Clean Up Resources

This notebook demonstrates how to clean up all the resources created in the previous set of notebooks.

This notebook uses the functions defined below, to iterate throught the resources inside a dataset group. The dataset group arn is saved from the Notebook `01_Data_Layer.pnyb`

In [5]:

```
%store -r
```

In [6]:

```
import sys
import getopt
import logging
import botocore
import boto3
import time
from packaging import version
from time import sleep
from botocore.exceptions import ClientError

logger = logging.getLogger()
personalize = None

def _get_dataset_group_arn(dataset_group_name):
    dsg_arn = None

    paginator = personalize.get_paginator('list_dataset_groups')
    for paginate_result in paginator.paginate():
        for dataset_group in paginate_result["datasetGroups"]:
            if dataset_group['name'] == dataset_group_name:
                dsg_arn = dataset_group['datasetGroupArn']
                break

        if dsg_arn:
            break

    if not dsg_arn:
        raise NameError(f'Dataset Group "{dataset_group_name}" does not exist; verify region is correct')

    return dsg_arn

def _get_solutions(dataset_group_arn):
    solution_arns = []

    paginator = personalize.get_paginator('list_solutions')
    for paginate_result in paginator.paginate(datasetGroupArn = dataset_group_arn):
        for solution in paginate_result['solutions']:
            solution_arns.append(solution['solutionArn'])

    return solution_arns
```

```
def _delete_campaigns(solution_arns):
    campaign_arns = []

    for solution_arn in solution_arns:
        paginator = personalize.get_paginator('list_campaigns')
        for paginate_result in paginator.paginate(solutionArn = solution_arn):
            for campaign in paginate_result['campaigns']:
                if campaign['status'] in ['ACTIVE', 'CREATE FAILED']:
                    logger.info('Deleting campaign: ' + campaign['campaignArn'])

                    personalize.delete_campaign(campaignArn = campaign['campaignArn'])
                elif campaign['status'].startswith('DELETE'):
                    logger.warning('Campaign {} is already being deleted so will wait for delete to complete'.format(campai
gn['campaignArn']))
                else:
                    raise Exception('Campaign {} has a status of {} so cannot be deleted'.format(campaign['campaignArn'], c
ampaign['status']))

                campaign_arns.append(campaign['campaignArn'])

    max_time = time.time() + 30*60 # 30 mins
    while time.time() < max_time:
        for campaign_arn in campaign_arns:
            try:
                describe_response = personalize.describe_campaign(campaignArn = campaign_arn)
                logger.debug('Campaign {} status is {}'.format(campaign_arn, describe_response['campaign']['status']))
            except ClientError as e:
                error_code = e.response['Error']['Code']
                if error_code == 'ResourceNotFoundException':
                    campaign_arns.remove(campaign_arn)

        if len(campaign_arns) == 0:
            logger.info('All campaigns have been deleted or none exist for dataset group')
            break
        else:
            logger.info('Waiting for {} campaign(s) to be deleted'.format(len(campaign_arns)))
            time.sleep(20)

    if len(campaign_arns) > 0:
        raise Exception('Timed out waiting for all campaigns to be deleted')

def _delete_solutions(solution_arns):
    for solution_arn in solution_arns:
```

```
        try:
            describe_response = personalize.describe_solution(solutionArn = solution_arn)
            solution = describe_response['solution']
            if solution['status'] in ['ACTIVE', 'CREATE FAILED']:
                logger.info('Deleting solution: ' + solution_arn)

                personalize.delete_solution(solutionArn = solution_arn)
            elif solution['status'].startswith('DELETE'):
                logger.warning('Solution {} is already being deleted so will wait for delete to complete'.format(solution_a
rn))
            else:
                raise Exception('Solution {} has a status of {} so cannot be deleted'.format(solution_arn, solution['statu
s']))
        except ClientError as e:
            error_code = e.response['Error']['Code']
            if error_code != 'ResourceNotFoundException':
                raise e

    max_time = time.time() + 30*60 # 30 mins
    while time.time() < max_time:
        for solution_arn in solution_arns:
            try:
                describe_response = personalize.describe_solution(solutionArn = solution_arn)
                logger.debug('Solution {} status is {}'.format(solution_arn, describe_response['solution']['status']))
            except ClientError as e:
                error_code = e.response['Error']['Code']
                if error_code == 'ResourceNotFoundException':
                    solution_arns.remove(solution_arn)

        if len(solution_arns) == 0:
            logger.info('All solutions have been deleted or none exist for dataset group')
            break
        else:
            logger.info('Waiting for {} solution(s) to be deleted'.format(len(solution_arns)))
            time.sleep(20)

    if len(solution_arns) > 0:
        raise Exception('Timed out waiting for all solutions to be deleted')

def _delete_event_trackers(dataset_group_arn):
    event_tracker_arns = []

    event_trackers_paginator = personalize.get_paginator('list_event_trackers')
```

```
    for event_tracker_page in event_trackers_paginator.paginate(datasetGroupArn = dataset_group_arn):
        for event_tracker in event_tracker_page['eventTrackers']:
            if event_tracker['status'] in [ 'ACTIVE', 'CREATE FAILED' ]:
                logger.info('Deleting event tracker {}'.format(event_tracker['eventTrackerArn']))
                personalize.delete_event_tracker(eventTrackerArn = event_tracker['eventTrackerArn'])
            elif event_tracker['status'].startswith('DELETE'):
                logger.warning('Event tracker {} is already being deleted so will wait for delete to complete'.format(event
_tracker['eventTrackerArn']))
            else:
                raise Exception('Solution {} has a status of {} so cannot be deleted'.format(event_tracker['eventTrackerAr
n'], event_tracker['status']))

            event_tracker_arns.append(event_tracker['eventTrackerArn'])

    max_time = time.time() + 30*60 # 30 mins
    while time.time() < max_time:
        for event_tracker_arn in event_tracker_arns:
            try:
                describe_response = personalize.describe_event_tracker(eventTrackerArn = event_tracker_arn)
                logger.debug('Event tracker {} status is {}'.format(event_tracker_arn, describe_response['eventTracker']['s
tatus']))
            except ClientError as e:
                error_code = e.response['Error']['Code']
                if error_code == 'ResourceNotFoundException':
                    event_tracker_arns.remove(event_tracker_arn)

        if len(event_tracker_arns) == 0:
            logger.info('All event trackers have been deleted or none exist for dataset group')
            break
        else:
            logger.info('Waiting for {} event tracker(s) to be deleted'.format(len(event_tracker_arns)))
            time.sleep(20)

    if len(event_tracker_arns) > 0:
        raise Exception('Timed out waiting for all event trackers to be deleted')

def _delete_filters(dataset_group_arn):
    filter_arns = []

    filters_response = personalize.list_filters(datasetGroupArn = dataset_group_arn, maxResults = 100)
    for filter in filters_response['Filters']:
        logger.info('Deleting filter ' + filter['filterArn'])
        personalize.delete_filter(filterArn = filter['filterArn'])
```

```
        filter_arns.append(filter['filterArn'])

    max_time = time.time() + 30*60 # 30 mins
    while time.time() < max_time:
        for filter_arn in filter_arns:
            try:
                describe_response = personalize.describe_filter(filterArn = filter_arn)
                logger.debug('Filter {} status is {}'.format(filter_arn, describe_response['filter']['status']))
            except ClientError as e:
                error_code = e.response['Error']['Code']
                if error_code == 'ResourceNotFoundException':
                    filter_arns.remove(filter_arn)

        if len(filter_arns) == 0:
            logger.info('All filters have been deleted or none exist for dataset group')
            break
        else:
            logger.info('Waiting for {} filter(s) to be deleted'.format(len(filter_arns)))
            time.sleep(20)

    if len(filter_arns) > 0:
        raise Exception('Timed out waiting for all filter to be deleted')

def _delete_datasets_and_schemas(dataset_group_arn):
    dataset_arns = []
    schema_arns = []

    dataset_paginator = personalize.get_paginator('list_datasets')
    for dataset_page in dataset_paginator.paginate(datasetGroupArn = dataset_group_arn):
        for dataset in dataset_page['datasets']:
            describe_response = personalize.describe_dataset(datasetArn = dataset['datasetArn'])
            schema_arns.append(describe_response['dataset']['schemaArn'])

            if dataset['status'] in ['ACTIVE', 'CREATE FAILED']:
                logger.info('Deleting dataset ' + dataset['datasetArn'])
                personalize.delete_dataset(datasetArn = dataset['datasetArn'])
            elif dataset['status'].startswith('DELETE'):
                logger.warning('Dataset {} is already being deleted so will wait for delete to complete'.format(dataset['da
tasetArn']))
            else:
                raise Exception('Dataset {} has a status of {} so cannot be deleted'.format(dataset['datasetArn'], dataset[
'status']))
```

```
            dataset_arns.append(dataset['datasetArn'])

    max_time = time.time() + 30*60 # 30 mins
    while time.time() < max_time:
        for dataset_arn in dataset_arns:
            try:
                describe_response = personalize.describe_dataset(datasetArn = dataset_arn)
                logger.debug('Dataset {} status is {}'.format(dataset_arn, describe_response['dataset']['status']))
            except ClientError as e:
                error_code = e.response['Error']['Code']
                if error_code == 'ResourceNotFoundException':
                    dataset_arns.remove(dataset_arn)

        if len(dataset_arns) == 0:
            logger.info('All datasets have been deleted or none exist for dataset group')
            break
        else:
            logger.info('Waiting for {} dataset(s) to be deleted'.format(len(dataset_arns)))
            time.sleep(20)

    if len(dataset_arns) > 0:
        raise Exception('Timed out waiting for all datasets to be deleted')

    for schema_arn in schema_arns:
        try:
            logger.info('Deleting schema ' + schema_arn)
            personalize.delete_schema(schemaArn = schema_arn)
        except ClientError as e:
            error_code = e.response['Error']['Code']
            if error_code == 'ResourceInUseException':
                logger.info('Schema {} is still in-use by another dataset (likely in another dataset group)'.format(schema_
arn))
            else:
                raise e

    logger.info('All schemas used exclusively by datasets have been deleted or none exist for dataset group')

def _delete_dataset_group(dataset_group_arn):
    logger.info('Deleting dataset group ' + dataset_group_arn)
    personalize.delete_dataset_group(datasetGroupArn = dataset_group_arn)

    max_time = time.time() + 30*60 # 30 mins
    while time.time() < max_time:
```

```
        try:
            describe_response = personalize.describe_dataset_group(datasetGroupArn = dataset_group_arn)
            logger.debug('Dataset group {} status is {}'.format(dataset_group_arn, describe_response['datasetGroup']['statu
s']))
            break
        except ClientError as e:
            error_code = e.response['Error']['Code']
            if error_code == 'ResourceNotFoundException':
                logger.info('Dataset group {} has been fully deleted'.format(dataset_group_arn))
            else:
                raise e

        logger.info('Waiting for dataset group to be deleted')
        time.sleep(20)

def delete_dataset_groups(dataset_group_arns, region = None):
    global personalize
    personalize = boto3.client(service_name = 'personalize', region_name = region)

    for dataset_group_arn in dataset_group_arns:
        logger.info('Dataset Group ARN: ' + dataset_group_arn)

        solution_arns = _get_solutions(dataset_group_arn)

        # 1. Delete campaigns
        _delete_campaigns(solution_arns)

        # 2. Delete solutions
        _delete_solutions(solution_arns)

        # 3. Delete event trackers
        _delete_event_trackers(dataset_group_arn)

        # 4. Delete filters
        _delete_filters(dataset_group_arn)

        # 5. Delete datasets and their schemas
        _delete_datasets_and_schemas(dataset_group_arn)

        # 6. Delete dataset group
        _delete_dataset_group(dataset_group_arn)

        logger.info(f'Dataset group {dataset_group_arn} fully deleted')
```

In [7]:

```
delete_dataset_groups([dataset_group_arn], region)
```

Clean up the S3 bucket and IAM role

Start by deleting the role, then empty the bucket, then delete the bucket.

In [8]:

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

Identify the name of the role you want to delete.

You cannot delete an IAM role which still has policies attached to it. So after you have identified the relevant role, let's list the attached policies of that role.

In [9]:

```
iam.list_attached_role_policies(
    RoleName = role_name
)
```

Out[9]:

```
{'AttachedPolicies': [{'PolicyName': 'AmazonS3ReadOnlyAccess',
   'PolicyArn': 'arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess'}],
 'IsTruncated': False,
 'ResponseMetadata': {'RequestId': 'ced84c50-0625-4c03-ad11-38fa97de2525',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'ced84c50-0625-4c03-ad11-38fa97de2525',
   'content-type': 'text/xml',
   'content-length': '548',
   'date': 'Sun, 18 Apr 2021 05:05:09 GMT'},
  'RetryAttempts': 0}}
```

You need to detach the policies in the result above using the code below. Repeat for each attached policy.

In [10]:

```
iam.detach_role_policy(
    RoleName = role_name,
    PolicyArn = 'arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess'
)
```

Out[10]:

```
{'ResponseMetadata': {'RequestId': '97580ec9-ba78-4227-b23e-232cd4c74a82',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '97580ec9-ba78-4227-b23e-232cd4c74a82',
   'content-type': 'text/xml',
   'content-length': '212',
   'date': 'Sun, 18 Apr 2021 05:05:10 GMT'},
  'RetryAttempts': 0}}
```

Finally, you should be able to delete the IAM role.

In [11]:

```
iam.delete_role(
    RoleName = role_name
)
```

Out[11]:

```
{'ResponseMetadata': {'RequestId': '4e0c5b38-c9ac-40a5-870b-eebef6099a83',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '4e0c5b38-c9ac-40a5-870b-eebef6099a83',
   'content-type': 'text/xml',
   'content-length': '200',
   'date': 'Sun, 18 Apr 2021 05:05:11 GMT'},
  'RetryAttempts': 0}}
```

To delete an S3 bucket, it first needs to be empty. The easiest way to delete an S3 bucket, is just to navigate to S3 in the AWS console, delete the objects in the bucket, and then delete the S3 bucket itself.

Deleting the Automation from the Operations Notebook

In [12]:

```
stack_name = "notebook-automation"
bucket= !aws cloudformation describe-stacks --stack-name $stack_name --query "Stacks[0].Outputs[?OutputKey=='InputBucketNam
e'].OutputValue" --output text
bucket_name = bucket[0]
print(bucket)
```

```
['notebook-automation-inputbucket-1g8dq180xs23x']
```

In [13]:

```
!aws s3 rb s3://$bucket_name --force
```

```
delete: s3://notebook-automation-inputbucket-1g8dq180xs23x/Items/items.csv
delete: s3://notebook-automation-inputbucket-1g8dq180xs23x/Interactions/interactions.csv
delete: s3://notebook-automation-inputbucket-1g8dq180xs23x/Users/users.csv
delete: s3://notebook-automation-inputbucket-1g8dq180xs23x/params.json
remove_bucket: notebook-automation-inputbucket-1g8dq180xs23x
```

In [14]:

```
!aws cloudformation delete-stack --stack-name $stack_name
time.sleep(120)
```

In [15]:

```
!aws cloudformation describe-stacks --stack-name $stack_name
```

```
An error occurred (ValidationError) when calling the DescribeStacks operation: Stack with id notebook-automati
on does not exist
```

Deleting the Automation from the Initial CloudFormation deployment

In [16]:

```
stack_name = "id-ml-ops"
bucket= !aws cloudformation describe-stacks --stack-name $stack_name --query "Stacks[0].Outputs[?OutputKey=='InputBucketNam
e'].OutputValue" --output text
bucket_name = bucket[0]
print(bucket_name)
```

```
id-ml-ops-inputbucket-xzk57si1iact
```

In [17]:

```
!aws s3 rb s3://$bucket_name --force
```

```
delete: s3://id-ml-ops-inputbucket-xzk57si1iact/params.json
delete: s3://id-ml-ops-inputbucket-xzk57si1iact/Interactions/interactions.csv
delete: s3://id-ml-ops-inputbucket-xzk57si1iact/Items/items.csv
remove_bucket: id-ml-ops-inputbucket-xzk57si1iact
```

In [18]:

```
!aws cloudformation delete-stack --stack-name $stack_name
time.sleep(120)
```

In [19]:

```
!aws cloudformation describe-stacks --stack-name $stack_name
```

```
An error occurred (ValidationError) when calling the DescribeStacks operation: Stack with id id-ml-ops does no
t exist
```

Deleting the bucket with the automation artifacts

In [20]:

```
stack_name = "AmazonPersonalizeImmersionDay"
bucket = !aws cloudformation describe-stack-resources --stack-name $stack_name --logical-resource-id SAMArtifactsBucket --q
uery "StackResources[0].PhysicalResourceId" --output text
bucket_name = bucket[0]
print(bucket_name)
```

```
amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p
```

In [21]:

```
!aws s3 rb s3://$bucket_name --force
```

```
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/0f0ec7c3abc208c2e199dce9c8ce4b51
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/2ca191e489fec709d8b11ad7355a370d
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/4c6edf07c8b83364b0dc8689909a8298
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/5dbc739ed2b0126ea25e5230f6521101
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/13f9869ea945e9895628bee171bc81d3
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/1c5871813c8629b4f064fda391486607
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/64a957bf7261071c26ad6cbb03fc3a16
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/4ef1473d88b7d622abfec2ba4ce99163
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/4409fcd8be1849a55297297e1c1c0a73
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/75be8b98a297077f44ee0ffbc8769d48
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/578972edf92ae01ed286b828d1e966d3
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/3dd4f0c0888dc8d0b3e5e9c320a75b69
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/7e570c56cf6dcdefd616657a22f72e1f
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/91a4c1cba5fba2b8c3d9337465bdda74
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/3c8afa7870c7a77297da60c33e11be3c
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/a9be891522d6fd4a5e9e4d44bf8cf654
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/63fbbf55f913adc6d4b55208538d6fd7
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/e7acb3535d07c48642f8bc573bd108fb
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/abc16ccc7c6fcc1efbc822cf1762649c
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/ac1a7db8883840540edd53457e85a977
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/c29d1652ff78d3b8f31e74e389bdde2c
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/f5903135967778e822b06c029778d9a3
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/d041117c2a834b1794a9685c9b7fdacd.t
emplate
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/ea74c09d688c735af9523d1d5f26b2d4
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/e00df2b3ea6d3ccdf86b84bb92f67c0c
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/8cec483a1255f0538f114b53d9c59621
delete: s3://amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p/c81096660862fbfd6d27ae42bef7ad61
remove_bucket: amazonpersonalizeimmersionday-samartifactsbucket-1izoghbemfa7p
```

Now you can navigate to your Cloudformation console (https://console.aws.amazon.com/cloudformation/) and delete the **AmazonPersonalizeImmersionDay** stack