Day18_Pandas_Introduction_with_GDP_Dataset2

June 6, 2025

Yesterday I started learning Pandas, and today I'm continuing with more practice like advanced filtering, grouping (groupby), merging DataFrames, and handling missing values.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# Load data
df = pd.read_csv(r'C:
↪\Users\aksha\OneDrive\Desktop\Dataset\Pandas_intro_gdp_data\data.csv')
```

[3]:
```
df
```

[3]:

	CountryName	CountryCode	BirthRate	InternetUsers	IncomeGroup
0	Aruba	ABW	10.244	78.9	High income
1	Afghanistan	AFG	35.253	5.9	Low income
2	Angola	AGO	45.985	19.1	Upper middle income
3	Albania	ALB	12.877	57.2	Upper middle income
4	United Arab Emirates	ARE	11.044	88.0	High income
..	…	…	…	…	…
190	Yemen, Rep.	YEM	32.947	20.0	Lower middle income
191	South Africa	ZAF	20.850	46.5	Upper middle income
192	Congo, Dem. Rep.	COD	42.394	2.2	Low income
193	Zambia	ZMB	40.471	15.4	Lower middle income
194	Zimbabwe	ZWE	35.715	18.5	

```
194           Low income

[195 rows x 5 columns]
```

Rename Column Name

[5]:
```
df.columns
```

[5]:
```
Index(['CountryName', 'CountryCode', 'BirthRate', 'InternetUsers',
       'IncomeGroup'],
      dtype='object')
```

[9]:
```
# Way 1
# Rename only column
df.rename(columns={'CountryName':'NameOfCountry'}, inplace=True)
df.rename(columns={'CountryCode':'CodeOfCountry'}, inplace=True)
```

[10]:
```
df.columns # Changed column names
```

[10]:
```
Index(['NameOfCountry', 'CodeOfCountry', 'BirthRate', 'InternetUsers',
       'IncomeGroup'],
      dtype='object')
```

[11]:
```
# Rename only column
df.rename(columns={'NameOfCountry': 'CountryName', 'CodeOfCountry':
 ↪'CountryCode'}, inplace=True)
```

[12]:
```
df.columns # Changed column names
```

[12]:
```
Index(['CountryName', 'CountryCode', 'BirthRate', 'InternetUsers',
       'IncomeGroup'],
      dtype='object')
```

[13]:
```
# Way 2
df.columns
```

[13]:
```
Index(['CountryName', 'CountryCode', 'BirthRate', 'InternetUsers',
       'IncomeGroup'],
      dtype='object')
```

[14]:
```
 df.columns = ['a','b','c','d','e'] # Any names
```

[15]:
```
df.columns
```

[15]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

[16]:
```
df.columns = ['CountryName', 'CountryCode', 'BirthRate', 'InternetUsers',
       'IncomeGroup']
df
```

[16]:
```
              CountryName CountryCode  BirthRate  InternetUsers  \
0                   Aruba         ABW     10.244           78.9
1             Afghanistan         AFG     35.253            5.9
2                  Angola         AGO     45.985           19.1
3                 Albania         ALB     12.877           57.2
4    United Arab Emirates         ARE     11.044           88.0
..                    …         …         …              …
190          Yemen, Rep.         YEM     32.947           20.0
191         South Africa         ZAF     20.850           46.5
192     Congo, Dem. Rep.         COD     42.394            2.2
193               Zambia         ZMB     40.471           15.4
194             Zimbabwe         ZWE     35.715           18.5

             IncomeGroup
0            High income
1             Low income
2    Upper middle income
3    Upper middle income
4            High income
..                    …
190  Lower middle income
191  Upper middle income
192           Low income
193  Lower middle income
194           Low income

[195 rows x 5 columns]
```

Selecting Columns in Pandas

[17]:
```
# This is just a Python list containing two column names as strings.

# It does not access any data - it's just a list.

['CountryName', 'BirthRate']
```

[17]:
```
['CountryName', 'BirthRate']
```

[18]:
```
# This is how you use that list to select multiple columns from a DataFrame df.

# It tells Pandas: "Give me only the CountryName and BirthRate columns from the␣
 ↪DataFrame."

# The result will be a new DataFrame with only those two columns.
df[['CountryName', 'BirthRate']]
```

[18]:
```
              CountryName  BirthRate
0                   Aruba     10.244
```

```
1              Afghanistan     35.253
2                  Angola      45.985
3                 Albania      12.877
4    United Arab Emirates      11.044
..                     …         …
190           Yemen, Rep.      32.947
191          South Africa      20.850
192      Congo, Dem. Rep.      42.394
193                Zambia      40.471
194              Zimbabwe      35.715

[195 rows x 2 columns]
```

Genrate New Column / Feature

[19]:
```
# make new columns
df.BirthRate * df.InternetUsers
```

[19]:
```
0      808.2516
1      207.9927
2      878.3135
3      736.5644
4      971.8720
         …
190    658.9400
191    969.5250
192     93.2668
193    623.2534
194    660.7275
Length: 195, dtype: float64
```

[20]:
```
df.head(2)
```

[20]:
```
   CountryName CountryCode  BirthRate  InternetUsers  IncomeGroup
0        Aruba         ABW     10.244           78.9  High income
1  Afghanistan         AFG     35.253            5.9   Low income
```

[21]:
```
# make new column and add to DF
df['MyCalculation']=df.BirthRate * df.InternetUsers
```

[22]:
```
df.columns
```

[22]:
```
Index(['CountryName', 'CountryCode', 'BirthRate', 'InternetUsers',
       'IncomeGroup', 'MyCalculation'],
      dtype='object')
```

[23]:
```
df.head(2)
```

[23]:
```
   CountryName CountryCode  BirthRate  InternetUsers  IncomeGroup  \
0        Aruba         ABW     10.244           78.9  High income
1  Afghanistan         AFG     35.253            5.9   Low income

   MyCalculation
0       808.2516
1       207.9927
```

Deleting a Column from a DataFrame in Pandas

To **delete a column** temporarily (without changing the original DataFrame), you can use the `drop()` function with `axis=1`.

"'python # axis=1 → means column # axis=0 → means row

[25]:
```
df.drop('MyCalculation', axis=1) # temporarily delete a column and show DF
```

[25]:
```
              CountryName CountryCode  BirthRate  InternetUsers  \
0                   Aruba         ABW     10.244           78.9
1             Afghanistan         AFG     35.253            5.9
2                  Angola         AGO     45.985           19.1
3                 Albania         ALB     12.877           57.2
4    United Arab Emirates         ARE     11.044           88.0
..                    …         …       …              …
190          Yemen, Rep.         YEM     32.947           20.0
191         South Africa         ZAF     20.850           46.5
192     Congo, Dem. Rep.         COD     42.394            2.2
193               Zambia         ZMB     40.471           15.4
194             Zimbabwe         ZWE     35.715           18.5

             IncomeGroup
0            High income
1             Low income
2    Upper middle income
3    Upper middle income
4            High income
..                   …
190  Lower middle income
191  Upper middle income
192           Low income
193  Lower middle income
194           Low income

[195 rows x 5 columns]
```

[26]:
```
df.columns # Still here
```

[26]:
```
Index(['CountryName', 'CountryCode', 'BirthRate', 'InternetUsers',
       'IncomeGroup', 'MyCalculation'],
```

```
      dtype='object')
```

Permanently Delete a Column from a DataFrame

If you want to **remove a column and keep it removed** in your DataFrame, you have two options:

“‘python # Option 1: Reassign the result back to the same DataFrame df = df.drop(‘MyCalculation’, axis=1)

[27]:
```
# Option 2: Use inplace=True to modify the original DataFrame directly
df.drop('MyCalculation', axis=1, inplace=True)
```

[28]:
```
df.columns
```

[28]:
```
Index(['CountryName', 'CountryCode', 'BirthRate', 'InternetUsers',
       'IncomeGroup'],
      dtype='object')
```

Filtering Data in Pandas using a Condition

Let’s say we want to filter rows where the number of **Internet Users is less than 2**.

[30]:
```
# Step 1: Write a condition
Filter = df.InternetUsers < 2
Filter
# This creates a Boolean Series (True/False) for each row in the DataFrame.
```

[30]:
```
0      False
1      False
2      False
3      False
4      False
       …
190    False
191    False
192    False
193    False
194    False
Name: InternetUsers, Length: 195, dtype: bool
```

[31]:
```
# Step 2: Apply the filter to the DataFrame
df[Filter]
# This returns only the rows where the condition is True (i.e., InternetUsers <␣
↪2).
```

[31]:

	CountryName	CountryCode	BirthRate	InternetUsers	IncomeGroup
11	Burundi	BDI	44.151	1.3	Low income
52	Eritrea	ERI	34.800	0.9	Low income
55	Ethiopia	ETH	32.925	1.9	Low income

```
64          Guinea          GIN     37.337            1.6           Low income
117        Myanmar          MMR     18.119            1.6  Lower middle income
127          Niger          NER     49.661            1.7           Low income
154   Sierra Leone          SLE     36.729            1.7           Low income
156        Somalia          SOM     43.891            1.5           Low income
172    Timor-Leste          TLS     35.755            1.1  Lower middle income
```

[32]:
```
# Step 3: Check how many rows match the condition
len(df[Filter])
# This gives you the count of rows where Internet Users are less than 2.
```

[32]:
```
9
```

We can also combine conditions using & (and), | (or), and ~ (not)

[33]:
```
df[(df.InternetUsers < 2) & (df.BirthRate > 30)]
```

[33]:
```
       CountryName CountryCode  BirthRate  InternetUsers           IncomeGroup
11         Burundi          BDI     44.151            1.3           Low income
52         Eritrea          ERI     34.800            0.9           Low income
55        Ethiopia          ETH     32.925            1.9           Low income
64          Guinea          GIN     37.337            1.6           Low income
127          Niger          NER     49.661            1.7           Low income
154   Sierra Leone          SLE     36.729            1.7           Low income
156        Somalia          SOM     43.891            1.5           Low income
172    Timor-Leste          TLS     35.755            1.1  Lower middle income
```

[34]:
```
len(df[(df.InternetUsers < 2) & (df.BirthRate > 30)])
```

[34]:
```
8
```

[35]:
```
# Second example
df.BirthRate > 40
```

[35]:
```
0      False
1      False
2       True
3      False
4      False
       …
190    False
191    False
192     True
193     True
194    False
Name: BirthRate, Length: 195, dtype: bool
```

[37]:
```
Filter2 = df.BirthRate > 40
Filter2
```

[37]:
```
0      False
1      False
2       True
3      False
4      False
       …
190    False
191    False
192     True
193     True
194    False
Name: BirthRate, Length: 195, dtype: bool
```

[38]:
```
df[Filter2]
```

[38]:

	CountryName	CountryCode	BirthRate	InternetUsers	IncomeGroup
2	Angola	AGO	45.985	19.1	Upper middle income
11	Burundi	BDI	44.151	1.3	Low income
14	Burkina Faso	BFA	40.551	9.1	Low income
65	Gambia, The	GMB	42.525	14.0	Low income
115	Mali	MLI	44.138	3.5	Low income
127	Niger	NER	49.661	1.7	Low income
128	Nigeria	NGA	40.045	38.0	Lower middle income
156	Somalia	SOM	43.891	1.5	Low income
167	Chad	TCD	45.745	2.3	Low income
178	Uganda	UGA	43.474	16.2	Low income
192	Congo, Dem. Rep.	COD	42.394	2.2	Low income
193	Zambia	ZMB	40.471	15.4	Lower middle income

[39]:
```
len(df[Filter2])
```

[39]:
```
12
```

[40]:
```
# combine both condition
Filter & Filter2
```

[40]:
```
0      False
1      False
2      False
3      False
4      False
       …
190    False
191    False
192    False
193    False
194    False
Length: 195, dtype: bool
```

[41]:
```
df[Filter & Filter2]
```

[41]:

	CountryName	CountryCode	BirthRate	InternetUsers	IncomeGroup
11	Burundi	BDI	44.151	1.3	Low income
127	Niger	NER	49.661	1.7	Low income
156	Somalia	SOM	43.891	1.5	Low income

[42]:
```
len(df[Filter & Filter2])
```

[42]:
```
3
```

Filtering Rows Where Income Group is 'High income'

[43]:
```
# Step 1: Apply condition directly inside the DataFrame
df[df.IncomeGroup == 'High income']
# This returns only the rows where the IncomeGroup is 'High income'.
```

[43]:

	CountryName	CountryCode	BirthRate	InternetUsers	IncomeGroup
0	Aruba	ABW	10.244	78.90	High income
4	United Arab Emirates	ARE	11.044	88.00	High income
5	Argentina	ARG	17.716	59.90	High income
7	Antigua and Barbuda	ATG	16.447	63.40	High income
8	Australia	AUS	13.200	83.00	High income
..	…	…	…	…	…
174	Trinidad and Tobago	TTO	14.590	63.80	High income
180	Uruguay	URY	14.374	57.69	High income
181	United States	USA	12.500	84.20	High income
184	Venezuela, RB	VEN	19.842	54.90	High income
185	Virgin Islands (U.S.)	VIR	10.700	45.30	High income

```
[67 rows x 5 columns]
```

[44]:
```
# Step 2: You can also save the condition into a variable (optional)
New_Filter = df.IncomeGroup == 'High income'
df[New_Filter]
# Both ways give the same result - a new filtered DataFrame.
```

[44]:

	CountryName	CountryCode	BirthRate	InternetUsers	IncomeGroup
0	Aruba	ABW	10.244	78.90	High income
4	United Arab Emirates	ARE	11.044	88.00	High income
5	Argentina	ARG	17.716	59.90	High income
7	Antigua and Barbuda	ATG	16.447	63.40	High income
8	Australia	AUS	13.200	83.00	High income
..	…	…	…	…	…
174	Trinidad and Tobago	TTO	14.590	63.80	High income
180	Uruguay	URY	14.374	57.69	High income
181	United States	USA	12.500	84.20	High income
184	Venezuela, RB	VEN	19.842	54.90	High income
185	Virgin Islands (U.S.)	VIR	10.700	45.30	High income

[67 rows x 5 columns]

[45]:
```
# Step 3: Count how many rows match the condition
len(df[df.IncomeGroup == 'High income'])
#This tells you how many countries belong to the 'High income' group.
```

[45]:
```
67
```

How to Get Unique Categories in a Column (Pandas)

[46]:
```
# Get all unique categories in the 'IncomeGroup' column
df.IncomeGroup.unique()
```

[46]:
```
array(['High income', 'Low income', 'Upper middle income',
       'Lower middle income'], dtype=object)
```

[47]:
```
# Get the number of unique categories
df.IncomeGroup.nunique()
```

[47]:
```
4
```

[48]:
```
# Import necessary libraries
import seaborn as sns
# Set default figure size for better visibility
plt.rcParams['figure.figsize'] = 6,2
import warnings
# Ignore warnings to keep output clean
warnings.filterwarnings('ignore')
```

[49]:
```
df.columns
```

[49]:
```
Index(['CountryName', 'CountryCode', 'BirthRate', 'InternetUsers',
       'IncomeGroup'],
      dtype='object')
```

[50]:
```
df['InternetUsers']
```

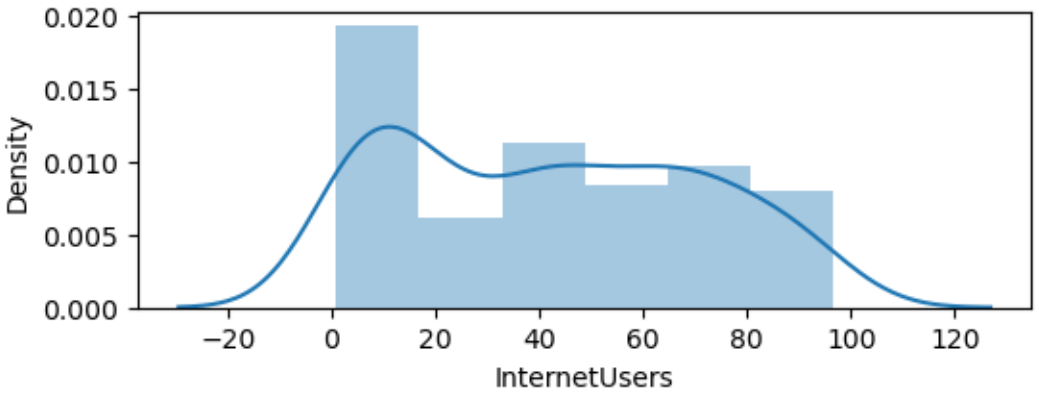
[50]:
```
0      78.9
1       5.9
2      19.1
3      57.2
4      88.0
       ...
190    20.0
191    46.5
192     2.2
193    15.4
194    18.5
Name: InternetUsers, Length: 195, dtype: float64
```

Introduction to Seaborn and statistic Distribuations = Distplot() Histograme = Displot()

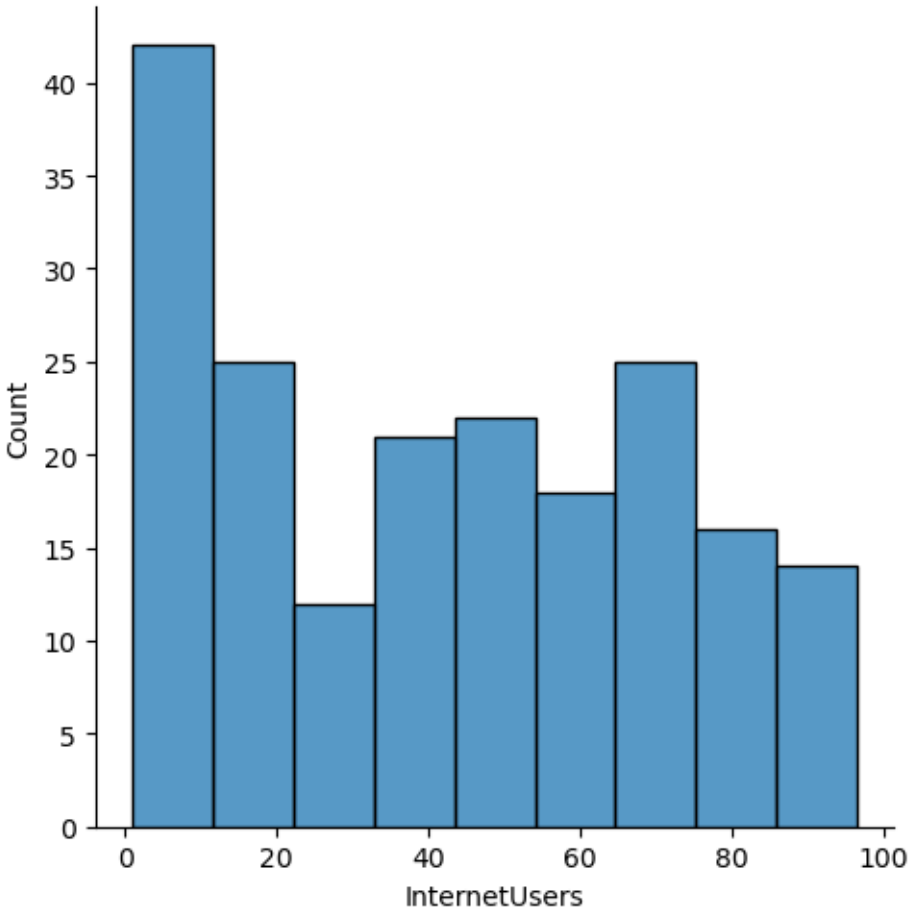
For curv set, kde = True

- **distplot** – old way to make histogram + curve (now not recommended)
- **displot** – new way to make histogram + curve (better and newer)

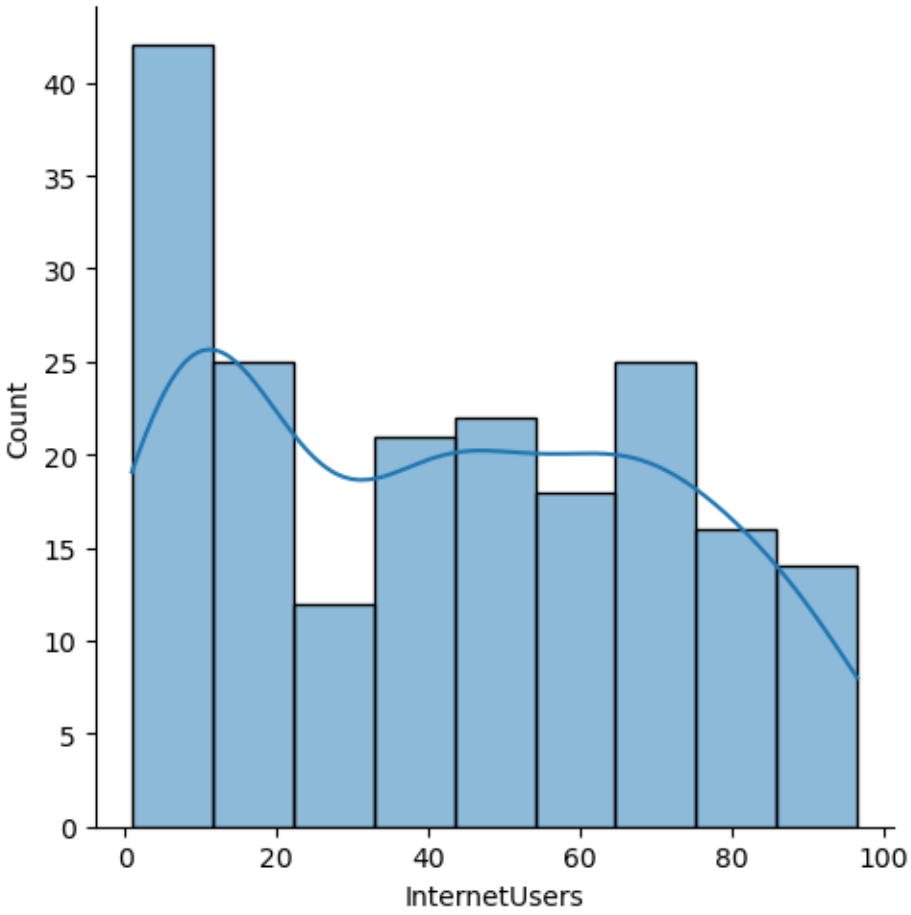
[51]:
```
vis1 = sns.distplot(df['InternetUsers'])
```



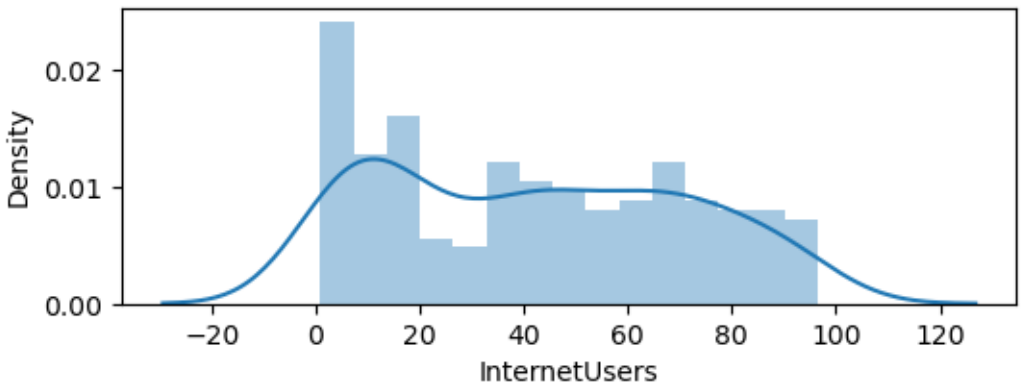
[52]:
```
vis2 = sns.displot(df['InternetUsers'])
```



[53]:
```
vis2 = sns.displot(df['InternetUsers'],kde = True) # dis and dist
```



[54]:
```
vis3 = sns.distplot(df['InternetUsers'], bins= 15) # Bins
```



Univariate Analysis, Bivariate Analysis and Multivariate Analysis

Univariate Analysis

- Univariate analysis focuses on the distribution of a single variable.
- We visualize how values in 'InternetUsers' are distributed.

Bivariate Analysis

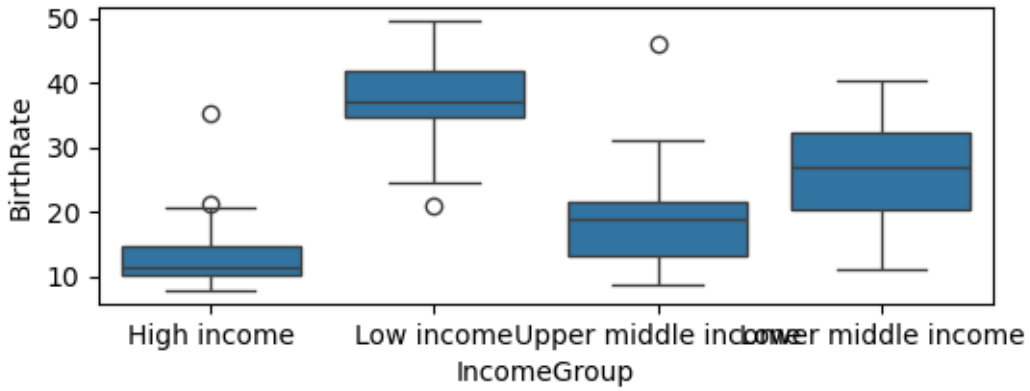
- Bivariate analysis explores the relationship between two variables.
- Let's visualize 'InternetUsers' against another variable, e.g., 'BirthRate'

Multivariate Analysis

- Multivariate analysis explores the relationship between three or more variables.
- We can use scatterplots with an additional dimension shown by color or size.
- Example: 'InternetUsers' vs 'GDP' with 'Population' represented by the size of points.

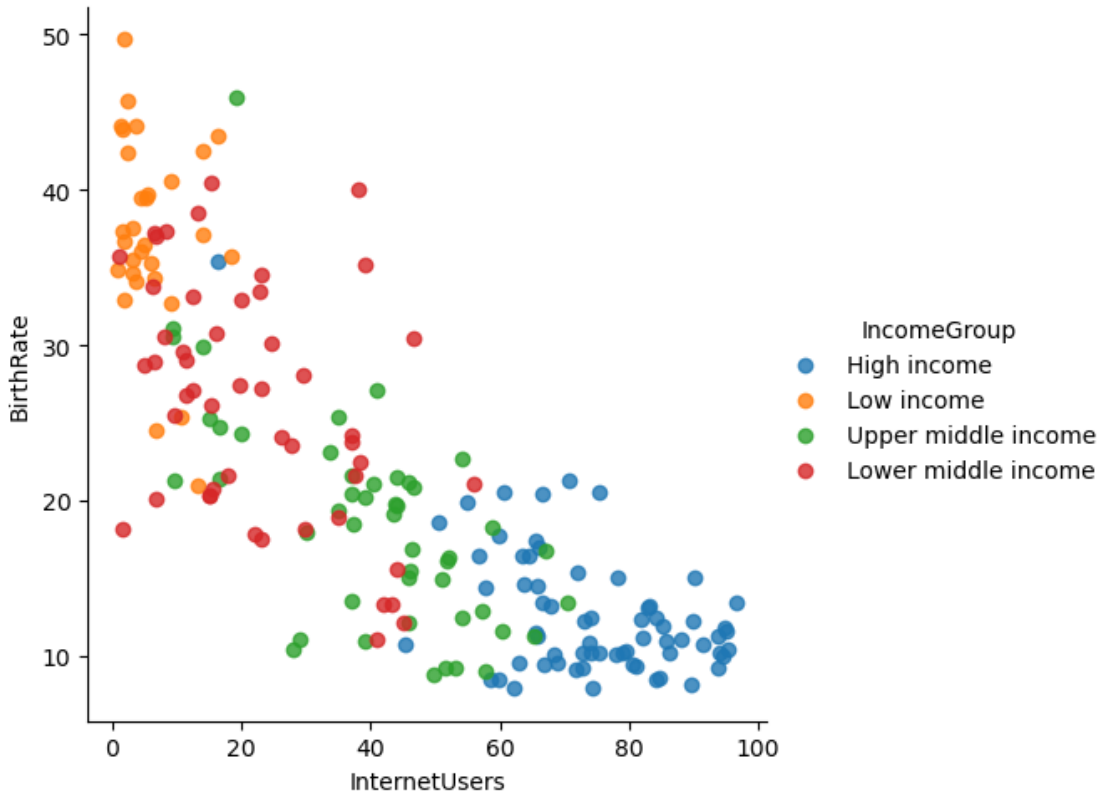
[ ]:
```
# vis4: Boxplot - Univariate/Bivariate Analysis
# This shows the spread of BirthRate in different Income Groups.
# Box shows the middle 50% of the data, line inside the box is the median.
# Helps us compare BirthRates across different income levels.
```

[55]:
```
vis4=sns.boxplot(data=df,x="IncomeGroup",y="BirthRate")
```



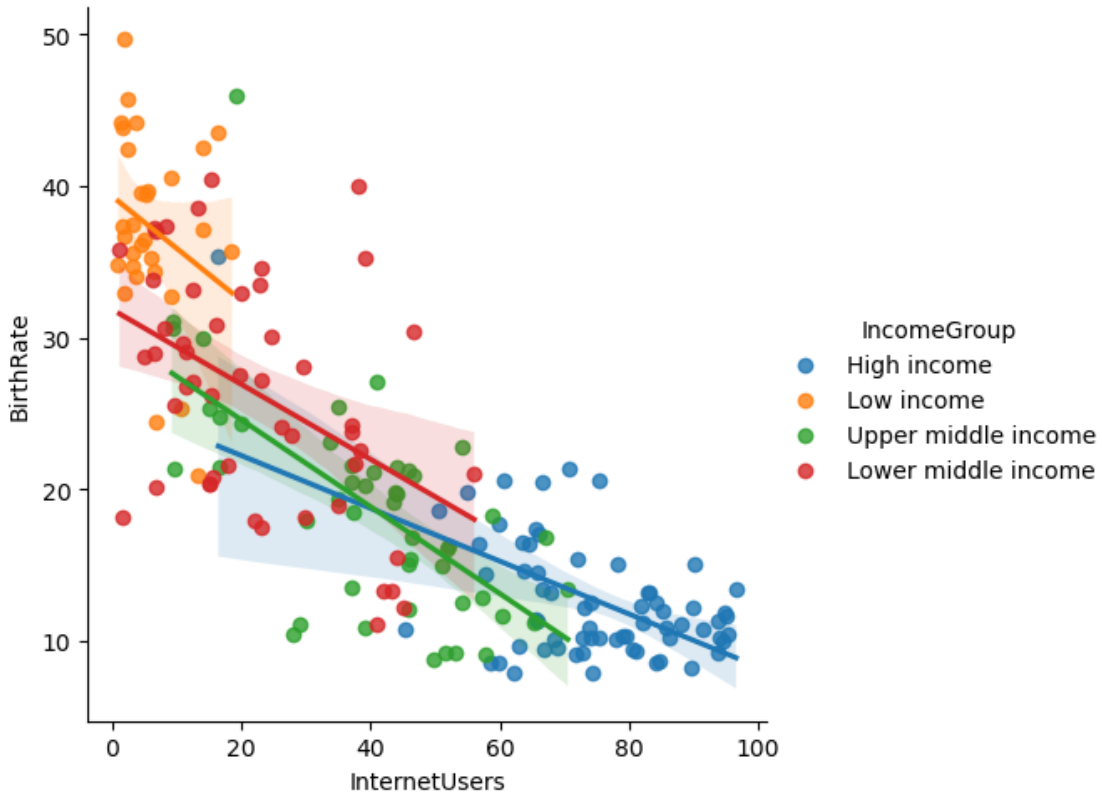
[ ]:
```
# vis5: Scatter Plot (lmplot) - Bivariate Analysis
# This shows a scatter plot between InternetUsers and BirthRate.
# fit_reg=False means do NOT draw a regression (trend) line.
# hue="IncomeGroup" means points are colored based on IncomeGroup.
```

[56]:
```
vis5=sns.
↪lmplot(data=df,x="InternetUsers",y="BirthRate",fit_reg=False,hue="IncomeGroup")
```



[ ]:
```
# vis6: Scatter Plot with Regression Line - Bivariate Analysis
# Same as above, but fit_reg=True adds a regression (trend) line for each␣
↪IncomeGroup.
# This helps see the direction of the relationship (e.g., if InternetUsers ↑,␣
↪does BirthRate ↓?)
```

[58]:
```
vis6=sns.
↪lmplot(data=df,x="InternetUsers",y="BirthRate",fit_reg=True,hue="IncomeGroup")
```



Explanation of Parameters

data=df

- Tells Seaborn which DataFrame to use.

x="InternetUsers" and y="BirthRate" - Sets what to show on the x-axis and y-axis.

hue="IncomeGroup"

- Colors the points by category (like IncomeGroup).
- Helps compare groups easily in one plot.

fit_reg=True or **fit_reg=False**

- `True`: Draws a **regression line** (trend line).
- `False`: No line, just scatter points.

kind="reg" (only for some plots like `lmplot`)

- Means it's a **regression plot** (scatter + line).

[59]:
```
sns.lmplot(data=df,
           x="InternetUsers",
           y="BirthRate",
           hue="IncomeGroup",    # color by group
```

```
            fit_reg=True)        # show trend line
```

[59]: <seaborn.axisgrid.FacetGrid at 0x1caca3efb60>