Resources

Cheat Sheet
Pandas/numpy: Python / Pandas / Numpy
Dataset: Dataset Interview Cheat Sheet- Python/Pandas Regression/ML
Optiver Kaggle (LGBM):
Strategy colab: mie_strategies_backtest.ipynb

Kaggle Examples
Stock EDA:
https://www.kaggle.com/code/faressayah/stock-market-analysis-prediction-using-lstm#2.-What-was-the-moving-average-of-the-various-stocks?
Stock correlation K-means:
https://www.kaggle.com/code/andrealunch/stock-correlation/notebook
LSTM/CatBoost:
https://www.kaggle.com/code/yokoinaba/catboost-gru-lstm-predictive-001-submission#%E7%89%B9%E5%BE%B4%E9%87%8F%E5%88%86%E6%9E%90

Correlation

Pearson correlation
"It is covariance divided by the product of standard deviations. It is between -1 and 1."

Covariance: $\mathrm{cov}(x, y) = \mathbb{E}[(x - \mu_x)(y - \mu_y)]$

Pearson correlation:

$$\rho_{xy} = \frac{\mathrm{cov}(x, y)}{\sigma_x \sigma_y}$$

```
corr_xz = df["x"].corr(df["z"]) #default
print("Pearson corr(x,z) =", corr_xz)
C = df[["x", "y", "z"]].corr()  # Pearson, pairwise dropna
```

Significance (p-value) for Pearson correlation
"If we assume bivariate normal data, we can test whether correlation is zero. We use a t statistic based on sample correlation and sample size."

Sample correlation r, sample size n

$$t = r\sqrt{\frac{n-2}{1-r^2}}, \quad \mathrm{df} = n-2$$

Two-sided p-value: $2(1 - F_t(|t|))$

```
from scipy import stats

xz = df[["x", "z"]].dropna()
r = xz["x"].corr(xz["z"])
n_eff = len(xz)
t = r * np.sqrt((n_eff - 2) / (1 - r**2))
p = 2 * (1 - stats.t.cdf(abs(t), df=n_eff - 2))

print("r =", r, "n =", n_eff, "t =", t, "p =", p)
```

Spearman correlation (rank correlation)
“It uses ranks, so it is more robust to outliers and non-linear shapes.”

Convert data to ranks: $r(x_i)$, $r(y_i)$

Then compute Pearson correlation on ranks:

$$\rho_s = \rho(r(x), r(y))$$

```
corr_s = df["x"].corr(df["y"], method="spearman")
print("Spearman corr(x,y) =", corr_s)
```

Kendall correlation (pairwise concordance)
“It counts how often two variables move in the same direction.”

For all pairs (i, j):

- concordant if $(x_i - x_j)(y_i - y_j) > 0$
- discordant if < 0

Kendall’s tau:

$$\tau = \frac{C - D}{\binom{n}{2}}$$

```
corr_k = df["x"].corr(df["y"], method="kendall")
```

```
print("Kendall tau(x,y) =", corr_k)
```

Rolling correlation (time series window)
"Rolling correlation measures correlation over a moving window. It shows how relationships change over time."
Math: Same Pearson formula, but computed on window t−w+1…t.

```
df_sorted = df.sort_values("date").reset_index(drop=True)

rolling_corr = df_sorted["x"].rolling(window=20,
min_periods=10).corr(df_sorted["z"])
df_sorted["roll_corr_xz_20"] = rolling_corr

print(df_sorted[["date", "x", "z", "roll_corr_xz_20"]].tail(5))
```

Grouped correlation (by stock/day/category)

"Grouped correlation computes correlation inside each group. "For example, correlation per stock or per sector.""

For each group g, compute $\rho_{xy}^{(g)}$ using only samples in g.

```
group_corr2 = df.groupby("group").apply(lambda g:
g["x"].corr(g["z"])).rename("corr_xz")
print(group_corr2)
```

Regression

OLS
"OLS finds the line or hyperplane that minimizes squared errors."

- Objective:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - x_i^{\top}\beta)^2$$

- Matrix solution (conceptually):

$$\hat{\beta} = (X^{\top}X)^{-1}X^{\top}y$$

```
# Build X with intercept
```

```
X = np.column_stack([np.ones(len(df)), df["x"].to_numpy(),
df["z"].fillna(df["z"].mean()).to_numpy()])
y = df["y"].to_numpy()

beta, residuals, rank, s = np.linalg.lstsq(X, y, rcond=None)
print("beta =", beta)  # [intercept, beta_x, beta_z]

y_hat = X @ beta
resid = y - y_hat
print("resid mean =", resid.mean(), "resid std =", resid.std())

# Use OLS directly
import statsmodels.api as sm
d = df[["y", "x", "z"]].dropna()
y = d["y"]
X = sm.add_constant(d[["x", "z"]])
model = sm.OLS(y, X).fit()
print(model.summary())

y_hat = res.predict(X)          # predictions
resid = y - y_hat               # residuals
```

R^2 and adjusted R^2

"R^2 tells how much variance is explained.Adjusted R^2 penalizes adding too many features."

- $RSS = \sum(y - \hat{y})^2$
- $TSS = \sum(y - \bar{y})^2$

$$R^2 = 1 - \frac{RSS}{TSS}$$

- Adjusted R^2, with p predictors (not counting intercept):

$$\bar{R}^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

```
n_ = len(y)
p = X.shape[1] - 1  # exclude intercept
rss = np.sum((y - y_hat)**2)
tss = np.sum((y - y.mean())**2)
```

```
r2 = 1 - rss / tss
adj_r2 = 1 - (1 - r2) * (n_ - 1) / (n_ - p - 1)
print("R2 =", r2, "Adj R2 =", adj_r2)

# Use OLS directly
print("R^2:", model.rsquared)
print("Adj R^2:", model.rsquared_adj)
```

Standard errors and t-stats

t-stat is coefficient divided by its standard error.
We can estimate uncertainty of coefficients.

Math (classic OLS assumptions)

- $\hat{\sigma}^2 = RSS/(n-k)$ where k is number of parameters (incl. intercept)
- $\mathrm{Var}(\hat{\beta}) = \hat{\sigma}^2(X^\top X)^{-1}$
- $SE_j = \sqrt{\mathrm{Var}(\hat{\beta})_{jj}}$
- $t_j = \hat{\beta}_j / SE_j$

```
from scipy import stats

k = X.shape[1]
sigma2 = rss / (n_ - k)
XtX_inv = np.linalg.inv(X.T @ X)
var_beta = sigma2 * XtX_inv
se = np.sqrt(np.diag(var_beta))
t_stats = beta / se
p_vals = 2 * (1 - stats.t.cdf(np.abs(t_stats), df=n_ - k))

out = pd.DataFrame({"beta": beta, "se": se, "t": t_stats, "p":
p_vals},
                   index=["intercept", "x", "z"])
```

Multicollinearity (VIF)

“If predictors are highly correlated, coefficients become unstable. VIF measures how much variance is inflated.”

For feature j, regress X_j on other features, get R_j^2

$$VIF_j = \frac{1}{1 - R_j^2}$$

```
def vif_for_column(X_no_intercept: np.ndarray, j: int) -> float:
    yj = X_no_intercept[:, j]
    X_other = np.delete(X_no_intercept, j, axis=1)
    X_other = np.column_stack([np.ones(len(X_other)), X_other])  #
intercept
    b, *_ = np.linalg.lstsq(X_other, yj, rcond=None)
    y_hat_j = X_other @ b
    rss_j = np.sum((yj - y_hat_j)**2)
    tss_j = np.sum((yj - yj.mean())**2)
    r2_j = 1 - rss_j / tss_j
    return 1.0 / (1.0 - r2_j)

X_feat = np.column_stack([
    df["x"].to_numpy(),
    df["z"].fillna(df["z"].mean()).to_numpy()
])
print("VIF x =", vif_for_column(X_feat, 0))
print("VIF z =", vif_for_column(X_feat, 1))
```

Robust standard errors

If you suspect heteroskedasticity. Note: **R^2 is the same**, but standard errors / t-stats / p-values change.

"Robust standard errors give more reliable t-stats and p-values under heteroskedasticity."

Homoskedasticity:

$$\mathrm{Var}(\varepsilon_i \mid X) = \sigma^2 \quad \text{(constant for all i)}$$

But in real data (finance especially), often:

$$\mathrm{Var}(\varepsilon_i \mid X) = \sigma_i^2 \quad \text{(depends on i / X)}$$

This is **heteroskedasticity**.

OLS variance formula under homoskedasticity:

$$\mathbf{Var}(\hat{\beta}) = \sigma^2 (X^\top X)^{-1}$$

Robust (heteroskedasticity-consistent) replaces $\sigma^2 I$ with a general “meat” matrix:

$$\widehat{\mathbf{Var}}(\hat{\beta}) = (X^\top X)^{-1} \Big(X^\top \hat{\Omega} X \Big) (X^\top X)^{-1}$$

```
model_robust = sm.OLS(y, X).fit(cov_type="HC3")
print(model_robust.summary())
print("R^2:", model_robust.rsquared)
```