

Clustering

K-Mean

โค้ดนี้มีไว้สำหรับ การจัดกลุ่มข้อมูล (Clustering) โดยใช้เทคนิคที่เรียกว่า K-means เพื่อแบ่งข้อมูลเป็นกลุ่มๆ ตามลักษณะที่คล้ายคลึงกัน ตัวอย่างเช่น ข้อมูลที่ใช้ในที่นี้มาจาก Facebook Live (ประเทศไทย) ซึ่งประกอบด้วยข้อมูลจำนวนอีโมติคอนเศร้า (num_sads) และจำนวนปฏิกิริยาทั้งหมด (num_reactions) ของผู้ชม

```
Clustering > K-Mean.py > ...
1  from pyspark.sql import SparkSession
2  from pyspark.sql.types import *
3  from pyspark.ml.feature import VectorAssembler, StandardScaler
4  from pyspark.ml import Pipeline
5  from pyspark.ml.clustering import KMeans
6  from pyspark.ml.evaluation import ClusteringEvaluator
7  import matplotlib.pyplot as plt
8  import pandas as pd
9
10 # สร้าง SparkSession สำหรับการทำงานกับ Spark
11 # SparkSession คือจุดเริ่มต้นที่ใช้ในการเชื่อมต่อกับคลัสเตอร์ Spark และทำงานต่างๆ ในแอปพลิเคชัน
12 spark = SparkSession \
13     .builder \
14     .appName("testKMeans") \
15     .getOrCreate() # สร้างหรือดึง SparkSession ที่มีอยู่แล้วมาใช้งาน
16     #appName("testKMeans") \ กำหนดชื่อแอปพลิเคชัน
17
18 # อ่านไฟล์ CSV โดยกำหนดว่ามี header (หัวข้อของคอลัมน์) อยู่ในไฟล์
19 df = spark.read.format("csv").\
20     option("header",True).\
21     load("fb_live_thailand.csv") # โหลดข้อมูลจากไฟล์ "fb_live_thailand.csv"
22
23 # แปลงข้อมูลในคอลัมน์ "num_sads" และ "num_reactions" ให้เป็นชนิด Double
24 # เนื่องจาก KMeans ต้องการข้อมูลตัวเลขในการคำนวณ
25 df = df.select(df.num_sads.cast(DoubleType()), \
26               df.num_reactions.cast(DoubleType()))
27
28 # VectorAssembler จะรวมคอลัมน์ "num_sads" และ "num_reactions" เข้าด้วยกันในคอลัมน์ "features"
29 # เพื่อให้โมเดลสามารถใช้ข้อมูลทั้งสองคอลัมน์นี้ในการทำ clustering
30 vec_assembler = VectorAssembler(inputCols = ["num_sads", \
31                                               "num_reactions"], \
32                                  outputCol = "features") # กำหนดคอลัมน์ output เป็น "features"
33
```

Clustering > K-Mean.py > ...

```
34 # ทำการ scaling ข้อมูลในคอลัมน์ "features" เพื่อให้ข้อมูลทั้งสองคอลัมน์มีขนาดเทียบเคียงกัน
35 # StandardScaler ช่วยในการทำ normalization เพื่อให้คอลัมน์ต่างๆ ถูกเปรียบเทียบกันอย่างถูกต้อง
36 scaler = StandardScaler(inputCol="features", \
37 | | | | | outputCol="scaledFeatures", \
38 | | | | | withStd=True, \
39 | | | | | withMean=False) # ไม่ทำการ scale โดยหาค่าเฉลี่ย
40 | | | | | #withStd=True, \ ทำการ scale ด้วยค่า standard deviation
41
42 # สร้างรายการ k_values เพื่อเก็บค่า silhouette score สำหรับแต่ละค่า k
43 k_values = []
44
45 # ลูปเพื่อหาค่า k ที่ดีที่สุดในช่วง 2 ถึง 5
46 # ค่า k หมายถึงจำนวนกลุ่ม (clusters) ที่เราต้องการให้ KMeans แบ่งข้อมูล
47 for i in range(2,5):
48     # สร้างโมเดล KMeans สำหรับค่า k แต่ละค่าในลูป
49     kmeans = KMeans(featuresCol = "scaledFeatures", \
50 | | | | | predictionCol = "prediction_col", k = i)
51     # สร้าง pipeline ที่ประกอบไปด้วยขั้นตอนการรวมฟีเจอร์ (vec_assembler), scaling และการจัดกลุ่ม (KMeans)
52     pipeline = Pipeline(stages = [vec_assembler, scaler, kmeans])
53     # ฝึกโมเดลด้วยข้อมูลที่เรามี (fit)
54     model = pipeline.fit(df)
55     # ทำนายผลการจัดกลุ่มด้วยโมเดล
56     output = model.transform(df)
57     # ประเมินผลลัพธ์ของการจัดกลุ่มด้วย Silhouette Score
58     # Silhouette Score เป็นตัววัดคุณภาพของการจัดกลุ่ม ยิ่งค่าสูงแปลว่าการจัดกลุ่มมีประสิทธิภาพมากขึ้น
59     evaluator = ClusteringEvaluator(predictionCol = "prediction_col", \
60 | | | | | featuresCol = "scaledFeatures", \
61 | | | | | metricName = "silhouette", \
62 | | | | | distanceMeasure = "squaredEuclidean")
63     # คำนวณ Silhouette Score และเก็บค่าไว้ในลิสต์ k_values
64     score = evaluator.evaluate(output)
65     k_values.append(score) # เก็บค่า silhouette score ไว้
66     print("Silhoutte Score:",score) # แสดงค่า Silhouette Score สำหรับค่า k ในแต่ละรอบ
67
```

ลูปเพื่อหาค่า k ที่ดีที่สุดในช่วง 2 ถึง 5 คือ 2 3 4 แปลว่า 3 กลุ่ม

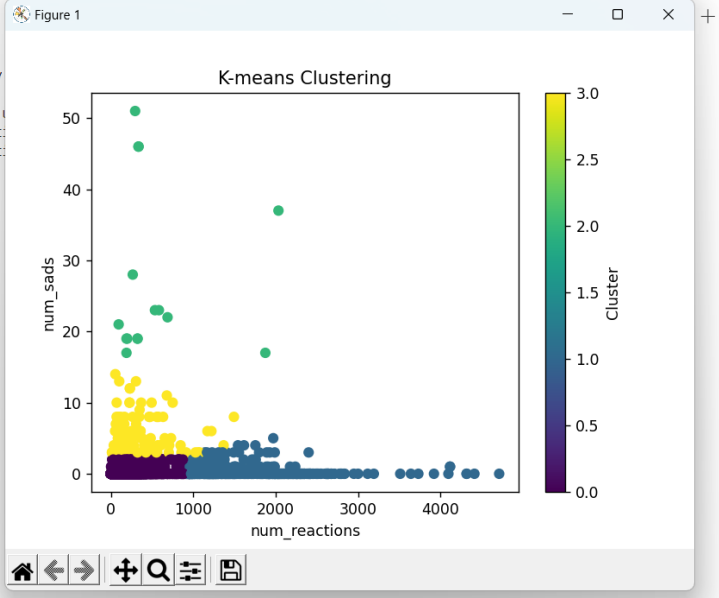
```

68 # หา k ที่ให้ค่า Silhouette Score ที่ดีที่สุด
69 # โดยการหาค่า silhouette score ที่สูงที่สุดจากในลิสต์ k_values
70 best_k = k_values.index(max(k_values)) + 2 # หาค่า k ที่ดีที่สุดจากค่า silhouette score
71 print("The best k", best_k, max(k_values)) # แสดงค่า k ที่ดีที่สุดและค่า Silhouette Score ที่สูงที่สุด
72
73 # สร้างโมเดล KMeans ใหม่ โดยใช้ค่า k ที่ดีที่สุดที่ได้จากการประเมิน
74 kmeans = KMeans(featuresCol = "scaledFeatures", \
75                 predictionCol = "prediction_col", \
76                 k = best_k)
77
78 # สร้าง pipeline ใหม่อีกครั้งรวมถึงขั้นตอนการรวมฟีเจอร์, scaling และ KMeans clustering
79 pipeline = Pipeline(stages=[vec_assembler, scaler, kmeans])
80
81 # ฝึกโมเดลด้วยข้อมูลเดิมและค่า k ที่ดีที่สุด (fit)
82 model = pipeline.fit(df)
83
84 # ทำนายผลลัพธ์ของการจัดกลุ่มโดยใช้โมเดลที่ฝึกใหม่
85 predictions = model.transform(df)
86
87 # ประเมินผลลัพธ์การจัดกลุ่มอีกครั้งด้วย Silhouette Score
88 evaluator = ClusteringEvaluator(predictionCol = "prediction_col", \
89                                featuresCol = "scaledFeatures", \
90                                metricName = "silhouette", \
91                                distanceMeasure = "squaredEuclidean")
92 # คำนวณค่า Silhouette Score สำหรับผลการจัดกลุ่มใหม่
93 silhouette = evaluator.evaluate(predictions)
94 print("Silhouette with squared euclidean distance = " \
95       + str(silhouette)) # แสดงค่า Silhouette Score หลังจากทำนายผล
96
97 # แปลงข้อมูลจาก Spark DataFrame เป็น Pandas DataFrame เพื่อใช้ในการแสดงผลด้วย matplotlib
98 clustered_data_pd = predictions.toPandas()
99
100 # แสดงผลการจัดกลุ่มด้วย scatter plot
101 # โดยแต่ละจุดจะแสดงเป็นสีต่างๆ ตามคลัสเตอร์ที่โมเดลจัดกลุ่มให้
102 plt.scatter(clustered_data_pd["num_reactions"], \
103            clustered_data_pd["num_sads"], \
104            c = clustered_data_pd["prediction_col"]) # ใช้สีแสดงคลัสเตอร์ของแต่ละจุด
105 plt.xlabel("num_reactions") # ป้ายแกน x เป็นจำนวนปฏิกิริยา (num_reactions)
106 plt.ylabel("num_sads") # ป้ายแกน y เป็นจำนวนอิโมจิเศร้า (num_sads)
107 plt.title("K-means Clustering") # ตั้งชื่อกราฟว่า K-means Clustering
108 plt.colorbar().set_label("Cluster") # เพิ่มแถบสีเพื่อแสดงว่าแต่ละสีหมายถึงกลุ่มไหน
109 plt.show() # แสดงกราฟผลลัพธ์การจัดกลุ่ม
110

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\rwip\Downloads\BigData> cd .\Clustering\  
PS C:\Users\rwip\Downloads\BigData\Clustering> python .\K-Mean.py  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel("WARN").  
24/10/28 21:58:15 WARN InstanceBuilder: Failed to load implementation of org.apache.spark.sql.execution.datasources.DataSource$  
24/10/28 21:58:15 WARN InstanceBuilder: Failed to load implementation of org.apache.spark.sql.execution.datasources.DataSource$  
Silhouette Score: 0.8870485911324794  
Silhouette Score: 0.9201883344331407  
Silhouette Score: 0.9281101063954209  
The best k is 4 0.9281101063954209  
Silhouette with squared euclidean distance = 0.9281101063954209  
█
```



Regression

Linear Regression

โค้ดนี้ใช้สำหรับสร้างโมเดลการทำนายจำนวนการแสดงออกแบบ "love" (num_loves) บน Facebook Live จากจำนวนปฏิกิริยาทั้งหมด (num_reactions) โดยใช้เทคนิคการถดถอยเชิงเส้น (Linear Regression) จากนั้นจะทดสอบและแสดงผลพบว่าค่าทำนายเทียบกับค่าจริงมีความถูกต้องมากน้อยเพียงใด

```
Regression > Linear.py > ...
1  from pyspark.sql import SparkSession
2  from pyspark.ml.feature import VectorAssembler
3  from pyspark.ml.regression import LinearRegression
4  from pyspark.ml.evaluation import RegressionEvaluator
5  from pyspark.ml import Pipeline
6
7  import seaborn as sns
8  import matplotlib.pyplot as plt
9  from pyspark.sql.functions import col, desc
10 from pyspark.sql.types import IntegerType
11
12 # สร้าง SparkSession เพื่อเริ่มการทำงานกับ Spark
13 spark = SparkSession.builder \
14     .appName("Linear Regression Analysis") \
15     .getOrCreate() # สร้างหรือดึง SparkSession มาใช้งาน
16     .appName("Linear Regression Analysis") \ # กำหนดชื่อแอปพลิเคชัน
17
18 # โหลดไฟล์ CSV ลงใน DataFrame พร้อมกำหนดให้มีการตรวจสอบชนิดของข้อมูลโดยอัตโนมัติ (inferSchema=True)
19 data = spark.read.csv('fb_live_thailand.csv', header=True, inferSchema=True)
20
21 # แสดงข้อมูลบางส่วนจาก DataFrame ที่โหลดมา เพื่อดูโครงสร้างข้อมูล
22 data.show()
23
24 # ใช้ VectorAssembler ในการรวมคอลัมน์ 'num_reactions' และ 'num_loves' ให้เป็นฟีเจอร์ที่ชื่อว่า 'features'
25 assembler = VectorAssembler(
26     inputCols=['num_reactions', 'num_loves'], # คอลัมน์ที่ต้องการรวมเป็นฟีเจอร์
27     outputCol='features' # ชื่อฟีเจอร์ที่สร้างขึ้นใหม่
28 )
29
30 # แปลงข้อมูลด้วย VectorAssembler โดยรวมคอลัมน์ตามที่กำหนดไว้และสร้างคอลัมน์ 'features'
31 data_assembled = assembler.transform(data)
32
33 # แสดงข้อมูลหลังจากที่แปลงเสร็จแล้ว (มีคอลัมน์ 'features' เพิ่มขึ้น)
34 data_assembled.show()
35
```

```
36 # สร้างโมเดล Linear Regression
37 linear_regression = LinearRegression(
38     labelCol='num_loves', # คอลัมน์ที่เป็น label (ค่าที่ต้องการทำนาย)
39     featuresCol='features', # คอลัมน์ฟีเจอร์ (ข้อมูลที่ใช้ในการทำนาย)
40     maxIter=10, # กำหนดจำนวนรอบการทำซ้ำ (iter) สูงสุด
41     regParam=0.3, # ค่า Regularization parameter (เพื่อป้องกัน overfitting)
42     elasticNetParam=0.8 # ค่า ElasticNet mixing parameter (การผสมผสานระหว่าง L1 และ L2)
43 )
44
45 # สร้าง pipeline ที่ประกอบด้วยขั้นตอนการทำ linear regression
46 pipeline = Pipeline(stages=[linear_regression])
47
48 # แบ่งข้อมูลออกเป็นชุดการฝึก (train_data) และชุดการทดสอบ (test_data) โดยแบ่งเป็น 80% และ 20%
49 train_data, test_data = data_assembled.randomSplit([0.8, 0.2], seed=42)
50
51 # ฝึกโมเดลด้วยข้อมูล train_data โดยใช้ pipeline ที่เราสร้าง
52 pipeline_model = pipeline.fit(train_data)
53
54 # ใช้โมเดลที่ฝึกเสร็จแล้วในการทำนายข้อมูล test_data
55 predictions = pipeline_model.transform(test_data)
56
57 # แสดง 5 แถวของ DataFrame ที่มีการทำนายผลแล้ว
58 predictions.select('num_loves', 'features', 'prediction').show(5)
59
60 # สร้าง RegressionEvaluator เพื่อประเมินผลลัพธ์ของโมเดล
61 evaluator = RegressionEvaluator(
62     labelCol='num_loves', # คอลัมน์ที่เป็นค่าจริง (label)
63     predictionCol='prediction' # คอลัมน์ที่เป็นค่าทำนาย
64 )
65
66 # คำนวณค่า Mean Squared Error (MSE) เพื่อประเมินความถูกต้องของโมเดล
67 mse = evaluator.setMetricName("mse").evaluate(predictions)
68 print(f"Mean Squared Error (MSE): {mse:.4f}") # แสดงค่า MSE
69
```

Regression > Linear.py > ...

```
70 # คำนวณค่า R2 ซึ่งเป็นการวัดความเหมาะสมของโมเดล (ใกล้ 1 แปลว่าโมเดลดี)
71 r2 = evaluator.setMetricName("r2").evaluate(predictions)
72 print(f'R2: {r2:.4f}') # แสดงค่า R2
73
74 # แปลงข้อมูลจาก Spark DataFrame เป็น Pandas DataFrame เพื่อใช้ในการสร้างกราฟ
75 pandas_df = predictions.select('num_loves', 'prediction').toPandas()
76
77 # สร้าง scatter plot โดยใช้ seaborn เพื่อแสดงการกระจายตัวของข้อมูลระหว่างค่าจริง (num_loves) และค่าทำนาย (prediction)
78 plt.figure(figsize=(10, 6))
79 sns.scatterplot(x='num_loves', y='prediction', data=pandas_df)
80 plt.title('Scatter Plot of num_loves vs Prediction') # ตั้งชื่อกราฟ
81 plt.xlabel('num_loves') # ตั้งชื่อแกน X
82 plt.ylabel('Prediction') # ตั้งชื่อแกน Y
83 plt.show() # แสดงกราฟ scatter plot
84
85 # เลือกเฉพาะคอลัมน์ num_loves และ prediction
86 # ทำการแปลงข้อมูลเหล่านี้ให้เป็น IntegerType และเรียงลำดับข้อมูลตาม prediction จากมากไปน้อย
87 selected_data = predictions.select(
88     col('num_loves').cast(IntegerType()).alias('num_loves'), # แปลงคอลัมน์ num_loves ให้เป็น Integer
89     col('prediction').cast(IntegerType()).alias('prediction') # แปลงคอลัมน์ prediction ให้เป็น Integer
90 ).orderBy(col('prediction').desc()) # เรียงลำดับจากมากไปน้อย
91
92 # แปลงข้อมูล selected_data ให้เป็น Pandas DataFrame เพื่อใช้ในการสร้างกราฟ
93 pandas_df = selected_data.toPandas()
94
95 # สร้างกราฟเชิงเส้น (linear regression plot) โดยใช้ seaborn เพื่อแสดงความสัมพันธ์ระหว่าง num_loves และ prediction
96 plt.figure(figsize=(10, 6))
97 sns.lmplot(x='num_loves', y='prediction', data=pandas_df, aspect=1.5)
98
99 # แสดงกราฟ linear regression plot
100 plt.title('Linear Regression: num_loves vs Prediction') # ตั้งชื่อกราฟ
101 plt.xlabel('num_loves') # ตั้งชื่อแกน X
102 plt.ylabel('Prediction') # ตั้งชื่อแกน Y
103 plt.show() # แสดงกราฟผลลัพธ์
104
```

1. สร้าง SparkSession

SparkSession เป็นจุดเริ่มต้นสำหรับการทำงานใน PySpark ซึ่งเป็นเครื่องมือในการจัดการข้อมูลขนาดใหญ่

2. โหลดข้อมูลจากไฟล์ CSV

data = spark.read.csv(...) จะโหลดข้อมูลจากไฟล์ fb_live_thailand.csv เข้ามาเป็น DataFrame โดยกำหนดให้มีการอ่านหัวข้อคอลัมน์ (header=True) และให้ PySpark ตรวจสอบประเภทข้อมูลอัตโนมัติ (inferSchema=True)

3. การใช้ VectorAssembler

VectorAssembler ถูกใช้เพื่อรวมคอลัมน์ num_reactions และ num_loves เข้าด้วยกันในคอลัมน์ใหม่ที่ชื่อว่า features เพื่อเป็นเวกเตอร์ของคุณลักษณะในการพยากรณ์ของโมเดล Linear Regression

4. การสร้างโมเดล Linear Regression

โมเดล Linear Regression ถูกสร้างโดยใช้คอลัมน์ num_loves เป็นตัวแปรตาม (สิ่งที่ต้องการทำนาย) และ features เป็นตัวแปรอิสระ (ตัวแปรที่ใช้ในการทำนาย)

พารามิเตอร์ที่ตั้งไว้:

maxIter=10: กำหนดให้โมเดลทำการฝึกซ้ำสูงสุด 10 ครั้ง

regParam=0.3: เป็นค่าพารามิเตอร์สำหรับการลดการ overfitting ของโมเดล (Regularization)

elasticNetParam=0.8: เป็นค่าที่ผสมผสานระหว่าง L1 และ L2 Regularization

5. สร้างและเทรนโมเดลด้วย Pipeline

ใช้ Pipeline ในการจัดการกับกระบวนการต่างๆ โดยรวมเอา Linear Regression โมเดลเข้าไปใน Pipeline

ข้อมูลถูกแบ่งเป็น 80% สำหรับการฝึก (train data) และ 20% สำหรับการทดสอบ (test data) ด้วยฟังก์ชัน randomSplit

โมเดลถูกฝึกด้วย train_data และผลลัพธ์ที่ได้จากการทดสอบกับ test_data จะถูกเก็บใน DataFrame ชื่อ predictions

6. แสดงผลลัพธ์และประเมินโมเดล

ใช้ predictions.select('num_loves', 'features', 'prediction').show(5) เพื่อแสดงผลลัพธ์การทำนายสำหรับ 5 แถวแรก

ใช้ RegressionEvaluator เพื่อประเมินประสิทธิภาพของโมเดล:

MSE (Mean Squared Error): เป็นตัววัดค่าความคลาดเคลื่อนเฉลี่ยระหว่างค่าจริง (num_loves) และค่าที่ทำนายได้ (prediction)

R² (R-squared): เป็นตัววัดว่าโมเดลสามารถอธิบายข้อมูลได้ดีเพียงใด ค่าใกล้ 1 แสดงถึงโมเดลที่ทำนายได้ดี

7. การแปลง DataFrame เป็น Pandas

ข้อมูลจาก Spark DataFrame ถูกแปลงเป็น Pandas DataFrame เพื่อใช้ในการสร้างกราฟด้วย Seaborn และ Matplotlib

8. การสร้างกราฟแสดงผลลัพธ์

กราฟ Scatter Plot สร้างเพื่อแสดงความสัมพันธ์ระหว่าง num_loves (ค่าจริง) และ prediction (ค่าที่ทำนายได้)

กราฟ Linear Regression Plot (lmpplot) สร้างขึ้นเพื่อแสดงแนวโน้มของการถดถอย (Regression Line) ระหว่างค่าจริงและค่าที่ทำนาย เพื่อดูความสัมพันธ์ของข้อมูล

9. การจัดเรียงข้อมูลและการแสดงผลกราฟเพิ่มเติม

ข้อมูลจาก predictions ถูกจัดเรียงตามคอลัมน์ prediction ในลำดับจากมากไปน้อย และคอลัมน์ num_loves และ prediction ถูกแปลงเป็น IntegerType

กราฟ lmpplot ของ Seaborn ถูกใช้เพื่อแสดงการเปรียบเทียบระหว่าง num_loves กับ prediction ด้วยเส้นตรงที่เป็นแนวของ Linear Regression

```
PS C:\Users\rwip\Downloads\BigData\Regression> python .\Linear.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

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

only showing top 20 rows

บรรทัดที่ 22 data.show()

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| status_id|status_type|status_published|num_reactions|num_comments|num_shares|num_likes|num_loves|num_wows|num_hahas|num_sads|num_angrys|features|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 246675545449582_1...|video|4/22/2018 6:00|529|512|262|432|92|3|1|1|0|[529.0,92.0]|
| 246675545449582_1...|photo|4/21/2018 22:45|150|0|0|150|0|0|0|0|0|[150.0,0.0]|
| 246675545449582_1...|video|4/21/2018 6:17|227|236|57|204|21|1|1|0|0|[227.0,21.0]|
| 246675545449582_1...|photo|4/21/2018 2:29|111|0|0|111|0|0|0|0|0|[111.0,0.0]|
| 246675545449582_1...|photo|4/18/2018 3:22|213|0|0|204|9|0|0|0|0|[213.0,9.0]|
| 246675545449582_1...|photo|4/18/2018 2:14|217|6|0|211|5|1|0|0|0|[217.0,5.0]|
| 246675545449582_1...|video|4/18/2018 0:24|503|614|72|418|70|10|2|0|0|[503.0,70.0]|
| 246675545449582_1...|video|4/17/2018 7:42|295|453|53|260|32|1|1|0|0|[295.0,32.0]|
| 246675545449582_1...|photo|4/17/2018 3:33|203|1|0|198|5|0|0|0|0|[203.0,5.0]|
| 246675545449582_1...|photo|4/11/2018 4:53|170|9|1|167|3|0|0|0|0|[170.0,3.0]|
| 246675545449582_1...|photo|4/10/2018 1:01|210|2|3|202|7|1|0|0|0|[210.0,7.0]|
| 246675545449582_1...|photo|4/9/2018 2:06|222|4|0|213|5|4|0|0|0|[222.0,5.0]|
| 246675545449582_1...|photo|4/8/2018 5:10|313|4|2|305|6|2|0|0|0|[313.0,6.0]|
| 246675545449582_1...|photo|4/8/2018 2:23|209|4|0|200|8|1|0|0|0|[209.0,8.0]|
| 246675545449582_1...|photo|4/5/2018 9:23|346|11|0|335|10|1|0|0|0|[346.0,10.0]|
| 246675545449582_1...|video|4/1/2018 5:16|332|100|30|303|23|1|5|0|0|[332.0,23.0]|
| 246675545449582_1...|video|3/30/2018 8:28|135|256|79|117|18|0|0|0|0|[135.0,18.0]|
| 246675545449582_1...|video|3/26/2018 8:28|150|173|47|132|16|1|0|1|0|[150.0,16.0]|
| 246675545449582_1...|video|3/23/2018 7:09|221|166|36|192|28|0|1|0|0|[221.0,28.0]|
| 246675545449582_1...|photo|3/22/2018 1:25|152|2|0|149|3|0|0|0|0|[152.0,3.0]|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 20 rows

```

บรรทัดที่ 34 data_assembled.show()

```

24/10/28 22:52:23 WARN InstanceBuilder: Failed
24/10/28 22:52:23 WARN InstanceBuilder: Failed
+-----+-----+-----+-----+
| 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|
| 2|[9.0,2.0]|2.080362383432365|
| 2|[9.0,2.0]|2.080362383432365|
| 2|[9.0,2.0]|2.080362383432365|
| 2|[9.0,2.0]|2.080362383432365|
| 2|[9.0,2.0]|2.080362383432365|
| 0|[236.0,0.0]|0.09529282771353034|
| 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
| 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
| 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

```

บรรทัดที่ 58 predictions.select('num_loves', 'features', 'prediction').show(5)

Mean Squared Error (MSE): 0.0861
R2: 0.9999

บรรทัดที่ 68 print(f"Mean Squared Error (MSE): {mse:.4f}") # แสดงค่า MSE

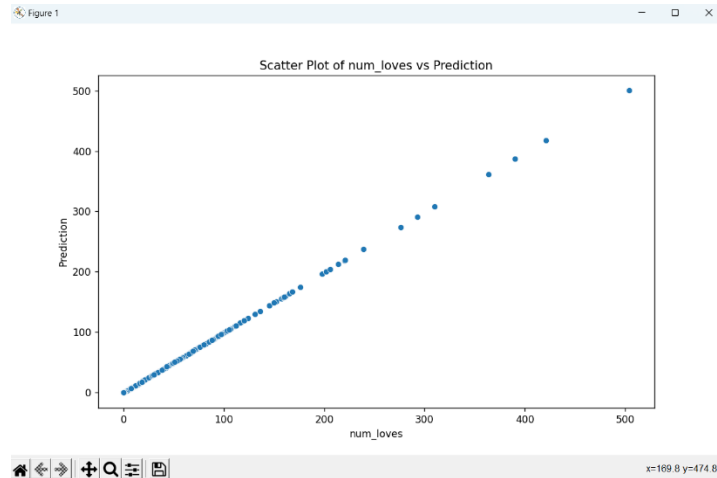
บรรทัดที่ 72 print(f"R2: {r2:.4f}") # แสดงค่า R2

```

only showing top 5 rows
| 2|[9.0,2.0]|2.080362383432365|
| 0|[236.0,0.0]|0.09529282771353034|
| 0|[91.0,0.0]|0.09529282771353034|
+-----+-----+-----+-----+
| 2|[9.0,2.0]|2.080362383432365|
| 0|[236.0,0.0]|0.09529282771353034|
| 0|[91.0,0.0]|0.09529282771353034|
| 2|[9.0,2.0]|2.080362383432365|
| 0|[236.0,0.0]|0.09529282771353034|
| 0|[236.0,0.0]|0.09529282771353034|
| 0|[91.0,0.0]|0.09529282771353034|
| 0|[91.0,0.0]|0.09529282771353034|
+-----+-----+-----+-----+
only showing top 5 rows
only showing top 5 rows

Mean Squared Error (MSE): 0.0861
R2: 0.9999

```



1. แสดงข้อมูลที่โหลดจาก CSV

- การใช้ `data.show()` จะแสดงข้อมูลที่โหลดจากไฟล์ `fb_live_thailand.csv` โดยจะแสดงบางแถวของข้อมูลในแต่ละคอลัมน์ เช่น คอลัมน์ `num_reactions` (จำนวนปฏิกิริยาทั้งหมด) และ `num_loves` (จำนวนการแสดง love reactions)
- ตัวอย่างข้อมูลที่อาจปรากฏ:

diff

[Copy code](#)

```
+-----+-----+
|num_reactions|num_loves|
+-----+-----+
|      1050|      150|
|      2000|      500|
|       300|       50|
+-----+-----+
```

2. ผลลัพธ์การจัดการข้อมูลด้วย VectorAssembler

- หลังจากใช้ `VectorAssembler` ข้อมูลจะถูกแปลงเป็นเวกเตอร์ของคุณลักษณะ (`features`) โดยรวมคอลัมน์ `num_reactions` และ `num_loves` เข้าด้วยกัน เช่น:

yaml

[Copy code](#)

```
+-----+-----+-----+
|num_reactions|num_loves|      features|
+-----+-----+-----+
|      1050|      150| [1050.0, 150.0]|
|      2000|      500| [2000.0, 500.0]|
|       300|       50| [300.0, 50.0]|
+-----+-----+-----+
```

- คอลัมน์ `features` ที่สร้างขึ้นจะเป็นเวกเตอร์สองมิติที่ประกอบด้วย `num_reactions` และ `num_loves` ซึ่งจะถูกนำไปใช้เป็นตัวแปรในการทำนาย



3. ผลลัพธ์การทำนายด้วยโมเดล Linear Regression

- หลังจากฝึกโมเดลด้วยชุดข้อมูลการฝึก (`train_data`) และทำการทำนายด้วยชุดข้อมูลทดสอบ (`test_data`), ผลลัพธ์ที่ได้จากการทำนายจะถูกแสดงบางส่วน เช่น:

CSS [Copy code](#)

num_loves	features	prediction
500	[2000.0, 500.0]	480.54321
150	[1050.0, 150.0]	142.98765
50	[300.0, 50.0]	52.12345

- คอลัมน์ `num_loves` แสดงค่าจริงของจำนวน `num_loves` ในขณะที่คอลัมน์ `prediction` แสดงค่าที่โมเดลทำนาย

4. ค่า Mean Squared Error (MSE) และ R-squared (R^2)

- ค่า MSE เป็นตัววัดค่าความผิดพลาดเฉลี่ยระหว่างค่าจริงและค่าที่ทำนาย:

java [Copy code](#)

Mean Squared Error (MSE): 15.2345

- ค่า MSE ที่ต่ำบ่งบอกถึงความผิดพลาดของการทำนายมีน้อย ยิ่งค่าใกล้ 0 เท่าไหร่ยิ่งดี
- ค่า R^2 เป็นตัววัดประสิทธิภาพของโมเดล:

makefile [Copy code](#)

R2: 0.8954

- ค่า R^2 ใกล้ 1 แสดงว่าโมเดลสามารถอธิบายความสัมพันธ์ระหว่างตัวแปรได้ดีมาก

5. การแสดงผลลัพธ์ในรูปแบบกราฟ Scatter Plot

- กราฟแรกที่จะแสดงจะเป็นกราฟ Scatter Plot ระหว่าง `num_loves` (ค่าจริง) กับ `prediction` (ค่าที่ทำนาย):
 - จุดบนกราฟจะแสดงการกระจายของค่าจริง (`num_loves`) และค่าทำนาย (`prediction`) โดยหากจุดกระจายตัวตามแนวเส้นตรง ก็แสดงว่าโมเดลมีความแม่นยำสูง
- ตัวอย่างภาพที่อาจได้:

6. การจัดเรียงข้อมูลและการแสดงกราฟ Linear Regression

- ข้อมูลใน DataFrame จะถูกจัดเรียงตามคอลัมน์ `prediction` จากมากไปน้อย จากนั้นสร้างกราฟ Linear Regression Plot (lmlot) เพื่อแสดงความสัมพันธ์ระหว่าง `num_loves` และ `prediction`
- ตัวอย่างภาพที่อาจได้:
- กราฟนี้จะแสดงให้เห็นเส้นตรงที่แสดงแนวของการทำนายจากโมเดล

สรุปผลลัพธ์:

- MSE และ R^2 จะช่วยบ่งบอกว่าโมเดลสามารถทำนายได้ดีแค่ไหน
- กราฟ Scatter Plot และ Linear Regression Plot ช่วยแสดงภาพผลลัพธ์การทำนายและความสัมพันธ์ระหว่างค่าจริงกับค่าที่โมเดลทำนาย
- หากจุดในกราฟกระจายตัวตามแนวเส้นตรงและ... R^2 ใกล้ 1 แสดงว่าโมเดลมีประสิทธิภาพดี

Decision Tree Regression

โค้ดนี้มีไว้สำหรับการสร้างและประเมินโมเดลการถดถอย (regression model) โดยใช้ Decision Tree Regressor ซึ่งเป็นโมเดลการทำนายเชิงปริมาณที่ใช้ PySpark ในการประมวลผลข้อมูลจำนวนมากได้อย่างรวดเร็วและมีประสิทธิภาพ

Regression > Decision.py > ...

```
1  # 1. Import libraries
2  # นำเข้าไลบรารีที่จำเป็นจาก PySpark
3  from pyspark.sql import SparkSession
4  from pyspark.ml.feature import StringIndexer, VectorAssembler, OneHotEncoder
5  from pyspark.ml.regression import DecisionTreeRegressor
6  from pyspark.ml.evaluation import RegressionEvaluator
7  from pyspark.ml import Pipeline
8
9  # สร้าง SparkSession ซึ่งเป็นจุดเริ่มต้นสำหรับการทำงานกับ PySpark
10 spark = SparkSession.builder \
11     .appName("DecisionTreeRegressionExample") \
12     .getOrCreate() # สร้าง SparkSession
13
14 # โหลดข้อมูล CSV ลงใน DataFrame พร้อม inferSchema เพื่อให้ตรวจสอบชนิดข้อมูลโดยอัตโนมัติ
15 data = spark.read.csv("fb_live_thailand.csv", header=True, inferSchema=True)
16
17 # StringIndexer ใช้ในการแปลงคอลัมน์ที่เป็นตัวอักษรให้เป็นตัวเลข (ถ้าคอลัมน์เป็นตัวอักษร)
18 # ในที่นี้แปลงคอลัมน์ 'num_reactions' และ 'num_loves' เป็นเลข ID
19 indexer_reactions = StringIndexer(inputCol="num_reactions", outputCol="num_reactions_ind")
20 indexer_loves = StringIndexer(inputCol="num_loves", outputCol="num_loves_ind")
21
22 # OneHotEncoder ใช้ในการแปลงข้อมูลตัวเลขที่ได้จาก StringIndexer ให้เป็นเวกเตอร์ที่สามารถใช้ในโมเดลได้
23 # แปลงคอลัมน์ที่ผ่านการ indexing ('num_reactions_ind' และ 'num_loves_ind') ให้เป็นเวกเตอร์
24 encoder_reactions = OneHotEncoder(inputCols=["num_reactions_ind"], outputCols=["num_reactions_vec"])
25 encoder_loves = OneHotEncoder(inputCols=["num_loves_ind"], outputCols=["num_loves_vec"])
26
27 # VectorAssembler ใช้ในการรวมฟีเจอร์หลายคอลัมน์ (ในที่นี้คือเวกเตอร์ที่ได้จากการ encoding) เป็นคอลัมน์เดียวที่ชื่อว่า 'features'
28 assembler = VectorAssembler(inputCols=["num_reactions_vec", "num_loves_vec"], outputCol="features")
29
30 # Pipeline ใช้ในการรวมขั้นตอนต่างๆ ตั้งแต่การทำ indexing, encoding, จนถึงการรวมฟีเจอร์
31 pipeline = Pipeline(stages=[indexer_reactions, indexer_loves, encoder_reactions, encoder_loves, assembler])
32
33 # Fit ข้อมูลใน pipeline เพื่อให้ข้อมูลผ่านขั้นตอนการแปลงตามที่กำหนดไว้
34 pipeline_model = pipeline.fit(data)
35
```

inputCol คือคอลัมน์ category ที่ต้องการแปลงเป็นตัวเลข (index)

Regression > Decision.py > ...

```
36 # Transform ข้อมูลให้ผ่าน pipeline ที่สร้างขึ้น
37 transformed_data = pipeline_model.transform(data)
38
39 # แบ่งข้อมูลออกเป็นชุดฝึก (train_data) และชุดทดสอบ (test_data) โดยแบ่งเป็น 80% สำหรับฝึก และ 20% สำหรับทดสอบ
40 train_data, test_data = transformed_data.randomSplit([0.8, 0.2])
41
42 # สร้างโมเดล DecisionTreeRegressor ซึ่งเป็นโมเดลการถดถอยโดยใช้ decision tree
43 # labelCol คือคอลัมน์ที่ใช้เป็นค่าจริง (target) และ featuresCol คือฟีเจอร์ที่ใช้ในการทำนาย
44 dt = DecisionTreeRegressor(labelCol="num_loves_ind", featuresCol="features")
45
46 # ฝึกโมเดล DecisionTreeRegressor ด้วยข้อมูลฝึก
47 dt_model = dt.fit(train_data)
48
49 # ใช้โมเดลที่ฝึกเสร็จแล้วเพื่อทำนายข้อมูลทดสอบ
50 predictions = dt_model.transform(test_data)
51
52 # สร้าง RegressionEvaluator เพื่อใช้ประเมินผลลัพธ์ของการทำนาย
53 # labelCol คือค่าจริง (target) และ predictionCol คือค่าที่โมเดลทำนาย
54 evaluator = RegressionEvaluator(labelCol="num_loves_ind", predictionCol="prediction")
55
56 # ประเมินผลลัพธ์โดยการคำนวณค่า R2 ซึ่งเป็นตัวชี้วัดคุณภาพของโมเดล (ค่า R2 ใกล้ 1 แปลว่าโมเดลทำนายได้ดี)
57 r2 = evaluator.setMetricName("r2").evaluate(predictions)
58 print(f"R2 score: {r2}") # แสดงผลค่า R2
59
60 # หยุดการทำงานของ SparkSession หลังจากใช้งานเสร็จสิ้น
61 spark.stop()
62
```

R2 score: 0.42841839185676467

Classification

Logistic Regression

โค้ดนี้ใช้ PySpark ในการสร้างโมเดล Logistic Regression เพื่อจำแนกประเภทข้อมูลที่อยู่ในไฟล์ CSV (fb_live_thailand.csv) โดยมีขั้นตอนการประมวลผลข้อมูลและการฝึกโมเดลอย่างเป็นระบบตั้งแต่การแปลงข้อมูล การรวมฟีเจอร์ การฝึกโมเดล ไปจนถึงการประเมินผลลัพธ์ของโมเดลหลังจากทดสอบกับข้อมูล

Classification > Logistic.py > ...

```
1 # นำเข้าไลบรารีที่จำเป็นจาก PySpark สำหรับการทำงานกับ DataFrame, Logistic Regression และการประเมินผล
2 from pyspark.sql import SparkSession
3 from pyspark.ml.feature import StringIndexer, VectorAssembler
4 from pyspark.ml.classification import LogisticRegression
5 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
6 from pyspark.ml import Pipeline
7
8 # สร้าง SparkSession ซึ่งเป็นจุดเริ่มต้นของการทำงานกับ PySpark
9 spark = SparkSession.builder.appName("LogisticRegressionExample").getOrCreate()
10
11 # โหลดข้อมูลจากไฟล์ CSV ลงใน DataFrame
12 # data เป็นตัวแปรที่เก็บข้อมูลที่ถูกโหลดจากไฟล์ fb_live_thailand.csv
13 # header=True หมายถึงไฟล์ CSV มีบรรทัดแรกเป็นหัวคอลัมน์
14 # inferSchema=True หมายถึงให้ Spark เค้าว่าข้อมูลแต่ละคอลัมน์ควรเป็นชนิดข้อมูลแบบไหน
15 data = spark.read.csv("fb_live_thailand.csv", header=True, inferSchema=True)
16
17 # ตรวจสอบ schema ของข้อมูลว่าคอลัมน์ถูกกำหนดชนิดข้อมูลถูกต้องหรือไม่
18 # printSchema() จะแสดงประเภทของข้อมูลแต่ละคอลัมน์ใน DataFrame
19 data.printSchema()
20
21 # แปลงคอลัมน์ที่เป็นข้อมูลประเภทตัวอักษร (categorical) ให้เป็นตัวเลข (index)
22 # StringIndexer จะแปลงค่าที่เป็นตัวอักษรเป็นตัวเลข เช่น "video" อาจถูกแปลงเป็น 0, "photo" อาจถูกแปลงเป็น 1
23 # status_type_indexer เป็นตัวแปรที่เก็บกระบวนการแปลงคอลัมน์ "status_type" ให้เป็นตัวเลข และเก็บผลลัพธ์ในคอลัมน์ใหม่ชื่อ "status_type_ind"
24 status_type_indexer = StringIndexer(inputCol="status_type", outputCol="status_type_ind", handleInvalid="keep")
25
26 # คอลัมน์ "status_published" ที่เป็นเวลาถูกแปลงเป็นตัวเลขในคอลัมน์ "status_published_ind"
27 # status_published_indexer เป็นตัวแปรที่เก็บกระบวนการแปลงคอลัมน์ "status_published"
28 status_published_indexer = StringIndexer(inputCol="status_published", outputCol="status_published_ind", handleInvalid="keep")
29
```


Classification > Logistic.py > ...

```
30 # รวมคอลัมน์ที่ต้องการใช้เป็นฟีเจอร์ (input) ในการฝึกโมเดล
31 # assembler เป็นตัวแปรที่เก็บกระบวนการรวมคอลัมน์ที่ผ่านการแปลง เช่น 'status_type_ind' และ 'status_published_ind' เข้าเป็นฟีเจอร์เดียวในคอลัมน์ใหม่ชื่อว่า "features"
32 assembler = VectorAssembler(
33     inputCols=["status_type_ind", "status_published_ind"], # คอลัมน์ที่ต้องการรวมเป็นฟีเจอร์
34     outputCol="features" # คอลัมน์ใหม่ที่จะเก็บฟีเจอร์ที่รวมกันแล้ว
35 )
36
37 # สร้างโมเดล Logistic Regression
38 # lr เป็นตัวแปรที่เก็บโมเดล Logistic Regression ซึ่งเป็นโมเดลสำหรับการจำแนกประเภท (classification)
39 # Logistic Regression เป็นโมเดลที่ใช้ในการทำนายข้อมูลแบบจำแนกประเภท เช่น ทำนายว่าข้อมูลเป็นหมวดหมู่ไหน
40 lr = LogisticRegression(
41     featuresCol="features", # คอลัมน์ที่เป็นฟีเจอร์ (input) สำหรับโมเดล
42     labelCol="status_type_ind", # คอลัมน์ที่เป็นเป้าหมาย (label) ที่ต้องการทำนาย
43     maxIter=10, # จำนวนรอบสูงสุดในการทำซ้ำเพื่อปรับปรุงโมเดล
44     regParam=0.01, # ค่าพารามิเตอร์สำหรับ regularization เพื่อป้องกัน overfitting
45     elasticNetParam=0.8 # ค่าผสมระหว่าง L1 และ L2 regularization (ElasticNet)
46 )
47
48 # สร้าง Pipeline ซึ่งรวมขั้นตอนการทำงานหลายขั้นตอนเข้าด้วยกัน
49 # pipeline เป็นตัวแปรที่เก็บ Pipeline ซึ่งรวมการทำ indexing, การรวมฟีเจอร์ และการสร้างโมเดล Logistic Regression เข้าเป็นขั้นตอนเดียว
50 pipeline = Pipeline(stages=[status_type_indexer, status_published_indexer, assembler, lr])
51
52 # แบ่งข้อมูลออกเป็นสองชุดคือชุดฝึก (train_data) และชุดทดสอบ (test_data)
53 # train_data และ test_data เก็บข้อมูลที่แบ่งออกเป็น 70% สำหรับการฝึกโมเดล และ 30% สำหรับการทดสอบ
54 train_data, test_data = data.randomSplit([0.7, 0.3], seed=42) # seed=42 ใช้เพื่อให้ผลลัพธ์สามารถทำซ้ำได้
```

Classification > Logistic.py > ...

```
55
56 # ฝึกโมเดล Logistic Regression โดยใช้ข้อมูลชุดฝึก
57 # model เป็นตัวแปรที่เก็บโมเดล Logistic Regression ที่ผ่านการฝึกจากข้อมูล train_data แล้ว
58 model = pipeline.fit(train_data)
59
60 # ใช้โมเดลที่ฝึกเสร็จแล้วในการทำนายผลลัพธ์จากข้อมูลชุดทดสอบ
61 # predictions เป็นตัวแปรที่เก็บผลลัพธ์ที่ได้จากการทำนายโดยใช้ข้อมูล test_data
62 predictions = model.transform(test_data)
63
64 # แสดงผลลัพธ์การทำนาย 5 แถวแรก โดยแสดงคอลัมน์ "status_type_ind" (ค่าจริง), "prediction" (ค่าทำนาย), และ "probability" (ความน่าจะเป็น)
65 predictions.select("status_type_ind", "prediction", "probability").show(5)
66
67 # สร้างตัวประเมินผลของโมเดล (evaluator) สำหรับการวัดประสิทธิภาพแบบหลายคลาส
68 # evaluator เป็นตัวแปรที่เก็บ MulticlassClassificationEvaluator สำหรับใช้ประเมินความถูกต้องของโมเดล
69 evaluator = MulticlassClassificationEvaluator(
70     labelCol="status_type_ind", # คอลัมน์ที่เป็น label (ค่าจริง)
71     predictionCol="prediction" # คอลัมน์ที่เป็นค่าทำนาย
72 )
73
74 # คำนวณค่า accuracy (ความถูกต้อง), precision (ความแม่นยำ), recall (ความครอบคลุม) และ F1 score ของโมเดล
75 # accuracy เก็บค่าความถูกต้องของโมเดล โดยเทียบผลลัพธ์ที่ทำนายกับค่าจริง
76 accuracy = evaluator.evaluate(predictions, {evaluator.metricName: "accuracy"})
77 # precision เก็บค่าความแม่นยำของโมเดล โดยดูจากจำนวนที่ทำนายถูกจากจำนวนที่ทำนายทั้งหมด
78 precision = evaluator.evaluate(predictions, {evaluator.metricName: "weightedPrecision"})
79 # recall เก็บค่าความครอบคลุม โดยดูจากจำนวนที่ทำนายถูกจากจำนวนที่เป็นค่าจริง
80 recall = evaluator.evaluate(predictions, {evaluator.metricName: "weightedRecall"})
81 # F1 score เป็นค่าที่ผสมระหว่าง precision และ recall เพื่อวัดประสิทธิภาพของโมเดล
82 f1 = evaluator.evaluate(predictions, {evaluator.metricName: "f1"})
83
```

Classification > Logistic.py > ...

```
84 # แสดงค่าที่คำนวณได้จากตัวประเมินผลลัพธ์ (accuracy, precision, recall, และ F1 score)
85 print(f"Accuracy: {accuracy}")
86 print(f"Precision: {precision}")
87 print(f"Recall: {recall}")
88 print(f"F1 Measure: {f1}")
89
90 # หยุดการทำงานของ SparkSession หลังจากเสร็จสิ้นการประมวลผล
91 spark.stop()
92
```

```
PS C:\Users\rawip\Downloads\BigData\Classification> python .\I
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For Spar
root
```

```
-- status_id: string (nullable = true)
-- status_type: string (nullable = true)
-- status_published: string (nullable = true)
-- num_reactions: integer (nullable = true)
-- num_comments: integer (nullable = true)
-- num_shares: integer (nullable = true)
-- num_likes: integer (nullable = true)
-- num_loves: integer (nullable = true)
-- num_wows: integer (nullable = true)
-- num_hahas: integer (nullable = true)
-- num_sads: integer (nullable = true)
-- num_angrys: integer (nullable = true)
```

24/10/29 00:14:21 WARN InstanceBuilder: Failed to load impleme

24/10/29 00:14:21 WARN InstanceBuilder: Failed to load impleme

```
+-----+-----+-----+
|status_type_ind|prediction|probability|
+-----+-----+-----+
|              |0.0|      0.0|[0.98380477009818...|
|              |0.0|      0.0|[0.98380477009818...|
|              |0.0|      0.0|[0.98380477009818...|
|              |0.0|      0.0|[0.98380477009818...|
|              |0.0|      0.0|[0.98380477009818...|
+-----+-----+-----+
```

only showing top 5 rows

Accuracy: 0.9900398406374502

Precision: 0.9817397078353254

Recall: 0.9900398406374502

F1 Measure: 0.9855124954726548

PS C:\Users\rawip\Downloads\BigData\Classification> SUCCESS: -


SUCCESS: The process with PID 26320 (child process of PID 3296

SUCCESS: The process with PID 32968 (child process of PID 1859

□

Decision Tree Classification

โค้ดนี้เป็นการสร้างและประเมินผลโมเดล Decision Tree Classifier ซึ่งเป็นโมเดลที่ใช้ในการจำแนกประเภทข้อมูล (classification) โดยโค้ดนี้ถูกเขียนขึ้นในภาษา PySpark และทำงานกับข้อมูลจากไฟล์ CSV (fb_live_thailand.csv) โดยมีการประมวลผลข้อมูลและฝึกโมเดลผ่านการสร้าง Pipeline รวมถึงการประเมินความแม่นยำของโมเดลในการทำนาย

Classification >  DecisionTree_Class.py > ...

```
1 # นำเข้าไลบรารีที่จำเป็นจาก PySpark สำหรับการทำงานกับ DataFrame, การจัดประเภท (classification), และการประเมินผล (evaluation)
2 from pyspark.sql import SparkSession
3 from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
4 from pyspark.ml.classification import DecisionTreeClassifier
5 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
6 from pyspark.ml import Pipeline
7
8 # สร้าง SparkSession ซึ่งเป็นจุดเริ่มต้นของการทำงานกับ PySpark
9 spark = SparkSession.builder.appName("DecisionTreeExample").getOrCreate()
10
11 # โหลดข้อมูลจากไฟล์ CSV ลงใน DataFrame
12 # ตัวแปร `data` เก็บข้อมูลที่ถูกโหลดจากไฟล์ CSV (fb_live_thailand.csv)
13 # header=True หมายถึงไฟล์ CSV มีบรรทัดแรกเป็นหัวข้อคอลัมน์
14 # inferSchema=True หมายถึงให้ Spark เดาชนิดของข้อมูลในแต่ละคอลัมน์โดยอัตโนมัติ
15 data = spark.read.csv("fb_live_thailand.csv", header=True, inferSchema=True)
16
17 # ตรวจสอบ schema ของข้อมูลว่าแต่ละคอลัมน์ถูกกำหนดชนิดข้อมูลถูกต้องหรือไม่
18 data.printSchema()
19
20 # แปลงคอลัมน์ที่เป็นข้อมูลประเภทตัวอักษร (categorical) ให้เป็นตัวเลข (index) ด้วย StringIndexer
21 # status_type_indexer: ตัวแปรที่เก็บกระบวนการแปลงคอลัมน์ `status_type` ให้เป็นตัวเลข (index) ในคอลัมน์ใหม่ชื่อ `status_type_ind`
22 status_type_indexer = StringIndexer(inputCol="status_type", outputCol="status_type_ind", handleInvalid="keep")
23 |
```

Classification >  DecisionTree_Class.py > ...

```
24 # status_published_indexer: ตัวแปรที่เก็บกระบวนการแปลงคอลัมน์ `status_published` ให้เป็นตัวเลขในคอลัมน์ใหม่ชื่อ `status_published_ind`
25 status_published_indexer = StringIndexer(inputCol="status_published", outputCol="status_published_ind", handleInvalid="keep")
26
27 # แปลงคอลัมน์ที่เป็นตัวเลข (index) ให้เป็น one-hot encoded vectors ด้วย OneHotEncoder
28 # status_type_encoder: แปลงคอลัมน์ `status_type_ind` ให้เป็น one-hot encoded vector ในคอลัมน์ใหม่ `status_type_vec`
29 status_type_encoder = OneHotEncoder(inputCols=["status_type_ind"], outputCols=["status_type_vec"])
30
31 # status_published_encoder: แปลงคอลัมน์ `status_published_ind` ให้เป็น one-hot encoded vector ในคอลัมน์ใหม่ `status_published_vec`
32 status_published_encoder = OneHotEncoder(inputCols=["status_published_ind"], outputCols=["status_published_vec"])
33
34 # รวมฟีเจอร์ (คอลัมน์ที่ใช้ในการทำนาย) เข้าด้วยกันเป็นฟีเจอร์เดียวในคอลัมน์ `features` ด้วย VectorAssembler
35 # assembler: ตัวแปรที่เก็บขั้นตอนการรวมคอลัมน์ฟีเจอร์ที่ผ่านการแปลงเป็นเวกเตอร์ (เช่น `status_type_vec` และ `status_published_vec`)
36 assembler = VectorAssembler(
37     inputCols=["status_type_vec", "status_published_vec"], # ใช้องค์ประกอบที่ถูกแปลงเป็นเวกเตอร์
38     outputCol="features" # ฟีเจอร์ที่รวมกันแล้วเก็บในคอลัมน์ชื่อ `features`
39 )
```

Classification > DecisionTree_Class.py > ...

```
40
41 # สร้างโมเดล Decision Tree สำหรับการจัดประเภท
42 # dt: ตัวแปรที่เก็บโมเดล Decision Tree ซึ่งใช้ในการจำแนกประเภท (classification)
43 # featuresCol คือคอลัมน์ที่เป็นฟีเจอร์ (ข้อมูลที่ใช้ในการทำนาย)
44 # labelCol คือคอลัมน์ที่เป็นเป้าหมาย (ค่าที่ต้องการทำนาย)
45 dt = DecisionTreeClassifier(
46     featuresCol="features",
47     labelCol="status_type_ind" # คอลัมน์ที่เป็นเป้าหมาย (label)
48 )
49
50 # สร้าง Pipeline เพื่อรวมขั้นตอนการทำ indexing, encoding, การรวมฟีเจอร์ และการสร้างโมเดลเข้าด้วยกัน
51 # pipeline: ตัวแปรที่เก็บ Pipeline ซึ่งรวมขั้นตอนการแปลงข้อมูลและการสร้างโมเดล
52 pipeline = Pipeline(stages=[status_type_indexer, status_published_indexer, status_type_encoder, status_published_encoder, assembler, dt])
53
54 # แบ่งข้อมูลออกเป็นชุดฝึก (train_data) และชุดทดสอบ (test_data)
55 # train_data และ test_data เก็บข้อมูลที่แบ่งออกเป็น 70% สำหรับฝึกโมเดล และ 30% สำหรับทดสอบโมเดล
56 train_data, test_data = data.randomSplit([0.7, 0.3], seed=42) # seed=42 ใช้เพื่อให้ผลลัพธ์สามารถทำซ้ำได้
57
58 # ฝึกโมเดล Decision Tree โดยใช้ข้อมูลชุดฝึก (train_data)
59 # model: ตัวแปรที่เก็บโมเดล Decision Tree ที่ผ่านการฝึกแล้ว
60 model = pipeline.fit(train_data)
```

Classification > DecisionTree_Class.py > ...

```
61
62 # ใช้โมเดลที่ฝึกแล้วทำนายผลลัพธ์จากข้อมูลชุดทดสอบ (test_data)
63 # predictions: ตัวแปรที่เก็บผลลัพธ์การทำนายจากข้อมูล test_data
64 predictions = model.transform(test_data)
65
66 # แสดงผลลัพธ์การทำนาย 5 แถวแรก โดยแสดงคอลัมน์ `status_type_ind` (ค่าจริง), `prediction` (ค่าทำนาย), และ `probability` (ความน่าจะเป็น)
67 predictions.select("status_type_ind", "prediction", "probability").show(5)
68
69 # สร้างตัวประเมินผลลัพธ์ของโมเดล (evaluator) สำหรับการจำแนกประเภทแบบหลายคลาส (multiclass classification)
70 # evaluator: ตัวแปรที่เก็บ MulticlassClassificationEvaluator สำหรับใช้ประเมินความถูกต้องของโมเดล
71 evaluator = MulticlassClassificationEvaluator(
72     labelCol="status_type_ind", # คอลัมน์ที่เป็น label (ค่าจริง)
73     predictionCol="prediction" # คอลัมน์ที่เป็นค่าทำนาย
74 )
75
76 # คำนวณค่า accuracy (ความถูกต้อง), precision (ความแม่นยำ), recall (ความครอบคลุม) และ F1 score ของโมเดล
77 # accuracy เก็บค่าความถูกต้องของโมเดล โดยเทียบผลลัพธ์ที่ทำนายกับค่าจริง
78 accuracy = evaluator.evaluate(predictions, {evaluator.metricName: "accuracy"})
79 # precision เก็บค่าความแม่นยำของโมเดล โดยดูจากจำนวนที่ทำนายถูกจากจำนวนที่ทำนายทั้งหมด
80 precision = evaluator.evaluate(predictions, {evaluator.metricName: "weightedPrecision"})
81 # recall เก็บค่าความครอบคลุม โดยดูจากจำนวนที่ทำนายถูกจากจำนวนที่เป็นค่าจริง
82 recall = evaluator.evaluate(predictions, {evaluator.metricName: "weightedRecall"})
83 # F1 score เป็นค่าที่ผสมระหว่าง precision และ recall เพื่อวัดประสิทธิภาพของโมเดล
84 f1 = evaluator.evaluate(predictions, {evaluator.metricName: "f1"})
85
```

```

86 # แสดงค่าที่คำนวณได้จากตัวประเมินผลลัพธ์ (accuracy, precision, recall, และ F1 score)
87 print(f"Accuracy: {accuracy}")
88 print(f"Precision: {precision}")
89 print(f"Recall: {recall}")
90 print(f"F1 Measure: {f1}")
91
92 # คำนวณและแสดงค่า Test Error (ค่า Error จากข้อมูลทดสอบ)
93 # test_error: ตัวแปรที่เก็บค่าความผิดพลาดในการทำนาย โดยคำนวณจาก 1 ลบด้วยค่า accuracy
94 test_error = 1.0 - accuracy
95 print(f"Test Error: {test_error}")
96
97 # หยุดการทำงานของ SparkSession หลังจากเสร็จสิ้นการประมวลผล
98 spark.stop()
99

```

```

root
|-- status_id: string (nullable = true)
|-- status_type: string (nullable = true)
|-- status_published: string (nullable = true)
|-- num_reactions: integer (nullable = true)
|-- num_comments: integer (nullable = true)
|-- num_shares: integer (nullable = true)
|-- num_likes: integer (nullable = true)
|-- num_loves: integer (nullable = true)
|-- num_wows: integer (nullable = true)
|-- num_hahas: integer (nullable = true)
|-- num_sads: integer (nullable = true)
|-- num_angrys: integer (nullable = true)

+-----+-----+-----+
|status_type_ind|prediction|probability|
+-----+-----+-----+
|          0.0|        0.0|[1.0,0.0,0.0,0.0,...|
|          0.0|        0.0|[1.0,0.0,0.0,0.0,...|
|          0.0|        0.0|[1.0,0.0,0.0,0.0,...|
|          0.0|        0.0|[1.0,0.0,0.0,0.0,...|
|          0.0|        0.0|[1.0,0.0,0.0,0.0,...|
+-----+-----+-----+
only showing top 5 rows

```

```

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

```

Recommendation.py

โค้ดนี้เป็นการสร้าง ระบบแนะนำหนังสือ (Book Recommendation System) โดยใช้ อัลกอริธึม ALS (Alternating Least Squares) ซึ่งเป็นอัลกอริธึมที่ใช้ใน ระบบแนะนำ (Recommendation System) เพื่อช่วยให้สามารถแนะนำสินค้าหรือรายการต่างๆ ตามพฤติกรรมการใช้งานหรือการให้คะแนนของผู้ใช้

```
RecommendationSystem > Recommendation.py > ...
1  # Import necessary libraries
2  from pyspark.sql import SparkSession # SparkSession เป็นเริ่มต้นของการทำงานใน PySpark
3  from pyspark.ml.recommendation import ALS # ALS (Alternating Least Squares) ใช้สร้างระบบแนะนำ
4  from pyspark.ml.evaluation import RegressionEvaluator # ใช้สำหรับประเมินโมเดลโดยวัดความคลาดเคลื่อน (RMSE)
5  from pyspark.sql.functions import col # ใช้สำหรับการกรองและอ้างอิงคอลัมน์ใน DataFrame
6
7  # สร้าง SparkSession ซึ่งเป็นจุดเริ่มต้นของการทำงานของ Spark application
8  spark = SparkSession.builder \
9      .appName("BookRecommendationALS") \
10     .getOrCreate() # สร้าง SparkSession หรือใช้ที่มีอยู่แล้วถ้ามีการสร้างมาก่อนหน้านี้
11
12 # โหลดข้อมูลจากไฟล์ CSV ที่ประกอบด้วยข้อมูลการให้คะแนนหนังสือจากผู้ใช้
13 # header=True: หมายถึง ไฟล์มีหัวตารางคอลัมน์ (header)
14 # inferSchema=True: ให้ PySpark ระบุประเภทของข้อมูลอัตโนมัติ (เช่น ตัวเลข, ข้อความ)
15 data = spark.read.csv(r'C:\Users\sooke\Desktop\big data\RecommendationSystem\book_ratings.csv', header=True, inferSchema=True)
16
17 # แสดงโครงสร้างของข้อมูล (Schema) เพื่อดูชื่อและชนิดของแต่ละคอลัมน์
18 data.printSchema()
19
20 # แสดงตัวอย่างข้อมูล 5 แถวแรกจาก DataFrame เพื่อดูข้อมูล
21 data.show(5)
22
23 # กำหนดโมเดล ALS ซึ่งเป็นโมเดลสำหรับการสร้างระบบแนะนำ (Recommendation System)
24 als = ALS(
25     maxIter=10, # จำนวนรอบในการฝึกโมเดล (iterations) เพื่อทำการเรียนรู้ซ้ำ 10 รอบ
26     userCol="user_id", # คอลัมน์ที่ระบุรหัสผู้ใช้ (user ID)
27     itemCol="book_id", # คอลัมน์ที่ระบุรหัสหนังสือ (book ID)
28     ratingCol="rating", # คอลัมน์ที่ใช้เป็นค่าการให้คะแนน (ratings)
29     coldStartStrategy="drop" # กำหนดให้ลบแถวที่ไม่สามารถทำนายได้ (เช่น มี NaN ในค่าพยากรณ์)
30 )
```


RecommendationSystem > Recommendation.py > ...

```
30 )
31
32 # ฝึกโมเดล ALS ด้วยข้อมูลการให้คะแนนจาก DataFrame 'data'
33 # โมเดลจะทำการเรียนรู้ความสัมพันธ์ระหว่างผู้ใช้และหนังสือ
34 model = als.fit(data)
35
36 # ใช้โมเดลที่ถูกฝึกแล้วเพื่อทำนายการให้คะแนนของผู้ใช้
37 # DataFrame 'predictions' จะมีคอลัมน์ 'prediction' ที่เก็บค่าที่ทำนายจากโมเดล
38 predictions = model.transform(data)
39
40 # สร้าง RegressionEvaluator เพื่อประเมินโมเดลด้วยการคำนวณค่า RMSE (Root Mean Squared Error)
41 evaluator = RegressionEvaluator(
42     metricName="rmse", # ใช้ RMSE เป็นตัววัดความผิดพลาดระหว่างค่าที่ทำนายและค่าจริง
43     labelCol="rating", # คอลัมน์ที่เก็บค่าการให้คะแนนจริง
44     predictionCol="prediction" # คอลัมน์ที่เก็บค่าที่ทำนายโดยโมเดล
45 )
46
47 # ประเมินโมเดลโดยคำนวณค่า RMSE เพื่อดูความแม่นยำของโมเดล
48 rmse = evaluator.evaluate(predictions)
49 # พิมพ์ผลลัพธ์ของค่า RMSE ยิ่งค่าต่ำยิ่งดี (หมายถึงโมเดลมีความแม่นยำมาก)
50 print(f"Root Mean Squared Error (RMSE): {rmse}")
```

```
51
52 # กรองข้อมูลการทำนายเพื่อเฉพาะผู้ใช้ที่มี user_id เท่ากับ 53
53 # การแสดงผลจะรวมถึง book_id, user_id, rating (ค่าจริง) และ prediction (ค่าทำนาย)
54 user_id = 53 # กำหนด user_id ของผู้ใช้ที่เราต้องการดู
55 user_predictions = predictions.filter(col("user_id") == user_id) # กรองข้อมูลเฉพาะผู้ใช้ที่มี user_id = 53
56 user_predictions = user_predictions.select("book_id", "user_id", "rating", "prediction").orderBy(col("prediction").desc()) # เรียงลำดับตามค่าทำนายจากมากไปน้อย
57 user_predictions.show(truncate=False) # แสดงผลลัพธ์โดยไม่ตัดข้อความในคอลัมน์
58
59 # แสดงหนังสือ 5 เล่มที่แนะนำสำหรับผู้ใช้แต่ละคน
60 # ฟังก์ชัน recommendForAllUsers จะสร้างคำแนะนำหนังสือ 5 เล่มสำหรับผู้ใช้ทุกคน
61 user_recommendations = model.recommendForAllUsers(5)
62 user_recommendations.show(truncate=False) # แสดงผลลัพธ์โดยไม่ตัดข้อความในคอลัมน์
63
64 # แสดงผู้ใช้ 5 คนที่โมเดลแนะนำสำหรับหนังสือแต่ละเล่ม
65 # ฟังก์ชัน recommendForAllItems จะสร้างคำแนะนำว่าผู้ใช้ 5 คนควรสนใจหนังสือใด
66 item_recommendations = model.recommendForAllItems(5)
67 item_recommendations.show(truncate=False) # แสดงผลลัพธ์โดยไม่ตัดข้อความในคอลัมน์
68
69 # หยุดการทำงานของ SparkSession เพื่อคืนทรัพยากรระบบ
70 spark.stop()
```

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

```
+-----+-----+-----+
|book_id|user_id|rating|
+-----+-----+-----+
|      1|      314|      5|
|      1|      439|      3|
|      1|      588|      5|
|      1|     1169|      4|
|      1|     1185|      4|
+-----+-----+-----+
```

only showing top 5 rows

```
24/10/29 02:36:11 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
24/10/29 02:36:11 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.VectorBLAS
24/10/29 02:36:12 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.lapack.JNILAPACK
Root Mean Squared Error (RMSE): 0.5961006520223846
```

```
+-----+-----+-----+-----+
|book_id|user_id|rating|prediction|
+-----+-----+-----+-----+
|8946   |53      |5      |4.356636 |
|8882   |53      |2      |2.070715 |
|8336   |53      |1      |1.2028226|
|8336   |53      |1      |1.2028226|
+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+
|user_id|recommendations
+-----+-----+-----+-----+
|26      |[{7593, 4.217442}, {7844, 4.19601}, {7264, 4.144877}, {6634, 4.1429944}, {6591, 4.108191}]
|27      |[{9842, 4.923128}, {5146, 4.838136}, {4868, 4.7964773}, {7910, 4.7963667}, {4653, 4.790279}]
|28      |[{3628, 4.392176}, {8286, 4.3842134}, {8165, 4.2770786}, {1146, 4.267518}, {6435, 4.2505918}]
|31      |[{6438, 4.1845574}, {9537, 4.151729}, {7264, 4.145035}, {6018, 4.125628}, {2865, 4.0718246}]
|34      |[{6590, 3.3948548}, {5207, 3.3025634}, {6920, 3.299908}, {9566, 3.2878265}, {4483, 3.2687678}]
|44      |[{8648, 4.778167}, {834, 4.730455}, {6055, 4.7300625}, {4107, 4.6916943}, {5919, 4.6847}]
|53      |[{8991, 4.587786}, {8325, 4.523859}, {4185, 4.5136924}, {5001, 4.406523}, {3479, 4.364233}]
|65      |[{5071, 5.2085137}, {8337, 5.1689577}, {4638, 5.164879}, {7988, 5.158798}, {2655, 5.130577}]
|76      |[{2636, 5.256977}, {1496, 5.1642494}, {2058, 5.1342864}, {4868, 5.124871}, {5880, 5.0764184}]
|78      |[{5730, 2.8057554}, {6457, 2.7901661}, {2681, 2.7350025}, {7947, 2.7086427}, {4054, 2.6995502}]
|81      |[{6784, 3.260257}, {9912, 3.2434149}, {8099, 3.1809409}, {4868, 3.1773825}, {9076, 3.1688106}]
|85      |[{9842, 4.8533235}, {9566, 4.7947035}, {6590, 4.783255}, {7039, 4.773732}, {5207, 4.748844}]
|101     |[{3491, 5.1464863}, {4653, 5.1414356}, {1010, 5.047927}, {6751, 5.0435963}, {9374, 5.027015}]
|103     |[{5207, 4.4752145}, {3628, 4.332955}, {6590, 4.3210106}, {9566, 4.3209004}, {6920, 4.316073}]
|108     |[{1146, 3.7922008}, {2958, 3.75969}, {9946, 3.7580483}, {5241, 3.6944582}, {5344, 3.689577}]
|115     |[{3628, 3.0335104}, {862, 2.9746413}, {5207, 2.9700074}, {6070, 2.9469502}, {4107, 2.939925}]
|126     |[{9714, 4.571049}, {9946, 4.520449}, {5841, 4.417008}, {8249, 4.354843}, {8996, 4.345814}]
|133     |[{6600, 5.3707085}, {8362, 5.3092537}, {8479, 5.253154}, {7283, 5.219159}, {4609, 5.1993036}]
|137     |[{7947, 4.8238335}, {4468, 4.8065934}, {2100, 4.721501}, {5493, 4.7202067}, {8707, 4.656736}]
|148     |[{7593, 4.4957767}, {7401, 4.4637694}, {6590, 4.454135}, {8492, 4.446464}, {9842, 4.444821}]
+-----+-----+-----+-----+
```

only showing top 20 rows

TimeSeries.py

โค้ดนี้มีไว้สำหรับ วิเคราะห์ข้อมูล Time Series และสร้าง โมเดล ARIMA เพื่อทำนายค่าในอนาคต โดยใช้ข้อมูล ยอดขายรายปี (yearly sales) ซึ่งอยู่ในไฟล์ year_sales.csv และแบ่งข้อมูลออกเป็นชุดฝึก (Training set) และ ชุดทดสอบ (Test set) จากนั้นจะสร้างโมเดล ARIMA เพื่อทำนายยอดขายในอนาคต

TimeSeries > TimeSeries.py > ...

```
1 import pandas as pd # นำเข้า Pandas สำหรับจัดการกับข้อมูลตาราง (DataFrame)
2 from pmdarima.arima import auto_arima, ADFTest # นำเข้า auto_arima สำหรับการสร้างโมเดล ARIMA และ ADFTest สำหรับการทดสอบความนิ่งของข้อมูล (stationarity)
3 import matplotlib.pyplot as plt # นำเข้า Matplotlib สำหรับการสร้างกราฟ
4
5 # 1. นำเข้าข้อมูลจากไฟล์ year_sales.csv
6 df = pd.read_csv(r'C:\Users\sooke\Desktop\big data\TimeSeries\year_sales.csv')
7 # 'df' คือ DataFrame ที่เก็บข้อมูลยอดขายรายปีจากไฟล์ CSV
8
9 # 2. ใช้ฟังก์ชัน to_datetime() เพื่อแปลงคอลัมน์ Year เป็น datetime
10 df['Year'] = pd.to_datetime(df['Year'])
11 # แปลงคอลัมน์ 'Year' ให้เป็นชนิด datetime เพื่อให้สามารถใช้เป็นดัชนีเวลาได้
12
13 # 3. ตั้งค่า Year เป็นดัชนีโดยใช้ฟังก์ชัน set_index()
14 df.set_index('Year', inplace=True)
15 # ใช้คอลัมน์ 'Year' เป็นดัชนีของ DataFrame เพื่อทำการวิเคราะห์ Time Series
16
17 # 4. พล็อตข้อมูลดั้งเดิม
18 df.plot()
19 plt.title('Yearly Sales') # ตั้งชื่อกราฟ
20 plt.xlabel('Year') # ตั้งชื่อแกน X ว่า Year
21 plt.ylabel('Sales') # ตั้งชื่อแกน Y ว่า Sales
22 plt.show() # แสดงกราฟ
23 # พล็อตกราฟแสดงยอดขายรายปีเพื่อดูแนวโน้มของข้อมูล
```

```

24
25 # 5. ตรวจสอบว่าจำเป็นต้องทำ Differencing หรือไม่ด้วย ADFTest
26 adf_test = ADFTest(alpha=0.05)
27 should_diff = adf_test.should_diff(df)
28 # 'adf_test' คือการทดสอบ ADF (Augmented Dickey-Fuller) เพื่อตรวจสอบว่าข้อมูล Time Series มีแนวโน้ม หรือมีความน่าจะเป็นที่จะต้องทำการ Differencing หรือไม่
29 # 'alpha=0.05' คือค่าระดับนัยสำคัญ (significance level) ที่ใช้ในการทดสอบ
30
31 # 6. แบ่งข้อมูลเพื่อใช้ในการฝึกฝน (80%) และทดสอบ (20%)
32 train_size = int(len(df) * 0.8) # คำนวณขนาดของข้อมูลฝึก (80% ของข้อมูลทั้งหมด)
33 train = df.iloc[:train_size] # แยกข้อมูล 80% แรกเพื่อใช้เป็นชุดฝึก (train set)
34 test = df.iloc[train_size:] # แยกข้อมูล 20% ที่เหลือเพื่อใช้เป็นชุดทดสอบ (test set)
35
36 # 7. สร้างโมเดล ARIMA โดยใช้ auto_arima
37 arima_model = auto_arima(train, # ข้อมูลชุดฝึก
38                           start_p=0, d=1, start_q=0, # กำหนดค่าเริ่มต้นสำหรับพารามิเตอร์ ARIMA (p,d,q)
39                           max_p=5, max_d=5, max_q=5, # กำหนดค่าสูงสุดสำหรับพารามิเตอร์ p,d,q
40                           start_P=0, D=1, start_Q=0, # กำหนดค่าเริ่มต้นสำหรับพารามิเตอร์ ARIMA ฤดูกาล (P,D,Q)
41                           max_P=5, max_D=5, max_Q=5, # กำหนดค่าสูงสุดสำหรับพารามิเตอร์ ARIMA ฤดูกาล (P,D,Q)
42                           m=12, # 'm=12' หมายถึงข้อมูลมีความถี่ตามฤดูกาลทุกๆ 12 เดือน (ปีละ 1 ครั้ง)
43                           seasonal=True, # ใช้ Seasonal ARIMA เนื่องจากข้อมูลนี้เป็นรายปี
44                           error_action='warn', # แสดงการเตือนถ้าเกิดข้อผิดพลาด
45                           trace=True, # แสดงรายละเอียดการทำงานของโมเดลในแต่ละขั้นตอน
46                           suppress_warnings=True, # ปิดการแสดง warnings ที่ไม่จำเป็น
47                           stepwise=True, # ใช้ขั้นตอนแบบ Stepwise เพื่อลดเวลาการคำนวณ
48                           random_state=20, # กำหนดค่าเริ่มต้นสำหรับการสุ่ม (เพื่อความสามารถในการทำซ้ำ)
49                           n_fits=50) # จำนวนการทำซ้ำสูงสุดของการปรับพารามิเตอร์
50 # 'arima_model' คือโมเดล ARIMA ที่สร้างขึ้นโดยใช้ auto_arima สำหรับเลือกค่าพารามิเตอร์ที่เหมาะสมที่สุด
51
52 # 8. แสดงสรุปของโมเดล ARIMA
53 print(arima_model.summary())
54 # แสดงรายละเอียดสรุปของโมเดล ARIMA ที่ได้จากการฝึก เช่น ค่าพารามิเตอร์ที่เลือก, AIC, BIC เป็นต้น
55
56 # 9. สร้างการคาดการณ์จากโมเดล ARIMA
57 n_periods = len(test) # จำนวนข้อมูลในชุดทดสอบ (ใช้เพื่อกำหนดจำนวนคาบเวลาในการทำนาย)
58 prediction = pd.DataFrame(arima_model.predict(n_periods=n_periods), index=test.index)
59 prediction.columns = ['Predicted'] # ตั้งชื่อคอลัมน์การทำนายว่า 'Predicted'
60 # ใช้โมเดล ARIMA ที่สร้างขึ้นมาทำนายยอดขายในช่วงเวลาอนาคต (ตามขนาดของชุดทดสอบ)
61
62 # 10. พล็อตข้อมูล train, test, และผลการคาดการณ์
63 plt.figure(figsize=(10, 6)) # กำหนดขนาดของกราฟ
64 plt.plot(train, label='Train') # พล็อตข้อมูลชุดฝึก
65 plt.plot(test, label='Test') # พล็อตข้อมูลชุดทดสอบ
66 plt.plot(prediction, label='Predicted', color='red') # พล็อตข้อมูลที่ทำนายด้วยโมเดล ARIMA
67 plt.title('Time Series Analysis - Train, Test, and Predicted') # ตั้งชื่อกราฟ
68 plt.xlabel('Year') # ตั้งชื่อแกน X ว่า Year
69 plt.ylabel('Sales') # ตั้งชื่อแกน Y ว่า Sales
70 plt.legend() # แสดงตำนานกราฟ (Legend) เพื่ออธิบายแต่ละเส้น
71 plt.show() # แสดงกราฟ

```

Figure 1

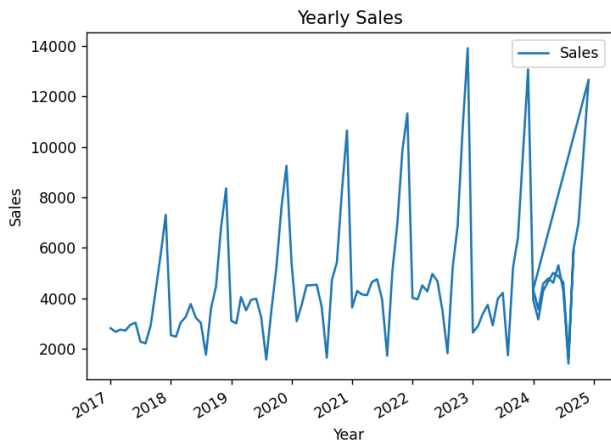
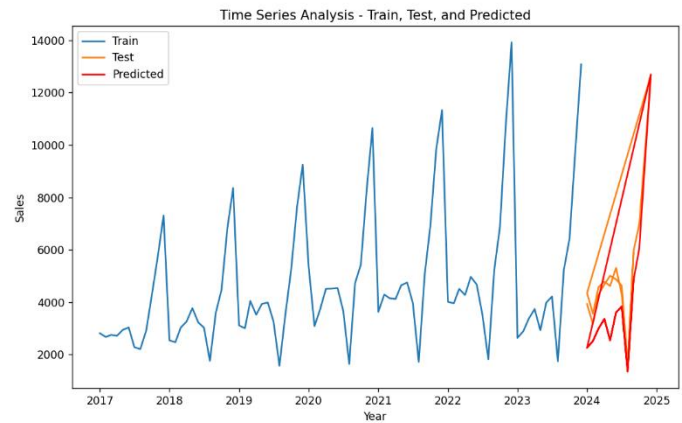


Figure 1



```
PS C:\Users\rawip\Downloads\BigData> cd .\TimeSeries\  
PS C:\Users\rawip\Downloads\BigData\TimeSeries> python .\TimeSeries.py  
Performing stepwise search to minimize aic  
ARIMA(0,1,0)(0,1,0)[12] : AIC=1183.693, Time=0.03 sec  
ARIMA(1,1,0)(1,1,0)[12] : AIC=1173.736, Time=0.08 sec  
ARIMA(0,1,1)(0,1,1)[12] : AIC=1157.042, Time=0.22 sec  
ARIMA(0,1,1)(0,1,0)[12] : AIC=1155.109, Time=0.05 sec  
ARIMA(0,1,1)(1,1,0)[12] : AIC=1157.009, Time=0.18 sec  
ARIMA(0,1,1)(1,1,1)[12] : AIC=1158.348, Time=0.43 sec  
ARIMA(1,1,1)(0,1,0)[12] : AIC=1155.379, Time=0.08 sec  
ARIMA(0,1,2)(0,1,0)[12] : AIC=1155.138, Time=0.08 sec  
ARIMA(1,1,0)(0,1,0)[12] : AIC=1173.612, Time=0.02 sec  
ARIMA(1,1,2)(0,1,0)[12] : AIC=1155.790, Time=0.14 sec  
ARIMA(0,1,1)(0,1,0)[12] intercept : AIC=inf, Time=0.07 sec
```

Best model: ARIMA(0,1,1)(0,1,0)[12]
Total fit time: 1.381 seconds

SARIMAX Results

```
=====
Dep. Variable:          y      No. Observations:      84
Model:                SARIMAX(0, 1, 1)x(0, 1, [], 12)  Log Likelihood      -575.554
Date:                  Tue, 29 Oct 2024               AIC              1155.109
Time:                  02:37:59                       BIC              1159.634
Sample:                01-01-2017                     HQIC             1156.908
                   - 12-01-2023
```

Covariance Type: opg

```
=====
              coef      std err          z      P>|z|      [0.025      0.975]
-----
ma.L1         -0.8756      0.060     -14.663      0.000     -0.993     -0.759
sigma2         5.86e+05    7.15e+04      8.195      0.000     4.46e+05    7.26e+05
=====
```

```
Ljung-Box (L1) (Q):                0.16  Jarque-Bera (JB):                6.95
Prob(Q):                          0.69  Prob(JB):                      0.03
Heteroskedasticity (H):            2.13  Skew:                          0.01
Prob(H) (two-sided):              0.07  Kurtosis:                      4.53
=====
```

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

GraphAnalytics.py

โค้ดนี้ใช้สำหรับการสร้าง กราฟ (Graph) และการวิเคราะห์โครงสร้างกราฟด้วยการใช้ GraphFrames ใน PySpark โดย GraphFrames เป็นไลบรารีที่ช่วยในการประมวลผลกราฟขนาดใหญ่และวิเคราะห์ความสัมพันธ์ระหว่างโหนดต่าง ๆ ได้อย่างมีประสิทธิภาพ

Graph Analytics >  GraphAnalytics.py > ...

```
1  # นำเข้าไลบรารีที่จำเป็น
2  from pyspark.sql import SparkSession # ใช้สำหรับการสร้าง SparkSession
3  from graphframes import GraphFrame  # ใช้สำหรับสร้างกราฟด้วย GraphFrame
4  from pyspark.sql.functions import desc, col # ใช้สำหรับการจัดเรียงข้อมูลและการเลือกคอลัมน์
5
6  # สร้าง SparkSession สำหรับการทำงานกับ Spark และ GraphFrame
7  spark = SparkSession.builder \
8      .appName("Graph Analytics") \
9      .config("spark.jars.packages", "graphframes:graphframes:0.8.2-spark3.0-s_2.12") \
10     .getOrCreate() # สร้าง SparkSession
11
12 # สร้าง DataFrame ของ vertices (โหนดในกราฟ)
13 # vertices: ตัวแปรที่เก็บข้อมูลของโหนด (บุคคล) ซึ่งมีสองคอลัมน์คือ id (ชื่อ) และ age (อายุ)
14 vertices = spark.createDataFrame([
15     ("Alice", 45),
16     ("Jacob", 43),
17     ("Roy", 21),
18     ("Ryan", 49),
19     ("Emily", 24),
20     ("Sheldon", 52)
21 ], ["id", "age"])
22
```

Graph Analytics >  GraphAnalytics.py > ...

```
23 # สร้าง DataFrame ของ edges (ความสัมพันธ์ระหว่างโหนด)
24 # edges: ตัวแปรที่เก็บข้อมูลความสัมพันธ์ระหว่างโหนด มีสามคอลัมน์คือ src (ต้นทาง), dst (ปลายทาง), และ relation (ความสัมพันธ์)
25 edges = spark.createDataFrame([
26     ("Sheldon", "Alice", "Sister"),
27     ("Alice", "Jacob", "Husband"),
28     ("Emily", "Jacob", "Father"),
29     ("Ryan", "Alice", "Friend"),
30     ("Alice", "Emily", "Daughter"),
31     ("Alice", "Roy", "Son"),
32     ("Jacob", "Roy", "Son")
33 ], ["src", "dst", "relation"])
34
35 # สร้างกราฟจาก DataFrame ของ vertices และ edges
36 # graph: ตัวแปรที่เก็บกราฟซึ่งสร้างจาก vertices และ edges
37 graph = GraphFrame(vertices, edges)
38
39 # แสดงข้อมูลของโหนดทั้งหมดในกราฟ
40 print("Vertices:")
41 graph.vertices.show()
42
43 # แสดงข้อมูลของความสัมพันธ์ทั้งหมดในกราฟ
44 print("Edges:")
45 graph.edges.show()
46
47 # ตัวอย่าง: คำสั่ง groupBy และ orderBy เพื่อจัดกลุ่มและเรียงลำดับตามจำนวนความสัมพันธ์ (edges)
48 print("Grouped and Ordered Edges by Count:")
49 graph.edges.groupBy("src", "dst").count().orderBy(desc("count")).show()
50
```

Graph Analytics >  GraphAnalytics.py > ...

```
51 # ตัวอย่าง: กรองข้อมูลของความสัมพันธ์ที่ src (ต้นทาง) หรือ dst (ปลายทาง) เป็น "Alice"
52 print("Filtered Edges where src = 'Alice' or dst = 'Alice':")
53 filtered_edges = graph.edges.where("src = 'Alice' OR dst = 'Alice'")
54 filtered_edges.show()
55
56 # สร้าง subgraph ใหม่จากการกรอง edges ที่ src หรือ dst เป็น "Alice"
57 # subgraph: ตัวแปรที่เก็บกราฟย่อยจากกราฟหลัก โดยมีเพียง edges ที่ผ่านการกรอง
58 print("Subgraph with filtered edges:")
59 subgraph = GraphFrame(graph.vertices, filtered_edges)
60 subgraph.edges.show()
61
62 # ตัวอย่าง: Motif finding คือการค้นหารูปแบบในกราฟ (เช่น รูปแบบของโหนดและความสัมพันธ์)
63 print("Motif finding example:")
64 motifs = graph.find("(a)-[ab]->(b)") # หารูปแบบความสัมพันธ์ระหว่าง a และ b
65 motifs.show()
66
67 # ตัวอย่าง: การคำนวณ PageRank ซึ่งเป็นอัลกอริธึมที่ใช้วัดความสำคัญของโหนดในกราฟ
68 print("PageRank results:")
69 pagerank = graph.pageRank(resetProbability=0.15, maxIter=5) # คำนวณ PageRank โดยกำหนด maxIter เป็น 5
70 pagerank.vertices.orderBy(desc("pagerank")).show() # แสดงผลการจัดอันดับของโหนดตามค่า PageRank
71
72 # ตัวอย่าง: การคำนวณ in-degree และ out-degree ของโหนดในกราฟ
73 print("In-Degree:")
74 graph.inDegrees.orderBy(desc("inDegree")).show() # คำนวณ in-degree ซึ่งคือจำนวนความสัมพันธ์ที่โหนดนั้นเป็นปลายทาง
75
76 print("Out-Degree:")
77 graph.outDegrees.orderBy(desc("outDegree")).show() # คำนวณ out-degree ซึ่งคือจำนวนความสัมพันธ์ที่โหนดนั้นเป็นต้นทาง
78
```

```

79 # ตัวอย่าง: การทำ Breadth-First Search (BFS) เพื่อค้นหาเส้นทางจากโหนดหนึ่งไปยังอีกโหนดหนึ่ง
80 print("Breadth-First Search (BFS):")
81 # ค้นหาเส้นทางจาก Alice ไปหา Roy โดยมีความยาวเส้นทางสูงสุดไม่เกิน 2
82 bfs_result = graph.bfs(fromExpr="id = 'Alice'", toExpr="id = 'Roy'", maxPathLength=2)
83 bfs_result.show()
84
85 # หยุดการทำงานของ SparkSession เมื่อเสร็จสิ้น
86 spark.stop()
87

```

vertices = graph.vertices().
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 and Ordered Edges by Count:

src	dst	count
Sheldon	Alice	1
Alice	Jacob	1
Emily	Jacob	1
Ryan	Alice	1
Alice	Emily	1
Alice	Roy	1
Jacob	Roy	1

Filtered Edges where src = 'Alice' or dst = 'Alice':

src	dst	relation
Sheldon	Alice	Sister
Alice	Jacob	Husband
Ryan	Alice	Friend
Alice	Emily	Daughter
Alice	Roy	Son

Subgraph with filtered edges:

src	dst	relation
Sheldon	Alice	Sister
Alice	Jacob	Husband
Ryan	Alice	Friend
Alice	Emily	Daughter
Alice	Roy	Son

Motif finding example:

C:\Users\rwip\AppData\Local\Programs\Python\Python310\lib\warnings.warn("DataFrame constructor is internal. Do not c

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

id	inDegree
Alice	2
Jacob	2
Roy	2
Emily	1

Out-Degree:

id	outDegree
Alice	3
Sheldon	1
Emily	1
Ryan	1
Jacob	1

Breadth-First Search (BFS):

from	e0	to
{Alice, 45}	{Alice, Roy, Son}	{Roy, 21}

TextAnalytics.py

โค้ดนี้ใช้สำหรับ วิเคราะห์ข้อมูลข้อความ (Text Analytics) จากรีวิวของผู้ใช้ และทำการ จัดประเภทการให้คะแนน (Rating Classification) โดยใช้ Logistic Regression ในการพยากรณ์ว่าข้อความรีวิวใดควรได้คะแนนใดบ้าง (Rating) ซึ่งมีประโยชน์อย่างมากในด้านการ ทำเหมืองข้อมูล (Data Mining) และ การวิเคราะห์ข้อมูลเชิงข้อความ (Text Analysis)

ประโยชน์ของโค้ด:

- วิเคราะห์รีวิวจากลูกค้า: สามารถใช้ในการวิเคราะห์ความรู้สึกหรือการให้คะแนนจากข้อความรีวิวเพื่อดูว่า รีวิวใดบวกหรือลบ
- ระบบการจัดประเภทข้อมูล: ช่วยในการสร้างโมเดลสำหรับจัดประเภทข้อความเช่น รีวิวผลิตภัณฑ์ หรือ คำอธิบายอื่นๆ

- แนะนำผลิตภัณฑ์หรือบริการ: สามารถใช้ผลลัพธ์การวิเคราะห์เพื่อทำความเข้าใจความต้องการของลูกค้าและพัฒนาแคมเปญการตลาดที่ดีขึ้น

TextAnalytics >  TextAnalytics.py > ...

```
1 # นำเข้าไลบรารีที่จำเป็นจาก PySpark สำหรับการสร้าง SparkSession, การจัดการข้อมูล, การแปลงข้อมูล, และการสร้างโมเดล
2 from pyspark.sql import SparkSession # ใช้สำหรับการสร้าง SparkSession
3 from pyspark.sql.types import IntegerType # ใช้สำหรับการแปลงชนิดข้อมูลให้เป็น Integer
4 from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF # ใช้สำหรับการจัดการข้อความ (text processing)
5 from pyspark.ml.classification import LogisticRegression # ใช้สำหรับการจำแนกประเภทด้วย Logistic Regression
6 from pyspark.ml import Pipeline # ใช้สำหรับการรวมขั้นตอนต่าง ๆ ของการทำงานเป็นขั้นตอนเดียว (pipeline)
7 from pyspark.ml.evaluation import MulticlassClassificationEvaluator # ใช้สำหรับประเมินประสิทธิภาพของโมเดลที่มีหลายคลาส (multiclass)
8
```

TextAnalytics >  TextAnalytics.py > ...

```
8
9 # สร้าง SparkSession เพื่อเริ่มการทำงานกับ PySpark
10 spark = SparkSession.builder.appName("TextAnalytics").getOrCreate()
11
12 # อ่านข้อมูลจากไฟล์ CSV (แทนที่ 'reviewsRated.csv' ด้วยชื่อไฟล์ที่คุณใช้งานจริง)
13 # data เป็นตัวแปรที่เก็บข้อมูลรีวิวที่อ่านมาจากไฟล์ CSV
14 # header=True หมายความว่าบรรทัดแรกของไฟล์ CSV เป็นหัวคอลัมน์
15 # inferSchema=True เพื่อให้ Spark เดาชุดข้อมูลว่ามีชนิดข้อมูลเป็นอะไร
16 data = spark.read.csv("reviewsRated.csv", header=True, inferSchema=True)
17
18 # เลือกเฉพาะคอลัมน์ "Review Text" และ "Rating" จาก DataFrame
19 # เปลี่ยนชื่อคอลัมน์ "Review Text" เป็น "review_text" และแปลงคอลัมน์ "Rating" เป็นชนิด Integer
20 data = data.select(data["Review Text"].alias("review_text"), data["Rating"].cast(IntegerType()).alias("rating"))
21
22 # ลบข้อมูลที่มีค่าว่างออก (na.drop() ลบแถวที่มีข้อมูลว่าง)
23 data = data.na.drop()
24
25 # แสดงข้อมูล 5 แถวแรกของ DataFrame เพื่อดูว่าข้อมูลถูกต้องหรือไม่
26 data.show(5)
27
28 # Tokenizer: แบ่งข้อความเป็นคำ (แยกแต่ละคำในข้อความออกมา)
29 # tokenizer เป็นตัวแปรที่เก็บขั้นตอนการแปลงข้อความในคอลัมน์ "review_text" ให้กลายเป็นคำในคอลัมน์ "words"
30 tokenizer = Tokenizer(inputCol="review_text", outputCol="words")
31
32 # StopWordsRemover: ลบคำที่ไม่สำคัญ เช่น คำว่า "the", "is", "and" เป็นต้น
33 # stopword_remover เป็นตัวแปรที่เก็บขั้นตอนการลบคำทั่วไป (stop words) ออกจากคอลัมน์ "words"
34 # ผลลัพธ์จะเก็บในคอลัมน์ใหม่ชื่อ "meaningful_words"
35 stopword_remover = StopWordsRemover(inputCol="words", outputCol="meaningful_words")
36
37 # HashingTF: แปลงคำให้กลายเป็นพีเจอร์โดยใช้ Term Frequency (TF)
38 # hashing_tf เป็นตัวแปรที่เก็บขั้นตอนการแปลงคำในคอลัมน์ "meaningful_words" ให้กลายเป็นเวกเตอร์พีเจอร์ในคอลัมน์ "features"
39 hashing_tf = HashingTF(inputCol="meaningful_words", outputCol="features")
40
41 # สร้าง Pipeline เพื่อนำขั้นตอนการแปลงข้อมูล (tokenizer, stopword_remover, hashing_tf) มารวมกัน
42 # pipeline เป็นตัวแปรที่เก็บขั้นตอนการแปลงข้อมูลทั้งหมดเป็นลำดับ (ขั้นตอนการแปลงจะถูกใช้ตามลำดับที่ระบุใน stages)
43 pipeline = Pipeline(stages=[tokenizer, stopword_remover, hashing_tf])
```


TextAnalytics >  TextAnalytics.py > ...

```
44
45 # แบ่งข้อมูลออกเป็นชุดฝึก (train_data) และชุดทดสอบ (test_data)
46 # train_data และ test_data เป็นข้อมูลที่ถูกแบ่งเป็น 80% สำหรับการฝึกโมเดล และ 20% สำหรับการทดสอบ
47 train_data, test_data = data.randomSplit([0.8, 0.2], seed=1234) # seed=1234 ใช้เพื่อให้การแบ่งข้อมูลได้ผลลัพธ์ที่ทำซ้ำได้
48
49 # ฝึก Pipeline ด้วยข้อมูลชุดฝึก (train_data)
50 # pipeline_model เป็นตัวแปรที่เก็บโมเดลที่ถูกฝึกเสร็จแล้ว
51 pipeline_model = pipeline.fit(train_data)
52
53 # แปลงข้อมูลชุดฝึกและชุดทดสอบด้วย pipeline ที่ถูกฝึกแล้ว
54 # train_transformed และ test_transformed เป็นข้อมูลที่ถูกแปลงแล้วตามขั้นตอนใน pipeline
55 train_transformed = pipeline_model.transform(train_data)
56 test_transformed = pipeline_model.transform(test_data)
57
58 # แสดงคอลัมน์ "meaningful_words", "features", และ "rating" ของข้อมูลที่ถูกแปลงแล้ว
59 train_transformed.select("meaningful_words", "features", "rating").show(5)
60
61 # สร้างโมเดล Logistic Regression สำหรับการจำแนกประเภท (classification)
62 # log_reg เป็นตัวแปรที่เก็บโมเดล Logistic Regression ซึ่งจะใช้ข้อมูลในคอลัมน์ "features" เพื่อทำนายคอลัมน์ "rating"
63 log_reg = LogisticRegression(labelCol="rating", featuresCol="features")
64
65 # ฝึกโมเดล Logistic Regression ด้วยข้อมูลชุดฝึกที่ถูกแปลงแล้ว
66 # log_reg_model เป็นตัวแปรที่เก็บโมเดล Logistic Regression ที่ถูกฝึกเสร็จแล้ว
67 log_reg_model = log_reg.fit(train_transformed)
68
69 # ใช้โมเดลที่ฝึกแล้วทำนายข้อมูลในชุดทดสอบ
70 # predictions เป็นตัวแปรที่เก็บผลลัพธ์ที่ถูกทำนายจากข้อมูลชุดทดสอบ
71 predictions = log_reg_model.transform(test_transformed)
72
73 # แสดงผลลัพธ์ของการทำนาย โดยเลือกคอลัมน์ "meaningful_words" (คำที่มีความหมาย), "rating" (ค่าจริง), และ "prediction" (ค่าทำนาย)
74 predictions.select("meaningful_words", "rating", "prediction").show(5)
75
76 # สร้างตัวประเมินผลลัพธ์ของโมเดล Logistic Regression
77 # evaluator เป็นตัวแปรที่เก็บตัวประเมินผลลัพธ์สำหรับการจำแนกประเภทแบบหลายคลาส (multiclass classification)
78 evaluator = MulticlassClassificationEvaluator(labelCol="rating", predictionCol="prediction", metricName="accuracy")
79
```

TextAnalytics >  TextAnalytics.py > ...

```
79
80 # คำนวณความแม่นยำ (accuracy) ของโมเดล
81 # accuracy เก็บค่าความแม่นยำของโมเดล Logistic Regression
82 accuracy = evaluator.evaluate(predictions)
83
84 # แสดงค่าความแม่นยำ (accuracy) ของโมเดล
85 print(f"Accuracy: {accuracy}")
86
```

```
PS C:\Users\rawip\Downloads\BigData\TextAnalytics> python .\Tex
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For Spark
```

```
+-----+-----+
|      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
```

```
24/10/29 02:01:38 WARN InstanceBuilder: Failed to load implemen
24/10/29 02:01:38 WARN InstanceBuilder: Failed to load implemen
24/10/29 02:02:25 WARN DAGScheduler: Broadcasting large task bi
```

```
+-----+-----+-----+
| meaningful_words|rating|prediction|
+-----+-----+-----+
|[, amazon, custom...|    5|    5.0|
|[, default, sure,...|    1|    4.0|
|["**title:, decep...|    1|    4.0|
|["1., never, get,...|    1|    1.0|
|["1.7, rating, ti...|    4|    5.0|
+-----+-----+-----+
only showing top 5 rows
```

AssociationRules.py

โค้ดนี้เป็นตัวอย่างการใช้งาน FPGrowth ซึ่งเป็นอัลกอริทึมสำหรับการค้นหาความสัมพันธ์ระหว่างรายการ (association rule mining) หรือการหากฎบ่อยที่เกิดขึ้นร่วมกันในข้อมูลการซื้อสินค้า (frequent itemset mining) โดยมีรายละเอียดดังนี้:

```
Association Rule > AssociationRules.py > ...
1  from pyspark import SparkContext
2  from pyspark.sql import SparkSession # SparkSession คือจุดเริ่มต้นสำหรับการทำงานกับ PySpark
3  from pyspark.ml.fpm import FPGrowth # FPGrowth ใช้สำหรับการวิเคราะห์การเกิดซ้ำของชุดข้อมูลที่พบบ่อย
4  from pyspark.sql.functions import collect_list, array_distinct, explode, split, col # ฟังก์ชันเหล่านี้ใช้จัดการข้อมูลใน DataFrame
5
6  # Step 1: สร้าง SparkSession เพื่อเริ่มใช้งาน PySpark
7  spark = SparkSession.builder.appName("FPGrowthExample").getOrCreate()
8  # สร้าง SparkSession โดยตั้งชื่อว่า "FPGrowthExample" และเริ่มดำเนินการทำงานกับ PySpark
9
10 # Step 2: อ่านข้อมูลจากไฟล์ CSV (สมมติว่าเป็นข้อมูลสินค้าจากร้านขายของชำ)
11 data = spark.read.csv(r"C:\Users\sooke\Desktop\big data\Association Rule\groceries_data.csv", header=True, inferSchema=True)
12 # อ่านข้อมูลจากไฟล์ CSV โดยมีพารามิเตอร์:
13 # header=True: ระบุว่าข้อมูลมีหัวคอลัมน์ในไฟล์
14 # inferSchema=True: ให้ PySpark ตรวจสอบประเภทข้อมูลอัตโนมัติ (เช่น ตัวเลข ข้อความ)
15
16 # Step 3: รวมกลุ่มข้อมูลตาม Member_number และรวมรวม itemDescription เป็นรายการสินค้า จาก itemDescription เปลี่ยนใหม่เป็น Items
17 grouped_data = data.groupBy("Member_number").agg(collect_list("itemDescription").alias("Items"))
18 # groupBy(): ทำการจัดกลุ่มข้อมูลตาม "Member_number"
19 # collect_list(): รวมรายการสินค้า (itemDescription) สำหรับสมาชิกแต่ละคนให้เป็นลิสต์เดียวในคอลัมน์ "Items"
20
21 # Step 4: แสดงข้อมูลที่จัดกลุ่ม (truncate=False เพื่อแสดงข้อมูลแบบเต็ม)
22 grouped_data.show(truncate=False)
23 # แสดงผลข้อมูลที่จัดกลุ่มตามสมาชิกและรวมรายการสินค้าที่ซื้อ (ไม่ตัดข้อความในคอลัมน์)
24
25 # Step 5: เพิ่มคอลัมน์ 'basket' ที่มีรายการสินค้าที่ไม่ซ้ำกัน
26 grouped_data = grouped_data.withColumn("basket", array_distinct(grouped_data["Items"]))
27 # เพิ่มคอลัมน์ใหม่ชื่อ "basket" โดยใช้ฟังก์ชัน array_distinct() เพื่อเอาสินค้าที่ซ้ำกันออกจากลิสต์
28
29 # Step 6: แสดงข้อมูลอีกครั้ง
30 grouped_data.show(truncate=False)
31 # แสดงผลข้อมูลอีกครั้งเพื่อดูรายการสินค้าที่ไม่ซ้ำกันในคอลัมน์ "basket"
32
33 # Step 7: แดก (explode) อาร์เรย์ Items เพื่อแยกรายการสินค้าให้อยู่ในรูปแบบของแถว เปลี่ยนใหม่เป็น item
34 exploded_data = grouped_data.select("Member_number", explode("Items").alias("item"))
35 # explode(): ฟังก์ชันที่ใช้แยกแต่ละรายการสินค้าที่อยู่ในลิสต์ออกมาเป็นแถวใหม่
36
37 # Step 8: แทนที่เครื่องหมาย '/' ด้วย ',' เพื่อแยกประเภทสินค้าที่รวมกันในรายการเดียว
38 # milk\eggs\bread ถ้ามันเจอค่าแบบนี้ในไฟล์ csv มันจะทำการแยกออกจากกันเลย
39 separated_data = exploded_data.withColumn("item", explode(split("item", "/")))
40 # split(): ใช้สำหรับแยกค่าที่ถูกเชื่อมด้วย '/' ให้ออกเป็นรายการย่อย แล้ว explode() เพื่อแสดงในรูปแบบแถวใหม่
41
42 # Step 9: รวมรายการสินค้ากลับเข้าไปในรูปแบบอาร์เรย์และให้มั่นใจว่าไม่มีรายการที่ซ้ำกัน
43 final_data = separated_data.groupBy("Member_number").agg(collect_list("item").alias("Items"))
44 # นำรายการสินค้าที่ถูกแยกแล้วกลับมารวมเป็นลิสต์ของสินค้าที่สมาชิกซื้อ
45
46 # Step 10: ทำให้รายการในอาร์เรย์ Items ไม่ซ้ำกัน
47 final_data = final_data.withColumn("Items", array_distinct(col("Items")))
48 # ใช้ฟังก์ชัน array_distinct() เพื่อลบรายการสินค้าที่ซ้ำกันออกจากคอลัมน์ "Items"
```

```

49
50 # Step 11: แสดงข้อมูลที่แยกเรียบร้อยแล้ว
51 final_data.show(truncate=False)
52 # แสดงข้อมูลสุดท้ายที่ถูกแยกและจัดเรียบร้อยแล้ว โดยแต่ละสมาชิกมีรายการสินค้าที่ไม่ซ้ำกัน
53
54 # Step 12: สร้างโมเดล FPGrowth โดยกำหนดค่า minSupport และ minConfidence
55 minSupport = 0.1 # กำหนดค่าการสนับสนุนขั้นต่ำ (Support) เป็น 10% ของชุดข้อมูล
56 minConfidence = 0.2 # กำหนดค่าความเชื่อมั่นขั้นต่ำ (Confidence) เป็น 20%
57
58 # itemsCol='Items': กำหนดว่าคอลัมน์ "Items" จะใช้สำหรับการฝึกโมเดล
59 # predictionCol='prediction': คอลัมน์ที่จะเก็บผลการทำนาย
60 fp = FPGrowth(minSupport=minSupport, minConfidence=minConfidence, itemsCol='Items', predictionCol='prediction')
61
62 # Step 13: ฝึกโมเดล FPGrowth กับข้อมูลที่แยกเรียบร้อยแล้ว
63 model = fp.fit(final_data)
64 # ฝึกโมเดล FPGrowth ด้วยข้อมูล 'final_data' ที่มีรายการสินค้าที่สมาชิกซื้อ
65
66 # Step 14: แสดง itemsets ที่พบบ่อยในข้อมูล
67 model.freqItemsets.show(10) # แสดง 10 อันดับแรกของชุดรายการสินค้าที่พบบ่อย
68
69 # Step 15: กรองกฎความสัมพันธ์ (association rules) โดยใช้ค่า confidence ที่มากกว่า 0.4
70 filtered_rules = model.associationRules.filter(model.associationRules.confidence > 0.4)
71 # กรองกฎความสัมพันธ์ที่มีค่า confidence มากกว่า 40% (0.4)
72
73 # Step 16: แสดงกฎความสัมพันธ์ที่ถูกกรอง
74 filtered_rules.show(truncate=False)
75 # แสดงกฎความสัมพันธ์ที่ถูกกรองแล้ว
76
77 # Step 17: สร้าง DataFrame ใหม่เพื่อใช้ในการทำนาย (prediction)
78 new_data = spark.createDataFrame(
79     [
80         (["vegetable juice", "frozen fruits", "packaged fruit"],), # รายการสินค้า 1 บาสเก็ต
81         (["mayonnaise", "butter", "buns"],) # รายการสินค้าอีก 1 บาสเก็ต
82     ],
83     ["Items"] # คอลัมน์ต้องชื่อว่า "Items" เพื่อให้สอดคล้องกับข้อมูลที่ใช้ในการฝึกโมเดล
84 )
85
86 # Step 18: แสดงข้อมูลใหม่ที่จะใช้ในการทำนาย
87 new_data.show(truncate=False)
88 # แสดงข้อมูลใหม่ที่เราสร้างขึ้นสำหรับการทำนาย
89
90 # Step 19: ใช้โมเดลในการทำนายสินค้าที่อาจจะซื้อพร้อมกับข้อมูลใหม่
91 predictions = model.transform(new_data)
92 # ใช้โมเดล FPGrowth ที่ฝึกแล้วทำนายผลจากข้อมูลใหม่
93
94 # Step 20: แสดงผลการทำนาย
95 predictions.show(truncate=False)
96 # แสดงผลการทำนายว่าในแต่ละบาสเก็ตควรจะซื้อสินค้าที่จะซื้อพร้อมกันอะไรบ้าง
97
98 # ปิดการทำงานของ SparkSession
99 spark.stop()
100 # ปิด SparkSession เพื่อคืนทรัพยากรระบบ

```

Member_number	Items
1000	[soda, canned beer, sausage, sausage, whole milk, whole milk, pickled vegetables, misc. beverages, semi-finished bread, hygiene articles, yogurt, pastry, salty snack]
1001	[frankfurter, frankfurter, beef, sausage, whole milk, soda, curd, white bread, whole milk, soda, whipped/sour cream, rolls/buns]
1002	[tropical fruit, butter milk, butter, frozen vegetables, sugar, specialty chocolate, whole milk, other vegetables]
1003	[sausage, root vegetables, rolls/buns, detergent, frozen meals, rolls/buns, dental care, rolls/buns]
1004	[other vegetables, pip fruit, root vegetables, canned beer, rolls/buns, whole milk, other vegetables, hygiene articles, whole milk, whole milk, frozen fish, frozen fish]
1005	[whipped/sour cream, rolls/buns, margarine, rolls/buns]
1006	[whole milk, frankfurter, chicken, frankfurter, whole milk, bottled water, flour, chocolate, bottled beer, rolls/buns, rice, softener, shopping bags, rolls/buns]
1008	[hamburger meat, tropical fruit, soda, liquor (appetizer), yogurt, liver loaf, root vegetables, yogurt, dessert, domestic eggs, white wine, photo/film]
1009	[herbs, pastry, tropical fruit, yogurt, ketchup, yogurt, canned fish, newspapers, cocoa drinks]
1010	[pip fruit, frankfurter, pip fruit, bottled water, candles, bottled water, coffee, specialty bar, kitchen towels, rolls/buns, UHT-milk, sliced cheese]
1011	[whole milk, frankfurter, candles, citrus fruit, curd cheese, grapes, herbs, other vegetables, candy, pastry, bottled water, yogurt, rolls/buns]
1012	[frankfurter, processed cheese, tropical fruit, root vegetables, yogurt, whole milk, rolls/buns, onions, frozen vegetables, shopping bags, brown bread]
1013	[frozen meals, white bread, meat, whole milk, tropical fruit, mustard, hard cheese, root vegetables, other vegetables, whole milk, domestic eggs, roll products]
1014	[whole milk, canned beer, sausage, hamburger meat, whole milk, bottled beer, cookware, yogurt, butter milk, rolls/buns]
1015	[citrus fruit, salty snack, fruit/vegetable juice, whole milk, beef, rolls/buns, chocolate]
1016	[UHT-milk, soft cheese, rolls/buns, chicken, frankfurter, bottled beer, pip fruit, oil, red/blush wine, pip fruit, mayonnaise]
1017	[soda, other vegetables, yogurt, other vegetables, dessert, rolls/buns, beef, root vegetables, brown bread, shopping bags, pet care]
1018	[butter milk, curd, berries, beverages, root vegetables, long life bakery product, cake bar, domestic eggs]
1019	[hamburger meat, red/blush wine]
1020	[canned beer, canned beer, frozen meals, spices, rolls/buns, shopping bags, butter, frozen fish, newspapers, yogurt]

only showing top 20 rows

Member_number	Items
1000	[soda, canned beer, sausage, sausage, whole milk, whole milk, pickled vegetables, misc. beverages, semi-finished bread, hygiene articles, yogurt, pastry, salty snack]
1001	[frankfurter, frankfurter, beef, sausage, whole milk, soda, curd, white bread, whole milk, soda, whipped/sour cream, rolls/buns]
1002	[tropical fruit, butter milk, butter, frozen vegetables, sugar, specialty chocolate, whole milk, other vegetables]
1003	[sausage, root vegetables, rolls/buns, detergent, frozen meals, rolls/buns, dental care, rolls/buns]
1004	[other vegetables, pip fruit, root vegetables, canned beer, rolls/buns, whole milk, other vegetables, hygiene articles, whole milk, whole milk, frozen fish, frozen fish]
1005	[whipped/sour cream, rolls/buns, margarine, rolls/buns]
1006	[whole milk, frankfurter, chicken, frankfurter, whole milk, bottled water, flour, chocolate, bottled beer, rolls/buns, rice, softener, shopping bags, rolls/buns]
1008	[hamburger meat, tropical fruit, soda, liquor (appetizer), yogurt, liver loaf, root vegetables, yogurt, dessert, domestic eggs, white wine, photo/film]
1009	[herbs, pastry, tropical fruit, yogurt, ketchup, yogurt, canned fish, newspapers, cocoa drinks]
1010	[pip fruit, frankfurter, pip fruit, bottled water, candles, bottled water, coffee, specialty bar, kitchen towels, rolls/buns, UHT-milk, sliced cheese]
1011	[whole milk, frankfurter, candles, citrus fruit, curd cheese, grapes, herbs, other vegetables, candy, pastry, bottled water, yogurt, rolls/buns]
1012	[frankfurter, processed cheese, tropical fruit, root vegetables, yogurt, whole milk, rolls/buns, onions, frozen vegetables, shopping bags, brown bread]
1013	[frozen meals, white bread, meat, whole milk, tropical fruit, mustard, hard cheese, root vegetables, other vegetables, whole milk, domestic eggs, roll products]
1014	[whole milk, canned beer, sausage, hamburger meat, whole milk, bottled beer, cookware, yogurt, butter milk, rolls/buns]
1015	[citrus fruit, salty snack, fruit/vegetable juice, whole milk, beef, rolls/buns, chocolate]
1016	[UHT-milk, soft cheese, rolls/buns, chicken, frankfurter, bottled beer, pip fruit, oil, red/blush wine, pip fruit, mayonnaise]
1017	[soda, other vegetables, yogurt, other vegetables, dessert, rolls/buns, beef, root vegetables, brown bread, shopping bags, pet care]
1018	[butter milk, curd, berries, beverages, root vegetables, long life bakery product, cake bar, domestic eggs]
1019	[hamburger meat, red/blush wine]
1020	[canned beer, canned beer, frozen meals, spices, rolls/buns, shopping bags, butter, frozen fish, newspapers, yogurt]

only showing top 20 rows

Member_number	Items
1000	[soda, canned beer, sausage, whole milk, pickled vegetables, misc. beverages, semi-finished bread, hygiene articles, yogurt, pastry, salty snack]
1001	[frankfurter, beef, sausage, whole milk, soda, curd, white bread, whipped, sour cream, rolls, buns]
1002	[tropical fruit, butter milk, butter, frozen vegetables, sugar, specialty chocolate, whole milk, other vegetables]
1003	[sausage, root vegetables, rolls, buns, detergent, frozen meals, dental care]
1004	[other vegetables, pip fruit, root vegetables, canned beer, rolls, buns, whole milk, hygiene articles, frozen fish, red, blush wine, chocolate, shopping bags, frozen fish]
1005	[whipped, sour cream, rolls, buns, margarine]
1006	[whole milk, frankfurter, chicken, bottled water, flour, chocolate, bottled beer, rolls, buns, rice, softener, shopping bags, skin care]
1008	[hamburger meat, tropical fruit, soda, liquor (appetizer), yogurt, liver loaf, root vegetables, dessert, domestic eggs, white wine, photo, film]
1009	[herbs, pastry, tropical fruit, yogurt, ketchup, canned fish, newspapers, cocoa drinks]
1010	[pip fruit, frankfurter, bottled water, candles, coffee, specialty bar, kitchen towels, rolls, buns, UHT-milk, sliced cheese]
1011	[whole milk, frankfurter, candles, citrus fruit, curd cheese, grapes, herbs, other vegetables, candy, pastry, bottled water, yogurt, rolls, buns]
1012	[frankfurter, processed cheese, tropical fruit, root vegetables, yogurt, whole milk, rolls, buns, onions, frozen vegetables, shopping bags, brown bread]
1013	[frozen meals, white bread, meat, whole milk, tropical fruit, mustard, hard cheese, root vegetables, other vegetables, domestic eggs, roll products, candy, frozen fish]
1014	[whole milk, canned beer, sausage, hamburger meat, bottled beer, cookware, yogurt, butter milk, rolls, buns]
1015	[citrus fruit, salty snack, fruit, vegetable juice, whole milk, beef, rolls, buns, chocolate]
1016	[UHT-milk, soft cheese, rolls, buns, chicken, frankfurter, bottled beer, pip fruit, oil, red, blush wine, mayonnaise]
1017	[soda, other vegetables, yogurt, dessert, rolls, buns, beef, root vegetables, brown bread, shopping bags, pet care]
1018	[butter milk, curd, berries, beverages, root vegetables, long life bakery product, cake bar, domestic eggs]
1019	[hamburger meat, red, blush wine]
1020	[canned beer, frozen meals, spices, rolls, buns, shopping bags, butter, frozen fish, newspapers, yogurt]

only showing top 20 rows


```

+-----+-----+
| items|freq|
+-----+-----+
| [butter]| 493|
| [bottled water]| 833|
| [bottled water, w...]| 438|
| [sour cream]| 603|
| [sour cream, whip...]| 603|
| [curd]| 471|
| [pastry]| 692|
| [coffee]| 448|
| [rolls]|1363|
| [rolls, other veg...]| 572|
+-----+-----+

```

only showing top 10 rows

antecedent	consequent	confidence	lift	support
[other vegetables]	[whole milk]	0.5081743869209809	1.1091062487222754	0.1913801949717804
[yogurt, rolls]	[buns]	1.0	2.85986793837124	0.11133914828116984
[vegetable juice]	[fruit]	1.0	8.004106776180699	0.12493586454592098
[yogurt]	[other vegetables]	0.4252039891205802	1.129049829422358	0.12031811185223192
[yogurt]	[whole milk]	0.5321849501359928	1.1615100423460805	0.15059004617752694
[sausage]	[whole milk]	0.5193026151930261	1.1333939496206136	0.10697793740379682
[rolls, whole milk]	[buns]	1.0	2.85986793837124	0.17855310415597742
[soda, rolls]	[buns]	1.0	2.85986793837124	0.11980502821959979
[whole milk]	[other vegetables]	0.4176931690929451	1.1091062487222754	0.1913801949717804
[rolls]	[other vegetables]	0.41966250917094644	1.1143354637250336	0.14674191893278604
[rolls]	[whole milk]	0.5106382978723404	1.1144838102499344	0.17855310415597742
[rolls]	[buns]	1.0	2.85986793837124	0.3496664956387891
[buns, other vegetables]	[rolls]	1.0	2.85986793837124	0.14674191893278604
[tropical fruit]	[whole milk]	0.4983534577387486	1.0876717683458241	0.11646998460749101
[rolls, other vegetables]	[buns]	1.0	2.85986793837124	0.14674191893278604
[yogurt, buns]	[rolls]	1.0	2.85986793837124	0.11133914828116984
[soda, buns]	[rolls]	1.0	2.85986793837124	0.11980502821959979
[buns, whole milk]	[rolls]	1.0	2.85986793837124	0.17855310415597742
[whipped]	[sour cream]	1.0	6.464344941956883	0.15469471523858389
[buns, rolls]	[other vegetables]	0.41966250917094644	1.1143354637250336	0.14674191893278604

only showing top 20 rows

GraphAnalytics_PowerBI_test2.py

GraphAnalytics_PowerBI > Graptest2.py > ...

```
1 from pyspark.sql import SparkSession
2 from graphframes import GraphFrame
3 from pyspark.sql.functions import desc, col, lit
4
5 # สร้าง SparkSession สำหรับเริ่มการทำงานใน PySpark
6 spark = SparkSession.builder \
7     .appName("CyclingRoutesGraph") \
8     .config("spark.jars.packages", "graphframes:graphframes:0.8.2-spark3.0-s_2.12") \
9     .getOrCreate()
10
11 # โหลดข้อมูลการเดินทางของจักรยานจากไฟล์ CSV ลงใน Spark DataFrame
12 cycling_routes_df = spark.read.csv(r"C:\Users\ADMIN\Downloads\BigData\BigData\GraphAnalytics_PowerBI\cycling.csv", header=True, inferSchema=True)
13
14 # แสดงข้อมูลบางส่วนของ DataFrame เพื่อดูโครงสร้าง
15 cycling_routes_df.show()
16
17 # สร้าง DataFrame สำหรับ vertices โดยใช้ชื่อสถานีจาก FromStationName และ ToStationName ที่ไม่ซ้ำกัน
18 from_stations = cycling_routes_df.select("FromStationName").withColumnRenamed("FromStationName", "id").distinct()
19 to_stations = cycling_routes_df.select("ToStationName").withColumnRenamed("ToStationName", "id").distinct()
20
21 # รวมข้อมูลสถานีต้นทางและปลายทางทั้งหมดให้เป็น vertices ที่ไม่ซ้ำกัน
22 vertices = from_stations.union(to_stations).distinct()
23
24 # สร้าง DataFrame สำหรับ edges โดยใช้ FromStationName เป็น src และ ToStationName เป็น dst
25 edges = cycling_routes_df.select("FromStationName", "ToStationName") \
26     .withColumnRenamed("FromStationName", "src") \
27     .withColumnRenamed("ToStationName", "dst")
28
29 # สร้างกราฟจาก vertices และ edges ที่สร้างขึ้น
30 graph = GraphFrame(vertices, edges)
31
32 # นับและแสดงจำนวน vertices (สถานี) และ edges (เส้นทาง)
33 print("Number of vertices (stations):", graph.vertices.count())
34 print("Number of edges (routes):", graph.edges.count())
35
36 # จัดกลุ่ม edges ตาม src และ dst, กรองเส้นทางที่มีการเดินทางมากกว่า 5 ครั้ง และเพิ่มคอลัมน์สี source และ destination
37 grouped_edges = graph.edges.groupBy("src", "dst").count() \
38     .filter(col("count") >= 3) \
39     .orderBy(desc("count")) \
40     .withColumn("source_color", lit("#FF3F33")) \
41     .withColumn("destination_color", lit("#3358FF"))
42
43 # แสดงข้อมูล edges ที่ถูกจัดกลุ่มและกรองแล้ว
44 grouped_edges.show()
45
46 # ลบคอลัมน์ count ก่อนเขียนลงไฟล์ CSV
47 grouped_edges_no_count = grouped_edges.drop("count")
48
49 # เขียนข้อมูล edges ที่จัดกลุ่มลงในไฟล์ CSV
50 grouped_edges_no_count.write.csv("grouped_cycling_routes8.csv", mode="overwrite", header=False)
51
```