databricks Pyspark Databricks Experiement

(https://databricks.com)

Below we will be experimenting with pyspark and microsoft azure databricks. We will use a databricks dataset for this experimentation and demonstrate both the powerful visualization functionality of this tool as well as its ability to perform CRUD operations on large datasets.

```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql.types import FloatType
spark = SparkSession \
    .builder \
    .appName("PySpark Demo") \
    .config("spark.some.config.option","some-values") \
    .getOrCreate()
```

```
df = (spark.read
  .format("csv")
  .option("mode", "PERMISSIVE")
  .option("header", "true")
  .load("/databricks-datasets/Rdatasets/data-001/csv/ggplot2/diamonds.csv")
)
```

First we want to look at how big the dataset we imported is, below we will see that it is 53,940 observation, which is decently large.

```
#display the size of the df
df.count()
```

```
53940
```

show is used to see the first few rows of our dataset

```
df.show()
```

```
+---+-----+---------+-----+-------+-----+-----+-----+----+----+----+
|_c0|carat|      cut|color|clarity|depth|table|price|   x|   y|   z|
+---+-----+---------+-----+-------+-----+-----+-----+----+----+----+
|  1| 0.23|    Ideal|    E|    SI2| 61.5|   55|  326|3.95|3.98|2.43|
|  2| 0.21|  Premium|    E|    SI1| 59.8|   61|  326|3.89|3.84|2.31|
|  3| 0.23|     Good|    E|    VS1| 56.9|   65|  327|4.05|4.07|2.31|
|  4| 0.29|  Premium|    I|    VS2| 62.4|   58|  334| 4.2|4.23|2.63|
|  5| 0.31|     Good|    J|    SI2| 63.3|   58|  335|4.34|4.35|2.75|
|  6| 0.24|Very Good|    J|   VVS2| 62.8|   57|  336|3.94|3.96|2.48|
|  7| 0.24|Very Good|    I|   VVS1| 62.3|   57|  336|3.95|3.98|2.47|
|  8| 0.26|Very Good|    H|    SI1| 61.9|   55|  337|4.07|4.11|2.53|
|  9| 0.22|     Fair|    E|    VS2| 65.1|   61|  337|3.87|3.78|2.49|
| 10| 0.23|Very Good|    H|    VS1| 59.4|   61|  338|   4|4.05|2.39|
| 11|  0.3|     Good|    J|    SI1|   64|   55|  339|4.25|4.28|2.73|
| 12| 0.23|    Ideal|    J|    VS1| 62.8|   56|  340|3.93| 3.9|2.46|
| 13| 0.22|  Premium|    F|    SI1| 60.4|   61|  342|3.88|3.84|2.33|
| 14| 0.31|    Ideal|    J|    SI2| 62.2|   54|  344|4.35|4.37|2.71|
| 15|  0.2|  Premium|    E|    SI2| 60.2|   62|  345|3.79|3.75|2.27|
| 16| 0.32|  Premium|    E|     I1| 60.9|   58|  345|4.38|4.42|2.68|
| 17|  0.3|    Ideal|    I|    SI2|   62|   54|  348|4.31|4.34|2.68|
| 18|  0.3|     Good|    J|    SI1| 63.4|   54|  351|4.23|4.29| 2.7|
```

printSchema is used to view the types of data that are in the dataset

```
df.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- carat: string (nullable = true)
 |-- cut: string (nullable = true)
 |-- color: string (nullable = true)
 |-- clarity: string (nullable = true)
 |-- depth: string (nullable = true)
 |-- table: string (nullable = true)
 |-- price: string (nullable = true)
 |-- x: string (nullable = true)
 |-- y: string (nullable = true)
 |-- z: string (nullable = true)
```

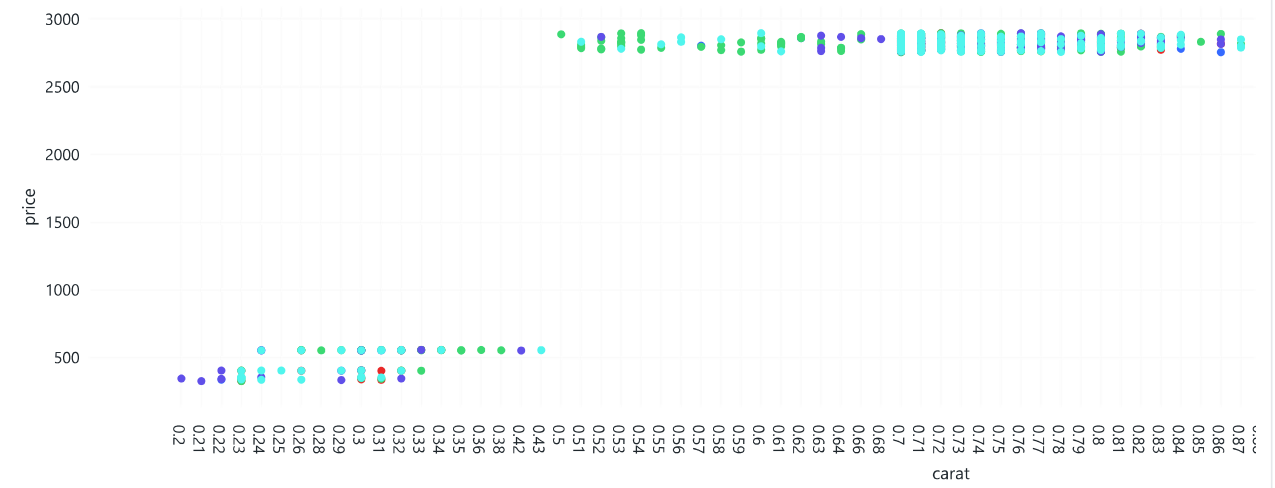
In order to get good plots with this data we have to clean the price variable from a string into a float, otherwise some of the visualizations struggle to capture it as a linear variable.

```
df = df.withColumn("price", df["price"].cast(FloatType()))
```

With display, we can make use of the built in visualizer which makes short work of many complex visualizations. Below, by plotting carat and price, we can see the bimodal clustering of diamonds. This is likely due to CZ diamonds and natural diamonds.

```
display(df)
```

Table **Visualization 1**

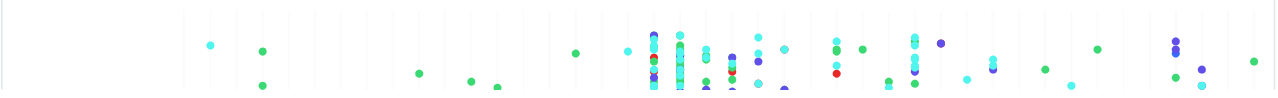


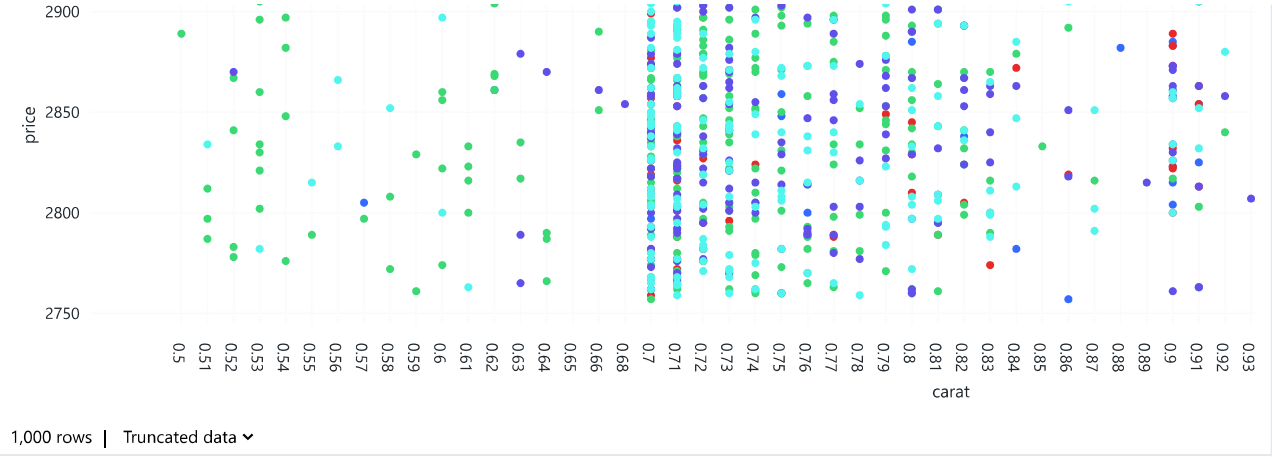
1,000 rows | Truncated data

By subsetting price we can look at the cut, carat, price distribution for natural diamonds

```
filter_price = df.filter(df.price>1000)
display(filter_price)
```

Table **Visualization 1**





1,000 rows | Truncated data

and we can conclude that, clearly, diamond pricing is much more complex than simply its carat and its cut. By not seeing any real discernable pattern here, other variables like color, clarity, table, and depth clearly play a role. What is of note from this plot is the preponderance of diamonds at 0.7 carats. This seems to be a distinct line that may have something to do with the way diamonds are refined.

Now we move on to sql CRUD operations. We will make a temporary database view of this csv file so we can then query it appropriately. We will also make a function to more easily execute queries:

```
df.createOrReplaceTempView("my_table")

def query(q_string):
    query = q_string
    results = spark.sql(query)
    return results
```

```
DataFrame[_c0: string, carat: string, cut: string, color: string, clarity: string, depth: string, table: string, price: float, x: st
ring, y: string, z: string]
```