April 2024

Throughput-Focused Vertical Handover Prediction based on User Mobility

Supervisor: Mah-Rukh Fida

Technical Computing

Dissertation Research Project Proposal 2023/24

Matt Newman (S4100701)

Abstract

Existing works focusing on handover prediction in static and mobile scenarios, as well as vertical handover prediction, commonly utilise signal strength variables as predictors and use simulated data over real-world data. They also use methods such as neural networks. Throughput is neglected despite indications that it is an important indicator of handover success, and real-world data is not frequently used to test methods despite simulated data making inaccurate assumptions. As such, this dissertation aimed to plug this gap by proposing throughput-focused vertical handover prediction models based on user mobility that use AI methods. Three AI methods, the existing commonly used neural network and two unused methods in the Naïve-Bayes Classifier and Random Forest, were tested. The most effective method was then used to test the efficacy of throughput as a predictor compared to existing commonly used signal strength variables. The results found that the Random Forest was the most effective AI method, with 97-100% accuracy and high complexity reduction potential. As Random Forests are unused in existing literature, this suggests that they have considerable untapped potential in handover prediction. The results also found that throughput was an effective predictor of handover, but that it did not notably improve on signal strength variables. Nonetheless, it was still found to be accurate, so throughput or signal strength variables could be used for an accurate model.

**Keywords**: Handover, Radio Access Technology, Prediction, Artificial Intelligence, Machine Learning, Throughput, Signal Strength, Random Forest, Neural Network, Naïve-Bayes Classifier

Ethical Issues

1. I confirm that I have submitted my ethical form based on the timeline provided in the module and received approval from my supervisor/university.
2. I declare that this dissertation has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree or anywhere else. Except where states otherwise by citation and reference or acknowledgment, the work presented is entirely my own.
3. I confirm that all the Tables and Figures in this dissertation are my own works or a regeneration from other people’s work with citation.
4. I confirm that all the tools, software and datasets have been used in this dissertation followed the terms and conditions in their license agreement and university REC code of conduct.
5. I confirm that all the citations in this dissertation have been listed in the bibliography and they are all accessible. In case that university wants to cross examine the citations, I will provide the content for the references which are not accessible.
6. I confirm in this dissertation no human/animal study has been conducted.

Source Code and Datasets

All source code and datasets used are downloadable from the following Google Drive link: <https://drive.google.com/drive/folders/1FWkKqVKboUsAfEJw5c1_zfDwbfADIKST?usp=sharing>

Source code can also be found in the appendices. Source code for the initial dataset can be found in Appendix 3, while source code for the validation dataset can be found in Appendix 4 and source code for result figures can be found in Appendix 5.

For the initial dataset, the dataset prior to any pre-processing is named “Throughput Tests – Speedtest – Active Measurements.csv”, while the final subsets used by the predictive models are named “Final [Subset Name] Dataset.csv”.

For the validation dataset, the dataset prior to any pre-processing is split across numerous CSV files contained within the directories “bus”, “car”, “pedestrian”, “static” and “train”, while the final subsets used by the predictive models are named “Filtered Combined Dataset ([Subset Name]).csv”.

Acknowledgements

I would firstly like to thank my dissertation supervisor, Dr Mah-Rukh Fida, for her advice and support throughout this project. Her help has been truly invaluable. I would also like to thank my family for their continued support throughout this project and my wider degree, and for helping me through some of my more stressful moments.

Contents

[1 Introduction 9](#_Toc163114525)

[1.1 Overview 9](#_Toc163114526)

[1.2 Problem Statement 10](#_Toc163114527)

[1.3 Research Questions 10](#_Toc163114528)

[1.4 Research Objectives 10](#_Toc163114529)

[1.5 Scope 11](#_Toc163114530)

[1.6 Conclusion 11](#_Toc163114531)

[2 Literature Review 12](#_Toc163114532)

[2.1 Static Handover Prediction 12](#_Toc163114533)

[2.2 Vertical Handover Prediction 16](#_Toc163114534)

[2.3 Mobility-Based Handover Prediction 20](#_Toc163114535)

[2.4 Literature Gap in Existing Studies 24](#_Toc163114536)

[3 Methodology 25](#_Toc163114537)

[3.1 Experimental Method 25](#_Toc163114538)

[3.2 Initial Dataset 25](#_Toc163114539)

[3.2.1 Dataset Attributes 26](#_Toc163114540)

[3.2.2 Pre-Processing 29](#_Toc163114541)

[3.3 Validation Dataset 29](#_Toc163114542)

[3.4 AI Algorithms 30](#_Toc163114543)

[4 Results and Evaluation 32](#_Toc163114544)

[4.1 Classifier Comparison 32](#_Toc163114545)

[4.2 Comparison of Throughput to Status Quo Variables 33](#_Toc163114546)

[4.3 Complexity Improvements 34](#_Toc163114547)

[4.4 Evaluation 37](#_Toc163114548)

[4.5 Limitations and Further Work 38](#_Toc163114549)

[5 References 39](#_Toc163114550)

[6 Appendices 46](#_Toc163114551)

[Appendix 1: Validation Dataset Pre-Processing Steps 46](#_Toc163114552)

[Appendix 2: Ethical Form 47](#_Toc163114553)

[Appendix 3: Initial Dataset Source Code 50](#_Toc163114554)

[Appendix 3.1: Pre-Processing 50](#_Toc163114555)

[Appendix 3.2: Dataset Visualisation 89](#_Toc163114556)

[Appendix 3.3: AI Models 95](#_Toc163114557)

[Appendix 3.4: Model GUIs 155](#_Toc163114558)

[Appendix 4: Validation Dataset Source Code 192](#_Toc163114559)

[Appendix 4.1: Pre-Processing 192](#_Toc163114560)

[Appendix 4.2: Dataset Visualisation 225](#_Toc163114561)

[Appendix 4.3: AI Models 231](#_Toc163114562)

[Appendix 4.4: Model GUIs 291](#_Toc163114563)

[Appendix 5: Result Figures Source Code 320](#_Toc163114564)

[Appendix 5.1: ML Model Comparison Figure (Initial Dataset).py 320](#_Toc163114565)

[Appendix 5.2: ML Model Comparison Figure (Validation Dataset).py 321](#_Toc163114566)

[Appendix 5.3: Tree Number Accuracy Comparison.py 322](#_Toc163114567)

[Appendix 5.4: Maximum Depth Accuracy Comparison.py 323](#_Toc163114568)

Figures

Figure 1: A UK network coverage map for an example carrier. Purple denotes 5G, while orange/red denotes 4G and 4G+. (NPerf.com, 2023) 9

Figure 2: A graph showing throughput across a 400s travel period on 2 trains. Handover occurs at 125s in Train 1 and 175s in Train 2. (Charitos and Kalivas, 2013) 10

Figure 3: The distribution of 4G versus 5G within the throughput subset 27

Figure 4: The distribution of 4G versus 5G within the status quo subset 28

Figure 5: The distribution of 4G versus 5G within the combined subset 29

Figure 6: A graph comparing classifier performances on the initial dataset 32

Figure 7: A graph comparing algorithm performances on the validation dataset 33

Figure 8: A scatter plot comparing tree number accuracies on both datasets 35

Figure 9: A scatter plot comparing maximum depth accuracies on both datasets 36

Tables

Table 1: A comparison of works focusing on static handover prediction 12

Table 2: A comparison of works focusing on vertical handover prediction 16

Table 3: A comparison of works focusing on handover prediction in mobile scenarios 20

Table 4: The variable combinations used 25

Table 5: The common variables used in all subsets 26

Table 6: The variables exclusive to the throughput and combined subsets 26

Table 7: The variables exclusive to the status quo and combined datasets 27

Table 8: A comparison of the chosen algorithms to alternatives 30

Table 9: A table comparing classifier performances on the initial dataset 32

Table 10: A table comparing classifier performances on the validation dataset 33

Table 11: A table comparing subset performances on the initial dataset 34

Table 12: A table comparing subset performances on the validation dataset 34

Table 13: A table comparing different throughput type performances on the initial dataset 34

Table 14: A table comparing different throughput type performances on the validation dataset 34

Table 15: A table comparing tree number performances on the initial dataset 35

Table 16: A table comparing tree number performances on the validation dataset 35

Table 17: A table comparing maximum depth performances on the initial dataset 36

Table 18: A table comparing maximum depth performances on the validation dataset 36

# Introduction

## Overview

Smartphone and mobile internet usage are ever-increasing. In 2023, 94% of UK adults owned a smartphone (Laricchia, 2023) compared to just 24% in 2008 (Statista, 2023). In 2022, there were an estimated five billion mobile internet users worldwide (Ceci, 2023).

Performance demand has also increased, leading to the advent of high-speed mobile network technologies such as 5G. Network performance is impaired, however, by handover events. Kousias *et al.* (2022) found that handover impaired 5G network performance to a far more significant degree than for earlier technologies.

Handover failure is particularly common in fast-mobility scenarios due to conventional handover prediction methods making inaccurate predictions at high speed (Khan *et al.*, 2022). This is exacerbated with 5G due to the smaller cells. 5G’s novelty also results in coverage gaps, as Figure 1 shows.

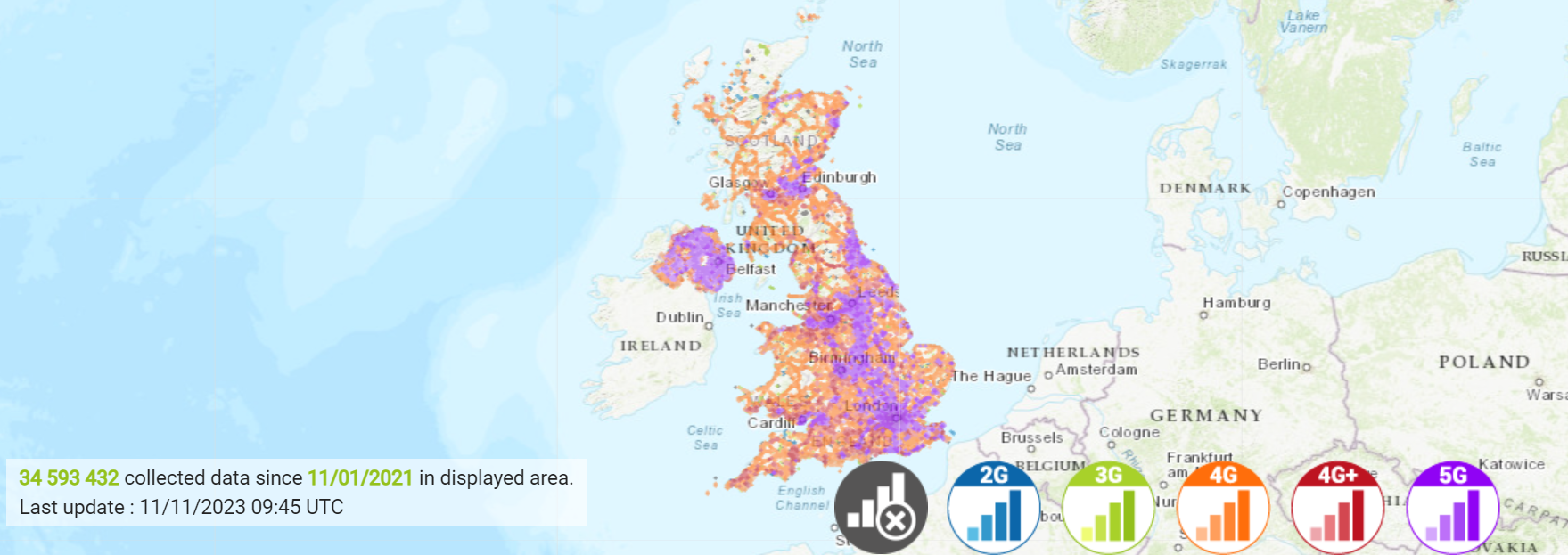


Figure 1: A UK network coverage map for an example carrier. Purple denotes 5G, while orange/red denotes 4G and 4G+. (NPerf.com, 2023)

As such, vertical handover between technologies is more desirable for a high quality of service (QoS). It occurs less frequently than horizontal handover between cells and allows access to multiple technologies (Alhabo and Zhang, 2018). However, the impact of mobility speed upon handover needs further research in a post-5G world (Hassan *et al.*, 2022).

This thesis proposes a vertical handover prediction model utilising artificial intelligence (AI) that aims to solve this issue. Similar works exist, but real-world data is not commonly used and throughput is a widely neglected predictor. Throughput can significantly affect handover success (Alhabo and Zhang, 2018). Charitos and Kalivas (2013) found that low throughput often causes handover. Figure 2 shows throughput dropping before handover for two different trains.

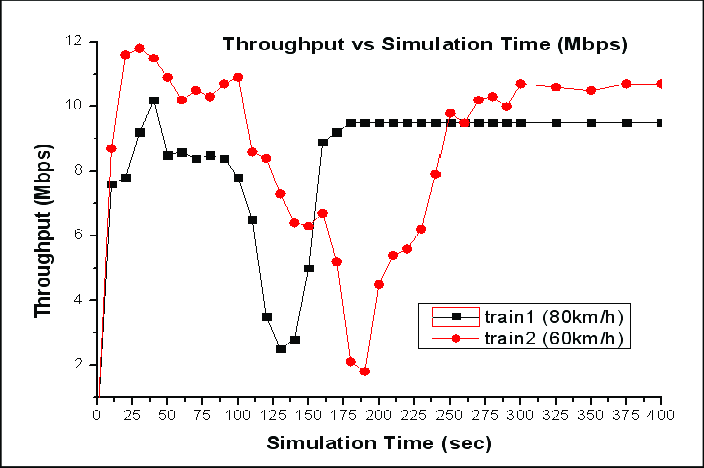


Figure 2: A graph showing throughput across a 400s travel period on 2 trains. Handover occurs at 125s in Train 1 and 175s in Train 2. (Charitos and Kalivas, 2013)

Qualcomm (2023) found that the most common smartphone activity among a group of 60,000 smartphone users was web browsing. Saverimoutou, Mathieu and Vaton (2020) found that slow loading can cause website abandonment. Therefore, having high, consistent throughput is imperative for optimal QoS.

## Problem Statement

How to accurately predict vertical handovers using a throughput-driven AI model based on user mobility?

## Research Questions

* What methods currently exist to predict handovers in static and mobile scenarios?
* What methods currently exist to predict vertical handovers?
* How accurately can AI methods predict vertical handover?
* How accurately can throughput as a parameter predict vertical handover?

## Research Objectives

* Investigate existing literature on vertical handover prediction methods and handover prediction methods in static and mobility-based scenarios.
* Develop throughput-driven vertical handover prediction models to predict the correct Radio Access Technology (RAT) to connect to using AI methods.
* Evaluate the performance of these models on real-world data.

## Scope

This study will focus on vertical handover prediction within mobile networks using AI methods. Throughput will be tested as the key predictor. More commonly used signal strength variables will be tested for comparison, and both combined will also be tested.

## Conclusion

In conclusion, increasing smartphone and mobile internet use has increased performance demand, leading to technologies such as 5G. However, handover events impact performance and prevent seamless network operation. Handover failure is particularly common in fast-mobility scenarios. As 5G’s design and novelty cause frequent horizontal handovers and coverage gaps, vertical handover is more desirable in these cases.

This study aims to solve this problem by investigating existing literature on handover prediction and producing throughput-driven AI models to predict vertical handover. Throughput is a commonly neglected predictor despite it being an important barometer of QoS and handover success. As web browsing is the most common smartphone activity and slow loading can cause website abandonment, maintaining high, consistent throughput is crucial.

# Literature Review

## Static Handover Prediction

Numerous works (Elkourdi, Mazin and Gitlin, 2019; Lee *et al.*, 2020; Kaya and Viswanathan, 2021; Nyangaresi, Rodrigues and Abeka, 2022; Vankayala *et al.*, 2022) utilised Long Short-Term Memory (LSTM) neural networks tested on simulated data to predict handover. Accuracy over 90% was often attained, but the approach needs testing upon real-world data to gauge its real-world efficacy more accurately.

Other works (Boutiba, Bagaa and Ksentini, 2021; Kaur, Goyal and Mehta, 2022; Paropkari, Thantharate and Beard, 2022) utilised neural network hybrid models, combining a regressor and a classifier to predict handover. While their accuracies were high, Support Vector Machines (SVMs) were frequently used as the classifier. SVMs are not well-suited to large datasets such as network datasets and are more prone to overfitting than alternatives (Lin *et al.*, 2011).

Prananto, Iskandar and Kurinawan (2023) utilised a hybrid of modified vector autoregression and a neural network to predict QoE and facilitate successful handover. While the approach improved on existing methods overall, it underperformed compared to them in some scenarios. This could be because not enough predictors were used, with only RSRP and RSRQ being used. Utilising other parameters could have improved accuracy.

Some works (Piran *et al.*, 2017; Cicioglu, 2021) utilised handover decision schemes using Markov chains or similar. While both schemes were successful, one flaw of Markov processes is that when many predictors are present, their solutions are approximate rather than exact (Abu Alsheikh *et al.*, 2015). Many predictors are often used in this context, possibly inhibiting accuracy.

Other works (Kaloxylos *et al.*, 2014; Alraih *et al.*, 2022) utilised fuzzy logic controllers to predict handover. While the methods were successful, with handover success rate increases and handover ping-pong reductions reaching 95%, fuzzy logic controllers are often inefficient and do not reach exact conclusions (Eker and Torun, 2006).

A more in-depth comparison of these works is displayed in Table 1.

Table 1: A comparison of works focusing on static handover prediction

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Citation** | **Dataset** | **Predictive Parameter(s)** | **Target Performance Parameter(s)** | **Machine Learning Technique(s) Used** | **Conclusions** |
| (Lee *et al.*, 2020) | Simulation in Python and MATLAB, using a Manhattan-style approach. Seven base stations, 200m apart, were simulated. | RSRP | Improved handover success rate | Deep neural network, with the input being a time series of RSRP at different base stations and the output being the best base station to conduct handover with. | The method outperformed existing methods, attaining an accuracy of 98.8%. |
| (Kaur, Goyal and Mehta, 2022) | Real-world data. Two datasets are used; one dataset logging office employee behaviours and locations, and another containing 5G data from streaming and file downloads. | Speed, RSRP, RSRQ and positional parameters | Low latency | Hybrid of LSTM and SVM. A time series of the predictive parameters was initially fed into the LSTM. The output prediction of the LSTM was used by the SVM to choose the best tower for handover. | The model outperformed existing methods while also reducing latency. |
| (Paropkari, Thantharate and Beard, 2022) | Partially real-world, partially simulation. Real data comprises 65% of the dataset and is taken from multiple third-party mobile applications. The KPIs of the current base station and 3-4 surrounding base stations were taken from these applications and compared. | Various, including RSSI, SINR and Fade Duration | Accuracy and loss | A recurrent neural network and LSTM were used alongside system-level inputs to make collective handover decisions. | The approach encapsulated detailed information of handover predictors while also being simple, flexible and accurate. |
| (Nyangaresi, Rodrigues and Abeka, 2022) | Simulation using MATLAB. The direction was randomly simulated in numerous scenarios. | Received carrier power, blocking probability, velocity, power density, path loss and traffic intensity | Packet loss and ping-pong rate | Artificial neural network that makes a handover prediction based on the predictive parameters. | The model boasted reduced packet loss and ping-pong rate alongside a 94.4% reduction in handovers. |
| (Kumar *et al.*, 2018) | Simulation using MATLAB. A simulated user dataset, containing the activities and bandwidth needs of each user, was used. | Bandwidth capacity | Handover success rate | K-means clustering and random forest classification. Clustering segregates individual users, and classification predicts their optimal tower. | The method was effective and qualified to improve handovers in 5G networks. |
| (Kaya and Viswanathan, 2021) | Simulation based on ray tracing. Two scenarios were constructed. One involved a college intersection mostly used by walkers and bikers. Another involved vehicular users on a busy road. | RSRP | Handover overhead reduction and signal-to-noise ratio enhancement | LSTMs and NLP-style methods are used to predict the optimal radio beams. | The method enhanced radio performance and reduced signaling overhead by 80%. |
| (Prananto, Iskandar and Kurniawan, 2023) | Simulation. The environment created contained three base stations, one user and a coverage hole created by a building. | RSRP and RSRQ | Improved target cell determination | Modified vector autoregression and a neural network are used to predict QoE. | The method improved on existing methods overall, but underperformed compared to them in some scenarios. |
| (Cicioglu, 2021) | Simulation using Riverbed Modeler. The environment contains 11 base stations and one user within a 5km\*5km area. | SNR | Low latency | Channel propagation models make handover predictions. | The method produced lower delay and latency than existing methods using received signal strength indicators (RSSI). |
| (Alraih *et al.*, 2022) | Simulation using MATLAB. The environment contains 61 base stations within a 3km\*3km area. They are 400m apart and each cover a 200m radius. Users can move straight within eight directions. | RSRP, RSRQ and UE velocity | Low time-to-trigger and handover margin | A fuzzy logic controller is used that exploits information on RSRP, RSRQ and UE velocity to automatically configure handover control protocols. | The technique was successful. Handover probability was reduced by 95%, handover failure rate was reduced by 95.8%, handover ping-pong rate was reduced by 97%, handover latency was reduced by 94.7% and handover interruption time was reduced by 95%. The overall improvement upon existing techniques was 95.5%. |
| (Vankayala *et al.*, 2022) | Simulation using ray tracing. An urban location with various blockages is modelled. | SINR | Accurate blockage prediction | Recurrent neural networks and convolutional neural networks are used to predict the 5G towers with future link blockages and enable proactive handover. | The method predicted blockages with 90% accuracy, which improves on using wireless features. The CNN-LSTM hybrid also needed relatively few predictive parameters compared to other methods. |
| (Elkourdi, Mazin and Gitlin, 2019) | Simulation. The environment contains eight base stations and their RSSI information. The three closest base stations to the user are used for prediction. | RSSI | High handover prediction accuracy | A recurrent neural network is used to predict future RSSI statistics based on past statistics and make an informed handover decision accordingly. | The method had an accuracy of 92% for handover prediction. |
| (Piran *et al.*, 2017) | Simulation. 10-20 states were set for each base station, and request arrival was assumed to follow a Poisson distribution. | Sensing accuracy and channel idle duration | Low latency and high QoE | A Markov model is used to predict traffic patterns and success probability for a given handover. | The model was effective at providing seamless streaming and lowering latency and handover delay. |
| (Kaloxylos *et al.*, 2014) | Simulation using NS3 network simulator and C++. The scenario is a simplified shopping mall, with two rows of cells denoting shops and a central corridor. Base stations are 20m apart. | RSRQ, base station load, user mobility information and latency sensitivity of used application. | Increased throughput, reduced delay, and reduced handover. | A fuzzy logic controller is used to combine various diverse inputs, including the load of potential base stations and user information, and predict handover. | Handovers were reduced by 53.6% under low load and 84.6% under high load. The proposal is a realistic method of gathering and using information without significant network changes. |
| (Boutiba, Bagaa and Ksentini, 2021) | Testbed. Values were collected based on connectivity being both available and not available. | RSRQ, channel quality index and power headroom | Connection status prediction accuracy | An LSTM was used to predict radio statistics and an SVM was used to classify connectivity status and predict radio link failure for handover prediction. | The model had an accuracy of 98% at predicting radio link failure and connectivity status. |

## Vertical Handover Prediction

Mei *et al.* (2022) utilised a classification-regression hybrid model using real-world data. While the approach predicted handover with 80% accuracy, the dataset only encompassed five public transport routes and eight trips. Using a more extensive dataset would better determine the approach’s real-world efficacy.

Majid, Shah and Marwat (2020) used extreme gradient boosting to predict the difference between 4G and 5G network features to predict handover. While a high handover success rate of 96-98% was achieved, only one usage scenario was tested. Testing more usage scenarios would have better evaluated real-world performance.

Numerous works (Zineb, Ayadi and Tabbane, 2017; Tan, Chen and Sun, 2020; Hussain and Yusof, 2021; Kunarak and Duangchan, 2021) utilised neural networks to predict vertical handover. While the results were successful, all works used simulated data. Using real-world data would have assessed practical efficacy more accurately.

Lahby, Essouri and Sekkaki (2019) utilised Dijkstra’s algorithm to predict the best handover path. While Dijkstra’s algorithm is a perfect search algorithm, which ensures greater efficacy, the A\* algorithm is a more efficient perfect search algorithm (Candra, Budiman and Pohan, 2021). Therefore, A\* may have been more optimal.

Pramod Kumar and Sagar (2019) utilised a Markov decision process to predict the most optimal handover. While the method did increase handover success rate, it could have used more metrics alongside RSSI.

Some works (Demydov *et al.*, 2015; Subramani and Kumaravelu, 2019) utilised fuzzy logic algorithms to predict handover. While the approaches increased handover success rate, they used relatively few metrics, with metrics including throughput being neglected.

Morattab, Dziong and Sohraby (2019) utilised a random waypoint model to predict handover based on the borders of network coverage regions. While this approach did predict handover more quickly and accurately than others, power was the only metric used alongside borders.

A more in-depth comparison of these works is displayed in Table 2.

Table 2: A comparison of works focusing on vertical handover prediction

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Citation** | **Dataset** | **Predictive Parameter(s)** | **Target Performance Parameter(s)** | **Machine Learning Technique(s) Used** | **Conclusions** |
| (Mei *et al.*, 2022) | Real-world dataset using the New York public transport system. Five different routes were tested, with eight trips being taken for each. To measure the statistics, a mobile app was used. | Bandwidth, LTE-neighbours, RSSI, RSRQ, Echng(Ech), Time advance, Speed, Band | High bandwidth | Classification and regression models were used to predict 4G/5G network handovers, as well as 4G/5G vertical handovers. | The method predicted handover with 80% accuracy, and for bandwidth prediction, it was found that LSTM could mine patterns and accurately predict. |
| (Majid, Shah and Marwat, 2020) | Simulation using Python. The environment had two cells, which were assumed to have a fixed radius and separation distance. Users were scattered throughout the network using Poisson point process. | RSRP | Lower latency, lower delay, and higher handover success rate | An algorithm is employed that predicts the difference between 4G and 5G network statistics by using true and false positive rates for area under the curve. If the difference is high enough, vertical handover will occur. | A handover success rate of 96-98% was exhibited when this technique was used. |
| (Tan, Chen and Sun, 2020) | Simulation using MATLAB. User speed was set to 3km/h, and various KPIs were set as constant for each network type. | Maximum transmission rate, minimum delay, SINR, bit error rate, user moving speed, and packet loss rate | Handover success rate | A neural network is used to predict when vertical handover should occur. | The handover success rate of the method was 90%. |
| (Lahby, Essouiri and Sekkaki, 2019) | Testbed. Base stations are arranged in a k-partite graph pattern and uniformly spaced. | Cost per byte, security, available bandwidth, packet delay, packet jitter, packet loss | Better QoS | K-partite graph theory is used, and the best handover is selected using a Dijkstra-based pathfinding algorithm. | The method increased average throughput and reduced average packet loss, packet delay and packet jitter compared to other methods. |
| (Hussain and Yusof, 2021) | Simulation using C++. The environment contains 100 vehicles, 2 base stations for 5G and two base stations for 4G across a 2.5km\*2.5km area. Vehicles moved straight at a constant speed. | Signal strength, distance, density of the vehicle, data type and line of sight | Handover success probability, handover failure rate, redundant handover rate, mean throughput, packet delay and packet loss | A fuzzy convolutional neural network (FCNN) is used to choose a network by considering the various prediction parameters. | Handovers occurred more effectively using the FCNN. The method provided increased handover success compared to existing methods. |
| (Pramod Kumar and Sagar, 2019) | Simulation using NS2. 15 users are simulated moving through the network in a straight line. WiMAX, LTE and WAN base stations are located within the network. WiMAX covers a 2.5km radius, LTE covers a 1km radius and WAN covers a 3km radius. | RSSI | Increased handover success rate | A Markov decision process determines whether vertical handover should take place. | The method increased handover success rate compared to existing methods. |
| (Demydov *et al.*, 2015) | Simulation. User mobility is simulated using Brownian motion space models. Users are given an initial direction and speed that may change. | User direction and velocity | High handover success rate | A cloud computing-based fuzzy logic algorithm is used to predict when vertical handovers will occur. | The method was successful at determining the correct cell for vertical handover. |
| (Morattab, Dziong and Sohraby, 2019) | Simulation. A device-to-device network is simulated, containing base stations that cover a 3km radius. The scenario is an urban area, with a constant user mobility speed of 5m/s. | Maximum power, distance from border, and critical direction | High throughput | A random waypoint-based mobility model is used to track user mobility and determine whether vertical handover is needed. | The method provided precise borders of different regions more quickly than other previously proposed methods. |
| (Kunarak and Duangchan, 2021) | Simulation. The network contains WLAN, WiMAX and LTE base stations, with WLAN having a 100m coverage radius, WiMAX having a 1km coverage radius and LTE having a 2km coverage radius. | RSSI, bandwidth and mobility speed | Reduced handovers, reduced blocked calls, high throughput, and low latency | A back-propagation neural network (BPNN) and radial basis function neural network (RBFNN) are combined to create a hybrid artificial neural network (HANN) that uses a switching mechanism to dynamically select the optimal network. | The method outperforms existing approaches, with higher handover accuracy being attained. |
| (Subramani and Kumaravelu, 2019) | Simulation using MATLAB. The network contains LTE, WiMAX, Wi-Fi, HSPA and GSM base stations. Each device has fixed power allocation, but bandwidth allocation varies according to demand. | RSS, latency, and data rate. | Low decision delay and computational complexity | A two-stage fuzzy logic-based decision algorithm is used to select a suitable network according to various QoS factors. | The method had lower decision delay than other methods. |
| (Zineb, Ayadi and Tabbane, 2017) | Simulation using MATLAB and SIMULINK. The network contains 3G, 4G and Wi-Fi networks covering radiuses of 100m (3G/4G) and 1km (Wi-Fi). Users move in a random waypoint fashion, and mobility speed can be up to 50km/h. Up to 20 users can be within the environment at once, although the exact total is random. | Data rate, packet delay, user class, latency, mean opinion score, RSS, SNR, battery level | High vertical handover accuracy | An artificial neural network (ANN) is proposed that chooses an optimal network depending on the various QoS factors. | The method exhibited better handover accuracy than other methods. |

## Mobility-Based Handover Prediction

Abdah, Barraca and Aguiar (2020) used forecasting based on user information and clustering based on time of day to predict handover. While the approach worked well, more predictive parameters, such as throughput, could have improved the approach further.

Ma, Chen and Zhang (2021) used a neural network hybrid model to predict handover. While the approach produced favourable results and reduced unnecessary handovers, using QoS metrics alongside user trajectory may have increased accuracy further.

Numerous works (Wang *et al.*, 2018; Panda, Ramakrishnan and Bhuyan, 2022; Wei *et al.*, 2023) utilised neural networks using simulated data to predict handover. While the models yielded successful results, the approach should be tested on real-world data to confirm its practical efficacy.

Many works (Plachy, Becvar and Strinati, 2016; Sharma, You and Kumar, 2018; Aboud, Touati and Hnich, 2021; Shaddad *et al.*, 2022) employed handover decision processes using simulated data. While the approaches worked successfully, they were not tested on real-world data, so their real-world effectiveness could not be assessed. They also neglected predictive parameters such as throughput.

Banna, Elattar and El-Dahab (2021) utilised a fuzzy logic algorithm to offer a customised handover solution. While the approach worked effectively, it neglected QoS metrics such as SINR and throughput.

Karmakar, Kaddoum and Chattopadhyay (2022) employed a hybrid of reinforcement learning and a Kalman filter to make an intelligent mobility management algorithm. The approach predicted 5G handovers accurately, but it was not tested on real-world data or with other network types at play, so its real-world efficacy is unknown.

Song, Fang and Yan (2014) utilised grey system theory to predict future QoS metric values and trigger handover accordingly. While the approach worked well at reducing handover failure rate and predicting the correct RAT more accurately, it neglected QoS metrics such as throughput.

Juan *et al.* (2022) used an antenna gain-based solution to predict handover on satellite cells. While the approach enhanced accuracy and reduced error, RSRP was the only QoS metric used. Using more QoS metrics may have enhanced accuracy even further.

Hasan, Kwon and Oh (2019) utilised a classification algorithm to classify users as fast-moving users or ping-pong users. While the approach worked well, reducing handover by 80% and increasing throughput by 11%, it only considered fast-moving users. Also considering static users may have further enhanced the approach.

A more in-depth comparison of these works is displayed in Table 3.

Table 3: A comparison of works focusing on handover prediction in mobile scenarios

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Citation** | **Dataset** | **Predictive Parameter(s)** | **Target Performance Parameter(s)** | **Machine Learning Technique(s) Used** | **Conclusions** |
| (Abdah, Barraca and Aguiar, 2020) | Real-world dataset. Data was collected by volunteers in cars, using an Android app that collected mobility and network information. The city used was Aveiro, Portugal, and 1,971 trajectories were analysed. | Power | Low latency | Probabilistic and enhanced handover prediction approaches, forecasting based on user information and clustering based on time of day. These were compared to the k-nearest neighbours and decision tree machine learning algorithms. | Both approaches usurped the machine learning algorithms in accuracy. The enhanced approach in particular yielded high accuracy. The machine learning algorithms performed better with more predictive features, however. |
| (Ma, Chen and Zhang, 2021) | Real-world dataset. The trajectory data of 182 users was measured over 5 years. For frequent users, 80% of their time was used as training data and 20% was used as testing data. | User trajectory | Reduced handover | Hybrid of LSTM and CNN. User trajectory data is given to the LSTM as an input. The LSTM’s prediction output is then given to the CNN so that it can make a handover decision. | The method was more stable for predicting user trajectory and reduced unnecessary handovers. |
| (Panda, Ramakrishnan and Bhuyan, 2022) | Testbed containing a trace-based 5G streaming dataset and a SUMO-based vehicular mobility dataset. The vehicular mobility dataset simulates 700,000 trips across 247 base stations. | Latency and loss | Low latency | Recurrent neural network that underpins a new 5G user plane function. | Latency was significantly reduced compared to existing user plane functions. |
| (Aboud, Touati and Hnich, 2021) | Simulation. A 2km\*2km environment was simulated, containing 200 vehicles and 10 base stations. Vehicles were moving at speeds of between 5 and 25m/s. | RSRP and vehicle trajectory | Reduction in handovers | Vehicle trajectory prediction paired with a Markov chain predictor to predict vehicle trajectory and close towers. | The method outperformed existing methods and reduced handovers by 60%. |
| (Banna, Elattar and El-Dahab, 2021) | Simulation using MATLAB. The method was tested under different network loads. | RSRP | Low handover failure rate, latency and transfer time. | Fuzzy logic algorithm that offers a customised network performance solution. | The method successfully overcame the issue of handover latency and reduced the overhead disruption of handovers. |
| (Karmakar, Kaddoum and Chattopadhyay, 2022) | Simulation using NS3.33. 50 base stations are present within the environment. 10 users were placed within the range of each base station following a Poisson distribution. User mobility speed varied between 50km/h and 350km/h. | RSRP and RSRQ | Low handover failure rate | A Kalman filter and reinforcement learning are used to make the Learning-based Intelligent Mobility Management (LIM2) algorithm. | The method was effective at predicting 5G handovers, with the target cell being intelligently selected. |
| (Shaddad *et al.*, 2022) | Simulation. A 4km\*4km simulation environment was used, containing 50 base stations that each covered a 100m radius. User mobility speeds varied between 30km/h and 130km/h. | RSRP and user speed | Low handover failure rate | A handover prediction algorithm is used to intelligently predict handovers in high-mobility scenarios. | Handover failure rate was reduced with the dynamic handover protocol algorithm compared to the static handover protocol algorithm. |
| (Song, Fang and Yan, 2014) | Simulation. A high-speed rail scenario was simulated. | RSRP, RSRQ and SIR | Lower time-to-trigger | Grey system theory is used to assume that future measurement results can be predicted from current measurements. | The method triggered handover earlier and improved handover success rate. |
| (Wang *et al.*, 2018) | Simulation. Mobility data was generated using the Self-Similar Least-Action Human Walk (SLAW) and SMOOTH models. A 4km\*4km simulation area was used, and base stations were placed using a Poisson point process. | RSSI and user mobility information | High handover prediction accuracy | An LSTM neural network is used to predict future user mobility based on past mobility. | The method exhibited high handover prediction accuracy for users with low and medium mobility speed. |
| (Juan *et al.*, 2022) | Simulation. A satellite network at an altitude of 600km, providing coverage to a sparse number of ground users, was simulated. Each satellite covered a 50km ground radius. | RSRP | Low location error, antenna radiation error and pointing error, low handover frequency. | An antenna gain-based solution is used to perform handover in orbital satellite cells. | The method enhanced accuracy and reduced error compared to traditional methods. |
| (Wei *et al.*, 2023) | Simulation using NS3. Three different scenarios were considered: free space, highway and high-speed rail. Mobility speeds varied between 30 and 100m/s. | RSRQ | High handover success rate. | A deep neural network is used to select the target cell. | The method was highly successful at completing prompt handovers in fast-mobility scenarios. |
| (Plachy, Becvar and Strinati, 2016) | Simulation using MATLAB. An 800m\*800m area containing 19 base stations and 200 users was simulated. User mobility speed was 1m/s. | SINR and capacity | Reduced handover | A Markov decision process is used to dynamically allocate resources to a given base station. | The method successfully in reducing handovers by 10-66% while also keeping overheads at a similar level to existing methods. |
| (Hasan, Kwon and Oh, 2019) | Simulation using NS3. A HetNet with three macro base stations and 20 small base stations was simulated. The macro base stations were placed in a hexagonal grid and the small base stations were placed in a Manhattan layout. | RSRP and user mobility information | Reduced handover and increased throughput | A classification algorithm is used to classify users as fast-moving users or ping-pong users. | The method reduced handovers by 79.6% and increased throughput by 10.8%. |
| (Sharma, You and Kumar, 2018) | Simulation using MATLAB. The number of users varied between 10,000 and 100,000. The range of each base station was 500m and the maximum user distance from a base station was 1km. | System resources including bandwidth | Reduced latency during handover | A Markov model is used to predict user mobility patterns, and an n-step algorithm is used for congestion prediction and optimal route selection. | Handover latency was reduced compared to existing solutions. The method had a minimum latency of 5.9ms, a maximum latency of 9.1ms, and an average latency of 6.5ms. |

## Literature Gap in Existing Studies

While studies relating to vertical handover prediction based on user mobility exist, gaps are present within current literature.

Firstly, no studies used throughput as a predictor despite it being an important QoS indicator (Mehmeti and Porta, 2021). Some works used theoretical measures, such as data rate and/or bandwidth, but these only consider the best-case throughput. Xiao and Rosdahl (2002) found that when theoretical measures are increased, real-world throughput does not necessarily increase in tandem.

Secondly, few studies used real-world data, with most using simulated or testbed data. While simulated data can replicate real-world data effectively where it is unavailable, the simulation parameters often make inaccurate assumptions.

# Methodology

## Experimental Method

In this study, three RAT selection methods were tested. These methods were used to select the correct RAT to connect to at a given time according to various network statistics. Two real-world datasets were used, which were both split into a *throughput* subset, a *status quo* subset containing commonly used signal strength variables, and a *combined* subset encapsulating both. Five different variable combinations, shown in Table 4, were tested.

Table 4: The variable combinations used

|  |  |
| --- | --- |
| **Combination** | **Subset and Predictors Used** |
| 1 | *Status quo*; signal strength variables only |
| 2 | *Throughput*; downlink and uplink throughput variables |
| 3 | *Combined*; downlink/uplink throughput and signal strength variables combined |
| 4 | *Throughput*; downlink throughput variables only |
| 5 | *Throughput*; uplink throughput variables only |

To address the research aim of evaluating the efficacy of AI methods for vertical handover prediction, three chosen methods were tested using combination 1. The leading method was then used to address the research aim of evaluating the efficacy of throughput as a predictor. This was tested using combination 2, with combinations 1 and 3 being used for comparison. Complexity reductions were then explored, with combinations 4 and 5 being tested to evaluate the efficacy of fewer predictors. Lower AI algorithm hyperparameters were also tested using combination 2.

All three algorithms were evaluated using 10-fold cross validation because it reduces bias potential and emphasises method evaluation only (Jung, 2018). While k-fold cross validation can be computationally intensive (Yadav and Shukla, 2016), limiting the k value can circumvent this. Here, a k value of 10 was chosen, as Marcot and Hanea (2021) found this to provide an optimal balance between reliability and computational complexity.

In this study, real-world data was used over simulated data because little existing literature used real-world data. While simulated datasets can be larger, the data quality is also often lower, as potentially inaccurate assumptions are made (Kang *et al.*, 2019).

## Initial Dataset

For this study, a real-world throughput dataset by Kousias *et al.* (2023) was initially used. The data was gained from a study of mobile networks in Rome, and it contains 4G and 5G network data. Three mobility scenarios are encapsulated: inside, outside walking and outside driving. The study was performed across seven weeks in December 2020 and January 2021, and the Android application Qualipoc was used to gain readings, running on a Samsung S20 with 5G enabled. To test throughput, speedtests were performed. Files were downloaded, and throughput figures were measured across each download.

### Dataset Attributes

Table 5 shows the common variables used across all three subsets.

Table 5: The common variables used in all subsets

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Numerical or Categorical?** | **Predictor or Target Variable?** | **Description** |
| GPS Long | Numerical | Predictor | The GPS-measured longitude of the user. |
| GPS Lat | Numerical | Predictor | The GPS-measured latitude of the user. |
| RAT Info | Categorical | Target | The RAT that the user is connected to. Here, this is either 4G or 5G. |
| Scenario | Categorical | Predictor | The mobility scenario. In this dataset, this is either Inside Static, Outside Walking or Outside Driving. |

Table 6 shows the variables exclusive to the *throughput* and *combined* subsets.

Table 6: The variables exclusive to the throughput and combined subsets

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Numerical or Categorical?** | **Predictor or Target Variable?** | **Description** |
| Current Netw. DL | Numerical | Predictor | The current downlink throughput. |
| Current Netw. UL | Numerical | Predictor | The current uplink throughput. |
| Mean Netw. DL | Numerical | Predictor | The mean downlink throughput for the given RAT at a given point. |
| Mean Netw. UL | Numerical | Predictor | The mean uplink throughput for the given RAT at a given point. |
| Current Netw. DL Avg | Numerical | Predictor | The average reading of current downlink throughput across the current measurement period. |
| Current Netw. DL Max | Numerical | Predictor | The highest reading of current downlink throughput across the current measurement period. |
| Current Netw. UL Avg | Numerical | Predictor | The average reading of current uplink throughput across the current measurement period. |
| Current Netw. UL Max | Numerical | Predictor | The highest reading of current uplink throughput across the current measurement period. |

Table 7 shows the variables exclusive to the *status quo* and *combined* datasets.

Table 7: The variables exclusive to the status quo and combined datasets

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Numerical or Categorical?** | **Predictor or Target Variable?** | **Description** |
| SS-RSRP | Numerical | Predictor | The amount of power received by the network. |
| SS-RSRQ | Numerical | Predictor | The quality of the main signal being received by the network. |
| SS-SINR | Numerical | Predictor | The ratio of wanted signal being received by the network compared to the amount of unwanted interference and noise being received. |

All three subsets also contained measurement date and time. These were not employed as predictors, but were utilised during pre-processing.

Following missing data removal, the *throughput* subset had 240,510 rows, while the *status quo* subset had 221,949 rows and the *combined* subset had 221,635 rows.

The dataset contains 4G and 5G network data. Figure 3 shows the distribution of 4G versus 5G in the *throughput* subset.

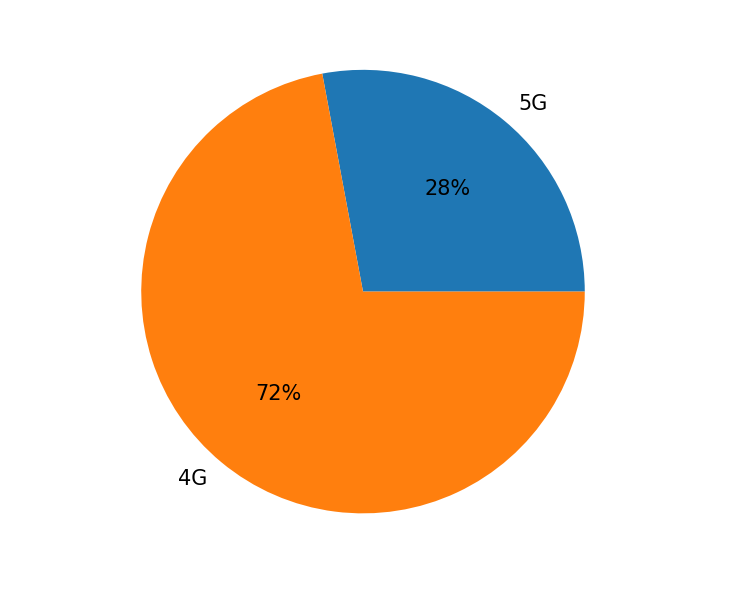


Figure 3: The distribution of 4G versus 5G within the throughput subset

Figure 4 shows the distribution within the *status quo* subset.

A blue and orange pie chart

Description automatically generated

Figure 4: The distribution of 4G versus 5G within the status quo subset

Figure 5 shows the distribution within the *combined* subset.

A blue and orange pie chart

Description automatically generated

Figure 5: The distribution of 4G versus 5G within the combined subset

All three subsets exhibit class imbalance, with 4G being almost three times more prevalent than 5G. Resampling was considered, but was not implemented. Although resampling can improve accuracy and reduce bias (Ramos-Pérez *et al.*, 2022), it can skew dataset relationships (Hall, 1985). Also, resampling is typically most successful with higher levels of class imbalance, such as 50:1, 100:1, 1000:1 or higher (Hakim *et al.*, 2022). In this instance, the imbalance ratio was less than 3:1, which was not stark enough to make gains from resampling. When methods such as random oversampling and undersampling, SMOTE oversampling and near-miss undersampling were tested, they were found to decrease accuracy, so resampling was not deemed worthwhile here.

### Pre-Processing

Prior to use, some pre-processing was performed on the dataset.

Firstly, Kousias *et al.* (2023) state that any blank dataset item represents the item remaining unchanged since the last reading. As such, the date and time values were used to separate the data into days and order rows chronologically. Forward fill was then performed on each day so that blank values were replaced with the previous value. Where any data item was still left blank (due to being at the beginning of the time-series), the row was removed, as interpolation could not be reliably performed without any prior reading. Available data remained plentiful without these rows.

## Validation Dataset

For validation, the approach was also tested on an alternative dataset from Raca *et al.* (2018). This was generated in Cork and contained RSRP, RSRQ and downlink/uplink throughput data alongside similar locational and scenario-based attributes to the initial dataset. As with the initial dataset, the scenario is a speedtest. The dataset encompasses data from 3G and 4G networks, with the proportion of 4G to 3G mirroring that of 4G to 5G in the first dataset. Five mobility scenarios are encapsulated: train, static, pedestrian, car, and bus.

The pre-processing performed on this dataset is described in Appendix 1.

## AI Algorithms

Three AI algorithms were tested on their RAT selection ability: a naïve-bayes classifier (NBC), a random forest (RF) and a neural network (NN). The NN was tested because the literature review revealed numerous works employing NNs successfully for handover prediction, while the RF and NBC were tested due to an absence of existing works using them for handover prediction.

The RF had 100 trees, with a maximum depth of 27. The NN was trained over 100 epochs. It also had three hidden layers, with 32, 64 and 128 nodes respectively, and the stochastic gradient descent (SGD) optimiser was used with a learning rate of 0.01.

More detailed reasoning behind the algorithm choices, as well as comparison with other alternatives, is shown in Table 8.

Table 8: A comparison of the chosen algorithms to alternatives

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Advantage** | **Disadvantage** | **Chosen? (Yes/No)** | **Reason for Decision** |
| Naïve-Bayes Classifier (NBC) | NBCs work well on large datasets with high dimensionality while being less computationally intensive than alternatives. | NBCs assume feature independence, which may not always reflect reality. | Yes | While NBCs assume feature independence, they can still work well when feature dependence exists, as they can ignore irrelevant variables. Therefore, an NBC was suitable overall given their low complexity and ability to work with large datasets. |
| Random Forest (RF) | RFs are flexible, as they make few assumptions about inter-variable relationships. They are also well-suited to large datasets. | RFs can be prone to overfitting if hyperparameters are not selected correctly. | Yes | The overfitting disadvantage can be easily overcome by selecting hyperparameters intelligently. Therefore, the flexibility of an RF makes it well-suited to this dataset. |
| Neural Network (NN) | NNs can implicitly detect nonlinear relationships between predictor and target variables, and any inter-variable relationships are detectable. | NNs can be computationally intensive and train slowly. | Yes | Despite their complexity, the ability of an NN to detect a variety of inter-variable relationships makes it a good fit for this dataset. |
| Logistic Regression (LoR) | LoR shares many advantages of linear regression (LiR) in terms of low complexity while not sharing the same assumptions regarding normal distribution and collinearity. | LoR is suited to use cases where the target variable is binary. The target variable may not always be binary here. | No | While LoR has low complexity and is not bound by the assumptions of LiR, the target variable may not always be binary in this context, which makes it sub-optimal here. |
| Support Vector Machine (SVM) | SVMs are advantageous for dealing with high-dimensionality datasets, as well as non-linear relationships between variables. | SVMs do not suit non-binary classification or class imbalance. They can also be computationally intensive when applied to large datasets. | No | While an SVM may have been advantageous given the dataset’s high dimensionality, it would not have been well-suited to its class imbalance or large size. |
| Decision Tree (DT) | DTs are simple, fast and can deal with noise while still producing high accuracy. | Unlike an RF, a standalone DT does not deal well with high-dimensionality datasets | No | As this dataset has high dimensionality, a DT would not have been optimal here. |

(Jarvis and Stuart, 1996; Tu, 1996; Rish, 2001; Pal and Mather, 2003; Cheng, Varshney and Arora, 2006; Kurt, Ture and Kurum, 2008; Jadhav and Channe, 2016; Cervantes *et al.*, 2020; Feng *et al.*, 2020; Jun, 2021; Langsetmo *et al.*, 2023)

# Results and Evaluation

As described above, all approaches were tested on an initial dataset and a validation dataset using 10-fold cross validation. The *status quo* subset was initially tested on all three AI algorithms, alongside two dummy classifiers using random or systematic methods to provide baseline accuracy measures. The *throughput* subset was then tested on the most optimal AI algorithm, alongside the *status quo* and *combined* subsets for comparison. Finally, the individual throughput types were tested in isolation and compared to the full *throughput* subset to explore the possibility of complexity reduction.

Accuracy, precision, and recall were used to assess each approach. Accuracy measures the percentage of correct classifier judgements. Precision measures the percentage of classified positives that are true positives. Recall measures the percentage of true positives that were highlighted.

## Classifier Comparison

Initially, the chosen classifiers were compared to address the research aim of testing whether AI methods could make accurate RAT selections. These classifiers were compared to dummy classifiers using the *status quo* subset as predictive parameters. The initial dataset results are shown in Table 9 and Figure 6.

Table 9: A table comparing classifier performances on the initial dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| Uniform Dummy Classifier (Baseline) | 50.1% | 50.1% | 50.1% |
| Most Frequent Dummy Classifier (Baseline) | 69.6% | 34.8% | 50.0% |
| Naïve-Bayes Classifier | 68.2% | 65.8% | 68.3% |
| Neural Network | 92.8% | 91.1% | 92.3% |
| Random Forest | 97.1% | 96.5% | 96.5% |

A graph of different colored bars

Description automatically generated

Figure 6: A graph comparing classifier performances on the initial dataset

The validation dataset results are shown in Table 10 and Figure 7.

Table 10: A table comparing classifier performances on the validation dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| Uniform Dummy Classifier (Baseline) | 50.1% | 50.1% | 50.1% |
| Most Frequent Dummy Classifier (Baseline) | 71.5% | 35.7% | 50.0% |
| Naïve-Bayes Classifier | 97.9% | 96.8% | 98.2% |
| Neural Network | 99.7% | 99.7% | 99.6% |
| Random Forest | 100.0% | 100.0% | 100.0% |

A close-up of a graph

Description automatically generated

Figure 7: A graph comparing algorithm performances on the validation dataset

These results prove that AI methods can make accurate RAT predictions; all methods improved on the dummy classifiers in terms of accuracy, precision and/or recall. However, the random forest (RF) was a notable frontrunner. With accuracy scores of 97-100% and precision and recall scores of 96-100%, it boasted 40-100% higher accuracy and 93-180% higher precision and recall than baseline accuracy, as well as stronger performance than the naïve-bayes classifier (NBC). While the results from the neural network (NN) corroborate the positive verdict of existing literature, the RF still had up to 5% stronger accuracy and up to 6% stronger precision and recall while also not sharing its high computational complexity (Maji and Mullins, 2018). Therefore, the RF has proven itself to be an optimal AI algorithm compared to other alternatives. Given its lack of use in existing literature, it can be concluded that RFs have considerable untapped potential in handover prediction.

## Comparison of Throughput to Status Quo Variables

With the RF having been selected as the optimal AI algorithm to apply here, the research aim of assessing the efficacy of throughput as a vertical handover predictor was then tested using the RF. The *throughput*, *status quo* and *combined* subsets were tested as predictors, using the RF as a classifier. The initial and validation dataset results are shown in Table 11 and Table 12 respectively.

Table 11: A table comparing subset performances on the initial dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Subset** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| Throughput | 97.4% | 96.7% | 96.7% |
| Status Quo | 97.1% | 96.5% | 96.5% |
| Combined | 97.4% | 97.0% | 96.9% |

Table 12: A table comparing subset performances on the validation dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Subset** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| Throughput | 99.3% | 99.0% | 99.2% |
| Status Quo | 100.0% | 100.0% | 100.0% |
| Combined | 100.0% | 100.0% | 100.0% |

These results evidence that throughput can predict vertical handover accurately. However, it is not notably more accurate than the status quo variables. In the initial dataset, it improved slightly upon the status quo variables, but in the validation dataset, it returned slightly weaker results. Across both datasets, it was found that the strongest results were most often gained using the *combined* subset. However, the difference was insignificant, and all subsets returned strong results. This would infer that both throughput and signal strength can be effective predictors of handover in isolation, and there is little to be gained from using both together.

## Complexity Improvements

With throughput being proven to be an effective predictor of vertical handover, the feasibility of lowering the classifier complexity was then tested. The classifier may be used on mobile phones rather than at the base stations themselves, so a model with lower complexity may be desirable.

Firstly, downlink and uplink throughput in isolation were each tested on the RF and compared to the combined *throughput* subset to see whether less variables can be used to make an effective prediction. The initial and validation dataset results are shown in Table 13 and Table 14 respectively.

Table 13: A table comparing different throughput type performances on the initial dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Predictors** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| Downlink and Uplink Throughput | 97.4% | 96.7% | 96.7% |
| Downlink Throughput | 97.3% | 96.6% | 96.6% |
| Uplink Throughput | 97.4% | 96.8% | 96.8% |

Table 14: A table comparing different throughput type performances on the validation dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Predictors** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| Downlink/Uplink Throughput | 99.3% | 99.0% | 99.2% |
| Downlink Throughput | 98.9% | 98.5% | 98.8% |
| Uplink Throughput | 98.5% | 97.9% | 98.3% |

These results elucidate that scope for using fewer predictive variables exists. While using downlink and uplink in isolation did often lead to slight performance drops, no drop higher than around 1% was logged. Therefore, this may be a worthwhile trade-off for a more lightweight model. The optimal throughput type varied. The initial dataset returned uplink as more accurate, while the validation dataset returned downlink as more accurate. Nevertheless, both returned accurate results in both datasets, so either alone could be used for an accurate model.

Secondly, the possibility of lowering the RF hyperparameters was tested. The RF with the initial hyperparameters of 100 trees and a maximum depth of 27, using the *throughput* subset as predictors, was compared to other RFs with lower tree numbers and lower maximum depths. The initial and validation dataset results for tree number are shown in Table 15 and Table 16 respectively.

Table 15: A table comparing tree number performances on the initial dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Tree Number** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| 100 (Baseline) | 97.4% | 96.7% | 96.7% |
| 50 | 97.4% | 96.7% | 96.7% |
| 25 | 97.3% | 96.7% | 96.7% |
| 10 | 97.2% | 96.6% | 96.6% |
| 5 | 97.2% | 96.5% | 96.5% |
| 1 | 96.7% | 95.8% | 96.0% |

Table 16: A table comparing tree number performances on the validation dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Tree Number** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| 100 (Baseline) | 99.3% | 99.0% | 99.2% |
| 50 | 99.3% | 99.1% | 99.2% |
| 25 | 99.3% | 99.0% | 99.2% |
| 10 | 99.3% | 99.0% | 99.2% |
| 5 | 99.2% | 99.0% | 99.1% |
| 1 | 98.9% | 98.5% | 98.7% |

A comparison of tree number accuracies in both datasets is shown in Figure 8.

A graph with numbers and dots

Description automatically generated with medium confidence

Figure 8: A scatter plot comparing tree number accuracies on both datasets

The initial and validation dataset results for maximum depth are shown in Table 17 and Table 18 respectively.

Table 17: A table comparing maximum depth performances on the initial dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Maximum Depth** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| 27 (Baseline) | 97.4% | 96.7% | 96.7% |
| 20 | 97.2% | 96.7% | 96.3% |
| 15 | 96.5% | 96.1% | 95.3% |
| 10 | 93.6% | 93.0% | 90.9% |
| 5 | 86.9% | 86.5% | 79.9% |
| 1 | 76.3% | 83.6% | 58.1% |

Table 18: A table comparing maximum depth performances on the validation dataset

|  |  |  |  |
| --- | --- | --- | --- |
| **Maximum Depth** | **Accuracy (1dp)** | **Precision (1dp)** | **Recall (1dp)** |
| 27 (Baseline) | 99.3% | 99.0% | 99.2% |
| 20 | 99.0% | 98.6% | 98.9% |
| 15 | 97.7% | 97.2% | 97.1% |
| 10 | 93.9% | 93.2% | 91.7% |
| 5 | 85.4% | 85.5% | 77.3% |
| 1 | 78.0% | 74.6% | 66.7% |

A comparison of maximum depth accuracies on both datasets is shown in Figure 9.

A screen shot of a graph

Description automatically generated

Figure 9: A scatter plot comparing maximum depth accuracies on both datasets

These results encapsulate that the RF hyperparameters can be reduced while retaining accuracy. Reducing the tree number had a limited impact on performance in both datasets. Reducing the maximum depth had a stronger impact, but this was still relatively limited until maximum depth reached 10 or below in both datasets. Therefore, high accuracy can be retained with lower RF hyperparameters.

## Evaluation

Research of existing literature on handover prediction found that throughput was underutilised as a predictor and real-world data was used in few studies. This study aimed to plug this gap by testing the applicability of throughput as a predictor in comparison with the signal strength variables commonly used in existing literature. Real-world data was also used to test the efficacy of the methods proposed. Previously unused AI methods were also tested in the form of a Naïve-Bayes Classifier (NBC) and a Random Forest (RF). These were compared to an existing successfully applied method in the form of a Neural Network (NN).

The study firstly aimed to test whether AI could accurately predict the correct RAT to connect to. It was found that overall, AI methods could make accurate predictions. All three tested methods made notable improvements on baseline accuracy in terms of accuracy, precision and/or recall across both datasets.

The most optimal AI method was the RF, which attained 97-100% accuracy and 96-100% precision and recall. This represents a 40-100% accuracy improvement and a 93-180% precision and recall improvement over baseline accuracy, indicating that they are highly effective at making accurate RAT selections. Later tests also found that there was scope for lowering the RF hyperparameters for decreased complexity while maintaining high accuracy, with tree number being particularly flexible. As RFs are unused in existing literature, these results, showing high accuracy and scalability potential, would evidence that they present considerable untapped opportunities in vertical handover prediction.

The NN was also highly accurate, attaining 93-99% accuracy and 91-99% precision and recall. This represents a 33-99% accuracy improvement and an 82-179% improvement in precision and recall over baseline accuracy. NNs have been applied successfully in existing literature, and these results corroborate earlier findings. However, the considerably higher computational complexity of NNs compared to the other tested approaches (Maji and Mullins, 2018) should be considered. Therefore, NNs, while successful in making accurate predictions, are not as optimal as RFs due to their increased computational complexity and slightly lower average performance.

The NBC attained 68-98% accuracy and 65-98% precision and recall, representing an accuracy improvement of up to 95% and precision and recall improvement of up to 171% over baseline accuracy. NBCs are unused in existing literature, and these results evidence that they can predict accurately. However, their average performance was weaker than that of the other tested methods. Therefore, they are not as optimal despite their favourable computational complexity.

The study also aimed to test whether throughput as a predictor could make accurate RAT selections. It was found that throughput could make accurate predictions, but it did not notably improve accuracy compared to existing commonly used variables.

While it was found that throughput was most effective as a predictor when paired with signal strength variables, the improvement from combining was not significant, and both were found to be highly accurate predictors in isolation. This could be because while high throughput is important to user experience (Wigelius and Väätäjä, 2009), signal strength variables are also important to consider, with SINR having a particularly strong impact (Balachandran *et al.*, 2014). Thus, both variable subsets are important and can both be highly accurate handover predictors.

An interesting finding is that signal strength variables were more accurate than throughput on the validation dataset, but slightly less accurate than throughput on the initial dataset. The initial dataset contains 5G network data and the validation dataset does not. This could indicate that the dawn of 5G networks has lessened the significance of signal strength variables in handover decisions, with throughput having increased significance. With 5G architecture being reliant on a larger number of smaller cells (Sutton, 2018), this could have shifted the focus from signal strength to throughput.

The potential for complexity reduction was also tested by testing the accuracy of downlink and uplink throughput alone compared to both combined. These tests often found that the two throughput types combined was more optimal than either alone. This could be because even though network use has traditionally centred around downlink throughput (Boccardi *et al.*, 2016) due to downlink-intensive activities being used more (Qualcomm, 2023), uplink throughput is becoming increasingly important with the rise of activities such as real-time gaming (Elshaer *et al.*, 2014). Therefore, neither throughput type should be ignored if optimal user experience is to be provided.

However, both downlink and uplink throughput in isolation still yielded high accuracy in these tests. The drop in accuracy compared to using both throughput types combined was not overly significant. Therefore, the trade-off between lost accuracy and decreased complexity may be worthwhile for a more lightweight model. In terms of the more powerful throughput type, the initial and validation datasets returned opposing results. The initial dataset returned higher performance with uplink, while the validation dataset returned higher performance with downlink. As the initial dataset considers 5G while the validation dataset does not, this could infer that uplink throughput is becoming increasingly important in handover decisions with the advent of 5G. Uplink-intensive activities such as hybrid working and IoT applications have risen in prominence (Eyceyurt, Egi and Zec, 2022), better reflecting the demands of modern users. Nevertheless, both throughput types returned high accuracy on both datasets, so both could be used for an accurate model.

## Limitations and Further Work

This study has limitations. Firstly, neither of the datasets used encapsulated more than two network types. In the real world, more than two network types will often coexist. This study could not consider this due to limitations of the data used, but it is still important to acknowledge.

Secondly, neither dataset considered a usage scenario other than a speedtest. With common smartphone activities including web browsing, streaming and online gaming (Qualcomm, 2023), other use cases could be considered. Different use cases may give different results.

Recommendations for further work can be made accordingly. Firstly, testing the approach on data with more coexisting network types could yield interesting results, as the problem may differ when more than two coexist. Secondly, considering use cases aside from a speedtest could also produce interesting insights. It may be particularly interesting to embed multiple use cases and consider the use case as a predictor. Each one will have different throughput requirements, which may change the outcome of the RAT selection and reduce unnecessary handovers.

# References

Abdah, H., Barraca, J.P. and Aguiar, R.L. (2020) ‘Handover Prediction Integrated with Service Migration in 5G Systems’, in *IEEE International Conference on Communications (ICC)*.

Aboud, A., Touati, H. and Hnich, B. (2021) ‘Handover optimization for VANET in 5G networks’, in *2021 IEEE 18th Annual Consumer Communications and Networking Conference, CCNC 2021*. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/CCNC49032.2021.9369552.

Abu Alsheikh, M. *et al.* (2015) ‘Markov decision processes with applications in wireless sensor networks: A survey’, *IEEE Communications Surveys and Tutorials*, 17(3), pp. 1239–1267. Available at: https://doi.org/10.1109/COMST.2015.2420686.

Alhabo, M. and Zhang, L. (2018) ‘Load-dependent handover margin for throughput enhancement and load balancing in HetNets’, *IEEE Access*, 6, pp. 67718–67731. Available at: https://doi.org/10.1109/ACCESS.2018.2878489.

Alraih, S. *et al.* (2022) ‘Robust Handover Optimization Technique with Fuzzy Logic Controller for Beyond 5G Mobile Networks’, *Sensors*, 22(16). Available at: https://doi.org/10.3390/s22166199.

Balachandran, A. *et al.* (2014) ‘Modeling web quality-of-experience on cellular networks’, in *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*. Association for Computing Machinery, pp. 213–224. Available at: https://doi.org/10.1145/2639108.2639137.

Banna, R. El, Elattar, H.M. and El-Dahab, M.M.A. (2021) ‘Fast Adaptive Handover using Fuzzy Logic for 5G Communications on High Speed Trains’, in *Proceedings of the 16th International Conference on Telecommunications, ConTEL 2021*. Institute of Electrical and Electronics Engineers Inc., pp. 10–17. Available at: https://doi.org/10.23919/ConTEL52528.2021.9495988.

Boccardi, F. *et al.* (2016) ‘Why to decouple the uplink and downlink in cellular networks and how to do it’, *IEEE Communications Magazine*, 54(3), pp. 110–117.

Boutiba, K., Bagaa, M. and Ksentini, A. (2021) ‘Radio Link Failure Prediction in 5G Networks’, in *2021 IEEE Global Communications Conference, GLOBECOM 2021 - Proceedings*. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/GLOBECOM46510.2021.9685587.

Candra, A., Budiman, M.A. and Pohan, R.I. (2021) ‘Application of A-Star Algorithm on Pathfinding Game’, in *Journal of Physics: Conference Series*. IOP Publishing Ltd. Available at: https://doi.org/10.1088/1742-6596/1898/1/012047.

Ceci, L. (2023) *Mobile internet usage worldwide - Statistics & Facts*, *Statista*. Available at: statista.com/topics/779/mobile-internet/#topicOverview (Accessed: 7 October 2023).

Cervantes, J. *et al.* (2020) ‘A comprehensive survey on support vector machine classification: Applications, challenges and trends’, *Neurocomputing*, 408, pp. 189–215. Available at: https://doi.org/10.1016/j.neucom.2019.10.118.

Charitos, M. and Kalivas, G. (2013) ‘WiMAX-WLAN vehicle-to-infrastructure network architecture during fast handover process’, in *Proceedings - 27th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2013*, pp. 431–436. Available at: https://doi.org/10.1109/WAINA.2013.137.

Cheng, Q., Varshney, P.K. and Arora, M.K. (2006) ‘Logistic regression for feature selection and soft classification of remote sensing data’, *IEEE Geoscience and Remote Sensing Letters*, 3(4), pp. 491–494. Available at: https://doi.org/10.1109/LGRS.2006.877949.

Cicioglu, M. (2021) ‘Performance analysis of handover management in 5G small cells’, *Computer Standards and Interfaces*, 75, p. 103502.

Demydov, I. *et al.* (2015) ‘Mobility Management and Vertical Handover Decision in an Always Best Connected Heterogeneous Network’, in *The Experience of Designing and Application of CAD Systems in Microelectronics*. Lviv, Ukraine: IEEE. Available at: http://www.ijeit.com/archivedescription.php?id=37.

Eker, I. and Torun, Y. (2006) ‘Fuzzy logic control to be conventional method’, *Energy Conversion and Management*, 47(4), pp. 377–394. Available at: https://doi.org/10.1016/j.enconman.2005.05.008.

Elkourdi, M., Mazin, A. and Gitlin, R.D. (2019) ‘Performance Analysis for Virtual-Cell Based CoMP 5G Networks Using Deep Recurrent Neural Nets’, in *2019 Wireless Communications Symposium*, pp. 1–6.

Elshaer, H. *et al.* (2014) ‘Downlink and Uplink Decoupling: a Disruptive Architectural Design for 5G Networks’, in *2014 IEEE Global Communications Conference*, pp. 1798–1803.

Eyceyurt, E., Egi, Y. and Zec, J. (2022) ‘Machine-Learning-Based Uplink Throughput Prediction from Physical Layer Measurements’, *Electronics (Switzerland)*, 11(8). Available at: https://doi.org/10.3390/electronics11081227.

Feng, W. *et al.* (2020) ‘FSRF: An Improved Random Forest for Classification’, in *2020 IEEE Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*. IEEE, pp. 173–178.

Hakim, M. *et al.* (2022) ‘Survey on Highly Imbalanced Multi-class Data’, *International Journal of Advanced Computer Science and Applications (IJACSA)*, 13(6). Available at: www.ijacsa.thesai.org.

Hall, P. (1985) ‘Resampling a Coverage Pattern’, *Stochastic Processes and their Applications*, 20, pp. 231–246.

Hasan, M.M., Kwon, S. and Oh, S. (2019) ‘Frequent-handover mitigation in ultra-dense heterogeneous networks’, *IEEE Transactions on Vehicular Technology*, 68(1), pp. 1035–1040. Available at: https://doi.org/10.1109/TVT.2018.2874692.

Hassan, A. *et al.* (2022) ‘Vivisecting mobility management in 5G cellular networks’, in *SIGCOMM 2022 - Proceedings of the ACM SIGCOMM 2022 Conference*. Association for Computing Machinery, Inc, pp. 86–100. Available at: https://doi.org/10.1145/3544216.3544217.

Hussain, S.M. and Yusof, K.M. (2021) ‘Dynamic Q-learning and Fuzzy CNN Based Vertical Handover Decision for Integration of DSRC, mmWave 5G and LTE in Internet of Vehicles (IoV)’, *Journal of Communications*, 16(5), pp. 155–166. Available at: https://doi.org/10.12720/jcm.16.5.155-166.

Jadhav, S.D. and Channe, H.P. (2016) ‘Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques’, *International Journal of Science and Research (IJSR)*, 5(1), pp. 1842–1845. Available at: www.ijsr.net.

Jarvis, C.H. and Stuart, N. (1996) ‘The Sensitivity of a Neural Network for Classifying Remotely Sensed Imagery’, *Computers & Geosciences*, 22(9), pp. 959–967.

Juan, E. *et al.* (2022) ‘Handover Solutions for 5G Low-Earth Orbit Satellite Networks’, *IEEE Access*, 10, pp. 93309–93325. Available at: https://doi.org/10.1109/ACCESS.2022.3203189.

Jun, Z. (2021) ‘The Development and Application of Support Vector Machine’, in *Journal of Physics: Conference Series*. IOP Publishing Ltd. Available at: https://doi.org/10.1088/1742-6596/1748/5/052006.

Jung, Y. (2018) ‘Multiple predicting K-fold cross-validation for model selection’, *Journal of Nonparametric Statistics*, 30(1), pp. 197–215. Available at: https://doi.org/10.1080/10485252.2017.1404598.

Kaloxylos, A. *et al.* (2014) ‘An efficient RAT selection mechanism for 5G cellular networks’, in *10th International Wireless Communications & Mobile Computing Conference*.

Kang, K. *et al.* (2019) ‘Generalization through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight’, in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6008–6014.

Karmakar, R., Kaddoum, G. and Chattopadhyay, S. (2022) ‘Mobility Management in 5G and Beyond: A Novel Smart Handover with Adaptive Time-to-Trigger and Hysteresis Margin’, *IEEE Transactions on Mobile Computing* [Preprint]. Available at: https://doi.org/10.1109/TMC.2022.3188212.

Kaur, G., Goyal, R.K. and Mehta, R. (2022) ‘An efficient handover mechanism for 5G networks using hybridization of LSTM and SVM’, *Multimedia Tools and Applications*, 81(26), pp. 37057–37085. Available at: https://doi.org/10.1007/s11042-021-11510-x.

Kaya, A.Ö. and Viswanathan, H. (2021) ‘Deep learning-based predictive beam management for 5G mmWave systems’, in *IEEE Wireless Communications and Networking Conference, WCNC*. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/WCNC49053.2021.9417452.

Khan, S.A. *et al.* (2022) ‘Handover management over dual connectivity in 5G technology with future ultra-dense mobile heterogeneous networks: A review’, *Engineering, Science and Technology, an international journal*, 35.

Kousias, K. *et al.* (2022) ‘Implications of handover events in commercial 5G non-standalone deployments in Rome’, in *5G-MeMU 2022 - Proceedings of the ACM SIGCOMM 2022 Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases - Part of SIGCOMM 2022*. Association for Computing Machinery, Inc, pp. 22–27. Available at: https://doi.org/10.1145/3538394.3546041.

Kousias, K. *et al.* (2023) ‘A Large-Scale Dataset of 4G, NB-IoT, and 5G Non-Standalone Network Measurements’, *IEEE Communications Magazine*, 4 September, pp. 1–7.

Kumar, S. *et al.* (2018) ‘Handover forecasting in 5G using machine learning’, *International Journal of Engineering & Technology*, 7(2), pp. 76–79. Available at: https://www.researchgate.net/publication/327111767.

Kunarak, S. and Duangchan, T. (2021) ‘Vertical Handover Decision based on Hybrid Artificial Neural Networks in HetNets of 5G’, in *TENSYMP 2021 - 2021 IEEE Region 10 Symposium*. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/TENSYMP52854.2021.9550952.

Kurt, I., Ture, M. and Kurum, A.T. (2008) ‘Comparing performances of logistic regression, classification and regression tree, and neural networks for predicting coronary artery disease’, *Expert Systems with Applications*, 34(1), pp. 366–374. Available at: https://doi.org/10.1016/j.eswa.2006.09.004.

Lahby, M., Essouiri, A. and Sekkaki, A. (2019) ‘A novel modeling approach for vertical handover based on dynamic k-partite graph in heterogeneous networks’, *Digital Communications and Networks*, 5(4), pp. 297–307. Available at: https://doi.org/10.1016/j.dcan.2019.10.001.

Langsetmo, L. *et al.* (2023) ‘Advantages and Disadvantages of Random Forest Models for Prediction of Hip Fracture Risk Versus Mortality Risk in the Oldest Old’, *JBMR Plus*, 7(8). Available at: https://doi.org/10.1002/jbm4.10757.

Laricchia, F. (2023) *Smartphone ownership in the United Kingdom (UK) 2012-2023, by age*, *Statista*. Available at: statista.com/statistics/271851/smartphone-owners-in-the-united-kingdom-uk-by-age (Accessed: 7 October 2023).

Lee, C. *et al.* (2020) ‘Prediction-Based Conditional Handover for 5G mm-Wave Networks: A Deep-Learning Approach’, *IEEE Vehicular Technology Magazine*, 15(1), pp. 54–62. Available at: https://doi.org/10.1109/MVT.2019.2959065.

Lin, Y. *et al.* (2011) ‘Large-scale image classification: Fast feature extraction and SVM training’, in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, pp. 1689–1696. Available at: https://doi.org/10.1109/CVPR.2011.5995477.

Ma, Y., Chen, X. and Zhang, L. (2021) ‘Base Station handover Based on User Trajectory Prediction in 5G Networks’, in *2021 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing, Communications, Social Computing & Networking*. Available at: https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00199.

Maji, P. and Mullins, R. (2018) ‘On the reduction of computational complexity of deep convolutional neural networks’, *Entropy*, 20(4). Available at: https://doi.org/10.3390/e20040305.

Majid, S.I., Shah, S.W. and Marwat, S.N.K. (2020) ‘Applications of extreme gradient boosting for intelligent handovers from 4G to 5G (mm waves) technology with partial radio contact’, *Electronics (Switzerland)*, 9(4). Available at: https://doi.org/10.3390/electronics9040545.

Marcot, B.G. and Hanea, A.M. (2021) ‘What is an optimal value of k in k-fold cross-validation in discrete Bayesian network analysis?’, *Computational Statistics*, 36(3), pp. 2009–2031. Available at: https://doi.org/10.1007/s00180-020-00999-9.

Mehmeti, F. and Porta, T.F.L. (2021) ‘Analyzing a 5G Dataset and Modeling Metrics of Interest’, in *Proceedings - 2021 17th International Conference on Mobility, Sensing and Networking, MSN 2021*. Institute of Electrical and Electronics Engineers Inc., pp. 81–88. Available at: https://doi.org/10.1109/MSN53354.2021.00027.

Mei, L. *et al.* (2022) ‘Realtime mobile bandwidth and handoff predictions in 4G/5G networks’, *Computer Networks*, 204, p. 108736.

Morattab, A., Dziong, Z. and Sohraby, K. (2019) ‘Mode selection map-based vertical handover in D2D enabled 5G networks’, *IET Communications*, 13(14), pp. 2173–2185. Available at: https://doi.org/10.1049/iet-com.2018.6133.

NPerf.com (2023) *O2 Mobile 3G/4G/5G coverage map, United Kingdom*, *NPerf.com*. Available at: nperf.com/en/map/GB/-/2012852.O2-Mobile/signal/ (Accessed: 11 November 2023).

Nyangaresi, V.O., Rodrigues, A.J. and Abeka, S.O. (2022) ‘Machine Learning Protocol for Secure 5G Handovers’, *International Journal of Wireless Information Networks*, 29(1), pp. 14–35. Available at: https://doi.org/10.1007/s10776-021-00547-2.

Pal, M. and Mather, P.M. (2003) ‘An assessment of the effectiveness of decision tree methods for land cover classification’, *Remote Sensing of Environment*, 86(4), pp. 554–565. Available at: https://doi.org/10.1016/S0034-4257(03)00132-9.

Panda, S., Ramakrishnan, K.K. and Bhuyan, L.N. (2022) ‘Synergy: A SmartNIC Accelerated 5G Dataplane and Monitor for Mobility Prediction’, in *Proceedings - International Conference on Network Protocols, ICNP*. IEEE Computer Society. Available at: https://doi.org/10.1109/ICNP55882.2022.9940261.

Paropkari, R.A., Thantharate, A. and Beard, C. (2022) ‘Deep-Mobility: A Deep Learning Approach for an Efficient and Reliable 5G Handover’, in *2022 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2022*. Institute of Electrical and Electronics Engineers Inc., pp. 244–250. Available at: https://doi.org/10.1109/WiSPNET54241.2022.9767158.

Piran, M.J. *et al.* (2017) ‘QoE-Driven Channel Allocation and Handoff Management for Seamless Multimedia in Cognitive 5G Cellular Networks’, *IEEE Transactions on Vehicular Technology*, 66(7), pp. 6569–6585. Available at: https://doi.org/10.1109/TVT.2016.2629507.

Plachy, J., Becvar, Z. and Strinati, E.C. (2016) ‘Dynamic Resource Allocation Exploiting Mobility Prediction in Mobile Edge Computing’, in *27th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 1–6.

Pramod Kumar, P. and Sagar, K. (2019) ‘Vertical handover decision algorithm based on several specifications in heterogeneous wireless networks’, *International Journal of Innovative Technology and Exploring Engineering*, 8(9), pp. 977–982. Available at: https://doi.org/10.35940/ijitee.i8120.078919.

Prananto, B.H., Iskandar and Kurniawan, A. (2023) ‘A New Method to Improve Frequent-Handover Problem in High-Mobility Communications Using RIC and Machine Learning’, *IEEE Access*, 11, pp. 72281–72294. Available at: https://doi.org/10.1109/ACCESS.2023.3294990.

Qualcomm (2023) *Top 10 smartphone uses: New consumer report reveals why we’re at the point of no return*, *Qualcomm OnQ Blog*.

Raca, D. *et al.* (2018) ‘Beyond throughput: A 4G LTE dataset with channel and context metrics’, in *Proceedings of the 9th ACM Multimedia Systems Conference, MMSys 2018*. Association for Computing Machinery, Inc, pp. 460–465. Available at: https://doi.org/10.1145/3204949.3208123.

Ramos-Pérez, I. *et al.* (2022) ‘When is resampling beneficial for feature selection with imbalanced wide data?’, *Expert Systems with Applications*, 188. Available at: https://doi.org/10.1016/j.eswa.2021.116015.

Rish, I. (2001) ‘An empirical study of the naive Bayes classifier’, *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, 3(22), pp. 41–46.

Saverimoutou, A., Mathieu, B. and Vaton, S. (2020) ‘Web View: A Measurement Platform for Depicting Web Browsing Performance and Delivery’, *IEEE Communications Magazine*, 58(3), pp. 33–39. Available at: https://doi.org/10.1109/MCOM.001.1900178.

Shaddad, R.Q. *et al.* (2022) ‘Mobility Management for Small Cells in 5G Ultra-Dense Wireless Network’, in *2022 2nd International Conference on Emerging Smart Technologies and Applications, eSmarTA 2022*. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/eSmarTA56775.2022.9935394.

Sharma, V., You, I. and Kumar, R. (2018) ‘Resource-based mobility management for video users in 5G using catalytic computing’, *Computer Communications*, 118, pp. 120–139. Available at: https://doi.org/10.1016/j.comcom.2017.10.009.

Song, H., Fang, X. and Yan, L. (2014) ‘Handover scheme for 5G C/U plane split heterogeneous network in high-speed railway’, *IEEE Transactions on Vehicular Technology*, 63(9), pp. 4633–4646. Available at: https://doi.org/10.1109/TVT.2014.2315231.

Statista (2023) *Ownership of smartphones in the United Kingdom (UK) 2008, 2019-2022*, *Statista*. Available at: statista.com/statistics/956297/ownership-of-smartphones-uk (Accessed: 7 October 2023).

Subramani, M. and Kumaravelu, V.B. (2019) ‘A Quality-Aware Fuzzy-Logic-Based Vertical Handover Decision Algorithm for Device-to-Device Communication’, *Arabian Journal for Science and Engineering*, 44(3), pp. 2413–2425. Available at: https://doi.org/10.1007/s13369-018-3560-0.

Sutton, A. (2018) ‘5G Network Architecture, Design and Optimisation’, in *IET 5G Conference*. Available at: https://www.itu.int/dms\_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf.

Tan, X., Chen, G. and Sun, H. (2020) ‘Vertical handover algorithm based on multi-attribute and neural network in heterogeneous integrated network’, *Eurasip Journal on Wireless Communications and Networking*, 2020(1). Available at: https://doi.org/10.1186/s13638-020-01822-1.

Tu, J. V (1996) ‘Advantages and Disadvantages of Using Artificial Neural Networks versus Logistic Regression for Predicting Medical Outcomes’, *Journal of Clinical Epidemiology*, 49(11), pp. 1225–1231.

Vankayala, S.K. *et al.* (2022) ‘Deep-Learning Based Proactive Handover for 5G/6G Mobile Networks using Wireless Information’, in *2022 IEEE GLOBECOM Workshops, GC Wkshps 2022 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., pp. 461–466. Available at: https://doi.org/10.1109/GCWkshps56602.2022.10008702.

Wang, C. *et al.* (2018) ‘Deep learning-based intelligent dual connectivity for mobility management in dense network’, in *2018 IEEE Vehicular Technology Conference*, pp. 1–5.

Wei, Y. *et al.* (2023) ‘Deep Q-Networks Assisted Pre-connect Handover Management for 5G Networks’, in *IEEE Vehicular Technology Conference*. Institute of Electrical and Electronics Engineers Inc. Available at: https://doi.org/10.1109/VTC2023-Spring57618.2023.10199527.

Wigelius, H. and Väätäjä, H. (2009) ‘Dimensions of Context Affecting User Experience in Mobile Work’, in *Human-Computer Interaction - INTERACT 2009*, pp. 604–617.

Xiao, Y. and Rosdahl, J. (2002) ‘Throughput and delay limits of IEEE 802.11’, *IEEE Communications Letters*, 6(8), pp. 355–357. Available at: https://doi.org/10.1109/LCOMM.2002.802035.

Yadav, S. and Shukla, S. (2016) ‘Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification’, in *Proceedings - 6th International Advanced Computing Conference, IACC 2016*. Institute of Electrical and Electronics Engineers Inc., pp. 78–83. Available at: https://doi.org/10.1109/IACC.2016.25.

Zineb, A. Ben, Ayadi, M. and Tabbane, S. (2017) ‘QoE-based Vertical Handover Decision Management for Cognitive Networks using ANN’, in *2017 Sixth International Conference on Communications and Networking*. Hammamet, Tunisia: IEEE.

# Appendices

## Appendix 1: Validation Dataset Pre-Processing Steps

Firstly, the CSVs in the different mobility scenario dataset directories (Bus, Car, Pedestrian, Static and Train) were merged into one dataset for each mobility scenario to make processing easier, and a mobility scenario column denoting the scenario was added to each record.

Then, the mobility scenario datasets were merged into a combined dataset encapsulating all the mobility scenarios so that a number of mobility scenarios could be covered by one model. This step was repeated for each of the subsets, so there were ultimately three combined subsets; a *throughput* subset, a *status quo* subset and a *combined* subset.

Finally, each subset was filtered to only contain data for LTE (4G) and HSPA+ (3G) networks. This was done because the other network types encapsulated within the original dataset were either obsolete network technologies or too infrequently present to give accurate results.

## Appendix 2: Ethical Form

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a questionnaire

Description automatically generated

A close-up of a text

Description automatically generated

## Appendix 3: Initial Dataset Source Code

### Appendix 3.1: Pre-Processing

#### Appendix 3.1.1: Throughput Dataset Filtering.py

#Importing Pandas

import pandas as pd

#Reading CSV, with only the desired columns selected

DataFrame = pd.read\_csv("Throughput Tests - Speedtest - Active Measurements.csv", usecols=["Date", "Time", "GPS Long", "GPS Lat", "RAT Info", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. UL Avg", "Current Netw. DL Max", "Current Netw. UL Max", "UE Mode", "Scenario"])

#Printing summary of DataFrame

print(DataFrame)

#Filtering data so that only records with 5G enabled were used

UEModeFilteredData = DataFrame.loc[DataFrame["UE Mode"] == "5G-enabled"]

#Printing summary of filtered data

print(UEModeFilteredData)

#Writing filtered data to a new CSV file

UEModeFilteredData.to\_csv("Filtered Throughput Dataset.csv")

#### Appendix 3.1.2: Status Quo Throughput Dataset Filtering.py

#Importing Pandas

import pandas as pd

#Reading CSV, with only the desired columns selected

DataFrame = pd.read\_csv("Throughput Tests - Speedtest - Active Measurements.csv", usecols=["Date", "Time", "GPS Long", "GPS Lat", "RAT Info", "SS-RSRP", "SS-RSRQ", "SS-SINR", "UE Mode", "Scenario"])

#Printing summary of unfiltered data

print(DataFrame)

#Filtering data to only contain records where 5G was enabled

UEModeFilteredData = DataFrame.loc[DataFrame["UE Mode"] == "5G-enabled"]

#Printing summary of filtered data

print(UEModeFilteredData)

#Writing filtered data to a new CSV

UEModeFilteredData.to\_csv("Filtered Status Quo Throughput Dataset.csv")

#### Appendix 3.1.3: Combined Dataset Filtering.py

#Importing Pandas

import pandas as pd

#Reading CSV and selecting only the appropriate columns

DataFrame = pd.read\_csv("Throughput Tests - Speedtest - Active Measurements.csv", usecols=["Date", "Time", "GPS Long", "GPS Lat", "RAT Info", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. UL Avg", "Current Netw. DL Max", "Current Netw. UL Max", "UE Mode", "Scenario"])

#Printing a summary of the dataset

print(DataFrame)

#Filtering data so that only readings with 5G enabled are used

UEModeFilteredData = DataFrame.loc[DataFrame["UE Mode"] == "5G-enabled"]

#Printing a summary of the filtered dataset

print(UEModeFilteredData)

#Writing the filtered dataset to a CSV file

UEModeFilteredData.to\_csv("Filtered Combined Dataset.csv")

#### Appendix 3.1.4: Sorting and Filling Throughput Dataset.py

#Importing Pandas

import pandas as pd

#Declaring missing value denoter

MissingValues = ["?"]

#Reading CSV

DataFrame = pd.read\_csv("Filtered Throughput Dataset.csv", na\_values=MissingValues)

#Dropping index column

DataFrame = DataFrame.drop("Unnamed: 0", axis=1)

#Filtering the data to only display records for Date 1

Date1FilteredData = DataFrame.loc[DataFrame["Date"] == "04.01.2021"]

#Printing a summary of the Date 1 dataset

print(Date1FilteredData)

#Sorting Date 1 values by time

Date1SortedData = Date1FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date1SortedData)

#Forward-filling the Date 1 data to fill in any gaps

Date1FilledData = Date1SortedData.ffill(axis = 0)

#Printing a summary of the forward-filled Date 1 data

print(Date1FilledData)

#Writing final Date 1 data to a CSV

Date1FilledData.to\_csv("Date 1 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 2

Date2FilteredData = DataFrame.loc[DataFrame["Date"] == "05.01.2021"]

#Printing a summary of the Date 2 dataset

print(Date2FilteredData)

#Sorting Date 2 values by time

Date2SortedData = Date2FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date2SortedData)

#Forward-filling the Date 2 data to fill in any gaps

Date2FilledData = Date2SortedData.ffill(axis = 0)

#Printing a summary of the forward-filled Date 2 data

print(Date2FilledData)

#Writing final Date 2 data to a CSV

Date2FilledData.to\_csv("Date 2 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 3

Date3FilteredData = DataFrame.loc[DataFrame["Date"] == "13.12.2020"]

#Printing a summary of the Date 3 dataset

print(Date3FilteredData)

#Sorting Date 3 values by time

Date3SortedData = Date3FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date3SortedData)

#Forward-filling the Date 3 data to fill in any gaps

Date3FilledData = Date3SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date3FilledData)

#Writing final Date 3 dataset to a CSV file

Date3FilledData.to\_csv("Date 3 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 4

Date4FilteredData = DataFrame.loc[DataFrame["Date"] == "27.01.2021"]

#Printing a summary of the Date 4 dataset

print(Date4FilteredData)

#Sorting Date 4 values by time

Date4SortedData = Date4FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date4SortedData)

#Forward-filling the Date 4 data to fill in any gaps

Date4FilledData = Date4SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date4FilledData)

#Writing final Date 4 dataset to a CSV file

Date4FilledData.to\_csv("Date 4 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 5

Date5FilteredData = DataFrame.loc[DataFrame["Date"] == "07.01.2021"]

#Printing a summary of the Date 5 dataset

print(Date5FilteredData)

#Sorting Date 5 values by time

Date5SortedData = Date5FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date5SortedData)

#Forward-filling the Date 5 data to fill in any gaps

Date5FilledData = Date5SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date5FilledData)

#Writing final Date 5 dataset to a CSV file

Date5FilledData.to\_csv("Date 5 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 6

Date6FilteredData = DataFrame.loc[DataFrame["Date"] == "15.12.2020"]

#Printing a summary of the Date 6 dataset

print(Date6FilteredData)

#Sorting Date 6 values by time

Date6SortedData = Date6FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date6SortedData)

#Forward-filling the Date 6 data to fill in any gaps

Date6FilledData = Date6SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date6FilledData)

#Writing final Date 6 dataset to a CSV file

Date6FilledData.to\_csv("Date 6 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 7

Date7FilteredData = DataFrame.loc[DataFrame["Date"] == "24.01.2021"]

#Printing a summary of the Date 7 dataset

print(Date7FilteredData)

#Sorting Date 7 values by time

Date7SortedData = Date7FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date7SortedData)

#Forward-filling the Date 7 dataset to fill in any gaps

Date7FilledData = Date7SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date7FilledData)

#Writing final Date 7 dataset to a CSV file

Date7FilledData.to\_csv("Date 7 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 8

Date8FilteredData = DataFrame.loc[DataFrame["Date"] == "08.01.2021"]

#Printing a summary of the Date 8 dataset

print(Date8FilteredData)

#Sorting Date 8 values by time

Date8SortedData = Date8FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date8SortedData)

#Forward-filling the Date 8 dataset to fill in any gaps

Date8FilledData = Date8SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date8FilledData)

#Writing final Date 8 dataset to a CSV file

Date8FilledData.to\_csv("Date 8 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 9

Date9FilteredData = DataFrame.loc[DataFrame["Date"] == "16.12.2020"]

#Printing a summary of the Date 9 dataset

print(Date9FilteredData)

#Sorting Date 9 values by time

Date9SortedData = Date9FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date9SortedData)

#Forward-filling the Date 9 dataset to fill in any gaps

Date9FilledData = Date9SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date9FilledData)

#Writing final Date 9 dataset to a CSV file

Date9FilledData.to\_csv("Date 9 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 10

Date10FilteredData = DataFrame.loc[DataFrame["Date"] == "26.01.2021"]

#Printing a summary of the Date 10 dataset

print(Date10FilteredData)

#Sorting Date 10 values by time

Date10SortedData = Date10FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date10SortedData)

#Forward-filling the Date 10 dataset to fill in any gaps

Date10FilledData = Date10SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date10FilledData)

#Writing final Date 10 dataset to a CSV file

Date10FilledData.to\_csv("Date 10 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 11

Date11FilteredData = DataFrame.loc[DataFrame["Date"] == "14.01.2021"]

#Printing a summary of the Date 11 dataset

print(Date11FilteredData)

#Sorting Date 11 values by time

Date11SortedData = Date11FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date11SortedData)

#Forward-filling the Date 11 dataset to fill in any gaps

Date11FilledData = Date11SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date11FilledData)

#Writing final Date 11 dataset to a CSV file

Date11FilledData.to\_csv("Date 11 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 12

Date12FilteredData = DataFrame.loc[DataFrame["Date"] == "06.01.2021"]

#Printing a summary of the Date 12 dataset

print(Date12FilteredData)

#Sorting Date 12 values by time

Date12SortedData = Date12FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date12SortedData)

#Forward-filling the Date 12 dataset to fill in any gaps

Date12FilledData = Date12SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date12FilledData)

#Writing final Date 12 dataset to a CSV file

Date12FilledData.to\_csv("Date 12 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 13

Date13FilteredData = DataFrame.loc[DataFrame["Date"] == "17.12.2020"]

#Printing a summary of the Date 13 dataset

print(Date13FilteredData)

#Sorting Date 13 values by time

Date13SortedData = Date13FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date13SortedData)

#Forward-filling the Date 13 dataset to fill in any gaps

Date13FilledData = Date13SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date13FilledData)

#Writing final Date 13 dataset to a CSV file

Date13FilledData.to\_csv("Date 13 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 14

Date14FilteredData = DataFrame.loc[DataFrame["Date"] == "25.01.2021"]

#Printing a summary of the Date 14 dataset

print(Date14FilteredData)

#Sorting Date 14 values by time

Date14SortedData = Date14FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date14SortedData)

#Forward-filling the Date 14 dataset to fill in any gaps

Date14FilledData = Date14SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date14FilledData)

#Writing final Date 14 dataset to a CSV file

Date14FilledData.to\_csv("Date 14 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 15

Date15FilteredData = DataFrame.loc[DataFrame["Date"] == "14.12.2020"]

#Printing a summary of the Date 15 dataset

print(Date15FilteredData)

#Sorting Date 15 values by time

Date15SortedData = Date15FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date15SortedData)

#Forward-filling the Date 15 dataset to fill in any gaps

Date15FilledData = Date15SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date15FilledData)

#Writing final Date 15 dataset to a CSV file

Date15FilledData.to\_csv("Date 15 Cleaned Throughput Data.csv")

#Filtering the data to only display records for Date 16

Date16FilteredData = DataFrame.loc[DataFrame["Date"] == "11.01.2021"]

#Printing a summary of the Date 16 dataset

print(Date16FilteredData)

#Sorting Date 16 values by time

Date16SortedData = Date16FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date16SortedData)

#Forward-filling the Date 16 dataset to fill in any gaps

Date16FilledData = Date16SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date16FilledData)

#Writing final Date 16 dataset to a CSV file

Date16FilledData.to\_csv("Date 16 Cleaned Throughput Data.csv")

#Combining all 16 days' worth of data into one DataFrame

CombinedDataset = pd.concat([Date1FilledData, Date2FilledData, Date3FilledData, Date4FilledData, Date5FilledData, Date6FilledData, Date7FilledData, Date8FilledData, Date9FilledData, Date10FilledData, Date11FilledData, Date12FilledData, Date13FilledData, Date14FilledData, Date15FilledData, Date16FilledData])

#Writing final combined dataset to a CSV file

CombinedDataset.to\_csv("Final Throughput Dataset.csv")

#### Appendix 3.1.5: Sorting and Filling Status Quo Throughput Dataset.csv

#Importing Pandas

import pandas as pd

#Declaring missing value denoter

MissingValues = ["?"]

#Reading CSV

DataFrame = pd.read\_csv("Filtered Status Quo Throughput Dataset.csv", na\_values=MissingValues)

#Dropping index column

DataFrame = DataFrame.drop("Unnamed: 0", axis=1)

#Filtering dataset to only show data for Date 1

Date1FilteredData = DataFrame.loc[DataFrame["Date"] == "04.01.2021"]

#Printing a summary of the Date 1 dataset

print(Date1FilteredData)

#Sorting the Date 1 dataset by time

Date1SortedData = Date1FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date1SortedData)

#Forward-filling the Date 1 dataset to fill in any gaps

Date1FilledData = Date1SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date1FilledData)

#Writing final Date 1 dataset to a CSV file

Date1FilledData.to\_csv("Date 1 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 2

Date2FilteredData = DataFrame.loc[DataFrame["Date"] == "05.01.2021"]

#Printing a summary of the Date 2 dataset

print(Date2FilteredData)

#Sorting the Date 2 dataset by time

Date2SortedData = Date2FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date2SortedData)

#Forward-filling the Date 2 dataset to fill in any gaps

Date2FilledData = Date2SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date2FilledData)

#Writing final Date 2 dataset to a CSV file

Date2FilledData.to\_csv("Date 2 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 3

Date3FilteredData = DataFrame.loc[DataFrame["Date"] == "13.12.2020"]

#Printing a summary of the Date 3 dataset

print(Date3FilteredData)

#Sorting the Date 3 dataset by time

Date3SortedData = Date3FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date3SortedData)

#Forward-filling the Date 3 dataset to fill in any gaps

Date3FilledData = Date3SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date3FilledData)

#Writing final Date 3 dataset to a CSV file

Date3FilledData.to\_csv("Date 3 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 4

Date4FilteredData = DataFrame.loc[DataFrame["Date"] == "27.01.2021"]

#Printing a summary of the Date 4 dataset

print(Date4FilteredData)

#Sorting the Date 4 dataset by time

Date4SortedData = Date4FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date4SortedData)

#Forward-filling the Date 4 dataset to fill in any gaps

Date4FilledData = Date4SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date4FilledData)

#Writing final Date 4 dataset to a CSV file

Date4FilledData.to\_csv("Date 4 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 5

Date5FilteredData = DataFrame.loc[DataFrame["Date"] == "07.01.2021"]

#Printing a summary of the Date 5 dataset

print(Date5FilteredData)

#Sorting the Date 5 dataset by time

Date5SortedData = Date5FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date5SortedData)

#Forward-filling the Date 5 dataset to fill in any gaps

Date5FilledData = Date5SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date5FilledData)

#Writing final Date 5 dataset to a CSV file

Date5FilledData.to\_csv("Date 5 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 6

Date6FilteredData = DataFrame.loc[DataFrame["Date"] == "15.12.2020"]

#Printing a summary of the Date 6 dataset

print(Date6FilteredData)

#Sorting the Date 6 dataset by time

Date6SortedData = Date6FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date6SortedData)

#Forward-filling the Date 6 dataset to fill in any gaps

Date6FilledData = Date6SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date6FilledData)

#Writing final Date 6 dataset to a CSV file

Date6FilledData.to\_csv("Date 6 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 7

Date7FilteredData = DataFrame.loc[DataFrame["Date"] == "24.01.2021"]

#Printing a summary of the Date 7 dataset

print(Date7FilteredData)

#Sorting the Date 7 dataset by time

Date7SortedData = Date7FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date7SortedData)

#Forward-filling the Date 7 dataset to fill in any gaps

Date7FilledData = Date7SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date7FilledData)

#Writing final Date 7 dataset to a CSV file

Date7FilledData.to\_csv("Date 7 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 8

Date8FilteredData = DataFrame.loc[DataFrame["Date"] == "08.01.2021"]

#Printing a summary of the Date 8 dataset

print(Date8FilteredData)

#Sorting the Date 8 dataset by time

Date8SortedData = Date8FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date8SortedData)

#Forward-filling the Date 8 dataset to fill in any gaps

Date8FilledData = Date8SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date8FilledData)

#Writing final Date 8 dataset to a CSV file

Date8FilledData.to\_csv("Date 8 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 9

Date9FilteredData = DataFrame.loc[DataFrame["Date"] == "16.12.2020"]

#Printing a summary of the Date 9 dataset

print(Date9FilteredData)

#Sorting the Date 9 dataset by time

Date9SortedData = Date9FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date9SortedData)

#Forward-filling the Date 9 dataset to fill in any gaps

Date9FilledData = Date9SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date9FilledData)

#Writing final Date 9 dataset to a CSV file

Date9FilledData.to\_csv("Date 9 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 10

Date10FilteredData = DataFrame.loc[DataFrame["Date"] == "26.01.2021"]

#Printing a summary of the Date 10 dataset

print(Date10FilteredData)

#Sorting the Date 10 dataset by time

Date10SortedData = Date10FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date10SortedData)

#Forward-filling the Date 10 dataset to fill in any gaps

Date10FilledData = Date10SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date10FilledData)

#Writing final Date 10 dataset to a CSV file

Date10FilledData.to\_csv("Date 10 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 11

Date11FilteredData = DataFrame.loc[DataFrame["Date"] == "14.01.2021"]

#Printing a summary of the Date 11 dataset

print(Date11FilteredData)

#Sorting the Date 11 dataset by time

Date11SortedData = Date11FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date11SortedData)

#Forward-filling the Date 11 dataset to fill in any gaps

Date11FilledData = Date11SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date11FilledData)

#Writing final Date 11 dataset to a CSV file

Date11FilledData.to\_csv("Date 11 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 12

Date12FilteredData = DataFrame.loc[DataFrame["Date"] == "06.01.2021"]

#Printing a summary of the Date 12 dataset

print(Date12FilteredData)

#Sorting the Date 12 dataset by time

Date12SortedData = Date12FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date12SortedData)

#Forward-filling the Date 12 dataset to fill in any gaps

Date12FilledData = Date12SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date12FilledData)

#Writing final Date 12 dataset to a CSV file

Date12FilledData.to\_csv("Date 12 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 13

Date13FilteredData = DataFrame.loc[DataFrame["Date"] == "17.12.2020"]

#Printing a summary of the Date 13 dataset

print(Date13FilteredData)

#Sorting the Date 13 dataset by time

Date13SortedData = Date13FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date13SortedData)

#Forward-filling the Date 13 dataset to fill in any gaps

Date13FilledData = Date13SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date13FilledData)

#Writing final Date 13 dataset to a CSV file

Date13FilledData.to\_csv("Date 13 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 14

Date14FilteredData = DataFrame.loc[DataFrame["Date"] == "25.01.2021"]

#Printing a summary of the Date 14 dataset

print(Date14FilteredData)

#Sorting the Date 14 dataset by time

Date14SortedData = Date14FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date14SortedData)

#Forward-filling the Date 14 data to fill in any gaps

Date14FilledData = Date14SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date14FilledData)

#Writing final Date 14 dataset to a CSV file

Date14FilledData.to\_csv("Date 14 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 15

Date15FilteredData = DataFrame.loc[DataFrame["Date"] == "14.12.2020"]

#Printing a summary of the Date 15 dataset

print(Date15FilteredData)

#Sorting the Date 15 dataset by time

Date15SortedData = Date15FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date15SortedData)

#Forward-filling the Date 15 dataset to fill in any gaps

Date15FilledData = Date15SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date15FilledData)

#Writing final Date 15 dataset to a CSV file

Date15FilledData.to\_csv("Date 15 Cleaned Status Quo Throughput Data.csv")

#Filtering dataset to only show data for Date 16

Date16FilteredData = DataFrame.loc[DataFrame["Date"] == "11.01.2021"]

#Printing a summary of the Date 16 dataset

print(Date16FilteredData)

#Sorting the Date 16 dataset by time

Date16SortedData = Date16FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date16SortedData)

#Forward-filling the Date 16 dataset to fill in any gaps

Date16FilledData = Date16SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date16FilledData)

#Writing final Date 15 dataset to a CSV file

Date16FilledData.to\_csv("Date 16 Cleaned Status Quo Throughput Data.csv")

#Concatenating all datasets together

CombinedDataset = pd.concat([Date1FilledData, Date2FilledData, Date3FilledData, Date4FilledData, Date5FilledData, Date6FilledData, Date7FilledData, Date8FilledData, Date9FilledData, Date10FilledData, Date11FilledData, Date12FilledData, Date13FilledData, Date14FilledData, Date15FilledData, Date16FilledData])

#Writing final combined dataset to a CSV file

CombinedDataset.to\_csv("Final Status Quo Throughput Dataset.csv")

#### Appendix 3.1.6: Sorting and Filling Combined Dataset.py

#Importing Pandas

import pandas as pd

#Declaring missing value denoter

MissingValues = ["?"]

#Reading CSV

DataFrame = pd.read\_csv("Filtered Combined Dataset.csv", na\_values=MissingValues)

#Dropping index column

DataFrame = DataFrame.drop("Unnamed: 0", axis=1)

#Filtering the dataset to only show data for Date 1

Date1FilteredData = DataFrame.loc[DataFrame["Date"] == "04.01.2021"]

#Printing a summary of the Date 1 dataset

print(Date1FilteredData)

#Sorting the Date 1 dataset by time

Date1SortedData = Date1FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date1SortedData)

#Forward-filling the Date 1 dataset to fill in any gaps

Date1FilledData = Date1SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date1FilledData)

#Writing final Date 1 dataset to a CSV file

Date1FilledData.to\_csv("Date 1 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 2

Date2FilteredData = DataFrame.loc[DataFrame["Date"] == "05.01.2021"]

#Printing a summary of the Date 2 dataset

print(Date2FilteredData)

#Sorting the Date 2 dataset by time

Date2SortedData = Date2FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date2SortedData)

#Forward-filling the Date 2 dataset to fill in any gaps

Date2FilledData = Date2SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date2FilledData)

#Writing final Date 2 dataset to a CSV file

Date2FilledData.to\_csv("Date 2 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 3

Date3FilteredData = DataFrame.loc[DataFrame["Date"] == "13.12.2020"]

#Printing a summary of the Date 3 dataset

print(Date3FilteredData)

#Sorting the Date 3 dataset by time

Date3SortedData = Date3FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date3SortedData)

#Forward-filling the Date 3 dataset to fill in any gaps

Date3FilledData = Date3SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date3FilledData)

#Writing final Date 3 dataset to a CSV file

Date3FilledData.to\_csv("Date 3 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 4

Date4FilteredData = DataFrame.loc[DataFrame["Date"] == "27.01.2021"]

#Printing a summary of the Date 4 dataset

print(Date4FilteredData)

#Sorting the Date 4 dataset by time

Date4SortedData = Date4FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date4SortedData)

#Forward-filling the Date 4 dataset to fill in any gaps

Date4FilledData = Date4SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date4FilledData)

#Writing final Date 4 dataset to a CSV file

Date4FilledData.to\_csv("Date 4 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 5

Date5FilteredData = DataFrame.loc[DataFrame["Date"] == "07.01.2021"]

#Printing a summary of the Date 5 dataset

print(Date5FilteredData)

#Sorting the Date 5 dataset by time

Date5SortedData = Date5FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date5SortedData)

#Forward-filling the Date 5 dataset to fill in any gaps

Date5FilledData = Date5SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date5FilledData)

#Writing final Date 5 dataset to a CSV file

Date5FilledData.to\_csv("Date 5 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 6

Date6FilteredData = DataFrame.loc[DataFrame["Date"] == "15.12.2020"]

#Printing a summary of the Date 6 dataset

print(Date6FilteredData)

#Sorting the Date 6 dataset by time

Date6SortedData = Date6FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date6SortedData)

#Forward-filling the Date 6 dataset to fill in any gaps

Date6FilledData = Date6SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date6FilledData)

#Writing final Date 6 dataset to a CSV file

Date6FilledData.to\_csv("Date 6 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 7

Date7FilteredData = DataFrame.loc[DataFrame["Date"] == "24.01.2021"]

#Printing a summary of the Date 7 dataset

print(Date7FilteredData)

#Sorting the Date 7 dataset by time

Date7SortedData = Date7FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date7SortedData)

#Forward-filling the Date 7 dataset to fill in any gaps

Date7FilledData = Date7SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date7FilledData)

#Writing final Date 7 dataset to a CSV file

Date7FilledData.to\_csv("Date 7 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 8

Date8FilteredData = DataFrame.loc[DataFrame["Date"] == "08.01.2021"]

#Printing a summary of the Date 8 dataset

print(Date8FilteredData)

#Sorting the Date 8 dataset by time

Date8SortedData = Date8FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date8SortedData)

#Forward-filling the Date 8 dataset to fill in any gaps

Date8FilledData = Date8SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date8FilledData)

#Writing final Date 8 dataset to a CSV file

Date8FilledData.to\_csv("Date 8 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 9

Date9FilteredData = DataFrame.loc[DataFrame["Date"] == "16.12.2020"]

#Printing a summary of the Date 9 dataset

print(Date9FilteredData)

#Sorting the Date 9 dataset by time

Date9SortedData = Date9FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date9SortedData)

#Forward-filling the Date 9 dataset to fill in any gaps

Date9FilledData = Date9SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date9FilledData)

#Writing final Date 9 dataset to a CSV file

Date9FilledData.to\_csv("Date 9 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 10

Date10FilteredData = DataFrame.loc[DataFrame["Date"] == "26.01.2021"]

#Printing a summary of the Date 10 dataset

print(Date10FilteredData)

#Sorting the Date 10 dataset by time

Date10SortedData = Date10FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date10SortedData)

#Forward-filling the Date 10 dataset to fill in any gaps

Date10FilledData = Date10SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date10FilledData)

#Writing final Date 10 dataset to a CSV file

Date10FilledData.to\_csv("Date 10 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 11

Date11FilteredData = DataFrame.loc[DataFrame["Date"] == "14.01.2021"]

#Printing a summary of the Date 11 dataset

print(Date11FilteredData)

#Sorting the Date 11 dataset by time

Date11SortedData = Date11FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date11SortedData)

#Forward-filling the Date 11 dataset to fill in any gaps

Date11FilledData = Date11SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date11FilledData)

#Writing final Date 11 dataset to a CSV file

Date11FilledData.to\_csv("Date 11 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 12

Date12FilteredData = DataFrame.loc[DataFrame["Date"] == "06.01.2021"]

#Printing a summary of the Date 12 dataset

print(Date12FilteredData)

#Sorting the Date 12 dataset by time

Date12SortedData = Date12FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date12SortedData)

#Forward-filling the Date 12 dataset to fill in any gaps

Date12FilledData = Date12SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date12FilledData)

#Writing final Date 12 dataset to a CSV file

Date12FilledData.to\_csv("Date 12 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 13

Date13FilteredData = DataFrame.loc[DataFrame["Date"] == "17.12.2020"]

#Printing a summary of the Date 13 dataset

print(Date13FilteredData)

#Sorting the Date 13 dataset by time

Date13SortedData = Date13FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date13SortedData)

#Forward-filling the Date 13 dataset to fill in any gaps

Date13FilledData = Date13SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date13FilledData)

#Writing final Date 13 dataset to a CSV file

Date13FilledData.to\_csv("Date 13 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 14

Date14FilteredData = DataFrame.loc[DataFrame["Date"] == "25.01.2021"]

#Printing a summary of the Date 14 dataset

print(Date14FilteredData)

#Sorting the Date 14 dataset by time

Date14SortedData = Date14FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date14SortedData)

#Forward-filling the Date 14 dataset to fill in any gaps

Date14FilledData = Date14SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date14FilledData)

#Writing final Date 14 dataset to a CSV file

Date14FilledData.to\_csv("Date 14 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 15

Date15FilteredData = DataFrame.loc[DataFrame["Date"] == "14.12.2020"]

#Printing a summary of the Date 15 dataset

print(Date15FilteredData)

#Sorting the Date 15 dataset by time

Date15SortedData = Date15FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date15SortedData)

#Forward-filling the Date 15 dataset to fill in any gaps

Date15FilledData = Date15SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date15FilledData)

#Writing final Date 15 dataset to a CSV file

Date15FilledData.to\_csv("Date 15 Cleaned Combined Data.csv")

#Filtering the dataset to only show data for Date 16

Date16FilteredData = DataFrame.loc[DataFrame["Date"] == "11.01.2021"]

#Printing a summary of the Date 16 dataset

print(Date16FilteredData)

#Sorting the Date 16 dataset by time

Date16SortedData = Date16FilteredData.sort\_values(by="Time", ascending=True)

#Printing a summary of the sorted dataset

print(Date16SortedData)

#Forward-filling the Date 16 dataset to fill in any gaps

Date16FilledData = Date16SortedData.ffill(axis = 0)

#Printing a summary of the filled dataset

print(Date16FilledData)

#Writing final Date 16 dataset to a CSV file

Date16FilledData.to\_csv("Date 16 Cleaned Combined Data.csv")

#Concatenating all datasets together

CombinedDataset = pd.concat([Date1FilledData, Date2FilledData, Date3FilledData, Date4FilledData, Date5FilledData, Date6FilledData, Date7FilledData, Date8FilledData, Date9FilledData, Date10FilledData, Date11FilledData, Date12FilledData, Date13FilledData, Date14FilledData, Date15FilledData, Date16FilledData])

#Writing final combined dataset to a CSV file

CombinedDataset.to\_csv("Final Combined Dataset.csv")

### Appendix 3.2: Dataset Visualisation

#### Appendix 3.2.1: Amount of Time Connected to 5G vs 4G (Throughput Subset).py

#Importing libraries

import pandas as pd

import matplotlib.pyplot as plt

#Reading CSV

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Grouping the DataFrame based on the RAT connected to

GroupedDataFrame = DataFrame.groupby(["RAT Info"]).count()

#Printing a summary of the grouped dataset

print(GroupedDataFrame)

#Creating a pie chart showing the proportion of records connected to each RAT

plt.pie(GroupedDataFrame["Unnamed: 0"], labels=["5G", "4G"], autopct='%1.0f%%')

#Showing the pie chart

plt.show()

#Dropping records with any missing data

CleanedDataFrame = DataFrame.dropna()

#Printing a summary of the cleaned dataset

print(CleanedDataFrame)

#Grouping the cleaned dataset based on the RAT connected to

GroupedCleanDataFrame = CleanedDataFrame.groupby(["RAT Info"]).count()

#Printing a summary of the grouped dataset

print(GroupedCleanDataFrame)

#Creating a pie chart showing the proportion of records connected to each RAT

plt.pie(GroupedCleanDataFrame["Unnamed: 0"], labels=["5G", "4G"], autopct='%1.0f%%')

#Showing the pie chart

plt.show()

#### Appendix 3.2.2: Amount of Time Connected to 5G vs 4G (Status Quo Subset).py

#Importing libraries

import pandas as pd

import matplotlib.pyplot as plt

#Reading CSV

DataFrame = pd.read\_csv("Final Status Quo Throughput Dataset.csv")

#Grouping the DataFrame based on the RAT connected to

GroupedDataFrame = DataFrame.groupby(["RAT Info"]).count()

#Printing a summary of the grouped dataset

print(GroupedDataFrame)

#Creating a pie chart showing the proportion of records connected to each RAT

plt.pie(GroupedDataFrame["Unnamed: 0"], labels=["5G", "4G"], autopct='%1.0f%%')

#Showing the pie chart

plt.show()

#Dropping records with any missing data

CleanedDataFrame = DataFrame.dropna()

#Printing a summary of the cleaned dataset

print(CleanedDataFrame)

#Grouping the cleaned dataset based on the RAT connected to

GroupedCleanDataFrame = CleanedDataFrame.groupby(["RAT Info"]).count()

#Printing a summary of the grouped dataset

print(GroupedCleanDataFrame)

#Creating a pie chart showing the proportion of records connected to each RAT

plt.pie(GroupedCleanDataFrame["Unnamed: 0"], labels=["5G", "4G"], autopct='%1.0f%%')

#Showing the pie chart

plt.show()

#### Appendix 3.2.3: Amount of Time Connected to 5G vs 4G (Combined Subset).py

#Importing libraries

import pandas as pd

import matplotlib.pyplot as plt

#Reading CSV

DataFrame = pd.read\_csv("Final Combined Dataset.csv")

#Grouping the DataFrame based on the RAT connected to

GroupedDataFrame = DataFrame.groupby(["RAT Info"]).count()

#Printing a summary of the grouped dataset

print(GroupedDataFrame)

#Creating a pie chart showing the proportion of records connected to each RAT

plt.pie(GroupedDataFrame["Unnamed: 0"], labels=["5G", "4G"], autopct='%1.0f%%')

#Showing the pie chart

plt.show()

#Dropping records with any missing data

CleanedDataFrame = DataFrame.dropna()

#Printing a summary of the cleaned dataset

print(CleanedDataFrame)

#Grouping the cleaned dataset based on the RAT connected to

GroupedCleanDataFrame = CleanedDataFrame.groupby(["RAT Info"]).count()

#Printing a summary of the grouped dataset

print(GroupedCleanDataFrame)

#Creating a pie chart showing the proportion of records connected to each RAT

plt.pie(GroupedCleanDataFrame["Unnamed: 0"], labels=["5G", "4G"], autopct='%1.0f%%')

#Showing the pie chart

plt.show()

### Appendix 3.3: AI Models

#### Appendix 3.3.1: Uniform Dummy Classifier (Status Quo Variables).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.dummy import DummyClassifier

from sklearn.metrics import \*

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Status Quo Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating dummy classification model

Dummy = DummyClassifier(strategy="uniform")

#Fitting the dummy classifier to the training data

Dummy.fit(XTrain, YTrain)

#Making predictions on the test dataset using the dummy model

YPrediction = Dummy.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

PrecisionArray = [FiveGPrecision, FourGPrecision]

RecallArray = [FiveGRecall, FourGRecall]

Precision = np.mean(PrecisionArray)

Recall = np.mean(RecallArray)

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("Precision:",Precision)

print("Recall:",Recall)

#Notifying the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation")

#Creating DummyClassifier for K-fold cross validation

KFoldDummy = DummyClassifier(strategy="uniform")

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldDummy, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.2: Most Frequent Dummy Classifier (Status Quo Variables)

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.dummy import DummyClassifier

from sklearn.metrics import \*

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Status Quo Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating dummy classification model

Dummy = DummyClassifier(strategy="most\_frequent")

#Fitting the dummy classifier to the training data

Dummy.fit(XTrain, YTrain)

#Making predictions on the test dataset using the dummy model

YPrediction = Dummy.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

PrecisionArray = [FiveGPrecision, FourGPrecision]

RecallArray = [FiveGRecall, FourGRecall]

Precision = np.mean(PrecisionArray)

Recall = np.mean(RecallArray)

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("Precision:",Precision)

print("Recall:",Recall)

#Notifying the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation")

#Creating DummyClassifier for K-fold cross validation

KFoldDummy = DummyClassifier(strategy="most\_frequent")

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldDummy, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.3: Naïve Bayes Classifier (Status Quo Variables).py

#Importing libraries

import numpy as np

import pandas as pd

from sklearn.preprocessing import RobustScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Status Quo Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

#Converting Scenario into a Category type and encoding the values as integers

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Denoting the columns of the dataset

Columns = XTrain.columns

#Declaring a scaler to scale the data for input into a Naive-Bayes Classifier

Scaler = RobustScaler()

#Fitting the scaler to the training dataset and transforming it

XTrain = Scaler.fit\_transform(XTrain)

#Transforming the testing dataset

XTest = Scaler.transform(XTest)

#Creating a Pandas DataFrame out of the scaled training data

XTrain = pd.DataFrame(XTrain, columns=[Columns])

#Creating a Pandas DataFrame out of the scaled testing data

XTest = pd.DataFrame(XTest, columns=[Columns])

#Declaring a Naive-Bayes Classifier

NaiveBayesClassifier = GaussianNB()

#Fitting the Naive-Bayes Classifier to the training and testing datasets

NaiveBayesClassifier.fit(XTrain, YTrain)

#Making predictions on the testing data using the Naive-Bayes Classifier

YPrediction = NaiveBayesClassifier.predict(XTest)

#Calculating the accuracy of the model's predictions

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting the model's prediction accuracy

print("Accuracy on test dataset:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model and scaler to separate files

pickle.dump(NaiveBayesClassifier, open("NaiveBayesClassifierStatusQuo.sav", "wb"))

pickle.dump(Scaler, open("RobustScalerStatusQuo.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Denoting the columns of the dataset for K-fold cross validation

KFoldColumns = X.columns

#Declaring a scaler to scale the data for input into a Naive-Bayes Classifier for K-fold cross validation

KFoldScaler = RobustScaler()

#Fitting the scaler to the training dataset and transforming it for K-fold cross validation

X = KFoldScaler.fit\_transform(X)

#Creating a Pandas DataFrame out of the scaled training data for K-fold cross validation

X = pd.DataFrame(X, columns=[KFoldColumns])

#Declaring a Naive-Bayes Classifier for K-fold cross validation

KFoldNaiveBayesClassifier = GaussianNB()

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldNaiveBayesClassifier, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.4: Neural Network (Status Quo Variables).py

#Importing libraries

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Activation, Dense, BatchNormalization, Dropout

from tensorflow.keras import optimizers

from tensorflow.keras.callbacks import EarlyStopping

from scikeras.wrappers import KerasClassifier

from sklearn.preprocessing import RobustScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn import metrics

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Status Quo Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

#Converting RAT Info into category codes

DataFrame["RAT Info"] = DataFrame["RAT Info"].astype("category")

DataFrame["RAT Info"] = DataFrame["RAT Info"].cat.codes

#Converting scenarios into category codes

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining used dataset categories

UsedData = DataFrame[["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category", "RAT Info"]]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing/validation data

Train, TempTest = train\_test\_split(UsedData, test\_size=0.2, random\_state=100)

#Dividing the testing/validation data into 50% testing data and 50% validation data

Test, Validation = train\_test\_split(TempTest, test\_size=0.5, random\_state=100)

#Removing the RAT Info column from the train, test and validation datasets

TrainLabels = Train.pop("RAT Info")

TestLabels = Test.pop("RAT Info")

ValidationLabels = Validation.pop("RAT Info")

#Declaring MinMaxScaler to normalise the data

Scaler = RobustScaler()

#Normalising train, test and validation datasets

NormalisedTrainData = Scaler.fit\_transform(Train)

NormalisedTestData = Scaler.transform(Test)

NormalisedValidationData = Scaler.transform(Validation)

#Declaring neural network

NeuralNetwork = Sequential()

#Adding a dense input layer, three dense hidden layers with increasing numbers of neurons for learning, and a dense output layer

NeuralNetwork.add(Dense(32, input\_shape=(NormalisedTrainData.shape[1],)))

NeuralNetwork.add(Dense(32, activation="tanh"))

NeuralNetwork.add(Dense(64, activation="tanh"))

NeuralNetwork.add(Dense(128, activation="tanh"))

NeuralNetwork.add(Dense(1, activation="sigmoid"))

#Setting the hyperparameters of the model

LearningRate = 0.01

Optimiser = optimizers.SGD(LearningRate)

NeuralNetwork.compile(loss="binary\_crossentropy", optimizer=Optimiser, metrics=["acc"])

BatchSize = 16

#Fitting the neural network to the training data

NeuralNetwork.fit(NormalisedTrainData, TrainLabels, batch\_size=BatchSize, epochs=100, verbose=2, shuffle=True, validation\_data = (NormalisedValidationData, ValidationLabels))

#Evaluating the neural network on the testing data

NeuralNetwork.evaluate(NormalisedTestData, TestLabels, verbose=2)

#Making predictions on the testing data

Prediction = NeuralNetwork.predict(NormalisedTestData, batch\_size=BatchSize, verbose=2)

PredictionBoolean = np.round(Prediction, 0)

#Printing classification report to summarise the efficacy of the model

print(metrics.classification\_report(TestLabels, PredictionBoolean))

#Saving model and RobustScaler to separate files

NeuralNetwork.save("NeuralNetworkStatusQuoVariables.h5")

pickle.dump(Scaler, open("NeuralNetworkScalerStatusQuoVariables.sav", "wb"))

#Notifying the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation")

#Subroutine to create neural network for K-fold cross validation

def CreateModel():

#Declaring neural network

KFoldNeuralNetwork = Sequential()

#Adding a dense input layer, three dense hidden layers with increasing numbers of neurons for learning, and a dense output layer

KFoldNeuralNetwork.add(Dense(32, input\_shape=(NormalisedData.shape[1],)))

KFoldNeuralNetwork.add(Dense(32, activation="tanh"))

KFoldNeuralNetwork.add(Dense(64, activation="tanh"))

KFoldNeuralNetwork.add(Dense(128, activation="tanh"))

KFoldNeuralNetwork.add(Dense(1, activation="sigmoid"))

#Setting the hyperparameters of the model

LearningRate = 0.01

Optimiser = optimizers.SGD(LearningRate)

KFoldNeuralNetwork.compile(loss="binary\_crossentropy", optimizer=Optimiser, metrics=["acc"])

BatchSize = 16

#Returning the neural network at the end of the subroutine

return KFoldNeuralNetwork

#Outlining X and Y for K-fold cross validation

X = DataFrame[["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category", "RAT Info"]]

Y = X.pop("RAT Info")

#Declaring RobustScaler to normalise the data for K-fold cross validation

KFoldScaler = RobustScaler()

#Normalising data for K-fold cross validation

NormalisedData = KFoldScaler.fit\_transform(X)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Creating KerasClassifier wrapper for neural network for K-fold cross validation, performing K-fold cross validation and showing the results

NeuralNetworkSKLearn = KerasClassifier(model=CreateModel, epochs=100, batch\_size=16, verbose=2)

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

Results = cross\_validate(NeuralNetworkSKLearn, NormalisedData, Y, scoring=Metrics, cv=Validator)

print(Results)

print("Accuracy:",np.mean(Results["test\_accuracy"]))

print("Precision:",np.mean(Results["test\_precision"]))

print("Recall:",np.mean(Results["test\_recall"]))

#### Appendix 3.3.5: Random Forest Classifier (Status Quo Variables).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Status Quo Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestStatusQuo.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.6: Random Forest Classifier (DL and UL Throughput).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.7: Random Forest Classifier (Combined).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Combined Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestCombined.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.8: Random Forest Classifier (DL Only).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Mean Netw. DL", "Current Netw. DL Avg", "Current Netw. DL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestDLOnly.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.9: Random Forest Classifier (UL Only).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. UL", "Mean Netw. UL", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestULOnly.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.10: Random Forest with 50 Trees.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=50, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=50, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.11: Random Forest with 25 Trees.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=25, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=25, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.12: Random Forest with 10 Trees.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=10, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=10, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.13: Random Forest with 5 Trees.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.14: Random Forest with 1 Tree.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=1, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=1, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.15: Random Forest with Max Depth of 20.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=20)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=20)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.16: Random Forest with Max Depth of 15.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=15)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=15)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.17: Random Forest with Max Depth of 10.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=10)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=10)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.18: Random Forest with Max Depth of 5.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=5)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=5)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 3.3.19: Random Forest with Max Depth of 1.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Final Throughput Dataset.csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"]]

Y = DataFrame["RAT Info"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=1)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

FiveGPrecision = precision\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGRecall = recall\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FiveGFOneScore = f1\_score(YTest, YPrediction, pos\_label="5G EN-DC")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("5G Precision:",FiveGPrecision)

print("5G Recall:",FiveGRecall)

print("5G F1 Score:",FiveGFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestDLAndUL.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=1)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

### Appendix 3.4: Model GUIs

#### Appendix 3.4.1: Naïve Bayes Classifier GUI (Status Quo Variables).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("NaiveBayesClassifierStatusQuo.sav","rb"))

#Loading pre-trained RobustScaler

Scaler = pickle.load(open("RobustScalerStatusQuo.sav", "rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

RSRPValue = float(RSRP.get())

RSRQValue = float(RSRQ.get())

SINRValue = float(SINR.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Static is 0, Driving is 1 and Walking is 2

if ScenarioString == "Static (Inside)":

ScenarioValue = 0

elif ScenarioString == "Driving (Outside)":

ScenarioValue = 1

elif ScenarioString == "Walking (Outside)":

ScenarioValue = 2

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, RSRPValue, RSRQValue, SINRValue, ScenarioValue]], columns=["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category"])

#Rescaling inputted values using the scaler

Data = Scaler.transform(Data)

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "5G EN-DC":

CorrectRAT = "5G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=8, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the RSRP and outputting it onto the user interface

RSRPLabel = Label(Interface, text="What is the RSRP?:")

RSRPLabel.grid(row=3, column=0)

#Declaring entry box for the user to input RSRP into and outputting it onto the user interface

RSRP = Entry(Interface, width=30)

RSRP.grid(row=3, column=1)

#Declaring label asking the user for the RSRQ and outputting it onto the user interface

RSRQLabel = Label(Interface, text="What is the RSRQ?:")

RSRQLabel.grid(row=4, column=0)

#Declaring entry box for the user to input RSRQ into and outputting it onto the user interface

RSRQ = Entry(Interface, width=30)

RSRQ.grid(row=4, column=1)

#Declaring label asking the user for the SINR and outputting it onto the user interface

SINRLabel = Label(Interface, text="What is the SINR?:")

SINRLabel.grid(row=5, column=0)

#Declaring entry box for the user to input SINR into and outputting it onto the user interface

SINR = Entry(Interface, width=30)

SINR.grid(row=5, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=6, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Static (Inside)", "Walking (Outside)", "Driving (Outside)")

Scenario.grid(row=6, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=7, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 3.4.2: Neural Network GUI (Status Quo Variables).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

import numpy as np

from tensorflow.keras.models import load\_model

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = load\_model("NeuralNetworkStatusQuoVariables.h5")

#Loading pre-trained RobustScaler

Scaler = pickle.load(open("NeuralNetworkScalerStatusQuoVariables.sav", "rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

RSRPValue = float(RSRP.get())

RSRQValue = float(RSRQ.get())

SINRValue = float(SINR.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Static is 0, Driving is 1 and Walking is 2

if ScenarioString == "Static (Inside)":

ScenarioValue = 0

elif ScenarioString == "Driving (Outside)":

ScenarioValue = 1

elif ScenarioString == "Walking (Outside)":

ScenarioValue = 2

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, RSRPValue, RSRQValue, SINRValue, ScenarioValue]], columns=["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category"])

#Rescaling inputted values using the scaler

Data = Scaler.transform(Data)

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

PredictionBoolean = np.round(Prediction, 0)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if PredictionBoolean == 0:

CorrectRAT = "5G"

elif PredictionBoolean == 1:

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=8, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the RSRP and outputting it onto the user interface

RSRPLabel = Label(Interface, text="What is the RSRP?:")

RSRPLabel.grid(row=3, column=0)

#Declaring entry box for the user to input RSRP into and outputting it onto the user interface

RSRP = Entry(Interface, width=30)

RSRP.grid(row=3, column=1)

#Declaring label asking the user for the RSRQ and outputting it onto the user interface

RSRQLabel = Label(Interface, text="What is the RSRQ?:")

RSRQLabel.grid(row=4, column=0)

#Declaring entry box for the user to input RSRQ into and outputting it onto the user interface

RSRQ = Entry(Interface, width=30)

RSRQ.grid(row=4, column=1)

#Declaring label asking the user for the SINR and outputting it onto the user interface

SINRLabel = Label(Interface, text="What is the SINR?:")

SINRLabel.grid(row=5, column=0)

#Declaring entry box for the user to input SINR into and outputting it onto the user interface

SINR = Entry(Interface, width=30)

SINR.grid(row=5, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=6, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Static (Inside)", "Walking (Outside)", "Driving (Outside)")

Scenario.grid(row=6, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=7, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 3.4.3: Random Forest GUI (Status Quo Variables).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestStatusQuo.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

RSRPValue = float(RSRP.get())

RSRQValue = float(RSRQ.get())

SINRValue = float(SINR.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Static is 0, Driving is 1 and Walking is 2

if ScenarioString == "Static (Inside)":

ScenarioValue = 0

elif ScenarioString == "Driving (Outside)":

ScenarioValue = 1

elif ScenarioString == "Walking (Outside)":

ScenarioValue = 2

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, RSRPValue, RSRQValue, SINRValue, ScenarioValue]], columns=["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "5G EN-DC":

CorrectRAT = "5G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=8, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the RSRP and outputting it onto the user interface

RSRPLabel = Label(Interface, text="What is the RSRP?:")

RSRPLabel.grid(row=3, column=0)

#Declaring entry box for the user to input RSRP into and outputting it onto the user interface

RSRP = Entry(Interface, width=30)

RSRP.grid(row=3, column=1)

#Declaring label asking the user for the RSRQ and outputting it onto the user interface

RSRQLabel = Label(Interface, text="What is the RSRQ?:")

RSRQLabel.grid(row=4, column=0)

#Declaring entry box for the user to input RSRQ into and outputting it onto the user interface

RSRQ = Entry(Interface, width=30)

RSRQ.grid(row=4, column=1)

#Declaring label asking the user for the SINR and outputting it onto the user interface

SINRLabel = Label(Interface, text="What is the SINR?:")

SINRLabel.grid(row=5, column=0)

#Declaring entry box for the user to input SINR into and outputting it onto the user interface

SINR = Entry(Interface, width=30)

SINR.grid(row=5, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=6, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Static (Inside)", "Walking (Outside)", "Driving (Outside)")

Scenario.grid(row=6, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=7, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 3.4.4: Random Forest GUI (DL and UL Throughput).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestDLAndUL.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

CurrentDLValue = float(CurrentDownlink.get())

CurrentULValue = float(CurrentUplink.get())

MeanDLValue = float(MeanDownlink.get())

MeanULValue = float(MeanUplink.get())

CurrentDLAvgValue = float(CurrentDownlinkAvg.get())

CurrentDLMaxValue = float(CurrentDownlinkMax.get())

CurrentULAvgValue = float(CurrentUplinkAvg.get())

CurrentULMaxValue = float(CurrentUplinkMax.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Static is 0, Driving is 1 and Walking is 2

if ScenarioString == "Static (Inside)":

ScenarioValue = 0

elif ScenarioString == "Driving (Outside)":

ScenarioValue = 1

elif ScenarioString == "Walking (Outside)":

ScenarioValue = 2

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, CurrentDLValue, CurrentULValue, MeanDLValue, MeanULValue, CurrentDLAvgValue, CurrentDLMaxValue, CurrentULAvgValue, CurrentULMaxValue, ScenarioValue]], columns=["GPS Long", "GPS Lat", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "5G EN-DC":

CorrectRAT = "5G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=13, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the current downlink throughput and outputting it onto the user interface

CurrentDownlinkLabel = Label(Interface, text="What is the current downlink throughput?:")

CurrentDownlinkLabel.grid(row=3, column=0)

#Declaring entry box for the user to input current downlink throughput into and outputting it onto the user interface

CurrentDownlink = Entry(Interface, width=30)

CurrentDownlink.grid(row=3, column=1)

#Declaring label asking the user for the current uplink throughput and outputting it onto the user interface

CurrentUplinkLabel = Label(Interface, text="What is the current uplink throughput?:")

CurrentUplinkLabel.grid(row=4, column=0)

#Declaring entry box for the user to input current uplink throughput into and outputting it onto the user interface

CurrentUplink = Entry(Interface, width=30)

CurrentUplink.grid(row=4, column=1)

#Declaring label asking the user for the mean downlink throughput and outputting it onto the user interface

MeanDownlinkLabel = Label(Interface, text="What is the mean downlink throughput?:")

MeanDownlinkLabel.grid(row=5, column=0)

#Declaring entry box for the user to input mean downlink throughput into and outputting it onto the user interface

MeanDownlink = Entry(Interface, width=30)

MeanDownlink.grid(row=5, column=1)

#Declaring label asking the user for the mean uplink throughput and outputting it onto the user interface

MeanUplinkLabel = Label(Interface, text="What is the mean uplink throughput?:")

MeanUplinkLabel.grid(row=6, column=0)

#Declaring entry box for the user to input mean uplink throughput into and outputting it onto the user interface

MeanUplink = Entry(Interface, width=30)

MeanUplink.grid(row=6, column=1)

#Declaring label asking the user for the current downlink throughput average and outputting it onto the user interface

CurrentDownlinkAvgLabel = Label(Interface, text="What is the current downlink throughput average?:")

CurrentDownlinkAvgLabel.grid(row=7, column=0)

#Declaring entry box for the user to input current downlink throughput average into and outputting it onto the user interface

CurrentDownlinkAvg = Entry(Interface, width=30)

CurrentDownlinkAvg.grid(row=7, column=1)

#Declaring label asking the user for the current downlink throughput maximum and outputting it onto the user interface

CurrentDownlinkMaxLabel = Label(Interface, text="What is the current downlink throughput maximum?:")

CurrentDownlinkMaxLabel.grid(row=8, column=0)

#Declaring entry box for the user to input current downlink throughput maximum into and outputting it onto the user interface

CurrentDownlinkMax = Entry(Interface, width=30)

CurrentDownlinkMax.grid(row=8, column=1)

#Declaring label asking the user for the current uplink throughput average and outputting it onto the user interface

CurrentUplinkAvgLabel = Label(Interface, text="What is the current uplink throughput average?:")

CurrentUplinkAvgLabel.grid(row=9, column=0)

#Declaring entry box for the user to input current uplink throughput average into and outputting it onto the user interface

CurrentUplinkAvg = Entry(Interface, width=30)

CurrentUplinkAvg.grid(row=9, column=1)

#Declaring label asking the user for the current uplink throughput maximum and outputting it onto the user interface

CurrentUplinkMaxLabel = Label(Interface, text="What is the current uplink throughput maximum?:")

CurrentUplinkMaxLabel.grid(row=10, column=0)

#Declaring entry box for the user to input current uplink throughput maximum into and outputting it onto the user interface

CurrentUplinkMax = Entry(Interface, width=30)

CurrentUplinkMax.grid(row=10, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=11, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Static (Inside)", "Walking (Outside)", "Driving (Outside)")

Scenario.grid(row=11, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=12, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 3.4.5: Random Forest GUI (Combined).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestCombined.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

RSRPValue = float(RSRP.get())

RSRQValue = float(RSRQ.get())

SINRValue = float(SINR.get())

CurrentDLValue = float(CurrentDownlink.get())

CurrentULValue = float(CurrentUplink.get())

MeanDLValue = float(MeanDownlink.get())

MeanULValue = float(MeanUplink.get())

CurrentDLAvgValue = float(CurrentDownlinkAvg.get())

CurrentDLMaxValue = float(CurrentDownlinkMax.get())

CurrentULAvgValue = float(CurrentUplinkAvg.get())

CurrentULMaxValue = float(CurrentUplinkMax.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Static is 0, Driving is 1 and Walking is 2

if ScenarioString == "Static (Inside)":

ScenarioValue = 0

elif ScenarioString == "Driving (Outside)":

ScenarioValue = 1

elif ScenarioString == "Walking (Outside)":

ScenarioValue = 2

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, RSRPValue, RSRQValue, SINRValue, CurrentDLValue, CurrentULValue, MeanDLValue, MeanULValue, CurrentDLAvgValue, CurrentDLMaxValue, CurrentULAvgValue, CurrentULMaxValue, ScenarioValue]], columns=["GPS Long", "GPS Lat", "SS-RSRP", "SS-RSRQ", "SS-SINR", "Current Netw. DL", "Current Netw. UL", "Mean Netw. DL", "Mean Netw. UL", "Current Netw. DL Avg", "Current Netw. DL Max", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "5G EN-DC":

CorrectRAT = "5G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=16, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the RSRP and outputting it onto the user interface

RSRPLabel = Label(Interface, text="What is the RSRP?:")

RSRPLabel.grid(row=3, column=0)

#Declaring entry box for the user to input RSRP into and outputting it onto the user interface

RSRP = Entry(Interface, width=30)

RSRP.grid(row=3, column=1)

#Declaring label asking the user for the RSRQ and outputting it onto the user interface

RSRQLabel = Label(Interface, text="What is the RSRQ?:")

RSRQLabel.grid(row=4, column=0)

#Declaring entry box for the user to input RSRQ into and outputting it onto the user interface

RSRQ = Entry(Interface, width=30)

RSRQ.grid(row=4, column=1)

#Declaring label asking the user for the SINR and outputting it onto the user interface

SINRLabel = Label(Interface, text="What is the SINR?:")

SINRLabel.grid(row=5, column=0)

#Declaring entry box for the user to input SINR into and outputting it onto the user interface

SINR = Entry(Interface, width=30)

SINR.grid(row=5, column=1)

#Declaring label asking the user for the current downlink throughput and outputting it onto the user interface

CurrentDownlinkLabel = Label(Interface, text="What is the current downlink throughput?:")

CurrentDownlinkLabel.grid(row=6, column=0)

#Declaring entry box for the user to input current downlink throughput into and outputting it onto the user interface

CurrentDownlink = Entry(Interface, width=30)

CurrentDownlink.grid(row=6, column=1)

#Declaring label asking the user for the current uplink throughput and outputting it onto the user interface

CurrentUplinkLabel = Label(Interface, text="What is the current uplink throughput?:")

CurrentUplinkLabel.grid(row=7, column=0)

#Declaring entry box for the user to input current uplink throughput into and outputting it onto the user interface

CurrentUplink = Entry(Interface, width=30)

CurrentUplink.grid(row=7, column=1)

#Declaring label asking the user for the mean downlink throughput and outputting it onto the user interface

MeanDownlinkLabel = Label(Interface, text="What is the mean downlink throughput?:")

MeanDownlinkLabel.grid(row=8, column=0)

#Declaring entry box for the user to input mean downlink throughput into and outputting it onto the user interface

MeanDownlink = Entry(Interface, width=30)

MeanDownlink.grid(row=8, column=1)

#Declaring label asking the user for the mean uplink throughput and outputting it onto the user interface

MeanUplinkLabel = Label(Interface, text="What is the mean uplink throughput?:")

MeanUplinkLabel.grid(row=9, column=0)

#Declaring entry box for the user to input mean uplink throughput into and outputting it onto the user interface

MeanUplink = Entry(Interface, width=30)

MeanUplink.grid(row=9, column=1)

#Declaring label asking the user for the current downlink throughput average and outputting it onto the user interface

CurrentDownlinkAvgLabel = Label(Interface, text="What is the current downlink throughput average?:")

CurrentDownlinkAvgLabel.grid(row=10, column=0)

#Declaring entry box for the user to input current downlink throughput average into and outputting it onto the user interface

CurrentDownlinkAvg = Entry(Interface, width=30)

CurrentDownlinkAvg.grid(row=10, column=1)

#Declaring label asking the user for the current downlink throughput maximum and outputting it onto the user interface

CurrentDownlinkMaxLabel = Label(Interface, text="What is the current downlink throughput maximum?:")

CurrentDownlinkMaxLabel.grid(row=11, column=0)

#Declaring entry box for the user to input current downlink throughput maximum into and outputting it onto the user interface

CurrentDownlinkMax = Entry(Interface, width=30)

CurrentDownlinkMax.grid(row=11, column=1)

#Declaring label asking the user for the current uplink throughput average and outputting it onto the user interface

CurrentUplinkAvgLabel = Label(Interface, text="What is the current uplink throughput average?:")

CurrentUplinkAvgLabel.grid(row=12, column=0)

#Declaring entry box for the user to input current uplink throughput average into and outputting it onto the user interface

CurrentUplinkAvg = Entry(Interface, width=30)

CurrentUplinkAvg.grid(row=12, column=1)

#Declaring label asking the user for the current uplink throughput maximum and outputting it onto the user interface

CurrentUplinkMaxLabel = Label(Interface, text="What is the current uplink throughput maximum?:")

CurrentUplinkMaxLabel.grid(row=13, column=0)

#Declaring entry box for the user to input current uplink throughput maximum into and outputting it onto the user interface

CurrentUplinkMax = Entry(Interface, width=30)

CurrentUplinkMax.grid(row=13, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=14, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Static (Inside)", "Walking (Outside)", "Driving (Outside)")

Scenario.grid(row=14, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=15, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 3.4.6: Random Forest GUI (DL Only).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestDLOnly.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

CurrentDLValue = float(CurrentDownlink.get())

MeanDLValue = float(MeanDownlink.get())

CurrentDLAvgValue = float(CurrentDownlinkAvg.get())

CurrentDLMaxValue = float(CurrentDownlinkMax.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Static is 0, Driving is 1 and Walking is 2

if ScenarioString == "Static (Inside)":

ScenarioValue = 0

elif ScenarioString == "Driving (Outside)":

ScenarioValue = 1

elif ScenarioString == "Walking (Outside)":

ScenarioValue = 2

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, CurrentDLValue, MeanDLValue, CurrentDLAvgValue, CurrentDLMaxValue, ScenarioValue]], columns=["GPS Long", "GPS Lat", "Current Netw. DL", "Mean Netw. DL", "Current Netw. DL Avg", "Current Netw. DL Max", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "5G EN-DC":

CorrectRAT = "5G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=9, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the current downlink throughput and outputting it onto the user interface

CurrentDownlinkLabel = Label(Interface, text="What is the current downlink throughput?:")

CurrentDownlinkLabel.grid(row=3, column=0)

#Declaring entry box for the user to input current downlink throughput into and outputting it onto the user interface

CurrentDownlink = Entry(Interface, width=30)

CurrentDownlink.grid(row=3, column=1)

#Declaring label asking the user for the mean downlink throughput and outputting it onto the user interface

MeanDownlinkLabel = Label(Interface, text="What is the mean downlink throughput?:")

MeanDownlinkLabel.grid(row=4, column=0)

#Declaring entry box for the user to input mean downlink throughput into and outputting it onto the user interface

MeanDownlink = Entry(Interface, width=30)

MeanDownlink.grid(row=4, column=1)

#Declaring label asking the user for the current downlink throughput average and outputting it onto the user interface

CurrentDownlinkAvgLabel = Label(Interface, text="What is the current downlink throughput average?:")

CurrentDownlinkAvgLabel.grid(row=5, column=0)

#Declaring entry box for the user to input current downlink throughput average into and outputting it onto the user interface

CurrentDownlinkAvg = Entry(Interface, width=30)

CurrentDownlinkAvg.grid(row=5, column=1)

#Declaring label asking the user for the current downlink throughput maximum and outputting it onto the user interface

CurrentDownlinkMaxLabel = Label(Interface, text="What is the current downlink throughput maximum?:")

CurrentDownlinkMaxLabel.grid(row=6, column=0)

#Declaring entry box for the user to input current downlink throughput maximum into and outputting it onto the user interface

CurrentDownlinkMax = Entry(Interface, width=30)

CurrentDownlinkMax.grid(row=6, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=7, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Static (Inside)", "Walking (Outside)", "Driving (Outside)")

Scenario.grid(row=7, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=8, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 3.4.7: Random Forest GUI (UL Only).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestULOnly.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

CurrentULValue = float(CurrentUplink.get())

MeanULValue = float(MeanUplink.get())

CurrentULAvgValue = float(CurrentUplinkAvg.get())

CurrentULMaxValue = float(CurrentUplinkMax.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Static is 0, Driving is 1 and Walking is 2

if ScenarioString == "Static (Inside)":

ScenarioValue = 0

elif ScenarioString == "Driving (Outside)":

ScenarioValue = 1

elif ScenarioString == "Walking (Outside)":

ScenarioValue = 2

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, CurrentULValue, MeanULValue, CurrentULAvgValue, CurrentULMaxValue, ScenarioValue]], columns=["GPS Long", "GPS Lat", "Current Netw. UL", "Mean Netw. UL", "Current Netw. UL Avg", "Current Netw. UL Max", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "5G EN-DC":

CorrectRAT = "5G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=9, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the current uplink throughput and outputting it onto the user interface

CurrentUplinkLabel = Label(Interface, text="What is the current uplink throughput?:")

CurrentUplinkLabel.grid(row=3, column=0)

#Declaring entry box for the user to input current uplink throughput into and outputting it onto the user interface

CurrentUplink = Entry(Interface, width=30)

CurrentUplink.grid(row=3, column=1)

#Declaring label asking the user for the mean uplink throughput and outputting it onto the user interface

MeanUplinkLabel = Label(Interface, text="What is the mean uplink throughput?:")

MeanUplinkLabel.grid(row=4, column=0)

#Declaring entry box for the user to input mean uplink throughput into and outputting it onto the user interface

MeanUplink = Entry(Interface, width=30)

MeanUplink.grid(row=4, column=1)

#Declaring label asking the user for the current uplink throughput average and outputting it onto the user interface

CurrentUplinkAvgLabel = Label(Interface, text="What is the current uplink throughput average?:")

CurrentUplinkAvgLabel.grid(row=5, column=0)

#Declaring entry box for the user to input current uplink throughput average into and outputting it onto the user interface

CurrentUplinkAvg = Entry(Interface, width=30)

CurrentUplinkAvg.grid(row=5, column=1)

#Declaring label asking the user for the current uplink throughput maximum and outputting it onto the user interface

CurrentUplinkMaxLabel = Label(Interface, text="What is the current uplink throughput maximum?:")

CurrentUplinkMaxLabel.grid(row=6, column=0)

#Declaring entry box for the user to input current uplink throughput maximum into and outputting it onto the user interface

CurrentUplinkMax = Entry(Interface, width=30)

CurrentUplinkMax.grid(row=6, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=7, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Static (Inside)", "Walking (Outside)", "Driving (Outside)")

Scenario.grid(row=7, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=8, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

## Appendix 4: Validation Dataset Source Code

### Appendix 4.1: Pre-Processing

#### Appendix 4.1.1: Cleaning Bus Data (Status Quo Variables).py

#Importing Pandas

import pandas as pd

#Reading CSV and using the relevant columns

BusData = pd.read\_csv("CombinedBusDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "State"])

#Printing top rows of bus dataset

print(BusData.head())

#Turning the timestamp into more manipulable date and time columns

BusData[["Date", "Time"]] = BusData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of the modified bus dataset

print(BusData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a scenario marker of "Bus" for each record in the bus dataset

for i in range(0, len(BusData)):

ScenarioMarkers.append("Bus")

#Assigning a scenario marker to each record in the dataset

BusDataWithScenario = BusData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final bus dataset

print(BusDataWithScenario.head())

#Writing final dataset to a CSV file

BusDataWithScenario.to\_csv("FinalBusDataset (Status Quo).csv", index=False)

#### Appendix 4.1.2: Cleaning Bus Data (Throughput).py

#Importing Pandas

import pandas as pd

#Reading CSV and using the relevant columns

BusData = pd.read\_csv("CombinedBusDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of bus dataset

print(BusData.head())

#Turning the timestamp into more manipulable date and time columns

BusData[["Date", "Time"]] = BusData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of the modified bus dataset

print(BusData.head())

#Creating array for scenario markers

ScenarioMarkers = []

#Adding a scenario marker of "Bus" for each record in the bus dataset

for i in range(0, len(BusData)):

ScenarioMarkers.append("Bus")

#Assigning a scenario marker to each record in the dataset

BusDataWithScenario = BusData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final bus dataset

print(BusDataWithScenario.head())

#Writing final dataset to a CSV file

BusDataWithScenario.to\_csv("FinalBusDataset (Throughput).csv", index=False)

#### Appendix 4.1.3: Cleaning Bus Data (Combined).py

#Importing Pandas

import pandas as pd

#Reading CSV and using the relevant columns

BusData = pd.read\_csv("CombinedBusDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of bus dataset

print(BusData.head())

#Turning the timestamp into more manipulable date and time columns

BusData[["Date", "Time"]] = BusData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of the modified dataset

print(BusData.head())

#Creating array for scenario markers

ScenarioMarkers = []

#Adding a scenario marker for each record in the dataset

for i in range(0, len(BusData)):

ScenarioMarkers.append("Bus")

#Assigning a scenario marker to each record in the dataset

BusDataWithScenario = BusData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(BusDataWithScenario.head())

#Writing final dataset to a CSV file

BusDataWithScenario.to\_csv("FinalBusDataset.csv", index=False)

#### Appendix 4.1.4: Cleaning Car Data (Status Quo Variables).py

#Importing Pandas

import pandas as pd

#Reading CSV and using the relevant columns

CarData = pd.read\_csv("CombinedCarDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "State"])

#Printing top rows of car dataset

print(CarData.head())

#Turning the timestamp into more manipulable date and time columns

CarData[["Date", "Time"]] = CarData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(CarData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(CarData)):

ScenarioMarkers.append("Car")

#Assigning a scenario marker to each record in the dataset

CarDataWithScenario = CarData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(CarDataWithScenario.head())

#Writing final dataset to a CSV file

CarDataWithScenario.to\_csv("FinalCarDataset (Status Quo).csv", index=False)

#### Appendix 4.1.5: Cleaning Car Data (Throughput).py

#Importing Pandas

import pandas as pd

#Reading CSV and using the relevant columns

CarData = pd.read\_csv("CombinedCarDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of car dataset

print(CarData.head())

#Turning the timestamp into more manipulable date and time columns

CarData[["Date", "Time"]] = CarData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(CarData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(CarData)):

ScenarioMarkers.append("Car")

#Assigning a scenario marker to each record in the dataset

CarDataWithScenario = CarData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(CarDataWithScenario.head())

#Writing final dataset to a CSV file

CarDataWithScenario.to\_csv("FinalCarDataset (Throughput).csv", index=False)

#### Appendix 4.1.6: Cleaning Car Data (Combined).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

CarData = pd.read\_csv("CombinedCarDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of car dataset

print(CarData.head())

#Turning the timestamp into more manipulable date and time columns

CarData[["Date", "Time"]] = CarData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(CarData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(CarData)):

ScenarioMarkers.append("Car")

#Assigning a scenario marker to each record in the dataset

CarDataWithScenario = CarData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(CarDataWithScenario.head())

#Writing final dataset to a CSV file

CarDataWithScenario.to\_csv("FinalCarDataset.csv", index=False)

#### Appendix 4.1.7: Cleaning Pedestrian Data (Status Quo Variables).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

PedestrianData = pd.read\_csv("CombinedPedestrianDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "State"])

#Printing top rows of pedestrian dataset

print(PedestrianData.head())

#Turning the timestamp into more manipulable date and time columns

PedestrianData[["Date", "Time"]] = PedestrianData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(PedestrianData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(PedestrianData)):

ScenarioMarkers.append("Pedestrian")

#Assigning a scenario marker to each record in the dataset

PedestrianDataWithScenario = PedestrianData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(PedestrianDataWithScenario.head())

#Writing final dataset to a CSV file

PedestrianDataWithScenario.to\_csv("FinalPedestrianDataset (Status Quo).csv", index=False)

#### Appendix 4.1.8: Cleaning Pedestrian Data (Throughput).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

PedestrianData = pd.read\_csv("CombinedPedestrianDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of pedestrian dataset

print(PedestrianData.head())

#Turning the timestamp into more manipulable date and time columns

PedestrianData[["Date", "Time"]] = PedestrianData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(PedestrianData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(PedestrianData)):

ScenarioMarkers.append("Pedestrian")

#Assigning a scenario marker to each record in the dataset

PedestrianDataWithScenario = PedestrianData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(PedestrianDataWithScenario.head())

#Writing final dataset to a CSV file

PedestrianDataWithScenario.to\_csv("FinalPedestrianDataset (Throughput).csv", index=False)

#### Appendix 4.1.9: Cleaning Pedestrian Data (Combined).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

PedestrianData = pd.read\_csv("CombinedPedestrianDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of pedestrian dataset

print(PedestrianData.head())

#Turning the timestamp into more manipulable date and time columns

PedestrianData[["Date", "Time"]] = PedestrianData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(PedestrianData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(PedestrianData)):

ScenarioMarkers.append("Pedestrian")

#Assigning a scenario marker to each record in the dataset

PedestrianDataWithScenario = PedestrianData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(PedestrianDataWithScenario.head())

#Writing final dataset to a CSV file

PedestrianDataWithScenario.to\_csv("FinalPedestrianDataset.csv", index=False)

#### Appendix 4.1.10: Cleaning Static Data (Status Quo Variables).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

StaticData = pd.read\_csv("CombinedStaticDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "State"])

#Printing top rows of static dataset

print(StaticData.head())

#Turning the timestamp into more manipulable date and time columns

StaticData[["Date", "Time"]] = StaticData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(StaticData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(StaticData)):

ScenarioMarkers.append("Static")

#Assigning a scenario marker to each record in the dataset

StaticDataWithScenario = StaticData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(StaticDataWithScenario.head())

#Writing final dataset to a CSV file

StaticDataWithScenario.to\_csv("FinalStaticDataset (Status Quo Variables).csv", index=False)

#### Appendix 4.1.11: Cleaning Static Data (Throughput).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

StaticData = pd.read\_csv("CombinedStaticDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of static dataset

print(StaticData.head())

#Turning the timestamp into more manipulable date and time columns

StaticData[["Date", "Time"]] = StaticData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(StaticData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(StaticData)):

ScenarioMarkers.append("Static")

#Assigning a scenario marker to each record in the dataset

StaticDataWithScenario = StaticData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(StaticDataWithScenario.head())

#Writing final dataset to a CSV file

StaticDataWithScenario.to\_csv("FinalStaticDataset (Throughput).csv", index=False)

#### Appendix 4.1.12: Cleaning Static Data (Combined).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

StaticData = pd.read\_csv("CombinedStaticDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of static dataset

print(StaticData.head())

#Turning the timestamp into more manipulable date and time columns

StaticData[["Date", "Time"]] = StaticData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(StaticData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(StaticData)):

ScenarioMarkers.append("Static")

#Assigning a scenario marker to each record in the dataset

StaticDataWithScenario = StaticData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(StaticDataWithScenario.head())

#Writing final dataset to a CSV file

StaticDataWithScenario.to\_csv("FinalStaticDataset.csv", index=False)

#### Appendix 4.1.13: Cleaning Train Data (Status Quo Variables).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

TrainData = pd.read\_csv("CombinedTrainDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "State"])

#Printing top rows of train dataset

print(TrainData.head())

#Turning the timestamp into more manipulable date and time columns

TrainData[["Date", "Time"]] = TrainData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(TrainData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(TrainData)):

ScenarioMarkers.append("Train")

#Assigning a scenario marker to each record in the dataset

TrainDataWithScenario = TrainData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(TrainDataWithScenario.head())

#Writing final dataset to a CSV file

TrainDataWithScenario.to\_csv("FinalTrainDataset (Status Quo Variables).csv", index=False)

#### Appendix 4.1.14: Cleaning Train Data (Throughput).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

TrainData = pd.read\_csv("CombinedTrainDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of train dataset

print(TrainData.head())

#Turning the timestamp into more manipulable date and time columns

TrainData[["Date", "Time"]] = TrainData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(TrainData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(TrainData)):

ScenarioMarkers.append("Train")

#Assigning a scenario marker to each record in the dataset

TrainDataWithScenario = TrainData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(TrainDataWithScenario.head())

#Writing final dataset to a CSV file

TrainDataWithScenario.to\_csv("FinalTrainDataset (Throughput).csv", index=False)

#### Appendix 4.1.15: Cleaning Train Data (Combined).py

#Importing Pandas

import pandas as pd

#Reading CSV and using relevant columns

TrainData = pd.read\_csv("CombinedTrainDataset.csv", usecols=["Timestamp", "Longitude", "Latitude", "Speed", "NetworkMode", "RSRP", "RSRQ", "DL\_bitrate", "UL\_bitrate", "State"])

#Printing top rows of train dataset

print(TrainData.head())

#Turning the timestamp into more manipulable date and time columns

TrainData[["Date", "Time"]] = TrainData["Timestamp"].str.split("\_", expand=True)

#Printing top rows of modified dataset

print(TrainData.head())

#Declaring array for scenario markers

ScenarioMarkers = []

#Adding a relevant scenario marker to each record in the dataset

for i in range(0, len(TrainData)):

ScenarioMarkers.append("Train")

#Assigning a scenario marker to each record in the dataset

TrainDataWithScenario = TrainData.assign(Scenario=ScenarioMarkers)

#Printing top rows of final dataset

print(TrainDataWithScenario.head())

#Writing final dataset to a CSV file

TrainDataWithScenario.to\_csv("FinalTrainDataset.csv", index=False)

#### Appendix 4.1.16: Combining All Data (Status Quo Variables).py

#Importing Pandas

import pandas as pd

#Declaring list to store DataFrames

DataFrameList = []

#Reading CSVs for each mobility scenario individually

BusDataFrame = pd.read\_csv("FinalBusDataset (Status Quo).csv")

CarDataFrame = pd.read\_csv("FinalCarDataset (Status Quo).csv")

PedestrianDataFrame = pd.read\_csv("FinalPedestrianDataset (Status Quo).csv")

StaticDataFrame = pd.read\_csv("FinalStaticDataset (Status Quo Variables).csv")

TrainDataFrame = pd.read\_csv("FinalTrainDataset (Status Quo Variables).csv")

#Adding each mobility scenario's dataset to the DataFrame list

DataFrameList.append(BusDataFrame)

DataFrameList.append(CarDataFrame)

DataFrameList.append(PedestrianDataFrame)

DataFrameList.append(StaticDataFrame)

DataFrameList.append(TrainDataFrame)

#Combining all datasets

CombinedDataFrame = pd.concat(DataFrameList, ignore\_index=True)

#Printing a summary of the final dataset

print(CombinedDataFrame)

#Writing the final dataset to a CSV file

CombinedDataFrame.to\_csv("FinalCombinedDataset (Status Quo Variables).csv", index=False)

#### Appendix 4.1.17: Combining All Data (Throughput).py

#Importing Pandas

import pandas as pd

#Declaring list to store DataFrames

DataFrameList = []

#Reading CSVs for each mobility scenario individually

BusDataFrame = pd.read\_csv("FinalBusDataset (Throughput).csv")

CarDataFrame = pd.read\_csv("FinalCarDataset (Throughput).csv")

PedestrianDataFrame = pd.read\_csv("FinalPedestrianDataset (Throughput).csv")

StaticDataFrame = pd.read\_csv("FinalStaticDataset (Throughput).csv")

TrainDataFrame = pd.read\_csv("FinalTrainDataset (Throughput).csv")

#Adding each mobility scenario's dataset to the DataFrame list

DataFrameList.append(BusDataFrame)

DataFrameList.append(CarDataFrame)

DataFrameList.append(PedestrianDataFrame)

DataFrameList.append(StaticDataFrame)

DataFrameList.append(TrainDataFrame)

#Combining all datasets

CombinedDataFrame = pd.concat(DataFrameList, ignore\_index=True)

#Printing a summary of the final dataset

print(CombinedDataFrame)

#Writing the final dataset to a CSV file

CombinedDataFrame.to\_csv("FinalCombinedDataset (Throughput).csv", index=False)

#### Appendix 4.1.18: Combining All Data (Combined).py

#Importing Pandas

import pandas as pd

#Declaring list to store DataFrames

DataFrameList = []

#Reading CSVs for each mobility scenario individually

BusDataFrame = pd.read\_csv("FinalBusDataset.csv")

CarDataFrame = pd.read\_csv("FinalCarDataset.csv")

PedestrianDataFrame = pd.read\_csv("FinalPedestrianDataset.csv")

StaticDataFrame = pd.read\_csv("FinalStaticDataset.csv")

TrainDataFrame = pd.read\_csv("FinalTrainDataset.csv")

#Adding each mobility scenario's dataset to the DataFrame list

DataFrameList.append(BusDataFrame)

DataFrameList.append(CarDataFrame)

DataFrameList.append(PedestrianDataFrame)

DataFrameList.append(StaticDataFrame)

DataFrameList.append(TrainDataFrame)

#Combining all datasets

CombinedDataFrame = pd.concat(DataFrameList, ignore\_index=True)

#Printing a summary of the final dataset

print(CombinedDataFrame)

#Writing the final dataset to a CSV file

CombinedDataFrame.to\_csv("FinalCombinedDataset.csv", index=False)

#### Appendix 4.1.19: Combining Bus Data.py

#Importing libraries

import pandas as pd

import os

#Declaring folder path of bus dataset files (This will differ if executed on a different computer, apart from the final "bus" part)

FolderPath = r'C:\Users\Matthew Newman\Documents\University of Gloucestershire\BSc Hons Computer Science\Level 6\CT6039 Dissertation\Validation Dataset\Dataset\bus'

#Listing all files in the folder path

AllFiles = os.listdir(FolderPath)

#Filtering files in the folder path so that only CSV files are used

CSVFiles = [f for f in AllFiles if f.endswith('.csv')]

#Declaring list to store DataFrames

DataFrameList = []

#Reading each CSV within the folder path and adding it to the DataFrame list

#If errors are encountered, the reading is terminated and the user is informed

for csv in CSVFiles:

FilePath = os.path.join(FolderPath, csv)

try:

DataFrame = pd.read\_csv(FilePath)

DataFrameList.append(DataFrame)

except UnicodeDecodeError:

try:

DataFrame = pd.read\_csv(FilePath, sep='\t', encoding='utf-16')

DataFrameList.append(DataFrame)

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

#Combining all DataFrames

CombinedDataFrame = pd.concat(DataFrameList, ignore\_index=True)

#Dropping index column from combined dataset

CombinedDataFrame.drop(["Unnamed: 0"], inplace=True, axis=1)

#Writing combined dataset to a CSV file

CombinedDataFrame.to\_csv(os.path.join(FolderPath, "CombinedBusDataset.csv"), index=False)

#### Appendix 4.1.20: Combining Car Data.py

#Importing libraries

import pandas as pd

import os

#Declaring folder path of car dataset files (This will differ if executed on a different computer, apart from the final "car" part)

FolderPath = r'C:\Users\Matthew Newman\Documents\University of Gloucestershire\BSc Hons Computer Science\Level 6\CT6039 Dissertation\Validation Dataset\Dataset\car'

#Listing all files in the folder path

AllFiles = os.listdir(FolderPath)

#Filtering files in the folder path so that only CSV files are used

CSVFiles = [f for f in AllFiles if f.endswith('.csv')]

#Declaring list to store DataFrames

DataFrameList = []

#Reading each CSV within the folder path and adding it to the DataFrame list

#If errors are encountered, the reading is terminated and the user is informed

for csv in CSVFiles:

FilePath = os.path.join(FolderPath, csv)

try:

DataFrame = pd.read\_csv(FilePath)

DataFrameList.append(DataFrame)

except UnicodeDecodeError:

try:

DataFrame = pd.read\_csv(FilePath, sep='\t', encoding='utf-16')

DataFrameList.append(DataFrame)

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

#Combining all DataFrames

CombinedDataFrame = pd.concat(DataFrameList, ignore\_index=True)

#Dropping index column from combined dataset

CombinedDataFrame.drop(["Unnamed: 0.1"], inplace=True, axis=1)

#Writing combined dataset to a CSV file

CombinedDataFrame.to\_csv(os.path.join(FolderPath, "CombinedCarDataset.csv"), index=False)

#### Appendix 4.1.21: Combining Pedestrian Data.py

#Importing libraries

import pandas as pd

import os

#Declaring folder path of pedestrian dataset files (This will differ if executed on a different computer, apart from the final "pedestrian" dataset)

FolderPath = r'C:\Users\Matthew Newman\Documents\University of Gloucestershire\BSc Hons Computer Science\Level 6\CT6039 Dissertation\Validation Dataset\Dataset\pedestrian'

#Listing all files in the folder path

AllFiles = os.listdir(FolderPath)

#Filtering files in the folder path so that only CSV files are used

CSVFiles = [f for f in AllFiles if f.endswith('.csv')]

#Declaring list to store DataFrames

DataFrameList = []

#Reading each CSV within the folder path and adding it to the DataFrame list

#If errors are encountered, the reading is terminated and the user is informed

for csv in CSVFiles:

FilePath = os.path.join(FolderPath, csv)

try:

DataFrame = pd.read\_csv(FilePath)

DataFrameList.append(DataFrame)

except UnicodeDecodeError:

try:

DataFrame = pd.read\_csv(FilePath, sep='\t', encoding='utf-16')

DataFrameList.append(DataFrame)

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

#Combining all DataFrames

CombinedDataFrame = pd.concat(DataFrameList, ignore\_index=True)

#Dropping index column from combined dataset

CombinedDataFrame.drop(["Unnamed: 0"], inplace=True, axis=1)

#Writing combined dataset to a CSV file

CombinedDataFrame.to\_csv(os.path.join(FolderPath, "CombinedPedestrianDataset.csv"), index=False)

#### Appendix 4.1.22: Combining Static Data.py

#Importing libraries

import pandas as pd

import os

#Declaring folder path of static dataset files (This will differ if executed on a different computer, apart from the final "static" part)

FolderPath = r'C:\Users\Matthew Newman\Documents\University of Gloucestershire\BSc Hons Computer Science\Level 6\CT6039 Dissertation\Validation Dataset\Dataset\static'

#Listing all files in the folder path

AllFiles = os.listdir(FolderPath)

#Filtering files in the folder path so that only CSV files are used

CSVFiles = [f for f in AllFiles if f.endswith('.csv')]

#Declaring list to store DataFrames

DataFrameList = []

#Reading each CSV within the folder path and adding it to the DataFrame list

#If errors are encountered, the reading is terminated and the user is informed

for csv in CSVFiles:

FilePath = os.path.join(FolderPath, csv)

try:

DataFrame = pd.read\_csv(FilePath)

DataFrameList.append(DataFrame)

except UnicodeDecodeError:

try:

DataFrame = pd.read\_csv(FilePath, sep='\t', encoding='utf-16')

DataFrameList.append(DataFrame)

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

#Combining all DataFrames

CombinedDataFrame = pd.concat(DataFrameList, ignore\_index=True)

#Dropping index column from combined dataset

CombinedDataFrame.drop(["Unnamed: 0"], inplace=True, axis=1)

#Writing combined dataset to a CSV file

CombinedDataFrame.to\_csv(os.path.join(FolderPath, "CombinedStaticDataset.csv"), index=False)

#### Appendix 4.1.23: Combining Train Data.py

#Importing libraries

import pandas as pd

import os

#Declaring folder path of train dataset files (This will differ if executed on a different computer, apart from the final "train" part)

FolderPath = r'C:\Users\Matthew Newman\Documents\University of Gloucestershire\BSc Hons Computer Science\Level 6\CT6039 Dissertation\Validation Dataset\Dataset\train'

#Filtering files in the folder path so that only CSV files are used

AllFiles = os.listdir(FolderPath)

#Filtering files in the folder path so that only CSV files are used

CSVFiles = [f for f in AllFiles if f.endswith('.csv')]

#Declaring list to store DataFrames

DataFrameList = []

#Reading each CSV within the folder path and adding it to the DataFrame list

#If errors are encountered, the reading is terminated and the user is informed

for csv in CSVFiles:

FilePath = os.path.join(FolderPath, csv)

try:

DataFrame = pd.read\_csv(FilePath)

DataFrameList.append(DataFrame)

except UnicodeDecodeError:

try:

DataFrame = pd.read\_csv(FilePath, sep='\t', encoding='utf-16')

DataFrameList.append(DataFrame)

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

except Exception as e:

print(f"Could not read the file {csv} because of error: {e}")

#Combining all DataFrames

CombinedDataFrame = pd.concat(DataFrameList, ignore\_index=True)

#Dropping index column from combined dataset

CombinedDataFrame.drop(["Unnamed: 0"], inplace=True, axis=1)

#Writing combined dataset to a CSV file

CombinedDataFrame.to\_csv(os.path.join(FolderPath, "CombinedTrainDataset.csv"), index=False)

#### Appendix 4.1.24: Filtering Combined Dataset (Status Quo Variables).py

#Importing Pandas

import pandas as pd

#Reading CSV

DataFrame = pd.read\_csv("FinalCombinedDataset (Status Quo Variables).csv")

#Finding LTE (4G) and HSPA+ (3G) data within the dataset

FourGData = DataFrame[DataFrame["NetworkMode"] == "LTE"]

print(FourGData.head())

ThreeGData = DataFrame[DataFrame["NetworkMode"] == "HSPA+"]

print(ThreeGData.head())

#Cleaning the data so that only 4G and 3G data is kept and writing the cleaned dataset to a CSV file

Data = [FourGData, ThreeGData]

CleanedData = pd.concat(Data)

print(CleanedData.head())

CleanedData.to\_csv("Filtered Combined Dataset (Status Quo Variables).csv", index=False)

#### Appendix 4.1.25: Filtering Combined Dataset (Throughput).py

#Importing Pandas

import pandas as pd

#Reading CSV

DataFrame = pd.read\_csv("FinalCombinedDataset (Throughput).csv")

#Finding LTE (4G) and HSPA+ (3G) data within the dataset

FourGData = DataFrame[DataFrame["NetworkMode"] == "LTE"]

print(FourGData.head())

ThreeGData = DataFrame[DataFrame["NetworkMode"] == "HSPA+"]

print(ThreeGData.head())

#Cleaning the data so that only 4G and 3G data is kept and writing the cleaned dataset to a CSV file

Data = [FourGData, ThreeGData]

CleanedData = pd.concat(Data)

print(CleanedData.head())

CleanedData.to\_csv("Filtered Combined Dataset (Throughput).csv", index=False)

#### Appendix 4.1.26: Filtering Combined Dataset (Combined).py

#Importing Pandas

import pandas as pd

#Reading CSV

DataFrame = pd.read\_csv("FinalCombinedDataset.csv")

#Finding LTE (4G) data and HSPA+ (3G) data within the dataset

FourGData = DataFrame[DataFrame["NetworkMode"] == "LTE"]

print(FourGData.head())

ThreeGData = DataFrame[DataFrame["NetworkMode"] == "HSPA+"]

print(ThreeGData.head())

#Cleaning the data so that only 4G and 3G data is kept and writing the cleaned dataset to a CSV file

Data = [FourGData, ThreeGData]

CleanedData = pd.concat(Data)

print(CleanedData.head())

CleanedData.to\_csv("Filtered Combined Dataset (All).csv", index=False)

### Appendix 4.2: Dataset Visualisation

#### Appendix 4.2.1: Proportion of Network Modes (Combined).py

#Importing libraries

import pandas as pd

import matplotlib.pyplot as plt

#Reading CSV

DataFrame = pd.read\_csv("Filtered Combined Dataset (All).csv", na\_values="-")

#Grouping dataset by the RAT connected to

GroupedDataFrame = DataFrame.groupby(["NetworkMode"]).count()

#Printing a summary of the grouped dataset

print(GroupedDataFrame)

#Creating a pie chart showing the proportion of each RAT

plt.pie(GroupedDataFrame["Longitude"], autopct='%1.0f%%')

#Showing pie chart

plt.show()

#Dropping any records with missing values from the dataset

CleanedDataFrame = DataFrame.dropna()

#Grouping dataset by the RAT connected to

GroupedCleanDataFrame = CleanedDataFrame.groupby(["NetworkMode"]).count()

#Printing a summary of the grouped dataset

print(GroupedCleanDataFrame)

#Creating a pie chart showing the proportion of each RAT

plt.pie(GroupedCleanDataFrame["Longitude"], autopct='%1.0f%%')

#Showing pie chart

plt.show()

#### Appendix 4.2.2: Proportion of Network Modes (Status Quo Variables).py

#Importing libraries

import pandas as pd

import matplotlib.pyplot as plt

#Reading CSV

DataFrame = pd.read\_csv("Filtered Combined Dataset (Status Quo Variables).csv", na\_values="-")

#Grouping dataset by the RAT connected to

GroupedDataFrame = DataFrame.groupby(["NetworkMode"]).count()

#Printing a summary of the grouped dataset

print(GroupedDataFrame)

#Creating a pie chart showing the proportion of each RAT

plt.pie(GroupedDataFrame["Longitude"], autopct='%1.0f%%')

#Showing pie chart

plt.show()

#Dropping any records with missing values from the dataset

CleanedDataFrame = DataFrame.dropna()

#Grouping dataset by the RAT connected to

GroupedCleanDataFrame = CleanedDataFrame.groupby(["NetworkMode"]).count()

#Printing a summary of the grouped dataset

print(GroupedCleanDataFrame)

#Creating a pie chart showing the proportion of each RAT

plt.pie(GroupedCleanDataFrame["Longitude"], autopct='%1.0f%%')

#Showing pie chart

plt.show()

#### Appendix 3.2.3: Proportion of Network Modes (Throughput).py

#Importing libraries

import pandas as pd

import matplotlib.pyplot as plt

#Reading CSV

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Grouping dataset by the RAT connected to

GroupedDataFrame = DataFrame.groupby(["NetworkMode"]).count()

#Printing a summary of the grouped dataset

print(GroupedDataFrame)

#Creating a pie chart showing the proportion of each RAT

plt.pie(GroupedDataFrame["Longitude"], autopct='%1.0f%%')

#Showing pie chart

plt.show()

#Dropping any records with missing values from the dataset

CleanedDataFrame = DataFrame.dropna()

#Grouping dataset by the RAT connected to

GroupedCleanDataFrame = CleanedDataFrame.groupby(["NetworkMode"]).count()

#Printing a summary of the grouped dataset

print(GroupedCleanDataFrame)

#Creating a pie chart showing the proportion of each RAT

plt.pie(GroupedCleanDataFrame["Longitude"], autopct='%1.0f%%')

#Showing pie chart

plt.show()

### Appendix 4.3: AI Models

#### Appendix 4.3.1: Uniform Dummy Classifier (Status Quo Variables).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.dummy import DummyClassifier

from sklearn.metrics import \*

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Status Quo Variables).csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

#Converting State into a Category type and encoding the values as integers

DataFrame["State"] = DataFrame["State"].astype("category")

DataFrame["State Category"] = DataFrame["State"].cat.codes

#Converting Scenario into a Category type and encoding the values as integers

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating dummy classification model

Dummy = DummyClassifier(strategy="uniform")

#Fitting the dummy classifier to the training data

Dummy.fit(XTrain, YTrain)

#Making predictions on the test dataset using the dummy model

YPrediction = Dummy.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

ThreeGPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

ThreeGRecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

PrecisionArray = [ThreeGPrecision, FourGPrecision]

RecallArray = [ThreeGRecall, FourGRecall]

Precision = np.mean(PrecisionArray)

Recall = np.mean(RecallArray)

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("Precision:",Precision)

print("Recall:",Recall)

#Notifying the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation")

#Creating DummyClassifier for K-fold cross validation

KFoldDummy = DummyClassifier(strategy="uniform")

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldDummy, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.2: Most Frequent Dummy Classifier (Status Quo Variables).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.dummy import DummyClassifier

from sklearn.metrics import \*

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Status Quo Variables).csv")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

#Converting State into a Category type and encoding the values as integers

DataFrame["State"] = DataFrame["State"].astype("category")

DataFrame["State Category"] = DataFrame["State"].cat.codes

#Converting Scenario into a Category type and encoding the values as integers

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating dummy classification model

Dummy = DummyClassifier(strategy="most\_frequent")

#Fitting the dummy classifier to the training data

Dummy.fit(XTrain, YTrain)

#Making predictions on the test dataset using the dummy model

YPrediction = Dummy.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

ThreeGPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

ThreeGRecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

PrecisionArray = [ThreeGPrecision, FourGPrecision]

RecallArray = [ThreeGRecall, FourGRecall]

Precision = np.mean(PrecisionArray)

Recall = np.mean(RecallArray)

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("Precision:",Precision)

print("Recall:",Recall)

#Notifying the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation")

#Creating DummyClassifier for K-fold cross validation

KFoldDummy = DummyClassifier(strategy="most\_frequent")

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldDummy, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.3: Naïve Bayes Classifier (Status Quo Variables)

#Importing libraries

import numpy as np

import pandas as pd

from sklearn.preprocessing import RobustScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Status Quo Variables).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

#Converting Scenario into a Category type and encoding the values as integers

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Denoting the columns of the dataset

Columns = XTrain.columns

#Declaring a scaler to scale the data for input into a Naive-Bayes Classifier

Scaler = RobustScaler()

#Fitting the scaler to the training dataset and transforming it

XTrain = Scaler.fit\_transform(XTrain)

#Transforming the testing dataset

XTest = Scaler.transform(XTest)

#Creating a Pandas DataFrame out of the scaled training data

XTrain = pd.DataFrame(XTrain, columns=[Columns])

#Creating a Pandas DataFrame out of the scaled testing data

XTest = pd.DataFrame(XTest, columns=[Columns])

#Declaring a Naive-Bayes Classifier

NaiveBayesClassifier = GaussianNB()

#Fitting the Naive-Bayes Classifier to the training and testing datasets

NaiveBayesClassifier.fit(XTrain, YTrain)

#Making predictions on the testing data using the Naive-Bayes Classifier

YPrediction = NaiveBayesClassifier.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

LTEPrecision = precision\_score(YTest, YPrediction, average="macro", labels=["LTE"])

LTERecall = recall\_score(YTest, YPrediction, average="macro", labels=["LTE"])

LTEFOneScore = f1\_score(YTest, YPrediction, average="macro", labels=["LTE"])

HSPAPlusPrecision = precision\_score(YTest, YPrediction, average="macro", labels=["HSPA+"])

HSPAPlusRecall = recall\_score(YTest, YPrediction, average="macro", labels=["HSPA+"])

HSPAPlusFOneScore = f1\_score(YTest, YPrediction, average="macro", labels=["HSPA+"])

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("LTE Precision:",LTEPrecision)

print("LTE Recall:",LTERecall)

print("LTE F1 Score:",LTEFOneScore)

print("HSPA+ Precision:",HSPAPlusPrecision)

print("HSPA+ Recall:",HSPAPlusRecall)

print("HSPA+ F1 Score:",HSPAPlusFOneScore)

#Saving the trained model and RobustScaler to separate files

pickle.dump(NaiveBayesClassifier, open("NaiveBayesClassifierStatusQuoVariables.sav", "wb"))

pickle.dump(Scaler, open("RobustScalerStatusQuoVariables.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Denoting the columns of the dataset for K-fold cross validation

KFoldColumns = X.columns

#Declaring a scaler to scale the data for input into a Naive-Bayes Classifier for K-fold cross validation

KFoldScaler = RobustScaler()

#Fitting the scaler to the training dataset and transforming it for K-fold cross validation

X = KFoldScaler.fit\_transform(X)

#Creating a Pandas DataFrame out of the scaled training data for K-fold cross validation

X = pd.DataFrame(X, columns=[KFoldColumns])

#Declaring a Naive-Bayes Classifier for K-fold cross validation

KFoldNaiveBayesClassifier = GaussianNB()

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldNaiveBayesClassifier, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.4: Neural Network (Status Quo Variables).py

#Importing libraries

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Activation, Dense, BatchNormalization, Dropout

from tensorflow.keras import optimizers

from tensorflow.keras.callbacks import EarlyStopping

from scikeras.wrappers import KerasClassifier

from sklearn.preprocessing import RobustScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn import metrics

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Status Quo Variables).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

#Converting RAT Info into category codes

DataFrame["NetworkMode"] = DataFrame["NetworkMode"].astype("category")

DataFrame["NetworkMode"] = DataFrame["NetworkMode"].cat.codes

#Converting scenarios into category codes

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining used dataset categories

UsedData = DataFrame[["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category", "NetworkMode"]]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing/validation data

Train, TempTest = train\_test\_split(UsedData, test\_size=0.2, random\_state=100)

#Dividing the testing/validation data into 50% testing data and 50% validation data

Test, Validation = train\_test\_split(TempTest, test\_size=0.5, random\_state=100)

#Removing the RAT Info column from the train, test and validation datasets

TrainLabels = Train.pop("NetworkMode")

TestLabels = Test.pop("NetworkMode")

ValidationLabels = Validation.pop("NetworkMode")

#Declaring MinMaxScaler to normalise the data

Scaler = RobustScaler()

#Normalising train, test and validation datasets

NormalisedTrainData = Scaler.fit\_transform(Train)

NormalisedTestData = Scaler.transform(Test)

NormalisedValidationData = Scaler.transform(Validation)

#Declaring neural network

NeuralNetwork = Sequential()

#Adding a dense input layer, three dense hidden layers with increasing numbers of neurons for learning, and a dense output layer

NeuralNetwork.add(Dense(32, input\_shape=(NormalisedTrainData.shape[1],)))

NeuralNetwork.add(Dense(32, activation="tanh"))

NeuralNetwork.add(Dense(64, activation="tanh"))

NeuralNetwork.add(Dense(128, activation="tanh"))

NeuralNetwork.add(Dense(1, activation="sigmoid"))

#Setting the hyperparameters of the model

LearningRate = 0.01

Optimiser = optimizers.SGD(LearningRate)

NeuralNetwork.compile(loss="binary\_crossentropy", optimizer=Optimiser, metrics=["acc"])

BatchSize = 16

#Fitting the neural network to the training data

NeuralNetwork.fit(NormalisedTrainData, TrainLabels, batch\_size=BatchSize, epochs=100, verbose=2, shuffle=True, validation\_data = (NormalisedValidationData, ValidationLabels))

#Evaluating the neural network on the testing data

NeuralNetwork.evaluate(NormalisedTestData, TestLabels, verbose=2)

#Making predictions on the testing data

Prediction = NeuralNetwork.predict(NormalisedTestData, batch\_size=BatchSize, verbose=2)

PredictionBoolean = np.round(Prediction, 0)

#Printing classification report to summarise the efficacy of the model

print(metrics.classification\_report(TestLabels, PredictionBoolean))

#Saving model and RobustScaler to separate files

NeuralNetwork.save("NeuralNetworkStatusQuoVariables.h5")

pickle.dump(Scaler, open("NeuralNetworkScalerStatusQuoVariables.sav", "wb"))

#Notifying the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation")

#Subroutine to create neural network for K-fold cross validation

def CreateModel():

#Declaring neural network

KFoldNeuralNetwork = Sequential()

#Adding a dense input layer, three dense hidden layers with increasing numbers of neurons for learning, and a dense output layer

KFoldNeuralNetwork.add(Dense(32, input\_shape=(NormalisedData.shape[1],)))

KFoldNeuralNetwork.add(Dense(32, activation="tanh"))

KFoldNeuralNetwork.add(Dense(64, activation="tanh"))

KFoldNeuralNetwork.add(Dense(128, activation="tanh"))

KFoldNeuralNetwork.add(Dense(1, activation="sigmoid"))

#Setting the hyperparameters of the model

LearningRate = 0.01

Optimiser = optimizers.SGD(LearningRate)

KFoldNeuralNetwork.compile(loss="binary\_crossentropy", optimizer=Optimiser, metrics=["acc"])

BatchSize = 16

#Returning the neural network at the end of the subroutine

return KFoldNeuralNetwork

#Outlining X and Y for K-fold cross validation

X = DataFrame[["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category", "NetworkMode"]]

Y = X.pop("NetworkMode")

#Declaring RobustScaler to normalise the data for K-fold cross validation

KFoldScaler = RobustScaler()

#Normalising data for K-fold cross validation

NormalisedData = KFoldScaler.fit\_transform(X)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Creating KerasClassifier wrapper for neural network for K-fold cross validation, performing K-fold cross validation and showing the results

NeuralNetworkSKLearn = KerasClassifier(model=CreateModel, epochs=100, batch\_size=16, verbose=2)

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

Results = cross\_validate(NeuralNetworkSKLearn, NormalisedData, Y, scoring=Metrics, cv=Validator)

print(Results)

print("Accuracy:",np.mean(Results["test\_accuracy"]))

print("Precision:",np.mean(Results["test\_precision"]))

print("Recall:",np.mean(Results["test\_recall"]))

#### Appendix 4.3.5: Random Forest (Status Quo Variables).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Status Quo Variables).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestStatusQuoVariables.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.6: Random Forest (Throughput).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.7: Random Forest (Combined).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (All).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "RSRP", "RSRQ", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestAll.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.8: Random Forest (Downlink Only).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestDownlinkOnly.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.9: Random Forest (Uplink Only).py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

pickle.dump(RandomForest, open("RandomForestUplinkOnly.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.10: Random Forest with 50 Trees.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=50, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=50, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.11: Random Forest with 25 Trees.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=25, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=25, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.12: Random Forest with 10 Trees.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=10, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=10, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.13: Random Forest with 5 Trees.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.14: Random Forest with 1 Tree.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=1, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=1, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=27)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.15: Random Forest with Max Depth of 20.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=20)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=20)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.16: Random Forest with Max Depth of 15.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=15)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=15)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.17: Random Forest with Max Depth of 10.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=10)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=10)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.18: Random Forest with Max Depth of 5.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=5)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=5)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

#### Appendix 4.3.19: Random Forest with Max Depth of 1.py

#Importing libraries

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_validate

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import \*

import pickle

#Reading CSV and converting it to a DataFrame

DataFrame = pd.read\_csv("Filtered Combined Dataset (Throughput).csv", na\_values="-")

#Dropping rows with any missing values from DataFrame

DataFrame = DataFrame.dropna()

DataFrame["Scenario"] = DataFrame["Scenario"].astype("category")

DataFrame["Scenario Category"] = DataFrame["Scenario"].cat.codes

#Outlining x and y. x represents the independent predictor variables, while y represents the dependent target variable

X = DataFrame[["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"]]

Y = DataFrame["NetworkMode"]

#Dividing the dataset into train and test datasets. 80% of the data is allocated to training, while the other 20% is set aside as testing data

XTrain, XTest, YTrain, YTest = train\_test\_split(X, Y, test\_size=0.2, random\_state=100)

#Generating random forest classification model

RandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=1)

#Searching for optimal random forest hyperparameters for the training data given

RandomForest.fit(XTrain, YTrain)

#Making predictions on the test dataset using the optimised random forest model

YPrediction = RandomForest.predict(XTest)

#Calculating model accuracy

Accuracy = accuracy\_score(YTest, YPrediction)

HSPAPrecision = precision\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPARecall = recall\_score(YTest, YPrediction, pos\_label="HSPA+")

HSPAFOneScore = f1\_score(YTest, YPrediction, pos\_label="HSPA+")

FourGPrecision = precision\_score(YTest, YPrediction, pos\_label="LTE")

FourGRecall = recall\_score(YTest, YPrediction, pos\_label="LTE")

FourGFOneScore = f1\_score(YTest, YPrediction, pos\_label="LTE")

#Outputting model accuracy

print("Model accuracy on test data is:",Accuracy)

print("HSPA+ Precision:",HSPAPrecision)

print("HSPA+ Recall:",HSPARecall)

print("HSPA+ F1 Score:",HSPAFOneScore)

print("4G Precision:",FourGPrecision)

print("4G Recall:",FourGRecall)

print("4G F1 Score:",FourGFOneScore)

#Saving the trained model to a separate file

#pickle.dump(RandomForest, open("RandomForestThroughput.sav", "wb"))

#Telling the user that K-fold cross validation is beginning

print("Beginning K-fold cross validation.")

#Generating random forest classification model for K-fold cross validation

KFoldRandomForest = RandomForestClassifier(n\_estimators=100, min\_samples\_split=2, min\_samples\_leaf=1, max\_features="sqrt", max\_depth=1)

#Creating K-fold cross validation validator

Validator = KFold(n\_splits=10, random\_state=1, shuffle=True)

#Defining cross-validation metrics

Metrics = {"accuracy" : "accuracy",

"precision" : "precision\_macro",

"recall" : "recall\_macro"}

#Performing K-fold cross validation on model

Scores = cross\_validate(KFoldRandomForest, X, Y, scoring=Metrics, cv=Validator, n\_jobs=-1)

#Outputting the accuracy gained from K-fold cross validation

print(Scores)

print("Accuracy:",np.mean(Scores["test\_accuracy"]))

print("Precision:",np.mean(Scores["test\_precision"]))

print("Recall:",np.mean(Scores["test\_recall"]))

### Appendix 4.4: Model GUIs

#### Appendix 4.4.1: Naïve Bayes Classifier GUI (Status Quo Variables).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("NaiveBayesClassifierStatusQuoVariables.sav","rb"))

#Loading pre-trained RobustScaler

Scaler = pickle.load(open("RobustScalerStatusQuoVariables.sav", "rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

RSRPValue = float(RSRP.get())

RSRQValue = float(RSRQ.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Bus is 0, Car is 1, Pedestrian is 2, Static is 3 and Train is 4

if ScenarioString == "Bus":

ScenarioValue = 0

elif ScenarioString == "Car":

ScenarioValue = 1

elif ScenarioString == "Pedestrian":

ScenarioValue = 2

elif ScenarioString == "Static":

ScenarioValue = 3

elif ScenarioString == "Train":

ScenarioValue = 4

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, RSRPValue, RSRQValue, ScenarioValue]], columns=["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category"])

#Rescaling inputted values using the scaler

Data = Scaler.transform(Data)

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "HSPA+":

CorrectRAT = "3G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=7, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the RSRP and outputting it onto the user interface

RSRPLabel = Label(Interface, text="What is the RSRP?:")

RSRPLabel.grid(row=3, column=0)

#Declaring entry box for the user to input RSRP into and outputting it onto the user interface

RSRP = Entry(Interface, width=30)

RSRP.grid(row=3, column=1)

#Declaring label asking the user for the RSRQ and outputting it onto the user interface

RSRQLabel = Label(Interface, text="What is the RSRQ?:")

RSRQLabel.grid(row=4, column=0)

#Declaring entry box for the user to input RSRQ into and outputting it onto the user interface

RSRQ = Entry(Interface, width=30)

RSRQ.grid(row=4, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=5, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Bus", "Car", "Pedestrian", "Static", "Train")

Scenario.grid(row=5, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=6, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 4.4.2: Neural Network GUI (Status Quo Variables).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import numpy as np

import pandas as pd

import pickle

from tensorflow.keras.models import load\_model

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = load\_model("NeuralNetworkStatusQuoVariables.h5")

#Loading pre-trained RobustScaler

Scaler = pickle.load(open("NeuralNetworkScalerStatusQuoVariables.sav", "rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

RSRPValue = float(RSRP.get())

RSRQValue = float(RSRQ.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Bus is 0, Car is 1, Pedestrian is 2, Static is 3 and Train is 4

if ScenarioString == "Bus":

ScenarioValue = 0

elif ScenarioString == "Car":

ScenarioValue = 1

elif ScenarioString == "Pedestrian":

ScenarioValue = 2

elif ScenarioString == "Static":

ScenarioValue = 3

elif ScenarioString == "Train":

ScenarioValue = 4

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, RSRPValue, RSRQValue, ScenarioValue]], columns=["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category"])

#Rescaling inputted values using the scaler

Data = Scaler.transform(Data)

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

PredictionBoolean = np.round(Prediction, 0)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if PredictionBoolean == 0:

CorrectRAT = "3G"

elif PredictionBoolean == 1:

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=7, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the RSRP and outputting it onto the user interface

RSRPLabel = Label(Interface, text="What is the RSRP?:")

RSRPLabel.grid(row=3, column=0)

#Declaring entry box for the user to input RSRP into and outputting it onto the user interface

RSRP = Entry(Interface, width=30)

RSRP.grid(row=3, column=1)

#Declaring label asking the user for the RSRQ and outputting it onto the user interface

RSRQLabel = Label(Interface, text="What is the RSRQ?:")

RSRQLabel.grid(row=4, column=0)

#Declaring entry box for the user to input RSRQ into and outputting it onto the user interface

RSRQ = Entry(Interface, width=30)

RSRQ.grid(row=4, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=5, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Bus", "Car", "Pedestrian", "Static", "Train")

Scenario.grid(row=5, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=6, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 4.4.3: Random Forest GUI (Status Quo Variables).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestStatusQuoVariables.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

RSRPValue = float(RSRP.get())

RSRQValue = float(RSRQ.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Bus is 0, Car is 1, Pedestrian is 2, Static is 3 and Train is 4

if ScenarioString == "Bus":

ScenarioValue = 0

elif ScenarioString == "Car":

ScenarioValue = 1

elif ScenarioString == "Pedestrian":

ScenarioValue = 2

elif ScenarioString == "Static":

ScenarioValue = 3

elif ScenarioString == "Train":

ScenarioValue = 4

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, RSRPValue, RSRQValue, ScenarioValue]], columns=["Longitude", "Latitude", "RSRP", "RSRQ", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "HSPA+":

CorrectRAT = "3G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=7, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the RSRP and outputting it onto the user interface

RSRPLabel = Label(Interface, text="What is the RSRP?:")

RSRPLabel.grid(row=3, column=0)

#Declaring entry box for the user to input RSRP into and outputting it onto the user interface

RSRP = Entry(Interface, width=30)

RSRP.grid(row=3, column=1)

#Declaring label asking the user for the RSRQ and outputting it onto the user interface

RSRQLabel = Label(Interface, text="What is the RSRQ?:")

RSRQLabel.grid(row=4, column=0)

#Declaring entry box for the user to input RSRQ into and outputting it onto the user interface

RSRQ = Entry(Interface, width=30)

RSRQ.grid(row=4, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=5, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Bus", "Car", "Pedestrian", "Static", "Train")

Scenario.grid(row=5, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=6, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 4.4.4: Random Forest GUI (Throughput).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestThroughput.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

DownlinkValue = float(Downlink.get())

UplinkValue = float(Uplink.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Bus is 0, Car is 1, Pedestrian is 2, Static is 3 and Train is 4

if ScenarioString == "Bus":

ScenarioValue = 0

elif ScenarioString == "Car":

ScenarioValue = 1

elif ScenarioString == "Pedestrian":

ScenarioValue = 2

elif ScenarioString == "Static":

ScenarioValue = 3

elif ScenarioString == "Train":

ScenarioValue = 4

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, DownlinkValue, UplinkValue, ScenarioValue]], columns=["Longitude", "Latitude", "DL\_bitrate", "UL\_bitrate", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "HSPA+":

CorrectRAT = "3G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=7, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the downlink throughput and outputting it onto the user interface

DownlinkLabel = Label(Interface, text="What is the downlink throughput?:")

DownlinkLabel.grid(row=3, column=0)

#Declaring entry box for the user to input downlink throughput into and outputting it onto the user interface

Downlink = Entry(Interface, width=30)

Downlink.grid(row=3, column=1)

#Declaring label asking the user for the uplink throughput and outputting it onto the user interface

UplinkLabel = Label(Interface, text="What is the uplink throughput?:")

UplinkLabel.grid(row=4, column=0)

#Declaring entry box for the user to input uplink throughput into and outputting it onto the user interface

Uplink = Entry(Interface, width=30)

Uplink.grid(row=4, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=5, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Bus", "Car", "Pedestrian", "Static", "Train")

Scenario.grid(row=5, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=6, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 4.4.5: Random Forest GUI (Combined).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestAll.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

RSRPValue = float(RSRP.get())

RSRQValue = float(RSRQ.get())

DLValue = float(Downlink.get())

ULValue = float(Uplink.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Bus is 0, Car is 1, Pedestrian is 2, Static is 3 and Train is 4

if ScenarioString == "Bus":

ScenarioValue = 0

elif ScenarioString == "Car":

ScenarioValue = 1

elif ScenarioString == "Pedestrian":

ScenarioValue = 2

elif ScenarioString == "Static":

ScenarioValue = 3

elif ScenarioString == "Train":

ScenarioValue = 4

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, RSRPValue, RSRQValue, DLValue, ULValue, ScenarioValue]], columns=["Longitude", "Latitude", "RSRP", "RSRQ", "DL\_bitrate", "UL\_bitrate", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "HSPA+":

CorrectRAT = "3G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=10, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the RSRP and outputting it onto the user interface

RSRPLabel = Label(Interface, text="What is the RSRP?:")

RSRPLabel.grid(row=3, column=0)

#Declaring entry box for the user to input RSRP into and outputting it onto the user interface

RSRP = Entry(Interface, width=30)

RSRP.grid(row=3, column=1)

#Declaring label asking the user for the RSRQ and outputting it onto the user interface

RSRQLabel = Label(Interface, text="What is the RSRQ?:")

RSRQLabel.grid(row=4, column=0)

#Declaring entry box for the user to input RSRQ into and outputting it onto the user interface

RSRQ = Entry(Interface, width=30)

RSRQ.grid(row=4, column=1)

#Declaring label asking the user for the downlink throughput and outputting it onto the user interface

DownlinkLabel = Label(Interface, text="What is the downlink throughput?:")

DownlinkLabel.grid(row=5, column=0)

#Declaring entry box for the user to input downlink throughput into and outputting it onto the user interface

Downlink = Entry(Interface, width=30)

Downlink.grid(row=5, column=1)

#Declaring label asking the user for the uplink throughput and outputting it onto the user interface

UplinkLabel = Label(Interface, text="What is the uplink throughput?:")

UplinkLabel.grid(row=6, column=0)

#Declaring entry box for the user to input uplink throughput into and outputting it onto the user interface

Uplink = Entry(Interface, width=30)

Uplink.grid(row=6, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=7, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Bus", "Car", "Pedestrian", "Static", "Train")

Scenario.grid(row=8, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=9, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 4.4.6: Random Forest GUI (Downlink Only).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestDownlinkOnly.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

DownlinkValue = float(Downlink.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Bus is 0, Car is 1, Pedestrian is 2, Static is 3 and Train is 4

if ScenarioString == "Bus":

ScenarioValue = 0

elif ScenarioString == "Car":

ScenarioValue = 1

elif ScenarioString == "Pedestrian":

ScenarioValue = 2

elif ScenarioString == "Static":

ScenarioValue = 3

elif ScenarioString == "Train":

ScenarioValue = 4

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, DownlinkValue, ScenarioValue]], columns=["Longitude", "Latitude", "DL\_bitrate", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "HSPA+":

CorrectRAT = "3G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=6, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the downlink throughput and outputting it onto the user interface

DownlinkLabel = Label(Interface, text="What is the downlink throughput?:")

DownlinkLabel.grid(row=3, column=0)

#Declaring entry box for the user to input downlink throughput into and outputting it onto the user interface

Downlink = Entry(Interface, width=30)

Downlink.grid(row=3, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=4, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Bus", "Car", "Pedestrian", "Static", "Train")

Scenario.grid(row=4, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=5, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

#### Appendix 4.4.7: Random Forest GUI (Uplink Only).py

#Importing libraries

import tkinter as tk

from tkinter import \*

import pandas as pd

import pickle

#Command that is triggered when button is pressed

#This selects the correct RAT to connect to according to the user's input data

def SelectCorrectRAT():

#Loading pre-trained model

Model = pickle.load(open("RandomForestUplinkOnly.sav","rb"))

#Fetching values from the Tkinter interface and converting them into the correct data type

LongitudeValue = float(Longitude.get())

LatitudeValue = float(Latitude.get())

UplinkValue = float(Uplink.get())

ScenarioString = str(SelectedScenario.get())

#Encoding scenario strings to their corresponding category code

#Bus is 0, Car is 1, Pedestrian is 2, Static is 3 and Train is 4

if ScenarioString == "Bus":

ScenarioValue = 0

elif ScenarioString == "Car":

ScenarioValue = 1

elif ScenarioString == "Pedestrian":

ScenarioValue = 2

elif ScenarioString == "Static":

ScenarioValue = 3

elif ScenarioString == "Train":

ScenarioValue = 4

#Collating the user's input data into a DataFrame for use by the model

Data = pd.DataFrame([[LongitudeValue, LatitudeValue, UplinkValue, ScenarioValue]], columns=["Longitude", "Latitude", "UL\_bitrate", "Scenario Category"])

#Making a prediction on the data using the model

Prediction = Model.predict(Data)

#Setting text output of correct RAT depending on what variable name from the dataset is predicted

if Prediction == "HSPA+":

CorrectRAT = "3G"

elif Prediction == "LTE":

CorrectRAT = "4G"

#Declaring output text informing the user of the correct RAT to connect to

Text = "The correct RAT to connect to is: "

Text = Text + CorrectRAT

#Declaring label confirming that button has been pressed and outputting it onto the user interface

RATSelection = Label(Interface, text=Text)

RATSelection.grid(row=6, column=0, columnspan=2)

#Creating Tkinter window to display user interface and giving it a title

Interface = tk.Tk()

Interface.title("RAT Selection Algorithm")

#Declaring label welcoming the user to the program and outputting it onto the user interface

Welcome = Label(Interface, text="Welcome! Please input the current network attributes so that the correct RAT to connect to can be selected.")

Welcome.grid(row=0, column=0, columnspan=2)

#Declaring label asking the user for the longitude and outputting it onto the user interface

LongitudeLabel = Label(Interface, text="What is the longitude?:")

LongitudeLabel.grid(row=1, column=0)

#Declaring entry box for the user to input longitude into and outputting it onto the user interface

Longitude = Entry(Interface, width=30)

Longitude.grid(row=1, column=1)

#Declaring label asking the user for the latitude and outputting it onto the user interface

LatitudeLabel = Label(Interface, text="What is the latitude?:")

LatitudeLabel.grid(row=2, column=0)

#Declaring entry box for the user to input latitude into and outputting it onto the user interface

Latitude = Entry(Interface, width=30)

Latitude.grid(row=2, column=1)

#Declaring label asking the user for the uplink throughput and outputting it onto the user interface

UplinkLabel = Label(Interface, text="What is the uplink throughput?:")

UplinkLabel.grid(row=3, column=0)

#Declaring entry box for the user to input uplink throughput into and outputting it onto the user interface

Uplink = Entry(Interface, width=30)

Uplink.grid(row=3, column=1)

#Declaring label asking the user for the mobility scenario and outputting it onto the user interface

ScenarioLabel = Label(Interface, text="What mobility scenario is the user in?:")

ScenarioLabel.grid(row=4, column=0)

#Declaring string variable SelectedScenario and providing drop down menu for user to select mobility scenario

SelectedScenario = StringVar()

Scenario = OptionMenu(Interface, SelectedScenario, "Bus", "Car", "Pedestrian", "Static", "Train")

Scenario.grid(row=4, column=1)

#Declaring button for user to press to insert details into the system

InsertDetails = Button(Interface, text="Insert Details", command=SelectCorrectRAT)

InsertDetails.grid(row=5, columnspan=2)

#Running interface

Interface.mainloop()

#Running Tkinter

tk.mainloop()

## Appendix 5: Result Figures Source Code

### Appendix 5.1: ML Model Comparison Figure (Initial Dataset).py

#Importing libraries

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

#Reading results from CSV

Data = pd.read\_csv("ML Model Comparison (Initial Dataset).csv", encoding="latin-1")

#Creating and showing a bar graph

#Measure is shown on the x axis, Score is shown on the y axis, and different bars are shown for each model

Plot = sns.catplot(data=Data, x="Measure", y="Score (1dp)", hue="Model", kind="bar")

for ax in Plot.axes.flat[1:]:

sns.despine(ax=ax, left=True)

for ax in Plot.axes.flat:

ax.set\_xlabel(ax.get\_title())

ax.set\_title("Comparison of AI Algorithm Performance")

ax.margins(x=0.1)

plt.subplots\_adjust(wspace=0, bottom=0.18, left=0.06)

plt.show()

### Appendix 5.2: ML Model Comparison Figure (Validation Dataset).py

#Importing libraries

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

#Reading results from CSV

Data = pd.read\_csv("ML Model Comparison (Validation Dataset).csv", encoding="latin-1")

#Creating and showing a bar graph

#Measure is shown on the x axis, Score is shown on the y axis, and different bars are shown for each model

Plot = sns.catplot(data=Data, x="Measure", y="Score (1dp)", hue="Model", kind="bar")

for ax in Plot.axes.flat[1:]:

sns.despine(ax=ax, left=True)

for ax in Plot.axes.flat:

ax.set\_xlabel(ax.get\_title())

ax.set\_title("Comparison of AI Algorithm Performance")

ax.margins(x=0.1)

plt.subplots\_adjust(wspace=0, bottom=0.18, left=0.06)

plt.show()

### Appendix 5.3: Tree Number Accuracy Comparison.py

#Importing libraries

import matplotlib.pyplot as plt

import numpy as np

#Plotting the initial dataset results for tree number accuracy

#X1 represents tree number, while Y1 represents accuracy

X1 = np.array([1, 5, 10, 25, 50, 100])

Y1 = np.array([96.7, 97.2, 97.2, 97.3, 97.4, 97.4])

plt.scatter(X1, Y1, label="Initial Dataset")

#Plotting the validation dataset results for tree number accuracy

#X2 represents tree number, while Y2 represents accuracy

X2 = np.array([1, 5, 10, 25, 50, 100])

Y2 = np.array([98.9, 99.2, 99.3, 99.3, 99.3, 99.3])

plt.scatter(X2, Y2, label="Validation Dataset")

#Adding a title, x and y axis labels, and a legend

plt.title("Tree Number Accuracy Comparison")

plt.xlabel("Tree Number")

plt.ylabel("Accuracy (1dp)")

plt.legend()

#Showing the figure

plt.show()

### Appendix 5.4: Maximum Depth Accuracy Comparison.py

#Importing libraries

import matplotlib.pyplot as plt

import numpy as np

#Plotting the initial dataset results for maximum depth accuracy

#X1 represents maximum depth, while Y1 represents accuracy

X1 = np.array([1, 5, 10, 15, 20, 27])

Y1 = np.array([76.3, 86.9, 93.6, 96.5, 97.2, 97.4])

plt.scatter(X1, Y1, label="Initial Dataset")

#Plotting the validation dataset results for maximum depth accuracy

#X2 represents maximum depth, while Y2 represents accuracy

X2 = np.array([1, 5, 10, 15, 20, 27])

Y2 = np.array([78, 85.4, 93.9, 97.7, 99, 99.3])

plt.scatter(X2, Y2, label="Validation Dataset")

#Adding a title, x and y axis labels, and a legend

plt.title("Maximum Depth Accuracy Comparison")

plt.xlabel("Maximum Depth")

plt.ylabel("Accuracy (1dp)")

plt.legend()

#Showing the figure

plt.show()