Rapport

(Projet de prévisions des prix des maisons pour Ames, Iowa)

M1 Informatique

Membre: LIN Tianyuan

Enseignant: Rakia Jaziri

I. Contexte du projet

Mon ami veut être le meilleur agent immobilier de Ames, Iowa. Pour l'aider à mieux rivaliser avec ses pairs, j'ai décidé d'appliquer certains concepts de base de l'apprentissage automatique et de lui demander d'aider ses clients à fixer le meilleur prix de vente pour leurs propriétés. Heureusement, j'ai trouvé un ensemble de données sur les prix des logements à Ames, Iowa, qui regroupe des données sur les prix des logements à Ames, Iowa qui contiennent plusieurs dimensions caractéristiques. Ma tâche consistait à réaliser une analyse statistique à l'aide des outils disponibles et à construire un modèle d'optimisation sur la base de cette analyse. Ce modèle sera utilisé pour l'aider à évaluer le meilleur prix de vente pour la propriété de son client.

II. Présentation des données

Nom de l'ensemble de données : Ensembles de données sur les prix des maisons à Ames, Iowa

Type de données : Numérique ou catégorique

Nombre de caractéristiques : 79

Nombre d'instances : 1460

Valeurs manquantes : 6965

Tâches connexes : Analyse descriptive (visualisation), analyse prédictive (modélisation)

No	Propriétés	Type de données	Description du champ
1	MSSubClass	int	La classe de bâtiment
2	MSZoning	object	La classification générale du zonage
3	LotFrontage	float	Pieds linéaires de la rue reliée à la propriété
4	LotArea	int	Taille du lot en pieds carrés
5	Street	object	Type d'accès routier

6	Alley	object	Type d'accès à la ruelle
7	LotShape	object	Forme générale de la propriété
8	LandContour	object	Planéité de la propriété
9	Utilities	object	Type de services publics disponibles
10	LotConfig	object	Configuration du terrain
11	LandSlope	object	Pente de la propriété
12	Neighborhood	object	Lieux physiques dans les limites de la ville d'Ames
13	Condition1	object	Proximité d'une route principale ou d'une voie ferrée
14	Condition2	object	Proximité d'une route principale ou d'une voie ferrée (si une seconde est présente)
15	BldgType	object	Type de logement
16	HouseStyle	object	Style de logement
17	OverallQual	int	Qualité générale des matériaux et des finitions
18	OverallCond	int	Évaluation de l'état général
19	YearBuilt	int	Date de construction originale
20	YearRemodAdd	int	Date de remodelage
21	RoofStyle	object	Type de toit
22	RoofMatl	object	Matériau de la toiture
23	Exterior1st	object	Revêtement extérieur de la maison

24	Exterior2nd	object	Revêtement extérieur de la maison (si plus d'un matériau)
25	MasVnrType	object	Type de placage de maçonnerie
26	MasVnrArea	float	Superficie du placage de maçonnerie en pieds carrés
27	ExterQual	object	Qualité des matériaux extérieurs
28	ExterCond	object	État actuel des matériaux à l'extérieur
29	Foundation	object	Type de fondation
30	BsmtQual	object	Hauteur du sous-sol
31	BsmtCond	object	État général du sous-sol
32	BsmtExposure	object	Murs des sous-sols de type Walkout ou rez-de-jardin
33	BsmtFinType1	object	Qualité de la surface finie du sous-sol
34	BsmtFinSF1	int	Type 1 pieds carrés finis
35	BsmtFinType2	object	Qualité de la deuxième zone finie (si elle existe)
36	BsmtFinSF2	int	Type 2 pieds carrés finis
37	BsmtUnfSF	int	Pieds carrés non finis de la superficie du sous-sol
38	TotalBsmtSF	int	Superficie totale du sous-sol en pieds carrés
39	Heating	object	Type de chauffage
40	HeatingQC	object	Qualité et état du chauffage

41	CentralAir	object	Climatisation centrale
42	Electrical	object	Système électrique
43	1stFlrSF	int	Pieds carrés du premier étage
44	2ndFlrSF	int	Pieds carrés du deuxième étage
45	LowQualFinSF	int	Pieds carrés finis de qualité inférieure (tous les étages)
46	GrLivArea	int	Surface habitable au-dessus du niveau du sol (pieds carrés)
47	BsmtFullBath	int	Salles de bains complètes au sous-sol
48	BsmtHalfBath	int	Demi-salles de bain au sous-sol
49	FullBath	int	Salles de bains complètes au-dessus du sol
50	HalfBath	int	Demi-bains au-dessus du niveau du sol
51	BedroomAbvGr	int	Nombre de chambres à coucher au-dessus du niveau du sous-sol
52	KitchenAbvGr	int	Nombre de cuisines
53	KitchenQual	object	Qualité de la cuisine
54	TotRmsAbvGrd	int	Nombre total de pièces au-dessus du niveau du sol (ne comprend pas les salles de bain)
55	Functional	object	Evaluation de la fonctionnalité de la maison

56	Fireplaces	int	Nombre de cheminées
57	FireplaceQu	object	Qualité de la cheminée
58	GarageType	object	Emplacement du garage
59	GarageYrBlt	float	Année de construction du garage
60	GarageFinish	object	Finition intérieure du garage
61	GarageCars	int	Taille du garage en nombre de voitures
62	GarageArea	int	Taille du garage en pieds carrés
63	GarageQual	object	Qualité du garage
64	GarageCond	object	État du garage
65	PavedDrive	object	Allée pavée
66	WoodDeckSF	int	Surface de la terrasse en bois en pieds carrés
67	OpenPorchSF	int	Surface du porche ouvert en pieds carrés
68	EnclosedPorch	int	Surface du porche fermé en pieds carrés
69	3SsnPorch	int	Surface du porche trois saisons en pieds carrés
70	ScreenPorch	int	Surface de la véranda en pieds carrés
71	PoolArea	int	Surface de la piscine en pieds carrés
72	PoolQC	object	Qualité de la piscine
73	Fence	object	Qualité des clôtures

74	MiscFeature	object	
75	MiscVal	int	Caractéristiques diverses non couvertes par d'autres catégories
76	MoSold	int	Valeur de l'élément divers
77	YrSold	int	Mois de vente
78	SaleType	object	Année de vente
79	SaleCondition	object	Conditions de vente

III. Explication détaillée du code

- Lire les données

src/data/make_dataset.py

```
# Importer d'abord les paquets python requis
import pandas as pd

def read_data():
  # Ensemble d'entraînement de lecture
  df_train = pd.read_csv('../../data/train.csv')
  # Ensemble d'essais de lecture
  df_test = pd.read_csv('../../data/test.csv')

  # Remplacer la valeur ID originale
  df_train.set_index('Id', inplace=True)
  df_test.set_index('Id', inplace=True)

  return df_train, df_test
```

- Prétraitement des données

src/features/build_features.py

```
# Importer d'abord les paquets python requis
import pandas as pd
```

```
from src.data import make_dataset

df_train, df_test = make_dataset.dataprocess()
print(df_train.head())
```

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	...	PoolArea	PoolQC	Fence	MiscFeature	MiscV
1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	
2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	...	0	NaN	NaN	NaN	
3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	...	0	NaN	NaN	NaN	
4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	...	0	NaN	NaN	NaN	
5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	...	0	NaN	NaN	NaN	

5 rows × 80 columns

Comme vous pouvez le voir ici, les données ont un total de 79 caractéristiques et il y a des valeurs manquantes, donc nous devons faire le traitement des valeurs manquantes.

```
def eda(df):
  if isinstance(df, pd.DataFrame):
    # Toutes les valeurs manquantes
    total_na = df.isna().sum().sum()
    # Lignes, colonnes
    print("Dimension : %d Lignes, %d Colonnes" % (df.shape[0], df.shape[1]))
    # Toutes les valeurs manquantes
    print("Total NA valeurs : %d " % (total_na))
    print("%38s %10s   %10s %10s %8s" % ("Colonne", "Type de donnée", "Modalité", "NA valeurs", "%isNaN"))
    # Nom de l'étiquette
    col_name = df.columns
    # Type de données
    dtyp = df.dtypes
    # Nombre de non-duplicatas
    uniq = df.nunique()
    # Nombre de valeurs manquantes par colonne
    na_val = df.isna().sum()
    # Pourcentage manquant par colonne
    percent_na = round((df.isna().sum())/len(df)*100, 2)
    for i in range(len(df.columns)):
      print("%38s %10s     %10s %10s %10s" % (col_name[i], dtyp[i], uniq[i], na_val[i], percent_na[i]))

eda(df_train)
```

```
Dimension : 1460 Lignes, 80 Colonnes
Total NA valeurs : 6965
                            Colonne Type de donnée     Modalité NA valeurs   %isNaN
                         MSSubClass      int64             15          0        0.0
                           MSZoning     object              5          0        0.0
                        LotFrontage    float64            110        259      17.74
                            LotArea      int64           1073          0        0.0
                             Street     object              2          0        0.0
                              Alley     object              2       1369      93.77
                           LotShape     object              4          0        0.0
                        LandContour     object              4          0        0.0
                          Utilities     object              2          0        0.0
                          LotConfig     object              5          0        0.0
                          LandSlope     object              3          0        0.0
                       Neighborhood     object             25          0        0.0
                         Condition1     object              9          0        0.0
```

Supprimer les colonnes présentant un pourcentage élevé de valeurs manquantes (>30%)

```python
# Création d'une liste contenant les colonnes avec un pourcentage élevé de valeurs manquantes
def drop_nan_features(df):
  col_name = df.columns
  na_val = df.isna().sum()
  percent_na = round((df.isna().sum()) / len(df) * 100, 2)
  to_drop = []
  for i in range(len(df.columns)):
    if percent_na[i] > 30:
      to_drop.append(col_name[i])

  return to_drop

high_nan_features = drop_nan_features(df_train)

# Suppression des colonnes avec un pourcentage élevé de valeurs manquantes
def drop_columns(df):
  return df.drop(columns=high_nan_features, inplace=True)

drop_columns(df_train)
print(df_train)
```

```
      MSSubClass MSZoning  LotFrontage  ...  SaleType SaleCondition SalePrice
Id                                      ...
1             60       RL         65.0  ...        WD        Normal    208500
2             20       RL         80.0  ...        WD        Normal    181500
3             60       RL         68.0  ...        WD        Normal    223500
4             70       RL         60.0  ...        WD       Abnorml    140000
5             60       RL         84.0  ...        WD        Normal    250000
...          ...      ...          ...  ...       ...           ...       ...
1456          60       RL         62.0  ...        WD        Normal    175000
1457          20       RL         85.0  ...        WD        Normal    210000
1458          70       RL         66.0  ...        WD        Normal    266500
1459          20       RL         68.0  ...        WD        Normal    142125
1460          20       RL         75.0  ...        WD        Normal    147500

[1460 rows x 75 columns]
```

Extraction des variables catégorielles et continues

```
def extract_cat_num(df):
  # Extrait les colonnes qui sont de type objet
  categorical=[col for col in df.columns if df[col].dtype=='object']
  # Extraction des colonnes qui ne sont pas de type objet
  numerical=[col for col in df.columns if df[col].dtype!='object']
  return categorical,numerical

categorical,numerical=extract_cat_num(df_train)
```

Diviser les ensembles de formation et de test

```
from sklearn.model_selection import train_test_split
# X Toutes les caractéristiques
# y Prix des logements
X, y = df_train.iloc[:,:-1], df_train.iloc[:,-1]
# 20% pour les ensemble d'essais
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Modèles de formation

src/models/train_model.py

fancyimpute:

est une boîte à outils tierce pour python qui fournit des implémentations de divers algorithmes de calcul matriciel et de remplissage.

IterativeImputer:

Une stratégie pour estimer les valeurs manquantes en modélisant chaque caractéristique avec une valeur manquante comme une fonction des autres caractéristiques dans un mode round-robin.

sklearn.compose.ColumnTransformer:

peut être utilisé pour construire un convertisseur de données qui permet à différentes colonnes ou sous-ensembles de colonnes de l'entrée d'être transformés individuellement, et les caractéristiques générées par chaque convertisseur seront concaténées pour former un espace de caractéristiques unique. Cela est utile pour les données hétérogènes ou en colonnes, où plusieurs mécanismes d'extraction de caractéristiques ou de transformations peuvent être combinés en un seul convertisseur.

sklearn.pipeline.Pipeline:

Le pipeline peut enchaîner de nombreux modèles algorithmiques.

sklearn.preprocessing.MinMaxScaler:

effectue la distribution des caractéristiques dans une plage de valeurs minimales et maximales données. Cette valeur est généralement comprise entre [0,1].

category_encoders.TargetEncoder:

Le codage cible est une méthode de codage des variables de catégorie basée non seulement sur les valeurs des caractéristiques elles-mêmes, mais aussi sur les variables dépendantes correspondantes.

sklearn.ensemble.RandomForestRegressor:

Une forêt aléatoire est un méta-estimateur qui fait correspondre plusieurs arbres de décision de classification sur différents sous-échantillons d'un ensemble de données et utilise des moyens pour améliorer la précision des prédictions et contrôler l'overfitting.

sklearn.ensemble.GradientBoostingRegressor:

Cet estimateur construit un modèle additif de manière progressive, par étapes, et permet l'optimisation de fonctions de perte différentiables arbitraires. À chaque étape, un arbre de régression est ajusté sur le gradient négatif de la fonction de perte donnée.

sklearn.tree.DecisionTreeRegressor:

Le modèle arborescent permet une sortie intuitive du processus de décision, rendant les prédictions interprétables.

```
from fancyimpute import IterativeImputer as MICE

# Entraîner le modèle
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
import category_encoders as ce
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from src.features import build_features

X_train, X_test, y_train, y_test = build_features.getdata()
categorical_features, numerical_features = build_features.getfeatures()

# Prétraitement des colonnes numériques,
# mise à l'échelle (normalisation des caractéristiques à un certain intervalle de valeurs)
# et imputation des valeurs manquantes (interpolation des valeurs manquantes dans les données par l'interpolateur).
numerical_transformer = Pipeline(steps=[
  ('scaler', MinMaxScaler()),
  ('mice_imputer', MICE())])

# Prétraitement des variables catégorielles,
```

```
# leur codage (remplacement des caractéristiques catégorielles
# par des probabilités postérieures correspondant aux valeurs cibles)
# et imputation des valeurs manquantes.
categorical_transformer = Pipeline(steps=[
  ('target_encoder', ce.TargetEncoder(handle_unknown='ignore')),
  ('mice_imputer', MICE())])

# Consolidation des étapes de prétraitement
preprocessor = ColumnTransformer(
  transformers=[
     ('num', numerical_transformer, numerical_features),
     ('cat', categorical_transformer, categorical_features)],
  )

# Créer un pipeline de prétraitement et de formation
# Formation sur les pipelines
# Convertisseurs : convertir les données dans le format requis pour les modèles d'apprentissage
automatique Estimateurs :
# modèles d'apprentissage automatique (algorithme de forêt aléatoire)
pipeline1 = Pipeline(steps=[('preprocessor', preprocessor),
                     ('regressor', RandomForestRegressor())])


pipeline2 = Pipeline(steps=[('preprocessor', preprocessor),
                     ('regressor',DecisionTreeRegressor())])

pipeline3 = Pipeline(steps=[('preprocessor', preprocessor),
                     ('regressor',GradientBoostingRegressor())])

# Random forest regression
def get_model1():
  # ajuster le pipeline pour entraîner un modèle de régression linéaire sur l'ensemble d'entraînement
  model_rf = pipeline1.fit(X_train, y_train)
  return model_rf

# Decision tree regression
def get_model2():
  # ajuster le pipeline pour entraîner un modèle de régression linéaire sur l'ensemble d'entraînement
  model_rf = pipeline2.fit(X_train, y_train)
  return model_rf

# Gradient boosting regression
def get_model3():
  # ajuster le pipeline pour entraîner un modèle de régression linéaire sur l'ensemble d'entraînement
  model_rf = pipeline2.fit(X_train, y_train)
  return model_rf
```

- Modèles prédictifs

src/models/predict_model.py

MSE:

MSE (Mean Squared Error), l'erreur quadratique moyenne, est la somme des carrés de toutes les erreurs de l'échantillon (la différence entre la valeur réelle et la valeur prédite), puis prise comme valeur moyenne.

RMSE:

RMSE (Root Mean Squared Error), la racine carrée de l'erreur quadratique moyenne, c'est-à-dire la racine carrée de la MSE.

MAE:

L'erreur absolue moyenne (MAE) est une mesure des erreurs entre des observations appariées exprimant le même phénomène.

R2 Score:

est le coefficient de détermination, utilisé pour indiquer le score de l'ajustement du modèle, les valeurs les plus élevées indiquant un meilleur ajustement du modèle, jusqu'à 1, éventuellement négatif.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
from src.models import train_model
from src.features import build_features

X_train, X_test, y_train, y_test = build_features.getdata()

model_rf1 = train_model.get_model1()
# Random forest regression
predictions_rf1 = model_rf1.predict(X_test)

model_rf2 = train_model.get_model2()
# Decision tree regression
predictions_rf2 = model_rf2.predict(X_test)
```

```
model_rf3 = train_model.get_model3()
# Gradient boosting regression
predictions_rf3 = model_rf3.predict(X_test)

# Afficher les métriques
# Erreur quadratique moyenne (vraie, prédite)
mse = mean_squared_error(y_test, predictions_rf1)
# Racine carrée de l'erreur quadratique moyenne
rmse = np.sqrt(mse)
print("Random forest regression: ")
print("RMSE:", rmse)
# Erreur absolue moyenne
mae = mean_absolute_error(y_test, predictions_rf1)
print("MAE:", mae)
# R2 coefficient de détermination (qualité de l'ajustement)
r2 = r2_score(y_test, predictions_rf1)
print("R2:", r2)
print("----------------------------")
# Afficher les métriques
# Erreur quadratique moyenne (vraie, prédite)
mse = mean_squared_error(y_test, predictions_rf2)
# Racine carrée de l'erreur quadratique moyenne
rmse = np.sqrt(mse)
print("Decision tree regression: ")
print("RMSE:", rmse)
# Erreur absolue moyenne
mae = mean_absolute_error(y_test, predictions_rf2)
print("MAE:", mae)
# R2 coefficient de détermination (qualité de l'ajustement)
r2 = r2_score(y_test, predictions_rf2)
print("R2:", r2)
print("----------------------------")
# Afficher les métriques
# Erreur quadratique moyenne (vraie, prédite)
mse = mean_squared_error(y_test, predictions_rf3)
# Racine carrée de l'erreur quadratique moyenne
rmse = np.sqrt(mse)
print("Gradient boosting regression: ")
print("RMSE:", rmse)
# Erreur absolue moyenne
mae = mean_absolute_error(y_test, predictions_rf3)
print("MAE:", mae)
# R2 coefficient de détermination (qualité de l'ajustement)
r2 = r2_score(y_test, predictions_rf3)
print("R2:", r2)
```

```
Random forest regression:
RMSE: 28102.836238245567
MAE: 16762.67270547945
R2: 0.897035747463565 3
-----------------------------
Decision tree regression:
RMSE: 37156.299680151584
MAE: 24845.37328767123
R2: 0.8200088798534261
-----------------------------
Gradient boosting regression:
RMSE: 38786.49170017229
MAE: 24988.304794520547
R2: 0.803868584233365
```

- Visualisation du modèle

src/visualization/visualize.py

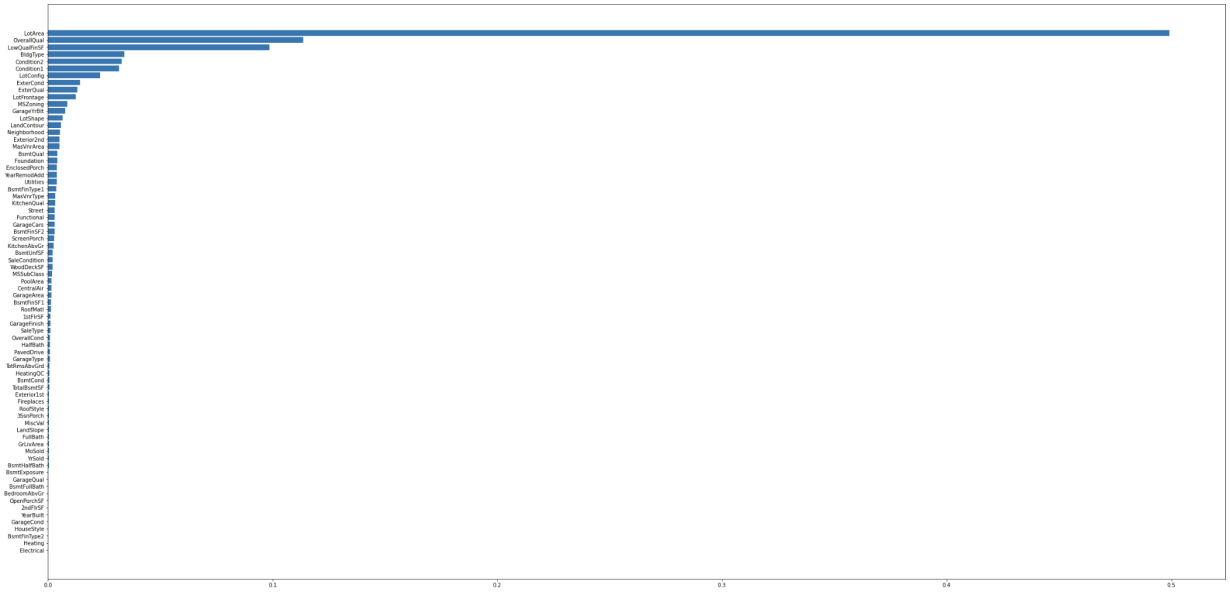
```
import matplotlib.pyplot as plt
import numpy as np
from src.features import build_features
from src.models import predict_model
from src.models.train_model import pipeline1

X_train, X_test, y_train, y_test = build_features.getdata()
predictions_rf = predict_model.predictions_rf1
```

```
# Plot predicted vs actual
plt.scatter(y_test, predictions_rf)
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.title('Sale Price Predictions')
z = np.polyfit(y_test, predictions_rf, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```


```
# On identifie les variables les plus pertinentes dans notre modèle
feature_importance = pipeline1.steps[1][1].feature_importances_
# Toutes les colonnes sont triées et la plus triée est la variable la plus pertinente.
sorted_idx = pipeline1.steps[1][1].feature_importances_.argsort()
print(sorted_idx)
plt.figure(figsize=(40,20))
plt.barh(X_train.columns[sorted_idx], feature_importance[sorted_idx])
```

```
[40 45 37 33 14 17 61 64 42 49 66 30 60 46 36 71 54  9 38 21 47 48 19 69
 55 29 44 41 52 62 16 72 57 70 59 20 32 39  0 68 35 63 18 50 53 73 58 34
 51 27 67  4 31 65 23  7 24 28 10  5  6 22  1 56  2 25 26  8 11 12 13 43
 15  3]
```



- Appliquer les modèles aux données réelles

src/test/test.py

```
from src.models import train_model
from src.data import make_dataset

df_train, df_test = make_dataset.read_data()

model_rf = train_model.get_model1()
# Appliquer à un ensemble de tests réels
predictions_rf = model_rf.predict(df_test)
df_test['SalePrice'] = predictions_rf
submission_rf = df_test['SalePrice'].to_frame()
submission_rf.to_csv('submission_rf.csv')
```

	A	B	C
1	Id	SalePrice	
2	1461	127903.1	
3	1462	156368.5	
4	1463	187021.2	
5	1464	190620	
6	1465	201521	
7	1466	183266.9	
8	1467	173354.4	
9	1468	177790.3	
10	1469	181355.8	
11	1470	123755	
12	1471	183483.2	
13	1472	91343	
14	1473	100045	
15	1474	149316.5	
16	1475	128384.7	
17	1476	367443.5	
18	1477	263213.3	
19	1478	295898.9	
20	1479	266854.1	
21	1480	440198.8	
22	1481	303421.3	
23	1482	214302.4	
24	1483	183620.3	
25	1484	166993.5	
26	1485	172000.6	
27	1486	201361.6	
28	1487	351832.6	
29	1488	243505.7	
30	1489	205482.6	
31	1490	211931.2	
32	1491	192058.2	
33	1492	118926	

submission_rf

IV. Conclusion

	RMSE	MAE	R2
Random forest regression	28102.84	16762.67	0.90
Decision tree regression	37156.30	24845.37	0.82
Gradient boosting regression	38786.49	24988.30	0.80

J'ai constaté que le score obtenu avec le Random forest regression était en fait proche de 90, ce qui est le meilleur modèle que nous ayons pu obtenir. Le deuxième est l'algorithme de Decision tree regression, qui atteint également environ 82, donc pour la prédiction du prix des maisons à Ames, Iowa, l'utilisation du Random forest regression est la meilleure, et c'est de loin le meilleur modèle que j'ai rencontré.

La variable la plus pertinente pour le prix des maisons est la taille du terrain（LotArea）.