DATA SOCIETY

Intro to Tableau

Day 4

"One should look for what is and not what he thinks should be."

- Albert Einstein

Warm-up chat question

- Today we will cover a few different types of function, including LOD (level of detail) functions
- Skim through the following blog post: https://www.tableau.com/about/blog/LOD-expressions

How do you use or plan to use LOD functions in your work?

Agenda

- Understand addressing and partitioning fields
- Explore level of detail (LOD) functions
- Implement number and aggregate calculations on given dataset
- Implement string and date calculations on given dataset
- Implement type and logic calculations on given dataset

Reminder: save your work!

- In today's class, we will be creating more elaborate visualizations
- We will see a lot of different insights from the data as we learn more in Tableau
- Make sure to save all your classwork (including Exercises)

Addressing vs. partitioning fields

- The **Specific Dimension** options that we just explored relate to the scope of the data on which the table function is performed and the direction in which the calculation moves through the table

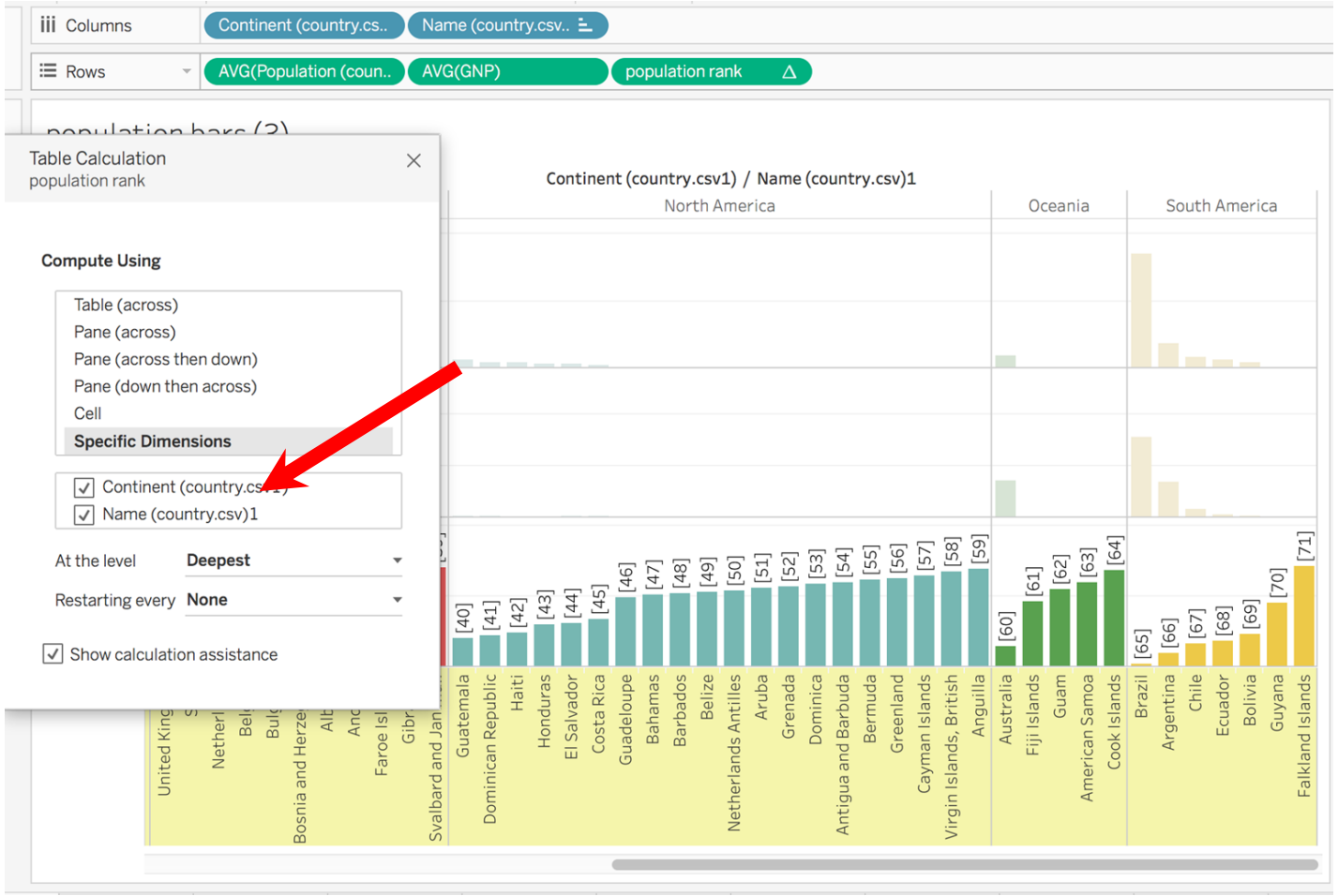
- **Addressing fields** are those fields that define how you are computing through the table – in other words, the ***direction*** of the function

- **Partitioning fields** are how you are dividing the table up – in other words, the ***scope*** of the data the function will address

Addressing fields

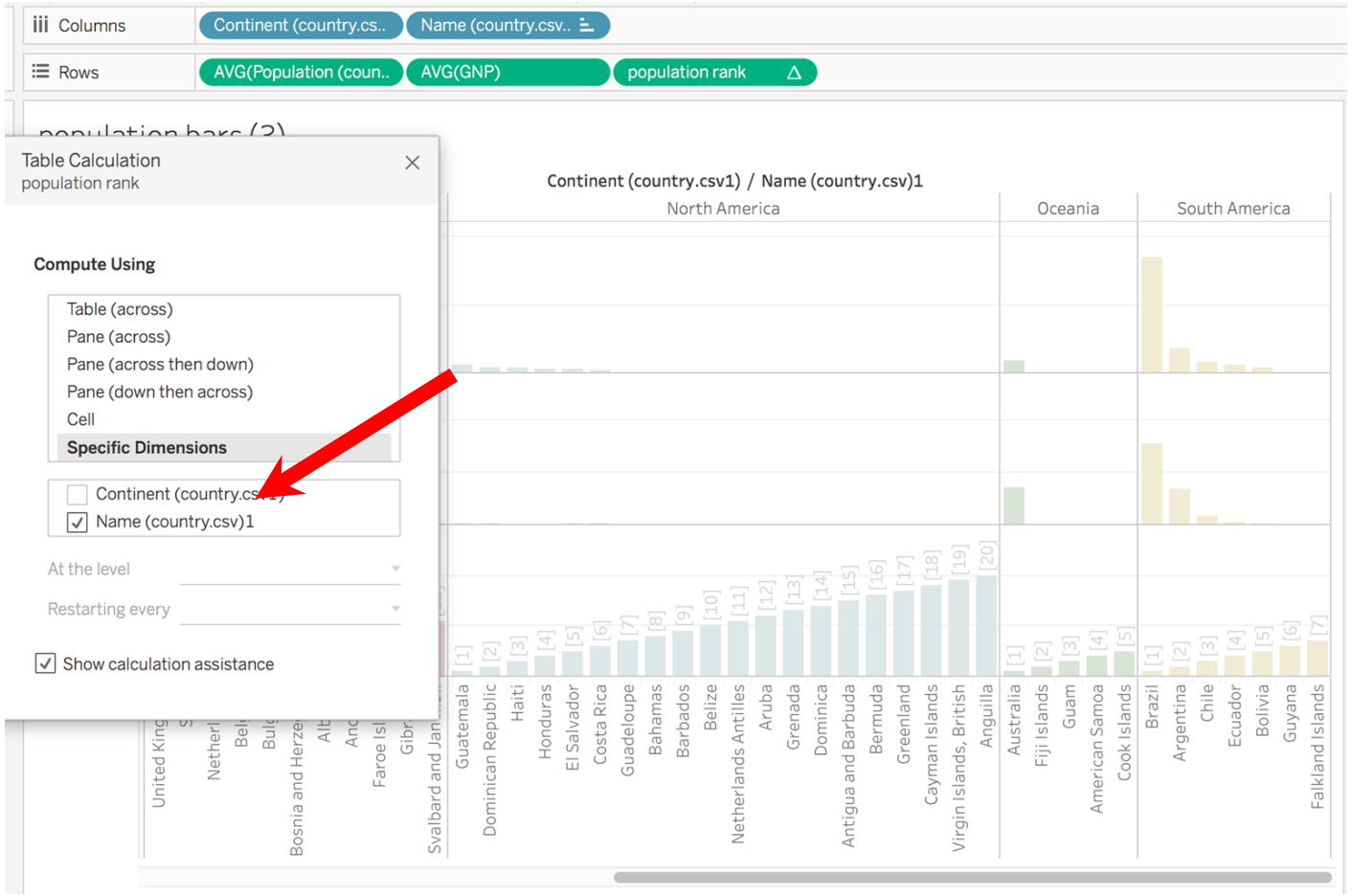
- We can change addressing and partitioning fields in the Edit Table Calculation menu, by selecting Specific Dimensions

- **Continent** is used for **addressing**
  - We see that the rank calculation is table-wide, and that it runs the length of the entire table



Partioning fields

- **Continent** is used for **partitioning**
  - We also see that each continent ranks relative to itself, separating the data into subsets



Agenda

- Understand addressing and partitioning fields
- Explore level of detail (LOD) functions
- Implement number and aggregate calculations on given dataset
- Implement string and date calculations on given dataset
- Implement type and logic calculations on given dataset

Level of Detail (LOD) functions

- **Level of Detail (LOD) functions** give you more control over the level of granularity you want to compute

- In the world dataset we could aggregate population by:
  - City
  - Country
  - Region
  - Continent

Level of Detail (LOD) function syntax

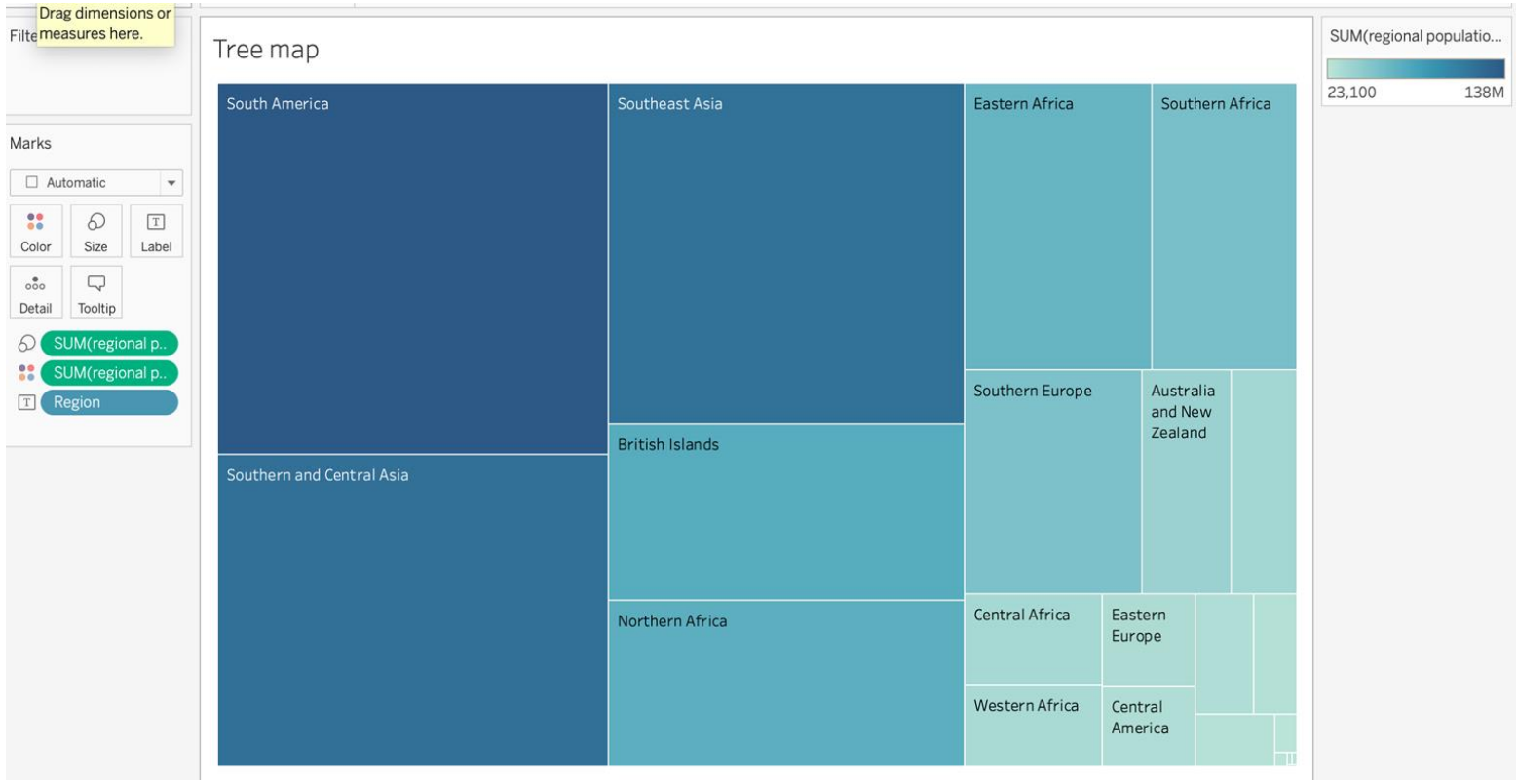
- The syntax for LOD functions has curly braces
  - **{ keyword ([column]): aggregation function ([data column]) }**
- The keyword can be:
  - **FIXED**: takes into account the keyword and the dimensions in the table
  - **INCLUDE**: only takes into account the keyword dimension
  - **EXCLUDE**: ignores specified dimension

Creating a treemap with a LOD function

regional population

```
{ INCLUDE ([Region]): AVG([Population (country.csv)])}
```

- We will now create a treemap with population summarized at the region level
- What does this plot tell you?



Agenda

- Understand addressing and partitioning fields
- Explore level of detail (LOD) functions
- Implement number and aggregate calculations on given dataset
- Implement string and date calculations on given dataset
- Implement type and logic calculations on given dataset

Number functions

- **Number functions** allow you to perform computations on the data values in your fields
- They can only be used with fields that contain numerical values
  - ABS(-7) = 7
  - ABS([Budget Variance])
- We can use a number function to clean up a messy column

GNP Old ▲
Null
Null
Null
Null
Null
Null
Null
Null
Null
243.00
272.00
325.00
325.00
373.00
383.00
573.00

GNP_old_clean

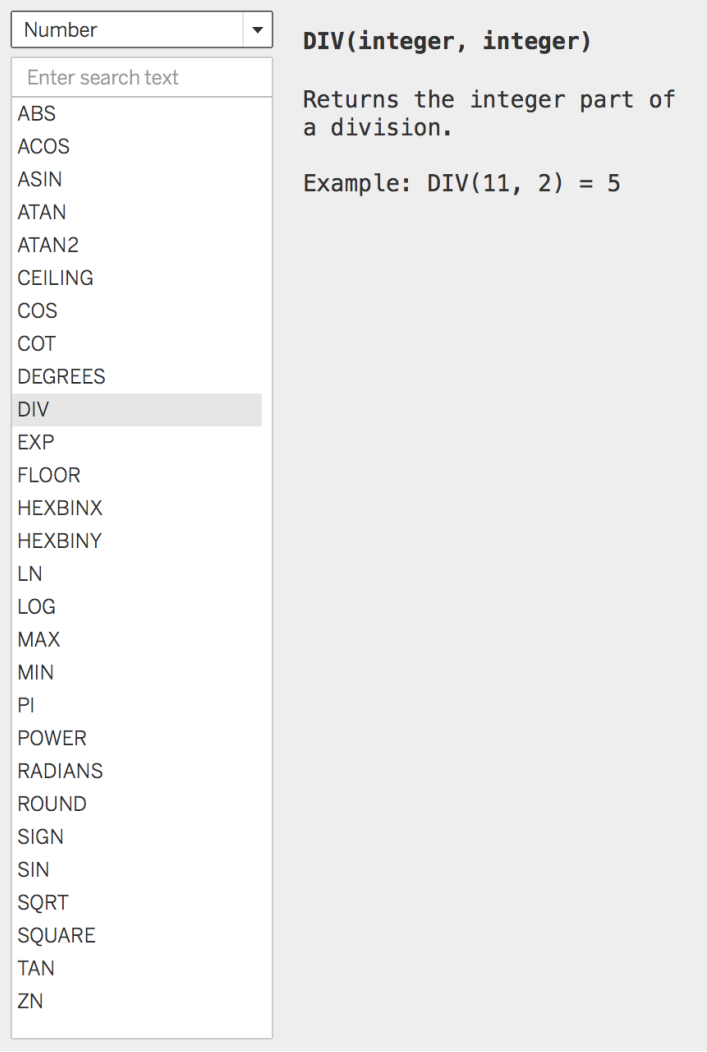
```
CEILING(ZN([GNP Old]))
```

GNP Old	GNP_old_clean
Null	0
Null	0
Null	0
Null	0
360,478.00	360,478
360,478.00	360,478
360,478.00	360,478
360,478.00	360,478
360,478.00	360,478
360,478.00	360,478

Number functions

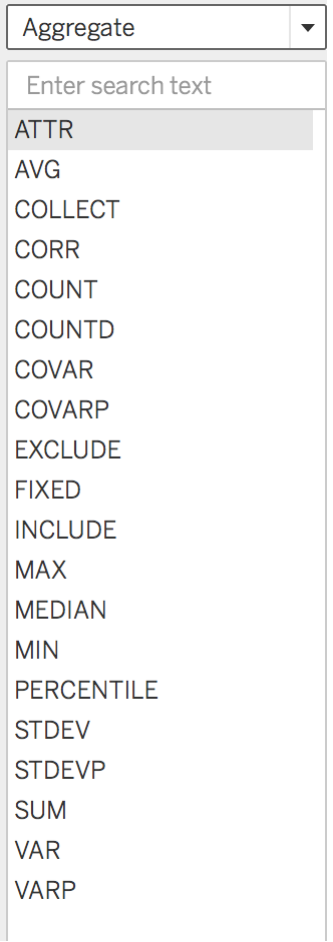
- Normal statistical summary and math functions are also available

- Notable functions include
  - Trigonometric functions
  - RADIANS
  - DIV for integer division
  - FLOOR and CEIL



Aggregate functions

- Aggregations involve a summary function, like **SUM()** or **AVG()**
- The resulting function will have key built-in functions **AGG()** and /or **ATTR()**
- These allow the user to conduct operations at a particular granularity
  - Granularity is controlled by an **attribute** such as year or continent

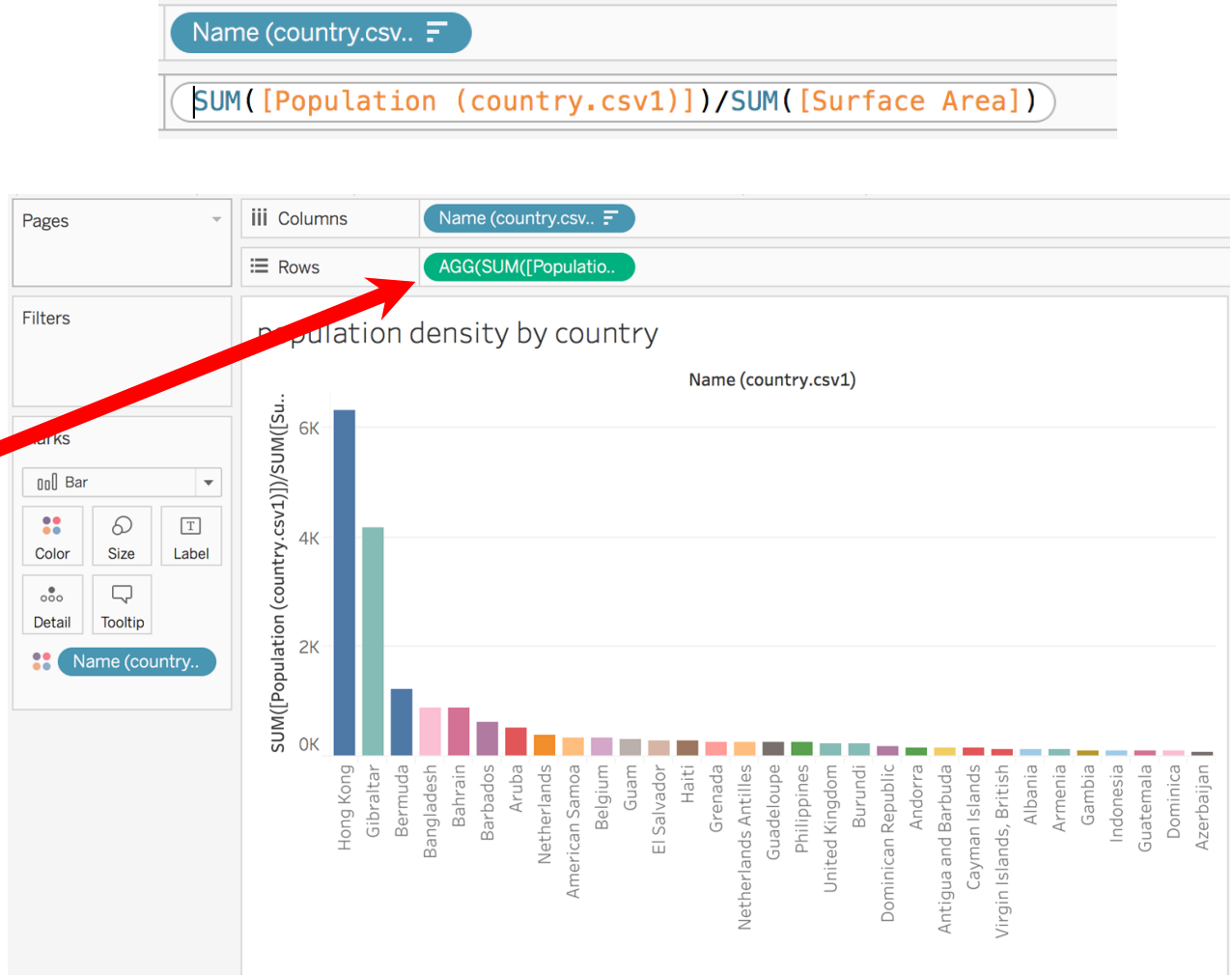


Aggregate functions

- We can use **AGG()** to get population density by country
  - Divide Population/area
  - Aggregate at country level

- Note that the aggregate formula was added directly to the pill

- Can you think of another aggregation we can apply to our data?



Knowledge check 1

Exercise 1

Agenda

- Understand addressing and partitioning fields
- Explore level of detail (LOD) functions
- Implement number and aggregate calculations on given dataset
- Implement string and date calculations on given dataset
- Implement type and logic calculations on given dataset

String functions

- **String functions** allow you to manipulate string data (i.e. text data)

- Using string functions, you can perform a lot of common operations:
  - capture part of a field
  - recast part of a string as an int
  - clean messy data
  - remove extraneous fields

- Notable functions include **MID**, **REPLACE**, **and TRIM**, which we will practice using on the **country data** today

String function: MID

- **MID**
  - Returns the string starting at a particular **position**
  - If the optional argument **length** is added, the returned string includes only that number of characters

- Example:

```
MID("Calculation", 2) =

"alculation"

MID("Calculation", 2, 5) ="alcul"
```

String function: REPLACE

- **REPLACE**
  - Searches a string for a substring and replaces it with a replacement substring

- Example:

```
REPLACE("Version8.5", "8.5",
"9.0") = "Version9.0"
```

String function: TRIM

- **TRIM**
  - Returns the string with leading and trailing spaces removed

- Example:

```
TRIM(" Calculation ") =
"Calculation"
```

Using string functions to clean data

- In our data, **local country name** has characters that might cause problems
- For example **spaces**, ‘, and **/** can be problematic since they can break up strings or cannot be interpreted in some programs
- Let’s make a column with the local names after data cleaning

Local Name	LocalName (country.csv1)
Al-Jaza'ir/Algérie	Al-Jaza'ir/Algérie
Al-Jaza'ir/Algérie	Al-Jaza'ir/Algérie
Al-Jaza'ir/Algérie	Al-Jaza'ir/Algérie
Amerika Samoa	Amerika Samoa
Amerika Samoa	Amerika Samoa
Andorra	Andorra
Angola	Angola
Angola	Angola
Angola	Angola
Angola	Angola
Angola	Angola
Anguilla	Anguilla
Anguilla	Anguilla
Antigua and Barbuda	Antigua and Barbuda
Al-Imarat al-´Arabiya al-Muttahida	Al-Imarat al-´Arabiya al-Muttahida
Al-Imarat al-´Arabiya al-Muttahida	Al-Imarat al-´Arabiya al-Muttahida
Al-Imarat al-´Arabiya al-Muttahida	Al-Imarat al-´Arabiya al-Muttahida
Al-Imarat al-´Arabiya al-Muttahida	Al-Imarat al-´Arabiya al-Muttahida
Al-Imarat al-´Arabiya al-Muttahida	Al-Imarat al-´Arabiya al-Muttahida
Argentina	Argentina
Argentina	Argentina
Argentina	Argentina
Argentina	Argentina
Argentina	Argentina

Nesting string functions

- We can simplify the process by using nested replacements

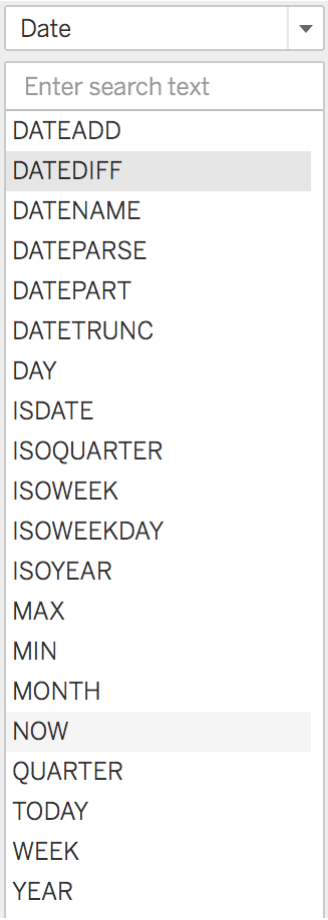
Localnames_clean

```
trim(REPLACE(REPLACE([LocalName (country.csv1)]," ","_"),"/","_"))
```

LocalName (country.csv1)	Localnames_clean
Al-Jaza'ir/Algérie	Al-Jaza'ir_Algérie
Al-Jaza'ir/Algérie	Al-Jaza'ir_Algérie
Al-Jaza'ir/Algérie	Al-Jaza'ir_Algérie
Amerika Samoa	Amerika_Samoa
Amerika Samoa	Amerika_Samoa
Andorra	Andorra
Angola	Angola
Angola	Angola
Angola	Angola
Angola	Angola
Angola	Angola
Anguilla	Anguilla
Anguilla	Anguilla
Antigua and Barbuda	Antigua_and_Barbuda

Date functions

- **Date functions** allow you to manipulate dates and extract certain features, like month, day, week, etc.



Date function: DATENAME

- **DATENAME** takes two arguments, returning a requested **date_part** for a given **date**
  - DATENAME('year', #2004-04-15#) = **"2004"**
  - DATENAME('month', #2004-04-15#) = **"April"**

DATEPARSE and DATEPART

- **DATEPARSE**
  - Allows you to control date formatting

```
Example: DATEPARSE
("dd.MMMM.yyyy", "15.April.
2004") = 2004-04-15 12:00:00
AM
```

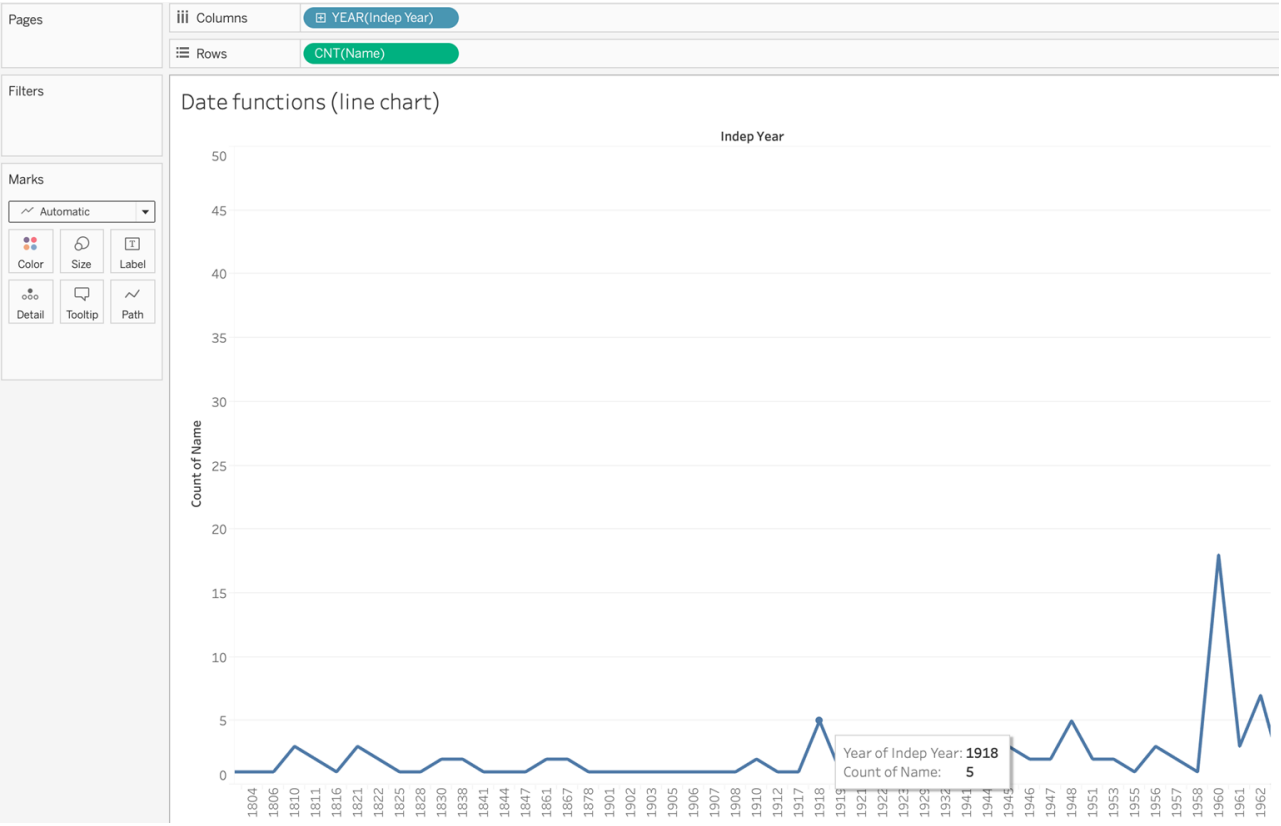
- **DATEPART**
  - Allows you to get a piece of a date

```
Example: DATEPART('month',
#2004-04-15#) = 4
```

Both **DATENAME** and **DATEPART** can be used to return the month of a given date – so what's the difference?

Setting up a date-related chart

- To practice using these functions, let's use the world data to plot a **line chart** showing the number of countries that became independent in a given year



Date functions in the world dataset

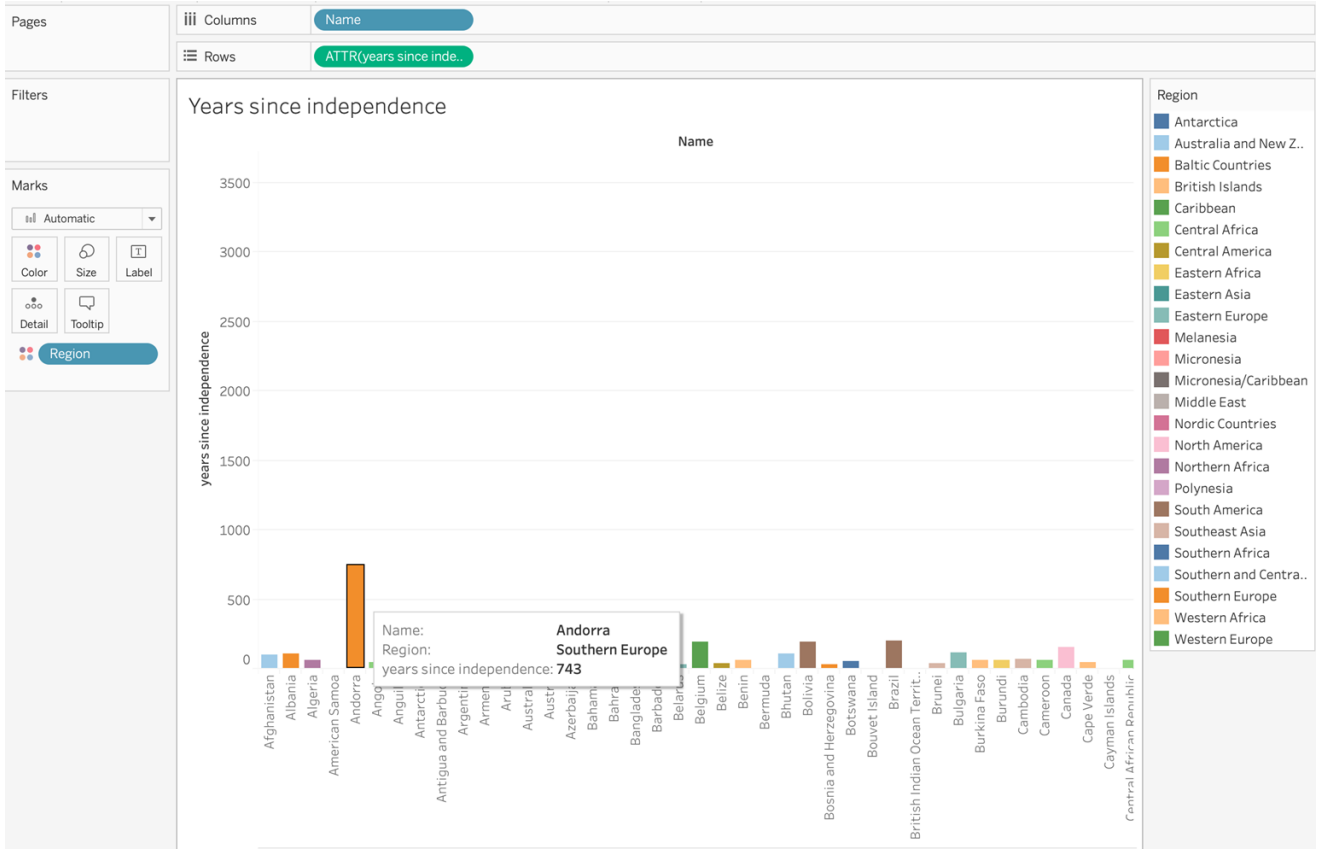
- Get the years since independence for a country using these functions:
  - TODAY
  - YEAR

- Next, replot the independence
  - graph as a bar chart

- What do you notice?

years since independence

```
ZN(DATEDIFF("year", [Indep Year], TODAY(), "sunday"))
```

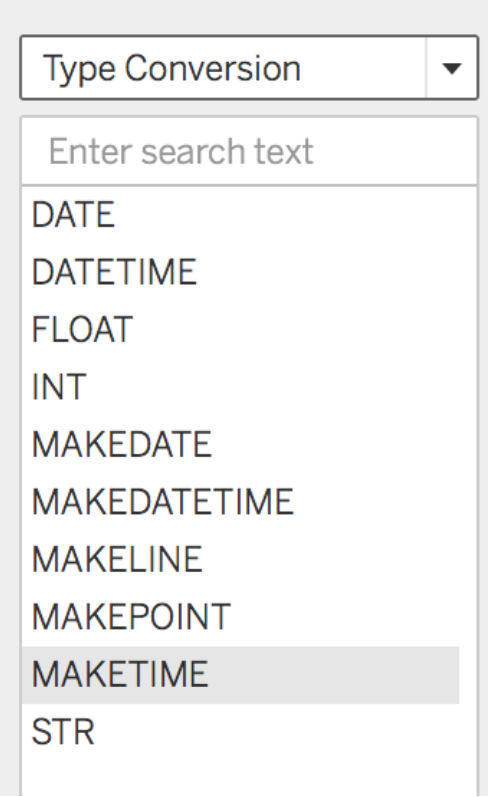


Agenda

- Understand addressing and partitioning fields
- Explore level of detail (LOD) functions
- Implement number and aggregate calculations on given dataset
- Implement string and date calculations on given dataset
- Implement type and logic calculations on given dataset

Type functions

- **Type functions** allow you to convert fields from one data type to another
- For example, you can convert **numbers to strings** so that Tableau does not try to aggregate them



Typecasting functions

- **Typecasting functions** are type functions that convert one data type to another

- **STR**

```
STR([Age])
```

- **INT**

```
INT(8.0/3.0) = 2
INT(4.0/1.5) = 2
INT(0.50/1.0) = 0
INT(-9.7) = -9
```

- **FLOAT**

```
FLOAT(3) = 3.000
```

MAKE functions

- **MAKE functions** return date and time values given certain arguments
- **MAKEDATE** returns a time value constructed from the specified **year**, **month**, and **date**
- **MAKEDATETIME** returns a datetime that combines a date and a time
  - The **date** can be a date, datetime, or a string type
  - The **time** must be a datetime

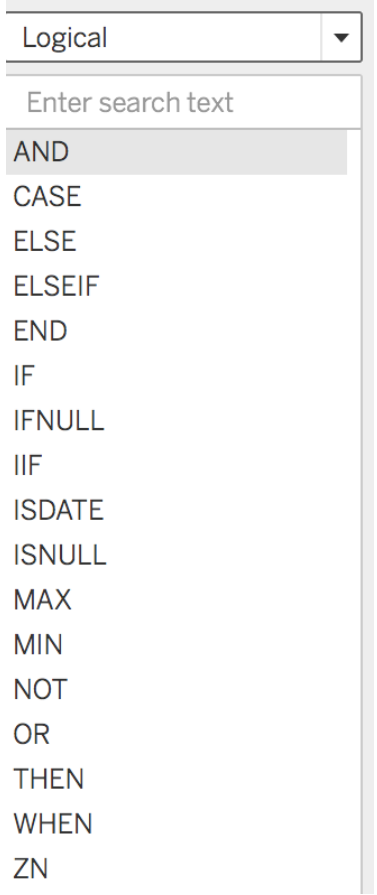
```
MAKEDATE(2004, 4, 15) = #April
15, 2004#
```

```
MAKEDATETIME("1899-12-30",
#07:59:00#) = #12/30/1899 7:59:00
AM#
MAKEDATETIME([Date], [Time]) =
#1/1/2001 6:00:00 AM#
```

Logical functions

- **Logical functions** allow you to determine if a certain condition is **true or false** (Boolean logic)

- Tableau offers all the basic logical functions for managing **control flow** and performing **Boolean tests**



Case switching and conditionals

- **Case switch functions :**
  - CASE , WHEN, THEN, ELSE , END

```
Example: CASE [RomanNumeral]
WHEN ‘I’ THEN 1 WHEN ‘II’
THEN 2 ELSE 3 END
```

- **IF, ELSE conditionals:**
  - IF , THEN, ELSEIF, THEN, END

```
Example: IF [Profit] > 0
THEN 'Profitable' ELSEIF
[Profit] = 0 THEN
'Breakeven' ELSE 'Loss' END
```

IIF test

- **IIF** checks whether a condition is met, and then returns:
  - One value if TRUE
  - Another value if FALSE
  - An optional third value or NULL if unknown

```
IIF(test, then, else,
[unknown])

Checks whether a condition
is met, and returns one
value if TRUE, another value
if FALSE, and an optional
third value or NULL if
unknown.

Example: IIF([Profit] > 0,
'Profit', 'Loss')
```

Logical functions in the world dataset

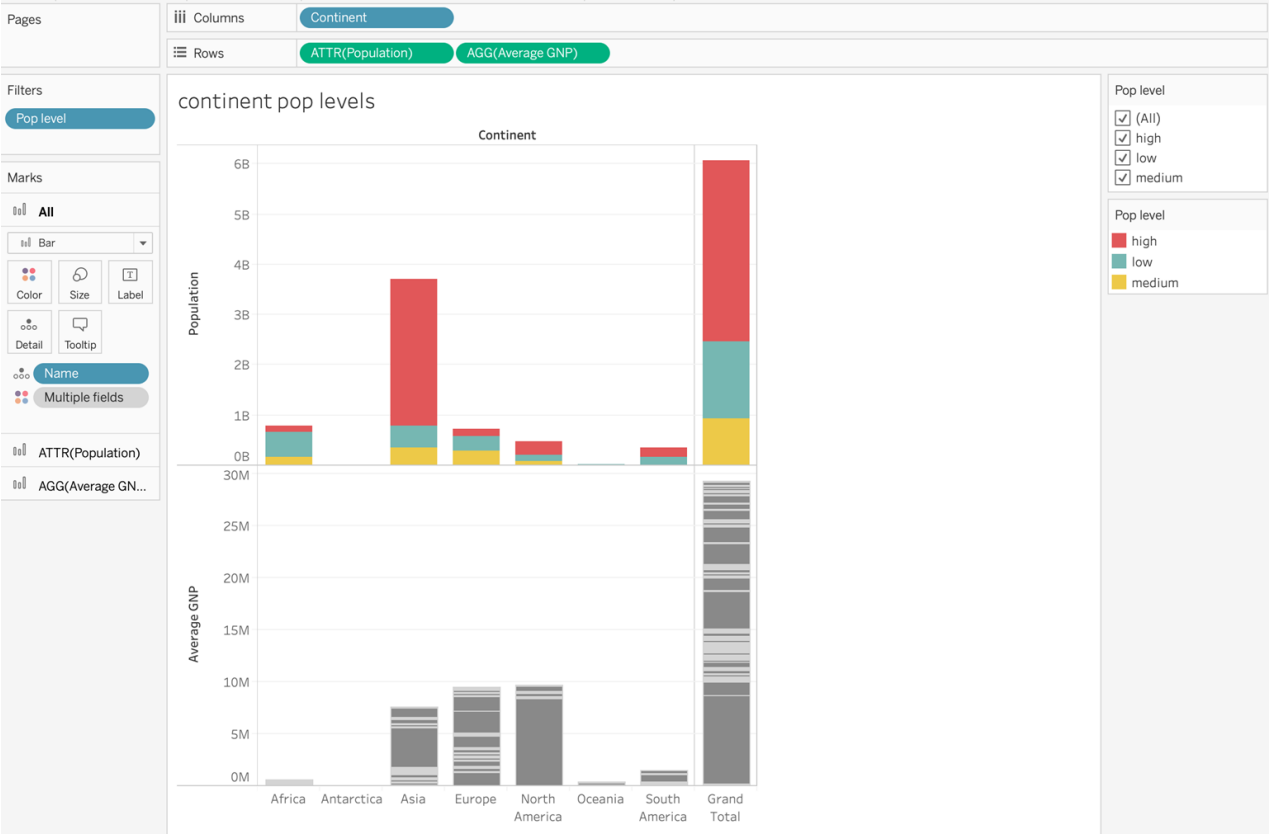
- Use an **IF ELSE** conditional to make a "high", "medium" and "low" population column

Pop Level

```
IF [Population (country.csv)] > 100000000
THEN "high"
ELSEIF [Population (country.csv)] > 50000000
THEN "medium"
ELSE "low"
END
```

The calculation is valid. 1 Dependency Apply OK

- Apply it to the population analysis



Knowledge check 2

Exercise 2

What we covered today

- Level of Detail functions
- Number functions
- Aggregate functions
- String Functions
- Date Functions
- Type Functions
- Logical Functions

DATASOCIETY:

Thank you!