Projet Seattle

Efkan TUREDI

Intro

Ce projet constitue notre premier projet d'apprentissage des méthodes de Machine Learning. Nos objectifs sont:

- Prédire les consommations d'énergie et les émissions en CO2 à partir de données non-mesuré sur compteur
- Donner un avis sur la pertinence de l'Energy Star Score sur la prédiction de l' émission en CO2 (i.e. faire tourner les modèles avec et sans Energy Star Score)

Nous divisons le travail en deux parties: la 1ere sur le nettoyage et la 2eme sur les modèles

Le nettoyage

Nous avons des problèmes courants...

- Il **manque des données** car nous avons beaucoup de NaNs
- Les **entrées ne sont pas uniformes**: par exemple ‘Delridge’, ‘DELRIDGE’, ‘DELRIDGE Neighborhoods’
- Il y a des **valeurs abérrantes**: par exemples des surfaces (GFA) négatives
- Les BDD 2015 et 2016 n’ont **pas les mêmes colonnes, ni les même labels**
- Il y a des **outliers**!

….que nous avons résolus!

- Les colonnes qui nous seront utiles sont bien remplis. Les NaNs touchent notamment les colonnes qui n’apportent pas de valeurs.
- Les valeurs abérrantes sont remplacées par des valeurs min ou max
- Nous avons uniformisé les entrées (cf. Delridge dans la slide d’avant)
- Nous avons fusionné les bdd en gardant les bases de colonnes communes tout en corrigeant certains labels
- Nous avons eliminé les observations ‘High Outlier’

Quelques lignes de codes pour illustration (1/2)

```
outlier_labels = {
    np.nan:'Not abnormal',
    'High outlier':'High Outlier',
    'Low outlier':'Low Outlier'
}
```

```
data['Outlier'].replace(outlier_labels, inplace=True)
```

Correction de labels de certains features

```
data_2015['TotalGHGEmissions'] = data_2015['GHGEmissions(MetricTonsCO2e)']
data_2015['GHGEmissionsIntensity'] = data_2015['GHGEmissionsIntensity(kgCO2e/ft2)']
data_2015['Comments'] = data_2015['Comment']
data_2015['ZipCode'] = data_2015['Zip Codes']
data_2015['DefaultData'] = data_2015['DefaultData'].map({'Yes' : True, 'No' : False}).head()

data_2015.drop(['GHGEmissions(MetricTonsCO2e)', 'GHGEmissionsIntensity(kgCO2e/ft2)','Comment','Zip Codes'], axis=1, inplace=True)
```

Correction de labels de certains features pour alignement entre les deux bases de données

```
my_list = ['PropertyGFAParking','PropertyGFABuilding(s)','SourceEUI(kBtu/sf)',
'SourceEUIWN(kBtu/sf)','Electricity(kWh)','Electricity(kBtu)','TotalGHGEmissions','GHGEmissionsIntensity']
```

```
for col in my_list:
    data[col] = data[col].apply(lambda x : x if x >= 0 else 0)
```

Correction de certaines valeurs abérrantes

Quelques lignes de codes pour illustration (2/2)

```python
data[['ThirdLargestPropertyUseType', 'SecondLargestPropertyUseType']] = data[['ThirdLargestPropertyUseType', 'SecondLargestPropertyUseType']].f
```

Python

```python
data['LargestPropertyUseType'] = data['LargestPropertyUseType'].fillna('Unknown')
data['LargestPropertyUseTypeGFA'] = data['LargestPropertyUseTypeGFA'].fillna(np.mean(data['LargestPropertyUseTypeGFA']))
data[['ThirdLargestPropertyUseTypeGFA', 'SecondLargestPropertyUseTypeGFA']] = data[['ThirdLargestPropertyUseTypeGFA', 'SecondLargestPropertyUse
```

Python

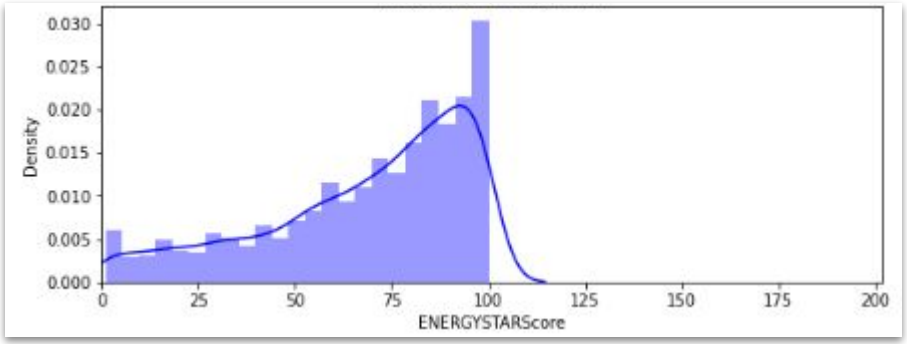
```python
fuel_data = ['SiteEnergyUse(kBtu)','SiteEUIWN(kBtu/sf)','SiteEnergyUseWN(kBtu)','SiteEUI(kBtu/sf)','NaturalGas(kBtu)','NaturalGas(therms)','Ste
for item in fuel_data:
    data[item] = data[item].fillna(np.mean(data[item]))
```

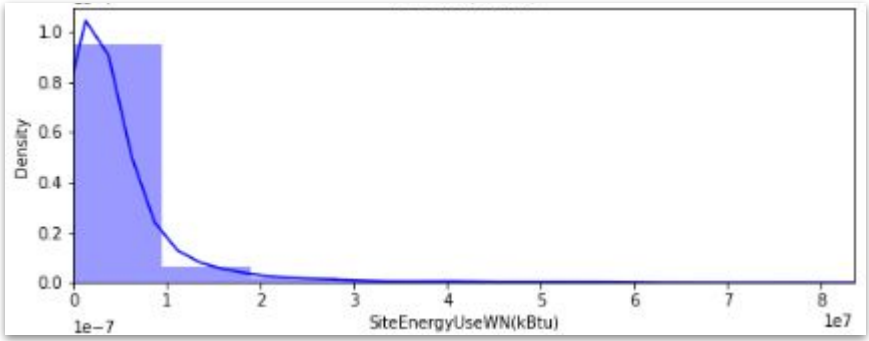
Python

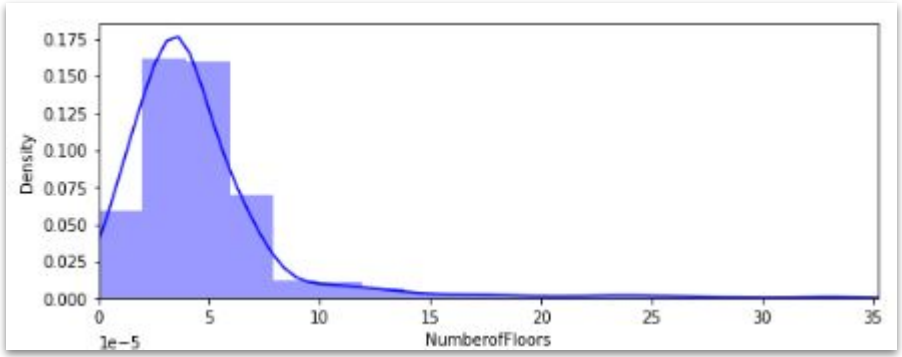
Remplacement de certains NaNs par la moyenne de la feature

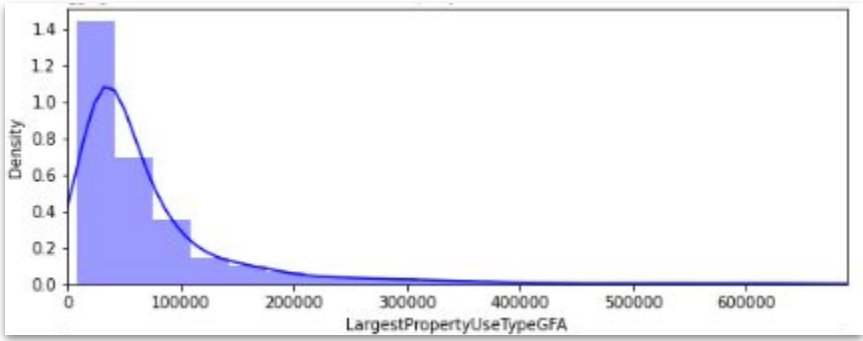
concernée

Les features ont des distributions différentes

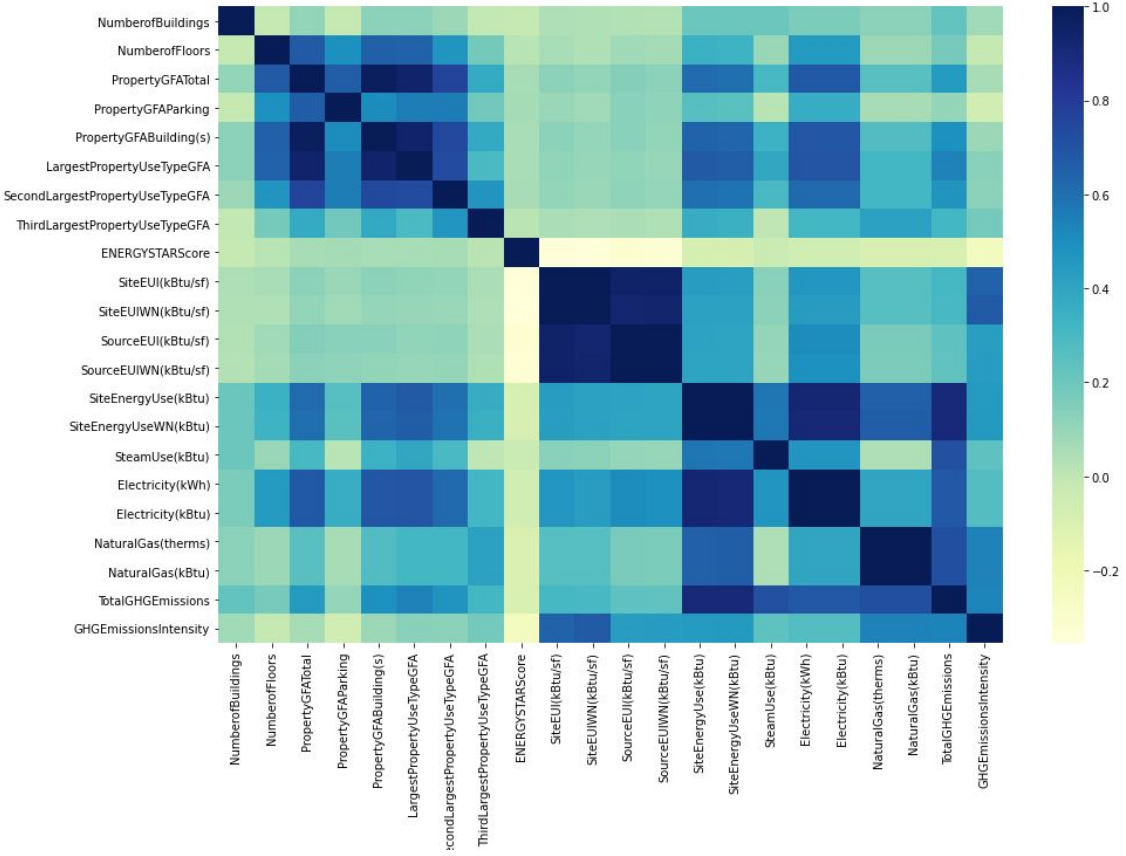








Corrélation entre les features et targets



Séparation et préparation des variables

```
full = data[full_features]

X = full.drop(columns=['TotalGHGEmissions','SiteEnergyUseWN(kBtu)'])
Y = full[['TotalGHGEmissions','SiteEnergyUseWN(kBtu)']]

X = X.reset_index(drop=True)
Y = Y.reset_index(drop=True)
```

- On separe nos features et nos targets. On a ici deux targets correspondant à la consommation électrique et à l'émission en CO2

Standardisation et One Hot Encoding des features

```
from sklearn.preprocessing import OneHotEncoder,StandardScaler

ss = StandardScaler()
X[numerical_columns] = ss.fit_transform(X[numerical_columns])
```

```
ohe = OneHotEncoder(sparse=False)
ohe.fit_transform(X[categorical_columns]);
```

```
test_df = pd.DataFrame(columns = ohe.get_feature_names(),data = ohe.fit_transform(X[categorical_columns]))
```

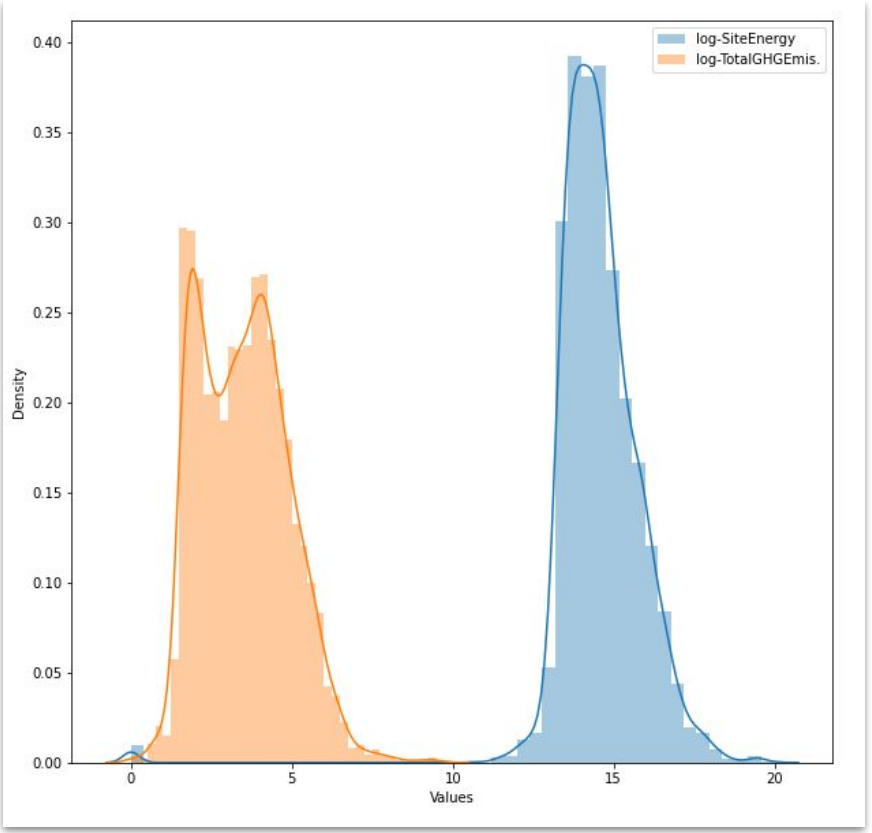
```
test_df.reset_index(drop=True)
```

Passage au Log pour les targets

```
Y['log-SiteEnergyUseWN(kBtu)'] = np.log(1+Y['SiteEnergyUseWN(kBtu)'])
Y['log-TotalGHGEmissions'] = np.log(1+Y['TotalGHGEmissions'])
```

```
Y.describe()
```

	TotalGHGEmissions	SiteEnergyUseWN(kBtu)	log-SiteEnergyUseWN(kBtu)	log-TotalGHGEmissions
count	5051.000000	5.051000e+03	5051.000000	5051.000000
mean	105.872871	5.019856e+06	14.564869	3.494640
std	491.799069	1.568653e+07	1.409341	1.388053
min	0.000000	0.000000e+00	0.000000	0.000000
25%	8.755000	9.910649e+05	13.806536	2.277780
50%	30.860000	1.893665e+06	14.454025	3.461351
75%	85.070000	4.223424e+06	15.256157	4.455161
max	16870.980000	4.716139e+08	19.971671	9.733410



Passage au log très intéressant car permet de réduire l'amplitude de nos variables, et avoir des distribution visuellement proche d'une gaussienne

Les modèles

Quatres modèles vs Dummy Regressor

- Nous allons utiliser les 4 modèles suivants: Elasticnet, Random Forest, SVM et XGBoost
- Nous utiliserons un DummyRegressor qui renvoie la moyenne. Cet estimateur nous servira de baseline pour évaluer la performance de nos modèles.

Nos choix d’indicateurs

- **RMSE**: Root Mean Squared Error
- **RMSE_rel**: Cet indicateur mesure l’amélioration du modèle vs Dummy regressor comme modèle de référence. Plus ce chiffre est élevé, plus le modèle est performant par rapport à notre référence

  RMSE_rel=(RMSE(modèle)-RMSE(Dummy))/RMSE(Dummy)

- **Inference_time**: On mesure ici le temps pris par notre algorithme pour exécuter la fonction .predict(). Peut être important quand il y a beaucoup plus de données

Dummy Regressor

```
from sklearn.dummy import DummyRegressor
import numpy as np

# On crée un modèle qui renvoie constamment la moyenne des données selectionnées
dummy_reg = DummyRegressor(strategy="mean")

# On entraîne ce modèle sur les données d'entrainement
dummy_reg.fit(X_train,Y_train)

results = pd.DataFrame(columns=['Modèle','RMSE','RMSE_rel','Inference_duration (microsecs)'])

start_time = datetime.now()
test = mean_squared_error(dummy_reg.predict(X_test), Y_test)
end_time = datetime.now()
dummy_time = end_time - start_time


ref = mean_squared_error(dummy_reg.predict(X_test), Y_test)
rmse_ref = math.sqrt(ref)

rmse_estimator_dummy = math.sqrt(test)
rmse_rel_dummy = abs((rmse_estimator_dummy - rmse_ref))/rmse_ref
percentage_rmse_rel_dummy = "{:.2%}".format(rmse_rel_dummy)

#just for this case, test == ref
```

Optimisation de paramètres (1/4)

```
from sklearn.linear_model import ElasticNet

#Fonction Coût Elasticnet
#1 / (2 * n_samples) * ||y - Xw||^2_2 + alpha * l1_ratio * ||w||_1 + 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2

#Si alpha = 0, ceci est un OLS classique

parameters = {'tol' : [0.1,0.01,0.001,0.0001], #
              "alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],  #alpha est le coefficient qui multiplie le terme
              "l1_ratio": np.arange(0.0, 1.0, 0.1)} #L1 ratio ,0: Ridge; 1:Lasso


elastic_net = GridSearchCV(estimator = ElasticNet(),
                           param_grid = parameters,
                           cv=5,
                           verbose=False
                          )

elastic_net.fit(X_train, Y_train);
```

```
elastic_net.best_params_
```

```
{'alpha': 0.001, 'l1_ratio': 0.0, 'tol': 0.1}
```

Optimisation de paramètres (2/4)

```
from sklearn.ensemble import RandomForestRegressor

parameters = {
    'n_estimators' : [10,50,100,300,500],
    'min_samples_leaf' : [1,3,5,10],
    'max_features': ['auto', 'sqrt']
}
```

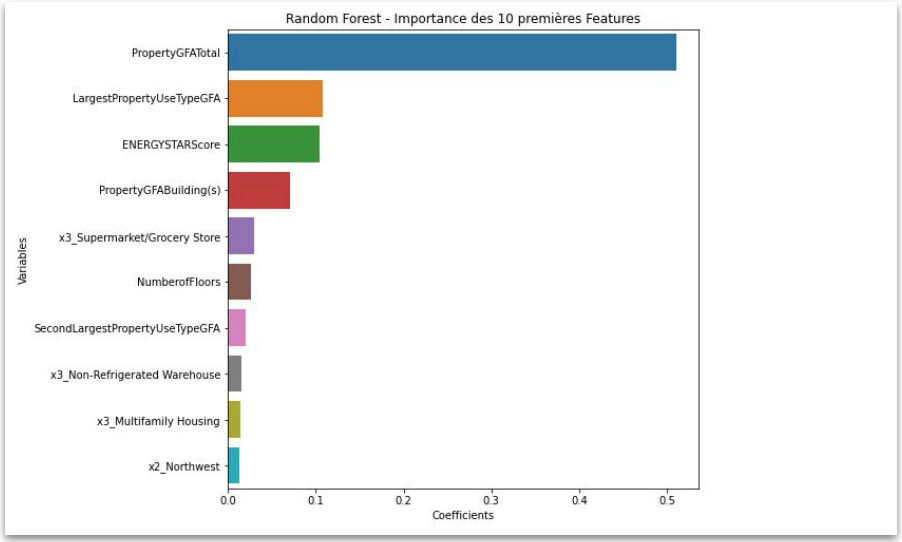
```
rfr = GridSearchCV(RandomForestRegressor(),
                   param_grid = parameters,
                   verbose=False,
                   cv=5)

rfr.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 3, 5, 10],
                         'n_estimators': [10, 50, 100, 300, 500]},
             verbose=False)
```

```
rfr.best_params_
```

```
{'max_features': 'auto', 'min_samples_leaf': 5, 'n_estimators': 300}
```



Optimisation de paramètres (3/4)

```
from sklearn.svm import SVR

parameters = {'gamma' : [1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1], #Kernel par defaut RBF
              'epsilon' : [0.001, 0.01, 0.1, 1], #tolérance par l'algorithme
              'C' : [0.001, 0.01, 0.1, 1, 10]} #Régularisation

svm = GridSearchCV(estimator = SVR(),
                   param_grid = parameters,
                   cv=5,
                   verbose=False
                   )

svm.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=SVR(),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10],
                         'epsilon': [0.001, 0.01, 0.1, 1],
                         'gamma': [1e-08, 1e-07, 1e-06, 1e-05, 0.0001, 0.001,
                                   0.01, 0.1]},
             verbose=False)
```

```
svm.best_params_
```

Press ^Enter to execute cell

```
{'C': 1, 'epsilon': 0.1, 'gamma': 0.1}
```

Optimisation de paramètres (4/4)

```
from xgboost import XGBRegressor
```

✓ 0.4s

```
parameters = {
    'n_estimators' : [10,20,50,100,500,1000,2000]
}
xgb = GridSearchCV(XGBRegressor(),
                   param_grid = parameters,
                   cv = 5,
                   verbose=False)
xgb.fit(X_train, Y_train)
```

✓ 73.9s

```
GridSearchCV(cv=5,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None, gamma=None,
                                    gpu_id=None, importance_type='gain',
                                    interaction_constraints=None,
                                    learning_rate=None, max_delta_step=None,
                                    max_depth=None, min_child_weight=None,
                                    missing=nan, monotone_constraints=None,
                                    n_estimators=100, n_jobs=None,
                                    num_parallel_tree=None, random_state=None,
                                    reg_alpha=None, reg_lambda=None,
                                    scale_pos_weight=None, subsample=None,
                                    tree_method=None, validate_parameters=None,
                                    verbosity=None),
             param_grid={'n_estimators': [10, 20, 50, 100, 500, 1000, 2000]},
             verbose=False)
```

```
xgb.best_params_
```

✓ 0.4s

```
{'n_estimators': 20}
```

Matrice de décision

	Target 1: Site Energy Use WN	Target 2: TotalGHGEmissions
Avec EnergyStar Score		

	Modèle	RMSE	RMSE_rel	Inference_duration (microsecs)
0	Dummy Regressor	1.246218	0.00%	404
1	Elasticnet	0.890220	28.57%	2480
2	Random Forest	0.838259	32.74%	79844
3	SVM	0.791441	36.49%	546030
4	XGBoost	0.910465	26.94%	4872

Winner: SVM/Random Forest

	Modèle	RMSE	RMSE_rel	Inference_duration (microsecs)
0	Dummy Regressor	1.376945	0.00%	691
1	Elasticnet	0.923781	32.91%	2095
2	Random Forest	0.637612	53.69%	133233
3	SVM	0.805790	41.48%	632222
4	XGBoost	0.634891	53.89%	16111

Winner: XGBoost

Sans EnergyStar Score

	Modèle	RMSE	RMSE_rel	Inference_duration (microsecs)
0	Dummy Regressor	1.427288	0.00%	403
1	Elasticnet	1.138766	20.21%	2672
2	Random Forest	1.030050	27.83%	81091
3	SVM	0.998682	30.03%	580638
4	XGBoost	1.152892	19.22%	5655

Winner: SVM/Random Forest

	Modèle	RMSE	RMSE_rel	Inference_duration (microsecs)
0	Dummy Regressor	1.411188	0.00%	459
1	Elasticnet	0.996132	29.41%	2767
2	Random Forest	0.705055	50.04%	125948
3	SVM	0.859328	39.11%	636878
4	XGBoost	0.690017	51.10%	8334

Winner: XGBoost

Conclusions:

(i) Pour EnergySiteUse, le meilleur modèle est Random Forest

(ii) Pour TotalGHGEmissions, le meilleur modèle est XGBoost

(iii) L'Energy Star Score est bénéfique pour nos modèles car les RMSE sont plus bas. Donc nous la conservons!

Merci de votre attention!

Seattle