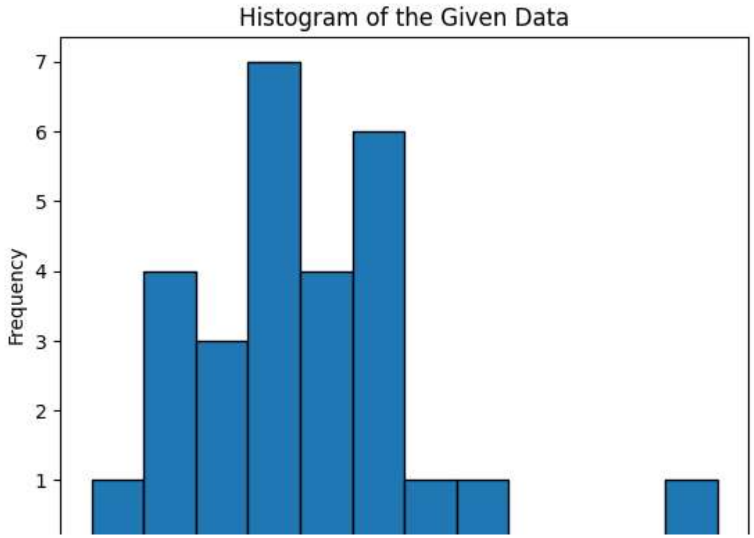
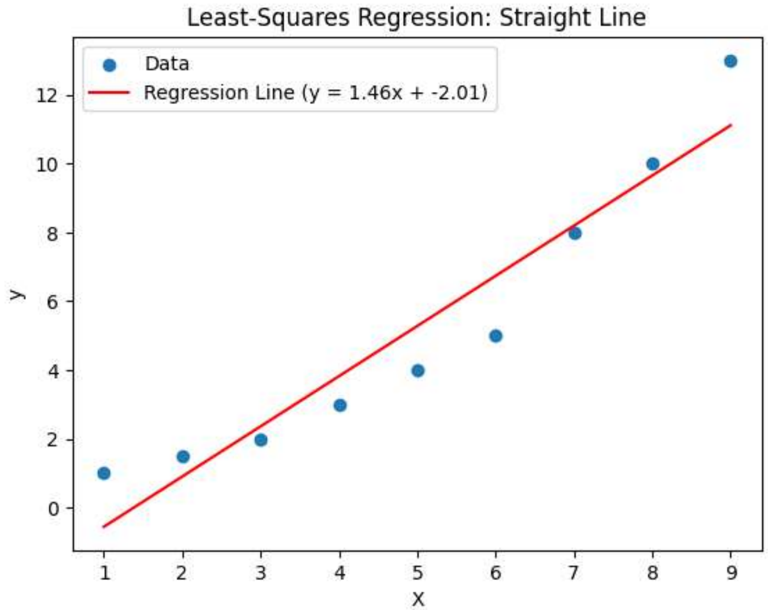
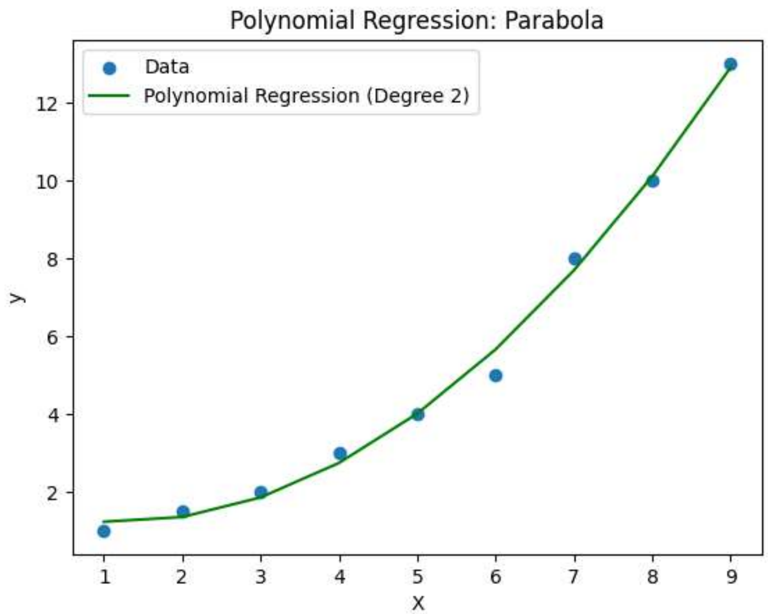
```
#Question 1
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# Given data
data = np.array([28.65, 26.55, 26.65, 27.65, 27.35, 28.35, 26.85,
                 28.65, 29.65, 27.85, 27.05, 28.25, 28.85, 26.75,
                 27.65, 28.45, 28.65, 28.45, 31.65, 26.35, 27.75,
                 29.25, 27.65, 28.65, 27.65, 28.55, 27.65, 27.25])

# (a) Mean
mean_value = np.mean(data)

# (b) Standard Deviation
std_deviation = np.std(data)

# (c) Variance
variance = np.var(data)

# (d) Coefficient of Variation
coeff_of_variation = (std_deviation / mean_value) * 100

# (e) 90% Confidence Interval for the Mean
confidence_interval = stats.t.interval(0.9, len(data)-1, loc=mean_value, scale=stats.sem(data))

# (f) Histogram
plt.hist(data, bins=np.arange(26, 32.5, 0.5), edgecolor='black')
plt.title('Histogram of the Given Data')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()

# (g) Range Encompassing 68% of Readings (assuming normal distribution)
lower_range = mean_value - std_deviation
upper_range = mean_value + std_deviation

# Print results
print(f"(a) Mean: {mean_value}")
print(f"(b) Standard Deviation: {std_deviation}")
print(f"(c) Variance: {variance}")
print(f"(d) Coefficient of Variation: {coeff_of_variation}")
print(f"(e) 90% Confidence Interval: {confidence_interval}")
print(f"(g) Range Encompassing 68% of Readings: ({lower_range}, {upper_range})")
```




```
#Question 2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm

# Given data
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 1.5, 2, 3, 4, 5, 8, 10, 13])

# Reshape X for sklearn
X_reshaped = X.reshape(-1, 1)

# (a) Least-squares regression for a straight line
model = LinearRegression()
model.fit(X_reshaped, y)

# Slope and intercept
slope = model.coef_[0]
intercept = model.intercept_

# Predicted values
y_pred = model.predict(X_reshaped)

# Standard error of the estimate (Sy/x)
syx = np.sqrt(mean_squared_error(y, y_pred))

# Coefficient of determination (R-square)
r_square = r2_score(y, y_pred)

# Correlation coefficient
correlation_coefficient = np.corrcoef(X, y)[0, 1]

# Plotting
```

```
plt.scatter(X, y, label='Data')
plt.plot(X, y_pred, color='red', label=f'Regression Line (y = {slope:.2f}x + {intercept:.2f})')
plt.title('Least-Squares Regression: Straight Line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()

# (b) Polynomial regression for a parabola
poly_degree = 2
X_poly = np.column_stack([X**i for i in range(poly_degree + 1)])

# Add a constant term to the design matrix
X_poly = sm.add_constant(X_poly)

model_poly = sm.OLS(y, X_poly).fit()

# Predicted values
y_pred_poly = model_poly.predict(X_poly)

# Plotting
plt.scatter(X, y, label='Data')
plt.plot(X, y_pred_poly, color='green', label=f'Polynomial Regression (Degree {poly_degree})')
plt.title('Polynomial Regression: Parabola')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()

# (c) Compare with statsmodels OLS
X_statsmodels = sm.add_constant(X)
model_ols = sm.OLS(y, X_statsmodels).fit()

# Print results
print(f"(a) Results for Straight Line:")
print(f"    Slope: {slope}")
print(f"    Intercept: {intercept}")
print(f"    Standard Error of the Estimate (Sy/x): {syx}")
print(f"    Coefficient of Determination (R-square): {r_square}")
print(f"    Correlation Coefficient: {correlation_coefficient}")
print(f"(b) Results for Polynomial Regression:")
print(f"    Coefficients: {model_poly.params}")
print(f"    Intercept: {model_poly.params[0]}")
print(f"(c) Results from statsmodels OLS:")
print(model_ols.summary())
```






```
(a) Results for Straight Line:
    Slope: 1.4583333333333337
    Intercept: -2.0138888888888891
    Standard Error of the Estimate (Sy/x): 1.1523593618161967
    Coefficient of Determination (R-square): 0.9143610668789809
    Correlation Coefficient: 0.9562222894698601
(b) Results for Polynomial Regression:
    Coefficients: [ 1.48809524 -0.45183983  0.19101732]
    Intercept: 1.488095238095239
```

```
(c) Results from statsmodels OLS:
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.914
Model:                            OLS   Adj. R-squared:                  0.902
Method:                 Least Squares   F-statistic:                     74.74
Date:                Sun, 03 Dec 2023   Prob (F-statistic):           5.54e-05
Time:                        16:39:52   Log-Likelihood:                -14.047
No. Observations:                   9   AIC:                             32.09
Df Residuals:                       7   BIC:                             32.49
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.0139      0.949     -2.122      0.072      -4.259       0.231
x1             1.4583      0.169      8.645      0.000       1.059       1.857
==============================================================================
Omnibus:                        0.429   Durbin-Watson:                   0.629
Prob(Omnibus):                  0.807   Jarque-Bera (JB):                0.477
```

```
#Question 3
import numpy as np
import statsmodels.api as sm
from scipy.stats import t

# Given data
X1 = np.array([0, 0, 1, 2, 0, 1, 2, 2, 1])
X2 = np.array([0, 2, 2, 4, 4, 6, 6, 2, 1])
Y = np.array([14, 21, 11, 12, 23, 23, 14, 6, 11])

# Create design matrix X with a constant term
X = sm.add_constant(np.column_stack((X1, X2)))

# Fit the OLS model
model = sm.OLS(Y, X).fit()

# Print coefficients
print("(a) Coefficients:")
print(model.params)

# Print standard error of the estimate (Sy/x)
syx = np.sqrt(model.mse_resid)
print(f"(a) Standard Error of the Estimate (Sy/x): {syx:.4f}")

# Confidence Intervals
alpha = 0.05  # 95% confidence interval
t_critical = t.ppf(1 - alpha/2, model.df_resid)

# Confidence Interval for Intercept (beta_0)
ci_intercept = model.conf_int(alpha=alpha)[0]
print(f"(a) 95% Confidence Interval for Intercept (beta_0): {ci_intercept[0]:.4f} - {ci_intercept[1]:.4f}")

# Confidence Interval for Coefficient of X1 (beta_1)
ci_x1 = model.conf_int(alpha=alpha)[1]
print(f"(a) 95% Confidence Interval for Coefficient of X1 (beta_1): {ci_x1[0]:.4f} - {ci_x1[1]:.4f}")

# Confidence Interval for Coefficient of X2 (beta_2)
ci_x2 = model.conf_int(alpha=alpha)[2]
print(f"(a) 95% Confidence Interval for Coefficient of X2 (beta_2): {ci_x2[0]:.4f} - {ci_x2[1]:.4f}")

# Print R-squared
print(f"(a) Coefficient of Determination (R-squared): {model.rsquared:.4f}")

# Print Correlation Coefficient
correlation_coefficient = np.corrcoef(Y, model.fittedvalues)[0, 1]
print(f"(a) Correlation Coefficient: {correlation_coefficient:.4f}")
print(" ")

# Polynomial regression for a parabola (degree 2)
poly_degree = 2
X_poly = np.column_stack([X1, X2, X1**2, X2**2, X1*X2])

# Fit the OLS model for polynomial regression
model_poly = sm.OLS(Y, sm.add_constant(X_poly)).fit()

# Print coefficients for polynomial regression
```

```
print("(b) Coefficients for Polynomial Regression:")
print(model_poly.params)

# Print standard error of the estimate (Sy/x) for polynomial regression
syx_poly = np.sqrt(model_poly.mse_resid)
print(f"(b) Standard Error of the Estimate (Sy/x) for Polynomial Regression: {syx_poly:.4f}")

# Confidence Intervals for polynomial regression
ci_poly = model_poly.conf_int(alpha=alpha)

# Confidence Interval for Intercept (beta_0)
print(f"(b) 95% Confidence Interval for Intercept (beta_0) for Polynomial Regression: {ci_poly[0, 0]:.4f} - {ci_poly[0, 1]:.4f}")

# Confidence Interval for Coefficient of X1 (beta_1)
print(f"(b) 95% Confidence Interval for Coefficient of X1 (beta_1) for Polynomial Regression: {ci_poly[1, 0]:.4f} - {ci_poly[1, 1]:.4f}")

# Confidence Interval for Coefficient of X2 (beta_2)
print(f"(b) 95% Confidence Interval for Coefficient of X2 (beta_2) for Polynomial Regression: {ci_poly[2, 0]:.4f} - {ci_poly[2, 1]:.4f}")

# Confidence Interval for Coefficient of X1^2 (beta_3)
print(f"(b) 95% Confidence Interval for Coefficient of X1^2 (beta_3) for Polynomial Regression: {ci_poly[3, 0]:.4f} - {ci_poly[3, 1]:.4f}")

# Confidence Interval for Coefficient of X2^2 (beta_4)
print(f"(b) 95% Confidence Interval for Coefficient of X2^2 (beta_4) for Polynomial Regression: {ci_poly[4, 0]:.4f} - {ci_poly[4, 1]:.4f}")

# Print R-squared for polynomial regression
print(f"(b) Coefficient of Determination (R-squared) for Polynomial Regression: {model_poly.rsquared:.4f}")

# Print Correlation Coefficient for polynomial regression
correlation_coefficient_poly = np.corrcoef(Y, model_poly.fittedvalues)[0, 1]
print(f"(b) Correlation Coefficient for Polynomial Regression: {correlation_coefficient_poly:.4f}")
```

```
(a) Coefficients:
[14.66666667 -6.66666667  2.33333333]
(a) Standard Error of the Estimate (Sy/x): 1.4142
(a) 95% Confidence Interval for Intercept (beta_0): 12.4479 - 16.8854
(a) 95% Confidence Interval for Coefficient of X1 (beta_1): -8.2142 - -5.1191
(a) 95% Confidence Interval for Coefficient of X2 (beta_2): 1.7015 - 2.9651
(a) Coefficient of Determination (R-squared): 0.9583
(a) Correlation Coefficient: 0.9789

(b) Coefficients for Polynomial Regression:
[ 1.42993827e+01 -6.56944444e+00  2.67129630e+00 -1.38888889e-02
 -4.62962963e-02 -3.24074074e-02]
(b) Standard Error of the Estimate (Sy/x) for Polynomial Regression: 1.9712
(b) 95% Confidence Interval for Intercept (beta_0) for Polynomial Regression: 8.4640 - 20.1347
(b) 95% Confidence Interval for Coefficient of X1 (beta_1) for Polynomial Regression: -16.0230 - 2.8841
(b) 95% Confidence Interval for Coefficient of X2 (beta_2) for Polynomial Regression: -2.1296 - 7.4722
(b) 95% Confidence Interval for Coefficient of X1^2 (beta_3) for Polynomial Regression: -5.5217 - 5.4940
(b) 95% Confidence Interval for Coefficient of X2^2 (beta_4) for Polynomial Regression: -1.0008 - 0.9082
(b) Coefficient of Determination (R-squared) for Polynomial Regression: 0.9595
(b) Correlation Coefficient for Polynomial Regression: 0.9796
```

```
#Question 4
import numpy as np
```

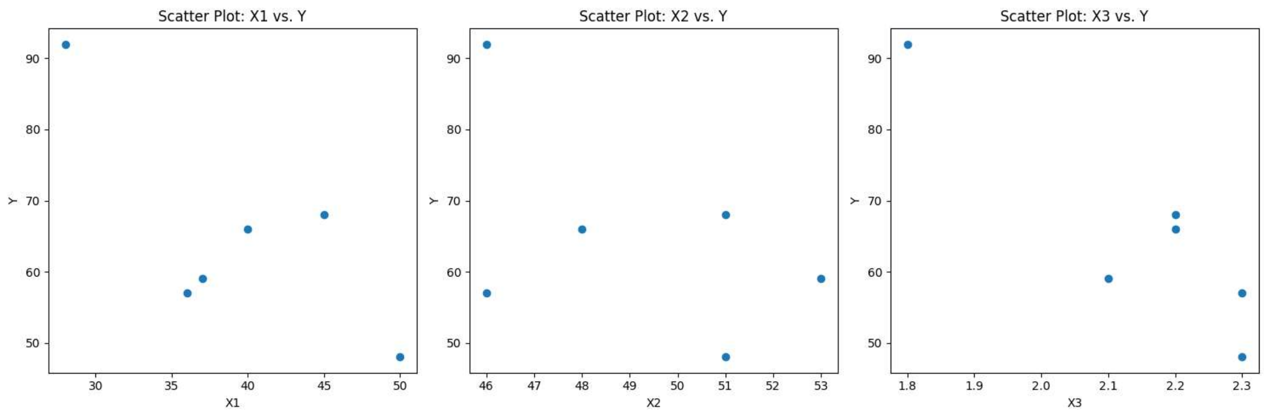
```
import statsmodels.api as sm
from scipy.stats import t

# Given data
X_data = np.array([3, 4, 5, 7, 8, 9, 11, 12])
Y_data = np.array([1.6, 3.6, 4.4, 3.4, 2.2, 2.8, 3.8, 4.6])

# Create design matrix X with a constant term and cubic term
X_poly = np.column_stack([np.ones_like(X_data), X_data, X_data**2, X_data**3])

# Fit the OLS model for polynomial regression
model_poly = sm.OLS(Y_data, X_poly).fit()

# Print coefficients
print("Coefficients for Cubic Equation:")
print(model_poly.params)

# Print R-squared
r_squared = model_poly.rsquared
print(f"R-squared: {r_squared:.4f}")

# Print standard error of the estimate (Sy/x)
syx_poly = np.sqrt(model_poly.mse_resid)
print(f"Standard Error of the Estimate (Sy/x) for Cubic Equation: {syx_poly:.4f}")

# Confidence Intervals for polynomial regression
alpha = 0.05  # 95% confidence interval
t_critical = t.ppf(1 - alpha/2, model_poly.df_resid)

# Confidence Intervals for Coefficients
ci_poly = model_poly.conf_int(alpha=alpha)

# Print Confidence Intervals for Coefficients
for i, coeff in enumerate(model_poly.params):
    print(f"95% Confidence Interval for Coefficient {i}: {ci_poly[i, 0]:.4f} - {ci_poly[i, 1]:.4f}")
```

```
Coefficients for Cubic Equation:
[-11.48870718   7.14381722  -1.04120692   0.04667602]
R-squared: 0.8290
Standard Error of the Estimate (Sy/x) for Cubic Equation: 0.5700
95% Confidence Interval for Coefficient 0: -22.6065 - -0.3709
95% Confidence Interval for Coefficient 1: 1.7963 - 12.4913
95% Confidence Interval for Coefficient 2: -1.8083 - -0.2741
95% Confidence Interval for Coefficient 3: 0.0130 - 0.0804
```

```
#Question 5
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm

# Given data
X1 = np.array([50, 36, 40, 45, 37, 28])
X2 = np.array([51, 46, 48, 51, 53, 46])
X3 = np.array([2.3, 2.3, 2.2, 2.2, 2.1, 1.8])
Y = np.array([48, 57, 66, 68, 59, 92])
```

```
# Scatter plots
plt.figure(figsize=(15, 5))

plt.subplot(131)
plt.scatter(X1, Y)
plt.title('Scatter Plot: X1 vs. Y')
plt.xlabel('X1')
plt.ylabel('Y')

plt.subplot(132)
plt.scatter(X2, Y)
plt.title('Scatter Plot: X2 vs. Y')
plt.xlabel('X2')
plt.ylabel('Y')

plt.subplot(133)
plt.scatter(X3, Y)
plt.title('Scatter Plot: X3 vs. Y')
plt.xlabel('X3')
plt.ylabel('Y')

plt.tight_layout()
plt.show()

# Create design matrix X
X = sm.add_constant(np.column_stack((X1, X2, X3)))

# Fit the OLS model
model = sm.OLS(Y, X).fit()

# Print coefficients
print("Coefficients:")
print(model.params)

# Confidence intervals
ci = model.conf_int(alpha=0.05)
print("\n95% Confidence Intervals:")
print(ci)

# R-squared
r_squared = model.rsquared
print(f"\nR-squared: {r_squared:.4f}")

# Residuals vs. Ŷ plots
plt.figure(figsize=(15, 5))

plt.subplot(131)
plt.scatter(model.fittedvalues, model.resid)
plt.title('Residuals vs. Ŷ')
plt.xlabel('Ŷ')
plt.ylabel('Residuals')

plt.subplot(132)
plt.scatter(X1, model.resid)
plt.title('Residuals vs. X1')
plt.xlabel('X1')
```

```
plt.xlabel('X1')
plt.ylabel('Residuals')

plt.subplot(133)
plt.scatter(X2, model.resid)
plt.title('Residuals vs. X2')
plt.xlabel('X2')
plt.ylabel('Residuals')

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




```
Coefficients:
[307.69877289   0.42723809  -1.86736835 -77.99590405]

95% Confidence Intervals:
[[ -77.03855488  692.43610066]
 [  -3.43403335    4.28850952]
 [  -8.54456025    4.80982354]
 [-209.43792128   53.44611319]]

R-squared: 0.8978
```