Lab4: spark MLlib

Task 1-2: Load Data & Show Schema

- **Relation to Course**: Basic Spark operations to ingest data (prerequisite for MLlib workflows).
- **Code**:

```
df = spark.read.csv("tp4_data/*", header=True, inferSchema=True)
df.printSchema()
```

Task 3: Fill Missing Values

- **Relation**: Data cleaning (similar to preprocessing in `RFormula`).
- **Code**:

```
df = df.fillna(0)
```

Task 4: Add "day_of_week" Column

- **Relation**: Feature engineering using Spark SQL functions.
- **Code**:

```
from pyspark.sql.functions import date_format
df = df.withColumn("day_of_week", date_format("InvoiceDate", "EEEE"))
```

Task 5: Temporal Split

- **Relation**: Data splitting for training/testing (non-random split, unlike the course example).
- **Code**:

```
train = df.filter(df.InvoiceDate < "2010-12-13")
test = df.filter(df.InvoiceDate >= "2010-12-13")
```

Task 6-7: StringIndexer & Ordinal Issue

- **Relation**: Encoding categorical variables manually (vs. `RFormula` automation).
- **Code**:

  ```
  from pyspark.ml.feature import StringIndexer
  indexer = StringIndexer(inputCol="day_of_week", outputCol="day_of_week_encoded")
  ```

  - **Ordinal Issue**: `StringIndexer` assigns arbitrary numerical labels (e.g., Monday=0, Tuesday=1). To avoid implying order, use `OneHotEncoder`:

    ```
    from pyspark.ml.feature import OneHotEncoder
    encoder = OneHotEncoder(inputCol="day_of_week_encoded", outputCol="day_of_week_vec")
    ```

Task 8: VectorAssembler

- **Relation**: Combining features into a vector (like `RFormula`'s output).
- **Code**:

  ```
  from pyspark.ml.feature import VectorAssembler
  assembler = VectorAssembler(
    inputCols=["UnitPrice", "Quantity", "day_of_week_encoded"],
    outputCol="features"
  )
  ```

Task 9: Pipeline

- **Relation**: Chains stages (transformers + estimator) like the course example.
- **Code**:

  ```
  from pyspark.ml import Pipeline
  pipeline = Pipeline(stages=[indexer, assembler, kmeans])
  ```

  - **The purpose:**

The pipeline consists of **multiple stages** (e.g., `StringIndexer`, `OneHotEncoder`, `VectorAssembler`). When you **fit** the pipeline on the training set, it learns transformations (e.g., how to encode categorical variables). Then, you can **reuse** the same pipeline to transform new data (like the test set) **without redefining everything manually**.

Task 10: Unique Values in StringIndexer

- **Relation**: `StringIndexer` must `fit()` to learn categories (like `RFormula`'s `fit()` step).
- **Solution**: Call `fit()` on the training data to ensure all categories are seen:

```
indexer_model = indexer.fit(train)
```

Task 11-14: Transform & Train KMeans

- **Relation**: Train a model (estimator) using transformed data.
- **Code**:

```
model = pipeline.fit(train)  # Includes KMeans training
test_predictions = model.transform(test)
```

Task 15: Silhouette Coefficient

- **Relation**: Evaluator for clustering (like `BinaryClassificationEvaluator` for classification).
- **Code**:

```
from pyspark.ml.evaluation import ClusteringEvaluator
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(test_predictions)
print(f"Silhouette: {silhouette}")
```