**AWS Glue Homework**

This report summarizes the AWS Glue data analysis tasks completed using New York City taxi databases from January to March. It includes step-by-step processes, code snippets, analytical insights, and reflections on challenges encountered.

**Part 1. AWS Glue and Cloudformation**

**Step-by-step Process and Code Snippets**

1. **Download the dataset**

Visit the NYC taxi database and download the Yellow taxi trip record data 2025 from January to March. The link: <https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page>

The data schema can be found in Yellow Trips Data Dictionary.

1. **AWS Glue CloudFormation Template**

Build up a file name HW1-AWS\_Glue.yaml with S3 bucket, Glue Database, Table, Crawler, with IAM roles and permissions. For the crawler, it is allowed to acquire data from the S3 bucket and detect the schema accordingly.

1. **Deploy Steps**
2. Upload the created HW1-AWS\_Glue.yaml template to AWS CloudFormation to create the resources.
3. Manually upload all the downloaded data to the created S3 bucket.
4. Start the crawler manually in the AWS Glue Console (AWS Glue – Data Catalog - Crawlers - Run) to acquire data schema. Once the crawler completes the task, go to the database table section (AWS Glue – Data Catalog Databases - Tables) in AWS Glue to view the results.
5. It can be concluded that the NYC taxi database has twenty columns in total with different data types.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1. Screenshot shows the crawler runs successfully

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2. The schema of the NYC taxi database

1. **Data Modification**
2. Modify the NYC taxi database using DataBrew.
3. Create a DataBrew *dataset* pointing to the raw Yellow Taxi Parquet files in S3. In this task, the January dataset is chosen as an example.
4. Create a project and recipe, which drops the Airport\_fee columns.
5. Upload the modified dataset to the original S3 bucket and execute the crawler to observe the changes in the schema.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3. Data modification using DataBrew

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4. Screenshot shows the crawler runs successfully

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5. The altered schema of the NYC taxi database

Figure 5 shows that the Glue crawler detected a schema change. Because only the January partition was modified, the table now mixes two schemas: January without Airport\_fee and February and March with it.

1. **Triggers**
2. Change the CloudFormation template used to add a trigger configuration. The trigger will be applied to run the crawler on a five-minute schedule. The configuration code is shown below.

  GlueTrigger:

    Type: AWS::Glue::Trigger

    Properties:

      Name: hw16-crawler-trigger

      Description: Trigger to run crawler on schedule

      Type: SCHEDULED

      Schedule: 'cron(0/5 \* \* \* ? \*)'

      StartOnCreation: true

      Actions:

      - CrawlerName: !Ref CrawlerName

      - JobName: !Ref GlueJobName

1. Upload the modified version of the CloudFormation. Then, go to the crawler section in the AWS Glue Catalog to start the crawler.
2. The crawler will be triggered in a five-minute schedule. The result will be shown in Figure 6.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6. The crawler is triggered in a five-minute schedule.

**Part 2. AWS Glue and Cloudformation**

**Step-by-step Process and Code Snippets**

1. **AWS Glue CloudFormation Template**

Build up a file name HW2-AWS\_Glue.yaml with S3 bucket, Glue Database, GlueJob, Crawler, Table, Security groups with IAM roles and permissions. For the Glue job, it is allowed to have access to S3 bucket, pass IAM role to the Glue service, and access the RDS information. For the crawler, it is allowed to acquire data from the S3 bucket and detect the schema accordingly.

Additionally, in this part, the connection between the RDS and the Glue job is set up. Moreover, a glue job script editor should also be set up to ensure that the glue script can be implemented successfully in AWS.

*# create connection between rds and glue*

  GlueConnection:

    Type: AWS::Glue::Connection

    Properties:

      CatalogId: !Ref AWS::AccountId

      ConnectionInput:

        Name: nyc-taxi-rds-conn

        ConnectionType: JDBC

        Description: JDBC to RDS for NYC Taxi

        ConnectionProperties:

          JDBC\_CONNECTION\_URL: !Sub 'jdbc:mysql://${MyRDS.Endpoint.Address}:3306/${DBName}'

          USERNAME: !Ref DBUsername

          PASSWORD: !Ref DBPassword

        PhysicalConnectionRequirements:

          SecurityGroupIdList:

          - !Ref GlueJobSecurityGroup

          SubnetId: !Select [ 0, !Ref SubnetIds ]

*# create a glue job script editor*

  GlueJob:

    Type: AWS::Glue::Job

    Properties:

      Name: !Ref GlueJobName

      Role: !GetAtt GlueJobRole.Arn

      GlueVersion: "5.0"

      WorkerType: G.1X

      NumberOfWorkers: 2

      Timeout: 2880

      MaxRetries: 1

      Command:

        Name: glueetl

        ScriptLocation: !Sub 's3://${DataBucketName}/scripts/${GlueJobName}.py'

      DefaultArguments:

        '--job-language': python

        '--enable-metrics': 'true'

        '--enable-continuous-cloudwatch-log': 'true'

        '--enable-spark-ui': 'true'

        '--spark-event-logs-path': !Sub 's3://${DataBucketName}/spark-events/'

        '--TempDir': !Sub 's3://${DataBucketName}/temp/'

        '--job-bookmark-option': 'job-bookmark-enable'

        '--enable-glue-datacatalog': 'true'

        '--database\_name': !Ref DatabaseName

        '--input\_path': !Sub 's3://${DataBucketName}/data/'

        '--zones\_path': !Sub 's3://${ZonesBucketName}/Zones/taxi\_zones\_centroids.parquet'

        '--output\_path': !Sub 's3://${DataBucketName}/output/'

        '--connection\_name': 'nyc-taxi-rds-conn'

        '--db\_name': !Ref DBName

        '--db\_user': !Ref DBUsername

        '--db\_password': !Ref DBPassword

        '--db\_jdbc\_url': !Sub 'jdbc:mysql://${MyRDS.Endpoint.Address}:3306/${DBName}'

        '--table\_name': !Ref GlueTableName

        '--additional-python-modules': 'pydeequ==1.0.1,requests==2.32.3'

      Connections:

        Connections:

        - !Ref GlueConnection

      ExecutionProperty:

        MaxConcurrentRuns: 1

In this template, the job bookmark option is enabled (see the ‘—job-bookmark-option’ argument), which means that when processing the data, the Glue job will skip the previous datasets and only ingest the newly-updated datasets.

1. **Deploy Steps**
2. Upload the created HW2-AWS\_Glue.yaml template to AWS CloudFormation to create the resources.
3. Manually upload all the NYC taxi January data to the created S3 bucket.
4. Start the crawler manually in the AWS Glue Console (AWS Glue – Data Catalog - Crawlers - Run) to acquire data schema. Once the crawler completes the task, go to the database table section (AWS Glue – Data Catalog Databases - Tables) in AWS Glue to view the results.
5. Start Glue Job script editing (AWS Glue – Data Integration and ETL – ETL jobs) to process the uploaded dataset. In this scenario, the dataset is processed using Glue Job Notebook.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Data Transformation Steps**
2. Before the data transformation, Glue and Spark should be initialized. The following code can do the task.



In this part, we also enable the Spark’s shuffle, this is an important configuration for workload partitioning.

1. The next step is to read data from catalog; this can be realized using the following code:

A screenshot of a computer program

AI-generated content may be incorrect.

1. Then, set up partition. In this analysis, we will partition the data using the pickup date, and write to the assigned S3 bucket partitioned by the pickup date. The following code will do the work.

A computer code with text

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

1. Begin the data transformation.
   1. Time-based Filtering:

The following code will do time-based filtering which the trips occurred during the weekend with the output result.

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Bucketing:

The following code will create the buckets based on the trip distance and count the trip respectively.

A screenshot of a computer code

AI-generated content may be incorrect.

A number in a row

AI-generated content may be incorrect.

* 1. Time-series Analysis:

The following code will calculate the total fare and total trip count per hour.

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Geospatial Analysis:

The following code is to calculate the Haversine Distance. In this analysis, we should first get the longitude and latitude of the pickup and drop-off location. We should go to the NYC taxi dataset website to download the taxi zones data file, which contains the location ID and zone information. Then, in taxi.py, compute a centroid for each location ID in the source projected CRS (EPSG:2263 for NYC), then reproject to WGS84 and extract longitude and latitude. After that, export a table with: LocationID, longitude, latitude in parquet format. Then, upload the transformed data to the create S3 bucket and start the data analysis. The data analysis code with corresponding results are shown in the following.

A screenshot of a computer

AI-generated content may be incorrect.

A table of numbers

AI-generated content may be incorrect.

* 1. Advanced Feature Engineering:

The following code will calculate the time duration and the speed.

A screenshot of a computer

AI-generated content may be incorrect.

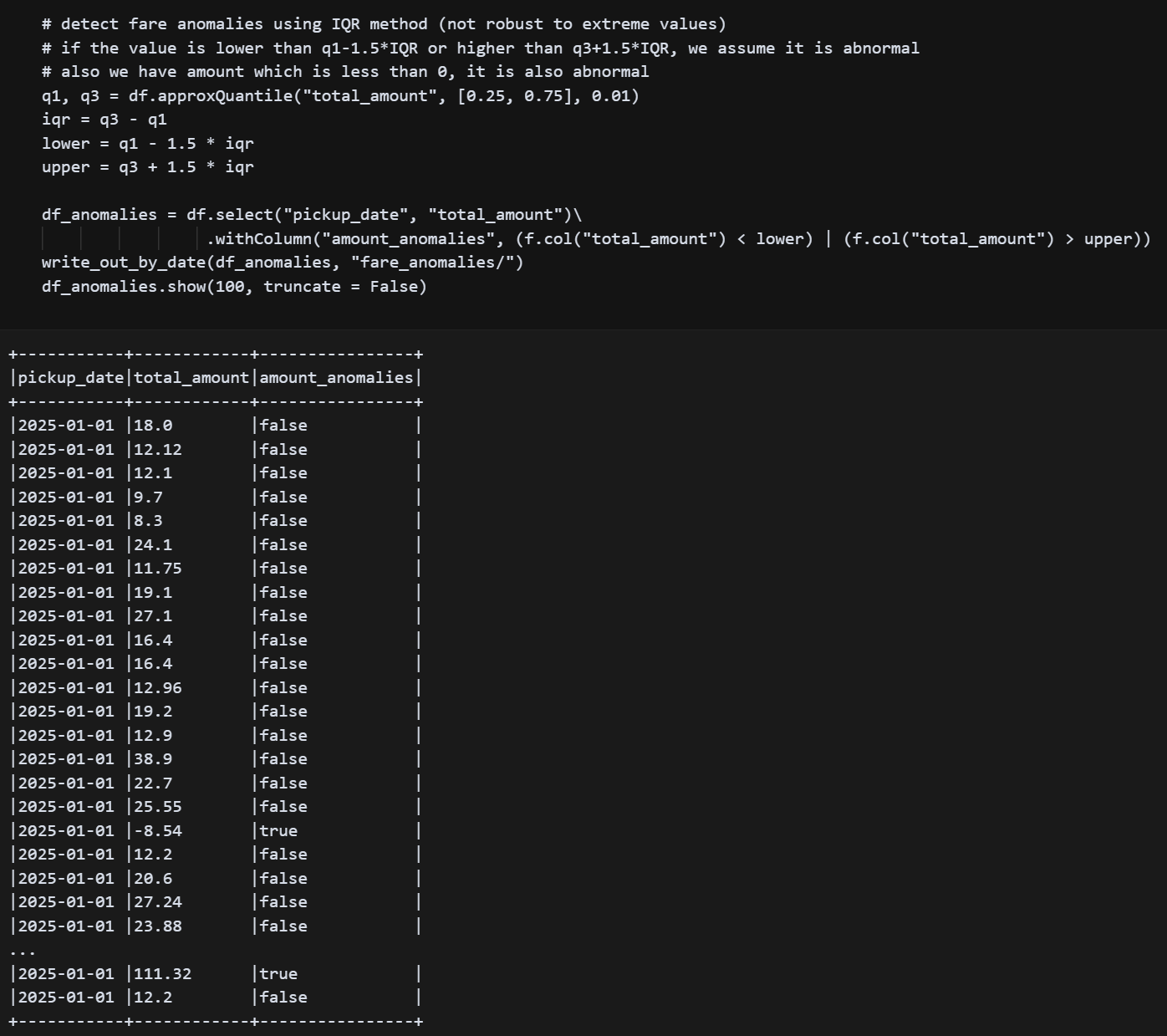
* 1. Advance Aggregation and Analysis:

The following code will identify the peak hour and the fare anomalies. To identify the peak hour, we can extract the number of pickups happening in each hour and identify the hour with the maximum pickup number. To detect the fare anomalies in fare amount, we can apply the IQR method, which is to find fare values lower than q1-1.5 \* IQR or higher than q3+1.5 \* IQR, where IQR = q3 – q1, q3 is the 75th percentile, q1 is the 25th percentile.

A screenshot of a computer

AI-generated content may be incorrect.

Based on the results, it can be concluded that most of the pickups happen between 17-19, implying that rush hour in New York is from 17-19.



1. In this step, we will upload the February and March NYC taxi dataset to analyze the results. After uploading the dataset, we will run the job to see the results. As it enables bookmark, it will process the newly updated dataset and add the new results to the previous results.

A screenshot of a computer

AI-generated content may be incorrect.

The results of the six tasks are shown below:

This is the result of the time-based filtering.

A screenshot of a computer

AI-generated content may be incorrect.

This is the result of the bucketing.

A screenshot of a computer

AI-generated content may be incorrect.

This is the result of the time-series analysis.

A screenshot of a computer

AI-generated content may be incorrect.

This is the result of the geospatial analysis.

A screen shot of a computer code

AI-generated content may be incorrect.

This is the result of the advanced feature engineering.

A screenshot of a computer program

AI-generated content may be incorrect.

This is the result of advanced aggregation and analysis.

A table of numbers

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

1. Partitioning

When the transformed data is uploaded to the S3 bucket, it will be partitioned according to the pickup date. The following are the results.

A screenshot of a computer

AI-generated content may be incorrect.

**Part 3. Debugging AWS Glue ETL Scripts**

**Step-by-step Process and Code Snippets**

1. **AWS Glue CloudFormation Template**

In this assignment, we will correct the ETL script using the same yaml template in Part 2. In this assignment, the script will be broken down into different part in ETL Notebook to do the error detection, explanation and correction.

1. **ETL Job Errors and Corrections**
   1. Glue Initialization:

In the Glue Initialization part, simply using sc = SparkContext will create multiple SparkContext. To avoid such issue, it is better to use sc = SparkContext.getOrCreate(). The correction is shown below:

A computer screen shot of a computer code

AI-generated content may be incorrect.

* 1. Filtering

In the filtering part, there are two significant issues. The first issue is that it compares trip distance with string “2”, it will cause data type mismatch. The second issue is that nulls or non-numeric strings in trip distance column can cause errors when using Filter.apply/to DF. To correct the errors, we first convert the dataset to Spark Dataframe, and change trip distance to numeric type to do the filtering. The corrected code snippet is shown below.

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Data analysis

In this part, there is one significant error that it tries to group a non-existent column in the dataset. To correct this error, we should replace “passenger” to a existing column in the dataset. The correction code is shown below.

A computer code with text

AI-generated content may be incorrect.

* 1. UDF errors

In this part, there are some errors in the code. The first error is the udf usage. UDF is a function in pyspark.sql, so simply using udf cannot correctly call it. The second error is that the return type should be an instance, so we should not use StringType. The third error is that withColumn requires a “list expression” rather than a string, so columns should not be called using data\_frame[‘trip\_distance’]. The corrected code is shown below.

A screenshot of a computer program

AI-generated content may be incorrect.

* 1. Options and Format

In this part, the error comes from the incompatible data types. For CSV format, it is plain text without any pattern. So, the timestamp NTZ must be converted to the supporting format (like string or timestamp) to be stored in CSV. The corrected code is shown below.

A screenshot of a computer program

AI-generated content may be incorrect.

1. **Script Deployment**

Run the correct ETL job script, the results will be uploaded and stored in the created S3 bucket. The result is shown below.

A screen shot of a computer

AI-generated content may be incorrect.