Big Data: Improve performance of an RF Classifier

Azhar Chowdhury
Statistical Programmer|Data Scientist[SAS, Python, R]

The following meta summary is helpful to follow the problem at hand and it's solution.

Dataset the Farmers Markets dataset https://catalog.data.gov/dataset/farmers-markets-directory-and-geographic-data

Domain Farmer's Market Data

Operations Application of ML models on Big Data

Components Using ML Pipeline, hyper-parameter tuning, Improve the performance of a ML model.

Problem Description Applying ML models to a dataset to observe improvements on model's hyper parameter tuning, feature engineering and overall performance of an existing ML model.

About the Problem

We have identified a fairly significant link by leveraging the ML pipeline, a more sophisticated model, and better hyperparameter tuning. However these results are still a bit disappointing. With that being said, we're working with very few features and we've likely made some assumptions that just aren't quite valid (like zip code shortening). Also, just because a rich zip code exists doesn't mean that the farmer's market would be held in that zip code too. In fact we might want to start looking at neighboring zip codes or doing some sort of distance measure to predict whether or not there exists a farmer's market in a certain mile radius from a wealthy zip code.

With that being said, we've got a lot of other potential features and plenty of other parameters to tune on our random forest so play around with the above pipeline and see if you can improve it further! Note: adding a feaure for the distance measure is just an example and not a mandatory change to improve the model's performance. We also aren't concerned about if the model's perforamnce is actually improved! We simply want to see if changes have been made to the code for possible improvements.

[1]:
```
# install necessary libraries
!pip install pyspark
```

```
Requirement already satisfied: pyspark in c:\users\azhar\anaconda3\lib\site-
packages (3.5.5)
Requirement already satisfied: py4j==0.10.9.7 in
c:\users\azhar\anaconda3\lib\site-packages (from pyspark) (0.10.9.7)
```

[2]:
```
# Improve a Random Forest Classifier:

# import libraries and modules.
from pyspark.sql.types import IntegerType
from pyspark.sql import SparkSession

#Create a Spark object:
spark = SparkSession.builder.config("spark.driver.host", "localhost").
 ↪appName("Read CSV").getOrCreate()
from pyspark.sql.types import IntegerType
```

[3]:
```
# Read datasets and create spark objects
diamonds = "C://Users//azhar//Downloads//diamonds.csv"
market_data = "C://Users//azhar//Downloads//market_data.csv"
tax = "C://Users//azhar//Downloads//2013_soi_zipcode_agi.csv"

taxes2013 = spark.read.csv(tax,          header=True, inferSchema=True)
diamonds  = spark.read.csv(diamonds,     header=True, inferSchema=True)
market    = spark.read.csv(market_data,  header=True, inferSchema=True)
```

[4]:
```
# Register Tax and Market data to Spark SQL table
taxes2013.createOrReplaceTempView("taxes2013")
market.createOrReplaceTempView("market")
taxes2013.schema  # check the dataset's structure/metadata
```

[4]:
```
StructType([StructField('STATEFIPS', IntegerType(), True), StructField('STATE',
StringType(), True), StructField('zipcode', IntegerType(), True),
StructField('agi_stub', IntegerType(), True), StructField('N1', DoubleType(),
True), StructField('MARS1', DoubleType(), True), StructField('MARS2',
DoubleType(), True), StructField('MARS4', DoubleType(), True),
StructField('PREP', DoubleType(), True), StructField('N2', DoubleType(), True),
StructField('NUMDEP', DoubleType(), True), StructField('A00100', DoubleType(),
True), StructField('N02650', DoubleType(), True), StructField('A02650',
DoubleType(), True), StructField('N00200', DoubleType(), True),
StructField('A00200', DoubleType(), True), StructField('N00300', DoubleType(),
True), StructField('A00300', DoubleType(), True), StructField('N00600',
DoubleType(), True), StructField('A00600', DoubleType(), True),
StructField('N00650', DoubleType(), True), StructField('A00650', DoubleType(),
True), StructField('N00700', DoubleType(), True), StructField('A00700',
DoubleType(), True), StructField('N00900', DoubleType(), True),
StructField('A00900', DoubleType(), True), StructField('N01000', DoubleType(),
True), StructField('A01000', DoubleType(), True), StructField('N01400',
DoubleType(), True), StructField('A01400', DoubleType(), True),
StructField('N01700', DoubleType(), True), StructField('A01700', DoubleType(),
True), StructField('SCHF', DoubleType(), True), StructField('N02300',
DoubleType(), True), StructField('A02300', DoubleType(), True),
StructField('N02500', DoubleType(), True), StructField('A02500', DoubleType(),
```

```
True), StructField('N26270', DoubleType(), True), StructField('A26270',
DoubleType(), True), StructField('N02900', DoubleType(), True),
StructField('A02900', DoubleType(), True), StructField('N03220', DoubleType(),
True), StructField('A03220', DoubleType(), True), StructField('N03300',
DoubleType(), True), StructField('A03300', DoubleType(), True),
StructField('N03270', DoubleType(), True), StructField('A03270', DoubleType(),
True), StructField('N03150', DoubleType(), True), StructField('A03150',
DoubleType(), True), StructField('N03210', DoubleType(), True),
StructField('A03210', DoubleType(), True), StructField('N03230', DoubleType(),
True), StructField('A03230', DoubleType(), True), StructField('N03240',
DoubleType(), True), StructField('A03240', DoubleType(), True),
StructField('N04470', DoubleType(), True), StructField('A04470', DoubleType(),
True), StructField('A00101', DoubleType(), True), StructField('N18425',
DoubleType(), True), StructField('A18425', DoubleType(), True),
StructField('N18450', DoubleType(), True), StructField('A18450', DoubleType(),
True), StructField('N18500', DoubleType(), True), StructField('A18500',
DoubleType(), True), StructField('N18300', DoubleType(), True),
StructField('A18300', DoubleType(), True), StructField('N19300', DoubleType(),
True), StructField('A19300', DoubleType(), True), StructField('N19700',
DoubleType(), True), StructField('A19700', DoubleType(), True),
StructField('N04800', DoubleType(), True), StructField('A04800', DoubleType(),
True), StructField('N05800', DoubleType(), True), StructField('A05800',
DoubleType(), True), StructField('N09600', DoubleType(), True),
StructField('A09600', DoubleType(), True), StructField('N07100', DoubleType(),
True), StructField('A07100', DoubleType(), True), StructField('N07300',
DoubleType(), True), StructField('A07300', DoubleType(), True),
StructField('N07180', DoubleType(), True), StructField('A07180', DoubleType(),
True), StructField('N07230', DoubleType(), True), StructField('A07230',
DoubleType(), True), StructField('N07240', DoubleType(), True),
StructField('A07240', DoubleType(), True), StructField('N07220', DoubleType(),
True), StructField('A07220', DoubleType(), True), StructField('N07260',
DoubleType(), True), StructField('A07260', DoubleType(), True),
StructField('N09400', DoubleType(), True), StructField('A09400', DoubleType(),
True), StructField('N10600', DoubleType(), True), StructField('A10600',
DoubleType(), True), StructField('N59660', DoubleType(), True),
StructField('A59660', DoubleType(), True), StructField('N59720', DoubleType(),
True), StructField('A59720', DoubleType(), True), StructField('N11070',
DoubleType(), True), StructField('A11070', DoubleType(), True),
StructField('N10960', DoubleType(), True), StructField('A10960', DoubleType(),
True), StructField('N06500', DoubleType(), True), StructField('A06500',
DoubleType(), True), StructField('N10300', DoubleType(), True),
StructField('A10300', DoubleType(), True), StructField('N85330', DoubleType(),
True), StructField('A85330', DoubleType(), True), StructField('N85300',
DoubleType(), True), StructField('A85300', DoubleType(), True),
StructField('N11901', DoubleType(), True), StructField('A11901', DoubleType(),
True), StructField('N11902', DoubleType(), True), StructField('A11902',
DoubleType(), True)])
```

[5]:
```
cleaned_taxes = taxes2013.select('state', (taxes2013.zipcode / 10).
 ↪cast(IntegerType()).alias('zipcode'), \
                                 (taxes2013.MARS1).cast(IntegerType()).
 ↪alias('single_returns'), (taxes2013.MARS2)\
                                 .cast(IntegerType()).
 ↪alias('joint_returns'),(taxes2013.NUMDEP).cast(IntegerType())\
                                 .alias('numdep'))
display(cleaned_taxes)
```

```
DataFrame[state: string, zipcode: int, single_returns: int, joint_returns: int,␣
 ↪numdep: int]
```

[6]:
```
# Convert back to a dataset from a table
summedTaxes = cleaned_taxes.groupBy("zipcode").sum() # because of AGI, where␣
 ↪groups income groups are broken out

cleanedMarkets = (market
  .selectExpr("*", "int(zip / 10) as zipcode")
  .groupBy("zipcode")
  .count()
  .selectExpr("double(count) as count", "zipcode as zip"))
#  selectExpr is short for Select Expression - equivalent to what we
#  might be doing in SQL SELECT expression

joined = (cleanedMarkets.join(summedTaxes, cleanedMarkets.zip == summedTaxes.
 ↪zipcode, "outer"))
```

[7]:
```
display(summedTaxes)
```

```
DataFrame[zipcode: int, sum(zipcode): bigint, sum(single_returns): bigint,␣
 ↪sum(joint_returns): bigint, sum(numdep): bigint]
```

[8]:
```
display(cleanedMarkets)
```

```
DataFrame[count: double, zip: int]
```

[9]:
```
display(joined)
```

```
DataFrame[count: double, zip: int, zipcode: int, sum(zipcode): bigint,␣
 ↪sum(single_returns): bigint, sum(joint_returns): bigint, sum(numdep): bigint]
```

[10]:
```
# imputing missing values with 0
prepped = joined.na.fill(0)
display(prepped)
prepped.schema  # final dataset to be used in the RFC
```

```
DataFrame[count: double, zip: int, zipcode: int, sum(zipcode): bigint,␣
 ↪sum(single_returns): bigint, sum(joint_returns): bigint, sum(numdep): bigint]
```

[10]:
```
StructType([StructField('count', DoubleType(), False), StructField('zip',
IntegerType(), True), StructField('zipcode', IntegerType(), True),
StructField('sum(zipcode)', LongType(), True),
StructField('sum(single_returns)', LongType(), True),
StructField('sum(joint_returns)', LongType(), True), StructField('sum(numdep)',
LongType(), True)])
```

[11]:
```
# Data Preprocessing:
    # Imputing missing values (If any leftover at all) - Fill missing values␣
 ↪(fillna)
    # Encode Categorical Columns - Convert categorical columns using␣
 ↪StringIndexer

from pyspark.sql.functions import col
from pyspark.ml.feature import StringIndexer, VectorAssembler

categorical_columns = [col_name for col_name, dtype in prepped.dtypes if dtype␣
 ↪== "string"]
display(categorical_columns) # we don't have any categorical columns in the␣
 ↪'joined' data frame displays an empty list.

indexers = [StringIndexer(inputCol=col, outputCol=col + "_indexed",␣
 ↪handleInvalid="keep") for col in categorical_columns]

feature_columns = [col for col in prepped.columns if col not in ["zip", "id"]]
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
```

[]

[12]:
```
# Defining  pipeline
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier

# Define the Random Forest model
rf = RandomForestClassifier(labelCol="zip", featuresCol="features", seed=42)

# Create a pipeline with preprocessing and model
pipeline = Pipeline(stages=indexers + [assembler, rf])
```

[13]:
```
# Hyperparameter Tuning with Cross-Validation

from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Define Hyperparameter Grid
paramGrid = ParamGridBuilder() \
    .addGrid(rf.numTrees, [10, 20, 50]) \
```

```
    .addGrid(rf.maxDepth, [5, 10, 15]) \
    .build()

# Define Evaluator
evaluator = MulticlassClassificationEvaluator(labelCol="target",␣
 ↪metricName="accuracy")

# Apply CrossValidator
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=3)  # 3-Fold Cross Validation
```

[14]:
```
# Train and Evaluate the performance (accuracy) of the model
# Split data into train and test
train_prep, test_prep = prepped.randomSplit([0.8, 0.2], seed=42)

# Train model with hyperparameter tuning
cv_model = crossval.fit(train_prep)

# Get best model
best_model = cv_model.bestModel

# Make predictions
predictions = best_model.transform(test_prep)

# Evaluate accuracy
accuracy = evaluator.evaluate(predictions)

print(f"Test Accuracy: {accuracy:.4f}")
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
Cell In[14], line 6
      3 train_prep, test_prep = prepped.randomSplit([0.8, 0.2], seed=42)
      5 # Train model with hyperparameter tuning
----> 6 cv_model = crossval.fit(train_prep)
      8 # Get best model
      9 best_model = cv_model.bestModel

File ~\anaconda3\Lib\site-packages\pyspark\ml\base.py:205, in Estimator.fit(self,␣
 ↪dataset, params)
    203         return self.copy(params)._fit(dataset)
    204     else:
--> 205         return self._fit(dataset)
    206 else:
    207     raise TypeError(
```

```
    208         "Params must be either a param map or a list/tuple of param maps, "
    209         "but got %s." % type(params)
    210     )

File ~\anaconda3\Lib\site-packages\pyspark\ml\tuning.py:847, in CrossValidator._fit(self, dataset)
    841 train = datasets[i][0].cache()
    843 tasks = map(
    844     inheritable_thread_target,
    845     _parallelFitTasks(est, train, eva, validation, epm, collectSubModelsParam),
    846 )
--> 847 for j, metric, subModel in pool.imap_unordered(lambda f: f(), tasks):
    848     metrics_all[i][j] = metric
    849     if collectSubModelsParam:

File ~\anaconda3\Lib\multiprocessing\pool.py:873, in IMapIterator.next(self, timeout)
    871 if success:
    872     return value
--> 873 raise value

File ~\anaconda3\Lib\multiprocessing\pool.py:125, in worker(inqueue, outqueue, initializer, initargs, maxtasks, wrap_exception)
    123 job, i, func, args, kwds = task
    124 try:
--> 125     result = (True, func(*args, **kwds))
    126 except Exception as e:
    127     if wrap_exception and func is not _helper_reraises_exception:

File ~\anaconda3\Lib\site-packages\pyspark\ml\tuning.py:847, in CrossValidator._fit.<locals>.<lambda>(f)
    841 train = datasets[i][0].cache()
    843 tasks = map(
    844     inheritable_thread_target,
    845     _parallelFitTasks(est, train, eva, validation, epm, collectSubModelsParam),
    846 )
--> 847 for j, metric, subModel in pool.imap_unordered(lambda f: f(), tasks):
    848     metrics_all[i][j] = metric
    849     if collectSubModelsParam:

File ~\anaconda3\Lib\site-packages\pyspark\util.py:342, in inheritable_thread_target.<locals>.wrapped(*args, **kwargs)
    340 assert SparkContext._active_spark_context is not None
    341 SparkContext._active_spark_context._jsc.sc().setLocalProperties(properties)
--> 342 return f(*args, **kwargs)
```

```
File ~\anaconda3\Lib\site-packages\pyspark\ml\tuning.py:113, in _parallelFitTasks.
 ↪<locals>.singleTask()
    112 def singleTask() -> Tuple[int, float, Transformer]:
--> 113     index, model = next(modelIter)
    114     # TODO: duplicate evaluator to take extra params from input
    115     #  Note: Supporting tuning params in evaluator need update method
    116     #  `MetaAlgorithmReadWrite.getAllNestedStages`, make it return
    117     #  all nested stages and evaluators
    118     metric = eva.evaluate(model.transform(validation, epm[index]))

File ~\anaconda3\Lib\site-packages\pyspark\ml\base.py:98, in _FitMultipleIterator.
 ↪__next__(self)
     96         raise StopIteration("No models remaining.")
     97     self.counter += 1
---> 98 return index, self.fitSingleModel(index)

File ~\anaconda3\Lib\site-packages\pyspark\ml\base.py:156, in Estimator.
 ↪fitMultiple.<locals>.fitSingleModel(index)
    155 def fitSingleModel(index: int) -> M:
--> 156     return estimator.fit(dataset, paramMaps[index])

File ~\anaconda3\Lib\site-packages\pyspark\ml\base.py:203, in Estimator.fit(self,
 ↪dataset, params)
    201 elif isinstance(params, dict):
    202     if params:
--> 203         return self.copy(params)._fit(dataset)
    204     else:
    205         return self._fit(dataset)

File ~\anaconda3\Lib\site-packages\pyspark\ml\pipeline.py:134, in Pipeline.
 ↪_fit(self, dataset)
    132     dataset = stage.transform(dataset)
    133 else:  # must be an Estimator
--> 134     model = stage.fit(dataset)
    135     transformers.append(model)
    136     if i < indexOfLastEstimator:

File ~\anaconda3\Lib\site-packages\pyspark\ml\base.py:205, in Estimator.fit(self,
 ↪dataset, params)
    203         return self.copy(params)._fit(dataset)
    204     else:
--> 205         return self._fit(dataset)
    206 else:
    207     raise TypeError(
    208         "Params must be either a param map or a list/tuple of param maps, "
    209         "but got %s." % type(params)
    210     )
```

```
File ~\anaconda3\Lib\site-packages\pyspark\ml\wrapper.py:381, in JavaEstimator.
 ↪_fit(self, dataset)
    380 def _fit(self, dataset: DataFrame) -> JM:
--> 381     java_model = self._fit_java(dataset)
    382     model = self._create_model(java_model)
    383     return self._copyValues(model)

File ~\anaconda3\Lib\site-packages\pyspark\ml\wrapper.py:378, in JavaEstimator.
 ↪_fit_java(self, dataset)
    375 assert self._java_obj is not None
    377 self._transfer_params_to_java()
--> 378 return self._java_obj.fit(dataset._jdf)

File ~\anaconda3\Lib\site-packages\py4j\java_gateway.py:1322, in JavaMember.
 ↪__call__(self, *args)
   1316 command = proto.CALL_COMMAND_NAME +\
   1317     self.command_header +\
   1318     args_command +\
   1319     proto.END_COMMAND_PART
   1321 answer = self.gateway_client.send_command(command)
-> 1322 return_value = get_return_value(
   1323     answer, self.gateway_client, self.target_id, self.name)
   1325 for temp_arg in temp_args:
   1326     if hasattr(temp_arg, "_detach"):

File ~\anaconda3\Lib\site-packages\pyspark\errors\exceptions\captured.py:179, in␣
 ↪capture_sql_exception.<locals>.deco(*a, **kw)
    177 def deco(*a: Any, **kw: Any) -> Any:
    178     try:
--> 179         return f(*a, **kw)
    180     except Py4JJavaError as e:
    181         converted = convert_exception(e.java_exception)

File ~\anaconda3\Lib\site-packages\py4j\protocol.py:326, in␣
 ↪get_return_value(answer, gateway_client, target_id, name)
    324 value = OUTPUT_CONVERTER[type](answer[2:], gateway_client)
    325 if answer[1] == REFERENCE_TYPE:
--> 326     raise Py4JJavaError(
    327         "An error occurred while calling {0}{1}{2}.\n".
    328         format(target_id, ".", name), value)
    329 else:
    330     raise Py4JError(
    331         "An error occurred while calling {0}{1}{2}. Trace:\n{3}\n".
    332         format(target_id, ".", name, value))

Py4JJavaError: An error occurred while calling o291.fit.
```

```
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 9 in␣
↪stage 13.0 failed 1 times, most recent failure: Lost task 9.0 in stage 13.0␣
↪(TID 247) (ATL_G2G22.cgocable.net executor driver): java.lang.RuntimeException ␣
↪Labels MUST be in [0, 100), but got 375.0
        at org.apache.spark.sql.catalyst.expressions.
↪GeneratedClass$GeneratedIteratorForCodegenStage1.
↪hashAgg_doAggregate_max_0$(Unknown Source)
        at org.apache.spark.sql.catalyst.expressions.
↪GeneratedClass$GeneratedIteratorForCodegenStage1.hashAgg_doConsume_0$(Unknown␣
↪Source)
        at org.apache.spark.sql.catalyst.expressions.
↪GeneratedClass$GeneratedIteratorForCodegenStage1.
↪hashAgg_doAggregateWithoutKey_0$(Unknown Source)
        at org.apache.spark.sql.catalyst.expressions.
↪GeneratedClass$GeneratedIteratorForCodegenStage1.processNext(Unknown Source)
        at org.apache.spark.sql.execution.BufferedRowIterator.
↪hasNext(BufferedRowIterator.java:43)
        at org.apache.spark.sql.execution.
↪WholeStageCodegenEvaluatorFactory$WholeStageCodegenPartitionEvaluator$$anon$1.
↪hasNext(WholeStageCodegenEvaluatorFactory.scala:43)
        at scala.collection.Iterator$$anon$10.hasNext(Iterator.scala:460)
        at org.apache.spark.shuffle.sort.BypassMergeSortShuffleWriter.
↪write(BypassMergeSortShuffleWriter.java:140)
        at org.apache.spark.shuffle.ShuffleWriteProcessor.
↪write(ShuffleWriteProcessor.scala:59)
        at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:
↪104)
        at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:
↪54)
        at org.apache.spark.TaskContext.runTaskWithListeners(TaskContext.scala:166)
        at org.apache.spark.scheduler.Task.run(Task.scala:141)
        at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$4(Executor
↪scala:620)
        at org.apache.spark.util.SparkErrorUtils.
↪tryWithSafeFinally(SparkErrorUtils.scala:64)
        at org.apache.spark.util.SparkErrorUtils.
↪tryWithSafeFinally$(SparkErrorUtils.scala:61)
        at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:94)
        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:623)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
        at java.lang.Thread.run(Unknown Source)

Driver stacktrace:
        at org.apache.spark.scheduler.DAGScheduler.
↪failJobAndIndependentStages(DAGScheduler.scala:2856)
        at org.apache.spark.scheduler.DAGScheduler.
↪$anonfun$abortStage$2(DAGScheduler.scala:2792)
```

```
        at org.apache.spark.scheduler.DAGScheduler.
↪$anonfun$abortStage$2$adapted(DAGScheduler.scala:2791)
        at scala.collection.mutable.ResizableArray.foreach(ResizableArray.scala:62)
        at scala.collection.mutable.ResizableArray.foreach$(ResizableArray.scala
↪55)
        at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffer.scala:49)
        at org.apache.spark.scheduler.DAGScheduler.abortStage(DAGScheduler.scala
↪2791)
        at org.apache.spark.scheduler.DAGScheduler.
↪$anonfun$handleTaskSetFailed$1(DAGScheduler.scala:1247)
        at org.apache.spark.scheduler.DAGScheduler.
↪$anonfun$handleTaskSetFailed$1$adapted(DAGScheduler.scala:1247)
        at scala.Option.foreach(Option.scala:407)
        at org.apache.spark.scheduler.DAGScheduler.
↪handleTaskSetFailed(DAGScheduler.scala:1247)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.
↪doOnReceive(DAGScheduler.scala:3060)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.
↪onReceive(DAGScheduler.scala:2994)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.
↪onReceive(DAGScheduler.scala:2983)
        at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:49)
Caused by: java.lang.RuntimeException: Labels MUST be in [0, 100), but got 375.0
        at org.apache.spark.sql.catalyst.expressions.
↪GeneratedClass$GeneratedIteratorForCodegenStage1.
↪hashAgg_doAggregate_max_0$(Unknown Source)
        at org.apache.spark.sql.catalyst.expressions.
↪GeneratedClass$GeneratedIteratorForCodegenStage1.hashAgg_doConsume_0$(Unknown␣
↪Source)
        at org.apache.spark.sql.catalyst.expressions.
↪GeneratedClass$GeneratedIteratorForCodegenStage1.
↪hashAgg_doAggregateWithoutKey_0$(Unknown Source)
        at org.apache.spark.sql.catalyst.expressions.
↪GeneratedClass$GeneratedIteratorForCodegenStage1.processNext(Unknown Source)
        at org.apache.spark.sql.execution.BufferedRowIterator.
↪hasNext(BufferedRowIterator.java:43)
        at org.apache.spark.sql.execution.
↪WholeStageCodegenEvaluatorFactory$WholeStageCodegenPartitionEvaluator$$anon$1.
↪hasNext(WholeStageCodegenEvaluatorFactory.scala:43)
        at scala.collection.Iterator$$anon$10.hasNext(Iterator.scala:460)
        at org.apache.spark.shuffle.sort.BypassMergeSortShuffleWriter.
↪write(BypassMergeSortShuffleWriter.java:140)
        at org.apache.spark.shuffle.ShuffleWriteProcessor.
↪write(ShuffleWriteProcessor.scala:59)
        at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:
↪104)
        at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:
↪54)
        at org.apache.spark.TaskContext.runTaskWithListeners(TaskContext.scala:166)
```

```
        at org.apache.spark.scheduler.Task.run(Task.scala:141)
        at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$4(Executor
↪scala:620)
        at org.apache.spark.util.SparkErrorUtils.
↪tryWithSafeFinally(SparkErrorUtils.scala:64)
        at org.apache.spark.util.SparkErrorUtils.
↪tryWithSafeFinally$(SparkErrorUtils.scala:61)
        at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:94)
        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:623)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
        at java.lang.Thread.run(Unknown Source)
```

Discussion: The error message here tells that a java plug in is not correctly installed. For me there are too many of them since I was not able to install Java VM (or something similar to support Python/Spark).

Since there is no syntax errors and applicaiton of ML algorithm as well as precedures, like, defining pipleline, and hyperparameter tuning is straightforward I assume upon correctly installing Java for Python the code above will produce desired result.

[15]:
```
## Using the Apache Spark ML pipeline, build a model to predict the price of a␣
↪diamond based on the available features.

# import/load necessary libraries/module
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when
from pyspark.ml.feature import VectorAssembler, StringIndexer, StandardScaler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
```

[16]:
```
# Initialize Spark Session -> Create a spark object
spark = SparkSession.builder.config("spark.driver.host", "localhost").
↪appName("Diamond Price Prediction").getOrCreate()

# Read Diamond.csv CSV File
df = spark.read.csv("C:/Users/azhar/OneDrive/Documents/WatSpeed/Big Data/Home␣
↪Work/assg4 data/diamonds.csv", header=True, inferSchema=True)

# checkout the structure of the newly read dataframe
df.printSchema()
df.count() # count number of rows after dropping missing values. (yields 53940)
```

```
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)
```

[16]:
```
53940
```

[17]:
```
# Data Cleaning: Imputing Missing Values
# For handling missing values we decided to removes all the missing values.

df = df.dropna()
df.count() # count number of rows after dropping missing values. (yields 53940)
```

[17]:
```
53940
```

Discussion: Before dropping the missing values the row-count was 53940. After applying df.dropna() function the count did not change. So the dataframe did not contain any missing value. Our dataframe is clean.

[18]:
```
# Convert Categorical Columns to Numerical
categorical_cols = ["cut", "color", "clarity"]
indexers = [StringIndexer(inputCol=col, outputCol=col + "_index").fit(df) for␣
 ↪col in categorical_cols]
```

[19]:
```
# Feature Selection
feature_cols = ["carat", "depth", "table", "x", "y", "z"] + [col + "_index" for␣
 ↪col in categorical_cols]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
```

[20]:
```
# Define Model
lr = LinearRegression(featuresCol="scaledFeatures", labelCol="price")
```

[21]:
```
# Create Pipeline
pipeline = Pipeline(stages=indexers + [assembler, scaler, lr])
```

[22]:
```
# Split Data
train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)
```

[23]:
```
# Train Model
model = pipeline.fit(train_data)
```

[24]:
```
# Predictions
predictions = model.transform(test_data)
predictions.select("price", "prediction").show(10)
```

```
+-----+------------------+
|price|        prediction|
+-----+------------------+
|  327|  81.91876910994142|
|  336|-446.44785896735993|
|  337|-1055.4915830042555|
|  344| -91.31692283015764|
|  351| -848.6998424215071|
|  353| -660.7043373959259|
|  357| 26.686460283548513|
|  402|-3.7498178275582177|
|  403|  25.45623265321592|
|  403|  243.4249214290012|
+-----+------------------+
only showing top 10 rows
```

[25]:
```
# Evaluate Model
training_summary = model.stages[-1].summary
print("RMSE:", training_summary.rootMeanSquaredError)
print("R2:", training_summary.r2)
```

```
RMSE: 1434.8583240283235
R2: 0.8708449427836911
```

To interpret RMSE - we got the value of RMSE as 1,434.85. This means that on average, the predicted diamond prices are off by $1,434.85 compared to the actual prices.

To interpret RMSE - we got the value of RMSE as 1,434.85. This means that on average, the predicted diamond prices are off by $1,434.85 compared to the actual prices.

A lower RMSE would mean better predictions, while a higher RMSE suggests room for improvement. In our case If diamond prices range from $10,000 to $50,000, an error of $1,434 might be acceptable (that is about 3% deviation). On the other hand, if prices range from $1,000 to $5,000, this error is quite large (about 30% deviation), meaning the model may need improvement.

For interpreting R^2, we received the R^2 value as 0.87. It means that 87.08% of the changes in the target value can be predicted based on the other factors in the model.