Manuel de Préparation: Régression Linéaire - Simple et Multiple

Introduction

La régression linéaire est l'outil fondamental de la modélisation statistique. Elle permet de quantifier la relation entre une variable dépendante (Y) et une ou plusieurs variables indépendantes (X). Dans le contexte bancaire, elle est utilisée pour la prévision, l'évaluation des risques, et l'analyse causale.

Partie 1: Régression Linéaire Simple

1.1 Définition et Objectif

Modèle:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Où:
- Y = Variable dépendante (à prédire)
- X = Variable indépendante (explicative)
- β_0 = Ordonnée à l'origine (intercept)
- β_1 = Pente (coefficient de régression)
- ε = Erreur résiduelle (terme d'erreur)

Objectifs: - Quantifier la relation entre X et Y - Prédire Y à partir de X - Tester si la relation est statistiquement significative

---

1.2 Méthode des Moindres Carrés Ordinaires (MCO/OLS)

Principe: Minimiser la somme des carrés des résidus

$$\text{Minimiser: } \Sigma(Y_i - \hat{Y}_i)^2 = \Sigma\varepsilon_i^2$$

Formules des coefficients:

$$\beta_1 = \Sigma(X_i - \bar{X})(Y_i - \bar{Y}) / \Sigma(X_i - \bar{X})^2$$
$$= \text{Cov}(X,Y) / \text{Var}(X)$$
$$= r \times (Sy/Sx)$$

$$\beta_0 = \bar{Y} - \beta_1\bar{X}$$

1.3 Implémentation Python

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy import stats
```

```python
# Exemple bancaire: Relation entre ancienneté client et solde moyen
np.random.seed(42)
n = 100
anciennete = np.random.uniform(1, 20, n)  # années
solde_moyen = 5000 + 800 * anciennete + np.random.normal(0, 3000, n)

df = pd.DataFrame({
    'anciennete_annees': anciennete,
    'solde_moyen_htg': solde_moyen
})

# Régression avec statsmodels
X = df['anciennete_annees']
y = df['solde_moyen_htg']
X_with_const = sm.add_constant(X)  # Ajouter l'intercept

model = sm.OLS(y, X_with_const).fit()
print(model.summary())
```

Sortie et Interprétation:

```
                            OLS Regression Results
==============================================================================
Dep. Variable:        solde_moyen_htg   R-squared:                       0.412
Model:                            OLS   Adj. R-squared:                  0.406
Method:                 Least Squares   F-statistic:                     68.67
Date:                                   Prob (F-statistic):           2.03e-12
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       4521.234    1156.423      3.909      0.000    2226.128    6816.341
anciennete   823.456      99.394      8.287      0.000     626.143    1020.769
==============================================================================
```

1.4 Interprétation des Résultats

Statistique	Valeur	Interprétation
R^2	0.412	41.2% de la variance de Y est expliquée par X
R^2 ajusté	0.406	R^2 corrigé pour le nombre de variables
F-statistic	68.67	Test global de significativité du modèle
Prob (F)	2.03e-12	p-value < 0.05 → modèle significatif
β_1 (coef anciennete)	823.46	+1 an d'ancienneté = +823 HTG de solde
t-stat	8.287	Test de significativité du coefficient
P>|t|	0.000	p-value < 0.05 → coefficient significatif
IC [0.025, 0.975]	[626, 1021]	Intervalle de confiance à 95% pour β_1

1.5 Mnémotechnique: “CIPT” pour Interpréter

```
C - Coefficient: Magnitude et signe de l'effet
I - Intervalle de confiance: Plage probable
P - P-value: Significativité statistique
T - T-statistic: Force de l'évidence
```

Partie 2: Régression Linéaire Multiple

2.1 Définition

Modèle:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \varepsilon$$

Exemple bancaire:
$$\text{Montant\_Prêt} = \beta_0 + \beta_1(\text{Revenu}) + \beta_2(\text{Ancienneté}) + \beta_3(\text{Score\_Crédit}) + \varepsilon$$

2.2 Implémentation Python Complète

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.stattools import durbin_watson
import matplotlib.pyplot as plt
import seaborn as sns

# Données bancaires simulées
np.random.seed(42)
n = 200

df = pd.DataFrame({
    'revenu_annuel': np.random.uniform(300000, 2000000, n),
    'anciennete_emploi': np.random.uniform(0, 25, n),
    'score_credit': np.random.uniform(500, 850, n),
    'ratio_endettement': np.random.uniform(0.1, 0.6, n),
    'age': np.random.uniform(22, 65, n)
})

# Variable dépendante: Montant prêt approuvé
df['montant_pret'] = (
    50000 +
    0.15 * df['revenu_annuel'] +
    5000 * df['anciennete_emploi'] +
    500 * df['score_credit'] -
    200000 * df['ratio_endettement'] +
    np.random.normal(0, 50000, n)
)

# Régression multiple
X = df[['revenu_annuel', 'anciennete_emploi', 'score_credit', 'ratio_endettement', 'age']]
```

```
y = df['montant_pret']
X_with_const = sm.add_constant(X)

model = sm.OLS(y, X_with_const).fit()
print(model.summary())
```

2.3 Différence R² vs R² Ajusté

```
R² = 1 - (SSR / SST)
   = 1 - Σ(Yᵢ - Ŷᵢ)² / Σ(Yᵢ - Ȳ)²

R² ajusté = 1 - [(1 - R²)(n - 1) / (n - p - 1)]

Où:
- n = nombre d'observations
- p = nombre de variables explicatives
```

Différence clé: - **R²** augmente TOUJOURS quand on ajoute une variable - **R² ajusté** pénalise l'ajout de variables non significatives - Utiliser R² ajusté pour comparer des modèles avec différents nombres de variables

---

Partie 3: Hypothèses de la Régression (LINE)

3.1 Mnémotechnique: "LINE"

```
L - Linéarité: Relation linéaire entre X et Y
I - Indépendance: Les observations sont indépendantes
N - Normalité: Les résidus suivent une loi normale
E - Égalité des variances (Homoscédasticité): Variance constante des résidus
```

3.2 Diagnostic: Linéarité

```
# Graphique des résidus vs valeurs ajustées
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# 1. Résidus vs Valeurs ajustées (Linéarité + Homoscédasticité)
axes[0, 0].scatter(model.fittedvalues, model.resid, alpha=0.5)
axes[0, 0].axhline(y=0, color='r', linestyle='--')
axes[0, 0].set_xlabel('Valeurs ajustées')
axes[0, 0].set_ylabel('Résidus')
axes[0, 0].set_title('Résidus vs Valeurs Ajustées')

# Pattern à observer:
# - Points aléatoirement dispersés autour de 0 = OK
# - Pattern courbe = Non-linéarité (transformer X ou Y)
# - Forme d'entonnoir = Hétéroscédasticité
```

3.3 Diagnostic: Indépendance (Durbin-Watson)

```python
from statsmodels.stats.stattools import durbin_watson

dw = durbin_watson(model.resid)
print(f"Durbin-Watson: {dw:.3f}")

# Interprétation:
# DW ≈ 2.0: Pas d'autocorrélation (IDÉAL)
# DW < 1.5: Autocorrélation positive (problème)
# DW > 2.5: Autocorrélation négative (problème)
```

Tableau d'interprétation:

Valeur DW	Interprétation	Action
1.5 - 2.5	Acceptable	Continuer
< 1.5	Autocorrélation positive	Inclure lag, différencier
> 2.5	Autocorrélation négative	Vérifier les données

3.4 Diagnostic: Normalité des Résidus

```python
from scipy import stats

# QQ-Plot
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# QQ-Plot
stats.probplot(model.resid, dist="norm", plot=axes[0])
axes[0].set_title('QQ-Plot des Résidus')

# Histogramme
axes[1].hist(model.resid, bins=30, density=True, alpha=0.7)
axes[1].set_xlabel('Résidus')
axes[1].set_title('Distribution des Résidus')

plt.tight_layout()
plt.show()

# Tests formels
# Shapiro-Wilk (n < 5000)
stat, p_shapiro = stats.shapiro(model.resid)
print(f"Shapiro-Wilk: stat={stat:.4f}, p-value={p_shapiro:.4f}")

# Jarque-Bera (inclus dans summary)
from statsmodels.stats.stattools import jarque_bera
jb, p_jb, skew, kurtosis = jarque_bera(model.resid)
print(f"Jarque-Bera: stat={jb:.4f}, p-value={p_jb:.4f}")

# Interprétation: p-value > 0.05 → Résidus normaux
```

3.5 Diagnostic: Homoscédasticité

```
from statsmodels.stats.diagnostic import het_breuschpagan, het_white

# Test de Breusch-Pagan
bp_stat, bp_pval, f_stat, f_pval = het_breuschpagan(model.resid, X_with_const)
print(f"Breusch-Pagan: stat={bp_stat:.4f}, p-value={bp_pval:.4f}")

# Test de White
white_stat, white_pval, f_stat, f_pval = het_white(model.resid, X_with_const)
print(f"White: stat={white_stat:.4f}, p-value={white_pval:.4f}")

# Interprétation:
# p-value > 0.05 → Homoscédasticité (OK)
# p-value < 0.05 → Hétéroscédasticité (problème)
```

Solutions si Hétéroscédasticité: 1. Transformation logarithmique de Y 2. Régression avec erreurs robustes (Huber-White) 3. Régression pondérée (WLS)

```
# Erreurs robustes (Huber-White)
model_robust = model.get_robustcov_results(cov_type='HC3')
print(model_robust.summary())
```

Partie 4: Multicolinéarité

4.1 Définition et Problème

Multicolinéarité: Corrélation élevée entre les variables explicatives.

Conséquences: - Coefficients instables (variances élevées) - Intervalles de confiance larges - Difficultés d'interprétation - Modèle reste prédictif mais non interprétable

---

4.2 Détection: VIF (Variance Inflation Factor)

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(X):
    """Calcul du VIF pour chaque variable"""
    vif_data = pd.DataFrame()
    vif_data["Variable"] = X.columns
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vif_data.sort_values('VIF', ascending=False)

vif_results = calculate_vif(X)
print(vif_results)
```

Tableau d'interprétation VIF:

VIF	Interprétation	Action
1	Pas de corrélation	Idéal
1 - 5	Corrélation modérée	Acceptable
5 - 10	Corrélation élevée	Attention

VIF	Interprétation	Action
> 10	Multicolinéarité sévère	Intervention requise

4.3 Solutions à la Multicolinéarité

```python
# 1. Supprimer une variable corrélée
# Identifier les paires fortement corrélées
corr_matrix = X.corr()
high_corr = np.where(np.abs(corr_matrix) > 0.7)
high_corr_pairs = [(corr_matrix.columns[x], corr_matrix.columns[y], corr_matrix.iloc[x, y]
                    for x, y in zip(*high_corr) if x != y and x < y]
print("Paires fortement corrélées:", high_corr_pairs)

# 2. Combiner des variables (créer un indice)
# Ex: au lieu de revenu et patrimoine séparés
df['richesse_score'] = (df['revenu_annuel'] + df['patrimoine']) / 2

# 3. Régression Ridge (pénalisation L2)
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)
ridge.fit(X, y)

# 4. ACP pour réduire les dimensions
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)
```

4.4 Mnémotechnique: “VISC” pour Multicolinéarité

```
V - VIF: Calculer pour chaque variable
I - Identifier: Paires corrélées > 0.7
S - Supprimer: Variable redondante
C - Combiner: Créer des indices composites
```

Partie 5: Tests d’Hypothèses en Régression

5.1 Test Global (F-test)

H_0: $\beta_1 = \beta_2 = ... = \beta_p = 0$ (aucune variable n'est significative)
H_1: Au moins un $\beta_j \neq 0$

$$F = (SSR/p) / (SSE/(n-p-1))$$
$$= MSR / MSE$$

Décision:
- Si Prob(F) < 0.05 → Rejeter H_0 → Modèle significatif

5.2 Tests Individuels (t-tests)

```
Pour chaque coefficient βⱼ:

H₀: βⱼ = 0 (variable non significative)
H₁: βⱼ ≠ 0 (variable significative)

t = βⱼ / SE(βⱼ)

Décision:
- Si P>|t| < 0.05 → Variable significative
- Si P>|t| > 0.05 → Variable non significative (peut être retirée)
```

5.3 Sélection de Variables

```python
# Méthode Backward (partir de toutes les variables)
import statsmodels.api as sm

def backward_elimination(X, y, significance_level=0.05):
    """Sélection backward des variables"""
    features = list(X.columns)

    while len(features) > 0:
        X_temp = sm.add_constant(X[features])
        model = sm.OLS(y, X_temp).fit()

        # Trouver la p-value maximale
        pvalues = model.pvalues.drop('const')
        max_pval = pvalues.max()

        if max_pval > significance_level:
            # Retirer la variable avec la p-value la plus élevée
            feature_to_remove = pvalues.idxmax()
            features.remove(feature_to_remove)
            print(f"Retrait de '{feature_to_remove}' (p-value: {max_pval:.4f})")
        else:
            break

    return features

selected_features = backward_elimination(X, y)
print(f"\nVariables sélectionnées: {selected_features}")
```

Partie 6: Prédiction et Intervalles

6.1 Types de Prédiction

```python
# Nouvelles données pour prédiction
new_data = pd.DataFrame({
    'revenu_annuel': [800000, 1500000],
    'anciennete_emploi': [5, 15],
```

```
    'score_credit': [700, 780],
    'ratio_endettement': [0.3, 0.2],
    'age': [35, 50]
})

new_data_const = sm.add_constant(new_data)

# Prédiction ponctuelle
predictions = model.predict(new_data_const)
print("Prédictions:", predictions)

# Intervalle de confiance pour la moyenne (où tombe la moyenne prédite)
pred_summary = model.get_prediction(new_data_const)
pred_ci = pred_summary.summary_frame(alpha=0.05)
print("\nIntervalles de Confiance (moyenne):")
print(pred_ci[['mean', 'mean_ci_lower', 'mean_ci_upper']])

# Intervalle de prédiction (où tombe une observation individuelle)
print("\nIntervalles de Prédiction (individuel):")
print(pred_ci[['mean', 'obs_ci_lower', 'obs_ci_upper']])
```

Différence IC vs IP: - **Intervalle de Confiance:** Incertitude sur E[Y|X] - plus étroit - **Intervalle de Prédiction:** Incertitude sur Y individuel - plus large (inclut ε)

Partie 7: Applications Bancaires

7.1 Cas 1: Prédiction du Solde Client

```
# Modèle: Solde = f(revenu, ancienneté, nb_produits, age)

import pandas as pd
import numpy as np
import statsmodels.api as sm

# Données
df_clients = pd.DataFrame({
    'client_id': range(1, 501),
    'revenu_mensuel': np.random.uniform(25000, 500000, 500),
    'anciennete_mois': np.random.uniform(1, 240, 500),
    'nb_produits': np.random.randint(1, 6, 500),
    'age': np.random.uniform(20, 70, 500),
    'segment': np.random.choice(['Retail', 'Premium', 'VIP'], 500)
})

# Variable cible
df_clients['solde_moyen'] = (
    10000 +
    0.08 * df_clients['revenu_mensuel'] +
    100 * df_clients['anciennete_mois'] +
    15000 * df_clients['nb_produits'] +
    np.random.normal(0, 20000, 500)
)
```

```python
# Variables explicatives
X = df_clients[['revenu_mensuel', 'anciennete_mois', 'nb_produits', 'age']]
y = df_clients['solde_moyen']

# Modèle
X_const = sm.add_constant(X)
model_solde = sm.OLS(y, X_const).fit()
print(model_solde.summary())

# Interprétation business:
# - Pour chaque 1 HTG de revenu supplémentaire, le solde augmente de β₁ HTG
# - Pour chaque mois d'ancienneté, le solde augmente de β₂ HTG
# - Chaque produit détenu ajoute β₃ HTG au solde
```

7.2 Cas 2: Facteurs de Défaut (Logistique)

Pour une variable binaire (défaut/non défaut), utiliser la régression logistique:

```python
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm

# Données de crédit
df_credit = pd.DataFrame({
    'revenu': np.random.uniform(15000, 200000, 500),
    'dette_revenu_ratio': np.random.uniform(0.1, 0.8, 500),
    'nb_retards_12m': np.random.poisson(1, 500),
    'anciennete_credit': np.random.uniform(0, 20, 500),
})

# Variable cible: défaut (binaire)
prob_defaut = 1 / (1 + np.exp(-(
    -3 + 0.00001 * df_credit['revenu'] +
    3 * df_credit['dette_revenu_ratio'] +
    0.5 * df_credit['nb_retards_12m'] -
    0.1 * df_credit['anciennete_credit']
)))
df_credit['defaut'] = (np.random.uniform(0, 1, 500) < prob_defaut).astype(int)

# Régression logistique avec statsmodels
X = df_credit[['revenu', 'dette_revenu_ratio', 'nb_retards_12m', 'anciennete_credit']]
y = df_credit['defaut']
X_const = sm.add_constant(X)

logit_model = sm.Logit(y, X_const).fit()
print(logit_model.summary())

# Interprétation: Odds Ratios
odds_ratios = np.exp(logit_model.params)
print("\nOdds Ratios:")
print(odds_ratios)
```

7.3 Cas 3: Estimation du Montant de Prêt

```
# Modèle bancaire: Combien prêter à ce client?

df_prets = pd.DataFrame({
    'revenu_annuel': np.random.uniform(300000, 3000000, 300),
    'apport_personnel': np.random.uniform(50000, 500000, 300),
    'score_credit': np.random.uniform(500, 850, 300),
    'dette_existante': np.random.uniform(0, 500000, 300),
    'anciennete_emploi_ans': np.random.uniform(0, 30, 300)
})

# Montant accordé (politique bancaire simulée)
df_prets['montant_accorde'] = np.maximum(0,
    -50000 +
    0.4 * df_prets['revenu_annuel'] +
    1.5 * df_prets['apport_personnel'] +
    300 * df_prets['score_credit'] -
    0.5 * df_prets['dette_existante'] +
    5000 * df_prets['anciennete_emploi_ans'] +
    np.random.normal(0, 100000, 300)
)

# Modèle
X = df_prets[['revenu_annuel', 'apport_personnel', 'score_credit',
              'dette_existante', 'anciennete_emploi_ans']]
y = df_prets['montant_accorde']

X_const = sm.add_constant(X)
model_pret = sm.OLS(y, X_const).fit()

# Diagnostics complets
print("=" * 60)
print("RAPPORT D'ANALYSE - MODÈLE DE PRÊT")
print("=" * 60)
print(model_pret.summary())

# Validation
print("\n--- DIAGNOSTICS ---")
print(f"Durbin-Watson: {durbin_watson(model_pret.resid):.3f}")

# VIF
vif = calculate_vif(X)
print("\nVIF (Multicolinéarité):")
print(vif)
```

Partie 8: Formules Essentielles

8.1 Tableau Récapitulatif

Formule	Expression	Utilisation
SST	$\Sigma(Y_i - \bar{Y})^2$	Variation totale
SSR	$\Sigma(\hat{Y}_i - \bar{Y})^2$	Variation expliquée
SSE	$\Sigma(Y_i - \hat{Y}_i)^2$	Variation résiduelle
R^2	1 - SSE/SST = SSR/SST	Pouvoir explicatif
R^2 ajusté	$1 - [(1-R^2)(n-1)/(n-p-1)]$	R^2 corrigé
F-stat	(SSR/p) / (SSE/(n-p-1))	Test global
t-stat	$\hat{\beta}$ / SE($\hat{\beta}$)	Test coefficient
VIF	$1 / (1 - R^2_j)$	Multicolinéarité
DW	$\Sigma(e_t - e_{t-1})^2 / \Sigma e_t^2$	Autocorrélation

8.2 Mnémotechnique Global: “RHINO”

```
R - R² et R² ajusté: Mesurer le pouvoir explicatif
H - Hypothèses LINE: Vérifier avant d'interpréter
I - Intervalles: IC pour coefficients, IP pour prédictions
N - Normalité des résidus: Tests et QQ-plot
O - Outliers et influence: Leverage, Cook's distance
```

Partie 9: Points d’Attention

9.1 Erreurs Courantes à Éviter

Erreur	Problème	Solution
Ignorer les diagnostics	Conclusions invalides	Toujours vérifier LINE
Confondre corrélation et causalité	Mauvaise interprétation	Régression ≠ causalité
VIF > 10 ignoré	Coefficients instables	Réduire multicolinéarité
R^2 comme seul critère	Modèle surajusté	Utiliser R^2 ajusté, AIC, BIC
Extrapolation	Prédictions hors domaine	Prédire dans la plage des X

9.2 Checklist Avant Conclusion

```
□ R² et R² ajusté acceptables (> 0.3 pour données réelles)
□ F-test significatif (p < 0.05)
□ Coefficients significatifs (p < 0.05)
□ VIF < 5 pour toutes les variables
□ Durbin-Watson entre 1.5 et 2.5
□ Résidus normaux (Shapiro-Wilk p > 0.05)
□ Homoscédasticité (Breusch-Pagan p > 0.05)
□ Pas de pattern dans résidus vs fitted
```

Partie 10: Comparaison Régression Linéaire vs Logistique

Aspect	Régression Linéaire	Régression Logistique
Variable Y	Continue	Binaire (0/1)
Fonction	$Y = \beta_0 + \beta_1 X$	$\log(p/(1-p)) = \beta_0 + \beta_1 X$
Prédiction	Valeur numérique	Probabilité (0-1)
Interprétation β	Changement absolu en Y	Changement en log-odds
Métrique	R^2, RMSE, MAE	AUC, Gini, Précision
Usage bancaire	Montant prêt, solde	Défaut oui/non, fraude

Résumé Express: Questions Probables

1. **"Quelles sont les hypothèses de la régression linéaire?"** → LINE: Linéarité, Indépendance, Normalité, Égalité des variances
2. **"Comment détecter la multicolinéarité?"** → VIF > 5 = attention, VIF > 10 = problème
3. **"Différence entre R^2 et R^2 ajusté?"** → R^2 ajusté pénalise l'ajout de variables inutiles
4. **"Qu'est-ce que le test de Durbin-Watson?"** → Détecte l'autocorrélation des résidus (DW ≈ 2 = OK)
5. **"Interprétez β = 0.35 pour le revenu"** → Pour chaque unité de revenu en plus, Y augmente de 0.35 unités (toutes choses égales par ailleurs)
6. **"Différence entre IC et IP?"** → IC pour la moyenne, IP pour une observation (plus large)

Code Express: Analyse Complète

```python
def analyse_regression_complete(X, y, alpha=0.05):
    """
    Analyse de régression complète avec diagnostics
    """
    import statsmodels.api as sm
    from statsmodels.stats.outliers_influence import variance_inflation_factor
    from statsmodels.stats.stattools import durbin_watson
    from statsmodels.stats.diagnostic import het_breuschpagan
    from scipy import stats

    # Ajuster le modèle
    X_const = sm.add_constant(X)
    model = sm.OLS(y, X_const).fit()

    print("=" * 70)
    print("RAPPORT D'ANALYSE DE RÉGRESSION")
    print("=" * 70)

    # Résumé
    print("\n1. RÉSUMÉ DU MODÈLE")
    print(model.summary())
```

```
    # VIF
    print("\n2. MULTICOLINÉARITÉ (VIF)")
    vif_data = pd.DataFrame({
        'Variable': X.columns,
        'VIF': [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    }).sort_values('VIF', ascending=False)
    print(vif_data)
    vif_alert = vif_data[vif_data['VIF'] > 5]
    if len(vif_alert) > 0:
        print("⚠ ATTENTION: Variables avec VIF > 5 détectées!")

    # Durbin-Watson
    dw = durbin_watson(model.resid)
    print(f"\n3. AUTOCORRÉLATION (Durbin-Watson): {dw:.3f}")
    if 1.5 <= dw <= 2.5:
        print("✓ Pas d'autocorrélation détectée")
    else:
        print("⚠ Autocorrélation potentielle!")

    # Normalité
    stat, p_shapiro = stats.shapiro(model.resid)
    print(f"\n4. NORMALITÉ (Shapiro-Wilk): p-value = {p_shapiro:.4f}")
    if p_shapiro > alpha:
        print("✓ Résidus normaux")
    else:
        print("⚠ Résidus non normaux!")

    # Homoscédasticité
    bp_stat, bp_pval, _, _ = het_breuschpagan(model.resid, X_const)
    print(f"\n5. HOMOSCÉDASTICITÉ (Breusch-Pagan): p-value = {bp_pval:.4f}")
    if bp_pval > alpha:
        print("✓ Homoscédasticité respectée")
    else:
        print("⚠ Hétéroscédasticité détectée!")

    return model

# Utilisation:
# model = analyse_regression_complete(X, y)
```
