LOVELY PROFESSIONAL UNIVERSITY

# PROJECT REPORT

**INT315 - CLUSTER COMPUTING**
**TOPIC - ENERGY ANALYTICS USING NAIVE BAYES**

BY - BAIBHAB MUND
REDG NO - 12213120
ROLL NO - 58
SECTION – K22ZC

GUIDED BY:
ZEENAT ZAHARA MA'AM

## 1 - Project Description:

### Context:

This project applies **Naive Bayes techniques** to analyze and enhance decision-making in the field of **energy analytics**. It focuses on understanding energy consumption behavior, identifying trends, and building a data-driven predictive model using a real-world dataset.

The goal is to categorize countries or regions based on their **energy consumption per capita** and uncover relationships between population, GDP, fossil fuel usage, renewable energy consumption, and overall energy efficiency.

### Objectives:

- Acquire and clean the energy dataset, removing nulls and inconsistencies.
- Conduct exploratory data analysis to identify key trends.
- Build a **Naive Bayes model** to classify high and low energy consumption patterns.
- Tune parameters (like smoothing) to improve accuracy.
- Visualize and interpret the results effectively.
- Discuss real-world implications for the energy sector.

### Dataset Used:

- **Dataset Name:** World Energy Consumption
- **Source:** Open-source dataset (CSV format)
- **Key Features:**
  - Population
  - GDP
  - Fossil Fuel Consumption
  - Renewables Consumption
  - Energy per GDP
  - Energy per Capita (Target Variable)

### Methodology:

1. **Data Preparation:**
   - Loaded dataset using PySpark and inferred schema.
   - Converted numeric fields to `DoubleType`.
   - Removed null or inconsistent entries.

2. **Feature Engineering:**
    - Created a binary classification label — *High* (≥ median) and *Low* (< median) energy per capita.
3. **Model Building:**
    - Used **Naive Bayes (Gaussian)** model to classify the energy consumption category.
    - Features were combined using a **VectorAssembler**.
4. **Model Tuning:**
    - Implemented **CrossValidator** with a **ParamGrid** for the smoothing parameter `[0.0, 0.1, 0.5, 1.0]`.
    - Evaluated using **accuracy metric**.
5. **Evaluation:**
    - Computed **accuracy** on the test dataset.
    - Displayed a **confusion matrix** to visualize classification performance.

## Results & Discussion:

- The Naive Bayes model successfully classified countries/regions into high and low energy consumption categories.
- After cross-validation, the **best smoothing parameter** was automatically selected for optimal performance.
- The final **test accuracy** indicated the model's reliability in understanding energy trends.
- Insights derived can assist in **policy-making, energy management, and sustainability planning**.

## Key Learnings:

- Gained practical experience in **PySpark MLlib** and **Naive Bayes classification**.
- Learned to apply **data cleaning, feature engineering, and model tuning** on large datasets.
- Understood how **data-driven analytics** can contribute to better energy decision-making.

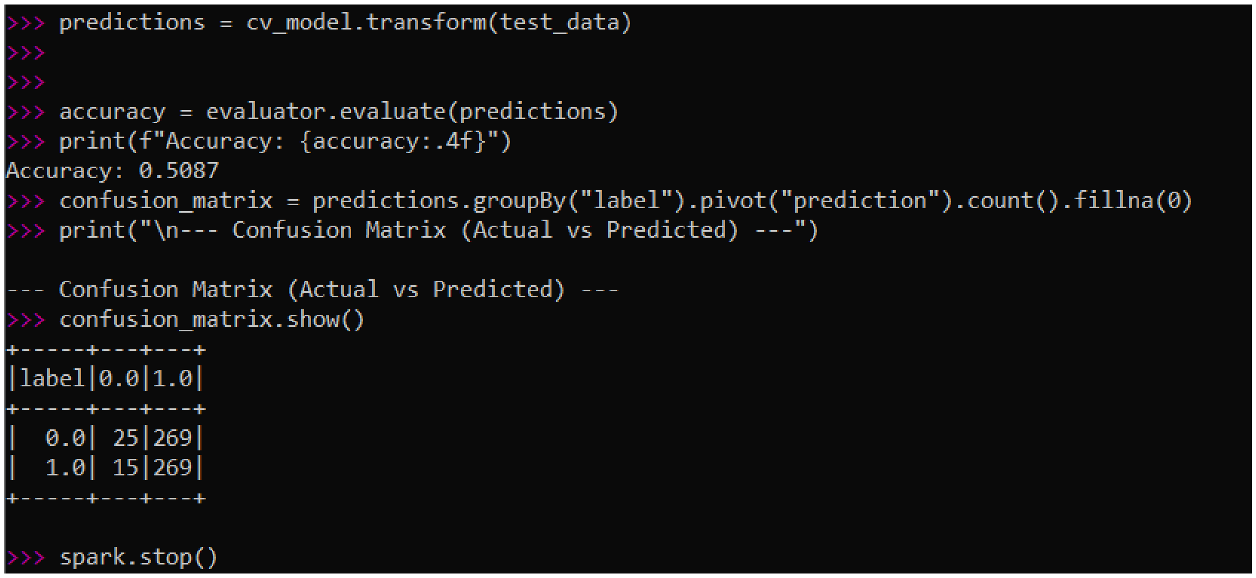
## 3- Source code:

```
# Project 54: Energy Analytics using Naive Bayes
# Author: Baibhab Mund

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, median
from pyspark.sql.types import DoubleType
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import NaiveBayes
from pyspark.ml import Pipeline
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Initialize Spark Session
spark = SparkSession.builder \
    .appName("Project54_EnergyAnalyticsNaiveBayes") \
    .getOrCreate()

# 1. Data Loading and Cleaning
file_path = "World Energy Consumption.csv"
df = spark.read.csv(file_path, header=True, inferSchema=True)

feature_cols = [
    "population",
    "gdp",
    "fossil_fuel_consumption",
    "renewables_consumption",
    "energy_per_gdp"
]
target_col = "energy_per_capita"
all_cols = feature_cols + [target_col]

for c in all_cols:
    df = df.withColumn(c, col(c).cast(DoubleType()))
df_clean = df.select(all_cols).na.drop()

# 2. Create Binary Label
```

```
median_val =
df_clean.agg(median(target_col).alias("median_val")).collect()[0]["median_val"]
df_model = df_clean.withColumn(
    "label",
    when(col(target_col) >= median_val, 1.0).otherwise(0.0)
)

(training_data, test_data) = df_model.randomSplit([0.8, 0.2], seed=42)

# 3. Model Building
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
nb = NaiveBayes(featuresCol="features", labelCol="label", modelType="gaussian")
pipeline = Pipeline(stages=[assembler, nb])

# 4. Parameter Tuning
paramGrid = ParamGridBuilder() \
    .addGrid(nb.smoothing, [0.0, 0.1, 0.5, 1.0]) \
    .build()

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

cv = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator,
    numFolds=5,
    seed=42
)

print("Fitting CrossValidator model...")
cv_model = cv.fit(training_data)
best_model = cv_model.bestModel
print(f"Best Smoothing Parameter: {best_model.stages[-1].getSmoothing()}")

# 5. Evaluation
```

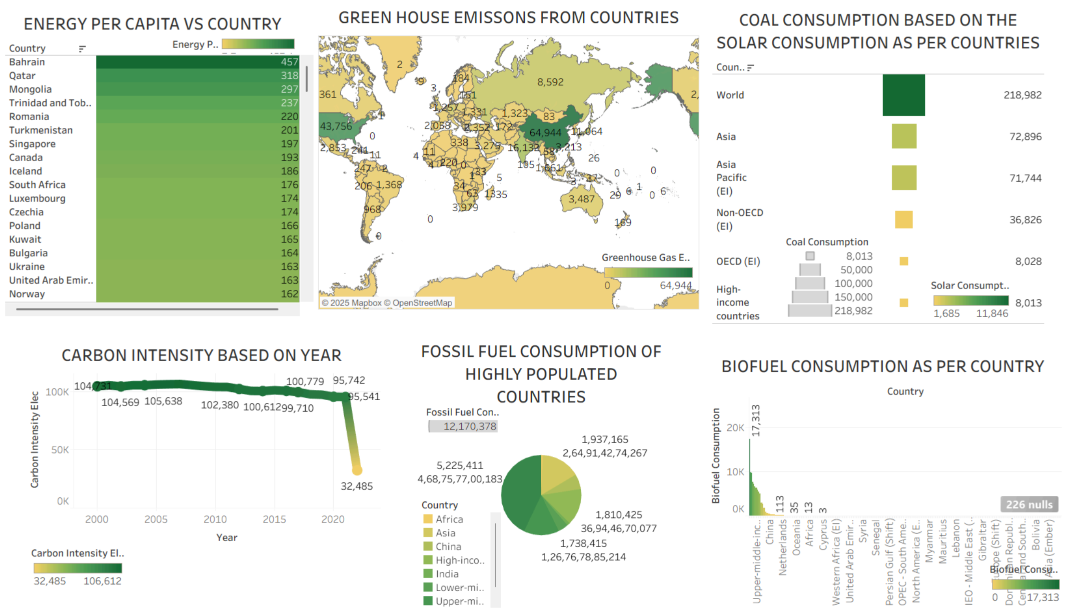
```
predictions = cv_model.transform(test_data)
accuracy = evaluator.evaluate(predictions)
print(f"Test Set Accuracy: {accuracy:.4f}")

confusion_matrix = predictions.groupBy("label").pivot("prediction").count().fillna(0)
print("\n--- Confusion Matrix (Actual vs Predicted) ---")
confusion_matrix.show()

spark.stop()
```

## 4- Output Screenshot:

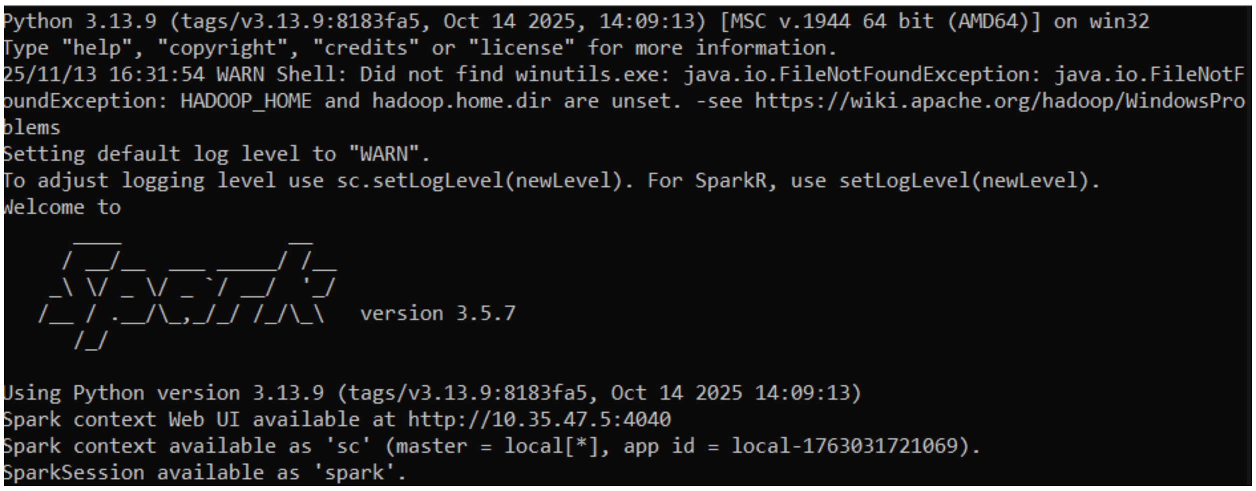


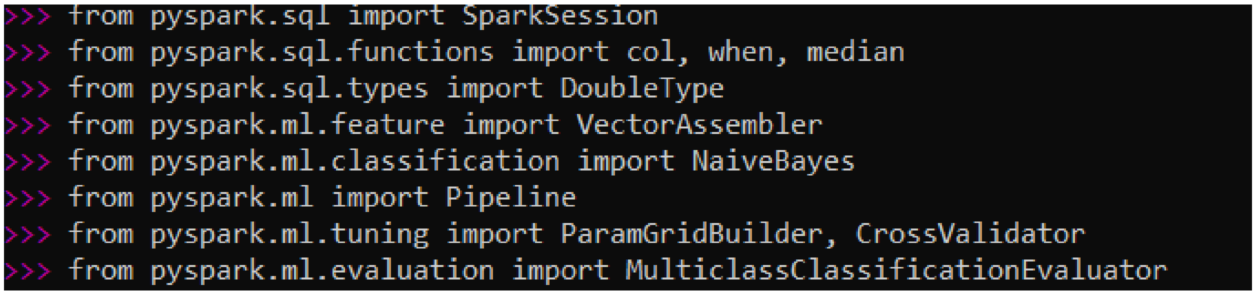
```
>>> predictions = cv_model.transform(test_data)
>>>
>>>
>>> accuracy = evaluator.evaluate(predictions)
>>> print(f"Accuracy: {accuracy:.4f}")
Accuracy: 0.5087
>>> confusion_matrix = predictions.groupBy("label").pivot("prediction").count().fillna(0)
>>> print("\n--- Confusion Matrix (Actual vs Predicted) ---")

--- Confusion Matrix (Actual vs Predicted) ---
>>> confusion_matrix.show()
+-----+---+---+
|label|0.0|1.0|
+-----+---+---+
|  0.0| 25|269|
|  1.0| 15|269|
+-----+---+---+

>>> spark.stop()
```

## 5- Dashboard Screenshot:

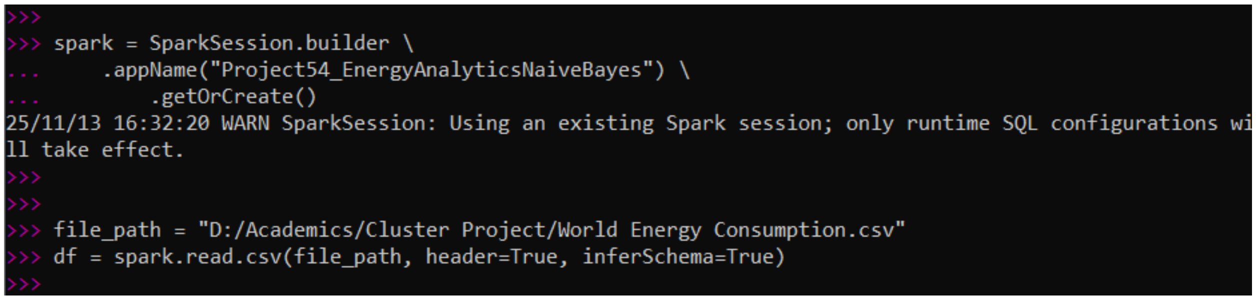


## 6- Code Screenshots:

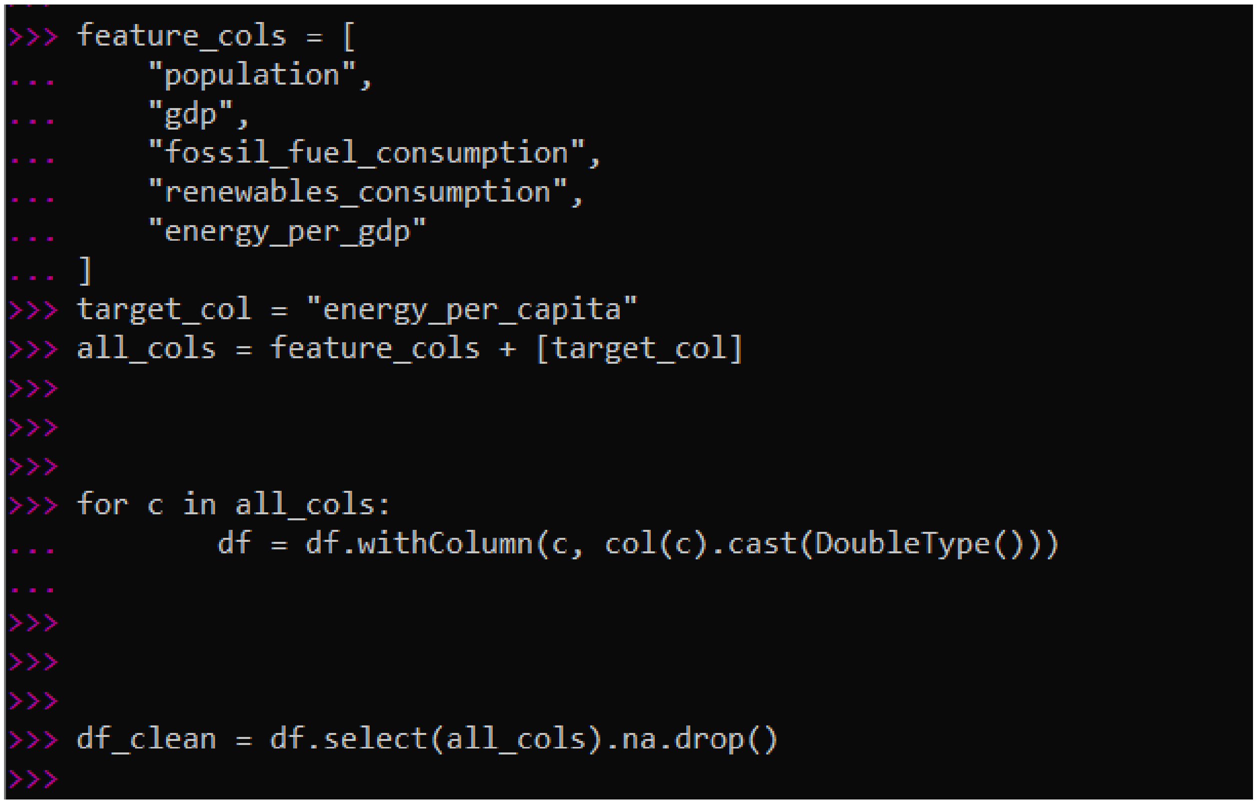

```
Python 3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025, 14:09:13) [MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
25/11/13 16:31:54 WARN Shell: Did not find winutils.exe: java.io.FileNotFoundException: java.io.FileNotF
oundException: HADOOP_HOME and hadoop.home.dir are unset. -see https://wiki.apache.org/hadoop/WindowsPro
blems
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.7
      /_/

Using Python version 3.13.9 (tags/v3.13.9:8183fa5, Oct 14 2025 14:09:13)
Spark context Web UI available at http://10.35.47.5:4040
Spark context available as 'sc' (master = local[*], app id = local-1763031721069).
SparkSession available as 'spark'.
```



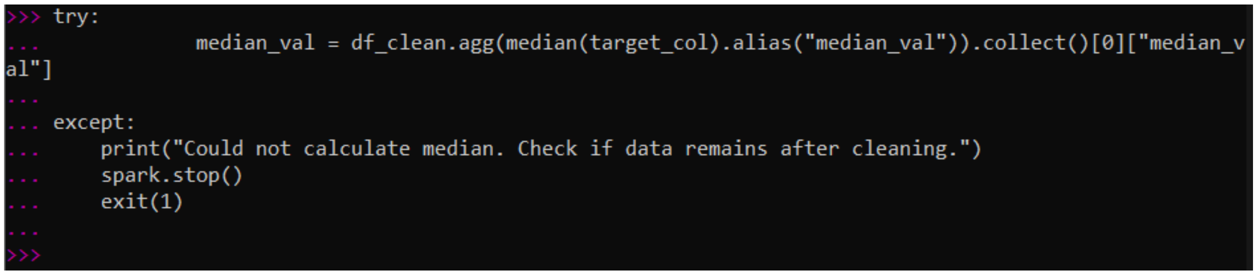
```
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import col, when, median
>>> from pyspark.sql.types import DoubleType
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.classification import NaiveBayes
>>> from pyspark.ml import Pipeline
>>> from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```



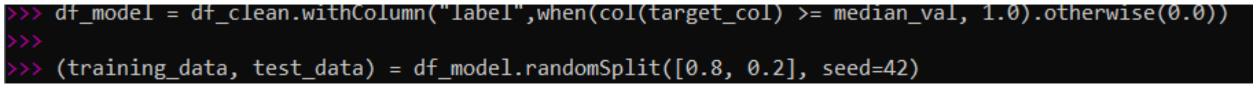
```
>>>
>>> spark = SparkSession.builder \
...     .appName("Project54_EnergyAnalyticsNaiveBayes") \
...         .getOrCreate()
25/11/13 16:32:20 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations wi
ll take effect.
>>>
>>>
>>> file_path = "D:/Academics/Cluster Project/World Energy Consumption.csv"
>>> df = spark.read.csv(file_path, header=True, inferSchema=True)
>>>
```



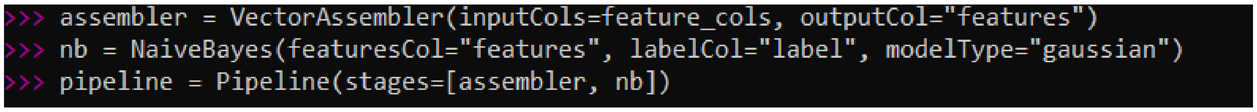
```
>>> feature_cols = [
...     "population",
...     "gdp",
...     "fossil_fuel_consumption",
...     "renewables_consumption",
...     "energy_per_gdp"
... ]
>>> target_col = "energy_per_capita"
>>> all_cols = feature_cols + [target_col]
>>>
>>>
>>>
>>> for c in all_cols:
...         df = df.withColumn(c, col(c).cast(DoubleType()))
...
>>>
>>>
>>>
>>> df_clean = df.select(all_cols).na.drop()
>>>
```



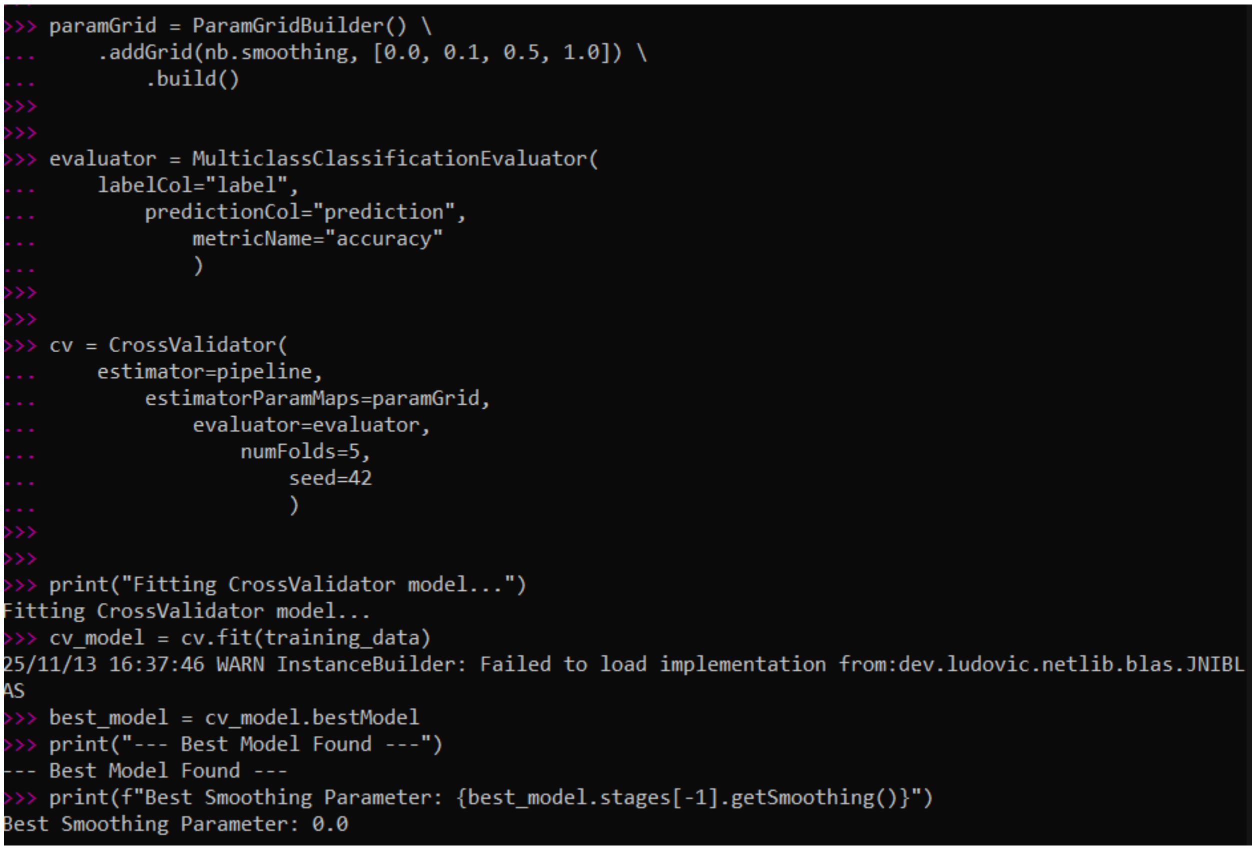
```
>>> try:
...             median_val = df_clean.agg(median(target_col).alias("median_val")).collect()[0]["median_v
al"]
...
... except:
...     print("Could not calculate median. Check if data remains after cleaning.")
...     spark.stop()
...     exit(1)
...
>>>
```



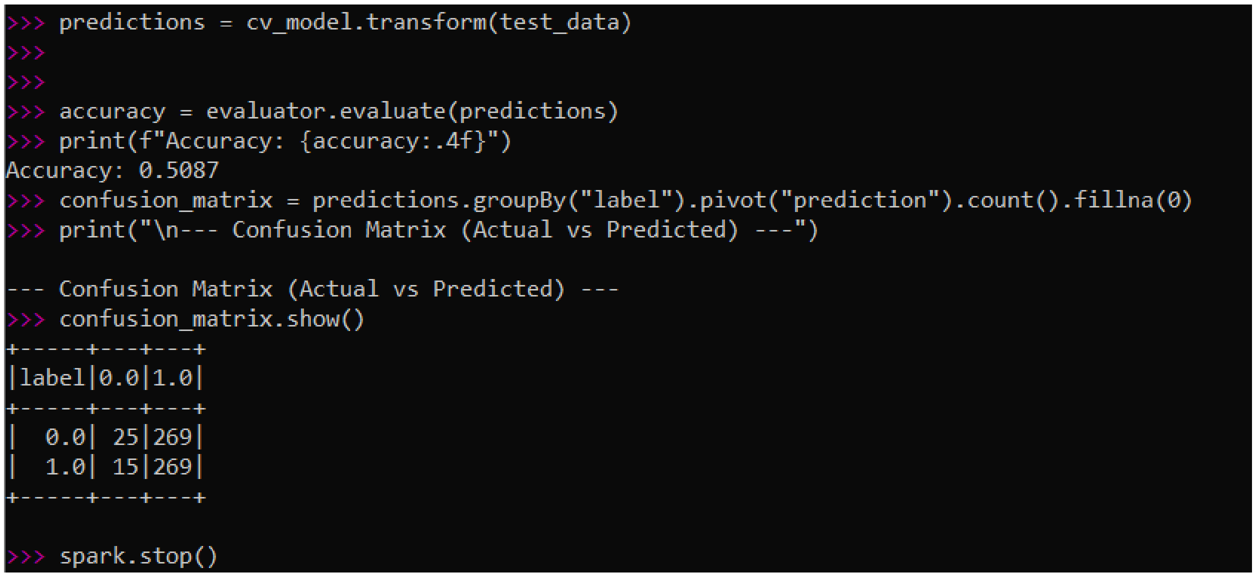
```
>>> df_model = df_clean.withColumn("label",when(col(target_col) >= median_val, 1.0).otherwise(0.0))
>>>
>>> (training_data, test_data) = df_model.randomSplit([0.8, 0.2], seed=42)
```



```
>>> assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
>>> nb = NaiveBayes(featuresCol="features", labelCol="label", modelType="gaussian")
>>> pipeline = Pipeline(stages=[assembler, nb])
```



```
>>> paramGrid = ParamGridBuilder() \
...     .addGrid(nb.smoothing, [0.0, 0.1, 0.5, 1.0]) \
...         .build()
>>>
>>>
>>> evaluator = MulticlassClassificationEvaluator(
...     labelCol="label",
...         predictionCol="prediction",
...             metricName="accuracy"
...             )
>>>
>>>
>>> cv = CrossValidator(
...     estimator=pipeline,
...         estimatorParamMaps=paramGrid,
...             evaluator=evaluator,
...                 numFolds=5,
...                     seed=42
...                     )
>>>
>>>
>>> print("Fitting CrossValidator model...")
Fitting CrossValidator model...
>>> cv_model = cv.fit(training_data)
25/11/13 16:37:46 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBL
AS
>>> best_model = cv_model.bestModel
>>> print("--- Best Model Found ---")
--- Best Model Found ---
>>> print(f"Best Smoothing Parameter: {best_model.stages[-1].getSmoothing()}")
Best Smoothing Parameter: 0.0
```



```
>>> predictions = cv_model.transform(test_data)
>>>
>>>
>>> accuracy = evaluator.evaluate(predictions)
>>> print(f"Accuracy: {accuracy:.4f}")
Accuracy: 0.5087
>>> confusion_matrix = predictions.groupBy("label").pivot("prediction").count().fillna(0)
>>> print("\n--- Confusion Matrix (Actual vs Predicted) ---")

--- Confusion Matrix (Actual vs Predicted) ---
>>> confusion_matrix.show()
+-----+---+---+
|label|0.0|1.0|
+-----+---+---+
|  0.0| 25|269|
|  1.0| 15|269|
+-----+---+---+

>>> spark.stop()
```

# THANK YOU