Projet 3 OC : anticipation des besoins en consommation électrique de bâtiments

Présenté par : Guy Anthony Nama Nyam

Mentor : Julien Hendrick

5 décembre 2019

OPENCLASSROOMS

Sommaire

Contexte

- Les besoins en énergie mondiale croient fortement(2,3% en 2018 selon l'Agence internationale de l'énergie).

- Les facteurs qui pèsent dans cette évolution sont :
  - La forte croissance économique, notamment dans les pays en développement
  - L'accroissement de la population mondiale(9 à 10 milliards d'habitants à l'horizon 2050 selon ined)

- Cette demande entraine le rejet des émissions de CO_2 dans l'atmosphère qu'elle que soit la source.

- Le gaz naturel, l'électricité et la vapeur constituent les sources majeures d'émission de carbone des bâtiments non destinés à l'habitation pour la ville de Seattle.

Problématique

- Dans la quête du contrôle de la consommation d'énergie et l'émission de CO_2 de bâtiments se dressent la relevé minutieuse pour chaque bâtiment des informations explicatives année après année.
- Un outil d'aide ou d'agrégation à l'évaluation du rendement énergétique d'un bâtiment trouve un aperçu complet dans l'**ENERGY STAR Score**
- L'obtention de ces relevés sont coûteux, d'où les questions suivantes remontent :
  1. Comment prédire les émissions de CO_2 et de la consommation totale d'énergie à partir des relevés déjà réalisés ?
  2. Quel est l'intérêt de l'**ENERGY STAR Score** pour la prédiction d'émissions ?
- Nous apportons des réponses en proposant des modèles de régressions(prédiction).

Sommaire

Description des fichiers de données

- Les fichiers utilisés sont les relevés de consommation de la ville de Seattle en 2015 et 2016 téléchargeables depuis le site kaggle.com
- Ils ont été recuperés au format .csv et ayant le caractère de séparation par défaut(,).
- "2015-building-energy-benchmarking.csv" - le fichier de consommation de 2015 comporte 3340 lignes pour 47 colonnes
- "2016-building-energy-benchmarking.csv" le fichier de consommation de 2016 comporte 3376 lignes pour 46 colonnes
- 19 colonnes de différence entre les deux fichiers soit des différences de noms pour la même information, soit l'absence d'équivalence d'information, soit l'agrégation de plusieurs variables.

Description des fichiers de données


```
#détecter la colonne absente dans chacune des data
for c in list(donnees1.columns):
    if c not in list(donnees2.columns):
        print(c)
print("----------")
for c in list(donnees2.columns):
    if c not in list(donnees1.columns):
        print(c)
```

```
Location
OtherFuelUse(kBtu)
GHGEmissions(MetricTonsCO2e)
GHGEmissionsIntensity(kgCO2e/ft2)
Comment
2010 Census Tracts
Seattle Police Department Micro Community Policing Plan Areas
City Council Districts
SPD Beats
Zip Codes
----------
Address
City
State
ZipCode
Latitude
Longitude
Comments
TotalGHGEmissions
GHGEmissionsIntensity
```

Sommaire

Normalisation des colonnes des données

- Adapter les valeurs du champs OSEBuildingID constituant la clé primaire.

```
#adapter les OSEBuildingID
for i, row in donnees1.iterrows():
    donnees1.loc[i, "OSEBuildingID"] = str(row["OSEBuildingID"]) + "_2015"

for j, row in donnees2.iterrows():
    donnees2.loc[j, "OSEBuildingID"] = str(row["OSEBuildingID"]) + "_2016"
```

- Forme normale de la variable **Location** en **Address, City, State, ZipCode, Longitude, Latitude**

```
#Disloquer Location en adresse, cité, etc.
locations = donnees1['Location']
address = []
city = []
state = []
longitude = []
latitude = []
zipCode = []
for i in range(0, len(locations)):
    location = ast.literal_eval(locations[i])
    h_a = location['human_address']
    h_a = ast.literal_eval(h_a)
    address.append(h_a['address'])
    city.append(h_a['city'])
    state.append(h_a['state'])
    zipCode.append(h_a['zip'])
    longitude.append(location['longitude'])
    latitude.append(location['latitude'])

donnees1['Address'] = pd.Series(address)
donnees1['City'] = pd.Series(city)
donnees1['State'] = pd.Series(state)
donnees1['ZipCode'] = pd.Series(zipCode)
donnees1['Longitude'] = pd.Series(longitude).astype('float')
donnees1['Latitude'] = pd.Series(latitude).astype('float')
```

Normalisation des colonnes des données

- Uniformiser les noms des variables représentant la même information.

```
#renommer pour normaliser à donnees2 certaines colonnes
donnees1 = donnees1.rename(columns={"Comment" : "Comments", "GHGEmissionsIntensity(kgCO2e/ft2)" : "GHGEmissionsIntensity",
                                    "GHGEmissions(MetricTonsCO2e)" : "TotalGHGEmissions"})
```

- Supprimer les colonnes à valeurs constantes.

```
#Supprimer les colonnes de valeurs constantes
def del_var_const(data):
    var_const = []
    for f in list(data.columns):
        if len(data[f].value_counts()) == 1:
            data.drop(f, axis=1, inplace=True)
            var_const.append(f)
    print("Colonnes supprimées {}".format(var_const))

del_var_const(donnees1)
del_var_const(donnees2)
```

```
Colonnes supprimées ['DataYear', 'City', 'State']
Colonnes supprimées ['DataYear', 'City', 'State']
```

- Concaténer les lignes des deux fichiers par jointure naturelle sur les colonnes.

```
donnees = pd.concat([donnees1, donnees2], axis=0, join='inner', ignore_index=True)#fusion
```

Détection d'erreurs

- Détecter et supprimer des bâtiments d'habitation.

```
df["BuildingType"].value_counts()
```

```
NonResidential          2921
Multifamily LR (1-4)    2047
Multifamily MR (5-9)    1134
Multifamily HR (10+)     217
SPS-District K-12        197
Nonresidential COS       153
Campus                    46
Nonresidential WA          1
Name: BuildingType, dtype: int64
```

```
#supprimer les batiments d'habitation
indexLR = df[df["BuildingType"] == "Multifamily LR (1-4)"].index
indexMR = df[df["BuildingType"] == "Multifamily MR (5-9)"].index
indexHR = df[df["BuildingType"] == "Multifamily HR (10+)"].index

df.drop(indexLR, inplace = True)
df.drop(indexMR, inplace = True)
df.drop(indexHR, inplace = True)

df.shape
```

```
(3318, 43)
```

Détection d'erreurs

- Détecter et supprimer des variables à fortes valeurs manquantes :

```
#suprimer les variables avec beaucoup de valeurs manquantes
def del_var_useless(data, seuil=1):
    var_useless = []
    for f in list(data.columns):
        if (data[f].isna().sum()/len(data)) >= seuil:
            data.drop(f, axis=1, inplace=True)
            var_useless.append(f)
    print("Colonnes supprimées {}".format(var_useless))

del_var_useless(df, 0.9)
```

```
Colonnes supprimées ['YearsENERGYSTARCertified', 'Comments', 'Outlier']
```

- Supprimer la variable résultante **PropertyGFATotal**

```
df[df["PropertyGFATotal"] != (df["PropertyGFAParking"] + df["PropertyGFABuilding(s)"])]
```

OSEBuildingID	BuildingType	PrimaryPropertyType	PropertyName	TaxParcelIdentificationNumber	CouncilDistrictCode	Neighborhood	YearBuilt	NumberofBuilc

0 rows × 40 columns

```
df.drop(["PropertyGFATotal"], axis=1, inplace=True)
```

- Détecter et remplacer les valeurs négatives des variables numériques qui n'ont pas de sens par np.nan

```
print(df[df["Electricity(kBtu)"] < 0].shape[0])
df.loc[df["Electricity(kBtu)"] < 0, "Electricity(kBtu)"] = np.nan
```

```
1
```

Détection d'erreurs

- Uniformiser les valeurs des variables catégorielles.

```
#PrimaryPropertyType
for i, row in df.iterrows():
    f = "PrimaryPropertyType"
    valeur = df.loc[i,f]
    if valeur == "Distribution Center\n":
        df.loc[i,f] = "Distribution Center"
        continue
    if valeur == "Supermarket / Grocery Store":
        df.loc[i,f] = "Supermarket/Grocery Store"
        continue
    if valeur == "Self-Storage Facility\n":
        df.loc[i,f] = "Self-Storage Facility"
        continue
    if valeur == "Restaurant\n":
        df.loc[i,f] = "Restaurant"
        continue
    if valeur == "College/University":
        df.loc[i,f] = "University"
        continue
    if valeur == "Residence Hall/Dormitory":
        df.loc[i,f] = "Residence Hall"
        continue
```

```
#Neighborhood
for i, row in df.iterrows():
    f = "Neighborhood"
    valeur = df.loc[i,f].strip()
    if valeur == "North":
        df.loc[i,f] = "NORTH"
        continue
    if valeur == "DELRIDGE NEIGHBORHOODS" or valeur == "Delridge":
        df.loc[i,f] = "DELRIDGE"
        continue
    if valeur == "Northwest":
        df.loc[i,f] = "NORTHWEST"
        continue
    if valeur == "Ballard":
        df.loc[i,f] = "BALLARD"
        continue
    if valeur == "Central":
        df.loc[i,f] = "CENTRAL"
        continue
#DefaultData
df["DefaultData"] = df["DefaultData"].astype("str")
for i, row in df.iterrows():
    f = "DefaultData"
    valeur = df.loc[i,f].strip()
    if valeur == "No":
        df.loc[i,f] = "False"
        continue
    if valeur == "Yes":
        df.loc[i,f] = "True"
        continue
#ZipCode
df["ZipCode"] = df["ZipCode"].astype("str")
```

Traiter les valeurs manquantes

- Imputer les variables numériques par la médiane.
- Imputer les variables catégorielles par la plus fréquente de valeur.

```
categorical_cols = [cname for cname in df.columns if df[cname].dtype == "object"]
numerical_cols = [cname for cname in df.columns if df[cname].dtype in ['int64', 'float64']]

# Prétraitement de la donnée catégorielle
categorical_transformer = SimpleImputer(strategy='most_frequent')

# Prétraitement de la donnée numérique
numerical_transformer = SimpleImputer(strategy='median')

X_train_init, X_test_init = train_test_split(df, test_size=0.2, random_state=0)
SIZE = X_train_init.shape[0]

preprocessor = ColumnTransformer(
        transformers=[
            ('cat', categorical_transformer, categorical_cols),
            ('num', numerical_transformer, numerical_cols)
        ]
    )

imputed_X_train = pd.DataFrame(preprocessor.fit_transform(X_train_init))
imputed_X_test = pd.DataFrame(preprocessor.transform(X_test_init))

imputed_X_train.columns = categorical_cols + numerical_cols
imputed_X_test.columns = categorical_cols + numerical_cols

df = pd.concat([imputed_X_train, imputed_X_test], axis=0, join='inner', ignore_index=True)
```

Sommaire

Élément de base

- Statistiques de base

```
df.describe()
```

	CouncilDistrictCode	YearBuilt	NumberofBuildings	NumberofFloors	PropertyGFATotal	PropertyGFAParking	PropertyGFABuilding(s)	LargestPropertyUs
count	3318.000000	3318.000000	3318.000000	3318.000000	3.318000e+03	3318.000000	3.318000e+03	3.3
mean	4.365883	1961.536769	1.117239	4.121760	1.156672e+05	13303.304702	1.023639e+05	9.4
std	2.195304	32.709772	2.219178	6.560238	2.511222e+05	43596.620504	2.340749e+05	2.2
min	1.000000	1900.000000	0.000000	0.000000	1.128500e+04	-2.000000	-5.055000e+04	5.6
25%	2.000000	1930.000000	1.000000	1.000000	2.951225e+04	0.000000	2.850775e+04	2.5
50%	4.000000	1965.000000	1.000000	2.000000	4.924600e+04	0.000000	4.736800e+04	4.3
75%	7.000000	1989.000000	1.000000	4.000000	1.047860e+05	0.000000	9.447150e+04	8.8
max	7.000000	2015.000000	111.000000	99.000000	9.320156e+06	512608.000000	9.320156e+06	9.3

8 rows × 26 columns

- Supprimer les variables dupliquées pour **Electricity** et **NaturalGas**

```
np.corrcoef(df["NaturalGas(kBtu)"], df["NaturalGas(therms)"])[1,1]
```

```
0.9999999999999998
```

```
np.corrcoef(df["Electricity(kBtu)"], df["Electricity(kWh)"])[1,1]
```

```
0.9999999999999998
```

```
#Suppression des colonnes Electricity(kWh) et NaturalGas(therms)
df.drop(["Electricity(kWh)", "NaturalGas(therms)"], axis=1, inplace=True)
df.shape
```

```
(3318, 38)
```

Variable qualitative

- Aperçu complet de tous les types de propriétés utilisées.

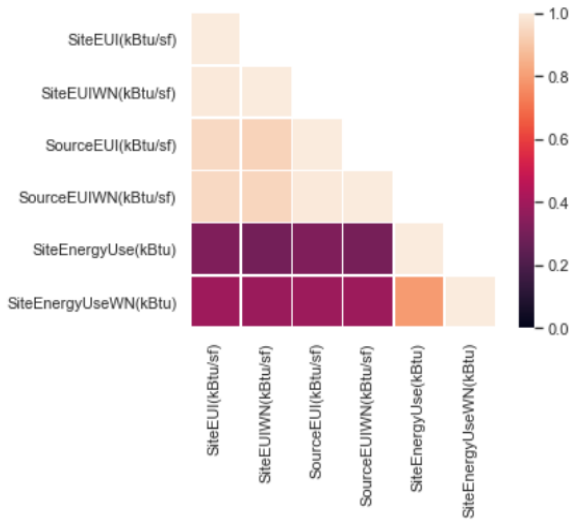
```
corpus = ' '.join(df["ListOfAllPropertyUseTypes"])
wordcloud = WordCloud(stopwords = STOPWORDS, background_color = 'white', height = 2000, width = 4000).generate(corpus)
plt.figure(figsize = (16,8))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

Variables quantitatives

- Analyse des énergies et leurs correspondants normalisées en fonction des conditions météorologiques(weather normalized).

```
fs = ['SiteEUI(kBtu/sf)', 'SiteEUIWN(kBtu/sf)', 'SourceEUI(kBtu/sf)','SourceEUIWN(kBtu/sf)',
      'SiteEnergyUse(kBtu)', 'SiteEnergyUseWN(kBtu)']

matrice_correlation(fs, df)
```




```
df.drop(["SiteEUIWN(kBtu/sf)", "SourceEUIWN(kBtu/sf)", "SiteEnergyUseWN(kBtu)"], axis=1, inplace=True)
df.shape
```

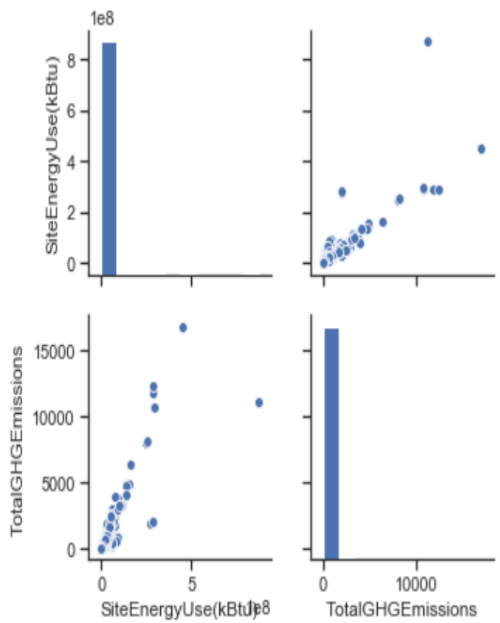
```
(3318, 35)
```

Variables quantitatives

- Corrélation entre les variables cibles **SiteEnergyUse(kBtu)** et **TotalGHGEmissions**

```
sns.set(style="ticks", color_codes=True)
sns.pairplot(df, vars=["SiteEnergyUse(kBtu)", "TotalGHGEmissions"])
```

```
<seaborn.axisgrid.PairGrid at 0x1501ba43978>
```

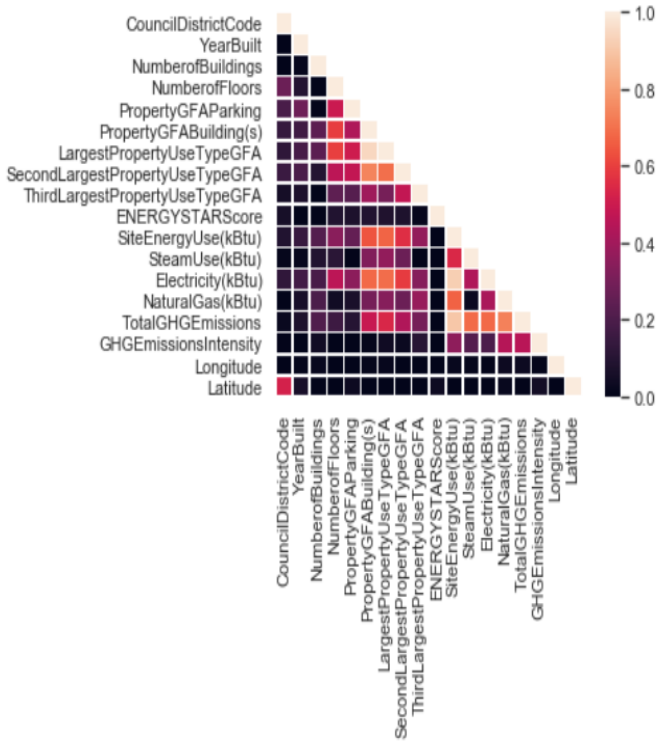


- Sans surprise, nous avons l'émission de CO_2 qui peut être expliquée par la consommation d'énergie.

Variables quantitatives

- Corrélation entre les variables numériques et interprétation

```
fs = ['CouncilDistrictCode', 'YearBuilt', 'NumberofBuildings', 'NumberofFloors', 'PropertyGFAParking','PropertyGFABuilding(s)',
      'LargestPropertyUseTypeGFA','SecondLargestPropertyUseTypeGFA','ThirdLargestPropertyUseTypeGFA','ENERGYSTARScore',
      'SiteEnergyUse(kBtu)', 'SteamUse(kBtu)', 'Electricity(kBtu)','NaturalGas(kBtu)','TotalGHGEmissions',
      'GHGEmissionsIntensity','Longitude', 'Latitude']
matrice_correlation(fs, df)
```



- Les émissions dépendent bien de la consommation d'énergie

Interprétation des corrélations

- Les émissions de CO_2 proviennent de toutes les sources d'énergies(pas d'énergies propres)
- Les sources utilisées par ordre décroissantes l'électricité, le gaz naturel et la vapeur.
- La consommation d'énergie dépend davantage des bâtiments d'utilisation(bureau, entrepot, etc...) que des parkings.
- La variable **ENERGYSTARScore** ne présente aucune explication avec les variables de consommation d'énergie et de d'émission de CO_2 encore moins avec les sources d'énergies. Ceci peut s'expliquer car elle dépend des bâtiments à l'échelle nationale.

Transformation des variables numériques

- Mesure d'asymétrie

```
print(df[numerical_cols].skew())
```

```
CouncilDistrictCode               -0.001385
YearBuilt                         -0.286850
NumberofBuildings                 20.073400
NumberofFloors                     6.071878
PropertyGFAParking                 5.265861
PropertyGFABuilding(s)             5.432046
LargestPropertyUseTypeGFA          5.309045
SecondLargestPropertyUseTypeGFA    5.960297
ThirdLargestPropertyUseTypeGFA    16.366270
ENERGYSTARScore                   -1.186056
SiteEUI(kBtu/sf)                   3.632820
SiteEUIWN(kBtu/sf)                 3.523221
SourceEUI(kBtu/sf)                 4.667560
SourceEUIWN(kBtu/sf)               4.640176
SiteEnergyUse(kBtu)               10.271318
SiteEnergyUseWN(kBtu)             10.452892
SteamUse(kBtu)                    21.265483
Electricity(kBtu)                  9.393663
NaturalGas(kBtu)                  21.318591
TotalGHGEmissions                 14.604245
GHGEmissionsIntensity              4.683783
Longitude                         -0.063001
Latitude                           0.236316
dtype: float64
```

- Appliquer le log(1+x) aux variables cibles

```
for f in ["TotalGHGEmissions", "SiteEnergyUse(kBtu)"]:
    df[f] = np.log1p(df[f])
```

Sommaire

Spécificités communes au deux problématiques

- Enlever la variable considérée comme l'identifiant : **OSEBuildingID**
- Enlever les variables qui ne sont pas toujours disponibles au moment de la prédiction(fuite cible ou target leakage) : **SiteEUI(kBtu/sf)**, **SourceEUI(kBtu/sf)**, **GHGEmissionsIntensity**
- Transformer les valeurs booléennes "True" et "False" de la variable **DefaultData** par respectivement 1 et 0
- Binarisation des variables **Electricity(kBtu)**, **SteamUse(kBtu)**, **NaturalGas(kBtu)** pour spécifier leur présence :

$$\begin{cases} 1 & \text{si valeur supérieur à 0} \\ 0 & \textit{sinon} \end{cases}$$

Spécificités communes au deux problématiques


```
#Transformer les valeurs de la variable DefaultData: True par 1 et False par 0
X.loc[X["DefaultData"] == "True", "DefaultData"] = 1
X.loc[X["DefaultData"] == "False", "DefaultData"] = 0
X["DefaultData"] = X["DefaultData"].astype("float64")

#traiter les variables Electricity, NaturalGas, SteamUse comme binaire; 1 si val > 0 et 0 sinon
X.loc[X["SteamUse(kBtu)"] > 0, "SteamUse(kBtu)"] = 1
X.loc[X["Electricity(kBtu)"] > 0, "Electricity(kBtu)"] = 1
X.loc[X["NaturalGas(kBtu)"] > 0, "NaturalGas(kBtu)"] = 1
```

- Sectionner l'ensemble des variables en trois groupes :
  1. Les variables numériques.
  2. Les variables catégorielles à valeurs uniques élevées.
  3. Les variables catégorielles à valeurs uniques faibles.

```
#variables catégorielles
def categorical_cols_split(X, num_split):
    categorical_cols_label = []
    categorical_cols_other = []
    for cname in X.columns:
        if X[cname].dtype == "object":
            if X[cname].nunique() > num_split:
                categorical_cols_label.append(cname)
            else:
                categorical_cols_other.append(cname)
    return categorical_cols_label, categorical_cols_other

categorical_cols_label, categorical_cols_other = categorical_cols_split(X, 50)

categorical_cols = categorical_cols_label + categorical_cols_other

#variables numériques
numerical_cols = [cname for cname in X.columns if X[cname].dtype in ['int64', 'float64']]
```

- Insérer les variables numériques dans le bunch des modèles.

Spécificités communes au deux problématiques

- Encodage des variables à valeurs uniques élevées par le **Label Encoder** et ajout dans le bunch des variables.

```
#Label encoder
X_tr = X_train.copy()
X_te = X_test.copy()

label_encoder = LabelEncoder()

for col in categorical_cols_label:
    bunch_train[col] = label_encoder.fit_transform(X_tr[col])
    lists_enc = list(label_encoder.classes_)
    lists_enc_not = []
    for i, row in X_te.iterrows():
        val = X_te.loc[i, col]
        if val not in lists_enc:
            lists_enc_not.append(val)
    label_encoder.classes_ = np.array(lists_enc + lists_enc_not)
    bunch_test[col] = label_encoder.transform(X_te[col])
```

- Listes d'encodeurs

```
#listes d'encodeurs
OH_encoder = OneHotEncoder(handle_unknown='ignore')#One-Hot-Encoding
CE_encoder = ce.CountEncoder()#Count Encoding
TE_encoder = ce.TargetEncoder(cols=categorical_cols_other)#Target Encoding
CatB_encode = ce.CatBoostEncoder(cols=categorical_cols_other)#CatBoost Encoding
```

- Listes d'estimateurs

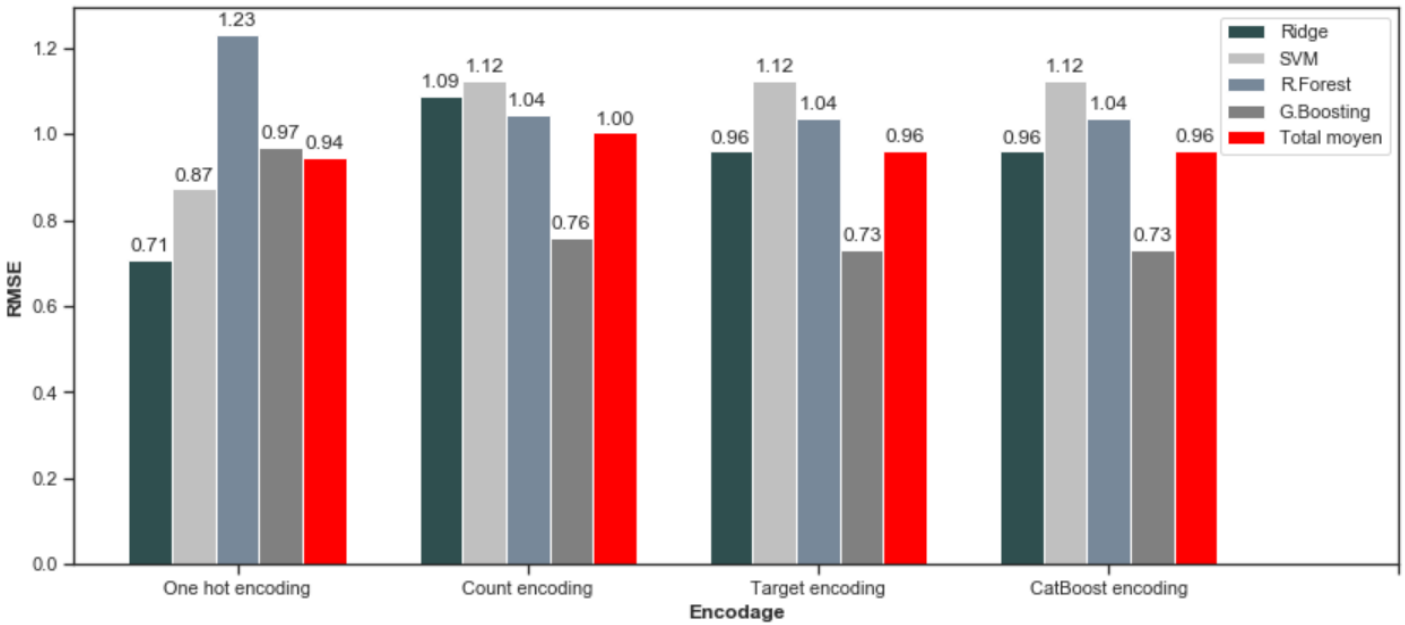
```
#listes d'estimateurs
lr = Ridge(alpha=1.0)#Linear Regressor Ridge
svr = SVR(kernel="rbf", gamma='scale', C=1.0)#SVM regressor
rf = RandomForestRegressor(max_depth=2, n_estimators=100, random_state=0)#Random Forest
gb = GradientBoostingRegressor(max_depth=2, n_estimators=100, learning_rate=0.1, loss='ls')#Gradient Boosting
```

Émission de CO_2 : choix de l'encodeur

- Variable expliqué et variables explicatives

```
y = df["TotalGHGEmissions"]
X = df.drop(['TotalGHGEmissions', "OSEBuildingID", "SiteEUI(kBtu/sf)", "SourceEUI(kBtu/sf)",
             "GHGEmissionsIntensity", "SiteEnergyUse(kBtu)"], axis=1)
```

- Résultats de la métrique RMSE sur les encodeurs.



- Le One Hot Encoding sera retenu pour l'encodage des variables à valeurs uniques faibles(soient 199 variables).

Émission de CO_2 : sélection d'attributs

- Nous utilisons l'estimateur linéaire éparse **Lasso** basé sur la norme L1 approprié pour la de sélection d'attributs.
- Les features sélectionnés sont les attributs de coefficients non nuls.

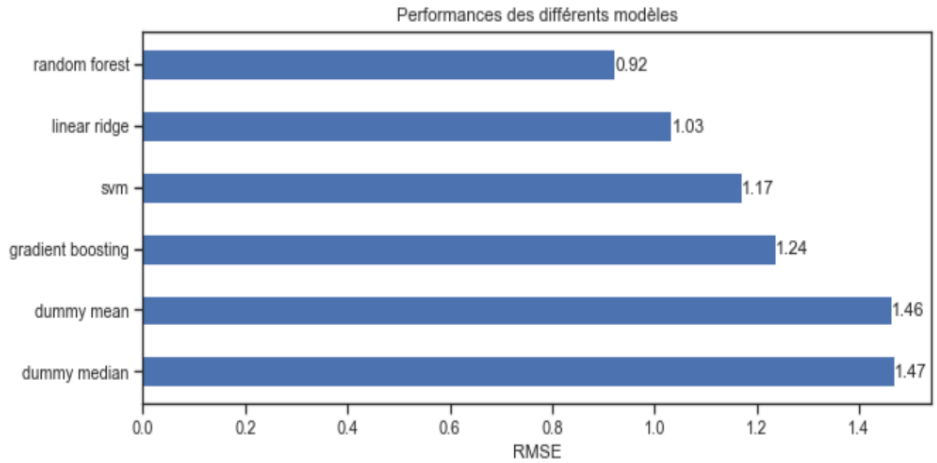
```
#Régularisation L1
X = bunch_train.copy()
y = y_train.copy()
regL = Lasso(alpha=0.015).fit(X, y)
model = SelectFromModel(regL, prefit=True)
X_new = model.transform(X)
print(X_new.shape)

selected_features = pd.DataFrame(model.inverse_transform(X_new), index=X.index,columns=X.columns)
# Dropped columns have values of all 0s, keep other columns
selected_columns = selected_features.columns[selected_features.var() != 0]
print(list(selected_columns))
```

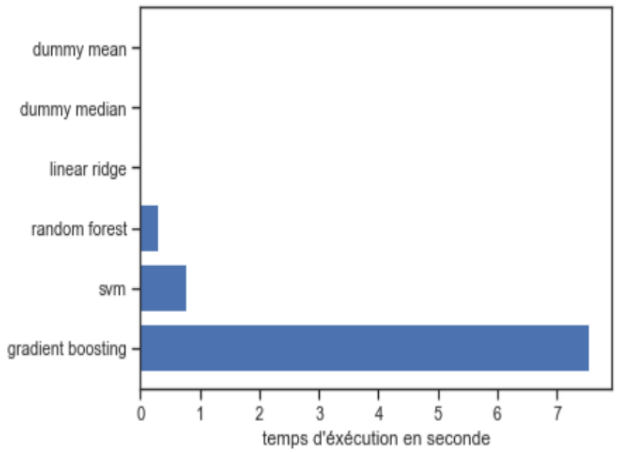
```
(2654, 25)
['CouncilDistrictCode', 'YearBuilt', 'NumberofBuildings', 'NumberofFloors', 'ENERGYSTARScore', 'SteamUse(kBtu)', 'NaturalGas(kB
tu)', 'PropertyName', 'TaxParcelIdentificationNumber', 'ListOfAllPropertyUseTypes', 'LargestPropertyUseType', 'Address', 'ZipCo
de', 'OH1', 'OH5', 'OH7', 'OH10', 'OH14', 'OH16', 'OH22', 'OH24', 'OH25', 'OH27', 'OH28', 'OH34']
```

Émission de CO_2 : choix du modèle

- Résultat de la validation croisée pour la recherche du meilleur modèle



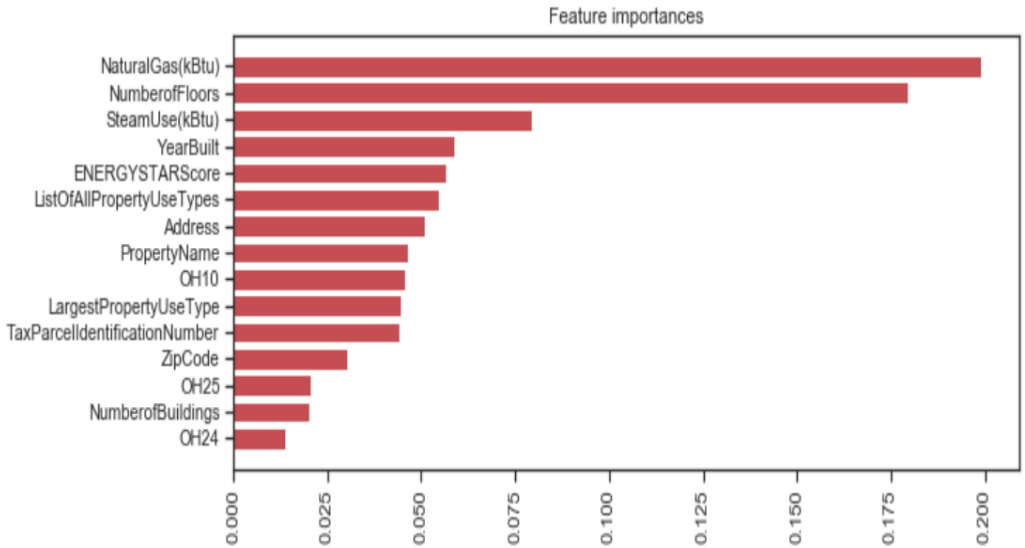
- Temps d'exécution d'apprentissage et de prédiction des meilleurs modèles.



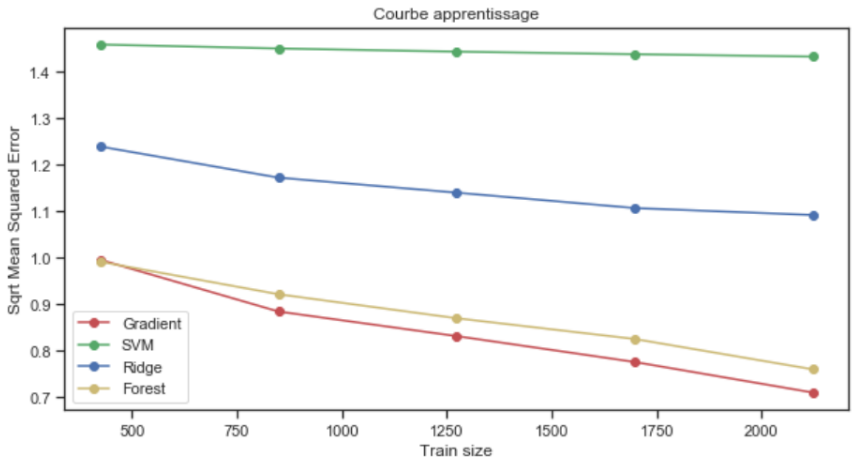
Importance des features et courbe d'apprentissage

- Importance des features pour le meilleur modèle(Forest)

```
<BarContainer object of 15 artists>
```



- Courbe d'apprentissage des différents meilleurs modèles.

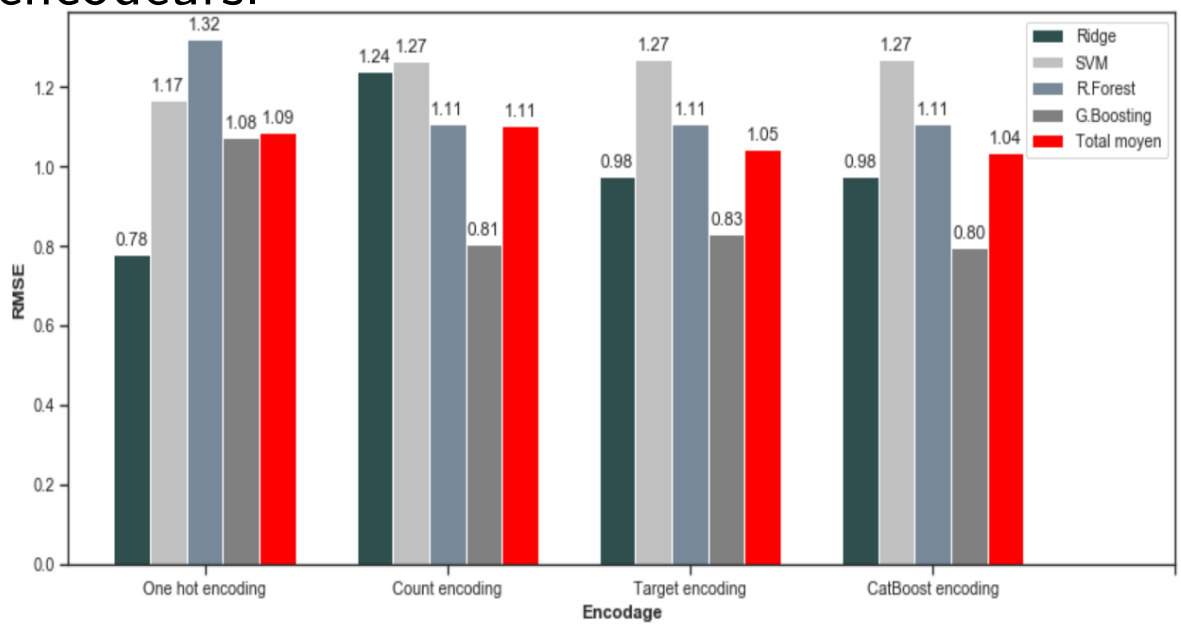


Consommation d'énergie : choix de l'encodeur

- Variable expliqué et variables explicatives

```
y = df["SiteEnergyUse(kBtu)"]
X = df.drop(["SiteEnergyUse(kBtu)", "OSEBuildingID", "SiteEUI(kBtu/sf)", "SourceEUI(kBtu/sf)",
             "GHGEmissionsIntensity", "TotalGHGEmissions"], axis=1)
```

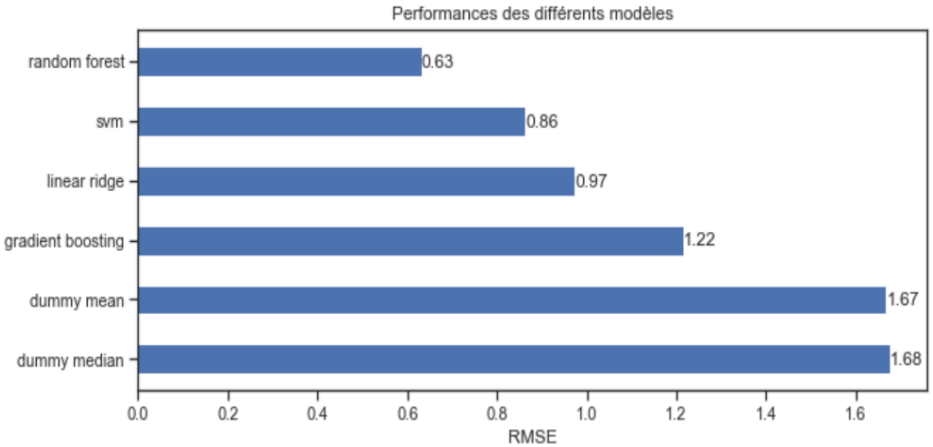
- Résultats de la métrique RMSE sur les différents encodeurs.



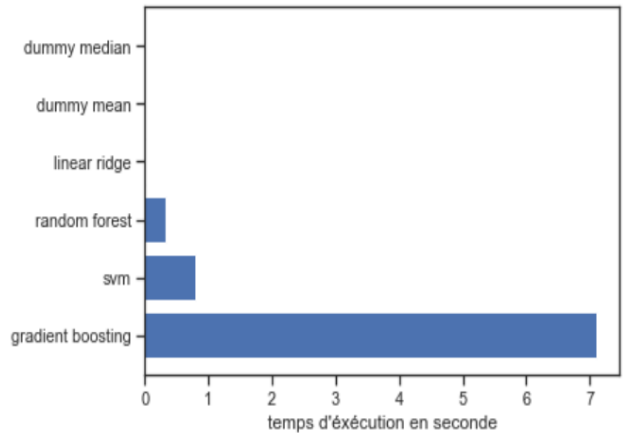
- Le CatBoost Encoding sera retenu pour l'encodage des variables à faibles valeurs uniques(soient 28 variables).

Consommation d'énergie : choix du modèle

- Résultat de la validation croisée pour la recherche du meilleur modèle



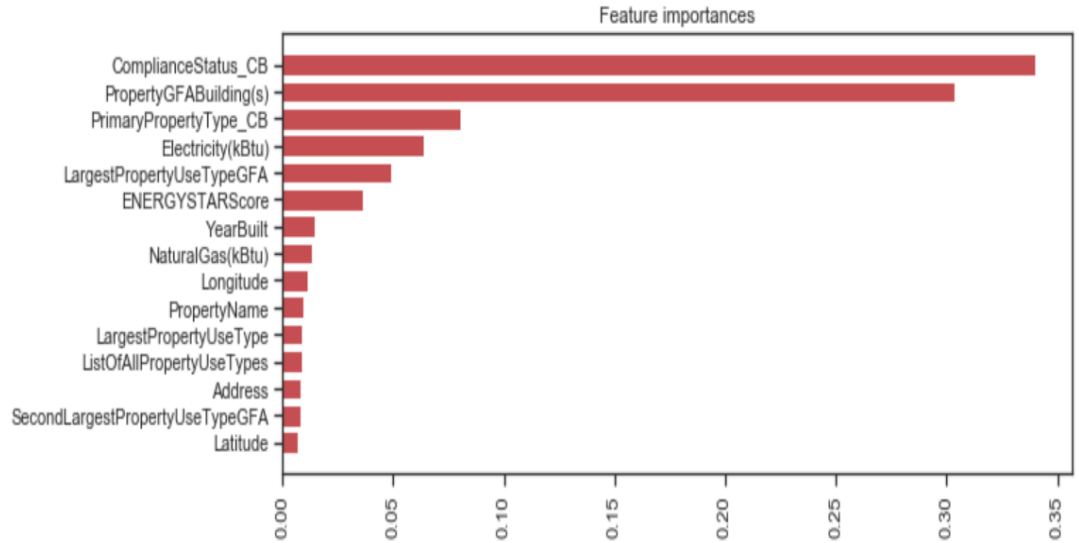
- Temps d'exécution d'apprentissage et de prédiction des meilleurs modèles.



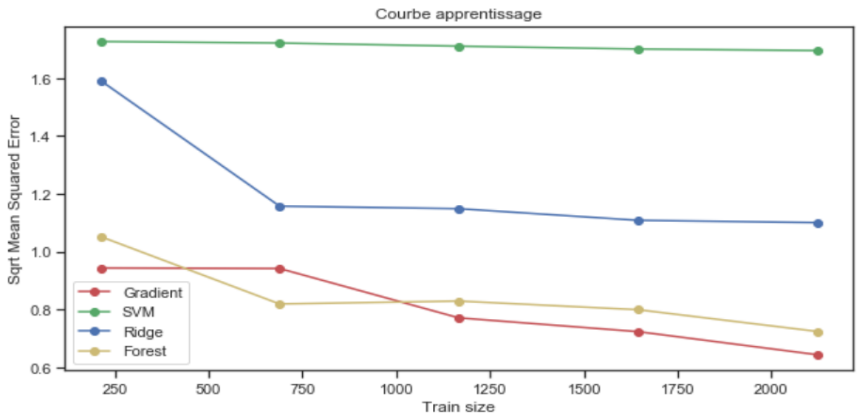
Importance des features et courbe d'apprentissage

- Importance des features pour le meilleur modèle(Forest)

```
<BarContainer object of 15 artists>
```



- Courbe d'apprentissage des différents meilleurs modèles.



Sommaire

Observations et discussions

A la fin de notre travail qui consistait à l'exploration de modèles de machines learning, elle dépend :

- Les fuites de données qu'il faut apprendre à détecter pour éviter le sur-apprentissage de modèles.
- La transformation(normalisation, log(1+x), binarisation) des variables pour l'amélioration des performances de modèles.
- Le choix de l'encodeur pour les variables catégorielles.
- La sélection des attributs de modèles à performer.
- Les performances des modèles de bases.
- Les métriques d'erreurs, le temps d'exécution, de la taille du jeu de données.

Perspectives

- Appliquer l'encodage **Entity Embeddings** pour les variables catégorielles.
- Utiliser le modèle **XGBoost** qui est une version plus avancée de la méthode de Gradient Boosting.
- Utiliser les méthodes d'élimination d'attributs par récursion(**RFE**) pour la sélection d'attributs.
- Exécuter les modèles sur des serveurs distribués avec plus de **puissances de calculs**.