In [1]:
```python
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]:
```
# k-NN Algorithm for Price Optimization

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

# The k-Nearest Neighbors algorithm is applied to the price optimization pro
# predicting the optimal price for a product based on historical data. The t
# includes information such as order items, prices, and product categories.
# product, the algorithm calculates Euclidean distances between its features
# historical products. The 'k' nearest neighbors, determined by the smallest
# contribute to predicting the target variable (month from shipping dates) f
# product. This process is repeated for each product in the testing set. The
# price is then determined by minimizing a predefined objective function, sp
# a negative R-squared value. This optimized price aims to enhance the overa
# of the price optimization strategy by maximizing predictive accuracy.

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

# The next function extracts features and target variable from a DataFrame.
# Parameters:
#   - df: DataFrame containing relevant data.
# Returns:
#   - X: Features, a subset of df with columns 'order_item_id', 'price',
#        and 'product_category_name_english'.
#   - y: Target variable, month extracted from 'shipping_limit_date' column.

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

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

    # Returning the features and target variable
    return X, y

# Next function splits input features (X) and target variable (y) into train
# and testing sets using random shuffling.
# Parameters:
#   - X: Input features.
#   - y: Target variable.
# Returns:
#   - X_train: Training set of input features.
#   - X_test: Testing set of input features.
#   - y_train: Training set of target variable.
#   - y_test: Testing set of target variable.

def split_data(X, y):
    # Setting random seed for reproducibility
    np.random.seed(42)

    # Creating array of indices and shuffling them
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)

    # Determining the size of the training set
    train_size = int(0.8 * X.shape[0])
```

```
    # Splitting indices into training and testing sets
    train_indices, test_indices = indices[:train_size], indices[train_size:]

    # Extracting training and testing sets based on the shuffled indices
    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


# The next function performs k-nearest neighbors (KNN) regression on the tes
# instances using the training set.
# Parameters:
#   - X_train: Training set of input features.
#   - y_train: Training set of target variable.
#   - X_test: Testing set of input features.
# Returns:
#   - predictions: Array of predicted target values for each test instance.

def knn(X_train, y_train, X_test):
    # List to store predicted target values for each test instance
    predictions = []

    # Iterating over each test instance
    for _, test_instance in X_test.iterrows():
        # Calculating Euclidean distances between the test instance and all
        # training instances
        distances = np.sqrt(((X_train - test_instance) ** 2).sum(axis=1))

        # Finding indices of the 5 closest neighbors
        closest_indices = np.argsort(distances)[:5]  # 5 neighbors

        # Calculating the mean of target values of the closest neighbors as
        prediction = y_train.iloc[closest_indices].mean()

        # Appending the prediction to the list
        predictions.append(prediction)

    # Converting the list of predictions to a NumPy array
    return np.array(predictions)

# The next function defines the objective (loss) function for optimization.
# Parameters:
#   - params: List of parameters to be optimized. In this case, contains onl
#             the 'price' parameter.
#   - X_train: Training set of input features.
#   - X_test: Testing set of input features.
#   - y_train: Training set of target variable.
#   - y_test: Testing set of target variable.
# Returns:
#   - Negative R-squared value as the objective for optimization.

def objective_function(params, X_train, X_test, y_train, y_test):
    # Extracting 'price' from the parameters
    price = params[0]

    # Setting 'price' for the entire training set
    X_train['price'] = price
```

```
    X_train['price'] = price

    # Getting predictions using KNN
    y_pred = knn(X_train, y_train, X_test)

    # Calculating R-squared as the objective (negative because it is a minim
    r_squared = 1 - ((y_test - y_pred) ** 2).sum() / ((y_test - y_test.mean(

    # Returning the negative R-squared as the objective
    return -r_squared

# optimize_price:
# This function optimizes the 'price' parameter by minimizing the negative
# R-squared using the Nelder-Mead optimization method.
# Parameters:
#   - X_train: Training set of input features.
#   - X_test: Testing set of input features.
#   - y_train: Training set of target variable.
#   - y_test: Testing set of target variable.
#   - initial_guess: Initial guess for the 'price' parameter.
# Returns:
#   - Optimized 'price' parameter.

def optimize_price(X_train, X_test, y_train, y_test, initial_guess):
    # Using the Nelder-Mead optimization method to minimize the objective fu
    result = minimize(objective_function, initial_guess, args=(X_train.copy(

    # Extracting the optimized 'price' parameter from the result
    return result.x[0]


def main():
    # Assuming df is your DataFrame

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

    initial_guess = [df['price'].mean()]

    optimal_price = optimize_price(X_train, X_test, y_train, y_test, initial

    print("Optimal Price:", optimal_price)

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

```
Optimal Price: 71.35858208955223
```

In [ ]: