Contents

[Instructions for Participants to run lab sessions on AWS Immersion Day 2](#_Toc58413168)

[General Instructions to Run the Script 5](#_Toc58413169)

[Specific and detailed instructions to run the labs 7](#_Toc58413170)

[Session 1: Data Preparation and Exploratory Data Analysis 8](#_Toc58413171)

[Session 2 : Feature Engineering 18](#_Toc58413172)

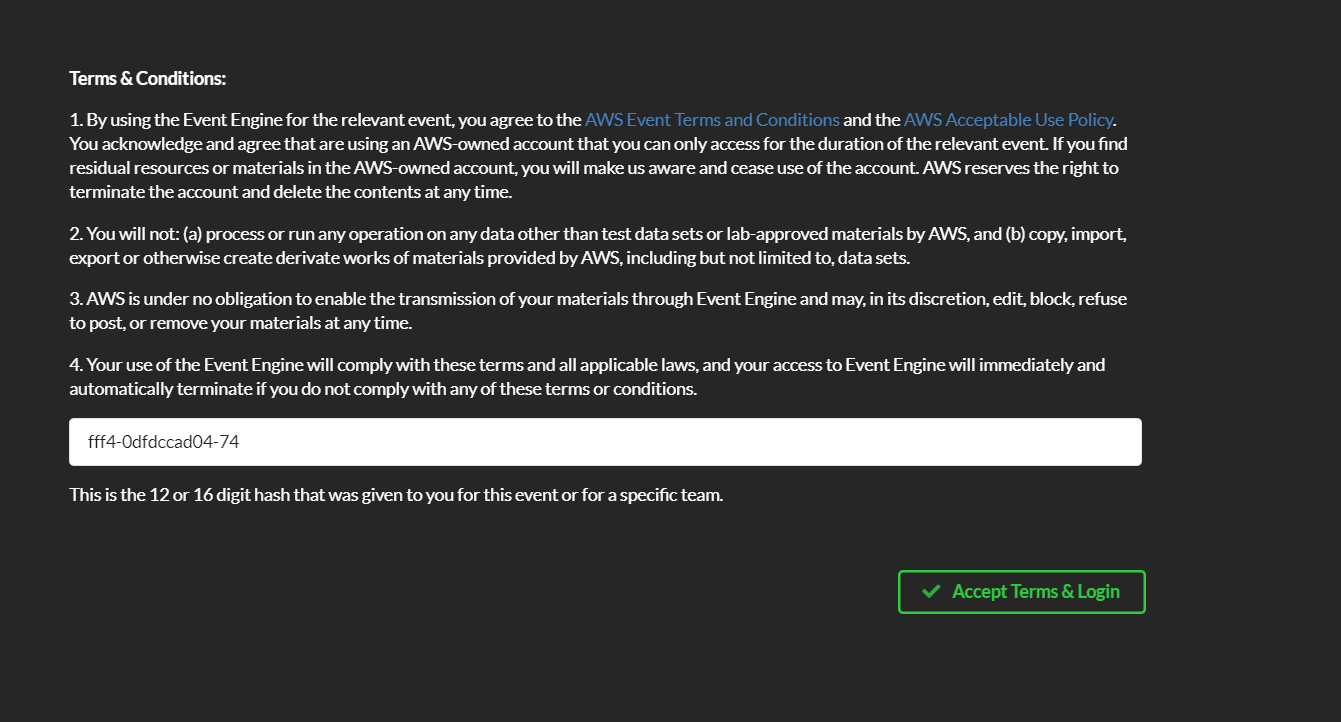
[Session 3 : Developing a benchmark model 22](#_Toc58413173)

[Session 4 : Operationalization and Evaluation 25](#_Toc58413174)

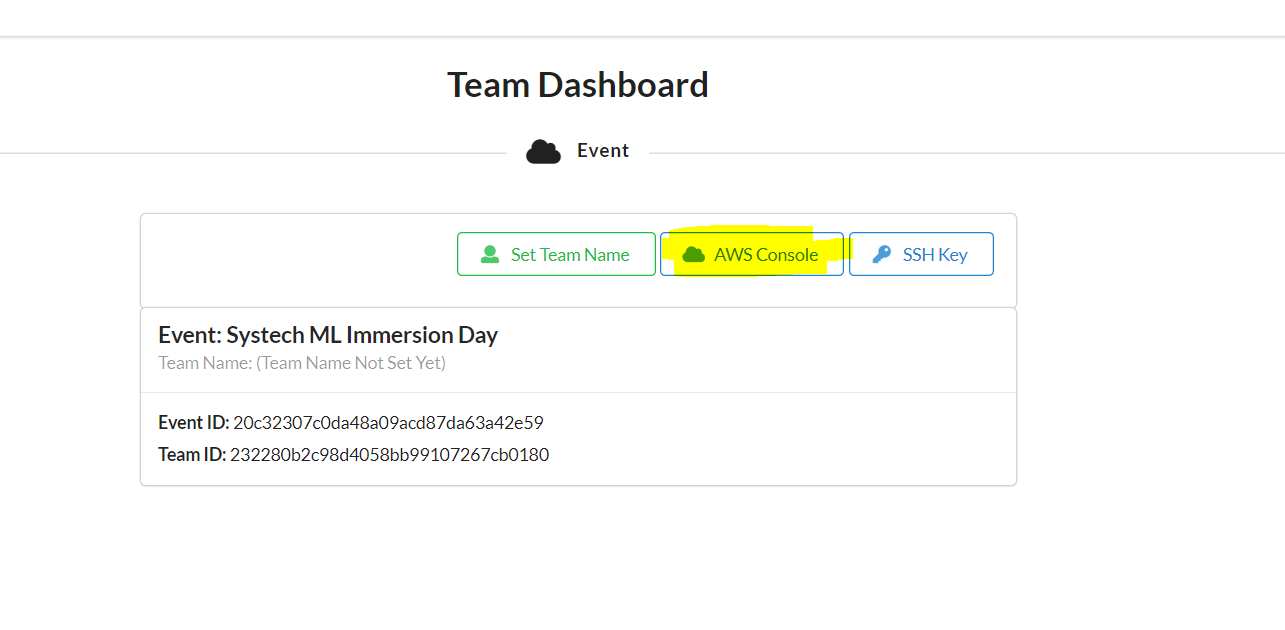
[Instructions in case of errors 28](#_Toc58413175)

# Instructions for Participants to run lab sessions on AWS Immersion Day

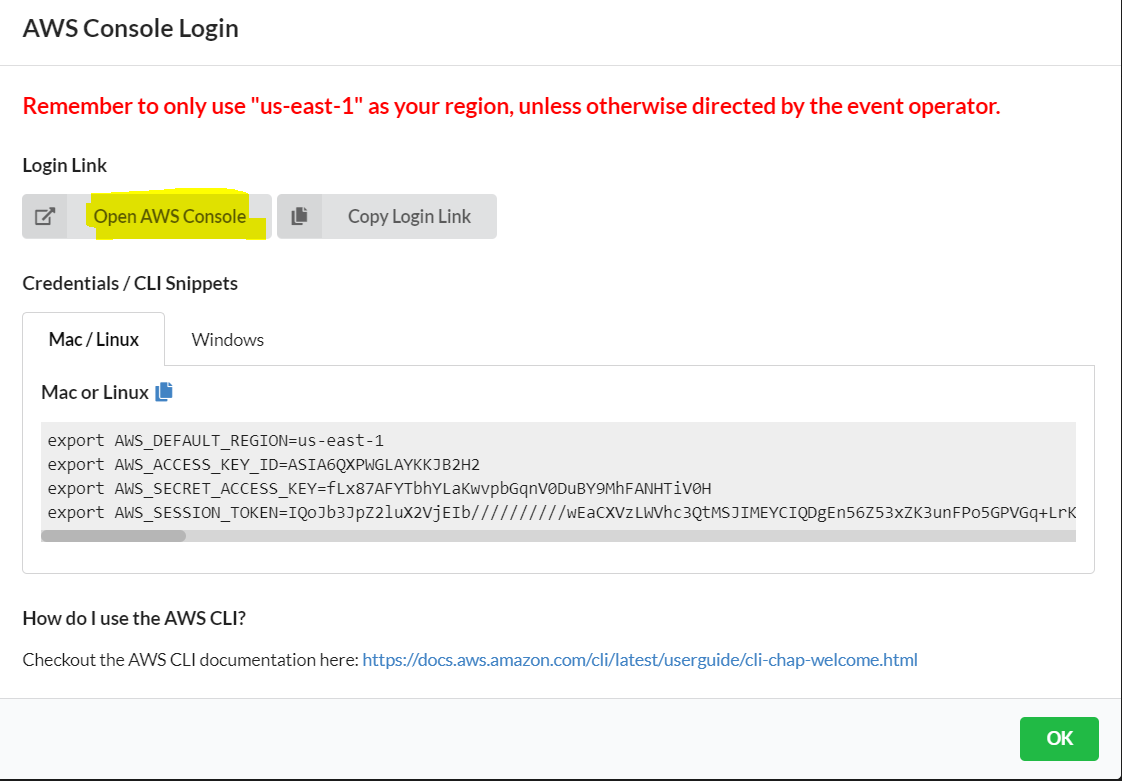
1. Login to AWS account using the link shared by the coordinators
2. Once you click on the link, it should direct you to the below screen, click on **Accept Terms & Login**



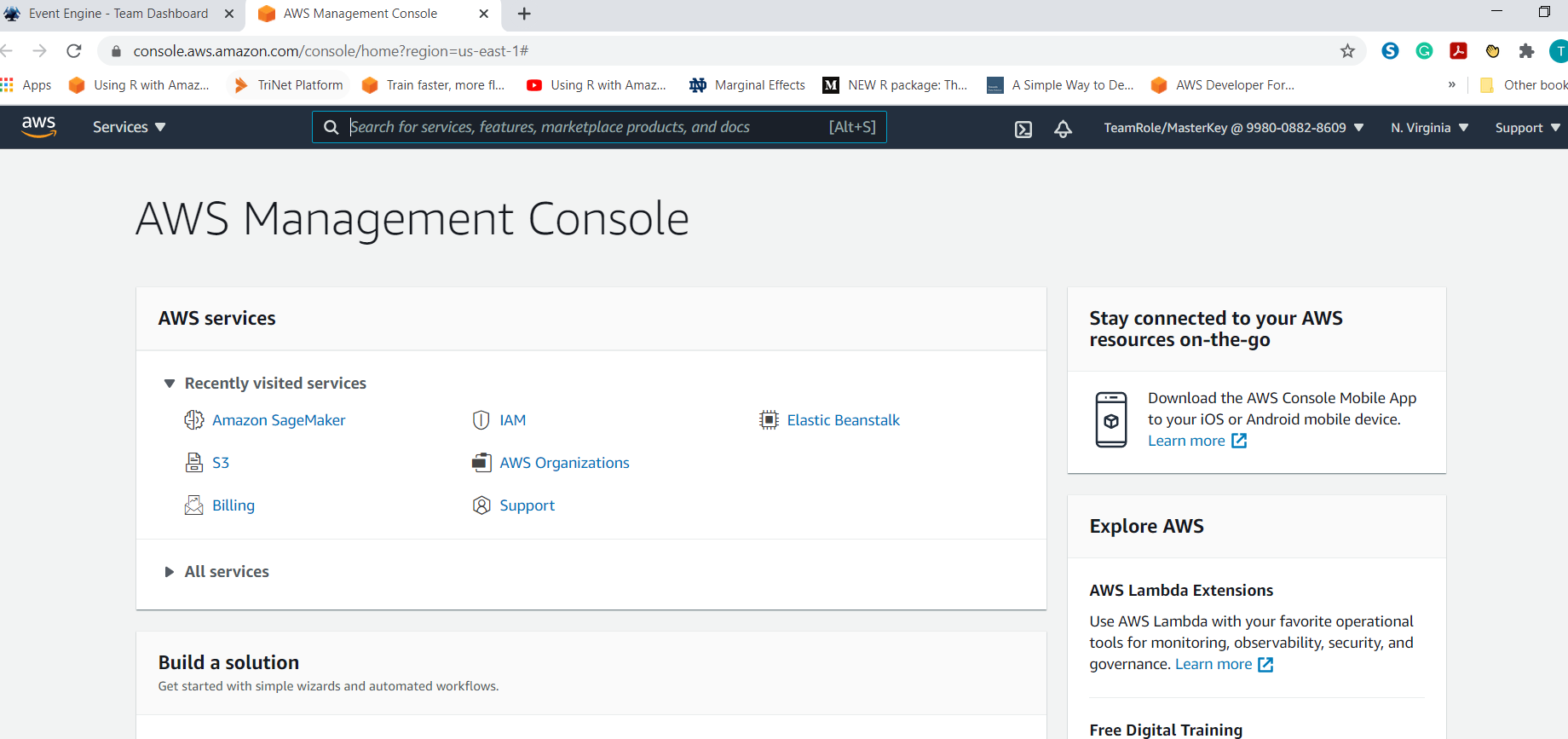
1. You should be redirected to another screen a shown below, click on **AWS console** as highlighted below



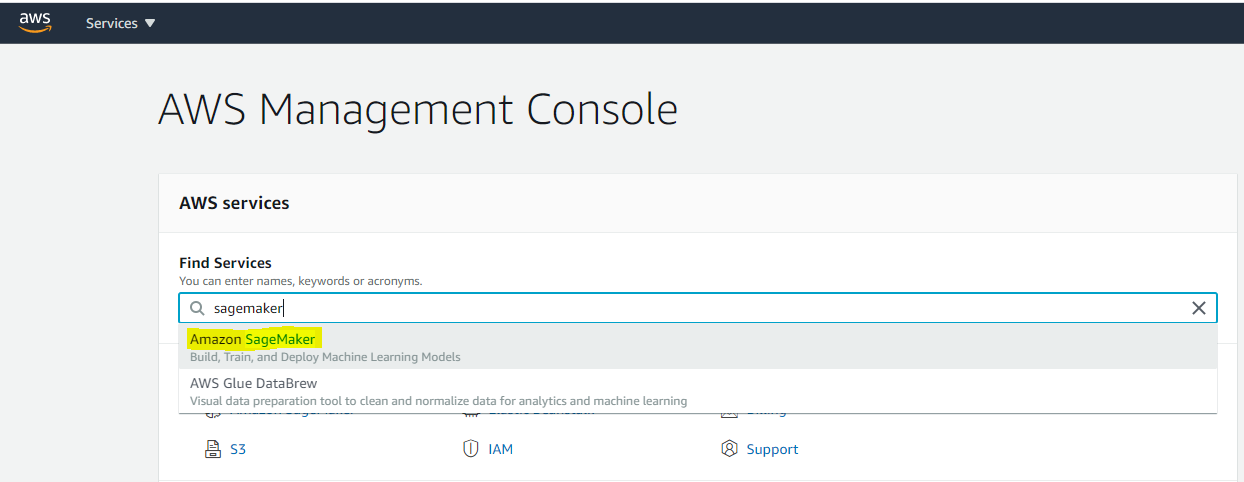
1. You should be redirected to another screen as shown below, click on **Open AWS Console** as highlighted below



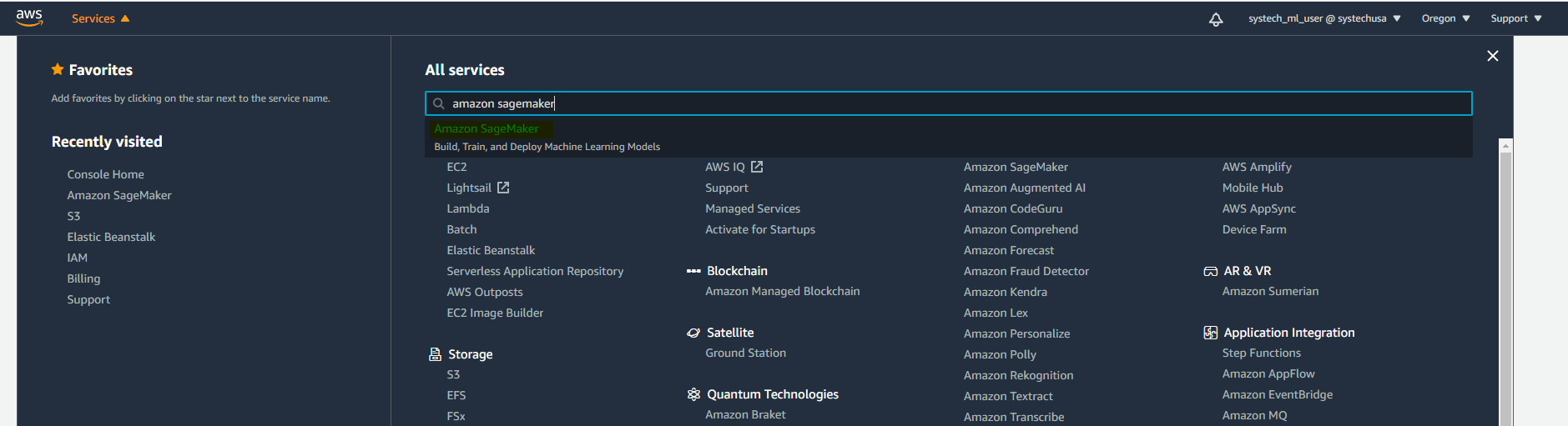
1. **You should be redirected to the AWS console in another tab as shown below**



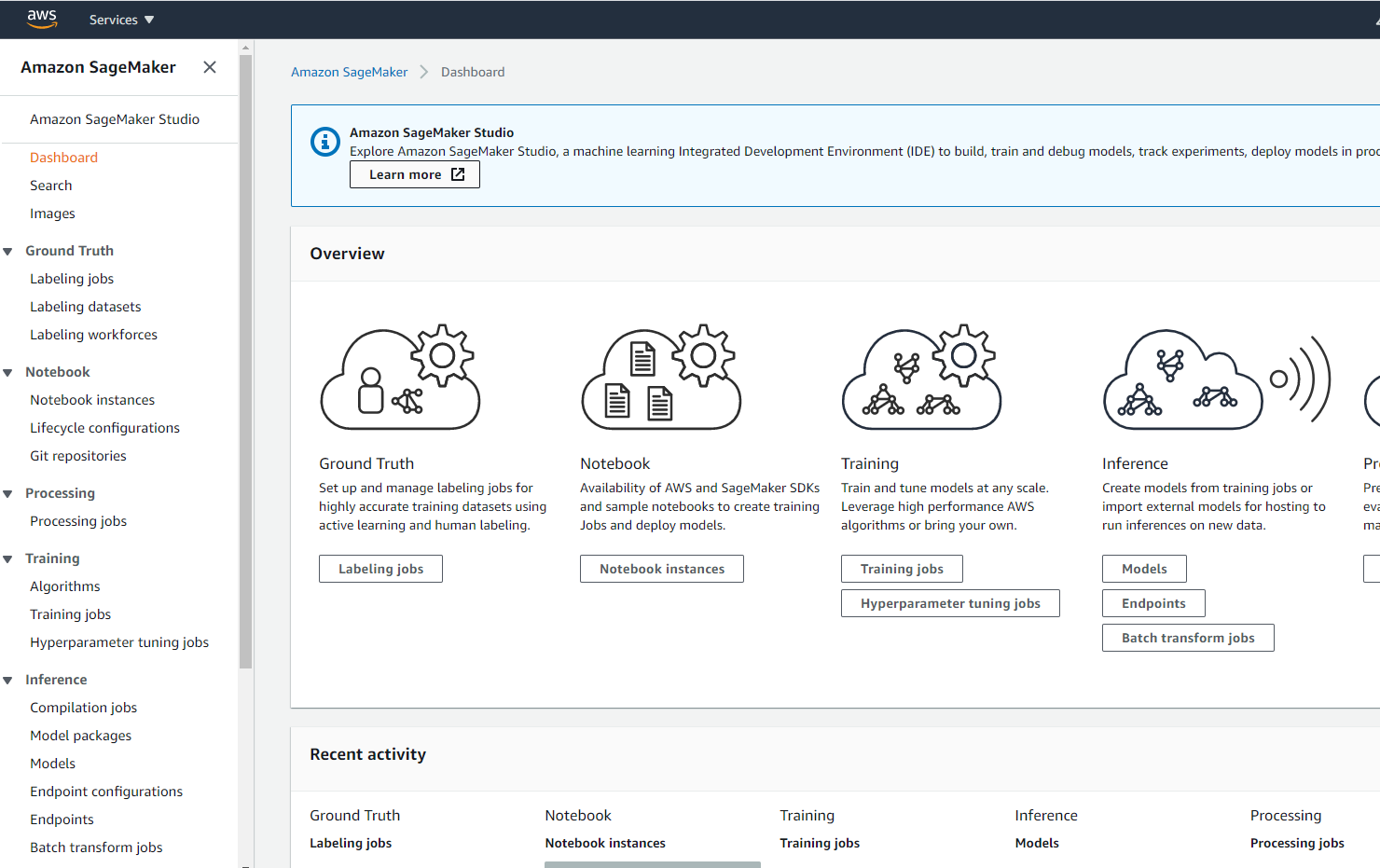
1. Open and access the SageMaker console by searching Amazon SageMaker in the **Find Services** search bar



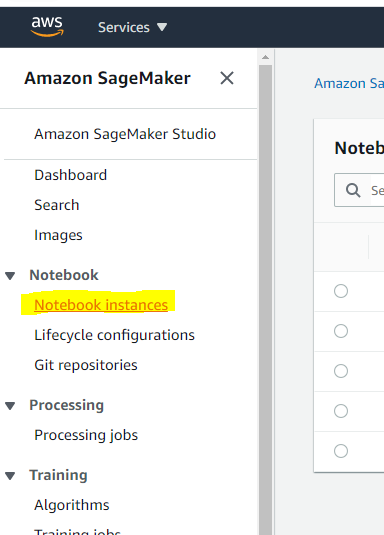
Or click on Services on your top left corner and the search Amazon SageMaker in the **All services search** bar

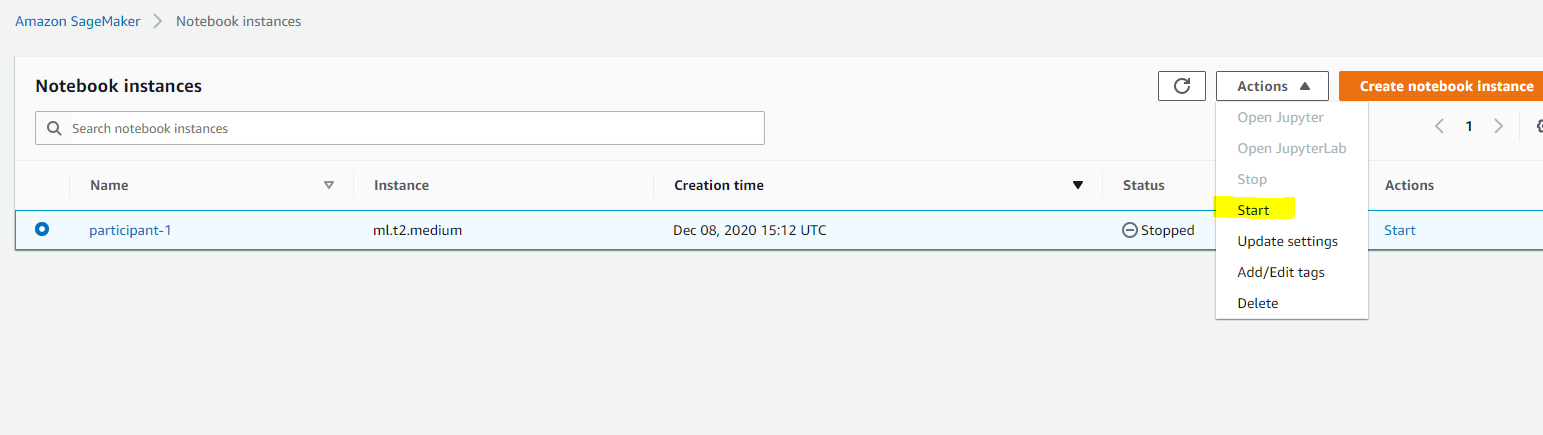


1. You should be able to see the SageMaker console as shown below

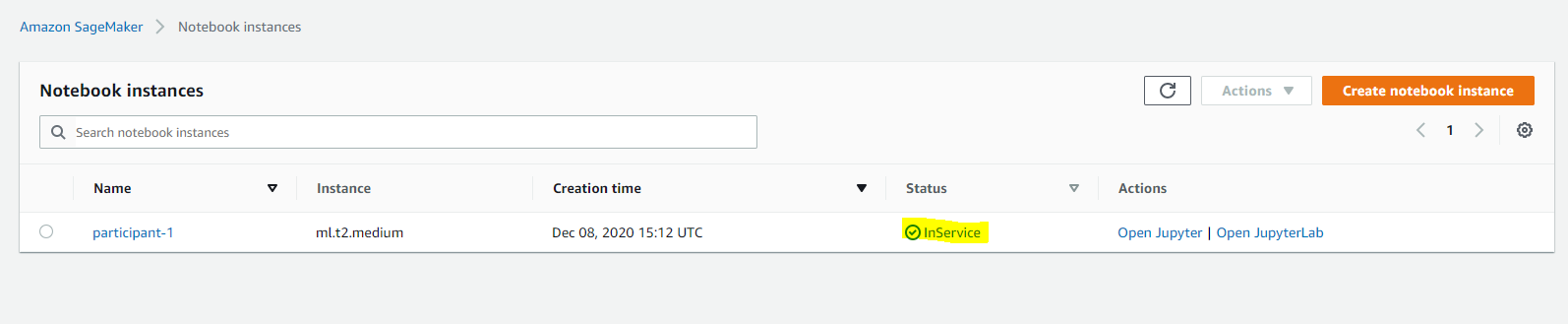


1. In the Amazon SageMaker console, Choose **Notebook instances**, then against the notebook instance assigned to each participant, click on **Start** under **Actions**

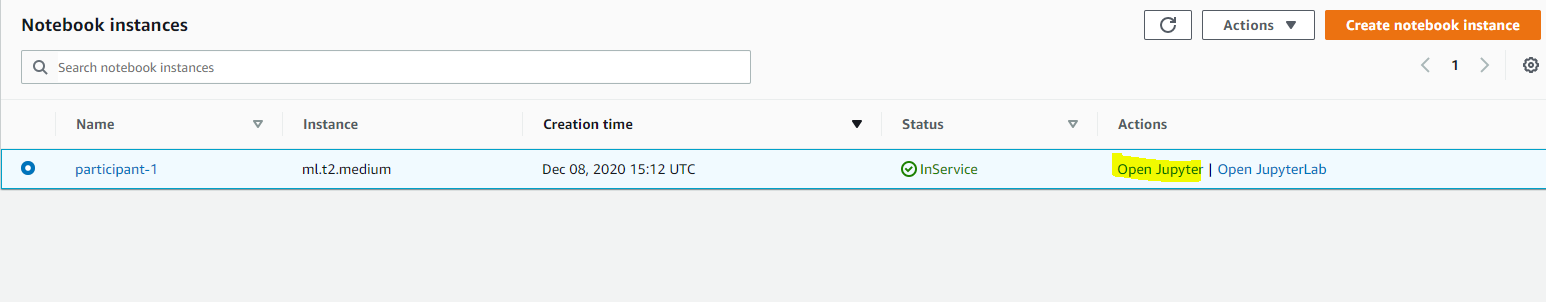




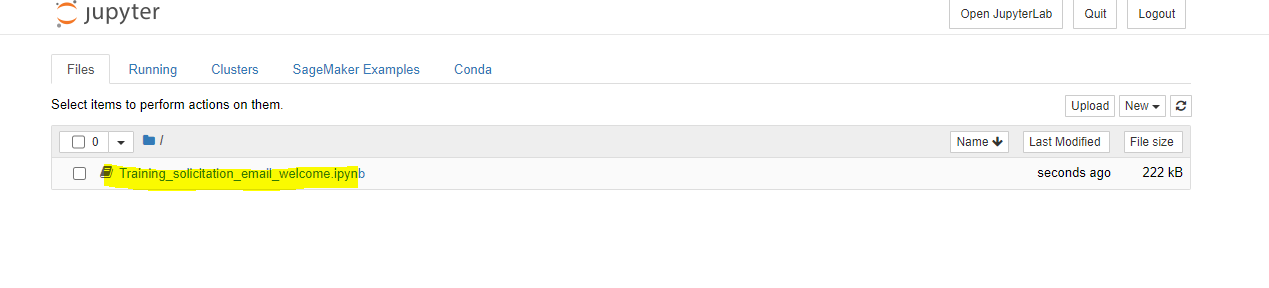
1. The Status of your assigned notebook should change to **InService.** This may take a few minutes.



1. Click on **Open Jupyter** to access the notebook instance console and further execute the script



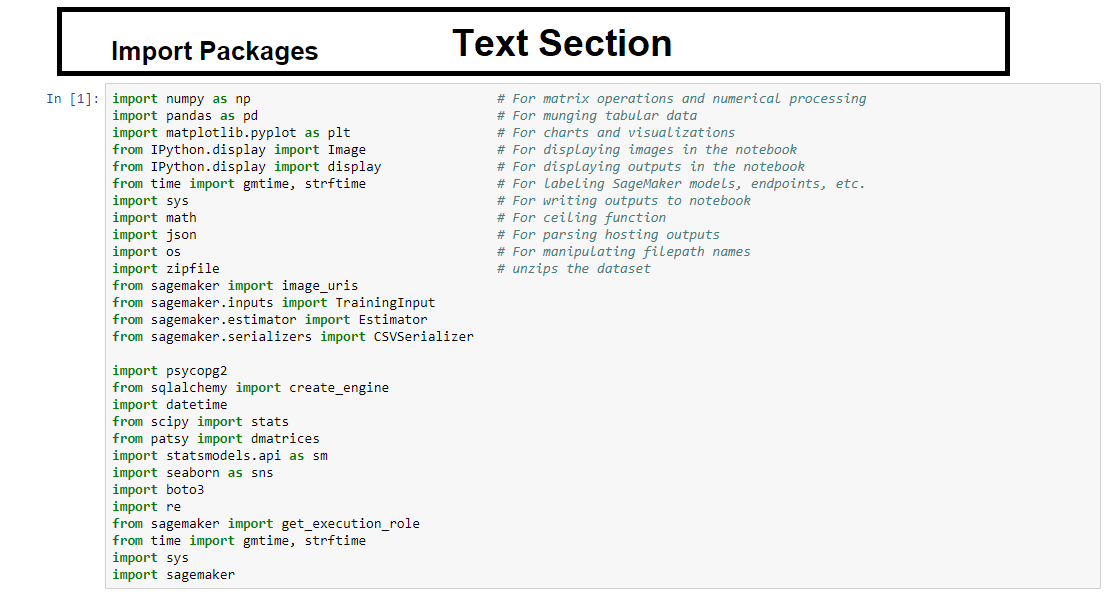
1. Inside the Jupyter console, you should be able to see a script by the name **Training\_solicitation\_email\_welcome.ipynb.** Click on it and you will be redirected to the script console



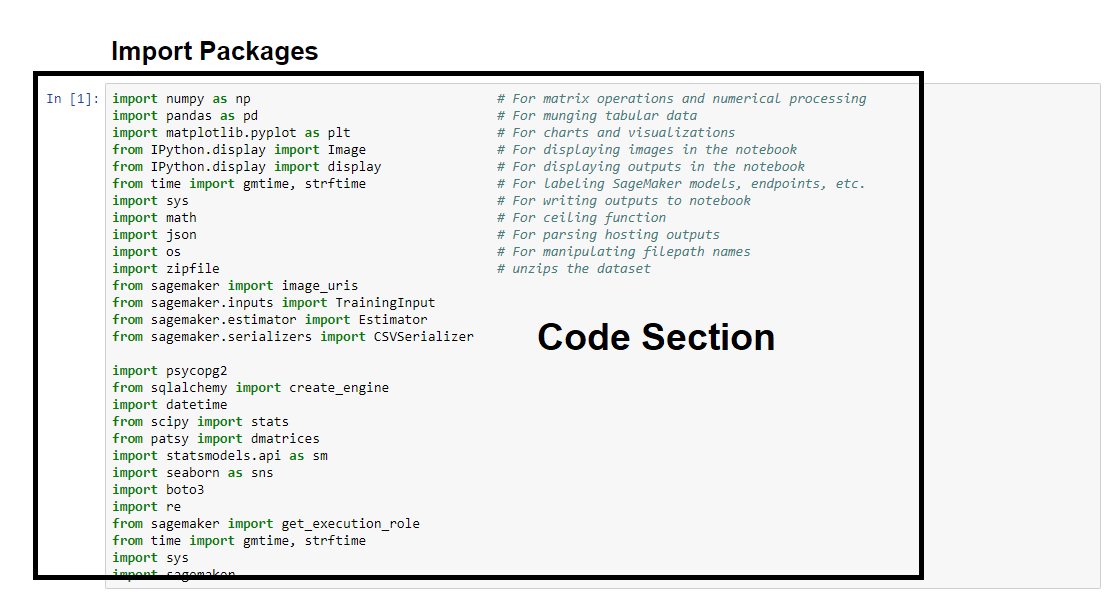
# General Instructions to Run the Script

You should be able to see multiple code cells arranged in a sequential order as soon as you open the script. There should two types of cells visible to you

**Text Cell** (The one with Text): There are just instructions that have been added to the script to explain certain parts using Text (this does not execute anything else)



**Code Cell** (The one with line of Code): These are code snippets that execute a particular action needed for the machine learning lab

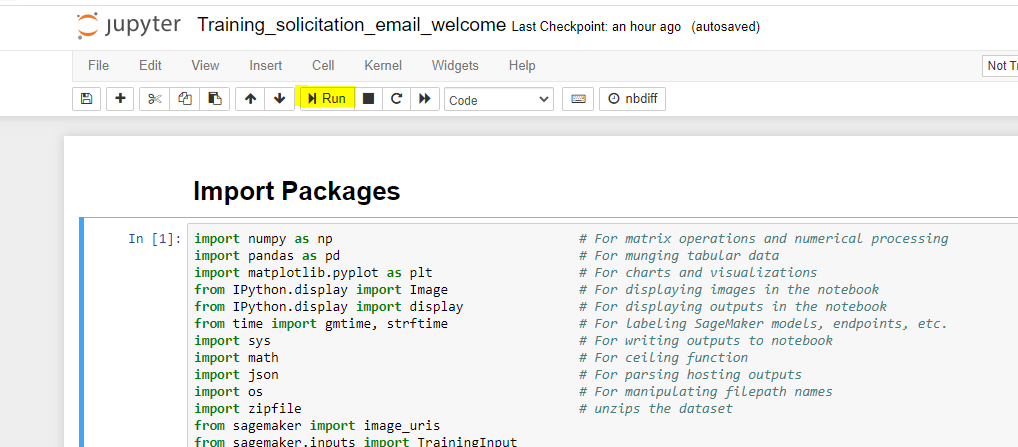


* For simplicity purposes, we will be referring to the two type of cells as **Text Cells** and **Code Cells**
* The Instructor will go through each cell explaining what it does and then execute it in a sequential manner.
* The participants can execute each cell along with the instructor and follow along with the machine learning process or come back later and execute.

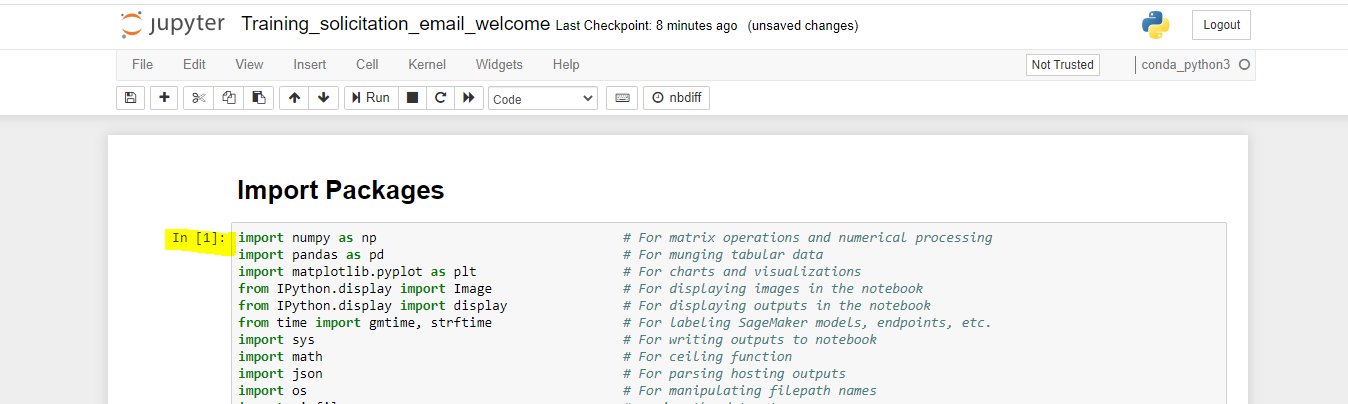
Important Note:

**Please run each code cell in order, and only once, to avoid repeated operations. For example, running the same training job cell twice might create two training jobs, possibly exceeding your service limits.**

* Inside the script, to run a code cell, simply click on it, then either click the **Run** Cell button in the notebook’s toolbar or use **Control+Enter** from your computer’s keyboard.



* When you run a cell, you will notice that there are some square braces **In []**: in the left margin next to each cell



* The square braces will auto fill with a number that indicates the order that you ran the cells. For example, if you open a fresh Notebook and run the first cell at the top of the Notebook, the square braces will fill with the number 1.
* It may take a few seconds to a few minutes for a code cell to run. You can determine whether a cell is running by examining the **In []:** indicator: a cell will show In **[\*]:** when running, and **In [a number]:** when complete.

# Specific and detailed instructions to run the labs

This is the script that we will be executing as part of our lab session divided into 4 sections :

Session 1- Data Preparation and Exploratory Data Analysis

Session 2 - Feature Engineering

Session 3 - Developing a Benchmark model

Session 4 - Operationalization and Evaluation

1. The first text cell contains text explaining that the code snippet followed by a code cell which imports packages required to run data and machine learning operations for this lab purpose.

(moving forward instructions would just explain what each **code cell** does in the lab. No specific instructions have been mentioned to run the text cells. Those would have to be run sequentially along with the code cells )

Run the code cell to import the packages.

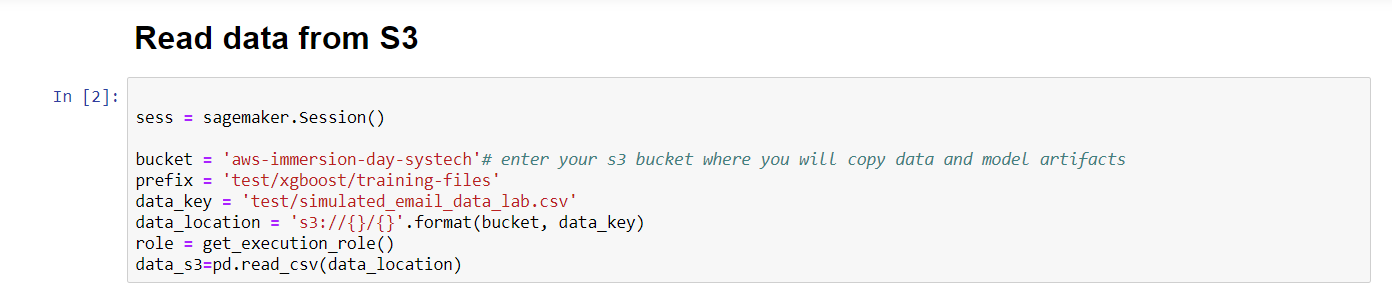


1. The next cell will read the dummy data into this SageMaker Instance from S3 (location where the dummy data is stored) for us to run the labs on it.

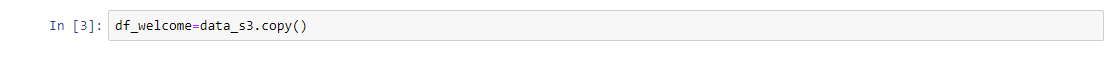
The following things are being setup in this code cell

* The name of the S3 bucket where the dummy data is stored
* The prefix(folders) that you want to use for training and model data.
* The IAM role arn used to give training and hosting access to your data.

Run the below cell to read the data from S3



1. In the next cell, we simply copy the dataset that is stored as data\_s3 into another dataset named “df\_welcome”

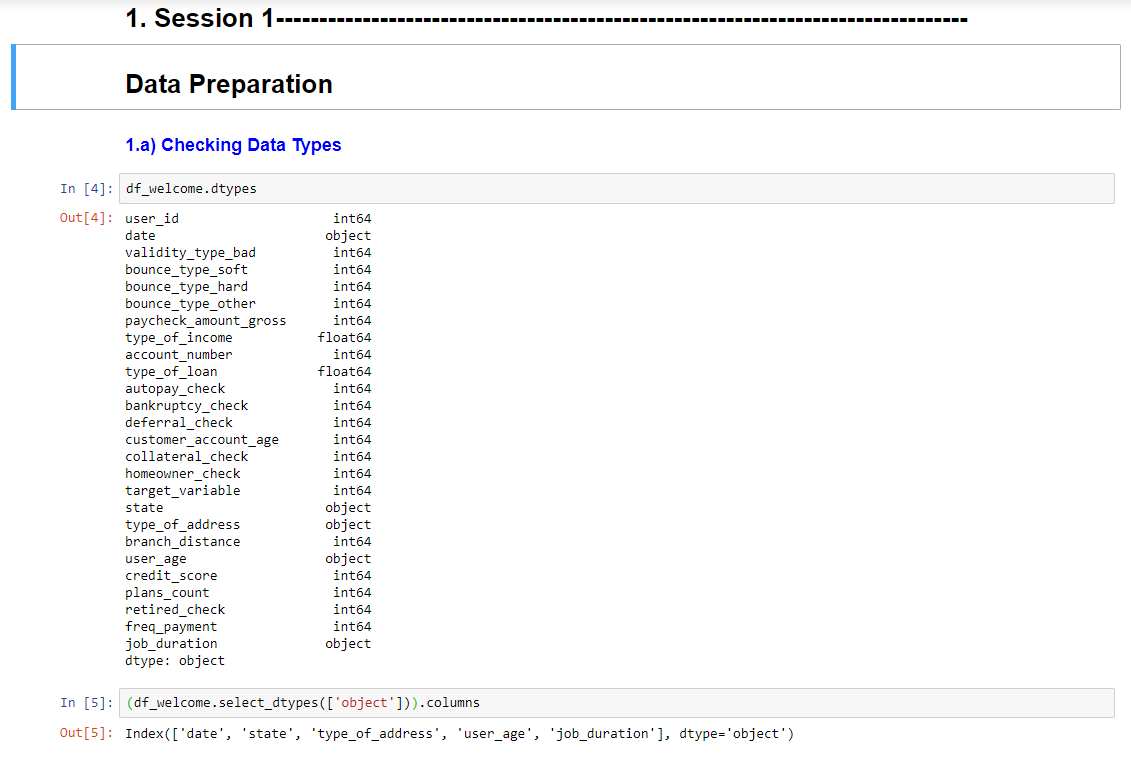


## Session 1: Data Preparation and Exploratory Data Analysis

1. In the next coming cells, we will work on the First Lab Session starting with Data Preparation where we will check the datatypes of the columns that are part of the dataset (which we just loaded in the previous steps).

* Python assigns a column an Object datatype if it is not able to understand what kind of data it is.
* Our goal is to identify such columns and assign the right data type to it.

Run the below cells to check the datatype and then specifically point out columns with datatype as object.



We can observe that date and user\_age columns are object types, let’s analyze them further.

1. In the next two code cells

* We used .head() function to check the first 5 records for date and observe that there are multiple date formats.
* We will fix them by changing the data type to Datetime using astype function. The new values should change to the same format as seen in the screenshot below



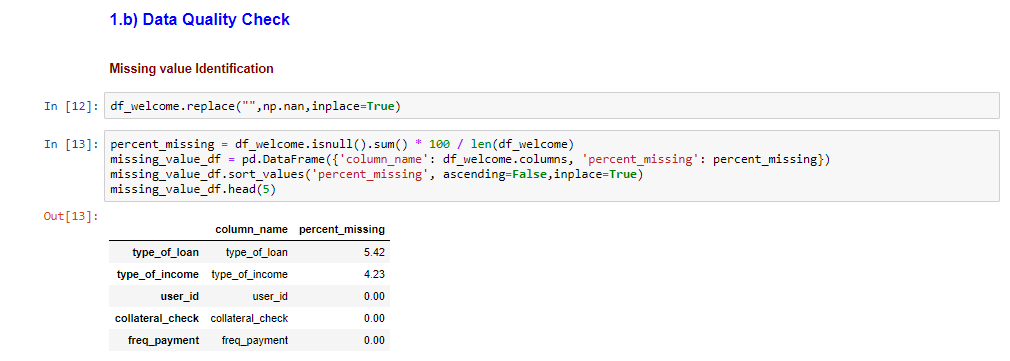
1. In the next 3 code cells

* We used .head() function to check the first 5 records for user age and observe that there is a string ‘years’ after every entry, we want to use this column as an integer or numeric so that we can analyze it further and will have to remove ‘years’ and convert it into a numeric data type.
* We will fix this column by splitting the string ‘years’ from all the age records and convert it to an integer type of datatype.
* Perform the same steps for job\_duration as it has the same problem.



1. In the next two code cells, we will do a quick quality check to identify the percentages of missing values in each column

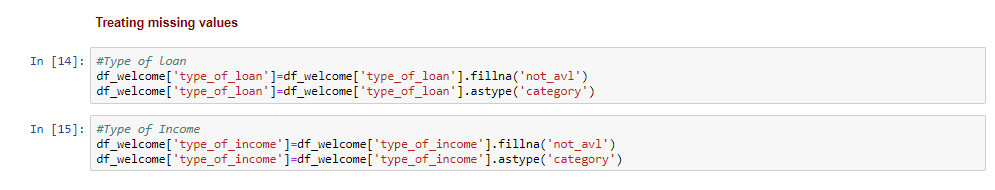
* In the first cell, we are converting empty values to nulls that python can understand.
* And then in the next cell, we are using the isnull() to calculate the percentage of missing values and checking the top 5



We can observe that type\_of\_loan and type\_of\_income have 4-5% of missing values.

1. In the next two code cells, we will simply add a new category ‘not\_avl’ instead of missing values in the two columns(type\_of\_loan and type\_of\_income)identifies in the previous steps

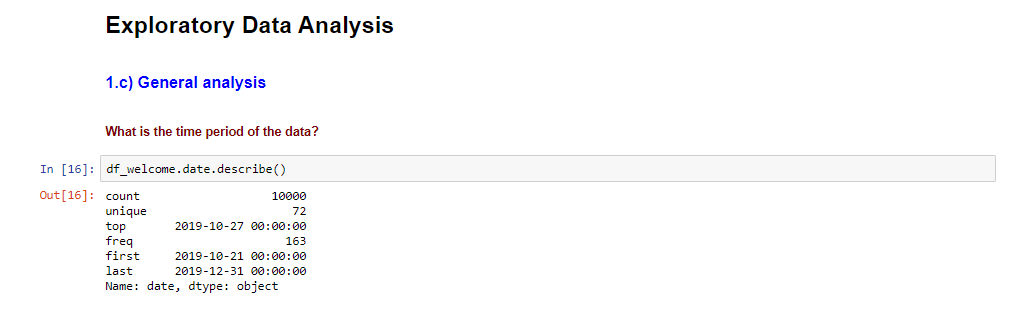
* Instead of sending empty values to the model, we have added a category for the model to identify the relationship and use it for inferencing.



1. Next is the Exploratory Data Analysis part of Session 1

* Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical
* We will explore the Data set and perform the exploratory data analysis starting with some general analysis, followed by univariate analysis sand then bivariate.

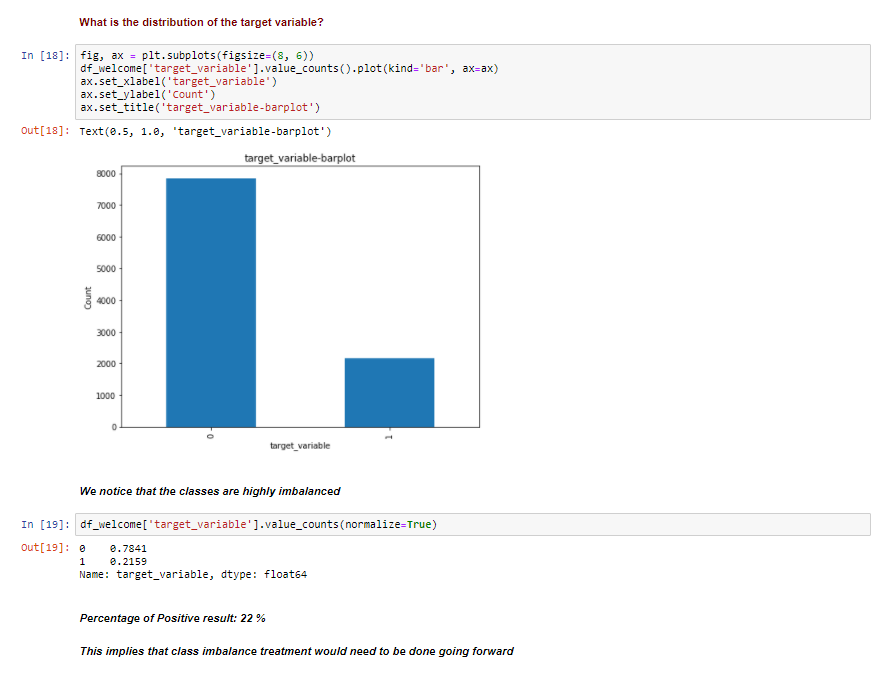
The next cell as shown below gives us the first and last date for the ‘date’ column indicating the range of data that we are dealing with



1. In the next cell, we are analyzing the target column

Which in this case has values 0 and 1

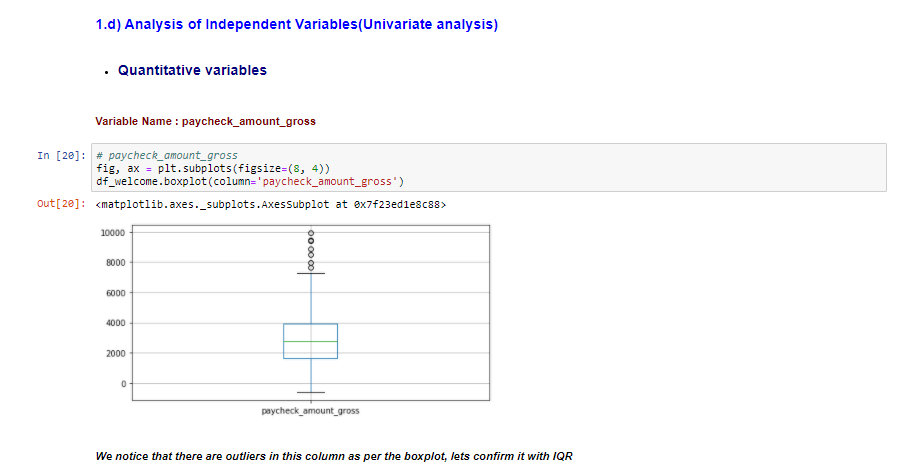
* 1 indicating that user started the application for which the welcome email was sent
* 0 indicating the user never started with the application
* In this use case, we are using machine learning using historical data to predict the propensity of a user to respond to email for a particular reason based on the success criteria mentioned above.
* The cell [26]as shown below outputs a bar plot indicating the distribution of positive class(0s) and negative class(1s)
* The next cells[27] gives us a percentage of the same distribution
* The observation is that Percentage of Positive result is 22 %
* Which implies that class imbalance treatment would need to be done going forward



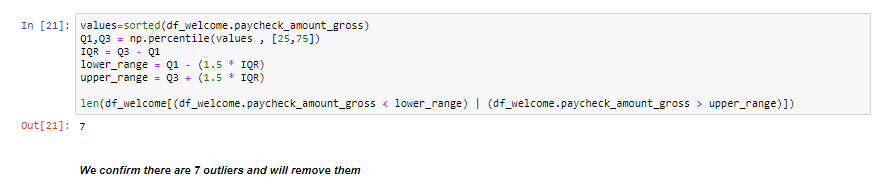
1. Now that we are done with some general analysis , let get into the uni-variate analysis part of the lab where we will pick up some columns and analyze them individually.

* We will start with the quantitative columns
* Quantitative. Quantitative columns are numerical. They represent a measurable quantity. For example, when we speak of the population of a city, we are talking about the number of people in the city - a measurable attribute of the city. Therefore, population would be a quantitative column.
* In the next code cell, we will use a Boxplot to analyze the column paycheck\_amoaunt\_gross
* Boxplots are a standardized way of displaying the distribution of data based on a five-number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”).
* A boxplot, also called a box and whisker plot, is a way to show the spread and centers of a data set. Measures of spread include the interquartile range and the mean of the data set. Measures of center include the mean or average and median (the middle of a data set).
* Data sets can sometimes contain outliers that are suspected to be anomalies (perhaps because of data collection errors or just plain old flukes). If outliers are present, you will observe data points above or below the end horizontal lines

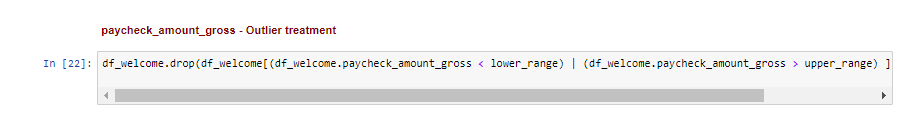
Execute the below code cell to display the box plot where you will observe outliers



1. In the next cell, we are simply checking outliers using a statical formula and can confirm that there are 7 outliers



1. In the next cell, we will remove the outliers that we detected in the previous step

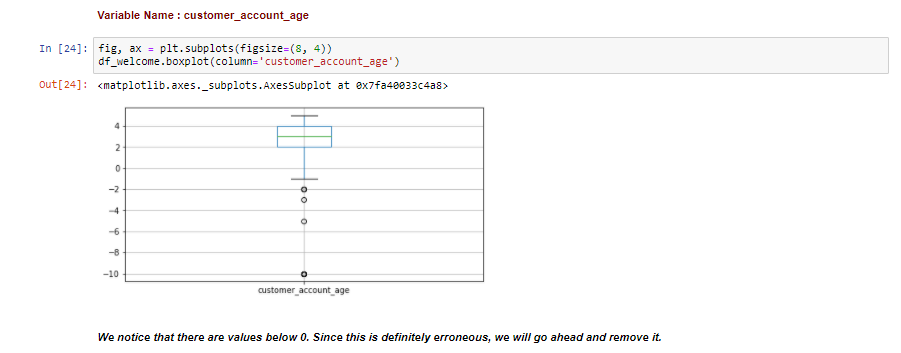


1. We observed in the boxplot that there are paycheck amounts less than 0 which is erroneous, and we will remove them using the next cell

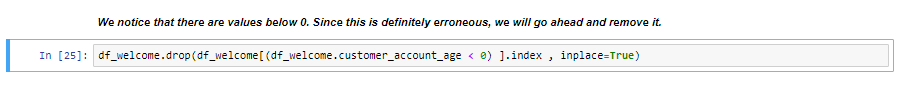


1. In the next cell we will analyze customer account age using a boxplot,

* We notice that there are values below 0. Since this is definitely erroneous, we will go ahead and remove it.

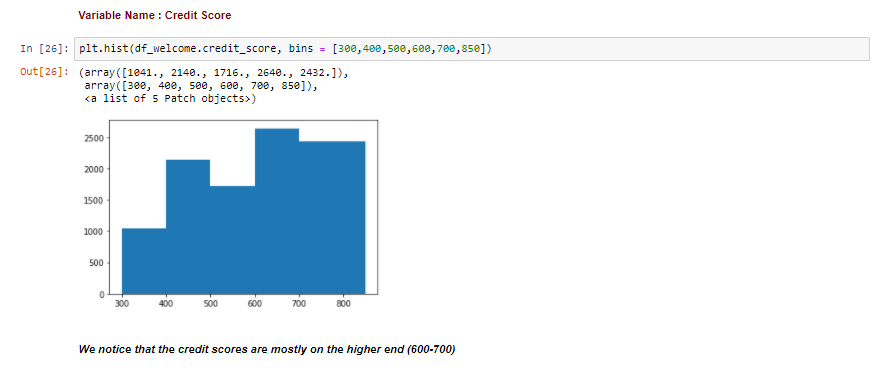


1. In the next cell, we will delete the records for which customer account age is less than 0 as identified in the previous step



1. In the next cell, we will analyze another numeric column **Credit score** using histogram

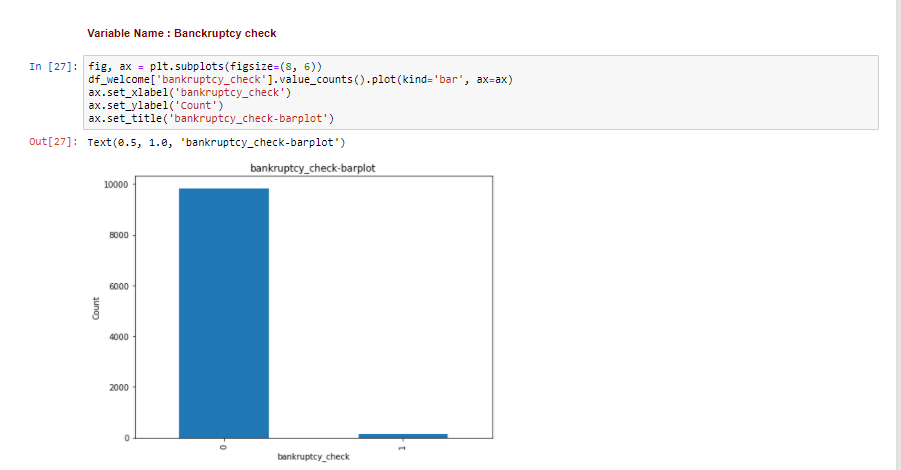
* A histogram is a plot that lets you discover, and show, the underlying frequency distribution (shape) of a set of continuous data. This allows the inspection of the data for its underlying distribution (e.g., normal distribution), outliers, skewness, etc.



1. In the next following cells, let analyze a few categorical columns

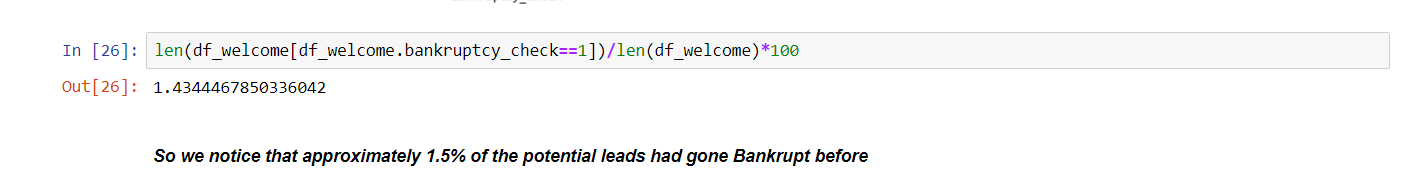
* Categorical. Categorical columns take on values that are names or labels. The color of a ball (e.g., red, green, blue) or the breed of a dog (e.g., collie, shepherd, terrier) would be examples of categorical columns.

Please run the next cell to analyze Bankruptcy check column which is a binary column (having values 0 and 1) using a barplot

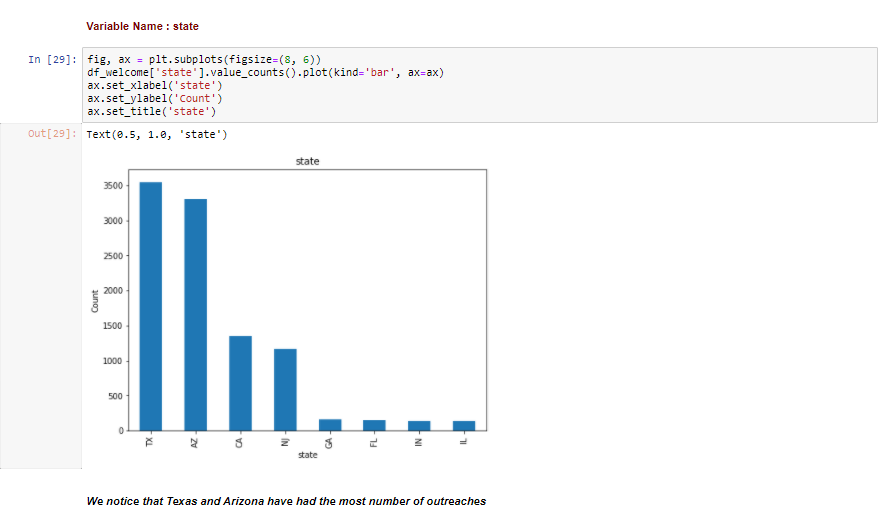


We can observe the difference in the count of 0s and 1s

1. In the next cell we can confirm the actual percentage of 1s in the same column



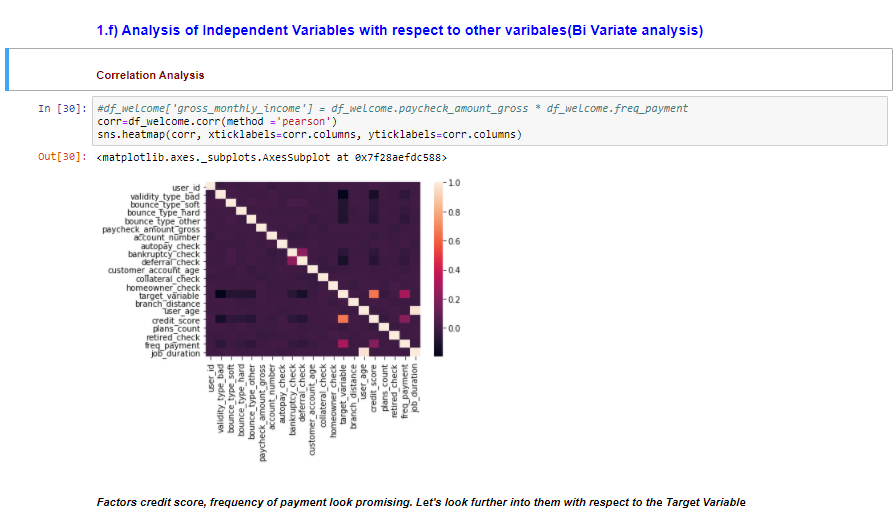
1. In the next cell, we will analyze state using a similar plot



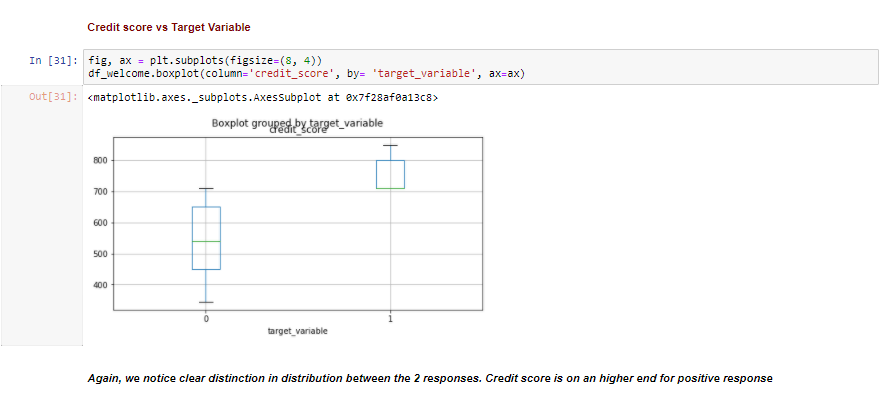
1. Now that we are done with individually analyzing some of the columns, let get into the bi-variate analysis part where we analysis combinations of columns

In the next cell, we will do a correlation analysis using Pearson coefficient

* **Pearson’s correlation coefficient** is the test statistics that measures the statistical relationship, or association, between two quantitative columns.  It is known as the best method of measuring the association between columns of interest because it is based on the method of covariance.  It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship.
* -1 indicates a perfectly negative linear correlation between two columns
* 0 indicates no linear correlation between two columns
* 1 indicates a perfectly positive linear correlation between two columns
* The diagonal of the table is always a set of ones, because the correlation between a column and itself is always 1
* Let’s find the target column in the vertical axis and then against it check the color of the rest of the column in the horizontal axis

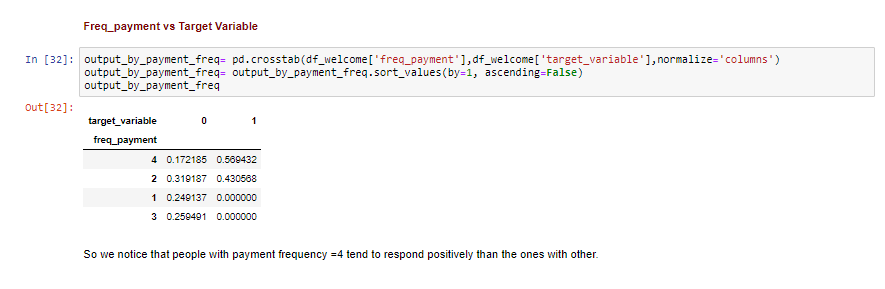


1. In the next code cell, we will analyze credit score against the target column using boxplot again, where **we notice clear distinction in distribution between the 2 responses. Credit score is on a higher end for positive response**

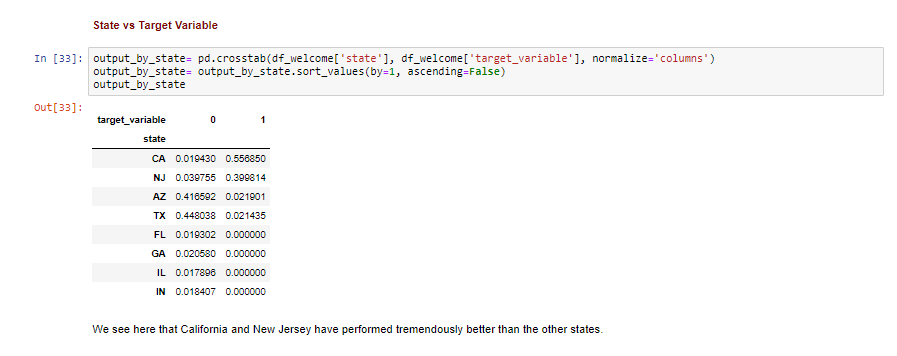


1. In the next couple of cells, let's analyze correlations between categorical columns and the target column using cross-tabulation, where we check the proportion of negative and positive results of the target column again all values of the independent column

Run the next cell to analyze relationship b/w target column and Frequency payment

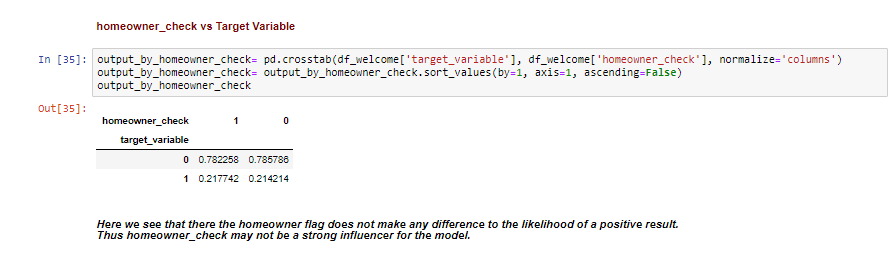


1. Run the next cell to analyze relationship b/w target column and state



1. Run the next cell to analyze relationship b/w target column and homeowner check

* You will observe that this column doesn’t affect the target column in any form and can be dropped before the modeling phase

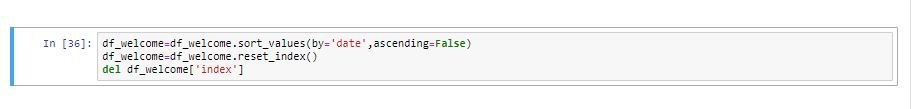


## Session 2 : Feature Engineering

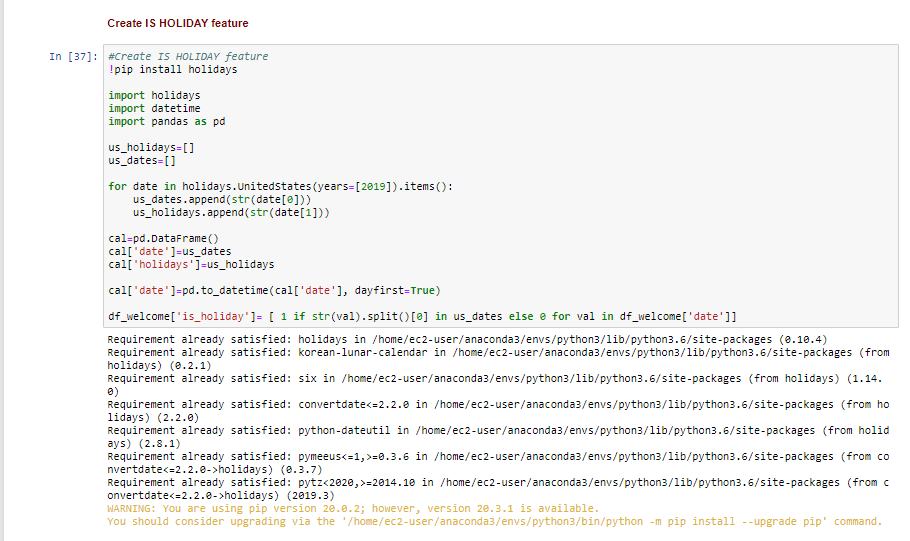
1. The code cells in the next session will create new features as part of the Feature Engineering Process

* Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process.
* In the next couple of cells, we will be extracting new feature based on the Date Feature

Run the next cell to sort the dataset based on ascending dates



1. Run the next cell to create a new feature which checks If the day on which the email was sent was a holiday or not



1. Run the next cell, to check the proportion of emails that we sent on holidays , which turns out to be almost 4%



1. Run the next cell to create new features based on the difference b/w the dates when the email was sent and the Holidays that occurred on that period of time.

* It should take around 4 minutes to execute this part of the script

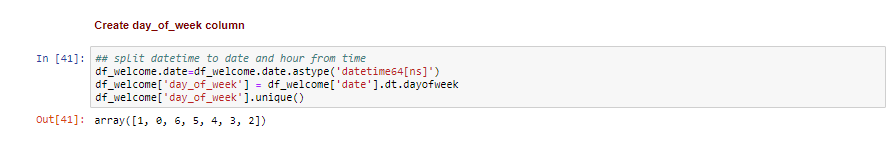


1. Run the net cell to see the bottom 5 records using the.tail() function,

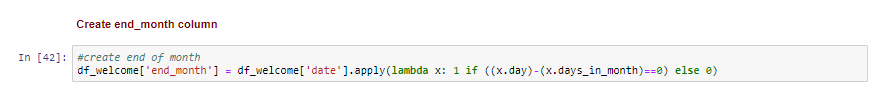
* As seen below, new columns have been created event\_Christmans Day and event\_thanksgiving, etc. with the date difference in hours from the date when the email was sent



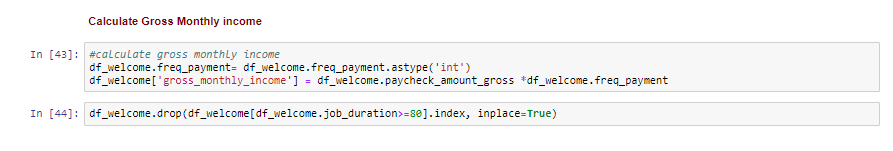
1. Run the next cell to extract day of week for te date when the email was sent. You can see in the output the unique set of values in the new column created



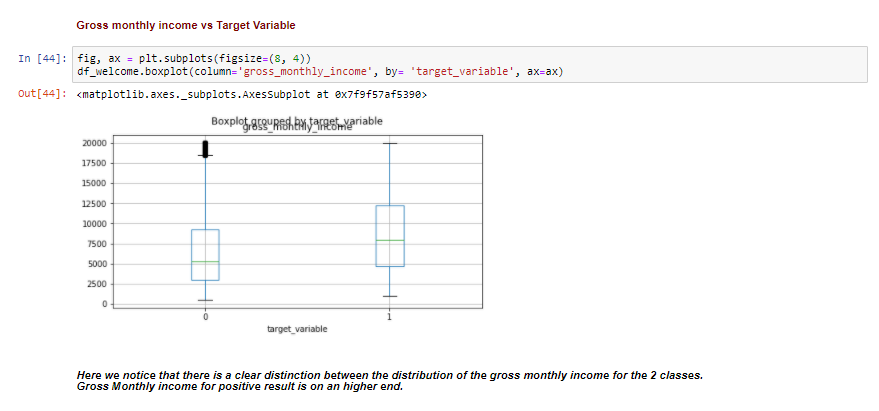
1. Run the next cell to check whether the email was sent on the end of month



1. Let us create new feature gross monthly income based on the relation b/w paycheck amount and frequency of payment. This can be based on an assumption or inputs from the business



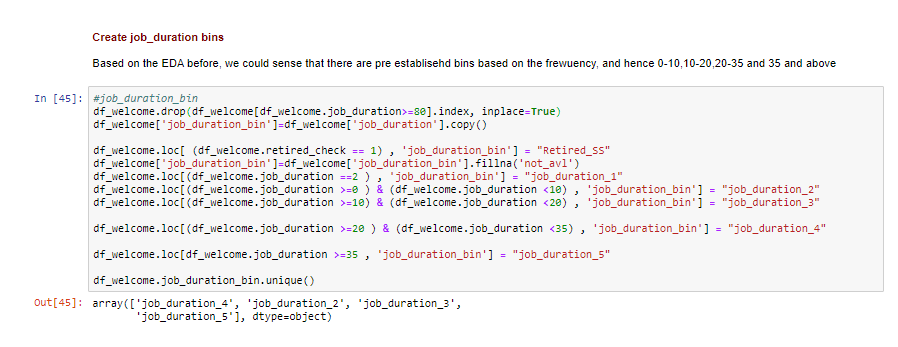
1. In the next cell, we are analyzing the newly create feature with the target column



1. In the next cell, we are using the concept of binning to make more sense out of a continuous feature

* Based on the EDA before, we could sense that there are pre-established bins based on the frequency, and hence 0-10,10-20,20-35 and 35andabove were chosen as the binning criteria.
* This can be based on equal frequency as well, but most of the time the right way is to analyze the column with the target column to figure out natural bins based on patterns.

Run the cell to create bins for job duration



1. Run the next cells to delete the columns not required for analysis/modeling



1. Run the next 3 code cells to execute one hot encoding for categorical columns

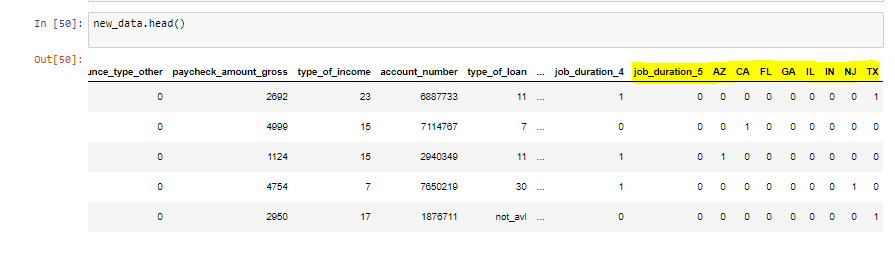
* One of the major problems with Machine Learning is the fact that you cannot work directly with categorical data. Machine Learning is, after all, a bunch of mathematical operations translated to a computer via low-level programming languages.
* Computers are brilliant when dealing with numbers. So, we must somehow convert our input data (in whichever sequential format it be) to numbers. Then only our innocent mathematically gifted GPUs and CPUs will be able to process the data. Once we are done with the processing on the numbers, we need another mechanism to somehow revert the output to the same format as that of the input data. This is where One-Hot Encoding kicks in! One hot encoding creates new (binary) columns, indicating the presence of each possible value from the original data. Let's work through an example.



* The values in the original data are Red, Yellow and Green. We create a separate column for each possible value. Wherever the original value was Red, we put a 1 in the Red column.



1. Run the next cell to check how the one hot encoded column look like



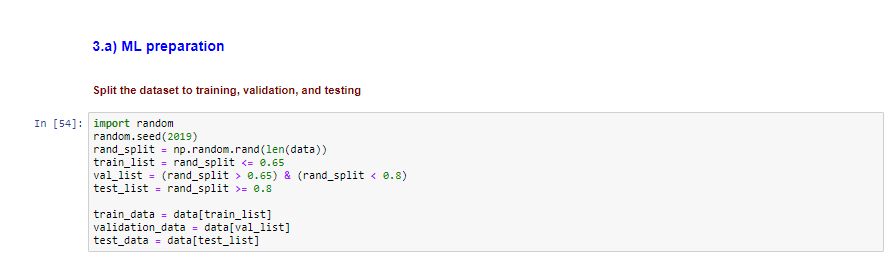
1. Run the next cell to delete the regular categorical columns as new one hot encoded column has been created and then rearange the columns to add user id and date at the beingin of the data set

## Session 3 : Developing a benchmark model

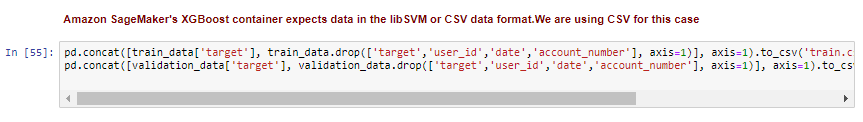
Next session will be about developing a benchmark model and see how it is done in the SageMaker ecosystem

1. Run the next cell to Split the dataset to training, validation, and testing

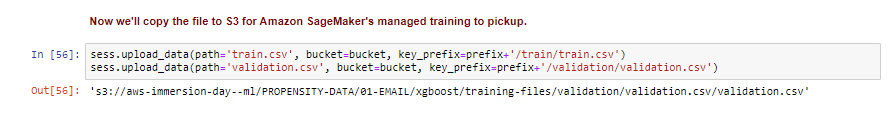
* When building a model whose primary goal is to predict a target value on new data, it is important to understand overfitting.
* Overfitting is an issue within machine learning and statistics where a model learns the patterns of a training dataset too well, perfectly explaining the training data set but failing to generalize its predictive power to other sets of data.
* To put that another way, in the case of an overfitting model it will often show extremely high accuracy on the training dataset but low accuracy on data collected and run through the model in the future
* The most common way of preventing this is to build models with the concept that a model shouldn't only be judged on its fit to the data it was trained on, but also on "new" data. There are several different ways of operationalizing this, holdout validation, cross-validation, leave-one-out validation, etc. For our purposes, we'll simply randomly split the data into 3 uneven groups. The model will be trained on 65% of data, it will then be evaluated on 15%of data to give us an estimate of the accuracy we hope to have on "new" data, and 20% will be held back as a final testing dataset which will be used later on.



1. Run the next cell to convert the data into csv format as Amazon SageMaker's XGBoost container expects data in the libSVM or CSV data format. We are using CSV for this case

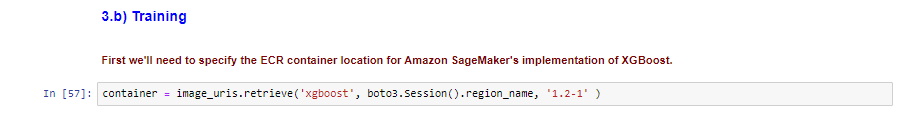


1. Run the next cell so that can copy the data files to S3 for Amazon SageMaker's managed training to pick up. You should be able to see the location as an output where the files are stored

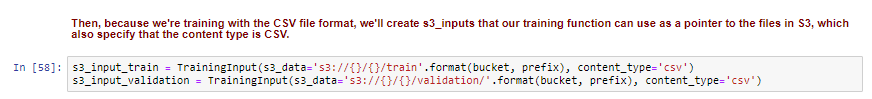


In the next cell, we will run codes cells specific for the training job

* XGBoost is an extremely popular, open-source package for gradient boosted trees. It is computationally powerful, fully featured, and has been successfully used in many machine learning competitions. Let's start with a simple XGBoost model, trained using Amazon SageMaker's managed, distributed training framework.
* Run the next cell to specify the ECR container location for Amazon SageMaker's implementation of XGBoost.



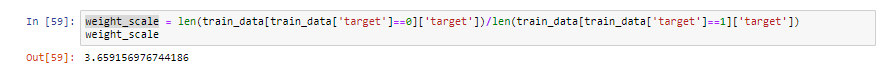
1. In the next cell, because we're training with the CSV file format, we'll create s3\_inputs that our training function can use as a pointer to the files in S3, which also specify that the content type is CSV.



1. In the next cell, we will handle the class imbalance which we identified in the EDA phase

* **scale\_pos\_weight** is parameter in XGBOOST which takes the ratio of number of negative class to the positive class.
* Suppose the dataset has 90 observations of negative class and 10 observations of positive class, then ideal value of scale\_pos\_weight should be 9.

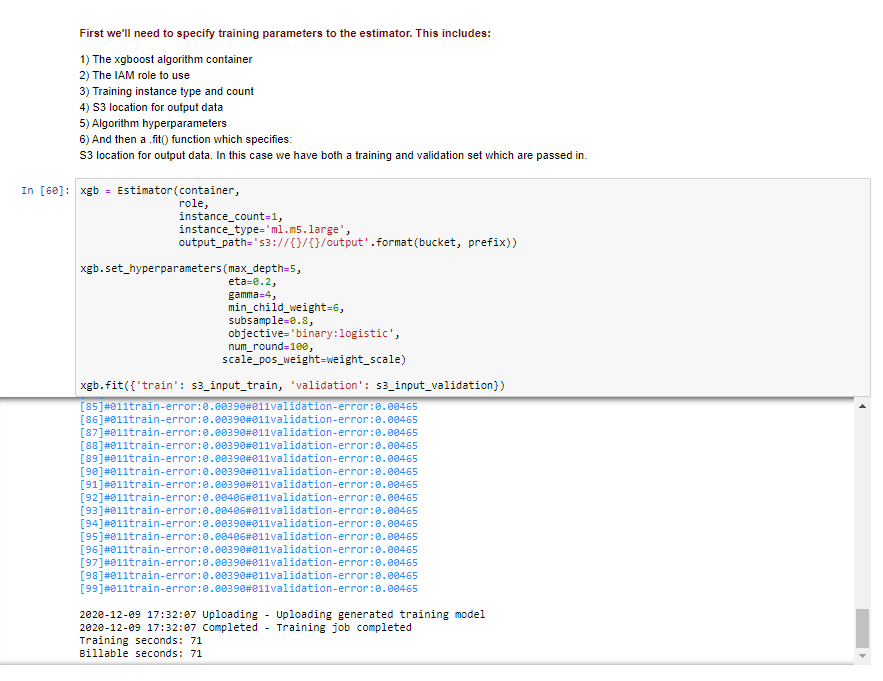
Run the next cell to calculate this value which will be used in the later phase when we will specify the model parameters



1. Run the next code cell to specify training parameters to the estimator. This includes:

* The XGBoost algorithm container
* The IAM role to use
* Training instance type and count
* S3 location for output data
* Algorithm hyperparameters
* And then a .fit() function which specifies S3 location for data. In this case we have both a training and validation set which are passed in.

This .fit() function executes the training job. You should be able to see the billable tiring seconds once the training is complete



1. Furthermore, as advanced functionality, we can employ the SageMaker hyperparameter tuning to find the best set of hyper parameters for tuning the ML model. On the algorithm XGBoost, SageMaker natively supports hyper parameter tuning. As part of the tuning job, SageMaker will run multiple training jobs and vary the hyper parameters in the ranges provided by the user. On execution, the results of the experiment are provided for the data scientists to choose the best model.

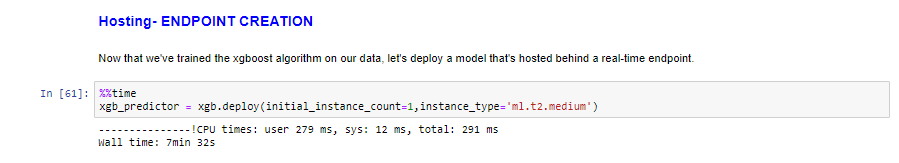
Next, the instructor will show you in the console on how to run a hyper parameter tuning job

## Session 4 : Operationalization and Evaluation

**In the final session, we will go over Operationalization and Evaluation**

* Amazon SageMaker enables you to one-click deploy your model onto autoscaling Amazon ML instances across multiple Availability Zones for high redundancy. Specify the type of instance and the maximum and minimum number desired, and Amazon SageMaker takes care of the rest. It launches the instances, deploys your model, and sets up a secure HTTPS endpoint
* Amazon SageMaker enables you to test multiple models or model versions behind the same endpoint using production variants.
* Amazon SageMaker multi-model endpoints enable you to deploy multiple trained models to an endpoint and serve them using a single serving container. Multi-model endpoints are fully managed and highly available to serve traffic in real time. You can easily invoke a specific model by specifying the target model name as a parameter in your prediction request. This feature is ideal when you have a large number of similar models that you can serve through a shared serving container and don’t need to access all the models at the same time

1. Now that we have trained the XGBoost algorithm on our data, Run the next cell to deploy a model that's hosted behind a real-time endpoint. It should almost 8 minutes



1. Now, if the requirement is to make prediction on predefined batches of data, we will use the match predictions method to extract inference form models hosted behind a SageMaker endpoint which we setup in the previous step

* Now let's determine how we pass data into and receive data from our endpoint. Our data is currently stored as NumPy arrays in memory of our notebook instance. To send it in an HTTP POST request, we'll serialize it as a CSV string and then decode the resulting CSV.
* Note: For inference with CSV format, SageMaker XGBoost requires that the data does NOT include the target column.

Run the next 2 code cells to run a function to

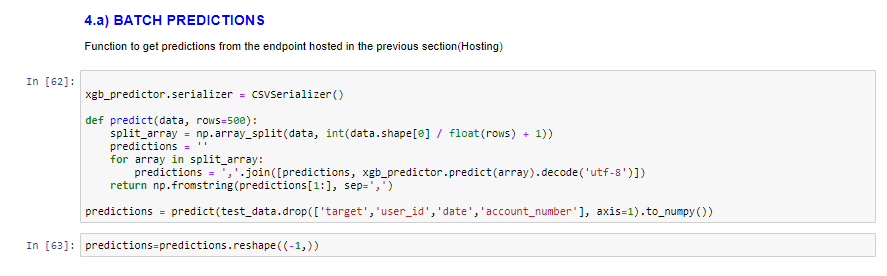
1) Loop over our test dataset

2) Split it into mini batches of rows

3) Convert those mini-batches to CSV string payloads (notice, we drop the target column from our dataset first)

4) Retrieve mini-batch predictions by invoking the XGBoost endpoint

5) Collect predictions and convert from the CSV output our model provides into a NumPy array

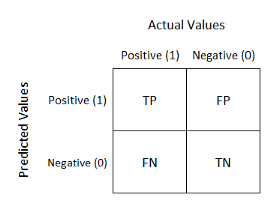


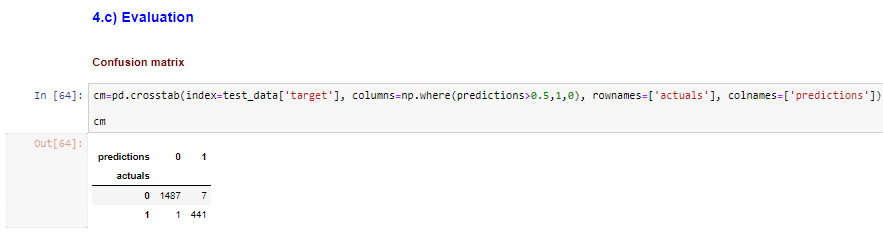
1. Next the instructor will talk about real time predictions

* Once you have created the endpoint, you need a way to invoke it outside a notebook. There are different ways you can invoke your endpoint and the model expects appropriate input (the model signature) when you invoke it. These input parameters can be in a file format such as .csv and .libsvm, an audio file, an image, or a video clip.
* You will use AWS Lambda and Amazon API Gateway to format the input request and invoke the endpoint from the web.
* The following diagram shows how the deployed model is called using serverless architecture. Starting from the client side, a client script calls an Amazon API Gateway API action and passes parameter values. API Gateway is a layer that provides API to the client. In addition, it seals the backend so that AWS Lambda stays and executes in a protected private network. API Gateway passes the parameter values to the Lambda function. The Lambda function parses the value and sends it to the SageMaker model endpoint. The model performs the prediction and returns the predicted value to AWS Lambda. The Lambda function parses the returned value and sends it back to API Gateway. API Gateway responds to the client with that value.



1. Run the next cell to create a confusion matrix for the results





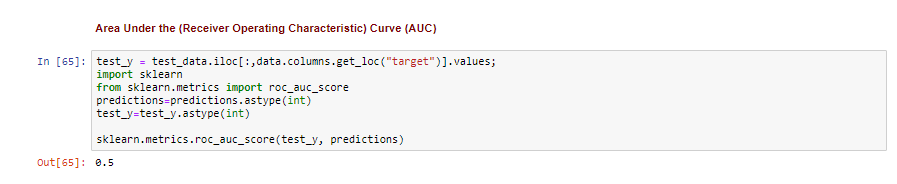
1. Run the next cell to evaluate the model, with AUC

**Area Under the (Receiver Operating Characteristic) Curve (AUC)**

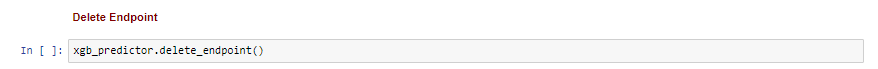
AUC-ROC curve helps us visualize how well our machine learning classifier is performing. AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0

AUC is desirable for the following two reasons:

AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values. AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.



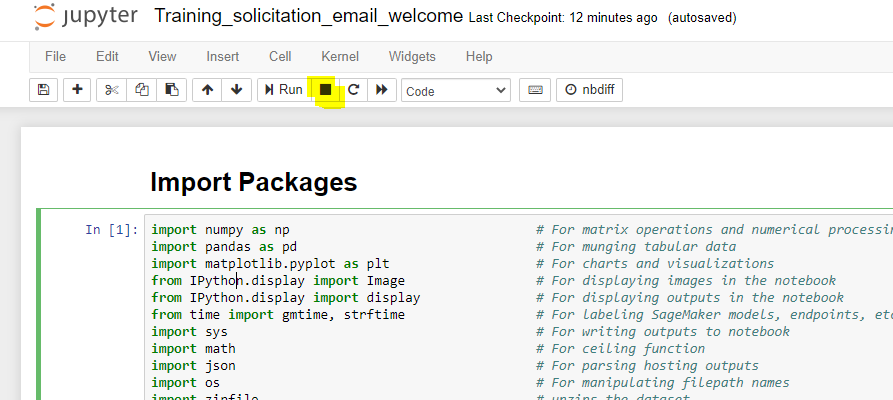
1. Run the next cell to delete the endpoint that was created for making inferences



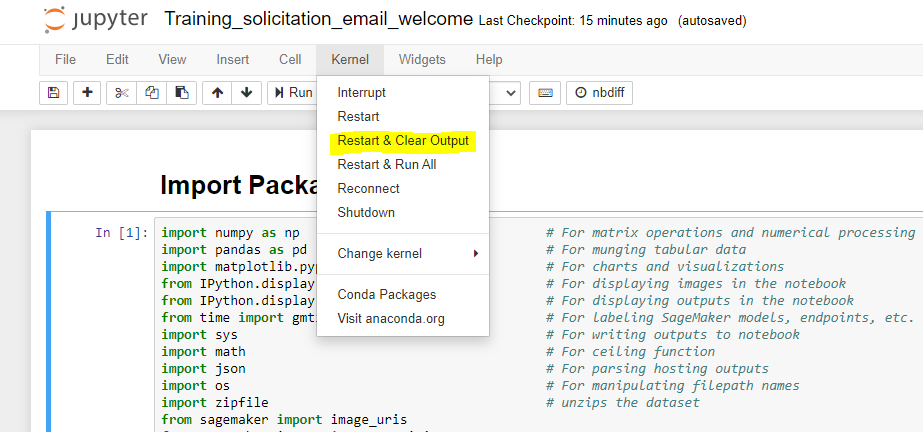
# Instructions in case of errors

In case, you see an error after running any cell on your notebook

1. Try to run the cell from the top in a sequential order
2. In case step 1 does not work, stop the cell if any currently running with an [\*] symbol next to the cell using the **stop square sign** in the menu as shown below



1. In case nothing else works, click on the menu and find **Kernel** and then click on **Restart and Clear Output** (**Kernel -> Restart**)



1. Now you can start running all the cells from the top of the script in a sequential order and follow along with the Instructor
2. In case you face any other challenge, please feel free to contact our coordinators