Database Project (SWE3033) (Fall 2024)

Project 3 (100pts, Due date: 11/5)

Student ID: _____2018310773____

Student Name: ____임승재____

Instruction: In this homework, we provide you with a jupyter notebook file (DBP _Project3.ipynb). You should follow the instructions in these documents carefully.

Submit two files as follows:

- DBP_Project3_StudentID.zip
  - DBP_Project3_StudentID.ipynb
  - DBP_Project3_StudentID.pdf

[Spark SQL]

1. [20pts] The following dataset is related to the document.

Data:

- **Already published document → used for problem (a):**

```
[Row(doc_id='1', topic='Mac', timestamp='10009820'),
      Row(doc_id='2', topic='iPhone', timestamp='10009830'),
      Row(doc_id='3', topic='iPhone', timestamp='10009900'),
      Row(doc_id='4', topic='Galaxy', timestamp='10009950'),
      Row(doc_id='5', topic='iPhone', timestamp='10010000'),
      Row(doc_id='6', topic='A100', timestamp='10010010'),
      Row(doc_id='7', topic='Galaxy', timestamp='10010030'),
      Row(doc_id='8', topic='iPhone', timestamp='10010050'),
      Row(doc_id='9', topic='A100', timestamp='10010070')
]
```

- **Additional document→ used for problem (b):**

```
[Row(doc_id='10', topic='A6000', timestamp='10010100'),
      Row(doc_id='11', topic='H100', timestamp='10010500')]
```

(a) Create a DataFrame with the given data and display the generated DataFrame.

[Answer]

Enter your code and result here. You must show your result (captured image).

```
# ============= EDIT HERE =============

RDD = sc.parallelize([
    Row(doc_id='1', topic='Mac', timestamp='10009820'),
    Row(doc_id='2', topic='iPhone', timestamp='10009830'),
    Row(doc_id='3', topic='iPhone', timestamp='10009900'),
```

```
    Row(doc_id='4', topic='Galaxy', timestamp='10009950'),
    Row(doc_id='5', topic='iPhone', timestamp='10010000'),
    Row(doc_id='6', topic='A100', timestamp='10010010'),
    Row(doc_id='7', topic='Galaxy', timestamp='10010030'),
    Row(doc_id='8', topic='iPhone', timestamp='10010050'),
    Row(doc_id='9', topic='A100', timestamp='10010070')
]

)
df = RDD.toDF()

# ========================================

df.show(truncate=False)
```

```
+------+------+---------+
|doc_id|topic |timestamp|
+------+------+---------+
|1     |Mac   |10009820 |
|2     |iPhone|10009830 |
|3     |iPhone|10009900 |
|4     |Galaxy|10009950 |
|5     |iPhone|10010000 |
|6     |A100  |10010010 |
|7     |Galaxy|10010030 |
|8     |iPhone|10010050 |
|9     |A100  |10010070 |
+------+------+---------+
```

(b) After adding the ***two additional documents***, find the timestamp for each document.

[Answer]
Enter your code and result here. You must show your result (captured image).

```
# ============= EDIT HERE =============

additional_doc = sc.parallelize([
    Row(doc_id='10', topic='A6000', timestamp='10010100'),
    Row(doc_id='11', topic='H100', timestamp='10010500')]
)
new_df = df.union(additional_doc.toDF())

# ========================================
new_df.show(truncate=False)
```

```
+------+------+---------+
|doc_id|topic |timestamp|
+------+------+---------+
|1     |Mac   |10009820 |
|2     |iPhone|10009830 |
|3     |iPhone|10009900 |
|4     |Galaxy|10009950 |
|5     |iPhone|10010000 |
|6     |A100  |10010010 |
|7     |Galaxy|10010030 |
|8     |iPhone|10010050 |
|9     |A100  |10010070 |
|10    |A6000 |10010100 |
|11    |H100  |10010500 |
+------+------+---------+
```

(c) Group the data in the joined DataFrame by 'topic' column and count the number of data for each topic.

[Answer]

Enter your code and result here. You must show your result (captured image).

```
# ============== EDIT HERE ==============

group_df = new_df.groupBy('topic').count()

# ========================================
group_df.show(truncate=False)
```

```
+------+-----+
|topic |count|
+------+-----+
|iPhone|4    |
|Galaxy|2    |
|Mac   |1    |
|A100  |2    |
|A6000 |1    |
|H100  |1    |
+------+-----+
```

2. [20pts] The following data are *documentation* and *views information*.

- ***Data 1 - This is the data you completed in problem 1.***
- ***Data 2 - views information:***

```
[
        Row(topic='Mac', view=1000, timestamp=10009820),
        Row(topic='Galaxy', view=200, timestamp=10009950),
        Row(topic='iPhone', view=400, timestamp=10009900),
        Row(topic='A100', view=3000, timestamp=10010070),
        Row(topic='A6000', view=2000, timestamp=10010100),
```

```
        Row(topic='H100',  view=9000,  timestamp=10010500)
]
```

(a) Create a DataFrame for the two given datasets and join Data 1 with Data 2 using an inner join based on the 'topic' and 'timestamp' column. (left side: Data 2, right side: Data 1)

[Hint: on=(condition1) & (condition2)]

[Answer]
Enter your code and result here. You must show your result (captured image).

```
# ============= EDIT HERE =============

RDD = sc.parallelize([
    Row(topic='Mac', view=1000, timestamp=10009820),
    Row(topic='Galaxy', view=200, timestamp=10009950),
    Row(topic='iPhone', view=400, timestamp=10009900),
    Row(topic='A100', view=3000, timestamp=10010070),
    Row(topic='A6000', view=2000, timestamp=10010100),
    Row(topic='H100', view=9000, timestamp=10010500)

])
DF_join = RDD.toDF()
DF_join = DF_join.withColumnRenamed("topic", "topic_join")
DF_join = DF_join.withColumnRenamed("timestamp", "timestamp_join")

DF_join = DF_join.join(new_df, (DF_join.topic_join == new_df.topic) &
(DF_join.timestamp_join == new_df.timestamp), "inner")

# ====================================
DF_join.show()
```

```
+----------+----+--------------+------+------+---------+
|topic_join|view|timestamp_join|doc_id| topic|timestamp|
+----------+----+--------------+------+------+---------+
|      A100|3000|      10010070|     9|  A100| 10010070|
|     A6000|2000|      10010100|    10| A6000| 10010100|
|    Galaxy| 200|      10009950|     4|Galaxy| 10009950|
|      H100|9000|      10010500|    11|  H100| 10010500|
|       Mac|1000|      10009820|     1|   Mac| 10009820|
|    iPhone| 400|      10009900|     3|iPhone| 10009900|
+----------+----+--------------+------+------+---------+
```

(b) Convert the data type of the 'view' and 'timestamp' columns to ***Integer***.

[Answer]
Enter your code and result here. You must show your result (captured image).

```
# ============= EDIT HERE =============

DF_join = DF_join.withColumn("doc_id", DF_join["doc_id"].cast("integer"))
DF_join = DF_join.withColumn("timestamp", DF_join["timestamp"].cast("integer"))

# ====================================
DF_join.printSchema()
```

```
root
 |-- topic_join: string (nullable = true)
 |-- view: long (nullable = true)
 |-- timestamp_join: long (nullable = true)
 |-- doc_id: integer (nullable = true)
 |-- topic: string (nullable = true)
 |-- timestamp: integer (nullable = true)
```

(c) Use an SQL query to select the data from joined DataFrame where the **'view'** is greater than 1500. And briefly explain the method you used.

[Answer]

Enter your code and result here. You must show your result (captured image).

Code:

```
# ============= EDIT HERE =============
DF_join.createOrReplaceTempView("join")
query = """
SELECT *
FROM join
WHERE view > 1500
"""

# ====================================
sqlDF = spark.sql(query)
sqlDF.show()
```

```
+----------+----+--------------+------+-----+---------+
|topic_join|view|timestamp_join|doc_id|topic|timestamp|
+----------+----+--------------+------+-----+---------+
|      A100|3000|      10010070|     9| A100| 10010070|
|     A6000|2000|      10010100|    10|A6000| 10010100|
|      H100|9000|      10010500|    11| H100| 10010500|
+----------+----+--------------+------+-----+---------+
```

Explanation: createOrReplaceTempView method creates a temporary SQL view based on the DataFrame. We can run any SQL queries on this temporary view. So we run "SELECT * FROM join WHERE view > 1500" which slects all rows from the temporary view where the 'view' column has a value greater than 1500.

[Spark ML]

3. [60pts] We provide you with a ***Wine*** dataset.

Dataset Description:
Training set: 142 examples
Test set: 36 examples

Target column:

Type	Description
0	Type 1 wine
1	Type 2 wine
2	Type 3 wine

Feature column:

Column	Description
Alcohol	The alcohol content of the wine, measured as a percentage of the total volume.
Phenols	The concentration of phenolic compounds, which affect the taste, color, and body of the wine.
Color	The color intensity of the wine, typically measured by its absorbance at specific wavelengths.

[Answer]
Enter your code and result here. You must show your result (captured image).

(a) Load the provided dataset, convert it into a DataFrame, and show it. You should follow the following instructions.

-Instructions 1: *Assemble the features into a vector column and name the column "features."*
-Instructions 2: *Rename the target column "Type" to "label."*

[Answer]
Enter your code and result here. You must show your result (captured image).

```
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.sql.functions import col

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

train_df = spark.read.csv("./dataset/train_wine.csv", header=True, inferSchema=True)
test_df = spark.read.csv("./dataset/test_wine.csv", header=True, inferSchema=True)

columns = train_df.columns[1:]

# ============= EDIT HERE =============

# Instruction 1: Assemble the features into a vector column and name the column "features"
# Instruction 2: Rename the target column to "label"
assembler = VectorAssembler(inputCols=columns, outputCol="features")
train_df = assembler.transform(train_df)
train_df = train_df.withColumnRenamed("Type", "label")
train_df = train_df.select("features", "label")

test_df = assembler.transform(test_df)
```

```
test_df = test_df.withColumnRenamed("Type", "label")
test_df = test_df.select("features", "label")

train_df.show()
test_df.show()

# ======================================
```

```
+----------------+-----+
|        features|label|
+----------------+-----+
|  [12.0,1.45,3.6]|  2.0|
|  [12.72,2.2,3.9]|  2.0|
| [12.08,2.56,2.9]|  2.0|
|  [14.1,2.75,6.2]|  1.0|
| [13.74,2.6,5.85]|  1.0|
|[12.37,1.98,1.95]|  2.0|
|[13.73,1.28,6.62]|  3.0|
|  [14.22,3.0,5.1]|  1.0|
| [14.22,3.2,6.38]|  1.0|
| [13.05,3.0,5.04]|  1.0|
|  [12.33,1.9,3.4]|  2.0|
| [13.76,2.95,5.4]|  1.0|
|  [14.19,3.3,8.7]|  1.0|
|  [13.72,3.4,6.8]|  1.0|
| [11.79,2.13,3.0]|  2.0|
| [14.16,1.68,9.7]|  3.0|
|[13.32,1.93,8.42]|  3.0|
|  [13.75,2.6,5.6]|  1.0|
|  [12.08,1.6,2.4]|  2.0|
|  [12.47,2.5,2.6]|  2.0|
+----------------+-----+
only showing top 20 rows

+----------------+-----+
|        features|label|
+----------------+-----+
|[13.69,1.83,5.88]|  3.0|
| [12.42,2.0,2.06]|  2.0|
|  [13.64,2.7,5.1]|  1.0|
|[12.21,1.85,2.85]|  2.0|
|  [13.77,3.0,6.3]|  1.0|
| [13.49,1.62,5.7]|  3.0|
| [11.76,1.75,3.8]|  2.0|
|  [14.38,3.3,7.5]|  1.0|
| [12.36,2.3,7.65]|  3.0|
| [12.72,1.38,3.3]|  2.0|
|  [14.12,2.2,5.0]|  1.0|
| [13.24,2.8,4.32]|  1.0|
| [12.22,2.36,2.7]|  2.0|
|[13.88,3.25,5.43]|  1.0|
| [11.84,2.2,3.05]|  2.0|
|[11.41,2.48,3.08]|  2.0|
|  [13.11,2.2,7.1]|  3.0|
|  [13.48,2.7,5.1]|  1.0|
| [12.42,1.68,2.7]|  2.0|
| [13.58,2.86,6.9]|  1.0|
+----------------+-----+
only showing top 20 rows
```

(b) Use K-means clustering to classify the data. Then, provide the prediction count for each wine type with following process.

- Step 1: *Fit k-means.* ***(k is 2)***
*- **Step 2:** Extract the output for the training data and 'select' the 'type' and 'prediction' columns.*
*- **Step 3:** Group the data by both the 'type' and 'prediction' columns, then the count the occurrence for each group.*

Type	prediction	Count
2.0	1	13
1.0	0	7
3.0	1	4
1.0	1	7
3.0	0	5

Note: There is no correct answer. Only the correctness of the code is considered when scoring.

[Answer]

Enter your code and result here. You must show your result (captured image).

```
### 3-(b)

# ============= EDIT HERE =============
kmeans = KMeans(
        featuresCol='features',
        predictionCol='prediction',
        k=2,
        maxIter=20,
        distanceMeasure='euclidean')
model = kmeans.fit(train_df)
predictions = model.transform(test_df)
selected_columns = predictions.select("label", "prediction")
predictions = predictions.withColumnRenamed("label","Type")
output_groupby_columns = predictions.groupBy('Type', 'prediction')

# ====================================

output_groupby_columns.count().show()

+----+----------+-----+
|Type|prediction|count|
+----+----------+-----+
| 2.0|         1|   13|
| 1.0|         0|    7|
| 3.0|         1|    4|
| 1.0|         1|    7|
| 3.0|         0|    5|
+----+----------+-----+
```

(c) Train models to classify the classes of the **Wine** dataset and **report the results for test data**. The models used are **Logistic Regression, Decision Tree, and SVM** (SVC)**.**

For detailed explanations of the usage of each model, please refer to the official documentation below.

- Logistic Regression: [Link]
- Decision Tree: [Link]
- SVM (SVC): [Link]

Note: Train with Type 1 and Type 2 wines only, meaning you must remove Type 3 wines from your dataset (we recommend utilizing *filter* function).

Type	Description
0	Type 1 wine
1	Type 2 wine

[Answer]
Fill in the table below.

	Logistic Regression	Decision Tree	SVM (SVC)
Test accuracy	1.0	0.9166666666666666	0.9629629629629629

Enter your code and result here. You must show your result (captured image).

1. Logistic Regression

```
from pyspark.ml.classification import LogisticRegression

# Training and Test
# ============= EDIT HERE =============
lr_model = LogisticRegression(featuresCol='features', labelCol='label', maxIter=10)
lr_preds = lr_model.fit(train_df).transform(test_df)
lr_accuracy = evaluator.evaluate(lr_preds)

# ======================================

print(f"Accuracy: {lr_accuracy}")
```

```
Accuracy: 1.0
```

2. Decision Tree

```
from pyspark.ml.classification import DecisionTreeClassifier

# On model declaration, fix seed, maxDepth,  to the following values
seed = 2024

# ============= EDIT HERE =============
```

```
dt_model = DecisionTreeClassifier(featuresCol='features', labelCol='label', seed=seed)
dt_preds = dt_model.fit(train_df).transform(test_df)
dt_accuracy = evaluator.evaluate(dt_preds)

# ========================================

print(f"Accuracy: {dt_accuracy}")
```

```
Accuracy: 0.9166666666666666
```

3. SVM

```
from pyspark.ml.classification import LinearSVC

# On model declaration, fix seed to the following values
seed = 2024

# ============== EDIT HERE ==============
from pyspark.sql.functions import when

filtered_train_df = train_df.filter((train_df['label'] == 1) | (train_df['label'] == 2)).withColumn('label', when(train_df['label'] == 1, 0).otherwise(1))
filtered_test_df = test_df.filter((test_df['label'] == 1) | (test_df['label'] == 2)).withColumn('label', when(test_df['label'] == 1, 0).otherwise(1))

svm_model = LinearSVC(featuresCol='features', labelCol='label')
svm_preds = svm_model.fit(filtered_train_df).transform(filtered_test_df)
rfc_accuracy = evaluator.evaluate(svm_preds)

# ========================================

print(f"Accuracy: {rfc_accuracy}")
```

```
Accuracy: 0.9629629629629629
```