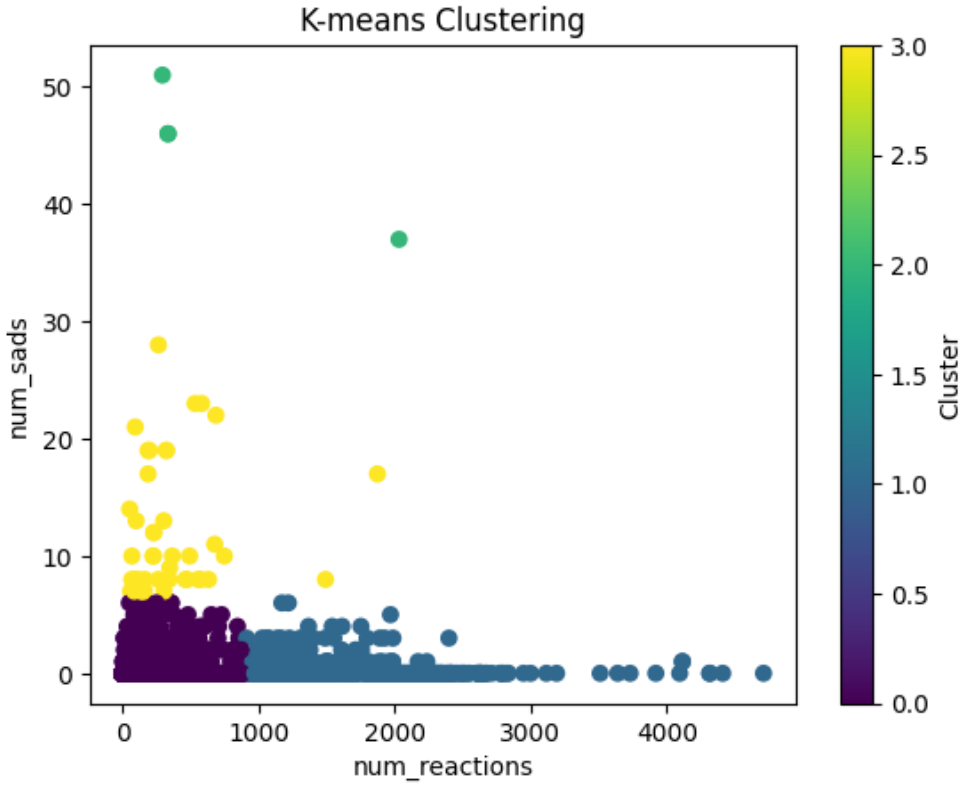
Clustering

K-Mean Implementation

```python
# Import necessary libraries
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml import Pipeline
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import pandas as pd
import matplotlib.pyplot as plt
from pyspark.sql.types import DoubleType

# สร้าง SparkSession สำหรับการใช้งาน PySpark
spark = SparkSession \
    .builder \
    .appName("testKMeans") \
    .getOrCreate()

# อ่านไฟล์ CSV ที่มี header โดยข้อมูลจะถูกเก็บใน DataFrame ชื่อ `df`
df = spark.read.format("csv").option("header", True).load("fb_live_thailand.csv")

# เลือกเฉพาะคอลัมน์ `num_sads` และ `num_reactions` แล้วแปลงข้อมูลเป็นชนิด `Double`
df = df.select(df.num_sads.cast(DoubleType()), df.num_reactions.cast(DoubleType()))

# รวมคอลัมน์ `num_sads` และ `num_reactions` เป็นคอลัมน์ `features` ที่ใช้ในการวิเคราะห์กลุ่ม
vec_assembler = VectorAssembler(inputCols=["num_sads", "num_reactions"], outputCol="features")

# ใช้ StandardScaler เพื่อทำให้ค่าของแต่ละคอลัมน์มีมาตรฐานและเปรียบเทียบกันได้ง่ายขึ้น
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=False)

# สร้างลิสต์สำหรับเก็บค่า Silhouette Score ของแต่ละค่า k
k_values = []

# ลูปหาค่า k ที่เหมาะสมในช่วง 2 ถึง 5
for i in range(2, 5):
    # สร้างโมเดล K-means โดยใช้ค่า k=i
    kmeans = KMeans(featuresCol="scaledFeatures", predictionCol="prediction_col", k=i)

    # รวม stages ของการแปลงและการสเกลข้อมูลลงใน Pipeline
    pipeline = Pipeline(stages=[vec_assembler, scaler, kmeans])

    # ฝึก pipeline model ด้วยข้อมูล `df`
    model = pipeline.fit(df)

    # แปลงข้อมูลด้วยโมเดลที่ฝึกแล้วเพื่อให้ได้ prediction
    output = model.transform(df)

    # ประเมินโมเดลด้วย Silhouette Score โดยใช้ระยะ Squared Euclidean
    evaluator = ClusteringEvaluator(predictionCol="prediction_col", featuresCol="scaledFeatures", metricName="silhouette", distanceMeasure="squaredEuclidean")
    score = evaluator.evaluate(output)

     # บันทึกคะแนน silhouette score ลงในลิสต์ k_values
     k_values.append(score)
     print("Silhouette Score:", score)

# เลือก k ที่ให้ค่า Silhouette Score สูงสุด
best_k = k_values.index(max(k_values)) + 2  # บวก 2 เพราะเริ่มจาก k=2
print("The best k", best_k, max(k_values))

# สร้างโมเดล K-means โดยใช้ k ที่ดีที่สุดที่ได้จากการประเมิน
kmeans = KMeans(featuresCol="scaledFeatures", predictionCol="prediction_col", k=best_k)

# รวม stages ของการแปลงและการสเกลข้อมูลลงใน Pipeline
pipeline = Pipeline(stages=[vec_assembler, scaler, kmeans])

# ฝึก pipeline model ด้วยข้อมูล `df`
model = pipeline.fit(df)

# ทำนายผลการจัดกลุ่มของข้อมูล
predictions = model.transform(df)

# ประเมิน Silhouette Score ของโมเดลที่ได้
evaluator = ClusteringEvaluator(predictionCol="prediction_col", featuresCol="scaledFeatures", metricName="silhouette", distanceMeasure="squaredEuclidean")
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance =", silhouette)

# แปลง DataFrame ของผลการทำนายเป็น Pandas DataFrame เพื่อใช้งานกับ Matplotlib
clustered_data_pd = predictions.toPandas()

# สร้างกราฟกระจายเพื่อแสดงผลการจัดกลุ่ม
plt.scatter(clustered_data_pd["num_reactions"], clustered_data_pd["num_sads"], c=clustered_data_pd["prediction_col"])
plt.xlabel("num_reactions")
plt.ylabel("num_sads")
plt.title("K-means Clustering")
plt.colorbar().set_label("Cluster")
plt.show()
```

Output K-Mean Implementation

```
Silhouette Score: 0.8870485911324794
Silhouette Score: 0.9201883344331407
Silhouette Score: 0.9301227836579924
The best k 4 0.9301227836579924
Silhouette with squared euclidean distance = 0.9301227836579924
```



1. Please provide the example problem that can be solved by using:
   - Regression (0.5 points)
   - Classification (0.5 points)
   - Clustering (0.5 points)
2. Please provide an example of a recommendation system (0.5 points)
3. Please provide an example of a relationship that can be shown by using a Graph (0.5 points)
4. Please code a Spark Structured Streaming for counting time in the column "status_published" in the files fb_part1.csv and fb_part2.csv. Then, write the output to the console where the results are grouped by time and "status_type". (2.5 points)

Please explain your answer and code to Aj. Dr. Sirintra Vaiwsri.

Regression

1.Linear Regression

```python
# Import necessary libraries for Spark, ML, and plotting
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline

import seaborn as sns
import matplotlib.pyplot as plt
from pyspark.sql.functions import col, desc
from pyspark.sql.types import IntegerType

# สร้าง SparkSession สำหรับการใช้งาน PySpark
spark = SparkSession.builder \
    .appName("Linear Regression Analysis") \
    .getOrCreate()

# อ่านข้อมูลจากไฟล์ CSV และตั้งค่าให้มี header และกำหนด schema ให้อัตโนมัติ
data = spark.read.csv('fb_live_thailand.csv', header=True, inferSchema=True)
data.show()  # แสดงข้อมูลตัวอย่างเพื่อให้เห็นภาพรวมของ dataset

# สร้าง VectorAssembler เพื่อรวมคอลัมน์ 'num_reactions' และ 'num_loves' เป็นคอลัมน์ 'features'
assembler = VectorAssembler(
    inputCols=['num_reactions', 'num_loves'],
    outputCol='features'
)

# แปลงข้อมูลโดยใช้ assembler เพื่อสร้างคอลัมน์ 'features' ที่ใช้ในการวิเคราะห์ regression
data_assembled = assembler.transform(data)
data_assembled.show()  # แสดงข้อมูลที่มีคอลัมน์ 'features'

# สร้างโมเดล Linear Regression โดยกำหนดให้ 'num_loves' เป็น label และ 'features' เป็น input
linear_regression = LinearRegression(
    labelCol='num_loves',        # คอลัมน์ที่เป็น label หรือค่าผลลัพธ์ที่ต้องการพยากรณ์
    featuresCol='features',      # คอลัมน์ที่เป็น input feature
    maxIter=10,                  # กำหนดจำนวนรอบการทำงานสูงสุดของการ train
    regParam=0.3,                # ค่าพารามิเตอร์การปกติ (regularization) เพื่อป้องกันการ overfitting
    elasticNetParam=0.8          # พารามิเตอร์ ElasticNet ใช้ผสมระหว่าง Ridge และ Lasso regression
)

# กำหนด Pipeline ซึ่งช่วยในการเรียงลำดับการทำงานของขั้นตอนการเตรียมและสร้างโมเดล
pipeline = Pipeline(stages=[linear_regression])

# แบ่งข้อมูลออกเป็น train (80%) และ test (20%) สำหรับการเทรนและการทดสอบโมเดล
train_data, test_data = data_assembled.randomSplit([0.8, 0.2], seed=42)

# ฝึกโมเดลด้วยข้อมูล train
pipeline_model = pipeline.fit(train_data)
```

```python
# ทำนายผลลัพธ์ของข้อมูล test และแสดงคอลัมน์ num_loves, features, และ prediction
predictions = pipeline_model.transform(test_data)
predictions.select('num_loves', 'features', 'prediction').show(5)  # แสดงตัวอย่างผลการพยากรณ์

# สร้าง evaluator สำหรับประเมินโมเดลโดยใช้ Mean Squared Error (MSE) และ R-squared (R2)
evaluator = RegressionEvaluator(
    labelCol='num_loves',         # ค่าที่แท้จริงของ label
    predictionCol='prediction'    # ค่าที่โมเดลพยากรณ์ได้
)

# คำนวณ Mean Squared Error (MSE) ของโมเดล
mse = evaluator.setMetricName("mse").evaluate(predictions)
print(f"Mean Squared Error (MSE): {mse:.4f}")

# คำนวณค่า R-squared (R2) ของโมเดล ซึ่งแสดงประสิทธิภาพของการพยากรณ์
r2 = evaluator.setMetricName("r2").evaluate(predictions)
print(f"R2: {r2:.4f}")

# แปลงผลลัพธ์การพยากรณ์เป็น Pandas DataFrame เพื่อใช้กับ Matplotlib
pandas_df = predictions.select('num_loves', 'prediction').toPandas()

# สร้างกราฟ scatter plot เพื่อแสดงความสัมพันธ์ระหว่าง num_loves กับ prediction
plt.figure(figsize=(10, 6))
sns.scatterplot(x='num_loves', y='prediction', data=pandas_df)
plt.title('Scatter Plot of num_loves vs Prediction')
plt.xlabel('num_loves')
plt.ylabel('Prediction')
plt.show()

# เลือกคอลัมน์ num_loves และ prediction เปลี่ยนเป็น IntegerType และเรียงข้อมูลตาม prediction ในลำดับมากไปน้อย
selected_data = predictions.select(
    col('num_loves').cast(IntegerType()).alias('num_loves'),
    col('prediction').cast(IntegerType()).alias('prediction')
).orderBy(col('prediction').desc())

# แปลงข้อมูลที่เลือกเป็น Pandas DataFrame เพื่อใช้สำหรับการวิเคราะห์ต่อ
pandas_df = selected_data.toPandas()

# สร้างกราฟ Linear Regression plot ระหว่าง num_loves และ prediction
plt.figure(figsize=(10, 6))
sns.lmplot(x='num_loves', y='prediction', data=pandas_df, aspect=1.5)
plt.title('Linear Regression: num_loves vs Prediction')
plt.xlabel('num_loves')
plt.ylabel('Prediction')
plt.show()
```

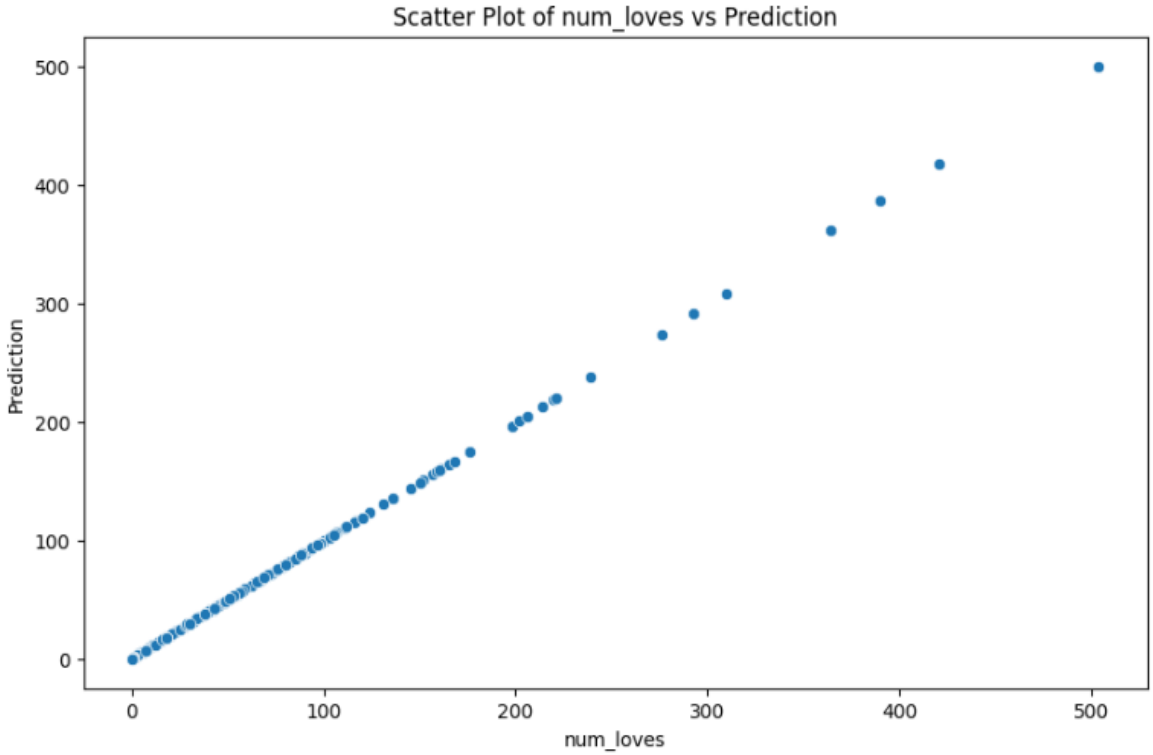
Output Linear Regression

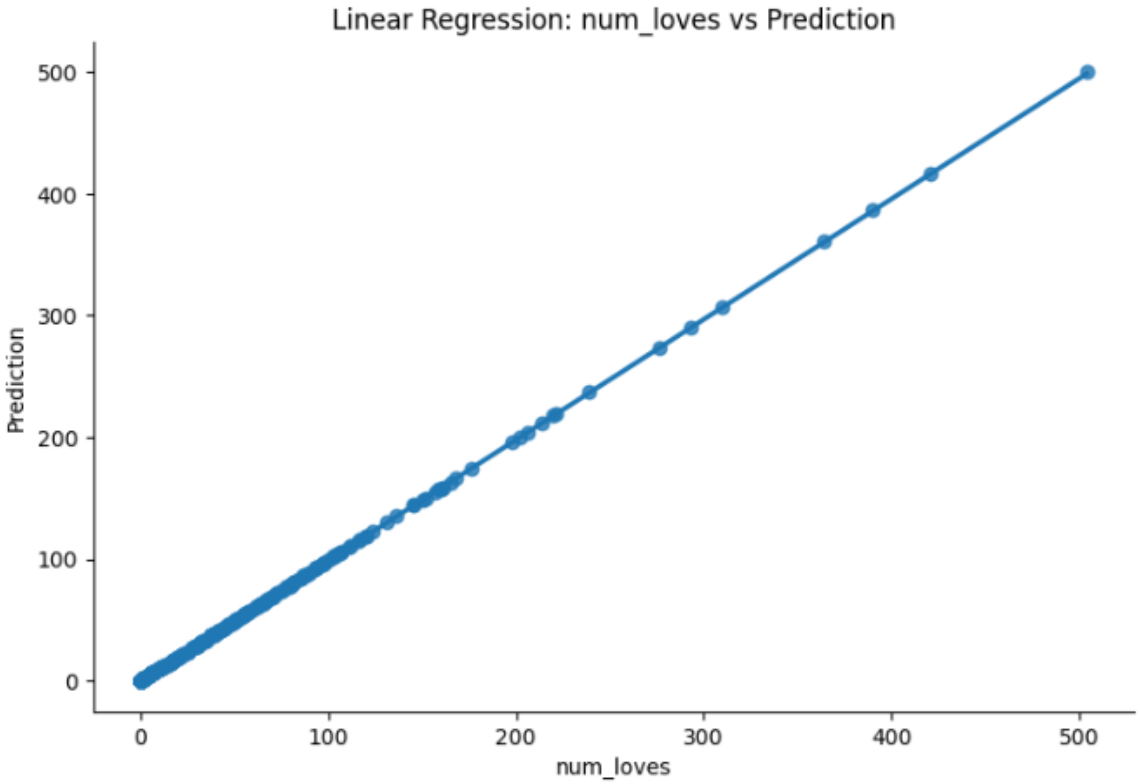
```
+--------------------+----------+---------------+------------+------------+----------+---------+---------+--------+--
|           status_id|status_type|status_published|num_reactions|num_comments|num_shares|num_likes|num_loves|num_wows|nu
+--------------------+----------+---------------+------------+------------+----------+---------+---------+--------+--
|246675545449582_1...|     video|  4/22/2018 6:00|         529|         512|       262|      432|       92|       3|
|246675545449582_1...|     photo| 4/21/2018 22:45|         150|           0|         0|      150|        0|       0|
|246675545449582_1...|     video|  4/21/2018 6:17|         227|         236|        57|      204|       21|       1|
|246675545449582_1...|     photo|  4/21/2018 2:29|         111|           0|         0|      111|        0|       0|
|246675545449582_1...|     photo|  4/18/2018 3:22|         213|           0|         0|      204|        9|       0|
|246675545449582_1...|     photo|  4/18/2018 2:14|         217|           6|         0|      211|        5|       1|
|246675545449582_1...|     video|  4/18/2018 0:24|         503|         614|        72|      418|       70|      10|
|246675545449582_1...|     video|  4/17/2018 7:42|         295|         453|        53|      260|       32|       1|
|246675545449582_1...|     photo|  4/17/2018 3:33|         203|           1|         0|      198|        5|       0|
|246675545449582_1...|     photo|  4/11/2018 4:53|         170|           9|         1|      167|        3|       0|
|246675545449582_1...|     photo|  4/10/2018 1:01|         210|           2|         3|      202|        7|       1|
|246675545449582_1...|     photo|   4/9/2018 2:06|         222|           4|         0|      213|        5|       4|
|246675545449582_1...|     photo|   4/8/2018 5:10|         313|           4|         2|      305|        6|       2|
|246675545449582_1...|     photo|   4/8/2018 2:23|         209|           4|         0|      200|        8|       1|
|246675545449582_1...|     photo|   4/5/2018 9:23|         346|          11|         0|      335|       10|       1|
|246675545449582_1...|     video|   4/1/2018 5:16|         332|         100|        30|      303|       23|       1|
|246675545449582_1...|     video|  3/30/2018 8:28|         135|         256|        79|      117|       18|       0|
|246675545449582_1...|     video|  3/26/2018 8:28|         150|         173|        47|      132|       16|       1|
|246675545449582_1...|     video|  3/23/2018 7:09|         221|         166|        36|      192|       28|       0|
|246675545449582_1...|     photo|  3/22/2018 1:25|         152|           2|         0|      149|        3|       0|
+--------------------+----------+---------------+------------+------------+----------+---------+---------+--------+--
only showing top 20 rows
```


```
+--------------------+----------+---------------+------------+------------+----------+---------+---------+--------+-
|           status_id|status_type|status_published|num_reactions|num_comments|num_shares|num_likes|num_loves|num_wows|n
+--------------------+----------+---------------+------------+------------+----------+---------+---------+--------+-
|246675545449582_1...|     video|  4/22/2018 6:00|         529|         512|       262|      432|       92|       3|
|246675545449582_1...|     photo| 4/21/2018 22:45|         150|           0|         0|      150|        0|       0|
|246675545449582_1...|     video|  4/21/2018 6:17|         227|         236|        57|      204|       21|       1|
|246675545449582_1...|     photo|  4/21/2018 2:29|         111|           0|         0|      111|        0|       0|
|246675545449582_1...|     photo|  4/18/2018 3:22|         213|           0|         0|      204|        9|       0|
|246675545449582_1...|     photo|  4/18/2018 2:14|         217|           6|         0|      211|        5|       1|
|246675545449582_1...|     video|  4/18/2018 0:24|         503|         614|        72|      418|       70|      10|
|246675545449582_1...|     video|  4/17/2018 7:42|         295|         453|        53|      260|       32|       1|
|246675545449582_1...|     photo|  4/17/2018 3:33|         203|           1|         0|      198|        5|       0|
|246675545449582_1...|     photo|  4/11/2018 4:53|         170|           9|         1|      167|        3|       0|
|246675545449582_1...|     photo|  4/10/2018 1:01|         210|           2|         3|      202|        7|       1|
|246675545449582_1...|     photo|   4/9/2018 2:06|         222|           4|         0|      213|        5|       4|
|246675545449582_1...|     photo|   4/8/2018 5:10|         313|           4|         2|      305|        6|       2|
|246675545449582_1...|     photo|   4/8/2018 2:23|         209|           4|         0|      200|        8|       1|
|246675545449582_1...|     photo|   4/5/2018 9:23|         346|          11|         0|      335|       10|       1|
|246675545449582_1...|     video|   4/1/2018 5:16|         332|         100|        30|      303|       23|       1|
|246675545449582_1...|     video|  3/30/2018 8:28|         135|         256|        79|      117|       18|       0|
|246675545449582_1...|     video|  3/26/2018 8:28|         150|         173|        47|      132|       16|       1|
|246675545449582_1...|     video|  3/23/2018 7:09|         221|         166|        36|      192|       28|       0|
|246675545449582_1...|     photo|  3/22/2018 1:25|         152|           2|         0|      149|        3|       0|
+--------------------+----------+---------------+------------+------------+----------+---------+---------+--------+-
only showing top 20 rows
```


```
only showing top 20 rows

+---------+-----------+-------------------+
|num_loves|   features|         prediction|
+---------+-----------+-------------------+
|        1|  [2.0,1.0]| 1.0878276055729477|
|        1|[196.0,1.0]| 1.0878276055729477|
|        2|  [9.0,2.0]|  2.080362383432365|
|        0|[236.0,0.0]|0.09529282771353034|
|        0| [91.0,0.0]|0.09529282771353034|
+---------+-----------+-------------------+
only showing top 5 rows
```

```
Mean Squared Error (MSE): 0.0861
R2: 0.9999
```





2.Decision Trees

```python
# 1. Import libraries
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, VectorAssembler, OneHotEncoder
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline

# สร้าง SparkSession สำหรับการทำงานกับ PySpark
spark = SparkSession.builder \
    .appName("DecisionTreeRegressionExample") \
    .getOrCreate()

# อ่านข้อมูลจากไฟล์ CSV โดยกำหนดให้มี header และให้ PySpark กำหนด schema ให้อัตโนมัติ
data = spark.read.csv("fb_live_thailand.csv", header=True, inferSchema=True)

# แปลงข้อมูลคอลัมน์ 'num_reactions' เป็นดัชนีตัวเลขแทนข้อความ (StringIndexer)
indexer_reactions = StringIndexer(inputCol="num_reactions", outputCol="num_reactions_ind")
indexer_loves = StringIndexer(inputCol="num_loves", outputCol="num_loves_ind")

# แปลงข้อมูลดัชนี (index) เป็นเวกเตอร์แบบ One-Hot Encoding เพื่อให้เป็นข้อมูลเชิงตัวเลขสำหรับการวิเคราะห์
encoder_reactions = OneHotEncoder(inputCols=["num_reactions_ind"], outputCols=["num_reactions_vec"])
encoder_loves = OneHotEncoder(inputCols=["num_loves_ind"], outputCols=["num_loves_vec"])

# สร้าง VectorAssembler เพื่อรวมคอลัมน์เวกเตอร์ของ 'num_reactions_vec' และ 'num_loves_vec' เป็นฟีเจอร์เดียว
assembler = VectorAssembler(inputCols=["num_reactions_vec", "num_loves_vec"], outputCol="features")

# สร้าง Pipeline ที่ประกอบไปด้วยขั้นตอนการแปลงข้อมูลทั้งหมด
pipeline = Pipeline(stages=[indexer_reactions, indexer_loves, encoder_reactions, encoder_loves, assembler])

# ฝึก Pipeline โมเดลด้วยข้อมูลที่มีอยู่
pipeline_model = pipeline.fit(data)

# แปลงข้อมูลด้วย pipeline ที่ฝึกมาแล้ว เพื่อเตรียมข้อมูลสำหรับการทำนาย
transformed_data = pipeline_model.transform(data)

# แบ่งข้อมูลออกเป็น train (80%) และ test (20%) สำหรับการทดสอบ
train_data, test_data = transformed_data.randomSplit([0.8, 0.2])

# สร้างโมเดล Decision Tree Regressor โดยใช้ 'num_loves_ind' เป็น label และ 'features' เป็น input
dt = DecisionTreeRegressor(labelCol="num_loves_ind", featuresCol="features")

# ฝึก Decision Tree โมเดลด้วยข้อมูล train
dt_model = dt.fit(train_data)

# ทำนายผลลัพธ์จากโมเดลโดยใช้ข้อมูลทดสอบ (test data)
predictions = dt_model.transform(test_data)

# สร้าง evaluator เพื่อวัดผลลัพธ์ของโมเดล โดยเลือกใช้ R2 Score ในการประเมิน
evaluator = RegressionEvaluator(labelCol="num_loves_ind", predictionCol="prediction")

# คำนวณค่า R2 Score ของโมเดล R2  = ค่าสัมประสิทธิ์การกำหนด
r2 = evaluator.setMetricName("r2").evaluate(predictions)
print(f"R2 score: {r2}")

# ปิด SparkSession หลังใช้งานเสร็จสิ้น
spark.stop()
```

Output Decision Trees

```
R2 score: 0.4518945989051336
```

มีแค่นี้แหละไอสัส

Classification

1.Logistic Regression

```python
# 1. Import necessary libraries
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline
# สร้าง SparkSession เพื่อทำงานกับ PySpark
spark = SparkSession.builder \
    .appName("LogisticRegressionExample") \
    .getOrCreate()

# อ่านข้อมูลจากไฟล์ CSV โดยมี header และให้ PySpark ทำการกำหนด schema ให้อัตโนมัติ
data = spark.read.csv("fb_live_thailand.csv", header=True, inferSchema=True)

# แปลงข้อมูลในคอลัมน์ 'status_type' เป็นตัวเลขโดยใช้ StringIndexer
status_type_indexer = StringIndexer(inputCol="status_type", outputCol="status_type_ind")
status_published_indexer = StringIndexer(inputCol="status_published", outputCol="status_published_ind")

# ใช้ StringIndexer ที่สร้างขึ้นกับข้อมูลจริง โดยแปลง 'status_type' และ 'status_published' ให้เป็นข้อมูลเชิงตัวเลข
data_indexed = status_type_indexer.fit(data).transform(data)
data_indexed = status_published_indexer.fit(data_indexed).transform(data_indexed)

# สร้าง VectorAssembler เพื่อรวมคอลัมน์ 'status_type_ind' และ 'status_published_ind' เป็นฟีเจอร์เดียวที่เรียกว่า 'features'
assembler = VectorAssembler(inputCols=["status_type_ind", "status_published_ind"], outputCol="features")

# กำหนด Logistic Regression โมเดลโดยใช้ 'features' เป็น input และ 'status_type_ind' เป็น label
lr = LogisticRegression(featuresCol="features", labelCol="status_type_ind")
lr.setMaxIter(100).setRegParam(0.1).setElasticNetParam(0.5)  # ตั้งค่าการทำซ้ำ (iterations), ค่าปรับค่า (regularization), และ ElasticNet

# สร้าง Pipeline ที่รวมขั้นตอน assembler และ logistic regression เข้าไว้ด้วยกัน
pipeline = Pipeline(stages=[assembler, lr])

# แบ่งข้อมูลออกเป็นชุดฝึกสอน (train) 80% และชุดทดสอบ (test) 20%
train_data, test_data = data_indexed.randomSplit([0.8, 0.2])

# ฝึก Pipeline โมเดลด้วยชุดข้อมูลฝึกสอน
pipeline_model = pipeline.fit(train_data)

# ใช้โมเดลที่ฝึกแล้วทำการทำนายผลลัพธ์บนชุดข้อมูลทดสอบ
predictions = pipeline_model.transform(test_data)

# แสดงตัวอย่างผลลัพธ์ โดยจะแสดงค่า status_type_ind, prediction, และ probability
predictions.select("status_type_ind", "prediction", "probability").show(5)

# สร้าง evaluator เพื่อประเมินผลลัพธ์ของโมเดลด้วยการคำนวณค่า accuracy, precision, recall และ F1 score
evaluator = MulticlassClassificationEvaluator(labelCol="status_type_ind", predictionCol="prediction")

# คำนวณค่า accuracy ของโมเดล
accuracy = evaluator.setMetricName("accuracy").evaluate(predictions)
print(f"Accuracy: {accuracy}")

# คำนวณค่า precision ของโมเดล (เฉลี่ยแบบถ่วงน้ำหนัก)
precision = evaluator.setMetricName("weightedPrecision").evaluate(predictions)
print(f"Precision: {precision}")

# คำนวณค่า recall ของโมเดล (เฉลี่ยแบบถ่วงน้ำหนัก)
recall = evaluator.setMetricName("weightedRecall").evaluate(predictions)
print(f"Recall: {recall}")

# คำนวณค่า F1 ของโมเดล
f1 = evaluator.setMetricName("f1").evaluate(predictions)
print(f"F1 measure: {f1}")

# ปิด SparkSession หลังจากใช้งานเสร็จ
spark.stop()
```

Output Logistic Regression

```
+---------------+----------+--------------------+
|status_type_ind|prediction|         probability|
+---------------+----------+--------------------+
|            0.0|       0.0|[0.85661940556496...|
|            0.0|       0.0|[0.85661940556496...|
|            0.0|       0.0|[0.85661940556496...|
|            1.0|       1.0|[0.26150798253639...|
|            0.0|       0.0|[0.85661940556496...|
+---------------+----------+--------------------+
only showing top 5 rows

Accuracy: 0.9392226148409893
Precision: 0.8877788995456839
Recall: 0.9392226148409893
F1 measure: 0.9113614150679032
```

2.Decision Trees Classification

```python
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, VectorAssembler, OneHotEncoder
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline

# สร้าง SparkSession เพื่อทำงานกับ PySpark
spark = SparkSession.builder.appName("FBLiveTH").getOrCreate()

# อ่านข้อมูลจากไฟล์ CSV โดยกำหนดให้มี header และ inferSchema เพื่อให้ PySpark กำหนด schema ให้อัตโนมัติ
data = spark.read.csv("fb_live_thailand.csv", header=True, inferSchema=True)

# ใช้ StringIndexer เพื่อแปลงข้อมูลในคอลัมน์ 'status_type' และ 'status_published' เป็นข้อมูลเชิงตัวเลข
status_type_indexer = StringIndexer(inputCol="status_type", outputCol="status_type_ind")
status_published_indexer = StringIndexer(inputCol="status_published", outputCol="status_published_ind")

# ใช้ OneHotEncoder เพื่อแปลงข้อมูลที่แปลงเป็นตัวเลขแล้วให้เป็นการเข้ารหัสแบบ one-hot
status_type_encoder = OneHotEncoder(inputCol="status_type_ind", outputCol="status_type_encoded")
status_published_encoder = OneHotEncoder(inputCol="status_published_ind", outputCol="status_published_encoded")

# รวมคอลัมน์ที่เข้ารหัสแล้วลงในคอลัมน์เดียว 'features' สำหรับใช้ในโมเดล
assembler = VectorAssembler(inputCols=["status_type_encoded", "status_published_encoded"], outputCol="features")

# สร้าง Pipeline รวมทุกขั้นตอนการเตรียมข้อมูลและการแปลงข้อมูลที่ต้องใช้
pipeline = Pipeline(stages=[status_type_indexer, status_published_indexer, status_type_encoder, status_published_encoder, assembler])

# ฝึก Pipeline model บนชุดข้อมูล
pipeline_model = pipeline.fit(data)

# แปลงข้อมูลด้วย pipeline model เพื่อให้ได้ข้อมูลพร้อมสำหรับฝึกโมเดล
transformed_data = pipeline_model.transform(data)

# แบ่งข้อมูลออกเป็นชุดฝึกสอน (train) 80% และชุดทดสอบ (test) 20%
train_data, test_data = transformed_data.randomSplit([0.8, 0.2])

# สร้าง Decision Tree Classifier โดยกำหนด 'features' เป็น input และ 'status_type_ind' เป็น label
decision_tree = DecisionTreeClassifier(labelCol="status_type_ind", featuresCol="features")

# ฝึก Decision Tree model บนชุดข้อมูลฝึกสอน
decision_tree_model = decision_tree.fit(train_data)

# ทำการทำนายผลลัพธ์บนชุดข้อมูลทดสอบ
predictions = decision_tree_model.transform(test_data)

# ประเมินผลลัพธ์ของโมเดลด้วย metric ต่างๆ เช่น accuracy, precision, recall, และ f1
evaluator = MulticlassClassificationEvaluator(labelCol="status_type_ind", predictionCol="prediction")

# คำนวณ accuracy ของโมเดล
accuracy = evaluator.evaluate(predictions, {evaluator.metricName: "accuracy"})

# คำนวณ weighted precision ของโมเดล
precision = evaluator.evaluate(predictions, {evaluator.metricName: "weightedPrecision"})

# คำนวณ weighted recall ของโมเดล
recall = evaluator.evaluate(predictions, {evaluator.metricName: "weightedRecall"})

# คำนวณ F1 measure ของโมเดล
f1 = evaluator.evaluate(predictions, {evaluator.metricName: "f1"})

# คำนวณ test error (ความคลาดเคลื่อนของโมเดล)
test_error = 1.0 - accuracy

# แสดงผลลัพธ์ metric ต่างๆ
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Measure: {f1}")
print(f"Test Error: {test_error}")
```

Output Decision Trees Classification

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Measure: 1.0
Test Error: 0.0
```

Recommendation System

Recommendation System Implementation

```python
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS

# สร้าง Spark session สำหรับการทำงานกับ PySpark
spark = SparkSession.builder.appName("BookRecommendations").getOrCreate()

# โหลดชุดข้อมูลจากไฟล์ CSV โดยให้ PySpark ทำการกำหนด schema ให้อัตโนมัติ
df = spark.read.csv('book_ratings.csv', header=True, inferSchema=True)

# แสดง schema ของข้อมูลเพื่อยืนยันโครงสร้างว่าโหลดข้อมูลถูกต้อง
df.printSchema()

# กำหนดโมเดล ALS (Alternating Least Squares) สำหรับการสร้างระบบแนะนำหนังสือ
als = ALS(
    maxIter=10,                # จำนวน iteration สูงสุดที่ ALS จะรันเพื่อหาผลลัพธ์
    regParam=0.1,              # ค่าพารามิเตอร์การปรับให้เหมาะสมเพื่อลด overfitting
    userCol="user_id",          # คอลัมน์ที่ใช้แทนผู้ใช้
    itemCol="book_id",          # คอลัมน์ที่ใช้แทนไอเท็ม (ในที่นี้คือหนังสือ)
    ratingCol="rating",        # คอลัมน์ที่ใช้แทนค่าคะแนนที่ให้โดยผู้ใช้
    coldStartStrategy="drop"   # วิธีการจัดการกับค่า NaN ในการทำนาย (ใช้ "drop" เพื่อละทิ้งค่า NaN)
)

# แบ่งข้อมูลออกเป็นชุดข้อมูลฝึกสอน (training) 80% และชุดข้อมูลทดสอบ (test) 20%
(training_data, test_data) = df.randomSplit([0.8, 0.2])

# ฝึกโมเดล ALS บนชุดข้อมูลฝึกสอน
model = als.fit(training_data)

# สร้างการทำนายผลลัพธ์บนชุดข้อมูลทดสอบ
predictions = model.transform(test_data)

# กำหนดตัวประเมินผลโดยใช้ RMSE (Root Mean Square Error) เป็น metric
evaluator = RegressionEvaluator(
    metricName="rmse",         # ใช้ RMSE เป็น metric ในการประเมินผล
    labelCol="rating",         # คอลัมน์ของค่าคะแนนจริง
    predictionCol="prediction"  # คอลัมน์ของค่าคะแนนที่โมเดลทำนาย
)

# ประเมินผลโมเดลโดยคำนวณ RMSE ซึ่งแสดงถึงความคลาดเคลื่อนของการทำนาย
rmse = evaluator.evaluate(predictions)
print(f"Root-mean-square error (RMSE): {rmse}")

# กรองข้อมูลเพื่อแสดงข้อมูล User ID, Book ID, Rating และ Prediction สำหรับผู้ใช้ที่ระบุ (ตัวอย่างเช่น User ID = 53)
user_id = 53
filtered_user_data = df.filter(df['user_id'] == user_id)

# แสดงข้อมูลการให้คะแนนจริงของผู้ใช้ที่เลือก
filtered_user_data.show()

# แสดงการทำนายผลลัพธ์สำหรับผู้ใช้ที่เลือก
user_predictions = model.transform(filtered_user_data)
user_predictions.orderBy('prediction', ascending=False).show()

# แสดงหนังสือที่แนะนำ 5 เล่มสำหรับผู้ใช้ทุกคน
model.recommendForAllUsers(5).show(truncate=False)

# แสดงผู้ใช้ 5 คนที่แนะนำให้หนังสือทุกเล่ม
model.recommendForAllItems(5).show(truncate=False)

# ปิด Spark session หลังจากทำงานเสร็จสิ้น
spark.stop()
```

Output Recommendation System Implementation

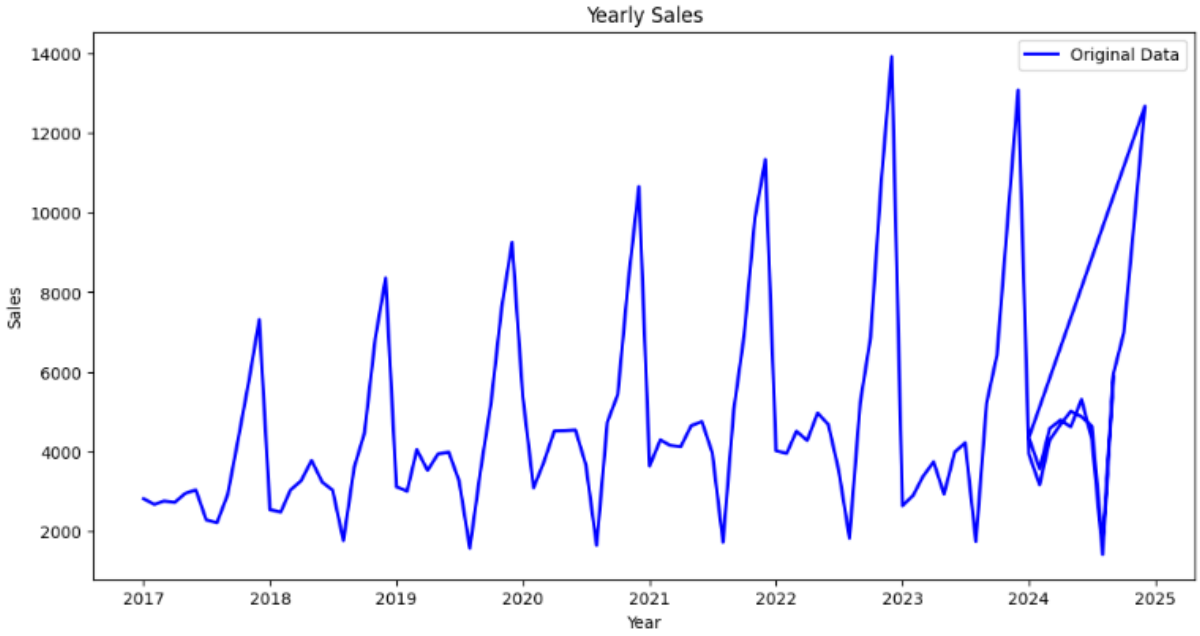
```
root
 |-- book_id: integer (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- rating: integer (nullable = true)

Root-mean-square error (RMSE): 0.9083333922089859
+-------+-------+------+
|book_id|user_id|rating|
+-------+-------+------+
|   8336|     53|     1|
|   8336|     53|     1|
|   8882|     53|     2|
|   8946|     53|     5|
+-------+-------+------+

+-------+-------+------+----------+
|book_id|user_id|rating|prediction|
+-------+-------+------+----------+
|   8882|     53|     2| 1.8147241|
|   8946|     53|     5|  1.664831|
|   8336|     53|     1| 1.0298772|
|   8336|     53|     1| 1.0298772|
+-------+-------+------+----------+

+-------+--------------------------------------------------------------------------------------------------------+
|user_id|recommendations                                                                                         |
+-------+--------------------------------------------------------------------------------------------------------+
|1      |[{4868, 4.783501}, {6902, 4.5888133}, {6862, 4.545219}, {7275, 4.46538}, {5634, 4.436862}]              |
|3      |[{7440, 1.1478753}, {7401, 1.1242633}, {1146, 1.1092325}, {780, 1.1082761}, {4868, 1.1049389}]          |
|5      |[{1275, 5.2502007}, {2441, 5.1921873}, {6590, 5.185659}, {3692, 5.137342}, {3753, 5.1194196}]           |
|6      |[{3836, 5.014157}, {296, 5.008198}, {641, 4.97537}, {9182, 4.9583907}, {9964, 4.9073863}]               |
|9      |[{6102, 4.501539}, {6070, 4.3734245}, {8109, 4.36571}, {8323, 4.2740636}, {5392, 4.251309}]             |
|12     |[{6784, 4.172377}, {4519, 4.138496}, {8976, 4.0544405}, {7440, 4.0480103}, {4468, 4.018967}]            |
|13     |[{723, 4.827312}, {3753, 4.669796}, {267, 4.6686625}, {8926, 4.6615977}, {3228, 4.637865}]              |
|15     |[{6425, 3.250804}, {8109, 3.1256897}, {6591, 3.1151812}, {2840, 3.0874515}, {192, 3.0744324}]           |
|16     |[{8973, 3.9810503}, {8946, 3.908478}, {9008, 3.8134089}, {653, 3.7855232}, {8062, 3.7524073}]           |
|17     |[{7063, 4.905998}, {3628, 4.655303}, {6590, 4.5271764}, {5580, 4.4374213}, {8757, 4.415572}]            |
|19     |[{296, 4.27357}, {7440, 4.2654557}, {5670, 4.2180405}, {6784, 4.197125}, {9849, 4.1881056}]             |
|20     |[{3282, 4.4283347}, {2334, 4.40883}, {3228, 4.3904324}, {7988, 4.3691125}, {7736, 4.3401523}]           |
|22     |[{307, 4.535504}, {9842, 4.5279055}, {192, 4.478823}, {8847, 4.465287}, {2877, 4.4638796}]              |
|26     |[{4344, 4.0697174}, {9024, 4.0050287}, {5101, 3.975672}, {4640, 3.9562206}, {7081, 3.9522126}]          |
|27     |[{4344, 4.9822583}, {3124, 4.880452}, {6018, 4.862772}, {3635, 4.8242598}, {5841, 4.7904925}]           |
|28     |[{4344, 4.196404}, {5175, 4.126596}, {9024, 4.121041}, {3953, 4.116617}, {8323, 4.108463}]              |
|31     |[{3628, 4.199583}, {4054, 4.1578813}, {3172, 4.132264}, {653, 4.0901036}, {2782, 3.9931068}]            |
|34     |[{4154, 3.2871072}, {8976, 3.2647533}, {8498, 3.1991436}, {1788, 3.1868522}, {3836, 3.16937}]           |
|35     |[{7401, 3.9677713}, {9842, 3.9605675}, {6590, 3.9165053}, {3628, 3.8568618}, {5344, 3.8501143}]         |
|37     |[{7615, 4.922957}, {7499, 4.8990483}, {2507, 4.888843}, {3491, 4.8253756}, {6784, 4.823132}]            |
+-------+--------------------------------------------------------------------------------------------------------+
only showing top 20 rows

+-------+--------------------------------------------------------------------------------------------------------+
|book_id|recommendations                                                                                         |
+-------+--------------------------------------------------------------------------------------------------------+
|1      |[{50307, 5.9419246}, {24127, 5.9225807}, {20618, 5.9172873}, {12759, 5.9099565}, {3573, 5.9071684}]     |
|3      |[{34547, 5.90882}, {5550, 5.7234826}, {38155, 5.644366}, {11440, 5.5918374}, {27969, 5.473458}]         |
|5      |[{38866, 5.6044984}, {26653, 5.5812335}, {13145, 5.563851}, {50307, 5.4908924}, {14816, 5.4854083}]     |
|6      |[{52579, 6.0837784}, {24688, 5.803474}, {34886, 5.7815795}, {50138, 5.7490354}, {48202, 5.744907}]      |
|9      |[{48202, 5.428156}, {43053, 5.4109}, {13832, 5.356568}, {15077, 5.3164124}, {37163, 5.2009373}]         |
|12     |[{27969, 5.707608}, {52579, 5.521587}, {26576, 5.486446}, {23176, 5.4503903}, {20124, 5.40377}]         |
|13     |[{13145, 5.7655625}, {28602, 5.712561}, {12360, 5.6571603}, {14816, 5.6080723}, {16530, 5.591783}]      |
|15     |[{50307, 5.893988}, {14816, 5.775786}, {12749, 5.7542334}, {42265, 5.730556}, {14227, 5.6796722}]       |
|16     |[{14135, 5.365951}, {41819, 5.3636565}, {28800, 5.305502}, {20618, 5.293181}, {14785, 5.28593}]         |
|17     |[{34886, 5.8095}, {13832, 5.7923965}, {12749, 5.7779846}, {27969, 5.759008}, {30641, 5.752398}]         |
|19     |[{13145, 6.012303}, {40925, 5.8686123}, {1541, 5.81984}, {38506, 5.7493505}, {12225, 5.743251}]         |
|20     |[{52579, 5.249507}, {25434, 5.204713}, {50307, 5.197488}, {30641, 5.1935654}, {34886, 5.1642933}]       |
|22     |[{47347, 5.2628074}, {13145, 5.206897}, {25434, 5.205415}, {51190, 5.1975107}, {7817, 5.190915}]        |
|26     |[{43053, 5.6235833}, {13832, 5.5643353}, {48202, 5.4496527}, {43853, 5.386092}, {45943, 5.3385787}]     |
|27     |[{14135, 5.996331}, {41819, 5.9039817}, {16486, 5.859773}, {15466, 5.8396726}, {24127, 5.768296}]       |
|28     |[{12360, 5.7005243}, {28602, 5.6809826}, {16561, 5.657517}, {16530, 5.591938}, {13145, 5.5531583}]      |
|31     |[{25434, 5.8717017}, {6932, 5.8131027}, {7817, 5.798622}, {12759, 5.7930555}, {51168, 5.7486496}]       |
|34     |[{11296, 6.0432825}, {50073, 5.337174}, {46098, 5.22721}, {10466, 5.203865}, {15496, 5.164677}]         |
|35     |[{41819, 5.5304317}, {26969, 5.3771396}, {49873, 5.363899}, {4486, 5.360492}, {9984, 5.360289}]         |
|37     |[{12759, 5.6604095}, {8958, 5.566295}, {42316, 5.5623755}, {23176, 5.555716}, {50124, 5.5523944}]       |
+-------+--------------------------------------------------------------------------------------------------------+
only showing top 20 rows
```

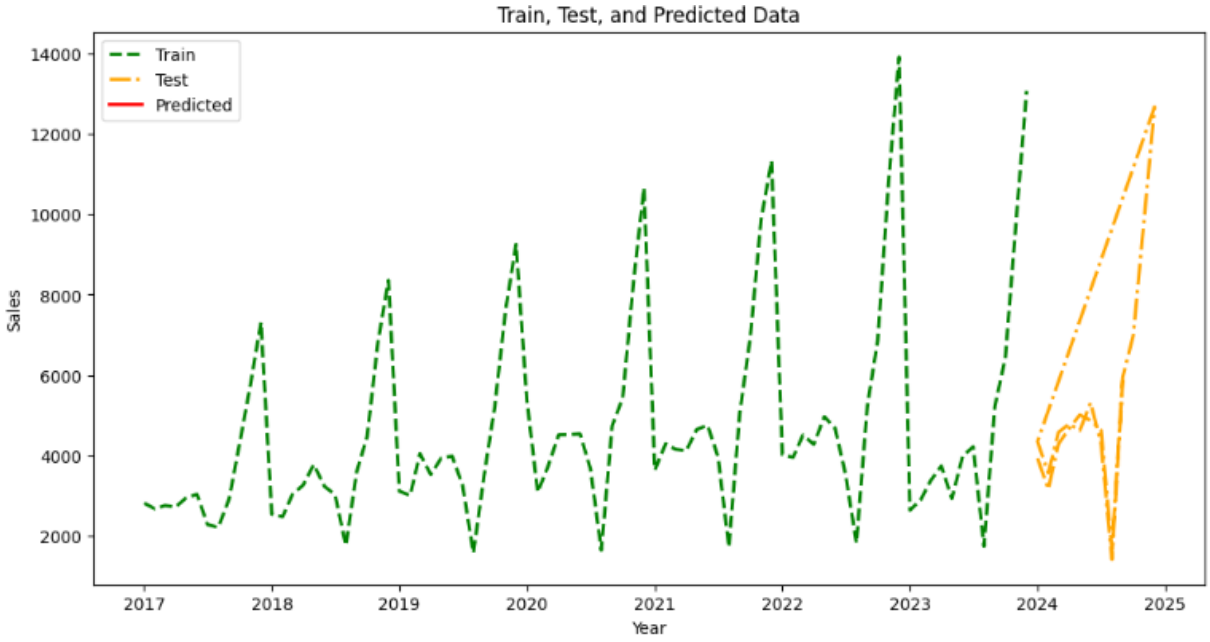
Time Series

Arima

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA  # Updated import statement
from statsmodels.tsa.stattools import adfuller

# โหลดข้อมูลจากไฟล์ CSV
df = pd.read_csv('year_sales.csv')

# แสดงข้อมูล 5 แถวแรกเพื่อตรวจสอบโครงสร้างของข้อมูล
print(df.head())

# แปลงคอลัมน์ 'Year' เป็น datetime format โดยใช้รูปแบบปีและเดือน (เช่น '2023-01')
df['Year'] = pd.to_datetime(df['Year'], format='%Y-%m')

# ตั้งค่า 'Year' เป็นดัชนี (index) ของ DataFrame เพื่อให้ง่ายต่อการทำงานกับข้อมูลเวลา
df.set_index('Year', inplace=True)

# วาดกราฟแสดงข้อมูล Time Series ของยอดขายในแต่ละปี
plt.figure(figsize=(12, 6))
plt.plot(df, label='Original Data', color='blue', linestyle='-', linewidth=2)
plt.title('Yearly Sales')        # กำหนดหัวข้อกราฟ
plt.xlabel('Year')               # กำหนดชื่อแกน x
plt.ylabel('Sales')              # กำหนดชื่อแกน y
plt.legend()
# plt.show()  # หากต้องการแสดงกราฟ ให้เอาคอมเมนต์ออก

# ทดสอบความนิ่งของข้อมูล (stationarity) โดยใช้ ADF test
result = adfuller(df['Sales'])
print('ADF Statistic:', result[0])  # แสดงค่าสถิติ ADF
print('p-value:', result[1])       # แสดงค่า p-value เพื่อตัดสินใจว่าข้อมูลนิ่งหรือไม่

# แบ่งข้อมูลออกเป็นชุดฝึกสอน (train) 80% และชุดทดสอบ (test) 20%
train_size = int(len(df) * 0.8)
train = df.iloc[:train_size]
test = df.iloc[train_size:]

# กำหนดและปรับแต่งโมเดล ARIMA โดยใช้ค่า order = (1,1,1) (สามารถปรับค่า order ตามความเหมาะสม)
model = ARIMA(train, order=(1,1,1))  # (p,d,q): p = AR, d = differencing, q = MA
model_fit = model.fit()

# แสดงสรุปผลการฝึกสอนโมเดล ARIMA
print(model_fit.summary())

# สร้างการทำนายสำหรับช่วงข้อมูล test
predictions = model_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
predictions = pd.DataFrame(predictions, index=test.index, columns=['Predicted'])  # กำหนด DataFrame สำหรับการทำนาย
 # วาดกราฟแสดงข้อมูลชุดฝึกสอน (train), ชุดทดสอบ (test) และค่าที่ทำนาย (predicted)
 plt.figure(figsize=(12, 6))
 plt.plot(train, label='Train', color='green', linestyle='--', linewidth=2)   # ข้อมูลชุดฝึกสอน
 plt.plot(test, label='Test', color='orange', linestyle='-.', linewidth=2)    # ข้อมูลชุดทดสอบ
 plt.plot(predictions, label='Predicted', color='red', linestyle='-', linewidth=2)  # ข้อมูลที่ทำนาย
 plt.title('Train, Test, and Predicted Data')   # หัวข้อกราฟ
 plt.xlabel('Year')                             # ชื่อแกน x
 plt.ylabel('Sales')                            # ชื่อแกน y
 plt.legend()
 plt.show()   # แสดงกราฟเปรียบเทียบข้อมูลจริงกับข้อมูลที่ทำนาย
```

Output Arima





Association Rule

Association Rule

```python
from pyspark.sql import SparkSession
from pyspark.ml.fpm import FPGrowth
from pyspark.sql.functions import trim, split, collect_list, array_distinct, expr

# Step 1: Initialize Spark Session with more memory
# สร้าง Spark Session พร้อมการกำหนดหน่วยความจำเพิ่มเติมสำหรับการประมวลผลข้อมูลขนาดใหญ่
spark = SparkSession.builder \
    .appName("FPGrowth Example") \
    .config("spark.executor.memory", "4g") \
    .config("spark.driver.memory", "4g") \
    .config("spark.executor.cores", "2") \
    .config("spark.task.maxFailures", "5") \
    .getOrCreate()

# Step 2: Read CSV data
# โหลดข้อมูลจากไฟล์ CSV โดยกำหนดให้ inferSchema เป็น True เพื่อให้ Spark ตรวจสอบประเภทข้อมูลอัตโนมัติ
data = spark.read.csv("groceries_data.csv", header=True, inferSchema=True)

# Step 3: Use trim to clean any leading/trailing spaces in 'itemDescription'
# Also replace 'rolls/buns' with 'rolls,buns'
# ใช้ฟังก์ชัน split ในการแยกข้อมูลที่มี '/' ในคอลัมน์ 'itemDescription' เพื่อให้แยกเป็นรายการเดี่ยว ๆ
data = data.withColumn("itemDescription", split(data["itemDescription"], "/"))

# Step 4: Split 'rolls,buns' into separate items
# ใช้ฟังก์ชัน split แยกคำใน 'itemDescription' โดยใช้ ',' เป็นตัวแบ่ง
# data = data.withColumn("itemDescription", split("itemDescription", ","))

# Step 5: Group by 'Member_number' and collect unique items into a basket
# รวมรายการซื้อของสมาชิกแต่ละคนไว้ใน 'basket' โดยใช้ collect_list และ array_distinct เพื่อให้รายการไม่ซ้ำกันในแต่ละตะกร้า
grouped_data = data.groupBy("Member_number").agg(collect_list("itemDescription").alias("basket"))
grouped_data = grouped_data.withColumn("basket", array_distinct("basket"))

# Step 6: Initialize and fit FPGrowth model
# สร้างและฝึกฝนโมเดล FPGrowth โดยกำหนด minSupport (การสนับสนุนต่ำสุด) และ minConfidence (ความเชื่อมั่นต่ำสุด) สำหรับกฎการเชื่อมโยง
fp = FPGrowth(minSupport=0.01, minConfidence=0.3, itemsCol="basket", predictionCol="prediction")
model = fp.fit(grouped_data)

# Step 7: Show frequent itemsets
# แสดงชุดไอเทมที่พบว่ามีความถี่สูงในการซื้อร่วมกัน
model.freqItemsets.show(10, truncate=False)

# Step 8: Show association rules
# แสดงกฎการเชื่อมโยงที่มีค่าความเชื่อมั่นสูงกว่า 0.5 (เช่น เมื่อซื้อสินค้าหนึ่งมักจะซื้ออีกสินค้าร่วมด้วย)
model.associationRules.filter(model.associationRules.confidence > 0.5).show(truncate=False)
```

```python
# Step 9: Create new data for predictions
# สร้างข้อมูลตัวอย่างใหม่สำหรับการทำนายเพื่อดูว่าโมเดลจะแนะนำสินค้าชิ้นใด
new_data = spark.createDataFrame([
    (["vegetable juice", "frozen fruits", "packaged fruit"],),
    (["mayonnaise", "butter", "rolls"],)  # Separate 'rolls' from 'buns'
], ["basket"])

# Step 10: Transform the model to make predictions
# ทำการทำนายโดยใช้ข้อมูลตัวอย่างใหม่เพื่อแนะนำสินค้าที่น่าจะซื้อร่วมกัน
predictions = model.transform(new_data)
predictions.show(truncate=False)

# Stop Spark session
# ปิด Spark Session หลังจากจบการทำงาน
spark.stop()
```

Output Association Rule

```
+-------------------------------------------------+----+
|items                                            |freq|
+-------------------------------------------------+----+
|[[specialty cheese]]                             |71  |
|[[zwieback]]                                     |60  |
|[[pet care]]                                     |85  |
|[[pet care], [rolls, buns]]                      |40  |
|[[pet care], [other vegetables]]                 |40  |
|[[house keeping products]]                       |45  |
|[[curd]]                                         |471 |
|[[curd], [sausage]]                              |125 |
|[[curd], [sausage], [rolls, buns]]               |59  |
|[[curd], [sausage], [rolls, buns], [whole milk]]|39  |
+-------------------------------------------------+----+
only showing top 10 rows
```

```
+-----------------------------------------------------+--------------------+--------------------+------------------+--------------------+
|antecedent                                           |consequent          |confidence          |lift              |support             |
+-----------------------------------------------------+--------------------+--------------------+------------------+--------------------+
|[[coffee], [yogurt], [other vegetables]]             |[[whole milk]]      |0.6507936507936508|1.4203771840949893|0.01051821446895844 |
|[[canned beer], [yogurt]]                            |[[whole milk]]      |0.5544554455445545|1.2101160843967935|0.028732683427398667|
|[[pastry], [bottled water], [rolls, buns]]           |[[other vegetables]]|0.5571428571428572|1.4793888672635267|0.010005130836326322|
|[[curd], [yogurt], [soda]]                           |[[whole milk]]      |0.711864406779661 |1.5536659897128324|0.0107747562852745  |
|[[sausage], [tropical fruit]]                        |[[whole milk]]      |0.539906103286385 |1.1783616968702848|0.029502308876346844|
|[[butter], [bottled beer]]                           |[[whole milk]]      |0.524390243902439 |1.1444978559528036|0.011031298101590559|
|[[pot plants]]                                       |[[other vegetables]]|0.5086206896551724|1.3505473080898243|0.015135967162647512|
|[[pot plants]]                                       |[[whole milk]]      |0.5258620689655172|1.1477101594779318|0.01564905079527963 |
|[[dessert], [root vegetables]]                       |[[whole milk]]      |0.5408163265306123|1.1803482871312019|0.013596716264751155|
|[[UHT-milk], [pip fruit]]                            |[[whole milk]]      |0.597014925373134 3|1.303003459744948 |0.01026167265264238 |
|[[pip fruit], [tropical fruit], [other vegetables]]|[[whole milk]]      |0.5128205128205128|1.1192465615757887|0.01026167265264238 |
|[[newspapers], [canned beer]]                        |[[whole milk]]      |0.5454545454545454|1.1904713427669753|0.01539250897896357 |
|[[domestic eggs], [pip fruit]]                       |[[whole milk]]      |0.5747126436781609|1.2543280431452806|0.012827090815802977|
|[[shopping bags], [soda], [whole milk]]              |[[rolls, buns]]     |0.5174825174825175|1.4799316604158865|0.018984094407388404|
|[[shopping bags], [soda], [whole milk]]              |[[other vegetables]]|0.5104895104895105|1.355509613002801 |0.018727552591072345|
|[[newspapers], [pip fruit]]                          |[[whole milk]]      |0.5384615384615384|1.1752088896545783|0.014366341713699333|
|[[fruit, vegetable juice], [domestic eggs]]          |[[whole milk]]      |0.5681818181818182|1.2400743153822662|0.012827090815802977|
|[[newspapers], [yogurt], [rolls, buns]]              |[[whole milk]]      |0.6052631578947368|1.3210054812282666|0.011800923550538737|
|[[frozen meals], [other vegetables]]                 |[[rolls, buns]]     |0.5277777777777778|1.5093747452514878|0.014622883530015392|
|[[frozen meals], [other vegetables]]                 |[[whole milk]]      |0.6203703703703703|1.3539774376840445|0.01718830169317599 |
+-----------------------------------------------------+--------------------+--------------------+------------------+--------------------+
only showing top 20 rows
```

```
+-------------------------------------------------+----------+
|basket                                           |prediction|
+-------------------------------------------------+----------+
|[vegetable juice, frozen fruits, packaged fruit]|[]        |
|[mayonnaise, butter, rolls]                      |[]        |
+-------------------------------------------------+----------+
```

Graph Analytics

1.Graph Analytics

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

# Create SparkSession with GraphFrames
# สร้าง SparkSession พร้อม GraphFrames (กำหนด memory ของ driver เป็น 4GB และเพิ่ม package graphframes ที่ใช้ version ที่เข้ากันได้กับ Spark 3.0)
spark = SparkSession.builder \
    .appName("GraphAnalytics") \
    .config("spark.driver.memory", "4g") \
    .config("spark.jars.packages", "graphframes:graphframes:0.8.2-spark3.0-s_2.12") \
    .getOrCreate()

# Create vertices DataFrame
# สร้าง DataFrame ของ vertices (จุดในกราฟ) พร้อมกับระบุข้อมูล 'id' (ชื่อ) และ 'age' (อายุ)
vertices = spark.createDataFrame([
    ("Alice", "45"),
    ("Jacob", "43"),
    ("Roy", "21"),
    ("Ryan", "49"),
    ("Emily", "24"),
    ("Sheldon", "52"),
], ["id", "age"])

# Create edges DataFrame
# สร้าง DataFrame ของ edges (เส้นเชื่อมในกราฟ) โดยระบุ 'src' (จุดเริ่ม), 'dst' (จุดปลาย), และ 'relation' (ความสัมพันธ์)
edges = spark.createDataFrame([
    ("Sheldon", "Alice", "Sister"),
    ("Alice", "Jacob", "Husband"),
    ("Emily", "Jacob", "Father"),
    ("Ryan", "Alice", "Friend"),
    ("Alice", "Emily", "Daughter"),
    ("Alice", "Roy", "Son"),
    ("Jacob", "Roy", "Son"),
], ["src", "dst", "relation"])

# Create GraphFrame
# สร้างกราฟด้วย GraphFrame โดยใช้ vertices และ edges ที่สร้างไว้ก่อนหน้า
try:
    graph = GraphFrame(vertices, edges)
    print("GraphFrame created successfully.")
except Exception as e:
    print(f"Error creating GraphFrame: {e}")

# Show vertices and edges
# แสดงข้อมูล vertices และ edges ที่มีอยู่ในกราฟ
print("Vertices:")
vertices.show()
```

```
print("Edges:")
edges.show()

# Group and order the nodes and edges
# จัดกลุ่ม edges ตาม 'src' และ 'dst' พร้อมทั้งนับจำนวนและเรียงลำดับตามจำนวนที่สูงสุด
grouped_edges = graph.edges.groupBy("src", "dst").count().orderBy(desc("count"))
print("Grouped Edges:")
grouped_edges.show(5)

# Filter the GraphFrame
# กรอง edges เพื่อแสดงเฉพาะเส้นเชื่อมที่ 'src' เป็น 'Alice' หรือ 'dst' เป็น 'Jacob' และเรียงลำดับตามจำนวนที่มากที่สุด
filtered_edges = graph.edges.where("src = 'Alice' OR dst = 'Jacob'").groupBy("src", "dst").count().orderBy(desc("count"))
print("Filtered Edges:")
filtered_edges.show(5)

# Create a subgraph
# สร้าง subgraph ที่กรองเฉพาะ edges ที่ 'src' เป็น 'Alice' หรือ 'dst' เป็น 'Jacob'
subgraph_query = graph.edges.where("src = 'Alice' OR dst = 'Jacob'")
subgraph = GraphFrame(graph.vertices, subgraph_query)
print("Subgraph Edges:")
subgraph.edges.show()

# Find motifs
# ค้นหา motifs (รูปแบบในกราฟ) โดยใช้ syntax `(a) - [ab] -> (b)` เพื่อค้นหาความสัมพันธ์ระหว่างจุด 'a' และ 'b'
motifs = graph.find("(a) - [ab] -> (b)")
print("Motifs:")
motifs.show()

# PageRank
# ใช้ PageRank algorithm เพื่อคำนวณคะแนนของจุดในกราฟ โดยใช้ค่า resetProbability เป็น 0.15 และวนซ้ำ 5 รอบ
rank = graph.pageRank(resetProbability=0.15, maxIter=5)
print("PageRank:")
rank.vertices.orderBy(desc("pagerank")).show(5)

# In-Degree and Out-Degree
# คำนวณ in-degree (จำนวน edges ที่เข้าสู่แต่ละจุด) และเรียงลำดับตามจำนวนสูงสุด
in_degree = graph.inDegrees
print("In-Degree:")
in_degree.orderBy(desc("inDegree")).show(5)

# คำนวณ out-degree (จำนวน edges ที่ออกจากแต่ละจุด) และเรียงลำดับตามจำนวนสูงสุด
out_degree = graph.outDegrees
print("Out-Degree:")
out_degree.orderBy(desc("outDegree")).show(5)

  # Connected Components
  # ค้นหา connected components (กลุ่มของจุดที่เชื่อมต่อถึงกัน) ในกราฟ
  try:
      connected_components = graph.connectedComponents()
      print("Connected Components:")
      connected_components.show()
  except Exception as e:
      print(f"Error calculating connected components: {e}")

  # Strongly Connected Components
  # ค้นหา strongly connected components (กลุ่มของจุดที่มีการเชื่อมต่อถึงกันทั้งไปและกลับ) โดยใช้การวนซ้ำ 5 รอบ
  scc = graph.stronglyConnectedComponents(maxIter=5)
  print("Strongly Connected Components:")
  scc.show()

  # Breadth-First Search (BFS)
  # ค้นหาเส้นทางที่สั้นที่สุดจากจุดที่ id เป็น 'Alice' ไปยังจุดที่ id เป็น 'Roy' โดยกำหนดความยาวสูงสุดของเส้นทางเป็น 2
  bfs_result = graph.bfs(fromExpr="id = 'Alice'", toExpr="id = 'Roy'", maxPathLength=2)
  print("BFS Result from id 'Alice' to id 'Roy' with maxPathLength = 2:")
  bfs_result.show()

  # Stop SparkSession when done
  # หยุดการทำงานของ SparkSession หลังจากเสร็จสิ้นกระบวนการทั้งหมด
  spark.stop()
```

Output Graph Analytics

```
GraphFrame created successfully.
Vertices:
+-------+---+
|     id|age|
+-------+---+
|  Alice| 45|
|  Jacob| 43|
|    Roy| 21|
|   Ryan| 49|
|  Emily| 24|
|Sheldon| 52|
+-------+---+

Edges:
+-------+-----+--------+
|    src|  dst|relation|
+-------+-----+--------+
|Sheldon|Alice|  Sister|
|  Alice|Jacob| Husband|
|  Emily|Jacob|  Father|
|   Ryan|Alice|  Friend|
|  Alice|Emily|Daughter|
|  Alice|  Roy|     Son|
|  Jacob|  Roy|     Son|
+-------+-----+--------+

Grouped Edges:
+-------+-----+-----+
|    src|  dst|count|
+-------+-----+-----+
|  Alice|Jacob|    1|
|Sheldon|Alice|    1|
|  Emily|Jacob|    1|
|  Alice|Emily|    1|
|  Alice|  Roy|    1|
+-------+-----+-----+
only showing top 5 rows

Filtered Edges:
+-----+-----+-----+
|  src|  dst|count|
+-----+-----+-----+
|Alice|Jacob|    1|
|Emily|Jacob|    1|
|Alice|Emily|    1|
|Alice|  Roy|    1|
+-----+-----+-----+

Subgraph Edges:
+-----+-----+--------+
|  src|  dst|relation|
+-----+-----+--------+
|Alice|Jacob| Husband|
|Emily|Jacob|  Father|
|Alice|Emily|Daughter|
|Alice|  Roy|     Son|
+-----+-----+--------+
```

```
Motifs:
+-------------+--------------------+-----------+
|            a|                  ab|          b|
+-------------+--------------------+-----------+
|  {Jacob, 43}|   {Jacob, Roy, Son}|  {Roy, 21}|
|  {Alice, 45}|   {Alice, Roy, Son}|  {Roy, 21}|
|  {Emily, 24}|{Emily, Jacob, Fa...|{Jacob, 43}|
|  {Alice, 45}|{Alice, Jacob, Hu...|{Jacob, 43}|
|{Sheldon, 52}|{Sheldon, Alice, ...|{Alice, 45}|
|   {Ryan, 49}|{Ryan, Alice, Fri...|{Alice, 45}|
|  {Alice, 45}|{Alice, Emily, Da...|{Emily, 24}|
+-------------+--------------------+-----------+

PageRank:
+-------+---+-------------------+
|     id|age|           pagerank|
+-------+---+-------------------+
|    Roy| 21| 1.9089989375092518|
|  Jacob| 43| 1.3728466605994618|
|  Alice| 45|  1.135192093597289|
|  Emily| 24| 0.7420792759997091|
|Sheldon| 52|0.42044151614714403|
+-------+---+-------------------+
only showing top 5 rows

In-Degree:
+-----+--------+
|   id|inDegree|
+-----+--------+
|Jacob|       2|
|Alice|       2|
|  Roy|       2|
|Emily|       1|
+-----+--------+

Out-Degree:
+-------+---------+
|     id|outDegree|
+-------+---------+
|  Alice|        3|
|Sheldon|        1|
|  Emily|        1|
|  Jacob|        1|
|   Ryan|        1|
+-------+---------+
```

2.Graph Analytics And Power BI

อันนี้ไม่น่าจะสอบเพราะเทสไปแล้ว

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

# สร้าง SparkSession และกำหนดการตั้งค่าเพื่อใช้ GraphFrames
spark = SparkSession.builder \
    .appName("Graph Analytics Assignment") \
    .config("spark.jars.packages", "graphframes:graphframes:0.8.2-spark3.0-s_2.12") \
    .getOrCreate()

# อ่านข้อมูลเส้นทางการบินจากไฟล์ CSV เป็น DataFrame
airline_routes_df = spark.read.csv("airline_routes.csv", header=True, inferSchema=True)

# แสดงข้อมูลที่อ่านมาเพื่อดูโครงสร้างของ DataFrame
airline_routes_df.show()

# สร้าง DataFrame ของ vertices จากคอลัมน์ 'source_airport' และตั้งชื่อคอลัมน์เป็น 'id'
vertices_df = airline_routes_df.select("source_airport").withColumnRenamed("source_airport", "id").distinct()

# สร้าง DataFrame ของ edges จากคอลัมน์ 'source_airport' และ 'destination_airport'
# เปลี่ยนชื่อคอลัมน์เป็น 'src' และ 'dst' เพื่อใช้กับ GraphFrame
edges_df = airline_routes_df.select("source_airport", "destination_airport") \
    .withColumnRenamed("source_airport", "src") \
    .withColumnRenamed("destination_airport", "dst")

# แสดงข้อมูล vertices และ edges
vertices_df.show()
edges_df.show()

# สร้าง GraphFrame โดยใช้ vertices และ edges ที่สร้างขึ้น
graph = GraphFrame(vertices_df, edges_df)

# แสดงจำนวน vertices และ edges ในกราฟ
print("Number of vertices:", graph.vertices.count())
print("Number of edges:", graph.edges.count())

# จัดกลุ่ม edges ตาม 'src' และ 'dst' พร้อมนับจำนวน, กรองเฉพาะเส้นทางที่นับได้มากกว่า 5, และเรียงลำดับตามจำนวนจากมากไปน้อย
# เพิ่มคอลัมน์สีสำหรับ 'src' และ 'dst' เพื่อการแสดงผล
grouped_edges_df = graph.edges.groupBy("src", "dst").count() \
    .filter(col("count") > 5) \
    .orderBy(desc("count")) \
    .withColumn("source_color", lit("#3358FF")) \
    .withColumn("destination_color", lit("#FF3F33"))

# แสดงผล grouped edges ที่ได้
grouped_edges_df.show()

 # บันทึกผลลัพธ์ grouped edges ลงในไฟล์ CSV
 grouped_edges_df.write.csv("output.csv", mode="overwrite", header=True)
```

Output Graph Analytics And Power BI

อันนี้ไม่น่าจะสอบเพราะเทสไปแล้ว

```
Number of vertices: 3409
Number of edges: 67663
+---+---+-----+------------+-----------------+
|src|dst|count|source_color|destination_color|
+---+---+-----+------------+-----------------+
|ORD|ATL|   20|     #3358FF|          #FF3F33|
|ATL|ORD|   19|     #3358FF|          #FF3F33|
|HKT|BKK|   13|     #3358FF|          #FF3F33|
|ORD|MSY|   13|     #3358FF|          #FF3F33|
|JFK|LHR|   12|     #3358FF|          #FF3F33|
|CAN|HGH|   12|     #3358FF|          #FF3F33|
|MIA|ATL|   12|     #3358FF|          #FF3F33|
|LHR|JFK|   12|     #3358FF|          #FF3F33|
|ATL|MIA|   12|     #3358FF|          #FF3F33|
|HKG|BKK|   12|     #3358FF|          #FF3F33|
|DOH|BAH|   12|     #3358FF|          #FF3F33|
|AUH|MCT|   12|     #3358FF|          #FF3F33|
|BKK|HKG|   12|     #3358FF|          #FF3F33|
|KGL|EBB|   11|     #3358FF|          #FF3F33|
|MSY|JFK|   11|     #3358FF|          #FF3F33|
|JFK|CDG|   11|     #3358FF|          #FF3F33|
|ATL|DEN|   11|     #3358FF|          #FF3F33|
|JFK|MSY|   11|     #3358FF|          #FF3F33|
|LAX|LHR|   11|     #3358FF|          #FF3F33|
|LHR|LAX|   11|     #3358FF|          #FF3F33|
+---+---+-----+------------+-----------------+
only showing top 20 rows
```

Text Analytics

Text Analytics

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import IntegerType
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

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

# อ่านข้อมูลจากไฟล์ CSV (แทนที่ 'reviews_rated.csv' ด้วยไฟล์ที่ต้องการ)
data = spark.read.csv("reviews_rated.csv", header=True, inferSchema=True)

# เลือกเฉพาะคอลัมน์ "Review Text" และ "Rating" แล้วเปลี่ยนชื่อคอลัมน์ "Review Text" เป็น "review_text" และแปลงประเภทข้อมูล "Rating" เป็น IntegerType
data = data.select(data["Review Text"].alias("review_text"), data["Rating"].cast(IntegerType()).alias("rating"))

# ลบแถวที่มีค่า missing
data = data.na.drop()
data.show(5)

# ขั้นตอนที่ 1: Tokenizer แยกคำในรีวิวออกเป็นคำ (tokens)
tokenizer = Tokenizer(inputCol="review_text", outputCol="words")

# ขั้นตอนที่ 2: StopWordsRemover ลบคำที่ใช้บ่อยแต่ไม่มีความหมายสำคัญ (เช่น "the", "is")
stopword_remover = StopWordsRemover(inputCol="words", outputCol="meaningful_words")

# ขั้นตอนที่ 3: HashingTF แปลงคำที่มีความหมายให้เป็นตัวเลข (features) โดยใช้ Term Frequency
hashing_tf = HashingTF(inputCol="meaningful_words", outputCol="features")

# สร้าง Pipeline เพื่อเรียกใช้ขั้นตอนการประมวลผลที่กำหนดไว้ตามลำดับ
pipeline = Pipeline(stages=[tokenizer, stopword_remover, hashing_tf])

# แบ่งข้อมูลออกเป็นชุดข้อมูลฝึก (80%) และชุดข้อมูลทดสอบ (20%)
train_data, test_data = data.randomSplit([0.8, 0.2], seed=1234)

# เรียกใช้ Pipeline กับชุดข้อมูลฝึก
pipeline_model = pipeline.fit(train_data)

# แปลงข้อมูลฝึกและข้อมูลทดสอบด้วย pipeline ที่ได้สร้างไว้
train_transformed = pipeline_model.transform(train_data)
test_transformed = pipeline_model.transform(test_data)

# แสดงตัวอย่างข้อมูลที่แปลงแล้วของชุดข้อมูลฝึก
train_transformed.select("meaningful_words", "features", "rating").show(5)
 # สร้างโมเดล Logistic Regression สำหรับการจำแนกประเภทโดยใช้คอลัมน์ 'features' เป็นตัวทำนายและ 'rating' เป็นตัว label
 log_reg = LogisticRegression(labelCol="rating", featuresCol="features")

 # ฝึกโมเดล Logistic Regression ด้วยชุดข้อมูลฝึกที่แปลงแล้ว
 log_reg_model = log_reg.fit(train_transformed)

 # ทำนายผลลัพธ์ด้วยชุดข้อมูลทดสอบที่แปลงแล้ว
 predictions = log_reg_model.transform(test_transformed)

 # แสดงผลการทำนาย
 predictions.select("meaningful_words", "rating", "prediction").show(5)

 # ประเมินโมเดลโดยใช้ MulticlassClassificationEvaluator เพื่อคำนวณค่าความแม่นยำ
 evaluator = MulticlassClassificationEvaluator(labelCol="rating", predictionCol="prediction", metricName="accuracy")

 # คำนวณความแม่นยำของโมเดล
 accuracy = evaluator.evaluate(predictions)
 print(f"Accuracy: {accuracy}")
```

Output Text Analytics

```
+--------------------+------+
|         review_text|rating|
+--------------------+------+
|I registered on t...|     1|
|Had multiple orde...|     1|
|I informed these ...|     1|
|I have bought fro...|     1|
|If I could give a...|     1|
+--------------------+------+
only showing top 5 rows

+--------------------+--------------------+------+
|    meaningful_words|            features|rating|
+--------------------+--------------------+------+
|[, , , amazon, cu...|(262144,[876,2564...|     1|
|[, amazon, take, ...|(262144,[5381,172...|     2|
|[, amazon, waste,...|(262144,[13130,31...|     1|
|[, ordered, two, ...|(262144,[9479,162...|     1|
|[, it2517516708as...|(262144,[10049,10...|     1|
+--------------------+--------------------+------+
only showing top 5 rows

+--------------------+------+----------+
|    meaningful_words|rating|prediction|
+--------------------+------+----------+
|[, amazon, custom...|     5|       5.0|
|[, default, sure,...|     1|       4.0|
|["""parcel, hande...|     1|       1.0|
|["""the, earth's,...|     1|       1.0|
|["""we, publish, ...|     1|       1.0|
+--------------------+------+----------+
only showing top 5 rows

Accuracy: 0.6086341118188252
```

https://colab.research.google.com/drive/1UCfIcUNkPLUeLQ3N3Sy4ONGBAkxZV3Pw?usp=sharing

curl -L -o Bigkuma.zip
https://github.com/thanormsaksudsee/BIGDATA/archive/refs/heads/main.zip