Setting up the role, bucket and region

In [1]:
```
import os
import boto3
import re
import sagemaker
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/co
re/computation/expressions.py:21: UserWarning: Pandas requires version '2.8.
0' or newer of 'numexpr' (version '2.7.3' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED

sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sa
gemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-u
ser/.config/sagemaker/config.yaml
```

In [2]:
```
role = sagemaker.get_execution_role()
region = boto3.Session().region_name

# S3 bucket for saving code and model artifacts.
bucket = "project01-ml-pipeline-bucket"

prefix = (
    "sagemaker/lung-cancer-prediction"
)
print (region)
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sa
gemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-u
ser/.config/sagemaker/config.yaml
us-east-1
```

In [3]:
```
role
```

Out[3]:
```
'arn:aws:iam::193734792448:role/fast-ai-academic-59-Student-Azure'
```

Getting the data and then analysising it

In [4]:
```
import pandas as pd

# loading the dataset
data = pd.read_csv("lung_cancer_ds.csv")
```

In [5]:
```
data
```

Out[5]:

	index	Patient Id	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPational Hazards	Genetic Risk	chronic Lung Disease
0	0	P1	33	1	2	4	5	4	3	2
1	1	P10	17	1	3	1	5	3	4	2
2	2	P100	35	1	4	5	6	5	5	4
3	3	P1000	37	1	7	7	7	7	6	7
4	4	P101	46	1	6	8	7	7	7	6
...	...	...	...	...	...	...	...	...	...	...
995	995	P995	44	1	6	7	7	7	7	6
996	996	P996	37	2	6	8	7	7	7	6
997	997	P997	25	2	4	5	6	5	5	4
998	998	P998	18	2	6	8	7	7	7	6
999	999	P999	47	1	6	5	6	5	5	4

1000 rows × 26 columns

In [6]:
```
data.shape
```

Out[6]:
```
(1000, 26)
```

In [7]:
```
data.columns
```

Out[7]:
```
Index(['index', 'Patient Id', 'Age', 'Gender', 'Air Pollution', 'Alcohol us
e',
       'Dust Allergy', 'OccuPational Hazards', 'Genetic Risk',
       'chronic Lung Disease', 'Balanced Diet', 'Obesity', 'Smoking',
       'Passive Smoker', 'Chest Pain', 'Coughing of Blood', 'Fatigue',
       'Weight Loss', 'Shortness of Breath', 'Wheezing',
       'Swallowing Difficulty', 'Clubbing of Finger Nails', 'Frequent Cold',
       'Dry Cough', 'Snoring', 'Level'],
      dtype='object')
```

In [8]:
```
data.describe()
```

Out[8]:

	index	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPation Hazar
count	1000.000000	1000.000000	1000.000000	1000.0000	1000.000000	1000.000000	1000.0000
mean	499.500000	37.174000	1.402000	3.8400	4.563000	5.165000	4.8400
std	288.819436	12.005493	0.490547	2.0304	2.620477	1.980833	2.1078
min	0.000000	14.000000	1.000000	1.0000	1.000000	1.000000	1.0000
25%	249.750000	27.750000	1.000000	2.0000	2.000000	4.000000	3.0000
50%	499.500000	36.000000	1.000000	3.0000	5.000000	6.000000	5.0000
75%	749.250000	45.000000	2.000000	6.0000	7.000000	7.000000	7.0000
max	999.000000	73.000000	2.000000	8.0000	8.000000	8.000000	8.0000

8 rows × 24 columns

In [9]:
```
# checking for missing values
data.isnull().sum()
```

Out[9]:
```
index                     0
Patient Id                0
Age                       0
Gender                    0
Air Pollution             0
Alcohol use               0
Dust Allergy              0
OccuPational Hazards      0
Genetic Risk              0
chronic Lung Disease      0
Balanced Diet             0
Obesity                   0
Smoking                   0
Passive Smoker            0
Chest Pain                0
Coughing of Blood         0
Fatigue                   0
Weight Loss               0
Shortness of Breath       0
Wheezing                  0
Swallowing Difficulty     0
Clubbing of Finger Nails  0
Frequent Cold             0
Dry Cough                 0
Snoring                   0
Level                     0
dtype: int64
```

Cleaning the data

In [10]:
```
# Dropping columns 'Patient Id', index
data.drop(['Patient Id', 'index'], axis = 1, inplace = True)
```

In [11]:
```
data
```

Out[11]:

	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPational Hazards	Genetic Risk	chronic Lung Disease	Balanced Diet	Obes
0	33	1	2	4	5	4	3	2	2	
1	17	1	3	1	5	3	4	2	2	
2	35	1	4	5	6	5	5	4	6	
3	37	1	7	7	7	7	6	7	7	
4	46	1	6	8	7	7	7	6	7	
...	...	...	...	...	...	...	...	...	...	
995	44	1	6	7	7	7	7	6	7	
996	37	2	6	8	7	7	7	6	7	
997	25	2	4	5	6	5	5	4	6	
998	18	2	6	8	7	7	7	6	7	
999	47	1	6	5	6	5	5	4	6	

1000 rows × 24 columns

In [12]:
```
# Converting the 'Level' column to numerical values
def level_numerical(value):
    if value == 'Low':
        return 0
    elif value == 'Medium':
        return 1
    else:
        return 2

data["Level_numerical"] = data.Level.apply(level_numerical)
```

In [13]:
```
data
```

Out[13]:

	Age	Gender	Air Pollution	Alcohol use	Dust Allergy	OccuPational Hazards	Genetic Risk	chronic Lung Disease	Balanced Diet	Obes
0	33	1	2	4	5	4	3	2	2	
1	17	1	3	1	5	3	4	2	2	
2	35	1	4	5	6	5	5	4	6	
3	37	1	7	7	7	7	6	7	7	
4	46	1	6	8	7	7	7	6	7	
...	...	...	...	...	...	...	...	...	...	
995	44	1	6	7	7	7	7	6	7	
996	37	2	6	8	7	7	7	6	7	
997	25	2	4	5	6	5	5	4	6	
998	18	2	6	8	7	7	7	6	7	
999	47	1	6	5	6	5	5	4	6	

1000 rows × 25 columns

In [14]:
```
# Dropping column Level
data.drop('Level', axis = 1, inplace = True)
```

In [15]:
```
data.shape
```

Out[15]:
```
(1000, 24)
```

Splitting the data into 80% training, 10% validation and 10% testing

In [16]:
```
import numpy as np

rand_split = np.random.rand(len(data))

train_list = rand_split < 0.8
val_list = (rand_split >=0.8) & (rand_split < 0.9)
test_list = rand_split >= 0.9

# Creating the dataset for training, validation and testing
train_data = data[train_list]
val_data = data[val_list]
test_data = data[test_list]

# 1. Separating into input features and output class
# This code gets all the columns first to last-1 as the input features while 1

# 1. Training data
train_x = train_data.iloc[:, :- 1]
train_y = train_data.iloc[:, -1]

# 2. Validation data
val_x = val_data.iloc[:, :- 1]
val_y = val_data.iloc[:, -1]

# 3. Testing data
test_x = test_data. iloc[:, :- 1]
test_y = test_data.iloc[:, -1]

# 2. Convert DataFrame to NumPy array
# 1. Training data
train_x = train_x.to_numpy().astype("float32")
train_y = train_y.to_numpy().astype("float32")

# 2. Validation data
val_x = val_x.to_numpy().astype("float32")
val_y = val_y.to_numpy().astype("float32")

# 3. Testing data
test_x = test_x.to_numpy().astype("float32")
test_y = test_y.to_numpy().astype("float32")
```

In [17]:
```
# Checking whether all the splitting are done correctly
train_x.shape, train_y.shape, val_x.shape, val_y.shape, test_x.shape, test_y.s
```

Out[17]:
```
((808, 23), (808,), (91, 23), (91,), (101, 23), (101,))
```

Converting the file to protobuf format

In [18]:
```
import io
import sagemaker.amazon.common as smac
import time
```

In [19]:
```
# Training file
train_file = "linear_train_lung.data"

f = io.BytesIO()

smac.write_numpy_to_dense_tensor(f, train_x.astype("float32"), train_y.astype(
f.seek(0)

boto3.Session().resource("s3").Bucket(bucket).Object(os.path.join(prefix, "tra
```

In [20]:
```
# Validation file
val_file = "linear_validation_lung.data"

f = io.BytesIO()

smac.write_numpy_to_dense_tensor(f, val_x.astype("float32"), val_y.astype("flo
f.seek(0)

boto3.Session().resource("s3").Bucket(bucket).Object(os.path.join(prefix, "val
```

In [21]:
```
# test file
test_file = "linear_test_lung.data"

f = io.BytesIO()

smac.write_numpy_to_dense_tensor(f, test_x.astype("float32"), test_y.astype("-
f.seek(0)

boto3.Session().resource("s3").Bucket(bucket).Object(os.path.join(prefix, "tes
```

Training the model

In [22]:
```
from sagemaker import image_uris

# getting the container image of linear learner algorithm
container = image_uris.retrieve(region=boto3.Session().region_name, framework=
```

In [23]:
```
linear_job = "Project01-linear-" + time.strftime("%Y-%m-%d-%H-%M-%S", time.gm

print("Job name is: ", linear_job)

linear_training_params = {
    "RoleArn": role,
    "TrainingJobName": linear_job,
    "AlgorithmSpecification": {"TrainingImage": container, "TrainingInputMode
    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.c4.2xlarge", "
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://{}/{}/train/".format(bucket, prefix),
                    "S3DataDistributionType": "ShardedByS3Key",
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None",
        },
        {
            "ChannelName": "validation",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://{}/{}/validation/".format(bucket, prefix),
                    "S3DataDistributionType": "FullyReplicated",
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None",
        },
    ],
    "OutputDataConfig": {"S3OutputPath": "s3://{}/{}/".format(bucket, prefix)
    "HyperParameters": {
        'feature_dim': '23',
        'mini_batch_size': '100',
        'predictor_type': 'regressor',
        'epochs': '100',
        'num_models': '256',
        'loss': 'squared_loss',
        'learning_rate': '0.01',
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 60 * 60},
}
```

```
Job name is:  Project01-linear-2023-12-12-22-03-15
```

In [24]:
```
region = boto3.Session().region_name
sm = boto3.client("sagemaker")

sm.create_training_job(**linear_training_params)

status = sm.describe_training_job(TrainingJobName = linear_job)["TrainingJobSt
print(status)
sm.get_waiter("training_job_completed_or_stopped").wait(TrainingJobName = line

if status == "Failed":
    message = sm.describe_training_job(TrainingJobName = linear_job)["FailureR
    print("Training failed with the following error: {}".format(message))
    raise Exception("Training job failed")
```

```
InProgress
```

Hosting

In [25]:
```
linear_hosting_container = {
    "Image": container,
    "ModelDataUrl": sm.describe_training_job(TrainingJobName=linear_job)["Mode
        "S3ModelArtifacts"
    ],
}

create_model_response = sm.create_model(
    ModelName=linear_job, ExecutionRoleArn=role, PrimaryContainer=linear_hosti
)

print(create_model_response["ModelArn"])
```

```
arn:aws:sagemaker:us-east-1:193734792448:model/project01-linear-2023-12-12-2
2-03-15
```

In [26]:
```
# Endpoint configuration
linear_endpoint_config = "DEMO-linear-endpoint-config-" + time.strftime(
    "%Y-%m-%d-%H-%M-%S", time.gmtime()
)
print(linear_endpoint_config)
create_endpoint_config_response = sm.create_endpoint_config(
    EndpointConfigName=linear_endpoint_config,
    ProductionVariants=[
        {
            "InstanceType": "ml.m4.xlarge",
            "InitialInstanceCount": 1,
            "ModelName": linear_job,
            "VariantName": "AllTraffic",
        }
    ],
)

print("Endpoint Config Arn: " + create_endpoint_config_response["EndpointConfi
```

```
DEMO-linear-endpoint-config-2023-12-12-22-07-17
Endpoint Config Arn: arn:aws:sagemaker:us-east-1:193734792448:endpoint-confi
g/demo-linear-endpoint-config-2023-12-12-22-07-17
```

In [27]:
```
# Endpoint creation
linear_endpoint = "DEMO-linear-endpoint-" + time.strftime("%Y%m%d%H%M", time.g
print(linear_endpoint)
create_endpoint_response = sm.create_endpoint(
    EndpointName=linear_endpoint, EndpointConfigName=linear_endpoint_config
)
print(create_endpoint_response["EndpointArn"])

resp = sm.describe_endpoint(EndpointName=linear_endpoint)
status = resp["EndpointStatus"]
print("Status: " + status)

sm.get_waiter("endpoint_in_service").wait(EndpointName=linear_endpoint)

resp = sm.describe_endpoint(EndpointName=linear_endpoint)
status = resp["EndpointStatus"]
print("Arn: " + resp["EndpointArn"])
print("Status: " + status)

if status != "InService":
    raise Exception("Endpoint creation did not succeed")
```

```
DEMO-linear-endpoint-202312122207
arn:aws:sagemaker:us-east-1:193734792448:endpoint/demo-linear-endpoint-20231
2122207
Status: Creating
Arn: arn:aws:sagemaker:us-east-1:193734792448:endpoint/demo-linear-endpoint-
202312122207
Status: InService
```

Prediction

In [28]:
```
# Converting the data to csv format
def np2csv(arr):
    csv = io.BytesIO()
    np.savetxt(csv, arr, delimiter=",", fmt="%g")
    return csv.getvalue().decode().rstrip()
```

In [29]:
```
# Testing the test data on our trained model
import json
import numpy as np

runtime = boto3.client("runtime.sagemaker")

payload = np2csv(test_x)

response = runtime.invoke_endpoint(
    EndpointName=linear_endpoint, ContentType="text/csv", Body=payload
)
result = json.loads(response["Body"].read().decode())
test_pred = np.array([r["score"] for r in result["predictions"]])
```

In [30]:
```
print (test_pred)
```

```
[ 0.99423122  0.14608788  0.19017959  0.19475055  1.82959175  0.95157528
  1.99286866  1.88949323  0.02950323  0.69260383  1.37502265  1.93845201
  1.03420186  1.88405657  0.41554689 -0.06257772  0.26215243  0.08908057
  1.0163219   0.10214639  1.04844499  0.94728518  0.98996568  0.9183557
  0.41554689  1.82959175  0.94728518  0.91720796  0.93309116  0.82033849
  0.89696836  0.95157528  1.61685395  2.06081438  2.28564858  1.0163219
 -0.34289384  0.88701534  0.92513967  1.05979538  0.02950323  2.07378936
  1.38924146  0.41554689  0.02950323  0.88701534 -0.38277149 -0.09351754
 -0.12829733  0.91720796  1.02233481  0.95157528  1.80684209  1.7058785
  1.88949323  0.02950323 -0.06025708 -0.1538595   0.9183557   1.04844499
  0.79298949  0.82033849  1.05979538  0.95157528  1.89696264  1.82025456
  1.37502265  1.02233481  1.88405657  0.22276306 -0.17929268  0.91720796
  1.05979538  0.95157528  2.06241441  2.02130604  2.06241441  0.91720796
  0.72468591 -0.34289384 -0.06672406  0.03386855  0.03027368  0.10214639
 -0.17929268  0.94728518  0.93309116  0.9183557   1.02233481  1.84851956
  0.69260383  2.48103213  1.82959175  2.28564858  0.23750377 -0.06257772
  1.89863992  0.92513967  0.93309116  1.85963392  1.94023442]
```

In [31]:
```
test_mae_linear = np.mean(np.abs(test_y - test_pred))
test_mae_baseline = np.mean(np.abs(test_y - np.median(train_y)))

print("Test MAE Baseline :", round(test_mae_baseline, 3))
print("Test MAE Linear:", round(test_mae_linear, 3))
```

```
Test MAE Baseline : 0.564
Test MAE Linear: 0.14
```

In [32]:
```
test_pred_class = (test_pred > 0.5) + 0

test_pred_baseline = np.repeat(np.median(train_y), len(test_y))

prediction_accuracy = np.mean((test_y == test_pred_class)) * 100
baseline_accuracy = np.mean((test_y == test_pred_baseline)) * 100

print("Prediction Accuracy:", round(prediction_accuracy, 1), "%")
print("Baseline Accuracy:", round(baseline_accuracy, 1), "%")
```

```
Prediction Accuracy: 73.3 %
Baseline Accuracy: 43.6 %
```

In [33]:
```
sm.delete_endpoint(EndpointName=linear_endpoint)
```

Out[33]:
```
{'ResponseMetadata': {'RequestId': 'b63188bd-fc95-4ce8-a93c-b13c5d718d03',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'b63188bd-fc95-4ce8-a93c-b13c5d718d0
3',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '0',
   'date': 'Tue, 12 Dec 2023 22:11:49 GMT'},
  'RetryAttempts': 0}}
```