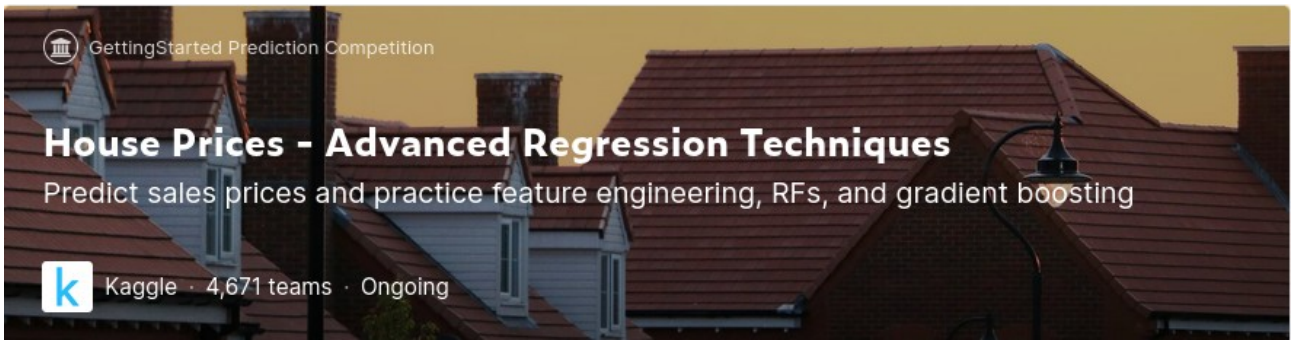


Concours Kaggle

Comment améliorer son score
dans la prédiction du prix immobiliers ?
Ames, Iowa Housing Dataset

20211123_Catherine_LE

Sommaire

Dataset

Ames, Iowa : alternative aux données de logement de Boston.

Les données sont réparties comme telles:

`df_train (1460, 81)`

`df_test (1459, 80)`

`df_submission (1459, 2)`

`df_init (2919, 80)`

L'objectif est de prédire la variable `SalePrice` grâce aux autres 80 variables. Pour se faire, des transformations de variables sont nécessaires à la fois sur le test set pour la prédiction et sur le train set pour entraîner et évaluer le modèle. Le test et train set sont combinés pour donner un `df_init` de dimension `(2919, 80)`.

Les données représentent la surface, la qualité de l'habitation, les années / mois de construction ou de vente.

Préprocessing

Erreur de formatage

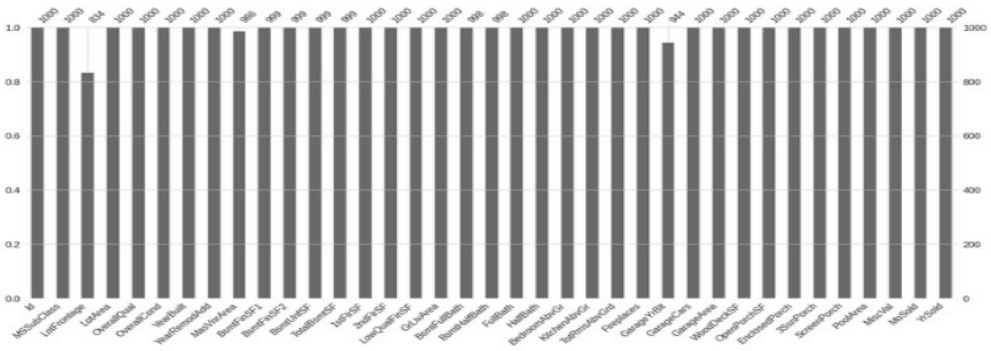
Parmi les variables catégoriques, 16 d'entre elles ont une modalité où le NaN porte une signification. Pour ces variables, les valeurs manquantes seront imputées par 'None'.

Exemple:

```
MiscFeature: Miscellaneous feature not covered in other categories

       Elev     Elevator
       Gar2     2nd Garage (if not described in garage section)
       Othr     Other
       Shed     Shed (over 100 SF)
       TenC     Tennis Court
       NA  None
```

Valeurs manquantes

Les «vraies» valeurs manquantes sont imputées en procédant par étapes. Première étape, les variables de type numérique `int64 et float` puis les variables de type catégorique `string`.



Valeurs manquantes
des variables numériques

Les variables numériques sont imputées par la méthode du KNN imputer. Comme il y peu de valeurs manquantes, cette imputation n’influe pas sur la distribution des données.

Les variables catégoriques sont imputées pas le mode. De même, il existe peu de valeurs manquantes pour ces variables.

MSZoning	4
Utilities	2
Exterior1st	1
Exterior2nd	1
Electrical	1
KitchenQual	1
Functional	2
SaleType	1

Valeurs manquantes
des variables catégoriques

Conversion du types des variables

Cette étape est primordiale, elle permet par la suite de récupérer des variables catégoriques dans l'analyse de corrélation avec la variable `SalePrice`.
Une fois les variable de type `int64` converties en variables catégoriques, un repérage se fait afin de sélectionner parmi elles les variables catégoriques ordinales pour les transformer en variables numériques.

Exemple:

```
PoolQC: Pool quality

       Ex  Excellent
       Gd  Good
       TA  Average/Typical
       Fa  Fair
       NA  No Pool
```

Le site fournit une grille d'évaluation permettant de remplacer ces valeurs par une note globale de 1 à 10:

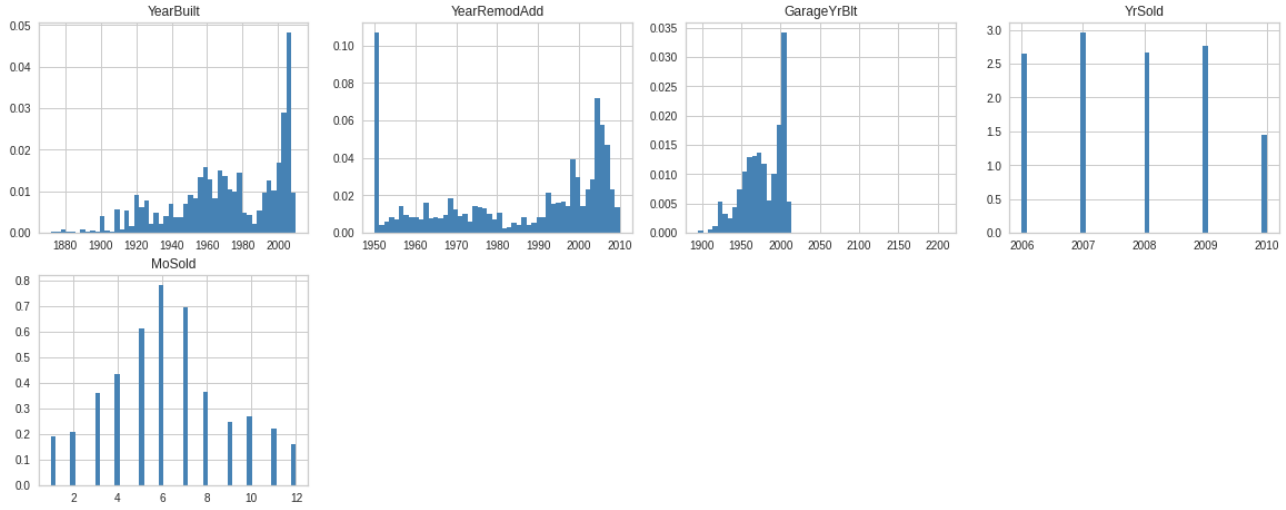
```
OverallQual: Rates the overall material and finish of the house

       10  Very Excellent
       9   Excellent
       8   Very Good
       7   Good
       6   Above Average
       5   Average
       4   Below Average
       3   Fair
       2   Poor
       1   Very Poor
```

Exploration des données

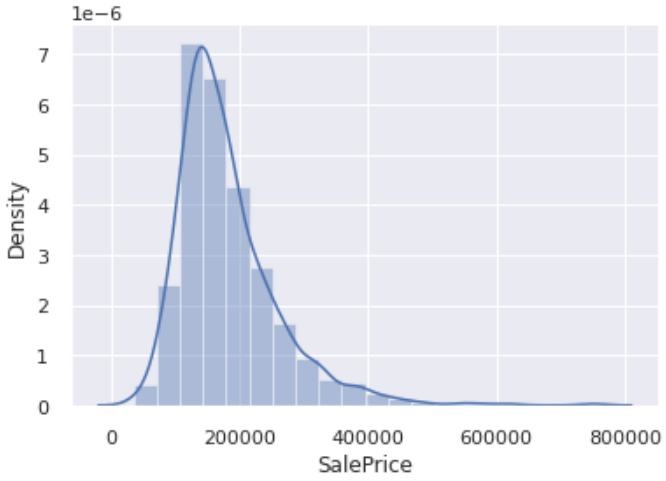
Analyse uni variée

Une vue d'ensemble pour les variables `int64` classées par type d'information : quantité, qualité, et année ou mois.

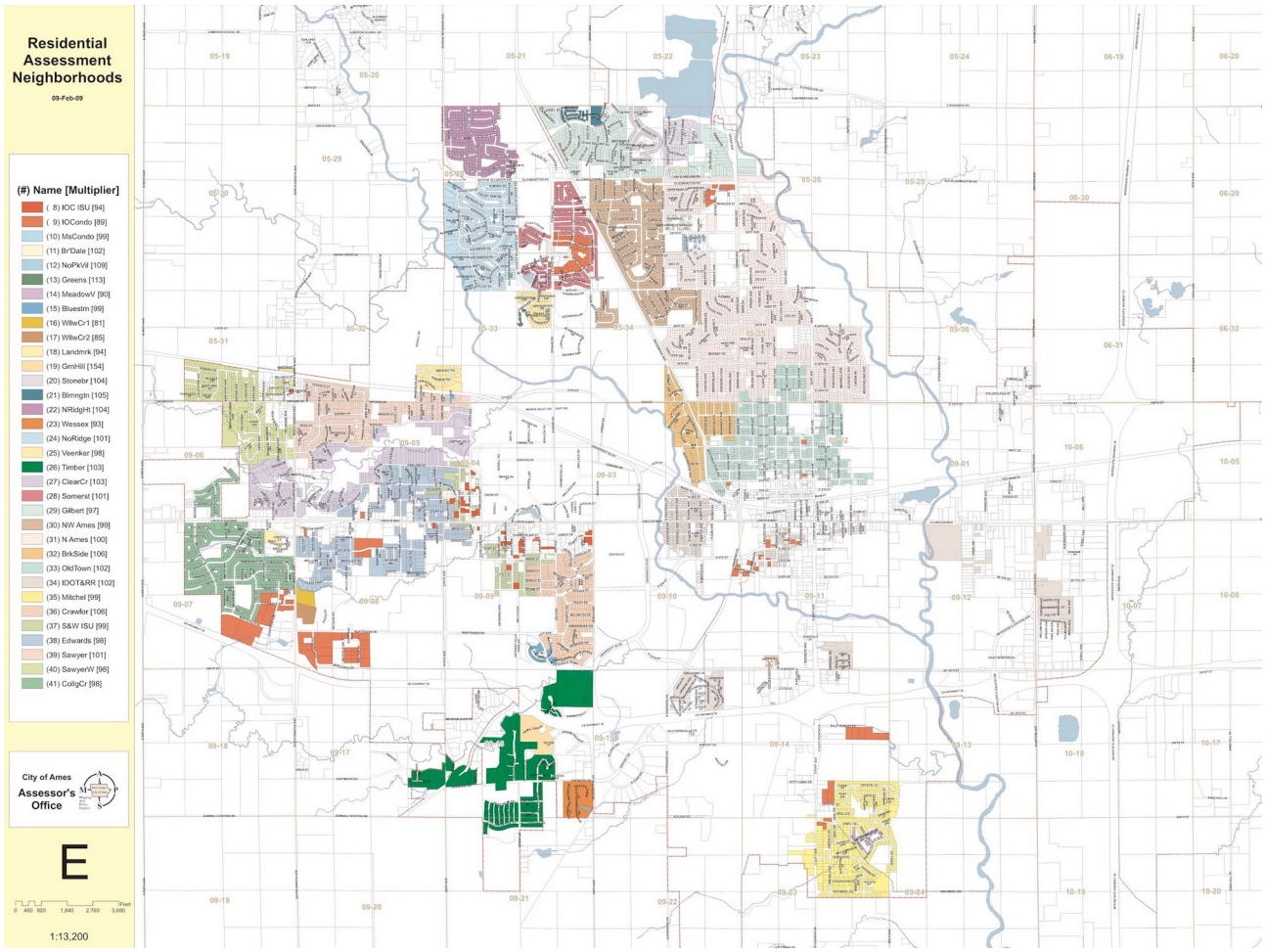
Exemple :

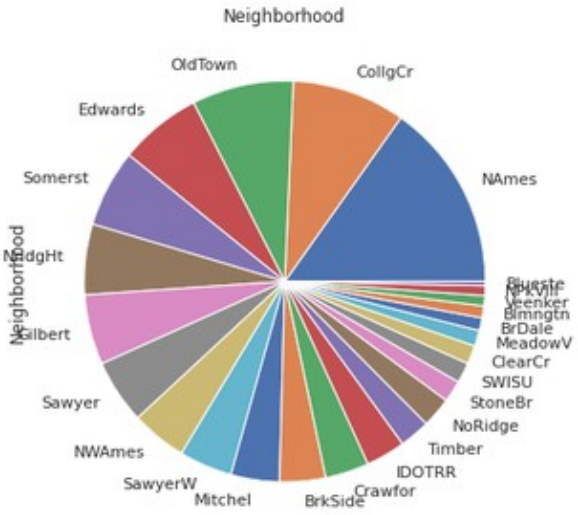


Les variables de type `float` sont des valeurs mesurant la surface en pieds carrés. On peut remarquer des queues de distributions vers la droite, notamment pour la variable SalePrice. D'où une transformation par le log() par la suite.



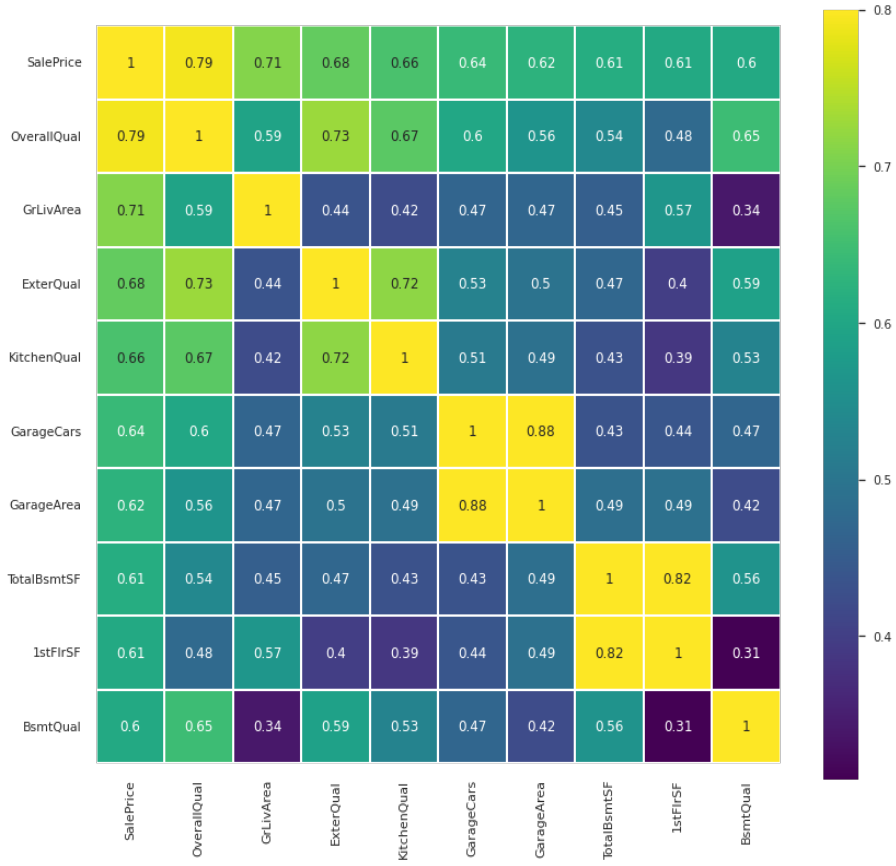
Les variables catégoriques sont difficilement interprétables comme la variables `MSSubClass` avec des codes propres au métiers mais on peut déjà visualiser celles qui le sont plus, par exemple:





Analyse bi-variée

Les analyses de variables en bi-variée vont permettre de repérer les outliers et les variables redondantes.



Les 10 variables les plus corrélées à SalePrice sur le train set

Les variables colinéaires à plus de 80% sont:

	level_0	level_1	corr_coeff
10	GarageQual	GarageCond	0.93
8	GarageCars	GarageArea	0.89
6	Fireplaces	FireplaceQu	0.85
4	GarageYrBlt	YearBuilt	0.81
2	TotRmsAbvGrd	GrLivArea	0.81
0	1stFlrSF	TotalBsmtSF	0.80

Parmi ces paires de variables, on retient celle étant la plus corrélée à `SalePrice`:

SalePrice	1.00
OverallQual	0.79
GrLivArea	0.71
ExterQual	0.68
KitchenQual	0.66
GarageCars	0.64
GarageArea	0.62
TotalBsmtSF	0.61
1stFlrSF	0.61
BsmtQual	0.60
FullBath	0.56
TotRmsAbvGrd	0.53
YearBuilt	0.52
FireplaceQu	0.52
YearRemodAdd	0.51
GarageYrBlt	0.50
MasVnrArea	0.48
Fireplaces	0.47
HeatingQC	0.43
BsmtFinSF1	0.39
BsmtExposure	0.36
LotFrontage	0.34
WoodDeckSF	0.32
2ndFlrSF	0.32
OpenPorchSF	0.32
HalfBath	0.28
GarageQual	0.28
GarageCond	0.26
LotArea	0.26
BsmtFullBath	0.23
BsmtCond	0.22
BsmtUnfSF	0.21
BedroomAbvGr	0.17
PoolQC	0.11
ScreenPorch	0.11
PoolArea	0.09
MoSold	0.05
3SsnPorch	0.04
ExterCond	0.02
BsmtFinSF2	-0.01
BsmtHalfBath	-0.02
MiscVal	-0.02
Id	-0.02
LowQualFinSF	-0.03
YrSold	-0.03
OverallCond	-0.08
EnclosedPorch	-0.13
KitchenAbvGr	-0.14

Importance des variables au prix **SalePrice**

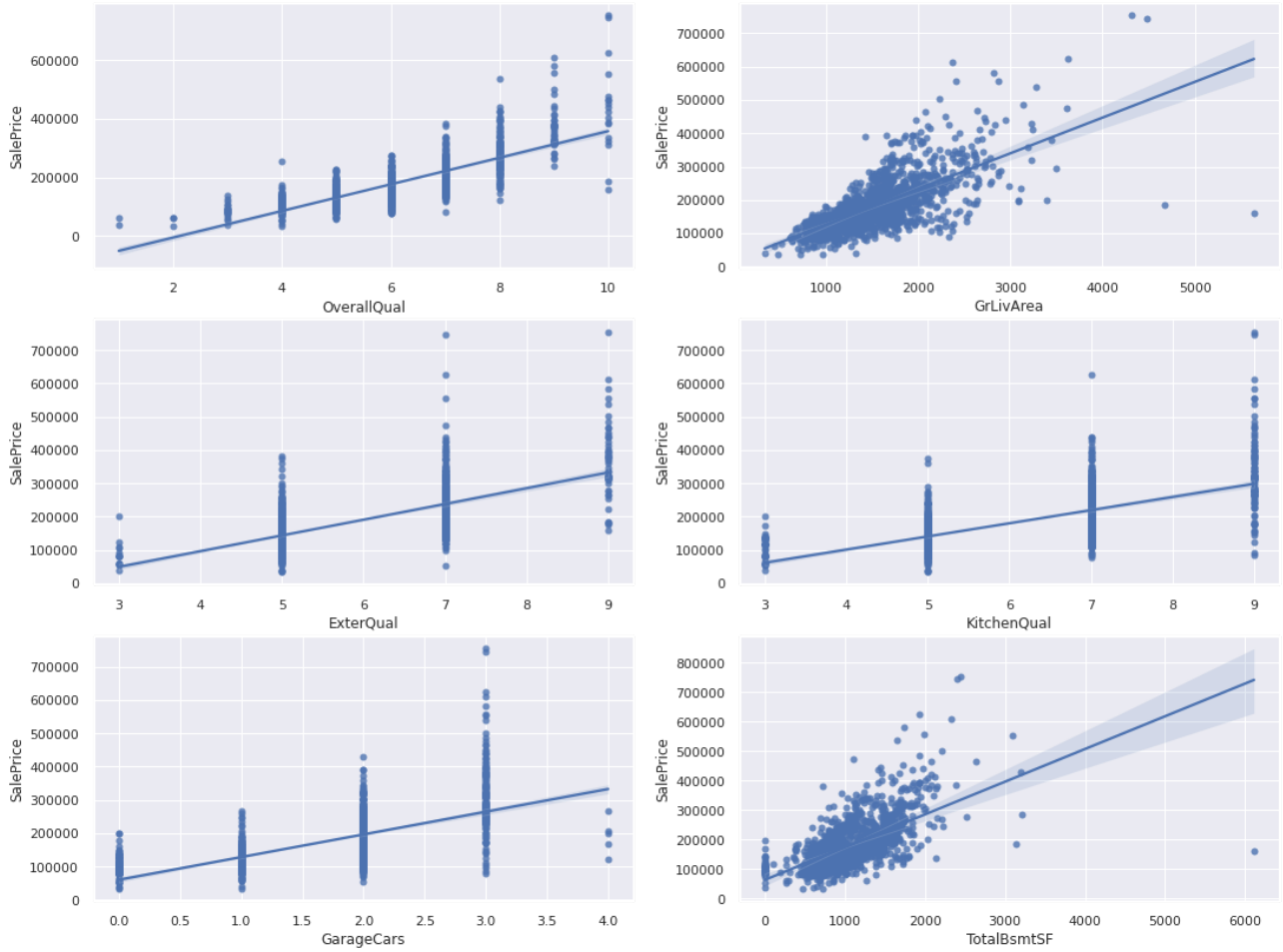
- `OverallQual` : qualité globale
- `GrLivArea` : surface habitable rez de chaussé en pieds carrés
- `ExterQual`: qualité des matériaux à l'extérieur
- `KitchenQual`: qualité de la cuisine
- `GarageCars`: taille du garage en nombre de voitures
- `TotalBsmtSF`: superficie totale du sous-sol en pieds carrés

Suppression de ces variables

Outliers

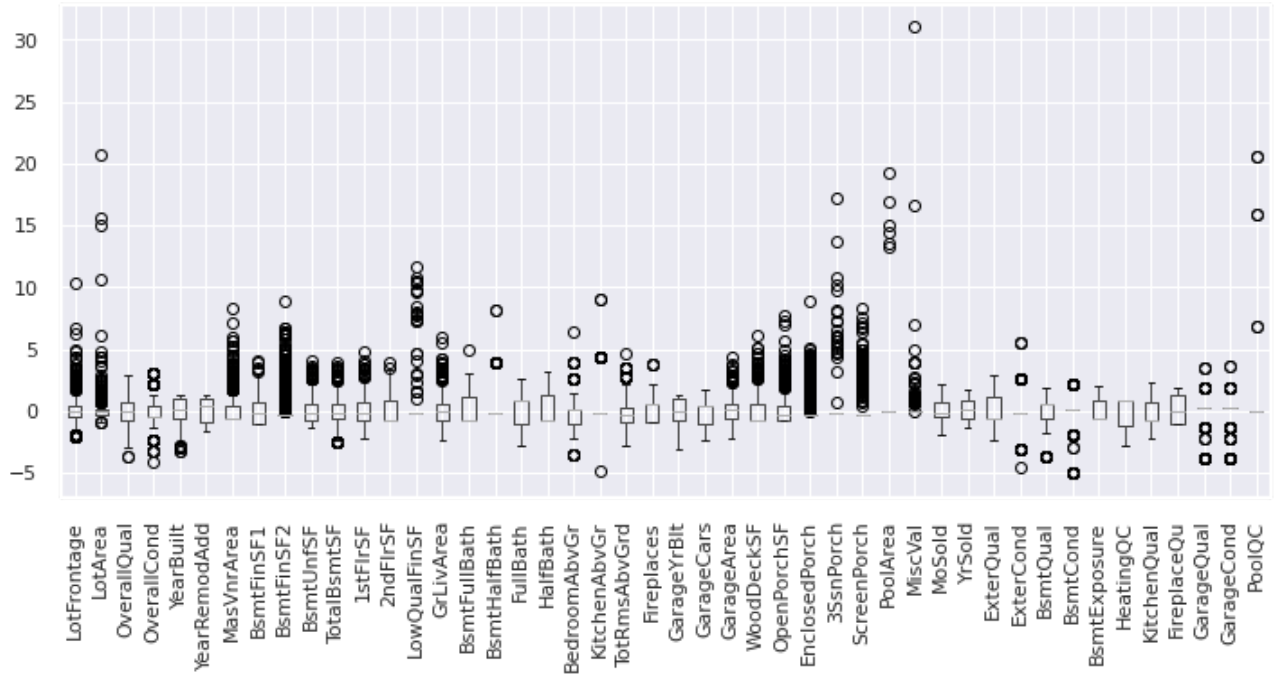
Variables corrélées à `SalePrice`

Procédé en deux étapes. Première étape, en observant les variables numériques les plus corrélées à `SalePrice`: `OverallQual`, `GrLivArea`, `ExterQual`, `KitchenQual`, `GarageCars`, `TotalBsmtSF`. Deuxième étape, en regardant toutes les variables numériques dans son ensemble.



Pour `GrLivArea`, `GarageCars`, `TotalBsmtSF`, on voit les points maximums en abscisse non situés en haut à droite. Pour supprimer ces outliers, on procède pas à pas par la fonction `max()`.

Vue d’ensemble des variables numériques



Et pour les variables vues dans son ensemble, on supprime les individus au delà d’un certain seuil. Ici le seuil est de 25. N’est supprimé qu’un seul individus. Ce seuil n’est pas plus bas car les variables `PoolQC` et `PoolArea` se retrouveraient avec des valeurs manquantes voir sans valeurs du tout.

A noter que les outliers sont supprimés uniquement sur le train set. Le test set reste intacte pour la soumission des résultats.

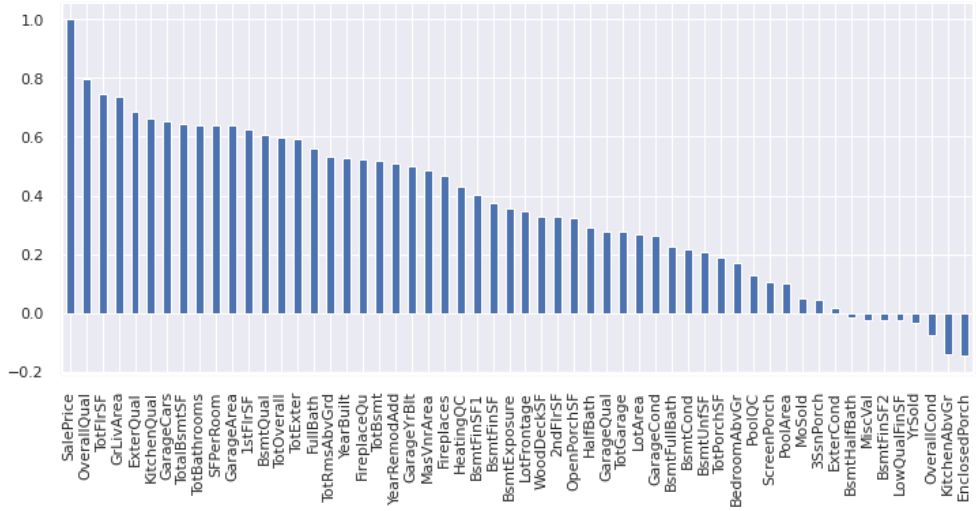
Features Engineering

Création de nouvelles variables

Les nouvelles variables sont crées en par la somme ou la somme pondérée de plusieurs variables. Il existe de nombreuse combinaison, en voici quelques une :

```
df_cleaned4["SFPerRoom"] : superficie par pièce
df_cleaned4['TotOverall'] : qualité générale totale
df_cleaned4['TotGarage'] : qualité générale des garages
```

Suppression des variables



Importance des variables numériques par rapport à `SalePrice`

Comme vu précédemment, ne sont supprimées que les variables négativement corrélées à SalePrice:

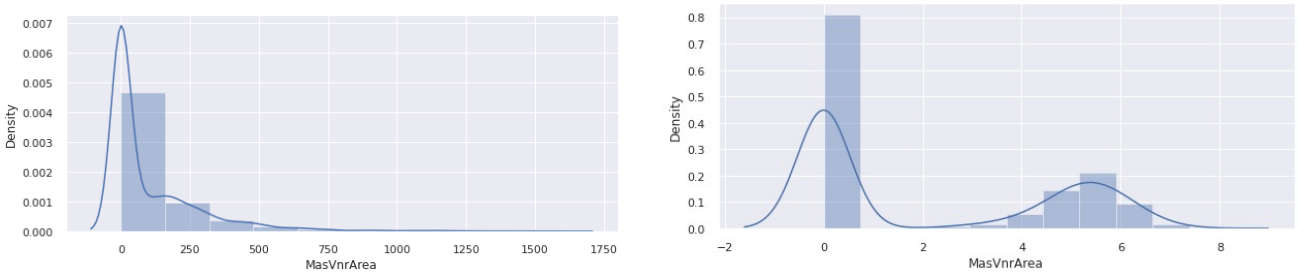
```
BsmtFinSF2
LowQualFinSF
YrSold
MiscVal
OverallCond
EnclosedPorch
KitchenAbvGr
```

et les variables colinéaires:

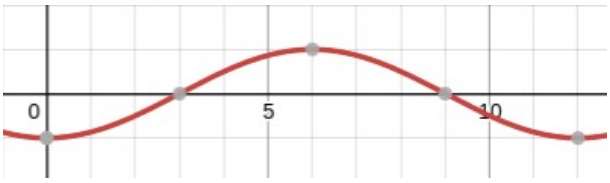
```
GrLivArea
TotGarage
GarageCond
TotBsmt
BsmtFinSF
PoolArea
GarageArea
Fireplaces
TotExter
TotRmsAbvGrd
BsmtCond
GarageYrBlt
1stFlrSF
```

Transformation des variables X

Après avoir supprimé les ouliers sur le train set, on voit que certaines variables sont tout de même tirées vers la droite loin des valeurs centrales. Cela est due en partie due aux outliers non supprimés sur le test set. Pour corriger cela, on applique la fonction `log(x+1)` pour concentrer ces valeurs.



Avant et après la transformation log(x+10)



-cos(0.5236x)

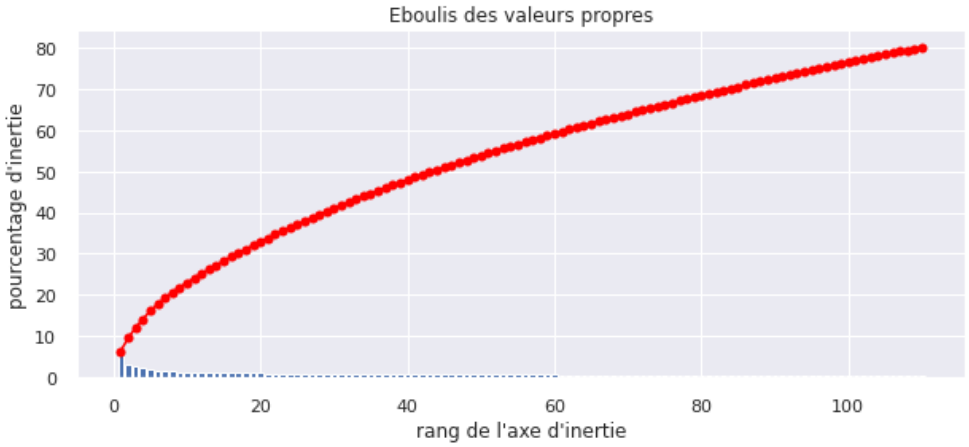
Le dataset contient une variable cyclique représentant les mois de l'année. Pour faciliter l'interprétation de ces valeurs on applique la fonction ci-dessus à la variable `MoSold` correspondant aux mois de vente.

Modélisation

Préparation des données

Récapitulatif de la préparation des données pour la modélisation :

- Encodage des variables catégoriques
- Normalisation de X, des variables dummies inclues
- Création de nouvelles variables pas la PCA – *Principal Component Analysis*
- Log(y)
- Split du dataset



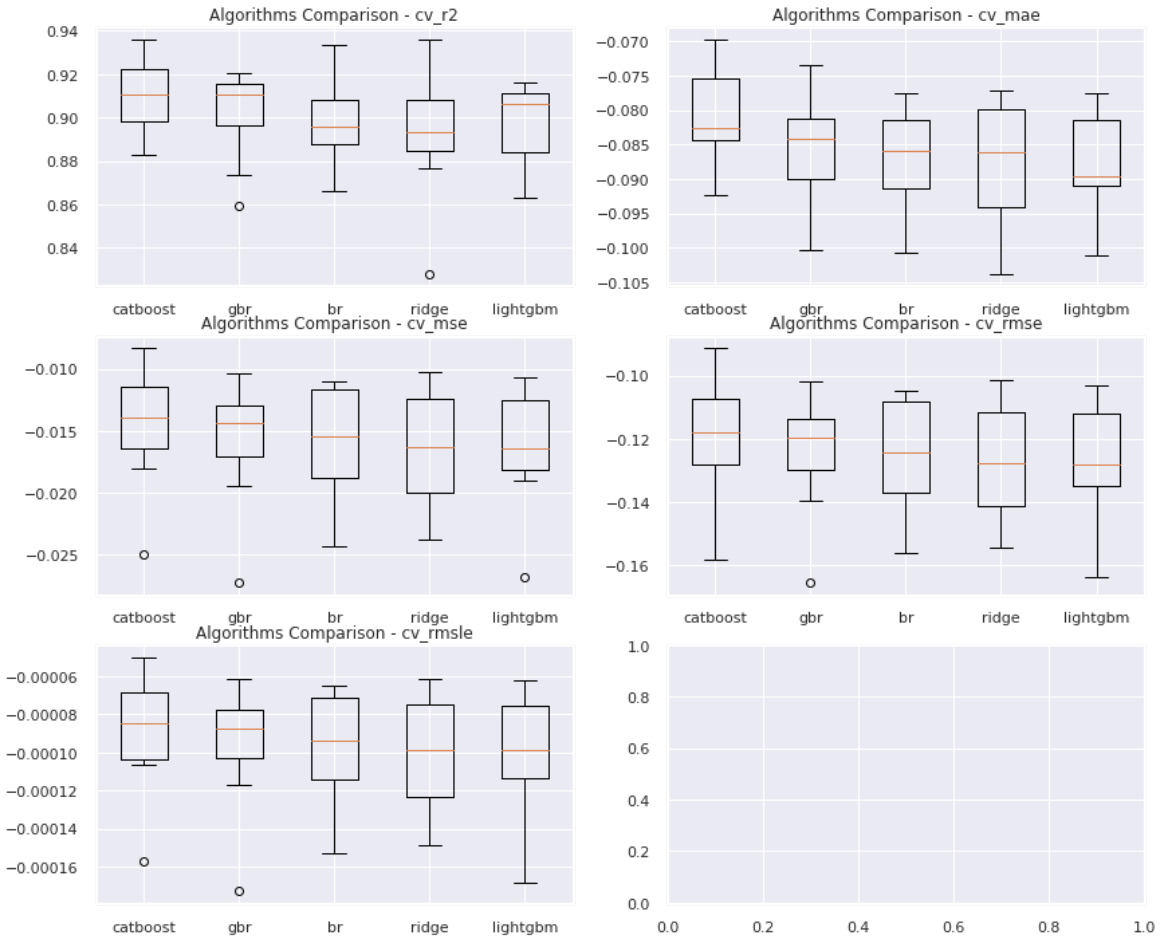
110 nouvelles variables pour 80 % de l'inertie totale

Évaluation des modèles

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
br	Bayesian Ridge	0.0848	0.0149	0.1213	0.8955	0.0095	0.0071	0.0170
catboost	CatBoost Regressor	0.0846	0.0151	0.1221	0.8948	0.0096	0.0071	5.6410
ridge	Ridge Regression	0.0889	0.0168	0.1290	0.8812	0.0101	0.0075	0.0050
gbr	Gradient Boosting Regressor	0.0917	0.0171	0.1305	0.8794	0.0102	0.0077	0.0660
lightgbm	Light Gradient Boosting Machine	0.0934	0.0175	0.1318	0.8776	0.0103	0.0078	0.0430
et	Extra Trees Regressor	0.0952	0.0196	0.1396	0.8631	0.0109	0.0080	0.1660
huber	Huber Regressor	0.0959	0.0197	0.1394	0.8611	0.0109	0.0081	0.0560
rf	Random Forest Regressor	0.0983	0.0201	0.1415	0.8592	0.0110	0.0082	0.1540
omp	Orthogonal Matching Pursuit	0.1011	0.0202	0.1418	0.8570	0.0111	0.0085	0.1280
xgboost	Extreme Gradient Boosting	0.1024	0.0213	0.1455	0.8508	0.0114	0.0086	37.1010
par	Passive Aggressive Regressor	0.1126	0.0234	0.1520	0.8346	0.0119	0.0094	0.0080
knn	K Neighbors Regressor	0.1133	0.0259	0.1604	0.8193	0.0125	0.0095	0.0100
ada	AdaBoost Regressor	0.1267	0.0273	0.1649	0.8077	0.0128	0.0106	0.0450
dt	Decision Tree Regressor	0.1469	0.0420	0.2043	0.6987	0.0159	0.0123	0.0080
lasso	Lasso Regression	0.2975	0.1458	0.3804	-0.0083	0.0292	0.0248	0.1320
en	Elastic Net	0.2975	0.1458	0.3804	-0.0083	0.0292	0.0248	0.0050
llar	Lasso Least Angle Regression	0.2975	0.1458	0.3804	-0.0083	0.0292	0.0248	0.2370
dummy	Dummy Regressor	0.2975	0.1458	0.3804	-0.0083	0.0292	0.0248	0.0040
lr	Linear Regression	0.7512	170.4820	5.8572	-1309.2050	0.1152	0.0612	0.2140

Avec la librairie d'AutoML Pycaret les modèles proposés ci-dessus correspondent au dataset retraité avec ajout et suppression de features **mais sans la PCA**. La seconde tentative a échoué avec PCA, car pycaret n'est pas compatible avec les dernières versions de scikit-learn. Après ré-installation de scikit-learn en version moins récente, la relance n'a pas fonctionné.

Donc j'ai conservé ces modèles par la suite.



Evaluation des performances avec cross validation cv = 10

```
----------
catboost
mean error: 1.1255705021722968
std error : 0.019166088729885393
----------
gbr
mean error: 1.1313293560334636
std error : 0.018247295842752285
----------
br
mean error: 1.1309322682595433
std error : 0.01677582200834476
----------
lightgbm
mean error: 1.1340964763343142
std error : 0.01605738440024673
----------
ridge
mean error: 1.1340701486514333
std error : 0.01579574106419592
```

Moyennes du `exp(rmse)` pour les écarts d'erreur en dollars

Baseline

```
baseline = CatBoostRegressor(verbose=0)
baseline.fit(df_finals[0],log_ys[0]["SalePrice"])

pred_CBR = np.exp(baseline.predict(df_finals[1]))
```

```
# First submission
Id = df_test["Id"]
df_submited = pd.concat([Id, pd.Series(pred_CBR, name='SalePrice')], axis=1
df_submited
```

	Id	SalePrice
0	1461	122280.80
1	1462	156660.81
2	1463	187629.83
3	1464	192667.31
4	1465	178686.50
...	...	...
1454	2915	84055.80
1455	2916	82374.96
1456	2917	154157.58
1457	2918	122527.66
1458	2919	208587.87

Bagging Ensemble

```
# Train models
models = {
    "catboost": CatBoostRegressor(),
    "gbr": GradientBoostingRegressor(),
    "br": BayesianRidge(),
    "lightgbm": LGBMRegressor(),
    "ridge": Ridge()
    #"xgboost": XGBRegressor(),
    #"et": ExtraTreesRegressor(),
    #"knn": KNeighborsRegressor(),
    #"omp": OrthogonalMatchingPursuit()
}
for name, model in models.items():
    model.fit(df_finals[0],log_ys[0]["SalePrice"])
    print(name + " trained.")
```

```
Learning rate set to 0.043414
0:      learn: 0.3867934        total: 5.96ms   remaining: 5.96s
1:      learn: 0.3756039        total: 11.5ms   remaining: 5.75s
2:      learn: 0.3656396        total: 17ms     remaining: 5.66s
3:      learn: 0.3567681        total: 22.8ms   remaining: 5.68s
4:      learn: 0.3478197        total: 28.9ms   remaining: 5.76s
5:      learn: 0.3386746        total: 34.7ms   remaining: 5.75s
6:      learn: 0.3299626        total: 39.9ms   remaining: 5.66s
7:      learn: 0.3217978        total: 45ms     remaining: 5.58s
8:      learn: 0.3143819        total: 50.3ms   remaining: 5.54s
9:      learn: 0.3059662        total: 59.3ms   remaining: 5.87s
10:     learn: 0.2984548        total: 64.8ms   remaining: 5.83s
11:     learn: 0.2916967        total: 71.2ms   remaining: 5.86s
12:     learn: 0.2841709        total: 77.1ms   remaining: 5.85s
13:     learn: 0.2776043        total: 82.3ms   remaining: 5.79s
14:     learn: 0.2706964        total: 87.4ms   remaining: 5.74s
15:     learn: 0.2642070        total: 92.4ms   remaining: 5.68s
16:     learn: 0.2589142        total: 97.6ms   remaining: 5.64s
17:     learn: 0.2528583        total: 102ms    remaining: 5.59s
```

```
# Predict on test set these models

final_predictions = (
    0.4 * np.exp(models['catboost'].predict(df_finals[1])) +
    0.2 * np.exp(models['gbr'].predict(df_finals[1])) +
    0.2 * np.exp(models['br'].predict(df_finals[1])) +
    0.1 * np.exp(models['ridge'].predict(df_finals[1])) +
    0.1 * np.exp(models['lightgbm'].predict(df_finals[1]))
    #0.1 * np.exp(models['et'].predict(dfs[1])))
    #0.5 * np.exp(models['knn'].predict(dfs[1]))+
    #0.5 * np.exp(models['omp'].predict(dfs[1]))
)
final_predictions
```

```
array([122045.73149514, 158389.51760678, 184292.28137136, ...,
       156052.84270186, 122624.39636893, 208398.34200618])
```

GridSearchCV sur le Bagging

```
# To known the rank models see. box plots im Models Comparison

final_predictions = (
    0.4 * y_pred + #catboost
    0.2 * pred_gbr +
    0.2 * pred_br +
    0.1 * pred_ridge +
    0.1 * pred_lightgbm)

    #0.1 * np.exp(models['et'].predict(dfs[1])))
    #0.5 * np.exp(models['knn'].predict(dfs[1]))+
    #0.5 * np.exp(models['omp'].predict(dfs[1]))

final_predictions
```

```
array([122140.469, 161014.742, 184077.834, ..., 160046.843, 121327.587,
       208635.439])
```

Une fois les modèles entraînés sur leurs meilleurs hyper paramètres, on obtient leurs prédictions. Ces prédictions sont sommées puis pondérées en fonction de leurs mesures de performance obtenues par la cross validation score.

Conclusion

Pour répondre à la problématique: comment améliorer son score dans la prédiction du prix immobilier ?

Steps	Score	Rank %	Rank
Baseline catboost	0.12301	7.11%	442
Bagging Ensemble	0.12386	7.17%	445
Add Features	0.12213	7.04%	439
Drop Features	0.12235	7.06%	440
Baseline : Add Features + Features Elimination + Outliers	0.12562	7.30%	451
Bagging : Add Features + Features Elimination + Outliers	0.12296	7.11%	442
Baseline with PCA	0.1294	7.61%	465
Bagging with PCA	0.12463	7.24%	448
Bagging with PCA and GridSearchCV	0.12364	7.15%	444
Best score:	0.12213		

Ci-dessus les résultats sur un nombre total de participants de 4676, en ne tenant pas compte des 118 premiers candidats pour leurs résultats identiques et un saut brutal du score de 0,11 à 0,08 à la 118ème place.

On voit que malgré les modifications apportés, les résultats sont sensiblement différents. Dans ce contexte, il est préférable de ne pas supprimer les variables.

Les modifications favorables au score sont:

- AutoML avec pycaret

- l'ajout de colonnes avec PCA ou bien par combinaison de colonne(s)

- le Bagging

- le GridSearchCV sur le Bagging

A noter que les résultats avec PCA pourraient être meilleur en relançant pycaret sur le nouveau jeu de données avec plus de colonnes. Chose qui n'a pas pu se faire car pycaret a cessé de fonctionner .

En perspective : tester d'autres modèles avec des techniques plus avancées de Features Selection , parmi elles, Lasso et Ridge. Résoudre pycaret ou bien trouver une autre librairie d'AutoML. Aussi, re-tester `StackingRegressor()`.

Difficultés rencontrées

Efficacité de pycaret pour l'AutoML dans le choix des modèles en fonction de nos données. Cependant, cette librairie n'est pas à jour avec les dernières versions de scikit-learn. Quelques difficultés techniques à relancer les calculs.

Comparaison entre Optuna et GridSaerchCV: Optuna à l'avantage de calculer les hyper paramètres au centième près mais le notebook ne peut stocker un tel poids de calculs, des crachs sont apparus. Il est donc préférable d'utiliser GridSearhCV en choisissant des valeurs finies sur un intervalle donné.

Sources

Features Engineering - AutoML :

https://www.kaggle.com/gcdatkin/top-10-house-price-regression-competition-nb

GridSearCV et Stacking :

https://github.com/chitresh28/Kaggle_house_price_prediction/blob/master/Advance_house_price_regression_kaggle.ipynb