

# polars\_reg Showcase

## Table of contents

<b>polars_reg Showcase</b>	<b>1</b>
1. Generate sample data . . . . .	1
2. OLS — Basic, Robust, and Clustered Standard Errors . . . . .	2
2b. HAC and Driscoll-Kraay Standard Errors . . . . .	4
3. High-Dimensional Fixed Effects (reghdfe-style) . . . . .	5
4. Instrumental Variables . . . . .	6
5. Panel Estimators . . . . .	8
6. Result Object — Programmatic Access . . . . .	9
7. regtable — Side-by-Side Regression Comparison (estout-style) . . . . .	11
8. GroupBy Regression — Run per Group . . . . .	13
9. Stata Equivalence — Generate Stata Code . . . . .	14
10. R Equivalence — Generate R Code . . . . .	15
11. Formula Syntax Reference . . . . .	17

## polars\_reg Showcase

A tour of everything polars\_reg can do: OLS, fixed effects, instrumental variables, panel estimators, robust/clustered standard errors, pretty-printed tables, and Stata equivalence checking.

```
pip install polars_reg
```

```
import numpy as np
import polars as pl
import polars_reg as pr

# Reproducible data
rng = np.random.default_rng(42)
```

### 1. Generate sample data

A simulated panel dataset with firm and industry fixed effects, an endogenous variable, and instruments.

```
n_firms, n_years = 100, 10
n = n_firms * n_years

firm_id = np.repeat(np.arange(n_firms), n_years)
year_id = np.tile(np.arange(2010, 2010 + n_years), n_firms)
industry = np.repeat(rng.choice(["Tech", "Finance", "Health", "Energy"], n_firms), n_years)
```

```

# Fixed effects
firm_fe = rng.standard_normal(n_firms)
year_fe = rng.standard_normal(n_years) * 0.5

# Regressors
x1 = rng.standard_normal(n)
x2 = rng.standard_normal(n)

# Endogenous variable + instruments
z1 = rng.standard_normal(n)
z2 = rng.standard_normal(n)
u = rng.standard_normal(n) # correlated error
x_endog = 0.5 * z1 + 0.3 * z2 + 0.8 * u

# Outcome
y = 2.0 + 1.0 * x1 - 0.5 * x2 + 1.5 * x_endog + firm_fe[firm_id] + year_fe[year_id % n_years] + u

df = pl.DataFrame({
    "y": y, "x1": x1, "x2": x2,
    "x_endog": x_endog, "z1": z1, "z2": z2,
    "firm_id": firm_id, "year": year_id, "industry": industry,
})
df.head()

```

y	x1	x2	x_endog	z1	z2	firm_id	year	industry
f64	f64	f64	f64	f64	f64	i64	i64	str
6.105845	-0.499296	-2.450872	0.743585	0.677485	-1.456985	0	2010	"Tech"
2.412468	-1.184944	-1.417684	-0.216587	-0.743861	-0.698221	0	2011	"Tech"
-1.115963	-0.965117	-1.18707	-1.341555	-0.521539	-0.051568	0	2012	"Tech"
-1.765926	-0.725226	-0.363261	-1.370323	-0.943954	0.546667	0	2013	"Tech"
6.496028	2.12847	-0.254608	1.24362	1.301158	0.554797	0	2014	"Tech"

## 2. OLS — Basic, Robust, and Clustered Standard Errors

```

# Basic OLS with iid standard errors
r_iid = pr.ols("y ~ x1 + x2 + x_endog", data=df)
print(r_iid.summary())

```

```

=====
      OLS Regression Results
=====
Dep. Variable:          y                R-squared:          0.8243
No. Observations:    1000              Adj. R-squared:    0.8238
Df Residuals:         996                SE type:          iid
=====

```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9862	0.0383	25.76	<0.01	[ 0.9111, 1.0613]
x2	-0.5080	0.0375	-13.53	<0.01	[ -0.5817, -0.4343]
x_endog	2.3318	0.0376	61.95	<0.01	[ 2.2579, 2.4057]
_cons	1.8787	0.0383	49.11	<0.01	[ 1.8036, 1.9538]

```

=====

```

```
# Robust standard errors (HC1, HC2, HC3)
r_robust = pr.ols("y ~ x1 + x2 + x_endog", data=df, vcov="HC1")
print(r_robust.summary())
```

```
=====
OLS Regression Results
=====
```

Dep. Variable:	y	R-squared:	0.8243
No. Observations:	1000	Adj. R-squared:	0.8238
Df Residuals:	996	SE type:	HC1

```
=====
```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9862	0.0421	23.42	<0.01	[ 0.9036, 1.0689]
x2	-0.5080	0.0379	-13.41	<0.01	[ -0.5824, -0.4337]
x_endog	2.3318	0.0375	62.18	<0.01	[ 2.2582, 2.4054]
_cons	1.8787	0.0380	49.46	<0.01	[ 1.8042, 1.9533]

```
=====
```

```
# Clustered standard errors (one-way)
r_cl1 = pr.ols("y ~ x1 + x2 + x_endog", data=df, cluster="firm_id")
print(r_cl1.summary())
```

```
=====
OLS Regression Results
=====
```

Dep. Variable:	y	R-squared:	0.8243
No. Observations:	1000	Adj. R-squared:	0.8238
Df Residuals:	99	SE type:	cluster
Clusters:	firm_id: 100		

```
=====
```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9862	0.0388	25.43	<0.01	[ 0.9093, 1.0632]
x2	-0.5080	0.0356	-14.27	<0.01	[ -0.5787, -0.4374]
x_endog	2.3318	0.0379	61.51	<0.01	[ 2.2566, 2.4070]
_cons	1.8787	0.0936	20.08	<0.01	[ 1.6930, 2.0644]

```
=====
```

```
# Multi-way clustered standard errors (Cameron-Gelbach-Miller)
r_cl2 = pr.ols("y ~ x1 + x2 + x_endog", data=df, cluster=["firm_id", "year"])
print(r_cl2.summary())
```

```
=====
OLS Regression Results
=====
```

Dep. Variable:	y	R-squared:	0.8243
No. Observations:	1000	Adj. R-squared:	0.8238
Df Residuals:	9	SE type:	cluster
Clusters:	firm_id: 100, year: 10		

```
=====
```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9862	0.0332	29.70	<0.01	[ 0.9111, 1.0613]
x2	-0.5080	0.0315	-16.12	<0.01	[ -0.5793, -0.4367]

```

x_endog      2.3318    0.0178   131.06   <0.01   [  2.2915,    2.3720]
_cons        1.8787    0.1903    9.87    <0.01   [  1.4483,    2.3091]

```

## 2b. HAC and Driscoll-Kraay Standard Errors

Newey-West (HAC) for autocorrelation-robust SEs, and Driscoll-Kraay for panel data with cross-sectional dependence.

```

# Newey-West HAC standard errors (robust to autocorrelation)
r_nw = pr.ols("y ~ x1 + x2 + x_endog", data=df, vcov="NW", time="year")
print(r_nw.summary())

```

```

=====
OLS Regression Results
=====
Dep. Variable:          y                R-squared:          0.8243
No. Observations:      1000            Adj. R-squared:     0.8238
Df Residuals:          996              SE type:           NW
=====

```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9862	0.0397	24.85	<0.01	[ 0.9083, 1.0641]
x2	-0.5080	0.0363	-14.01	<0.01	[ -0.5792, -0.4368]
x_endog	2.3318	0.0353	66.04	<0.01	[ 2.2625, 2.4011]
_cons	1.8787	0.0538	34.90	<0.01	[ 1.7731, 1.9843]

```

=====

```

```

# Driscoll-Kraay SEs: robust to cross-sectional dependence + autocorrelation
# Works with absorbed FE – ideal for macro panels with common shocks
r_dk = pr.ols(
    "y ~ x1 + x2 + x_endog | firm_id",
    data=df,
    vcov="DK",
    time="year",
)
print(r_dk.summary())

```

```

=====
OLS Regression Results
=====
Dep. Variable:          y                R-squared:          0.9127
No. Observations:      1000            Adj. R-squared:     0.9027
Df Residuals:          897              SE type:           DK
Absorbed FE:           firm_id (100 DoF)
=====

```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9937	0.0210	47.41	<0.01	[ 0.9525, 1.0348]
x2	-0.4707	0.0128	-36.76	<0.01	[ -0.4958, -0.4455]
x_endog	2.3105	0.0156	148.00	<0.01	[ 2.2798, 2.3411]

```

=====

```

```
# Panel FE with Driscoll-Kraay SEs
```

```
r_pfe_dk = pr.panel_fe(
    "y ~ x1 + x2 + x_endog",
    data=df,
    entity="firm_id",
    time="year",
    vcov="DK",
    bandwidth=3,
)
print(r_pfe_dk.summary())
```

Panel FE Regression Results						
Dep. Variable:	y		R-squared:	0.9507		
No. Observations:	1000		Adj. R-squared:	0.9445		
Df Residuals:	888		SE type:	DK		
Absorbed FE:	firm_id, year (109 DoF)					
	Coef	Std.Err.	t	P> t	[95% Conf. Interval]	
x1	1.0021	0.0162	61.96	<0.01	[ 0.9704,	1.0339]
x2	-0.4837	0.0122	-39.61	<0.01	[ -0.5076,	-0.4597]
x_endog	2.3005	0.0165	139.64	<0.01	[ 2.2681,	2.3328]

### 3. High-Dimensional Fixed Effects (reghdfe-style)

Absorb firm and/or year fixed effects via iterative demeaning. The | separator in the formula specifies which variables to absorb.

```
# One-way FE: absorb firm
```

```
r_fel = pr.ols("y ~ x1 + x2 + x_endog | firm_id", data=df, cluster="firm_id")
print(r_fel.summary())
```

OLS Regression Results						
Dep. Variable:	y			R-squared:	0.9127	
No. Observations:	1000			Adj. R-squared:	0.9027	
Df Residuals:	99			SE type:	cluster	
Absorbed FE:	firm_id (100 DoF)					
Clusters:	firm_id: 100					
	Coef	Std.Err.	t	P> t	[95% Conf. Interval]	
x1	0.9937	0.0271	36.64	<0.01	[ 0.9398,	1.0475]
x2	-0.4707	0.0230	-20.45	<0.01	[ -0.5163,	-0.4250]
x_endog	2.3105	0.0279	82.82	<0.01	[ 2.2551,	2.3658]

```
# Two-way FE: absorb firm + year, two-way clustered SEs
```

```
r_fe2 = pr.ols("y ~ x1 + x2 + x_endog | firm_id + year", data=df, cluster=["firm_id", "year"])
print(r_fe2.summary())
```

```
=====
OLS Regression Results
=====
Dep. Variable:          y                R-squared:          0.9507
No. Observations:      1000             Adj. R-squared:    0.9445
Df Residuals:          9               SE type:          cluster
Absorbed FE:           firm_id, year (109 DoF)
Clusters:              firm_id: 100, year: 10
=====
```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	1.0021	0.0165	60.85	<0.01	[ 0.9649, 1.0394]
x2	-0.4837	0.0161	-30.03	<0.01	[ -0.5201, -0.4472]
x_endog	2.3005	0.0152	151.55	<0.01	[ 2.2661, 2.3348]

```
=====
```

```
# Three-way FE: firm + year + industry
r_fe3 = pr.ols("y ~ x1 + x2 | firm_id + year + industry", data=df, cluster="firm_id")
print(r_fe3.summary())
```

```
=====
OLS Regression Results
=====
Dep. Variable:          y                R-squared:          0.1618
No. Observations:      1000             Adj. R-squared:    0.0581
Df Residuals:          99              SE type:          cluster
Absorbed FE:           firm_id, year, industry (109 DoF)
Clusters:              firm_id: 100
=====
```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9234	0.0737	12.53	<0.01	[ 0.7771, 1.0696]
x2	-0.5049	0.0798	-6.33	<0.01	[ -0.6632, -0.3466]

```
=====
```

## 4. Instrumental Variables

Use || to separate the IV equation:  $y \sim \text{exog\_vars} \parallel \text{endog\_var} \sim \text{instruments}$

With fixed effects:  $y \sim \text{exog\_vars} \mid \text{fe\_vars} \mid \text{endog\_var} \sim \text{instruments}$

```
# 2SLS: instrument x_endog with z1 and z2
r_2sls = pr.iv2sls("y ~ x1 + x2 || x_endog ~ z1 + z2", data=df)
print(r_2sls.summary())
```

```
=====
2SLS Regression Results
=====
Dep. Variable:          y                R-squared:          0.7364
No. Observations:      1000             Adj. R-squared:    0.7356
Df Residuals:          996             SE type:          iid
First-stage F:          301.22
=====
```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
--	------	----------	---	------	----------------------

```
-----
```

```

x1          0.9747    0.0469    20.78    <0.01    [  0.8827,    1.0667]
x2         -0.5207    0.0460   -11.32    <0.01    [ -0.6109,   -0.4304]
_cons       1.8684    0.0469    39.87    <0.01    [  1.7765,    1.9604]
x_endog     1.4916    0.0751    19.87    <0.01    [  1.3442,    1.6389]
=====

```

```

# 2SLS with absorbed firm FE
r_2sls_fe = pr.iv2sls("y ~ x1 + x2 | firm_id | x_endog ~ z1 + z2", data=df)
print(r_2sls_fe.summary())

```

```

=====
2SLS Regression Results
=====
Dep. Variable:          y                R-squared:          0.8289
No. Observations:      1000             Adj. R-squared:    0.8094
Df Residuals:          897              SE type:          iid
Absorbed FE:           firm_id (100 DoF)
First-stage F:          302.57
=====

```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9676	0.0372	26.00	<0.01	[ 0.8946, 1.0407]
x2	-0.4772	0.0364	-13.13	<0.01	[ -0.5486, -0.4059]
x_endog	1.5441	0.0595	25.96	<0.01	[ 1.4274, 1.6609]

```

=====

```

```

# LIML – more robust to weak instruments than 2SLS
r_liml = pr.liml("y ~ x1 + x2 || x_endog ~ z1 + z2", data=df)
print(r_liml.summary())

```

```

=====
LIML Regression Results
=====
Dep. Variable:          y                R-squared:          0.7357
No. Observations:      1000             Adj. R-squared:    0.7349
Df Residuals:          996              SE type:          iid
=====

```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9747	0.0470	20.76	<0.01	[ 0.8825, 1.0668]
x2	-0.5207	0.0461	-11.31	<0.01	[ -0.6111, -0.4303]
_cons	1.8684	0.0469	39.82	<0.01	[ 1.7763, 1.9605]
x_endog	1.4883	0.0753	19.77	<0.01	[ 1.3406, 1.6359]

```

=====

```

```

# GMM-IV – efficient two-step GMM with Hansen J overidentification test
r_gmm = pr.gmm_iv("y ~ x1 + x2 || x_endog ~ z1 + z2", data=df)
print(r_gmm.summary())

```

```

=====
GMM Regression Results
=====
Dep. Variable:          y                R-squared:          0.7387
No. Observations:      1000             Adj. R-squared:    0.7379
Df Residuals:          996              SE type:          robust

```

Hansen J:	1.3978 (p = 0.2371)					
	Coef	Std.Err.	t	P> t	[95% Conf. Interval]	
x1	0.9751	0.0496	19.66	<0.01	[ 0.8778,	1.0724]
x2	-0.5203	0.0456	-11.41	<0.01	[ -0.6098,	-0.4309]
_cons	1.8701	0.0461	40.57	<0.01	[ 1.7797,	1.9606]
x_endog	1.5023	0.0747	20.12	<0.01	[ 1.3558,	1.6488]

```
# Weak instrument diagnostics
# Staiger-Stock rule of thumb (F > 10) + Stock-Yogo critical values
wit = pr.weak_instrument_test(r_2sls, n_instruments=2)
print(f"First-stage F: {wit['f_stat']:.2f}")
print(f"Staiger-Stock (F > 10): {'Pass' if wit['staiger_stock'] else 'Fail'}")
if wit['stock_yogo']:
    cv = wit['stock_yogo']['critical_values']
    print(f"Stock-Yogo critical values (maximal relative bias):")
    for pct, val in sorted(cv.items()):
        reject = wit['stock_yogo'][f'reject_{pct}pct']
        print(f" {pct}%: {val:.2f} {'(reject)' if reject else '(fail to reject)'}")
```

First-stage F: 301.22

Staiger-Stock (F > 10): Pass

Stock-Yogo critical values (maximal relative bias):

5%: 19.93 (reject)

10%: 11.59 (reject)

20%: 7.54 (reject)

30%: 5.96 (reject)

## 5. Panel Estimators

Fixed effects (within), random effects (Swamy-Arora GLS), and first-difference.

```
# Panel fixed effects (within estimator, SEs clustered by entity)
r_pfe = pr.panel_fe("y ~ x1 + x2 + x_endog", data=df, entity="firm_id", time="year")
print(r_pfe.summary())
```

Panel FE Regression Results						
=====						
Dep. Variable:	y			R-squared:	0.9507	
No. Observations:	1000			Adj. R-squared:	0.9445	
Df Residuals:	99			SE type:	cluster	
Absorbed FE:	firm_id, year (109 DoF)					
Clusters:	firm_id: 100					
=====						
	Coef	Std.Err.	t	P> t	[95% Conf. Interval]	
-----						
x1	1.0021	0.0200	50.03	<0.01	[ 0.9624,	1.0418]
x2	-0.4837	0.0165	-29.27	<0.01	[ -0.5164,	-0.4509]
x_endog	2.3005	0.0205	112.07	<0.01	[ 2.2597,	2.3412]



```
# Panel random effects (GLS with Swamy-Arora variance components)
r_pre = pr.panel_re("y ~ x1 + x2 + x_endog", data=df, entity="firm_id")
print(r_pre.summary())
```

```
=====
Panel RE Regression Results
=====
```

Dep. Variable:	y	R-squared:	0.8241
No. Observations:	1000	Adj. R-squared:	0.8236
Df Residuals:	996	SE type:	iid

```
=====
```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	0.9931	0.0265	37.54	<0.01	[ 0.9412, 1.0450]
x2	-0.4735	0.0259	-18.30	<0.01	[ -0.5243, -0.4228]
x_endog	2.3121	0.0260	88.88	<0.01	[ 2.2611, 2.3632]
_cons	1.8815	0.0948	19.85	<0.01	[ 1.6955, 2.0676]

```
=====
```

```
# Panel first-difference
r_pfd = pr.panel_fd("y ~ x1 + x2 + x_endog", data=df, entity="firm_id", time="year")
print(r_pfd.summary())
```

```
=====
Panel FD Regression Results
=====
```

Dep. Variable:	y	R-squared:	0.9282
No. Observations:	900	Adj. R-squared:	0.9279
Df Residuals:	99	SE type:	cluster
Clusters:	firm_id: 100		

```
=====
```

	Coef	Std.Err.	t	P> t	[95% Conf. Interval]
x1	1.0088	0.0289	34.85	<0.01	[ 0.9514, 1.0662]
x2	-0.4798	0.0262	-18.33	<0.01	[ -0.5317, -0.4278]
x_endog	2.3033	0.0285	80.80	<0.01	[ 2.2467, 2.3598]
_cons	-0.0087	0.0090	-0.97	0.336	[ -0.0266, 0.0092]

```
=====
```

```
# Hausman test: FE vs RE – test whether RE is consistent
hausman = pr.hausman_test(r_pfe, r_pre)
print(f"Hausman test: chi2={hausman['statistic']:.2f}, "
      f"p={hausman['pvalue']:.4f}, df={hausman['df']}")
print(f"Coefficients compared: {hausman['coefficients_compared']}")
```

```
Hausman test: chi2=0.00, p=1.0000, df=3
Coefficients compared: ['x1', 'x2', 'x_endog']
```

## 6. Result Object — Programmatic Access

Every regression returns a `RegressionResult` with coefficients, standard errors, t-stats, p-values, confidence intervals, coefficient tables, fitted values, and predictions.

```

r = pr.ols("y ~ x1 + x2", data=df)

print("Coefficients:", r.coef)
print("Std errors: ", r.se)
print("t-stats:      ", r.tstat)
print("p-values:     ", r.pvalue)
print("95% CI:\n", r.confint())
print("\nR²:", r.r_squared, " Adj R²:", r.r_squared_adj, " N:", r.n_obs)

```

```

Coefficients: [ 0.95428992 -0.54312729  1.85016554]
Std errors:   [0.08428604 0.08264979 0.08423072]
t-stats:      [11.32203953 -6.57142988 21.96544941]
p-values:      [4.83693656e-28 8.01761476e-11 1.56643938e-87]
95% CI:
[[ 0.78889152  1.11968832]
 [-0.70531479 -0.38093979]
 [ 1.68487571  2.01545537]]

```

```

R²: 0.1474211973818752 Adj R²: 0.14571090891122696 N: 1000

```

```

# Coefficient table as a Polars DataFrame – easy to filter, export, etc.
r.coef_table()

```

name	coef	se	t	p	ci_lower	ci_upper
str	f64	f64	f64	f64	f64	f64
"x1"	0.95429	0.084286	11.32204	4.8369e-28	0.788892	1.119688
"x2"	-0.543127	0.08265	-6.57143	8.0176e-11	-0.705315	-0.38094
"_cons"	1.850166	0.084231	21.965449	1.5664e-87	1.684876	2.015455

```

# Fitted values and prediction
y_hat = r.fitted()
print("In-sample fitted values (first 5):", y_hat[:5])

# Out-of-sample prediction with new data
new_df = pl.DataFrame({"x1": [1.0, 2.0], "x2": [0.5, -0.5]})
print("Predictions:", r.predict(new_df))

```

```

In-sample fitted values (first 5): [2.70482832 1.48936861 1.57389469 1.35538639 4.01962739]
Predictions: [2.53289182 4.03030902]

```

```

# Wald test: test that beta_x1 = 0 (should reject)
wald = r.wald_test(np.array([[1, 0, 0]]))
print(f"Wald test H0: beta_x1 = 0 -> "
      f"F={wald['statistic']:.2f}, p={wald['pvalue']:.4f}")

# Wald test: test joint significance of x1 and x2
wald_joint = r.wald_test(np.array([[1, 0, 0], [0, 1, 0]]))
print(f"Wald test H0: beta_x1 = beta_x2 = 0 -> "
      f"F={wald_joint['statistic']:.2f}, p={wald_joint['pvalue']:.4f}")

```

```

Wald test H0: beta_x1 = 0 -> F=128.19, p=0.0000
Wald test H0: beta_x1 = beta_x2 = 0 -> F=86.20, p=0.0000

```

## 7. regtable — Side-by-Side Regression Comparison (estout-style)

Compare multiple specifications in a single compact table. Each FE and cluster variable gets its own Y/N indicator row.

```
# Build up a specification progressively
m1 = pr.ols("y ~ x1 + x2 + x_endog", data=df)
m2 = pr.ols("y ~ x1 + x2 + x_endog", data=df, vcov="HC1")
m3 = pr.ols("y ~ x1 + x2 + x_endog | firm_id", data=df, cluster="firm_id")
m4 = pr.ols(
    "y ~ x1 + x2 + x_endog | firm_id + year",
    data=df,
    cluster=["firm_id", "year"],
)

print(pr.regtable(m1, m2, m3, m4, labels=["OLS", "Robust", "Firm FE", "Towway FE"]))
```

	OLS	Robust	Firm FE	Towway FE
	OLS	OLS	OLS	OLS
x1	0.9862*** (0.03828)	0.9862*** (0.04212)	0.9937*** (0.02712)	1.002*** (0.01647)
x2	-0.508*** (0.03754)	-0.508*** (0.03788)	-0.4707*** (0.02301)	-0.4837*** (0.0161)
x_endog	2.332*** (0.03764)	2.332*** (0.0375)	2.31*** (0.0279)	2.3*** (0.01518)
_cons	1.879*** (0.03826)	1.879*** (0.03798)		
Fixed Effects				
firm_id	N	N	Y	Y
year	N	N	N	Y
Clustering				
firm_id	N	N	Y	Y
year	N	N	N	Y
N	1000	1000	1000	1000
R <sup>2</sup>	0.8243	0.8243	0.9127	0.9507
Adj. R <sup>2</sup>	0.8238	0.8238	0.9027	0.9445

\* p<0.10, \*\* p<0.05, \*\*\* p<0.01

```
# Compare IV estimators side by side
print(pr.regtable(r_2sls, r_liml, r_gmm, labels=["2SLS", "LIML", "GMM"]))
```

	2SLS	LIML	GMM
	2SLS	LIML	GMM
x1	0.9747*** (0.0469)	0.9747*** (0.04696)	0.9751*** (0.04959)
x2	-0.5207*** (0.04599)	-0.5207*** (0.04605)	-0.5203*** (0.04559)
_cons	1.868*** (0.04686)	1.868*** (0.04693)	1.87*** (0.0461)

```

x_endog          1.492***      1.488***      1.502***
                (0.07508)      (0.07527)      (0.07466)
-----
N                  1000          1000          1000
R²                 0.7364         0.7357         0.7387
Adj. R²            0.7356         0.7349         0.7379
=====
* p<0.10, ** p<0.05, *** p<0.01

```

```

# Compare panel estimators
print(pr.regtbl(r_pfe, r_pre, r_pfd, labels=["Panel FE", "Panel RE", "Panel FD"]))

```

```

=====
                Panel FE      Panel RE      Panel FD
                Panel FE      Panel RE      Panel FD
-----
x1              1.002***      0.9931***      1.009***
                (0.02003)      (0.02645)      (0.02894)
x2             -0.4837***      -0.4735***      -0.4798***
                (0.01652)      (0.02587)      (0.02617)
x_endog         2.3***        2.312***        2.303***
                (0.02053)      (0.02601)      (0.0285)
_cons           1.882***      -0.008707
                (0.0948)      (0.009008)
-----
Fixed Effects
  firm_id          Y          N          N
  year            Y          N          N
Clustering
  firm_id          Y          N          Y
-----
N                  1000          1000          900
R²                 0.9507         0.8241         0.9282
Adj. R²            0.9445         0.8236         0.9279
=====
* p<0.10, ** p<0.05, *** p<0.01

```

```

# Customize: no stars, higher precision
print(pr.regtbl(m1, m3, stars=False, precision=6))

```

```

=====
                (1)          (2)
                OLS          OLS
-----
x1              0.986214      0.993656
                (0.0382822)    (0.0271224)
x2             -0.508013      -0.470659
                (0.0375399)    (0.0230137)
x_endog         2.33179       2.31047
                (0.0376397)    (0.0278959)
_cons           1.87872
                (0.0382563)
-----
Fixed Effects
  firm_id          N          Y

```

```

Clustering
  firm_id          N          Y
-----
N          1000      1000
R2        0.8243      0.9127
Adj. R2    0.8238      0.9027
=====

```

## 8. GroupBy Regression — Run per Group

`groupby_reg()` runs the same regression for each group in the data — useful for estimating factor loadings per stock, running regressions per industry, etc.

```

# Run OLS per industry group
grp = pr.groupby_reg(pr.ols, "y ~ x1 + x2 + x_endog", df, group_by="industry")
print(grp.summary())

```

GroupBy Regression: 4 groups succeeded

```

--- Group: Tech (N=220) ---
  Coefs: x1=1.0385, x2=-0.4486, x_endog=2.3278, _cons=1.9915
  R2=0.8827

--- Group: Energy (N=280) ---
  Coefs: x1=0.8939, x2=-0.5258, x_endog=2.3360, _cons=1.9813
  R2=0.7670

--- Group: Health (N=240) ---
  Coefs: x1=1.0364, x2=-0.5471, x_endog=2.4097, _cons=1.6810
  R2=0.8461

--- Group: Finance (N=260) ---
  Coefs: x1=0.9920, x2=-0.4957, x_endog=2.2685, _cons=1.8508
  R2=0.8192

```

```

# Stacked coefficient table across all groups
grp.coef_table()

```

group str	name str	coef f64	se f64	t f64	p f64	ci_lower f64	ci_upper f64
"Tech"	"x1"	1.038527	0.069704	14.899151	5.2881e-35	0.901141	1.175914
"Tech"	"x2"	-0.448646	0.067057	-6.690551	1.8797e-10	-0.580816	-0.316477
"Tech"	"x_endog"	2.327811	0.064988	35.818861	7.9869e-93	2.199719	2.455904
"Tech"	"_cons"	1.991462	0.069086	28.825915	5.6705e-76	1.855293	2.12763
"Energy"	"x1"	0.893921	0.080314	11.130391	5.1340e-24	0.735816	1.052026
...	...	...	...	...	...	...	...
"Health"	"_cons"	1.680963	0.075295	22.324897	4.1863e-60	1.532626	1.8293
"Finance"	"x1"	0.992039	0.080879	12.265783	1.6303e-27	0.832767	1.151311
"Finance"	"x2"	-0.495732	0.077845	-6.368233	8.8052e-10	-0.649029	-0.342435
"Finance"	"x_endog"	2.268491	0.072877	31.12774	5.2973e-89	2.124976	2.412005
"Finance"	"_cons"	1.850838	0.07461	24.80698	4.7860e-70	1.703912	1.997765

```
print(pr.regtable(grp))
```

```
=====
                    Tech          Energy          Health          Finance
                    OLS           OLS           OLS           OLS
-----
x1                  1.039***      0.8939***      1.036***      0.992***
                  (0.0697)      (0.08031)     (0.07144)     (0.08088)
x2                 -0.4486***     -0.5258***     -0.5471***     -0.4957***
                  (0.06706)      (0.07835)     (0.07259)     (0.07784)
x_endog             2.328***      2.336***      2.41***        2.268***
                  (0.06499)      (0.08509)     (0.07378)     (0.07288)
_cons              1.991***      1.981***      1.681***      1.851***
                  (0.06909)      (0.08227)     (0.0753)      (0.07461)
-----
N                   220           280           240           260
R²                  0.8827        0.7670        0.8461        0.8192
Adj. R²             0.8811        0.7645        0.8442        0.8171
=====
* p<0.10, ** p<0.05, *** p<0.01
```

## 9. Stata Equivalence — Generate Stata Code

`to_stata()` translates any `polars_reg` call into the equivalent Stata command, so you can verify results in Stata or share reproducible code with Stata users.

```
# OLS → reg
print(pr.to_stata("ols", "y ~ x1 + x2"))
print()

# OLS with robust SEs → reg, vce(robust)
print(pr.to_stata("ols", "y ~ x1 + x2", vcov="HC1"))
print()

# reghdfe with multi-way FE and clustering
print(pr.to_stata("ols", "y ~ x1 + x2 | firm_id + year", cluster=["firm_id", "year"]))
print()

# 2SLS → ivregress 2sls
print(pr.to_stata("iv2sls", "y ~ x1 + x2 || x_endog ~ z1 + z2"))
print()

# 2SLS with FE → ivreghdfe
print(pr.to_stata("iv2sls", "y ~ x1 + x2 | firm_id | x_endog ~ z1 + z2"))
print()

# LIML
print(pr.to_stata("liml", "y ~ x1 + x2 || x_endog ~ z1 + z2"))
print()

# GMM
print(pr.to_stata("gmm_iv", "y ~ x1 + x2 || x_endog ~ z1 + z2"))
print()

# Panel FE → xtreg, fe
```

```

print(pr.to_stata("panel_fe", "y ~ x1 + x2", entity="firm_id", time="year"))
print()

# Panel RE → xtreg, re
print(pr.to_stata("panel_re", "y ~ x1 + x2", entity="firm_id"))
print()

# Panel FD → reg D.y D.x1 D.x2
print(pr.to_stata("panel_fd", "y ~ x1 + x2", entity="firm_id", time="year"))

```

```

reg y x1 x2

reg y x1 x2, vce(robust)

reghdfe y x1 x2, absorb(firm_id year) vce(cluster firm_id year)

ivregress 2sls y x1 x2 (x_endog = z1 z2), small

ivreghdfe y x1 x2 (x_endog = z1 z2), absorb(firm_id)

ivregress liml y x1 x2 (x_endog = z1 z2), small

ivregress gmm y x1 x2 (x_endog = z1 z2), wmatrix(robust)

xtset firm_id year
xtreg y x1 x2, fe

xtset firm_id
xtreg y x1 x2, re

xtset firm_id year
reg D.y D.x1 D.x2

```

```

# Generate pystata-compatible Python code for automated comparison
print(pr.to_stata("ols", "y ~ x1 + x2 | firm_id", cluster=["firm_id"], pystata=True))

```

```

import stata_setup
stata_setup.config('/path/to/stata', 'mp') # adjust path and edition
from pystata import stata

# Load your data first:
# stata.pdataframe_to_data(df.to_pandas(), force=True)

# Equivalent to polars_reg.ols():
stata.run("reghdfe y x1 x2, absorb(firm_id) vce(cluster firm_id)")

# Extract results:
stata.run("matrix list e(b)")
stata.run("matrix list e(V)")

```

## 10. R Equivalence — Generate R Code

`to_r()` translates any `polars_reg` call into equivalent R code, mapping to `lm()`, `fixest::feols()`, `AER::ivreg()`, or `plm::plm()` as appropriate.

```

# OLS → lm()
print(pr.to_r("ols", "y ~ x1 + x2"))
print()

# OLS with robust SEs → sandwich::vcovHC
print(pr.to_r("ols", "y ~ x1 + x2", vcov="HC1"))
print()

# OLS with clustering → fixest::feols
print(pr.to_r("ols", "y ~ x1 + x2", cluster=["firm_id"]))
print()

# reghdfe-style FE + clustering → fixest::feols
print(pr.to_r("ols", "y ~ x1 + x2 | firm_id + year", cluster=["firm_id", "year"]))
print()

# 2SLS → fixest::feols with IV syntax
print(pr.to_r("iv2sls", "y ~ x1 + x2 || x_endog ~ z1 + z2"))
print()

# 2SLS with FE → fixest::feols
print(pr.to_r("iv2sls", "y ~ x1 + x2 | firm_id | x_endog ~ z1 + z2"))
print()

# LIML → AER::ivreg
print(pr.to_r("liml", "y ~ x1 + x2 || x_endog ~ z1 + z2"))
print()

# GMM → advisory note + feols fallback
print(pr.to_r("gmm_iv", "y ~ x1 + x2 || x_endog ~ z1 + z2"))
print()

# Panel FE → plm with model="within"
print(pr.to_r("panel_fe", "y ~ x1 + x2", entity="firm_id", time="year"))
print()

# Panel RE → plm with model="random"
print(pr.to_r("panel_re", "y ~ x1 + x2", entity="firm_id"))
print()

# Panel FD → plm with model="fd"
print(pr.to_r("panel_fd", "y ~ x1 + x2", entity="firm_id", time="year"))

```

```

model <- lm(y ~ x1 + x2, data=df)
summary(model)

```

```

library(sandwich)
library(lmtest)

```

```

model <- lm(y ~ x1 + x2, data=df)
coeftest(model, vcov=vcovHC(model, type="HC1"))

```

```

library(fixest)

```

```

model <- feols(y ~ x1 + x2, data=df, vcov=~firm_id)
summary(model)

```



```

library(fixest)

model <- feols(y ~ x1 + x2 | firm_id + year, data=df, vcov=~firm_id + year)
summary(model)

library(fixest)

model <- feols(y ~ x1 + x2 | x_endog ~ z1 + z2, data=df, vcov="iid")
summary(model)

library(fixest)

model <- feols(y ~ x1 + x2 | firm_id | x_endog ~ z1 + z2, data=df, vcov="iid")
summary(model)

library(AER)

model <- ivreg(y ~ x1 + x2 + x_endog | x1 + x2 + z1 + z2, data=df, model="liml")
summary(model)

# No direct single-function R equivalent for two-step efficient GMM.
# Options:
# 1. gmm::gmm() with custom moment conditions
# 2. fixest::feols() for 2SLS (not identical to GMM)
#
# For approximate comparison, 2SLS with robust SEs:

library(fixest)
model <- feols(y ~ x1 + x2 | x_endog ~ z1 + z2, data=df, vcov="HC1")
summary(model)

library(plm)

model <- plm(y ~ x1 + x2, data=df, model="within", index=c("firm_id", "year"))
summary(model)

library(plm)

model <- plm(y ~ x1 + x2, data=df, model="random", index=c("firm_id"))
summary(model)

library(plm)

model <- plm(y ~ x1 + x2, data=df, model="fd", index=c("firm_id", "year"))
summary(model)

```

## 11. Formula Syntax Reference

polars_reg	Stata	R
$y \sim x1 + x2$	reg y x1 x2	lm(y ~ x1 + x2)
$y \sim x1 + x2 - 1$	reg y x1 x2, noconstant	lm(y ~ x1 + x2 - 1)
$y \sim x1 \mid fe1$	reghdfe y x1, absorb(fe1)	feols(y ~ x1   fe1)
$y \sim x1 \mid fe1 + fe2$	reghdfe y x1, absorb(fe1 fe2)	feols(y ~ x1   fe1 + fe2)
$y \sim x1 \parallel x\_end \sim z1 + z2$	ivregress 2sls y x1 (x_end = z1 z2)	feols(y ~ x1   x_end ~ z1 + z2)

```
y ~ x1 | fe1 | x_end ~ z1          ivreghdfe y x1 (x_end=z1), absorb(fe1)  feols(y ~ x1 | fe1 | x_end ~ z1)
```

**R packages:** `fixest::feols()` for FE/clustering, `AER::ivreg()` for IV, `base lm()` for OLS.