# Naya College
## Cloud Data Engineering


# Apache Spark
# Lab4: Anomalies Detection

# Contents

## Lab4-Task1: Identify 300% Average Delay Anomalies

**Objective:**
Identify flights that exhibit an arrival delay which is 300% greater than the historical average delay for its respective carrier.

Tip:
The statement ***"Identify flights that exhibit an arrival delay which is 300% greater than the historical average delay for its respective carrier"*** means that you should find flights whose arrival delay is exceptionally longer than what's typically observed for that particular airline (carrier).

To clarify, let's break it down:

1. **Historical Average Delay for a Carrier**: This refers to the average arrival delay of all flights, up to a certain date, for a specific airline (carrier).
2. **300% Greater than the Historical Average**: This means that the delay of a particular flight should be three times more than this historical average.

**Example**:
Let's take a fictional airline named "FlyHigh" as an example:

1. **Historical Average Delay for FlyHigh**: Over the past year, the average delay for FlyHigh flights has been 10 minutes.

2. **300% Greater than this Average**: A delay that's 300% greater than 10 minutes is $10 minutes + 3 \times 10 = 40 minutes$.

3. **Conclusion**: Any FlyHigh flight that arrives with a delay of more than 40 minutes would meet the criteria and be identified as having an exceptional delay compared to its historical performance.

In simpler terms, if a specific FlyHigh flight had a delay of 45 minutes on a particular day, it would be flagged based on the criteria because its delay is more than three times the historical average for that airline.

**Instructions:**

**Spark Environment Setup:**
**IDE Settings**:

- Set up your project directory with a new folder **exercises_four** to keep your workspace organized.
- Create new python file named average_Delay_Anomalies.

**Spark Session Initialization:**

- Start a new Spark session:
  - Application Name: **ex4_anomalies_detection**
  - Run Mode: Local, utilizing all available cores.
  - Driver Memory: Allocate 4GB for optimized performance.
  - Import libraries:

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql import Window
```

**Define Window Specification:**

- Create a window specification named **all_history_window**. This window will:
  - Partition data by the **Carrier**.
  - Order the data by **flight_date**. This is essential to maintain a chronological order of the flights and compute a rolling average.

**Load Data:**

- **Source Path**: Load the flights data from the S3 path: **s3a://spark/data/transformed/flights/**.
- To optimize subsequent operations, cache the **flights_df** DataFrame in memory.

**Anomaly Detection Logic:**

- Calculate the historical average delay up to the current flight's date for each carrier using the **avg** function over the defined window. Store this value in a new column named **avg_till_now**.
- Determine the percentage difference between the current flight's delay and the historical average. Store this in the **avg_diff_percent** column.
  - The difference is calculated as an absolute value to ensure both positive and negative anomalies are captured.
- Filter the data to retain only the flights where the **avg_diff_percent** is greater than 300% (or 3.0 as a decimal fraction).

**Display Results:**

- Use the **show** method to display the records that exhibit significant deviation from the historical average.

**Clean Up:**

- Unpersist the **flights_df** DataFrame to release memory cache.
- Terminate the Spark session, ensuring a graceful release of all associated resources.

**Note**: The code assumes no null values in the **arr_delay** column. If nulls exist, additional preprocessing might be required to handle or filter them out. Always consider edge cases like zero average delay, which could lead to division-by-zero issues.

## Lab4-Task2: Detect 500% Carrier Delay Outliers

**Objective:**
Determine flights that have an arrival delay which is 500% greater than the overall average delay for its respective carrier.

Tip:
This task aims to identify flights that are exceptionally delayed when compared to the overall average delays of their respective carriers. To clarify:

1. **Overall Average Delay for a Carrier:** This represents the average arrival delay across all recorded flights for a specific airline (carrier).
2. **500% Greater than the Overall Average:** The delay of a particular flight should be five times more than this overall average.

**Example:**
Assume "Skyways" as a fictional airline:

1. **Overall Average Delay for Skyways:** Throughout its entire operation, Skyways flights have had an average delay of 5 minutes.
2. **500% Greater than this Average:** A delay that's 500% greater than 5 minutes is $5+5\times5 = 30$ minutes.
3. **Conclusion:** Any Skyways flight with a delay more than 30 minutes would be highlighted based on the criteria as it's exceptionally long compared to the carrier's general trend.

In simple words:

- The **current_window** computes values based on the entire set of data for each carrier, without considering the order of dates. Think of it as calculating values based on all-time data for each carrier.
- The **Previous_window**, on the other hand, computes values in a sequential manner, considering the order of flight dates. For each date, it looks at all the prior data up to that date to compute values. Think of it as calculating a running total or rolling average for each carrier based on historical data up to each flight date.

**Instructions:**

**Spark Environment Setup:**

**IDE Settings:**

- Organize your workspace by creating a folder named **exercises_four**.
- Within this folder, create a new Python file named **average_Delay_Anomalies_02**.

**Spark Session Initialization:**

- Start a new Spark session:
  - Application Name: **ex4_anomalies_detection**
  - Run Mode: Local
  - Driver Memory: Assign 4GB to optimize operations.
  - Import libraries:

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql import Window
```

**Define Window Specification:**

- Configure an **unbounded_window** specification that:
  - Partitions data by the **Carrier**.
  - Considers all rows within the partition with **.rowsBetween(Window.unboundedPreceding, Window.unboundedFollowing)**. This will compute the average over all rows, providing an all-time average.

**Load Data:**

- **Source Path:** Load the flights data from the S3 directory: **s3a://spark/data/transformed/flights/**
- Cache **flights_df** DataFrame to enhance performance for subsequent operations.

**Anomaly Detection Logic:**

- Calculate the overall average delay for each carrier using the **avg** function over the defined window. Save this value in a new column, **avg_all_time**.
- Compute the percentage difference between a flight's delay and the overall average. Store this in **avg_diff_percent**.
- Filter to keep only records where **avg_diff_percent** is greater than 500% (or 5.0 when represented as a decimal).

**Display Results:**

- Display anomalous records with the **show()** method.

**Cleanup Operations:**

- Release the cache for **flights_df** to free up memory.
- Terminate the Spark session gracefully to release all resources.

**Note:** Ensure that there are no null values in the **arr_delay** column. If null values exist, additional preprocessing may be necessary. Also, consider potential edge cases, such as a zero average delay, which could lead to division-by-zero scenarios.

## Lab4-Task3: Detect 30% Delay Change by Carrier

**Objective:**

Detect significant changes in flight delays for each carrier using a sliding window analysis. Specifically, identify instances where there's a 30% difference in total delays between two consecutive 10-day windows.

**Tip**:

A sliding window is like a "moving average" but over a fixed period. Here, for each day, we're analyzing the total delays of the previous 10 days and comparing it to the 10 days that follow. A 30% change indicates a substantial difference in performance or conditions that might require attention.

**Another tip** 😊:
Imagine you're tracking the delays of flights for a specific airline, and you want to see how these delays change over time. Instead of looking at every single day on its own, you decide to look at chunks of 10 days at a time because it provides a more holistic view of the performance.
**Simple Example**:

1. Let's consider you have data for a month:
    - Days: 1 to 30
2. You start with the first 10 days as your first "window":
    - Window 1: Days 1 to 10
3. On the next day, you shift or "slide" this window by one day to get:
    - Window 2: Days 2 to 11
4. You continue this for the entire month. So, after Window 2, you'll have:
    - Window 3: Days 3 to 12
    - ... and so on.
5. Now, you want to see if there's a significant change in delays from one window to the next. Let's say, in our example:
    - Total delay in Window 1: 100 minutes
    - Total delay in Window 2: 140 minutes (a 40% increase from Window 1)

Since the increase is more than 30%, you flag this as a significant change.
By analyzing data this way, you can spot trends, spikes, or drops in flight delays over time.

**Instructions:**

**Setting Up Your Spark Environment:**

**IDE Configuration:**

- Set up your project workspace with the folder named **exercises_four**.
- Inside this folder, create a new python file: **anomalies_detection_03**.

**Initiate Spark Session:**

- Application Name: **ex4_anomalies_detection**.
- Running Mode: Local mode using all available cores.
- Driver Memory Allocation: 4GB
- Import libraries:

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql import Window
```

**Define the Sliding Window Specification:**

- Create a window specification named **sliding_range_window**. This window will:
  - Partition the data by the **Carrier**.
  - Order the data by the **start_range**, which will later be defined as the start date of each 10-day window.

**Load and Cache Data:**

- **Source Data Path**: **s3a://spark/data/transformed/flights/**.
- Load this data into the **flights_df** DataFrame.
- To ensure optimal performance for subsequent operations, cache **flights_df**.

**Sliding Window Analysis:**

- Group by each **Carrier** and a 10-day window that slides by 1 day for each **flight_date**.
- For each of these groups, calculate the total delay as the sum of **departure and arrival delays**.
- Rename the resulting columns for better clarity and understanding:
  - **date_window.start** -> **start_range**
  - **date_window.end** -> **end_range**
- Create a new column, **last_window_delay**, to store the delay value from the previous 10-day window.
- Calculate the percentage change between consecutive windows and store this in **change_percent**.
- Filter results to only display entries with more than a 30% change.

**Display and Clean Up:**

- Display the filtered results using the **show** method, limited to the first 100 rows for a concise sample.
- Release the cache for **flights_df**.
- Finally, ensure a clean end to your Spark session by stopping it.