Classification of car's acceptability

```
!pip -q install pyspark
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.9/316.9 MB 4.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
  Building wheel for pyspark (setup.py) ... done
```

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder.master("local").appName('mini_projecr').getOrCreate()
```

```
df=spark.read.csv("/content/car.csv",header=True,inferSchema=True)
df.show(10,truncate=False)
```

```
+------------+-----------------+-----------+---------------+---------------+------+-----------------+
|Buying_Price|Maintenance_Price|No_of_Doors|Person_Capacity|Size_of_Luggage|Safety|Car_Acceptability|
+------------+-----------------+-----------+---------------+---------------+------+-----------------+
|vhigh       |vhigh            |2          |2              |small          |low   |unacc            |
|vhigh       |vhigh            |2          |2              |small          |med   |unacc            |
|vhigh       |vhigh            |2          |2              |small          |high  |unacc            |
|vhigh       |vhigh            |2          |2              |med            |low   |unacc            |
|vhigh       |vhigh            |2          |2              |med            |med   |unacc            |
|vhigh       |vhigh            |2          |2              |med            |high  |unacc            |
|vhigh       |vhigh            |2          |2              |big            |low   |unacc            |
|vhigh       |vhigh            |2          |2              |big            |med   |unacc            |
|vhigh       |vhigh            |2          |2              |big            |high  |unacc            |
|vhigh       |vhigh            |2          |4              |small          |low   |unacc            |
+------------+-----------------+-----------+---------------+---------------+------+-----------------+
only showing top 10 rows
```

```
for i in df.columns:
  print(i)
```

```
Buying_Price
Maintenance_Price
No_of_Doors
Person_Capacity
Size_of_Luggage
Safety
Car_Acceptability
```

```
(df.count(), len(df.columns))
```

```
(13824, 7)
```

```
df.printSchema()
```

```
root
 |-- Buying_Price: string (nullable = true)
 |-- Maintenance_Price: string (nullable = true)
 |-- No_of_Doors: string (nullable = true)
 |-- Person_Capacity: string (nullable = true)
 |-- Size_of_Luggage: string (nullable = true)
 |-- Safety: string (nullable = true)
 |-- Car_Acceptability: string (nullable = true)
```

```
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.ml.feature import *
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

```
col = [StructField('Buying_Price',StringType(),True),\
       StructField('Maintenance_Price',StringType(),True),\
       StructField('Doors',IntegerType(),True),\
       StructField('Persons',IntegerType(),True),\
       StructField('Luggages',StringType(),True),\
       StructField('Safety',StringType(),True),\
       StructField('Acceptability',StringType(),True)]
```

```
schema = StructType(col)
```

```
df=spark.read.csv("/content/car.csv",header=True,inferSchema=True,schema=schema)
df.show(5,truncate=False)
```

```
+------------+-----------------+-----+-------+--------+------+-------------+
|Buying_Price|Maintenance_Price|Doors|Persons|Luggages|Safety|Acceptability|
+------------+-----------------+-----+-------+--------+------+-------------+
|vhigh       |vhigh            |2    |2      |small   |low   |unacc        |
|vhigh       |vhigh            |2    |2      |small   |med   |unacc        |
|vhigh       |vhigh            |2    |2      |small   |high  |unacc        |
|vhigh       |vhigh            |2    |2      |med     |low   |unacc        |
|vhigh       |vhigh            |2    |2      |med     |med   |unacc        |
+------------+-----------------+-----+-------+--------+------+-------------+
only showing top 5 rows
```

```
df.printSchema()
```

```
root
 |-- Buying_Price: string (nullable = true)
 |-- Maintenance_Price: string (nullable = true)
 |-- Doors: integer (nullable = true)
 |-- Persons: integer (nullable = true)
 |-- Luggages: string (nullable = true)
 |-- Safety: string (nullable = true)
 |-- Acceptability: string (nullable = true)
```

```
null_value_list = list()
for col_ in df.columns:
    print(df[col_].isNull())
```

```
Column<'(Buying_Price IS NULL)'>
Column<'(Maintenance_Price IS NULL)'>
Column<'(Doors IS NULL)'>
Column<'(Persons IS NULL)'>
Column<'(Luggages IS NULL)'>
Column<'(Safety IS NULL)'>
Column<'(Acceptability IS NULL)'>
```

```
df=df.dropna()
df.show(5,truncate=False)
```

```
+------------+-----------------+-----+-------+--------+------+-------------+
|Buying_Price|Maintenance_Price|Doors|Persons|Luggages|Safety|Acceptability|
+------------+-----------------+-----+-------+--------+------+-------------+
|vhigh       |vhigh            |2    |2      |small   |low   |unacc        |
|vhigh       |vhigh            |2    |2      |small   |med   |unacc        |
|vhigh       |vhigh            |2    |2      |small   |high  |unacc        |
|vhigh       |vhigh            |2    |2      |med     |low   |unacc        |
|vhigh       |vhigh            |2    |2      |med     |med   |unacc        |
+------------+-----------------+-----+-------+--------+------+-------------+
only showing top 5 rows
```

```
(df.count(), len(df.columns))
```

```
(6912, 7)
```

```
df.groupBy(df.Maintenance_Price).count().show()
```

```
+-----------------+-----+
|Maintenance_Price|count|
+-----------------+-----+
|              low| 1728|
|            vhigh| 1728|
|              med| 1728|
|             high| 1728|
+-----------------+-----+
```

```
for cols in df.columns[:7]:
  indexer = StringIndexer(inputCol=cols, outputCol="temp")
  df = indexer.fit(df).transform(df)
  df=df.drop(cols).withColumnRenamed("temp",cols)
df.show(5)
```

```
+------------+-----------------+-----+-------+--------+------+-------------+
|Buying_Price|Maintenance_Price|Doors|Persons|Luggages|Safety|Acceptability|
+------------+-----------------+-----+-------+--------+------+-------------+
|         3.0|              3.0|  0.0|    0.0|     2.0|   1.0|          0.0|
|         3.0|             3.0|  0.0|    0.0|     2.0|   2.0|          0.0|
|         3.0|             3.0|  0.0|    0.0|     2.0|   0.0|          0.0|
|         3.0|             3.0|  0.0|    0.0|     1.0|   1.0|          0.0|
|         3.0|             3.0|  0.0|    0.0|     1.0|   2.0|          0.0|
+------------+----------------+-----+-------+--------+------+-------------+
only showing top 5 rows
```

```
df.withColumn("Acceptability",df.Acceptability.cast("Integer")).show(5)
```

```
+------------+----------------+-----+-------+--------+------+-------------+
|Buying_Price|Maintenance_Price|Doors|Persons|Luggages|Safety|Acceptability|
+------------+----------------+-----+-------+--------+------+-------------+
|         3.0|             3.0|  0.0|    0.0|     2.0|   1.0|            0|
|         3.0|             3.0|  0.0|    0.0|     2.0|   2.0|            0|
|         3.0|             3.0|  0.0|    0.0|     2.0|   0.0|            0|
|         3.0|             3.0|  0.0|    0.0|     1.0|   1.0|            0|
|         3.0|             3.0|  0.0|    0.0|     1.0|   2.0|            0|
+------------+----------------+-----+-------+--------+------+-------------+
only showing top 5 rows
```

```
inputColumns = ['Buying_Price', 'Maintenance_Price', 'Doors', 'Persons', 'Luggages', 'Safety']
outputColumn = "features"
```

```
for col_name in inputColumns:
    df = df.withColumn(col_name, df[col_name].cast(IntegerType()))
```

```
vector_assembler = VectorAssembler(inputCols=inputColumns, outputCol=outputColumn)
```

```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml import Pipeline
```

```
dt_model = DecisionTreeClassifier(labelCol="Acceptability", featuresCol=outputColumn)
```

```
stages = [vector_assembler, dt_model]
```

```
pipeline = Pipeline(stages=stages)
```

```
(train_df2, test_df2) = df.randomSplit([0.8, 0.2], seed=11)
```

```
final_pipeline = pipeline.fit(train_df2)
```

```
test_predictions_from_pipeline = final_pipeline.transform(test_df2)
```

```
test_predictions_from_pipeline.select("Acceptability", "prediction").show(5)
```

```
+-------------+----------+
|Acceptability|prediction|
+-------------+----------+
|          0.0|       0.0|
|          0.0|       0.0|
|          0.0|       0.0|
|          0.0|       0.0|
|          0.0|       0.0|
+-------------+----------+
only showing top 5 rows
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="Acceptability", predictionCol="prediction", metricName="accuracy")
```

```
accuracy = evaluator.evaluate(test_predictions_from_pipeline)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.895741556534508
```

```
from sklearn.metrics import classification_report
```

```
test_predictions_pd = test_predictions_from_pipeline.select("Acceptability", "prediction").toPandas()
```

```
true_labels = test_predictions_pd["Acceptability"].tolist()
predicted_labels = test_predictions_pd["prediction"].tolist()


report = classification_report(true_labels, predicted_labels)
print(report)
```

```
                   precision    recall  f1-score   support

             0.0       0.93      0.99      0.96      1057
             1.0       0.92      0.58      0.71       246
             2.0       0.00      0.00      0.00        31
             3.0       0.34      1.00      0.51        28

        accuracy                           0.90      1362
       macro avg       0.55      0.64      0.55      1362
    weighted avg       0.90      0.90      0.89      1362

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are
  _warn_prf(average, modifier, msg_start, len(result))
```

HyperParameter Tuning

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.feature import OneHotEncoder


paramGrid = ParamGridBuilder() \
    .addGrid(dt_model.maxDepth, [3, 5, 7]) \
    .addGrid(dt_model.minInstancesPerNode, [1, 3, 5]) \
    .build()


crossval = CrossValidator(estimator=pipeline, estimatorParamMaps=paramGrid,
                      evaluator=MulticlassClassificationEvaluator(
                      labelCol='Acceptability', predictionCol='prediction', metricName='accuracy'),
                      numFolds=5)

cvModel = crossval.fit(train_df2)


best_model = cvModel.bestModel
predictions = best_model.transform(test_df2)


evaluator = MulticlassClassificationEvaluator(labelCol="Acceptability", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)


print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9419970631424376
```

RandomForest

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="Acceptability", featuresCol=outputColumn, numTrees=100)


pipeline2 = Pipeline(stages=[vector_assembler,rf])


rf_model = pipeline.fit(train_df2)


rf_predictions = rf_model.transform(test_df2)


evaluator = MulticlassClassificationEvaluator(labelCol="Acceptability", predictionCol="prediction", metricName="accuracy")
accuracy2 = evaluator.evaluate(rf_predictions)


print(f"Accuracy: {accuracy2}")
```

```
Accuracy: 0.895741556534508
```

```
paramGrid2 = ParamGridBuilder() \
    .addGrid(rf.numTrees, [5, 7, 9]) \
    .addGrid(rf.maxDepth, [3, 5, 7]) \
    .addGrid(rf.featureSubsetStrategy, ["auto", "sqrt", "log2"]) \
    .build()


crossval2 = CrossValidator(estimator=pipeline2, estimatorParamMaps=paramGrid2,
                      evaluator=MulticlassClassificationEvaluator(
                      labelCol='Acceptability', predictionCol='prediction', metricName='accuracy'),
                      numFolds=5)

cvModel2 = crossval2.fit(train_df2)


best_model2 = cvModel2.bestModel
predictions2 = best_model2.transform(test_df2)


evaluator = MulticlassClassificationEvaluator(labelCol="Acceptability", predictionCol="prediction", metricName="accuracy")
accuracy_rf = evaluator.evaluate(predictions)


print(f"Accuracy: {accuracy_rf}")
```

```
     Accuracy: 0.9419970631424376
```