Interacting with Campaigns

In this notebook, you will deploy and interact with campaigns in Amazon Personalize.

Introduction

Back to top

At this point, you should have several solutions and at least one solution version for each. Once a solution version is created, it is possible to get recommendations from them, and to get a feel for their overall behavior.

This notebook starts off by deploying each of the solution versions from the previous notebook into individual campaigns. Once they are active, there are resources for querying the recommendations, and helper functions to digest the output into something more human-readable.

As you with your customer on Amazon Personalize, you can modify the helper functions to fit the structure of their data input files to keep the additional rendering working.

To get started, once again, we need to import libraries, load values from previous notebooks, and load the SDK.

In [1]:

```
import time
from time import sleep
import json
from datetime import datetime
import uuid
import random

import boto3
import pandas as pd
```

In [2]:

```
%store -r
```

In [3]:

```
personalize = boto3.client('personalize')
personalize_runtime = boto3.client('personalize-runtime')

# Establish a connection to Personalize's event streaming
personalize_events = boto3.client(service_name='personalize-events')
```

Adding some helper functions to make results more readable

In [4]:

```
def get_item_name_from_id ( item_id ):
    item_name = item_metadata_df [item_metadata_df ['id'] == item_id]['name'].values[0]
    return item_name
```

In [5]:

```
def get_item_category_from_id ( item_id ):
    item_name = item_metadata_df [item_metadata_df ['id'] == item_id]['category'].values[0]
    return item_name
```

In [6]:

```
def get_item_style_from_id ( item_id ):
    item_name = item_metadata_df [item_metadata_df ['id'] == item_id]['style'].values[0]
    return item_name
```

In [7]:

```
def add_item_name_to_df ( df ):
    test = df.copy()

    test['ITEM_NAME'] = test.apply(
          lambda row:
            get_item_name_from_id(row['ITEM_ID'] ), axis=1
      )

    display(test)
    return (test)
```

Test Campaigns

Now that our campaigns have been fully created, let's test each campaign and evaluate the results.

Test Related Product Recommendations Campaign

Let's test the recommendations made by the related items/products campaign by selecting a product from the Retail Demo Store's Products (https://github.com/aws-samples/retail-demo-store/tree/master/src/products) microservice and requesting related item recommendations for that product.

Select a Product

We'll just pick a random product for simplicity. Feel free to change the `product_id` below and execute the following cells with a different product to get a sense for how the recommendations change.

In [8]:

```
product_id = '020a5afe-fb13-4499-a1fa-8594d326eaa0'

display (get_item_name_from_id ( product_id ))
```

```
'Elegant Ceramic Bowl'
```

Get Related Product Recommendations for Product

Now let's call Amazon Personalize to get related item/product recommendations for our product from the related item campaign.

In [9]:

```
get_recommendations_response = personalize_runtime.get_recommendations(
    campaignArn = related_campaign_arn,
    itemId = str(product_id),
    numResults = 10
)

item_list = get_recommendations_response['itemList']
```

In [10]:

```
df = pd.DataFrame()
df['Item'] = [ itm['itemId'] for itm in item_list ]
df['Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in item_list ]
df['Category'] = [ get_item_category_from_id ( itm['itemId']) for itm in item_list ]
df['Style'] = [ get_item_style_from_id ( itm['itemId']) for itm in item_list ]
display (df)
```

	Item	Name	Category	Style
0	eb8f10ab-1317-4a11-b058-b2098bb64326	Chef Knife	housewares	kitchen
1	a4c0f41d-4e7d-422c-86f1-57432c0fdba2	White Plates	housewares	kitchen
2	3f90e04e-9bfe-4fd4-a137-387b694baad2	Chic Ceramic Bowl	housewares	bowls
3	ef446e39-c864-46ea-b273-f2a48b7dc2a5	Teapot	housewares	kitchen
4	57a7d4c1-03f7-4a5b-a618-cbfb5a0004f1	Honey Dipper	housewares	kitchen
5	cacf945a-4c63-4797-a9a9-361e14b7001e	Wooden Plates	housewares	kitchen
6	68e865bc-3db7-4f5d-86e3-8e7a651cf0b7	Kettle	housewares	kitchen
7	8bdfaf9c-4ff6-46bc-a304-33db265f36ef	Classic Ceramic Bowl	housewares	bowls
8	2b8f89d0-4078-4701-8aac-89c48d8ba392	Christmas Wreath	seasonal	christmas
9	cacb5fe5-f77f-4bd8-979c-8eec17cb3255	Everyday Glass	housewares	kitchen

Based on the random product selected above, do the similar item recommendations from Personalize make sense? Keep in mind that the similar item recommendations from the SIMS recipe are based on the interactions we generated as input into the solution creation process above.

Test Product Recommendations Campaign

Let's test the recommendations made by the product recommendations campaign by selecting a user from the Retail Demo Store's Users microservice and requesting item recommendations for that user.

Select a User

We'll just pick a random user for simplicity. Feel free to change the `user_id` below and execute the following cells with a different user to get a sense for how the recommendations change.

In [11]:

```
user_id = 555
user_metadata_df[user_metadata_df['id']==user_id]
```

Out[11]:

	id	gender	first_name	last_name	email	age	name	username	persona	disc
554	555	F	Autumn	Rodriguez	autumn.rodriguez@example.com	51	Autumn Rodriguez	user555	outdoors_instruments_groceries	lower_p

Get Product Recommendations for User

Now let's call Amazon Personalize to get recommendations for our user from the product recommendations campaign.

In [12]:

```
get_recommendations_response = personalize_runtime.get_recommendations(
    campaignArn = recommend_campaign_arn,
    userId = str(user_id),
    numResults = 10
)

item_list = get_recommendations_response['itemList']
```

In [13]:

```
df = pd.DataFrame()
df['Item'] = [ itm['itemId'] for itm in item_list ]
df['Score'] = [ itm['score'] for itm in item_list ]
df['Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in item_list ]
df['Category'] = [ get_item_category_from_id ( itm['itemId']) for itm in item_list ]
df['Style'] = [ get_item_style_from_id ( itm['itemId']) for itm in item_list ]
display (df)
```

	Item	Score	Name	Category	Style
0	18465f60-709e-4ab1-add2-2d8e677e16a0	0.332782	Foolproof Fishing Lure	outdoors	fishing
1	75f06592-a0a2-4dc5-acfb-76b275dae3aa	0.113493	Kayak	outdoors	kayaking
2	bdc9a89f-de9e-4f99-b5f8-1c381df80970	0.079766	Fishing Net	outdoors	fishing
3	5b7611a0-1093-470f-8aca-7f85da315624	0.045039	Upright Piano	instruments	keys
4	4ea3fcbe-665b-4c33-b8b6-1d593460153d	0.026588	Electric Guitar	instruments	strings
5	cc42e0f4-abaf-445b-b843-54884c4f6845	0.024620	Fishing Reel	outdoors	fishing
6	56ca3f0a-03cc-4496-9530-41d8c6069459	0.022542	Electric Guitar	instruments	strings
7	c5b52a98-63db-4924-98d3-c05a319dd8ac	0.017872	Cymbal	instruments	percussion
8	a4634431-29bd-45a8-801f-55fb33b68bb4	0.011805	Acoustic Drum	instruments	percussion
9	10a400a4-3afa-4b7b-989e-dc7ed6298efd	0.009037	Drum Set	instruments	percussion

Notice that in this response we have a `score` field returned with each `itemId`. For all recipes except SIMS and Popularity-Count, Personalize calculates a score (https://docs.aws.amazon.com/personalize/latest/dg/getting-real-time-recommendations.html) for each recommended item. Score values are between 0.0 and 1.0 and the sum of all scores across all items in your interactions and items datasets will total to 1.0. Therefore, the absolute value of scores will be smaller for larger item catalogs. We'll see how scores are calculated a bit differently for the personalized-ranking recipe below.

Applying the discount context

We'll get the user recommendations when discount context is applied for comparison. This is a using the "contextual metadata" feature of Amazon Personalize.

In [14]:

```
get_recommendations_response = personalize_runtime.get_recommendations(
    campaignArn = recommend_campaign_arn,
    userId = str(user_id),
    numResults = 10,
    context = {'DISCOUNT':'Yes'} # Here we provide the context for the recommendations
)

item_list_context = get_recommendations_response['itemList']

df = pd.DataFrame()
df['Item'] = [ itm['itemId'] for itm in item_list_context ]
df['Score'] = [ itm['score'] for itm in item_list_context ]
df['Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in item_list_context ]
df['Category'] = [ get_item_category_from_id ( itm['itemId']) for itm in item_list_context ]
df['Style'] = [ get_item_style_from_id ( itm['itemId']) for itm in item_list_context ]
display (df)
```

	Item	Score	Name	Category	Style
0	18465f60-709e-4ab1-add2-2d8e677e16a0	0.347940	Foolproof Fishing Lure	outdoors	fishing
1	75f06592-a0a2-4dc5-acfb-76b275dae3aa	0.110822	Kayak	outdoors	kayaking
2	bdc9a89f-de9e-4f99-b5f8-1c381df80970	0.079337	Fishing Net	outdoors	fishing
3	5b7611a0-1093-470f-8aca-7f85da315624	0.041870	Upright Piano	instruments	keys
4	4ea3fcbe-665b-4c33-b8b6-1d593460153d	0.024942	Electric Guitar	instruments	strings
5	cc42e0f4-abaf-445b-b843-54884c4f6845	0.023850	Fishing Reel	outdoors	fishing
6	56ca3f0a-03cc-4496-9530-41d8c6069459	0.021719	Electric Guitar	instruments	strings
7	c5b52a98-63db-4924-98d3-c05a319dd8ac	0.018982	Cymbal	instruments	percussion
8	a4634431-29bd-45a8-801f-55fb33b68bb4	0.010950	Acoustic Drum	instruments	percussion
9	10a400a4-3afa-4b7b-989e-dc7ed6298efd	0.009103	Drum Set	instruments	percussion

Let us compare it to the previous set of recommendations without context.

In [15]:

```
df = pd.DataFrame()
df['Item (No context)'] = [ itm['itemId'] for itm in item_list ]
df['Score (No context)'] = [ itm['score'] for itm in item_list ]
df['Name (No context)'] = [ get_item_name_from_id ( itm['itemId']) for itm in item_list ]

df['Item (Context)'] = [ itm['itemId'] for itm in item_list_context ]
df['Score (Context)'] = [ itm['score'] for itm in item_list_context ]
df['Name (Context)'] = [ get_item_name_from_id ( itm['itemId']) for itm in item_list_context ]
display (df)
```

	Item (No context)	Score (No context)	Name (No context)	Item (Context)	Score (Context)	Name (Context)
0	18465f60-709e-4ab1-add2-2d8e677e16a0	0.332782	Foolproof Fishing Lure	18465f60-709e-4ab1-add2-2d8e677e16a0	0.347940	Foolproof Fishing Lure
1	75f06592-a0a2-4dc5-acfb-76b275dae3aa	0.113493	Kayak	75f06592-a0a2-4dc5-acfb-76b275dae3aa	0.110822	Kayak
2	bdc9a89f-de9e-4f99-b5f8-1c381df80970	0.079766	Fishing Net	bdc9a89f-de9e-4f99-b5f8-1c381df80970	0.079337	Fishing Net
3	5b7611a0-1093-470f-8aca-7f85da315624	0.045039	Upright Piano	5b7611a0-1093-470f-8aca-7f85da315624	0.041870	Upright Piano
4	4ea3fcbe-665b-4c33-b8b6-1d593460153d	0.026588	Electric Guitar	4ea3fcbe-665b-4c33-b8b6-1d593460153d	0.024942	Electric Guitar
5	cc42e0f4-abaf-445b-b843-54884c4f6845	0.024620	Fishing Reel	cc42e0f4-abaf-445b-b843-54884c4f6845	0.023850	Fishing Reel
6	56ca3f0a-03cc-4496-9530-41d8c6069459	0.022542	Electric Guitar	56ca3f0a-03cc-4496-9530-41d8c6069459	0.021719	Electric Guitar
7	c5b52a98-63db-4924-98d3-c05a319dd8ac	0.017872	Cymbal	c5b52a98-63db-4924-98d3-c05a319dd8ac	0.018982	Cymbal
8	a4634431-29bd-45a8-801f-55fb33b68bb4	0.011805	Acoustic Drum	a4634431-29bd-45a8-801f-55fb33b68bb4	0.010950	Acoustic Drum
9	10a400a4-3afa-4b7b-989e-dc7ed6298efd	0.009037	Drum Set	10a400a4-3afa-4b7b-989e-dc7ed6298efd	0.009103	Drum Set

Test Personalized Ranking Campaign

Next let's evaluate the results of the personalized ranking campaign. As a reminder, given a list of items and a user, this campaign will rerank the items based on the preferences of the user. For the Retail Demo Store, we will use this campaign to rerank the products listed for each category and the featured products list as well as reranking catalog search results displayed in the search widget.

Get Featured Products List

First let's get the list of featured products from the Products data.

In [16]:

```
featured_products = item_metadata_df[item_metadata_df['featured']==True]
```

ReRank Featured Products

Using the featured products list just retrieved, first we'll create a list of item IDs that we want to rerank for a specific user. This reranking will allow us to provide ranked products based on the user's behavior. These behaviors should be consistent the same persona that was mentioned above (since we're going to use the same `user_id`).

In [17]:

```
unranked_product_ids = list (featured_products['id'])

df = pd.DataFrame()
df['Item'] = [ itm for itm in unranked_product_ids ]
df['Name'] = [ get_item_name_from_id ( itm) for itm in unranked_product_ids ]
df['Category'] = [ get_item_category_from_id ( itm) for itm in unranked_product_ids ]
df['Style'] = [ get_item_style_from_id ( itm) for itm in unranked_product_ids ]
display (df)
```

	Item	Name	Category	Style
0	2b67230f-dc22-462e-9afe-c9e459f74093	Tan Handbag	accessories	handbag
1	6bd74f2d-90c0-4ca6-9663-f3bbe9bf405b	Dark Red Jacket	apparel	jacket
2	b87da3f8-9a3e-417d-abd7-16329c5be1ba	Luxurious Soap	beauty	bathing
3	5d37a44b-d121-426e-b528-59e603ba5923	Visit Egypt	books	travel
4	3b145528-d5fc-4c2a-b2a5-e119128caa5f	Faultless Headphones	electronics	headphones
5	4bb66b8a-cf13-4959-87ce-ca506fa568a2	Wedding Bouquet	floral	bouquet
6	22552eb1-57f1-4fa3-a93a-a9fa22851f9f	Red Sneakers	footwear	sneaker
7	8b9733b9-cbea-4de3-978b-5e3f0e8c796c	Chocolate Sofa	furniture	sofas
8	a31ad4b3-f9a8-4a9b-a8b3-3034af7bacec	Kiwi	groceries	fruits
9	01a8978b-2a84-4dbd-acc4-aff74a468681	Ceramic Vase	homedecor	decorative
10	1e96e374-be23-4c97-b87e-b5c45cb8999f	Pots Set	housewares	kitchen
11	3f9a39b2-0d63-4751-b6ee-4ecd08dd2276	Electric Guitar	instruments	strings
12	2ad09e8e-fd41-4d29-953e-546b924d7cb8	Trendy Necklace	jewelry	necklace
13	7160b264-e3ed-4ac3-9dd7-2c537b00e5ed	Dog Frisbee	outdoors	pet
14	6f04daee-7387-442f-bc99-a9b0072b29ce	Halloween Lights	seasonal	halloween
15	8bffb5fb-624f-48a8-a99f-b8e9c64bbe29	Screwdriver	tools	screwdriver

Now let's have Personalize rank the featured product IDs based on our random user.

In [18]:

```
response = personalize_runtime.get_personalized_ranking(
    campaignArn=ranking_campaign_arn,
    inputList=unranked_product_ids,
    userId=str(user_id)
)
reranked = response['personalizedRanking']
print(json.dumps(response['personalizedRanking'], indent = 4))
```

```
[
    {
        "itemId": "3f9a39b2-0d63-4751-b6ee-4ecd08dd2276",
        "score": 0.8012908
    },
    {
        "itemId": "7160b264-e3ed-4ac3-9dd7-2c537b00e5ed",
        "score": 0.1718156
    },
    {
        "itemId": "a31ad4b3-f9a8-4a9b-a8b3-3034af7bacec",
        "score": 0.0113003
    },
    {
        "itemId": "3b145528-d5fc-4c2a-b2a5-e119128caa5f",
        "score": 0.0076272
    },
    {
        "itemId": "6f04daee-7387-442f-bc99-a9b0072b29ce",
        "score": 0.0041661
    },
    {
        "itemId": "5d37a44b-d121-426e-b528-59e603ba5923",
        "score": 0.0025272
    },
    {
        "itemId": "8bffb5fb-624f-48a8-a99f-b8e9c64bbe29",
        "score": 0.0006437
    },
    {
        "itemId": "b87da3f8-9a3e-417d-abd7-16329c5be1ba",
        "score": 0.0002364
    },
    {
        "itemId": "2b67230f-dc22-462e-9afe-c9e459f74093",
        "score": 0.0001071
    },
    {
        "itemId": "01a8978b-2a84-4dbd-acc4-aff74a468681",
        "score": 9.39e-05
    },
    {
        "itemId": "4bb66b8a-cf13-4959-87ce-ca506fa568a2",
```

```
        "score": 7.4e-05
    },
    {
        "itemId": "22552eb1-57f1-4fa3-a93a-a9fa22851f9f",
        "score": 7.25e-05
    },
    {
        "itemId": "2ad09e8e-fd41-4d29-953e-546b924d7cb8",
        "score": 2.81e-05
    },
    {
        "itemId": "1e96e374-be23-4c97-b87e-b5c45cb8999f",
        "score": 1.31e-05
    },
    {
        "itemId": "8b9733b9-cbea-4de3-978b-5e3f0e8c796c",
        "score": 2.6e-06
    },
    {
        "itemId": "6bd74f2d-90c0-4ca6-9663-f3bbe9bf405b",
        "score": 1.4e-06
    }
]
```

In [19]:

```
df = pd.DataFrame()
df['Item'] = [ itm['itemId'] for itm in reranked ]
df['Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in reranked ]
df['Category'] = [ get_item_category_from_id ( itm['itemId']) for itm in reranked ]
df['Style'] = [ get_item_style_from_id ( itm['itemId']) for itm in reranked ]
display (df)
```

	Item	Name	Category	Style
0	3f9a39b2-0d63-4751-b6ee-4ecd08dd2276	Electric Guitar	instruments	strings
1	7160b264-e3ed-4ac3-9dd7-2c537b00e5ed	Dog Frisbee	outdoors	pet
2	a31ad4b3-f9a8-4a9b-a8b3-3034af7bacec	Kiwi	groceries	fruits
3	3b145528-d5fc-4c2a-b2a5-e119128caa5f	Faultless Headphones	electronics	headphones
4	6f04daee-7387-442f-bc99-a9b0072b29ce	Halloween Lights	seasonal	halloween
5	5d37a44b-d121-426e-b528-59e603ba5923	Visit Egypt	books	travel
6	8bffb5fb-624f-48a8-a99f-b8e9c64bbe29	Screwdriver	tools	screwdriver
7	b87da3f8-9a3e-417d-abd7-16329c5be1ba	Luxurious Soap	beauty	bathing
8	2b67230f-dc22-462e-9afe-c9e459f74093	Tan Handbag	accessories	handbag
9	01a8978b-2a84-4dbd-acc4-aff74a468681	Ceramic Vase	homedecor	decorative
10	4bb66b8a-cf13-4959-87ce-ca506fa568a2	Wedding Bouquet	floral	bouquet
11	22552eb1-57f1-4fa3-a93a-a9fa22851f9f	Red Sneakers	footwear	sneaker
12	2ad09e8e-fd41-4d29-953e-546b924d7cb8	Trendy Necklace	jewelry	necklace
13	1e96e374-be23-4c97-b87e-b5c45cb8999f	Pots Set	housewares	kitchen
14	8b9733b9-cbea-4de3-978b-5e3f0e8c796c	Chocolate Sofa	furniture	sofas
15	6bd74f2d-90c0-4ca6-9663-f3bbe9bf405b	Dark Red Jacket	apparel	jacket

Are the reranked results different than the original results from the Search service? Notice that we are also given a score for each item but this time the score values are larger. This is because scores for personalized-ranking results are calculated just across the items being reranked. Experiment with a different `user_id` in the cells above to see how the item ranking changes.

Pick products for discount

Using the featured products list we'll pick some products for discount from the featured products.

We'll get the ranking when discount context is applied for comparison. This is a using the "contextual metadata" feature of Amazon Personalize.

In [20]:

```
response = personalize_runtime.get_personalized_ranking(
    campaignArn=ranking_campaign_arn,
    inputList=unranked_product_ids,
    userId=str(user_id),
    context={'DISCOUNT': 'Yes'} # Here we provide the context for the ranking
)
discount_reranked = response['personalizedRanking']
print('Discount context ranking:', json.dumps(discount_reranked, indent = 4))
```

Discount context ranking: [


```
    {
        "itemId": "3f9a39b2-0d63-4751-b6ee-4ecd08dd2276",
        "score": 0.7739404
    },
    {
        "itemId": "7160b264-e3ed-4ac3-9dd7-2c537b00e5ed",
        "score": 0.1971011
    },
    {
        "itemId": "a31ad4b3-f9a8-4a9b-a8b3-3034af7bacec",
        "score": 0.0136081
    },
    {
        "itemId": "3b145528-d5fc-4c2a-b2a5-e119128caa5f",
        "score": 0.0071166
    },
    {
        "itemId": "6f04daee-7387-442f-bc99-a9b0072b29ce",
        "score": 0.0043265
    },
    {
        "itemId": "5d37a44b-d121-426e-b528-59e603ba5923",
        "score": 0.00267
    },
    {
        "itemId": "8bffb5fb-624f-48a8-a99f-b8e9c64bbe29",
        "score": 0.0006252
    },
    {
        "itemId": "b87da3f8-9a3e-417d-abd7-16329c5be1ba",
        "score": 0.0002417
    },
    {
        "itemId": "01a8978b-2a84-4dbd-acc4-aff74a468681",
        "score": 9.42e-05
    },
    {
        "itemId": "2b67230f-dc22-462e-9afe-c9e459f74093",
        "score": 9.28e-05
    },
    {
        "itemId": "4bb66b8a-cf13-4959-87ce-ca506fa568a2",
```

```
        "score": 7e-05
    },
    {
        "itemId": "22552eb1-57f1-4fa3-a93a-a9fa22851f9f",
        "score": 6.86e-05
    },
    {
        "itemId": "2ad09e8e-fd41-4d29-953e-546b924d7cb8",
        "score": 2.59e-05
    },
    {
        "itemId": "1e96e374-be23-4c97-b87e-b5c45cb8999f",
        "score": 1.54e-05
    },
    {
        "itemId": "8b9733b9-cbea-4de3-978b-5e3f0e8c796c",
        "score": 1.9e-06
    },
    {
        "itemId": "6bd74f2d-90c0-4ca6-9663-f3bbe9bf405b",
        "score": 1.4e-06
    }
]
```

In [21]:

```
df = pd.DataFrame()
df['Item'] = [ itm['itemId'] for itm in discount_reranked ]
df['Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in discount_reranked ]
df['Category'] = [ get_item_category_from_id ( itm['itemId']) for itm in discount_reranked ]
df['Style'] = [ get_item_style_from_id ( itm['itemId']) for itm in discount_reranked ]
display (df)
```

	Item	Name	Category	Style
0	3f9a39b2-0d63-4751-b6ee-4ecd08dd2276	Electric Guitar	instruments	strings
1	7160b264-e3ed-4ac3-9dd7-2c537b00e5ed	Dog Frisbee	outdoors	pet
2	a31ad4b3-f9a8-4a9b-a8b3-3034af7bacec	Kiwi	groceries	fruits
3	3b145528-d5fc-4c2a-b2a5-e119128caa5f	Faultless Headphones	electronics	headphones
4	6f04daee-7387-442f-bc99-a9b0072b29ce	Halloween Lights	seasonal	halloween
5	5d37a44b-d121-426e-b528-59e603ba5923	Visit Egypt	books	travel
6	8bffb5fb-624f-48a8-a99f-b8e9c64bbe29	Screwdriver	tools	screwdriver
7	b87da3f8-9a3e-417d-abd7-16329c5be1ba	Luxurious Soap	beauty	bathing
8	01a8978b-2a84-4dbd-acc4-aff74a468681	Ceramic Vase	homedecor	decorative
9	2b67230f-dc22-462e-9afe-c9e459f74093	Tan Handbag	accessories	handbag
10	4bb66b8a-cf13-4959-87ce-ca506fa568a2	Wedding Bouquet	floral	bouquet
11	22552eb1-57f1-4fa3-a93a-a9fa22851f9f	Red Sneakers	footwear	sneaker
12	2ad09e8e-fd41-4d29-953e-546b924d7cb8	Trendy Necklace	jewelry	necklace
13	1e96e374-be23-4c97-b87e-b5c45cb8999f	Pots Set	housewares	kitchen
14	8b9733b9-cbea-4de3-978b-5e3f0e8c796c	Chocolate Sofa	furniture	sofas
15	6bd74f2d-90c0-4ca6-9663-f3bbe9bf405b	Dark Red Jacket	apparel	jacket

Let us compare the original list, the reranked list and the reranked list with context.

In [22]:

```
df = pd.DataFrame()
df['Original List'] = [ itm for itm in unranked_product_ids]
df['Original List Name'] = [ get_item_name_from_id ( itm) for itm in unranked_product_ids ]

df['Original Reranking'] = [ itm['itemId'] for itm in reranked]
df['Original Reranking Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in reranked ]

df['Discount Reranking'] = [ itm['itemId'] for itm in discount_reranked]
df['Discount Reranking Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in discount_reranked ]

display (df)
```

	Original List	Original List Name	Original Reranking	Original Reranking Name	Discount Reranking	Discount Reranking Name
0	2b67230f-dc22-462e-9afe-c9e459f74093	Tan Handbag	3f9a39b2-0d63-4751-b6ee-4ecd08dd2276	Electric Guitar	3f9a39b2-0d63-4751-b6ee-4ecd08dd2276	Electric Guitar
1	6bd74f2d-90c0-4ca6-9663-f3bbe9bf405b	Dark Red Jacket	7160b264-e3ed-4ac3-9dd7-2c537b00e5ed	Dog Frisbee	7160b264-e3ed-4ac3-9dd7-2c537b00e5ed	Dog Frisbee
2	b87da3f8-9a3e-417d-abd7-16329c5be1ba	Luxurious Soap	a31ad4b3-f9a8-4a9b-a8b3-3034af7bacec	Kiwi	a31ad4b3-f9a8-4a9b-a8b3-3034af7bacec	Kiwi
3	5d37a44b-d121-426e-b528-59e603ba5923	Visit Egypt	3b145528-d5fc-4c2a-b2a5-e119128caa5f	Faultless Headphones	3b145528-d5fc-4c2a-b2a5-e119128caa5f	Faultless Headphones
4	3b145528-d5fc-4c2a-b2a5-e119128caa5f	Faultless Headphones	6f04daee-7387-442f-bc99-a9b0072b29ce	Halloween Lights	6f04daee-7387-442f-bc99-a9b0072b29ce	Halloween Lights
5	4bb66b8a-cf13-4959-87ce-ca506fa568a2	Wedding Bouquet	5d37a44b-d121-426e-b528-59e603ba5923	Visit Egypt	5d37a44b-d121-426e-b528-59e603ba5923	Visit Egypt
6	22552eb1-57f1-4fa3-a93a-a9fa22851f9f	Red Sneakers	8bffb5fb-624f-48a8-a99f-b8e9c64bbe29	Screwdriver	8bffb5fb-624f-48a8-a99f-b8e9c64bbe29	Screwdriver
7	8b9733b9-cbea-4de3-978b-5e3f0e8c796c	Chocolate Sofa	b87da3f8-9a3e-417d-abd7-16329c5be1ba	Luxurious Soap	b87da3f8-9a3e-417d-abd7-16329c5be1ba	Luxurious Soap
8	a31ad4b3-f9a8-4a9b-a8b3-3034af7bacec	Kiwi	2b67230f-dc22-462e-9afe-c9e459f74093	Tan Handbag	01a8978b-2a84-4dbd-acc4-aff74a468681	Ceramic Vase
9	01a8978b-2a84-4dbd-acc4-aff74a468681	Ceramic Vase	01a8978b-2a84-4dbd-acc4-aff74a468681	Ceramic Vase	2b67230f-dc22-462e-9afe-c9e459f74093	Tan Handbag
10	1e96e374-be23-4c97-b87e-b5c45cb8999f	Pots Set	4bb66b8a-cf13-4959-87ce-ca506fa568a2	Wedding Bouquet	4bb66b8a-cf13-4959-87ce-ca506fa568a2	Wedding Bouquet
11	3f9a39b2-0d63-4751-b6ee-4ecd08dd2276	Electric Guitar	22552eb1-57f1-4fa3-a93a-a9fa22851f9f	Red Sneakers	22552eb1-57f1-4fa3-a93a-a9fa22851f9f	Red Sneakers
12	2ad09e8e-fd41-4d29-953e-546b924d7cb8	Trendy Necklace	2ad09e8e-fd41-4d29-953e-546b924d7cb8	Trendy Necklace	2ad09e8e-fd41-4d29-953e-546b924d7cb8	Trendy Necklace
13	7160b264-e3ed-4ac3-9dd7-2c537b00e5ed	Dog Frisbee	1e96e374-be23-4c97-b87e-b5c45cb8999f	Pots Set	1e96e374-be23-4c97-b87e-b5c45cb8999f	Pots Set
14	6f04daee-7387-442f-bc99-a9b0072b29ce	Halloween Lights	8b9733b9-cbea-4de3-978b-5e3f0e8c796c	Chocolate Sofa	8b9733b9-cbea-4de3-978b-5e3f0e8c796c	Chocolate Sofa
15	8bffb5fb-624f-48a8-a99f-b8e9c64bbe29	Screwdriver	6bd74f2d-90c0-4ca6-9663-f3bbe9bf405b	Dark Red Jacket	6bd74f2d-90c0-4ca6-9663-f3bbe9bf405b	Dark Red Jacket

Has the ranking changed?

Event Tracking - Keeping up with evolving user intent

Up to this point we have trained and deployed three Amazon Personalize campaigns based on historical data that we generated in this workshop. This allows us to make related product, user recommendations, and rerank product lists based on already observed behavior of our users. However, user intent often changes in real-time such that what products the user is interested in now may be different than what they were interested in a week ago, a day ago, or even a few minutes ago. Making recommendations that keep up with evolving user intent is one of the more difficult challenges with personalization. Fortunately, Amazon Personalize has a mechanism for this exact issue.

Amazon Personalize supports the ability to send real-time user events (i.e. clickstream) data into the service. Personalize uses this event data to improve recommendations. It will also save these events and automatically include them when solutions for the same dataset group are re-created (i.e. model retraining).

The Retail Demo Store's Web UI already has logic to send events (https://github.com/aws-samples/retail-demo-store/blob/master/src/web-ui/src/analytics/AnalyticsHandler.js) such as 'ProductViewed', 'ProductAdded', 'OrderCompleted', and others as they occur in real-time to a Personalize Event Tracker. These are the same event types we used to initially create the solutions and campaigns for our three use-cases. All we need to do is create an event tracker in Personalize, set the tracking Id for the tracker in an SSM parameter, and rebuild the Web UI service to pick up the change.

Create Personalize Event Tracker

Let's start by creating an event tracker for our dataset group.

In [23]:

```
event_tracker_response = personalize.create_event_tracker(
    datasetGroupArn=dataset_group_arn,
    name='retaildemostore-event-tracker'
)

event_tracker_arn = event_tracker_response['eventTrackerArn']
event_tracking_id = event_tracker_response['trackingId']

print('Event Tracker ARN: ' + event_tracker_arn)
print('Event Tracking ID: ' + event_tracking_id)
```

```
Event Tracker ARN: arn:aws:personalize:us-east-1:968824662561:event-tracker/0d3904d4
Event Tracking ID: 8b67590d-195d-4c46-b4f4-ea0995fcfeb1
```

Wait for Event Tracker Status to Become ACTIVE

The event tracker should take a minute or so to become active.

In [24]:

```
status = None
max_time = time.time() + 60*60 # 1 hours
while time.time() < max_time:
    describe_event_tracker_response = personalize.describe_event_tracker(
        eventTrackerArn = event_tracker_arn
    )
    status = describe_event_tracker_response["eventTracker"]["status"]
    print("EventTracker: {}".format(status))

    if status == "ACTIVE" or status == "CREATE FAILED":
        break

    time.sleep(15)
```

```
EventTracker: CREATE PENDING
EventTracker: ACTIVE
```

Cold User Recommendations

One of the key features of Personalize is being able to cold start users. Cold users are typically those who are new to your site or application and cold starting a user is getting from no personalization to making personalized recommendations in real-time.

Personalize accomplishes cold starting users via the Event Tracker, just as we saw above with existing users. However, since new users are typically anonymous for a period of time before they create an account or may choose to transact as a guest, personalization is a valuable tool to help convert those anonymous users to transacting users.

The challenge here is that Personalize needs a `userId` for anonymous users before it can make personalized recommendations. The Retail Demo Store solves this challenge by creating a provisional user ID the moment an anonymous user first hits the site. This provisional user ID is then used when streaming events to the Event Tracker and when retrieving recommendations from the Recommendations service. This allows the Retail Demo Store to start serving personalized recommendations after the first couple events are streamed to Personalize. Before recommendations can be personalized, Personalize will provide recommendations for popular items as a fallback.

To see this behavior in action, browse to the Retail Demo Store storefront using a different browser, an Incognito/Private window, or sign out of your existing account. What you should see on the home page is that instead of **"Inspired by your shopping behavior"**, the section is **"Trending products"**. After you click on a couple provide detail pages, return to the home page and see that the section title and recommendations have changed. This indicates that recommendations are now being personalized and will continue to become more relevant as you engage with products.

Similarly, the category pages will rerank products at first based on popularity and then become more and more personalized.

There are some challenges with this approach, though. First is the question of what to do with the provisional user ID when the user creates an account. To maintain continuity of the user's interaction history, the Retail Demo Store passes the provisional user ID to the Users microservice when creating a new user account. The Users service then uses this ID as the user's ID going forward. Another challenge is how to handle a user that anonymously browses the site using multiple devices such as on the mobile device and then on a desktop/laptop. In this case, separate provisional user IDs are generated for sessions on each device. However, once the user creates an account on one device and then signs in with that account on the other device, both devices will starting using the same user ID going forward. A side effect here is that the interaction history from one of the devices will be orphaned. This is an acceptable tradeoff given the benefit of cold starting users earlier and is functionally the same UX without this scheme. Additional logic could be added to merge the interaction history from both prior anonymous sessions when the user creates an account. Also, customer data platforms can be used to help manage this for you.

Test Purchased Products Filter

To test our purchased products filter, we will request recommendations for a random user. Then we will send an `OrderCompleted` event for one of the recommended products to Personalize using the event tracker created above. Finally, we will request recommendations again for the same user but this time specify our filter.

In [25]:

```
# Pick a user ID in the range of test users and fetch 5 recommendations.
user_id = 456
display(user_metadata_df[user_metadata_df['id']==user_id])
```

	id	gender	first_name	last_name	email	age	name	username	persona	discount_persona	t
455	456	M	Alfred	Lang	alfred.lang@example.com	35	Alfred Lang	user456	housewares_floral_seasonal	all_discounts	

In [26]:

```
get_recommendations_response = personalize_runtime.get_recommendations(
    campaignArn = recommend_campaign_arn,
    userId = str(user_id),
    numResults = 5
)

item_list = get_recommendations_response['itemList']
df = pd.DataFrame()
df['Item'] = [ itm['itemId'] for itm in item_list ]
df['Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in item_list ]
df['Category'] = [ get_item_category_from_id ( itm['itemId']) for itm in item_list ]
df['Style'] = [ get_item_style_from_id ( itm['itemId']) for itm in item_list ]
display (df)
```

	Item	Name	Category	Style
0	dfd7c361-dc70-4bb4-9c05-e6357ecabc49	Cocktail Glass	housewares	kitchen
1	bbcda337-3411-47e4-aeec-079663f729df	Baking Dish	housewares	kitchen
2	d826bc9c-a212-49b0-b5f9-e18c631fc5db	Floral Design Plates	housewares	kitchen
3	5b53ab8d-701c-4139-bdab-dc457e546157	Wedding Bouquet	floral	bouquet
4	323ca3fe-7849-490a-933d-e742866a2843	Chef Knife	housewares	kitchen

Next let's randomly select an item from the returned list of recommendations to be our product to purchase.

In [27]:

```
product_id_to_purchase = random.choice(item_list)['itemId']
print(f'Product to simulate purchasing: {product_id_to_purchase}')
print(f'Product name: {get_item_name_from_id ( product_id_to_purchase)}')
```

```
Product to simulate purchasing: d826bc9c-a212-49b0-b5f9-e18c631fc5db
Product name: Floral Design Plates
```

Next let's send an `OrderCompleted` event to Personalize to simulate that the product was just purchased. This will match the criteria for our filter. In the Retail Demo Store web application, this event is sent for each product in the order after the order is completed.

In [28]:

```
response = personalize_events.put_events(
    trackingId = event_tracking_id,
    userId = str(user_id),
    sessionId = str(uuid.uuid4()),
    eventList = [
        {
            'eventId': str(uuid.uuid4()),
            'eventType': 'OrderCompleted',
            'itemId': str(product_id_to_purchase),
            'sentAt': int(time.time()),
            'properties': '{"discount": "No"}'
        }
    ]
)

# Wait for OrderCompleted event to become consistent.
time.sleep(10)

print(json.dumps(response, indent=2))
```


```
{
  "ResponseMetadata": {
    "RequestId": "e601a463-9e1c-4d2d-a3c9-3ba1b658b142",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "content-type": "application/json",
      "date": "Sun, 18 Apr 2021 04:37:47 GMT",
      "x-amzn-requestid": "e601a463-9e1c-4d2d-a3c9-3ba1b658b142",
      "content-length": "0",
      "connection": "keep-alive"
    },
    "RetryAttempts": 0
  }
}
```

Finally, let's retrieve recommendations for the user again but this time specifying the filter to exclude recently purchased items. We do this by passing the filter's ARN via the `filterArn` parameter. In the Retail Demo Store, this is done in the Recommendations (https://github.com/aws-samples/retail-demo-store/tree/master/src/recommendations) service.

In [29]:

```
get_recommendations_response = personalize_runtime.get_recommendations(
    campaignArn = recommend_campaign_arn,
    userId = str(user_id),
    numResults = 5,
    filterArn = filter_arn
)

item_list = get_recommendations_response['itemList']
df = pd.DataFrame()
df['Item'] = [ itm['itemId'] for itm in item_list ]
df['Name'] = [ get_item_name_from_id ( itm['itemId']) for itm in item_list ]
df['Category'] = [ get_item_category_from_id ( itm['itemId']) for itm in item_list ]
df['Style'] = [ get_item_style_from_id ( itm['itemId']) for itm in item_list ]
display (df)
```

	Item	Name	Category	Style
0	bbcda337-3411-47e4-aeec-079663f729df	Baking Dish	housewares	kitchen
1	c42cfabf-8978-4936-b71a-c15c7e62058f	Roses Arrangement	floral	arrangement
2	0c4744e2-b989-4509-a7e2-7d8dc43ff404	Drought-Resistant Indoor Plant	floral	plant
3	4ac6fe0c-cc84-4c7b-99e3-45c95de4e68f	Utensils Set	housewares	kitchen
4	d85e36ba-bf6e-42e6-81a1-d8f555f6cf93	Drought-Resistant Indoor Plant	floral	plant

The following code will raise an assertion error if the product we just purchased is still recommended.

In [30]:

```
found_item = next((item for item in item_list if item['itemId'] == product_id_to_purchase), None)
if found_item:
    assert found_item == False, 'Purchased item found unexpectedly in recommendations'
else:
    print('Purchased item filtered from recommendations for user!')
```

```
Purchased item filtered from recommendations for user!
```

Wrap up

Back to top

With that you now have a fully working collection of models to tackle various recommendation and personalization scenarios, as well as the skills to manipulate customer data to better integrate with the service, and a knowledge of how to do all this over APIs and by leveraging open source data science tools.

Use these notebooks as a guide to getting started with your customers for POCs. As you find missing components, or discover new approaches, cut a pull request and provide any additional helpful components that may be missing from this collection.

You'll want to make sure that you clean up all of the resources deployed during this POC. We have provided a separate notebook which shows you how to identify and delete the resources in `05_Clean_Up_Resources.ipynb` .

In [31]:

```
%store event_tracker_arn
```

```
Stored 'event_tracker_arn' (str)
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```