Modeling Approaches

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, f1_score, confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
from collections import defaultdict
# Week 7
# from sklearn.model_selection import cross_val_score
# from sklearn.metrics import accuracy_score
# import xgboost as xgb

# Week 8
from sklearn.ensemble import RandomForestClassifier
```

Reading into the data

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	

Pre-processing

Changing columns name and dropping columns so both datasets are the same

In [5]:
```
train = train.rename(columns={'country': 'country_spain'})
```

In [6]:
```
train = train.drop(columns=['Unnamed: 0'])
validation = validation.drop(columns=['Unnamed: 0', 'Unnamed: 0.1'])
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',
```

Setting products we want to predict

In [7]:
```
products = ['savings_acct', 'guarantees', 'current_acct',
       'derivada_acct', 'payroll_acct', 'junior_acct', 'mas_particular_acct',
       'particular_acct', 'particular_plus_acct', 'short_term_depo',
       'medium_term_depo', 'long_term_depo', 'e_acct', 'funds', 'mortgage',
       'pension', 'loans', 'taxes', 'credit_card', 'securities', 'home_acct',
       'pensions_2', 'direct_debt']
```

Dropping duplicates on customer code column since the last instance will show all the products a client has

In [8]:
```
train = train.drop_duplicates(subset=['customer_code'], keep='last')
validation = validation.drop_duplicates(subset=['customer_code'], keep='last')

# Removing customers from validation set that appear in training set
validation = validation[~validation['customer_code'].isin(train['customer_code'])]
```

In [9]:
```
print(train.info())
print(validation.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 706866 entries, 39288 to 6579716
Data columns (total 42 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   date                   706866 non-null  object 
 1   customer_code          706866 non-null  int64  
 2   country_spain          706866 non-null  int64  
 3   female                 706866 non-null  int64  
 4   age                    706866 non-null  float64
 5   new_cust               706866 non-null  int64  
 6   seniority_in_months    706866 non-null  float64
 7   cust_type              706866 non-null  int64  
 8   residency_spain        706866 non-null  int64  
 9   birth_spain            706866 non-null  int64  
 10  active_cust            706866 non-null  int64  
 11  income                 706866 non-null  float64
 12  savings_acct           706866 non-null  int64  
 13  guarantees             706866 non-null  int64  
 14  current_acct           706866 non-null  int64  
 15  derivada_acct          706866 non-null  int64  
 16  payroll_acct           706866 non-null  int64  
 17  junior_acct            706866 non-null  int64  
 18  mas_particular_acct    706866 non-null  int64  
 19  particular_acct        706866 non-null  int64  
 20  particular_plus_acct   706866 non-null  int64  
 21  short_term_depo        706866 non-null  int64  
 22  medium_term_depo       706866 non-null  int64  
 23  long_term_depo         706866 non-null  int64  
 24  e_acct                 706866 non-null  int64  
 25  funds                  706866 non-null  int64  
 26  mortgage               706866 non-null  int64  
 27  pension                706866 non-null  int64  
 28  loans                  706866 non-null  int64  
 29  taxes                  706866 non-null  int64  
 30  credit_card            706866 non-null  int64  
 31  securities             706866 non-null  int64  
 32  home_acct              706866 non-null  int64  
 33  pensions_2             706866 non-null  int64  
 34  direct_debt            706866 non-null  int64  
 35  01 - TOP               706866 non-null  int64  
 36  02 - PARTICULARES      706866 non-null  int64  
 37  03 - UNIVERSITARIO     706866 non-null  int64  
 38  join_channel_encoded   706866 non-null  float64
 39  province_name_encoded  706866 non-null  float64
 40  employee_index_encoded 706866 non-null  float64
 41  income_to_age          706866 non-null  float64
dtypes: float64(7), int64(34), object(1)
memory usage: 231.9+ MB
None
<class 'pandas.core.frame.DataFrame'>
Index: 200333 entries, 51 to 2100200
Data columns (total 42 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   date                   200333 non-null  object 
 1   customer_code          200333 non-null  int64  
 2   country_spain          200333 non-null  int64  
 3   female                 200333 non-null  int64  
 4   age                    200333 non-null  float64
```

```
 5   new_cust                200333 non-null  int64
 6   seniority_in_months     200333 non-null  float64
 7   cust_type               200333 non-null  int64
 8   residency_spain         200333 non-null  int64
 9   birth_spain             200333 non-null  int64
 10  active_cust             200333 non-null  int64
 11  income                  200333 non-null  float64
 12  savings_acct            200333 non-null  int64
 13  guarantees              200333 non-null  int64
 14  current_acct            200333 non-null  int64
 15  derivada_acct           200333 non-null  int64
 16  payroll_acct            200333 non-null  int64
 17  junior_acct             200333 non-null  int64
 18  mas_particular_acct     200333 non-null  int64
 19  particular_acct         200333 non-null  int64
 20  particular_plus_acct    200333 non-null  int64
 21  short_term_depo         200333 non-null  int64
 22  medium_term_depo        200333 non-null  int64
 23  long_term_depo          200333 non-null  int64
 24  e_acct                  200333 non-null  int64
 25  funds                   200333 non-null  int64
 26  mortgage                200333 non-null  int64
 27  pension                 200333 non-null  int64
 28  loans                   200333 non-null  int64
 29  taxes                   200333 non-null  int64
 30  credit_card             200333 non-null  int64
 31  securities              200333 non-null  int64
 32  home_acct               200333 non-null  int64
 33  pensions_2              200333 non-null  int64
 34  direct_debt             200333 non-null  int64
 35  01 - TOP                200333 non-null  int64
 36  02 - PARTICULARES       200333 non-null  int64
 37  03 - UNIVERSITARIO      200333 non-null  int64
 38  join_channel_encoded    200333 non-null  float64
 39  province_name_encoded   200333 non-null  float64
 40  employee_index_encoded  200333 non-null  float64
 41  income_to_age           200333 non-null  float64
dtypes: float64(7), int64(34), object(1)
memory usage: 65.7+ MB
None
```

Defining our Xs and Ys

In [10]:
```
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 [11]:
```
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: (706866, 17)
Shape of y_train: (706866, 23)
Shape of X_val: (200333, 17)
Shape of y_val: (200333, 23)
```

Week 8 - Develop Third Modeling Approach

Setting hyperparameters

In [12]:
```
hyperparameter_variations = [
    {'n_estimators': 50, 'max_depth': 10, 'min_samples_leaf': 5, 'random_state':17},
    {'n_estimators': 100, 'max_depth': 15, 'min_samples_leaf': 3, 'random_state':17},
    {'n_estimators': 150, 'max_depth': 20, 'min_samples_leaf': 2, 'random_state':17}
]
```

Creating funtion to train and evaluate the Random Forest model

This week we will keep the same metrics used last week -ROC AUC, F1 Score and Confusion Matrix, so we can compare in the end which model is better

In [13]:
```
def train_and_evaluate_rf(X_train, y_train, X_val, y_val, params):
    metrics = {}
    for product in y_train.columns:
        # Train the model
        model = RandomForestClassifier(**params, n_jobs=-1)
        model.fit(X_train, y_train[product])

        # Make predictions
        y_train_pred = model.predict(X_train)
        y_val_pred = model.predict(X_val)

        # Calculate metrics
        metrics[product] = {
            'train': calculate_metrics(y_train[product], y_train_pred),
            'val': calculate_metrics(y_val[product], y_val_pred)
        }

    return metrics

def calculate_metrics(y_true, y_pred):
    return {
        'ROC AUC': roc_auc_score(y_true, y_pred),
        'F1 Score': f1_score(y_true, y_pred),
        'Accuracy': accuracy_score(y_true, y_pred),
        'Confusion Matrix': confusion_matrix(y_true, y_pred)
    }
```

In [ ]:
```
# Train and evaluate models for each variation
results = []
for i, params in enumerate(hyperparameter_variations):
    metrics = train_and_evaluate_rf(X_train, y_train, X_val, y_val, params)

    avg_train_roc_auc = np.mean([metrics[product]['train']['ROC AUC'] for product in y
    avg_train_f1 = np.mean([metrics[product]['train']['F1 Score'] for product in y_tra
    avg_train_accuracy = np.mean([metrics[product]['train']['Accuracy'] for product ir

    avg_val_roc_auc = np.mean([metrics[product]['val']['ROC AUC'] for product in y_tra
    avg_val_f1 = np.mean([metrics[product]['val']['F1 Score'] for product in y_train.c
    avg_val_accuracy = np.mean([metrics[product]['val']['Accuracy'] for product in y_t
```

```
    results.append({
        'Variation': f'Variation {i+1}',
        'Train ROC AUC': avg_train_roc_auc,
        'Train F1 Score': avg_train_f1,
        'Train Accuracy': avg_train_accuracy,
        'Val ROC AUC': avg_val_roc_auc,
        'Val F1 Score': avg_val_f1,
        'Val Accuracy': avg_val_accuracy
    })
```

Creating a table to summarize results and define which variation is the best

In [15]:

```
results_df = pd.DataFrame(results)
print("Results Table:")
print(results_df.to_string(index=False))
```

```
Results Table:
  Variation  Train ROC AUC  Train F1 Score  Train Accuracy  Val ROC AUC  Val F1 Score
Val Accuracy
Variation 1       0.550031        0.138405        0.959554     0.541662      0.112408
0.971892
Variation 2       0.565750        0.184944        0.961033     0.550482      0.138092
0.971856
Variation 3       0.591426        0.260752        0.964707     0.554772      0.152463
0.971651
```

In [25]:

```
# Identify the best model
best_model_idx = results_df['Val ROC AUC'].idxmax()
best_model = results_df.loc[best_model_idx]
print("\nBest Model:")
print(best_model.to_string())
```

```
Best Model:
Variation         Variation 3
Train ROC AUC        0.591426
Train F1 Score       0.260752
Train Accuracy       0.964707
Val ROC AUC          0.554772
Val F1 Score         0.152463
Val Accuracy         0.971651
```

Creating a table to see the results in a easier to interpret way

In [26]:

```
# Print detailed metrics for each product
for product in y_train.columns:
    print(f"Product: {product}")
    print(f"Train - ROC AUC: {metrics[product]['train']['ROC AUC']:.4f}, F1 Score: {me
    print(f"Val - ROC AUC: {metrics[product]['val']['ROC AUC']:.4f}, F1 Score: {metric
    print(f"Confusion Matrix (Validation):\n{metrics[product]['val']['Confusion Matrix
```

```
Product: savings_acct
Train - ROC AUC: 0.5000, F1 Score: 0.0000
Val - ROC AUC: 0.5000, F1 Score: 0.0000
Confusion Matrix (Validation):
[[200325      0]
 [     8      0]]

Product: guarantees
Train - ROC AUC: 0.5000, F1 Score: 0.0000
Val - ROC AUC: 0.5000, F1 Score: 0.0000
Confusion Matrix (Validation):
[[200331      0]
 [     2      0]]

Product: current_acct
Train - ROC AUC: 0.7536, F1 Score: 0.8195
Val - ROC AUC: 0.7306, F1 Score: 0.8072
Confusion Matrix (Validation):
[[ 48475  29677]
 [ 19423 102758]]

Product: derivada_acct
Train - ROC AUC: 0.5000, F1 Score: 0.0000
Val - ROC AUC: 0.5000, F1 Score: 0.0000
Confusion Matrix (Validation):
[[200289      0]
 [    44      0]]

Product: payroll_acct
Train - ROC AUC: 0.5469, F1 Score: 0.1710
Val - ROC AUC: 0.5132, F1 Score: 0.0517
Confusion Matrix (Validation):
[[193931    188]
 [  6044    170]]

Product: junior_acct
Train - ROC AUC: 0.9982, F1 Score: 0.9885
Val - ROC AUC: 0.9703, F1 Score: 0.9331
Confusion Matrix (Validation):
[[199129     85]
 [    66   1053]]

Product: mas_particular_acct
Train - ROC AUC: 0.5690, F1 Score: 0.2426
Val - ROC AUC: 0.5123, F1 Score: 0.0475
Confusion Matrix (Validation):
[[197453     73]
 [  2737     70]]

Product: particular_acct
Train - ROC AUC: 0.7095, F1 Score: 0.5502
Val - ROC AUC: 0.6319, F1 Score: 0.3857
Confusion Matrix (Validation):
[[182885   2385]
 [ 10894   4169]]

Product: particular_plus_acct
Train - ROC AUC: 0.5536, F1 Score: 0.1935
Val - ROC AUC: 0.5140, F1 Score: 0.0541
Confusion Matrix (Validation):
```

```
[[195730    127]
 [  4348    128]]

Product: short_term_depo
Train - ROC AUC: 0.6368, F1 Score: 0.4289
Val - ROC AUC: 0.5520, F1 Score: 0.1232
Confusion Matrix (Validation):
[[199815    194]
 [   290     34]]

Product: medium_term_depo
Train - ROC AUC: 0.5000, F1 Score: 0.0000
Val - ROC AUC: 0.5000, F1 Score: 0.0000
Confusion Matrix (Validation):
[[200152      0]
 [   181      0]]

Product: long_term_depo
Train - ROC AUC: 0.7055, F1 Score: 0.5533
Val - ROC AUC: 0.5961, F1 Score: 0.2877
Confusion Matrix (Validation):
[[194894    778]
 [  3747    914]]

Product: e_acct
Train - ROC AUC: 0.6412, F1 Score: 0.4283
Val - ROC AUC: 0.5716, F1 Score: 0.2337
Confusion Matrix (Validation):
[[189518   1255]
 [  8129   1431]]

Product: funds
Train - ROC AUC: 0.5372, F1 Score: 0.1384
Val - ROC AUC: 0.5052, F1 Score: 0.0206
Confusion Matrix (Validation):
[[198221     25]
 [  2065     22]]

Product: mortgage
Train - ROC AUC: 0.5049, F1 Score: 0.0195
Val - ROC AUC: 0.5000, F1 Score: 0.0000
Confusion Matrix (Validation):
[[199812      0]
 [   521      0]]

Product: pension
Train - ROC AUC: 0.5088, F1 Score: 0.0346
Val - ROC AUC: 0.5000, F1 Score: 0.0000
Confusion Matrix (Validation):
[[199454      0]
 [   879      0]]

Product: loans
Train - ROC AUC: 0.6591, F1 Score: 0.4822
Val - ROC AUC: 0.5872, F1 Score: 0.2871
Confusion Matrix (Validation):
[[199981     14]
 [   279     59]]

Product: taxes
```

```
Train - ROC AUC: 0.5300, F1 Score: 0.1133
Val - ROC AUC: 0.5024, F1 Score: 0.0099
Confusion Matrix (Validation):
[[195328     68]
 [  4912     25]]

Product: credit_card
Train - ROC AUC: 0.5211, F1 Score: 0.0811
Val - ROC AUC: 0.5037, F1 Score: 0.0152
Confusion Matrix (Validation):
[[195901     68]
 [  4330     34]]

Product: securities
Train - ROC AUC: 0.5414, F1 Score: 0.1529
Val - ROC AUC: 0.5011, F1 Score: 0.0046
Confusion Matrix (Validation):
[[197270     11]
 [  3045      7]]

Product: home_acct
Train - ROC AUC: 0.5000, F1 Score: 0.0000
Val - ROC AUC: 0.5000, F1 Score: 0.0000
Confusion Matrix (Validation):
[[200015      0]
 [   318      0]]

Product: pensions_2
Train - ROC AUC: 0.5496, F1 Score: 0.1801
Val - ROC AUC: 0.5136, F1 Score: 0.0534
Confusion Matrix (Validation):
[[193412    197]
 [  6534    190]]

Product: direct_debt
Train - ROC AUC: 0.6363, F1 Score: 0.4195
Val - ROC AUC: 0.5545, F1 Score: 0.1918
Confusion Matrix (Validation):
[[181671   2561]
 [ 14121   1980]]
```

Generate product recommendations

Here we want to visualize the product recommendations for each customer

We will generate the product recommendations using Variation 3 - that had the best performance between all variations

In [21]:
```
best_variation = 'Variation 3'
best_params = hyperparameter_variations[int(best_variation.split()[-1]) - 1]

# Train models and generate predictions
product_models = {}
for product in y_train.columns:
    clf = RandomForestClassifier(**best_params, n_jobs=-1)
    clf.fit(X_train, y_train[product])
    product_models[product] = clf
```

```
train_preds = {}
for product in y_train.columns:
    proba = product_models[product].predict_proba(X_train)[:, 1]

    for customer_id, prob in zip(train['customer_code'], proba):
        if customer_id not in train_preds:
            train_preds[customer_id] = []
        train_preds[customer_id].append((product, prob))
```

We want to make sure that we are not recommending a product that the customer already own, so we will store the products that customers already have

In [22]:
```
def get_active_products(customer_data):
    return set(product for product in y_train.columns if customer_data[product] > 0)
```

We will sort the recommended products by the probability of a client getting it and we will get the top 7 recommendations

In [23]:
```
# Sort and filter recommendations for each customer
for customer_id in train_preds:
    # Sort by probability
    sorted_prods = sorted(train_preds[customer_id], key=lambda x: x[1], reverse=True)
    customer_data = train[train['customer_code'] == customer_id].iloc[0]

    # Filter out products already owned
    active_products = get_active_products(customer_data)
    recommended_products = [prod for prod, _ in sorted_prods if prod not in active_pro

    # Get top 7
    train_preds[customer_id] = recommended_products[:7]

print("Example Recommendations:")
for customer_id in list(train_preds.keys())[:5]:  # First 5 customers
    print(f"Customer {customer_id}: {train_preds[customer_id]}")
```

```
Example Recommendations:
Customer 1225385: ['current_acct', 'direct_debt', 'pensions_2', 'taxes', 'credit_car
d', 'payroll_acct', 'long_term_depo']
Customer 1358829: ['direct_debt', 'pensions_2', 'payroll_acct', 'taxes', 'e_acct', 'c
redit_card', 'long_term_depo']
Customer 1436539: ['direct_debt', 'pensions_2', 'payroll_acct', 'securities', 'short_
term_depo', 'e_acct', 'long_term_depo']
Customer 1448049: ['current_acct', 'taxes', 'pensions_2', 'payroll_acct', 'junior_acc
t', 'mas_particular_acct', 'securities']
Customer 1396837: ['pensions_2', 'direct_debt', 'payroll_acct', 'e_acct', 'mas_partic
ular_acct', 'credit_card', 'junior_acct']
```

Lastly, we want to identify which product has been recommended the most and least in the model

In [24]:
```
product_recommendation_counts = {product: sum(1 for recs in train_preds.values() if pr
most_recommended = max(product_recommendation_counts, key=product_recommendation_count
least_recommended = min(product_recommendation_counts, key=product_recommendation_cour

print(f"Most frequently recommended product: {most_recommended} ({product_recommendati
print(f"Least frequently recommended product: {least_recommended} ({product_recommenda
```

```
Most frequently recommended product: taxes (623341 times)
Least frequently recommended product: guarantees (1401 times)
```