Exploratory Data Analysis Starter

Import packages

```
# Importing lib
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
# Shows plots in jupyter notebook
%matplotlib inline

# Set plot style
sns.set(color_codes=True)
```

Loading data with Pandas

We need to load client_data.csv and price_data.csv into individual dataframes so that we can work with them in Python. For this notebook and all further notebooks, it will be assumed that the CSV files will the placed in the same file location as the notebook. If they are not, please adjust the directory within the read_csv method accordingly.

```
#load the datasets
client_df= pd.read_csv('/content/client_data.csv')
price_df = pd.read_csv('/content/price_data.csv')
```

You can view the first 3 rows of a dataframe using the head method. Similarly, if you wanted to see the last 3, you can use tail(3)

```
client_df.head(3)
```

index	id	channel_sales	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_end	date_modif_pr
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	54946	0	2013-06-15	2016-06-15	2015-11-01
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	0	0	2009-08-21	2016-08-30	2009-08-21
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	0	0	2010-04-16	2016-04-16	2010-04-16

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

```
Warning: Total number of columns (26) exceeds max_columns (20) limiting to first (20) columns.
```

```
price_df.head(3)
```

index	id	price_date	price_off_peak_var	price_peak_var	price_mid_pe
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	

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

Descriptive statistics of data

Data types

It is useful to first understand the data that you're dealing with along with the data types of each column. The data types may dictate how you transform and engineer features.

```
# checking data shapes
client_df.shape
```

```
(14606, 26)
```

```
price_df.shape
```

```
(193002, 8)
```

```
# checking all of the columns
client_df.columns.values
```

```
array(['id', 'channel_sales', 'cons_12m', 'cons_gas_12m',
       'cons_last_month', 'date_activ', 'date_end', 'date_modif_prod',
       'date_renewal', 'forecast_cons_12m', 'forecast_cons_year',
       'forecast_discount_energy', 'forecast_meter_rent_12m',
       'forecast_price_energy_off_peak', 'forecast_price_energy_peak',
       'forecast_price_pow_off_peak', 'has_gas', 'imp_cons',
       'margin_gross_pow_ele', 'margin_net_pow_ele', 'nb_prod_act',
       'net_margin', 'num_years_antig', 'origin_up', 'pow_max', 'churn'],
      dtype=object)
```

```
price_df.columns.values
```

```
array(['id', 'price_date', 'price_off_peak_var', 'price_peak_var',
       'price_mid_peak_var', 'price_off_peak_fix', 'price_peak_fix',
       'price_mid_peak_fix'], dtype=object)
```

```
client_df.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
```

```
price_df.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
```

Statistics

Now let's look at some statistics about the datasets. We can do this by using the describe() method.

```
client_df.describe()
```

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_year	forecast_discount_energy	forecast_meter_rent_
count	1.460600e+04	1.460600e+04	14606.000000	14606.000000	14606.000000	14606.000000	14606.000
mean	1.592203e+05	2.809238e+04	16090.269752	1868.614880	1399.762906	0.966726	63.086
std	5.734653e+05	1.629731e+05	64364.196422	2387.571531	3247.786255	5.108289	66.165
min	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000
25%	5.674750e+03	0.000000e+00	0.000000	494.995000	0.000000	0.000000	16.180
50%	1.411550e+04	0.000000e+00	792.500000	1112.875000	314.000000	0.000000	18.795
75%	4.076375e+04	0.000000e+00	3383.000000	2401.790000	1745.750000	0.000000	131.030
max	6.207104e+06	4.154590e+06	771203.000000	82902.830000	175375.000000	30.000000	599.310

```
price_df.describe()
```

	price_off_peak_var	price_peak_var	price_mid_peak_var	price_off_peak_fix	price_peak_fix	price_mid_peak_fix
count	193002.000000	193002.000000	193002.000000	193002.000000	193002.000000	193002.000000
mean	0.141027	0.054630	0.030496	43.334477	10.622875	6.409984
std	0.025032	0.049924	0.036298	5.410297	12.841895	7.773592
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.125976	0.000000	0.000000	40.728885	0.000000	0.000000
50%	0.146033	0.085483	0.000000	44.266930	0.000000	0.000000
75%	0.151635	0.101673	0.072558	44.444710	24.339581	16.226389
max	0.280700	0.229788	0.114102	59.444710	36.490692	17.458221

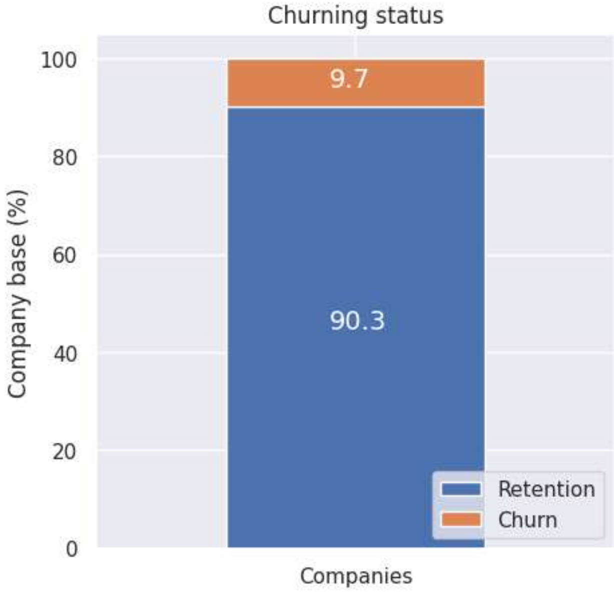
Data visualization

If you're working in Python, two of the most popular packages for visualization are matplotlib and seaborn. We highly recommend you use these, or at least be familiar with them because they are ubiquitous!

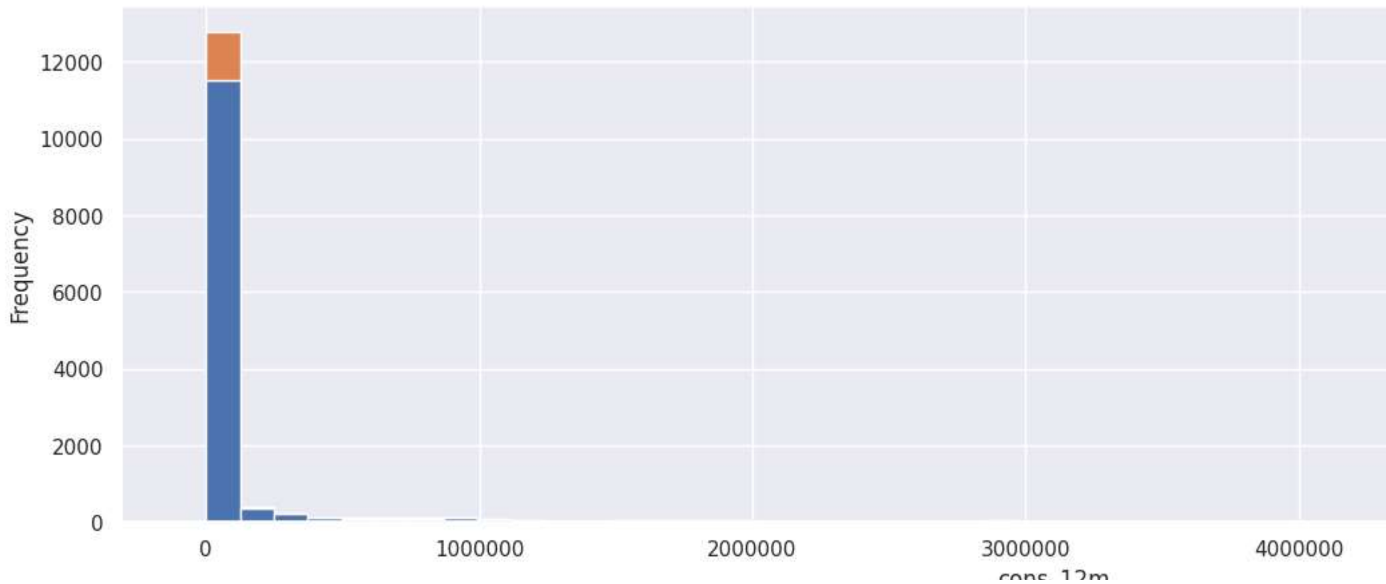
Below are some functions that you can use to get started with visualizations.

```
def plot_stacked_bars(dataframe, title_, size_=(18, 10), rot_=0, legend_="upper right"):
    """
    Plot stacked bars with annotations
    """
    ax = dataframe.plot(
        kind="bar",
        stacked=True,
        figsize=size_,
        rot=rot_,
        title=title_
    )

    # Annotate bars
    annotate_stacked_bars(ax, textsize=14)
```

```
    # Rename legend
    plt.legend(["Retention", "Churn"], loc=legend_)
    # Labels
    plt.ylabel("Company base (%)")
    plt.show()

def annotate_stacked_bars(ax, pad=0.99, colour="white", textsize=13):
    """
    Add value annotations to the bars
    """

    # Iterate over the plotted rectanges/bars
    for p in ax.patches:

        # Calculate annotation
        value = str(round(p.get_height(),1))
        # If value is 0 do not annotate
        if value == '0.0':
            continue
        ax.annotate(
            value,
            ((p.get_x()+ p.get_width()/2)*pad-0.05, (p.get_y()+p.get_height()/2)*pad),
            color=colour,
            size=textsize
        )

def plot_distribution(dataframe, column, ax, bins_=50):
    """
    Plot variable distirbution in a stacked histogram of churned or retained company
    """
    # Create a temporal dataframe with the data to be plot
    temp = pd.DataFrame({"Retention": dataframe[dataframe["churn"]==0][column],
    "Churn":dataframe[dataframe["churn"]==1][column]})
    # Plot the histogram
    temp[["Retention","Churn"]].plot(kind='hist', bins=bins_, ax=ax, stacked=True)
    # X-axis label
    ax.set_xlabel(column)
    # Change the x-axis to plain style
    ax.ticklabel_format(style='plain', axis='x')


churn = client_df[['id', 'churn']]
churn.columns = ['Companies', 'churn']
churn_total = churn.groupby(churn['churn']).count()
churn_percentage = churn_total / churn_total.sum() * 100
plot_stacked_bars(churn_percentage.transpose(), "Churning status", (5, 5), legend_="lower right")
```

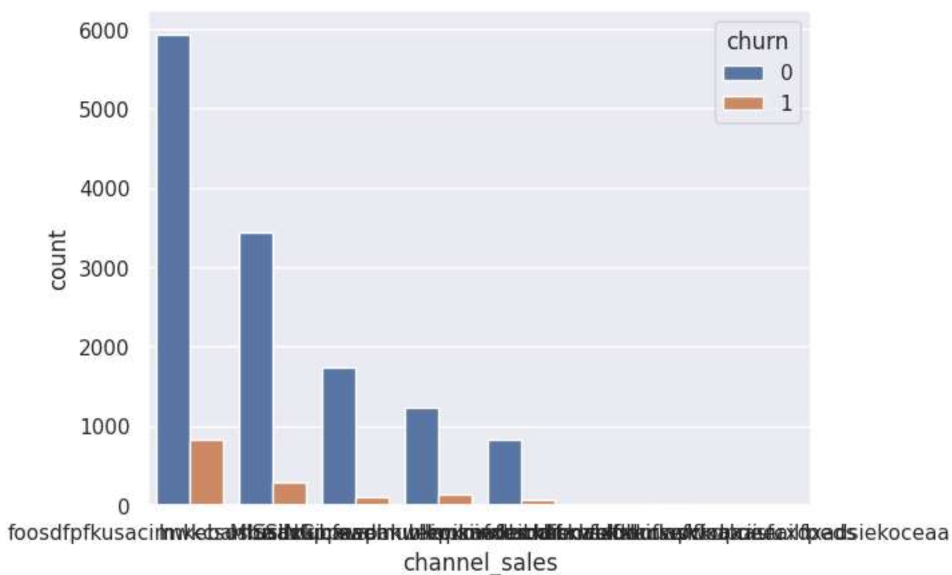


```
consumption = client_df[['id', 'cons_12m', 'cons_gas_12m', 'cons_last_month', 'imp_cons', 'has_gas', 'churn']]

fig, axs = plt.subplots(nrows=1, figsize=(18, 5))

plot_distribution(consumption, 'cons_12m', axs)
```




```
sns.countplot(x= 'channel_sales', hue = 'churn', data = client_df )
```

```
<Axes: xlabel='channel_sales', ylabel='count'>
```




```
for column in client_df.columns:
  if client_df[column].dtype == np.number:
    continue
  client_df[column] = LabelEncoder().fit_transform(client_df[column])
```

```
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
```

```
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
<ipython-input-102-a94138612816>:2: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current r
  if client_df[column].dtype == np.number:
```

```
client_df.dtypes
```

```
id                               int64
channel_sales                    int64
cons_12m                         int64
cons_gas_12m                     int64
cons_last_month                  int64
date_activ                       int64
date_end                         int64
date_modif_prod                  int64
date_renewal                     int64
forecast_cons_12m              float64
forecast_cons_year               int64
forecast_discount_energy       float64
forecast_meter_rent_12m        float64
forecast_price_energy_off_peak float64
forecast_price_energy_peak     float64
forecast_price_pow_off_peak    float64
has_gas                          int64
imp_cons                       float64
margin_gross_pow_ele           float64
margin_net_pow_ele             float64
nb_prod_act                      int64
net_margin                     float64
num_years_antig                  int64
origin_up                        int64
pow_max                        float64
churn                            int64
dtype: object
```

```
# scaled data
x = client_df.drop('churn', axis = 1)
y = client_df["churn"]

x = StandardScaler().fit_transform(x)
```

```
#split the data inti 80% traning 20% testing
x_train,x_test, y_train , y_test = train_test_split(x,y,test_size = 0.2, random_state = 42)
```

```
#crete the model
model = LogisticRegression()
#train model
model.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
#predict
prediction = model.predict(x_test)

print(prediction)
```

```
[0 0 0 ... 0 0 0]
```

```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.94      2617
           1       0.50      0.01      0.03       305
```

```
    accuracy                           0.90      2922
   macro avg       0.70      0.51      0.49      2922
weighted avg       0.86      0.90      0.85      2922
```

- The overall accuracy of the model is 90% with a weighted average F1-score of 85%. The precision, recall, and F1-score for class 0 are high, indicating that the model is good at predicting this class. However, the precision, recall, and F1-score for class are low, indicating that the model is not good at predicting this class.
- The macro average of precision, recall, and F1-score are 70%, 51%, and 49%, respectively. The macro average is calculated by taking the average of precision, recall, and F1-score across all classes without considering their support