In [2]:
```python
import pandas as pd

# Load the CSV file into a pandas DataFrame

file_path = "/Volumes/Untitled/aca2eb7d00ea1a7b8ebd4e68314663af.csv"
df = pd.read_csv(file_path)
```

In [3]:
```
# Price Optimization using SVM

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

# SVM (Support Vector Machine) is employed for price optimization to e
# decision-making. In this context, SVM acts as a regression model, le
# patterns in the data to predict optimal prices. The SVM model is tra
# using input features like 'order_item_id,' 'price,' and
# 'product_category_name_english.' The 'shipping_limit_date' is utiliz
# the target variable, with months extracted for simplicity.

# The optimization process involves adjusting the 'price' parameter wi
# the Nelder-Mead method. The SVM regression model is manually impleme
# using gradient descent to find the optimal parameters, considering a
# regularization term (C). The negative R-squared value is minimized d
# optimization, providing a measure of how well the model fits the dat

# The run_optimization function utilizes the scipy.optimize.minimize
# framework to find the optimal 'price' parameter for effective price
# optimization, facilitating better decision support in various scenar

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

# extract_features_target
# This function extracts features and target variable from a DataFrame

# Parameters:
# - df: DataFrame containing necessary columns ('order_item_id', 'pric
#   'product_category_name_english', 'shipping_limit_date').

# Returns:
# - X: DataFrame with features ('order_item_id', 'price',
#   'product_category_name_english').
# - y: Series representing the month extracted from 'shipping_limit_da
#   The target variable.

def extract_features_target(df):
    # Extract features and target variable
    X = df[['order_item_id', 'price', 'product_category_name_english']
    y = pd.to_datetime(df['shipping_limit_date']).dt.month  # Extracti
    return X, y

# split_data:
# This function splits the given features (X) and target variable (y)
# training and testing sets.

# Parameters:
# - X: DataFrame of features.
# - y: Series of the target variable.
# - test_size: Proportion of the data to be used for testing (default
# - random_state: Seed for reproducibility (default is 42).

# Returns:
```

```
# - X_train: DataFrame of features for training set.
# - X_test: DataFrame of features for testing set.
# - y_train: Series of the target variable for training set.
# - y_test: Series of the target variable for testing set.

def split_data(X, y, test_size=0.2, random_state=42):
    # Split the data into training and testing sets
    np.random.seed(random_state)
    msk = np.random.rand(len(df)) < (1 - test_size)
    X_train, X_test = X[msk], X[~msk]
    y_train, y_test = y[msk], y[~msk]
    return X_train, X_test, y_train, y_test

# Data normalization:
# This function normalizes the input features (X) and target variable

# Parameters:
# - X: DataFrame of input features.
# - y: Series representing the target variable.

# Returns:
# - X_normalized: Normalized input features using mean and standard de
# - y_normalized: Normalized target variable using mean and standard d

def normalize_data(X, y):
    # Normalize input features and target variable
    X_normalized = (X - X.mean()) / X.std()
    y_normalized = (y - y.mean()) / y.std()
    return X_normalized, y_normalized

# svm_regression:
# This function performs Support Vector Machine (SVM) regression with
# descent on the given data. It updates the 'price' parameter in the i
# features, normalizes the data, and calculates the negative R-squared

# Parameters:
# - params: List of parameters; only 'price' is extracted for this fun
# - X_train: DataFrame of input features for training.
# - y_train: Series representing the target variable for training.
# - X_test: DataFrame of input features for testing.
# - y_test: Series representing the target variable for testing.
# - C: Regularization parameter for SVM (default is 1.0).

# Returns:
# - Negative R-squared value calculated on the test data.

def svm_regression(params, X_train, y_train, X_test, y_test, C=1.0):
    # Extract parameters
    price = params[0]

    # Update the price in the dataset
    X_train['price'] = price
    X_test['price'] = price

    # Normalize input features and target variable
    X_train_normalized, y_train_normalized = normalize_data(X_train, y
```

```
    X_test_normalized, y_test_normalized = normalize_data(X_test, y_te

    # Perform SVM regression manually with gradient descent
    m, n = X_train_normalized.shape
    X_train_np = np.c_[np.ones(m), X_train_normalized.values]  # Add a

    theta = np.random.rand(n + 1)  # Initialize weights randomly
    learning_rate = 0.01
    num_iterations = 1000

    for _ in range(num_iterations):
        # Calculate gradients
        gradients = -2 * X_train_np.T @ (y_train_normalized.values - X
        theta -= learning_rate * gradients  # Update weights

    # Predict the target variable on the test data
    X_test_np = np.c_[np.ones(X_test_normalized.shape[0]), X_test_norm
    y_pred_normalized = X_test_np @ theta

    # Denormalize the predicted target variable
    y_pred = y_pred_normalized * y_test.std() + y_test.mean()

    # Calculate the negative R-squared value (minimize negative R-squa
    residual_sum_of_squares = np.sum((y_test.values - y_pred)**2)
    total_sum_of_squares = np.sum((y_test.values - np.mean(y_test.valu
    r_squared = 1 - (residual_sum_of_squares / total_sum_of_squares)

    return -r_squared

# run_optimization:
# This function runs optimization using the Nelder-Mead method from
# scipy.optimize.minimize. It optimizes the SVM regression model by mi
# the negative R-squared value.

# Parameters:
# - X_train: DataFrame of input features for training.
# - X_test: DataFrame of input features for testing.
# - y_train: Series representing the target variable for training.
# - y_test: Series representing the target variable for testing.
# - initial_guess: Initial guess for the parameters.

# Returns:
# - Result object containing information about the optimization proces
#   (e.g., optimized parameters, success status, final function value)

def run_optimization(X_train, X_test, y_train, y_test, initial_guess):
    # Run the optimization using scipy.optimize.minimize
    result = minimize(svm_regression, initial_guess, args=(X_train.cop
    return result


def main():

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

```
    # Define the initial guess for the optimal price
    initial_guess = [df['price'].mean()]

    # Run the optimization
    result = run_optimization(X_train, X_test, y_train, y_test, initia

    # Display the optimal price
    optimal_price = result.x[0]
    print("Optimal Price:", optimal_price)

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

```
Optimal Price: 71.35858208955223
```

In [ ]: