HH Personalize Recommender

This notebook will walk you through the steps to build a Domain dataset group and a recommender that returns product recommendations based on data generated for our fictitious retail store data set. The goal is to recommend products that are relevant based on a particular user.

This synthetic data comes from the Retail Demo Store project. Follow the link to learn more about the data and potential uses.

How to Use the Notebook

The code is broken up into cells like the one below. There's a triangular Run button at the top of this page that you can click to execute each cell and move onto the next, or you can press `Shift` + `Enter` while in the cell to execute it and move onto the next one.

As a cell is executing you'll notice a line to the side showcase an `*` while the cell is running or it will update to a number to indicate the last cell that completed executing after it has finished exectuting all the code within a cell.

Simply follow the instructions below and execute the cells to get started with Amazon Personalize using case optimized recommenders.

Install Dependencies

22 hours ago (16s) 3

```
%python %pip install --upgrade s3fs botocore %restart_python
```

Imports

Python ships with a broad collection of libraries and we need to import those as well as the ones installed to help us like boto3 (AWS SDK for python) and Pandas/Numpy which are core data science tools.

22 hours ago (1s) 5

```
# Imports
import boto3
import json
import numpy as np
import pandas as pd
import time
import datetime
```

Setup Environment Variables

Load Environment Variables from File

Run this cell to load environment variables from a file named `.env`.

8

```
from dotenv import load_dotenv

if load_dotenv(dotenv_path="../.env"):
    print("Loaded .env file")
else:
    print("No .env file found")
```

Set Environment Variables

Run this cell if you don't have a `.env` file.

22 hours ago (<1s) 10 Python

```
%env AWS_ACCESS_KEY_ID= %env AWS_SECRET_ACCESS_KEY= %env AWS_REGION=ap-southeast ...
```

Set Configuration Values from Environment Variables

```
✓ 22 hours ago (<1s)                                                    12

import os

AWS_ACCESS_KEY_ID = os.getenv("AWS_ACCESS_KEY_ID")
AWS_SECRET_ACCESS_KEY = os.getenv("AWS_SECRET_ACCESS_KEY")
AWS_REGION = os.getenv("AWS_REGION")
S3_BUCKET_NAME = os.getenv("S3_BUCKET_NAME")
S3_INTERACTIONS_PREFIX = os.getenv("S3_INTERACTIONS_PREFIX")
S3_ITEMS_PREFIX = os.getenv("S3_ITEMS_PREFIX")
S3_USERS_PREFIX = os.getenv("S3_USERS_PREFIX")
```

Verify that you're on the right account by running the following cell.

```
✓ 22 hours ago (1s)                                                     14

iam = boto3.client(
    "iam",
    region_name=AWS_REGION,
    aws_access_key_id=AWS_ACCESS_KEY_ID,
    aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
)

response = iam.list_account_aliases()
account_alias = (
    response["AccountAliases"][0] if response["AccountAliases"] else "No alias found"
)

print("Account Alias: ", account_alias)
```

```
Account Alias:  hungryhub-prod
```

Next you will want to validate that your environment can communicate successfully with Amazon Personalize, the lines below do just that.

```
✓ 22 hours ago (<1s)                                                    16

# Configure the SDK to Personalize:
personalize = boto3.client(
    "personalize",
    region_name=AWS_REGION,
    aws_access_key_id=AWS_ACCESS_KEY_ID,
    aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
)
personalize_runtime = boto3.client(
    "personalize-runtime",
    region_name=AWS_REGION,
    aws_access_key_id=AWS_ACCESS_KEY_ID,
    aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
)
```

Specify an S3 Bucket and Data Output Location

Amazon Personalize will need an S3 bucket to act as the source of your data. The code bellow will create a bucket with a unique `bucket_name`.

The Amazon S3 bucket needs to be in the same region as the Amazon Personalize resources.

```
✓ 22 hours ago (<1s)                                                    18

print('AWS_REGION:', AWS_REGION)
print("S3_BUCKET_NAME: ", S3_BUCKET_NAME)
```

```
AWS_REGION: ap-southeast-1
S3_BUCKET_NAME:  production-hh-personalize-s3-bucket
```

```
✓ 22 hours ago (1s)                                                     19

s3 = boto3.client('s3',
    region_name=AWS_REGION,
    aws_access_key_id=AWS_ACCESS_KEY_ID,
    aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
)

try:
```

```
    if AWS_REGION == "us-east-1":
        s3.create_bucket(Bucket=S3_BUCKET_NAME)
    else:
        s3.create_bucket(
            Bucket = S3_BUCKET_NAME,
            CreateBucketConfiguration={'LocationConstraint': AWS_REGION}
            )
except s3.exceptions.BucketAlreadyOwnedByYou:
    print("Bucket already exists. Using bucket", S3_BUCKET_NAME)
```

```
Bucket already exists. Using bucket production-hh-personalize-s3-bucket
```

Load Data

▶ ✓ 22 hours ago (<1s) 21

```
import requests

# Available Hosts:
# - http://localhost:4001
# - https://personalize-worker.hh-engineering.my.id
# - https://personalize-worker.hungryhub.com
HOST = "https://personalize-worker.hungryhub.com"
API_KEY = "worker.w9WmL3odnEKThRxPTZYSKhyFlRYuS0Zy"
COMMON_HEADERS = {
    "Content-Type": "application/json",
    "X-REQUEST-TYPE": "GraphQL",
    "X-Hh-Language": "en",
    "X-API-Key": API_KEY
}

def send_graphql_request(mutation: str, operation_name: str = None):
    """Send GraphQL mutation request with standardized headers"""
    url = f"{HOST}/graphql"

    payload = {
        "query": mutation.strip()
    }

    if operation_name:
        payload["operationName"] = operation_name

    try:
        response = requests.post(
            url,
            json=payload,
            headers=COMMON_HEADERS,
            timeout=10
        )
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        return {"error": str(e)}
```

▶ ✓ 22 hours ago (<1s) 22

```
result = send_graphql_request(
    """
      mutation {
        LoadReservationData {
          success
          message
        }
      }
    """
)

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

```
{
  "data": {
    "LoadReservationData": {
      "success": true,
      "message": "Event queued"
    }
  }
}
```

▶ ✓ 22 hours ago (<1s) 23

```
result = send_graphql_request(
    """
      mutation {
        LoadUserData {
          success
          message
        }
      }
```

```
    """
)

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


```
{
  "data": {
    "LoadUserData": {
      "success": true,
      "message": "Event queued"
    }
  }
}
```


22 hours ago (<1s) 24

```
result = send_graphql_request(
    """
    mutation {
      LoadRestaurantData {
        success
        message
      }
    }
    """
)

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


```
{
  "data": {
    "LoadRestaurantData": {
      "success": true,
      "message": "Event queued"
    }
  }
}
```


Verify Loaded Data

22 hours ago (<1s) 26

```
from IPython.display import display

def verify_loaded_data(prefix):
    response = s3.list_objects_v2(Bucket=S3_BUCKET_NAME, Prefix=prefix)
    # Check if the response contains any files

    if "Contents" in response:
        # Get all file objects
        files = response["Contents"]

        # Sort the files by last modified timestamp
        files = sorted(files, key=lambda x: x["LastModified"])

        # Get the last file
        last_file = files[-1]["Key"]

        # Read the CSV directly into a pandas DataFrame
        s3_url = f"s3://{S3_BUCKET_NAME}/{last_file}"
        df = pd.read_csv(s3_url)

        print(f"Last file: {s3_url}")
        display(df.head())
    else:
        print("No files found in the specified folder.")
```

21 hours ago (<1s) 27

```
verify_loaded_data(S3_INTERACTIONS_PREFIX)
```

```
Last file: s3://production-hh-personalize-s3-bucket/interactions-v1/production-reservations-1738570098741.csv
```

	user_id	item_id	timestamp	event_type
0	731920	4086	1738125543	Purchase
1	1010322	2003	1738125662	Purchase
2	704205	3377	1738125733	Purchase
3	422671	4875	1738125815	View
4	1015141	4503	1738125985	Purchase

21 hours ago (<1s) 28

```
verify_loaded_data(S3_USERS_PREFIX)
```

```
Last file: s3://production-hh-personalize-s3-bucket/users-v1/production-users-1738570110906.csv
```

	user_id	gender	loyalty_level_id
0	1012400	unknown	1
1	1012401	unknown	1
2	1012402	unknown	1
3	1012403	unknown	1
4	1012404	unknown	1

21 hours ago (<1s) 29

```
verify_loaded_data(S3_ITEMS_PREFIX)
```

```
Last file: s3://production-hh-personalize-s3-bucket/items-v1/production-restaurants-1738570112681.csv
```

	item_id	name	city_id	price	category_l1	category_l2	creation_timestamp
0	5342	Indigo Bar at Bay Beach Resort Jomtien (Pattaya)	4	422	International|Pub/Bar|Pattaya	International|Casual Dining|Hotel Restaurant|F...	1735084800
1	5266	Nava 6	1	396	Thai|Open Air/Outdoor|Asiatique	Thai|Casual Dining|Family Friendly|Open Air/Ou...	1735084800
2	5296	Lab Ped Rama 4	1	538	Thai Isaan|Casual Dining|Rama 4	Thai Isaan|Casual Dining|Family Friendly|Phra ...	1734998400
3	5367	Suea Noi Mookrata	1	389	Thai|Grill/BBQ|Phra Khanong	Thai|Casual Dining|Family Friendly|Grill/BBQ|P...	1735516800
4	5274	Jojo Soba	1	230	Japanese|Cafe/Dessert|MRT Sam Yan	Japanese|Casual Dining|Family Friendly|Cafe/De...	1733961600

Explore the Dataset

Let's learn more about the dataset by viewing its charateristics

32

```
df = pd.read_csv('./interactions.csv')
df

bucket = 'your-bucket'
prefix = 'your-folder/'

# List CSV files
response = s3.list_objects_v2(Bucket=bucket, Prefix=prefix)
csv_files = [f"s3://{bucket}/{obj['Key']}" for obj in response['Contents'] if obj['Key'].endswith('.csv')]

# Read all CSVs into a DataFrame
df = pd.concat((pd.read_csv(f) for f in csv_files), ignore_index=True)

# Analyze
print(df.head())
print(df.describe())
```

33

```
df.EVENT_TYPE.value_counts()
```

The ECOMMERCE recommenders require you to provide specific EVENT_TYPE values in order to understand when users view products and purchase products. These event types must be `View` and `Purchase`, respectively. Note from the output above, the dataset already has the appropriate `View` and `Purchase` events. Additional event types can also be included in training data, and they will be used in training, but they all should represent positive intent/interest by the user.

35

```
df.info()
```

From the 2 cells above, we've learned that our data has has 5 columns, over 600K rows and the headers are: ITEM_ID, USER_ID, EVENT_TYPE, TIMESTAMP and DISCOUNT.

To be compatible with an Amazon Personalize interactions schema, this dataset requires column headings compatible with Amazon Personalize default column names (read about column names here)

Configure an S3 bucket and an IAM role

So far, we have downloaded, manipulated, and saved the data onto the Amazon EBS instance attached to instance running this Jupyter notebook. However, Amazon Personalize will need an S3 bucket to act as the source of your data, as well as IAM roles for accessing that bucket. Let's set all of that up.

Set the S3 bucket policy

Amazon Personalize needs to be able to read the contents of your S3 bucket. So add a bucket policy which allows that.

Note: Make sure the role you are using to run the code in this notebook has the necessary permissions to modify the S3 bucket policy.

✓ 21 hours ago (1s) 39

```
s3 = boto3.client(
    "s3",
    region_name=AWS_REGION,
    aws_access_key_id=AWS_ACCESS_KEY_ID,
    aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
)

policy = {
    "Version": "2012-10-17",
    "Id": "PersonalizeS3BucketAccessPolicy",
    "Statement": [
        {
            "Sid": "PersonalizeS3BucketAccessPolicy",
            "Effect": "Allow",
            "Principal": {"Service": "personalize.amazonaws.com"},
            "Action": ["s3:GetObject", "s3:ListBucket"],
            "Resource": [
                "arn:aws:s3:::{}".format(S3_BUCKET_NAME),
                "arn:aws:s3:::{}/*".format(S3_BUCKET_NAME),
            ],
        }
    ],
}

s3.put_bucket_policy(Bucket=S3_BUCKET_NAME, Policy=json.dumps(policy, indent=2))
```

```
{'ResponseMetadata': {'RequestId': '50XASHPBV3SVPW95',
  'HostId': 'DNaeuqrSuJyeyXUa8i+4HyWfXVPtbV945jxIGXcIdYUjV09dOCbnvdW1eu3QhznMM8XSpxenwec=',
  'HTTPStatusCode': 204,
  'HTTPHeaders': {'x-amz-id-2': 'DNaeuqrSuJyeyXUa8i+4HyWfXVPtbV945jxIGXcIdYUjV09dOCbnvdW1eu3QhznMM8XSpxenwec=',
   'x-amz-request-id': '50XASHPBV3SVPW95',
   'date': 'Mon, 03 Feb 2025 08:16:20 GMT',
   'server': 'AmazonS3'},
  'RetryAttempts': 0}}
```

Create and Wait for Dataset Group

The largest grouping in Personalize is a Dataset Group, this will isolate your data, event trackers, solutions, Recommenders, and campaigns. Grouping things together that share a common collection of data. Feel free to alter the name below if you'd like.

Create Dataset Group

✓ 21 hours ago (<1s) 41

```
def get_existing_dataset_group_arn(dataset_group_name):
    response = personalize.list_dataset_groups()
    for dataset_group in response['datasetGroups']:
        if dataset_group['name'] == dataset_group_name:
            return dataset_group['datasetGroupArn']
    return None

dataset_group_name = 'hh_personalize_dataset_group'
dataset_group_arn = get_existing_dataset_group_arn(dataset_group_name)

if dataset_group_arn:
    print(f"Dataset group already exists: {dataset_group_arn}")
else:
    response = personalize.create_dataset_group(
        name=dataset_group_name,
        domain='ECOMMERCE'
    )
    dataset_group_arn = response['datasetGroupArn']
    print("Created new dataset group:")
    print(json.dumps(response, indent=2))

print(f"Final dataset group ARN: {dataset_group_arn}")
```


```
{
  "datasetGroupArn": "arn:aws:personalize:ap-southeast-1:202255947274:dataset-group/hh_personalize_dataset_group",
  "domain": "ECOMMERCE",
  "ResponseMetadata": {
    "RequestId": "d3be7623-dafe-47c8-b00c-5968e4235345",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 08:16:33 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "133",
```

```
    "connection": "keep-alive",
    "x-amzn-requestid": "d3be7623-dafe-47c8-b00c-5968e4235345",
    "strict-transport-security": "max-age=47304000; includeSubDomains",
    "x-frame-options": "DENY",
    "cache-control": "no-cache",
    "x-content-type-options": "nosniff"
   },
   "RetryAttempts": 0
  }
}
Final dataset group ARN: arn:aws:personalize:ap-southeast-1:202255947274:dataset-group/hh_personalize_dataset_group
```

Wait for Dataset Group to Have ACTIVE Status Before we can use the Dataset Group in any items below it must be active, execute the cell below and wait for it to show active.

```
✓ 21 hours ago (<1s)                                    43

%%time

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    describe_dataset_group_response = personalize.describe_dataset_group(
        datasetGroupArn = dataset_group_arn
    )
    status = describe_dataset_group_response["datasetGroup"]["status"]
    print("DatasetGroup: {}".format(status))

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

    time.sleep(15)
```

```
DatasetGroup: ACTIVE
CPU times: user 3.24 ms, sys: 0 ns, total: 3.24 ms
Wall time: 11 ms
```

Create Interactions Schema

A core component of how Personalize understands your data comes from the Schema that is defined below. This configuration tells the service how to digest the data provided via your CSV file. Note the columns and types align to what was in the file you created above.

```
✓ 21 hours ago (<1s)                                    45

# user_id  item_id   timestamp event_type
interactions_schema = schema = {
    "type": "record",
    "name": "Interactions",
    "namespace": "com.amazonaws.personalize.schema",
    "fields": [
        {
            "name": "user_id",
            "type": "string"
        },
        {
            "name": "item_id",
            "type": "string"
        },
        {
            "name": "timestamp",
            "type": "long"
        },
        {
            "name": "event_type",
            "type": "string"

        }
    ],
    "version": "1.0"
}

create_interactions_schema_response = personalize.create_schema(
    name = "hh_personalize_interactions_schema",
    domain = "ECOMMERCE",
    schema = json.dumps(interactions_schema)
)

interaction_schema_arn = create_interactions_schema_response['schemaArn']
print(json.dumps(create_interactions_schema_response, indent=2))
```

```
{
  "schemaArn": "arn:aws:personalize:ap-southeast-1:202255947274:schema/hh_personalize_interactions_schema",
  "ResponseMetadata": {
    "RequestId": "d5441f81-fdd9-4a10-b8b4-d5818c590475",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 08:17:19 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "105",
```

```
    "connection": "keep-alive",
    "x-amzn-requestid": "d5441f81-fdd9-4a10-b8b4-d5818c590475",
    "strict-transport-security": "max-age=47304000; includeSubDomains",
    "x-frame-options": "DENY",
    "cache-control": "no-cache",
    "x-content-type-options": "nosniff"
   },
   "RetryAttempts": 0
  }
}
```


Create Items (Restaurants) Schema

▶ ✓ 21 hours ago (<1s) 47

```
# item_id,name,city_id,price,primary_cuisine,secondary_cuisines,primary_dining_style,secondary_dining_styles,
primary_location,secondary_locations

items_schema = {
    "type": "record",
    "name": "Items",
    "namespace": "com.amazonaws.personalize.schema",
    "fields": [
        {
            "name": "item_id",
            "type": "string"
        },
        {
            "name": "name",
            "type": "string",
            "textual": True
        },
        {
            "name": "city_id",
            "type": "string",
            "categorical": True
        },
        {
            "name": "price",
            "type": "float"
        },
        {
            "name": "category_l1",
            "type": ["string"],
            "categorical": True
        },
        {
            "name": "category_l2",
            "type": ["string"],
            "categorical": True
        },
        {
            "name": "creation_timestamp",
            "type": "long"
        }
    ],
    "version": "1.0"
}

create_items_schema_response = personalize.create_schema(
    name = "hh_personalize_items_schema",
    domain = "ECOMMERCE",
    schema = json.dumps(items_schema)
)

items_schema_arn = create_items_schema_response['schemaArn']
print(json.dumps(create_items_schema_response, indent=2))
```


```
{
  "schemaArn": "arn:aws:personalize:ap-southeast-1:202255947274:schema/hh_personalize_items_schema",
  "ResponseMetadata": {
    "RequestId": "95bf0728-a583-45a8-9b8e-7a255502e202",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 08:17:29 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "98",
      "connection": "keep-alive",
      "x-amzn-requestid": "95bf0728-a583-45a8-9b8e-7a255502e202",
      "strict-transport-security": "max-age=47304000; includeSubDomains",
      "x-frame-options": "DENY",
      "cache-control": "no-cache",
      "x-content-type-options": "nosniff"
    },
    "RetryAttempts": 0
  }
}
```


Create Users Schema

✓ 21 hours ago (<1s) 49

```
# user_id,gender,loyalty_level_id

users_schema = {
    "type": "record",
    "name": "Users",
    "namespace": "com.amazonaws.personalize.schema",
    "fields": [
        {
            "name": "user_id",
            "type": "string"
        },
        {
            "name": "gender",
            "type": ["string", "null"],
            "categorical": True
        },
        {
            "name": "loyalty_level_id",
            "type": ["string", "null"],
            "categorical": True
        }
    ],
    "version": "1.0"
}

create_users_schema_response = personalize.create_schema(
    name = "hh_personalize_users_schema",
    domain = "ECOMMERCE",
    schema = json.dumps(users_schema)
)

users_schema_arn = create_users_schema_response['schemaArn']
print(json.dumps(create_users_schema_response, indent=2))
```


```
{
  "schemaArn": "arn:aws:personalize:ap-southeast-1:202255947274:schema/hh_personalize_users_schema",
  "ResponseMetadata": {
    "RequestId": "73d36fa0-a160-4e7f-8eb9-bad46c125d73",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 08:17:33 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "98",
      "connection": "keep-alive",
      "x-amzn-requestid": "73d36fa0-a160-4e7f-8eb9-bad46c125d73",
      "strict-transport-security": "max-age=47304000; includeSubDomains",
      "x-frame-options": "DENY",
      "cache-control": "no-cache",
      "x-content-type-options": "nosniff"
    },
    "RetryAttempts": 0
  }
}
```


Create Datasets

After the group, the next thing to create is the actual datasets.

Create Interactions Dataset

✓ 21 hours ago (<1s) 52

```
dataset_type = "INTERACTIONS"

create_dataset_response = personalize.create_dataset(
    name = "hh_personalize_interactions_dataset",
    datasetType = dataset_type,
    datasetGroupArn = dataset_group_arn,
    schemaArn = interaction_schema_arn
)

interactions_dataset_arn = create_dataset_response['datasetArn']
print(json.dumps(create_dataset_response, indent=2))
```


```
{
  "datasetArn": "arn:aws:personalize:ap-southeast-1:202255947274:dataset/hh_personalize_dataset_group/INTERACTIONS",
  "ResponseMetadata": {
    "RequestId": "52589f02-41d0-4877-9883-bd545f2482d0",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 08:18:04 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "114",
      "connection": "keep-alive",
      "x-amzn-requestid": "52589f02-41d0-4877-9883-bd545f2482d0",
```

```
    "strict-transport-security": "max-age=47304000; includeSubDomains",
    "x-frame-options": "DENY",
    "cache-control": "no-cache",
    "x-content-type-options": "nosniff"
   },
   "RetryAttempts": 0
  }
}
```


Create Items Dataset

```
▶  ✓ 21 hours ago (<1s)                                    54

dataset_type = "ITEMS"

create_dataset_response = personalize.create_dataset(
    name = "hh_personalize_items_dataset",
    datasetType = dataset_type,
    datasetGroupArn = dataset_group_arn,
    schemaArn = items_schema_arn
)

items_dataset_arn = create_dataset_response['datasetArn']
print(json.dumps(create_dataset_response, indent=2))
```


```
{
  "datasetArn": "arn:aws:personalize:ap-southeast-1:202255947274:dataset/hh_personalize_dataset_group/ITEMS",
  "ResponseMetadata": {
    "RequestId": "bc99b548-68c5-4222-91f9-acbdad6567e6",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 08:18:25 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "107",
      "connection": "keep-alive",
      "x-amzn-requestid": "bc99b548-68c5-4222-91f9-acbdad6567e6",
      "strict-transport-security": "max-age=47304000; includeSubDomains",
      "x-frame-options": "DENY",
      "cache-control": "no-cache",
      "x-content-type-options": "nosniff"
    },
    "RetryAttempts": 0
  }
}
```


Create Users Dataset

```
▶  ✓ 21 hours ago (<1s)                                    56

dataset_type = "USERS"
dataset_name = "hh_personalize_users_dataset"

# Function to get the dataset ARN if it exists
def get_existing_dataset_arn(dataset_group_arn, dataset_name):
    response = personalize.list_datasets(
        datasetGroupArn=dataset_group_arn
    )
    for dataset in response['datasets']:
        if dataset['name'] == dataset_name:
            return dataset['datasetArn']
    return None

# Check if the dataset already exists
users_dataset_arn = get_existing_dataset_arn(dataset_group_arn, dataset_name)

if users_dataset_arn:
    print(f"Dataset already exists: {users_dataset_arn}")
else:
    # Create the dataset if it does not exist
    create_dataset_response = personalize.create_dataset(
        name=dataset_name,
        datasetType=dataset_type,
        datasetGroupArn=dataset_group_arn,
        schemaArn=users_schema_arn
    )
    users_dataset_arn = create_dataset_response['datasetArn']
    print("Created new dataset:")
    print(json.dumps(create_dataset_response, indent=2))

print(f"Final dataset ARN: {users_dataset_arn}")
```


```
Created new dataset:
{
  "datasetArn": "arn:aws:personalize:ap-southeast-1:202255947274:dataset/hh_personalize_dataset_group/USERS",
  "ResponseMetadata": {
    "RequestId": "f5fcd915-7704-4e7f-8084-7958e2720b09",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
```

```
    "date": "Mon, 03 Feb 2025 08:18:33 GMT",
    "content-type": "application/x-amz-json-1.1",
    "content-length": "107",
    "connection": "keep-alive",
    "x-amzn-requestid": "f5fcd915-7704-4e7f-8084-7958e2720b09",
    "strict-transport-security": "max-age=47304000; includeSubDomains",
    "x-frame-options": "DENY",
    "cache-control": "no-cache",
    "x-content-type-options": "nosniff"
   },
   "RetryAttempts": 0
  }
}
Final dataset ARN: arn:aws:personalize:ap-southeast-1:202255947274:dataset/hh_personalize_dataset_group/USERS
```


Create Personalize Role

Also Amazon Personalize needs the ability to assume Roles in AWS in order to have the permissions to execute certain tasks, the lines below grant that.

Note: Make sure the role you are using to run the code in this notebook has the necessary permissions to create a role.

▶ ✓ 19 hours ago (1m) 58

```python
from botocore.exceptions import ClientError
iam = boto3.client("iam")

role_name = "HHPersonalizeRole"
assume_role_policy_document = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "personalize.amazonaws.com",
                    "opensearch.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}

def get_existing_role_arn(role_name):
    try:
        response = iam.get_role(RoleName=role_name)
        return response['Role']['Arn']
    except ClientError as e:
        if e.response['Error']['Code'] == 'NoSuchEntity':
            return None
        else:
            raise

# Check if the role already exists
role_arn = get_existing_role_arn(role_name)

if role_arn:
    print(f"Role already exists: {role_arn}")
else:
    # Create the role if it doesn't exist
    create_role_response = iam.create_role(
        RoleName=role_name,
        AssumeRolePolicyDocument=json.dumps(assume_role_policy_document)
    )

    # AmazonPersonalizeFullAccess provides access to any S3 bucket with "personalize" in the name
    policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonPersonalizeFullAccess"
    iam.attach_role_policy(
        RoleName=role_name,
        PolicyArn=policy_arn
    )

    # Attach Amazon S3 full access policy to the role
    iam.attach_role_policy(
        PolicyArn='arn:aws:iam::aws:policy/AmazonS3FullAccess',
        RoleName=role_name
    )

    print("Role created; waiting for it to propagate")
    time.sleep(60)  # Wait to allow IAM role policy attachment to propagate

    role_arn = create_role_response["Role"]["Arn"]
    print(f"New role created: {role_arn}")

print(f"Final role ARN: {role_arn}")
```

```
Role created; waiting for it to propagate
New role created: arn:aws:iam::202255947274:role/HHPersonalizeRole
Final role ARN: arn:aws:iam::202255947274:role/HHPersonalizeRole
```

Import the data

Earlier you created the DatasetGroup and Dataset to house your information, now you will execute an import job that will load the data from S3 into Amazon Personalize for usage building your model.

Create Interactions Dataset Import Job

▶ ✓ 19 hours ago (<1s) 60

```
create_interactions_dataset_import_job_response = personalize.create_dataset_import_job(
    jobName = "hh_personalize_interactions_import_job",
    datasetArn = interactions_dataset_arn,
    dataSource = {
        "dataLocation": "s3://{}/{}".format(S3_BUCKET_NAME, S3_INTERACTIONS_PREFIX)
    },
    roleArn = role_arn
)

dataset_interactions_import_job_arn = create_interactions_dataset_import_job_response['datasetImportJobArn']
print(json.dumps(create_interactions_dataset_import_job_response, indent=2))
```


```
{
  "datasetImportJobArn": "arn:aws:personalize:ap-southeast-1:202255947274:dataset-import-job/hh_personalize_interactions_impo
rt_job",
  "ResponseMetadata": {
    "RequestId": "adf3e7e1-1ae5-4515-b5b1-48dce7550d10",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 10:18:24 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "131",
      "connection": "keep-alive",
      "x-amzn-requestid": "adf3e7e1-1ae5-4515-b5b1-48dce7550d10",
      "strict-transport-security": "max-age=47304000; includeSubDomains",
      "x-frame-options": "DENY",
      "cache-control": "no-cache",
      "x-content-type-options": "nosniff"
    },
    "RetryAttempts": 0
  }
}
```


Wait for Dataset Import Job to Have ACTIVE Status It can take a while before the import job completes, please wait until you see that it is active below.

▶ ✓ 19 hours ago (4m) 62

```
%%time

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    describe_dataset_import_job_response = personalize.describe_dataset_import_job(
        datasetImportJobArn = dataset_interactions_import_job_arn
    )
    status = describe_dataset_import_job_response["datasetImportJob"]['status']
    print("DatasetImportJob: {}".format(status))

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

    time.sleep(60)
```

```
DatasetImportJob: CREATE PENDING
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: ACTIVE
CPU times: user 748 ms, sys: 111 ms, total: 859 ms
Wall time: 4min
```

Create Items Dataset Import Job

▶ ✓ 19 hours ago (<1s) 64

```
create_items_dataset_import_job_response = personalize.create_dataset_import_job(
    jobName = "hh_personalize_items_import_job",
    datasetArn = items_dataset_arn,
    dataSource = {
        "dataLocation": "s3://{}/{}".format(S3_BUCKET_NAME, S3_ITEMS_PREFIX)
    },
    roleArn = role_arn
)

dataset_items_import_job_arn = create_items_dataset_import_job_response['datasetImportJobArn']
print(json.dumps(create_items_dataset_import_job_response, indent=2))
```

```
{
  "datasetImportJobArn": "arn:aws:personalize:ap-southeast-1:202255947274:dataset-import-job/hh_personalize_items_import_job",
  "ResponseMetadata": {
    "RequestId": "49bb3597-c159-4443-b2de-9c37f95dbf36",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 10:31:20 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "124",
      "connection": "keep-alive",
      "x-amzn-requestid": "49bb3597-c159-4443-b2de-9c37f95dbf36",
      "strict-transport-security": "max-age=47304000; includeSubDomains",
      "x-frame-options": "DENY",
      "cache-control": "no-cache",
      "x-content-type-options": "nosniff"
    },
    "RetryAttempts": 0
  }
}
```


Wait for Dataset Import Job to Have ACTIVE Status

19 hours ago (6m) 66

```
%%time

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    describe_dataset_import_job_response = personalize.describe_dataset_import_job(
        datasetImportJobArn = dataset_items_import_job_arn
    )
    status = describe_dataset_import_job_response["datasetImportJob"]['status']
    print("DatasetImportJob: {}".format(status))

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

    time.sleep(60)
```

```
DatasetImportJob: CREATE PENDING
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: ACTIVE
CPU times: user 776 ms, sys: 86.9 ms, total: 863 ms
Wall time: 6min 1s
```

Create Users Dataset Import Job

19 hours ago (<1s) 68

```
create_users_dataset_import_job_response = personalize.create_dataset_import_job(
    jobName = "hh_personalize_users_import_job",
    datasetArn = users_dataset_arn,
    dataSource = {
        "dataLocation": "s3://{}/{}".format(S3_BUCKET_NAME, S3_USERS_PREFIX)
    },
    roleArn = role_arn
)

dataset_users_import_job_arn = create_users_dataset_import_job_response['datasetImportJobArn']
print(json.dumps(create_users_dataset_import_job_response, indent=2))
```


```
{
  "datasetImportJobArn": "arn:aws:personalize:ap-southeast-1:202255947274:dataset-import-job/hh_personalize_users_import_job",
  "ResponseMetadata": {
    "RequestId": "6e950c2f-93dc-4f0d-88a0-ab8819131082",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "date": "Mon, 03 Feb 2025 10:45:27 GMT",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "124",
      "connection": "keep-alive",
      "x-amzn-requestid": "6e950c2f-93dc-4f0d-88a0-ab8819131082",
      "strict-transport-security": "max-age=47304000; includeSubDomains",
      "x-frame-options": "DENY",
      "cache-control": "no-cache",
      "x-content-type-options": "nosniff"
    },
    "RetryAttempts": 0
  }
}
```

19 hours ago (5m) 69

```
%%time

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    describe_dataset_import_job_response = personalize.describe_dataset_import_job(
        datasetImportJobArn = dataset_users_import_job_arn
    )
    status = describe_dataset_import_job_response["datasetImportJob"]['status']
    print("DatasetImportJob: {}".format(status))

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

    time.sleep(60)
```

```
DatasetImportJob: CREATE PENDING
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: CREATE IN_PROGRESS
DatasetImportJob: ACTIVE
CPU times: user 1.22 s, sys: 410 ms, total: 1.63 s
Wall time: 5min
```

Choose a recommender use cases

Each domain has different use cases. When you create a recommender you create it for a specific use case, and each use case has different requirements for getting recommendations.

19 hours ago (<1s) 71

```
ecommerce_recipes = personalize.list_recipes(domain='ECOMMERCE') # See a list of recommenders for the domain.
display (ecommerce_recipes['recipes'])
```

```
 'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 182000, tzinfo=tzlocal()),
 'domain': 'ECOMMERCE'},
{'name': 'aws-ecomm-popular-items-by-purchases',
 'recipeArn': 'arn:aws:personalize:::recipe/aws-ecomm-popular-items-by-purchases',
 'status': 'ACTIVE',
 'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 182000, tzinfo=tzlocal()),
 'domain': 'ECOMMERCE'},
{'name': 'aws-ecomm-popular-items-by-views',
 'recipeArn': 'arn:aws:personalize:::recipe/aws-ecomm-popular-items-by-views',
 'status': 'ACTIVE',
 'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 182000, tzinfo=tzlocal()),
 'domain': 'ECOMMERCE'},
{'name': 'aws-ecomm-recommended-for-you',
 'recipeArn': 'arn:aws:personalize:::recipe/aws-ecomm-recommended-for-you',
 'status': 'ACTIVE',
 'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 183000, tzinfo=tzlocal()),
 'domain': 'ECOMMERCE'}]
```

19 hours ago (<1s) 72

```
available_recipes = personalize.list_recipes() # See a list of all recipes.
display (available_recipes['recipes'])
```

```
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 182000, tzinfo=tzlocal())},
{'name': 'aws-personalized-ranking',
 'recipeArn': 'arn:aws:personalize:::recipe/aws-personalized-ranking',
 'status': 'ACTIVE',
 'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 182000, tzinfo=tzlocal())},
{'name': 'aws-personalized-ranking-v2',
 'recipeArn': 'arn:aws:personalize:::recipe/aws-personalized-ranking-v2',
 'status': 'ACTIVE',
 'creationDateTime': datetime.datetime(2024, 1, 1, 0, 0, tzinfo=tzlocal()),
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 182000, tzinfo=tzlocal())},
{'name': 'aws-popularity-count',
 'recipeArn': 'arn:aws:personalize:::recipe/aws-popularity-count',
 'status': 'ACTIVE',
 'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 182000, tzinfo=tzlocal())},
{'name': 'aws-similar-items',
 'recipeArn': 'arn:aws:personalize:::recipe/aws-similar-items',
 'status': 'ACTIVE',
 'creationDateTime': datetime.datetime(2019, 6, 10, 0, 0, tzinfo=tzlocal()),
 'lastUpdatedDateTime': datetime.datetime(2024, 6, 19, 19, 15, 10, 182000, tzinfo=tzlocal())}]
```

We are going to create a recommender of the type "Customers who viewed X also viewed". This recommender gives recommendations for items that customers also viewed based on an item that you specify. With this use case, Amazon Personalize automatically filters items the user purchased based on the userId that you specify and `Purchase` events.

74

```
create_recommender_response = personalize.create_recommender(
  name = 'hh_personalize_viewed_x_also_viewed',
  recipeArn = 'arn:aws:personalize:::recipe/aws-ecomm-customers-who-viewed-x-also-viewed',
  datasetGroupArn = dataset_group_arn
)
viewed_x_also_viewed_arn = create_recommender_response["recommenderArn"]
print (json.dumps(create_recommender_response))
```

We wait until the recomenders have finished creating and have status `ACTIVE`. We check periodically on the status of the recommender

```
76
%%time

max_time = time.time() + 5*60*60 # 5 hours

while time.time() < max_time:

    version_response = personalize.describe_recommender(
        recommenderArn = viewed_x_also_viewed_arn
    )
    status = version_response["recommender"]["status"]

    if status == "ACTIVE":
        print("Build succeeded for {}".format(viewed_x_also_viewed_arn))

    elif status == "CREATE FAILED":
        print("Build failed for {}".format(viewed_x_also_viewed_arn))


    if status == "ACTIVE" or status == "CREATE FAILED":
        break
    else:
        print('The "Customers who viewed X also viewed" Recommender build is still in progress')

    time.sleep(60)
```

We are going to create a second recommender of the type "Recommended For You". This type of recommender offers personalized recommendations for items based on a user that you specify. With this use case, Amazon Personalize automatically filters items the user purchased based on the userId that you specify and `Purchase` events.

More use cases per domain

```
✓ 18 hours ago (<1s)
78
create_recommender_response = personalize.create_recommender(
  name = 'hh_personalize_recommended_for_you',
  recipeArn = 'arn:aws:personalize:::recipe/aws-ecomm-recommended-for-you',
  datasetGroupArn = dataset_group_arn
)
recommended_for_you_arn = create_recommender_response["recommenderArn"]
print (json.dumps(create_recommender_response))
```

```
{"recommenderArn": "arn:aws:personalize:ap-southeast-1:202255947274:recommender/hh_personalize_recommended_for_you", "ResponseMetadata": {"RequestId": "477f5d18-f06d-42d3-b78b-9cc31dc87520", "HTTPStatusCode": 200, "HTTPHeaders": {"date": "Mon, 03 Feb 2025 10:53:11 GMT", "content-type": "application/x-amz-json-1.1", "content-length": "115", "connection": "keep-alive", "x-amzn-requestid": "477f5d18-f06d-42d3-b78b-9cc31dc87520", "strict-transport-security": "max-age=47304000; includeSubDomains", "x-frame-options": "DENY", "cache-control": "no-cache", "x-content-type-options": "nosniff"}, "RetryAttempts": 0}}
```

We wait until the recomenders have finished creating and have status `ACTIVE`. We check periodically on the status of the recommender

```
✓ 18 hours ago 1h 13m
80
%%time

max_time = time.time() + 5*60*60 # 5 hours

while time.time() < max_time:

    version_response = personalize.describe_recommender(
        recommenderArn = recommended_for_you_arn
    )
    status = version_response["recommender"]["status"]

    if status == "ACTIVE":
        print("Build succeeded for {}".format(recommended_for_you_arn))

    elif status == "CREATE FAILED":
        print("Build failed for {}".format(recommended_for_you_arn))
        break
```

```
    if status == "ACTIVE" or status == "CREATE FAILED":
        break
    else:
        print('The "Recommended for you" Recommender build is still in progress')

    time.sleep(60)
```

```
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
The "Recommended for you" Recommender build is still in progress
Build succeeded for arn:aws:personalize:ap-southeast-1:202255947274:recommender/hh_personalize_recommended_for_you
CPU times: user 6.28 s, sys: 1.43 s, total: 7.71 s
Wall time: 1h 13min 11s
```

Personalize Ranking

▶ 82

```
rerank_create_solution_response = personalize.create_solution(
    name = "hh_personalize_ranking_solution",
    datasetGroupArn = dataset_group_arn,
    recipeArn = "arn:aws:personalize:::recipe/aws-personalized-ranking-v2"
)

rerank_solution_arn = rerank_create_solution_response['solutionArn']
print(json.dumps(rerank_create_solution_response, indent=2))
```

▶ 83

```
rerank_create_solution_version_response = personalize.create_solution_version(
    solutionArn = rerank_solution_arn
)

rerank_solution_version_arn = rerank_create_solution_version_response['solutionVersionArn']
print(json.dumps(rerank_create_solution_version_response, indent=2))
```

▶ 84

```
in_progress_solution_versions = [
    rerank_solution_version_arn
]

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    for solution_version_arn in in_progress_solution_versions:
        version_response = personalize.describe_solution_version(
            solutionVersionArn = solution_version_arn
        )
        status = version_response["solutionVersion"]["status"]

        if status == "ACTIVE":
            print("Build succeeded for {}".format(solution_version_arn))
            in_progress_solution_versions.remove(solution_version_arn)
        elif status == "CREATE FAILED":
            print("Build failed for {}".format(solution_version_arn))
            in_progress_solution_versions.remove(solution_version_arn)

    if len(in_progress_solution_versions) <= 0:
        break
    else:
        print("At least one solution build is still in progress")

    time.sleep(60)
```

▶ 85

```
rerank_create_campaign_response = personalize.create_campaign(
    name = "hh_personalize_ranking_campaign",
    solutionVersionArn = rerank_solution_version_arn,
    minProvisionedTPS = 1
)
```

```
rerank_campaign_arn = rerank_create_campaign_response['campaignArn']
print(json.dumps(rerank_create_campaign_response, indent=2))
```

```
                                                                    86
in_progress_campaigns = [
    rerank_campaign_arn
]

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    for campaign_arn in in_progress_campaigns:
        version_response = personalize.describe_campaign(
            campaignArn = campaign_arn
        )
        status = version_response["campaign"]["status"]

        if status == "ACTIVE":
            print("Build succeeded for {}".format(campaign_arn))
            in_progress_campaigns.remove(campaign_arn)
        elif status == "CREATE FAILED":
            print("Build failed for {}".format(campaign_arn))
            in_progress_campaigns.remove(campaign_arn)

    if len(in_progress_campaigns) <= 0:
        break
    else:
        print("At least one campaign build is still in progress")

    time.sleep(60)
```

Getting recommendations with a recommender

Now that the recommenders have been trained, lets have a look at the recommendations we can get for our users!

```
                                                                    88
# reading the original data in order to have a dataframe that has both item_ids
# and the corresponding titles to make out recommendations easier to read.
# items_df = pd.read_csv('./items.csv')
# items_df.sample(10)
```

```
                                                                    89
# List all objects in the folder
prefix = S3_INTERACTIONS_PREFIX
# prefix = S3_USERS_PREFIX
# prefix = S3_ITEMS_PREFIX
response = s3.list_objects_v2(Bucket=S3_BUCKET_NAME, Prefix=prefix)

if 'Contents' in response:
    # Get all file objects
    files = response['Contents']

    # Sort the files by last modified timestamp
    files = sorted(files, key=lambda x: x['LastModified'])

    # Get the last file
    last_file = files[-1]['Key']

    # Read the CSV directly into a pandas DataFrame
    s3_url = f"s3://{S3_BUCKET_NAME}/{last_file}"
    items_df = pd.read_csv(s3_url)
    items_df.sample(10)
    print(f"Last file read: {s3_url}")
    print(items_df.head())
else:
    print("No files found in the specified folder.")
```

```
                                                                    90
def get_item_by_id(item_id, item_df):
    """
    This takes in an item_id from a recommendation in string format,
    converts it to an int, and then does a lookup in a default or specified
    dataframe and returns the item description.

    A really broad try/except clause was added in case anything goes wrong.

    Feel free to add more debugging or filtering here to improve results if
    you hit an error.
    """
    try:
        return items_df.loc[items_df["item_id"]==str(item_id)]['name'].values[0]
```

```
    except:
        print (item_id)
        return "Error obtaining item description"
```

Let us get some recommendations using the "Customers who viewed X also viewed" Recommender:

92

```
# use a random valid id for a quick sanity check, modify the line of code bellow to a valid id in your dataset
get_item_by_id("1645", items_df)
```

93

```
# First pick a user
test_user_id = "189034"

# Select a random item
test_item_id = "1645" # a random item: 1645

# Get recommendations for the user for this item
get_recommendations_response = personalize_runtime.get_recommendations(
    recommenderArn = viewed_x_also_viewed_arn,
    itemId = test_item_id,
    userId = test_user_id,
    numResults = 10
)

# Build a new dataframe for the recommendations
item_list = get_recommendations_response['itemList']
recommendation_list = []

for item in item_list:
    item = get_item_by_id(item['itemId'], items_df)
    recommendation_list.append(item)

user_recommendations_df = pd.DataFrame(recommendation_list, columns = [get_item_by_id(test_item_id, items_df)])

pd.options.display.max_rows =10
display(user_recommendations_df)
```

Get recommendations from the recommender returning "Recommended for you":

95

```
# First pick a user
test_user_id = "189034"
recommended_for_you_arn = "arn:aws:personalize:ap-southeast-1:079994049689:recommender/
hh_personalize_recommended_for_you"

# Get recommendations for the user
get_recommendations_response = personalize_runtime.get_recommendations(
    recommenderArn = recommended_for_you_arn,
    userId = test_user_id,
    numResults = 20
)

print(json.dumps(get_recommendations_response, indent=2))
# Build a new dataframe for the recommendations
item_list = get_recommendations_response['itemList']
print(json.dumps(item_list, indent=2))
# recommendation_list = []

# for item in item_list:
#     item = get_item_by_id(item['itemId'], items_df)
#     recommendation_list.append(item)


# user_recommendations_df = pd.DataFrame(recommendation_list, columns = [test_user_id])

# pd.options.display.max_rows =20
# display(user_recommendations_df)
```

Real time events

Start by creating an event tracker that is attached to the campaign.

✓ 17 hours ago (<1s) 97

```
response = personalize.create_event_tracker(
```

```
    name='HHPersonalizeEventsTracker',
    datasetGroupArn=dataset_group_arn
)
print(response['eventTrackerArn'])
print(response['trackingId'])
TRACKING_ID = response['trackingId']
event_tracker_arn = response['eventTrackerArn']
```

```
arn:aws:personalize:ap-southeast-1:202255947274:event-tracker/73b04d61
c43fbbed-21b9-46cf-a990-8fa5aff35db9
```

▶ ✓ 17 hours ago (<1s) 98

```
response = personalize.list_event_trackers(
  datasetGroupArn=dataset_group_arn
)

event_trackers = response['eventTrackers']
for tracker in event_trackers:
  print(json.dumps(tracker, indent=2, default=str))
```

```
{
  "name": "HHPersonalizeEventsTracker",
  "eventTrackerArn": "arn:aws:personalize:ap-southeast-1:202255947274:event-tracker/73b04d61",
  "status": "CREATE PENDING",
  "creationDateTime": "2025-02-03 12:38:09.493000+00:00",
  "lastUpdatedDateTime": "2025-02-03 12:38:09.493000+00:00"
}
```

Create Filters

Create filter by city_id

▶ ✓ 17 hours ago (<1s) 100

```
create_filter_city_id_response = personalize.create_filter(
    name = 'personalize-auto-context-city-id-filter',
    datasetGroupArn = dataset_group_arn,
    filterExpression = 'INCLUDE ItemID WHERE Items.city_id IN ($CONTEXT_CITY_ID)'
)

filter_city_id_arn = create_filter_city_id_response["filterArn"]
print("Filter ARN: " + filter_city_id_arn)
```

```
Filter ARN: arn:aws:personalize:ap-southeast-1:202255947274:filter/personalize-auto-context-city-id-filter
```

▶ ✓ 17 hours ago (15s) 101

```
%%time
max_time = time.time() + 10*60*60 # 10 hours
while time.time() < max_time:
    version_response = personalize.describe_filter(
        filterArn = filter_city_id_arn
    )
    status = version_response["filter"]["status"]
    if status == "ACTIVE":
        print("Build succeeded for {}".format(filter_city_id_arn))

    elif status == "CREATE FAILED":
        print("Build failed for {}".format(filter_city_id_arn))
        break

    if status == "ACTIVE" or status == "CREATE FAILED":
        break
    else:
        print('The Filter build is still in progress')
    time.sleep(15)
```

```
The Filter build is still in progress
Build succeeded for arn:aws:personalize:ap-southeast-1:202255947274:filter/personalize-auto-context-city-id-filter
CPU times: user 270 ms, sys: 0 ns, total: 270 ms
Wall time: 15.1 s
```

▶ 102

```
# First pick a user
test_user_id = "189034"
context_city_id = "9" # 1: bankok, 9: singapore
recommended_for_you_arn = "arn:aws:personalize:ap-southeast-1:079994049689:recommender/
hh_personalize_recommended_for_you"

# Get recommendations for the user
get_recommendations_response = personalize_runtime.get_recommendations(
    recommenderArn = recommended_for_you_arn,
    numResults = 20,
    userId = test_user_id,
    filterArn = filter_city_id_arn,
    filterValues={"CONTEXT_CITY_ID" : f"\"{context_city_id}\""}
)
```

```
print(json.dumps(get_recommendations_response, indent=2))
# Build a new dataframe for the recommendations
item_list = get_recommendations_response['itemList']
print(json.dumps(item_list, indent=2))
```

Review

Using the codes above you have successfully trained a deep learning model to generate item recommendations based on prior user behavior. You have created two recommenders for two foundational use cases. Going forward, you can adapt this code to create other recommenders.

Notes for the Next Notebook:

There are a few values you will need for the next notebook, execute the cell below to store them so they can be used in the `Clean_Up_Resources.ipynb` notebook.

This will overwite any data stored for those variables and set them to the values specified in this notebook.

```
✓ 17 hours ago (<1s)                                                    105

print("- dataset_group_arn: ", dataset_group_arn)
print("- role_arn: ", role_arn)
print("- region: ", AWS_REGION)
print("\n")
# print("- viewed_x_also_viewed_arn: ", viewed_x_also_viewed_arn)
print("- recommended_for_you_arn: ", recommended_for_you_arn)
print("- filter_city_id_arn: ", filter_city_id_arn)
print("- event_tracker_arn: ", event_tracker_arn)
print("- TRACKING_ID: ", TRACKING_ID)
```

```
- dataset_group_arn:  arn:aws:personalize:ap-southeast-1:202255947274:dataset-group/hh_personalize_dataset_group
- role_arn:  arn:aws:iam::202255947274:role/HHPersonalizeRole
- region:  ap-southeast-1


- recommended_for_you_arn:  arn:aws:personalize:ap-southeast-1:202255947274:recommender/hh_personalize_recommended_for_you
- filter_city_id_arn:  arn:aws:personalize:ap-southeast-1:202255947274:filter/personalize-auto-context-city-id-filter
- event_tracker_arn:  arn:aws:personalize:ap-southeast-1:202255947274:event-tracker/73b04d61
- TRACKING_ID:  c43fbbed-21b9-46cf-a990-8fa5aff35db9
```

Store Variables for Next Notebook

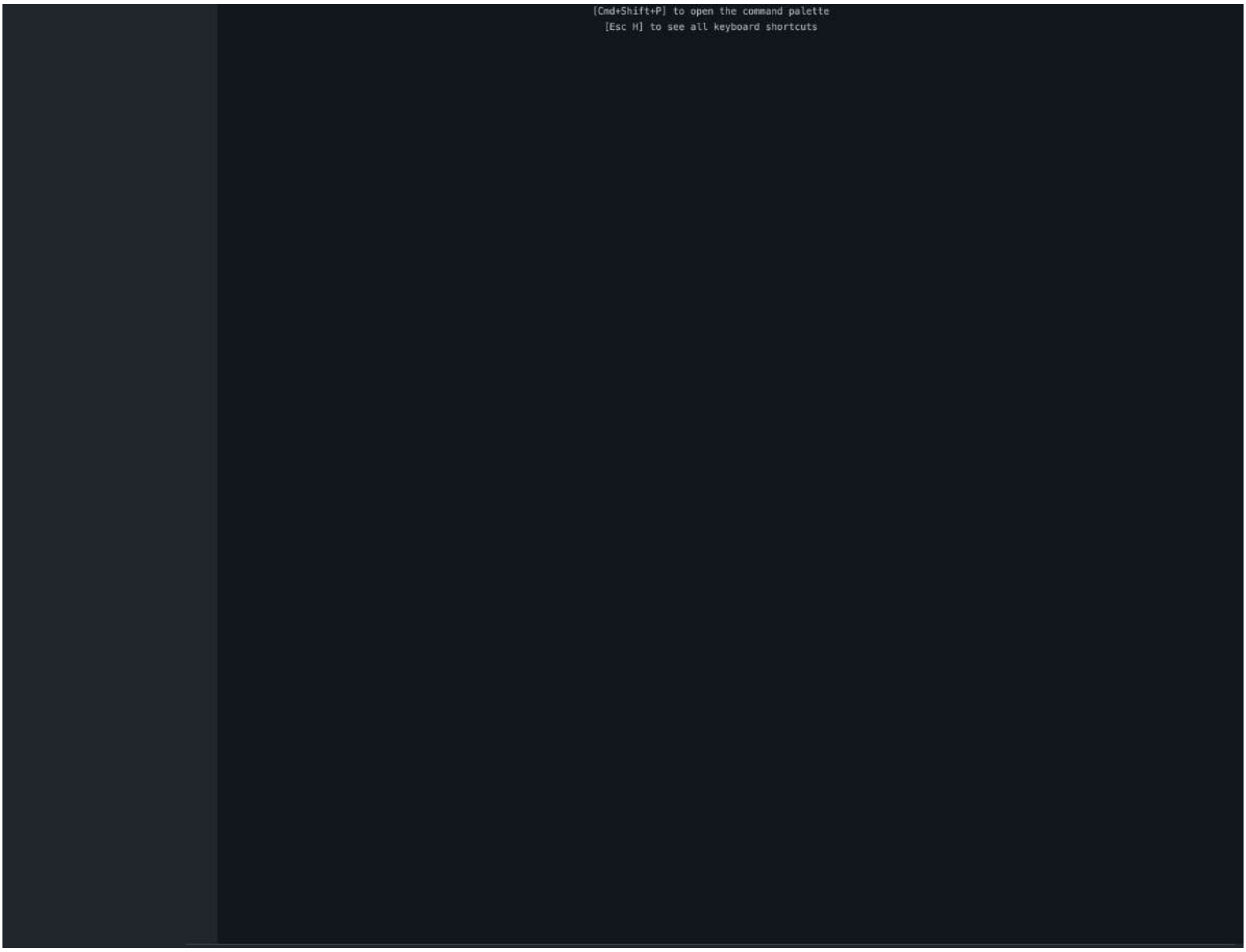
Staging

- dataset_group_arn: arn:aws:personalize:ap-southeast-1:079994049689:dataset-group/hh_personalize_dataset_group
- role_arn: arn:aws:iam::079994049689:role/HHPersonalizeRole
- region: ap-southeast-1
- viewed_x_also_viewed_arn: arn:aws:personalize:ap-southeast-1:079994049689:recommender/hh_personalize_viewed_x_also_viewed
- recommended_for_you_arn: arn:aws:personalize:ap-southeast-1:079994049689:recommender/hh_personalize_recommended_for_you
- event_tracker_arn: arn:aws:personalize:ap-southeast-1:079994049689:event-tracker/292a44ac
- TRACKING_ID: eb454454-1aec-41c8-95d4-5745255b18a5

Production

- dataset_group_arn: arn:aws:personalize:ap-southeast-1:202255947274:dataset-group/hh_personalize_dataset_group
- role_arn: arn:aws:iam::202255947274:role/HHPersonalizeRole
- region: ap-southeast-1
- recommended_for_you_arn: arn:aws:personalize:ap-southeast-1:202255947274:recommender/hh_personalize_recommended_for_you
- filter_city_id_arn: arn:aws:personalize:ap-southeast-1:202255947274:filter/personalize-auto-context-city-id-filter
- event_tracker_arn: arn:aws:personalize:ap-southeast-1:202255947274:event-tracker/73b04d61
- TRACKING_ID: c43fbbed-21b9-46cf-a990-8fa5aff35db9

If you have run the `Building_Your_First_Recommender_Video_On_Demand.ipynb` notebook, please make sure you re-run the previous step in the `Building_Your_First_Recommender_Video_On_Demand.ipynb` notebook and re-run the `Clean_Up_Resources.ipynb` to remove the resources created in that notebook after you run the `Clean_Up_Resources.ipynb` with the resources created here.


[Cmd+Shift+P] to open the command palette
[Esc H] to see all keyboard shortcuts