Importing Required Libraries

1. **pandas, numpy**: Used for data manipulation and numerical operations.
2. **datetime**: To handle date conversions and calculate recency.
3. **random**: To generate random scores for business logic.

```
import pandas as pd
import numpy as np
from datetime import datetime
import random
```

SettingUp Constants

Defines key parameters:

1. alpha, beta: Weights for calculating the combined score (frequency and recency).
2. purchasing_power: Represents the consultant's budget.
3. k: Maximum bundle size.
4. C_max: Maximum cost allowed for a bundle.
5. gamma, delta: Weights for scoring candidates.
6. n_categories: Maximum categories per bundle.
7. num_bundles: Number of bundles to generate per consultant.

```
alpha = 0.4
beta = 0.6
purchasing_power = 100.0
k = 7   # size of bundles
C_max = 100.0
theta = 1
gamma = 0.25
delta = 0.75
n_categories = 7  # max number of categories allowed per bundle
num_bundles = 4  # number of bundles
```

Reads a CSV file named '*sample_data_1000_C.csv*' into a DataFrame.

```
df = pd.read_csv('/content/sample_data_1000_C.csv')
df.head()
```

	CODEBELISTA	CODPRODUCTOSAP	DESCATEGORIA	DESMARCA	PRECIOOFERTA	FECHAPROCESO
0	44109905	200112294	MAQUILLAJE	CYZONE	9.50	2023-02-09
1	36949902	200086399	FRAGANCIAS	LBEL	19.95	2023-02-02
2	49968221	200089498	TRATAMIENTO CORPORAL	ESIKA	36.90	2023-02-10
3	46114531	210102388	BIJOUTERIE	ESIKA	59.90	2023-02-21
4	50368645	210100620	COMPLEMENTOS	CYZONE	74.90	2023-03-01

Data Pre-processing

Data Cleaning Column names are renamed for clarity. Missing values in the category column are filled with "OTHERS."

Dates are converted to datetime format for aggregation.

```
#Renaming the columns
column_rename_map={
                "CODEBELISTA": "consultant_id",
                "CODPRODUCTOSAP": "product_id",
                "DESCATEGORIA": "category",
                "DESMARCA": "brand",
                "PRECIOOFERTA": "price",
                "FECHAPROCESO": "date"
            }
df.rename(columns=column_rename_map, inplace=True)
df.head()
```

	consultant_id	product_id	category	brand	price	date
0	44109905	200112294	MAQUILLAJE	CYZONE	9.50	2023-02-09
1	36949902	200086399	FRAGANCIAS	LBEL	19.95	2023-02-02
2	49968221	200089498	TRATAMIENTO CORPORAL	ESIKA	36.90	2023-02-10
3	46114531	210102388	BIJOUTERIE	ESIKA	59.90	2023-02-21
4	50368645	210100620	COMPLEMENTOS	CYZONE	74.90	2023-03-01

```
#Filling Null Values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36310 entries, 0 to 36309
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   consultant_id  36310 non-null  int64  
 1   product_id     36310 non-null  int64  
 2   category       36228 non-null  object 
 3   brand          36310 non-null  object 
 4   price          36310 non-null  float64
 5   date           36310 non-null  object 
dtypes: float64(1), int64(2), object(3)
memory usage: 1.7+ MB
```

```
df['category'].fillna('OTHERS', inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36310 entries, 0 to 36309
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   consultant_id  36310 non-null  int64  
 1   product_id     36310 non-null  int64  
 2   category       36310 non-null  object 
 3   brand          36310 non-null  object 
 4   price          36310 non-null  float64
 5   date           36310 non-null  object 
dtypes: float64(1), int64(2), object(3)
memory usage: 1.7+ MB
<ipython-input-29-d823663f0706>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df['category'].fillna('OTHERS', inplace=True)
```

```
# Convert `date` column to datetime
df["date"] = pd.to_datetime(df["date"], errors="coerce")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36310 entries, 0 to 36309
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   consultant_id  36310 non-null  int64         
 1   product_id     36310 non-null  int64         
 2   category       36310 non-null  object        
 3   brand          36310 non-null  object        
 4   price          36310 non-null  float64       
 5   date           36310 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(1), int64(2), object(2)
memory usage: 1.7+ MB
```

Daily Aggregation Groups data by consultant_id and date to compute frequency and daily_total_spent

```
# Perform daily aggregation
group_by_columns = ["consultant_id", "date"]
daily_agg = df.groupby(group_by_columns, as_index=False).agg({
    "product_id": "count",
    "price": "sum"
})

# Rename columns for clarity
daily_agg.rename(columns={
    "product_id": "frequency",
    "price": "daily_total_spent"
}, inplace=True)

daily_agg.head()
```

	consultant_id	date	frequency	daily_total_spent
0	3441296	2022-12-27	10	263.40
1	3441296	2023-02-07	8	273.30
2	3441296	2023-02-15	1	23.90
3	3441296	2023-02-28	7	261.45
4	3441296	2023-04-11	17	391.10

IQR and Purchasing Power Interquartile Range (IQR) is calculated for daily_total_spent to identify spending patterns. average_purchasing_power is set to Q3 (75th percentile of spending).

```
# Calculate IQR for daily_total_spent
Q1 = np.percentile(daily_agg["daily_total_spent"], 25)
Q3 = np.percentile(daily_agg["daily_total_spent"], 75)
IQR = Q3 - Q1

# Add average purchasing power (Q3) as a new column
daily_agg["average_purchasing_power"] = Q3
```

Merge Aggregates

1. Daily aggregates are merged back into the main dataset.
2. Recency is calculated as days since the last purchase.

```
# Merge the daily aggregates back into the original DataFrame
df = pd.merge(df, daily_agg, on=group_by_columns, how="left")
df.head()
```

	consultant_id	product_id	category	brand	price	date	frequency	daily_total_spent	average_purchasing_power
0	44109905	200112294	MAQUILLAJE	CYZONE	9.50	2023-02-09	17	427.40	430.195
1	36949902	200086399	FRAGANCIAS	LBEL	19.95	2023-02-02	16	403.90	430.195
2	49968221	200089498	TRATAMIENTO CORPORAL	ESIKA	36.90	2023-02-10	21	406.30	430.195
3	[illegible]	[illegible]	BIJOUTERIE	ESIKA	[illegible]	2023-02-	11	[illegible]	430.195

```
# Calculate recency in days
current_date = datetime.now()
df["recency"] = (current_date - df["date"]).dt.days
df.head()
```

	consultant_id	product_id	category	brand	price	date	frequency	daily_total_spent	average_purchasing_power	recency
0	44109905	200112294	MAQUILLAJE	CYZONE	9.50	2023-02-09	17	427.40	430.195	688
1	36949902	200086399	FRAGANCIAS	LBEL	19.95	2023-02-02	16	403.90	430.195	695
2	49968221	200089498	TRATAMIENTO CORPORAL	ESIKA	36.90	2023-02-10	21	406.30	430.195	687
3						2023-				

Normalization Frequency and Recency Normalization: Converts raw values to a range between 0 and 1. Handles edge cases where all values are identical.

Recency is calculated as days since the last purchase.

```
# Normalize frequency
freq_min = df["frequency"].min()
freq_max = df["frequency"].max()

if freq_min != freq_max:
    df["frequency_normalized"] = (df["frequency"] - freq_min) / (freq_max - freq_min)
else:
    df["frequency_normalized"] = 0.5
```

```
# Normalize recency
rec_min = df["recency"].min()
rec_max = df["recency"].max()
ddf?
if rec_min != rec_max:
    df["recency_normalized"] = (df["recency"] - rec_min) / (rec_max - rec_min)
else:
    df["recency_normalized"] = 0.5
```

Consultant-Level Metrics total_spent: Total spending by each consultant.

purchase_frequency: Total number of purchases.

unique_products: Number of unique products purchased.

```
# Consultant-level metrics
df["total_spent"] = df.groupby("consultant_id")["price"].transform("sum")
df["purchase_frequency"] = df.groupby("consultant_id")["product_id"].transform("count")
df["unique_products"] = df.groupby("consultant_id")["product_id"].transform("nunique")
```

```
df.head()
```

	consultant_id	product_id	category	brand	price	date	frequency	daily_total_spent	average_purchasing_power	recency	f
0	44109905	200112294	MAQUILLAJE	CYZONE	9.50	2023-02-09	17	427.40	430.195	688	
1	36949902	200086399	FRAGANCIAS	LBEL	19.95	2023-02-02	16	403.90	430.195	695	
2	49968221	200089498	TRATAMIENTO CORPORAL	ESIKA	36.90	2023-02-10	21	406.30	430.195	687	
3	46114531	210102388	BIJOUTERIE	ESIKA	59.90	2023-02-21	11	469.10	430.195	676	
4	50368645	210100620	COMPLEMENTOS	CYZONE	74.90	2023-03-01	5	202.55	430.195	668	

```
df.to_csv("preprocessed_data_1000_C.csv", index=False)
```

Combined Score for each row based on normalized frequency and recency

Combined Score : A weighted combination of normalized frequency and recency: **combined_score** = alpha * normalized_f + beta * normalized_r

```
df = pd.read_csv('/content/preprocessed_data_1000_C.csv')

def normalize(value, min_value, max_value):
    if min_value == max_value:
        return 0
    return (value - min_value) / (max_value - min_value)

min_f = df['purchase_frequency'].min()
max_f = df['purchase_frequency'].max()
min_r = df['recency'].min()
max_r = df['recency'].max()

df['normalized_f'] = df['purchase_frequency'].apply(lambda x: normalize(x, min_f, max_f))
df['normalized_r'] = df['recency'].apply(lambda x: normalize(x, min_r, max_r))
df['combined_score'] = (alpha * df['normalized_f']) + (beta * df['normalized_r'])
df.head()
```

ct_id	category	brand	price	date	frequency	daily_total_spent	average_purchasing_power	recency	fr
12294	MAQUILLAJE	CYZONE	9.50	2023-02-09	17	427.40	430.195	688	
86399	FRAGANCIAS	LBEL	19.95	2023-02-02	16	403.90	430.195	695	
89498	TRATAMIENTO CORPORAL	ESIKA	36.90	2023-02-10	21	406.30	430.195	687	
02388	BIJOUTERIE	ESIKA	59.90	2023-02-21	11	469.10	430.195	676	
00620	COMPLEMENTOS	CYZONE	74.90	2023-03-01	5	202.55	430.195	668	

select_anchor_product(df): Finds the product with the highest combined_score in the DataFrame, serving as the bundle's starting point.

generate_candidates(bundle, df): Returns products not already in the bundle by filtering out existing product IDs.

category_score(product, bundle): Checks if a product's category exists in the bundle, returning 1 for a match or 0 otherwise.

business_score(product): Generates a random score for a product as a placeholder for advanced business logic.

```
def select_anchor_product(df):
        return df.loc[df['combined_score'].idxmax()]

def generate_candidates(bundle, df):
        bundle_product_ids = {p['product_id'] for p in bundle}
        return df[~df['product_id'].isin(bundle_product_ids)]

def category_score(product, bundle):
    return 1 if product['category'] in [b['category'] for b in bundle] else 0

def business_score(product):
    return random.random()
```

score_candidates(candidates, bundle): Computes a composite score for candidate products based on their category match with the bundle and business logic scores.

build_bundle(df, anchor_product, purchasing_power, k, C_max, theta, n_categories): Constructs a product bundle starting with the anchor product, adhering to constraints like max size, cost threshold, and category limits.

```
def score_candidates(candidates, bundle):
        candidates = candidates.copy()
        candidates['category_score'] = candidates.apply(lambda p: category_score(p, bundle), axis=1)
        candidates['business_score'] = candidates.apply(business_score, axis=1)
        candidates['score'] = gamma * candidates['category_score'] + delta * candidates['business_score']
        return candidates

def build_bundle(df, anchor_product, purchasing_power, k, C_max, theta, n_categories):

        bundle = [anchor_product]
        current_total_cost = anchor_product['price']
        categories_in_bundle = {anchor_product['category']}

        # Exclude the anchor product from candidates
        candidates = df[df['product_id'] != anchor_product['product_id']]
        candidates = score_candidates(candidates, bundle)
        candidates = candidates[candidates['price'] <= theta * C_max].sort_values(by='score', ascending=False)

        for _, candidate in candidates.iterrows():
            if len(bundle) >= k:
                break
            if candidate['category'] in categories_in_bundle:
                # n_categories = 1 => do not add new categories
                if current_total_cost + candidate['price'] <= C_max:
                    bundle.append(candidate)
                    current_total_cost += candidate['price']
                    # We do not add a new category if n_categories=1
            else:
                # If your logic allows new category, incorporate that here
                pass

        return bundle
```

Purpose: This below snippet generates multiple product bundles for each consultant.

Logic:

1. Iterates through each consultant's products (consultant_id).
2. For each consultant, creates a specified number of bundles (num_bundles).
3. Selects an anchor product using select_anchor_product.
4. Constructs a bundle using build_bundle, considering constraints like size, cost, and category limits.
5. Assigns a unique bundle ID, consultant ID, and flags the anchor product (is_anchor).
6. Appends all products in the bundle to the bundles list with metadata.

```
bundles = []
for consultant_id, group in df.groupby("consultant_id"):
            for bundle_index in range(num_bundles):
                anchor_product = select_anchor_product(group)
                bundle = build_bundle(
                    group,
                    anchor_product,
                    purchasing_power,  # from config
                    k,
                    C_max,
                    theta,
                    n_categories
                )
                unique_bundle_id = f"{consultant_id}_Bundle_{bundle_index+1}"
                for idx, product in enumerate(bundle):
                    product_copy = product.copy()
                    product_copy["consultant_id"] = consultant_id
                    product_copy["bundle_id"] = unique_bundle_id
                    product_copy["is_anchor"] = 1 if idx == 0 else 0
                    bundles.append(product_copy)
```

```
bundles
```

```
purchase_frequency                      261
unique_products                         194
normalized_f                       0.343915
normalized_r                       0.370421
combined_score                     0.359819
category_score                            1
business_score                     0.879163
score                              0.909372
bundle_id                  3441296_Bundle_1
is_anchor                                 0
Name: 20479, dtype: object,
consultant_id                       3441296
product_id                        200078931
category                   CUIDADO PERSONAL
brand                                 ESIKA
price                                 13.27
date                             2023-06-08
frequency                                16
daily_total_spent                     385.9
average_purchasing_power            430.195
recency                                 569
frequency_normalized               0.263158
recency_normalized                  0.75848
total_spent                         7206.76
purchase_frequency                      261
unique_products                         194
normalized_f                       0.343915
normalized_r                        0.75848
combined_score                     0.592654
category_score                            1
business_score                     0.857582
score                              0.893186
bundle_id                  3441296_Bundle_1
is_anchor                                 0
Name: 18988, dtype: object,
consultant_id                       3441296
product_id                        200105195
category                   CUIDADO PERSONAL
brand                                 ESIKA
price                                  7.38
```

```
data = pd.DataFrame(bundles)
data.head()
```

	consultant_id	product_id	category	brand	price	date	frequency	daily_total_spent	average_purchasing_power	recency	...
1793	3441296	200095159	CUIDADO PERSONAL	ESIKA	7.38	2022-12-27	10	263.40	430.195	732	...
4744	3441296	200108807	CUIDADO PERSONAL	ESIKA	7.38	2022-12-27	10	263.40	430.195	732	...
22714	3441296	200095465	CUIDADO PERSONAL	ESIKA	11.90	2023-06-08	16	385.90	430.195	569	...
9107	3441296	200103025	CUIDADO PERSONAL	ESIKA	24.98	2023-08-15	11	156.66	430.195	501	...
20479	3441296	200115451	CUIDADO PERSONAL	ESIKA	18.30	2024-03-20	7	183.50	430.195	283	...

5 rows × 23 columns

Save the DataFrame to a CSV file named ***bundels_data_1000_C.csv***.

```
data.to_csv('bundels_data_1000_C.csv', index=False)
```