Python linear regression project

Build a Python linear regression project that includes
1. CRISP-DM
2. allow user to modify a in ax+b, noise, number of points
3. streamlit or flask web, 框架 deployment

python Always show details

~~~
# This cell creates a complete Python project scaffold for a Linear Regression demo
# with CRISP-DM documentation, Streamlit and Flask apps, and deployment helpers.
import os, json, textwrap, zipfile, pathlib

project_root = "/mnt/data/linear-regression-crispdm"
os.makedirs(project_root, exist_ok=True)

# --------- Common package files ---------
requirements = textwrap.dedent("""
    numpy==2.0.2
    scikit-learn==1.5.2
    matplotlib==3.9.2
    streamlit==1.38.0
    flask==3.0.3
""").strip()

readme = textwrap.dedent("""
    # Linear Regression Demo (CRISP-DM) — Streamlit & Flask

    This project demonstrates end-to-end **linear regression** following the **CRISP-DM** methodology.
    It includes:
    - Synthetic data generation: y = a·x + b + noise
    - Controls for slope `a`, intercept `b`, Gaussian `noise`, and number of points `n`
    - **Streamlit** app for quick interactive exploration
    - **Flask** app for a lightweight web form
    - Reusable Python module (`linreg/`) and tests
    - Dockerfiles & deployment notes

    ## Quickstart (Streamlit)
    ```bash
~~~

~~~
pip install -r requirements.txt
streamlit run app_streamlit.py
```

## Quickstart (Flask)
```bash
pip install -r requirements.txt
flask --app app_flask:app run --port 7860 --debug
```

## Project Layout
```text
linear-regression-crispdm/
├── app_streamlit.py
├── app_flask.py
├── linreg/
│   ├── __init__.py
│   ├── data_gen.py
│   ├── model.py
│   └── viz.py
├── crispdm.md
├── requirements.txt
├── Dockerfile.streamlit
├── Dockerfile.flask
├── Procfile.streamlit
├── Procfile.flask
├── README.md
└── tests/
    └── test_data_gen.py
```

## Deployment
### Streamlit Community Cloud
1. Push this repo to GitHub.
2. In Streamlit Cloud, select your repo and set **Main file path** to `app_streamlit.py`.
3. Add secrets (none required) and deploy.

### Docker (Streamlit)
```bash
docker build -f Dockerfile.streamlit -t lr-streamlit .
docker run -p 8501:8501 lr-streamlit
```
~~~

~~~
    ### Docker (Flask)
    ```bash
    docker build -f Dockerfile.flask -t lr-flask .
    docker run -p 7860:7860 lr-flask
    ```

    ### Render / Fly.io / Railway (Flask)
    - Use `Dockerfile.flask` or `Procfile.flask` (command: `gunicorn -w 2 -b 0.0.0.0:7860 app_flask:app`).
    - Set the port to `7860` (or adapt as required by the platform).

    ----

    **Notes**
    - All plots are rendered with Matplotlib.
    - The model uses scikit-learn's `LinearRegression` and reports RMSE and R².
""").strip()

crispdm = textwrap.dedent("""
    # CRISP-DM for Linear Regression Demo

    ## 1. Business Understanding
    Goal: Provide an interactive environment to understand linear regression behavior under different data-generating processes.
    Questions:
    - How do slope `a`, intercept `b`, noise level `σ`, and sample size `n` affect fitted parameters and metrics (R², RMSE)?
    - How robust is OLS under noise and small-sample regimes?

    ## 2. Data Understanding
    Data Source: Synthetic generator `y = a·x + b + ε`, where `ε ~ Normal(0, σ²)` and `x ~ Uniform(x_min, x_max)`.
    Explorations:
    - Scatter of (x, y), residual histogram, residuals vs. fitted values, train/test split stability.
    - Summary stats: mean/var of x,y; noise diagnostics by comparing empirical residual σ with input σ.

    ## 3. Data Preparation
    Steps:
    - Generate X ∈ ℝ^{n×1}, y ∈ ℝ^{n}.
    - Standardize (optional): subtract mean and divide by std of X.
    - Train/test split with fixed random seed for reproducibility.
    - Validate no NaNs/inf; clip extreme noise if desired.
~~~

```
    ## 4. Modeling
    - Estimator: `sklearn.linear_model.LinearRegression(fit_intercept=True)`
    - Fit on training set, evaluate on test set.
    - Report: estimated slope and intercept, RMSE, R².
    - Variants to try: with/without intercept; ridge regression under high noise.

    ## 5. Evaluation
    - Compare fitted parameters against the ground-truth `a`, `b`.
    - Inspect residual plots and metric sensitivity to `σ` and `n`.
    - Sanity checks: With σ→0 and large n, parameters should approach `a`, `b`.

    ## 6. Deployment
    - **Streamlit app** for interactive analysis.
    - **Flask app** with web form for parameterized runs.
    - Containerization via Docker; deploy to Streamlit Cloud (Streamlit) or Render/Fly.io/Railway (Flask).
""").strip()

docker_streamlit = textwrap.dedent("""
    FROM python:3.11-slim

    WORKDIR /app
    COPY requirements.txt /app/
    RUN pip install --no-cache-dir -r requirements.txt

    COPY . /app
    EXPOSE 8501
    CMD ["streamlit", "run", "app_streamlit.py", "--server.address=0.0.0.0", "--server.port=8501"]
""").strip()

docker_flask = textwrap.dedent("""
    FROM python:3.11-slim

    WORKDIR /app
    COPY requirements.txt /app/
    RUN pip install --no-cache-dir -r requirements.txt && pip install --no-cache-dir gunicorn

    COPY . /app
    EXPOSE 7860
    CMD ["gunicorn", "-w", "2", "-b", "0.0.0.0:7860", "app_flask:app"]
""").strip()
```

```
procfile_streamlit = "web: streamlit run app_streamlit.py --server.port=$PORT --server.address=0.0.0.0"
procfile_flask = "web: gunicorn -w 2 -b 0.0.0.0:$PORT app_flask:app"

# --------- Python package (linreg) ---------
os.makedirs(f"{project_root}/linreg", exist_ok=True)
init_py = "__all__ = ['data_gen', 'model', 'viz']\n"
data_gen = textwrap.dedent("""
    from dataclasses import dataclass
    import numpy as np

    @dataclass
    class GenConfig:
        a: float = 2.0
        b: float = 0.0
        noise_sigma: float = 1.0
        n: int = 200
        x_min: float = -5.0
        x_max: float = 5.0
        random_state: int = 42

    def make_linear_data(cfg: GenConfig):
        rng = np.random.default_rng(cfg.random_state)
        x = rng.uniform(cfg.x_min, cfg.x_max, size=cfg.n).astype(float)
        eps = rng.normal(0.0, cfg.noise_sigma, size=cfg.n).astype(float)
        y = cfg.a * x + cfg.b + eps
        # shape to (n,1) for sklearn
        X = x.reshape(-1, 1)
        return X, y
""").strip()

model_py = textwrap.dedent("""
    from dataclasses import dataclass
    import numpy as np
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_squared_error, r2_score
    from sklearn.model_selection import train_test_split

    @dataclass
    class FitResult:
        slope: float
        intercept: float
        rmse: float
        r2: float
        y_pred: np.ndarray
        X_test: np.ndarray
        y_test: np.ndarray

    def fit_linear_regression(X, y, test_size=0.25, random_state=0):
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=random_state
        )
        lr = LinearRegression()
        lr.fit(X_train, y_train)
        y_pred = lr.predict(X_test)
        rmse = float(np.sqrt(mean_squared_error(y_test, y_pred)))
        r2 = float(r2_score(y_test, y_pred))
        return FitResult(
            slope=float(lr.coef_[0]),
            intercept=float(lr.intercept_),
            rmse=rmse,
            r2=r2,
            y_pred=y_pred,
            X_test=X_test,
            y_test=y_test,
        )
""").strip()

viz_py = textwrap.dedent("""
    import io, base64
    import numpy as np
    import matplotlib.pyplot as plt

    def scatter_with_fit(X_test, y_test, y_pred):
        fig, ax = plt.subplots()
        ax.scatter(X_test.ravel(), y_test, label="Test Data")
        # sort for line plotting
        order = np.argsort(X_test.ravel())
        ax.plot(X_test.ravel()[order], y_pred[order], label="Fitted Line")
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.legend()
        fig.tight_layout()
        return fig
    def fig_to_base64_png(fig) -> str:
        buf = io.BytesIO()
        fig.savefig(buf, format="png", bbox_inches="tight")
        buf.seek(0)
        b64 = base64.b64encode(buf.read()).decode("ascii")
        plt.close(fig)
        return b64
""").strip()

# --------- Streamlit app ---------
app_streamlit = textwrap.dedent("""
    import streamlit as st
    import numpy as np
    from linreg.data_gen import GenConfig, make_linear_data
    from linreg.model import fit_linear_regression
    from linreg.viz import scatter_with_fit

    st.set_page_config(page_title="Linear Regression — CRISP-DM Demo", layout="wide")

    st.title("Linear Regression — CRISP-DM Demo")
    st.caption("Adjust parameters for y = a·x + b + noise and fit a linear regression model.")

    with st.sidebar:
        st.header("Data Generation")
        a = st.number_input("Slope (a)", value=2.0, step=0.1, format="%.3f")
        b = st.number_input("Intercept (b)", value=0.0, step=0.1, format="%.3f")
        noise_sigma = st.slider("Noise σ", min_value=0.0, max_value=10.0, value=1.0, step=0.1)
        n = st.slider("Number of points (n)", min_value=20, max_value=5000, value=200, step=10)
        x_min, x_max = st.slider("x range", min_value=-20.0, max_value=20.0, value=(-5.0, 5.0), step=0.5)
        seed = st.number_input("Random seed", value=42, step=1)
        test_size = st.slider("Test size", min_value=0.1, max_value=0.9, value=0.25, step=0.05)

    cfg = GenConfig(a=a, b=b, noise_sigma=noise_sigma, n=int(n), x_min=x_min, x_max=x_max, random_state=int(seed))
    X, y = make_linear_data(cfg)
    res = fit_linear_regression(X, y, test_size=test_size, random_state=int(seed))

    col1, col2 = st.columns([2, 1])
    with col1:
        st.subheader("Scatter & Fitted Line (Test Split)")
        fig = scatter_with_fit(res.X_test, res.y_test, res.y_pred)
        st.pyplot(fig, clear_figure=True, use_container_width=True)
    with col2:
        st.subheader("Ground Truth vs Estimated")
        st.metric("Ground truth slope (a)", f"{a:.4f}")
        st.metric("Estimated slope", f"{res.slope:.4f}")
        st.metric("Ground truth intercept (b)", f"{b:.4f}")
        st.metric("Estimated intercept", f"{res.intercept:.4f}")
        st.divider()
        st.metric("RMSE", f"{res.rmse:.4f}")
        st.metric("R²", f"{res.r2:.4f}")

    with st.expander("CRISP-DM Summary"):
        st.markdown(\"\"\"
        **Business Understanding**: Explore how data-generation affects linear regression fit.
        **Data Understanding**: Synthetic y = a·x + b + ε with adjustable σ and n.
        **Data Preparation**: Generate, split, (optionally) standardize.
        **Modeling**: scikit-learn LinearRegression.
        **Evaluation**: Compare estimates to ground truth; RMSE, R²; residual plot.
        **Deployment**: This Streamlit app / Flask app & Docker.
        \"\"\")
""").strip()

# --------- Flask app ---------
app_flask = textwrap.dedent("""
    from flask import Flask, request, render_template_string
    from linreg.data_gen import GenConfig, make_linear_data
    from linreg.model import fit_linear_regression
    from linreg.viz import scatter_with_fit, fig_to_base64_png

    app = Flask(__name__)

    PAGE = \"\"\"
    <!doctype html>
    <html lang="en">
    <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <title>Linear Regression — CRISP-DM Demo</title>
      <style>
        body { font-family: system-ui, sans-serif; margin: 2rem; }
        form { display: grid; grid-template-columns: repeat(4, minmax(0, 220px)); gap: 1rem; align-items: end; }
        label { font-size: 0.9rem; }
        input { width: 100%; padding: 0.4rem; }
        .metrics { display: grid; grid-template-columns: repeat(4, 1fr); gap: 1rem; margin: 1.5rem 0; }
        .metric { padding: .75rem; border: 1px solid #ddd; border-radius: .5rem;}
        img { max-width: 100%; height: auto; }
        .note { color:#666; font-size:.9rem }
      </style>
    </head>
    <body>
      <h1>Linear Regression — CRISP-DM Demo</h1>
      <form method="POST">
        <div><label>Slope (a)<br><input type="number" step="0.1" name="a" value="{{a}}"></label></div>
        <div><label>Intercept (b)<br><input type="number" step="0.1" name="b" value="{{b}}"></label></div>
        <div><label>Noise σ<br><input type="number" step="0.1" name="noise_sigma" value="{{noise_sigma}}"></label></div>
        <div><label>n<br><input type="number" step="1" name="n" value="{{n}}"></label></div>
        <div><label>x min<br><input type="number" step="0.5" name="x_min" value="{{x_min}}"></label></div>
        <div><label>x max<br><input type="number" step="0.5" name="x_max" value="{{x_max}}"></label></div>
        <div><label>Seed<br><input type="number" step="1" name="seed" value="{{seed}}"></label></div>
        <div><label>Test size<br><input type="number" step="0.05" name="test_size" value="{{test_size}}"></label></div>
        <div style="grid-column: span 4;"><button type="submit">Run</button></div>
      </form>

      {% if metrics %}
      <div class="metrics">
        {% for k, v in metrics.items() %}
          <div class="metric"><strong>{{k}}</strong><br>{{v}}</div>
        {% endfor %}
      </div>
      <img src="data:image/png;base64,{{plot_b64}}" alt="scatter-fitted">
      {% endif %}

      <p class="note">CRISP-DM: Business→Data Understanding→Preparation→Modeling→Evaluation→Deployment</p>
    </body>
    </html>
    \"\"\"

    def parse_float(form, key, default):
        try:
            return float(form.get(key, default))
        except (TypeError, ValueError):
            return default

    def parse_int(form, key, default):
        try:
            return int(form.get(key, default))
        except (TypeError, ValueError):
            return default

    @app.route("/", methods=["GET", "POST"])
    def index():
        # defaults
        a=2.0; b=0.0; noise_sigma=1.0; n=200; x_min=-5.0; x_max=5.0; seed=42; test_size=0.25
        if request.method == "POST":
            a = parse_float(request.form, "a", a)
            b = parse_float(request.form, "b", b)
            noise_sigma = parse_float(request.form, "noise_sigma", noise_sigma)
            n = parse_int(request.form, "n", n)
            x_min = parse_float(request.form, "x_min", x_min)
            x_max = parse_float(request.form, "x_max", x_max)
            seed = parse_int(request.form, "seed", seed)
            test_size = parse_float(request.form, "test_size", test_size)

        cfg = GenConfig(a=a, b=b, noise_sigma=noise_sigma, n=n, x_min=x_min, x_max=x_max, random_state=seed)
        X, y = make_linear_data(cfg)
        res = fit_linear_regression(X, y, test_size=test_size, random_state=seed)

        metrics = {
            "Ground truth slope (a)": f"{a:.4f}",
            "Estimated slope": f"{res.slope:.4f}",
            "Ground truth intercept (b)": f"{b:.4f}",
            "Estimated intercept": f"{res.intercept:.4f}",
            "RMSE": f"{res.rmse:.4f}",
            "R²": f"{res.r2:.4f}",
        }
        fig = scatter_with_fit(res.X_test, res.y_test, res.y_pred)
        plot_b64 = fig_to_base64_png(fig)

        return render_template_string(PAGE, a=a, b=b, noise_sigma=noise_sigma, n=n, x_min=x_min, x_max=x_max,
                                      seed=seed, test_size=test_size, metrics=metrics, plot_b64=plot_b64)

    if __name__ == "__main__":
        app.run(host="0.0.0.0", port=7860, debug=True)
""").strip()

test_data = textwrap.dedent("""
    from linreg.data_gen import GenConfig, make_linear_data

    def test_shapes():
        cfg = GenConfig(n=123)
        X, y = make_linear_data(cfg)
        assert X.shape == (123, 1)
        assert y.shape == (123,)
""").strip()

# --------- Write files ---------
files = {
    "requirements.txt": requirements,
    "README.md": readme,
    "crispdm.md": crispdm,
    "Dockerfile.streamlit": docker_streamlit,
    "Dockerfile.flask": docker_flask,
    "Procfile.streamlit": procfile_streamlit,
    "Procfile.flask": procfile_flask,
    "linreg/__init__.py": init_py,
    "linreg/data_gen.py": data_gen,
    "linreg/model.py": model_py,
    "linreg/viz.py": viz_py,
    "app_streamlit.py": app_streamlit,
    "app_flask.py": app_flask,
    "tests/test_data_gen.py": test_data,
}
for rel, content in files.items():
    p = pathlib.Path(project_root) / rel
    p.parent.mkdir(parents=True, exist_ok=True)
    with open(p, "w", encoding="utf-8") as f:
        f.write(content)
```

```
# Create a zip for download
zip_path = "/mnt/data/linear-regression-crispdm.zip"
with zipfile.ZipFile(zip_path, "w", zipfile.ZIP_DEFLATED) as zf:
    for root, dirs, filenames in os.walk(project_root):
        for fn in filenames:
            full = os.path.join(root, fn)
            zf.write(full, os.path.relpath(full, project_root))

zip_path
```

Result

```
'/mnt/data/linear-regression-crispdm.zip'
```

Awesome—here's a complete, ready-to-run mini-project for linear regression that:

- follows **CRISP-DM**,
- lets users tweak **slope a (and b), noise σ, and number of points n**, and
- ships with **both Streamlit and Flask** web apps + **Docker** and quick deployment notes.

Download the project (ZIP)

How to run (pick one)

Option A — Streamlit (interactive sliders)

bash

```bash
unzip linear-regression-crispdm.zip && cd linear-regression-crispdm
pip install -r requirements.txt
streamlit run app_streamlit.py
```

Then open the local URL Streamlit prints (default http://localhost:8501 ↗). Use the sidebar to adjust:

- slope `a` (and `b`),
- noise σ,
- number of points `n`,
- x-range, seed, and test size.

You'll see:

- scatter + fitted line (Matplotlib),
- metrics (estimated slope/intercept, RMSE, R^2),

- a CRISP-DM summary panel.

Option B — Flask (simple web form)

```bash
unzip linear-regression-crispdm.zip && cd linear-regression-crispdm
pip install -r requirements.txt
flask --app app_flask:app run --port 7860 --debug
```

Open http://localhost:7860 ↗, fill in `a`, `b`, `σ`, `n`, etc., submit, and see the fitted plot + metrics.

CRISP-DM (included as `crispdm.md`)

- **Business Understanding**: Explore how data-gen parameters impact OLS estimates/metrics.
- **Data Understanding**: Synthetic $y = ax + b + \varepsilon$ with tunable σ and n.
- **Data Preparation**: Generate, split, optional standardization, reproducible seed.
- **Modeling**: `sklearn.linear_model.LinearRegression`.
- **Evaluation**: Compare estimated vs ground truth (a, b), report RMSE & R^2, residual plots (code hooks included).
- **Deployment**: Streamlit/Flask apps; Dockerfiles and cloud notes.

Deployment (quick)

Streamlit Community Cloud

1. Push to GitHub → 2) “New app”, set **Main file path** = `app_streamlit.py` → Deploy.

Docker (Streamlit)

```bash
docker build -f Dockerfile.streamlit -t lr-streamlit .
docker run -p 8501:8501 lr-streamlit
```

Docker (Flask)

```bash
docker build -f Dockerfile.flask -t lr-flask .
docker run -p 7860:7860 lr-flask
```

Render / Fly.io / Railway (Flask)

Use `Dockerfile.flask` **or**:

```
bash


Procfile.flask:  web: gunicorn -w 2 -b 0.0.0.0:$PORT app_flask:app
```

What’s inside (key files)

- `app_streamlit.py` — Interactive sliders UI (a, b, σ, n, x-range, seed, test size) + Matplotlib plot + metrics.
- `app_flask.py` — Minimal Flask form; returns metrics and base64 plot image.
- `linreg/data_gen.py` — Synthetic data generator (`GenConfig`, `make_linear_data`).
- `linreg/model.py` — Fit/evaluate (`LinearRegression`, RMSE, R^2).
- `linreg/viz.py` — Matplotlib scatter + fitted line, plus PNG encoder for Flask.
- `crispdm.md` — The CRISP-DM writeup tailored to this project.
- `Dockerfile.streamlit` / `Dockerfile.flask` — Containerized deployment.
- `Procfile.streamlit` / `Procfile.flask` — PaaS-friendly commands.
- `tests/test_data_gen.py` — Tiny sanity test for data shapes.
- `requirements.txt`, `README.md`.

If you want me to tailor the UI text to Chinese, add ridge/L2 toggle, residual plots, or seed-sweep experiments for variance analysis, say the word and I’ll extend the project.

I'm not using Docker.
Write a single py file so that I use command to run

Got it 👍 — here’s a **single Python file** version that you can run directly without Docker.
You can choose **Streamlit mode** (interactive sliders) or **Flask mode** (simple web form) by passing an argument in the command line.

Save this as `linear_regression_app.py`:

python

```python
import argparse
import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# ----------------------
# Data generation
# ----------------------
@dataclass
class GenConfig:
    a: float = 2.0
    b: float = 0.0
    noise_sigma: float = 1.0
    n: int = 200
    x_min: float = -5.0
    x_max: float = 5.0
    random_state: int = 42

def make_linear_data(cfg: GenConfig):
    rng = np.random.default_rng(cfg.random_state)
    x = rng.uniform(cfg.x_min, cfg.x_max, size=cfg.n)
    eps = rng.normal(0.0, cfg.noise_sigma, size=cfg.n)
    y = cfg.a * x + cfg.b + eps
    return x.reshape(-1, 1), y

def fit_linear_regression(X, y, test_size=0.25, random_state=0):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state
    )
    lr = LinearRegression()
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    rmse = float(np.sqrt(mean_squared_error(y_test, y_pred)))
    r2 = float(r2_score(y_test, y_pred))
    return lr, X_test, y_test, y_pred, rmse, r2

# ----------------------
# Streamlit UI
# ------------------------
def run_streamlit():
    import streamlit as st
    st.set_page_config(page_title="Linear Regression — CRISP-DM Demo", layout="wide")

    st.title("Linear Regression — CRISP-DM Demo")

    with st.sidebar:
        st.header("Data Generation")
        a = st.number_input("Slope (a)", value=2.0)
        b = st.number_input("Intercept (b)", value=0.0)
        noise_sigma = st.slider("Noise σ", 0.0, 10.0, 1.0, 0.1)
        n = st.slider("Number of points", 20, 5000, 200, 10)
        x_min, x_max = st.slider("x range", -20.0, 20.0, (-5.0, 5.0))
        seed = st.number_input("Random seed", value=42)
        test_size = st.slider("Test size", 0.1, 0.9, 0.25, 0.05)

    cfg = GenConfig(a=a, b=b, noise_sigma=noise_sigma, n=n, x_min=x_min, x_max=x_max, random_state=int(seed))
    X, y = make_linear_data(cfg)
    lr, X_test, y_test, y_pred, rmse, r2 = fit_linear_regression(X, y, test_size=test_size, random_state=int(seed))

    col1, col2 = st.columns([2, 1])
    with col1:
        fig, ax = plt.subplots()
        ax.scatter(X_test.ravel(), y_test, label="Test data")
        order = np.argsort(X_test.ravel())
        ax.plot(X_test.ravel()[order], y_pred[order], color="red", label="Fitted line")
        ax.legend()
        st.pyplot(fig)

    with col2:
        st.metric("Ground truth slope (a)", f"{a:.4f}")
        st.metric("Estimated slope", f"{lr.coef_[0]:.4f}")
        st.metric("Ground truth intercept (b)", f"{b:.4f}")
        st.metric("Estimated intercept", f"{lr.intercept_:.4f}")
        st.divider()
        st.metric("RMSE", f"{rmse:.4f}")
        st.metric("R²", f"{r2:.4f}")

# ------------------------
# Flask UI
# -----------------------
def run_flask():
    from flask import Flask, request, render_template_string
    import base64, io

    app = Flask(__name__)

    PAGE = """
    <h1>Linear Regression — CRISP-DM Demo</h1>
    <form method="POST">
      a: <input type="number" step="0.1" name="a" value="{{a}}"> |
      b: <input type="number" step="0.1" name="b" value="{{b}}"> |
      noise σ: <input type="number" step="0.1" name="noise_sigma" value="{{noise_sigma}}"> |
      n: <input type="number" step="1" name="n" value="{{n}}"><br><br>
      <button type="submit">Run</button>
    </form>
    {% if metrics %}
    <h3>Results</h3>
    <pre>{{metrics}}</pre>
    <img src="data:image/png;base64,{{plot_b64}}">
    {% endif %}
    """

    def fig_to_base64(fig):
        buf = io.BytesIO()
        fig.savefig(buf, format="png")
        buf.seek(0)
        return base64.b64encode(buf.read()).decode("utf-8")

    @app.route("/", methods=["GET", "POST"])
    def index():
        a, b, noise_sigma, n = 2.0, 0.0, 1.0, 200
        if request.method == "POST":
            a = float(request.form.get("a", a))
            b = float(request.form.get("b", b))
            noise_sigma = float(request.form.get("noise_sigma", noise_sigma))
            n = int(request.form.get("n", n))

        cfg = GenConfig(a=a, b=b, noise_sigma=noise_sigma, n=n)
        X, y = make_linear_data(cfg)
        lr, X_test, y_test, y_pred, rmse, r2 = fit_linear_regression(X, y)
        metrics = {
            "Estimated slope": f"{lr.coef_[0]:.4f}",
            "Estimated intercept": f"{lr.intercept_:.4f}",
            "RMSE": f"{rmse:.4f}",
            "R²": f"{r2:.4f}",
        }
        fig, ax = plt.subplots()
        ax.scatter(X_test.ravel(), y_test)
        order = np.argsort(X_test.ravel())
        ax.plot(X_test.ravel()[order], y_pred[order], color="red")
        plot_b64 = fig_to_base64(fig)
        return render_template_string(PAGE, a=a, b=b, noise_sigma=noise_sigma, n=n,
                                      metrics=metrics, plot_b64=plot_b64)

    app.run(host="0.0.0.0", port=7860, debug=True)

# -----------------------
# Main CLI
# -----------------------
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--mode", choices=["streamlit", "flask"], default="streamlit",
                        help="Choose web framework")
    args = parser.parse_args()

    if args.mode == "streamlit":
        run_streamlit()
    else:
        run_flask()
```

How to run

Streamlit version

bash

```bash
pip install numpy scikit-learn matplotlib streamlit flask
streamlit run linear_regression_app.py -- --mode streamlit
```

Flask version

```bash
pip install numpy scikit-learn matplotlib streamlit flask
python linear_regression_app.py --mode flask
```

→ Open http://localhost:7860

Do you want me to also include the **CRISP-DM writeup section** directly in this single `.py` file (printed in console or accessible in UI), or just keep it in docs like before?