Validating and Importing User-Item-Interaction Data

In this notebook, you will choose a dataset and prepare it for use with Amazon Personalize.

Introduction

For the most part, the algorithms in Amazon Personalize (called recipes) look to solve different tasks, explained here:

1. **User Personalization** - New release that supports ALL HRNN workflows / user personalization needs, it will be what we use here.
2. **HRNN & HRNN-Metadata** - Recommends items based on previous user interactions with items.
3. **HRNN-Coldstart** - Recommends new items for which interaction data is not yet available.
4. **Personalized-Ranking** - Takes a collection of items and then orders them in probable order of interest using an HRNN-like approach.
5. **SIMS (Similar Items)** - Given one item, recommends other items also interacted with by users.
6. **Popularity-Count** - Recommends the most popular items, if HRNN or HRNN-Metadata do not have an answer - this is returned by default.

No matter the use case, the algorithms all share a base of learning on user-item-interaction data which is defined by 3 core attributes:

1. **UserID** - The user who interacted
2. **ItemID** - The item the user interacted with
3. **Timestamp** - The time at which the interaction occurred

We also support event types and event values defined by:

1. **Event Type** - Categorical label of an event (browse, purchased, rated, etc).
2. **Event Value** - A value corresponding to the event type that occurred. Generally speaking, we look for normalized values between 0 and 1 over the event types. For example, if there are three phases to complete a transaction (clicked, added-to-cart, and purchased), then there would be an event_value for each phase as 0.33, 0.66, and 1.0 respectfully.

The event type and event value fields are additional data which can be used to filter the data sent for training the personalization model. In this particular exercise we will not have an event type or event value.

Choose a dataset or data source

Back to top

As we mentioned, the user-item-interaction data is key for getting started with the service. This means we need to look for use cases that generate that kind of data, a few common examples are:

1. Video-on-demand applications
2. E-commerce platforms
3. Social media aggregators / platforms

There are a few guidelines for scoping a problem suitable for Personalize. We recommend the values below as a starting point, although the official limits (https://docs.aws.amazon.com/personalize/latest/dg/limits.html) lie a little lower.

- Authenticated users
- At least 50 unique users
- At least 100 unique items
- At least 2 dozen interactions for each user

Most of the time this is easily attainable, and if you are low in one category, you can often make up for it by having a larger number in another category.

Generally speaking your data will not arrive in a perfect form for Personalize, and will take some modification to be structured correctly. This notebook looks to guide you through all of that.

Open and Explore the Simulated Retail Interactions Dataset

In [76]:

```
!export PATH=/Library/TeX/texbin:$PATH
#!pip install xelatex
```

In [33]:

```
# Import Dependencies

import boto3
import json
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import time
import requests
import csv
import sys
import botocore
import uuid
from collections import defaultdict
import random
import numpy as np
import yaml

from packaging import version
from botocore.exceptions import ClientError
from pathlib import Path

%matplotlib inline

# Setup Clients

personalize = boto3.client('personalize')
personalize_runtime = boto3.client('personalize-runtime')
personalize_events = boto3.client('personalize-events')

# We will upload our training data in these files:
raw_items_filename = "../../automation/ml_ops/domain/Retail/data/Items/items.csv"                # Do Not Change
raw_users_filename = "../../automation/ml_ops/domain/Retail/data/Users/users.csv"                # Do Not Change
raw_interactions_filename = "../../automation/ml_ops/domain/Retail/data/Interactions/interactions.csv"  # Do Not Change
items_filename = "items.csv"                # Do Not Change
users_filename = "users.csv"                # Do Not Change
interactions_filename = "interactions.csv"  # Do Not Change
```

First let us see a few lines of the raw CSV data:

- An EVENT_TYPE column which can be used to train different Personalize campaigns and also to filter on recommendations.
- The custom DISCOUNT column which is a contextual metadata field, that Personalize reranking and user recommendation campaigns can take into account to guess on the best next product.

In [45]:

```
interactions_df = pd.read_csv(raw_interactions_filename)
interactions_df.head()
```

Out[45]:

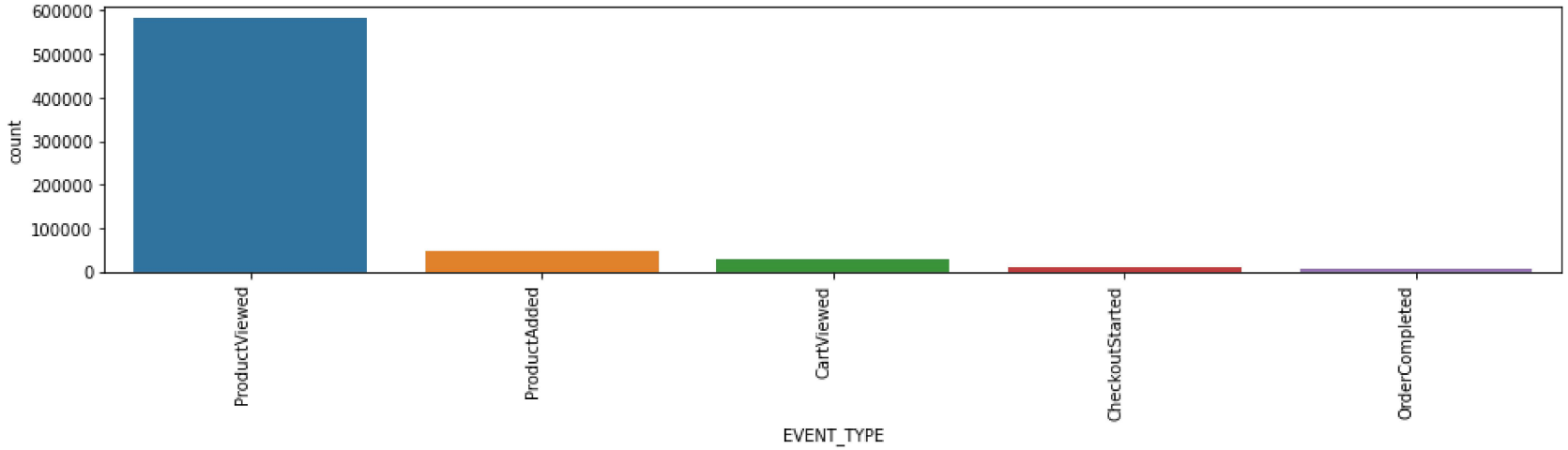
	ITEM_ID	USER_ID	EVENT_TYPE	TIMESTAMP	DISCOUNT
0	1def0093-96b2-4cc4-a022-071941f75b92	3156	ProductViewed	1591803788	No
1	1def0093-96b2-4cc4-a022-071941f75b92	3156	ProductViewed	1591803788	No
2	4df77d59-732e-4194-b9aa-7ad3878345e7	332	ProductViewed	1591803812	Yes
3	4df77d59-732e-4194-b9aa-7ad3878345e7	332	ProductViewed	1591803812	Yes
4	31b83eb4-bd8a-4b5a-87ff-f52abe6aa1f4	3981	ProductViewed	1591803830	Yes

Chart the counts of each `EVENT_TYPE` generated for the interactions dataset. We're simulating a site where visitors heavily view/browse products and to a lesser degree add products to their cart and checkout.

In [39]:

```
categorical_attributes = interactions_df.select_dtypes(include = ['object'])

plt.figure(figsize=(16,3))
chart = sns.countplot(data = categorical_attributes, x = 'EVENT_TYPE')
plt.xticks(rotation=90, horizontalalignment='right')
plt.show()
```



Explore the Users Dataset

In [43]:

```
users_df = pd.read_csv(raw_users_filename)
pd.set_option('display.max_rows', 5)

users_df.head()
```

Out[43]:

	USER_ID	AGE	GENDER
0	1	31	M
1	2	58	F
2	3	43	M
3	4	38	M
4	5	24	M

We can also leverage some user metadata to get some additional information about our users.

In [47]:

```
user_metadata_file_name = '../../automation/ml_ops/domain/Retail/metadata/Users/users.json'
zipped_file_name = user_metadata_file_name+'.gz'

# this will delete the zipped file. -k is not supported with the current version of gzip on this instance.
!gzip -d $zipped_file_name

user_metadata_df = pd.read_json (user_metadata_file_name)
user_metadata_df
```

```
gzip: ../../automation/ml_ops/domain/Retail/metadata/Users/users.json.gz: No such file or directory
```

Out[47]:

	id	gender	first_name	last_name	email	age	name	username	persona	
0	1	M	Mark	Johnson	mark.johnson@example.com	31	Mark Johnson	user1	furniture_homedecor_housewares	low
1	2	F	Kristen	Calderon	kristen.calderon@example.com	58	Kristen Calderon	user2	tools_housewares_apparel	
...	...	...	...	...	...	...	...	...	...	
5998	5999	F	Samantha	Reed	samantha.reed@example.com	34	Samantha Reed	user5999	electronics_outdoors_footwear	
5999	6000	F	Shannon	Martinez	shannon.martinez@example.com	24	Shannon Martinez	user6000	books_apparel_homedecor	

6000 rows × 13 columns

In [6]:

```
# zipping the metadata file
!gzip $user_metadata_file_name
```

Explore the Items Dataset

In [51]:

```
products_df = pd.read_csv(raw_items_filename)
pd.set_option('display.max_rows', 5)

products_df.head()
```

Out[51]:

	ITEM_ID	CATEGORY	STYLE
0	6579c22f-be2b-444c-a52b-0116dd82df6c	accessories	backpack
1	2e852905-c6f4-47db-802c-654013571922	accessories	backpack
2	4ec7ff5c-f70f-4984-b6c4-c7ef37cc0c09	accessories	backpack
3	7977f680-2cf7-457d-8f4d-afa0aa168cb9	accessories	backpack
4	b5649d7c-4651-458d-a07f-912f253784ce	accessories	backpack

We are going to load some metadata and create an auxiliary method to make the products more readable.

In [52]:

```
# load the item meta data
item_metadata_file_name = '../../automation/ml_ops/domain/Retail/metadata/Items/products.yaml'
with open(item_metadata_file_name) as f:
    item_metadata_df = pd.json_normalize(yaml.load(f, Loader=yaml.FullLoader))[['id', 'name', 'category', 'style', 'feature
d']]
```

In [53]:

```
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
```

Let's look at some examples

In [54]:

```
test = products_df.copy()

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

	ITEM_ID	CATEGORY	STYLE	ITEM_NAME
0	6579c22f-be2b-444c-a52b-0116dd82df6c	accessories	backpack	Tan Backpack
1	2e852905-c6f4-47db-802c-654013571922	accessories	backpack	Pale Pink Backpack
2	4ec7ff5c-f70f-4984-b6c4-c7ef37cc0c09	accessories	backpack	Gainsboro Backpack
3	7977f680-2cf7-457d-8f4d-afa0aa168cb9	accessories	backpack	Gray Backpack
4	b5649d7c-4651-458d-a07f-912f253784ce	accessories	backpack	Peru-Orange Backpack

Configure Amazon Personalize

Now that we've prepared our three datasets we'll need to configure the Amazon Personalize service to understand our data so that it can be used to train models for generating recommendations.

Create Schemas for Datasets

Amazon Personalize requires a schema for each dataset so it can map the columns in our CSVs to fields for model training. Each schema is declared in JSON using the Apache Avro (https://avro.apache.org/) format.

Let's define and create schemas in Personalize for our datasets.

Items Datsaset Schema

In [55]:

```
items_schema = {
    "type": "record",
    "name": "Items",
    "namespace": "com.amazonaws.personalize.schema",
    "fields": [
        {
            "name": "ITEM_ID",
            "type": "string"
        },
        {
            "name": "CATEGORY",
            "type": "string",
            "categorical": True,
        },
        {
            "name": "STYLE",
            "type": "string",
            "categorical": True,
        }
    ],
    "version": "1.0"
}

try:
    create_schema_response = personalize.create_schema(
        name = "personalize-immersion-day-retail-schema-items",
        schema = json.dumps(items_schema)
    )
    items_schema_arn = create_schema_response['schemaArn']
    print(json.dumps(create_schema_response, indent=2))

except personalize.exceptions.ResourceAlreadyExistsException:
    print('You aready created this schema, seemingly')
    schemas = personalize.list_schemas(maxResults=100)['schemas']
    for schema_response in schemas:
        if schema_response['name'] == "personalize-immersion-day-retail-schema-items":
            items_schema_arn = schema_response['schemaArn']
            print(f"Using existing schema: {items_schema_arn}")
```

```
You aready created this schema, seemingly
Using existing schema: arn:aws:personalize:us-east-1:968824662561:schema/personalize-immersion-day-retail-sche
ma-items
```

Users Dataset Schema

In [56]:

```
users_schema = {
    "type": "record",
    "name": "Users",
    "namespace": "com.amazonaws.personalize.schema",
    "fields": [
        {
            "name": "USER_ID",
            "type": "string"
        },
        {
            "name": "AGE",
            "type": "int"
        },
        {
            "name": "GENDER",
            "type": "string",
            "categorical": True,
        }
    ],
    "version": "1.0"
}

try:
    create_schema_response = personalize.create_schema(
        name = "rpersonalize-immersion-day-retail-schema-users",
        schema = json.dumps(users_schema)
    )
    print(json.dumps(create_schema_response, indent=2))
    users_schema_arn = create_schema_response['schemaArn']
except personalize.exceptions.ResourceAlreadyExistsException:
    print('You aready created this schema, seemingly')
    schemas = personalize.list_schemas(maxResults=100)['schemas']
    for schema_response in schemas:
        if schema_response['name'] == "personalize-immersion-day-retail-schema-users":
            users_schema_arn = schema_response['schemaArn']
            print(f"Using existing schema: {users_schema_arn}")
```

```
You aready created this schema, seemingly
```

Interactions Dataset Schema

In [57]:

```
interactions_schema = {
    "type": "record",
    "name": "Interactions",
    "namespace": "com.amazonaws.personalize.schema",
    "fields": [
        {
            "name": "ITEM_ID",
            "type": "string"
        },
        {
            "name": "USER_ID",
            "type": "string"
        },
        {
            "name": "EVENT_TYPE",  # "ProductViewed", "OrderCompleted", etc.
            "type": "string"
        },
        {
            "name": "TIMESTAMP",
            "type": "long"
        },
        {
            "name": "DISCOUNT",  # This is the contextual metadata - "Yes" or "No".
            "type": "string",
            "categorical": True,
        },
    ],
    "version": "1.0"
}

try:
    create_schema_response = personalize.create_schema(
        name = "personalize-immersion-day-retail-schema-interactions",
        schema = json.dumps(interactions_schema)
    )
    print(json.dumps(create_schema_response, indent=2))
    interactions_schema_arn = create_schema_response['schemaArn']
except personalize.exceptions.ResourceAlreadyExistsException:
    print('You aready created this schema, seemingly')
    schemas = personalize.list_schemas(maxResults=100)['schemas']
    for schema_response in schemas:
```

```
        if schema_response['name'] == "personalize-immersion-day-retail-schema-interactions":
            interactions_schema_arn = schema_response['schemaArn']
            print(f"Using existing schema: {interactions_schema_arn}")
```

```
You aready created this schema, seemingly
Using existing schema: arn:aws:personalize:us-east-1:968824662561:schema/personalize-immersion-day-retail-sche
ma-interactions
```

Create and Wait for Dataset Group

The highest level of isolation and abstraction with Amazon Personalize is a *dataset group*. Information stored within one of these dataset groups has no impact on any other dataset group or models created from one - they are completely isolated. This allows you to run many experiments and is part of how we keep your models private and fully trained only on your data.

Before importing the data prepared earlier, there needs to be a dataset group and a dataset added to it that handles the interactions.

Dataset groups can house the following types of information:

- User-item-interactions
- Event streams (real-time interactions)
- User metadata
- Item metadata

We need to create the dataset group that will contain our three datasets.

Create Dataset Group

The following cell will create a new dataset group with the name personalize-immersion-day-retail

In [58]:

```
create_dataset_group_response = personalize.create_dataset_group(
    name = 'personalize-immersion-day-retail'
)
dataset_group_arn = create_dataset_group_response['datasetGroupArn']
print(json.dumps(create_dataset_group_response, indent=2))

print(f'DatasetGroupArn = {dataset_group_arn}')
```

```
---------------------------------------------------------------------------
ResourceAlreadyExistsException                Traceback (most recent call last)
<ipython-input-58-4ed28c0d0c1a> in <module>
      1 create_dataset_group_response = personalize.create_dataset_group(
----> 2     name = 'personalize-immersion-day-retail'
      3 )
      4 dataset_group_arn = create_dataset_group_response['datasetGroupArn']
      5 print(json.dumps(create_dataset_group_response, indent=2))

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operation_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _make_api_call(self, operation_nam
e, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ResourceAlreadyExistsException: An error occurred (ResourceAlreadyExistsException) when calling the CreateData
setGroup operation: Another resource with Arn arn:aws:personalize:us-east-1:968824662561:dataset-group/persona
lize-immersion-day-retail already exists.
```

Wait for Dataset Group to Have ACTIVE Status

Before we can use the dataset group, it must be active. This can take a minute or two. Execute the cell below and wait for it to show the ACTIVE status. It checks the status of the dataset group every 15 seconds, up to a maximum of 3 hours.

In [59]:

```
status = None
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
```

Create Items Dataset

Next we will create the datasets in Personalize for our three dataset types. Let's start with the items dataset.

In [60]:

```
dataset_type = "ITEMS"
create_dataset_response = personalize.create_dataset(
    name = "personalize-immersion-day-dataset-items",
    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))
```

```
---------------------------------------------------------------------------
ResourceAlreadyExistsException                Traceback (most recent call last)
<ipython-input-60-85545f9133b0> in <module>
      4     datasetType = dataset_type,
      5     datasetGroupArn = dataset_group_arn,
----> 6     schemaArn = items_schema_arn
      7 )
      8

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operation_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _make_api_call(self, operation_nam
e, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ResourceAlreadyExistsException: An error occurred (ResourceAlreadyExistsException) when calling the CreateData
set operation: Another dataset of type: ITEMS already exists in the provided dataset group
```

Create Users Dataset

In [61]:

```
dataset_type = "USERS"
create_dataset_response = personalize.create_dataset(
    name = "personalize-immersion-day-dataset-users",
    datasetType = dataset_type,
    datasetGroupArn = dataset_group_arn,
    schemaArn = users_schema_arn
)

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

```
---------------------------------------------------------------------------
ResourceAlreadyExistsException                Traceback (most recent call last)
<ipython-input-61-32f1ea52172b> in <module>
      4     datasetType = dataset_type,
      5     datasetGroupArn = dataset_group_arn,
----> 6     schemaArn = users_schema_arn
      7 )
      8

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operation_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _make_api_call(self, operation_nam
e, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ResourceAlreadyExistsException: An error occurred (ResourceAlreadyExistsException) when calling the CreateData
set operation: Another dataset of type: USERS already exists in the provided dataset group
```

Create Interactions Dataset

In [62]:

```
dataset_type = "INTERACTIONS"
create_dataset_response = personalize.create_dataset(
    name = "personalize-immersion-day-dataset-interactions",
    datasetType = dataset_type,
    datasetGroupArn = dataset_group_arn,
    schemaArn = interactions_schema_arn
)

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

```
---------------------------------------------------------------------------
ResourceAlreadyExistsException            Traceback (most recent call last)
<ipython-input-62-58598916c7fc> in <module>
      4     datasetType = dataset_type,
      5     datasetGroupArn = dataset_group_arn,
----> 6     schemaArn = interactions_schema_arn
      7 )
      8

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operation_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _make_api_call(self, operation_nam
e, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ResourceAlreadyExistsException: An error occurred (ResourceAlreadyExistsException) when calling the CreateData
set operation: Another dataset of type: INTERACTIONS already exists in the provided dataset group
```

Import Datasets to Personalize

Up to this point we have generated CSVs containing data for our users, items, and interactions and staged them in an S3 bucket. We also created schemas in Personalize that define the columns in our CSVs. Then we created a datset group and three datasets in Personalize that will receive our data. In the following steps we will create import jobs with Personalize that will import the datasets from our S3 bucket into the service.

Setup Permissions

By default, the Personalize service does not have permission to acccess the data we uploaded into the S3 bucket in our account. In order to grant access to the Personalize service to read our CSVs, we need to set a Bucket Policy and create an IAM role that the Amazon Personalize service will assume.

Configure an S3 bucket and an IAM role

Back to top

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.

Use the metadata stored on the instance underlying this Amazon SageMaker notebook, to determine the region it is operating in. If you are using a Jupyter notebook outside of Amazon SageMaker, simply define the region as a string below. The Amazon S3 bucket needs to be in the same region as the Amazon Personalize resources we have been creating so far.

In [63]:

```
with open('/opt/ml/metadata/resource-metadata.json') as notebook_info:
    data = json.load(notebook_info)
    resource_arn = data['ResourceArn']
    region = resource_arn.split(':')[3]
    print(data)
print(region)
```

```
{'ResourceArn': 'arn:aws:sagemaker:us-east-1:968824662561:notebook-instance/amazonpersonalizeimmersionday', 'R
esourceName': 'AmazonPersonalizeImmersionDay'}
us-east-1
```

Amazon S3 bucket names are globally unique. To create a unique bucket name, the code below will append the string `personalizepocvod` to your AWS account number. Then it creates a bucket with this name in the region discovered in the previous cell.

In [64]:

```
s3 = boto3.client('s3')
account_id = boto3.client('sts').get_caller_identity().get('Account')
bucket_name = account_id + "-" + region + "-" + "personalize-immersionday-retail"
print(bucket_name)
if region == "us-east-1":
    s3.create_bucket(Bucket=bucket_name)
else:
    s3.create_bucket(
        Bucket=bucket_name,
        CreateBucketConfiguration={'LocationConstraint': region}
        )
```

```
968824662561-us-east-1-personalize-immersionday-retail
```

Upload data to S3

Now that your Amazon S3 bucket has been created, upload the CSV file of our user-item-interaction data.

In [65]:

```
boto3.Session().resource('s3').Bucket(bucket_name).Object(users_filename).upload_file(raw_users_filename)
boto3.Session().resource('s3').Bucket(bucket_name).Object(items_filename).upload_file(raw_items_filename)
boto3.Session().resource('s3').Bucket(bucket_name).Object(interactions_filename).upload_file(raw_interactions_filename)
```

Attach policy to S3 bucket

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.

In [66]:

```
s3 = boto3.client("s3")

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(bucket_name),
                "arn:aws:s3:::{}/*".format(bucket_name)
            ]
        }
    ]
}

s3.put_bucket_policy(Bucket=bucket_name, Policy=json.dumps(policy));
```

Create S3 Read Only Access Role

In [67]:

```
iam = boto3.client("iam")

role_name = account_id+"-PersonalizeS3"
assume_role_policy_document = {
    "Version": "2012-10-17",
    "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "personalize.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
    ]
}

try:
    create_role_response = iam.create_role(
        RoleName = role_name,
        AssumeRolePolicyDocument = json.dumps(assume_role_policy_document)
    );

except iam.exceptions.EntityAlreadyExistsException as e:
    print('Warning: role already exists:', e)
    create_role_response = iam.get_role(
        RoleName = role_name
    );

role_arn = create_role_response["Role"]["Arn"]

print('IAM Role: {}'.format(role_arn))

attach_response = iam.attach_role_policy(
    RoleName = role_name,
    PolicyArn = "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
);

role_arn = create_role_response["Role"]["Arn"]

# Pause to allow role to be fully consistent
```

```
time.sleep(30)
print('Done.')
```

```
Warning: role already exists: An error occurred (EntityAlreadyExists) when calling the CreateRole operation: R
ole with name 968824662561-PersonalizeS3 already exists.
IAM Role: arn:aws:iam::968824662561:role/968824662561-PersonalizeS3
Done.
```

Create Import Jobs

With the permissions in place to allow Personalize to access our CSV files, let's create three import jobs to import each file into its respective dataset. Each import job can take several minutes to complete so we'll create all three and then wait for them all to complete.

Create Items Dataset Import Job

In [68]:

```
items_create_dataset_import_job_response = personalize.create_dataset_import_job(
    jobName = "immersion-day-retail-dataset-items-import",
    datasetArn = items_dataset_arn,
    dataSource = {
        "dataLocation": "s3://{}/{}".format(bucket_name, items_filename)
    },
    roleArn = role_arn
)

items_dataset_import_job_arn = items_create_dataset_import_job_response['datasetImportJobArn']
print(json.dumps(items_create_dataset_import_job_response, indent=2))
```

```
---------------------------------------------------------------------------
ResourceAlreadyExistsException                Traceback (most recent call last)
<ipython-input-68-b2cc28e38b5e> in <module>
      5         "dataLocation": "s3://{}/{}".format(bucket_name, items_filename)
      6     },
----> 7     roleArn = role_arn
      8 )
      9

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operation_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _make_api_call(self, operation_nam
e, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ResourceAlreadyExistsException: An error occurred (ResourceAlreadyExistsException) when calling the CreateData
setImportJob operation: Another resource with Arn arn:aws:personalize:us-east-1:968824662561:dataset-import-jo
b/immersion-day-retail-dataset-items-import already exists.
```

Create Users Dataset Import Job

In [69]:

```
users_create_dataset_import_job_response = personalize.create_dataset_import_job(
    jobName = "immersion-day-retail-dataset-users-import",
    datasetArn = users_dataset_arn,
    dataSource = {
        "dataLocation": "s3://{}/{}".format(bucket_name, users_filename)
    },
    roleArn = role_arn
)

users_dataset_import_job_arn = users_create_dataset_import_job_response['datasetImportJobArn']
print(json.dumps(users_create_dataset_import_job_response, indent=2))
```

```
---------------------------------------------------------------------------
ResourceAlreadyExistsException                Traceback (most recent call last)
<ipython-input-69-557633ee87d0> in <module>
      5         "dataLocation": "s3://{}/{}".format(bucket_name, users_filename)
      6     },
----> 7     roleArn = role_arn
      8 )
      9

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operation_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _make_api_call(self, operation_nam
e, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ResourceAlreadyExistsException: An error occurred (ResourceAlreadyExistsException) when calling the CreateData
setImportJob operation: Another resource with Arn arn:aws:personalize:us-east-1:968824662561:dataset-import-jo
b/immersion-day-retail-dataset-users-import already exists.
```

Create Interactions Dataset Import Job

In [70]:

```
interactions_create_dataset_import_job_response = personalize.create_dataset_import_job(
    jobName = "immersion-day-retail-dataset-interactions-import",
    datasetArn = interactions_dataset_arn,
    dataSource = {
        "dataLocation": "s3://{}/{}".format(bucket_name, interactions_filename)
    },
    roleArn = role_arn
)

interactions_dataset_import_job_arn = interactions_create_dataset_import_job_response['datasetImportJobArn']
print(json.dumps(interactions_create_dataset_import_job_response, indent=2))
```

```
---------------------------------------------------------------------------
ResourceAlreadyExistsException                Traceback (most recent call last)
<ipython-input-70-4f1b316ff098> in <module>
      5         "dataLocation": "s3://{}/{}".format(bucket_name, interactions_filename)
      6     },
----> 7     roleArn = role_arn
      8 )
      9

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _api_call(self, *args, **kwargs)
    355                     "%s() only accepts keyword arguments." % py_operation_name)
    356             # The "self" in this scope is referring to the BaseClient.
--> 357             return self._make_api_call(operation_name, kwargs)
    358
    359         _api_call.__name__ = str(py_operation_name)

~/anaconda3/envs/python3/lib/python3.6/site-packages/botocore/client.py in _make_api_call(self, operation_nam
e, api_params)
    674             error_code = parsed_response.get("Error", {}).get("Code")
    675             error_class = self.exceptions.from_code(error_code)
--> 676             raise error_class(parsed_response, operation_name)
    677         else:
    678             return parsed_response

ResourceAlreadyExistsException: An error occurred (ResourceAlreadyExistsException) when calling the CreateData
setImportJob operation: Another resource with Arn arn:aws:personalize:us-east-1:968824662561:dataset-import-jo
b/immersion-day-retail-dataset-interactions-import already exists.
```

Wait for Import Jobs to Complete

It will take 10-15 minutes for the import jobs to complete, while you're waiting you can learn more about Datasets and Schemas in the documentation (https://docs.aws.amazon.com/personalize/latest/dg/how-it-works-dataset-schema.html).

We will wait for all three jobs to finish.

Wait for Items Import Job to Complete

In [71]:

```
%%time

import_job_arns = [ items_dataset_import_job_arn, users_dataset_import_job_arn, interactions_dataset_import_job_arn ]

max_time = time.time() + 3*60*60 # 3 hours
while time.time() < max_time:
    for job_arn in reversed(import_job_arns):
        import_job_response = personalize.describe_dataset_import_job(
            datasetImportJobArn = job_arn
        )
        status = import_job_response["datasetImportJob"]['status']

        if status == "ACTIVE":
            print(f'Import job {job_arn} successfully completed')
            import_job_arns.remove(job_arn)
        elif status == "CREATE FAILED":
            print(f'Import job {job_arn} failed')
            if import_job_response.get('failureReason'):
                print('   Reason: ' + import_job_response['failureReason'])
            import_job_arns.remove(job_arn)

    if len(import_job_arns) > 0:
        print('At least one dataset import job still in progress')
        time.sleep(60)
    else:
        print("All import jobs have ended")
        break
```

```
Import job arn:aws:personalize:us-east-1:968824662561:dataset-import-job/immersion-day-retail-dataset-interact
ions-import successfully completed
Import job arn:aws:personalize:us-east-1:968824662561:dataset-import-job/immersion-day-retail-dataset-users-im
port successfully completed
Import job arn:aws:personalize:us-east-1:968824662561:dataset-import-job/immersion-day-retail-dataset-items-im
port successfully completed
All import jobs have ended
CPU times: user 11.7 ms, sys: 0 ns, total: 11.7 ms
Wall time: 59.2 ms
```

When the dataset import is active, you are ready to start building models with SIMS, Personalized-Ranking, and User Personalization. This process will continue in other notebooks. Run the cell below before moving on to store a few values for usage in the next notebooks.

In [72]:

```
%store items_dataset_arn
%store interactions_dataset_arn
%store dataset_group_arn
%store bucket_name
%store role_arn
%store role_name
%store region
%store bucket_name
%store item_metadata_df
%store user_metadata_df
```

```
Stored 'items_dataset_arn' (str)
Stored 'interactions_dataset_arn' (str)
Stored 'dataset_group_arn' (str)
Stored 'bucket_name' (str)
Stored 'role_arn' (str)
Stored 'role_name' (str)
Stored 'region' (str)
Stored 'bucket_name' (str)
Stored 'item_metadata_df' (DataFrame)
Stored 'user_metadata_df' (DataFrame)
```

In [ ]:

```
```