databricksSpark 2 Priya

```
# Question: How would you handle non-numerical data?
# Word2vec to convert words/non-numerics/text into numbers/vectors appropriately and properly
# using Vector Assembler to complete the transformation/conversion
# Pipeline to relay the conversion into the process of regression modeling and prediction analysis
```

```
# import libraries and built-in functions
from pyspark import SparkContext
from pyspark import SparkConf
import pandas as pd
from pyspark.sql import SQLContext, DataFrame
from pyspark.sql.functions import *
```

```
# initiate SQL Context contained in a variable
sqlContext = SQLContext(sc)
```

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

path
dbfs:/databricks-datasets/Rdatasets/data-001/Makefile
dbfs:/databricks-datasets/Rdatasets/data-001/README.rst
dbfs:/databricks-datasets/Rdatasets/data-001/Rdatasets.R
dbfs:/databricks-datasets/Rdatasets/data-001/csv/
dbfs:/databricks-datasets/Rdatasets/data-001/datasets.csv
dbfs:/databricks-datasets/Rdatasets/data-001/datasets.html

```
dbfs:/databricks-datasets/Rdatasets/data-001/doc/
```

```
#loading in data set
dataPath = "/databricks-datasets/Rdatasets/data-001/csv/ggplot2/diamonds.csv"
diamonds1 = sqlContext.read.format("com.databricks.spark.csv").option("header","true").load(dataPath)


#displaying dataset in spark dataframe
display(diamonds1)
```

_c0	carat	cut	color	clarity	depth	table
1	0.23	Ideal	E	SI2	61.5	55
2	0.21	Premium	E	SI1	59.8	61
3	0.23	Good	E	VS1	56.9	65
4	0.29	Premium	I	VS2	62.4	58
5	0.31	Good	J	SI2	63.3	58
6	0.24	Very Good	J	VVS2	62.8	57
7	0.24	Very Good	I	VVS1	62.3	57
8	0.26	Very Good	H	SI1	61.9	55
9	0.22	Fair	E	VS2	65.1	61

Showing the first 1000 rows.

```
# showing data types of each column
diamonds1.dtypes

Out[6]:
[('_c0', 'string'),
 ('carat', 'string'),
 ('cut', 'string'),
 ('color', 'string'),
 ('clarity', 'string'),
 ('depth', 'string'),
 ('table', 'string'),
 ('price', 'string'),
 ('x', 'string'),
 ('y', 'string'),
 ('z', 'string')]
```

```
# importing data types
from pyspark.sql.types import StructField,IntegerType, StructType,StringType, DecimalType, FloatType,
ArrayType
```

```
# defining data types by casting each column with appropriate types
diamonds1=diamonds1.withColumn("_c0",diamonds1["_c0"].cast('integer')).withColumn("carat",diamonds1["carat"].c
ast('float')).withColumn("cut",diamonds1["cut"].cast("string")).withColumn("color",diamonds1["color"].cast('st
ring')).withColumn("clarity",diamonds1["clarity"].cast('string')).withColumn("depth",diamonds1["depth"].cast("
float")).withColumn("table",diamonds1["table"].cast('integer')).withColumn("price",diamonds1["price"].cast('fl
oat')).withColumn("x",diamonds1["x"].cast("float")).withColumn("y",diamonds1["y"].cast("float")).withColumn("z
",diamonds1["z"].cast("float"))
```

```
# showing column data types of this dataframe
diamonds1.dtypes
```

```
Out[9]:
[('_c0', 'int'),
 ('carat', 'float'),
 ('cut', 'string'),
 ('color', 'string'),
 ('clarity', 'string'),
 ('depth', 'float'),
 ('table', 'int'),
 ('price', 'float'),
 ('x', 'float'),
 ('y', 'float'),
 ('z', 'float')]
```

```
# displaying the dataframe
display(diamonds1)
```

_c0	carat	cut	color	clarity	depth	table
1	0.23	Ideal	E	SI2	61.5	55
2	0.21	Premium	E	SI1	59.8	61
3	0.23	Good	E	VS1	56.9	65
4	0.29	Premium	I	VS2	62.4	58
5	0.31	Good	J	SI2	63.3	58
6	0.24	Very Good	J	VVS2	62.8	57
7	0.24	Very Good	I	VVS1	62.3	57
8	0.26	Very Good	H	SI1	61.9	55
9	0.22	Fair	E	VS2	65.1	61

Showing the first 1000 rows.

```
# ensuring data types of each column in this dataframe
diamonds1.dtypes

Out[11]:
[('_c0', 'int'),
 ('carat', 'float'),
 ('cut', 'string'),
 ('color', 'string'),
 ('clarity', 'string'),
 ('depth', 'float'),
 ('table', 'int'),
 ('price', 'float'),
 ('x', 'float'),
 ('y', 'float'),
 ('z', 'float')]

# importing libraries
from pyspark import SparkContext
from pyspark import SparkConf
import pandas as pd
from pyspark.sql import SQLContext, DataFrame
from pyspark.sql.functions import *
sqlContext = SQLContext(sc)


# import libraries
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import Word2Vec


# printing schema of dataframe
diamonds1.printSchema
```

```
Out[14]: <bound method DataFrame.printSchema of DataFrame[_c0: int, carat: float, cut: string, color: string,
clarity: string, depth: float, table: int, price: float, x: float, y: float, z: float]>
```

```
# casting string data types to array<string>
diamondsa = diamonds1.withColumn("cutVec",split(col("cut"),
",\s*").cast("array<string>").alias("cut")).withColumn("colorVec",split(col("color"),
",\s*").cast("array<string>").alias("color")).withColumn("clarityVec",split(col("clarity"),
",\s*").cast("array<string>").alias("clarity"))



# using word2vec to convert words/non-numerics/text into numbers appropriately and properly
# column "cutVec"
word2Veccut = Word2Vec(vectorSize=5, seed=42, inputCol="cutVec", outputCol="cutVec2")
model = word2Veccut.fit(diamondsa)
# adding the transformed column to dataframe
diamondsa = model.transform(diamondsa)

# column "colorVec"
word2Veccolor = Word2Vec(vectorSize=5, seed=42, inputCol="colorVec", outputCol="colorVec2")
model2 = word2Veccolor.fit(diamondsa)
# adding the transformed column to dataframe
diamondsa = model2.transform(diamondsa)

# column "clarityVec"
word2Vecclarity = Word2Vec(vectorSize=5, seed=42, inputCol="clarityVec", outputCol="clarityVec2")
model3 = word2Vecclarity.fit(diamondsa)
# adding the transformed column to dataframe
diamondsa = model3.transform(diamondsa)

# display dataframe
display(diamondsa)
```

_c0	carat	cut	color	clarity	depth	table	price	x	y	z	cutVec	colorVec	clarityVec
1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43	▸ ["Ideal"]	▸ ["E"]	▸ ["SI2"]
2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31	▸ ["Premium"]	▸ ["E"]	▸ ["SI1"]
3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31	▸ ["Good"]	▸ ["E"]	▸ ["VS1"]
4	0.29	Premium	I	VS2	62.4	58	334	4.2	4.23	2.63	▸ ["Premium"]	▸ ["I"]	▸ ["VS2"]

Showing the first 1000 rows.

```
# checking dataframe data types
diamondsa.dtypes
```

```
Out[17]:
[('_c0', 'int'),
 ('carat', 'float'),
 ('cut', 'string'),
 ('color', 'string'),
 ('clarity', 'string'),
 ('depth', 'float'),
 ('table', 'int'),
 ('price', 'float'),
 ('x', 'float'),
 ('y', 'float'),
 ('z', 'float'),
 ('cutVec', 'array<string>'),
 ('colorVec', 'array<string>'),
 ('clarityVec', 'array<string>'),
 ('cutVec2', 'vector'),
 ('colorVec2', 'vector'),
 ('clarityVec2', 'vector')]

# importing libraries
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline

# creating a features column with necessary variables
vecAssembler = VectorAssembler(inputCols=['_c0','carat', 'depth', 'cutVec2', 'colorVec2', 'clarityVec2', 'x',
'y', 'z', 'price'], outputCol="features")

# chosen regression type - linear regression
lr = (LinearRegression(maxIter=10,
elasticNetParam=0.8).setLabelCol("price").setFeaturesCol("features").setElasticNetParam(0.5))

# pipeline is included
pipeline = Pipeline(stages=[vecAssembler, lr])


# creating a variable to select necessary columns from data set
DF = diamondsa.select('_c0', 'carat', 'cutVec2', 'colorVec2', 'clarityVec2', 'depth', 'x', 'y', 'z', 'price')


# splitting the 'DF' dataset into training and test
training11, test11 = DF.randomSplit([0.7, 0.3])


# fitting the pipeline with the training set
model11 = pipeline.fit(training11)


# transforming the test set with the fitted pipeline training set
prediction11 = model11.transform(test11)
```

```
# adjusting the training set by adding a 'label' column
from pyspark.sql.functions import lit


training11 = training11.withColumn('label', lit(0))


# importing crossvalidator and paramgridbuilder libraries
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder



# defining the paramgrid
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.1, 0.01]) \
    .build()


# import multiclassclassificationevaluator since this dataset is not binary
from pyspark.ml.evaluation import MulticlassClassificationEvaluator


# defining crossval with params
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=MulticlassClassificationEvaluator(),
                          numFolds=10)


# defining the crossval model with the fitted training set
cvModel = crossval.fit(training11)
```

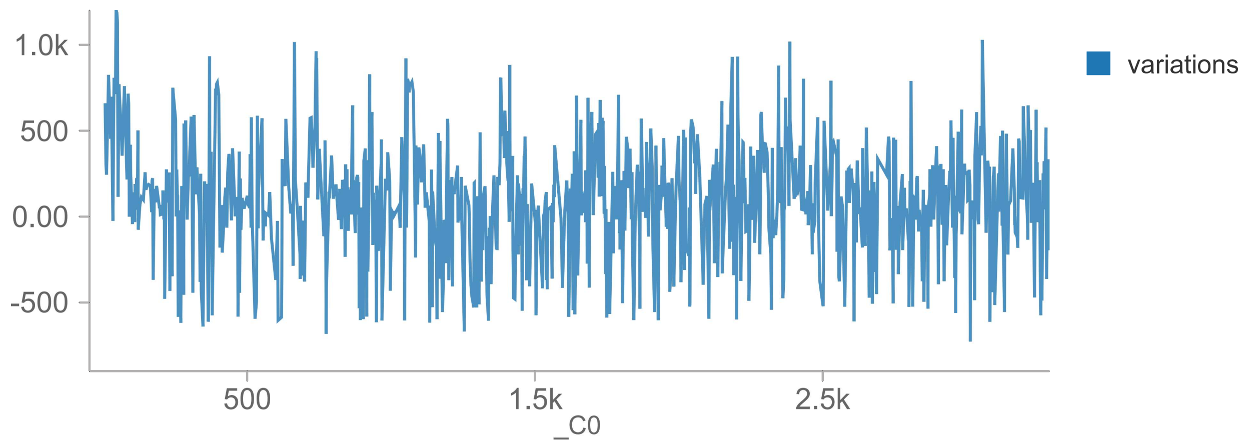
```
# adding the label column with the test set
from pyspark.sql.functions import lit


test11 = test11.withColumn('label', lit(0))


# prediction variable is defined with the cross val model and test set
prediction = cvModel.transform(test11)


# viewing the prediction dataframe
display(prediction)
```

_c0	carat	cutVec2
7	0.24	▸ [1,5,[], [-0.05374675989151001,-0.04417412355542183,-0.09714653342962265,-0.04203122854232788,-0.0021275400649756193]]
10	0.23	▸ [1,5,[], [-0.05374675989151001,-0.04417412355542183,-0.09714653342962265,-0.04203122854232788,-0.0021275400649756193]]
12	0.23	▸ [1,5,[], [0.03289894014596939,-0.05745129659771919,-0.013347899541258812,-0.02437640354037285,0.03119708225131035]]
19	0.3	▸ [1,5,[], [-0.06139902025461197,-0.04793813079595566,-0.05153780058026314,0.05547511577606201,0.02127288654446602]]
26	0.23	▸ [1,5,[],

Showing the first 1000 rows.

```
# creating a new variable to add a variations column by finding the differences between price and prediction
pred_withvariations = prediction.withColumn("variations", (prediction["price"] - prediction["prediction"]))
```

```
# taking a look at the visualized version of the difference between price and prediction for each entry
display(pred_withvariations)
```



Aggregated in the backend.
Showing the first 1000 rows.

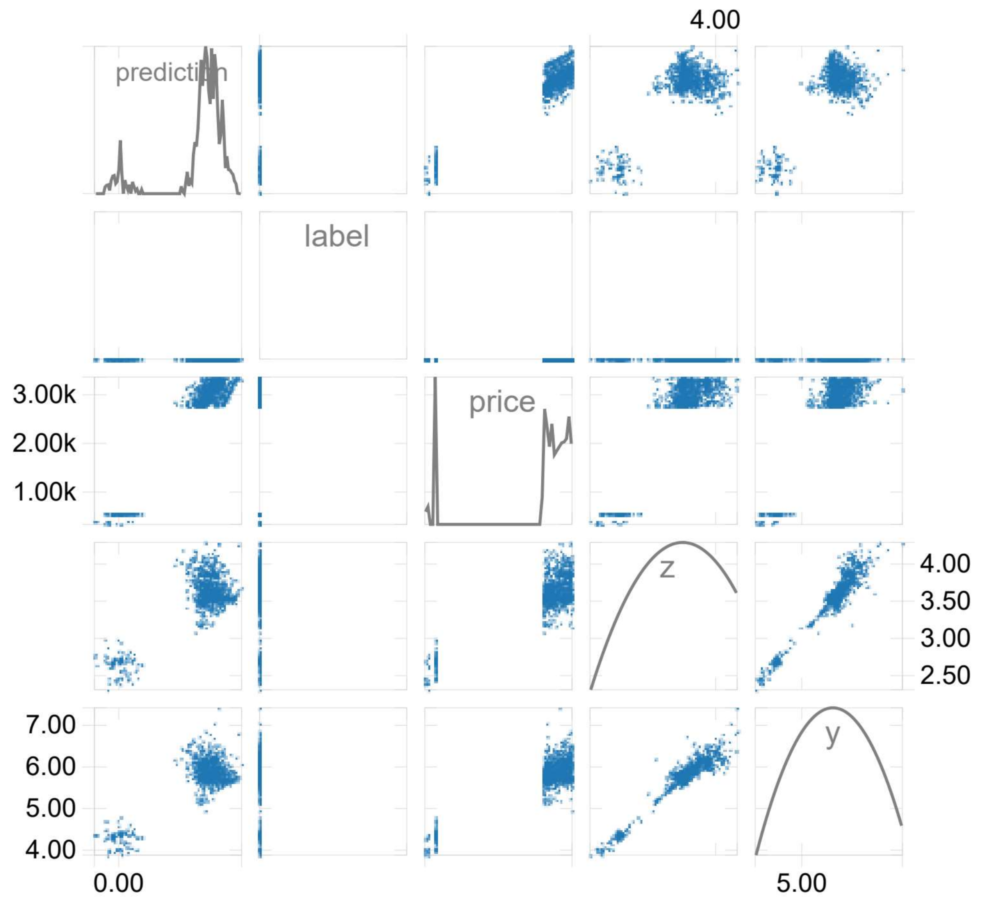
```
# viewing the configuration of 'sc'
sc
```

```
Out[34]: <SparkContext master=local[8] appName=Databricks Shell>
```

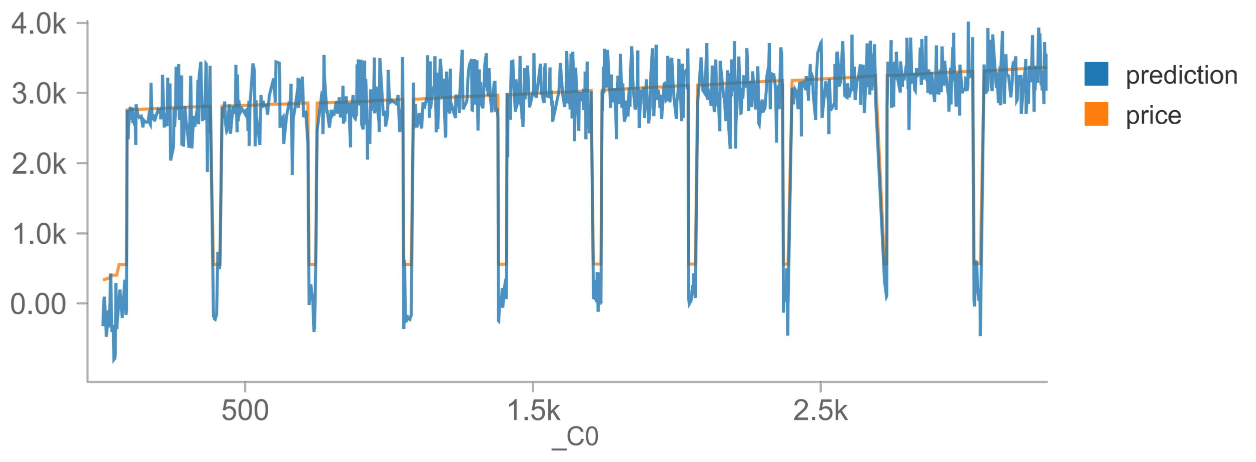
```
%python
DFF = prediction.select('prediction', 'price')
rdd = sc.parallelize(DFF.collect())

# creating a rdd of the prediction and price prediction dataframe



# importing RegressionMetrics and creating a variable to be used for statistical information
from pyspark.mllib.evaluation import RegressionMetrics
metrics = RegressionMetrics(rdd)



# importing squareroot to find the 'r-value'
from math import sqrt

# printing out the results from Regression Metrics
print("explained variance: ", metrics.explainedVariance)
print("root mean squared error: ", metrics.rootMeanSquaredError)

print("mean absolute error: ", metrics.meanAbsoluteError)
print("mean squared error: ", metrics.meanSquaredError)
print("r-squared value: ", metrics.r2)
print("r-value: ", sqrt(metrics.r2))

# looks like the r2 and r values are very similar and close to 1. x and y variables may be associated to each
other very strongly.
```

```
('explained variance: ', 15044395.48860137)
('root mean squared error: ', 374.67580128732357)
('mean absolute error: ', 285.22642932269696)
('mean squared error: ', 140381.95607029798)
('r-squared value: ', 0.9912818832544059)
('r-value: ', 0.9956313992911262)
```

```
# Scatter plot impression
display(prediction)
```



Showing sample based on the first 1000 rows.



Aggregated in the backend.
Showing the first 1000 rows.