Regression in PySpark on Databricks platform

- This notebook covers developing a machine learning model in `PySpark` framework on `Databricks`.
- Dataset used for analysis is Sarah Gets a Diamond (http://store.darden.virginia.edu/sarah-gets-a-diamond) taken from University of Virginia, Darden Business Publishing. I have purposefully and randomly removed values from 30 rows for two different columns, so that different data transformation techniques in PySpark framework can be applied.
- A regression model is developed on this dataset to calculate the price(Y variable) of a diamond based on its multiple physical attributes/features(X variables).
- Different data transformations, data imputation techniques, data aggregation in PySpark environment are applied for cleaning, imputing, pre-processing, and extracting summary statistics on model development data.
- Regression model developement and tuning is performed under PySpark framework.
- Additionaly, in this tutorial I have combined some pre-processing and modelling steps into a single model pipeline.

```
# importing libraries necessary for data transformation, pre-processing, ml
model developement etc

import pandas as pd
import numpy as np

from pyspark.sql.functions import desc, isnan, when, count, col, countDistinct,
regexp_replace
from pyspark.ml import Pipeline
from pyspark.ml.feature import Imputer, StringIndexer, OneHotEncoder,
VectorAssembler
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import RegressionEvaluator
```

1. Data import

```
# Model developement data is stored as Delta table stored in default database
of my Databricks workspace.
# Importing model development data as a spark dataframe, by using both SQL and
Spark commands

df_sql = '''SELECT * FROM default.reg_data_missing'''
df = spark.sql(df_sql)
print('Dataframe data type : {}'.format(type(df)))
print('Data Shape : {} Rows - {} Columns'.format((df.count()),
(len(df.columns))))
print('\nTop 5 rows')
print(df.show(5))
print('\nBottom 5 rows')
print(df.orderBy(desc('ID')).show(5))
print(df.show(5))
```

```
Dataframe data type : <class 'pyspark.sql.dataframe.DataFrame'>
Data Shape : 6000 Rows - 9 Columns

Top 5 rows
+---+------------+-----+-----+-------+------+--------+------+------+
| ID|Carat Weight|  Cut|Color|Clarity|Polish|Symmetry|Report| Price|
+---+------------+-----+-----+-------+------+--------+------+------+
|  1|         1.1|Ideal|    H|    SI1|    VG|      EX|   GIA|5169.0|
|  2|        0.83|Ideal|    H|    VS1|    ID|      ID|  AGSL|3470.0|
|  3|        0.85|Ideal|    H|    SI1|    EX|      EX|   GIA|3183.0|
|  4|        0.91|Ideal|    E|    SI1|    VG|      VG|   GIA|4370.0|
|  5|        0.83|Ideal|    G|    SI1|    EX|      EX|   GIA|3171.0|
+---+------------+-----+-----+-------+------+--------+------+------+
only showing top 5 rows

None

Bottom 5 rows
+----+------------+---------------+-----+-------+------+--------+------+------
-+
|  ID|Carat Weight|            Cut|Color|Clarity|Polish|Symmetry|Report|  Pric
```

```
print('All columns in the dataset with their datatypes')
print(df.dtypes)
```

```
All columns in the dataset with their datatypes
[('ID', 'int'), ('Carat Weight', 'double'), ('Cut', 'string'), ('Color', 'stri
```

```
ng'), ('Clarity', 'string'), ('Polish', 'string'), ('Symmetry', 'string'), ('R
eport', 'string'), ('Price', 'double')]
```

```
# replacing all the whitespaces with '_' in all the column names

column_names = [column.replace(' ', '_') for column in df.columns]
df = df.toDF(*column_names)
print(df.show(2))
```

```
+---+------------+-----+-----+-------+------+--------+------+------+
| ID|Carat_Weight|  Cut|Color|Clarity|Polish|Symmetry|Report| Price|
+---+------------+-----+-----+-------+------+--------+------+------+
|  1|         1.1|Ideal|    H|    SI1|    VG|      EX|   GIA|5169.0|
|  2|        0.83|Ideal|    H|    VS1|    ID|      ID|  AGSL|3470.0|
+---+------------+-----+-----+-------+------+--------+------+------+
only showing top 2 rows

None
```

2. Summary Statistics and Data Imputation on model development data

```
# collecting all the numerical and categorical variables and saving them in two
different lists

categorical_variables = [item[0] for item in df.dtypes if
item[1].startswith('string')]
print('All categorical variables')
print(categorical_variables)
numerical_variables = [item[0] for item in df.dtypes if
item[1].startswith('double') or item[1].startswith('int')]
print('All numeric variables')
print(numerical_variables)
```

```
All categorical variables
['Cut', 'Color', 'Clarity', 'Polish', 'Symmetry', 'Report']
All numeric variables
['ID', 'Carat_Weight', 'Price']
```

```
# counting null/nan/missing values in all the columns

dict_1 = dict(df.dtypes)
df_info = pd.DataFrame(dict_1.items(), columns=['Column', 'DataType'])
null_count = df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for
c in df.columns]).toPandas()
null_count = list(null_count.iloc[0,])
df_info['NullCount'] = null_count
print(df_info)

         Column DataType  NullCount
0            ID      int          0
1  Carat_Weight   double         30
2           Cut   string          0
3         Color   string          0
4       Clarity   string          0
5        Polish   string          0
6      Symmetry   string         30
7        Report   string          0
8         Price   double          0


# checking if any of the numerical columns has a missing values

missing_numerical_variables = list(df_info['Column'].loc[(df_info['NullCount']
> 0) & (df_info['DataType'] != 'string')])
missing_numerical_variables

Out[7]: ['Carat_Weight']


# summary statistics on all the numerical columns

print(df.select([column for column in numerical_variables if column !=
'ID']).describe().show())

+-------+------------------+------------------+
|summary|      Carat_Weight|             Price|
+-------+------------------+------------------+
|  count|              5970|              6000|
|   mean|1.3346180904522886|11791.579333333333|
| stddev|0.4757117705068891|10184.350050741188|
|    min|              0.75|            2184.0|
|    max|              2.91|          101561.0|
+-------+------------------+------------------+

None
```

```
# summary statistics on all the numerical columns

print(df.agg(*(countDistinct(col(c)).alias(c) for c in
categorical_variables)).show())
```

```
+---+-----+-------+------+--------+------+
|Cut|Color|Clarity|Polish|Symmetry|Report|
+---+-----+-------+------+--------+------+
|  5|    6|      7|     4|       4|     2|
+---+-----+-------+------+--------+------+

None
```

```
# counting the frequency of each sub-level for all the categorical columns :
null values identified

for column in categorical_variables:
    df.groupBy(column).count().orderBy('count').show()
```

```
+---------------+-----+
|            Cut|count|
+---------------+-----+
|           Fair|  129|
|Signature-Ideal|  253|
|           Good|  708|
|      Very Good| 2428|
|          Ideal| 2482|
+---------------+-----+

+-----+-----+
|Color|count|
+-----+-----+
|    D|  661|
|    E|  778|
|    I|  968|
|    F| 1013|
|    H| 1079|
|    G| 1501|
+-----+-----+
```

```
# replacing whitespace and hyphen characters in all the sub-level values for
all the categorical variables

for column in categorical_variables:
    df = df.withColumn('{}'.format(column), regexp_replace('{}'.format(column),
'[\\s-]', '_'))

for column in categorical_variables:
    df.select(column).distinct().show()
```

```
+---------------+
|            Cut|
+---------------+
|      Very_Good|
|Signature_Ideal|
|          Ideal|
|           Good|
|           Fair|
+---------------+

+-----+
|Color|
+-----+
|    F|
|    E|
|    D|
|    G|
|    I|
|    H|
+-----+
```

```
# checking all the rows where Carat_Weight value is missing

print(df.filter(col('Carat_Weight').isNull()).show())
```

```
+----+------------+---------------+-----+-------+------+--------+------+------
-+
|  ID|Carat_Weight|            Cut|Color|Clarity|Polish|Symmetry|Report|  Pric
e|
+----+------------+---------------+-----+-------+------+--------+------+------
-+
| 121|        null|           Good|    H|    SI1|    EX|      EX|   GIA| 4563.
0|
| 405|        null|           Good|    I|    SI1|    EX|       G|   GIA|15743.
0|
```

```
| 591|        null|           Fair|    D|    SI1|    EX|      EX|   GIA| 4537.
0|
| 640|        null|Signature_Ideal|    I|    SI1|    ID|      ID|  AGSL|11110.
0|
| 816|        null|      Very_Good|    H|    VS2|    EX|      VG|   GIA| 5571.
0|
| 878|        null|      Very_Good|    I|    SI1|     G|    null|   GIA| 4664.
0|
|1162|        null|      Very_Good|    I|    VS2|    VG|      VG|   GIA|15895.
0|
```

```
# imputing the missing Carat_Weight rows using PySpark's Imputer functions
df_impute = df
imputer = Imputer(inputCols=missing_numerical_variables, outputCols=
['{}_imputed'.format(c) for c in missing_numerical_variables])
df_impute = imputer.fit(df).transform(df_impute)

# checking all the rows where Carat_Weight value is missing : all missing
values imputed
print(df_impute.show(5))
```

```
+---+------------+-----+-----+-------+------+--------+------+------+----------
----------+
| ID|Carat_Weight|  Cut|Color|Clarity|Polish|Symmetry|Report| Price|Carat_Weig
ht_imputed|
+---+------------+-----+-----+-------+------+--------+------+------+----------
----------+
|  1|         1.1|Ideal|    H|    SI1|    VG|      EX|   GIA|5169.0|
1.1|
|  2|        0.83|Ideal|    H|    VS1|    ID|      ID|  AGSL|3470.0|
0.83|
|  3|        0.85|Ideal|    H|    SI1|    EX|      EX|   GIA|3183.0|
0.85|
|  4|        0.91|Ideal|    E|    SI1|    VG|      VG|   GIA|4370.0|
0.91|
|  5|        0.83|Ideal|    G|    SI1|    EX|      EX|   GIA|3171.0|
0.83|
+---+------------+-----+-----+-------+------+--------+------+------+----------
----------+
only showing top 5 rows

None
```

```
# checking all the rows where Symmetry value is missing
print(df_impute.filter(col('Carat_Weight').isNull()).show(5))

# dropping the column Carat_Weight as all the original and imputed values are
saved in Carat_Weight_imputed column
df_impute = df_impute.drop('Carat_Weight')
```

```
+---+------------+---------------+-----+-------+------+--------+------+-------
+--------------------+
| ID|Carat_Weight|            Cut|Color|Clarity|Polish|Symmetry|Report|  Price
|Carat_Weight_imputed|
+---+------------+---------------+-----+-------+------+--------+------+-------
+--------------------+
|121|        null|           Good|    H|    SI1|    EX|      EX|   GIA| 4563.0
|  1.3346180904522886|
|405|        null|           Good|    I|    SI1|    EX|       G|   GIA|15743.0
|  1.3346180904522886|
|591|        null|           Fair|    D|    SI1|    EX|      EX|   GIA| 4537.0
|  1.3346180904522886|
|640|        null|Signature_Ideal|    I|    SI1|    ID|      ID|  AGSL|11110.0
|  1.3346180904522886|
|816|        null|      Very_Good|    H|    VS2|    EX|      VG|   GIA| 5571.0
|  1.3346180904522886|
+---+------------+---------------+-----+-------+------+--------+------+-------
+--------------------+
only showing top 5 rows

None
```

```
print(df_impute.filter(col('Symmetry').isNull()).show())
```

```
+----+---------+-----+-------+------+--------+------+-------+-----------------
---+
|  ID|      Cut|Color|Clarity|Polish|Symmetry|Report|  Price|Carat_Weight_impu
ted|
+----+---------+-----+-------+------+--------+------+-------+-----------------
---+
|  71|    Ideal|    G|    VS2|    EX|    null|   GIA| 6264.0|
1.02|
| 141|    Ideal|    I|    SI1|    EX|    null|   GIA|16718.0|
2.04|
| 191|Very_Good|    G|   VVS2|     G|    null|   GIA| 8013.0|
1.02|
| 244|    Ideal|    F|     IF|    EX|    null|   GIA|11327.0|
1.07|
| 309|    Ideal|    H|    SI1|    VG|    null|   GIA| 5529.0|
```

```
1.15|
| 878|Very_Good|    I|    SI1|     G|    null|   GIA| 4664.0|  1.3346180904522
886|
| 975|     Good|    F|   VVS2|    EX|    null|   GIA|17203.0|
1.51|
```

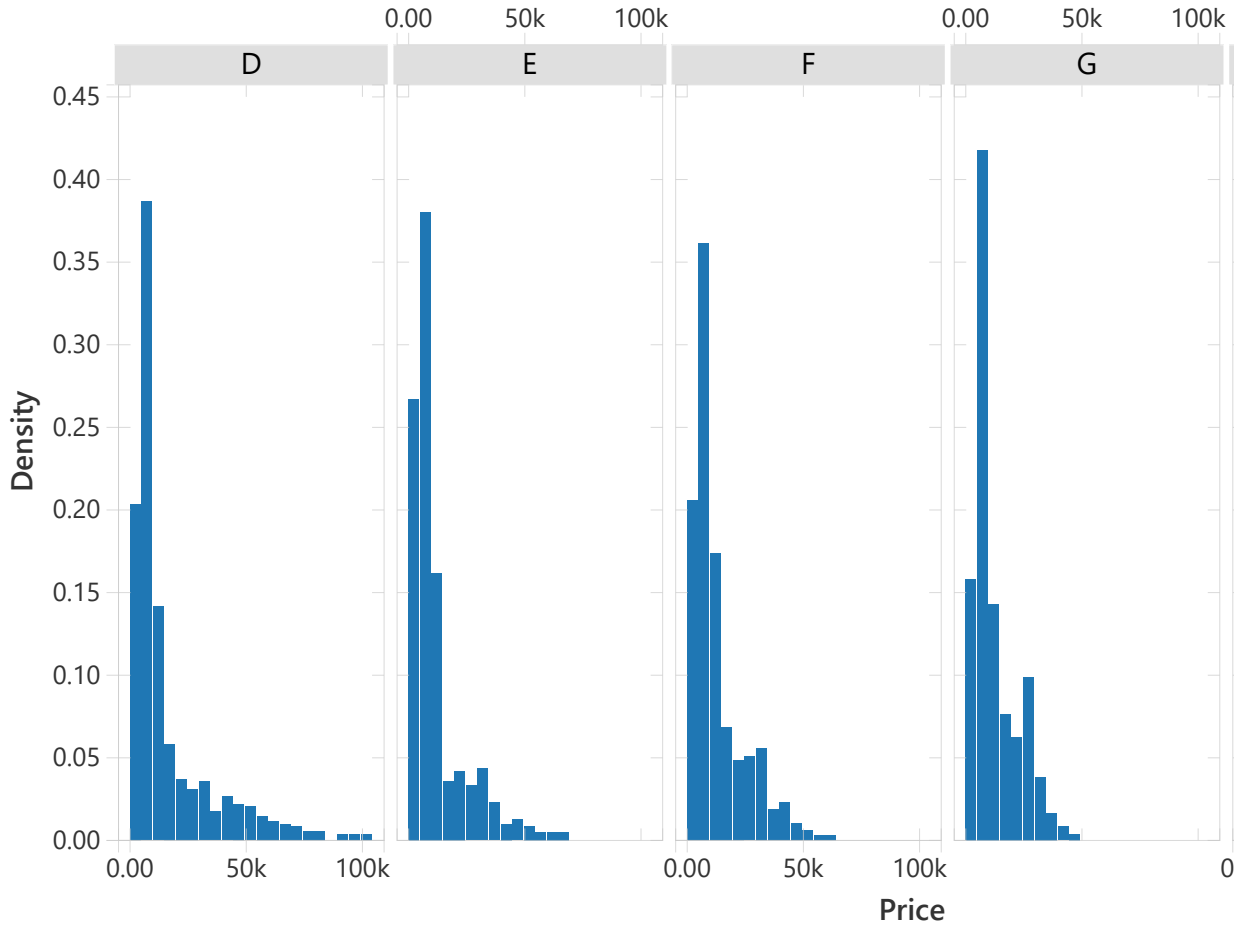
```
# imputing all missing Symmetry values with mode Symmetry value : VG in this
case

temp_df= df_impute.groupBy('Symmetry').count()
mode_variable = temp_df.orderBy(temp_df['count'].desc()).collect()[0][0]
mode_count_value = temp_df.orderBy(temp_df['count'].desc()).collect()[0][1]
print('Mode Value : {}\n'.format(mode_variable), 'Mode Count :
{}'.format(mode_count_value))
df_impute = df_impute.fillna({'Symmetry':mode_variable})
df_impute = df_impute.withColumnRenamed('Symmetry', 'Symmetry_imputed')
print(df_impute.show(5))
```

```
Mode Value : VG
 Mode Count : 2401
+---+-----+-----+-------+------+---------------+------+------+---------------
-----+
| ID|  Cut|Color|Clarity|Polish|Symmetry_imputed|Report| Price|Carat_Weight_im
puted|
+---+-----+-----+-------+------+---------------+------+------+---------------
-----+
|  1|Ideal|    H|    SI1|    VG|              EX|   GIA|5169.0|
1.1|
|  2|Ideal|    H|    VS1|    ID|              ID|  AGSL|3470.0|
0.83|
|  3|Ideal|    H|    SI1|    EX|              EX|   GIA|3183.0|
0.85|
|  4|Ideal|    E|    SI1|    VG|              VG|   GIA|4370.0|
0.91|
|  5|Ideal|    G|    SI1|    EX|              EX|   GIA|3171.0|
0.83|
+---+-----+-----+-------+------+---------------+------+------+---------------
-----+
only showing top 5 rows
```

3. Visual analysis using some SQL and Databricks inbuilt plot functions

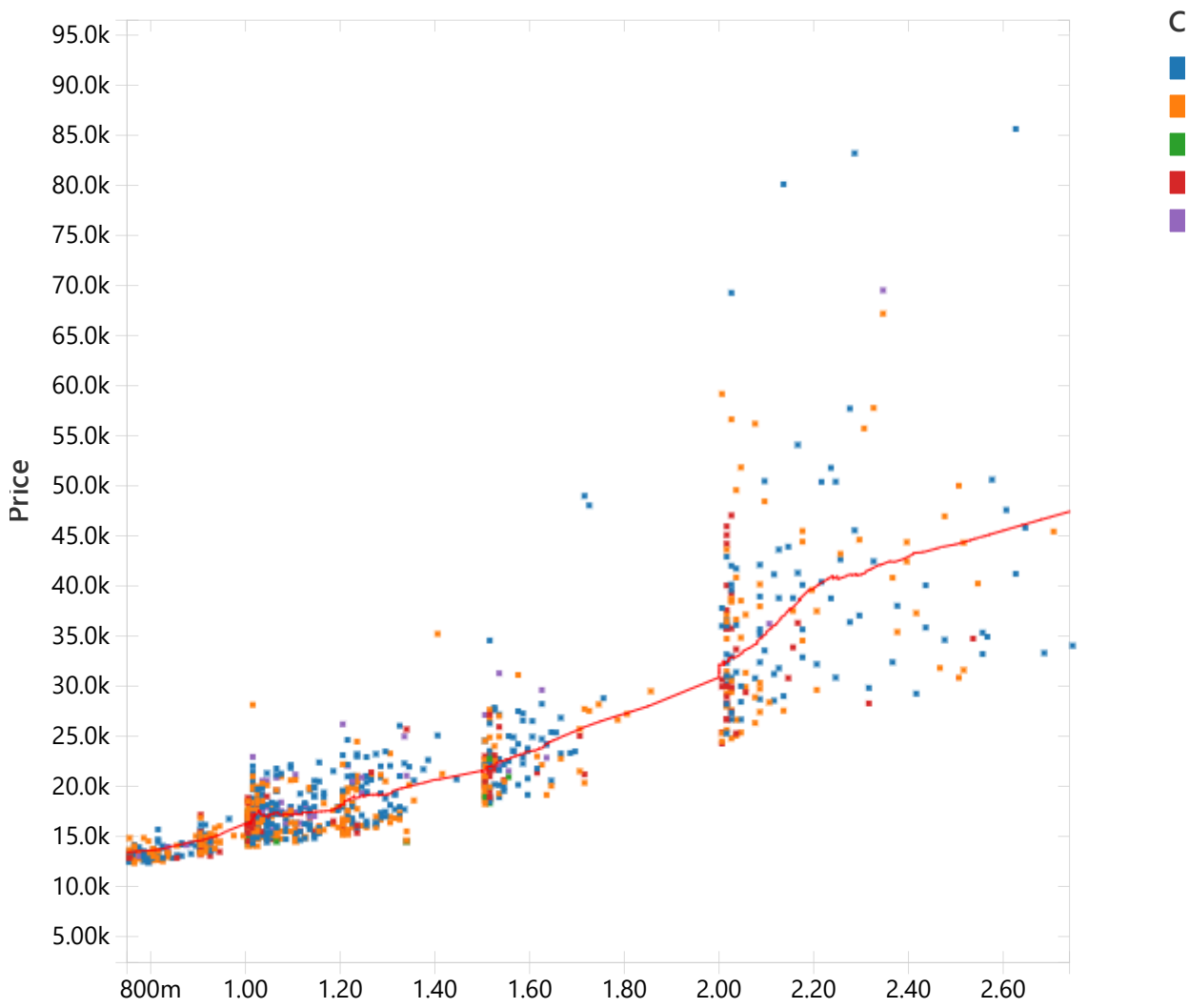
```
# creating separate lists for categorical and numerical variables

categorical_variables = [item[0] for item in df_impute.dtypes if
item[1].startswith('string')]
numerical_variables = [item[0] for item in df_impute.dtypes if
item[1].startswith('double') or item[1].startswith('int')]
```

```
%sql
SELECT ID, Price, Cut, Color
FROM default.reg_data_missing;
```



Aggregated (by count) in the backend

```
display(df_impute)
```



4. Data Pre-processing and Model developement

```
# Splitting model development data into train-test data

df_train, df_test = df_impute.randomSplit([0.8, 0.2], seed=21)
(df_train.cache().count(), df_test.count())
```

```
Out[19]: (4746, 1254)
```

```
# creating StringIndexer and OneHotEncoder objects to convert encode
categorical data
# creating VectorAssembler objects to collect all the encoded and numerical
features into a sparse vector
# saving all the objects into a 'stage' list, which will be passed into a model
developement pipeline

stages = []

for column in categorical_variables:
    string_indexer = StringIndexer(inputCol = column, outputCol =
column+'_Index')
    oh_encoder = OneHotEncoder(inputCols=[string_indexer.getOutputCol()],
outputCols=[column+'_classVec'], dropLast=True)
    stages +=[string_indexer, oh_encoder]

assemble_inputs = [c+'_classVec' for c in categorical_variables] +
numerical_variables
assemble_inputs.remove('ID')
vector_assembler = VectorAssembler(inputCols=assemble_inputs,
outputCol='x_features')
stages +=[vector_assembler]


# creating an object for a base Gradient Boosting regressor model
gbt = GBTRegressor(labelCol='Price', featuresCol='x_features', maxIter=15,
maxDepth=5, seed=21)

# adding model object to 'stages' list
stages.append(gbt)

# passing the 'stages' to a Pipeline object
pipeline = Pipeline(stages=stages)
print('Pipeline with satges')
print(pipeline)
print(stages)

Pipeline with satges
Pipeline_9ad1332d7939
[StringIndexer_adf2904cb17a, OneHotEncoder_ea22da22ef55, StringIndexer_adc6af6
25fee, OneHotEncoder_76738f3743c0, StringIndexer_84c0d334203a, OneHotEncoder_0
310e5531a97, StringIndexer_8b9906fc1c84, OneHotEncoder_425ac27bc92e, StringInd
exer_917bf2c8a455, OneHotEncoder_97fdd39e2fd0, StringIndexer_b092586198d4, One
HotEncoder_cbd148b27d36, VectorAssembler_5dd87a1b3005, GBTRegressor_9ff002c9b0
ca]
```

```
# fitting the Pipeline object on train data : all the pre-processing and and
model fitting will occur in this step
pipelineModel = pipeline.fit(df_train)

# fitting the trained pre-processing and model-fit pipeline on test data to
calculate predictions on test data
gbt_predictions = pipelineModel.transform(df_test)

# selecting variables to check the predictions on test data
gbt_predictions.select('ID', 'x_features', 'Price', 'prediction').take(3)

Out[22]: [Row(ID=12, x_features=SparseVector(24, {2: 1.0, 8: 1.0, 9: 1.0, 20:
1.0, 21: 1.0, 22: 5161.0, 23: 1.01}), Price=5161.0, prediction=5306.9535728705
9),
 Row(ID=22, x_features=SparseVector(24, {1: 1.0, 6: 1.0, 11: 1.0, 15: 1.0, 20:
1.0, 21: 1.0, 22: 3559.0, 23: 0.8}), Price=3559.0, prediction=3813.52864952471
24),
 Row(ID=41, x_features=SparseVector(24, {2: 1.0, 4: 1.0, 9: 1.0, 16: 1.0, 20:
1.0, 21: 1.0, 22: 4905.0, 23: 1.01}), Price=4905.0, prediction=4566.6853523534
1)]


# defining MAE as metric to evaluate model performance
# evaluating model predictions on test-data using Mean Absolute Error metric

gbt_evaluator = RegressionEvaluator(metricName='mae',
labelCol=gbt.getLabelCol(), predictionCol=gbt.getPredictionCol())
mae = round((gbt_evaluator.evaluate(gbt_predictions)),2)
print('Test MAE = {}'.format(mae))

Test MAE = 594.62
```

```
# defining model hyperparameter grid
parameter_grid = (ParamGridBuilder()
                  .addGrid(gbt.maxDepth, [4,5,6])
                  .addGrid(gbt.maxIter, [10, 15, 20])
                  .build())

# defining crosss-validation object with model, hyparameter grid, and evaluator
object values
cv = CrossValidator(estimator=pipeline, evaluator=gbt_evaluator,
estimatorParamMaps=parameter_grid, numFolds=3, parallelism = 4)

# fitting model on train data and calculating predictions on test data
gbt_cv_model = cv.fit(df_train)
gbt_cv_predictions = gbt_cv_model.transform(df_test)
gbt_cv_predictions.select('ID', 'x_features', 'Price', 'prediction').take(3)

Out[24]: [Row(ID=12, x_features=SparseVector(24, {2: 1.0, 8: 1.0, 9: 1.0, 20:
1.0, 21: 1.0, 22: 5161.0, 23: 1.01}), Price=5161.0, prediction=5068.8561038184
82),
 Row(ID=22, x_features=SparseVector(24, {1: 1.0, 6: 1.0, 11: 1.0, 15: 1.0, 20:
1.0, 21: 1.0, 22: 3559.0, 23: 0.8}), Price=3559.0, prediction=3732.51773107623
24),
 Row(ID=41, x_features=SparseVector(24, {2: 1.0, 4: 1.0, 9: 1.0, 16: 1.0, 20:
1.0, 21: 1.0, 22: 4905.0, 23: 1.01}), Price=4905.0, prediction=4687.2772597766
725)]

Test MAE = 525.19
```