Approach 1: The AI Architect

My Mission:is to use BigQuery's built-in generative AI to architect intelligent business applications that generate creative content, summarize complex information, and forecast future trends directly within the data warehouse — for example, by building a hyper-personalized marketing engine that creates unique emails for every customer based on their individual purchase history and preferences

[Open in Colab](#) [Google Cloud Colab Enterprise logo Open in Colab Enterprise](#) [Open in Vertex AI Workbench](#) [Open in BigQuery Studio](#) [View on GitHub](#) **Share to:**

My Mission: Use BigQuery's built-in generative AI to architect intelligent business applications and workflows. Build solutions that can generate creative content, summarize complex information, or even forecast future trends directly within your data warehouse.

Your Toolbox (Must use at least one):

- ***Generative AI in SQL***:
  - `ML.GENERATE_TEXT`: The classic function for large-scale text generation.
  - `AI.GENERATE`: Generate free-form text or structured data based on a schema from a prompt.
  - `AI.GENERATE_BOOL`: Get a simple True/False answer about your data.
  - `AI.GENERATE_DOUBLE`: Extract a specific decimal number from text.
  - `AI.GENERATE_INT`: Extract a specific whole number from text.
  - `AI.GENERATE_TABLE`: Create a structured table of data from a single prompt.
  - `AI.FORECAST`: Predict future values for time-series data with a single function.
- ***Generative AI in BigFrames (Python)***:
  - `bigframes.ml.llm.GeminiTextGenerator`: Leverage the power of Gemini models in your Python workflows.
  - `bigframes.DataFrame.ai.forecast()`: Run powerful forecasting models directly on your DataFrames.

Inspiration:

- Build a ***Hyper-Personalized Marketing Engine***: Generate unique marketing emails for every customer based on their individual purchase history and preferences.
- Create an ***Executive "Insight" Dashboard***: Develop a dashboard that automatically ingests raw support call logs and transforms them into summarized, categorized, and actionable business insights.

Before you begin

Set up your Google Cloud project

The following steps are required, regardless of your notebook environment.

1. [Select or create a Google Cloud project](#). When you first create an account, you get a $300 free credit towards your compute/storage costs.
2. [Make sure that billing is enabled for your project](#).
3. [Enable the BigQuery, BigQuery Connection, and Vertex AI APIs](#).
4. If you are running this notebook locally, you need to install the [Cloud SDK](#).

```
! gcloud auth login
```

```
You are running on a Google Compute Engine virtual machine.
It is recommended that you use service accounts for authentication.
```

```
You can run:

  $ gcloud config set account `ACCOUNT`

to switch accounts if necessary.

Your credentials may be visible to others with access to this
virtual machine. Are you sure you want to authenticate with
your personal account?

Do you want to continue (Y/n)?  Y

Go to the following link in your browser, and complete the sign-in prompts:

    https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=32555940559.apps.googleusercontent.com&

Once finished, enter the verification code provided in your browser: 4/0AVMBsJhz3IofhcPSvEY-LzmoZd_wMyfoG_mz_M-Wt-

You are now logged in as [idriss.choudjem@gmail.com].
Your current project is [gothic-list-469006-v9].  You can change this setting by running:
  $ gcloud config set project PROJECT_ID
```

```
PROJECT_ID = "gothic-list-469006-v9"  # @param {type:"

# Set the project id
! gcloud config set project {PROJECT_ID}
# Dans un bloc Colab (préfixe !), remplace par ton PRO
! gcloud services enable bigquery.googleapis.com bigqu

```

PROJECT_ID: " gothic-list-469006-v9 "

```
Updated property [core/project].
Operation "operations/acat.p2-670703985696-f21c0221-e0c5-4040-aff5-ceb52dbd2d45" finished successfully.
```

Authenticate to your Google Cloud account

Depending on your Jupyter environment, you may have to manually authenticate. Follow the relevant instructions below.

1. Colab Enterprise or BigQuery Studio Notebooks

- Do nothing as you are already authenticated.

2. Colab, uncomment and run:

```
from google.colab import auth

auth.authenticate_user()
```

3. Créer le dataset pour le modèle:

```
%%bigquery marketing_model_dataset --project {PROJECT_ID}
CREATE SCHEMA IF NOT EXISTS `marketing_email_model_dataset`
OPTIONS (location="US");
```

Job ID 7299d1c9-e06c-4f0a-87f7-ac5fc0250780 successfully executed: 100%

Create BigQuery Cloud resource connection

You will need to create a [Cloud resource connection](#) to enable BigQuery to interact with Vertex AI services.

```
!bq mk --connection --location=us --project_id=gothic-list-469006-v9 \
    --connection_type=CLOUD_RESOURCE gothic-list-469006-v9
```

```
BigQuery error in mk operation: Already Exists: Connection
projects/670703985696/locations/us/connections/gothic-list-469006-v9
```

Set permissions for Service Account

The resource connection service account requires certain project-level permissions to interact with Vertex AI.

```
SERVICE_ACCT = !bq show --format=prettyjson --connection us.gothic-list-469006-v9 | grep bqcx-670703985696-ati7 |
SERVICE_ACCT_EMAIL = SERVICE_ACCT[-1]
print(SERVICE_ACCT_EMAIL)
```

bqcx-670703985696-ati7@gcp-sa-bigquery-condel.iam.gserviceaccount.com

```
import time

!gcloud projects add-iam-policy-binding --format=none $PROJECT_ID --member=serviceAccount:$SERVICE_ACCT_EMAIL --ro
!gcloud projects add-iam-policy-binding --format=none $PROJECT_ID --member=serviceAccount:$SERVICE_ACCT_EMAIL --ro

# wait 60 seconds, give IAM updates time to propagate, otherwise, following cells will fail
time.sleep(60)
```

```
Updated IAM policy for project [gothic-list-469006-v9].
Updated IAM policy for project [gothic-list-469006-v9].
```

I - Build a Hyper-Personalized Marketing Engine

Generate text and structured data with `AI.GENERATE` and `AI.GENERATE_TABLE`

These functions allow you to leverage the power of large language models (LLMs) directly within BigQuery to generate new text content, including creating content with a specified schema. Both functions work by sending requests to your choice of a generally available or preview Gemini model, and then returning that model's response.

Using `ML.GENERATE_TEXT`: for Hyper-Personalized Marketing email

Let's use the `ML.GENERATE_TEXT` function to generate unique marketing emails for every customer based on their individual purchase history and preferences from the `bigquery-public-data.thelook_ecommerce` datasets.

In today's highly competitive e-commerce landscape, customer retention and campaign optimization are critical success factors. Traditional marketing approaches often fall short because they fail to address the unique preferences and purchase histories of individual customers. By leveraging BigQuery's built-in generative AI capabilities, businesses can move beyond generic campaigns and instead generate hyper-personalized marketing emails tailored to each customer. This approach enables the creation of intelligent workflows and business applications that not only deliver engaging content, but also summarize insights, predict future trends, and drive stronger customer relationships. The result is a Hyper-Personalized Marketing Engine designed to maximize engagement, loyalty, and long-term growth.

🔑 Building of a Parametric User Table for Behavior Analysis

> the goal here is creating a **parametric table** that consolidates key user-level indicators from the public The Look eCommerce dataset. It builds an enriched customer base by calculating `total orders`, `average basket`, `total revenue`, `recency of last purchase`, and `purchase frequency (monthly and quarterly)`. It also tracks `product preferences by counting items purchased`, identifying the `top three products per user`, and determining `the top three products per country`. Additionally, it captures each user's most recent event. Finally, the table aggregates these insights to provide a comprehensive view of each customer, combining personal details, purchasing behavior, product preferences, last interactions, and `market-level trends`.

```
%%bigquery df_parametric --project {PROJECT_ID}

CREATE OR REPLACE TABLE `gothic-list-469006-v9.ETL.parametric` AS
WITH base_with_freq AS (
  SELECT
    u.id AS user_id,
    u.first_name,
    u.email,
    u.country,
    COUNT(o.order_id) AS total_orders,
    AVG(oi.sale_price) AS avg_basket,
    SUM(oi.sale_price) AS total_revenue,
    DATE_DIFF(CURRENT_DATE(), DATE(MAX(o.created_at)), MONTH) AS months_since_last_order,
    COUNT(o.order_id) / NULLIF(DATE_DIFF(CURRENT_DATE(), DATE(MIN(o.created_at)), MONTH), 0) AS freq_orders_per_m
    COUNT(o.order_id) / NULLIF(DATE_DIFF(CURRENT_DATE(), DATE(MIN(o.created_at)), MONTH) / 3, 0) AS freq_orders_p
  FROM `bigquery-public-data.thelook_ecommerce.users` u
  LEFT JOIN `bigquery-public-data.thelook_ecommerce.orders` o
    ON u.id = o.user_id
  LEFT JOIN `bigquery-public-data.thelook_ecommerce.order_items` oi
    ON o.order_id = oi.order_id
  GROUP BY u.id, u.first_name, u.email, u.country
),

# -- Comptage des produits par utilisateur
product_counts AS (
  SELECT
    u.id AS user_id,
    p.name AS product_name,
    COUNT(*) AS product_count
  FROM `bigquery-public-data.thelook_ecommerce.users` u
  JOIN `bigquery-public-data.thelook_ecommerce.orders` o
    ON u.id = o.user_id
  JOIN `bigquery-public-data.thelook_ecommerce.order_items` oi
    ON o.order_id = oi.order_id
  JOIN `bigquery-public-data.thelook_ecommerce.products` p
    ON oi.product_id = p.id
  WHERE p.name IS NOT NULL
  GROUP BY u.id, p.name
),

# -- Top 3 produits par utilisateur
top_products_by_user AS (
  SELECT
    user_id,
    ARRAY_AGG(product_name ORDER BY product_count DESC LIMIT 3) AS top_products_user
  FROM product_counts
  GROUP BY user_id
),

# -- Dernier évènement par utilisateur
latest_event AS (
  SELECT
    user_id,
    MAX(event_type) AS last_event
  FROM `bigquery-public-data.thelook_ecommerce.events`
  GROUP BY user_id
),

# -- Top 3 produits par pays
top_products_by_country AS (
  SELECT
    country,
    ARRAY_AGG(product_name ORDER BY product_count DESC LIMIT 3) AS top_products_country
  FROM (
    SELECT
      u.country,
      p.name AS product_name,
      COUNT(*) AS product_count
    FROM `bigquery-public-data.thelook_ecommerce.orders` o
    JOIN `bigquery-public-data.thelook_ecommerce.order_items` oi
      ON o.order_id = oi.order_id
    JOIN `bigquery-public-data.thelook_ecommerce.products` p
```

```
      ON oi.product_id = p.id
    JOIN `bigquery-public-data.thelook_ecommerce.users` u
      ON o.user_id = u.id
    WHERE p.name IS NOT NULL
    GROUP BY u.country, p.name
  )
  GROUP BY country
)

# -- Final : joindre les infos utilisateur avec leurs top produits, évènements et pays
SELECT
  b.user_id,
  b.first_name,
  b.email,
  b.country,
  IFNULL(b.total_orders, 0) AS total_orders,
  IFNULL(b.avg_basket, 0) AS avg_basket,
  IFNULL(b.total_revenue, 0) AS total_revenue,
  t.top_products_user,
  l.last_event,
  c.top_products_country,
  IFNULL(b.months_since_last_order, 0) AS months_since_last_order,
  IFNULL(b.freq_orders_per_month, 0) AS freq_orders_per_month,
  IFNULL(b.freq_orders_per_quarter, 0) AS freq_orders_per_quarter
FROM base_with_freq b
LEFT JOIN top_products_by_user t
  ON b.user_id = t.user_id
LEFT JOIN latest_event l
  ON b.user_id = l.user_id
LEFT JOIN top_products_by_country c
  ON b.country = c.country
ORDER BY b.user_id
# -- LIMIT 10000;

```

Job ID 9eb4f049-3386-4ff2-81b0-ba384ea6f974 successfully executed: 100%

```
#✅ Dataset exported to CSV and Parquet in /content/ETL
#✅ Table saved in BigQuery at gothic-list-469006-v9.ETL.parametric
```

```
✅ Dataset exported to CSV and Parquet in /content/ETL
✅ Table saved in BigQuery at gothic-list-469006-v9.ETL.parametric
```

Customer Segmentation with Parametric Features using KMeans and `ML.GENERATE_TEXT`

`The underlying idea of using K-Means combined with "ML.GENERATE_TEXT" is to group data into affinity clusters based on their parametric features, and then reduce the volume of data that needs labeling. By leveraging the centroid, labeling can first be done manually or automatically, before generalizing the centroid labels to the entire cluster related to. This provides a first level of customer profiling by group, before diving into a more granularity level for hyper-personalized marketing`

> Here's a complete **Hand off** aproach for customer segmentation on an e-commerce dataset by loading user-level parametric features from BigQuery, handling missing data and scaling features, reducing dimensionality via PCA, evaluating multiple KMeans clusterings using `inertia`, `silhouette`, `Calinski-Harabasz`, and `Davies-Bouldin` metrics to identify the optimal number of clusters `"k"` with a `combined score`, assigning users to segments and calculating centroids, saving the segmented dataset back to BigQuery, and generating diagnostic plots to visualize cluster evaluation metrics, with the business goal of providing actionable customer segments for targeted marketing, and personalized recommendations.

```
import numpy as np
import pandas as pd
from google.cloud import bigquery
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
from umap import UMAP
import plotly.graph_objects as go
from plotly.subplots import make_subplots


# --Connexion BigQuery
client = bigquery.Client(project=PROJECT_ID)


# --Charger les données depuis BigQuery
query = f"""
      SELECT *
FROM `{PROJECT_ID}.ETL.parametric`
"""
df_segmented = client.query(query).to_dataframe()


# --Features pour clustering
features = [
    "total_orders",
    "avg_basket",
    "total_revenue",
    "months_since_last_order",
    "freq_orders_per_month",
    "freq_orders_per_quarter"
]
X = df_segmented[features]


# --Imputation des valeurs manquantes + scaling
imputer = SimpleImputer(strategy="mean")
X_imputed = imputer.fit_transform(X)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)


# --Réduction dimensionnelle PCA
pca = PCA(n_components=min(5, X_scaled.shape[1]), random_state=42)
X_reduced = pca.fit_transform(X_scaled)


# --Boucle KMeans pour trouver le meilleur k
k_min, k_max = 2, 22
k_values = list(range(k_min, k_max+1))

inertias, silhouettes, calinski, davies = [], [], [], []
sample_size = min(1000, X_reduced.shape[0])  # échantillon silhouette si dataset trop grand

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10, algorithm="elkan")
    labels = kmeans.fit_predict(X_reduced)

    inertias.append(kmeans.inertia_)
    silhouettes.append(silhouette_score(X_reduced, labels, sample_size=sample_size))
    calinski.append(calinski_harabasz_score(X_reduced, labels))
    davies.append(davies_bouldin_score(X_reduced, labels))


# --Normalisation & score combiné
scaler_norm = MinMaxScaler()
inertia_norm = 1 - scaler_norm.fit_transform(np.array(inertias).reshape(-1,1)).flatten()
silhouette_norm = scaler_norm.fit_transform(np.array(silhouettes).reshape(-1,1)).flatten()
calinski_norm = scaler_norm.fit_transform(np.array(calinski).reshape(-1,1)).flatten()
davies_norm = 1 - scaler_norm.fit_transform(np.array(davies).reshape(-1,1)).flatten()

combined_score = (inertia_norm + silhouette_norm + calinski_norm + davies_norm) / 4
best_k = k_values[np.argmax(combined_score)]
```

```
print(f"✅ Best k according to combined score : {best_k}")


# --Clustering final
kmeans_final = KMeans(n_clusters=best_k, random_state=42, n_init=10, algorithm="elkan")
df_segmented["segment_raw"] = kmeans_final.fit_predict(X_scaled)
df_segmented["segment"] = ["seg_" + str(i+1) for i in df_segmented["segment_raw"]]


# --Centroïdes par segment
centroids = df_segmented.groupby("segment")[features].mean().reset_index()
print("✅ centroïds Tab :")
display(centroids)


# --Tableau enrichi final
df_result = df_segmented[[
    "user_id","email","first_name", "segment", "country",
    "total_orders", "avg_basket", "total_revenue",
    "top_products_user" ,"last_event", "top_products_country",
    "months_since_last_order", "freq_orders_per_month", "freq_orders_per_quarter"
]]

print("✅ Final DataFrame with segmentation ready to use")
display(df_result.head())


# --Sauvegarde vers BigQuery
table_id = f"{PROJECT_ID}.ETL.semi_label_parametric"
job = client.load_table_from_dataframe(
    df_result,
    table_id,
    job_config=bigquery.LoadJobConfig(write_disposition="WRITE_TRUNCATE"),
)
job.result()
print(f"✅ Results saved in {table_id}")


# --Graphiques diagnostics
fig = make_subplots(
    rows=2, cols=3,
    subplot_titles=("Inertia (Elbow)", "Silhouette", "Calinski-Harabasz", "Davies-Bouldin", "Score combiné", "")
)
fig.add_trace(go.Scatter(x=k_values, y=inertias, mode='lines+markers', name='Inertia'), row=1, col=1)
fig.add_trace(go.Scatter(x=k_values, y=silhouettes, mode='lines+markers', name='Silhouette'), row=1, col=2)
fig.add_trace(go.Scatter(x=k_values, y=calinski, mode='lines+markers', name='Calinski-Harabasz'), row=1, col=3)
fig.add_trace(go.Scatter(x=k_values, y=davies, mode='lines+markers', name='Davies-Bouldin'), row=2, col=1)
fig.add_trace(go.Scatter(x=k_values, y=combined_score, mode='lines+markers+text',
                         text=[f"{v:.2f}" for v in combined_score],
                         textposition="top center", name='Score combiné'), row=2, col=2)

fig.add_shape(type="line", x0=best_k, x1=best_k, y0=0, y1=1,
              line=dict(color="red", dash="dash"), row=2, col=2)

fig.update_layout(
    height=800, width=1300,
    title_text="Comparing metrics to determine the optimal number of clusters (K)",
    showlegend=False
)
fig.update_xaxes(title_text="k (clusters)")
fig.update_yaxes(title_text="metric Values")
fig.show()

```

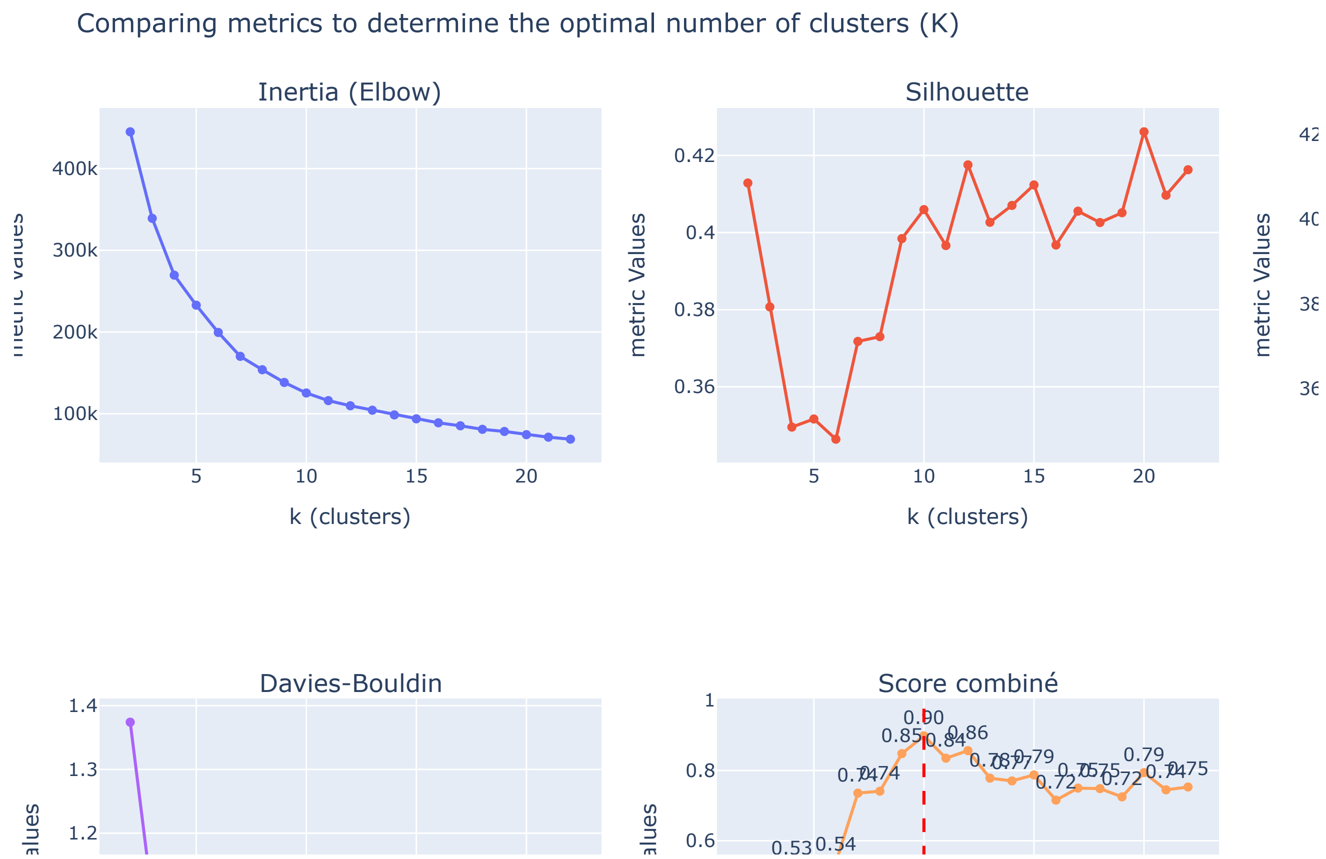
```
Best k according to combined score : 10
centroïds Tab :
```

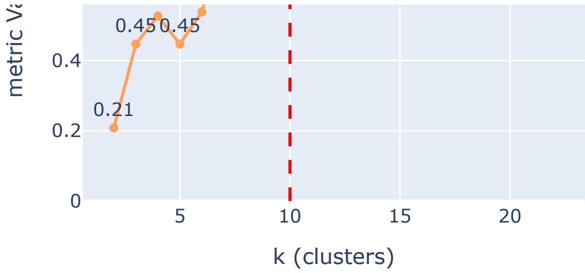
segment	total_orders	avg_basket	total_revenue	months_since_last_order	freq_orders_per_month	freq_orders_pe
seg_1	1.617485	148.680380	228.435312	15.482529	0.130856	
seg_10	6.092105	83.192744	475.947377	9.972588	0.296769	
seg_2	1.878878	47.875371	88.144137	2.035542	0.767615	
seg_3	0.061792	1.191212	1.191212	0.033671	0.001246	
seg_4	1.470852	41.461919	59.944883	11.297364	0.118169	
seg_5	5.010499	59.029332	295.659528	0.503937	4.716535	
seg_6	4.005712	52.644164	204.208923	11.151397	0.188059	
seg_7	1.711806	411.879188	636.499688	16.946181	0.204305	
seg_8	1.56305	46.868280	73.797649	43.569151	0.035904	
seg_9	3.450895	58.016835	200.115293	1.004838	2.035889	

```
Final DataFrame with segmentation ready to use
```

user_id	email	first_name	segment	country	total_orders	avg_basket	total_revenue	top_pr
1	reneewilson@example.net	Renee	seg_10	United Kingdom	6	53.201667	319.209999	Womer Neck
4	erikzavala@example.net	Erik	seg_1	United States	2	96.995000	193.990000	Sa Slim
6	williambarber@example.net	William	seg_10	Brasil	7	77.068571	539.480000	[N2N - Sleev
11	marialloyd@example.org	Maria	seg_8	United States	1	15.670000	15.670000	[Gr
12	samuelthomas@example.org	Samuel	seg_8	Brasil	1	9.950000	9.950000	[Ha Cushio

```
Results saved in gothic-list-469006-v9.ETL.semi_label_parametric
```

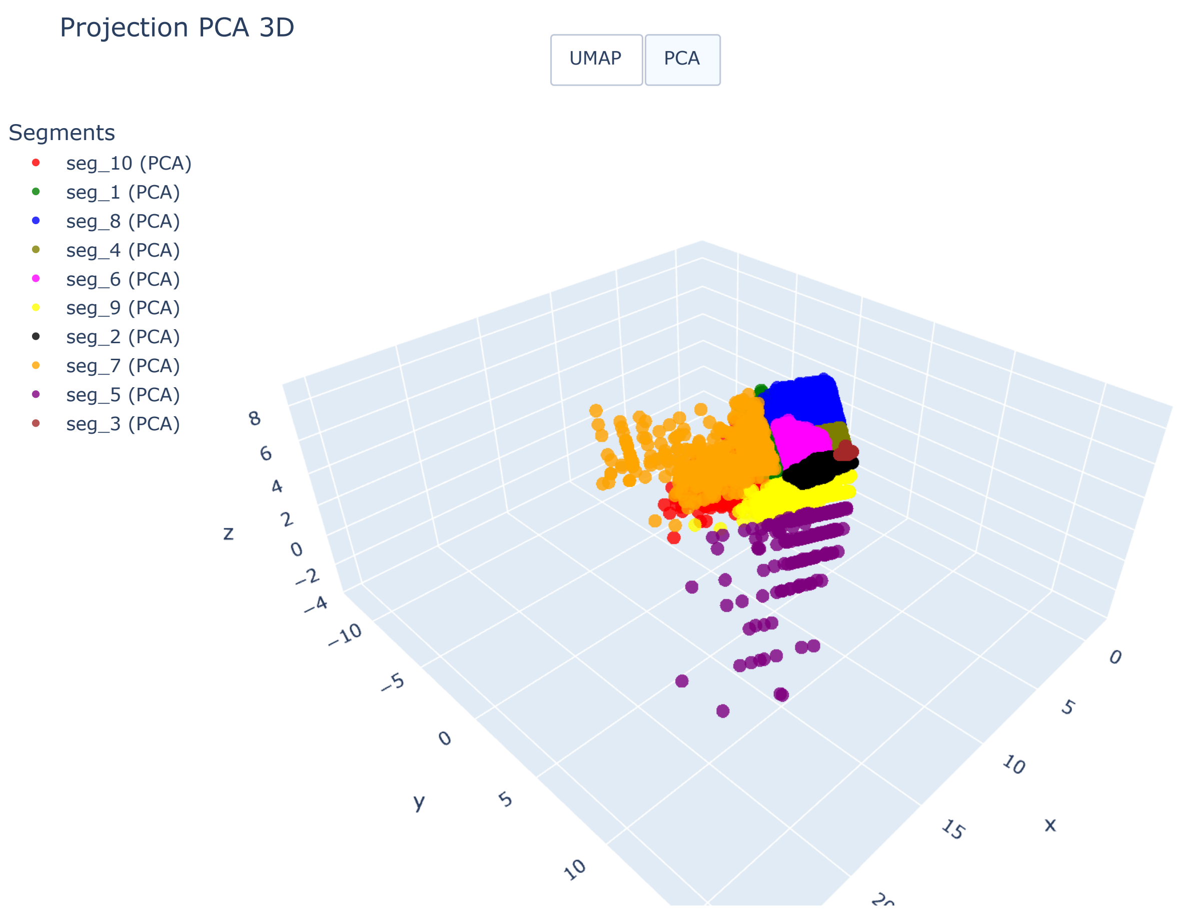




Interactive 3D Visualization of **semi-labeled** Customer Segments using UMAP and PCA

```
from umap import UMAP
import plotly.graph_objects as go
import seaborn as sns
from matplotlib import colors as mcolors


# --UMAP 3D + PCA 3D
umap_3d = UMAP(n_components=3, n_jobs=-1, random_state=None)
X_umap = umap_3d.fit_transform(X_scaled)

pca_3d = PCA(n_components=3, random_state=42)
X_pca = pca_3d.fit_transform(X_scaled)


# --Palette de couleurs fixe
colors_palette = ['red','green','blue','olive','magenta','yellow','black',
                  'orange','purple','brown','pink','lime','teal','navy',
                  'gold','salmon','cyan','maroon','aqua','violet']

segments = df_segmented["segment"].unique()
color_map = {seg: colors_palette[i % len(colors_palette)] for i, seg in enumerate(segments)}


# --Graphique 3D interactif UMAP/PCA
fig_3d = go.Figure()

# --UMAP traces
for seg in segments:
    idx = df_segmented["segment"] == seg
    fig_3d.add_trace(go.Scatter3d(
        x=X_umap[idx,0],
        y=X_umap[idx,1],
        z=X_umap[idx,2],
        mode="markers",
        marker=dict(size=6, color=color_map[seg], opacity=0.8),
        text=df_segmented.loc[idx, "segment"],
        name=f"{seg} (UMAP)",
        visible=True
    ))

# --PCA traces
for seg in segments:
    idx = df_segmented["segment"] == seg
    fig_3d.add_trace(go.Scatter3d(
        x=X_pca[idx,0],
        y=X_pca[idx,1],
        z=X_pca[idx,2],
        mode="markers",
        marker=dict(size=5, color=color_map[seg], opacity=0.8),
        text=df_segmented.loc[idx, "segment"],
        name=f"{seg} (PCA)",
        visible=False
    ))

# --Boutons UMAP/PCA
```

```python
n = len(segments)
fig_3d.update_layout(
    updatemenus=[dict(
        type="buttons",
        direction="right",
        x=0.5, y=1.1,
        buttons=[
            dict(label="UMAP", method="update",
                 args=[{"visible": [True]*n + [False]*n},
                       {"scene": dict(xaxis_title="UMAP-1", yaxis_title="UMAP-2", zaxis_title="UMAP-3"),
                        "title": "Projection UMAP 3D"}]),
            dict(label="PCA", method="update",
                 args=[{"visible": [False]*n + [True]*n},
                       {"scene": dict(xaxis_title="PCA-1", yaxis_title="PCA-2", zaxis_title="PCA-3"),
                        "title": "Projection PCA 3D"}])
        ]
    )]
)

# --Layout final
fig_3d.update_layout(
    title="Interactive 3D projection of segments",
    scene=dict(xaxis_title="UMAP-1", yaxis_title="UMAP-2", zaxis_title="UMAP-3"),
    height=700, width=950,
    legend=dict(
        title="Segments",
        orientation="v",
        yanchor="top",
        y=1,
        xanchor="left",
        x=-0.1
    )
)

fig_3d.show()

```



Automated Customer Segmentation with K-Means and BigQuery

> Here we're going to load customer data from BigQuery, applies K-Means to find the optimal clusters, assigns users to default segments (not custom ones), and produces both a segmented customer dataset and cluster centroids, saved in `semi_label_parametric` and `centroid_data` for advanced labeling purpose

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from google.cloud import bigquery


# --Charger les données depuis BigQuery
query = f"""
    SELECT *
    FROM `{PROJECT_ID}.ETL.parametric`
"""

client = bigquery.Client(project=PROJECT_ID)
df = client.query(query).to_dataframe()


# --Features & Préparation
features = [
    "total_orders",
    "avg_basket",
    "total_revenue",
```

```
    "months_since_last_order",
    "freq_orders_per_month",
    "freq_orders_per_quarter"
]

X = df[features].fillna(0)
X_scaled = StandardScaler().fit_transform(X)


# --Évaluer les k avec plusieurs métriques
k_values = list(range(2, 21))
inertias, silhouettes, calinski, davies = [], [], [], []

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10, algorithm="elkan")
    labels = kmeans.fit_predict(X_scaled)

    inertias.append(kmeans.inertia_)
    silhouettes.append(silhouette_score(X_scaled, labels))
    calinski.append(calinski_harabasz_score(X_scaled, labels))
    davies.append(davies_bouldin_score(X_scaled, labels))


# --Normalisation des scores & choix du meilleur k
scaler = MinMaxScaler()
inertia_norm = 1 - scaler.fit_transform(np.array(inertias).reshape(-1,1)).flatten()
silhouette_norm = scaler.fit_transform(np.array(silhouettes).reshape(-1,1)).flatten()
calinski_norm = scaler.fit_transform(np.array(calinski).reshape(-1,1)).flatten()
davies_norm = 1 - scaler.fit_transform(np.array(davies).reshape(-1,1)).flatten()

combined_score = (inertia_norm + silhouette_norm + calinski_norm + davies_norm) / 4
best_k = k_values[np.argmax(combined_score)]

print(f"✅ Best k according to combined score : {best_k}")


# --Clustering final avec meilleur k
kmeans_final = KMeans(n_clusters=best_k, random_state=42, n_init=10, algorithm="elkan")
df["segment_raw"] = kmeans_final.fit_predict(X_scaled)
df["segment"] = ["seg_" + str(i+1) for i in df["segment_raw"]]


# --Tableau enrichi final
df_result = df[[
    "user_id","email","first_name", "segment", "country",
    "total_orders", "avg_basket", "total_revenue",
    "top_products_user" ,"last_event", "top_products_country",
    "months_since_last_order", "freq_orders_per_month", "freq_orders_per_quarter"
]]

print("✅ Final DataFrame with segmentation ready to use")
display(df_result.head())


# --Centroïdes par segment
centroids = df.groupby("segment")[features].mean().reset_index()
print("✅ Centroids table :")
display(centroids)


# --Sauvegarde vers BigQuery

# --Table segmentation enrichie
table_segmentation = f"{PROJECT_ID}.ETL.semi_label_parametric"
job1 = client.load_table_from_dataframe(
    df_result,
    table_segmentation,
    job_config=bigquery.LoadJobConfig(
        write_disposition="WRITE_TRUNCATE"
    ),
)
job1.result()
```

```
print(f"✅ Segmentation results saved in {table_segmentation}")

# --Table centroïdes
table_centroids = f"{PROJECT_ID}.ETL.centroid_data"
job2 = client.load_table_from_dataframe(
    centroids,
    table_centroids,
    job_config=bigquery.LoadJobConfig(
        write_disposition="WRITE_TRUNCATE"
    ),
)
job2.result()
print(f"✅ Centroids results saved in {table_centroids}")

```

✅ Best k according to combined score : 10
✅ Final DataFrame with segmentation ready to use

	user_id	email	first_name	segment	country	total_orders	avg_basket	total_revenue	top_pr
0	1	reneewilson@example.net	Renee	seg_10	United Kingdom	6	53.201667	319.209999	Wome Neck
1	4	erikzavala@example.net	Erik	seg_1	United States	2	96.995000	193.990000	Sa Slir
2	6	williambarber@example.net	William	seg_10	Brasil	7	77.068571	539.480000	[N2N Slee
3	11	marialloyd@example.org	Maria	seg_8	United States	1	15.670000	15.670000	[G
4	12	samuelthomas@example.org	Samuel	seg_8	Brasil	1	9.950000	9.950000	[H Cushic

✅ Centroids table :

	segment	total_orders	avg_basket	total_revenue	months_since_last_order	freq_orders_per_month	freq_orders_p
0	seg_1	1.617485	148.680380	228.435312	15.482529	0.130856	
1	seg_10	6.092105	83.192744	475.947377	9.972588	0.296769	
2	seg_2	1.878878	47.875371	88.144137	2.035542	0.767615	
3	seg_3	0.061792	1.191212	1.191212	0.033671	0.001246	
4	seg_4	1.470852	41.461919	59.944883	11.297364	0.118169	
5	seg_5	5.010499	59.029332	295.659528	0.503937	4.716535	
6	seg_6	4.005712	52.644164	204.208923	11.151397	0.188059	
7	seg_7	1.711806	411.879188	636.499688	16.946181	0.204305	
8	seg_8	1.56305	46.868280	73.797649	43.569151	0.035904	
9	seg_9	3.450895	58.016835	200.115293	1.004838	2.035889	

✅ Segmentation results saved in gothic-list-469006-v9.ETL.semi_label_parametric
✅ Centroids results saved in gothic-list-469006-v9.ETL.centroid_data

There're 2 approaches for advanced labeling purpose, especially the `HAND-ON` and `HAND-OFF` approaches, then let's dive in

1. HAND ON

Here we define a mapping data object to create a correspondence and assign the correct label to the initial standard label based on centroid data analysis. However, it requires some expertise to determine the most accurate customer designation

```
%%bigquery df_label_parametric_Semi_hand_off --project $PROJECT_ID

CREATE OR REPLACE TABLE `gothic-list-469006-v9.ETL.label_parametric_Sho` AS #--sho for Semi hand_on
```

```
SELECT
  P.*,
  CASE segment
    WHEN 'seg_1'  THEN 'Dormant'
    WHEN 'seg_2'  THEN 'At Risk'
    WHEN 'seg_3'  THEN 'Occasional Big Spenders'
    WHEN 'seg_4'  THEN 'New / Active Low Spenders'
    WHEN 'seg_5'  THEN 'Loyal Customers'
    WHEN 'seg_6'  THEN 'Churned / Lost'
    WHEN 'seg_7'  THEN 'One-Timers'
    WHEN 'seg_8'  THEN 'Active Mid-Value'
    WHEN 'seg_9'  THEN 'Active / Promising'
    WHEN 'seg_10' THEN 'High-Value Dormant'
    ELSE 'Unknown'
  END AS segments
FROM `gothic-list-469006-v9.ETL.semi_label_parametric` P;

```

Job ID 428fcc19-25ff-4752-b334-7b6e4307a955 successfully executed: 100%

2. HAND OFF

That's what we're going to use next, so keep your eyes open. To carry the approach forward, we're going to use `ML.GENERATE_TEXT` function, by parsing the parametric centroid data along with a specific prompt to generate high-quality designations for each row, in order to build a highly customized mapping object in a hands-off way

```
%%bigquery df_label_parametric_Hand_off --project $PROJECT_ID

CREATE OR REPLACE TABLE `gothic-list-469006-v9.ETL.label_parametric` AS
WITH AI_labels AS (
  SELECT
    t.segment AS segment,
    -- Extract text from JSON output and then clean
    TRIM(SPLIT(SAFE_CAST(JSON_VALUE(t.ml_generate_text_result, '$.candidates[0].content.parts[0].text') AS STRING)
    TRIM(SPLIT(SAFE_CAST(JSON_VALUE(t.ml_generate_text_result, '$.candidates[0].content.parts[0].text') AS STRING)
  FROM ML.GENERATE_TEXT(
    MODEL `gothic-list-469006-v9.marketing_email_model_dataset.marketing_email_model`,
    (
      SELECT
        segment,
        CONCAT(
          'Segment: ', segment, '. ',
          'Total Orders: ', total_orders, '. ',
          'Average Basket: ', avg_basket, '. ',
          'Total Revenue: ', total_revenue, '. ',
          'Months Since Last Order: ', months_since_last_order, '. ',
          'Frequency per Month: ', freq_orders_per_month, '. ',
          'Frequency per Quarter: ', freq_orders_per_quarter, '. ',
          'Based on this data, you MUST assign a concise human-readable segment label, (without bold formatting) j
          'First line = segment label, second line = short designation.'
        ) AS prompt
      FROM `gothic-list-469006-v9.ETL.centroid_data`
    ),
    STRUCT(0.2 AS temperature, 150 AS max_output_tokens, 0.7 AS top_p, 1 AS top_k)
  ) AS t
)
SELECT
  P.*,
  A.segments,
  A.designation
FROM `gothic-list-469006-v9.ETL.semi_label_parametric` P
LEFT JOIN AI_labels A
USING(segment);
```

Job ID 1459d1f6-93d2-4c66-995f-7375bdcbd959 successfully executed: 100%

Here's the outcome related to the `HAND OFF` approach of the labeling process

```
%%bigquery --project {PROJECT_ID}

SELECT
  user_id,
  email,
  first_name,
  segment,
  segments AS type_of_customer,
  designation,
  country,
  total_orders,
  avg_basket,
  total_revenue,
  top_products_user,
  last_event,
  top_products_country,
  months_since_last_order,
  freq_orders_per_month,
  freq_orders_per_quarter
 FROM `gothic-list-469006-v9.ETL.label_parametric`
 ORDER BY user_id;
```

Job ID 23b48763-7813-4798-be66-f777fdd5e492 successfully executed: 100%

Downloading: 100%

	user_id	email	first_name	segment	type_of_customer	designation	country	total_or
0	1	reneewilson@example.net	Renee	seg_10	Potential Churn		United Kingdom	
1	2	valerieporter@example.net	Valerie	seg_3	Infrequent Recent Buyers	Low-value, very recent customers with minimal ...	United States	
2	3	gabrielhuff@example.org	Gabriel	seg_3	Infrequent Recent Buyers	Low-value, very recent customers with minimal ...	United States	
3	4	erikzavala@example.net	Erik	seg_1	Lapsed Buyers	Infrequent purchasers with a declining engagem...	United States	
4	5	adamfrazier@example.org	Adam	seg_3	Infrequent Recent Buyers	Low-value, very recent customers with minimal ...	China	
...	...	...	...	...	...	...	...	
99995	99996	taylorle@example.net	Taylor	seg_6	Lapsed Regulars	Infrequent purchasers with a moderate average ...	United States	
99996	99997	mackenzieramirez@example.com	Mackenzie	seg_4	Lapsed Purchasers	Infrequent buyers with a high average basket v...	United States	
99997	99998	kellygoodman@example.org	Kelly	seg_6	Lapsed Regulars	Infrequent purchasers with a moderate average ...	Brasil	
99998	99999	maryjohnson@example.net	Mary	seg_2	Recent High-Value Customer	Recently acquired customer with a high average...	Brasil	
99999	100000	amandaholloway@example.com	Amanda	seg_6	Lapsed Regulars	Infrequent purchasers with a moderate average ...	China	

100000 rows × 16 columns

Interactive 3D Visualization for now **labeled** Customer Segments using UMAP and PCA

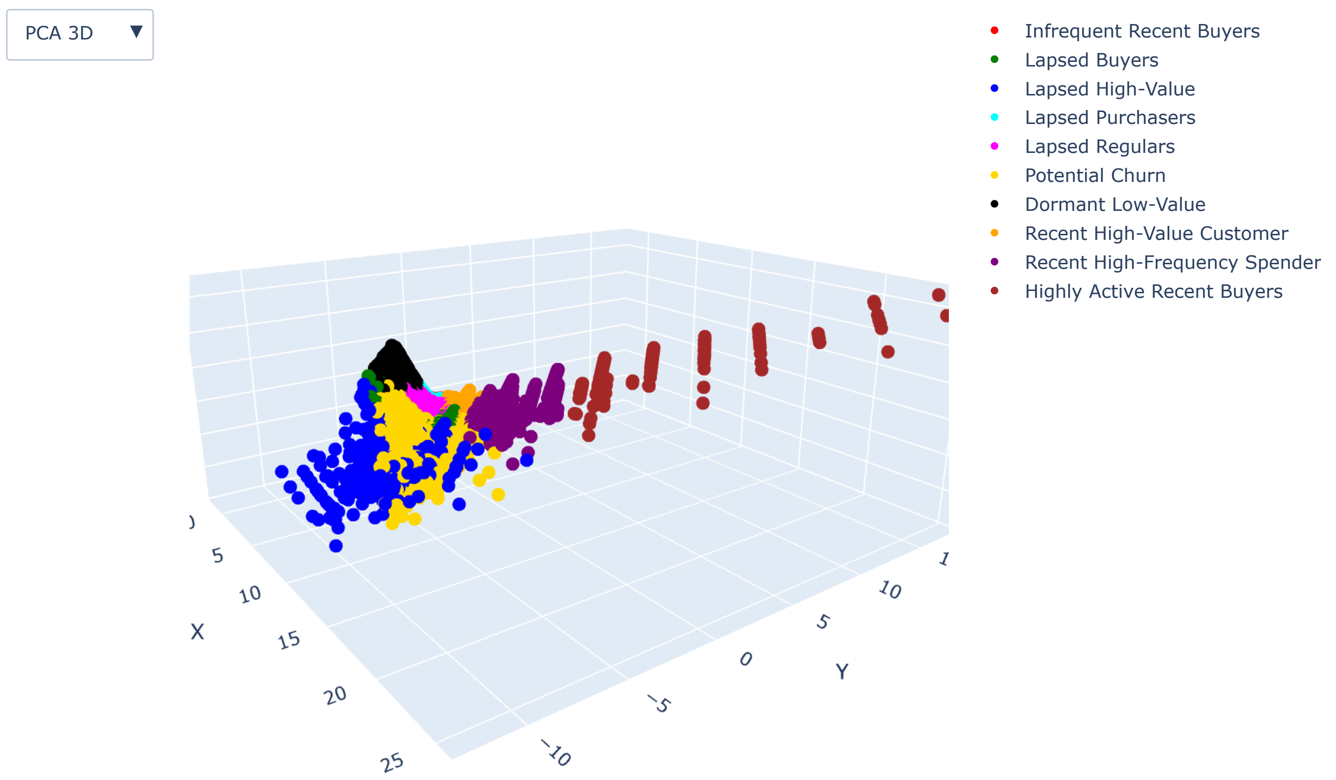
```
import pandas as pd
import numpy as np
import re
from google.cloud import bigquery
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
from umap import UMAP
```

```python
from umap import UMAP
import plotly.graph_objects as go


# --Connexion BigQuery et chargement des données
client = bigquery.Client(project=PROJECT_ID)

query = f"""
SELECT *
FROM `{PROJECT_ID}.ETL.label_parametric`
"""
df = client.query(query).to_dataframe()


# --Features numériques
features = [
    "total_orders", "avg_basket", "total_revenue",
    "months_since_last_order", "freq_orders_per_month", "freq_orders_per_quarter"
]

X = df[features]
imputer = SimpleImputer(strategy="mean")
X_imputed = imputer.fit_transform(X)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)


# --PCA 3D
pca_3d = PCA(n_components=3, random_state=42)
X_pca = pca_3d.fit_transform(X_scaled)
df["PCA1"], df["PCA2"], df["PCA3"] = X_pca[:,0], X_pca[:,1], X_pca[:,2]


# --UMAP 3D
umap_3d = UMAP(n_components=3, random_state=42, n_jobs=-1)
X_umap = umap_3d.fit_transform(X_scaled)
df["UMAP1"], df["UMAP2"], df["UMAP3"] = X_umap[:,0], X_umap[:,1], X_umap[:,2]


# --Palette de couleurs et mapping segments
colors_palette = [
    'red','green','blue','cyan','magenta','gold','black',
    'orange','purple','brown','pink','lime','teal','navy',
    'yellow','salmon','olive','maroon','aqua','violet'
]

segments = df["segments"].unique()

color_map = {seg: colors_palette[i % len(colors_palette)] for i, seg in enumerate(segments)}

# Nettoyage pour enlever les * au début et à la fin
# clean_segments = [re.sub(r"^\*+|\*+$", "", str(seg)).strip() for seg in segments]

# Génération du color_map avec les segments nettoyés
# color_map = {clean_seg: colors_palette[i % len(colors_palette)] for i, clean_seg in enumerate(clean_segments)}


# --Graphique 3D interactif PCA et UMAP
fig = go.Figure()

# Ajouter PCA 3D (visible par défaut)
for seg in segments:
    # clean_seg = re.sub(r"^\*+|\*+$", "", str(seg)).strip()
    df_seg = df[df["segments"] == seg]
    fig.add_trace(go.Scatter3d(
        x=df_seg["PCA1"], y=df_seg["PCA2"], z=df_seg["PCA3"],
        mode='markers',
        marker=dict(size=5, color=color_map[seg]),
        name=f"{seg}",
        visible=True
    ))

```

```
# --Ajouter UMAP 3D (invisible au départ)
for seg in segments:
    # clean_seg = re.sub(r"^\*+|\*+$", "", str(seg)).strip()
    df_seg = df[df["segments"] == seg]
    fig.add_trace(go.Scatter3d(
        x=df_seg["UMAP1"], y=df_seg["UMAP2"], z=df_seg["UMAP3"],
        mode='markers',
        marker=dict(size=5, color=color_map[seg]),
        name=f"{seg}",
        visible=False
    ))

# --Dropdown pour basculer entre PCA et UMAP
buttons = [
    dict(label="PCA 3D",
         method="update",
         args=[{"visible": [True]*len(segments) + [False]*len(segments)},
               {"title": "PCA 3D customers per segment"}]),
    dict(label="UMAP 3D",
         method="update",
         args=[{"visible": [False]*len(segments) + [True]*len(segments)},
               {"title": "UMAP 3D customer per segment"}])
]

fig.update_layout(
    updatemenus=[dict(active=0, buttons=buttons)],
    height=700, width=900,
    title="PCA 3D of customer segment",
    scene=dict(
        xaxis_title='X',
        yaxis_title='Y',
        zaxis_title='Z'
    )
)

fig.show()

```

```
/usr/local/lib/python3.12/dist-packages/umap/umap_.py:1952: UserWarning:

n_jobs value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
```

PCA 3D customers per segment



Customer Segmentation and Multilingual Marketing Email Prompt Generation in BigQuery

Now we build a final dataset: `final_data` in BigQuery `(ETL.label_parametric)` by enriching customer data with country-based language codes and detailed behavioral attributes. It assigns each customer a preferred communication language based on their country, then generates a dynamic marketing prompt tailored to their profile. The prompt incorporates purchase history, `frequency`, `basket size`, `last activity`, and `product preferences` and so on, while adapting the tone to their customer segment prevuously defined in the hand off labeling process. The final output provides a structured foundation for creating personalized, multilingual, and engaging marketing emails designed.

```
%%bigquery df_final_data --project {PROJECT_ID}

CREATE OR REPLACE TABLE `gothic-list-469006-v9.marketing_email_model_dataset.final_data` AS
WITH segmented AS (
  SELECT
    b.*,
    CASE
      WHEN LOWER(b.country) IN (
        'france', 'belgium', 'switzerland', 'luxembourg',
        'senegal', 'ivory coast', 'cameroon',
        'madagascar', 'togo', 'mali', 'morocco', 'tunisia', 'algeria'
      ) THEN 'fr'
      WHEN LOWER(b.country) IN (
        'spain', 'mexico', 'argentina', 'chile', 'colombia', 'peru',
```

```
        'venezuela', 'ecuador', 'uruguay', 'paraguay', 'bolivia',
        'guatemala', 'cuba', 'dominican republic'
      ) THEN 'es'
      WHEN LOWER(b.country) IN ('brasil', 'portugal', 'mozambique', 'angola') THEN 'pt'
      WHEN LOWER(b.country) IN (
        'united states', 'usa', 'united kingdom', 'canada', 'england', 'australia',
        'ireland', 'nigeria', 'ghana', 'kenya', 'south africa', 'india',
        'philippines'
      ) THEN 'en'
      WHEN LOWER(b.country) IN (
        'egypt', 'saudi arabia', 'uae', 'united arab emirates', 'qatar',
        'jordan', 'lebanon', 'iraq', 'syria', 'yemen'
      ) THEN 'ar'
      WHEN LOWER(b.country) IN ('china', 'taiwan', 'singapore') THEN 'zh'
      WHEN LOWER(b.country) IN ('japan') THEN 'ja'
      WHEN LOWER(b.country) IN ('germany') THEN 'de'
      WHEN LOWER(b.country) IN ('south korea', 'korea', 'republic of korea') THEN 'ko'
      ELSE 'en'
    END AS lang
  FROM `gothic-list-469006-v9.ETL.label_parametric` b
),

final_data AS (
  SELECT
    s.user_id,
    s.email,
    s.first_name,
    s.country,
    s.lang,
    s.segment,
    s.segments,
    s.designation,
    s.total_orders,
    s.avg_basket,
    s.freq_orders_per_month,
    s.freq_orders_per_quarter,
    s.last_event,
    s.top_products_user,
    s.top_products_country,
    CONCAT(
      "Write a short, friendly, wel structured and engaging marketing email", "just the ","result"," in language o
      "This email is intended for a ", "'",s.segments,"'", "without explicitely mentioned", s.segments,",", "custo
      " from ", s.country, ". ",
      "They have made ", CAST(s.total_orders AS STRING), " purchases this year with an average basket of $", FORMA
      "They buy approximately ", FORMAT('%.2f', s.freq_orders_per_month), " times per month. ",
      "Their last activity: ", IFNULL(s.last_event, 'no recent events'), ". ",
      "With Customer’s top products: ", ARRAY_TO_STRING(s.top_products_user, ', '), ". ",
      "Top trending products in their country: ", ARRAY_TO_STRING(s.top_products_country, ', '), ". ",
      "Adapt tone based on segment: exclusive for VIP, motivating for Active, welcoming for Dormant, and exciting
      "The goal is to maximize customer retention, attention and encourage engagement depending on", s.segments, "
    ) AS prompt
  FROM segmented s
)

SELECT * FROM final_data ORDER BY user_id;
```

Job ID 0fced6ca-6a48-4bf5-b5f1-271cac7a1fe8 successfully executed: 100%

The use of `ML.GENERATE_TEXT()` to generate Personalized email for business activity purpose

So far, everything is working well. At this stage, our main focus is to extract first the `email and prompt` features along with any other optional attributes and feed them into [ML.GENERATE_TEXT()](https://colab.research.google.com) to generate hyper-personalized emails, and there after storing through `marketing_email_P500` ou `marketing_email_Pxxx`

```
%%bigquery Hyper_personalized_email --project {PROJECT_ID}

# -- Sélectionner top 500 prompts
CREATE OR REPLACE TABLE `gothic-list-469006-v9.marketing_email_model_dataset.marketing_email_P500` AS
WITH prompts AS (
  SELECT
    email,
    country,
    lang AS language_code,
    prompt,
    ROW_NUMBER() OVER (ORDER BY FARM_FINGERPRINT(email)) AS prompt_id
  FROM `gothic-list-469006-v9.marketing_email_model_dataset.final_data`
  LIMIT 500
  -- QUALIFY prompt_id <= 50  pour restreindre
),

-- Génération du contenu marketing avec ML.GENERATE_TEXT() function.
marketing_mail AS (
  SELECT
    p.email,
    p.language_code,
    p.country,
    SAFE_CAST(
      JSON_VALUE(
        ml_output.ml_generate_text_result,
        '$.candidates[0].content.parts[0].text'
      ) AS STRING
    ) AS marketing_email
  FROM
    ML.GENERATE_TEXT(
      MODEL `gothic-list-469006-v9.marketing_email_model_dataset.marketing_email_model`,
      (
        SELECT prompt, prompt_id FROM prompts
      ),
      STRUCT(
        300 AS max_output_tokens,
        0.7 AS temperature
        -- ['\n\n'] AS stop_sequences
      )
    ) AS ml_output
  JOIN prompts p
    ON ml_output.prompt = p.prompt
   AND ml_output.prompt_id = p.prompt_id
)

-- Résultats finaux filtrés et ordonnés
SELECT
  email,
  country,
  language_code,
  marketing_email
FROM marketing_mail
WHERE marketing_email IS NOT NULL
ORDER BY email;
```