Sagemaker_Code

IMPORT LIBRARIES

```
import sagemaker
import boto3
import pandas as pd
from sagemaker import get_execution_role
from time import gmtime, strftime, sleep
from datautils.snowflake.SnowflakeJDBC import SnowflakeJDBC
from datetime import datetime,date
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder
```

LOAD DATA

```
def read_files(config_path):
    '''
    Read the data files and config files that are required for generating the insights.
    Input parameters:
        - config_path : Path that contains config file
    Returns
        - span     : Time span of the data
    '''
    with open(config_path, "r") as stream:
        file_content = yaml.full_load(stream)
    req_dict = {}
    for each in file_content:
        req_dict.update(each)
    span = int(req_dict['TIME_SPAN'][0])
    return span

 def load_data(con,query):
    '''
    Function to load the data from Snowflake
    Input paramters:
        - con   : Connection string to Snowflake
        - query : Query to the Snowflake
    Returns:
        - df : Pandas DataFrame returning the loaded data
    '''
    cursor = con.cursor()
    cursor.execute(query)
    df = cursor.fetch_pandas_all()
    #logging.info('Loading Data module starts!')
    df = df.sort_values('SNAP_MONTH').reset_index(drop = True)
    return df
```

```
def load_run():
    config_file = './config.yaml'
    con = SnowflakeJDBC(use_vault_for_creds=False
                        ,role = 'DM_GDSO_READ_INTERNAL_PRD'
                        ,user = 'ggupta1@SALESFORCE.COM'
                        ,authenticator='externalbrowser'
                        ,warehouse = 'WH_GDSO_POWER_USERS'
                        ,schema = 'GDSO_ORDER'
                        ,database = 'SSE_DM_GDSO_PRD'
                        ,account = 'sfdc-dp-prd'
                       )
    query = """ select * from sse_dm_gdso_prd.gdso_order.FINAL_MONTH_QUOTAATT_TEST_3M
    df = load_data(con,query)
    df = df.sort_values('SNAP_MONTH').reset_index(drop = True)
    df['SNAP_MONTH'] = pd.to_datetime(df['SNAP_MONTH'])
    remove_feats = ['MAX_VERT_IND','TENURE_ROLE','PCC_COUNTRY','PCC_COUNTRY_CLUSTER',
                    'EMP_EMAIL_ADDR','EMP_ID','FULL_NM','FISCAL_YR_NM','FIN_FISCAL_QT
                    'VAL_QUOTA','REM_QUOTA_ATT_NUM','REM_QUOTA_ATT','QUOTA_ACHIEVED_P
                    'QUOTA_ATT_NUM','MAPPEDSUBREGION','MAPPEDMARKETSEGMENT','MAPPEDSE
    df = df.drop(columns = remove_feats)
    mpa_dict = {'FQ 2':2,'FQ 1':1,'FQ 3':3,'FQ 4':4}
    df['FISCAL_QTR_NM'] = df['FISCAL_QTR_NM'].map(mpa_dict)
    df.to_csv('./artifacts/Raw_Data_new_spp.csv',index = False)
    return df

df = load_run()
```

SPLIT DATA

```
region = boto3.Session().region_name
bucket = "uip-datalake-bucket-prod"
session = sagemaker.Session(default_bucket=bucket)
prefix = "gdso/datascience/users/ggupta1"
tagValue = [{"Key": "app_group", "Value": "uip_gdso_ds"}]
role = get_execution_role()
sm = boto3.client(service_name="sagemaker", region_name=region)
train_data = df[(df['SNAP_MONTH'] >= '2020-05-01') & (df['SNAP_MONTH'] <= '2021-04-01

test_data = df[(df['SNAP_MONTH'] >= '2021-05-01') & (df['SNAP_MONTH'] <= '2021-10-01
holdout_data = df[(df['SNAP_MONTH'] >= '2021-11-01')]
train_data,test_data,holdout_data = train_data.drop(columns = ['SNAP_MONTH']),test_da
test_data_no_target = test_data.drop(columns=["QUOTA_ATT"])
# Upload to S3
train_file = "train_data.csv"

df_majority = train_data[train_data.QUOTA_ATT == 0]
df_minority = train_data[train_data.QUOTA_ATT == 1]
df_majority_downsampled = df_majority.sample(n=len(df_minority), random_state=42)
train_data = pd.concat([df_majority_downsampled, df_minority])
print(train_data.QUOTA_ATT.value_counts())

train_data.to_csv(train_file, index=False, header=True)
train_data_s3_path = session.upload_data(path=train_file, key_prefix=prefix + "/train"
```

```
test_file = "test_data.csv"
test_data_no_target.to_csv(test_file, index=False, header=False)
test_data_s3_path = session.upload_data(path=test_file, key_prefix=prefix + "/test")
holdout_file = "holdout_data.csv"
holdout_data.to_csv(holdout_file, index=False, header=False)
holdout_data_s3_path = session.upload_data(path=holdout_file, key_prefix=prefix + "/ho
```

SAGEMAKER SETUP

```
auto_ml_job_config = {"CompletionCriteria": {"MaxCandidates": 3}}
input_data_config = [
    {
        "DataSource": {
            "S3DataSource": {
                "S3DataType": "S3Prefix",
                "S3Uri": f"s3://{bucket}/{prefix}/train"
            }
        },
        "TargetAttributeName": "QUOTA_ATT",
    }
]
output_data_config = {"S3OutputPath": f"s3://{bucket}/{prefix}/output"}
timestamp_suffix = strftime("%d-%H-%M-%S", gmtime())
auto_ml_job_name = f"automl-seed-{timestamp_suffix}"
# Create AutoML job
sm.create_auto_ml_job(
    AutoMLJobName=auto_ml_job_name,
    InputDataConfig=input_data_config,
    OutputDataConfig=output_data_config,
    AutoMLJobConfig=auto_ml_job_config,
    RoleArn=role,
    Tags=tagValue
)
# Monitor job
while True:
    describe_response = sm.describe_auto_ml_job(AutoMLJobName=auto_ml_job_name)
    job_status = describe_response["AutoMLJobStatus"]
    if job_status in ("Failed", "Completed", "Stopped"):
        break
    sleep(30)
# Get and print best candidate
best_candidate = describe_response["BestCandidate"]
print("Best Candidate:", best_candidate)
# Create model
model_name = f"automl-seed-model-{timestamp_suffix}"
model = sm.create_model(
    Containers=best_candidate["InferenceContainers"],
    ModelName=model_name,
    ExecutionRoleArn=role,
    Tags=tagValue
)
# Create transform job
transform_job_name = f"automl-seed-transform-{timestamp_suffix}"
transform_input = {
```

```
    "DataSource": {"S3DataSource": {"S3DataType": "S3Prefix", "S3Uri": test_data_s3_pa
    "ContentType": "text/csv",
    "CompressionType": "None",
    "SplitType": "Line"
}
transform_output = {"S3OutputPath": f"s3://{bucket}/{prefix}/inference-results"}
transform_resources = {"InstanceType": "ml.m5.4xlarge", "InstanceCount": 1}
sm.create_transform_job(
    TransformJobName=transform_job_name,
    ModelName=model_name,
    TransformInput=transform_input,
    TransformOutput=transform_output,
    TransformResources=transform_resources,
    Tags=tagValue
)
# Monitor transform job
while True:
    describe_response = sm.describe_transform_job(TransformJobName=transform_job_name
    job_status = describe_response["TransformJobStatus"]
    if job_status in ("Failed", "Completed", "Stopped"):
        break
    sleep(30)
# Download and print results
s3_output_key = f"{prefix}/inference-results/test_data.csv.out"
local_inference_results_path = "inference_results.csv"
s3 = boto3.resource("s3")
inference_results_bucket = s3.Bucket(session.default_bucket())
inference_results_bucket.download_file(s3_output_key, local_inference_results_path)
results = pd.read_csv(local_inference_results_path, sep=";")
print(results)
# List all candidates
candidates = sm.list_candidates_for_auto_ml_job(AutoMLJobName=auto_ml_job_name,
                                                SortBy="FinalObjectiveMetricValue")["C
for idx, candidate in enumerate(candidates, 1):
    print(f"{idx}  {candidate['CandidateName']}  {candidate['FinalAutoMLJobObjectiveMe

model = next(iter(best_candidate.items()))
print("Best Model:", model)
```

SAGEMAKER MODEL DEPLOYMENT

```
import boto3
import sagemaker
import time
from sklearn.metrics import roc_auc_score

# Initial Setup
region = boto3.Session().region_name
bucket = "uip-datalake-bucket-prod"
session = sagemaker.Session(default_bucket=bucket)
sm = boto3.client(service_name="sagemaker", region_name=region)
client = boto3.client('sagemaker')
```

```
# AutoML Job Details
automl_job_name = auto_ml_job_name # Replace with your AutoML job name
best_candidate = client.describe_auto_ml_job(AutoMLJobName=automl_job_name)['BestCandi

# Model Creation
model_name = 'seed-model-test-{}'.format(int(time.time()))
role = role # Replace with your IAM role ARN
response = sm.create_model(
ModelName=model_name,
Containers=best_candidate['InferenceContainers'],
ExecutionRoleArn=role,
Tags=[{"Key": "app_group", "Value": "uip_gdso_ds"}]
)

# Endpoint Configuration & Deployment
endpoint_config_name = 'seed-endpoint-config-test-ggupta1-{}'.format(int(time.time()))
response = sm.create_endpoint_config(
EndpointConfigName=endpoint_config_name,
ProductionVariants=[
{
'VariantName': 'AllTraffic',
'ModelName': model_name,
'InitialInstanceCount': 1,
'InstanceType': 'ml.m5.4xlarge',
'InitialVariantWeight': 1
},
],
Tags=[{"Key": "app_group", "Value": "uip_gdso_ds"}]
)
endpoint_name = 'seed-endpoint-test-ggupta-{}'.format(int(time.time()))
response = sm.create_endpoint(
EndpointName=endpoint_name,
EndpointConfigName=endpoint_config_name,
Tags=[{"Key": "app_group", "Value": "uip_gdso_ds"}]
)

# Wait for the endpoint to be in 'InService' state (You can also add more error handli
time.sleep(10) # A short delay to give AWS time to start deploying.
while sm.describe_endpoint(EndpointName=endpoint_name)['EndpointStatus'] == 'Creating':
time.sleep(60)
```

END TO END CODE

```
# Import Library

import sagemaker
import boto3
import pandas as pd
from sagemaker import get_execution_role
from time import gmtime, strftime, sleep
from datautils.snowflake.SnowflakeJDBC import SnowflakeJDBC
from datetime import datetime,date
```

```
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import LabelEncoder

# Define Functions

def read_files(config_path):
    '''
    Read the data files and config files that are required for generating the insights.
    Input parameters:
        - config_path : Path that contains config file
    Returns
        - span     : Time span of the data
    '''
    with open(config_path, "r") as stream:
        file_content = yaml.full_load(stream)
    req_dict = {}
    for each in file_content:
        req_dict.update(each)
    span = int(req_dict['TIME_SPAN'][0])
    return span


def load_data(con,query):
    '''
    Function to load the data from Snowflake
    Input paramters:
        - con   : Connection string to Snowflake
        - query : Query to the Snowflake
    Returns:
        - df : Pandas DataFrame returning the loaded data
    '''
    cursor = con.cursor()
    cursor.execute(query)
    df = cursor.fetch_pandas_all()
    #logging.info('Loading Data module starts!')
    df = df.sort_values('SNAP_MONTH').reset_index(drop = True)
    return df


def load_run():
    config_file = './config.yaml'
    con = SnowflakeJDBC(use_vault_for_creds=False
                    ,role = 'DM_GDSO_READ_INTERNAL_PRD'
                    ,user = 'ggupta1@SALESFORCE.COM'
                    ,authenticator='externalbrowser'
                    ,warehouse = 'WH_GDSO_POWER_USERS'
                    ,schema = 'GDSO_ORDER'
                    ,database = 'SSE_DM_GDSO_PRD'
                    ,account = 'sfdc-dp-prd'
                   )
    #query = """ select * from sse_dm_gdso_prd.gdso_order.FINAL_MONTH_QUOTAATT_TEST_3M
    query = """ select * fromSSE_DM_GDSO_PRD.SELLER_INSIGHTS.SPP_FINAL_DATASET """


    df = load_data(con,query)
```

```
    df = df.sort_values('SNAP_MONTH').reset_index(drop = True)

    df=df[['SNAP_MONTH','HIST_3MON_AVG_DEALSIZE','HIST_3MON_AVG_DEALSIZE_RATIO','HIS
       'HIST_3MON_IN_QTR_ACV_PERC','HIST_3MON_CLOSE_RATE','HIST_3MON_NEWBUSINESS_PI
       'HIST_3MON_RUNRATE_PIPEGEN','HIST_3MON_KEYDEALS_PIPEGEN','HIST_3MON_LARGEDEALS
       'HIST_3MON_PG_QUOTA_RATIO','HIST_3MON_SALESDEV_PIPEGEN','CURR_Q_OP_STG2TO3','
       'PERC_OP_DEALS_NEXT_STEPS','CURR_Q_RED_ACTVTY_OP','CURRQ_OP_QUOTA_RATIO','PERC
       'PERC_OP_DEALS_PLAY_ATTCHD','MAX_VERT_PERC','HIST_3MON_TOT_MEETINGS','HIST_3MO
       'HIST_3MON_SE_HOURS_PER_ACCT','PERC_IN_OP','PERC_COMMIT_OP','CURR_Q_MANU_OP',
       ,'CURR_Q_PROF_SERVICES_OP','HIST_3MON_MLTI_CLD_CLSD_PERC','CURR_Q_HIGH_TECH_O
       'CURR_Q_SERVICE_OP','CURR_Q_INTEGRATION_OP','HIST_3MON_MLTI_CLD_ACV_PERC','CUR
       'CURR_Q_TRAVEL_OP','CURR_Q_ENG_REALESTATE_OP','HIST_3MON_TOT_CNVRTD_LDS','HIST
       'HIST_3MON_DIRCTR_BELOW_LVL_CONNECTS','HIST_3MON_DIRCTR_PLS_LVL_MEETINGS','HI
       'HIST_3MON_DIRCTR_BELOW_OPTY_CONNECTS','HIST_3MON_DIRCTR_PLS_OPTY_MEETINGS','
       'TOT_ACCOUNTS','HIST_3MON_TOT_DSR','HIST_3MON_DSR_PER_ACCT','HIST_3MON_CC_ACV
       'AE_SFDC_TENURE','TENURE_ROLE','TENURE_IN_ROLE','HIST_3MON_CLOSE_RATE_COUNT','
       'HIST_3MON_QUOTA_ATT','QUOTA_ACHIEVED_PERC','QUOTA_ATT']]
    df['SNAP_MONTH'] = pd.to_datetime(df['SNAP_MONTH'])
#     remove_feats = ['MAX_VERT_IND','TENURE_ROLE','PCC_COUNTRY','PCC_COUNTRY_CLUSTER'
#                     'EMP_EMAIL_ADDR','EMP_ID','FULL_NM','FISCAL_YR_NM','FIN_FISCAL_Q
#                     'VAL_QUOTA','REM_QUOTA_ATT_NUM','REM_QUOTA_ATT','QUOTA_ACHIEVED_
#                     'QUOTA_ATT_NUM','MAPPEDSUBREGION','MAPPEDMARKETSEGMENT','MAPPEDS


#     df = df.drop(columns = remove_feats)
#     mpa_dict = {'FQ 2':2,'FQ 1':1,'FQ 3':3,'FQ 4':4}
#     df['FISCAL_QTR_NM'] = df['FISCAL_QTR_NM'].map(mpa_dict)
    df.to_csv('./artifacts/Raw_Data_new_spp.csv',index = False)
    return df

df = load_run()



# Sagemaker Settings

region = boto3.Session().region_name
bucket = "uip-datalake-bucket-prod"
session = sagemaker.Session(default_bucket=bucket)
prefix = "gdso/datascience/users/ggupta1"
tagValue = [{"Key": "app_group", "Value": "uip_gdso_ds"}]
role = get_execution_role()
sm = boto3.client(service_name="sagemaker", region_name=region)



# Train Test Split

train_data = df[(df['SNAP_MONTH'] >= '2020-05-01') & (df['SNAP_MONTH'] <= '2021-04-01
test_data = df[(df['SNAP_MONTH'] >= '2021-05-01') & (df['SNAP_MONTH'] <= '2021-10-01
holdout_data = df[(df['SNAP_MONTH'] >= '2021-11-01')]
train_data,test_data,holdout_data = train_data.drop(columns = ['SNAP_MONTH']),test_da
test_data_no_target = test_data.drop(columns=["QUOTA_ATT"])
# Upload to S3
train_file = "train_data.csv"
```

```
# df_majority = train_data[train_data.QUOTA_ATT == 0]
# df_minority = train_data[train_data.QUOTA_ATT == 1]
# df_majority_downsampled = df_majority.sample(n=len(df_minority), random_state=42)
# train_data = pd.concat([df_majority_downsampled, df_minority])
print(train_data.QUOTA_ATT.value_counts())

train_data.to_csv(train_file, index=False, header=True)
train_data_s3_path = session.upload_data(path=train_file, key_prefix=prefix + "/train"
test_file = "test_data.csv"
test_data_no_target.to_csv(test_file, index=False, header=False)
test_data_s3_path = session.upload_data(path=test_file, key_prefix=prefix + "/test")
holdout_file = "holdout_data.csv"
holdout_data.to_csv(holdout_file, index=False, header=False)
holdout_data_s3_path = session.upload_data(path=holdout_file, key_prefix=prefix + "/ho



# Auto ML Setting


auto_ml_job_config = {"CompletionCriteria": {"MaxCandidates": 3}}
input_data_config = [
    {
        "DataSource": {
            "S3DataSource": {
                "S3DataType": "S3Prefix",
                "S3Uri": f"s3://{bucket}/{prefix}/train"
            }
        },
        "TargetAttributeName": "QUOTA_ATT",
    }
]
output_data_config = {"S3OutputPath": f"s3://{bucket}/{prefix}/output"}


timestamp_suffix = strftime("%d-%H-%M-%S", gmtime())
auto_ml_job_name = f"automl-seed-{timestamp_suffix}"
# Create AutoML job
sm.create_auto_ml_job(
    AutoMLJobName=auto_ml_job_name,
    InputDataConfig=input_data_config,
    OutputDataConfig=output_data_config,
    AutoMLJobConfig=auto_ml_job_config,
    RoleArn=role,
    Tags=tagValue
)
# Monitor job
while True:
    describe_response = sm.describe_auto_ml_job(AutoMLJobName=auto_ml_job_name)
    job_status = describe_response["AutoMLJobStatus"]
    if job_status in ("Failed", "Completed", "Stopped"):
        break
    sleep(30)
# Get and print best candidate
```

```
best_candidate = describe_response["BestCandidate"]
print("Best Candidate:", best_candidate)
# Create model
model_name = f"automl-seed-model-{timestamp_suffix}"

best_candidate_containers = best_candidate['InferenceContainers']

best_candidate_containers[1]['Environment'].update({'SAGEMAKER_INFERENCE_OUTPUT': 'pred
best_candidate_containers[2]['Environment'].update({'SAGEMAKER_INFERENCE_INPUT': 'predi
best_candidate_containers[2]['Environment'].update({'SAGEMAKER_INFERENCE_OUTPUT': 'pred


model = sm.create_model(
    Containers=best_candidate["InferenceContainers"],
    ModelName=model_name,
    ExecutionRoleArn=role,
    Tags=tagValue
)



# Create transform job
transform_job_name = f"automl-seed-transform-{timestamp_suffix}"
transform_input = {
    "DataSource": {"S3DataSource": {"S3DataType": "S3Prefix", "S3Uri": test_data_s3_pa
    "ContentType": "text/csv",
    "CompressionType": "None",
    "SplitType": "Line"
}
transform_output = {"S3OutputPath": f"s3://{bucket}/{prefix}/inference-results"}
transform_resources = {"InstanceType": "ml.m5.4xlarge", "InstanceCount": 1}
sm.create_transform_job(
    TransformJobName=transform_job_name,
    ModelName=model_name,
    TransformInput=transform_input,
    TransformOutput=transform_output,
    TransformResources=transform_resources,
    Tags=tagValue
)
# Monitor transform job
while True:
    describe_response = sm.describe_transform_job(TransformJobName=transform_job_name
    job_status = describe_response["TransformJobStatus"]
    if job_status in ("Failed", "Completed", "Stopped"):
        break
    sleep(30)
# Download and print results
s3_output_key = f"{prefix}/inference-results/test_data.csv.out"
local_inference_results_path = "inference_results.csv"
s3 = boto3.resource("s3")
inference_results_bucket = s3.Bucket(session.default_bucket())
inference_results_bucket.download_file(s3_output_key, local_inference_results_path)
```

```
results = pd.read_csv(local_inference_results_path, sep=";")
print(results)
# List all candidates
candidates = sm.list_candidates_for_auto_ml_job(AutoMLJobName=auto_ml_job_name,
                                                SortBy="FinalObjectiveMetricValue")["C
for idx, candidate in enumerate(candidates, 1):
    print(f"{idx}  {candidate['CandidateName']}  {candidate['FinalAutoMLJobObjectiveMe


# Best Model

model = next(iter(best_candidate.items()))
print("Best Model:", model)


# Deploy Sagemaker model

import boto3
import sagemaker
import time

# Initial Setup
region = boto3.Session().region_name
bucket = "uip-datalake-bucket-prod"
session = sagemaker.Session(default_bucket=bucket)
sm = boto3.client(service_name="sagemaker", region_name=region)
client = boto3.client('sagemaker')

# Define a function to split the data into smaller batches
def split_dataframe(df, batch_size):
    """Split a DataFrame into smaller DataFrames of size batch_size."""
    num_batches = len(df) // batch_size + (1 if len(df) % batch_size else 0)
    return [df[i*batch_size:(i+1)*batch_size] for i in range(num_batches)]

# AutoML Job Details
automl_job_name = auto_ml_job_name  # Replace with your AutoML job name
best_candidate = client.describe_auto_ml_job(AutoMLJobName=automl_job_name)['BestCandi

# Model Creation
model_name = 'seed-model-test-{}'.format(int(time.time()))
role = role

best_candidate_containers[1]['Environment']['SAGEMAKER_INFERENCE_OUTPUT'] = 'predicted
best_candidate_containers[2]['Environment']['SAGEMAKER_INFERENCE_OUTPUT'] = 'predicted
response = sm.create_model(
    ModelName=model_name,
    Containers=best_candidate_containers,  # Make sure you're using this updated list
    ExecutionRoleArn=role,
    Tags=[{"Key": "app_group", "Value": "uip_gdso_ds"}]
)


# Endpoint Configuration & Deployment
endpoint_config_name = 'seed-endpoint-config-test-ggupta1-{}'.format(int(time.time()))
response = sm.create_endpoint_config(
```

```
    EndpointConfigName=endpoint_config_name,
    ProductionVariants=[
        {
            'VariantName': 'AllTraffic',
            'ModelName': model_name,
            'InitialInstanceCount': 1,
            'InstanceType': 'ml.m5.4xlarge',
            'InitialVariantWeight': 1
        },
    ],
    Tags=[{"Key": "app_group", "Value": "uip_gdso_ds"}]
)
endpoint_name = 'seed-endpoint-test-ggupta-{}'.format(int(time.time()))
response = sm.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name,
    Tags=[{"Key": "app_group", "Value": "uip_gdso_ds"}]
)



# Wait for the endpoint to be in 'InService' state
time.sleep(10)  # A short delay to give AWS time to start deploying.
while sm.describe_endpoint(EndpointName=endpoint_name)['EndpointStatus'] == 'Creating':
    time.sleep(60)




# # Iterate over each batch and invoke the endpoint
# runtime = boto3.client('runtime.sagemaker')
# for batch in batches:
#     test_data_batch = batch.to_csv(index=False).encode('utf-8')
#     response = runtime.invoke_endpoint(EndpointName=endpoint_name, ContentType='text
#     prediction_batch = response['Body'].read().decode('utf-8').splitlines()
#     all_predictions.extend(prediction_batch)

# Now, all_predictions will contain the predictions for all the batches

# prob = [round(float(p), 2) for p in prob]


### Prediction


%%time
# Assuming df is defined and has the required data
holdout = df[df['SNAP_MONTH'] == '2023-09-01']
holdout_no_target = holdout.drop(columns=['SNAP_MONTH', 'QUOTA_ATT'])
batches = split_dataframe(holdout_no_target, 1000)

label=[]
prob=[]

# Iterate over each batch and invoke the endpoint
```

```
runtime = boto3.client('runtime.sagemaker')
for batch in batches:
    test_data_batch = batch.to_csv(index=False).encode('utf-8')
    response = runtime.invoke_endpoint(
        EndpointName=endpoint_name,
        ContentType='text/csv',
        Body=test_data_batch,
        Accept='application/jsonlines'  # Modified this line
    )
    la=[]
    pr=[]
    prediction_batch = response['Body'].read().decode('utf-8').splitlines()
    for line in prediction_batch:
        data = json.loads(line)
        la.append(data['predicted_label'])
        pr.append(data['probability'])
    label.append(la[1:])
    prob.append(pr[1:])
label = [item for sublist in label for item in sublist]
prob = [item for sublist in prob for item in sublist]
y_truth = holdout['QUOTA_ATT'].to_list()

prob = [round(float(p), 2) for p in prob]
pred = list(map(int, prediction))
print("ROC AUC Score:", roc_auc_score(list(y_truth),list(prob)))


# Clear EndPoint

# import boto3

# # Initialize the SageMaker client
# sagemaker_client = boto3.client('sagemaker')

# # List all endpoints
# endpoints = sagemaker_client.list_endpoints(MaxResults=100)['Endpoints']  # MaxResul

# # Print and delete each endpoint
# for endpoint in endpoints:
#     endpoint_name = endpoint['EndpointName']
#     print(f"Deleting endpoint: {endpoint_name}")

#     # Delete the endpoint
#     sagemaker_client.delete_endpoint(EndpointName=endpoint_name)
```