```
from pyspark.sql import SparkSession
from pyspark.sql.functions import when, col, trim, lower
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.sql.types import DoubleType

# Initialize Spark session
spark = SparkSession.builder \
    .appName("TrafficPrediction") \
    .getOrCreate()

# Load CSV file
df = spark.read.csv("/content/Traffic.csv", header=True, inferSchema=True)
print("Total rows before processing:", df.count())

# Clean up Traffic Situation column (remove whitespace and lowercase)
df = df.withColumn("Traffic Situation", trim(lower(col("Traffic Situation"))))

# Show distinct traffic situation values for debugging
df.select("Traffic Situation").distinct().show(truncate=False)

# Map string labels to integers
df = df.withColumn(
    "Traffic Situation",
    when(col("Traffic Situation") == "low", 0)
    .when(col("Traffic Situation") == "moderate", 1)
    .when(col("Traffic Situation") == "heavy", 2)
    .otherwise(None)
)

# Drop rows with nulls in 'Traffic Situation'
```

```
df = df.dropna(subset=["Traffic Situation"])
print("Rows after mapping:", df.count())

# Convert Traffic Situation to DoubleType (required by MLlib)
df = df.withColumn("Traffic Situation", col("Traffic Situation").cast(DoubleType()))

# Feature columns
feature_cols = ["CarCount", "BikeCount", "BusCount", "TruckCount", "Total"]

# Assemble features into vector
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
df = assembler.transform(df)

# Split into train/test
train_df, test_df = df.randomSplit([0.8, 0.2], seed=42)
print("Train rows:", train_df.count())
print("Test rows:", test_df.count())

# Train Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol="Traffic Situation")
model = lr.fit(train_df)

# Predict on test set
predictions = model.transform(test_df)

# Map predictions to Traffic Situation categories (0, 1, 2)
predictions = predictions.withColumn(
    "Predicted Traffic Situation",
    when(col("prediction") < 0.5, 0)
    .when((col("prediction") >= 0.5) & (col("prediction") < 1.5), 1)
    .otherwise(2)
```

```
)


# Show predictions with mapped traffic situations

predictions.select("features", "Traffic Situation", "prediction", "Predicted Traffic Situation").show(10, truncate=False)


# Stop Spark session

spark.stop()
```