Visualise with Plotly

Explore visualisation types

Take a look at this website, as it offers a comprehensive overview of various types of graphs that one can create while analysing data.
It provides in-depth explanations for each representation, highlighting common interpretation errors to avoid.

Now, suggest a relevant type of graph for each of the following situations:

a) We wish to observe the distribution of a numerical variable to know its extreme values.

> **Histogram**

b) We wish to represent the evolution of the composition of a financial portfolio over time. The portfolio is the sum of several assets.

> **Line plot**

c) We wish to observe the relationship between two numerical variables,aiming to determine whether a correlation exists or not.

Would we use the same graph if we were to investigate the correlation between several numerical variables ?

> **Scatterplot, correlogram for many variables**

Visualisation with Plotly

Plotly Graph Objects

Before delving into Plotly Express practice, let's take a moment to familiarize ourselves with graphical objects.
These elements enable you to enhance basic graphics by incorporating visual customizations.

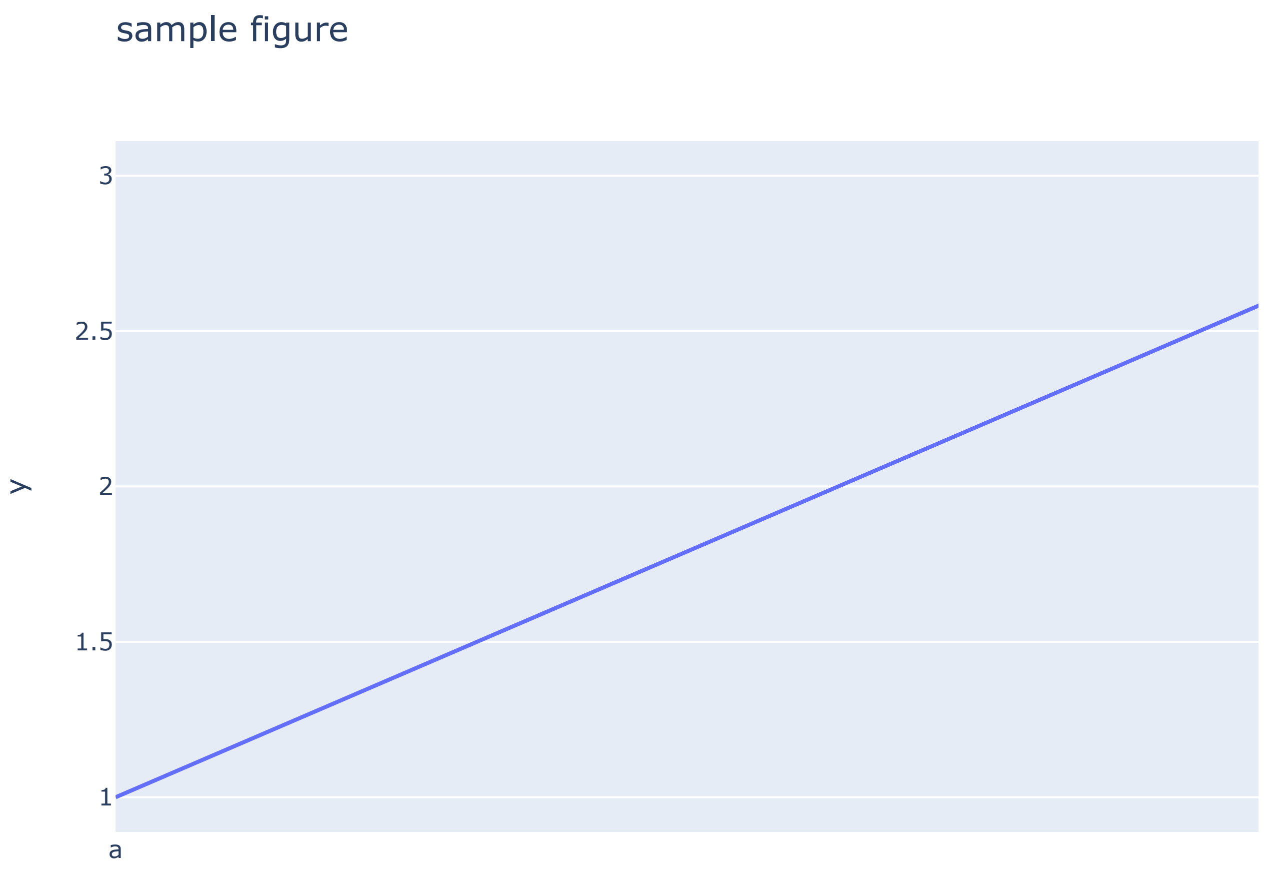
Import the following librairies 👇

```
import numpy as np
from collections import Counter

import pandas as pd

import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

Create a Figure object called `fig` and display it.

```
import plotly.express as px

fig = px.line(x=["a","b","c"], y=[1,3,2], title="sample figure")
print(fig)
fig.show()
```

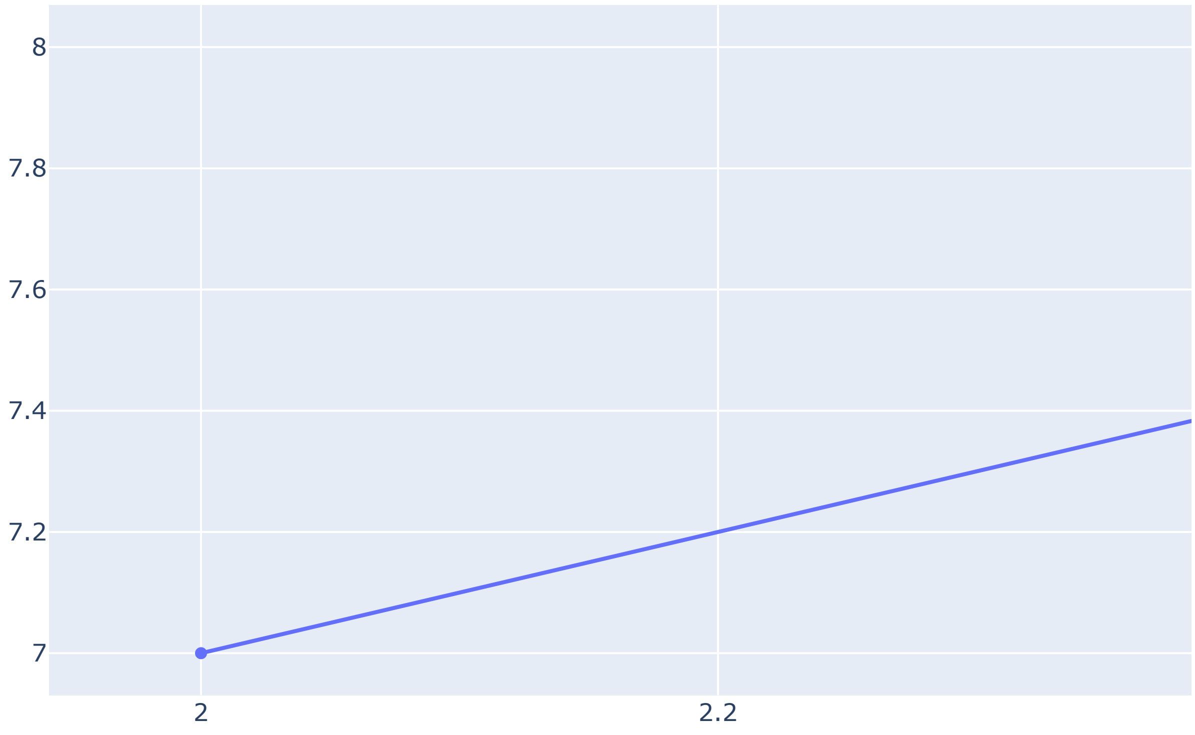
```
Figure({
    'data': [{'hovertemplate': 'x=%{x}<br>y=%{y}<extra></extra>',
              'legendgroup': '',
              'line': {'color': '#636efa', 'dash': 'solid'},
              'marker': {'symbol': 'circle'},
              'mode': 'lines',
              'name': '',
              'orientation': 'v',
              'showlegend': False,
              'type': 'scatter',
              'x': array(['a', 'b', 'c'], dtype=object),
              'xaxis': 'x',
              'y': array([1, 3, 2]),
              'yaxis': 'y'}],
    'layout': {'legend': {'tracegroupgap': 0},
               'template': '...',
               'title': {'text': 'sample figure'},
               'xaxis': {'anchor': 'y', 'domain': [0.0, 1.0], 'title': {'text
               'yaxis': {'anchor': 'x', 'domain': [0.0, 1.0], 'title': {'text
})
```



Now create a graphical object of type Scatter. This object should contain a single line between two points **A(2,7)** and **B(3,8)**.
Store this object in a variable named `first_line_object`.

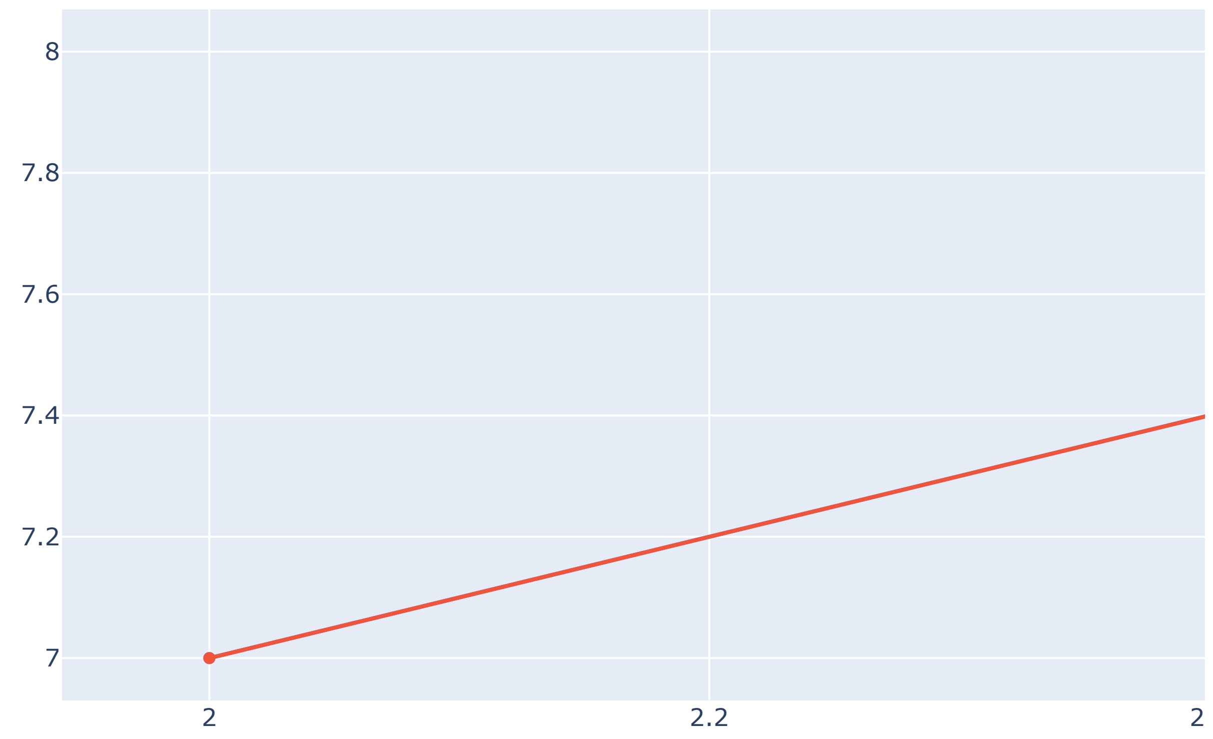
▶ Help 📚

```
first_line_object = go.Scatter(x=[2,3],y=[7,8])
fig = go.Figure(data=[first_line_object])
fig.show()
```



Add this variable to your figure using the `.add_trace()` method and then display the result using the `.show()` method.

```
# first_line_object = go.Scatter(x=[2,7],y=[3,8])
fig.add_trace(first_line_object)
fig.show()
```



Plotly Express

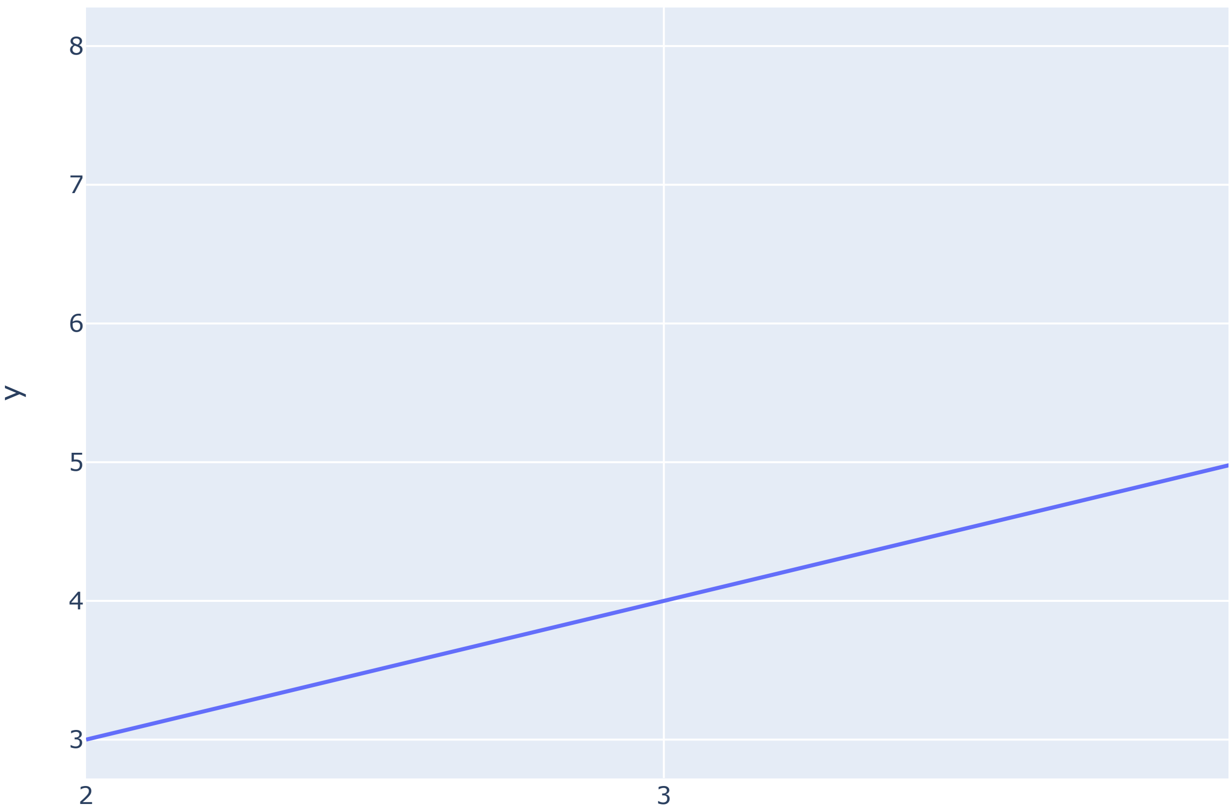
Graphical objects enables the creation of very complex graphs by incorporating multiple elements, but the syntax is quite heavy.

This is where the **Plotly Express** library comes in handy, as it allows to build nice graphs with a much lighter syntax. This also combines really well with DataFrames. 🔥

It is recommended to always start building graphs with Plotly Express, and then consider using Plotly Graph Objects if additional customization (beyond what Plotly Express provides) is needed.

Recreate the same graph as before using the `line` function of **Plotly Express**.

```
# import plotly.express as px already in memory
px.line(x=[2,7], y=[3,8])
```



Explore countries data with Plotly

Plotly Express offers access to various datasets, including the well-known `gapminder` dataset.

Run the following cell to load the data.

```
df = px.data.gapminder()
df
```

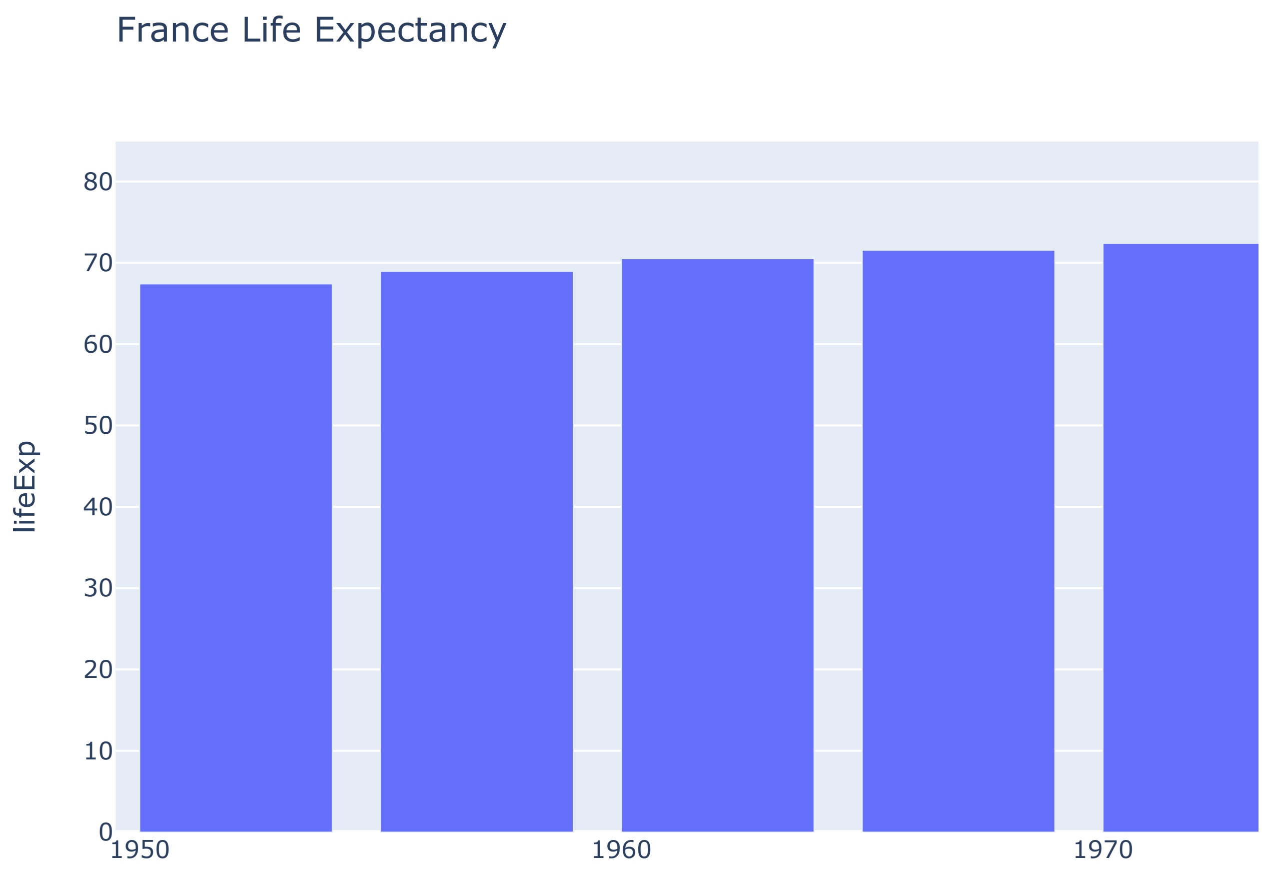
	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_nu
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	
...	...	...	...	...	...	...	...	
1699	Zimbabwe	Africa	1987	62.351	9216418	706.157306	ZWE	7
1700	Zimbabwe	Africa	1992	60.377	10704340	693.420786	ZWE	7
1701	Zimbabwe	Africa	1997	46.809	11404948	792.449960	ZWE	7
1702	Zimbabwe	Africa	2002	39.989	11926563	672.038623	ZWE	7
1703	Zimbabwe	Africa	2007	43.487	12311143	469.709298	ZWE	7

1704 rows × 8 columns

Simple plots

Display the evolution of the French Life Expectancy over the years in a **bar chart**.

```
data_france = df[df['country'] == 'France']
fr = px.bar(data_france, x="year",y='lifeExp',title='France Life Expectancy')
fr.show()
```



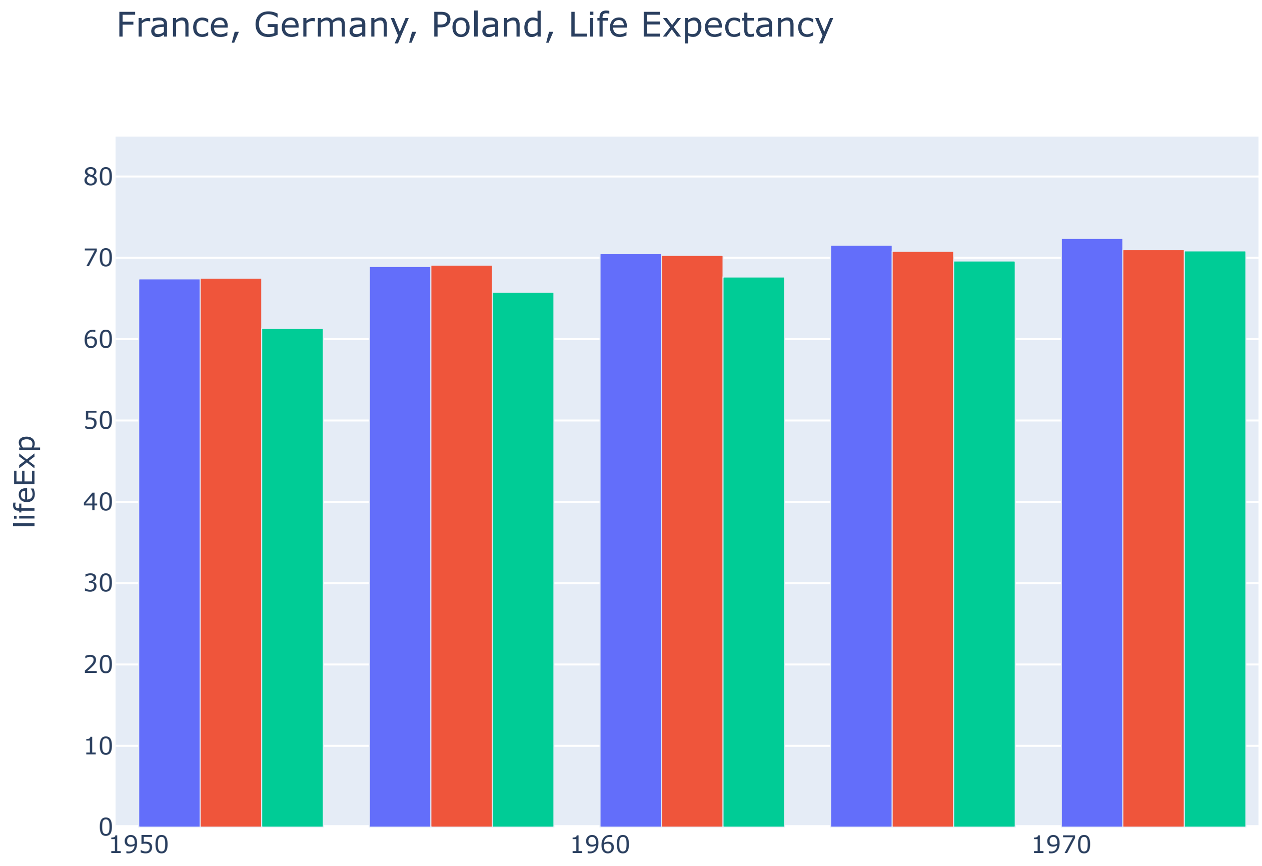
Let's repeat the process, but this time we will display the evolution of Life Expectancy for France, Germany and Poland.

Additional requirements :

- Display three bars for each year, each representing a different country and using distinct colors.
- Make sure that the three bars are separate and not stacked.

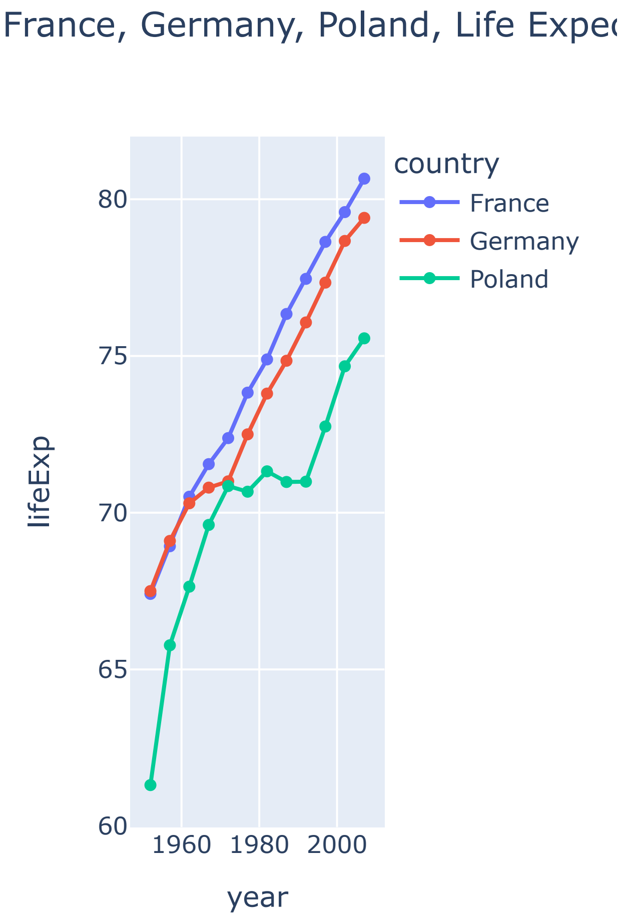
▶ Hint 🔍

```
FGP = df[df['country'].isin(['France','Germany','Poland'])]

fgp_bar = px.bar(data_frame=FGP, x='year', y='lifeExp', color='country', barmode=

fgp_bar.show()
```



Bar charts might not the best suited type of representation.
Modify the previous code to represent the evolution of Life Expectancy with a **line chart**. Does it look better ?

```
# FGP = df[df['country'].isin(['France','Germany','Poland'])]

fgp_line = px.line(FGP, x='year', y='lifeExp', color='country', markers=True, tit

fgp_line.show()
```



Complex plots

We can use additional parameters like sizes and colors to add more information to our graphs.

For the year 2007, visualize the relationship between the GDP per capita and the life expectancy. Think about the best type of graph to visualise correlations.

Additional requirements:

- Countries should be representated with different colors based on their continent
- Each country should have a different size proportional to its population

Note: the argument `size_max` can be set to 60 to increase the size of the data points while keeping the same proportions.

```
import math

year_2007 = df[df['year'] == 2007]
year_2007 = year_2007.sort_values(['continent', 'country'])

hover_text = []
bubble_size = []

for index, row in year_2007.iterrows():
    hover_text.append(('Country: {country}<br>'+
                      'Life Expectancy: {lifeExp}<br>'+
                      'GDP per capita: {gdp}<br>'+
                      'Population: {pop}<br>').format(country=row['country'],
                                            lifeExp=row['lifeExp'],
                                            gdp=row['gdpPercap'],
                                            pop=row['pop']))
    bubble_size.append(math.sqrt(row['pop']))

year_2007['text'] = hover_text
year_2007['size'] = bubble_size
sizeref = 2.*max(year_2007['size'])/(100**2)

#dict for continent names
continent_names = ['Africa', 'Americas', 'Asia', 'Europe', 'Oceania']
continent_data = {continent:year_2007.query("continent == '%s'" %continent)
                              for continent in continent_names}

for continent_name, continent in continent_data.items():
    fig.add_trace(go.Scatter(
        x=continent['gdpPercap'], y=continent['lifeExp'],
        name=continent_name, text=continent['text'],
        marker_size=continent['size'],
        ))

# Tune marker appearance and layout
fig.update_traces(mode='markers', marker=dict(sizemode='area',
                                              sizeref=sizeref, line_width=2))

fig.update_layout(
    title='Life Expectancy v. Per Capita GDP, 2007',
    xaxis=dict(
        title='GDP per capita (2000 dollars)',
        gridcolor='white',
        type='log',
        gridwidth=2,
    ),
    yaxis=dict(
        title='Life Expectancy (years)',
        gridcolor='white',
        gridwidth=2,
    ),
    paper_bgcolor='rgb(243, 243, 243)',
    plot_bgcolor='rgb(243, 243, 243)',
)
fig.show()
```

Life Expectancy v. Per Capita GDP, 2