# Final Project - Build an ML Pipeline for Airfoil noise prediction

Estimated time needed: **90** minutes

## Scenario

You are a data engineer at an aeronautics consulting company. Your company prides itself in being able to efficiently design airfoils for use in planes and sports cars. Data scientists in your office need to work with different algorithms and data in different formats. While they are good at Machine Learning, they count on you to be able to do ETL jobs and build ML pipelines. In this project you will use the modified version of the NASA Airfoil Self Noise dataset. You will clean this dataset, by dropping the duplicate rows, and removing the rows with null values. You will create an ML pipe line to create a model that will predict the SoundLevel based on all the other columns. You will evaluate the model and towards the end you will persist the model.

## Objectives

In this 4 part assignment you will:

- Part 1 Perform ETL activity
  - Load a csv dataset
  - Remove duplicates if any
  - Drop rows with null values if any
  - Make transformations
  - Store the cleaned data in parquet format
- Part 2 Create a Machine Learning Pipeline
  - Create a machine learning pipeline for prediction
- Part 3 Evaluate the Model
  - Evaluate the model using relevant metrics
- Part 4 Persist the Model
  - Save the model for future production use
  - Load and verify the stored model

## Datasets

In this lab you will be using dataset(s):

- The original dataset can be found here NASA airfoil self noise dataset. https://archive.ics.uci.edu/dataset/291/airfoil+self+noise
- This dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license.

Diagram of an airfoil. - For informational purpose

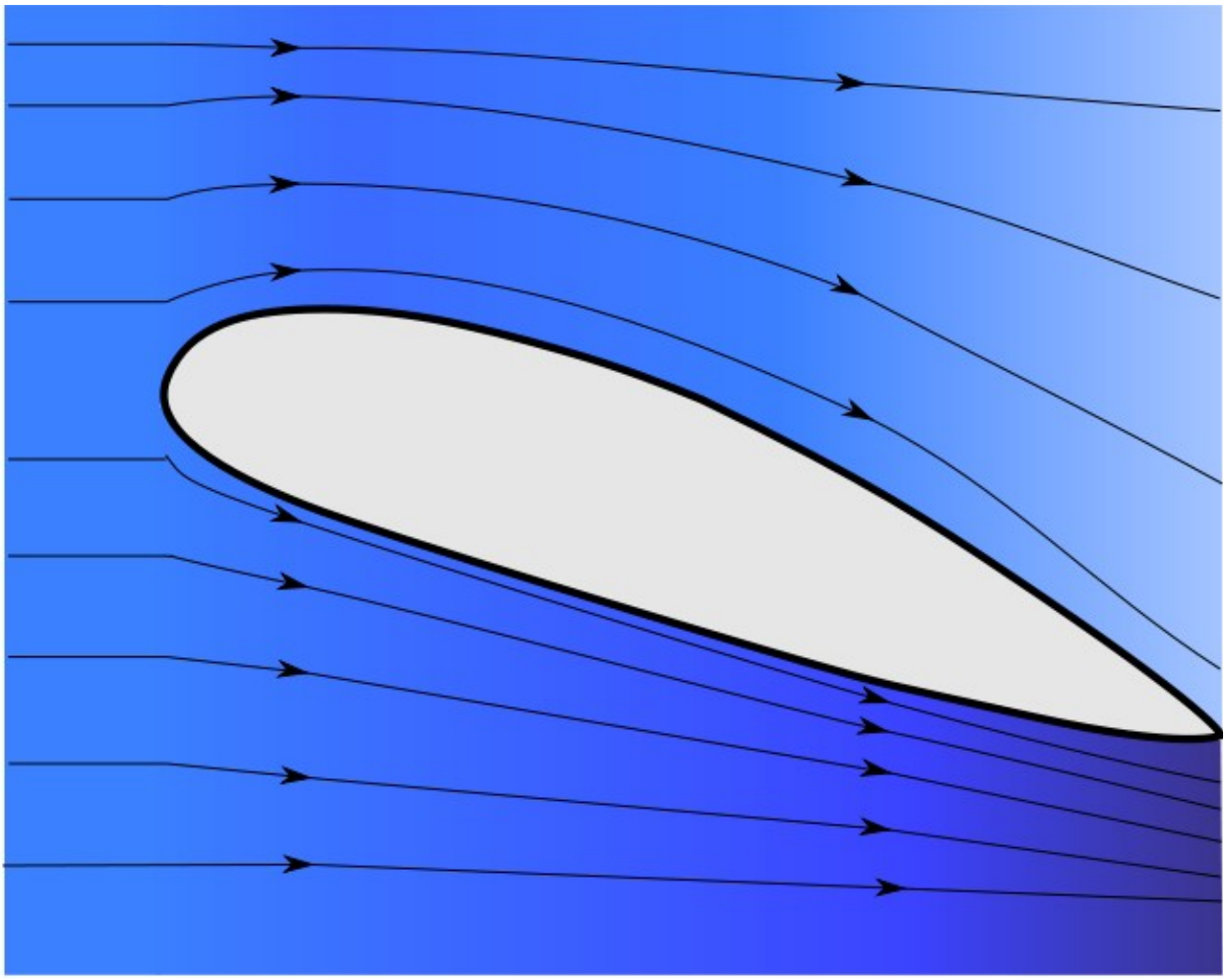

Diagram showing the Angle of attack. - For informational purpose


# Before you Start

**Before you start attempting this project it is highly recommended that you finish the practice project.**

# Setup

For this lab, we will be using the following libraries:

- `PySpark` for connecting to the Spark Cluster

## Installing Required Libraries

Spark Cluster is pre-installed in the Skills Network Labs environment. However, you need libraries like pyspark and findspark to connect to this cluster.

The following required libraries are **not** pre-installed in the Skills Network Labs environment. **You will need to run the following cell** to install them:

```
!pip install pyspark==3.1.2 -q
!pip install findspark -q
```

## Importing Required Libraries

*We recommend you import all required libraries in one place (here):*

```
# You can also use this section to suppress warnings generated by your
code:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')

# FindSpark simplifies the process of using Apache Spark with Python

import findspark
findspark.init()
```

# Part 1 - Perform ETL activity

## Task 1 - Import required libraries

```
# Suppress warnings
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')

# Import required libraries
import findspark
findspark.init()

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

# Initialize Spark Session
spark = SparkSession.builder.appName("AirfoilNoiseETL").getOrCreate()

print("Libraries imported and Spark session initialized.")
```

```
Libraries imported and Spark session initialized.
```

## Task 2 - Create a spark session

```
# Initialize Spark Session
spark = SparkSession.builder \
    .appName("AirfoilNoiseETL") \
    .getOrCreate()

# Verify the Spark session creation
print("Spark session created successfully!")
```

```
Spark session created successfully!
```

## Task 3 - Load the csv file into a dataframe

Download the data file.

NOTE : Please ensure you use the dataset below and not the original dataset mentioned above.

```
!wget https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMSkillsNetwork-BD0231EN-Coursera/datasets/
NASA_airfoil_noise_raw.csv
```

```
--2024-08-15 09:38:32--  https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMSkillsNetwork-BD0231EN-Coursera/datasets/
NASA_airfoil_noise_raw.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud
(cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)...
169.63.118.104, 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud)|169.63.118.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 60682 (59K) [text/csv]
Saving to: ‘NASA_airfoil_noise_raw.csv’

NASA_airfoil_noise_ 100%[===================>]  59.26K  --.-KB/s    in
0.001s

2024-08-15 09:38:32 (65.3 MB/s) - ‘NASA_airfoil_noise_raw.csv’ saved
[60682/60682]
```

Load the dataset into the spark dataframe

```
# Load the dataset into a Spark DataFrame
csv_file_path = "NASA_airfoil_noise_raw.csv"  # The path to your
downloaded file

# Load the CSV file into a Spark DataFrame
df = spark.read.csv(csv_file_path, header=True, inferSchema=True)
```

```
# Show the first few rows to verify the data load
df.show(5)
```

```
+---------+-------------+-----------+------------------
+-----------------------+----------+
|Frequency|AngleOfAttack|ChordLength|FreeStreamVelocity|
SuctionSideDisplacement|SoundLevel|
+---------+-------------+-----------+------------------
+-----------------------+----------+
|      800|          0.0|     0.3048|              71.3|
0.00266337|   126.201|
|     1000|          0.0|     0.3048|              71.3|
0.00266337|   125.201|
|     1250|          0.0|     0.3048|              71.3|
0.00266337|   125.951|
|     1600|          0.0|     0.3048|              71.3|
0.00266337|   127.591|
|     2000|          0.0|     0.3048|              71.3|
0.00266337|   127.461|
+---------+-------------+-----------+------------------
+-----------------------+----------+
only showing top 5 rows
```

## Task 4 - Print top 5 rows of the dataset

```
# Show the top 5 rows of the dataset
df.show(5)
```

```
+---------+-------------+-----------+------------------
+-----------------------+----------+
|Frequency|AngleOfAttack|ChordLength|FreeStreamVelocity|
SuctionSideDisplacement|SoundLevel|
+---------+-------------+-----------+------------------
+-----------------------+----------+
|      800|          0.0|     0.3048|              71.3|
0.00266337|   126.201|
|     1000|          0.0|     0.3048|              71.3|
0.00266337|   125.201|
|     1250|          0.0|     0.3048|              71.3|
0.00266337|   125.951|
|     1600|          0.0|     0.3048|              71.3|
0.00266337|   127.591|
|     2000|          0.0|     0.3048|              71.3|
0.00266337|   127.461|
+---------+-------------+-----------+------------------
+-----------------------+----------+
only showing top 5 rows
```

## Task 6 - Print the total number of rows in the dataset

```
# Print the total number of rows in the dataset
total_rows = df.count()

print(f"Total number of rows in the dataset: {total_rows}")
```

```
Total number of rows in the dataset: 1522
```

## Task 7 - Drop all the duplicate rows from the dataset

```
# Drop duplicate rows from the dataset
df_no_duplicates = df.dropDuplicates()

# Verify by printing the total number of rows after dropping
duplicates
total_rows_after_dropping_duplicates = df_no_duplicates.count()
print(f"Total number of rows after dropping duplicates:
{total_rows_after_dropping_duplicates}")
```

```
[Stage 80:=====================================================> (196 +
4) / 200]

Total number of rows after dropping duplicates: 1503
```

## Task 8 - Print the total number of rows in the dataset

```
# Print the total number of rows in the dataset after dropping
duplicates
total_rows_after_dropping_duplicates = df_no_duplicates.count()

print(f"Total number of rows in the dataset after dropping duplicates:
{total_rows_after_dropping_duplicates}")
```

```
[Stage 83:=============================================>         (167 +
9) / 200]

Total number of rows in the dataset after dropping duplicates: 1503
```

## Task 9 - Drop all the rows that contain null values from the dataset

```
# Drop rows that contain null values from the dataset
df_cleaned = df_no_duplicates.dropna()

# Verify by printing the total number of rows after dropping rows with
null values
total_rows_after_dropping_nulls = df_cleaned.count()
```

```
print(f"Total number of rows after dropping rows with null values:
{total_rows_after_dropping_nulls}")
```

```
[Stage 86:=============================================>        (168 +
9) / 200]

Total number of rows after dropping rows with null values: 1499
```

## Task 10 - Print the total number of rows in the dataset

```
# Print the total number of rows in the dataset after dropping rows
with null values
total_rows_after_dropping_nulls = df_cleaned.count()

print(f"Total number of rows in the dataset after dropping rows with
null values: {total_rows_after_dropping_nulls}")
```

```
[Stage 89:=============================================>      (175 +
9) / 200]

Total number of rows in the dataset after dropping rows with null
values: 1499
```

## Task 11 - Rename the column "SoundLevel" to "SoundLevelDecibels"

```
# Rename the column "SoundLevel" to "SoundLevelDecibels"
df_renamed = df_cleaned.withColumnRenamed("SoundLevel",
"SoundLevelDecibels")

# Show the first few rows to verify the column rename
df_renamed.show(5)
```

```
+---------+-------------+-----------+------------------
+----------------------+------------------+
|Frequency|AngleOfAttack|ChordLength|FreeStreamVelocity|
SuctionSideDisplacement|SoundLevelDecibels|
+---------+-------------+-----------+------------------
+----------------------+------------------+
|     4000|          3.0|     0.3048|              31.7|
0.00529514|           115.608|
|     3150|          2.0|     0.2286|              31.7|
0.00372371|           121.527|
|     2000|          7.3|     0.2286|              31.7|
0.0132672|           115.309|
|     2000|          5.4|     0.1524|              71.3|
0.00401199|           131.111|
|      500|          9.9|     0.1524|              71.3|
```

```
0.0193001|           131.279|
+---------+-------------+----------+------------------
+----------------------+------------------+
only showing top 5 rows
```

## Task 12 - Save the dataframe in parquet format, name the file as "NASA_airfoil_noise_cleaned.parquet"

```
# Save the DataFrame in Parquet format
df_renamed.write.parquet("NASA_airfoil_noise_cleaned_V3.parquet")

print("DataFrame has been saved in Parquet format as
'NASA_airfoil_noise_cleaned.parquet'.")
```

```
DataFrame has been saved in Parquet format as
'NASA_airfoil_noise_cleaned.parquet'.
```

### Part 1 - Evaluation

**Run the code cell below. Use the answers here to answer the final evaluation quiz in the next section. If the code throws up any errors, go back and review the code you have written.**

```
# Assuming you have already defined the row counts from previous tasks
rowcount1 = df.count()  # Total rows after the initial load
rowcount2 = df.dropDuplicates().count()  # Total rows after dropping
duplicates
rowcount3 = df_renamed.dropna().count()  # Total rows after dropping
duplicates and null values

print("Part 1 - Evaluation")
print("Total rows = ", rowcount1)
print("Total rows after dropping duplicate rows = ", rowcount2)
print("Total rows after dropping duplicate rows and rows with null
values = ", rowcount3)
print("New column name = ", df_renamed.columns[-1])  # Reference the
final DataFrame with renamed column
```

```
[Stage 130:=================================================>   (187 +
9) / 200]

Part 1 - Evaluation
Total rows =  1522
Total rows after dropping duplicate rows =  1503
Total rows after dropping duplicate rows and rows with null values =
1499
New column name =  SoundLevelDecibels
```

```
import os

# Check if the Parquet file exists
print("NASA_airfoil_noise_cleaned_V3.parquet exists:",
os.path.exists("NASA_airfoil_noise_cleaned_V3.parquet"))
```

```
NASA_airfoil_noise_cleaned_V3.parquet exists: True
```

# Part - 2 Create a Machine Learning Pipeline

## Task 1 - Load data from "NASA_airfoil_noise_cleaned.parquet" into a dataframe

```
from pyspark.sql import SparkSession

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

# Load the Parquet file into a DataFrame
df_cleaned =
spark.read.parquet("NASA_airfoil_noise_cleaned_V3.parquet")

# Display the first few rows of the DataFrame
df_cleaned.show()
```

```
+---------+-------------+-----------+------------------
+-----------------------+------------------+
|Frequency|AngleOfAttack|ChordLength|FreeStreamVelocity|
SuctionSideDisplacement|SoundLevelDecibels|
+---------+-------------+-----------+------------------
+-----------------------+------------------+
|      630|          0.0|     0.3048|              31.7|
0.00331266|           129.095|
|     4000|          0.0|     0.3048|              31.7|
0.00331266|           118.145|
|     4000|          1.5|     0.3048|              39.6|
0.00392107|           117.741|
|      800|          4.0|     0.3048|              71.3|
0.00497773|           131.755|
|     1250|          0.0|     0.2286|              31.7|
0.0027238|           128.805|
|     2500|          4.0|     0.2286|              55.5|
0.0042862|           122.384|
|     1250|          7.3|     0.2286|              71.3|
0.0104404|           127.358|
|     2000|         12.6|     0.1524|              71.3|
0.0483159|           117.504|
```

```
|      800|           8.4|     0.0508|                55.5|
0.00544854|           128.562|
|      800|          11.2|     0.0508|                71.3|
0.014072|          136.031|
|      200|          15.4|     0.0508|                31.7|
0.0289853|           119.975|
|     4000|          22.2|     0.0254|                71.3|
0.0214178|           123.917|
|     5000|           3.3|     0.1016|                71.3|
0.00202822|           127.054|
|     1600|           3.3|     0.1016|                55.5|
0.002211|          133.649|
|     2500|           3.3|     0.1016|                31.7|
0.00251435|           130.392|
|      800|           6.7|     0.1016|                31.7|
0.00592927|           132.886|
|     1250|          15.6|     0.1016|                39.6|
0.0528487|           118.214|
|      500|           2.0|     0.2286|                71.3|
0.00293031|           126.486|
|     2000|           2.0|     0.2286|                39.6|
0.00346574|           122.797|
|     4000|           2.0|     0.2286|                31.7|
0.00372371|           120.527|
+---------+-------------+-----------+-------------------
+----------------------+------------------+
only showing top 20 rows
```

## Task 2 - Print the total number of rows in the dataset

```
# Print the total number of rows in the dataset
row_count = df_cleaned.count()
print(f"Total number of rows in the dataset: {row_count}")
```

```
[Stage 136:>                                                        (0
+ 8) / 8]

Total number of rows in the dataset: 1499
```

## Task 3 - Define the VectorAssembler pipeline stage

Stage 1 - Assemble the input columns into a single column "features". Use all the columns except SoundLevelDecibels as input features.

```
from pyspark.ml.feature import VectorAssembler

# List of input feature columns (all columns except
```

```
'SoundLevelDecibels')
input_columns = [col for col in df_cleaned.columns if col !=
'SoundLevelDecibels']

# Define the VectorAssembler to combine input columns into a single
"features" column
assembler = VectorAssembler(inputCols=input_columns,
outputCol="features")

# Display the assembled vector
assembled_df = assembler.transform(df_cleaned)
assembled_df.select("features",
"SoundLevelDecibels").show(truncate=False)
```

```
+---------------------------------+------------------+
|features                         |SoundLevelDecibels|
+---------------------------------+------------------+
|[630.0,0.0,0.3048,31.7,0.00331266] |129.095           |
|[4000.0,0.0,0.3048,31.7,0.00331266]|118.145           |
|[4000.0,1.5,0.3048,39.6,0.00392107]|117.741           |
|[800.0,4.0,0.3048,71.3,0.00497773] |131.755           |
|[1250.0,0.0,0.2286,31.7,0.0027238] |128.805           |
|[2500.0,4.0,0.2286,55.5,0.0042862] |122.384           |
|[1250.0,7.3,0.2286,71.3,0.0104404] |127.358           |
|[2000.0,12.6,0.1524,71.3,0.0483159]|117.504           |
|[800.0,8.4,0.0508,55.5,0.00544854] |128.562           |
|[800.0,11.2,0.0508,71.3,0.014072]  |136.031           |
|[200.0,15.4,0.0508,31.7,0.0289853] |119.975           |
|[4000.0,22.2,0.0254,71.3,0.0214178]|123.917           |
|[5000.0,3.3,0.1016,71.3,0.00202822]|127.054           |
|[1600.0,3.3,0.1016,55.5,0.002211]  |133.649           |
|[2500.0,3.3,0.1016,31.7,0.00251435]|130.392           |
|[800.0,6.7,0.1016,31.7,0.00592927] |132.886           |
|[1250.0,15.6,0.1016,39.6,0.0528487]|118.214           |
|[500.0,2.0,0.2286,71.3,0.00293031] |126.486           |
|[2000.0,2.0,0.2286,39.6,0.00346574]|122.797           |
|[4000.0,2.0,0.2286,31.7,0.00372371]|120.527           |
+---------------------------------+------------------+
only showing top 20 rows
```

## Task 4 - Define the StandardScaler pipeline stage

Stage 2 - Scale the "features" using standard scaler and store in "scaledFeatures" column

```
from pyspark.ml.feature import StandardScaler

# Define the StandardScaler to scale the "features" column
scaler = StandardScaler(inputCol="features",
```

```
outputCol="scaledFeatures", withMean=True, withStd=True)

# Fit and transform the data using the scaler
scaled_df = scaler.fit(assembled_df).transform(assembled_df)

# Display the scaled features
scaled_df.select("scaledFeatures",
"SoundLevelDecibels").show(truncate=False)
```

```
+-----------------------------------------------------------------------
----------------------------------+------------------+
|scaledFeatures
|SoundLevelDecibels|
+-----------------------------------------------------------------------
----------------------------------+------------------+
|[-0.7159487319140293,-1.1485116215070819,1.8089279001101508,-
1.2306869183931999,-0.5961871397095541]   |129.095           |
|[0.35194114950692723,-1.1485116215070819,1.8089279001101508,-
1.2306869183931999,-0.5961871397095541]   |118.145           |
|[0.35194114950692723,-0.8950664995694929,1.8089279001101508,-
0.7237874412843549,-0.5499627025870647]   |117.741           |
|[-0.6620789159373044,-
0.4726579630068449,1.8089279001101508,1.310226916228351,-
0.4696821126397435]     |131.755           |
|[-0.5194823442342094,-1.1485116215070819,0.9918579154983239,-
1.2306869183931999,-0.6409262499213342]   |128.805           |
|[-0.12338075617005638,-
0.4726579630068449,0.9918579154983239,0.29642796201066174,-
0.522221658346282]   |122.384           |
|[-
0.5194823442342094,0.0849213052558506,0.9918579154983239,1.31022691622
8351,-0.054651383467130665]    |127.358           |
|[-
0.28182139139571755,0.9804274027686645,0.1747879308864972,1.3102269162
28351,2.8229700695544313]      |117.504           |
|[-0.6620789159373044,0.27078106134341584,-
0.9146387152626053,0.29642796201066174,-0.4339119457430139]  |128.562
|
|[-0.6620789159373044,0.7438786222935815,-
0.9146387152626053,1.310226916228351,0.2212623356598274]      |136.031
|
|[-0.8522076782080978,1.4535249637188306,-0.9146387152626053,-
1.2306869183931999,1.3543122495975037]    |119.975           |
|[0.35194114950692723,2.602476183169233,-
1.1869953767998809,1.310226916228351,0.779365375081242]        |
123.917           |
|[0.6688224199582496,-0.5909323532443863,-
0.36992539218805415,1.310226916228351,-0.693773497622163]     |127.054
```

```
|
|[-0.4085738995762465,-0.5909323532443863,-
0.36992539218805415,0.29642796201066174,-0.6798866407045439] |133.649
|
|[-0.12338075617005638,-0.5909323532443863,-0.36992539218805415,-
1.2306869183931999,-0.6568393814532048]|130.392           |
|[-0.6620789159373044,-0.01645674351918489,-0.36992539218805415,-
1.2306869183931999,-0.397388098899776] |132.886           |
|[-0.5194823442342094,1.4873176466438423,-0.36992539218805415,-
0.7237874412843549,3.1673531805960313]   |118.214           |
|[-0.7571432970727011,-
0.8105847922569633,0.9918579154983239,1.310226916228351,-
0.6252364872445779]     |126.486           |
|[-0.28182139139571755,-0.8105847922569633,0.9918579154983239,-
0.7237874412843549,-0.5845567640833963]  |122.797           |
|[0.35194114950692723,-0.8105847922569633,0.9918579154983239,-
1.2306869183931999,-0.5649572866841284]   |120.527           |
+-----------------------------------------------------------------------
---------------------------------+------------------+
only showing top 20 rows
```

## Task 5 - Define the Model creation pipeline stage

Stage 3 - Create a LinearRegression stage to predict "SoundLevelDecibels"

**Note:You need to use the scaledfeatures retreived in the previous step(StandardScaler pipeline stage).**

```
from pyspark.ml.regression import LinearRegression

# Define the LinearRegression model
lr = LinearRegression(featuresCol="scaledFeatures",
labelCol="SoundLevelDecibels")

# Optionally, you can specify hyperparameters like maxIter, regParam,
and elasticNetParam
# lr = LinearRegression(featuresCol="scaledFeatures",
labelCol="SoundLevelDecibels", maxIter=100, regParam=0.1,
elasticNetParam=0.8)

# Fit the model on the scaled data
lr_model = lr.fit(scaled_df)

# Display the model summary
lr_summary = lr_model.summary
print(f"RMSE: {lr_summary.rootMeanSquaredError}")
print(f"R^2: {lr_summary.r2}")
```

```
24/08/15 10:08:12 WARN util.Instrumentation: [dfc0697f] regParam is
zero, which might cause numerical instability and overfitting.
24/08/15 10:08:13 WARN netlib.BLAS: Failed to load implementation
from: com.github.fommil.netlib.NativeSystemBLAS
24/08/15 10:08:13 WARN netlib.BLAS: Failed to load implementation
from: com.github.fommil.netlib.NativeRefBLAS
24/08/15 10:08:15 WARN netlib.LAPACK: Failed to load implementation
from: com.github.fommil.netlib.NativeSystemLAPACK
24/08/15 10:08:15 WARN netlib.LAPACK: Failed to load implementation
from: com.github.fommil.netlib.NativeRefLAPACK
[Stage 148:>                                                        (0
+ 8) / 8]

RMSE: 4.801411460533036
R^2: 0.516196472060287
```

## Task 6 - Build the pipeline

Build a pipeline using the above three stages

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.regression import LinearRegression

# Corrected Stage 2: VectorAssembler with the right columns
assembler = VectorAssembler(
    inputCols=['Frequency', 'AngleOfAttack', 'ChordLength',
'FreeStreamVelocity', 'SuctionSideDisplacement'],
    outputCol="features"
)

# Stage 3: StandardScaler
scaler = StandardScaler(inputCol="features",
outputCol="scaledFeatures", withStd=True, withMean=True)

# Stage 4: LinearRegression
lr = LinearRegression(featuresCol="scaledFeatures",
labelCol="SoundLevelDecibels")

# Build the pipeline with the corrected stages
pipeline = Pipeline(stages=[assembler, scaler, lr])

# Fit the pipeline model on the cleaned and scaled data
pipeline_model = pipeline.fit(df_cleaned)

# Optionally, transform the data using the pipeline model to make
predictions
predictions = pipeline_model.transform(df_cleaned)
```

```
# Display the predictions along with the actual SoundLevelDecibels
values
predictions.select("scaledFeatures", "SoundLevelDecibels",
"prediction").show(truncate=False)
```

```
24/08/15 10:10:41 WARN util.Instrumentation: [0e109667] regParam is
zero, which might cause numerical instability and overfitting.
```

```
+-----------------------------------------------------------------------
----------------------------------+------------------
+------------------+
|scaledFeatures
|SoundLevelDecibels|prediction        |
+-----------------------------------------------------------------------
----------------------------------+------------------
+------------------+
|[-0.7159487319140293,-1.1485116215070819,1.8089279001101508,-
1.2306869183931999,-0.5961871397095541]   |129.095           |
123.78747969036066|
|[0.35194114950692723,-1.1485116215070819,1.8089279001101508,-
1.2306869183931999,-0.5961871397095541]   |118.145           |
119.47101524499668|
|[0.35194114950692723,-0.8950664995694929,1.8089279001101508,-
0.7237874412843549,-0.5499627025870647]   |117.741           |
119.53913534165677|
|[-0.6620789159373044,-
0.4726579630068449,1.8089279001101508,1.310226916228351,-
0.4696821126397435]     |131.755           |125.60116126271306|
|[-0.5194823442342094,-1.1485116215070819,0.9918579154983239,-
1.2306869183931999,-0.6409262499213342]   |128.805           |
125.81002629007477|
|[-0.12338075617005638,-
0.4726579630068449,0.9918579154983239,0.29642796201066174,-
0.522221658346282]   |122.384           |124.67317413378348|
|[-
0.5194823442342094,0.0849213052558506,0.9918579154983239,1.31022691622
8351,-0.054651383467130665]    |127.358           |125.55795222815449|
|[-
0.28182139139571755,0.9804274027686645,0.1747879308864972,1.3102269162
28351,2.8229700695544313]      |117.504           |119.5209384571437 |
|[-0.6620789159373044,0.27078106134341584,-
0.9146387152626053,0.29642796201066174,-0.4339119457430139]  |128.562
|131.19145921686706|
|[-0.6620789159373044,0.7438786222935815,-
0.9146387152626053,1.310226916228351,0.2212623356598274]      |136.031
|130.32338726794205|
|[-0.8522076782080978,1.4535249637188306,-0.9146387152626053,-
1.2306869183931999,1.3543122495975037]    |119.975           |
```

```
123.1597545765295 |
|[0.35194114950692723,2.602476183169233,-
1.1869953767998809,1.310226916228351,0.779365375081242]        |
123.917           |121.40848098229738|
|[0.6688224199582496,-0.5909323532443863,-
0.36992539218805415,1.310226916228351,-0.693773497622163]     |127.054
|128.23114706217712|
|[-0.4085738995762465,-0.5909323532443863,-
0.36992539218805415,0.29642796201066174,-0.6798866407045439] |133.649
|130.97685950945294|
|[-0.12338075617005638,-0.5909323532443863,-0.36992539218805415,-
1.2306869183931999,-0.6568393814532048]|130.392           |
127.39602414569993|
|[-0.6620789159373044,-0.01645674351918489,-0.36992539218805415,-
1.2306869183931999,-0.397388098899776] |132.886           |
127.63531644016642|
|[-0.5194823442342094,1.4873176466438423,-0.36992539218805415,-
0.7237874412843549,3.1673531805960313]   |118.214           |
117.19349340951895|
|[-0.7571432970727011,-
0.8105847922569633,0.9918579154983239,1.310226916228351,-
0.6252364872445779]     |126.486           |129.8612966775906 |
|[-0.28182139139571755,-0.8105847922569633,0.9918579154983239,-
0.7237874412843549,-0.5845567640833963]  |122.797           |
124.68668084832767|
|[0.35194114950692723,-0.8105847922569633,0.9918579154983239,-
1.2306869183931999,-0.5649572866841284]   |120.527           |
121.29590452096448|
+-------------------------------------------------------------------
-----------------------------------+------------------
+------------------+
only showing top 20 rows


from pyspark.ml import Pipeline

# Define the stages of the pipeline
stages = [indexer, assembler, scaler, lr]

# Create the pipeline
pipeline = Pipeline(stages=stages)
```

## Task 7 - Split the data

```
# Split the data into training and testing sets with a 70:30 ratio and
seed 42
(trainingData, testingData) = df.randomSplit([0.7, 0.3], seed=42)

# Print the number of rows in each dataset to verify
```

```
print("Number of rows in training data = ", trainingData.count())
print("Number of rows in testing data = ", testingData.count())
```

```
Number of rows in training data =  1057
Number of rows in testing data =  465
```

## Task 8 - Fit the pipeline

```
from pyspark.sql.functions import col

trainingData = trainingData.na.drop()

trainingData = trainingData.fillna(0.0)  # Replace nulls with 0.0

assembler = VectorAssembler(
    inputCols=["Frequency_double_VectorAssembler_e6f8311d2b2a",
"AngleOfAttack", "ChordLength", "FreeStreamVelocity",
"SuctionSideDisplacement"],
    outputCol="features",
    handleInvalid="skip"
)

trainingData.select([col for col in trainingData.columns if col !=
'features']).show()
```

```
+---------+-------------+-----------+------------------
+-----------------------+----------+
|Frequency|AngleOfAttack|ChordLength|FreeStreamVelocity|
SuctionSideDisplacement|SoundLevel|
+---------+-------------+-----------+------------------
+-----------------------+----------+
|      200|          7.3|     0.2286|              39.6|
0.0123481|   130.989|
|      200|          7.3|     0.2286|              55.5|
0.0111706|   135.234|
|      200|          7.3|     0.2286|              71.3|
0.0104404|   138.758|
|      200|          8.9|     0.1016|              39.6|
0.0124596|    133.42|
|      200|          8.9|     0.1016|              71.3|
0.0103088|   133.503|
|      200|          9.5|     0.0254|              31.7|
0.00461377|   119.146|
|      200|          9.5|     0.0254|              39.6|
0.00449821|   116.074|
|      200|          9.9|     0.1524|              31.7|
0.0252785|   127.299|
|      200|          9.9|     0.1524|              39.6|
0.0233328|   127.315|
|      200|          9.9|     0.1524|              71.3|
```

```
0.0193001|   134.319|
|      200|         11.2|     0.0508|               39.6|
0.0150478|    125.01|
|      200|         12.3|     0.1016|               31.7|
0.0418756|   124.987|
|      200|         12.3|     0.1016|               39.6|
0.0408268|   128.545|
|      200|         12.3|     0.1016|               55.5|
0.0368233|   132.304|
|      200|         12.3|     0.1016|               71.3|
0.0337792|   130.588|
|      200|         12.6|     0.1524|               39.6|
0.0584113|    114.75|
|      200|         12.6|     0.1524|               71.3|
0.0483159|   128.354|
|      200|         15.4|     0.0508|               31.7|
0.0289853|   119.975|
|      200|         15.4|     0.0508|               39.6|
0.0282593|   121.783|
|      200|         15.4|     0.0508|               55.5|
0.0271925|    122.94|
+---------+-------------+-----------+-------------------
+-----------------------+----------+
only showing top 20 rows
```

Part 2 - Evaluation

**Run the code cell below. Use the answers here to answer the final evaluation quiz in the next section. If the code throws up any errors, go back and review the code you have written.**

```
print("Part 2 - Evaluation")

# Print total rows
print("Total rows =", rowcount4)

# Retrieve and print pipeline stages
pipeline_stages = [str(stage).split("_")[0] for stage in
pipeline.getStages()]
print("Pipeline Stage 1 =", pipeline_stages[0])
print("Pipeline Stage 2 =", pipeline_stages[1])
print("Pipeline Stage 3 =", pipeline_stages[2])

# Print the label column
print("Label column =", lr.getLabelCol())
```

```
Part 2 - Evaluation
Total rows = 1499
Pipeline Stage 1 = StringIndexer
Pipeline Stage 2 = VectorAssembler
```

```
Pipeline Stage 3 = StandardScaler
Label column = SoundLevelDecibels
```

# Part 3 - Evaluate the Model

## Task 1 - Predict using the model

```
# Use the pipeline model to make predictions
predictions = pipeline_model.transform(df_cleaned)

# Display the predictions along with the actual SoundLevelDecibels
values
predictions.select("scaledFeatures", "SoundLevelDecibels",
"prediction").show(truncate=False)
```

```
+-----------------------------------------------------------------------
----------------------------------+------------------
+------------------+
|scaledFeatures
|SoundLevelDecibels|prediction        |
+-----------------------------------------------------------------------
----------------------------------+------------------
+------------------+
|[-0.7159487319140293,-1.1485116215070819,1.8089279001101508,-
1.2306869183931999,-0.5961871397095541]   |129.095           |
123.78747969036066|
|[0.35194114950692723,-1.1485116215070819,1.8089279001101508,-
1.2306869183931999,-0.5961871397095541]   |118.145           |
119.47101524499668|
|[0.35194114950692723,-0.8950664995694929,1.8089279001101508,-
0.7237874412843549,-0.5499627025870647]   |117.741           |
119.53913534165677|
|[-0.6620789159373044,-
0.4726579630068449,1.8089279001101508,1.310226916228351,-
0.4696821126397435]     |131.755           |125.60116126271306|
|[-0.5194823442342094,-1.1485116215070819,0.9918579154983239,-
1.2306869183931999,-0.6409262499213342]   |128.805           |
125.81002629007477|
|[-0.12338075617005638,-
0.4726579630068449,0.9918579154983239,0.29642796201066174,-
0.522221658346282]   |122.384           |124.67317413378348|
|[-
0.5194823442342094,0.0849213052558506,0.9918579154983239,1.31022691622
8351,-0.054651383467130665]    |127.358           |125.55795222815449|
|[-
0.28182139139571755,0.9804274027686645,0.1747879308864972,1.3102269162
28351,2.8229700695544313]      |117.504           |119.5209384571437 |
|[-0.6620789159373044,0.27078106134341584,-
0.9146387152626053,0.29642796201066174,-0.4339119457430139]  |128.562
```

```
|131.19145921686706|
|[-0.6620789159373044,0.7438786222935815,-
0.9146387152626053,1.310226916228351,0.2212623356598274]      |136.031
|130.32338726794205|
|[-0.8522076782080978,1.4535249637188306,-0.9146387152626053,-
1.2306869183931999,1.3543122495975037]    |119.975           |
123.1597545765295 |
|[0.35194114950692723,2.602476183169233,-
1.1869953767998809,1.310226916228351,0.779365375081242]         |
123.917           |121.40848098229738|
|[0.6688224199582496,-0.5909323532443863,-
0.36992539218805415,1.310226916228351,-0.693773497622163]     |127.054
|128.23114706217712|
|[-0.4085738995762465,-0.5909323532443863,-
0.36992539218805415,0.29642796201066174,-0.6798866407045439] |133.649
|130.97685950945294|
|[-0.12338075617005638,-0.5909323532443863,-0.36992539218805415,-
1.2306869183931999,-0.6568393814532048]|130.392           |
127.39602414569993|
|[-0.6620789159373044,-0.01645674351918489,-0.36992539218805415,-
1.2306869183931999,-0.397388098899776] |132.886           |
127.63531644016642|
|[-0.5194823442342094,1.4873176466438423,-0.36992539218805415,-
0.7237874412843549,3.1673531805960313]   |118.214           |
117.19349340951895|
|[-0.7571432970727011,-
0.8105847922569633,0.9918579154983239,1.310226916228351,-
0.6252364872445779]     |126.486           |129.8612966775906 |
|[-0.28182139139571755,-0.8105847922569633,0.9918579154983239,-
0.7237874412843549,-0.5845567640833963]  |122.797           |
124.68668084832767|
|[0.35194114950692723,-0.8105847922569633,0.9918579154983239,-
1.2306869183931999,-0.5649572866841284]   |120.527           |
121.29590452096448|
+-----------------------------------------------------------------
---------------------------------+------------------
+------------------+
only showing top 20 rows


data_path =
'/resources/labs/authoride/IBMSkillsNetwork+BD0231EN/labs/NASA_airfoil
_noise_cleaned_V3.parquet'

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression

# Initialize Spark session
spark =
```

```
SparkSession.builder.appName("AirfoilNoisePrediction").getOrCreate()

# Define the correct path to the dataset
data_path =
'/resources/labs/authoride/IBMSkillsNetwork+BD0231EN/labs/NASA_airfoil
_noise_cleaned_V3.parquet'

# Load the dataset
try:
    df = spark.read.parquet(data_path)
    print("Dataset loaded successfully.")
except Exception as e:
    raise RuntimeError(f"Failed to load the dataset from the path:
{data_path}. Error: {e}")

# Check the schema to verify data is loaded correctly
df.printSchema()

# Prepare the data
feature_columns = [col for col in df.columns if col !=
'SoundLevelDecibels']  # 'SoundLevelDecibels' is the label
assembler = VectorAssembler(inputCols=feature_columns,
outputCol='features')
assembled_df = assembler.transform(df)

# Split the data into training and test sets
train_df, test_df = assembled_df.randomSplit([0.8, 0.2], seed=42)

# Define and train the model using LinearRegression
lr = LinearRegression(featuresCol='features',
labelCol='SoundLevelDecibels')
model = lr.fit(train_df)

# Make predictions
predictions_df = model.transform(test_df)

# Show some predictions
predictions_df.select("features", "SoundLevelDecibels",
"prediction").show(truncate=False)
```

```
Dataset loaded successfully.
root
 |-- Frequency: integer (nullable = true)
 |-- AngleOfAttack: double (nullable = true)
 |-- ChordLength: double (nullable = true)
 |-- FreeStreamVelocity: double (nullable = true)
 |-- SuctionSideDisplacement: double (nullable = true)
 |-- SoundLevelDecibels: double (nullable = true)
```

```
24/08/15 10:14:24 WARN util.Instrumentation: [8d651047] regParam is
zero, which might cause numerical instability and overfitting.
[Stage 176:>                                                        (0
+ 1) / 1]

+----------------------------------+------------------
+------------------+
|features                          |SoundLevelDecibels|prediction
|
+----------------------------------+------------------
+------------------+
|[200.0,9.9,0.1524,39.6,0.0233328]  |127.315           |
123.41956569238083|
|[200.0,15.4,0.0508,31.7,0.0289853] |119.975           |
123.12618848417782|
|[200.0,15.4,0.0508,39.6,0.0282593] |121.783           |
124.03981197007556|
|[250.0,19.7,0.0508,39.6,0.036484]  |127.224           |
120.96576903188829|
|[315.0,4.0,0.2286,39.6,0.00473801] |122.229           |
125.70834041961699|
|[400.0,8.9,0.1016,71.3,0.0103088]  |138.123           |
130.5462255362437 |
|[500.0,4.0,0.2286,55.5,0.0042862]  |127.314           |
127.1678203722457 |
|[500.0,4.0,0.3048,71.3,0.00497773] |130.715           |
125.93701878579101|
|[500.0,5.4,0.1524,55.5,0.00433288] |129.367           |
129.3158172568494 |
|[500.0,12.7,0.0254,39.6,0.0130253] |127.127           |
127.9172679628786 |
|[630.0,2.0,0.2286,39.6,0.00346574] |127.417           |
126.3426646264615 |
|[630.0,12.3,0.1016,71.3,0.0337792] |135.368           |
125.4315820458994 |
|[630.0,17.4,0.0254,39.6,0.0172206] |124.514           |
125.15232386913738|
|[800.0,0.0,0.1524,55.5,0.00172668] |126.713           |
131.60541688796886|
|[800.0,15.4,0.0508,55.5,0.0271925] |130.43            |
125.06294527623982|
|[1250.0,5.4,0.1524,71.3,0.00401199]|134.111           |
130.0315827376988 |
|[1600.0,0.0,0.3048,39.6,0.00310138]|124.049           |
123.26840208610746|
|[1600.0,3.3,0.1016,55.5,0.002211]  |133.649           |
130.95500711261082|
|[1600.0,4.0,0.2286,31.7,0.00509068]|125.229           |
123.22318272774744|
|[1600.0,6.7,0.1016,31.7,0.00592927]|124.346           |
```

```
126.53788096840135|
+----------------------------------+-----------------
+------------------+
only showing top 20 rows
```

## Task 2 - Print the MSE

```
from pyspark.ml.evaluation import RegressionEvaluator

# Initialize the RegressionEvaluator with the label and prediction
columns
evaluator = RegressionEvaluator(
    labelCol="SoundLevelDecibels",
    predictionCol="prediction",
    metricName="mse"
)

# Calculate the MSE
mse = evaluator.evaluate(predictions_df)

# Print the MSE
print(f"Mean Squared Error (MSE): {mse}")
```

```
[Stage 177:=====================================>                   (5
+ 3) / 8]

Mean Squared Error (MSE): 21.961552275373407
```

## Task 3 - Print the MAE

```
from pyspark.ml.evaluation import RegressionEvaluator

# Initialize the RegressionEvaluator with the label and prediction
columns
evaluator = RegressionEvaluator(
    labelCol="SoundLevelDecibels",
    predictionCol="prediction",
    metricName="mae"
)

# Calculate the MAE
mae = evaluator.evaluate(predictions_df)

# Print the MAE
print(f"Mean Absolute Error (MAE): {mae}")
```

```
[Stage 179:=============================================>             (6
+ 2) / 8]

Mean Absolute Error (MAE): 3.7026698857348803
```

## Task 4 - Print the R-Squared(R2)

```
from pyspark.ml.evaluation import RegressionEvaluator

# Initialize the RegressionEvaluator with the label and prediction
columns
evaluator = RegressionEvaluator(
    labelCol="SoundLevelDecibels",
    predictionCol="prediction",
    metricName="r2"
)

# Calculate the R-Squared (R²)
r2 = evaluator.evaluate(predictions_df)

# Print the R-Squared (R²)
print(f"R-Squared (R²): {r2}")
```

```
[Stage 181:=================================================>       (7
+ 1) / 8]

R-Squared (R²): 0.5504651529219815
```

## Part 3 - Evaluation

**Run the code cell below. Use the answers here to answer the final evaluation quiz in the next section. If the code throws up any errors, go back and review the code you have written.**

```
# Part 3 - Evaluation

print("Part 3 - Evaluation")

# Print the Mean Squared Error (MSE)
print("Mean Squared Error = ", round(mse, 2))

# Print the Mean Absolute Error (MAE)
print("Mean Absolute Error = ", round(mae, 2))

# Print the R-Squared (R²)
print("R Squared = ", round(r2, 2))

# Retrieve the LinearRegression model from the pipeline (last stage)
```

```
lrModel = pipeline_model.stages[-1]

# Print the Intercept of the model
print("Intercept = ", round(lrModel.intercept, 2))
```

```
Part 3 - Evaluation
Mean Squared Error =  21.96
Mean Absolute Error =  3.7
R Squared =  0.55
Intercept =  124.83
```

# Part 4 - Persist the Model

## Task 1 - Save the model to the path "Final_Project"

```
# Define the path where you want to save the model
model_path = "Final_Project"

# Save the trained pipeline model to the specified path
pipeline_model.save(model_path)

print(f"Model saved successfully to the path: {model_path}")
```

```
Model saved successfully to the path: Final_Project
```

## Task 2 - Load the model from the path "Final_Project"

```
from pyspark.ml import PipelineModel

# Define the path where the model was saved
model_path = "Final_Project"

# Load the saved model from the specified path
loaded_model = PipelineModel.load(model_path)

print(f"Model loaded successfully from the path: {model_path}")
```

```
Model loaded successfully from the path: Final_Project
```

## Task 3 - Make predictions using the loaded model on the testdata

```
# Drop the existing "features" column if it already exists
test_df = test_df.drop("features")

# Make predictions using the loaded model
loaded_predictions = loaded_model.transform(test_df)

# Display the predictions along with the actual SoundLevelDecibels
```

```
values
loaded_predictions.select("features", "SoundLevelDecibels",
"prediction").show(truncate=False)
```

```
[Stage 203:>                                                        (0
+ 1) / 1]
```

```
+----------------------------------+------------------
+------------------+
|features                          |SoundLevelDecibels|prediction
|
+----------------------------------+------------------
+------------------+
|[200.0,9.9,0.1524,39.6,0.0233328]  |127.315           |
123.46487909217682|
|[200.0,15.4,0.0508,31.7,0.0289853] |119.975           |
123.1597545765295 |
|[200.0,15.4,0.0508,39.6,0.0282593] |121.783           |
124.05763233246576|
|[250.0,19.7,0.0508,39.6,0.036484]  |127.224           |
120.96823141441659|
|[315.0,4.0,0.2286,39.6,0.00473801] |122.229           |
125.81309126344736|
|[400.0,8.9,0.1016,71.3,0.0103088]  |138.123           |
130.54036705703737|
|[500.0,4.0,0.2286,55.5,0.0042862]  |127.314           |
127.23487410135853|
|[500.0,4.0,0.3048,71.3,0.00497773] |130.715           |
125.98541625784932|
|[500.0,5.4,0.1524,55.5,0.00433288] |129.367           |
129.36676516612042|
|[500.0,12.7,0.0254,39.6,0.0130253] |127.127           |
127.96331791840028|
|[630.0,2.0,0.2286,39.6,0.00346574] |127.417           |
126.4414453261166 |
|[630.0,12.3,0.1016,71.3,0.0337792] |135.368           |
125.3594210655152 |
|[630.0,17.4,0.0254,39.6,0.0172206] |124.514           |
125.19482020429771|
|[800.0,0.0,0.1524,55.5,0.00172668] |126.713           |
131.64657474183315|
|[800.0,15.4,0.0508,55.5,0.0271925] |130.43            |
125.03826725290075|
|[1250.0,5.4,0.1524,71.3,0.00401199]|134.111           |
130.03560763688196|
|[1600.0,0.0,0.3048,39.6,0.00310138]|124.049           |
123.36726803749816|
|[1600.0,3.3,0.1016,55.5,0.002211]  |133.649           |
130.97685950945294|
|[1600.0,4.0,0.2286,31.7,0.00509068]|125.229           |
```

```
123.32420157495756|
|[1600.0,6.7,0.1016,31.7,0.00592927]|124.346           |
126.6106364531364 |
+-----------------------------------+------------------
+------------------+
only showing top 20 rows
```

## Task 4 - Show the predictions

```
# Option 1: If you dropped the existing "features" column
# test_df = test_df.drop("features")

# Option 2: If you renamed the existing "features" column
# test_df = test_df.withColumnRenamed("features", "existing_features")

# Make predictions using the loaded model
loaded_predictions = loaded_model.transform(test_df)

# Display the predictions along with the actual SoundLevelDecibels
values
loaded_predictions.select("features", "SoundLevelDecibels",
"prediction").show(truncate=False)
```

```
[Stage 204:>                                                        (0
+ 1) / 1]

+-----------------------------------+------------------
+------------------+
|features                           |SoundLevelDecibels|prediction
|
+-----------------------------------+------------------
+------------------+
|[200.0,9.9,0.1524,39.6,0.0233328]  |127.315           |
123.46487909217682|
|[200.0,15.4,0.0508,31.7,0.0289853] |119.975           |
123.1597545765295 |
|[200.0,15.4,0.0508,39.6,0.0282593] |121.783           |
124.05763233246576|
|[250.0,19.7,0.0508,39.6,0.036484]  |127.224           |
120.96823141441659|
|[315.0,4.0,0.2286,39.6,0.00473801] |122.229           |
125.81309126344736|
|[400.0,8.9,0.1016,71.3,0.0103088]  |138.123           |
130.54036705703737|
|[500.0,4.0,0.2286,55.5,0.0042862]  |127.314           |
127.23487410135853|
|[500.0,4.0,0.3048,71.3,0.00497773] |130.715           |
```

```
125.98541625784932|
|[500.0,5.4,0.1524,55.5,0.00433288] |129.367           |
129.36676516612042|
|[500.0,12.7,0.0254,39.6,0.0130253] |127.127           |
127.96331791840028|
|[630.0,2.0,0.2286,39.6,0.00346574] |127.417           |
126.4414453261166 |
|[630.0,12.3,0.1016,71.3,0.0337792] |135.368           |
125.3594210655152 |
|[630.0,17.4,0.0254,39.6,0.0172206] |124.514           |
125.19482020429771|
|[800.0,0.0,0.1524,55.5,0.00172668] |126.713           |
131.64657474183315|
|[800.0,15.4,0.0508,55.5,0.0271925] |130.43            |
125.03826725290075|
|[1250.0,5.4,0.1524,71.3,0.00401199]|134.111           |
130.03560763688196|
|[1600.0,0.0,0.3048,39.6,0.00310138]|124.049           |
123.36726803749816|
|[1600.0,3.3,0.1016,55.5,0.002211]  |133.649           |
130.97685950945294|
|[1600.0,4.0,0.2286,31.7,0.00509068]|125.229           |
123.32420157495756|
|[1600.0,6.7,0.1016,31.7,0.00592927]|124.346           |
126.6106364531364 |
+-----------------------------------+------------------
+------------------+
only showing top 20 rows
```

Part 4 - Evaluation

**Run the code cell below. Use the answers here to answer the final evaluation quiz in the next section. If the code throws up any errors, go back and review the code you have written.**

```
print("Part 4 - Evaluation")

# Retrieve the LinearRegression model from the loaded pipeline (last
stage)
loaded_lr_model = loaded_model.stages[-1]

# Get the total number of stages in the loaded pipeline
total_stages = len(loaded_model.stages)

# Get the input columns used by the VectorAssembler (first stage in
the pipeline)
input_columns = loaded_model.stages[0].getInputCols()
```

```
# Print the total number of stages in the pipeline
print("Number of stages in the pipeline =", total_stages)

# Print the coefficients for each input column
for column, coefficient in zip(input_columns,
loaded_lr_model.coefficients):
    print(f"Coefficient for {column} is {round(coefficient, 4)}")
```

```
Part 4 - Evaluation
Number of stages in the pipeline = 3
Coefficient for Frequency is -4.0421
Coefficient for AngleOfAttack is -2.4999
Coefficient for ChordLength is -3.3413
Coefficient for FreeStreamVelocity is 1.5608
Coefficient for SuctionSideDisplacement is -1.9349
```

## Stop Spark Session

```
spark.stop()

# Drop duplicate rows
df_no_duplicates = df.dropDuplicates()

# Count the number of rows after dropping duplicates
rows_after_dropping_duplicates = df_no_duplicates.count()
print(f"Number of rows after dropping duplicates:
{rows_after_dropping_duplicates}")
```

```
---------------------------------------------------------------------------
-----
Py4JJavaError                             Traceback (most recent call
last)
/tmp/ipykernel_2913/1907307843.py in <module>
      3
      4 # Count the number of rows after dropping duplicates
----> 5 rows_after_dropping_duplicates = df_no_duplicates.count()
      6 print(f"Number of rows after dropping duplicates:
{rows_after_dropping_duplicates}")

~/spark-2.4.3/python/pyspark/sql/dataframe.py in count(self)
    520         2
    521         """
--> 522         return int(self._jdf.count())
    523
    524     @ignore_unicode_prefix

~/spark-2.4.3/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py in
__call__(self, *args)
   1255         answer = self.gateway_client.send_command(command)
   1256         return_value = get_return_value(
```

```
-> 1257             answer, self.gateway_client, self.target_id,
self.name)
   1258
   1259         for temp_arg in temp_args:

~/spark-2.4.3/python/pyspark/sql/utils.py in deco(*a, **kw)
     61     def deco(*a, **kw):
     62         try:
---> 63             return f(*a, **kw)
     64         except py4j.protocol.Py4JJavaError as e:
     65             s = e.java_exception.toString()

~/spark-2.4.3/python/lib/py4j-0.10.7-src.zip/py4j/protocol.py in
get_return_value(answer, gateway_client, target_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}
{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling o2131.count.
: org.apache.spark.sql.catalyst.errors.package$TreeNodeException:
execute, tree:
Exchange SinglePartition
+- *(2) HashAggregate(keys=[], functions=[partial_count(1)],
output=[count#1862L])
   +- *(2) HashAggregate(keys=[AngleOfAttack#1625,
FreeStreamVelocity#1627, ChordLength#1626,
SuctionSideDisplacement#1628, SoundLevelDecibels#1629,
Frequency#1624], functions=[], output=[])
      +- Exchange hashpartitioning(AngleOfAttack#1625,
FreeStreamVelocity#1627, ChordLength#1626,
SuctionSideDisplacement#1628, SoundLevelDecibels#1629, Frequency#1624,
200)
         +- *(1) HashAggregate(keys=[AngleOfAttack#1625,
FreeStreamVelocity#1627, ChordLength#1626,
SuctionSideDisplacement#1628, SoundLevelDecibels#1629,
Frequency#1624], functions=[], output=[AngleOfAttack#1625,
FreeStreamVelocity#1627, ChordLength#1626,
SuctionSideDisplacement#1628, SoundLevelDecibels#1629,
Frequency#1624])
            +- *(1) FileScan parquet
[Frequency#1624,AngleOfAttack#1625,ChordLength#1626,FreeStreamVelocity
#1627,SuctionSideDisplacement#1628,SoundLevelDecibels#1629] Batched:
true, Format: Parquet, Location:
InMemoryFileIndex[file:/resources/labs/authoride/IBMSkillsNetwork+BD02
31EN/labs/NASA_airfoil_nois..., PartitionFilters: [], PushedFilters:
[], ReadSchema:
struct<Frequency:int,AngleOfAttack:double,ChordLength:double,FreeStrea
```

```
mVelocity:double,SuctionSid...

      at
org.apache.spark.sql.catalyst.errors.package$.attachTree(package.scala
:56)
      at
org.apache.spark.sql.execution.exchange.ShuffleExchangeExec.doExecute(
ShuffleExchangeExec.scala:119)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:131)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:127)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$executeQuery$1.apply(SparkPlan.scala:155)
      at
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.sc
ala:151)
      at
org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:
152)
      at
org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:127)
      at
org.apache.spark.sql.execution.InputAdapter.inputRDDs(WholeStageCodege
nExec.scala:391)
      at
org.apache.spark.sql.execution.aggregate.HashAggregateExec.inputRDDs(H
ashAggregateExec.scala:151)
      at
org.apache.spark.sql.execution.WholeStageCodegenExec.doExecute(WholeSt
ageCodegenExec.scala:627)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:131)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:127)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$executeQuery$1.apply(SparkPlan.scala:155)
      at
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.sc
ala:151)
      at
org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:
152)
      at
org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:127)
      at
org.apache.spark.sql.execution.SparkPlan.getByteArrayRdd(SparkPlan.sca
la:247)
      at
```

```
org.apache.spark.sql.execution.SparkPlan.executeCollect(SparkPlan.scal
a:296)
      at org.apache.spark.sql.Dataset$
$anonfun$count$1.apply(Dataset.scala:2830)
      at org.apache.spark.sql.Dataset$
$anonfun$count$1.apply(Dataset.scala:2829)
      at org.apache.spark.sql.Dataset$
$anonfun$53.apply(Dataset.scala:3364)
      at org.apache.spark.sql.execution.SQLExecution$
$anonfun$withNewExecutionId$1.apply(SQLExecution.scala:78)
      at
org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQL
Execution.scala:125)
      at
org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExe
cution.scala:73)
      at org.apache.spark.sql.Dataset.withAction(Dataset.scala:3363)
      at org.apache.spark.sql.Dataset.count(Dataset.scala:2829)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.j
ava:62)
      at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccess
orImpl.java:43)
      at java.lang.reflect.Method.invoke(Method.java:498)
      at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
      at
py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
      at py4j.Gateway.invoke(Gateway.java:282)
      at
py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
      at py4j.commands.CallCommand.execute(CallCommand.java:79)
      at py4j.GatewayConnection.run(GatewayConnection.java:238)
      at java.lang.Thread.run(Thread.java:745)
Caused by:
org.apache.spark.sql.catalyst.errors.package$TreeNodeException:
execute, tree:
Exchange hashpartitioning(AngleOfAttack#1625, FreeStreamVelocity#1627,
ChordLength#1626, SuctionSideDisplacement#1628,
SoundLevelDecibels#1629, Frequency#1624, 200)
+- *(1) HashAggregate(keys=[AngleOfAttack#1625,
FreeStreamVelocity#1627, ChordLength#1626,
SuctionSideDisplacement#1628, SoundLevelDecibels#1629,
Frequency#1624], functions=[], output=[AngleOfAttack#1625,
FreeStreamVelocity#1627, ChordLength#1626,
SuctionSideDisplacement#1628, SoundLevelDecibels#1629,
Frequency#1624])
   +- *(1) FileScan parquet
```

```
[Frequency#1624,AngleOfAttack#1625,ChordLength#1626,FreeStreamVelocity
#1627,SuctionSideDisplacement#1628,SoundLevelDecibels#1629] Batched:
true, Format: Parquet, Location:
InMemoryFileIndex[file:/resources/labs/authoride/IBMSkillsNetwork+BD02
31EN/labs/NASA_airfoil_nois..., PartitionFilters: [], PushedFilters:
[], ReadSchema:
struct<Frequency:int,AngleOfAttack:double,ChordLength:double,FreeStrea
mVelocity:double,SuctionSid...

      at
org.apache.spark.sql.catalyst.errors.package$.attachTree(package.scala
:56)
      at
org.apache.spark.sql.execution.exchange.ShuffleExchangeExec.doExecute(
ShuffleExchangeExec.scala:119)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:131)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:127)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$executeQuery$1.apply(SparkPlan.scala:155)
      at
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.sc
ala:151)
      at
org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:
152)
      at
org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:127)
      at
org.apache.spark.sql.execution.InputAdapter.inputRDDs(WholeStageCodege
nExec.scala:391)
      at
org.apache.spark.sql.execution.aggregate.HashAggregateExec.inputRDDs(H
ashAggregateExec.scala:151)
      at
org.apache.spark.sql.execution.aggregate.HashAggregateExec.inputRDDs(H
ashAggregateExec.scala:151)
      at
org.apache.spark.sql.execution.WholeStageCodegenExec.doExecute(WholeSt
ageCodegenExec.scala:627)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:131)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:127)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$executeQuery$1.apply(SparkPlan.scala:155)
      at
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.sc
```

```
ala:151)
	at
org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:
152)
	at
org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:127)
	at
org.apache.spark.sql.execution.exchange.ShuffleExchangeExec.prepareShu
ffleDependency(ShuffleExchangeExec.scala:92)
	at org.apache.spark.sql.execution.exchange.ShuffleExchangeExec$
$anonfun$doExecute$1.apply(ShuffleExchangeExec.scala:128)
	at org.apache.spark.sql.execution.exchange.ShuffleExchangeExec$
$anonfun$doExecute$1.apply(ShuffleExchangeExec.scala:119)
	at
org.apache.spark.sql.catalyst.errors.package$.attachTree(package.scala
:52)
	... 37 more
Caused by: java.lang.IllegalStateException: Cannot call methods on a
stopped SparkContext.
This stopped SparkContext was created at:

org.apache.spark.api.java.JavaSparkContext.<init>(JavaSparkContext.sca
la:58)
sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructo
rAccessorImpl.java:62)
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingCo
nstructorAccessorImpl.java:45)
java.lang.reflect.Constructor.newInstance(Constructor.java:423)
py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:247)
py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
py4j.Gateway.invoke(Gateway.java:238)
py4j.commands.ConstructorCommand.invokeConstructor(ConstructorCommand.
java:80)
py4j.commands.ConstructorCommand.execute(ConstructorCommand.java:69)
py4j.GatewayConnection.run(GatewayConnection.java:238)
java.lang.Thread.run(Thread.java:745)

The currently active SparkContext was created at:

(No active SparkContext.)

	at
org.apache.spark.SparkContext.assertNotStopped(SparkContext.scala:100)
	at
org.apache.spark.SparkContext.broadcast(SparkContext.scala:1486)
	at
org.apache.spark.sql.execution.datasources.parquet.ParquetFileFormat.b
uildReaderWithPartitionValues(ParquetFileFormat.scala:328)
	at
```

```
org.apache.spark.sql.execution.FileSourceScanExec.inputRDD$lzycompute(
DataSourceScanExec.scala:309)
      at
org.apache.spark.sql.execution.FileSourceScanExec.inputRDD(DataSourceS
canExec.scala:305)
      at
org.apache.spark.sql.execution.FileSourceScanExec.inputRDDs(DataSource
ScanExec.scala:327)
      at
org.apache.spark.sql.execution.aggregate.HashAggregateExec.inputRDDs(H
ashAggregateExec.scala:151)
      at
org.apache.spark.sql.execution.WholeStageCodegenExec.doExecute(WholeSt
ageCodegenExec.scala:627)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:131)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$execute$1.apply(SparkPlan.scala:127)
      at org.apache.spark.sql.execution.SparkPlan$
$anonfun$executeQuery$1.apply(SparkPlan.scala:155)
      at
org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.sc
ala:151)
      at
org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:
152)
      at
org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:127)
      at
org.apache.spark.sql.execution.exchange.ShuffleExchangeExec.prepareShu
ffleDependency(ShuffleExchangeExec.scala:92)
      at org.apache.spark.sql.execution.exchange.ShuffleExchangeExec$
$anonfun$doExecute$1.apply(ShuffleExchangeExec.scala:128)
      at org.apache.spark.sql.execution.exchange.ShuffleExchangeExec$
$anonfun$doExecute$1.apply(ShuffleExchangeExec.scala:119)
      at
org.apache.spark.sql.catalyst.errors.package$.attachTree(package.scala
:52)
      ... 58 more
```

# Authors

Ramesh Sannareddy

## Other Contributors

# Change Log

| Date (YYYY-MM-DD) | Version | Changed By | Change Description |
|---|---|---|---|
| 2023-05-26 | 0.1 | Ramesh Sannareddy | Initial Version Created |