Homework Assignment: Multimodal Dataset Analysis with PySpark and Weights & Biases

Objective

You will analyze two real-world multimodal datasets using PySpark and log summary statistics to Weights & Biases (wandb). You will:

- Write distributed data analysis pipelines
- Simulate a MapReduce-style workflow
- Log dataset insights for visual exploration
- Practice working with real-world AI benchmarks (Qilin and MSCOCO)

Installation:

1. Prerequisites

1. **Spark Environment**
    - A working Apache Spark 3.x installation (either local or cluster).
    - Python 3.8+ available to both driver and executors.

The following Python libraries installed in your Spark environment (driver & executors):

```
pip install wandb matplotlib seaborn
```

-

    - Access to a W&B account for logging results.

Script File
Download (or copy) the provided PySpark script into your project directory and name it:

Environment Setup

1. **Install spark**

If you get error as “JAVA_HOME is not set” means Spark can’t find a Java installation. You need to point JAVA_HOME at your JDK’s install directory before running spark-submit. For example, on most Ubuntu/Debian systems:

Install a JDK (if you haven’t yet):

```
sudo apt update

sudo apt install openjdk-11-jdk
```

Find where Java was installed.

Typically it will be under /usr/lib/jvm/. You can list available JDKs:

```
ls /usr/lib/jvm/
```

You might see something like:

```
java-11-openjdk-amd64  java-8-openjdk-amd64  ...
```

Export JAVA_HOME to that path. For example, if you installed OpenJDK 11:

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64

export PATH=$JAVA_HOME/bin:$PATH
```

You can add those two lines to your ~/.bashrc or ~/.profile so they persist across sessions.

Verify:

```
echo $JAVA_HOME

# should print /usr/lib/jvm/java-11-openjdk-amd64 (or your JDK path)

java -version

# should show your Java version (e.g., openjdk version "11.0.x")
```

Option 1: Install Apache Spark System-Wide

Spark requires a JDK. On Ubuntu/Debian:

```
sudo apt update
sudo apt install -y openjdk-11-jdk
```

1.

Download & unpack Spark
Go to https://spark.apache.org/downloads.html, choose a Spark release (e.g. “Spark 3.4.0 with Hadoop 3.3 and later”), then in your terminal:

```
wget
https://dlcdn.apache.org/spark/spark-3.4.0/spark-3.4.0-bin-hadoo
p3.tgz
tar -xvzf spark-3.4.0-bin-hadoop3.tgz
mv spark-3.4.0-bin-hadoop3 ~/spark
```

2.

Set environment variables
Add these lines to your `~/.bashrc` (or `~/.profile`):

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export SPARK_HOME=~/spark
export PATH=$SPARK_HOME/bin:$PATH
```

Then reload:

```
source ~/.bashrc
```

3.

Verify

```
spark-submit --version
# should print Spark version and other info
```

4.

Run your script
Now you can do:

```
spark-submit msmarco_analysis_spark_wandb.py \
  --queries /home/youruser/data/queries.tsv \
  --qrels   /home/youruser/data/qrels.tsv \
  --coll    /home/youruser/data/collection.tsv \
  --project your_wandb_project \
```

5. `--run_name your_spark_run_name`

2. **create a project folder** (e.g. `msmarco-spark-homework`).

Install dependencies in the same Python environment Spark uses:

```
pip install wandb matplotlib seaborn
```

3. **Verify Spark access**. On the command line, run:

   ```
   spark-submit --version
   ```
4. Ensure it prints your Spark version without errors.

Log into W&B:

```
wandb login
```

5. Use your API key to authenticate.

Part 1: Qilin Dataset Analysis

Task

Create a file named `qilin_analysis_spark_wandb.py`.

Your script should:

1. Load one of the following configs from the Hugging Face dataset `THUIR/Qilin`:
    - `search_train`, `search_test`, `recommendation_train`, or `recommendation_test`
2. Save the dataset to `.parquet` using `datasets` + `pandas`
3. Load it using PySpark
4. Compute and log the following:

For search configs:

- Average query length
- Average number of candidates per query
- Average number of clicks per query
- Top 10 most-clicked `note_id`s (use `explode()` and `groupBy()`)

For recommendation configs:

- Average recommendation list size (`rec_result_details_with_idx`)
- Average number of recent clicks (`recent_clicked_note_idxs`)
- Top 10 most recommended `note_idx`s

Logging

Log these to wandb:

- Summary metrics (average lengths, click counts)
- Top 10 clicked/recommended items using `wandb.Table`

Part 2: MSCOCO Dataset Analysis

Task

Create a file named `coco_analysis_spark_wandb.py`.

Your script should:

1. Load the `HuggingFaceM4/COCO` dataset using `trust_remote_code=True`
2. Extract:
    - `image_id`
    - `caption`
    - Image size (`width`, `height`)
3. Save as `.parquet`
4. Load with PySpark
5. Implement a MapReduce-style pipeline:

Mapper:

- Add `caption_length` column (char count)

Reducer:

- Group by `image_id`
- Compute:
    - Number of captions per image
    - Average caption length per image

Bonus:

- Bin caption lengths and count per bin (e.g. 0–4, 5–9, 10–14…)
- Log image size stats (average width/height)

Logging

Log to wandb:

- `avg_caption_length`, `avg_captions_per_image`
- Caption length distribution histogram (as `wandb.Table`)
- Average image width and height

Deliverables

- `qilin_analysis_spark_wandb.py`
- `coco_analysis_spark_wandb.py`
- Screenshots or shared wandb links showing:
  - Metrics
  - Tables
  - Charts

Example Run Commands

```
spark-submit qilin_analysis_spark_wandb.py \
  --config search_train \
  --project wandb_qilin \
  --run_name qilin_search_stats
```

```
spark-submit coco_analysis_spark_wandb.py \
  --project wandb_coco \
  --run_name coco_caption_mapreduce
```

Grading Rubric (Total: 100 pts)

Criterion	Points
Script runs without error	10
Dataset correctly loaded and saved	10
Mapper + reducer logic implemented correctly	10
Accurate stats computation	10
Top-N item computation (explode + groupBy)	10
wandb logging (metrics + tables)	10
Caption length histogram (MSCOCO)	10
Image width/height stats (MSCOCO)	10
Code readability and structure	10
Bonus: Join Qilin `note_idx` to `notes` corpus	+5

1. Starter Template: `qilin_analysis_spark_wandb.py`

```
import argparse
import os
import wandb
from datasets import load_dataset
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, length, count, avg, floor
```

```
def get_spark(app_name="COCOAnalysis"):
    return SparkSession.builder.appName(app_name).getOrCreate()


def save_coco_to_parquet(output_path="./coco_parquet"):
    ds = load_dataset("HuggingFaceM4/COCO", split="train",
trust_remote_code=True)
    records = []
    for row in ds:
        img = row["image"]
        records.append({
            "image_id": row["image_id"],
            "caption": row["caption"],
            "width": img.size[0],
            "height": img.size[1],
        })
    import pandas as pd
    df = pd.DataFrame(records)
    path = os.path.join(output_path, "coco_train.parquet")
    os.makedirs(output_path, exist_ok=True)
    df.to_parquet(path)
    return path


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--parquet_path",
default="./coco_parquet/coco_train.parquet")
    parser.add_argument("--project", default="wandb_coco")
    parser.add_argument("--run_name", default="coco_analysis")
    args = parser.parse_args()

    wandb.init(project=args.project, name=args.run_name)
    run = wandb.run

    spark = get_spark()

    if not os.path.exists(args.parquet_path):
        args.parquet_path =
save_coco_to_parquet(os.path.dirname(args.parquet_path))

    df = spark.read.parquet(args.parquet_path)
    # TODO: Add caption_length column
    # TODO: Group by image_id to get num_captions and avg_caption_length
    # TODO: Log caption length histogram
    # TODO: Log image width/height summary

    run.finish()
    spark.stop()


if __name__ == "__main__":
    main()
```

2. Starter Template: coco_analysis_spark_wandb.py

```
import argparse
import os
import wandb
from datasets import load_dataset
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, length, count, avg, floor


def get_spark(app_name="COCOAnalysis"):
    return SparkSession.builder.appName(app_name).getOrCreate()


def save_coco_to_parquet(output_path="./coco_parquet"):
    ds = load_dataset("HuggingFaceM4/COCO", split="train",
trust_remote_code=True)
    records = []
    for row in ds:
        img = row["image"]
        records.append({
            "image_id": row["image_id"],
            "caption": row["caption"],
            "width": img.size[0],
            "height": img.size[1],
        })
    import pandas as pd
    df = pd.DataFrame(records)
    path = os.path.join(output_path, "coco_train.parquet")
    os.makedirs(output_path, exist_ok=True)
    df.to_parquet(path)
    return path


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--parquet_path",
default="./coco_parquet/coco_train.parquet")
    parser.add_argument("--project", default="wandb_coco")
    parser.add_argument("--run_name", default="coco_analysis")
    args = parser.parse_args()

    wandb.init(project=args.project, name=args.run_name)
    run = wandb.run

    spark = get_spark()

    if not os.path.exists(args.parquet_path):
        args.parquet_path =
save_coco_to_parquet(os.path.dirname(args.parquet_path))

    df = spark.read.parquet(args.parquet_path)

    # TODO: Add caption_length column
    # TODO: Group by image_id to get num_captions and avg_caption_length
    # TODO: Log caption length histogram
    # TODO: Log image width/height summary

    run.finish()
    spark.stop()


if __name__ == "__main__":
    main()
```