Chapter 4: Statistical Modelling

Regression Analysis

Prepared by Dr. Priyanga D. Talagala (Copyright 2025 P. D. Talagala)

14-5-2025

Simple Linear Regression Analysis

The data in the table are the twelve observations corresponding to concert attendance (in thousand) and total worldwide CD sales by the performing artist (or band) in the previous year (also in thousand). Draw a scatter diagram for the data and determine by inspection if there exists an approximate linear relationship between X and Y

Number of CD sales ('000)	Concert attendance ('000)
61	4.280
63	4.080
67	4.420
69	4.170
70	4.480
74	4.300
76	4.820
81	4.700
86	5.110
91	5.130
95	5.640
97	5.560

You may then use the code below in order to create the DataFrame for our example:

In [10]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns


data = {'CDsale': [61,63,67,69,70,74,76,81, 86,91,95,97],
```

```
        'attendance': [4.28,4.08,4.42,4.17,4.48,4.3,4.82,4.7,5.11,5.13,5.64,5.56]
        }

df = pd.DataFrame(data)

print (df)
```

```
    CDsale  attendance
0       61        4.28
1       63        4.08
2       67        4.42
3       69        4.17
4       70        4.48
5       74        4.30
6       76        4.82
7       81        4.70
8       86        5.11
9       91        5.13
10      95        5.64
11      97        5.56
```

Correlation analysis

In [11]:
```
correlation_df = df.corr()
print(correlation_df)
```

```
            CDsale  attendance
CDsale      1.0000      0.9482
attendance  0.9482      1.0000
```

If you apply .corr directly to your dataframe, it will return all pairwise correlations between your columns.

That's why you then observe 1s at the diagonal of your matrix (each column is perfectly correlated with itself).

In [12]:
```
df['CDsale'].corr(df['attendance'])  #x.corr(y)
```

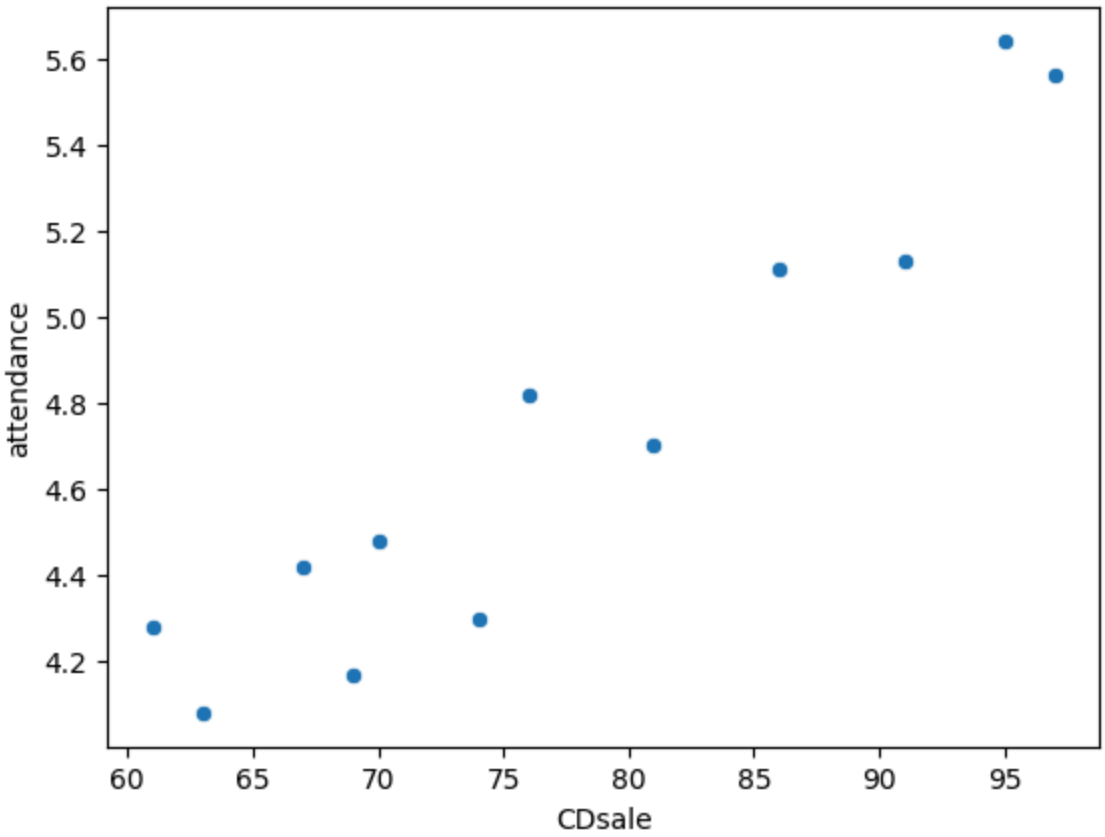
Out[12]:
```
np.float64(0.9482003279477094)
```

In [13]:
```
# Alternative way
x = df['CDsale']
y = df['attendance']
x.corr(y)
```

Out[13]:
```
np.float64(0.9482003279477094)
```

In [14]:
```
import seaborn as sns
sns.scatterplot(data=df, x="CDsale", y="attendance")
```

Out[14]:
```
<Axes: xlabel='CDsale', ylabel='attendance'>
```



Regression Analysis with Python

- Two popular options are scikit-learn and StatsModels.
- The differences between them highlight what each in particular has to offer
  - scikit-learn's other popular topics are machine-learning and data-science
  - StatsModels are econometrics, generalized-linear-models, timeseries-analysis, and regression-models.

Method 1 : Using StatsModels

In [29]:
```
# With one column name, single pair of brackets returns a Series,
# while double brackets return a dataframe.

X = df[["CDsale"]]
y = df[["attendance"]]
X1 = sm.add_constant(X) # adds a column of ones to the X array
```

About add_constant

This column of ones corresponds to x_0 in the simple linear regression equation:
$\hat{y} = \beta_0 * X_0 + \beta_1 * x_1$. X_0 is a column of ones.

By default, statsmodels.OLS does not add an intercept (constant) term to the model. Then we fit a regression line that goes through the origin (0,0), which may be statistically inappropriate unless you have a theoretical reason to assume that.

In [7]:
```
X
```

Out[7]:

	CDsale
0	61
1	63
2	67
3	69
4	70
5	74
6	76
7	81
8	86
9	91
10	95
11	97

In [8]:
```
X1
```

Out[8]:

	const	CDsale
0	1.0	61
1	1.0	63
2	1.0	67
3	1.0	69
4	1.0	70
5	1.0	74
6	1.0	76
7	1.0	81
8	1.0	86
9	1.0	91
10	1.0	95
11	1.0	97

In [9]:
```
# Fit a linear regression model
model = sm.OLS(y, X1)
result = model.fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             attendance   R-squared:                       0.899
Model:                            OLS   Adj. R-squared:                  0.889
Method:                 Least Squares   F-statistic:                     89.09
Date:                Tue, 13 May 2025   Prob (F-statistic):           2.69e-06
Time:                        22:46:27   Log-Likelihood:                 4.8312
No. Observations:                  12   AIC:                            -5.662
Df Residuals:                      10   BIC:                            -4.693
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.5698      0.338      4.643      0.001       0.816       2.323
CDsale         0.0407      0.004      9.439      0.000       0.031       0.050
==============================================================================
Omnibus:                        1.556   Durbin-Watson:                   2.962
Prob(Omnibus):                  0.459   Jarque-Bera (JB):                0.872
Skew:                          -0.253   Prob(JB):                        0.647
Kurtosis:                       1.780   Cond. No.                         518.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```

```
C:\Users\B1500\AppData\Local\Programs\Python\Python313\Lib\site-packages\scipy\stats
\_axis_nan_policy.py:430: UserWarning: `kurtosistest` p-value may be inaccurate with
fewer than 20 observations; only n=12 observations were given.
  return hypotest_fun_in(*args, **kwds)
```

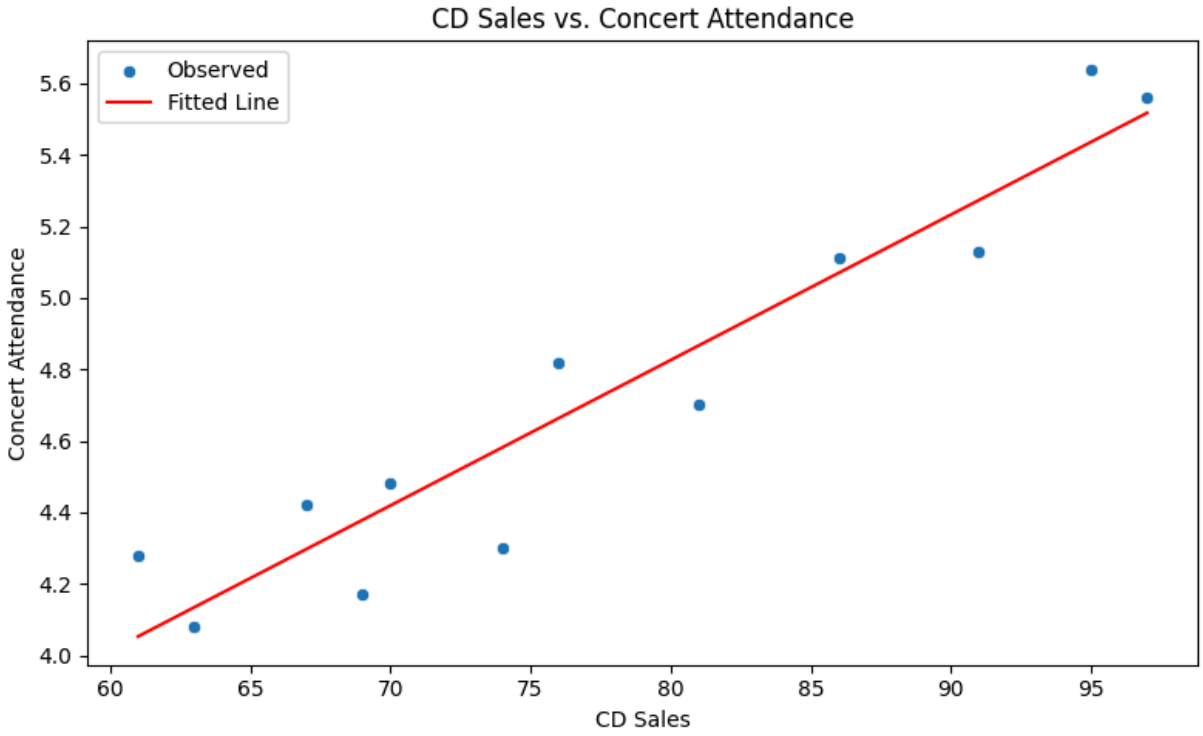
In [16]:
```
df['predicted'] = result.fittedvalues
df['residual'] = result.resid

df
```

Out[16]:

	CDsale	attendance	predicted	residual
0	61	4.28	4.052590	0.227410
1	63	4.08	4.133993	-0.053993
2	67	4.42	4.296800	0.123200
3	69	4.17	4.378203	-0.208203
4	70	4.48	4.418905	0.061095
5	74	4.30	4.581711	-0.281711
6	76	4.82	4.663114	0.156886
7	81	4.70	4.866622	-0.166622
8	86	5.11	5.070130	0.039870
9	91	5.13	5.273638	-0.143638
10	95	5.64	5.436445	0.203555
11	97	5.56	5.517848	0.042152

In [22]:
```
plt.figure(figsize=(8, 5))
sns.scatterplot(x='CDsale', y='attendance', data=df, label='Observed')
sns.lineplot(x='CDsale', y='predicted', data=df, color='red', label='Fitted Line')
plt.title('CD Sales vs. Concert Attendance')
plt.xlabel('CD Sales')
plt.ylabel('Concert Attendance')
plt.legend()
plt.tight_layout()
plt.show()
```



In [23]:
```
residual_sum  = df['residual'].sum()
residual_sum
```

Out[23]: np.float64(-3.6415315207705135e-14)

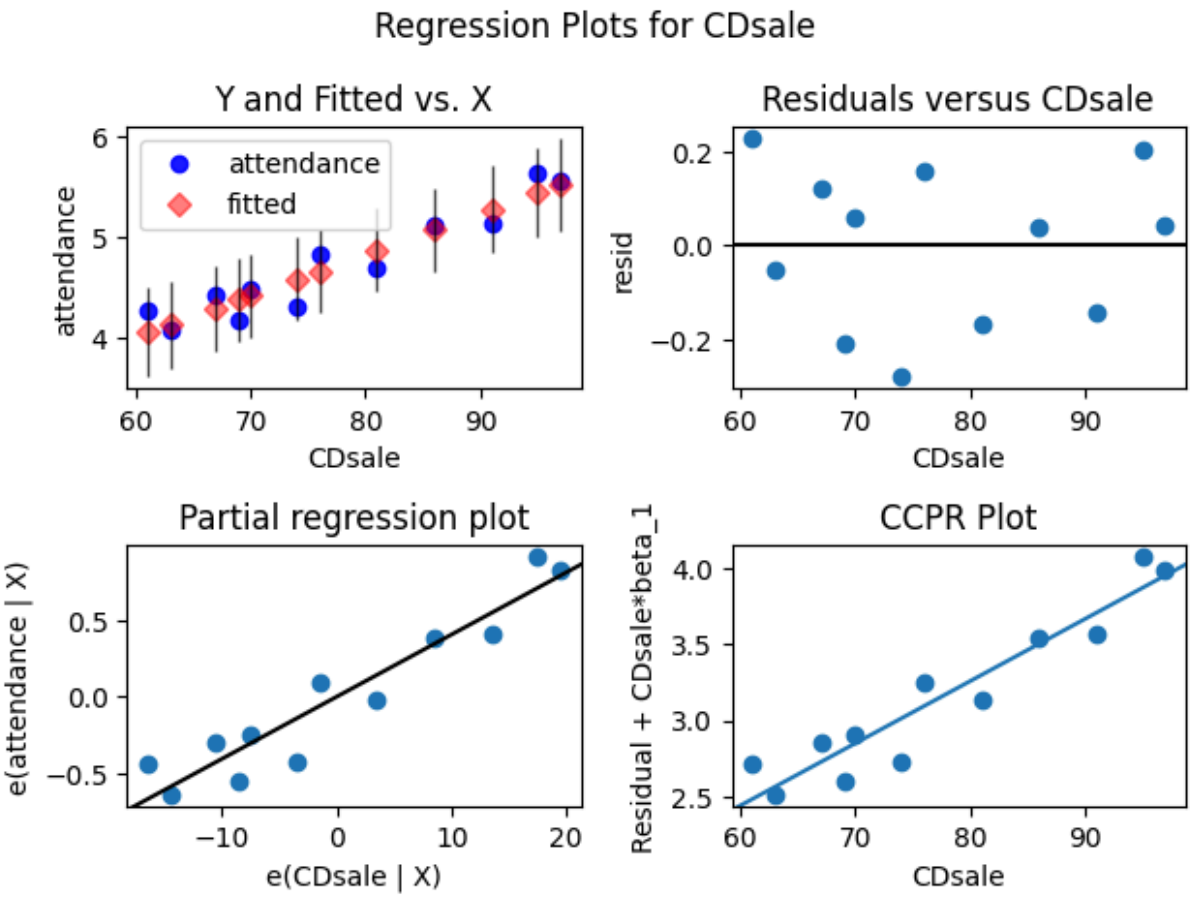
In theory, the sum of residuals in an ordinary least squares (OLS) regression model that includes an intercept is exactly zero. This is a fundamental property of OLS, ensuring that the regression line balances the observed data points above and below the line.

However, in practice, the computed sum of residuals may not be exactly zero due to floating-point arithmetic limitations in computer calculations. These small deviations arise from rounding errors inherent in numerical computations and are typically negligible.

In [20]:
```
print(f"Sum of residuals (rounded): {round(residual_sum, 10)}")
```

```
Sum of residuals (rounded): -0.0
```

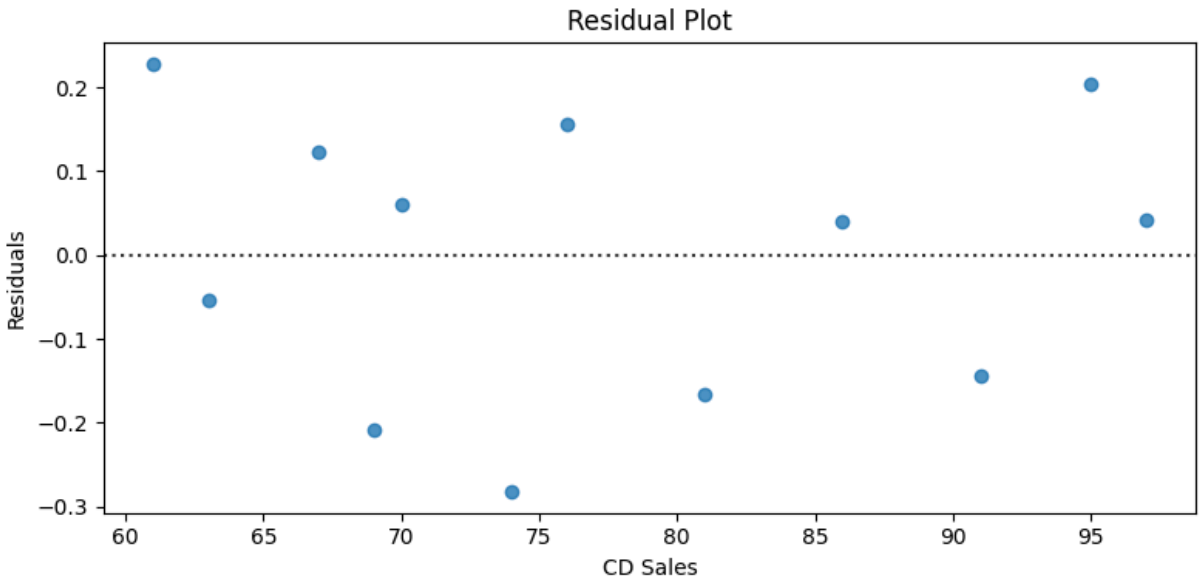
In [52]:
```
# Generate regression diagnostics plots
fig = sm.graphics.plot_regress_exog(result, 'CDsale')
plt.tight_layout()
plt.show()
```



Residual plots

- Residuals are the difference between the dependent variable (y) and the predicted variable (y_predicted).
- The residual plot is a type of graph in which the residuals for a particular regression model are plotted along with their associated value of x and an ordered pair $(x, y - \hat{y})$.
- Residual plots are more meaningful with larger sample sizes.
- For small sample sizes, residual plot analyses can be problematic and subject to over-interpretation.
- this example contains only 12 pairs of data.
- Therefore, one should be cautious in reaching conclusions from the Residual plots.
- Visual inspection of these residual plots will let you know if you have bias in your independent variables and thus are breaking either the constant variance or independent assumptions of regression analysis.

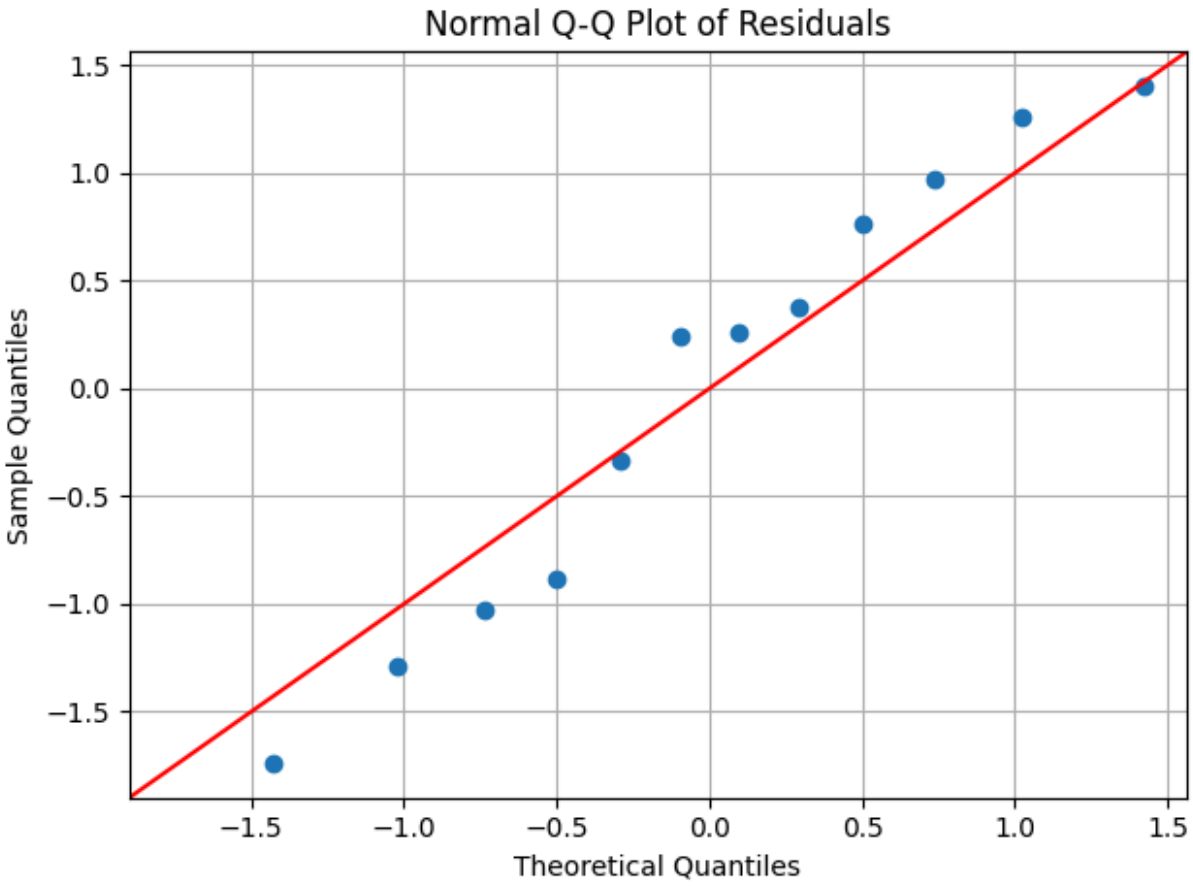
In [25]:
```
plt.figure(figsize=(8, 4))
sns.residplot(x='CDsale', y='attendance', data=df)
plt.title('Residual Plot')
plt.xlabel('CD Sales')
plt.ylabel('Residuals')
plt.tight_layout()
plt.show()
```



Normal Q–Q (quantile-quantile) Plot

- Residuals should be normally distributed.
- A normal quantile plot (also known as a quantile-quantile plot or QQ plot) is a graphical way of checking whether your data are normally distributed.
- If residuals follow close to a straight line on this plot, it is a good indication they are normally distributed.

In [26]:
```
sm.qqplot(result.resid, line='45', fit=True)
plt.title("Normal Q-Q Plot of Residuals")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Sample Quantiles")
plt.grid(True)
plt.tight_layout()
plt.show()
```



For our model, the Q-Q plot shows pretty good alignment to the the line with a few points at the middle slightly offset. Probably not significant and a reasonable alignment.

Method 2: Using Sklearn

Let's see how we can come up with a regression model using the popular python package for machine learning, Sklearn.

In [36]:
```
# import sys
# !{sys.executable} -m pip install scikit-learn
```

In [38]:
```
from sklearn import linear_model as lm
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

X = df[["CDsale"]]
y = df[["attendance"]]
# Fit the linear model
model = lm.LinearRegression()
results = model.fit(X, y)
```

When using **sklearn**'s LinearRegression, you do not need to manually add an intercept by adding a column of ones to your input data (like sm.add_constant in statsmodels).

sklearn's LinearRegression automatically includes an intercept by default when fitting the model. The intercept is estimated alongside the coefficients during the fitting process.

In [40]:
```
df['predicted'] = model.predict(X)
df['residual'] = y.values.flatten() - df['predicted']

df
```

Out[40]:

	CDsale	attendance	predicted	residual
0	61	4.28	4.052590	0.227410
1	63	4.08	4.133993	-0.053993
2	67	4.42	4.296800	0.123200
3	69	4.17	4.378203	-0.208203
4	70	4.48	4.418905	0.061095
5	74	4.30	4.581711	-0.281711
6	76	4.82	4.663114	0.156886
7	81	4.70	4.866622	-0.166622
8	86	5.11	5.070130	0.039870
9	91	5.13	5.273638	-0.143638
10	95	5.64	5.436445	0.203555
11	97	5.56	5.517848	0.042152

In [44]:
```
model.coef_
```

Out[44]:
```
array([[0.0407016]])
```

In [46]:
```
# Extract model coefficients
intercept = model.intercept_[0]
coefficient = model.coef_[0][0]
print("Intercept:", intercept)
print("Coefficient for CDsale:", coefficient)
```

```
Intercept: 1.5697927767910014
Coefficient for CDsale: 0.04070159857904085
```

sklearn does not provide a built-in .summary() like statsmodels, but you can manually compute metrics like R^2 or MSE if needed.

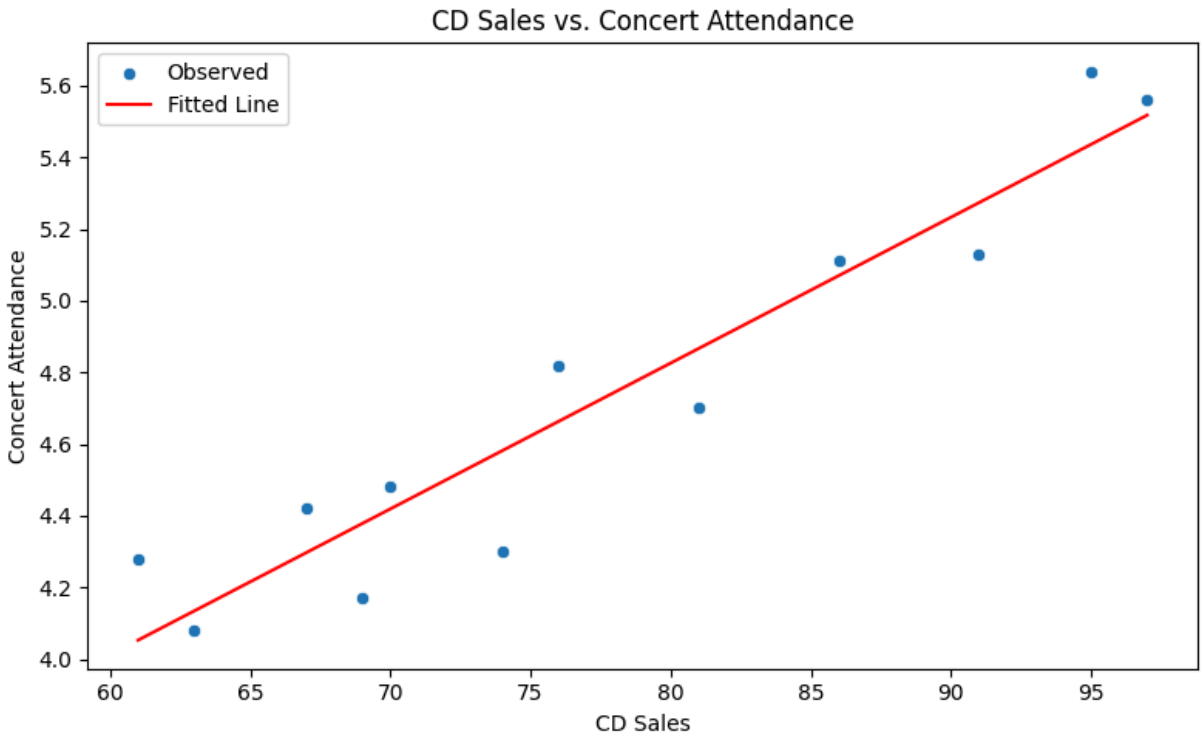
In [48]:
```
# Evaluation metrics
r2 = r2_score(y, df['predicted'])
mse = mean_squared_error(y, df['predicted'])
rmse = np.sqrt(mse)
mae = mean_absolute_error(y, df['predicted'])
```

```
print("\nModel Evaluation Metrics:")
print(f"R²: {r2:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"MAE: {mae:.4f}")
```

```
Model Evaluation Metrics:
R²: 0.8991
MSE: 0.0262
RMSE: 0.1618
MAE: 0.1424
```

In general, the higher the R-squared, the better the model fits your data

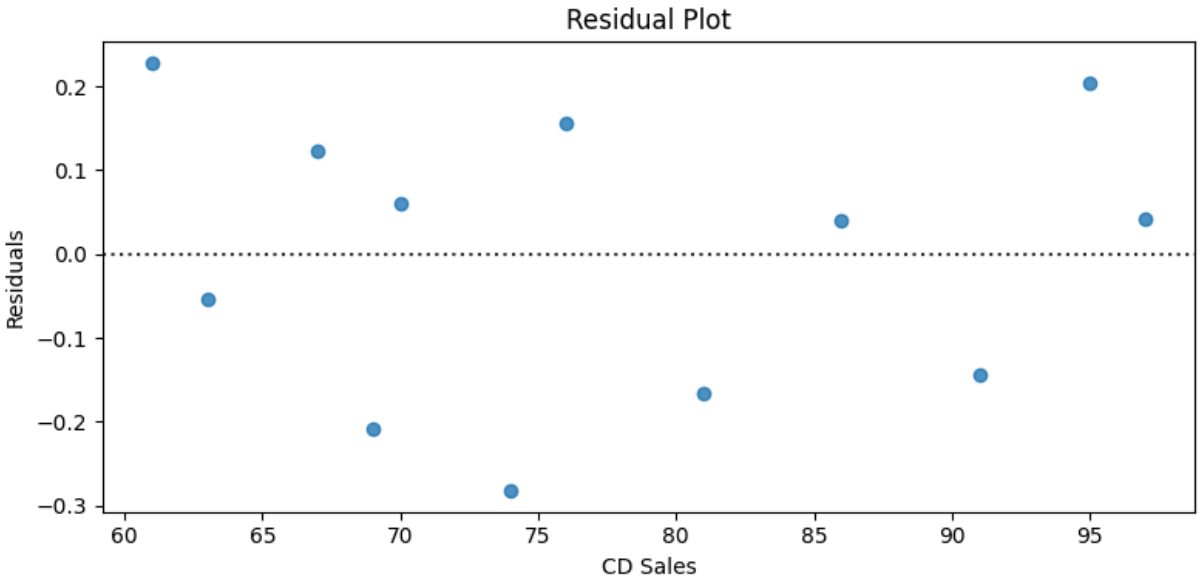
In [49]:
```
plt.figure(figsize=(8, 5))
sns.scatterplot(x='CDsale', y='attendance', data=df, label='Observed')
sns.lineplot(x='CDsale', y='predicted', data=df, color='red', label='Fitted Line')
plt.title("CD Sales vs. Concert Attendance")
plt.xlabel("CD Sales")
plt.ylabel("Concert Attendance")
plt.legend()
plt.tight_layout()
plt.show()
```



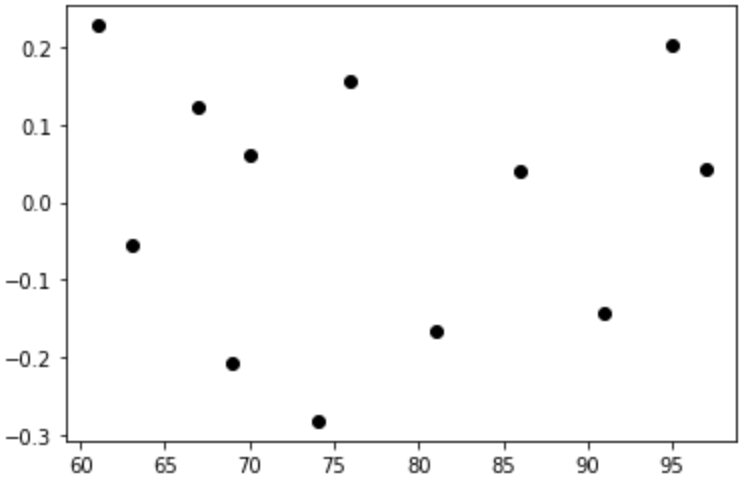
Residual Plots

scikit-learn is a modeling-focused library. It focuses on fitting, predicting, and scoring models — not on detailed diagnostic plots. So you need to compute residuals manually and use matplotlib or seaborn to plot them.

In [51]:
```
plt.figure(figsize=(8, 4))
sns.residplot(x='CDsale', y='attendance', data=df)
plt.title("Residual Plot")
plt.xlabel("CD Sales")
plt.ylabel("Residuals")
plt.tight_layout()
plt.show()
```

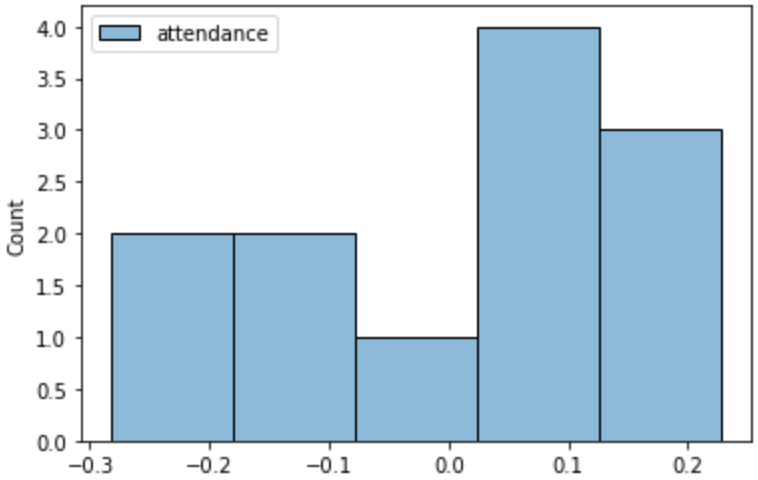


In [65]:
```
res = y - y_predicted
plt.plot(X, res, 'o', color='black');
```



In [67]:
```
sns.histplot(res)
```

Out[67]:
```
<AxesSubplot:ylabel='Count'>
```



In predictive modeling, particularly with libraries like scikit-learn, residual analysis can be skipped if the goal is simply accurate predictions.

Scikit-learn focuses on minimizing error through metrics like RMSE or R^2 and generalizing to unseen data.

Even if model assumptions are violated, predictions can still be reliable, especially with well-regularized models.

In contrast, statistical inference emphasizes model assumptions because they ensure unbiased, reliable estimates and valid significance tests.

Violating assumptions like homoscedasticity or normality can lead to misleading conclusions, so residual analysis is crucial for ensuring robustness and accuracy in inferential statistics

In [ ]: