In [1]:
```
import pandas as pd

# Replace the file path with your actual file path
file_path = "/Volumes/Untitled/aca2eb7d00ea1a7b8ebd4e68314663af.csv"

# Load the CSV file into a pandas DataFrame
df = pd.read_csv(file_path)

# Display the structure of the DataFrame
print("DataFrame Structure:")
print(df.info())

# Display the first few rows of the DataFrame
print("\nFirst few rows of the DataFrame:")
print(df.head())
```

```
DataFrame Structure:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 536 entries, 0 to 535
Data columns (total 4 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   order_item_id                  536 non-null    int64
 1   shipping_limit_date            536 non-null    object
 2   price                          536 non-null    float64
 3   product_category_name_english  536 non-null    int64
dtypes: float64(1), int64(2), object(1)
memory usage: 16.9+ KB
None

First few rows of the DataFrame:
   order_item_id shipping_limit_date  price  product_category_name_english
0              1          2018-05-14   69.9                             39
1              1          2018-01-09   75.0                             39
2              1          2018-03-15   69.9                             39
3              1          2017-08-11   75.0                             39
4              2          2017-08-11   75.0                             39
```

In [2]:

```
# Differential Evolution Algorithm for Price Optimization

# Igor Mol <igor.mol@makes.ai>

# Differential evolution is employed in pricing optimization to find the
# optimal value to improve sales performance. In our code, the aim consists
# minimizing an objective function that measures the quality of the pricing
# strategy. The optimization process explores the parameter space by evolvir
# a population of candidate solutions over multiple generations. By iterativ
# evaluating and selecting individuals with better objective function values
# differential evolution efficiently navigates the solution space, seeking t
# parameter configuration that maximizes or minimizes the specified criteric
# This optimization approach is particularly useful for fine-tuning pricing
# strategies in a data-driven manner, ensuring optimal performance based on
# given training and testing datasets.

import pandas as pd
import numpy as np
from scipy.optimize import differential_evolution

# split_data:
# This function splits the input features and target variable into training
# testing sets. It shuffles the data and assigns a portion of it to the test

# Parameters:
# - X: Features to be split
# - y: Target variable to be split
# - test_size: Percentage of data to be allocated to the testing set (defaul
# - random_state: Seed for reproducibility (default is 42)

def split_data(X, y, test_size=0.2, random_state=42):
    # Setting the random seed for reproducibility
    np.random.seed(random_state)

    # Creating an array of indices and shuffling them
    indices = np.arange(len(X))
    np.random.shuffle(indices)

    # Calculating the size of the testing set
    test_size = int(test_size * len(X))

    # Splitting indices into test and train sets
    test_indices, train_indices = indices[:test_size], indices[test_size:]

    # Creating training and testing sets for features and target variable
    X_train, X_test = X.iloc[train_indices], X.iloc[test_indices]
    y_train, y_test = y.iloc[train_indices], y.iloc[test_indices]

    # Returning the split datasets
    return X_train, X_test, y_train, y_test


# linear_regression:
# This function performs linear regression and calculates coefficients using
# closed-form solution with regularization (L2 regularization, also known as
# It takes the feature matrix 'X_train', target variable 'y_train', and an o
# regularization parameter 'alpha' as input, and returns the coefficients.

# Parameters:
```

```
# Parameters:
# - X_train: Features of the training set
# - y_train: Target variable of the training set
# - alpha: Regularization parameter (default value is 1e-5)

def linear_regression(X_train, y_train, alpha=1e-5):
    # Adding a column of ones for the intercept
    X_train = np.c_[np.ones(X_train.shape[0]), X_train]

    # Creating an identity matrix with the same number of columns as X_trair
    identity_matrix = np.identity(X_train.shape[1])

    # Calculating coefficients using Ridge regression closed-form solution
    coefficients = np.linalg.inv(X_train.T @ X_train + alpha * identity_matr

    # Returning the calculated coefficients
    return coefficients


# predict:
# This function performs predictions using linear regression coefficients.
# It takes the feature matrix 'X_test' and the coefficients as input and
# returns the predicted values.

# Parameters:
# - X_test: Features of the testing set
# - coefficients: Coefficients obtained from linear regression

def predict(X_test, coefficients):
    # Adding a column of ones for the intercept
    X_test = np.c_[np.ones(X_test.shape[0]), X_test]

    # Calculating predictions using linear regression formula
    predictions = X_test @ coefficients

    # Returning the predicted values
    return predictions


# objective_function():
# This function defines an objective function for optimization.
# The objective is to maximize the negative R-squared value, indicating the
# optimization seeks to minimize the squared residuals in a linear regressic

# Parameters:
# - params: A list containing the parameter(s) to be optimized, in this case
# - X_train: Features of the training set
# - X_test: Features of the testing set
# - y_train: Target variable of the training set
# - y_test: Target variable of the testing set

def objective_function(params, X_train, X_test, y_train, y_test):
    # Extracting the parameter to be optimized (price)
    price = params[0]

    # Creating copies of feature sets for manipulation
    X_train_copy, X_test_copy = X_train.copy(), X_test.copy()

    # Setting the 'price' column in the training set to the optimized value
    X_train_copy['price'] = price
```

```
    X_train_copy['price'] = price

    # Calculating coefficients for linear regression
    coefficients = linear_regression(X_train_copy[['price']].values, y_train

    # Predicting target variable for the testing set
    y_pred = predict(X_test_copy[['price']].values, coefficients)

    # Calculating R-squared value
    residual = y_test.values - y_pred
    r_squared = 1 - (np.sum(residual ** 2) / np.sum((y_test.values - np.mean

    # Returning the negative of R-squared (to be maximized during optimizati
    return -r_squared


# optimize_price:
# This function optimizes a price-related parameter using differential evolu
# It aims to find the optimal value for the parameter by minimizing the obje
# function, which is defined externally.

# Parameters:
# - X_train: Features of the training set
# - X_test: Features of the testing set
# - y_train: Target variable of the training set
# - y_test: Target variable of the testing set
# - lower_bound: Lower bound of the parameter for optimization
# - upper_bound: Upper bound of the parameter for optimization

def optimize_price(X_train, X_test, y_train, y_test, lower_bound, upper_bour
    # Defining a wrapper function for differential evolution
    wrapper_function = lambda params: objective_function(params, X_train, X_

    # Performing differential evolution optimization
    result_de = differential_evolution(wrapper_function, bounds=[(lower_bour

    # Returning the optimized value of the parameter
    return result_de.x[0]


# extract_features_and_target:
# This function extracts features and target from a DataFrame representing s
# It takes a DataFrame 'df' as input, which is expected to have columns like
# 'order_item_id', 'price', and 'product_category_name_english'.
# The target variable 'y' is derived from the month of the 'shipping_limit_d

def extract_features_and_target(df):
    # Extracting features: order_item_id, price, product_category_name_engli
    X = df[['order_item_id', 'price', 'product_category_name_english']]

    # Extracting target: month from the 'shipping_limit_date'
    y = pd.to_datetime(df['shipping_limit_date']).dt.month

    # Returning features (X) and target (y)
    return X, y

def main():
    # Assuming df is your DataFrame
    X, y = extract_features_and_target(df)
```

```
    X_train, X_test, y_train, y_test = split_data(X, y)

    lower_bound = df['price'].min()
    upper_bound = df['price'].max()

    optimal_price_de = optimize_price(X_train, X_test, y_train, y_test, lowe

    print("Optimal Price (Differential Evolution):", optimal_price_de)

if __name__ == "__main__":
    main()
```

```
Optimal Price (Differential Evolution): 70.21336264030145
```

In [ ]: