Rule Based Engineen -1

Here's a full Python project structure to implement the enhanced PySpark code, handle runtime configuration, and include unit tests.

Project Structure

```
1   feature_derivation/
2   ├── config/
3   │   └── example_config.json  # Sample configuration file
4   ├── src/
5   │   ├── __init__.py
6   │   ├── feature_derivation_engine.py  # Main logic for feature derivation
7   │   └── main.py  # Entry point for spark-submit
8   ├── tests/
9   │   ├── __init__.py
10  │   └── test_feature_derivation_engine.py  # Unit test cases
11  ├── requirements.txt  # Dependencies
12  └── README.md  # Project documentation
13
```

`example_config.json` **(Configuration File)**

```
1   {
2     "features": [
3       {
4         "output_column": "category",
5         "type": "complex",
6         "rule": {
7           "conditions": [
8             {
9               "condition": "age < 18",
10              "sub_conditions": [
11                {"condition": "score < 10", "value": "child with score less than 10"},
12                {"condition": "score >= 10 and score < 20", "value": "child with score between 10 and 20"},
13                {"condition": "score >= 20", "value": "child with score more than 20"}
14              ]
15            },
16            {
17              "condition": "age >= 18 and age < 60",
18              "value": "adult"
19            },
20            {
21              "condition": "age >= 60",
22              "value": "senior"
23            }
24          ],
25          "default": "unknown"
26        }
27      },
28      {
29        "output_column": "score_bucket",
30        "type": "transformation",
```

```
31        "rule": {
32          "expression": "CASE WHEN score < 10 THEN 'low' WHEN score BETWEEN 10 AND 20 THEN 'medium' ELSE 'high'
   END"
33        }
34      },
35      {
36        "output_column": "age_squared",
37        "type": "arithmetic",
38        "rule": {
39          "expression": "age * age"
40        }
41      }
42    ]
43  }
44
```

`feature_derivation_engine.py` **(Engine Logic)**

```python
1  from pyspark.sql import functions as F
2
3  class FeatureDerivationEngine:
4      def __init__(self, config):
5          self.config = config
6
7      def apply_complex_condition(self, df, conditions, default_value):
8          exprs = None
9          for condition in conditions:
10             parent_condition = F.expr(condition["condition"])
11             if "sub_conditions" in condition:
12                 sub_exprs = F.when(F.expr(condition["sub_conditions"][0]["condition"]),
   condition["sub_conditions"][0]["value"])
13                 for sub in condition["sub_conditions"][1:]:
14                     sub_exprs = sub_exprs.when(F.expr(sub["condition"]), sub["value"])
15                 sub_exprs = sub_exprs.otherwise(default_value)
16                 parent_expr = F.when(parent_condition, sub_exprs)
17             else:
18                 parent_expr = F.when(parent_condition, condition["value"])
19             exprs = parent_expr if exprs is None else exprs.otherwise(parent_expr)
20         exprs = exprs.otherwise(default_value)
21         return exprs
22
23     def apply_transformation(self, df, rule, output_col):
24         expression = rule["expression"]
25         return df.withColumn(output_col, F.expr(expression))
26
27     def apply_rules(self, df):
28         for feature in self.config["features"]:
29             output_col = feature["output_column"]
30             rule_type = feature["type"]
31             rule = feature["rule"]
32
33             if rule_type == "complex":
34                 conditions = rule["conditions"]
35                 default_value = rule.get("default", "unknown")
36                 complex_expr = self.apply_complex_condition(df, conditions, default_value)
37                 df = df.withColumn(output_col, complex_expr)
```

```
38
39            elif rule_type in ["transformation", "arithmetic", "string"]:
40                df = self.apply_transformation(df, rule, output_col)
41
42        return df
43
```

main.py (Entry Point)

```
1  import json
2  import sys
3  from pyspark.sql import SparkSession
4  from src.feature_derivation_engine import FeatureDerivationEngine
5
6  def main(config_path):
7      # Initialize Spark
8      spark = SparkSession.builder.appName("FeatureDerivation").getOrCreate()
9
10     # Load Configuration
11     with open(config_path, "r") as f:
12         config = json.load(f)
13
14     # Sample Data
15     data = [
16         (1, 15, 5), (2, 15, 15), (3, 15, 25), (4, 30, 50), (5, 70, 50)
17     ]
18     columns = ["id", "age", "score"]
19     df = spark.createDataFrame(data, columns)
20
21     # Apply Feature Derivation
22     engine = FeatureDerivationEngine(config)
23     result_df = engine.apply_rules(df)
24
25     # Show Results
26     result_df.show()
27
28  if __name__ == "__main__":
29      if len(sys.argv) != 2:
30          print("Usage: spark-submit main.py <config_file>")
31          sys.exit(-1)
32
33      config_path = sys.argv[1]
34      main(config_path)
35
```

test_feature_derivation_engine.py (Unit Tests)

```
1  import unittest
2  from pyspark.sql import SparkSession
3  from src.feature_derivation_engine import FeatureDerivationEngine
4
5  class FeatureDerivationTest(unittest.TestCase):
6      @classmethod
7      def setUpClass(cls):
```

```
 8            cls.spark = SparkSession.builder.appName("UnitTest").getOrCreate()
 9
10       @classmethod
11       def tearDownClass(cls):
12            cls.spark.stop()
13
14       def test_apply_rules(self):
15            # Mock Configuration
16            config = {
17                "features": [
18                    {
19                        "output_column": "category",
20                        "type": "complex",
21                        "rule": {
22                            "conditions": [
23                                {"condition": "age < 18", "value": "minor"},
24                                {"condition": "age >= 18", "value": "adult"}
25                            ],
26                            "default": "unknown"
27                        }
28                    }
29                ]
30            }
31
32            # Sample Data
33            data = [(1, 15), (2, 25), (3, None)]
34            columns = ["id", "age"]
35            df = self.spark.createDataFrame(data, columns)
36
37            # Apply Feature Derivation
38            engine = FeatureDerivationEngine(config)
39            result_df = engine.apply_rules(df)
40
41            # Expected Results
42            expected_data = [(1, 15, "minor"), (2, 25, "adult"), (3, None, "unknown")]
43            expected_df = self.spark.createDataFrame(expected_data, columns + ["category"])
44
45            # Assert Equality
46            self.assertEqual(sorted(result_df.collect()), sorted(expected_df.collect()))
47
48  if __name__ == "__main__":
49       unittest.main()
50
```

`requirements.txt`

```
1  pyspark==3.4.1
2  unittest2==1.1.0
3
```

Running the Project

1. **Run the PySpark Job**:

```
1  spark-submit src/main.py config/example_config.json
```

```
2
```

2. **Run Unit Tests**:

```
python -m unittest discover -s tests
```

This setup provides a production-ready project that supports runtime configuration and unit testing, ensuring maintainability and scalability.