Generer de data

Pour faire ce TP, on va generer de donnees:

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

# Génération de données aléatoires
np.random.seed(0)  # pour la reproductibilité
X1 = np.random.rand(100)
X2 = np.random.rand(100)
beta0 = 1
beta1 = 2
beta2 = 3
epsilon = np.random.randn(100)
Y = beta0 + beta1 * X1 + beta2 * X2 + epsilon

# Création d'un DataFrame
df = pd.DataFrame({'X1': X1, 'X2': X2, 'Y': Y})
```

Specification du modele

```python # Définition des variables dépendantes et indépendantes X = df[['X1', 'X2']] y = df['Y']

# Ajout d'une constante à nos variables explicatives

X = sm.add_constant(X)

# Spécification du modèle de régression linéaire multiple

model = sm.OLS(y, X)
```

```
<h4 style="color:red;"> Estimation des paramètres du modèle</h4>

```python
# Estimation des paramètres du modèle
results = model.fit()

# Affichage des résultats
print(results.summary())
```

Validation des hypothèses

Linearité: On peut vérifier graphiquement cette hypothèse en traçant les valeurs prédites par rapport aux valeurs observées.

```
plt.scatter(results.fittedvalues, y)
plt.xlabel('Valeurs prédites')
plt.ylabel('Valeurs observées')
plt.title('Valeurs prédites vs Valeurs observées')
plt.show()
```

Indépendance des erreurs: Nous utilisons à nouveau le test de Durbin-Watson. Une valeur proche de 2 indique une absence d'autocorrélation.

```
from statsmodels.stats.stattools import durbin_watson

print('Statistique de Durbin-Watson:', durbin_watson(results.resid))
```

Homoscédasticité: Cette hypothèse peut être vérifiée en traçant les résidus contre les valeurs prédites.

```
plt.scatter(results.fittedvalues, results.resid)
plt.xlabel('Valeurs prédites')
plt.ylabel('Résidus')
plt.title('Résidus vs Valeurs prédites')
plt.show()
```

le test de Breusch-Pagan est un moyen couramment utilisé pour tester l'homoscédasticité, c'est-à-dire l'hypothèse que la variance des erreurs est constante
```

```
from statsmodels.stats.diagnostic import het_breuschpagan

# Calculer le test de Breusch-Pagan
bp_test = het_breuschpagan(results.resid, results.model.exog)

# Imprimer les résultats
labels = ['Statistique de test de Lagrange multiplier', 'p-valeur de LM',
          'Statistique de test à base de F', 'p-valeur de F']
print(dict(zip(labels, bp_test)))
```

Dans cet exemple, `results.resid` est un tableau des résidus du modèle et `results.model.exog` est un tableau des variables exogènes utilisées pour ajuster le modèle, y compris la constante. Le test de Breusch-Pagan retourne quatre valeurs: la statistique de test de Lagrange multiplier, la p-valeur associée, la statistique de test à base de F et sa p-valeur.

Si la p-valeur est inférieure à un seuil donné (souvent 0,05), nous rejetons l'hypothèse nulle d'homoscédasticité.

Normalité des erreurs

vous pouvez utiliser le test de Jarque-Bera pour vérifier l'hypothèse de normalité des résidus. Ce test est basé sur le skewness (asymétrie) et le kurtosis (aplatissement) des données

```
from scipy import stats

# Calculer la statistique de test de Jarque-Bera et la p-valeur
jb_stats = stats.jarque_bera(results.resid)
jb_stats
```

La sortie sera une paire de valeurs. La première est la statistique de test de Jarque-Bera et la seconde est la p-valeur associée. Si la p-valeur est inférieure à un seuil (généralement 0,05), on rejette l'hypothèse nulle de normalité des résidus.

Remarque: Statsmodels fournit également la statistique de test de Jarque-Bera dans le résumé des résultats de la régression. Vous pouvez y accéder avec `results.summary()`.

Absence de multicollinéarité: Nous pouvons vérifier cette hypothèse en utilisant le facteur d'inflation de la variance (VIF). Un VIF supérieur à 5 indique généralement une multicollinéarité.

```
# Calcul du VIF
VIF = pd.DataFrame()
VIF["Variable"] = X.columns
```

```
VIF["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(VIF)
```

Réalisation de différents tests

Test de Student: On peut utiliser la méthode t_test pour effectuer le test de Student.

```
# Test de Student pour le coefficient associé à 'hp'
hypotheses = '(hp = 0)'
t_test = results.t_test(hypotheses)
print(t_test)
```

Test de Fisher: Le test F est déjà inclus dans le résumé des résultats.

```
print('Statistique de F:', results.fvalue)
print('p-valeur de F:', results.f_pvalue)
```

Le coefficient de détermination: R carré (R-squared) et le R-squared ajusté (Adjusted R-squared) sont deux mesures statistiques utilisées pour évaluer la qualité de l'ajustement d'un modèle de régression.

i. **R-squared:** Il mesure la proportion de la variance de la variable dépendante qui est prédite à partir des variables indépendantes. Il varie de 0 à 1, où un R-squared de 1 indique que toutes les variations de la variable dépendante peuvent être expliquées par les variables indépendantes. Un R-squared de 0 indique que les variables indépendantes n'expliquent aucune variation de la variable dépendante.

ii. **Adjusted R-squared:** Il ajuste le R-squared en fonction du nombre de prédicteurs dans le modèle. C'est particulièrement utile dans le contexte d'une régression multiple où plusieurs variables sont incluses dans le modèle. Le R-squared ajusté tient compte du nombre de prédicteurs et ajuste le R-squared en conséquence. Cela aide à résoudre le problème du R-squared qui tend à augmenter lorsque nous ajoutons plus de variables, même si elles n'apportent pas d'informations supplémentaires utiles.

```
results = model.fit()

print('R-squared:', results.rsquared)
print('Adjusted R-squared:', results.rsquared_adj)
```

Ces deux mesures peuvent vous aider à comprendre combien de la variation de la variable dépendante votre modèle est capable d'expliquer, et peuvent vous aider à comparer différents modèles de régression.

Intervalle de confiance: statsmodels fournit déjà les intervalles de confiance pour les coefficients.

```
print('Intervalles de confiance:\n', results.conf_int())
```

L'intervalle de confiance est une fourchette de valeurs, dérivée à partir des données, dans laquelle nous avons une certaine confiance de trouver la valeur réelle d'un paramètre. Par exemple, un intervalle de confiance de 95 % pour un coefficient de régression signifie que, si nous répétions l'expérience de nombreux fois, nous nous attendrions à ce que l'intervalle contienne le vrai coefficient environ 95 % du temps. Si l'intervalle de confiance pour un coefficient ne comprend pas zéro, cela suggère que la variable associée est significative.

Tests de robustesse

Analyse des résidus: Nous pouvons créer un graphique Q-Q pour examiner si les résidus suivent une distribution normale.

```
sm.qqplot(results.resid, line='s')
plt.show()
```

Analyse d'influence: statsmodels propose plusieurs mesures d'influence, dont la distance de Cook.

```
influence = results.get_influence()
(c, p) = influence.cooks_distance
plt.stem(np.arange(len(c)), c, markerfmt=",")
plt.show()
```

Cette analyse mesure l'effet des observations individuelles sur les résultats de la régression. En d'autres termes, elle vous indique comment les estimations de vos paramètres changeraient si vous excluiez certaines observations. La distance de Cook est une mesure couramment utilisée pour cela. Les observations avec une grande distance de Cook peuvent être considérées comme des points influents. Ils peuvent potentiellement biaiser les résultats de votre régression et peuvent être susceptibles d'être des valeurs aberrantes.

Validation croisée: Pour cela, nous devons utiliser scikit-learn.

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

# Création du modèle
model = LinearRegression()

# Validation croisée
scores = cross_val_score(model, X, y, cv=5)

# Affichage des scores
print('Scores de validation croisée:', scores)
```

Il s'agit d'une technique utilisée pour évaluer la performance d'un modèle sur un sous-ensemble indépendant des données, ou pour vérifier comment le modèle d'entraînement se généralise à un ensemble de données indépendant. C'est une bonne pratique pour vérifier la robustesse du modèle et éviter le surajustement. La validation croisée divise les données en "plis" (généralement entre 5 et 10) et crée autant de modèles. Chaque modèle est formé sur tous les plis à l'exception d'un, qui est utilisé comme ensemble de test. La moyenne des erreurs sur tous les plis donne une mesure de l'erreur de généralisation du modèle.

Previsions

```
# Prédiction avec le modèle
df['pred'] = results.predict(X)

# Comparaison des valeurs prédites avec les valeurs réelles
plt.scatter(df['pred'], y)
plt.xlabel('Valeurs prédites')
plt.ylabel('Valeurs réelles')
plt.title('Valeurs prédites vs Valeurs réelles')
plt.show()
```

Autres methodes:

Si les hypothèses de la régression linéaire sont violées, plusieurs alternatives peuvent être envisagées. Le choix de la méthode dépendra de l'hypothèse spécifique qui est violée. Voici quelques options :

1. **Régression par les moindres carrés pondérés (WLS) :** Si l'hypothèse d'homoscédasticité est violée (c'est-à-dire que la variance des erreurs n'est pas constante),

alors une régression par les moindres carrés pondérés peut être utilisée. Cette méthode accorde moins de poids aux observations avec une plus grande variabilité.

2. **Régression robuste :** Les régressions robustes sont utilisées lorsque les données contiennent des outliers ou sont sujettes à des erreurs de mesure. Ces méthodes essaient de minimiser l'effet des outliers.
3. **Régression quantile :** Si les résidus ne sont pas normalement distribués ou si la relation entre les variables indépendantes et dépendantes n'est pas strictement linéaire ou homoscédastique, la régression quantile peut être une bonne option.
4. **Régression Ridge ou Lasso (méthodes de régularisation) :** Ces techniques peuvent être utiles lorsque vous avez une multicollinéarité dans vos données ou lorsque vous voulez éviter le surapprentissage dans un contexte de grande dimensionnalité (plus de variables que d'observations).
5. **Modèles de séries chronologiques :** Si vos données sont des séries chronologiques et qu'il y a une autocorrélation dans les erreurs, vous devriez utiliser des modèles de séries chronologiques comme ARIMA ou SARIMA.
6. **Régression non linéaire ou modèles de transformation :** Si la relation entre les variables indépendantes et dépendantes n'est pas linéaire, une régression non linéaire ou une transformation des variables peut être nécessaire.
7. **Modèles à effets mixtes ou à effets aléatoires :** Si vos données sont structurées en groupes (par exemple, des mesures répétées sur les mêmes sujets), ces modèles peuvent être utiles pour tenir compte de la structure de dépendance.
8. **Modèles de régression logistique ou probit :** Si la variable dépendante est binaire, vous devez utiliser un modèle de régression logistique ou probit.