**Section 1**: Data Extraction from Google Ads and Google Analytics Objective: Demonstrate your ability to extract data from Google Ads and Google Analytics.

1. Google Ads Data Extraction:

Before starting this I enabled google ads api in gcp console.

Created service account.

Created a key and set it in json format

Task: Write a Python script or use a tool (e.g., Google Cloud Dataflow or Airflow) to extract campaign performance data (e.g., impressions, clicks, cost) from Google Ads.

from google.ads.google\_ads.client import GoogleAdsClient

import datetime

import csv

# Load Google Ads API client

client = GoogleAdsClient.load\_from\_storage("google-ads.yaml")

# Replace with your Google Ads Customer ID (without dashes)

CUSTOMER\_ID = "1234567890"

# Define date range (last 30 days)

end\_date = datetime.date.today()

start\_date = end\_date - datetime.timedelta(days=30)

# Define the Google Ads Query Language (GAQL) query

query = f"""

SELECT

campaign.id,

campaign.name,

metrics.impressions,

metrics.clicks,

metrics.cost\_micros

FROM campaign

WHERE segments.date BETWEEN '{start\_date}' AND '{end\_date}'

"""

def fetch\_campaign\_data():

"""Fetch campaign performance data from Google Ads API."""

client = GoogleAdsClient.load\_from\_storage("google-ads.yaml")

ga\_service = client.get\_service("GoogleAdsService")

# Execute query

response = ga\_service.search(customer\_id=CUSTOMER\_ID, query=query)

# Store results in a list

data = []

for row in response:

data.append([

row.campaign.id,

row.campaign.name,

row.metrics.impressions,

row.metrics.clicks,

row.metrics.cost\_micros / 1\_000\_000 # Convert micros to standard currency

])

return data

def save\_to\_csv(data, filename="google\_ads\_data.csv"):

"""Save data to a CSV file."""

with open(filename, "w", newline="") as file:

writer = csv.writer(file)

writer.writerow(["Campaign ID", "Campaign Name", "Impressions", "Clicks", "Cost (Currency)"])

writer.writerows(data)

print(f" Data saved to {filename}")

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

ads\_data = fetch\_campaign\_data()

save\_to\_csv(ads\_data)

2. Google Analytics Data Extraction: o Task: Write a Python script or use a tool (e.g., Google Cloud Dataflow or Airflow) to extract session data (e.g., session count, bounce rate, page views) from Google Analytics

import pandas as pd

import datetime

from google.analytics.data\_v1beta import BetaAnalyticsDataClient

from google.analytics.data\_v1beta.types import RunReportRequest, DateRange, Metric, Dimension

SERVICE\_ACCOUNT\_FILE = "service\_account.json"

# Google Analytics Property ID (Replace with your GA4 Property ID)

PROPERTY\_ID = "123456789"

# Initialize Analytics Data API Client

client = BetaAnalyticsDataClient.from\_service\_account\_file(SERVICE\_ACCOUNT\_FILE)

end\_date = datetime.date.today().strftime("%Y-%m-%d")

start\_date = (datetime.date.today() - datetime.timedelta(days=30)).strftime("%Y-%m-%d")

# Create request to fetch session data

request = RunReportRequest(

property=f"properties/{PROPERTY\_ID}",

date\_ranges=[DateRange(start\_date=start\_date, end\_date=end\_date)],

dimensions=[Dimension(name="date")], # Fetch data per day

metrics=[

Metric(name="sessions"),

Metric(name="bounceRate"),

Metric(name="screenPageViews"),

],

)

# Fetch data from Google Analytics

response = client.run\_report(request)

# Process response into a DataFrame

data = []

for row in response.rows:

data.append([row.dimension\_values[0].value, # Date

row.metric\_values[0].value, # Sessions

row.metric\_values[1].value, # Bounce Rate

row.metric\_values[2].value]) # Page Views

df = pd.DataFrame(data, columns=["Date", "Sessions", "Bounce Rate", "Page Views"])

# Save to CSV

csv\_filename = "google\_analytics\_data.csv"

df.to\_csv(csv\_filename, index=False)

print(f"Data saved to {csv\_filename}")

**Section 2**: Data Loading into BigQuery Objective: Load the extracted data into BigQuery tables.

1. Google Ads Data Loading: Task: Load the extracted Google Ads data into a BigQuery table.

from google.cloud import bigquery

import pandas as pd

# Path to your service account key file

SERVICE\_ACCOUNT\_FILE = "service\_account.json"

# Google Cloud project details

PROJECT\_ID = "your-gcp-project-id"

DATASET\_ID = "marketing\_data"

TABLE\_ID = "google\_ads\_performance"

# Initialize BigQuery client

client = bigquery.Client.from\_service\_account\_json(SERVICE\_ACCOUNT\_FILE)

# Define the BigQuery table schema

schema = [

bigquery.SchemaField("Campaign\_ID", "INTEGER"),

bigquery.SchemaField("Campaign\_Name", "STRING"),

bigquery.SchemaField("Impressions", "INTEGER"),

bigquery.SchemaField("Clicks", "INTEGER"),

bigquery.SchemaField("Cost", "FLOAT"),

]

# Read extracted Google Ads CSV data

csv\_filename = "google\_ads\_data.csv"

df = pd.read\_csv(csv\_filename)

# Define BigQuery table reference

table\_ref = client.dataset(DATASET\_ID).table(TABLE\_ID)

# Load data into BigQuery

job\_config = bigquery.LoadJobConfig(schema=schema, write\_disposition="WRITE\_TRUNCATE")

job = client.load\_table\_from\_dataframe(df, table\_ref, job\_config=job\_config)

job.result()

print(f" Data successfully loaded into {DATASET\_ID}.{TABLE\_ID}")

2. Google Analytics Data Loading: o Task: Load the extracted Google Analytics data into a BigQuery table

from google.cloud import bigquery

import pandas as pd

# Path to your service account key file

SERVICE\_ACCOUNT\_FILE = "service\_account.json"

# Google Cloud project details

PROJECT\_ID = "your-gcp-project-id"

DATASET\_ID = "marketing\_data"

TABLE\_ID = "google\_analytics\_sessions"

# Initialize BigQuery client

client = bigquery.Client.from\_service\_account\_json(SERVICE\_ACCOUNT\_FILE)

# Define the BigQuery table schema

schema = [

bigquery.SchemaField("Date", "DATE"),

bigquery.SchemaField("Sessions", "INTEGER"),

bigquery.SchemaField("Bounce\_Rate", "FLOAT"),

bigquery.SchemaField("Page\_Views", "INTEGER"),

]

# Read extracted Google Analytics CSV data

csv\_filename = "google\_analytics\_data.csv"

df = pd.read\_csv(csv\_filename)

# Convert "Date" column to datetime format

df["Date"] = pd.to\_datetime(df["Date"]).dt.date

# Define BigQuery table reference

table\_ref = client.dataset(DATASET\_ID).table(TABLE\_ID)

# Load data into BigQuery

job\_config = bigquery.LoadJobConfig(schema=schema, write\_disposition="WRITE\_TRUNCATE")

job = client.load\_table\_from\_dataframe(df, table\_ref, job\_config=job\_config)

# Wait for the job to complete

print(f" Data successfully loaded into {DATASET\_ID}.{TABLE\_ID}")

**Section 3**: Streaming Data with Pub/Sub Objective: Set up a Pub/Sub topic and a Dataflow pipeline to process streaming data.

1. Pub/Sub Setup: Task: Create a Pub/Sub topic named streaming\_data\_topic and a subscription named streaming\_data\_subscription.

from google.cloud import pubsub\_v1

# Google Cloud project details

PROJECT\_ID = "your-gcp-project-id"

TOPIC\_NAME = "streaming\_data\_topic"

SUBSCRIPTION\_NAME = "streaming\_data\_subscription"

# Initialize Publisher and Subscriber clients

publisher = pubsub\_v1.PublisherClient()

subscriber = pubsub\_v1.SubscriberClient()

# Create the Pub/Sub Topic

topic\_path = publisher.topic\_path(PROJECT\_ID, TOPIC\_NAME)

publisher.create\_topic(request={"name": topic\_path})

print(f"Pub/Sub Topic Created: {TOPIC\_NAME}")

# Create the Pub/Sub Subscription

subscription\_path = subscriber.subscription\_path(PROJECT\_ID, SUBSCRIPTION\_NAME)

subscriber.create\_subscription(request={"name": subscription\_path, "topic": topic\_path})

print(f"Pub/Sub Subscription Created: {SUBSCRIPTION\_NAME}")

1. Dataflow Pipeline: o Task: Create a Dataflow pipeline that reads data from the Pub/Sub subscription, processes it (e.g., filtering or transformation), and loads it into a BigQuery table named streaming\_data

import apache\_beam as beam

from apache\_beam.options.pipeline\_options import PipelineOptions, StandardOptions

from google.cloud import bigquery

PROJECT\_ID = "your-gcp-project-id"

SUBSCRIPTION\_NAME = "projects/your-gcp-project-id/subscriptions/streaming\_data\_subscription"

BQ\_DATASET = "your\_dataset"

BQ\_TABLE = "streaming\_data"

class ProcessData(beam.DoFn):

"""Custom transformation logic for streaming data"""

def process(self, element):

import json

record = json.loads(element.decode('utf-8')) # Decode Pub/Sub message

if record.get("impressions", 0) > 1000: # Example filter condition

yield {

"campaign\_id": record.get("campaign\_id"),

"campaign\_name": record.get("campaign\_name"),

"impressions": record.get("impressions"),

"clicks": record.get("clicks"),

"cost": record.get("cost"),

"event\_timestamp": record.get("event\_timestamp"),

}

# Define Pipeline Options

pipeline\_options = PipelineOptions(

streaming=True, # Enables streaming mode

project=PROJECT\_ID,

region="us-central1",

runner="DataflowRunner",

temp\_location=f"gs://your-bucket/temp",

)

pipeline\_options.view\_as(StandardOptions).streaming = True

# Define BigQuery Table Schema

BQ\_SCHEMA = """

campaign\_id:INTEGER,

campaign\_name:STRING,

impressions:INTEGER,

clicks:INTEGER,

cost:FLOAT,

event\_timestamp:TIMESTAMP

"""

# Create Dataflow Pipeline

with beam.Pipeline(options=pipeline\_options) as pipeline:

(

pipeline

| "Read from Pub/Sub" >> beam.io.ReadFromPubSub(subscription=SUBSCRIPTION\_NAME)

| "Process Data" >> beam.ParDo(ProcessData())

| "Write to BigQuery" >> beam.io.WriteToBigQuery(

f"{PROJECT\_ID}:{BQ\_DATASET}.{BQ\_TABLE}",

schema=BQ\_SCHEMA,

create\_disposition=beam.io.BigQueryDisposition.CREATE\_IF\_NEEDED,

write\_disposition=beam.io.BigQueryDisposition.WRITE\_APPEND

)

)

print(" Dataflow pipeline started successfully!")