Retail Demand Forecasting and Inventory Planning using PySpark and MLlib

Sai Ranga Reddy Nukala[1], Akshith Reddy Jonnalagadda[2], Shaila Reddy Kankanala[3], Sadhvi Singh[4]
[1,2,3,4]Department of Applied Data Science, San Jose State University, San Jose, CA
[1]sairangareddy.nukala@sjsu.edu, [2]akshithreddy.jonnalagadda@sjsu.edu, [3]shailareddy.kankanala@sjsu.edu, [4]sadhvi.singh@sjsu.edu

***Abstract*—Retail demand forecasting remains a critical challenge for inventory planning, particularly during seasonal peaks and public holidays. This project addresses the problem using a real-world dataset containing over one million UK-based online retail transactions, enriched with public holiday data from 2009–2011. Our objective is to build a scalable, interpretable pipeline that forecasts monthly product-level demand and recommends stocking actions.**

We leverage PySpark for distributed processing and feature engineering, and employ MLlib to train Random Forest models for both regression and classification tasks. Feature engineering was guided by exploratory data analysis (EDA), revealing seasonal cycles and holiday-driven demand spikes. The final regression model achieved an RMSE of £373.99 and MAE of £195.06. The inventory classifier predicts stock actions (Increase, Maintain, Reduce) with 91.89% accuracy. Results and insights are presented via an interactive Streamlit dashboard, enabling data-driven inventory decisions.

This system demonstrates how big data technologies and interpretable ML methods can power actionable forecasting and inventory optimization in the retail domain.

***Index Terms*—Retail demand forecasting, PySpark, MLlib, Random Forest, Inventory classification, Time-series features, Holiday-aware modeling, Big data pipeline, Streamlit visualization, Product-level forecasting**

I. Introduction

Forecasting retail demand remains a fundamental challenge for businesses seeking to manage inventory, optimize logistics, and reduce waste. In particular, consumer behavior during holiday periods and seasonal shifts introduces volatility that traditional planning methods often fail to anticipate. Missed demand can lead to lost revenue, while overstocking incurs storage and markdown costs.

Recent advances in big data technologies and machine learning provide an opportunity to improve demand forecasting by leveraging historical transaction data, enriched with contextual features such as calendar events and product categories. Several studies and industry use cases suggest that combining time-based features with external signals—like holidays or promotions—improves predictive accuracy and downstream planning decisions.

This project explores a demand forecasting solution built on a large-scale online retail dataset. Using Apache Spark, we process over 1 million transactions to extract product-level monthly trends. We then apply Random Forest models for both regression and classification tasks. The project also integrates public holiday data to measure and visualize its effect on demand, a technique increasingly adopted in time-series forecasting literature (e.g., Prophet, ARIMAX).

We explored simpler models like logistic and linear regression but found they underperformed due to the nonlinear, seasonal nature of the data.

While forecasting models are well studied, their application in noisy, real-world retail datasets still faces issues like high-cardinality product data, irregular buying patterns, and underdocumented metadata. Our work seeks to address these gaps through engineered features and dashboard-driven interpretation of predictions.

II. Data Exploration and Processing

The dataset used in this project originates from the **UCI Online Retail II** repository. It includes approximately **1.06 million transactions** recorded between **December 2009 and December 2011** by a UK-based online retailer. Each transaction contains an invoice number, product description, quantity, unit price, timestamp, and country.

A. Raw Data Challenges

Initial exploration revealed several common data quality issues:

- **Cancellations and returns** (invoices prefixed with ”C”)
- **Missing `CustomerID`** values
- **Negative or zero `Quantity` and `Price`** entries

These anomalies were cleaned using PySpark filters before further processing. Transactions with invalid values were excluded to ensure integrity in trend analysis and modeling. The PySpark code snippet below illustrates this cleaning process.

```
import os
import logging
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date
    , sum as spark_sum, expr
from pyspark.sql.types import FloatType

LOG_DIR = "logs"
os.makedirs(LOG_DIR, exist_ok=True)

logging.basicConfig(
    filename=os.path.join(LOG_DIR, "spark_eda.
        log"),
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(
        message)s",
)
```

```
log = logging.getLogger(__name__)

log.info("Starting Spark EDA script")

spark = SparkSession.builder.appName("Retail
    Demand Forecasting - EDA").getOrCreate()
log.info("Spark session started")

try:
    df = (
        spark.read.option("header", "true")
        .option("inferSchema", "true")
        .csv("data/online_retail_raw.csv")
    )
    log.info(f"Loaded dataset with {df.count()
        } rows")
except Exception as e:
    log.error("Failed to load data", exc_info=
        True)
    raise e

log.info("Starting data cleaning")
df_clean = (
    df.filter(~col("Invoice").startswith("C"))
    .filter((col("Quantity") > 0) & (col("
        Price") > 0))
    .filter(col("Customer ID").isNotNull())
    .withColumn("TotalPrice", col("Quantity")
        * col("Price"))
    .withColumn("InvoiceDate", to_date("
        InvoiceDate"))
    .withColumn(
        "InvoiceMonth", expr("make_date(year(
            InvoiceDate), month(InvoiceDate),
            1)")
    )
)
log.info("Cleaning complete")

log.info("Selecting top 1000 products by
    revenue")
top_products = (
    df_clean.groupBy("Description")
    .agg(spark_sum("TotalPrice").alias("
        Revenue"))
    .orderBy(col("Revenue").desc())
    .limit(1000)
    .select("Description")
)

df_top = df_clean.join(top_products, on="
    Description", how="inner")

log.info("Aggregating monthly sales per
    product")
monthly_sales = (
    df_top.groupBy("Description", "
        InvoiceMonth")
    .agg(spark_sum("TotalPrice").cast(
        FloatType()).alias("TotalPrice"))
    .orderBy("Description", "InvoiceMonth")
)

output_path = "output/aggregated_sales_monthly
    "
try:
    monthly_sales.coalesce(1).write.option("
        header", "true").csv(
        output_path, mode="overwrite"
    )
    log.info(f"Saved output to {output_path}")
except Exception as e:
    log.error("Failed to write output",
        exc_info=True)
    raise e

log.info("Spark EDA script completed
    successfully")
```

Code Block 1. PySpark Data Cleaning and Aggregation

This code performs the following operations:

- **Loads the Data:** Reads the raw CSV data into a Spark DataFrame.
- **Cleans the Data:** Removes cancelled orders, ensures positive quantities and prices, and filters out records with missing customer IDs.
- **Calculates Total Price:** Computes the total price for each transaction.
- **Aggregates Monthly Sales:** Groups the data by product and month to calculate total monthly sales.
- **Outputs the Results:** Writes the aggregated data to a CSV file.

B. Aggregation and Product Filtering

Due to the large cardinality of the `Description` column (4,000+ unique products), we applied a **top-N filtering strategy**. Specifically, we selected the **filtered to the top 1000 products by total revenue to ensure robust pattern learning per SKU** using Spark's `groupBy + sum` functions. This allowed us to focus modeling efforts on SKUs with sufficient transaction volume for meaningful time-series learning.

We then aggregated the cleaned dataset into a **product–month level view**:

- **Group by:** `Description`, `InvoiceMonth`
- **Aggregate:** total `Quantity`, total `TotalPrice`

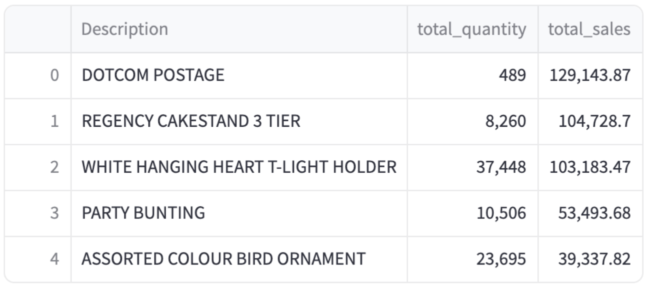
This monthly time-series format formed the foundation for all modeling and EDA tasks downstream.

C. Holiday Data Integration

To analyze seasonal and calendar-driven demand, we joined the retail data with a curated dataset of **UK public holidays** (2009–2011). Using Spark's date parsing and formatting functions, we matched transactions to their respective `holiday_month` and added binary indicators for holiday-aware feature engineering.

D. Exploratory Data Analysis

To capture the seasonal and calendar-driven demand patterns inherent in retail data, we integrated our transactional dataset with a curated collection of **UK public holidays** (2009–2011). This integration allowed us to identify periods of heightened consumer activity and align our forecasting models accordingly. Using Spark's date parsing and formatting functions, we matched transactions to their respective `holiday_month` and created binary indicators for holiday-aware feature engineering. This approach enabled us to capture critical demand spikes, long-term seasonality, and post-holiday effects, which are essential for accurate demand forecasting.



	Description	total_quantity	total_sales
0	DOTCOM POSTAGE	489	129,143.87
1	REGENCY CAKESTAND 3 TIER	8,260	104,728.7
2	WHITE HANGING HEART T-LIGHT HOLDER	37,448	103,183.47
3	PARTY BUNTING	10,506	53,493.68
4	ASSORTED COLOUR BIRD ORNAMENT	23,695	39,337.82

Fig. 1. Top 5 Most In-Demand Products During Holiday Months.

The table in Figure 1 captures the most in-demand products during public holiday months, emphasizing the importance of home decor and gift items. These products, like ”WHITE HANGING HEART T-LIGHT HOLDER”, ”REGENCY CAKESTAND 3 TIER” and ”PARTY BUNTING,” consistently perform well in peak retail periods and leading in total revenue, making them critical for inventory planning.

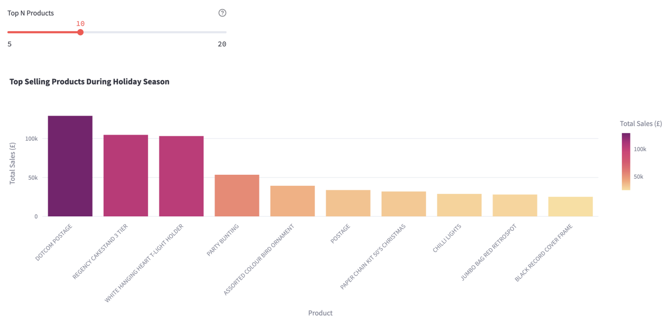


Fig. 2. Top Selling Products During Holiday Season.

Figure 2 presents the same data in a more visual format using a bar graph ranking, products by total sales while highlighting the dominance of decor items like ”WHITE HANGING HEART T-LIGHT HOLDER” and ”PARTY BUNTING” during festive periods. This view provides a clearer understanding of the scale of demand for each product, emphasizing the need for targeted inventory strategies.

The heatmap in Figure 3 provides a quarter-wise breakdown of average sales, revealing distinct seasonal trends. It highlights that certain products, like ”WHITE HANGING HEART T-LIGHT HOLDER” and other decorative items, experience pronounced demand spikes in Q4, aligning with major holidays.

This graph in Figure 4 illustrates a comparison between Christmas and New Year sales across multiple years, demonstrating that some products like ”ASSORTED COLOUR BIRD ORNAMENT” can maintain strong post-holiday de-

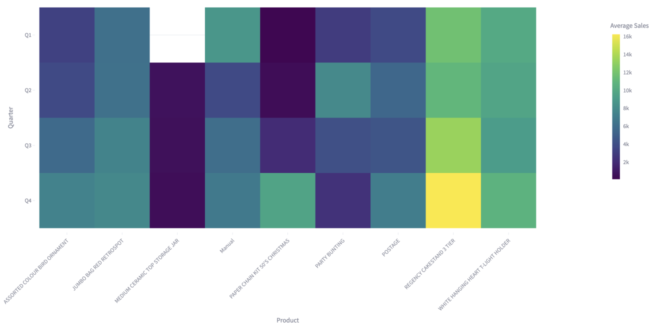


Fig. 3. Average Quarterly Sales by Product.

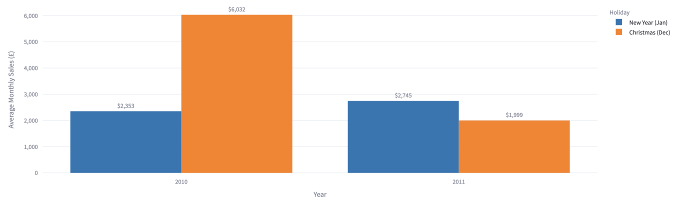


Fig. 4. Year-by-Year Holiday Sales Comparison.

mand. This insight is crucial for optimizing stock levels in the weeks following the peak retail season.

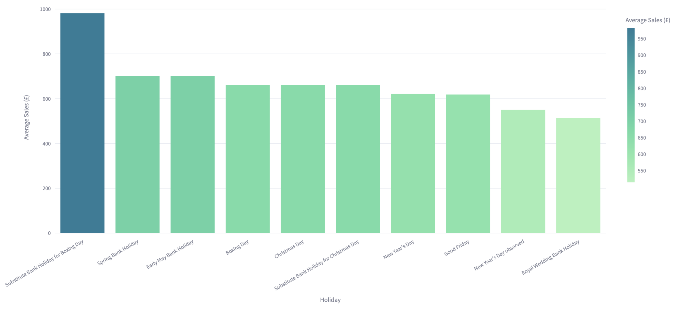


Fig. 5. Average Monthly Sales by Holiday.

Finally, Figure 5 was plotted to reveal which public holidays in the UK correspond to the highest average monthly sales. This revealed that substitute bank holidays often outperform Christmas. This finding underscores the importance of aligning promotional strategies with less obvious, but highly profitable, holiday periods.

Collectively, these visualizations informed our feature engineering approach, including the creation of key features like `is_holiday_month`, `prev_month_sales`, and `sales_delta`, which directly contributed to the accuracy and interpretability of our forecasting models.

The cleaned and transformed dataset, enriched with holiday context and monthly aggregations, enabled us to engineer meaningful features such as `prev_month_sales`, `rolling_3m_avg`, `sales_delta`, and `is_holiday_month`. These features, grounded in exploratory findings, form the input to our machine learning models described in the next section.

III. Proposed Method

The goal of this project is to create a predictive pipeline that can both **forecast monthly sales** and **recommend inventory actions** for each product. To achieve this, we built two distinct machine learning workflows using **PySpark MLlib**:

1) A regression model to forecast future sales in pounds.
2) A classification model to label stock decisions as **Increase**, **Maintain**, or **Reduce**.

A. Architecture Overview

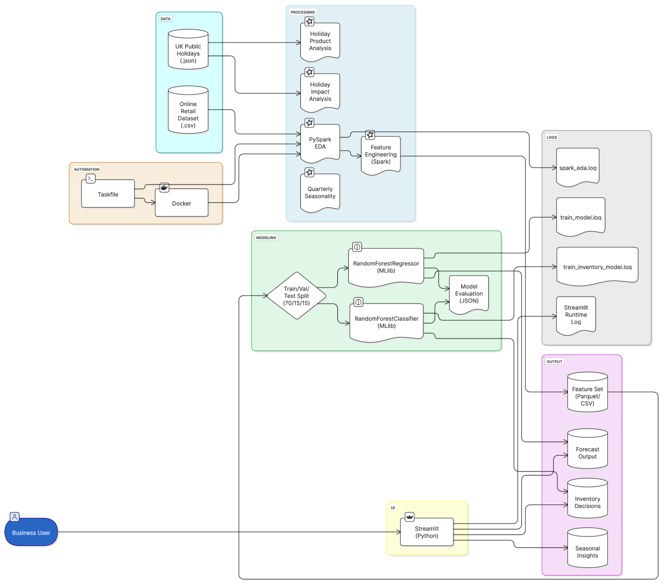


Fig. 6. System Architecture Overview

The overall architecture of our retail demand forecasting system is designed to be modular, reproducible, and scalable – combining **Spark** for big data processing, **MLlib** for modeling, and **Streamlit** for interactive visualization. Below is a detailed description of each component in the architecture.

1) User Interface (Streamlit Dashboard):

- Built with Python and **Streamlit**
- Provides business users access to:
  - Demand forecasts per product
  - Seasonal and holiday trend visualizations
  - Inventory action recommendations
- Connects directly to model outputs (`predictions`, `inventory`, `seasonality`)

2) Data Sources:

- **Retail Dataset (CSV)**: Contains over 1 million online transaction records from 2009–2011. Includes fields like `InvoiceDate`, `Quantity`, `Price`, and `Description`.
- **UK Public Holidays (JSON)**: A locally stored JSON file with structured holiday records. This was parsed and joined with the main dataset to generate holiday-aware features (`is_holiday_month`).

3) Automation & Execution Layer:

- **Docker**: Ensures consistent runtime for **Spark** jobs and dashboard execution.
- **Taskfile (CLI)**: Automates the orchestration of **Spark** jobs for EDA, feature generation, modeling, and dashboard launch.

4) Data Processing Layer (PySpark):

- **`spark_eda.py`**: Cleans raw data, filters returns, aggregates sales by product-month.
- **`spark_features.py`**: Engineers predictive features such as:
  - `prev_month_sales`
  - `rolling_3m_avg`
  - `sales_delta`
  - `is_holiday_month`
  - `Month`
  - `DescriptionIndex`
- **Holiday Analysis Scripts** (`analyze_holiday_*`): Generate exploratory datasets like top holiday SKUs and quarterly trends.

5) Modeling Layer (Spark MLlib):

- **Train/Validation/Test Split (70%/15%/15%)**:
  - Applied to both regression and classification pipelines

a) → Regression Model:

- **Model**: `RandomForestRegressor`
- **Tuning**: Grid search over `numTrees` ∈ {10, 20}, `maxDepth` ∈ {5, 10}
- **Best Config**: `numTrees = 20`, `maxDepth = 5`
- **Output**: Forecasted `TotalPrice` per product-month

b) → Classification Model:

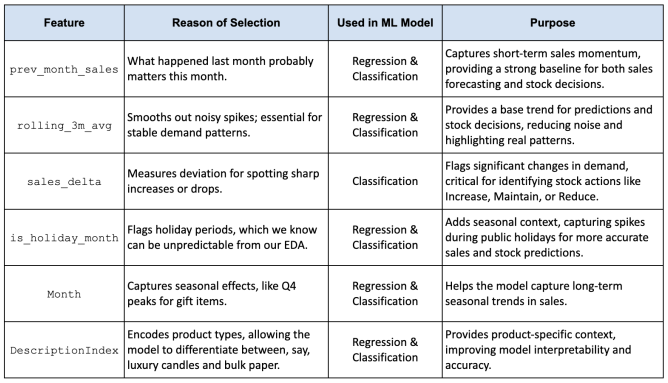
- **Model**: `RandomForestClassifier`
- **Labeling**:
  - **Increase**: sales $>$ 120% of 3-month avg
  - **Reduce**: sales $<$ 80%
  - **Maintain**: everything else
- **Decision**: Used `maxBins = 2048` to handle high-cardinality feature (`DescriptionIndex`)
- **Output**: `stock_decision` labels for each product-month

6) Outputs:

- **`features/`**: Final feature vectors used in modeling
- **`predictions/`**: Regression outputs
- **`predictions_inventory/`**: Classification outputs
- **`seasonality/`**: Results from holiday and quarterly analysis scripts

7) Logging:

- Each **Spark** job outputs logs (e.g., `spark_eda.log`, `train_model.log`)
- **Streamlit** logs runtime activity (`dashboard_log`)



Feature	Reason of Selection	Used in ML Model	Purpose
prev_month_sales	What happened last month probably matters this month.	Regression & Classification	Captures short-term sales momentum, providing a strong baseline for both sales forecasting and stock decisions.
rolling_3m_avg	Smooths out noisy spikes; essential for stable demand patterns.	Regression & Classification	Provides a base trend for predictions and stock decisions, reducing noise and highlighting real patterns.
sales_delta	Measures deviation for spotting sharp increases or drops.	Classification	Flags significant changes in demand, critical for identifying stock actions like Increase, Maintain, or Reduce.
is_holiday_month	Flags holiday periods, which we know can be unpredictable from our EDA.	Regression & Classification	Adds seasonal context, capturing spikes during public holidays for more accurate sales and stock predictions.
Month	Captures seasonal effects, like Q4 peaks for gift items.	Regression & Classification	Helps the model capture long-term seasonal trends in sales.
DescriptionIndex	Encodes product types, allowing the model to differentiate between, say, luxury candles and bulk paper.	Regression & Classification	Provides product-specific context, improving model interpretability and accuracy.

Fig. 7. Feature Engineering

B. Feature Engineering

Feature engineering is a critical step in building accurate Machine Learning models, especially in the context of retail demand forecasting. Our goal is to create meaningful features that capture the underlying patterns identified during exploratory data analysis (EDA). This section describes the key features engineered for both the **Random Forest Regressor** and **Random Forest Classifier** models used in this project.

The features listed in figure 7 below were carefully selected based on insights from EDA. Each feature captures a specific aspect of product demand, enhancing the predictive power of our models:

- **prev_month_sales** and **rolling_3m_avg** provide strong baselines for both models by capturing short-term trends and smoothing out noisy fluctuations.

- **sales_delta** is particularly critical for the RandomForest-Classifier as it captures sharp deviations from the average trend, making it ideal for stock decision classification (Increase, Maintain, Reduce).

- **is_holiday_month** and **Month** capture seasonal effects, which are crucial for accurate forecasting in the retail domain.

- **DescriptionIndex** encodes product-level context, improving model interpretability and helping the models differentiate between high-value and low-value items.

We set maxBins=2048 in MLlib to accommodate 996 distinct product categories encoded via DescriptionIndex.

Overall, these features allowed our **RandomForestRegressor** to be solid at capturing overall sales trends, while our **RandomForestClassifier** proved to be highly reliable for making critical stock decisions, effectively identifying when to Increase, Maintain, or Reduce inventory levels.

The most important stage in the pipeline was crafting **time-aware features** to capture product behavior and calendar context. The Python code shows the feature engineering processes.

```
import os
import logging
from pyspark.sql import SparkSession, Window
from pyspark.sql.functions import (
    col,
    lag,
    avg,
    month,
    year,
    date_format,
    to_date,
    concat_ws,
)

LOG_DIR = "logs"
os.makedirs(LOG_DIR, exist_ok=True)

logging.basicConfig(
    filename=os.path.join(LOG_DIR, "
        spark_features.log"),
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(
        message)s",
)
log = logging.getLogger(__name__)
log.info("Starting Spark feature engineering
    script")

spark = SparkSession.builder.appName(
    "Retail Forecasting - Feature Engineering"
).getOrCreate()
log.info("Spark session started")

input_path = "output/aggregated_sales_monthly"
df = spark.read.option("header", "true").csv(
    input_path, inferSchema=True)
log.info(f"Loaded aggregated data with {df.
    count()} rows")

df = df.withColumn("Year", year("InvoiceMonth"
    )).withColumn(
    "Month", month("InvoiceMonth")
)

window_spec = Window.partitionBy("Description"
    ).orderBy("InvoiceMonth")
df = df.withColumn("prev_month_sales", lag("
    TotalPrice", 1).over(window_spec))
df = df.withColumn(
    "rolling_3m_avg", avg("TotalPrice").over(
        window_spec.rowsBetween(-2, 0))
)
df = df.dropna(subset=["prev_month_sales", "
    rolling_3m_avg"])


# Load all columns from your CSV
holidays = spark.read.option("header", "true")
    .csv(
    "data/public_holidays_uk_2009_2011.csv",
        inferSchema=True
)

# Combine Year + Date columns (e.g., 2010 +
    Dec 25)
holidays = holidays.withColumn("DateString",
    concat_ws(" ", col("Year"), col("Date")))
holidays = holidays.withColumn("DateParsed",
    to_date(col("DateString"), "yyyy MMM d"))

# Extract yyyy-MM
holidays = holidays.withColumn(
    "holiday_month", date_format(col("
        DateParsed"), "yyyy-MM")
)

# Log preview
holidays.select("Date", "DateParsed", "
    holiday_month", "Name").show(5, truncate=
    False)

holiday_months = holidays.select("
    holiday_month").distinct()

df = df.withColumn("sales_delta", col("
    TotalPrice") - col("rolling_3m_avg"))

features = df.withColumn("InvoiceMonthYM",
    date_format(col("InvoiceMonth"), "yyyy-MM"
    ))
features = (
    features.join(
        holiday_months, features.
            InvoiceMonthYM == holiday_months.
            holiday_month, "left"
    )
    .withColumn("is_holiday_month", col("
        holiday_month").isNotNull().cast("int"
        ))
    .drop("holiday_month")
    .drop("InvoiceMonthYM")
)

output_path = "output/features"
features.coalesce(1).write.option("header", "
    true").csv(output_path, mode="overwrite")
log.info(f"Feature set (with holiday flags)
    written to: {output_path}")

from pyspark.sql.functions import when

features = features.withColumn("sales_delta",
    col("TotalPrice") - col("rolling_3m_avg"))
features = features.withColumn(
    "stock_decision",
    when(col("TotalPrice") > col("
        rolling_3m_avg") * 1.2, "Increase")
    .when(col("TotalPrice") < col("
        rolling_3m_avg") * 0.8, "Reduce")
    .otherwise("Maintain"),
)

output_path = "output/features_with_labels"
features.coalesce(1).write.option("header", "
    true").csv(output_path, mode="overwrite")
log.info(f"Feature set with labels written to:
     {output_path}")
```

Code Block 2. PySpark Feature Engineering

For each product–month pair, we computed:

- prev_month_sales: sales from the previous month
- rolling_3m_avg: rolling average of sales over the past 3 months
- sales_delta: deviation from the rolling average
- is_holiday_month: binary indicator if the month contains a UK public holiday
- Month: calendar month number
- DescriptionIndex: product-level categorical encoding using StringIndexer

C. Modeling Strategy

The next step was building the models. The following codes will show the regression and classification model.

```
import os
import logging
import json
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
    , StringIndexer
from pyspark.ml.regression import
    RandomForestRegressor
from pyspark.ml.evaluation import
    RegressionEvaluator
from pyspark.ml.tuning import
    TrainValidationSplit, ParamGridBuilder

LOG_DIR = "logs"
os.makedirs(LOG_DIR, exist_ok=True)

logging.basicConfig(
    filename=os.path.join(LOG_DIR, "
        train_model.log"),
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(
        message)s",
)
log = logging.getLogger(__name__)
log.info("Starting training script")

spark = SparkSession.builder.appName(
    "Retail Demand Forecasting - Train Model"
).getOrCreate()

log.info("Spark session started")

df = spark.read.option("header", "true").csv("
    output/features", inferSchema=True)
log.info(f"Loaded features with {df.count()}
    rows")

indexer = StringIndexer(inputCol="Description"
    , outputCol="DescriptionIndex")
df = indexer.fit(df).transform(df)
feature_cols = [
    "prev_month_sales",  # retains lag signal
    "sales_delta",  # better than Month
    "rolling_3m_avg",  # strongest signal
    "is_holiday_month",  # keep as binary
        context
    "DescriptionIndex",  # keep for product-
        specific bias
]

assembler = VectorAssembler(inputCols=
    feature_cols, outputCol="features")
df = assembler.transform(df)

train_data, val_data, test_data = df.
    randomSplit([0.7, 0.15, 0.15], seed=42)
log.info(
    f"Train: {train_data.count()}, Val: {
        val_data.count()}, Test: {test_data.
        count()}"
)

rf = RandomForestRegressor(featuresCol="
    features", labelCol="TotalPrice", maxBins
    =2048)

paramGrid = (
    ParamGridBuilder()
    .addGrid(rf.numTrees, [10, 20])
    .addGrid(rf.maxDepth, [5, 10])
    .build()
)

evaluator = RegressionEvaluator(
    labelCol="TotalPrice", predictionCol="
        prediction", metricName="rmse"
)

tvs = TrainValidationSplit(
    estimator=rf, estimatorParamMaps=paramGrid
        , evaluator=evaluator, trainRatio=0.8
)

model = tvs.fit(train_data)
log.info("Model training and validation
    complete")
# Log best hyperparameters
best_model = model.bestModel
log.info(f"Best numTrees: {best_model.
    getNumTrees}")
log.info(f"Best maxDepth: {best_model.
    getOrDefault('maxDepth')}")

predictions = model.transform(test_data)
rmse = evaluator.evaluate(predictions)
mae = RegressionEvaluator(
    labelCol="TotalPrice", predictionCol="
        prediction", metricName="mae"
).evaluate(predictions)

log.info(f"Test RMSE: {rmse:.2f}")
log.info(f"Test MAE: {mae:.2f}")

predictions.select("Description", "
    InvoiceMonth", "TotalPrice", "prediction")
    .coalesce(
    1
).write.option("header", "true").csv("output/
    predictions", mode="overwrite")

log.info("Predictions written to output/
    predictions/")


metrics = {
    "RMSE": round(rmse, 2),
    "MAE": round(mae, 2),
    "Best Hyperparameters": {
        "numTrees": best_model.getNumTrees,
        "maxDepth": best_model.getOrDefault("
            maxDepth"),
    },
}
```

```
output_json_path = "output/predictions/
    model_metrics.json"
with open(output_json_path, "w") as f:
    json.dump(metrics, f, indent=4)

log.info(f"Exported model metrics to: {
    output_json_path}")
```

Code Block 3. PySpark Regression model

1) Regression:

- **Model**: `RandomForestRegressor`
- **Target**: `TotalPrice`
- **Objective**: Forecast dollar sales per product-month
- **Tuning**: Grid search over `numTrees` ∈ {10, 20}, `maxDepth` ∈ {5, 10}
- **Validation**: Train/Val/Test split (70/15/15)

```
import os
import json
import logging
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer,
    VectorAssembler, IndexToString
from pyspark.ml.classification import
    RandomForestClassifier
from pyspark.ml.evaluation import
    MulticlassClassificationEvaluator
from pyspark.ml.tuning import
    TrainValidationSplit, ParamGridBuilder

LOG_DIR = "logs"
os.makedirs(LOG_DIR, exist_ok=True)

logging.basicConfig(
    filename=os.path.join(LOG_DIR, "
        train_inventory_model.log"),
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(
        message)s",
)
log = logging.getLogger(__name__)
log.info("Starting inventory classifier
    training")

spark = SparkSession.builder.appName("
    Inventory Planning Classifier").
    getOrCreate()
log.info("Spark session started")

df = spark.read.option("header", "true").csv(
    "output/features_with_labels", inferSchema
        =True
)
log.info(f"Loaded {df.count()} rows")

label_indexer = StringIndexer(inputCol="
    stock_decision", outputCol="label")
label_model = label_indexer.fit(df).transform(
    df)

desc_indexer = StringIndexer(inputCol="
    Description", outputCol="DescriptionIndex"
    )
df = desc_indexer.fit(df).transform(df)
```

```
feature_cols = [
    "prev_month_sales",
    "sales_delta",  # New, medium correlation
    "is_holiday_month",  # Categorical binary
    "DescriptionIndex",  # Or
        ProductCategoryIndex
]


assembler = VectorAssembler(inputCols=
    feature_cols, outputCol="features")
df = assembler.transform(df)

train_data, test_data = df.randomSplit([0.8,
    0.2], seed=42)
log.info(f"Train: {train_data.count()}, Test:
    {test_data.count()}")

rf = RandomForestClassifier(labelCol="label",
    featuresCol="features", maxBins=2048)
paramGrid = (
    ParamGridBuilder()
    .addGrid(rf.numTrees, [10, 20])
    .addGrid(rf.maxDepth, [5, 10])
    .build()
)

evaluator = MulticlassClassificationEvaluator(
    labelCol="label", predictionCol="
        prediction", metricName="accuracy"
)
tvs = TrainValidationSplit(
    estimator=rf, estimatorParamMaps=paramGrid
        , evaluator=evaluator, trainRatio=0.8
)

model = tvs.fit(train_data)
log.info("Model training and validation
    complete")

predictions = model.transform(test_data)

metrics = {
    "accuracy": evaluator.evaluate(predictions
        , {evaluator.metricName: "accuracy"}),
    "f1": MulticlassClassificationEvaluator(
        labelCol="label", predictionCol="
            prediction", metricName="f1"
    ).evaluate(predictions),
    "weightedPrecision":
        MulticlassClassificationEvaluator(
        labelCol="label", predictionCol="
            prediction", metricName="
            weightedPrecision"
    ).evaluate(predictions),
    "weightedRecall":
        MulticlassClassificationEvaluator(
        labelCol="label", predictionCol="
            prediction", metricName="
            weightedRecall"
    ).evaluate(predictions),
}

for metric, value in metrics.items():
    log.info(f"{metric.capitalize()}: {value
        :.4f}")
label_converter = IndexToString(
    inputCol="prediction", outputCol="
        predicted_label", labels=label_model.
        labels
)
predictions = label_converter.transform(
    predictions)

predictions.select(
    "Description", "InvoiceMonth", "
        stock_decision", "predicted_label"
).coalesce(1).write.option("header", "true").
    csv(
    "output/predictions_inventory", mode="
        overwrite"
)

log.info("Inventory prediction output written
    to: output/predictions_inventory/")

metrics_output_path = "output/
    predictions_inventory/
    model_metrics_inventory.json"
with open(metrics_output_path, "w") as f:
    json.dump({k: round(v, 4) for k, v in
        metrics.items()}, f, indent=4)

log.info(f"Exported inventory classifier
    metrics to: {metrics_output_path}")
```

Code Block 4. PySpark Classification model

Validation was performed using TrainValidationSplit with a 70/15/15 train/val/test ratio.

2) Classification:

- **Model**: `RandomForestClassifier`
- **Target**: `stock_decision` (Increase, Maintain, Reduce)
- **Labeling logic**:
  - Increase: sales greater than 120% of 3-month avg
  - Reduce: sales less than 80% of 3-month avg
  - Maintain: everything else
- **Metric**: Accuracy, F1, precision, recall

IV. Results and Analysis

This section presents the evaluation results of both the regression and classification models developed in this project, along with the key visualizations that illustrate their performance.

A. Regression Results

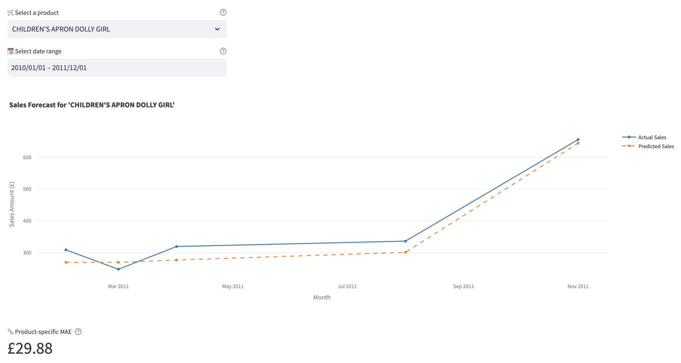
The regression model was trained to forecast the `TotalPrice` for each product-month using the engineered features. This model aimed to capture overall sales trends, leveraging features like `prev_month_sales`, `rolling_3m_avg`, and `is_holiday_month` to provide accurate predictions.

The metrics in Table I indicate that the regression model effectively captures overall sales trends. An RMSE of **£373.99** suggests that, on average, the model's predictions are within

TABLE I
REGRESSION MODEL PERFORMANCE

Metric	Value
RMSE (Root Mean Square Error)	**£373.99**
MAE (Mean Absolute Error)	**£195.06**
Best Parameters	`numTrees=20, maxDepth=5`

£374 of actual monthly sales, while the MAE of **£195.06** reflects a tighter average error. These values are reasonable given that many product-months range from **£500** to **£15,000+** in sales, confirming the model's reliability for high-level demand forecasting.



Select a product: CHILDREN'S APRON DOLLY GIRL

Select date range: 2010/01/01 – 2011/12/01

Product-specific MAE

£29.88

Fig. 8. Sales Forecast for "CHILDREN'S APRON DOLLY GIRL". This graph compares actual and predicted sales, showing the model's ability to capture overall trends despite occasional lag in sharp spikes.

B. *Classification Results (Inventory Planning)*

The classification model was trained to predict `stock_decision` (Increase, Maintain, Reduce) for each product-month. This model is particularly well-suited for inventory planning, as it captures critical decision points using features like `sales_delta`, `prev_month_sales`, and `is_holiday_month`.

TABLE II
CLASSIFICATION MODEL PERFORMANCE

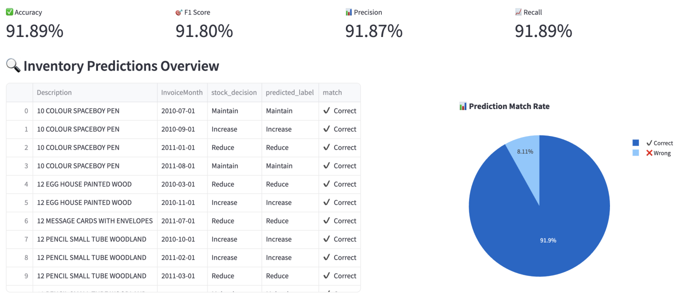
Metric	Value
Accuracy	**91.89%**
F1 Score	**91.80%**
Precision	**91.87%**
Recall	**91.89%**

The metrics in Table II indicate that the classification model is highly reliable for stock decision-making. With an accuracy of **91.89%**, the model correctly predicted stock actions (Increase, Maintain, Reduce) in most cases, supported by balanced precision and recall scores.

C. *Visualizations and Output*

All model outputs — including the predictions, class assignments, and metrics — were rendered in a **Streamlit dashboard** for interactive exploration. Key visualizations include:

- **Sales Forecast** – Actual vs predicted sales for each product (Figure 8).



Accuracy	F1 Score	Precision	Recall
91.89%	91.80%	91.87%	91.89%

Inventory Predictions Overview

	Description	InvoiceMonth	stock_decision	predicted_label	match
0	10 COLOUR SPACEBOY PEN	2010-07-01	Maintain	Maintain	✓ Correct
1	10 COLOUR SPACEBOY PEN	2010-09-01	Increase	Increase	✓ Correct
2	10 COLOUR SPACEBOY PEN	2011-01-01	Reduce	Reduce	✓ Correct
3	10 COLOUR SPACEBOY PEN	2011-08-01	Maintain	Maintain	✓ Correct
4	12 EGG HOUSE PAINTED WOOD	2010-03-01	Reduce	Reduce	✓ Correct
5	12 EGG HOUSE PAINTED WOOD	2010-11-01	Increase	Increase	✓ Correct
6	12 MESSAGE CARDS WITH ENVELOPES	2011-07-01	Reduce	Reduce	✓ Correct
7	12 PENCIL SMALL TUBE WOODLAND	2010-10-01	Increase	Increase	✓ Correct
8	12 PENCIL SMALL TUBE WOODLAND	2011-02-01	Increase	Increase	✓ Correct
9	12 PENCIL SMALL TUBE WOODLAND	2011-03-01	Reduce	Reduce	✓ Correct

Fig. 9. Inventory Predictions Overview. This chart visualizes the model's overall prediction match rate, showing that 91.9% of decisions were correct, with only 8.1% incorrect.

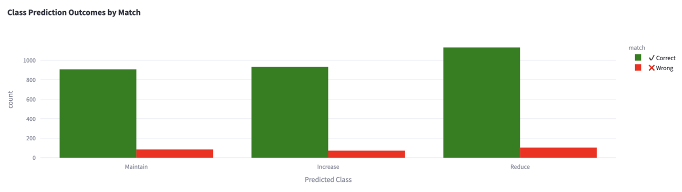


Fig. 10. Class Distribution of Predictions. This bar chart breaks down the model's classification performance by stock action, highlighting its strength in correctly identifying Increase, Maintain, and Reduce decisions.

- **Inventory Predictions Overview** – Pie chart showing classifier correctness (Figure 9).
- **Class Distribution of Predictions** – Bar chart breaking down the accuracy of Increase, Maintain, and Reduce decisions (Figure 10).

These visualizations provide critical insights into our model performance, highlighting where the models excel and where they face challenges, such as capturing sharp sales spikes or maintaining balanced class distributions.

V. DISCUSSION

This project highlights the value of combining scalable data processing with interpretable machine learning to solve a high-impact business problem: **retail demand forecasting**.

A. *Impact and Insights*

Our pipeline proved effective in producing both **forecasting models** and **operational inventory decisions** from raw transactional data. Key takeaways include:

- **Holiday and seasonal signals matter.** Contrary to expectations, **Spring Bank Holiday** and **Substitute Holidays** drove more average sales than Christmas. This kind of insight is only visible through granular, time-aware analysis.
- **Post-holiday demand is underestimated.** In multiple years, **January sales exceeded December**, likely due to gift redemption, returns, or delayed buying behavior.
- **Classifier success reveals feature strength.** With over **91% accuracy**, our inventory classifier shows that features like `sales_delta`, `prev_month_sales`, and `holiday_month` are not only predictive, but **actionable**.

- **Forecasting works better for stable products.** Product-specific MAE analysis revealed that models perform best on consistently selling items. Products with volatile or sparse data show higher error — a known challenge in demand modeling.

B. Challenges Faced

TABLE III
CHALLENGES AND SOLUTIONS

Challenge	How We Handled It
Dirty and imbalanced data	Filtered cancellations, removed nulls, used class balancing
High-cardinality product labels	Used `StringIndexer`, explored NLP clusters and product categories
Outlier impact on regression	Added `sales_delta` and rolling avg features
Classification label overlap	Tuned decision thresholds to separate Increase/Maintain
Streamlit chart readability	Fixed truncated names, added sorting, improved axis handling

C. Future Opportunities

- Move to **weekly forecasts** for short-term inventory planning.
- Integrate **profit or margin** data to prioritize profitable items.
- Switch to **real-time forecasting** using Spark Structured Streaming or Kafka.
- Apply **Word2Vec or BERT** for richer product embeddings.

VI. Conclusion

This project successfully demonstrated how scalable data processing and interpretable machine learning can be combined to solve a complex, real-world business problem — **retail demand forecasting and inventory planning**.

By processing over **1 million transaction records**, we developed a complete pipeline that:

- Forecasts monthly product-level demand using a **Random Forest regression model**.
- Recommends inventory actions using a **3-class classification model**.
- Incorporates **holiday effects and seasonality** into both modeling and analysis.

The final models achieved:

- **RMSE of £373.99** and **MAE of £195.06** for sales prediction.
- **92% classification accuracy** for inventory recommendation.

We presented results through an interactive Streamlit dashboard, giving end users — such as business managers or supply chain teams — clear visualizations and actionable insights.

A. Future Work

This work lays the foundation for a more advanced retail planning system. Future enhancements could include:

- Weekly or real-time forecasting.
- Incorporating **external signals** (e.g., promotions, weather).
- Moving from descriptive insights to **profit-driven decision support**.

Ultimately, this project shows that big data techniques are not just scalable — they are essential for uncovering meaningful trends and decisions in high-volume, volatile domains like retail.

Our architecture was fully local, reproducible, and modular — powered by Spark, MLlib, and Streamlit. The models delivered strong performance while remaining interpretable and deployment-friendly.

References

1 UCI Machine Learning Repository – Online Retail II Data Set. Available: https://archive.ics.uci.edu/ml/datasets/Online+Retail+II

2 Apache Spark MLlib Documentation. Available: https://spark.apache.org/docs/latest/ml-guide.html

3 Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. Available: https://doi.org/10.1023/A:1010933404324

4 Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice* (2nd ed.). OTexts. Available: https://otexts.com/fpp2/

5 Taylor, B., & Letham, E. (2017). Forecasting at Scale. *PeerJ Preprints*. (Related to Prophet and holiday-aware forecasting).

6 Streamlit Docs – Interactive Dashboards. Available: https://docs.streamlit.io

7 PySpark MLlib Feature Engineering. Available: https://spark.apache.org/docs/latest/ml-features.html

8 Retail Demand Big Data. Available: https://github.com/irangareddy/demand-forecasting-bigdata