Anomaly Detection in Cloud Servers

Team 4 SLACKERS
Unnati Aggarwal (002741568)
Shantanu Sachdeva (002748942)

WHY?

The purpose of this project is to leverage telecom network logs to identify and flag anomalous activities, ultimately enhancing cybersecurity threat detection. By analyzing these logs, the project aims to develop algorithms and models that can automatically detect patterns indicative of potential threats or abnormalities within the network. This proactive approach can help network administrators and security teams to quickly identify and respond to potential cybersecurity incidents, thereby strengthening the overall security posture of the telecom network.

Use Case

- Network Administrator inputs stream of telecom server logs to the pub/sub topic (Updated: Apache Kafka Topic) to detect network attacks.
- Server Health Check software runs a batch of input logs hourly to detect a spike in outliers and emails the generated report to the administrator. (Updated: We plan to send daily updates of the network profile)

Data Sources

- We used the KDD Cup 1999 Dataset
- Raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN.
- Data is peppered with multiple attacks

Attacks fall into four main categories with 24 further subtypes:

- DOS: denial-of-service, e.g. syn flood; (Teardrop, Back, Land, Smurf, Neptune, Pod)
- R2L: unauthorized access from a remote machine, e.g. guessing password; (guess_passwd, map, spy, phd, warezmaster, wazerclient, multi hop, ftp_write)
- U2R: unauthorized access to local superuser (root) privileges, e.g., various ``buffer overflow'' attacks; (perl, rootkit, buffer_overflow, load module)
- Probing: surveillance and other probing, e.g., port scanning. (Nmap, portsweep, ipsweep, satan)

Features

feature name	*description*	*type*
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous
feature name	*description*	*type*
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete
feature name	*description*	*type*
count	number of connections to the same host as the current connection in the past two	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Data Sources

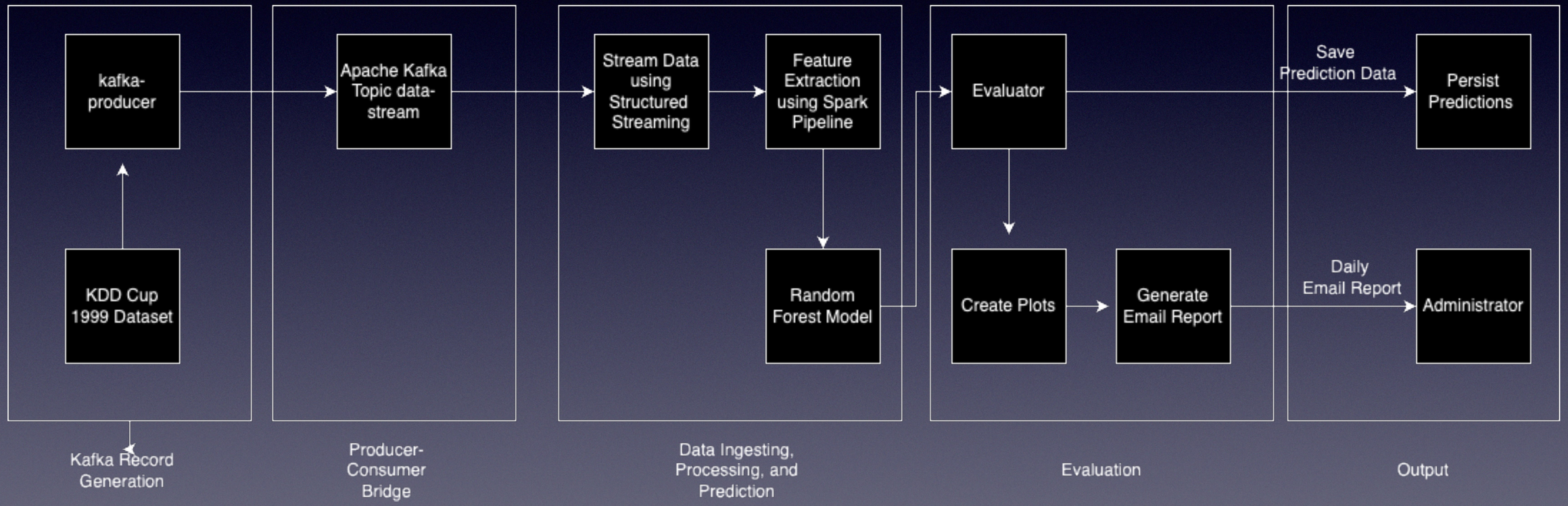
- Initially, we plan to use a pipeline of simulated data using DataFlow.
- Eventually, we aim to use live data streams.
- Update: We decided not to used simulated data, as simulated data may introduce bias into the model, as it is not diverse enough and does not cover all relevant scenarios. This can lead to overfitting, where the model performs well on the training data but poorly on new, unseen data. This would have resulted in a model that fails to perform well in real-world applications.

Methodology

Machine learning-based anomaly detection solution by highlighting the following key components:

1. Generating synthetic data to simulate production volume using Dataflow and Pub/Sub. (Updated: We implemented the Pub/Sub infrastructure using Google Cloud, but had to adopt Apache Kafka instead, due to high costs of cloud infrastructure, lack of documentation and connectors used with Google Cloud Pub/Sub)
2. Update: We introduced structured streaming using Apache Spark. A read stream context is always alive, and ready to process new logs from the Apache Kafka topic.
3. Extracting features and real time prediction using Scala/Spark.
4. Training and normalizing data using BigQuery ML's built-in k-means clustering model.(Updated: Due to high cost of using BigQuery, we switched to Apache Spark Machine Learning Libraries. We trained and tested Logistic Regression and Random Forest model)
5. Update: To send the daily network log report, we set up a DNS name and SPF record, we first created a DNS A record pointing to our server's IP address for the domain, and then added a TXT record containing the SPF policy to authorize our mail server to send emails on behalf of our domain.

Updated Methodology Diagram



Milestones

- 21st March: Establishing log schemas and mocking data to the Pub/Sub in real time. Establishing the feature extraction criteria on aggregated logs using Scala/Spark. (Updated: Implemented Apache Kafka Topic, data publisher and subscriber programs)
- 28th March: Creating the K-Means model using BQ ML. (Updated: We implemented Logistic Regression and Random Forest Model. We switched to Random Forest Model as our data was labelled)
- 4th April : Testing with outlier mock date to see if the model is able to detect it
- 11th April : Setting up output email notification
- 18th April : Final Presentation

The Big Question, Scala

- Data Publishing
- Data Ingestion
- Feature Extraction

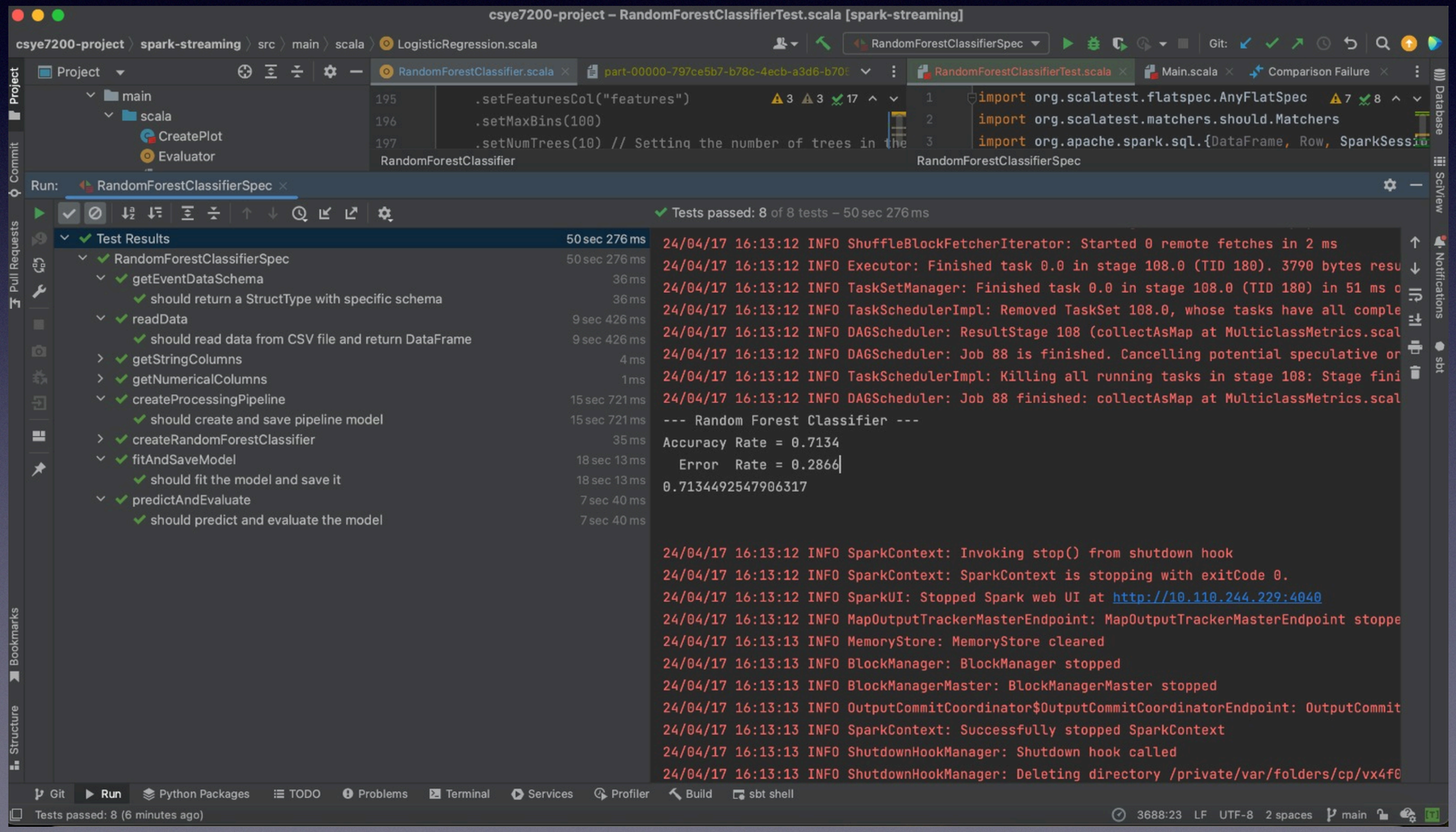
Acceptance Criteria

- Data loss during ingestion from pub/sub topic should be less than 10% (Satisfied for our use case, but unsure if it can scale)
- Ensure the prediction model has a minimum accuracy of 90% in identifying outliers (Not Satisfied: We are able to achieve an accuracy of 71.3% using random forest on test data)

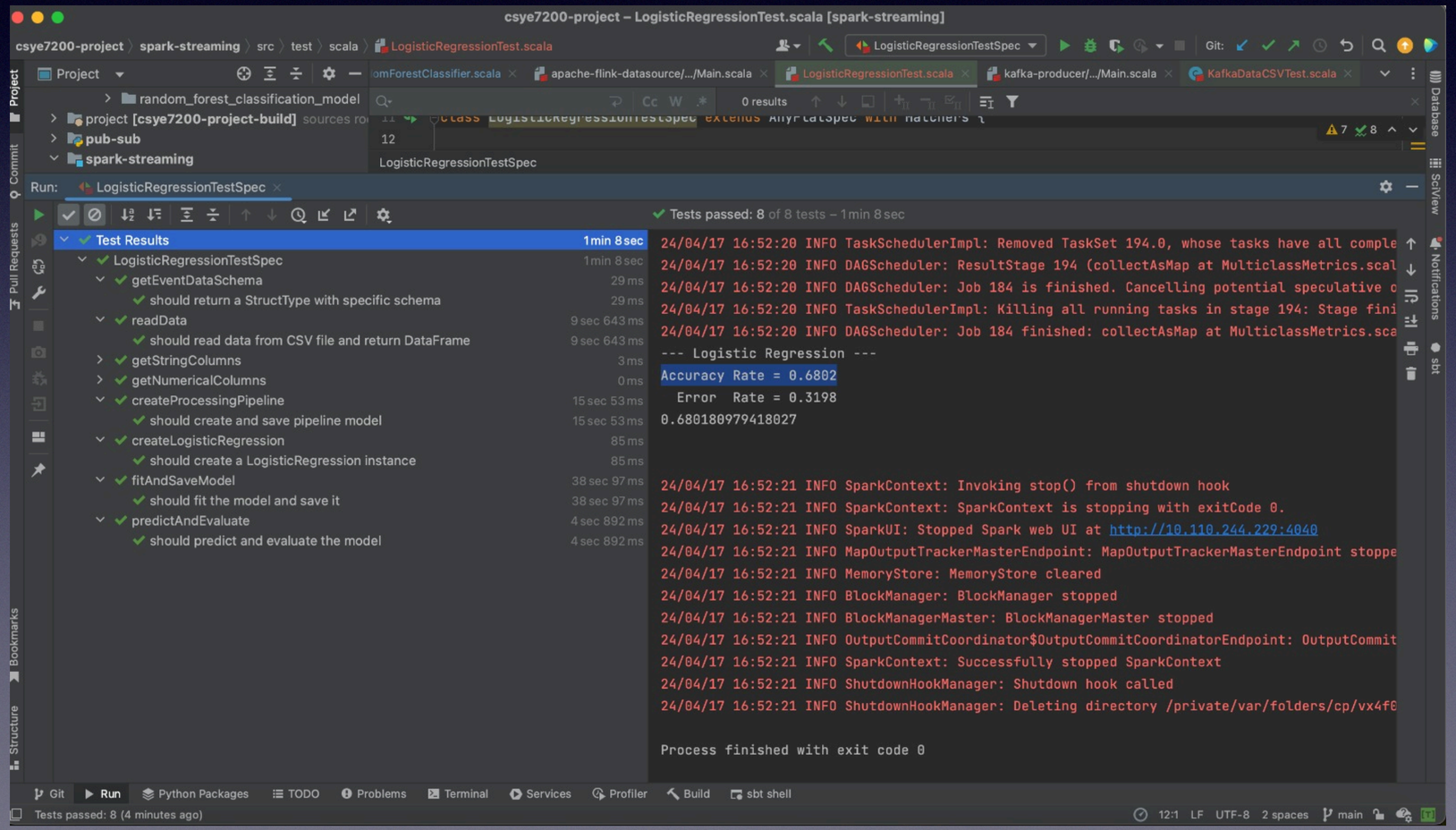
Goals

- Getting hands-on and understanding streaming analytics.
- Working with Google managed services like Dataflow, BigQuery ML
- Writing efficient and Scalable code in Scala
- Getting domain knowledge of networks, and anomaly detection using networks logs

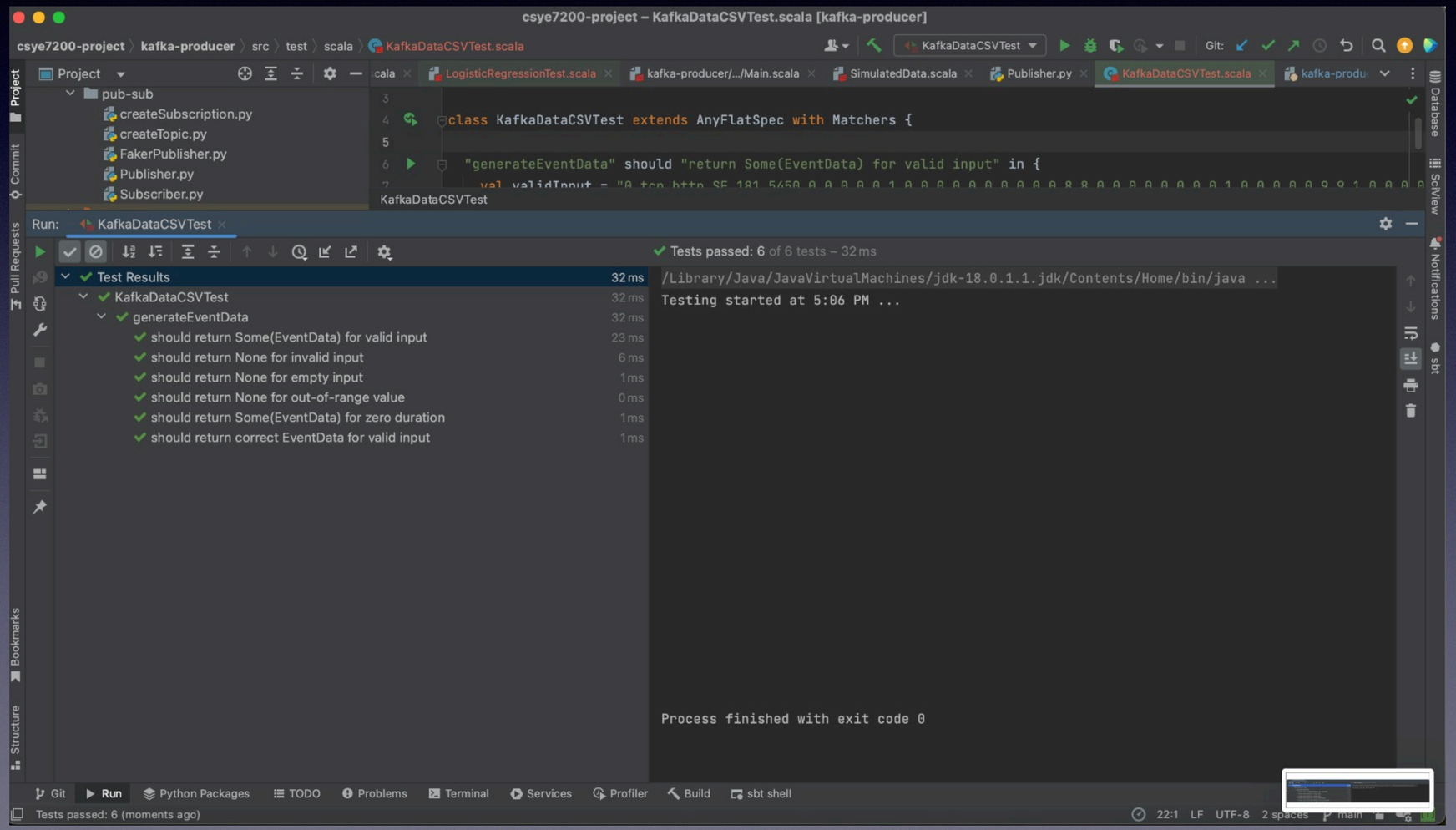
Test Cases for Random Forest Model



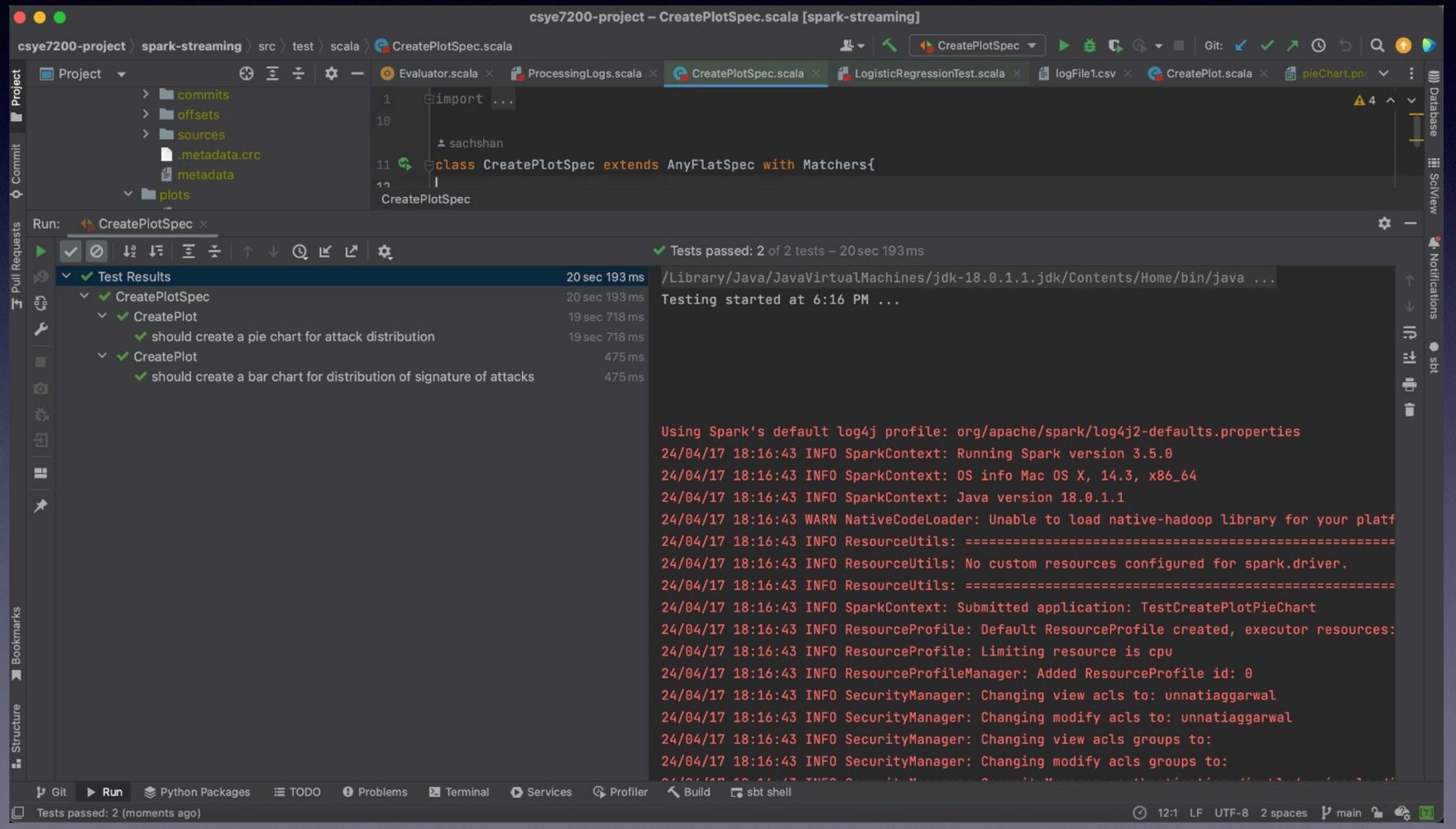
Test Cases for Logistic Regression Model



Test Cases for Kafka Producer



Test Cases for CreatePlot



Test Cases for Utils