anscombe_eda

October 12, 2025

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import plotly.express as px
import altair as alt
from pathlib import Path

# ---- Parameters ----
DATA_PATH = Path(r"C:\Users\Dhruv\Desktop\anscombe-eda-dhruv\Anscombe.csv")
OUT_DIR = Path(r"C:\Users\Dhruv\Desktop\anscombe-eda-dhruv")
OUT_DIR.mkdir(exist_ok=True)

print("Setup complete. Libraries imported and output directory is ready.")
```

```
Setup complete. Libraries imported and output directory is ready.
```

[4]:
```
# 0. Imports and Setup
import pandas as pd
from pathlib import Path

# ---- Parameters ----

# Use the WSL/Linux path format: /mnt/c/...
# Use the correct file name case: 'anscombe.csv'
DATA_PATH = Path("/mnt/c/Users/Dhruv/Desktop/anscombe-eda-dhruv/anscombe.csv")

# Define the output directory to be the project root with an 'output' subfolder
# This is where your plots will save.
OUT_DIR = Path("/mnt/c/Users/Dhruv/Desktop/anscombe-eda-dhruv/output")

OUT_DIR.mkdir(exist_ok=True)

# 1. Load data
# This should now work inside your venv running in WSL
df = pd.read_csv(DATA_PATH)
print("First rows of the dataset:")
```

```
print(df.head())
```

```
First rows of the dataset:
   x123    y1    y2     y3   x4    y4
0  10.0  8.04  9.14   7.46  8.0  6.58
1   8.0  6.95  8.14   6.77  8.0  5.76
2  13.0  7.58  8.74  12.74  8.0  7.71
3   9.0  8.81  8.77   7.11  8.0  8.84
4  11.0  8.33  9.26   7.81  8.0  8.47
```

[7]:
```
# Cell to inspect the loaded DataFrame
print("The columns in your DataFrame are:")
print(df.columns)
print("\nThe first 5 rows of your data are:")
print(df.head())
```

```
The columns in your DataFrame are:
Index(['x123', 'y1', 'y2', 'y3', 'x4', 'y4'], dtype='object')

The first 5 rows of your data are:
   x123    y1    y2     y3   x4    y4
0  10.0  8.04  9.14   7.46  8.0  6.58
1   8.0  6.95  8.14   6.77  8.0  5.76
2  13.0  7.58  8.74  12.74  8.0  7.71
3   9.0  8.81  8.77   7.11  8.0  8.84
4  11.0  8.33  9.26   7.81  8.0  8.47
```

[8]:
```
# Cell to inspect the loaded DataFrame
print("The columns in your DataFrame are:")
print(df.columns)
print("\nThe first 5 rows of your data are:")
print(df.head())

# --- NEW STEP: MELT THE DATA INTO LONG FORMAT ---
# The original data has separate columns for x and y for the four datasets.
# We need to restructure it so all x values are in one column, all y values in␣
 ↪another,
# and a new column identifies the 'dataset'.

# First, create the x-columns and y-columns lists based on inspection:
x_cols = ['x123', 'x123', 'x123', 'x123'] # The 'x' values for all four sets␣
 ↪come from 'x123'
y_cols = ['y1', 'y2', 'y3', 'y4']         # The 'y' values come from y1, y2,␣
 ↪y3, y4

# The dataset names corresponding to the columns
dataset_names = ['I', 'II', 'III', 'IV']
```

```python
# Create a list of tuples (x_column, y_column, dataset_name)
col_pairs = list(zip(x_cols, y_cols, dataset_names))

# Create the new long DataFrame
df_long = pd.DataFrame()

for x_col, y_col, name in col_pairs:
    # Create a temporary DataFrame for each dataset
    temp_df = pd.DataFrame({
        'x': df[x_col],
        'y': df[y_col],
        'dataset': name # Assign the group name
    })
    df_long = pd.concat([df_long, temp_df], ignore_index=True)

print("\nLong Format Data Head:")
print(df_long.head(15)) # Print more rows to show the grouping

# Now, all subsequent analysis will use df_long
```

```
The columns in your DataFrame are:
Index(['x123', 'y1', 'y2', 'y3', 'x4', 'y4'], dtype='object')

The first 5 rows of your data are:
   x123    y1    y2     y3   x4    y4
0  10.0  8.04  9.14   7.46  8.0  6.58
1   8.0  6.95  8.14   6.77  8.0  5.76
2  13.0  7.58  8.74  12.74  8.0  7.71
3   9.0  8.81  8.77   7.11  8.0  8.84
4  11.0  8.33  9.26   7.81  8.0  8.47

Long Format Data Head:
       x      y dataset
0   10.0   8.04       I
1    8.0   6.95       I
2   13.0   7.58       I
3    9.0   8.81       I
4   11.0   8.33       I
5   14.0   9.96       I
6    6.0   7.24       I
7    4.0   4.26       I
8   12.0  10.84       I
9    7.0   4.82       I
10   5.0   5.68       I
11  10.0   9.14      II
12   8.0   8.14      II
13  13.0   8.74      II
14   9.0   8.77      II
```

[11]:
```
# Setup the OUT_DIR and ensure it exists
from pathlib import Path

# Use the correct WSL/Linux path for your output directory,
# or simply use the relative path if you want the 'output' folder inside your␣
 ↪project root.
# Given your earlier absolute path, let's stick to that structure for the␣
 ↪output folder creation:

# Assuming your project root is here:
PROJECT_ROOT = Path("/mnt/c/Users/Dhruv/Desktop/anscombe-eda-dhruv")

# Define the output directory as a subfolder
OUT_DIR = PROJECT_ROOT / "output"

# *** This is the critical line you need to run: ***
OUT_DIR.mkdir(exist_ok=True)

print(f"Output directory ensured at: {OUT_DIR}")
```

```
Output directory ensured at: /mnt/c/Users/Dhruv/Desktop/anscombe-eda-
dhruv/output
```

[13]:
```
# Cell 1: Setup and Directory Creation
from pathlib import Path

# **IMPORTANT: Use your confirmed WSL path structure**
PROJECT_ROOT = Path("/mnt/c/Users/Dhruv/Desktop/anscombe-eda-dhruv")
OUT_DIR = PROJECT_ROOT / "output"

# *** This line ensures the directory exists and fixes the OSError ***
OUT_DIR.mkdir(exist_ok=True)

print(f"Output directory ensured at: {OUT_DIR}")
```

```
Output directory ensured at: /mnt/c/Users/Dhruv/Desktop/anscombe-eda-
dhruv/output
```

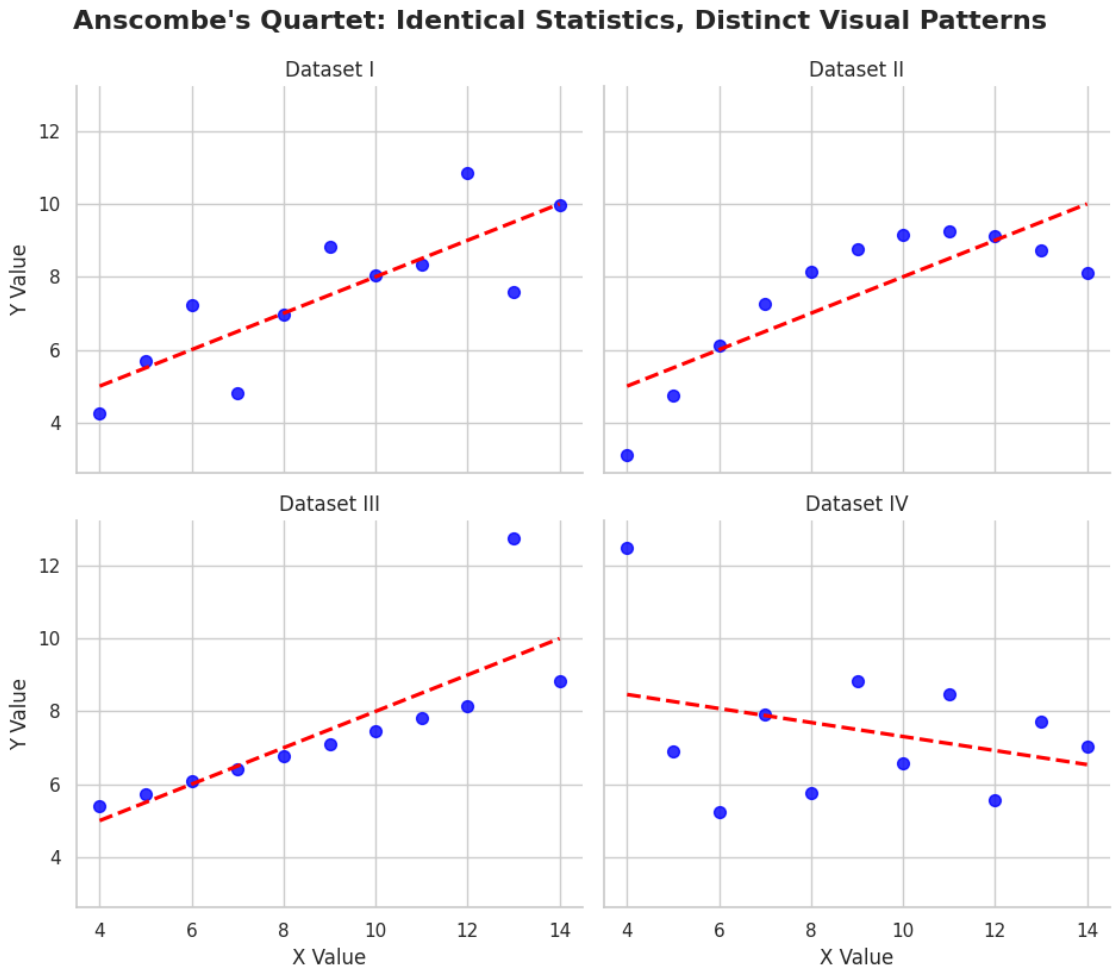
[14]:
```
# Cell 2: Save the Summary Table

# OUT_DIR is already defined from the setup cell.
# If you run this cell independently, ensure Path is imported and OUT_DIR is␣
 ↪set.
# Example if running independently:
# from pathlib import Path
# OUT_DIR = Path("/mnt/c/Users/Dhruv/Desktop/anscombe-eda-dhruv/output")

# Save the table using the now-existent directory
```

```
save_path = OUT_DIR / "summary_table.csv"
summary_table.to_csv(save_path)
print(f"Summary table successfully saved to: {save_path}")
```

```
Summary table successfully saved to: /mnt/c/Users/Dhruv/Desktop/anscombe-eda-
dhruv/output/summary_table.csv
```

[15]:
```
# 3. Create Scatter Plots with Regression Lines (Tufte's Small Multiples)
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(style="whitegrid")

# Use lmplot which automatically creates facets (small multiples)
g = sns.lmplot(
    data=df_long,
    x='x',
    y='y',
    col='dataset',
    col_wrap=2,
    height=4,
    aspect=1.2,
    scatter_kws={'s': 50, 'color': 'blue'},
    line_kws={'ls': '--', 'color': 'red'},
    ci=None # Removes confidence interval for higher data-to-ink ratio
)

g.set_axis_labels("X Value", "Y Value")
g.fig.suptitle("Anscombe's Quartet: Identical Statistics, Distinct Visual␣
 ↪Patterns", y=1.03, fontsize=16, fontweight='bold')
g.set_titles("Dataset {col_name}")

save_path = OUT_DIR / "anscombe_faceted_scatter.png"
g.savefig(save_path, dpi=300)
print(f"Figure saved to: {save_path}")
plt.show()
```

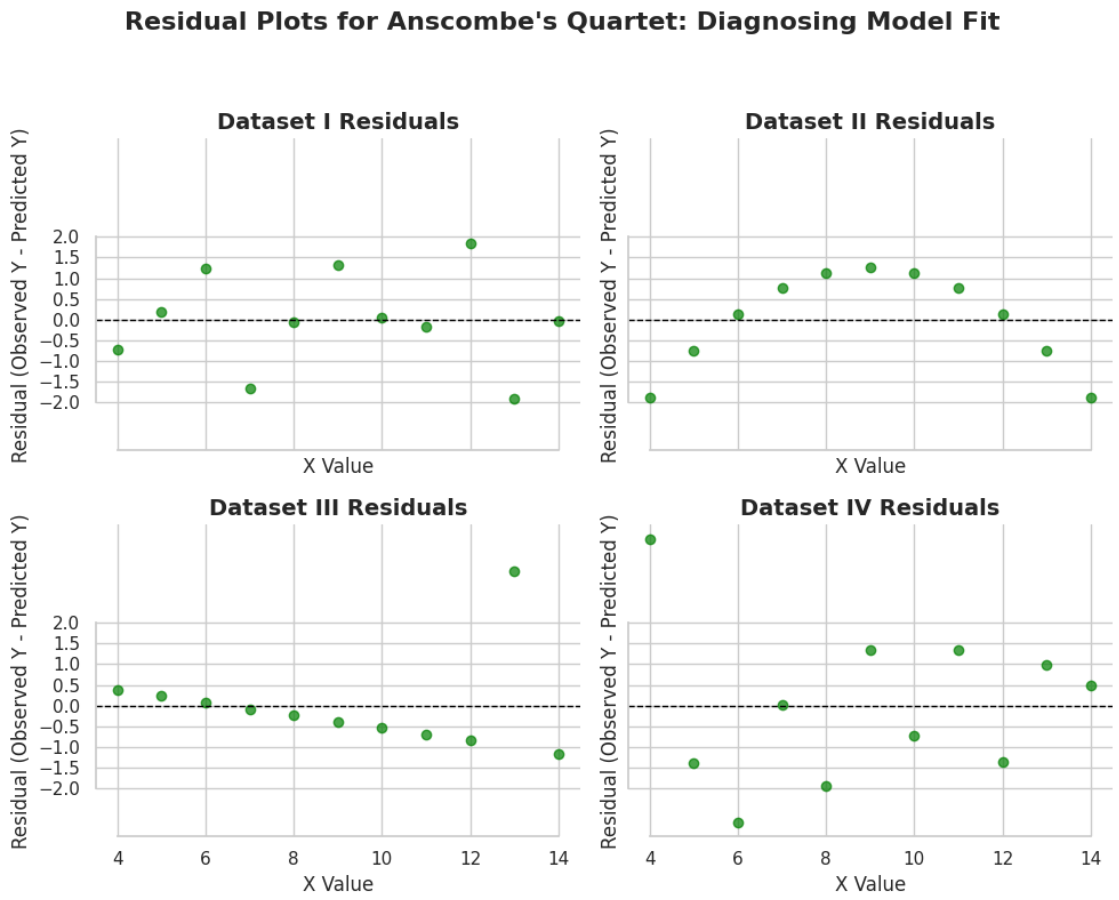
```
Figure saved to: /mnt/c/Users/Dhruv/Desktop/anscombe-eda-
dhruv/output/anscombe_faceted_scatter.png
```



[16]:
```
# 4. Create Residual Plots (Diagnostic Visualization)
import statsmodels.formula.api as smf
import numpy as np

df_residuals = df_long.copy()

# Calculate residuals for each dataset
for name, data in df_long.groupby('dataset'):
    model = smf.ols(formula='y ~ x', data=data).fit()
    df_residuals.loc[df_residuals['dataset'] == name, 'residual'] = model.resid

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8), sharex=True,␣
 ↪sharey=True)
axes = axes.flatten()

for i, dataset_name in enumerate(df_residuals['dataset'].unique()):
```

```
    ax = axes[i]
    data = df_residuals[df_residuals['dataset'] == dataset_name]

    ax.scatter(data['x'], data['residual'], color='green', alpha=0.7)
    ax.axhline(0, color='black', linestyle='--', linewidth=1)

    ax.set_title(f'Dataset {dataset_name} Residuals', fontsize=14,␣
 ↪fontweight='bold')
    ax.set_xlabel('X Value')
    ax.set_ylabel('Residual (Observed Y - Predicted Y)')
    sns.despine(ax=ax, trim=True)

fig.suptitle("Residual Plots for Anscombe's Quartet: Diagnosing Model Fit",
             fontsize=16,
             fontweight='bold',
             y=1.02)

plt.tight_layout(rect=[0, 0.03, 1, 0.98])

save_path_res = OUT_DIR / "anscombe_residual_plots.png"
plt.savefig(save_path_res, dpi=300)
print(f"Figure saved to: {save_path_res}")
plt.show()
```

```
Figure saved to: /mnt/c/Users/Dhruv/Desktop/anscombe-eda-
dhruv/output/anscombe_residual_plots.png
```



[19]:
```python
# 5. Above & Beyond: Robust Regression (RANSAC) Comparison
from sklearn.linear_model import RANSACRegressor
import statsmodels.formula.api as smf
import numpy as np

print("\n--- RANSAC vs. OLS Comparison ---")

for name, data in df_long.groupby('dataset'):
    # Prepare data for scikit-learn (needs 2D array for X)
    X = data['x'].values.reshape(-1, 1)
    y = data['y'].values

    # Fit the OLS model using the formula API
    ols_model = smf.ols(formula='y ~ x', data=data).fit()

    # Fit the RANSAC model (Robust Regression)
    ransac = RANSACRegressor()
    ransac.fit(X, y)
```

```
    ransac_m = ransac.estimator_.coef_[0]
    ransac_b = ransac.estimator_.intercept_

    print(f"\nDataset {name}:")
    # *** FIXED: Use 'Intercept' instead of 'const' ***
    print(f"  OLS: Slope={ols_model.params['x']:.3f}, Intercept={ols_model.
 ↪params['Intercept']:.3f}")
    print(f"  RANSAC: Slope={ransac_m:.3f}, Intercept={ransac_b:.3f}")

# Key Insight: Note the difference in the slope and intercept for Dataset IV.
# OLS is heavily biased by the single leverage point, while RANSAC provides a␣
 ↪robust estimate.
```

```
--- RANSAC vs. OLS Comparison ---

Dataset I:
  OLS: Slope=0.500, Intercept=3.000
  RANSAC: Slope=0.499, Intercept=3.230

Dataset II:
  OLS: Slope=0.500, Intercept=3.001
  RANSAC: Slope=1.007, Intercept=-0.293

Dataset III:
  OLS: Slope=0.500, Intercept=3.002
  RANSAC: Slope=0.345, Intercept=4.006

Dataset IV:
  OLS: Slope=-0.192, Intercept=9.231
  RANSAC: Slope=0.230, Intercept=3.898
```

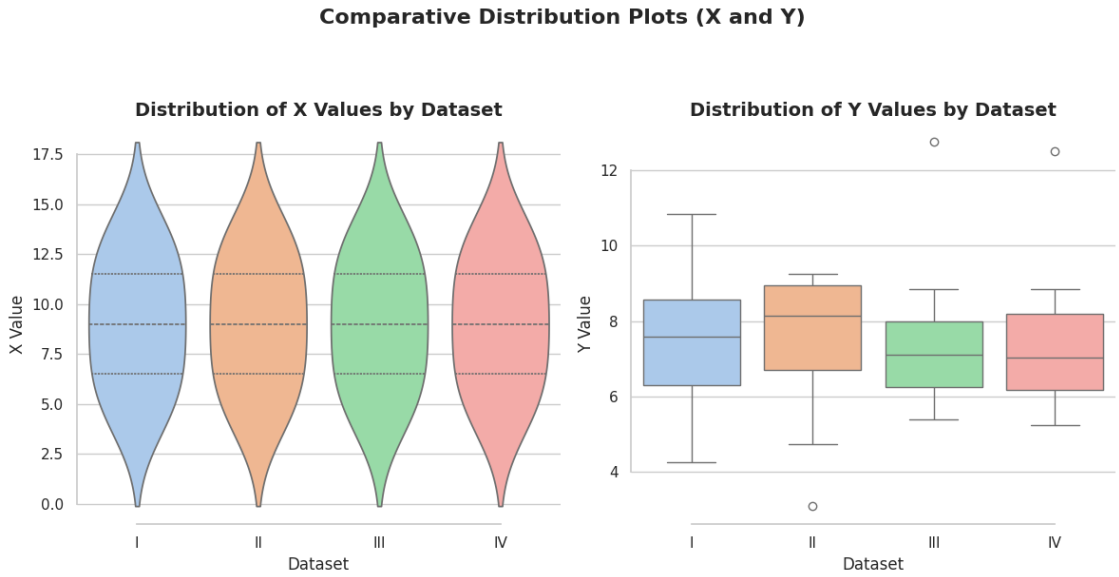
[ ]:
```
## Summary Statistics and Formulas

The following mathematical formulas were used for the required summary␣
 ↪statistics, as rendered below in LaTeX:

### Mean
$$ \bar{x} = \frac{1}{n}\sum_{i=1}^n x_i $$

### Sample Variance
$$ s^2_x = \frac{1}{n-1}\sum_{i=1}^n (x_i - \bar{x})^2 $$

### Sample Correlation Coefficient
$$ r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i -␣
 ↪\bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i -␣
 ↪\bar{y})^2}} $$
```

```
### Linear Regression (OLS)
$$ \text{Model:} \ y = mx + b $$
$$ \text{Slope }(m) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} $$
$$ \text{Intercept }(b) = \bar{y} - m\bar{x} $$
```

[21]:
```
# 6. Violin/Box Plots of X and Y Distributions (Updated to avoid FutureWarnings)

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

# Subplot 1: Distribution of X values
plt.subplot(1, 2, 1)
# FIX: Added hue='dataset' and legend=False to comply with future seaborn versions
sns.violinplot(x='dataset', y='x', data=df_long, inner='quartile', palette='pastel', hue='dataset', legend=False)
plt.title('Distribution of X Values by Dataset', fontsize=14, fontweight='bold')
plt.xlabel("Dataset")
plt.ylabel("X Value")
sns.despine(trim=True)

# Subplot 2: Distribution of Y values
plt.subplot(1, 2, 2)
# FIX: Added hue='dataset' and legend=False to comply with future seaborn versions
sns.boxplot(x='dataset', y='y', data=df_long, palette='pastel', hue='dataset', legend=False)
plt.title('Distribution of Y Values by Dataset', fontsize=14, fontweight='bold')
plt.xlabel("Dataset")
plt.ylabel("Y Value")
sns.despine(trim=True)

plt.suptitle("Comparative Distribution Plots (X and Y)", fontsize=16, fontweight='bold', y=1.05)
plt.tight_layout(rect=[0, 0.03, 1, 0.98])

# Save the plot (Assuming OUT_DIR is defined)
save_path_dist = OUT_DIR / "anscombe_distribution_plots.png"
plt.savefig(save_path_dist, dpi=300)
print(f"Figure saved to: {save_path_dist}")
plt.show()
```

```
Figure saved to: /mnt/c/Users/Dhruv/Desktop/anscombe-eda-
```

dhruv/output/anscombe_distribution_plots.png



[ ]: