Big Data Laboratory CS4830

To develop a prediction model for NYC Parking Tickets dataset on DataProc Cluster using Spark and perform Batch Computation and real-time predictions of test data streaming to Kafka using the trained model

Final Project Report

MM16B027
Shubham Kashyapi

MM17B105
A Ramprasad

MM17B026
R Arun Prakash

FINAL PROJECT REPORT

I. OBJECTIVE:

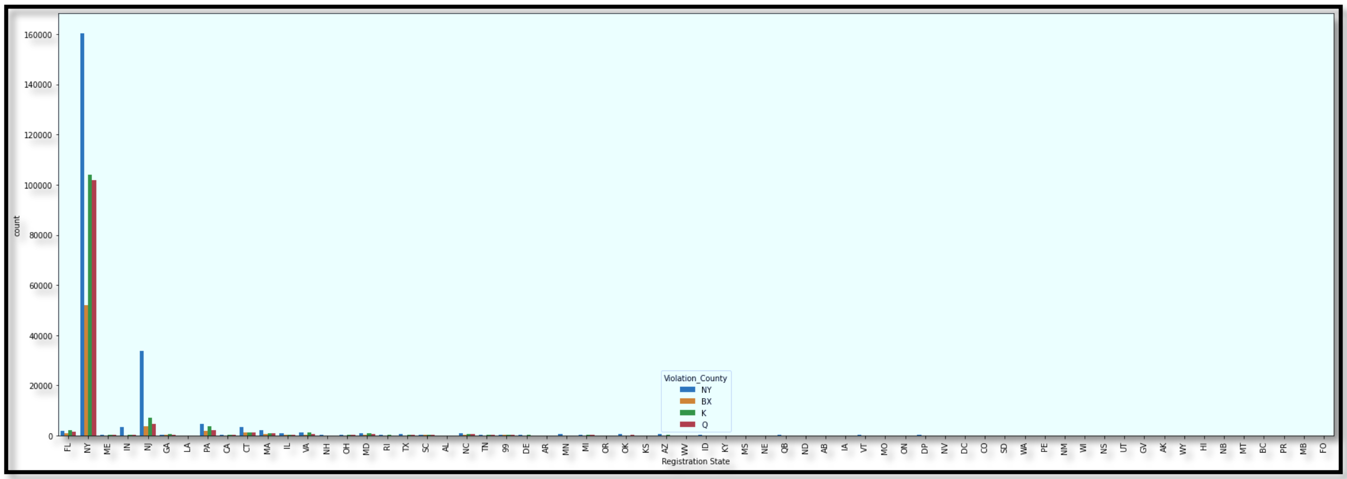
The objective of the project is as follows:

- ❖ Perform data pre-processing and feature engineering techniques on the NYC Parking Tickets dataset followed by training a model in a DataProc cluster using Spark
- ❖ Storing the best model onto the GCS bucket for further computations
- ❖ Streaming the test data stored on the GCS bucket into Kafka
- ❖ Using Structured Spark Streaming to read the data and make real-time predictions

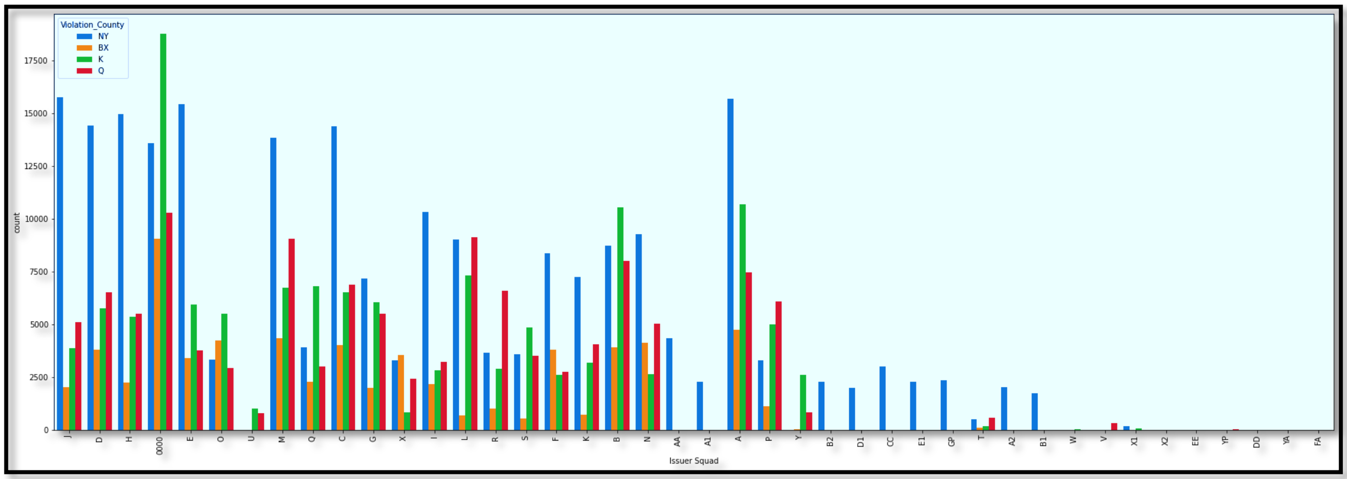
II. DATA VISUALIZATION

- ❖ Almost all the data/features in the dataset were categorical data
- ❖ These features provided various information about the parking tickets issued in New York City such as agency which issued the ticket, registration state, violation code and so on
- ❖ Performed visualizations of some of these features to get an understanding about the provided data
- ❖ Plotted the count of the occurrences of the various categories in these features contributing to different labels in the target column i.e. "*Violation County*"

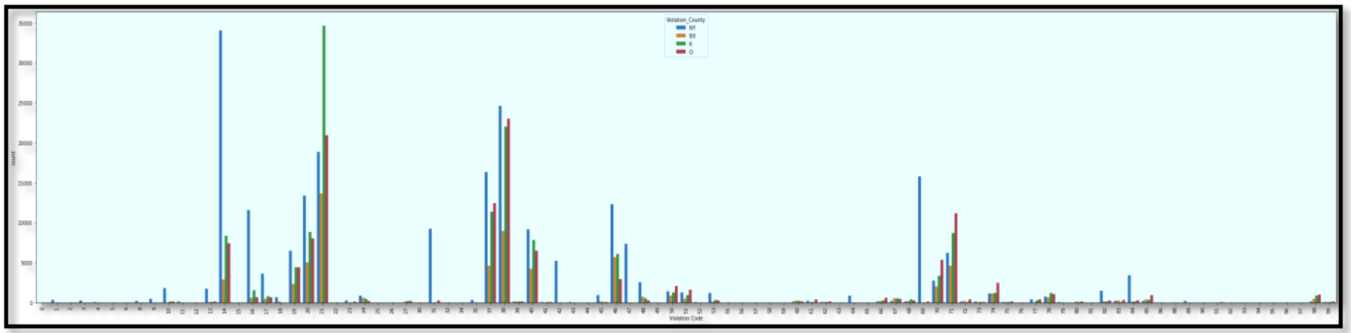
"Registration State" counts by "Violation County"



"Issuer Squad" counts by "Violation County"



"Violation Code" counts by "Violation County"

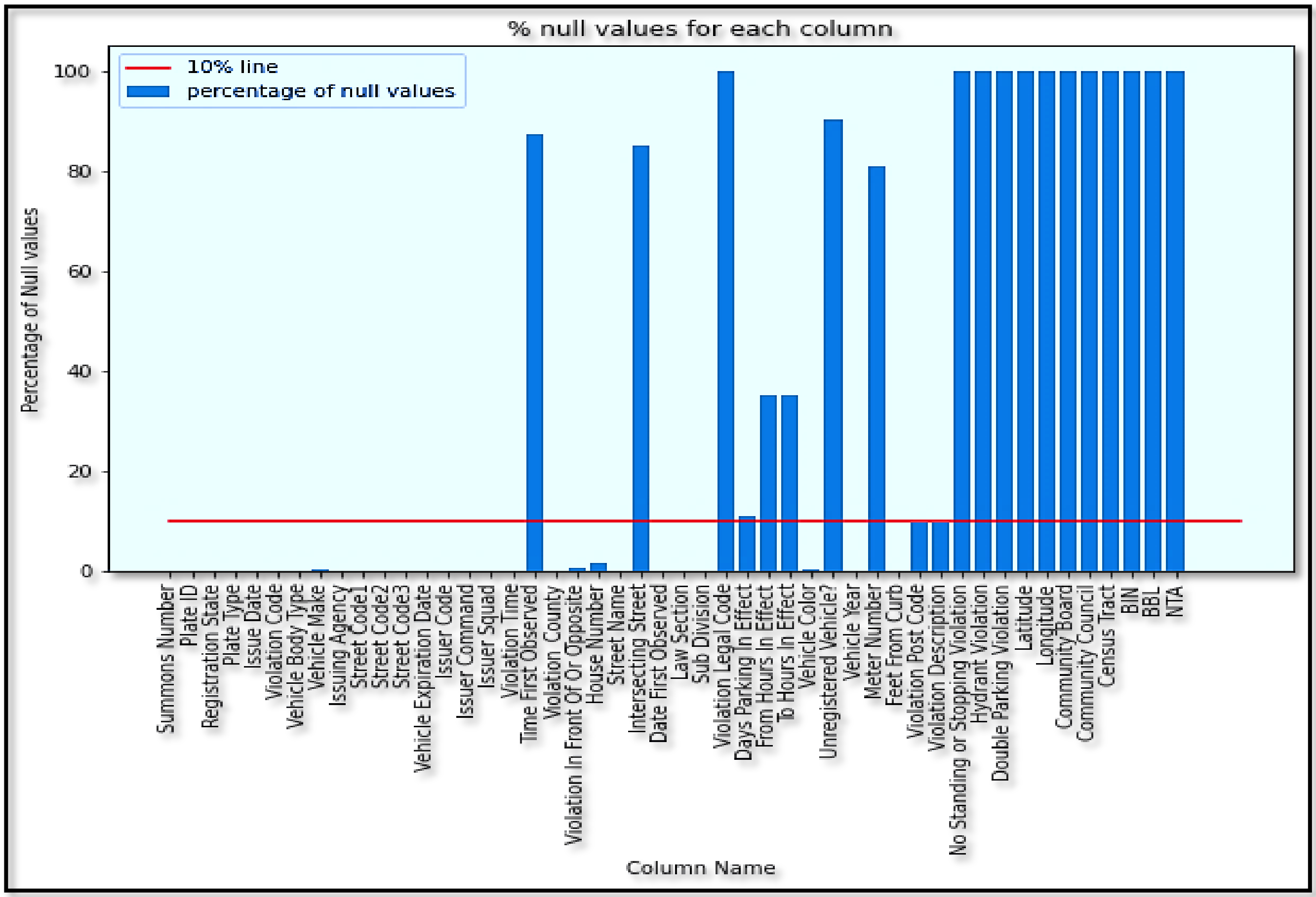


- ❖ Similarly, more such plots were plotted used to visually understand the effect of other features on the target column "Violation County"
- ❖ Other visualizations, such as plotting the percentage of null values per column, unique values per column, constant values per column and so on were also carried out
- ❖ From the above plots we can infer some important details regarding dataset such as:
  - ▪ From the "Registration State" plot, we can observe that more that 90-95% of the vehicles are registered from the 'NY' state of the country (quite obvious as the name of the dataset suggests the same)
  - ▪ From the "Issuer Squad" plot, we can observe that some squads issued none to very few tickets and other squads issued most of the tickets in 'NY' county followed by 'BX'; county 'K' and 'Q' had almost similar tickets issued by all the issuer squads
  - ▪ From the "Violation Code" plot, we can observe that some violations such as code 14, 20, 21, 37, 38, 40, 46,72 are more frequent than the others
  - ▪ Overall we can observe that most of the tickets were issued from the 'NY' city in the state and that this is an imbalanced dataset

III. DATA PRE-PROCESSING:

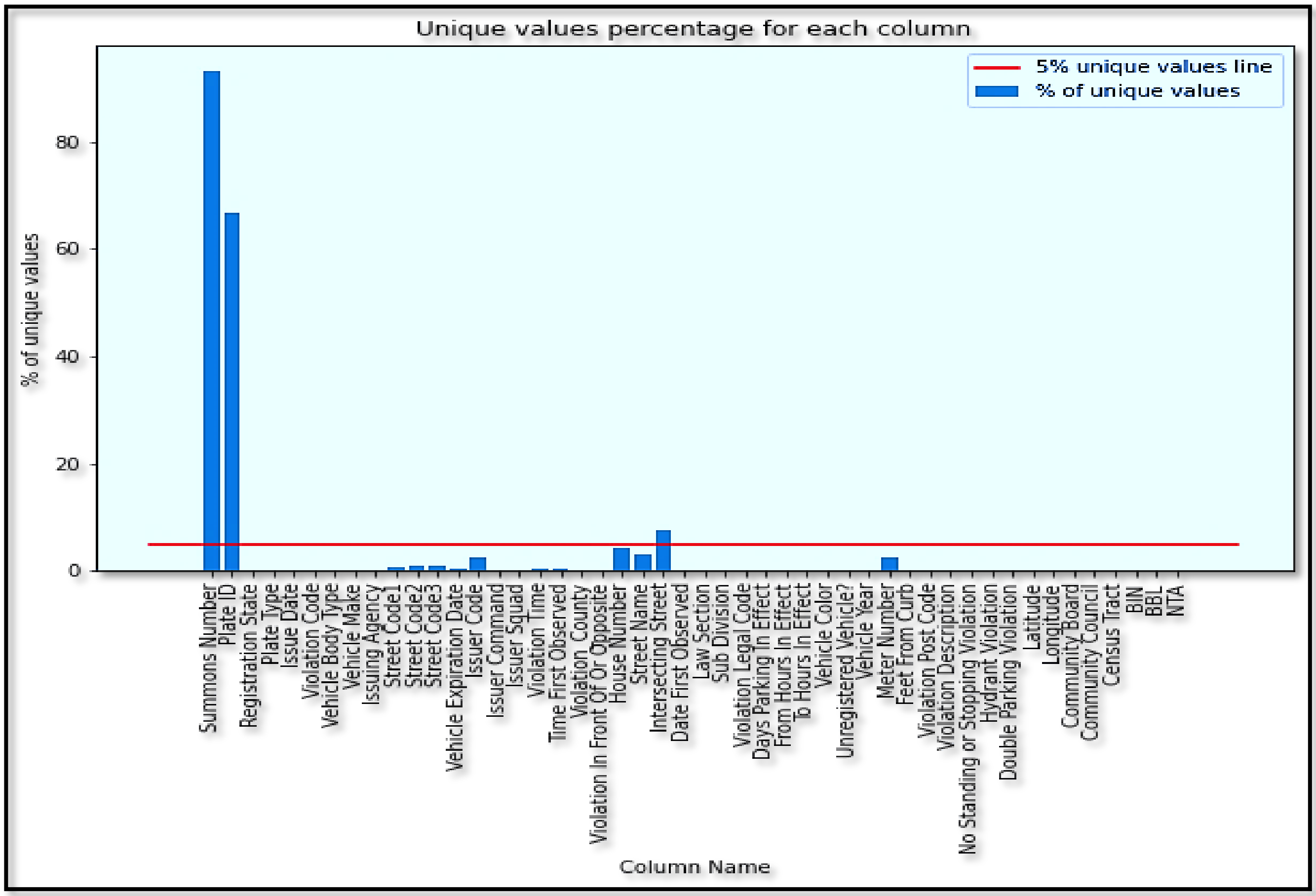
❖ Null / missing value columns:

- Many features in the dataset had missing values
- Plotted the percentage of null values for all the features in the dataset
- Removed the features having more than 10% null values
- These features would just increase the complexity of the dataset and would be of no help for the model



❖ Unique value columns:

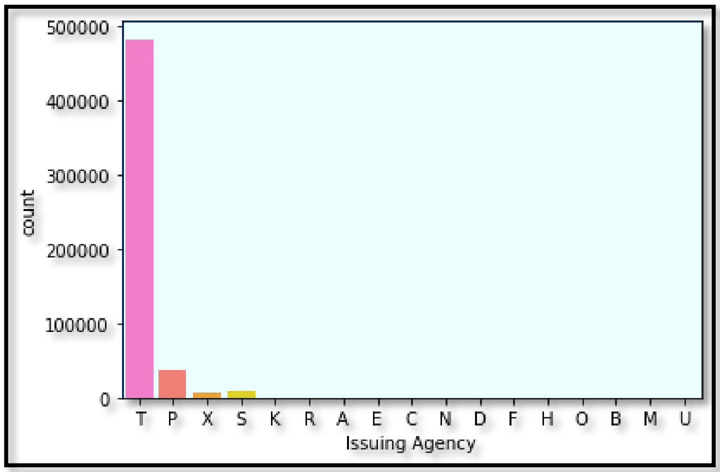
- Some features contain data unique to each row/vehicle
- Plotted the number of unique values for each remaining column
- Removed the features having more than 5% unique values as these features would not help the model in learning trends and patterns in the data



❖ Constant value columns:

- Some features contain data with very low variance (constant values)
- In other words, it has almost same values for each row/vehicle
- "Issuing Agency" is one such column where the entry 'T' occurs almost 90% of the time
- Such features were removed in the pre-processing step

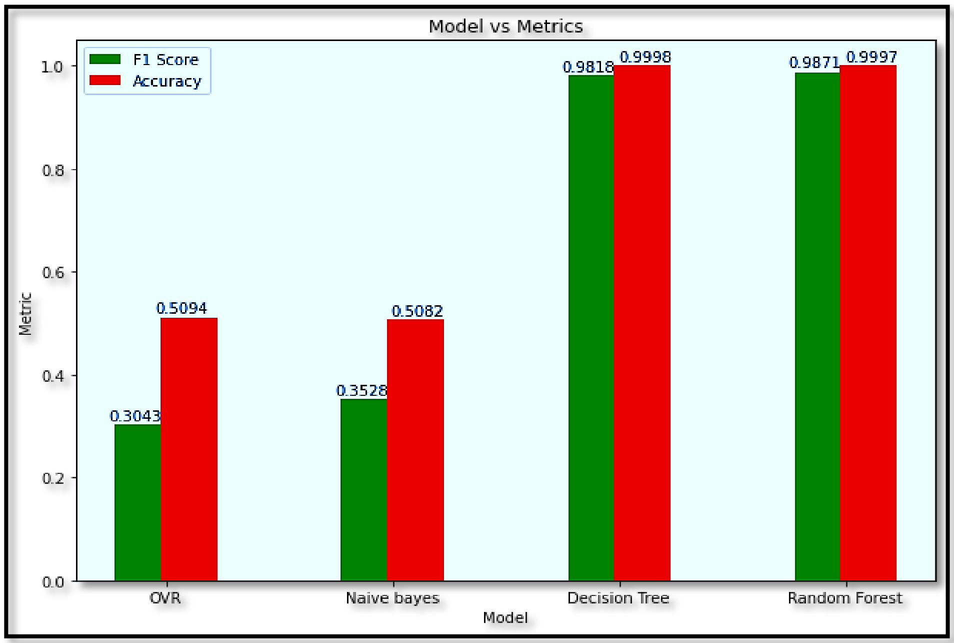
Count of various categories in "Issuing Agency"



- Label Encoding:
  - Almost all the features in the dataset were categorical
  - We Label Encoded these remaining features to get a numerical data in order to try out some of the ML models
  - Used StringIndexer() function in PySpark to carry out this operation
- Scaling:
  - Scaling the features with respect to only Standard Deviation and not Mean improved the model performance significantly
  - Used StandardScaler() function in PySpark to carry out this operations

IV. MODEL:

- After performing above mentioned data pre-processing, we trained several ML classifications models on this dataset to check their performance
- Following plot outlines the performance of each model with respect to accuracy/F1 score:



- We compared the performance of each model against each other and the inferences deduced from them are as follows:
  - We trained Logistic Regression model on the dataset using an OneVsRest classifier to accommodate multiple labels in the target column. F1 score on validation set was only about 30%, probably because most of the features were categorical and the model was not able to find a relation between them

- Then we trained a Naïve Bayed model but again got a poor performance, reason being that Naïve Bayes assumes independent features which is not the case in the dataset. It gave an F1 score of about 35% on the held-out data.
- Then we trained Decision Tree model on the dataset, which gave an impressive performance of about 98% on the held-out data. Observation here being that as most of the data was categorical, Decision Tree model was able to split the feature space into multiple regions based on the features and assign a target label to each one of these regions with high confidence.
- Then we tried to improve upon the performance of Decision Tree model, by introducing Random Forest model. This will help in reducing the variance of the model by combining multiple Decision Trees, each being trained on a subset of data with subset of features only available for split at each interior node.

❖ "Random Forest" model outperformed other and was chosen as the final model.

Parameter Tuning

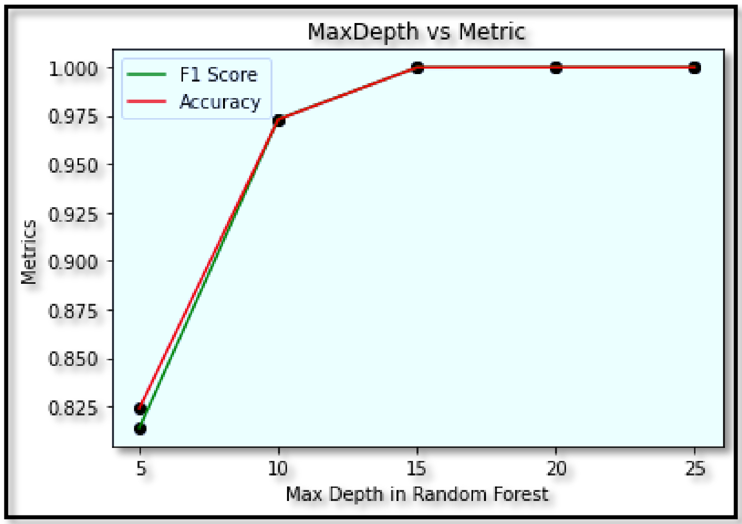
❖ Tuned the parameters of the Random Forest classifier by building a ParamGridBuilder() function in PySpark for the maxDepth parameter

❖ Created a pipeline consisting of the following stages:
- Assembler
- Scaler
- Random Forest classifier

❖ Performed a K-Fold cross validation for this pipeline model using the CrossValidator()

❖ Chose the best model from this cross validator as the FinalTrainedModel and stored it into GCS bucket for further real time predictions by streaming test data

❖ Following plot visualizes the performance of Random Forest classifier by maxDepth



V. Real-Time Computation:

Implemented Spark Structured streaming for producing real time predictions using Kafka.

The procedure is as follows:

- ❖ Deployed a Kafka cluster, obtained its internal IP and created the topic "nycdata" to publish the messages into Kafka and to subscribe to the messages of this topic
- ❖ Implemented a producer code on a DataProc cluster that reads the NYC Parking Tickets dataset line by line and writes each row into the topic "nycdata" in Kafka
- ❖ Then ran a consumer code on DataProc cluster that uses spark structured streaming for producing real time predictions on these rows by utilizing the model trained above
- ❖ Consumer code also computes the accuracy for each batch of data streamed in and prints the results on the console

Producer

Source code: project_producer.py

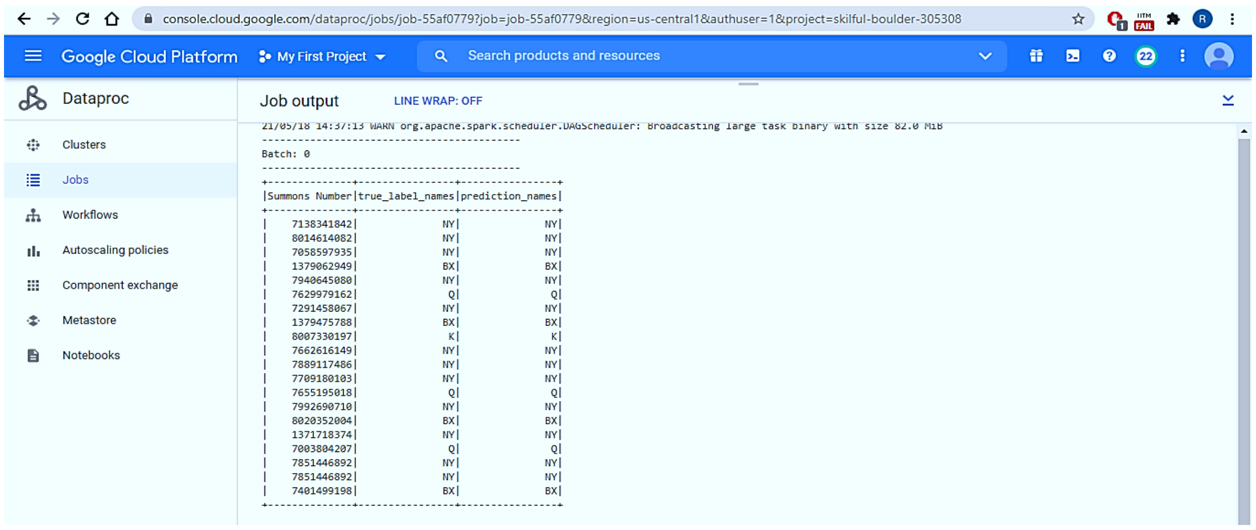
Comments:

- Producer is deployed as PySpark job in the DataProc cluster which publishes message/data to the "nycdata" topic in Kafka
- Producer reads the lines in NYC Parking Tickets dataset and sends it line by line to the "nycdata" topic in Kafka

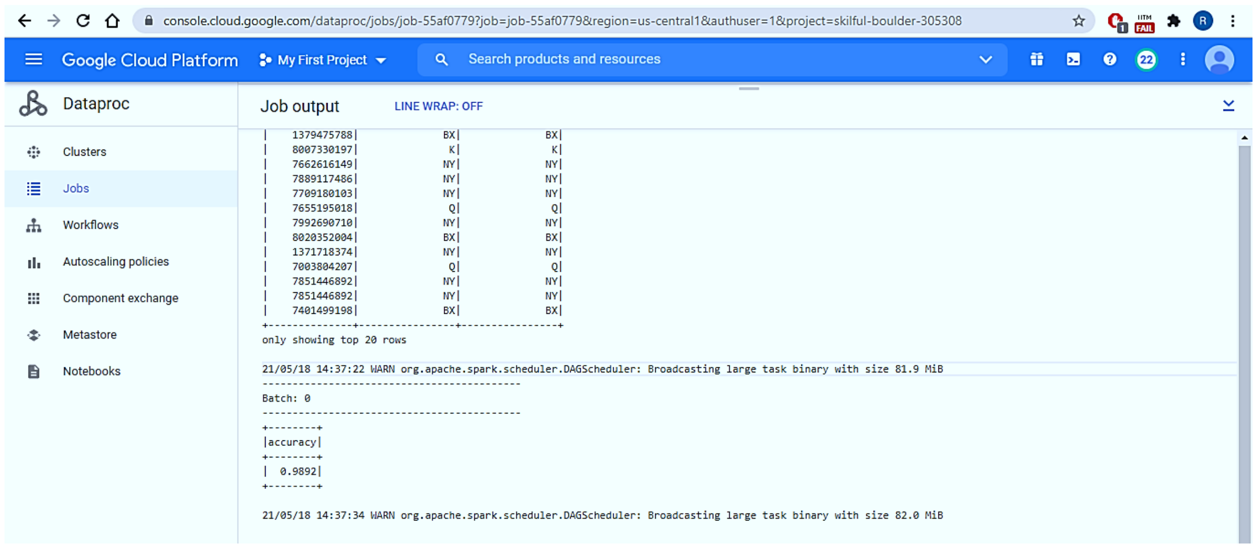
Consumer

Source code: project_consumer.py

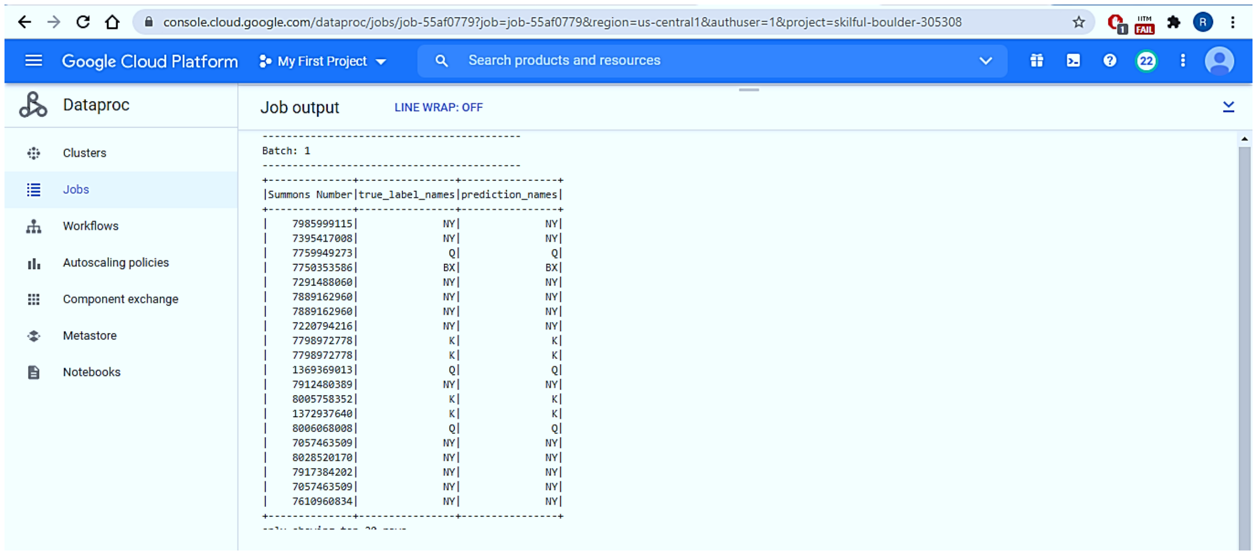
Screenshots:


console.cloud.google.com/dataproc/jobs/job-55af0779?job=job-55af0779&region=us-central1&authuser=1&project=skilful-boulder-305308
Google Cloud Platform
My First Project
Search products and resources
Dataproc
Clusters
Jobs
Workflows
Autoscaling policies
Component exchange
Metastore
Notebooks
Job output
LINE WRAP: OFF

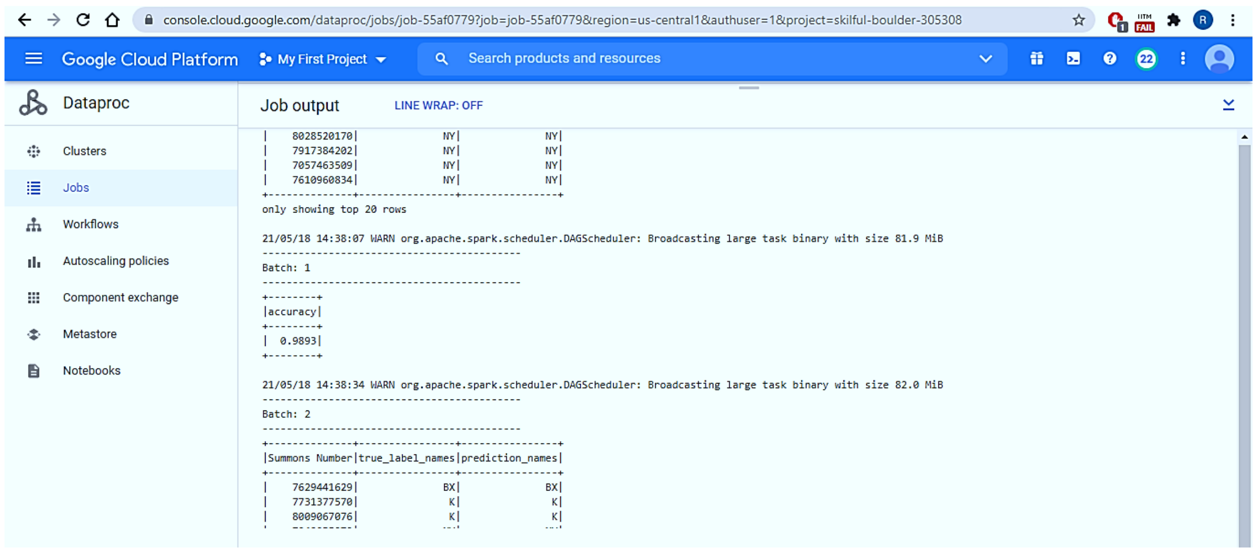
```
21/05/18 14:37:13 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 82.0 MiB
-------------------------------------------
Batch: 0
-------------------------------------------
+--------------+----------------+----------------+
|Summons Number|true_label_names|prediction_names|
+--------------+----------------+----------------+
|    7138341842|              NY|              NY|
|    8014614082|              NY|              NY|
|    7058597935|              NY|              NY|
|    1379062949|              BX|              BX|
|    7940645080|              NY|              NY|
|    7629979162|               Q|               Q|
|    7291458067|              NY|              NY|
|    1379475788|              BX|              BX|
|    8007330197|               K|               K|
|    7662616149|              NY|              NY|
|    7889117486|              NY|              NY|
|    7709180103|              NY|              NY|
|    7655195018|               Q|               Q|
|    7992690710|              NY|              NY|
|    8020352004|              BX|              BX|
|    1371718374|              NY|              NY|
|    7003804207|               Q|               Q|
|    7851446892|              NY|              NY|
|    7851446892|              NY|              NY|
|    7401499198|              BX|              BX|
+--------------+----------------+----------------+
```



```
|    1379475788|              BX|             BX|
|    8007330197|               K|              K|
|    7662616149|              NY|             NY|
|    7889117486|              NY|             NY|
|    7709180103|              NY|             NY|
|    7655195018|               Q|              Q|
|    7992690710|              NY|             NY|
|    8020352004|              BX|             BX|
|    1371718374|              NY|             NY|
|    7003804207|               Q|              Q|
|    7851446892|              NY|             NY|
|    7851446892|              NY|             NY|
|    7401499198|              BX|             BX|
+--------------+----------------+---------------+
only showing top 20 rows

21/05/18 14:37:22 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 81.9 MiB
-------------------------------------------
Batch: 0
-------------------------------------------
+--------+
|accuracy|
+--------+
|  0.9892|
+--------+

21/05/18 14:37:34 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 82.0 MiB
```



```
-------------------------------------------
Batch: 1
-------------------------------------------
+--------------+----------------+---------------+
|Summons Number|true_label_names|prediction_names|
+--------------+----------------+---------------+
|    7985999115|              NY|             NY|
|    7395417008|              NY|             NY|
|    7759949273|               Q|              Q|
|    7750353586|              BX|             BX|
|    7291488060|              NY|             NY|
|    7889162960|              NY|             NY|
|    7889162960|              NY|             NY|
|    7220794216|              NY|             NY|
|    7798972778|               K|              K|
|    7798972778|               K|              K|
|    1369369013|               Q|              Q|
|    7912480389|              NY|             NY|
|    8005758352|               K|              K|
|    1372937640|               K|              K|
|    8006068008|               Q|              Q|
|    7057463509|              NY|             NY|
|    8028520170|              NY|             NY|
|    7917384202|              NY|             NY|
|    7057463509|              NY|             NY|
|    7610960834|              NY|             NY|
+--------------+----------------+---------------+
```



```
|    8028520170|              NY|             NY|
|    7917384202|              NY|             NY|
|    7057463509|              NY|             NY|
|    7610960834|              NY|             NY|
+--------------+----------------+---------------+
only showing top 20 rows

21/05/18 14:38:07 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 81.9 MiB
-------------------------------------------
Batch: 1
-------------------------------------------
+--------+
|accuracy|
+--------+
|  0.9893|
+--------+

21/05/18 14:38:34 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 82.0 MiB
-------------------------------------------
Batch: 2
-------------------------------------------
+--------------+----------------+---------------+
|Summons Number|true_label_names|prediction_names|
+--------------+----------------+---------------+
|    7629441629|              BX|             BX|
|    7731377570|               K|              K|
|    8009067076|               K|              K|
```

- Above screenshots demonstrate data/message received by the consumer

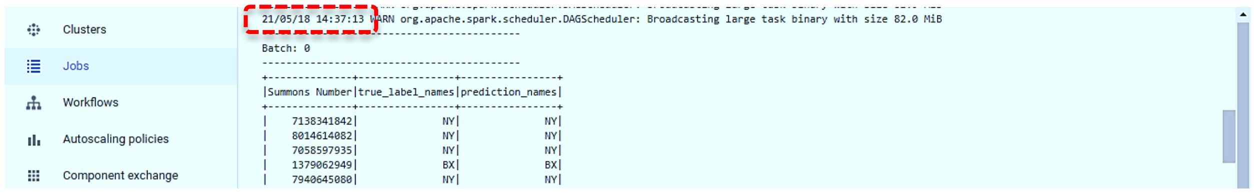
Comments:

- Consumer is deployed as PySpark job in the DataProc cluster which has been subscribed to the “nycdata” topic in Kafka
- Once and when producer sends any message to the “nycdata” topic, the message is passed to the consumer here. In this case, lines of the NYC Parking Tickets dataset are passed in json format as messages
- Consumer then uses the trained model on passed NYC Parking Tickets dataset to make predictions on the incoming message lines in batches and also computes and prints a unique ID (Summons Number) for each row, true target, predicted target along with the accuracy on the console for each batch

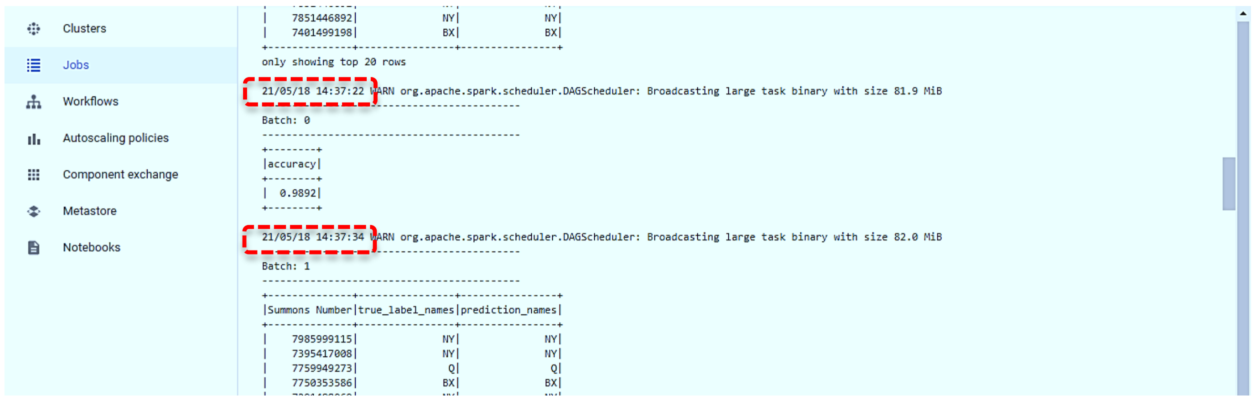
Latency of processing each window (batch)

Screenshots:

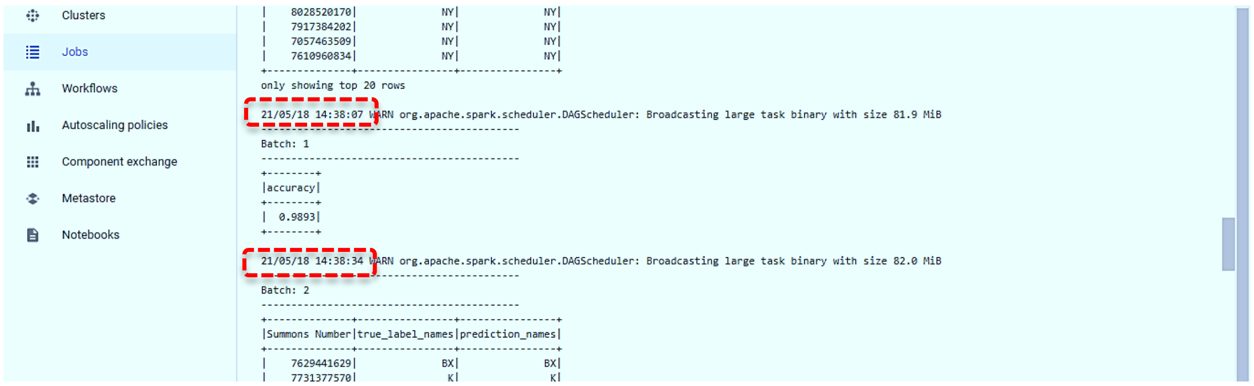
- Boxes marked in red in the following screenshots convey the information regarding the processing time for each window/batch



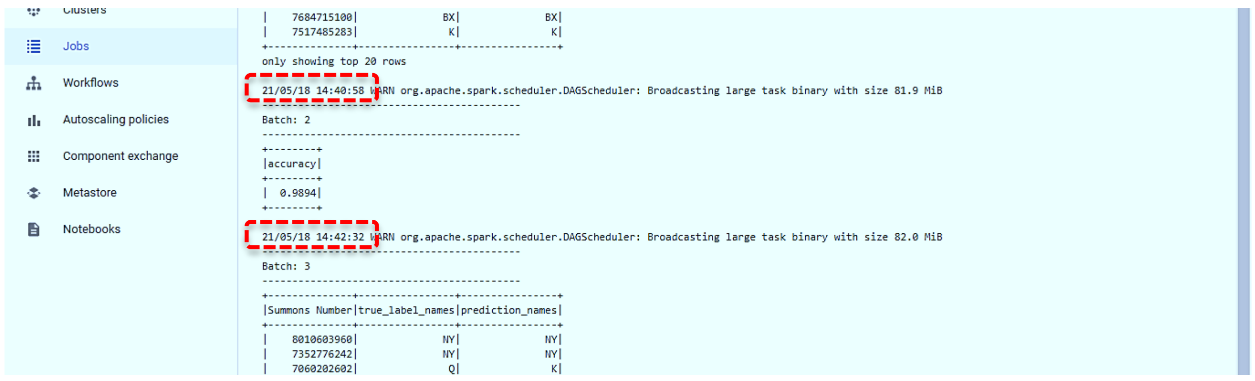
```
21/05/18 14:37:13 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 82.0 MiB
-------------------------------------------
Batch: 0
-------------------------------------------
+--------------+----------------+----------------+
|Summons Number|true_label_names|prediction_names|
+--------------+----------------+----------------+
|    7138341842|              NY|              NY|
|    8014614082|              NY|              NY|
|    7058597935|              NY|              NY|
|    1379062949|              BX|              BX|
|    7940645080|              NY|              NY|
```



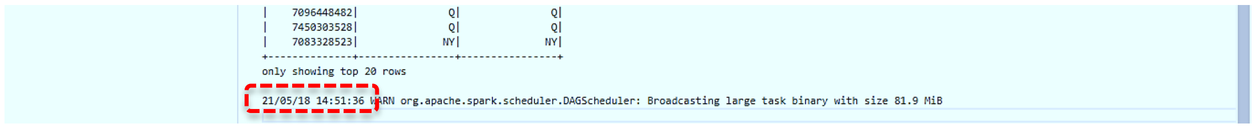
```
|    7851446892|              NY|              NY|
|    7401499198|              BX|              BX|
+--------------+----------------+----------------+
only showing top 20 rows

21/05/18 14:37:22 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 81.9 MiB
-------------------------------------------
Batch: 0
-------------------------------------------
+--------+
|accuracy|
+--------+
|  0.9892|
+--------+

21/05/18 14:37:34 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 82.0 MiB
-------------------------------------------
Batch: 1
-------------------------------------------
+--------------+----------------+----------------+
|Summons Number|true_label_names|prediction_names|
+--------------+----------------+----------------+
|    7985999115|              NY|              NY|
|    7395417008|              NY|              NY|
|    7759949273|               Q|               Q|
|    7750353586|              BX|              BX|
```



```
|    8028520170|              NY|              NY|
|    7917384202|              NY|              NY|
|    7057463509|              NY|              NY|
|    7610960834|              NY|              NY|
+--------------+----------------+----------------+
only showing top 20 rows

21/05/18 14:38:07 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 81.9 MiB
-------------------------------------------
Batch: 1
-------------------------------------------
+--------+
|accuracy|
+--------+
|  0.9893|
+--------+

21/05/18 14:38:34 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 82.0 MiB
-------------------------------------------
Batch: 2
-------------------------------------------
+--------------+----------------+----------------+
|Summons Number|true_label_names|prediction_names|
+--------------+----------------+----------------+
|    7629441629|              BX|              BX|
|    7731377570|               K|               K|
```



```
|    7684715100|              BX|              BX|
|    7517485283|               K|               K|
+--------------+----------------+----------------+
only showing top 20 rows

21/05/18 14:40:58 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 81.9 MiB
-------------------------------------------
Batch: 2
-------------------------------------------
+--------+
|accuracy|
+--------+
|  0.9894|
+--------+

21/05/18 14:42:32 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 82.0 MiB
-------------------------------------------
Batch: 3
-------------------------------------------
+--------------+----------------+----------------+
|Summons Number|true_label_names|prediction_names|
+--------------+----------------+----------------+
|    8010603960|              NY|              NY|
|    7352776242|              NY|              NY|
|    7060202602|               Q|               K|
```



```
|    7096448482|               Q|               Q|
|    7450303528|               Q|               Q|
|    7083328523|              NY|              NY|
+--------------+----------------+----------------+
only showing top 20 rows

21/05/18 14:51:36 WARN org.apache.spark.scheduler.DAGScheduler: Broadcasting large task binary with size 81.9 MiB
```

<u>Comments</u>:

- From the time mentioned to process each batch in the above screenshots, we compute the time taken for implementing predictions and computing accuracy by each batch as follows:

Batch	**Latency of processing predictions per batch (in seconds)**	**Latency of computing accuracy per batch (in seconds)**
0	*14:37:13 ---> 14:37:22* **9**	*14:37:22 ---> 14:37:34* **12**
1	*14:37:34 ---> 14:38:07* **33**	*14:38:07 ---> 14:38:34* **27**
2	*14:38:34 ---> 14:40:58* **144**	*14:40:58 ---> 14:42:32* **94**
3	*14:42:32 ---> 14:51:36* **544**	*14:51:36 ---> 14:53:18* **102**

VI. CONCLUSION AND KEY TAKEAWAYS:

- ❖ In this project we successfully created a prediction model which performs best to predict "Violation County" from the NYC Parking Tickets dataset.
- ❖ We plotted and visualized a few data columns and selected the required columns and preprocessed them before using in the model (discussed above).
- ❖ Using this best saved model we even performed the Real time predictions using Kafka Streaming.
- ❖ Our key takeaway from this project is that now we could process and model an ML algorithm on any Big Data using PySpark.
- ❖ Not only batch computation (as we used to do before) but also Real time data inflow can be readily analyzed using streaming techniques, which was our major takeaway.
- ❖ As almost all fields' stream real time data, it is an important skill which we developed over the course of this lab.