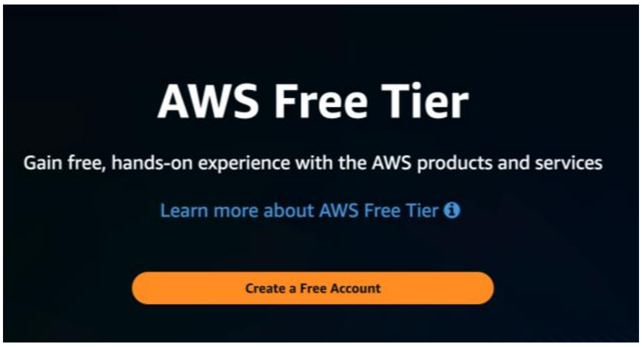
Data Engineering Core Skills – Exercises

1. Cracking the Case

1.1. Configuration

- **AWS Account Setup –** You'll need an AWS Account for this tutorial. To create an account, click HERE:
  - Click Free Account

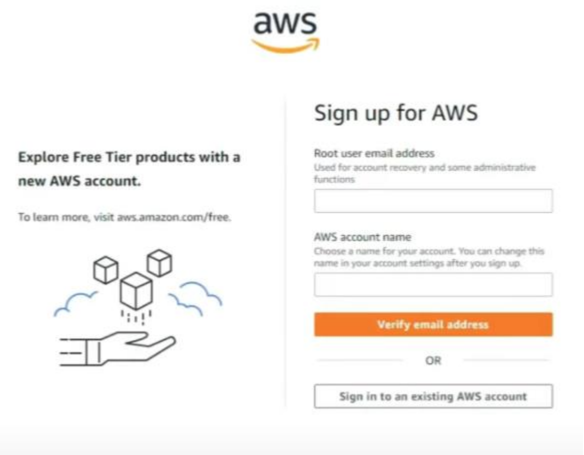


AWS Free Tier

Gain free, hands-on experience with the AWS products and services

Learn more about AWS Free Tier

  - Enter your email as root user
  - Enter an account name and credit card details (you will not be charged)



Explore Free Tier products with a new AWS account.

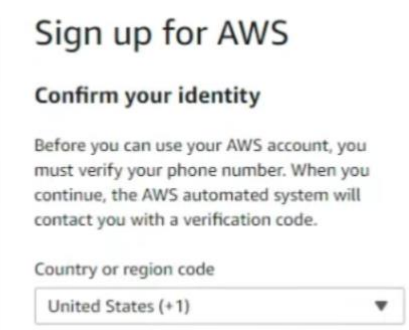
To learn more, visit aws.amazon.com/free.

Sign up for AWS

Root user email address
Used for account recovery and some administrative functions

AWS account name
Choose a name for your account. You can change this name in your account settings after you sign up.

  - Enter a region – close to your country



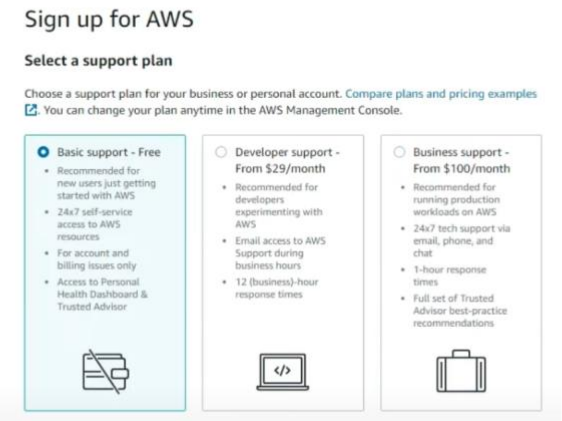
Sign up for AWS

Confirm your identity

Before you can use your AWS account, you must verify your phone number. When you continue, the AWS automated system will contact you with a verification code.

Country or region code
United States (+1)

  - Select support plan - Free



Sign up for AWS

Select a support plan

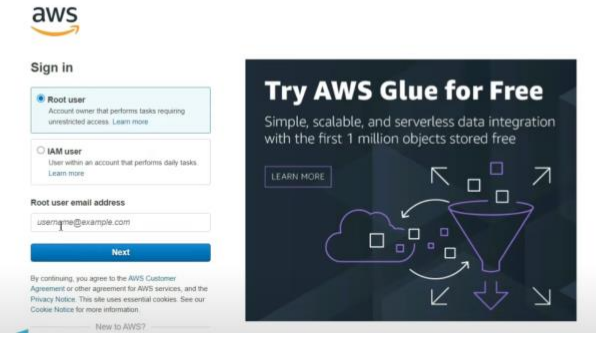
Choose a support plan for your business or personal account. Compare plans and pricing examples. You can change your plan anytime in the AWS Management Console.

● Basic support - Free
- Recommended for new users just getting started with AWS
- 24x7 self-service access to AWS resources
- For account and billing issues only
- Access to Personal Health Dashboard & Trusted Advisor

○ Developer support - From $29/month
- Recommended for developers experimenting with AWS
- Email access to AWS Support during business hours
- 12 (business)-hour response times

○ Business support - From $100/month
- Recommended for running production workloads on AWS
- 24x7 tech support via email, phone, and chat
- 1-hour response times
- Full set of Trusted Advisor best-practice recommendations

- Login – as root user



Sign in

● Root user
Account owner that performs tasks requiring unrestricted access. Learn more

○ IAM user
User within an account that performs daily tasks. Learn more

Root user email address

username@example.com

Next

By continuing, you agree to the AWS Customer Agreement or other agreement for AWS services, and the Privacy Notice. This site uses essential cookies. See our Cookie Notice for more information.

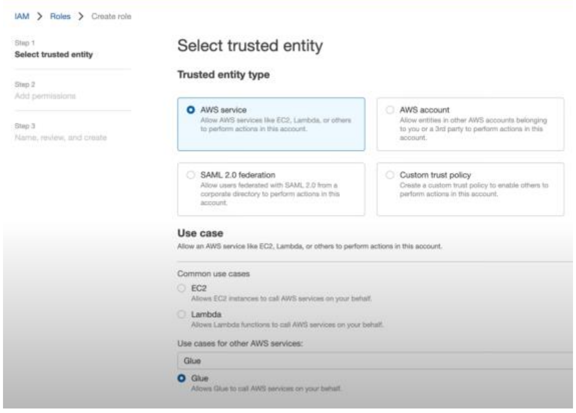
New to AWS?

Try AWS Glue for Free

Simple, scalable, and serverless data integration with the first 1 million objects stored free

LEARN MORE

- **IAM Role Setup for AWS Glue**
  - Navigate to the IAM console
  - Click on **Roles** and **Create Role**.
  - Select Glue service from the section
  - **Choose the service that will use this role**.
  - Choose Glue for *Select your use case.*



IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity

Trusted entity type

● AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

○ AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

○ SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

○ Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Common use cases

○ EC2
Allows EC2 instances to call AWS services on your behalf.

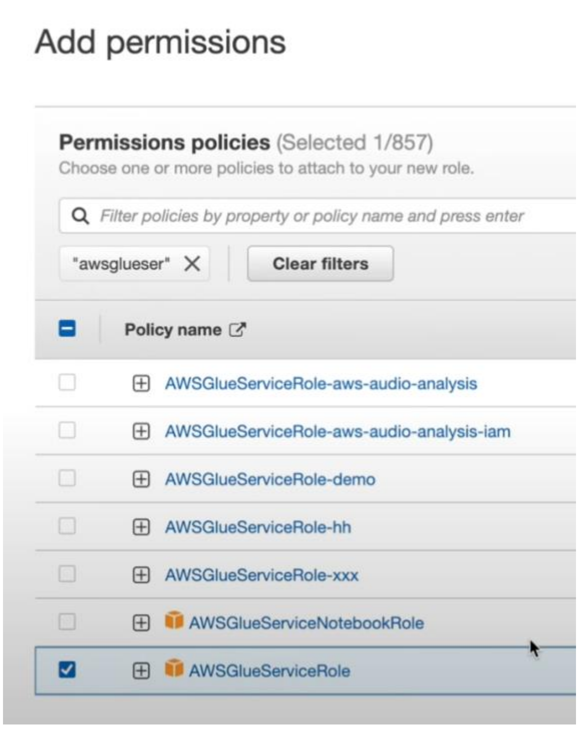
○ Lambda
Allows Lambda functions to call AWS services on your behalf.

Use cases for other AWS services:

Glue

● Glue
Allows Glue to call AWS services on your behalf.

  - Select **AWSGlueServiceRole** from **Attach Permissions Policies**.



Add permissions

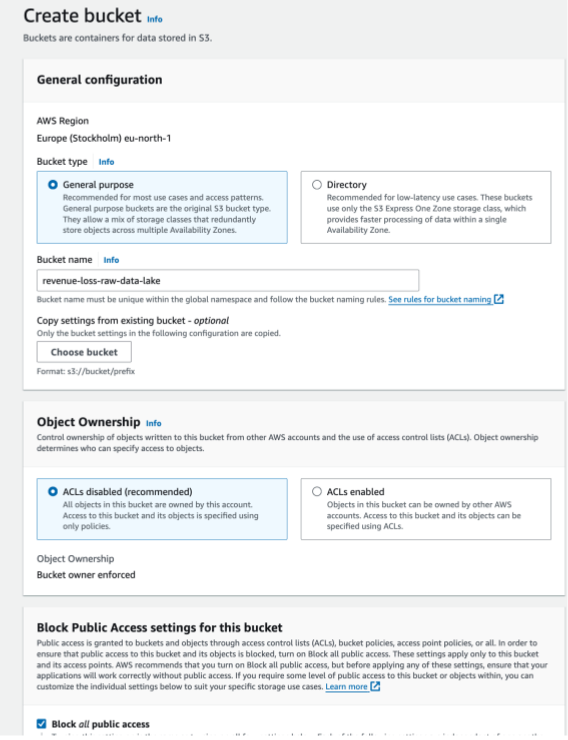
Permissions policies (Selected 1/857)
Choose one or more policies to attach to your new role.

Filter policies by property or policy name and press enter

"awsglueser" ✕ | Clear filters

■	Policy name
□	⊞ AWSGlueServiceRole-aws-audio-analysis
□	⊞ AWSGlueServiceRole-aws-audio-analysis-iam
□	⊞ AWSGlueServiceRole-demo
□	⊞ AWSGlueServiceRole-hh
□	⊞ AWSGlueServiceRole-xxx
□	⊞ AWSGlueServiceNotebookRole
☑	⊞ AWSGlueServiceRole

  - Now, write S3Full in the filter box above to add **S3FullAccess**
  - Leave the tag section blank and then create the role.
- **S3 Bucket Setup**
  - Navigate to the S3 console.
  - Click **Create bucket**. Give it a name: revenue-loss-raw-data-lake



Create bucket Info
Buckets are containers for data stored in S3.

General configuration

AWS Region
Europe (Stockholm) eu-north-1

Bucket type Info

● General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

○ Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info
revenue-loss-raw-data-lake
Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*
Only the bucket settings in the following configuration are copied.
Choose bucket
Format: s3://bucket/prefix

Object Ownership Info
Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

● ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

○ ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership
Bucket owner enforced

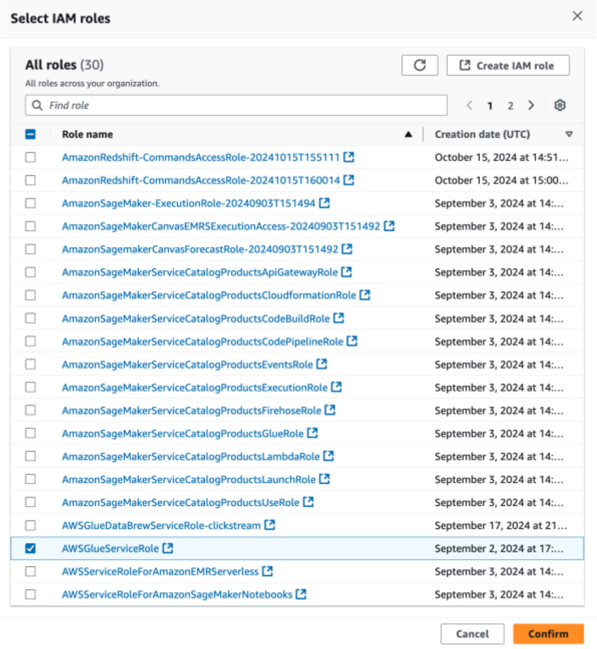
Block Public Access settings for this bucket
Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. Learn more

☑ **Block *all* public access**

  - Select your region
  - Finish creating bucket.

- **AWS Glue Setup**

- Navigate to the Glue console
- In getting started, **Click on Setup roles and users from**
- Click on 'Choose Roles' and Confirm.



Select IAM roles

All roles (30)

All roles across your organization.

Find role

☐	Role name	Creation date (UTC)
☐	AmazonRedshift-CommandsAccessRole-20241015T155111	October 15, 2024 at 14:51...
☐	AmazonRedshift-CommandsAccessRole-20241015T160014	October 15, 2024 at 15:00...
☐	AmazonSageMaker-ExecutionRole-20240903T151494	September 3, 2024 at 14:...
☐	AmazonSageMakerCanvasEMRSExecutionAccess-20240903T151492	September 3, 2024 at 14:...
☐	AmazonSagemakerCanvasForecastRole-20240903T151492	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsApiGatewayRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsCloudformationRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsCodeBuildRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsCodePipelineRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsEventsRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsExecutionRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsFirehoseRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsGlueRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsLambdaRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsLaunchRole	September 3, 2024 at 14:...
☐	AmazonSageMakerServiceCatalogProductsUseRole	September 3, 2024 at 14:...
☐	AWSGlueDataBrewServiceRole-clickstream	September 17, 2024 at 21...
☑	AWSGlueServiceRole	September 2, 2024 at 17:...
☐	AWSServiceRoleForAmazonEMRServerless	September 3, 2024 at 14:...
☐	AWSServiceRoleForAmazonSageMakerNotebooks	September 3, 2024 at 14:...

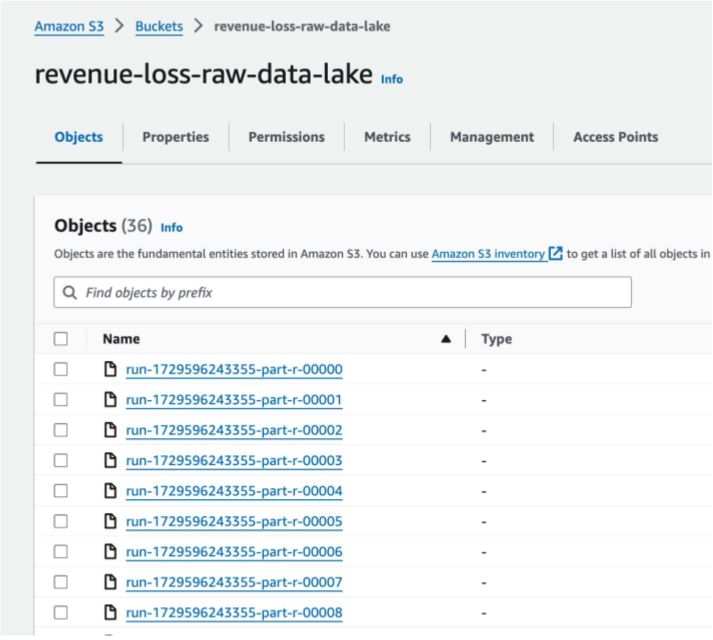
- Click on Next, Next and Apply Changes.

1.2. Create Glue Script

- Navigate to the Glue console
- Click ETL Jobs
- Click on Script editor
- Enter the following

```
import requests
from awsglue.context import GlueContext
from pyspark.context import SparkContext
from awsglue.dynamicframe import DynamicFrame  # Import the correct module

# Initialize GlueContext and SparkContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Make the GET request to the API
api_url = "https://github.com/nextgendata-academy/revenue-loss/blob/main/data/clickstream_data.csv"
response = requests.get(api_url)

# Check if the request was successful
if response.status_code == 200:

    # Read the CSV data directly from the API response
    csv_data = response.content.decode('utf-8')

    # API response is a string, split it into rows by breaking it at (\n)
    # Convert the CSV data into an RDD format that Spark understands
    rdd = sc.parallelize(csv_data.splitlines())
    # Convert RDD into Spark DataFrame, with a schema
    df = glueContext.spark_session.read.csv(rdd, header=True, inferSchema=True)

    # Convert the Spark DataFrame to a Glue DynamicFrame
    dynamic_frame = DynamicFrame.fromDF(df, glueContext)  # Corrected method usage

    # Write the data to S3 in CSV format
    glueContext.write_dynamic_frame.from_options(
        frame=dynamic_frame,
        connection_type="s3",
        connection_options={"path": "s3://revenue-loss-raw-data-lake"},
        format="csv"
    )

else:
    print(f"Failed to fetch data. Status code: {response.status_code}")
```

- Save Script and Run
- Click on “Runs” Tab to see the if the job succeeded.
- Navigate to the S3 console.
- Select the bucket to see if the data is ingested.



Amazon S3 > Buckets > revenue-loss-raw-data-lake
revenue-loss-raw-data-lake Info
Objects
Properties
Permissions
Metrics
Management
Access Points

Objects (36) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in

Find objects by prefix

Name	Type
run-1729596243355-part-r-00000	-
run-1729596243355-part-r-00001	-
run-1729596243355-part-r-00002	-
run-1729596243355-part-r-00003	-
run-1729596243355-part-r-00004	-
run-1729596243355-part-r-00005	-
run-1729596243355-part-r-00006	-
run-1729596243355-part-r-00007	-
run-1729596243355-part-r-00008	-

2. Section 2: Hands on Exercise

2.1. Create Data Brew Job for Data Cleaning

2.2. Create Glue Job for Data Cleaning

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import col, when

# Initialize Glue context
glueContext = GlueContext(SparkContext.getOrCreate())
spark = glueContext.spark_session

# Load data from S3 (assuming it's already in the Glue Data Catalog)
clickstream_data = glueContext.create_dynamic_frame.from_catalog(database="your_database",
table_name="your_table")

# Convert to DataFrame for easier manipulation
df = clickstream_data.toDF()

# Data cleaning and transformation
# Handle missing ProductID by removing rows
df_cleaned = df.filter(df.ProductID.isNotNull())

# Standardize date formats (assuming SessionDate is in multiple formats)
df_cleaned = df_cleaned.withColumn("SessionDate", date_format(col("SessionDate"), "yyyy-MM-dd"))

# Handle outliers in TimeSpent by capping values above a threshold
df_cleaned = df_cleaned.withColumn("TimeSpent", when(col("TimeSpent") > 3600,
3600).otherwise(col("TimeSpent")))

# Remove duplicate entries based on SessionID
df_cleaned = df_cleaned.dropDuplicates(["SessionID"])

# Convert DataFrame back to DynamicFrame for writing
dynamic_frame_cleaned = DynamicFrame.fromDF(df_cleaned, glueContext, "dynamic_frame_cleaned")

# Save the cleaned data back to S3
glueContext.write_dynamic_frame.from_options(
    frame = dynamic_frame_cleaned,
    connection_type = "s3",
    connection_options = {"path": "s3://your-bucket/cleaned_data/"},
    format = "parquet"
)

# End Glue job
job.commit()
```

3. Section 2: Hands on Exercise

3.1. Create Glue Job to solve mystery: Lost customers

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import col, count, when

# Initialize Glue context and job parameters
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# Input S3 path and output paths
input_s3_path = "s3://your-bucket-name/clickstream_data.csv"
output_s3_path_cleaned = "s3://your-bucket-name/cleaned_clickstream_data/"
output_s3_path_disappearing = "s3://your-bucket-name/disappearing_visitors/"
output_s3_path_checkout = "s3://your-bucket-name/checkout_dropoff/"

# Load clickstream data from S3
clickstream_df = spark.read.csv(input_s3_path, header=True, inferSchema=True)

# === 1. The Disappearing Visitors: Identify users who only visited pages without engagement ===
disappearing_visitors = clickstream_df.groupBy("customerId", "sessionId").agg(
    count(when(col("eventType") == "pageView", 1)).alias("pageViewCount"),
    count(when(col("eventType") != "pageView", 1)).alias("otherEventCount")
).filter((col("pageViewCount") > 0) & (col("otherEventCount") == 0))

# Write disappearing visitors data to S3
disappearing_visitors.write.mode("overwrite").csv(output_s3_path_disappearing)

# === 2. The Missing Product Insights: Data Cleaning ===
cleaned_clickstream = clickstream_df.fillna({'productId': -1}) \
    .fillna({'customerId': 'Unknown'}) \
    .dropna(subset=['sessionId', 'eventType'])

# Drop duplicates
cleaned_clickstream = cleaned_clickstream.dropDuplicates()

# Write cleaned data to S3
cleaned_clickstream.write.mode("overwrite").csv(output_s3_path_cleaned)

# === 3. The Checkout Drop-Off: Identify users who added items to cart but didn't checkout ===
checkout_drop_off = cleaned_clickstream.groupBy("customerId", "sessionId").agg(
    count(when(col("eventType") == "addToCart", 1)).alias("addToCartCount"),
    count(when(col("eventType") == "checkout", 1)).alias("checkoutCount")
).filter((col("addToCartCount") > 0) & (col("checkoutCount") == 0))

# Write checkout drop-off data to S3
checkout_drop_off.write.mode("overwrite").csv(output_s3_path_checkout)

# Commit job
job.commit()
```

4. Section 4: Hands on Exercise

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import col, count, when

# Initialize Glue context and job parameters
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

from pyspark.sql import functions as F

# Load the cleaned clickstream data
clickstream_df = spark.read.csv("s3://your-bucket-name/cleaned_clickstream_data/", header=True,
inferSchema=True)

# === Fact Table: Customer Sessions ===
fact_customer_sessions = clickstream_df.groupBy("customerId", "sessionId").agg(
    F.count(when(col("eventType") == "pageView", 1)).alias("pageViewCount"),
    F.count(when(col("eventType") == "view_product", 1)).alias("productViewCount"),
    F.count(when(col("eventType") == "addToCart", 1)).alias("addToCartCount"),
    F.count(when(col("eventType") == "checkout", 1)).alias("checkoutCount"),
    F.max("sessionDate").alias("sessionDate"),  # Optional to store the latest session date
    F.max("timeSpent").alias("timeSpent")  # Total time spent in session
)

# Create a sessionType column to label the session based on conditions
fact_customer_sessions = fact_customer_sessions.withColumn(
    "sessionType",
    F.when((F.col("pageViewCount") > 0) & (F.col("productViewCount") == 0), "ViewingButNoProduct")
     .when((F.col("addToCartCount") > 0) & (F.col("checkoutCount") == 0), "CheckoutDropOff")
     .when((F.col("pageViewCount") > 0) & (F.col("productViewCount") == 0) & (F.col("addToCartCount") == 0),
"Disappearing")
     .otherwise("Other")
)

# Write the fact table to S3
fact_customer_sessions.write.mode("overwrite").csv("s3://your-bucket-name/fact_customer_sessions/")

# === Dimension Table: Customer ===
dim_customer = clickstream_df.select("customerId", "customerName", "email", "age", "city").distinct()

# Write the customer dimension table to S3
dim_customer.write.mode("overwrite").csv("s3://your-bucket-name/dim_customer/")

# === Dimension Table: Product ===
dim_product = clickstream_df.select("productId").distinct()

# Optionally, join product metadata if available
# product_metadata = spark.read.csv("s3://your-bucket-name/product_metadata.csv", header=True,
inferSchema=True)
# dim_product = dim_product.join(product_metadata, "productId", "left")

# Write the product dimension table to S3
dim_product.write.mode("overwrite").csv("s3://your-bucket-name/dim_product/")


# Commit job
job.commit()
```