.dropna(subset=['date_first_activ'],how='any',inplace=False)

# Test whether two objects contain the same elements.
pco_activity_cc.date_activ.equals(pco_activity_cc.date_first_activ)

# Describe it
pco_activity_cc.describe(datetime_is_numeric=True) # Comparing dates in .describe() is dep
```

	date_activ	date_first_activ
count	3173	3508
mean	2011-01-19 11:27:33.072801792	2011-06-17 13:14:17.924743424
min	2003-01-08 00:00:00	2001-04-18 00:00:00
25%	2010-01-12 00:00:00	2010-08-01 00:00:00
50%	2011-02-12 00:00:00	2011-10-25 00:00:00
75%	2012-04-14 00:00:00	2012-06-28 00:00:00

```
# Drop the column activity_new and campaign_disc_elec
pco_main_drop = pco_main.drop(labels= ['activity_new','campaign_disc_ele'] , axis=1)

# Remove date_end date_modif_prod date_renewal origin_up pow_max margin_gross_pow_ele marg
brush = ['date_end','date_modif_prod','date_renewal','origin_up','pow_max','margin_gross_p
         'margin_net_pow_ele', 'net_margin','forecast_discount_energy','forecast_price_ene
         'forecast_price_energy_peak','forecast_price_pow_off_peak']
pco_main_drop.dropna(subset=brush, how='any',inplace=True)

msno.matrix(pco_main_drop)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8671c7a590>
```



Observations

- The variable `activity_new` is **MCAR** and has very low correlation with any of the variables. We can safely *drop* this column.
- The variable `campaign_disc_elec` is completely missing at random on all rows. We can get rid of this column. This suggests that subscribers are not subscribing through campaings offers.
- The variable `date_first_activ` cannot be replace by the values of the `date_activ` variable. **MAR**
- `net_margin` is showing strong correlation between `margin_gross_pow_elec` and `margin_net_pow_ele`. Multicolinearity is likely here.
- The variables `origin_up` and `pow_max` display no correlation with any variable and contain 0.54% and 0.01% of missingness respectively. These are **MCAR** and can be dropped listwise.
- `Forecast_base_bill_ele`, `forecast_base_bill_year`, `forecast_bill_12m` and `forecast_cons` variables are highly correlated with the `date_first_activ` variable's missingness. Accouting for 78% of missing values in the formerly mention columns and therfore are **MNAR**.
- Cannot replace the `date_first_activ` column with the `date_activ` column since in some of the cases the dates are not identical.

```
# Choose the columns without missing values
incomplete_cols = ['channel_sales','date_first_activ','forecast_base_bill_ele','forecast_b

complete_cols = [column_name for column_name in pco_main_drop.columns if column_name not i

pco_main_cc = pco_main_drop[complete_cols]

# Fix negative numeric variables
numeric = [column_name for column_name in pco_main_cc.columns
           if pco_main_cc[column_name].dtype == 'float64'
           or pco_main_cc[column_name].dtype == 'int64']

# Overwrite positive values on negative values
pco_main_cc[numeric] = pco_main_cc[numeric].apply(abs)

# Describe
pco_main_cc.describe()
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3641: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
  self[k1] = value[k2]
```

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons
count	1.460600e+04	1.460600e+04	14606.000000	14606.000000	14606.0
mean	1.592203e+05	2.809238e+04	16090.269752	1868.614880	1399.7
std	5.734653e+05	1.629731e+05	64364.196422	2387.571531	3247.7
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.0
25%	5.674750e+03	0.000000e+00	0.000000	494.995000	0.0

```
# Convert the has_gas column to  Yes/No
pco_main_cc['has_gas'] = pco_main_cc['has_gas'].replace({'t':'Yes','f':'No'})

# Merge the main dataset with the output dataset
pco_main_cc_merged = pco_main_cc.merge(right=pco_output,on=['id'])

# Convert the churn column to Churned/Stayed
pco_main_cc_merged['churn'] = pco_main_cc_merged['churn'].replace({1:'Churned',0:'Stayed'}
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
```

```
pco_main_cc_merged.head()
```

	id	cons_12m	cons_gas_12m	cons_last_month	date_a
0	24011ae4ebbe3035111d65fa7c15bc57	0.0	54946.0	0.0	2013-(
1	d29c2c54acc38ff3c0614d0a653813dd	4660.0	0.0	0.0	2009-(
2	764c75f661154dac3a6c254cd082ea7d	544.0	0.0	0.0	2010-(
3	bba03439a292a1e166f80264c16191cb	1584.0	0.0	0.0	2010-(
4	149d57cf92fc41cf94415803a877cb4b	4425.0	0.0	526.0	2010-(

```
# Obtain all the variables except for id
variables = [column_name for column_name in pco_main_cc_merged.columns if column_name != '

# Obtain all the categorical variables except for id
categorical = [column_name for column_name in variables if pco_main_cc_merged[column_name]

# Obtain all the Date Variables
dates = [column_name for column_name in variables if pco_main_cc_merged[column_name].dtype

# Obtain all the numeric columns
numeric = [column_name for column_name in variables
           if column_name not in categorical
           and column_name != 'id'
           and column_name != 'churn'
           and column_name not in dates]
```

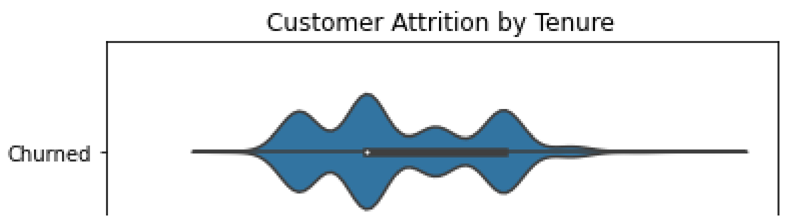
Data Visualization

Let's visualize what we've found.

The Client Output Dataset

```
# Calculate the zcores of tenure
tenure_zcores = zscore(a=pco_main_cc_merged['num_years_antig'])
# Convert to absolute values
abs_tenure_zscores = np.abs(tenure_zcores)
# Extract Columns of interest
churn_tenure = pco_main_cc_merged[['churn','num_years_antig']]
# Add z-score column
churn_tenure['z_score'] = list(abs_tenure_zscores)
# Remove outliers
churned_tenure_filtered = churn_tenure[churn_tenure['z_score'] < 3]
# Visualize tenure by retained customer and churner
vio = sns.violinplot( y=churned_tenure_filtered["churn"], x=churned_tenure_filtered["num_y
# Settings
vio.set(xlabel='Years', ylabel='')
vio.set_title("Customer Attrition by Tenure")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: SettingWithCopyWarni
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
```



Facts

- The median age of churners is 4 years
- Customers are more likely to churn during the 4th year than the 7th year
- The median age of retained customers is 5 years

2 4 6 8 10

The Main Dataset

```
# Most popular electricty campaign
ele_nm = pco_main_cc_merged.loc[(pco_main_cc_merged['churn']>='Stayed') & (pco_main_cc_mer

ele_nm.value_counts(subset=['origin_up'])
```

```
origin_up
lxidpiddsbxsbosboudacockeimpuepw    6155
kamkkxfxxuwbdslkwifmmcsiusiuosws    4002
ldkssxwpmemidmecebumciepifcamkci    2801
MISSING                               58
usapbepcfoloekilkwsdiboslwaxobdp       2
ewxeelcelemmiwuafmddpobolfuxioce       1
dtype: int64
```

```
# Highest netting electricity subscription campaign
print(ele_nm.groupby('origin_up')['net_margin'].agg('sum').sort_values(ascending=False))
```

```
origin_up
lxidpiddsbxsbosboudacockeimpuepw    1230753.01
kamkkxfxxuwbdslkwifmmcsiusiuosws     627964.96
ldkssxwpmemidmecebumciepifcamkci     564951.43
MISSING                               16386.00
usapbepcfoloekilkwsdiboslwaxobdp        250.40
ewxeelcelemmiwuafmddpobolfuxioce         46.22
Name: net_margin, dtype: float64
```

Facts

- The most popular electricity campaign is `lxidpiddsbxsbosboudacockeimpuepw` which has brought 6,155 current customers.

- The electricity campaign attributable to the highest total net margin is `lxidpiddsbxsbosboudacockeimpuepw`. Netting $1,230,753.01 in 2015.

Caveats

```
# Select current customers with positive net margins
top_customers = pco_main_cc_merged.loc[(pco_main_cc_merged['churn']>='Stayed') & (pco_mair

# Top 10 customers by net margin
top_customers.sort_values(by=['net_margin'],ascending=False).head(10)
```

	id	num_years_antig	net_margin
10718	d00e8a9951b5551d8f02e45f9ed2b0dd	3.0	10203.50
12348	818b8bca0a9d7668252d46b978169325	4.0	4346.37
7794	a3a739686fbd5ba8b4a21ec835507b6d	4.0	4305.79
12624	ee98a86efa759681cc59c7d4e0d0312f	4.0	3407.65
4876	9590c7a6100ae76ec078aa177ffb8d0d	3.0	3215.03
3478	e7bdc7743d73a9bf94cc3c6a293fca93	4.0	2711.19
4958	9a0411074f84ea385f555943f27a2d81	3.0	2653.59
7236	41b7c011f9d87044bb2e297264e95080	6.0	2625.38
10685	e5636f7ada7a80747af18b285632767e	10.0	2467.98
9345	078b4e5f8ea9a2f5f4c667f2d2236791	4.0	2340.78

These are the most profitable customers for PowerCo in terms of net margin. Beware most of them are within the likely tenure of attrition. Time for a marketing campaign!

✓ 0s completed at 10:56 PM