DSCI 5350-Big data Analytics Created by: Sangeetha Gunasekaran

Spark ML pipeline-Activity 4

1)Load the dataset as a data frame

```
1  val diamondsDF = sqlContext.read
2      .format("com.databricks.spark.csv") // to use spark.csv package
3      .option("header", "true") // Use first line as header
4      .option("inferSchema", "true") // Automatically infer data types
5      //.option("delimiter", ",") // Specify the delimiter as comma or ','
6      .load("dbfs:/FileStore/shared_uploads/jg.sangeetha@gmail.com/Diamonds.csv")

▸ (2) Spark Jobs
▾ diamondsDF: org.apache.spark.sql.DataFrame
      _c0: integer
      carat: double
      cut: string
      color: string
      clarity: string
      depth: double
      table: double
      price: integer
      x: double
      y: double
      z: double

diamondsDF: org.apache.spark.sql.DataFrame = [_c0: int, carat: double ... 9 more fields]
```

2) Print the schema of the data set.

Nullable=true means that the value can be null in those columns

```
1  diamondsDF.printSchema()

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

3)Count the number of records and display first 10 records.

```
1   diamondsDF.count()

 ▸ (2) Spark Jobs

res4: Long = 53940
```

Independent variables→_co, carat, cut, color, clarity, depth, table, x, y, z

Dependent variable→price

We should get rid of first column (i.e., _c0) as it is of no use.

```
1   diamondsDF.show(10)

 ▸ (1) Spark Jobs

+---+-----+---------+-----+-------+-----+-----+-----+----+----+----+
|_c0|carat|      cut|color|clarity|depth|table|price|   x|   y|   z|
+---+-----+---------+-----+-------+-----+-----+-----+----+----+----+
|  1| 0.23|    Ideal|    E|    SI2| 61.5| 55.0|  326|3.95|3.98|2.43|
|  2| 0.21|  Premium|    E|    SI1| 59.8| 61.0|  326|3.89|3.84|2.31|
|  3| 0.23|     Good|    E|    VS1| 56.9| 65.0|  327|4.05|4.07|2.31|
|  4| 0.29|  Premium|    I|    VS2| 62.4| 58.0|  334| 4.2|4.23|2.63|
|  5| 0.31|     Good|    J|    SI2| 63.3| 58.0|  335|4.34|4.35|2.75|
|  6| 0.24|Very Good|    J|   VVS2| 62.8| 57.0|  336|3.94|3.96|2.48|
|  7| 0.24|Very Good|    I|   VVS1| 62.3| 57.0|  336|3.95|3.98|2.47|
|  8| 0.26|Very Good|    H|    SI1| 61.9| 55.0|  337|4.07|4.11|2.53|
|  9| 0.22|     Fair|    E|    VS2| 65.1| 61.0|  337|3.87|3.78|2.49|
| 10| 0.23|Very Good|    H|    VS1| 59.4| 61.0|  338| 4.0|4.05|2.39|
+---+-----+---------+-----+-------+-----+-----+-----+----+----+----+
only showing top 10 rows
```

4) By default, the data type of price (Dependent variable) is taken as **integer**. We have to change that to **double.**

```scala
import org.apache.spark.sql.types.DoubleType
val diamondsnewDF = diamondsDF.select($"carat", $"cut", $"color", $"clarity", $"depth", $"table"
                            ,$"price".cast(DoubleType).as("price"), $"x", $"y", $"z")
diamondsnewDF.printSchema
diamondsnewDF.cache()
```

```
▸ diamondsnewDF: org.apache.spark.sql.DataFrame = [carat: double, cut: string ... 8 more fields]
root
 |-- carat: double (nullable = true)
 |-- cut: string (nullable = true)
 |-- color: string (nullable = true)
 |-- clarity: string (nullable = true)
 |-- depth: double (nullable = true)
 |-- table: double (nullable = true)
 |-- price: double (nullable = true)
 |-- x: double (nullable = true)
 |-- y: double (nullable = true)
 |-- z: double (nullable = true)

import org.apache.spark.sql.types.DoubleType
diamondsnewDF: org.apache.spark.sql.DataFrame = [carat: double, cut: string ... 8 more fields]
res13: diamondsnewDF.type = [carat: double, cut: string ... 8 more fields]
```

5)Display the top 5 records and verify the price column

```scala
diamondsnewDF.show(5,false)
```

▸ (1) Spark Jobs

```
+-----+-------+-----+-------+-----+-----+-----+----+----+----+
|carat|cut    |color|clarity|depth|table|price|x   |y   |z   |
+-----+-------+-----+-------+-----+-----+-----+----+----+----+
|0.23 |Ideal  |E    |SI2    |61.5 |55.0 |326.0|3.95|3.98|2.43|
|0.21 |Premium|E    |SI1    |59.8 |61.0 |326.0|3.89|3.84|2.31|
|0.23 |Good   |E    |VS1    |56.9 |65.0 |327.0|4.05|4.07|2.31|
|0.29 |Premium|I    |VS2    |62.4 |58.0 |334.0|4.2 |4.23|2.63|
|0.31 |Good   |J    |SI2    |63.3 |58.0 |335.0|4.34|4.35|2.75|
+-----+-------+-----+-------+-----+-----+-----+----+----+----+
only showing top 5 rows
```

```
display(diamondsnewDF)
```

(1) Spark Jobs

	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

Truncated results, showing first 1000 rows.

Categorical feature→ cut, color, clarity

Continuous feature→depth, x, y, z

6)Show distinct values in categorical features

```
//Identifying distinct values in categorical variables
val cutsDistinctDF = diamondsnewDF.select("cut").distinct()
val colorsDistinctDF = diamondsnewDF.select("color").distinct()
val claritiesDistinctDF = diamondsnewDF.select("clarity").distinct()
```

- cutsDistinctDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [cut: string]
- colorsDistinctDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [color: string]
- claritiesDistinctDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [clarity: string]

```
cutsDistinctDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [cut: string]
colorsDistinctDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [color: string]
claritiesDistinctDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [clarity: string]
```

```
cutsDistinctDF.show()
colorsDistinctDF.show()
claritiesDistinctDF.show()
```

cut
Premium
Ideal
Good
Fair
Very Good

color
F
E
D
J
G
I
H

clarity
VVS2
SI1
IF
I1
VVS1
VS2
SI2
VS1

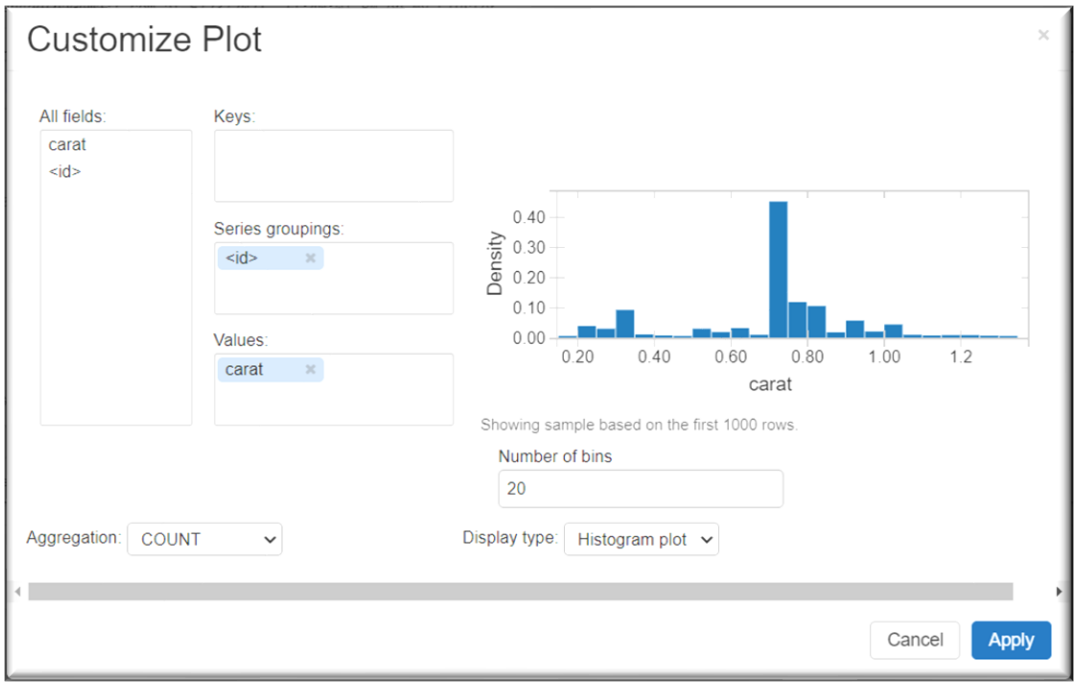
7) Display the distinct values in **cut.** Click drop down in chart options and select bar chart

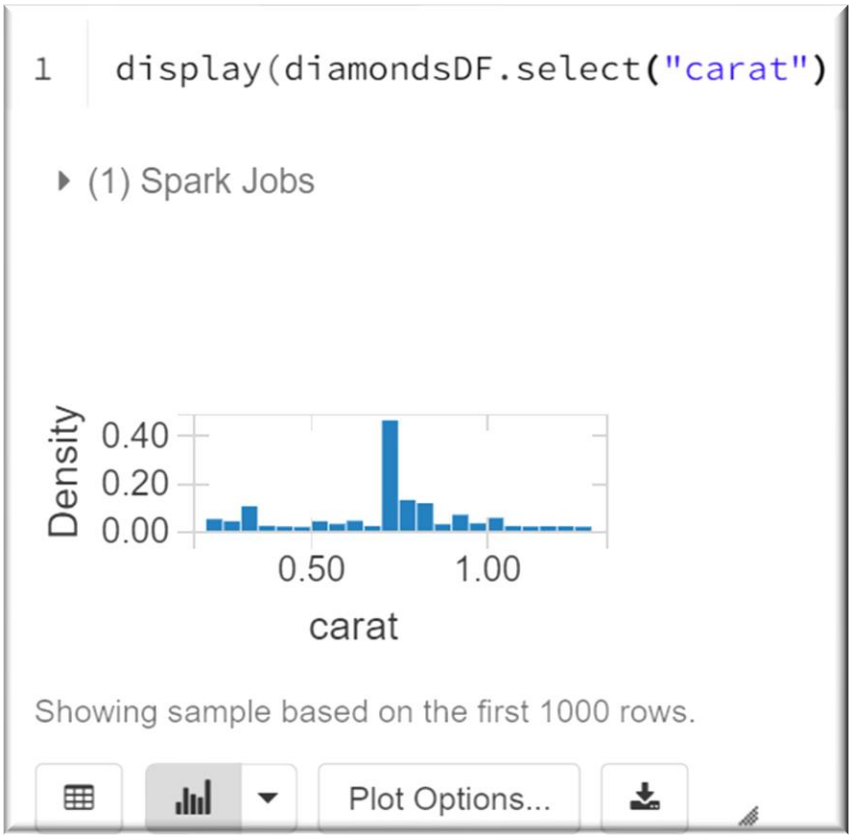
Select plot options and do the following settings

Value=<id> ; Series groupings='cut';Aggregation=COUNT

Do the same for color and clarity

8) Visualize the continuous features





Most of the diamonds are small . The above visualization shows a skewed distribution.

Decision trees works well with this type of distribution.

9) Using interactive SQL to explore the data

```
1  val diamondsDColoredDF = diamondsDF.select("carat", "color", "price").filter($"color" === "D")

▸ diamondsDColoredDF:  org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [carat: double, color: string ... 1 more fields]
diamondsDColoredDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [carat: double, color: string ... 1 more field]
```

```
1  diamondsDColoredDF.show(5)

▸ (1) Spark Jobs

+-----+-----+-----+
|carat|color|price|
+-----+-----+-----+
| 0.23|    D|  357|
| 0.23|    D|  402|
| 0.26|    D|  403|
| 0.26|    D|  403|
| 0.26|    D|  403|
+-----+-----+-----+
only showing top 5 rows
```

10) Create a temporary view

```
1  diamondsnewDF.createTempView("DiamondTemptable")
2  sqlContext.tables.show()

+--------+----------------+-----------+
|database|       tableName|isTemporary|
+--------+----------------+-----------+
|        |         diamond|       true|
|        |diamondtemptable|       true|
+--------+----------------+-----------+
```

11) Write sql query to fetch data from the view

```
1  %sql
2  select carat, color, price from DiamondTemptable where color='D'
```

▸ (1) Spark Jobs

	carat	color	price
1	0.23	D	357
2	0.23	D	402
3	0.26	D	403
4	0.26	D	403
5	0.26	D	403
6	0.22	D	404
7	0.3	D	552

Truncated results, showing first 1000 rows.

Alternate method:

```
1  val diamondsDColoredDF_FromTable = spark.sql("SELECT carat, color, price FROM DiamondTemptable WHERE color='D'")
2  display(diamondsDColoredDF_FromTable)
```

▸ (1) Spark Jobs

▸ diamondsDColoredDF_FromTable: org.apache.spark.sql.DataFrame = [carat: double, color: string ... 1 more fields]

	carat	color	price
1	0.23	D	357
2	0.23	D	402
3	0.26	D	403
4	0.26	D	403
5	0.26	D	403
6	0.22	D	404
7	0.3	D	552

10) Select all the values from the temp table where clarity starts with ‘I’

```
1  display(spark.sql("SELECT * FROM DiamondTemptable WHERE clarity LIKE 'I%'"))
```

▸ (1) Spark Jobs

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.32	Premium	E	I1	60.9	58	345	4.38	4.42	2.68
2	1.17	Very Good	J	I1	60.2	61	2774	6.83	6.9	4.13
3	1.01	Premium	F	I1	61.8	60	2781	6.39	6.36	3.94
4	0.52	Ideal	F	IF	62.2	55	2783	5.14	5.18	3.21
5	1.01	Fair	E	I1	64.5	58	2788	6.29	6.21	4.03
6	0.55	Ideal	G	IF	60.9	57	2789	5.28	5.3	3.22
7	0.64	Ideal	G	IF	61.3	56	2790	5.54	5.58	3.41

Truncated results, showing first 1000 rows.

11) Execute Aggregate function to get the average price of the diamonds

```
1  display(spark.sql("select avg(price) as avgprice from DiamondTemptable"))
```

▸ (2) Spark Jobs

	avgprice
1	3932.799721913237

Try other aggregate functions

12)Display carat, clarity, price of top 5 records sorted by carat field

```
1  display(spark.sql("select carat, clarity, price from DiamondTemptable order by carat desc, price desc limit 5"))
```

▸ (1) Spark Jobs

	carat	clarity	price
1	5.01	I1	18018
2	4.5	I1	18531
3	4.13	I1	17329
4	4.01	I1	15223
5	4.01	I1	15223