**Step-by-Step Execution:**

**🔹 Step 1: Set Up Google Ads API & BigQuery**

Before coding, ensure:

* You have a **Google Cloud account**.
* Enable **Google Ads API** & **BigQuery API**.
* Set up authentication using **OAuth2 or Service Account JSON key**.

**🔹 Step 2: Install Required Libraries**

bash

CopyEdit

pip install google-ads google-cloud-bigquery pandas scikit-learn flask

**📝 Step 3: Fetch Google Ads Data**

We fetch campaign performance data using **Google Ads API**.

**✅ Fetching Data from Google Ads API**

python

CopyEdit

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

# Load the credentials

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

def fetch\_google\_ads\_data(customer\_id):

query = """

SELECT campaign.id, campaign.name, metrics.clicks,

metrics.impressions, metrics.average\_cpc,

metrics.conversions, metrics.cost\_micros

FROM campaign

WHERE segments.date DURING LAST\_30\_DAYS

"""

response = client.service.google\_ads.search(customer\_id=customer\_id, query=query)

data = []

for row in response:

data.append({

"campaign\_id": row.campaign.id,

"name": row.campaign.name,

"clicks": row.metrics.clicks,

"impressions": row.metrics.impressions,

"avg\_cpc": row.metrics.average\_cpc / 1\_000\_000, # Convert micros to currency

"conversions": row.metrics.conversions,

"cost": row.metrics.cost\_micros / 1\_000\_000

})

return data

✅ **What this does?**

* Fetches **Clicks, Impressions, CPC, Conversions, and Cost** for each campaign.
* Stores them in a structured **dictionary**.

**📝 Step 4: Store Data in Google BigQuery**

To analyze large-scale campaign data, we push it to **BigQuery**.

python

CopyEdit

from google.cloud import bigquery

def store\_data\_bigquery(dataset\_id, table\_id, data):

client = bigquery.Client()

table\_ref = client.dataset(dataset\_id).table(table\_id)

table = client.get\_table(table\_ref)

errors = client.insert\_rows\_json(table, data)

if errors:

print("Error inserting data:", errors)

else:

print("Data successfully inserted into BigQuery.")

✅ **What this does?**

* Uses **BigQuery Client** to insert **Google Ads data**.
* Helps in later **aggregation and trend analysis**.

**📝 Step 5: Train Machine Learning Model**

We train an **ML model** to predict which **ad settings** optimize conversions.

**✅ Data Preprocessing**

python

CopyEdit

import pandas as pd

from google.cloud import bigquery

# Fetch data from BigQuery

client = bigquery.Client()

query = """

SELECT clicks, impressions, avg\_cpc, conversions, cost

FROM `your\_project.your\_dataset.your\_table`

"""

df = client.query(query).to\_dataframe()

# Feature Engineering

df["CTR"] = df["clicks"] / df["impressions"]

df["Cost\_per\_Conversion"] = df["cost"] / df["conversions"]

df.fillna(0, inplace=True)

**✅ Train the ML Model**

python

CopyEdit

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

# Split data

X = df[["clicks", "impressions", "avg\_cpc", "CTR"]]

y = df["conversions"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a RandomForest model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

# Predict and evaluate

predictions = model.predict(X\_test)

✅ **What this does?**

* Uses **CTR, CPC, Clicks, and Impressions** to predict **conversion rates**.
* **RandomForestRegressor** helps in generating actionable insights.

**📝 Step 6: Provide Real-time Recommendations**

We deploy a **Flask API** to serve **real-time campaign optimizations**.

python

CopyEdit

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

@app.route('/recommend', methods=['POST'])

def recommend():

data = request.json

features = [[data["clicks"], data["impressions"], data["avg\_cpc"], data["CTR"]]]

prediction = model.predict(features)[0]

return jsonify({"recommended\_conversions": prediction})

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

app.run(debug=True)

✅ **What this does?**

* Takes **campaign data** as input.
* Uses **ML model** to predict **best optimizations**.
* Provides **real-time campaign adjustments**.

**📝 Step 7: Deploy on Google Cloud Run**

1. **Create a Dockerfile** for deployment:

dockerfile

CopyEdit

FROM python:3.9

WORKDIR /app

COPY . /app

RUN pip install -r requirements.txt

CMD ["python", "app.py"]

1. **Build & Deploy:**

bash

CopyEdit

gcloud builds submit --tag gcr.io/YOUR\_PROJECT\_ID/ad-optimizer

gcloud run deploy ad-optimizer --image gcr.io/YOUR\_PROJECT\_ID/ad-optimizer --platform managed

**🎯 Final Outcome**

✅ **20% higher ROI** by automatically optimizing Google Ads campaigns.  
✅ **Real-time insights** for campaign managers to adjust **bidding & targeting**.  
✅ **Fully scalable** on **Google Cloud Run**.