**Final Project Report**

**Brewery Sales Forecasting**

Team D

Shaun Mendes

Bhushan Vinod Karande

**Problem Objectives:**

* **Sales Prediction**: Analyze the brewery dataset to identify key factors influencing beer sales and build predictive models using Linear Regression, Decision Tree, and Random Forest algorithms.​
* **Scaling Operations**: Examine the impact of scaling up (increasing dataset size) and scaling out (expanding cluster capacity) on model accuracy and performance.
* **Explore factors** that affect the sales the most.

**Sample Script:**

We will be using the brewery\_pyspark\_rf\_grid.py that also has code for grid search.

import argparse

from pyspark.sql import SparkSession

from pyspark.sql.types import FloatType

from pyspark.sql.functions import col, date\_format, to\_timestamp

from pyspark.ml.feature import VectorAssembler, StringIndexer

from pyspark.ml.regression import RandomForestRegressor

from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.ml import Pipeline

def main(input\_filepath):

*# Initialize spark session*

    spark = SparkSession.builder.appName("Beer analysis").getOrCreate()

*# Load dataset and infer schema*

    train = spark.read.csv(

        input\_filepath,

        header=True,

        inferSchema=True,

    )

*# Count of rows in dataset*

    train.count()

*# Drop duplicates and NAs*

    train = train.dropDuplicates()

    train = train.na.drop()

*# Cast the target variable to float*

    train = train.withColumn("Total\_Sales", col("Total\_Sales").cast(FloatType()))

*# As we cannot use the datatime value directly, we split it into Year, Month and Day*

    train = train.withColumn(

        "Brew\_Date", to\_timestamp(col("Brew\_Date"), "yyyy-MM-dd HH:mm:ss")

    )

    train = (

        train.withColumn("Month", date\_format(col("Brew\_Date"), "MM"))

        .withColumn("Day", date\_format(col("Brew\_Date"), "dd"))

        .withColumn("Year", date\_format(col("Brew\_Date"), "yyyy"))

    )

*# Convert categorical columns to numeric values*

    categorical\_columns = ["Beer\_Style", "SKU", "Location"]

    indexers = [

        StringIndexer(inputCol=c, outputCol="{0}\_indexed".format(c))

        for c in categorical\_columns

    ]

    numeric\_columns = [

        "Fermentation\_Time",

        "Temperature",

        "pH\_Level",

        "Gravity",

        "Alcohol\_Content",

        "Bitterness",

        "Color",

        "Volume\_Produced",

        "Quality\_Score",

        "Brewhouse\_Efficiency",

        "Loss\_During\_Brewing",

        "Loss\_During\_Fermentation",

        "Loss\_During\_Bottling\_Kegging",

    ]

    assembler\_inputs = [c + "\_indexed" for c in categorical\_columns] + numeric\_columns

*# Initialize VectorAssembler*

    assembler = VectorAssembler(inputCols=assembler\_inputs, outputCol="features")

*# We trained on partial data during in the notebook. Over here we split all the datasets 80:20 as we pass in 100% of the data*

*# Split the data into train and test*

    train\_data, test\_data = train.randomSplit([0.8, 0.2], seed=42)

*# Initialize Model*

    rf = RandomForestRegressor(featuresCol="features", labelCol="Total\_Sales")

    pipeline = Pipeline(stages=indexers + [assembler, rf])

*# Initialize params for grid search*

    paramGrid = (

        ParamGridBuilder()

        .addGrid(rf.numTrees, [10, 20, 50])

        .addGrid(rf.maxDepth, [5, 10, 20])

        .build()

    )

*# Initialize the CrossValidator along with metric*

    crossval = CrossValidator(

        estimator=pipeline,

        estimatorParamMaps=paramGrid,

        evaluator=RegressionEvaluator(

            labelCol="Total\_Sales", predictionCol="prediction", metricName="rmse"

        ),

        numFolds=3,

    )

*# Train the model on all possible param combinations*

    model = crossval.fit(train\_data)

*# Provide output predictions*

    predictions = model.transform(test\_data)

*# Initialize evaluator*

    evaluator = RegressionEvaluator(

        labelCol="Total\_Sales", predictionCol="prediction", metricName="rmse"

    )

*# Compute RMSE*

    rmse = evaluator.evaluate(predictions)

    print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

    evaluator = RegressionEvaluator(

        labelCol="Total\_Sales", predictionCol="prediction", metricName="r2"

    )

*# Compute R Square. Adjusted R square wasn't used as there is no direct implementation in pyspark*

    r2 = evaluator.evaluate(predictions)

    print("R Squared on test data = %g" % r2)

if \_\_name\_\_ == "\_\_main\_\_":

*# Parser is used to pass input file path through command line*

    parser = argparse.ArgumentParser(description="PySpark Job Arguments")

    parser.add\_argument("input\_path", type=str, help="Input file path")

    args = parser.parse\_args()

    main(args.input\_path)

**GCP:**

Sample Job:

**A screenshot of a computer

Description automatically generated**

Runs:

A screenshot of a computer

Description automatically generated

**Scaling Out:**

A graph of different numbers of workers

Description automatically generated

**Scaling Up:**

A graph with different colored lines

Description automatically generated