Mounting GDrive directory

In [1]:
```
from google.colab import drive
drive.mount('/content/gdrive')
%cd '/content/gdrive/MyDrive/Colab Notebooks/UoA_MSDS/Course_8/Capstone1_Ad_
```

```
Mounted at /content/gdrive
/content/gdrive/MyDrive/Colab Notebooks/UoA_MSDS/Course_8/Capstone1_Ad_Campa
ign_Recommender
```

Import libraries

In [2]:
```
import os
import csv

from IPython.core.display import display, HTML

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')



from sklearn import config_context
from sklearn.compose import ColumnTransformer
from sklearn import preprocessing
from sklearn.pipeline import make_pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from xgboost import XGBClassifier
from sklearn.multiclass import OneVsRestClassifier

from sklearn import model_selection
from sklearn.model_selection import RandomizedSearchCV
```

Reading data

In [3]:
```
# Load into dataframes, disble quote parsing inside strings when loading
# Ref: https://stackoverflow.com/a/29857126

# Scenario 1: devices with event data
segment_1_train = pd.read_csv('data/train/segment_1_train.csv')
segment_1_test = pd.read_csv('data/test/segment_1_test.csv')
```

```
# Scenario 2: devices without event data
segment_2_train = pd.read_csv('data/train/segment_2_train.csv')
segment_2_test = pd.read_csv('data/test/segment_2_test.csv')
```

Subtask 5: Model building - Different models (Stacking)

1. Common functions for training models

In [4]:
```
from types import SimpleNamespace

# Generate stacking models
def gen_stacking_models():
    # Classifiers, using One-Vs-Rest strategy if target is multiclass proble
    clf1 = LogisticRegression(multi_class='ovr', random_state=0)
    clf2 = RandomForestClassifier(n_estimators=25, random_state=0) # Use sma
    xgb = XGBClassifier(random_state=0)

    # Stacking model
    sclf = StackingClassifier(
        estimators=[
            ('LogisticRegression', clf1),
            ('RandomForest', clf2),
        ],
        final_estimator=xgb,
        stack_method='predict_proba',
        passthrough=True,
        cv=5,
        n_jobs=-1,
    )
    return SimpleNamespace(
        clf1=clf1,
        clf2=clf2,
        xgb=xgb,
        sclf=sclf,
    )
```

In [5]:
```
# Do CV to check performance of stacking model vs base models
def cross_val_check(stacking_models, X_train, y_train):
    for clf, label in zip(
        [
            stacking_models.clf1,
            stacking_models.clf2,
            stacking_models.sclf,
        ],
        [
            'LogisticRegression',
            'RandomForest',
            'StackingClassifier',
        ]
    ):
```

```
        scores = model_selection.cross_val_score(
            clf,
            X_train,
            y_train,
            cv=3, # Just do only 3 folds as we are limited in power
            scoring='roc_auc_ovr', # Scoring for both single-class and multi
        )
        print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.st
```

In [6]:
```
# Call `get_params()` on stacking classifier object to see the params that w
gen_stacking_models().sclf.get_params()
```

Out[6]:
```
{'cv': 5,
 'estimators': [('LogisticRegression',
   LogisticRegression(multi_class='ovr', random_state=0)),
  ('RandomForest', RandomForestClassifier(n_estimators=25, random_state=
0))],
 'final_estimator__objective': 'binary:logistic',
 'final_estimator__base_score': None,
 'final_estimator__booster': None,
 'final_estimator__callbacks': None,
 'final_estimator__colsample_bylevel': None,
 'final_estimator__colsample_bynode': None,
 'final_estimator__colsample_bytree': None,
 'final_estimator__device': None,
 'final_estimator__early_stopping_rounds': None,
 'final_estimator__enable_categorical': False,
 'final_estimator__eval_metric': None,
 'final_estimator__feature_types': None,
 'final_estimator__gamma': None,
 'final_estimator__grow_policy': None,
 'final_estimator__importance_type': None,
 'final_estimator__interaction_constraints': None,
 'final_estimator__learning_rate': None,
 'final_estimator__max_bin': None,
 'final_estimator__max_cat_threshold': None,
 'final_estimator__max_cat_to_onehot': None,
 'final_estimator__max_delta_step': None,
 'final_estimator__max_depth': None,
 'final_estimator__max_leaves': None,
 'final_estimator__min_child_weight': None,
 'final_estimator__missing': nan,
 'final_estimator__monotone_constraints': None,
 'final_estimator__multi_strategy': None,
 'final_estimator__n_estimators': None,
 'final_estimator__n_jobs': None,
 'final_estimator__num_parallel_tree': None,
 'final_estimator__random_state': 0,
 'final_estimator__reg_alpha': None,
 'final_estimator__reg_lambda': None,
 'final_estimator__sampling_method': None,
 'final_estimator__scale_pos_weight': None,
 'final_estimator__subsample': None,
 'final_estimator__tree_method': None,
 'final_estimator__validate_parameters': None,
 'final_estimator__verbosity': None,
 'final_estimator': XGBClassifier(base_score=None, booster=None, callbacks=
None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, device=None, early_stopping_rounds=No
ne,
               enable_categorical=False, eval_metric=None, feature_types=No
ne,
               gamma=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=None, max_bin=No
ne,
               max_cat_threshold=None, max_cat_to_onehot=None,
```

```
                                step=None, max_depth=None, max_leaves=None,
```

```
               min_child_weight=None, missing=nan, monotone_constraints=Non
e,
               multi_strategy=None, n_estimators=None, n_jobs=None,
               num_parallel_tree=None, random_state=0, ...),
 'n_jobs': -1,
 'passthrough': True,
 'stack_method': 'predict_proba',
 'verbose': 0,
 'LogisticRegression': LogisticRegression(multi_class='ovr', random_state=
0),
 'RandomForest': RandomForestClassifier(n_estimators=25, random_state=0),
 'LogisticRegression__C': 1.0,
 'LogisticRegression__class_weight': None,
 'LogisticRegression__dual': False,
 'LogisticRegression__fit_intercept': True,
 'LogisticRegression__intercept_scaling': 1,
 'LogisticRegression__l1_ratio': None,
 'LogisticRegression__max_iter': 100,
 'LogisticRegression__multi_class': 'ovr',
 'LogisticRegression__n_jobs': None,
 'LogisticRegression__penalty': 'l2',
 'LogisticRegression__random_state': 0,
 'LogisticRegression__solver': 'lbfgs',
 'LogisticRegression__tol': 0.0001,
 'LogisticRegression__verbose': 0,
 'LogisticRegression__warm_start': False,
 'RandomForest__bootstrap': True,
 'RandomForest__ccp_alpha': 0.0,
 'RandomForest__class_weight': None,
 'RandomForest__criterion': 'gini',
 'RandomForest__max_depth': None,
 'RandomForest__max_features': 'sqrt',
 'RandomForest__max_leaf_nodes': None,
 'RandomForest__max_samples': None,
 'RandomForest__min_impurity_decrease': 0.0,
 'RandomForest__min_samples_leaf': 1,
 'RandomForest__min_samples_split': 2,
 'RandomForest__min_weight_fraction_leaf': 0.0,
 'RandomForest__n_estimators': 25,
 'RandomForest__n_jobs': None,
 'RandomForest__oob_score': False,
 'RandomForest__random_state': 0,
 'RandomForest__verbose': 0,
 'RandomForest__warm_start': False}
```

In [7]:

```
# Tune stacking models
def tune_stacking_models(
    stacking_models,
    X_train,
    y_train,
    cv=3, # By default, just do only 3 folds as we are limited in power
    n_iter=10, # For quick result, just iterate 10 combinations
):
    params = {
        # Find best params for RandomForest
        'RandomForest__max_depth': range(10, 100, 10),
```

```
        'RandomForest__min_samples_leaf': range(1, 8, 3),
        'RandomForest__min_samples_split': range(2, 10, 2),
        'RandomForest__n_estimators': range(25, 150, 25),
        # Find best params for XGBoost
        'final_estimator__min_child_weight': [1, 5, 10],
        'final_estimator__gamma': [0.5, 1, 1.5, 2, 5],
        'final_estimator__subsample': [0.6, 0.8, 1.0],
        'final_estimator__colsample_bytree': [0.6, 0.8, 1.0],
        'final_estimator__max_depth': [3, 4, 5],
        'final_estimator__n_estimators': range(60, 360, 40),
        'final_estimator__learning_rate': [0.1, 0.01, 0.05]
    }

    # Randomly search 50 models as runtime of Google Colab is limited for fr
    # Should run grid search on powerful resource with longer runtime
    searchCV = RandomizedSearchCV(
        estimator=stacking_models.sclf,
        param_distributions=params,
        scoring='roc_auc_ovr', # Scoring for both single-class and multi-cla
        cv=cv,
        n_iter=n_iter,
        refit=True,
        random_state=0,
        verbose=1,
    )
    searchCV.fit(X_train, y_train)

    return searchCV
```

In [8]:

```
def print_search_stats(searchCV):
    cv_keys = ('mean_test_score', 'std_test_score', 'params')

    for r, _ in enumerate(searchCV.cv_results_['mean_test_score']):
        print("%0.3f +/- %0.2f %r"
            % (searchCV.cv_results_[cv_keys[0]][r],
                searchCV.cv_results_[cv_keys[1]][r] / 2.0,
                searchCV.cv_results_[cv_keys[2]][r]))

    print('Best parameters: %s' % searchCV.best_params_)
    print('Accuracy: %.2f' % searchCV.best_score_)
```

2. Gender prediction model

a) Scenario 1: Devices with event data

In [9]:

```
# Format train/test data
X_train = segment_1_train.drop(columns=['device_id', 'gender', 'age_group'])
y_train = segment_1_train['gender']

X_test = segment_1_test.drop(columns=['device_id', 'gender', 'age_group'])
y_test = segment_1_test[['device_id', 'gender']]
```

```
    [
        X_train,
        y_train,
        X_test,
        y_test,
    ],
    [
        'X_train',
        'y_train',
        'X_test',
        'y_test',
    ],
):
    display(HTML(f'<h2>{label}</h2>'))
    display(df)
```

X_train

	average_daily_events	location_cluster_-1	location_cluster_0	location_cluster_1
0	0.380427	1.0	0.0	0.0
1	-0.621224	1.0	0.0	0.0
2	-2.669088	1.0	0.0	0.0
3	1.157734	1.0	0.0	0.0
4	-0.123856	1.0	0.0	0.0
...	...	...	...	...
2836	-0.623716	1.0	0.0	0.0
2837	1.288079	0.0	0.0	0.0
2838	-0.705781	1.0	0.0	0.0
2839	-0.026953	1.0	0.0	0.0
2840	1.041286	0.0	1.0	0.0

2841 rows × 1856 columns

y_train

```
0       1
1       1
2       1
3       1
4       0
       ..
2836    1
2837    1
2838    0
2839    1
2840    1
Name: gender, Length: 2841, dtype: int64
```

X_test

	average_daily_events	location_cluster_-1	location_cluster_0	location_cluster_1	l
0	2.356397	0.0	0.0	0.0	
1	-0.523348	1.0	0.0	0.0	
2	0.073159	1.0	0.0	0.0	
3	1.744514	0.0	0.0	0.0	
4	1.057478	1.0	0.0	0.0	
...	...	...	...	...	
1213	0.480375	1.0	0.0	0.0	
1214	-0.893056	1.0	0.0	0.0	
1215	0.478324	1.0	0.0	0.0	
1216	-0.013296	0.0	0.0	0.0	
1217	0.272121	1.0	0.0	0.0	

1218 rows × 1856 columns

y_test

	device_id	gender
0	-4968154927622700000	0
1	5164709194749140000	1
2	-446534884923407000	1
3	4929004728683190000	1
4	-6540623292245040000	0
...	...	...
1213	3558602119006800000	0
1214	-3049092807223440000	0
1215	4610975622206370000	1
1216	6339023951586040000	1
1217	-4479379906801590000	0

1218 rows × 2 columns

In [10]:

```
gender_models_sc1 = gen_stacking_models()
cross_val_check(gender_models_sc1, X_train, y_train)
```

```
Accuracy: 0.59 (+/- 0.01) [LogisticRegression]
Accuracy: 0.56 (+/- 0.02) [RandomForest]
Accuracy: 0.55 (+/- 0.02) [StackingClassifier]
```

In [11]:

```
# Tune stacking model
search_result_gender_sc1 = tune_stacking_models(
    gender_models_sc1,
    X_train,
    y_train,
    cv=3,
    n_iter=10, # Quick tune
)
print_search_stats(search_result_gender_sc1)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
0.603 +/- 0.01 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 260, 'final_estimator__min_child_weight': 1, 'final_estimator__max_d
epth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 2, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 80}
0.585 +/- 0.01 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
0.5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators':
75, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 30}
0.600 +/- 0.01 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 60, 'final_estimator__min_child_weight': 5, 'final_estimator__max_de
pth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.600 +/- 0.01 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 2
5, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 90}
0.601 +/- 0.01 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 5, 'final_estimator__max_d
epth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
1, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.598 +/- 0.01 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 5, 'final_estimator__max_d
epth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
0.5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators':
100, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.596 +/- 0.01 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 1, 'final_estimator__max_d
epth': 3, 'final_estimator__learning_rate': 0.1, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 1
00, 'RandomForest__min_samples_split': 8, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 10}
0.586 +/- 0.01 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 340, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 4, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
1, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 50}
0.577 +/- 0.01 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 340, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 5, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 0.6, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 90}
mator__subsample': 0.8, 'final_estimator__n_estim
```

```
ators': 260, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 10}
Best parameters: {'final_estimator__subsample': 0.6, 'final_estimator__n_est
imators': 260, 'final_estimator__min_child_weight': 1, 'final_estimator__max
_depth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamm
a': 5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimator
s': 75, 'RandomForest__min_samples_split': 2, 'RandomForest__min_samples_lea
f': 7, 'RandomForest__max_depth': 80}
Accuracy: 0.60
```

In [12]:
```
# # Fit best model on train data / predict on test data
gender_model_sc1 = search_result_gender_sc1.best_estimator_
gender_model_sc1_fit = gender_model_sc1.fit(X_train, y_train)
gender_preds_sc1 = gender_model_sc1_fit.predict_proba(X_test)

# Get IDs and predictions
preddf = y_test.copy()

for cls in search_result_gender_sc1.classes_:
    # Probabilities for classes (1,0)
    preddf['target_' + str(cls)] = [i[cls] for i in gender_preds_sc1]

# Look at predictions
preddf.head()
```

Out[12]:

	device_id	gender	target_0	target_1
0	-4968154927622700000	0	0.260571	0.739429
1	5164709194749140000	1	0.173655	0.826345
2	-446534884923407000	1	0.338526	0.661474
3	4929004728683190000	1	0.275155	0.724845
4	-6540623292245040000	0	0.183865	0.816135

b) Scenario 2: Devices without event data

In [13]:
```
# Format train/test data
X_train = segment_2_train.drop(columns=['device_id', 'gender', 'age_group'])
y_train = segment_2_train['gender']

X_test = segment_2_test.drop(columns=['device_id', 'gender', 'age_group'])
y_test = segment_2_test[['device_id', 'gender']]

for df, label in zip(
    [
        X_train,
        y_train,
        X_test,
```

```
    ],
    [
        'X_train',
        'y_train',
        'X_test',
        'y_test',
    ],
):
    display(HTML(f'<h2>{label}</h2>'))
    display(df)
```

X_train

	phone_brand_AUX	phone_brand_Bacardi	phone_brand_Bifer	phone_brand_CU
0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0
...	...	...	...	
49541	0.0	0.0	0.0	0
49542	0.0	0.0	0.0	0
49543	0.0	0.0	0.0	0
49544	0.0	0.0	0.0	0
49545	0.0	0.0	0.0	0

49546 rows × 97 columns

y_train

```
0        1
1        1
2        1
3        1
4        1
        ..
49541    0
49542    1
49543    1
49544    1
49545    0
Name: gender, Length: 49546, dtype: int64
```

X_test

	phone_brand_AUX	phone_brand_Bacardi	phone_brand_Bifer	phone_brand_CUB
0	0.0	0.0	0.0	0.
1	0.0	0.0	0.0	0.
2	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	0.
...	...	...	...	.
21230	0.0	0.0	0.0	0.
21231	0.0	0.0	0.0	0.
21232	0.0	0.0	0.0	0.
21233	0.0	0.0	0.0	0.
21234	0.0	0.0	0.0	0.

21235 rows × 97 columns

y_test

	device_id	gender
0	-191669847070955000	0
1	2403589567148540000	1
2	4604662545429270000	0
3	4019394794123470000	0
4	-1021613832219450000	1
...	...	...
21230	3557324664602540000	1
21231	-931978254629029000	0
21232	-2171067714073140000	0
21233	-7956453462733460000	1
21234	5294464040764260000	1

21235 rows × 2 columns

In [14]:
```
gender_models_sc2 = gen_stacking_models()
cross_val_check(gender_models_sc2, X_train, y_train)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Accuracy: 0.57 (+/- 0.00) [LogisticRegression]
Accuracy: 0.56 (+/- 0.00) [RandomForest]
Accuracy: 0.55 (+/- 0.01) [StackingClassifier]
```

In [15]:
```
# Tune stacking model
search_result_gender_sc2 = tune_stacking_models(
    gender_models_sc2,
    X_train,
    y_train,
    cv=3,
    n_iter=10, # Quick tune
)
print_search_stats(search_result_gender_sc2)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
0.565 +/- 0.00 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 260, 'final_estimator__min_child_weight': 1, 'final_estimator__max_d
epth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 2, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 80}
0.564 +/- 0.00 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
0.5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators':
75, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 30}
0.564 +/- 0.00 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 60, 'final_estimator__min_child_weight': 5, 'final_estimator__max_de
pth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.564 +/- 0.00 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 2
5, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 90}
0.563 +/- 0.00 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 5, 'final_estimator__max_d
epth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
1, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.563 +/- 0.00 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 5, 'final_estimator__max_d
epth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
0.5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators':
100, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.565 +/- 0.00 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 1, 'final_estimator__max_d
epth': 3, 'final_estimator__learning_rate': 0.1, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 1
00, 'RandomForest__min_samples_split': 8, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 10}
0.563 +/- 0.00 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 340, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 4, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
1, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 50}
0.564 +/- 0.00 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 340, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 5, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 0.6, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 90}
mator__subsample': 0.8, 'final_estimator__n_estim
```

```
ators': 260, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 10}
Best parameters: {'final_estimator__subsample': 0.8, 'final_estimator__n_est
imators': 260, 'final_estimator__min_child_weight': 10, 'final_estimator__ma
x_depth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamm
a': 5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimator
s': 50, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_lea
f': 4, 'RandomForest__max_depth': 10}
Accuracy: 0.57
```

In [16]:

```
# # Fit best model on train data / predict on test data
gender_model_sc2 = search_result_gender_sc2.best_estimator_
gender_model_sc2_fit = gender_model_sc2.fit(X_train, y_train)
gender_preds_sc2 = gender_model_sc2_fit.predict_proba(X_test)

# Get IDs and predictions
preddf = y_test.copy()

for cls in search_result_gender_sc2.classes_:
    # Probabilities for classes (1,0)
    preddf['target_' + str(cls)] = [i[cls] for i in gender_preds_sc2]

# Look at predictions
preddf.head()
```

Out[16]:

	device_id	gender	target_0	target_1
0	-1916698470709550000	0	0.360289	0.639711
1	2403589567148540000	1	0.398954	0.601046
2	4604662545429270000	0	0.459624	0.540376
3	4019394794123470000	0	0.440108	0.559892
4	-1021613832219450000	1	0.296910	0.703090

3. Age group prediction model

a) Scenario 1: Devices with event data

In [17]:

```
# Format train/test data
X_train = segment_1_train.drop(columns=['device_id', 'gender', 'age_group'])
y_train = segment_1_train['age_group']

X_test = segment_1_test.drop(columns=['device_id', 'gender', 'age_group'])
y_test = segment_1_test[['device_id', 'age_group']]

for df, label in zip(
    [
```

```
        y_train,
        X_test,
        y_test,
    ],
    [
        'X_train',
        'y_train',
        'X_test',
        'y_test',
    ],
):
    display(HTML(f'<h2>{label}</h2>'))
    display(df.info())
    display(df)
```

X_train

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2841 entries, 0 to 2840
Columns: 1856 entries, average_daily_events to app_categories_zombies game
dtypes: float64(1856)
memory usage: 40.2 MB
None
```

	average_daily_events	location_cluster_-1	location_cluster_0	location_cluster_1
0	0.380427	1.0	0.0	0.0
1	-0.621224	1.0	0.0	0.0
2	-2.669088	1.0	0.0	0.0
3	1.157734	1.0	0.0	0.0
4	-0.123856	1.0	0.0	0.0
...	...	...	...	...
2836	-0.623716	1.0	0.0	0.0
2837	1.288079	0.0	0.0	0.0
2838	-0.705781	1.0	0.0	0.0
2839	-0.026953	1.0	0.0	0.0
2840	1.041286	0.0	1.0	0.0

2841 rows × 1856 columns

y_train

```
<class 'pandas.core.series.Series'>
RangeIndex: 2841 entries, 0 to 2840
Series name: age_group
Non-Null Count  Dtype
--------------  -----
2841 non-null   int64
dtypes: int64(1)
memory usage: 22.3 KB
None
0       2
1       2
2       0
3       2
4       1
       ..
2836    2
2837    1
2838    0
2839    2
2840    2
Name: age_group, Length: 2841, dtype: int64
```

X_test

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1218 entries, 0 to 1217
Columns: 1856 entries, average_daily_events to app_categories_zombies game
dtypes: float64(1856)
memory usage: 17.2 MB
None
```

	average_daily_events	location_cluster_-1	location_cluster_0	location_cluster_1	l
0	2.356397	0.0	0.0	0.0	
1	-0.523348	1.0	0.0	0.0	
2	0.073159	1.0	0.0	0.0	
3	1.744514	0.0	0.0	0.0	
4	1.057478	1.0	0.0	0.0	
...	...	...	...	...	
1213	0.480375	1.0	0.0	0.0	
1214	-0.893056	1.0	0.0	0.0	
1215	0.478324	1.0	0.0	0.0	
1216	-0.013296	0.0	0.0	0.0	
1217	0.272121	1.0	0.0	0.0	

1218 rows × 1856 columns

y_test

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1218 entries, 0 to 1217
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   device_id  1218 non-null   int64
 1   age_group  1218 non-null   int64
dtypes: int64(2)
memory usage: 19.2 KB
None
```

	device_id	age_group
0	-4968154927622700000	3
1	5164709194749140000	1
2	-446534884923407000	1
3	4929004728683190000	3
4	-6540623292245040000	2
...	...	...
1213	3558602119006800000	1
1214	-3049092807223440000	2
1215	4610975622206370000	1
1216	6339023951586040000	1
1217	-4479379906801590000	1

1218 rows × 2 columns

In [18]:
```
age_models_sc1 = gen_stacking_models()
cross_val_check(age_models_sc1, X_train, y_train)
```

```
Accuracy: 0.61 (+/- 0.00) [LogisticRegression]
Accuracy: 0.58 (+/- 0.01) [RandomForest]
Accuracy: 0.58 (+/- 0.00) [StackingClassifier]
```

In [19]:
```
# Tune XGBoost meta learner
search_result_age_sc1 = tune_stacking_models(
    age_models_sc1,
    X_train,
    y_train,
    cv=3,
    n_iter=10, # Quick tune
)
print_search_stats(search_result_age_sc1)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
0.613 +/- 0.01 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 260, 'final_estimator__min_child_weight': 1, 'final_estimator__max_d
epth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 2, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 80}
0.602 +/- 0.01 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
0.5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators':
75, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 30}
0.614 +/- 0.01 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 60, 'final_estimator__min_child_weight': 5, 'final_estimator__max_de
pth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.614 +/- 0.01 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 2
5, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 90}
0.614 +/- 0.01 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 5, 'final_estimator__max_d
epth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
1, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.608 +/- 0.01 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 5, 'final_estimator__max_d
epth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
0.5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators':
100, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.611 +/- 0.01 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 1, 'final_estimator__max_d
epth': 3, 'final_estimator__learning_rate': 0.1, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 1
00, 'RandomForest__min_samples_split': 8, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 10}
0.609 +/- 0.01 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 340, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 4, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
1, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 50}
0.605 +/- 0.01 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 340, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 5, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 0.6, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 90}
mator__subsample': 0.8, 'final_estimator__n_estim
```

```
ators': 260, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 10}
Best parameters: {'final_estimator__subsample': 0.6, 'final_estimator__n_est
imators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__ma
x_depth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamm
a': 5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimator
s': 25, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_lea
f': 7, 'RandomForest__max_depth': 90}
Accuracy: 0.61
```

In [20]:
```
# # Fit best model on train data / predict on test data
age_model_sc1 = search_result_age_sc1.best_estimator_
age_model_sc1_fit = age_model_sc1.fit(X_train, y_train)
age_preds_sc1 = age_model_sc1_fit.predict_proba(X_test)

# Get IDs and predictions
preddf = y_test.copy()

for cls in search_result_age_sc1.classes_:
    # Probabilities for classes (1,0)
    preddf['target_' + str(cls)] = [i[cls] for i in age_preds_sc1]

# Look at predictions
preddf.head()
```

Out[20]:

	device_id	age_group	target_0	target_1	target_2	target_3
0	-4968154927622700000	3	0.161505	0.297810	0.319064	0.221620
1	5164709194749140000	1	0.254838	0.349330	0.230596	0.165237
2	-446534884923407000	1	0.256015	0.322561	0.247072	0.174353
3	4929004728683190000	3	0.164227	0.281160	0.326102	0.228511
4	-6540623292245040000	2	0.173422	0.312385	0.316651	0.197542

b) Scenario 2: Devices without event data

In [21]:
```
# Format train/test data
X_train = segment_2_train.drop(columns=['device_id', 'gender', 'age_group'])
y_train = segment_2_train['age_group']

X_test = segment_2_test.drop(columns=['device_id', 'gender', 'age_group'])
y_test = segment_2_test[['device_id', 'age_group']]

for df, label in zip(
    [
        X_train,
        y_train,
        X_test,
```

```
    ],
    [
        'X_train',
        'y_train',
        'X_test',
        'y_test',
    ],
):
    display(HTML(f'<h2>{label}</h2>'))
    display(df)
```

X_train

	phone_brand_AUX	phone_brand_Bacardi	phone_brand_Bifer	phone_brand_CUI
0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0
...	...	...	...	
49541	0.0	0.0	0.0	0
49542	0.0	0.0	0.0	0
49543	0.0	0.0	0.0	0
49544	0.0	0.0	0.0	0
49545	0.0	0.0	0.0	0

49546 rows × 97 columns

y_train

```
0        1
1        0
2        1
3        1
4        1
        ..
49541    2
49542    0
49543    0
49544    1
49545    2
Name: age_group, Length: 49546, dtype: int64
```

X_test

	phone_brand_AUX	phone_brand_Bacardi	phone_brand_Bifer	phone_brand_CUB
0	0.0	0.0	0.0	0.
1	0.0	0.0	0.0	0.
2	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	0.
...	...	...	...	.
21230	0.0	0.0	0.0	0.
21231	0.0	0.0	0.0	0.
21232	0.0	0.0	0.0	0.
21233	0.0	0.0	0.0	0.
21234	0.0	0.0	0.0	0.

21235 rows × 97 columns

y_test

	device_id	age_group
0	-191669847070955000	3
1	2403589567148540000	2
2	4604662545429270000	1
3	4019394794123470000	2
4	-1021613832219450000	2
...	...	...
21230	3557324664602540000	1
21231	-931978254629029000	1
21232	-2171067714073140000	0
21233	-7956453462733460000	0
21234	5294464040764260000	0

21235 rows × 2 columns

In [22]:
```
age_models_sc2 = gen_stacking_models()
cross_val_check(age_models_sc2, X_train, y_train)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Accuracy: 0.56 (+/- 0.00) [LogisticRegression]
Accuracy: 0.56 (+/- 0.00) [RandomForest]
Accuracy: 0.56 (+/- 0.00) [StackingClassifier]
```

In [23]:

```python
# Tune stacking model
search_result_age_sc2 = tune_stacking_models(
    age_models_sc2,
    X_train,
    y_train,
    cv=3,
    n_iter=10, # Quick tune
)
print_search_stats(search_result_age_sc2)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
0.562 +/- 0.00 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 260, 'final_estimator__min_child_weight': 1, 'final_estimator__max_d
epth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 2, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 80}
0.561 +/- 0.00 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
0.5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators':
75, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 30}
0.562 +/- 0.00 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 60, 'final_estimator__min_child_weight': 5, 'final_estimator__max_de
pth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.562 +/- 0.00 {'final_estimator__subsample': 0.6, 'final_estimator__n_estim
ators': 100, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 2
5, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 90}
0.561 +/- 0.00 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 5, 'final_estimator__max_d
epth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
1, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.561 +/- 0.00 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 5, 'final_estimator__max_d
epth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
0.5, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators':
100, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
0.561 +/- 0.00 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 300, 'final_estimator__min_child_weight': 1, 'final_estimator__max_d
epth': 3, 'final_estimator__learning_rate': 0.1, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 1
00, 'RandomForest__min_samples_split': 8, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 10}
0.562 +/- 0.00 {'final_estimator__subsample': 1.0, 'final_estimator__n_estim
ators': 340, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 4, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
1, 'final_estimator__colsample_bytree': 0.8, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 50}
0.562 +/- 0.00 {'final_estimator__subsample': 0.8, 'final_estimator__n_estim
ators': 340, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 5, 'final_estimator__learning_rate': 0.05, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 0.6, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 90}
mator__subsample': 0.8, 'final_estimator__n_estim
```

```
ators': 260, 'final_estimator__min_child_weight': 10, 'final_estimator__max_
depth': 4, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
5, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 5
0, 'RandomForest__min_samples_split': 6, 'RandomForest__min_samples_leaf':
4, 'RandomForest__max_depth': 10}
Best parameters: {'final_estimator__subsample': 0.6, 'final_estimator__n_est
imators': 60, 'final_estimator__min_child_weight': 5, 'final_estimator__max_
depth': 3, 'final_estimator__learning_rate': 0.01, 'final_estimator__gamma':
2, 'final_estimator__colsample_bytree': 1.0, 'RandomForest__n_estimators': 7
5, 'RandomForest__min_samples_split': 4, 'RandomForest__min_samples_leaf':
7, 'RandomForest__max_depth': 40}
Accuracy: 0.56
```

In [24]:

```
# # Fit best model on train data / predict on test data
age_model_sc2 = search_result_age_sc2.best_estimator_
age_model_sc2_fit = age_model_sc2.fit(X_train, y_train)
age_preds_sc2 = age_model_sc2_fit.predict_proba(X_test)

# Get IDs and predictions
preddf = y_test.copy()

for cls in search_result_age_sc2.classes_:
    # Probabilities for classes (1,0)
    preddf['target_' + str(cls)] = [i[cls] for i in age_preds_sc2]

# Look at predictions
preddf.head()
```

Out[24]:

	device_id	age_group	target_0	target_1	target_2	target_3
0	-191669847070955000	3	0.227483	0.288712	0.255148	0.228657
1	2403589567148540000	2	0.223313	0.304401	0.264650	0.207636
2	4604662545429270000	1	0.282612	0.303827	0.224053	0.189508
3	4019394794123470000	2	0.276634	0.301501	0.232793	0.189072
4	-1021613832219450000	2	0.244803	0.291603	0.258171	0.205422

4. Save the models for evaluation later

In [25]:

```
import pickle
```

In [26]:

```
os.makedirs('deploy', exist_ok=True)

with open('deploy/gender_model_sc1_fit.pickle', 'wb') as f:
  pickle.dump(gender_model_sc1_fit, f)

with open('deploy/gender_model_sc2_fit.pickle', 'wb') as f:
  pickle.dump(gender_model_sc2_fit, f)

with open('deploy/age_model_sc1_fit.pickle', 'wb') as f:
  pickle.dump(age_model_sc1_fit, f)
```

```python
with open('deploy/age_model_sc2_fit.pickle', 'wb') as f:
  pickle.dump(age_model_sc2_fit, f)
```

In [27]:
```python
os.listdir('deploy')
```

Out[27]:
```
['gender_model_sc1_fit.pickle',
 'gender_model_sc2_fit.pickle',
 'age_model_sc1_fit.pickle',
 'age_model_sc2_fit.pickle',
 'input_processor.pickle',
 'age_label_encoder.pickle',
 'gender_label_encoder.pickle',
 'deploy_data.csv']
```