In [24]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
df = pd.read_csv('income.data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	income	happiness
0	1	3.862647	2.314489
1	2	4.979381	3.433490
2	3	4.923957	4.599373
3	4	3.214372	2.791114
4	5	7.196409	5.596398

In [4]:
```
df.drop(columns='Unnamed:  0', inplace=True)
```

In [5]:
```
df.head()
```

Out[5]:

	income	happiness
0	3.862647	2.314489
1	4.979381	3.433490
2	4.923957	4.599373
3	3.214372	2.791114
4	7.196409	5.596398

Implementing Statistical Simple Linear Regression from scratch

$$\hat{y} = \widehat{B_0} + \widehat{B_1}\, x$$

$$\widehat{B_1} = {SXY}/{SXX}$$

$$SXY = \sum xy - n\,\overline{x}\,\overline{y}$$

$$SXX = \sum x^2 - n\,\overline{x}^2$$

$$\widehat{B_0} = \overline{y} - \widehat{B_1}\,\overline{x}$$

$$SSE = \sum(y - \hat{y})^2$$

$$MSE = {SEE}/{n-2}$$

```
In [6]: y = df['happiness']
        x = df['income']
```

```
In [7]: y_bar = y.mean()
        x_bar = x.mean()
        n = len(y)
```

```
In [8]: SXX = sum(x**2) - n * x_bar**2
        SXY = sum(x * y) - n * x_bar * y_bar
```

```
In [9]: B_1 = SXY / SXX
```

```
In [10]: B_0 = y_bar - B_1 * x_bar
```

```
In [17]: print(f'y = {B_0:.3f} + {B_1:.3f} x')
```

```
y = 0.204 + 0.714 x
```

```
In [12]: y_hat = B_0 + B_1 * x
```

```
In [13]: e = y - y_hat
```

```
In [14]: SSE = sum(e**2)
```

```
In [15]: MSE = SSE / (n-2)
```

```
In [27]: print(f'MSE = {MSE:.3f}')
```

```
MSE = 0.516
```

In [18]:

Implementing Statistical Simple Linear Regression from scratch

$$\hat{y} = \widehat{B_0} + \widehat{B_1}\, x$$

$$\widehat{B_1} = {SXY}/{SXX}$$

$$\widehat{B_0} = \bar{y} - \widehat{B_1}\,\bar{x}$$

$$SXY = \sum xy - n\,\bar{x}\,\bar{y}$$

$$SSE = \sum (y - \hat{y})^2$$

$$SXX = \sum x^2 - n\,\bar{x}^2$$

$$MSE = {SEE}/{n-2}$$

```
class SimpleLinearRegression:

    def __init__(self):
        """
        Initializes the SimpleLinearRegression model.

        Attributes:
        -----------
        B_0 : float
```

```
            The intercept of the regression line.
        B_1 : float
            The slope of the regression line.
        MSE : float
            Mean squared error.
        """
        self.B_0 = None
        self.B_1 = None
        self.MSE = None

    def fit(self, X, y):
        """
        Fits the simple linear regression model to the training data.
        Calculates and sets the intercept (B_0) and slope (B_1) of the
        regression line and then calculates the evaluation metrics for
        the model which is the MSE.

        Parameters:
        -----------
        X : array-like
            The input feature array.
        y : array-like
            The target variable array.
        """
        # 1. Ensure X and y are numpy arrays
        X = np.array(X)
        y = np.array(y)

        # 2. Calculate the mean of X and y
        x_bar = X.mean()
        y_bar = y.mean()

        # 3. Calculate SXX and SXY
        SXX = sum(X**2) - len(X) * x_bar**2
        SXY = sum(X * y) - len(X) * x_bar * y_bar

        # 4. Calculate coefficients
        self.B_1 = SXY / SXX
        self.B_0 = y_bar - self.B_1 * x_bar

        # 5. Calculate estimated error
        y_hat = self.predict(X)
        e = y - y_hat
        SSE = sum(e**2)
        self.MSE = SSE / (len(X) - 2)

    def predict(self, X):
        """
        Predicts the target variable for new input values using the fitted model.

        Parameters:
        -----------
        X : array-like
            The input feature array for which to predict the target variable.

        Returns:
```

```
        --------
        array-like
            Predicted values of the target variable.

        Raises:
        -------
        ValueError
            If the model has not been fitted yet.
        """
        # 1. Raise a ValueError if the model has not been fitted yet
        if self.B_0 is None or self.B_1 is None:
            raise ValueError("The model has not been fitted yet.")

        # 2. Ensure that X is a numpy array
        X = np.array(X)

        # 3. Return the estimated y
        return self.B_0 + self.B_1 * X

    def plot(self, X, y):
        """
        Plots the scatter plot of the data points and the regression line.

        Parameters:
        -----------
        X : array-like
            The input feature array.
        y : array-like
            The target variable array.
        """
        if self.B_0 is None or self.B_1 is None:
            raise ValueError("The model has not been fitted yet.")

        # Ensure X and y are numpy arrays
        X = np.array(X)
        y = np.array(y)

        # Predicted values
        y_pred = self.predict(X)

        # Plot
        plt.scatter(X, y, color='blue', label='Data points')
        plt.plot(X, y_pred, color='red', label='Regression line')
        plt.xlabel('X')
        plt.ylabel('y')
        plt.title(f'Simple Linear Regression with line of best fit: y = {round(self
        plt.legend()
        plt.show()
```

Taking an object and fitting our data to it

In [19]:

```
model = SimpleLinearRegression()
model.fit(x, y)
```

In [26]:
```
print("Evaluation of the model")
print(" ------------------------------------------ ")
print(f'Line of best fit is: y = {model.B_0:.3f} + {model.B_1:.3f} x')
print(f'Mean squared error is: {model.MSE:.3f}')
```

```
Evaluation of the model
-------------------------
Line of best fit is: y = 0.204 + 0.714 x
Mean squared error is: 0.516
```

In [22]:
```
predictions = model.predict([6, 7])
print(predictions)
```

```
[4.48722347 5.20104898]
```

In [23
```
model.plot(x, y)
```