```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

1 BCG EDA

Description of fields in the data set

Field name	Description	Field name	Description
id	contact id	forecast_price_energy_p1	forecasted energy price for 1st period
activity_new	category of the company's activity	forecast_price_energy_p2	forecasted energy price for 2nd period
campaign_disc_ele	code of the electricity campaign the customer last subscribed to	forecast_price_pow_p1	forecasted power price for 1st period
channel_sales	code of the sales channel	has_gas	indicated if client is also a gas client
cons_12m	electricity consumption of the past 12 months	imp_cons	current paid consumption
cons_gas_12m	gas consumption of the past 12 months	margin_gross_pow_ele	gross margin on power subscription
cons_last_month	electricity consumption of the last month	margin_net_pow_ele	net margin on power subscription
date_activ	date of activation of the contract	nb_prod_act	number of active products and services
date_end	registered date of the end of the contract	net_margin	total net margin
date_first_activ	date of first contract of the client	num_years_antig	antiquity of the client (in number of years)
date_modif_prod	date of last modification of the product	origin_up	code of the electricity campaign the customer first subscribed to
date_renewal	date of the next contract renewal	pow_max	subscribed power
forecast_base_bill_ele	forecasted electricity bill baseline for next month	price_date	reference date
forecast_base_bill_year	forecasted electricity bill baseline for calendar year	price_p1_var	price of energy for the 1st period
forecast_bill_12m	forecasted electricity bill baseline for 12 months	price_p2_var	price of energy for the 2nd period
forecast_cons	forecasted electricity consumption for next month	price_p3_var	price of energy for the 3rd period
forecast_cons_12m	forecasted electricity consumption for next 12 months	price_p1_fix	price of power for the 1st period
forecast_cons_year	forecasted electricity consumption for next calendar year	price_p2_fix	price of power for the 2nd period
forecast_discount_energy	forecasted value of current discount	price_p3_fix	price of power for the 3rd period
forecast_meter_rent_12m	forecasted bill of meter rental for the next 12 months	churned	has the client churned over the next 3 months

```python
train_data = pd.read_csv('ml_case_training_data.csv')
train_hist_data = pd.read_csv('ml_case_training_hist_data.csv')
output_data = pd.read_csv('ml_case_training_output.csv')
```

```python
# Merge the 'churn' to train data
full_train_data = train_data.merge(output_data, on='id')
```

```python
train_data.shape
```

```
(16096, 32)
```

```python
full_train_data.shape
```

```
(16096, 33)
```

```
total_null = full_train_data.isnull().sum().sort_values(ascending = False)
null_percentage = (full_train_data.isnull().sum() / full_train_data.isnull().co

missing_data = pd.concat([total_null, null_percentage],keys=['Total_Null','Null_
```

```
missing_data.head(20)
```

	Total_Null	Null_percentage
campaign_disc_ele	16096	1.000000
forecast_base_bill_ele	12588	0.782058
date_first_activ	12588	0.782058
forecast_cons	12588	0.782058
forecast_bill_12m	12588	0.782058
forecast_base_bill_year	12588	0.782058
activity_new	9545	0.593004
channel_sales	4218	0.262053
date_modif_prod	157	0.009754
forecast_discount_energy	126	0.007828
forecast_price_energy_p2	126	0.007828
forecast_price_pow_p1	126	0.007828
forecast_price_energy_p1	126	0.007828
origin_up	87	0.005405
date_renewal	40	0.002485
net_margin	15	0.000932
margin_net_pow_ele	13	0.000808
margin_gross_pow_ele	13	0.000808
pow_max	3	0.000186
date_end	2	0.000124

1.1 We can see that there are multiple variable present too much null valuse, which cannot provide vital information.

General should not exceed 15% null percentage

```python
# these are the variables contains too much null values, which may
drop_categories = missing_data[missing_data['Null_percentage'] > 0.15].index
drop_categories
```

```
Index(['campaign_disc_ele', 'forecast_base_bill_ele', 'date_first_activ',
       'forecast_cons', 'forecast_bill_12m', 'forecast_base_bill_year',
       'activity_new', 'channel_sales'],
      dtype='object')
```

```python
# data: data after drop too much null values
data = full_train_data.drop(drop_categories, axis = 1)
```

```
missing_data.loc[data.columns].sort_values(by='Null_percentage',ascending = Fal
```

	Total_Null	Null_percentage
date_modif_prod	157	0.009754
forecast_price_energy_p1	126	0.007828
forecast_discount_energy	126	0.007828
forecast_price_energy_p2	126	0.007828
forecast_price_pow_p1	126	0.007828
origin_up	87	0.005405
date_renewal	40	0.002485
net_margin	15	0.000932
margin_net_pow_ele	13	0.000808
margin_gross_pow_ele	13	0.000808
pow_max	3	0.000186
date_end	2	0.000124
imp_cons	0	0.000000
num_years_antig	0	0.000000
nb_prod_act	0	0.000000
id	0	0.000000
has_gas	0	0.000000
cons_12m	0	0.000000
forecast_meter_rent_12m	0	0.000000
forecast_cons_year	0	0.000000
forecast_cons_12m	0	0.000000
date_activ	0	0.000000
cons_last_month	0	0.000000
cons_gas_12m	0	0.000000
churn	0	0.000000

1.2 deal with null values

```python
# find the numeric columns
num_col = data._get_numeric_data().columns.tolist()
cat_col = set(data.columns) - set(num_col)
print("Num cols :{}, \n\nCat cols: {}".format(num_col, cat_col))
```

```
Num cols :['cons_12m', 'cons_gas_12m', 'cons_last_month', 'forecast_cons_12
m', 'forecast_cons_year', 'forecast_discount_energy', 'forecast_meter_rent_12
m', 'forecast_price_energy_p1', 'forecast_price_energy_p2', 'forecast_price_p
ow_p1', 'imp_cons', 'margin_gross_pow_ele', 'margin_net_pow_ele', 'nb_prod_ac
t', 'net_margin', 'num_years_antig', 'pow_max', 'churn'],

Cat cols: {'has_gas', 'origin_up', 'date_activ', 'date_modif_prod', 'id', 'da
te_renewal', 'date_end'}
```

```python
# Replace null with medain for numeric values
for col in num_col:
    data[col] = data[col].fillna(data[col].median())
```

```python
data[num_col].isnull().sum()
```

```
cons_12m                    0
cons_gas_12m                0
cons_last_month             0
forecast_cons_12m           0
forecast_cons_year          0
forecast_discount_energy    0
forecast_meter_rent_12m     0
forecast_price_energy_p1    0
forecast_price_energy_p2    0
forecast_price_pow_p1       0
imp_cons                    0
margin_gross_pow_ele        0
margin_net_pow_ele          0
nb_prod_act                 0
net_margin                  0
num_years_antig             0
pow_max                     0
churn                       0
dtype: int64
```

```python
# fill categoriacal column by adding an new "Unknown" Category
# Doing so is due to there are some variable such as date_end, if fill with mod
for col in cat_col:
    print(col)
    data[col].fillna('Unknown', inplace=True)
```

```
has_gas
origin_up
date_activ
date_modif_prod
id
date_renewal
date_end
```

```python
data[cat_col].isnull().sum()
```

```
has_gas            0
origin_up          0
date_activ         0
date_modif_prod    0
id                 0
date_renewal       0
date_end           0
dtype: int64
```

```
data.isnull().sum()
```

```
id                          0
cons_12m                    0
cons_gas_12m                0
cons_last_month             0
date_activ                  0
date_end                    0
date_modif_prod             0
date_renewal                0
forecast_cons_12m           0
forecast_cons_year          0
forecast_discount_energy    0
forecast_meter_rent_12m     0
forecast_price_energy_p1    0
forecast_price_energy_p2    0
forecast_price_pow_p1       0
has_gas                     0
imp_cons                    0
margin_gross_pow_ele        0
margin_net_pow_ele          0
nb_prod_act                 0
net_margin                  0
num_years_antig             0
origin_up                   0
pow_max                     0
churn                       0
dtype: int64
```

1.3 EDA

1.3.1 Detect Outliers

```
def detect_outlier(df, col):
    print(col)
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - (1.5 * iqr)
    upper_bound = q3 + (1.5 * iqr)
    l_outlier = df[col].apply(lambda x: x <= lower_bound).sum()
    u_outlier = df[col].apply(lambda x: x >= upper_bound).sum()
    print("lower outlier :{}, Upper outliers: {}".format(l_outlier,u_outlier))
```

```python
for col in num_col:
    detect_outlier(data, col)
```

```
cons_12m
lower outlier :3, Upper outliers: 2540
cons_gas_12m
lower outlier :13176, Upper outliers: 16090
cons_last_month
lower outlier :29, Upper outliers: 2469
forecast_cons_12m
lower outlier :9, Upper outliers: 1369
forecast_cons_year
lower outlier :10, Upper outliers: 1594
forecast_discount_energy
lower outlier :15517, Upper outliers: 16096
forecast_meter_rent_12m
lower outlier :1, Upper outliers: 383
forecast_price_energy_p1
lower outlier :100, Upper outliers: 367
forecast_price_energy_p2
lower outlier :0, Upper outliers: 0
forecast_price_pow_p1
lower outlier :102, Upper outliers: 749
imp_cons
lower outlier :10, Upper outliers: 1522
margin_gross_pow_ele
lower outlier :124, Upper outliers: 638
margin_net_pow_ele
lower outlier :173, Upper outliers: 616
nb_prod_act
lower outlier :12560, Upper outliers: 16096
net_margin
lower outlier :31, Upper outliers: 1181
num_years_antig
lower outlier :1, Upper outliers: 597
pow_max
lower outlier :1, Upper outliers: 2007
churn
lower outlier :14501, Upper outliers: 16096
```

```python
data.shape
```

```
(16096, 25)
```

1.3.2 Churned vs Non-Churned customers:

Consumption wise:

- 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

Relationship wise:

- net_margin: total net margin
- num_years_antig: antiquity of the client (in number of years)

```
# Churn Percentage
churn_percentage = data['churn'].sum() / data['churn'].count()
print("Churn Percentage: {:.2}%".format(churn_percentage))
```

```
Churn Percentage: 0.099%
```

```
data[['cons_12m','churn']].groupby(['churn'], as_index = False).mean().sort_val
```

	churn	cons_12m
0	0	206468.613406
1	1	88758.628213

```
data[['forecast_cons_12m','churn']].groupby(['churn'], as_index = False).mean()
```

	churn	forecast_cons_12m
1	1	2460.528978
0	0	2360.659598

```
data[['forecast_cons_year','churn']].groupby(['churn'], as_index = False).mean(
```

	churn	forecast_cons_year
1	1	1951.033856
0	0	1902.542032

From Consumption wise, we can find most of customer churn due to the uprise in forecast consumption

As for non-churned customers, mostly already devote lots of resources in the past.

```
data[['net_margin','churn']].groupby(['churn'], as_index = False).mean().sort_v
```

	churn	net_margin
1	1	250.378539
0	0	214.322518

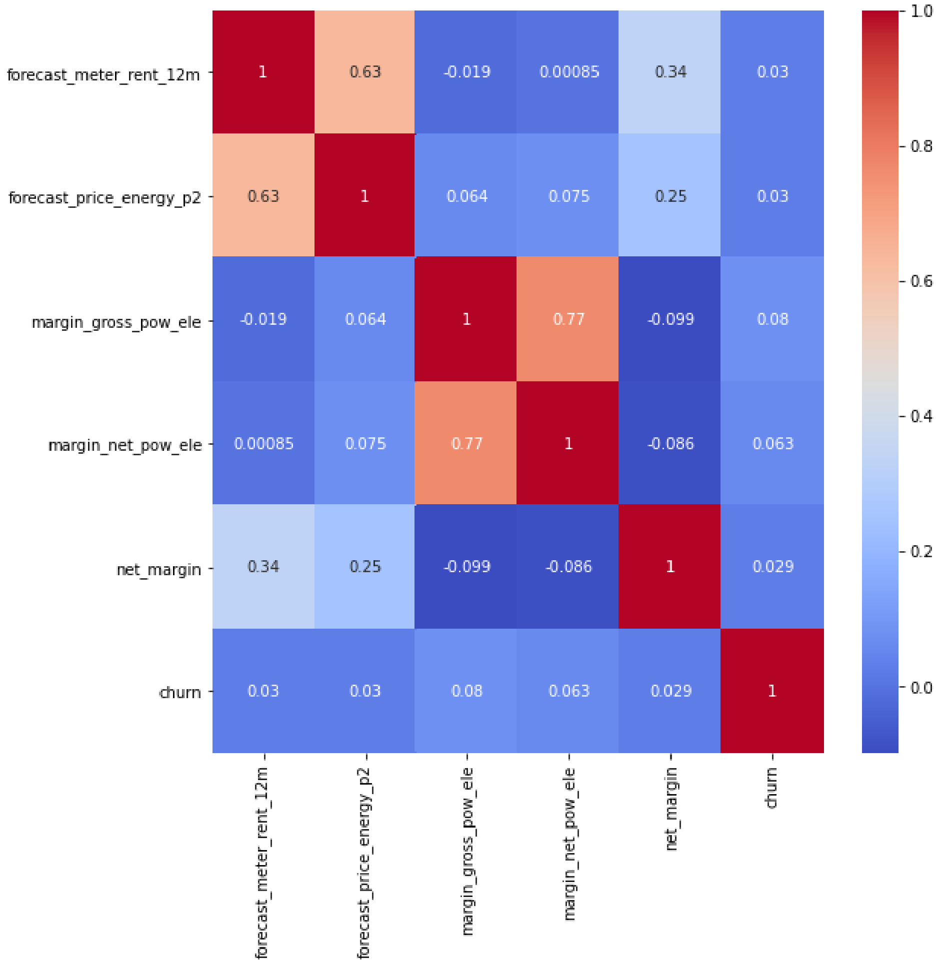
```
data[['num_years_antig','churn']].groupby(['churn'], as_index = False).mean().s
```

	churn	num_years_antig
0	0	5.070409
1	1	4.668966

1.4 model

```
import seaborn as sns

corrmat = data.corr()
top_corr_features = corrmat[abs(corrmat['churn'] > 0.02)].index
plt.figure(figsize=(9,9))
# Use these attributes to form the heatmap
# train_data[top_corr_features]
g = sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="coolwarm")
```




```
data.shape
```

```
(16096, 25)
```

```
# drop duplicate data entries
data = data.T.drop_duplicates().T
```

```
data.shape
```

```
(16096, 25)
```

1.5 Model

1.5.1 Utilize Lanel Encoder to Encode Data

```
from sklearn.preprocessing import LabelEncoder
labelcoder = LabelEncoder()
```

```
for col in cat_col:
    data[col] = labelcoder.fit_transform(data[col])
```

```
churn = data['churn']
churn=churn.astype('int')
```

```
train_data = data.drop(['churn'], axis = 1)
```

```
train_data.head()
```

	id	cons_12m	cons_gas_12m	cons_last_month	date_activ	date_en
0	4666	309275	0	10025	1737	283
1	2361	0	54946	0	1928	140
2	13250	4660	0	0	743	216
3	7430	544	0	0	941	80
4	11748	1584	0	0	928	63

5 rows × 24 columns

```
from sklearn.model_selection import train_test_split
# X_train,X_test,Y_train,Y_test = train_test_split(one_hot_ticket,Survived, tes
X_train,X_test,Y_train,Y_test = train_test_split(train_data,churn, test_size=.3
```

```python
# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_test, Y_test) * 100, 2)

# Stochastic Gradient Descent
sgd = SGDClassifier()
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)
acc_sgd = round(sgd.score(X_test, Y_test) * 100, 2)
acc_sgd

# Support Vector Machines
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_test, Y_test) * 100, 2)
acc_svc

# KNN
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_test, Y_test) * 100, 2)
acc_knn

# Gaussian Naive Bayes

gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_test, Y_test) * 100, 2)
acc_gaussian

# Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
Y_pred = perceptron.predict(X_test)
acc_perceptron = round(perceptron.score(X_test, Y_test) * 100, 2)
acc_perceptron
# Linear SVC

linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)
Y_pred = linear_svc.predict(X_test)
acc_linear_svc = round(linear_svc.score(X_test, Y_test) * 100, 2)
```

```
acc_linear_svc

# Decision Tree

decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_test, Y_test) * 100, 2)
acc_decision_tree

# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_test, Y_test) * 100, 2)
acc_random_forest
```

```
/home/brian/miniconda3/lib/python3.8/site-packages/sklearn/linear_model/_logi
stic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mo
dules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-le
arn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/home/brian/miniconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:985:
 ConvergenceWarning: Liblinear failed to converge, increase the number of ite
rations.
  warnings.warn("Liblinear failed to converge, increase "
```

```
90.27
```

1.6 Model Comparison:

We can see using Random Forest has the highest score

```python
# drop string
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree'],
    'Accuracy_Score': [acc_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_linear_svc, acc_decision_tree]})
models.sort_values(by='Accuracy_Score', ascending=False)
```

	Model	Accuracy_Score
3	Random Forest	90.29
0	Support Vector Machines	89.81
1	KNN	87.84
4	Naive Bayes	86.60
8	Decision Tree	82.29
5	Perceptron	80.55
7	Linear SVC	80.04
2	Logistic Regression	52.06
6	Stochastic Gradient Decent	20.19

1.7 Dimension Reduction

```python
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```python
myPCA = PCA(10)
mySVD = TruncatedSVD(10)
myLDA = LinearDiscriminantAnalysis(10)
```

```python
myPCA.fit(X_train)
```

```
PCA(n_components=10)
```

```
RX_train = myPCA.transform(X_train)
RX_test = myPCA.transform(X_test)
```

1.8 PCA

```
# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_test, Y_test) * 100, 2)
acc_random_forest
```

```
90.31
```

```
acc_random_forest
```

```
90.31
```

1.9 SVD

```
mySVD.fit(X_train)
```

```
TruncatedSVD(n_components=10)
```

```
RX_train = mySVD.transform(X_train)
RX_test = mySVD.transform(X_test)
```

```
# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_test, Y_test) * 100, 2)
acc_random_forest
```

```
90.31
```

1.10 We can observe that using SVD and PCA did'nt help at the accuracy of the data

```
from sklearn import metrics
```

```
print('Precision:', metrics.precision_score(Y_test, Y_pred))
print('Recall:', metrics.recall_score(Y_test, Y_pred))
print('F1:', metrics.f1_score(Y_test, Y_pred))
```

```
Precision: 0.9615384615384616
Recall: 0.0508130081300813
F1: 0.09652509652509651
```

```
fpr, tpr, threshold = metrics.roc_curve(Y_test, Y_pred)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```