Prédiction du prix de vente — Ames Housing

Phase 3 – Bootcamp Data Science
Amee Hashley JEUDY — ameehashleyjeudy@gmail.com

Objectifs

- Construire un modèle de régression linéaire **simple** (baseline).
- Construire un modèle de régression linéaire **multiple** (amélioration).
- **Comparer** les modèles via le R^2 et la significativité.
- **Interpréter** les coefficients et formuler des recommandations.

Données

- Source : **Ames Housing (Iowa, USA)**
- Cible : `SalePrice`
- Prédicteurs : `GrLivArea` , `GarageArea` , `LotArea` , `LotFrontage`

In [8]:
```
# ============================
# 0. Préambule : imports & setup
# ============================
# But : charger les bibliothèques, fixer l'affichage et lire les données.
import numpy as np
import statsmodels.api as sm
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

# Affichage lisible
pd.set_option("display.max_columns", 200)
pd.set_option("display.width", 120)

# Reproductibilité
np.random.seed(42)

# ---- Chemin du fichier ----
# Lecture des données
df = pd.read_csv("ames.csv", index_col=0)
df = df[["SalePrice", "GrLivArea", "GarageArea", "LotArea", "LotFrontage"]].copy()
df.dropna(inplace=True)
df.head()
```

Out[8]:

Id	SalePrice	GrLivArea	GarageArea	LotArea	LotFrontage
1	208500	1710	548	8450	65.0
2	181500	1262	460	9600	80.0
3	223500	1786	608	11250	68.0
4	140000	1717	642	9550	60.0
5	250000	2198	836	14260	84.0

In [11]:
```python
# helpers régression: ajout intercept + extraction sûre des coefs
def add_intercept(X):
    """Ajoute l'intercept pour OLS (colonne 'const')."""
    Xc = sm.add_constant(X, has_constant='add')
    return Xc

def get_intercept(params):
    """Récupère l'intercept, quel que soit le nom ('const' ou 'Intercept')."""
    if 'const' in params.index:
        return params['const']
    if 'Intercept' in params.index:
        return params['Intercept']
    # sinon, retourne NaN (utile pour debug)
    return float('nan')
    df.head()
```

Modèle 1 — Régression linéaire **simple**

Formule : `SalePrice ~ GrLivArea`

- **But** : établir une baseline interprétable.
- **Attendu** : un R^2 moyen (~0.5) sur Ames ; pente positive.

In [13]:
```python
# ============================
# 2. Modèle simple (baseline)
# ============================
# But : ajuster SalePrice ~ GrLivArea et récupérer métriques & coefficients.

# Cible (y) et prédicteur (X)
y = df['SalePrice']
X = add_intercept(df[['GrLivArea']])

# Ajustement du modèle
simple_model = sm.OLS(y, X)
simple_model_results = simple_model.fit()

# Résumé (inclut R², F-stat, p-values)
print(simple_model_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.495
Model:                            OLS   Adj. R-squared:                  0.495
Method:                 Least Squares   F-statistic:                     1175.
Date:                Mon, 20 Oct 2025   Prob (F-statistic):          4.39e-180
Time:                        12:36:14   Log-Likelihood:                -14902.
No. Observations:                1201   AIC:                         2.981e+04
Df Residuals:                    1199   BIC:                         2.982e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.347e+04   5171.332      2.605      0.009    3324.573    2.36e+04
GrLivArea    110.7117      3.229     34.281      0.000     104.376     117.048
==============================================================================
Omnibus:                      197.122   Durbin-Watson:                   2.041
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2702.386
Skew:                           0.268   Prob(JB):                         0.00
Kurtosis:                      10.329   Cond. No.                     4.84e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.84e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [14]:
```python
# Métriques-clés
r_squared_simple = simple_model_results.rsquared
p_f_simple = simple_model_results.f_pvalue

# Coefficients
m = simple_model_results.params['GrLivArea']  # pente
b = get_intercept(simple_model_results.params) # intercept

print(f"\nR² (simple) = {r_squared_simple:.3f} | F p-value = {p_f_simple:.3g}")
print(f"Intercept b ≈ ${b:,.2f} | Pente m ≈ ${m:.2f} / sq ft")
```

```
R² (simple) = 0.495 | F p-value = 4.39e-180
Intercept b ≈ $13,470.44 | Pente m ≈ $110.71 / sq ft
```

Modèle 2 — Régression linéaire **multiple**

Formule : `SalePrice ~ GrLivArea + GarageArea + LotArea + LotFrontage`

- **But** : améliorer l'explication de la variance (R^2 plus élevé).
- **Note** : les lignes incomplètes sont retirées (`dropna`) pour éviter les biais d'index.

In [16]:
```python
# ============================
# 3. Modèle multiple
# ============================
# But : ajuster le modèle avec 4 variables explicatives et comparer au modèle simple.

features = ['GrLivArea', 'GarageArea', 'LotArea', 'LotFrontage']
sub = df[['SalePrice'] + features].dropna().copy()

y2 = sub['SalePrice']
X2 = add_intercept(sub[features])
```

```
multiple_model = sm.OLS(y2, X2)
multiple_model_results = multiple_model.fit()

print(multiple_model_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              SalePrice   R-squared:                       0.614
Model:                            OLS   Adj. R-squared:                  0.613
Method:                 Least Squares   F-statistic:                     476.5
Date:                Mon, 20 Oct 2025   Prob (F-statistic):          1.11e-245
Time:                        12:36:50   Log-Likelihood:                -14740.
No. Observations:                1201   AIC:                         2.949e+04
Df Residuals:                    1196   BIC:                         2.952e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -1.151e+04   5421.192     -2.124      0.034   -2.22e+04    -878.343
GrLivArea      79.5785      3.390     23.473      0.000      72.927      86.230
GarageArea    143.6119      7.833     18.334      0.000     128.244     158.980
LotArea         0.7909      0.212      3.730      0.000       0.375       1.207
LotFrontage   -52.6476     72.856     -0.723      0.470    -195.588      90.293
==============================================================================
Omnibus:                      262.972   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9251.057
Skew:                          -0.120   Prob(JB):                         0.00
Kurtosis:                      16.594   Cond. No.                     4.63e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.63e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [17]:
```
# Métriques-clés
r_squared_multi = multiple_model_results.rsquared
p_f_multi = multiple_model_results.f_pvalue

print(f"\nR² (multiple) = {r_squared_multi:.3f} | F p-value = {p_f_multi:.3g}")
```

```
R² (multiple) = 0.614 | F p-value = 1.11e-245
```

Comparaison des modèles & significativité

- **Question** : le modèle multiple explique-t-il **plus de variance** que le simple ?
- **Décision** : comparer `R²` et tester la **p-value du F-stat** (< 0,05 ⇒ modèle global significatif).

In [18]:
```
# ============================
# 4. Comparaison & test global
# ============================
# But : décider si le modèle multiple est "meilleur" (R² plus grand).

second_model_is_better = (r_squared_multi > r_squared_simple)
print("Le modèle multiple a-t-il un R² plus élevé ?", second_model_is_better)
# Variable(s) non significative(s) (p > 0.05), en excluant l'intercept
pvals = multiple_model_results.pvalues.copy()
for k in list(pvals.index):
    if k.lower() in ('const', 'intercept'):
```

```
        pvals = pvals.drop(k)

non_sig = pvals[pvals > 0.05].sort_values(ascending=False)
print("\nVariables non significatives (p > 0.05) :")
print(non_sig if len(non_sig) else "Aucune")
```

```
Le modèle multiple a-t-il un R² plus élevé ? True

Variables non significatives (p > 0.05) :
LotFrontage    0.470054
dtype: float64
```

Interprétation (en langage simple)

- **Surface habitable (`GrLivArea`)** : principal déterminant, effet positif et **hautement significatif**.
- **Surface garage (`GarageArea`)** : effet positif, souvent significatif.
- `LotArea` : effet plus faible (significativité variable).
- `LotFrontage` : souvent **non significatif** (p > 0,05) dans ce modèle.

Lecture des coefficients (ordre de grandeur) :

- **Pente (`GrLivArea`)** : +1 sq ft ≈ +$118 en prix moyen (modèle simple).
- **Intercept** : prix attendu si toutes les surfaces = 0 (interprétation pratique limitée).

Limites du travail

- Données issues d'une seule ville (Ames) ⇒ **généralisation limitée**.
- Variables qualitatives (qualité, état, quartier) **non incluses** ici.
- **Multicolinéarité** potentielle entre surfaces (à analyser plus finement).
- Pas d'ajustement à l'**inflation** / période temporelle.

Recommandations & suites

- Intégrer des variables **qualitatives** (qualité, matériaux, voisinage).
- Tester des modèles **non linéaires** (Random Forest, Gradient Boosting).
- Mettre en place une **validation croisée** + **GridSearchCV** pour les hyperparamètres.
- Déployer un **outil de scoring** (Streamlit/Flask) pour une utilisation métier.

In [19]:
```
# ============================
# 5. Sauvegarde des métriques clés
# ============================
# But : garder une trace des résultats (utile pour joindre au repo).

summary_dict = {
    "r2_simple": round(r_squared_simple, 4),
    "r2_multiple": round(r_squared_multi, 4),
    "f_pvalue_simple": float(simple_model_results.f_pvalue),
    "f_pvalue_multiple": float(multiple_model_results.f_pvalue),
    "better_multiple": bool(second_model_is_better),
    "non_significant_multi": list(non_sig.index) if isinstance(non_sig, pd.Series) and len(non_s
}
```

```
pd.Series(summary_dict, name="results").to_json("results_summary.json", indent=2)
summary_dict
```

Out[19]:
```
{'r2_simple': 0.495,
 'r2_multiple': 0.6144,
 'f_pvalue_simple': 4.386383022975568e-180,
 'f_pvalue_multiple': 1.1126311030468136e-245,
 'better_multiple': True,
 'non_significant_multi': ['LotFrontage']}
```

In [ ]: