intro-to-pandas-world-happiness

November 7, 2024

1 Exploratory Data Analysis - Intro to Pandas

Welcome to the Pandas tutorial lab. This is the first notebook of the exploratory data analysis (EDA) series, where you will get your hands dirty applying the skills you have learned in the course on an actual data problem, similar to those you might encouter in real life! Here you will see and try out some basics of Pandas and get familiar with some of the useful functions that you will use across the other labs and assignments. If you already know Pandas well, feel free to skip this notebook.

For the demonstration purposes you will use the World Happiness Report dataset. The dataset consists of 2199 rows, where each row contains various hapiness-related metrics for a certain country in a given year. Right now you'll just use this dataset to understand some fundamental operations in Pandas. You will see this dataset again later in week 3, where you will dig deeper into the data and explore relationships to better understand which factors seem to best predict happiness.

This notebook is not a comprehensive guide to Pandas, but rather shows and explains the functions you will use through this course. For a more comprehensive guide on Pandas, please see the official tutorial or check the documentation.

2 1. Importing the Libraries

The most important library you will need in this notebook is - you guessed it - `Pandas`. You will also use the `Seaborn` library for plotting the data. To import the libraries run the cell below.

[1]:
```
# Import the Pandas library
import pandas as pd
# Import the Seaborn library for plotting
#!pip install seaborn
import seaborn as sns
```

3 2. Importing the Data

Now that you have the pandas library imported, you'll need to load your dataset. The dataset you will use is saved as a `.csv` file and all you need to do to load is call the function `pd.read_csv(filename)`. If you have your data in another format, there exists a variety of functions to load it, you can check the documentation here. When you load the dataset, it will be stored as a `DataFrame` type (see the documentation here). This is the most commonly used Pandas datastructure that you will use throughout this and other notebooks.

[2]:
```python
# Load the dataset and save it to the df variable
df = pd.read_csv('data/world_happiness.csv')
```

4 3. Basic Operations With a Dataframe

4.1 3.1 View the Dataframe

You can use `DataFrame.head()` and `DataFrame.tail()` to view the first or last rows of the frame respectively. By default it will show you five rows, but you can specify the number of rows you want to see as a parameter. Technically, neither of the functions actually display anything, but just return a new dataframe. The dataframe is displayed because Jupyter notebooks show the output of the last row in the cell. You can also display the contents of your dataframe by simply writing `df`. If your dataframe is too long, it will then display only the first and the last few rows.

Note that all of this only works if you use it in the last line of code in the cell, because the cells automatically display the output of the last line. If you want to see more than one dataframe by running a single cell or if you want to perform some other tasks after displaying the dataframe, then you better encapsulate it with `print()` or `display()`. `display()` function will print the dataframe, but with the same format as just calling `df`, whereas `print()` will print as plain text.

Try commenting and uncommenting lines below, to see how this plays out. Try different combiations of rows.

[4]:
```python
# This line will display the first few rows of the dataframe if there are no␣
↪lines of code after.
df.head()

# Try uncommenting different combinations of the lines below.
# print("Cats are cool.")
# print(df.head())
# print(df)
print("Some more text about cats being cool.")
display(df)
```

```
Some more text about cats being cool.

     Country name  year  Life Ladder  Log GDP per capita  Social support  \
0     Afghanistan  2008        3.724               7.350           0.451
1     Afghanistan  2009        4.402               7.509           0.552
2     Afghanistan  2010        4.758               7.614           0.539
3     Afghanistan  2011        3.832               7.581           0.521
4     Afghanistan  2012        3.783               7.661           0.521
…               …   …           …                   …             …
2194     Zimbabwe  2018        3.616               7.783           0.775
2195     Zimbabwe  2019        2.694               7.698           0.759
2196     Zimbabwe  2020        3.160               7.596           0.717
2197     Zimbabwe  2021        3.155               7.657           0.685
2198     Zimbabwe  2022        3.296               7.670           0.666
```

```
      Healthy life expectancy at birth  Freedom to make life choices  \
0                               50.500                         0.718
1                               50.800                         0.679
2                               51.100                         0.600
3                               51.400                         0.496
4                               51.700                         0.531
…                                 …                             …
2194                            52.625                         0.763
2195                            53.100                         0.632
2196                            53.575                         0.643
2197                            54.050                         0.668
2198                            54.525                         0.652

      Generosity  Perceptions of corruption  Positive affect  Negative affect
0          0.168                      0.882            0.414            0.258
1          0.191                      0.850            0.481            0.237
2          0.121                      0.707            0.517            0.275
3          0.164                      0.731            0.480            0.267
4          0.238                      0.776            0.614            0.268
…            …                          …              …              …
2194      -0.051                      0.844            0.658            0.212
2195      -0.047                      0.831            0.658            0.235
2196       0.006                      0.789            0.661            0.346
2197      -0.076                      0.757            0.610            0.242
2198      -0.070                      0.753            0.641            0.191

[2199 rows x 11 columns]
```

Now display the last few rows of the dataframe. Pay attention to the additional parameter that specifies the number of rows.

[5]:
```python
# This line will display only the last two rows of the dataframe.
df.tail(2)
```

[5]:
```
     Country name  year  Life Ladder  Log GDP per capita  Social support  \
2197     Zimbabwe  2021        3.155               7.657           0.685
2198     Zimbabwe  2022        3.296               7.670           0.666

      Healthy life expectancy at birth  Freedom to make life choices  \
2197                            54.050                         0.668
2198                            54.525                         0.652

      Generosity  Perceptions of corruption  Positive affect  Negative affect
2197      -0.076                      0.757            0.610            0.242
2198      -0.070                      0.753            0.641            0.191
```

4.2 3.2 Index and Column Names

In the `DataFrame`, the data is stored in a two dimensional grid (rows and columns). The rows are indexed and the columns are named. To see the index or the column names, you can use `DataFrame.index` or `DataFrame.columns` respectively.

[6]:
```
df.index
```

[6]:
```
RangeIndex(start=0, stop=2199, step=1)
```

As you can see, the index is a range of numbers between 0 (inclusive) and 2199 (not inclusive).

Run the cell below to see the column names.

[7]:
```
df.columns
```

[7]:
```
Index(['Country name', 'year', 'Life Ladder', 'Log GDP per capita',
       'Social support', 'Healthy life expectancy at birth',
       'Freedom to make life choices', 'Generosity',
       'Perceptions of corruption', 'Positive affect', 'Negative affect'],
      dtype='object')
```

The column names are saved as strings. As you can see, they can include spaces. This can lead to difficulties when accessing the columns (you will see this very soon), so it is a good idea to rename them to get rid of the spaces. A common practice is to replace them with underscores. To rename the columns, you can use `DataFrame.rename()` and pass the columns you want to rename in a dictionary.

In the next example, you will see how you can automatically replace all spaces with underscores

[8]:
```
# A dictionary mapping old column names to new column names. In addition to␣
↪replacing spaces
# with underscores, you will make all of the text lowercase.
columns_to_rename = {i: "_".join(i.split(" ")).lower() for i in df.columns}
# Note that this dictionary is created automatically from the column names.
# You can also create it by hand and rename only the columns you want to rename
# For example, see the commented line below:
# columns_to_rename = {"Country name": "country_name", "Life Ladder":␣
↪"life_ladder"}

# Rename the columns
df = df.rename(columns=columns_to_rename)
# Display the new dataframe
df.head()
```

[8]:
```
  country_name  year  life_ladder  log_gdp_per_capita  social_support  \
0  Afghanistan  2008        3.724               7.350           0.451
1  Afghanistan  2009        4.402               7.509           0.552
2  Afghanistan  2010        4.758               7.614           0.539
3  Afghanistan  2011        3.832               7.581           0.521
```

```
4  Afghanistan  2012        3.783             7.661          0.521

   healthy_life_expectancy_at_birth  freedom_to_make_life_choices  generosity  \
0                              50.5                         0.718       0.168
1                              50.8                         0.679       0.191
2                              51.1                         0.600       0.121
3                              51.4                         0.496       0.164
4                              51.7                         0.531       0.238

   perceptions_of_corruption  positive_affect  negative_affect
0                      0.882            0.414            0.258
1                      0.850            0.481            0.237
2                      0.707            0.517            0.275
3                      0.731            0.480            0.267
4                      0.776            0.614            0.268
```

4.3 3.3 Data Types

One cool thing about the DataFrame type is that the columns of the resulting DataFrame can have different `dtypes`. This is something you simply can not do with a Numpy array. You can look at them and if needed to you can change them.

[9]:
```
df.dtypes
```

[9]:
```
country_name                         object
year                                  int64
life_ladder                         float64
log_gdp_per_capita                  float64
social_support                      float64
healthy_life_expectancy_at_birth    float64
freedom_to_make_life_choices        float64
generosity                          float64
perceptions_of_corruption           float64
positive_affect                     float64
negative_affect                     float64
dtype: object
```

You can see that the columns above are of different types and if you compare it to how the data actually looks like, it seems that the types are correct. Sometimes if your data is incorrectly formatted, the imported types will be wrong. In this case you will want to change the types of the columns manually before proceeding. Check the code below on how you can do that. Note that nothing will change after running the code below, as the data is already of correct types.

[10]:
```
# List all of the columns that should be floats
float_columns = [i for i in df.columns if i not in  ["country_name", "year"]]
# Change the type of all float columns to float
df = df.astype({i: float for i in float_columns})
# Show the types of all columns
```

```
df.dtypes
```

[10]:
```
country_name                        object
year                                 int64
life_ladder                        float64
log_gdp_per_capita                 float64
social_support                     float64
healthy_life_expectancy_at_birth   float64
freedom_to_make_life_choices       float64
generosity                         float64
perceptions_of_corruption          float64
positive_affect                    float64
negative_affect                    float64
dtype: object
```

The `df.info()` provides some additional information. In addition to data types it also tells you the number of non-null values per column.

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2199 entries, 0 to 2198
Data columns (total 11 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   country_name                      2199 non-null   object
 1   year                              2199 non-null   int64
 2   life_ladder                       2199 non-null   float64
 3   log_gdp_per_capita                2179 non-null   float64
 4   social_support                    2186 non-null   float64
 5   healthy_life_expectancy_at_birth  2145 non-null   float64
 6   freedom_to_make_life_choices      2166 non-null   float64
 7   generosity                        2126 non-null   float64
 8   perceptions_of_corruption         2083 non-null   float64
 9   positive_affect                   2175 non-null   float64
 10  negative_affect                   2183 non-null   float64
dtypes: float64(9), int64(1), object(1)
memory usage: 189.1+ KB
```

4.4 3.4 Selecting Columns

One way of selecting a single column is to use `DataFrame.column_name`. Here you can see why it was a good idea that you renamed the columns to not include any whitespaces. This returns a Pandas `Series`, which is a different datatype from a `DataFrame`. You will see how to return a `DataFrame` a bit later.

[12]:
```
# Select the life_ladder column and store it in x
x = df.life_ladder
```

```
print(f"type(x):\n {type(x)}\n")
print(f"x:\n{x}")
```

```
type(x):
 <class 'pandas.core.series.Series'>

x:
0       3.724
1       4.402
2       4.758
3       3.832
4       3.783
        …
2194    3.616
2195    2.694
2196    3.160
2197    3.155
2198    3.296
Name: life_ladder, Length: 2199, dtype: float64
```

Another way to do this is to use square brackets and the name of the column in quortes, much as you would do when accessing an entry in a dictionary. As with dictionaries, you can use double quotes or simple quotes.

[13]:
```
x = df["life_ladder"]

print(f"type(x):\n {type(x)}\n")
print(f"x:\n{x}")
```

```
type(x):
 <class 'pandas.core.series.Series'>

x:
0       3.724
1       4.402
2       4.758
3       3.832
4       3.783
        …
2194    3.616
2195    2.694
2196    3.160
2197    3.155
2198    3.296
Name: life_ladder, Length: 2199, dtype: float64
```

Passing a list of labels rather than a single label selects the columns and returns a DataFrame (rather than a Series), with only the selected columns. You can use it to select one or more columns.

[14]:
```
x = df[["life_ladder"]]
# x = df[["life_ladder", "year"]]

print(f"type(x):\n {type(x)}\n")
print(f"x:\n{x}")
```

```
type(x):
 <class 'pandas.core.frame.DataFrame'>

x:
      life_ladder
0           3.724
1           4.402
2           4.758
3           3.832
4           3.783
…           …
2194        3.616
2195        2.694
2196        3.160
2197        3.155
2198        3.296

[2199 rows x 1 columns]
```

4.5 3.5 Selecting Rows

Passing a slice `:` selects matching rows and returns a DataFrame with all columns in your original dataframe.

[15]:
```
df[2:5]
```

[15]:
```
  country_name  year  life_ladder  log_gdp_per_capita  social_support  \
2  Afghanistan  2010        4.758               7.614           0.539
3  Afghanistan  2011        3.832               7.581           0.521
4  Afghanistan  2012        3.783               7.661           0.521

   healthy_life_expectancy_at_birth  freedom_to_make_life_choices  generosity  \
2                              51.1                         0.600       0.121
3                              51.4                         0.496       0.164
4                              51.7                         0.531       0.238

   perceptions_of_corruption  positive_affect  negative_affect
2                      0.707            0.517            0.275
3                      0.731            0.480            0.267
4                      0.776            0.614            0.268
```

4.6 3.6 Iterating Over Rows

If you want to iterate over the rows, you can use the `.iterrows()` method. For each row it yields a (index, row) tuple, where the row is a `Series` object containing the data. Note that this does not preserve the data types (dtypes) across the rows (dtypes are preserved across columns for DataFrames).

[16]:
```
index, row = next(df.iterrows())
row
```

[16]:
```
country_name                          Afghanistan
year                                         2008
life_ladder                                 3.724
log_gdp_per_capita                           7.35
social_support                              0.451
healthy_life_expectancy_at_birth             50.5
freedom_to_make_life_choices                0.718
generosity                                  0.168
perceptions_of_corruption                   0.882
positive_affect                             0.414
negative_affect                             0.258
Name: 0, dtype: object
```

4.7 3.7 Boolean Indexing

Now to the more fun part. If you looked carefully at the dataset that was displayed above, you probably saw that the datapoints are available for different years. What if you are interested only in data from a certain year? Or from a certain country? Or perhaps where a value in a certain column is greater than some predetermined value? You can use boolean indexing.

Run the cell below to select rows where the year equals to 2022. Try to uncomment some other row to see what it does.

[17]:
```
df[df["year"] == 2022]
# df[df["life_ladder"] > 5] # Select rows where life_ladder > 5
# df[df["life_ladder"] > 11] # This one should return an empty dataframe
```

[17]:
```
     country_name  year  life_ladder  log_gdp_per_capita  social_support  \
13    Afghanistan  2022        1.281                 NaN           0.228
28        Albania  2022        5.212               9.626           0.724
59      Argentina  2022        6.261              10.011           0.893
75        Armenia  2022        5.382               9.668           0.811
91      Australia  2022        7.035              10.854           0.942
…             …   …            …                  …             …
2104      Uruguay  2022        6.671              10.084           0.905
2120   Uzbekistan  2022        6.016               8.990           0.879
2137    Venezuela  2022        5.949                 NaN           0.899
2154      Vietnam  2022        6.267               9.333           0.879
2198     Zimbabwe  2022        3.296               7.670           0.666
```

```
      healthy_life_expectancy_at_birth  freedom_to_make_life_choices  \
13                              54.875                         0.368
28                              69.175                         0.802
59                              67.250                         0.825
75                              67.925                         0.790
91                              71.125                         0.854
…                                  …                            …
2104                            67.500                         0.878
2120                            65.600                         0.959
2137                            63.875                         0.770
2154                            65.600                         0.975
2198                            54.525                         0.652

      generosity  perceptions_of_corruption  positive_affect  negative_affect
13           NaN                      0.733            0.206            0.576
28        -0.066                      0.846            0.547            0.255
59        -0.128                      0.810            0.724            0.284
75        -0.154                      0.705            0.531            0.549
91         0.153                      0.545            0.711            0.244
…            …                          …              …               …
2104      -0.052                      0.631            0.775            0.267
2120       0.309                      0.616            0.741            0.225
2137         NaN                      0.798            0.754            0.292
2154      -0.179                      0.703            0.774            0.108
2198      -0.070                      0.753            0.641            0.191

[114 rows x 11 columns]
```

Note that now that you selected only the certain rows, the index column does not make much sense anymore because you have a lot of gaps. While this is not a problem, in some cases you might want the index to correspond to the actual row number. To achieve this you can use `reset_inex()`. In other cases you might want to keep the index as it is to more easily refer back to the original dataframe. It all depends on the context of your project. Run the cell below to reset the index and take a look at the output.

[18]:
```
new_df = df[df["year"] == 2022]
new_df = new_df.reset_index(drop=True)
new_df
```

[18]:
```
    country_name  year  life_ladder  log_gdp_per_capita  social_support  \
0    Afghanistan  2022        1.281                 NaN           0.228
1        Albania  2022        5.212               9.626           0.724
2      Argentina  2022        6.261              10.011           0.893
3        Armenia  2022        5.382               9.668           0.811
4      Australia  2022        7.035              10.854           0.942
..           …     …          …                   …               …
```

```
109        Uruguay  2022         6.671              10.084            0.905
110     Uzbekistan  2022         6.016               8.990            0.879
111      Venezuela  2022         5.949                 NaN            0.899
112        Vietnam  2022         6.267               9.333            0.879
113       Zimbabwe  2022         3.296               7.670            0.666

     healthy_life_expectancy_at_birth  freedom_to_make_life_choices  \
0                              54.875                         0.368
1                              69.175                         0.802
2                              67.250                         0.825
3                              67.925                         0.790
4                              71.125                         0.854
..                                …                             …
109                            67.500                         0.878
110                            65.600                         0.959
111                            63.875                         0.770
112                            65.600                         0.975
113                            54.525                         0.652

     generosity  perceptions_of_corruption  positive_affect  negative_affect
0           NaN                      0.733            0.206            0.576
1        -0.066                      0.846            0.547            0.255
2        -0.128                      0.810            0.724            0.284
3        -0.154                      0.705            0.531            0.549
4         0.153                      0.545            0.711            0.244
..          …                          …                …                …
109      -0.052                      0.631            0.775            0.267
110       0.309                      0.616            0.741            0.225
111         NaN                      0.798            0.754            0.292
112      -0.179                      0.703            0.774            0.108
113      -0.070                      0.753            0.641            0.191

[114 rows x 11 columns]
```

5 4. Summary Statistics

Later in this course you will learn about summary statistics. For now, this is just to show you that Pandas allows for a very simple way to calculate all sorts of statistics using `describe()`. Run the cell below to see a quick statistical summary of your data. It doesn't matter if you don't know what each row means, you will learn all about it in the coming weeks.

[19]:
```
df.describe()
```

[19]:
```
              year  life_ladder  log_gdp_per_capita  social_support  \
count  2199.000000  2199.000000         2179.000000     2186.000000
mean   2014.161437     5.479227            9.389760        0.810681
std       4.718736     1.125527            1.153402        0.120953
```

```
min    2005.000000    1.281000             5.527000        0.228000
25%    2010.000000    4.647000             8.500000        0.747000
50%    2014.000000    5.432000             9.499000        0.836000
75%    2018.000000    6.309500            10.373500        0.905000
max    2022.000000    8.019000            11.664000        0.987000

       healthy_life_expectancy_at_birth  freedom_to_make_life_choices  \
count                       2145.000000                   2166.000000
mean                          63.294582                      0.747847
std                            6.901104                      0.140137
min                            6.720000                      0.258000
25%                           59.120000                      0.656250
50%                           65.050000                      0.770000
75%                           68.500000                      0.859000
max                           74.475000                      0.985000

        generosity  perceptions_of_corruption  positive_affect  \
count  2126.000000                2083.000000      2175.000000
mean      0.000091                   0.745208         0.652148
std       0.161079                   0.185835         0.105913
min      -0.338000                   0.035000         0.179000
25%      -0.112000                   0.688000         0.572000
50%      -0.023000                   0.800000         0.663000
75%       0.092000                   0.869000         0.738000
max       0.703000                   0.983000         0.884000

       negative_affect
count      2183.000000
mean          0.271493
std           0.086872
min           0.083000
25%           0.208000
50%           0.261000
75%           0.323000
max           0.705000
```

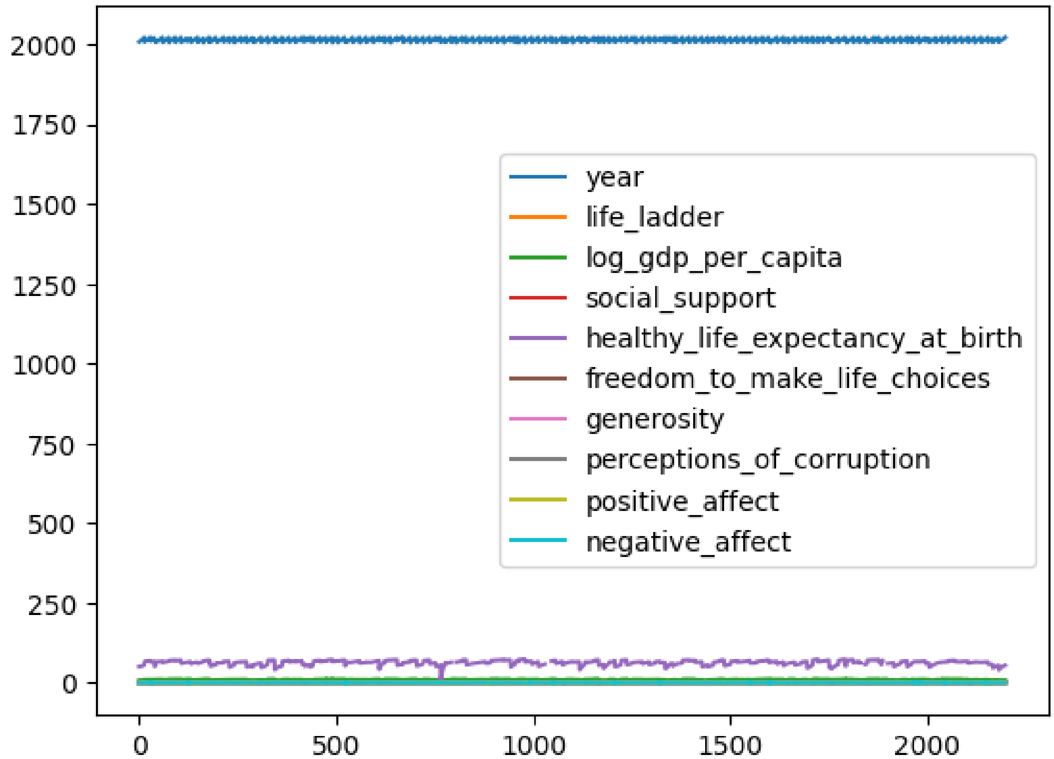
Not all of the summary statistics always make sense. In your case, for example, you are looking at the summary statistics across various columns. But are you sure you know what the final numbers actually mean? You have data for many different countries, but are you sure that you have the same amount of datapoints for each country or for each year? Also the countries can have vastly different populations, is it fair to just average the numbers out?

6 5. Plotting

If you want to plot the data, you can use `DataFrame.plot()`. By default it uses the index as the x axis and plots all the numeric columns as y axes. Run the cell below to see the output for your dataframe.

[20]:
```
# If the plot doesn't render, first try re-running this cell. If that doesn't work,
# you can restart the kernel (from the Kernel menu above) and try running the notebook again
df.plot()
```

[20]: <Axes: >

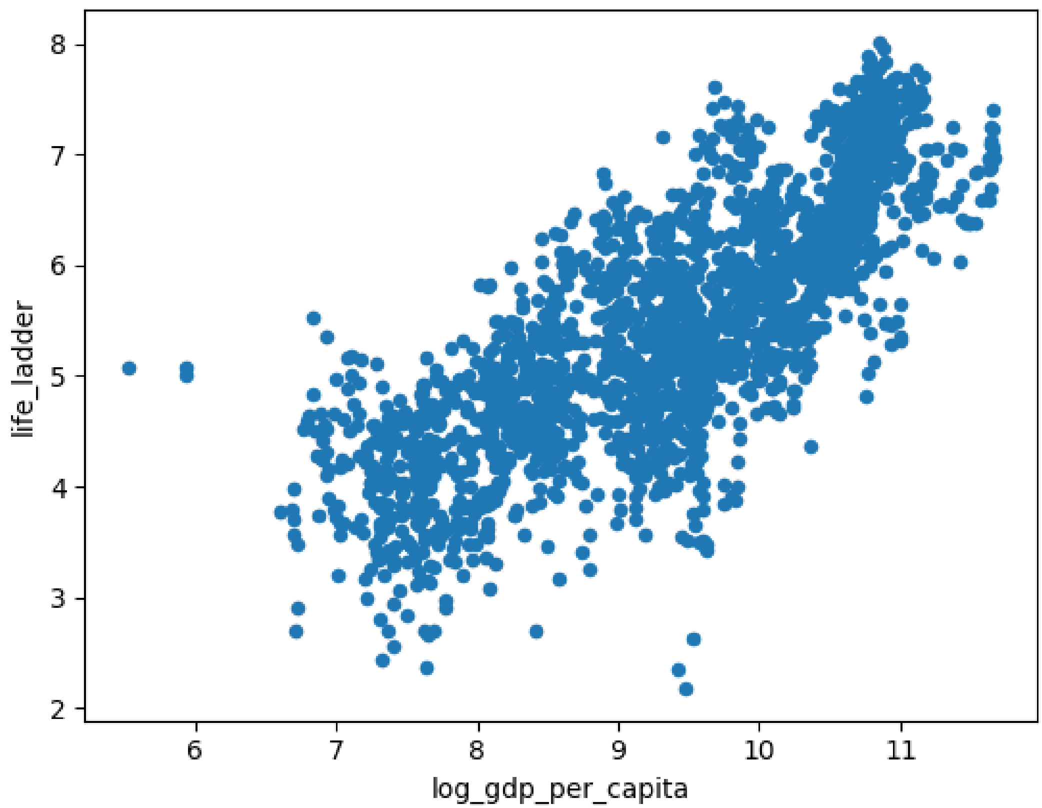


As you can see, in this case the plot is not very useful. The index does not have any specific meaning, and the values of various columns differ greatly (years are all around 2000, but the values in the other columns are much lower) and thus you cannot see much in the plot. Try setting some parameters of the `.plot()` method to see what it allows you to do. You can find the documentation here.

Run the cell below to see a scatter plot with specifically chosen x and y variables. On the x axis there is logarithm of the GDP (measuring the wealth) while on the y axis there is the life ladder. This column contains values which are an estimate of self-assessed life quality on a scale of 1 to 10 as given by a survey among the people.

[21]:
```
df.plot(kind='scatter', x='log_gdp_per_capita', y='life_ladder')
```

[21]: <Axes: xlabel='log_gdp_per_capita', ylabel='life_ladder'>



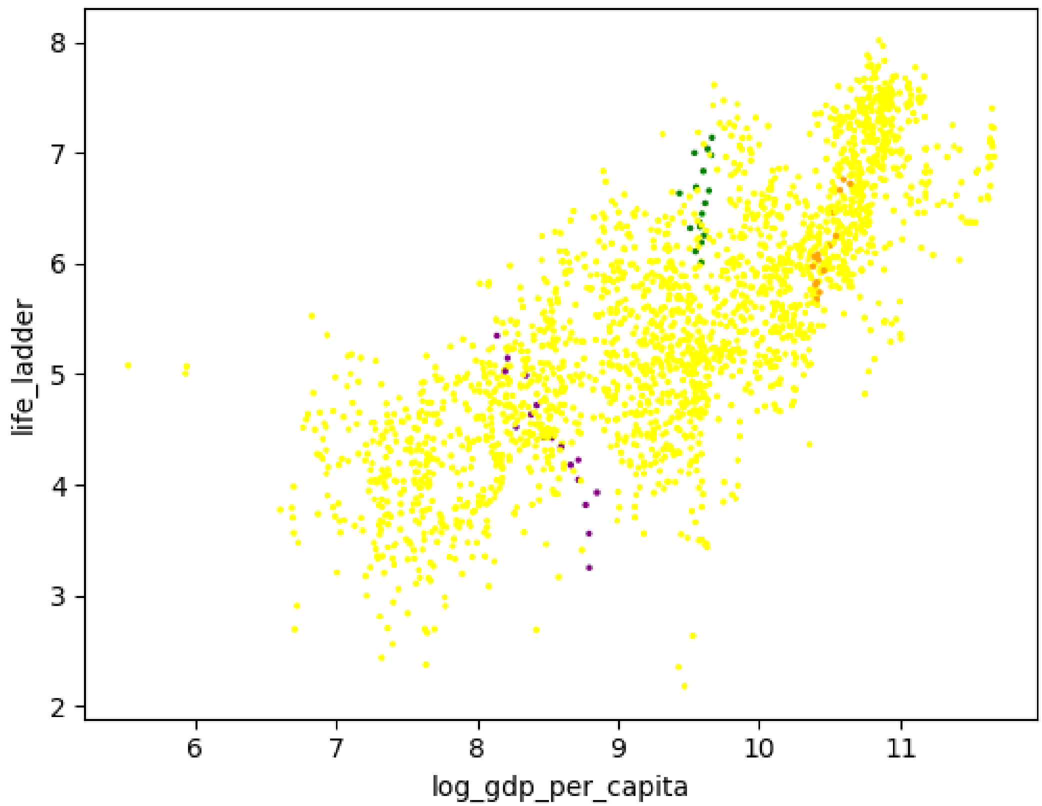
You can see that there is some sort of trend between the wealth of the country and the happiness of the population and you can say that it looks like that wealthier people are to some extent happier. In week three, you will explore this kind of relationship further.

Sometimes it is very insightful to separate the points by colors to highlight different characteristics or some points you are most interested in. Take a look at the example below

[22]:
```
# Create a dictionary to map the country names to colors
cmap = {
    'Brazil': 'Green',
    'Slovenia': 'Orange',
    'India': 'purple'
}

df.plot(
    kind='scatter',
    x='log_gdp_per_capita',
    y='life_ladder',
    c=[cmap.get(c, 'yellow') for c in df.country_name], # Set the colors
    s=2 # Set the size of the points
    )
```

[22]: <Axes: xlabel='log_gdp_per_capita', ylabel='life_ladder'>

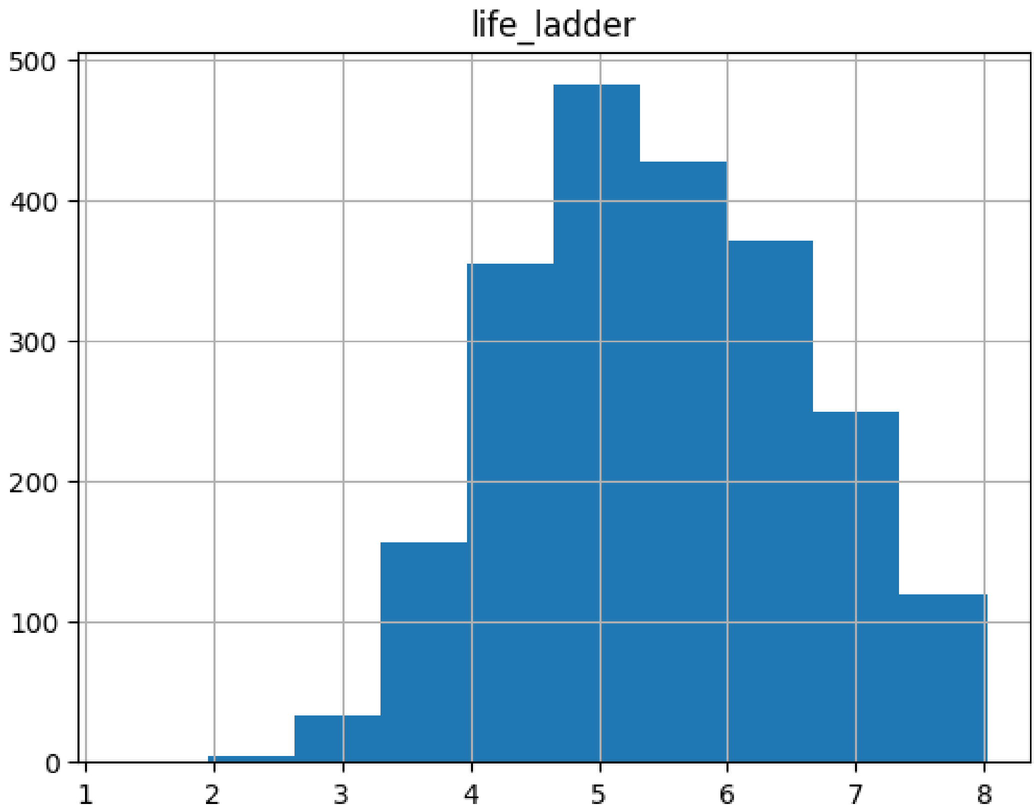


You can see that even though in general higher GDP means higher value on the life ladder, this is not an universal truth. Comparing Slovenia (orange) with Brazil (green), you can see that people in Brazil earn less, but are on average happier than Slovenians through the years.

Another very useful task you can do with plots is to visulize the distribution of your data. You will learn how to do this in more detail later, but for example you can easily plot a histogram using Pandas. Ise `DataFrame.hist()` on the dataframe you want to plot. Note that if you have many columns in the dataframe, this command will plot a histogram for each of the columns. You can select a single column from the dataframe if you only want to plot that one.

[23]:
```
df.hist("life_ladder")
```

[23]: array([[<Axes: title={'center': 'life_ladder'}>]], dtype=object)

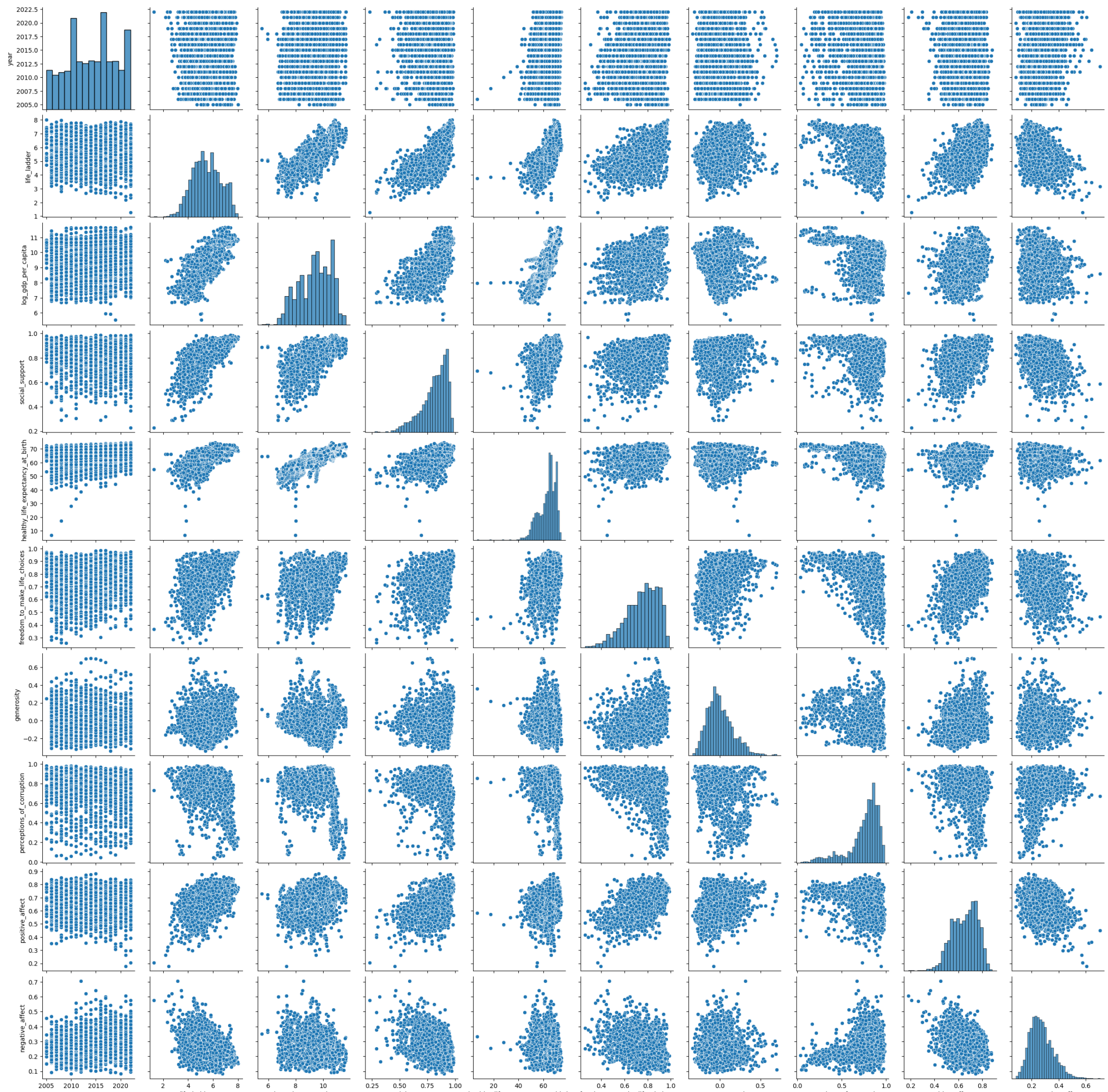


What you see in this histogram is a distribution of values in the "life_ladder" column. What do you think about this distribution on the first glance? Are the people generally happy about their quality of life? Note that to answer this question properly, you need to dig a bit deeper into the data: understand where each value comes from, as the values are not single datapoints (single answers by people), but already aggregated values across countries and at various points in time.

You can use other external libraries to easily produce various advanced plots. One of such libraries is Seaborn. You have already imported it at the beginning of this lab using `import seaborn as sns`. Run the cell below to see one of the many simple and efficient plotting possibilities (you will use this one later on in the other notebooks as well). Since the dataset has many columns it might take a few seconds to run.

[25]:
```
# If the plot doesn't render, first try re-running this cell. If that doesn't␣
↪work,
# you can restart the kernel (from the Kernel menu above) and try running the␣
↪notebook again
sns.pairplot(df)
```

[25]:
```
<seaborn.axisgrid.PairGrid at 0x7600a3746290>
```



With this kind of plot, you can see pairwise scatter plots for each pair of columns. On the diagonal (where both columns are the same), you don't have a scatter plot (which would only show a line), but a histogram showing the distribution of datapoints.

You can see that both the scatter plots and histograms have very different shapes across columns. Think about various insights you could get from this kind of visualization.

7 6. Operations on Columns

Sometimes the values in the columns are not giving you the information that you need, but there is a way to calculate that information from the values you have.

For example you can create a new column, which is a sum of two columns.

[26]:
```python
# Create a new column which is the sum of the year and the value on the life␣
↪ladder.
df["this_column_makes_no_sense"] = df["year"] + df["life_ladder"]
# Create a new column which is the difference of two columns.
df["net_affect_difference"] = df["positive_affect"] - df["negative_affect"]

df.head()
```

[26]:
```
  country_name  year  life_ladder  log_gdp_per_capita  social_support  \
0  Afghanistan  2008        3.724               7.350           0.451
1  Afghanistan  2009        4.402               7.509           0.552
2  Afghanistan  2010        4.758               7.614           0.539
3  Afghanistan  2011        3.832               7.581           0.521
4  Afghanistan  2012        3.783               7.661           0.521

   healthy_life_expectancy_at_birth  freedom_to_make_life_choices  generosity  \
0                              50.5                         0.718       0.168
1                              50.8                         0.679       0.191
2                              51.1                         0.600       0.121
3                              51.4                         0.496       0.164
4                              51.7                         0.531       0.238

   perceptions_of_corruption  positive_affect  negative_affect  \
0                      0.882            0.414            0.258
1                      0.850            0.481            0.237
2                      0.707            0.517            0.275
3                      0.731            0.480            0.267
4                      0.776            0.614            0.268

   this_column_makes_no_sense  net_affect_difference
0                    2011.724                  0.156
1                    2013.402                  0.244
2                    2014.758                  0.242
3                    2014.832                  0.213
4                    2015.783                  0.346
```

Above you can see your dataframe with both new columns. The first one doesn't make much sense, it's just adding the year to the life ladder. The second one, however, find the net difference between positive and negative affect. Perhaps there's an interesting set of patterns between this new column and other columns that you'd now be able to explore. What other columns might you want to calculate? In general, the ability to create new columns using operations on existing columns can be a powerful tool.

If you want to perform some more advanced operations on columns, you can use `DataFrame.apply()`, with which you can apply practically any function to a column. Below you can see how to use the `DataFrame.apply()` in various ways. Try to edit `my_function` to perform an operation of your choice.

[27]:
```
# Using df.apply() with a lambda function
# Rescale the life_ladder column to values between 0 and 1 and save it to a new␣
↪column
df['life_ladder_rescaled'] = df['life_ladder'].apply(lambda x: x / 10)

# Using df.apply() with your own function
# First define a function. The function can do whatever you want. This example␣
↪will double the column's values
def my_function(x):
    # do stuff to x
    y = x * 2
    return y
# Apply the function.
df['my_function'] = df['life_ladder'].apply(my_function)

# Show the new dataframe
df.head()
```

[27]:
```
  country_name  year  life_ladder  log_gdp_per_capita  social_support  \
0  Afghanistan  2008        3.724               7.350           0.451
1  Afghanistan  2009        4.402               7.509           0.552
2  Afghanistan  2010        4.758               7.614           0.539
3  Afghanistan  2011        3.832               7.581           0.521
4  Afghanistan  2012        3.783               7.661           0.521

   healthy_life_expectancy_at_birth  freedom_to_make_life_choices  generosity  \
0                              50.5                         0.718       0.168
1                              50.8                         0.679       0.191
2                              51.1                         0.600       0.121
3                              51.4                         0.496       0.164
4                              51.7                         0.531       0.238

   perceptions_of_corruption  positive_affect  negative_affect  \
0                      0.882            0.414            0.258
1                      0.850            0.481            0.237
2                      0.707            0.517            0.275
3                      0.731            0.480            0.267
4                      0.776            0.614            0.268

   this_column_makes_no_sense  net_affect_difference  life_ladder_rescaled  \
0                    2011.724                  0.156                0.3724
1                    2013.402                  0.244                0.4402
2                    2014.758                  0.242                0.4758
3                    2014.832                  0.213                0.3832
4                    2015.783                  0.346                0.3783

   my_function
```

```
0        7.448
1        8.804
2        9.516
3        7.664
4        7.566
```

Congratulations on finishing this lab. If you understand the code above, you are well suited to start working on this week's programming assignment and other labs and assignments throughout the course which use Pandas. If you need a refresher on Pandas in other Exploratory Data Analysis labs, come back to this one and review the skills taught here.

[ ]:

```

```