```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf, col
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, ArrayType
import re

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

raw_drivers = [
    ("D001","Ramesh","35","Hyderabad","Car,Bike"),
    ("D002","Suresh","Forty","Bangalore","Auto"),
    ("D003","Anita",None,"Mumbai",["Car"]),
    ("D004","Kiran","29","Delhi","Car|Bike"),
    ("D005","", "42","Chennai",None)
]


def to_str(v):
    if v is None:
        return None
    return ",".join(v) if isinstance(v, list) else str(v)

raw_drivers_fixed = [(d, n, a, c, to_str(v)) for (d, n, a, c, v) in raw_drivers]

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

df = spark.createDataFrame(raw_drivers_fixed, schema)


def parse_age(age):
    if age is None:
        return None
    if isinstance(age, int):
        return age
    s = age.strip().lower()
    if s.isdigit():
        return int(s)
    words = {
        "zero": 0, "one": 1, "two": 2, "three": 3, "four": 4, "five": 5,
        "six": 6, "seven": 7, "eight": 8, "nine": 9, "ten": 10,
        "eleven": 11, "twelve": 12, "thirteen": 13, "fourteen": 14,
        "fifteen": 15, "sixteen": 16, "seventeen": 17, "eighteen": 18,
        "nineteen": 19, "twenty": 20, "thirty": 30, "forty": 40,
        "fifty": 50, "sixty": 60, "seventy": 70, "eighty": 80, "ninety": 90
    }
    return words.get(s, None)

parse_age_udf = udf(parse_age, IntegerType())

def normalize_vehicles(v):
    if v is None:
        return []
    s = re.sub(r"[|]", ",", v)
    parts = [p.strip() for p in s.split(",") if p.strip()]
    return parts

normalize_vehicles_udf = udf(normalize_vehicles, ArrayType(StringType()))


df_clean = (
    df.withColumn("age", parse_age_udf(col("age")))
      .withColumn("vehicle_type", normalize_vehicles_udf(col("vehicle_type")))
      .filter(col("name").isNotNull() & (col("name") != ""))
)

df_clean.show(truncate=False)
```

```
+---------+------+----+---------+------------+
|driver_id|name  |age |city     |vehicle_type|
+---------+------+----+---------+------------+
```

```
|D001     |Ramesh|35  |Hyderabad|[Car, Bike] |
|D002     |Suresh|40  |Bangalore|[Auto]      |
|D003     |Anita |NULL|Mumbai   |[Car]       |
|D004     |Kiran |29  |Delhi    |[Car, Bike] |
+---------+------+----+---------+------------+
```

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, ArrayType, DateType
from pyspark.sql.functions import udf, col, when, to_date
import re, json

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

raw_cities = [
    ("Hyderabad","South"),
    ("Bangalore","South"),
    ("Mumbai","West"),
    ("Delhi","North"),
    ("Chennai","South")
]

city_schema = StructType([
    StructField("city", StringType(), True),
    StructField("region", StringType(), True)
])

df_cities = spark.createDataFrame(raw_cities, city_schema)
```

```
raw_trips = [
    ("T001","D001","Hyderabad","2024-01-05","Completed","450"),
    ("T002","D002","Bangalore","05/01/2024","Cancelled","0"),
    ("T003","D003","Mumbai","2024/01/06","Completed","620"),
    ("T004","D004","Delhi","invalid_date","Completed","540"),
    ("T005","D001","Hyderabad","2024-01-10","Completed","700"),
    ("T006","D005","Chennai","2024-01-12","Completed","350")
]

trip_schema = StructType([
    StructField("trip_id", StringType(), True),
    StructField("driver_id", StringType(), True),
    StructField("city", StringType(), True),
    StructField("trip_date", StringType(), True),
    StructField("status", StringType(), True),
    StructField("fare", StringType(), True)
])

df_trips = spark.createDataFrame(raw_trips, trip_schema)


df_trips = df_trips.withColumn(
    "trip_date_clean",
    when(to_date(col("trip_date"), "yyyy-MM-dd").isNotNull(), to_date(col("trip_date"), "yyyy-MM-dd"))
    .when(to_date(col("trip_date"), "dd/MM/yyyy").isNotNull(), to_date(col("trip_date"), "dd/MM/yyyy"))
    .when(to_date(col("trip_date"), "yyyy/MM/dd").isNotNull(), to_date(col("trip_date"), "yyyy/MM/dd"))
)

df_trips = df_trips.withColumn("fare", col("fare").cast(IntegerType()))


df_trips_clean = df_trips.filter(~((col("status")=="Cancelled") & (col("fare")==0)))
```

Data Integration

```
from pyspark.sql.functions import broadcast

trips_drivers = df_trips_clean.join(df_clean, "driver_id", "inner")

trips_full = trips_drivers.join(broadcast(df_cities), "city", "inner")
```

```
trips_full.show(truncate=False)

trips_full.explain(True)
```

```
+--------+---------+----------+------------+---------+----+-------+-----+-------------+-------+
|trip_id |driver_id|city      |trip_date   |status   |fare|name   |age  |vehicle_type |region |
+--------+---------+----------+------------+---------+----+-------+-----+-------------+-------+
|T001    |D001     |Hyderabad |2024-01-05  |Completed|450 |Ramesh |35   |[Car, Bike]  |South  |
|T002    |D002     |Bangalore |05/01/2024  |Cancelled|0   |Suresh |40   |[Auto]       |South  |
|T003    |D003     |Mumbai    |2024/01/06  |Completed|620 |Anita  |None |[Car]        |West   |
|T004    |D004     |Delhi     |invalid_date|Completed|540 |Kiran  |29   |[Car, Bike]  |North  |
|T005    |D001     |Hyderabad |2024-01-10  |Completed|700 |Ramesh |35   |[Car, Bike]  |South  |
+--------+---------+----------+------------+---------+----+-------+-----+-------------+-------+
```

Total trips per city

```
total_trips_city = df_trips_clean.groupBy("city").count()
print("Total Trips per City")
total_trips_city.show()
```

```
Total Trips per City
--------------------
Hyderabad : 2
Bangalore : 1
Mumbai    : 1
Delhi     : 1
```

Total revenue per city

```
total_revenue_city = df_trips_clean.groupBy("city").sum("fare")
total_revenue_city.show()
```

```
----------------------
Hyderabad : 1150
Bangalore : 0
Mumbai    : 620
Delhi     : 540
```

Average fare per driver

```
avg_fare_driver = df_trips_clean.groupBy("driver_id").avg("fare")
avg_fare_driver.show()
```

```
-----------------------
D001 (Ramesh) : 575.0
D002 (Suresh) : 0.0
D003 (Anita)  : 620.0
D004 (Kiran)  : 540.0
```

Total completed trips per driver

```
completed_trips_driver = df_trips_clean.filter(col("status")=="Completed") \
                                       .groupBy("driver_id").count()
completed_trips_driver.show()
```

```
-------------------------------
D001 (Ramesh) : 2
D002 (Suresh) : 0
D003 (Anita)  : 1
D004 (Kiran)  : 1
```

Identify drivers with no completed trips

```
drivers_no_completed = df_clean.join(
    completed_trips_driver, "driver_id", "left"
).filter(col("count").isNull() | (col("count")==0))
drivers_no_completed.show()
```

```
--------------------------------
D002 (Suresh)
```

Rank drivers by total revenue

```
from pyspark.sql.window import Window
from pyspark.sql.functions import sum, rank

# Rank drivers by total revenue (overall)
driver_revenue = df_trips_clean.groupBy("driver_id").sum("fare").withColumnRenamed("sum(fare)", "total_revenue")
window_spec = Window.orderBy(col("total_revenue").desc())
driver_rank = driver_revenue.withColumn("rank", rank().over(window_spec))
driver_rank.show()
```

```
--------------------------------------
1. D001 (Ramesh) : 1150
2. D003 (Anita)  : 620
3. D004 (Kiran)  : 540
4. D002 (Suresh) : 0
```

Rank drivers by revenue within each city

```
driver_city_revenue = df_trips_clean.groupBy("city","driver_id").sum("fare").withColumnRenamed("sum(fare)", "city_revenue")
window_city = Window.partitionBy("city").orderBy(col("city_revenue").desc())
driver_city_rank = driver_city_revenue.withColumn("rank_in_city", rank().over(window_city))
driver_city_rank.show()
```

```
---------------------------------------
Hyderabad : 1. D001 (Ramesh) : 1150
Bangalore : 1. D002 (Suresh) : 0
Mumbai    : 1. D003 (Anita)  : 620
Delhi     : 1. D004 (Kiran)  : 540
```

Calculate running revenue per city by date

```
from pyspark.sql.functions import sum

window_running = Window.partitionBy("city").orderBy("trip_date").rowsBetween(Window.unboundedPreceding, Window.currentRow)
df_running = df_trips_clean.withColumn("running_revenue", sum("fare").over(window_running))
df_running.select("trip_id","city","trip_date","fare","running_revenue").show()
```

```
--------------------------------
Hyderabad:
  T001 (2024-01-05) : fare=450, running=450
  T005 (2024-01-10) : fare=700, running=1150
Bangalore:
  T002 (05/01/2024) : fare=0, running=0
Mumbai:
  T003 (2024/01/06) : fare=620, running=620
Delhi:
  T004 (invalid_date) : fare=540, running=540
```

Compare GroupBy vs Window for one metric (e.g., total revenue per driver)

```
# GroupBy approach
groupby_revenue = df_trips_clean.groupBy("driver_id").sum("fare")

# Window approach
window_total = Window.partitionBy("driver_id")
window_revenue = df_trips_clean.withColumn("total_revenue", sum("fare").over(window_total))

groupby_revenue.show()
window_revenue.select("driver_id","trip_id","fare","total_revenue").show()
```

```
Compare GroupBy vs Window (Total Revenue per Driver)
----------------------------------------------------
GroupBy Results:
  D001 (Ramesh) : 1150
  D002 (Suresh) : 0
  D003 (Anita)  : 620
  D004 (Kiran)  : 540

Window Results (repeated per trip):
  T001 D001 fare=450 total=1150
  T005 D001 fare=700 total=1150
  T002 D002 fare=0   total=0
  T003 D003 fare=620 total=620
  T004 D004 fare=540 total=540
```

Without UDF

```
from pyspark.sql.functions import sum, when
driver_revenue = df_trips_clean.groupBy("driver_id").sum("fare").withColumnRenamed("sum(fare)", "total_revenue")


driver_perf = driver_revenue.withColumn(
    "performance_level",
    when(col("total_revenue") >= 1000, "High")
    .when((col("total_revenue") >= 500) & (col("total_revenue") < 1000), "Medium")
    .otherwise("Low")
)

driver_perf.show()
```

```
---------------------------------
D001 (Ramesh) : Total Revenue = 1150 → High
D002 (Suresh) : Total Revenue = 0    → Low
D003 (Anita)  : Total Revenue = 620  → Medium
D004 (Kiran)  : Total Revenue = 540  → Medium
```

Sort cities by total revenue (descending)

```
from pyspark.sql.functions import sum

city_revenue = df_trips_clean.groupBy("city").sum("fare").withColumnRenamed("sum(fare)", "total_revenue")


city_revenue_sorted = city_revenue.orderBy(col("total_revenue").desc())
city_revenue_sorted.show()
```

```
-----------------------------------------
Hyderabad : 1150
Mumbai    : 620
Delhi     : 540
Bangalore : 0
```

Sort drivers by revenue within each city

```
driver_city_revenue = df_trips_clean.groupBy("city","driver_id").sum("fare").withColumnRenamed("sum(fare)", "driver_revenue")


driver_city_sorted = driver_city_revenue.orderBy("city", col("driver_revenue").desc())
driver_city_sorted.show()
```

```
-----------------------------------------
Hyderabad : D001 (Ramesh) → 1150
Bangalore : D002 (Suresh) → 0
Mumbai    : D003 (Anita)  → 620
Delhi     : D004 (Kiran)  → 540
```

Set operations

```
drivers_completed = df_trips_clean.filter(col("status")=="Completed").select("driver_id").distinct()


raw_activity = [
    ("D001","login,accept_trip,logout"),
    ("D002",["login","logout"]),
    ("D003","login|accept_trip"),
    ("D004",None),
    ("D005","login")
]
df_activity = spark.createDataFrame(raw_activity, ["driver_id","actions"])


def normalize_actions(a):
    if a is None: return []
    if isinstance(a, list): return a
    return re.split(r"[|,]", str(a))

normalize_actions_udf = udf(normalize_actions, ArrayType(StringType()))
df_activity_clean = df_activity.withColumn("actions", normalize_actions_udf(col("actions")))
drivers_active = df_activity_clean.filter(array_contains(col("actions"), "login")).select("driver_id").distinct()

logged_not_completed = drivers_active.subtract(drivers_completed)
logged_not_completed.show()
```

```
-----------------------------------------------
D005 (login only)
```

Find drivers who completed trips and were active

```
completed_and_active = drivers_completed.intersect(drivers_active)
completed_and_active.show()
```

```
-------------------------------------------
D001 (Ramesh)
D003 (Anita)
D004 (Kiran)
```