# Seaborn for Visualizations

Seaborn builds off the matplotlib package and is easier to learn if you have previously used matplotlib.

- **Datacamp Python Tutorial for Begineers:** Best short tutorial for getting started with Seaborn. https://www.datacamp.com/community/tutorials/seaborn-python-tutorial
- **Datacampe One-Page Cheatsheet:** Best for quickly referencing possible arguments https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Seaborn_Cheat_Sheet.p
- **Seaborn Built in Datasets:** Great for learning seaborn commands. https://github.com/mwaskom/seaborn-data
- **Seaborn Documentaion:** https://seaborn.pydata.org/

The Nations dataset was a dataset provided in QMB 6930 Intro to Python. The code below builds on the Seaborn fundamentals from that class, Code Academy's Intro to Data Vistualization in Python and DataCamp's Data Visualization courses along with Seaborn's documentation.

## Document Setup

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
import numpy as np
```

In [2]:
```
# current version of seaborn generates a bunch of warnings that we'll ignore
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
#Document Defaults

#Sets default font size
sns.set(rc={"font.size":12,"axes.labelsize":8})
```

In [4]:
```
#Sets the color for all your graphs
sns.set_palette("tab10")
```

In [5]:
```
#Sets the background style to white
sns.set(style="white")
```

In [6]:
```
sns.set(rc={'figure.figsize':(15,5)})
```

### Load Built in Dataset

Seaborn has multiple built-in datasets for documentation purposes. Including cars which we will use for numerous examples below.

In [7]:
```
cars = sns.load_dataset("car_crashes")
print(cars.head())
```

```
   total  speeding  alcohol  not_distracted  no_previous  ins_premium  \
0   18.8     7.332    5.640          18.048       15.040       784.55
1   18.1     7.421    4.525          16.290       17.014      1053.48
2   18.6     6.510    5.208          15.624       17.856       899.47
3   22.4     4.032    5.824          21.056       21.280       827.34
4   12.0     4.200    3.360          10.920       10.680       878.41

   ins_losses abbrev
0      145.08     AL
1      133.93     AK
2      110.35     AZ
3      142.39     AR
4      165.63     CA
```

## Load Nations Data From CSV File

In [8]:
```
#Get working directory
os.getcwd()
```

Out[8]: '/Users/amandapiter/Documents/Projects/Github?'

In [9]:
```
#Set Working Directory
os.chdir("/Users/amandapiter/Documents/Spring 2021 Python/PythonDataWranglingVis
```

In [10]:
```
#Load the Nations File
nations = pd.read_csv("nations.csv")
print(nations.head())
```

```
  iso2c iso3c  country  year  gdp_percap  life_expect  population  birth_rate  \
0    AD   AND  Andorra  1996         NaN          NaN     64291.0        10.9
1    AD   AND  Andorra  1994         NaN          NaN     62707.0        10.9
2    AD   AND  Andorra  2003         NaN          NaN     74783.0        10.3
3    AD   AND  Andorra  1990         NaN          NaN     54511.0        11.9
4    AD   AND  Andorra  2009         NaN          NaN     85474.0         9.9

   neonat_mortal_rate                 region       income
0                 2.8  Europe & Central Asia  High income
1                 3.2  Europe & Central Asia  High income
2                 2.0  Europe & Central Asia  High income
3                 4.3  Europe & Central Asia  High income
4                 1.7  Europe & Central Asia  High income
```

## Subset Nations Data By Year

In [11]:
```
# Let's just pick year 2008
nations2008 = nations.loc[nations['year'] == 2008]
print(nations2008.head())
```

```
    iso2c iso3c              country  year    gdp_percap  life_expect  \
12     AD   AND              Andorra  2008           NaN          NaN
```

```
40     AE   ARE  United Arab Emirates  2008  69124.396211    76.307756
71     AF   AFG           Afghanistan  2008   1283.040986    58.225024
96     AG   ATG   Antigua and Barbuda  2008  24723.111892    74.979171
107    AL   ALB               Albania  2008   8769.094336    76.652073

     population  birth_rate  neonat_mortal_rate                     region  \
12      85616.0      10.400                 1.8       Europe & Central Asia
40    6900142.0      12.423                 4.6  Middle East & North Africa
71   26528741.0      41.560                39.8                  South Asia
96      85350.0      17.514                 6.6   Latin America & Caribbean
107   2947314.0      11.561                 8.2       Europe & Central Asia

                  income
12           High income
40           High income
71            Low income
96   Upper middle income
107  Upper middle income
```

In [12]:

```
# Let's see how many countries we have of each region
print(nations['region'].value_counts())
```

```
Europe & Central Asia         1400
Sub-Saharan Africa            1175
Latin America & Caribbean     1025
East Asia & Pacific            875
Middle East & North Africa     525
South Asia                     200
North America                   75
Name: region, dtype: int64
```

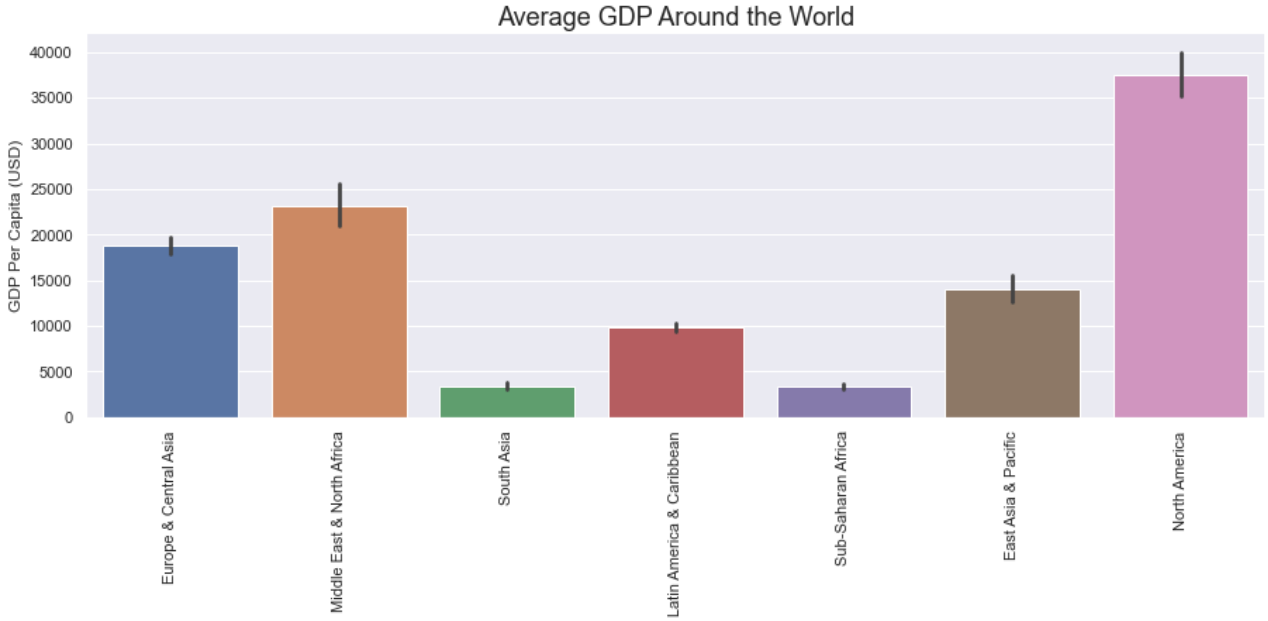
# Barplots

## Default Barplots with Error Bars

#Explination from Code Academy, Learn Data Visualization with Python Course Error bars are the small lines that extend above and below the top of each bar. Errors bars visually indicate the range of values that might be expected for that bar. By default, Seaborn uses something called a bootstrapped confidence interval. Roughly speaking, this interval means that “based on this data, 95% of similar situations would have an outcome within this range”. If you’re calculating a mean and would prefer to use standard deviation for your error bars, you can pass in the keyword argument ci="sd" to sns.barplot() which will represent one standard deviation. It would look like this:

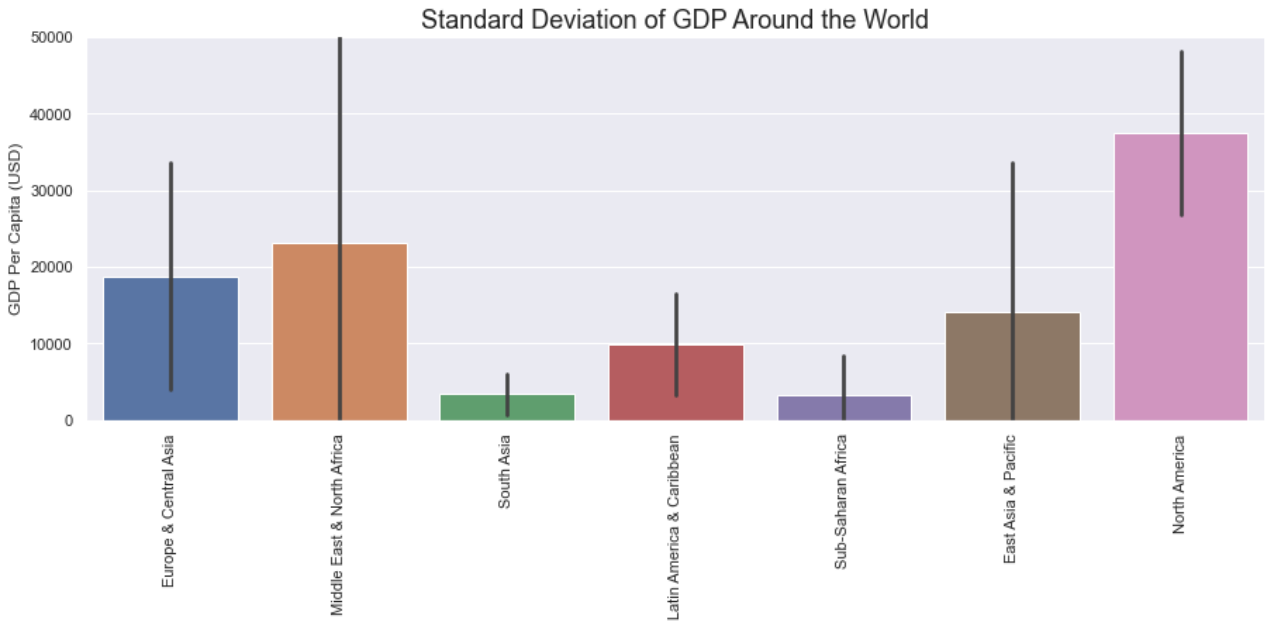
In [13]:

```
#Default Barplots in Seaborn, with Error Bar
g = sns.barplot(
    data= nations,
    x= "region",
    y= "gdp_percap")

plt.title("Average GDP Around the World", size=18)
plt.ylabel("GDP Per Capita (USD)")
plt.xlabel("")
plt.xticks(rotation=90) #Specified 90 because at 45 and they were not under corr
sns.despine()
```




## Barplot with Standard Deviation Error Bar

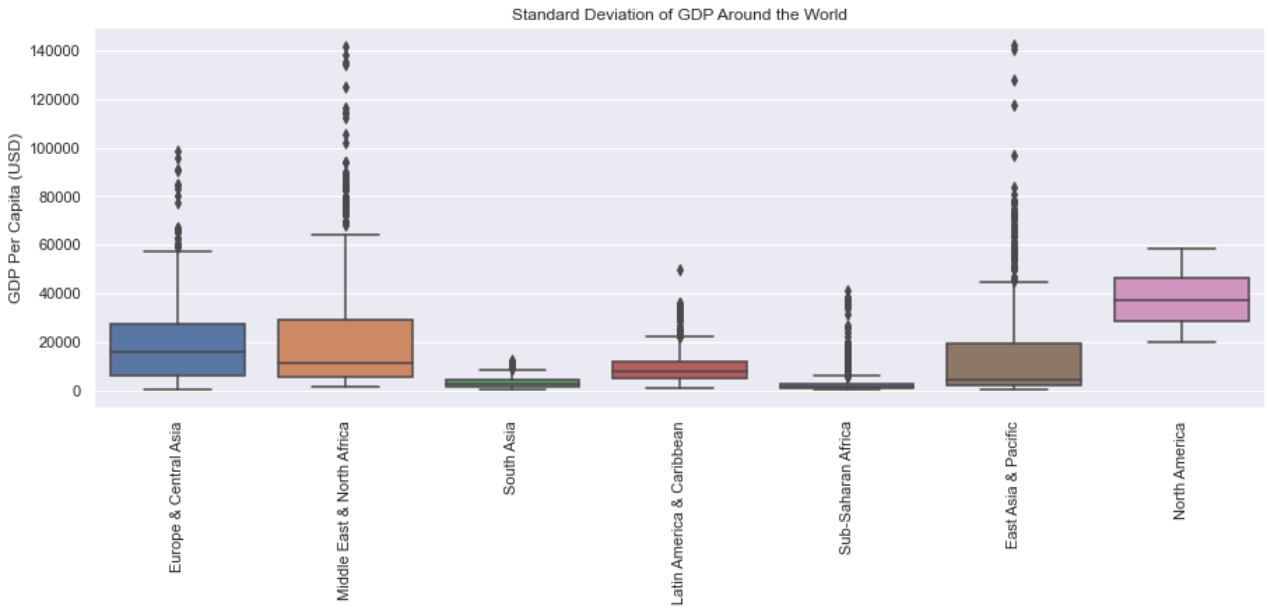
In [14]:
```
#Barplot with Standard Devation Error Bar
g = sns.barplot(
    data= nations,
    x= "region",
    y= "gdp_percap",
    ci="sd")

plt.title("Standard Deviation of GDP Around the World", size=18)
plt.ylabel("GDP Per Capita (USD)")
plt.xlabel("")
plt.xticks(rotation=90) #Specified 90 because at 45 and they were not under corr
g.set(ylim=(0,50000))
sns.despine()
```




## Boxplots

In [15]:
```
#changes the background for all future plots in your code
sns.set(style="darkgrid")
```

In [16]:
```
#Adjustments made from QMB693 Example
#We can look at an individual feature in Seaborn through a boxplot
sns.boxplot(x="region", y="gdp_percap", data=nations)

plt.title("Standard Deviation of GDP Around the World")
plt.ylabel("GDP Per Capita (USD)")
plt.xlabel("")
plt.xticks(rotation=90) #Specified 90 because at 45 and they were not under corr
g.set(ylim=(0,50000))
sns.despine()
```

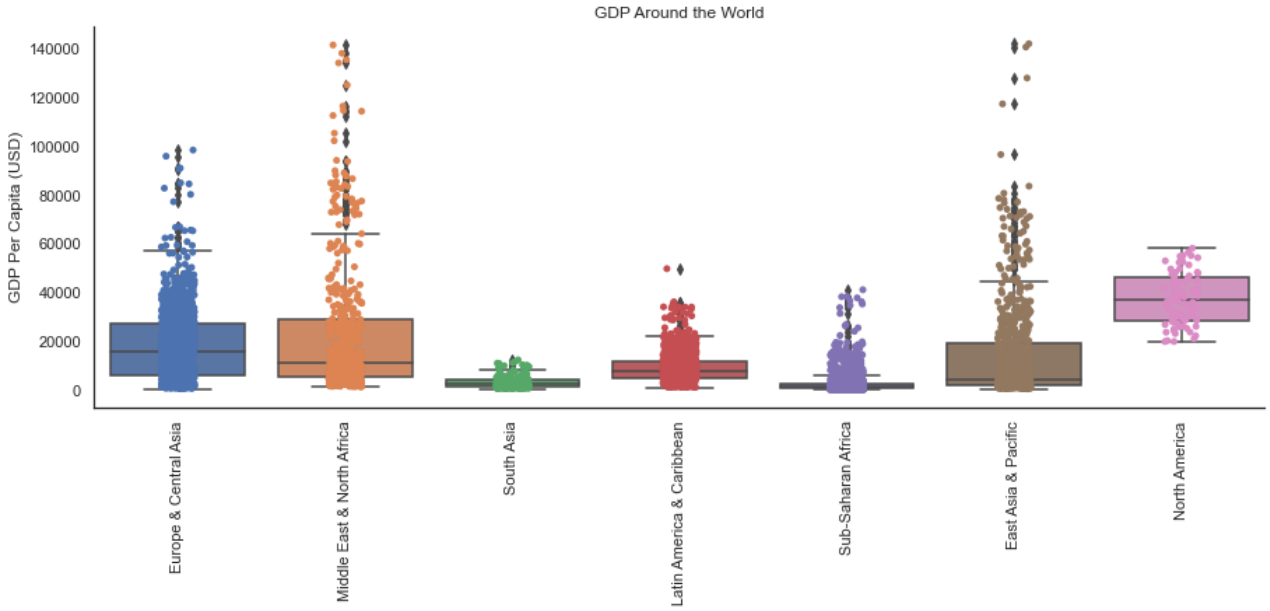



## Layered Boxplot with Striplot

In [17]:
```
#changes the background back to white which is my preference
sns.set(style="white")
```

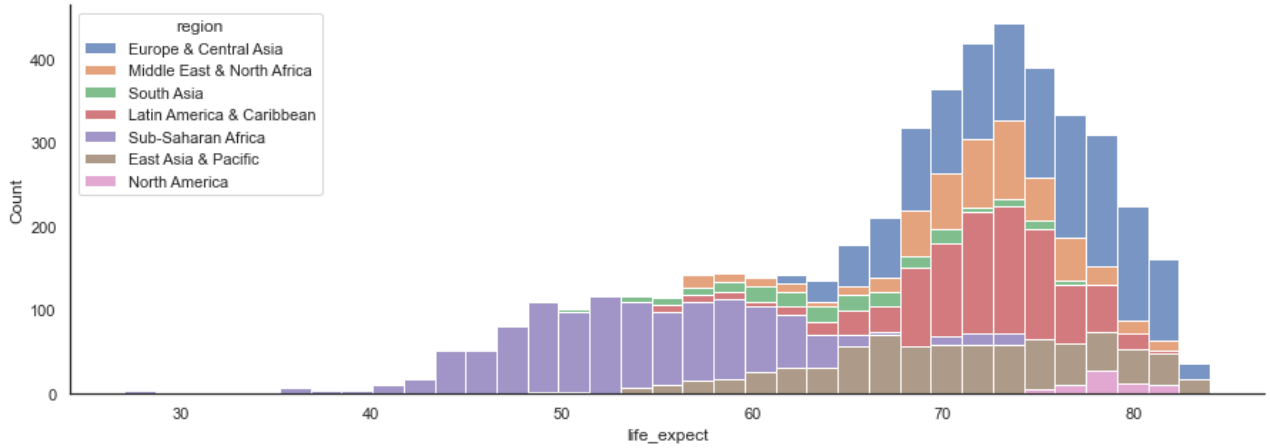
In [18]:
```
#Modifications from Example in QMB 6930
# One way we can extend this plot is adding a layer of individual points on top
# it through Seaborn's striplot
#
# We'll use jitter=True so that all the points don't fall in single vertical lin
# above the species
#
# Saving the resulting axes as ax each time causes the resulting plot to be show
# on top of the previous axes
ax = sns.boxplot(x="region", y="gdp_percap", data=nations)
ax = sns.stripplot(x="region", y="gdp_percap", data=nations, jitter=True, edgeco

plt.title("GDP Around the World")
plt.ylabel("GDP Per Capita (USD)")
plt.xlabel("")
plt.xticks(rotation=90) #Specified 90 because at 45 and they were not under corr
```

```
g.set(ylim=(0,50000))
sns.despine()
```

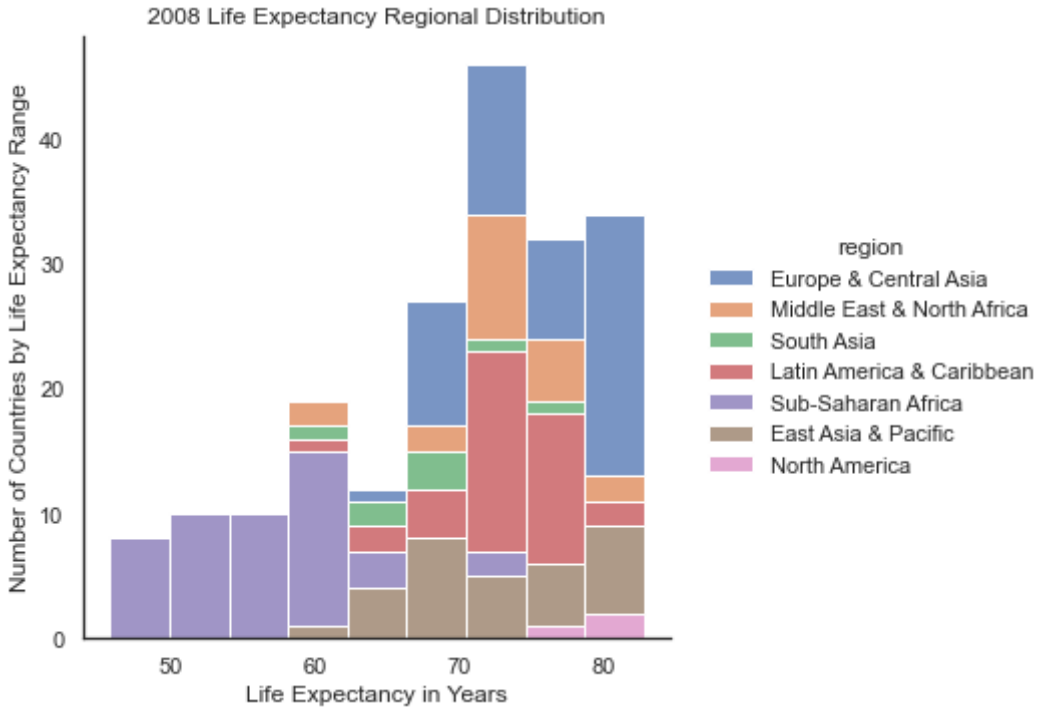



# Histograms

In [19]:
```
#Use argument multiple to create a stacked histogram.
sns.histplot(data=nations, x="life_expect", hue="region", multiple="stack")

sns.despine()
```

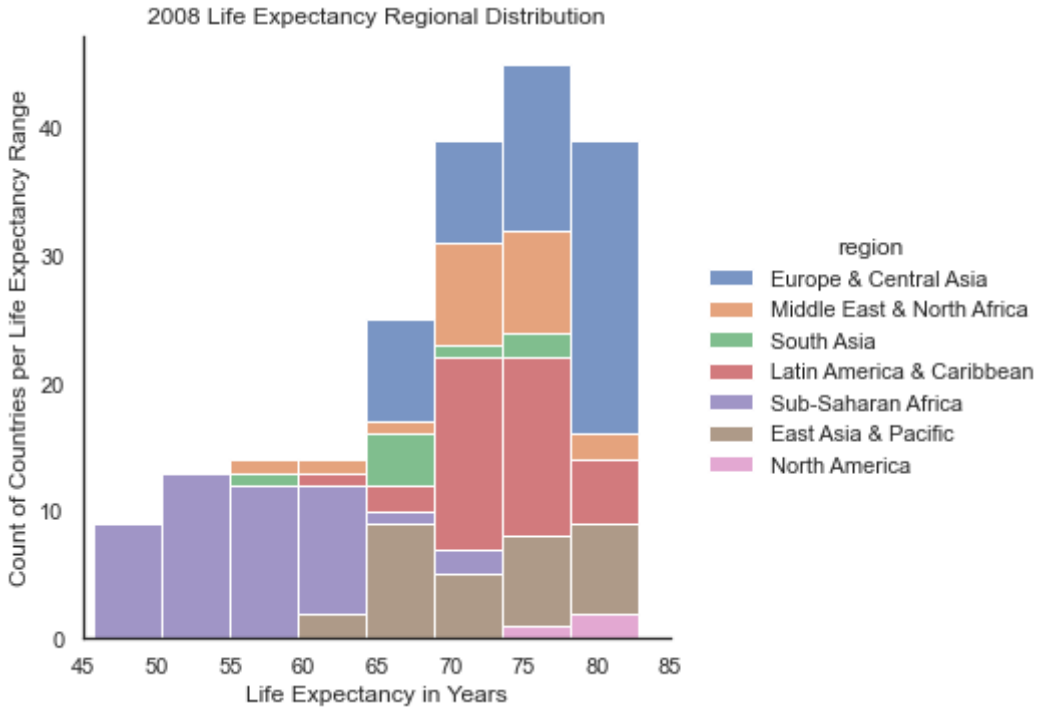



In [20]:
```
#Use displot to create a figure level plot, which places the legend outside the
plot = sns.displot(data=nations2008, x="life_expect", hue="region", multiple="st

plt.title("2008 Life Expectancy Regional Distribution")
plt.xlabel("Life Expectancy in Years")
plt.ylabel("Number of Countries by Life Expectancy Range")

sns.despine()
```



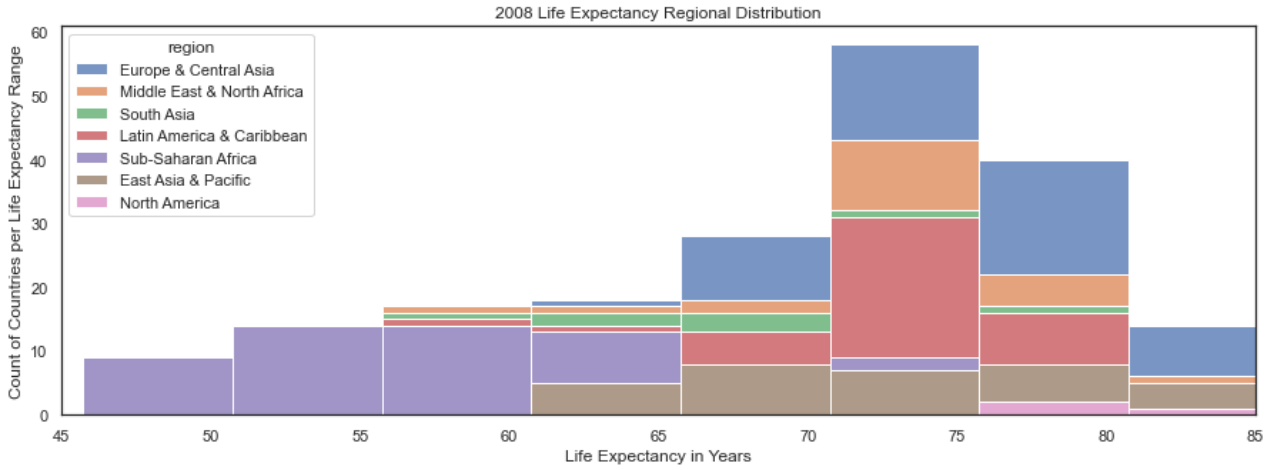

## Specify Bin Size

In [21]:
```
#Creates 8 equally sized bins. Since the last value in our dataset is before 85

sns.displot(data=nations2008, x="life_expect", hue="region", multiple="stack", b
plt.title("2008 Life Expectancy Regional Distribution")
plt.xlabel("Life Expectancy in Years")
plt.ylabel("Count of Countries per Life Expectancy Range")
plt.xlim(45,85)
plt.show()
```



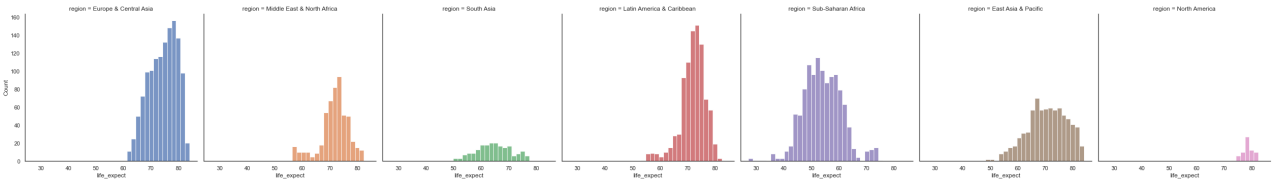

In [22]:

```
#To specify the exact binwidth use a sns.histplot

sns.histplot(data=nations2008, x="life_expect", hue="region", multiple="stack",
plt.title("2008 Life Expectancy Regional Distribution")
plt.xlabel("Life Expectancy in Years")
plt.ylabel("Count of Countries per Life Expectancy Range")
plt.xlim(45,85)
plt.show()
```

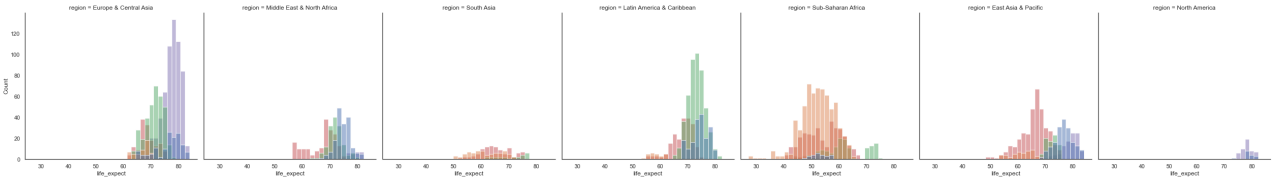



## Small Multiples

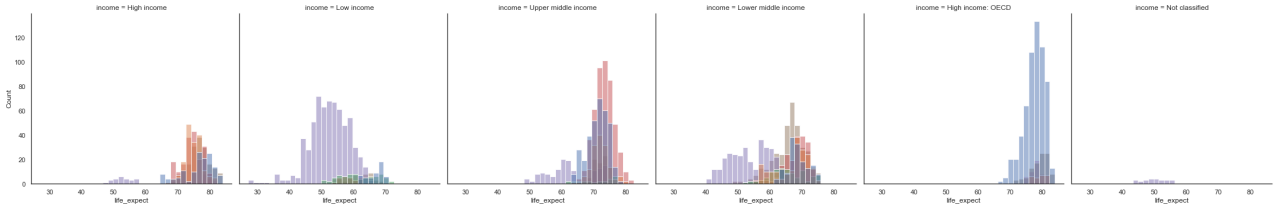
In [23]:
```
#Figure level plots also allow you to create small multiples using the col= argu
sns.displot(data=nations, x="life_expect", hue="region", col="region", legend=Fa

sns.despine()
```



In [24]:
```
#Seperated by Region, Colors by Income
sns.displot(data=nations, x="life_expect", hue="income", col="region", legend=Fa

sns.despine()
```

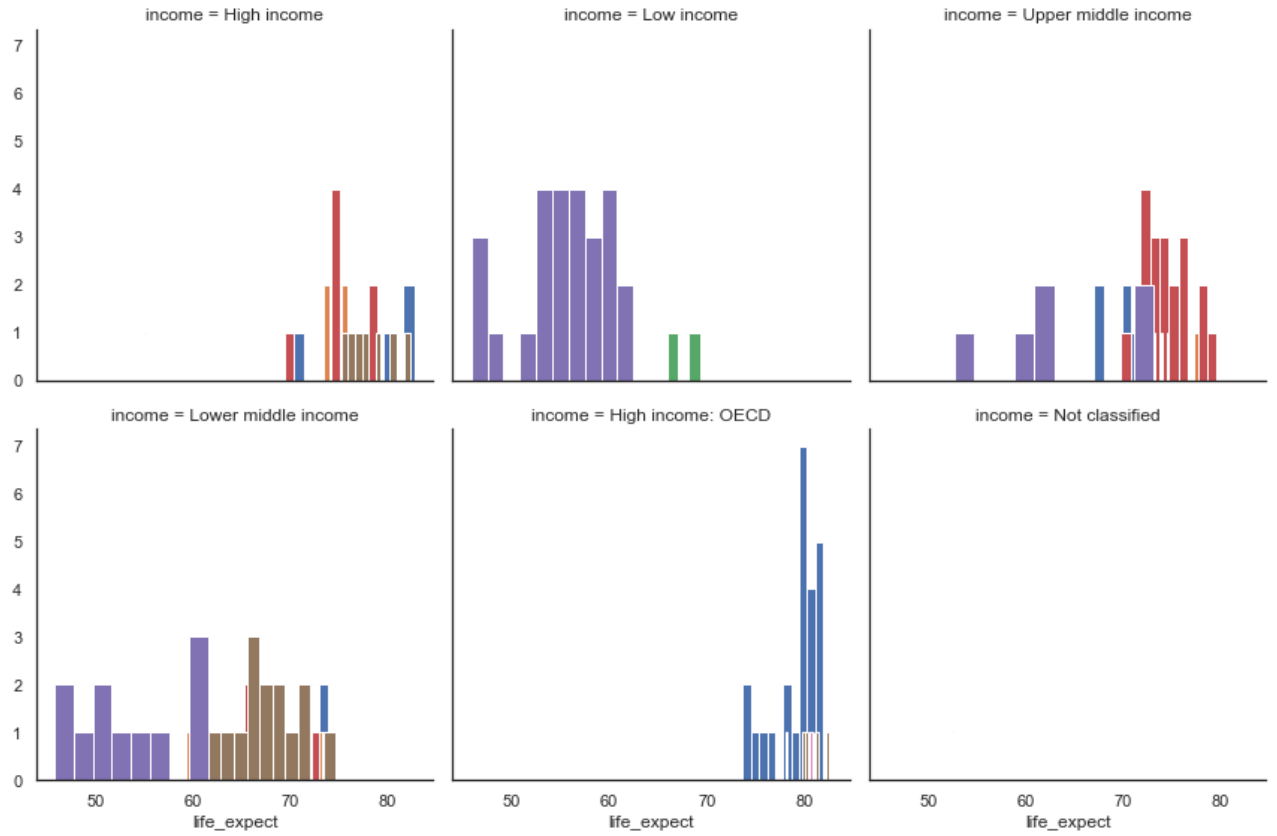


In [25]:
```
#Seperated by Icome, Colors by Region
sns.displot(data=nations, x="life_expect", hue="region", col="income", legend=Fa
sns.despine()

#Notice High Income Sub-Sahran African countries still have a lower life expecta
```




## Facetgrids

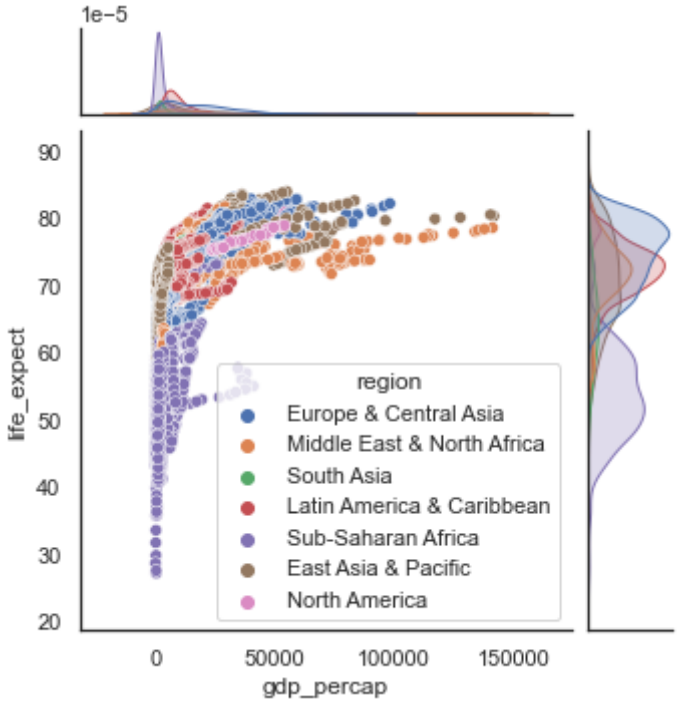
In [26]:
```
#Use facetgrids to specify the height and width of the column wrap

g = sns.FacetGrid(data=nations2008, col="income", col_wrap=3,height=4, hue='regi
g = (g.map(plt.hist, "life_expect"))
```



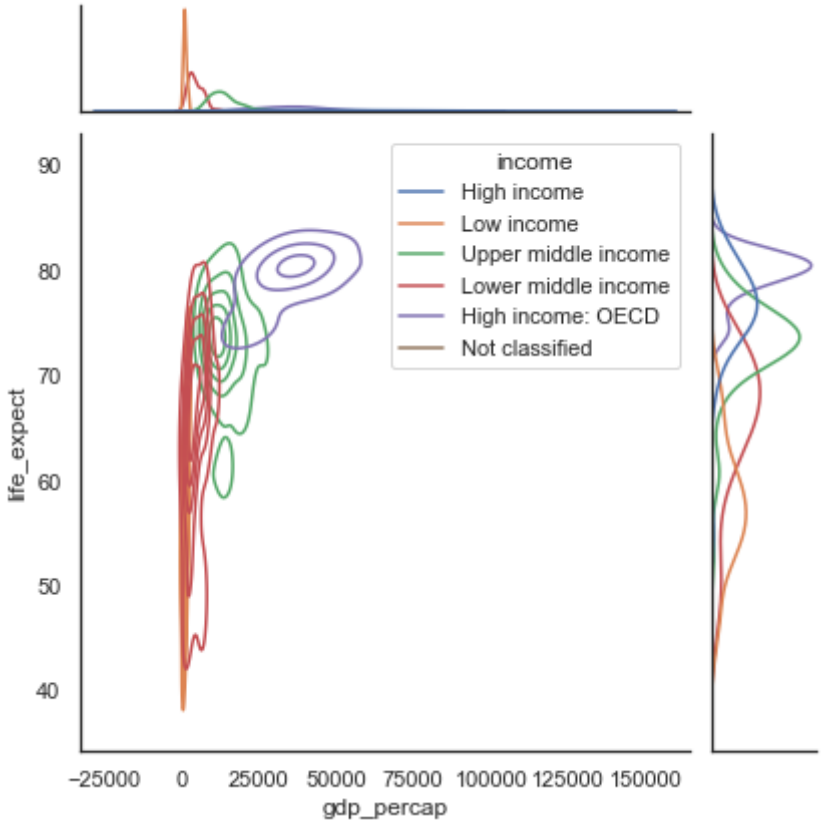

## Jointplots

We can also use the seaborn library to make a similar plot A seaborn jointplot shows bivariate scatterplots and univariate histograms in the same figure

In [27]:
```
# Example from QMB 6930
sns.jointplot(x="gdp_percap", y="life_expect", data=nations, hue='region', heigh

sns.despine()
```

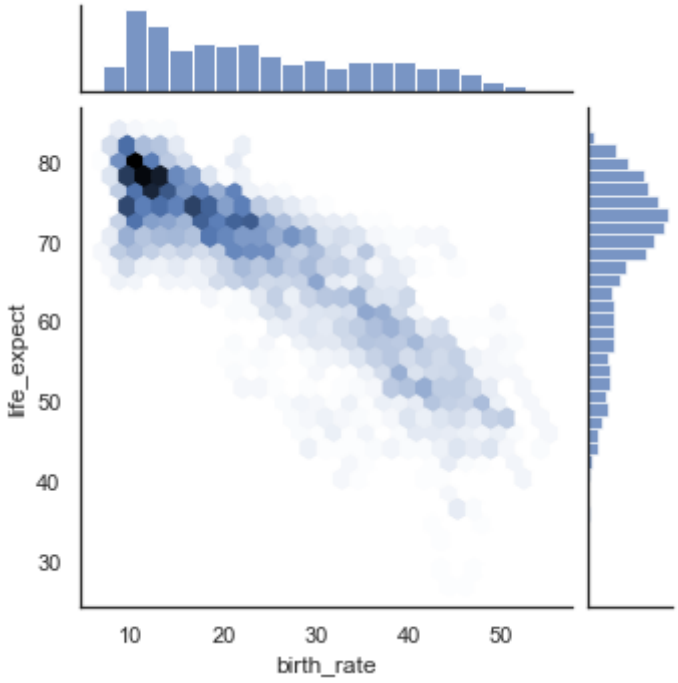



In [28]:
```
# Joint plots can also be made with kind='kde' or kind='hex'
plot = sns.jointplot(x="gdp_percap", y="life_expect", data=nations2008, hue='inc

sns.despine()
```




In [29]:
```
#hex plots look better with more disbursed datasets
sns.jointplot(x="birth_rate", y="life_expect", data=nations, height=5, kind='hex
```

```
sns.despine()
```




# KDE Plots

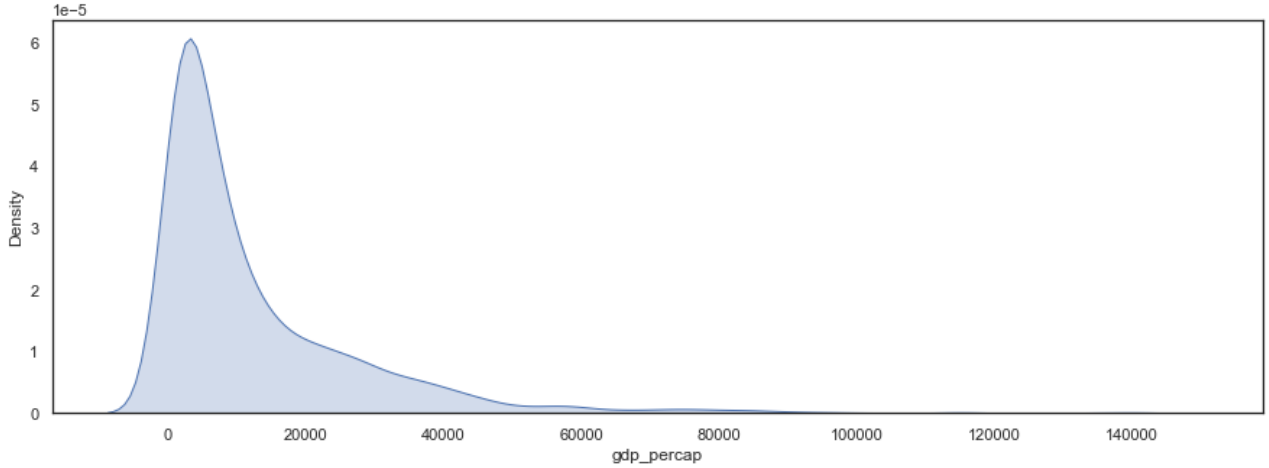
KDE plots show the distribution of the dataset values. KDE stands for Kernel Density Estimator. A KDE plot gives us the sense of a univariate as a curve. A univariate dataset only has one variable and is also referred to as being one-dimensional, as opposed to bivariate or two-dimensional datasets which have two variables.

*Summarized from Code Academy's, Learn Data Visualization with Python*

**A KDE plot takes the following arguments:**

- data - the univariate dataset being visualized, like a Pandas DataFrame, Python list, or NumPy array
- shade - a boolean that determines whether or not the space underneath the curve is shaded

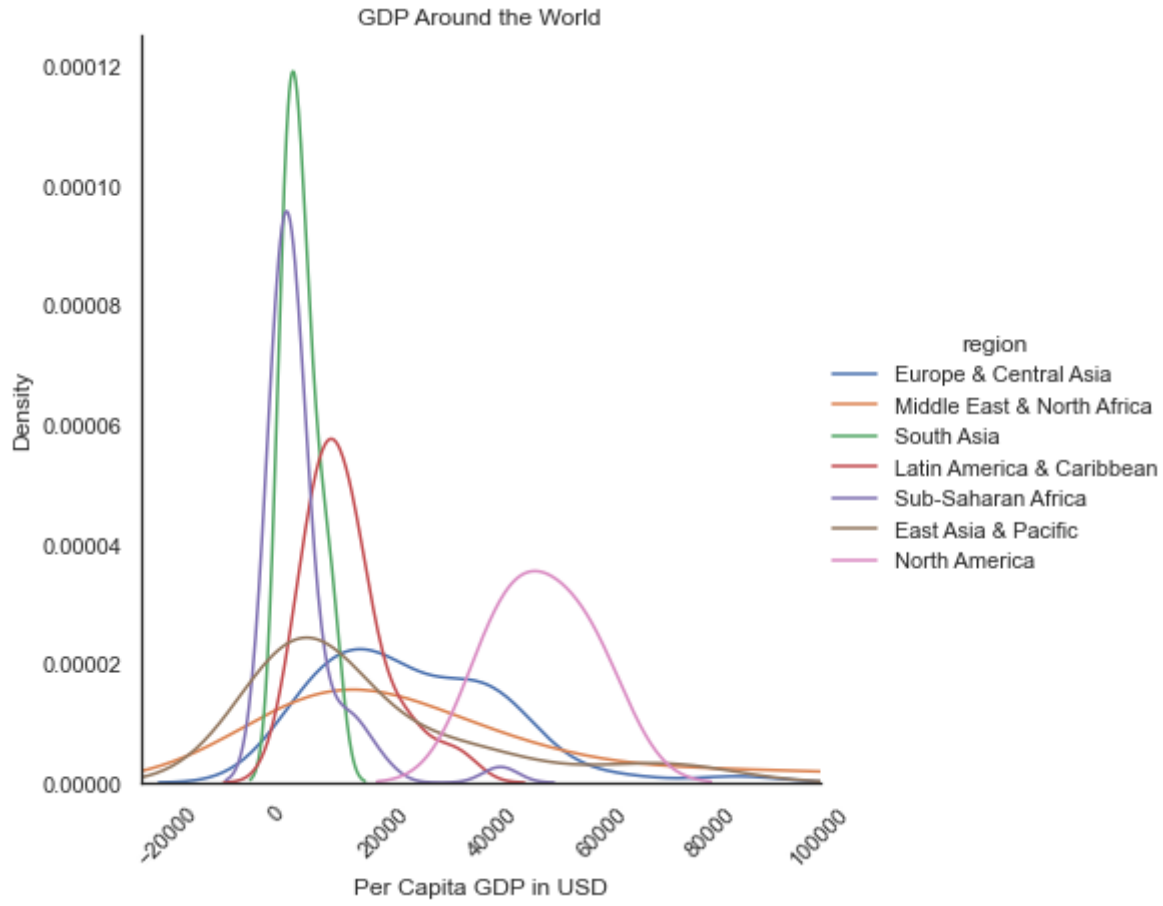
In [30]:
```
sns.kdeplot(data=nations, x="gdp_percap", shade=True, label="gdp_percap")

#sns.kdeplot(dataset2, shade=True, label="dataset2")
#sns.kdeplot(dataset3, shade=True, label="dataset3")
plt.show()

sns.despine() #does not get rid of the le-5
```




```
<Figure size 1080x360 with 0 Axes>
```

In [31]:

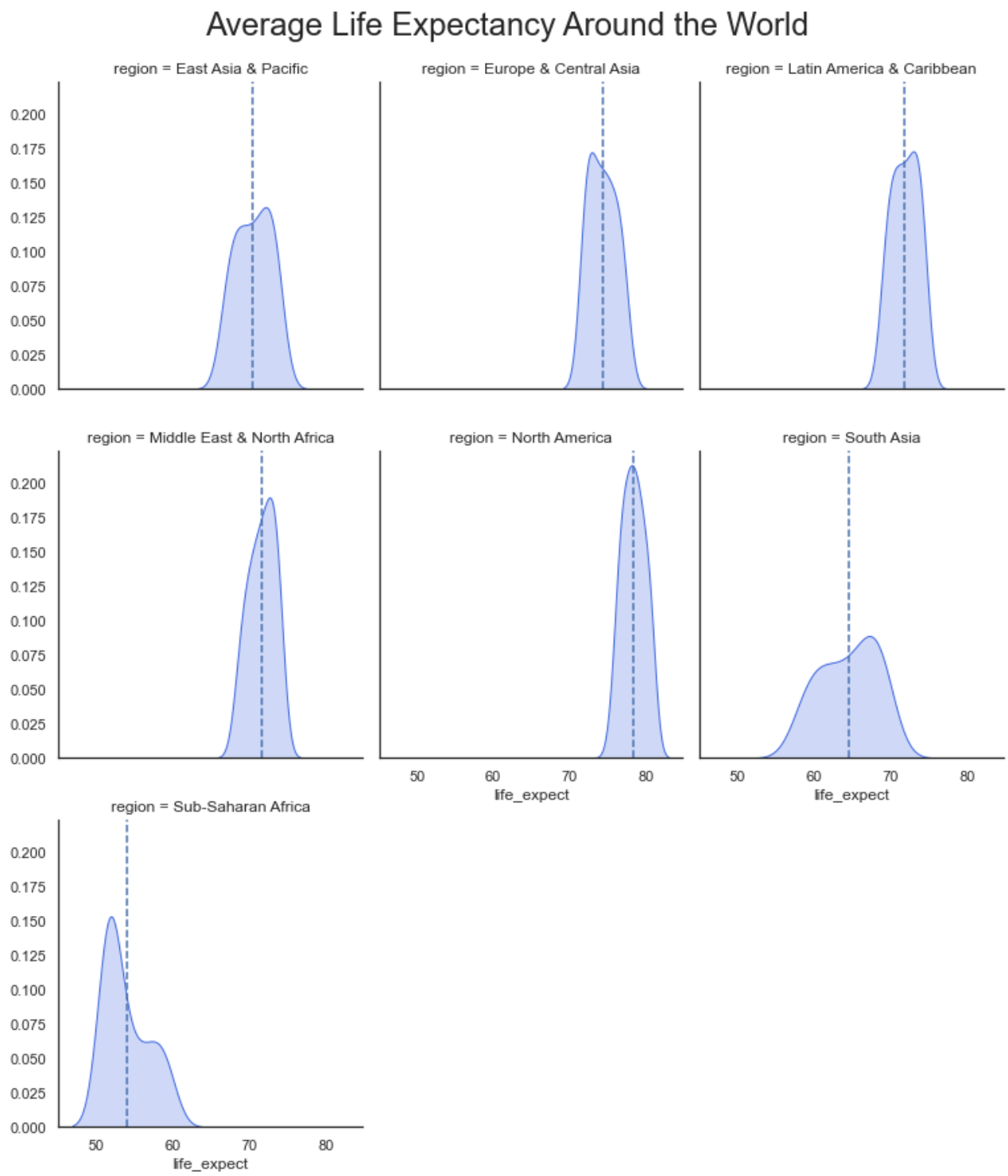
```
# Modifications from QMB 6930 Example
# A final seaborn plot useful for looking at univariate relations is the kdeplot
# which creates and visualizes a kernel density estimate of the underlying featu
g = sns.FacetGrid(nations2008, hue="region", size=6) \
.map(sns.kdeplot, "gdp_percap") \
.add_legend()

plt.title("GDP Around the World")
plt.ylabel("Density")
plt.xlabel("Per Capita GDP in USD")
g.set(xlim=(-25000, 100000))
plt.xticks(rotation=45) #call after xlim=
sns.despine()
```



## KDE with Mean Line

In [32]:
```
#Example from [Bolser Medium Article](#Bosler,-F.)

def vertical_mean_line(x, **kwargs):
    plt.axvline(x.mean(), linestyle ="--",
                color = kwargs.get("color", "r"))
    txkw = dict(size=10, color = kwargs.get("color", "r"))
```

In [33]:
```
_ = nations.groupby(['region','year'])['life_expect'].mean().reset_index()

g = sns.FacetGrid(_, col="region", height=4, aspect=0.9, col_wrap=3, margin_titl
g.map(sns.kdeplot, "life_expect", shade=True, color='royalblue')
g.map(vertical_mean_line, "life_expect")

#FacetGrids object has no attribute title
#g.title("Average Life Expectancy Around the World")

#Instead use fig.suptitle
g.fig.suptitle("Average Life Expectancy Around the World", x=.5, y=1.03,
                   fontsize=24, fontdict={"weight": "bold"})

sns.despine()
```



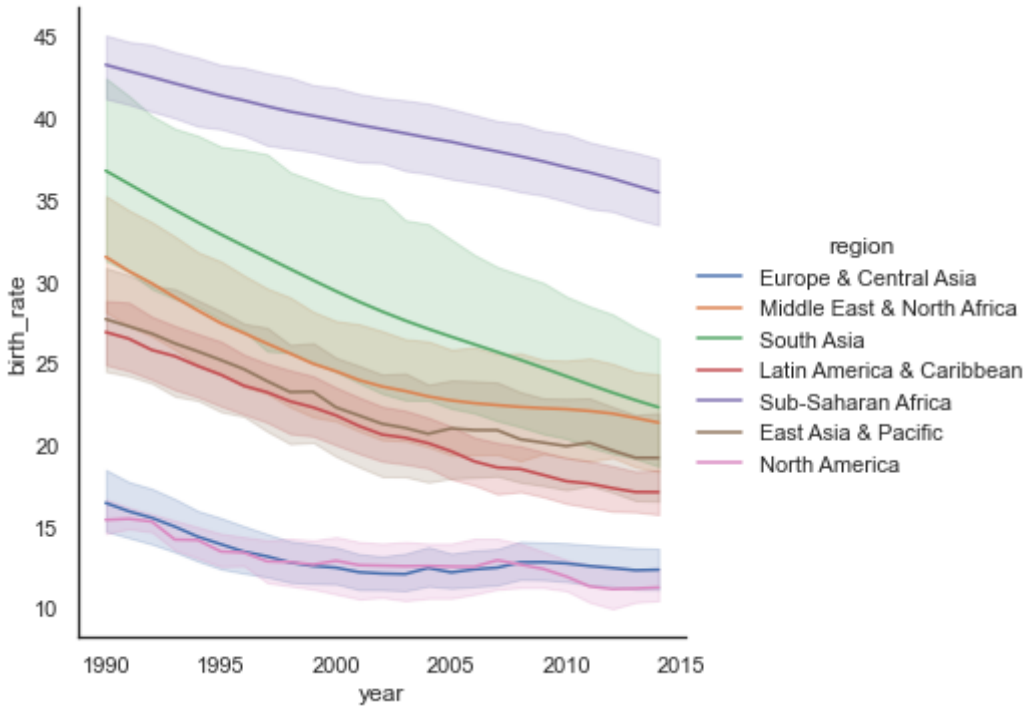

# Line Plots

- **sns.pointplot():** Show Point Estimates and Confidence Intervals as Bars sns.pointplot(x="region", y="gdp_percap", data=nations)
- **sns.relplot():** kind="line" specifies line plot sns.relplot(x="region", y="gdp_percap", data=nations, kind="line")

## Line Plot with relplot

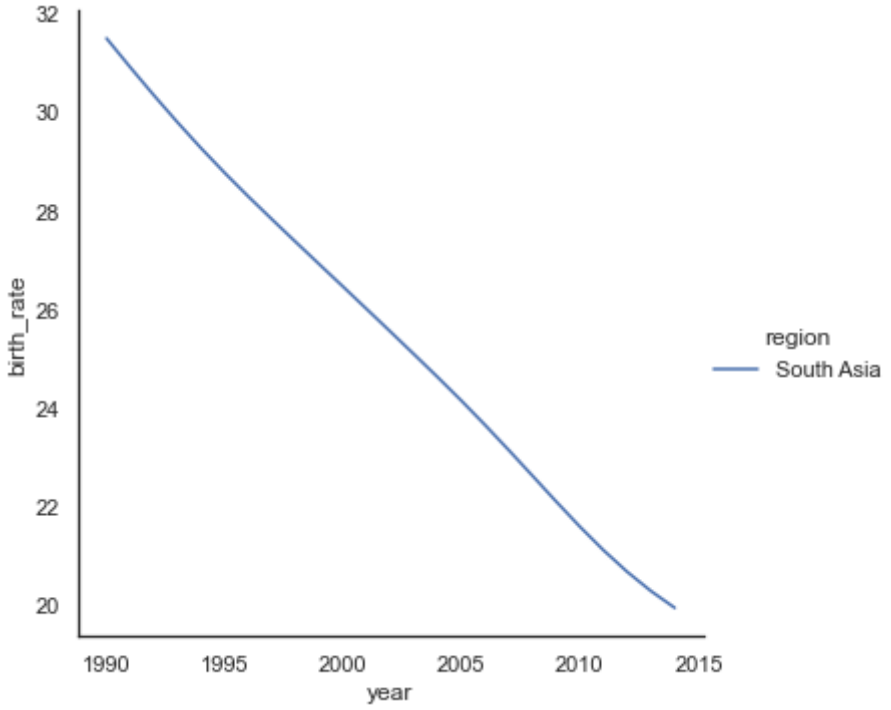
In [34]:
```
#kind="line" specifies the type of plot
```

```
sns.relplot(
    data=nations,
    kind="line",
    y="birth_rate", x="year",
    hue="region",
    facet_kws=dict(sharex=False))

sns.despine()
```



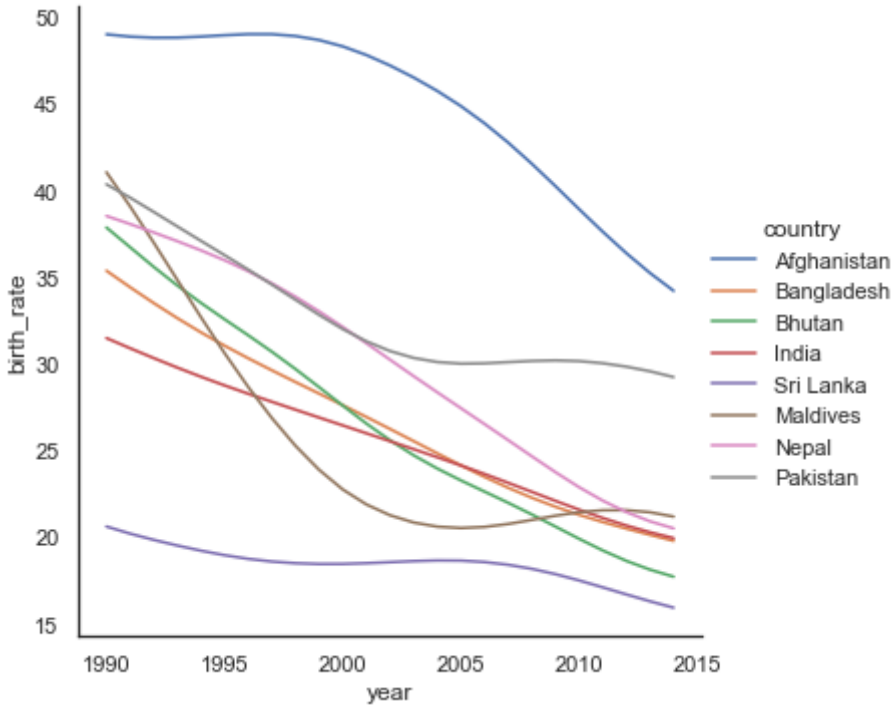

In [35]:

```
sns.relplot(
    data=nations[nations['country'] == 'India'],
    kind="line",
    y="birth_rate", x="year",
    hue="region")

sns.despine()
```




In [36]:

```
sns.relplot(
    data=nations[nations['region'] == 'South Asia'],
    kind="line",
    y="birth_rate", x="year",
    hue="country")

sns.despine()
```

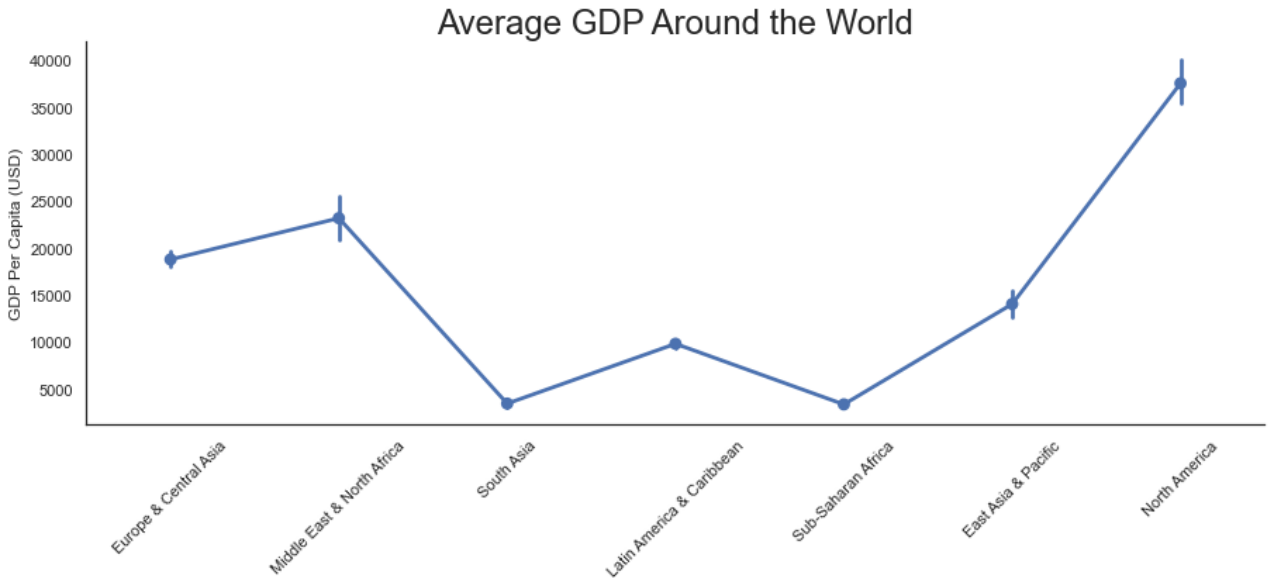



## Pointplots

In [37]:

```
#Show Point Estimates and Confidence Intervals as Bars
```

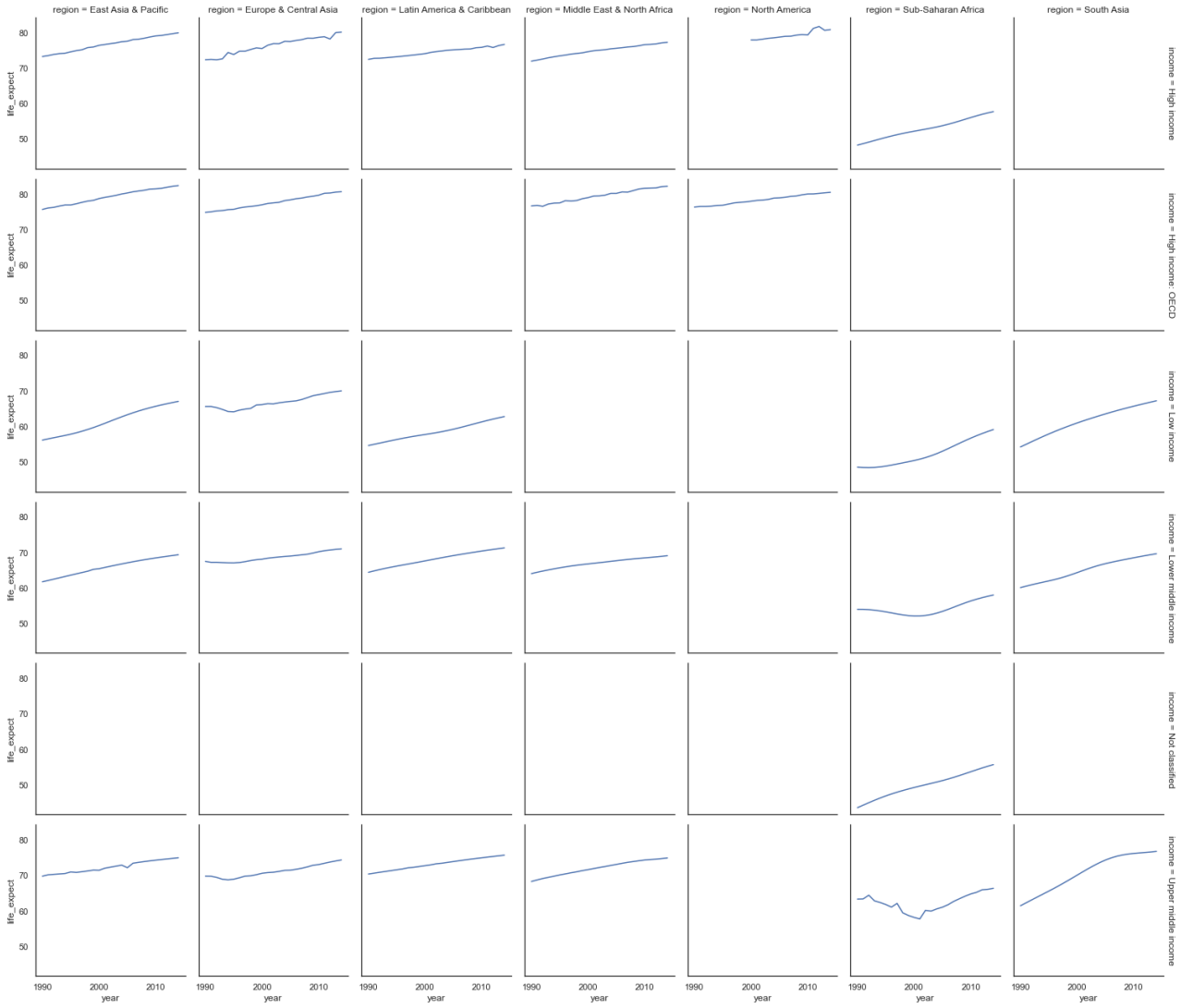
```
sns.pointplot(x="region", y="gdp_percap", data=nations)
plt.title("Average GDP Around the World", size=24)
plt.ylabel("GDP Per Capita (USD)")
plt.xlabel("")
plt.xticks(rotation=45)
sns.despine()
```




## Facet Grid, Lineplots

In [38]:

```
#Large facet grid using 4 variables.
#Time as x, and life expectanacy as y for each graph.
#Then region and income as graph rows and columns.

g = sns.FacetGrid(
    nations.groupby(['income','year','region'])['life_expect'].mean().reset_inde
    row='income',
    col='region',
    margin_titles=True
)
g = (g.map(plt.plot, 'year','life_expect'))
```




# Regression & Residual Plots

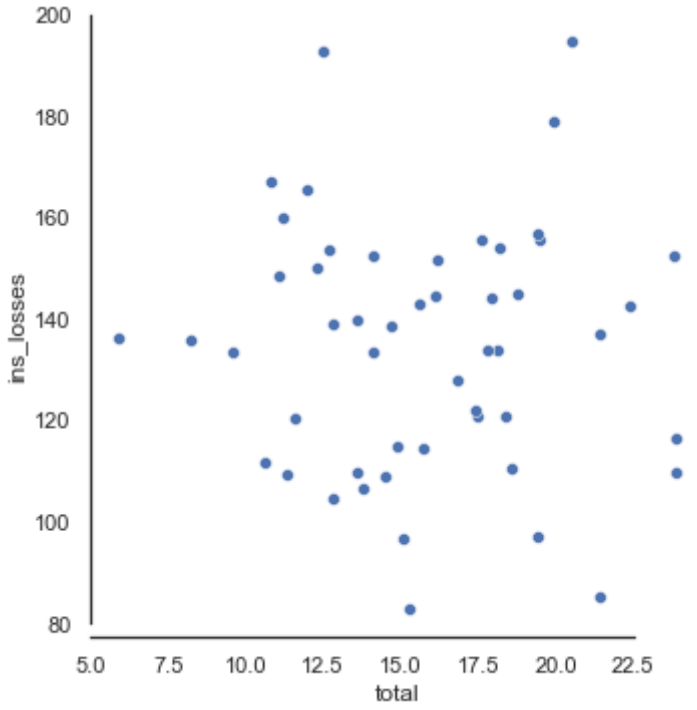
## Residual Plots

#load the dataset variable = sns.load_dataset('nameoffile') sns.residplot(x='columnname', y='columnnameb', data=variable, color='indianred') plt.show() #residual plots have similar arguments to lmplot but #x, and y can by numPy arrays or strings # data argument is opitional #opitional arguments are consistant with matplotlib, for example color =

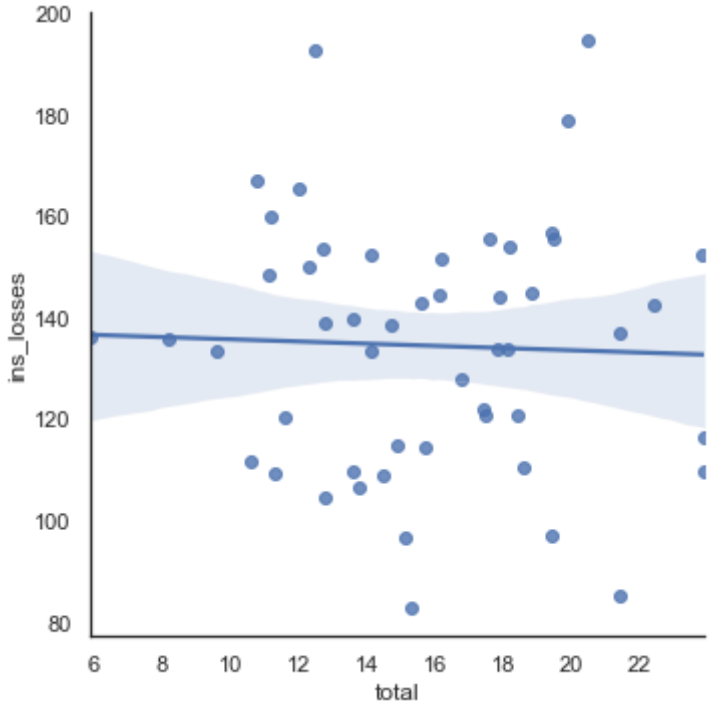
# Scatterplots

- **sns.relplot():** Figure-level interface for drawing relational plots onto a FacetGrid.
  sns.relplot(x="species", y="petal_length", data=iris, kind="scatter")
- **sns.scatterplot():** Draw a scatter plot with possibility of several semantic groupings.
  sns.scatterplot(x="species", y="petal_length", data=iris)
- **sns.stripplot():** Scatterplot with one categorical variable and a jitter
  sns.stripplot(x="species", y="petal_length", data=iris)
- **sns.swarmplot():** Categorical scatterplot with non-overlapping points
  sns.swarmplot(x="species", y="petal_length", data=iris)

In [40]:
```
sns.relplot(
    data=cars,
    x="total", y="ins_losses")
sns.despine(trim=True)
```

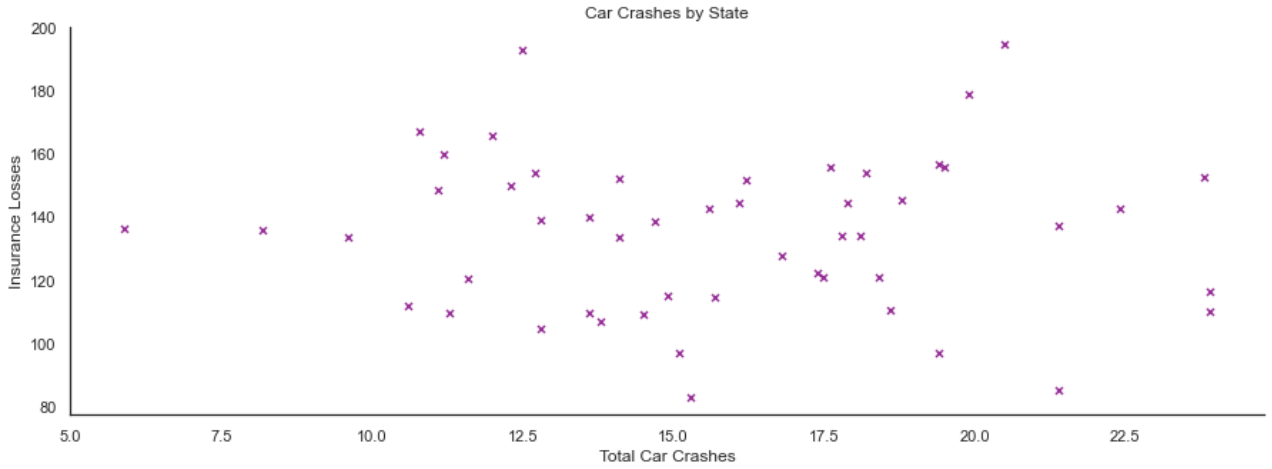



In [41]:
```
#Add a. regression line
sns.lmplot(data=cars, x="total", y="ins_losses")
sns.despine()
```




In [42]:
```
#Change the marker, size and color
plot =sns.regplot(data=cars, x="total", y="ins_losses", fit_reg=False, marker="x
plt.title("Car Crashes by State")
```

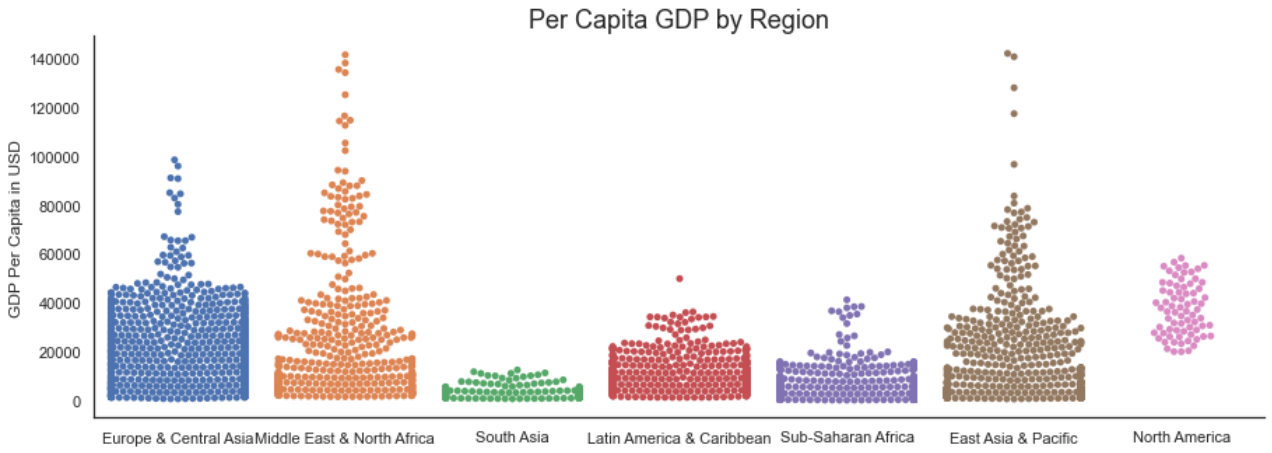
```
plt.xlabel("Total Car Crashes")
plt.ylabel("Insurance Losses")
sns.despine()
```




## Swarm Plots

#Swarm plots, unlike strip plots automatically show points representing repeated values to avoid overlap. import pandas as pd import matplotlib.pyplot as plt import seaborn as sns #load the dataset variable = sns.load_dataset('nameoffile') sns.swarmplot(x='columnname'. y='columnnameb', data=dataframe) plt.ylabel('Label') plt.show() #You can group the data further by making dots different colors based on variables in a specified column using the argument hue= import pandas as pd import matplotlib.pyplot as plt import seaborn as sns #load the dataset variable = sns.load_dataset('nameoffile') sns.swarmplot(x='columnname', y='columnnameb', data=dataframe, hue='columnnamec') plt.ylabel('Label') plt.show() #You can also change the orientation by using the argument orient='h' import pandas as pd import matplotlib.pyplot as plt import seaborn as sns #load the dataset variable = sns.load_dataset('nameoffile') sns.swarmplot(x='columname', y='columnnameb', data=dataframe, hue='columnnamec', orient='h') plt.xlabel('Label') plt.show()

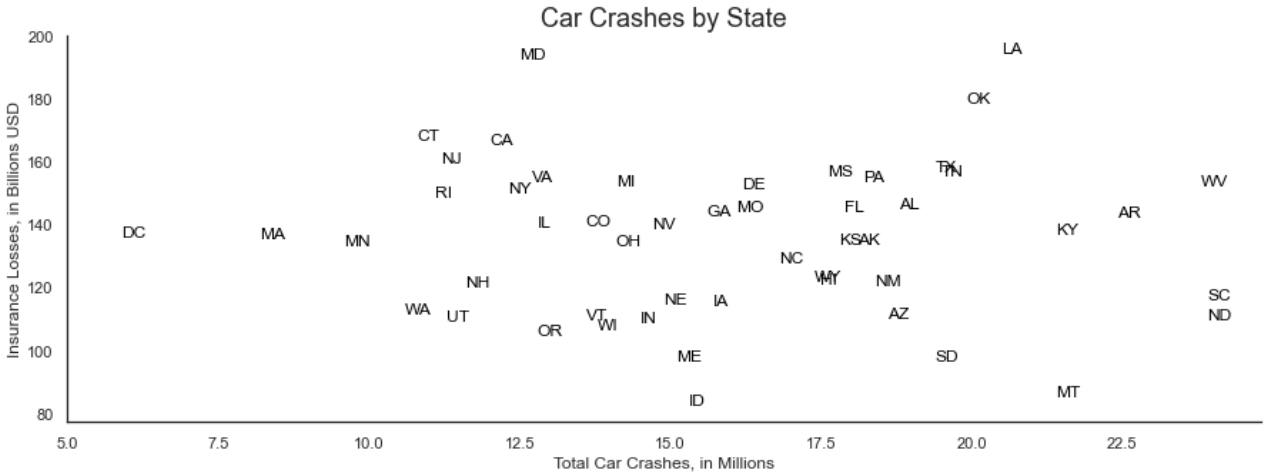
In [44]:
```
#Takes awhile to load
sns.swarmplot(data=nations, x="region", y="gdp_percap")
plt.title("Per Capita GDP by Region", size=18)
plt.xlabel("")
plt.ylabel("GDP Per Capita in USD")
sns.despine()
```




## Point Annotations

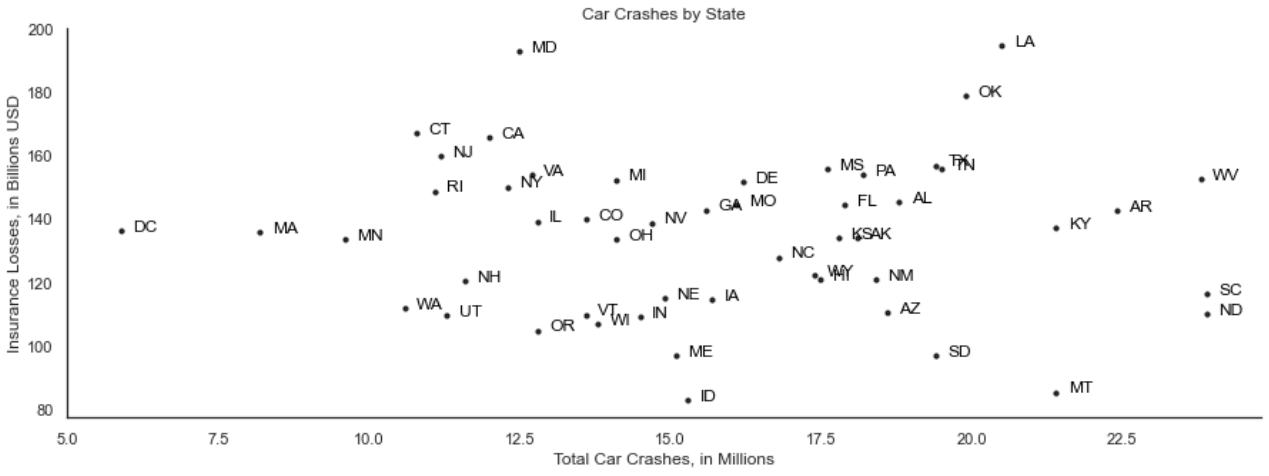
In [45]:
```
#Use State Abbrevations instead of points for markers
```

```
#Note by specifying scatter_kws={'s':0} we are making the points disappear

plot =sns.regplot(data=cars, x="total", y="ins_losses", fit_reg=False, scatter_k

for line in range(0, cars.shape[0]):
     plot.text(cars.total[line], cars.ins_losses[line], cars.abbrev[line], horiz

plt.title("Car Crashes by State", size=18)
plt.xlabel("Total Car Crashes, in Millions")
plt.ylabel("Insurance Losses, in Billions USD")

sns.despine()
```




In [46]:

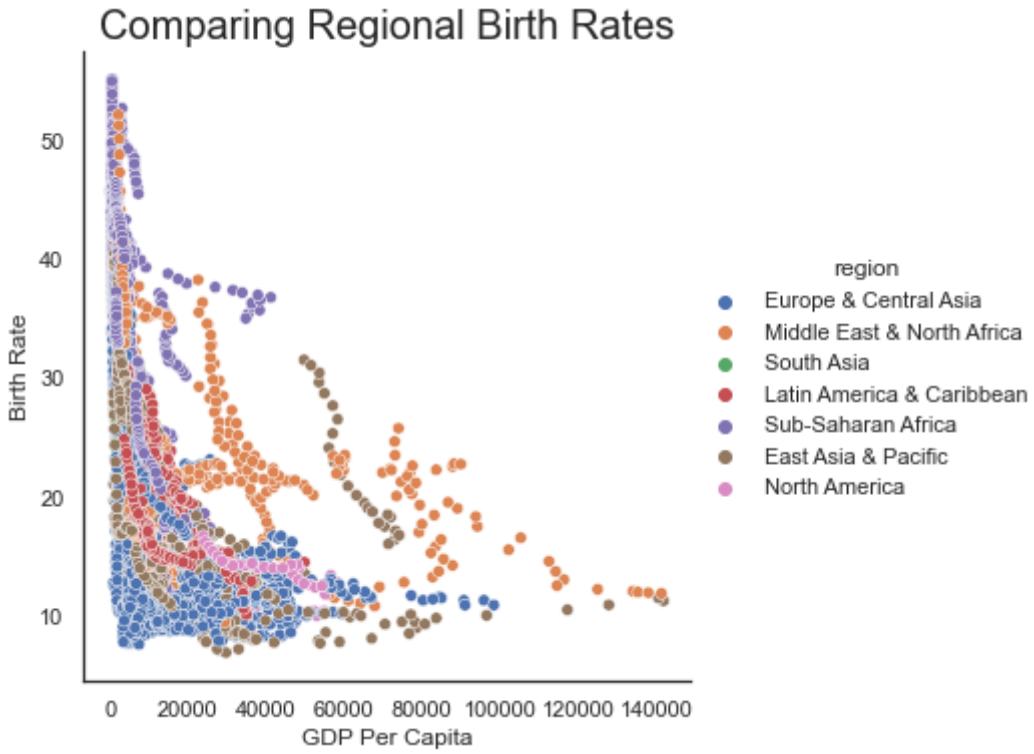
```
#Use both points and State Abbrevations
#The +0.2 moves the abbrevation a tad to the right
plot =sns.regplot(data=cars, x="total", y="ins_losses", fit_reg=False, marker="o

for line in range(0, cars.shape[0]):
     plot.text(cars.total[line]+0.2, cars.ins_losses[line], cars.abbrev[line], h

plt.title("Car Crashes by State")
plt.xlabel("Total Car Crashes, in Millions")
plt.ylabel("Insurance Losses, in Billions USD")

sns.despine()
```




In [47]:

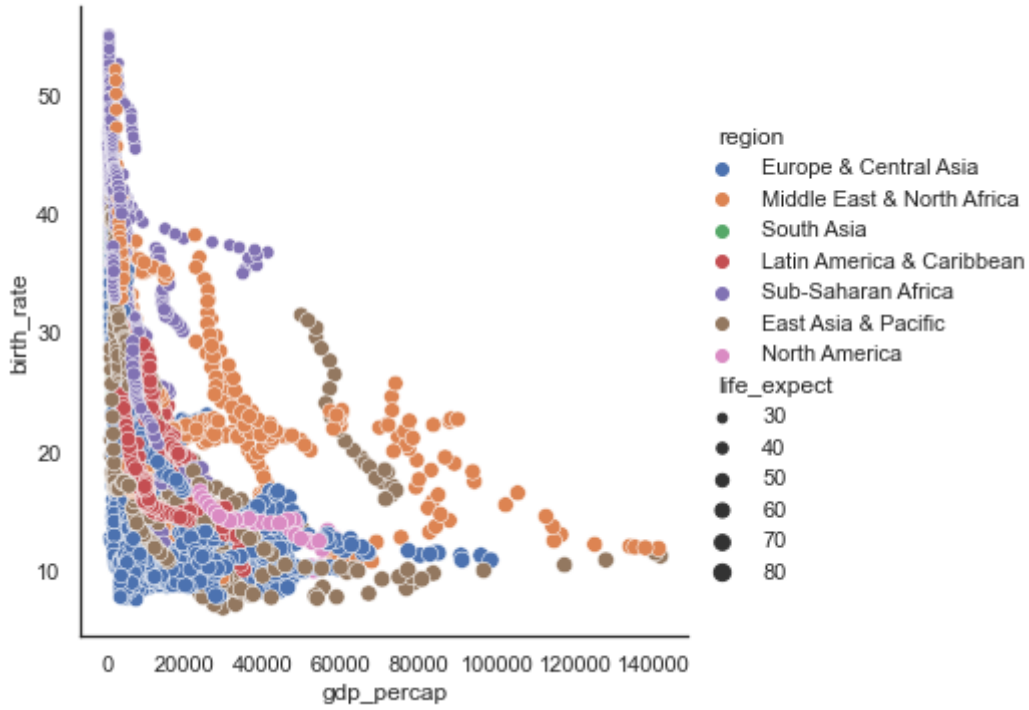
```
#hue= changes the colors of the points based on 3rd variable
```

```
sns.relplot(
    data=nations,
    y="birth_rate", x="gdp_percap",
    hue="region")

plt.title("Comparing Regional Birth Rates", size=20)
plt.xlabel("GDP Per Capita")
plt.ylabel("Birth Rate")

sns.despine()
```




In [48]:

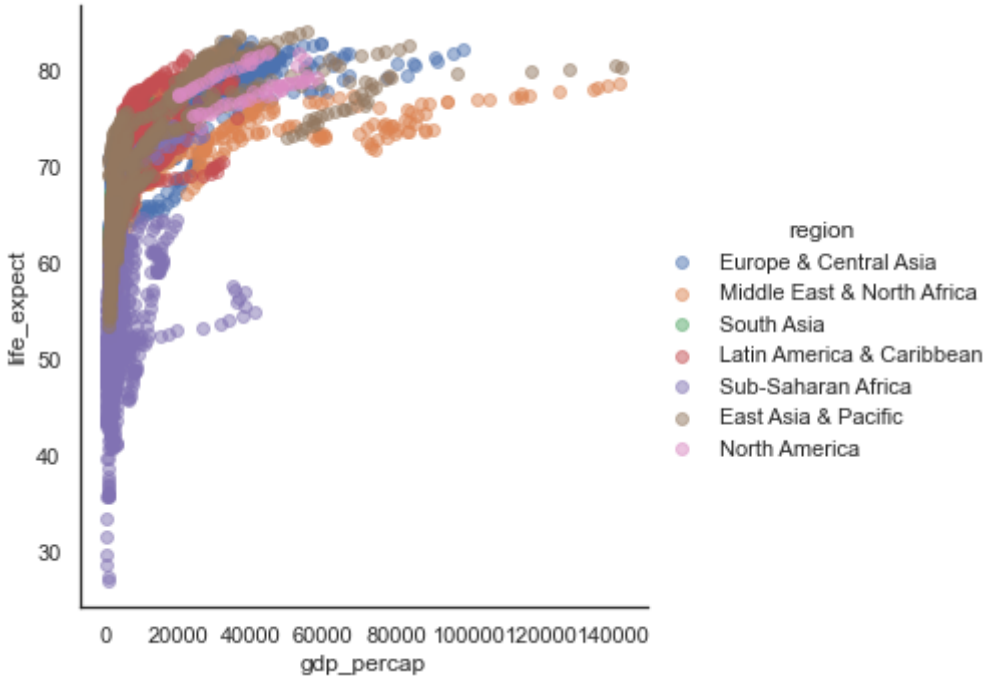
```
#size= changes the size of the points based on variable
sns.relplot(
    data=nations,
    y="birth_rate", x="gdp_percap",
    hue="region",
    size="life_expect")

sns.despine()
```




## Facetgrid, Layer Single Plot

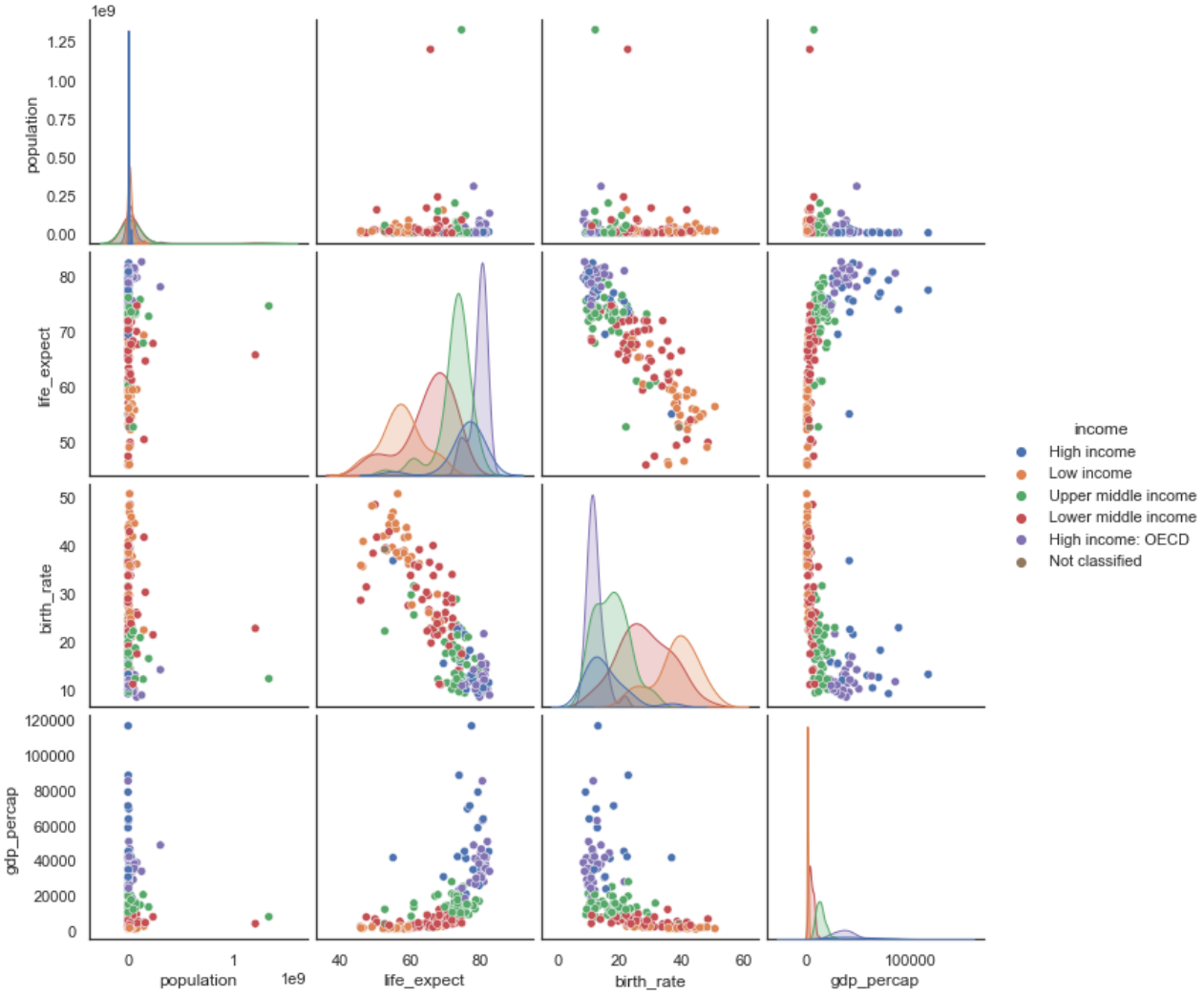
Facetgrid allows multiple plots like small-multiples, trellis, lattice graphs of different plots (line, bar, scatter)

In [49]:
```
# Example from QMB 6930
# FacetGrid to color the scatterplot by regions
sns.FacetGrid(nations, hue="region", size=5) \
.map(plt.scatter, "gdp_percap", "life_expect", alpha = 0.5) \
.add_legend()

sns.despine()
```



## Matrix Multiples

In [50]:
```
sns.pairplot(
    nations[nations.year == 2008][[
        'income', 'population', 'life_expect', 'birth_rate', 'gdp_percap'
    ]].dropna(),
    hue='income'
)

sns.despine()
```

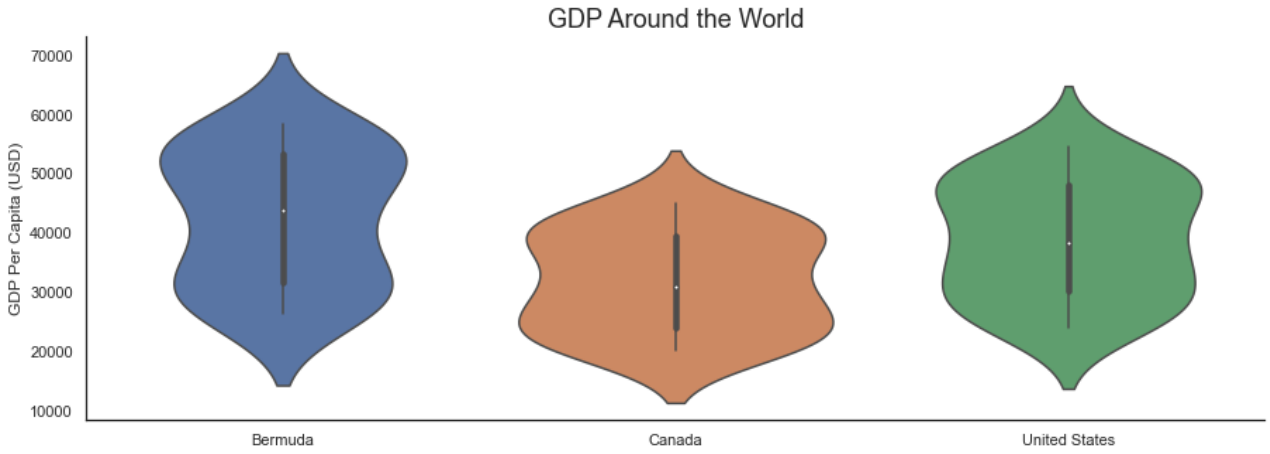



## Strip Plots

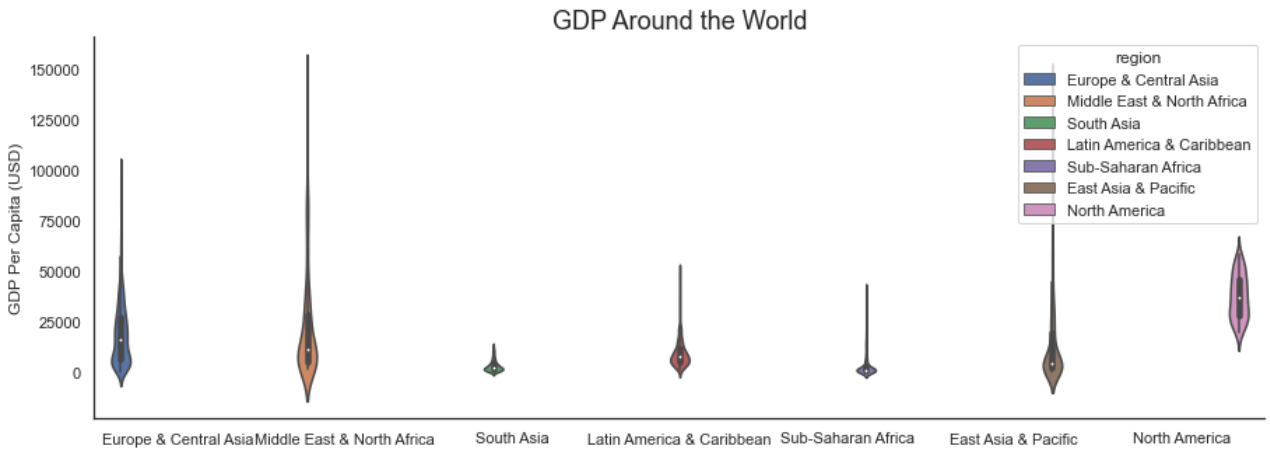
#Strip Plots draws values on a number line to visualize samples of a single variable import pandas as pd import matplotlib.pyplot as plt import seaborn as sns #load the dataset variable = sns.load_dataset('nameoffile') sns.stripplot(y='columnname', data=variable) plt.ylabel('Label') plt.show() #You can draw parrell strip plots for each variable in x import pandas as pd import matplotlib.pyplot as plt import seaborn as sns #load the dataset variable = sns.load_dataset('nameoffile') sns.stripplot(x='columnname', y ='columnnameb', data=variable) plt.ylabel('Label') plt.show() #In a strip plot repeated values are drawn ontop of each other. #To show repeated values you can add the arguments size= and jitter=True or use a swarm plot. import pandas as pd import matplotlib.pyplot as plt import seaborn as sns #load the dataset variable = sns.load_dataset('nameoffile') sns.stripplot(x='columnname', y='columnnameb', data=variable, size=4, jitter=True) plt.ylabel('Label') plt.show()

## Violin Plots

In [52]:
```
# Denser regions of the data are fatter, and sparser thiner in a violin plot
sns.violinplot(x="country", y="gdp_percap", data= nations[nations.region == "Nor

plt.title("GDP Around the World", size=18)
plt.ylabel("GDP Per Capita (USD)")
plt.xlabel("")

sns.despine()
```




In [53]:
```
#Use rc={'figure.figsize': to specify chart sizes, in this case making it wider

sns.set(
    rc={'figure.figsize':(15,5)},
    style="white"
)
sns.violinplot(
    x='region',
    y='gdp_percap',
    hue='region',
    data=nations
)

plt.title("GDP Around the World", size=18)
plt.ylabel("GDP Per Capita (USD)")
plt.xlabel("")

sns.despine()
```



#Takes forever to load g = sns.catplot(x='life_expect', y='gdp_percap',hue='region', row="income", data=nations, orient="h", palette="Set3", kind="violin", dodge=True, cut=0, bw=.2)

# Appendix 1: Summary of Arguments

## sns.despine()

seaborn.despine(fig=None, ax=None, top=True, right=True, left=False, bottom=False, offset=None, trim=False) Remove the top and right spines from plot(s).

- **Best Use:** Removes the <AxesSubplot:xlabel='region', ylabel='gdp_percap'> from the top of plots
- **Doucmentation:** https://seaborn.pydata.org/generated/seaborn.despine.html
- **Example:** Frequently used on Line Plots, Histograms, Scatterplots and Violin Plots above

## sns.set()

seaborn.set_theme(context='notebook', style='darkgrid', palette='deep', font='sans-serif', font_scale=1, color_codes=True, rc=None)

- **Best Use:** Specify the size of charts
- **Documentation:** https://seaborn.pydata.org/generated/seaborn.set_theme.html#seaborn.set_theme
- **Example:** Boxplots for changing theme color and Violin Plots for changing the plot size.
- **Note:** sns.set is the shorthand of seaborn.set_theme

# Appendix 2: Colors

- **color_palette():** define a color map that you want to be using and the number of colors with the argument n_colors (Willems 2017). Documentation: https://seaborn.pydata.org/generated/seaborn.color_palette.html Seaborn Tutorial: https://seaborn.pydata.org/tutorial/color_palettes.html#palette-tutorial

In [55]:
```
#Sets the color for all your graphs
sns.set_palette("tab10")
```

In [56]:
```
sns.color_palette()
```

Out[56]:

In [57]:
```
sns.color_palette(n_colors=5)
```

Out[57]:

In [58]:
```
#Desaturate colors by specifing a float to desat=
sns.color_palette(n_colors=5, desat=.33)
```

Out[58]:

In [59]:
```
sns.color_palette(palette='colorblind')
```

Out[59]:

In [60]:
```
sns.color_palette("flare")
```

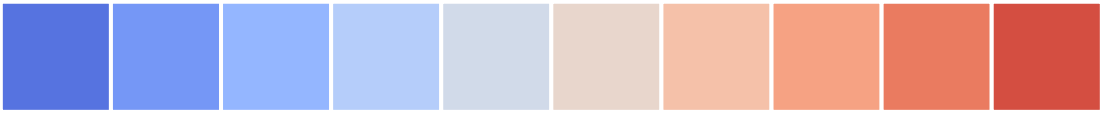
Out[60]:

In [61]:
```
#Reverse order
sns.color_palette("flare_r")
```

Out[61]:

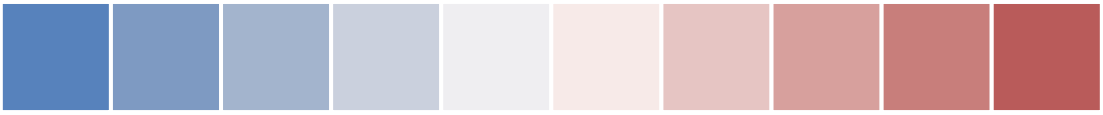
In [62]:
```
sns.color_palette("coolwarm", n_colors=10)
```

Out[62]:



In [63]:
```
sns.color_palette('vlag', n_colors=10)
```

Out[63]:



## Appendix 3: Combining Plots

sns.violinplot(x='columnname', y='columnnameb', data=variable, inner=None, color='lightgray') sns.stripplot(x='columnname', y='columnnameb', data=variable, size=4, jitter=True) plt.ylabel('Label') plt.xlabel('Label') Plt.show()

# References

Formated in APA with the authors name on it's own line in order to hyperlink in document. Then bullet points below on what I found particularly useful.

Lastname, F. M. (Year, Month Date). *Title of page*. Site name. URL

**Bosler, F.**

(2019, October 20). *Learn how to create beautiful and insightful charts with Python — the Quick, the Pretty, and the Awesome*. Median, Towards Data Science. https://towardsdatascience.com/plotting-with-python-c2561b8c0f1f

- Great examples of facet grids and the matrix small multiples.

**Willems, K.**

(2017, August 10) *Python Seaborn Tutorial For Beginners*. DataCamp Tutorials. https://www.datacamp.com/community/tutorials/seaborn-python-tutorial#xlim

In [ ]: