A person sitting at a desk

AI-generated content may be incorrect.

A person with his hands on his hips

AI-generated content may be incorrect.

Practice Apache Spark

#To select the columns that are required to study

flights\_column\_needed\_df = flights.df.select(“col1”,”col2”,”col3”,”col4”)

#this is wrong. filter needs a conditions not just a column name.

flights\_column\_needed\_df = flights.df.filter(“col1”,”col2”,”col3,”col4”)

display(flights\_column\_needed\_df).limit(10)

#count the number of rows in the datatable

count\_data\_rows = flight\_column\_needed\_df.count()

#createOrReplaceTempView is only to create a sql view so you can use SQL queries later

new\_dataframe\_df = flights.df.selectExp(“"Year",

        "Month",

        "DayofMonth",

        "CAST(DepTime AS INT) AS DepTime",

        "FlightNum",

        "CAST(ActualElapsedTime AS INT) AS ActualElapsedTime",

        "CRSElapsedTime",

        "CAST(ArrDelay AS INT) AS ArrDelay"

    ) \

    .createOrReplaceTempView("flights\_temp")

#sprk.sql

Below why do we put 3 quotations marks.

invalid\_counts\_sql = spark.sql(“”” select count\_if(year IS NULL) as null\_year,

count\_if(Month IS NULL) as null\_month,

count\_if(dayofyear IS NULL) as null\_dayofyear,

COUNT\_IF(DepTime IS NULL) AS Null\_DepTime\_Count,

    COUNT\_IF(FlightNum IS NULL) AS Null\_FlightNum\_Count,

    COUNT\_IF(ActualElapsedTime IS NULL) AS Null\_ActualElapsedTime\_Count,

    COUNT\_IF(CRSElapsedTime IS NULL) AS Null\_CRSElapsedTime\_Count,

    COUNT\_IF(ArrDelay IS NULL) AS Null\_ArrDelay\_Count

FROM flights\_temp

""")

#convert temp view dataframe to spark table

flights\_temp\_df = spark.table(“flights\_temp”)

#so looks like below will sum the amount of “1”s we get.

invalid\_counts\_df = flights\_temp\_df.select(sum(when(col(“year”).isNull(),1.otherwise(0).alias(“Null\_year\_count”),

sum(when(col(“month”).isNull(),1.otherwise(0).alias(“Null\_month\_count”))

display(invalid\_counts\_df)

#drop rows whereever we have nulls. Looks like w can select one columns and drop those rows

valid\_rows\_df = flghts\_df.na.drop(how = ‘any’, subset = [‘CRSElapsedTime’])

#les use filter to drop the rows we don’t need

valid\_rows\_filter\_df = flights.df.filter(col(“col1”).cast(“integer”).isNotNull() & col("ActualElapsedTime").cast("integer").isNotNull() &

    col("DepTime").cast("integer").isNotNull()

)

why are we renaming the columns below to same name and casting as integer even if we have done that above?

flights\_data\_required = valid\_rows\_filter\_df.withColumn(“ArrDelay”,col(“ArrDelay).cast(“integer”)). withColumn("ActualElapsedTime", col("ActualElapsedTime").cast("integer"))

clean\_flights\_df.printSchema()

#make time stamp no time zone

from pyspark.sql.functions import col, make\_timestamp\_ntz, lpad, substr

flight\_with\_datetime\_df = clean\_flights\_df.withColumn(“FlightDateTime”, col(“year”),col(“month”),lpad(col(“deptime”),3,0)

Now here’s the part for extracting **hour and minute from DepTime**:

python

CopyEdit

substr(lpad(col("DepTime"), 4, "0"), lit(1), lit(2)).cast("integer"), # Hour

substr(lpad(col("DepTime"), 4, "0"), lit(3), lit(2)).cast("integer"), # Minute

* lpad(col("DepTime"), 4, "0"): Ensures values like 700 become "0700" (pads on left with zeroes)
* substr(..., 1, 2): Takes "07" → **Hour**
* substr(..., 3, 2): Takes "00" → **Minute**

**Homework: Build a Cleaned & Enriched Flight Dataset**

**Objective**: You are given a raw flights dataset (like flights\_df). Your task is to clean it, create new time-based columns, and calculate insights such as normalized elapsed time difference.

flights\_df = spark.read.table("dbacademy\_airline.v01.flights\_small")

drop rows with nulls

flights\_without\_nulls\_df = flights\_df.na.drop(col(“DepTime”) isNull & col(“DepTime”) isNull & col(“CRSDeptime”) isNull & col(ArrTime”) isNull & col(“CRSArrTime”) isNull)

#Na needs a subset of column names not “col”

flights\_without\_nulls\_df = flights\_df.na.drop(subset = [“DepTime”, “CRSDeptime”, “ArrTime”, CRSArrTime”])

display(flights\_without\_nulls\_df)

#createOrReplaceTempView is only to create a sql view so you can use SQL queries later

new\_dataframe\_df = flights.df.selectExp(“"Year",

        "Month",

        "DayofMonth",

        "CAST(DepTime AS INT) AS DepTime",

        "FlightNum",

        "CAST(ActualElapsedTime AS INT) AS ActualElapsedTime",

        "CRSElapsedTime",

        "CAST(ArrDelay AS INT) AS ArrDelay"

    ) \

    .createOrReplaceTempView("flights\_temp")

from pyspark.sql.functions import col, when, sum

# Count nulls in DepTime column – This was a night mare

flights\_nulls\_per\_column\_df = flights\_new\_df.select(

sum(when(col("DepTime").isNull(), 1).otherwise(0)).alias("DepTime\_nulls")

)

display(flights\_nulls\_per\_column\_df)

time\_calculation\_df = flights\_without\_nulls\_df.withColumn(“DepTimeDelay”,col(“DepTime)-col(“CRSDeptime”).withColumn(“ArriveTimeDelay”,col(“ArrTime”) – col(“CRSArrTime”))

#some syntax and withColumn collections are wrong aswell as brackets

time\_calculations\_df = flights\_without\_nulls\_df.withColumn(“DepTimeDelay”,col(“DepTime)-col(“CRSDeptime”)).withColumn(“ArriveTimeDelay”,col(“ArrTime”) – col(“CRSArrTime”))

labeling\_df = time\_calculation\_df.withColumn(“DepTime\_Delays”.when(“DepTimeDelay” >= 15, “Slight Delay”),when(“DepTimeDelay” >15 but <= 30,”Moderate Delay”), when(“DepTimeDelay” >30,”Severe Delay”))

#when cannot be allowed on column name directly, it needs to use Col. And when is like if statement you need to close it later.

from pyspark.sql.functions import when

labelling\_df = time\_calculation\_df.withColumn(“DeptDelays”, when(col(“DepTimeDelay”) > 30, “Severe Delay”).when(col(“DepTimeDelay”) >= 15) & col(“DepTimeDelay”) < 30, “Moderate Delay”). when(col(“DepTimeDelay”) >= 1) & col(“DepTimeDelay”) < 15, “Slight Delay”).otherwise “OnTime”)

pandas is excel + sql. So Import pandas\_udf. Define a functions which needs a bracket expression which is bait. Here you can give whatever column you want.

from pyspark.sql.functions import pandas\_udf

from pyspark.sql.functions import DoubleType

@pandas\_udf(DoubleType())

def normalized\_time(diff\_series):

return (diff\_series - diff\_series.mean())/diff\_series.std()

udf\_formula = time\_calculation\_df.withColumn(“normalized\_time”, normalized\_time(“DepTimeDelay”))