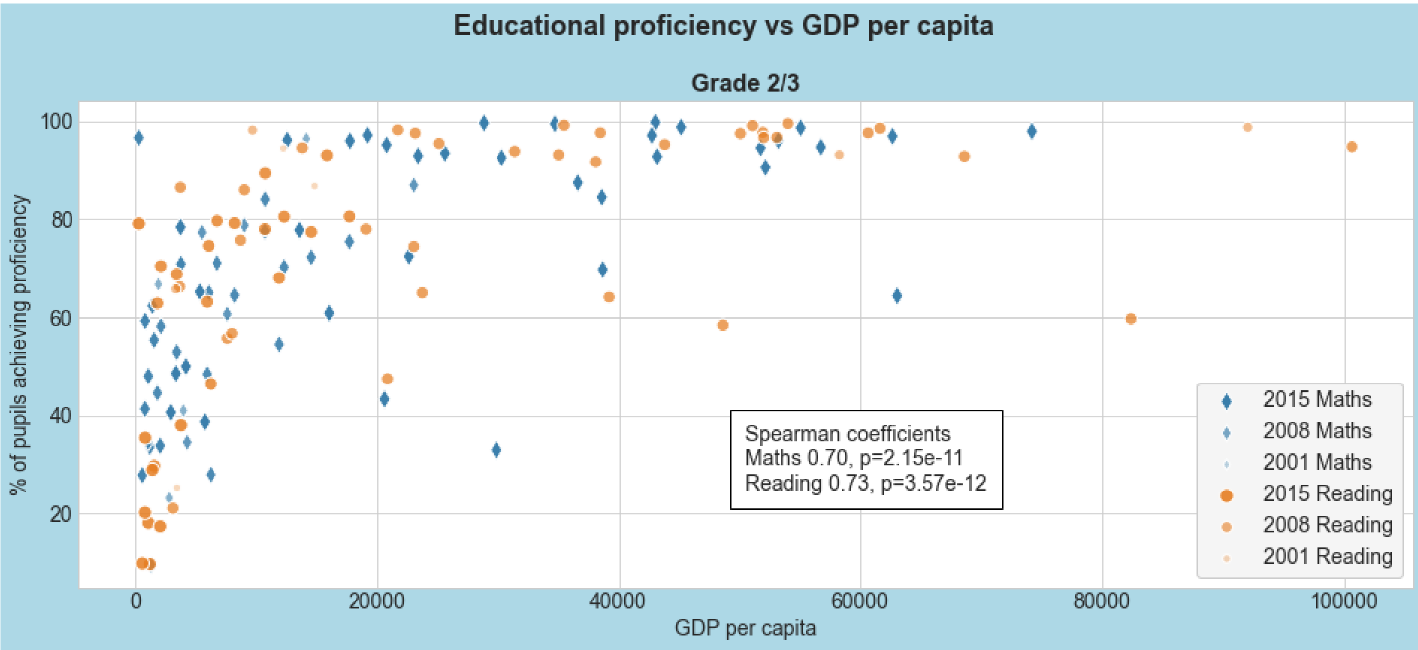
Introduction

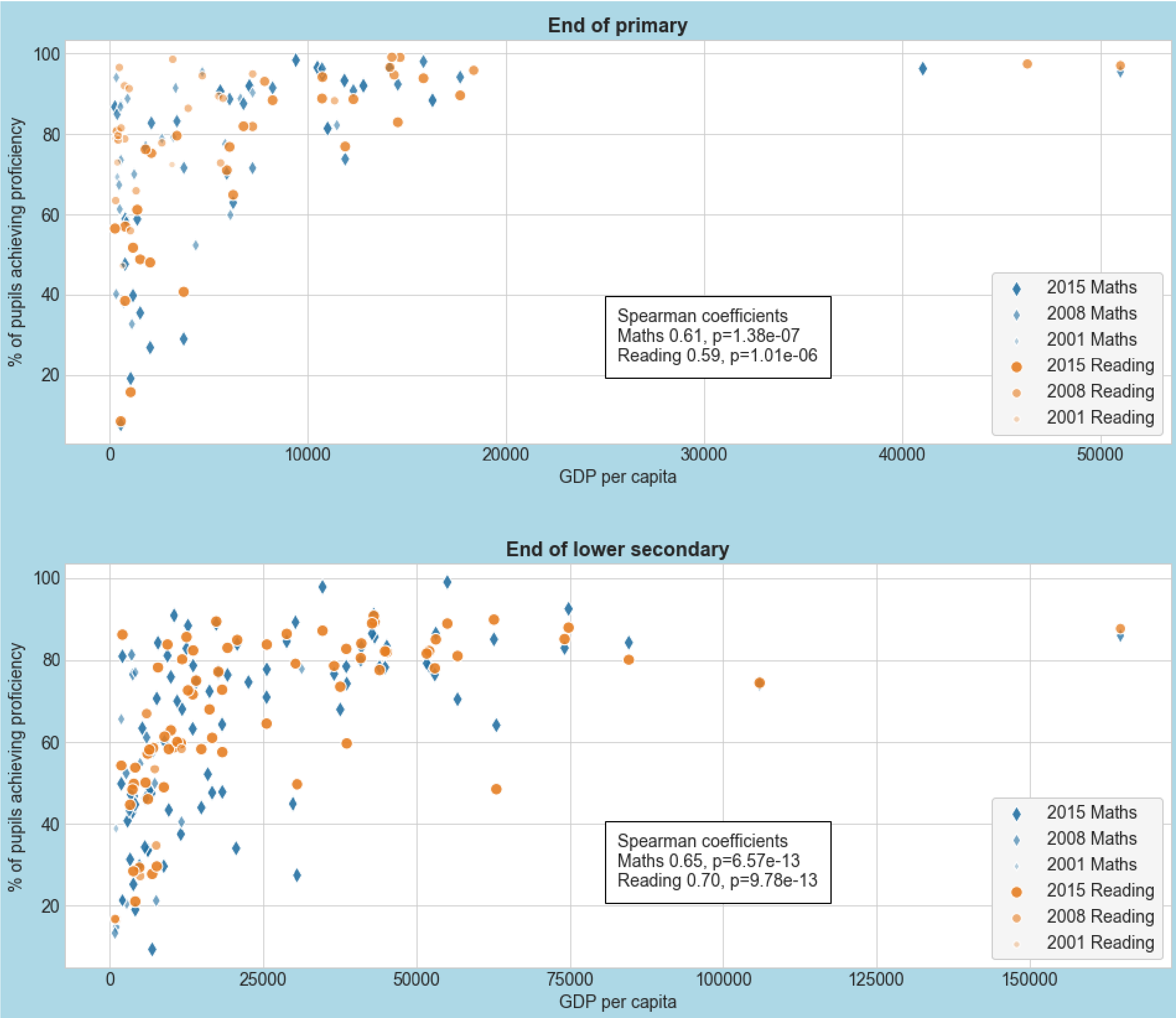
This is a little exercise I set myself in using Pandas and MatPlotLib to investigate a dataset. I decided to use one of the publicly available datasets from Google - I chose the UN Sustainable Development Goals dataset with the idea of trying to extract some form of insight from somewhere.

It turned out to be excellent training for digging around with Pandas primarily, and reinforced the ubiquitous message that most of the work in data science is in exploring, cleaning and structuring data before the analysis even starts.

Results

The purpose of this exercise was primarily to practise techniques, but nevertheless it had a direction. The rough question I ended up examining was "Is there a correlation between a country's economic strength and the success of its education?". To give some kind of (very rough) answer I compared reading proficiency and mathematical proficiency (according to UN definitions) against GDP per capita. Here are the results...

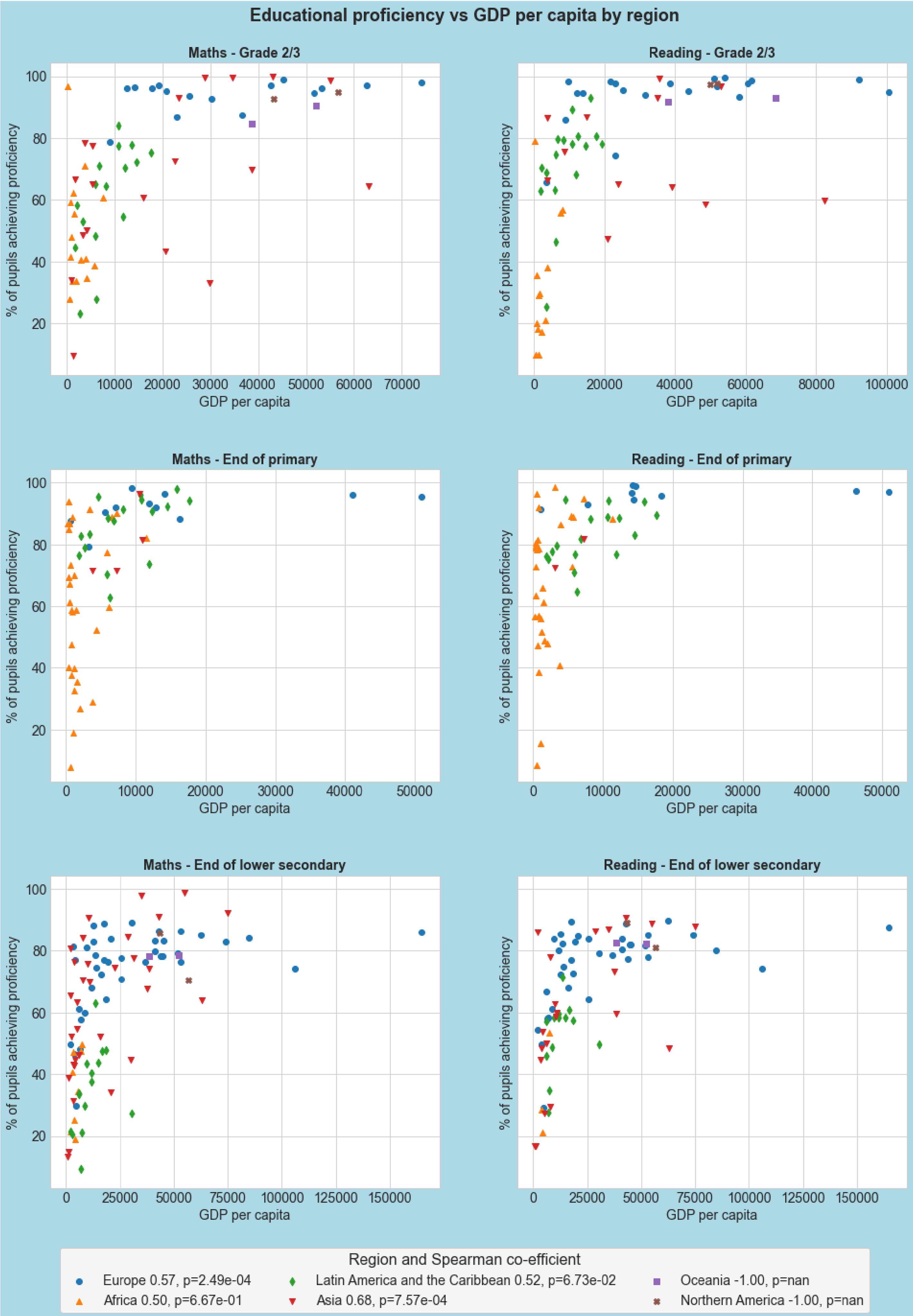




So the answer to the question "Is there a correlation between a country's economic strength and the success of its education?" seems to be "Yes". Looking at the Spearman rank coefficients, we tend to have a "strong" correlation according to this guide (i.e., in the range 0.60 - 0.79).

However, although wealthier countries do tend to have better results, there are plenty of exceptions to the rule, and perhaps more interestingly, there are many poorer countries that achieve good results.

The results by region are shown below. Not much more to note except perhaps Asia's strong maths performance in the early years of education despite relatively low GDP per capita. (The Spearman rank coefficients are unreliable because of insufficient data.)

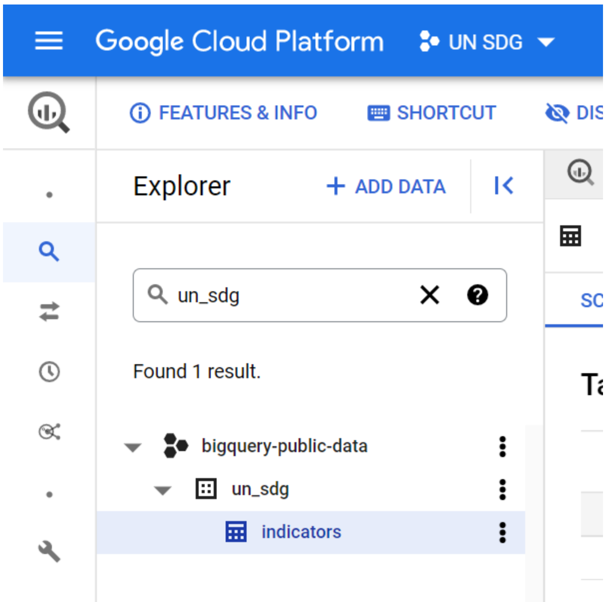

Educational proficiency vs GDP per capita by region
Maths - Grade 2/3
Reading - Grade 2/3
Maths - End of primary
Reading - End of primary
Maths - End of lower secondary
Reading - End of lower secondary
% of pupils achieving proficiency
GDP per capita
Region and Spearman co-efficient
Europe 0.57, p=2.49e-04
Africa 0.50, p=6.67e-01
Latin America and the Caribbean 0.52, p=6.73e-02
Asia 0.68, p=7.57e-04
Oceania -1.00, p=nan
Northern America -1.00, p=nan

And now for the more interesting part (at least for me) - the rest of this notebook shows how to produce these charts...

The Dataset on BigQuery

Access the dataset

To access public datasets on BigQuery we first need an account on Google Cloud Console and then a new project (details here). Thereafter we can search for the dataset in the Explorer.



Investigate the dataset

I decided I wanted to work on a small-ish dataset because when using BigQuery:

- Your queries can only use 1TB/mth without paying
- It's only possible to download query results of up to 10MB at a time direct to your computer. (You can store larger files in Google Cloud Storage but I wanted to work locally.)

Structure and size

To start with, the table schema gives info on column types and description (the image below shows an extract - click to see all columns)

Field name	Type	Mode	Policy tags	Description
goal	INTEGER	NULLABLE		High-level goal for sustainable development
target	STRING	NULLABLE		Each goal has multiple targets. Specific data points that, when achieved, indicate substantial progress toward a goal
indicator	STRING	NULLABLE		Quantifiable metric used to determine progress towards reaching a target. Each target has between 1 and 3 indicators
seriescode	STRING	NULLABLE		Abbreviated string of characters for each specific indicator
seriesdescription	STRING	NULLABLE		Full text description of indicator
geoareacode	INTEGER	NULLABLE		Numeric code of GeoArea
geoareaname	STRING	NULLABLE		Full text of GeoArea. Includes countries, regions, and continents
timeperiod	STRING	NULLABLE		Time period for which the value is relevant
value	STRING	NULLABLE		Numeric value of GeoArea
time_detail	STRING	NULLABLE		Time period in which the data was collected to calculate value
source	STRING	NULLABLE		Original source of data
footnote	STRING	NULLABLE		Specific details regarding data for individual observation
nature	STRING	NULLABLE		
age	STRING	NULLABLE		

We can also preview the table contents to get a first idea of what sort of data we have.

FEATURES & INFO SHORTCUT DISABLE EDITOR TABS

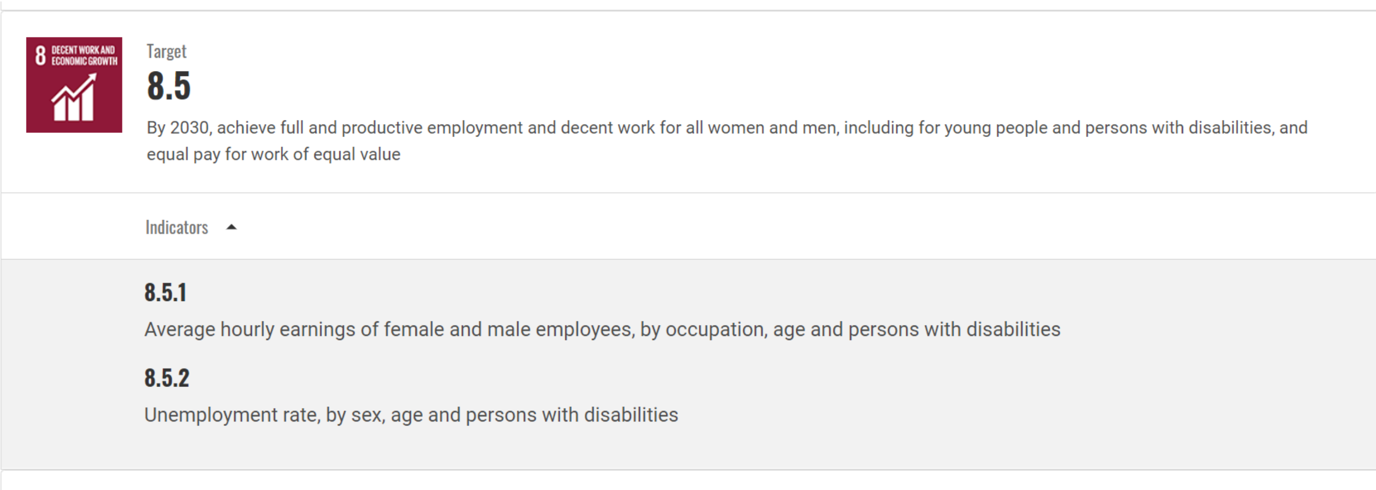
EDITOR UN SDG ... INDICAT... COMPOSE NEW QUERY

indicators QUERY SHARE COPY SNAPSHOT DELETE EXPORT

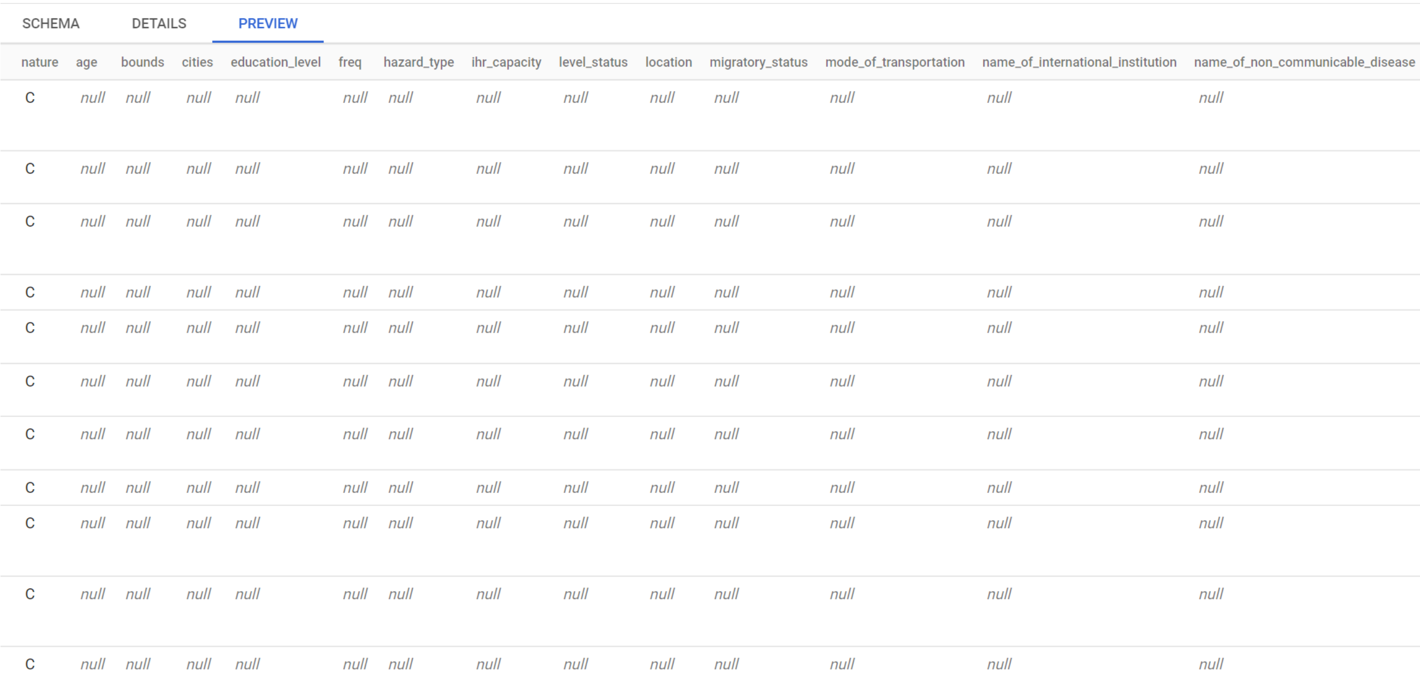
SCHEMA DETAILS PREVIEW

Row	goal	target	indicator	seriescode	seriesdescription	geoareacode	geoareaname	timeperiod	value	time_detail	source
1	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	56	Belgium	2006	14.5	*null*	ILOSTAT - I
2	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	76	Brazil	2011	11.1	*null*	ILOSTAT - I
3	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	100	Bulgaria	2014	3.6	*null*	ILOSTAT - I
4	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	170	Colombia	2015	4947.0	*null*	ILOSTAT - I
5	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	203	Czechia	2012	147.6	*null*	ILOSTAT - I
6	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	203	Czechia	2013	98.6	*null*	ILOSTAT - I
7	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	203	Czechia	2013	214.0	*null*	ILOSTAT - I
8	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	218	Ecuador	2014	2.7	*null*	ILOSTAT - I

We can immediately see that the goal, target, indicator and value columns are the essential columns which correspond with the structure set out on the UN SDG homepage. For example, taking the first row of the table, we can see how it corresponds with goal 8, target 8.5, and indicator 8.5.1 as shown below (copied from the Targets and Indicators sub-section on the Goal 8 page).

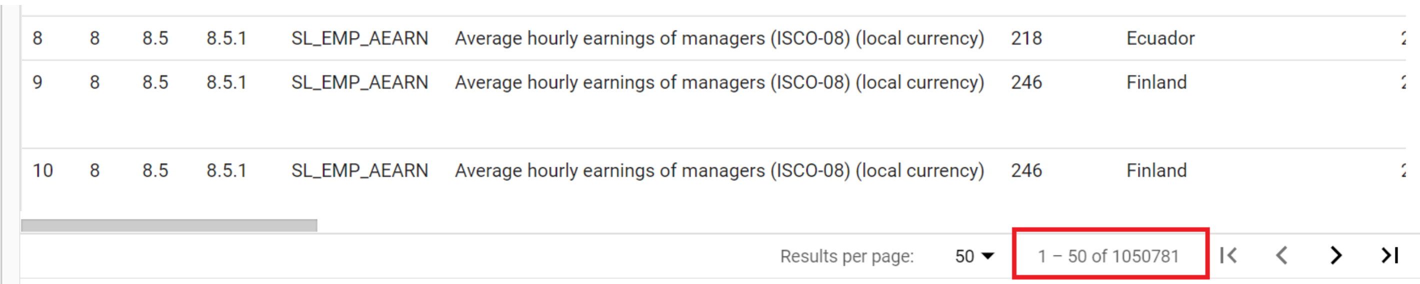


Scrolling horizontally we can see there are a few columns that might have a lot of `null` values. Something to investigate...



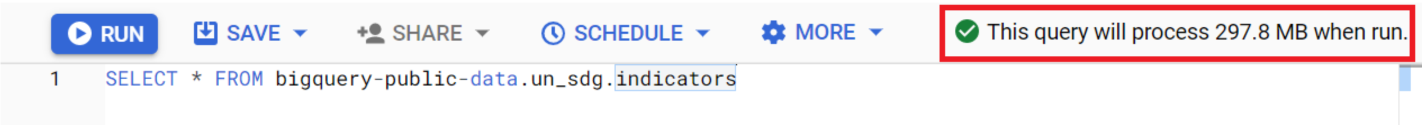
nature	age	bounds	cities	education_level	freq	hazard_type	ihr_capacity	level_status	location	migratory_status	mode_of_transportation	name_of_international_institution	name_of_non_communicable_disease
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null
C	null	null	null	null	null	null	null	null	null	null	null	null	null

And at the bottom of the preview window we can see that the table holds just over 1 million rows.



8	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	218	Ecuador
9	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	246	Finland
10	8	8.5	8.5.1	SL_EMP_AEARN	Average hourly earnings of managers (ISCO-08) (local currency)	246	Finland

1 million rows isn't that huge. Selecting the whole table seems possible using `SELECT * FROM bigquery-public-data.un_sdg.indicators` as BigQuery displays how much of our query quota will be used, and in this case it's relatively small. Unfortunatley though, when we run the query and then download the results the file is limited to about 12,000 rows of the 1 million so as not to exceed 10MB. So that's no good.



OK - in that case we'll maybe focus on just one goal and extract that data. Time to use some SQL to work out what we've got, and also to understand what is practically accessible without chewing through our query quota or exceeding the download file size maximum. The query below gives me the number of rows by goal:

```
SELECT goal, COUNT(*) AS num_rows
FROM bigquery-public-data.un_sdg.indicators
GROUP BY goal
ORDER BY goal
```

We can look at the downloaded csv file of results using pandas. One way to reduce the amount of data is to focus on a subset of the goals, so let's see what volume of data we have per goal.

In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
rows_per_goal = pd.read_csv('data/count-rows-per-goal.csv')
total_rows = rows_per_goal['num_rows'].sum()
print(f"Total rows : {total_rows:,}")
rows_per_goal.style.format("{:,}")
```

```
Total rows : 1,050,781
```

Out[2]:

	goal	num_rows
0	1	62,742
1	2	38,659
2	3	134,617
3	4	41,533
4	5	21,703
5	6	36,062
6	7	19,670
7	8	270,140
8	9	34,880
9	10	19,241
10	11	10,557
11	12	225,302
12	13	2,740
13	14	12,614
14	15	66,540
15	16	18,719
16	17	35,062

Let's take a look at the `null` values as well. If we're going to select only one or two goals then maybe some columns aren't relevant for certain goals. That way we can exclude them. To count all the `null` values in the columns I'll use a query of the following form:

```
SELECT goal, COUNT(goal) AS num_rows,
SUM(CASE WHEN col1_name IS NULL THEN 1 ELSE 0 END) as col1_name,
SUM(CASE WHEN col2_name IS NULL THEN 1 ELSE 0 END) as col2_name
FROM bigquery-public-data.un_sdg.indicators
GROUP BY goal
ORDER BY goal
```

That means we need a list of the column names so we can stitch them together into the query. Of course we could easily copy and paste from the BigQuery schema and then use Excel to concatenate, but that feels like cheating - the whole point is to practise with Python! Instead we can use SQL to extract column names and then Pandas / Python to create the query. Here's the query for column names:

```
SELECT COLUMN_NAME
FROM `bigquery-public-data`.un_sdg.INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'indicators'
ORDER BY ORDINAL_POSITION
```

...which gives us the following result (removing the goal column):

In [3]:
```
column_names = pd.read_csv('data/un-sdg-column-names.csv')
column_names_list = column_names['COLUMN_NAME'].tolist()
del column_names_list[0]
print(column_names_list)
```

```
['target', 'indicator', 'seriescode', 'seriesdescription', 'geoareacode', 'geoareaname', 'tim
eperiod', 'value', 'time_detail', 'source', 'footnote', 'nature', 'age', 'bounds', 'cities',
'education_level', 'freq', 'hazard_type', 'ihr_capacity', 'level_status', 'location', 'migrat
ory_status', 'mode_of_transportation', 'name_of_international_institution', 'name_of_non_comm
unicable_disease', 'sex', 'tariff_regime_status', 'type_of_mobile_technology', 'type_of_occup
ation', 'type_of_product', 'type_of_skill', 'type_of_speed', 'units']
```

Now we can create a string from the list for the query.

In [4]:
```
sql_col_expressions = list(map(lambda col_name : 'SUM(CASE WHEN ' + col_name + ' IS NULL THEN 1
sql_col_str = ', '.join(sql_col_expressions)

# Display a truncated extract of the string
print(sql_col_str[0:400] + "...")
```

```
SUM(CASE WHEN target IS NULL THEN 1 ELSE 0 END) AS target, SUM(CASE WHEN indicator IS NULL TH
EN 1 ELSE 0 END) AS indicator, SUM(CASE WHEN seriescode IS NULL THEN 1 ELSE 0 END) AS seriesc
ode, SUM(CASE WHEN seriesdescription IS NULL THEN 1 ELSE 0 END) AS seriesdescription, SUM(CAS
E WHEN geoareacode IS NULL THEN 1 ELSE 0 END) AS geoareacode, SUM(CASE WHEN geoareaname IS NU
LL THEN 1 ELSE 0 END) AS geo...
```

Here's the result of the query.

In [5]:
```
nulls_per_goal = pd.read_csv('data/un-sdg-non-null entries-by-goal.csv')
nulls_per_goal.style.set_table_styles([

    # Add styles so we can have narrow columns but show the column title in full on hover.
    {'selector': ' .col_heading', 'props': 'max-width: 30px; overflow: hidden'},
```

```
    {'selector': '.col_heading:hover', 'props': '; overflow: visible'},
    {'selector': '.col_heading:hover~.col_heading', 'props': '; visibility: hidden'},

], overwrite=False)
```

Out[5]:

	goal	num_rov	target	indicat	seriesc	seriesc	geoare	geoare	timepe	value	time_d	source	footnot	nature	age
0	1	62742	0	0	0	0	0	0	0	0	9353	0	10606	0	54372
1	2	38659	0	0	0	0	0	0	0	0	0	0	5212	0	34958
2	3	134617	0	0	0	0	0	0	0	0	2668	0	114972	0	85929
3	4	41533	0	0	0	0	0	0	0	0	36695	0	39627	0	41397
4	5	21703	0	0	0	0	0	0	0	0	2076	0	4740	0	15136
5	6	36062	0	0	0	0	0	0	0	0	0	0	31874	0	36062
6	7	19670	0	0	0	0	0	0	0	0	1080	0	19670	0	19670
7	8	270140	0	0	0	0	0	0	0	0	10413	0	238315	0	244885
8	9	34880	0	0	0	0	0	0	0	0	1	0	14860	0	34880
9	10	19241	0	0	0	0	0	0	0	0	750	0	14429	0	19241
10	11	10557	0	0	0	0	0	0	0	0	0	0	732	0	10557
11	12	225302	0	0	0	0	0	0	0	0	0	0	225302	0	225302
12	13	2740	0	0	0	0	0	0	0	0	0	0	0	0	2740
13	14	12614	0	0	0	0	0	0	0	0	0	21	11787	0	12614
14	15	66540	0	0	0	0	0	0	0	0	0	0	56337	0	66540
15	16	18719	0	0	0	0	0	0	0	0	0	0	14984	0	18073
16	17	35062	0	0	0	0	0	0	0	0	3060	0	17624	0	35062

We'll need to convert those numbers to percentages for them to be useful, but just before that let's add in the list of the UN SDG goals so we know what they're about. I copied the list from Wikipedia and pasted it into a text file.

In [6]:
```
with open("data/un-sdg-goals-list.txt", "r") as f:
    goals_list = f.read().splitlines()
print(goals_list)
```

```
['No Poverty', 'Zero Hunger', 'Good Health and Well-being', 'Quality Education', 'Gender Equa
lity', 'Clean Water and Sanitation', 'Affordable and Clean Energy', 'Decent Work and Economic
Growth', 'Industry, Innovation and Infrastructure', 'Reduced Inequality', 'Sustainable Cities
and Communities', 'Responsible Consumption and Production', 'Climate Action', 'Life Below Wat
er', 'Life On Land', 'Peace', 'Justice', 'Strong Institutions', 'Partnerships for the Goals']
```

Now we can convert to percentages and add in the goals. The table shows the percentage of `null` values, and with a bit of formatting, we can easily see where there are holes and therefore columns to exclude. (The darker the green, the fewer the nulls.)

In [7]:
```
# Calculate percent of nulls per row
nulls_per_goal_percentages = nulls_per_goal.iloc[:,2:].divide(
    nulls_per_goal['num_rows'], axis=0
)

# Add in the goals and goal title columns
nulls_per_goal_percentages.insert(0, 'goal', nulls_per_goal['goal'])
```

```python
nulls_per_goal_percentages.insert(1, 'goal_title', pd.Series(goals_list))

# Define a dataframe to apply 'nowrap' class to all cells in the goal_title column
classes = pd.DataFrame(
    [['nowrap']],
    index=nulls_per_goal_percentages.index,
    columns=['goal_title']
)

idx = pd.IndexSlice
display(
    nulls_per_goal_percentages.style

    # Set the background gradient - Dark green for 0 nulls, light green for 100% nulls.
    .background_gradient(cmap='Greens_r', vmin=0, vmax=1, subset=idx[:, idx['target':]])

    # Hide the zeros and format other numbers as percentages.
    .format(lambda v: "" if v==0 else f"{v:.1%}", subset=idx[:, idx['target':]])

    # Apply the classes to the table cells.
    .set_td_classes(classes)

    .set_table_styles([

        # Add internal styles to nowrap class to prevent goal titles wrapping.
        {'selector': '.nowrap', 'props': 'min-width: 100px;'},

        # Add styles so we can have narrow columns but show the column title in full on hover.
        {'selector': '.col_heading', 'props': 'max-width: 30px; overflow: hidden'},
        {'selector': '.col_heading:hover', 'props': '; overflow: visible'},
        {'selector': '.col_heading:hover~.col_heading', 'props': '; visibility: hidden'},

    ], overwrite=False)
)
```

oal	goal_title	target	indicat	seriesc	seriesc	geoare	geoare	timepe	value	time_d	source	footnot	nature	age	bounds
1	No Poverty									14.9%		16.9%		86.7%	100.0%
2	Zero Hunger											13.5%		90.4%	90.4%
3	Good Health and Well-being									2.0%		85.4%		63.8%	82.9%
4	Quality Education									88.4%		95.4%		99.7%	100.0%
5	Gender Equality									9.6%		21.8%		69.7%	100.0%
6	Clean Water and Sanitation											88.4%		100.0%	100.0%
7	Affordable and Clean Energy									5.5%		100.0%		100.0%	100.0%
8	Decent Work and Economic Growth									3.9%		88.2%		90.7%	100.0%
9	Industry, Innovation and Infrastructure									0.0%		42.6%		100.0%	100.0%
10	Reduced Inequality									3.9%		75.0%		100.0%	100.0%
11	Sustainable Cities and Communities											6.9%		100.0%	100.0%
12	Responsible Consumption and Production											100.0%		100.0%	100.0%
13	Climate Action													100.0%	100.0%
14	Life Below Water										0.2%	93.4%		100.0%	7.4%
15	Life On Land											84.7%		100.0%	27.6%
16	Peace											80.0%		96.5%	100.0%
17	Justice									8.7%		50.3%		100.0%	100.0%

It looks like many columns are only relevant for certain goals. We should probably dive down to the level of indicators to understand what is being measured and which columns are relevant.

In [8]:

```
nulls_per_indicator = pd.read_csv('data/un-sdg-non-null-entries-by-indicator.csv')

# Add in the goal titles and reposition
nulls_per_indicator = nulls_per_indicator.merge(
    nulls_per_goal_percentages.loc[:, 'goal':'goal_title'],
    left_on='goal', right_on='goal',
)
nulls_per_indicator.insert(1, 'goal_title', nulls_per_indicator.pop('goal_title'))

total_rows = nulls_per_indicator['goal'].count()
print(f"Total number of indicators : {total_rows:,}")

with pd.option_context('display.max_colwidth', None, 'display.max_rows', None):
    display(nulls_per_indicator.head())
```

```
Total number of indicators : 374
```

	goal	goal_title	target	indicator	seriescode	seriesdescription	num_rows	geoareacode	geoarea
0	1	No Poverty	1.1	1.1.1	SI_POV_DAY1	Proportion of population below international poverty line (%)	1345	0	
1	1	No Poverty	1.1	1.1.1	SI_POV_EMP1	Employed population below international poverty line, by sex and age (%)	8370	0	
2	1	No Poverty	1.2	1.2.1	SI_POV_NAHC	Proportion of population living below the national poverty line (%)	732	0	
3	1	No Poverty	1.3	1.3.1	SI_COV_BENFTS	Proportion of population covered by at least one social protection benefit (%)	105	0	
4	1	No Poverty	1.3	1.3.1	SI_COV_CHLD	Proportion of children/households receiving child/family cash benefit (%)	94	0	

5 rows × 36 columns

A long list - 374 indicators! (You can see them all by removing `.head()` from the last line in the cell above. I do this throughout so that a pdf version isn't massive...)

A quick scan through and I decided it might be interesting to look at *No Poverty*, *Quality Education* and *Decent Work and Economic Growth* to see if there were any correlations.

In [9]:
```
selected_indicators = [1, 4, 8]
nulls_per_selected_indicators = nulls_per_indicator.loc[nulls_per_indicator['goal'].isin(select

descriptions_selected_indicators = nulls_per_selected_indicators[['goal', 'goal_title', 'series
with pd.option_context('display.max_colwidth', None):
    display(descriptions_selected_indicators.head())
```

	goal	goal_title	seriescode	seriesdescription
0	1	No Poverty	SI_POV_DAY1	Proportion of population below international poverty line (%)
1	1	No Poverty	SI_POV_EMP1	Employed population below international poverty line, by sex and age (%)
2	1	No Poverty	SI_POV_NAHC	Proportion of population living below the national poverty line (%)
3	1	No Poverty	SI_COV_BENFTS	Proportion of population covered by at least one social protection benefit (%)
4	1	No Poverty	SI_COV_CHLD	Proportion of children/households receiving child/family cash benefit (%)

Looking through the descriptions I initially chose 7 indicators that I thought of interest - the idea being that there might be a correlation between successful students and economic strength.

```
In [10]: selected_codes = [
    'SI_POV_DAY1',
    'SE_MAT_PROF',
    'SE_REA_PROF',
    'SE_ADT_EDUCTRN',
    'SE_ADT_FUNS',
    'SE_TRA_GRDL',
    'NY_GDP_PCAP',
    'SL_TLF_UEM',
]
nulls_per_selected_codes = nulls_per_selected_indicators[nulls_per_selected_indicators['series
with pd.option_context('display.max_colwidth', None):
    display(nulls_per_selected_codes.loc[:, 'goal':'num_rows'])
```

	goal	goal_title	target	indicator	seriescode	seriesdescription	num_rows
0	1	No Poverty	1.1	1.1.1	SI_POV_DAY1	Proportion of population below international poverty line (%)	1345
115	4	Quality Education	4.1	4.1.1	SE_MAT_PROF	Minimum proficiency in mathematics, by education level and sex (%)	2172
116	4	Quality Education	4.1	4.1.1	SE_REA_PROF	Minimum proficiency in reading, by education level and sex (%)	1698
119	4	Quality Education	4.3	4.3.1	SE_ADT_EDUCTRN	Participation rate in formal and non-formal education and training, by sex (%)	273
138	4	Quality Education	4.6	4.6.1	SE_ADT_FUNS	Proportion of population achieving at least a fixed level of proficiency in functional skills, by sex, age and type of skill (%)	68
147	4	Quality Education	4.c	4.c.1	SE_TRA_GRDL	Proportion of teachers who have received at least the minimum organized teacher training (e.g. pedagogical training) pre-service or in-service required for teaching at the relevant level in a given country, by education level (%)	12207
205	8	Decent Work and Economic Growth	8.1	8.1.1	NY_GDP_PCAP	Annual growth rate of real GDP per capita (%)	4210
218	8	Decent Work and Economic Growth	8.5	8.5.2	SL_TLF_UEM	Unemployment rate, by sex and age (%)	16508

Extracting the data

Finally! Let's get the data. Running the query in this form, selecting only the relevant columns and series codes, produced a file of less than 10MB.

```
SELECT
    seriescode, geoareacode, geoareaname, timeperiod, value, time_detail,
    nature, age, education_level, sex, type_of_skill, units
FROM bigquery-public-data.un_sdg.indicators
```

```
    WHERE
        seriescode IN (
            'SI_POV_DAY1',
            'SE_MAT_PROF',
            'SE_REA_PROF',
            'SE_ADT_EDUCTRN',
            'SE_ADT_FUNS',
            'SE_TRA_GRDL',
            'NY_GDP_PCAP',
            'SL_TLF_UEM'
        )
```

Here are the first few lines of the data.

In [11]:
```python
all_data = pd.read_csv('data/un-sdg-goals-selected-codes-results.csv')
all_data.head()
```

Out[11]:

	seriescode	geoareacode	geoareaname	timeperiod	value	time_detail	nature	age	education_level
0	SE_ADT_FUNS	250	France	2012	90.83946	NaN	C	16-65	NaN
1	SE_ADT_FUNS	616	Poland	2012	96.05922	NaN	C	16-65	NaN
2	SE_ADT_FUNS	418	Lao People's Democratic Republic	2012	73.78760	NaN	C	15-65	NaN
3	SE_ADT_FUNS	40	Austria	2012	96.53522	NaN	C	16-65	NaN
4	SE_ADT_FUNS	152	Chile	2015	79.63343	NaN	C	16-65	NaN

Manipulating the data using Pandas

Examining the data

I chose to look first at SE_MAT_PROF and SE_REA_PROF: *Minimum proficiency in mathematics, by education level and sex (%)* and *Minimum proficiency in reading, by education level and sex (%)*. Using `describe()` it looks like the columns of interest (in addition to `geoareacode`, `geoarename`, `value` and `timeperiod`) are `education_level` and `sex`, as `time_detail`, `age` and `type_of_skill` contain only `null` values, and `nature` and `units` have only one value.

In [12]:
```python
chart_series_codes = ['SE_MAT_PROF', 'SE_REA_PROF']

for chart_series_code in chart_series_codes:
    print(f"Describing series : {chart_series_code}")
    display(all_data[all_data['seriescode'] == chart_series_code].describe(include='all'))
```

```
Describing series : SE_MAT_PROF
```

	seriescode	geoareacode	geoareaname	timeperiod	value	time_detail	nature	age	educa
count	2172	2172.000000	2172	2172.000000	2172.000000	0.0	2172	0	
unique	1	NaN	131	NaN	NaN	NaN	1	0	
top	SE_MAT_PROF	NaN	China, Hong Kong Special Administrative Region	NaN	NaN	NaN	C	NaN	
freq	2172	NaN	36	NaN	NaN	NaN	2172	NaN	
mean	NaN	444.128453	NaN	2008.790055	69.058890	NaN	NaN	NaN	
std	NaN	242.341232	NaN	4.604399	23.171540	NaN	NaN	NaN	
min	NaN	8.000000	NaN	2000.000000	5.610000	NaN	NaN	NaN	
25%	NaN	246.000000	NaN	2006.000000	52.047895	NaN	NaN	NaN	
50%	NaN	428.000000	NaN	2009.000000	76.598550	NaN	NaN	NaN	
75%	NaN	643.000000	NaN	2012.000000	87.901095	NaN	NaN	NaN	
max	NaN	894.000000	NaN	2015.000000	99.870000	NaN	NaN	NaN	

```
Describing series : SE_REA_PROF
```

	seriescode	geoareacode	geoareaname	timeperiod	value	time_detail	nature	age	educa
count	1698	1698.000000	1698	1698.000000	1698.000000	0.0	1698	0	
unique	1	NaN	121	NaN	NaN	NaN	1	0	
top	SE_REA_PROF	NaN	Colombia	NaN	NaN	NaN	C	NaN	
freq	1698	NaN	27	NaN	NaN	NaN	1698	NaN	
mean	NaN	437.830389	NaN	2008.109541	73.414752	NaN	NaN	NaN	
std	NaN	240.655446	NaN	4.823925	20.409998	NaN	NaN	NaN	
min	NaN	8.000000	NaN	2000.000000	7.680000	NaN	NaN	NaN	
25%	NaN	233.000000	NaN	2006.000000	61.157138	NaN	NaN	NaN	
50%	NaN	428.000000	NaN	2009.000000	79.002285	NaN	NaN	NaN	
75%	NaN	642.000000	NaN	2012.000000	88.909627	NaN	NaN	NaN	
max	NaN	894.000000	NaN	2015.000000	99.670000	NaN	NaN	NaN	

We can use a pivot table to check that we've correctly identified all the dimensions by which we can group `value` :

In [13]:
```
se_prof_data = all_data[all_data['seriescode'].isin(chart_series_codes)]
se_prof_pivot_table = se_prof_data.pivot_table(
    'value',
    index=['seriescode', 'geoareaname', 'sex'],
    columns=['education_level', 'timeperiod'],
    aggfunc='count'
)
with pd.option_context('display.max_columns', None, 'display.max_rows', None):
    display(se_prof_pivot_table.head(10).style.format("{:.1f}", na_rep=""))
```

		education_level	GRAD23									
		timeperiod	2001	2003	2006	2007	2011	2013	2014	2015	2000	2003
seriescode	geoareaname	sex										
SE_MAT_PROF	Albania	BOTHSEX									1.0	
		FEMALE									1.0	
		MALE									1.0	
	Algeria	BOTHSEX				1.0						
		FEMALE				1.0						
		MALE				1.0						
	Argentina	BOTHSEX						1.0			1.0	
		FEMALE						1.0			1.0	
		MALE						1.0			1.0	
	Armenia	BOTHSEX										1.0

We can check visually that all the values in the table are 1.0 or empty, but better to check that programatically. We should get zero rows in the pivot table if we only display rows that contain anything other than 1.0 or NaN.

In [14]:
```
print("Displaying any rows where values are not 1.0 or NaN")
display(se_prof_pivot_table[se_prof_pivot_table.isin([1.0, np.nan]).all(axis=1) == False])
```

```
Displaying any rows where values are not 1.0 or NaN
```

		education_level	GRAD23										I
		timeperiod	2001	2003	2006	2007	2011	2013	2014	2015	2000	2003	...
seriescode	geoareaname	sex											

0 rows × 25 columns

It looks like wherever there is data, there is data for `BOTHSEX`, `FEMALE` and `MALE`. We can check that by dropping the grouping of `sex`. Visual check first...

In [15]:
```
se_prof_pivot_table = se_prof_data.pivot_table(
    'value',
    index=['seriescode', 'geoareaname'],
    columns=['education_level', 'timeperiod'],
    aggfunc='count'
)
with pd.option_context('display.max_columns', None, 'display.max_rows', None):
    display(se_prof_pivot_table.head(10).style.format("{:.1f}", na_rep=""))
```

	education_level								GRAD23				
	timeperiod	2001	2003	2006	2007	2011	2013	2014	2015	2000	2003	2006	2007
seriescode	geoareaname												
SE_MAT_PROF	Albania									3.0			
	Algeria				3.0								3.0
	Argentina						3.0			3.0		3.0	
	Armenia										3.0		3.0
	Australia		3.0		3.0	3.0			3.0	3.0	3.0	3.0	
	Austria									3.0	3.0	3.0	
	Azerbaijan											3.0	
	Bahrain					3.0			3.0		3.0		3.0
	Belgium									3.0	3.0	3.0	
	Benin							3.0					

Only 3.0 or empty. And the programmatic check confirms it.

In [16]:
```
print("Displaying any rows where values are not 3.0 or Nan")
display(se_prof_pivot_table[se_prof_pivot_table.isin([3.0, np.nan]).all(axis=1) == False])
```

```
Displaying any rows where values are not 3.0 or Nan
```

	education_level								GRAD23				LOWSEC	
	timeperiod	2001	2003	2006	2007	2011	2013	2014	2015	2000	2003	...	2015	2000
seriescode	geoareaname													

0 rows × 25 columns

Since timeperiod data is scattered all over the place we need to decide how to handle that. What's more, from the table below it's clear that many countries have no data for certain education levels, and some countries with no data for a given proficiency at all (e.g. Belize - SE_MAT_PROF).

In [17]:
```
with pd.option_context('display.max_columns', None, 'display.max_rows', None):
    display(
        se_prof_data.pivot_table(
            'value',
            index=['geoareaname'],
            columns=['seriescode','education_level'],
            aggfunc='count',
            fill_value = ""
        ).head(10)
    )
```

seriescode	SE_MAT_PROF			SE_REA_PROF		
education_level	GRAD23	LOWSEC	PRIMAR	GRAD23	LOWSEC	PRIMAR
geoareaname						
Albania		12.0			12.0	
Algeria	3.0	6.0			3.0	
Argentina	3.0	15.0	6.0	6.0	15.0	6.0
Armenia		9.0	9.0			
Australia	12.0	18.0		3.0	18.0	
Austria		18.0	6.0		18.0	6.0
Azerbaijan		6.0	3.0		6.0	3.0
Bahrain	6.0	12.0				
Belgium		18.0			18.0	
Belize				3.0		

Lastly, we should understand what the education levels actually mean. The full descriptions for all indicators can be found here on the UN SDG website and there we find this definition of indicator 4.1.1:

> Proportion of children and young people (a) in grades 2/3; (b) at the end of primary; and (c) at the end of lower secondary achieving at least a minimum proficiency level in (i) reading and (ii) mathematics, by sex

So let's create some human friendly labels we can use later for the education level codes:

In [18]:
```
edu_level_labels = {
    'GRAD23' : 'Grade 2/3',
    'PRIMAR' : 'End of primary',
    'LOWSEC' : 'End of lower secondary'
}
```

Selecting data

First I decided to simplify by only looking at the data for both sexes combined. We lose the `sex` dimension but we know we don't lose any coverage in the other dimensions thanks to the analysis above showing that where we have data, we have data for all categories of `sex`. So let's select only `BOTHSEX` data and drop the irrelevant columns of `time_detail`, `age`, `type_of_skill`, `nature` and `units`.

In [19]:
```
se_prof_chart_data = (
    se_prof_data[(se_prof_data['sex'] == 'BOTHSEX')]
    .drop(['time_detail', 'age', 'type_of_skill', 'nature', 'units'], axis=1)
)

se_prof_chart_data.pivot_table(
    'value',
    index=['geoareaname'],
    columns=['seriescode', 'education_level'],
    aggfunc='count',
    fill_value = ""
)
```

Out[19]:

seriescode	SE_MAT_PROF			SE_REA_PROF		
education_level	**GRAD23**	**LOWSEC**	**PRIMAR**	**GRAD23**	**LOWSEC**	**PRIMAR**
geoareaname						
Albania		4.0			4.0	
Algeria	1.0	2.0			1.0	
Argentina	1.0	5.0	2.0	2.0	5.0	2.0
Armenia		3.0	3.0			
Australia	4.0	6.0		1.0	6.0	
...	...	...	...	...	...	...
Venezuela (Bolivarian Republic of)		1.0			1.0	
Viet Nam		2.0			2.0	
Yemen	3.0					
Zambia			2.0			2.0
Zimbabwe			1.0			1.0

132 rows × 6 columns

Now we need to decide how to handle the timeperiod. We know our data is scattered all over the place time-wise as shown below.

In [20]:

```
total_countries = len(se_prof_chart_data['geoareacode'].unique())

data_time_coverage = (
    se_prof_chart_data.pivot_table(
        'value',
        index=['timeperiod'],
        columns=['seriescode', 'education_level'],
        aggfunc='count',
        margins=True,
        margins_name='Total'
    )
    .style
    .set_caption(f'Number of countries (out of {total_countries}) with data for given year and
    .set_table_styles([
        {
            'selector': 'caption',
            'props': [('font-weight', 'bold'), ('font-size', '16px'), ('color', 'black')]
        },
        {
            'selector': 'th',
            'props': [('text-align', 'left')]
        },
    ])
    .format("{:.0f}", na_rep="")
)
data_time_coverage
```

Out[20]:

Number of countries (out of 132) with data for given year and education level

seriescode	SE_MAT_PROF			SE_REA_PROF			Total
education_level	**GRAD23**	**LOWSEC**	**PRIMAR**	**GRAD23**	**LOWSEC**	**PRIMAR**	
timeperiod							
2000		42	15		43	2	102
2001				25		22	47
2003	17	67	8		41	2	135
2006		56	19	27	55	47	204
2007	25	45	29				99
2009		72			72		144
2011	36	42	15	36		12	141
2012		62			62		124
2013	15		15	15		15	60
2014	10		10	10		10	40
2015	35	79	10		70		194
Total	138	465	121	113	343	110	1290

Given the goal of trying to compare with some economic indicator, the ideal would be to have complete data for a given year for both educational proficiency and economic indicator. Clearly we don't have that for educational proficiency; what about economic indicator?

Initially I had the idea of using *Annual growth rate of real GDP per capita (%)* as the economic indicator, but when I looked at the data I saw that it varied significantly from one year to the next for many countries (unsurprisingly). I wanted something that better reflected the economic strength of a country over time (i.e., GDP per capita), but nothing like that was available in the UN SDG dataset. So I had to look elsewhere...

Sourcing GDP per capita data

I found GDP per capita data (also from the UN) at National Accounts - Analysis of Main Aggregates (AMA). The important thing to check was that it followed the same coding system for countries so it would be easy to marry up the data - and yes they both follow the M49 system.

Let's see what we've got...

In [21]:

```
gdp_per_capita_raw_data = (
    pd.read_csv('data/un-gdp-per-capita-2000-2015.csv')

    # Remove Unit column as it's US$ throughout
    .drop('Unit', axis=1)

    # Rename the column for easier referencing
    .rename({'GDP, Per Capita GDP - US Dollars' : 'GDP per capita'}, axis=1)
)

gdp_per_capita_raw_data.pivot_table(
```

```
    'GDP per capita',
    index=['Country/Area'],
    columns=['Year'],
    aggfunc='sum',
    fill_value = ""
)
```

Out[21]:

Year	2000	2001	2002	2003	
Country/Area					
Afghanistan	160.82972700182722	166.54198056776517	183.24702843070884	199.69882776561923	217.92174427
Africa	810.8811716574647	776.1216123868226	779.3997701125529	904.2519946405133	1073.0086237
Albania	1114.514373610592	1254.7153329890555	1393.3478351464644	1783.6492741179677	2311.5233524
Algeria	1761.0489984569401	1750.5272737364571	1783.6765490570183	2103.382140931998	2610.182685
Americas	15975.197996901728	16108.949756545497	16072.081954675621	16700.722509128387	17877.261397
...	...	...	...	...	
Western Europe	24219.62127445152	24342.83003104896	26105.249897930327	31623.622994493973	35806.3559
World	5491.033875998066	5392.494201722851	5534.590780143235	6133.366223652267	6819.8508163
Yemen	624.0747832656903	627.3139871514516	664.0258848351101	714.0105620442428	799.210563
Zambia	345.6844982573868	382.93846650227215	382.24157911549463	435.46074766810983	538.5942809
Zimbabwe	733.9588613435855	726.2036336390886	687.2401495195448	646.4996163483507	616.9989551

244 rows × 16 columns

The above all looks as we'd expect - let's check the data types:

In [22]:
```
gdp_per_capita_raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3904 entries, 0 to 3903
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Country/Area    3904 non-null   object
 1   Year            3904 non-null   int64 
 2   GDP per capita  3904 non-null   object
dtypes: int64(1), object(2)
memory usage: 91.6+ KB
```

Strange that GDP per capita is type `object` and not `float`. Let's try to convert...

In [23]:
```
gdp_per_capita_raw_data['GDP per capita'] = pd.to_numeric(gdp_per_capita_raw_data['GDP per cap
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File ~\miniconda3\envs\datascience\lib\site-packages\pandas\_libs\lib.pyx:2315, in pandas._li
bs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "..."

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
Input In [23], in <cell line: 1>()
----> 1 gdp_per_capita_raw_data['GDP per capita'] = pd.to_numeric(gdp_per_capita_raw_data['GD
P per capita'])

File ~\miniconda3\envs\datascience\lib\site-packages\pandas\core\tools\numeric.py:184, in to_
numeric(arg, errors, downcast)
    182 coerce_numeric = errors not in ("ignore", "raise")
    183 try:
--> 184     values, _ = lib.maybe_convert_numeric(
    185         values, set(), coerce_numeric=coerce_numeric
    186     )
    187 except (ValueError, TypeError):
    188     if errors == "raise":

File ~\miniconda3\envs\datascience\lib\site-packages\pandas\_libs\lib.pyx:2357, in pandas._li
bs.lib.maybe_convert_numeric()

ValueError: Unable to parse string "..." at position 912
```

The error message tells us what's happening - there is at least one entry that is a string `'...'`. Let's check where that happens...

In [24]:
```
gdp_per_capita_raw_data[gdp_per_capita_raw_data['GDP per capita'] == '...'].pivot_table(
    'GDP per capita',
    index=['Country/Area'],
    columns=['Year'],
    aggfunc='count',
    fill_value = ""
)
```

Out[24]:

Year	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	20
Country/Area															
Curaçao	1.0	1.0	1.0	1.0	1.0										
Former Netherlands Antilles											1.0	1.0	1.0	1.0	1
Former Sudan									1.0	1.0	1.0	1.0	1.0	1.0	1
Sint Maarten (Dutch part)	1.0	1.0	1.0	1.0	1.0										
South Sudan	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0							
Sudan	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0							

The string is pretty rare - let's just check whether we're even using these countries...

In [25]:
```
se_prof_chart_data['geoareaname'].unique()
```

Out[25]:
```
array(['Albania', 'Denmark', 'Finland', 'Kazakhstan', 'Norway', 'Poland',
       'Saudi Arabia', 'Uruguay', 'Austria', 'Colombia', 'Comoros',
       'France', 'Ghana', 'Japan', 'Portugal', 'Bulgaria',
       'Iran (Islamic Republic of)', 'Algeria', 'Armenia', 'Australia',
       'Chad', 'China, Hong Kong Special Administrative Region',
       'Tunisia', 'United Arab Emirates', 'Azerbaijan', 'Chile',
       'New Zealand', 'Thailand', 'Brazil', 'Lebanon', 'Luxembourg',
       'Republic of Moldova', 'Slovakia', 'Slovenia',
       'United Kingdom of Great Britain and Northern Ireland',
       'Burkina Faso', 'Hungary', 'Malaysia', 'Russian Federation',
       'Estonia', 'Georgia', 'Kenya', 'Belgium',
       'China, Macao Special Administrative Region', 'Czechia',
       'Guatemala', 'Mauritius', 'Netherlands', 'Iceland', 'Indonesia',
       'Switzerland', 'Croatia', 'Lithuania', 'Germany', 'Panama',
       'Canada', 'Jordan', 'Mexico', 'Bahrain', 'Egypt', 'Malta', 'Spain',
       'Sweden', 'Peru', 'Singapore', 'Botswana',
       'The former Yugoslav Republic of Macedonia', 'Zimbabwe', 'Morocco',
       'Dominican Republic', 'Uganda', 'Israel', 'Liechtenstein',
       'Malawi', 'Serbia', 'El Salvador', 'Trinidad and Tobago', 'Kuwait',
       'Ecuador', 'Togo', 'Latvia', 'Montenegro', 'China', 'Gabon',
       'Argentina', 'Nicaragua', 'Romania', 'Italy', 'Philippines',
       'Turkey', 'Viet Nam', 'Qatar', 'Yemen', 'Benin', 'Ireland',
       'South Africa', 'State of Palestine',
       'United Republic of Tanzania', 'Paraguay',
       'United States of America', 'Burundi', 'Cuba', 'Cyprus', 'Senegal',
       'Kyrgyzstan', 'Mongolia', 'Syrian Arab Republic', 'Niger',
       'Costa Rica', 'Lesotho', 'Congo', 'Cameroon', 'Republic of Korea',
       'Honduras', 'Greece', 'Oman', 'Namibia', "Côte d'Ivoire",
       'Puerto Rico', 'Mozambique', 'Belize',
       'Democratic Republic of the Congo', 'Seychelles',
       'Bosnia and Herzegovina', 'Mauritania', 'Zambia', 'Eswatini',
       'Ukraine', 'Venezuela (Bolivarian Republic of)', 'Madagascar',
       'India', 'Mali'], dtype=object)
```

None of the countries with missing GDP data appear in our list of countries with educational proficiency data, so let's just convert to numeric and drop the invalid values:

In [26]:
```
# Convert to numeric
gdp_per_capita_raw_data.loc[:, 'GDP per capita'] = pd.to_numeric(gdp_per_capita_raw_data['GDP
len(gdp_per_capita_raw_data)
```

Out[26]:
```
3904
```

In [27]:
```
gdp_per_capita_cleaned_data = gdp_per_capita_raw_data.dropna()
len(gdp_per_capita_cleaned_data)
```

Out[27]:
```
3864
```

Now we should check that the countries do indeed match up. I had assumed that the country names used would be identical in both sets of data since both use the M49 system, so I used the country name as a key to merge. I expected all countries in the educational proficiency data to be found in the GDP per capita data, but not the reverse, as certainly we don't have educational proficiency data for all countries.

As a result, when merging the two lists of country names together using an outer join I expected to only have `null` values in the educational proficiency column, but that turned out not to be the case...

In [28]:
```
series_data_countries = pd.DataFrame(
    {'Proficiency country' : se_prof_chart_data['geoareaname'].unique()}
```

```
)

gdp_countries = pd.DataFrame(
    {'GDP country' : gdp_per_capita_cleaned_data['Country/Area'].unique()}
)

merged_countries = series_data_countries.merge(
    gdp_countries,
    how='outer',
    left_on = 'Proficiency country',
    right_on = 'GDP country'
)
mismatched_countries = merged_countries[merged_countries.isna().any(axis=1)]
with pd.option_context('display.max_rows', None):
    display(mismatched_countries.head(10))
```

	Proficiency country	GDP country
16	Iran (Islamic Republic of)	NaN
21	China, Hong Kong Special Administrative Region	NaN
66	The former Yugoslav Republic of Macedonia	NaN
82	China	NaN
97	United Republic of Tanzania	NaN
99	United States of America	NaN
126	Eswatini	NaN
132	NaN	Afghanistan
133	NaN	Africa
134	NaN	Americas

7 countries in educational proficiency couldn't be matched with GDP countries. Looking through the full list of countries in the GDP data I could find the corresponding matches, so let's fix those now. (The lesson in hindsight was to use the M49 codes, not the country names - my assumption that the names would also be consistent was flawed!)

In [29]:
```
country_fixes = {
    'China (mainland)' : 'China',
    'China, Hong Kong SAR' : 'China, Hong Kong Special Administrative Region',
    'Iran, Islamic Republic of' : 'Iran (Islamic Republic of)',
    'United States' : 'United States of America',
    'Kingdom of Eswatini' : 'Eswatini',
    'United Republic of Tanzania: Mainland' : 'United Republic of Tanzania',
    'Republic of North Macedonia' : 'The former Yugoslav Republic of Macedonia'
}
```

In [30]:
```
for old, new in country_fixes.items():
    gdp_per_capita_cleaned_data.loc[gdp_per_capita_cleaned_data['Country/Area'] == old, 'Count
```

Picking the year

We have almost entirely complete data for all years for GDP per capita. Since this is certainly not the case for our educational proficiency data, we need to choose how to handle the year. Options are:

1. **Take an average of values over the whole time period** - not great because changes in how education is delivered over the time period might affect the results.
2. **Pick a year and only use data from that year** - not great because we ignore a lot of data that could be useful.
3. **Pick a year and use data from that year when it is available, and when it is not, replace with data from the closest year where data is available** - a hybrid between 1 and 2.

Option 3 seemed the best bet, so we only have to pick the year. I wanted to make it as recent as possible, so I chose 2015 as it doesn't seem to stand out as being particularly poor in coverage compared to other years.

In [31]:
```
data_time_coverage
```

Out[31]:

Number of countries (out of 132) with data for given year and education level

seriescode	SE_MAT_PROF			SE_REA_PROF			Total
education_level	GRAD23	LOWSEC	PRIMAR	GRAD23	LOWSEC	PRIMAR	
timeperiod							
2000		42	15		43	2	102
2001				25		22	47
2003	17	67	8		41	2	135
2006		56	19	27	55	47	204
2007	25	45	29				99
2009		72			72		144
2011	36	42	15	36		12	141
2012		62			62		124
2013	15		15	15		15	60
2014	10		10	10		10	40
2015	35	79	10		70		194
Total	138	465	121	113	343	110	1290

The ideal would be to always use the same year for both sets of data. So if 2015 data is missing for, say, Algeria educational proficiency data, but we can replace with 2014 data, then we should really use 2014 GDP per capita data too. Let's do that then.

First we set the year we want...

In [32]:
```
selected_year = 2015
```

We're going to iterate through each educational proficiency, each education level and each country to then select the data with year closest to 2015. For that we'll need a list of countries grouped by `seriescode` and `education_level`, so let's create that now.

In [33]:
```
unique_country_edu_level = (
    se_prof_chart_data.groupby(['seriescode', 'education_level', 'geoareaname'], as_index=Fals
    .agg({'timeperiod' : 'count'})
```

```
)
unique_country_edu_level.head()
```

Out[33]:

	seriescode	education_level	geoareaname	timeperiod
0	SE_MAT_PROF	GRAD23	Algeria	1
1	SE_MAT_PROF	GRAD23	Argentina	1
2	SE_MAT_PROF	GRAD23	Australia	4
3	SE_MAT_PROF	GRAD23	Bahrain	2
4	SE_MAT_PROF	GRAD23	Benin	1

Now let's cycle through, and for each combination of `seriescode`, `education_level` and `geoareaname`, we will:

- Retrieve the index of the row in the educational proficiency data with the year closest to 2015, and add it to a list
- Retrieve the index of the row in the GDP data with the corresponding year and add it to a set so that we don't have duplicates

Then we use those indices to create two dataframes, one for education proficiency data and one for GDP data, each containing only data corresponding to the selected year (either 2015 or the closest year to it).

In [34]:

```
selected_indices_prof_data = []
selected_indices_gdp_data = set()
edu_levels = ['GRAD23', 'PRIMAR', 'LOWSEC']

for series_code in chart_series_codes:
    series_data = se_prof_chart_data[se_prof_chart_data['seriescode'] == series_code]

    for edu_level in edu_levels:

        # Select all the countries for this series code and education level
        countries = (
            unique_country_edu_level[
                (unique_country_edu_level['seriescode'] == series_code) &
                (unique_country_edu_level['education_level'] == edu_level)
            ]
            .loc[:, 'geoareaname']
        )

        # Iterate through extracting data for all years for each country in turn
        for country in countries:
            country_data = series_data[
                (series_data['geoareaname'] == country) &
                (series_data['education_level'] == edu_level)
            ]

            # Select index of row if selected year exists...
            if selected_year in country_data['timeperiod'].values:
                best_match_year = selected_year

            # ...otherwise we search for the closest year and use the index for that row
            else:
                closest_year = country_data['timeperiod'].values[0]
                smallest_gap = abs(closest_year - selected_year)
                for year in country_data['timeperiod'].values:
                    if abs(year - selected_year) < smallest_gap:
```

```python
                    smallest_gap = abs(year - selected_year)
                    closest_year = year
            best_match_year = closest_year

        # Add the index of the selected row for educational proficiency data
        selected_indices_prof_data.extend(
            series_data[
                (series_data['geoareaname'] == country) &
                (series_data['education_level'] == edu_level) &
                (series_data['timeperiod'] == best_match_year)
            ].index.to_list()
        )

        # Get the correct row for gdp data
        gdp_row = gdp_per_capita_cleaned_data[
            (gdp_per_capita_cleaned_data['Country/Area'] == country) &
            (gdp_per_capita_cleaned_data['Year'] == best_match_year)
        ]

        # This should never happen, but just in case, throw an error if the data is missin
        if len(gdp_row) == 0:
            raise ValueError(f"No GDP per capita data for {country} in {best_match_year}")

        selected_indices_gdp_data.add(gdp_row.index[0])

# Select only the GDP data with the selected indices
gdp_per_capita_selected_year = gdp_per_capita_cleaned_data.loc[list(selected_indices_gdp_data)
with pd.option_context('display.max_rows', None):
    display(gdp_per_capita_selected_year.head())

# Select only the educational data with the selected indices
se_prof_chart_data = se_prof_chart_data.loc[selected_indices_prof_data]
with pd.option_context('display.max_columns', None, 'display.max_rows', None):
    display(se_prof_chart_data.head())
```

	Country/Area	Year	GDP per capita
2054	Malawi	2006	308.163186
2055	Malawi	2007	332.259176
2571	Norway	2011	100697.293522
3083	Singapore	2011	53072.918866
2575	Norway	2015	74194.945777

	seriescode	geoareacode	geoareaname	timeperiod	value	education_level	sex
676	SE_MAT_PROF	12	Algeria	2007	40.93487	GRAD23	BOTHSEX
4644	SE_MAT_PROF	32	Argentina	2013	72.24475	GRAD23	BOTHSEX
4953	SE_MAT_PROF	36	Australia	2015	90.56623	GRAD23	BOTHSEX
4727	SE_MAT_PROF	48	Bahrain	2015	72.47670	GRAD23	BOTHSEX
910	SE_MAT_PROF	204	Benin	2014	33.58194	GRAD23	BOTHSEX

Now all we have to do is marry up the two tables using the year and country info, and we can start plotting! But before doing that, I decided it would probably be interesting to be able to view the data by region / continent as well as country...

Add region information

I downloaded the full list of M49 data (which includes regional groupings of countries) here. Here's what the data looks like...

In [35]:
```
region_info = pd.read_csv('data/unsd-m49-geoareacodes.csv', sep=";")
region_info.head()
```

Out[35]:

	Global Code	Global Name	Region Code	Region Name	Sub-region Code	Sub-region Name	Intermediate Region Code	Intermediate Region Name	Country or Area	M49 Code	ISO-alpha2 Code	al[illegible]
0	1	World	2.0	Africa	15.0	Northern Africa	NaN	NaN	Algeria	12	DZ	
1	1	World	2.0	Africa	15.0	Northern Africa	NaN	NaN	Egypt	818	EG	
2	1	World	2.0	Africa	15.0	Northern Africa	NaN	NaN	Libya	434	LY	
3	1	World	2.0	Africa	15.0	Northern Africa	NaN	NaN	Morocco	504	MA	
4	1	World	2.0	Africa	15.0	Northern Africa	NaN	NaN	Sudan	729	SD	

Each country is mapped to a Region and Sub-region.

In [36]:
```
region_mapping = region_info.loc[:, ['Region Name', 'Sub-region Name', 'M49 Code']]
region_mapping.groupby(['Region Name', 'Sub-region Name']).agg('count')
```

Out[36]:

Region Name	Sub-region Name	M49 Code
Africa	Northern Africa	7
	Sub-Saharan Africa	53
Americas	Latin America and the Caribbean	52
	Northern America	5
Asia	Central Asia	5
	Eastern Asia	7
	South-eastern Asia	11
	Southern Asia	9
	Western Asia	18
Europe	Eastern Europe	10
	Northern Europe	17
	Southern Europe	16
	Western Europe	9
Oceania	Australia and New Zealand	6
	Melanesia	5
	Micronesia	8
	Polynesia	10

Let's split the Americas into the conventional North and "South" (i.e., Latin America and the Caribbean)

In [37]:
```
for americas_region in ['Latin America and the Caribbean', 'Northern America']:
    region_mapping.loc[
        region_mapping['Sub-region Name'] == americas_region,
        'Region Name'
    ] = americas_region
region_mapping.groupby(['Region Name', 'Sub-region Name']).agg('count')
```

Out[37]:

Region Name	Sub-region Name	M49 Code
Africa	Northern Africa	7
	Sub-Saharan Africa	53
Asia	Central Asia	5
	Eastern Asia	7
	South-eastern Asia	11
	Southern Asia	9
	Western Asia	18
Europe	Eastern Europe	10
	Northern Europe	17
	Southern Europe	16
	Western Europe	9
Latin America and the Caribbean	Latin America and the Caribbean	52
Northern America	Northern America	5
Oceania	Australia and New Zealand	6
	Melanesia	5
	Micronesia	8
	Polynesia	10

And now we can drop the sub-region mapping, and merge in with our educational proficiency data.

In [38]:

```
region_mapping.drop('Sub-region Name', axis=1, inplace=True)

se_prof_chart_data = se_prof_chart_data.merge(
    region_mapping,
    how = 'left',
    left_on = 'geoareacode',
    right_on = 'M49 Code'
)
se_prof_chart_data.head(10)
```

Out[38]:

	seriescode	geoareacode	geoareaname	timeperiod	value	education_level	sex	Region Name	M4 Cod
0	SE_MAT_PROF	12	Algeria	2007	40.93487	GRAD23	BOTHSEX	Africa	1
1	SE_MAT_PROF	32	Argentina	2013	72.24475	GRAD23	BOTHSEX	Latin America and the Caribbean	3
2	SE_MAT_PROF	36	Australia	2015	90.56623	GRAD23	BOTHSEX	Oceania	3
3	SE_MAT_PROF	48	Bahrain	2015	72.47670	GRAD23	BOTHSEX	Asia	4
4	SE_MAT_PROF	204	Benin	2014	33.58194	GRAD23	BOTHSEX	Africa	20
5	SE_MAT_PROF	72	Botswana	2011	60.65287	GRAD23	BOTHSEX	Africa	7
6	SE_MAT_PROF	76	Brazil	2013	70.23770	GRAD23	BOTHSEX	Latin America and the Caribbean	7
7	SE_MAT_PROF	854	Burkina Faso	2014	59.23316	GRAD23	BOTHSEX	Africa	85
8	SE_MAT_PROF	108	Burundi	2014	96.66259	GRAD23	BOTHSEX	Africa	10
9	SE_MAT_PROF	120	Cameroon	2014	55.34140	GRAD23	BOTHSEX	Africa	12

Creating plottable data

We merge the educational proficiency data with the GDP per capita.

In [39]:

```
prof_vs_gdp_data = (
    se_prof_chart_data.rename(
        {'geoareaname' : 'Country', 'timeperiod' : 'Year'},
        axis=1
    )
    .set_index(['Country', 'Year'])
    .merge(
        gdp_per_capita_cleaned_data.rename(
            {'Country/Area' : 'Country'},
            axis=1
        ).set_index(['Country', 'Year']),
        how='outer',
        left_index=True,
        right_index=True
    )
    .drop('sex', axis=1)
    .dropna(subset='seriescode')
)
display(prof_vs_gdp_data)
```

		seriescode	geoareacode	value	education_level	Region Name	M49 Code	GDP per capita
Country	**Year**							
Albania	**2015**	SE_MAT_PROF	8.0	46.71703	LOWSEC	Europe	8.0	3939.413126
	2015	SE_REA_PROF	8.0	49.72273	LOWSEC	Europe	8.0	3939.413126
Algeria	**2007**	SE_MAT_PROF	12.0	40.93487	GRAD23	Africa	12.0	3950.513625
	2015	SE_MAT_PROF	12.0	19.04324	LOWSEC	Africa	12.0	4177.884976
	2015	SE_REA_PROF	12.0	21.03063	LOWSEC	Africa	12.0	4177.884976
...	**...**	...	...	...	...	...	...	...
Yemen	**2011**	SE_MAT_PROF	887.0	9.39027	GRAD23	Asia	887.0	1305.418200
Zambia	**2006**	SE_REA_PROF	894.0	55.91099	PRIMAR	Africa	894.0	1047.926445
	2007	SE_MAT_PROF	894.0	32.67655	PRIMAR	Africa	894.0	1124.284733
Zimbabwe	**2006**	SE_REA_PROF	716.0	81.49745	PRIMAR	Africa	716.0	579.898236
	2007	SE_MAT_PROF	716.0	73.45482	PRIMAR	Africa	716.0	567.749599

432 rows × 7 columns

In [40]:
```
prof_vs_gdp_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 432 entries, ('Albania', 2015) to ('Zimbabwe', 2007)
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   seriescode       432 non-null    object 
 1   geoareacode      432 non-null    float64
 2   value            432 non-null    float64
 3   education_level  432 non-null    object 
 4   Region Name      432 non-null    object 
 5   M49 Code         432 non-null    float64
 6   GDP per capita   432 non-null    float64
dtypes: float64(4), object(3)
memory usage: 35.7+ KB
```

Plotting

In [41]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
plt.style.use('seaborn-whitegrid')
```

For the first chart, let's take each education level and plot Educational proficiency against GDP per capita. We can show the two series (Maths and Reading) separately on the same chart, using different colours. I decided I also wanted to show how close to 2015 each data point was, so for that I used the size and transparency of the dots.

First a bit of preparation...

In [42]:
```
# Set human friendly labels.
series_labels = {
    'SE_REA_PROF' : 'Reading',
```

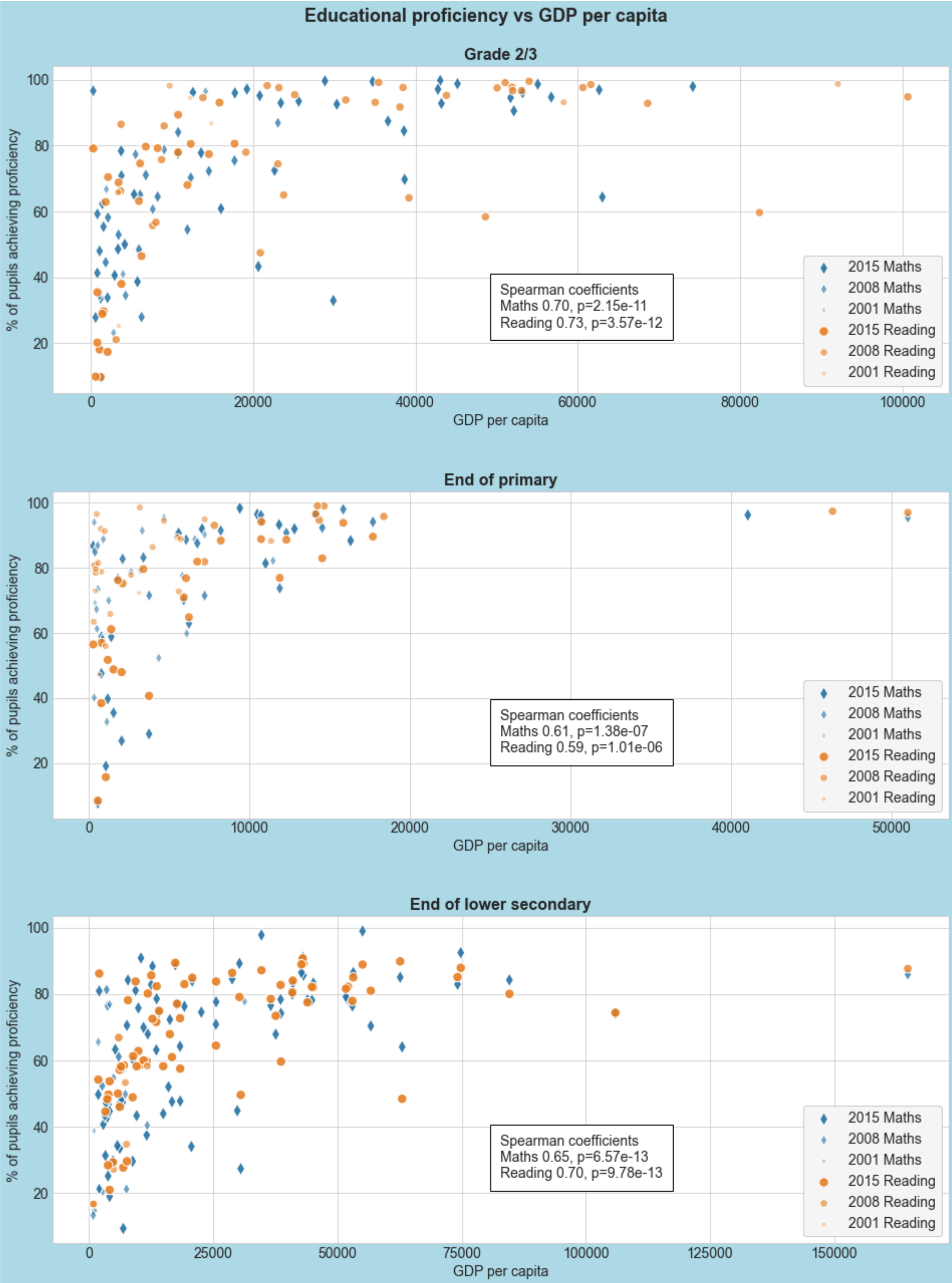
```
    'SE_MAT_PROF' : 'Maths',
}

'''
Alpha (transparency) ranges from 0 to 1. We want marker size value to be proportional to alpha
we fix a scaling value that will give a sensible size for the markers when we plot them.
'''
marker_alpha_scaling = 100

# Get the earliest and latest years of all data to use as bounds for alpha and marker size.
years = prof_vs_gdp_data.index.get_level_values(1)
min_year = years.min()
max_year = years.max()

'''
Define two functions we will use to convert a year to a colour of the required transparency.

The first is used to generate the alpha value for a year. We want to convert to the range
0.3 to 0.9: 0.3 so that the lowest year is still reasonably visible, and 0.9 so that 2015
(the highest year) is not entirely opaque and we can see other colour dots through it if a
dot overlaps.

The second function combines a single rgb colour with an array of alpha values to create an
array of rgba values.
'''

def convert_year_to_alpha(year, min, max):
    '''
    Converts an array of year values to an array of corresponding alpha values.
    A year equal to the maximum value returns an alpha of 0.9 (almost opaque),
    while a year equal to the minimum value returns an alpha of 0.3. Year values
    inbetween min and max return alpha values mapped linearly between 0.3 and 0.9.

            Parameters:
                    year (numpy.ndarray of dtype int64): Array of year values
                    min (int): Earliest year
                    max (int): Latest year

            Returns:
                    (numpy.ndarray of dtype float64): Array of alpha values
    '''
    return 0.3 + 0.6 * (year - min) / (max - min)

def rgb_to_rgba(rgb, a):
    '''
    Combines a single rgb value with an array of alpha values to generate an
    array of rgba values.

            Parameters:
                    rgb (list): List of normalised rgb values e.g. [128/255, 0, 1]
                    a (numpy.ndarray of dtype float64): Array of alpha values

            Returns:
                    (numpy.ndarray of dtype float64): Array of rgba values
    '''
    rgb = np.array([rgb]*len(a))
    return np.append(np.array(rgb), a.reshape(len(a),1),axis=1)

# Set up years for a legend - we choose the earliest, latest and halfway inbetween.
legend_years = np.array([max_year, round((max_year + min_year) / 2), min_year])
legend_alphas = convert_year_to_alpha(legend_years, min_year, max_year)

# Import and define a function in preparation for calculating Spearman rank coefficienets and
from scipy.stats import spearmanr
```

```python
from math import ceil

def spearmanr_pval(x,y):
    return spearmanr(x,y)[1]
```

And now we can produce some plots.

In [43]:
```python
# Create 3 plots in a column.
fig, axs = plt.subplots(3, 1, figsize=(16, 20))
plt_num = 0

# Define our colours
series_colors = {
    'SE_REA_PROF' : [230 / 255, 126 / 255, 34 / 255], # Orange
    'SE_MAT_PROF' : [36 / 255, 113 / 255, 163 / 255], # Blue
}

# Define series markers as circles and thin diamonds.
series_markers = {'SE_REA_PROF' : 'o', 'SE_MAT_PROF' : 'd'}

for edu_level in edu_levels:

    spearman_labels = ['Spearman coefficients']

    axs[plt_num].set_xlabel('GDP per capita', fontsize=14)
    axs[plt_num].set_ylabel('% of pupils achieving proficiency', fontsize=14)
    axs[plt_num].tick_params(axis='both', labelsize=14)

    for series_code in chart_series_codes:

        chart_data = prof_vs_gdp_data.loc[
            (prof_vs_gdp_data['seriescode'] == series_code) &
            (prof_vs_gdp_data['education_level'] == edu_level)
        ]

        alphas = convert_year_to_alpha(
            chart_data.index.get_level_values(1).to_numpy(),
            min_year,
            max_year
        )

        axs[plt_num].scatter(
            chart_data['GDP per capita'],
            chart_data['value'],
            edgecolors = 'w',
            c = rgb_to_rgba(series_colors[series_code], alphas),
            s = alphas * marker_alpha_scaling,
            marker = series_markers[series_code]
        )

        # Add blank series that will define the legend
        for i in range(len(legend_years)):
            rgba = series_colors[series_code].copy()
            rgba.append(legend_alphas[i])
            axs[plt_num].scatter(
                [], [],
                edgecolors = 'w',
                c = [rgba],
                s = legend_alphas[i] * marker_alpha_scaling,
                marker = series_markers[series_code],
                label = f"{str(legend_years[i])} {series_labels[series_code]}"
            )
```

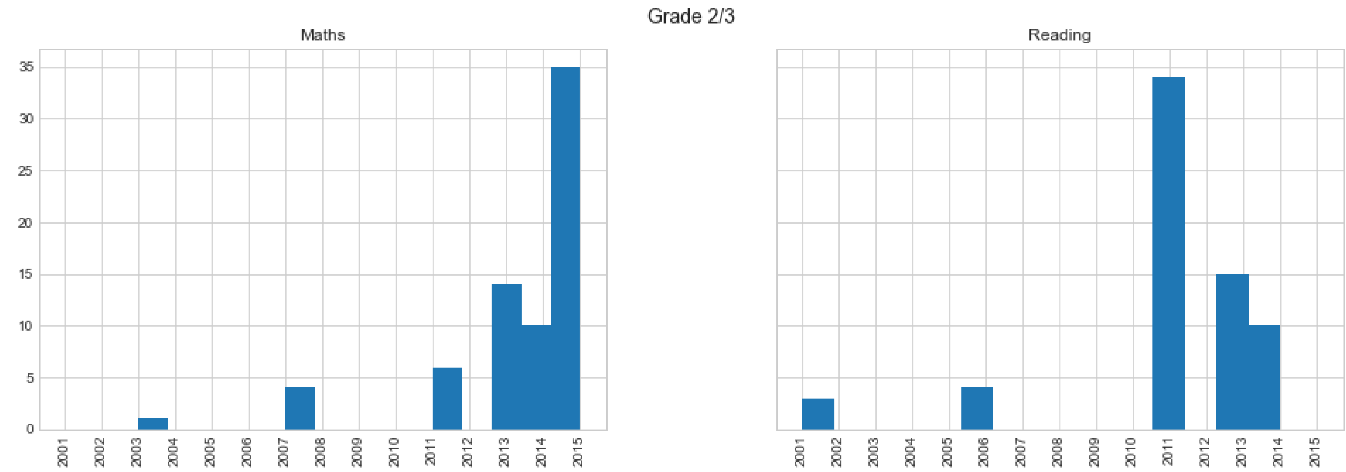
```
        # Calculate Spearman coefficient and p value
        spearman = chart_data.loc[:, ['GDP per capita', 'value']].corr('spearman').at['GDP per
        if not np.isnan(spearman):
            pval = chart_data.loc[:, ['GDP per capita', 'value']].corr(spearmanr_pval).at['GDP
            spearman_labels.append(f"{series_labels[series_code]} {spearman:.2f}, p={pval:.2e}

    axs[plt_num].set_title(
        edu_level_labels[edu_level],
        fontdict = {'fontsize' : 16, 'fontweight' : 'bold'}
    )

    axs[plt_num].legend(loc='lower right', frameon=True, facecolor='whitesmoke', framealpha=1,

    axs[plt_num].text(
        0.5, 0.2,
        "\n".join(spearman_labels),
        transform=axs[plt_num].transAxes,
        fontsize=14,
        bbox={
            'facecolor' : 'white',
            'pad' : 10
        }
    )

    plt_num += 1

fig.set_facecolor('lightblue')
plt.subplots_adjust(hspace=0.3, top=0.94)
fig.suptitle('Educational proficiency vs GDP per capita', fontsize=18, fontweight='bold')
plt.show()

# Uncomment for a copy to display in results
# fig.savefig(fname='images/result1.png', bbox_inches='tight')
```

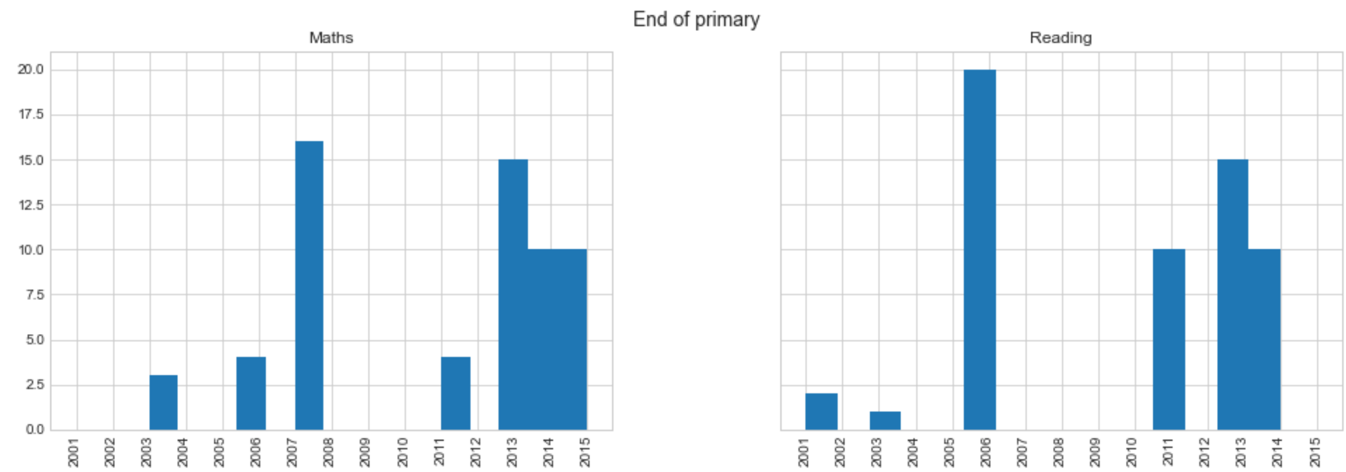


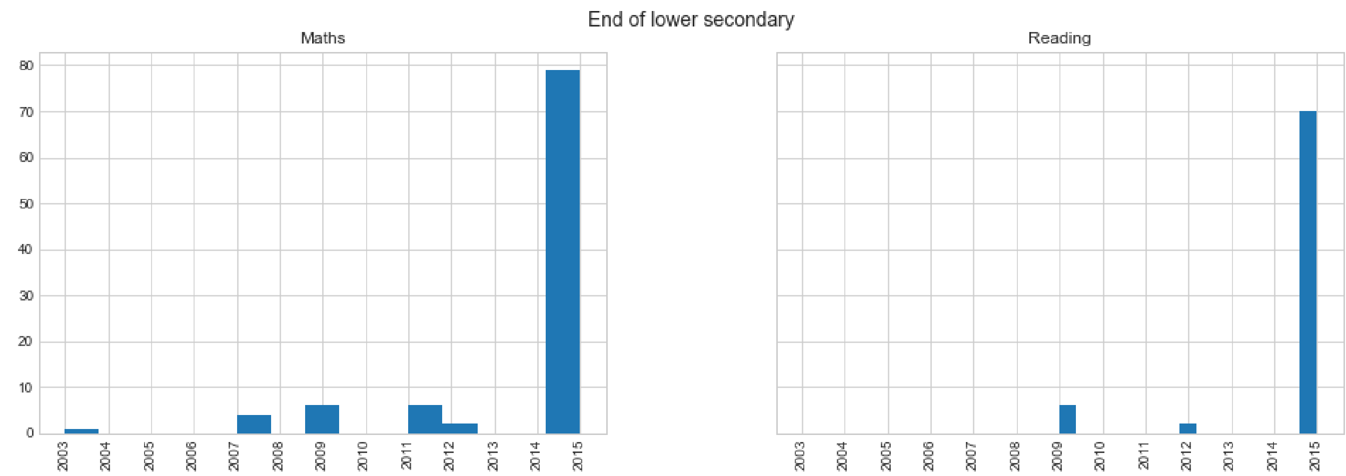
We can display the distribution of the data over time:

In [44]:
```
for edu_level in edu_levels:

        chart_data = prof_vs_gdp_data.loc[
            prof_vs_gdp_data['education_level'] == edu_level
        ].reset_index(level=1)
```

```
chart_data['seriesname'] = chart_data['seriescode'].map(series_labels)

axs = chart_data.hist(
    column='Year',
    by='seriesname',
    sharex=True,
    sharey=True,
    figsize=(16, 5),
    layout=(1, 2),
    bins=15,
)
for i in range(len(axs)):
    plt.suptitle(edu_level_labels[edu_level], fontsize=14)
    axs[i].xaxis.set_major_locator(plt.MultipleLocator(1))
```



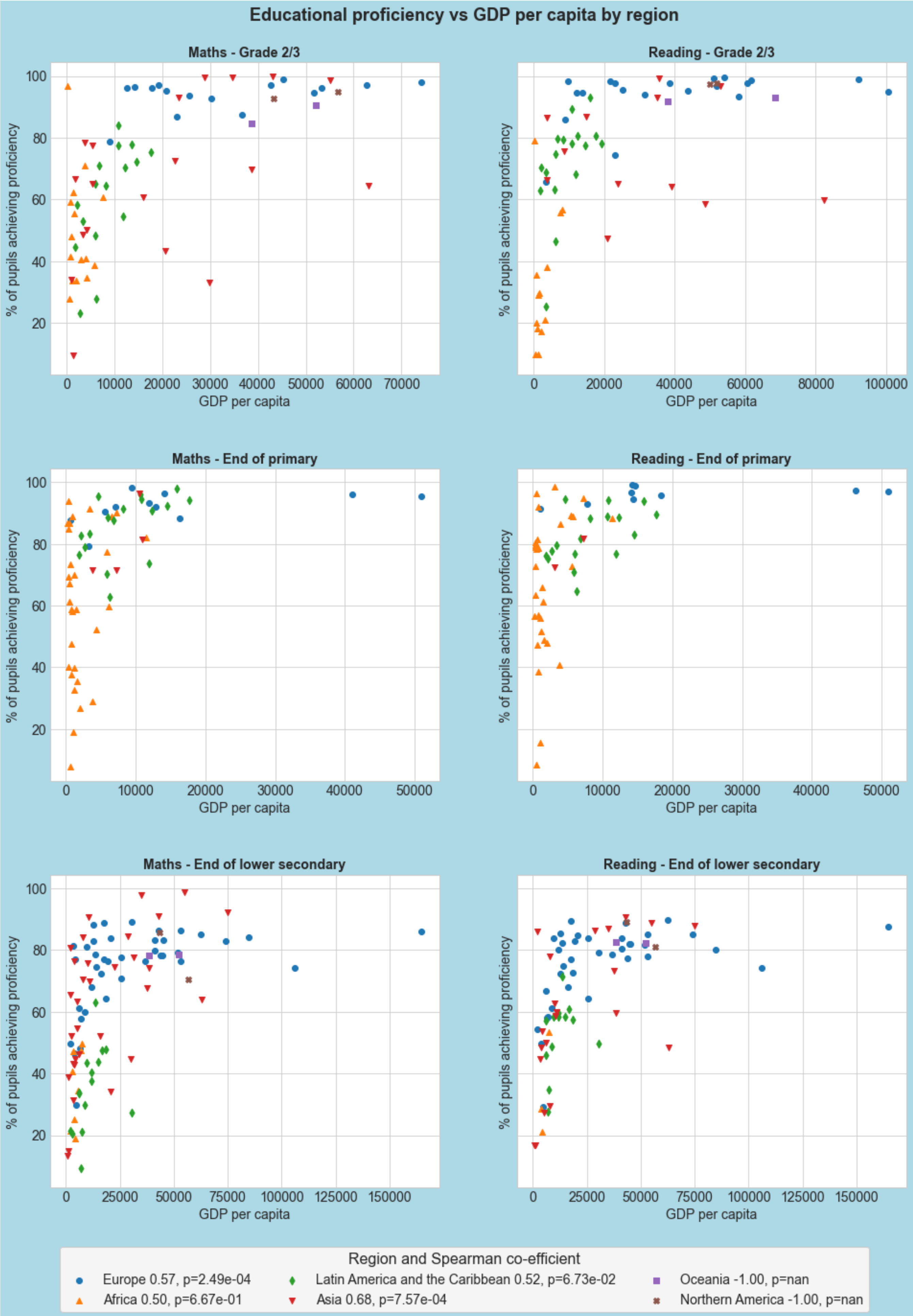




Now let's also show by region, this time splitting between reading and maths, and without indicating the year by size and transparency.

In [45]:
```python
region_markers = ["o", "^", "d", "v", "s", "X"]

# Create a 3 row by 2 col layout of plots
fig, axs = plt.subplots(3, 2, figsize=(16, 20), sharey=True)

plt_col = 0
for series_code in chart_series_codes:

    plt_row = 0

    for edu_level in edu_levels:
        marker_index = 0
        axs[plt_row, plt_col].set_xlabel('GDP per capita', fontsize=14)
        axs[plt_row, plt_col].set_ylabel('% of pupils achieving proficiency', fontsize=14)
        axs[plt_row, plt_col].tick_params(axis='both', labelsize=14)

        for region in prof_vs_gdp_data['Region Name'].unique():

            chart_data = prof_vs_gdp_data.loc[
                (prof_vs_gdp_data['seriescode'] == series_code) &
                (prof_vs_gdp_data['education_level'] == edu_level) &
                (prof_vs_gdp_data['Region Name'] == region)
            ]

            label = region
            spearman = chart_data.loc[:, ['GDP per capita', 'value']].corr('spearman').at['GDP
            if not np.isnan(spearman):
                pval = chart_data.loc[:, ['GDP per capita', 'value']].corr(spearmanr_pval).at[
                label = f"{region} {spearman:.2f}, p={pval:.2e}"

            axs[plt_row, plt_col].scatter(
                chart_data['GDP per capita'],
                chart_data['value'],
                marker = region_markers[marker_index],
                label = label
            )

            marker_index += 1

        axs[plt_row, plt_col].set_title(
            f"{series_labels[series_code]} - {edu_level_labels[edu_level]}",
            fontdict = {'fontsize' : 14, 'fontweight' : 'bold'}
        )
        plt_row += 1

    plt_col += 1

# Convert all legend labels into a dictionary so as to remove duplicates
handles, labels = plt.gca().get_legend_handles_labels()
by_label = dict(zip(labels, handles))
fig.legend(
    by_label.values(),
    by_label.keys(),
    loc='lower center',
    bbox_to_anchor=(0.5, 0.03),
    bbox_transform=fig.transFigure,
    ncol=ceil(len(labels) / 2),
    frameon=True, facecolor='whitesmoke', framealpha=1, fontsize=14,
    title="Region and Spearman co-efficient",
    title_fontsize=16
)

fig.set_facecolor('lightblue')
```

```
fig.suptitle('Educational proficiency vs GDP per capita by region', fontsize=18, fontweight='b
plt.subplots_adjust(hspace=0.3, top=0.94)
plt.show()

# Uncomment for a copy to display in results
# fig.savefig(fname='images/result2.png', bbox_inches='tight')
```


Educational proficiency vs GDP per capita by region
Maths - Grade 2/3
Reading - Grade 2/3
Maths - End of primary
Reading - End of primary
Maths - End of lower secondary
Reading - End of lower secondary
% of pupils achieving proficiency
GDP per capita
Region and Spearman co-efficient
Europe 0.57, p=2.49e-04
Africa 0.50, p=6.67e-01
Latin America and the Caribbean 0.52, p=6.73e-02
Asia 0.68, p=7.57e-04
Oceania -1.00, p=nan
Northern America -1.00, p=nan