ILLINOIS INSTITUTE OF TECHNOLOGY

Tone Recognition using Big Data Technologies and Machine Learning

BIG DATA TECHNOLOGIES

(CSP – 554)

Submitted by:

Karan Verma

A20506421

INTRODUCTION

Machine learning is the process through which an algorithm generates predictions without explicitly coding that function into it. The key input to the machine learning model is data. Big data allows for the handling of large datasets. By merging Big Data with machine learning, we can analyse massive volumes of data and produce predictions. It can forecast numerical or categorical values using regression and classification algorithms. Before predictions can be formed, the input data must be organized and cleaned; this technique is known as data cleaning. To understand more about the dataset, exploratory data analysis might be utilized. Divide the input dataset into train and test datasets, then use the training dataset to construct a model to predict the target variable on fresh or test datasets.

Apache Spark is a data processing platform that can effectively analyze large datasets and spread data processing jobs over several computers, either alone or in conjunction with other distributed computing technologies. To do machine learning, Apache Spark makes use of Spark MLlib. MLlib, a scalable Machine Learning library built on Spark, prioritizes both fast performance and high-quality algorithms.At a high level, it provides tools such as:

- Classification, regression, grouping, and collaborative filtering are examples of frequent learning algorithms.
- Featurization includes the following steps: feature extraction, transformation, dimensionality reduction, and selection.
- Pipelines are tools for building, assessing, and adjusting ML pipelines.
- Saving and loading algorithms, models, and Pipelines with consistency
- Utilities include linear algebra, statistics, data processing, and so forth.

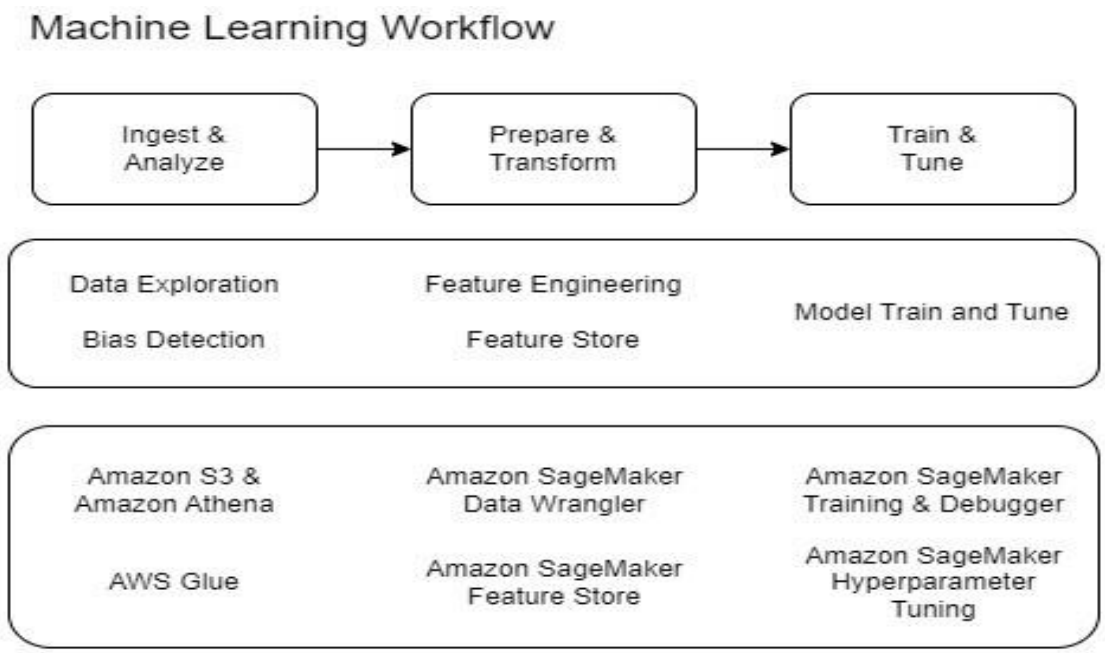
MLlib is a Java, Python, R, and Scala-based software that can analyze large data on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can be implemented in Java, Python, and R and operate on Hadoop, Apache Mesos, and Kubernetes. Almost all machine learning algorithms are now accessible with Spark MLlib, which is faster than Map Reduce.

LITERATURE REVIEW

1. Sentiment Analysis

Machine Learning models are fed numerical values. Because the reviews are made up of sentences, we need to find a way to represent them in a way that a machine learning algorithm can understand, i.e. as a list of numbers, in order to extract patterns from the data. Emotions are important not just in personal life but also in business. How your consumers and target audience perceive your goods or brand gives you the context you need to analyze and enhance your product, company, marketing, and communications strategy. Sentiment analysis, often known as opinion mining, assists researchers and businesses in extracting insights from user-generated social media and online material.

Regardless of business or vertical, understanding consumers' opinions about the brand and goods has become critical. With fierce competition for high-paying employment in the NLP and ML industries, a dull cookie-cutter résumé may not suffice. Working on a sentiment analysis project using actual datasets, on the other hand, will help you stand out in job applications and increase your chances of getting a call back from your desired firm.



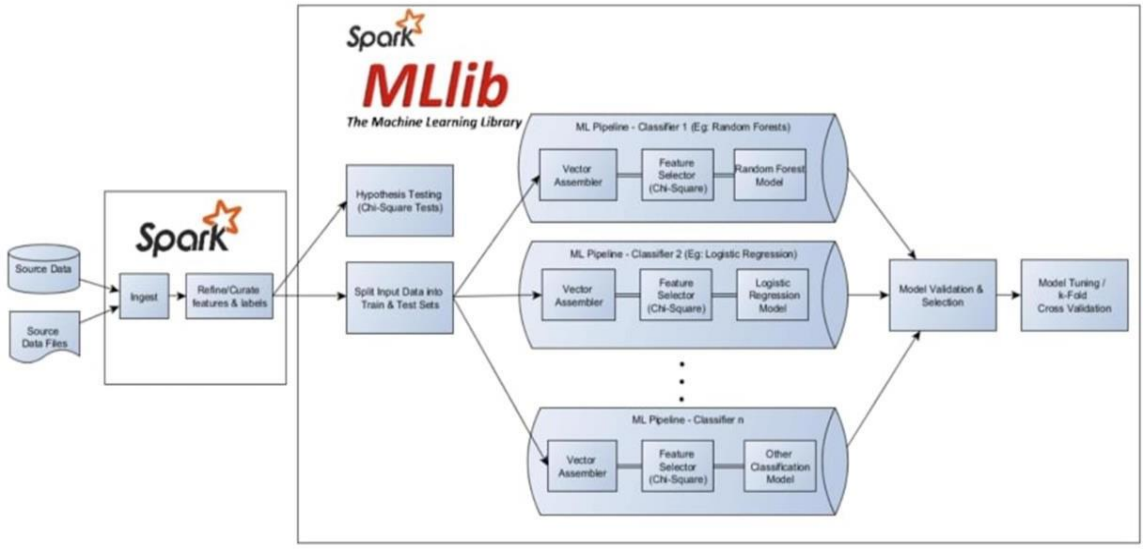
PROBLEM STATEMENT

It is much more difficult to determine someone's tone in writing than in conversation. To begin, you cannot determine an author's emotional state based on their facial expressions or body language. For example, the tone of a communication is rarely affected by a single factor.

Effective communication involves many factors. Because, in some cases, what you say is as essential as how you say it. Our technique is intended to provide feedback on a variety of keywords that convey the message's sentiment. By making smart decisions about how you transmit your information, you may ensure that your audience will focus on the facts you're presenting rather than the unintended repercussions of your tone.

WORK-FLOW

The steps implemented in this project are explained in the flowchart diagram below.



OBJECTIVE

To create a tone detector utilizing Spark-MLibs and Sagemaker that outperforms or equals state-of-the-art models.

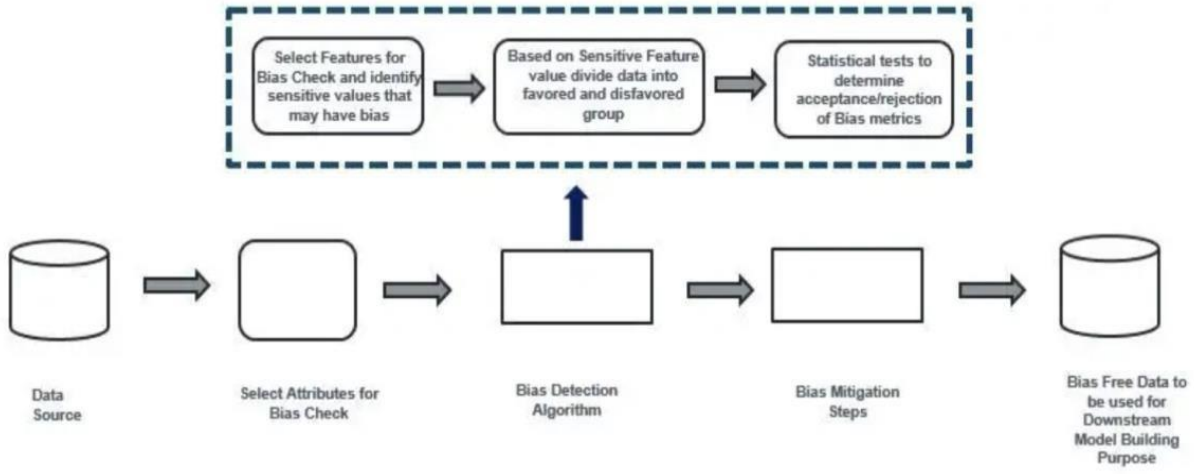
Bias Detection using AWS SageMaker Clarify

To demonstrate the effectiveness of Bias Detection solution we applied AWS SageMaker Clarify for bias detection on two datasets:

• Problem Statement:

1. Identify bias in datasets
2. Identify bias in Model Predictions.

• Solution: Approach for Detecting Biases Using SageMaker Clarify: We use SageMaker Clarify to discover biases in the aforementioned datasets. A high-level solution strategy for detecting bias in datasets is provided here.



Steps:

i. In the first step we identify those attributes that may cause bias.

ii. Once the sensitive attributes have been identified, we can identify the range of values in these attributes that represent a disfavored group.

iii. The sensitive attribute and sensitive value is fed to the Bias Algorithm.

iv. Within the Bias algorithm, the dataset is divided into a favored and disfavoured group based on the sensitive features and values specified.

v. Statistical tests are done, and pre-training and post-training metrics are computed for bias detection. Based on interpretation of statistical metrics, we identify the presence/absence of Bias.

Results of Exploratory Analysis: Data exploration was performed on the two datasets prior to the bias identification method.

AWS Sagemaker

SageMaker

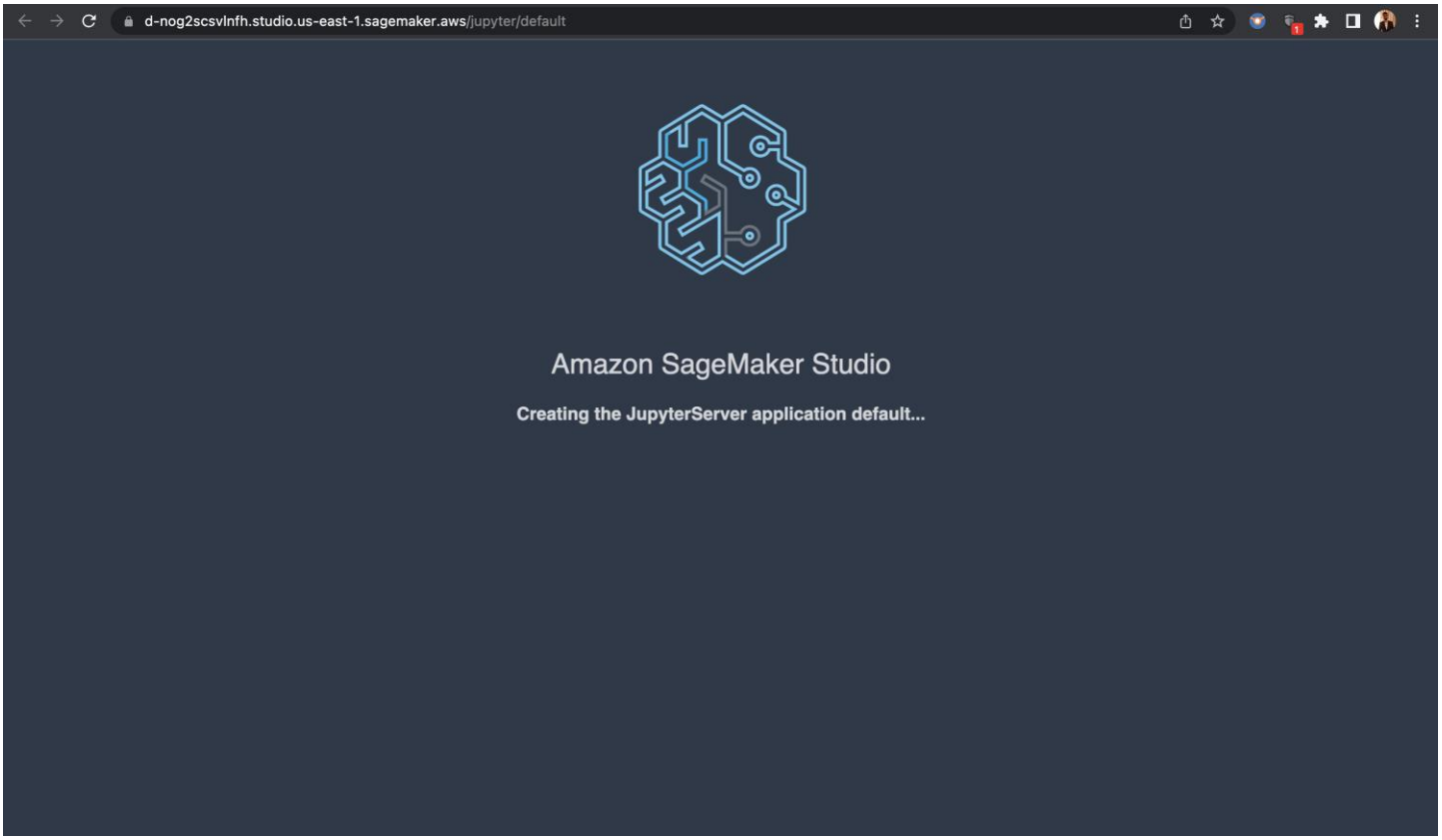
Amazon SageMaker automated model tuning, also known as hyperparameter tuning, identifies the optimum version of a model by executing a large number of training jobs on your dataset with the method and hyperparameter ranges that you select. It then selects the hyperparameter values that result in the best-performing model, as assessed by a metric you choose. When running training jobs, Amazon SageMaker automatic model tuning can use an Amazon EC2 Spot instance to reduce costs. Before you begin using hyperparameter tuning, you should have a well-defined machine learning problem that includes the following elements:

- A dataset
- An comprehension of the sort of algorithm to be trained
- A clear concept of how success is measured

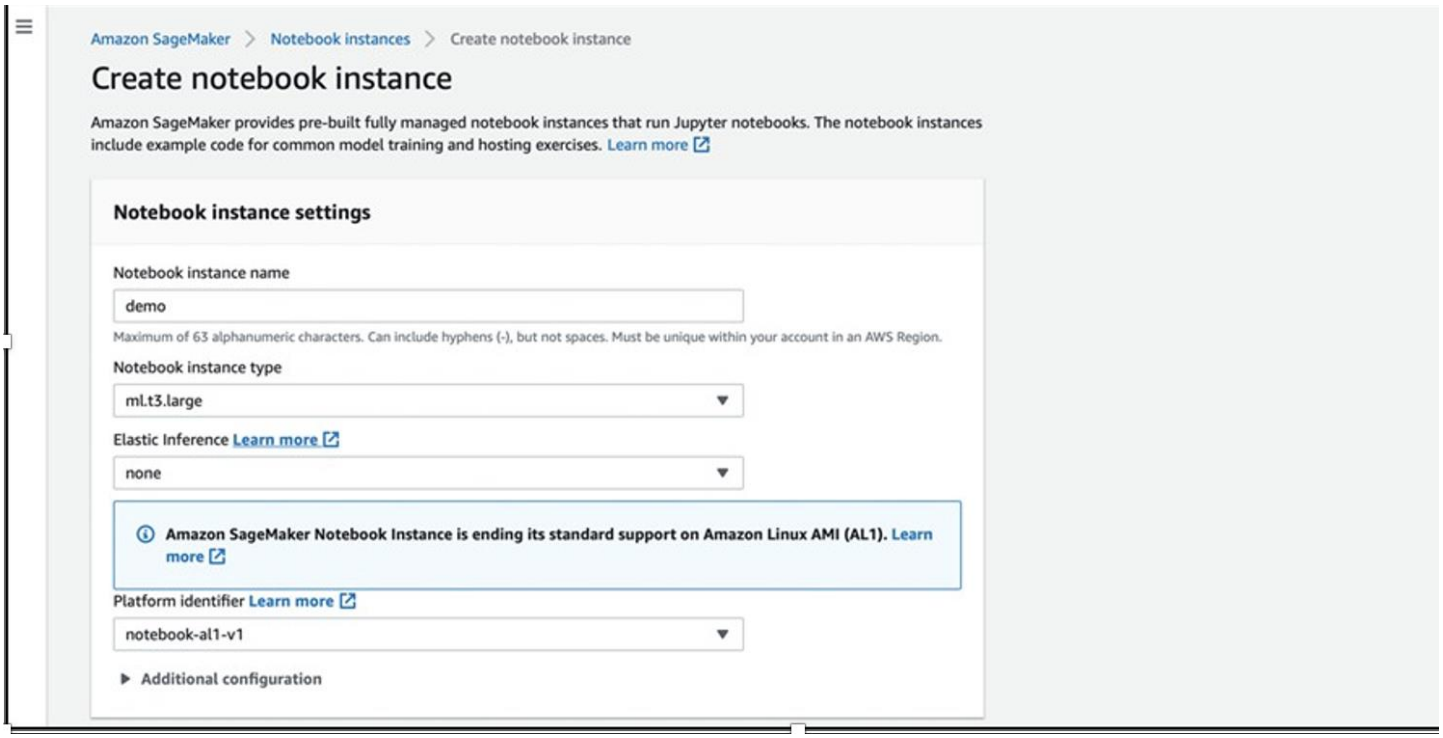
The fully-managed service Amazon SageMaker covers the whole machine learning process, including labeling and preparing data, picking an algorithm, training the model, fine-tuning and optimizing it for deployment, generating predictions, and taking action.

We used Sage Maker to create the model with the provided dataset; the stages are as follows:

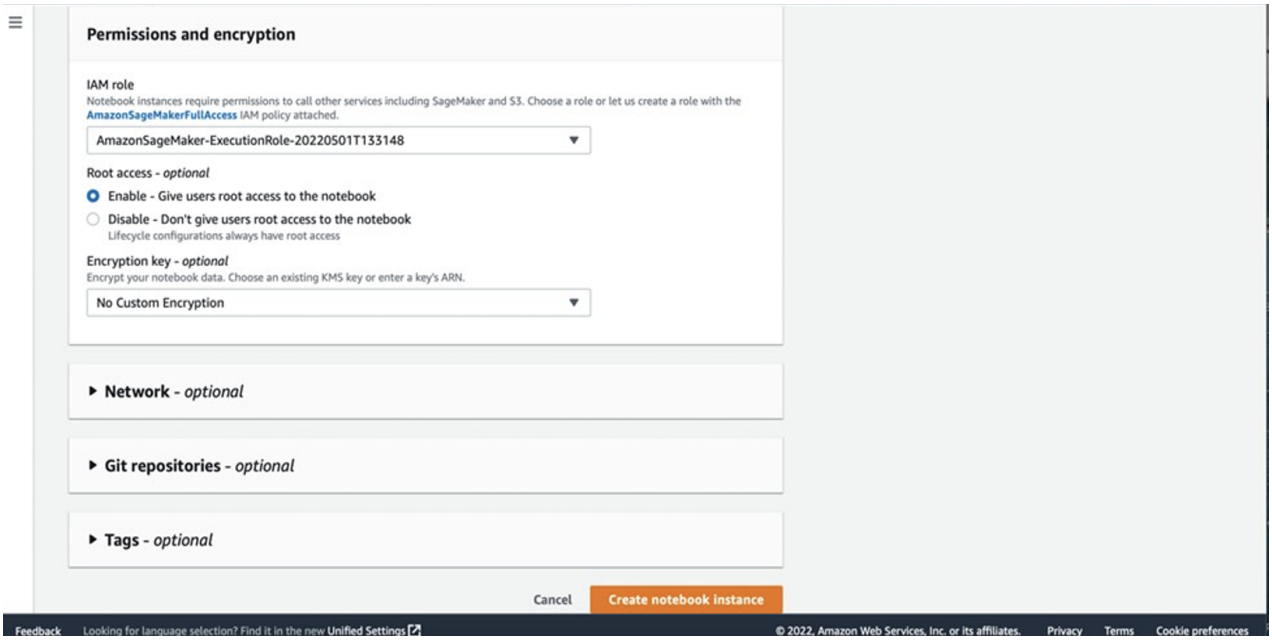
1. Starting up Amazon Sagemaker Studio



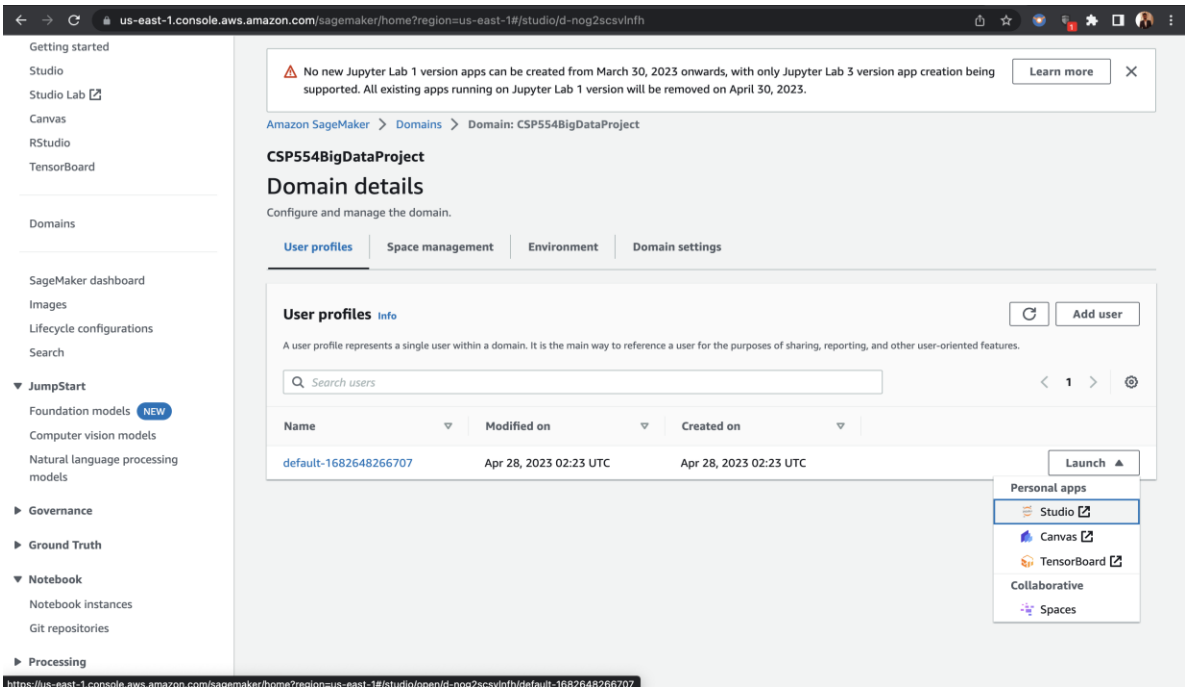
1. Create a notebook Instance



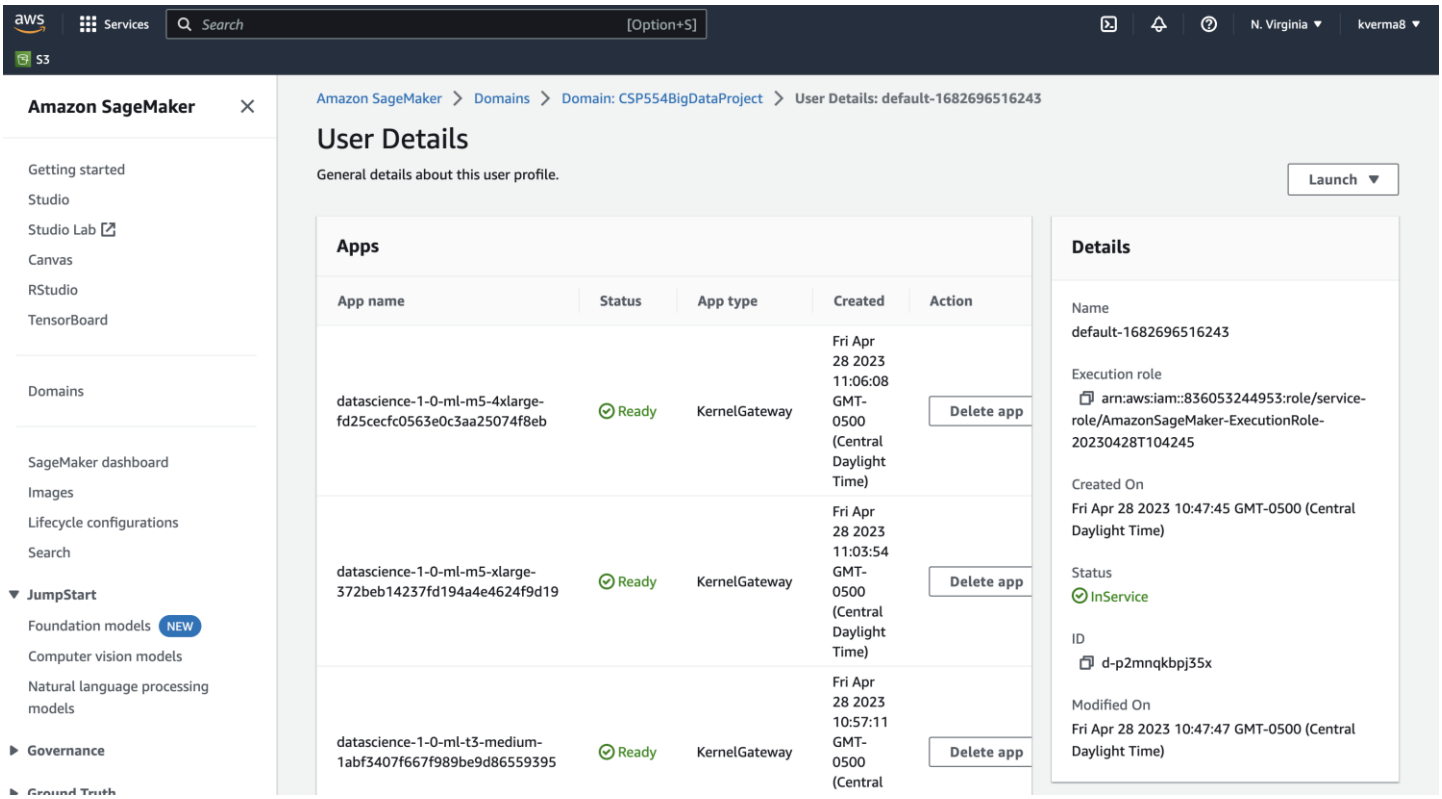
2. Configuring notebook instance for aws sagemaker



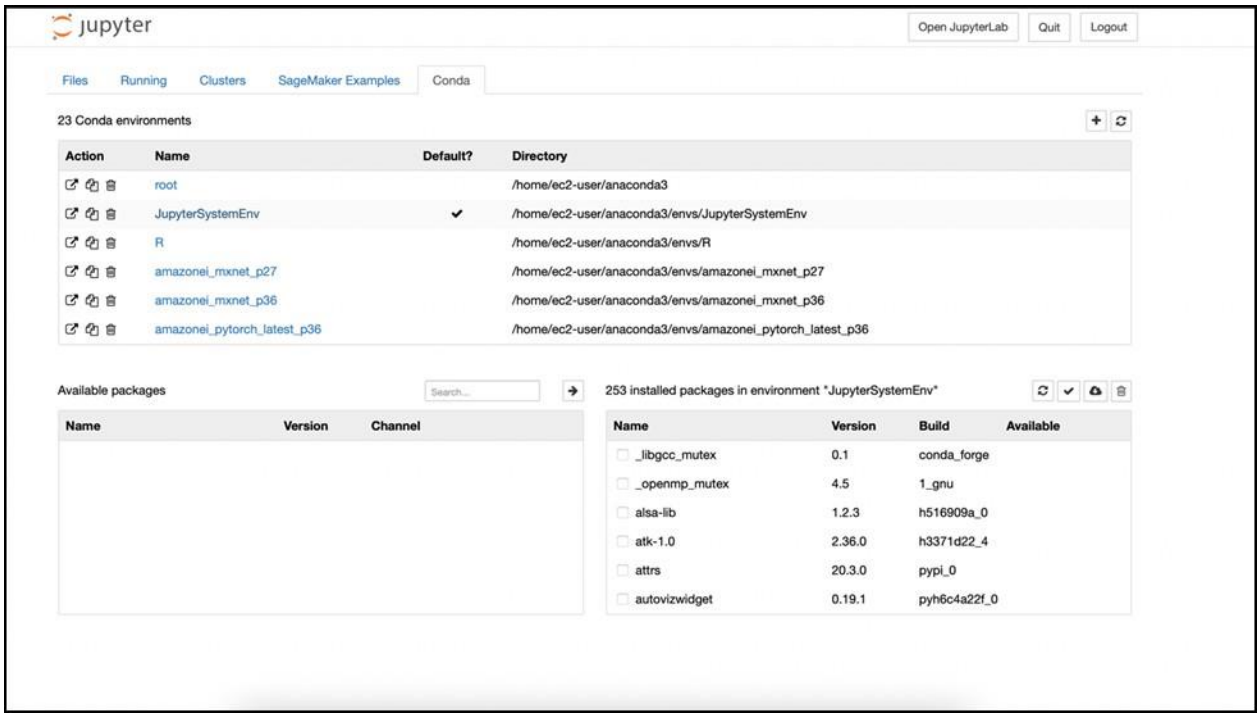
3. Setting up Studio prerequisites



4. Setting up User & Cluster configuration



5. Verify and check configuration for notebook instance we created



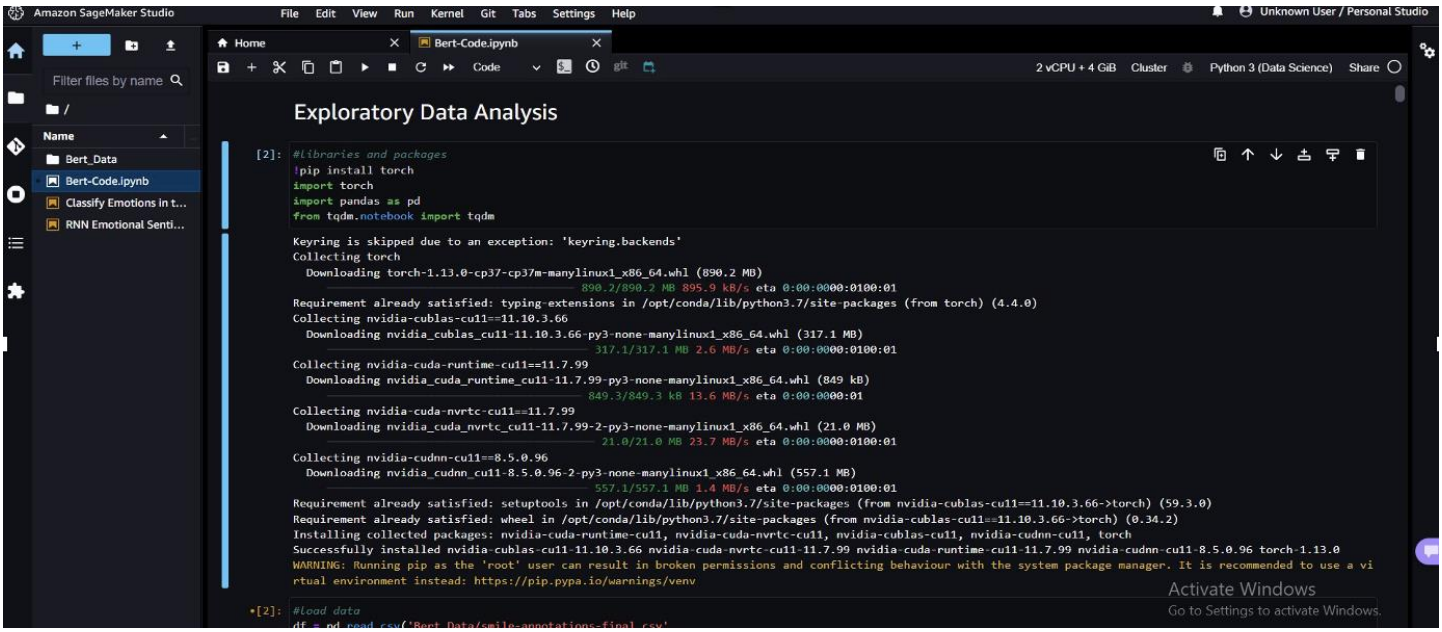
We also verified and checked configuration for notebook instances we created like the environment, programming language of your notebook, processing configuration (gpu, number of cores, etc.)

Implementation of Machine Learning Model in AWS Sagemaker

Stage-1 : Ingest And Analyze (Data Loading, Exploration & Data Preprocessing)

In this stage, we have cleaned and preprocessed data in order to fit the data into models. we startedwith following steps

- Installing required Python packages



- Then read the given data in your notebook (here we have used CSV format).

```
[2]: #load data
     df = pd.read_csv('Bert_Data/smile-annotations-final.csv',
                      names = ['id', 'text', 'category'])

     #reset index
     df.set_index('id', inplace = True)

[3]: #preview
     df.head()
```

[3]:

id	text	category
611857364396965889	@aandraous @britishmuseum @AndrewsAntonio Merc...	nocode
614484565059596288	Dorian Gray with Rainbow Scarf #LoveWins (from...	happy
614746522043973632	@SelectShowcase @Tate_StIves ... Replace with ...	happy
614877582664835073	@Sofabsports thank you for following me back. ...	happy
611932373039644672	@britishmuseum @TudorHistory What a beautiful ...	happy

- after loading the data, we start with data Exploration with understanding on **column type**, checking for **null values**

```
In [9]: #info
        df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 3085 entries, 611857364396965889 to 611566876762640384
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   text      3085 non-null   object
 1   category  3085 non-null   object
dtypes: object(2)
memory usage: 72.3+ KB

In [10]: #check for null
         df.isnull().sum()

Out[10]: text        0
         category    0
         dtype: int64
```

Following On same Line, we **calculated categorical classes** already mentioned in dataset

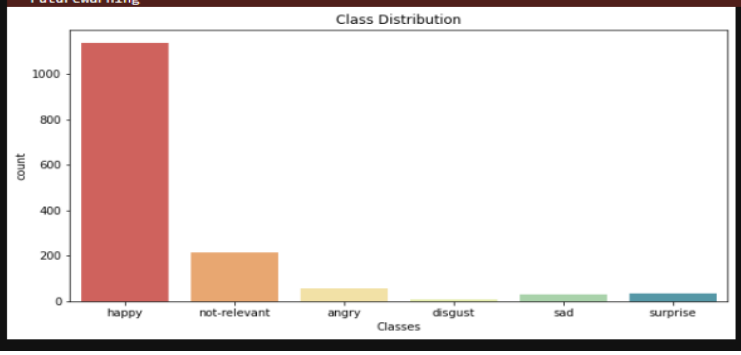
```
In [12]:
#count for each class
df.category.value_counts()

Out[12]: nocode              1572
         happy               1137
         not-relevant         214
         angry                 57
         surprise              35
         sad                   32
         happy|surprise        11
         happy|sad              9
         disgust|angry          7
         disgust                6
         sad|disgust            2
         sad|angry              2
         sad|disgust|angry      1
         Name: category, dtype: int64
```

- Dropping unwanted/undefined classes for category

```
In [13]:
#drop irrelevent class
df = df[~df.category.str.contains('\|')]

In [14]:
#drop irrelevent class
df = df[df.category != 'nocode']

In [15]:
#final classes
df.category.value_counts()

Out[15]: happy           1137
         not-relevant     214
         angry             57
         surprise          35
         sad               32
         disgust            6
         Name: category, dtype: int64
```

- Visualizing final Prepared Data

```
In [16]:
import matplotlib.pyplot as plt
import seaborn as sns

#plot class distribution
plt.figure(figsize=(10, 5))
sns.countplot(df.category, palette='Spectral')
plt.xlabel('Classes')
plt.title('Class Distribution');

/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, th
e only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

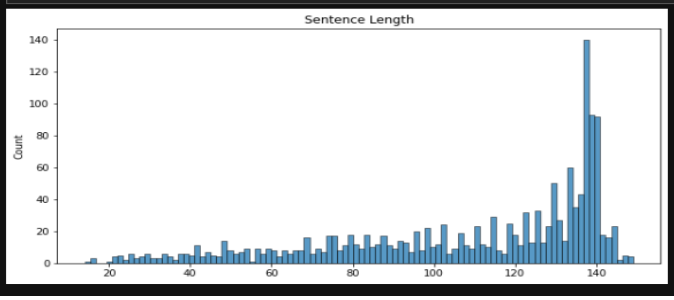


- Finally, **Converted labels** to numerical values

```
In [18]:
#convert Labels into numeric values
label_dict = {}
for index, possible_label in enumerate(possible_labels):
    label_dict[possible_label] = index

In [19]:
label_dict

Out[19]: {'happy': 0,
 'not-relevant': 1,
 'angry': 2,
 'disgust': 3,
 'sad': 4,
 'surprise': 5}
```

- Plotting **histogram** to understand Sentence length

```
In [21]:
#need equal length sentences
#plot hist of sentence length
plt.figure(figsize=(10, 5))
sns.histplot([len(s) for s in df.text], bins=100)
plt.title('Sentence Length')
plt.show()
```



- Splitting dataset into Training dataset, validation dataset and Test dataset

```
In [23]:
from sklearn.model_selection import train_test_split

#train test split
X_train, X_val, y_train, y_val = train_test_split(df.index.values,
                                                  df.label.values,
                                                  test_size = 0.15,
                                                  random_state = 17,
                                                  stratify = df.label.values)
```

```
[20]: #fill in data type
      df.loc[X_train, 'data_type'] = 'train'
      df.loc[X_val, 'data_type'] = 'val'

[21]: df.groupby(['category', 'label', 'data_type']).count()
```

[21]:

category	label	data_type	text
angry	2	train	48
		val	9
disgust	3	train	5
		val	1
happy	0	train	966
		val	171
not-relevant	1	train	182
		val	32
sad	4	train	27
		val	5
surprise	5	train	30
		val	5

- **Now we are ready to develop models and implement them**.

- Here we have implemented 2 machine learning models for the classification of the given dataset:

  1) RoBERTa model

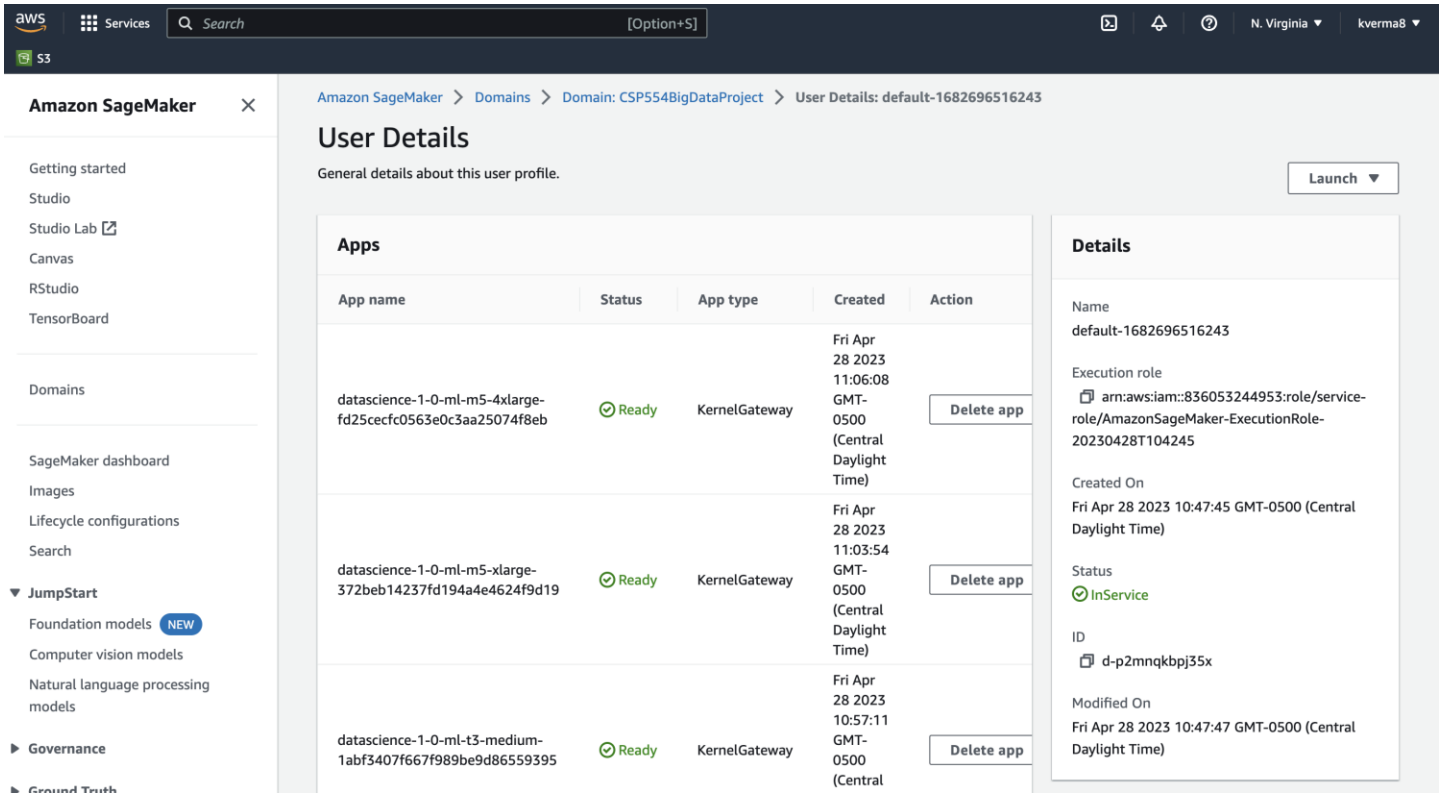
  2) RNN Model

Analyzed the performance of all three models on amazon sagemaker using 2 parameters accuracy and loss.

Implementing Sentiment Analysis using RobERTa Model

RoBERTa: RoBERTa(Robustly Optimized BERT pre-training Approach) expands on the BERT model architecture while changing some model hyperparameters and the model's training process, including the use of more training data. In contrast to BERT, this can actually result in notable performance gains in a number of NLP tasks.

1. Sagemaker Domain-Cluster Setup for RoBERTa Model



2. **Installing required Packages** for RoBERTa Implementation.

Step-1 Install packages

In [2]:

```
!pip3 uninstall keras-nightly
!pip3 uninstall -y tensorflow
!pip3 install keras==2.1.6
!pip3 install tensorflow==2.7.0
!pip3 install h5py==2.10.0
```

```
Keyring is skipped due to an exception: 'keyring.backends'
WARNING: Skipping keras-nightly as it is not installed.
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended t
o use a virtual environment instead: https://pip.pypa.io/warnings/venv
Keyring is skipped due to an exception: 'keyring.backends'
WARNING: Skipping tensorflow as it is not installed.
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended t
o use a virtual environment instead: https://pip.pypa.io/warnings/venv
Keyring is skipped due to an exception: 'keyring.backends'
Collecting keras==2.1.6
  Using cached Keras-2.1.6-py2.py3-none-any.whl (339 kB)
Requirement already satisfied: h5py in /opt/conda/lib/python3.7/site-packages (from keras==2.1.6) (2.10.0)
```

3. **Load dataSet** to Jupyter file using readCSV

In [8]:

```
train_data = pd.read_csv('emotions-dataset-for-nlp/train.txt', names=['text', 'emotion'], sep=';')
val_data = pd.read_csv('emotions-dataset-for-nlp/val.txt', names=['text', 'emotion'], sep=';')
test_data = pd.read_csv('emotions-dataset-for-nlp/test.txt', names=['text', 'emotion'], sep=';')
train_data.head()
```

Out[8]:

	text	emotion
0	i didnt feel humiliated	sadness
1	i can go from feeling so hopeless to so damned...	sadness
2	im grabbing a minute to post i feel greedy wrong	anger
3	i am ever feeling nostalgic about the fireplac...	love
4	i am feeling grouchy	anger

4. Data Pre-Processing

Data preprocessing

In [10]:

```
data = {'Train Data': train_data, 'Validation Data': val_data, 'Test Data': test_data}
for temp in data:
    print(temp)
    print(data[temp].isnull().sum())
    print('*'*20)
```

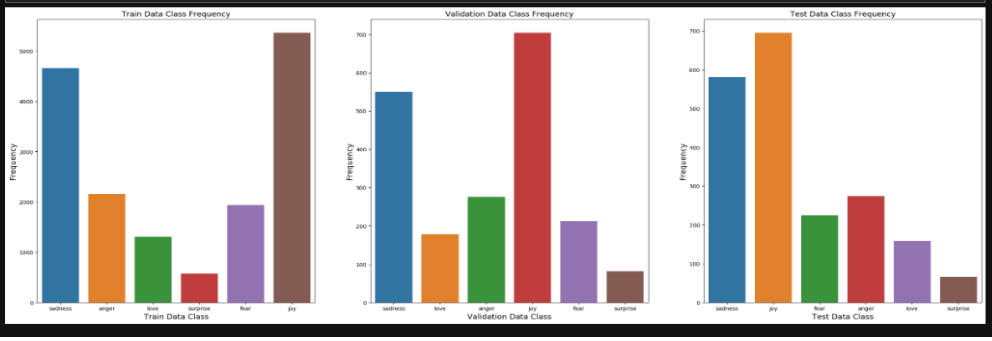
```
Train Data
text       0
emotion    0
dtype: int64
********************
Validation Data
text       0
emotion    0
dtype: int64
********************
Test Data
text       0
emotion    0
dtype: int64
********************
```

5. **Class Distribution** for Training, validation and Testing dataset

Class Distribution in Train, val and test dataset

In [11]:

```
bar, ax = plt.subplots(1,3, figsize=(30, 10))
for index, temp in enumerate(data):
    sns.countplot(ax = ax[index],x = 'emotion', data = data[temp])
    ax[index].set_title(temp+' Class Frequency', size=14)
    ax[index].set_ylabel('Frequency', size=14)
    ax[index].set_xlabel(temp+' Class', size=14)

plt.show()
```



6. Data Processing, Label Creation and Sentence Listing

Data Processing, Label Creation and Seentence Listing

In [14]:

```
def preprocess(sentence):
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    sentence = re.sub('[^A-z]', ' ', sentence)
    negative = ['not', 'neither', 'nor', 'but', 'however', 'although', 'nonetheless', 'despite', 'except',
                'even though', 'yet']
    stop_words = [z for z in stop_words if z not in negative]
    preprocessed_tokens = [lemmatizer.lemmatize(contractions.fix(temp.lower())) for temp in sentence.split() if temp not in stop_words] #Lemmatization
    return ' '.join([x for x in preprocessed_tokens]).strip()
```

In [16]:

```
train_data['text'] = train_data['text'].apply(lambda x: preprocess(x))
val_data['text'] = val_data['text'].apply(lambda x: preprocess(x))
test_data['text'] = test_data['text'].apply(lambda x: preprocess(x))
```

Note: As class imbalanced is evident, RandomOverSampler is used to add data(repetition) to all classes except highest frequency class.

In [17]:

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
train_x, train_y = ros.fit_resample(np.array(train_data['text']).reshape(-1, 1), np.array(train_data['emotion']).reshape(-1, 1))
train = pd.DataFrame(list(zip([x[0] for x in train_x], train_y)), columns = ['text', 'emotion'])
```

Applying OneHotEncoder on target of all dataset

In [18]:

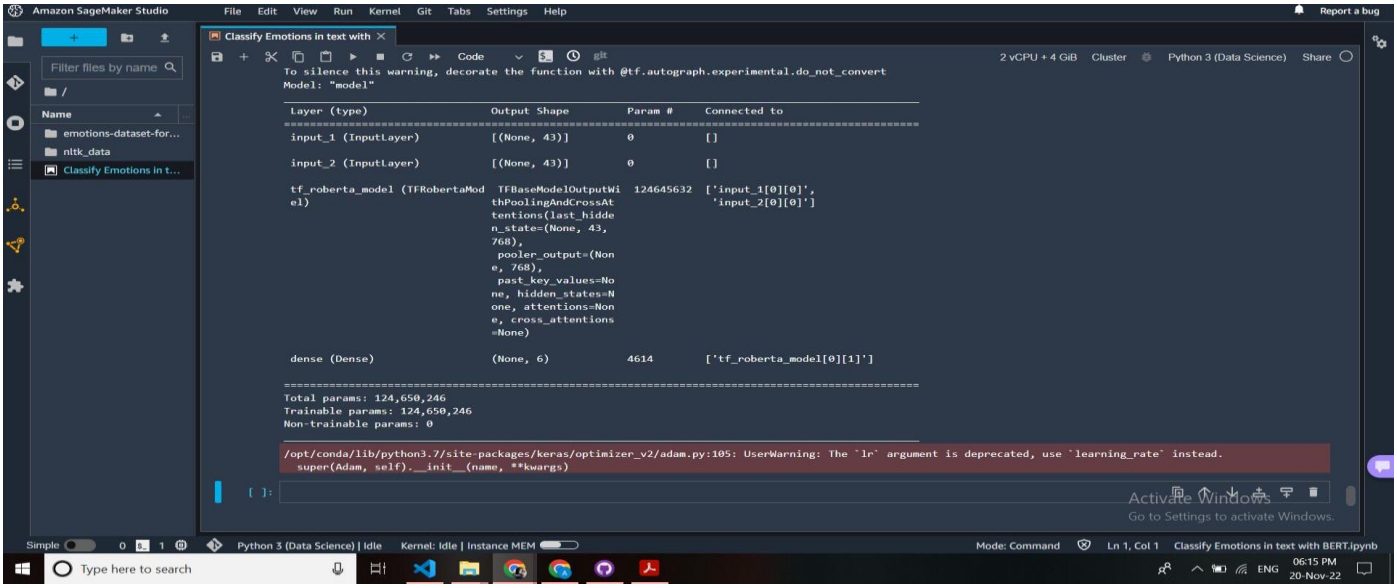
```
from sklearn import preprocessing
le = preprocessing.OneHotEncoder()
y_train= le.fit_transform(np.array(train['emotion']).reshape(-1, 1)).toarray()
y_test= le.fit_transform(np.array(test_data['emotion']).reshape(-1, 1)).toarray()
y_val= le.fit_transform(np.array(val_data['emotion']).reshape(-1, 1)).toarray()
```

7. Model Creation

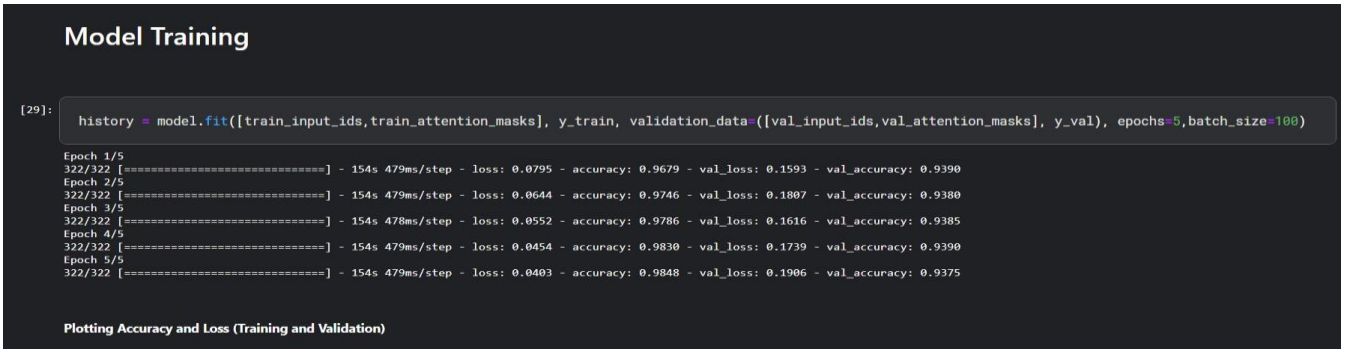
In [24]:

```
def create_model(bert_model, max_len):
    input_ids = tf.keras.Input(shape=(max_len,),dtype='int32')
    attention_masks = tf.keras.Input(shape=(max_len,),dtype='int32')

    output = bert_model([input_ids,attention_masks])
    output = output[1]

    output = tf.keras.layers.Dense(6, activation='softmax')(output)
    model = tf.keras.models.Model(inputs = [input_ids,attention_masks],outputs = output)
    model.compile(Adam(lr=1e-5), loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

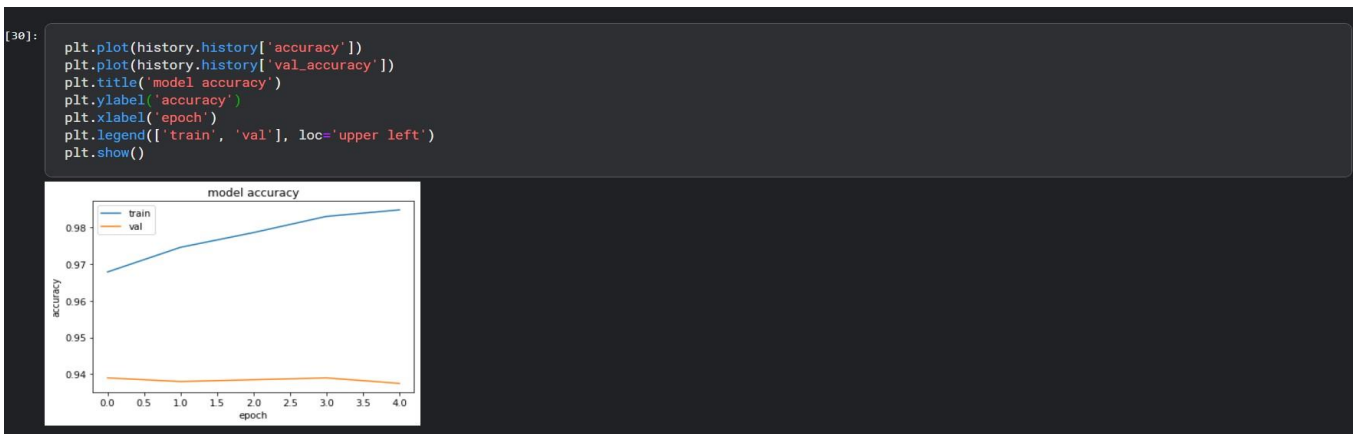
8. RobertA Model Description



9. RoBERTa Model Training

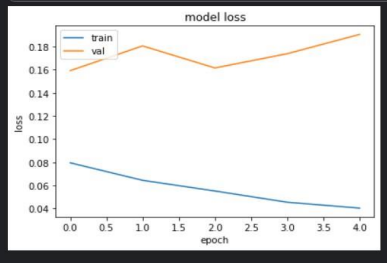


10. Plotting Model Accuracy of Training and Validation dataset



11. Plotting Model Loss of Training and validation dataset

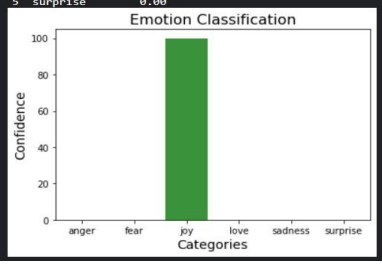
```
[31]: plt.plot(history.history['loss'])
      plt.plot(history.history['val_loss'])
      plt.title('model loss')
      plt.ylabel('loss')
      plt.xlabel('epoch')
      plt.legend(['train', 'val'], loc='upper left')
      plt.show()
```



12. Test results on Different text example

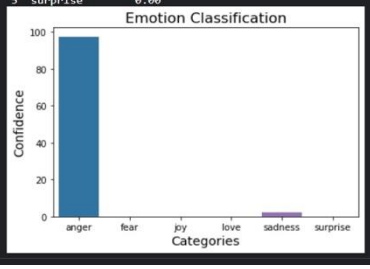
```
[39]: result = inference("I am happy", max_len)
      print(result)
```

```
/opt/conda/lib/python3.7/site-packages/transformers/tokenizat
e `padding=True` or `padding='longest'` to pad to the longes
max_length` (e.g. `max_length=45`) or leave max_length to No
  FutureWarning,
   Category  Confidence
0     anger        0.02
1      fear        0.00
2       joy       99.97
3      love        0.00
4   sadness        0.01
5  surprise        0.00
```



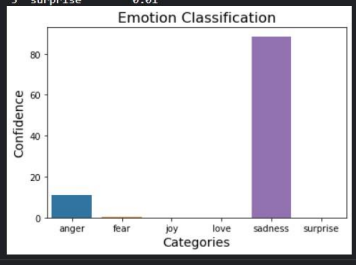

```
[40]: result = inference("I am unhappy", max_len)
      print(result)
```

```
/opt/conda/lib/python3.7/site-packages/transformers/tokenizat
e `padding=True` or `padding='longest'` to pad to the longes
max_length` (e.g. `max_length=45`) or leave max_length to No
  FutureWarning,
   Category  Confidence
0     anger       97.39
1      fear        0.01
2       joy        0.14
3      love        0.00
4   sadness        2.45
5  surprise        0.00
```

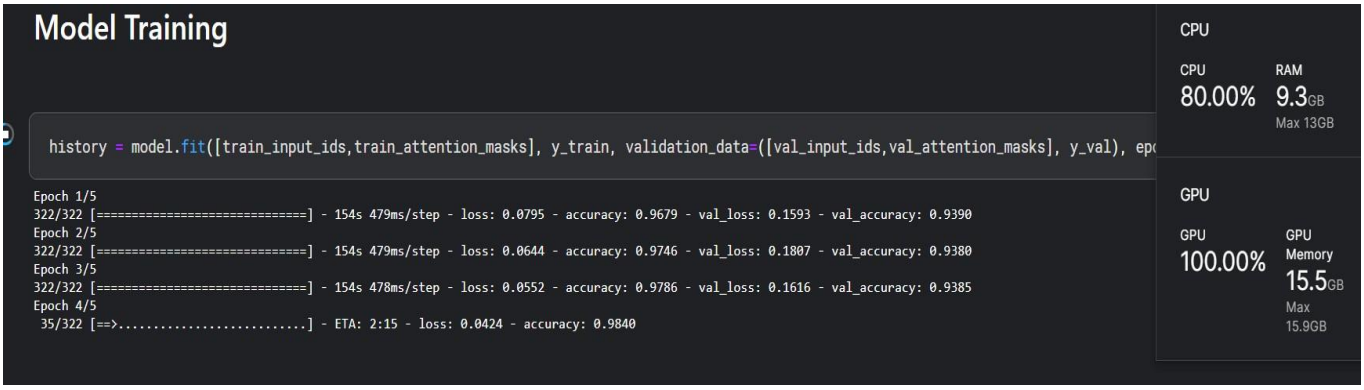



```
[41]: result = inference("I am depressed", max_len)
      print(result)
```

```
/opt/conda/lib/python3.7/site-packages/transformers/tokenizat
e `padding=True` or `padding='longest'` to pad to the longes
max_length` (e.g. `max_length=45`) or leave max_length to No
  FutureWarning,
   Category  Confidence
0     anger       11.15
1      fear        0.19
2       joy        0.09
3      love        0.00
4   sadness       88.56
5  surprise        0.01
```



13. Capturing CPU & GPU utilization



14. Comparison of Performance on different Clusters:

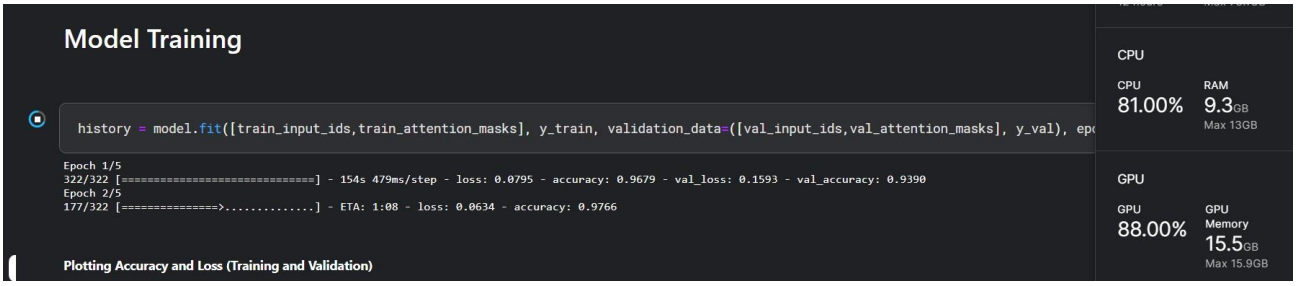
We ran the aforementioned Machine Learning model on "m5-large" (3 node) and "m5-xlarge" (5 node) cluster instances after receiving results on "t3-medium" (3 node) cluster instances.

Observation:

- Code execution was quick; epoch estimation time was less in m5-large than in t3-medium. The "m5-xlarge"cluster was even faster.

- During training, the CPU usage for t3-medium hit 100% (above snap), while it reached 88% for m5-large and m5-xlarge instances.

- As a result, we can once again state that better CPU arrangement improves execution performance.



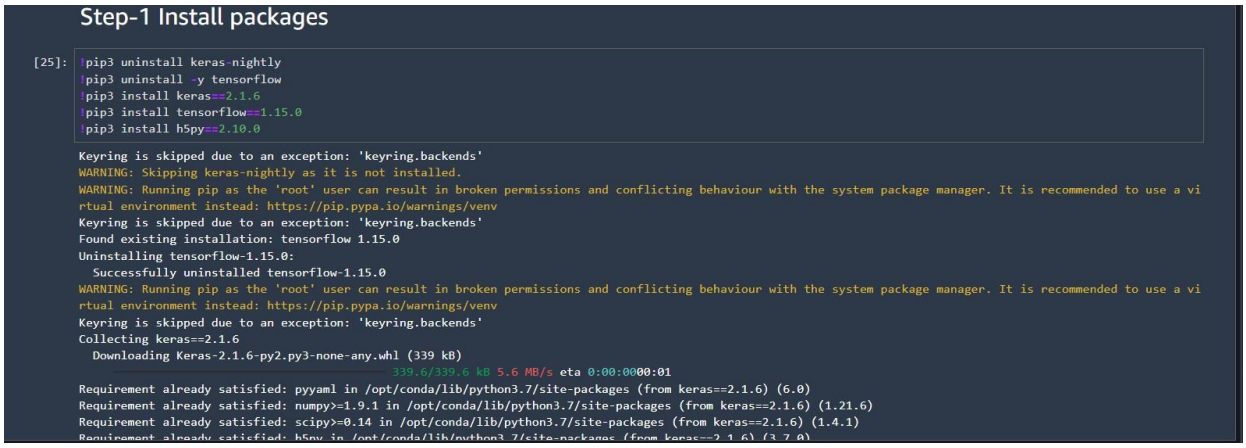
Implementing Sentiment Analysis using RNN Model

Recurrent neural networks (RNN) are the most fundamental and powerful neural networks. These algorithms have grown in prominence since they have generated promising results for a variety of technologies. The primary purpose of RNN is to efficiently handle sequential data. The concept of internal memory distinguishes RNN from traditional neural networks.

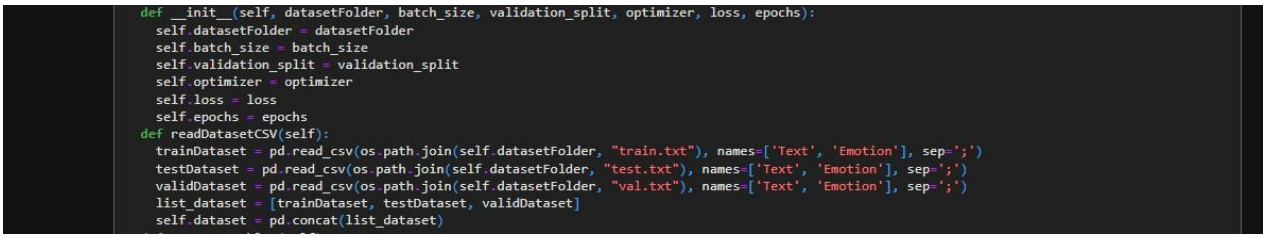
Recurrent neural networks are older than other forms of neural networks, having been around since the 1980s but just lately gained prominence. With the advent of technology, RNN has grabbed the lead since we now have more computer power and a large amount of recently obtained data.

For the implementation of this, we wrote a class called 'Emotions' comprised of following steps:

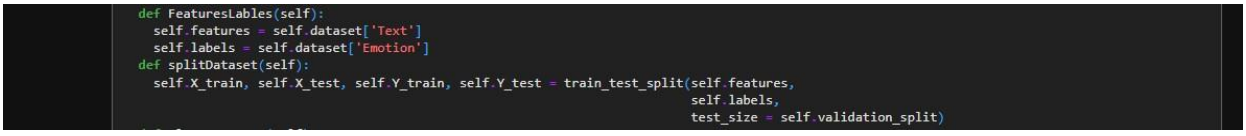
1. Installing Required packages.



2. Loading Dataset into Notebook



2. Feature Labeling & splitting dataset



3. Cleaning Dataset feature

- In our dataset, we had capital letters and camel-cased Alphabets; for sentiment analysis, we determined

that these were unnecessary, therefore we changed the data to lowercase.

- Remove any unnecessary spaces, punctuation, etc., completing the Data preparation stage.

```
                                                                test_size = self.validation_split)
def CleanFeatures(self):
  self.features = self.features.apply(lambda sequence:
                                      [ltrs.lower() for ltrs in sequence if ltrs not in string.punctuation])
  self.features = self.features.apply(lambda wrd: ''.join(wrd))
```

4. **Tokenizing textual input and creating Word Embedding using GloVe:**

   GloVe model is an unsupervised learning algorithm for obtaining vector representations forwords. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linearsubstructures of the word vector space

```
def tokenizerDataset(self):
  self.tokenizer = Tokenizer(num_words=5000)
  self.tokenizer.fit_on_texts(self.features)
  train = self.tokenizer.texts_to_sequences(self.features)
  self.features = pad_sequences(train)
  le = LabelEncoder()
  self.labels = le.fit_transform(self.labels)
  self.vocabulary = len(self.tokenizer.word_index)
def label_categorical(self):
  self.labels = to_categorical(self.labels, 6)
def glove_word_embedding(self, file_name):
  self.embeddings_index = {}
  file_ = open(file_name)
  for line in file_:
      arr = line.split()
      single_word = arr[0]
      w = np.asarray(arr[1:],dtype='float32')
      self.embeddings_index[single_word] = w
  file_.close()
  max_words = self.vocabulary + 1
  word_index = self.tokenizer.word_index
  self.embedding_matrix = np.zeros((max_words,300)).astype(object)
  for word , i in word_index.items():
          embedding_vector = self.embeddings_index.get(word)
          if embedding_vector is not None:
              self.embedding_matrix[i] = embedding_vector
```

5. Function definition using activation, dropouts, regularizers, measurements, and fit.

   -Here we defined the function-related dropouts, regularizers, and functions for compiling and fitting the model.

```
def model(self):
  m = Sequential()
  m.add(Input(shape=(self.features.shape[1], )))
  m.add(Embedding(self.vocabulary + 1,300))
  m.add(GRU(128, recurrent_dropout=0.3, return_sequences=False, activity_regularizer = tf.keras.regularizers.L2(0.0001)))
  m.add(Dense(6, activation="softmax", activity_regularizer = tf.keras.regularizers.L2(0.0001)))
  self.m = m
def compiler(self):
  self.m.compile(loss= self.loss,optimizer=self.optimizer,metrics=['accuracy'])
def fit(self):
  earlyStopping = EarlyStopping(monitor = 'loss', patience = 20, mode = 'min', restore_best_weights = True)
  self.history_training = self.m.fit(self.X_train, self.Y_train, epochs= self.epochs,batch_size = self.batch_size,
                                     callbacks=[ earlyStopping])
```

- This function is used to assemble and fit the model on training data..

6. Complete picture of Class Emotions

After defining all of the aforementioned functions, we chose to group them all into a class named "Emotion," from which an object would be formed to access all of these functions.

In [10]:

```
class Emotion:
  def __init__(self, datasetFolder, batch_size, validation_split, optimizer, loss, epochs):
    self.datasetFolder = datasetFolder
    self.batch_size = batch_size
    self.validation_split = validation_split
    self.optimizer = optimizer
    self.loss = loss
    self.epochs = epochs
  def readDatasetCSV(self):
    trainDataset = pd.read_csv(os.path.join(self.datasetFolder, "train.txt"), names=['Text', 'Emotion'], sep=';')
    testDataset = pd.read_csv(os.path.join(self.datasetFolder, "test.txt"), names=['Text', 'Emotion'], sep=';')
    validDataset = pd.read_csv(os.path.join(self.datasetFolder, "val.txt"), names=['Text', 'Emotion'], sep=';')
    list_dataset = [trainDataset, testDataset, validDataset]
    self.dataset = pd.concat(list_dataset)
  def FeaturesLables(self):
    self.features = self.dataset['Text']
    self.labels = self.dataset['Emotion']
  def splitDataset(self):
    self.X_train, self.X_test, self.Y_train, self.Y_test = train_test_split(self.features,
                                                                            self.labels,
                                                                            test_size = self.validation_split)
  def CleanFeatures(self):
    self.features = self.features.apply(lambda sequence:
                                        [ltrs.lower() for ltrs in sequence if ltrs not in string.punctuation])
    self.features = self.features.apply(lambda wrd: ''.join(wrd))
  def tokenizerDataset(self):
    self.tokenizer = Tokenizer(num_words=5000)
    self.tokenizer.fit_on_texts(self.features)
    train = self.tokenizer.texts_to_sequences(self.features)
    self.features = pad_sequences(train)
    le = LabelEncoder()
    self.labels = le.fit_transform(self.labels)
    self.vocabulary = len(self.tokenizer.word_index)
  def label_categorical(self):
    self.labels = to_categorical(self.labels, 6)
  def glove_word_embedding(self, file_name):
    self.embeddings_index = {}
    file_ = open(file_name)
    for line in file_:
        arr = line.split()
        single_word = arr[0]
        w = np.asarray(arr[1:],dtype='float32')
        self.embeddings_index[single_word] = w
    file_.close()
    max_words = self.vocabulary + 1
    word_index = self.tokenizer.word_index
    self.embedding_matrix = np.zeros((max_words,300)).astype(object)
    for word , i in word_index.items():
          embedding_vector = self.embeddings_index.get(word)
          if embedding_vector is not None:
              self.embedding_matrix[i] = embedding_vector
  def model(self):
    m = Sequential()
    m.add(Input(shape=(self.features.shape[1], )))
    m.add(Embedding(self.vocabulary + 1,300))
    m.add(GRU(128, recurrent_dropout=0.3, return_sequences=False, activity_regularizer = tf.keras.regularizers.L2(0.0001)))
    m.add(Dense(6, activation="softmax", activity_regularizer = tf.keras.regularizers.L2(0.0001)))
    self.m = m
  def compiler(self):
    self.m.compile(loss= self.loss,optimizer=self.optimizer,metrics=['accuracy'])
  def fit(self):
    earlyStopping = EarlyStopping(monitor = 'loss', patience = 20, mode = 'min', restore_best_weights = True)
    self.history_training = self.m.fit(self.X_train, self.Y_train, epochs= self.epochs,batch_size = self.batch_size,
                                       callbacks=[ earlyStopping])
```

7. **Result of DataSet Reading,** getting glimpse of Data using data.head()

```
In [25]:
emotion = Emotion(dataset_emotion, 256, 0.1, 'adam', 'categorical_crossentropy', 20)
emotion.readDatasetCSV()
emotion.dataset.head()
```

Out[25]:

	Text	Emotion
0	i didnt feel humiliated	sadness
1	i can go from feeling so hopeless to so damned...	sadness
2	im grabbing a minute to post i feel greedy wrong	anger
3	i am ever feeling nostalgic about the fireplac...	love
4	i am feeling grouchy	anger

8. Result of **Data Cleaning**, **feature labeling** (step of Data Preparation)

```
In [26]:
emotion.FeaturesLables()
emotion.CleanFeatures()
emotion.features.head()
```

```
Out[26]: 0                          i didnt feel humiliated
         1    i can go from feeling so hopeless to so damned...
         2     im grabbing a minute to post i feel greedy wrong
         3    i am ever feeling nostalgic about the fireplac...
         4                                i am feeling grouchy
         Name: Text, dtype: object
```

9. **Result of TokenizerDataSet function**

```
In [28]:
emotion.tokenizerDataset()
emotion.features
```

```
Out[28]: array([[   0,    0,    0, ...,  138,    2,  625],
                [   0,    0,    0, ...,    3,   21, 1383],
                [   0,    0,    0, ...,    2,  495,  420],
                ...,
                [   0,    0,    0, ...,    5,  215,  191],
                [   0,    0,    0, ...,   30,   57, 2181],
                [   0,    0,    0, ...,   75,    5,   70]], dtype=int32)
```

10. Summary of RNN neural network

```
In [35]:
emotion.compiler()
emotion.m.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 63, 300)           5129100

 gru_2 (GRU)                 (None, 128)               165120

 dense_2 (Dense)             (None, 6)                 774

=================================================================
Total params: 5,294,994
Trainable params: 165,894
Non-trainable params: 5,129,100
_________________________________________________________________
```

11. Training RNN Model

```
In [36]: emotion.fit()

Epoch 1/20
71/71 [==============================] - 39s 519ms/step - loss: 1.3828 - accuracy: 0.4860
Epoch 2/20
71/71 [==============================] - 35s 499ms/step - loss: 0.8182 - accuracy: 0.7173
Epoch 3/20
71/71 [==============================] - 38s 542ms/step - loss: 0.4484 - accuracy: 0.8485
Epoch 4/20
71/71 [==============================] - 38s 539ms/step - loss: 0.2896 - accuracy: 0.8990
Epoch 5/20
71/71 [==============================] - 38s 532ms/step - loss: 0.2210 - accuracy: 0.9172
Epoch 6/20
71/71 [==============================] - 38s 531ms/step - loss: 0.1796 - accuracy: 0.9279
Epoch 7/20
71/71 [==============================] - 36s 512ms/step - loss: 0.1543 - accuracy: 0.9361
Epoch 8/20
71/71 [==============================] - 35s 497ms/step - loss: 0.1355 - accuracy: 0.9435
Epoch 9/20
71/71 [==============================] - 36s 504ms/step - loss: 0.1205 - accuracy: 0.9496
Epoch 10/20
71/71 [==============================] - 35s 495ms/step - loss: 0.1127 - accuracy: 0.9524
Epoch 11/20
71/71 [==============================] - 35s 495ms/step - loss: 0.1017 - accuracy: 0.9588
Epoch 12/20
71/71 [==============================] - 35s 497ms/step - loss: 0.0951 - accuracy: 0.9601
Epoch 13/20
71/71 [==============================] - 35s 499ms/step - loss: 0.0885 - accuracy: 0.9634
Epoch 14/20
71/71 [==============================] - 35s 498ms/step - loss: 0.0830 - accuracy: 0.9666
Epoch 15/20
71/71 [==============================] - 35s 499ms/step - loss: 0.0746 - accuracy: 0.9698
Epoch 16/20
71/71 [==============================] - 36s 500ms/step - loss: 0.0752 - accuracy: 0.9694
Epoch 17/20
71/71 [==============================] - 36s 505ms/step - loss: 0.0660 - accuracy: 0.9753
Epoch 18/20
71/71 [==============================] - 36s 500ms/step - loss: 0.0626 - accuracy: 0.9754
Epoch 19/20
71/71 [==============================] - 35s 498ms/step - loss: 0.0587 - accuracy: 0.9783
Epoch 20/20
71/71 [==============================] - 35s 496ms/step - loss: 0.0577 - accuracy: 0.9779
```

12. Evaluation of Model:

a) Accuracy during Training

```
[38]: import matplotlib.pyplot as plt
      mpl.style.use('seaborn')
      figure = plt.figure(figsize=(15, 4))
      plt.plot(emotion.history_training.history['accuracy'], 'darkorange', label = 'Accuracy')
      plt.title("Accuracywhile training")
      plt.show()
```

b) Plotting Loss Function During Train

```
[39]: figure = plt.figure(figsize=(15, 4))
      plt.plot(emotion.history_training.history['loss'], 'darkblue', label = 'Loss')
      plt.title("Loss while training")
      plt.show()
```

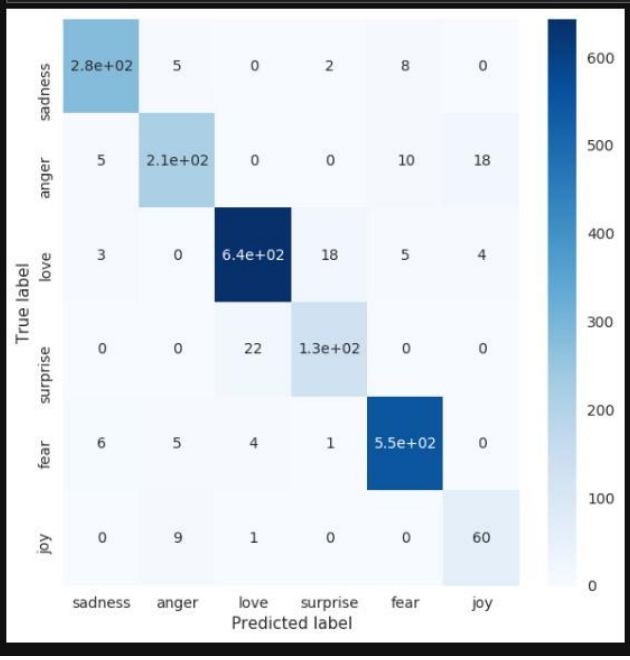

```
[40]: emotion.m.evaluate(emotion.X_test, emotion.Y_test, batch_size = 256)
```

c) Confusion Matrix

d) Predicting on Test Data

```
[41]: y_pred = emotion.m.predict(emotion.X_test)
      y_pred = np.argmax(y_pred, axis = 1)
```

```
In [45]: import seaborn as sns
         figure = plt.figure(figsize=(7, 7))
         sns.heatmap(cm, annot=True, cmap=plt.cm.Blues)
         plt.ylabel('True label')
         plt.xlabel('Predicted label')
         plt.show()
```



e) Heatmap Visualization of Predicted labels

13. Comparison of Performance on different Clusters:

Because RNN is such a powerful model, we were getting results on "t3-medium" cluster instances (3 node) within seconds for the same dataset, so we kept it.

As a result, RNN is an excellent model for implementing sentiment analysis in NLP problems.

2. Implementing ML Model using Spark MLlib

Model selection, or using data to determine the optimum model or parameters for a particular job, is an essential topic in ML. This is also known as tuning. Individual Estimators, such as LogisticRegression, may be tuned, as can complete Pipelines that comprise numerous algorithms, featurization, and other processes. Instead of tweaking each element in the Pipeline individually, users may tune the entire Pipeline at once.
Model selection is supported by MLlib via functions such as CrossValidator and TrainValidationSplit. The following elements are required for these tools:

• Estimator: a tuning method or pipeline

• A set of ParamMaps: a set of parameters to pick from, commonly referred to as a "parameter grid" to search through.

• Evaluator: statistic used to assess how well a fitted Model performs on held-out test data.

These model selection tools function on a high level as follows:

• They separated the input data into training and test datasets.

• They cycle over the collection of ParamMaps for each (training, test) pair: for each ParamMap, they fit the Estimator using those parameters, acquire the fitted Model, and assess the Model's performance using the Evaluator.

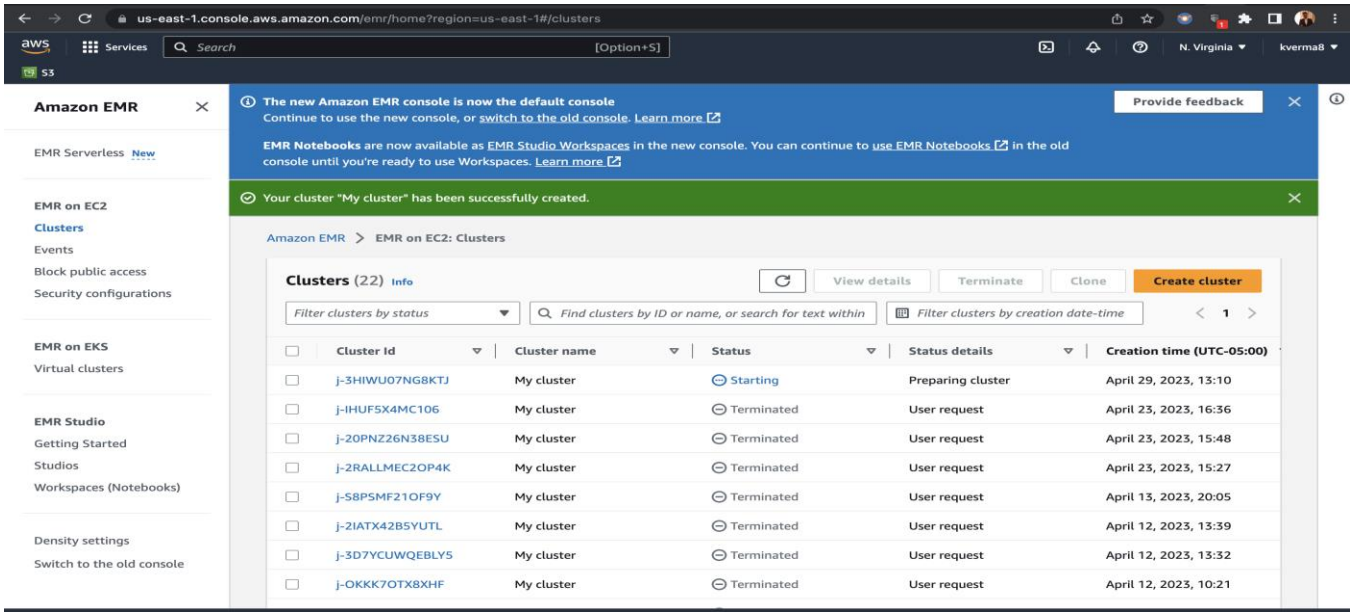
• They choose the Model resulting from the best-performing set of parameters.

For regression issues, the Evaluator can be a Regression-Evaluator, for binary data, a Binary-Classification-Evaluator, for binary data, a Multiclass-Classification-Evaluator, for multiclass problems, a Multilabel Classification-Evaluator, for multi-label classifications, or a Ranking-Evaluator, for ranking problems.
The set-Metric-Name method in each of these evaluations can override the default metric used to select the best ParamMap.

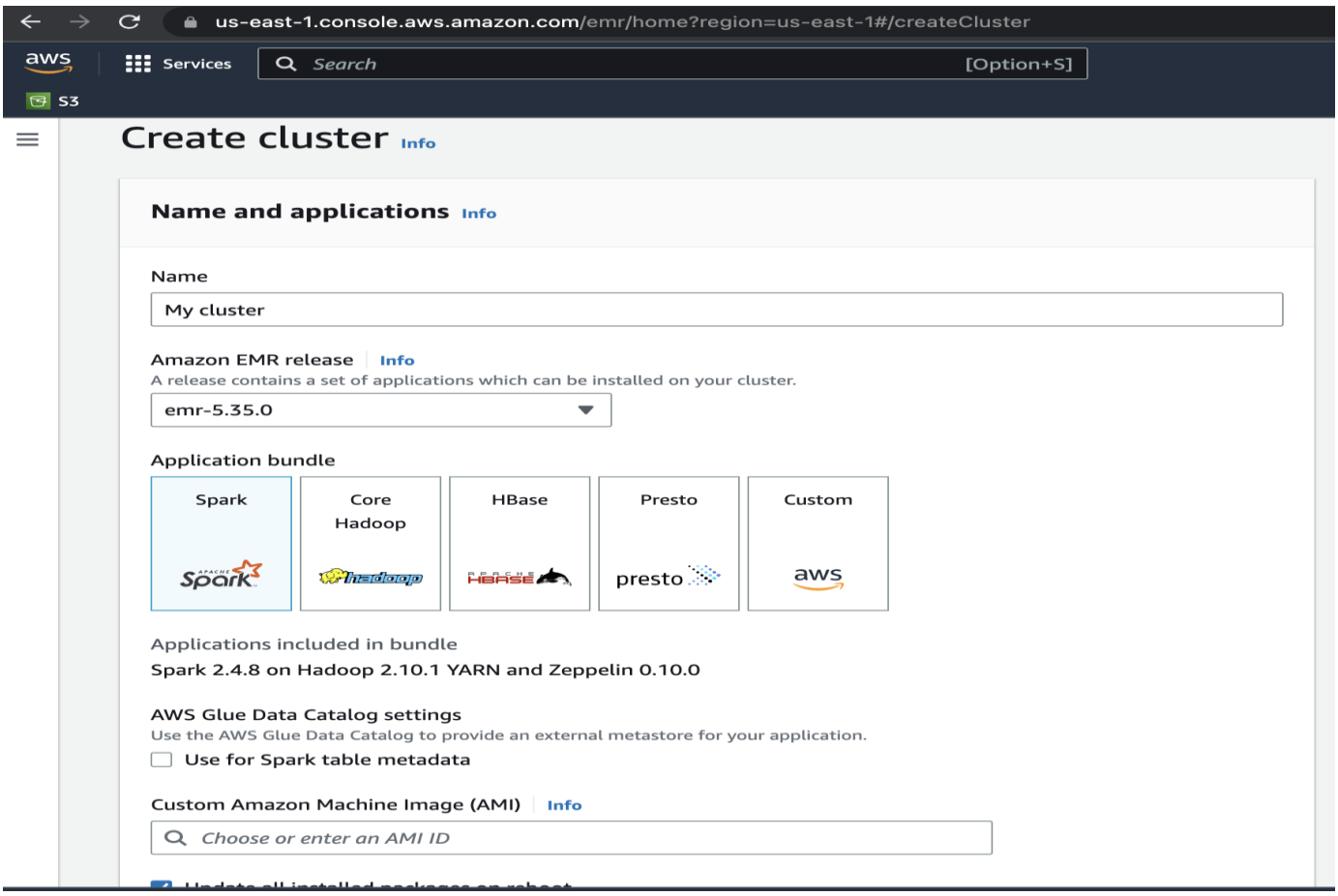
Below we have implemented this part in the following steps -

There are several cloud providers to select from when analyzing spark R services, including AWS, Azure, Cloudera, data bricks, Google Cloud, and others. To execute, we select AWS EMR with Spark.

1. **Create an EMR Cluster on your AWS Account.**

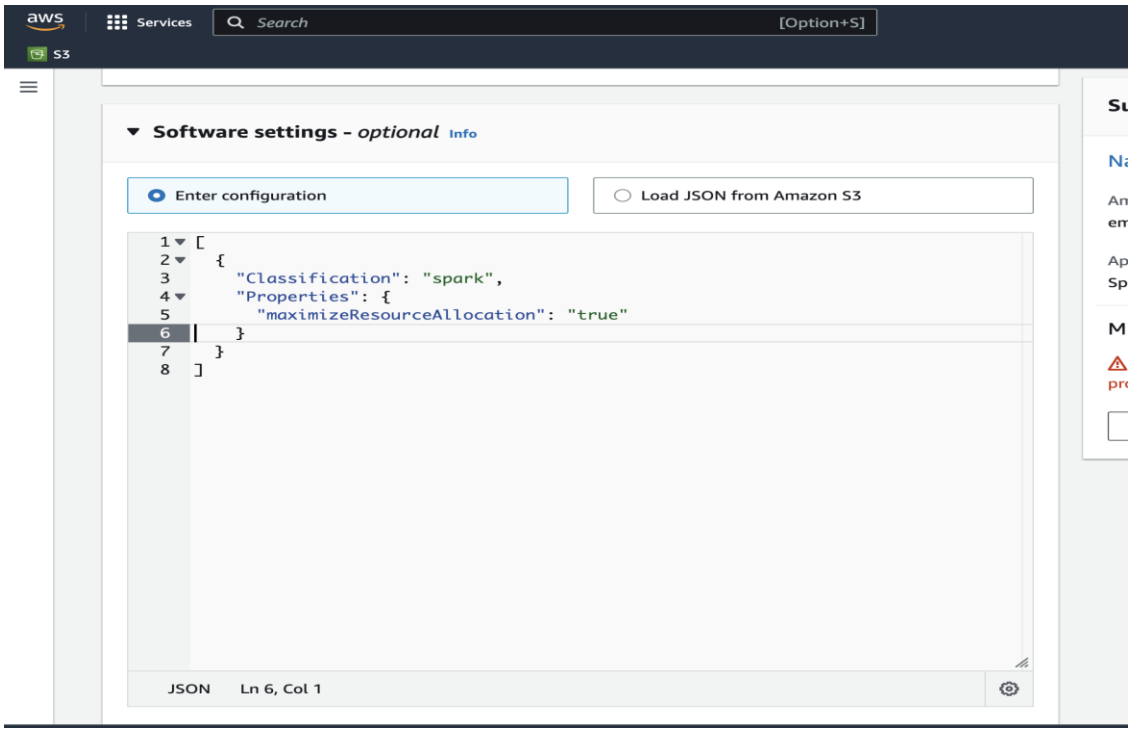


2. Go to advanced options

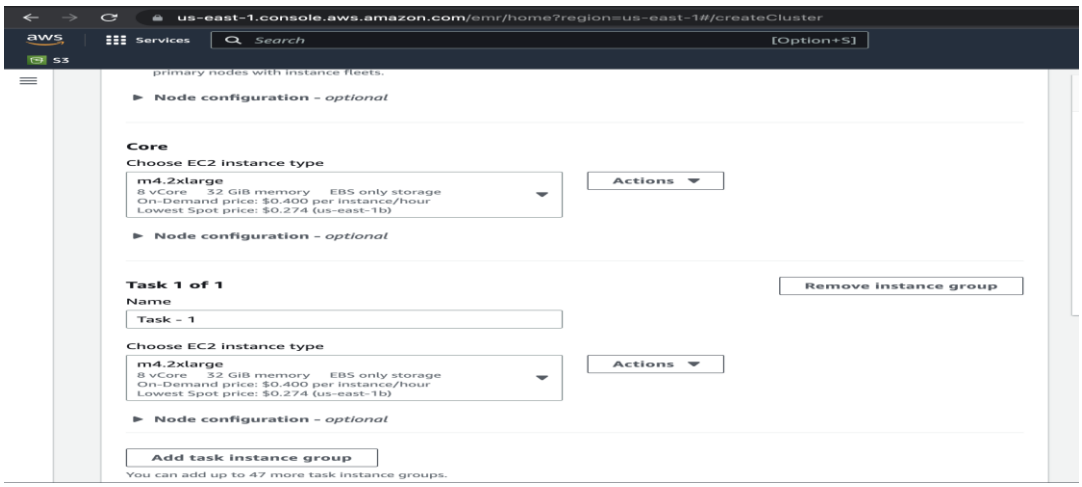


3. **Maximize resource allocation for your current EMR cluster using the below configuration**

```
{
        "Classification": "spark",
        "Properties": {
                "maximizeResourceAllocation": "true"
                }
}
```



4. **In the second step, configure the Hardware as below**



5. **Go to third step, to configure general cluster settings so here we configure bootstrap action**.

    a. Next go to amazon blog -

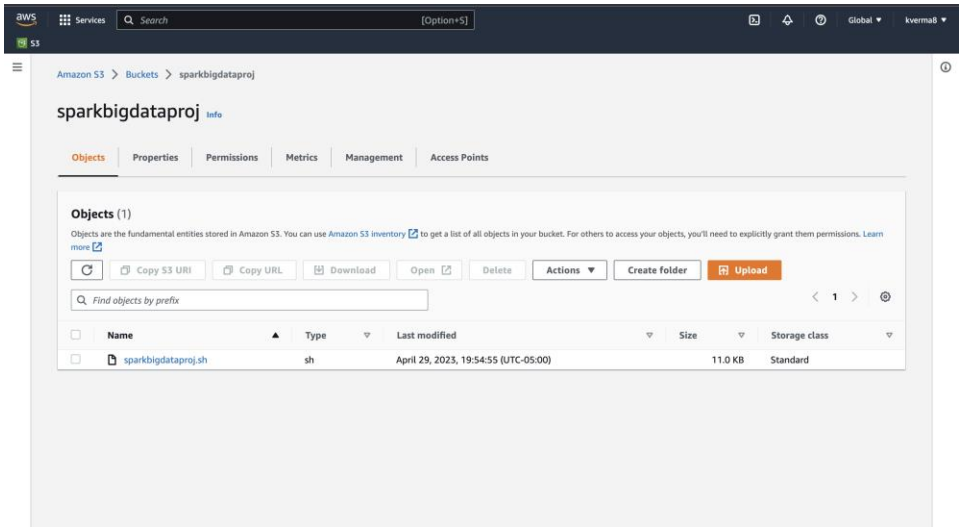
https://aws.amazon.com/blogs/big-data/running-sparklyr-rstudios-r-interface-to-spark-on-amazon-emr/

    b. and edit the shell script as below and save it as a .sh file in R:

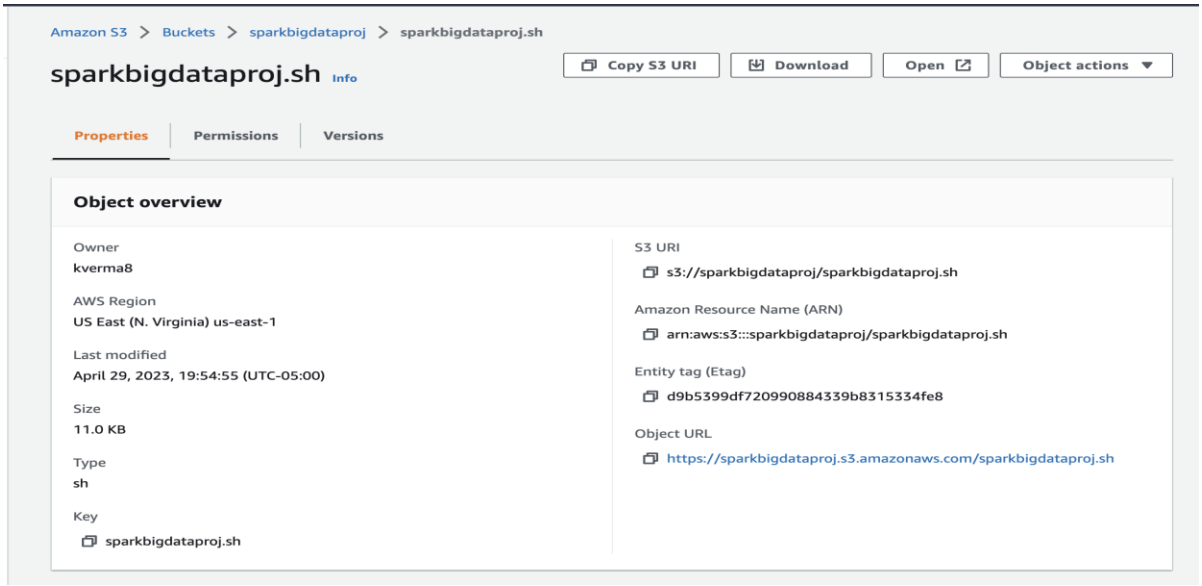
https://aws-bigdata-blog.s3.amazonaws.com/artifacts/aws-blog-emr-rstudio-sparklyr/rstudio_sparklyr_emr5.sh

    c. Go to aws and open S3, Create a bucket - sparkbigdataproj

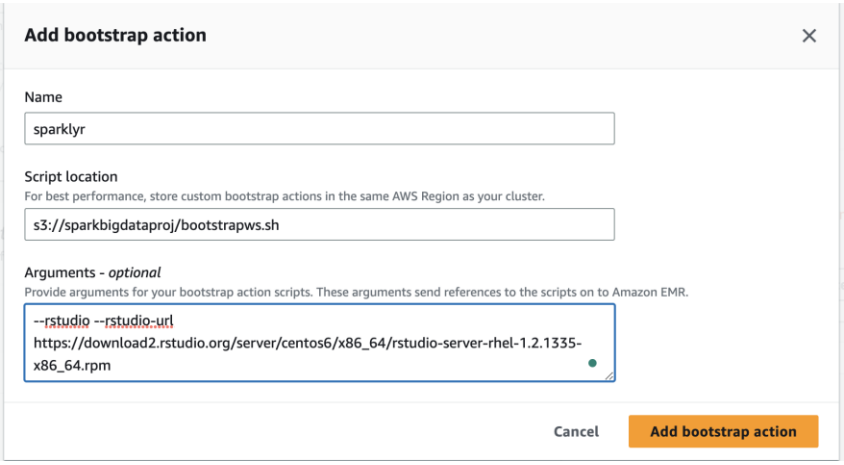
6. **Upload the sh file into it**



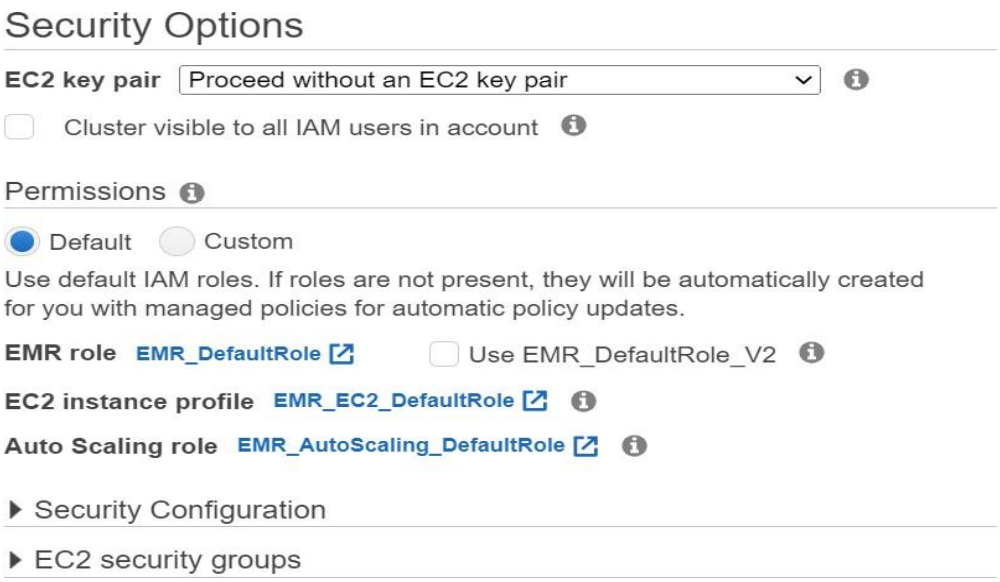
7. **Open the object and Copy the S3 URI Path**



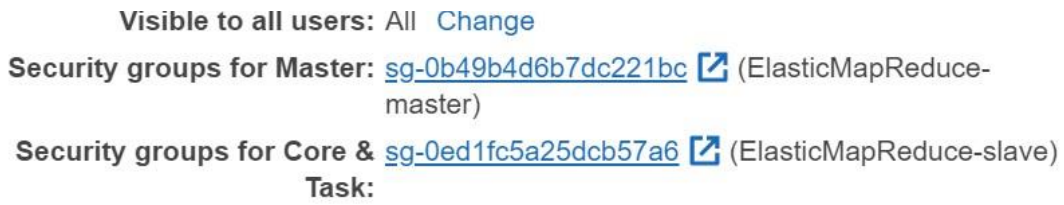
8. **Choose Custom Action then Configure and add the URI Path in emr cluster.**

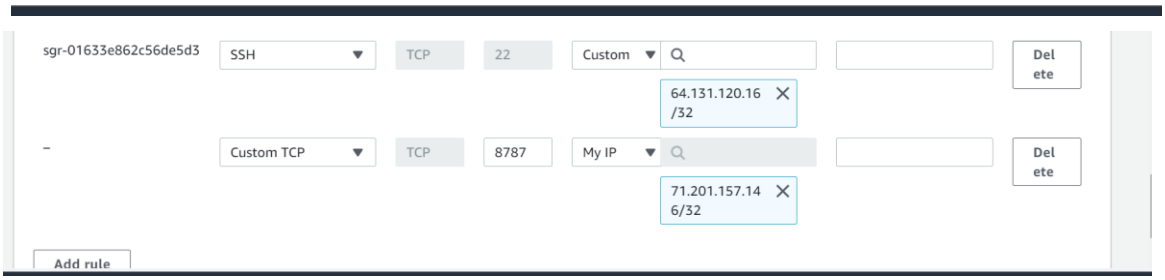


9. **Go to last step, Security unselect the cluster visible option and create a cluster**

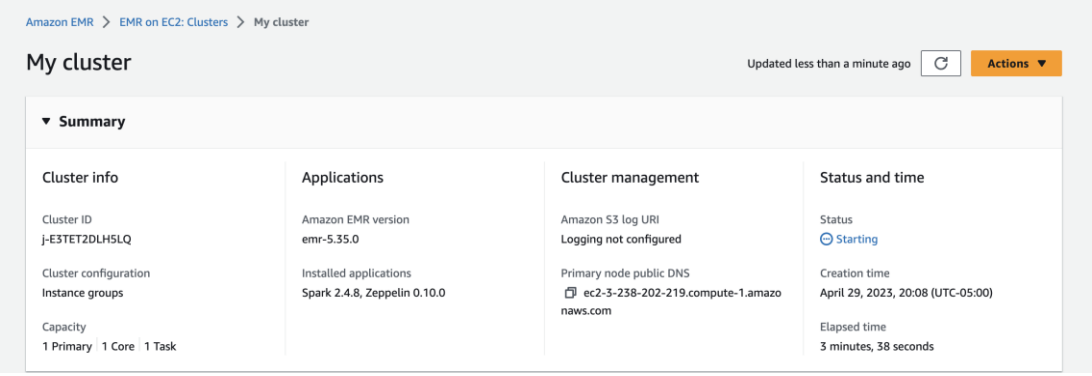


10. **We will wait cluster to start and then configure the security groups for Master and add the edit inbound rules as below:**

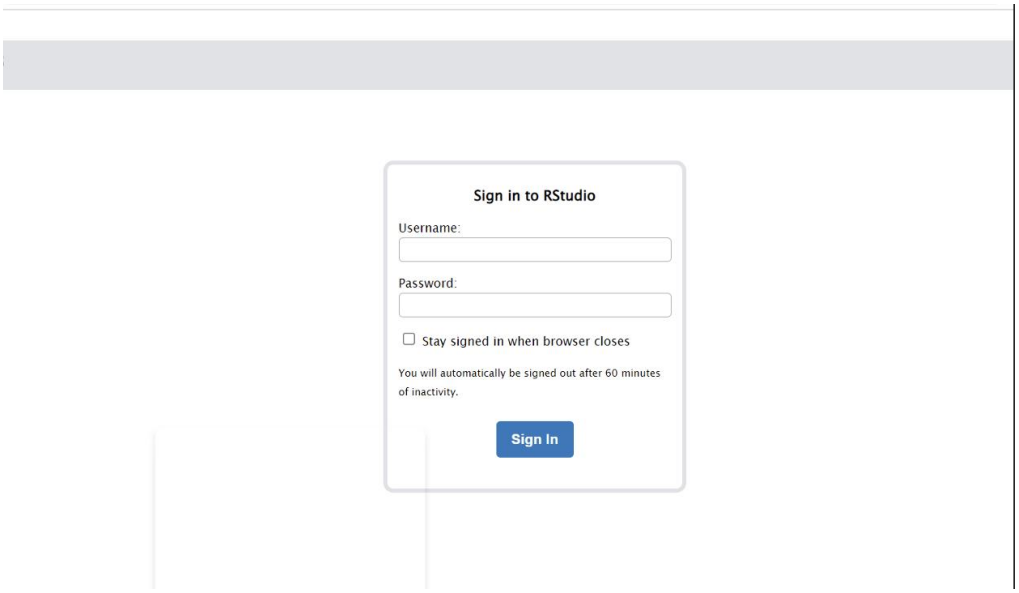




11. **Then, create a cluster, after sometime the cluster status changes to starting.**



12. **Add the 8787 port to the url - http://ec2-3-238-202-219.compute-1.amazonaws.com:8787/ and open it in a new tab- you will be able to see the login page of R studio**.



Now we are ready to implement the Machine Learning Model using the Spark-R configuration we just set up.

What are we implementing in Spark-MLlib

1. Determining Sentiment using **Random Forest**.
2. Determining Sentiment using **Logistic Regression**

```
> install.packages("dplyr")
also installing the dependencies 'glue', 'cli', 'rlang', 'vctrs'

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/glue_1.6.2.tgz'
Content type 'application/x-gzip' length 155475 bytes (151 KB)
==================================================
downloaded 151 KB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/cli_3.3.0.tgz'
Content type 'application/x-gzip' length 1171398 bytes (1.1 MB)
==================================================
downloaded 1.1 MB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/rlang_1.0.2.tgz'
Content type 'application/x-gzip' length 1822104 bytes (1.7 MB)
==================================================
downloaded 1.7 MB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/vctrs_0.4.1.tgz'
Content type 'application/x-gzip' length 1761486 bytes (1.7 MB)
==================================================
downloaded 1.7 MB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/dplyr_1.0.9.tgz'
Content type 'application/x-gzip' length 1327720 bytes (1.3 MB)
==================================================
downloaded 1.3 MB


The downloaded binary packages are in
        /var/folders/d_/23y7m6ns5kg1n8jjr5kh1cbc0000gn/T//Rtmp0wXCK8/downloaded_packages
>
```

```
> install.packages("sparklyr")
also installing the dependencies 'htmlwidgets', 'config', 'forge', 'r2d3', 'rprojroot', 'tidyr'

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/htmlwidgets_1.5.4.tgz'
Content type 'application/x-gzip' length 898177 bytes (877 KB)
==================================================
downloaded 877 KB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/config_0.3.1.tgz'
Content type 'application/x-gzip' length 78547 bytes (76 KB)
==================================================
downloaded 76 KB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/forge_0.2.0.tgz'
Content type 'application/x-gzip' length 38522 bytes (37 KB)
==================================================
downloaded 37 KB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/r2d3_0.2.6.tgz'
Content type 'application/x-gzip' length 1905942 bytes (1.8 MB)
==================================================
downloaded 1.8 MB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/rprojroot_2.0.3.tgz'
Content type 'application/x-gzip' length 100267 bytes (97 KB)
==================================================
downloaded 97 KB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/tidyr_1.2.0.tgz'
Content type 'application/x-gzip' length 1022523 bytes (998 KB)
==================================================
downloaded 998 KB

trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/sparklyr_1.7.5.tgz'
Content type 'application/x-gzip' length 5785886 bytes (5.5 MB)
==================================================
downloaded 5.5 MB
```

13. Installed the required packages

```
The downloaded binary packages are in
        /var/folders/d_/23y7m6ns5kg1n8jjr5kh1cbc0000gn/T//Rtmp0wXCK8/downloaded_packages
> install.packages("data.table")
trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/data.table_1.14.2.tgz'
Content type 'application/x-gzip' length 2354815 bytes (2.2 MB)
==================================================
downloaded 2.2 MB


The downloaded binary packages are in
        /var/folders/d_/23y7m6ns5kg1n8jjr5kh1cbc0000gn/T//Rtmp0wXCK8/downloaded_packages
> install.packages("ggplot2")
trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-arm64/contrib/4.1/ggplot2_3.3.6.tgz'
Content type 'application/x-gzip' length 4122229 bytes (3.9 MB)
==================================================
downloaded 3.9 MB


The downloaded binary packages are in
        /var/folders/d_/23y7m6ns5kg1n8jjr5kh1cbc0000gn/T//Rtmp0wXCK8/downloaded_packages
> |
```

We are now implementing the real section once we have set up the environment. Importing the required libraries. A variety of Python tools and frameworks will be used. We will read the contents of a text file and output the contents as a list of single words using the function readFile(filename). We will read all CSV files using this function, where each row represents a text file and the columns hold the counts of each word in that text file. Read the CSV files and show the data (training, testing, and validation).

1. Training Data Image

```
> glimpse(df_train)
Rows: 170,883
Columns: 31
$ Time   <dbl> 51742, 61413, 74745, 144191, 77866, 138901, 28314, 119597, 161002, 121976, 42390, 123162, 48874, 21503, 140347, 169847, 39258, 40772, 55256, 143400, 4076, 1…
$ V1     <dbl> 1.21777027, 0.66529375, 1.27217637, -7.72533613, -0.51741575, 2.14992728, -0.51959247, -0.69251468, -0.69582680, 1.95734543, -5.45335867, -0.23939182, -1.63…
$ V2     <dbl> 0.077243995, -0.779358247, 0.169924869, -2.354525834, 0.313685579, -1.955008075, 0.887040653, 1.074982165, -0.292627271, 0.110680440, -0.231388334, -0.23551…
$ V3     <dbl> 0.498320921, 1.680699937, -0.179971299, -4.607707356, 1.537755598, -0.511217981, 1.440903939, -0.214693143, 1.243020585, -1.632382884, -1.507947139, -1.6858…
$ V4     <dbl> 0.52081540, 3.15837257, 1.06222072, 0.16461745, -1.55839611, -1.85509676, 1.33028614, -0.48961654, -1.96064228, 1.42177447, -1.05100263, 1.02082718, -1.0151…
$ V5     <dbl> -0.57493216, -1.44556215, 0.52113991, -6.98564210, 0.70002786, -1.40519345, 0.28485258, 1.19239059, -1.01894794, 0.38608355, -0.88952104, 1.19676810, 1.7418…
$ V6     <dbl> -0.70860220, 0.72112018, 0.61467613, 5.63762828, 1.28801100, 0.88406274, -0.01240266, -0.51072267, -0.05348007, -1.13756538, -2.14250592, -2.85250088, 0.510…
$ V7     <dbl> -0.15910140, -0.73334910, 0.01776800, 5.31388329, 0.27189790, -1.99249895, 0.54990329, 1.17717094, -0.92444059, 0.69048453, -0.69071343, 0.83313809, -0.1646…
$ V8     <dbl> 0.03974627, 0.35181687, 0.08778932, 0.66060851, 0.50785636, 0.35322381, 0.11750182, -0.37740411, 0.50319274, -0.42686523, 1.62623071, -0.37323827, 0.5443738…
$ V9     <dbl> 0.04662027, 0.94769094, 0.41939825, 0.06467695, 0.03874012, -0.67957025, -0.28542418, 0.44992879, -2.38251375, 0.20385026, 0.28120308, -0.24838164, -0.31510…
$ V10    <dbl> 0.15416414, 0.24338193, -0.11428525, -1.27791475, -0.46023339, 1.65873457, -0.09904113, -0.71702185, 0.65392202, 0.24216188, -1.88736834, -1.46248628, -0.57…
$ V11    <dbl> 1.18739570, -1.65843187, -1.82494214, 0.51420147, 2.21283228, -0.39765542, -0.62739817, -0.77397431, -1.32835501, -0.97896262, -0.95992515, 0.81502624, 2.13…
$ V12    <dbl> 0.14041972, -0.19943952, -0.04508140, 0.56761746, 0.73287217, -0.91132998, 0.15465282, -0.46458157, -1.48685407, 0.30331634, 2.05764194, -0.74798787, 0.0202…
$ V13    <dbl> -1.49365270, -1.25173693, -0.09806199, 0.59535051, -0.61883494, -0.20927264, -0.16824800, -0.30603555, -0.07741579, -0.29891972, 1.58922066, -1.07850222, -1…
$ V14    <dbl> 0.778755068, -0.703287023, 0.022784387, 0.737137725, 0.113775958, -0.542535017, 0.026868926, -2.148160000, -0.144727134, 0.550597636, 1.247583973, -2.677376…
$ V15    <dbl> 0.52057694, -1.44303735, 0.03103779, 1.40786805, 0.98407363, -0.07748705, 0.22485432, -0.69270992, 0.44892596, -0.50939667, 0.07651487, 0.60025407, 1.119257…
$ V16    <dbl> 0.58043838, 0.23800276, -0.30547969, 1.83604611, -0.48070189, 0.53662657, -1.22143933, -0.03052407, -0.78638800, -0.73258439, -0.21621394, 0.38071848, -0.78…
$ V17    <dbl> -0.57581565, 0.08258451, -0.23349454, -0.18545377, 0.18041636, -0.30796733, 0.63322382, 1.00386814, 1.08161413, -0.12311094, 0.61404035, 2.79859005, 0.53151…
$ V18    <dbl> 0.164915866, -0.091704277, -0.496386172, -0.439953804, -1.792958946, 1.278902925, -0.670814150, 0.278332173, 0.385911950, -0.647856510, -0.287374850, 0.9238…
$ V19    <dbl> 0.183083986, -0.587018607, 0.162940224, -0.642606025, -2.015641436, 0.175551053, 0.722868501, -0.227393384, 0.882431905, -0.235162987, 0.323835153, -0.35551…
$ V20    <dbl> -0.194238948, 0.173779642, -0.167845299, -5.028522602, -0.062245780, -0.292541879, 0.186150070, -0.021684075, -0.208347522, -0.229382732, -0.967483857, 0.47…
$ V21    <dbl> -0.226153731, 0.046142968, -0.199115064, -1.073900540, 0.010926041, -0.091000411, -0.019202261, -0.022695751, -0.087784279, 0.071818891, -0.117051821, 0.171…
$ V22    <dbl> -0.77301817, -0.01765075, -0.32912090, 1.17360022, 0.33847836, 0.02869474, 0.26010819, 0.42127313, -0.04729308, 0.35135162, 0.12359577, 0.04862732, 0.594241…
$ V23    <dbl> 0.145242554, -0.209463297, -0.228424917, -0.183291096, 0.066528800, 0.204367543, -0.106747404, -0.165215854, -0.236493437, -0.033844403, -0.579949776, 0.606…
$ V24    <dbl> 0.276868331, 0.385114789, -1.313536521, -0.584830819, -0.989922227, -0.412525548, 0.096712750, 0.530683751, 0.654293114, -0.006930706, 0.569689850, 0.665258…
$ V25    <dbl> 0.129511016, 0.296199660, 0.844640504, 0.008990326, -0.638208162, -0.503210266, -0.081635853, -0.191540072, 0.384780489, 0.443082256, 1.236152967, -0.528511…
$ V26    <dbl> 0.083098376, 0.082171404, -0.224776172, 0.618217507, 0.806935439, -0.217686199, -0.223809540, 0.500289835, 0.103345352, -0.493266120, -0.604587731, 0.640381…
$ V27    <dbl> -0.04399679, 0.02549195, 0.03753611, -0.57952974, 0.07574998, 0.04153347, 0.42188220, -0.14282286, -0.04176558, -0.02190505, 0.38435179, -0.13437705, -0.221…
$ V28    <dbl> 0.0049825747, 0.0710640262, 0.0005805126, -0.6069469475, -0.1814167096, -0.0347870456, 0.2076466615, 0.0168861053, -0.0229249781, -0.0556870970, -1.13190491…
$ Amount <dbl> 0.99, 214.04, 3.76, 1395.00, 7.68, 73.50, 17.18, 14.99, 15.00, 45.50, 7.50, 145.00, 28.75, 14.19, 22.08, 0.99, 830.78, 2.99, 1.46, 9.39, 9.60, 56.00, 19.99,…
```

2. Validation Data glimpse(df_val)

```
> glimpse(df_val)
Rows: 170,883
Columns: 31
$ Time   <dbl> 51742, 61413, 74745, 144191, 77866, 138901, 28314, 119597, 161002, 121976, 42390, 123162, 48874, 21503, 140347, 169847, 39258, 40772, 55256, 143400, 4076, 1…
$ V1     <dbl> 1.21777027, 0.66529375, 1.27217637, -7.72533613, -0.51741575, 2.14992728, -0.51959247, -0.69251468, -0.69582680, 1.95734543, -5.45335867, -0.23939182, -1.63…
$ V2     <dbl> 0.077243995, -0.779358247, 0.169924869, -2.354525834, 0.313685579, -1.955008075, 0.887040653, 1.074982165, -0.292627271, 0.110680440, -0.231388334, -0.23551…
$ V3     <dbl> 0.498320921, 1.680699937, -0.179971299, -4.607707356, 1.537755598, -0.511217981, 1.440903939, -0.214693143, 1.243020585, -1.632382884, -1.507947139, -1.6858…
$ V4     <dbl> 0.52081540, 3.15837257, 1.06222072, 0.16461745, -1.55839611, -1.85509676, 1.33028614, -0.48961654, -1.96064228, 1.42177447, -1.05100263, 1.02082718, -1.0151…
$ V5     <dbl> -0.57493216, -1.44556215, 0.52113991, -6.98564210, 0.70002786, -1.40519345, 0.28485258, 1.19239059, -1.01894794, 0.38608355, -0.88952104, 1.19676810, 1.7418…
$ V6     <dbl> -0.70860220, 0.72112018, 0.61467613, 5.63762828, 1.28801100, 0.88406274, -0.01240266, -0.51072267, -0.05348007, -1.13756538, -2.14250592, -2.85250088, 0.510…
$ V7     <dbl> -0.15910140, -0.73334910, 0.01776800, 5.31388329, 0.27189790, -1.99249895, 0.54990329, 1.17717094, -0.92444059, 0.69048453, -0.69071343, 0.83313809, -0.1646…
$ V8     <dbl> 0.03974627, 0.35181687, 0.08778932, 0.66060851, 0.50785636, 0.35322381, 0.11750182, -0.37740411, 0.50319274, -0.42686523, 1.62623071, -0.37323827, 0.5443738…
$ V9     <dbl> 0.04662027, 0.94769094, 0.41939825, 0.06467695, 0.03874012, -0.67957025, -0.28542418, 0.44992879, -2.38251375, 0.20385026, 0.28120308, -0.24838164, -0.31510…
$ V10    <dbl> 0.15416414, 0.24338193, -0.11428525, -1.27791475, -0.46023339, 1.65873457, -0.09904113, -0.71702185, 0.65392202, 0.24216188, -1.88736834, -1.46248628, -0.57…
$ V11    <dbl> 1.18739570, -1.65843187, -1.82494214, 0.51420147, 2.21283228, -0.39765542, -0.62739817, -0.77397431, -1.32835501, -0.97896262, -0.95992515, 0.81502624, 2.13…
$ V12    <dbl> 0.14041972, -0.19943952, -0.04508140, 0.56761746, 0.73287217, -0.91132998, 0.15465282, -0.46458157, -1.48685407, 0.30331634, 2.05764194, -0.74798787, 0.0202…
$ V13    <dbl> -1.49365270, -1.25173693, -0.09806199, 0.59535051, -0.61883494, -0.20927264, -0.16824800, -0.30603555, -0.07741579, -0.29891972, 1.58922066, -1.07850222, -1…
$ V14    <dbl> 0.77875568, -0.70328023, 0.022784387, 0.737137725, 0.113775958, -0.542535017, 0.026868926, -2.148160000, -0.144727134, 0.550597636, 1.247583973, -2.677376…
$ V15    <dbl> 0.52057694, -1.44303735, 0.03103779, 1.40786805, 0.98407363, -0.07748705, 0.22485432, -0.69270992, 0.44892596, -0.50939667, 0.07651487, 0.60025407, 1.119257…
$ V16    <dbl> 0.58043838, 0.23800276, -0.30547969, 1.83604611, -0.48070189, 0.53662657, -1.22143933, -0.03052407, -0.78638800, -0.73258439, -0.21621394, 0.38071848, -0.78…
$ V17    <dbl> -0.57581565, 0.08258451, -0.23349454, -0.18545377, 0.18041636, -0.30796733, 0.63322382, 1.00386814, 1.08161413, -0.12311094, 0.61404035, 2.79859005, 0.53151…
$ V18    <dbl> 0.164915866, -0.091704277, -0.496386172, -0.439953804, -1.792958946, 1.278902925, -0.670814150, 0.278332173, 0.385911950, -0.647856510, -0.287374850, 0.9238…
$ V19    <dbl> 0.183083986, -0.587018607, 0.162940224, -0.642606025, -2.015641436, 0.175551053, 0.722868501, -0.227393384, 0.882431905, -0.235162987, 0.323835153, -0.35551…
$ V20    <dbl> -0.194238948, 0.173779642, -0.167845299, -5.028552602, -0.062245780, -0.292541879, 0.186150070, -0.021684075, -0.208347522, -0.229382732, -0.967483857, 0.47…
$ V21    <dbl> -0.226153731, 0.046142968, -0.199115064, -1.073900540, 0.010926041, -0.091000411, -0.019202261, -0.022695751, -0.087784279, 0.071818891, -0.117051821, 0.171…
$ V22    <dbl> -0.77301817, -0.01765075, -0.32912090, 1.17360022, 0.33847836, 0.02869474, 0.26010819, 0.42127313, -0.04729308, 0.35135162, 0.12359577, 0.04862732, 0.594241…
$ V23    <dbl> 0.145242554, -0.209463297, -0.228424917, -0.183291096, 0.066528800, 0.204367543, -0.106747404, -0.165215854, -0.236493437, -0.033844403, -0.579949776, 0.606…
$ V24    <dbl> 0.276868331, 0.385114789, -1.313536521, -0.584830819, -0.989922227, -0.412525548, 0.096712750, 0.530683751, 0.654293114, -0.000930706, 0.569689850, 0.665258…
$ V25    <dbl> 0.129511016, 0.296199660, 0.844640504, 0.008990326, -0.638208162, -0.503210266, -0.081635853, -0.191540072, 0.384780489, 0.443082256, 1.236152967, -0.528511…
$ V26    <dbl> 0.083098376, 0.082171404, -0.224776172, 0.618217507, 0.806935439, -0.217686199, -0.223809540, 0.500289835, 0.103345352, -0.493266120, -0.604587731, 0.640381…
$ V27    <dbl> -0.04399679, 0.02549195, 0.03753611, -0.57952974, 0.07574998, 0.04153347, 0.42188220, -0.14282286, -0.04176558, -0.02190505, 0.38435179, -0.13437705, -0.221…
$ V28    <dbl> 0.0049825747, 0.0710640262, 0.0005805126, -0.6069469475, -0.1814167096, -0.0347870456, 0.2076466615, 0.0168861053, -0.0229249781, -0.0556870970, -1.13190491…
$ Amount <dbl> 0.99, 214.04, 3.76, 1395.00, 7.68, 73.50, 17.18, 14.99, 15.00, 45.50, 7.50, 145.00, 28.75, 14.19, 22.08, 0.99, 830.78, 2.99, 1.46, 9.39, 9.60, 56.00, 19.99,…
$ Class  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
>
```

3. Viewing Glimpse of Dataset and Preprocessin

```
+    select(Class) %>%
+    group_by(Class) %>%
+    summarise(count = n()) %>%
+    glimpse
Rows: 2
Columns: 2
$ Class <fct> 0, 1
$ count <int> 170588, 295
```

4. Implement logistic regression in Spark-R

To categorize the mood of the reviews, we now train a basic logistic regression model. If you previously included the filename in the Dataframe, make sure you don't utilize it as a feature. Examine the accuracy of this simple model on the training and validation datasets. Are you overtraining? To reduce overfitting, experiment with altering the logistic regression parameters, such as adding a regularization term.

```
install.packages("dplyr")
df_train <- read.csv("/usr/mounika/Desktop/CSP_Project/Dataset/
train.csv")
df_test <- read.csv("/usr/mounika/Desktop/CSP_Project/Dataset/train.csv")
df_val <- read.csv("/usr/mounika/Desktop/CSP_Project/Dataset/train.csv")

spark_write_parquet(df_train, path="/user/rstudio-user",
mode="overwrite")
spark_write_parquet(df_test, path="/user/rstudio-user", mode="overwrite")
spark_write_parquet(df_val, path="/user/rstudio-user", mode="overwrite")

dim(df_train)
dim(df_val)
dim(df_test)

df_train$Class <- factor(df_train$Class)

train  %>%
  select(Class) %>%
  group_by(Class) %>%
  summarise(count = n()) %>%
  glimpse

test %>%
  select(Class) %>%
  group_by(Class) %>%
  summarise(count = n()) %>%
  glimpse

#Logistic Regression model
Logistic_Model=glm(Class~.,test_data,family=binomial())
summary(Logistic_Model)
```

Logistic Regression output-1

```
> summary(Logistic_Model)

Call:
glm(formula = Class ~ ., family = binomial(), data = df_test)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.7005  -0.0290  -0.0177  -0.0101   4.1608

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -8.525e+00  3.399e-01 -25.080  < 2e-16 ***
Time        -4.941e-06  2.860e-06  -1.727 0.084110 .
V1           1.397e-01  5.491e-02   2.544 0.010966 *
V2           5.574e-02  8.832e-02   0.631 0.527959
V3          -9.386e-02  6.724e-02  -1.396 0.162740
V4           7.831e-01  1.153e-01   6.793 1.10e-11 ***
V5           1.679e-01  9.973e-02   1.683 0.092324 .
V6          -1.671e-01  9.540e-02  -1.752 0.079845 .
V7          -9.329e-02  9.375e-02  -0.995 0.319690
V8          -1.979e-01  3.831e-02  -5.166 2.39e-07 ***
V9          -5.196e-02  1.761e-01  -0.295 0.767896
V10         -7.846e-01  1.478e-01  -5.310 1.09e-07 ***
V11         -7.781e-02  1.055e-01  -0.738 0.460796
V12          1.511e-01  1.114e-01   1.356 0.174961
V13         -3.745e-01  1.058e-01  -3.540 0.000400 ***
V14         -6.649e-01  8.045e-02  -8.265  < 2e-16 ***
V15         -1.838e-01  1.084e-01  -1.696 0.089825 .
V16          1.898e-02  2.211e-01   0.086 0.931600
V17         -2.416e-02  9.246e-02  -0.261 0.793863
V18         -1.596e-01  2.121e-01  -0.752 0.452002
V19          2.615e-01  1.470e-01   1.778 0.075333 .
V20         -3.901e-01  1.128e-01  -3.457 0.000545 ***
V21          4.734e-01  8.580e-02   5.517 3.45e-08 ***
V22          8.765e-01  1.803e-01   4.860 1.17e-06 ***
V23         -1.337e-01  7.490e-02  -1.785 0.074310 .
V24          1.602e-01  1.848e-01   0.867 0.385821
```

Logistic Regression output-2

```
V10         -7.846e-01  1.478e-01  -5.310 1.09e-07 ***
V11         -7.781e-02  1.055e-01  -0.738 0.460796
V12          1.511e-01  1.114e-01   1.356 0.174961
V13         -3.745e-01  1.058e-01  -3.540 0.000400 ***
V14         -6.649e-01  8.045e-02  -8.265  < 2e-16 ***
V15         -1.838e-01  1.084e-01  -1.696 0.089825 .
V16          1.898e-02  2.211e-01   0.086 0.931600
V17         -2.416e-02  9.246e-02  -0.261 0.793863
V18         -1.596e-01  2.121e-01  -0.752 0.452002
V19          2.615e-01  1.470e-01   1.778 0.075333 .
V20         -3.901e-01  1.128e-01  -3.457 0.000545 ***
V21          4.734e-01  8.580e-02   5.517 3.45e-08 ***
V22          8.765e-01  1.803e-01   4.860 1.17e-06 ***
V23         -1.337e-01  7.490e-02  -1.785 0.074310 .
V24          1.602e-01  1.848e-01   0.867 0.385821
V25          3.093e-02  1.659e-01   0.186 0.852130
V26         -1.228e-01  2.433e-01  -0.505 0.613668
V27         -9.467e-01  1.409e-01  -6.718 1.85e-11 ***
V28         -4.860e-01  1.389e-01  -3.498 0.000469 ***
Amount       1.151e-03  4.907e-04   2.346 0.019001 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4342.9  on 170882  degrees of freedom
Residual deviance: 1375.6  on 170852  degrees of freedom
AIC: 1437.6

Number of Fisher Scoring iterations: 12
```

4. **Implement Random Forest in Spark-R**

A Random Forest classifier is a type of classifier. It is an ensemble learning approach in which numerous decision trees (thus the name forest) make distinct predictions and the majority outcome is chosen as the forecast. Random forest approaches, which are commonly employed for classification issues, are also noted for their resistance to over-fitting. It prevents overfitting by deploying a sequence of weaker trees that cannot overfit the training set and must classify with a majority vote. Training a Classifier on the Same Dataset progressively teaches weak learners to focus on the points that are difficult to classify, thus we took care to restrict each individual tree such that it is individually weak.

```
test$predictedTrim10 <- predict(rfModelTrim10, test)
 # build random forest model using every variable
Frf.fit <- randomForest(class ~ ., data=df_train, ntree=1000,
F                       keep.forest=FALSE, importance=TRUE)

# build dataframe of number of variables and scores
numVariables <- c(1,2,3,4,5,10,17)
F1_Score <- c(F1_1, F1_2, F1_3, F1_4, F1_5, F1_10, F1_all)
variablePerf <- data.frame(numVariables, F1_Score)


# plot score performance against number of variables
options(repr.plot.width=4, repr.plot.height=3)
ggplot(variablePerf, aes(numVariables, F1_Score)) + geom_point() + labs(x
= "Number of Variables", y = "F1 Score", title = "F1 Score Performance")
```

```
Training acc: 0.935714285714
Validation acc: 0.817857142857
```

CONCLUSION

This project aimed to explore, analyze, and build a machine learning algorithm using Python, R, SageMaker, Spark-MLlib, SparkR, TensorFlow. We implemented various ML models to perform Sentiment Analysis of Emotions on a Text Dataset containing 20000 records. We implemented a few Deep Learning classifier models using BERT, RoBERTa and RNN on AWS Sagemaker and SparkMLlib.

The Model with BERT provided us with an accuracy of 97% while the RoBERTa provided 93.75%, and RNN classifier with 97.79% accuracy. The Loss parameters were similar and small for all of these models. Hence, the RNN model appears to be a better option for such data. Although RoBERTa is a BERT model without NSP objective & improved dynamic mask generation, we found BERT worked better for our data. Additionally, we compared performance of these models on different Cluster instances with distinct nodes. We found better CPU/GPU configuration definitely impacts execution speed in AWS.

One of the drawbacks of the data however is that it is past data. We might use a recent data frame that has been updated in future. These new data points can then be used to make predictions or train new models for more accurate results. With larger dataset tuning parameters, we may try to improve the model's accuracy.

REFERENCES

1. Emotions dataset for NLP - https://www.kaggle.com/datasets/praveengovi/emotionsdataset-for-nlp
2. Practical Data Science on AWS Cloud - https://github.com/Ashleshk/Practical-Data-Scienceon-the-AWSCloud-Specialization
3. Machine Learning using Spark MLlib - https://www.youtube.com/watch?v=BxOhyCQ008&t=2s
4. Tensorflow in AWS - https://docs.aws.amazon.com/deep-learningcontainers/latest/devguide/deep-learningcontainers-ecs-tutorials-training.html#deeplearning-containers-ecs-tutorials-training-tf
5. SageMaker Case Study - https://sagemakerexamples.readthedocs.io/en/latest/introduction_to_applying_machine_learning/breast_cancer_prediction/Breast%20Cancer%20Prediction.html
6. SageMaker Studio Notebook Architecture - https://aws.amazon.com/blogs/machinelearning/dive-deep-into-amazon-sagemaker-studio-notebook-architecture/
7. GloVe Word Mapping - https://en.wikipedia.org/wiki/GloVe