Project Objective:

- To perform a sentiment analysis on 400,000+ Amazon product reviews using the Apache Spark framework, an open-source distributed data processing platform which utilizes distributed memory abstraction.
- Perform data preprocessing on the given text and apply ML text feature extraction steps in Classification.
- Evaluate performance of classification algorithms used such as Naive Bayes, Random Forest, and Logistic Regression for the sentiment analysis
- The goal of using Apache Spark's Machine learning library (MLlib) is to handle an extraordinary amount of data effectively using Pyspark, an interface for Apache Spark in Python

**The Dataset:**
The source website from where the dataset was taken, consists of links to product metadata (descriptions, category information, price, brand, and picture features) and product reviews (ratings, text, helpfulness votes) for products belonging to the 30+ product categories sold on Amazon.
The product review datasets for product categories Beauty and Apparel from the year 2018 were chosen. These datasets were combined to get a total of 400,000+ records, which was used for the purpose of this project.

For the project the downloaded data files were in the JSON format. I uploaded the file to an Amazon S3 bucket. I then used Amazon EMR Notebooks along with Amazon EMR clusters running Apache Spark to create and open Jupyter Notebook and JupyterLab interfaces within the Amazon EMR console. In short, EMR allows me to store data in Amazon S3 and run compute as we need to process that data.

**Dataset exploration:**
After setting up the Machine Learning Environment we proceeded to explore our dataset. Data Exploration allows for deeper understanding of a dataset, making it easier to navigate and use the data.
The DataFrame.describe() Pyspark function is used to calculate basic statistics such as count, mean, stddev, min, and max. for the numeric column 'overall'.

**Data cleaning and feature engineering:**
Identified and Removed Null /NA values. As a part of this step the two columns with relevant data for sentiment analysis- 'reviewText' and 'summary' are concatenated into one column called 'text.'(please refer the code)

**Bucketing:**
The Bucketizer() functions available in the Spark MLlib, map a column of continuous features to a column of feature buckets. In simple terms it assigns a bucket value (string) to a numerical input (column), based on the array boundaries provided.
Bucketizer helps us set the labels on the dataset by choosing 0 which stands for negative sentiment , 1 for positive sentiment which are assigned as labels in the process.

**Sampling:**
For the purpose of this project, I have used the 'stratified sampling without replacement' method to create our sample, which will be further used to create the training and testing datasets.

**Creating the Test-Train split:**
Created the 80-20 training and test set split from the sample dataset to train and test the model.

**Tokenizing:**
Used the Regextokenizer() function in Pyspark that converts the input string into lowercase, divides it by whitespaces and does not tokenize any non-character. An index value is assigned to each token.

**Removing Stopwords:**
StopWordsRemover() takes as input a sequence of strings (e.g. the output of a Tokenizer) and drops all the stop words from the input sequences.

**Generating N-Grams:**

Implemented a feature transformer that converts the input array of strings into an array of n-grams.It returns an array of n-grams where each n-gram is represented by a space-separated string of words.

**Hashing:**
Using CountVectorizer and HashingTF estimators are used to generate term frequency vectors.

**Vectorizing:**
Once the text is split into n-grams, it is turned into numerical vectors that only ML algorithms can process—vectorized the n-grams using the TF-IDF method, once indexes have been assigned.

**Created a data pipeline consisting of all these functions and fit the training dataset into the pipeline.**

-
- I compared the performance of the classifiers (Naive Bayes, Logistic Regression, Random Forest) using metrics: Accuracy, Precision, and AUC. At first, we ran the models on our local machine using Databricks and created checkpoints accordingly. During the process we found the benchmark point on a local machine which is Databricks that runs on a single spark node where the compute node was broken during the execution of Gradient Boosted Trees model. I later moved the computation by instantiating 3 nodes (master and 2 slaves) on the EMR cluster, which ran within an hour of exec time.

- I also performed K fold cross validation technique and tuned our model hyperparameters to boost performance. We even dribbled with ensemble methods such as Gradient Boosted trees and XGBoost, however, that did not give any better results as compared to traditional logistic Regression with Bigrams, HashingTF and IDF Vectorization.

- In the analysis, it was found that Logistic Regression with Bigram features has the highest accuracy.  The results indicate that the runtime was the least for Naive Bayes and Logistic Regression with Bigrams when the number of cluster nodes was increased, and data scale was kept the same. It is also found that the scale of data is directly proportional to the runtime for the spark job, irrespective of the number of nodes in the cluster.