I was asked to submit below files regarding this assignment project.

Zip file should contain folders with this organization

code/

github\_build\_action.yml Dockerfile.Dockerfile

script/run.sh : Bash Script to start to generate the 2 files.

Config files in code/config:

config.yaml : config file in yaml format in config/ sub-folder Source code in code/src/

screenshots/ docker\_build.png pip\_freeze.png dataprocessed.png data\_processed\_all.png

logs/

Docker\_build.txt : Docker build log pip\_list.txt : pip list inside the docker Data\_processed.txt: Pipeline logs Data\_processed all.txt: Pipeline logs|

outputs/

word\_count\_{YYYMMDD). parquet word\_count\_all\_{YYYMMDD}.parquet

More screenshots and logs can be added (this is advised to add more screenshots/logs).

More files can be added if required.

give the other modifications,codefiles and instructions

below are the code files up to now.

cfg.yaml

input\_path: "data/test.jsonl"

src/process\_data.py

import os

import logging

import glob

import shutil

from datetime import datetime

from typing import Dict, Any

from pyspark.sql import functions as F

from pyspark.sql import DataFrame

from pyspark.sql.types import StructType, StructField, StringType, IntegerType

from src.utils import get\_spark\_session

# Compute the absolute path to the Python executable in your virtual environment (relative to project root)

venv\_python: str = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..", "venv", "Scripts", "python.exe"))

# Explicitly force Spark worker processes to use the correct Python interpreter.

os.environ["PYSPARK\_PYTHON"] = venv\_python

os.environ["PYSPARK\_DRIVER\_PYTHON"] = venv\_python

def process\_data(config: Dict[str, Any], dataset: str, output\_dir: str) -> None:

"""

Process the dataset to count occurrences of specific target words in the description column.

The description text is cleaned by converting to lower-case and removing punctuation.

Then the cleaned text is split into words and grouped by word.

The target words (["president", "the", "asia"]) are filtered and joined with a default

DataFrame (to include words with 0 count) using aliases to avoid ambiguous column names.

The resulting table (with columns "word" and "count") is saved as a single Parquet file

named "word\_count\_{YYYYMMDD}.parquet" in the specified output directory.

Args:

config (Dict[str, Any]): Configuration loaded from YAML.

dataset (str): Dataset identifier (e.g., 'news').

output\_dir (str): Relative directory to store the output Parquet file.

"""

logging.info("Starting process\_data")

# Create Spark session.

spark = get\_spark\_session("ProcessData")

# Read dataset from JSONL file.

input\_path: str = config.get("input\_path", "data/test.jsonl")

logging.info(f"Reading dataset from {input\_path}")

df: DataFrame = spark.read.json(input\_path)

# Clean the description text:

# 1. Convert to lower-case.

# 2. Remove punctuation (keeping whitespace).

cleaned\_df: DataFrame = df.select(

F.regexp\_replace(F.lower(F.col("description")), r'[^\w\s]', '').alias("cleaned")

)

# Split the cleaned description into words and filter out empty strings.

words\_df: DataFrame = cleaned\_df.select(

F.explode(F.split(F.col("cleaned"), "\\s+")).alias("word")

).filter(F.col("word") != "")

# Define target words (all in lower-case).

target\_words = ["president", "the", "asia"]

# Group by word and count occurrences.

grouped\_df: DataFrame = words\_df.groupBy("word").agg(F.count("\*").alias("count"))

# Filter to only the target words and alias as "t".

target\_counts\_df: DataFrame = grouped\_df.filter(F.col("word").isin(target\_words)).alias("t")

# Create a default DataFrame for target words with count 0 and alias as "d".

target\_schema = StructType([

StructField("word", StringType(), False),

StructField("count", IntegerType(), True)

])

default\_data = [(w, 0) for w in target\_words]

default\_df: DataFrame = spark.createDataFrame(default\_data, schema=target\_schema).alias("d")

# Left join the default DataFrame with the computed counts using "word" as key.

# Use t.count if available, otherwise the default value from d.count.

final\_df: DataFrame = default\_df.join(target\_counts\_df, on="word", how="left") \

.select(F.col("word"), F.coalesce(F.col("t.count"), F.col("d.count")).alias("count"))

# Coalesce to one partition so that only one output file is produced.

final\_df = final\_df.coalesce(1)

# Define temporary output folder and final output file path.

current\_date: str = datetime.now().strftime("%Y%m%d")

temp\_output\_path: str = os.path.join(output\_dir, f"temp\_word\_count\_{current\_date}")

final\_file\_path: str = os.path.join(output\_dir, f"word\_count\_{current\_date}.parquet")

logging.info(f"Writing temporary Parquet output to {temp\_output\_path}")

final\_df.write.mode("overwrite").parquet(temp\_output\_path)

# Find the single part file in the temporary directory.

part\_files = glob.glob(os.path.join(temp\_output\_path, "part-\*"))

if not part\_files:

logging.error("No part file found in temporary output directory.")

spark.stop()

return

part\_file: str = part\_files[0]

logging.info(f"Found part file: {part\_file}")

# Move the part file to the final output location.

logging.info(f"Moving file to final output: {final\_file\_path}")

shutil.move(part\_file, final\_file\_path)

# Remove the temporary directory.

shutil.rmtree(temp\_output\_path)

logging.info(f"Removed temporary directory: {temp\_output\_path}")

# Read back the final file for verification.

logging.info(f"Reading back the final Parquet file from {final\_file\_path}")

read\_back\_df: DataFrame = spark.read.parquet(final\_file\_path)

read\_back\_df.show(truncate=False)

logging.info("process\_data completed successfully")

spark.stop()

if \_\_name\_\_ == "\_\_main\_\_":

pass

src/process\_data\_all.py

import logging

import os

import glob

import shutil

from datetime import datetime

from typing import Dict, Any

from pyspark.sql import functions as F

from pyspark.sql import DataFrame

from src.utils import get\_spark\_session

def process\_data\_all(config: Dict[str, Any], dataset: str, output\_dir: str) -> None:

"""

Process the dataset to count occurrences of all unique words in the description column.

The resulting table has two columns: "word" and "count". The description is tokenized

by splitting on whitespace, converting tokens to lower-case, and then removing punctuation.

The resulting table is saved as a single Parquet file named "word\_count\_all\_{YYYYMMDD}.parquet"

in the specified output directory.

Steps:

1. Read the dataset.

2. Tokenize the 'description' column into words.

3. Convert tokens to lower-case.

4. Remove punctuation from each word.

5. Filter out empty strings.

6. Group by word and count occurrences.

7. Coalesce to a single partition and write to a temporary directory.

8. Move the single part file to the final filename.

9. Remove the temporary directory.

10. Read the final file and display its content.

Args:

config (Dict[str, Any]): Configuration loaded from YAML.

dataset (str): Dataset identifier (e.g., 'news').

output\_dir (str): Relative directory to store the output Parquet file.

"""

logging.info("Starting process\_data\_all")

# Create Spark session.

spark = get\_spark\_session("ProcessDataAll")

# Read dataset.

input\_path: str = config.get("input\_path", "data/test.jsonl")

logging.info(f"Reading dataset from {input\_path}")

df: DataFrame = spark.read.json(input\_path)

# Tokenize the 'description' column by splitting on whitespace and converting to lower-case.

# Then remove punctuation (retain only word characters: letters, digits, and underscore).

words\_df: DataFrame = df.select(F.explode(F.split(F.col("description"), "\\s+")).alias("raw\_word"))

words\_df = words\_df.select(F.lower(F.col("raw\_word")).alias("word"))

words\_df = words\_df.select(F.regexp\_replace(F.col("word"), r'[^\w]', '').alias("word"))

# Filter out any empty strings.

words\_df = words\_df.filter(F.col("word") != "")

# Group by word and count occurrences.

count\_df: DataFrame = words\_df.groupBy("word").agg(F.count("\*").alias("count"))

# Coalesce to a single partition so that only one output file is produced.

count\_df = count\_df.coalesce(1)

# Define temporary output folder and final output file path.

current\_date: str = datetime.now().strftime("%Y%m%d")

temp\_output\_path: str = os.path.join(output\_dir, f"temp\_word\_count\_all\_{current\_date}")

final\_file\_path: str = os.path.join(output\_dir, f"word\_count\_all\_{current\_date}.parquet")

logging.info(f"Writing temporary Parquet output to {temp\_output\_path}")

count\_df.write.mode("overwrite").parquet(temp\_output\_path)

# Find the single part file in the temporary directory.

part\_files = glob.glob(os.path.join(temp\_output\_path, "part-\*"))

if not part\_files:

logging.error("No part file found in temporary output directory.")

spark.stop()

return

part\_file: str = part\_files[0]

logging.info(f"Found part file: {part\_file}")

# Move the part file to the final output path with the desired filename.

logging.info(f"Moving file to final output: {final\_file\_path}")

shutil.move(part\_file, final\_file\_path)

# Remove the temporary directory (which contains metadata files).

shutil.rmtree(temp\_output\_path)

logging.info(f"Removed temporary directory: {temp\_output\_path}")

# Read back the final file and show its content for verification.

logging.info(f"Reading back the final Parquet file from {final\_file\_path}")

read\_back\_df: DataFrame = spark.read.parquet(final\_file\_path)

read\_back\_df.show(truncate=False)

logging.info("process\_data\_all completed successfully")

spark.stop()

if \_\_name\_\_ == "\_\_main\_\_":

pass

src/run.py

#!/usr/bin/env python

import os

import sys

# If the script is run directly (instead of via -m), add the project root to sys.path

if \_\_name\_\_ == "\_\_main\_\_" and \_\_package\_\_ is None:

# Insert the project root (parent of 'src') into sys.path

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..")))

\_\_package\_\_ = "src"

import logging

import yaml

from typing import Dict, Any, List

from src.process\_data import process\_data

from src.process\_data\_all import process\_data\_all

def parse\_args(args: List[str]) -> Dict[str, str]:

"""

Parse command-line arguments.

Args:

args (List[str]): List of command-line arguments.

Returns:

Dict[str, str]: Dictionary mapping argument names to their values.

"""

if len(args) % 2 != 0:

raise ValueError("Invalid number of arguments. Arguments must be provided in pairs.")

return {args[i].strip('-'): args[i+1] for i in range(0, len(args), 2)}

def main() -> None:

"""

Main function to parse arguments, load configuration, and invoke the appropriate data processing function.

"""

logging.basicConfig(level=logging.INFO, format='%(asctime)s %(levelname)s %(message)s')

if len(sys.argv) < 5:

print("Usage: python src/run.py <command> -cfg <config\_file> -dataset <dataset> -dirout <output\_dir>")

sys.exit(1)

command: str = sys.argv[1]

try:

args: Dict[str, str] = parse\_args(sys.argv[2:])

except ValueError as e:

logging.error(e)

sys.exit(1)

config\_file: str = args.get("cfg")

dataset: str = args.get("dataset")

output\_dir: str = args.get("dirout")

with open(config\_file, "r") as f:

config: Dict[str, Any] = yaml.safe\_load(f)

if command == "process\_data":

process\_data(config, dataset, output\_dir)

elif command == "process\_data\_all":

process\_data\_all(config, dataset, output\_dir)

else:

logging.error("Invalid command. Use 'process\_data' or 'process\_data\_all'.")

sys.exit(1)

if \_\_name\_\_ == "\_\_main\_\_":

main()

src/utils.py

import logging

import os

from pyspark.sql import SparkSession

from typing import Any

def get\_spark\_session(app\_name: str) -> SparkSession:

"""

Create and return a SparkSession with the given application name.

This session is configured so that both the driver and worker processes use

the specified Python executable.

Args:

app\_name (str): The name of the Spark application.

Returns:

SparkSession: An active SparkSession.

"""

python\_path: str = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..", "venv", "Scripts", "python.exe"))

logging.info(f"Initializing Spark session with app name: {app\_name}")

spark: SparkSession = SparkSession.builder.appName(app\_name) \

.config("spark.pyspark.python", python\_path) \

.config("spark.pyspark.driver.python", python\_path) \

.getOrCreate()

return spark

tests/test\_processing.py

import os

import sys

from datetime import datetime

# Ensure the project root is in sys.path so that "src" can be imported.

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..")))

import pytest

from pyspark.sql import SparkSession

from pyspark.sql.types import StringType, IntegerType, LongType

from pyspark.sql import functions as F

from src.process\_data import process\_data

from src.process\_data\_all import process\_data\_all

from src.utils import get\_spark\_session

def get\_output\_file(output\_dir: str, prefix: str) -> str:

"""

Returns the expected output file path given the output directory and a prefix.

"""

current\_date = datetime.now().strftime("%Y%m%d")

return os.path.join(output\_dir, f"{prefix}\_{current\_date}.parquet")

def get\_new\_spark\_session(app\_name: str = "TestRead") -> SparkSession:

"""

Creates a new Spark session for reading output.

"""

return get\_spark\_session(app\_name)

@pytest.fixture

def config() -> dict:

"""

Fixture returning a configuration dictionary.

"""

return {"input\_path": "data/test.jsonl"}

@pytest.fixture

def output\_dir(tmp\_path) -> str:

"""

Fixture creating a temporary output directory and returning its path as a string.

"""

out\_dir = tmp\_path / "output"

out\_dir.mkdir(parents=True, exist\_ok=True)

return str(out\_dir)

def test\_process\_data\_file\_exists(output\_dir: str, config: dict) -> None:

"""

Test that process\_data produces a file with the expected name.

"""

process\_data(config, "news", output\_dir)

output\_file = get\_output\_file(output\_dir, "word\_count")

assert os.path.exists(output\_file), "Output Parquet file for specific words does not exist"

def test\_process\_data\_all\_file\_exists(output\_dir: str, config: dict) -> None:

"""

Test that process\_data\_all produces a file with the expected name.

"""

process\_data\_all(config, "news", output\_dir)

output\_file = get\_output\_file(output\_dir, "word\_count\_all")

assert os.path.exists(output\_file), "Output Parquet file for all words does not exist"

def test\_specific\_schema(output\_dir: str, config: dict) -> None:

"""

Run process\_data and verify that the output Parquet file has the expected schema.

"""

process\_data(config, "news", output\_dir)

output\_file = get\_output\_file(output\_dir, "word\_count")

spark = get\_new\_spark\_session("TestRead\_Schema")

df = spark.read.parquet(output\_file)

schema = df.schema

field\_names = schema.fieldNames()

assert "word" in field\_names, "Schema missing 'word' column"

assert "count" in field\_names, "Schema missing 'count' column"

assert isinstance(schema["word"].dataType, StringType), "'word' column is not StringType"

# Accept either IntegerType or LongType for count.

assert isinstance(schema["count"].dataType, (IntegerType, LongType)), "'count' column is not numeric (expected IntegerType or LongType)"

spark.stop()

def test\_specific\_target\_values(output\_dir: str, config: dict) -> None:

"""

Run process\_data and check that the target words appear in the output.

"""

process\_data(config, "news", output\_dir)

output\_file = get\_output\_file(output\_dir, "word\_count")

spark = get\_new\_spark\_session("TestRead\_Targets")

df = spark.read.parquet(output\_file)

words = [row["word"] for row in df.collect()]

for target in ["president", "the", "asia"]:

assert target in words, f"Target word {target} is missing in output"

spark.stop()

def test\_specific\_and\_all\_consistency(output\_dir: str, config: dict) -> None:

"""

Run both process\_data and process\_data\_all, then verify that for each target word

the count is the same between the two outputs.

"""

process\_data(config, "news", output\_dir)

process\_data\_all(config, "news", output\_dir)

specific\_file = get\_output\_file(output\_dir, "word\_count")

all\_file = get\_output\_file(output\_dir, "word\_count\_all")

spark = get\_new\_spark\_session("TestRead\_Consistency")

df\_specific = spark.read.parquet(specific\_file)

df\_all = spark.read.parquet(all\_file)

for target in ["president", "the", "asia"]:

specific\_count\_rows = df\_specific.filter(F.col("word") == target).select("count").collect()

all\_count\_rows = df\_all.filter(F.col("word") == target).select("count").collect()

specific\_count = specific\_count\_rows[0]["count"] if specific\_count\_rows else 0

all\_count = all\_count\_rows[0]["count"] if all\_count\_rows else 0

assert specific\_count == all\_count, f"Count for {target} mismatch: specific({specific\_count}) != all({all\_count})"

spark.stop()

I was asked to submit below files regarding this assignment project.

Zip file should contain folders with this organization

code/

github\_build\_action.yml Dockerfile.Dockerfile

script/run.sh : Bash Script to start to generate the 2 files.

Config files in code/config:

config.yaml : config file in yaml format in config/ sub-folder Source code in code/src/

screenshots/ docker\_build.png pip\_freeze.png dataprocessed.png data\_processed\_all.png

logs/

Docker\_build.txt : Docker build log pip\_list.txt : pip list inside the docker Data\_processed.txt: Pipeline logs Data\_processed all.txt: Pipeline logs|

outputs/

word\_count\_{YYYMMDD). parquet word\_count\_all\_{YYYMMDD}.parquet

More screenshots and logs can be added (this is advised to add more screenshots/logs).

More files can be added if required.

give the other modifications,codefiles and instructions.

when defining paths use relative paths always.

run.sh.

should contain below commands

python src/run.py process\_data -cfg config/cfg.yaml -dataset news -dirout “ztmp/data/”

python src/run.py process\_data\_all -cfg config/cfg.yaml -dataset news -dirout “ztmp/data/”

give all complete code files.

use the below project structure.

don't use a folder named "code"

UZABASE/ # Root folder (D:\UZABASE)

├── config/

│ └── cfg.yaml # YAML configuration file (e.g. input file path)

├── data/

│ └── test.jsonl # Input dataset file (your provided test.jsonl)

├── src/

│ ├── \_\_init\_\_.py

│ ├── run.py # Entry point; parses command-line args and calls job

│ ├── process\_data.py # Contains process\_data() for counting specific words

│ ├── process\_data\_all.py# Contains process\_data\_all() for counting all unique words

│ └── utils.py # Contains helper functions (e.g. to get Spark session)

├── tests/

│ ├── \_\_init\_\_.py

│ └── test\_data\_processing.py # Basic tests for the processing functions

├── requirements.txt # Python dependencies (pyspark, pyyaml, pytest, etc.)

└── README.md # Ins

give github\_build\_action.yml and dDockerfile.Dockerfile.

below logs should be generate in logs folder.

give the modifications in process\_data.py,process\_data\_all.py and etc to generate logs.

Docker\_build.txt : Docker build log

pip\_list.txt : pip list inside the docker

Data\_processed.txt: Pipeline logs

Data\_processed all.txt: Pipeline logs|

script/run.sh : Bash Script to start to generate the 2 output parquet files.