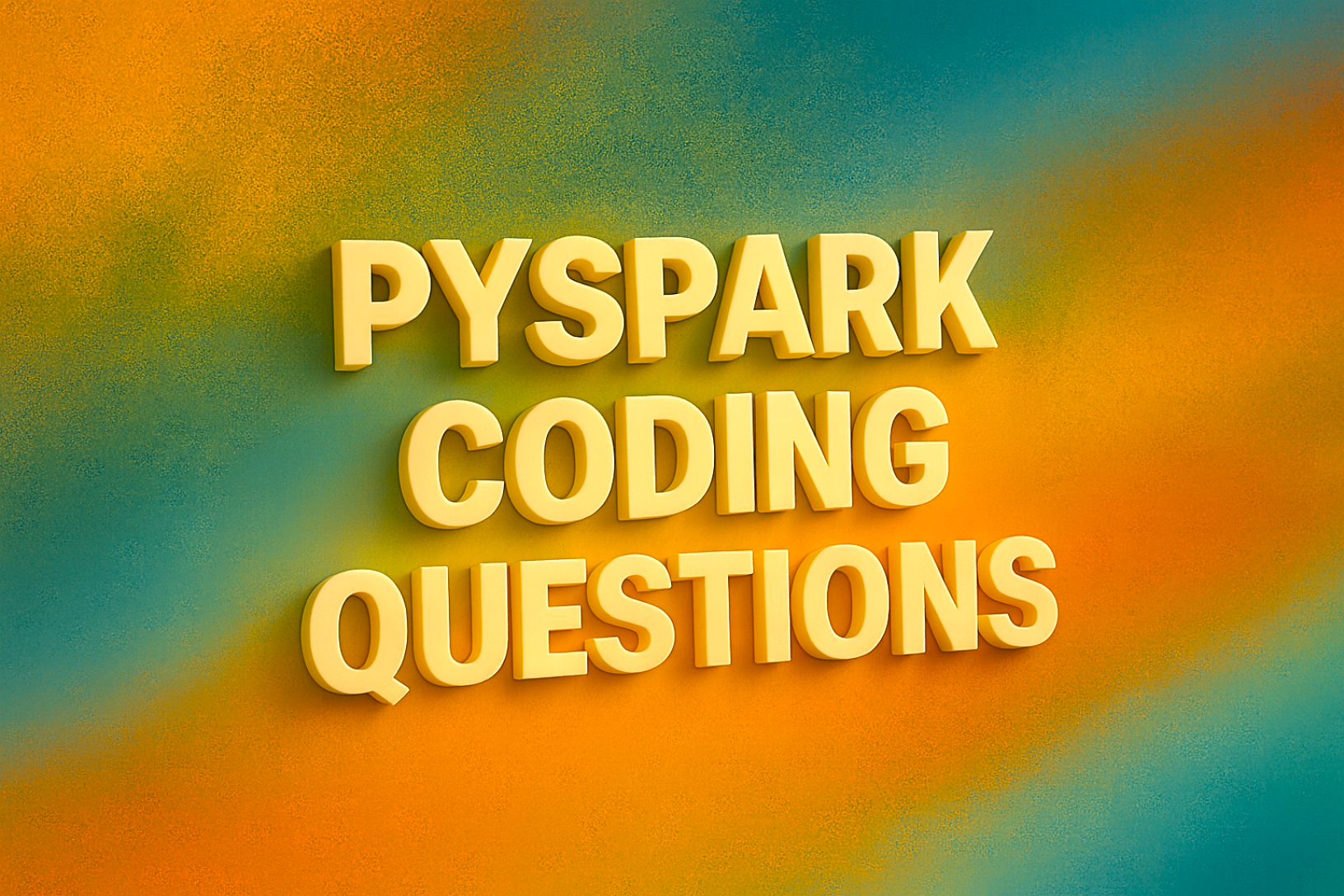
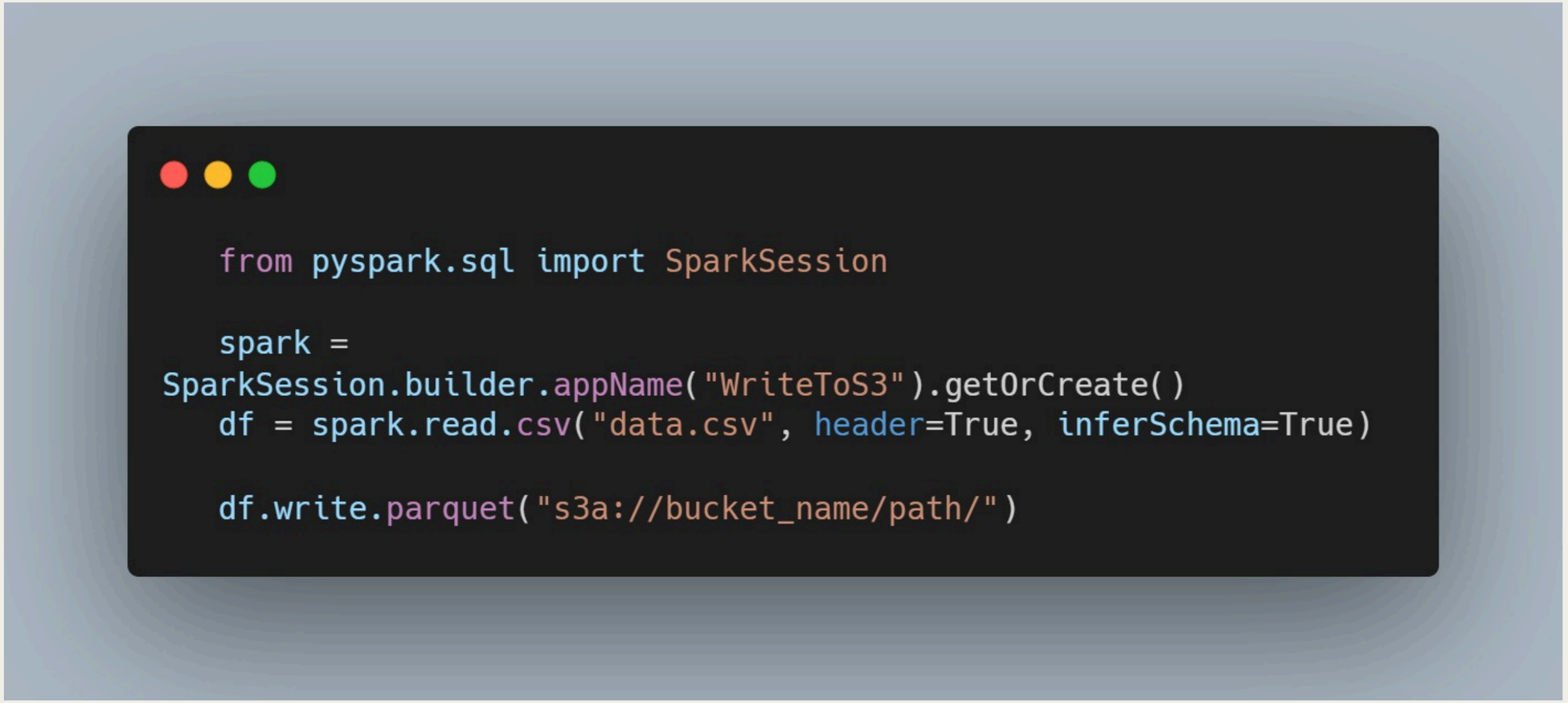

PYSPARK
CODING
QUESTIONS

Write a PySpark code to write a DataFrame to AWS S3 in Parquet format.

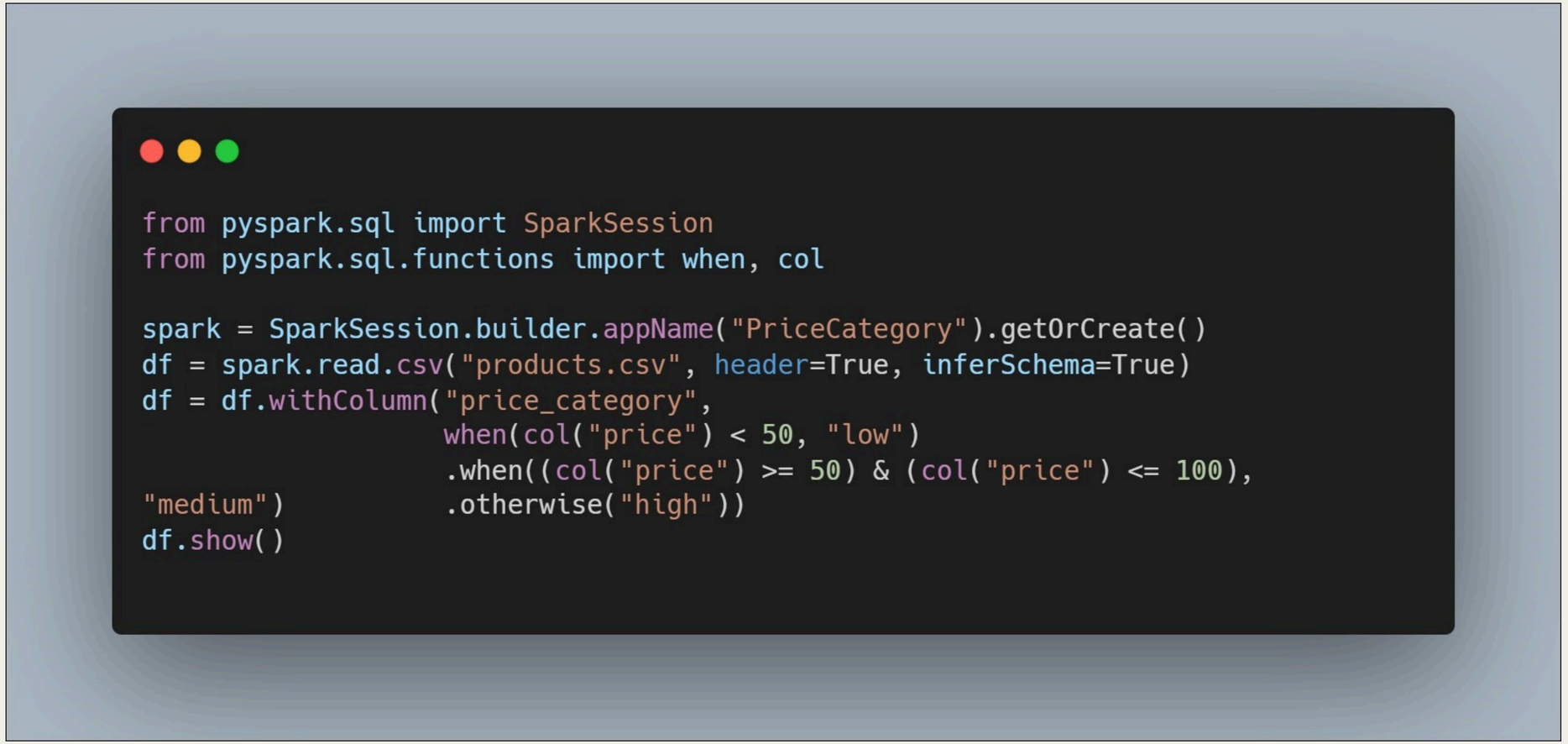


```
from pyspark.sql import SparkSession

spark =
SparkSession.builder.appName("WriteToS3").getOrCreate()
df = spark.read.csv("data.csv", header=True, inferSchema=True)

df.write.parquet("s3a://bucket_name/path/")
```

Write a PySpark code to add a new column price_category to a DataFrame based on the conditions for price.



```
from pyspark.sql import SparkSession
from pyspark.sql.functions import when, col

spark = SparkSession.builder.appName("PriceCategory").getOrCreate()
df = spark.read.csv("products.csv", header=True, inferSchema=True)
df = df.withColumn("price_category",
                   when(col("price") < 50, "low")
                   .when((col("price") >= 50) & (col("price") <= 100),
"medium")          .otherwise("high"))
df.show()
```

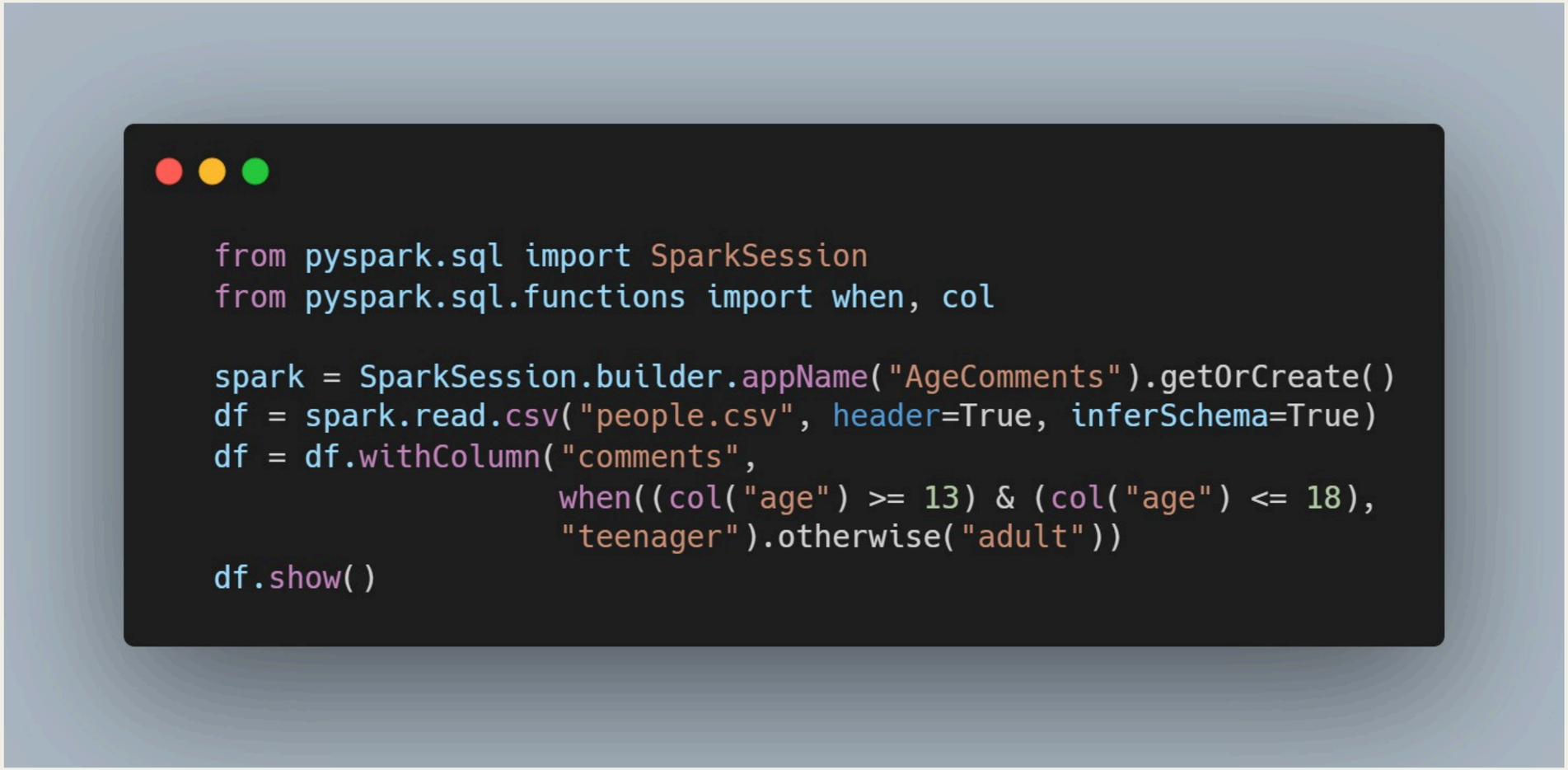
Write a PySpark code to read a DataFrame with a predefined schema without inferring it.

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType
spark = SparkSession.builder.appName("SchemaDefined").getOrCreate()

schema = StructType([
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("city", StringType(), True)
])

df = spark.read.schema(schema).csv("input.csv")
df.show()
```

Write a PySpark code to add a new column "comments" where if the age is between 13-18 years, the value is "teenager" and for other cases, the value is "adult".



```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import when, col

spark = SparkSession.builder.appName("AgeComments").getOrCreate()
df = spark.read.csv("people.csv", header=True, inferSchema=True)
df = df.withColumn("comments",
                   when((col("age") >= 13) & (col("age") <= 18),
                   "teenager").otherwise("adult"))
df.show()
```

Write a PySpark code to read the sales log, compute the total sales amount for each sale_type and date, aggregate the sales data by sale_type and date, and write the aggregated sales data to a separate table.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum

spark = SparkSession.builder.appName("AggregateSales").getOrCreate()
sales_df = spark.read.csv("sales_log.csv", header=True, inferSchema=True)
aggregated_df = sales_df.groupBy("sale_type", "date")\
                        .agg(sum("sales_amount").alias("total_sales"))
aggregated_df.write.mode("overwrite").saveAsTable("aggregated_sales")
```

Write a PySpark code to detect columns with JSON log entries, infer the schema, convert JSON columns to fields, and persist the updated DataFrame.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StructField, StringType, IntegerType

spark = SparkSession.builder.appName("JsonColumns").getOrCreate()

schema = StructType([
    StructField("field1", StringType(), True),
    StructField("field2", IntegerType(), True)
])

df = spark.read.json("logs.json")
df = df.withColumn("json_column", from_json(col("json_column"), schema))
updated_df = df.select("json_column.*", *[col(c) for c in df.columns if c != "json_column"])
updated_df.write.parquet("output_path")
```

Write a PySpark code to check for duplicates in primary keys (columns 1-10), filter out rows with null or empty primary keys, replace nulls in non-primary keys with 0, and reject rows where all values in columns 11 to 20 are null (non-primary keys).

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("DataProcessing").getOrCreate()
df = spark.read.csv("dataset.csv", header=True, inferSchema=True)

# 1. Drop duplicates based on primary keys (columns 1-10)
primary_key_columns = df.columns[:10]  # Assuming first 10 columns are primary keys
df = df.dropDuplicates(primary_key_columns)

# 2. Filter out rows where any primary key column is null or empty
for col_name in primary_key_columns:
    df = df.filter((df[col_name].isNotNull()) & (df[col_name] != ""))

# 3. Replace nulls in non-primary key columns (columns 11-20) with 0
non_primary_key_columns = df.columns[10:20]  # Assuming columns 11-20 are non-primary keys
df = df.fillna(0, subset=non_primary_key_columns)

# 4. Filter out rows where all non-primary key columns (columns 11-20) are null
from functools import reduce
condition = ~reduce(lambda x, y: x & y, [df[col_name].isNull() for col_name in
non_primary_key_columns])
df = df.filter(condition)

# 5. Display the resulting DataFrame
df.show()
```