01_regression_simple_univariate

May 7, 2025

```
[ ]: import os
     user = os.getenv('USER')
     os.chdir(f'/scratch/cd82/{user}/notebooks')
```

1 Linear Regression -

This workbook is an introduction to the equations and the mathmatics that describe and compute the parameters to fit linear models. The maths used is the 'long-hand' way to get a result uding `NumPy` matrix operations. Packages such as `Scikit-Learn` and `statsmodels` simplify the process and provide a suite of the statistical computations to determine the validity of a proposed model. These notes are just to give you an idea of what is happening 'under the hood' of these packages.

1.0.1 Simple linear regression (no intercept):

$$y = x_1\beta_1 + \epsilon$$

y The dependent variable (response variable) x_1 The independent variable β_1 The regression coefficient (also known as the slope). (In artificial neural networks, β are called the weights) ϵ The residual (error)

In matrix form:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

$\mathbf{y}$ A (column) vector of responses $\mathbf{X}$ A matrix of independent variables (row for each sample, column for each independant variable) β A vector of coefficients.

Equation to solve for β:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.linear_model import LinearRegression
     from sklearn.model_selection import train_test_split
```

Generate Data Our first examples are going to use synthetic data i.e. data derived from a known distribution

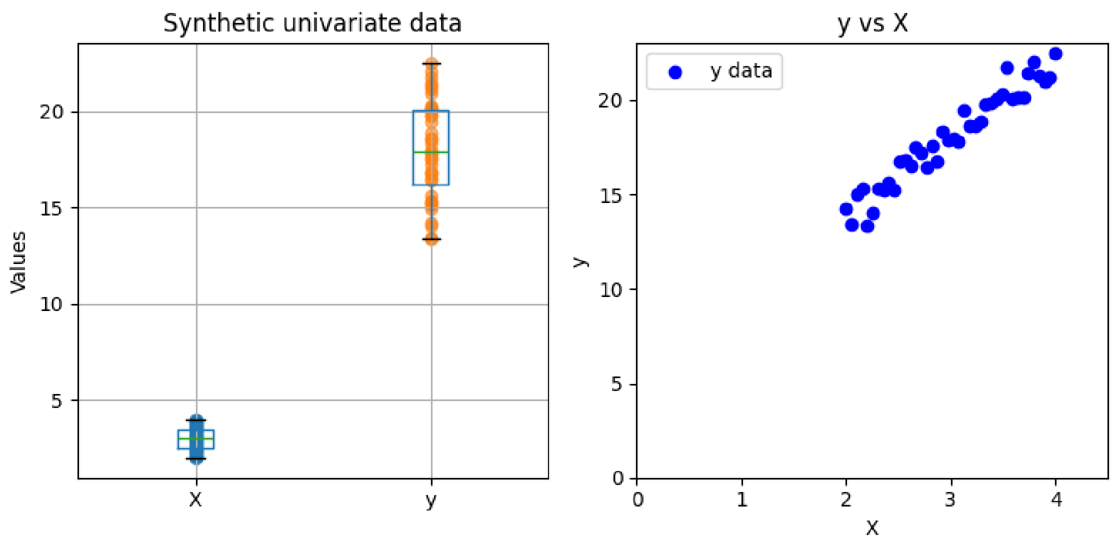
[2]:
```
# Generate sample data

N = 40  # the number of samples to be created

# The seed for the random number generator.
seed_seq = np.random.SeedSequence(42)
# Create a random number generator instance
rng = np.random.default_rng(seed_seq)

rndg = rng.normal(loc=0.0, scale=1, size=N)
rndg = rndg.reshape((N, 1))
print('rndg shape: ', rndg.shape)

# Create X data
start_x = 2.0
range_x = 2.0

X = np.linspace(start_x, start_x+range_x, num=N )
X = X.reshape((N, 1))
print('X shape: ', X.shape)

pc_rand = 0.75  # +- how much randomness to be added to y data

# Create y data
offset_y = 6.0
slope_y = 4.0

add_offset = (start_x * slope_y) + offset_y
# add_offset =  offset_y
y = (( add_offset )+ slope_y * (X - start_x)) + (pc_rand * rndg)
print('y shape: ',y.shape)
```

```
rndg shape:  (40, 1)
X shape:  (40, 1)
y shape:  (40, 1)
```

Plot the data

[3]:
```
x_max = np.max(X)
x_min = np.min(X)
y_max = np.max(y)
y_min = np.min(y)
pe = 0.5  # extend the plot axis by this much in each direction
```

```python
# Create a figure with two subplots side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))

# First subplot: Box and whisker plot with scatter
X_df = pd.DataFrame(X)
y_df = pd.DataFrame(y, columns=['y'])
Xy_df = pd.concat([X_df, y_df], axis=1)
Xy_df.boxplot(ax=ax1)

# Plot the actual points
ax1.scatter(np.ones_like(X), X, alpha=0.6)
ax1.scatter(np.ones_like(X)+1, y, alpha=0.6)

ax1.set_title('Synthetic univariate data')
ax1.set_xticks([1, 2])
ax1.set_xticklabels(['X', 'y'])
ax1.set_ylabel('Values')
ax1.grid(True)

# Second subplot: Scatter plot of y vs X
ax2.scatter(X, y, color='blue', label='y data')
ax2.axis([0.0, x_max+pe, 0.0, y_max+pe])
ax2.set_xlabel('X')
ax2.set_ylabel('y')
ax2.set_title('y vs X')
ax2.legend()

# Adjust layout to prevent overlap
fig.tight_layout()

# Display the plots
plt.show()
```



Create our model We will be fitting the data to a single β coefficient.
Just as a reminder:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

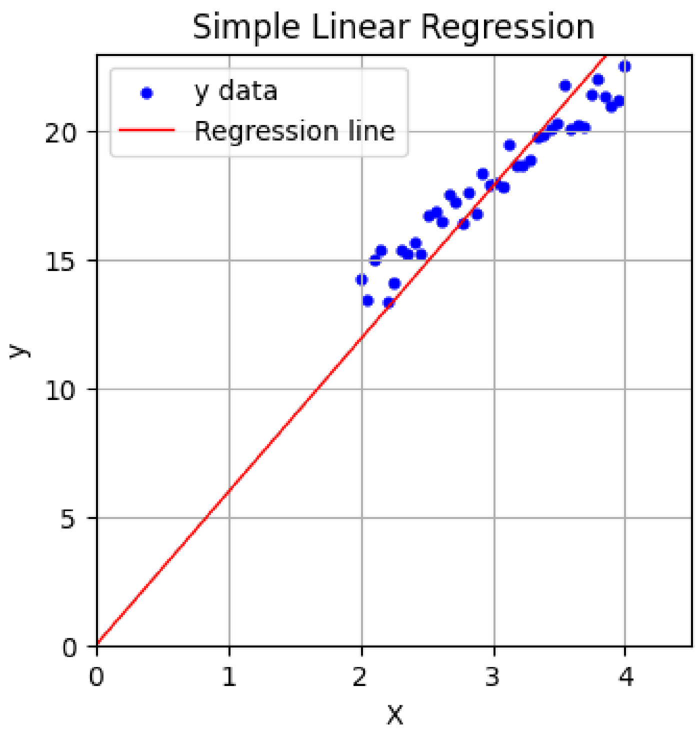
[4]:
```
# Compute the beta
# We are using the whole data set here
dot_product = np.dot(np.transpose(X), X)
betas_noint = np.linalg.inv(dot_product) * np.dot(np.transpose(X), y)
print('betas[0]: ', betas_noint[0])

# Predict our dependent value with our model
y_pred = betas_noint[0] * X
```

```
betas[0]:  [5.93862516]
```

Plot the results

[5]:
```
plt.figure(figsize=(4, 4))

# Plot the results
plt.scatter(X, y, color='blue', s=12, label='y data')
# plt.scatter(X, y_pred, color='red',marker='x',  label='predicted y')

X_extended = np.linspace(0, max(X), 100)
y_extended = betas_noint[0] * X_extended
```

```
plt.plot(X_extended, y_extended, color='red', linewidth=1.0, label='Regression␣
↪line')
plt.axis([0.0, x_max+pe, 0.0, y_max+pe])

# Define the points for the rise and run
rx = [0.0, 1.0 ]  # x-coordinates
ry = [0.0, 0.0]   # y-coordinates

rux = [ 1.0, 1.0]   # x-coordinates
ruy = [0.0, betas_noint[0][0]]   # y-coordinates

# Plot the line
# plt.plot(rx, ry, marker='o', label='run (1.0)')
# plt.plot(rux, ruy, marker='o', label='rise␣
↪('+str(round(betas_noint[0][0],3))+')')
plt.grid(True)

plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```



1.1 Linear regression, R^2 coefficient of determination:

R^2 is the measure of variance in the dataset explained by the independent variable.

$$R^2 = 1 - \frac{\sum_i^m (y_i - \hat{y}_i)^2}{\sum_i^m (y_i - \bar{y})^2}$$

- m Is the number of samples (y values)
- $\hat{y}_i$ Is the predicted value of y of sample i
- $\bar{y}$ Is the mean value of y

R^2 is the amount of variance (as a proportion of the total variance) that the independent variable explains in the variation of the depenent variable. Please note, R^2 is the Pearsons correlation coefficient squared for a single independent variable, however it is the sum of the variance of each additional independent variable, so the equivalence ends with multiple predictor variables.

Where can we find a function in Python:

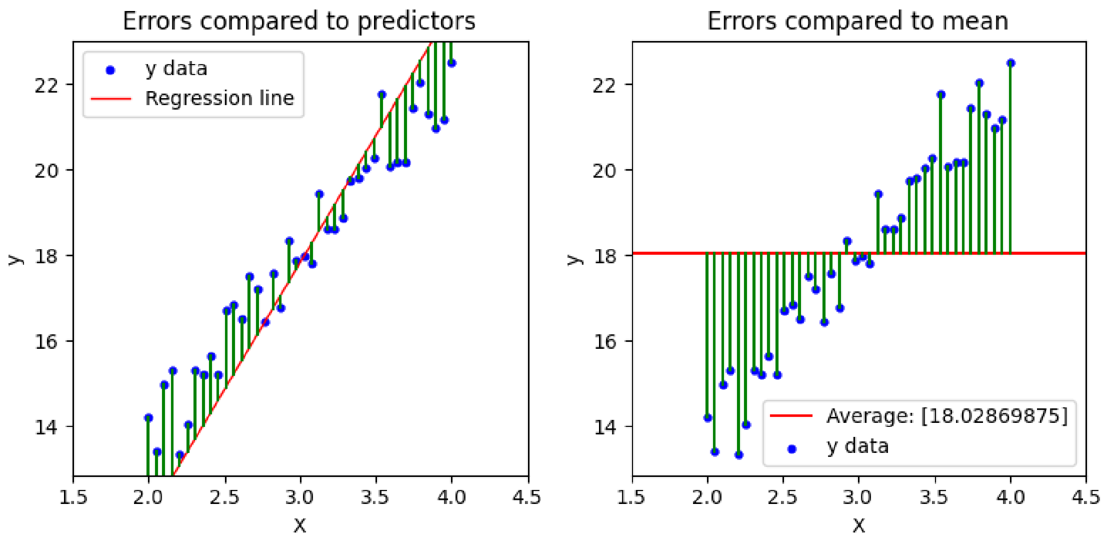
```
from sklearn.metrics import r2_score
rsc = r2_score (y, y_pred)
```

1.1.1 Plots to show values of interest in the R^2computation

The following plots show the differences between the data and the predictions and the data and the estimate of the mean.

[6]:
```
x_max = np.max(X)
x_min = np.min(X)
y_max = np.max(y)
y_min = np.min(y)

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)

plt.scatter(X, y, color='blue', label='y data',s=12)
# plt.scatter(X, y_pred, color='red',  label='model predicted y')
plt.plot(X, y_pred, color='red', linewidth=1.0, label='Regression line')

# Plot the error lines
for i in range(len(y)):
    plt.plot([X[i], X[i]], [y_pred[i], y[i]], color='green')

plt.axis([x_min-pe, x_max+pe, y_min-pe, y_max+pe])
plt.xlabel('X')
plt.ylabel('y')
plt.title('Errors compared to predictors')
plt.legend()
```

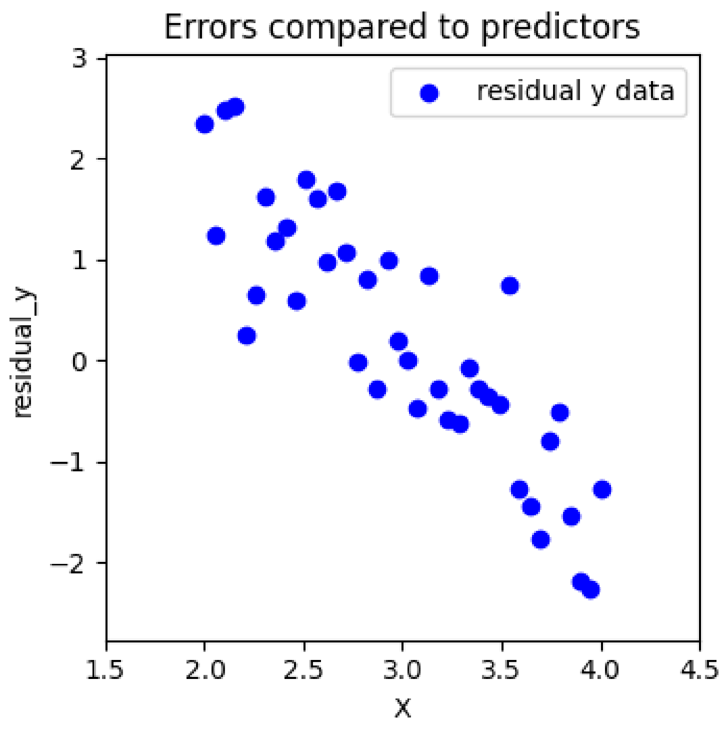
```python
# Plot the results showing the difference between the data and the estimate of␣
↪beta
plt.subplot(1, 2, 2)

average_y = sum(y) / len(y)
# Plot the average line
plt.axhline(y=average_y, color='r', linestyle='-', label=f'Average:␣
↪{average_y}')
plt.scatter(X, y, color='blue', label='y data',s=12)
plt.axis([x_min-pe, x_max+pe, y_min-pe, y_max+pe])

for i in range(len(y)):
    plt.plot([X[i], X[i]], [average_y, y[i]], color='green')

plt.xlabel('X')
plt.ylabel('y')
plt.title('Errors compared to mean')
plt.legend()

plt.tight_layout()
```



2 Plot of the errors (the residuals)

A model that has randomly distributed residuals (the η values) typically means that the model has accounted for the variation in the data.
Plotting of the errors (the differences between the prediction and the actual data) can visually indicate if there is further *structure* to the data that is not being accounted for.

[7]:
```
average_y = sum(y) / len(y)

residual_y = y - y_pred
residual_y_ave = y - average_y

x_max = np.max(X)
x_min = np.min(X)
y_max = np.max(residual_y)
y_min = np.min(residual_y)

plt.figure(figsize=(4, 4))


plt.scatter(X, residual_y, color='blue', label='residual y data')

plt.axis([x_min-pe, x_max+pe, y_min-pe, y_max+pe])
plt.xlabel('X')
plt.ylabel('residual_y')
plt.title('Errors compared to predictors')
plt.legend()

# Plot the results showing the difference between the data and the estimate of␣
↪beta

plt.tight_layout()
```



$$R^2 = 1 - \frac{\sum_i^m (y_i - \hat{y}_i)^2}{\sum_i^m (y_i - \bar{y})^2}$$

```
[8]: # This is the Coefficient of determination
     from sklearn.metrics import r2_score
     # This is the Pearsons R function
     from sklearn.feature_selection import r_regression


     ss_residual_y = np.dot(np.transpose(residual_y), residual_y)
     ss_residual_y_ave = np.dot(np.transpose(residual_y_ave), residual_y_ave)

     R_sqrd = 1.0 - (ss_residual_y/ ss_residual_y_ave)
     print('R squared (computed above): ',R_sqrd)

     # We can compare our computed value with what Scikit-Learn computes
     rsc = r2_score (y.ravel(), y_pred)
     r_pearson = r_regression(y_pred, y.ravel())  # Pearsons correlation␣
      ↪(standardized variance)
     print('R squared (SKL): ', rsc)
```

```
R squared (computed above):  [[0.74986138]]
R squared (SKL):  0.7498613828583759
```

Why do I have a negative R^2 The ratio seen in the R^2 calculation can result in spurious values if the *sums of differences from the mean* are smaller than the *sums of differences from predicted* data.

2.0.1 Linear regression, with addition of the slope coefficient

$$y = \beta_0 + x_1\beta_1 + \epsilon$$

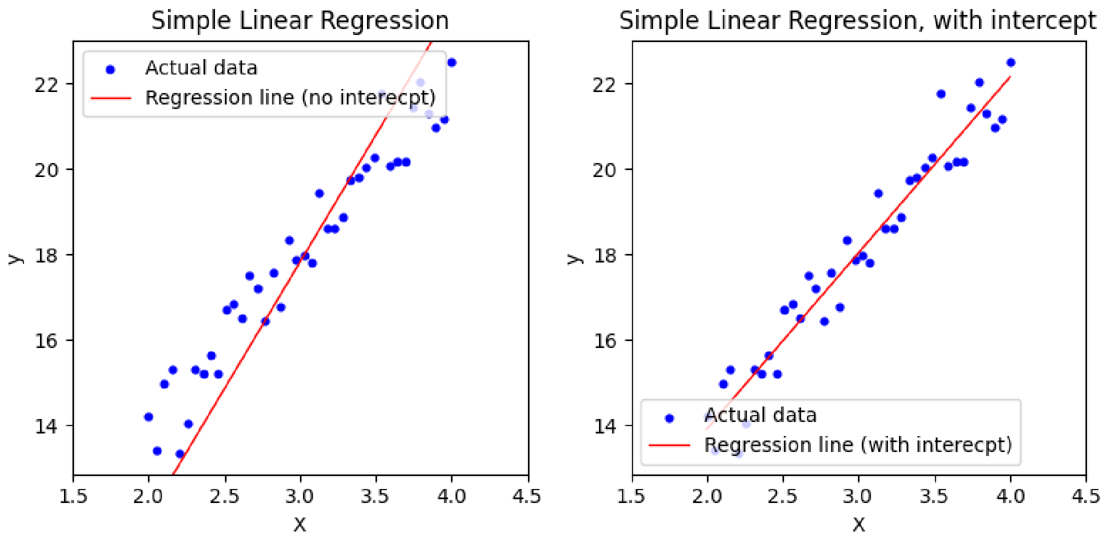
$ { }_{0}$ Added as the intercept (also known as the bias or offset)

```
[9]: # To model the intercept, we need to add a column of 1's to the X data
     X_intercept = np.hstack([np.ones((X.shape[0], 1)), X])
     print('Shape(X)       : ',np.shape(X_intercept))

     # Perform the matrix operation X^T x X
     dot_product = np.dot(np.transpose(X_intercept), X_intercept)
     print('Shape(X^T x X): ', np.shape(dot_product))

     # Compute the coefficients
```

```
betas = np.dot(np.linalg.inv(dot_product), np.dot(np.transpose(X_intercept), y))
print('Shape(betas)  : ',np.shape(betas))
print('betas[0] (intercept)  : ', betas[0])
print('betas[1]              : ', betas[1])
```

```
Shape(X)      :  (40, 2)
Shape(X^T x X):  (2, 2)
Shape(betas)  :  (2, 1)
betas[0] (intercept)  :  [5.67874796]
betas[1]              :  [4.11665026]
```

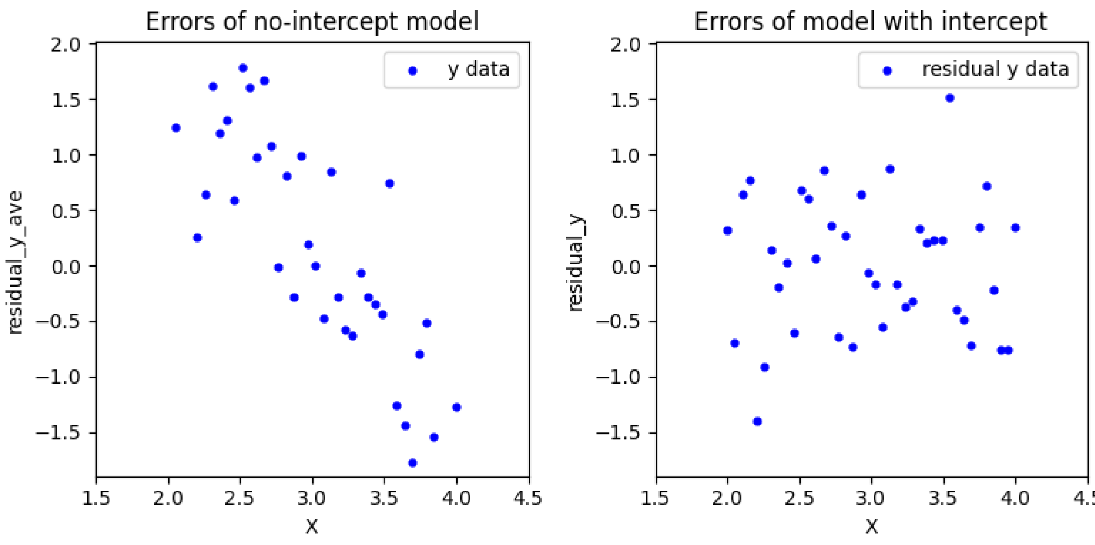
[10]:
```
x_max = np.max(X)
x_min = np.min(X)
y_max = np.max(y)
y_min = np.min(y)


# Plot the results - compare the two models, without and with an intercept
plt.figure(figsize=(8,4))
plt.subplot(1, 2, 1)
# Plot the results
average_y = sum(y) / len(y)
# Plot the average line
# plt.axhline(y=average_y, color='r', linestyle='--', label=f'Average:␣
 ↪{average_y}')

y_pred = betas_noint[0] * X
plt.scatter(X, y, color='blue', label='Actual data',s=12)
plt.plot(X, y_pred, color='red',
         linewidth=1, label='Regression line (no interecpt)')
plt.axis([x_min-pe, x_max+pe, y_min-pe, y_max+pe])
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()


# Model with the intercept
y_pred_i = betas[0] + betas[1] * X
plt.subplot(1, 2, 2)
plt.scatter(X, y, color='blue', label='Actual data',s=12)
plt.plot(X, y_pred_i, color='red',
         linewidth=1, label='Regression line (with interecpt)')
plt.axis([x_min-pe, x_max+pe, y_min-pe, y_max+pe])
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression, with intercept')
plt.legend()

plt.tight_layout()
```



Plot the residuals for new model We will view the ϵ values from the model equation:

$$y = \beta_0 + x_1\beta_1 + \epsilon$$

[11]:
```
average_y = sum(y) / len(y)
residual_y_i = y - y_pred_i

x_max = np.max(X)
x_min = np.min(X)
y_max = np.max(residual_y_i)
y_min = np.min(residual_y_i)


plt.figure(figsize=(8,4))
plt.subplot(1, 2, 1)

# Plot single coefficient model errors
plt.scatter(X, residual_y, color='blue', label='y data',s=12)
plt.axis([x_min-pe, x_max+pe, y_min-pe, y_max+pe])

plt.xlabel('X')
plt.ylabel('residual_y_ave')
plt.title('Errors of no-intercept model')
plt.legend()

# Plot the results showing the difference between the data and the estimate of␣
↪beta
plt.subplot(1, 2, 2)
plt.scatter(X, residual_y_i, color='blue', label='residual y data',s=12)

plt.axis([x_min-pe, x_max+pe, y_min-pe, y_max+pe])
plt.xlabel('X')
plt.ylabel('residual_y')
plt.title('Errors of model with intercept')
plt.legend()


plt.tight_layout()
```



The Shapiro-Wilk test for normality We can use Scipy library `shapiro` function to test if the residuals are normal

[12]:
```
from scipy.stats import shapiro

# test of model with no intercept
stat, p_value = shapiro(residual_y)

print("Shapiro-Wilk Test Statistic (no intercept) :", stat)
print("p-value:", p_value)

stat_i, p_value_i = shapiro(residual_y_i)
```

```
print("Shapiro-Wilk Test Statistic (with intercept):", stat_i)
print("p-value:", p_value_i)

# Values closer to 1 indicate normality, and p values e.g. > 0.05, means␣
 ↪normality cannot be rejected.
```

```
Shapiro-Wilk Test Statistic (no intercept) : 0.9790877382583247
p-value: 0.6556867809435238
Shapiro-Wilk Test Statistic (with intercept): 0.9811723530213634
p-value: 0.7329321038414265
```

The Pearson's r coefficient For simple univariate regression, with an intercept, we can compute the R^2 value by squaring the Pearson's r correlation:

$$r = \frac{cov(x,y)}{\sigma(x)\cdot\sigma(y)}$$

$cov(x,y) = \frac{\sum(x-\bar{x})(y-\bar{y})}{m-1}$

$\sigma(x) = \sqrt{\frac{\sum(x-\bar{x})^2}{m-1}}$

$$R^2 = r^2$$

[13]:
```
from sklearn.feature_selection import r_regression

def persons_r_ours(x, y):
    xy_cov = np.cov(x,y, rowvar=False, bias=True)
    x_var = (np.std(x))  # the standard deviation
    y_var = (np.std(y))
    r = xy_cov[1,0] / (x_var * y_var)
    return r

r_xy1 = persons_r_ours(X, y)
print('r_xy1 : ', r_xy1)



# This is the Scikit-Learn Pearson's r function
r_pearson = r_regression(X, y.ravel())
print('r_pearson SKL: ', r_pearson)
```

```
r_xy1 :  0.9702872712667591
r_pearson SKL:  [0.97028727]
```

[14]:
```
# This is the Coefficient of determination
from sklearn.metrics import r2_score
# This is the Pearsons R function
```

```
from sklearn.feature_selection import r_regression

rsc_2 = r2_score (y.ravel(), y_pred_i)

print('R squared (SKL): ', rsc_2)
print('R squared (Pearsons sqrd): ', r_xy1* r_xy1)
```

```
R squared (SKL):  0.9414573887822925
R squared (Pearsons sqrd):  0.9414573887822933
```

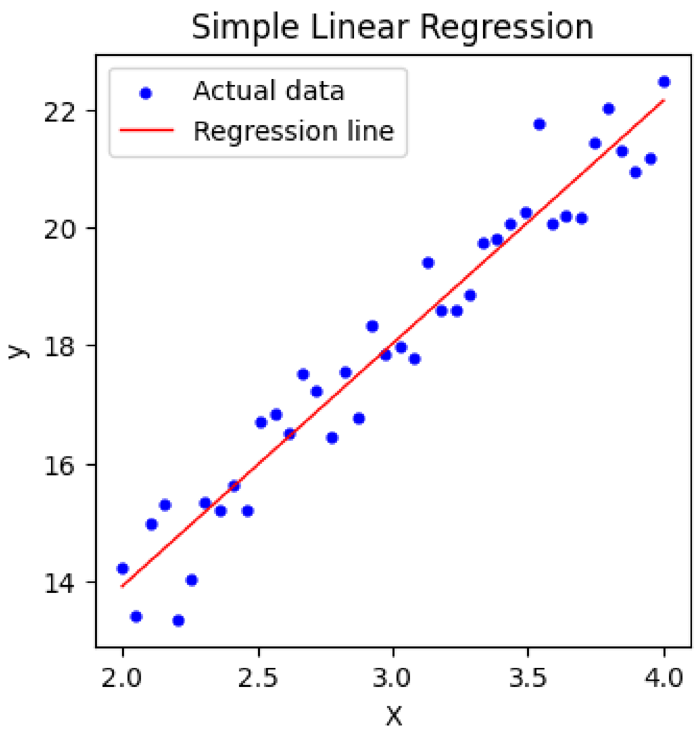
2.0.2 Using higher level libraries

`Scikit-Learn` and `statsmodels` libraries provide a wide range of regression methods, pre-configured to compute statistical tests and provide metriscs for the model created.

[15]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Train the model
model_skl = LinearRegression()
model_skl.fit(X, y)
```

[15]:
```
LinearRegression()
```

[16]:
```
y_pred_skl = model_skl.predict(X)

plt.figure(figsize=(4, 4))
plt.scatter(X, y, color='blue', label='Actual data',s=12)
plt.plot(X, y_pred_skl, color='red',
         linewidth=1, label='Regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```



[17]:
```
# Evaluate the model and compare with our original results
mse = mean_squared_error(y, y_pred_skl)
r2 = r2_score(y, y_pred_skl)

print(f"Intercept: {model_skl.intercept_}")
print(f"Coefficient: {model_skl.coef_}")

print('betas[0] (intercept)   : ', betas[0])
print('betas[1]               : ', betas[1])
```

```
Intercept: [5.67874796]
Coefficient: [[4.11665026]]
betas[0] (intercept)   :   [5.67874796]
betas[1]               :   [4.11665026]
```

Using the statsmodels library

[18]:
```
import statsmodels.api as sm

X_int= sm.add_constant(X)
ols_model = sm.OLS(y, X_int)

results_sm = ols_model.fit()
# Print the summary of the model
```

```python
print(results_sm.summary())

# Make predictions - If we had some other data
# y_pred_sm = results_sm.predict(X_test)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.941
Model:                            OLS   Adj. R-squared:                  0.940
Method:                 Least Squares   F-statistic:                     611.1
Date:                Wed, 07 May 2025   Prob (F-statistic):           5.05e-25
Time:                        09:38:46   Log-Likelihood:                -36.834
No. Observations:                  40   AIC:                             77.67
Df Residuals:                      38   BIC:                             81.04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.6787      0.509     11.152      0.000       4.648       6.710
x1             4.1167      0.167     24.720      0.000       3.780       4.454
==============================================================================
Omnibus:                        0.063   Durbin-Watson:                   2.055
Prob(Omnibus):                  0.969   Jarque-Bera (JB):                0.254
Skew:                           0.054   Prob(JB):                        0.881
Kurtosis:                       2.625   Cond. No.                         17.4
==============================================================================

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

In the results from an Ordinary Least Squares (OLS) regression using the `statsmodels` package, several test statistics are provided to help you understand the model's performance and the significance of the predictors. Here are some key test statistics and what they tell us: (Summary of outputs thanks to CoPilot)

Coefficients (coef): These are the estimated values for the regression parameters. They represent the change in the dependent variable for a one-unit change in the predictor variable, holding other variables constant.

Standard Errors (std err): These measure the variability of the coefficient estimates. Smaller standard errors indicate more precise estimates.

t-Statistics (t): These are calculated as the coefficient divided by its standard error. They test the null hypothesis that the coefficient is equal to zero (no effect). A larger absolute value of the t-statistic indicates stronger evidence against the null hypothesis.

P-values (P>|t|): These indicate the probability of observing the given t-statistic, assuming the null hypothesis is true. Smaller p-values (typically < 0.05) suggest that the corresponding predictor is statistically significant.

R-squared (R^2): This measures the proportion of the variance in the dependent variable that is explained by the independent variables. Values range from 0 to 1, with higher values indicating a better fit.

Adjusted R-squared: This is a modified version of R-squared that adjusts for the number of predictors in the model. It is useful for comparing models with different numbers of predictors.

F-statistic: This tests the overall significance of the model. It compares the model with no predictors (intercept only) to the specified model. A higher F-statistic indicates that the model provides a better fit than the intercept-only model.

Prob (F-statistic): This is the p-value associated with the F-statistic. It tests the null hypothesis that all regression coefficients are equal to zero. A small p-value suggests that at least one predictor is significantly related to the dependent variable.

Akaike Information Criterion (AIC) and **Bayesian Information Criterion** (BIC): These are measures of model quality that penalize for the number of predictors. Lower values indicate a better model fit.

Summary In this workbook we have: - Created a synthetic dataset that relates y to X, with some added noise. - Using Numpy matrix primitives, we created two competing models:
$y = x_1\beta_1 + \epsilon$
$y = \beta_0 + x_1\beta_1 + \epsilon$ - We compare their R^2 values which compares the predicted values to the simplest model, that being the mean of the data ($\bar{y}$). - We discussed some possible issues with R^2 and we looked at Pearson's r as an alternative for univariate models with an intercept. - We plotted the data, the predicted regression model and the residuals to check our results. - We used the `Scikit-Learn` library to perform the modelling and showed the predicted β values were the same. - We used the `statsmodels` library and also got a lot of summary statistics to go with the model.

```
[ ]:
```