```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "098da034",
   "metadata": {},
   "source": [
    "# Assignment4 : Setting Up a Machine Learning Pipeline in The Cloud"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "12f56985",
   "metadata": {},
   "source": [
    "## Description\n",
    "\n",
    "In this assignment, you will learn how to train, save, load, deploy and run a machine learning model in a cloud environment. You will use the Amazon SageMaker SDK as a use case. SageMaker SDK enables you to train a model in \"script mode\" and deploy the endpoint. Let's start with a simple ML code to learn how it can be run and deployed in the cloud. Next, you will need to deploy and run a more complex ML code in the cloud using the steps you learnt.  "
   ]
  },
  {
   "cell_type": "markdown",
   "id": "acdb4444",
   "metadata": {},
   "source": [
    "## Learning Outcomes:\n",
    "After completing this assignment, you should be able to:\n",
    "- Gain experience working with SageMaker SDK.\n",
    "- Understand the basics of machine learning pipeline in the cloud environment.\n",
    "- Learn how to set up and deploy a machine learning pipeline for ML models."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "ff7b2c1b",
   "metadata": {},
   "source": [
    "How long does it take to train and test your model on a local host?"
```

```
   ]
  },
  {
   "cell_type": "markdown",
   "id": "8724b87b",
   "metadata": {},
   "source": [
    "## Step 1.0 Training a Regression model on a local host"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "b89bb661",
   "metadata": {},
   "source": [
    "Ensure you have Python 3.10 or higher. If you are not sure how to
check you can usually in most terminals you can run \"python --version\""
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "861e67b7",
   "metadata": {
    "vscode": {
     "languageId": "shellscript"
    }
   },
   "outputs": [],
   "source": [
    "pip install -U scikit-learn "
   ]
  },
  {
   "cell_type": "markdown",
   "id": "c48b4019",
   "metadata": {},
   "source": [
    "### 1- Create a dataset and save"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "4e1b2241",
   "metadata": {},
```

```
 "outputs": [],
 "source": [
  "from sklearn import datasets\n",
  "import pickle"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "d83b2d29",
 "metadata": {},
 "outputs": [],
 "source": [
  "X, y = datasets.make_regression(100, 1, noise=5, bias=0)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "c25527fc",
 "metadata": {},
 "outputs": [],
 "source": [
  "pickle.dump([X,y], open('./train.pickle', 'wb'))"
 ]
},
{
 "cell_type": "markdown",
 "id": "a601e752",
 "metadata": {},
 "source": [
  "### 2- Create a model from the dataset"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "8aa04224",
 "metadata": {},
 "outputs": [],
 "source": [
  "from sklearn.linear_model import LinearRegression\n",
  "import pickle"
 ]
},
{
```

```
 "cell_type": "code",
 "execution_count": null,
 "id": "00caf709",
 "metadata": {},
 "outputs": [],
 "source": [
  "[XX, yy] = pickle.load(open('./train.pickle', 'rb'))"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "fc9f9158",
 "metadata": {},
 "outputs": [],
 "source": [
  "model = LinearRegression()"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "47ba5aca",
 "metadata": {},
 "outputs": [],
 "source": [
  "model.fit(XX,yy)"
 ]
},
{
 "cell_type": "markdown",
 "id": "0f0ba432",
 "metadata": {},
 "source": [
  "### 3- Make a test prediction"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "ee21fc81",
 "metadata": {},
 "outputs": [],
 "source": [
  "model.predict([[0],[1],[2],[3]])"
 ]
```

```
},
{
 "cell_type": "markdown",
 "id": "e8d7df6e",
 "metadata": {},
 "source": [
  "### 4- Save the model to a file"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "fb971804",
 "metadata": {},
 "outputs": [],
 "source": [
  "p = pickle.dumps(model)"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "fe2d9f92",
 "metadata": {},
 "outputs": [],
 "source": [
  "pickle.dump(model, open('./model.pickle', 'wb'))"
 ]
},
{
 "cell_type": "markdown",
 "id": "cbaa7fed",
 "metadata": {},
 "source": [
  "### 5- Later load the model from a file"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "be9d2abf",
 "metadata": {},
 "outputs": [],
 "source": [
  "from sklearn.linear_model import LinearRegression\n",
  "import pickle"
```

```
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "94e63048",
 "metadata": {},
 "outputs": [],
 "source": [
  "loaded_model = pickle.load(open('./model.pickle', 'rb'))"
 ]
},
{
 "cell_type": "markdown",
 "id": "8cbb920e",
 "metadata": {},
 "source": [
  "### 6- Make a test prediction"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "65309fa2",
 "metadata": {},
 "outputs": [],
 "source": [
  "loaded_model.predict([[0],[1],[2],[3]])"
 ]
},
{
 "cell_type": "markdown",
 "id": "59f878f0",
 "metadata": {},
 "source": [
  "How long does it take to train and test your model on a local host?"
 ]
},
{
 "cell_type": "markdown",
 "id": "fa84a8bd",
 "metadata": {},
 "source": [
  "## Step 2.0 Training a Regression Model in the Cloud\n",
  "\n",
  "One of the easiest ways to run a training job in the cloud is the use
```

```
of SageMaker Notebooks. Sagemaker runs and trains custom models in Python by executing a Python script. In the following, You will learn how to apply the entire lifecycle of an ML application (load data, train the model, save the model, load the model, predictions) using SageMaker Notebooks. SageMaker automatically uses an EC2 instance to handle the training job. We use an S3 bucket to save the data and model and use it for predictions in the future. \n",
    "\n",
    "This is broken into three parts:\n",
    "1) Setting up Sagemaker Notebook in AWS\n",
    "2) Creating Sagemaker Script\n",
    "3) Creating Deployment Script\n",
    "\n",
    "\n",
    "\n",
    "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "6ce7f272",
   "metadata": {},
   "source": [
    "## Step 2.1 Setting up Sagemaker In AWS\n",
    "\n",
    "### 1) Create a Sagemaker Notebook\n",
    "Open your AWS management console.\n",
    "- Search SageMaker in the services. \n",
    "- Go to the SageMaker menu on the left and click on the option \"Notebook\" then in the drop down click on “Notebook instances”.\n",
    "- Create a Notebook instance.\n",
    "- Choose default options except for IAM role \n",
    "- For IAM role choose \"create a new role\". A pop up will appear with some preselected options - don't change any of these. Click on the button \"Create role\" and create the instance (it will take a couple of minutes that status is changed to \"InService\"). \n",
    "- Once the status changes from \"Pending\" to \"InService\", press \"Open JupyterLab\"\n",
    "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "7ed45790",
   "metadata": {},
   "source": [
```

```
    "### 2) Create Folder Structure Sagemaker\n",
    "Once inside Jupyter labs you will need to create some new file / folders and upload some files created from the previous section.\n",
    "\n",
    "Upload these files into sagemaker:\n",
    "- train.pickle\n",
    "\n",
    "<br/>\n",
    "\n",
    "Folder Structure:\n",
    "\n",
    "``` bash\n",
    "train.pickle (file - Step 1.0)\n",
    "regressionDeployment.ipynb (file - Step 2.2)\n",
    "\n",
    "script (folder)\n",
    "    regressionScript.py (file - Step 2.3)\n",
    "```\n",
    "\n",
    "<br/>"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "b7163582",
   "metadata": {},
   "source": [
    "## Step 2.2 Creating Sagemaker Script\n",
    "\n",
    "### <span style=\"color:red\">You copy the code below combine all of these scripts together to create one training script called: <br/> <b>regressionScript.py </b></script>\n",
    "\n",
    "Sage Maker Scripts can be broken down into 5 distinct parts. \n",
    "1) Creating Arguments\n",
    "2) Load Dataset\n",
    "3) Load Model\n",
    "4) Predict with Model\n",
    "5) Main Function"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "f96cf94c",
   "metadata": {},
   "source": [
```

```
    "### 1. Creating Arguments"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "863c2688",
   "metadata": {},
   "source": [
    "- Your your training script will contain a function called \"parse_args\" that will save environment variables to your training file \n",
    "- These arguments will act like environment variables and will be utilized throughout the rest of your script\n",
    "- Many of these arguments will be utilized as file paths to upload your artifacts (model, training data, etc) into an s3 bucket "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "b5d64ecb",
   "metadata": {},
   "outputs": [],
   "source": [
    "import argparse\n",
    "import os\n",
    "from sklearn.linear_model import LinearRegression\n",
    "import pickle\n",
    "\n",
    "def parse_args():\n",
    "    \"\"\"\n",
    "    Parse arguments.\n",
    "    \"\"\"\n",
    "    parser = argparse.ArgumentParser()\n",
    "\n",
    "    # hyperparameters sent by the client are passed as command-line arguments to the script\n",
    "    # We don't use these but I left them in as a useful template for future development\n",
    "    parser.add_argument(\"--copy_X\",        type=bool, default=True)\n",
    "    parser.add_argument(\"--fit_intercept\", type=bool, default=True)\n",
    "    parser.add_argument(\"--normalize\",     type=bool, default=False)\n",
    "    \n",
```

```
    "    # data directories\n",
    "    parser.add_argument(\"--train\", type=str,
default=os.environ.get(\"SM_CHANNEL_TRAIN\"))\n",
    "    parser.add_argument(\"--test\", type=str,
default=os.environ.get(\"SM_CHANNEL_TEST\"))\n",
    "\n",
    "    # model directory: we will use the default set by SageMaker,
/opt/ml/model\n",
    "    parser.add_argument(\"--model_dir\", type=str,
default=os.environ.get(\"SM_MODEL_DIR\"))\n",
    "\n",
    "    return parser.parse_known_args()"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "ef44db1a",
   "metadata": {},
   "source": [
    "### Important Arguments Explained (Don't copy code below)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "9fe916d1",
   "metadata": {},
   "outputs": [],
   "source": [
    "# model_dir \n",
    "# - The most important argument. \n",
    "# - File Path where training job writes the model artifacts to\n",
    "\n",
    "# - SM_MODEL_DIR - ALWAYS SET TO \"/opt/ml/model\"\n",
    "#                - Sagemaker looks for saves the artifacts in this
directory as a .tar.gz file\n",
    "parser.add_argument(\"--model_dir\", type=str,
default=os.environ.get(\"SM_MODEL_DIR\"))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "dce2a64f",
   "metadata": {},
   "outputs": [],
```

```
   "source": [
    "# train\n",
    "# - Stores the location of the training data\n",
    "# - Stores the s3 string to your training data that you passed when
calling the .fit function\n",
    "parser.add_argument(\"--train\", type=str,
default=os.environ.get(\"SM_CHANNEL_TRAIN\"))"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "3ca029ec",
   "metadata": {},
   "source": [
    "### 2. Load Dataset"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "09423794",
   "metadata": {},
   "source": [
    "After creating arguments we want to load our training data onto the
instance using the train argument that we have.\n",
    "\n",
    "So, we will create functions to load training (and optionally testing)
data. The way you define these functions\n",
    "depends on how your data is stored but it should be fairly easy to
create your own version. Here the argument path is the location of the S3
bucket containing your training (and test) data. If you do not have offline
training data and want to import data from a repository you can do so by
simply ignoring the argument and directly\n",
    "importing your data."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "28f51e63",
   "metadata": {},
   "outputs": [],
   "source": [
    "def load_dataset(path):\n",
    "    \"\"\"\n",
    "    Load entire dataset.\n",
    "    \"\"\"\n",
```

```
    "    # Find all files with a pickle ext but we only load the first one
in this sample:\n",
    "    files = [os.path.join(path, file) for file in os.listdir(path) if
file.endswith(\"pickle\")]\n",
    "\n",
    "    if len(files) == 0:\n",
    "        raise ValueError(\"Invalid # of files in dir:
{}\".format(path))\n",
    "    \n",
    "    [X, y] = pickle.load(open(files[0], 'rb'))\n",
    "    \n",
    "    return X, y"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "07e32615",
   "metadata": {},
   "source": [
    "### 3) Load Model"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "5ce7b179",
   "metadata": {},
   "source": [
    "We want to have a function that stores and returns a regression model
"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "7a12675d",
   "metadata": {},
   "outputs": [],
   "source": [
    "def model_fn(model_dir):\n",
    "    \"\"\"\n",
    "    Load the model for inference\n",
    "    \"\"\"\n",
    "    loaded_model = pickle.load(open(model_dir + \"/model.pickle\",
'rb'))\n",
    "    return loaded_model\n",
    "\n"
```

```
 ]
},
{
 "cell_type": "markdown",
 "id": "bf62bacb",
 "metadata": {},
 "source": [
  "### 4) Predict With Model"
 ]
},
{
 "cell_type": "markdown",
 "id": "1d41a72f",
 "metadata": {},
 "source": [
  "You will use a prediction function for future inferences. "
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "7beef349",
 "metadata": {},
 "outputs": [],
 "source": [
  "def predict_fn(input_data, model):\n",
  "    \"\"\"\n",
  "    Apply model to the incoming request\n",
  "    \"\"\"\n",
  "    return model.predict(input_data)"
 ]
},
{
 "cell_type": "markdown",
 "id": "cd9c175d",
 "metadata": {},
 "source": [
  "### 5) Main Function"
 ]
},
{
 "cell_type": "markdown",
 "id": "8650976b",
 "metadata": {},
 "source": [
  "The training script will have a main function that will combine all of
```

```
these functions and train the model. <br/><br/>\n",
    "When you launch the training job through this training script,
whatever is executed on the training script constitutes the training job.
So you can customize this as much as you want. <br/><br/>However, it is
important to note that to save things and access them outside of this
training job, you must make use of the argument args.model dir. Note that
this is where we are saving the trained model. We define this later when
running the training script through the notebook to be ”/opt/ml/model” and
this is usually the directory where Sagemaker looks at and saves the
artifacts in this directory as a .tar.gz file"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "7007811b-23d6-414c-b543-01b08540c168",
   "metadata": {},
   "outputs": [],
   "source": [
    "if __name__ == \"__main__\":\n",
    "    args, _ = parse_args()\n",
    "    \n",
    "    \"\"\"\n",
    "    Train a Linear Regression\n",
    "    \"\"\"\n",
    "    print(\"Training mode\")\n",
    "\n",
    "    try:\n",
    "        X_train, y_train = load_dataset(args.train)\n",
    "        # X_test, y_test = load_dataset(args.test)\n",
    "\n",
    "        hyperparameters = {\n",
    "            \"copy_X\": args.copy_X,\n",
    "            \"fit_intercept\": args.fit_intercept,\n",
    "            \"normalize\": args.normalize,\n",
    "        }\n",
    "        \n",
    "        print(\"Training...\")\n",
    "        model = LinearRegression()\n",
    "        model.set_params(**hyperparameters)\n",
    "                \n",
    "        model.fit(X_train, y_train)\n",
    "\n",
    "        pickle.dump(model, open(os.path.join(args.model_dir,
\"model.pickle\"), 'wb'))\n",
    "       \n",
```

```
    "\n",
    "    except Exception as e:\n",
    "        # Write out an error file. This will be returned as the
failureReason in the\n",
    "        # DescribeTrainingJob result.\n",
    "        trc = traceback.format_exc()\n",
    "        with open(os.path.join(output_path, \"failure\"), \"w\") as
s:\n",
    "            s.write(\"Exception during training: \" + str(e) +
\"\\\\n\" + trc)\n",
    "\n",
    "        # Printing this causes the exception to be in the training job
logs, as well.\n",
    "        print(\"Exception during training: \" + str(e) + \"\\\\n\" +
trc, file=sys.stderr)\n",
    "\n",
    "        # A non-zero exit code causes the training job to be marked as
Failed.\n",
    "        sys.exit(255)  "
   ]
  },
  {
   "cell_type": "markdown",
   "id": "c4110115",
   "metadata": {},
   "source": [
    "## Step 2.3 Creating Deployment Script"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "549ca5a2",
   "metadata": {},
   "source": [
    "### 1) Prepare the notebook to run the training job"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2011cc95",
   "metadata": {},
   "source": [
    "In the below snippet, we pass in the location to the training script
stored on our notebook instance, which is /script/script.py in our case.
Note that we can also create additional arguments (like hyperparameters) to
pass to our training script if needed. We are also creating an estimator on
```

```
an EC2 instance defined by the instance type. After we create this estimator, we are running the .fit command which passes the location of the S3 bucket of our training data to the training script and starts a training job. By default, this is sent as args.train to the training script. If you are not using offline data and will import data directly from a Python package or git repository in your training script, you may leave this argument empty and download your data within the training script."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "794e0c74",
   "metadata": {},
   "source": [
    "### <span style=\"color:red\">You copy the code below combine all of these scripts together to create one deployment notebook: <br/> <b>regressionDeployment.ipynb </b></span>\n",
    "\n",
    "Deployment Scripts can be broken down into 5 distinct parts. \n",
    "1) Creating Initial Variables\n",
    "2) Upload Training Data to S3\n",
    "3) Create Hyperparameters\n",
    "4) Estimator Parameters\n",
    "5) Running the Training Job\n",
    "6) Deploying the Model to an Endpoint\n",
    "7) Deleting The Endpoint"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2ee24025",
   "metadata": {},
   "source": [
    "### 1) Creating Initial Variables"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "7be40539",
   "metadata": {},
   "outputs": [],
   "source": [
    "# Create Sagemaker execution role\n",
    "import sagemaker\n",
    "from sagemaker.sklearn.estimator import SKLearn\n",
```

```
    "import boto3\n",
    "import os\n",
    "\n",
    "role = sagemaker.get_execution_role()\n",
    "sess = sagemaker.Session()\n",
    "bucket = sess.default_bucket()\n",
    "\n",
    "s3_prefix = \"script-mode-workflow\"\n",
    "pickle_s3_prefix = f\"{s3_prefix}/pickle\"\n",
    "pickle_s3_uri = f\"s3://{bucket}/{s3_prefix}/pickle\"\n",
    "pickle_train_s3_uri = f\"{pickle_s3_uri}/train\"\n",
    "\n",
    "train_dir = os.path.join(os.getcwd(), \"\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "22f24090",
   "metadata": {},
   "source": [
    "### 2) Upload Training Data to S3"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "ccfe3382",
   "metadata": {},
   "source": [
    "We want to upload the training data to S3, so that it's available for
SageMaker training:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "e24cc9f2",
   "metadata": {},
   "outputs": [],
   "source": [
    "s3_resource_bucket =
boto3.Session().resource(\"s3\").Bucket(bucket)\n",
    "s3_resource_bucket.Object(os.path.join(pickle_s3_prefix,
\"train.pickle\")).upload_file(\n",
    "    train_dir + \"/train.pickle\"\n",
    ")"
   ]
```

```
  },
  {
   "cell_type": "markdown",
   "id": "aa276651",
   "metadata": {},
   "source": [
    "### 3) Create Hyperparameters"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "08b12c05",
   "metadata": {},
   "outputs": [],
   "source": [
    "# This is not required as these values are the defaults:\n",
    "hyperparameters = {\n",
    "    \"copy_X\": True,\n",
    "    \"fit_intercept\": True,\n",
    "    \"normalize\": False,\n",
    "}"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "216859c5",
   "metadata": {},
   "source": [
    "### 4) Estimator Parameters"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "e75f39d3",
   "metadata": {},
   "source": [
    "The SageMaker Estimator object is a high-level interface for SageMaker training. This object represents the algorithm, the data, and other configurations. \n",
    "\n",
    "If you are interested, you can read more about Sagemaker estimators [here](https://sagemaker.readthedocs.io/en/stable/api/training/estimators.html)"
   ]
  },
```

```
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "3989d745",
   "metadata": {},
   "outputs": [],
   "source": [
    "# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Modify this based on your script
name !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n",
    "entry_point = \"regressionScript.py\"\n",
    "\n",
    "# Modify this based on your instance type / size\n",
    "train_instance_type = \"ml.m5.large\"\n",
    "\n",
    "estimator_parameters = {\n",
    "    \"entry_point\": entry_point,\n",
    "    \"source_dir\": \"script\",\n",
    "    \"framework_version\": \"0.23-1\",\n",
    "    \"py_version\": \"py3\",\n",
    "    \"instance_type\": train_instance_type,\n",
    "    \"instance_count\": 1,\n",
    "    \"hyperparameters\": hyperparameters,\n",
    "    \"role\": role,\n",
    "    \"base_job_name\": \"linearregression-model\",\n",
    "}\n",
    "\n",
    "estimator = SKLearn(**estimator_parameters)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "a3d5035f",
   "metadata": {},
   "source": [
    "### 5) Running the Training Job"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2063774d",
   "metadata": {},
   "source": [
    "When we call 'fit' SageMaker will spin up managed containers, transfer
the code and data to the container and then start the training.  All this
happens off of the notebook server. We can watch the training through the
console, and watch the logs in CloudWatch Logs."
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "4b9190be",
   "metadata": {},
   "outputs": [],
   "source": [
    "inputs = {\n",
    "    \"train\": pickle_train_s3_uri\n",
    "}\n",
    "\n",
    "# starting the training job\n",
    "estimator.fit(inputs)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "f411616f",
   "metadata": {},
   "source": [
    "### 6) Deploying the Model to an Endpoint"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "e34a55be",
   "metadata": {},
   "source": [
    "An endpoint serves as a hub that runs the model. We can connect to an endpoint and obtain predictions. We can create a 'predictor' by deploying the estimator.  Then we can use it to make new predictions.\n",
    "\n",
    "(Make sure that the 'endpoint_name' used is not currently running.)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "ed1b7e25",
   "metadata": {},
   "outputs": [],
   "source": [
    "sklearn_predictor = estimator.deploy(initial_instance_count=1,\n",
    "                                     instance_type='ml.m5.large',\n",
```

```
    "
endpoint_name='linearregression-endpoint')"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "e637516c",
   "metadata": {},
   "outputs": [],
   "source": [
    "sklearn_predictor.predict([[0],[1],[2],[3]])"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "ec0a73ee",
   "metadata": {},
   "source": [
    "### 7) Deleting The Endpoint"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "a79fc5a4",
   "metadata": {},
   "source": [
    "Running this cell will remove the endpoint and configuration:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "1b1ed7cb",
   "metadata": {},
   "outputs": [],
   "source": [
    "sklearn_predictor.delete_endpoint(True)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "9f3d5729",
   "metadata": {},
   "source": [
    "## Step 3.0 Image Recognition using SageMaker Notebook\n",
```

```
    "\n",
    "Now you are able to simply use the SageMaker Notebook to run and deploy any machine learning application. For simplicity, you can consider the image recognition program that you used in Assignment 3, and apply steps 2 and 3 to develop a training script and a jupyter notebook for running and deploying this program in the cloud.\n",
    "\n",
    "The general steps you should follow are:\n",
    "1) Download CIFAR 10 Data to Sagemaker. \n",
    "\n",
    "<br/>\n",
    "\n",
    "2) Suggested File Structure:\n",
    "\n",
    "    ``` bash\n",
    "\n",
    "    --potentialTrainingDataFile-- (file)\n",
    "    cnnDeployment.ipynb (file)\n",
    "\n",
    "    script (folder)\n",
    "        cnnScript.py \n",
    "\n",
    "    cifar-10-batches-py (folder)\n",
    "        data_batch_1 (file)\n",
    "        data_batch_2 (file)\n",
    "        ...\n",
    "        test_batch   (file)\n",
    "    ```\n",
    "\n",
    "\n",
    "<br/>\n",
    "\n",
    "3) Script Setup: Utilize the script you created in Step 2.2 and make modification to it based on the new model. \n",
    "\n",
    "<br/>\n",
    "\n",
    "4) Notebook Setup: Utilize the notebook script from Step 2.3 and make modification to it based on your new model. "
   ]
  },
  {
   "cell_type": "markdown",
   "id": "255dd1fd",
   "metadata": {},
   "source": [
```

```
    "### Helper Functions\n",
    "\n",
    "Below is some helper code that you can utilize in your project. YOU DO
NOT HAVE TO USE THESE FUNCTIONS. But these are here to hopefully make your
life easier. \n",
    "\n",
    "Even if you utilize these helper functions there will still be
additional code that needs to be written.\n",
    "\n",
    "``` python\n",
    "\n",
    "\n",
    "# CODE HELPER 1 - notice how data is saved -> training.cnn format\n",
    "def pickleTrainingData():\n",
    "    def unpickle(file):\n",
    "        with open(file, 'rb') as fo:\n",
    "            dict = pickle.load(fo, encoding='bytes')\n",
    "        return dict\n",
    "\n",
    "    train_data = np.empty((0, 32*32*3))\n",
    "    train_labels = []\n",
    "\n",
    "    for i in range(1, 2):\n",
    "        fileNameDataBatch = './cifar-10-batches-py/data_batch_' +
str(i)\n",
    "        batch = unpickle(fileNameDataBatch)\n",
    "        train_data = np.vstack((train_data, batch[b'data']))\n",
    "        train_labels += batch[b'labels']\n",
    "\n",
    "    train_labels = np.array(train_labels)\n",
    "    train_data = train_data.reshape(-1, 32, 32, 3) / 255.0\n",
    "    \n",
    "    # !!!!! NOTICE HOW THE DATA IS SAVED !!!!!!\n",
    "    # Will be returned in form of:\n",
    "    # train_label, train_data  = getDataBack()\n",
    "    pickle.dump([train_labels,train_data], open('./train.cnn',
'wb'))\n",
    "\n",
    "pickleTrainingData()\n",
    "\n",
    "# CODE HELPER 2\n",
    "def getTestData():\n",
    "    def unpickle(file):\n",
    "        with open(file, 'rb') as fo:\n",
    "            dict = pickle.load(fo, encoding='bytes')\n",
    "        return dict\n",
```

```
    "    fileNameTestBatch = './cifar-10-batches-py/test_batch'\n",
    "    test_data = unpickle(fileNameTestBatch)[b'data']\n",
    "    test_data = test_data.reshape(-1, 32, 32, 3) / 255.0\n",
    "    test_labels = np.array(unpickle(fileNameTestBatch)[b'labels'])\n",
    "    \n",
    "    num_samples_to_select = 600\n",
    "    random_indices = np.random.choice(test_data.shape[0],
num_samples_to_select, replace=False)\n",
    "    selected_test_data = test_data[random_indices]\n",
    "    selected_test_labels = test_labels[random_indices]\n",
    "    \n",
    "    return selected_test_data, selected_test_labels\n",
    "\n",
    "test_data, test_labels = getTestData()\n",
    "\n",
    "# CODE HELPER 3\n",
    "from sklearn.metrics import accuracy_score\n",
    "def getAccuracyOfPrediction(cnn_predictions, test_labels):\n",
    "    cnn_predicted_labels = np.argmax(cnn_predictions, axis=1)\n",
    "    accuracy = accuracy_score(test_labels, cnn_predicted_labels)\n",
    "    print(\"Accuracy:\", accuracy)\n",
    "\n",
    "\n",
    "# CODE HELPER 4\n",
    "import argparse\n",
    "import os\n",
    "import pickle\n",
    "import subprocess\n",
    "subprocess.run([\"pip\", \"install\", \"Werkzeug==2.0.3\"])\n",
    "subprocess.run([\"pip\", \"install\", \"tensorflow==2.4\"])\n",
    "import tensorflow as tf\n",
    "from tensorflow import keras\n",
    "\n",
    "# CODE HELPER 5 -> notice that the file name ends in .cnn\n",
    "files = [os.path.join(path, file) for file in os.listdir(path) if
file.endswith(\".cnn\")]\n",
    "\n",
    "# CODE HELPER 6\n",
    "loaded_model = tf.keras.models.load_model(os.path.join(model_dir,
\"modelCNN\"))\n",
    "\n",
    "# CODE HELPER 7\n",
    "model = keras.Sequential([\n",
    "    keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',
input_shape=(32, 32, 3)),\n",
    "    keras.layers.MaxPooling2D(pool_size=2),\n",
```

```
    "    keras.layers.Conv2D(filters=64, kernel_size=3,
activation='relu'),\n",
    "    keras.layers.MaxPooling2D(pool_size=2),\n",
    "    keras.layers.Flatten(),\n",
    "    keras.layers.Dense(units=128, activation='relu'),\n",
    "    keras.layers.Dense(units=10, activation='softmax')\n",
    "])\n",
    "model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])\n",
    "\n",
    "model.fit(train_data, train_labels, epochs=3, batch_size=32,
validation_split=0.1)\n",
    "\n",
    "model.save(os.path.join(args.model_dir, \"modelCNN\"))\n",
    "\n",
    "```"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "d9238132",
   "metadata": {},
   "source": [
    "## Deliverables:\n",
    "\n",
    "Upon completing this assignment, you need to submit the following:\n",
    "\n",
    "For the image recognition program:\n",
    "- Training scripts and notebooks\n",
    "- Accuracy result of testing your image recognition program.  \n",
    "- The total cost of running the program\n",
    "- The time taken to train, deploy and predict with sagemaker"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "07e57431",
   "metadata": {},
   "source": [
    "# Good Luck!"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
```

```
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.10.13"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```