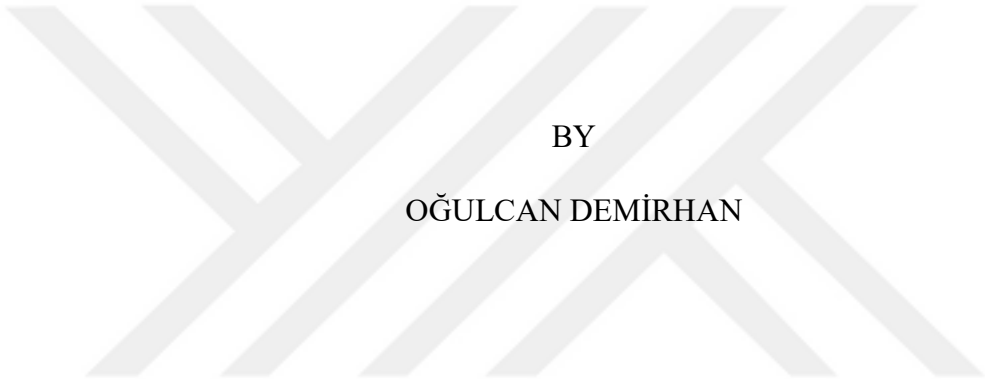


IDENTIFICATION OF THE ABNORMAL FUEL CONSUMPTION IN A
COMMERCIAL FLIGHT BY AN ARTIFICIAL NEURAL NETWORK
SURROGATE MODEL

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY



BY
OĞULCAN DEMİRHAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
AEROSPACE ENGINEERING

SEPTEMBER 2022

Approval of the thesis:

**IDENTIFICATION OF THE ABNORMAL FUEL CONSUMPTION IN A
COMMERCIAL FLIGHT BY AN ARTIFICIAL NEURAL NETWORK
SURROGATE MODEL**

submitted by **OĞULCAN DEMİRHAN** in partial fulfillment of the requirements
for the degree of **Master of Science in Aerospace Engineering, Middle East
Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Serkan Özgen
Head of the Department, **Aerospace Engineering**

Prof. Dr. Utku Kânoğlu
Supervisor, **Aerospace Engineering Dept., METU**

Examining Committee Members:

Asst. Prof. Dr. Ali Türker Kutay
Aerospace Engineering Dept., METU

Prof. Dr. Utku Kânoğlu
Aerospace Engineering Dept., METU

Prof. Dr. Mehmet Zülfü Aşık
Engineering Sciences Dept., METU

Prof. Dr. Işık Hakan Tarman
Mechanical Engineering Dept., METU

Asst. Prof. Dr. Baran Aydın
Civil Engineering Dept., ATU

Date: 02.09.2022



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Oğulcan Demirhan

Signature :

ABSTRACT

IDENTIFICATION OF THE ABNORMAL FUEL CONSUMPTION IN A COMMERCIAL FLIGHT BY AN ARTIFICIAL NEURAL NETWORK SURROGATE MODEL

Demirhan, Oğulcan
Master of Science, Aerospace Engineering
Supervisor : Prof. Dr. Utku Kânoğlu

September 2022, 122 pages

Since the 1980s, numerous fuel consumption modeling studies have been conducted in the literature. Lately, fuel efficiency studies have become more of an issue due to carbon emission and its consequence; global warming. In this study, a feed-forward neural network is modeled, and trained with high-fidelity simulation data (operational flight plans). Later, the actual flight data from Quick Access Recorder (QAR) is used for tuning the hyperparameters of the model. The ten models which have the least loss have been selected and subjected to test with a portion of QAR data. The best model has been selected at the end of the statistical comparison between these ten models. Finally, an identification process has been explained for the flights whose fuel consumption estimation errors exceed the three-sigma boundary. Although this model is built only by five fundamental parameters (takeoff weight, air distance, mean cruise Mach number, altitude parameter, and fuel mileage deviation), the accuracy level of the model is promising when compared to the studies in the literature. Moreover, the study proposes an additional method for the identification of abnormal fuel consumption.

Keywords: Aircraft Fuel Consumption Modeling, Neural Networks, Surrogate Models, Medium-Fidelity Modeling

ÖZ

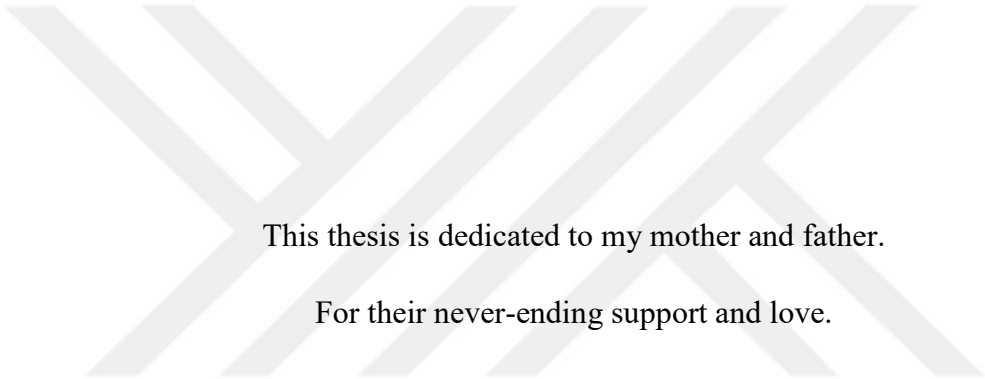
YAPAY BİR SİNİR AĞI VEKİL MODEL İLE TİCARİ UÇUŞTA ANORMAL YAKIT TÜKETİMİNİN BELİRLENMESİ

Demirhan, Oğulcan
Yüksek Lisans, Havacılık ve Uzay Mühendisliği
Tez Yöneticisi: Prof. Dr. Utku Kânoğlu

Eylül 2022, 122 sayfa

1980'lerden bu yana literatürde çok sayıda yakıt tüketimi modelleme çalışması yapılmıştır. Son zamanlarda, karbon emisyonu ve bunun sonucu olan küresel ısınma sonucunda yakıt verimliliği çalışmaları daha çok gündeme gelmiştir. Bu çalışmada, ileri beslemeli bir sinir ağı modellenmiş ve yüksek doğruluklu simülasyon verileriyle (operasyonel uçuş planları) eğitilmiştir. Daha sonra, Quick Access Recorder'dan (QAR) gelen gerçek uçuş verileri, modelin hiperparametrelerini ayarlamak için kullanılmıştır. En az hataya sahip on model seçilerek, QAR verilerinin bir kısmı ile teste tabi tutulmuştur. Bu on model arasındaki istatistiksel karşılaştırma sonunda en iyi model seçilmiştir. Son olarak, yakıt tüketimi tahmin hataları üç sigma sınırını aşan uçuşlar için bir tanımlama süreci açıklanmıştır. Bu model sadece beş temel parametre (kalkış ağırlığı, hava mesafesi, ortalama seyir Mach sayısı, irtifa parametresi ve yakıt kilometre sapması) ile oluşturulmuş olsa da, modelin doğruluk düzeyi literatürdeki çalışmalarla karşılaştırıldığında umut vericidir. Ayrıca, çalışma anormal yakıt tüketiminin belirlenmesi için ek bir yöntem önermektedir.

Anahtar Kelimeler: Uçak Yakıt Tüketimi Modellemesi, Yapay Sinir Ağları, Vekil Model, Orta Duyarlı Modelleme



This thesis is dedicated to my mother and father.

For their never-ending support and love.

ACKNOWLEDGMENTS

The compilation of this study would not be possible without the participation and assistance of my thesis advisor Prof. Dr. Utku Kânoğlu. His contributions are well acknowledged and sincerely appreciated.

Secondly, I would like to express my sincere gratitude to my teachers, mentors, and big brothers Mr. Gökmen Düzgören, Corendon Airlines Assistant Engineering Manager/Flight Operations, and Mr. Arslan Yoldaş Akyürek, Corendon Airlines Operation Control Center Manager.

Last but not least, many thanks to my family and friends who have never-ending support and love for me.

TABLE OF CONTENTS

ABSTRACT.....	v
ÖZ	vi
ACKNOWLEDGMENTS	viii
TABLE OF CONTENTS.....	ix
LIST OF TABLES	xii
LIST OF FIGURES	xiii
LIST OF ABBREVIATIONS.....	xiv
LIST OF SYMBOLS	xvii
1 INTRODUCTION	1
1.1 Purpose of the Study	2
1.2 Proposed Model.....	3
2 LITERATURE REVIEW	5
2.1 Low Fidelity Models	7
2.2 High Fidelity Models	9
2.3 Surrogate Modeling.....	13
2.4 Artificial Neural Networks (ANN)	17
3 DATA ACQUISITION AND REVIEW.....	23
3.1 Data Sources and Acquisition	23
3.1.1 Actual Flight Data.....	24
3.1.2 Operational Flight Plan Data	32
3.2 Database Relationships	38
3.3 Data Validity	41

4	METHODOLOGY	45
4.1	The Data Acquisition and Review Process.....	49
4.1.1	Standardization	53
4.1.2	Train-Validation-Test Datasets	53
4.2	The Computational Setup	55
4.3	Training the Neural Network.....	56
4.3.1	Hyperparameters.....	57
4.3.2	The Optimization Algorithm Selection	59
4.3.3	The Loss Function Selection	61
4.3.4	The Activation Function Selection.....	62
4.3.5	Training and Validation.....	63
4.4	Testing the Model and Accuracy Check.....	65
4.5	Identification of Abnormal Fuel Consumption.....	66
4.5.1	Three-Sigma Boundary	66
5	RESULTS.....	69
5.1	The Best Model Results.....	69
5.2	Identification of Abnormal Fuel Consumption.....	71
6	CONCLUSIONS AND DISCUSSION.....	77
6.1	Discussion.....	77
6.2	Future Work.....	79
	REFERENCES	81
7	APPENDICES	87
A.	Letter of Consent for Use of Company Data	87
B.	Example of an Operational Flight Plan.....	89

C.	Parsing OFP Data	91
D.	Parsing OFP Waypoint Data	102
E.	Gathering Altitude Values.....	106
F.	Altitude Parametrization	109
G.	Gathering Master DB	111
H.	Train-Validate Multiple Models.....	113
I.	Testing the Models and Saving the Results	117
J.	Statistical Accuracy Check	119
K.	Training and Testing Best Model Only.....	120

LIST OF TABLES

TABLES

Table 3.1 Fuel Related Parameters included in the database	30
Table 3.2 Parsed OFP parameters	37
Table 3.3 Input and Output Parameters	43
Table 4.1 Input and Output Parameters	49
Table 4.2 Sample Standardized Parameters	53
Table 4.3 Hyperparameters and their spectrums	58
Table 4.4 Sample of the Logbook	64
Table 5.1 Ten best models.....	69
Table 5.2 Results of accuracy check	70
Table 5.3 Flights to be investigated.....	73
Table 6.1 Existing Studies.....	78

LIST OF FIGURES

FIGURES

Figure 3.1. DFDAU system block diagram	26
Figure 3.2. Wireless Quick Access Recorder (WQAR) system block diagram	27
Figure 3.3. Sample Raw DFDAU data	28
Figure 3.4. FDM data structure	32
Figure 3.5. Altitude profile in RVSM airspace	39
Figure 3.6. Data acquisition and relationship scheme	40
Figure 4.1. Study methodology flowchart	48
Figure 4.2. Sample flight profile and parametrization method A	50
Figure 4.3. Sample flight profile and parametrization method B	51
Figure 4.4. True airspeed (TAS) vs ground speed (GS)	52
Figure 4.5. Data structure.....	54
Figure 4.6. Typical Feedforward Neural Network.....	57
Figure 4.7. ReLU graphical interpretation	63
Figure 5.1. Best model training and validation loss.....	71
Figure 5.2. Histogram of Percentage Error	72
Figure 5.3. Flight Profile of Flight 1	73
Figure 5.4. Flight 2 EFF log.....	74
Figure 5.5. Flight profile of Flight 3	75
Figure 5.6. The flight path of the landing phase	75

LIST OF ABBREVIATIONS

ABBREVIATIONS

ACMS	Aircraft Condition Monitoring System
ANN	Artificial Neural Network
APM	Aircraft Performance Monitoring
ATC	Air Traffic Controller
ATM	Air Traffic Management
ATS	Air Traffic Services
BADA	Base of Aircraft Data
CF	Compact Flash
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
CORSIA	Carbon Offsetting and Reduction Scheme for International Aviation
DAP	Data Acquisition Processor
DFDAU	Digital Flight Data Acquisition Unit
DSIRF	Digital Standard Interface Record Format
EASA	the European Aviation Safety Agency
EFB	Electronic Flight Bag
EFF	Electronic Flight Folder
ETOPS	Extended-range Twin-engine Operation Performance Standards
EU ETS	European Union Emission Trading Scheme

FAA	Federal Aviation Administration
FAF	Final Approach Fix
FCO	Fuel Consumption Optimization
FDM	Flight Data Monitoring
FDR	Flight Data Recorder
FNN	Feed-forward Neural Network
FM	Fuel Mileage
FOQA	Flight Operations Quality Assurance
FPPM	Flight Planning and Performance Manual
GD	Gradient Descent
GS	Ground Speed
ICAO	International Civil Aviation Organization
IFR	Instrumental Flight Rules
LMBP	Levenberg-Marquardt backpropagation
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MBE	Mean Bias Error
ML	Machine Learning
MLP	Multilayer Perceptron
MRE	Mean Relative Error
MSE	Mean Square Error
NASA	National Aeronautics and Space Administration

NN	Neural Network
NOTAM	Notice to Airmen
OFP	Operational Flight Plan
PET	Performance Engineering's Tool
PE	Percentage Error
QAR	Quick Access Recorder
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RVSM	Reduced Vertical Separation Minima
SAGE	System for Assessing Aviation's Global Emissions
SGD	Stochastic Gradient Descent
SID	Standard Instrumental Departure
SMM	Safety Management Manual
SMS	Safety Management Systems
SOP	Standard Operating Procedures
STAR	Standard Arrival Route
TAS	True Airspeed
TOW	Takeoff Weight
VFR	Visual Flight Rules
WQAR	Wireless Quick Access Recorder

LIST OF SYMBOLS

SYMBOLS

x^+	Rectified Linear Unit
x_{std}^i	Standard Scaler
ΔFM	Fuel Mileage Deviation
$\Delta F_{N\ Req'd}$	Thrust Required Deviation
ΔFF	Fuel Flow Deviation
$\Delta N1_{Req'd}$	N1 Required
$Param_A$	Altitude Parameter A
$Param_B$	Altitude Parameter B
R^2	Coefficient of Determination
σ	Standard Deviation
$\bar{y}$	Mean Percentage Error

CHAPTER 1

INTRODUCTION

After the dramatic increase in fuel prices in the period between 2005-2008 and the deployment of carbon emission offsetting programs such as European Union-Emission Trading Scheme (EU-ETS) and International Civil Aviation Organization-Carbon Offsetting and Reduction Scheme for International Aviation (ICAO-CORSIA), airlines, authorities, and researchers focused on and analyzed the “Aircraft Fuel Efficiency” phenomena more intensively. According to Singh & Sharma [1], the number of publications on Fuel Consumption Optimization (FCO) significantly increased by almost 110% after 2000.

In order to optimize fuel efficiency, all the independent variables that affect the efficiency should be determined based on the nature of the problem and their positive or negative contribution to the efficiency should be analyzed in detail. Therefore, it is crucial to model a fuel consumption estimation to measure the efficiency of the case study. One of the common FCO concerns is about the efficiency of terminal and en-route airspaces. Especially, for the busy airports near big cities, the airspace of an aerodrome should be designated with a special concern regarding fuel consumption and environmental pollution. To provide better-optimized trajectories and less fuel consumption and carbon emission, the designator should pay regard to the fuel efficiency by optimizing the fuel consumption in that airspace [2]. Hence, fuel consumption data and its modeling constitute high importance to the aviation authorities and airspace designators. On the other hand, airline operators are highly interested in the fuel efficiency concept because the fuel costs consist of %10-12 of the total expenses of an airline [3]. Because even a small percentage of saving for each flight can make a significant difference, airlines deploy

“Fuel Efficiency Management Programs”, designates their Standard Operating Procedures (SOPs) based on fuel efficiency, and make use of the 3rd party software that helps airlines to save fuel. In this manner, fuel consumption modeling lends assistance to the airlines to establish the most efficient SOP, identify and eliminate non-normal fuel consumption and estimate the future fuel demand for the operation.

1.1 Purpose of the Study

The aircraft fuel consumption estimation models have evolved for the last forty years in the literature. At first, analytical estimations based on energy conservation law have been deployed with various assumptions to simplify the mathematical representation of the model. Even though these models provide practical benefits to air traffic controllers (ATC) and aircraft operators, more precise and easy-to-use models were required.

Later, aircraft data acquisition methods became advanced, and the modeling methodologies benefitted from this data. Low-fidelity models have been developed using linear or nonlinear regressions. These models were easy-to-use, but the models' accuracies were relatively low. Additionally, the standard air operations necessitated more exact fuel burn calculations for safer operations. As a consequence, the regulators and manufacturers developed high-fidelity, detailed models that do not need simplifying assumptions thanks to the operational data. Solely, these models require too many parameters to estimate the fuel consumption or fuel flow. The parameters are supposed to be accessible and deployed into the model each time with an expensive calculation cost. On the other hand, since these models are difficult to build considerably, they are mostly commercial tools. Therefore, the researchers are not able to access these models easily.

Lately, surrogate models mimicking high-fidelity simulations by a data-driven approach have been advanced to achieve better accuracy without compromising calculation costs. The surrogate models aim to achieve to

approximate the behavior of the high-fidelity simulations as closely as possible. On the other hand, artificial neural networks have been employed for aircraft fuel consumption modeling with different scopes, datasets, and methodologies. In one way or another, researchers targeted cost-effective calculations with high accuracy.

On the contrary, this study aimed to achieve a superior model which has the below advantages by harmonizing methodologies in the literature;

- Accurate and cost-effective fuel consumption modeling with a minimum number of parameters,
- Easy to develop,
- Flight phase segmentation is not necessary,
- Generic, effective to aircraft type.
- Identifies abnormal fuel burn.

1.2 Proposed Model

After a literature survey about existing fuel consumption models, two main data sources will be introduced along with their acquisition process. One of these is the “Operational Flight Plan (OFP)” parameters gathered from the flight plan excerpts of the daily operation of Corendon Airlines. These plans comprise crucial information about the flight such as trip fuel, altitude, and waypoints. The fuel consumption information is calculated by the high-fidelity simulations integrated into the tool generating the flight plan. On the other side, the actual flight data is utilized. The data is obtained from Corendon Airline’s Flight Data Monitoring (FDM) program. In the end, a master database has been generated, by defining the relationship between the data. The database consists of 12765 flights operated with Boeing 737-800s in 2019. The details about the data sources and their acquisition processes are given in Chapter 3.

In Chapter 4, the proposed model is explained in detail. It is intended to develop a neural network model trained by OFP simulations, tuned, and tested with

actual flight data. The feed-forward neural network is utilized because of its superiority in these kinds of tasks. Hyperparameters tuning strategy and the philosophy behind optimization algorithm, loss function, and activation function selection are explained. In addition, two altitude parametrization methodologies are proposed to achieve the altitude parameter that represents all segments for that flight. 20% of the actual flight data is utilized as validation data to tune hyperparameters. The best ten models are selected according to minimum validation loss. This means that hyperparameters are tuned and the models are ready to test. The rest of the actual flight data (80%) is used to test models. The best model that has the lowest statistical error is selected by utilizing test results. Finally, a flight identification methodology is defined through percentage error. The standard deviation of the percentage error over the test dataset is calculated and multiplied by three. An upper and a lower boundary are set. The flights that exceed these three-sigma boundaries are defined as flights with abnormal fuel consumption. Three random samples exceeding these boundaries are selected and analyzed in detail.

CHAPTER 2

LITERATURE REVIEW

In 2002, Stolzer [4] conducted a literature review research which reveals that fuel efficiency and conservation studies mostly focus on engineering rather than the operational view. Then, the operational studies that led to efficient decisions and strategies made rapid strides along with the developments in engineering, the advancements in the aircraft and airport data flow structure, statistical analyses such as neural networks, and an unfortunate spike in fuel prices.

It might be claimed that modern civil aviation and operation-oriented fuel consumption modeling research are started in 1980 by Collins [5]. In the study, Collins [5] proposed a new fuel consumption estimation method to be used by air traffic controllers (ATC). The fundamental concept behind the study was the energy balance of the aircraft in terms of thrust and drag,

$$f(E_T) - f(E_D) = f(\Delta KE) + f(\Delta PE) \quad (2.1)$$

where $f(E_T)$, $f(E_D)$, $f(\Delta KE)$, $f(\Delta PE)$ are thrust energy, drag energy, kinetic energy, and potential energy respectively.

In the study, variables such as altitude (in means of density), flight time, velocity, and aircraft weight were used as input to the mathematical model, and fuel flow was calculated. Collins's [5] strategy that optimizes fuel burn was implemented on the computer system used by ATC and resulted in a significant amount of fuel savings.

Another example of fuel efficiency research before 2000 is by Gibbons and McDonald [6]. This research investigated the success of the linear regression model over the fuel economy of the 1988-1990 model car and truck dataset. Although the

study was implemented on motor vehicles, the statistical effort of fuel consumption modeling behind it was the same with the aircraft. The authors estimated the effect of technology used in the vehicle on fuel efficiency with the following methods;

Ordinary Least Squares (OLS)

$$Y = X\beta + \epsilon \quad (2.2)$$

The above linear model represents the problem where Y is the observation matrix of the dependent variable, X is the fixed matrix of input variables, β is the vector of the unknown regression coefficient. The OLS estimator ($\hat{\beta}$) minimizes the residual sum of the square of errors and given by,

$$\hat{\beta} = (X'X)^{-1}X'Y \quad (2.3)$$

where X' is the inverse matrix of X .

Equality-Constrained Least Squares (ECLS)

In the cases, additional conditions about the coefficient vector are employed, such as $R\beta = r$, and the residual sum of squares subject to the constraint ($R\beta = r$) was minimized. By classical Lagrangian methods whose mathematical details are given in Gibbons and McDonald [6], the below estimator was obtained,

$$E(b^*) = \beta + (X'X)^{-1}R'[R(X'X)^{-1}R']^{-1}(r - R\beta) \quad (2.4)$$

where r, R, R' are the vector of known elements, known prior information matrix that expresses the structure of the information on the individual parameters, and inverse of R matrix, respectively.

Inequality-Constrained Least Squares (ICLS)

This approach is based on the idea that the linear combination of variables or the information is restricted by an upper and a lower bound. At this time, minimizing the residual sum of squares subject to the bound ($R\beta > r$) resulted in a quadratic equation problem. The estimator was formulated as follows,

$$b^+ = \hat{\beta} + (X'X)^{-1}R'\lambda^+ \quad (2.5)$$

where $\hat{\beta}$ is OLS estimator and λ^+ is a vector of multipliers associated with the inequality constraint.

The ICLS method was implemented on a problem and compared with the OLS where R^2 values were .955 for OLS and .957 for ICLS. The author concluded that the ECLS and ICLS models exhibit successful performance and could be easily deployed in different situations for different problem structures.

Although the pre-2000s studies mostly focused on the analytical and statistical approaches, the modern fuel consumption modeling strategies are built in the fashion of high-fidelity and computerized models. However, the computational expenses are tried to be lowered in means of accessibility and effort-result trade.

In this manner, the modern fuel consumption modeling strategies can be divided into four main categories;

- 1) Low Fidelity Models
- 2) High Fidelity Models
- 3) Surrogate Models
- 4) Artificial Neural Network Models

2.1 Low Fidelity Models

In literature, numerous studies where the relation between measurable and/or computable parameters and fuel conservation is modeled through analytical equations exist. The goal of the method is to examine the impact of a single parameter on the target parameter. The mathematical relation used in the model is mostly based upon the physics of the study and technical expertise. Besides, this method introduces lower computational expenses when compared with the high-fidelity physical models because of simplifying assumptions on the problem with the cost of model accuracy.

In 2011, Qian-Qian, Jing-Jie, and Jian-Kang [7], presented their research on the relation between the common parameters of commercial flight operation and fuel efficiency. The fuel efficiency was obtained in the following way;

$$\text{Fuel Efficiency} = \text{Actual Fuel Consumption} / \text{Total Transport Turnover}$$

where “Total Transport Turnover” is the total income for the respective flight. After examining multiple linear regression models, they decided on the “Quadratic Polynomial Regression Model” and “Third-Order Polynomial Regression Model” which achieves the most successful fit over the database.

The quadratic polynomial equation was modeled as below;

$$y = (a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n)^2 + c + u \quad (2.6)$$

where a_n s are weight constants, x_n s are the independent parameters, c is the bias and finally, u is noise. The coefficients were estimated by least square methods after arranging the model formula with two separate methods. The first method was transforming the formula into multiple linear equations with high dimensional mapping while the second was normalizing the formula.

Finally, the regression model was examined with an existing airline transportation database for a year of the Xin-Beijing route with A319 aircraft. With the help of ICAO Circular No. 303 “Operational Opportunities to Minimize Fuel Use and Reduce Emissions,” the variables that have an impact on fuel efficiency were selected as; payload rate, average speed, take-off fuel consumption, and roll time (refer to the paper for the parameter details). After the results were obtained ($R^2 = 0.782$), the goodness of fit, f , and t -tests were applied to the model. They concluded the model successfully estimates fuel efficiency and helps to optimize energy-saving measures.

On the other hand, Khadilkar and Balakrishnan [8], estimated the aircraft taxi fuel consumption with a linear regression model in which the flight data recorder (FDR) information is used. Although the parameters given in the ICAO fuel burn index related to the taxi were investigated, the correction of model assumptions in

the ICAO index was done by comparing the ICAO and FDR databases. Additionally, the parameters such as the number of acceleration events were included in the model because of their significant impact. They have built two different models;

- 1- Taxi time and number of stops/turns as independent variables

$$\frac{f}{\sqrt{T_{amb}}} = a_1 + b_1 t + c_1 n_s + d_1 n_t \quad (2.7)$$

where f is the fuel consumed, T_{amb} is ambient temperature, t is the taxi time, n_s is the number of stops, n_t is the number of turns made by the aircraft during taxi, and a, b, c, d are the constants to be determined by the model.

- 2- Taxi time and number of acceleration events as independent variables

Since it was seen that the fuel flow during the acceleration start from a standstill is much greater than the gradual acceleration, the model has evolved as below;

$$\frac{f}{\sqrt{T_{amb}}} = a_1 + b_1 t + c_1 n_a \quad (2.8)$$

where all variables are the same as the previous model and n_a is the number of acceleration events.

The authors concluded that although the main drive behind the taxi fuel burn is the taxi time, the proposed model provides more realistic results rather than the ICAO indices. Moreover, despite the physical models, this study includes pilot behavior and thrust configuration during the taxi stops and turns.

2.2 High Fidelity Models

A vast amount of aircraft fuel studies focus on the physical modeling of the aircraft in different phases of flight. Starting from Collins' [5] research, detailed mathematical modeling of the nature of the problem constituted great importance in means of strategic action plans such as better routes, terminal procedures, speed and altitude schedules, and pilot behavior. Additionally, it helps to identify a non-normal

situation caused by an unusual fuel burn. On the other hand, manufacturers build a physical model for their aircraft to estimate the fuel burn of an operation through empirical data.

Even though the individuals put a significant amount of work into the kinematic models, the late studies do not mostly try to develop a model from the start. The major reason behind this is that the national or international regulators already developed and continuously improve their high-fidelity models. While some of them are commercial, the majority of them are free of charge for scientists and researchers. Below are the most common performance/fuel consumption models used in the literature;

- Eurocontrol – Base of Aircraft Data (BADA),
- Federal Aviation Agency (FAA) - System for Assessing Aviation's Global Emissions (SAGE).

Before giving brief information about these softwares, it is important to note that there are various flight planning softwares (mostly for commercial use) in the market, which could be considered a high-fidelity fuel burn model. They can calculate fuel consumption concerning the route, payload, and realistic weather forecasts and observations. The detailed information about the output has been given in Chapter 3.1.2. Operational Flight Plan Data.

Eurocontrol BADA is an aircraft performance tool that aims to accurately model the aircraft behavior under nominal operation. According to the Eurocontrol design report [9], the modeling process can be separated into three phases;

- Data preparation,
- Identification process,
- Validation process.

The data preparation section defines the stored aircraft data, explains how to manage the data, and finally how to export the necessary data from the database. In the identification process, a generalized model set where aerodynamic drag, engine

thrust, and fuel consumption are formulized and included in the model. After that, the coefficients are estimated. Although the model includes nonlinear, coupled multivariate parameters, it was realized that the model could be separated into several linear sub-problem by iteration. It is important to note that the model is capable of generating trajectories utilizing altitude and speed. In the end, the coefficients on the sub-models were fitted and syntax, cross, and validation checks were done.

Another high-fidelity computer model SAGE has been introduced by the FAA Office of Environment and Energy. Besides the fuel burn computations, SAGE can estimate the carbon emission of the aircraft. Since the concept of the software is similar to the BADA, detailed information about the model and its validation process will not be provided. However, one can refer to the SAGE validation report [10].

Because mentioned models have a significant amount of success for modeling the fuel burn and emission and require type exclusive data, most researchers do not develop a high-fidelity model from the start. Instead, they mostly investigate and correct the existing models or generate a consumption model from the time-dependent position trajectory of the aircraft again using existing models. Below, some of the existing studies are summarized.

Zhou [11] attempted to calculate the overall fuel consumption and carbon emission of the Continental United States. It was claimed that the estimation of the overall fuel burn and carbon emission would help to investigate the energy efficiency and environmental effect of the carbon emission. To achieve this objective, the author developed a model using multiple existing models. Below databases are governed to estimate the model outputs;

- Airport data (Latitude and Longitude Coordinates and Elevation) has been parsed with respect to the Origin-Destination pair.
- The flight path profile has been created by using the BADA equations and coefficients.

- FAA Enhanced Traffic Management System (ETMS) has been used to obtain the waypoints and cruise flight levels.
- The Performance Data Analysis and Reporting System (PDARS) has been used to detect detour factors of the terminal.
- The FAA Aviation System Performance Metrics (ASPM) has been used to estimate the taxi-in and taxi-out durations.
- The National Centers for Environmental Prediction (NCEP) and the National Center for Atmospheric Research (NCAR) datasets have been used to detect the wind effects.

Finally, generated flight profile exposed to wind state and terminal area detour analysis. Respectively, en-route fuel consumption and emission were calculated. To estimate the fuel consumption and emission on the ground, taxi durations were identified. After, the total consumption and emission were determined. In conclusion, the author stated the model validation resulted in a success and the model is ready to be implemented on any other software as a module.

In 2011, Chatterji [12] put an effort to estimate fuel burn with tracking data. During the development of the model, the BADA was used as the baseline. Diversely, his model was derived from the equation of motion under the point mass assumption. Moreover, similar to Zhou's study, wind and aircraft states were considered during the modeling process.

The estimation process starts with the aircraft's weight. After the track and wind data are obtained, the aircraft state was identified by a Proportional-Integral-Derivative (PID) controller-based estimator which estimates the vertical acceleration and altitude rate. The same methodology was implemented on the latitude and longitude measurements that provide angles, angular rates, and angular accelerations estimations. With respect to the wind effects, required lift and corresponding drag were calculated. Later, applied thrust and corresponding fuel flow rate were computed concerning the drag. Moreover, the mathematical details of the model could be found in [12].

Chatterji [12] validates the model with flight test data provided by FAA. It is stated that the fuel consumption could be estimated with a 1% error. Finally, the model was implemented on radar-based position information, and a 5.4% deviation from the actual value was obtained.

Very similar to Chatterji's [12] research, Oaks and Paglione [13] also developed a model based on the BADA and National Weather Service data using a 4-D trajectory. The model was validated based on actual flight data. The validation process showed that the model is able to estimate the fuel burn with less than 2% error for six hours of flight.

2.3 Surrogate Modeling

Model fidelity can be described as the proximity of the model to imitate the real problem in case of disturbances from boundaries or internal changes. It can be classified as low, medium, and high fidelity. The nature of simulation requires different fidelity levels constrained by the model cost. In other words, higher fidelity means higher computational expense. However, a simulation might be constructed by various fidelity models. It is called the medium-fidelity simulation which contains a mixture of models with different fidelity levels.

Yamazaki & Mavriplis [14] states that,

Surrogate model approaches have attracted increased attention recently in aerospace engineering because they offer substantial benefits for design optimization, aerodynamic database construction, and uncertainty analysis.

Yamazaki & Mavriplis [14] stated the goal behind the surrogate modeling is to reproduce the costly solutions such as high-fidelity computational fluid dynamics (CFD) simulations by using an analytic regression model. The model is built by outcome samples of the high-fidelity model. According to the authors, most of the surrogate models in the aerospace engineering field have been developed based on

least square polynomial (LSP), multi-layer perceptron (MLP), Kriging, or radial basis function (RBF). However, the mathematical details of the surrogate model proposed in this paper are not within the scope of this thesis. Therefore, the most relevant papers to the thesis scope have been selected and explained in detail.

Yanto & Liem [15] proposed a fuel burn model that combines the low-fidelity analytical model and aircraft performance data. Besides, the BADA trajectory simulations were integrated into the model to provide more accuracy, especially in climb and cruise segments. They evaluated more than 40 aircraft types in which the data should be clustered.

The study aimed to develop a linear regression model that can easily calculate the fuel burn based on payload and range while maintaining the model accuracy similar to the high fidelity models. To achieve this through the range of different types of aircraft, the below steps were followed;

1) Obtaining flight mission database:

A fuel burn database for 40 aircraft has been generated based on “The Bureau of Transportation Statistics (BTS) flight missions’ database 2015,” which is publicly available. For each aircraft type, the type-specific information has been obtained from the manufacturer’s publications.

2) Generating fuel burn inventories for each flight:

Each flight has been fitted into a flight profile, and the fuel burn computation procedure has been followed for each segment (takeoff, climb, cruise, descent, and landing). A flowchart has been introduced to calculate the segment fuel burns which are separated for the long and short-haul flights. Breguet range equation has been used as the baseline equation to compute the fuel burn while the BADA model corrects the low-fidelity model with fuel increment & decrement factor and decides climb and descent ranges. Refer to [15] for the mathematical model details.

3) Deriving linear regression model for each aircraft type:

The linear model has been expressed as below;

$$W_{ij} = \alpha_j R_{ij} + \beta_j P_{ij} + \gamma_j + \epsilon_{ij} \quad (2.9)$$

where i and j indices express flight and aircraft type respectively. α_j , β_j , γ_j are regression coefficients for that aircraft type. W_{ij} indicates the fuel burn and ϵ_{ij} is error-term. Last, R_{ij} and P_{ij} are range and payload variables, respectively. The estimation has been done with the least-squares method.

After that, two separate validation methods have been followed. The first one is “individual-sampling” validation, which compares the 10 sets of 100 randomly selected samples to the estimation. The standard error has been computed based on the standard deviation of the percentage error data set. Later, the aggregate fuel burn computation has been done on the same data set. The error between the estimation and the data has been summed compared with the aggregate fuel burn for each aircraft type.

To conclude, the authors stated they achieved to develop a medium-fidelity model that reduces the computational costs by $O(10^4)$. Type-specific linear regressions have been obtained with the total aggregate fuel burn error less than 1%. Along with that, the model resulted in a significant amount of speed up.

Secondly, the study of Ryerson, Hansen & Bonn [16], could be classified as a medium-fidelity model. The authors aimed to identify the impact of Air Traffic Management technologies in means of schedule padding, airborne delay, and departure delay. To achieve this, they developed a model with the help of three different data sources;

1) Flight Data from a major U.S. Carrier

This is the operational data between 12-29 November 2010, which includes the origin-destination, actual take-off weight, scheduled and actual airborne time, and departure time. Additionally, the actual fuel consumption of the aircraft has been

reported in the data. Through this data, the difference between actual and real airborne and departure times could be computed as delays.

2) Data from FAA NASPAC Model

National Airspace System Performance Analysis Capability (NASPAC) is an FAA flight model that contains a trajectory interpolator and fuel estimator. The estimator is based on the BADA fuel burn estimation tool. The authors used the model to estimate the nominal cruise fuel burn for a flight. However, the model is only able to use the last filed flight plan as flight information, therefore the take-off weight may vary from the actual value. The difference between the actual and scheduled value has been used as an input variable to the model.

3) Data from FAA ASPM Database

The weather data of the departure and arrival airports during the operation has been obtained from the FAA Aviation System Performance Metrics ASPM database. According to the data, the operation is classified as instrumental meteorological conditions (IMC) or visual meteorological conditions (VMC).

To conclude, a fuel consumption model has been developed using the NASPAC's nominal fuel burn (high-fidelity simulation output) and deviation of actual values compared to the schedule in means of airborne and departure delay, take-off weight. Moreover, the weather effects on the departure and arrival airports have been included, and finally, an analytical model has been proposed. After the coefficients have been estimated, it was revealed that operational performance parameters are responsible for the 10% of fuel consumption. The importance of the potential Air Traffic Management (ATM) improvements has been referred to.

2.4 Artificial Neural Networks (ANN)

There are several studies in the literature on fuel consumption or fuel flow calculation with machine learning models. In these studies, different data sources have been used to train and test the different algorithms to predict fuel flow or consumption in different flight phases or for the entire flight. It is important to understand and digest the present studies in means of use scenarios and vulnerable points.

In 1997, two Master theses were presented studies using ANN in the Civil Engineering Department of Virginia Polytechnic Institute and State University by Schilling [17] and Cheung [18]. Schilling [17] developed a neural network model trained by outputs of Collins' [5] mathematical model. The purpose of the study was to develop a neural network model to map the outputs of the energy balance model [5] and optimize the neural network (NN) model. The model was intended to be used by air transportation shareholders in means of accurate, cost-effective, generalized, and fast fuel burn calculations. The feed-forward (2 layers, 7 neurons) model was designed with "tansig" and "purelin" transfer (activation) functions and "Levenberg-Marquardt Backpropagation (LMBP)" (enhanced backpropagation) gradient descent learning rule in Matlab®. Afterward, the author compared the outputs of the neural network with FAA Airport and Airspace Simulation Model (SIMMOD) outputs that has developed by Collins [5]. The model achieved 38% faster calculation with the same accuracy rate only with two input parameters; velocity and altitude.

Cheung [18] also developed a neural network model trained by aircraft performance data in flight manuals. The author aimed to examine the existing fuel consumption models, accurately design a NN model, and justify the model by means of an alternate method of fuel consumption estimations. The main difference between the studies [17] and [18] is the training sources. Additionally, Cheung [18] split the flight into phases and designed sub-models for each flight phase. Therefore, the input and output parameters are particular for each flight segment. However, the programming medium (Matlab®) to design the model is the same with [17]. The

author used the LMBP gradient descent learning rule and “logsig-tansig-purelin” activation functions to obtain the best performance model. After that, the author performed justification with different methods such as normal distribution, mean and standard deviation, histogram, and t-test. It is concluded that the proposed neural network model estimated fuel consumption at the confidence level of 5% when compared to the flight manual’s data.

A similar study in the literature is performed by Trani et al. [19] in 2004. The purpose of the study is outlined below;

- To look into other options for fuel consumption modeling in fast-time simulation systems,
- To construct an accurate fuel consumption model using information from the aircraft performance flight manual, as well as neural network methods,
- To see if neural networks are a suitable alternative for calculating fuel usage in applications.

The performance charts of Fokker F-100’s flight manuals have been used as the data source. Matlab® Neural Network Toolbox has been employed for the development and training of the model. Again, LMBP learning method and transfer functions of logsig-tansig-purelin have been used. As a result, they stated the developed model is an alternative method to estimate fuel consumption and the neural network models can be implemented in high fidelity simulations exist in the literature such as SIMMOD, Reorganized ATC Mathematical Simulator (RAMS), Total Airspace & Airport Modeller (TAAM). Additionally, the neural network model has better accuracy than the analytical model BADA as claimed by the authors.

The feed-forward neural network has been modeled for the studies [17], [18] and [19], and accurate results have been obtained. However, using the energy balance equation or flight manual data as the data source has its positives and negatives. Although the energy balance equation is more generalized, it is very

complicated to map the scalar values and requires a bunch of assumptions. On the other side, the flight manual data is vulnerable to the disturbances such as pilot techniques, and non-normal weather operations and provides information for specific cases. Additionally, the optimizer and activation functions in these studies are outdated.

In recent studies, the objectives of the fuel consumption models have not evolved too much even though the technical utilities have been developed. For instance, Baumann and Klingauf [20] indicated the fuel consumption models in the literature use data in highly aggregated form to make statistical estimations or physical models. The author criticized the previous studies because their data is either the book values from manuals or the data from the performance monitoring systems. In the paper, flight operational data has been employed to build two time-series models to estimate the fuel flow in-flight phases; neural networks and decision trees. The data is obtained from the National Aeronautics and Space Administration's (NASA) public database. Multiple parameters such as air data (i.e. temperature, pressure, altitude, climb/sink rate, angle of attack) engine parameters (i.e. engine speed, exhaust gas temperature, engine number and cycle, oil pressure/temperature, fuel flow, vibration), control inputs (i.e. control surface positions, inputs in the flight management system), navigation data (i.e. latitude and longitude, course, aircraft orientation, distance to waypoint, acceleration), environmental data (i.e. wind direction/speed) and system parameters (brake pressure, warnings, errors and status parameters) have been employed during the training process. Out of 176 parameters, the best-correlated 30 parameters have been selected based on a Spearman rank correlation coefficient. A feed-forward neural network with a multi-layer perceptron model has been developed. Since it minimizes the errors and weights of quadratic equations, Levenberg-Marquardt optimization has been used with a nonlinear sigmoid activation function. Mean absolute error (MAE), Mean relative error (MRE), mean squared error (MSE), and coefficient of determination (R^2) have been used to evaluate the model quality and calculation results. Finally, the fuel flow for the entire flight and cruise was only calculated

separately from the models. The author concluded that neural networks and decision trees can be easily applied to the flight operational data and the NN model shows better performance when compared to the decision tree and a quarter of the parameters is enough to achieve the same accuracy on models.

Another study in the field is Baklacioglu's [21] study which presents a neural network fuel flow model of transport aircraft. The model comprises a backpropagation optimized model for climb and cruise and Levenberg-Marquardt optimized descent phase. Real flight data of Pegasus Airline's 737-800 fleet in 2009 has been utilized including Mach number, flight time, engine speeds, and fuel flow rate. The author concluded the developed genetic algorithm model presents better results than the previous studies.

Finally, Uzun, Demirezen & Inalhan [22] target to improve the data-driven models by completing data gaps in the flight regimes with a physics-guided approach. They stated that conventional machine learning models are limited by the flight envelope of the training data. Therefore, artificial datasets have been created by physical equations as a criterion. Later, a neural network has been trained by labeled operational data and unlabeled physics-guided data. Hence the framework can produce the physically-consistent unseen and untrained output of fuel flow. The results show that the deep learning model predicts fuel consumption in unseen flight regimes that are consistent with BADA calculations. They concluded including key physical principles in the training/learning phase of data-driven models improves accuracy and explainability and the ability of the developed models to generalize.

Even though the existing fuel consumption modeling studies in the literature vary based on the modeling methodology, data source, and use-cases, the models are either excessively time-consuming because high-fidelity simulations are employed or they have lower accuracy because numerous assumptions have been done to simplify the model and achieve cost-effective calculations. In this study, a surrogate model has been proposed to overcome the above dilemma. The proposed model uses high-fidelity simulations as the model's training dataset and the trained model is

validated and tested with the actual flight data. As opposed to the existing surrogate models, the proposed model does not separate the flight into its phases and needs only five parameters as the input to achieve the same or better accuracy than the existing models. Additionally, it helps to identify the abnormal fuel consumption in a flight by testing the model with actual flight data.





CHAPTER 3

DATA ACQUISITION AND REVIEW

There are two main data sources in this study. The first one is the actual flight data obtained from the Quick Access Recorder (QAR) of aircraft. Second is flight plans prepared by a licensed dispatcher before every flight for the operation. These sources are processed, parsed, and turned into a database to be used for different purposes in an airline.

It is important to digest the dynamics of these data sources. Thus, the potential error sources can be determined and eliminated in the future. Furthermore, because two data sources will be merged, compared, and analyzed for each flight, the merged database will be the key aspect of this study.

The collected data is a property of Corendon Airlines. Corendon Airlines has given permission to the author to use data to conduct the research only in the scope of this study. Unauthorized use, copying, distribution or publication of data by third parties is prohibited. Moreover, data confidentiality has been ensured and the data de-identification process has been carried out. Corendon Airlines has issued the Letter of Consent, given in Appendix A, for the use of its data by the researcher as required by the study.

3.1 Data Sources and Acquisition

Data sources and acquisition process are explained in the following subsections.

3.1.1 Actual Flight Data

In the aviation industry, flight data monitoring and analyzing procedures are getting more widespread each day. Accident-incident investigations, safety assessment procedures, determining best practices, and many other benefits of the flight data monitoring concept led people to focus on this area more intensively.

Firstly, the industry introduced a concept called “Flight Operational Quality Assessment (FOQA).” The target was to obtain and analyze the actual flight data to improve overall operational performance and safety. According to Stolzer [4], in June 1995, FAA funded \$5.5 million to the demonstration project that aims to promote to initiate the development of organizational strategies for FOQA programs because the implementation of FOQA was the most promising initiative to reduce the increasing trend of aircraft accidents. After the ten U.S airlines implemented the program, results and findings were documented. Many airlines reported that they notice the exceedance and unsafe actions on different phases of flight easier than before. Hence, the corrective action was being decided appropriately and in time.

In 2008, International Civil Aviation Organization (ICAO) Annex 6 was amended [23] to introduce “Safety Management Systems (SMS)”. In Part I, Operation of Aircraft, the following statements are given;

3.3.2 The operator of an aeroplane of a maximum certificated take-off mass in excess of 27 000 kg shall establish and maintain a flight data analysis programme as part of its safety management system.

As of 7 November 2019, the essential of data protection has been amended as below;

3.3.3 As of 7 November 2019, a flight data analysis programme shall contain adequate safeguards to protect the source(s) of the data in accordance with Appendix 3 to Annex 19.

ICAO Annex 19 designates the fundamentals of Safety Management, called as Safety Management Manual (SMM) (Doc 9859). The flight data analysis program guidance can be found in this manual.

On the other hand, European Union Aviation Safety Agency (EASA) mandates the Flight Data Monitoring (FDM) program to the aircraft operators based on “Easy Access Rules for Air Operations (Regulation (EU) No 965/2012)” [24];

ORO.AOC.130 Flight data monitoring – aeroplanes

(a) The operator shall establish and maintain a flight data monitoring programme, which shall be integrated in its management system, for aeroplanes with a maximum certificated take-off mass of more than 27 000 kg.

(b) The flight data monitoring programme shall be non-punitive and contain adequate safeguards to protect the source(s) of the data.

Thus, it can be deduced that EASA applies the ICAO amendment and mandates the FDM program to the European Union aircraft operators, while FAA is contented with advice.

After the historical background of FDM concept has been summarized, the technical details can be given. The dataflow scheme through the Digital Flight Data Acquisition Unit (DFDAU) for a typical B737-800 could be visualized in Figure 3.1.

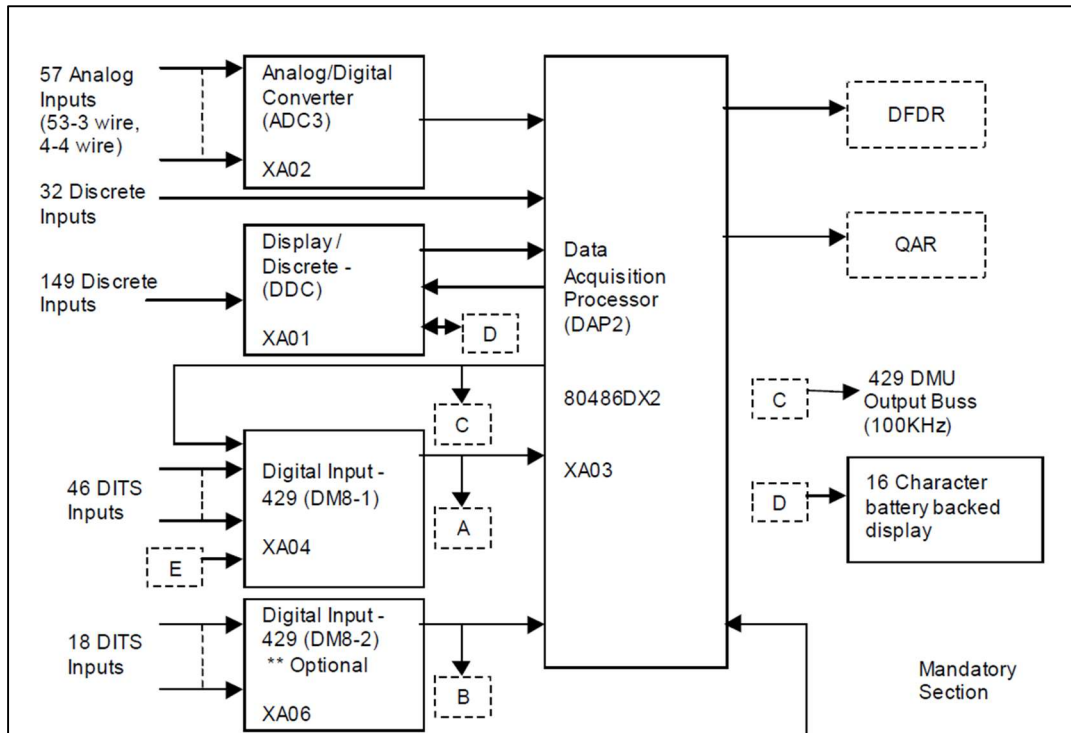


Figure 3.1. DFDAU system block diagram

Aircraft sensors and equipment send analog and discrete signals to the Data Acquisition Processor (DAP). DAP processes and gather the data and send it to the Digital Flight Data Recorder (DFDR) or QAR with ARINC 429 data bus which is the most common two-wire data transmission method for commercial and transport aircraft. The whole process is managed and maintained by the DFDAU block.

DFDR (known as “black box”) is an avionic unit for recording the common aircraft performance parameter. The unit is a crash protective system that records up to 25 hours of data and endures the impact of 3400 g and temperatures over 1000°C. For aircraft accident and incident investigations, the DFDR unit is crucial. Historically, the DFDR design has been improved in many ways against corrosion and extreme conditions. Additionally, underwater beacon locators, which is a system transmitting a signal when it is under the water, have been added to the design in case of missing DFDR under the water.

In addition to DFDR, DFDAU transmits the flight data to QAR. QAR unit records the data and allows easy access to the flight data. Advantageously, QAR can record more than 2000 flight parameters at much higher recording rates than DFDR. The flight data is received in a “Bipolar, Biphase, or Plessey” format and stored in an internal Compact Flash Card (CF Card). The data can be accessed by removing the CF card or wireless. When configured with the wireless option, the data is compressed and encrypted for transmission utilizing cellular technology to the remote Ground Base station server.

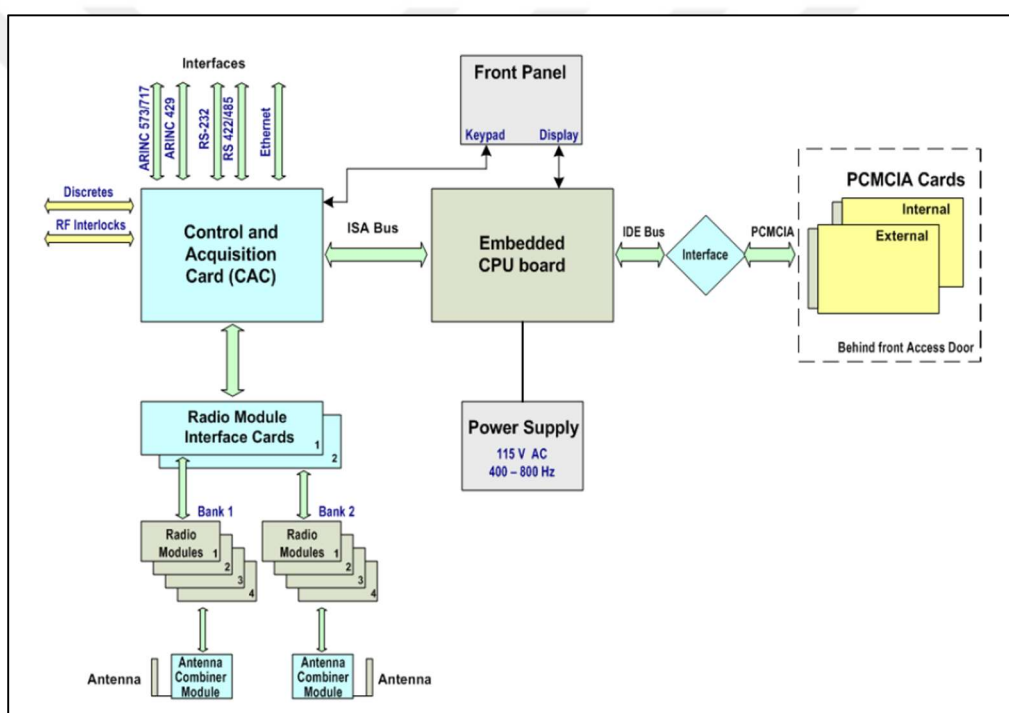


Figure 3.2. Wireless Quick Access Recorder (WQAR) system block diagram

File Posn	Status	File Bytes
011A00h (72192)	Good	47 AA 76 18 30 00 E7 8F FE 5D 60 03 8C 2F 00 E8 6F 01 80 E8 FE 08 B0 FF FD 3F 00
011D00h (72960)	Good	B8 CD 76 18 20 00 E7 AF FE 5D 60 03 88 2F 00 E8 6F 01 70 F8 FE 07 B0 FF FC FF FF
012000h (73728)	Good	47 E2 76 18 20 00 E7 AF FE 5D 60 03 88 4F 00 E8 6F 01 88 F8 FE 08 B0 FF FC 3F C0
012300h (74496)	Good	B8 95 76 18 20 00 E7 8F FE 5D 60 03 88 4F 00 E8 6F 01 70 E8 FE 08 B0 FF FD FF FF
012600h (75264)	Good	47 CA 76 14 10 00 E7 8F FE 5D 60 03 88 4F 00 E8 6F 01 78 08 FF 08 B0 FF FD 3F 00
012900h (76032)	Good	B8 DD 76 14 10 00 E7 8F FE 5D 60 03 88 5F 00 E8 6F 01 80 08 FF 08 90 FF FC FF FF
012C00h (76800)	Good	47 B2 76 14 10 00 E7 8F FE 5D 60 03 88 5F 00 E8 6F 01 7C F8 FE 08 B0 FF FC 3F C0
012F00h (77568)	Good	B8 E5 76 14 10 00 E7 AF FE 5D 60 03 88 2F 00 E8 6F 01 80 08 FF 07 90 FF FD FF FF
013200h (78336)	Good	47 9A 76 14 10 00 E7 8F FE 5D 60 03 88 2F 00 E8 6F 01 6C E8 FE 08 90 FF FD 3F 00
013500h (79104)	Good	B8 FD 76 14 10 00 E7 8F FE 5D 60 03 88 3F 00 E8 6F 01 7C F8 FE 08 90 FF FC BF FF
013800h (79872)	Good	47 92 76 14 10 00 E7 AF FE 5D 60 03 88 0F 00 E8 6F 01 74 08 FF 08 90 FF FC 3F C0
013B00h (80640)	Good	B8 E5 76 14 10 00 E7 AF FE 5D 60 03 88 1F 00 E8 6F 01 84 F8 FE 09 90 FF FC BF FF
013E00h (81408)	Good	47 9A 76 14 10 00 E7 8F FE 5D 60 03 88 1F 00 E8 6F 01 74 08 FF 08 90 FF FD 3F 00
014100h (82176)	Good	B8 CD 76 14 10 00 E7 8F FE 5D 60 03 88 EF FF E8 6F 01 88 08 FF 08 90 FF FD FF FF
014400h (82944)	Good	47 E2 76 10 10 00 E7 AF FE 5D 60 03 88 0F 00 E8 6F 01 78 08 FF 08 90 FF FC 3F C0
014700h (83712)	Good	B8 95 76 10 10 00 E7 AF FE 5D 60 03 8C FF FF E8 6F 01 68 E8 FE 08 90 FF FC FF FF
014A00h (84480)	Good	47 0A 77 10 10 00 E7 AF FE 5D 60 03 88 0F 00 E8 6F 01 60 F8 FE 07 90 FF FD 3F 00
014D00h (85248)	Good	B8 AD 76 10 10 00 E7 8F FE 5D 60 03 8C 1F 00 E8 6F 01 68 18 FF 07 90 FF FD FF FF
015000h (86016)	Good	47 92 76 10 10 00 E7 8F FE 5D 60 03 8C 3F 00 E8 6F 01 68 08 FF 08 90 FF FD 3F C0
015300h (86784)	Good	B8 F5 76 10 10 00 E7 AF FE 5D 60 03 88 4F 00 E8 6F 01 64 E8 FE 07 90 FF FC FF FF
015600h (87552)	Good	47 BA 76 10 10 00 E7 AF FE 5D 60 03 88 4F 00 E8 6F 01 54 08 FF 07 90 FF FC 3F 00
015900h (88320)	Good	B8 CD 76 0C 10 00 E7 8F FE 5D 60 03 88 3F 00 E8 6F 01 60 08 FF 08 90 FF FC FF FF
015C00h (89088)	Good	47 A2 76 0C 10 00 E7 8F FE 5D 60 03 88 4F 00 E8 6F 01 6C 18 FF 08 90 FF FD 3F C0
015F00h (89856)	Good	B8 D5 76 0C 10 00 E7 8F FE 5D 60 03 88 5F 00 E8 6F 01 64 08 FF 07 90 FF FD FF FF
016200h (90624)	Good	47 AA 76 0C 10 00 E7 AF FE 5D 60 03 88 5F 00 E8 6F 01 64 18 FF 08 70 FF FC 3F 00
016500h (91392)	Good	B8 BD 76 0C 10 00 E7 AF FE 5D 60 03 88 5F 00 E8 6F 01 68 D8 FE 08 90 FF FC FF FF
016800h (92160)	Good	47 C2 76 0C 10 00 E7 AF FE 5D 60 03 88 5F 00 E8 6F 01 70 E8 FE 07 70 FF FC 3F C0
016B00h (92928)	Good	B8 C5 76 0C 10 00 E7 8F FE 5D 60 03 88 4F 00 E8 6F 01 60 08 FF 07 90 FF FD FF FF
016E00h (93696)	Good	47 9A 76 0C 10 00 E7 8F FE 5D 60 03 88 4F 00 E8 6F 01 74 08 FF 08 90 FF FD 3F 00
017100h (94464)	Good	B8 CD 76 0C 10 00 E7 AF FE 5D 60 03 88 5F 00 E8 6F 01 50 08 FF 07 70 FF FC FF FF
017400h (95232)	Good	47 D2 76 0C 10 00 E7 8F FE 5D 60 03 88 4F 00 E8 6F 01 70 08 FF 07 90 FF FD 3F C0
017700h (96000)	Good	B8 E5 76 0C 10 00 E7 AF FE 5D 60 03 8C 5F 00 E8 6F 01 70 08 FF 07 70 FF FC FF FF
017A00h (96768)	Good	47 CA 76 0C 10 00 E7 AF FE 5D 60 03 88 4F 00 E8 6F 01 68 08 FF 08 90 FF FC 3F 00
017D00h (97536)	Good	B8 AD 76 0C 10 00 E7 8F FE 5D 60 03 8C 4F 00 E8 6F 01 78 08 FF 08 90 FF FD FF FF

Figure 3.3. Sample Raw DFDAU data

After each flight is completed, the WQAR unit transmits the flight data to the Ground Server, which automatically sends the data to FDM software. Corendon Airlines uses “Aerobytes FDM” software to parse and process data. Aerobytes recognize raw data with the help of the predefined DFDAU frames of the specific aircraft. Raw data is parsed as blocks and separated into words. Each word defines an airborne or ground parameter related to that flight. It is important to know that Aerobytes software distinguishes the bad data producing insignificant parameters. Although some of the data validity problems will be mentioned in Section 3.2 have been solved by Aerobytes (mismatched data, inconsistent data, misdirected data), the rest of the problems remain due to the nature of DFDAU. The resolution and time discretion errors are the most common missing data errors that can be encountered in studies with QAR data. However, a part of the errors will be eliminated via efficient methods, while some will be disregarded because the effort is non-worthy or unfeasible.

Aerobytes FDM keeps the decoded data in an SQL Server. To provide a user command output, Aerobytes queries this SQL server and returns the result accordingly. The data structure can be outlined below [25];

- Each flight is indexed with a unique “Flight ID.”
- States represent a logical part of a flight (such as ‘take-off’, ‘cruise’ or ‘landing’) and are fundamental to the way that Aerobytes FDM analyses flights. States can be defined by users as well.
- State Values store key values for the States and are useful both for finding abnormal events for an individual flight and for longer-term trending across many flights.
- The event represents the situation that which a state value exceeded the predefined threshold for a given state.

In Corendon Airlines, SMS Department defines these thresholds, monitors the events, and analyzes the overall operational safety by using the Aerobytes software. However, the safety concerns are beyond the scope of this study.

Besides the safety relation of a state value, various further calculations and analyses can be done by airlines, especially on aircraft performance, maintenance, and quality aspects. Some common practices can be easily followed by an airline such as cruise degradation factor, which is the estimated fuel deviation of the aircraft from a benchmark aircraft, hard landing detection, and incident investigation.

To perform a fuel study based on FDM data, a database should be created. The database should include all the recorded fuel-related data. A state including all the fuel-related data has been created to provide easy access to the data. The following parameters given in Table 3.1 have been decided to be included in the database;

Table 3.1 Fuel Related Parameters included in the database

Takeoff Datetime	Duration - C Initial Climb	Ground Distance - B Takeoff
Touchdown Datetime	Duration - D Climb	Ground Distance - C Initial Climb
Flight ID	Duration - E Cruise	Ground Distance - D Climb
Aircraft Registration	Duration - F Descent	Ground Distance - E Cruise
Takeoff Airport	Duration - G Approach	Ground Distance - F Descent
Touchdown Airport	Duration - H Landing	Ground Distance - G Approach
Air Distance - C Initial Climb	Duration - I Taxi In	Ground Distance - H Landing
Air Distance - D Climb	Engine Time - B Takeoff	Ground Distance - I Taxi In
Air Distance - E Cruise	Flap - B Takeoff	ISA Temp Deviation - B Takeoff
Air Distance - F Descent	Flap - H Landing	ISA Temp Deviation - H Landing
Air Distance - G Approach	Fuel Burn - A Taxi Out	Mach - Cruise (max for 5 mins+)
Altitude - B Takeoff	Fuel Burn - B Takeoff	Mach - Cruise (mean)
Altitude - C Start of Climb	Fuel Burn - C Initial Climb	Mach - E Cruise Top of Climb
Altitude - D Top of Climb	Fuel Burn - D Climb	Mach - E Cruise Top of Descent
Altitude - E Cruise (max)	Fuel Burn - E Cruise	Mach 50ple - E Cruise
Altitude - F Top of Descent	Fuel Burn - F Descent	Mach 90ple - E Cruise
Altitude - G End of Descent	Fuel Burn - G Approach	Mach 95ple - E Cruise
Altitude - H Landing	Fuel Burn - H Landing	Weight - A Taxi Out
Duration - A Taxi Out	Fuel Burn - I Taxi In	Weight - B Take Off
Duration - B Takeoff	Ground Distance - A Taxi Out	Weight - H Landing

The enlisted parameters have been chosen based on the present studies, technical background, and capability of the present aircraft data flow structure, conservatively. In the future, various fuel performance items can be included among them, if they can be recorded and are necessary.

It should be noted that because the fuel burn is recorded separately for each engine, the total value should be calculated by adding them. Air distance, duration, fuel burn, ground distance, and weight are given separately for different phases of flight. Additionally, the Mach number is calculated and given for cruise segments with different statistical representations. Therefore, the database has been elected and reorganized in the calculation state in accordance with the purpose of the study.

Moreover, as the altitude values play important role in the study and an altitude parametrization methodology will be performed, altitude values have been

employed by calling them from the “Level” state. “Level” state in the FDM system stores altitude values for all flights in “level-off” phases. They will be put in use in the altitude parametrization process explained in Chapter 4.1.

An essential concept called “Airplane Performance Monitoring (APM)” should be introduced to analyze the fuel performance of an aircraft. According to Boeing [26], APM is determining the aircraft’s fuel mileage and comparing it to a “baseline/book” level for the same flight conditions, fundamentally. APM concept can be also followed to determine the current deterioration status of the airframe, engine, and other aircraft systems. The 4 specific parameters are compared to determine deviations from “baseline” performance;

- Fuel mileage (FM) (nm/kg)
- Thrust required ($F_{N\text{Req'd}}$) (airframe effect)
- Fuel flow (FF) (engine effect)
- Power setting required ($N1_{\text{Req'd}}$) (%N1).

In Equations 3.1-3.4, the mathematical representations of the above parameters are given;

$$\% \Delta FM = \frac{FM_{\text{observed}} - FM_{\text{database}}}{FM_{\text{database}}} * 100 \quad (3.1)$$

$$\% \Delta F_{N\text{Req'd}} = \frac{F_{N\text{observed}} - F_{N\text{database}}}{F_{N\text{database}}} * 100 \quad (3.2)$$

$$\% \Delta FF = \frac{FF_{\text{observed}} - FF_{\text{database}}}{FF_{\text{database}}} * 100 \quad (3.3)$$

$$\% \Delta N1_{\text{Req'd}} = \% N1_{\text{observed}} - \% N1_{\text{database}} \quad (3.4)$$

In Corendon Airlines, the data for APM is gathered by Aircraft Condition Monitoring System (ACMS), downloaded via QAR, and automatically transferred to the FDM software. Aerobytes software generates a report in Digital Standard Interface Record Format (DSIRF). This data includes information such as Flight ID (unique ID generated by Aerobytes), date, Mach number, altitude, total air

temperature, gross weight, thrust setting, and fuel flow per engine. The data is recorded at a few stable cruise points. Therefore, for each flight at least 3 records exist. DSIRF files including these records are processed with appropriate APM software to calculate the FM deviation corresponding to each flight. Corendon Airlines uses Boeing Performance Engineer’s Tool (PET) to process DSIRF files. PET software calculates the relative FM deviation of the aircraft according to the baseline. The outputs are stored in an Structured Query Language (SQL) table. The trend of the FM deviation of an aircraft is also used in the flight plans for accurate planning. This practice and how this involves the study will be explained under Section 3.1.2.

In the end, three different SQL tables are generated with actual flight data. Level State values and FM deviation tables are connected to FDM main table with many-to-one relation. This relation is valid due to the records of FM deviation and altitudes being more than one for a flight. The key parameter is “Flight ID.” The structure of actual flight data tables are given in Figure 3.4;

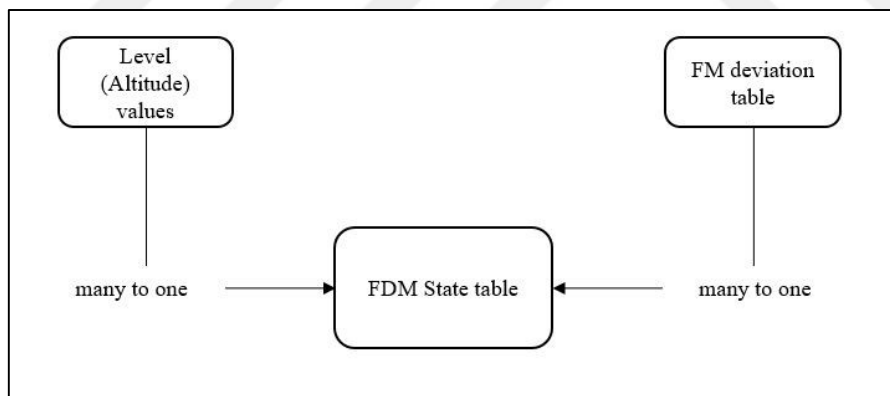


Figure 3.4. FDM data structure

3.1.2 Operational Flight Plan Data

Prior to departure, pilot or flight dispatcher prepares and forwards the flight plan document, which indicates the aircraft’s planned route, to the Air Navigation Service Provider. ICAO Annex 2 [27], “Rules of the Air” mandates preparing an Air

Traffic Services (ATS) flight plan, before operating conditions outlined in article 3.3.1.2 of the “Rules of the Air.” According to this document, information relative to an intended flight should be provided to the air traffic services unit in the form of a flight plan. Content of the ATS flight plan is given in this document as well;

- Aircraft identification,
- Flight rules and type of flight,
- Number and type(s) of aircraft and wake turbulence category,
- Equipment,
- Departure aerodrome,
- Estimated off-block time,
- Cruising speed(s),
- Cruising level(s),
- Route to be followed,
- Destination aerodrome and total estimated elapsed time,
- Alternate aerodrome(s),
- Fuel endurance,
- Total number of persons on board,
- Emergency and survival equipment,
- Other information.

Furthermore, the essentials of submitting and changing the flight plan are outlined in ICAO Annex 2 [27].

The format requirement of the filed ATS flight plan document is identified in ICAO Doc 4444 Appendix 2 [28]. According to this document;

A flight plan form based on the model in Appendix 2 should be provided and should be used by operators and air traffic services units to complete the flight.

Annex 2 states the important changes such as Instrumental or Visual Flight Rules (IFR/VFR) change, fuel endurance information, the total number of people

carried, and incorrect time of departure shall be reported as soon as possible and the ATS flight plan is subjected to a change.

Besides the ATS flight plan, ICAO Annex 6 [29] mandates an operational flight plan (OFP) to be completed for every intended flight in article 4.3.3. After the flight operations officer/flight dispatcher prepares and signs the OFP, the pilot in command (PIC) shall approve and sign it. EU AIR-OPS [24], establishes the content of the OFP, in “AMC1 CAT.OP.MPA.175(a) Flight preparation;”

- Aircraft registration,
- Aircraft type and variant,
- Date of flight,
- Flight identification,
- Names of flight crew members,
- Duty assignment of flight crew members,
- Place of departure,
- Time of departure (actual off-block time, take-off time),
- Place of arrival (planned and actual),
- Time of arrival (actual landing and on-block time),
- Type of operation (ETOPS, VFR, ferry flight, etc.),
- Route and route segments with checkpoints/waypoints, distances, time and tracks,
- Planned cruising speed and flying times between checkpoints/waypoints (estimated and actual times overhead),
- Safe altitudes and minimum levels,
- Planned altitudes and flight levels,
- Fuel calculations (records of in-flight fuel checks),
- Fuel on board when starting engines,
- Alternate(s) for destination and, where applicable, take-off and en-route, including information required in (a)(12) to (15),
- Initial ATS flight plan clearance and subsequent reclearance,

- In-flight replanning calculations,
- Relevant meteorological information.

As could be seen above, the OFP contains all the necessary aircraft and performance information. Nowadays, most aircraft operators embrace “The Paperless Cockpit” which is a concept that all the documents and calculations used by flight crew are contained in an Electronic Flight Bag (EFB). Therefore, the flight crew receives, takes notice, and processes the OFP through the necessary calculations from their EFB interface, Electronic Flight Folder (EFF). EU AIR-OPS “CAT.GEN.MPA.141 Use of electronic flight bags (EFBs)” section defines the boundaries of an EFB system for EU airline operators. A sample of the OFP has been given in Appendix B.

Corendon Airlines uses software called “JetPlan” made by Boeing Jeppesen to prepare the OFPs. JetPlan is a flight planning software capable of making cost-effective performance calculations, considering weather effects based on observation and forecasts. It is able to consider the nature of the complex operational environment and define constraints. Via JetPlan software, the operators may prepare and file ATS flight plans, and transmit flight plans, weather, and Notice to Airmen (NOTAM) information to the flight crew.

JetPlan software carries out the necessary performance calculations using the built-in aircraft databases. For a specific route and payload, the software is capable of determining the most efficient speed schedule, altitude profile, departure (SID), and arrival (STAR) procedure. If a different speed schedule, altitude scheme, or SID/STAR is required, the parameters could be modified in accordance with the necessity. The JetPlan software takes all the input parameters into account and calculates the necessary fuel information for a trip. Required off-block and trip fuel are calculated according to the pre-defined aircraft parameters exclusive to the aircraft type stored in the database. After that, a certain amount of extra fuel is added to the plan as a safety margin. Some of the extra margins are mandated by the authority, while some are left to the decision of the operator. To perform such a

calculation, JetPlan software takes advantage of the aircraft performance database of a specific aircraft type which is created by outputs of a model of equations of motion and aerodynamics of the aircraft. This is the data of the manufacturer contained in the frequently used document known as “Flight Planning and Performance Manual (FPPM).” In the Introduction section, the FPPM states that;

This Flight Planning and Performance Manual [30] has been prepared by Boeing Commercial Airplanes, Commercial Aviation Services. The purpose of this manual is to provide a complete operational performance data reference including dispatch and inflight performance data applicable to a specific airframe, engine rating, and regulatory agency as identified on the cover page of the document.

Because the details of Boeing’s calculation method of performance data are confidential, and there are already non-confidential high-fidelity simulations in the literature such as Eurocontrol Base of Aircraft Data (BADA), the dynamics of the calculation have not been covered in this study.

Since the aircraft performance database behind the calculation is generated for a baseline/new manufactured aircraft, the “Fuel Mileage” deviation of the actual aircraft should be considered. This deviation might be originated because of dents and buckles on the airframe and engine deterioration. Flight plans should take into consideration this deviation to maintain the accuracy of flight plans. JetPlan updates the trip fuel based on input fuel mileage deviation to consider this effect on the flight plan. As it is mentioned in the previous sub-chapter “Actual Flight Data,” the fuel mileage of the aircraft is investigated for specific intervals. For each flight, Boeing PET evaluates the fuel mileage deviation of the aircraft for present air parameters. Corendon Airlines Flight Operations Engineering department keeps the track of FM deviation. If the moving average for the last 30 days has been changed for an aircraft, the dispatchers are informed and requested to update the FM deviation in the JetPlan system. Although the fuel mileage deviation in the OFP represents the aircraft

condition for a time interval, the deviation of the aircraft for a particular flight might be different than the average. Because this difference may result to reduce the accuracy of the fuel burn calculation, each flight's FM deviation of plan and actual should be compared and the difference between them should be incorporated into the model.

To forward the flight planning information to the EFF, JetPlan prepares a package. The package includes OFP in “pdf” and “xml” format, NOTAM and weather information of the departure, arrival, and alternate aerodromes and en-route weather charts. In order to parse the OFP information, it is convenient to use ARINC 633 format xml file rather than a pdf to avoid code generalization errors. A python script, which can be found in Appendix C, is used to parse every OFP for every dispatched flight and record it to the SQL table. Parameters of the fields of the SQL table are given in Table 3.2.

Table 3.2 Parsed OFP parameters

Flight Plan ID	Route Name	Minimum TO Fuel Weight
Date	Flight Level	Full Duration
Departure Hour	Ground Distance	Taxi Fuel Weight
Departure Airport	Air Distance	Block Fuel Weight
Arrival Airport	Great Circle Distance	Block Full Duration
Aircraft	Trip Fuel Weight	Arrival Fuel Weight
AC Model	Trip Fuel Duration	Arrival Fuel Duration
Cruise Degradation	Contingency Fuel Weight	Dry Operating Weight
CPT	Contingency Fuel Duration	Payload
Flight ID	Alternate Fuel Weight	Zero Fuel Weight
Departure Hour	Alternate Fuel Duration	Takeoff Weight
Arrival Hour	Final Fuel Reserve	Landing Weight
Route Optimization	Capable Duration	

Additionally, the OFP contains necessary information about the waypoints of the route. Waypoints are pre-defined coordinates used by aircraft to navigate in the airspace. At each of these coordinates, JetPlan calculates the corresponding altitude, speed, and remaining fuel. Although the commercial aircraft fly in the

transonic flow region, the speed schedule determined by JetPlan considering the company policy called cost index.

$$\text{Cost Index (CI)} = \frac{\text{Operational Time Cost (\$/hr)}}{\text{Fuel Cost (\$/lb)}} \quad (3.5)$$

Because the mean Mach number and altitudes at waypoints will be used as an input parameter in the model, and there is no information about these parameters in the first part of OFP, the waypoints should be parsed as well. For a flight, there is more than one waypoint, they need to be parsed into another table and two tables should be connected with one-to-many relations to call the data. A Python script has been used to parse waypoints and is given in Appendix D.

3.2 Database Relationships

In Section 3.1, two main data sources (Flight Plans and Actual Flight Data) have been introduced and data acquisition methods have been outlined. Even though their relationship is required to be defined in order to achieve a statistical analysis, there is no common key between them. Therefore, a primary key should be set. This can be done by concatenating several columns.

Key: “Departure Date_Departure Airport_Arrival Airport_Aircraft Registration”

Example: “2019-XX-XX_XXXX_XXXX_TC-XXX”

After that, the two tables are bounded. The parameters that will be included in the models should be selected. Too many parameters may result in the model overfit where the model mimics the training data instead of learning the pattern successfully. Moreover, the some predictors may not be related to the dependent variable significantly. Below, the most significant parameters exist in the Master DB are elected and the reasons behind the selection strategy are explained.

- 1) **Takeoff Weight (TOW):** It is directly related to the fuel performance of the aircraft. An extra amount of carried weight causes extra fuel burn.

- 2) **FM Deviation (FMdev):** The fuel mileage deviation concept and its effect on fuel performance have been explained in Section 3.1 for both data sources.
- 3) **Altitude Parameter:** Because the airspace is separated vertically for safety and operational concerns, the commercial aircraft cannot perform continuous climbs in the cruise segment of the flight as presented by the “Breguet Range Equation.” Therefore, the aircraft should maintain the assigned altitude in the airspace. Fortunately, the step-climb method reduces the fuel burn and maximizes the range for that airspace. Since the “Reduced Vertical Separation Minima (RVSM)¹” procedure (2000 ft between each cruise level) is mandated in Corendon Airlines’ operational area, the most convenient altitude schedule is “Optimum Altitude + 1000 ft”, where the optimum altitude is the most efficient altitude for instantaneous conditions and 1000 ft extra margin added due to step climb. In order to perform the parametrization mentioned Chapter 4.1, the level values from the FDM state and OFP waypoints should be gathered by the Python code given in Appendix E. Afterwards, the parametrization is performed by the Python code given in Appendix F.

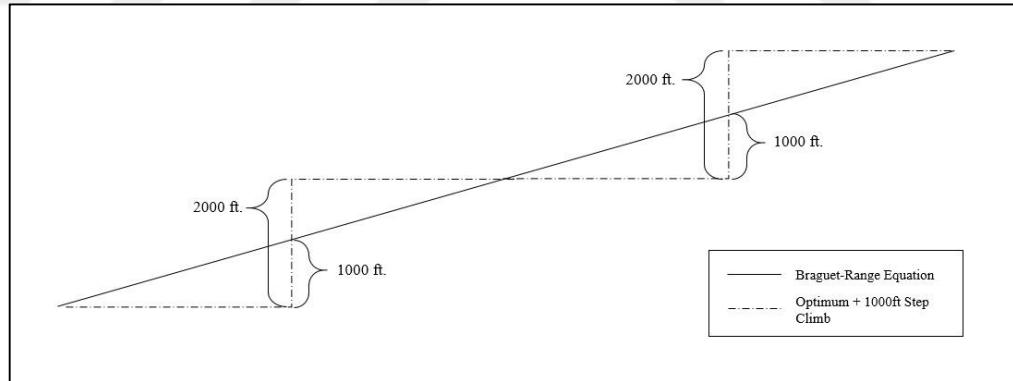


Figure 3.5. Altitude profile in RVSM airspace

¹ Reduced vertical separation minimum (RVSM) airspace is the airspace that standard vertical separation needed between aircraft flying between flight levels 290 (29,000 ft) and 410 (41,000 ft) that is reduced from 2,000 feet to 1,000 feet.

- 4) **Mean Cruise Mach Number:** The mean cruise Mach number value is directly related to the fuel performance. In Corendon Airlines, the cost index is determined as 20 where speed schedule is determined by the cost index, and deviation from the optimum speed schedule results in high fuel burn.
- 5) **Distance/Duration Parameter:** In some of the studies, the distance/time parameter has been used as air distance, ground distance, great circle distance (range), or duration. Because of the data acquisition structure of FDM, distance and duration parameters are recorded separately for different phases of flight. Hence, they are added to obtain total distance/duration. The final schema can be visualized as Figure 3.6.

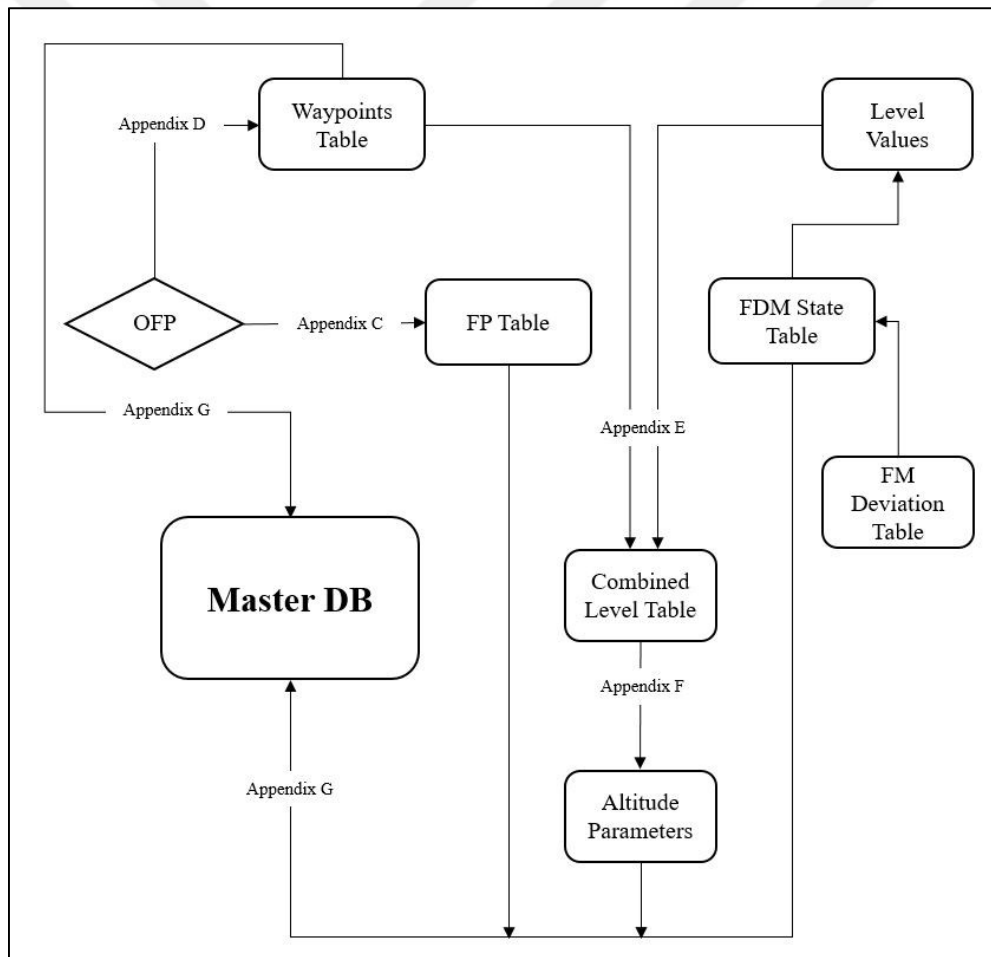


Figure 3.6. Data acquisition and relationship scheme

3.3 Data Validity

A successful data analysis begins with gathering significant data. A model can only be as good as its inputs. Performing statistical analysis with irrelevant or erroneous data input causes misconception, ineffective modeling, and reaching an incorrect conclusion. According to Di Zio & Guarnera [30], data quality can be assessed in terms of the following items;

- **Accuracy:** Some of the parameters may not be easily measured or calculated without an error. The complex structure of data acquisition might cause the need for estimation based on assumptions. In this manner accuracy can be defined as the difference between “target parameter” and “estimated parameter.”
- **Coherence and Comparability:** Internal consistency over time, regions, and countries.
- **Clarity and Accessibility:** Formal tool/technology check for misunderstood/erroneous data. Zieman et al. [31] stated numerous gene studies concluded false results because the spreadsheet software MS Excel converts gene names to dates and floating-point numbers.
- **Timeliness:** Due to complex and time-consuming checks on data, the timing of the final data release might be delayed. Timely data can be assessed as an important quality dimension for some cases.

Some of the concerns above might be taken into consideration for the present case, while the rest might not be because of the nature of the problem. This will be discussed during parameter selection.

Quality concerns can be secured by various data validation methodologies. To perform a validation method, the analyst should be aware of the validation problems on the dataset. Sadiq et al. [32] sum up data validation problems and explain to them as follows;

- Redundant data that does not affect the model,
- Lost data exposed fork/synchronized process,
- Missing data,
- Mismatched data, false structure,
- Inconsistent data,
- Misdirected data, false dataflow,
- Insufficient data.

The items above are possible reasons for having low-quality data that will reduce the overall efficiency of the statistical model. Therefore, they need to be checked and eliminated from the present dataset.

After composing the database structure, and eliminating the flights with missing data the Master DB contains 12765 flights in 2019. The database has been subjected to data validity checks;

- **Redundant data:** Only the parameters that are related to the fuel burn are selected, and the remaining fields are eliminated.
- **Lost data exposed fork/synchronized process:** There is no synchronized process in the data acquisition methods.
- **Missing data:** If there is a missing data in a field, the code snippet given in Appendix H identifies and removes this data row permanently. Fortunately, there is still enough data to train, validate and test the model when these flights are eliminated.
- **Mismatched data, false structure:** The relation between the tables and the structure of the database has been deployed successfully.
- **Inconsistent data:** The data sources (both avionic equipment and flight planning software) are reliable due to the safety policy of the aviation sector.
- **Misdirected data, false dataflow:** Not applicable for this study.
- **Insufficient data:** As stated above, there are enough flights to model.

Finally, the database with the parameters given in Table 3.3 for both high fidelity simulation data outcome (OFP) and the actual flight parameters (FDM) is ready to build a fuel consumption model.

Table 3.3 Input and Output Parameters

Input Parameters	Output Parameters
Mean Cruise Mach	Trip Fuel Burn (kg)
Takeoff Weight (kg)	
Maximum Cruise Altitude (ft) <i>or</i>	
Altitude Parameter A (ft) <i>or</i>	
Altitude Parameter B (ft)	
Air Distance (nm) <i>or</i>	
Ground Distance (nm) <i>or</i>	
Duration (min)	
Fuel Mileage Deviation (%)	



CHAPTER 4

METHODOLOGY

In the literature, various strategies are used for different purposes to build fuel consumption models. For example, in the study [8], the authors developed a linear aircraft type-clustered model using Flight Data Recorder data in order to estimate the fuel burn during the taxi phase. Zou [11] estimated the fuel burn and emissions by domestic commercial aviation in the Continental United States over a year using the physical model Eurocontrol BADA [9] as a baseline model. The author developed a workflow to enhance the incapable sides of BADA by studying each flight phase separately. However, this approach induces high computational expense and readjustment of many parameters such as flight profile, wind effects, and terminal area detour. Finally, studies such as [18], [19], [20] implemented genetic algorithms and neural networks to model aircraft fuel consumption. In order to train the model, they used different databases parsed from performance and pilot manuals or flight data recorder outputs. Afterward, the comparisons between these models and analytical/statistical models have been performed. Undoubtedly, the machine learning algorithms exhibited significantly better results compared to the conventional analytical models.

In summary, it could be said three fundamental approaches exist in the literature to estimate fuel burn;

- 1) **Low-fidelity statistical regressions:** Lower accuracy with a low computational expense,
- 2) **Developing high-fidelity simulations and improving the existing models:** Higher accuracy with a high computational expense,
- 3) **Machine learning algorithms:** The cost effectiveness depends on the case.

On the other hand, there are a couple of studies performed to eliminate the problem of trade between the computational cost and accuracy [15], [16], [33]. These are called surrogate models or medium-fidelity models in the literature. The essential concept behind the study is to build a statistical regression model based on the outcomes of a high-fidelity simulation. The existing surrogate models' workstream could be outlined as below;

- 1) Gather flight mission information for a sufficient number of flights to cover all possible operations,
- 2) Run high-fidelity simulations with these missions and build a database including the outputs,
- 3) Derive surrogate statistical regression model using the database,
- 4) Cluster the models based on the aircraft types.

Therefore, implementing the surrogate models would be beneficial for individuals and organizations in means of reduced computational costs and improved accuracy. Additionally, since most of the high-fidelity models are commercial; students and researchers could take advantage of these models for non-commercial purposes.

This study is intended to combine a medium-fidelity and artificial neural network models to estimate fuel burn using generic aviation parameters which can be accessed easily. Although the existing ANN models are trained by the performance/pilot manuals or DFDR data, the machine learning model in this study has been trained by the high-fidelity simulation output data which have been performed before each flight as a part of daily operation. The genetic algorithm deployed would forecast the fuel burn by looking at the five selected parameters. The simulation data which the ANN model is trained is based on the regular conditions of operation of a fleet with a single aircraft type Boeing 737-800. Therefore, it reflects the baseline characteristics of the aircraft type for corresponding flight missions. If the actual conditions of any flight diverge from the baseline in means of the parameters not included in the model, the fuel consumption estimation also

diverges. Within this assumption, using FDM data as the test dataset would expose the flights that diverge from the usual conditions. Thus, the procedure allows us to identify non-normal fuel consumption in the flight mission due to unexpected disturbances. Some of the possible disturbances can be found below;

- Improper pilot technique during climb and descent,
- Turbulence/Wind shear conditions,
- Holding/Excessive air traffic in the terminal area,
- Instrument fault on the aircraft,
- Flight data defect.

The fuel burn because of the generic flight parameters and optimization of these parameters is a common subject. Most of the studies focus on these parameters' optimization to lower the fuel burn. This study provides a powerful tool to estimate fuel consumption using the minimum number of inputs. However, more importantly, it identifies the non-normal fuel consumption of a flight, just by testing the model with actual flight data. If the model's estimation deviates from the actual fuel burn, this means the above disturbances (or another) might be incurred during the flight.

The methodology and calculation are given and summarized;

- 1) Obtaining the “Master DB” which includes all the parameters related to the fuel consumption. For each flight, the database should include both simulated and actual flight data for the same parameter.
- 2) Determining the best dataset to model the nature of a flight.
- 3) Training various machine learning (ML) models to estimate the fuel burn using the simulation data (train dataset).
- 4) Optimizing hyper-parameters of the model with the validation dataset.
- 5) Measuring the performance of the model with the test dataset.
- 6) Identifying the flights discrepant from the model.
- 7) Sampling some of the flights that have non-normal fuel consumption as a test of the entire dissertation.

Additionally, a flowchart visualizing the methodology is given in Figure 4.1.

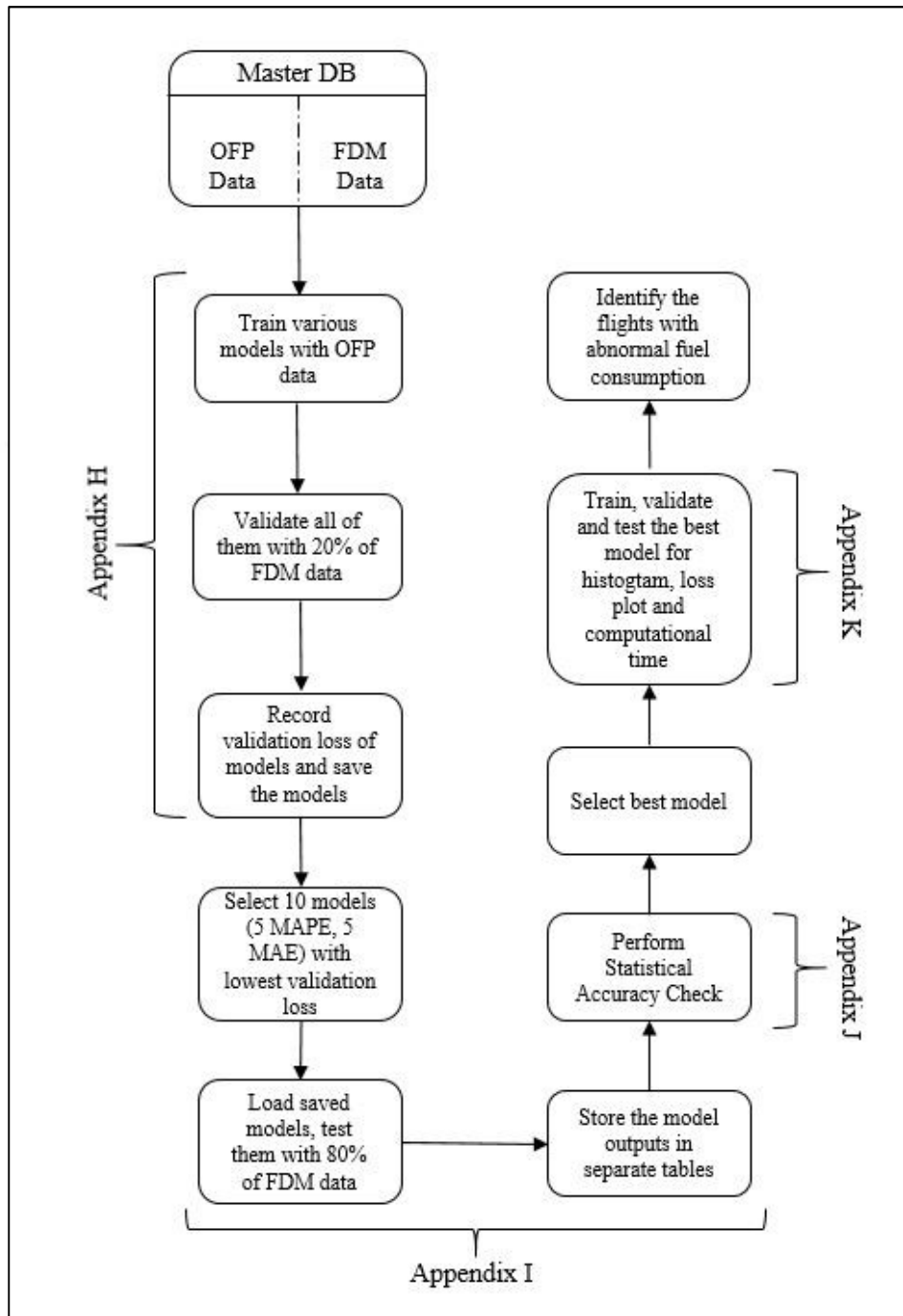


Figure 4.1. Study methodology flowchart

4.1 The Data Acquisition and Review Process

The data acquisition and review process has been outlined in the previous chapter. In the end, we had the below data structure for both high fidelity simulation data outcome (OFP) and the actual flight parameters (FDM) are available and given in Table 4.1.

Table 4.1 Input and Output Parameters

Input Parameters	Output Parameters
Mean Cruise Mach	Trip Fuel Burn (kg)
Takeoff Weight (kg)	
Maximum Cruise Altitude (ft) <i>or</i> Altitude Parameter A (ft) <i>or</i> Altitude Parameter B (ft)	
Air Distance (nm) <i>or</i> Ground Distance (nm) <i>or</i> Duration (min)	
Fuel Mileage Deviation (%)	

As previously mentioned, the number and content of the independent parameters in the model are decided in accordance with the physics of the problem and accessibility. The selected parameters define the general characteristic of any flight successfully and any disturbance on these parameters causes a strong effect on the target parameter which is the trip fuel burn. Moreover, as these parameters could be accessed easily, individuals and organizations will be able to benefit from the model with their dataset and problem behavior.

The parametrization approach on *Mean cruise Mach number* calculated over planned waypoints and *Mean altitude parametrization* could help to handle the flight segmentation difficulties. Some of the researchers, such as [12], simplify the fuel burn calculations under the assumption that the flights consisted of only a cruise segment while the others, such as [34] and [15], develop complex algorithms and calculation processes to mimic the complex nature of the climb and descent phases. However, the proposed approach, the parametrization of the mean Mach number through the waypoints given in the flight plan and the altitude parametrization over

the whole flight segments, lends assistance to estimating the fuel burn through the trip without any necessity for splitting the model based on the flight phases.

Two possible altitude parametrization methods have been proposed in this study. These are named “Parameter A” and “Parameter B.” “Parameter A” parametrizes the cruise altitude over the entire flight duration while “Parameter B” takes the entire flight profile into account. An example of a one-step-climb cruise flight profile and mathematical representations of “Parameter A” and “Parameter B” are given below.

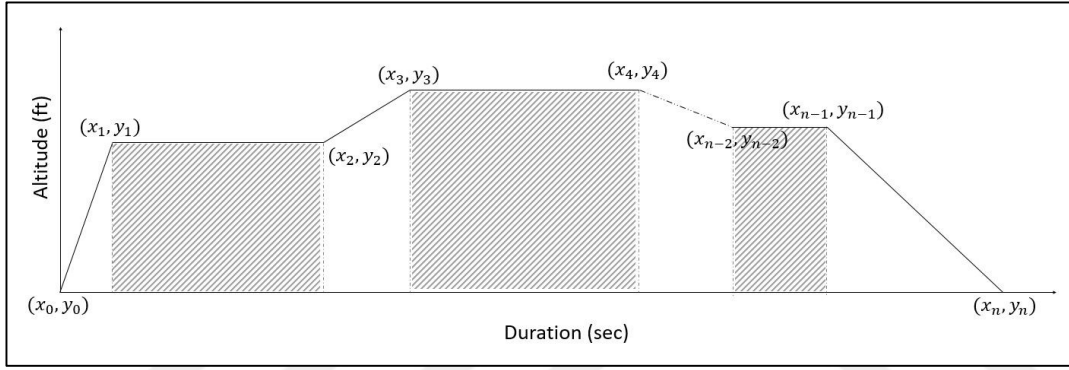


Figure 4.2. Sample flight profile and parametrization method A

$$Param_A = \frac{\sum_{i_c=1}^n (y_{i_c+1} + y_{i_c})(x_{i_c+1} - x_{i_c})/2}{\sum_{i_c=1}^n (x_{i_c+1} - x_{i_c})} \bigg/ \frac{\sum_{i_c=1}^n (x_{i_c+1} - x_{i_c})}{\sum_{i=0}^n (x_{i+1} - x_i)} \quad (4.1)$$

In equation 4.1, x and y correspond to duration in seconds and altitude in feet respectively. For “Parameter A”, “ i_c ” corresponds to the level-off altitudes, therefore y_{i_c+1} and y_{i_c} are equal. “Parameter A” calculates the cruise altitude’s contribution over the entire flight profile with cruise duration’s contribution over the trip (entire) duration.

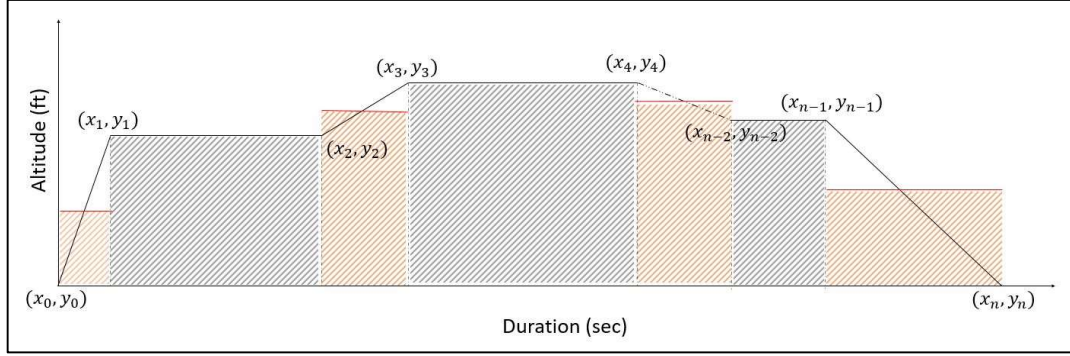


Figure 4.3. Sample flight profile and parametrization method B

$$Param_B = \frac{\sum_{i=0}^n (y_{i+1} + y_i)(x_{i+1} - x_i)/2}{\sum_{i=0}^n (x_{i+1} - x_i)} \quad (4.2)$$

“Parameter B” averages the altitudes during level changes to take the altitude change phases (climb, step climb, and descent) into account. Therefore, an altitude parametrization over the entire flight profile can be considered. This methodology has been achieved by the Python code given in Appendix F.

Furthermore, the takeoff weight is taken into account to include all the effects of payload, zero fuel weight, and fuel carried. The aircraft that carries more weight results in more fuel consumption. Any time-dependent parameter also shall be considered in the model. The possible parameters to be included are decided in the previous chapter;

- Duration,
- Air distance,
- Ground distance.

The ground speed of an aircraft is the sum of the wind speed and airspeed component.

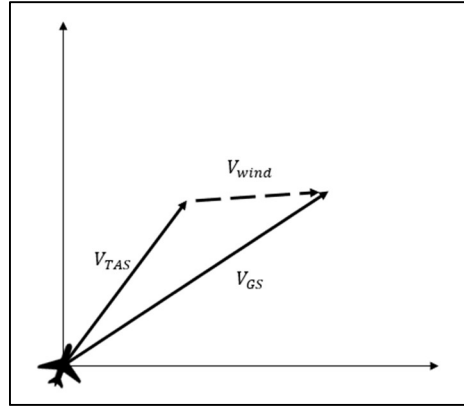


Figure 4.4. True airspeed (TAS) vs ground speed (GS)

Because the distance is directly correlated with speed, also the ground distance of an aircraft is the product of the wind distance and air distance components. The aircraft consumes fuel only for the air distance component. Therefore, the air distance theoretically precedes the rest of the parameters because it is independent of the steady wind component that has no significant effect on fuel consumption.

Moreover, the fuel mileage deviation represents the deterioration of the aircraft in means of airframe and engine. It is well-known that aircraft manufactured sooner exhibit better fuel efficiency performance. However, the deterioration of an aircraft varies with time because of the maintenance overhaul procedures. Therefore, the fuel mileage deviation is taken into consideration in the model.

Finally, it can be said two parameters have three options to select from them. Theoretically, “Altitude Parameter B” and “Air Distance” are the ones that comply the most with the nature of the problem although the other options will be tested out.

4.1.1 Standardization

It is well-known fact that bringing data onto the same scale is important. As it is explained in [35], the scale difference between the fields results in slower convergence on the model due to bigger errors. The model loss function will be dominated by the larger-scaled data. To avoid this, a built-in function in the Python library `skicit-learn`, `StandardScaler` would be used. The mathematical representation of the function can be seen below;

$$x_{std}^i = \frac{x^{(i)} - \mu_x}{\sigma_x} \quad (4.3)$$

Above, $x^{(i)}$, x_{std}^i , μ_x , σ_x represent given input, standard form of the input, the mean and the standard deviation of the input set, respectively. A small portion of standardized data from the “Takeoff Weight (kg)” field has been sampled in Table 4.2.

Table 4.2 Sample Standardized Parameters

Input	Standardized
73282	0.973032
70052	0.288228
65908	-0.590356
67509	-0.250923
61528	-1.518975

4.1.2 Train-Validation-Test Datasets

In order to build a successful model, the datasets should be divided into train-validation-test carefully. Therefore, it is important to understand the main philosophy behind the segmentation. According to [36], in data-rich situations, it is the best approach in both the model selection and assessment processes to split the data into three datasets; train, validation, and test datasets. The below definitions are also given in [36];

- The training sets are used to fit the model into given data,
- The validation sets are used in the model selection as means of estimating the model's performance,
- The test datasets are kept in a “vault” and used at the end of the model selection process to assess the generalization error of the final chosen model.

If the test set has been used to evaluate the model performance repeatedly during the model selection process, the true test error of the final model will be underestimated.

Furthermore, the validation set has been explained in the [37] as follows;

The validation set is used on fitted model to predict the responses for the observations in means of validation set error rate. The assessment would provide quantitative response to evaluate the model performance.

Regarding the above explanations, it could be said that it is beneficial to use a validation set to select the best model and tune the hyper-parameters of the model. The tuned hyper-parameters can be seen in the next section.

Because the trained model should be tuned to show the best performance over actual flight data, the validation set will be chosen amongst the FDM data. The separated dataset is visualized in Figure 4.5.

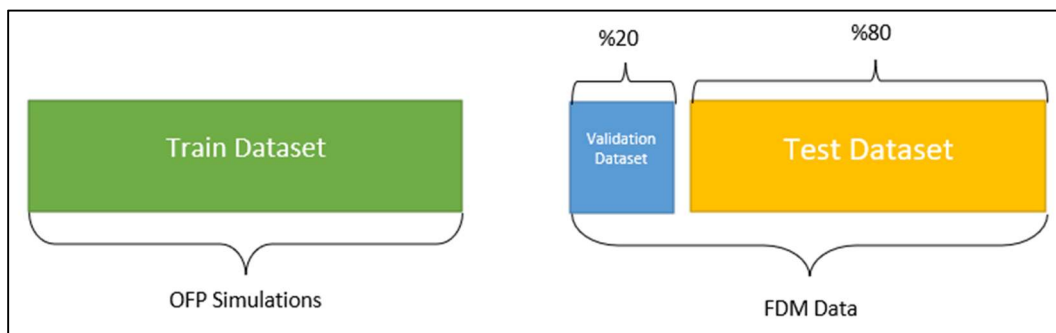


Figure 4.5. Data structure

4.2 The Computational Setup

The model has been trained with the following hardware;

- Intel i5-8300H 2.30GHz CPU,
- Nvidia GeForce GTX 1050 GPU,
- 8 GB RAM.

The programming language Python v3.7 has been used to build the model. An open-source cross-platform integrated development environment (IDE) for scientific programming in the Python language [38] Spyder v5.2.0, has been used during the coding process.

The reasons why Python has been selected to build the model could be explained below;

- Simple and consistent,
- Wide variety of libraries to build machine learning models, data analysis, and visualization,
- Independent platform,
- Easily accessible documentation and active community.

The databases to import to the model have been created, reorganized, and merged by the Pandas library. Finally, an open-source machine learning framework library, PyTorch has been used to code the neural network model. PyTorch's simple tensor-oriented nature and built-in functions inside allow users to handle statistical difficulties and problems during modeling. Furthermore, PyTorch has CUDA support which is a parallel computing platform and programming model developed by NVIDIA® for general computing on graphical processing units (GPUs) [39]. While the training process, CUDA has been employed to reduce the computational time up to 70%.

4.3 Training the Neural Network

The nature of the problem brought forward is a mathematical function that can be approximated by a Feed-Forward Neural Network (FNN) whose original goal is to estimate a target function. Besides FNN, there are so many complex neural network architectures that are specialized to perform particular tasks. Arguably, the most well-known ones are Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). RNNs have been repeatedly shown to be powerful to process sequential or time-series data e.g. speech signal. On the other hand, CNNs are mainly specialized to learn from temporal data to capture local patterns such as in object recognition tasks. However, despite their power on these aforementioned tasks, these methods are complex and expensive when compared with FNNs (also called as multilayer perceptron). Due to their simplicity, FNNs are more efficient choice to capture nonlinear relationship and approximate the function.

The fundamental concept of the deep feedforward network is explained in the [40] as follows;

In order to transform the linear models to exhibit the nonlinear nature of the problem, one may apply the linear model to not the input itself but to the nonlinear transformation of the input. The input function x is mapped through a broad class of nonlinear transformation functions $y = f^*(x, \theta)$ where θ is the nonlinear representation of parameterized x . After that, nonlinear parameters are optimized to find the best representation. Therefore, the neural network learns the best nonlinear transformation to approximate the function.

These models are referred to as feedforward because information flows from input function x to the function being evaluated, then to the intermediate computations used to define f , and finally to the output y . There are no feedback connections, so the model's outputs are not fed back into it. A typical feedforward neural network could be seen in Figure 4.6.

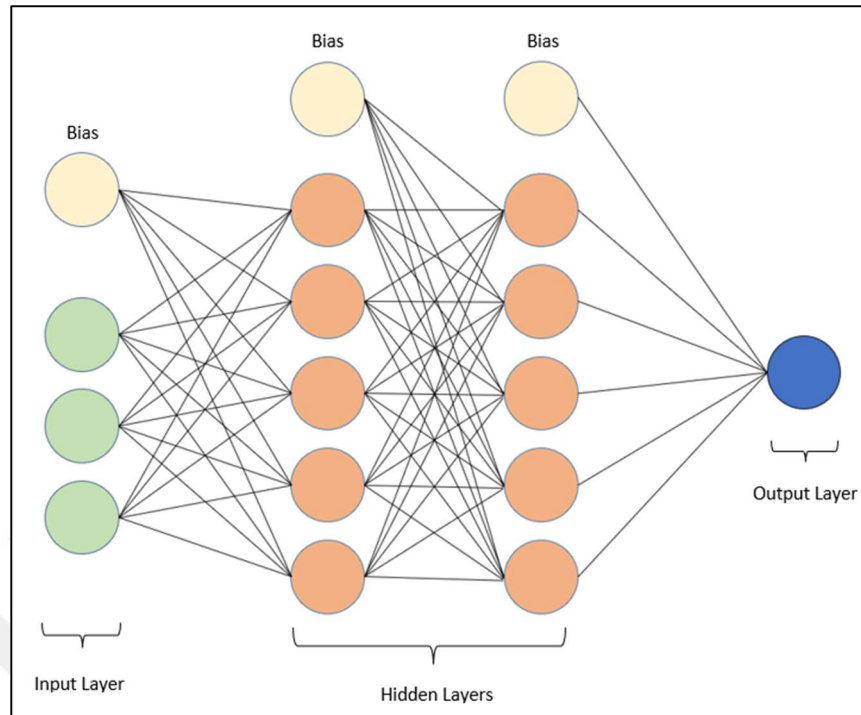


Figure 4.6. Typical Feedforward Neural Network

4.3.1 Hyperparameters

Hyperparameters are the parameters whose values control the learning process and determine the values of model parameters learned by a learning algorithm. They are 'top-level' parameters that control the learning process and the model parameters that result from it, as the prefix 'hyper_' suggests.

According to [41], various machine learning models are parametrized by a set of hyperparameters to maximize the accuracy of the model's learning approach. The default hyperparameters may not show the optimal performance of the ML technique, therefore additional attention should be paid to tuning them [42].

There are studies to optimize the hyperparameters with different approaches such as Bayesian and related sequential model-based optimization techniques. On the other hand, the random sampling method has been also used as the baseline compared with the other techniques.

Some of the hyperparameters in the feed-forward neural networks can be given below;

- Train-validation-test split ratio,
- Learning rate in optimization algorithms,
- Choice of optimization algorithm (e.g., gradient descent, stochastic gradient descent, or Adam optimizer),
- Choice of activation function in a neural network layer (e.g. Sigmoid, ReLU, Tanh),
- The choice of cost or loss function in the model,
- Number of hidden layers,
- Number of activation units in each layer,
- Number of iterations (epochs) in the training phase,
- Batch size.

In this study, a couple of the above, such as split ratio, choice of the optimizer, activation function, and batch size, have been selected in accordance with the existing studies on feed-forward neural networks (FNN) to give the best accuracy. The random sampling method has been used by iterating the model in the hyperparameter spectrum. Hyperparameters and their spectrums have been given in Table 4.3.

Table 4.3 Hyperparameters and their spectrums

Hyperparameter	Spectrum
Number of hidden layers in the linear model	[3, 2]
Number of static nodes in the hidden layers	[16,32,64]
Number of iterations in the training process	[50,100,200]
Learning rate of the optimizer	[0.01,0.001,0.0001]
Loss Function	[MSE, MAPE]
Altitude parameter	[ParamA, ParamB, Mean Cruise Altitude]
Time/Distance parameter	[Ground Distance, Air Distance, Duration]

4.3.2 The Optimization Algorithm Selection

The use of an optimization algorithm is explained by [43] as follows;

An optimization algorithm is used to incrementally improve the design until it can no longer be improved or until the budgeted time or cost has been reached.

The optimization process should be suitable for the final design of the system to provide the most successful results.

In machine learning applications, the most common types of optimization algorithms are continuous function optimization in which the input parameters are floating-point values in a continuous range. Within this context, the output parameters are likely to be a float valued too. In this thesis, the input and output parameters of the proposed model are float point values in a continuous range because of the physics of the problem. Therefore, the optimization algorithm should be suitable for continuous function optimization in which many different types of algorithms can be used for that case.

In the following, the common differentiable objective functions are briefly introduced and the most successful derivative function in means of the proposed study will be chosen. The detailed definitions and classification could be found in the literature easily.

- **Gradient Descent (GD):** GD algorithms find the local minimum of the error by updating the input parameters in the opposite direction of the gradient of the loss function for the parameters [44]. The goal is to minimize the loss by finding the local minima. The learning rate is the size of the step that the algorithm takes in order to reach this minimum. There are several GD algorithm variants in the literature.

- **Stochastic Gradient Descent (SGD):** Because the batch gradient descent calculates the gradient of the loss function concerning all parameters for the entire dataset, it could be very slow. SGD overcomes this situation by performing a parameter update at a time. This prevents redundant calculations and speeds up the learning process. However, SGD results in frequent updates with big variance resulting in the cost function fluctuating.
- **Momentum:** Momentum is an algorithm for accelerating SGD in the desired direction while dampening oscillations. It accomplishes this by adding a fraction of the previous time step's update vector to the current update vector.
- **AdaGrad:** In order to overcome the problems caused by the constant learning rate, Adagrad [45], an adaptive subgradient method has been proposed. The method adjusts the learning rate according to the parameters, with greater updates for infrequent parameters and smaller updates for frequent ones. However, Adagrad has a weakness that causes the learning rate decreases until it becomes infinitesimally small, at which point the algorithm can no longer acquire new information.
- **Adam:** The algorithm proposed in [46] is the combination of the advantages of Adagrad and RMSProp (not mentioned in this study). Instead of using the mean first moment as proposed in RMSProp, Adam uses the average of the second moments of the gradients to adjust the parameter learning rates. The algorithm creates an exponential moving average of the gradient and the squared gradient, with the decay rates of these moving averages controlled by the constants β_1 and β_2 .

According to many contributors, the Adam algorithm is superior to the other common algorithms for most of the tasks in different manners. For example, the paper [44] states that;

Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. Kingma et al. [46] show that its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice.

Additionally, it is recommended to use adaptive learning-rate algorithms for sparse data. Therefore, the Adam optimizer algorithm will be used in this study.

4.3.3 The Loss Function Selection

As [40] states, the loss function is the function targeted to minimize or maximize. It is also known as the cost function, loss function, or error function while minimizing it. The general idea behind the loss function in statistical regressions is similar to the loss function in the machine learning model. It is critical that the function accurately reflects the design objectives. If a poor error function is used and unsatisfactory results are obtained, it is because the search goal was not stated well enough [47]. The common loss functions in the literature can be seen below;

- Mean squared error (MSE),
- Mean absolute percentage error (MAPE),
- Mean absolute error (MAE),
- Mean bias error (MBE),
- Cross-entropy/Logarithmic loss,
- Hinge loss/Multi class SVM loss.

The first 4 are the loss function for the regression models. The others can be used for classification problems. MSE and MAPE will be used as loss functions

while tuning the hyperparameters since they are able to reflect the character of the model most successfully. Below, the mathematical representations of the functions are given in equations 4.4 and 4.5;

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad (4.4)$$

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (4.5)$$

where y_i and $\hat{y}_i$ are the target and model outputs respectively.

4.3.4 The Activation Function Selection

In a neural network, an activation function defines how the weighted sum of the input is transformed into an output from a node or nodes in a network layer. In other words, an activation function determines whether or not a neuron should fire. Whether the information received by the neuron is relevant for prediction or should be ignored. The conventional types of activation functions are given as below;

- Linear function
- Sigmoid function
- Tanh
- ReLU
- Softmax

Although it is not practically possible to predict which activation function gives the best results, as it is stated in [40], the Rectified Linear Unit (ReLU) is usually the best option among the others in most of the models. Recent studies suggest the ReLU is the most successful because of its nature of non-saturating nonlinearity which causes faster convergence compared to the other activation functions [48]. Furthermore, there are discussions in the literature that reveal the benefits of the simple nature of the ReLU that thresholds a matrix of activations at zero, despite the expensive operations of tanh and sigmoid functions. ReLU function

will be used as the activation function in the proposed model. The mathematical explanations of the other activation function are behind the scope of this study.

The mathematical and graphical representation of the function can be seen in Equation 4.6 and Figure 4.7.

$$f(x) = x^+ = \max(x, 0) \quad (4.6)$$

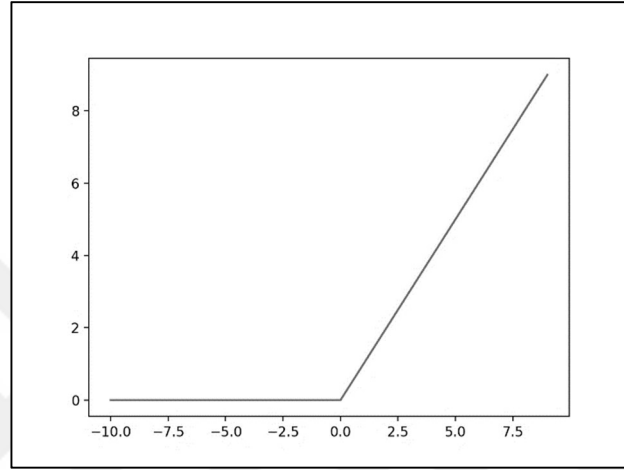


Figure 4.7. ReLU graphical interpretation

4.3.5 Training and Validation

At this point, the proposed model has been designated, and its parameters and hyperparameters have been defined. The Python code given in Appendix H trains the model and tunes the hyperparameters with minimum loss. At each epoch, a shuffled data batch has been forwarded through the model and total loss has been calculated over the batch. During this feed-forward, the backpropagation algorithm performs a backward pass while adjusting the model's weights and biases. Finally, the optimizer performs a single parameter update. After that, the validation dataset has been forwarded through the network of this epoch, and validation loss has been calculated. Differently, there is no back-propagation for the validation set because the dataset is separated from the training dataset and it should not be involved in the training process. The training loss indicates how well the model fits the training data,

whereas the validation loss indicates how well the model fits new data. At each epoch, the training and validation losses are compared to the previous losses. If the current validation loss is less than the minimum validation loss at the current epoch, then the model is saved at that epoch in this optimizer step. In the end, the model is trained with the minimum validation loss.

In the code, “for” loop blocks have been used to iterate over models with different parameters and hyperparameters. In order to record the model’s performance with respect to the most successful parameters and hyperparameters, at the end of the training and validation phase of each model, the configuration and corresponding model loss have been taken into a log table.

The final log file comprising the trained model is structured as below given in Table 4.4.

Table 4.4 Sample of the Logbook

Altitude Parameter	Distance/Duration Parameter	Number of Nodes in a Layer	Learning Rate of the Optimization Algorithm	Number of Epochs	Number of Layers	Loss Function	Training Loss	Validation Loss
ParamB_FP	AirDistanceFP	64	0.001	200	4layers	MSE	0.000922	0.001257
Altitude_MaxFP	AirDistanceFP	16	0.001	200	3layers	MAPE	0.173154	0.193785
ParamA_FP	GroundDistanceFP	64	0.0001	200	4layers	MSE	0.017397	0.02814
ParamA_FP	AirDistanceFP	16	0.0001	50	3layers	MAPE	0.621963	0.67584
ParamB_FP	DurationFP	32	0.0001	200	4layers	MAPE	0.218821	0.23723
ParamA_FP	DurationFP	64	0.01	50	3layers	MSE	0.00382	0.00418
ParamB_FP	DurationFP	16	0.0001	100	4layers	MAPE	0.446256	0.45516
Altitude_MaxFP	DurationFP	64	0.01	50	4layers	MSE	0.273174	0.286210
Altitude_MaxFP	GroundDistanceFP	32	0.001	200	4layers	MSE	0.012116	0.024155
ParamA_FP	DurationFP	64	0.001	200	3layers	MAPE	0.193677	0.212526
Altitude_MaxFP	DurationFP	64	0.001	50	3layers	MSE	0.003477	0.004138
ParamA_FP	AirDistanceFP	16	0.01	200	3layers	MAPE	0.198436	0.207852

The validation loss helps in tuning hyperparameters and gives hints about the best model that fits the test data. However, it does not indicate the best model directly in a statistical manner. Due to the randomness of the training process, the different models that have closer validation losses do not give clues about their accuracy comparison. For example, the model with lower validation loss may have lower statistical accuracy. Additionally, the orders of validation loss of MAPE loss function and MSE loss function are different. They should be treated separately.

Therefore, the five models with the lowest validation loss have been selected for both of the loss functions MAPE and MSE out of the log file.

4.4 Testing the Model and Accuracy Check

The Python code given in Appendix I has loaded the selected saved models. The ten (5 MAPE, 5 MSE) best-trained models have predicted the fuel consumption of the 80% of the previously separated FDM data. The estimations and the actual value have been recorded in excel with an index of the Flight ID of the FDM system.

Finally, each of the model prediction databases has been exposed to the statistical accuracy check by the Python code given in Appendix J. The metrics that have below mathematical expressions have been used to decide the most accurate model;

- 1) Mean Absolute Error

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (4.7)$$

- 2) Mean Absolute Percentage Error

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (4.8)$$

- 3) Mean Squared Error

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad (4.9)$$

- 4) Coefficient of Determination

$$R^2 = 1 - \frac{Unexplained\ Variation}{Total\ Variation} \quad (4.10)$$

Over the above statistical metrics, the model with the best performance has been selected. Next, the model will be used to identify the flights with abnormal fuel consumption.

4.5 Identification of Abnormal Fuel Consumption

After numerous models are trained and the best model has been selected over these models, the final objective can be achieved. For a flight, the abnormal fuel consumption has been diagnosed with a straightforward procedure; the deviation from the model's estimation has been counted as abnormal fuel consumption.

Because the model has been trained with the simulation data created by Boeing's experimental data and physical equations, the deviation from this model based on the simulation data addresses the unseen disturbances over the flight envelope. Three-Sigma Boundary methodology has been followed to identify these flights.

4.5.1 Three-Sigma Boundary

The term "three-sigma boundary" refers to a statistical computation in which the data are within three standard deviations of the mean. In statistical quality control charts, three-sigma limits are used to determine the upper and lower control limits. Control charts are used to set limitations for processes that are statistically controlled.

The mathematical representations of standard deviation and three-sigma boundary are given below;

$$3\sigma_{limits} = [\bar{y} + 3\sigma, \bar{y} - 3\sigma] \quad (4.11)$$

where $\bar{y}$ is the mean of population and

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{N}} \quad (4.12)$$

with N represents population size.

As the prediction error of fuel consumption data depends on the magnitude of the target value, it is accurate to calculate the standard deviation in percentage.

Therefore, the standard deviation of “percentage error” has been calculated. Thus, the MPE will be used as $\bar{y}$.

$$\bar{y} = MPE = \frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{y_i} \right) \quad (4.13)$$

Finally, the flights out of this boundary have been identified and indexed with Flight ID (FID) in the FDM system.





CHAPTER 5

RESULTS

The OFP and FDM data of 12765 flights have been gathered in Master DB. All of the OFP data has been used for training the neural network model where 20% (2553 flights) of the FDM data validated the model. Within this context, the Python algorithm trained, validated, and saved 962 models that have different parameters, hyperparameters, weight, and biases. The computational time is 87456.29 seconds which corresponds to 24.29 hours.

As explained earlier, the loss function of half of the models is MAPE, and the rest of it is MSE. Since the order of the loss functions is different, they have been evaluated and compared based on the same loss function. Ten models with the best performance (5 MAPE, 5 MSE models) and their training and validation loss are given in Table 5.1.

Table 5.1 Ten best models

Model Index	Altitude Parameter	Distance/Duration Parameter	Number of Nodes in a Layer	Learning Rate	Number of Epochs	Number of Layers	Loss Function	Training Loss	Validation Loss
1	ParamB_FP	AirDistanceFP	64	0.001	100	4layers	MAPE	0.123456	0.147237
2	Altitude_MaxFP	AirDistanceFP	64	0.001	200	4layers	MAPE	0.130137	0.149823
3	ParamB_FP	AirDistanceFP	32	0.001	200	3layers	MAPE	0.137180	0.150480
4	ParamB_FP	AirDistanceFP	64	0.001	200	4layers	MAPE	0.127317	0.152700
5	ParamA_FP	AirDistanceFP	64	0.001	200	4layers	MAPE	0.127975	0.153190
6	ParamB_FP	AirDistanceFP	64	0.001	200	4layers	MSE	0.000921	0.001257
7	ParamB_FP	AirDistanceFP	64	0.001	200	3layers	MSE	0.000966	0.001305
8	ParamB_FP	AirDistanceFP	32	0.001	200	4layers	MSE	0.001070	0.001335
9	ParamB_FP	AirDistanceFP	64	0.0001	200	4layers	MSE	0.001069	0.001352
10	ParamB_FP	AirDistanceFP	32	0.001	200	3layers	MSE	0.001091	0.001374

5.1 The Best Model Results

The previously separated 80% (10212 flights) of the FDM data has been deployed into the models. The test results of the above models have been written in

tables separately. The tables have three fields “Flight ID,” “yhat” and “y” where “yhat” and “y” stand for estimation and target value respectively. After these test tables have been obtained by Appendix I, each table has been examined in Appendix J in means of statistical accuracy check. The statistical accuracy metrics explained in Chapter 4.4 have been used to decide the best model with the help of the code snippet given in Appendix J. In Table 5.2, the models, and their accuracy levels have been tabulated over the metrics.

Table 5.2 Results of accuracy check

Model Index	MAE	MAPE	MSE	RMSE	R^2
8	95.91976	0.011736	15285.04	123.6327	0.997322
10	96.62383	0.012089	15309	123.7295	0.997318
1	96.20231	0.01221	15518.7	124.5741	0.997281
9	98.69248	0.012486	15909.86	126.1343	0.997213
3	98.85913	0.012363	16505.16	128.4724	0.997109
6	102.9525	0.013162	17167.42	131.0245	0.996993
7	102.4611	0.01284	17317.94	131.5976	0.996966
4	102.8845	0.013015	19056.77	138.0462	0.996662
2	120.0805	0.015176	24230.17	155.6604	0.995755
5	118.9533	0.014618	24238.35	155.6867	0.995754

The model that has the lowest error is Model 8. The parameters and hyperparameters which have been employed to train the model are;

- Parameter B,
- Air Distance,
- 32 Nodes in hidden layers,
- 0.001 Learning rate,
- 200 Epochs,
- 1 input, 2 hidden, 1 output layers,
- Mean squared error loss function.

Finally, a dummy Model 8 has been tested in order to measure the computational time for training and test by the Python code given in Appendix K. The elapsed time for the training and test is 69 and 10.5 seconds respectively. The validation and training losses over epoch numbers are given in Figure 5.1 that clearly indicates there is no overfit observed.

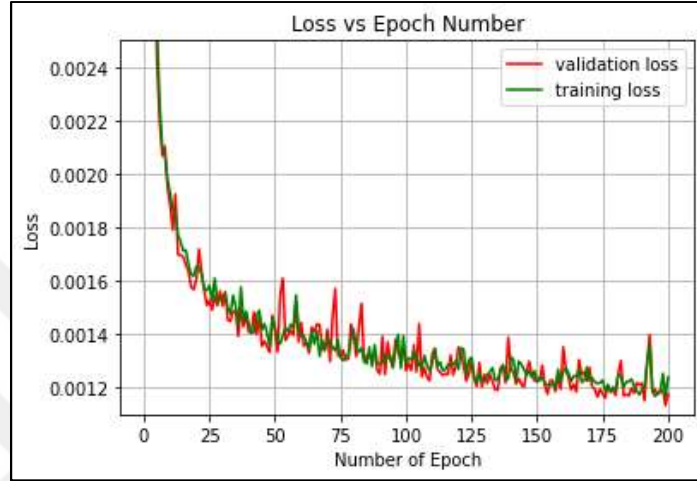


Figure 5.1. Best model training and validation loss

5.2 Identification of Abnormal Fuel Consumption

As explained in Chapter 4.5., the three-sigma boundary methodology has been followed to identify the flights with abnormal fuel burn. Three-sigma boundary for Model 8 has been calculated in Appendix J. It can be seen below;

$$\sigma_{Model\ 8} = 0.01530$$

$$MPE = -0.00085$$

$$3\sigma_{Model\ 8} = [-0.04676, 0.04505]$$

Additionally, a histogram chart of percentage error has been plotted. The normal distribution and three-sigma boundaries could be seen in Figure 5.2.

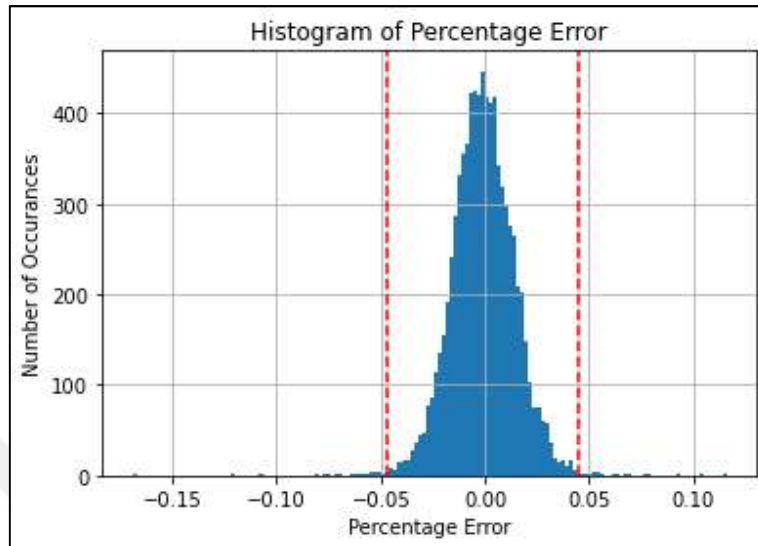


Figure 5.2. Histogram of Percentage Error

It is simple to detect flights with PE error beyond this boundary and corresponding Flight ID. In MS Excel, corresponding PEs have been calculated and the flights beyond the boundaries have been filtered. Over 10212 flights, only 55 flights are out of this boundary.

In order to examine the assumption, three random flights beyond the boundaries have been investigated from the FDM system “Aerobytes” in means of the below criteria;

- Improper pilot technique during climb and descent,
- Turbulence/Wind Shear conditions,
- Holding/Excessive Air Traffic in the terminal area,
- Instrument fault on the aircraft,
- Flight data defect.

Information that belongs to the randomly selected flights is given in Table 5.3.

Table 5.3 Flights to be investigated

Index	Aircraft Type	Period	Estimated Fuel Burn (kg)	Actual Fuel Burn (kg)	Percentage Error (%)
Flight 1	B738	June 2019	5853.7	6200.8	5.6
Flight 2	B738	Dec 2019	6664.8	6201.9	-7.4
Flight 3	B738	Jul 2019	8148.8	8614.1	5.4

Model 8 estimated the fuel burn as 5.6% less than the actual fuel burn for the flight 1. When the FDM data quality is investigated, no defective data was encountered. The climb and cruise profile are usual and there was no windshear/turbulence condition according to the aircraft's weather radar (No "WINDSHEAR CAUTION"). However, the flight profile exhibits unordinary behavior during the descent. Near 15000 ft, the aircraft started to gain altitude and climbed to 16000 ft instead of level-off or descent. This could be caused by pilot trim to decrease the speed. More importantly, after that, the landing gear has been extended at an altitude of 14000 ft.

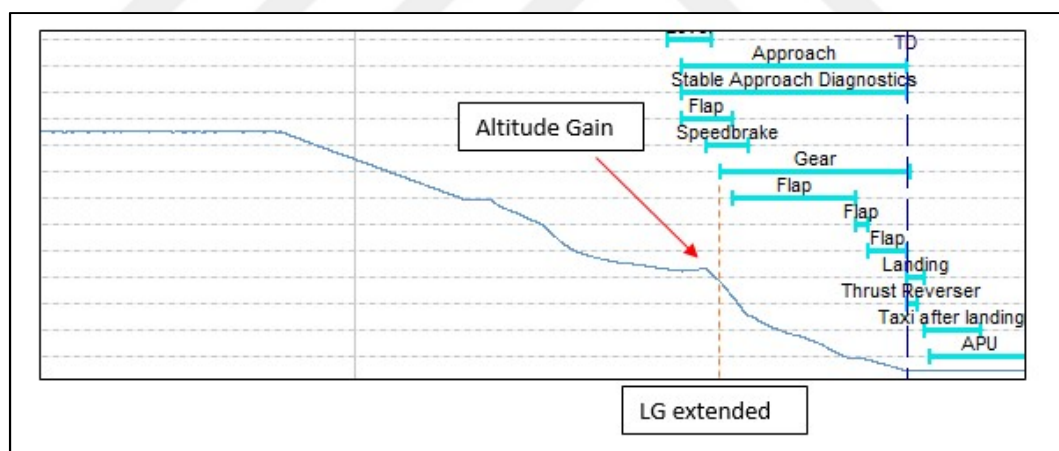


Figure 5.3. Flight Profile of Flight 1

As it is stated in Boeing 737 NG/MAX Flight Crew Training Manual [49], for instrumental approaches landing gear should be extended just after the localizer capture or 2 nm before the Final Approach Fix (FAF). For visual approaches, the landing gear should be extended at Minimum Decision Altitude/Height MDA(H).

In this flight, LOC has been captured at 2550 ft, therefore the landing gear was extended 8.5 minutes earlier. According to Boeing FPPM [50], a minute with landing gear extended results in approximately 40 kg of additional fuel burn in the descent. Therefore, the pilot action caused 340 kg additional fuel burn. This phenomenon explains the model's deviation in means of fuel consumption.

For flight 2, the model predicted that the aircraft should have been burning 6664.8 kg of fuel. However, the FDM data indicates only 6201.9 kg of fuel has been consumed. In order to confirm this actual fuel burn, the flight log for this flight has been investigated. The pilot reported that 6600 kg of fuel has been burned during the operation. This difference might be caused by the fuel slush in the tanks. The calculation by FDM has been performed based on exact moments that states start and end. At these moments, the fuel can be slushed and the equipment that measures fuel quantity may give false measurements. Therefore, pilot logs have been investigated as a primary source of fuel consumption information. The Electronic Flight Folder (EFF) is used for the data connection between the pilot and ground crew. The flight information package includes essential information such as flight plans, weather information, and NOTAMs. On the other side, pilots are responsible to record trip information such as fuel, duration, number of passengers on board, and RVSM records. The data has been acquired from EFF log is given in Figure 5.4. It indicates the FDM records are mistaken due to the false measurements of the equipment. The actual fuel burn is very close to the estimation made by the model.

Post Flight Fuel and Landing	
Fuel on Blocks (Kg):	4400
Fuel Used (Kg):	6600

Figure 5.4. Flight 2 EFF log

Last, the flight 3 has been investigated through FDM. The estimated fuel burn for the flight is 8148.9 kg whereas the actual fuel burn is 5.4% higher. When the flight profile has been analyzed, a go-around during the landing draws attention. The flight profile is given in Figure 5.5. Furthermore, the flight path of the landing phase could be seen below where go-around causes additional 14 minutes duration on landing.

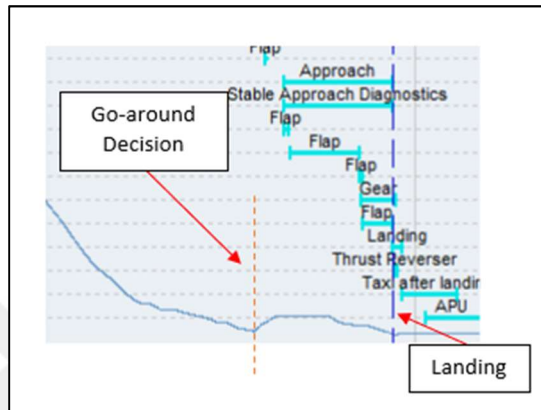


Figure 5.5. Flight profile of Flight 3



Figure 5.6. The flight path of the landing phase

The fuel quantity indicated by FDM is 3770 kg whereas the landing fuel quantity is 3180 kg. It shows the go-around methodology caused 490 kg of additional fuel burn on the entire trip. The difference between the model prediction and actual consumption (450 kg) can be explained by this go-around.



CHAPTER 6

CONCLUSIONS AND DISCUSSION

The methodology outlined in this study has developed a neural network model as a surrogate model using OFP simulations as the training dataset. A portion of the actual flight data gathered by QAR has been used for tuning hyperparameters over a spectrum. The models have been sorted based on the minimum validation losses and the best ten models have been selected. Later, the best ten have been tested with the rest of the QAR data. The best of the best ten has been selected by comparing the models in means of their statistical accuracies. Model 8 has shown the highest accuracy with 1.17% MAPE, -0.85% MRE and 0.9973 R^2 .

6.1 Discussion

The proposed model has been compared with the existing studies in means of accuracy. However, it is difficult to match existing models to this study. The purpose and structure of the proposed model show a difference from the others. Firstly, one of the data sources has been used to train the model while the other is used for testing. Additionally, there is no flight segmentation as it is aimed to estimate the total trip fuel instead of the fuel consumption or fuel flow rate in different flight segments. Furthermore, the aircraft type, origin-destination airport pairs, and global air traffic situation in the data might be different than the data the research in the literature has used.

Fortunately, some of the previous neural network/genetic algorithm models could be compared to the purposed model. For example, Trani et al. [19], split the performance manual data of an aircraft by half for training and test. The estimated

fuel flow with MRE of 1.026%-1.423% for different flight segments. [20] have trained their model with actual flight data and achieved 0.8% and 1.9% relative error for the entire flight and cruise respectively. On the other hand, Baklacioglu's [21] model has been trained and tested with flight data and estimates fuel flow in the cruise segment with $R^2=0.95492$. The overall estimation accuracy is about 1%. Finally, Uzun, Demirezen & Inalhan [22] trained their model with actual flight data and the equations of fuel flow dynamics. They obtained the model results with 1.568% MRE for the entire flight mission.

Additionally, a physics-based surrogate model has been enhanced with aircraft performance data in [15]. Yanto & Liem [15] integrated BADA trajectory simulation results into their model to increase the accuracy. They obtained a 6% of relative error when compared to the actual fuel burn data. In the end, the study [16] proposed a statistical regression model with NASPAC flight plan data and actual flight and operational data. They achieved a model with an error of $R^2=0.993$ for entire flight mission.

In Table 5.3, previous and proposed models and their accuracy performance have been tabulated.

Table 6.1 Existing Studies

Reference	Model	Target	Segment	MRE (%)	R^2
[19]	Neural Network	Fuel Burn (lb)	Climb	1.026	-
		Fuel Burn (lb)	Descent	1.423	-
[20]	Neural Network	Fuel Flow (kg/h)	Entire Flight	0.8	0.9995
		Fuel Flow (kg/h)	Cruise	1.9	0.88
[21]	Neural Network	Fuel Flow	Climb	0.02	0.99988
		Fuel Flow	Cruise	-	0.95492
		Fuel Flow	Descent	0.745	0.98976
[22]	Neural Network	Fuel Flow (kg/h)	Entire Flight	1.568	-
[15]	Surrogate Model	Fuel Burn	Entire Flight	6	-
[16]	Surrogate Model	Fuel Burn	Entire Flight	-	0.993
Model 8	NN-SM	Fuel Burn (kg)	Entire Flight	0.85	0.9973

In summary, in this study, building an accurate, easy-to-use, and cost-effective fuel efficiency model is targeted. It is clear that the proposed model's results are significantly better than most of the models in the literature. Although the entire flight profile has been analyzed with only five fundamental parameters, it has even presented better accuracy than some of the estimations in a specific flight phase. Additionally, Table 5.3 dramatically indicates the existing models are not being sufficient while modeling the fuel consumption in flight phases other than the cruise. The reason behind the question “Why does Model 8 show better performance for the entire flight section?”, might be explained by the altitude parametrization strategy. Parametrization method A considers the entire flight as the cruise section where the climb and descent phases are neglected. Parametrization method B considers and includes every phase. Out of the ten best models, only one of them is using Parameter A and another one is using maximum altitude. The other eight models use Parameter B as their altitude parameter. Therefore, the top ten models explain the importance of altitude parametrization itself.

Finally, the abnormal fuel consumption identification methodology detected 55 flights over 10212 samples. These flights are identified by checking if the percentage error is within the three-sigma boundaries. The flights whose PE is beyond the boundary are called flights with abnormal fuel consumption. Three flights were sampled over the 55 flights. Their FDM data were subjected to a detailed inspection. The non-normal situations in these flights have been explained in Section 5.2. Therefore, it can be concluded that the identification methodology can be successfully employed.

6.2 Future Work

One of the purposes of the study is to introduce an easy-to-build and open-access model to the researchers, air traffic shareholders, and airlines. Although the model built is only generic for the Boeing 737-800 aircraft type, the proposed model methodology can be used for different aircraft types. If there can be a sufficient

model database for all commercial aircrafts, fuel burn and emission calculations over a period or airspace can be done. Additionally, airlines can plan their financial strategy by using the model. It is recommended to follow the altitude parametrization methodology to achieve better accuracy.

On the other hand, the model and abnormal fuel consumption identification strategy can be advanced by making them a part of the fuel efficiency management measures. If the model detects a non-normal flight, an advanced analysis of their FDM data may reveal false maintenance practices, pilot errors, false procedure management, and even false air traffic management strategies.



REFERENCES

- [1] V. Singh and S. K. Sharma, "Fuel consumption optimization in air transport: a review, classification, critique, simple meta-analysis, and future research implications," *European Transport Research Review*, vol. 7, no. 12, 2015.
- [2] N. Hong and L. Li, "A Data-Driven Fuel Consumption Estimation Model for Airspace Redesign Analysis," in *IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, Hong Kong, 2018.
- [3] B. Beers, "https://www.investopedia.com," 29 12 2021. [Online]. Available: <https://www.investopedia.com/ask/answers/040715/what-are-major-expenses-affect-companies-airline-industry.asp>.
- [4] A. J. Stolzer, "Fuel Consumption Modeling of a Transport Category Aircraft: A Flight Operations Quality Assurance (FOQA) Analysis," *Journal of Air Transportation*, vol. 8, 2003.
- [5] B. P. Collins, "Estimation of Aircraft Fuel Consumption," *AIAA J. Aircraft*, vol. 19, no. 11, pp. 969-975, 1982.
- [6] D. I. Gibbons and G. C. McDonald, "Constrained Regression Estimates of Technology Effects on Fuel Economy," *Journal of Quality Technology*, vol. 31, no. 2, pp. 235-245, 1999.
- [7] T. Qian-qian, C. Jing-jie and D. Jian-kang, "A Method of Constructing the Fuel Efficiency Model Based on Quadratic Polynomial Regression," *Procedia Engineering*, vol. 15, pp. 3749-3753, 2011.
- [8] H. Khadilkar and H. Balakrishnan, "Estimation of aircraft taxi fuel burn using flight data recorder archives," *Transportation Research Part D*, vol. 12, p. 532–537, 2012.

- [9] D. Poles, "Base of Aircraft Data (BADA) Aircraft Performance Modelling Report," Eurocontrol, Bruxelles, 2009.
- [10] M. A. a. U. SAGE v1.5 Validation Assesment, "Andrew Malwitz; Brian Kim; Gregg Fleming; Joosung Lee; Sathya Balasubramanian; Ian Waitz and Kelly KlimaIan Waitz; Kelly Klima; Maryalice Locke; Curtis Holsclaw; Angel Morales; Edward McQueen; Warren Gillette," Federal Aviation Administration Office of Environment and Energy, Washington, DC, 2005.
- [11] Z. Zou, "A Computer Model to Estimate Commercial Aviation Fuel Consumption and Emissions in the Continental United States," Virginia Polytechnic Institute and State University - MSc Thesis, Blacksburg, Virginia, 2012.
- [12] G. B. Chatterji, "Fuel Burn Estimation Using Real Track Data," in *11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, Virginia Beach, VA, 2011.
- [13] R. D. Oaks and M. Paglione, "Prototype Implementation and Concept Validation of a 4-D Trajectory Fuel Burn Model Application," in *AIAA Guidance, Navigation, and Control Conference*, Toronto, Ontario, 2010.
- [14] W. Yamazaki and D. J. Mavriplis, "Derivative-Enhanced Variable Fidelity Surrogate Modeling for Aerodynamic Functions," *AIAA Journal*, vol. 51, no. 1, pp. 126-137, 2013.
- [15] J. Yanto and R. P. Liem, "Aircraft fuel burn performance study: A data-enhanced modeling approach," *Transportation Research Part D*, vol. 65, pp. 574-595, 2018.
- [16] M. S. Ryerson, M. Hansen and J. Bonn, "Fuel Consumption and Operational Performance," in *Ninth USA/Europe Air Traffic Management Research and Development Seminar (ATM2011)*, 2011.

- [17] G. D. Schilling, "MODELING AIRCRAFT FUEL CONSUMPTION WITH A NEURAL NETWORK," Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1997.
- [18] W. H. Cheung, "Neural Network Aided Aviation Fuel Consumption Modeling," Virginia Polytechnic Institute and State University Master of Science thesis in Civil Engineering, Virginia, 1997.
- [19] A. Trani, F. Wing-Ho, G. Schilling, H. Baik and A. Seshadri, "A Neural Network Model to Estimate Aircraft Fuel Consumption," in *AIAA 4th Aviation Technology, Integration and Operations (ATIO) Forum*, Chicago, 2004.
- [20] S. Baumann and U. Klingauf, "Modeling of aircraft fuel consumption using machine learning algorithms," *CEAS Aeronautical Journal*, 2018.
- [21] T. Baklacioglu, "Modeling the fuel flow-rate of transport aircraft during flight phases using genetic algorithm-optimized neural networks," *Aerospace Science and Technology*, vol. 49, pp. 52-62, 2016.
- [22] M. Uzun, M. U. Demirezen and G. Inalhan, "Physics Guided Deep Learning for Data-Driven Aircraft Fuel Consumption Modeling," *aerospace*, vol. 8, no. 44, 2021.
- [23] I. C. A. Organization, "Annex 6, Part I, Operation of Aircraft," INTERNATIONAL CIVIL AVIATION ORGANIZATION, Montreal, Quebec, 2018.
- [24] E. U. A. S. Agency, "Easy Access Rules for Air Operations (Regulation (EU) No 965/2012)," EASA eRules, 2019.
- [25] A. Ltd., Aerobytes FDM User Guide, Surrey: Ltd., Aerobytes, 2018.

- [26] B. F. O. E. Department, "Operational Performance and Technical Methods; Part C, Section 6 - Airplane Performance Monitoring," The Boeing Company, Seattle, Washington, 2017.
- [27] ICAO, Annex 2, Rules of the Air, Montreal, Quebec: International Civil Aviation Organization, 2005.
- [28] I. C. A. Organization, Doc 4444, Air Traffic Management, Montréal, Quebec: International Civil Aviation Organization, 2016.
- [29] I. C. A. Organization, "Annex 6, Operation of Aircraft," ICAO, Quebec, 2018.
- [30] M. D. Zio, N. Fursova, T. Gelsema, S. Gießing, U. Guarnera, J. Petrauskienė, L. Q. Kalben, M. Scanu, K. t. Bosch, M. v. d. Loo and K. Walsdorfer, "Methodology for data validation 1.0," DE, 2016.
- [31] M. Ziemann, Y. Eren and A. El-Osta, "Gene name errors are widespread in the scientific literature," *Genome Biology*, vol. 17, no. 177, 2016.
- [32] S. Sadiq, M. Orłowska, W. Sadiq and C. Foulger, "Data Flow and Validation in Workflow Modelling," in *The 15th Australasian Database Conference*, 2004.
- [33] J. Yanto and R. P. Liem, "Efficient Fast Approximation for Aircraft Fuel Consumption for Decision-Making and Policy Analysis," in *AIAA Modeling and Simulation Technologies Conference*, Denver, 2017.
- [34] J. G.-H. Carretero, F. J. S. Nieto and R. R. Cordon, "Aircraft trajectory simulator using a three degrees of freedom aircraft point mass model," in *3rd International Conference on Application and Theory of Automation in Command and Control Systems*, 2013.
- [35] S. Raschka, "Bringing features onto the same scale," in *Python Machine Learning*, Packt Publishing, 2015, p. 110.

- [36] T. Hastie, R. Tibshirani and J. Friedman, "Model Assessment and Selection," in *The Elements of Statistical Learning*, Springer, 2017, p. 222.
- [37] G. James, D. Witten, T. Hastie and R. Tibshirani, "Resampling Methods," in *An Introduction to Statistical Learning with Applications in R*, Springer, 2021, p. 198.
- [38] C. Cordoba, "https://github.com/," Spyder Project Contributors, 2009-2021. [Online]. Available: <https://github.com/spyder-ide/spyder>.
- [39] NVIDIA, NVIDIA Corporation, [Online]. Available: <https://developer.nvidia.com/cuda-zone>. [Accessed 26 November 2021].
- [40] I. Goodfellow, Y. Bengio and A. Courville, Deep Feedforward Networks, MIT Press, 2016.
- [41] C. Marc and B. De Moor, "Hyperparameter Search in Machine Learning," in *MIC 2015: The XI Metaheuristics International Conference*, 2015.
- [42] P. Schratza, M. Jannes, E. Iturritxab, R. Jakob and B. Alexander, "Hyperparameter tuning and performance assessment of statistical and machine-learning algorithms using spatial data," *Ecological Modelling*, vol. 406, pp. 109-120, 2019.
- [43] K. J. Mykel and W. A. Tim, Algorithms for Optimization, London: The MIT Press, 2019.
- [44] S. Ruder, "An overview of gradient descent optimization algorithms," Dublin, 2017.
- [45] J. Duchi, E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, pp. 2121-2159, 2011.

- [46] D. P. Kingma and J. Lei Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," in *International Conference on Learning Representations*, 2015.
- [47] R. D. Reed and R. J. Marks II, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, London: The MIT Press, 1999.
- [48] A. Krizhevsky, S. Ilya and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, 2012.
- [49] T. B. Company, "737 NG/MAX Flight Crew Training Manual," The Booeing Company, Seattle, 2021.
- [50] F. O. Engineering, *Flight Planning and Performance Manual*,, Seattle, Washington: Boeing Commercial Airplane Group, 2010.
- [51] S. A. J, "Fuel Consumption Modeling of a Transport Category Using Flight Opreations Quality Assurance Data: A Literature Review," *Journal of Air Transportation*, vol. 7, no. 1, 2002.

APPENDICES

A. Letter of Consent for Use of Company Data

The letter of consent is attached at the back of this page.



THIS PAGE INTENTIONALLY LEFT BLANK

LETTER OF CONSENT FOR USE OF COMPANY DATA

Corendon Airlines
Safety Management Systems (SMS) Department
Turistik Hava Tařımacılık A.ř.
Güzeloluk Mah. 1879 Sok. No:148
07200 ANTALYA / TURKEY

To whom it may concern;

Corendon Airlines authorizes Oğulcan Demirhan who is working as a Flight Operations Engineer in the Engineering Department to use collected “Quick Access Recorder” data for his thesis research “IDENTIFICATION OF THE ABNORMAL FUEL CONSUMPTION IN A COMMERCIAL FLIGHT BY AN ARTIFICIAL NEURAL NETWORK SURROGATE MODEL” where the terms are defined in nondisclosure agreement signed by both parties.

Best Regards,

Selami Selçuk ÇAKMAKLI
SMS Manager

TURİSTİK HAVA TAřIMACILIK A.ř.

İSTANBUL(Head Office)
Sahrayı Cedit Mh. Atatürk Cd.
7. Blok Diküm St. No: 59/5
Kadıköy, İstanbul, TURKEY
MERNİS No:7228-9275-3469-4952
Trade Register No: 537941/485523

AMSTERDAM (Branch)
Singaporestraat 82
1175 RA Lijnden
BTW NR.
NL822330830B01

ANTALYA (Branch)
Güzeloluk Mh. 1879 Sk. No: 148
07200 Muratpařa, Antalya, TURKEY
MERNİS No:8512-1775-8562-8960
Trade Register No: 43725/45918

Tel.: +90 (242) 310 90 00
Fax.: +90 (242) 324 32 40
Web: corendonairlines.com
Email: info@corendon-airlines.com
Tax No: 871 041 8994

B. Example of an Operational Flight Plan

FLIGHT PLAN ROUTE : COMPANY ROUTE : R4
 EBBR/SOPO2M/SOPOK/Y863/ROPUV/L607/NONKO-VIBOM-NOMBO-RIDAR-GOMIG-
 VABEK/UL603/OLOTA-DISOR/UL608/FSK/UN128/IMR/G80/OKESA/OKES1J/LTFE
 M80 330/ETENO 370

GCD 1285 G/D 1399 A/D 1368 W/C P014 ISAM03 MAXS 11/TOD

ALTERNATES GMORA DIST FL MAC W/C TIME FUEL
 ALTN-1 LTBS/DLM 109 0064 110 .50 P001 0.16 1000
 DLM

ENROUTE ALTERNATE LGTS/SKG

TRIP	LTFE/BJV	8122	0311	MAX	61688	79015	65317	019767
CONT	%3	244	0007	EST	56321	66756	58634	14400
ALTN	LTBS/DLM	1000	0016					
FINAL RESERVE		1069	0030	MINIMUM DIVERSION FUEL : 2069				
MIN T/O FUEL		10435	0404					
EXTRA/ETOPS		0	0000					
TAXI/APU		204		1000 KGS	0069	0000	1	
BLOCK FUEL		10639	0404	10 KNOT	0163	0003	2	
COMMANDER EXTRA				FL 0350	0217	0001	3	
TOTAL FOB				FL 0350	0217	0001	4	

EBBR	ELEV	00175FT	AWY	WPT	FL	TAS	WIND	MCS	DISTANCE	TIME	ETA	FUEL	ACCF
GMORA	FREQ	MAC	G/S	S/ISA	VAR	REMD	ACCD	REMT	ACCT	ATA	REMF	AREMF	
SOPO2	BR421	CLB	260	02015	291	0006		0002		..	.000.3	000.3	
035		CLB	259	01M07	02E	1393	0006	0309	0002	..	.010.1	..	
SOPO2	BR422	CLB	284	36012	221	0008		0001		..	.000.2	000.5	
035		CLB	292	01M03	02E	1385	0014	0308	0003	..	.010.0	..	
SOPO2	BR413	CLB	328	35012	154	0005		0001		..	.000.1	000.6	
035		CLB	340	01M03	02E	1380	0019	0307	0004	..	.009.9	..	
SOPO2	BR414	CLB	345	36015	087	0008		0001		..	.000.1	000.7	
035		CLB	346	01M03	02E	1372	0027	0306	0005	..	.009.7	..	
SOPO2	BR415	CLB	356	36017	105	0005		0001		..	.000.1	000.8	
035		CLB	362	01M04	02E	1367	0032	0305	0006	..	.009.7	..	
SOPO2	BULUX	CLB	406	36024	105	0026		0004		..	.000.3	001.1	
037		CLB	412	01M04	02E	1341	0058	0301	0010	..	.009.3	..	
SOPO2	SOPOK	CLB	456	35026	133	0029		0005		..	.000.4	001.5	
037		CLB	477	01M02	02E	1312	0087	0256	0015	..	.008.9	..	
Y863	TOC	330	466	35024	102	0009		0001		..	.000.1	001.6	
043		CLB	474	00P01	02E	1303	0096	0255	0016	..	.008.8	..	
Y863	ETENO	330	466	35024	102	0008		0001		..	.000.0	001.7	
043		M80	474	00P01	02E	1295	0104	0254	0017	..	.008.8	..	
Y863	OGLOR	CLB	453	35022	082	0017		0002		..	.000.1	001.8	
043		CLB	459	01P05	02E	1278	0121	0252	0019	..	.008.6	..	
Y863	ROPUV	370	453	33021	082	0008		0001		..	.000.1	001.9	
043		M78	459	00P06	03E	1270	0129	0251	0020	..	.008.6	..	



C. Parsing OFP Data

```
import glob
from pyunpack import Archive
import time
import os
import shutil
import xml.etree.ElementTree as ET
import re
from datetime import datetime, timedelta
import pyodbc

start_time = time.time()
odpath=(os.getcwd()).replace("python\db_python\Fuel Scripts\SQL implemented","")
currpath=os.getcwd()
today=datetime.strftime(datetime.now(), '%Y-%#m-%#d')
def createFolder(directory):
    try:
        if not os.path.exists(directory):
            os.makedirs(directory)
    except OSError:
        print ('Error: Creating directory. ' + directory)
def xmlread(xmlfiletoparse):
    global Parameters
    global keyparams
    tree=ET.parse(xmlfiletoparse)
    root = tree.getroot()
    try:
        FPID= root.attrib['flightPlanId']
    except: FPID=""

    ns='{http://aeec.aviation-ia.net/633}'
    nsall='.{http://aeec.aviation-ia.net/633}'

    for M633SupplementaryHeader in root.iter(ns+ 'M633SupplementaryHeader'):
        for Flight in root.findall(nsall+'Flight'):
            try:
                DepTm=Flight.attrib['scheduledTimeOfDeparture']
                DepTm=list(DepTm)
                DepTime=DepTm[0:10]
                DepHour=DepTm[11:16]
                DepTim="".join(DepTime)
                DepHou="".join(DepHour)
            except:
                DepTim=""
                DepHou=""
```

```

for DepartureAirport in root.findall(nsall + 'DepartureAirport'):
    try:
        ICAOdep=DepartureAirport.find(nsall+ 'AirportICAOCODE').text
    except: ICAOdep=""
for ArrivalAirport in root.findall(nsall + 'ArrivalAirport'):
    try:
        ICAOarr=ArrivalAirport.find(nsall+ 'AirportICAOCODE').text
    except: ICAOarr=""
for Aircraft in root.findall(nsall+ 'Aircraft'):

    try:
        ACTail=Aircraft.attrib['aircraftRegistration']
    except: ACTail=""

    for AircraftModel in root.findall(nsall + 'AircraftModel'):

        try:
            ACModel=AircraftModel.find(nsall+ 'AircraftICAOTYPE').text
        except: ACModel=""

for FlightInfo in root.iter(ns + 'FlightInfo'):

    try:
        CPT=FlightInfo.attrib['captain']
    except: CPT=""
    try:
        CallSign=FlightInfo.attrib['aTCCallsign']
    except: CallSign=""
for Remarks in root.iter(ns+ 'Remarks'):
    for Remark in Remarks.iter(ns + 'Remark'):
        for Paragraph in Remark.findall(nsall+ 'Paragraph'):
            try:

                TimeHours=Paragraph.find(nsall+ 'Text').text
                TimeHours=TimeHours.replace('STD-',")
                TimeHours=TimeHours.replace("STA-", "")
                (DepHour, ArrHour)=TimeHours.split(" ",1)
                DepHourr=DepHour[:2]+":"+DepHour[2:]
                ArrHourr=ArrHour[:2]+":"+ArrHour[2:]
            except:
                DepHourr=""
                ArrHourr=""

for FlightPlanHeader in root.iter(ns+ 'FlightPlanHeader'):
    try: CruDeg=FlightPlanHeader.find(nsall+ 'PerformanceFactor').text
    except: CruDeg=""
    for RouteInformation in FlightPlanHeader.findall(ns+'RouteInformation'):

        try:

```

```

    Optim=RouteInformation.attrib['optimization']
except: Optim=""
try: RouNam=RouteInformation.attrib['routeName']
except: RouNam=""

try: RouteD=RouteInformation.find(nsall+ 'RouteDescription').text
except: RouteD=""

FLS=[]
try:
    for FL in re.finditer("F", RouteD):
        if RouteD[FL.end():FL.end()+3].isnumeric() == True:
            FLS=int(RouteD[FL.end():FL.end()+3])
            if FLS % 10 == 0:
                FLS.append(FLS)
    FLS=list(dict.fromkeys(FLS))
    Flev=max(FLS)
except: Flev=""

for GroundDistance in RouteInformation.findall(ns+'GroundDistance'):
    try: GrnD=GroundDistance.find(nsall+ 'Value').text
    except: GrnD=""
for AirDistance in RouteInformation.findall(ns+'AirDistance'):
    try: AirD=AirDistance.find(nsall+ 'Value').text
    except: AirD=""
for GreatCircleDistance in RouteInformation.findall(ns+'GreatCircleDistance'):
    try: GCD=GreatCircleDistance.find(nsall+ 'Value').text
    except: GCD=""

for FuelHeader in root.iter(ns+ 'FuelHeader'):
    for TripFuel in FuelHeader.findall(ns+ 'TripFuel'):
        for EstimatedWeight in TripFuel.findall(ns+ 'EstimatedWeight'):
            try: EstW=EstimatedWeight.find(nsall+ 'Value').text
            except: EstW=""

        for Duration in TripFuel.findall(ns+ 'Duration'):
            try:
                RawDrt=Duration.find(nsall+ 'Value').text
                CntHr=RawDrt.find("H")
                CntT=RawDrt.find("T")
                CntMin=RawDrt.find("M")
                if CntMin==-1:
                    Min="00"
                    if CntHr==CntT+2:
                        Hr="0"+RawDrt[CntT+1]
                    else:
                        Hr=RawDrt[CntT+1:CntT+3]
                elif CntHr==-1:
                    Hr="00"

```

```

        if CntMin==CntT+2:
            Min="0"+RawDrt[CntT+1]
        else:
            Min=RawDrt[CntT+1:CntT+3]
    elif CntHr==CntT+2:
        Hr="0"+RawDrt[CntT+1]
        if CntHr==CntMin-2:
            Min="0"+RawDrt[CntHr+1]
        else:
            Min=RawDrt[CntHr+1:CntHr+3]
    elif CntHr==CntT+3:
        Hr=RawDrt[CntT+1:CntT+3]
        if CntHr==CntMin-2:
            Min="0"+RawDrt[CntHr+1]
        else:
            Min=RawDrt[CntHr+1:CntHr+3]
    Drt=Hr+ ":" + Min
except: Drt=""
for ContingencyFuel in FuelHeader.findall(ns+ 'ContingencyFuel'):
    for EstimatedWeight in ContingencyFuel.findall(nsall + 'EstimatedWeight'):
        try: CEstW=EstimatedWeight.find(nsall+ 'Value').text
        except: CEstW=""

for Duration in ContingencyFuel.findall(ns+ 'Duration'):
    try:
        RawDrt=Duration.find(nsall+ 'Value').text
        CntHr=RawDrt.find("H")
        CntT=RawDrt.find("T")
        CntMin=RawDrt.find("M")
        if CntMin==-1:
            Min="00"
            if CntHr==CntT+2:
                Hr="0"+RawDrt[CntT+1]
            else:
                Hr=RawDrt[CntT+1:CntT+3]
        elif CntHr==-1:
            Hr="00"
            if CntMin==CntT+2:
                Min="0"+RawDrt[CntT+1]
            else:
                Min=RawDrt[CntT+1:CntT+3]
        elif CntHr==CntT+2:
            Hr="0"+RawDrt[CntT+1]
            if CntHr==CntMin-2:
                Min="0"+RawDrt[CntHr+1]
            else:
                Min=RawDrt[CntHr+1:CntHr+3]
        elif CntHr==CntT+3:
            Hr=RawDrt[CntT+1:CntT+3]

```

```

        if CntHr==CntMin-2:
            Min="0"+RawDrt[CntHr+1]
        else:
            Min=RawDrt[CntHr+1:CntHr+3]
        cDrt=Hr+ ":" + Min
    except: cDrt=""

for ContingencyPolicy in ContingencyFuel.findall(ns+ 'ContingencyPolicy'):
    try: ContPol=ContingencyPolicy.attrib['policyName']
    except: ContPol=""

for AlternateFuels in FuelHeader.findall(ns+ 'AlternateFuels'):
    for AlternateFuel in AlternateFuels.findall(nsall + 'AlternateFuel'):
        for EstimatedWeight in AlternateFuel.findall(nsall + 'EstimatedWeight'):
            try: AEstW=EstimatedWeight.find(nsall+ 'Value').text
            except: AEstW=""

        for Duration in AlternateFuel.findall(ns+ 'Duration'):
            try:
                RawDrt=Duration.find(nsall+ 'Value').text
                CntHr=RawDrt.find("H")
                CntT=RawDrt.find("T")
                CntMin=RawDrt.find("M")
                if CntMin==-1:
                    Min="00"
                    if CntHr==CntT+2:
                        Hr="0"+RawDrt[CntT+1]
                    else:
                        Hr=RawDrt[CntT+1:CntT+3]
                elif CntHr==-1:
                    Hr="00"
                    if CntMin==CntT+2:
                        Min="0"+RawDrt[CntT+1]
                    else:
                        Min=RawDrt[CntT+1:CntT+3]
                elif CntHr==CntT+2:
                    Hr="0"+RawDrt[CntT+1]
                    if CntHr==CntMin-2:
                        Min="0"+RawDrt[CntHr+1]
                    else:
                        Min=RawDrt[CntHr+1:CntHr+3]
                elif CntHr==CntT+3:
                    Hr=RawDrt[CntT+1:CntT+3]
                    if CntHr==CntMin-2:
                        Min="0"+RawDrt[CntHr+1]
                    else:
                        Min=RawDrt[CntHr+1:CntHr+3]
                aDrt=Hr+ ":" + Min
            except: aDrt=""

```

```

for FinalReserve in FuelHeader.findall(ns+ 'FinalReserve'):
    for EstimatedWeight in FinalReserve.findall(nsall + 'EstimatedWeight'):
        try: FEstW=EstimatedWeight.find(nsall+ 'Value').text
        except: FEstW=""

    for Duration in FinalReserve.findall(ns+ 'Duration'):
        try:
            RawDrt=Duration.find(nsall+ 'Value').text
            CntHr=RawDrt.find("H")
            CntT=RawDrt.find("T")
            CntMin=RawDrt.find("M")
            if CntMin==-1:
                Min="00"
                if CntHr==CntT+2:
                    Hr="0"+RawDrt[CntT+1]
                else:
                    Hr=RawDrt[CntT+1:CntT+3]
            elif CntHr==-1:
                Hr="00"
                if CntMin==CntT+2:
                    Min="0"+RawDrt[CntT+1]
                else:
                    Min=RawDrt[CntT+1:CntT+3]
            elif CntHr==CntT+2:
                Hr="0"+RawDrt[CntT+1]
                if CntHr==CntMin-2:
                    Min="0"+RawDrt[CntHr+1]
                else:
                    Min=RawDrt[CntHr+1:CntHr+3]
            elif CntHr==CntT+3:
                Hr=RawDrt[CntT+1:CntT+3]
                if CntHr==CntMin-2:
                    Min="0"+RawDrt[CntHr+1]
                else:
                    Min=RawDrt[CntHr+1:CntHr+3]
            FDrt=Hr+ ":" + Min
        except: FDrt=""

    for TakeOffFuel in FuelHeader.findall(ns+ 'TakeOffFuel'):
        for EstimatedWeight in TakeOffFuel.findall(nsall + 'EstimatedWeight'):
            try: TOEstW=EstimatedWeight.find(nsall+ 'Value').text
            except: TOEstW=""

    for Duration in TakeOffFuel.findall(ns+ 'Duration'):
        try:
            RawDrt=Duration.find(nsall+ 'Value').text
            CntHr=RawDrt.find("H")
            CntT=RawDrt.find("T")

```

```

CntMin=RawDrt.find("M")
if CntMin== -1:
    Min="00"
    if CntHr==CntT+2:
        Hr="0"+RawDrt[CntT+1]
    else:
        Hr=RawDrt[CntT+1:CntT+3]
elif CntHr== -1:
    Hr="00"
    if CntMin==CntT+2:
        Min="0"+RawDrt[CntT+1]
    else:
        Min=RawDrt[CntT+1:CntT+3]
elif CntHr==CntT+2:
    Hr="0"+RawDrt[CntT+1]
    if CntHr==CntMin-2:
        Min="0"+RawDrt[CntHr+1]
    else:
        Min=RawDrt[CntHr+1:CntHr+3]
elif CntHr==CntT+3:
    Hr=RawDrt[CntT+1:CntT+3]
    if CntHr==CntMin-2:
        Min="0"+RawDrt[CntHr+1]
    else:
        Min=RawDrt[CntHr+1:CntHr+3]
TODrt=Hr+ ":" + Min
except: TODrt=""

```

```

for TaxiFuel in FuelHeader.findall(ns+ 'TaxiFuel'):
    for EstimatedWeight in TaxiFuel.findall(nsall + 'EstimatedWeight'):
        try: TaEstW=EstimatedWeight.find(nsall+ 'Value').text
        except: TaEstW=""

```

```

for BlockFuel in FuelHeader.findall(ns+ 'BlockFuel'):
    for EstimatedWeight in BlockFuel.findall(nsall + 'EstimatedWeight'):
        try: BEstW=EstimatedWeight.find(nsall+ 'Value').text
        except: BEstW=""

```

```

for Duration in BlockFuel.findall(ns+ 'Duration'):
    try:
        RawDrt=Duration.find(nsall+ 'Value').text
        CntHr=RawDrt.find("H")
        CntT=RawDrt.find("T")
        CntMin=RawDrt.find("M")
        if CntMin== -1:
            Min="00"
            if CntHr==CntT+2:
                Hr="0"+RawDrt[CntT+1]
            else:

```

```

        Hr=RawDrt[CntT+1:CntT+3]
    elif CntHr==-1:
        Hr="00"
        if CntMin==CntT+2:
            Min="0"+RawDrt[CntT+1]
        else:
            Min=RawDrt[CntT+1:CntT+3]
    elif CntHr==CntT+2:
        Hr="0"+RawDrt[CntT+1]
        if CntHr==CntMin-2:
            Min="0"+RawDrt[CntHr+1]
        else:
            Min=RawDrt[CntHr+1:CntHr+3]
    elif CntHr==CntT+3:
        Hr=RawDrt[CntT+1:CntT+3]
        if CntHr==CntMin-2:
            Min="0"+RawDrt[CntHr+1]
        else:
            Min=RawDrt[CntHr+1:CntHr+3]
    BDrt=Hr+ ":" + Min
except: BDrt=""

for ArrivalFuel in FuelHeader.findall(ns+ 'ArrivalFuel'):
    for EstimatedWeight in ArrivalFuel.findall(nsall + 'EstimatedWeight'):
        try: AFestW=EstimatedWeight.find(nsall+ 'Value').text
        except: AFestW=""

for Duration in ArrivalFuel.findall(ns+ 'Duration'):
    try:
        RawDrt=Duration.find(nsall+ 'Value').text
        CntHr=RawDrt.find("H")
        CntT=RawDrt.find("T")
        CntMin=RawDrt.find("M")
        if CntMin==-1:
            Min="00"
            if CntHr==CntT+2:
                Hr="0"+RawDrt[CntT+1]
            else:
                Hr=RawDrt[CntT+1:CntT+3]
        elif CntHr==-1:
            Hr="00"
            if CntMin==CntT+2:
                Min="0"+RawDrt[CntT+1]
            else:
                Min=RawDrt[CntT+1:CntT+3]
        elif CntHr==CntT+2:
            Hr="0"+RawDrt[CntT+1]
            if CntHr==CntMin-2:
                Min="0"+RawDrt[CntHr+1]

```

```

        else:
            Min=RawDrt[CntHr+1:CntHr+3]
        elif CntHr==CntT+3:
            Hr=RawDrt[CntT+1:CntT+3]
            if CntHr==CntMin-2:
                Min="0"+RawDrt[CntHr+1]
            else:
                Min=RawDrt[CntHr+1:CntHr+3]
        AFDrt=Hr+ ":" + Min
    except: AFDrt=""

for WeightHeader in root.iter(ns + 'WeightHeader'):
    for DryOperatingWeight in WeightHeader.findall(ns+ 'DryOperatingWeight'):
        for EstimatedWeight in DryOperatingWeight.findall(ns+ 'EstimatedWeight'):
            try: DOW=EstimatedWeight.find(nsall+ 'Value').text
            except: DOW=""

    for Load in WeightHeader.findall(ns+ 'Load'):
        for EstimatedWeight in Load.findall(ns+ 'EstimatedWeight'):
            try: Payload=EstimatedWeight.find(nsall+ 'Value').text
            except: Payload=""

    for ZeroFuelWeight in WeightHeader.findall(ns+ 'ZeroFuelWeight'):
        for EstimatedWeight in ZeroFuelWeight.findall(ns+ 'EstimatedWeight'):
            try: ZFW=EstimatedWeight.find(nsall+ 'Value').text
            except: ZFW=""

    for TakeoffWeight in WeightHeader.findall(ns+ 'TakeoffWeight'):
        for EstimatedWeight in TakeoffWeight.findall(ns+ 'EstimatedWeight'):
            try: TOW=EstimatedWeight.find(nsall+ 'Value').text
            except: TOW=""

    for LandingWeight in WeightHeader.findall(ns+ 'LandingWeight'):
        for EstimatedWeight in LandingWeight.findall(ns+ 'EstimatedWeight'):
            try: LDGWT=EstimatedWeight.find(nsall+ 'Value').text
            except: LDGWT=""
    DepDateTime= DepTim + " " + DepHou
    DepDateTimesTime=datetime.strptime(DepDateTime, '%Y-%m-%d %H:%M')
    DepDateasstr=datetime.strftime(DepDateTimesTime, '%Y-%m-%d')
    FPID=FPID.lstrip("0")
    KeyFP=FPID+ "_" + DepDateasstr
    Parameters=[FPID, DepDateTime, ICAOdep, ICAOarr, ACTail, ACModel, CruDeg,
    CPT, CallSign, DepHourr, ArrHourr, Optim,
    RouNam, Flev, GrnD, AirD, GCD, EstW, Drt, CEstW, cDrt, AEstW, aDrt,
    FEstW, FDrt, TOEstW, TODrt, TaEstW,
    BEstW, BDrt, AFEstW, AFDrt, DOW, Payload, ZFW, TOW, LDGWT, KeyFP]
    keyparams=[DepDateTime, ICAOdep, ICAOarr, ACTail]
    dlt = r'C:\Users\Engineering\Desktop\Engineering\dummyFP'
    conn=pyodbc.connect('Driver={SQL Server};'

```

```

        'Server=localhost;'
        'Database=FuelMasterDB;'
        'Trusted_Connection=yes;')

n=0
succ=0
unsucc=0
for iii in range(3):
    yesterday=datetime.strftime(datetime.now() - timedelta(iii), '%Y%m%d')
    flts=glob.glob(r'C:\FliteBrief\Archive/' + str(yesterday)+'/*')
    flts.sort()
    print(flts)
    if len(flts)>0:
        for k in range(len(flts)):
            try:
                createFolder(dlt)
                Archive(flts[k]).extractall(dlt)
                datfile=glob.glob(dlt+'/*.dat')
                datfile=datfile[0]
                Archive(datfile).extractall(dlt)
                xmlfile=glob.glob(dlt+'/*.xml')
                for cntXML in range(len(xmlfile)):
                    if 'OFP' in xmlfile[cntXML]:
                        xmlfiletoparse=xmlfile[cntXML]
                xmlread(xmlfiletoparse)
                keyfp0=Parameters[-1]
                cur=conn.cursor()
                cur.execute("SELECT * from dbo.FlightPlans WHERE Date=? AND
DepartureAirport=? AND ArrivalAirport=? AND Aircraft=? ", keyparams)
                ftcur=cur.fetchone()
                if not ftcur:
                    cur.execute("""INSERT INTO dbo.FlightPlans
VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?),
Parameters)
                    conn.commit()
                else:
                    cur.execute("DELETE FROM dbo.FlightPlans WHERE Date=? AND
DepartureAirport=? AND ArrivalAirport=? AND Aircraft=?", keyparams)
                    cur.execute("""INSERT INTO dbo.FlightPlans
VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?),
Parameters)
                    conn.commit()
                succ+=1
            except Exception as exceptionnn:
                unsucc+=1
                with open(currpath + "\LOG" + "FP_LOG"+today+".txt", "a") as file:
                    file.write(str(flts[k])+"\n" + str(exceptionnn) + "\n")
            finally:
                shutil.rmtree(dlt)
with open(currpath + "\LOG" + "FP_LOG"+today+".txt", "a") as file:

```

```

file.write("""
    unsucc: """+ str(unsucc)+ "succ: "+ str(succ)+""""
    *****\n""")
print(n)
elapsed_time = time.time() - start_time
print(elapsed_time)
conn.commit()
conn.close()

```



D. Parsing OFP Waypoint Data

```
from pyunpack import Archive
import os
import shutil
import xml.etree.ElementTree as ET
from datetime import datetime, timedelta
import pyodbc

def createFolder(directory):
    try:
        if not os.path.exists(directory):
            os.makedirs(directory)
    except OSError:
        print ('Error: Creating directory. ' + directory)

def waypoints(path):
    global param
    tree=ET.parse(path)
    root = tree.getroot()
    try:
        FPID= root.attrib['flightPlanId']
    except: FPID=""
    ns='{http://aeec.aviation-ia.net/633}'
    nsall='.{http://aeec.aviation-ia.net/633}'
    for M633SupplementaryHeader in root.iter(ns+ 'M633SupplementaryHeader'):
        for Flight in root.findall(nsall+'Flight'):
            try:
                DepTm=Flight.attrib['scheduledTimeOfDeparture']
                DepTm=list(DepTm)
                DepTime=DepTm[0:10]
                DepTim=""
                DepTim=""
            except:
                DepTim=""
    for Waypoints in root.iterfind(ns + 'Waypoints'):
        for Waypoint in Waypoints.iterfind(ns+'Waypoint'):
            Mach=""
            TFPW=""
            CFT=""
            Burnoff=""
            CBurnoff=""
            try:
                SequenceId=Waypoint.attrib['sequenceId']
                WaypointName=Waypoint.attrib['waypointName']
            except:
                SequenceId=""
                WaypointName=""
            for Coordinates in Waypoint.iterfind(ns +'Coordinates'):
                try:
                    longi=Coordinates.attrib['longitude']
```

```

        latti=Coordinates.attrib['latitude']
    except:
        longi=""
        latti=""
for Altitude in Waypoint.iterfind(ns +'Altitude'):
    try:
        EstAlt=Altitude.find(nsall+ 'Value').text
    except:
        EstAlt=""
for MachNumber in Waypoint.iterfind(ns +'MachNumber'):
    try:
        Mach=MachNumber.find(nsall+ 'Value').text
    except:
        Mach=""
for GroundDistance in Waypoint.iterfind(ns +'GroundDistance'):
    try:
        GD=GroundDistance.find(nsall+ 'Value').text
    except:
        GD=""
for RemainingGroundDistance in Waypoint.iterfind(ns
+'RemainingGroundDistance'):
    try:
        RemGD=RemainingGroundDistance.find(nsall+ 'Value').text
    except:
        RemGD=""
for TimeFromPreviousWaypoint in Waypoint.iterfind(ns
+'TimeFromPreviousWaypoint'):
    try:
        TFPW=TimeFromPreviousWaypoint.find(nsall+ 'Value').text
    except:
        TFPW=""
for CumulatedFlightTime in Waypoint.iterfind(ns +'CumulatedFlightTime'):
    try:
        CFT=CumulatedFlightTime.find(nsall+ 'Value').text
    except:
        CFT=""
for RemainingFlightTime in Waypoint.iterfind(ns +'RemainingFlightTime'):
    try:
        RFT=RemainingFlightTime.find(nsall+ 'Value').text
    except:
        RFT=""
for BurnOff in Waypoint.iterfind(ns +'BurnOff'):
    try:
        Burnoff=BurnOff.find(nsall+ 'Value').text
    except:
        Burnoff=""
for CumulatedBurnOff in Waypoint.iterfind(ns +'CumulatedBurnOff'):
    try:
        CBurnoff=CumulatedBurnOff.find(nsall+ 'Value').text

```

```

except:
    CBurnoff=""
for FuelOnBoard in Waypoint.iterfind(ns +'FuelOnBoard'):
    try:
        FuelOnBoard=FuelOnBoard.find(nsall+ 'Value').text
    except:
        FuelOnBoard=""
for AircraftWeight in Waypoint.iterfind(ns +'AircraftWeight'):
    try:
        AircraftWeight=AircraftWeight.find(nsall+ 'Value').text
    except:
        AircraftWeight=""
FPID=FPID.lstrip("0")
KeyWyp=FPID+" _ "+DepTim
param=[SequenceId, FPID, DepTim, WaypointName, latti, longi, EstAlt, Mach,
GD, RemGD, TFPW, CFT, RFT, Burnoff, CBurnoff, FuelOnBoard, AircraftWeight,
KeyWyp]
print(param)
keywyp0=param[-1]
cur=conn.cursor()
cur.execute("SELECT * from dbo.Waypoints WHERE KeyWYP=? AND
SequenceID=?", [keywyp0, param[0]])
ftcur=cur.fetchone()
if not ftcur:
    cur.execute("""INSERT INTO dbo.Waypoints
VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?),
param)
else:
    cur.execute("DELETE FROM dbo.Waypoints WHERE KeyWYP=?",
[keywyp0])
    cur.execute("""INSERT INTO dbo.Waypoints
VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?),
param)
conn.commit()
dlt = r'C:\Users\Engineering\Desktop\Engineering\dummyWYP'
succ_read=0
fail_read=0
curpath=os.getcwd()
conn=pyodbc.connect('Driver={SQL Server};'
'Server=localhost;'
'Database=FuelMasterDB;'
'Trusted_Connection=yes;')
for cnt1 in range(3):
    yesterday=datetime.strftime(datetime.now() - timedelta(cnt1), '%Y%m%d')
    print(yesterday)
    n=0
    flts=glob.glob(r'C:\FliteBrief\Archive/' + str(yesterday)+'/*')
    flts.sort()
    for k in range(len(flts)):

```

```

try:
    createFolder(dlt)
    Archive(flts[k]).extractall(dlt)
    datfile=glob.glob(dlt+'/*.dat')
    datfile=datfile[0]
    Archive(datfile).extractall(dlt)
    xmlfile=glob.glob(dlt+'/*.xml')
    for cntXML in range(len(xmlfile)):
        if 'OFP' in xmlfile[cntXML]:
            xmlfiletoparse=xmlfile[cntXML]
            waypoints(xmlfiletoparse)
            conn.commit()
            succ_read+=1
except Exception as E:
    print(E)
    fail_read+=1
finally:
    shutil.rmtree(dlt)
conn.commit()
conn.close()
print("succ=", succ_read)
print("fail=", fail_read)

```

E. Gathering Altitude Values

```
import pyodbc
import pandas as pd
from datetime import datetime, timedelta, date
import math
import isodate
def cumulativeFB(counter):
    subAeroDB=AeroDB[(AeroDB["flight_id"]==AeroDB.iat[counter, 5]) &
AeroDB["Value_Name"].str.startswith("Fuel Burn") & (AeroDB["Value
Start"]<=AeroDB.iat[counter, -3])]
    totalburn=subAeroDB['Value'].sum()
    return totalburn
for cntday in range(5):
    today = date.today()
    specdate= datetime.strptime((today - timedelta(days=cntday)), '%Y-%m-%d
%H:%M:%S')
    print(specdate)
    specplus= datetime.strptime((today - timedelta(days=cntday) + timedelta(days=1)), '%Y-
%m-%d %H:%M:%S')
    print(specplus)
    username='aerobytes_app'
    pwd='aerobytes'
    conn155=pyodbc.connect('Driver={SQL Server};'
        'Server=10.1.0.155;'
        'Database=fdm_v03_48;'
        'UID='+username+';PWD='+pwd+';'
        )
    conn36=pyodbc.connect('Driver={SQL Server};'
        'Server=localhost;'
        'Database=FuelMasterDB;'
        'Trusted_Connection=yes;'
        )
    AeroDB=pd.read_sql_query("exec CAI_SP_GET_LEVEL_AND_FB ?, ?",conn155,
params=(specdate, specplus))
    WypDB=pd.read_sql_query("Select * From Waypoints WHERE Date=?", conn36,
params=[specdate])
    FPDB=pd.read_sql_query("""Select DepartureAirport, ArrivalAirport, Aircraft, KeyFP
        From FlightPlans WHERE Date>? AND Date<?""", conn36,
params=(specdate, specplus))
    WypDB=pd.merge(WypDB, FPDB, left_on='KeyWYP', right_on='KeyFP' )
    AeroDB=AeroDB.groupby(["A/C TO Datetime","A/C TD Datetime","aircraft_reg","TO
Airport","TD Airport","flight_id",
        "State ID","Value_Name","Value Start","Value
End"])[['Value']].agg('sum').reset_index()
    WypDB['FID']="N/A"
```

```

for cnt0 in range(len(WypDB.index)):
    parameterwyp=WypDB.iat[cnt0, -3]+WypDB.iat[cnt0, -5]+WypDB.iat[cnt0, -
4]+WypDB.iat[cnt0, 2]
    for cnt1 in range(len(AeroDB.index)):
        dummy1=datetime.strptime(AeroDB.iat[cnt1, 0], '%Y-%m-%d')
        dummy2=AeroDB.iat[cnt1, 2]
        dummy2=dummy2.replace("-", "")
        parameterae=dummy2+AeroDB.iat[cnt1, 3]+ AeroDB.iat[cnt1, 4]+dummy1
        if parameterwyp==parameterae:
            WypDB.iat[cnt0,-1]=AeroDB.iat[cnt1, 5]
            break
    DF2write=pd.DataFrame([])
    for cnt2 in range(len(AeroDB.index)):
        try:
            if AeroDB.iat[cnt2, 7]=="Altitude - average":
                if DF2write.empty:
                    dummydf1=pd.DataFrame([AeroDB.iat[cnt2, 5], AeroDB.iat[cnt2, 0 ],
"Altitude- FDM", 0])
                    dummydf2=pd.DataFrame([AeroDB.iat[cnt2, 5], AeroDB.iat[cnt2, 1 ],
"Altitude- FDM", 0])
                    DF2write=DF2write.append(dummydf1.T)
                    DF2write=DF2write.append(dummydf2.T)
                else:
                    checkFIDexist=len(DF2write[(DF2write.iloc[:, 0]==AeroDB.iat[cnt2, 5]) &
(DF2write.iloc[:, 2]=="Altitude- FDM")])
                    if checkFIDexist==0:
                        dummydf1=pd.DataFrame([AeroDB.iat[cnt2, 5], AeroDB.iat[cnt2, 0 ],
"Altitude- FDM", 0])
                        dummydf2=pd.DataFrame([AeroDB.iat[cnt2, 5], AeroDB.iat[cnt2, 1 ],
"Altitude- FDM", 0])
                        DF2write=DF2write.append(dummydf1.T)
                        DF2write=DF2write.append(dummydf2.T)
                        dummylist1=[AeroDB.iat[cnt2, 5], AeroDB.iat[cnt2, 8 ], "Altitude- FDM",
math.ceil(AeroDB.iat[cnt2, 10])]
                        dummyDF11=pd.DataFrame(dummylist1)
                        dummylist2=[AeroDB.iat[cnt2, 5], AeroDB.iat[cnt2, 9 ], "Altitude- FDM",
math.ceil(AeroDB.iat[cnt2, 10])]
                        dummyDF22=pd.DataFrame(dummylist2)
                        DF2write=DF2write.append(dummyDF11.T)
                        DF2write=DF2write.append(dummyDF22.T)
                    else:
                        if not AeroDB.iat[cnt2, 7]=="Fuel Burn - A Taxi Out":
                            dummydf2=pd.DataFrame([AeroDB.iat[cnt2, 5], AeroDB.iat[cnt2, -2 ], "Fuel
Burn- FDM" , math.ceil(cumulativeFB(cnt2))])
                            DF2write=DF2write.append(dummydf2.T)
            except Exception as E1:
                print(E1)
                pass
        for cnt3 in range(len(WypDB.index)):

```

```

try:
    aa=WypDB.iat[cnt3,11]
    try:
        duration=isodate.parse_duration(aa).total_seconds()
    except:
        duration=0
    subDB=AeroDB[AeroDB["flight_id"]==WypDB.iat[cnt3, -1]]
    TOdatetime=subDB.iat[0,0]
    instantDT=TOdatetime+ timedelta(seconds=duration)
    dummydf1=pd.DataFrame([WypDB.iat[cnt3, -1], instantDT, "Fuel Burn- FP" ,
WypDB.iat[cnt3, 14] ])
    DF2write=DF2write.append(dummydf1.T)
    subwypDB=WypDB[WypDB["KeyFP"]==WypDB.iat[cnt3,-2]]
    if WypDB.iat[cnt3, 0]==1 or WypDB.iat[cnt3, 0]==len(subwypDB):
        dummydf2=pd.DataFrame([WypDB.iat[cnt3, -1], instantDT, "Altitude- FP" , 0
])
        DF2write=DF2write.append(dummydf2.T)
    else:
        dummydf3=pd.DataFrame([WypDB.iat[cnt3, -1], instantDT, "Altitude- FP" ,
WypDB.iat[cnt3, 6]*100 ])
        DF2write=DF2write.append(dummydf3.T)
    except:
        pass
    for cnt4 in range(len(DF2write.index)):
        try:
            dfaslist=DF2write.iloc[cnt4].astype(str).tolist()
            try:
                dfaslist[1]=datetime.strptime(dfaslist[1], '%Y-%m-%d %H:%M:%S.%f')
            except:
                dfaslist[1]=datetime.strptime(dfaslist[1], '%Y-%m-%d %H:%M:%S')
            cur1=conn36.cursor()
            cur1.execute("SELECT * from dbo.Level WHERE FID=? AND InstantTime=?
AND Category=?", [dfaslist[0], dfaslist[1], dfaslist[2]])
            ftcu=cur1.fetchone()
            if not ftcu:
                cur1.execute("INSERT INTO dbo.Level VALUES(?,?,?,?)",
                    dfaslist)
                conn36.commit()
        except Exception as E3:
            print(E3)
            pass

```

F. Altitude Parametrization

```
import pyodbc
import pandas as pd
from datetime import datetime, timedelta, date
conn36=pyodbc.connect('Driver={SQL Server};'
                      'Server=localhost;'
                      'Database=FuelMasterDB;'
                      'Trusted_Connection=yes;'
                      )
query="Select * from Level Where (FID=?) and (Category='Altitude- FP' or
Category='Altitude- FDM')"
FIDdb=pd.read_sql_query("Select Distinct FID from Level Where InstantTime>'2021-06-
01'", con=conn36)
FIDdb=FIDdb.applymap(str)
df2w=pd.DataFrame([], columns=("FID", "ParamA_FDM", "ParamB_FDM",
"ParamA_FP", "ParamB_FP") )
for cnt1 in range(len(FIDdb)):
    param={FIDdb.iat[cnt1,0]}
    cnt5=0
    try:
        LVL=pd.read_sql_query(query, con=conn36, params=param)
        AltFDM=LVL[LVL["Category"]=="Altitude- FDM"]
        AltFDM=AltFDM.sort_values(by='InstantTime')
        AltFDM=AltFDM.reset_index()
        cnt2=0
        LPFDM=0
        totalcruiseduration=0
        levelparamA=0
        levelparamB=0
        totalduration=((AltFDM.iat[-1,2]-AltFDM.iat[0,2]).total_seconds())/60
        while cnt2<len(AltFDM)-1:
            tpa=AltFDM.iat[cnt2+1,2]
            ta=AltFDM.iat[cnt2,2]
            lpa=AltFDM.iat[cnt2+1,4]
            la=AltFDM.iat[cnt2,4]
            duration=((tpa-ta).total_seconds())/60
            if lpa==la:
                dummylevelparamA=lpa*duration
                totalcruiseduration=totalcruiseduration+duration
                levelparamA=levelparamA+dummylevelparamA
                dummylevelparamB=lpa*duration
                levelparamB=levelparamB+dummylevelparamB
            else:
                dummylevelparamB=(lpa+la)*duration/2
                levelparamB=levelparamB+dummylevelparamB
            cnt2+=1
        ParamA_FDM=levelparamA*totalduration/(totalcruiseduration**2)
```

```

ParamB_FDM=levelparamB/totalduration
AltFP=LVL[LVL["Category"]=="Altitude- FP"]
AltFP=AltFP.sort_values(by='InstantTime')
AltFP=AltFP.reset_index()
totalcruiseduration=0
levelparamA=0
levelparamB=0
totalduration=((AltFP.iat[-1,2]-AltFP.iat[0,2]).total_seconds())/60
cnt3=0
while cnt3<len(AltFP)-1:
    tpa=AltFP.iat[cnt3+1,2]
    ta=AltFP.iat[cnt3,2]
    lpa=AltFP.iat[cnt3+1,4]
    la=AltFP.iat[cnt3,4]
    duration=((tpa-ta).total_seconds())/60
    if lpa==la:
        dummylevelparamA=lpa*duration
        totalcruiseduration=totalcruiseduration+duration
        levelparamA=levelparamA+dummylevelparamA
        dummylevelparamB=lpa*duration
        levelparamB=levelparamB+dummylevelparamB
    else:
        dummylevelparamB=(lpa+la)*duration/2
        levelparamB=levelparamB+dummylevelparamB
    cnt3+=1
ParamA_FP=levelparamA*totalduration/(totalcruiseduration**2)
ParamB_FP=levelparamB/totalduration
dummyDF=pd.DataFrame({"FID":[FIDdb.iat[cnt1,0]],"ParamA_FDM":
[ParamA_FDM], "ParamB_FDM": [ParamB_FDM],"ParamA_FP": [ParamA_FP],
"ParamB_FP": [ParamB_FP]})
df2w=pd.concat([df2w, dummyDF])
except:
    cnt5+=1
    pass
df2w.to_excel("leveldata21hazsonrasi.xlsx")

```

G. Gathering Master DB

```
import pyodbc
import pandas as pd
from datetime import datetime, timedelta, date

conn36=pyodbc.connect('Driver={SQL Server};'
                      'Server=localhost;'
                      'Database=FuelMasterDB;'
                      'Trusted_Connection=yes;')

FP=pd.read_sql_query("Select * from FlightPlans Where Date>'2019-01-01'",
con=conn36)
FDM=pd.read_sql_query("Select * from [FDM State Values] where [A C TO
Datetime]>'2019-01-01'", con=conn36)
APM=pd.read_sql_query("Select * from APM Where [Date]>'2019-01-01'", con=conn36)
Mach=pd.read_sql_query("Select Mach, KeyWYP from Waypoints Where [Date]>'2019-
01-01' AND Mach<>'null' and Mach IS NOT NULL and Mach<>'null'", con=conn36)
LevelParam=pd.read_excel("leveldata_2019danberi.xlsx") #leveloutput.py
FP["Datesas"]=pd.to_datetime(FP["Date"], format="%Y-%m-%d")
FP["DatesasFP"]=FP["Datesas"].dt.date
FP["keytoFDM"]=FP["DatesasFP"].astype(str)+"_"+FP["DepartureAirport"]+"_"+FP["Arri
valAirport"]+"_"+FP["Aircraft"]
FP = FP.drop_duplicates(subset=['keytoFDM'],keep='last')
FDM["Datesas"]=pd.to_datetime(FDM["A C TO Datetime"], format="%Y-%m-%d")
FDM["DatesasFDM"]=FDM["Datesas"].dt.date
FDM["aircraft_reg"]=FDM["aircraft_reg"].str.replace("-", "", regex=True)
FDM["keytoFP"]=FDM["DatesasFDM"].astype(str)+"_"+FDM["TO
Airport"]+"_"+FDM["TD Airport"]+"_"+FDM["aircraft_reg"]
FDM = FDM.drop_duplicates(subset=['keytoFP'],keep='last')
MergedDF=pd.merge(left=FDM, right=FP, left_on='keytoFP', right_on='keytoFDM')
APM["Aircraft"]=APM["Aircraft"].str.replace("-", "", regex=True)
APM["dep-dest"]=APM["Dept Dest"].str[0:4]+"_"+APM["Dept Dest"].str[4:8]
APM["FM Dev (%)"]=pd.to_numeric(APM["FM Dev (%)"], downcast="float")
APM["APMkey"]=APM["Date"].astype(str)+"_"+APM["dep-dest"]+"_"+APM["Aircraft"]
APM=APM[["APMkey", "FM Dev (%)"]]
APM=APM.groupby(['APMkey']).mean()
MergedDF=pd.merge(left=MergedDF, right=APM, left_on='keytoFDM',
right_on='APMkey')
MergedDF["FID"]=MergedDF["FID"].astype(int)
LevelParam["FID"]=LevelParam["FID"].astype(int)
MergedDF=pd.merge(left=MergedDF, right=LevelParam, left_on='FID', right_on='FID')
Mach["MachFP"]=pd.to_numeric(Mach["Mach"], errors='coerce', downcast="float")
Mach=Mach.groupby(['KeyWYP']).mean()
MergedDF=pd.merge(left=MergedDF, right=Mach, left_on='KeyFP',
right_on='KeyWYP')
MergedDF=MergedDF.apply(pd.to_numeric, errors='ignore' )
MergedDF["MachFDM"]=MergedDF["Mach - Cruise (mean)"]
```

```

MergedDF["FBcalc0"] = MergedDF["Fuel Burn - B Takeoff"] + MergedDF["Fuel Burn - H Landing"] + MergedDF["Fuel Burn - C Initial Climb"] + MergedDF["Fuel Burn - D Climb"] + MergedDF["Fuel Burn - E Cruise"] + MergedDF["Fuel Burn - F Descent"] + MergedDF["Fuel Burn - G Approach"]
MergedDF["FBcalc1"] = MergedDF["Fuel Burn - Cruise"] + MergedDF["Fuel Burn - Climb"] + MergedDF["Fuel Burn - Descent"]
MergedDF["FuelBurnFDM"] = MergedDF[["FBcalc0", "FBcalc1"]].max(axis=1)
MergedDF["ADcalc0"] = MergedDF["Air Distance - C Initial Climb"] + MergedDF["Air Distance - D Climb"] + MergedDF["Air Distance - E Cruise"] + MergedDF["Air Distance - F Descent"] + MergedDF["Air Distance - G Approach"]
MergedDF["ADcalc1"] = MergedDF["Air Distance - Cruise"] + MergedDF["Air Distance - Climb"] + MergedDF["Air Distance - Descent"]
MergedDF["AirDistanceFDM"] = MergedDF[["ADcalc0", "ADcalc1"]].max(axis=1)
MergedDF["GDcalc0"] = MergedDF["Ground Distance - B Takeoff"] + MergedDF["Ground Distance - H Landing"] + MergedDF["Ground Distance - C Initial Climb"] + MergedDF["Ground Distance - D Climb"] + MergedDF["Ground Distance - E Cruise"] + MergedDF["Ground Distance - F Descent"] + MergedDF["Ground Distance - G Approach"]
MergedDF["GDcalc1"] = MergedDF["Ground Distance - Cruise"] + MergedDF["Ground Distance - Climb"] + MergedDF["Ground Distance - Descent"]
MergedDF["GroundDistanceFDM"] = MergedDF[["GDcalc0", "GDcalc1"]].max(axis=1)
MergedDF["Durcalc0"] = MergedDF["Duration - B Takeoff"] + MergedDF["Duration - H Landing"] + MergedDF["Duration - C Initial Climb"] + MergedDF["Duration - D Climb"] + MergedDF["Duration - E Cruise"] + MergedDF["Duration - F Descent"] + MergedDF["Duration - G Approach"]
MergedDF["Durcalc1"] = MergedDF["Duration - Cruise"] + MergedDF["Duration - Climb"] + MergedDF["Duration - Descent"]
MergedDF["DurationFDM"] = MergedDF[["Durcalc0", "Durcalc1"]].max(axis=1)
MergedDF["TOW_FDM"] = MergedDF["FuelBurnFDM"] + MergedDF["Weight - H Landing"]
MergedDF["FMdevFDM"] = MergedDF["FM Dev (%)"]
MergedDF["Altitude_MaxFDM"] = MergedDF["Altitude - E Cruise (max)"]
MergedDF[["Hour", "Minute", "Sec"]] = MergedDF["TripFuelDuration"].str.split(':', expand=True)
MergedDF["DurationFP"] = MergedDF["Hour"].astype(int)*60 + MergedDF["Minute"].astype(int)
MergedDF["FuelBurnFP"] = MergedDF["TripFuelWeight"]
MergedDF["GroundDistanceFP"] = MergedDF["GroundDistance"]
MergedDF["AirDistanceFP"] = MergedDF["AirDistance"]
MergedDF["TOW_FP"] = MergedDF["TakeoffWeight"]
MergedDF["FMdevFP"] = (MergedDF["CruiseDegradation"] - 1) * (-100)
MergedDF["Altitude_MaxFP"] = MergedDF["FlightLevel"] * 100
MergedDF = MergedDF[MergedDF["aircraft_reg"] != "TCMKS"]
MergedDF = MergedDF[["FID", "aircraft_reg", "DateasFDM", "FuelBurnFDM", "MachFDM", "GroundDistanceFDM", "AirDistanceFDM", "DurationFDM", "TOW_FDM", "FMdevFDM", "ParamA_FDM", "ParamB_FDM", "Altitude_MaxFDM", "FuelBurnFP", "MachFP", "GroundDistanceFP", "AirDistanceFP", "DurationFP", "TOW_FP", "FMdevFP", "ParamA_FP", "ParamB_FP", "Altitude_MaxFP"]]
MergedDF.to_excel("MasterDB2019danberi_nomks_TOLD.xlsx")

```

H. Train-Validate Multiple Models

```
from time import process_time
import time
import torch
import torchvision
from torchvision import transforms, datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl
import os
import scipy.stats as stats
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
t0= process_time()
DF=pd.read_excel("MasterDB2019danberi_nomks_TOLD.xlsx")
DF = DF.sample(frac=1).reset_index(drop=True)
DF=DF[DF["DateasFDM"]<'2020-01-01']
altitudeparameterFP=['ParamA_FP',
altitudeparameterFDM=['ParamA_FDM',
distanceparameterFP=['AirDistanceFP',
distanceparameterFDM=['AirDistanceFDM',
def MAPEloss(output, target):
    loss=torch.mean(torch.abs((output - target))/torch.abs(target))
    return loss
for cntDF in range(len(altitudeparameterFP)):
    for cntDF2 in range(len(distanceparameterFP)):
        DFtrain=DF[["MachFP", 'TOW_FP', altitudeparameterFP[cntDF],
distanceparameterFP[cntDF2], "FMdevFP", "FuelBurnFP"]]
        DFtest=DF[["FID","MachFDM", 'TOW_FDM', altitudeparameterFDM[cntDF],
distanceparameterFDM[cntDF2], "FMdevFDM", "FuelBurnFDM"]]
        device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
        DFtrain=DFtrain.dropna()
        DFtest=DFtest.dropna()
        trainlen=int(len(DFtrain))
        testlen=int(len(DFtest)*.8)
        train00=DFtrain[:trainlen]
        valid00=DFtest[testlen:]
        test00=DFtest[:testlen]

test00.to_excel("Test_Dataset/"+altitudeparameterFDM[cntDF]+"_"+distanceparameterFDM[cntDF2]+"_DF.xlsx")
```

```

    valid00=valid00[["MachFDM", 'TOW_FDM', altitudeparameterFDM[cntDF],
distanceparameterFDM[cntDF2], "FMdevFDM", "FuelBurnFDM"]]
    for cntnorm in range(len(train00.columns)):
        scaler = StandardScaler()
        train00[train00.columns[cntnorm]] =
scaler.fit_transform(train00[train00.columns[cntnorm]].values.reshape(-1,1))
    for cntnorm1 in range(len(valid00.columns)):
        scaler1 = StandardScaler()
        valid00[valid00.columns[cntnorm1]] =
scaler1.fit_transform(valid00[valid00.columns[cntnorm1]].values.reshape(-1,1))
    indepparams=5
    train0 =
torch.tensor(train00.iloc[:,indepparams].values.astype(np.float32),device=device)
    traintarget0 =
torch.tensor(train00.iloc[:,indepparams:].values.astype(np.float32),device=device)
    train = torch.utils.data.TensorDataset(train0, traintarget0)
    valid0 =
torch.tensor(valid00.iloc[:,indepparams].values.astype(np.float32),device=device)
    validtarget0 =
torch.tensor(valid00.iloc[:,indepparams:].values.astype(np.float32),device=device)
    valid = torch.utils.data.TensorDataset(valid0, validtarget0)
    #test0 =
torch.tensor(test00.iloc[:,indepparams].values.astype(np.float32),device=device)
    #testtarget0 =
torch.tensor(test00.iloc[:,indepparams:].values.astype(np.float32),device=device)
    #test = torch.utils.data.TensorDataset(test0, testtarget0)
    trainset=torch.utils.data.DataLoader(train, batch_size=100, shuffle=True)
    validset=torch.utils.data.DataLoader(valid, batch_size=100, shuffle=True)
    #testset=torch.utils.data.DataLoader(test, batch_size=100, shuffle=True)
    staticnodearray=[16,32,64]
    class Net1(nn.Module):
        def __init__(self):
            super().__init__()
            self.fc1=nn.Linear(indepparams, staticnode)
            self.fc2=nn.Linear(staticnode,staticnode)
            self.fc3=nn.Linear(staticnode,staticnode)
            self.fc4=nn.Linear(staticnode,1)
        def forward(self, x):
            x=F.relu(self.fc1(x))
            x=F.relu(self.fc2(x))
            x=F.relu(self.fc3(x))
            x=self.fc4(x)
            return x
    class Net2(nn.Module):
        def __init__(self):
            super().__init__()
            self.fc1=nn.Linear(indepparams, staticnode)
            self.fc2=nn.Linear(staticnode,staticnode)
            self.fc3=nn.Linear(staticnode,1)

```

```

def forward(self, x):
    x=F.relu(self.fc1(x))
    x=F.relu(self.fc2(x))
    x=self.fc3(x)
    return x
lratearray=[0.01,0.001,0.0001]#
epochsarray=[50,100,200]#
for cntnode in range(len(staticnodearray)):
    staticnode=staticnodearray[cntnode]
    for cntlr in range(len(lratearray)):
        lrate=lratearray[cntlr]
        for cntepoch in range(len(epochsarray)):
            epochs=epochsarray[cntepoch]
            for cntmodelnet in range(2):
                if cntmodelnet==0:
                    net=Net1()
                    modellayer="4layers"
                else:
                    net=Net2()
                    modellayer="3layers"
            net=net.to(device)
            optimizer=optim.Adam(net.parameters(), lr=lrate)
            validloss_list = []
            trainloss_list = []
            for cntloss in range(2):
                for epoch in range(epochs):
                    epochloss=[]
                    for ts in trainset:
                        X,y=ts
                        net.zero_grad()
                        output= net(X.view(-1,indepparams))
                        if cntloss==0:
                            loss=MAPEloss(output, y)
                            lossfunc="MAPE"
                        else:
                            loss=nn.MSELoss()
                            loss=loss(output, y)
                            lossfunc="MSE"
                        epochloss.append(loss.item())
                    loss.backward()
                    optimizer.step()
            trainloss_param=sum(epochloss)/len(epochloss)
            for vs in validset:
                X,y=vs
                net.zero_grad()
                output= net(X.view(-1,indepparams))
                if cntloss==0:
                    loss=MAPEloss(output, y)
                    lossfunc="MAPE"

```

```

else:
    loss=nn.MSELoss()
    loss=loss(output, y)
    lossfunc="MSE"
    epochloss.append(loss.item())
    validloss_param=sum(epochloss)/len(epochloss)
    validloss_list.append(validloss_param)
    if validloss_param<=min(validloss_list):
        try:

modelname='SavedModel_180422/'+str(altitudeparameterFP[cntDF])+ "_"+str(distancepar
ameterFP[cntDF2])+ "_"+str(staticnode)+"_"+str(lrate)+"_"+str(epochs)+"_"+modellayer+
"+"_"+lossfunc+'.pth'

        torch.save(net.state_dict(), modelname)
        min_valid_loss=validloss_param

inv_min_valid_loss=scaler1.inverse_transform([[min_valid_loss]])
    min_train_loss=trainloss_param
    inv_min_train_loss=scaler.inverse_transform([[min_train_loss]])
except Exception as E:
    time.sleep(3)
    print(E)
    try:

modelname='SavedModel_180422/'+str(altitudeparameterFP[cntDF])+ "_"+str(distancepar
ameterFP[cntDF2])+ "+"_"+str(staticnode)+"_"+str(lrate)+"_"+str(epochs)+"_"+modellayer
+"_"+lossfunc+'.pth'

        torch.save(net.state_dict(), modelname)
        min_valid_loss=validloss_param

inv_min_valid_loss=scaler1.inverse_transform([[min_valid_loss]])
    min_train_loss=trainloss_param

inv_min_train_loss=scaler.inverse_transform([[min_train_loss]])
    except:
        with open("failed.txt", "a") as fileobj2:
            fileobj2.write(modelname+"\n")
        with open("LOGS_180422.txt", "a") as fileobj:

fileobj.write(str(altitudeparameterFP[cntDF])+","+str(distanceparameterFP[cntDF2])+","+
str(staticnode)+","+str(lrate)+","+str(epochs)+","+modellayer+","+lossfunc+","+

str(min_train_loss)+","+str(min_valid_loss)+","+str(inv_min_train_loss)+","+str(inv_min_
valid_loss)+"\n")
t1 = process_time()
print("elapsed time on", device, t1-t0)

```

I. Testing the Models and Saving the Results

```
import torch
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
import os
import pyodbc
import pandas as pd
from datetime import datetime, timedelta, date
from joblib import dump, load
from torch import jit
from torch import no_grad
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
modelname="ParamB_FP_AirDistanceFP_64_0.001_200_4layers_MAPE"
DF=pd.read_excel("Test_Dataset/ParamB_FDM_AirDistanceFDM_DF.xlsx",index_col=0)
)
DF = DF.sample(frac=1).reset_index(drop=True)
DF=DF.dropna()
print(DF)
for cntnorm in range(1,len(DF.columns)):
    scaler = StandardScaler()
    DF[DF.columns[cntnorm]] =
scaler.fit_transform(DF[DF.columns[cntnorm]].values.reshape(-1,1))
device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
indepparams=5
nodes=64
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1=nn.Linear(indepparams, nodes)
        self.fc2=nn.Linear(nodes,nodes)
        self.fc3=nn.Linear(nodes,nodes)
        self.fc4=nn.Linear(nodes,1)
    def forward(self, x):
        x=F.relu(self.fc1(x))
        x=F.relu(self.fc2(x))
        x=F.relu(self.fc3(x))
        x=self.fc4(x)
        return x
net=Net()
net.load_state_dict(torch.load("SavedModel_180422/"+modelname+".pth",
map_location=device))
```

```

net.eval()
net.to(device)
cnt1=0
cnt2=0
cnt3=0
with open("Model Results/"+modelname+".csv", "a") as fileobj:
    fileobj.write("FID"+"","YHAT"+"","Y"+"\\n")
for cntmodel in range(len(DF)):
    serie = torch.tensor(DF.iloc[cntmodel,1:6].values.astype(np.float32), device=device)
    output=net(serie)
    output=output.cpu().detach().numpy()
    ModelOutput=scaler.inverse_transform(output.reshape(-1,1))
    TargetOutput=scaler.inverse_transform([[DF.iat[cntmodel,6]]])
    FID=int(DF.iat[cntmodel,0])
    with open("Model Results/"+modelname+".csv", "a") as fileobj:

fileobj.write(str(FID)+"", "+str(float(ModelOutput[0]))+", "+str(float(TargetOutput[0]))+"\\n"
)

```

J. Statistical Accuracy Check

```
import pandas as pd
import numpy as np
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
from glob import glob
bestmodels=glob("Model Results/*.csv")
#modelname="ParamB_FP_AirDistanceFP_64_0.001_200_4layers_MSE"
for cntmodel in range(len(bestmodels)):
    DF=pd.read_csv(bestmodels[cntmodel])
    dfnumpy=DF.to_numpy()
    yhat=DF[["YHAT"]].to_numpy()
    y=DF[["Y"]].to_numpy()
    mae = metrics.mean_absolute_error(y, yhat)
    mape = metrics.mean_absolute_percentage_error(y, yhat)
    mre= metrics.mean_relative_error(y, yhat)
    mse = metrics.mean_squared_error(y, yhat)
    rmse = np.sqrt(mse) # or mse**(0.5)
    r2 = metrics.r2_score(y,yhat)
    print("Results of sklearn.metrics:")
    print("MAE:",mae)
    print("MAPE:",mape)
    print("MSE:", mse)
    print("RMSE:", rmse)
    print("R-Squared:", r2)
    x=(yhat-y)/y
    threesigma_boundaries=[x.mean()-3*x.std(),x.mean()+3*x.std()]
    plt.hist(x, bins=150)
    plt.show()
    with open("AccuracyCheckLogbook_2.csv", "a") as fileobj:

fileobj.write(bestmodels[cntmodel]+","+str(mae)+","+str(mape)+","+str(mse)+","+str(rms
e)+","+str(r2)+"\n")
```

K. Training and Testing Best Model Only

```
from time import process_time
import torch
import torchvision
from torchvision import transforms, datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl
import os
import scipy.stats as stats
import sklearn.metrics as metrics
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
t0= process_time()
DF=pd.read_excel("MasterDB2019danberi_nomks_TOLD.xlsx")
DF = DF.sample(frac=1).reset_index(drop=True)
DF=DF[DF["DateasFDM"]<'2020-01-01']
DF=DF.dropna()
DFtrain=DF[["MachFP", 'TOW_FP', 'ParamB_FP', "AirDistanceFP", "FMdevFP",
"FuelBurnFP"]]
DFtest=DF[["MachFDM", 'TOW_FDM', 'ParamB_FDM', "AirDistanceFDM",
"FMdevFDM", "FuelBurnFDM"]]
device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
for cntnorm in range(len(DFtrain.columns)):
    scaler = StandardScaler()
    DFtrain[DFtrain.columns[cntnorm]] =
scaler.fit_transform(DFtrain[DFtrain.columns[cntnorm]].values.reshape(-1,1))
for cntnorm1 in range(len(DFtest.columns)):
    scaler1 = StandardScaler()
    DFtest[DFtest.columns[cntnorm1]] =
scaler1.fit_transform(DFtest[DFtest.columns[cntnorm1]].values.reshape(-1,1))
trainlen=int(len(DFtrain))
testlen=int(len(DFtest)*.8)
train00=DFtrain[:trainlen]
valid00=DFtest[testlen:]
test00=DFtest[:testlen]
test00 = test00.sample(frac=1).reset_index(drop=True)
indepparams=5
train0 = torch.tensor(train00.iloc[:,indepparams].values.astype(np.float32),device=device)
traintarget0 =
torch.tensor(train00.iloc[:,indepparams:].values.astype(np.float32),device=device)
```

```

train = torch.utils.data.TensorDataset(train0, traintarget0)
valid0 =
torch.tensor(valid00.iloc[:,indepparams].values.astype(np.float32),device=device)
validtarget0 =
torch.tensor(valid00.iloc[:,indepparams:].values.astype(np.float32),device=device)
valid = torch.utils.data.TensorDataset(valid0, validtarget0)
trainset=torch.utils.data.DataLoader(train, batch_size=100, shuffle=True)
validset=torch.utils.data.DataLoader(train, batch_size=100, shuffle=True)
staticnode=32
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1=nn.Linear(indepparams, staticnode)
        self.fc2=nn.Linear(staticnode,staticnode)
        self.fc3=nn.Linear(staticnode,staticnode)
        self.fc4=nn.Linear(staticnode,1)
    def forward(self, x):
        x=F.relu(self.fc1(x))
        x=F.relu(self.fc2(x))
        x=F.relu(self.fc3(x))
        x=self.fc4(x)
        return x
net=Net()
net=net.to(device)
optimizer=optim.Adam(net.parameters(), lr=0.001)
epochs=200
validloss_list = []
trainloss_list = []
for epoch in range(epochs):
    epochloss=[]
    for ts in trainset:
        X,y=ts
        net.zero_grad()
        output= net(X.view(-1,indepparams))
        loss=nn.MSELoss()
        loss=loss(output, y)
        epochloss.append(loss.item())
        loss.backward()
        optimizer.step()
    trainloss_param=sum(epochloss)/len(epochloss)
    trainloss_list.append(trainloss_param)
    for vs in validset:
        X,y=vs
        net.zero_grad()
        output= net(X.view(-1,indepparams))
        loss=nn.MSELoss()
        loss=loss(output, y)
        epochloss.append(loss.item())
    validloss_param=sum(epochloss)/len(epochloss)

```

```

validloss_list.append(validloss_param)
if validloss_param<=min(validloss_list):
    torch.save(net.state_dict(), 'SavedModel_250422/best_model_validloss.pth')
    print( "Valid Loss:", validloss_list[epoch])
plt.plot(np.linspace(1, epochs, epochs).astype(int), validloss_list, "r", label="validation
loss")
plt.plot(np.linspace(1, epochs, epochs).astype(int), trainloss_list, "g", label="training loss")
plt.ylim(ymax = 0.05, ymin = 0)
plt.legend()
plt.show()
t1 = process_time()
print("elapsed time on training", device, t1-t0)
net.load_state_dict(torch.load('SavedModel_250422/best_model_validloss.pth'))
net.eval()
error_list=[]
DFtest_check=pd.DataFrame([], columns=["yhat","y"])
for cnttest in range(len(test00)):
    serie = torch.tensor(test00.iloc[cnttest,0:5].values.astype(np.float32), device=device)
    output=net(serie)
    output=output.cpu().detach().numpy()
    ModelOutput=scaler1.inverse_transform(output.reshape(-1,1))
    TargetOutput=scaler1.inverse_transform([[test00.iat[cnttest,5]]])
    DFdummy=pd.DataFrame(np.append(ModelOutput[0],TargetOutput[0])).transpose()
    DFdummy.columns=["yhat", "y"]
    DFtest_check=pd.concat([DFtest_check, DFdummy])
yhat=DFtest_check[["yhat"]].to_numpy()
y=DFtest_check[["y"]].to_numpy()
mae = metrics.mean_absolute_error(y, yhat)
mape = metrics.mean_absolute_percentage_error(y, yhat)
mse = metrics.mean_squared_error(y, yhat)
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(y,yhat)
print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MAPE:",mape)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
t2 = process_time()
print("elapsed time on test", device, t2-t1)

```