Appendix A — Code Listings and Figures

data_utils.py

```
"""
Utility functions for the IU Akademie Python visual-analytics assignment.

Comment styles shown
--------------------
1. Module doc-string (this block) – explains the whole file.
2. Function doc-strings – describe purpose, parameters, return value.
3. Inline comments (# like this) – clarify single statements.

Functions
---------
align_grid(df_a, df_b)
    Align two data frames to their common x-grid.

rmse(a, b)
    Root-mean-square error between two numeric arrays.

match_train_to_ideal(train_df, ideal_df)
    Map each train curve to the ideal curve with the lowest RMSE.

compute_residuals_df(train_df, ideal_df)
    Aggregated RMSE per train curve (long form).

residuals_long(train_df, ideal_df)
    Point-wise residuals for KDE or histogram plots.

build_residual_table(test_df, ideal_df)
    Classify each test point to the closest ideal curve (for Bokeh demo).
"""

from __future__ import annotations
```

```
import numpy as np
import pandas as pd


# ---------------------------------------------------------
# Grid handling helpers
# ---------------------------------------------------------
def align_grid(
    df_a: pd.DataFrame,
    df_b: pd.DataFrame,
) -> tuple[pd.DataFrame, pd.DataFrame]:
    """
    Filter *df_a* and *df_b* so they share exactly the same x-values,
    then return the two aligned frames sorted by x.
    """
    common_mask = df_a["x"].isin(df_b["x"])
    df_a_aligned = df_a[common_mask].sort_values("x")
    df_b_aligned = df_b[df_b["x"].isin(df_a["x"])].sort_values("x")
    return df_a_aligned, df_b_aligned


# ---------------------------------------------------------
# Error metric
# ---------------------------------------------------------
def rmse(array_a: np.ndarray, array_b: np.ndarray) -> float:
    """Return the root-mean-square error between two equal-length arrays."""
    return float(np.sqrt(np.mean((array_a - array_b) ** 2)))


# ---------------------------------------------------------
# Curve matching
# ---------------------------------------------------------
def match_train_to_ideal(
```

```
    train_df: pd.DataFrame,
    ideal_df: pd.DataFrame,
) -> dict[str, str]:
    """
    For every y-column in *train_df* find the y-column in *ideal_df*
    with the lowest RMSE.  Returns a mapping {train_curve: ideal_curve}.
    """
    matches: dict[str, str] = {}
    train_df, ideal_df = align_grid(train_df, ideal_df)

    train_cols = [c for c in train_df.columns if c != "x"]
    ideal_cols = [c for c in ideal_df.columns if c != "x"]

    for tcol in train_cols:
        best_col = None
        best_err = float("inf")
        for icol in ideal_cols:
            err = rmse(train_df[tcol].to_numpy(), ideal_df[icol].to_numpy())
            if err < best_err:
                best_err, best_col = err, icol
        matches[tcol] = best_col  # type: ignore[arg-type]
    return matches



# ---------------------------------------------------------------
# Residual analysis (aggregated)
# ---------------------------------------------------------------
def compute_residuals_df(
    train_df: pd.DataFrame,
    ideal_df: pd.DataFrame,
) -> pd.DataFrame:
    """
    Return a tidy DataFrame with one row per train curve and a single
    *rmse* value, suitable for bar- or point-plots.
```

```
    """
    matches = match_train_to_ideal(train_df, ideal_df)
    rows: list[dict[str, float | str]] = []

    train_df, ideal_df = align_grid(train_df, ideal_df)
    for tcol, icol in matches.items():
        error = rmse(train_df[tcol].to_numpy(), ideal_df[icol].to_numpy())
        rows.append({"curve": tcol, "rmse": error})
    return pd.DataFrame(rows)


# ------------------------------------------------------------
# Residual analysis (point-wise, for KDE)
# ------------------------------------------------------------
def residuals_long(
    train_df: pd.DataFrame,
    ideal_df: pd.DataFrame,
) -> pd.DataFrame:
    """
    Create a long-form DataFrame of *all* point-wise residuals between
    each train curve and its best-matching ideal curve.
    """
    matches = match_train_to_ideal(train_df, ideal_df)
    train_df, ideal_df = align_grid(train_df, ideal_df)

    rows: list[dict[str, float | str]] = []
    for tcol, icol in matches.items():
        diff = train_df[tcol].to_numpy() - ideal_df[icol].to_numpy()
        for d in diff:
            rows.append({"curve": tcol, "residual": d})
    return pd.DataFrame(rows)


# ------------------------------------------------------------
```

```
# Classification of test points
# ------------------------------------------------------------
def build_residual_table(
    test_df: pd.DataFrame,
    ideal_df: pd.DataFrame,
) -> pd.DataFrame:
    """
    For every (x, y) pair in *test_df* find the ideal-curve value with the
    minimum absolute error.  Returned table drives the Bokeh explorer.
    """
    _, ideal_df = align_grid(test_df, ideal_df)
    ideal_cols = [c for c in ideal_df.columns if c != "x"]

    records: list[dict[str, float | str]] = []
    for _, row in test_df.iterrows():
        x_val, y_val = row["x"], row["y"]
        slice_ideal = ideal_df.loc[ideal_df["x"] == x_val, ideal_cols].iloc[0]
        best_col = (slice_ideal - y_val).abs().idxmin()
        records.append(
            {
                "x": x_val,
                "y_test": y_val,
                "y_ideal": slice_ideal[best_col],
                "curve": best_col,
            }
        )
    return pd.DataFrame(records)
```

plot_overlay_matplotlib.py

```
"""
Overlay plot: each train curve with its best-matching ideal curve.

Produces `fig_overlay.png` (300 dpi) in the project root.
"""
```

```
import matplotlib.pyplot as plt
import pandas as pd

from data_utils import match_train_to_ideal

# ---------- load data ----------
train_df = pd.read_csv(
    "C:/Users/lanta/Desktop/python/assignment/dataset_train.csv")
ideal_df = pd.read_csv(
    "C:/Users/lanta/Desktop/python/assignment/dataset_ideal.csv")

# ---------- find best matches ----------
matches = match_train_to_ideal(train_df, ideal_df)

# ---------- plotting ----------
fig, ax = plt.subplots(figsize=(8, 4.5))

for train_col, ideal_col in matches.items():
    ax.plot(  # solid line: train data
        train_df["x"],
        train_df[train_col],
        label=f"{train_col} (train)",
        linewidth=1.0,
    )
    ax.plot(  # dashed line: ideal data
        ideal_df["x"],
        ideal_df[ideal_col],
        linestyle="--",
        label=f"{ideal_col} (ideal)",
    )

ax.set_title("Train vs. Ideal Curves (Overlay)")
ax.set_xlabel("x")
```

```
ax.set_ylabel("y")
ax.legend(ncol=4, fontsize="small")
fig.tight_layout()
fig.savefig("fig_overlay.png", dpi=300)
plt.show()
```

plot_residual_kde_seaborn.py

```
"""
Kernel-density plot of RMSE values for all train curves.

Outputs `fig_kde.png` (300 dpi).
"""

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from data_utils import compute_residuals_df

# ---------- load ----------
train_df = pd.read_csv(
    "C:/Users/lanta/Desktop/python/assignment/dataset_train.csv")
ideal_df = pd.read_csv(
    "C:/Users/lanta/Desktop/python/assignment/dataset_ideal.csv")

# ---------- residuals ----------
residuals = compute_residuals_df(train_df, ideal_df)

# ---------- bar plot ----------
sns.set_style("whitegrid")
ax = sns.barplot(
    data=residuals,
    x="curve",
    y="rmse",
```

```
    palette="pastel",
)
ax.set_title("RMSE per Train Curve")
ax.set_xlabel("train curve")
ax.set_ylabel("root mean square error")
fig = ax.get_figure()
fig.tight_layout()
fig.savefig("fig_kde.png", dpi=300)   # keep same filename for consistency
plt.show()
```

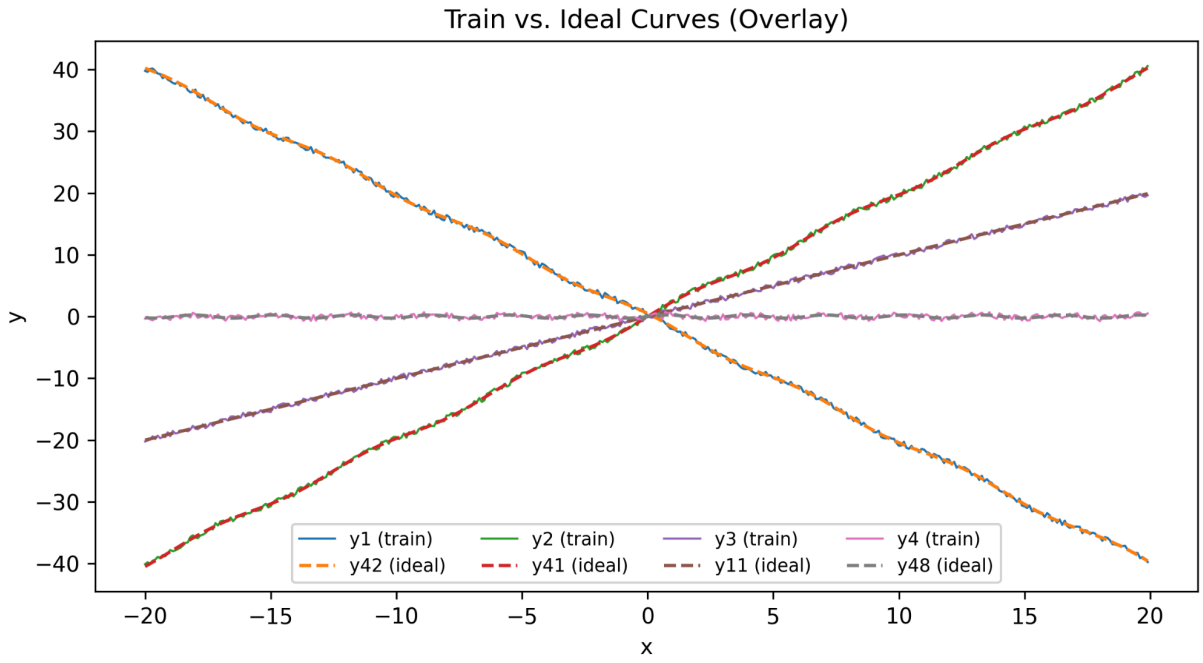
interactive_curve_explorer_bokeh.py

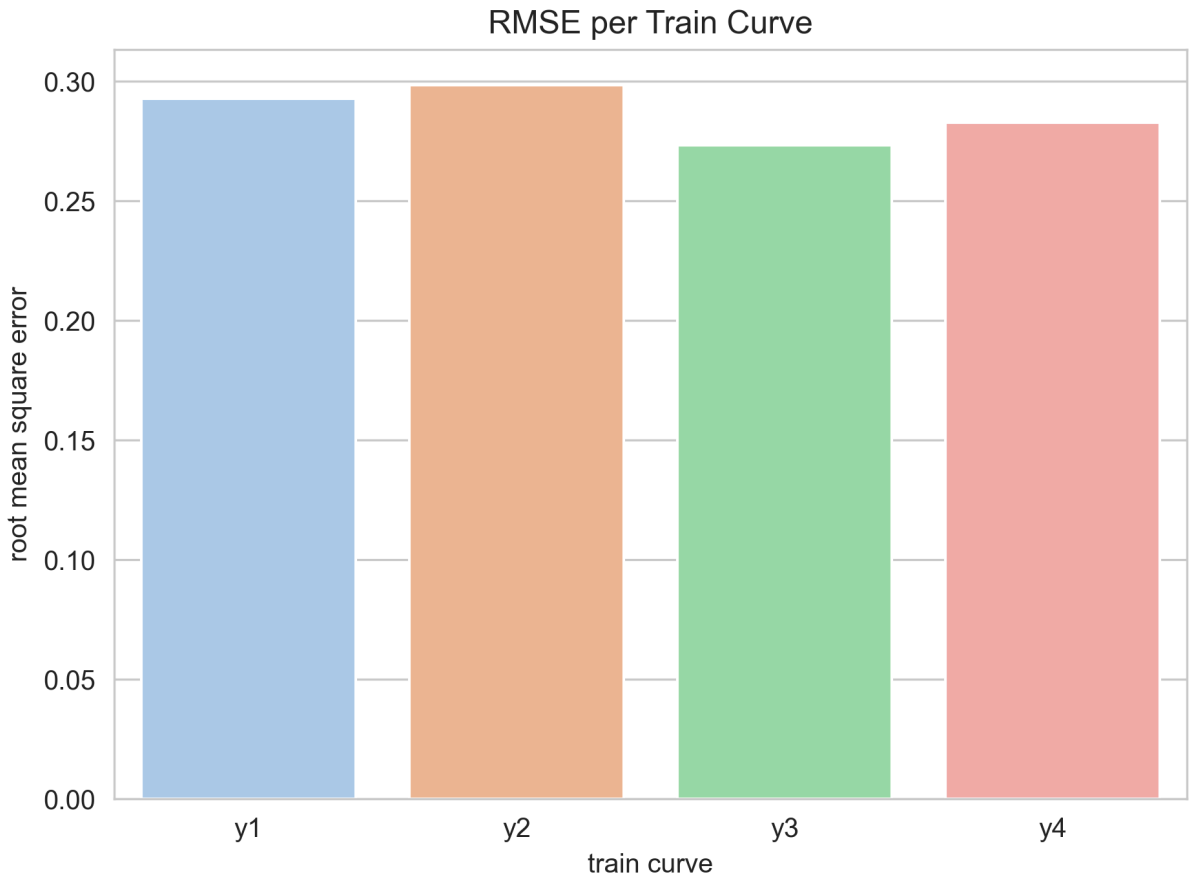
```
"""
Interactive explorer: select an ideal curve (dropdown) and compare
its values with the corresponding test points.

Outputs `bokeh_explorer.html` (open in any browser or via Jupyter).
"""

from bokeh.io import output_file, show
from bokeh.layouts import column
from bokeh.models import ColumnDataSource, CustomJS, Select
from bokeh.plotting import figure
import pandas as pd

from data_utils import build_residual_table

# ---------- load ----------
test_df = pd.read_csv(
    "C:/Users/lanta/Desktop/python/assignment/dataset_test.csv")
ideal_df = pd.read_csv(
    "C:/Users/lanta/Desktop/python/assignment/dataset_ideal.csv")

table = build_residual_table(test_df, ideal_df)
first_curve = table["curve"].iloc[0]
```

```
source = ColumnDataSource(table[table["curve"] == first_curve])

# ---------- base figure ----------
plot = figure(
    title="Interactive Curve Explorer",
    x_axis_label="x",
    y_axis_label="y",
    width=800,
    height=400,
)
plot.circle("x", "y_test", source=source, size=5,
            color="blue", legend_label="test")
plot.line("x", "y_ideal", source=source, color="red", legend_label="ideal")

# ---------- dropdown widget ----------
dropdown = Select(
    title="Curve",
    value=first_curve,
    options=sorted(table["curve"].unique()),
)

dropdown.js_on_change(
    "value",
    CustomJS(
        args=dict(src=source, full_table=table.to_dict(orient="list")),
        code="""
        const curve = cb_obj.value;
        const data  = {x: [], y_test: [], y_ideal: []};
        for (let i = 0; i < full_table['curve'].length; i++) {
            if (full_table['curve'][i] === curve) {
                data['x'].push(full_table['x'][i]);
                data['y_test'].push(full_table['y_test'][i]);
                data['y_ideal'].push(full_table['y_ideal'][i]);
            }
```

```
        }
        src.data = data;
        src.change.emit();
    """,
    ),
)

# ---------- output ----------
output_file("bokeh_explorer.html", title="Interactive Curve Explorer")
show(column(dropdown, plot))
```

Figures




RMSE per Train Curve
root mean square error
0.30
0.25
0.20
0.15
0.10
0.05
0.00
y1
y2
y3
y4
train curve