DataFrame ด้วย PySpark และ Pandas

```
df.show(5)  # แสดง 5 แถวแรก
```

```
df.printSchema()  # แสดงโครงสร้างของ DataFrame
```

```
df.select("name", "age").show()
```

```
df.filter(df.age > 30).show()  # แสดงข้อมูลที่มีอายุมากกว่า 30
```

```
df.groupBy("age").count().show()  # นับจำนวนของแต่ละกลุ่มอายุ
```

```
import pandas as pd
from pyspark.sql import SparkSession

# สร้าง Pandas DataFrame
pandas_df = pd.DataFrame({
    "name": ["Alice", "Bob", "Charlie"],
    "age": [25, 30, 35]
})

# แปลง Pandas DataFrame ไปเป็น PySpark DataFrame
spark_df = spark.createDataFrame(pandas_df)
spark_df.show()
```

```
from pyspark.sql.functions import col

# เพิ่มคอลัมน์ใหม่ที่ชื่อว่า 'age_plus_10' ซึ่งเพิ่มอายุขึ้น 10 ปี
df = df.withColumn("age_plus_10", col("age") + 10)
df.show()
```

```
+-------+---+-----------+
|   name|age|age_plus_10|
+-------+---+-----------+
|  Alice| 25|         35|
|    Bob| 30|         40|
|Charlie| 35|         45|
+-------+---+-----------+
```

```python
from pyspark.sql.functions import when

# เพิ่มคอลัมน์ 'age_group' เพื่อกำหนดกลุ่มอายุ
df = df.withColumn("age_group", when(col("age") < 30, "Young")
                   .when((col("age") >= 30) & (col("age") < 40), "Middle-aged")
                   .otherwise("Old"))
df.show()
```

```
+-------+---+-----------+-----------+
|   name|age|age_plus_10|  age_group|
+-------+---+-----------+-----------+
|  Alice| 25|         35|      Young|
|    Bob| 30|         40|Middle-aged|
|Charlie| 35|         45|Middle-aged|
+-------+---+-----------+-----------+
```

```python
from pyspark.sql.functions import avg, count, max, min

# หาอายุเฉลี่ย
df_avg = df.agg(avg("age").alias("average_age"))
df_avg.show()

# นับจำนวนคนในแต่ละกลุ่มอายุ
df_group_count = df.groupBy("age_group").agg(count("*").alias("count"))
df_group_count.show()
```

```
+-----------+
|average_age|
+-----------+
|       30.0|
+-----------+
```

```python
# บันทึกเป็น CSV
df.write.csv("path/to/output.csv", header=True)

# บันทึกเป็น Parquet
df.write.parquet("path/to/output.parquet")
```

Graph Analytics PowerBI

```python
from pyspark.sql import SparkSession
from graphframes import GraphFrame
from pyspark.sql.functions import desc, col, lit

spark = SparkSession.builder \
    .appName("CyclingRoutesGraph") \
    .config("spark.jars.packages", "graphframes:graphframes:0.8.2-spark3.0-s_2.12") \
    .getOrCreate()

cycling_routes_df = spark.read.csv("cycling.csv", header=True, inferSchema=True)

vertices = cycling_routes_df.select("FromStationName").withColumnRenamed("FromStationName", "id").distinct()

edges = cycling_routes_df.select("FromStationName", "ToStationName") \
    .withColumnRenamed("FromStationName", "src") \
    .withColumnRenamed("ToStationName", "dst")

graph = GraphFrame(vertices, edges)

grouped_edges = graph.edges.groupBy("src", "dst").count() \
    .filter(col("count") >= 3) \
    .withColumn("source_color", lit("#FF3F33")) \
    .withColumn("destination_color", lit("#3358FF"))

grouped_edges.select("src", "dst", "source_color", "destination_color") \
    .write.csv("grouped_cycling_routes_no_header.csv", mode="overwrite", header=False)

grouped_edges.show()

print("Processing and writing to file completed.")
```

```
warnings.warn(
+--------------------+--------------------+-----+------------+-----------------+
|                 src|                 dst|count|source_color|destination_color|
+--------------------+--------------------+-----+------------+-----------------+
|  Union St & 4th Ave|  3rd Ave & Broad St|    8|     #FF3F33|          #3358FF|
|   2nd Ave & Pine St|Occidental Park /...|   34|     #FF3F33|          #3358FF|
|NE 42nd St & Univ...|UW Intramural Act...|    6|     #FF3F33|          #3358FF|
|15th Ave E & E Th...|NE Pacific St/UW ...|    3|     #FF3F33|          #3358FF|
|Dexter Ave N & Al...|6th Ave & Blancha...|    8|     #FF3F33|          #3358FF|
|Pier 69 / Alaskan...|  3rd Ave & Broad St|    9|     #FF3F33|          #3358FF|
|15th Ave E & E Th...|Thomas St & 5th A...|    3|     #FF3F33|          #3358FF|
|REI / Yale Ave N ...|Dexter Ave N & Al...|   15|     #FF3F33|          #3358FF|
|12th Ave & E Denn...|Occidental Park /...|    4|     #FF3F33|          #3358FF|
|E Blaine St & Fai...|15th Ave NE & NE ...|    3|     #FF3F33|          #3358FF|
|15th Ave NE & NE ...|Eastlake Ave E & ...|    7|     #FF3F33|          #3358FF|
|   2nd Ave & Vine St|Key Arena / 1st A...|   12|     #FF3F33|          #3358FF|
|E Pine St & 16th Ave|Bellevue Ave & E ...|   15|     #FF3F33|          #3358FF|
|PATH / 9th Ave & ...|  Union St & 4th Ave|   16|     #FF3F33|          #3358FF|
|King Street Stati...|   Pine St & 9th Ave|    6|     #FF3F33|          #3358FF|
|Thomas St & 5th A...|  Union St & 4th Ave|    4|     #FF3F33|          #3358FF|
|Summit Ave & E De...|  Union St & 4th Ave|    6|     #FF3F33|          #3358FF|
|E Harrison St & B...|Broadway and E De...|   13|     #FF3F33|          #3358FF|
|Cal Anderson Park...|Broadway and E De...|    6|     #FF3F33|          #3358FF|
|2nd Ave & Blancha...|Thomas St & 5th A...|    9|     #FF3F33|          #3358FF|
+--------------------+--------------------+-----+------------+-----------------+
only showing top 20 rows
```

Output CSV

A	B	C	D
Union St &	3rd Ave & E	#FF3F33	#3358FF
2nd Ave & I	Occidenta	#FF3F33	#3358FF
NE 42nd St	UW Intram	#FF3F33	#3358FF
15th Ave E	NE Pacific	#FF3F33	#3358FF
Dexter Ave	6th Ave & E	#FF3F33	#3358FF
Pier 69 / Al	3rd Ave & E	#FF3F33	#3358FF
15th Ave E	Thomas St	#FF3F33	#3358FF
REI / Yale A	Dexter Ave	#FF3F33	#3358FF
12th Ave &	Occidenta	#FF3F33	#3358FF
E Blaine St	15th Ave N	#FF3F33	#3358FF
15th Ave N	Eastlake A	#FF3F33	#3358FF
2nd Ave &	Key Arena	#FF3F33	#3358FF
E Pine St &	Bellevue A	#FF3F33	#3358FF
PATH / 9th	Union St &	#FF3F33	#3358FF

Graph Analytics

```python
import os
os.environ["JAVA_HOME"] = r"C:\Program Files\Java\jdk-22"
os.environ["SPARK_HOME"] = r"C:\spark\spark-3.5.1-bin-hadoop3"

from pyspark.sql import SparkSession
from graphframes import GraphFrame
from pyspark.sql.functions import desc

# สร้าง SparkSession สำหรับใช้งาน PySpark
spark = SparkSession.builder \
    .appName("Graph Analytics") \
    .config("spark.jars.packages", "graphframes:graphframes:0.8.2-spark3.0-s_2.12") \
    .config("spark.executor.memory", "4g") \
    .config("spark.driver.memory", "4g") \
    .config("spark.pyspark.python", "python") \
    .config("spark.pyspark.driver.python", "python") \
    .getOrCreate()  # สร้างหรือดึง SparkSession

# ข้อมูลเวอร์เท็กซ์ (จุดยอด) ที่มีชื่อและอายุ
vertices_data = [
    ("Alice", 45),
    ("Jacob", 43),
    ("Roy", 21),
    ("Ryan", 49),
    ("Emily", 24),
    ("Sheldon", 52)
]
# ข้อมูลเอดจ์ (เชื่อมโยง) ที่มีความสัมพันธ์ระหว่างเวอร์เท็กซ์
edges_data = [
    ("Sheldon", "Alice", "Sister"),
    ("Alice", "Jacob", "Husband"),
    ("Emily", "Jacob", "Father"),
    ("Ryan", "Alice", "Friend"),
    ("Alice", "Emily", "Daughter"),
    ("Alice", "Roy", "Son"),
    ("Jacob", "Roy", "Son")
]

```

```python
# สร้าง DataFrame สำหรับเวอร์เท็กซ์
vertices_df = spark.createDataFrame(vertices_data, ["id", "age"])
# สร้าง DataFrame สำหรับเอดจ์
edges_df = spark.createDataFrame(edges_data, ["src", "dst", "relationship"])

# สร้างกราฟจาก DataFrame ของเวอร์เท็กซ์และเอดจ์
graph = GraphFrame(vertices_df, edges_df)

# แสดงเอดจ์ที่จัดกลุ่มและเรียงตามจำนวน
print("Grouped and Ordered Edges:")
graph.edges.groupBy("src", "dst").count().orderBy(desc("count")).show(truncate=False)

# แสดงเอดจ์ที่เกี่ยวข้องกับ 'Alice'
print("Filtered Edges where src or dst is 'Alice':")
graph.edges.where("src = 'Alice' OR dst = 'Alice'").groupBy("src", "dst").count().orderBy(desc("count")).show(truncate=False)

# สร้างซับกราฟที่มี 'Alice' เป็นส่วนหนึ่ง
print("Subgraph where 'Alice' is involved:")
subgraph_edges = graph.edges.where("src = 'Alice' OR dst = 'Alice'")
subgraph = GraphFrame(graph.vertices, subgraph_edges)
subgraph.edges.show(truncate=False)

```

```python
# แสดง motifs ในกราฟที่เกี่ยวข้องกับ 'Alice'
print("Motifs in the Graph (connections involving Alice):")
motifs = graph.find("(a)-[ab]->(b)")
motifs_filtered = motifs.filter("ab.relationship = 'Friend' OR ab.relationship = 'Daughter'")
motifs_filtered.show(truncate=False)

```

```python
# คำนวณ PageRank
print("PageRank Results:")
page_rank = graph.pageRank(resetProbability=0.15, maxIter=5)
page_rank.vertices.orderBy(desc("pagerank")).show(truncate=False)

# แสดง In-Degree ของแต่ละเวอร์เท็กซ์
print("In-Degree of Each Vertex:")
in_degree = graph.inDegrees
in_degree.orderBy(desc("inDegree")).show(truncate=False)

# ตั้งค่าพ้อยเช็คพ้อยสำหรับการประมวลผล
spark.sparkContext.setCheckpointDir("/tmp/checkpoints")
# แสดง Connected Components
print("Connected Components:")
connected_components = graph.connectedComponents()
connected_components.show(truncate=False)

# แสดง Strongly Connected Components
print("Strongly Connected Components:")
strongly_connected_components = graph.stronglyConnectedComponents(maxIter=5)
strongly_connected_components.show(truncate=False)

# ค้นหาเส้นทางจาก 'Alice' ไป 'Roy' โดยใช้ BFS
print("Breadth-First Search (BFS):")
bfs_results = graph.bfs(fromExpr="id = 'Alice'", toExpr="id = 'Roy'", maxPathLength=2)
bfs_results.show(truncate=False)
```

```python

# สรุปว่าโค้ดนี้ทำการวิเคราะห์กราฟด้วย GraphFrames โดยเริ่มจากการสร้างกราฟจากข้อมูลเวอร์เท็กซ์และเอดจ์ แล้วดำเนินการวิเคราะห์ต่างๆ
# เช่น การค้นหาเอดจ์ที่เกี่ยวข้องกับ 'Alice', คำนวณ PageRank, และหาส่วนประกอบที่เชื่อมโยงกันในกราฟ
print('END')

# ปิด Spark session
spark.stop()
```

```
Grouped and Ordered Edges:
+-------+-----+-----+
|src    |dst  |count|
+-------+-----+-----+
|Alice  |Jacob|1    |
|Sheldon|Alice|1    |
|Emily  |Jacob|1    |
|Alice  |Emily|1    |
|Alice  |Roy  |1    |
|Jacob  |Roy  |1    |
|Ryan   |Alice|1    |
+-------+-----+-----+

Filtered Edges where src or dst is 'Alice':
+-------+-----+-----+
|src    |dst  |count|
+-------+-----+-----+
|Alice  |Jacob|1    |
|Sheldon|Alice|1    |
|Alice  |Emily|1    |
|Alice  |Roy  |1    |
|Ryan   |Alice|1    |
+-------+-----+-----+
```

```
Subgraph where 'Alice' is involved:
+-------+-----+------------+
|src    |dst  |relationship|
+-------+-----+------------+
|Sheldon|Alice|Sister      |
|Alice  |Jacob|Husband     |
|Ryan   |Alice|Friend      |
|Alice  |Emily|Daughter    |
|Alice  |Roy  |Son         |
+-------+-----+------------+

Motifs in the Graph (connections involving Alice):
/usr/local/lib/python3.10/dist-packages/pyspark/sql/dataframe.py:147: UserWarni
  warnings.warn("DataFrame constructor is internal. Do not directly use it.")
+-----------+-----------------------+-----------+
|a          |ab                     |b          |
+-----------+-----------------------+-----------+
|{Ryan, 49} |{Ryan, Alice, Friend}  |{Alice, 45}|
|{Alice, 45}|{Alice, Emily, Daughter}|{Emily, 24}|
+-----------+-----------------------+-----------+

PageRank Results:
+-------+---+------------------+
|id     |age|pagerank          |
+-------+---+------------------+
|Roy    |21 |1.9089989375092518|
|Jacob  |43 |1.3728466605994618|
|Alice  |45 |1.135192093597289 |
|Emily  |24 |0.7420792759997091|
|Sheldon|52 |0.42044151614714403|
|Ryan   |49 |0.42044151614714403|
+-------+---+------------------+
```

```
In-Degree of Each Vertex:
+-----+--------+
|id   |inDegree|
+-----+--------+
|Jacob|2       |
|Alice|2       |
|Roy  |2       |
|Emily|1       |
+-----+--------+

Connected Components:
+-------+---+------------+
|id     |age|component   |
+-------+---+------------+
|Alice  |45 |197568495616|
|Jacob  |43 |197568495616|
|Roy    |21 |197568495616|
|Ryan   |49 |197568495616|
|Emily  |24 |197568495616|
|Sheldon|52 |197568495616|
+-------+---+------------+

Strongly Connected Components
+-------+---+-------------+
|id     |age|component    |
+-------+---+-------------+
|Roy    |21 |197568495616 |
|Jacob  |43 |395136991232 |
|Sheldon|52 |730144440320 |
|Emily  |24 |1022202216448|
|Alice  |45 |1065151889408|
|Ryan   |49 |1477468749824|
+-------+---+-------------+
```

```
Breadth-First Search (BFS):
+-----------+-----------------+---------+
|from       |e0               |to       |
+-----------+-----------------+---------+
|{Alice, 45}|{Alice, Roy, Son}|{Roy, 21}|
+-----------+-----------------+---------+
```

คำสั่ง groupBy("src", "dst").count() ใช้ในการนับจำนวนการเชื่อมต่อระหว่างสนามบินต้นทางและปลายทาง
มีการใช้คำสั่ง .filter(col("count") > 5) เพื่อกรองเส้นทางที่มีการบินซ้ำมากกว่า 5 ครั้ง
คำสั่ง .orderBy(desc("count")) ทำการเรียงลำดับจากมากไปน้อย
ส่วนสี (source_color, destination_color) ถูกเพิ่มเข้าไปใน DataFrame เพื่อใช้ในงานแสดงผล เช่น การวิเคราะห์กราฟขั้นสูง หรือการ visualizing กราฟ.

ASC เรียงจากน้อยไปหามาก
DESC เรียงจากมากไปหาน้อย
.orderBY เรียง

.filter(col("distance") > 10) # ตัวอย่างกรองระยะทางมากกว่า 10 กิโลเมตร
.filter(col("trip_date").between('2023-01-01', '2023-12-31')) # ตัวอย่างกรองการเดินทางในปี 2023
.filter(col("src") != col("dst")) # ตัวอย่างกรองเส้นทางที่มีจุดเริ่มต้นและปลายทางไม่ซ้ำกัน
hub_stations = graph.edges.groupBy("src").count().orderBy(desc("count"))
# ตัวอย่างหาสถานีที่มีการออกเดินทางมากที่สุด
.filter(col("trip_type") == "leisure") # ตัวอย่างกรองเฉพาะการเดินทางเพื่อพักผ่อน

```
grouped_edges = graph.edges.groupBy("src", "dst").count()
.filter(col("count") > 5)
.filter(col("src") != col("dst"))  # กรองเฉพาะเส้นทางที่ไม่มีการย้อนกลับ
.filter(col("distance") > 10)  # กรองเฉพาะเส้นทางที่มีระยะทางมากกว่า 10 กิโลเมตร
.orderBy(desc("count"))
.withColumn("source_color", lit("#3358FF"))
.withColumn("destination_color", lit("#FF3F33"))

หาสถานีที่มีการเดินทางออกจากมากที่สุด
popular_start_stations = graph.edges.groupBy("src").count().orderBy(desc("count"))
หาสถานีที่มีการเดินทางเข้ามามากที่สุด
popular_end_stations = graph.edges.groupBy("dst").count().orderBy(desc("count"))

popular_start_stations.show()
popular_end_stations.show()

popular_routes = graph.edges.groupBy("src","dst").count().orderBy(desc("count"))

หาสถานีที่มีเส้นทางเชื่อมโยงกับหลายสถานีที่สุด
station_connectivity = graph.degrees.orderBy(desc("degree"))
```

การ Join Table แบบ Pandas Output เอาเป็นไฟล์ csv

```python
import pandas as pd

# Load the two uploaded CSV files
file1_path = 'fb_live_thailand2.csv'
file2_path = 'fb_live_thailand3.csv'

df1 = pd.read_csv(file1_path)
df2 = pd.read_csv(file2_path)

# Performing an outer join on the dataframes
# Assuming default join on common columns
outer_joined_df = pd.merge(df1, df2, how='outer') # how='outer,inner,left,right'

# Display the joined data to the user
print(outer_joined_df)

output_path = 'outer_joined_fb_live_thailand.csv' # สร้างไฟล์ใหม่
outer_joined_df.to_csv(output_path, index=False)

```

ผลลัพธ์ index=False

	A	B	C	D	E	F	G
1	Table	status_id	status_type	status_pub	num_react	num_comr	num_share
2	FB2	24667554	photo	########	145	9	
3	FB2	24667554	video	########	90	78	
4	FB2	24667554	video	########	75	36	
5	FB2	24667554	video	########	37	0	
6	FB2	24667554	photo	########	102	0	
7	FB2	24667554	video	########	18	0	
8	FB2	24667554	photo	########	98	0	
9	FB2	24667554	photo	########	227	7	
10	FB2	24667554	photo	########	234	15	
11	FB2	24667554	photo	########	152	2	
12	FB2	24667554	photo	########	203	1	

ผลลัพธ์ index=True

	A	B	C	D	E	F	G	H
1		Table	status_id	status_type	status_pub	num_react	num_comr	num_share
2	0	FB2	24667554	photo	########	145	9	
3	1	FB2	24667554	video	########	90	78	
4	2	FB2	24667554	video	########	75	36	
5	3	FB2	24667554	video	########	37	0	
6	4	FB2	24667554	photo	########	102	0	
7	5	FB2	24667554	video	########	18	0	
8	6	FB2	24667554	photo	########	98	0	
9	7	FB2	24667554	photo	########	227	7	
10	8	FB2	24667554	photo	########	234	15	
11	9	FB2	24667554	photo	########	152	2	

การ Join Table แบบ PySpark ไม่มี Output csv

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import trim, col  # ใช้ในการตัดช่องว่างและเลือกคอลัมน์
from pyspark.sql.types import IntegerType  # ใช้สำหรับแปลงข้อมูลให้เป็นชนิด Integer

# สร้าง SparkSession
spark = SparkSession.builder.appName("OuterJoinFillNA").getOrCreate()

# โหลดข้อมูลจากไฟล์ CSV
file_path_1 = "fb_live_thailand2.csv"
file_path_2 = "fb_live_thailand3.csv"

df1 = spark.read.csv(file_path_1, header=True, inferSchema=True)
df2 = spark.read.csv(file_path_2, header=True, inferSchema=True)

join_key = "status_id"  # แทนที่ด้วยชื่อคอลัมน์ที่ใช้เป็น key

data = df1.join(df2, on=join_key, how="outer") # inner , left , right , outer

print(data)

fill_data = data.fillna({"num_comments": 0},{"num_shares": 0})

# แสดงข้อมูลที่เติมค่า null เป็น 0
fill_data.show()

```

PySpark จัดการ Datframe โดยใช้ภาษา SQL

```python
# Import necessary libraries
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("DataFrame SQL Example") \
    .getOrCreate()

# Sample data and schema
data = [("John", 28), ("Jane", 30), ("Jake", 25)]
columns = ["Name", "Age"]

# Create DataFrame
df = spark.createDataFrame(data, columns)

# Register DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

# Run SQL query
result = spark.sql("SELECT Name, Age FROM people WHERE Age > 25")

# Show the result
result.show()

# Stop the SparkSession
spark.stop()
```

ตัวอย่างการใช้ต่าง ๆ

```python
from pyspark.sql import SparkSession

# Initialize a Spark session
spark = SparkSession.builder.master("local").appName("ReactionAnalysis").getOrCreate()

# Load the CSV files
file_path_1 = 'fb_live_thailand3.csv'
file_path_2 = 'fb_live_thailand2.csv'

# Read the data with header inference
df1 = spark.read.option("header", "true").csv(file_path_1)
df2 = spark.read.option("header", "true").csv(file_path_2)
```

```python
combined_df = df1.union(df2)
combined_df.createOrReplaceTempView("reactions")
```

DISTINCT เป็นคำสั่งใน SQL (รวมถึง PySpark SQL) ที่ใช้ในการกรองข้อมูลเพื่อให้ได้เฉพาะค่าที่ไม่ซ้ำกันในคอลัมน์หรือชุดของคอลัมน์ที่ระบุ ซึ่งมีประโยชน์มากเมื่อคุณต้องการทราบค่าที่แตกต่างกันที่ปรากฏอยู่ในคอลัมน์ใดคอลัมน์หนึ่ง โดยไม่ต้องการให้มีค่าซ้ำกัน

+ Code + Markdown

```python
result = spark.sql("SELECT DISTINCT num_reactions FROM reactions")
result.show()
```

Python

```
+-------------+
|num_reactions|
+-------------+
|          203|
|          221|
|          503|
|          227|
|          111|
|          135|
|          213|
|          150|
|          170|
|          217|
|          209|
|          313|
```

```python
# Filter for num_reactions where the value is '262'
result = spark.sql("SELECT * FROM reactions WHERE status_type = 'video'")
result.show()
```

```
+-----+--------------------+-----------+----------------+-------------+-----------+
|Table|           status_id|status_type|status_published|num_reactions|num_shares|
+-----+--------------------+-----------+----------------+-------------+-----------+
|  FB3|246675545449582_1...|      video|  4/22/2018 6:00|          529|        262|
|  FB3|246675545449582_1...|      video|  4/21/2018 6:17|          227|         57|
|  FB3|246675545449582_1...|      video|  4/18/2018 0:24|          503|         72|
|  FB3|246675545449582_1...|      video|  4/17/2018 7:42|          295|         53|
|  FB3|246675545449582_1...|      video|   4/1/2018 5:16|          332|         30|
|  FB3|246675545449582_1...|      video|  3/30/2018 8:28|          135|         79|
|  FB3|246675545449582_1...|      video|  3/26/2018 8:28|          150|         47|
|  FB3|246675545449582_1...|      video|  3/23/2018 7:09|          221|         36|
|  FB2|246675545449582_1...|      video|  4/22/2018 6:00|          529|        512|
|  FB2|246675545449582_1...|      video|  4/21/2018 6:17|          227|        236|
|  FB2|246675545449582_1...|      video|  4/18/2018 0:24|          503|        614|
|  FB2|246675545449582_1...|      video|  4/17/2018 7:42|          295|        453|
|  FB2|246675545449582_1...|      video|  3/20/2018 1:28|           18|          0|
|  FB2|246675545449582_1...|      video| 3/19/2018 22:34|           37|          0|
|  FB2|246675545449582_1...|      video|  3/17/2018 8:07|           75|         36|
|  FB2|246675545449582_1...|      video|  3/17/2018 7:47|           90|         78|
+-----+--------------------+-----------+----------------+-------------+-----------+
```

GROUP BY เป็นคำสั่งที่ใช้ใน SQL (รวมถึง PySpark SQL) เพื่อจัดกลุ่มข้อมูลตามค่าที่เหมือนกันในคอลัมน์ที่กำหนด ซึ่งจะช่วยให้เราสามารถสรุปข้อมูลได้สะดวกมากขึ้น เช่น การนับจำนวน การหาค่าเฉลี่ย หรือหาค่าสูงสุดต่ำสุดในแต่ละกลุ่ม

```python
result = spark.sql("SELECT num_reactions, COUNT(*) as count FROM reactions GROUP BY num_reactions")
result.show()
```

Python

```
+-------------+-----+
|num_reactions|count|
+-------------+-----+
|          203|    2|
|          221|    1|
|          503|    2|
|          227|    3|
|          111|    2|
|          135|    1|
|          213|    2|
|          150|    3|
|          170|    1|
|          217|    2|
```

**ORDER BY **ใช้ในการจัดเรียงข้อมูลตามลำดับที่ต้องการ เช่นเรียงจากน้อยไปมากหรือมากไปน้อย ASC: เรียงจากน้อยไปมาก (ค่าเริ่มต้น) DESC: เรียงจากมากไปน้อย

```
result = spark.sql("SELECT * FROM reactions ORDER BY num_reactions DESC")
result.show()
```

```
+-----+--------------------+-----------+----------------+-------------+----------+
|Table|           status_id|status_type|status_published|num_reactions|num_shares|
+-----+--------------------+-----------+----------------+-------------+----------+
|  FB2|246675545449582_1...|      photo|  3/20/2018 1:54|           98|         0|
|  FB2|246675545449582_1...|      video|  3/17/2018 7:47|           90|        78|
|  FB2|246675545449582_1...|      video|  3/17/2018 8:07|           75|        36|
|  FB3|246675545449582_1...|      video|  4/22/2018 6:00|          529|       262|
|  FB2|246675545449582_1...|      video|  4/22/2018 6:00|          529|       512|
|  FB3|246675545449582_1...|      video|  4/18/2018 0:24|          503|        72|
|  FB2|246675545449582_1...|      video|  4/18/2018 0:24|          503|       614|
|  FB2|246675545449582_1...|      video| 3/19/2018 22:34|           37|         0|
|  FB3|246675545449582_1...|      photo|   4/5/2018 9:23|          346|         0|
```

HAVING ใช้กับ GROUP BY เพื่อกรองผลลัพธ์หลังจากทำการสรุปข้อมูลแล้ว โดยมีลักษณะการใช้งานคล้ายกับ WHERE แต่ใช้กับกลุ่มข้อมูลที่สรุปแล้ว

```
result = spark.sql("SELECT num_reactions, COUNT(*) as count FROM reactions GROUP BY num_reactions HAVING count > 1")
result.show()
```

Python

```
+-------------+-----+
|num_reactions|count|
+-------------+-----+
|          203|    2|
|          503|    2|
|          227|    3|
|          111|    2|
|          213|    2|
|          150|    3|
|          217|    2|
|          295|    2|
|          529|    2|
+-------------+-----+
```

Fillter PySpark

```
from pyspark.sql import SparkSession

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

data = [(1, "Alice", 3000),
        (2, "Bob", 4500),
        (3, "Charlie", 5000),
        (4, "David", 3500)]

df = spark.createDataFrame(data, ["id", "name", "salary"])

# กรองพนักงานที่มีเงินเดือนมากกว่า 4000
filtered_df = df.filter(df.salary > 4000)
filtered_df.show()
```

```python
# กรองข้อมูลที่มีเงินเดือนมากกว่า 3000 และน้อยกว่า 5000
filtered_df = df.filter((df.salary > 3000) & (df.salary < 5000))
filtered_df.show()

# กรองข้อมูลที่ชื่อเป็น Bob หรือเงินเดือนมากกว่า 4000
filtered_df = df.filter((df.name == "Bob") | (df.salary > 4000))
filtered_df.show()
```

```python
# กรองข้อมูลที่มีชื่อขึ้นต้นด้วย 'A'
filtered_df = df.filter(df.name.startswith("A"))
filtered_df.show()

# กรองข้อมูลที่ชื่อมีตัวอักษร 'ar'
filtered_df = df.filter(df.name.like("%ar%"))
filtered_df.show()

# กรองข้อมูลที่ชื่อมีลักษณะเป็น regex pattern
filtered_df = df.filter(df.name.rlike("a.*e"))  # กรองชื่อที่ขึ้นต้นด้วย 'a' และลงท้ายด้วย 'e'
filtered_df.show()
```

```python
# กรองพนักงานที่ id เป็น 1 หรือ 3
filtered_df = df.filter(df.id.isin([1, 3]))
filtered_df.show()
```

```python
# กรองข้อมูลที่คอลัมน์ salary มีค่าไม่เป็น null
filtered_df = df.filter(df.salary.isNotNull())
filtered_df.show()

# กรองข้อมูลที่คอลัมน์ salary เป็น null
filtered_df = df.filter(df.salary.isNull())
filtered_df.show()
```

จาก Code Text Analytics

```
26  # กรองออกข้อมูลที่ ReviewText หรือ Rating เป็น null หรือว่างเปล่า
27  data = data.filter(
28      (col("ReviewText").isNotNull()) & (col("ReviewText") != "") &
29      (col("Rating").isNotNull()) & (col("Rating") != float('nan'))
30  )
```

```
# กรองข้อมูลใน DataFrame โดยใช้เงื่อนไขหลายประการ
data = data.filter(
    # เช็คว่าคอลัมน์ "ReviewText" มีค่า (ไม่เป็น null)
    (col("ReviewText").isNotNull()) &
    # เช็คว่าคอลัมน์ "ReviewText" ไม่ใช่ข้อความว่างเปล่า
    (col("ReviewText") != "") &
    # เช็คว่าคอลัมน์ "Rating" มีค่า (ไม่เป็น null)
    (col("Rating").isNotNull()) &
    # เช็คว่าคอลัมน์ "Rating" ไม่ใช่ค่า NaN (Not a Number)
    (col("Rating") != float('nan'))
)
```

ถ้าหากค่ามันว่างใน dataset

```
# แทนค่าว่างใน ReviewText ด้วย "No Review" และ Rating ด้วย 0
data = data.fillna({"ReviewText": "No Review", "Rating": 0.0})

# กรองข้อมูลที่เหลือ โดยไม่มี ReviewText เป็นข้อความว่าง
data = data.filter(
    (col("ReviewText") != "") &
    (col("Rating") != float('nan'))
)
```

```python
from pyspark.sql.functions import col, mean

# คำนวณค่าเฉลี่ยของ Rating
avg_rating = data.select(mean(col("Rating"))).collect()[0][0]

# แทนที่ค่าว่างใน Rating ด้วยค่าเฉลี่ยที่คำนวณได้
data = data.fillna({"Rating": avg_rating})

# แทนค่าว่างใน ReviewText ด้วย "No Review"
data = data.fillna({"ReviewText": "No Review"})

data.show()
```

```python
# ลบแถวที่มีค่า null หรือ NaN ในคอลัมน์ ReviewText หรือ Rating
data = data.dropna(subset=["ReviewText", "Rating"])

# ลบแถวที่ ReviewText เป็นข้อความว่างเปล่า
data = data.filter(col("ReviewText") != "")

data.show()
```