databricks Jessie Ma - Assignment 4 - Question 2

(https://databricks.com)

Diamond's Pricing model:

Input dataset:

`/databricks-datasets/Rdatasets/data-001/csv/ggplot2/diamonds.csv`

Using the Apache Spark ML pipeline, build a model to predict the price of a diamond base on the available features. How would you handle non-numerical data?

Information about the dataset:

- http://ggplot2.tidyverse.org/reference/diamonds.html (http://ggplot2.tidyverse.org/reference/diamonds.html)
- You can find plenty of exploratory analysis examples around the web for this particular dataset

Read https://spark.apache.org/docs/latest/ml-features.html (https://spark.apache.org/docs/latest/ml-features.html) to learn more about transforming features, dealing with categorical variables, etc.

```
dataPath = "/databricks-datasets/Rdatasets/data-001/csv/ggplot2/diamonds.csv"
diamonds = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option("inferSchema", "true").load(dataPath)
```

```
display(diamonds)
```

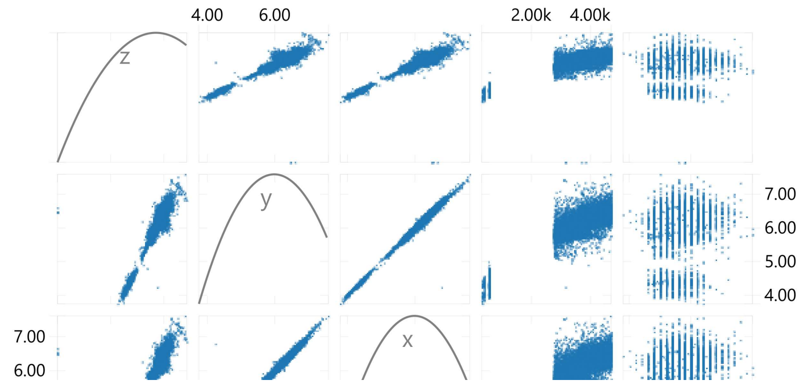
Table

	_c0	carat	cut	color	clarity	depth	table	price	x
1	1	0.23	Ideal	E	SI2	61.5	55	326	3.95
2	2	0.21	Premium	E	SI1	59.8	61	326	3.89
3	3	0.23	Good	E	VS1	56.9	65	327	4.05
4	4	0.29	Premium	I	VS2	62.4	58	334	4.2
5	5	0.31	Good	J	SI2	63.3	58	335	4.34
6	6	0.24	Very Good	J	VVS2	62.8	57	336	3.94
7	7	0.24	Very Good	I	VVS1	62.3	57	336	3.95

10,000 rows | Truncated data

```
display(diamonds)
```

Visualization



Truncated to 10,000 rows.

```
%fs ls /databricks-datasets/Rdatasets/data-001/csv/ggplot2/
```

Table

	path	name	size	modificationTime
1	dbfs:/databricks-datasets/Rdatasets/data-001/csv/ggplot2/diamonds.csv	diamonds.csv	3192560	1416619980000
2	dbfs:/databricks-datasets/Rdatasets/data-001/csv/ggplot2/economics.csv	economics.csv	20731	1416619980000
3	dbfs:/databricks-datasets/Rdatasets/data-001/csv/ggplot2/midwest.csv	midwest.csv	100539	1416619980000
4	dbfs:/databricks-datasets/Rdatasets/data-001/csv/ggplot2/movies.csv	movies.csv	6000709	1416619980000
5	dbfs:/databricks-datasets/Rdatasets/data-001/csv/ggplot2/mpg.csv	mpg.csv	17345	1416619980000
6	dbfs:/databricks-datasets/Rdatasets/data-001/csv/ggplot2/msleep.csv	msleep.csv	7182	1416619980000
7	dbfs:/databricks-datasets/Rdatasets/data-001/csv/ggplot2/presidential.csv	presidential.csv	512	1416619981000

8 rows

Jessie Ma - Assignment 4 - Question 2 Starts Here

```
display(diamonds)
```

Table

	_c0	carat	cut	color	clarity	depth	table	price	x
1	1	0.23	Ideal	E	SI2	61.5	55	326	3.95
2	2	0.21	Premium	E	SI1	59.8	61	326	3.89
3	3	0.23	Good	E	VS1	56.9	65	327	4.05
4	4	0.29	Premium	I	VS2	62.4	58	334	4.2
5	5	0.31	Good	J	SI2	63.3	58	335	4.34
6	6	0.24	Very Good	J	VVS2	62.8	57	336	3.94
7	7	0.24	Very Good	I	VVS1	62.3	57	336	3.95

10,000 rows | Truncated data

```
diamonds.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)
```

```
diamonds.columns
```

```
['_c0',
 'carat',
 'cut',
 'color',
 'clarity',
 'depth',
 'table',
 'price',
 'x',
 'y',
 'z']
```

First treat missing values:

```
# Find count for Null, None, NaN of all columns in df, clean up data if necessary
from pyspark.sql.functions import col, isnan, when, count
diamonds.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in diamonds.columns]).show()
```

```
+---+-----+---+-----+-------+-----+-----+-----+---+---+---+
|_c0|carat|cut|color|clarity|depth|table|price|  x|  y|  z|
+---+-----+---+-----+-------+-----+-----+-----+---+---+---+
|  0|    0|  0|    0|      0|    0|    0|    0|  0|  0|  0|
+---+-----+---+-----+-------+-----+-----+-----+---+---+---+
```

There are none which is great!

Let's rename the x, y and z columns:

```
diamonds = diamonds.withColumnRenamed("depth", "depth_percentage") \
          .withColumnRenamed("x", "length") \
          .withColumnRenamed("y", "width") \
          .withColumnRenamed("z", "depth")
display(diamonds)
```

Table

	_c0	carat	cut	color	clarity	depth_percentage	table	price	length
1	1	0.23	Ideal	E	SI2	61.5	55	326	3.95
2	2	0.21	Premium	E	SI1	59.8	61	326	3.89
3	3	0.23	Good	E	VS1	56.9	65	327	4.05
4	4	0.29	Premium	I	VS2	62.4	58	334	4.2
5	5	0.31	Good	J	SI2	63.3	58	335	4.34
6	6	0.24	Very Good	J	VVS2	62.8	57	336	3.94
7	7	0.24	Very Good	I	VVS1	62.3	57	336	3.95

10,000 rows | Truncated data

Convert categorical variables to numerical:

```
data_ml=diamonds.drop("_c0")
from pyspark.ml.feature import StringIndexer
indexer=StringIndexer(inputCol="cut",outputCol="cut_num")
indexed=indexer.fit(data_ml).transform(data_ml)
indexer=StringIndexer(inputCol="color",outputCol="color_num")
indexed=indexer.fit(indexed).transform(indexed)
indexer=StringIndexer(inputCol="clarity",outputCol="clarity_num")
indexed=indexer.fit(indexed).transform(indexed)
```

```
indexed.show()
```

```
+-----+---------+-----+-------+----------------+-----+-----+------+-----+-----+-------+---------+-----------+
|carat|      cut|color|clarity|depth_percentage|table|price|length|width|depth|cut_num|color_num|clarity_num|
+-----+---------+-----+-------+----------------+-----+-----+------+-----+-----+-------+---------+-----------+
| 0.23|    Ideal|    E|    SI2|            61.5| 55.0|  326|  3.95| 3.98| 2.43|    0.0|      1.0|        2.0|
| 0.21|  Premium|    E|    SI1|            59.8| 61.0|  326|  3.89| 3.84| 2.31|    1.0|      1.0|        0.0|
| 0.23|     Good|    E|    VS1|            56.9| 65.0|  327|  4.05| 4.07| 2.31|    3.0|      1.0|        3.0|
| 0.29|  Premium|    I|    VS2|            62.4| 58.0|  334|   4.2| 4.23| 2.63|    1.0|      5.0|        1.0|
| 0.31|     Good|    J|    SI2|            63.3| 58.0|  335|  4.34| 4.35| 2.75|    3.0|      6.0|        2.0|
| 0.24|Very Good|    J|   VVS2|            62.8| 57.0|  336|  3.94| 3.96| 2.48|    2.0|      6.0|        4.0|
| 0.24|Very Good|    I|   VVS1|            62.3| 57.0|  336|  3.95| 3.98| 2.47|    2.0|      5.0|        5.0|
| 0.26|Very Good|    H|    SI1|            61.9| 55.0|  337|  4.07| 4.11| 2.53|    2.0|      3.0|        0.0|
| 0.22|     Fair|    E|    VS2|            65.1| 61.0|  337|  3.87| 3.78| 2.49|    4.0|      1.0|        1.0|
| 0.23|Very Good|    H|    VS1|            59.4| 61.0|  338|   4.0| 4.05| 2.39|    2.0|      3.0|        3.0|
|  0.3|     Good|    J|    SI1|            64.0| 55.0|  339|  4.25| 4.28| 2.73|    3.0|      6.0|        0.0|
| 0.23|    Ideal|    J|    VS1|            62.8| 56.0|  340|  3.93|  3.9| 2.46|    0.0|      6.0|        3.0|
| 0.22|  Premium|    F|    SI1|            60.4| 61.0|  342|  3.88| 3.84| 2.33|    1.0|      2.0|        0.0|
| 0.31|    Ideal|    J|    SI2|            62.2| 54.0|  344|  4.35| 4.37| 2.71|    0.0|      6.0|        2.0|
|  0.2|  Premium|    E|    SI2|            60.2| 62.0|  345|  3.79| 3.75| 2.27|    1.0|      1.0|        2.0|
| 0.32|  Premium|    E|     I1|            60.9| 58.0|  345|  4.38| 4.42| 2.68|    1.0|      1.0|        7.0|
|  0.3|    Ideal|    I|    SI2|            62.0| 54.0|  348|  4.31| 4.34| 2.68|    0.0|      5.0|        2.0|
|  0.3|     Good|    J|    SI1|            63.4| 54.0|  351|  4.23| 4.29|  2.7|    3.0|      6.0|        0.0|
```

Assemble columns into one single vector:

```
from pyspark.ml.linalg import Vector
from pyspark.ml.feature import VectorAssembler
assembler=VectorAssembler(inputCols=
['carat','depth_percentage','table','length','width','depth','cut_num','color_num','clarity_num'],outputCol='features')
assembler
```

```
VectorAssembler_e4c8975b6aaf
```

```
output=assembler.transform(indexed)
output.show()
```

```
+-----+---------+-----+-------+----------------+-----+-----+------+-----+-----+-------+---------+-----------+------------------
--+
|carat|      cut|color|clarity|depth_percentage|table|price|length|width|depth|cut_num|color_num|clarity_num|            featur
es|
+-----+---------+-----+-------+----------------+-----+-----+------+-----+-----+-------+---------+-----------+------------------
--+
| 0.23|    Ideal|    E|    SI2|            61.5| 55.0|  326|  3.95| 3.98| 2.43|    0.0|      1.0|        2.0|[0.23,61.5,55.0,
3...|
| 0.21|  Premium|    E|    SI1|            59.8| 61.0|  326|  3.89| 3.84| 2.31|    1.0|      1.0|        0.0|[0.21,59.8,61.0,
```

```
3...|
| 0.23|     Good|    E|    VS1|           56.9| 65.0|  327|  4.05| 4.07| 2.31|    3.0|      1.0|        3.0|[0.23,56.9,65.0,
4...|
| 0.29|  Premium|    I|    VS2|           62.4| 58.0|  334|   4.2| 4.23| 2.63|    1.0|      5.0|        1.0|[0.29,62.4,58.0,
4...|
| 0.31|     Good|    J|    SI2|           63.3| 58.0|  335|  4.34| 4.35| 2.75|    3.0|      6.0|        2.0|[0.31,63.3,58.0,
4...|
| 0.24|Very Good|    J|   VVS2|           62.8| 57.0|  336|  3.94| 3.96| 2.48|    2.0|      6.0|        4.0|[0.24,62.8,57.0,
3...|
| 0.24|Very Good|    I|   VVS1|           62.3| 57.0|  336|  3.95| 3.98| 2.47|    2.0|      5.0|        5.0|[0.24,62.3,57.0,
3...|
```

```
final_data=output.select('features','price')
final_data.show()
```

```
+--------------------+-----+
|            features|price|
+--------------------+-----+
|[0.23,61.5,55.0,3...|  326|
|[0.21,59.8,61.0,3...|  326|
|[0.23,56.9,65.0,4...|  327|
|[0.29,62.4,58.0,4...|  334|
|[0.31,63.3,58.0,4...|  335|
|[0.24,62.8,57.0,3...|  336|
|[0.24,62.3,57.0,3...|  336|
|[0.26,61.9,55.0,4...|  337|
|[0.22,65.1,61.0,3...|  337|
|[0.23,59.4,61.0,4...|  338|
|[0.3,64.0,55.0,4....|  339|
|[0.23,62.8,56.0,3...|  340|
|[0.22,60.4,61.0,3...|  342|
|[0.31,62.2,54.0,4...|  344|
|[0.2,60.2,62.0,3....|  345|
|[0.32,60.9,58.0,4...|  345|
|[0.3,62.0,54.0,4....|  348|
|[0.3,63.4,54.0,4....|  351|
```

Split data into training and test:

```
train_data,test_data=final_data.randomSplit([0.7,0.3])
```

```
train_data.describe().show()
```

```
+-------+------------------+
|summary|             price|
+-------+------------------+
|  count|             37714|
|   mean| 3947.754043591239|
| stddev|4004.8924181494663|
|    min|               326|
|    max|             18818|
+-------+------------------+
```

```
test_data.describe().show()
```

```
+-------+------------------+
|summary|             price|
+-------+------------------+
|  count|             16226|
```

```
|   mean|3898.0414766424256|
| stddev|3953.1940505962266|
|    min|               334|
|    max|             18823|
+-------+------------------+
```

Build and train linear regression model, and evaluate model:

```
from pyspark.ml.regression import LinearRegression
from pyspark.mllib.evaluation import RegressionMetrics
lrobject=LinearRegression(featuresCol='features',labelCol='price')
trained_model=lrobject.fit(train_data)
ship_results=trained_model.evaluate(train_data)
print('Rsquared Error:',ship_results.r2)
```

```
Rsquared Error: 0.8695507651846006
```

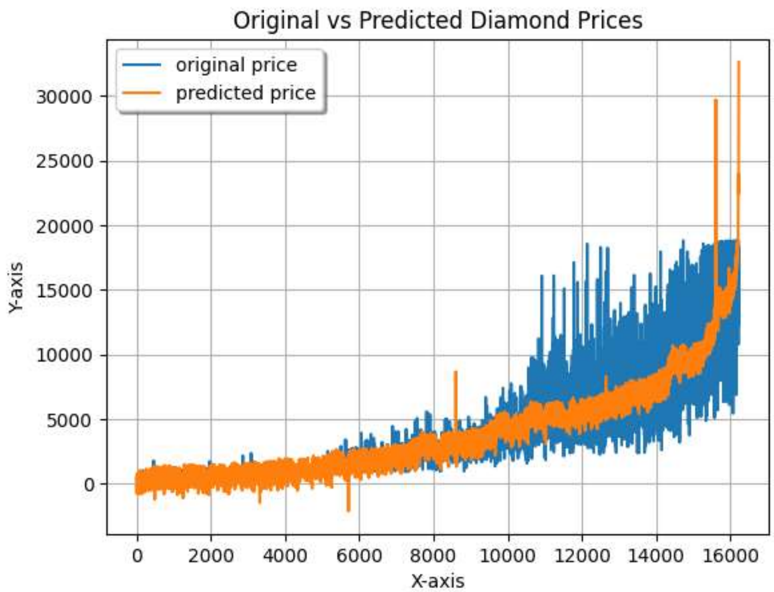
RSquared error is decently high, indicating the data fits well in the regression model and model accuracy is good and can be used for predictive analysis.

Predictions made by model:

```
pred=trained_model.transform(test_data)
pred.show()
```

```
+--------------------+-----+-------------------+
|            features|price|         prediction|
+--------------------+-----+-------------------+
|[0.2,59.0,60.0,3....|  367| 144.87623782566334|
|[0.2,61.7,60.0,3....|  367| -762.0055068959737|
|[0.2,62.2,57.0,3....|  367|  85.95149292067799|
|[0.2,62.6,59.0,3....|  367| -396.5929130641671|
|[0.21,60.6,60.0,3...|  386| -591.2493325386349|
|[0.21,63.2,54.0,3...|  386|-199.33600023468898|
|[0.22,59.3,62.0,3...|  404| -473.7152740346137|
|[0.23,56.2,60.0,4...|  395|  689.3064364313286|
|[0.23,58.1,52.0,4...|  458|  585.3349642050252|
|[0.23,58.1,59.0,4...|  550|  758.5837886684221|
|[0.23,58.1,63.0,4...|  468|   85.5971494766327|
|[0.23,58.5,61.0,4...|  411| -388.6618723298416|
|[0.23,58.6,61.0,4...|  530|  411.2149448132095|
|[0.23,59.2,61.0,4...|  389|-109.52578348185853|
|[0.23,59.3,57.0,4...|  530|  263.2296663039724|
|[0.23,59.3,60.0,4...|  505|  330.0272893373931|
|[0.23,59.4,61.0,4...|  338|-217.08939751048274|
|[0.23,59.4,61.0,4...|  434|  -133.518593736002|
```

Graph showing how well predictions made by the model approximate the original diamond prices:


Original vs Predicted Diamond Prices
original price
predicted price
30000
25000
20000
15000
10000
5000
0
Y-axis
0
2000
4000
6000
8000
10000
12000
14000
16000
X-axis