Hands-On Lab — Exercise Handout

Time Series Forecasting: From VAR to Transformers and Diffusion

All six exercises live in `exercises.py` and run on a CPU with only `numpy` and `matplotlib`. Each block below is the worked starter code; the shaded boxes are the parts you complete. Stretch goals add `chronos-forecasting`, `gluonts`, and `statsforecast`.

1 Build (or load) the dataset

Create the synthetic hourly "load" series and hold out the last $H = 48$ hours.

```
import numpy as np

def make_series(n=24*30, seed=2):
    rng = np.random.default_rng(seed)
    t = np.arange(n)
    trend  = 50 + 0.005*t
    daily  = 12*np.sin(2*np.pi*t/24)
    weekly =  2*np.sin(2*np.pi*t/(24*7))
    noise  = rng.normal(0, 2.5, n)
    return t, trend + daily + weekly + noise

t, y = make_series()
H = 48                                  # forecast horizon
y_train, y_test = y[:-H], y[-H:]
```

Your turn. Replace `make_series()` with `pd.read_csv("your.csv")["value"].values`; plot the autocorrelation (`np.correlate`) and find the dominant period.

2 Classical VAR(1)

Simulate a bivariate VAR(1) with a known A, recover it by OLS, and read off impulse responses $B^{(h)} = A^h$.

```
A_true = np.array([[0.5, 0.3],
                   [0.0, 0.4]])     # note: y2 does NOT depend on y1's past

def simulate_var1(A, T=400, seed=3):
    rng = np.random.default_rng(seed)
    Y = np.zeros((T, A.shape[0]))
    for s in range(1, T):
        Y[s] = A @ Y[s-1] + rng.normal(0, 1, A.shape[0])
    return Y

def fit_var1(Y):                    # OLS: Y_t = A Y_{t-1} + e
    X, Z = Y[1:], Y[:-1]
    return np.linalg.lstsq(Z, X, rcond=None)[0].T

Y = simulate_var1(A_true)
A_hat = fit_var1(Y)                                     # ~ A_true
B = [np.linalg.matrix_power(A_hat, h) for h in range(11)]    # IRF
```

Your turn. Implement the Granger F-test of H_0: lags of y_1 do not enter the y_2 equation; turn F into a p-value with `scipy.stats.f`. You should find $F_{y_1 \to y_2} \approx 0$ (cannot reject) but $F_{y_2 \to y_1} \gg 0$. Then add an intercept and choose p by AIC.

3 Baselines and MASE

Seasonal-naive forecast with an 80% interval from a rolling-origin backtest.

```
m = 24                                  # daily period
sn = y_train[-m:][np.arange(H) % m]

def bt_resid(y, m, H, n=120):           # rolling-origin residuals
    R = []
    for o in range(len(y)-H-n, len(y)-H):
        f = y[o-m:o][np.arange(H) % m]
        R.append(y[o:o+H] - f)
    return np.array(R)

sig = bt_resid(y_train, m, H).std(0)
lo, hi = sn - 1.28*sig, sn + 1.28*sig                  # 80% interval

mase = (np.abs(y_test - sn).mean() /
        np.abs(y_train[m:] - y_train[:-m]).mean())    # ~1.06
```

Your turn. Add an AutoETS forecast (`statsforecast`) and report MASE for both. Explain why a one-step-error interval would be too narrow here.

4 Self-attention from scratch

Scaled dot-product attention over patch tokens.

```
def softmax(x, axis=-1):
    x = x - x.max(axis, keepdims=True)
    e = np.exp(x)
    return e / e.sum(axis, keepdims=True)

def attention(X, Wq, Wk, Wv):
    Q, K, V = X @ Wq, X @ Wk, X @ Wv
    dk = Q.shape[-1]
    S = Q @ K.T / np.sqrt(dk)          # scores
    A = softmax(S)                     # weights (rows sum to 1)
    return A @ V, A                    # output, attention map

Z, A = attention(X, Wq, Wk, Wv)
```

Your turn. Add a causal mask ($S_{ij} = -\infty$ for $j > i$) and confirm `A` is lower-triangular; then split into h heads, run each, and concatenate.

5 Diffusion forward process

Noise a window with the closed-form q, then stub the reverse step.

```
N = 100
betas  = np.linspace(1e-4, 0.08, N)
abar   = np.cumprod(1 - betas)         # alpha-bar

def q_sample(x0, n, rng):              # jump straight to level n
    eps = rng.standard_normal(len(x0))
    return np.sqrt(abar[n])*x0 + np.sqrt(1-abar[n])*eps
```

```
x0 = (y_test - y_test.mean()) / y_test.std()
noised = {n: q_sample(x0, n, np.random.default_rng(n))
          for n in (0, 20, 50, 99)}
```

Your turn. Given a trained `eps_theta(x, n)`, implement one reverse step $x^{(n-1)} = \frac{1}{\sqrt{\alpha_n}}\left(x^{(n)} - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\,\epsilon_\theta\right) + \sqrt{\beta_n}\,z$ and sample a window starting from $x^{(N)} \sim \mathcal{N}(0, I)$.

6 Probabilistic forecast and scoring

Correlated sample paths, a quantile fan, CRPS, and coverage.

```
S, phi = 200, 0.6
g = np.random.default_rng(7)
ar = np.zeros((S, H)); ar[:,0] = g.normal(0, sig[0], S)
for h in range(1, H):
    e = g.normal(0, sig[h]*np.sqrt(1-phi**2), S)
    ar[:,h] = phi*ar[:,h-1] + e
samples = sn[None,:] + ar
q10, q50, q90 = np.percentile(samples, [10, 50, 90], 0)

def crps(samps, obs):                # sample CRPS
    a = np.abs(samps - obs).mean(0)
    b = np.abs(samps[:,None] - samps[None]).mean((0,1))
    return (a - 0.5*b).mean()

cov = np.mean((y_test >= q10) & (y_test <= q90))    # ~0.7 for an 80% band
```

Your turn. The 80% band covers only ∼70% — diagnose and recalibrate. Stretch: replace `samples` with Chronos-2 zero-shot draws and beat this CRPS.

Deliverable. Report MASE *and* CRPS for the seasonal-naive baseline, one neural/foundation model, and one generative model on a dataset of your choice, using a rolling-origin backtest — and justify the model class you would ship.