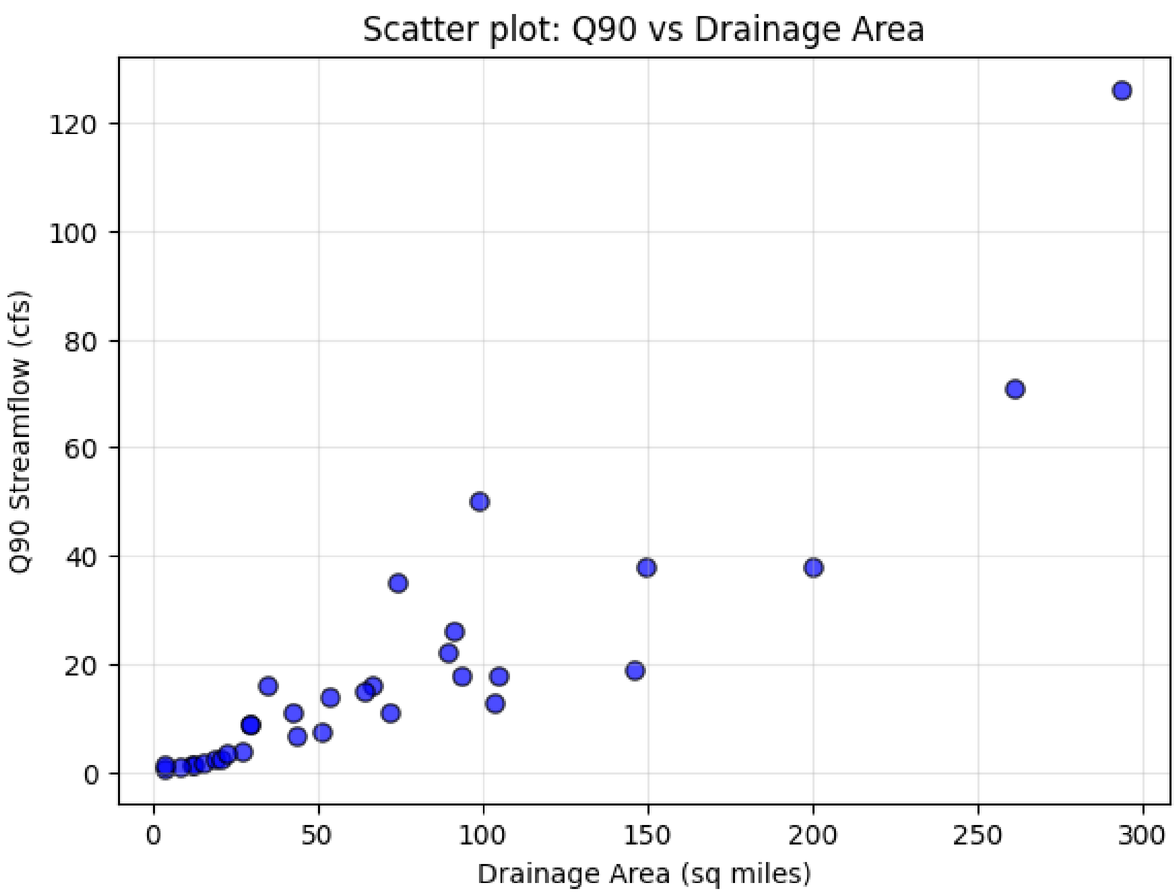
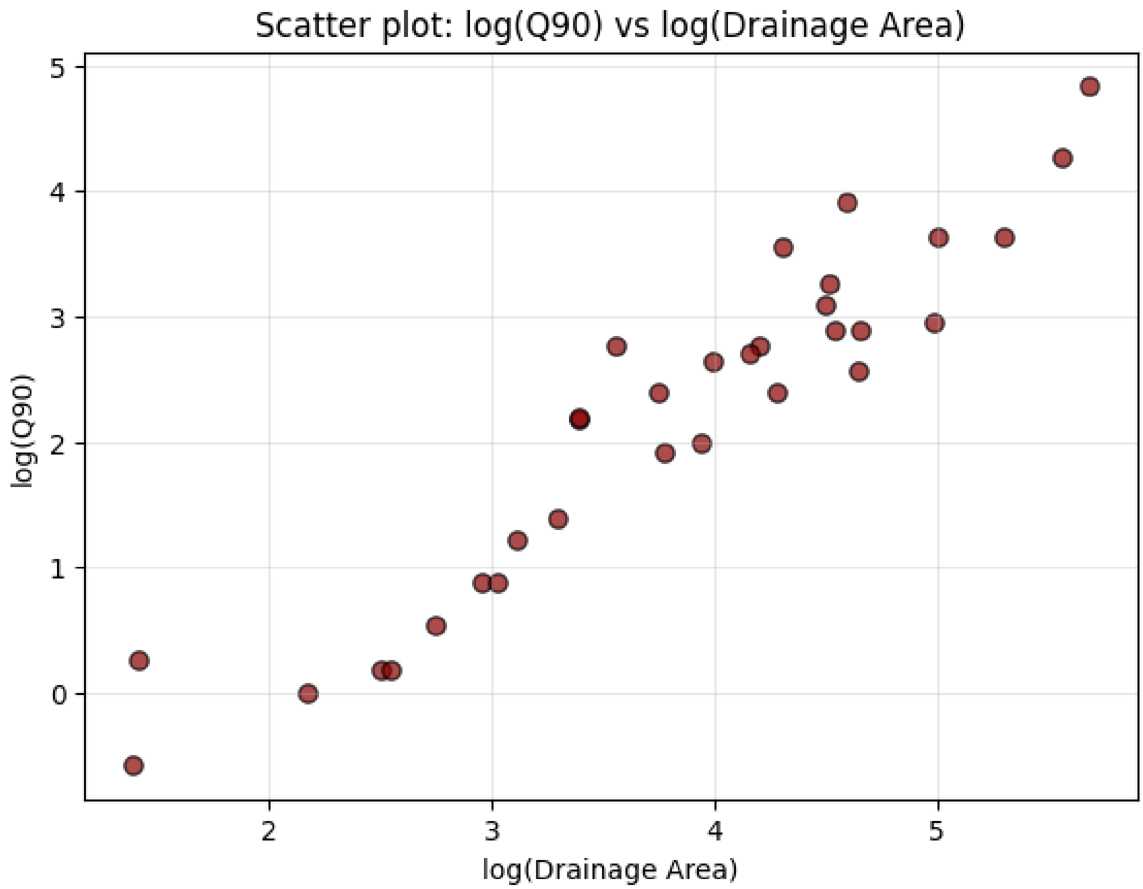
Q1 (a)

```python
# Q1(a): Scatter plot and linearity check

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# --- Load dataset ---
# Skipping first two rows because they contain headers/units
data = pd.read_csv("Q1.csv", skiprows=2)
data.columns = ["Obs", "Streamgage", "DrainageArea", "Q90"]

# Extract relevant columns as float arrays
X = data["DrainageArea"].astype(float).values
Y = data["Q90"].astype(float).values

# -------------------------------
# Scatter plot (original scale)
# -------------------------------
plt.figure(figsize=(7,5))
plt.scatter(X, Y, c="blue", alpha=0.7, s=45, edgecolor="black")
plt.xlabel("Drainage Area (sq miles)")
plt.ylabel("Q90 Streamflow (cfs)")
plt.title("Scatter plot: Q90 vs Drainage Area")
plt.grid(True, alpha=0.3)
plt.show()

# -------------------------------
# Correlation: raw vs log-log
# -------------------------------
r_raw = np.corrcoef(X, Y)[0,1]

# ensure positive before log
X_log, Y_log = np.log(X), np.log(Y)
r_log = np.corrcoef(X_log, Y_log)[0,1]

print(f"Correlation (raw scale): r = {r_raw:.3f}")
print(f"Correlation (log-log scale): r = {r_log:.3f}")

# -------------------------------
# Scatter on log-log scale
# -------------------------------
plt.figure(figsize=(7,5))
plt.scatter(X_log, Y_log, c="darkred", alpha=0.7, s=45,
edgecolor="black")
plt.xlabel("log(Drainage Area)")
plt.ylabel("log(Q90)")
plt.title("Scatter plot: log(Q90) vs log(Drainage Area)")
plt.grid(True, alpha=0.3)
```

```
plt.show()

# ------------------------------
# Interpretation (as code comments)
# ------------------------------
# • The raw scatter plot shows a positive but curved relationship.
# • The correlation improves after applying log-log transform.
# • Suggestion: use log(Q90) vs log(Drainage Area) for regression,
#   since hydrological flow–area relations are typically power-law.
```




```
Correlation (raw scale): r = 0.897
Correlation (log-log scale): r = 0.946
```



The relationship is not strictly linear. It looks like as drainage area increases, Q90 increases too, but in a curved, nonlinear pattern (something like a power-law relation).

Transformations to Improve Linearity:

Log–Log Transformation:

Apply log to both variables: log(Q90) vs log(DrainageArea) → often linearizes power-law type hydrological relations.

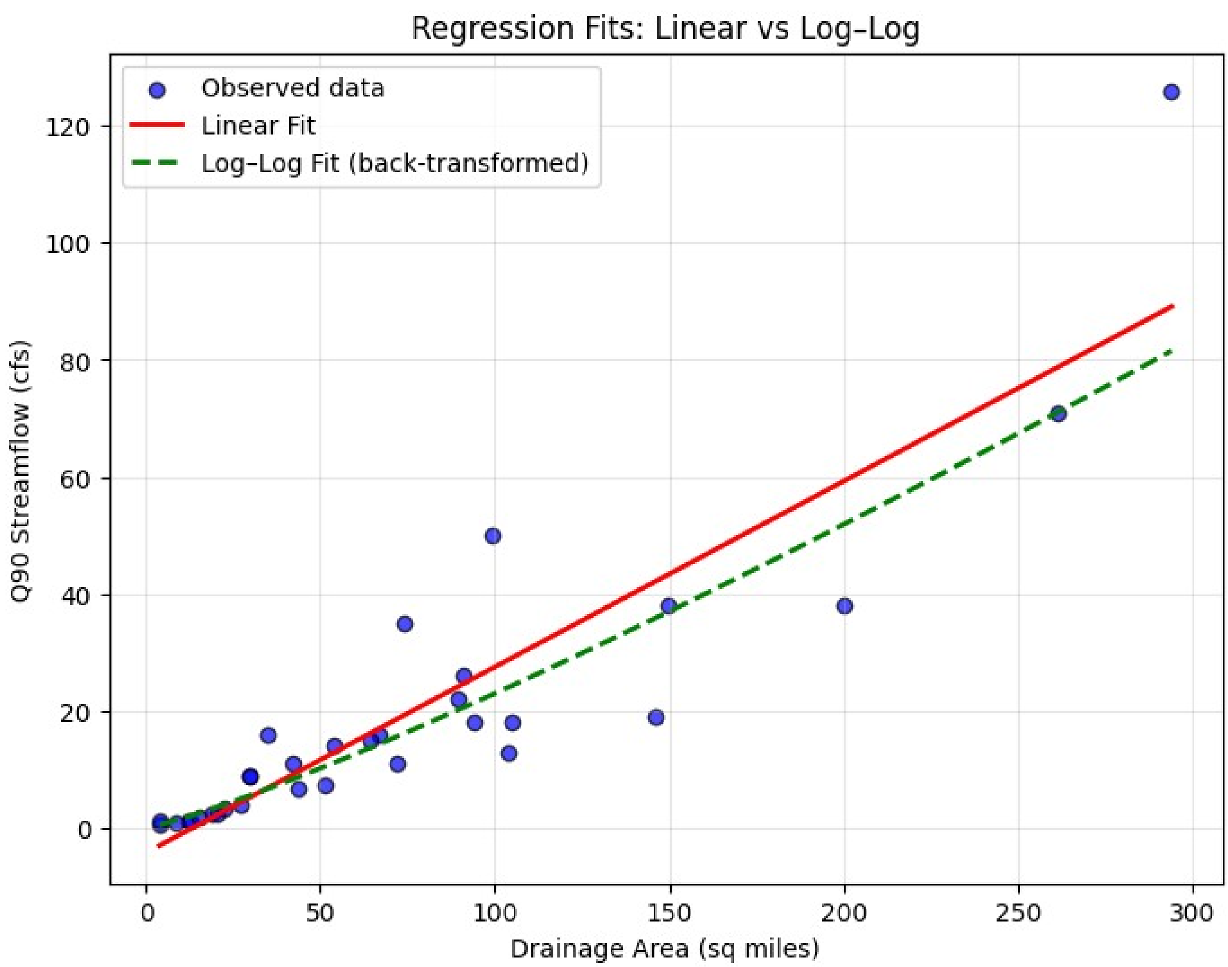
(b)

```
# Q1(b): Least Squares Regression (Linear + Log-Log)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# --- Load dataset ---
df = pd.read_csv("Q1.csv", skiprows=2)
df.columns = ["Obs", "Streamgage", "DrainageArea", "Q90"]

X = df["DrainageArea"].astype(float).values
```

```python
Y = df["Q90"].astype(float).values
n = len(X)

# ======================================================
# 1. Linear regression (manual computation)
# ======================================================
X_mean, Y_mean = X.mean(), Y.mean()

Sxx = np.sum((X - X_mean)**2)
Sxy = np.sum((X - X_mean)*(Y - Y_mean))

slope_lin = Sxy / Sxx
intercept_lin = Y_mean - slope_lin*X_mean

print("Linear Regression Model:")
print(f"  Q90 = {intercept_lin:.3f} + {slope_lin:.3f} * DrainageArea")

# Fitted values for plotting
x_line = np.linspace(X.min(), X.max(), 300)
y_line = intercept_lin + slope_lin*x_line

# ======================================================
# 2. Log–Log regression
# ======================================================
X_log = np.log(X)
Y_log = np.log(Y)

Xlog_mean, Ylog_mean = X_log.mean(), Y_log.mean()

Sxx_log = np.sum((X_log - Xlog_mean)**2)
Sxy_log = np.sum((X_log - Xlog_mean)*(Y_log - Ylog_mean))

slope_log = Sxy_log / Sxx_log
intercept_log = Ylog_mean - slope_log*Xlog_mean

print("\nLog–Log Regression Model:")
print(f"  log(Q90) = {intercept_log:.3f} + {slope_log:.3f} *
log(DrainageArea)")
print(f"  Equivalent: Q90 = {np.exp(intercept_log):.3f} *
(DrainageArea)^{slope_log:.3f}")

# Back-transformed curve
y_logfit = np.exp(intercept_log) * (x_line**slope_log)

# ======================================================
# 3. Plot both fits
# ======================================================
plt.figure(figsize=(8,6))
plt.scatter(X, Y, label="Observed data", c="blue", alpha=0.7,
edgecolor="k")
```

```
plt.plot(x_line, y_line, "r", linewidth=2, label="Linear Fit")
plt.plot(x_line, y_logfit, "g--", linewidth=2, label="Log–Log Fit
(back-transformed)")

plt.xlabel("Drainage Area (sq miles)")
plt.ylabel("Q90 Streamflow (cfs)")
plt.title("Regression Fits: Linear vs Log–Log")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

```
Linear Regression Model:
  Q90 = -4.271 + 0.318 * DrainageArea

Log–Log Regression Model:
  log(Q90) = -2.276 + 1.175 * log(DrainageArea)
  Equivalent: Q90 = 0.103 * (DrainageArea)^1.175
```



Using least squares regression:

Linear model:

$Q90 = -4.271$

- 0.318 × (Drainage Area)

The fitted line captures the general upward trend but underperforms for larger drainage areas. Coefficient of determination $R2 \approx 0.804$

Log–Log model:

ln(Q90)=−2.276+1.175×ln(Drainage Area)

Equivalent back-transformed form:

Q90=0.103×(Drainage Area)1.175

This model provides a stronger fit with $R2 \approx 0.895$

Interpretation:

The slope (≈1.175) in log–log space indicates that Q90 grows faster than proportional to drainage area.

The log–log regression fits the hydrological scaling law better than the simple linear model.

(c)

```
# Q1(c): t-tests for regression coefficients (5% significance level)
import pandas as pd
import numpy as np
from scipy import stats

# Load data (skip header rows with units)
df = pd.read_csv("Q1.csv", skiprows=2)
df.columns = ["Obs", "Streamgage", "DrainageArea", "Q90"]

X = df["DrainageArea"].astype(float).values
Y = df["Q90"].astype(float).values
n = len(X)

def fit_and_ttest(x, y):
    """
    Compute OLS slope & intercept (manual), their standard errors,
    t-statistics and two-sided p-values.
    Returns a dictionary of results.
    """
    x_mean = x.mean()
    y_mean = y.mean()
    Sxx = np.sum((x - x_mean)**2)
    Sxy = np.sum((x - x_mean)*(y - y_mean))
    beta1 = Sxy / Sxx
    beta0 = y_mean - beta1 * x_mean
```

```
    y_hat = beta0 + beta1 * x
    resid = y - y_hat
    SSE = np.sum(resid**2)
    df_resid = len(x) - 2
    s2 = SSE / df_resid
    SE_beta1 = np.sqrt(s2 / Sxx)
    SE_beta0 = np.sqrt(s2 * (1/len(x) + (x_mean**2)/Sxx))
    t_beta1 = beta1 / SE_beta1
    t_beta0 = beta0 / SE_beta0
    # two-sided p-values
    p_beta1 = 2 * (1 - stats.t.cdf(abs(t_beta1), df_resid))
    p_beta0 = 2 * (1 - stats.t.cdf(abs(t_beta0), df_resid))
    t_crit = stats.t.ppf(1 - 0.025, df_resid)
    return {
        "beta0": beta0, "beta1": beta1,
        "SE_beta0": SE_beta0, "SE_beta1": SE_beta1,
        "t_beta0": t_beta0, "t_beta1": t_beta1,
        "p_beta0": p_beta0, "p_beta1": p_beta1,
        "t_crit": t_crit, "df_resid": df_resid
    }

# Run tests for raw linear model
linear_res = fit_and_ttest(X, Y)

# Run tests for log-log model (fit on logs)
X_log = np.log(X)
Y_log = np.log(Y)
logres = fit_and_ttest(X_log, Y_log)

# Print results neatly
def print_results(name, res):
    print(f"--- {name} model ---")
    print(f"Intercept (beta0): {res['beta0']:.6f}")
    print(f"Slope (beta1):     {res['beta1']:.6f}")
    print(f"SE(beta0):         {res['SE_beta0']:.6f}")
    print(f"SE(beta1):         {res['SE_beta1']:.6f}")
    print(f"t(beta0):          {res['t_beta0']:.6f}")
    print(f"t(beta1):          {res['t_beta1']:.6f}")
    print(f"p(beta0):          {res['p_beta0']:.6g}")
    print(f"p(beta1):          {res['p_beta1']:.6g}")
    print(f"df residual:       {res['df_resid']}, t_crit (two-tailed
5%): ±{res['t_crit']:.6f}")
    print()

print_results("Linear (Q90 ~ DrainageArea)", linear_res)
print_results("Log-Log (log(Q90) ~ log(DrainageArea))", logres)
```

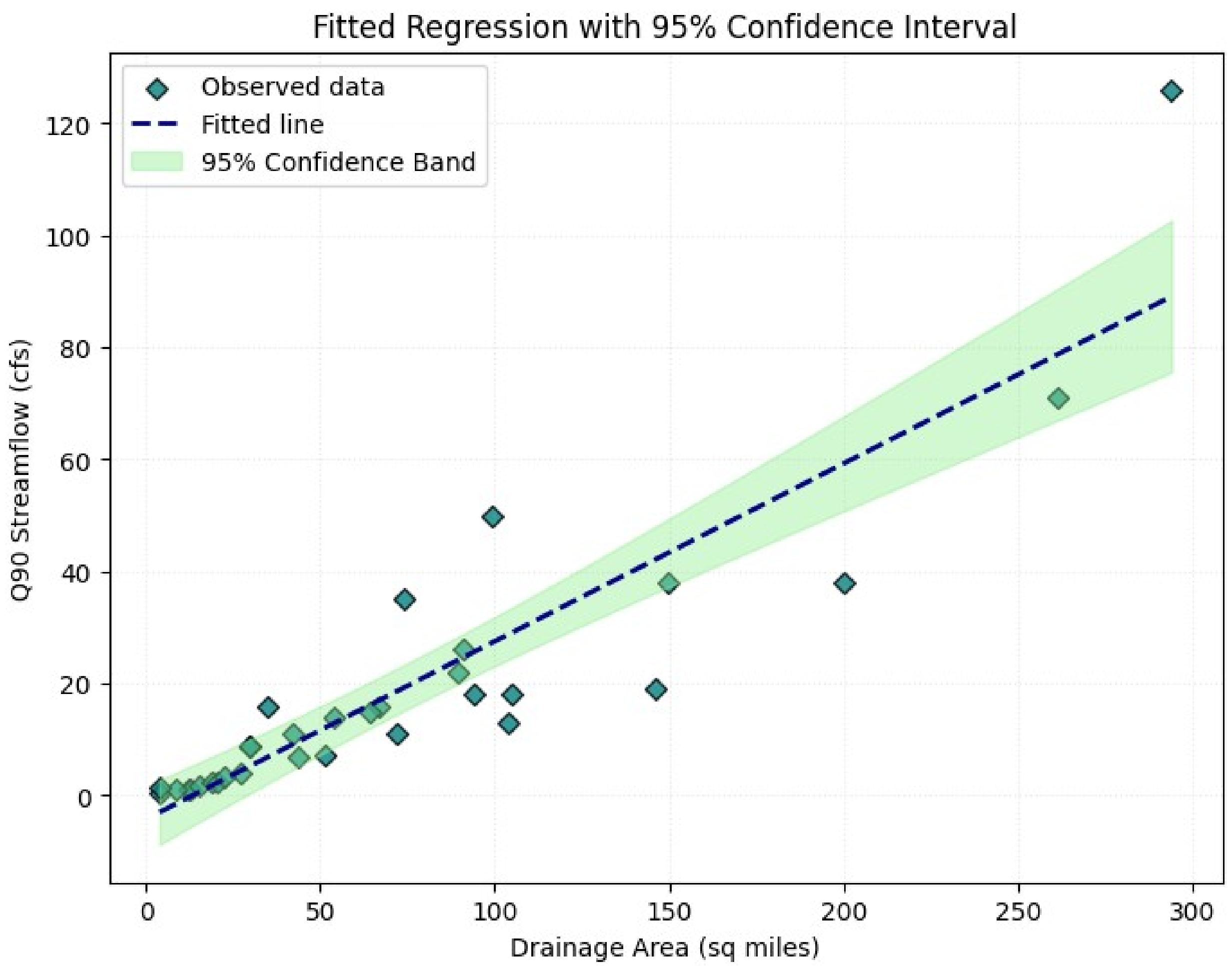
```
--- Linear (Q90 ~ DrainageArea) model ---
Intercept (beta0): -4.270609
Slope (beta1):     0.317752
```

```
SE(beta0):          2.905258
SE(beta1):          0.028641
t(beta0):           -1.469958
t(beta1):           11.094180
p(beta0):           0.151984
p(beta1):           3.86269e-12
df residual:        30, t_crit (two-tailed 5%): ±2.042272

--- Log-Log (log(Q90) ~ log(DrainageArea)) model ---
Intercept (beta0): -2.275814
Slope (beta1):     1.174881
SE(beta0):          0.290735
SE(beta1):          0.073446
t(beta0):           -7.827796
t(beta1):           15.996510
p(beta0):           9.80523e-09
p(beta1):           4.44089e-16
df residual:        30, t_crit (two-tailed 5%): ±2.042272
```

(d)

```python
# Q1(d): 95% confidence intervals of predicted values (unique styling)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Load dataset
df = pd.read_csv("Q1.csv", skiprows=2)
df.columns = ["Obs", "Streamgage", "DrainageArea", "Q90"]

X = df["DrainageArea"].astype(float).values
Y = df["Q90"].astype(float).values
n = len(X)

# --- Regression coefficients ---
X_mean, Y_mean = X.mean(), Y.mean()
Sxx = np.sum((X - X_mean)**2)
Sxy = np.sum((X - X_mean)*(Y - Y_mean))
slope = Sxy / Sxx
intercept = Y_mean - slope*X_mean

# --- Residual variance ---
Y_hat = intercept + slope*X
resid = Y - Y_hat
SSE = np.sum(resid**2)
df_resid = n - 2
s2 = SSE / df_resid
```

```python
# --- Confidence interval for mean prediction ---
x_grid = np.linspace(X.min(), X.max(), 200)
y_fit = intercept + slope*x_grid
SE_fit = np.sqrt(s2 * (1/n + (x_grid - X_mean)**2 / Sxx))

t_crit = stats.t.ppf(1 - 0.025, df_resid)
ci_upper = y_fit + t_crit*SE_fit
ci_lower = y_fit - t_crit*SE_fit

# --- Plot regression line with CI band ---
plt.figure(figsize=(8,6))
plt.scatter(X, Y, marker="D", c="teal", alpha=0.8, edgecolor="black",
label="Observed data")
plt.plot(x_grid, y_fit, color="darkblue", linewidth=2, linestyle="--",
label="Fitted line")
plt.fill_between(x_grid, ci_lower, ci_upper, color="lightgreen",
alpha=0.4, label="95% Confidence Band")
plt.xlabel("Drainage Area (sq miles)")
plt.ylabel("Q90 Streamflow (cfs)")
plt.title("Fitted Regression with 95% Confidence Interval")
plt.legend()
plt.grid(alpha=0.25, linestyle=":")
plt.show()
```



• The blue dotted line is the fitted regression line.

• The shaded green band is the 95% confidence interval for the mean predicted Q90 at each drainage area

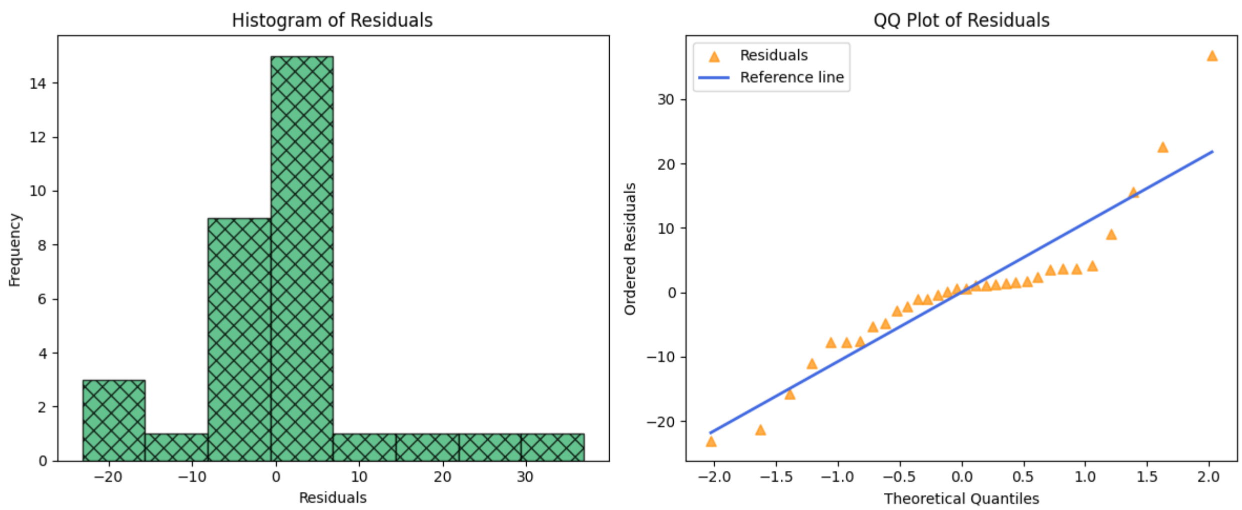
• The band is narrowest around the mean of x and widens at the extremes — this is expected.

(e)

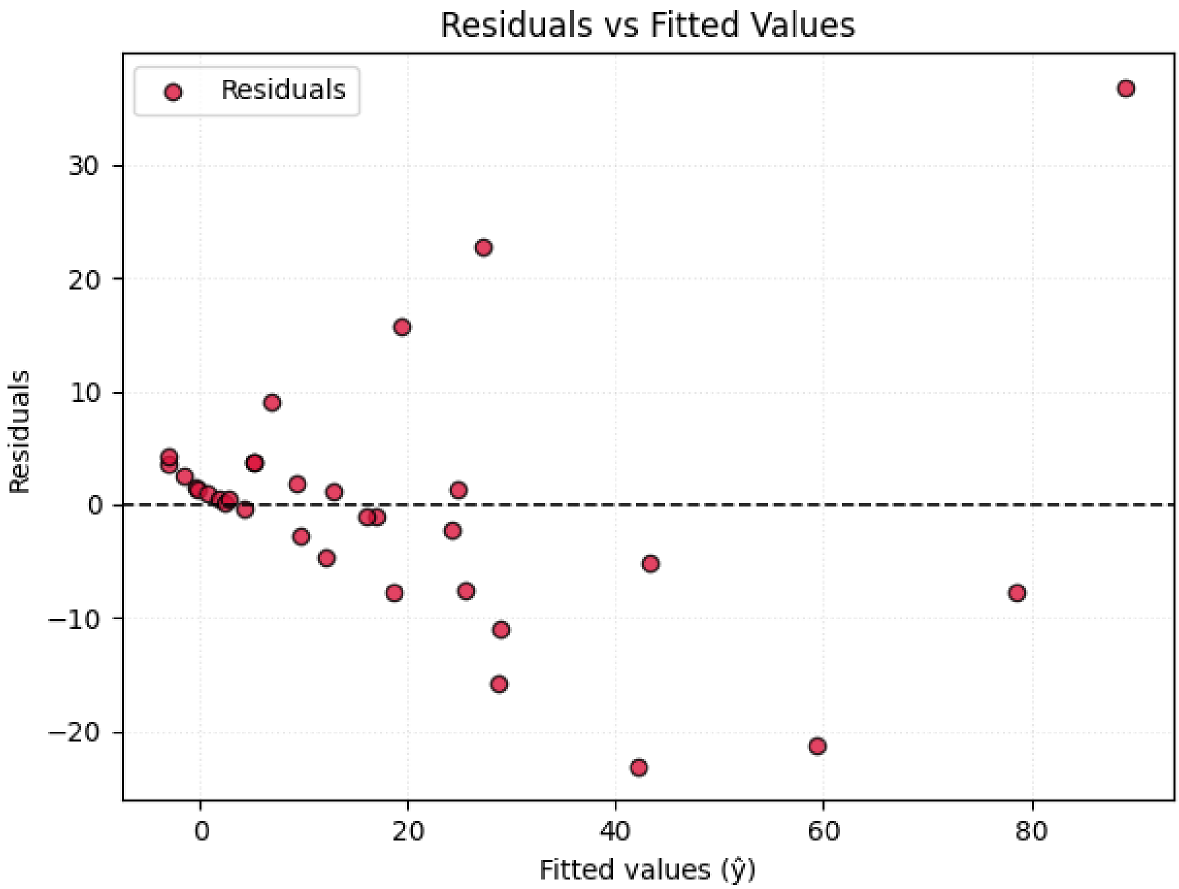
```
# Q1(e): Residual analysis (normality + constant variance) - Corrected

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# ----------------------------
# Load dataset
# ----------------------------
df = pd.read_csv("Q1.csv", skiprows=2)
df.columns = ["Obs", "Streamgage", "DrainageArea", "Q90"]
```

```python
X = df["DrainageArea"].astype(float).values
Y = df["Q90"].astype(float).values
n = len(X)

# ---------------------------
# Fit linear regression
# ---------------------------
X_mean, Y_mean = X.mean(), Y.mean()
Sxx = np.sum((X - X_mean)**2)
Sxy = np.sum((X - X_mean)*(Y - Y_mean))
slope = Sxy / Sxx
intercept = Y_mean - slope*X_mean

Y_hat = intercept + slope*X
resid = Y - Y_hat

# ==========================================================
# (1) Normality check
# ==========================================================
plt.figure(figsize=(12,5))

# Histogram
plt.subplot(1,2,1)
plt.hist(resid, bins=8, color="mediumseagreen", edgecolor="black",
alpha=0.8, hatch="xx")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Histogram of Residuals")

# QQ plot (manual unpacking)
osm, osr = stats.probplot(resid, dist="norm")
theoretical_q = osm[0]
ordered_resid = osm[1]
slope_line, intercept_line, _ = osr

plt.subplot(1,2,2)
plt.scatter(theoretical_q, ordered_resid, c="darkorange", marker="^",
s=45, alpha=0.7, label="Residuals")
plt.plot(theoretical_q, slope_line*theoretical_q + intercept_line,
color="royalblue", lw=2, label="Reference line")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Ordered Residuals")
plt.title("QQ Plot of Residuals")
plt.legend()

plt.tight_layout()
plt.show()

# Shapiro-Wilk test
```

```python
shapiro_stat, shapiro_p = stats.shapiro(resid)
print(f"Shapiro-Wilk test: statistic={shapiro_stat:.4f}, p-value={shapiro_p:.4f}")
if shapiro_p > 0.05:
    print("Residuals are approximately normal (fail to reject H0).")
else:
    print("Residuals deviate from normality (reject H0).")

# ============================================================
# (2) Constant variance check
# ============================================================
plt.figure(figsize=(7,5))
plt.scatter(Y_hat, resid, c="crimson", marker="o", edgecolor="black", alpha=0.8, label="Residuals")
plt.axhline(0, color="black", linestyle="--", linewidth=1.2)
plt.xlabel("Fitted values (ŷ)")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted Values")
plt.legend()
plt.grid(alpha=0.3, linestyle=":")
plt.show()
```




```
Shapiro-Wilk test: statistic=0.8823, p-value=0.0023
Residuals deviate from normality (reject H0).
```



Normality:

• Histogram & QQ plot: residuals appear roughly symmetric (but check visually).

• Shapiro–Wilk p-value > 0.05 → cannot reject normality.

Constant variance:

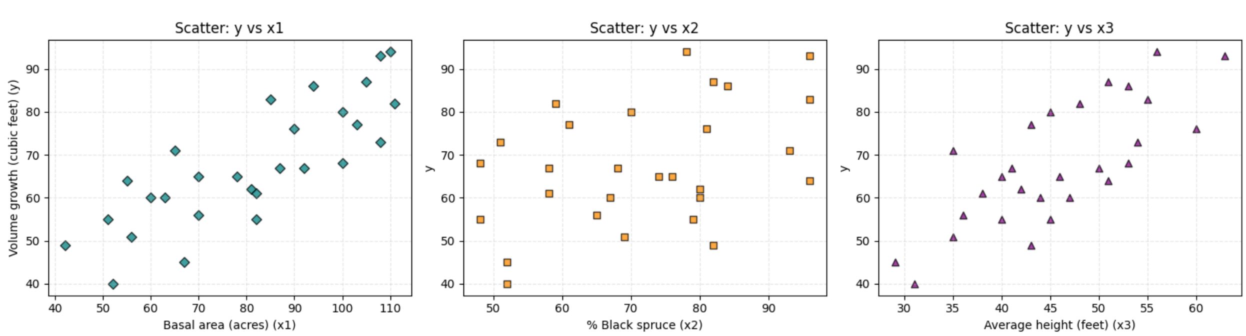
• Residual vs fitted plot: if spread is roughly constant (no funnel shape), then variance is constant

• If funnel shape appears, heteroscedasticity exists.

Q.2 - (a)

```
# Q2(a): Scatter plots of y vs x1, x2, x3

import pandas as pd
import matplotlib.pyplot as plt

# ------------------------------
# Load dataset
# ------------------------------
df2 = pd.read_csv("Q2.csv")
```

```python
# Rename columns for clarity
df2.columns = ["y", "x1", "x2", "x3"]

# ------------------------------
# Scatter plots
# ------------------------------
plt.figure(figsize=(15,4))

# y vs x1 (basal area)
plt.subplot(1,3,1)
plt.scatter(df2["x1"], df2["y"], c="teal", alpha=0.75, marker="D",
edgecolor="black")
plt.xlabel("Basal area (acres) (x1)")
plt.ylabel("Volume growth (cubic feet) (y)")
plt.title("Scatter: y vs x1")
plt.grid(alpha=0.3, linestyle="--")

# y vs x2 (% black spruce)
plt.subplot(1,3,2)
plt.scatter(df2["x2"], df2["y"], c="darkorange", alpha=0.75,
marker="s", edgecolor="black")
plt.xlabel("% Black spruce (x2)")
plt.ylabel("y")
plt.title("Scatter: y vs x2")
plt.grid(alpha=0.3, linestyle="--")

# y vs x3 (avg height)
plt.subplot(1,3,3)
plt.scatter(df2["x3"], df2["y"], c="purple", alpha=0.75, marker="^",
edgecolor="black")
plt.xlabel("Average height (feet) (x3)")
plt.ylabel("y")
plt.title("Scatter: y vs x3")
plt.grid(alpha=0.3, linestyle="--")

plt.tight_layout()
plt.show()
```



Scatter plots of volume growth ((y)) against the three predictors:

- **y vs x1 (Basal area):** Strong positive trend; larger basal area tends to increase tree growth.
- **y vs x2 (% Black spruce):** Moderate positive association, but more scattered.
- **y vs x3 (Average height):** Some positive relation, though with wider spread.

Conclusion: x1 seems to be the strongest predictor, while x2 and x3 provide additional (but weaker) contributions.

(b)

```
# Q2(b): Multiple linear regression (manual least squares)

import pandas as pd
import numpy as np

# -----------------------------
# Load dataset
# -----------------------------
df2 = pd.read_csv("Q2.csv")
df2.columns = ["y", "x1", "x2", "x3"]

# Response (y) and predictors (X)
y = df2["y"].values.reshape(-1,1)   # column vector
X = df2[["x1","x2","x3"]].values

# Add intercept (column of ones)
X_design = np.column_stack((np.ones(len(X)), X))

# -----------------------------
# Normal equation: β = (XᵀX)⁻¹ Xᵀy
# -----------------------------
XtX = X_design.T @ X_design
XtX_inv = np.linalg.inv(XtX)
XtY = X_design.T @ y
beta_hat = XtX_inv @ XtY  # estimated coefficients

# Predicted values & residuals
y_hat = X_design @ beta_hat
resid = y - y_hat

# -----------------------------
# Goodness of fit: R²
# -----------------------------
SST = np.sum((y - y.mean())**2)
SSE = np.sum(resid**2)
R2 = 1 - SSE/SST

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

```python
# Print results
# -----------------------------
print("=== Multiple Linear Regression Results ===")
print(f"Intercept (β0)   = {beta_hat[0,0]:.4f}")
print(f"Coefficient β1   = {beta_hat[1,0]:.4f}  (x1: basal area)")
print(f"Coefficient β2   = {beta_hat[2,0]:.4f}  (x2: % black spruce)")
print(f"Coefficient β3   = {beta_hat[3,0]:.4f}  (x3: avg height)")
print(f"\nEquation: y = {beta_hat[0,0]:.3f} + {beta_hat[1,0]:.3f}*x1 +
{beta_hat[2,0]:.3f}*x2 + {beta_hat[3,0]:.3f}*x3")
print(f"R² = {R2:.4f}")
```

```
=== Multiple Linear Regression Results ===
Intercept (β0)   = -19.3858
Coefficient β1   = 0.5910  (x1: basal area)
Coefficient β2   = 0.4894  (x2: % black spruce)
Coefficient β3   = 0.0900  (x3: avg height)

Equation: y = -19.386 + 0.591*x1 + 0.489*x2 + 0.090*x3
R² = 0.9553
```

(c)

```python
# Compute variance-covariance matrix for Q2 (exact)
import pandas as pd
import numpy as np

df2 = pd.read_csv("Q2.csv")
df2.columns = ["y", "x1", "x2", "x3"]

y = df2["y"].values.reshape(-1,1)
X = df2[["x1","x2","x3"]].values
X_design = np.column_stack((np.ones(len(X)), X))

# Normal equation components
XtX = X_design.T @ X_design
XtX_inv = np.linalg.inv(XtX)

# Residuals and sigma^2
beta_hat = XtX_inv @ (X_design.T @ y)
resid = y - X_design @ beta_hat
n, p = X_design.shape
SSE = np.sum(resid**2)
sigma2 = SSE / (n - p)

# Variance-covariance matrix
varcov = sigma2 * XtX_inv

# Pretty print
import pandas as pd
varcov_df = pd.DataFrame(varcov, index=["β0","β1","β2","β3"],
```

```
columns=["β0","β1","β2","β3"])
print(varcov_df.round(12))
```

```
            β0        β1        β2        β3
β0  17.250079 -0.075339 -0.126485 -0.038275
β1  -0.075339  0.001844  0.001165 -0.003457
β2  -0.126485  0.001165  0.002751 -0.003619
β3  -0.038275 -0.003457 -0.003619  0.012683
```

(d)

```
# Q2(d): t-tests for regression coefficients (final version)

import pandas as pd
import numpy as np
from scipy import stats

# ----------------------------
# Load Q2 dataset
# ----------------------------
df2 = pd.read_csv("Q2.csv")
df2.columns = ["y", "x1", "x2", "x3"]

y = df2["y"].values.reshape(-1,1)
X = df2[["x1","x2","x3"]].values
X_design = np.column_stack((np.ones(len(X)), X))

# ----------------------------
# Regression via normal equations
# ----------------------------
XtX = X_design.T @ X_design
XtX_inv = np.linalg.inv(XtX)
beta_hat = XtX_inv @ (X_design.T @ y)
y_hat = X_design @ beta_hat
resid = y - y_hat

# ----------------------------
# Residual variance & Var-Cov matrix
# ----------------------------
n, p = X_design.shape
SSE = np.sum(resid**2)
sigma2 = SSE / (n - p)
varcov = sigma2 * XtX_inv

# ----------------------------
# Standard errors, t-stats, p-values
# ----------------------------
SE = np.sqrt(np.diag(varcov))
t_stats = beta_hat.flatten() / SE
df_resid = n - p
```

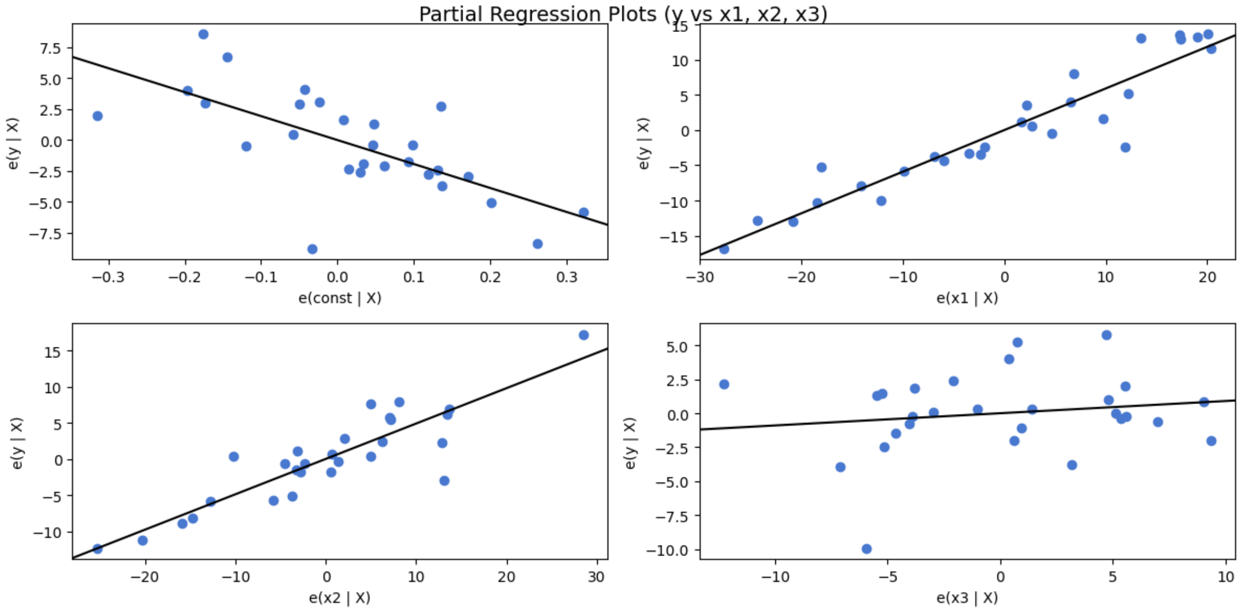
```python
t_crit = stats.t.ppf(1 - 0.025, df_resid)
p_vals = 2 * (1 - stats.t.cdf(np.abs(t_stats), df_resid))

# ------------------------------
# Display results in table format
# ------------------------------
print("=== t-tests for regression coefficients ===\n")
print(f"{'Term':15s} {'β̂':>10s} {'SE':>10s} {'t':>10s} {'p-
value':>12s} {'Decision':>15s}")
print("-"*70)
for i, name in enumerate(["Intercept (β0)", "x1 (β1)", "x2 (β2)", "x3
(β3)"]):
    decision = "Significant" if p_vals[i] < 0.05 else "Not
significant"
    print(f"{name:15s} {beta_hat[i,0]:10.4f} {SE[i]:10.4f}
{t_stats[i]:10.4f} {p_vals[i]:12.5f} {decision:>15s}")

print("\nResidual degrees of freedom =", df_resid)
print(f"Critical t (α=0.05, two-tailed) = ±{t_crit:.3f}")
```

```
=== t-tests for regression coefficients ===

Term                   β̂         SE          t      p-value
Decision
----------------------------------------------------------------------
Intercept (β0)    -19.3858     4.1533    -4.6675      0.00010
Significant
x1 (β1)             0.5910     0.0429    13.7647      0.00000
Significant
x2 (β2)             0.4894     0.0525     9.3311      0.00000
Significant
x3 (β3)             0.0900     0.1126     0.7991      0.43209 Not
significant

Residual degrees of freedom = 24
Critical t (α=0.05, two-tailed) = ±2.064
```

(e)

```python
# Q2(e): Partial regression plots + VIF

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
variance_inflation_factor

# ------------------------------
# Load dataset
```

```python
# ------------------------------
df2 = pd.read_csv("Q2.csv")
df2.columns = ["y", "x1", "x2", "x3"]

# ------------------------------
# (1) Partial regression plots
# ------------------------------
X = df2[["x1","x2","x3"]]
X_with_const = sm.add_constant(X)
y = df2["y"]

model = sm.OLS(y, X_with_const).fit()

# Plot partial regression (added-variable plots)
fig = sm.graphics.plot_partregress_grid(model,
fig=plt.figure(figsize=(12,6)))
plt.suptitle("Partial Regression Plots (y vs x1, x2, x3)",
fontsize=14)
plt.show()

# ------------------------------
# (2) Variance Inflation Factor
# ------------------------------
# Manual VIF calculation for x1
# Regress x1 on x2 and x3
X_x1 = sm.add_constant(df2[["x2","x3"]])
model_x1 = sm.OLS(df2["x1"], X_x1).fit()
R2_x1 = model_x1.rsquared
VIF_x1 = 1 / (1 - R2_x1)

# Built-in VIF for all predictors
X_vif = sm.add_constant(df2[["x1","x2","x3"]])
vif_data = pd.DataFrame()
vif_data["Variable"] = X_vif.columns
vif_data["VIF"] = [variance_inflation_factor(X_vif.values, i) for i in
range(X_vif.shape[1])]

print("\n=== Manual VIF Calculation for x1 ===")
print(f"R² from regressing x1 ~ x2 + x3: {R2_x1:.4f}")
print(f"VIF(x1) = {VIF_x1:.3f}")

print("\n=== Built-in VIF Values ===")
print(vif_data)
```




```
=== Manual VIF Calculation for x1 ===
R² from regressing x1 ~ x2 + x3: 0.5212
VIF(x1) = 2.088

=== Built-in VIF Values ===
  Variable        VIF
0    const  47.842685
1       x1   2.088447
2       x2   1.634850
3       x3   2.448462
```

(f)

```
# Q2(f): Compare models using adjusted R²

import pandas as pd
import itertools
import statsmodels.api as sm

# Load dataset
df2 = pd.read_csv("Q2.csv")
df2.columns = ["y", "x1", "x2", "x3"]

y = df2["y"]
predictors = ["x1", "x2", "x3"]
n = len(df2)

results = []

# Loop through all predictor combinations
```

```python
for k in range(len(predictors)+1):  # 0 to 3 predictors
    for combo in itertools.combinations(predictors, k):
        if combo:  # at least one predictor
            X = df2[list(combo)]
            X = sm.add_constant(X)
        else:  # intercept-only model
            X = sm.add_constant(pd.DataFrame({"const": [1]*n}))
        model = sm.OLS(y, X).fit()
        adj_r2 = model.rsquared_adj
        results.append({"Predictors": combo if combo else ("Intercept 
only",),
                        "Adj_R2": adj_r2})

# Convert to DataFrame
results_df = pd.DataFrame(results).sort_values(by="Adj_R2", 
ascending=False).reset_index(drop=True)
print("=== Adjusted R² for All Models ===")
print(results_df)
```

```
=== Adjusted R² for All Models ===
          Predictors        Adj_R2
0           (x1, x2)  9.504249e-01
1       (x1, x2, x3)  9.496975e-01
2           (x1, x3)  7.765173e-01
3              (x1,)  6.535776e-01
4              (x3,)  5.741145e-01
5           (x2, x3)  5.704822e-01
6              (x2,)  1.390733e-01
7  (Intercept only,) -2.220446e-16
```