ANALYSIS OF MACHINE LEARNING MODELS FOR AIRLINE PASSENGERS’ SATISFACTION USING PYSPARK

COURSEWORK:
BIG DATA ANALYTICS AND DATA VISUALISATION
7153CEM

BY

AYOBAMI ENIOLA AGBOOLA
15620073

DATA SCIENCE AND COMPUTATIONAL INTELLIGENCE
COVENTRY UNIVERSITY,
COVENTRY, UK.

DECEMBER, 2024

ANALYSIS OF MACHINE LEARNING MODELS FOR AIRLINE PASSENGERS' SATISFACTION USING PYSPARK

ABSTRACT

The rapid growth of airline travel and large volume of data in the industry has necessitated advanced tools to analyse and predict passenger satisfaction, a critical factor in maintaining competitive advantage. This study employs PySpark to implement machine learning models for predicting airline passenger satisfaction. The dataset, consisting more than 120,000 records of passenger feedback across various categories, was preprocessed to handle missing values, encode categorical variables, and assemble feature vectors. Four machine learning models, logistic regression, support vector machine (SVM), random forest, and gradient boosted trees (GBT) were trained, evaluated, and also optimized through hyperparameter tuning. The results revealed that ensemble models outperformed linear models in the predictive performance, achieving accuracy of 96% after tuning. Feature importance analysis highlighted technological service quality, travel type, and customer type as significant contributors to satisfaction predictions. Additionally, the study observed consistent evaluation metrics across models, underscoring the robustness of the PySpark framework in handling large, complex datasets. This research demonstrates the applicability of big data analytics in customer satisfaction analysis and provides a pathway for future studies to enhance airline customer experience through data driven insights. The findings emphasize the critical role of machine learning in transforming passenger feedback into actionable strategies for airlines.

INTRODUCTION

The aviation industry plays a pivotal role in global transportation and connects millions of passengers daily across various destinations. In order to retain customer patronage, increase market-share and, ultimately profitability, airlines need to prioritize high-quality service to their passengers, thus ensuring customers' satisfaction has become a critical factor for the sustainability and success of airlines in a highly competitive market (Liou et al., 2011, p. 131).

Satisfaction levels are influenced by a range of factors, including flight schedules, in-flight services, customer service, and ticket pricing (Clemes et al., 2008, p. 49). Understanding and predicting passenger satisfaction is essential for airlines to improve service quality, retain customers, and maintain a competitive edge (Shah et al, 2020, p. 159). Therefore, it is very important for airlines to periodically evaluate customer satisfaction, perceived service quality and loyalty intentions of the business (Liou et al., 2011).

Data driven approaches leveraging machine learning models have emerged as powerful tools to uncover insights into customer satisfaction patterns ad predict outcomes effectively. The analysis, processing, and storing of big data can be challenging due to data imperfection, massive data size, computational difficulty, and lengthy evaluation time because of the sheer volume, velocity, and variety. Big Data frameworks like Hadoop MapReduce and Spark are potent tools that provide an effective way to analyse huge datasets and Apache Spark is one of the most widely used large-scale data processing engines due to its speed, low latency in-memory computing, and powerful analytics (Malik et al., 2024, p. 151785).

This study investigates the ability of four classification models to predict airline passengers' satisfaction while employing PySpark to implement scalable machine learning pipelines due to its suitability in handling large datasets.

RELATED WORKS

Customer satisfaction has become a key intermediary objective in service operations due to the benefits it brings to organizations. Previous research has demonstrated that satisfaction is strongly associated with re-purchase. Chen (2008, p. 709) highlighted the need for organizations to provide their passengers with quality services in order to gain a competitive advantage and earn a high profit with sustained development in the airline industry.

Chang and Yeh (2002) used fuzzy multicriteria analysis modelling to formulate the service quality of airlines, applying fuzzy set theory to describe the ambiguity between the criteria weight and performance ratings of each airline, and they found that flight safety is the most important factor among the 15 attributes used to measure the quality of service. In their work, Chen and Chang (2005) examined the gap between the actual service rendered by airlines and passengers' service expectations in order to determine airlines' service quality. They reported that gaps did exist and that passengers were most interested in the responsiveness and assurance dimensions from airline frontline staff.

Feng and Park (2023) utilized applied machine learning based classifiers on a public data from the national information society of South Korea to predict users' satisfaction and identified 10 essential features that act as indicators in the prediction. Anand and Supriya (2023) utilized PySpark framework approach for airline delay prediction, implementing multiple algorithms and reported that logistic regression and random forest classifies performed better than others.

DATASET DESCRIPTION

The airline passenger satisfaction dataset used in this study was obtained from Kaggle (https://www.kaggle.com/api/v1/datasets/download/mysarahmadbhat/airline-passenger-satisfaction). The dataset contains 24 features, the information about more than 120,000 passengers and their evaluation of different factors such as cleanliness, comfort, service and overall experience. The dataset has 5 categorical columns, 18 integer columns and 1 column of type float. All columns have no null values, except 'Arrival Delay' which has 393 null values. The target variable is the 'Satisfaction' column with 71858 'Neutral/Dissatisfied' instances and 55088 instances of 'Satisfied' passengers. Table 1 shows the description of each variable column.

METHODOLOGY

The growing demand for processing large-scale datasets has necessitated the development of efficient tools like Apache Spark and to leverage the features Hadoop and Spark offers in big data analysis, therefore it is imperative to install and configure the programs.

Hadoop

In order to install the Hadoop, Linux was firstly installed on VirtualBox using Ubuntu. After that, a "temp" folder was created on Desktop from the Terminal and the openjdk version of java installed. Using 'wget', hadoop 3.4.0 version was installed and uncompressed with 'tar -xzf'. The location for Java "export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64" and Hadoop "$ export HADOOP_HOME=/home/Ayobami/Desktop/temp/hadoop-3.4.0" were set and added to path, and the hadoop was installed. The installation confirmation of Hadoop and some of its usage for map reduce are shown in "Appendix A".

Spark

After installing Hadoop 3.4.0, Spark was downloaded and uncompressed with the 'tar –xzf'. The location for the Spark was set "$ export SPARK_HOME=/home/Ayobami/Desktop/temp/spark-3.5.3-bin-hadoop3 and added to path. Java and Hadoop location were also set and added to path prior. The installation confirmation for Spark is shown in "Appendix B".

Data Preprocessing

PySpark, the Python API for Apache Spark, offers robust distributed computing capabilities, making it ideal for handling large datasets and performing advanced machine learning tasks. The PySpark's architecture supports iterative computations and in-memory processing, which significantly accelerates data analysis and machine learning workflows. PySpark's MLlib library includes tools for classification, regression, clustering, and feature transformation, making it a versatile solution for diverse data science tasks.

The dataset was uploaded into PySpark, and an exploratory data analysis process was conducted to assess its structure ("Figure 1"), identify missing values, and evaluate data quality. It was observed that the 'Arrival Delay' column contained 393 missing values, representing about 3% of the total dataset. Given the relatively small proportion of missing data, these rows were removed following best practices in data preprocessing (Kaiser, 2014, p. 44). As revealed by the box plot ("Figure 2"), outliers were removed from the 'Arrival Delay', 'Departure Delay' and 'Flight Distance' columns.

The PySpark session was initialized to enable distributed data processing. This session served as the foundation for managing data transformations and model training. The dataset's Numerical features were directly used in model training, while categorical features were preprocessed for compatibility with machine learning algorithms. The Categorical features were transformation using the PySpark's StringIndexer. The 'Satisfaction' column was designated as the target variable (label) and other columns were combined into a single feature vector using PySpark's VectorAssembler.

The preprocessed dataset was split into training and testing subsets using PySpark's randomSplit method. The training set Contained 80% of the dataset for model training, ensuring sufficient data diversity for learning patterns and the testing set has 20% of the data for unbiased evaluation of model performance.

Model Development

This study employed four popular machine learning models, logistic regression, support vector machines (SVM), random forest, and gradient boosted trees (GBT) to predict customer satisfaction using the Airline Passenger Satisfaction dataset. After splitting the dataset into training and test sets for each model, the models were defined with 'features' for 'featuresCol' and 'label' for 'labelCol'. The training process involved fitting the models to the training set and predictions made with the test set, using the transform method of each model. The hyperparameter optimization was performed by defining grids using ParamGridBuilder and cross validation with the CrossValidator, which utilized the training set to identify the best-performing hypaparameters and predictions from the test set generated using the best model.

The performance of each model was evaluated with the accuracy, precision, recall, and F1-score metrics. Additionally, feature importance analysis was conducted, the absolute coefficients values were ranked to determine the important features for logistic regression and SVM models and the built-in importance scores used to rank the important features of random forest and GBT.

Logistic regression

The logistic function is a mathematical function which converts the output of linear regression into a probability score between 0 and 1, where the score represents the likelihood of the event being predicted (Anand and Supriya, 2023, p. 3). Logistic Regression is a statistical approach to analyse methods of predicting a data value that is entirely based on the observations and is suitable for interpreting the influence of input features on the output (Sen et al., 2023*)*.

SVM

Support vector machine (SVM) is a statistical learning theory that seeks to identify the hyperplane that most effectively divides instances. It is excellent at defining ideal decision borders because it can find data points that affect where the boundary between classes is placed. SVM's performance might vary depending on the parameters selected and the features of the dataset (Haque and Hassan 2023, p. 5)

Random Forest

Random forest classification is based on concept of ensemble learning, a technique for integrating many classifiers to handle tough problems. It contains a number of decision trees on various subsets of a given dataset and takes the average to improve the predictive accuracy of that dataset. The outcome depends on majority votes of projections rather than relying solely on one decision tree (Viswanatha et al., 2023, p. 9).

Gradient Boosting Trees (GBT)

Gradient boosting tree, known for its ability to handle complex relationships and large feature spaces is a method where trees are built in series and compared to each other based on mathematically scores of splits which combines the principles of gradient boosting with decision trees. An ensemble of decision trees are built sequentially and each subsequent tree corrects the mistakes of the previous tree which iteratively improves the model's predictions by minimizing a loss function using gradient descent optimization. Therefore, the GBT creates a strong predictive model by combining multiple weak decision tress to classify observations accurately (Anand and Supriya, 2023, p 3).

RESULTS

Model Performance Metrics

The models' performances were analysed using accuracy, precision, recall, and F1-score as performance metrics. Each model underwent both baseline and hyperparameter-tuned evaluations to enhance performance.

All the models achieved high accuracy ("Figure 3") and were excellent at predicting both classes ("Figure 4"). The GBT model (95%) particularly outperformed others, followed by random forest (92%). The SVM and logistic regression models had identical performance metrics (87%), indicating similar behaviour in predicting customers' satisfaction from the dataset. Hyperparameter tuning improved the performance of the random forest and GBT models to 96%, demonstrating the impact of optimized parameter selection on ensemble methods. While logistic regression and SVM demonstrated comparable performance, they were however significantly outperformed by random forest and GBT. This performance gap highlights the advantage of ensemble learning techniques in capturing complex feature interactions and their robustness in handling diverse datasets (Wei et al, 2023, p. 349).

The models were uniformly effective across the evaluation metrics, as it was observed that the values of accuracy, precision, recall, and F1-score were identical in each model, highlighting the reliability of the modelling process and the suitability of the dataset for the predictive tasks. This comprehensive analysis demonstrates the effectiveness of machine learning models in predicting customer satisfaction while identifying actionable insights through feature importance.

Feature Importance Analysis

Feature importance was analysed ("Figure 5") for each model to understand the factors contributing most significantly to customer satisfaction predictions. Features like 'On board Wifi Service' and 'Class' emerged as critical predictors across all models, Logistic Regression and SVM gave utmost priority to 'Type of Travel' and 'Customer Type' while the ensemble methods (Random Forest and Gradient Boosted Trees) gave highest importance to 'Online Boarding', which indicate passenger satisfaction with technological services.

Conclusion

In this study, machine learning models were applied using PySpark, to predict customer satisfaction from airline passenger data. The dataset, comprising various features that reflected customers' experiences, which enabled the exploration of predictive performance with logistic regression, support vector machine (SVM), random forest, and gradient boosted trees (GBT) models. Comprehensive preprocessing steps which include handling missing values, indexing categorical features, and vectorizing the predictors ensured that the data was suitably prepared for analysis.

The results demonstrated that ensemble methods, particularly GBT, outperformed other models, whereas logistic regression and SVM maintained consistent metrics across all evaluations. The consistent performance of logistic regression and SVM can be attributed to their linear structures, which suited the dataset's characteristics.

Feature importance analysis revealed that customer centric attributes, such as 'travel type', 'customer type', and 'onboard services', were significant predictors of satisfaction across all the models. Ensemble methods particularly emphasized features linked to technological engagement, such as 'online boarding' and inflight entertainment, indicating these elements play a critical role in modern customer satisfaction.

Overall, this study has shown the effectiveness of machine learning models using PySpark for large datasets to provide robust, scalable solutions for predictive analytics. While ensemble methods exhibited superior performance, simpler models like logistic regression and SVM remain valuable for interpretability and computational efficiency in less complex scenarios. This study has therefore shown that businesses can understand and anticipate customer needs better by leveraging big data analytics, which would help foster improved customer satisfaction and in return, loyalty.

Future work could explore other algorithms and include additional metrics to deepen insights into model behaviour and performance nuances. Also, more granular analysis is recommended to further refine and explore subtle differences between the models' performances.

REFERENCES

Anand, R. N., & Supriya, M. (2023). Enhancing Airline Operations by Flight Delay Prediction - A PySpark Framework Approach. *2023 International Conference on Ambient Intelligence, Knowledge Informatics and Industrial Electronics*, 1–4. https://doi.org/10.1109/aikiie60097.2023.10389960

Chang, W.-J., Chen, G.-J., & Yeh, Y.-L. (2002). Fuzzy Control of Dynamic Positioning Systems for Ships. *Journal of Marine Science and Technology*, *10*(1). https://doi.org/10.51400/2709-6998.2300

Chen, C.-F. (2008). Investigating structural relationships between service quality, perceived value, satisfaction, and behavioral intentions for air passengers: Evidence from Taiwan. *Transportation Research Part A: Policy and Practice*, *42*(4), 709–717. https://doi.org/10.1016/j.tra.2008.01.007

Chen, F.-Y., & Chang, Y.-H. (2005). Examining airline service quality from a process perspective. *Journal of Air Transport Management*, *11*(2), 79–87. https://doi.org/10.1016/j.jairtraman.2004.09.002

Clemes, M. D., Gan, C., Kao, T.-H., & Choong, M. (2008). An empirical analysis of customer satisfaction in international air travel. *Innovative Marketing*, *4*(2), 48–62.

Feng, Y., & Park, J. (2023). Using machine learning-based binary classifiers for predicting organizational members' user satisfaction with collaboration software. *PeerJ Computer Science*, *9*, *e1481*.

Haque, F. M., & Hassan, M. M. (n.d.). Bank Loan Prediction Using Machine Learning Techniques. *ArXiv Preprint ArXiv:2410.08886*.

Kaiser, J. (2014). Dealing with Missing Values in Data. *Journal of Systems Integration*, *5*(1), 42–51. https://doi.org/10.20470/jsi.v5i1.178

Liou, J. J. H., & Tzeng, G.-H. (2007). A non-additive model for evaluating airline service quality. *Journal of Air Transport Management*, *13*(3), 131–138. https://doi.org/10.1016/j.jairtraman.2006.12.002

Malik, H., Sronach, R. L., & Witzig, S. B. (n.d.). Nature photography: a spark for discussion and action on environmental issues. *Journal of Outdoor and Environmental Education*, *1-23*.

Shah, S. S. A., & Khan, Z. (2019). Corporate social responsibility: a pathway to sustainable competitive advantage? *International Journal of Bank Marketing*, *ahead-of-print*(ahead-of-print), 159–174. https://doi.org/10.1108/ijbm-01-2019-0037

Viswanatha, V., Ramachandra, A. C., Vishwas, K. N., & Adithya, G. (2023). LOAN APPROVAL PREDICTION USING MACHINE LEARNING. *International Research Journal of Modernization in Engineering Technology and Science*, *13*(4), 7–19. https://doi.org/10.56726/irjmets39658

Wei, Y., Zhang, D., Gao, M., Tian, Y., He, Y., Huang, B., & Zheng, C. (2023). Breast cancer prediction based on machine learning. *Journal of Software Engineering and Applications*, *16*(18), 348–360.

Table 1. Data Description

Variable Name	Description
ID	Unique passenger identifier
Gender	Gender of the passenger (Female/Male)
Age	Age of the passenger
Customer Type	Type of airline customer (First-time/Returning)
Type of Travel	Purpose of the flight (Business/Personal)
Class	Travel class in the airplane for the passenger seat
Flight Distance	Flight distance in miles
Departure Delay	Flight departure delay in minutes
Arrival Delay	Flight arrival delay in minutes
Departure and Arrival Time Convenience	Satisfaction level with the convenience of the flight departure and arrival times from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Ease of Online Booking	Satisfaction level with the online booking experience from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Check-in Service	Satisfaction level with the check-in service from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Online Boarding	Satisfaction level with the online boarding experience from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Gate Location	Satisfaction level with the gate location in the airport from 1 (lowest) to 5 (highest) - 0 means "not applicable"
On-board Service	Satisfaction level with the on-boarding service in the airport from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Seat Comfort	Satisfaction level with the comfort of the airplane seat from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Leg Room Service	Satisfaction level with the leg room of the airplane seat from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Cleanliness	Satisfaction level with the cleanliness of the airplane from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Food and Drink	Satisfaction level with the food and drinks on the airplane from 1 (lowest) to 5 (highest) - 0 means "not applicable"
In-flight Service	Satisfaction level with the in-flight service from 1 (lowest) to 5 (highest) - 0 means "not applicable"
In-flight Wifi Service	Satisfaction level with the in-flight Wifi service from 1 (lowest) to 5 (highest) - 0 means "not applicable"
In-flight Entertainment	Satisfaction level with the in-flight entertainment from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Baggage Handling	Satisfaction level with the baggage handling from the airline from 1 (lowest) to 5 (highest) - 0 means "not applicable"
Satisfaction	Overall satisfaction level with the airline (Satisfied/Neutral or unsatisfied)

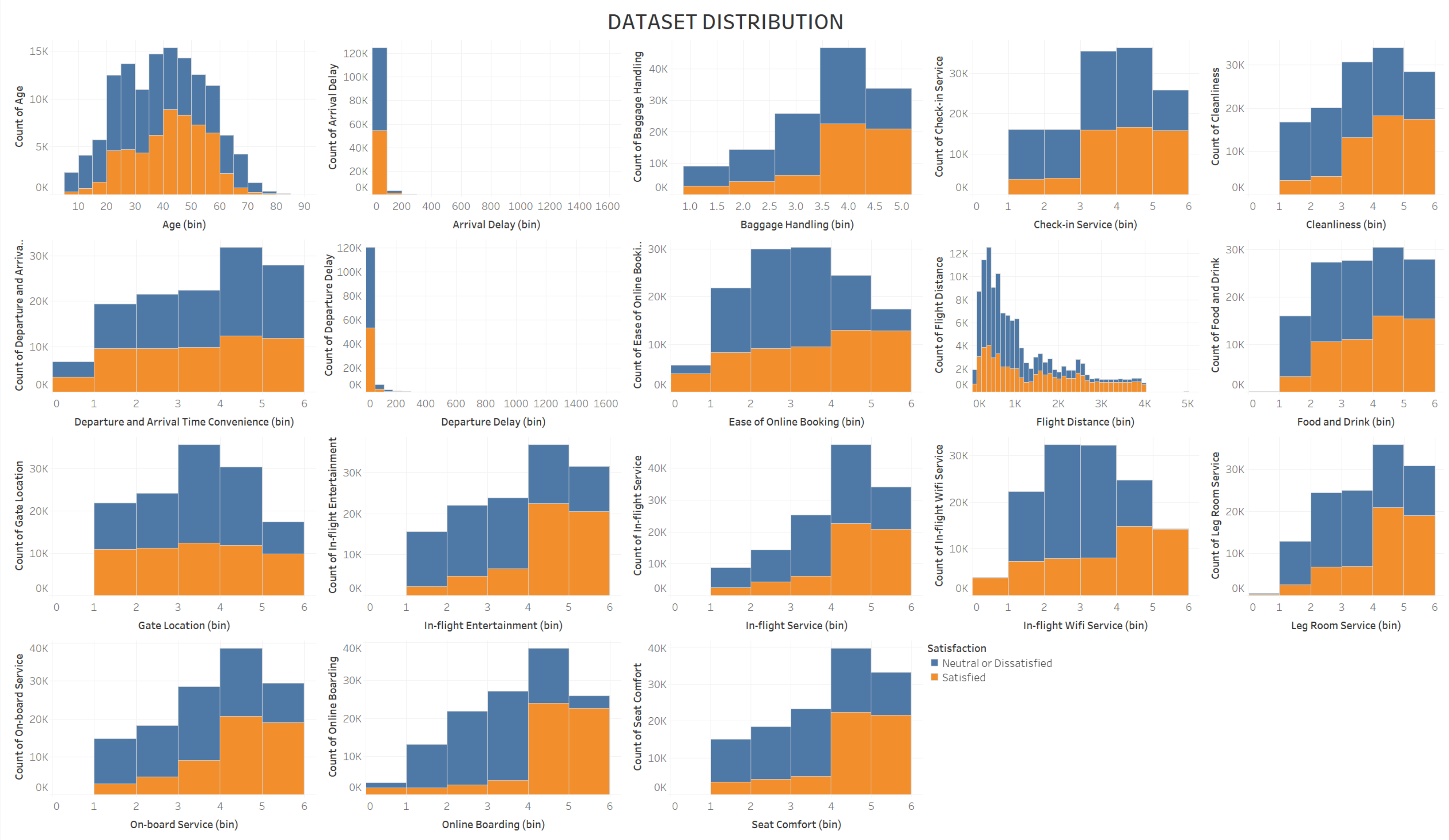


Fig. 1 Histogram showing distribution of the numerical features by 'Satisfaction'

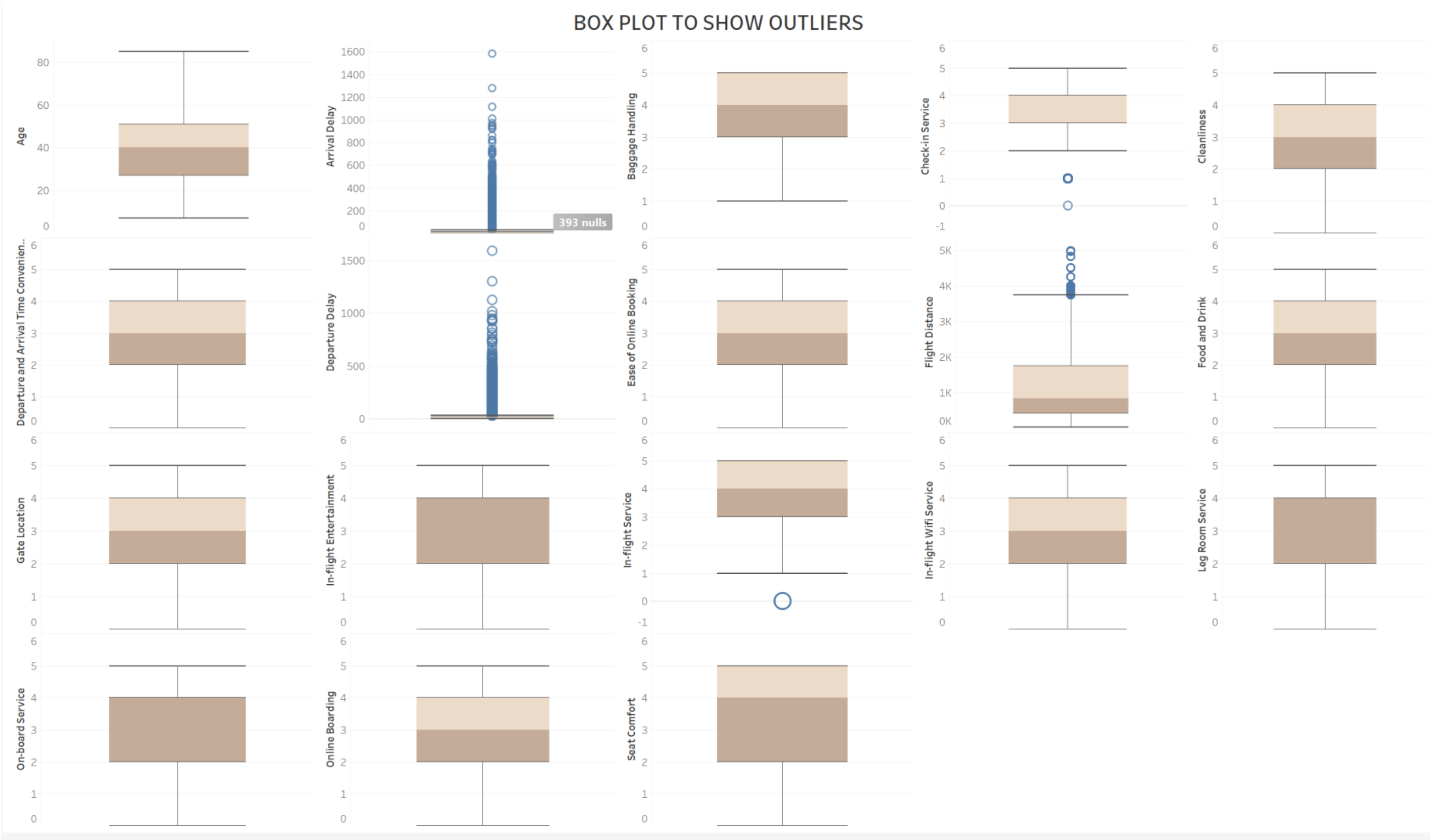


Fig. 2. Box plot showing outliers

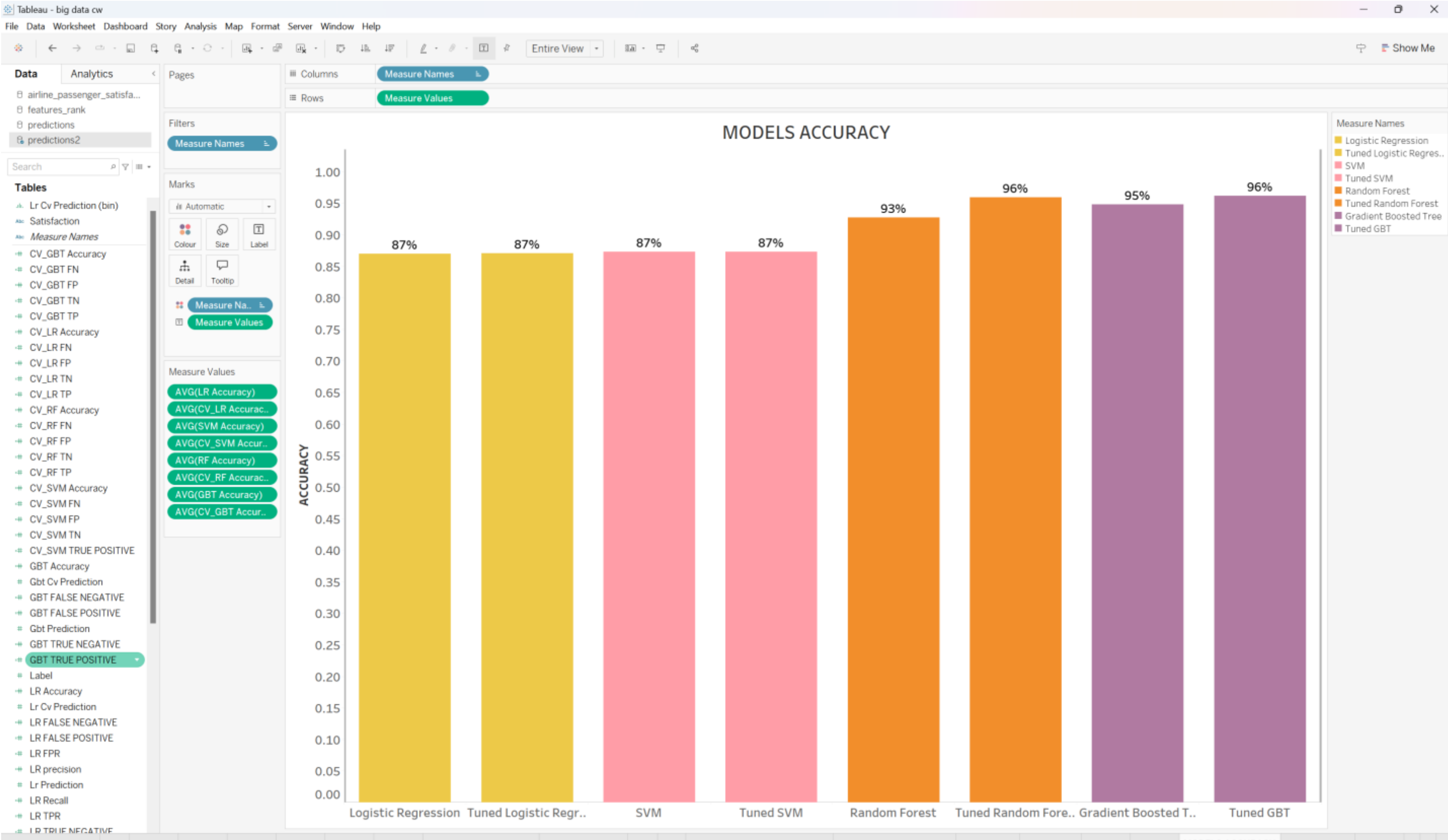


Fig 3. Models' Accuracy

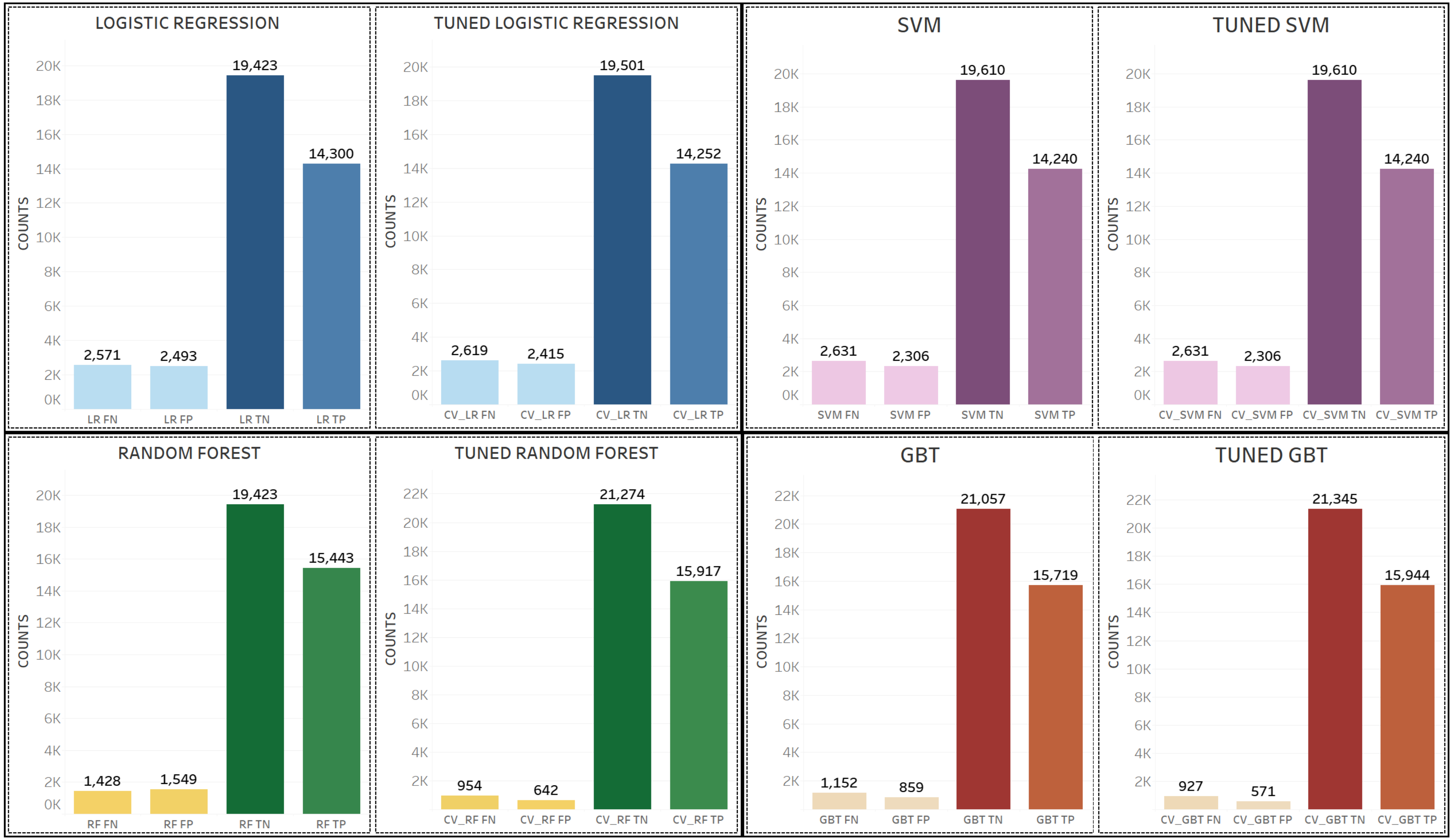
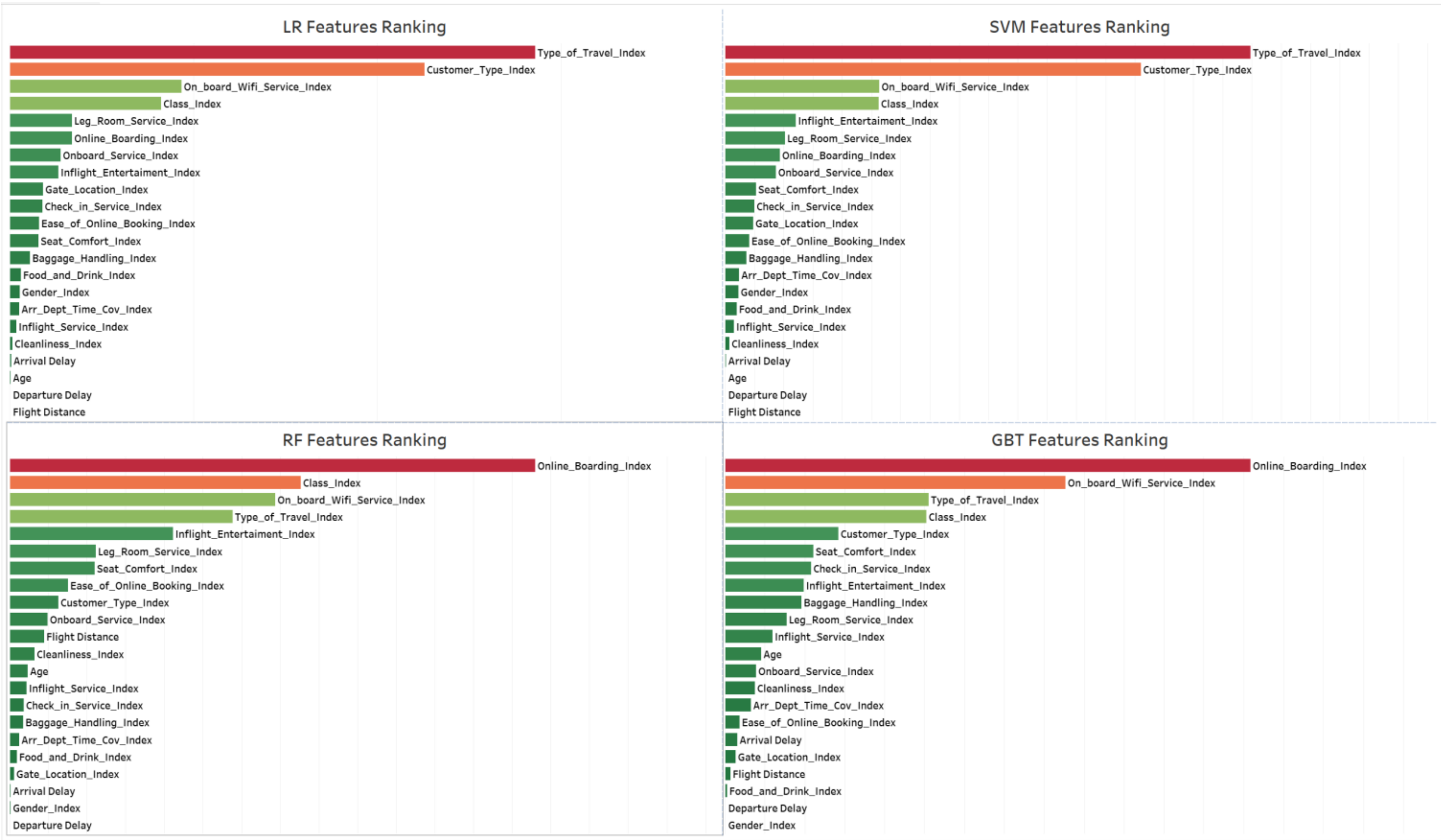
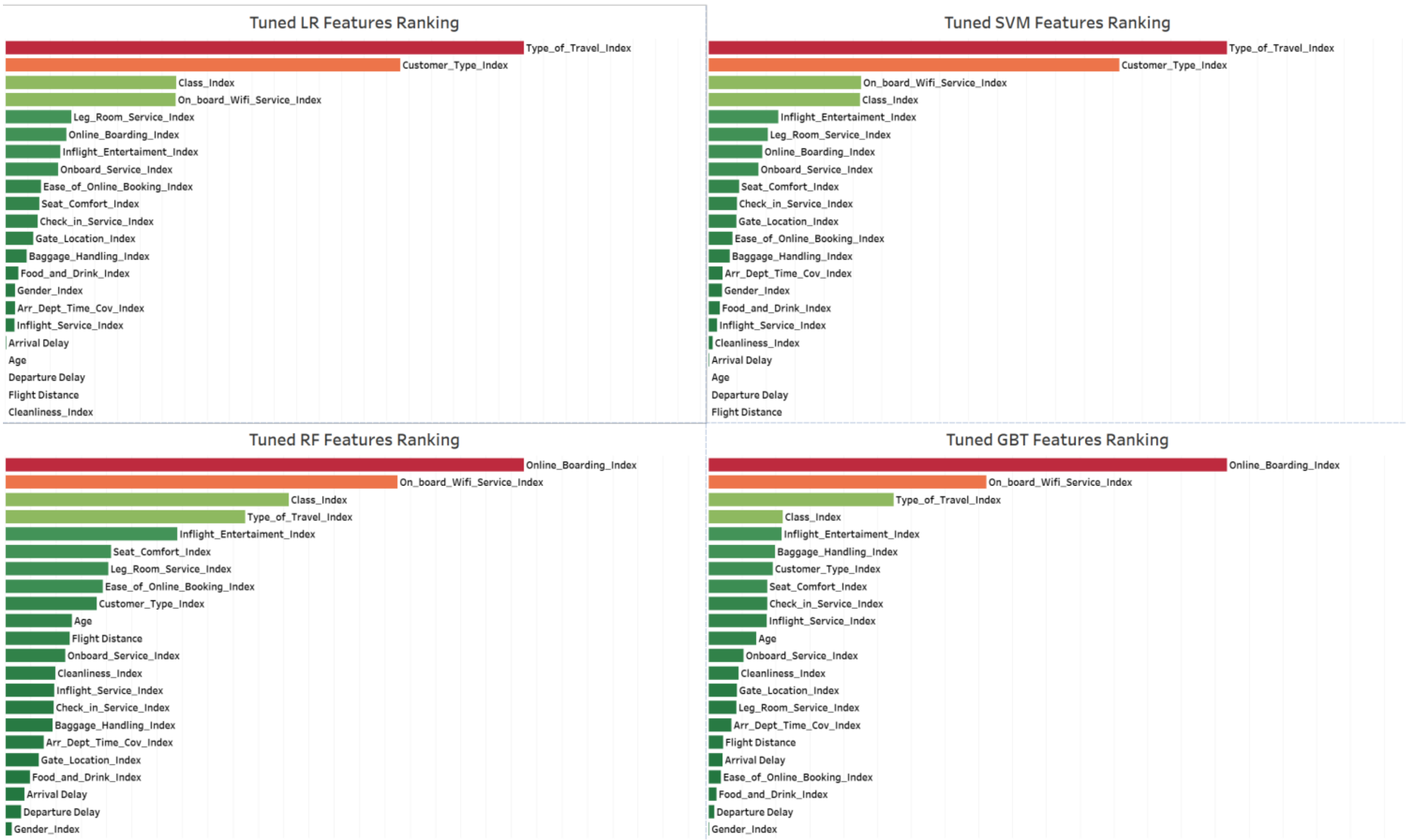


Fig. 4. Models' Predictions



(a) Normal models

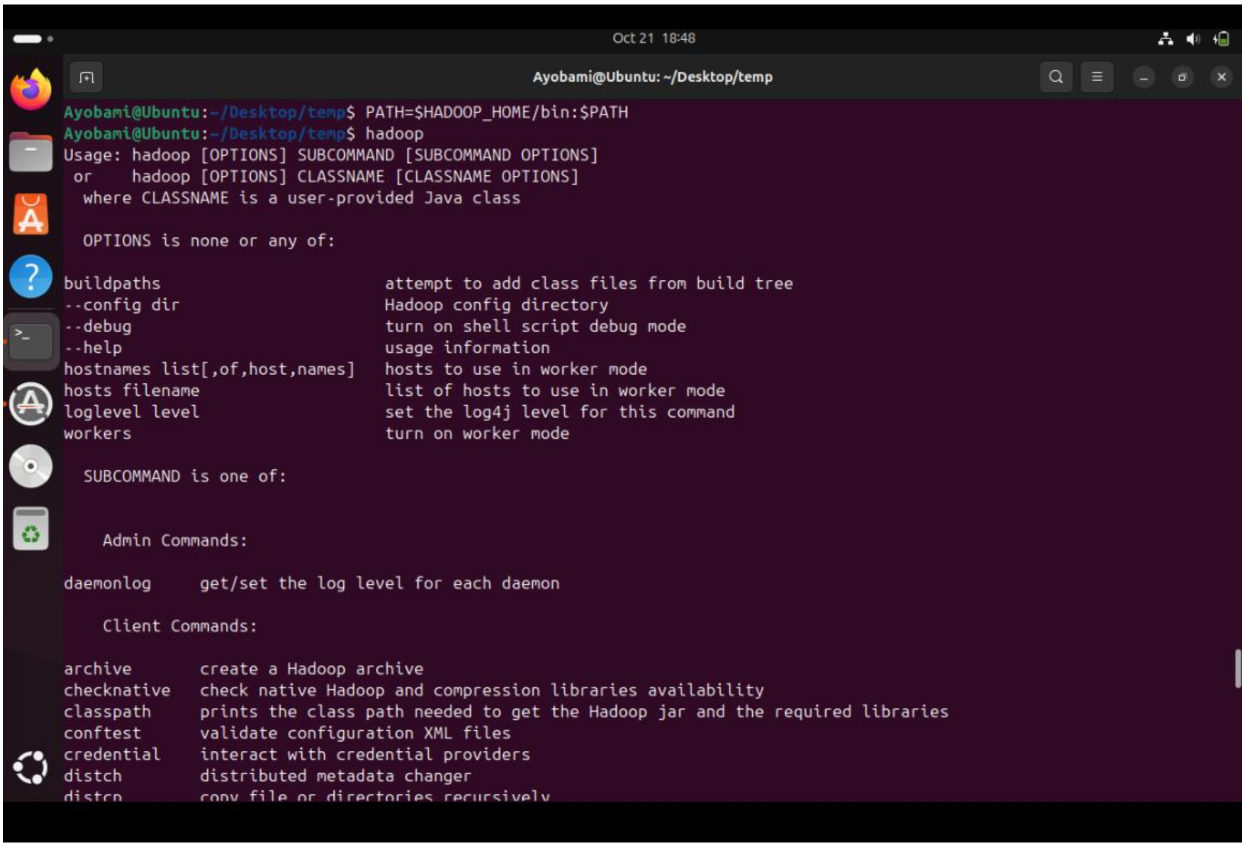


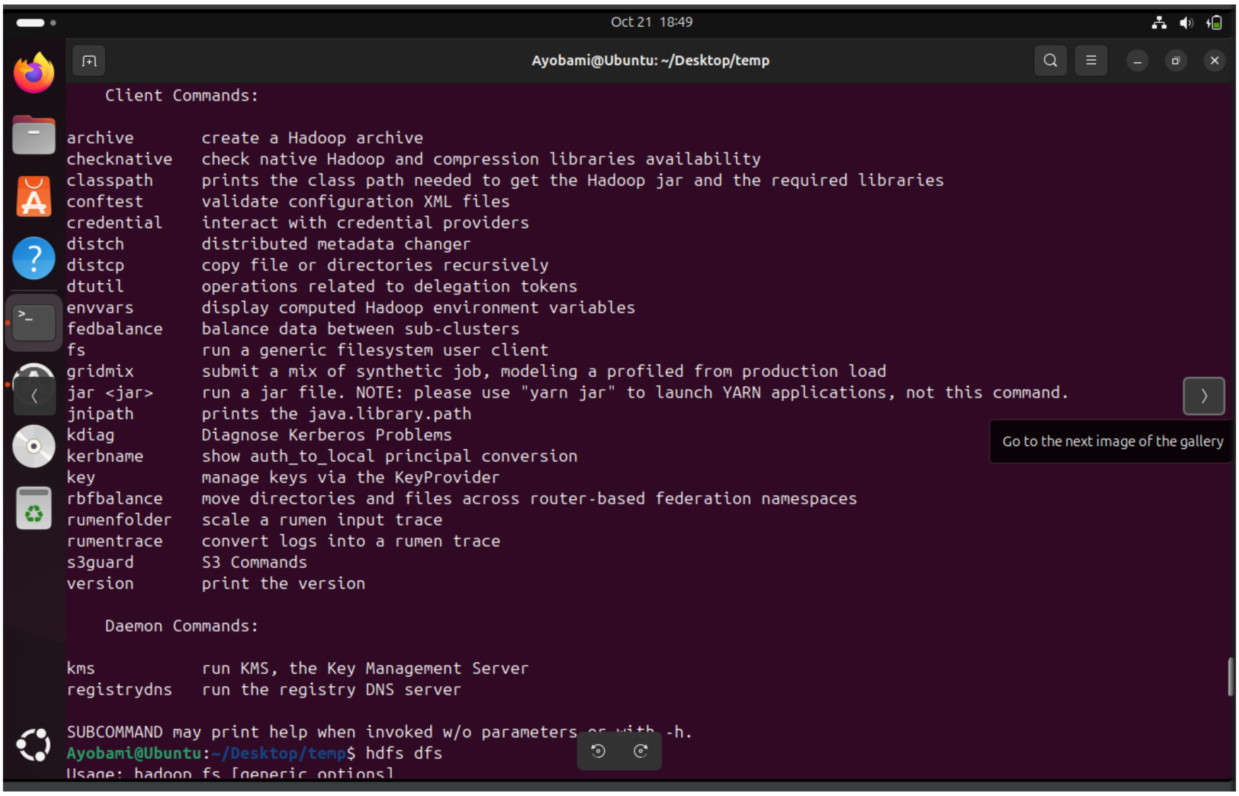
(b) Optimized models

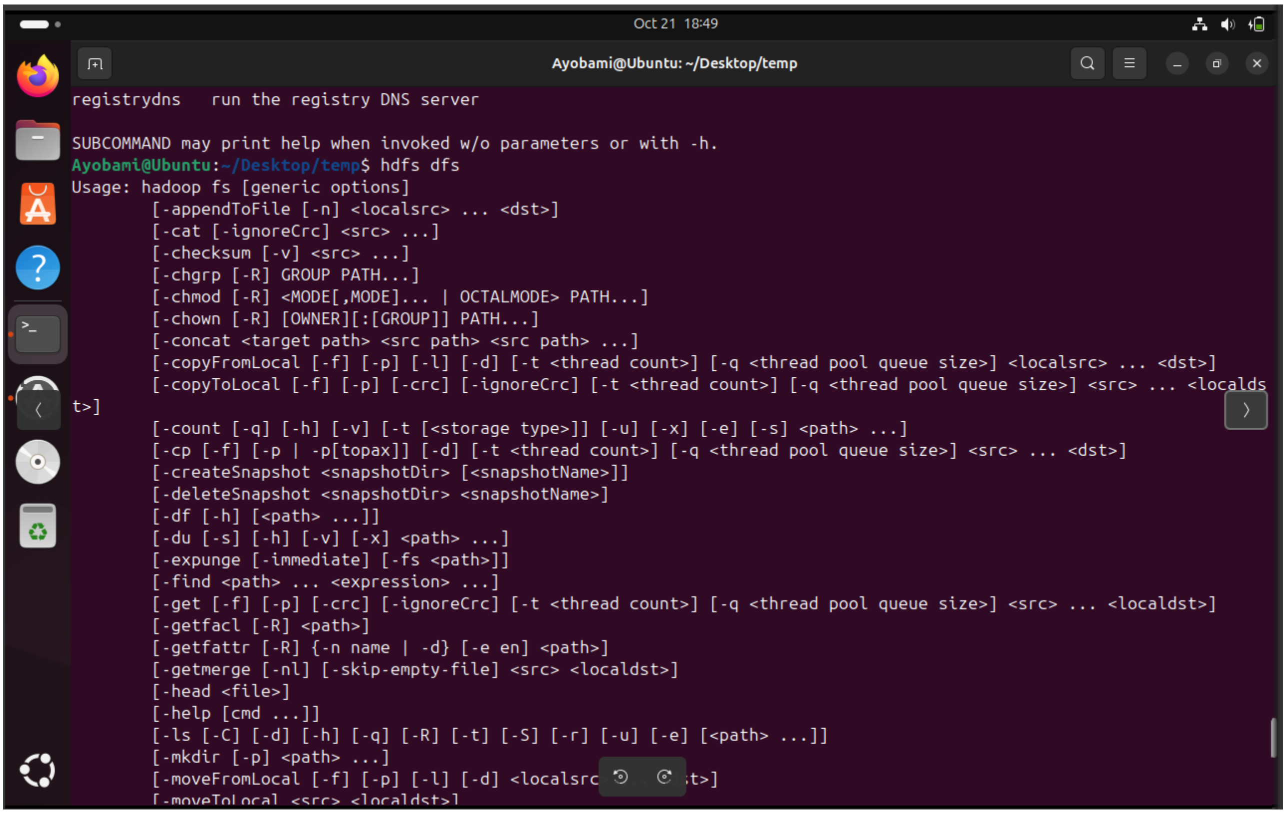
Fig. 5. Important Features

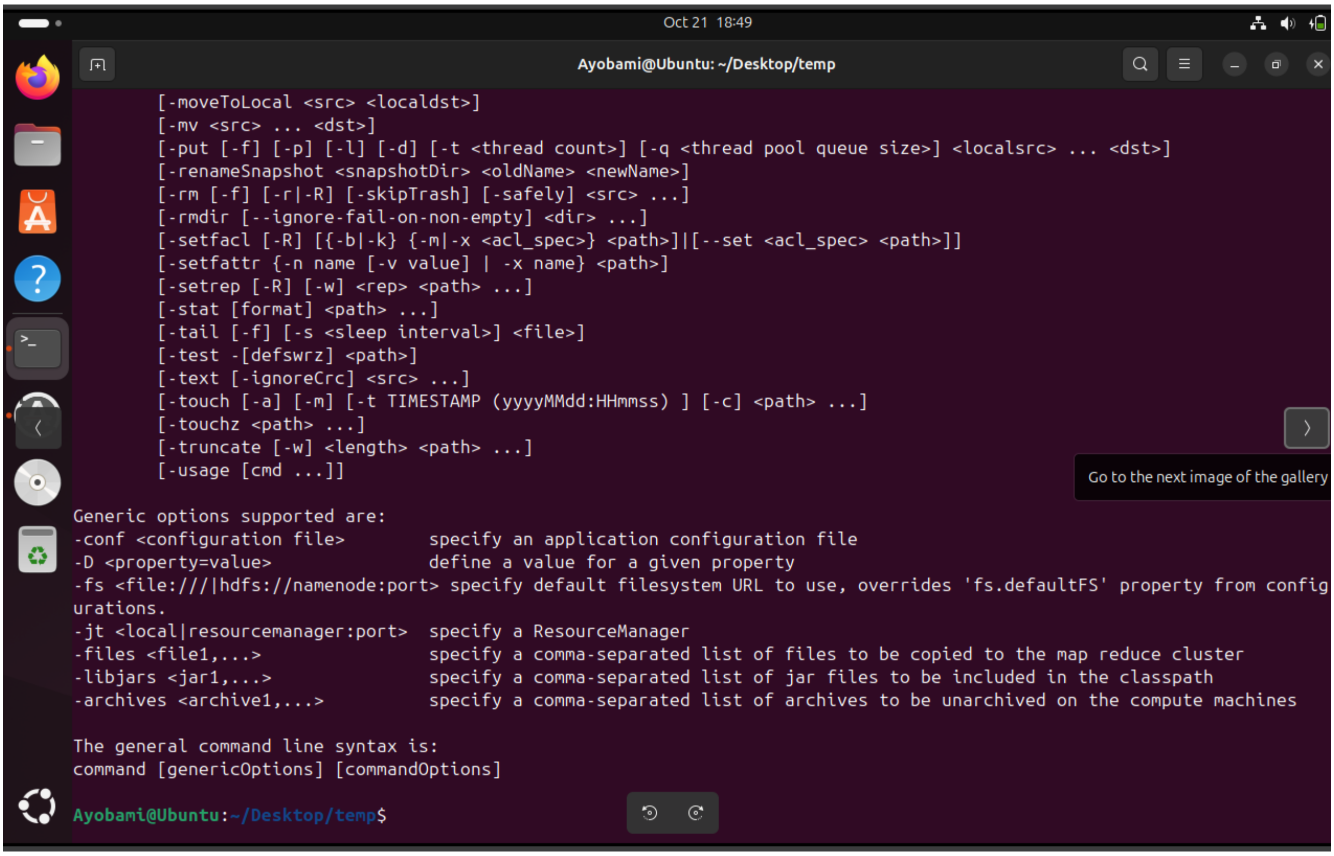
APPENDIX A

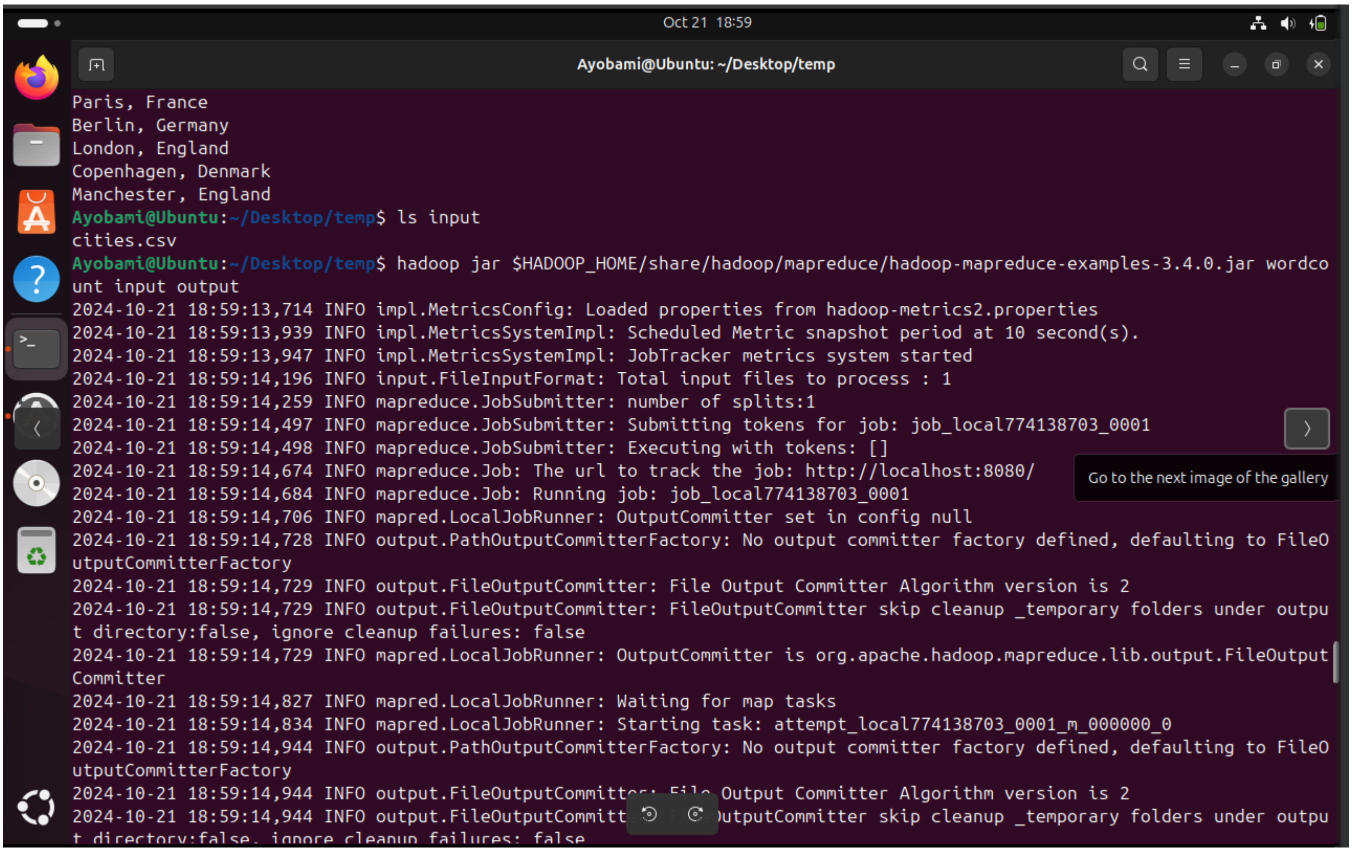
HADOOP

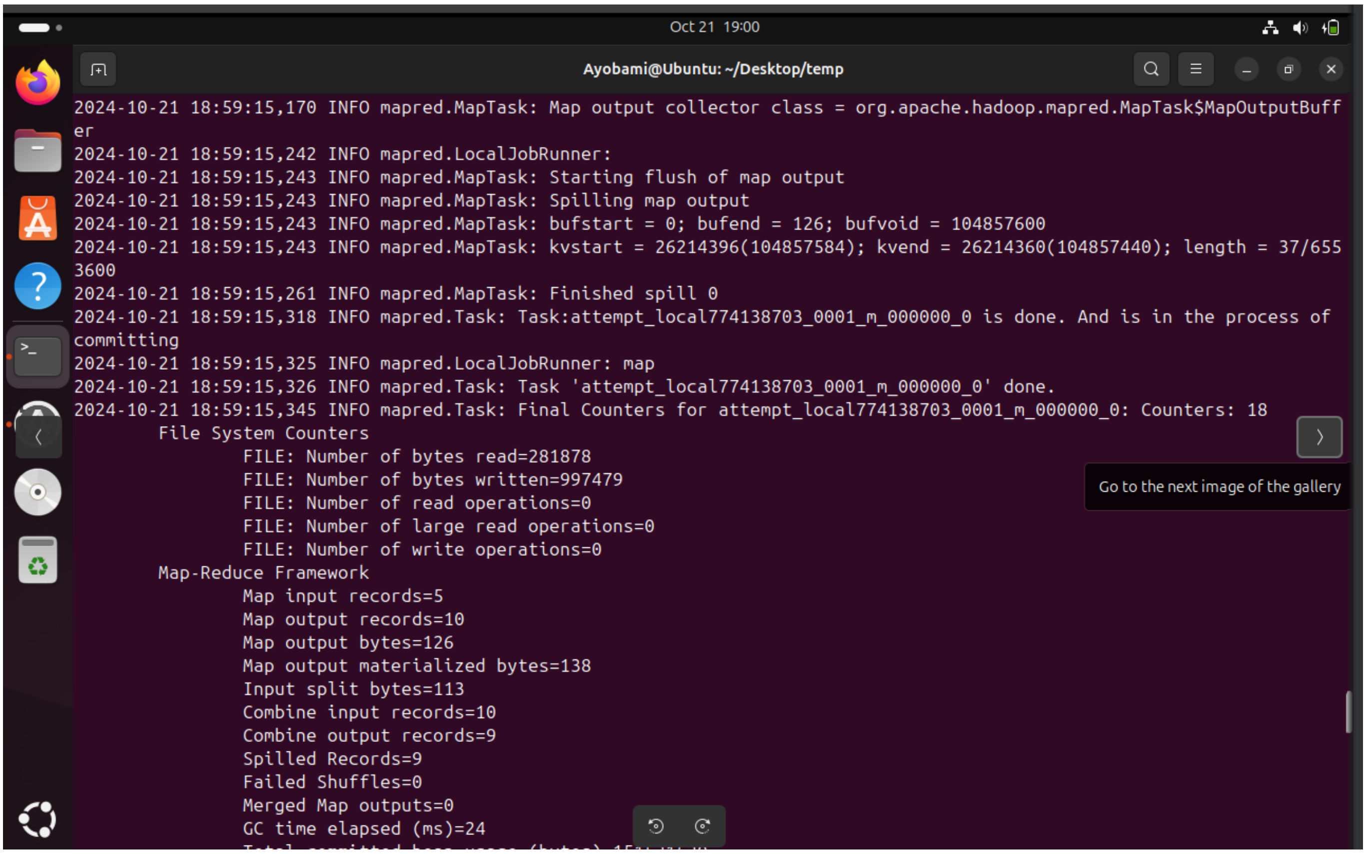


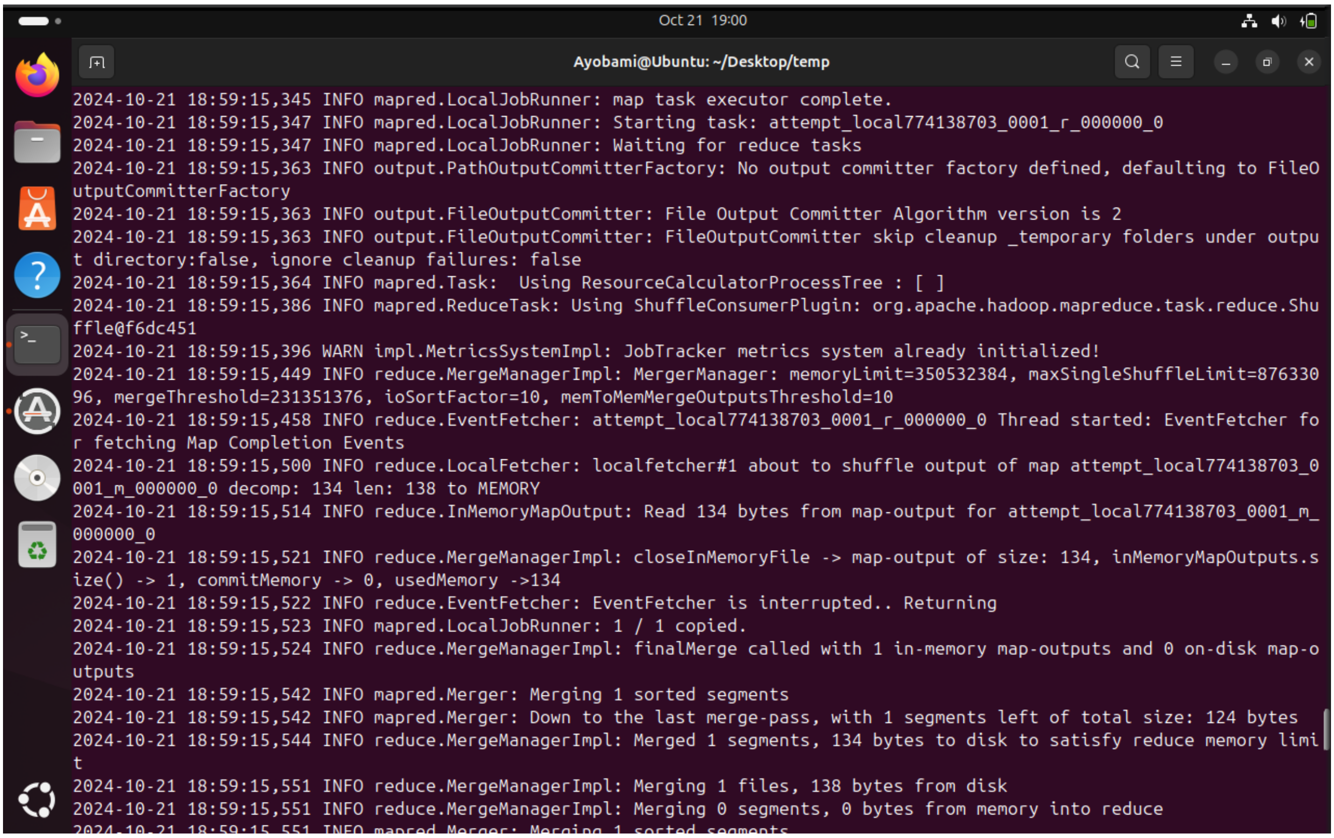


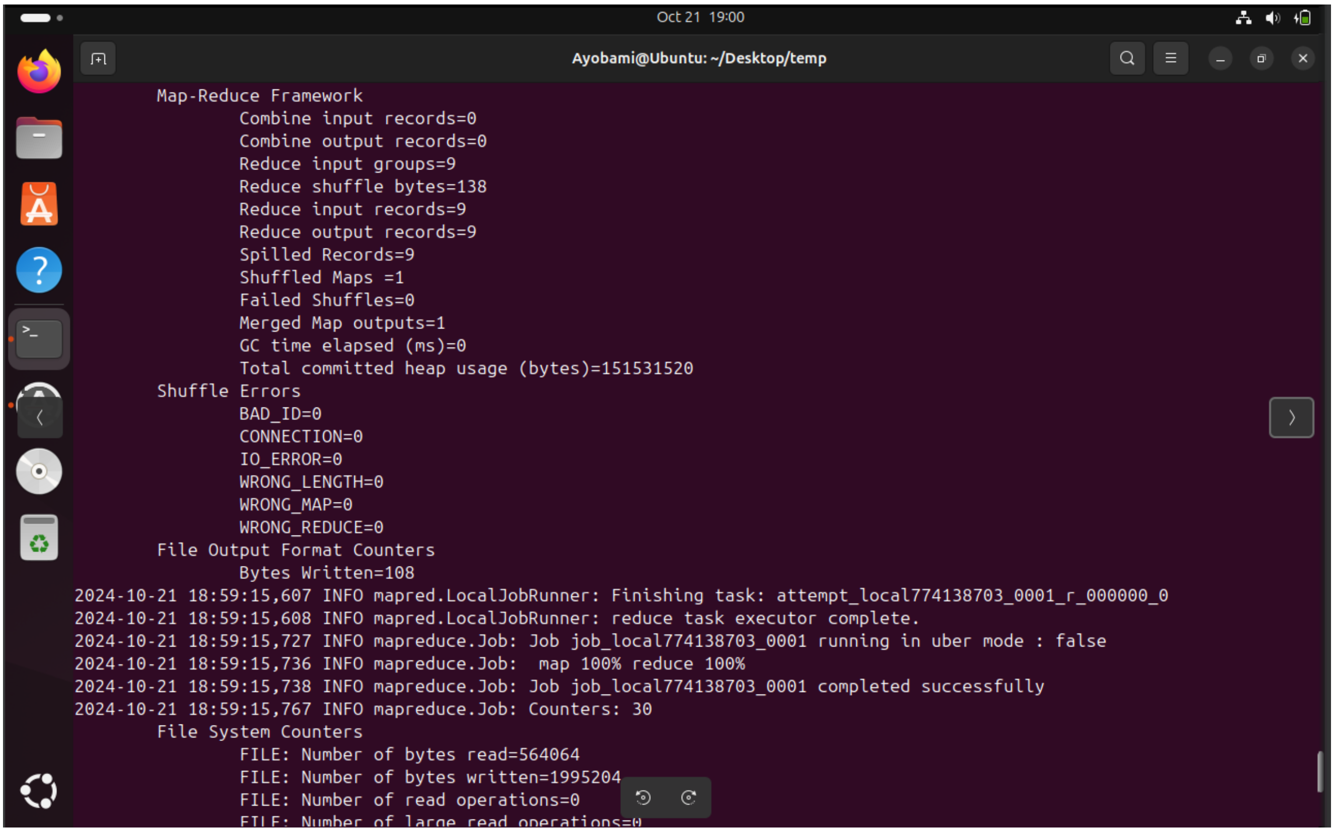

Oct 21 18:49
Ayobami@Ubuntu: ~/Desktop/temp
registrydns   run the registry DNS server
SUBCOMMAND may print help when invoked w/o parameters or with -h.
Ayobami@Ubuntu:~/Desktop/temp$ hdfs dfs
Usage: hadoop fs [generic options]
[-appendToFile [-n] <localsrc> ... <dst>]
[-cat [-ignoreCrc] <src> ...]
[-checksum [-v] <src> ...]
[-chgrp [-R] GROUP PATH...]
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
[-chown [-R] [OWNER][:[GROUP]] PATH...]
[-concat <target path> <src path> <src path> ...]
[-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] [-q <thread pool queue size>] <localsrc> ... <dst>]
[-copyToLocal [-f] [-p] [-crc] [-ignoreCrc] [-t <thread count>] [-q <thread pool queue size>] <src> ... <localds
t>]
[-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] [-e] [-s] <path> ...]
[-cp [-f] [-p | -p[topax]] [-d] [-t <thread count>] [-q <thread pool queue size>] <src> ... <dst>]
[-createSnapshot <snapshotDir> [<snapshotName>]]
[-deleteSnapshot <snapshotDir> <snapshotName>]
[-df [-h] [<path> ...]]
[-du [-s] [-h] [-v] [-x] <path> ...]
[-expunge [-immediate] [-fs <path>]]
[-find <path> ... <expression> ...]
[-get [-f] [-p] [-crc] [-ignoreCrc] [-t <thread count>] [-q <thread pool queue size>] <src> ... <localdst>]
[-getfacl [-R] <path>]
[-getfattr [-R] {-n name | -d} [-e en] <path>]
[-getmerge [-nl] [-skip-empty-file] <src> <localdst>]
[-head <file>]
[-help [cmd ...]]
[-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path> ...]]
[-mkdir [-p] <path> ...]
[-moveFromLocal [-f] [-p] [-l] [-d] <localsrc t>]

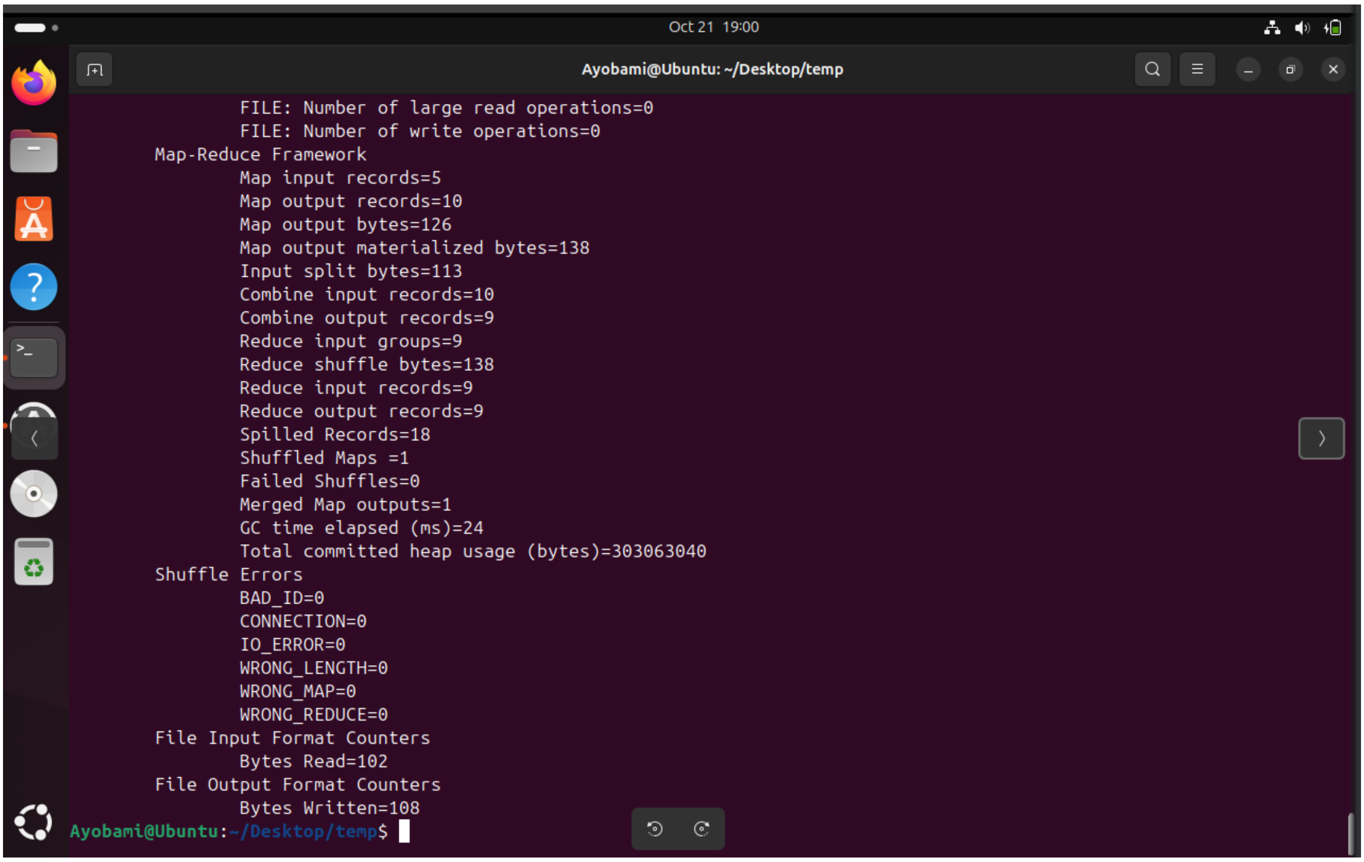

Oct 21 18:49
Ayobami@Ubuntu: ~/Desktop/temp
[-moveToLocal <src> <localdst>]
[-mv <src> ... <dst>]
[-put [-f] [-p] [-l] [-d] [-t <thread count>] [-q <thread pool queue size>] <localsrc> ... <dst>]
[-renameSnapshot <snapshotDir> <oldName> <newName>]
[-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...]
[-rmdir [--ignore-fail-on-non-empty] <dir> ...]
[-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set <acl_spec> <path>]]
[-setfattr {-n name [-v value] | -x name} <path>]
[-setrep [-R] [-w] <rep> <path> ...]
[-stat [format] <path> ...]
[-tail [-f] [-s <sleep interval>] <file>]
[-test -[defswrz] <path>]
[-text [-ignoreCrc] <src> ...]
[-touch [-a] [-m] [-t TIMESTAMP (yyyyMMdd:HHmmss)] [-c] <path> ...]
[-touchz <path> ...]
[-truncate [-w] <length> <path> ...]
[-usage [cmd ...]]
Go to the next image of the gallery
Generic options supported are:
-conf <configuration file>   specify an application configuration file
-D <property=value>   define a value for a given property
-fs <file:///|hdfs://namenode:port> specify default filesystem URL to use, overrides 'fs.defaultFS' property from config
urations.
-jt <local|resourcemanager:port>   specify a ResourceManager
-files <file1,...>   specify a comma-separated list of files to be copied to the map reduce cluster
-libjars <jar1,...>   specify a comma-separated list of jar files to be included in the classpath
-archives <archive1,...>   specify a comma-separated list of archives to be unarchived on the compute machines
The general command line syntax is:
command [genericOptions] [commandOptions]
Ayobami@Ubuntu:~/Desktop/temp$

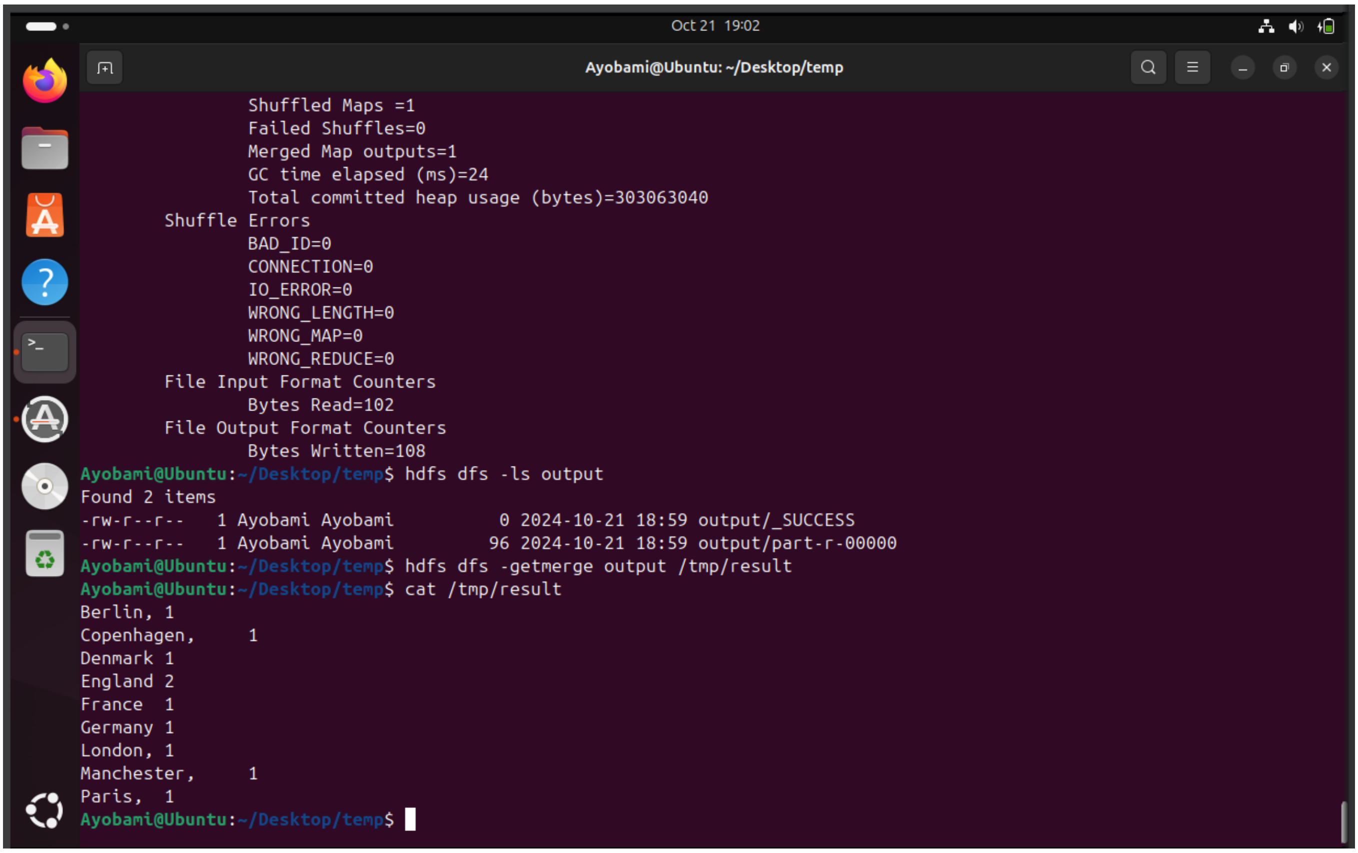

Oct 21 18:59
Ayobami@Ubuntu: ~/Desktop/temp
Paris, France
Berlin, Germany
London, England
Copenhagen, Denmark
Manchester, England
Ayobami@Ubuntu:~/Desktop/temp$ ls input
cities.csv
Ayobami@Ubuntu:~/Desktop/temp$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.4.0.jar wordcount input output
2024-10-21 18:59:13,714 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2024-10-21 18:59:13,939 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2024-10-21 18:59:13,947 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2024-10-21 18:59:14,196 INFO input.FileInputFormat: Total input files to process : 1
2024-10-21 18:59:14,259 INFO mapreduce.JobSubmitter: number of splits:1
2024-10-21 18:59:14,497 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local774138703_0001
2024-10-21 18:59:14,498 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-10-21 18:59:14,674 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
Go to the next image of the gallery
2024-10-21 18:59:14,684 INFO mapreduce.Job: Running job: job_local774138703_0001
2024-10-21 18:59:14,706 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2024-10-21 18:59:14,728 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2024-10-21 18:59:14,729 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-10-21 18:59:14,729 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2024-10-21 18:59:14,729 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
2024-10-21 18:59:14,827 INFO mapred.LocalJobRunner: Waiting for map tasks
2024-10-21 18:59:14,834 INFO mapred.LocalJobRunner: Starting task: attempt_local774138703_0001_m_000000_0
2024-10-21 18:59:14,944 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2024-10-21 18:59:14,944 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-10-21 18:59:14,944 INFO output.FileOutputCommitt utputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false


Oct 21 19:00
Ayobami@Ubuntu: ~/Desktop/temp
2024-10-21 18:59:15,170 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
2024-10-21 18:59:15,242 INFO mapred.LocalJobRunner:
2024-10-21 18:59:15,243 INFO mapred.MapTask: Starting flush of map output
2024-10-21 18:59:15,243 INFO mapred.MapTask: Spilling map output
2024-10-21 18:59:15,243 INFO mapred.MapTask: bufstart = 0; bufend = 126; bufvoid = 104857600
2024-10-21 18:59:15,243 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 26214360(104857440); length = 37/6553600
2024-10-21 18:59:15,261 INFO mapred.MapTask: Finished spill 0
2024-10-21 18:59:15,318 INFO mapred.Task: Task:attempt_local774138703_0001_m_000000_0 is done. And is in the process of committing
2024-10-21 18:59:15,325 INFO mapred.LocalJobRunner: map
2024-10-21 18:59:15,326 INFO mapred.Task: Task 'attempt_local774138703_0001_m_000000_0' done.
2024-10-21 18:59:15,345 INFO mapred.Task: Final Counters for attempt_local774138703_0001_m_000000_0: Counters: 18
File System Counters
FILE: Number of bytes read=281878
FILE: Number of bytes written=997479
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
Go to the next image of the gallery
Map-Reduce Framework
Map input records=5
Map output records=10
Map output bytes=126
Map output materialized bytes=138
Input split bytes=113
Combine input records=10
Combine output records=9
Spilled Records=9
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=24


Oct 21 19:00
Ayobami@Ubuntu: ~/Desktop/temp
2024-10-21 18:59:15,345 INFO mapred.LocalJobRunner: map task executor complete.
2024-10-21 18:59:15,347 INFO mapred.LocalJobRunner: Starting task: attempt_local774138703_0001_r_000000_0
2024-10-21 18:59:15,347 INFO mapred.LocalJobRunner: Waiting for reduce tasks
2024-10-21 18:59:15,363 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2024-10-21 18:59:15,363 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-10-21 18:59:15,363 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2024-10-21 18:59:15,364 INFO mapred.Task: Using ResourceCalculatorProcessTree : [ ]
2024-10-21 18:59:15,386 INFO mapred.ReduceTask: Using ShuffleConsumerPlugin: org.apache.hadoop.mapreduce.task.reduce.Shuffle@f6dc451
2024-10-21 18:59:15,396 WARN impl.MetricsSystemImpl: JobTracker metrics system already initialized!
2024-10-21 18:59:15,449 INFO reduce.MergeManagerImpl: MergerManager: memoryLimit=350532384, maxSingleShuffleLimit=87633096, mergeThreshold=231351376, ioSortFactor=10, memToMemMergeOutputsThreshold=10
2024-10-21 18:59:15,458 INFO reduce.EventFetcher: attempt_local774138703_0001_r_000000_0 Thread started: EventFetcher for fetching Map Completion Events
2024-10-21 18:59:15,500 INFO reduce.LocalFetcher: localfetcher#1 about to shuffle output of map attempt_local774138703_0001_m_000000_0 decomp: 134 len: 138 to MEMORY
2024-10-21 18:59:15,514 INFO reduce.InMemoryMapOutput: Read 134 bytes from map-output for attempt_local774138703_0001_m_000000_0
2024-10-21 18:59:15,521 INFO reduce.MergeManagerImpl: closeInMemoryFile -> map-output of size: 134, inMemoryMapOutputs.size() -> 1, commitMemory -> 0, usedMemory ->134
2024-10-21 18:59:15,522 INFO reduce.EventFetcher: EventFetcher is interrupted.. Returning
2024-10-21 18:59:15,523 INFO mapred.LocalJobRunner: 1 / 1 copied.
2024-10-21 18:59:15,524 INFO reduce.MergeManagerImpl: finalMerge called with 1 in-memory map-outputs and 0 on-disk map-outputs
2024-10-21 18:59:15,542 INFO mapred.Merger: Merging 1 sorted segments
2024-10-21 18:59:15,542 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 124 bytes
2024-10-21 18:59:15,544 INFO reduce.MergeManagerImpl: Merged 1 segments, 134 bytes to disk to satisfy reduce memory limit
2024-10-21 18:59:15,551 INFO reduce.MergeManagerImpl: Merging 1 files, 138 bytes from disk
2024-10-21 18:59:15,551 INFO reduce.MergeManagerImpl: Merging 0 segments, 0 bytes from memory into reduce

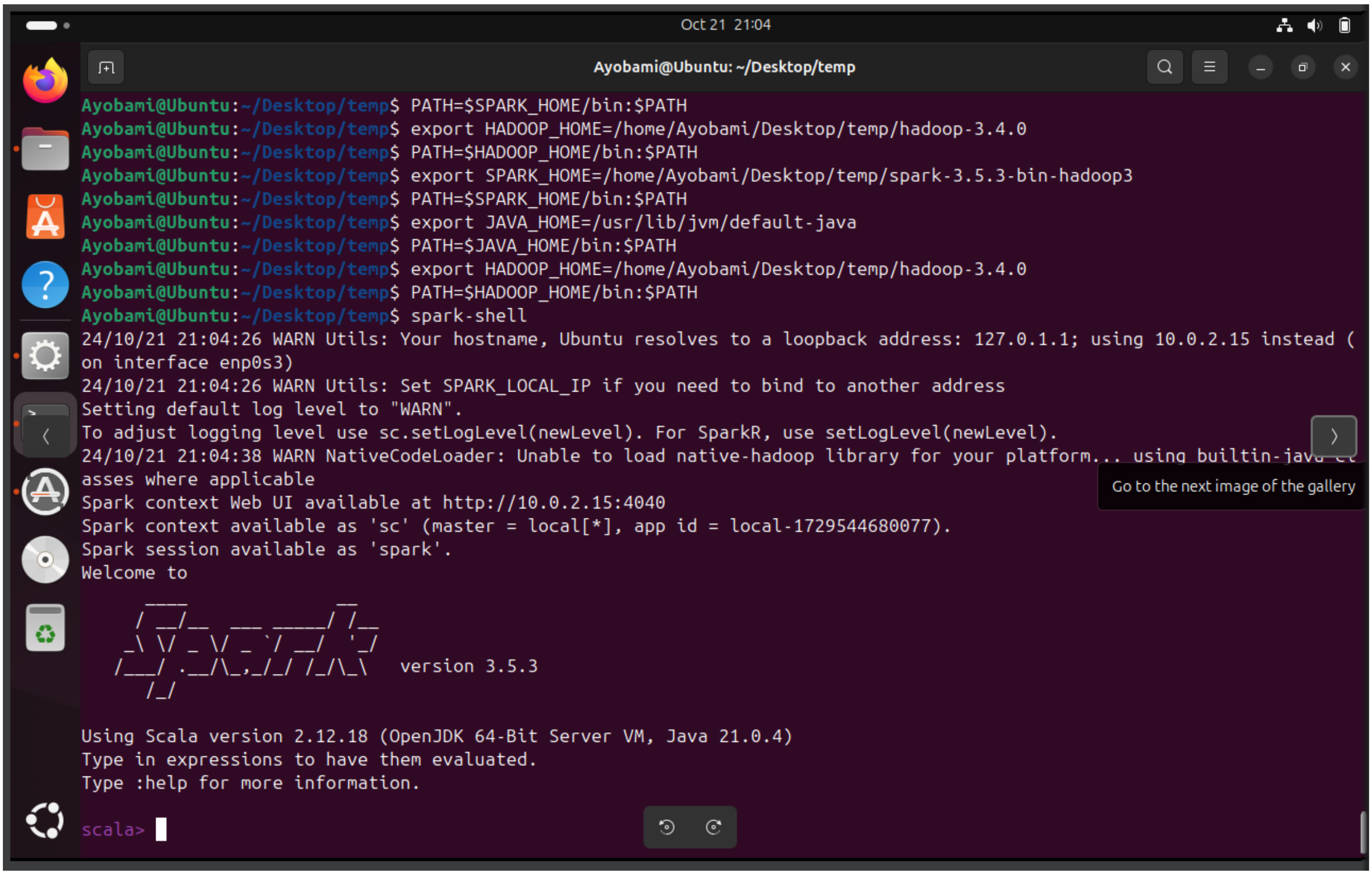

Oct 21 19:00
Ayobami@Ubuntu: ~/Desktop/temp
Map-Reduce Framework
Combine input records=0
Combine output records=0
Reduce input groups=9
Reduce shuffle bytes=138
Reduce input records=9
Reduce output records=9
Spilled Records=9
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=0
Total committed heap usage (bytes)=151531520
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Output Format Counters
Bytes Written=108
2024-10-21 18:59:15,607 INFO mapred.LocalJobRunner: Finishing task: attempt_local774138703_0001_r_000000_0
2024-10-21 18:59:15,608 INFO mapred.LocalJobRunner: reduce task executor complete.
2024-10-21 18:59:15,727 INFO mapreduce.Job: Job job_local774138703_0001 running in uber mode : false
2024-10-21 18:59:15,736 INFO mapreduce.Job: map 100% reduce 100%
2024-10-21 18:59:15,738 INFO mapreduce.Job: Job job_local774138703_0001 completed successfully
2024-10-21 18:59:15,767 INFO mapreduce.Job: Counters: 30
File System Counters
FILE: Number of bytes read=564064
FILE: Number of bytes written=1995204
FILE: Number of read operations=0


Oct 21 19:00
Ayobami@Ubuntu: ~/Desktop/temp
FILE: Number of large read operations=0
FILE: Number of write operations=0
Map-Reduce Framework
Map input records=5
Map output records=10
Map output bytes=126
Map output materialized bytes=138
Input split bytes=113
Combine input records=10
Combine output records=9
Reduce input groups=9
Reduce shuffle bytes=138
Reduce input records=9
Reduce output records=9
Spilled Records=18
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=24
Total committed heap usage (bytes)=303063040
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=102
File Output Format Counters
Bytes Written=108
Ayobami@Ubuntu:~/Desktop/temp$


Oct 21 19:02
Ayobami@Ubuntu: ~/Desktop/temp
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=24
Total committed heap usage (bytes)=303063040
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=102
File Output Format Counters
Bytes Written=108
Ayobami@Ubuntu:~/Desktop/temp$ hdfs dfs -ls output
Found 2 items
-rw-r--r-- 1 Ayobami Ayobami 0 2024-10-21 18:59 output/_SUCCESS
-rw-r--r-- 1 Ayobami Ayobami 96 2024-10-21 18:59 output/part-r-00000
Ayobami@Ubuntu:~/Desktop/temp$ hdfs dfs -getmerge output /tmp/result
Ayobami@Ubuntu:~/Desktop/temp$ cat /tmp/result
Berlin, 1
Copenhagen, 1
Denmark 1
England 2
France 1
Germany 1
London, 1
Manchester, 1
Paris, 1
Ayobami@Ubuntu:~/Desktop/temp$

APPENDIX B

SPARK



APPENDIX C

CODE SYNTAX

```
# **ANALYSIS OF MACHINE LEARNING MODELS FOR AIRLINE PASSENGER'S SATISFACTION USING PYSPARK**
---
---
We wanted to understand what factors make airline passengers more or less satisfied with their flight
experience. By studying this, airlines could improve services and prioritize what passengers care most about.
To do this, we employed four machine learning models (logistic regression, Support Vector Machine,
Random Forest and Gradient Boosted Tree) to help predict whether a passenger is satisfied or not based
on other related information such as the comfort of the seats, inflight Wi-Fi service, distance
traveled, etc. We were also able to determine which factors are most important for passenger satisfaction.
#Importing Data Exploratory libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#Loading the dataset
data = pd.read_csv('/content/sample_data/airline_passenger_satisfaction.csv')
data.head()
#Checking the number of rows and columns
data.shape
#Counts of the label
data['Satisfaction'].value_counts()
#Checking information about the dataset
data.info()
```

```
#Checking the columns with null values

data.isnull().sum()

#Drop the null values since they are less than 10% of the data

data.dropna(inplace=True)
data.shape

data['Satisfaction'].value_counts()

#Description statistics of the dataset

data.describe()

#Checking the number of unique values each column has

data.nunique()

#Dropping outlier data

arr_delay_outlier = data['Arrival Delay'] < 650
dep_delay_outlier = data['Departure Delay'] < 650
flight_dist_outlier = data['Flight Distance'] < 4000
Data = data[arr_delay_outlier & dep_delay_outlier & flight_dist_outlier]
Data.shape

**PYSPARK ENVIRONMENT**

#Install pyspark environment
!pip install pyspark

#Importing the required spark libraries
from pyspark.sql import SparkSession
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.classification import LogisticRegression, LinearSVC, RandomForestClassifier, GBTClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

#Initialising the Spark Session
spark = SparkSession.builder.appName('AirlineSatisfaction_RF').getOrCreate()

#Load the RF dataset
```

```
    New_Data = spark.createDataFrame(Data, verifySchema=True)
    New_Data.printSchema()

    #Indexing categorical columns

    #Satisfaction column
    satisfaction_indexer = StringIndexer(inputCol='Satisfaction', outputCol='label')
    New_Data = satisfaction_indexer.fit(New_Data).transform(New_Data)

    #Gender column
    gender_indexer = StringIndexer(inputCol='Gender', outputCol='Gender_Index')
    New_Data = gender_indexer.fit(New_Data).transform(New_Data)

    #Customer Type column
    customer_type_indexer = StringIndexer(inputCol='Customer Type', outputCol='Customer_Type_Index')
    New_Data = customer_type_indexer.fit(New_Data).transform(New_Data)

    #Type of Travel column
    type_of_travel_indexer = StringIndexer(inputCol='Type of Travel', outputCol='Type_of_Travel_Index')
    New_Data = type_of_travel_indexer.fit(New_Data).transform(New_Data)
    #Class column
    class_indexer = StringIndexer(inputCol='Class', outputCol='Class_Index')
    New_Data = class_indexer.fit(New_Data).transform(New_Data)

    #Departure and Arrival Time Convenience column
    Arrival_Depart_Conv_indexer = StringIndexer(inputCol='Departure and Arrival Time Convenience', outputCol='Arr_Dept_Time_Cov_Index')
    New_Data = Arrival_Depart_Conv_indexer.fit(New_Data).transform(New_Data)

    #Ease of Online Booking column
    Ease_of_Online_Booking_indexer = StringIndexer(inputCol='Ease of Online Booking', outputCol='Ease_of_Online_Booking_Index')
    New_Data = Ease_of_Online_Booking_indexer.fit(New_Data).transform(New_Data)

    #Check-in Service column
    Check_in_Service_indexer = StringIndexer(inputCol='Check-in Service', outputCol='Check_in_Service_Index')
```

```
    New_Data = Check_in_Service_indexer.fit(New_Data).transform(New_Data)

    #Online Boarding column
    Online_Boarding_indexer = StringIndexer(inputCol='Online Boarding', outputCol='Online_Boarding_Index')
    New_Data = Online_Boarding_indexer.fit(New_Data).transform(New_Data)

    #Gate Location column
    Gate_Location_indexer = StringIndexer(inputCol='Gate Location', outputCol='Gate_Location_Index')
    New_Data = Gate_Location_indexer.fit(New_Data).transform(New_Data)

    #On-board Service column
    Onboard_Service_indexer = StringIndexer(inputCol='On-board Service', outputCol='Onboard_Service_Index')
    New_Data = Onboard_Service_indexer.fit(New_Data).transform(New_Data)

    #Seat Comfort column
    Seat_Comfort_indexer = StringIndexer(inputCol='Seat Comfort', outputCol='Seat_Comfort_Index')
    New_Data = Seat_Comfort_indexer.fit(New_Data).transform(New_Data)

    #Leg Rom Service column
    Leg_Room_Service_indexer = StringIndexer(inputCol='Leg Room Service', outputCol='Leg_Room_Service_Index')
    New_Data = Leg_Room_Service_indexer.fit(New_Data).transform(New_Data)

    #Cleanliness column
    Cleanliness_indexer = StringIndexer(inputCol='Cleanliness', outputCol='Cleanliness_Index')
    New_Data = Cleanliness_indexer.fit(New_Data).transform(New_Data)

    #Food and Drink column
    Food_and_Drink_indexer = StringIndexer(inputCol='Food and Drink', outputCol='Food_and_Drink_Index')
    New_Data = Food_and_Drink_indexer.fit(New_Data).transform(New_Data)

    #In-flight Service column
    Inflight_Service_indexer = StringIndexer(inputCol='In-flight Service', outputCol='Inflight_Service_Index')
```

```
    New_Data = Inflight_Service_indexer.fit(New_Data).transform(New_Data)

    #In-flight Entertainment column
    Inflight_Entertaiment_indexer = StringIndexer(inputCol='In-flight Entertainment', outputCol='Inflight_Entertaiment_Index')
    New_Data = Inflight_Entertaiment_indexer.fit(New_Data).transform(New_Data)

    #In-flight Wifi Service column
    On_board_Wifi_Service_indexer = StringIndexer(inputCol='In-flight Wifi Service', outputCol='On_board_Wifi_Service_Index')
    New_Data = On_board_Wifi_Service_indexer.fit(New_Data).transform(New_Data)

    #Baggage Handling column
    Baggage_Handling_indexer = StringIndexer(inputCol='Baggage Handling', outputCol='Baggage_Handling_Index')
    New_Data = Baggage_Handling_indexer.fit(New_Data).transform(New_Data)

    #A peep into the new columns

    New_Data.columns

    #Extracting the indexed categorical columns with the numerical columns as relevant features

    feature_columns = ['Age', 'Flight Distance', 'Departure Delay', 'Arrival Delay', 'Gender_Index',
                       'Customer_Type_Index', 'Type_of_Travel_Index','Class_Index', 'Arr_Dept_Time_Cov_Index',
                       'Ease_of_Online_Booking_Index','Check_in_Service_Index', 'Online_Boarding_Index',
                       'Gate_Location_Index', 'Onboard_Service_Index', 'Seat_Comfort_Index', 'Leg_Room_Service_Index',
                       'Cleanliness_Index', 'Food_and_Drink_Index', 'Inflight_Service_Index',
                       'Inflight_Entertaiment_Index', 'On_board_Wifi_Service_Index', 'Baggage_Handling_Index']

    #A peep into the first 5 rows of the feature columns

    New_Data.select(feature_columns).show(5)

    #Transforming the feature columns using Vector Assembler

```

```
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
output = assembler.transform(New_Data)
output.select("features").show(5)

#Extracting the transformed features column and label column for the classification

final_data = output.select("features", "label")
final_data.printSchema()

#Defining Evaluation Metrics for the Models

accuracy_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
precision_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")
recall_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName= "weightedRecall")
f1_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")


**LOGISTIC REGRESSION MODEL**

#Copying transformed dataset for the for the Logistic Regression

lr_data = final_data
lr_data.printSchema()

#Splitting the lr_data into training and test sets

train_data_lr, test_data_lr = lr_data.randomSplit([0.7, 0.3], seed=20)

#Initialising the Logistic Regression classifeir

lr = LogisticRegression(featuresCol="features", labelCol="label")

#Trainning the lr model

lr_model = lr.fit(train_data_lr)

```

```
#Predictions with the lr model on the test set
lr_predictions = lr_model.transform(test_data_lr)
#Evaluating the lr model
lr_accuracy = accuracy_evaluator.evaluate(lr_predictions)
lr_weightedPrecision = precision_evaluator.evaluate(lr_predictions)
lr_weightedRecall = recall_evaluator.evaluate(lr_predictions)
lr_f1 = f1_evaluator.evaluate(lr_predictions)
print(f"Logistic Regression Model Accuracy: {lr_accuracy: .2f}")
print(f"Logistic Regression Model Weighted Precision: {lr_weightedPrecision: .2f}")
print(f"Logistic Regression Model Weighted Recall: {lr_weightedRecall: .2f}")
print(f"Logistic Regression Model f1: {lr_f1: .2f}")
#Preview first three rows of the lr prediction
lr_predicted= lr_predictions.select("label", "prediction", "features")
lr_predicted.show(3)
#Extract the logistic coefficients
lr_coefficients = lr_model.coefficients.toArray()
#Matching features to their absolute coefficients
lr_feature_importance = list(zip(feature_columns, abs(lr_coefficients)))
#Sort the feature importances in descending order
sorted_lr_importances = sorted(lr_feature_importance, key=lambda x: x[1], reverse=True)
lr_features_ranking = pd.DataFrame(sorted_lr_importances, columns=['Feature', 'LR_coefficient'])
lr_features_ranking
***Applying Hyperparameter tunning to the Logistic Regression Classifer***
```

```
    #Parameter grid for the logistic regression Hyperparameter tunning

    lr_paramGrid = (ParamGridBuilder()
                    .addGrid(lr.regParam, [0.1, 0.01])
                    .addGrid(lr.maxIter, [50, 100])
                    .build())

    # Defining the Cross-validator for the logistic regression

    lr_crossval = CrossValidator(estimator=lr,
                                 estimatorParamMaps=lr_paramGrid,
                                 evaluator=accuracy_evaluator, numFolds=5)

    #Cross-validation fitting for the lr model

    lr_cv_model = lr_crossval.fit(train_data_lr)

    # Extracting lr_cv Best model

    lr_best_model = lr_cv_model.bestModel

    #Predictions with the SVM CV on the test data

    lr_cv_predictions = lr_best_model.transform(test_data_lr)

    #Evaluating the lr_cv model

    lr_cv_accuracy = accuracy_evaluator.evaluate(lr_cv_predictions)
    lr_cv_weightedPrecision = precision_evaluator.evaluate(lr_cv_predictions)
    lr_cv_weightedRecall = recall_evaluator.evaluate(lr_cv_predictions)
    lr_cv_f1 = f1_evaluator.evaluate(lr_cv_predictions)

    print(f"CV Logistic Regression Model Accuracy: {lr_cv_accuracy: .2f}")
    print(f"CV Logistic Regression Model Weighted Precision: {lr_cv_weightedPrecision: .2f}")
    print(f"CV Logistic Regression Model Weighted Recall: {lr_cv_weightedRecall: .2f}")
    print(f"CV Logistic Regression Model f1: {lr_cv_f1: .2f}")

    #Preview first three rows of the lr_Cv prediction

    cv_lr_predicted= lr_predictions.select("label", "prediction", "features")
```

```
cv_lr_predicted.show(3)

#Extract the logistic coefficients

lr_cv_coefficients = lr_best_model.coefficients.toArray()

#Matching features to their absolute coefficients

lr_cv_feature_importance = list(zip(feature_columns, abs(lr_cv_coefficients)))

#Sort the feature importances in descending order

sorted_lr_cv_importances = sorted(lr_cv_feature_importance, key=lambda x: x[1], reverse=True)

cv_lr_feature_ranking = pd.DataFrame(sorted_lr_cv_importances, columns=['Feature', 'CV_LR_coefficient'])
cv_lr_feature_ranking

**LINEAR SUPPORT VECTOR MACHINE (SVM) MODEL**

#Copying transformed dataset for the SVM model

svm_data = final_data
svm_data.printSchema()

#Splitting the svm_data into training and test sets

train_data_svm, test_data_svm = svm_data.randomSplit([0.7, 0.3], seed=20)

#Initialising the Linear SVM classifeir

svm = LinearSVC(featuresCol="features", labelCol="label", maxIter=100, regParam=0.01)

#Trainning the SVM model

svm_model = svm.fit(train_data_svm)

#Predictions with the SVM model on the test set

svm_predictions = svm_model.transform(test_data_svm)

#Evaluating the SVM model

svm_accuracy = accuracy_evaluator.evaluate(svm_predictions)
```

```
svm_weightedPrecision = precision_evaluator.evaluate(svm_predictions)
svm_weightedRecall = recall_evaluator.evaluate(svm_predictions)
svm_f1 = f1_evaluator.evaluate(svm_predictions)

print(f"SVM Model Accuracy: {svm_accuracy: .2f}")
print(f"SVM Model Weighted Precision: {svm_weightedPrecision: .2f}")
print(f"SVM Model Weighted Recall: {svm_weightedRecall: .2f}")
print(f"SVM Model f1: {svm_f1: .2f}")

#Preview first three rows of the svm prediction

svm_predicted= svm_predictions.select("label", "prediction", "features")
svm_predicted.show(3)

#Extract the SVM coefficients

svm_coefficients = svm_model.coefficients.toArray()

#Matching features to their absolute coefficients

svm_feature_importance = list(zip(feature_columns, abs(svm_coefficients)))

#Sort the feature importances in descending order

sorted_svm_importances = sorted(svm_feature_importance, key=lambda x: x[1], reverse=True)

svm_feature_ranking = pd.DataFrame(sorted_svm_importances, columns=['Feature', 'SVM_coefficient'])
svm_feature_ranking

***Applying hyperparameter tunning to the Linear SVM Classifer***

#Parameter grid for the SVM Hyperparameter tunning

svm_param_grid = (ParamGridBuilder()
                  .addGrid(svm.maxIter, [50, 100, 200])
                  .addGrid(svm.regParam, [0.01, 0.1, 1.0])
                  .build())

# Defining the Cross-validator for the SVM
svm_crossval = CrossValidator(estimator=svm,
                              estimatorParamMaps=svm_param_grid,
```

```
                              evaluator=accuracy_evaluator,
                              numFolds=3)

    #Cross-validation fitting for the svm model

    svm_cv_model = svm_crossval.fit(train_data_svm)

    # Extracting SVM Best model

    svm_best_model = svm_cv_model.bestModel

    #Predictions with the SVM CV on the test data

    svm_cv_predictions = svm_best_model.transform(test_data_svm)

    #Evaluating the cv SVM model

    svm_cv_accuracy = accuracy_evaluator.evaluate(svm_cv_predictions)
    svm_cv_weightedPrecision = precision_evaluator.evaluate(svm_cv_predictions)
    svm_cv_weightedRecall = recall_evaluator.evaluate(svm_cv_predictions)
    svm_cv_f1 = f1_evaluator.evaluate(svm_cv_predictions)

    print(f"CV SVM Model Accuracy: {svm_cv_accuracy: .2f}")
    print(f"CV SVM Model Weighted Precision: {svm_cv_weightedPrecision: .2f}")
    print(f"CV SVM Model Weighted Recall: {svm_cv_weightedRecall: .2f}")
    print(f"CV SVM Model f1: {svm_cv_f1: .2f}")

    #Preview first three rows of the SVM_cv prediction

    cv_svm_predicted= svm_predictions.select("label", "prediction", "features")
    cv_svm_predicted.show(3)

    #Extract the CV_SVM coefficients

    svm_cv_coefficients = svm_best_model.coefficients.toArray()

    #Matching features to their absolute coefficients

    svm_cv_feature_importance = list(zip(feature_columns, abs(svm_cv_coefficients)))

    #Sort the feature importances in descending order

```

```
sorted_svm_cv_importances = sorted(svm_cv_feature_importance, key=lambda x: x[1], reverse=True)

cv_svm_feature_ranking = pd.DataFrame(sorted_svm_cv_importances, columns=['Feature', 'CV_SVM_coefficient'])
cv_svm_feature_ranking

**RANDOM FOREST MODEL**

#Copying transformed dataset for the Logistic Regression

rf_data = final_data
rf_data.printSchema()

#Split the data into training and test sets

train_data_rf, test_data_rf = rf_data.randomSplit([0.7, 0.3], seed=20)

#Initialising Random Forest Model

rf = RandomForestClassifier(featuresCol='features', labelCol='label', numTrees=100)

#Training the Model

rf_model = rf.fit(train_data_rf)

#The prediction on the test data

rf_predictions = rf_model.transform(test_data_rf)

#Evaluating the random forest model

rf_accuracy = accuracy_evaluator.evaluate(rf_predictions)
rf_weightedPrecision = precision_evaluator.evaluate(rf_predictions)
rf_weightedRecall = recall_evaluator.evaluate(rf_predictions)
rf_f1 = f1_evaluator.evaluate(rf_predictions)

print(f"Random Forest Model Accuracy: {rf_accuracy: .2f}")
print(f"Random Forest Model Weighted Precision: {rf_weightedPrecision: .2f}")
print(f"Random Forest Model Weighted Recall: {rf_weightedRecall: .2f}")
print(f"Random Forest Model f1: {rf_f1: .2f}")

#Preview first three rows of the rf prediction
```

```

rf_predicted= rf_predictions.select("label", "prediction", "features")
rf_predicted.show(3)

#Extract the feature importances

rf_importances = rf_model.featureImportances

#Matching features to their importances

rf_feature_importances = list(zip(feature_columns, rf_importances))

#Sort the feature importances in descending order

sorted_rf_importances = sorted(rf_feature_importances, key=lambda x: x[1], reverse=True)

rf_feature_ranking = pd.DataFrame(sorted_rf_importances, columns=['Feature', 'RF_Importance'])
rf_feature_ranking

***Applying Hyperparameter tunning to the Random Forest***

#Defining the

rf_param_grid = (ParamGridBuilder()
                 .addGrid(rf.numTrees, [50, 100, 150])
                 .addGrid(rf.maxDepth, [5, 10, 15])
                 .addGrid(rf.maxBins, [32, 64])
                 .build())

#The cross-validator

rf_crossval = CrossValidator(estimator=rf,
                             estimatorParamMaps=rf_param_grid,
                             evaluator=accuracy_evaluator,
                             numFolds=3)

#Training the model

rf_cv_model = rf_crossval.fit(train_data_rf)

#The best model

best_rf_model = rf_cv_model.bestModel

#Predictions on the test data
```

```
rf_cv_predictions = best_rf_model.transform(test_data_rf)

#Evaluating the cv random froest model

rf_cv_accuracy = accuracy_evaluator.evaluate(rf_cv_predictions)
rf_cv_weightedPrecision = precision_evaluator.evaluate(rf_cv_predictions)
rf_cv_weightedRecall = recall_evaluator.evaluate(rf_cv_predictions)
rf_cv_f1 = f1_evaluator.evaluate(rf_cv_predictions)

print(f"CV RF Model Accuracy: {rf_cv_accuracy: .2f}")
print(f"CV RF Model Weighted Precision: {rf_cv_weightedPrecision: .2f}")
print(f"CV RF Model Weighted Recall: {rf_cv_weightedRecall: .2f}")
print(f"CV RF Model f1: {rf_cv_f1: .2f}")

#Preview first three rows of the svm prediction

cv_rf_predicted= rf_cv_predictions.select("label", "prediction", "features")
cv_rf_predicted.show(3)

#Extract feature importances
cv_rf_feature_importances = best_rf_model.featureImportances

#Matching features to their importances
cv_rf_importances = list(zip(feature_columns, cv_rf_feature_importances))

#Sort the feature importances in descending order
sorted_cv_rf_importances = sorted(cv_rf_importances, key=lambda x: x[1], reverse=True)

cv_rf_features_ranking = pd.DataFrame(sorted_cv_rf_importances, columns=['Feature', 'CV_RF_Importance'])
cv_rf_features_ranking

**GRADIENT-BOOSTED TREE (GBT) CLASSIFIER MODEL**

#Copying transformed dataset for the GBT model

gbt_data = final_data
gbt_data.printSchema()

```

```
    #Split the data into training and test sets

    train_data_gbt, test_data_gbt = gbt_data.randomSplit([0.7, 0.3], seed=20)

    # Initialize Gradient-Boosted Trees classifier

    gbt = GBTClassifier(featuresCol="features", labelCol="label", maxIter=50)

    # Trainning the GBT model

    gbt_model = gbt.fit(train_data_gbt)

    #Prediction on the test set

    gbt_predictions = gbt_model.transform(test_data_gbt)

    #Evaluating the GBT model

    gbt_accuracy = accuracy_evaluator.evaluate(gbt_predictions)
    gbt_weightedPrecision = precision_evaluator.evaluate(gbt_predictions)
    gbt_weightedRecall = recall_evaluator.evaluate(gbt_predictions)
    gbt_f1 = f1_evaluator.evaluate(gbt_predictions)

    print(f"Gradient Boost Tree Model Accuracy: {gbt_accuracy: .2f}")
    print(f"Gradient Boost Tree Model Weighted Precision: {gbt_weightedPrecision: .2f}")
    print(f"Gradient Boost Tree Model Weighted Recall: {gbt_weightedRecall: .2f}")
    print(f"Gradient Boost Tree Model f1: {gbt_f1: .2f}")

    #Preview first three rows of the GBT prediction

    gbt_predicted= gbt_predictions.select("label", "prediction", "features")
    gbt_predicted.show(3)

    # Extracting feature importances from GBT model
    gbt_importances = gbt_model.featureImportances.toArray()

    # Combine with feature names
    gbt_feature_importance = list(zip(feature_columns, gbt_importances))

    #Sorting by importances
```

```
gbt_sorted_importance = sorted(gbt_feature_importance, key=lambda x: x[1], reverse=True)

gbt_feature_ranking = pd.DataFrame(gbt_sorted_importance, columns=['Feature', 'GBT_Importance'])
gbt_feature_ranking

*Hyperparameter tunning for the Gradient Boosted Tree Classifer*

# Parameter grid for the GBT hyperparameter tuning

gbt_param_grid = (ParamGridBuilder()
                  .addGrid(gbt.maxIter, [10, 50, 100])
                  .addGrid(gbt.maxDepth, [3, 5, 7])
                  .addGrid(gbt.stepSize, [0.1, 0.2])
                  .build())

# Defining the Cross-validator for the GBT

gbt_crossval = CrossValidator(estimator=gbt,
                              estimatorParamMaps=gbt_param_grid,
                              evaluator=accuracy_evaluator,
                              numFolds=3)

#Performing the GBT cross-validation

gbt_cv_model = gbt_crossval.fit(train_data_gbt)

# Extracting GBT Best model

gbt_best_model = gbt_cv_model.bestModel

#Predictions on the test data

gbt_cv_predictions = gbt_best_model.transform(test_data_gbt)

#Evaluating the cv GBT model

gbt_cv_accuracy = accuracy_evaluator.evaluate(gbt_cv_predictions)
gbt_cv_weightedPrecision = precision_evaluator.evaluate(gbt_cv_predictions)
gbt_cv_weightedRecall = recall_evaluator.evaluate(gbt_cv_predictions)
gbt_cv_f1 = f1_evaluator.evaluate(gbt_cv_predictions)

print(f"CV GBT Model Accuracy: {gbt_cv_accuracy: .2f}")
```

```
    print(f"CV GBT Model Weighted Precision: {gbt_cv_weightedPrecision: .2f}")
    print(f"CV GBT Model Weighted Recall: {gbt_cv_weightedRecall: .2f}")
    print(f"CV GBT Model f1: {gbt_cv_f1: .2f}")

    #Preview first three rows of the GBT prediction

    cv_gbt_predicted= gbt_cv_predictions.select("label", "prediction", "features")
    cv_gbt_predicted.show(3)

    # Extracting feature importances from tuned GBT model
    cv_gbt_importances = gbt_best_model.featureImportances.toArray()

    # Combine with feature names
    cv_gbt_feature_importance = list(zip(feature_columns, cv_gbt_importances))

    #Sorting by importances
    cv_gbt_sorted_importance = sorted(cv_gbt_feature_importance, key=lambda x: x[1], reverse=True)

    cv_gbt_feature_ranking = pd.DataFrame(cv_gbt_sorted_importance, columns=['Feature', 'GBT_CV_Importance'])
    cv_gbt_feature_ranking

    features_rank = pd.concat([lr_features_ranking, cv_lr_feature_ranking, svm_feature_ranking, cv_svm_feature_ranking, rf_feature_ranking, cv_rf_features_ranking, gbt_feature_ranking, cv_gbt_feature_ranking], axis=1)
    features_rank

    Satisfaction = Data['Satisfaction']
    logistic_prediction = pd.DataFrame(lr_predicted.select("label", "prediction").collect(), columns=["Label", "lr_prediction"])
    tuned_logistic_prediction = pd.DataFrame(lr_cv_predictions.select("prediction").collect(), columns=["lr_cv_prediction"])
    svm_prediction = pd.DataFrame(svm_predicted.select("prediction").collect(), columns=["svm_prediction"])
    tuned_svm_prediction = pd.DataFrame(svm_cv_predictions.select("prediction").collect(), columns=["svm_cv_prediction"])
```

```
random_forest_prediction = pd.DataFrame(rf_predicted.select("prediction").collect(), columns=["rf_prediction"])
tuned_random_forest_prediction = pd.DataFrame(rf_cv_predictions.select("prediction").collect(), columns=["rf_cv_prediction"])
gbt_prediction = pd.DataFrame(gbt_predicted.select("prediction").collect(), columns=["gbt_prediction"])
tuned_gbt_prediction = pd.DataFrame(gbt_cv_predictions.select("prediction").collect(), columns=["gbt_cv_prediction"])

all_predictions = pd.concat([Satisfaction, logistic_prediction, tuned_logistic_prediction, svm_prediction, tuned_svm_prediction, random_forest_prediction, tuned_random_forest_prediction, gbt_prediction, tuned_gbt_prediction], axis=1)
all_predictions.head(50)
all_predictions.to_csv('predictions2.csv', index=False)

features_rank.to_csv('features_rank.csv', index=False)
all_predictions.to_csv('predictions.csv', index=False)
```