In [1]:
```
import pandas as pd
import numpy as np
import scipy.stats
```

```
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\computation\expression
s.py:20: UserWarning: Pandas requires version '2.7.3' or newer of 'numexp
r' (version '2.7.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
```

Test of Independence

$H_0: B_1 = 0 \; (no \; relationship \; between \; x \; and \; y)$

$H_a: B_1 \neq 0 \; (there's \; a \; relationship)$

1) Get F_0 from ANOVA table
2) Get F_c from F table $F_c = F(\alpha, n-2)$ where α is the level of significance
3) Compare F_0 and F_c, if $F_0 > F_c$ we reject the null hypothesis which means that there is a relationship between the independent variable and the response

---

Interval Estimation

$$\hat{B}_0 \sim N(B_0, \sigma^2\left(\frac{1}{n} + \frac{\overline{x}}{SXX}\right))$$

When σ^2 is known, $B_0: \hat{B}_0 \pm Z_{\frac{\alpha}{2}} \sqrt{\sigma^2\left(\frac{1}{n} + \frac{\overline{x}}{SXX}\right)}$

When it's not, $B_0: \hat{B}_0 \pm t_{\frac{\alpha}{2}, n-2} \sqrt{MSE\left(\frac{1}{n} + \frac{\overline{x}}{SXX}\right)}$

$$\hat{B}_1 \sim N(B_1, \frac{\sigma^2}{SXX})$$

When σ^2 is known, $B_1: \hat{B}_1 \pm Z_{\frac{\alpha}{2}} \sqrt{\frac{\sigma^2}{SXX}}$

When it's not, $B_1: \hat{B}_1 \pm t_{\frac{\alpha}{2}, n-2} \sqrt{\frac{MSE}{SXX}}$

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

In [3]:
```
df.drop(columns='Unnamed: 0', inplace=True)
```

In [4]:
```
df.head()
```

Out[4]:

	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

In [5]:
```
y = df['happiness']
x = df['income']
```

In [6]:
```
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.
        r_squared : float
            Coefficient of determination.
        """
        self.B_0 = None
        self.B_1 = None
        self.MSE = None
        self.r_squared = None

    def fit(self, X, y):
        """
        Fits the simple linear regression model to the training data. Calcu
lates and sets the intercept (B_0) and
        slope (B_1) of the regression line and then calculates the evaluati
on metrics for the model which are
        MSE and r_squared.

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

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

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

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

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

```
        # Calculate coefficient of determination
        self.SST = sum((y - y_bar)**2)
        self.SSR = sum((y_hat - y_bar)**2)
        self.r_squared = self.SSR / self.SST

    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 variabl
e.

        Returns:
        --------
        array-like
            Predicted values of the target variable.

        Raises:
        -------
        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.")

        X = np.array(X)
        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 = {ro
und(self.B_0,2)} + {round(self.B_1,2)} x')
        plt.legend()
```

```
        plt.show()

    def anova_table(self):
        """
        Constructs and returns the ANOVA table.

        Returns:
        --------
        pd.DataFrame
            The ANOVA table showing the Sum of Squares, Degrees of Freedom, 
Mean Squares, F-Statistic,
            and p-value for the model.
        """
        if self.B_0 is None or self.B_1 is None:
            raise ValueError("The model has not been fitted yet.")

        # Degrees of freedom
        self.df_regression = 1
        self.df_error = self.n - 2
        df_total = self.n - 1

        # Mean squares
        MSR = self.SSR / self.df_regression
        MSE = self.SSE / self.df_error

        # F-statistic
        self.F_stat = MSR / MSE

        # Assemble the ANOVA table
        anova_data = {
            'Source': ['Regression', 'Error', 'Total'],
            'Sum of Squares': [self.SSR, self.SSE, self.SST],
            'Degrees of Freedom': [self.df_regression, self.df_error, df_to
tal],
            'Mean Square': [MSR, MSE, ""],
            'F-Statistic': [self.F_stat, "", ""]
        }

        anova_table = pd.DataFrame(anova_data)
        return anova_table


    def hypothesis_test(self, alpha=0.05):
        """
        Perform a hypothesis test to determine the relationship between the 
independent variable (x)
        and the dependent variable (y) based on the F-statistic.

        Parameters:
        -----------
        alpha : float
            The significance level for the test. Default value = 0.05

        Returns:
        --------
        None
            The function prints the hypothesis test results, including the 
F-statistic,
            critical value, and the conclusion.

        Notes:
```

```
        ------
        The null hypothesis (H_0) assumes that there is no relationship bet
ween x and y,
        while the alternative hypothesis (H_a) suggests that there is a rel
ationship.
        """
        # Calculate the critical value for the given significance level
        F_c = scipy.stats.f.ppf(1 - alpha, self.df_regression, self.df_erro
r)

        # Determine the conclusion based on the F-statistic
        if self.F_stat > F_c:
            conclusion = ("Since F_0 > F_c, we reject the null hypothesi
s.\n"
                          "Therefore, there's a relationship between x and
y.")
        else:
            conclusion = ("Since F_c > F_0, we don't reject the null hypoth
esis.\n"
                          "Therefore, there's no relationship between x and
y.")

        # Print the results in a structured and visually appealing format
        print("="*45)
        print("         Hypothesis Testing Results           ")
        print("="*45)
        print(f"{'Null Hypothesis (H_0):':<25} B_1 = 0")
        print(f"{'Alternative Hypothesis (H_a):':<25} B_1 ≠ 0")
        print("-"*45)
        print(f"{'F-statistic (F_0):':<25} {self.F_stat:.4f}")
        print(f"{'Critical value (F_c):':<25} {F_c:.4f}")
        print("-"*45)
        print(f"{conclusion}")
        print("="*45)

    def interval_estimation(self, alpha=0.05, sigma=None):

        if sigma is None:
            t = scipy.stats.t.ppf(1-(alpha/2), self.df_error)
            B_0_lower = self.B_0 - t * np.sqrt(self.MSE*((1/self.n) + (sel
f.x_bar/self.SXX)))
            B_0_upper = self.B_0 + t * np.sqrt(self.MSE*((1/self.n) + (sel
f.x_bar/self.SXX)))
            B_1_lower = self.B_1 - t * np.sqrt(self.MSE/self.SXX)
            B_1_upper = self.B_1 + t * np.sqrt(self.MSE/self.SXX)

        else:
            z = scipy.stats.norm.ppf(1-(alpha/2))
            B_0_lower = self.B_0 - z * sigma * np.sqrt((1/self.n) + (self.x
_bar/self.SXX))
            B_0_upper = self.B_0 + z * sigma * np.sqrt(((1/self.n) + (self.
x_bar/self.SXX)))
            B_1_lower = self.B_1 - z * sigma * np.sqrt(1/self.SXX)
            B_1_upper = self.B_1 + z * sigma * np.sqrt(1/self.SXX)


        return ([B_0_lower, B_0_upper],[B_1_lower, B_1_upper])
```

In [7]:

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

In [8]:
```
print("Evaluation of the model")
print("-----------------------")
print(f'Line of best fit is: y = {round(model.B_0,2)} + {round(model.B_1,2)} x')
print(f'Mean squared error is: {round(model.MSE,3)}')
print(f'Coefficient of determination is: {round(model.r_squared,3)}')
```

```
Evaluation of the model
-----------------------
Line of best fit is: y = 0.2 + 0.71 x
Mean squared error is: 0.516
Coefficient of determination is: 0.749
```

In [9]:
```
model.anova_table()
```

Out[9]:

	Source	Sum of Squares	Degrees of Freedom	Mean Square	F-Statistic
0	Regression	764.546359	1	764.546359	1482.632007
1	Error	255.771488	496	0.515668	
2	Total	1020.317847	497		

In [10]:
```
model.hypothesis_test()
```

```
=============================================
         Hypothesis Testing Results
=============================================
Null Hypothesis (H_0):     B_1 = 0
Alternative Hypothesis (H_a): B_1 ≠ 0
---------------------------------------------
F-statistic (F_0):         1482.6320
Critical value (F_c):      3.8603
---------------------------------------------
Since F_0 > F_c, we reject the null hypothesis.
Therefore, there's a relationship between x and y.
=============================================
```

In [11]:
```
B_0_est, B_1_est = model.interval_estimation()
print(f"95% CI for B_0: {B_0_est}")
print(f"95% CI for B_1: {B_1_est}")
```

```
95% CI for B_0: [0.1046540279959846, 0.3038867644124284]
95% CI for B_1: [0.6774017746996904, 0.7502492498607113]
```

In [12]:
```
B_0_est, B_1_est = model.interval_estimation(sigma=x.std())
print(f"95% CI for B_0: {B_0_est}")
print(f"95% CI for B_1: {B_1_est}")
```

```
95% CI for B_0: [-0.03617479319616132, 0.44471558560457436]
95% CI for B_1: [0.6259091122071826, 0.8017419123532191]
```

In [ ]:
```

```