Alright, let's walk through it — using statsmodels.formula.api to build a regression model is actually pretty straightforward.

Here's a **basic step-by-step**:

1. Install and import packages

```
  □ Copy

python
                                                                                  import statsmodels.formula.api as smf
import pandas as pd
```

You need your data in a Pandas DataFrame.

2. Prepare your data

Example:

python

```
data = pd.DataFrame({
    'y': [1, 2, 3, 4, 5],
    'x1': [2, 1, 4, 3, 5],
    'x2': [5, 3, 6, 2, 7]
})
```

 \downarrow

• " $y \sim x1 + x2$ " Means predict y using x1 and x2.

In formula notation:

python

python

markdown

3. Write the regression formula

4. Fit the model

model = $smf.ols(formula='y \sim x1 + x2', data=data)$ results = model.fit()

Use .ols() for ordinary least squares (linear regression):

```
5. Check the results
You can see a full summary of the regression:
```

print(results.summary())

Quick Example Output

```
OLS Regression Results
                                        R- ↓ ared:
Dep. Variable:
                                                                          0.987
                                        Adj. R-squared:
                                                                          0.981
Model:
                        Least Squares
                                                                          161.3
Method:
                                        F-statistic:
. . .
```

• For polynomial terms, you can manually add squared columns to your DataFrame, or use libraries

like patsy to automate it.

Extra tips:

• If you want an interaction between x1 and x2, you can use x1 * x2 in the formula.

Would you like me to show you a real-world examp ↓ o, like Boston housing or something? 🖋

If you only want the interaction (no main effects separately), use x1:x2.

i've got the target variable and the independent variable

python

python

Then you can do:

Suppose:

- import statsmodels.formula.api as smf
- # Build and fit the model model = $smf.ols(formula='y \sim x', data=data)$

Or maybe logistic regression instead of linear?

Here's the **simplest possible setup** for your case:

Perfect — that's all you really need!

Your target variable is called y

Your independent variable is called x

Both are in a Pandas DataFrame called data

See the output

If your variable names have spaces or weird characters, you can surround them with $\mathbb{Q}()$ like this:

results = model.fit()

print(results.summary())

```
Want me to show you how to plot it too, like line of best fit on a scatter plot? of
(It's super helpful for checking if the model makes sense.)
```

how do i asses the validity of the assumptions of linearity using this

model = smf.ols(formula='intent1_01 ~ attitude1_04 + peruse03 +

model = smf.ols(formula='Q("target var") ~ Q("input var")', data=data)

regression model?

Selection deleted

results = model.fit()

print(results.summary())

import statsmodels.formula.api as smf

peruse01 + satis02 + satis01', data=cols_to_use)

Dep. Variable: intent1_01 R-squared:

Model: OLS Adj. R-squared:

Method: Least Squares F-statistic:

Time: 17:12:50 Log-Likelihood:

Wed, 30 Apr 2025 Prob (F-statistic):

nonrobust

201 AIC:

0.481

36.12

0.467

-222.79

457.6

477.4

OLS Regression Results

Date:

Df Residuals:

Df Model:

No. Observations:

Covariance Type:

4.40e-26

```
coef std err t P>|t| [0.025 0.975]
                               Intercept 1.5116 0.365 4.138
                                                                     0.000
                                                                              0.791
                                                                                       2.232
                               attitude1_04 0.1555 0.067 2.326 0.021
                               0.287
                                             0.1597
                                                      0.076
                                                              2.101
                                                                      0.037
                                                                               0.010
                               peruse03
                               0.310
                                                      0.075
                                                              3.193
                                                                       0.002
                                                                                0.092
                                            0.2404
                               peruse01
                               0.389
                               satis02
                                                     0.094
                                                             0.733
                                                                      0.464
                                                                               -0.117
                                           0.0691
                               0.255
                                           0.1860
                                                    0.100
                                                                     0.064
                                                                              -0.011
                                                                                       0.383
                               satis01
                                                             1.866
                               Omnibus:
                                                    46.340 Durbin-Watson:
                                                                                      2.039
                               Prob(Omnibus):
                                                       0.000 Jarque-Bera (JB):
                               182.369
                                                   -0.832 Prob(JB):
                                                                               2.51e-40
                               Skew:
                                                    7.360 Cond. No.
                               Kurtosis:
                                                                                  91.3
Awesome — you're really close.
  how do I assess the validity of the linearity assumption using this regression model?
In linear regression, the linearity assumption means that the relationship between each independent
variable and the dependent variable is linear.
To check it properly after fitting your model (results), you mainly want to:
                                               \downarrow
1. Plot residuals vs fitted values (most important)
You want to make a scatter plot:
 x-axis = predicted (fitted) values
If the assumption holds, the residuals should be randomly scattered around zero (no patterns).
```

plt.xlabel('Fitted values') plt.ylabel('Residuals') plt.title('Residuals vs Fitted')

plt.axhline(0, linestyle='--', color='black')

If you see random scatter around zero, you're good.

2. Partial regression plots (bonus, deepr theck)

If you see a curve, trend, or funnel shape, it suggests non-linearity.

 \downarrow

sns.residplot(x=fitted_vals, y=residuals, lowess=True, line_kws={'color': 'red'})

Now, you asked:

y-axis = residuals (errors)

import matplotlib.pyplot as plt

fitted_vals = results.fittedvalues

import seaborn as sns

residuals = results.resid

Predicted values

Residuals

Plot

plt.show()

Here's how to plot it:

python

You can also create partial regression plots for each independent variable to see their individual linear relationships: python

import statsmodels.api as sm

fig.tight_layout(pad=1.0)

3. Look at the residual distribution

Create a figure

plt.show()

Each small plot will show one predictor's adjusted relationship with the target, controlling for others. Look for **straight lines** — curves = possible non-linearity.

fig = sm.graphics.plot_partregress_grid(results)

strictly linearity. (Still good practice!) Example: python

 \downarrow

Plotting a histogram or a Q-Q plot of residuals helps too, but that's more about checking normality, not

```
sm.qqplot(residuals, line='45')
 plt.title('Q-Q plot of residuals')
 plt.show()
Quick Reminder
```

✓ You already have <code>.summary()</code> output — that's mostly **overall model performance**, not detailed

 \downarrow

assumption checking. ▼ The Omnibus and Jarque-Bera you see check normality, not linearity. ▼ The Durbin-Watson checks autocorrelation, not linearity.

TL;DR:

Plot residuals vs fitted values and check for randomness.

If you want, I can help you also automatically flag problems (like code that tells you "non-linearity detected!" if patterns exist). Would you like that? 🚀