In [1]:
```
import pandas as pd
import numpy as np
# import dask.dataframe as dd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from datetime import datetime
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from collections import defaultdict
```

In [2]:
```
pd.set_option('display.max_columns', None)

train = pd.read_csv('train_final.csv', low_memory=False)
validation = pd.read_csv('val_set_final.csv')
```

In [3]:
```
train.head()
```

Out[3]:

	Unnamed: 0	date	customer_code	employee_index	country	female	age	new_cust	seniority_
0	0	2015-07-28	664160	N	1	0	0.632653	0	
1	1	2016-01-28	1076784	N	1	0	0.214286	0	
2	2	2015-12-28	672465	N	1	0	0.387755	0	
3	3	2015-10-28	774528	N	1	0	0.397959	0	
4	4	2016-05-28	569598	N	1	0	0.459184	0	

In [4]:
```
validation.head()
```

Out[4]:

	Unnamed: 0.1	Unnamed: 0	date	customer_code	employee_index	country_spain	female	age	fi
0	0	0	2015-11-28	161428	N	1	1	0.744898	
1	1	1	2015-12-28	367478	N	1	1	0.418367	
2	2	2	2015-11-28	643150	N	1	0	0.520408	
3	3	3	2016-04-28	1385854	N	1	0	0.367347	
4	4	4	2015-08-28	495733	N	1	0	0.346939	

In [5]:
```
train = train.drop(columns=['Unnamed: 0'])
validation = validation.drop(columns=['Unnamed: 0', 'Unnamed: 0.1'])
```

Reading into the data

In [6]:
```
drop = ['join_channel', 'province_name', 'employee_index', 'segment', 'total_products'
train = train.drop(columns=drop + ['customer_code_encoded'])
validation = validation.drop(columns=drop + ['payroll_acct.1', 'first_contract_date',
```

In [7]:
```
train = train.rename(columns={'country': 'country_spain'})
```

In [8]:
```
products = ['savings_acct', 'guarantees', 'current_acct', 'derivada_acct', 'payroll_ac
            'junior_acct', 'mas_particular_acct', 'particular_acct', 'particular_plus_
            'short_term_depo', 'medium_term_depo', 'long_term_depo', 'e_acct', 'funds'
            'mortgage', 'pension', 'loans', 'taxes', 'credit_card', 'securities',
            'home_acct', 'payroll_acct', 'pensions_2', 'direct_debt']
```

In [9]:
```
# Check for missing values and duplicates
# train = train.drop_duplicates(subset=['customer_code_encoded'], keep='last') 702609
```

In [10]:
```
print(train.info())
print(validation.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6579717 entries, 0 to 6579716
Data columns (total 42 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   date                    object 
 1   customer_code           int64  
 2   country_spain           int64  
 3   female                  int64  
 4   age                     float64
 5   new_cust                int64  
 6   seniority_in_months     float64
 7   cust_type               int64  
 8   residency_spain         int64  
 9   birth_spain             int64  
 10  active_cust             int64  
 11  income                  float64
 12  savings_acct            int64  
 13  guarantees              int64  
 14  current_acct            int64  
 15  derivada_acct           int64  
 16  payroll_acct            int64  
 17  junior_acct             int64  
 18  mas_particular_acct     int64  
 19  particular_acct         int64  
 20  particular_plus_acct    int64  
 21  short_term_depo         int64  
 22  medium_term_depo        int64  
 23  long_term_depo          int64  
 24  e_acct                  int64  
 25  funds                   int64  
 26  mortgage                int64  
 27  pension                 int64  
 28  loans                   int64  
 29  taxes                   int64  
 30  credit_card             int64  
 31  securities              int64  
 32  home_acct               int64  
 33  pensions_2              int64  
 34  direct_debt             int64  
 35  01 - TOP                int64  
 36  02 - PARTICULARES       int64  
 37  03 - UNIVERSITARIO      int64  
 38  join_channel_encoded    float64
 39  province_name_encoded   float64
 40  employee_index_encoded  float64
 41  income_to_age           float64
dtypes: float64(7), int64(34), object(1)
memory usage: 2.1+ GB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2100202 entries, 0 to 2100201
Data columns (total 42 columns):
 #   Column                  Dtype  
---  ------                  -----  
 0   date                    object 
 1   customer_code           int64  
 2   country_spain           int64  
 3   female                  int64  
 4   age                     float64
```

```
 5   new_cust                int64
 6   seniority_in_months     float64
 7   cust_type               int64
 8   residency_spain         int64
 9   birth_spain             int64
 10  active_cust             int64
 11  income                  float64
 12  savings_acct            int64
 13  guarantees              int64
 14  current_acct            int64
 15  derivada_acct           int64
 16  payroll_acct            int64
 17  junior_acct             int64
 18  mas_particular_acct     int64
 19  particular_acct         int64
 20  particular_plus_acct    int64
 21  short_term_depo         int64
 22  medium_term_depo        int64
 23  long_term_depo          int64
 24  e_acct                  int64
 25  funds                   int64
 26  mortgage                int64
 27  pension                 int64
 28  loans                   int64
 29  taxes                   int64
 30  credit_card             int64
 31  securities              int64
 32  home_acct               int64
 33  pensions_2              int64
 34  direct_debt             int64
 35  01 - TOP                int64
 36  02 - PARTICULARES       int64
 37  03 - UNIVERSITARIO      int64
 38  join_channel_encoded    float64
 39  province_name_encoded   float64
 40  employee_index_encoded  float64
 41  income_to_age           float64
dtypes: float64(7), int64(34), object(1)
memory usage: 673.0+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2100202 entries, 0 to 2100201
Data columns (total 42 columns):
 #   Column                  Dtype
---  ------                  -----
 0   date                    object
 1   customer_code           int64
 2   country_spain           int64
 3   female                  int64
 4   age                     float64
 5   new_cust                int64
 6   seniority_in_months     float64
 7   cust_type               int64
 8   residency_spain         int64
 9   birth_spain             int64
 10  active_cust             int64
 11  income                  float64
 12  savings_acct            int64
 13  guarantees              int64
 14  current_acct            int64
```

```
 15  derivada_acct           int64
 16  payroll_acct            int64
 17  junior_acct             int64
 18  mas_particular_acct     int64
 19  particular_acct         int64
 20  particular_plus_acct    int64
 21  short_term_depo         int64
 22  medium_term_depo        int64
 23  long_term_depo          int64
 24  e_acct                  int64
 25  funds                   int64
 26  mortgage                int64
 27  pension                 int64
 28  loans                   int64
 29  taxes                   int64
 30  credit_card             int64
 31  securities              int64
 32  home_acct               int64
 33  pensions_2              int64
 34  direct_debt             int64
 35  01 - TOP                int64
 36  02 - PARTICULARES       int64
 37  03 - UNIVERSITARIO      int64
 38  join_channel_encoded    float64
 39  province_name_encoded   float64
 40  employee_index_encoded  float64
 41  income_to_age           float64
dtypes: float64(7), int64(34), object(1)
memory usage: 673.0+ MB
None
```

Pre-processing

In [11]:
```
# Store customer IDs separately
customer_ids = train['customer_code'].values
```

In [12]:
```
# Split data into features (X) and labels (y)
X_train = train.drop(['customer_code', 'date'] + products, axis=1)
y_train = train[products]

X_val = validation.drop(['customer_code', 'date'] + products, axis=1)
y_val = validation[products]
```

In [13]:
```
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)

print("Shape of X_val:", X_val.shape)
print("Shape of y_val:", y_val.shape)
```

```
Shape of X_train: (6579717, 17)
Shape of y_train: (6579717, 24)
Shape of X_val: (2100202, 17)
Shape of y_val: (2100202, 24)
```

In [14]:
```
# Store already active products for each customer
already_active = {}
for row in train[['customer_code'] + products].values:
```

```
    row = list(row)
    customer_id = row.pop(0)
    active_products = [product for product, value in zip(products, row) if value > 0]
    already_active[customer_id] = active_products
```

In [15]:
```
# Hyperparameters
hyperparameter_variations = [
    {'C': 0.01, 'solver': 'liblinear', 'max_iter': 100},
    {'C': 1, 'solver': 'lbfgs', 'max_iter': 200},
    {'C': 10, 'solver': 'liblinear', 'max_iter': 300},
]
```

In [16]:
```
# Dictionary to store results
results = {}
id_preds = defaultdict(list)
```

Training

In [18]:
```
# Train and evaluate each variation
for i, params in enumerate(hyperparameter_variations):
    print(f"\nTraining variation {i + 1} with parameters: {params}")

    # Create logistic regression model with current parameters
    clf = LogisticRegression(**params)

    # Fit the model to the training data for each product
    for product in products:
        # Ensure target is a 1D array/Series
        target = y_train[product]
        # if target.ndim > 1:
        #     target = target.iloc[:, 0]  # Convert to 1D by selecting the first colum

        # Train the logistic regression model
        clf.fit(X_train, target)

        # Predict probabilities on the validation set
        y_val_pred = clf.predict_proba(X_val)[:, 1]  # Take probabilities for the posi

        # Check shape of y_val[product] and y_val_pred for debugging
        print(f"Shape of y_val[product]: {y_val[product].shape}")
        print(f"Shape of y_val_pred: {y_val_pred.shape}")

        # Calculate ROC AUC score
        roc_auc = roc_auc_score(y_val[product], y_val_pred)  # Ensure both are 1D arra

        # Store results for the product
        results.setdefault(f'Variation {i + 1}', {})[product] = roc_auc
        print(f"ROC AUC for product '{product}': {roc_auc}")
```

```
Training variation 1 with parameters: {'C': 0.01, 'solver': 'liblinear', 'max_iter':
100}
Shape of y_val[product]: (2100202,)
Shape of y_val_pred: (2100202,)
ROC AUC for product 'savings_acct': 0.8000330611127483
Shape of y_val[product]: (2100202,)
Shape of y_val_pred: (2100202,)
ROC AUC for product 'guarantees': 0.5226201579695183
Shape of y_val[product]: (2100202,)
Shape of y_val_pred: (2100202,)
ROC AUC for product 'current_acct': 0.7465143682243353
Shape of y_val[product]: (2100202,)
Shape of y_val_pred: (2100202,)
ROC AUC for product 'derivada_acct': 0.8599193987964853
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[18], line 16
     11 target = y_train[product]
     12 # if target.ndim > 1:
     13 #     target = target.iloc[:, 0]  # Convert to 1D by selecting the first colu
mn
     14
     15 # Train the logistic regression model
---> 16 clf.fit(X_train, target)
     18 # Predict probabilities on the validation set
     19 y_val_pred = clf.predict_proba(X_val)[:, 1]  # Take probabilities for the pos
itive class

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\base.py:1151, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args,
**kwargs)
   1144     estimator._validate_params()
   1146 with config_context(
   1147     skip_parameter_validation=(
   1148         prefer_skip_nested_validation or global_skip_validation
   1149     )
   1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\linear_model\_logistic.py:1207, in LogisticRegression.fit(self, X, y, sample_weight)
   1204 else:
   1205     _dtype = [np.float64, np.float32]
-> 1207 X, y = self._validate_data(
   1208     X,
   1209     y,
   1210     accept_sparse="csr",
   1211     dtype=_dtype,
   1212     order="C",
   1213     accept_large_sparse=solver not in ["liblinear", "sag", "saga"],
   1214 )
   1215 check_classification_targets(y)
   1216 self.classes_ = np.unique(y)

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\base.py:621, in BaseEstimator._validate_data(self, X, y, reset, validate_separately,
cast_to_ndarray, **check_params)
    619         y = check_array(y, input_name="y", **check_y_params)
    620     else:
--> 621         X, y = check_X_y(X, y, **check_params)
    622     out = X, y
    624 if not no_val_X and check_params.get("ensure_2d", True):

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\utils\validation.py:1163, in check_X_y(X, y, accept_sparse, accept_large_sparse, dty
pe, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samp
les, ensure_min_features, y_numeric, estimator)
   1143     raise ValueError(
   1144         f"{estimator_name} requires y to be passed, but the target y is None"
   1145     )
   1147 X = check_array(
   1148     X,
   1149     accept_sparse=accept_sparse,
   (...)
```

```
   1160     input_name="X",
   1161 )
-> 1163 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, estimator=est
imator)
   1165 check_consistent_length(X, y)
   1167 return X, y

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\utils\validation.py:1184, in _check_y(y, multi_output, y_numeric, estimator)
   1182 else:
   1183     estimator_name = _check_estimator_name(estimator)
-> 1184     y = column_or_1d(y, warn=True)
   1185     _assert_all_finite(y, input_name="y", estimator_name=estimator_name)
   1186     _ensure_no_complex_data(y)

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\utils\validation.py:1245, in column_or_1d(y, dtype, warn)
   1234         warnings.warn(
   1235             (
   1236                 "A column-vector y was passed when a 1d array was"
   (...)
   1241             stacklevel=2,
   1242         )
   1243     return _asarray_with_order(xp.reshape(y, (-1,)), order="C", xp=xp)
-> 1245 raise ValueError(
   1246     "y should be a 1d array, got an array of shape {} instead.".format(shape)
   1247 )

ValueError: y should be a 1d array, got an array of shape (6579717, 2) instead.
```

In [19]:
```
# Print comparison of results
print("\nComparison of Variations:")
for variation, product_results in results.items():
    for product, roc_auc in product_results.items():
        print(f"{variation} - Product: {product}, ROC AUC: {roc_auc}")
```

```
Comparison of Variations:
Variation 1 - Product: savings_acct, ROC AUC: 0.8000330611127483
Variation 1 - Product: guarantees, ROC AUC: 0.5226201579695183
Variation 1 - Product: current_acct, ROC AUC: 0.7465143682243353
Variation 1 - Product: derivada_acct, ROC AUC: 0.8599193987964853
```

Generate product recommendations

In [21]:
```
train_preds = {}
for customer_id in customer_ids:
    preds = []
    for i, params in enumerate(hyperparameter_variations):
        clf = LogisticRegression(**params)
        for product in products:
            clf.fit(X_train, y_train[product])
            p = clf.predict_proba(X_train)[:, 1]  # Get prediction probabilities for t
            preds.append((product, p[customer_id]))

    # Sort the products based on predicted probabilities, excluding already active pro
    recommended_products = sorted(preds, key=lambda x: x[1], reverse=True)
    recommended_products = [prod for prod, prob in recommended_products if prod not in
```

```
    # Get top 7 product recommendations
    train_preds[customer_id] = recommended_products[:7]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[21], line 7
      5 clf = LogisticRegression(**params)
      6 for product in products:
----> 7     clf.fit(X_train, y_train[product])
      8     p = clf.predict_proba(X_train)[:, 1]  # Get prediction probabilities for
the entire dataset
      9     preds.append((product, p[customer_id]))

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\base.py:1151, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args,
**kwargs)
   1144     estimator._validate_params()
   1146 with config_context(
   1147     skip_parameter_validation=(
   1148         prefer_skip_nested_validation or global_skip_validation
   1149     )
   1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\linear_model\_logistic.py:1207, in LogisticRegression.fit(self, X, y, sample_weight)
   1204 else:
   1205     _dtype = [np.float64, np.float32]
-> 1207 X, y = self._validate_data(
   1208     X,
   1209     y,
   1210     accept_sparse="csr",
   1211     dtype=_dtype,
   1212     order="C",
   1213     accept_large_sparse=solver not in ["liblinear", "sag", "saga"],
   1214 )
   1215 check_classification_targets(y)
   1216 self.classes_ = np.unique(y)

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\base.py:621, in BaseEstimator._validate_data(self, X, y, reset, validate_separately,
cast_to_ndarray, **check_params)
    619         y = check_array(y, input_name="y", **check_y_params)
    620     else:
--> 621         X, y = check_X_y(X, y, **check_params)
    622     out = X, y
    624 if not no_val_X and check_params.get("ensure_2d", True):

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\utils\validation.py:1163, in check_X_y(X, y, accept_sparse, accept_large_sparse, dty
pe, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samp
les, ensure_min_features, y_numeric, estimator)
   1143     raise ValueError(
   1144         f"{estimator_name} requires y to be passed, but the target y is None"
   1145     )
   1147 X = check_array(
   1148     X,
   1149     accept_sparse=accept_sparse,
   (...)
   1160     input_name="X",
   1161 )
-> 1163 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric, estimator=est
imator)
```

```
   1165 check_consistent_length(X, y)
   1167 return X, y

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\utils\validation.py:1184, in _check_y(y, multi_output, y_numeric, estimator)
   1182 else:
   1183     estimator_name = _check_estimator_name(estimator)
-> 1184     y = column_or_1d(y, warn=True)
   1185     _assert_all_finite(y, input_name="y", estimator_name=estimator_name)
   1186     _ensure_no_complex_data(y)

File c:\Users\MARIA\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\utils\validation.py:1245, in column_or_1d(y, dtype, warn)
   1234         warnings.warn(
   1235             (
   1236                 "A column-vector y was passed when a 1d array was"
   (...)
   1241             stacklevel=2,
   1242         )
   1243     return _asarray_with_order(xp.reshape(y, (-1,)), order="C", xp=xp)
-> 1245 raise ValueError(
   1246     "y should be a 1d array, got an array of shape {} instead.".format(shape)
   1247 )

ValueError: y should be a 1d array, got an array of shape (6579717, 2) instead.
```