Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# DEGREE FINAL PROJECT

**TITLE: Prediction of 4D drone trajectories from demonstration data**

**DEGREE: Bachelor's degree in Aerospace Systems Engineering**

**AUTHOR: Laura Vazquez Husillos**

**DIRECTOR: Cristina Barrado Muxí**

**DATE: September 5th, 2022**

**Title:** Prediction of 4D drone trajectories from demonstration data

**Author:** Laura Vazquez Husillos

**Director:** Cristina Barrado Muxí

**Date:** September 5th, 2022

## Abstract

Drones are nowadays an air vehicle with many possible applications and with a huge field of development and research. This work deals, more specifically, with the delivery drones used for the distribution of packages to citizens. That is, they could be considered as a new way of distributing packages to the city. Today, this is seen as a hypothetical, because in order to achieve this, many points must be studied and new tools for their control and security must be developed.

This work studies the field of predicting the trajectory of drones through Machine Learning and thus, providing a new point of view in the research and development of this new branch of aviation.

The aim of this work is to predict, through Machine Learning, the trajectory of a drone in 4D based on the data collected in the test flights made by the different air operators participating in the CORUS-XUAM. The objective is to achieve results with sufficient precision to be able to project them valid in terms of air safety.

To carry out this work, it has been necessary the data previously collected from the VLDs operated and the processing of them by means of the code made in Google Coolab. For this reason, different libraries have been needed, such as "Pandas", "Matplotlib", "Numpy" or "Geopandas", among others. Finally, a Machine Learning has been used to carry out the predictions. To do this, it has been necessary to decide which will be the best method used, in our case it has been seen that the most appropriate is the Regression model.

Regarding the results, it has been confirmed that the chosen method has been correct, since the margins of error are quite low and acceptable and the accuracy of it could be considered good with a margin error of maximum 3 minutes.

**Títol:** Predicció de la trajectòria d'un dron en 4D a partir de dades de vol

**Autor:** Laura Vazquez Husillos

**Director:** Cristina Barrado Muxí

**Data:** 5 de setembre del 2022

**Resum**

Els drons són avui en dia un vehicle aeri amb moltes possibles aplicacions i amb un camp enorme de desenvolupament i investigació. En aquest treball es tracta més específicament dels drons utilitzats per la distribució de paquets als ciutadans. És a dir, se'ls podria considerar com una nova forma de repartir paquets a la ciutat. Avui en dia això es veu com un ideal, ja que per poder aconseguir-ho s'han d'estudiar molts punts i desenvolupar noves eines pel seu control i seguretat.

En aquest treball s'estudia l'àmbit de la predicció de la trajectòria dels drons mitjançant una Machine Learning i d'aquesta manera, aportar un punt de vista nou en la investigació i desenvolupament d'aquesta nova branca de l'aviació.

L'objectiu d'aquest treball és predir mitjançant una Machine Learning, la trajectòria d'un dron en 4 dimensions a partir de les dades recollides en els vols de prova fets pels diferents operadors aeris que participaven en el CORUS-XUAM. Es vol assolir uns resultats amb una precisió suficient per poder considerar-ho vàlid en termes de seguretat aèria.

Per la realització d'aquest treball, s'ha necessitat les dades prèviament recollides als VLDs operats i el processament d'elles mitjançant el codi realitzat a Google Coolab. Per això, s'han necessitat diferents llibreries com són la llibreria "Pandas", "Matplotlib", "Numpy" o "Geopandas", entre d'altres. Finalment, s'ha utilitzat una Machine Learning per dur a terme les prediccions. Per fer-ho, ha fet falta decidir quin serà el mètode emprat, en el nostre cas s'ha pogut veure que el més adequat és el mètode de Regressió.

Pel que fa als resultats, s'ha pogut confirmar que el mètode elegit ha estat encertat, ja que, els marges d'error són bastant baixos i acceptables i la precisió del mateix es podria considerar bona amb un marge màxim de 3 minuts.

# INDEX

# LIST OF FIGURES

# LIST OF TABLES

# ACRONYMS

| Acronym | Meaning |
|---|---|
| AI | Artificial Intelligence |
| ATC | Air Traffic Controller |
| CONOPs | Concept of Operations |
| CSV | Comma Separated Values |
| EASA | European Union Aviation Safety Agency |
| eVTOL | Electric Vertical Take-Off and Landing Vehicle |
| GCS | Ground Control Station |
| JSON | Java Script Object Notation |
| KML | Keyhole Markup Language |
| LASSO | Least Absolute Shrinkage and Selection Operator |
| LiDAR | Light Detection and Ranging |
| MAE | Mean Absolute Error |
| RMSE | Root Mean Square Error |
| RPAS | Remotely Piloted Air System |
| SESAR JU | SESAR Joint Undertaking |
| UAM | Urban Air Mobility |
| UAS | Unmanned Aircraft System |
| VLD | Very Large-Scale Demonstration |

# INTRODUCTION

Among all the functions that a drone can perform stand out as most useful, searching survivors after a catastrophe or helping with scientific search. This type of vehicle has been revolutionary and very useful because of their level of safety and efficiency. Nowadays the biggest challenge for drones is to be integrated in the airspace with all the other aircrafts flying in the airspace today.

To provide this integration, it is needed to implement the Urban Air Mobility (UAM). UAM is an ecosystem where different aerial vehicles coexist with the aim of transporting passengers or cargo. Above all, it wants to be implemented in urban and suburban environments.

It is believed that in about 5 or 10 years this system could be implemented and it could have a great growth, so there could be about 23.000 UAM generating about €60 billion.

In order to create this system, it is needed to test it in real life to study and improve it. That is why the CORUS-XUAM project was created, it has the objective of investigating how all air vehicles can coexist in the same airspace, maintaining safety. It is based on the execution of several very large-scale demonstrations where it is intended to test the passenger transport execution, the package delivery and also, the emergency response and surveillance.

In this work, we focus on the test that took place in Spain, more specifically on Castelldefels' beach. With this activity it was intended to emulate operations for transport people or packages with different drones' operators flying at once. From these flights, the necessary data was extracted to carry out this work.

More specifically, this work aims to confirm whether it is possible to predict the trajectory of a drone in 4D from machine learning, with a range of certainty sufficient for the flight to be safe.

# 1 CHAPTER 1. DRONES

## 1.1 What is a drone?

It is said that the word drone comes from the Middle English word *"Drone"* that means male bee. This is because of the sound that this animal makes, which resembles the sound made by a drone. Nowadays we define a drone as a Remotely Piloted Air System (RPAS).

Drones have some restrictions when someone wants to fly them. Not everyone can fly a drone, as it must maintain the safety range. Drones can even kill people or destroy properties.
Every country has different regulations for drones. EASA (European Union Aviation Safety Agency) is the body of the general administration that watches over a safety activity from civil aeronautics and establishes next minimum requirements to allow it to fly a drone [1].

- The operator has to be registered as such
- Train as a pilot
- Have the liability insurance
- Consult the general rules of operation of drones depending on the drone that is going to be flying
- Be aware of the air regulations of the place where the flight is going to be made.

## 1.2 How drones fly

Their movement is based on the sensors and LiDAR detectors that drones have. Drones use a system that calculates their movement taking into account the flight plan and the obstacles. In order to understand how drones work, it is important to be familiar with the components that took sides. The system that allows the drone to fly is the Unmanned Aerial System (UAS) which is controlled by the Ground Control Station (GCS). The GCS can be controlled by a user or via satellites and is responsible for controlling the packages sensors, the flight and some more. Data Links allows the drone to communicate with the ground controller, sending through them important information such as airspeed altitude, flight time, distance from the target and others.

In order to know how drones fly, we have to understand how drones rise. Drones have rotors that push air down and that make the air push the rotor up. So, the faster the rotors roll, the greater the lift is. Once the drone is in the air, it can hover when the pushing force is counteracted by the gravitational force. Drone can go up by increasing the thrust until it will be higher than the gravitational force. Finally, if we want to make a descent, it is needed to decrease the rotor thrust until it is smaller than the gravitational force. Figure 1.1 shows how forces are distributed in the drone.

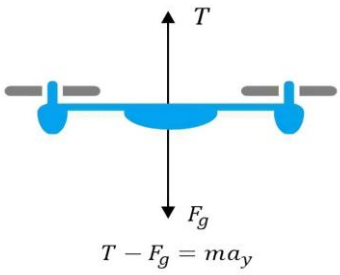


**Figure 1.1** Drone forces [2]

The turning can be explained by the figure 1.2. As we can see red rotors are rotating counter clockwise and green rotators clockwise, that means that the angular momentum is null. If we want to turn the drone to one direction, we have to decrease the angular velocity of the rotor that rotates in the same direction we want to turn the drone. That means, if we want to turn the drone to the left, we will have to decrease the angular velocity of the rotor 2. We have to take into account that decreasing the angular velocity of the rotor 2 also did that the thrust of this rotor decreases and then the drone also goes down. In order to compensate for this, we have to decrease the spin of the same rotors, in this case, the 2 and 4, and increase the spin of the opposite rotors, in this case, the 1 and 3.

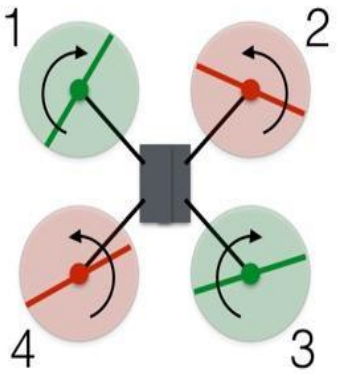


**Figure 1.2** Drone rotors [3]

Finally, we must explain how the drone goes forward, backward or sidewise. We can explain this together because it follows the same principle. In order to make this, we need something that pushes the drone to the wanted direction. In the figure 1.3 we can see how the drone must be placed in order to move the drone forwards. To obtain this, we have 2 options. Decrease the push of the

rotors 1 and 2 or increase the push of the 3 and 4 rotors. The thrust force will compensate with the weight and then, the drone will not ascend or descent [3].

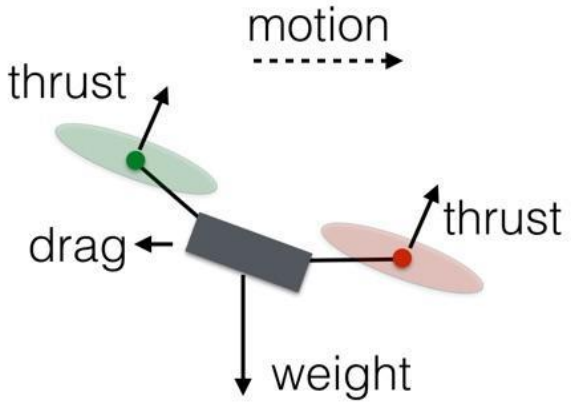


**Figure 1.3** Drone movement forces [3]

## 1.3 Delivery drones

One of the most required functionalities for a drone is to transport packages or passengers. Drones can revolutionise and improve the world with this functionality, because they could get to places where we couldn't think to get there before and raise supplies to isolated villages or with very difficult access. Another function of the delivery drones is to distribute delivery packages and thereby reduce pollution, improve delivery time, or even reduce traffic in cities.

When the VLDs flights were made, different types of delivery drones were used. OMAHA operators used the drones DJI M300 and DIJ M2AE, UTAH used also the DJI M300 and the DJI M600 Pro. SWORD used the S900 drone and JUNO used the DJI M300 RTK.

## 1.4 Urban Air Mobility

The Urban Air Mobility (UAM) is a concept that has the purpose of interconnecting people and goods of a city. This goal takes into account different factors such as technology, politics, infrastructure culture, etc. Through the UAM it is wanted to achieve a city with less congestion and more sustainable. The challenge of urban air mobility is to maintain safety on flights and drones while they increase in quantity, in addition to reducing flight time, reducing CO2 in cities and creating more sustainable and intelligent mobility [4].

The UAM is also a challenge for the U-Space in order to integrate drones in low level airspace. The UAM is intended to be an ecosystem placed in urban and

suburban environments where the automated aerial passenger and cargo services will be found [5].

It is expected to have UAM solutions with a good level of safety, efficiency and sustainability by 2025-2030 [6].

## 1.5 U-Space

U-Space is a set of services that allows drones to fly safely. It is useful in order to manage air traffic from the processing of applications and authorizations for drones' flights. Its goal is to completely integrate drones in the civil airspace. It is mandatory to incorporate drones in the air space to be able to achieve the delivery processes with drones, among other functions that are intended to be done with drones. According to [7] the main objective of the U-space is:

> *Providing those services that are fundamental to the safe navigation of drones, as well as an interface that integrates manned aircraft, suppliers and relevant authorities*

In order to organise the U-Space in a proper way CONOPs (Concept of Operations) is designed by CORUS [7]. This is a protocol that manages the operation condition of a system. CONOPs define three different operational volumes X, Y and Z.
The X volume is the first volume of the U-Space, and it is where the drone operator has to maintain a safe distance visually. The Y volume is the volume where the operator needs an approved plan to fly, because this space needs certain technical requirements. The last volume, the Z volume is where all the tactical services that need to be authorised are located. The VLDs made for this work would be found in the volumes Y and Z.

### 1.5.1 Implementation phases of the U-Space

The integration of the U-Space will be gradual and in order to make that, 4 phases have been described. The first phase is called **U1** and it is based on the technological identification of pilots and aeroplanes. In order to make it, geofencing must be provided to the operator, that means, the zones where the drone cannot cross. The **U2** phase is based on the integration of a security service in a drone's operation by using geofencing, digital management of flight plans, meteorological information, among others. The **U3** phase is based on the development of dynamic geofencing, that means, a constant communication with the ATC. The last phase, **U4** has the aim of offering an active communication system between all aircrafts [8].
The figure 1.4 shows the phases names and the scheduled years for the implementation.

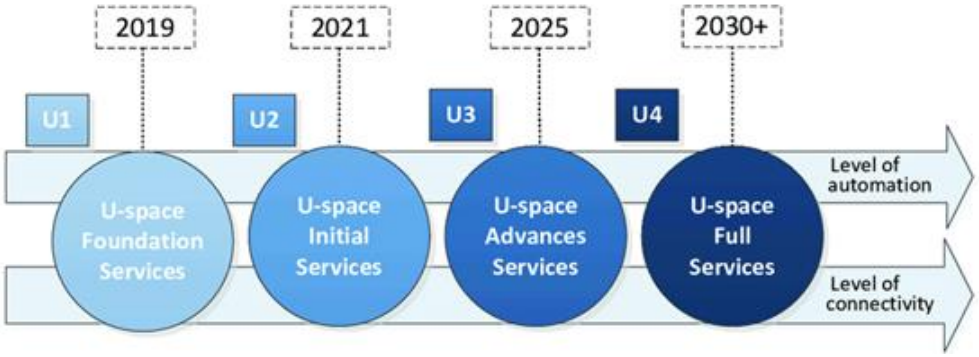


**Figure 1.4** U-Space roadmap [8]

# 2 CHAPTER 2. MACHINE LEARNING

## 2.1. What is a Machine Learning

Machine Learning is a computer system that can learn through algorithms and models from experience, without explicit instructions or human interference. In other words, a Machine Learning is an Artificial Intelligence field that maintains the algorithms updated. Figure 2.1 shows the process of Machine Learning.

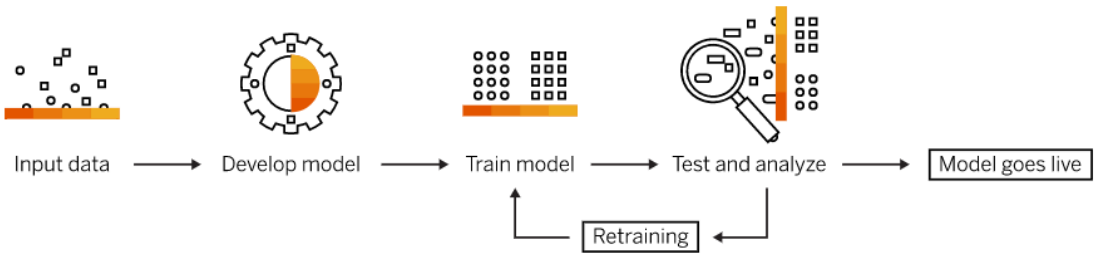


**Figure 2.1** Machine learning process [9]

Machine Learning can be configured with different learning systems. The learning system is supervised when the training data you feed to the algorithm includes the desired solutions. It is understood as an unsupervised system if the training data is unlabelled. Reinforcement learning is defined as a model that learns from its actions [10].

The process of manipulating the data for the prediction and the use of the machine learning for our project is explained in detail in CHAPTER 5. METHODOLOGY. In general, the common steps that have to be followed when using Machine Learning are as follows.
First, we have to manipulate the data, in order to create two different tables. The X table where is placed all the necessary data for the prediction, and the Y table where is placed the data we want to predict. Then, we have to split the data in two parts, the training data that commonly is the 80% of the total data and the test data that is the 20%. Training data is the data that the machine will use for discovering and learning patterns of the machine learning algorithm and then, predict our desired result. On the other hand, the test data is used to evaluate the progress and then make an adjustment or optimise the algorithm with the aim of achieving better results. Finally, we have to train the data, then fit the model we want to use for our prediction and at last, make the prediction.

Machine learning is nowadays used in the real world for speech recognition, chatbots, photo tagging on social media, self-driving cars, discovering data trends or even spotting suspicious transactions, among others.

Artificial Intelligence (AI from now on) is a branch of computer science in which machines operate like human intelligence. Its objective is to make decisions and learn from them based on the information they collect. Through Machine

Learning, AI processes the data and learns about it, getting smarter on its own [13].

## 2.2. Machine Learning Algorithms

When all the data is well structured in the prediction tables, it is useful to use a machine learning algorithm to train the models. Figure 2.2 shows all the possible algorithms, classified by their learning system and their model type. There are three types of learning systems which are explained in the previous section and four types of models that are described next.

- **Clustering** that is based on categorising data into clusters.
- **Dimensionality reduction** based on the reduction of the number of inputs.
- **Classification** based on specifying the conditional probability distributions of the output variables given the input data.
- **Regression** is based on the relation between one dependent variable which is the one that can be controlled and one or more independent variables which shows the changes. The objective of this model is to estimate a variable from the previous ones.

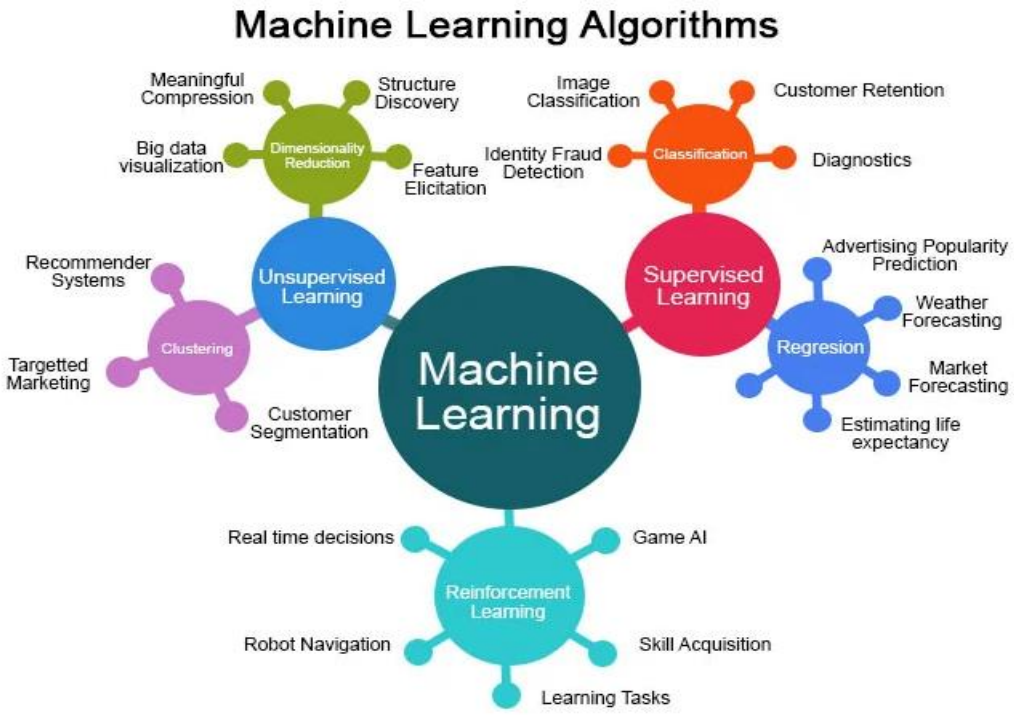


**Figure 2.2** Machine learning algorithms [11]

In this work, we will only explain the algorithms that we used for the prediction.

- **Linear Regression Model** works with the linear relation between the input data. It is very useful to predict numerical values [12].
- **Logistic Regression Model** which estimates the data using two possible scenarios, 0 or 1.
- **Ridge Regression Model** is like linear regression but with correlated data.
- **Bayesian Linear Regression** is a variation of the linear regression but with inaccurate data
- **Elastic Net Regression Model** which is the combination of the two variants of the linear regression, Ridge (explained before) and LASSO where the data is shrunk to a central point.

# 3 CHAPTER 3. VERY LARGE-SCALE DEMONSTRATION

## 3.1 CORUS-XUAM project

The CORUS-XUAM project is a two year very large-scale demonstration (VLD from here on) with the aim of studying how the services and solutions of U-Space could help to incorporate Urban Air Mobility flight operations. Their objective is to obtain a safe, sustainable and efficient air space, where the electric vertical take-off and landing vehicles (eVTOL), unmanned aircraft systems (UAS) and other airspace users can coexist. The interaction between all these types of flight is intended to be made from digital data exchange provided by the U-Space services [5].

It will be necessary for coordination between ATC, U-Space and drone pilots. In this project, VLDs will work on passenger transport, package delivery, emergency response and surveillance.

The CORUS-XUAM project was created by a project consortium coordinated by EUROCONTROL and carried out in SESAR Joint Undertaking (SESAR JU) and Europe's Horizon 2020 framework [14].
The first step in this project is to make an update of the U-Space CONOPs, integrating the UAM in the airspace and creating the new services U3 and U4 of the U-Space phases.

CORUS-XUAM has the aim of carrying out six different VLDs in six different countries. Spain is one of these countries and this work is based on the VLD performed in Spain.

The CORUS-XUAM project in Spain focused on the relation between UAM operators and the urban restrictions. It is intended to create a test of door-to-door package delivery in Castelldefels, a city that is located within the CTR of Barcelona's airport. The activity was carried out for 3 consecutive days and that we will name in all this work as days A, B and C.

## 3.3 Operators

In the demonstration, there were four different operators. The first operator is UTAH, this operator belongs to UAB university, and it was placed on UTAH vertiport which is located at the right of the flying area. The second operator is OMAHA which belongs to UPC university, and it was placed on OMAHA vertiport located to the right of UTAH. JUNO was the third operator and belongs to MARS, a security, drone service and cybersecurity consultancy. JUNO is placed on JUNO vertiport which is located next to OMAHA. Finally, SWORD is the operator that belongs to HEMAV, a technological and artificial intelligence company. SWORD is placed on SWORD vertiport which is located at the left of

the flying area. In front of all these vertiports, along the shore, there were all the delivery ports where the drones will do the delivery procedure. We can see in figure 3.1 a Google Earth scheme of the demonstration area layout.


**Figure 3.1** Demonstration area layout

## 3.4 Delivery ports

In all the demonstration area where the flights were performed, there were 13 delivery ports numerically ordered from right to left. Every delivery port has its own coordinates that will help to correctly deliver the packages by the drone.

## 3.5 Airspace Organization

The demonstration took place in a reserved space in Castelldefels Beach. In this area the drones made their deliveries. Prior to the demonstration, it was established a maximum of two drones flying at the same time and it was mandatory to maintain a safety distance between them of, at least, 5 metres throughout the flight.

In the demonstration, the pilot had to perform the take-off manually, and therefore, the data collected from the take-off did not follow the flight plan. In the same way it happened for the landing, so the landing and take-off data collected from the demonstration will not be useful for our prediction.

Once the drones reached the airway, they followed the flight plan previously created and approved. The airways that drones follow are delimited vertically and horizontally. Drones had to reach the airways after the take-off at 30 metres, then follow the airway and reach the delivery port point at 20 metres. At the delivery port drones did not leave any package or person, they only

simulated it by staying 10 seconds above the delivery port about 20 metres high. After that, the drone returns to the airway at 25 metres, raises its height to the height of the airway that will follow and follows it until it arrives at the landing point, and in it descends to the 25 metres. Finally, the landing process was manually carried out by the pilot.

Focusing on the airways used by these drones, 4 airways were defined, 2 of them used for going from Est to West and the other 2 used for going from West to Est. These different airways have different heights at 30, 40, 70 and 80 metres. Each of them was chosen according to the flight plan.

## 3.6 VLD process

Prior to performing VLD it is necessary to go through a few previous steps.
The first step in order to design a VLD is to prepare a flight plan by the operator or operators who are going to do the VLD. The flight plan is a file where all the flight is defined. Normally, the flight plan is made in a KML language (explained in more detail in the section 4.2 File languages) and it contains all the flight information about each waypoint. It is useful to use the KML language in order to afterwards visualise the flight plan in Google Earth. Once the operator has done the flight plan, sends it to the U-Space who is going to review it thoroughly. The U-Space will transform the flight plan in a 4D trajectory to check if the chosen airways are not occupied. Finally, the U-Space will approve the flight plan if there is no conflict between the operators. Therefore, the operator will only fly its demonstration if it has been approved by U-Space.

# 4 CHAPTER 4. Tools

## 4.1 Google Coolab

At this point of the document, it’s important to talk about Google Collaboratory. Google Collaboratory (Colab from here on out) is the Drive tool used for this project. This tool allows you to program and run code in Python language with the advantages that it is free, online, doesn't need pre-configuration and multiple users can use it at the same time. Colab is also useful because as it is a Jupyter notebook, you can run each cell separately. Moreover, the documents can be previously uploaded in Drive and then can be used in the code. [15] Colab has a special tool that differentiates it from the others programming software, the possibility of adding a text cell where you can append some information and even images if it's desired.

Python is an object-oriented open-source language, and it is not needed to compile the source code to run it. Therefore, it could also be defined as an interpreted language. This type of language is useful because it is easier than other languages and has a large number of libraries.

There are some additional tools used later in the project as the following.

- Pandas is a library of Python that is useful to manipulate data tables. It is very useful for designing a Machine Learning project.
- Geopandas is an open library in Python which permits work with vector data.
- Basemap is a tool that creates maps.
- Pyplot.plot, which draws a graph of the data you pass to it. It should be noted that Pyplot is a function which belongs to matplotlib library, and it is necessary to import them to use it.

DataFrame is the most used tool in this project and is a two-dimensional data structure which allows to make operations on rows and columns, like delating, adding and some more. It is useful to process the data because it is flexible, the axes are labelled and has lots of different operations that can be done in the columns and rows [16].

## 4.2 File languages

In this project, we use three different types of files that contain the data created and obtained in the VLDs. The files including the scheduled flights data are written in JSON language, the files that contain the flight plan data are written in KML language and the files that contain the data obtained from the VLDs flights use the CSV format.

We can define JSON language as a lightweight data-interchange format that is simple to read and write for humans and uncomplicated to parse and generate

for machines [17]. It's a completely independent language but uses familiar conventions for the programmers. If we talk about the JSON structures, we can differentiate a selection of names and a group of arrays that contain their values. In a specific way, JSON language uses universal data structures, and it is for that reason that JSON text can be converted easily into JavaScript text.

KML language is defined as an *"XML-based markup language designed to annotate and overlay visualisations on various two-dimensional, Web-based online maps or three-dimensional Earth browsers"* [18].

CSV format is very common, and is used for Excel. In this format the data is divided by commas, by this way, the data can be placed in a table.

# 5 CHAPTER 5. METHODOLOGY

## 5.1 Obtain the data

The first step can be seen in figure 5.1. In our case we have three different type of data, and each one is collected in a different way. The first data collected is the flight plan data, as we explained before, this data is written in KML language. The flight plan is created by the operator and sent to the U-Space to approve it.
The second data used in this project is the scheduled flight data or trajectory data. This data is written in JSON language and is created by the U-Space from the Flight Plan.
The last data of the project is the VLDs data, that means the data obtained when the flights are performed and is collected in CSV format.


**Figure 5.1** Methodology process part 0

## 5.2 Data cleaning

When it comes to predicting any information from an AI, the most important procedure is the processing and analysis of the data. It is crucial to clean the data to later be able to make the prediction itself, with the data displaced in the correct way. See figure 5.2.


**Figure 5.2** Methodology process part I

The following step in this procedure is to do the data cleaning, that means eliminate the incorrect measurements or irrelevant information for the prediction process, combine the data or change the data format. In our case, I had three different types of files.

The first file that I adapted was the flight plan file that had the information about the Flight Plan in KML language. Each flight plan file had recorded data from every single waypoint of the trajectory. This type of file is useful for drawing the trajectory in Google Earth and seeing its waypoints. We can see an example of this file in figure 5.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
	<Document xmlns="">
		<name>Waypoint1</name>
		<open>1</open>
		<ExtendedData xmlns:mis="www.dji.com">
			<mis:type>Waypoint</mis:type>
			<mis:stationType>0</mis:stationType>
		</ExtendedData>
		<Style id="waylineGreenPoly">
			<LineStyle>
				<color>FF0AEE8B</color>
				<width>6</width>
			</LineStyle>
		</Style>
		<Style id="waypointStyle">
			<IconStyle>
				<Icon>
					<href>https://cdnen.dji-flighthub.com/static/app/images/point.png</href>
				</Icon>
			</IconStyle>
		</Style>
		<Folder>
			<name>Waypoints</name>
			<description>Waypoints in the Mission.</description>
			<Placemark>
				<name>Waypoint1</name>
				<visibility>1</visibility>
				<description>Waypoint</description>
				<styleUrl>#waypointStyle</styleUrl>
				<ExtendedData xmlns:mis="www.dji.com">
					<mis:useWaylineAltitude>false</mis:useWaylineAltitude>
					<mis:heading>0</mis:heading>
					<mis:turnMode>Auto</mis:turnMode>
					<mis:gimbalPitch>0.0</mis:gimbalPitch>
					<mis:useWaylineSpeed>false</mis:useWaylineSpeed>
					<mis:speed>9</mis:speed>
					<mis:useWaylineHeadingMode>true</mis:useWaylineHeadingMode>
					<mis:useWaylinePointType>true</mis:useWaylinePointType>
					<mis:pointType>CurvaturePassed</mis:pointType>
					<mis:cornerRadius>75.0</mis:cornerRadius>
				</ExtendedData>
```

**Figure 5.3** Flight plan file in KML language

Then, I adapted the scheduled flight file, that was the one that had the information about the flight's trajectory written in JSON language.
This type of file had, for each flight, their callsign, its take off time, the trajectory coordinates, among others. This data will be very useful to predict the drone position. We can see an example of this file in figure 5.4.

```
[
  {
    "callsign": "B_JUNO_1631",
    "request": "2022-03-09T14:17:00.000",
    "geometry":
    [
      {
        "type": "Trajectory",
        "coordinates":
        [
          [
            1.97890135,
            41.26524176,
            30,
            0
          ],
          [
            1.9789006061137058,
            41.26406284600235,
            40,
            44
          ],
          [
            1.9789002088966998,
            41.263433320966165,
            40,
            58
          ],
          [
            1.9807892744478184,
            41.26343263203988,
            40,
            74
          ],
          [
            1.9807896898758919,
            41.26406215707302,
            40,
            88
          ],
```


**Figure 5.4** Scheduled flight file in JSON language

Finally, the last file that I had to adapt was the real data. That means the data obtained from the VLDs. There was one file per each flight, and each one included information about the real flight. That means, its coordinates, altitude, speed and distance for every 100 milliseconds. Figure 5.5 shows a file written in CSV format.

| time(millisecc | datetime(utc) | latitude | longitude | height_above | height_above | ground_eleva | altitude_abov | height_sonar( | speed(mph) | distance(feet) | mileage(feet) | satellites | gpslevel | voltage(v) | max_altitude( |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 09/03/2022 12:09 | 41,2651124 | 1,9967029 | 0 | Available with | Available with | 4,55457167 | 0 | 0,00510163 | 0 | 0 | 17 | 0 | 0 | 4,55457167 |
| 200 | 09/03/2022 12:09 | 41,2651124 | 1,9967029 | 0 | | | 4,55457167 | 0 | 0,01037117 | 0,00071458 | 0,00071458 | 17 | 0 | 0 | 4,55457167 |
| 300 | 09/03/2022 12:09 | 41,2651124 | 1,9967029 | 0 | | | 4,55457167 | 0 | 0,01726202 | 0,00080345 | 0,00093473 | 17 | 0 | 0 | 4,55457167 |
| 400 | 09/03/2022 12:09 | 41,2651124 | 1,99670291 | 0 | | | 4,55457167 | 0 | 0,02203404 | 0,00229442 | 0,00291452 | 17 | 0 | 0 | 4,55457167 |
| 500 | 09/03/2022 12:09 | 41,2651124 | 1,99670291 | 0 | | | 4,55457167 | 0 | 0,02807181 | 0,00238871 | 0,00333793 | 17 | 0 | 0 | 4,55457167 |
| 600 | 09/03/2022 12:09 | 41,2651124 | 1,99670291 | 0 | | | 4,55457167 | 0 | 0,02234399 | 0,00366342 | 0,00461476 | 17 | 0 | 0 | 4,55457167 |
| 700 | 09/03/2022 12:09 | 41,2651124 | 1,99670291 | 0 | | | 4,55457167 | 0 | 0,02580825 | 0,00459656 | 0,00569806 | 17 | 0 | 0 | 4,55457167 |
| 800 | 09/03/2022 12:09 | 41,2651124 | 1,99670296 | 0 | | | 4,55457167 | 0 | 0,02781411 | 0,0177941 | 0,01918402 | 17 | 0 | 0 | 4,55457167 |
| 900 | 09/03/2022 12:09 | 41,2651124 | 1,99670299 | 0 | | | 4,55457167 | 0 | 0,03255866 | 0,026442 | 0,02815824 | 17 | 0 | 0 | 4,55457167 |
| 1000 | 09/03/2022 12:09 | 41,2651124 | 1,99670298 | 0 | | | 4,55457167 | 0 | 0,03341608 | 0,02301168 | 0,0317458 | 17 | 0 | 47,334 | 4,55457167 |
| 1100 | 09/03/2022 12:09 | 41,2651124 | 1,99670297 | 0 | | | 4,55457167 | 0 | 0,0557224 | 0,02069508 | 0,03439665 | 17 | 0 | 47,334 | 4,55457167 |
| 1200 | 09/03/2022 12:09 | 41,2651124 | 1,99670296 | 0 | | | 4,55457167 | 0 | 0,04054185 | 0,01679849 | 0,03840322 | 17 | 0 | 47,334 | 4,55457167 |
| 1300 | 09/03/2022 12:09 | 41,2651124 | 1,99670295 | 0 | | | 4,55457167 | 0 | 0,04563939 | 0,01499068 | 0,0403061 | 17 | 0 | 47,334 | 4,55457167 |
| 1400 | 09/03/2022 12:09 | 41,2651124 | 1,996703 | 0 | | | 4,55457167 | 0 | 0,04477817 | 0,02813022 | 0,05364731 | 17 | 0 | 47,334 | 4,55457167 |
| 1500 | 09/03/2022 12:09 | 41,2651124 | 1,99670303 | 0 | | | 4,55457167 | 0 | 0,06086915 | 0,03551247 | 0,06120571 | 17 | 0 | 47,334 | 4,55457167 |
| 1600 | 09/03/2022 12:09 | 41,2651124 | 1,996703 | 0 | | | 4,55457167 | 0 | 0,08963687 | 0,0281795 | 0,06894029 | 17 | 0 | 47,334 | 4,55457167 |
| 1700 | 09/03/2022 12:09 | 41,2651124 | 1,99670298 | 0 | | | 4,55457167 | 0 | 0,05233321 | 0,02443616 | 0,0735187 | 17 | 0 | 47,334 | 4,55457167 |
| 1800 | 09/03/2022 12:09 | 41,2651124 | 1,99670295 | 0 | | | 4,55457167 | 0 | 0,08127116 | 0,01687795 | 0,08168406 | 17 | 0 | 47,334 | 4,55457167 |
| 1900 | 09/03/2022 12:09 | 41,2651124 | 1,99670294 | 0 | | | 4,55457167 | 0 | 0,04465044 | 0,01533104 | 0,0848189 | 17 | 0 | 47,334 | 4,55457167 |

**Figure 5.5** Real data file in CSV format

## 5.3 Data processing

Structuring the data is the third step that we have to develop. In our case, we structure the data in tables because it was easier for us.


**Figure 5.6** Methodology process part II

Once all the files are revised, we must upload them to Drive to use them in Colab afterwards. Next step in the process is to codify a function that reads the file and then, converts the information in a DataFrame.

### 5.3.1 Steps for data processing

The function *"pandas.DataFrame.from_dict (json file path)"* was used to convert the scheduled flight files' information into a DataFrame. This DataFrame is displayed in figure 5.7 (the whole cell is placed in the Annex page 59).

| | callsign | request | geometry.coordinates |
|---|---|---|---|
| 0 | A_UTAH_418 | 2022-03-08 13:22:18 | [[1.99640653, 41.265035, 30, 0], [1.9964056491... |
| 1 | A_SWORD_544 | 2022-03-08 14:58:51 | [[1.96874383, 41.26500991, 30, 0], [1.96874338... |
| 2 | A_JUNO_348 | 2022-03-08 12:29:45 | [[1.97890135, 41.26524176, 30, 0], [1.97890060... |
| 3 | A_JUNO_142 | 2022-03-08 09:58:04 | [[1.97890135, 41.26524176, 30, 0], [1.97890060... |
| 4 | A_JUNO_288 | 2022-03-08 11:41:59 | [[1.97890135, 41.26524176, 30, 0], [1.97890049... |
| ... | ... | ... | ... |
| 221 | A_JUNO_238 | 2022-03-08 11:14:32 | [[1.97890135, 41.26524176, 30, 0], [1.97890060... |
| 222 | A_SWORD_113 | 2022-03-08 09:33:11 | [[1.96874383, 41.26500991, 30, 0], [1.96874338... |
| 223 | A_UTAH_253 | 2022-03-08 11:09:45 | [[1.99640653, 41.265035, 30, 0], [1.9964056491... |
| 224 | A_UTAH_528 | 2022-03-08 14:34:18 | [[1.99640653, 41.265035, 30, 0], [1.9964056491... |
| 225 | A_OMAHA_426 | 2022-03-08 13:32:25 | [[1.99106893, 41.26502089, 30, 0], [1.99106799... |

**Figure 5.7** Scheduled flights file's DataFrame of Day A

As we can see in the figure 5.7, this DataFrame is divided in three columns and 225 rows, one per each flight made on day A. The left column is named "callsign", and it is where the callsign of the flight is displayed. That has a special nomenclature; day of flight, then a low bar, flight operator, low bar and at last, an identificatory number. The middle column is used for the take off time of each flight and is called "request". Finally, the right column is named "geometry.coordinates", and it shows a long array that has inside more little arrays, one per each point of the flight, that are made up of the latitude, longitude, height and time of the flight. The coordinates data will be, in the following step, processed in order to create a DataFrame with the trajectory data.

Transforming a flight plan file written in KML language into DataFrame is a little hard process. First it is necessary to split every single row and place it in a new DataFrame. A KML row is composed of the type of data and the data itself in this way *"<type of data> data </type of data>".* Once we have the DataFrame with a column of type of data and another column with the data, we can take the important data for us from the DataFrame.
In our case, we needed the name of the waypoint, the speed in each point, the corner radius, the longitude, latitude and altitude.
An example of this DataFrame is displayed in the figure 5.8 (the whole cell is placed in the Annex page 58).

| | Name | Speed | CornerRadius | Lon | Lat | Alt |
|---|---|---|---|---|---|---|
| **0** | Waypoint1 | 5 | 64.0 | 1.99640653 | 41.265035 | 27 |
| **1** | Waypoint2 | 10 | 64.0 | 1.996405487464239 | 41.26387541519116 | 27 |
| **2** | Waypoint3 | 5 | 9.0 | 1.9939939631828607 | 41.263876615038775 | 27 |
| **3** | Waypoint4 | 3 | 9.0 | 1.9939941182468857 | 41.264056479322406 | 27 |
| **4** | Waypoint5 | 0 | 0.2 | 1.993994414806366 | 41.26440046653493 | 17 |
| **5** | Waypoint6 | 2 | 0.2 | 1.993994414806366 | 41.26440046653493 | 27 |
| **6** | Waypoint7 | 5 | 19.0 | 1.9939941182468857 | 41.264056479322406 | 77 |
| **7** | Waypoint8 | 10 | 39.0 | 1.9939934979954927 | 41.2633370221879 | 77 |
| **8** | Waypoint9 | 5 | 39.0 | 1.9965475584316519 | 41.2633357498461 | 77 |
| **9** | Waypoint10 | 3 | 39.0 | 1.9965482068221247 | 41.26405520696725 | 77 |
| **10** | Waypoint11 | 0 | 54.0 | 1.99654909 | 41.26503516 | 22 |

**Figure 5.8** Flight plan file's DataFrame

At last, it is needed to convert the real data file's information into a DataFrame. In this case, the process is very simple. It is only necessary to use the function *"Pandas.read_csv(filename)"* that will return straight away the DataFrame with the data contained in the file. An example of this DataFrame is displayed in figure 5.9 (the whole cell is placed in the Annex page 62).

| | time(millisecond) | datetime(utc) | latitude | longitude | height_above_takeoff(meters) | height_above_ground_at_drone_location(meters) | ground_elevation_at_drone_location(meters) |
|---|---|---|---|---|---|---|---|
| 0 | 41500 | 2022-03-10 09:29:51 | 41.265052 | 1.991126 | 0.0 | 0.00 | 1.585506 |
| 1 | 41600 | 2022-03-10 09:29:51 | 41.265052 | 1.991126 | 0.0 | 0.00 | 1.585506 |
| 2 | 41700 | 2022-03-10 09:29:51 | 41.265052 | 1.991126 | 0.0 | 0.00 | 1.585506 |
| 3 | 41800 | 2022-03-10 09:29:52 | 41.265052 | 1.991126 | 0.0 | 0.00 | 1.585506 |
| 4 | 41900 | 2022-03-10 09:29:52 | 41.265052 | 1.991126 | 0.0 | 0.00 | 1.585506 |
| ... | ... | ... | ... | ... | ... | ... | ... |

**Figure 5.9** Real data file's DataFrame

Figure 5.9 shows the flight data at different times presented in a DataFrame. Each row displays the information of the flight every 100 milliseconds. We can see some of the columns of the DataFrame in figure 5.9, there are some more,

but for us, the most important ones are time, datetime, latitude, longitude, and height above take-off.

In this point it is important to mention that, as we explained before, the take-off and landing in the VLDs were made manually, therefore the take-off data will not be useful for us. In addition the flight plan data does not have the one that belongs to the take-off or landing, so it is necessary to eliminate the take-off data from the real data DataFrame. That's why the following function “retocarcsv” is created.

The ”retocarcsv” function has the aim of reducing the number of samples, because there are many samples in real data DataFrame and also, eliminating the take-off data. This function first eliminates the rows that have the datetime duplicated in order to reduce the samples to one per each second. Then, the function concludes by eliminating the samples relative to the take-off and landing. To do this, the function firstly searches the first point where the altitude is bigger than 30 metres, that will be the point the take-off has ended, and eliminate all the prior points. Then, a column is added to the DataFrame which will be filled with “True” if the altitude is less than 25 metres and “False” if not. At last, the function eliminates all the rows with “True” that are in the end of the DataFrame, that means, the rows belonging to the landing.

## 5.4 Trajectories

After structuring the data in DataFrame, the following step is to paint the trajectories of the flights in order to understand them properly.

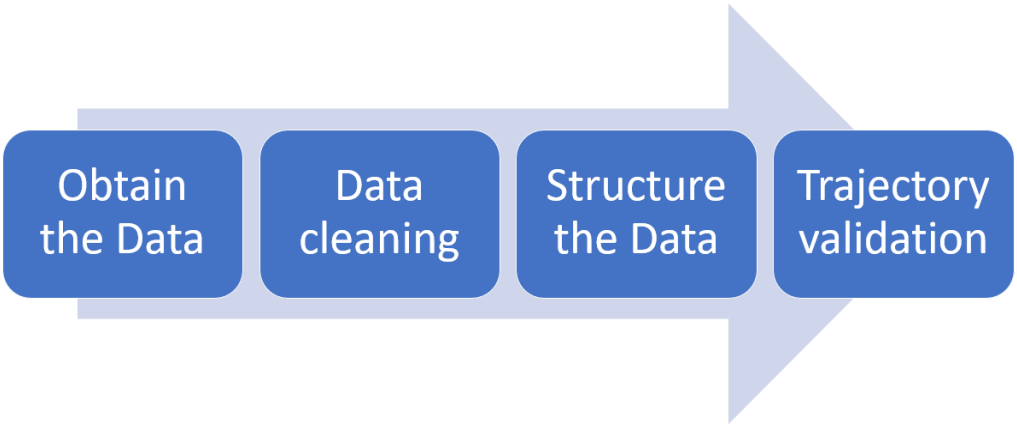


**Figure 5.10** Methodology process part III

Once all DataFrame are generated and modified, the following step is to create a function that selects the most important data of the flight and then paints their

trajectory. In this step it is important to compare the different trajectories of a flight in function of the scheduled flight, flight plan or real flight data.
In order to structure the scheduled flight trajectory, it is needed to make a function that first, searches in the DataFrame the row where the callsign is placed in order to paint its trajectory. Then, we select from the desired row of the DataFrame, the useful columns, that are the latitude, longitude, altitude and time. Later, we noticed that this function “trajectory” is useless when the data file is from the day A because the coordinates column of the DataFrame does not have the same format. Therefore, we adapt the function “trajectoryA” for this type of DataFrame. The trajectory DataFrame of a scheduled flight is presented in figure 5.11.

| | lon | lat | alt | secs |
|---|---|---|---|---|
| **0** | 1.996407 | 41.265035 | 30 | 0 |
| **1** | 1.996405 | 41.263875 | 30 | 26 |
| **2** | 1.993994 | 41.263877 | 30 | 47 |
| **3** | 1.993994 | 41.264056 | 30 | 51 |
| **4** | 1.993994 | 41.264400 | 20 | 64 |
| **5** | 1.993994 | 41.264400 | 20 | 74 |
| **6** | 1.993994 | 41.264400 | 30 | 79 |
| **7** | 1.993994 | 41.264056 | 80 | 92 |
| **8** | 1.993993 | 41.263337 | 80 | 108 |
| **9** | 1.996548 | 41.263336 | 80 | 130 |
| **10** | 1.996548 | 41.264055 | 80 | 146 |
| **11** | 1.996549 | 41.265035 | 25 | 183 |

**Figure 5.11** Scheduled flight trajectory DataFrame

For flight plan and real flight DataFrame, it is not necessary to modify anything because the DataFrame itself has the data needed to print the trajectory. So, once we have these three DataFrame, next step is to make a function that draws the three different trajectories. At first, it used the function *“Basemap()”*. Next, in order to get the X and Y positions of each type of DataFrame, is used the function *“map (longitude, latitude)”*. Finally, is plotted the different trajectories by the function *“plt.plot(X,Y)”*. The whole cell is placed in the Annex page 60.
Plotting the three trajectories was useful to prove if they are similar or identical. We find that some trajectories are identical, or at least, very similar and others

are simply contingencies. Next, we can see these examples in the figures 5.12, 5.13, 5.14 and 5.15.

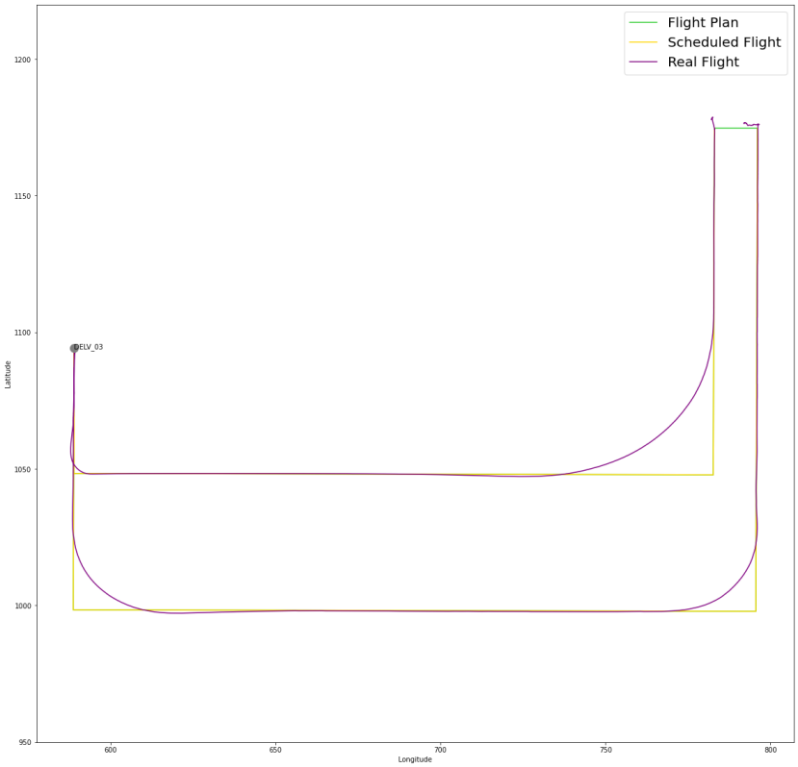


**Figure 5.12** Same trajectory example

The previous figure shows the trajectory of a flight. We can see the real trajectory of the flight painted in purple and, as it is an identical flight, it is almost impossible to see the flight plan and the scheduled flight. We can see in the turns the scheduled flight which is painted in yellow. The flight plan is identical, so we can't see it but it is painted in green. We can also see the point where the delivery is and the name of the delivery, that in this example is the delivery 03.

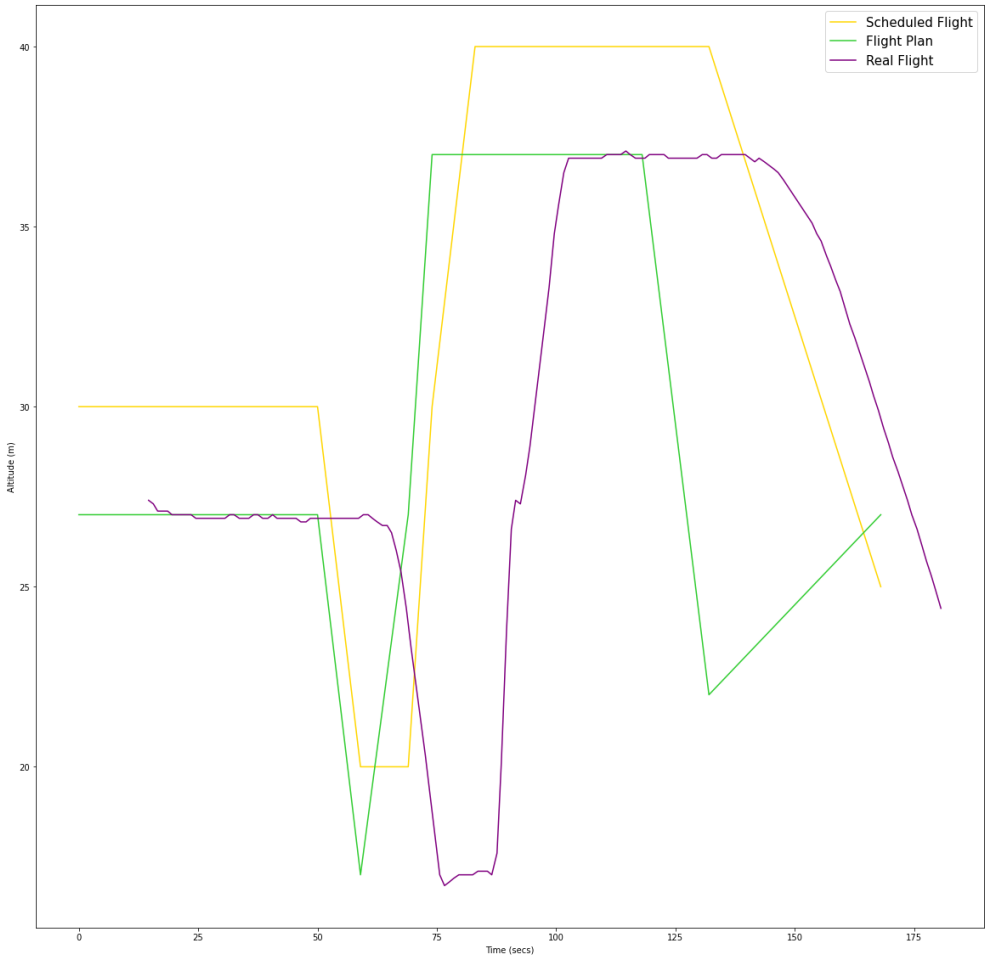


**Figure 5.13** Vertical profile of the same trajectory example

We can see in the figure 5.13 the vertical profile of the scheduled flight painted in yellow, and the flight plan painted in green. They do not have a delay on time but have a difference in altitude of about 3 metres. We also can see the real flight painted in purple, which is practically identical to the flight plan but with a delay of 15 seconds. If we compare the real flight with the scheduled flight, we can see also a delay of 15 seconds and also a vertical divergence of 3 metres.

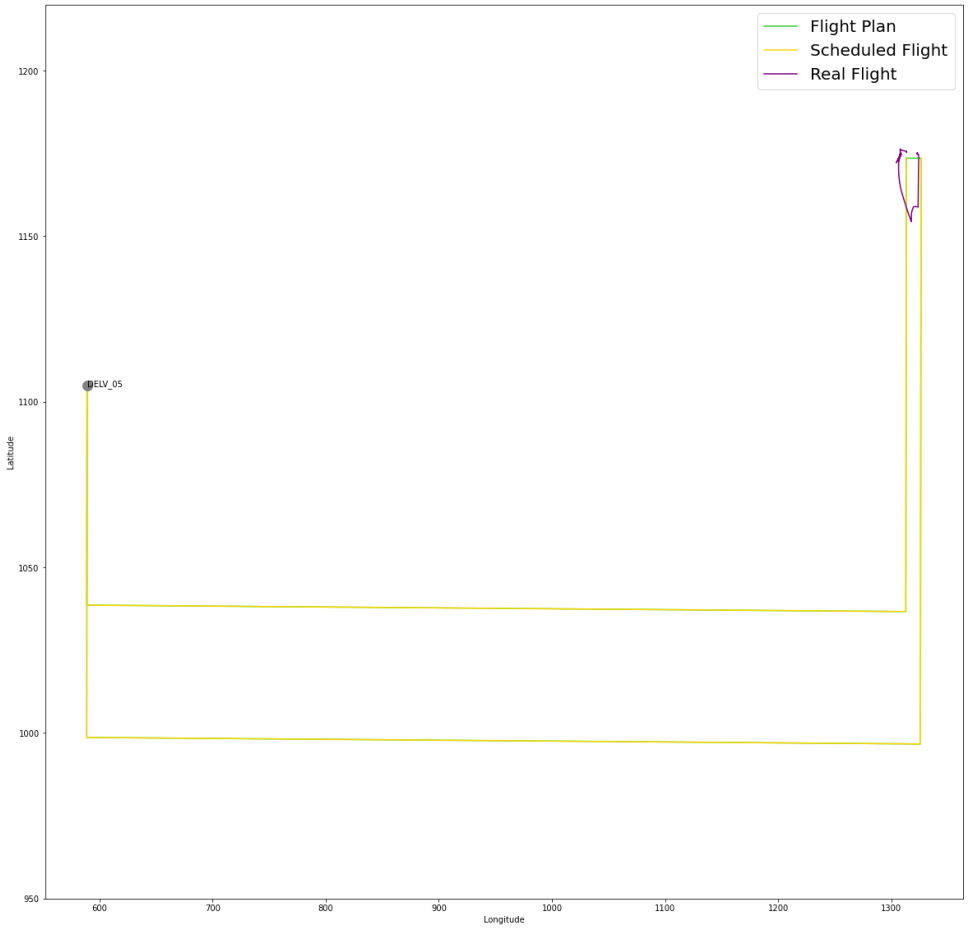


**Figure 5.14** Contingency trajectory example

This figure shows the trajectory of a contingency flight, and it is clearly differentiable between the scheduled flight and the real flight. As previously, the scheduled flight is painted in yellow and below is painted the flight plan in green, which cannot be seen because they are identical. Also, we can see the real flight painted in purple. It is also painted the delivery port with its name.

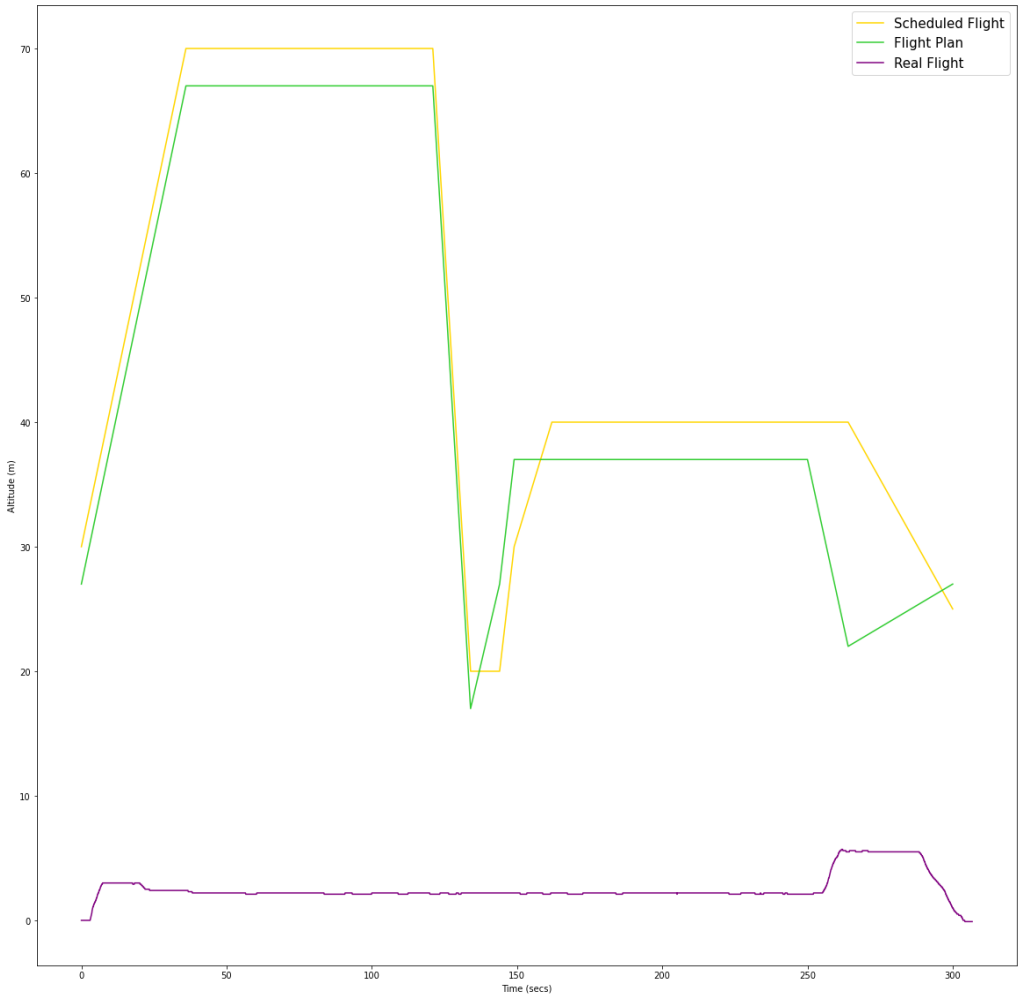


**Figure 5.15** Vertical profile of the contingency example

As before we can see the different vertical profiles. We can see the scheduled flight and the flight plan vertical profiles painted in yellow and green respectively, they have a divergence of 3 metres vertically but have practically no delay. We can also see in purple the real flight vertical profile. As it is a contingency the scheduled and real vertical profiles are clearly different. Even though in this case the flight time is the same, the altitude is clearly different because the contingency flight did not take enough height.

## 5.5 Flight matching

Next step in this prediction process is to match the flights.

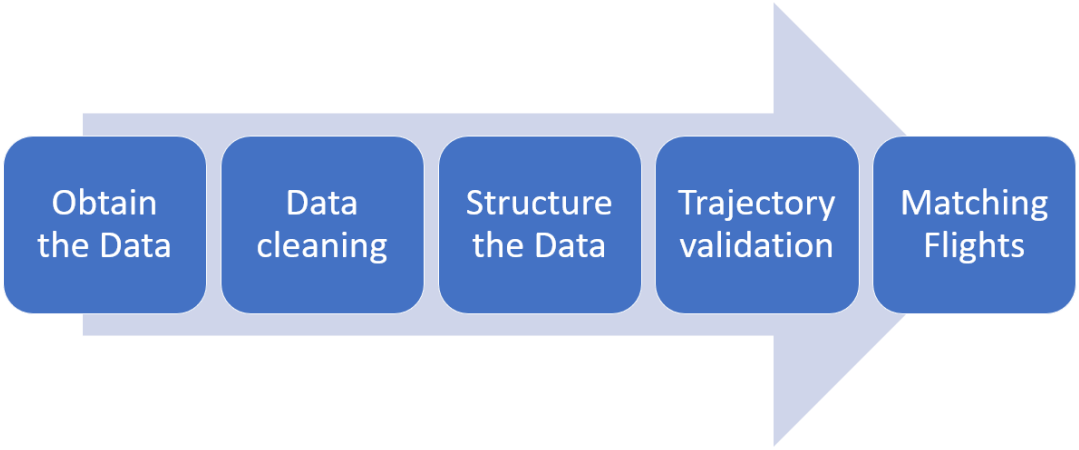


**Figure 5.16** Methodology process part IV

Next step is to match the scheduled flights and its flight plan with the performed flights recorded in real data files. This step is very important because it is necessary to, later, make the prediction table. The prediction table must have information about real and scheduled data, so we have to relate them.

This was the most difficult part of the program process because real flight files did not have registered the callsign of the flight performed, so, we had to match real flight data with scheduled flight data through the take-off time. At first, we started matching the flights one by one manually, but we realised that it was impossible because there were too many flights, about more than 650, and the take-off time was not the same in scheduled flights than in real flights. We spent a lot of time with this problem. At first, we thought that maybe some flights were not performed or recorded correctly, but there were too many flights that did not add up, so, that was an unbelievable hypothesis. After some weeks, we realise that there is one scheduled flight file per each day, and we must relate the data taking into account the day of the flight.

After all these problems, we decided to do a function that searches every flight recorded in the scheduled flights file and try to relate it with a real flights file, considering the take-off time, the delivery port and the vertiport. As for U-Space strategic deconfliction service the safety margin is about 3 minutes, the function checks if the take-off time is the same with a margin of 3 minutes after the desired take-off time. Then, the delivery port is calculated and if it's the same, we consider that the real flight and the scheduled flight are related. In the case that the delivery port is not the same, we looked at the file manually in order to

find if the flights are the same or if the calculations are correct. To confirm our hypothesis, we looked at the trajectory in google earth or with the trajectory plot function to see if they are the same.

Later we make the function “renameCSV” that rename the CSV’s files in order to change the name of the real flight file if we have found the related flight in scheduled flights files with the previous parameters. The criteria taken for renaming the real flight file is to name it in the same way as the scheduled flights file. That means, add at the beginning the day of the flight, A for first day, B for second day or C for the last day. After, add a low bar and next the callsign of this flight. Then, another low bar and finally, DELV with a low bar and then the number of the delivery port of the flight. For example “C_SWORD_928_DELV_10” is the flight number #928 is the 3rd day from SWORD to Delivery 10.

Once all the real flights files have been renamed, we make a function that reads every file and then creates a DataFrame for each type of file and adds there the name of the flights scheduled or performed. We made one table per each day and operator. So, as it is explained, these files have 3 columns, one for the scheduled flights, other for the flight plan and the last for the real flights. The name for each flight added to the table has the day of the flight, its callsign, its take-off time and its delivery port. Finally, these DataFrames are exported to a text file and saved in Drive. Figure 5.17 shows an example of this text file.

```
,callsignJSON,callsignKML,callsignCSV
0,C_UTAH_19_08:18:16_DELV_03,C_UTAH_19_DELV-03,C_UTAH_747_DELV_01
1,C_UTAH_20_08:02:17_DELV_02,C_UTAH_20_DELV-02,C_UTAH_945_DELV_02
2,C_UTAH_27_08:32:06_DELV_04,C_UTAH_27_DELV-04,C_UTAH_1005_DELV_04
3,C_UTAH_30_08:07:50_DELV_02,C_UTAH_30_DELV-02,C_UTAH_1112_DELV_04
4,C_UTAH_33_08:25:05_DELV_03,C_UTAH_33_DELV-03,C_UTAH_1492_DELV_01
5,C_UTAH_34_08:13:12_DELV_02,C_UTAH_34_DELV-02,C_UTAH_1720_DELV_02
6,C_UTAH_42_08:48:09_DELV_01,C_UTAH_42_DELV-01,
7,C_UTAH_47_08:39:31_DELV_05,C_UTAH_47_DELV-05,
8,C_UTAH_76_08:53:47_DELV_01,C_UTAH_76_DELV-01,
9,C_UTAH_129_08:59:35_DELV_02,C_UTAH_129_DELV-02,
10,C_UTAH_180_09:05:06_DELV_03,C_UTAH_180_DELV-03,
11,C_UTAH_231_09:11:34_DELV_05,C_UTAH_231_DELV-05,
12,C_UTAH_309_09:20:34_DELV_01,C_UTAH_309_DELV-01,
13,C_UTAH_352_09:26:49_DELV_05,C_UTAH_352_DELV-05,
14,C_UTAH_426_09:35:54_DELV_05,C_UTAH_426_DELV-05,
15,C_UTAH_485_09:44:27_DELV_02,C_UTAH_485_DELV-02,
16,C_UTAH_543_09:49:35_DELV_01,C_UTAH_543_DELV-01,
17,C_UTAH_589_09:55:31_DELV_03,C_UTAH_589_DELV-03,
18,C_UTAH_649_10:02:04_DELV_01,C_UTAH_649_DELV-01,
19,C_UTAH_701_10:07:47_DELV_01,C_UTAH_701_DELV-01,
20,C_UTAH_747_10:14:15_DELV_01,C_UTAH_747_DELV-01,
21,C_UTAH_832_10:20:32_DELV_02,C_UTAH_832_DELV-02,
22,C_UTAH_858_10:25:46_DELV_05,C_UTAH_858_DELV-05,
23,C_UTAH_945_10:35:11_DELV_02,C_UTAH_945_DELV-02,
24,C_UTAH_1005_10:43:25_DELV_04,C_UTAH_1005_DELV-04,
25,C_UTAH_1096_10:50:59_DELV_02,C_UTAH_1096_DELV-02,
26,C_UTAH_1112_10:56:41_DELV_04,C_UTAH_1112_DELV-04,
27,C_UTAH_1175_11:04:26_DELV_02,C_UTAH_1175_DELV-02,
28,C_UTAH_1214_11:10:13_DELV_05,C_UTAH_1214_DELV-05,
29,C_UTAH_1307_11:18:53_DELV_01,C_UTAH_1307_DELV-01,
30,C_UTAH_1353_11:24:40_DELV_01,C_UTAH_1353_DELV-01,
31,C_UTAH_1416_11:33:53_DELV_05,C_UTAH_1416_DELV-05,
32,C_UTAH_1492_11:42:22_DELV_01,C_UTAH_1492_DELV-01,
33,C_UTAH_1550_11:48:16_DELV_05,C_UTAH_1550_DELV-05,
34,C_UTAH_1611_11:57:32_DELV_02,C_UTAH_1611_DELV-02,
35,C_UTAH_1666_12:03:09_DELV_02,C_UTAH_1666_DELV-02,
36,C_UTAH_1720_12:08:59_DELV_02,C_UTAH_1720_DELV-02,
```

**Figure 5.17** Example of Flights file

## 5.6 Data processing for prediction table

Finally, before making the prediction, it is needed to structure the data in a prediction table.

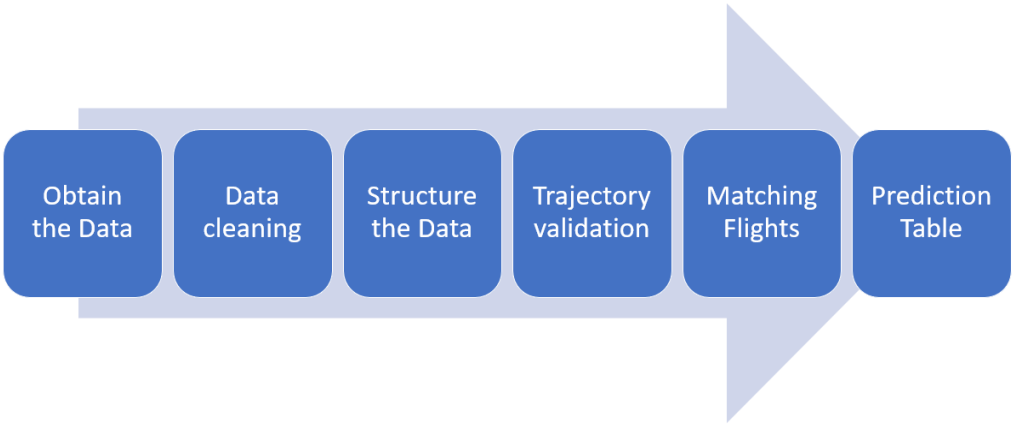


**Figure 5.18** Methodology process part V

Our objective is to make a DataFrame that contains the data that will be useful for prediction. So, as we want to predict, at first, the time of delivery of a particular flight, we need to know for each planned flight its latitude, longitude, and height when it is at entering of the airway, its latitude, longitude, and height when it is at the delivery point and the cruise altitude and velocity and at last, the real time of entering to the airway.
After these text files are made, the function checks if there are any real flights made in it, because if not, all the flights operated by this operator that day are contingencies and there is not useful data for prediction. Then, we take from the name of the real flight using the split tool; the day, the operator, the callsign and the delivery port.
Using the trajectory function (the whole function is placed on the Annex page 55) and the information obtained in the previous step, we get the trajectory DataFrame of the scheduled flight. Later, this DataFrame will help us to know the delivery altitude, by searching the trajectory position where the latitude and the longitude are the same that the ones defined previously of each delivery port.
In order to find the cruise velocity, we have to search in the flight plan the row where the altitude of the following point is smaller than in the current one, that will be the cruise altitude. Then, it is needed to find at the flight plan file which is the velocity when the flight is at cruise altitude and do the average of all the speed data found. From the flight plan, we also take the corner radius at the time entering the airway. From the real flight file, we take the time entering the airway and the delivery time. Finally, the entering airway latitude, longitude and height we get it from the trajectory file and the delivery latitude and longitude is

obtained from the data of the delivery ports defined previously at the beginning of the program.
Every data explained above is what defines the columns of this prediction DataFrame, the rows are defined by all the performed flights. The time of delivery is placed on the Y prediction DataFrame, that means, that is the data that we want to predict. The rest of the data is placed on the X prediction DataFrame. Figure 5.19 shows a section of these DataFrame.

| | lat0 | lon0 | alt0 | cornerRadius0 | latdelv | londelv | altdelv | altcreuer | velcreuer | Time Entering Airway (min) |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 41.265242 | 1.978901 | 30.0 | 65.0 | 41.264380 | 1.980790 | 30.0 | 40.0 | 8.25 | 538.0 |
| **1** | 41.265242 | 1.978901 | 30.0 | 65.0 | 41.264429 | 1.974632 | 30.0 | 70.0 | 7.75 | 707.0 |
| **2** | 41.265242 | 1.978901 | 30.0 | 75.0 | 41.264413 | 1.971627 | 30.0 | 30.0 | 8.60 | 723.0 |
| **3** | 41.265242 | 1.978901 | 30.0 | 75.0 | 41.264429 | 1.974632 | 30.0 | 30.0 | 8.40 | 731.0 |
| **4** | 41.265021 | 1.991069 | 30.0 | 53.0 | 41.264602 | 1.999477 | 30.0 | 80.0 | 8.00 | 524.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **70** | 41.265035 | 1.996407 | 30.0 | 64.0 | 41.264400 | 1.993994 | 30.0 | 30.0 | 5.20 | 575.0 |
| **71** | 41.265035 | 1.996407 | 30.0 | 64.0 | 41.264292 | 1.987206 | 30.0 | 30.0 | 4.80 | 583.0 |
| **72** | 41.265035 | 1.996407 | 30.0 | 54.0 | 41.264292 | 1.987206 | 30.0 | 70.0 | 5.25 | 596.0 |
| **73** | 41.265035 | 1.996407 | 30.0 | 54.0 | 41.264602 | 1.999477 | 30.0 | 80.0 | 5.50 | 643.0 |
| **74** | 41.265035 | 1.996407 | 30.0 | 64.0 | 41.264400 | 1.993994 | 30.0 | 30.0 | 5.00 | 669.0 |

**Figure 5.19** X prediction DataFrame

| | Time of Delivery |
|---|---|
| **0** | 539.0 |
| **1** | 710.0 |
| **2** | 725.0 |
| **3** | 732.0 |
| **4** | 524.0 |
| **...** | ... |
| **70** | 577.0 |
| **71** | 585.0 |
| **72** | 599.0 |
| **73** | 645.0 |
| **74** | 670.0 |

**Figure 5.20** Y prediction DataFrame

## 5.7 Folder schema

All the files defined before are uploaded in Drive and structured in different folders. The structure and its contents are explained in detail below.

In the CSVs folder is saved the real flight files renamed with the parameters explained in the section 5.5. Flight matching. The CSV1 folder contains the real flight files with the original names. The flight plan files are placed in the KML folder. In the JSON folder we can find the scheduled flights file and a text file per each day and operator with the information of the scheduled flights showing its callsign, take-off time and delivery. Finally, in the TXTFLIGHTS folder we can find the text generated in the step explained at the end of the section 5.5. Flight matching, of each operator and day where is placed the name of all the flights scheduled and performed.

## 5.8 Prediction through Machine Learning

Finally, we make the prediction process with Machine Learning and then we analyse the results.

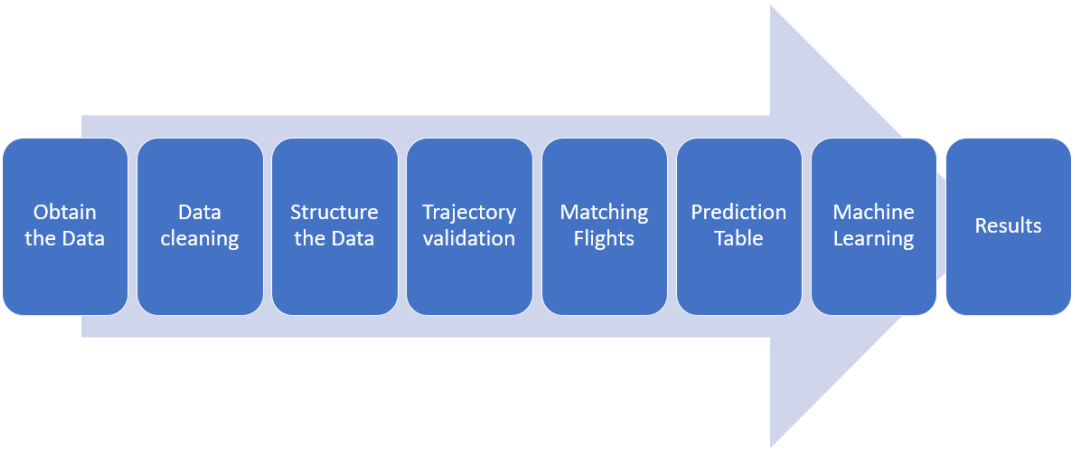


**Figure 5.21** Methodology process part VI

As soon as the predicting table is completed, the last step is to use the Machine Learning to do the prediction itself. The first thing we must consider when it comes to predicting something, is to know what we want to predict and what type of data is. In our case, we want to predict, at first, the time of delivery in the port. Our type of data aims to predict something that is limited, does not vary over time, and only exists in a single group of results. Hence, the machine learning model that fits better with our type of data is the Regression model.

In order to use a model, there are some steps that are always the same. The first step is to split the DataFrame in two different parts, the test table, and the train table. This procedure is used to analyse the efficiency of the Machine Learning algorithm. Usually, the data is divided in two parts, 80% of the data is

assigned for training data, and the 20% is assigned for test data. Once the data is divided, next step is to train it. To do it, we have to use the sklearn function *"train_test_split(dfx, dfy, test_size)"*. In this function we will have to insert the X prediction table, the Y prediction table and at last, the number of test size, that in our case is 0,2. After training it, the function output will be the X train vector, the X test vector, the Y train vector and the Y test vector.

Finally, we have to fit the model, to do it we have to define the model and then use the function *"model.fit(X_train, Y_train)"* that will fit the model with the vectors inserted. In our case we choose the regression models. We use some different algorithms. It is convenient to try as many as possible, to see which will be the best, that means, the one that has less error, so the one that fits better the prediction. For this project we use the following algorithms, explained more in detail in the section 2.2. Machine Learning Algorithms.

- Linear Regression
- Logistic Regression
- Ridge Regression
- Bayesian Linear Regression
- Elastic Net Regression

After fitting the model, we must predict the data. To do it, we have to use the function *"modelfit.predict(X_test)"* defining modelfit as the result of the fitting function.

Finally, the program will give us an Y prediction vector with the results of the prediction that the machine has calculated. The result of the prediction is by the moment, not very understandable, not quite optimal, so next step is to represent the result in a proper way.

There are some functions that give us the result in different ways. In our case, we used two different functions. The Mean Absolute Error (MAE from here on out) function that is displayed as follows "mean_absolute_error(y_test, y_pred)" and is used to know the average difference between the calculated values and actual value. The Root Mean Squared Error (RMSE from here on out) is used to know the average of the divergence between the observed values and the predicted values. The RMSE is defined in numpy as follows "sqrt(metrics.mean_squared_error(y_test, y_pred)". The last function we used is the Score function which is defined as "modelfit.score(X_train,y_train)" and is used to know the accuracy of the model, that means in a few words, how "good" the machine learns the model.

# 6 CHAPTER 6. RESULTS

## 6.1 Flight analysis

We thought that in order to see and understand the flight information better it is easy to see it visually in different plots in function of different parameters. This type of graph gives us a lot of information and is also useful for explaining the information to others.

Figure 6.1 was the first graph we made. This plot shows the information about the total programmed flights per each operator and each day. In the X axis we can see the days that the flights were programmed, in the Y axis we can see the amount of flights programmed and each bar represents each operator. As we can see in the legend, the JUNO operator is represented in blue, the OMAHA operator in orange, the UTAH operator in green and the SWORD operator in red.

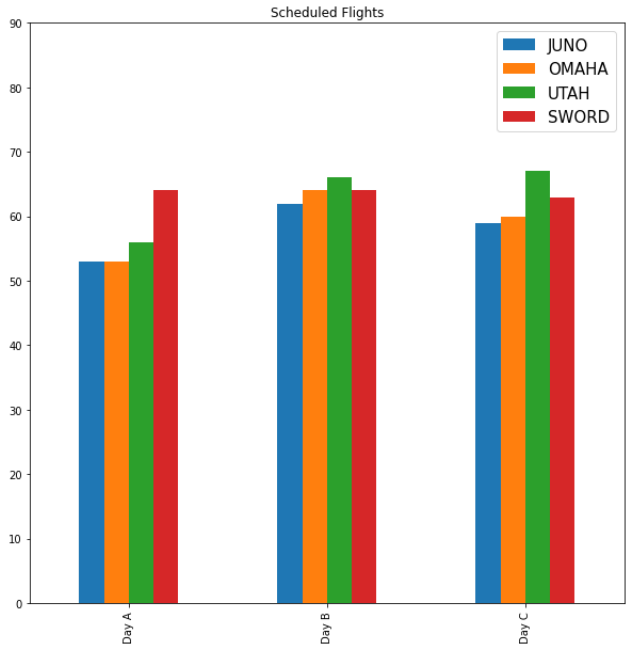


**Figure 6.1** Programmed flights per each day and operator

Emphasising the figure 6.1, we can see that the first day JUNO and OMAHA program less flights than the others and SWORD programs the most flights, with more than 60 flights. If we compare the first with the second day, we can see that all the operators except SWORD, that program less, program more flights. Day B the operator that programs more flights is UTAH, followed by OMAHA and SWORD, JUNO this day is also the operator that programs less flights. The last day, the operators OMAHA, JUNO and SWORD reduce the amount of flights programmed. Instead, UTAH increased a little the flights programmed. Also, we can see that on the last day, JUNO and OMAHA are the operators that program less flights, followed by SWORD and then, UTAH that program almost 70 flights.

Making an overview of the three days, JUNO is the operator that plans the fewest number of flights on average. Then we would have OMAHA with the amount of 177 flights scheduled. Finally, UTAH and SWORD are the operators with the highest average of scheduled flights. As an own reflection, it is not as important to schedule many flights as to fly them.

The figure 6.2 shows the amount of flights realised per each operator, each day. The Y axis shows the amount of flights that if we compared with the previous graph we can see that the maximum number of flights performed compared to the flights programmed is significantly reduced. That means that although the operators program a huge amount of flights, they cannot perform all of them. The X axis shows the day when the flights are performed.

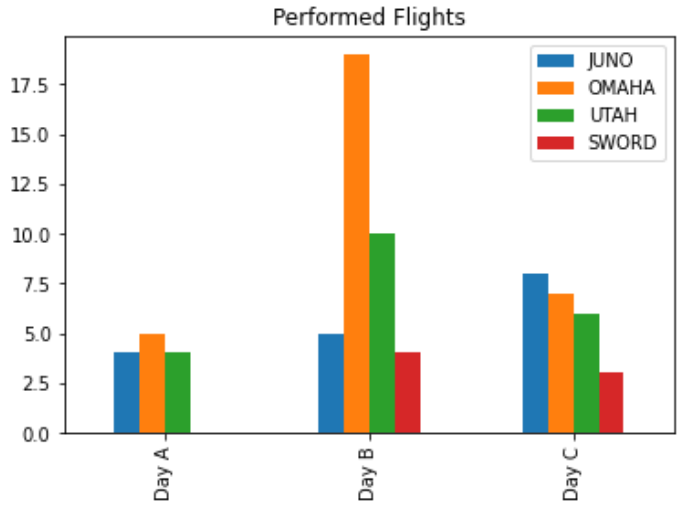


**Figure 6.2** Performed flights per each day and operator

In the figure 6.2 we can see that the first day a little amount of flights were performed. That day JUNO and UTAH performed more or less the same number of flights and OMAHA was the operator that performed more flights, performing the amount of 5 flights. Instead, SWORD did not fly any flight.

The second day, the number of flights performed increased. SWORD is the operator with the fewest flights of the day. Ahead it is JUNO with 5 flights performed and UTAH with 10 flights. This day, the operator with more operations is OMAHA which almost quadruples the number of the previous day.

The last day, the amount of flights performed went back down. The operator with less flights was SWORD followed by UTAH and OMAHA and as a difference, JUNO is the operator with more flights performed the last day.

The figure 6.3 represents the amount of flights scheduled, performed and contingencies made by OMAHA in the different days. In general, we can see that the amount of flights scheduled compared to those performed is much larger. The first day this difference is abysmal, with a divergence of around 50 flights. The second day we can see that the difference is reduced and the last day, this divergence is drastically reduced. The amount of contingencies is in the three days very small, with a maximum of 2 contingencies in the second day. Therefore, we can consider the contingencies as not worrying data.

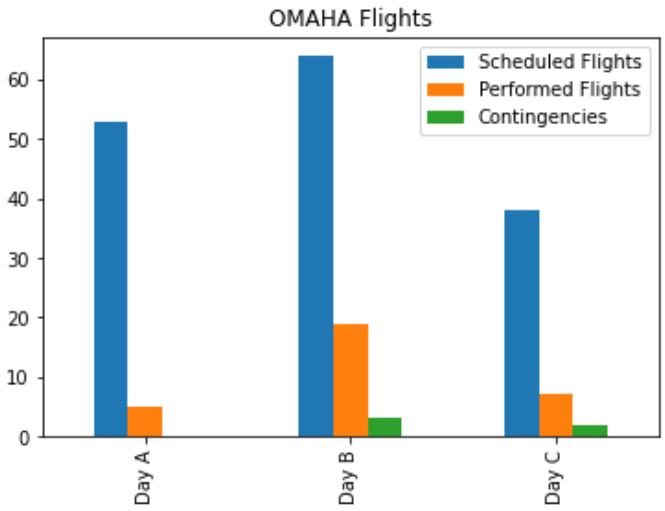


**Figure 6.3** Scheduled and performed flights by OMAHA

The figure 6.4 shows, as the previous one, the number of flights programmed, performed and contingencies by JUNO operator in the three days. As in the previous one, the difference between the scheduled and the performed flights is very significant. In this case, the difference is bigger than with the OMAHA operator. JUNO only has 1 contingency the last day. So, we can still consider the contingency, a not worrisome data.

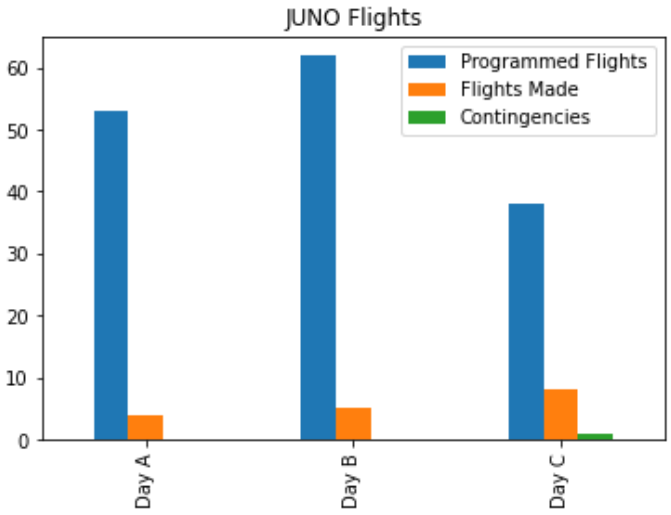


**Figure 6.4** Scheduled and performed flights by JUNO

The UTAH operator, as we can see in the figure 6.5, has also a big divergence between the scheduled flights and the performed flights. This difference is more or less the same than with the two previous operators. The contingency is again a not worrying data since there is only one contingency flight operated the second day.

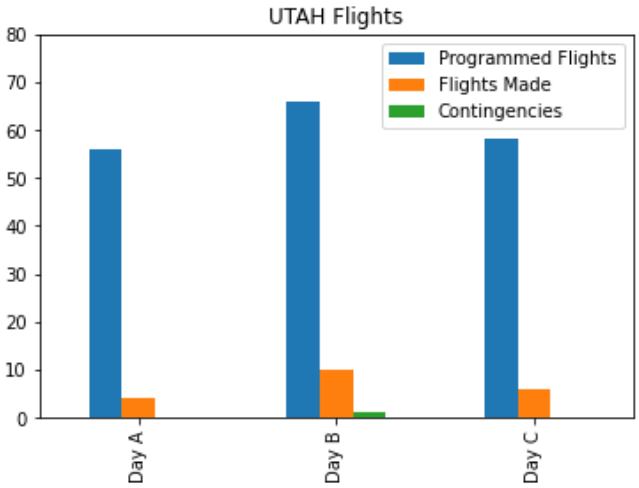


**Figure 6.5** Scheduled and performed flights by UTAH

The last operator, SWORD, shows in the figure 6.6 a divergence between performed and scheduled much larger than the previous ones. The first day, we can see that the number of flights programmed were 60 and there were any

flights performed. The second day, 60 were scheduled and was performed no more than 5. The last day, the number of flights programmed was reduced, but the number of flights performed was also. There were 3 flights performed and 4 contingencies In this case, the number of contingency flights are bigger but not worrying, in no case does it exceed the 5 contingency flights in a day.

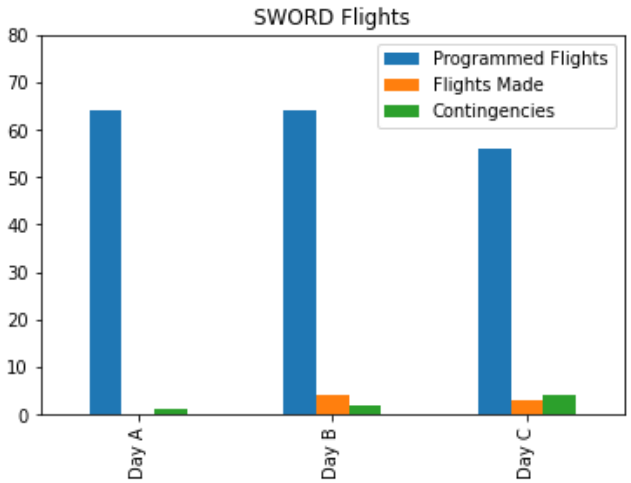


**Figure 6.6** Scheduled and performed flights by SWORD

The figure 6.7 shows what we have been seeing so far. The number of flights scheduled compared to the flights performed has a big difference.

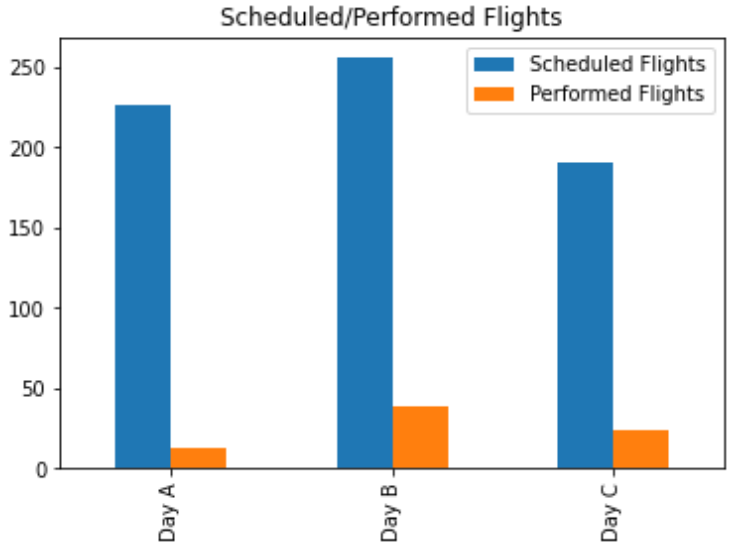


**Figure 6.7** Scheduled and performed flights by all the operators

## 6.2 Prediction results

In this section, the prediction results from time of delivery will be displayed. After making all the steps explained in the previous chapter, machine learning will give us the following results data. In our case, first we try to predict the time of delivery of a flight. In order to do this, we prove different types of models and some types of functions that explain the results in different ways.

Next, the results are displayed in tables 6.1, 6.2, 6.3, 6.4 and 6.5. Each table belongs to each model. Each one has some columns that are associated with the different types of results functions used. The rows belonging to each run did it in the prediction, the last row is reserved for the mean of each run results.

| | MAE (minutes) | RMSE (minutes) | Score (%accuracy) |
|---|---|---|---|
| 1st Run | 0,825246448496667 | 1,14584735085511 | 0,9998121504360 |
| 2nd Run | 1,542014504197860 | 2,43314712865172 | 0,9999455551668 |
| 3rd Run | 0,881523386789679 | 1,29813881588196 | 0,9998302967796 |
| 4th Run | 0,584434589733791 | 0,65615541764733 | 0,9998055838284 |
| 5th Run | 0,593499539530118 | 0,69482173005167 | 0,9997894391698 |
| 6th Run | 0,710315532918563 | 1,10518170950124 | 0,9998236868500 |
| 7th Run | 0,938565259114693 | 1,53358505891121 | 0,9998425731768 |
| 8th Run | 0,565396923775551 | 0,69782359034944 | 0,9998029863076 |
| 9th Run | 0,556311341461453 | 0,642740379569623 | 0,9997750797583 |
| 10th Run | 1,120940657878350 | 2,192312058555160 | 0,9999141517648 |
| **Mean** | **0,767780990707615** | **1,125514530178180** | **0,9998179186430** |

**Table 6.1** Prediction results with Linear Regression Model

| | MAE (minutes) | RMSE (minutes) | Score (%accuracy) |
|---|---|---|---|
| 1st Run | 16,266666666 | 31,36976056643281 | 0,56666666667 |
| 2nd Run | 20,266666666666666 | 27,995237690245343 | 0,4 |
| 3rd Run | 32,13333333333333 | 44,00303019868821 | 0,4666666666666667 |
| 4th Run | 23,866666666666667 | 30,201545214331887 | 0,5 |

**Table 6.2** Prediction results with Logistic Regression Model

| | MAE (minutes) | RMSE (minutes) | Score (%accuracy) |
|---|---|---|---|
| 1st Run | 0,82991327478812 | 1,19352747905960 | 0,9997893060081 |
| 2nd Run | 1,42585818285609 | 2,33376363357557 | 0,9999010000431 |
| 3rd Run | 1,11830807608275 | 1,34291965778656 | 0,9998049924319 |
| 4th Run | 1,15696065761160 | 1,54598707251556 | 0,9998257592239 |
| 5th Run | 1,14909832433573 | 1,54374059533946 | 0,9998116210211 |
| 6th Run | 0,64305703765647 | 0,85451105567601 | 0,9997590952975 |
| 7th Run | 1,34512658303014 | 2,13555692013043 | 0,9998997241782 |
| 8th Run | 0,91712527723617 | 1,41607352325226 | 0,9998141250196 |
| 9th Run | 1,23834694099081 | 2,11051154922602 | 0,9998883238939 |
| 10th Run | 1,03506695337239 | 1,96128750578478 | 0,9998701646420 |
| **Mean** | **1,133703200209240** | **1,544863833927510** | **0,9998199421217** |

**Table 6.3** Prediction results with Ridge Regression Model

| | MAE (minutes) | RMSE (minutes) | Score (%accuracy) |
|---|---|---|---|
| 1st Run | 1,00130271746060 | 1,36167738114975 | 0,999808547412885 |
| 2nd Run | 0,69301734046664 | 0,86329114333595 | 0,999776928252762 |
| 3rd Run | 0,85037158262154 | 1,00734744130254 | 0,999791665960475 |
| 4th Run | 0,71418670246438 | 0,89603367351382 | 0,999796212560750 |
| 5th Run | 0,82217687284856 | 0,94720900777748 | 0,999787643122995 |
| 6th Run | 0,72754258831211 | 1,14222292982484 | 0,999793433495611 |
| 7th Run | 1,43152012751642 | 2,38349854797981 | 0,999917505116647 |
| 8th Run | 0,74852542631031 | 0,99956528375089 | 0,999784798239090 |
| 9th Run | 1,36194250130285 | 2,24843738801856 | 0,999901641505013 |
| 10th Run | 0,83616061245693 | 0,95959470906614 | 0,999793539829642 |
| **Mean** | **0,82916874265275** | **1,00345636252671** | **0,99979348666263** |

**Table 6.4** Prediction results with Bayesian Linear Regression Model

| | MAE (minutes) | RMSE (minutes) | Score (%Accuracy) |
|---|---|---|---|
| 1st Run | 0,488817953136056 | 0,626628824162782 | 0,999759419373152 |
| 2nd Run | 0,888960017006941 | 1,043999295752000 | 0,999778785463948 |
| 3rd Run | 0,825170926928497 | 0,942095423488324 | 0,999756681507377 |
| 4th Run | 0,797505392313860 | 0,991088044658980 | 0,999796238590901 |
| 5th Run | 0,808127412631211 | 1,062132500900980 | 0,999764039947374 |
| 6th Run | 0,718313149129244 | 1,009814653244770 | 0,999788974039623 |
| 7th Run | 1,207727949253770 | 2,034534951387060 | 0,999869501507223 |
| 8th Run | 0,961874505623844 | 1,285694038684730 | 0,999796624381939 |
| 9th Run | 1,020944800907470 | 1,246675495477200 | 0,999787107087608 |
| 10th Run | 0,965550124499388 | 1,163899460248860 | 0,999788508450030 |

| | | | |
|---|---|---|---|
| **Mean** | **0,85706547196772** | **1,05306589832649** | **0,99978780776882** |

**Table 6.5** Prediction results with Elastic Net Regression Model

In Tables 6.1, 6.2, 6.3, 6.4 and 6.5, we can see that in all the models, except the Logistic Regression Model, the error is about 1 minute and the accuracy is about 0,999%.

With the results presented before, we make some graphs in order to see them visually and examine in detail the numbers. The first graph shows the results of the MAE and RMSE for each algorithm model. We have to take into account that the smaller these results are, the better the prediction is. Therefore, it is obvious that the Ridge Regression Model is the algorithm with worse results, so we discard it. Now, we can see that the Linear Regression Model has better MAE but the Bayesian Linear Regression Model has the best RMSE. The Elastic Net Regression Model has a mix of both. So as we see we cannot affirm which is better, it is needed to analyse the score results.


**Figure 6.8** MAE and RMSE results for each algorithm

The figure 6.9 shows the score results for each algorithm. As this results show the % of accuracy of the algorithm, that means, how close it gets to reality, it is needed to obtain as close as possible to 1. So, taking into account the previous results, the best algorithm for this prediction is the Linear Regression Model. Even so, the difference between the accuracy of all the algorithms is practically null.

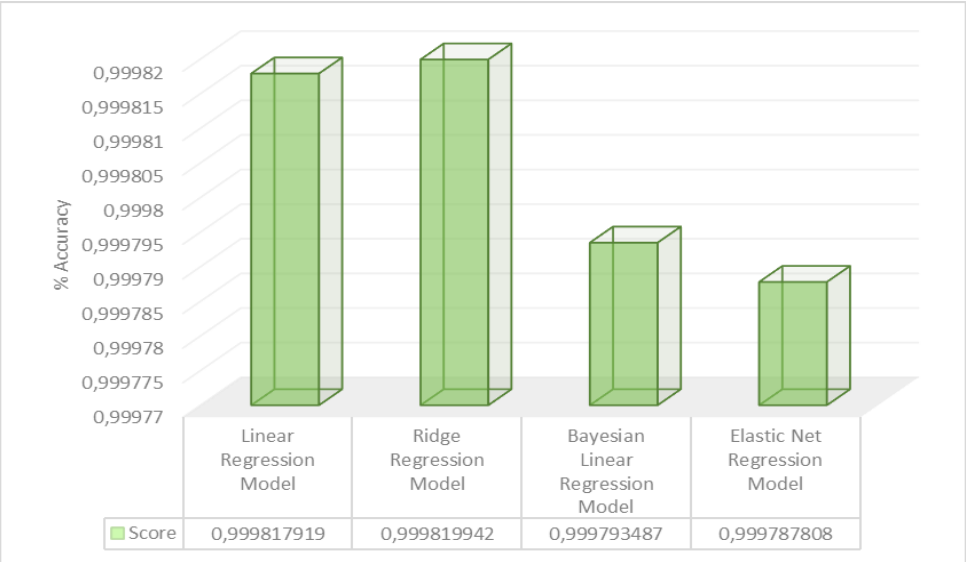


**Figure 6.9** Score results for each algorithm

# CHAPTER 7. CONCLUSIONS AND FURTHER IMPLEMENTATIONS

The last chapter of this work, analyses the results exposed in the previous chapter in addition to understanding it and commenting on them.

It can be seen that for any model shown in the figure 6.9, the accuracy will be good enough to make the VLDs safely since it is within the 3-minute range required by the U-Space, even so, it could be improved. So, the objective of this project is reached, the VLD flights carried out have been analysed and it has been seen that it is possible to predict with 1 minute accuracy the time when a drone will arrive at its delivery port in a UAM with an urban environment with restrictions, traffic, and emergency urban services.

From the VLDs carried out by the CORUS-XUAM project in Castelldefels, it can be shown that the U-Space can have enough capacity to share airspace between drones and other aerial vehicles without risks. Although, the capacity that the organisation of the U-Space allowed was much greater than the one used by the operators for the flights made.

From this point of analysis, new fields could be investigated, such as the prediction of the entire trajectory of the drone, so that it can be known exactly where the drone is at all times. This will allow delivery companies to create an application for the mobile phone where the customer could know at all times where their package is. Or even, get the drone to fly autonomously from the prediction of its position and the sensors it incorporates. Only being controlled by an ATC. On the other hand, more demonstration flights could also be made with a higher capacity in the airspace, being able to reach the maximum that the organization of the U-Space allows.

# BIBLIOGRAPHY

[1] AESA. (s. f.). *¿Tienes un UAS/dron? | AESA-Agencia Estatal de Seguridad Aérea - Ministerio de Fomento*. AESA Agencia Estatal de Seguridad Aérea. https://www.seguridadaerea.gob.es/es/ambitos/drones/tienes-un-uas-dron

[2] DroneOmega.com. (2018). *The Beginner's Guide to Drone Motor Essentials*. Drone Omega. https://droneomega.com/drone-motor-essentials/

[3] Allain, R. (2017, 19 mayo). *The Physics of How Drones Fly*. WIRED. https://www.wired.com/2017/05/the-physics-of-drones/

[4] European Comission. (s. f.). *Urban mobility*. Mobility and Transport. https://transport.ec.europa.eu/transport-themes/clean-transport-urban-transport/urban-mobility_en

[5] CORUS-XUAM. (2021). *About CORUS-XUAM - Urban Air Mobility for Europe*. CORUS-XUAM - Concept of Operations for Urban Air Mobility in Europe. https://corus-xuam.eu/about/

[6] CORUS-XUAM. (2022). *CORUS-XUAM - Urban Air Mobility for Europe - Home*. CORUS-XUAM - Concept of Operations for Urban Air Mobility in Europe. https://corus-xuam.eu/

[7] ICARUS DRONE SYSTEMS. (2022). *¿Qué es el U-Space?* https://icarusds.es/que-es-el-u-space/

[8] Capitán, C., Pérez-León, H., Capitán, J., Castaño, N., Rodríguez, A., & Ollero, A. (2021). Unmanned Aerial Traffic Management System Architecture for U-Space In-Flight Services. *Applied Sciences, 11*(9), 3995. https://doi.org/10.3390/app11093995

[9] SAP Insights. (s. f.). *What is machine learning?* SAP. https://www.sap.com/insights/what-is-machine-learning.html

[10] IBM Cloud Education. (2022b). *Machine Learning*. IBM. https://www.ibm.com/cloud/learn/machine-learning#:%7E:text=Machine%20learning%20is%20a%20branch%20of%20artificial%20intelligence,IBM%20has%20a%20rich%20history%20with%20machine%20learning.

[11] Pedamkar, P. (2022). *Machine Learning Algorithms*. EDUCBA. https://www.educba.com/machine-learning-algorithms/

[12] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2.ª ed.). O'Reilly Media. https://doi.org/10.5555/3378999

[13] IBM Cloud Education. (2022a). *Artificial Intelligence (AI)*. IBM. https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

[14] EUROCONTROL. (s. f.). *Concept of operations for European UTM systems – Extension for urban air mobility (CORUS-XUAM)*. https://www.eurocontrol.int/project/corus-xuam

[15] Prabhu, T. N. (2021). *Mastering the features of Google Colab*. Towards Data Science. https://towardsdatascience.com/mastering-the-features-of-google-colaboratory-92850e75701

[16] Databricks. (2021). *DataFrames*. https://www.databricks.com/glossary/what-are-dataframes

[17] JSON. (s. f.). *Introducing JSON*. https://www.json.org/json-en.html

[18] TechTarget Contributor. (2008). *Keyhole Markup Language (KML)*. TechTarget. https://www.techtarget.com/whatis/definition/Keyhole-Markup-Language-KML

# ANNEXES

```
#Instalation
!pip install basemap
!pip install basemap-data-hires
!pip install geopandas

######################################################################

#Importation
import pandas as pd
import json
import time
import os
from datetime import datetime
from string import Template
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mlt
from mpl_toolkits.basemap import Basemap
from google.colab import drive
import geopandas as gpd
from shapely import wkt
from datetime import time
from numpy.ma.core import transpose
import shutil

from sklearn import datasets
from sklearn import metrics
from sklearn.model_selection import train_test_split
from mlxtend.plotting import plot_decision_regions
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Ridge
from sklearn import linear_model
from sklearn.linear_model import ElasticNet

drive.mount('/content/drive')

######################################################################

#Delivery Ports
DELV_01=[1.9994768500328066,41.264602080919474]
DELV_02=[1.993994414806366,41.26440046653493]
DELV_03=[1.9887560606002805,41.26429159450844]
```

```
DELV_04=[1.9872057437896726,41.26429159450844]
DELV_05=[1.982431411743164,41.26438836965208]
DELV_06=[1.9807898998260498,41.26438030506223]
DELV_07=[1.9773888587951658,41.264396434240886]
DELV_08=[1.9746315479278564,41.26442869258624]
DELV_09=[1.9716274738311765,41.26441256341556]
DELV_10=[1.9666171073913574,41.26446901549553]
DELV_11=[1.9652545452117918,41.26445288633479]
DELV_12=[1.9600296020507812,41.264396434240886]

DP=np.array([DELV_01, DELV_02, DELV_03, DELV_04, DELV_05, DELV_06,
DELV_07, DELV_08, DELV_09, DELV_10, DELV_11, DELV_12])

######################################################################

#Definition of functions
#Trajectory function
def trajectory(callsign, df):
  found=False
  i = 0
  for i in df.index:
    if callsign==df['callsign'][i]:
      found=True
      dfi=pd.DataFrame(pd.json_normalize(df['geometry'][i]))
      FourDT=pd.DataFrame(dfi['coordinates'][0],columns=['lon','lat
','alt','secs'])
      return (FourDT)

#Trajectory function for day A
def trajectoryA(callsign, df):
  dfj = None
  i = 0
  for i in df.index:
    if callsign==df['callsign'][i]:
      dfj = pd.DataFrame(df['geometry.coordinates'][i],columns=['lo
n','lat','alt','secs'])
      return(dfj)

#Find delivery by latitude and longitude function
def find_delivery(lon, lat, DP):
  minlat = 10
  minlon = 10
  i = 0
  posicion1 = 0
  posicion2 = 0
  for i in range(len(DP)):
    clon = np.absolute(DP[i,0] - lon)
    clat = np.absolute(DP[i,1] - lat)
    if clon < minlon:
```

```
      minlon = clon
      posicion1 = i + 1
    if clat < minlat:
      minlat = clat
      posicion2 = i + 1
  if posicion1 == posicion2:
    return 'DELV_%02d'%(posicion1)
  else:
    if DP[posicion1 - 1,0] == DP[posicion2 - 1,0]:
      return 'DELV_%02d'%(posicion2)
    if DP[posicion1 - 1,1] == DP[posicion2 - 1,1]:
      return 'DELV_%02d'%(posicion1)
    else:
      if minlon < minlat:
        return 'DELV_%02d'%(posicion1)
      else:
        return 'DELV_%02d'%(posicion2)

#Find geometry from KML DataFrame
def geometry_kml(df):
  i = 0
  dffinal = pd.DataFrame()
  for i in df.index:
    dfr = df['str_geometry'][i].split()
    striuno = dfr[2]
    stridos = dfr[4]
    characters = "()"
    for x in range(len(characters)):
      striuno = striuno.replace(characters[x],"")
      stridos = stridos.replace(characters[x],"")
    nueva_fila = {'lon': striuno, 'lat': dfr[3], 'alt': stridos}
    dffinal = dffinal.append(nueva_fila, ignore_index=True)
  return (dffinal)


#Modify CSV DataFrame function
def retocarcsv (dfcsv,callsign):
  dfcsvfinal = pd.DataFrame()
  dfcsvfinal = dfcsv.drop_duplicates(subset=['datetime(utc)'], igno
re_index=True)
  dfcsvfinal['datetime(utc)'] = pd.to_datetime(dfcsvfinal['datetime
(utc)'], format="%Y-%m-%d %H:%M:%S")
  dfcsvfinal['time(millisecond)'] = (dfcsvfinal['time(millisecond)'
] - dfcsvfinal['time(millisecond)'].iloc[0])*0.001
  dfcsvfinal = dfcsvfinal.rename(columns={'time(millisecond)':'time
(secs)'})
  if callsign == 'UTAH':
    dfcsvfinal.rename(columns = {'height_above_takeoff(feet)':'heig
ht_above_takeoff(meters)'}, inplace = True)
```

```
    dfcsvfinal['height_above_takeoff(meters)'] = dfcsvfinal['height
_above_takeoff(meters)'] * 0.3048
  dfcsvfinal.rename(columns = {'height_above_takeoff(meters)':'alti
tude'}, inplace = True)
  found = False
  for i in dfcsvfinal.index:
    if dfcsvfinal['altitude'][i] >= 26 and found == False:
      found = True
      finalTO = i
  if found == False:
    print ('CONTINGENCIA')
    contingenica = pd.DataFrame()
    return (contingenica)
  if found == True:
    dfcsvfinal = dfcsvfinal.tail(len(dfcsvfinal)-finalTO)
  dfcsvfinal = dfcsvfinal.reset_index(inplace=False, drop=False)
  del dfcsvfinal['index']
  dfcsvfinal['Landing'] = [True if x <= 25 else False for x in dfcs
vfinal['altitude']]
  found = False
  for i in dfcsvfinal.index:
    if dfcsvfinal['Landing'][i] == True and i > ((len(dfcsvfinal)*0
.5) + (len(dfcsvfinal)*0.25)) and found == False:
      found = True
      landing = i
  if found == True:
    dfcsvfinal = dfcsvfinal.head(landing)
  return(dfcsvfinal)


#Modify CSV DataFrame function for SWORD operator
def retocarcsvSWORD (dfcsv,callsign):
  dfcsvfinal = dfcsv.drop_duplicates(subset=['time'], ignore_index=
True)
  for i in dfcsvfinal.index:
    split = dfcsvfinal['time'][i].split('+')
    dfcsvfinal['time'][i] = split[0]
  #dfcsvfinal['Vehiculo'] = vehiculo
  dfcsvfinal['time'] = pd.to_datetime(dfcsvfinal['time'], format="%
Y-%m-%d %H:%M:%S")
  dfcsvfinal['time(secs)'] = ""
  for i in dfcsvfinal.index:
    dfcsvfinal['time(secs)'][i] = (dfcsvfinal['time'][i].hour -
 dfcsvfinal['time'].iloc[0].hour)*3600 + (dfcsvfinal['time'][i].min
ute -
 dfcsvfinal['time'].iloc[0].minute)*60 + (dfcsvfinal['time'][i].sec
ond - dfcsvfinal['time'].iloc[0].second)
  found = False
  for i in dfcsvfinal.index:
```

```
    if dfcsvfinal['altitude'][i] >= 26 and found == False:
      found = True
      finalTO = i
  if found == False:
    print ('CONTINGENCIA')
    contingenica = pd.DataFrame()
    return (contingenica)
  if found == True:
    dfcsvfinal = dfcsvfinal.tail(len(dfcsvfinal)-finalTO)
  dfcsvfinal = dfcsvfinal.reset_index(inplace=False, drop=False)
  del dfcsvfinal['index']
  dfcsvfinal['Landing'] = [True if x <= 25 else False for x in dfcs
vfinal['altitude']]
  found = False
  for i in dfcsvfinal.index:
    if dfcsvfinal['Landing'][i] == True and i > ((len(dfcsvfinal)*0
.5)) and found == False:
      found = True
      landing = i
  if found == True:
    dfcsvfinal = dfcsvfinal.head(landing)
  return(dfcsvfinal)

#Read KML file function
def KML(kmlfile):
  dataframe = pd.DataFrame()
  kml = pd.DataFrame()
  filekml = pd.DataFrame()
  with open("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/KML/"+
 kmlfile + ".kml") as f:
    for linea in f:
      spl = linea.split('>')
      spl2 = spl[1].split('<')
      if len(spl2) > 1:
        nueva_fila = {'type': spl2[1], 'num': spl2[0]}
        dataframe = dataframe.append(nueva_fila, ignore_index=True)
  dataframe = dataframe.drop([0,1,2,3,4,5,6,7,8])
  for i in dataframe.index:
    if dataframe['type'][i] == '/name':
      nueva_fila = {'Name': dataframe['num'][i]}
      kml = kml.append(nueva_fila, ignore_index=True)
    if dataframe['type'][i] == '/mis:speed':
      nueva_fila = {'Speed': dataframe['num'][i]}
      kml = kml.append(nueva_fila, ignore_index=True)
    if dataframe['type'][i] == '/mis:cornerRadius':
      nueva_fila = {'CornerRadius': dataframe['num'][i]}
      kml = kml.append(nueva_fila, ignore_index=True)
    if dataframe['type'][i] == '/coordinates':
```

```
      coor = dataframe['num'][i].split(',')
      nueva_fila = {'Lon': coor[0],'Lat': coor[1],'Alt': coor[2]}
      kml = kml.append(nueva_fila, ignore_index=True)

  name = pd.DataFrame()
  speed = pd.DataFrame()
  cornerradius = pd.DataFrame()
  alt = pd.DataFrame()
  lat = pd.DataFrame()
  lon = pd.DataFrame()
  name = kml['Name'].dropna()
  name = name.reset_index(inplace=False, drop=False)
  speed = kml['Speed'].dropna()
  speed = speed.reset_index(inplace=False, drop=False)
  cornerradius = kml['CornerRadius'].dropna()
  cornerradius = cornerradius.reset_index(inplace=False, drop=False
)
  alt = kml['Alt'].dropna()
  alt = alt.reset_index(inplace=False, drop=False)
  lat = kml['Lat'].dropna()
  lat = lat.reset_index(inplace=False, drop=False)
  lon = kml['Lon'].dropna()
  lon = lon.reset_index(inplace=False, drop=False)
  kml = pd.concat([name, speed, cornerradius, lon, lat, alt], axis=
1)
  kml = kml[['Name','Speed','CornerRadius','Lon','Lat','Alt']]
  return(kml)


#Read JSON file function
def JSON(VERSION):
  filetemplate=Template('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/JSON/c
astelldefels.corusxuam.${version}.mitigated.traffic.${ext}')
  FILENAME=filetemplate.substitute(version=VERSION, ext="json")

  with open(FILENAME) as f:
    contents = json.loads(f.read())
  dfjson = pd.DataFrame.from_dict(pd.json_normalize(contents), orie
nt='columns')
  if VERSION == "v8.A.2":
    dfjson = dfjson[['callsign', 'request','geometry.coordinates']]
  else:
    dfjson = dfjson[['callsign', 'request','geometry']]

  dfjson['request'] = pd.to_datetime(dfjson['request'])
  return(dfjson)

#Map plot function
```

```
def mapa(traj, trajkml, trajreal, color, colorb, colorc):
  map = Basemap (llcrnrlon=traj['lon'].min()-
0.007, llcrnrlat=traj['lat'].min()-
0.009,urcrnrlon=traj['lon'].max()+0.007,urcrnrlat=traj['lat'].min()
+0.009,resolution='i',projection='tmerc',lat_0=41.3879,lon_0=2.1699
2)
  fig = plt.figure(figsize=(20,20))
  lon=[]
  lat=[]
  i = 0
  for i in traj.index:
    lon.append(traj['lon'][i])
    lat.append(traj['lat'][i])
  lonkml=[]
  latkml=[]
  i = 0
  for i in trajkml.index:
    lonkml.append(np.float64(trajkml['Lon'][i]))
    latkml.append(np.float64(trajkml['Lat'][i]))
  lonreal=[]
  latreal=[]
  j = 0
  for j in trajreal.index:
    lonreal.append(trajreal['longitude'][j])
    latreal.append(trajreal['latitude'][j])
  x1, y1 = map(lon,lat)
  x1kml, y1kml = map(lonkml,latkml)
  x1real, y1real = map(lonreal,latreal)
  xdelv, ydelv = map(DP[numdelv-1,0], DP[numdelv-1,1])
  plt.plot(x1kml,y1kml,color=colorc, label = "Flight Plan")
  plt.plot(x1,y1,color=color, label = "Scheduled Flight")
  plt.plot(x1real,y1real,color=colorb, label = "Real Flight")
  plt.plot(xdelv, ydelv, marker = "o", ms = 12, mec = 'grey', mfc =
 'grey')
  plt.annotate(delv, map(DP[numdelv-1,0], DP[numdelv-1,1]))
  plt.ylim(950,1220)
  plt.xlabel('Longitude')
  plt.ylabel('Latitude')
  plt.legend(fontsize=20, loc='upper right')

#Rename file in drive
def renamefile (file_oldname,file_newname_newfile):
  shutil.move(file_oldname, file_newname_newfile)
  print('renamed file')

######################################################################

#Day/File Version relation
dia = 'C'
```

```
if dia == 'A':
  VERSION="v8.A.2"
if dia == 'B':
  VERSION="v9.B.1"
if dia == 'C':
  VERSION="v9.C.1"
dfjson = JSON(VERSION)

#####################################################################
#Callsign Assignment
callsignComparar = 'C_OMAHA_1092'

#####################################################################

#Trajectory of a flight in JSON file
tra = pd.DataFrame()
if VERSION == "v8.A.2":
  tra = trajectoryA(callsignComparar,dfjson)
else:
  tra = trajectory(callsignComparar,dfjson)

alt_min = tra['alt'].idxmin()
ind = dfjson.loc[dfjson['callsign'] == callsignComparar]
TOT = ind['request']
delv = find_delivery(tra['lon'].iloc[alt_min], tra['lat'].iloc[alt_
min], DP)


#####################################################################

#Get the data needed and trajectory from KML
i = 0
for i in range(15):
  if i < 10:

    if os.path.isfile("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/KML/"+
 callsignComparar + "-DELV-0" + str(i) + ".kml"):
      kml = KML(callsignComparar + "-DELV-0" + str(i))
      kml['Alt'][len(kml)-1] = kml['Alt'][len(kml)-1].split(" ")[0]
  else:
    if os.path.isfile("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/KML/"+
 callsignComparar + "-DELV-" + str(i) + ".kml"):
      kml = KML(callsignComparar + "-DELV-" + str(i))
      kml['Alt'][len(kml)-1] = kml['Alt'][len(kml)-1].split(" ")[0]

#####################################################################

#Get the data needed from CSV
```

```
dfcsv = pd.read_csv("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
+ callsignComparar + "_" + delv +".csv")
spl1 = callsignComparar.split('_')
csvfinal = retocarcsv(dfcsv,spl1[1])

#####################################################################

#Trajectory of a flight in CSV file
if len(csvfinal) != 0:
  alt_min = csvfinal['altitude'].idxmin()
  TOT = csvfinal['datetime(utc)'].iloc[0]
  delv = find_delivery(csvfinal['longitude'].iloc[alt_min], csvfina
l['latitude'].iloc[alt_min],DP)
else:
  print('The maximum height of the flight is less than 30 meters')

#####################################################################

#Delivery port number calculation
d = delv.split("_")
numdelv = int(d[1])

#####################################################################

#Map plot
mapa(tra, kml, dfcsv, 'gold', 'purple', 'limegreen')

#####################################################################

#Vertical profile plot
fig = plt.figure(figsize=(20,20))
plt.plot(tra['secs'],tra['alt'],color='gold',label = "Scheduled Fli
ght")
plt.plot(tra['secs'],kml['Alt'],color='limegreen', label = "Flight
Plan")
plt.plot(dfcsv['time(millisecond)']/1000,dfcsv['height_above_takeof
f(meters)'],color='purple', label = "Real Flight")
plt.xlabel('Time (secs)')
plt.ylabel('Altitude (m)')
plt.legend(fontsize=15, loc='upper right')

#####################################################################

#JSON files to DataFrames
VERSION='v8.A.2'
filetemplate=Template('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/JSON/c
astelldefels.corusxuam.${version}.mitigated.traffic.${ext}')
FILENAME=filetemplate.substitute(version=VERSION, ext="json")
with open(FILENAME) as f:
```

```
  contents = json.loads(f.read())
dfjsonA = pd.DataFrame.from_dict(pd.json_normalize(contents), orien
t='columns')

VERSION='v9.B.1'
filetemplate=Template('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/JSON/c
astelldefels.corusxuam.${version}.mitigated.traffic.${ext}')
FILENAME=filetemplate.substitute(version=VERSION, ext="json")
with open(FILENAME) as f:
  contents = json.loads(f.read())
dfjsonB = pd.DataFrame.from_dict(pd.json_normalize(contents), orien
t='columns')

VERSION='v9.C.1'
filetemplate=Template('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/JSON/c
astelldefels.corusxuam.${version}.mitigated.traffic.${ext}')
FILENAME=filetemplate.substitute(version=VERSION, ext="json")
with open(FILENAME) as f:
  contents = json.loads(f.read())
dfjsonC = pd.DataFrame.from_dict(pd.json_normalize(contents), orien
t='columns')

dfjson = pd.concat([dfjsonA, dfjsonB, dfjsonC], axis=0,ignore_index
=True)

######################################################################

#Total flights in day A
vuelosAJUNO = 0
vuelosAOMAHA = 0
vuelosAUTAH = 0
vuelosASWORD = 0
for i in dfjsonA.index:
  spl1 = dfjsonA['callsign'][i].split('_')
  if spl1[1] == 'JUNO':
    vuelosAJUNO = vuelosAJUNO + 1
  if spl1[1] == 'OMAHA':
    vuelosAOMAHA = vuelosAOMAHA + 1
  if spl1[1] == 'UTAH':
    vuelosAUTAH = vuelosAUTAH + 1
  if spl1[1] == 'SWORD':
    vuelosASWORD = vuelosASWORD + 1

######################################################################

#Total flights in day B
vuelosBJUNO = 0
vuelosBOMAHA = 0
```

```
vuelosBUTAH = 0
vuelosBSWORD = 0
for i in dfjsonB.index:
  spl1 = dfjsonB['callsign'][i].split('_')
  if spl1[1] == 'JUNO':
    vuelosBJUNO = vuelosBJUNO + 1
  if spl1[1] == 'OMAHA':
    vuelosBOMAHA = vuelosBOMAHA + 1
  if spl1[1] == 'UTAH':
    vuelosBUTAH = vuelosBUTAH + 1
  if spl1[1] == 'SWORD':
    vuelosBSWORD = vuelosBSWORD + 1

######################################################################

#Total flights in day C
vuelosCJUNO = 0
vuelosCOMAHA = 0
vuelosCUTAH = 0
vuelosCSWORD = 0
for i in dfjsonC.index:
  spl1 = dfjsonC['callsign'][i].split('_')
  if spl1[1] == 'JUNO':
    vuelosCJUNO = vuelosCJUNO + 1
  if spl1[1] == 'OMAHA':
    vuelosCOMAHA = vuelosCOMAHA + 1
  if spl1[1] == 'UTAH':
    vuelosCUTAH = vuelosCUTAH + 1
  if spl1[1] == 'SWORD':
    vuelosCSWORD = vuelosCSWORD + 1

######################################################################

#Scheduled Flights Plot
index = ['Day A','Day B','Day C']
jsonA = pd.DataFrame({'JUNO': [vuelosAJUNO,vuelosBJUNO,vuelosCJUNO]
,'OMAHA': [vuelosAOMAHA,vuelosBOMAHA,vuelosCOMAHA],'UTAH': [vuelosA
UTAH,vuelosBUTAH,vuelosCUTAH],'SWORD': [vuelosASWORD,vuelosBSWORD,v
uelosCSWORD]}, index=index)
ax = jsonA.plot.bar(figsize=(10,10))
plt.ylim(0,90)
plt.legend(fontsize=15, loc='upper right')
plt.title('Scheduled Flights')

######################################################################

#Scheduled/Perfromed/Contingency flights of JUNO day C
dia='C'
operador ='JUNO'
vuelosCJUNO = 0
```

```
vuelosCJUNOcsv = 0
vuelosCJUNOcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosCJUNO=vuelosCJUNO + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosCJUNOcsv = vuelosCJUNOcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosCJUNOcont = vuelosCJUNOcont + 1

######################################################################

#Scheduled/Perfromed/Contingency flights of OMAHA day C
dia='C'
operador ='OMAHA'
vuelosCOMAHA = 0
vuelosCOMAHAcsv = 0
vuelosCOMAHAcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosCOMAHA=vuelosCOMAHA + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosCOMAHAcsv = vuelosCOMAHAcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosCOMAHAcont = vuelosCOMAHAcont + 1

######################################################################

#Scheduled/Perfromed/Contingency flights of UTAH day C
dia='C'
operador ='UTAH'
vuelosCUTAH = 0
vuelosCUTAHcsv = 0
vuelosCUTAHcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosCUTAH=vuelosCUTAH + 1
```

```
  if vuelos['callsignCSV'][i] is not None:
    vuelosCUTAHcsv = vuelosCUTAHcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosCUTAHcont = vuelosCUTAHcont + 1

#####################################################################

#Scheduled/Perfromed/Contingency flights of SWORD day C
dia='C'
operador ='SWORD'
vuelosCSWORD = 0
vuelosCSWORDcsv = 0
vuelosCSWORDcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosCSWORD=vuelosCSWORD + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosCSWORDcsv = vuelosCSWORDcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosCSWORDcont = vuelosCSWORDcont + 1

#####################################################################

#Scheduled/Perfromed/Contingency flights of JUNO day B
dia='B'
operador ='JUNO'
vuelosBJUNO = 0
vuelosBJUNOcsv = 0
vuelosBJUNOcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosBJUNO=vuelosBJUNO + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosBJUNOcsv = vuelosBJUNOcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosBJUNOcont = vuelosBJUNOcont + 1

#####################################################################

#Scheduled/Perfromed/Contingency flights of OMAHA day B
dia='B'
operador ='OMAHA'
```

```
vuelosBOMAHA = 0
vuelosBOMAHAcsv = 0
vuelosBOMAHAcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosBOMAHA=vuelosBOMAHA + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosBOMAHAcsv = vuelosBOMAHAcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosBOMAHAcont = vuelosBOMAHAcont + 1

######################################################################

#Scheduled/Perfromed/Contingency flights of UTAH day B
dia='B'
operador ='UTAH'
vuelosBUTAH = 0
vuelosBUTAHcsv = 0
vuelosBUTAHcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosBUTAH=vuelosBUTAH + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosBUTAHcsv = vuelosBUTAHcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosBUTAHcont = vuelosBUTAHcont + 1

######################################################################

#Scheduled/Perfromed/Contingency flights of SWORD day B
dia='B'
operador ='SWORD'
vuelosBSWORD = 0
vuelosBSWORDcsv = 0
vuelosBSWORDcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosBSWORD=vuelosBSWORD + 1
```

```
  if vuelos['callsignCSV'][i] is not None:
    vuelosBSWORDcsv = vuelosBSWORDcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosBSWORDcont = vuelosBSWORDcont + 1

#####################################################################

#Scheduled/Perfromed/Contingency flights of JUNO day A
dia='A'
operador ='JUNO'
vuelosAJUNO = 0
vuelosAJUNOcsv = 0
vuelosAJUNOcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosAJUNO=vuelosAJUNO + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosAJUNOcsv = vuelosAJUNOcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosAJUNOcont = vuelosAJUNOcont + 1

#####################################################################

#Scheduled/Perfromed/Contingency flights of OMAHA day A
dia='A'
operador ='OMAHA'
vuelosAOMAHA = 0
vuelosAOMAHAcsv = 0
vuelosAOMAHAcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosAOMAHA=vuelosAOMAHA + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosAOMAHAcsv = vuelosAOMAHAcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosAOMAHAcont = vuelosAOMAHAcont + 1

#####################################################################

#Scheduled/Perfromed/Contingency flights of UTAH day A
dia='A'
operador ='UTAH'
```

```
vuelosAUTAH = 0
vuelosAUTAHcsv = 0
vuelosAUTAHcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosAUTAH=vuelosAUTAH + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosAUTAHcsv = vuelosAUTAHcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosAUTAHcont = vuelosAUTAHcont + 1

######################################################################

#Scheduled/Perfromed/Contingency flights of SWORD day A
dia='A'
operador ='SWORD'
vuelosASWORD = 0
vuelosASWORDcsv = 0
vuelosASWORDcont = 0
vuelos =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_'+ dia + '_' + operador +'.txt', delimiter = ',')
vuelos = vuelos.replace({np.nan:None})
for i in vuelos.index:
  if vuelos['callsignJSON'][i] is not None:
    vuelosASWORD=vuelosASWORD + 1
  if vuelos['callsignCSV'][i] is not None:
    vuelosASWORDcsv = vuelosASWORDcsv + 1
  if vuelos['Contingencia'][i] is not None:
    vuelosASWORDcont = vuelosASWORDcont + 1

######################################################################

#All the performed flights
index = ['Day A','Day B','Day C']
jsonC = pd.DataFrame({'JUNO': [vuelosAJUNOcsv, vuelosBJUNOcsv, vuel
osCJUNOcsv],'OMAHA': [vuelosAOMAHAcsv, vuelosBOMAHAcsv, vuelosCOMAH
Acsv],'UTAH': [vuelosAUTAHcsv, vuelosBUTAHcsv, vuelosCUTAHcsv],'SWO
RD': [vuelosASWORDcsv, vuelosBSWORDcsv, vuelosCSWORDcsv]}, index=in
dex)
ax = jsonC.plot.bar()
plt.title("Performed Flights")

######################################################################

#OMAHA Flights
```

```
index = ['Day A','Day B','Day C']
jsonC = pd.DataFrame({'Scheduled Flights': [vuelosAOMAHA, vuelosBOM
AHA, vuelosCOMAHA],'Performed Flights': [vuelosAOMAHAcsv, vuelosBOM
AHAcsv, vuelosCOMAHAcsv],'Contingencies': [vuelosAOMAHAcont, vuelos
BOMAHAcont, vuelosCOMAHAcont]}, index=index)
ax = jsonC.plot.bar()
plt.title('OMAHA Flights')

######################################################################

#JUNO Flights
index = ['Day A','Day B','Day C']
jsonC = pd.DataFrame({'Scheduled Flights': [vuelosAJUNO, vuelosBJUN
O, vuelosCJUNO],'Performed Flights': [vuelosAJUNOcsv, vuelosBJUNOcs
v, vuelosCJUNOcsv],'Contingencies': [vuelosAJUNOcont, vuelosBJUNOco
nt, vuelosCJUNOcont]}, index=index)
ax = jsonC.plot.bar()

plt.title('JUNO Flights')

######################################################################

#UTAH Flights
index = ['Day A','Day B','Day C']
jsonC = pd.DataFrame({'Scheduled Flights': [vuelosAUTAH, vuelosBUTA
H, vuelosCUTAH],'Performed Flights': [vuelosAUTAHcsv, vuelosBUTAHcs
v, vuelosCUTAHcsv],'Contingencies': [vuelosAUTAHcont, vuelosBUTAHco
nt, vuelosCUTAHcont]}, index=index)
ax = jsonC.plot.bar()
plt.ylim(0,80)
plt.title('UTAH Flights')

######################################################################

#SWORD Flights
index = ['Day A','Day B','Day C']
jsonC = pd.DataFrame({'Scheduled Flights': [vuelosASWORD, vuelosBSW
ORD, vuelosCSWORD],'Performed Flights': [vuelosASWORDcsv, vuelosBSW
ORDcsv, vuelosCSWORDcsv],'Contingencies': [vuelosASWORDcont, vuelos
BSWORDcont, vuelosCSWORDcont]}, index=index)
ax = jsonC.plot.bar()
plt.ylim(0,80)
plt.title('SWORD Flights')

######################################################################

#Calculation Total Scheduled/Performed Flights
PvuelosC = vuelosCJUNO + vuelosCOMAHA + vuelosCUTAH + vuelosCSWORD
PvuelosB = vuelosBJUNO + vuelosBOMAHA + vuelosBUTAH + vuelosBSWORD
PvuelosA = vuelosAJUNO + vuelosAOMAHA + vuelosAUTAH + vuelosASWORD
```

```
RvuelosC = vuelosCJUNOcsv + vuelosCOMAHAcsv + vuelosCUTAHcsv + vuel
osCSWORDcsv
RvuelosB = vuelosBJUNOcsv + vuelosBOMAHAcsv + vuelosBUTAHcsv + vuel
osBSWORDcsv
RvuelosA = vuelosAJUNOcsv + vuelosAOMAHAcsv + vuelosAUTAHcsv + vuel
osASWORDcsv

######################################################################

#Scheduled/Performed Flights
index = ['Day A','Day B','Day C']
jsonC = pd.DataFrame({'Scheduled Flights': [PvuelosA, PvuelosB, Pvu
elosC],'Performed Flights': [RvuelosA, RvuelosB, RvuelosC]}, index=
index)
ax = jsonC.plot.bar()
plt.title('Scheduled/Performed Flights')

######################################################################

#List of Scheduled flights and Flights made
dia = 'C'
callsign = 'SWORD'

jsontotal = pd.DataFrame()
jsontraffic =  pd.read_table('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/JSON/'
+ dia + '_' + callsign +'_JSON.txt', delimiter = ',')
jsontraffic = jsontraffic.drop('Unnamed: 0', axis=1)
jsontotal = jsontraffic['callsign'] + '_' + jsontraffic['Take Off T
ime'] + '_' + jsontraffic['delivery']
jsontot = pd.DataFrame()
jsontot['callsignJSON'] = jsontotal

csvtotal = pd.DataFrame()
i = 0
for i in range(2000):
  for x in range(15):
    if x < 10:
      if os.path.isfile('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/'
 + dia + '_' + callsign + '_'+ str(i) + '_DELV_0' + str(x) +'.csv')
:
        nueva_fila = {'callsignCSV': dia + '_' +callsign + '_' + st
r(i) + '_' + 'DELV_0' + str(x)}
        csvtotal = csvtotal.append(nueva_fila, ignore_index=True)
    else:
      if os.path.isfile('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/'
 + dia + '_' + callsign + '_'+ str(i) + '_DELV_' + str(x) +'.csv'):
```

```
        nueva_fila = {'callsignCSV': dia + '_' +callsign + '_' + st
r(i) + '_' + 'DELV_' + str(x)}
        csvtotal = csvtotal.append(nueva_fila, ignore_index=True)


kmltotal = pd.DataFrame()
i = 0
for i in range(2000):
  for x in range(15):
    if x < 10:
      if os.path.isfile('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/KML/'
+ dia + '_' + callsign + '_'+ str(i) + '-DELV-0' + str(x) +'.kml'):
        nueva_fila = {'callsignKML': dia + '_' +callsign + '_' + st
r(i) + '_' + 'DELV-0' + str(x)}
        kmltotal = kmltotal.append(nueva_fila, ignore_index=True)
    else:
      if os.path.isfile('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/KML/'
+ dia + '_' + callsign + '_'+ str(i) + '-DELV-' + str(x) +'.kml'):
        nueva_fila = {'callsignKML': dia + '_' +callsign + '_' + st
r(i) + '_' + 'DELV-' + str(x)}
        kmltotal = kmltotal.append(nueva_fila, ignore_index=True)


comparacion = pd.concat([jsontot, kmltotal, csvtotal], axis=1)

comparacion.to_csv('Flights_'+ dia + '_' + callsign + '.txt', sep="
,")

#####################################################################

#Match Scheduled and Performed Flights and Rename CSV's
dia = 'C'
callsign = 'SWORD'
comparacion = pd.read_csv("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/TXTFLI
GHTS/Vuelos_"+ dia + '_' + callsign + '.txt')
i = 0
for i in comparacion.index:
  spl1 = comparacion['callsignJSON'][i].split('_')
  spl11 = spl1[3].split(':')
  hora1 = spl11[0]
  minute1 = int(spl11[1])
  j = 0
  for j in comparacion.index:
    spl2 = str(comparacion['callsignCSV'][j]).split('_')
    if spl2!=['nan']:
      spl3 = spl2[2].split(':')
```

```
      hora = int(spl3[0])
      minute = int(spl3[1])
      if hora < 9:
        if callsign == 'SWORD':
          horaf = '0'+str(hora)
        else:
          horaf = '0'+str(hora + 1)
      if hora == 9 and callsign == 'SWORD':
          horaf = '0'+str(hora)
      else:
        if callsign == 'SWORD':
          horaf = str(hora)
        else:
          horaf = str(hora + 1)
      if spl2[4] == spl1[5] and horaf == hora1:
        if minute1 <= minute and (minute-minute1) <= 3:
          print(spl1[2] + ' ' + spl1[3] + ' ' + spl1[5] + '/' + spl
2[2] + ' ' + spl2[4])
          if callsign == 'SWORD':
            if os.path.isfile("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "-HAR-" + callsign + "-" + str(j + 1) + ".csv"):
              csv = pd.read_csv("drive/MyDrive/TFG -
 Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "-HAR-" + callsign + "-" + str(j + 1) + ".csv")
              csvf = retocarcsvSWORD(csv,callsign)
              TOT = str(csvf['time'].iloc[0]).split(" ")
              time = TOT[1]
              if time == spl2[2]:
                print('a')
                renamefile("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "-HAR-" + callsign + "-
" + str(j + 1) + ".csv", "drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + spl1[0] + "_" + spl1[1] + "_" + spl1[2] + "_" + spl1[4] + "_" +
spl1[5] + ".csv")
            if os.path.isfile("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "-HEFO-" + callsign + "-" + str(j + 1) + ".csv"):
              csv = pd.read_csv("drive/MyDrive/TFG -
 Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "-HEFO-" + callsign + "-" + str(j + 1) + ".csv")
              csvf = retocarcsvSWORD(csv,callsign)
              TOT = str(csvf['time'].iloc[0]).split(" ")
              time = TOT[1]
              if time == spl2[2]:
```

```
                print('a')
                renamefile("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "-HEFO-" + callsign + "-
" + str(j + 1) + ".csv", "drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + spl1[0] + "_" + spl1[1] + "_" + spl1[2] + "_" + spl1[4] + "_" +
spl1[5] + ".csv")
          else:
            if os.path.isfile("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "_" + callsign + "_" + str(j + 1) + ".csv"):
              print('a')
              csv = pd.read_csv("drive/MyDrive/TFG -
 Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "_" + callsign + "_" + str(j + 1) + ".csv")
              csvf = retocarcsv(csv,callsign)
              if len(csvf) != 0:
                TOT = str(csvf['datetime(utc)'].iloc[0]).split(" ")
                time = TOT[1]
                if time == spl2[2]:
                  print('a')
                  renamefile("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + dia + "_" + callsign + "_" + str(j + 1) + ".csv", "drive/MyDrive
/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + spl1[0] + "_" + spl1[1] + "_" + spl1[2] + "_" + spl1[4] + "_" +
spl1[5] + ".csv")

######################################################################

#Creating JSON TXT
delvjson = pd.DataFrame()

for i in range(3000):
  tr = pd.DataFrame()
  dia = 'A'
  CS = 'UTAH'
  callsign = dia + '_' + CS + '_' + str(i)
  df = dfjson
  if dia == 'A':
    tr = trajectoryA(callsign,dfjson)
  else:
    tr = trajectory(callsign,dfjson)
  if tr is not None:
    alt_min = tr['alt'].idxmin()
```

```
    delv = find_delivery(tr['lon'].iloc[alt_min], tr['lat'].iloc[al
t_min], DP)
    dfc = dfjson.loc[dfjson['callsign'] == callsign]
    time = dfc['request'].iloc[0]
    hora = "{:02d}:{:02d}:{:02d}".format(time.hour, time.minute, ti
me.second)
    nueva_fila = {'callsign': callsign, 'Take Off Time': hora , 'de
livery': delv}
    delvjson = delvjson.append(nueva_fila, ignore_index=True)
with open('drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/JSON/J
SON-A-UTAH.txt','w') as fo:
  fo.write(delvjson.to_string(index=False))

delvjson.to_csv('A_UTAH_JSON.txt', sep=",")

######################################################################

#Prediction Table

Dia = ['A','B','C']
dfx = pd.DataFrame()
dfx2 = pd.DataFrame()
dfy = pd.DataFrame()
dfy2 = pd.DataFrame()
kmlcornerradiusdelv = pd.DataFrame()
for i in range(len(Dia)):
  dia = Dia[i]
  if dia == 'A':
    VERSION="v8.A.2"
  if dia == 'B':
    VERSION="v9.B.1"
  if dia == 'C':
    VERSION="v9.C.1"
  dfjson = JSON(VERSION)
  Callsign = ['JUNO','OMAHA','SWORD','UTAH']
  for j in range(len(Callsign)):
    callsign = Callsign[j]
    flights = pd.read_csv("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/FLIGHT
S/Flights_" + dia + "_" + callsign + ".txt")
    if 'callsignCSV' in flights:
      flightcsv = flights['callsignCSV'].dropna()
      for z in range(len(flightcsv)):
        spl = flightcsv[z].split('_')
        delv = DP[int(spl[4])-1]
        callsignComparar = str(spl[0]) + '_' + str(spl[1]) + '_' +
str(spl[2])
        if dia == 'A':
```

```
          tra = trajectoryA(callsignComparar,dfjson)
        else:
          tra = trajectory(callsignComparar,dfjson)
        for y in tra.index:
          if tra['lat'][y] == delv[1] and tra['lon'][y] == delv[0]:
            altdelv = tra['alt'][y]
            posaltdelv = y
          if altdelv != "":
            if y < posaltdelv:
              if tra['alt'][y] == tra['alt'][y+1]:
                altcreuer = tra['alt'][y]
                break
        if callsign == 'SWORD' and dia=='C':
          vel = 'Nan'
        else:
          kml = KML(spl[0] + '_' + spl[1] + '_'+ spl[2] + '-
' + spl[3] + '-' + spl[4])
          kml = kml.drop(kml.index[-1])
        num = 0
        speed = 0
        for k in kml.index:
          if kml['Alt'][k] == kml['Alt'][k+1] and k < posaltdelv:
            altcreukml = kml['Alt'][k]
            break
        for k in kml.index:
          if kml['Alt'][k] == altcreukml:
            num = num + 1
            speed = speed + int(kml['Speed'][k])
        speed = speed / num
        csv = pd.read_csv("drive/MyDrive/TFG - Laura Vazquez -
 PREDICTION OF 4D DRONE TRAJECTORIES FROM DEMONSTRATION DATA/CSVs/"
 + flightcsv[z] + ".csv")
        if callsign =='SWORD':
          csvf = retocarcsvSWORD(csv,callsign)
          TOF = csvf['time'].iloc[0]
          alt_min = csvf['altitude'].idxmin()
          TOD = csvf['time'].iloc[alt_min]
        else:
          csvf = retocarcsv(csv,callsign)
          TOF = csvf['datetime(utc)'].iloc[0]
          alt_min = csvf['altitude'].idxmin()
          TOD = csvf['datetime(utc)'].iloc[alt_min]
        TOFm = str(TOF).split(' ')
        TOFm = TOFm[1].split(':')
        TOFm = int(TOFm[0])*60 + int(TOFm[1])
        TODm = str(TOD).split(' ')
        TODm = TODm[1].split(':')
        TODm = int(TODm[0])*60 + int(TODm[1])
```

```
        nueva_fila = {'lat0': tra['lat'][0], 'lon0': tra['lon'][0],
'alt0':tra['alt'][0], 'cornerRadius0':kml['CornerRadius'][0], 'latd
elv': delv[1], 'londelv': delv[0], 'altdelv':altdelv,  'altcreuer':
 altcreuer, 'velcreuer': speed, 'Time Entering Airway (min)': TOFm}

        dfx = dfx.append(nueva_fila, ignore_index=True)
        nueva_filay = {'Time of Delivery (min)': TODm}
        dfy = dfy.append(nueva_filay, ignore_index=True)

######################################################################

#Prediction with Linear Regression Model
X_train, X_test, y_train, y_test = train_test_split(dfx, dfy, test_
size=0.2)

regression_model=LinearRegression()
regresion = regression_model.fit(X_train,y_train)

y_pred = regresion.predict(X_test)
print('MAE',mean_absolute_error(y_test, y_pred))
print('SQMES',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('score', regresion.score(X_train,y_train))

######################################################################

#Prediction with Logistic Regression Model
X_train, X_test, y_train, y_test = train_test_split(dfx, dfy, test_
size=0.2)
LRG = LogisticRegression()
LGRRegression = LRG.fit(X_train,y_train)
y_pred = LGRRegression.predict(X_test)

print('MAE',mean_absolute_error(y_test, y_pred))
print('MSE',mean_squared_error(y_test, y_pred))
print('SQMES',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('score', LGRRegression.score(X_train,y_train))

######################################################################

#Prediction with Ridge Regression Model
X_train, X_test, y_train, y_test = train_test_split(dfx, dfy, test_
size=0.2)

model = Ridge(alpha = 0.5, normalize = False, tol = 0.001, solver =
'auto', random_state = 42)
modelr = model.fit(X_train, y_train)
y_pred = model.predict(X_test)


print('MAE',mean_absolute_error(y_test, y_pred))
```

```
print('MSE',mean_squared_error(y_test, y_pred))
print('SQMES',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('score', modelr.score(X_train,y_train))

######################################################################

#Prediction with Bayesian Regression Model
X_train, X_test, y_train, y_test = train_test_split(dfx, dfy, test_
size=0.2)

BayReg = linear_model.BayesianRidge()
BRModel = BayReg.fit(X_train, y_train)



y_pred = BayReg.predict(X_test)
print('MAE',mean_absolute_error(y_test, y_pred))
print('SQMES',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('score', BRModel.score(X_train,y_train))

score = BayReg.score(X_test, y_test)
print("\n\nModel score : ", score)

######################################################################

#Prediction with Elastic Net Regression Model
from sklearn.linear_model import ElasticNet
X_train, X_test, y_train, y_test = train_test_split(dfx, dfy, test_
size=0.2)

EN = ElasticNet()
ENModel = EN.fit(X_train, y_train)

y_pred = EN.predict(X_test)
print('MAE',mean_absolute_error(y_test, y_pred))
print('SQMES',np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('score', ENModel.score(X_train,y_train))
```