Project Dogs

Step 1: Uploading of data with below details.

Project name - dogs-468706

Dataset name - dogs_data

Table name - dogs_breed

Below is the Schema:

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Data Policies	Description
Dog breed	STRING	NULLABLE	-	-	-	-	-	-
category	STRING	NULLABLE	-	-	-	-	-	-
popularity ranking	INTEGER	NULLABLE	-	-	-	-	-	-
Intelligence ranking	STRING	NULLABLE	-	-	-	-	-	-
Intelligence category	STRING	NULLABLE	-	-	-	-	-	-
Life expectancy	STRING	NULLABLE	-	-	-	-	-	-
Average price	STRING	NULLABLE	-	-	-	-	-	-
Price bracket	STRING	NULLABLE	-	-	-	-	-	-
Food per lifetime	STRING	NULLABLE	-	-	-	-	-	-
Size category	STRING	NULLABLE	-	-	-	-	-	-
Cuteness rating	INTEGER	NULLABLE	-	-	-	-	-	-

Step 2: Cleaning and Sorting of data.

Creating a new table with the cleaned data. To save time so that I don't have to run the cleaning process every time I query the data and it will improve performance also.

During the preview I noticed field names are not in the proper snake case, also other than popularity ranking and cuteness rating every other value is stored as string which can cause a problem in future.

So I am going to create a new table with the name `dogs_breed_cleaned_table`. This table will store a clean version of the original data.

So I am going with below query to achieve this:

```
CREATE TABLE `dogs-468706.dogs_data.dogs_breed_cleaned_table` AS
SELECT
    `Dog breed` AS dog_breed,
```

```
    `category`,
    CAST(REPLACE(`popularity ranking`, 'no data', NULL) AS FLOAT64) AS
popularity_ranking,
    CAST(REPLACE(`Intelligence ranking`, 'no data', NULL) AS FLOAT64) AS
intelligence_ranking,
    `Intelligence category` AS intelligence_category,
    CAST(REPLACE(`Life expectancy`, 'no data', NULL) AS FLOAT64) AS life_expectancy,
    CAST(REPLACE(`Average price`, 'no data', NULL) AS FLOAT64) AS average_price,
    `Price bracket` AS price_bracket,
    CAST(REPLACE(`Food per lifetime`, 'no data', NULL) AS FLOAT64) AS
food_per_lifetime,
    `Size category` AS size_category,
    CAST(REPLACE(`Cuteness rating`, 'no data', NULL) AS FLOAT64) AS cuteness_rating
FROM
    `dogs-468706.dogs_data.dogs_breed`
```

What I am Doing and Why

- **`CREATE TABLE`**: This is the core command to create a new table.
- **`dogs-468706.dogs_data.dogs_breed_cleaned_table`**: Name of the new table being created with project ID, dataset Id and new name in the end.
- **`AS`**: Using this after `CREATE TABLE` to indicate that the new table's structure and data will be defined by the result of the query that follows. Basically telling the database to "create a new table *as* the result of this `SELECT` statement."
- **`SELECT`**: Using the standard SQL command for retrieving data from the original table.
- **`CAST(REPLACE(...) AS FLOAT64):`** `Repeated multiple times.`
    - **`REPLACE('column name', 'no data', NULL)`**: This function is finding and replacing specific text. It's looking for the string 'no data' within a column and replacing it with `NULL` (In SQL, `NULL` is crucial for mathematical operations).
    - **`CAST(... AS FLOAT64)`**: To change the data type of the column. After replacing 'no data' with `NULL`, this ensures the column is stored as a `FLOAT64` (a 64-bit floating-point number, which can handle decimal values). This is essential for performing calculations or aggregations on the data.
- **`Column name AS new_column_name`**: To rename columns for clarity and consistency in snake case. Ex: `Intelligence category` AS intelligence_category.
- **`FROM dogs-468706.dogs_data.dogs_breed`**: This specifies the original table from which all the data is being selected. The `SELECT` statement is pulling data from this source table to populate the new, cleaned table.

But after running the query it showed me an error:

```
No matching signature for function REPLACE Argument types: INT64, STRING, NULL Signature: REPLACE(STRING, STRING, STRING) Argument 1: Unable to coerce type INT64 to expected type STRING Signature: REPLACE(BYTES, BYTES, BYTES) Argument 1: Unable to coerce type INT64 to expected type BYTES at [5:10]
```

Reason: SQL(BigQuery) is trying to run the REPLACE function on a column(popularity ranking) that it has automatically detected as an integer (INT64), but the REPLACE function only works on strings. Also I can see the same thing in one more column(Cuteness rating).

How to fix this:

I need to explicitly tell SQL(BigQuery) to treat the columns as a string *first*, perform the REPLACE, and then convert them to a number.

I will add an additional CAST function inside the REPLACE function for each numeric column.

Updated Query:

```
CREATE TABLE `dogs-468706.dogs_data.dogs_breed_cleaned_table` AS
SELECT
    `Dog breed` AS dog_breed,
    `category`,
    CAST(REPLACE(CAST(`popularity ranking` AS STRING), 'no data', NULL) AS FLOAT64)
AS popularity_ranking,
    CAST(REPLACE(CAST(`Intelligence ranking` AS STRING), 'no data', NULL) AS
FLOAT64) AS intelligence_ranking,
    `Intelligence category` AS intelligence_category,
    CAST(REPLACE(CAST(`Life expectancy` AS STRING), 'no data', NULL) AS FLOAT64) AS
life_expectancy,
    CAST(REPLACE(CAST(`Average price` AS STRING), 'no data', NULL) AS FLOAT64) AS
average_price,
    `Price bracket` AS price_bracket,
    CAST(REPLACE(CAST(`Food per lifetime` AS STRING), 'no data', NULL) AS FLOAT64)
AS food_per_lifetime,
    `Size category` AS size_category,
    CAST(REPLACE(CAST(`Cuteness rating` AS STRING), 'no data', NULL) AS FLOAT64) AS
```

```
cuteness_rating
FROM
    `dogs-468706.dogs_data.dogs_breed`
```

Now one new error:

```
Unrecognized name: `Cuteness rating`; Did you mean Cuteness rating ? at [13:23]
```

Reason: Cuteness rating column has a trailing space as suggested by SQL.

How to fix:
Just adding a trailing space in column name Cuteness rating.

Updated Query:

```
CREATE TABLE `dogs-468706.dogs_data.dogs_breed_cleaned_table` AS
SELECT
    `Dog breed` AS dog_breed,
    `category`,
    CAST(REPLACE(CAST(`popularity ranking` AS STRING), 'no data', NULL) AS FLOAT64)
AS popularity_ranking,
    CAST(REPLACE(CAST(`Intelligence ranking` AS STRING), 'no data', NULL) AS
FLOAT64) AS intelligence_ranking,
    `Intelligence category` AS intelligence_category,
    CAST(REPLACE(CAST(`Life expectancy` AS STRING), 'no data', NULL) AS FLOAT64) AS
life_expectancy,
    CAST(REPLACE(CAST(`Average price` AS STRING), 'no data', NULL) AS FLOAT64) AS
average_price,
    `Price bracket` AS price_bracket,
    CAST(REPLACE(CAST(`Food per lifetime` AS STRING), 'no data', NULL) AS FLOAT64)
AS food_per_lifetime,
    `Size category` AS size_category,
    CAST(REPLACE(CAST(`Cuteness rating ` AS STRING), 'no data', NULL) AS FLOAT64) AS
cuteness_rating
FROM
    `dogs-468706.dogs_data.dogs_breed`
```

Finally no error and got the below message in result.

This statement created a new table named dogs_breed_cleaned_table.

A new table has been created successfully. I can see it in the dataset.

Schema of the new table:

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Data Policies	Description
dog_breed	STRING	NULLABLE	-	-	-	-	-	-
category	STRING	NULLABLE	-	-	-	-	-	-
popularity_ranking	FLOAT	NULLABLE	-	-	-	-	-	-
intelligence_ranking	FLOAT	NULLABLE	-	-	-	-	-	-
intelligence_category	STRING	NULLABLE	-	-	-	-	-	-
life_expectancy	FLOAT	NULLABLE	-	-	-	-	-	-
average_price	FLOAT	NULLABLE	-	-	-	-	-	-
price_bracket	STRING	NULLABLE	-	-	-	-	-	-
food_per_lifetime	FLOAT	NULLABLE	-	-	-	-	-	-
size_category	STRING	NULLABLE	-	-	-	-	-	-
cuteness_rating	FLOAT	NULLABLE	-	-	-	-	-	-

Field names have been changed to snake case. The data type of some columns has been changed to Float.

Now I came across a big problem: all numeric columns turned into NULL – It is happening because the cleaning query was flawed. SQL was likely misinterpreting the data types, which caused all numeric values to be cast incorrectly.

Row	dog_breed	category	popularity_ra...	intelligence_r...	intelligence_category	life_expectan...	average_price	price_bracket	food_per_life...	size_category
1	Sources - see right	null	null	null	null	null	null	null	null	null
2	null	null	null	null	null	null	null	null	null	null
3	Bouvier des Flandres	herding	null	null	Above average	null	null	mid-price	null	large
4	Bearded Collie	herding	null	null	Above average	null	null	budget	null	medium
5	Puli	herding	null	null	Above average	null	null	mid-price	null	medium
6	Briard	herding	null	null	Above average	null	null	budget	null	large
7	Pharaoh Hound	hound	null	null	Above average	null	null	mid-price	null	medium
8	Norwegian Elkhound	hound	null	null	Above average	null	null	budget	null	medium
9	Dalmatian	non-sporting	null	null	Above average	null	null	budget	null	medium
10	Gordon Setter	sporting	null	null	Above average	null	null	mid-price	null	large
11	English Setter	sporting	null	null	Above average	null	null	budget	null	large
12	Irish Setter	sporting	null	null	Above average	null	null	budget	null	large
13	Chesapeake Bay Retriever	sporting	null	null	Above average	null	null	budget	null	large
14	Welsh Springer Spaniel	sporting	null	null	Above average	null	null	mid-price	null	medium
15	Clumber Spaniel	sporting	null	null	Above average	null	null	mid-price	null	medium
16	Field Spaniel	sporting	null	null	Above average	null	null	no data	null	medium
17	Manchester Terrier	terrier	null	null	Above average	null	null	mid-price	null	small
18	Cairn Terrier	terrier	null	null	Above average	null	null	budget	null	small
19	Border Terrier	terrier	null	null	Above average	null	null	mid-price	null	small
20	Norwich Terrier	terrier	null	null	Above average	null	null	mid-price	null	small
21	Australian Terrier	terrier	null	null	Above average	null	null	budget	null	small
22	American Staffordshire Terrier	terrier	null	null	Above average	null	null	mid-price	null	medium

Made some changes in the original CSV(deleted the last two rows in CSV which are causing error) and then reuploaded the table **dogs_breed**.

Also this time using advanced functions like **NULLIF**, **SAFE_CAST** etc.

```
CREATE OR REPLACE TABLE `dogs-468706.dogs_data.dogs_breed_cleaned_final` AS
SELECT
    `Dog breed` AS dog_breed,
    `category`,
    SAFE_CAST(NULLIF(TRIM(CAST(`popularity ranking` AS STRING)), 'no data') AS
FLOAT64) AS popularity_ranking,
    SAFE_CAST(NULLIF(TRIM(`Intelligence ranking`), 'no data') AS FLOAT64) AS
intelligence_ranking,
    `Intelligence category` AS intelligence_category,
    SAFE_CAST(NULLIF(TRIM(`Life expectancy`), 'no data') AS FLOAT64) AS
life_expectancy,
    SAFE_CAST(NULLIF(TRIM(`Average price`), 'no data') AS FLOAT64) AS average_price,
    `Price bracket` AS price_bracket,
    SAFE_CAST(NULLIF(TRIM(`Food per lifetime`), 'no data') AS FLOAT64) AS
food_per_lifetime,
    `Size category` AS size_category,
    SAFE_CAST(NULLIF(TRIM(CAST(`Cuteness rating ` AS STRING)), 'no data') AS
FLOAT64) AS cuteness_rating
FROM
    `dogs-468706.dogs_data.dogs_breed`
WHERE
    `Dog breed` IS NOT NULL AND `Dog breed` <> ''
```

What This Query Does

- **CREATE OR REPLACE TABLE**: This will create a brand new table, dogs_breed_cleaned_final. If I run it again, it will replace the existing one with a new, cleaned version.
- **NULLIF(TRIM(column_name), 'no data')**: Using this to handle missing data. TRIM removes any leading or trailing spaces, and NULLIF then replaces the exact string 'no data' with NULL, which is what I want for any missing value.

- **SAFE_CAST(... AS FLOAT64)**: Very important part of this query. SAFE_CAST will attempt to convert the cleaned strings into a numeric data type (FLOAT64). If it encounters a value it can't convert (e.g., a text string), it will simply return NULL for that value instead of throwing an error.
- **WHERE clause**: This filters out any rows where the Dog breed is empty or null, ensuring no blank rows are included in the final table.

Preview of new table:

Row	dog_breed	category	popularity_ra...	intelligence_r...	intelligence_category	life_expectan...	average_price	price_bracket	food_per_life...	size_category	cuteness_rat...
1	Neapolitan Mastiff	working	110.0	null	no data	null	1760.0	high-end	null	large	1.0
2	Tibetan Mastiff	working	122.0	null	no data	11.92	3460.0	high-end	4907.81	large	2.0
3	Beauceron	herding	144.0	null	no data	null	966.67	mid-price	null	large	2.0
4	Cane Corso	working	67.0	null	no data	null	1070.0	mid-price	null	large	1.0
5	Irish Red and White Setter	sporting	147.0	null	no data	11.57	1000.0	mid-price	null	large	3.0
6	Spinone Italiano	sporting	123.0	null	no data	9.0	1725.0	high-end	null	large	3.0
7	Black Russian Terrier	working	128.0	null	no data	10.5	2833.33	high-end	null	large	3.0
8	Bluetick Coonhound	hound	136.0	null	no data	null	370.0	budget	null	large	3.0
9	Redbone Coonhound	hound	126.0	null	no data	null	425.0	budget	null	large	3.0
10	Leonberger	working	103.0	null	no data	6.98	1480.0	mid-price	4373.28	large	3.0
11	American English Coonhound	hound	33.0	null	no data	null	283.33	budget	null	large	2.0
12	Komondor	working	166.0	null	no data	9.17	656.25	budget	null	large	2.0
13	Greater Swiss Mountain Dog	working	82.0	null	no data	6.8	1605.0	high-end	null	large	3.0
14	Anatolian Shepherd Dog	working	111.0	null	no data	10.75	685.11	budget	6735.35	large	3.0
15	German Shepherd	herding	2.0	3.0	Brightest	9.73	819.5	mid-price	4006.12	large	3.0
16	Doberman Pinscher	working	13.0	5.0	Brightest	10.33	789.5	mid-price	4253.16	large	2.0
17	Rottweiler	working	10.0	9.0	Brightest	9.11	1117.5	mid-price	5707.81	large	2.0
18	Belgian Shepherd Dog (Tervuren)	herding	108.0	14.0	Excellent	10.6	1070.0	mid-price	4364.33	large	3.0
19	Belgian Shepherd Dog	herding	118.0	15.0	Excellent	null	1200.0	mid-price	null	large	3.0
20	Collie	herding	36.0	16.0	Excellent	null	650.0	budget	null	large	3.0
21	German Shorthaired Pointer	sporting	15.0	17.0	Excellent	11.46	545.0	budget	null	large	4.0
22	Weimaraner	sporting	32.0	21.0	Excellent	null	562.0	budget	null	large	3.0
23	Belgian Malinois	herding	74.0	22.0	Excellent	null	1080.0	mid-price	null	large	3.0
24	Bernese Mountain Dog	working	34.0	22.0	Excellent	7.56	1320.0	mid-price	4736.67	large	4.0
25	Chesapeake Bay Retriever	sporting	46.0	27.0	Above average	9.48	522.0	budget	3903.19	large	3.0
26	Giant Schnauzer	working	95.0	28.0	Above average	10.0	810.0	mid-price	null	large	4.0
27	Bouvier des Flandres	herding	83.0	29.0	Above average	10.34	1335.0	mid-price	4257.28	large	3.0

This time all the cleaning steps are executed successfully:

- **Column Names** are now clean and easy to use (e.g., popularity_ranking).
- **Data Types** have been correctly cast to FLOAT64 for all numeric columns.
- **Missing Values** are correctly represented as null.
- The raw header and any blank rows have been filtered out.

Now permanently sorting the dogs_breed_cleaned_final table alphabetically by the dog_breed column. This means overwriting the existing table so that all future queries will reflect this new order.

```
CREATE OR REPLACE TABLE `dogs-468706.dogs_data.dogs_breed_cleaned_final` AS
SELECT *
FROM `dogs-468706.dogs_data.dogs_breed_cleaned_final`
ORDER BY dog_breed ASC;
```

Data has been stored alphabetically:

Row	dog_breed	category	popularity_ra...	intelligence_r...	intelligence_category	life_expectan...	average_price	price_bracket	food_per_life...	size_category	cuteness_rat...
1	Affenpinscher	toy	139.0	37.0	Above average	11.42	510.0	budget	3270.92	small	4.0
2	Afghan Hound	hound	88.0	80.0	Lowest	11.92	890.0	mid-price	7468.4	large	2.0
3	Airedale Terrier	terrier	55.0	29.0	Above average	11.45	732.5	mid-price	null	medium	4.0
4	Akita	working	47.0	54.0	Average	10.16	1201.5	mid-price	6365.69	large	2.0
5	Alaskan Malamute	working	58.0	50.0	Average	10.67	1209.5	mid-price	6685.22	large	3.0
6	American English Coonhound	hound	33.0	null	no data	null	283.33	budget	null	large	2.0
7	American Eskimo Dog	non-sporting	116.0	null	no data	null	560.0	budget	null	small	3.0
8	American Foxhound	hound	173.0	46.0	Average	null	757.14	mid-price	null	medium	3.0
9	American Staffordshire Terrier	terrier	72.0	34.0	Above average	null	1043.0	mid-price	null	medium	1.0
10	American Water Spaniel	sporting	157.0	44.0	Average	null	730.0	mid-price	null	medium	3.0
11	Anatolian Shepherd Dog	working	111.0	null	no data	10.75	685.11	budget	6735.35	large	3.0
12	Australian Cattle Dog	herding	60.0	10.0	Brightest	11.67	530.0	budget	4804.88	medium	2.0
13	Australian Shepherd	herding	24.0	42.0	Average	12.28	565.0	budget	5056.03	medium	3.0
14	Australian Terrier	terrier	121.0	34.0	Above average	11.05	640.0	budget	3164.94	small	4.0
15	Basenji	hound	93.0	79.0	Lowest	13.58	940.0	mid-price	3889.58	medium	3.0
16	Basset Hound	hound	41.0	71.0	Lowest	11.43	489.5	budget	3273.78	small	5.0
17	Beagle	hound	3.0	73.0	Lowest	12.3	287.5	budget	3522.97	small	5.0
18	Bearded Collie	herding	112.0	34.0	Above average	12.77	675.0	budget	5257.78	medium	5.0
19	Beauceron	herding	144.0	null	no data	null	966.67	mid-price	null	large	2.0
20	Bedlington Terrier	terrier	134.0	40.0	Average	13.51	1058.33	mid-price	3869.53	small	2.0
21	Belgian Malinois	herding	74.0	22.0	Excellent	null	1080.0	mid-price	null	large	3.0
22	Belgian Shepherd Dog	herding	118.0	15.0	Excellent	null	1200.0	mid-price	null	large	3.0
23	Belgian Shepherd Dog (Tervuren)	herding	108.0	14.0	Excellent	10.6	1070.0	mid-price	4364.33	large	3.0
24	Bernese Mountain Dog	working	34.0	22.0	Excellent	7.56	1320.0	mid-price	4736.67	large	4.0
25	Bichon Frise	non-sporting	39.0	45.0	Average	12.21	692.5	budget	3497.19	small	6.0
26	Black Russian Terrier	working	128.0	null	no data	10.5	2833.33	high-end	null	large	3.0
27	Black and Tan Coonhound	hound	109.0	44.0	Average	null	325.0	budget	null	large	4.0

Still after running everything I can see some cells containing no data as input under the **intelligence_category** column. To change them to NULL we need to make a small change in the query.

```
CREATE OR REPLACE TABLE `dogs-468706.dogs_data.dogs_breed_cleaned_final` AS
SELECT
    `Dog breed` AS dog_breed,
    `category`,
    SAFE_CAST(NULLIF(TRIM(CAST(`popularity ranking` AS STRING)), 'no data') AS
FLOAT64) AS popularity_ranking,
    SAFE_CAST(NULLIF(TRIM(`Intelligence ranking`), 'no data') AS FLOAT64) AS
intelligence_ranking,
    NULLIF(TRIM(CAST(`Intelligence category` AS STRING)), 'no data') AS
intelligence_category,
    SAFE_CAST(NULLIF(TRIM(`Life expectancy`), 'no data') AS FLOAT64) AS
life_expectancy,
    SAFE_CAST(NULLIF(TRIM(`Average price`), 'no data') AS FLOAT64) AS average_price,
    `Price bracket` AS price_bracket,
```

```
    SAFE_CAST(NULLIF(TRIM(`Food per lifetime`), 'no data') AS FLOAT64) AS
food_per_lifetime,
    `Size category` AS size_category,
    SAFE_CAST(NULLIF(TRIM(CAST(`Cuteness rating ` AS STRING)), 'no data') AS
FLOAT64) AS cuteness_rating
FROM
    `dogs-468706.dogs_data.dogs_breed`
WHERE
    `Dog breed` IS NOT NULL AND `Dog breed` <> ''
    ORDER BY
    dog_breed ASC
```

Also I added an ORDER BY query in the end for alphabetical order.

Row	dog_breed	category	popularity_ra...	intelligence_r...	intelligence_category	life_expectan...	average_price	price_bracket	food_per_life...	size_category	cuteness_rat...
1	Affenpinscher	toy	139.0	37.0	Above average	11.42	510.0	budget	3270.92	small	4.0
2	Afghan Hound	hound	88.0	80.0	Lowest	11.92	890.0	mid-price	7468.4	large	2.0
3	Airedale Terrier	terrier	55.0	29.0	Above average	11.45	732.5	mid-price	null	medium	4.0
4	Akita	working	47.0	54.0	Average	10.16	1201.5	mid-price	6365.69	large	2.0
5	Alaskan Malamute	working	58.0	50.0	Average	10.67	1209.5	mid-price	6685.22	large	3.0
6	American English Coonhound	hound	33.0	null	null	null	283.33	budget	null	large	2.0
7	American Eskimo Dog	non-sporting	116.0	null	null	null	560.0	budget	null	small	3.0
8	American Foxhound	hound	173.0	46.0	Average	null	757.14	mid-price	null	medium	3.0
9	American Staffordshire Terrier	terrier	72.0	34.0	Above average	null	1043.0	mid-price	null	medium	1.0
10	American Water Spaniel	sporting	157.0	44.0	Average	null	730.0	mid-price	null	medium	3.0
11	Anatolian Shepherd Dog	working	111.0	null	null	10.75	685.11	budget	6735.35	large	3.0
12	Australian Cattle Dog	herding	60.0	10.0	Brightest	11.67	530.0	budget	4804.88	medium	2.0
13	Australian Shepherd	herding	24.0	42.0	Average	12.28	565.0	budget	5056.03	medium	3.0
14	Australian Terrier	terrier	121.0	34.0	Above average	11.05	640.0	budget	3164.94	small	4.0
15	Basenji	hound	93.0	79.0	Lowest	13.58	940.0	mid-price	3889.58	medium	3.0
16	Basset Hound	hound	41.0	71.0	Lowest	11.43	489.5	budget	3273.78	small	5.0
17	Beagle	hound	3.0	73.0	Lowest	12.3	287.5	budget	3522.97	small	5.0
18	Bearded Collie	herding	112.0	34.0	Above average	12.77	675.0	budget	5257.78	medium	5.0
19	Beauceron	herding	144.0	null	null	null	966.67	mid-price	null	large	2.0
20	Bedlington Terrier	terrier	134.0	40.0	Average	13.51	1058.33	mid-price	3869.53	small	2.0
21	Belgian Malinois	herding	74.0	22.0	Excellent	null	1080.0	mid-price	null	large	3.0
22	Belgian Shepherd Dog	herding	118.0	15.0	Excellent	null	1200.0	mid-price	null	large	3.0
23	Belgian Shepherd Dog (Tervuren)	herding	108.0	14.0	Excellent	10.6	1070.0	mid-price	4364.33	large	3.0
24	Bernese Mountain Dog	working	34.0	22.0	Excellent	7.56	1320.0	mid-price	4736.67	large	4.0
25	Bichon Frise	non-sporting	39.0	45.0	Average	12.21	692.5	budget	3497.19	small	6.0
26	Black Russian Terrier	working	128.0	null	null	10.5	2833.33	high-end	null	large	3.0
27	Black and Tan Coonhound	hound	109.0	44.0	Average	null	325.0	budget	null	large	4.0

The table is now in its final, clean state:

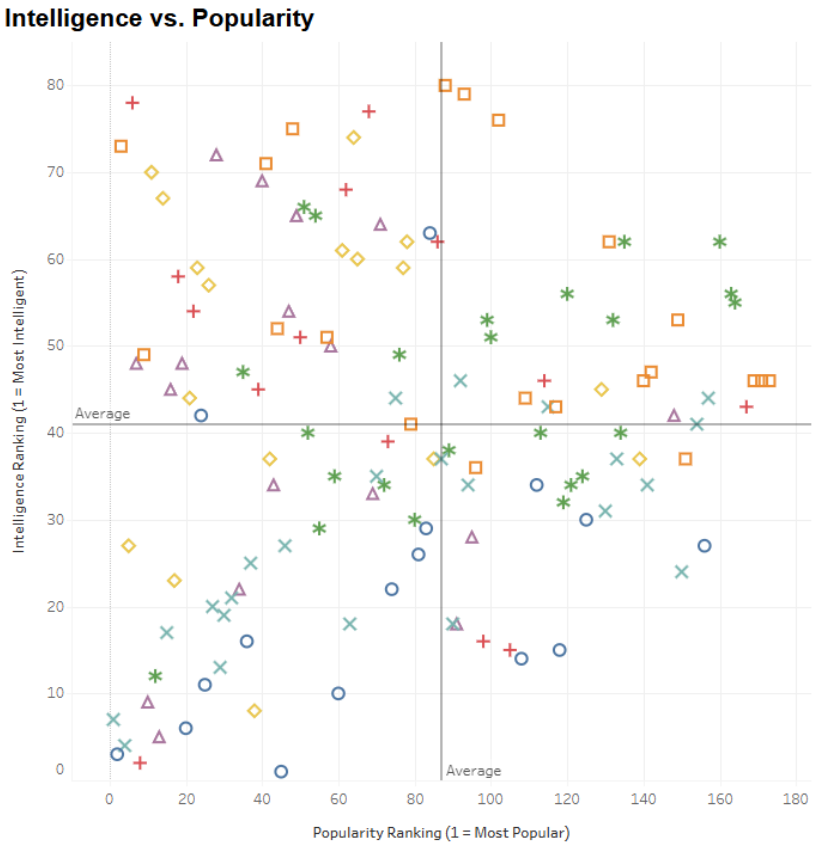
- The data is **permanently sorted** in alphabetical order by `dog_breed`.
- All the numeric columns have been correctly cast to `FLOAT64`, with `'no data'` and other invalid entries replaced by `NULL`.
- The `intelligence_category` column has also been cleaned, with `'no data'` values converted to `NULL`.

Steps in tableau:

Download the cleaned file dogs_breed_cleaned_final and upload it into tableau public.

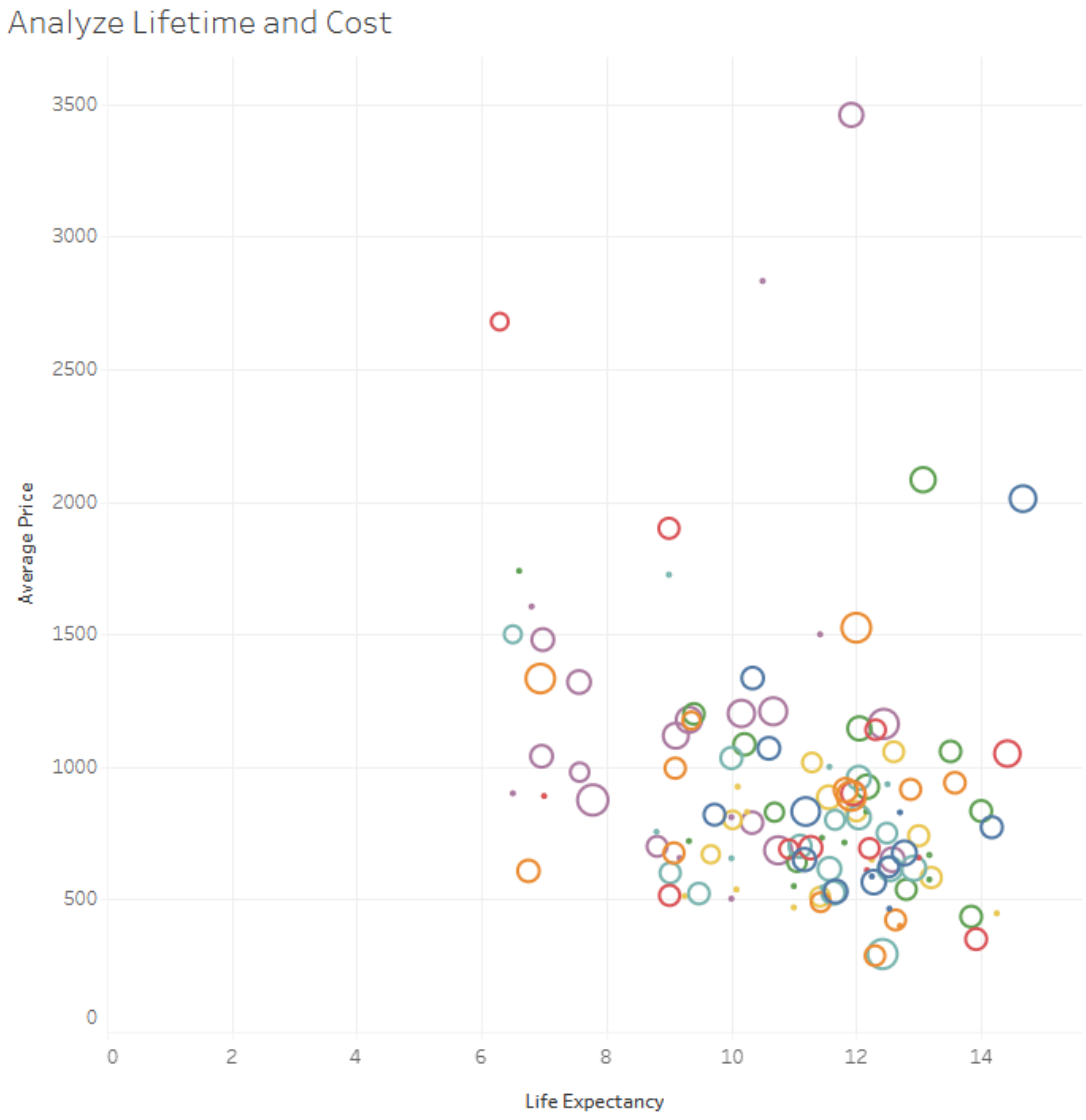
Analyzing Intelligence vs. Popularity

- Creating a **scatter plot** by dragging popularity_ranking to the Columns shelf and intelligence_ranking to the Rows shelf. By default, Tableau tries to aggregate measures (like popularity_ranking and intelligence_ranking) by summing them up.
- To create a separate point for each dog I had to Disaggregate the Data by dragging the **dog_breed** column from the Data pane and dropping it onto the **Detail** card in the Marks pane.
- Adding categories to the color card and shape card. To add another layer of insight to the scatter plot.
- Renamed this sheet to Intelligence vs Popularity.



Analyzing Lifetime and Cost

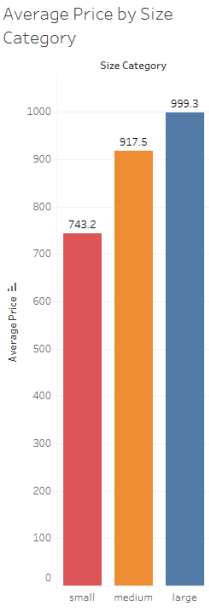
- Creating the Base Scatter Plot
    1. Opening a new, blank worksheet.
    2. Dragging **life_expectancy** to the **Columns** shelf.
    3. Dragging **average_price** to the **Rows** shelf.
- Disaggregate the Data by dragging the **dog_breed** column from the Data pane and dropping it onto the **Detail** card in the Marks pane.
- Adding food_per_lifetime to the **Size** card to add a third dimension to the analysis. The size of each point will now correspond to the food cost for that dog's lifetime.



Adding some bar charts to compare Categories

To create a bar chart that compares average values, we need two types of columns: a **dimension** and a **measure**.

- **Dimension**: Dragging the size_category column from the Data pane to the **Columns** shelf. This will place the different size categories along the horizontal axis.
- **Measure**: Dragging the average_price column to the **Rows** shelf. By default, Tableau is showing a single bar with the sum of all prices. But we need an average.
- Changing the aggregation method by Right clicking on the average_price pill in the Rows shelf. Then from the menu, in "**Measure**" and then selecting "**Average**".
- **Sort the Bars**: To easily compare the average prices, using the **Sort** button sorted the bars in ascending order.
- **Adding Colors**: Drag the size_category column to the **Color** card in the Marks pane. This will color each bar differently, making the chart easier to read.
- **Add Labels**: To show the exact average price on each bar, drag the average_price column to the **Label** card. I need to change the Measure from SUM to AVERAGE. This provides a clear, quantitative value for each bar.
- Renaming the sheet to Average Price by Size Category.



Creating a New Dashboard

1. In Tableau, click the **New Dashboard** icon at the bottom of the workspace. It looks like a grid of four squares.
2. This will open a blank canvas. You can set the dashboard size on the left in the Dashboard pane. For most cases, Automatic is a good choice as it will adjust to the screen size.

Adding Interactivity

Step 1: Open the Actions Menu 📝

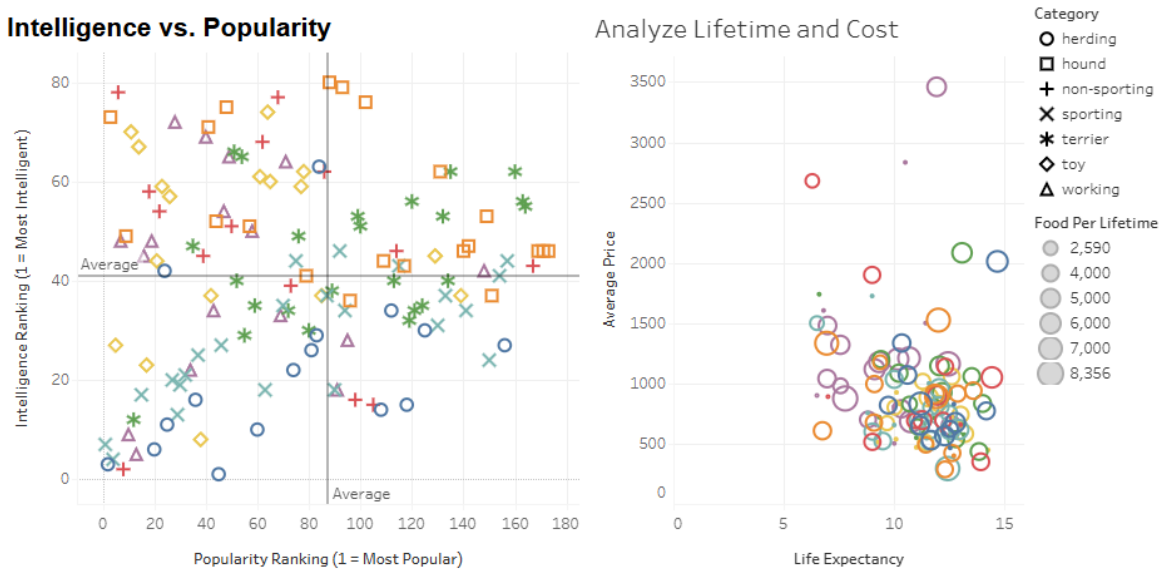
1. Make sure you are on the **dashboard** you created, not on an individual worksheet.
2. From the top menu bar, select **Dashboard** > **Actions**.
3. In the Actions dialog box that opens, click the **Add Action** button and select **Filter**.

Step 2: Configure the Filter Action ⚙️

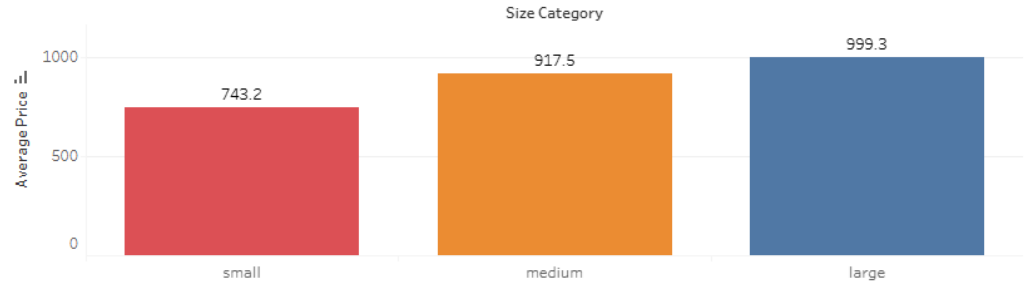
This is the most critical part. Fill out the Add Filter Action dialog box exactly as follows:

1. **Name the Action**: Give it a clear name like "Filter by Category."
2. **Source Sheets**: This is the chart you want to click on to filter others. In your case, check the box for your "**Average Price by Size Category**" bar chart.
3. **Target Sheets**: These are the charts that will be filtered. Check the boxes for your two scatter plots (e.g., "Intelligence vs Popularity" and "Lifetime and Cost").
4. **Run Action On**: Select **Select**. This means the filter will activate when you click on a bar in your source sheet.
5. **Clearing the selection will**: Choose **Show all values**. This is a very important step! It ensures that when you click off a bar, all the data points return to the other charts.
6. Click **OK** to close the dialog box, and then **OK** again to close the Actions menu.

Dog Breed Analysis Dashboard



Average Price by Size Category



Link - https://public.tableau.com/views/DogBreedAnalysisDashboard/DogBreedAnalysisDashboard?:language=en-US&:sid=&:redirect=auth&:display_count=n&:origin=viz_share_link