Exploratory Data Analysis of Anscombe's Quartet

Author: Avveer Singh Khurana
Date: 2025-10-21
GitHub: AvveerSchoolAccount
Notebook / Code: GitHub Repository

Short Description

This notebook performs an exploratory data analysis (EDA) on **Anscombe's Quartet** — a dataset that demonstrates how identical summary statistics can hide very different data patterns.
We compute statistics, plot visualizations, and discuss why **EDA must include visual inspection**.

Abstract / Executive Summary

Anscombe's Quartet (Francis Anscombe, 1973) shows that four datasets can share identical means, variances, correlations, and regression lines — yet look completely different when plotted.

This notebook demonstrates the importance of visualization in exploratory data analysis (EDA) by computing these statistics and visualizing patterns using scatter, residual, and distribution plots.

Introduction

Anscombe's Quartet was created by statistician *Francis Anscombe* in 1973 to highlight the importance of graphing data before analyzing it.
Each of the four datasets has nearly identical summary statistics (mean, variance, correlation, regression line, and R^2), but the data points form completely different patterns when plotted.

The purpose of this analysis is to:

- Compute key summary statistics for each dataset
- Visualize the datasets using multiple plot types
- Show that **statistics alone can be misleading without visualization**

Data

The dataset used here is **Anscombe's Quartet**, consisting of four small datasets labeled I, II, III, and IV.
Each dataset contains pairs of `x` and `y` values.

For reproducibility, the dataset was recreated directly within the notebook instead of being loaded from a file.

In [1]:
```
# --- Imports ---
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

# --- Step 1: Create the dataset ---
data = {
    'x123': [10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5],
    'y1': [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5
    'y2': [9.14, 8.14, 8.74, 8.77, 9.26, 8.1, 6.13, 3.1, 9.13, 7.26, 4.74
    'y3': [7.46, 6.77, 12.74, 7.11, 7.81, 8.84, 6.08, 5.39, 8.15, 6.42, 5
    'x4': [8, 8, 8, 8, 8, 8, 8, 19, 8, 8, 8],
    'y4': [6.58, 5.76, 7.71, 8.84, 8.47, 7.04, 5.25, 12.5, 5.56, 7.91, No
}

df = pd.DataFrame(data)
df.head()
```

Out[1]:

	x123	y1	y2	y3	x4	y4
0	10	8.04	9.14	7.46	8	6.58
1	8	6.95	8.14	6.77	8	5.76
2	13	7.58	8.74	12.74	8	7.71
3	9	8.81	8.77	7.11	8	8.84
4	11	8.33	9.26	7.81	8	8.47

Methods

We calculate the following summary statistics for each dataset:

- Mean
- Variance
- Standard deviation
- Covariance
- Pearson correlation coefficient
- Linear regression slope, intercept, and R^2

Summary Statistics Formulas

Mean: [\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i]

Sample Variance: [s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2]

Sample Standard Deviation: $$ s_x = \sqrt{s_x^2} $$

Covariance: $$ \text{cov}(x, y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) $$

Pearson Correlation: $$ r_{xy} = \frac{\text{cov}(x, y)}{s_x s_y} $$

Linear Regression (Least Squares): $$ y = b_0 + b_1x $$ where
$$ b_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}, \quad b_0 = \bar{y} - b_1\bar{x} $$

Coefficient of Determination (R^2): $$ R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2} $$

In [2]:
```python
# --- Step 2: Reshape the data (melt) ---
df_melted = pd.melt(
    df,
    id_vars=['x123', 'x4'],
    value_vars=['y1', 'y2', 'y3', 'y4'],
    var_name='dataset',
    value_name='y'
)

# Assign correct x column for each dataset
df_melted['x'] = df_melted.apply(
    lambda row: row['x123'] if row['dataset'] in ['y1', 'y2', 'y3'] else
    axis=1
)

df_melted.drop(columns=['x123', 'x4'], inplace=True)

# --- Step 3: Define summary statistics function ---
def summary_stats(g):
    n = g.shape[0]
    mean_x = g['x'].mean()
    mean_y = g['y'].mean()
    var_x = g['x'].var(ddof=1)
    var_y = g['y'].var(ddof=1)
    std_x = g['x'].std(ddof=1)
    std_y = g['y'].std(ddof=1)
    cov = g[['x', 'y']].cov().iloc[0, 1]
    corr = g['x'].corr(g['y'])

    X = sm.add_constant(g['x'])
    model = sm.OLS(g['y'], X).fit()
    slope = model.params['x']
    intercept = model.params['const']
    r2 = model.rsquared

    return pd.Series({
        'n': n,
        'mean_x': mean_x, 'mean_y': mean_y,
        'var_x': var_x, 'var_y': var_y,
        'std_x': std_x, 'std_y': std_y,
        'cov_xy': cov, 'r_xy': corr,
        'slope': slope, 'intercept': intercept, 'r2': r2
    })

# --- Step 4: Compute summary table ---
```

```
summary_table = df_melted.groupby('dataset').apply(lambda g: summary_stat
summary_table
```

```
/tmp/ipykernel_13640/960195626.py:46: FutureWarning: DataFrameGroupBy.appl
y operated on the grouping columns. This behavior is deprecated, and in a
future version of pandas the grouping columns will be excluded from the op
eration. Either pass `include_groups=False` to exclude the groupings or ex
plicitly select the grouping columns after groupby to silence this warnin
g.
  summary_table = df_melted.groupby('dataset').apply(lambda g: summary_sta
ts(g)).round(4)
```

Out[2]:

dataset	n	mean_x	mean_y	var_x	var_y	std_x	std_y	cov_xy	r_xy	slope
y1	11.0	9.0	7.5009	11.0	4.1273	3.3166	2.0316	5.5010	0.8164	0.5001
y2	11.0	9.0	7.5009	11.0	4.1276	3.3166	2.0317	5.5000	0.8162	0.5000
y3	11.0	9.0	7.5000	11.0	4.1226	3.3166	2.0304	5.4970	0.8163	0.4997
y4	11.0	9.0	7.5620	11.0	4.5358	3.3166	2.1297	6.0353	0.8147	NaN

Visualizations

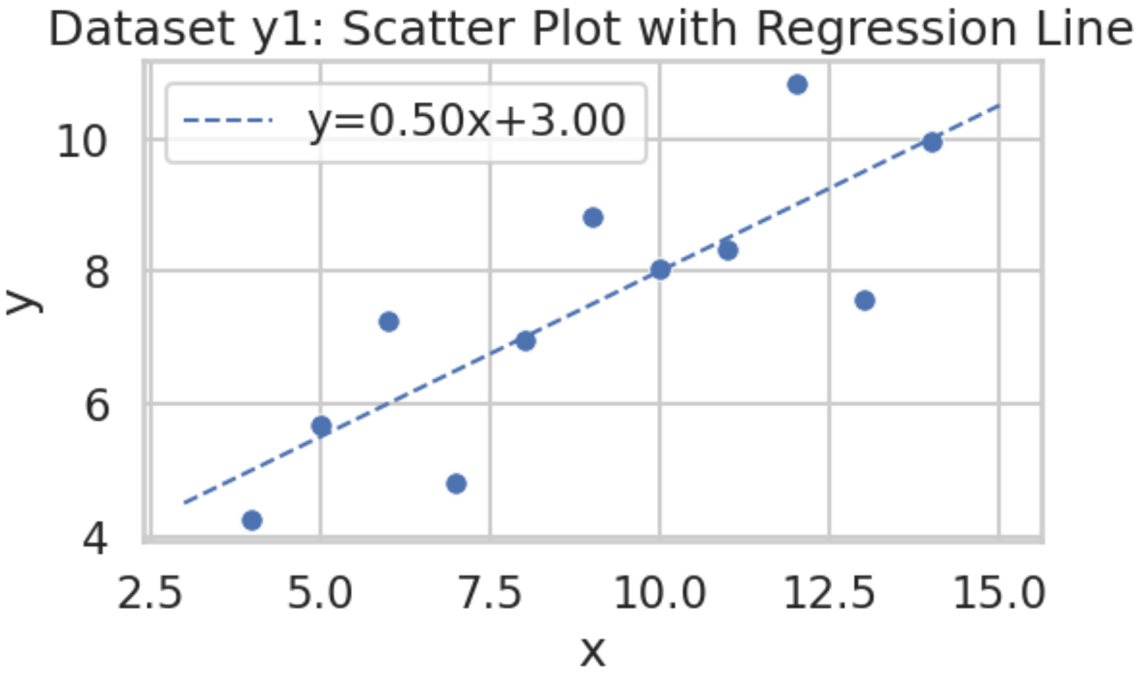
Below are plots that reveal how different the datasets truly are, despite having identical summary statistics.

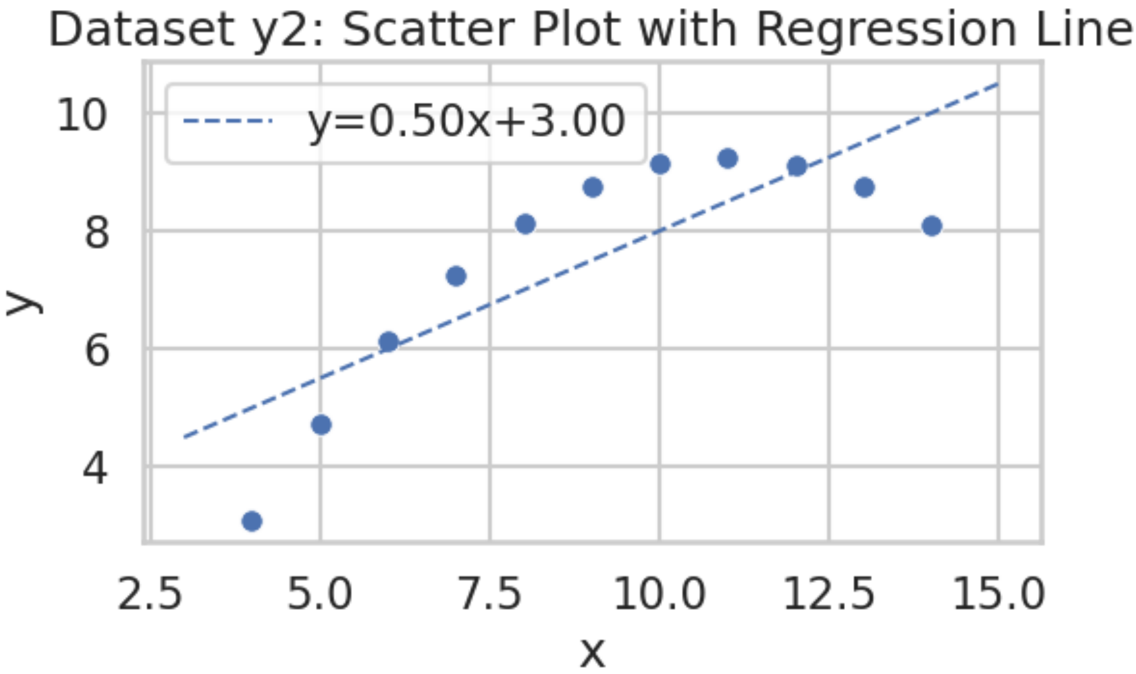
1. **Scatter plots with regression lines**
2. **Residual plots**
3. **Overlaid scatter plot for comparison**
4. **Violin plots** showing Y distribution across datasets

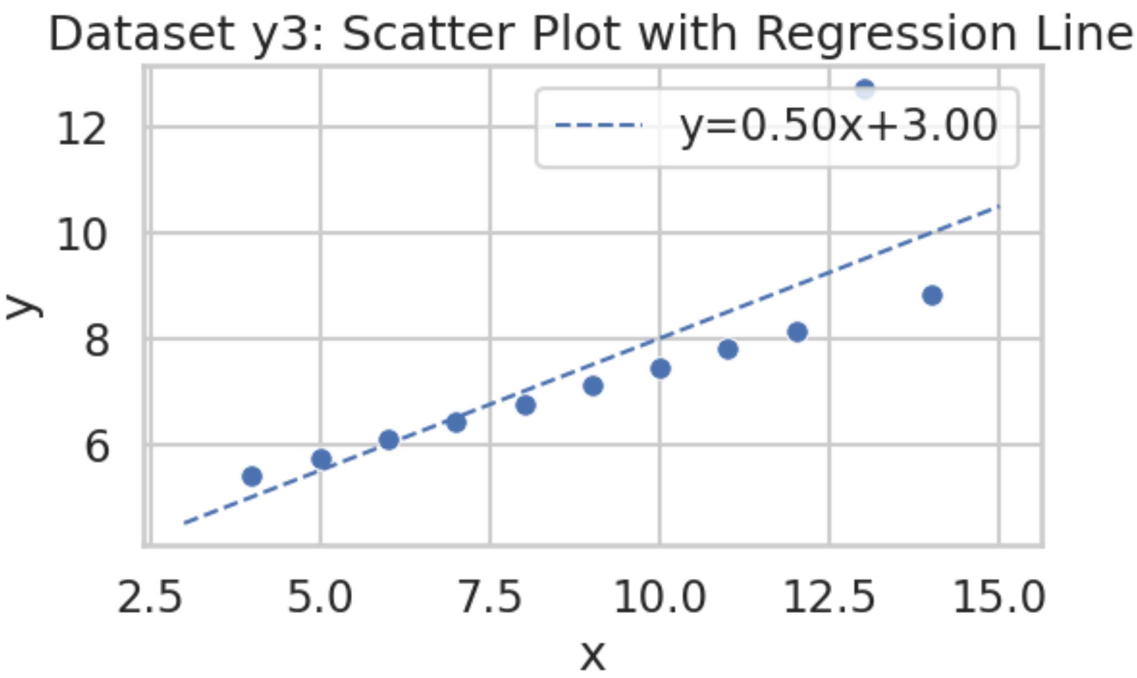
In [3]:

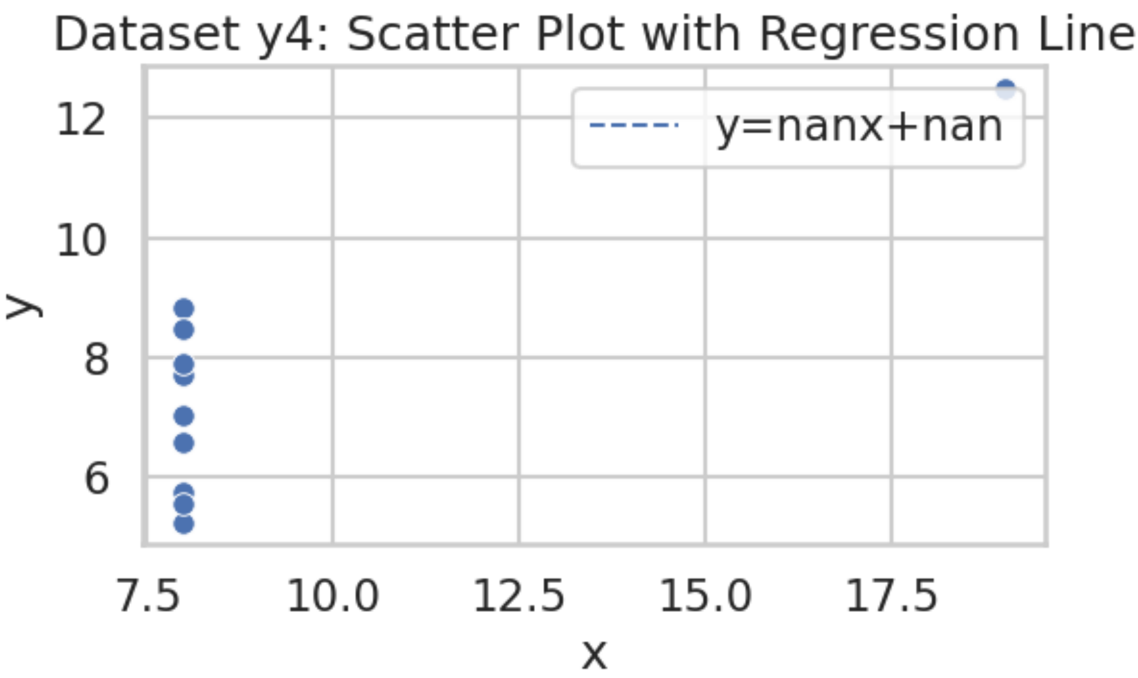
```
sns.set(style="whitegrid", context="talk")

# Scatter plots with regression line for each dataset
for name, group in df_melted.groupby('dataset'):
    plt.figure(figsize=(6, 4))
    sns.scatterplot(data=group, x='x', y='y', s=70)
    X = sm.add_constant(group['x'])
    model = sm.OLS(group['y'], X).fit()
    xs = np.linspace(group['x'].min() - 1, group['x'].max() + 1, 100)
    ys = model.params['const'] + model.params['x'] * xs
    plt.plot(xs, ys, '--', linewidth=1.5, label=f"y={model.params['x']:.2
    plt.title(f"Dataset {name}: Scatter Plot with Regression Line")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

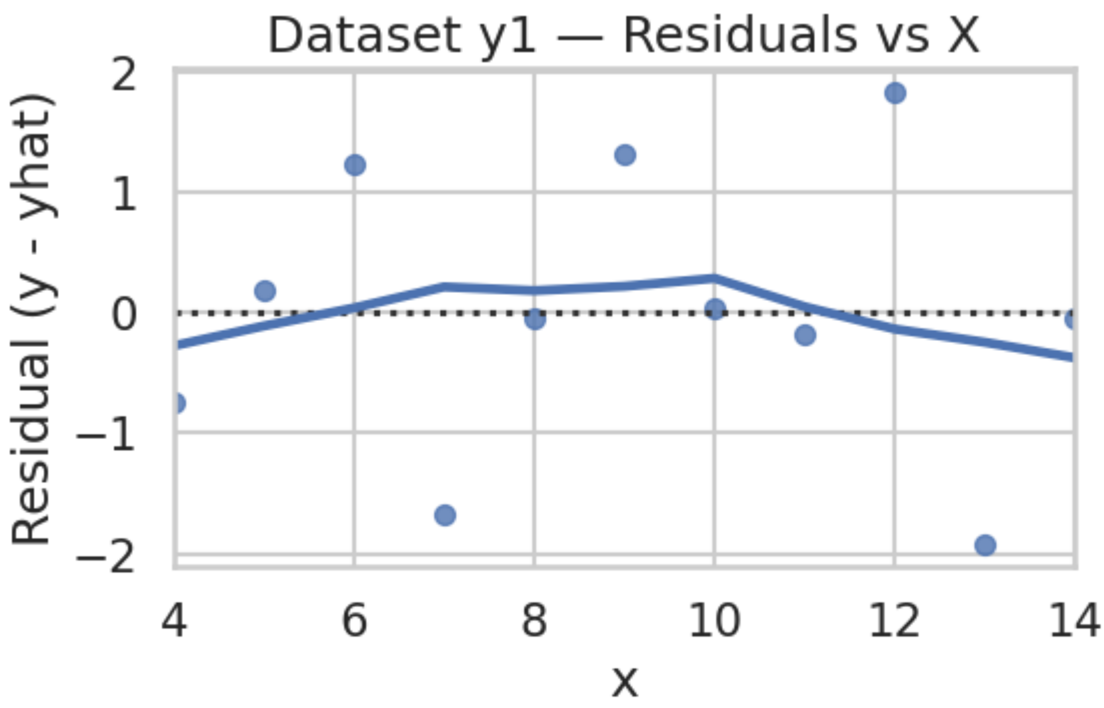

Dataset y1: Scatter Plot with Regression Line
y=0.50x+3.00
y
x
4
6
8
10
2.5
5.0
7.5
10.0
12.5
15.0

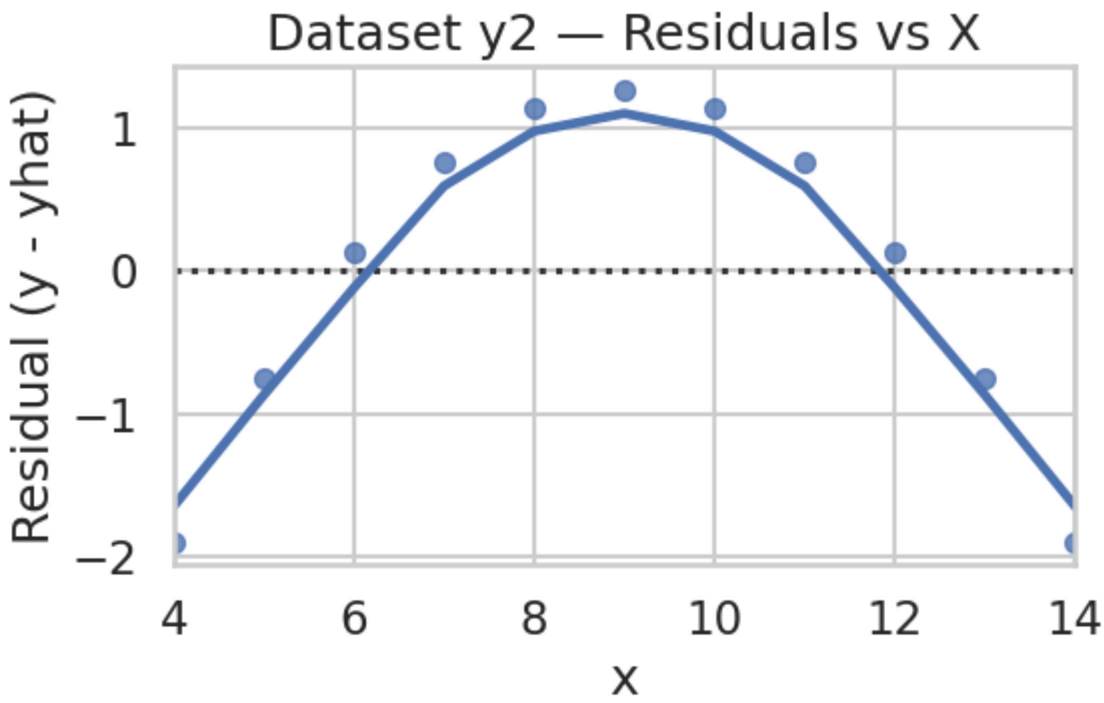

Dataset y2: Scatter Plot with Regression Line
y=0.50x+3.00
y
x
4
6
8
10
2.5
5.0
7.5
10.0
12.5
15.0

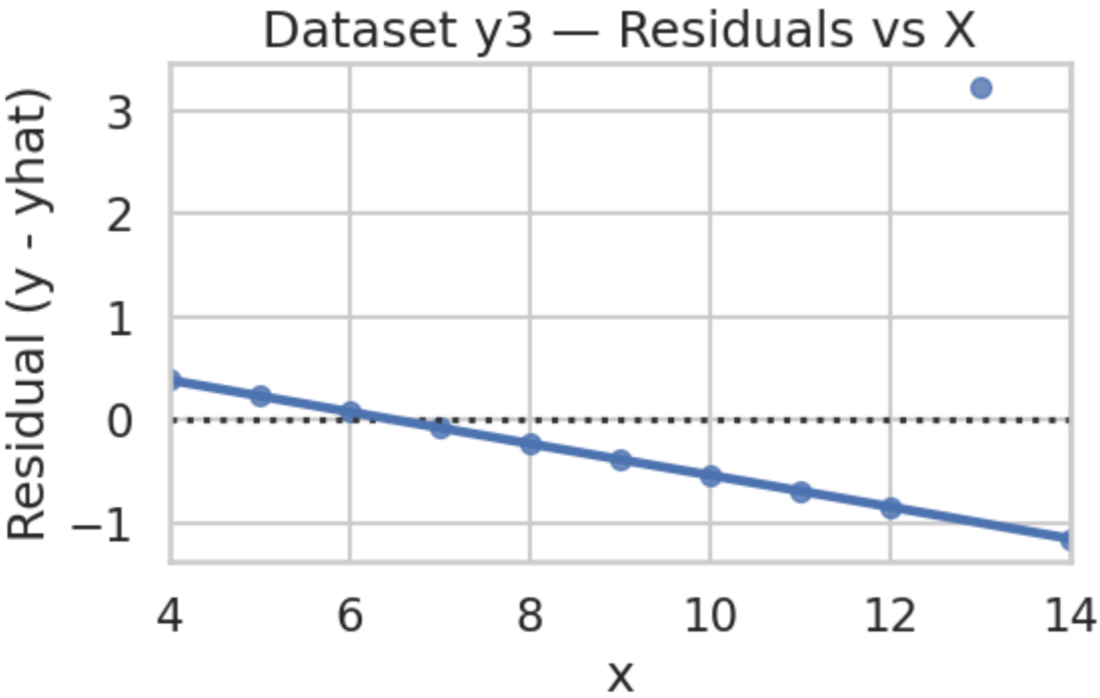




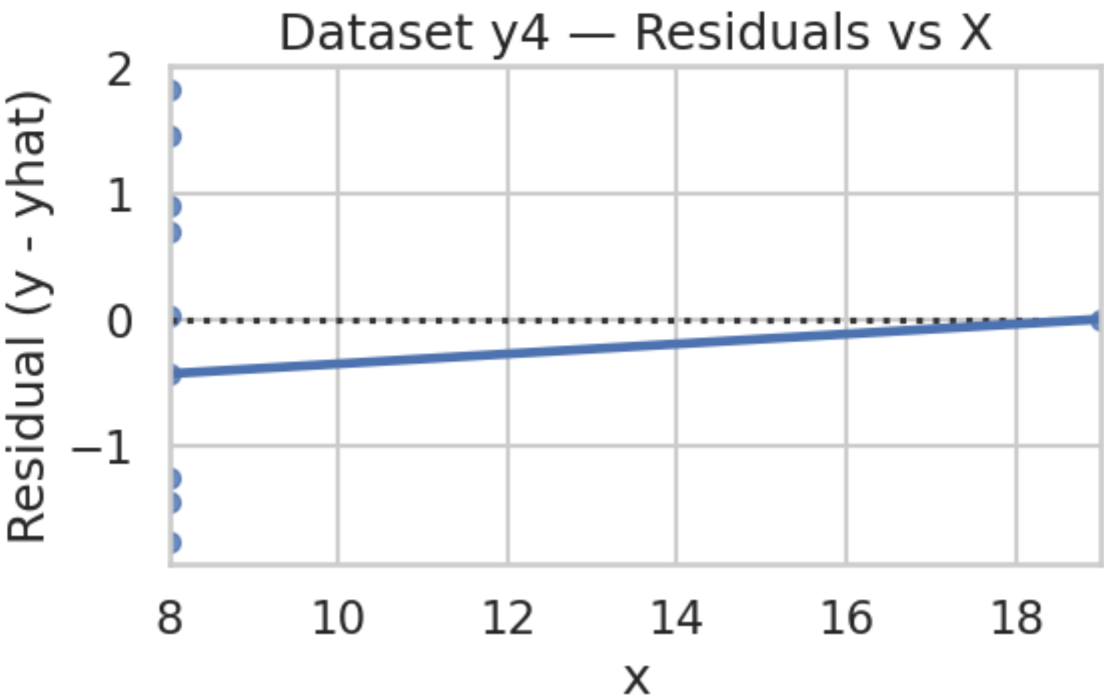
In [4]:
```
# Residual plots
for name, group in df_melted.groupby('dataset'):
    X = sm.add_constant(group['x'])
    model = sm.OLS(group['y'], X).fit()
    plt.figure(figsize=(6, 4))
    sns.residplot(x='x', y='y', data=group, lowess=True, scatter_kws={'s'
    plt.title(f"Dataset {name} — Residuals vs X")
    plt.xlabel("x")
    plt.ylabel("Residual (y - yhat)")
    plt.tight_layout()
    plt.show()
```


Dataset y1 — Residuals vs X
Residual (y - yhat)
2
1
0
−1
−2
4
6
8
10
12
14
x


Dataset y2 — Residuals vs X
Residual (y - yhat)
1
0
−1
−2
4
6
8
10
12
14
x



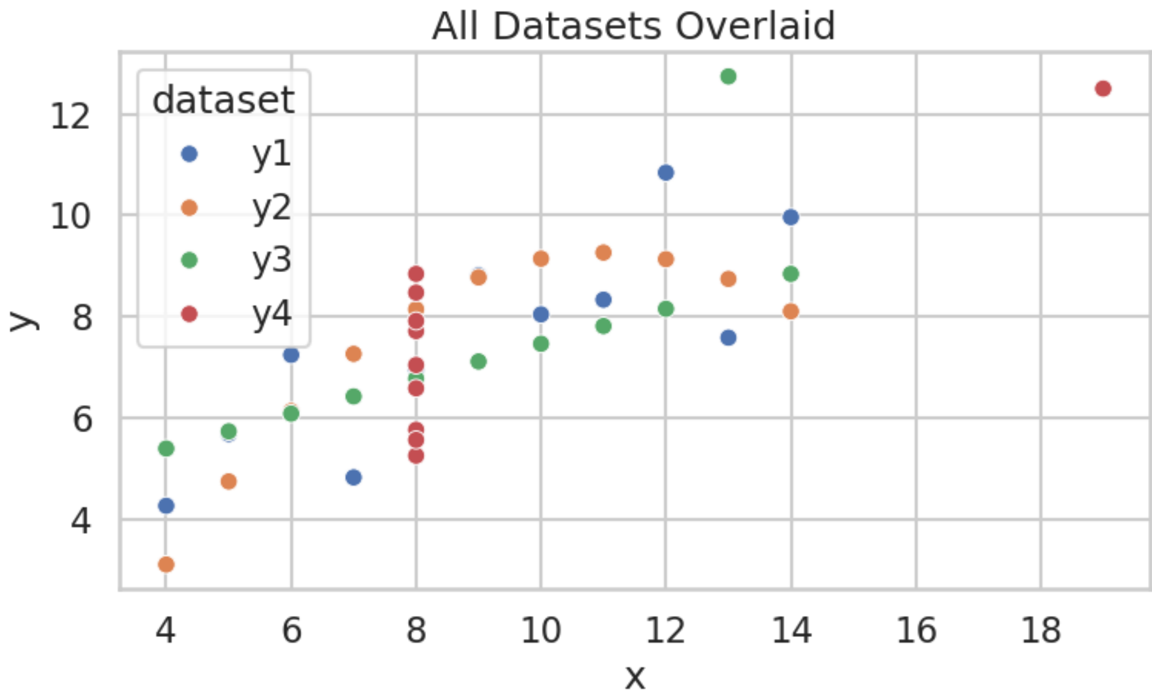

```
/home/avveer-singh-khurana/anscombe-project/venv/lib/python3.12/site-packa
ges/statsmodels/nonparametric/smoothers_lowess.py:226: RuntimeWarning: inv
alid value encountered in divide
  res, _ = _lowess(y, x, x, np.ones_like(x),
```



In [5]:
```
# Overlaid scatter comparison
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df_melted, x='x', y='y', hue='dataset', s=70)
plt.title("All Datasets Overlaid")
plt.xlabel("x")
plt.ylabel("y")
plt.tight_layout()
plt.show()

# Violin plot of Y distributions
plt.figure(figsize=(8, 5))
```

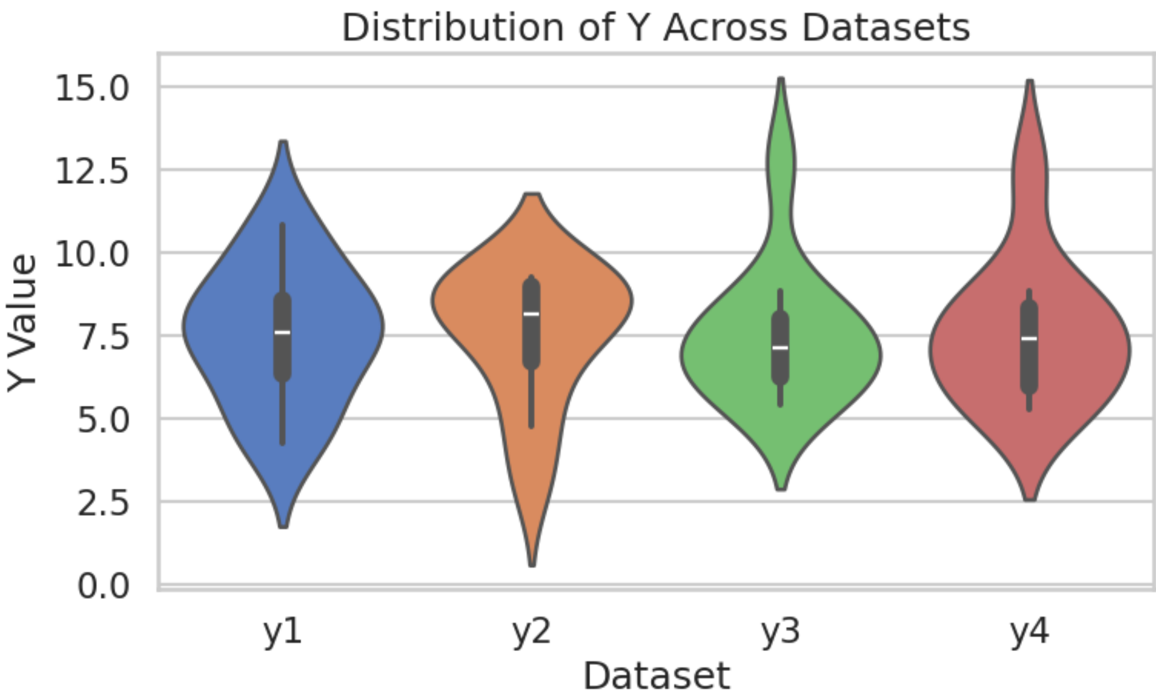
```
sns.violinplot(data=df_melted, x='dataset', y='y', palette='muted')
plt.title("Distribution of Y Across Datasets")
plt.xlabel("Dataset")
plt.ylabel("Y Value")
plt.tight_layout()
plt.show()
```




```
/tmp/ipykernel_13640/3771097717.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.violinplot(data=df_melted, x='dataset', y='y', palette='muted')
```



Interpretation

The summary statistics of all four datasets are nearly identical:

- Same mean and variance for x and y
- Same correlation (≈ 0.816)
- Same linear regression line and R^2

However, when plotted:

- Dataset I shows a normal linear relationship.
- Dataset II is clearly **nonlinear**.
- Dataset III has a single **outlier** that changes the line.
- Dataset IV has all points identical except one **extreme outlier**.

This proves that **summary statistics alone are insufficient** — visualization is essential for accurate data understanding.

Reproducibility & Code

The full notebook and source code are available here:
🔗 GitHub Repository

This notebook was created in **Python** using:

- pandas, numpy
- seaborn, matplotlib
- statsmodels

Collaboration Notes

This project was completed individually by **Avveer Singh Khurana**.
All code, analysis, and writing are original.

Appendix

Full code is included in this notebook.
For online access, visit the GitHub repository below:
🔗 https://github.com/AvveerSchoolAccount/anscombe-eda

All figures and plots were generated automatically from the Python code in this file.

In [ ]: