

# Exploratory data analysis on Azure Databricks

**Name:** PRIYESHWAR

**Mail:** [priyesh2664@gmail.com](mailto:priyesh2664@gmail.com)

---

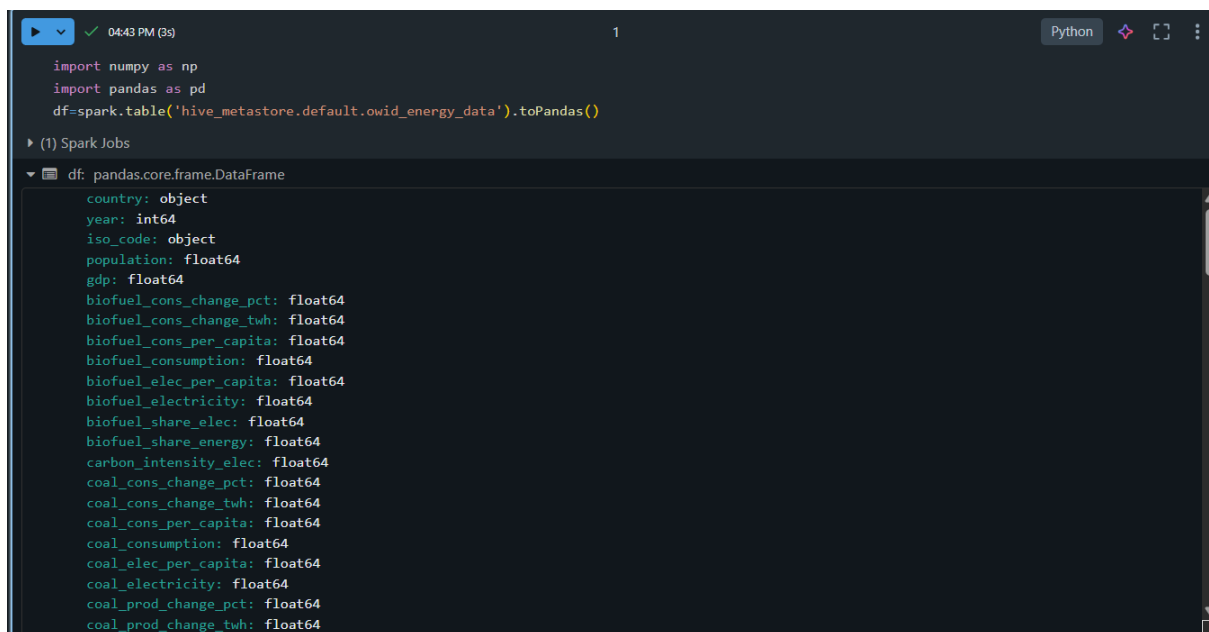
## Load CSV file

Create a pandas DataFrame from the dataset for easier processing and visualization. Replace the file path below with the one you copied earlier.

```
import numpy as np
```

```
import pandas as pd
```

```
df=spark.table('hive_metastore.default.owid_energy_data').toPandas()
```



The screenshot shows a Jupyter Notebook interface with a Python kernel. The code cell contains the following code:

```
import numpy as np
import pandas as pd
df=spark.table('hive_metastore.default.owid_energy_data').toPandas()
```

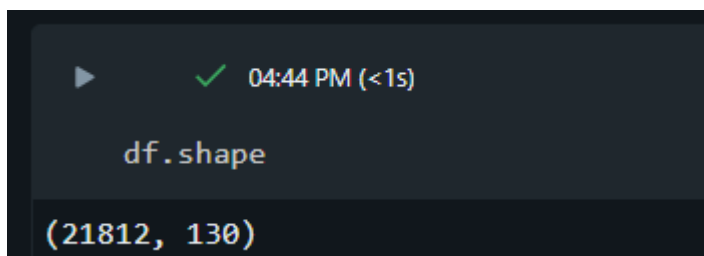
Below the code cell, the output shows the Spark job status and the resulting pandas DataFrame structure:

```
(1) Spark Jobs
df: pandas.core.frame.DataFrame
country: object
year: int64
iso_code: object
population: float64
gdp: float64
biofuel_cons_change_pct: float64
biofuel_cons_change_twh: float64
biofuel_cons_per_capita: float64
biofuel_consumption: float64
biofuel_elec_per_capita: float64
biofuel_electricity: float64
biofuel_share_elec: float64
biofuel_share_energy: float64
carbon_intensity_elec: float64
coal_cons_change_pct: float64
coal_cons_change_twh: float64
coal_cons_per_capita: float64
coal_consumption: float64
coal_elec_per_capita: float64
coal_electricity: float64
coal_prod_change_pct: float64
coal_prod_change_twh: float64
```

## Use pandas for data insights

The `df.shape` command returns the dimensions of the DataFrame, giving you a quick overview of the number of rows and columns.

**df.shape**



The screenshot shows a Jupyter Notebook interface with a Python kernel. The code cell contains the following code:

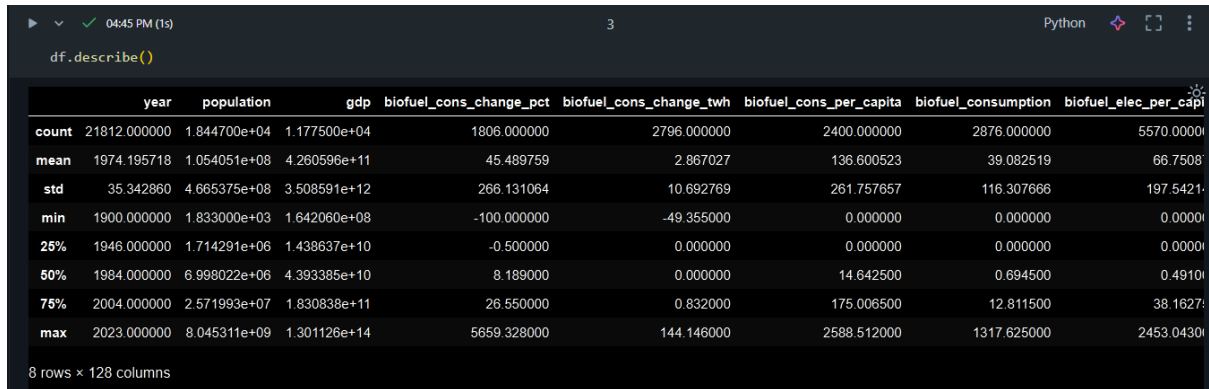
```
df.shape
```

Below the code cell, the output shows the dimensions of the DataFrame:

```
(21812, 130)
```

## df.describe()

The `df.describe()` command generates descriptive statistics for numerical columns, such as mean, standard deviation, and percentiles, which can help you identify patterns, detect anomalies, and understand the distribution of your data.

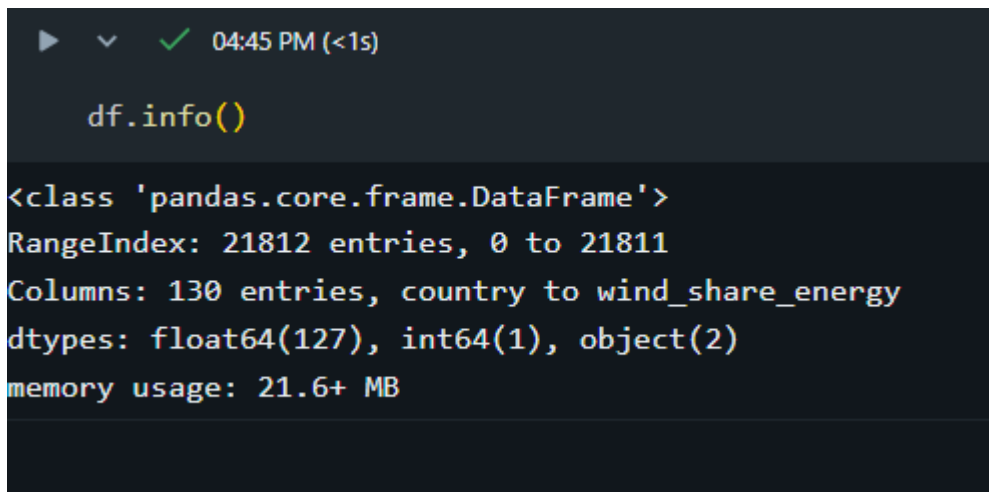


```
df.describe()
```

	year	population	gdp	biofuel_cons_change_pct	biofuel_cons_change_twh	biofuel_cons_per_capita	biofuel_consumption	biofuel_elec_per_capita
count	21812.000000	1.844700e+04	1.177500e+04	1806.000000	2796.000000	2400.000000	2876.000000	5570.000000
mean	1974.195718	1.054051e+08	4.260596e+11	45.489759	2.867027	136.600523	39.082519	66.7508
std	35.342860	4.665375e+08	3.508591e+12	266.131064	10.692769	261.757657	116.307666	197.5421
min	1900.000000	1.833000e+03	1.642060e+08	-100.000000	-49.355000	0.000000	0.000000	0.000000
25%	1946.000000	1.714291e+06	1.438637e+10	-0.500000	0.000000	0.000000	0.000000	0.000000
50%	1984.000000	6.998022e+06	4.393385e+10	8.189000	0.000000	14.642500	0.694500	0.491000
75%	2004.000000	2.571993e+07	1.830838e+11	26.550000	0.832000	175.006500	12.811500	38.162700
max	2023.000000	8.045311e+09	1.301126e+14	5659.328000	144.146000	2588.512000	1317.625000	2453.043000

8 rows x 128 columns

## df.info()



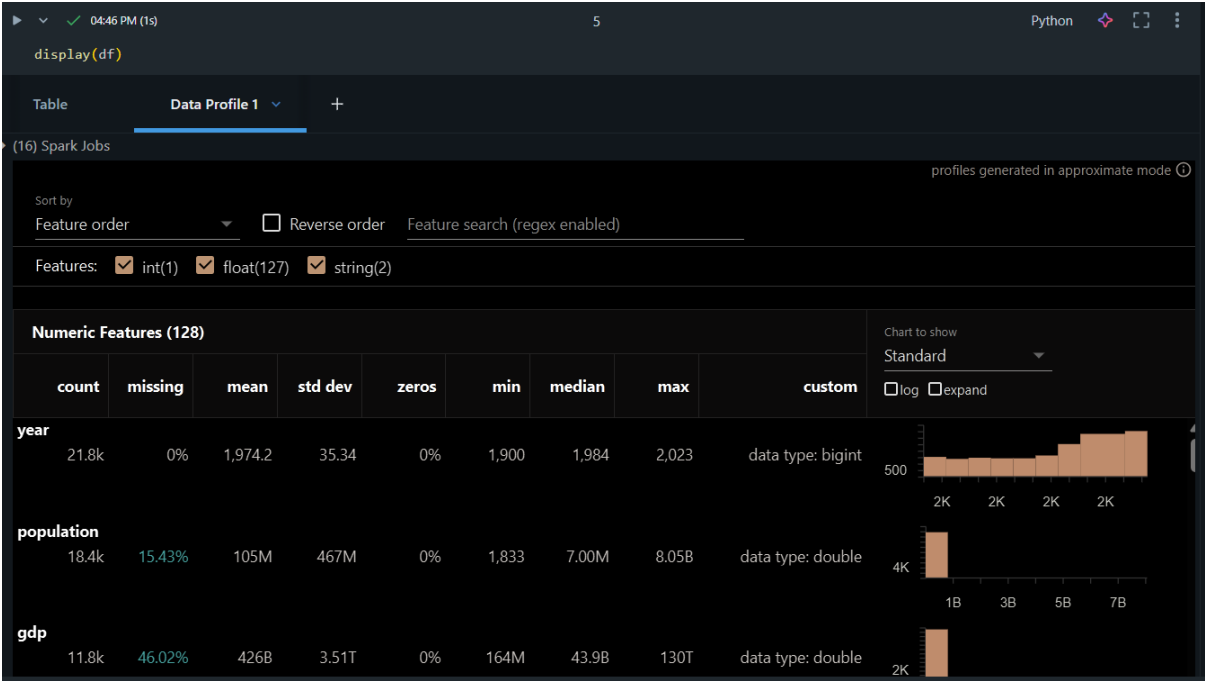
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21812 entries, 0 to 21811
Columns: 130 entries, country to wind_share_energy
dtypes: float64(127), int64(1), object(2)
memory usage: 21.6+ MB
```

Generate a data profile

Click + > Data Profile next to the Table in the output. This runs a new command that generates a profile of the data in the DataFrame.

`display(df)`

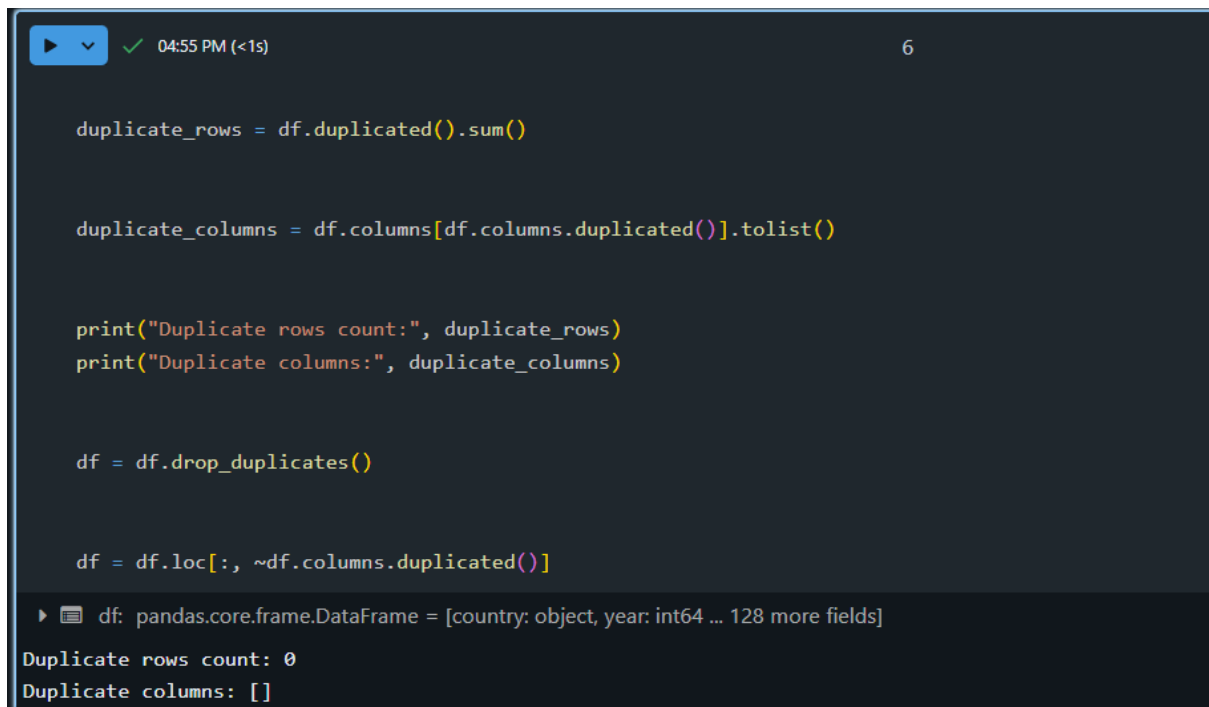


## Clean the data

### Remove duplicate data

Check if the data has any duplicate rows or columns. If so, remove them.

```
duplicate_rows = df.duplicated().sum()
duplicate_columns = df.columns[df.columns.duplicated()].tolist()
print("Duplicate rows count:", duplicate_rows)
print("Duplicate columns:", duplicate_columns)
df = df.drop_duplicates()
df = df.loc[:, ~df.columns.duplicated()]
```

A screenshot of a Jupyter Notebook cell. The cell contains Python code to identify and remove duplicate rows and columns from a DataFrame. The code uses the `duplicated()` method to find duplicates, `sum()` to count them, and `drop_duplicates()` to remove them. It also uses `loc` to drop duplicate columns. The output shows that there are 0 duplicate rows and an empty list for duplicate columns. The top bar of the notebook shows a play button, a green checkmark, the time 04:55 PM (<1s), and the cell number 6.

```
duplicate_rows = df.duplicated().sum()

duplicate_columns = df.columns[df.columns.duplicated()].tolist()

print("Duplicate rows count:", duplicate_rows)
print("Duplicate columns:", duplicate_columns)

df = df.drop_duplicates()

df = df.loc[:, ~df.columns.duplicated()]

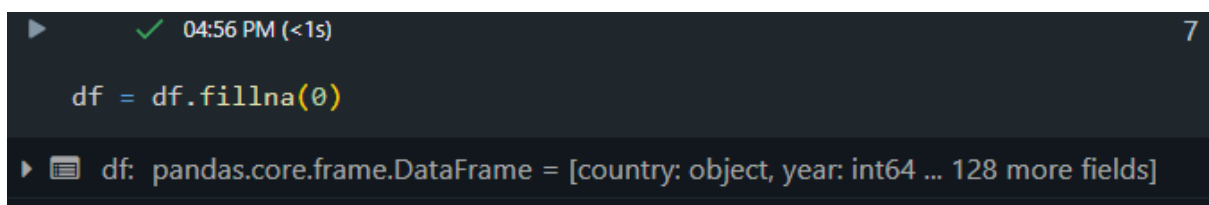
df: pandas.core.frame.DataFrame = [country: object, year: int64 ... 128 more fields]
Duplicate rows count: 0
Duplicate columns: []
```

---

### Handle null or missing values

A common way to treat NaN or Null values is to replace them with 0 for easier mathematical processing.

```
df = df.fillna(0)
```

A screenshot of a Jupyter Notebook cell. The cell contains a single line of Python code: `df = df.fillna(0)`. The output shows the DataFrame object. The top bar of the notebook shows a play button, a green checkmark, the time 04:56 PM (<1s), and the cell number 7.

```
df = df.fillna(0)

df: pandas.core.frame.DataFrame = [country: object, year: int64 ... 128 more fields]
```

## Reformat dates

Dates are often formatted in various ways in different datasets. They might be in date format, strings, or integers.

```
df['year'] = pd.to_datetime(df['year'], format='%Y', errors='coerce').dt.year
```

```
df.year.dtype
```

```
display(df)
```

```
df['year'] = pd.to_datetime(df['year'], format='%Y', errors='coerce').dt.year
|
df.year.dtype
display(df)
```

▸ (2) Spark Jobs

/databricks/spark/python/pyspark/sql/pandas/conversion.py:516: UserWarning: createDataFrame attempted Arrow optimization because 'spark.sql.execution.arrow.pyspark.enabled' is set to true; however, failed by the reason below:  
Expected bytes, got a 'int' object  
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is set to true.  
warn(msg)

	country	year	iso_code	population	gdp	biofuel_cons_change_pct	biofuel_cons_change_twh
60	India	2011	IND	1257621248	> 5720693...	113.681	1.698
61	India	2012	IND	1274487168	> 6052427...	-7.678	-0.245
62	India	2013	IND	1291132032	> 6424470...	16.383	0.483
63	India	2014	IND	1307246464	> 6875186...	-12.718	-0.436
64	India	2015	IND	1322866560	> 7391521...	20.01	0.599
65	India	2016	IND	1338636288	> 8057466...	111.333	3.999
66	India	2017	IND	1354195712	> 8550029...	-27.967	-2.123
67	India	2018	IND	1369003264	> 9173200...	92.854	5.077
68	India	2019	IND	1383112064	> 9595463...	23.248	2.451
69	India	2020	IND	1396387072	> 8945313...	-6.26	-0.813

## Filter for specific conditions

You can use built-in table filters to filter your columns for specific conditions. There are several ways to create a filter.

```
df['year'] = pd.to_datetime(df['year'], format='%Y', errors='coerce').dt.year
|
df.year.dtype
display(df)
```

▸ (2) Spark Jobs

/databricks/spark/python/pyspark/sql/pandas/conversion.py:516: UserWarning: createDataFrame attempted Arrow optimization because 'spark.sql.execution.arrow.pyspark.enabled' is set to true; however, failed by the reason below:  
Expected bytes, got a 'int' object  
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is set to true.  
warn(msg)

	country	year	iso_code	population	gdp	biofuel_cons_change_pct	biofuel_cons_change_twh
60	India	2011	IND	1257621248	> 5720693...	113.681	1.698
61	India	2012	IND	1274487168	> 6052427...	-7.678	-0.245
62	India	2013	IND	1291132032	> 6424470...	16.383	0.483
63	India	2014	IND	1307246464	> 6875186...	-12.718	-0.436
64	India	2015	IND	1322866560	> 7391521...	20.01	0.599
65	India	2016	IND	1338636288	> 8057466...	111.333	3.999
66	India	2017	IND	1354195712	> 8550029...	-27.967	-2.123
67	India	2018	IND	1369003264	> 9173200...	92.854	5.077
68	India	2019	IND	1383112064	> 9595463...	23.248	2.451
69	India	2020	IND	1396387072	> 8945313...	-6.26	-0.813

## Create visualizations using the dataset

At the top of the output table, click + > **Visualization** to open the visualization editor.

05:14 PM (14s) 9 Python

```
display(df)
```

(2) Spark Jobs

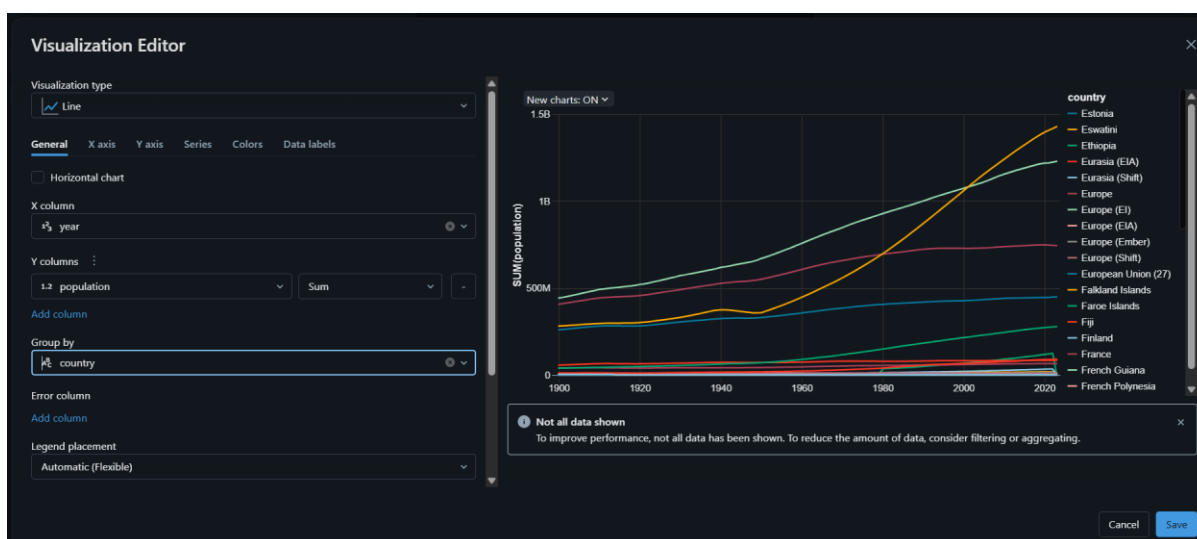
/databricks/spark/python/pyspark/sql/pandas/conversion.py:516: UserWarning: createDataFrame attempted Arrow optimization because 'spark.sql.execution.arrow.pyspark.enabled' is set to true; however, failed by the reason below:  
Expected bytes, got a 'int' object  
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is set to true.  
warn(msg)

Table		+						
	country	Visualization		1.2 population	1.2 gdp	1.2 biofuel_cons_change_pct	1.2 biofuel_cons_change_twh	1.2 biofuel_cons_change_pct
1	Estonia	Data Profile		1322682	30160490496	-14.179	-0.007	
2	Estonia	2013	EST	1317985	30600421376	-16.667	-0.007	
3	Estonia	2014	EST	1314531	31521830912	60	0.022	
4	Estonia	2015	EST	1314658	32106016768	-50	-0.03	
5	Estonia	2016	EST	1315928	33119084544	-25	-0.007	
6	Estonia	2017	EST	1317550	35037376512	28.694	0.006	
7	Estonia	2018	EST	1322146	36363239424	593.902	0.171	
8	Estonia	2019	EST	1327039	37723176960	50.869	0.101	
9	Estonia	2020	EST	1329449	37515272192	40.117	0.121	
10	Estonia	2021	EST	1328704	40521482240	14.439	0.061	
11	Estonia	2022	EST	1326064	39999856640	-42.598	-0.206	
12	Estonia	2023	EST	1322764	0	8.177	0.023	
13	Eswatini	1900	SWZ	109096	0	0	0	
14	Eswatini	1900	SWZ	109096	0	0	0	

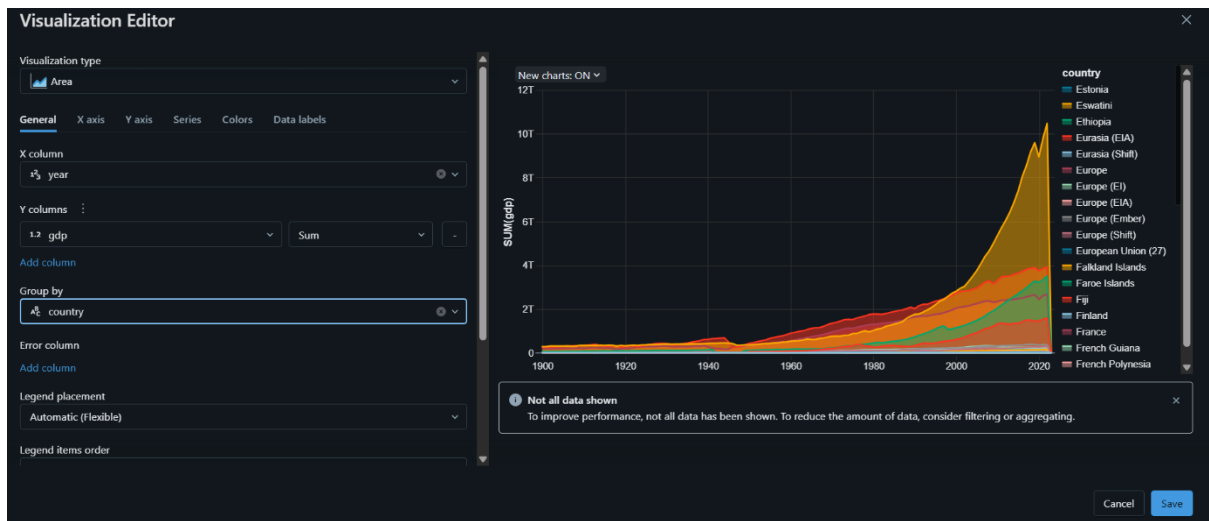
## Visualization

Select the visualization type and columns you'd like to visualize. The editor displays a preview of the chart based on your configuration

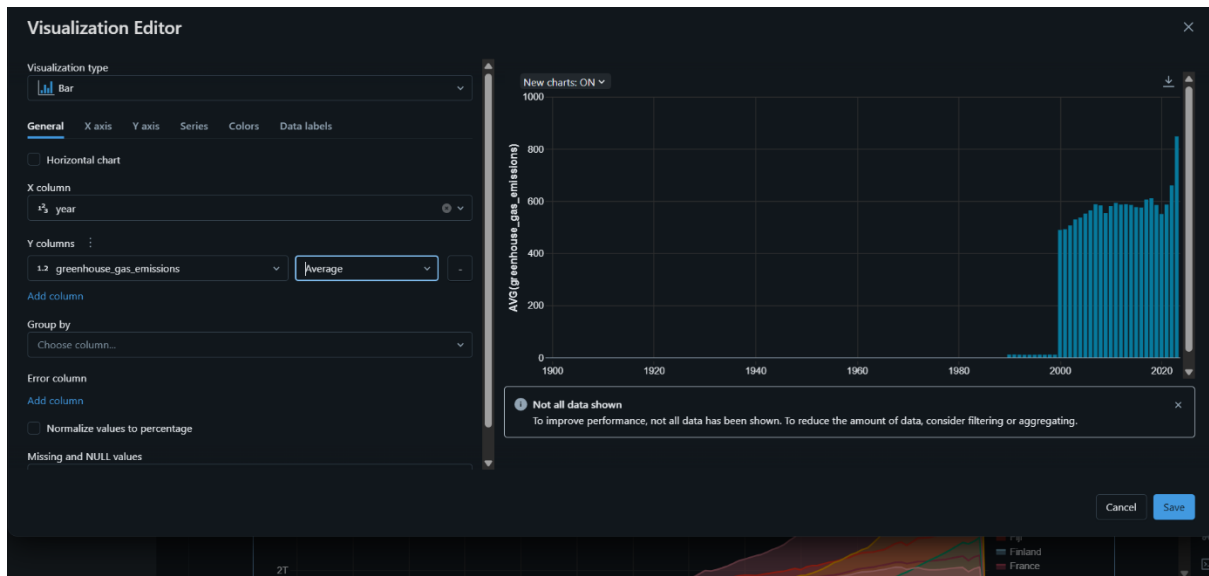
### Line Chart



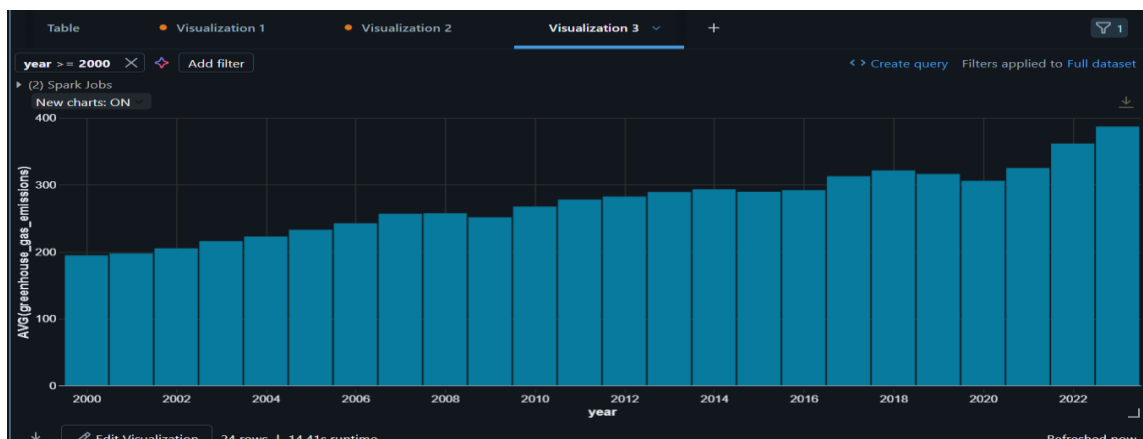
## Area Chart



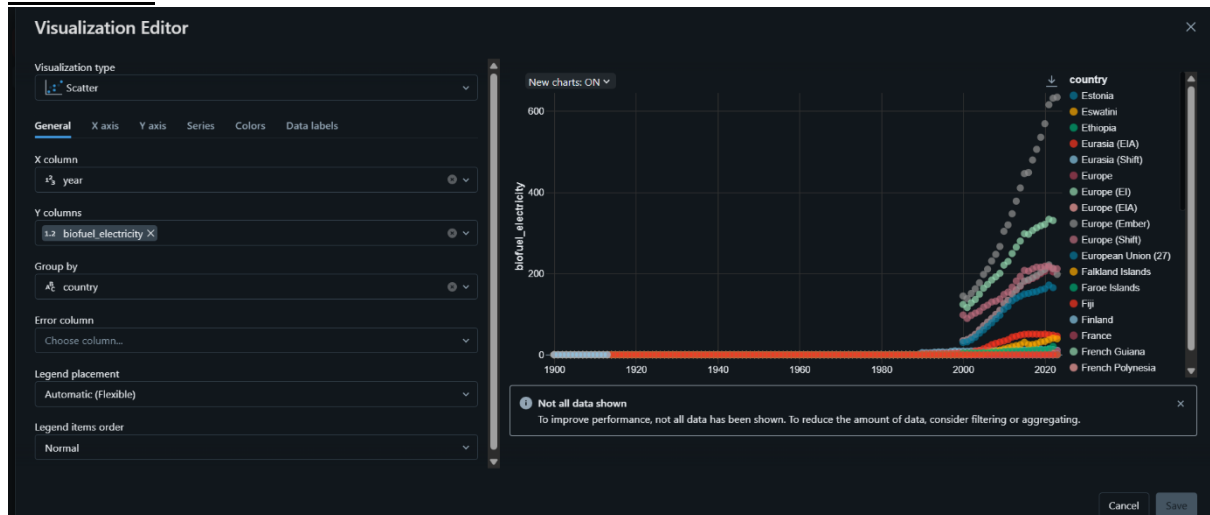
## Bar Chart



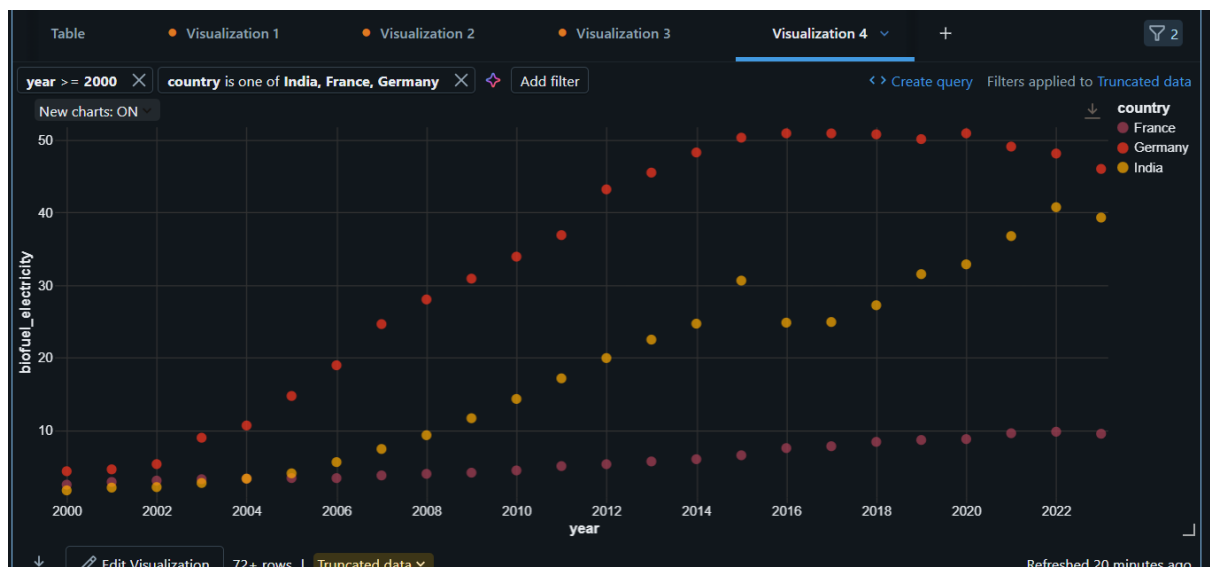
## Bar Chart with Filters



## Scatter Plot

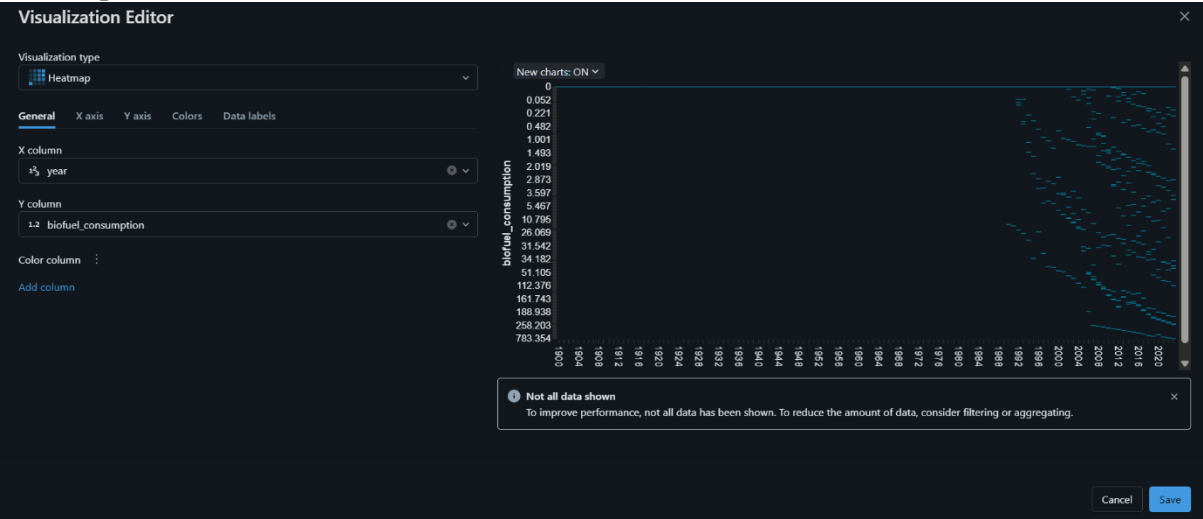


## Scatter Plot with Filters

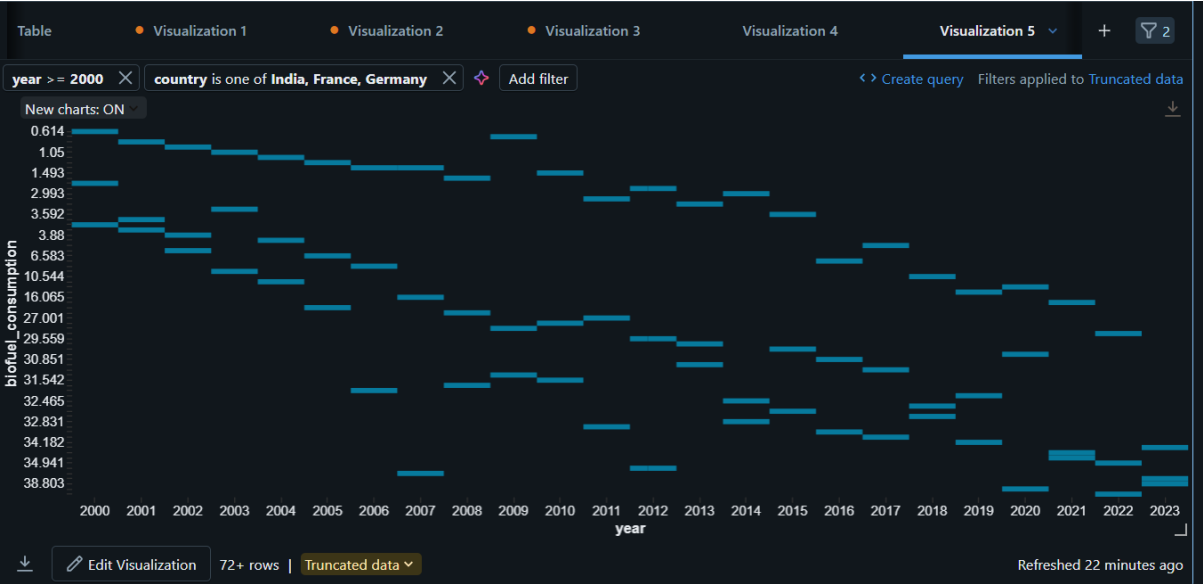




# Heatmap



# Heatmap with Filters



## Combo Chart

