databricks 12_Mo_Split_Impute_Nulls_12_03_2023

(https://databricks.com)

PLEASE CLONE THIS NOTEBOOK INTO YOUR PERSONAL FOLDER

DO NOT RUN CODE IN THE SHARED FOLDER

THERE IS A 2 POINT DEDUCTION IF YOU RUN ANYTHING IN THE SHARED FOLDER. THANKS!

```
from pyspark.sql.functions import isnan, when, count, col, split, trim, lit, avg, max, sum, desc, lpad, concat, to_timestar
month, lag
from pyspark.sql import Window, DataFrame
from pyspark.sql.types import TimestampType
from datetime import timedelta
print("Welcome to the W261 final project!")
```

```
Welcome to the W261 final project!
```

Know your mount

Here is the mounting for this class, your source for the original data! Remember, you only have Read access, not Write! Also, become familiar with `dbutils` the equivalent of `gcp` in DataProc

```
# data_BASE_DIR = "dbfs:/mnt/mids-w261/"
# display(dbutils.fs.ls(f"{data_BASE_DIR}"))
```

```
#dbutils.fs.help()
```

Data for the Project

For the project you will have 4 sources of data:

1. Airlines Data: This is the raw data of flights information. You have 3 months, 6 months, 1 year, and full data from 2015 to 2019. Remember the maxima: "Test, Test, Test", so a lot of testing in smaller samples before scaling up! Location of the data? `dbfs:/mnt/mids-w261/datasets_final_project_2022/parquet_airlines_data/`, `dbfs:/mnt/mids-w261/datasets_final_project_2022/parquet_airlines_data_1y/`, etc. (Below the dbutils to get the folders)
2. Weather Data: Raw data for weather information. Same as before, we are sharing 3 months, 6 months, 1 year
3. Stations data: Extra information of the location of the different weather stations. Location `dbfs:/mnt/mids-w261/datasets_final_project_2022/stations_data/stations_with_neighbors.parquet/`
4. OTPW Data: This is our joined data (We joined Airlines and Weather). This is the main dataset for your project, the previous 3 are given for reference. You can attempt your own join for Extra Credit. Location `dbfs:/mnt/mids-w261/OTPW_60M/` and more, several samples are given!

```
# # Airline Data
# df_flights = spark.read.parquet(f"dbfs:/mnt/mids-w261/datasets_final_project_2022/parquet_airlines_data_3m/")
# #display(df_flights)
```

```
# # Weather data
# df_weather = spark.read.parquet(f"dbfs:/mnt/mids-w261/datasets_final_project_2022/parquet_weather_data_3m/")
# #display(df_weather)
```

```
# # Stations data
# df_stations = spark.read.parquet(f"dbfs:/mnt/mids-
w261/datasets_final_project_2022/stations_data/stations_with_neighbors.parquet/")
# #display(df_stations)
```

```
# # OTPW
# df_otpw = spark.read.format("csv").option("header","true").load(f"dbfs:/mnt/mids-w261/OTPW_12M")
# #display(df_otpw)
```

Beginning of data cleansing and one hot encoding (categorical data)

```
# ######################################################
# # import libraries
# ######################################################
# from pyspark.ml.feature import StringIndexer,OneHotEncoder
# from pyspark.sql.functions import col, regexp_replace, when, expr, desc, lag, lead
# from pyspark.sql.window import Window
```

```
# ######################################################
# # create a temp table tb_otpw to leverage spark sql.
# # Extracted needed model columns from OTPW.
# # create a new data frame df_model from needed cols.
# ######################################################

# df_otpw.createOrReplaceTempView("tb_otpw")

# df_model = spark.sql('''SELECT
#             QUARTER,
#             DAY_OF_MONTH,
#             DAY_OF_WEEK,
#             FL_DATE,
#             OP_UNIQUE_CARRIER,
#             TAIL_NUM,
#             ORIGIN_AIRPORT_ID,
#             ORIGIN_CITY_MARKET_ID,
#             ORIGIN_STATE_ABR,
#             ORIGIN_WAC,
#             DEST_AIRPORT_ID,
#             DEST_CITY_MARKET_ID,
#             DEST_STATE_ABR,
#             DEST_WAC,
#             CRS_DEP_TIME,
#             DEP_DEL15,
#             DEP_TIME_BLK,
#             FLIGHTS,
#             YEAR,
#             MONTH,
#             ELEVATION,
#             HourlyAltimeterSetting,
#             HourlyDewPointTemperature,
#             HourlyDryBulbTemperature,
#             HourlyPrecipitation,
#             HourlyRelativeHumidity,
#             HourlySkyConditions,
#             HourlySeaLevelPressure,
#             HourlyStationPressure,
#             HourlyVisibility,
#             HourlyWetBulbTemperature,
#             HourlyWindDirection,
#             HourlyWindSpeed
#  FROM tb_otpw''')
```

```
# original data types for attributes in df_model before conversion
df_model.dtypes
df_model.schema
```

```
NameError: name 'df_model' is not defined
```

```
# ######################################################
# # create a temp table tb_model from df_model.
# ######################################################
# df_model.createOrReplaceTempView("tb_model")
```

```
# ######################################################################
# # Write data imputation rules for non-categorical data.
# # For categorical attributes implement StringIndexer package.
# # Store the final data in a new dataframe indexed_df
# ######################################################################
# df_model_cat = spark.sql('''SELECT

#         --  Attributes for data formatting/casting/imputation
#             CASE
#             WHEN HourlyAltimeterSetting LIKE '%*%' THEN NULL
#             WHEN HourlyAltimeterSetting RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyAltimeterSetting, 's',
'') AS DECIMAL(10, 2))
#             WHEN HourlyAltimeterSetting RLIKE '[^-?\\d+$]' THEN CAST(HourlyAltimeterSetting AS DECIMAL(10, 2))
#             ELSE NULL
#             END
#             AS HourlyAltimeterSetting,

#             CASE
#             WHEN HourlyDewPointTemperature LIKE '%*%' THEN NULL
#             WHEN HourlyDewPointTemperature RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyDewPointTemperature,
's', '') AS INT)
#             WHEN HourlyDewPointTemperature RLIKE '[^-?\\d+$]' THEN CAST(HourlyDewPointTemperature AS INT)
#             ELSE NULL
#             END
#             AS HourlyDewPointTemperature,

#             CASE
#             WHEN HourlyDryBulbTemperature LIKE '%*%' THEN NULL
#             WHEN HourlyDryBulbTemperature RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyDryBulbTemperature,
's', '') AS INT)
#             WHEN HourlyDryBulbTemperature RLIKE '[^-?\\d+$]' THEN CAST(HourlyDryBulbTemperature AS INT)
#             ELSE NULL
#             END
#             AS HourlyDryBulbTemperature,

#             CASE
#             WHEN HourlyPrecipitation LIKE '%*%' THEN NULL
#             WHEN HourlyPrecipitation RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyPrecipitation, 's', '') AS
DECIMAL(10, 2))
#             WHEN HourlyPrecipitation RLIKE '[^-?\\d+$]' THEN CAST(HourlyPrecipitation AS DECIMAL(10, 2))
#             ELSE NULL
#             END
#             AS HourlyPrecipitation,

#             CASE
#             WHEN HourlyRelativeHumidity LIKE '%*%' THEN NULL
#             WHEN HourlyRelativeHumidity RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyRelativeHumidity, 's',
'') AS INT)
#             WHEN HourlyRelativeHumidity RLIKE '[^-?\\d+$]' THEN CAST(HourlyRelativeHumidity AS INT)
#             ELSE NULL
#             END
#             AS HourlyRelativeHumidity,

#             CASE
#             WHEN HourlySeaLevelPressure LIKE '%*%' THEN NULL
#             WHEN HourlySeaLevelPressure RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlySeaLevelPressure, 's',
'') AS  DECIMAL(10, 2))
#             WHEN HourlySeaLevelPressure RLIKE '[^-?\\d+$]' THEN CAST(HourlySeaLevelPressure  AS DECIMAL(10,2))
#             ELSE NULL
#             END
#             AS HourlySeaLevelPressure,

#             CASE
#             WHEN HourlyStationPressure LIKE '%*%' THEN NULL
#             WHEN HourlyStationPressure RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyStationPressure, 's', '')
AS  DECIMAL(10, 2))
#             WHEN HourlyStationPressure RLIKE '[^-?\\d+$]' THEN CAST(HourlyStationPressure  AS DECIMAL(10,2))
```

```
#             ELSE NULL
#             END
#             AS HourlyStationPressure,

#             CASE
#             WHEN HourlyVisibility LIKE '%*%' THEN NULL
#             WHEN HourlyVisibility RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyVisibility, 's', '') AS
DECIMAL(10, 2))
#             WHEN HourlyVisibility RLIKE '[^-?\\d+$]' THEN CAST(HourlyVisibility  AS DECIMAL(10,2))
#             ELSE NULL
#             END
#             AS HourlyVisibility,

#             CASE
#             WHEN HourlyWetBulbTemperature LIKE '%*%' THEN NULL
#             WHEN HourlyWetBulbTemperature RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyWetBulbTemperature,
's', '') AS  INT)
#             WHEN HourlyWetBulbTemperature RLIKE '[^-?\\d+$]' THEN CAST(HourlyWetBulbTemperature  AS INT)
#             ELSE NULL
#             END
#             AS HourlyWetBulbTemperature,

#             CASE
#             WHEN HourlyWindDirection LIKE '%*%' THEN NULL
#             WHEN HourlyWindDirection RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyWindDirection, 's', '') AS
INT)
#             WHEN HourlyWindDirection RLIKE '[^-?\\d+$]' THEN CAST(HourlyWindDirection  AS INT)
#             ELSE NULL
#             END
#             AS HourlyWindDirection,

#             CASE
#             WHEN HourlyWindSpeed LIKE '%*%' THEN NULL
#             WHEN HourlyWindSpeed RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyWindSpeed, 's', '') AS  INT)
#             WHEN HourlyWindSpeed RLIKE '[^-?\\d+$]' THEN CAST(HourlyWindSpeed  AS INT)
#             ELSE NULL
#             END
#             AS HourlyWindSpeed,

#             CASE
#             WHEN Flights LIKE '%*%' THEN NULL
#             WHEN Flights RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(Flights, 's', '') AS DECIMAL(10,1))
#             WHEN Flights RLIKE '[^-?\\d+$]' THEN CAST(Flights  AS DECIMAL(10,1))
#             ELSE NULL
#             END
#             AS Flights,

#             CASE
#             WHEN Year LIKE '%*%' THEN NULL
#             WHEN Year RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(Year, 's', '') AS INT)
#             WHEN Year RLIKE '[^-?\\d+$]' THEN CAST(Year  AS INT)
#             ELSE NULL
#             END
#             AS Year,

#             CASE
#             WHEN CRS_DEP_TIME LIKE '%*%' THEN NULL
#             WHEN CRS_DEP_TIME RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(CRS_DEP_TIME, 's', '') AS INT)
#             WHEN CRS_DEP_TIME RLIKE '[^-?\\d+$]' THEN CAST(CRS_DEP_TIME  AS INT)
#             ELSE NULL
#             END
#             AS CRS_DEP_TIME,

#             CASE
#             WHEN FL_DATE LIKE '%*%' THEN NULL
#             WHEN FL_DATE RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(FL_DATE, 's', '') AS DATE)
#             WHEN FL_DATE RLIKE '[^-?\\d+$]' THEN CAST(CRS_DEP_TIME  AS DATE)
```

```
#             ELSE NULL
#             END
#             AS FL_DATE,

#             CASE
#             WHEN Elevation LIKE '%*%' THEN NULL
#             WHEN Elevation RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(Elevation, 's', '') AS  DECIMAL(10, 1))
#             WHEN HourlyVisibility RLIKE '[^-?\\d+$]' THEN CAST(Elevation  AS DECIMAL(10,1))
#             ELSE NULL
#             END
#             AS Elevation,

#         -- Leaving this attribute as string for now
#             HourlySkyConditions AS HourlySkyConditions,

#         -- Attributes for catergorical (one-hot encoding set)
#             QUARTER,
#             DAY_OF_MONTH,
#             CASE WHEN DAY_OF_MONTH <= 15 THEN "1stHalf"
#             WHEN DAY_OF_MONTH >15 THEN "2ndHalf"
#             ELSE NULL
#             END
#             AS DAY_OF_MONTH_Cat,
#             DAY_OF_WEEK,
#             OP_UNIQUE_CARRIER,
#             TAIL_NUM,
#             ORIGIN_AIRPORT_ID,
#             ORIGIN_CITY_MARKET_ID,
#             ORIGIN_STATE_ABR,
#             ORIGIN_WAC,
#             DEST_AIRPORT_ID,
#             DEST_CITY_MARKET_ID,
#             DEST_STATE_ABR,
#             DEST_WAC,
#             DEP_DEL15,
#             DEP_TIME_BLK,
#             MONTH
#             -- ,Elevation
#              from tb_model''') ##.display(10)

# ## manual casting of quarter as current dataset has only 1 quarter data failing with error during StringIndexer
step
# df_model_cat=df_model_cat.withColumn("QUARTER", col("QUARTER").cast("int"))

# indexer = StringIndexer(inputCols=
["DAY_OF_MONTH_Cat","DAY_OF_WEEK","OP_UNIQUE_CARRIER","TAIL_NUM","ORIGIN_AIRPORT_ID",
#                                   "ORIGIN_CITY_MARKET_ID","ORIGIN_STATE_ABR","ORIGIN_WAC","DEST_AIRPORT_ID",
#
"DEST_CITY_MARKET_ID","DEST_STATE_ABR","DEST_WAC","DEP_DEL15","DEP_TIME_BLK","MONTH"],
#                       outputCols=
["DAY_OF_MONTH_ind","DAY_OF_WEEK_ind","OP_UNIQUE_CARRIER_ind","TAIL_NUM_ind","ORIGIN_AIRPORT_ID_ind",
#
"ORIGIN_CITY_MARKET_ID_ind","ORIGIN_STATE_ABR_ind","ORIGIN_WAC_ind","DEST_AIRPORT_ID_ind",
#
"DEST_CITY_MARKET_ID_ind","DEST_STATE_ABR_ind","DEST_WAC_ind","DEP_DEL15_ind","DEP_TIME_BLK_ind","MONTH_ind"])

# ## Handling of nulls by keeping them during string indexer conversion
# ## https://stackoverflow.com/questions/36112684/handling-null-values-in-spark-stringindexer
# indexer=indexer.setHandleInvalid("keep")

# indexerModel = indexer.fit(df_model_cat)

# # Transform the DataFrame using the fitted StringIndexer model
# indexed_df = indexerModel.transform(df_model_cat)

# indexed_df.show(1)
```

```
## NOTE:unable to apply string indexer since quarter has only 1 category for this dataset (revisit bigger dataset)
#indexed_df.select("QUARTER").distinct().show(1)
```

```
# ######################################################################
# # Generate 1 hot encoding on indexed_df dataframe for all the
# # categorical attributes
# ######################################################################

# # df_test2 = df_model_cat.withColumn("QUARTER", col("QUARTER").cast("int"))\
# #                         .withColumn("DAY_OF_MONTH", col("DAY_OF_MONTH").cast("int"))\
# #                         .withColumn("DAY_OF_WEEK", col("DAY_OF_WEEK").cast("int"))\
# #                             .withColumn("DAY_OF_WEEK", col("DAY_OF_WEEK").cast("int"))\
# ##df_test2 = indexed_df.select("QUARTER","DAY_OF_MONTH_Ind")
# #df_model_vec = indexed_df

# encoder = OneHotEncoder(inputCols=["QUARTER","DAY_OF_MONTH_ind","DAY_OF_WEEK_ind","OP_UNIQUE_CARRIER_ind",
#                                    "TAIL_NUM_ind","ORIGIN_AIRPORT_ID_ind",
#
"ORIGIN_CITY_MARKET_ID_ind","ORIGIN_STATE_ABR_ind","ORIGIN_WAC_ind","DEST_AIRPORT_ID_ind",
#
"DEST_CITY_MARKET_ID_ind","DEST_STATE_ABR_ind","DEST_WAC_ind","DEP_DEL15_ind","DEP_TIME_BLK_ind","MONTH_ind"
#                                    ],
#                         outputCols=["QUARTER_Vec","DAY_OF_MONTH_Vec","DAY_OF_WEEK_Vec","OP_UNIQUE_CARRIER_Vec",
#                                    "TAIL_NUM_Vec","ORIGIN_AIRPORT_ID_Vec",
#
"ORIGIN_CITY_MARKET_ID_Vec","ORIGIN_STATE_ABR_Vec","ORIGIN_WAC_Vec","DEST_AIRPORT_ID_Vec",
#
"DEST_CITY_MARKET_ID_Vec","DEST_STATE_ABR_Vec","DEST_WAC_Vec","DEP_DEL15_Vec","DEP_TIME_BLK_Vec","MONTH_Vec"
#                                    ])
# model_cat = encoder.fit(indexed_df)
# df_encoded_cat = model_cat.transform(indexed_df)
# df_encoded_cat.show(1)
```

```
#df_encoded_cat.show(2)
#type(df_encoded_cat)
```

Ignore starter code for sample/unit testing

```
####################################################
# create a temp table tb_model from df_model
####################################################
## HourlyAltimeterSetting

# spark.sql('''SELECT
#             distinct HourlyAltimeterSetting,
#             CAST(substr(HourlyAltimeterSetting,1,5) AS DECIMAL(10, 2)) AS extracted_numeric_value
#              from tb_model''').display()


# spark.sql('''SELECT
#             distinct HourlyAltimeterSetting,
#             CASE
#             WHEN HourlyAltimeterSetting LIKE '%*%' THEN NULL
#             WHEN HourlyAltimeterSetting RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyAltimeterSetting, 's',
'') AS DECIMAL(10, 2))
#             WHEN HourlyAltimeterSetting RLIKE '[^-?\\d+$]' THEN CAST(HourlyAltimeterSetting AS DECIMAL(10, 2))
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
## HourlyDewPointTemperature

# spark.sql('''SELECT

#             distinct HourlyDewPointTemperature,
#             CASE
#             WHEN HourlyDewPointTemperature LIKE '%*%' THEN NULL
#             WHEN HourlyDewPointTemperature RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyDewPointTemperature,
's', '') AS INT)
#             WHEN HourlyDewPointTemperature RLIKE '[^-?\\d+$]' THEN CAST(HourlyDewPointTemperature AS INT)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
## HourlyDryBulbTemperature

# spark.sql('''SELECT
#             distinct HourlyDryBulbTemperature,
#             CASE
#             WHEN HourlyDryBulbTemperature LIKE '%*%' THEN NULL
#             WHEN HourlyDryBulbTemperature RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyDryBulbTemperature,
's', '') AS INT)
#             WHEN HourlyDryBulbTemperature RLIKE '[^-?\\d+$]' THEN CAST(HourlyDryBulbTemperature AS INT)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
## HourlyPrecipitation

# spark.sql('''SELECT
#             distinct HourlyPrecipitation,
#             CASE
#             WHEN HourlyPrecipitation LIKE '%*%' THEN NULL
#             WHEN HourlyPrecipitation RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyPrecipitation, 's', '') AS
DECIMAL(10, 2))
#             WHEN HourlyPrecipitation RLIKE '[^-?\\d+$]' THEN CAST(HourlyPrecipitation AS DECIMAL(10, 2))
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display(10)
```

```
## HourlyRelativeHumidity

# spark.sql('''SELECT
#             distinct HourlyRelativeHumidity,
#             CASE
#             WHEN HourlyRelativeHumidity LIKE '%*%' THEN NULL
#             WHEN HourlyRelativeHumidity RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyRelativeHumidity, 's',
'') AS INT)
#             WHEN HourlyRelativeHumidity RLIKE '[^-?\\d+$]' THEN CAST(HourlyRelativeHumidity AS INT)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display(10)
```

```
## HourlySkyConditions


# spark.sql('''SELECT
#             distinct HourlySkyConditions
#              from tb_model''').display(10)
```

```
# HourlySeaLevelPressure
# spark.sql('''SELECT
#             distinct HourlySeaLevelPressure,
#             CASE
#             WHEN HourlySeaLevelPressure LIKE '%*%' THEN NULL
#             WHEN HourlySeaLevelPressure RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlySeaLevelPressure, 's',
'') AS  DECIMAL(10, 2))
#             WHEN HourlySeaLevelPressure RLIKE '[^-?\\d+$]' THEN CAST(HourlySeaLevelPressure  AS DECIMAL(10,2))
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display(10)
```

```
# HourlyStationPressure
# spark.sql('''SELECT
#             distinct HourlyStationPressure,
#             CASE
#             WHEN HourlyStationPressure LIKE '%*%' THEN NULL
#             WHEN HourlyStationPressure RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyStationPressure, 's', '')
AS  DECIMAL(10, 2))
#             WHEN HourlyStationPressure RLIKE '[^-?\\d+$]' THEN CAST(HourlyStationPressure  AS DECIMAL(10,2))
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display(10)
```

```
# HourlyVisibility
# spark.sql('''SELECT
#             distinct HourlyVisibility,
#             CASE
#             WHEN HourlyVisibility LIKE '%*%' THEN NULL
#             WHEN HourlyVisibility RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyVisibility, 's', '') AS
DECIMAL(10, 2))
#             WHEN HourlyVisibility RLIKE '[^-?\\d+$]' THEN CAST(HourlyVisibility  AS DECIMAL(10,2))
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
# HourlyWetBulbTemperature
# spark.sql('''SELECT
#             distinct HourlyWetBulbTemperature,
#             CASE
#             WHEN HourlyWetBulbTemperature LIKE '%*%' THEN NULL
#             WHEN HourlyWetBulbTemperature RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyWetBulbTemperature,
's', '') AS  INT)
#             WHEN HourlyWetBulbTemperature RLIKE '[^-?\\d+$]' THEN CAST(HourlyWetBulbTemperature  AS INT)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
# HourlyWindDirection
# spark.sql('''SELECT
#             distinct HourlyWindDirection,
#             CASE
#             WHEN HourlyWindDirection LIKE '%*%' THEN NULL
#             WHEN HourlyWindDirection RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyWindDirection, 's', '') AS
INT)
#             WHEN HourlyWindDirection RLIKE '[^-?\\d+$]' THEN CAST(HourlyWindDirection  AS INT)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
# HourlyWindSpeed
# spark.sql('''SELECT
#             distinct HourlyWindSpeed,
#             CASE
#             WHEN HourlyWindSpeed LIKE '%*%' THEN NULL
#             WHEN HourlyWindSpeed RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(HourlyWindSpeed, 's', '') AS  INT)
#             WHEN HourlyWindSpeed RLIKE '[^-?\\d+$]' THEN CAST(HourlyWindSpeed  AS INT)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
# Flights
# spark.sql('''SELECT
#             distinct Flights,
#             CASE
#             WHEN Flights LIKE '%*%' THEN NULL
#             WHEN Flights RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(Flights, 's', '') AS DECIMAL(10,1))
#             WHEN Flights RLIKE '[^-?\\d+$]' THEN CAST(Flights  AS DECIMAL(10,1))
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
# Year
# spark.sql('''SELECT
#             distinct Year,
#             CASE
#             WHEN Year LIKE '%*%' THEN NULL
#             WHEN Year RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(Year, 's', '') AS INT)
#             WHEN Year RLIKE '[^-?\\d+$]' THEN CAST(Year  AS INT)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
## CRS_DEP_TIME

# spark.sql('''SELECT
#             distinct CRS_DEP_TIME,
#             CASE
#             WHEN CRS_DEP_TIME LIKE '%*%' THEN NULL
#             WHEN CRS_DEP_TIME RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(CRS_DEP_TIME, 's', '') AS INT)
#             WHEN CRS_DEP_TIME RLIKE '[^-?\\d+$]' THEN CAST(CRS_DEP_TIME  AS INT)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
## FL_DATE

# spark.sql('''SELECT
#             distinct FL_DATE,
#             CASE
#             WHEN FL_DATE LIKE '%*%' THEN NULL
#             WHEN FL_DATE RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(FL_DATE, 's', '') AS DATE)
#             WHEN FL_DATE RLIKE '[^-?\\d+$]' THEN CAST(CRS_DEP_TIME  AS DATE)
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
# Elevation
# spark.sql('''SELECT
#             distinct Elevation,
#             CASE
#             WHEN Elevation LIKE '%*%' THEN NULL
#             WHEN Elevation RLIKE '[^-?\\d+s$]' THEN CAST(REGEXP_REPLACE(Elevation, 's', '') AS  DECIMAL(10, 1))
#             WHEN HourlyVisibility RLIKE '[^-?\\d+$]' THEN CAST(Elevation  AS DECIMAL(10,1))
#             ELSE NULL
#             END
#             AS extracted_numeric_value
#              from tb_model''').display()
```

```
from pyspark.sql import SparkSession

# # Create a Spark session
# spark = SparkSession.builder.appName("example").getOrCreate()

# # Create a PySpark DataFrame (replace this with your actual DataFrame creation logic)
# data = [("47s",), ("*",), ("-42s",), ("45",), ("45",)]
# columns = ["value"]
# df = spark.createDataFrame(data, columns)

# # Register the DataFrame as a temporary SQL table
# df.createOrReplaceTempView("my_table")

# # Define a Spark SQL query to handle the formatting and conversion
# formatted_df = spark.sql("""
#     SELECT
#         CASE
#             WHEN value LIKE '%*%' THEN 9999
#             WHEN value RLIKE '[^(-?\\d+)s$]' THEN 124
#             WHEN value RLIKE '[^-?\\d+$]' THEN 568
#             ELSE NULL
#         END AS formatted_value
#     FROM my_table
# """)

# # Show the result
# formatted_df.show()
```

```
# df_test1 = df_model.select("QUARTER").distinct().withColumn("QUARTER", col("QUARTER").cast("int"))
# encoder = OneHotEncoder(inputCols=["QUARTER"],
#                         outputCols=["QUARTERVec1"])
# model = encoder.fit(df_test1)
# encoded = model.transform(df_test1)
# encoded.display()
```

```
# # Sam's code below this point
# display(dbutils.fs.ls(f"{data_BASE_DIR}"))
```

```
# The following blob storage is accessible to team members only (read and write)
# access key is valid til TTL
# after that you will need to create a new SAS key and authenticate access again via DataBrick command line
blob_container  = "fpteam41container"       # The name of your container created in https://portal.azure.com
storage_account = "fpteam41"  # The name of your Storage account created in https://portal.azure.com
secret_scope    = "fpteam41scope"           # The name of the scope created in your local computer using the
Databricks CLI
secret_key      = "fpteam41key"             # The name of the secret key created in your local computer using the
Databricks CLI
team_blob_url   = f"wasbs://{blob_container}@{storage_account}.blob.core.windows.net"  #points to the root of your
team storage bucket
```

```
# display(dbutils.fs.ls(f"{team_blob_url}/TP"))
```

```
#df_model.count()
```

```
# # Specify the fraction you want to sample (e.g., 1 million out of the total number of rows)
# sample_fraction = 1000 / df_model.count()

# # Use the sample method to perform random sampling
# sampled_df_model = df_model.sample(withReplacement=False, fraction=sample_fraction, seed=42)

# Show the first few rows of the sampled DataFrame
# sampled_df_model.describe().show()
```

```
#sampled_df_model.count()
```

```
#sampled_df_model.printSchema()
```

```
#type(df_model_cat)
```

```
#df_encoded_cat
```

```
#df_encoded_cat.count()
```

```
#df_encoded_cat.printSchema()
```

```
# # Configure Path
# DELTALAKE_GOLD_PATH = "/ml/flights.deltaHH02_12M"

# # Remove table if it exists
# dbutils.fs.rm(DELTALAKE_GOLD_PATH, recurse=True)

# # Save table as Delta Lake
# df_encoded_cat.write.format("delta").mode("overwrite").save(DELTALAKE_GOLD_PATH)
```

Load from checkpoint /ml/flights.deltaHH02_12M

```
# Configure Path
DELTALAKE_GOLD_PATH = "/ml/flights.deltaHH02_12M"

# Re-read as Delta Lakeb
df_encoded = spark.read.format("delta").load(DELTALAKE_GOLD_PATH)

# Review data
display(df_encoded)
```

Table

	HourlyAltimeterSetting	HourlyDewPointTemperature	HourlyDryBulbTemperature	HourlyPrecipitation	HourlyRelativeH
1	30.43	16	44	0.00	32
2	29.92	51	67	0.00	57
3	29.87	61	66	null	83
4	29.78	32	60	0.00	35
5	29.73	58	68	0.00	70
6	29.87	67	88	0.00	50

1,533 rows | Truncated data

Missing values before Imputation with Moving Average

```
# Create a table with details of missing values
# List of vector columns to exclude
vector_columns = [
    "QUARTER_Vec", "DAY_OF_MONTH_Vec", "DAY_OF_WEEK_Vec", "OP_UNIQUE_CARRIER_Vec",
    "TAIL_NUM_Vec", "ORIGIN_AIRPORT_ID_Vec", "ORIGIN_CITY_MARKET_ID_Vec",
    "ORIGIN_STATE_ABR_Vec", "ORIGIN_WAC_Vec", "DEST_AIRPORT_ID_Vec",
    "DEST_CITY_MARKET_ID_Vec", "DEST_STATE_ABR_Vec", "DEST_WAC_Vec",
    "DEP_DEL15_Vec", "DEP_TIME_BLK_Vec", "MONTH_Vec"
]

def calculate_missing_data(df: DataFrame, vector_columns: list) -> DataFrame:
    """
    Calculates the number and percentage of missing values for each column in the DataFrame.

    Args:
    df: The input DataFrame.
    vector_columns: List of column names to exclude from the missing value calculation.

    Returns:
    DataFrame with columns 'Variable', 'Number of missing values', and 'Percentage of missing values'.
    """
    # Calculate total number of rows for percentage calculation
    total_rows = df.count()

    # List to store the result
    missing_data = []

    # Iterating over each column, excluding vector columns, and calculating missing values
    for column in [col for col in df.columns if col not in vector_columns]:
        missing_count = df.filter((col(column).isNull()) | (col(column) == "")).count()
        missing_percent = (missing_count / total_rows) * 100
        missing_data.append((column, missing_count, missing_percent))

    # Creating a DataFrame to display the result
    missing_df = spark.createDataFrame(missing_data, ["Variable", "Number of missing values", "Percentage of missing
values"])

    # Sort the DataFrame by 'Percentage of missing values' in descending order
    sorted_missing_df = missing_df.orderBy(desc("Percentage of missing values"))

    return sorted_missing_df
```

```
# Before appliying Moving Average Imputing #
# Call the calculate_missing_data function
df_encoded_sorted_missing = calculate_missing_data(df_encoded, vector_columns)
# Show the DataFrame
df_encoded_sorted_missing.show(50)
```

```
+--------------------+-----------------------+----------------------------+
|            Variable|Number of missing values|Percentage of missing values|
+--------------------+-----------------------+----------------------------+
| HourlyPrecipitation|                1920356|          16.521027541297492|
|HourlySeaLevelPre...|                1139688|           9.804857451684093|
| HourlyWindDirection|                 556180|           4.784875876097369|
|HourlyAltimeterSe...|                 494928|           4.257918385423997|
| HourlySkyConditions|                 232084|          1.9966434118957563|
|           DEP_DEL15|                 172118|          1.4807495164193731|
|HourlyWetBulbTemp...|                  81406|          0.7003445028040965|
|HourlyStationPres...|                  76980|          0.6622671526160154|
|    HourlyVisibility|                  38732|          0.3332155281257926|
|     HourlyWindSpeed|                  34210|          0.2943122796959456|
|HourlyRelativeHum...|                  32400|          0.2787406565959847|
|HourlyDewPointTem...|                  31888|         0.27433586597323334|
|HourlyDryBulbTemp...|                  30468|          0.2621194544804464|
```

```
|           TAIL_NUM|                29406|         0.2529829551809113|
|         DAY_OF_WEEK|                    0|                        0.0|
|             Flights|                    0|                        0.0|
|   OP_UNIQUE_CARRIER|                    0|                        0.0|
|ORIGIN_STATE_ABR_ind|                    0|                        0.0|
```

Split the data into train (train_df) and test (test_df)

```
def split_flight_data(df: DataFrame, test_period: int) -> (DataFrame, DataFrame):
    """
    Splits the given DataFrame into training and testing datasets based on the specified test period.

    Args:
    df: The input DataFrame with flight data.
    test_period: The period (in days) to be used for the test dataset.

    Returns:
    A tuple containing the training and testing DataFrames.
    """
    # Format CRS_DEP_TIME to HH:MM format
    df_formatted = df.withColumn("CRS_DEP_TIME_STR", format_string("%04d", col("CRS_DEP_TIME")))
    df_formatted = df_formatted.withColumn("CRS_DEP_TIME_FMT", concat(col("CRS_DEP_TIME_STR").substr(1, 2), lit(":"),
col("CRS_DEP_TIME_STR").substr(3, 2)))

    # Combine 'FL_DATE' and 'CRS_DEP_TIME' into a single timestamp
    df_formatted = df_formatted.withColumn(
        "DateTime",
        to_timestamp(concat(col("FL_DATE"), lit(" "), col("CRS_DEP_TIME_FMT")), "yyyy-MM-dd HH:mm")
    )

    # Find the latest timestamp in the dataset
    latest_datetime = df_formatted.agg(max("DateTime")).collect()[0][0]

    # Check if latest_datetime is None
    if latest_datetime is None:
        raise ValueError("The latest datetime could not be determined. Check the data transformation.")

    # Calculate the split datetime
    split_datetime = latest_datetime - timedelta(days=test_period)

    # Split the dataset
    train_df = df_formatted.filter(col("DateTime") <= split_datetime)
    test_df = df_formatted.filter(col("DateTime") > split_datetime)

    return train_df, test_df
```

```
# Split the data into train and test
# Define the test period
test_period = 73
train_df, test_df = split_flight_data(df_encoded, test_period)
```

Impute null values for test and train datasets with moving averages

```
def calculate_moving_averages(df: DataFrame, range_low: int, range_high: int) -> DataFrame:
    """
    Applies moving average to specified columns in the DataFrame.

    Args:
    df: The input DataFrame.
    range_low: The lower bound of the window for the moving average.
    range_high: The upper bound of the window for the moving average.

    Returns:
    DataFrame with moving averages applied.
    """
    # Define window specification
    windowSpec = Window.partitionBy("ORIGIN_AIRPORT_ID").orderBy("DateTime").rowsBetween(range_low, range_high)

    # Columns to calculate moving average
    weather_columns = ["HourlyPrecipitation", "HourlySeaLevelPressure", "HourlyAltimeterSetting",
"HourlyWindDirection", "HourlySkyConditions", "HourlyWetBulbTemperature", "HourlyStationPressure",
"HourlyVisibility", "HourlyRelativeHumidity", "HourlyWindSpeed", "HourlyDewPointTemperature",
"HourlyDryBulbTemperature"]

    # Calculate moving average and impute missing values
    for column in weather_columns:
        moving_avg_col = f"{column}_MovingAvg"
        df = df.withColumn(moving_avg_col, avg(col(column)).over(windowSpec))

        # Replace null values in the original column with the moving average
        df = df.withColumn(column, when(col(column).isNull(), col(moving_avg_col)).otherwise(col(column)))

        # Optionally, drop the moving average column if it's no longer needed
        df = df.drop(moving_avg_col)

    return df
```

```
# Impute null values by calling the calculate_moving_averages for train_df
train_df = calculate_moving_averages(train_df, -10, -1)
train_df = calculate_moving_averages(train_df, -16, -1)
train_df = calculate_moving_averages(train_df, -30, -1)
train_df = calculate_moving_averages(train_df, -30, -1)
train_df = calculate_moving_averages(train_df, -50, -1)
```

```
# Impute null values by calling the calculate_moving_averages for test_df
test_df = calculate_moving_averages(test_df, -10, -1)
test_df = calculate_moving_averages(test_df, -16, -1)
test_df = calculate_moving_averages(test_df, -16, -1)
test_df = calculate_moving_averages(test_df, -30, -1)
# test_df = calculate_moving_averages(test_df, -50, -1)
```

```
# Call the calculate_missing_data function on train_df
train_df_sorted_missing = calculate_missing_data(train_df, vector_columns)
# Show the DataFrame
train_df_sorted_missing.show(50)
```

```
+--------------------+-----------------------+----------------------------+
|            Variable|Number of missing values|Percentage of missing values|
+--------------------+-----------------------+----------------------------+
| HourlyPrecipitation|                 184250|           1.968357990956557|
| HourlySkyConditions|                 154264|          1.6480150725477465|
|           DEP_DEL15|                 144858|          1.5475299964938123|
|HourlySeaLevelPre...|                 105302|          1.1249499764651687|
```

```
|           TAIL_NUM|                25752|          0.2751107461770054|
| HourlyWindDirection|                16734|         0.17877070621800284|
|HourlyWetBulbTemp...|                 2446|          0.0261308203304192|
|HourlyStationPres...|                 2444|         0.02610945416498141|
|    HourlyVisibility|                  688|        0.007349960910600331|
|HourlyAltimeterSe...|                   90|        9.614774447006248E-4|
|     HourlyWindSpeed|                   56|        5.982526322581666E-4|
|HourlyDewPointTem...|                   50|        5.341541359447916E-4|
|HourlyRelativeHum...|                   50|        5.341541359447916E-4|
|HourlyDryBulbTemp...|                   48|        5.127879705069999E-4|
|         DAY_OF_WEEK|                    0|                         0.0|
|OP_UNIQUE_CARRIER...|                    0|                         0.0|
|   OP_UNIQUE_CARRIER|                    0|                         0.0|
```

```
# Call the calculate_missing_data function on test_df
test_df_sorted_missing = calculate_missing_data(test_df, vector_columns)
# Show the DataFrame
test_df_sorted_missing.show(50)
```

```
+--------------------+-----------------------+----------------------------+
|            Variable|Number of missing values|Percentage of missing values|
+--------------------+-----------------------+----------------------------+
| HourlyPrecipitation|                  81158|           3.586120716852973|
| HourlySkyConditions|                  52798|           2.332980132684434|
|HourlySeaLevelPre...|                  39088|           1.727177685260221|
|           DEP_DEL15|                  27260|             1.2045349902833|
| HourlyWindDirection|                   5866|          0.2592003761189229|
|HourlyStationPres...|                   3864|         0.17073819524778688|
|HourlyWetBulbTemp...|                   3864|         0.17073819524778688|
|           TAIL_NUM|                   3654|         0.16145894550605935|
|HourlyAltimeterSe...|                    320|        0.014139809130251502|
|    HourlyVisibility|                    160|        0.007069904565125751|
|     HourlyWindSpeed|                      6|        2.651214211922157E-4|
|HourlyRelativeHum...|                      6|        2.651214211922157E-4|
|HourlyDewPointTem...|                      6|        2.651214211922157E-4|
|HourlyDryBulbTemp...|                      6|        2.651214211922157E-4|
|   OP_UNIQUE_CARRIER|                      0|                         0.0|
|OP_UNIQUE_CARRIER...|                      0|                         0.0|
|   ORIGIN_AIRPORT_ID|                      0|                         0.0|
|           MONTH_ind|                      0|                         0.0|
```

Time-based features: Seasonality, Part of Day, Recency.

```
def add_seasonality_feature(df: DataFrame, date_column: str) -> DataFrame:
    """
    Adds a 'Season' column to the DataFrame based on the month of the year from the specified date column.

    Args:
    df: The input DataFrame.
    date_column: The name of the column containing date information.

    Returns:
    DataFrame with an additional 'Season' column.
    """
    # Add a seasonality column
    df_with_season = df.withColumn("Season",
                                   when((month(date_column) >= 3) & (month(date_column) <= 5), lit("Spring"))
                                   .when((month(date_column) >= 6) & (month(date_column) <= 8), lit("Summer"))
                                   .when((month(date_column) >= 9) & (month(date_column) <= 11), lit("Autumn"))
                                   .otherwise(lit("Winter")))
    return df_with_season
```

```
# Call the add_seasonality_feature function on train_df
train_df = add_seasonality_feature(train_df, "FL_DATE")

# Call the add_seasonality_feature function on test_df
test_df = add_seasonality_feature(test_df, "FL_DATE")
```

```
def add_part_of_day_feature(df: DataFrame, timestamp_column: str) -> DataFrame:
    """
    Adds a 'PartOfDay' column to the DataFrame based on the time of day from the specified timestamp column.

    Args:
    df: The input DataFrame.
    timestamp_column: The name of the column containing timestamp information.

    Returns:
    DataFrame with an additional 'PartOfDay' column.
    """
    # Function to categorize part of day
    def part_of_day(hour_col):
        return (when((hour_col >= 6) & (hour_col < 12), lit("Morning"))
                .when((hour_col >= 12) & (hour_col < 18), lit("Afternoon"))
                .when((hour_col >= 18) & (hour_col < 24), lit("Evening"))
                .otherwise(lit("Night")))

    # Add a part of day column
    df_with_part_of_day = df.withColumn("PartOfDay", part_of_day(hour(timestamp_column)))

    return df_with_part_of_day
```

```
# Call the add_part_of_day_feature function on train_df
train_df = add_part_of_day_feature(train_df, "DateTime")

# Call the add_part_of_day_feature function on test_df
test_df = add_part_of_day_feature(test_df, "DateTime")
```

```
NameError: name 'hour' is not defined
```

```
def add_recency_feature(df: DataFrame, timestamp_column: str, partition_column: str) -> DataFrame:
    """
    Adds a 'Recency' column to the DataFrame, indicating the time difference in hours from the last event
    at the same partition (like an airport).

    Args:
    df: The input DataFrame.
    timestamp_column: The name of the column containing timestamp information.
    partition_column: The column name to partition by (e.g., airport ID).

    Returns:
    DataFrame with an additional 'Recency' column.
    """
    # Define the window specification
    windowSpec = Window.partitionBy(partition_column).orderBy(timestamp_column)

    # Calculate the time difference from the last event
    df_with_recency = df.withColumn("LastEventTime", lag(timestamp_column).over(windowSpec))
    df_with_recency = df_with_recency.withColumn("Recency",
                                                 (col(timestamp_column).cast("long") -
col("LastEventTime").cast("long")) / 3600) # in hours

    return df_with_recency
```

```
# Call the add_recency_feature function on train_df
train_df = add_recency_feature(train_df, "DateTime", "ORIGIN_AIRPORT_ID")

# Call the add_recency_feature function on test_df
test_df = add_recency_feature(test_df, "DateTime", "ORIGIN_AIRPORT_ID")
```

```
def add_prev_flight_delay_feature(df, tail_num_col, delay_flag_col, datetime_col):
    """
    Adds a feature to indicate whether the previous flight for the same plane was delayed.

    Args:
    df (DataFrame): Spark DataFrame containing flight data.
    tail_num_col (str): Column name for the tail number identifying each plane.
    delay_flag_col (str): Column name indicating whether a flight is delayed.
    datetime_col (str): Column name for the datetime of each flight.

    Returns:
    DataFrame: Modified DataFrame with an additional 'Prev_Flight_Delayed' column.
    """

    # Define window specification: partition by plane and order by datetime
    windowSpec = Window.partitionBy(tail_num_col).orderBy(datetime_col)

    # Create 'Prev_Flight_Delayed' column using lag function
    df_with_feature = df.withColumn("Prev_Flight_Delayed", lag(delay_flag_col, 1).over(windowSpec))

    # Replace nulls in 'Prev_Flight_Delayed' for first flights of each plane
    df_with_feature = df_with_feature.fillna({"Prev_Flight_Delayed": 0})

    return df_with_feature
```

```
# Call the add_prev_flight_delay_feature function on train_df
train_df = add_prev_flight_delay_feature(train_df, 'TAIL_NUM', 'DEP_DEL15', 'DateTime')

# Call the add_prev_flight_delay_feature function on test_df
test_df = add_prev_flight_delay_feature(test_df, 'TAIL_NUM', 'DEP_DEL15', 'DateTime')
```

```
train_df.printSchema()
```

```
root
 |-- HourlyAltimeterSetting: decimal(30,22) (nullable = true)
 |-- HourlyDewPointTemperature: double (nullable = true)
 |-- HourlyDryBulbTemperature: double (nullable = true)
 |-- HourlyPrecipitation: decimal(30,22) (nullable = true)
 |-- HourlyRelativeHumidity: double (nullable = true)
 |-- HourlySeaLevelPressure: decimal(30,22) (nullable = true)
 |-- HourlyStationPressure: decimal(30,22) (nullable = true)
 |-- HourlyVisibility: decimal(30,22) (nullable = true)
 |-- HourlyWetBulbTemperature: double (nullable = true)
 |-- HourlyWindDirection: double (nullable = true)
 |-- HourlyWindSpeed: double (nullable = true)
 |-- Flights: decimal(10,1) (nullable = true)
 |-- Year: integer (nullable = true)
 |-- CRS_DEP_TIME: integer (nullable = true)
 |-- FL_DATE: date (nullable = true)
 |-- Elevation: decimal(10,1) (nullable = true)
 |-- HourlySkyConditions: string (nullable = true)
 |-- QUARTER: integer (nullable = true)
 |-- DAY_OF_MONTH: string (nullable = true)
 |-- DAY_OF_MONTH_Cat: string (nullable = true)
```

```
train_df.select("Prev_Flight_Delayed").show(70)
```

```
+-------------------+
|Prev_Flight_Delayed|
+-------------------+
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
|                  0|
```

```
train_df.show(50)
```

```
+---------------------+-------------------------+-----------------------+--------------------+----------------------+--
-------------------+---------------------+--------------------+------------------------+-------------------+-----------
---+-------+----+------------+----------+---------+-------------------+-------+------------+---------------+-----------
+-----------------+--------+-----------------+---------------------+---------------+-----------+---------------+-----------
----------+-------------+--------+---------+------------+------+-------------------+---------------+--------------------+--
----------+---------------------+-------------------------+--------------------+--------------+-------------------+------
-----------------+------------------+-----------+-------------+---------------+----------+--------------+---------------
+---------------+---------------------+-------------------+----------------------+-------------------------+--------------
------+---------------+------------------+---------------------+------------------+---------------+-------------+-----
------------+--------------+----------------+----------------+-------------------+------+------------------+------------
-------+------------------+
|HourlyAltimeterSetting|HourlyDewPointTemperature|HourlyDryBulbTemperature| HourlyPrecipitation|HourlyRelativeHumidity|Ho
urlySeaLevelPressure|HourlyStationPressure|    HourlyVisibility|HourlyWetBulbTemperature|HourlyWindDirection|HourlyWindSp
eed|Flights|Year|CRS_DEP_TIME|   FL_DATE|Elevation| HourlySkyConditions|QUARTER|DAY_OF_MONTH|DAY_OF_MONTH_Cat|DAY_OF_WEEK
|OP_UNIQUE_CARRIER|TAIL_NUM|ORIGIN_AIRPORT_ID|ORIGIN_CITY_MARKET_ID|ORIGIN_STATE_ABR|ORIGIN_WAC|DEST_AIRPORT_ID|DEST_CITY
_MARKET_ID|DEST_STATE_ABR|DEST_WAC|DEP_DEL15|DEP_TIME_BLK|MONTH|DAY_OF_MONTH_ind|DAY_OF_WEEK_ind|OP_UNIQUE_CARRIER_ind|TA
IL_NUM_ind|ORIGIN_AIRPORT_ID_ind|ORIGIN_CITY_MARKET_ID_ind|ORIGIN_STATE_ABR_ind|ORIGIN_WAC_ind|DEST_AIRPORT_ID_ind|DEST_C
ITY_MARKET_ID_ind|DEST_STATE_ABR_ind|DEST_WAC_ind|DEP_DEL15_ind|DEP_TIME_BLK_ind|MONTH_ind|  QUARTER_Vec|DAY_OF_MONTH_Vec
|DAY_OF_WEEK_Vec|OP_UNIQUE_CARRIER_Vec|        TAIL_NUM_Vec|ORIGIN_AIRPORT_ID_Vec|ORIGIN_CITY_MARKET_ID_Vec|ORIGIN_STATE_A
BR_Vec| ORIGIN_WAC_Vec|DEST_AIRPORT_ID_Vec|DEST_CITY_MARKET_ID_Vec|DEST_STATE_ABR_Vec|   DEST_WAC_Vec|DEP_DEL15_Vec|DEP_T
IME_BLK_Vec|     MONTH_Vec|CRS_DEP_TIME_STR|CRS_DEP_TIME_FMT|           DateTime|Season|      LastEventTime|
```

```
+---------------------+-------------------------+-----------------------+--------------------+----------------------+--
-------------------+---------------------+--------------------+------------------------+-------------------+-----------
---+-------+----+------------+----------+---------+-------------------+-------+------------+---------------+-----------
+-----------------+--------+-----------------+---------------------+---------------+-----------+---------------+-----------
----------+-------------+--------+---------+------------+------+-------------------+---------------+--------------------+--
----------+---------------------+-------------------------+--------------------+--------------+-------------------+------
-----------------+------------------+-----------+-------------+---------------+----------+-----------+---------------+-
--------------+---------------------+-------------------+----------------------+-------------------------+--------------
----+---------------+------------------+---------------------+------------------+---------------+-------------+--------
---------+--------------+----------------+----------------+-------------------+------+------------------+--------------
-----+------------------+
|HourlyAltimeterSetting|HourlyDewPointTemperature|HourlyDryBulbTemperature| HourlyPrecipitation|HourlyRelativeHumidity|Ho
urlySeaLevelPressure|HourlyStationPressure|    HourlyVisibility|HourlyWetBulbTemperature|HourlyWindDirection|HourlyWindSp
eed|Flights|Year|CRS_DEP_TIME|   FL_DATE|Elevation| HourlySkyConditions|QUARTER|DAY_OF_MONTH|DAY_OF_MONTH_Cat|DAY_OF_WEEK
|OP_UNIQUE_CARRIER|TAIL_NUM|ORIGIN_AIRPORT_ID|ORIGIN_CITY_MARKET_ID|ORIGIN_STATE_ABR|ORIGIN_WAC|DEST_AIRPORT_ID|DEST_CITY
_MARKET_ID|DEST_STATE_ABR|DEST_WAC|DEP_DEL15|DEP_TIME_BLK|MONTH|DAY_OF_MONTH_ind|DAY_OF_WEEK_ind|OP_UNIQUE_CARRIER_ind|TA
IL_NUM_ind|ORIGIN_AIRPORT_ID_ind|ORIGIN_CITY_MARKET_ID_ind|ORIGIN_STATE_ABR_ind|ORIGIN_WAC_ind|DEST_AIRPORT_ID_ind|DEST_C
ITY_MARKET_ID_ind|DEST_STATE_ABR_ind|DEST_WAC_ind|DEP_DEL15_ind|DEP_TIME_BLK_ind|MONTH_ind|QUARTER_Vec|DAY_OF_MONTH_Vec|D
AY_OF_WEEK_Vec|OP_UNIQUE_CARRIER_Vec|        TAIL_NUM_Vec|ORIGIN_AIRPORT_ID_Vec|ORIGIN_CITY_MARKET_ID_Vec|ORIGIN_STATE_ABR
_Vec| ORIGIN_WAC_Vec|DEST_AIRPORT_ID_Vec|DEST_CITY_MARKET_ID_Vec|DEST_STATE_ABR_Vec|   DEST_WAC_Vec|DEP_DEL15_Vec|DEP_TIM
E_BLK_Vec|     MONTH_Vec|CRS_DEP_TIME_STR|CRS_DEP_TIME_FMT|           DateTime|Season|      LastEventTime|            Pr
```