08_Linear_regression

November 2, 2025

1 08-Linear regression

This notebook shows how to perform simple and multiple linear regression in Python.

So far, we have seen how to do simple data analysis, e.g. `value_counts`, `describe`, `corr` etc.

In linear regression, we take the analysis one step further by estimating a line that quantifies the relationship between variables in our data.

OLS (ordinary least squares) is the most common method for finding the regression line. In OLS, we find the intercept and slope of the line by minimizing the sum of the squared *residuals*, i.e, the difference between the predicted value by the line and the actual value.

We will use the OLS estimator from the package `statsmodels`. The sub-module `formula` allows us to fit statistical models using R-style formulas.

We import `statsmodels` by giving it the shorter name `smf`.

[ ]:
```
import statsmodels.formula.api as smf
import pandas as pd
import matplotlib.pyplot as plt
```

[ ]:
```
plt.style.use('ggplot')
```

[ ]:
```

```

1.1 Simple linear regression

In simple linear regression, we use data to estimate the relationship between a variable y and a single variable x:

$y_i = + x_i $

where: - y is the dependent variable - x is the explanatory (independent) variable - α is the constant term (i.e., intercept) - β is the slope of the regression line

Let us explore the variation in cars' fuel economy (mpg).

[ ]:
```
mpg_df = pd.read_excel('data/mpg.xlsx')

mpg_df.head()
```

[ ]:
```
len(mpg_df)
```

By applying `corr` on the `DataFrame`, we see that there is a strong negative relationship between fuel economy and horsepower (-0.778).

[ ]:
```
mpg_df.corr(numeric_only = True)
```

[ ]:
```
correlation = round(mpg_df.corr(numeric_only = True).loc['horsepower', 'mpg'],␣
↪2)

print(correlation)
```

Let us create a scatter plot between `mpg` and `horsepower`, and add the correlation coefficient as a title to the plot.

[ ]:
```
fig, ax = plt.subplots()

ax.scatter(mpg_df['horsepower'], mpg_df['mpg'])

# Add axis labels
ax.set_xlabel('horsepower')
ax.set_ylabel('mpg')

# Add title
ax.set_title('R = ' + str(correlation))

plt.show()
```

However, note that there are a few missing observations in our data.

[ ]:
```
mpg_df.isna().sum()
```

Before estimating a statistical model, we should use `dropna` in order to drop all rows with missing observations.

When applying `dropna` on a `DataFrame`, it will drop *all* rows with `NaN`. Therefore, we specify `subset = ['mpg', 'horsepower']` so that we only drop the rows with `NaN` in the columns that we will be using in our linear regression model.

[ ]:
```
mpg_df.dropna(subset = ['mpg', 'horsepower'], axis = 0, inplace = True)

len(mpg_df)
```

We are estimating the following model:

$mpg_i = \alpha + \beta \times horsepower_i$

We start by transforming the model to a R-style formula. Note that we do not have to add a constant term (i.e., intercept) to the model formula as it will be added automatically by `statsmodels`.

[ ]:
```
formula = 'mpg ~ horsepower'

print(formula)
```

We create an OLS model by using the `ols` function from `statsmodels`.

`ols` requires two inputs: 1. model formula 2. data that we want to estimate the model with

[ ]:
```
# Create OLS model
model = smf.ols(formula, data = mpg_df)
```

After we have created the OLS model, we must apply `fit` on our model object in order to actually *estimate* the model.

[ ]:
```
# Estimate model
model = model.fit()
```

[ ]:
```
model
```

In order to see the regression results, we must apply `summary` on our fitted model object.

[ ]:
```
model.summary()
```

The table contains a lof of information that is stored in the attributes of our fitted model object.

- The `param` attribute contains the model coefficients, i.e., intercept and slope:

[ ]:
```
model.params
```

[ ]:
```
model.params['horsepower']
```

- The `bse` attribute contains the standard errors of the coefficients:

[ ]:
```
model.bse
```

- The `rsquared` and `rsquared_adj` attributes of contains the model's R-squared and adjusted R-squared.

[ ]:
```
model.rsquared
```

[ ]:
```
model.rsquared_adj
```

> **Note:** The R-squared measures the share of variation in the dependent variable (y) that is explained by our model. In this example, over 60% of the variation in cars' fuel economy in our data can be explained by the number of horsepower alone. Not bad for such a simple model!

Your turn

The file survey_data.csv contains the results from a survey of 2,884 individuals regarding their earnings. Import the file and store it in a variable called df_wage:

1. Estimate a simple linear regression model in which hourly earnings (column hourly_earnings) is the dependent variable and years of schooling (column years_schooling) is the explanatory variable. Store the model in a variable called mod_wage.
2. Print the model's β coefficient. Does hourly earnings change a lot for each additional year of schooling?
3. Print the model's adjusted R-squared. Does years of schooling appear to be a good predictor of variation in hourly earnings?

PS: Don't overwrite previous variable names.

[ ]:

In-sample prediction Once we have estimated the OLS model, can use it to predict values of the dependent variable given observations of the explanatory variable. By creating these predictions, it also becomes easy to visualize the regression line.

To generate predictions, we apply `predict` on the fitted model object.

`predict` will calculate the predicted value for the dependent variable using the estimates from the regression model and the observed value on the explanatory variable for each observation in the data that was used to estimate the model. This is known as *in-sample* prediction.

We can store the predictions in the original `DataFrame` by assigning the predicted values to a new column.

[ ]:
```
# Generate and store in-sample predictions
mpg_df['pred'] = model.predict(mpg_df)

mpg_df.head()
```

Let us create a line plot of `pred` on the y-axis, and `horsepower` on the x-axis. This will give us the regression line.

[ ]:
```
mpg_df.sort_values('horsepower', inplace = True)
```

[ ]:
```
fig, ax = plt.subplots()

# Line plot
ax.plot(mpg_df['horsepower'], mpg_df['pred'], color = 'black', label = 'OLS␣
 ↪line')

# Add axis labels
ax.set_xlabel('horsepower')
ax.set_ylabel('mpg')

# Add title and legend
ax.set_title(formula)
ax.legend()

plt.show()
```

We can add a scatter plot of actual `mpg` and `horsepower` in the same plot with the regression line.

[ ]:
```
fig, ax = plt.subplots()

# Scatter plot
ax.scatter(mpg_df['horsepower'], mpg_df['mpg'])

# Line plot
ax.plot(mpg_df['horsepower'], mpg_df['pred'], color = 'black', label = 'OLS␣
 ↪line')

# Add axis labels
ax.set_xlabel('horsepower')
ax.set_ylabel('mpg')

# Add title and legend
ax.set_title(formula)
ax.legend()

plt.show()
```

Your turn

Use the simple linear regression model from the previous exercise to generate in-sample predictions of hourly earnings given the observed years of schooling for each respondent in the survey data.

Show a scatter plot with years of schooling on the x-axis and hourly earnings on the y-axis, and add the regression line to the plot using the in-sample predictions.

[ ]:
```

```

We can also inspect visually how well our fitted model explains variation in fuel economy: - First, we plot *actual* `mpg` against *actual* `mpg`. This will create a 45 degree line. - Second, we add a scatter plot of *actual* `mpg` and *predicted* `pred`.

The idea is that the closer the predictions in the scatter plot are to the 45 degree line, the better job does our model at explaining the variation in fuel economy.

[ ]:
```
fig, ax = plt.subplots()

# 45 degree line
mpg_df.sort_values('mpg', inplace = True)
ax.plot(mpg_df['mpg'], mpg_df['mpg'], color = 'black', label = '45-degree line')

# Scatter plot
ax.scatter(mpg_df['mpg'], mpg_df['pred'])

# Formatting
ax.set_xlabel('Actual mpg')
ax.set_ylabel('Predicted mpg')
```

```
ax.set_title(formula)
ax.legend()

plt.show()
```

There are three potential cases for the observations in the scatter plot: - Observation is below the 45 degree line → the model *underpredicts* `mpg` - Observation is on the 45 degree line → the model correctly predicts `mpg` - Observation is above the 45 degree line → the model *overpredicts* `mpg`

From the plot, we see that the model does a good job at predicting fuel economy at low levels of `mpg`. However, at high levels of `mpg`, the model systematically underpredicts the fuel economy of the car.

Your turn

Use the in-sample predictions from the previous exercise, and show a scatter plot with actual hourly earnings on the x-axis and predicted hourly earnings on the y-axis. Add a 45 degree line of actual hourly earnings in the plot.

Does it seem like the linear regression model does a good job at predicting hourly earnings based only on years of schooling? Or does the model systematically over- or underpredict hourly earnings for some respondents?

[ ]:
```

```

There are several other potential explanatory variables of fuel economy in our data. However, if we want to use a different variable than `horsepower`, we have two options:

- Re-write our program above using a new explanatory variable. Boring…
- Write a function that estimates an OLS model for any given explanatory variable. Fun

Let us create two functions:

1. `get_predictions` estimates an OLS model given two inputs: 1) a model formula and 2) data. The function creates in-sample predictions and returns a *copy* of the original data with the predictions.

[ ]:
```
def get_predictions(formula, df):
    """Fit a linear regression model given a model formula and return df with␣
 ↪in-sample predictions."""

    # Copy dataframe (important! Otherwise, we change the original df)
    df_copy = df.copy()

    # Create and fit OLS model
    model = smf.ols(formula, data = df_copy).fit()

    # Add predictions to copied dataframe
    df_copy['pred'] = model.predict(df_copy)

    return df_copy
```

2. `plot_predictions` plots the predictions along with the 45 degree line given two inputs: 1) a model formula and 2) data.

[ ]:
```
def plot_predictions(formula, df):
    """Plot in-sample predictions along 45-degree line given a model formula."""

    # Get yvar from formula
    depvar = formula.split('~')[0].strip()

    fig, ax = plt.subplots()

    # 45 degree line
    df.sort_values(depvar, inplace = True)
    ax.plot(df[depvar], df[depvar], color = 'black', label = '45-degree line')

    # Scatter plot
    ax.scatter(df[depvar], df['pred'])

    # Formatting
    ax.set_xlabel(depvar)
    ax.set_ylabel('pred')
    ax.set_title(formula)
    ax.legend()

    plt.show()
```

Let us re-run the estimation above, but using our new functions instead.

We re-importing the data to remove columns with predictions from previous models.

[ ]:
```
mpg_df = pd.read_excel('data/mpg.xlsx')

mpg_df.head()
```

- Step 1: estimate the model and get in-sample predictions

[ ]:
```
# Define formula
formula = 'mpg ~ weight'

# Estimate model and get in-sample predictions (store in new df)
mpg_df_new = get_predictions(formula, mpg_df)

mpg_df_new.head()
```

[ ]:
```
# Note that original data has not been altered (because we copied the df)
# mpg_df.head()
```

- Step 2: Plot predictions along the 45 degree line

[ ]:
```
# Plot predictions
plot_predictions(formula, mpg_df_new)
```

Note: Recall that it is possible to pass the output of a function call directly as an input to another function call in Python.

[ ]:
```
formula = 'mpg ~ weight'
plot_predictions(formula, get_predictions(formula, mpg_df))
```

[ ]:
```

```

1.2 Multiple linear regression

In multiple linear regression, we expand the simple model to include multiple explanatory variables:

$y_i = \alpha + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3}$ ...

In general, we can increase the explanatory power of our model (i.e., the R-squared) by including more explanatory variables.

[ ]:
```
mpg_df = pd.read_excel('data/mpg.xlsx')
mpg_df.dropna(inplace = True)

mpg_df.head()
```

`corr` shows that there is also a high correlation between a car's fuel economy and its weight (-0.832).

[ ]:
```
mpg_df.corr(numeric_only = True)
```

We will now estimate the following model:

$mpg_i = \alpha + \beta_1 \times horsepower_i + \beta_2 \times weight_i$

We expand the model formula with weight as an explanatory variable.

[ ]:
```
formula = 'mpg ~ horsepower + weight'
```

[ ]:
```
smf.ols(formula, data = mpg_df).fit().summary()
```

Let us use the functions that we created above to generate the prediction plot for the new model with the two explanatory variables `horsepower` and `weight`.

[ ]:
```
plot_predictions(formula, get_predictions(formula, mpg_df))
```

If the new model does a good job at explaining the variation in fuel economy, we would expect the predictions to be close to actual observations. However, we see that the model still underpredicts fuel economy for high levels of mpg despite including a second explanatory variable.

Side note: When a linear regression model includes more than one predictor (i.e., explanatory variable), we can no longer generate the regression line from the in-sample predictions.

[ ]:
```
# Generate predictions from regression model with two explanatory variables
formula = 'mpg ~ horsepower + weight'
```

```
df_temp = get_predictions(formula, mpg_df)
df_temp.sort_values('horsepower', inplace = True)

fig, ax = plt.subplots()

# Scatter plot
ax.scatter(df_temp['horsepower'], df_temp['mpg'])

# Line plot
ax.plot(df_temp['horsepower'], df_temp['pred'], color = 'black', label = 'OLS␣
 ↪line')

# Add axis labels
ax.set_xlabel('horsepower')
ax.set_ylabel('mpg')

# Add title and legend
ax.set_title(formula)
ax.legend()

plt.show()
```

Your turn

Expand the simple linear regression model from previous exercises to also take into account the number of years of work experience:

- Estimate the model: $hourly\_earnings_i = \alpha + \beta_1 \times years\_schooling_i + \beta_2 \times experience_i$
- Print the model's adj. R-squared. Does including years of experience improve the explanatory power of the model?
- Generate the in-sample predictions from the model and display a scatter plot of actual vs predicted hourly earnings along the 45-degree line.

[ ]:
```

```

Polynomials Polynomial regression is a good option when the relationship between the dependent and explanatory variables does not seem to be linear. Instead of estimating a regression line, we fit a *curve* to the data.

In fact, in our fuel economy data, there seems to be a non-linear relationship between cars' fuel economy and horsepower.

[ ]:
```
mpg_df = pd.read_excel('data/mpg.xlsx')
mpg_df.dropna(subset = ['horsepower', 'mpg'], inplace = True)

fig, ax = plt.subplots()

ax.scatter(mpg_df['horsepower'], mpg_df['mpg'])
```

```
ax.set_xlabel('horsepower')
ax.set_ylabel('mpg')

plt.show()
```

Instead of estimating a linear regression model, we now estimate a polynomial model. Specifically, we estimate a 2nd order polynomial model by adding the *square* of the explanatory variable to our model formula:

$mpg_i = \alpha + \beta_1 \times horsepower_i + \beta_2 \times horsepower_i^2$

In R-style formula, we can include the squared explanatory variable by adding the term `I(horsepower**2)`.

[ ]:
```
f_poly = 'mpg ~ horsepower + I(horsepower**2)'
```

[ ]:
```
# Estimate OLS model
model_poly = smf.ols(f_poly, mpg_df).fit()

# Model summary
model_poly.summary()
```

We can again use our function `get_predictions` to generate a new `DataFrame` with the in-sample predictions, but now from our polynomial model.

[ ]:
```
# Create in-sample predictions
mpg_df = get_predictions(f_poly, mpg_df)

mpg_df.head()
```

We can then use `plot_predictions` to visually inspect how well the model predicts.

[ ]:
```
plot_predictions(f_poly, mpg_df)
```

From the plot, it seems that 2nd order polynomial model does a better job at predicting fuel economy compared to the simple regression model, although the model still slightly underpredicts fuel economy at high levels of `mpg`.

Your turn

Expand the multiple linear regression model from the previous exercise with the following polynomial term:

$hourly\_earnings_i = \alpha + \beta_1 \times years\_schooling_i + \beta_2 \times experience_i + \beta_3 \times experience_i^2$

Display a scatter plot of actual vs predicted hourly earnings from the model along the 45 degree line. Does it seem like adding the polynomial term has improved the model's ability to predict the hourly earnings of the respondents?

[ ]:
```

```

Side note: Although our model of fuel economy is no longer a simple regression model due to the second polynomial term, note that we are technically still using only one variable (i.e., horsepower)

to explain variation in fuel economy. In that case, we can visualize the regression line using the predicted values the same way as before.

[ ]:
```
mpg_df.sort_values('horsepower', inplace = True)
```

[ ]:
```
fig, ax = plt.subplots()

# Line plot
ax.plot(mpg_df['horsepower'], mpg_df['pred'], color = 'black', label = 'OLS␣
 ↪line')

# Scatter plot
ax.scatter(mpg_df['horsepower'], mpg_df['mpg'])

# Formatting
ax.set_xlabel('horsepower')
ax.set_ylabel('mpg')
ax.set_title(f_poly)
ax.legend()

plt.show()
```

Categorical variables So far, we have only used *numerical* variables as explanatory variables in our regression models.

However, in data analysis, we often work with *categorical* data, i.e., observations are categories/labels and not numbers. For example, in timeseries data, we cannot say that the value "January" is smaller or larger than the value "July", or that "1am" is more or less than "3pm".

However, we can still use non-numerical variables as a explanatory variables in regression models, but we must include then as *categorical* variables. Such variables are also known as factor or indicator variables.

Note that our `mpg` data contains the categorical variable `origin`. In fact, the average fuel economy is very different for cars from the US compared to cars from Europe and Japan, and `origin` could therefore be an important explanatory variable.

[ ]:
```
mpg_df.groupby('origin')['mpg'].mean()
```

In R-style formula, we can include categorical variables by adding the term `C(origin)`.

[ ]:
```
f_cat = 'mpg ~ horsepower + C(origin)'
```

> **Note:** By adding categorical variables in a regression model, we create a different constant term for each group/category in the data, i.e., the regression line has a different intercept for each group.

From the regression results below, we see that there is still a negative relationship between `mpg` and `horsepower`.

However, there is now a different intercept for American, Japanese and European cars. The intercept for European cars is 38.3695, while the intercept is +2.7510 higher for Japanese cars and -2.4253 lower for American cars. This means that Japanese cars tend to be more fuel efficient than European cars, and European cars are more fuel efficient than American cars, even when they have the same number of horsepower.

[ ]:
```
smf.ols(f_cat, mpg_df).fit().summary()
```

As before, we can use our function `plot_predictions` to plot the predictions from the model with `origin` along the 45 degree line.

[ ]:
```
plot_predictions(f_cat, get_predictions(f_cat, mpg_df))
```

Your turn

Estimate a multiple linear regression model of hourly earnings that also takes into account whether the respondent works in the public or private sector. Use the survey data and estimate the model

$$hourly\_earnings_i = \alpha + \beta_1 \times years\_schooling_i + \beta_2 \times experience_i + \beta_3 \times sector_i,$$

where $sector_i$ is a categorical variable ("private" or "public").

On average, how much less do respondents in the public sector earn compared to respondents in the private sector?

[ ]:
```

```

Categorical vs numerical In reality, datasets often have variables that can be interpreted both as categorical and numerical.

For example, in `mpg`, should we interpret `cylinders` as a numerical variable or a categorical variable? Although the variable is numerical (int), it has only a few unique values and could potentially by included as a categorical variable in a regression model.

[ ]:
```
mpg_df['cylinders'].unique()
```

Let us first estimate a regression model with both `horsepower` and `cylinders` as numerical variables.

[ ]:
```
# Define model formula
f_cyl_num = 'mpg ~ horsepower + cylinders'

# Estimate OLS model and show output
model_cyl_num = smf.ols(f_cyl_num, mpg_df).fit()
model_cyl_num.summary()
```

Alternatively, we can instead estimate a model in which we include `cylinders` as a categorical variable.

[ ]:
```
# Define model formula
f_cyl_cat = 'mpg ~ horsepower + C(cylinders)'

```

```
# Estimate OLS model and show output
model_cyl_cat = smf.ols(f_cyl_cat, mpg_df).fit()
model_cyl_cat.summary()
```

Despite both models including `cylinders` as an explanatory variable, we see that including `cylinders` as a categorical variable actually increases the explanatory power of our model (i.e., adj. R-squared) from 65.5 to 70.1%

[ ]:
```
print(round(model_cyl_num.rsquared_adj, 3))
```

[ ]:
```
print(round(model_cyl_cat.rsquared_adj, 3))
```

[ ]:
```

```

2 Home exercises

2.0.1 Exercise 1: Outliers in hourly earnings

The file `survey_data.csv` contains information on the hourly earnings (in DKK) of 2,884 respondents. In statistical analysis, the presence of outliers (i.e., extreme values) can have a large impact on the results of the estimation and how well the model predicts the observed outcome. Therefore, in this exercise, you will investigate the presence of outliers in the survey data and its effect on the estimates from a linear regression model.

Task 1: Load the data and do the following: - Calculate the number of respondents that had an hourly wage of less than 10 DKK or above 1000 DKK. - Calculate the average hourly wage for males and females in the private and public sector. - Create a single plot that shows histograms of the hourly earnings for males and females seperately.

Task 2: Create a function called `get_beta` that estimates a regression model and returns the beta coefficient for a specific explanatory variable. The function should take three inputs: `df` (the dataset), `formula` (formula for the regression model), and `exp` (column name of an explanatory variable).

Test the function by estimating the following regression model

$hourly\_earnings_i = \alpha + \beta_1 \times years\_schooling_i + \beta_2 \times experience_i + \beta_3 \times experience_i^2$,

and print the β-coefficient on the number of years of schooling.

Task 3: There are some respondents in the data that have an extremely high or low hourly wage. We want to explore how much dropping a single observation, i.e., respondent, from our data affects the estimated coefficient on years of schooling in the regression model from the previous task.

1. Write a `for` loop where you in each iteration:
   - drop an observation from the data
   - use `get_beta` to retrieve the coefficient on years of schooling
   - store the coefficient in a list

   Note: In the first iteration you should drop the first respondent from the data. In the second iteration you should keep the first respondent but drop the second respondent. In the third

iteration you should keep the first and second respondents, but drop the third one, and so on…

2. Use the list with the estimated coefficients on years of schooling from the previous task and display the distribution of the coefficients in a histogram. What is your verdict? Does it seem that the β coefficient on `years_schooling` is affected by the presence of outliers?

2.0.2 Exercise 2: Fuel economy and polynomials

We have estimated a 2nd order polynomial model in which we used the number of horsepower to explain variation in fuel economy. However, there could also be a non-linear relationship between fuel economy and other car attributes. Including polynomial terms can often improve the explanatory power of our regression models. Therefore, in this exercise, you will explore the adjusted R-squared from using different car attributes in a 2nd order polynomial model.

Task 1: Create a function called `get_rsqr` that estimates a regression model and returns the model's adjusted R-squared. The function should take two inputs: `df` (the dataset) and `formula` (formula for the regression model). Import the `mpg` data and test the function by estimating the model

$mpg_i = \alpha + \beta_1 \times horsepower_i + \beta_2 \times horsepower_i^2$,

and print the adjusted R-square from the model.

Task 2: We now want to compare the adjusted R-squared from the 2nd order polynomial model in the previous task, but using four different car attributes: `horsepower`, `weight`, `acceleration` and `model_year`.

Write a `for` loop where you in each iteration: - Update the model formula for the polynomial model to include one of the four car attributes - Use the function `get_rsqr` to get the adjusted R-squared from the model. - Print the adjusted R-squared from each of the polynomial models

Note: In each iteration, the polynomial model should include only one car attribute. In the first iteration, the model should use `horsepower`; in the second iteration, the model should use `weight`; and so on.

Which 2nd order polynomial model has the highest adj. R-squared?

Task 3: In addition to comparing the adjusted R-squared, we also want to inspect the estimated regression line from each of the polynomial models with the four different car attributes: `horsepower`, `weight`, `acceleration` and `model_year`.

Create a single graph with four subplots side-by-side (1x4). In each subplot: - Show a scatter plot with one of the car attributes on the x-axis and `mpg` on the y-axis. - Show the regression line using the in-sample predictions from the polynomial model with the car attrbitue - Add the adjusted R-squared from the polynomial model in the title of the sub-plot

Hint: Use a `for` loop to iterate over the axes object to avoid duplicating the code for generating each subplot. Note also that you can use the function `get_predictions` from the lecture to get a `DataFrame` with the in-sample predictions for each polynomial model.

2.0.3 Exercise 3: Drivers of CO2 emissions

In this exercise, you are asked to explore CO2 emissions around the world, and potential drivers for why some countries have higher emissions than other countries. To do this, you are given two datasets.

The first file `co2_emissions.csv` contains the following country-level data on CO2 emissions in 2021: - `country`: Country name - `year`: Year of observation - `co2_total`: Total carbon dioxide (CO2) emissions (Mt CO2e) - `population`: Total population

In addition, the file contains data on the following six potential explanatory variables of country-level CO2 emissions: - `urban`: Urban population (% of total population) - `gdp_pc`: GDP per capita (current US$) - `electricity`: Access to electricity (% of population) - `agriculture`: Agricultural land (% of land area) - `nat_resources`: Total natural resources rents (% of GDP) - `renew_energy`: Renewable energy consumption (% of total final energy consumption)

Note that the emissions dataset contains data not only for countries, but also for aggregates such as "Africa Eastern and Southern" and "Heavily indebted poor countries (HIPC)".

The second file `country_info.csv` contains information about the countries and aggregates observed in the emissions dataset: - `name`: Name of the location (country or aggregate) - `region`: Region of the location - `incomeLevel`: Income level of the location (e.g., "Low income")

Task 1: Create a dataset that contains countries only:

1. Import and merge the two datasets. Explore the merged data, e.g., data types, missing values, unique values etc.
2. Drop all observations that are not countries, e.g., "Africa Eastern and Southern" so that the data contains observations for countries only.
3. Create a new column called `co2_pc`, which is the *per capita* CO2 emissions (t CO2e/capita) for each country.

   Hint: Multiple total CO2 emissions with 1,000,000 to convert from million tons to tons (otherwise, you'll get very small numbers).

Task 2: Use the country-level dataset from the previous task to visualize CO2 emissions across countries:

1. Create a figure with two subplots:

   - In the first subplot, show a scatter plot of total population (`population`) and per capita CO2 emissions (`co2_pc`).
   - In the second subplot, show a histogram of the distribution of per capita CO2 emissions (`co2_pc`).

   From the plots, are there any outliers in the data?

2. Create a figure with 6 subplots (either 2x3 or 3x2):

   - In each subplot, show a scatter plot of per capita CO2 emissions (`co2_pc`) and one of the potential explanatory variables of emissions (`urban`, `gdp_pc`, `electricity`, `agriculture`, `nat_resources`, `renew_energy`).

- Add the correlation coefficient between the explanatory variable and per capita CO2 emissions in the title of the subplot.

Hint: To avoid using a nested `for` loop to generate the 2x3 or 3x2 plot, you can apply the `flatten` method on the `ax` object to create a one-dimensional object that you can use a single `for` loop to iterate over.

Task 3: You will now estimate a multiple linear regression model where per-capita CO2 emissions (`co2_pc`) is the depdenent variable, and the model includes three out of the six potential explanatory variables: `urban`, `gdp_pc`, `electricity`, `agriculture`, `nat_resources`, `renew_energy`.

Note: Do not include any polynomial terms in the model.

In general, there are 20 possible combinations when you can choose three explanatory variables from six different variables. Your task is to find the combination of three variables that has the highest adjusted R-squared.

Write a `for` loop in which you loop over the 20 possible combinations of three explanatory variables. For each possible combination: - Estimate the linear regression model: $co2\_pc_i = \alpha + \beta_1 \times X1_i + \beta_2 \times X2_i + \beta_3 \times X3_i$ - Extract the adjusted R-squared from the model

Which combination of explanatory variables has the highest adjusted R-squared?

Hint: The function `combinations` from `itertools` can be used to generate all possible combinations from a set of values (see here).

[ ]:

[ ]: