

# ESCUELA INTERNACIONAL DE POSTGRADOS

**TRABAJO FINAL DE MÁSTER: Aviation Accident Prediction (AAP)**

**TITULACION:** Máster de programación Avanzada en Python para Big Data, Hacking y Machine Learning

**ALUMNOS:**

- Jose Cabezas Pulgarín
- Pablo Ruiz Veredía
- Victor Simo Lozano. 53879955V

**FECHA DE ENTREGA:** septiembre 2023

Agradecimientos

## Contenido

Abstract .....	2
Introducción .....	3
1    Objetivos generales del trabajo .....	3
2    Justifica la elección del tema. ....	3
3    Resumen metodología.....	4
3.1    Procedimientos.....	4
3.2    Agile Project Management (APM) .....	4
3.3    Control de versiones: Git.....	5
4    Estructura.....	6
<b>Planteamiento del problema</b> .....	<b>7</b>
Objetivos del trabajo.....	7
Metodología .....	8
1    Preprocesamiento del Dataset.....	8
1.1    Dataset global.....	8
1.2    Forecast.....	13
1.3    Crew.....	34
<b>1.4    Airplane(PABLO)</b> .....	<b>54</b>
2    Generar el modelo .....	54
2.1    Forecast.....	54
2.2    Crew.....	60
<b>2.3    Airplane(PABLO)</b> .....	<b>67</b>
3    GUI: Streamlit.....	67
3.1    Directorios de la aplicación Streamlit .....	67
3.2    StreamlitCloud.....	80
<b>4    Unión de las partes y prueba en local</b> .....	<b>81</b>
<b>Evaluación de los resultados</b> .....	<b>82</b>
Conclusiones.....	83
Referencias.....	84
<b>Anexos</b> .....	<b>85</b>

## Abstract

This master's thesis focuses on the development of a Python-based system for aviation accident prediction (AAP). To achieve this, PyCaret library has been used to obtain a Machine Learning Model with a pre-processed aircraft dataset. In addition, Streamlit has been used to create a GUI able to introduce the model values and get the incident prediction.

To get the result, an exhausted data preprocessing with the aircraft accident dataset has been necessary. This includes tasks such as data cleaning, outlier removal, and variable transformation. Through this process, is obtained a refined dataset that can be used in the predictive models.

PyCaret library is used to develop a machine learning model capable of predict the occurrence of aircraft accidents. PyCaret allows exploring different classification algorithms and determining the best fit for a dataset. In addition, it greatly simplifies the process of model training and evaluation.

Finally, Streamlit library is used to create an interactive user interface (GUI), which allows create a form where users can enter relevant data for forecasting, including date, aircraft type and airline, etc. Streamlit makes it easy to create the GUI, enabling rapid development and deployment of interactive web applications.

In summary, this thesis focuses on the development of a Python-based aircraft accident prediction system that uses Streamlit to create user input forms and PyCaret to train and evaluate predictive models. This work includes preprocessing the aircraft accident data set, choosing the most appropriate classification algorithm and creating a user interface. With this system, we aim to contribute to improving the safety of the aviation industry and prevent potential accidents.

**KEYWORDS:** AAP, Accident, Aviation, BigData, Machine Learning, PyCaret, Python, Streamlit

## Introducción

### 1 Objetivos generales del trabajo

El objetivo general del proyecto de Fin de Máster (TFM) es desarrollar un sistema de predicción de accidentes aéreos utilizando programación Python y Machine Learning. Haciendo uso de las bibliotecas Streamlit y PyCaret junto con un conjunto BigData de accidentes aéreos preprocesados.

El objetivo principal es ayudar a mejorar la seguridad en la industria de la aviación proporcionando herramientas para predecir accidentes de aeronaves. La capacidad de predecir posibles accidentes puede facilitar la toma de decisiones y la implementación de medidas preventivas por parte de las autoridades, las aerolíneas y otros actores de la industria. Además, se fija el objetivo de desarrollar un sistema eficiente y fácil de usar que utilice la programación Python y las bibliotecas Streamlit y PyCaret.

Además, se proporciona una interfaz intuitiva que permita a los usuarios ingresar datos relevantes y predecir con precisión la probabilidad de un accidente aéreo.

En resumen, este trabajo se centra en el desarrollo de sistemas de predicción de accidentes de aviación para mejorar la seguridad en la industria aeronáutica. Utilizando la programación de Python, las bibliotecas Streamlit y PyCaret y los conjuntos de datos preprocesados, ofreciendo así una herramienta eficiente y fácil de usar para predecir la probabilidad de accidentes aéreos.

### 2 Justifica la elección del tema.

La elección de los temas del TFM se basa en varias razones importantes. La seguridad en la industria de la aviación es un tema muy importante para proteger la vida humana y la economía mundial. Los accidentes de aeronaves pueden tener consecuencias devastadoras, que incluyen la pérdida de vidas, daños a la propiedad y consecuencias legales y financieras.

Por lo tanto, el desarrollo de un sistema de predicción de accidentes de aviación es una medida importante para prevenir y reducir los riesgos asociados y mejorar la seguridad en esta importante área. Además, los avances en el procesamiento de datos y los métodos de aprendizaje automático brindan nuevas oportunidades para resolver problemas complejos, como la predicción de accidentes de aeronaves. Usando las capacidades de programación de Python y bibliotecas especializadas como Streamlit y PyCaret, se puede desarrollar sistemas eficientes y precisos.

La razón por la que se opta por utilizar conjuntos de datos de accidentes de aeronaves recopilados previamente es también la disponibilidad de información valiosa y la reducción de costos y tiempo asociados con la recopilación de nuevos datos.

Al analizar los patrones y los factores de riesgo en estos datos históricos, se puede crear modelos predictivos confiables que ayuden a comprender mejor los accidentes de aeronaves e implementar medidas preventivas más efectivas. Es por ello que el desarrollo de un sistema de predicción de accidentes de aeronaves es una herramienta invaluable para los tomadores de decisiones de la industria de la aviación, como los reguladores, las aerolíneas y los investigadores de seguridad.

En resumen, al proporcionar información precisa sobre la probabilidad de incidentes, se puede implementar estrategias de seguridad más efectivas y se puede tomar medidas preventivas proactivas. Es por ello que, la selección del tema para el TFM se basa en la **importancia de mejorar la seguridad en la industria de la aviación**, los avances tecnológicos en aprendizaje automático, el uso de datos existentes y la contribución a la toma de decisiones en esta área.

### 3 Resumen metodología

Para conseguir el resultado final del TFM, además de realizar un trabajo estructurado entre todos los integrantes del grupo de trabajo, se hace uso de diferentes herramientas que facilitan este trabajo, como es el uso de un **control de versiones** basado en tecnología Git y un método de trabajo **Agile** con una metodología Scrum.

#### 3.1 Procedimientos

El TFM se ha realizado por un total de tres integrantes, utilizando un enfoque colaborativo para desarrollar un sistema de predicción de accidentes de aviación. Cada miembro del equipo se enfoca en construir un modelo predictivo utilizando un grupo diferente de columnas de datos. Donde cada integrante se hace responsable de los datos de la tripulación, otro de los datos de vuelo y un tercero de la meteorología.

Primero, cada miembro del equipo crea individualmente su propio modelo de predicción. Usando grupos específicos de columnas, algoritmos y técnicas de aprendizaje automático para entrenar un modelo capaz de predecir la probabilidad de un accidente aéreo.

Luego se crea la interfaz de usuario para cada uno de estos modelos, ordenado en páginas de Streamlit, con la que cada miembro del equipo diseña un formulario para ingresar los datos de su modelo.

Adicionalmente, se crea una página final para combinar y analizar los resultados obtenidos de cada modelo. En esta, se aúna los resultados de cada algoritmo y quedan representados globalmente. Donde se evalúa la posibilidad de un accidente se tiene en cuenta por el número de accidentes obtenidos en los modelos individuales.

#### 3.2 Agile Project Management (APM)

Debido a que el desarrollo del TFM ha sido realizado por un total de 3 integrantes, la necesidad de poder organizar las tareas a desarrollar, así como la organización de las mismas, es algo relevante.

Gracias a la integración de una metodología Agile en un grupo de desarrolladores, se consigue desarrollar el producto con mayor velocidad y colaboración. “Software teams that embrace agile project management methodologies increase their development speed, expand collaboration, and foster the ability to better respond to market trends.” (DRUMOND, s.f.)

De las metodologías Agile existentes, Kanban y Scrum, la escogida para el desarrollo del TFM, ha sido **Scrum**. El principal motivo de esta selección, se debe al flujo de trabajo, en el que se plantea una serie de sprints, o etapas de trabajo, que se planifican previamente para decidir las tareas que se acometerá por cada miembro en el tiempo estipulado de sprint y que serán revisadas en las reuniones periódicas programadas para el control de avances.

Para este proyecto, la duración de los sprints, se plantea en 3 semanas, añadiendo dos reuniones semanales de no más de 15 minutos para ir controlando el avance de las tareas, así como los

problemas que surgen durante el desarrollo. Cabe mencionar que, estas reuniones han sido llevadas a cabo únicamente por los integrantes del grupo de desarrollo.

En cuanto al planteamiento de las tareas, estas han sido englobadas en una serie de *Epics*, es decir, en una serie de hitos a conseguir para dar por finalizado el proyecto. Los *Epics* planteados en el trabajo son:

- **Datos PHA:** Consiste en la obtención y tratamiento de los datos extraídos de la fuente para el desarrollo de los modelos.
- **Modelo:** Distintos modelos de aprendizaje automático desarrollado por cada uno de los integrantes del grupo de trabajo.
  - **Airplane:** Pablo Ruiz Veredia.
  - **Crew:** Jose Cabezas Pulgarín.
  - **Forecast:** Victor Simó Lozano.
- **Streamlit:** Representación de los datos y uso de GUI con la librería Streamlit de Python.
- **Documentación:** Desarrollo de la memoria del proyecto.

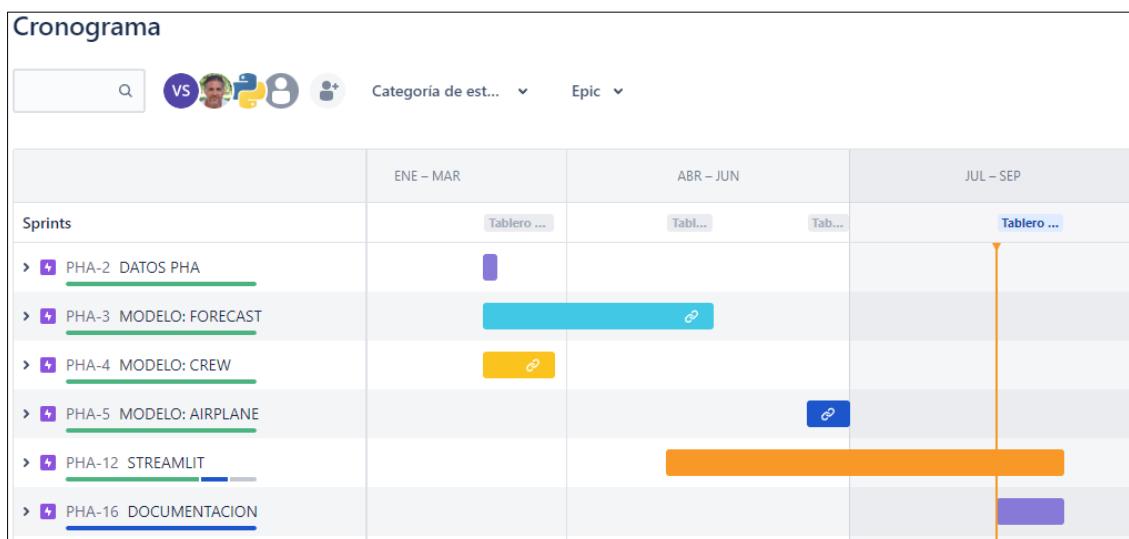


Ilustración 1 Cronograma Epics

### 3.3 Control de versiones: Git

Hoy en día, no se entiende un desarrollo de Software sin un control de versiones. En este proyecto, se ha utilizado el sistema Git con el software **GitHub**.

En este caso, se crea un repositorio público cuyo enlace es [Vic-silo/TFM-EIP: Master Thesis: Aviation Accident Prediction \(AAP\) \(github.com\)](https://github.com/Vic-silo/TFM-EIP-Master-Thesis-Aviation-Accident-Prediction-AAP).

En el repositorio, se hace uso de las ramas para seguir una integración continua (CI) dentro del ámbito **DevOps**.

Existe como rama principal **main\_new**, en la que se despliega, mediante el uso de PullRequest aprobadas por el resto de integrantes del equipo, los cambios realizados como parte del desarrollo en las ramas **feat/**. Además, existe una rama secundaria **feat/streamlit** que hace las veces de rama de desarrollo, es decir, cualquier cambio que exista en una rama **feat**, antes de ser desplegado en **main\_new**, será desplegado y testado en esta rama de desarrollo. De este modo, se asegura el correcto funcionamiento de los cambios realizados.

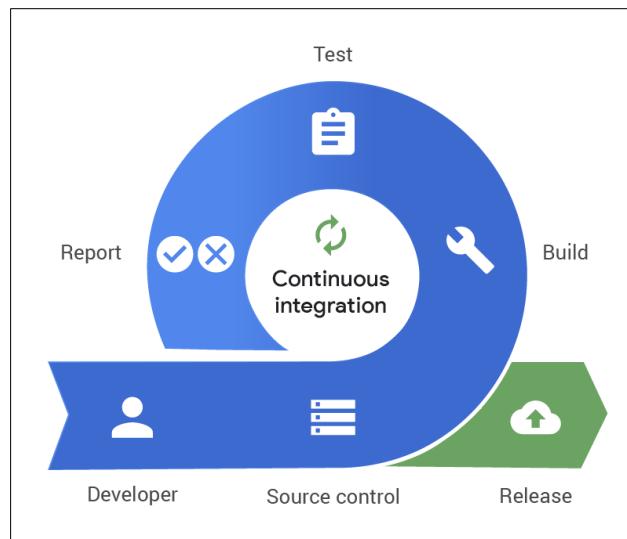


Ilustración 2 CI flow

#### 4 Estructura

La estructura que se utiliza en el TFM se basa en un enfoque colaborativo de tres personas, detallada a continuación:

- **Distribución de la responsabilidad**

Cada miembro del equipo asume la responsabilidad de una parte específica del proyecto. Uno de ellos se centra en los datos de la tripulación, el segundo en los datos de vuelo y el tercero en los datos meteorológicos. Esta división permite un enfoque más especializado y eficiente de cada área.

- **Construcción de modelos individuales**

Cada miembro del equipo trabaja individualmente para construir modelos predictivos utilizando grupos específicos de columnas. Se usa algoritmos y técnicas de aprendizaje automático para entrenar y adaptar estos modelos para predecir la probabilidad de un accidente aéreo.

- **Desarrollo de páginas individuales en Streamlit**

Se usa la biblioteca Streamlit de Python para crear páginas individuales donde cada miembro del equipo diseña un formulario para ingresar datos específicos para su modelo. En estas páginas, los usuarios pueden ingresar información relacionada con la tripulación de vuelo, la aeronave y el clima, respectivamente.

- **Página final para análisis global**

Además de las páginas individuales, se crea una página final en Streamlit donde se recopila los resultados de cada modelo para su análisis. En esta página, se explica los resultados de cada algoritmo y se brinda una visión global de la probabilidad de accidentes aéreos, teniendo en cuenta la información de todos los modelos.

Con este enfoque colaborativo, se permite aprovechar la experiencia personal y el conocimiento de cada miembro del equipo sobre diferentes aspectos del problema de predicción de

accidentes de aviación. Donde juntos, se logra desarrollar un sistema más completo y sólido que cubra aspectos clave como la tripulación, la aeronave y los datos meteorológicos.

## Planteamiento del problema

En la formulación del problema, existe el desafío de mejorar la seguridad en la industria de la aviación mediante la predicción de accidentes de aeronaves. Los accidentes de aeronaves tendrán importantes consecuencias, como pérdidas, daños materiales y consecuencias financieras. Por lo tanto, es muy importante desarrollar herramientas para predecir la ocurrencia de estos eventos y tomar medidas preventivas para evitarlos.

A pesar de las numerosas medidas y protocolos de seguridad vigentes en la industria de la aviación, los accidentes aún ocurren y es necesario abordar el problema de manera proactiva. Actualmente, las decisiones de gestión de seguridad de la aviación se basan en datos históricos y análisis manual, que pueden ser limitantes y no brindan una imagen completa del riesgo.

El objetivo principal de nuestro TFM es desarrollar un sistema de predicción de accidentes aéreos utilizando técnicas de aprendizaje automático y datos relevantes. La idea es aprovechar la disponibilidad de conjuntos de datos de accidentes de aviones recopilados previamente y utilizar algoritmos de aprendizaje automático para crear modelos predictivos.

El modelo nos permite estimar la probabilidad de un accidente aéreo en función de varios factores, como la información de la tripulación, los datos de la aeronave y las condiciones meteorológicas. Anticipar la probabilidad de un accidente puede conducir a medidas proactivas para mejorar la seguridad, como ajustar los procedimientos, realizar cambios en el diseño de la aeronave o mejorar la capacitación y educación de la tripulación de vuelo.

Nuestra declaración del problema TFM se centra en la necesidad de mejorar la seguridad en la industria de la aviación mediante la predicción de accidentes de aeronaves. Utilizando técnicas de aprendizaje automático y datos relacionados, nuestro objetivo es desarrollar un sistema predictivo que proporcione información valiosa para la toma de decisiones y medidas preventivas, ayudando así a reducir el riesgo y garantizar la seguridad en la industria de la aviación.

## Objetivos del trabajo

El objetivo general de nuestro trabajo es desarrollar un sistema de predicción de accidentes aéreos utilizando técnicas de aprendizaje automático y datos relacionados. El objetivo de la iniciativa es mejorar la seguridad en la industria de la aviación y prevenir activamente los accidentes y sus consecuencias. Para lograr este objetivo general, formulamos una serie de objetivos secundarios que se derivan del objetivo principal y justifican y permiten que se desarrollen.

En primer lugar, decidimos realizar un análisis exploratorio exhaustivo de los datos existentes sobre accidentes de aeronaves. A través de este análisis, tratamos de comprender la estructura de los datos, identificar patrones importantes, detectar valores atípicos e identificar posibles correlaciones entre variables.

El segundo objetivo se centra en el preprocesamiento de datos. Aquí, nos enfocamos en realizar tareas de limpieza, manejo de valores faltantes, estandarización de variables y transformaciones

necesarias para garantizar la calidad y consistencia de los datos utilizados en los modelos predictivos.

El tercer objetivo es que cada miembro del equipo tenga la tarea de crear un modelo de predicción individual utilizando un grupo específico de columnas de datos. Estos paquetes incluyen información de la tripulación, datos de vuelo y datos meteorológicos. Para hacer esto, usamos algoritmos de aprendizaje automático para entrenar y adaptar un modelo en un intento de predecir la probabilidad de un accidente aéreo individual.

El siguiente objetivo es implementar la interfaz de usuario utilizando la biblioteca Streamlit. La interfaz está diseñada para ser intuitiva y fácil de usar, lo que permite a los usuarios ingresar datos relevantes en cada modelo de predicción creado por los miembros del equipo. Esta implementación facilita la interacción con el sistema y aumenta su usabilidad.

Finalmente, el objetivo principal es integrar los resultados de los modelos individuales y realizar un análisis global. Este análisis nos permite considerar combinaciones de resultados individuales para explicar conjuntamente la probabilidad de un accidente aéreo. Este enfoque le permite obtener una imagen más precisa y confiable de los posibles accidentes. En resumen, nuestro trabajo se basa en el desarrollo de un sistema de predicción de accidentes aeronáuticos, donde los objetivos secundarios son realizar análisis exploratorios, preprocessar los datos, crear modelos individuales, implementar una interfaz de usuario y realizar un análisis global de los resultados buscando complementarse entre sí para lograr mejores objetivos generales de seguridad en la industria de la aviación.

## Metodología

### 1 Preprocesamiento del Dataset

#### 1.1 Dataset global

Los modelos de predicción de accidentes que se desarrolla en el proyecto, parten de un conjunto de datos común obtenido de (U.S. Department of Transportation, Federal Aviation Administration, s.f.).

En este repositorio, se puede encontrar los datos en formato *txt* desde el año 1975 hasta el presente. Para hacer uso de los datos agrupados todos en un único conjunto de datos, se realiza un código en un Jupyter Notebook que permita realizar dicha operación para el uso común de los datos que se pretende.

Primeramente, se realiza un web scraping con el que se es capaz de acceder a los ficheros *txt* de la fuente. En este primer paso se almacena en una lista, todos aquellos ficheros que se ha encontrado disponibles, para finalmente recorrer la lista y descargarlos en nuestro directorio local accediendo directamente a su url.

```
● ● ● script 1: web scraping

# URL de los datos a analizar: Leyenda
url_legend = 'https://av-info.faa.gov/data/AID/Afilelayout.txt'

# URL de los datos a analizar: Lustros con datos de estudio de 1975 a actualidad
# Obtener ficheros que se desea obtener los datos
lust = [year for year in range(1975, 2025, 5)]
AID_files = [f'a{year}_{str(year+4)[-2:]}.txt' for year in lust]
# El fichero de datos actual, sabemos que es de 2020 a 2025 actualmente
AID_files[-1:] = 'a2020_25.txt',

with urlopen("https://av-info.faa.gov/dd_sublevel.asp?Folder=%5CAID") as r:
    bs = BeautifulSoup(r.read(), "html.parser")

url_AID = []
for link in bs.find_all("a"):
    # Buscar el texto del fichero en los ficheros creados previamente
    # Se desea aquellos que estan delimitados por \t, por ello buscar /tab/
    if link.next_element in AID_files and '/tab/' in link.get("href"):
        url_AID.append(link.get("href"))
```

Script 1 Web scraping – Datasets ficheros

```
● ● ● Web scraping - Datasets

# Descarga de ficheros previamente encontrados con BeautifulSoup
for url, file_name in zip(url_AID, AID_files):
    # Se envia la petición HTTP Get para la obtención de los datos
    data = requests.get(url)

    # Guardamos el archivo de manera local
    with open(data_path+file_name, 'wb')as file:
        file.write(data.content)
        print(f'[+] FILE_CREATED\t{file_name}')
```

Script 2 Web scraping - Datasets descarga

Del mismo modo que con los ficheros de datos, se realiza web scraping para la leyenda de los datos, almacenada en *txt* en la fuente de datos. Una vez se obtenga esta, se guarda en el directorio local en formato *csv* para poder trabajar con ella más adelante. Esta leyenda nos permite conocer el título de las columnas, que como se verá más adelante, en la fuente de los datos se define de forma abreviada por c + número identificativo.

```
● ● ● Web scraping - Leyenda

# Leemos el contenido de la leyenda
with urlopen(url_legend) as content:
    soup = BeautifulSoup(content, "html.parser")
    soup_lines = str(soup).split('\r\n')

# Transformamos la respuesta en un diccionario con el nombre de la columna y la descripción
legend_df = {"Column_name": [], "Description": []}
# Recorrer las líneas con datos de líneas. Se salta las dos primeras y las tres últimas
# filas por no tener datos relevantes de la leyenda
for line in soup_lines[2:-3]:
    # Se extrae los 5 primeros caracteres para conocer el nombre de la columna
    legend_df["Column_name"].append(line[0:5].strip())
    # Se extrae la descripción, esta comienza en la posición 53
    legend_df["Description"].append(line[53:].strip())

# Convertir el diccionario en pandas.DataFrame
legend_df = pd.DataFrame.from_dict(data=legend_df)
legend_df
```

*Script 3 Web scraping – Leyenda*

Una vez se ha obtenido todos los ficheros, se realiza los pasos para poder crear el conjunto de todos los datos en un único Dataset. Mediante un análisis previo de los datos obtenidos, se sabe que el número de columnas existente es de 180, dado que hay un gran número de columnas, se realiza una función que valide la creación del Dataset global. En esta función **validate\_df** se realiza las siguientes comprobaciones:

- Longitud del fichero es de 180 columnas.
- Las columnas se encuentran en la leyenda.

Con estas validaciones, en el caso de no ser cumplidas, se devuelve un error y no continua la creación del Dataframe global hasta sanar los errores.

```
● ● ● Data validation

def validate_df(df) -> Union[bool, list]:
    """
    Validar las columnas existentes en el dataframe creado
    """
    if len(df.columns) != 180:
        err = f'[-] COLUMNS_NUMBER\t{len(df.columns)}'
        return [False, err]

    for column in df.columns:
        # Saltar la columna "end_of_record"
        if column == 'end_of_record':
            continue
        # Acceder al registro de la leyenda y comprobar que hay resultados
        if legend_df.loc[legend_df['Column_name'] == column].empty:
            err = f'[-] COLUMN_UNMATCH\t{column}'
            return [False, err]

    return True
```

*Script 4 Validación de Dataframe*

Una vez se tiene el conjunto de datos global, como es de esperar en cualquier proceso de recogida de datos, existe valores que no son los deseados o valores nulos. Es por ello que previo a dar por finalizado el Dataframe global, se realiza una serie de comprobaciones y limpieza de datos.

Para la limpieza de los datos, se hace uso de una función creada ***clean\_data***, con la que poder imputar como valores ***Nan*** aquellos nulos o vacíos. De este modo, teniendo como ***Nan*** todos los valores indeseados, se puede realizar posteriormente una limpieza de las columnas que contenga todos sus datos nulos. Como resultado de esta operación, se elimina la columna ***end\_of\_record*** quedando, por lo tanto, un Dataframe global de 179 atributos y 214887 registros. Un total de más de 38 millones de datos.

```
● ● ● Data cleaning

logging.basicConfig(filename='nan-convert.log', level=logging.DEBUG)

def clean_data(value):
    # Si se trata de un valor en blanco sustituir a NaN
    try:
        if isinstance(value, str) and (value.isspace() or value == ''):
            return np.nan

    return value

except Exception as e:
    logging.info(f'[-] ERROR\t{e.__str__()}\\tcol:{column}\\trow:{i}\\tvalue:{value}')
    return value

# Iterar sobre todas las columnas y comprobar sus valores
columns = len(df_clean.columns)

for column in df_clean.columns:
    # Utilizar compresión de listas y aplicar el filtrado de valores
    values = [clean_data(value) for value in df_clean[column]]
    df_clean[column] = values
    columns -= 1
    print(f'[+] REMAINING_COLUMNS\\t{columns}\\t', end='\r')

# Eliminar las columnas que tenga todos los datos nulos
empty_cols = [col for col in df_clean.columns if df_clean[col].isna().all()]

df_clean.drop(empty_cols, axis=1, inplace=True)
display(df_clean)
print(f'[+] DROPPED_COLUMNS\\t{empty_cols}'')
```

Script 5 Dataframe global – Limpieza

Dado el elevado número de datos, se realiza a continuación, un cribado de todos los datos, de modo que se pueda reducir la dimensión del conjunto de datos. En este cribado de los datos, se tiene en cuenta aquellas columnas que no van a aportar ninguna información útil a ninguno de los modelos a desarrollar o su información no es analizable.

Tras el proceso de eliminación de columnas, un total de 30 columnas son eliminadas, se queda una dimensión de 149 atributos, lo que hace un total de 32 millones de datos, reduciéndose en un 15% el conjunto de datos.

[+]	UNUSED_COL	c5	Unique control number used to relate to AID_MAIN table.
[+]	UNUSED_COL	c3	Form on which the latest data was received.
[+]	UNUSED_COL	c2	FAR part number
[+]	UNUSED_COL	c4	Agency conducting investigation.
[+]	UNUSED_COL	c9	Date the accident/incident happened.
[+]	UNUSED_COL	c75	First, second, or third airplane involved or Not a midair.
[+]	UNUSED_COL	c140	Extent of investigation
[+]	UNUSED_COL	c139	Code for related reports pertaining to the accident/incident.
[+]	UNUSED_COL	c203	This field is generated by the system so no edits are performed.
[+]	UNUSED_COL	c204	Sequential number assigned to cases dealt with in the current year.
[+]	UNUSED_COL	c214	Group of the air operator involved in the investigation.
[+]	UNUSED_COL	c790	Consolidated Statement Rebuttal
[+]	UNUSED_COL	c26	Technical certificate data sheet information of the aircraft
[+]	UNUSED_COL	c37	Technical certificate data sheet information of the engine
[+]	UNUSED_COL	c15	Name of the airport
[+]	UNUSED_COL	c16	Airport runway number
[+]	UNUSED_COL	c17	Location identification code of the nearest airport, if not on airport.
[+]	UNUSED_COL	c18	Direction from the airport location identifier.
[+]	UNUSED_COL	c19	Distance from the airport location identifier.
[+]	UNUSED_COL	c143	Airport identification code of the accident/incident location, if on airport.
[+]	UNUSED_COL	c205	Date that the case information is originally entered.
[+]	UNUSED_COL	c206	Date the .19 form was received by the office.
[+]	UNUSED_COL	c207	Date the .4 form was received by the office.
[+]	UNUSED_COL	c208	Date that any supplemental data was received by the office.
[+]	UNUSED_COL	c210	Date that the document was last modified.
[+]	UNUSED_COL	c43	Not used at present time.
[+]	UNUSED_COL	c129	NTSB File number of the accident/incident.
[+]	UNUSED_COL	c124	Y or N.
[+]	UNUSED_COL	c125	Y or N.
[+]	UNUSED_COL	c77	Primary cause factor text

Ilustración 3 Columnas eliminadas

Una vez con la dimensión deseada del conjunto de datos, se realiza el guardado de Dataframe. Por el elevado número de registros, se valora realizarlo en formato “**.parquet**” por estar preparado para conjunto de datos **BigData**, para ello, se usa de la librería **pyarrow**.

Durante la conversión del fichero, salta una serie de errores **ArrowInvalid** ya que el modo de proceder para generar el fichero es realizar una conversión binaria de los datos. Por lo tanto, es necesario que todos los datos de una columna tengan el mismo formato. Para poder subsanar las excepciones y convertir finalmente el fichero, se realiza una serie de conversiones en las columnas afectadas.

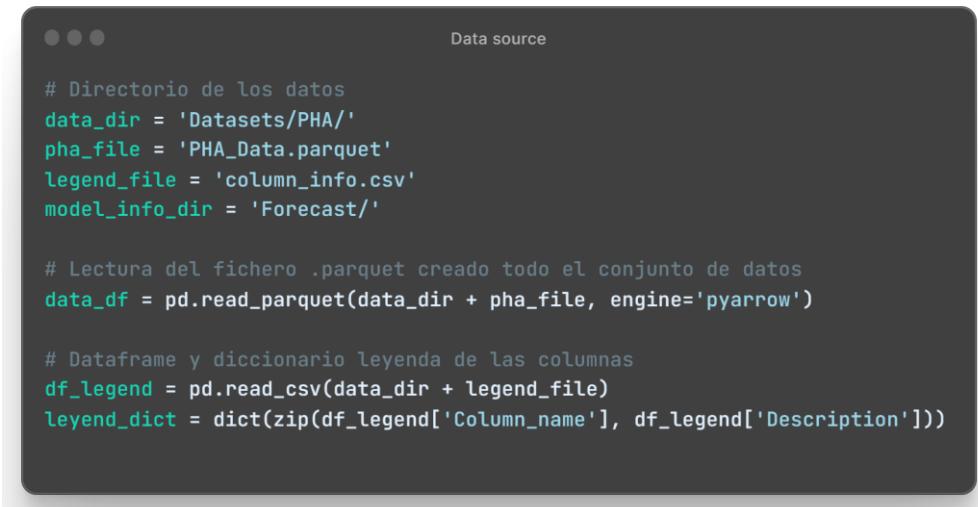
## 1.2 Forecast

### 1.2.1 Obtención de datos

Como se menciona, previamente se ha obtenido los datos de la fuente y preprocesados para poder trabajar en la creación de los modelos, partiendo de la misma base.

En este caso, para obtener los datos en un Dataframe, se accede a los datos previamente almacenador en formato “**.parquet**”.

Además, se crea el Dataframe, leyenda, es decir, la descripción de las columnas para trabajar con la información completa.



```
# Directorio de los datos
data_dir = 'Datasets/PHA/'
pha_file = 'PHA_Data.parquet'
legend_file = 'column_info.csv'
model_info_dir = 'Forecast/'

# Lectura del fichero .parquet creado todo el conjunto de datos
data_df = pd.read_parquet(data_dir + pha_file, engine='pyarrow')

# Dataframe y diccionario leyenda de las columnas
df_legend = pd.read_csv(data_dir + legend_file)
legend_dict = dict(zip(df_legend['Column_name'], df_legend['Description']))
```

Script 6 Obtención de los datos

### 1.2.2 Exploratory Data Analysis (EDA)

#### Reducción de dimensionalidad

Dado que el número de columnas es elevado y alguna de estas no aporta información al caso de estudio, se realiza primero la selección de los atributos de estudio dejando así un Dataframe que sea más reducido en cuanto a sus atributos de partida para poder trabajar el análisis de datos con mayor facilidad y relación al modelo necesario.

Como primer estudio para ver la relación de los datos y tomar la decisión de los atributos a seleccionar, se comprueba la correlación de los atributos. Esto es, la dependencia que tiene una columna con respecto a las otras si estas varían.

Para este estudio, se emplea el método *corr* de Pandas. Pero, previamente se requiere codificar las variables no numéricas ya que esta función, únicamente actúa sobre las columnas numéricas, y es este momento, se quiere ver la relación de todas las columnas. Para ello, se hace uso del módulo *Preprocessing* de **scikit-learn**.

Con este módulo se puede hacer uso de métodos que permite codificar los atributos categóricos a numéricos. Se utiliza el encoder **OrdinalEncoder** que transforma el valor en el ordinal para la columna, dando como resultado valores numéricos de 0 a n-1 característica.

Para poder representar los datos, se realiza un Dataframe. Este Dataframe se ajusta a partir de los datos obtenidos de la matriz de correlaciones obtenida con el método *corr*. Concretamente, se elimina la diagonal principal, por ser todos sus datos de correlación igual a 1, así como los datos por debajo de esta al ser duplicados de los de arriba de esta diagonal.

```

● ● ● Dimensionality reduction

# Instancia del codificador
oe = OrdinalEncoder()

# Ajuste del modelo (fit) y codificación de los datos (transform)
oe.fit(data_df)
data_encoded = oe.transform(data_df)

# Crear un datafram auxiliar con los atributos y valores transformados
names = oe.get_feature_names_out()
df_encoded = pd.DataFrame(data_encoded, columns=names)

# Correlacion de atributos
corr = df_encoded.corr()
# Filtrado de correlaciones
corr_filter = corr.where(np.triu(
    np.ones(corr.shape),
    k=1).astype(np.bool_))

# Ordenar de mayor a menor
corr_sort = corr_filter.stack().sort_values(ascending=False).to_frame()
# Añadir nombres a las columnas para entender las relaciones
corr_sort = corr_sort.rename(columns=leyend_dict, index=leyend_dict)

# Mostrar datos em bloques
corr_sort.head(60)

```

Script 7 Reducción de dimensionalidad

<b>Including crew and passengers.</b>	<b>Number of passengers onboard</b>	0.999322
<b>Region of the air operator</b>	<b>District office of the air operator</b>	0.994042
<b>Region of the accident/incident location.</b>	<b>District office of the accident/incident location.</b>	0.986830
<b>Code for Weight Class</b>	<b>Powered, Nonpowered, Optional</b>	0.969616
<b>Under or over 750 hp.</b>	<b>Type of engine.</b>	0.967706
<b>Type of the engine code.</b>	<b>Piston, turbine, turboprop, etc.</b>	0.960989
<b>Residence region code of the pilot in command</b>	<b>Residence district office code of the pilot in command</b>	0.958168
<b>Under or over 750 hp.</b>	<b>Type of the engine code.</b>	0.951808
<b>Type of the engine code.</b>	<b>Type of engine.</b>	0.947604
<b>Latitude coordinates of the accident/incident</b>	<b>Text for element C153.</b>	0.940691
<b>Second remedial action area code</b>	<b>Longitude coordinates of the accident/incident</b>	0.939748
	<b>Second remedial action area text</b>	0.938856
	<b>Second remedial action taken code</b>	0.932946

Ilustración 4 Ejemplo de correlaciones

Se representa los 180 primeros datos del Dataframe para visualizar las correlaciones. Analizando los resultados, se observa que las primeras 60 correlaciones, hace referencia a atributos que no tienen relación con el caso de estudios, estos datos hacen referencia a número de personas muertas o heridas o incluso el número de tripulantes y pasajeros. Estas fuertes correlaciones se entienden lógicas ya que, frente a un incidente o accidente, no existe discrepancia en el tipo de persona del avión.

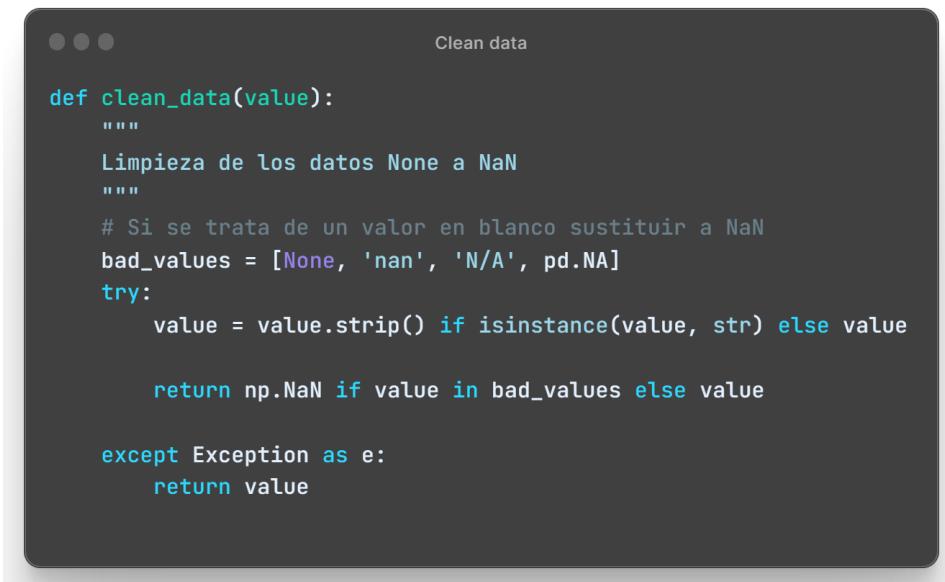
En las siguientes correlaciones, se aporta un mayor número de relaciones interesantes para el estudio del modelo. Se puede extraer que las **Primary flying condition** tienen una importante correlación con el resultado del suceso. Además, los atributos climáticos como **velocidad y dirección del viento** toman relevancia.

También se observa que el tipo de vuelo y la calificación del piloto tienen cierta relación. Esto se considera interesante ya que se entiende de la calificación/experiencia de un piloto puede ser determinante en determinadas situaciones climáticas.

Teniendo en cuenta las correlaciones y el modelo de datos necesario para la predicción, se selecciona las columnas en base a:

- Condiciones atmosféricas como: visibilidad, viento, altura de las nubes.
- Condiciones básicas del avión como: aeronavegabilidad del avión, horas de vuelo o antigüedad del avión.
- Aptitudes del piloto como: horas de vuelo o tipo de piloto.
- Condiciones primarias de vuelo, fase de vuelo.

Para crear el Dataframe del modelo, se emplea una función que permite realizar limpieza de los datos de cada atributo que se escoge. Esto se debe a que en la conversión del fichero “parquet” se definen algunos valores en diferentes formatos, pero todos ellos, datos nulos. Para ello, se hace uso de la función *clean\_data (Script 8 Limpieza de datos)*.



```
def clean_data(value):
    """
    Limpieza de los datos None a NaN
    """
    # Si se trata de un valor en blanco sustituir a NaN
    bad_values = [None, 'nan', 'N/A', pd.NA]
    try:
        value = value.strip() if isinstance(value, str) else value
        return np.NaN if value in bad_values else value
    except Exception as e:
        return value
```

Script 8 Limpieza de datos

Para generar el Dataframe del modelo, se parte del global y se elimina de este aquellas columnas que no están entre las candidatas. Finalmente, se obtiene un conjunto de datos de 28 atributos.

[+] SELECTED_COL	c1	Type of Event
[+] SELECTED_COL	c6	Year the accident/incident happened.
[+] SELECTED_COL	c7	Month the accident/incident happened.
[+] SELECTED_COL	c10	Local time of the accident/incident.
[+] SELECTED_COL	c11	Region of the accident/incident location.
[+] SELECTED_COL	c12	District office of the accident/incident location.
[+] SELECTED_COL	c13	State of the accident/incident location.
[+] SELECTED_COL	c14	City of the accident/incident location.
[+] SELECTED_COL	c20	Latitude coordinates of the accident/incident
[+] SELECTED_COL	c21	Longitude coordinates of the accident/incident
[+] SELECTED_COL	c30	Airworthiness class code of the aircraft
[+] SELECTED_COL	c31	Airframe hours of the aircraft
[+] SELECTED_COL	c32	This field contains the year of manufacture of the aircraft
[+] SELECTED_COL	c41	Certificate type code of the pilot in command
[+] SELECTED_COL	c49	Qualification code of the pilot in command .
[+] SELECTED_COL	c56	Total number of hours the pilot has flown.
[+] SELECTED_COL	c96	Phase of flight code
[+] SELECTED_COL	c106	Primary flying condition code
[+] SELECTED_COL	c108	Secondary flying condition code
[+] SELECTED_COL	c110	Light condition code
[+] SELECTED_COL	c112	Sky condition code
[+] SELECTED_COL	c113	Cloud ceiling
[+] SELECTED_COL	c114	Visibility code
[+] SELECTED_COL	c115	Visibility restriction code.
[+] SELECTED_COL	c240	Wind direction
[+] SELECTED_COL	c241	Wind speed in miles per hours
[+] SELECTED_COL	c242	Gust indicator flag
[+] SELECTED_COL	c243	Gust speed in miles per hour

Ilustración 5 Columnas seleccionadas

## Análisis de datos

Con el Dataset que será el conjunto de datos de estudio, se realiza una serie de análisis para poder encontrar un patrón en los datos o alguna característica de estos que sea reseñable o a tener en cuenta a la hora de procesar los datos en pos de obtener el modelo de los datos.

Como primer paso, se muestra el conjunto de datos en su totalidad, ofreciendo así el detalle, o información atendiendo al método pandas, del Dataframe. Además, para ver de forma visual el impacto de los datos NaN, se realiza una función *plot\_nan* que muestre en un gráfico los datos NaN en porcentaje.

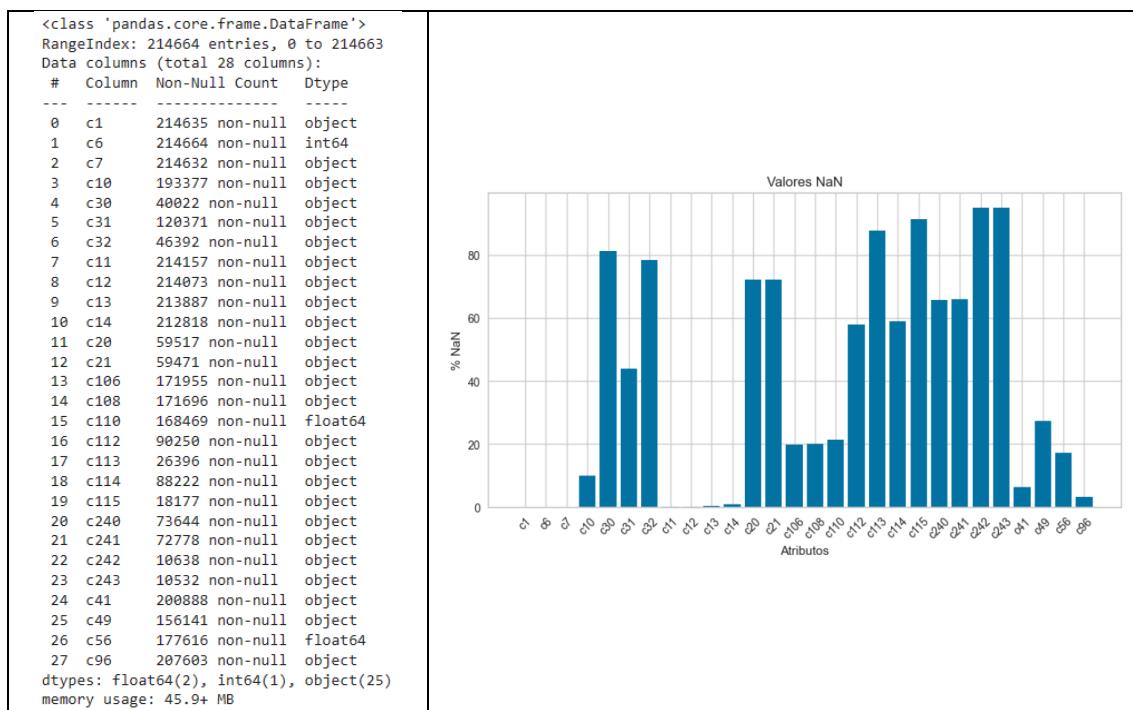


Ilustración 6 Información de los datos del Dataframe

```

NaN Plot

def df_info_perc(df) -> dict:
    """
    Obtener el porcentaje de datos faltantes para plotear
    """
    # Diccionario con datos
    info_dic = {}
    # Numero de datos faltantes para las diferentes columnas
    rows = len(df)

    for col in df:
        nan_values = df[col].isna().sum()
        perc = round((nan_values/rows) * 100, 2)

        # Almacenar datos descriptivos
        info_dic.update({col: perc})

    return info_dic

def plot_nan(df):
    """
    Imprimir el porcentaje de valores faltantes
    """
    # Describir los datos faltantes
    info = df_info_perc(df)

    # Mostrar informacion
    plt.figure(figsize=(10, 5))
    plt.bar(info.keys(), info.values())
    plt.xlabel('Atributos')
    plt.xticks(rotation=45)
    plt.ylabel('% NaN')
    plt.title('Valores NaN')

    plt.show()

```

Script 9 Representación de datos

En los modelos de clasificación, como se trata del caso de estudio, es de importancia comprobar que las clases se encuentran balanceadas. Es decir, que para cada uno de las clases que existe en el Dataset, existe una misma cantidad de registros. Esto es de importancia ya que un desbalanceo, puede inferir en el resultado del entrenamiento haciendo que el modelo actúe mejor frente a un tipo de datos y peor frente a otros.

En este modelo, se realiza la predicción del tipo de accidente, columna **c1**, por lo que, para poder comprobar el balanceo, se cuenta el total de valores únicos que existe en dicho atributo o variable objetivo, es decir, obtener sus clases.

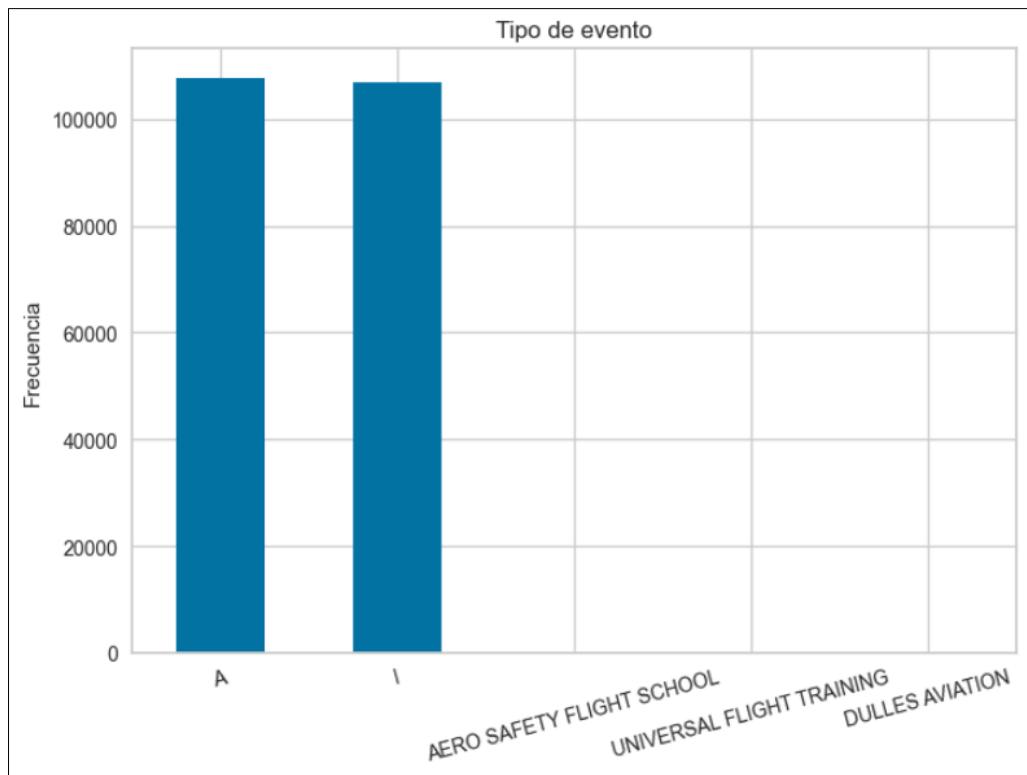


Ilustración 7 Balanceo de clases

Para este conjunto de datos, se observa que existe un balanceo de clases, concretamente, ambas tienen en torno a 110000 registros. Además, este análisis, muestra que existe un total de 3 clases que no tienen que ver con las esperadas. Y es que, las clases esperadas en el Dataframe son **A** (Accidente) e **I** (Incidente).

Otro aspecto importante a estudiar se trata de la geolocalización del avión en el momento del suceso. Es por esto que se añade los atributos de latitud y longitud al conjunto de características a trabajar en el Dataset. No obstante, estos atributos son presentados de forma categórica, por lo que, para poder emplearlos, se convierte a numérico.

Esto se debe a que las coordenadas están presentadas en grados y minutos en su origen, por lo tanto, estos valores se convierten a coordenadas decimales como se presenta a continuación.

```
def convert_coords(coord):
    """
    Transformar las coordenadas en Grados y Minutos a Coordenadas decimales
    """
    err = 999
    dir_tuple = ('N', 'S', 'E', 'W')
    # Las coordenadas en el dataframe se dan en string. Si no es string,
    # devolver valor atípico 0
    if not isinstance(coord, str):
        return err

    # Encontrar los valores numéricos
    coord = coord.strip()
    match = re.findall(r'[\d.]+', coord)

    # Son datos de coordenada válidos
    if not coord.endswith(dir_tuple) or not match:
        return err
    direction = coord[-1:]

    # Si viene Grados y Minutos sin espacio
    if len(match) == 1:
        grade, minute = 0, 0
        if len(match[0]) == 5:
            grade = match[0][:3]
            minute = match[0][3:5]

        elif len(match[0]) == 4:
            grade = match[0][:2]
            minute = match[0][2:4]

        if grade or minute:
            match = [grade, minute]

    # Si viene separado Grados de Minutos
    if len(match) == 2:
        match = [float(match[0]), float(match[1])]
        # Comprobación de Grados
        if direction in ['N', 'S']:
            if match[0] < 0 or 90 < match[0]:
                return err
        if direction in ['E', 'W']:
            if match[0] < 0 or 180 < match[0]:
                return err
        # Comprobación de Minutos
        if match[1] < 0 or 59 < match[1]:
            return err

        # Convertir a decimal
        decimal_coord = match[0] + match[1]/60

        # Convertir dirección
        return decimal_coord if direction in ['N', 'E'] else decimal_coord * -1
    else:
        return err

# Convertir las coordenadas a decimales
df['c20'] = df['c20'].apply(convert_coords)
df['c21'] = df['c21'].apply(convert_coords)
```

Script 10 Coordenadas conversión

Con las coordenadas en valores numéricos, se puede representar estas y así poder tener la idea visual de la distribución de los sucesos. Para ello se hace uso de la librería **geopandas** para poder representar en un heatmap las coordenadas de los sucesos registrados.

```
# Crear los puntos geometricos para el mapa a partir de Latitud y Longitud
geometry = geopandas.points_from_xy(df.c20, df.c21)
geo_df = geopandas.GeoDataFrame(df[["c20", "c21"]], geometry=geometry)

# Crear instancia de mapa de folium
geo_map = folium.Map(location=[0, 0], tiles="Cartodb dark_matter", zoom_start=2)

# Crear lista de puntos de coordenadas que sean diferentes a 999
heat_data = [[point.xy[0][0], point.xy[1][0]] for point in geo_df.geometry if point.xy[1][0] != 999 and point.xy[0][0] != 999]

# Añadir al mapa los puntos como Heatmap
plugins.HeatMap(heat_data).add_to(geo_map)
geo_map
```

Script 11 Plot heatmap

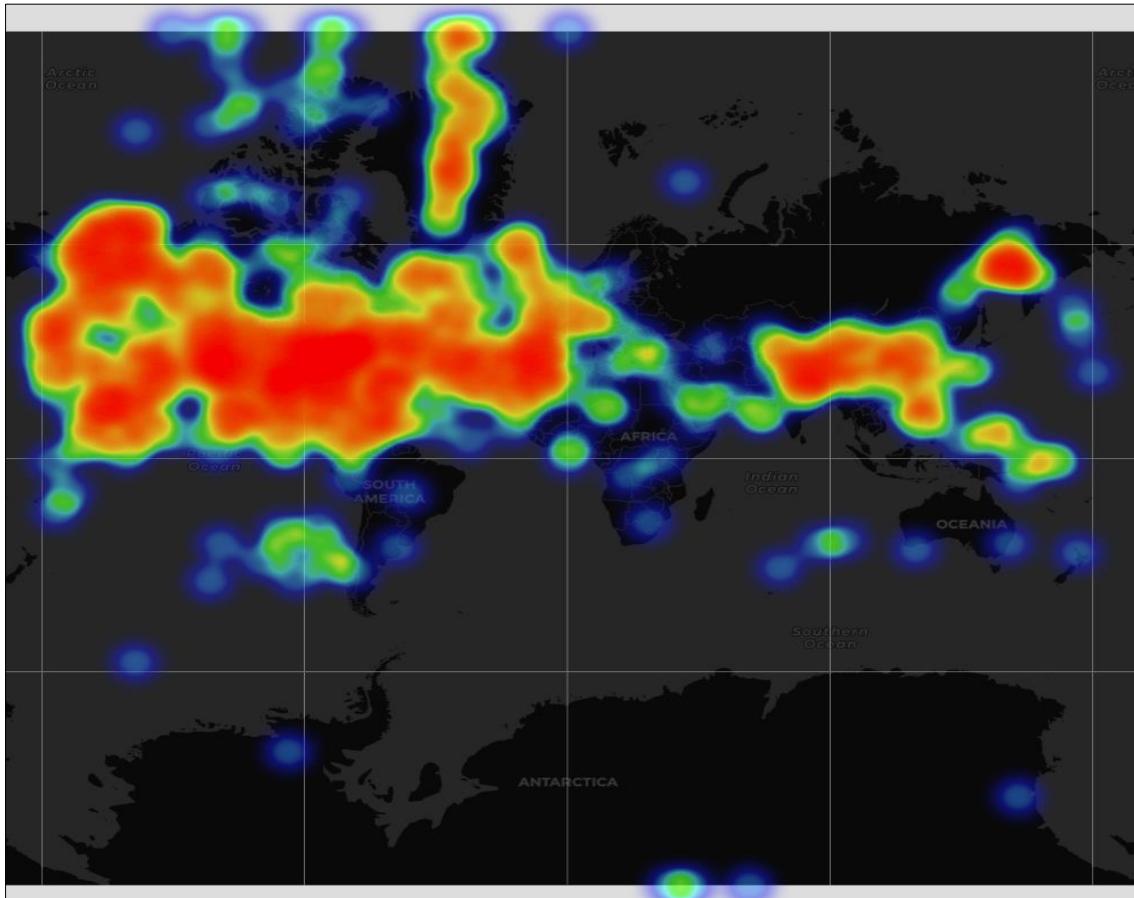


Ilustración 8 HeatMap

## Conclusión

Tras realizar los análisis necesarios y observar los valores de todas las columnas, se concluye:

- Existe la mitad de columnas que contiene más del 50% de datos nulos o vacíos a tener en cuenta a la hora de la selección de características para el estudio. Además de valores atípicos para las columnas por encontrarse valores desplazados.
- Se observa que las clases de estudio (*Accidente o Incidente*) están balanceadas, así como el lugar de los sucesos que se distribuye por gran parte de la geografía mundial. En resumen, se puede decir que se está con un conjunto de datos balanceado.
- Existe el atributo c32 que hace referencia al año de construcción del avión, pero todos sus valores son \* o bien *NaN*. Esta característica se escoge en un principio junto con el año del suceso para conocer la antigüedad del avión en el momento del suceso. Por lo tanto, dada la falta de valores de dicha columna, se desestima del conjunto de datos ambos atributos.
- El atributo c242 contiene datos que no son claros sin más indicaciones a las que aporta la información del Dataset. Es por ello que se elimina dicha columna del conjunto de datos, además, a nivel conceptual en el Dataset, para las rachas de viento existe un segundo atributo, el c243, por lo que dicha característica del incidente, se seguirá pudiendo analizar.
- Se observa que la gran mayoría de atributos hace referencia a tipo objeto. Se debe a que determinados de estos atributos, tiene datos categóricos que mezcla números y letras en sus datos. No obstante, se analiza más adelante apoyándose de la descripción del Dataset cuál de estas son realmente este tipo de columnas y cuáles son fruto de un error en los datos.

### 1.2.3 Data Preprocessing (DP)

Una vez se ha analizado el Dataset y se conoce las características principales de los datos, es necesario procesar los datos antes de crear un modelo de datos. En este punto, se va a corregir o realizar una “limpieza” de los datos que se tiene.

#### Eliminación de características

Como detalla las conclusiones del punto anterior, la característica c32 no tiene los valores esperados, así que se elimina junto a c6 por ser la columna de apoyo escogida. También se elimina c242 por no aportar datos claros como se ha comentado.

En resumen, los atributos **c6**, **c32** y **c242** se eliminan del Dataset.

#### Valores desplazados

Tras el EDA, se observa que puede existir valores desplazados entre los atributos del Dataframe. Para comprobar y corregir este suceso, se realiza un análisis de los valores que ha de tener cada uno de los atributos categóricos con los que realmente tiene en el Dataset y así hallar los desplazamientos y poder relocalizarlos en caso de ser posible.

Este análisis se realiza sobre todos los atributos de Dataset y no sobre el filtrado para el modelo de estudio, de este modo, se permite hallar los desplazamientos de los atributos filtrados en todo el conjunto de datos. Además, son las columnas categóricas las escogidas para el análisis ya que son las únicas que pueden aportar la información de datos esperados.

Como primer paso de este análisis, con la ayuda de la información del Dataset descrita en el **ANEXO A**, se define las variables que contiene los diferentes valores que cada atributo, del Dataset filtrado, puede tomar.

```
# Airworthiness class code of the aircraft
c30 = ['STRD', 'LIMT', 'REST', 'EXPT', 'PROV', 'MULT', 'FERY']
# Certificate type code of the pilot in command
c41 = ['F1', 'F3', 'F9', 'TT', 'XX', '00', '01', '02', '03', '09']
# Qualification code of the pilot in command
c49 = ['E', 'F', 'G', 'H','M', 'N', 'P', 'R', 'S', 'T', 'U', 'V', 'W', 'X']
# Phase of flight code
c96 = ['AA','AB','AC','AD','BA','BB','BC','BD','BE','BF','BG','BH','BI','CX',
       'DA','DB','DC','DD','DE','DF','DG','DH','EX','FA','FB','FC','FD','FE',
       'FF','FG','GX','HA','HB','HC','HD','HE','HF','HG','HH','HK','HM','IA',
       'IB','IC','ID','IE','IF','IG','IH','JX','KA','KB','KC','KD','KE','KF',
       'KG','PJ','PL','SR']
```

Script 12 Asignación de datos categóricos

Con las listas definidas, se crea un bucle que itera sobre una lista de columnas categóricas a analizar. Este análisis consiste en crear un match de los valores de los atributos de Dataframe filtrado con los del Dataframe original, de este modo, se puede conocer los valores esperados de cada atributo y en que otras columnas se encuentran para así poder analizar la posibilidad de desplazamiento o no. A continuación se muestra el código que permite realizar dicho análisis y un ejemplo del resultado obtenido.

```

● ● ●

# Lista tuplas que contienen la lista de valores y nombre de la columna
cat_cols = [('c1', c1), ('c11', c11), ('c12', c12), ('c13', c13), ('c30', c30),
('c41', c41), ('c49', c49), ('c96', c96), ('c106', c106), ('c108', c108),
('c110', c110), ('c112', c112), ('c114', c114), ('c115', c115)]


# Buscar columnas que coincide con valores que son de la columna categótrica del
# modelo de estudio
match_cols = {}
for index, data in enumerate(cat_cols):
    # Crear un dataframa con valores True/False para aquellos que hay
    coincidencia
    df_match_values = data_df.isin(data[1])

    # Crear un dataframe de dos columnas que indica para columna si hay
    coincidencia o no
    match = df_match_values.any()

    print(f'[+] COLUMN_DATA\t{data[0]}')
    print(f'COLUMN_VALUES\t{data[1]}')
    # Devolver el indice del match que es True, es decir, que tiene coincidencia
    match = list(match[match == True].index)
    match_cols.update({data[0]: match})
print(f'MATCH_COLUMNS\t{match}', end='\n\n')

```

Script 13 Análisis de valores desplazados

```

[+] COLUMN_DATA c30
COLUMN_VALUES  ['STRD', 'LIMIT', 'REST', 'EXPT', 'PROV', 'MULT', 'FERY']
MATCH_COLUMNS   ['c30']

[+] COLUMN_DATA c41
COLUMN_VALUES  ['F1', 'F3', 'F9', 'TT', 'XX', '00', '01', '02', '03', '09']
MATCH_COLUMNS   ['c13', 'c240', 'c241', 'c243', 'c41', 'c59', 'c121', 'c127']

```

Ilustración 9 Análisis de valores desplazados

Con este análisis se observa que existe para todos los atributos valores que a su vez están en otras columnas del Dataset. Este resultado no es en sí un claro indicador de que existe valores desplazados, pues como se muestra en el ejemplo, existe valores como *00* o *01* que podrían ser perfectamente compartidos por otros atributos, pero en cambio, valores como *STRD* o *LIMIT* sí que se trata de únicos para la columna *c30*.

Visto esto, se realiza un segundo análisis en el que, excluyendo los atributos cuyas coincidencias no indican un desplazamiento de valores como se ha detallado, se analiza cuáles son los valores repetidos en los diferentes atributos. Para este análisis, se tiene en cuenta que los valores coincidentes no son los de un estado ya que estos, existen en otras columnas y son un elevado número de datos para poder monitorizarlos.

A continuación, se muestra el código que permite realizar el análisis y el ejemplo de su resultado.

```

● ● ●

# Existe unos códigos de estados los cuales son compartidos por varias columnas
states = c13

def moved_values(cat_column: str, conflict_col: list):
    # Obtener los valores de la columna categórica
    cat_values = [col[1] for col in cat_cols if col[0] == cat_column][0]
    print(f"CAT_COLUMN\t{cat_column}\t{col_name(cat_column)}\nVALUES\t{cat_values}\n")

    # Iterar las columnas y mostrar los valores únicos
    for col in data_df:
        # Continuar si no hay coincidencia de valores o es la misma columna
        if col not in conflict_col or col == cat_column:
            continue

        match_col_values = data_df[col].unique()

        # Imprimir las coincidencias
        res = [val for val in match_col_values if str(val) not in states]
        print(f"[+] {col}\t{col_name(col)}")
        # print(f"VALUES\t{match_col_values}")
        matches = set(cat_values) & set(res)
        if not matches:
            print('STATE MATCHES\n')
            continue

        print(f"COMMON_VALUES\t{matches}\n")

```

*Ilustración 10 Análisis de valores movidos en un atributo*

```

CAT_COLUMN      c41      Certificate type code of the pilot in command
VALUES  ['F1', 'F3', 'F9', 'TT', 'XX', '00', '01', '02', '03', '09']

[+] c13 State of the accident/incident location.
COMMON_VALUES  {'TT'}

[+] c240      Wind direction
COMMON_VALUES  {'03', '00', '01', '02', '09'}

[+] c241      Wind speed in miles per hours
COMMON_VALUES  {'03', '00', '01', '02', '09'}

[+] c243      Gust speed in miles per hour
COMMON_VALUES  {'03', '00', '01', '02', '09'}

[+] c59 Residence state of the pilot in command
COMMON_VALUES  {'TT', '00'}

[+] c121      State of the owner / operator
COMMON_VALUES  {'00'}

[+] c127      Region of the air operator
COMMON_VALUES  {'03'}

```

*Ilustración 11 Análisis de valores movidos en un atributo*

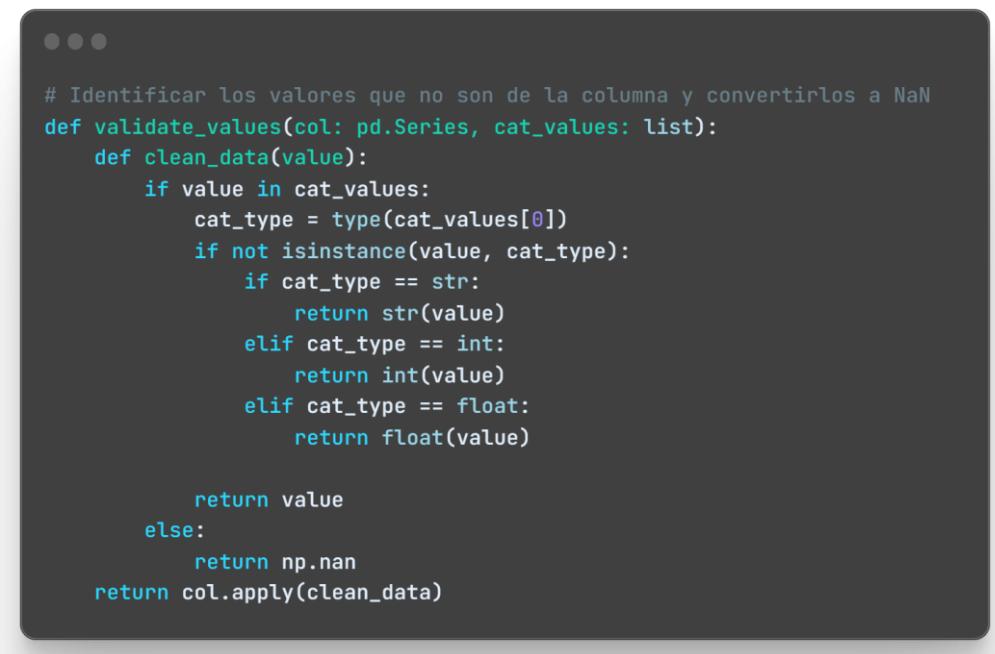
En el ejemplo que se muestra, existe gran número de coincidencia para los valores *01, 02, etc.*, por lo que no se decide aplicar ninguna corrección para estos valores, no obstante, se observa para el atributo c13 y c59 una coincidencia de valores que son únicos para c41. Por lo tanto, estos valores sí que son relocalizados.

Tras las comprobaciones de todas las columnas seleccionadas como objeto de análisis, se concluye que:

- Atributos c13 y c59 tienen valores del atributo c41.
- Atributo c94 tiene valores del atributo c96.
- Atributo c113 tiene valores del atributo c112.

La recolocación de valores se realiza si, para el atributo objetivo, existe un valor *NaN* y el de la misma fila del atributo erróneo contiene el dato que debiera estar en la columna original.

Para poder realizar estos cambios, previamente se convierte a *NaN* todos aquellos valores de las columnas categóricas preseleccionadas que sean inválidos ya que no se necesita un preprocesado de los valores de las columnas objetivo al ser óptimos en gran medida y no encontrar ningún valor atípico.



```
# Identificar los valores que no son de la columna y convertirlos a NaN
def validate_values(col: pd.Series, cat_values: list):
    def clean_data(value):
        if value in cat_values:
            cat_type = type(cat_values[0])
            if not isinstance(value, cat_type):
                if cat_type == str:
                    return str(value)
                elif cat_type == int:
                    return int(value)
                elif cat_type == float:
                    return float(value)

            return value
        else:
            return np.nan
    return col.apply(clean_data)
```

Script 14 Validación de valores en atributo

Para la recolocación de los valores foráneos, se crea una función que compruebe los valores de los atributos objetivo y aquellos que tienen valores foráneos y, tras validar las condiciones de relocalización mencionadas, recolocar los valores.

```
def foreign_values(c_origin,
                   searched_values,
                   c_foreign,
                   dataframe: pd.DataFrame,
                   source_df: pd.DataFrame = data_df):
    moved = 0
    match = 0
    origin_values = []

    for index, values in source_df.iterrows():
        if values[c_foreign] not in searched_values:
            continue
        match += 1
        if pd.isna(values[c_origin]):
            dataframe.loc[index, c_origin] = values[c_foreign]
            moved += 1
        else:
            origin_values.append(values[c_foreign])

    print(f'\tMATCH\t{match}\tMOVED\t{moved}')
    if origin_values:
        print(f'\t[!] VALUES_IN_ORIGIN\t{origin_values}'')
```

*Script 15 Recolocación de valores*

Para la recolocación de los valores, se indica, cuáles son los valores que se ha de encontrar, es decir, aquellos que se han desplazado, y cuáles son las columnas en las que buscar del Dataframe original en el que se encuentra el desplazamiento. A continuación, se muestra el ejemplo del código que permite la acción, así como el ejemplo de resultado.

```
column = 'c41'
column_foreign = ['c13', 'c59']
col_values = c41
searched_values = ['TT']

# Limpieza de los datos foraneos de la columna
df[column] = validate_values(df[column], col_values)

for c_for in column_foreign:
    print(f'[+] COLUMN\t{c_for}')
    foreign_values(c_origin=column, searched_values=searched_values,
                   c_foreign=c_for, dataframe=df)
```

*Script 16 Recolocación de valores*

```
[+] COLUMN      c13
    MATCH      6      MOVED      0
    [!] VALUES_IN_ORIGIN      ['TT', 'TT', 'TT', 'TT', 'TT', 'TT']
[+] COLUMN      c59
    MATCH      1      MOVED      0
    [!] VALUES_IN_ORIGIN      ['TT']
```

Ilustración 12 Recolocación de valores

En este ejemplo, no se ha realizado ningún cambio ya que el atributo c41 contiene, en la misma fila, un valor.

Finalmente, tras las comprobaciones, el atributo c112 es el único que recoloca un total de 45 valores de 57 coincidencias.

### Valores atípicos

Una vez se consigue todas las columnas con sus datos sin desplazar, se trabaja sobre aquellos valores atípicos o que sea faltantes en cada atributo. Este consiste en uno de los principales procesos del preprocesado de datos dado que lo más normal, es que exista Dataset con sus datos mal tipados o perdidos.

El análisis y modificaciones se comprueba la descripción de los valores de cada atributo y se analiza en función de que datos se debe esperar para cada atributo. En aquellos casos que sea necesario, se imputará un valor NaN, un valor flotante, para poder operar más adelante la limpieza de datos y eliminación de valores faltantes.

Teniendo en cuenta el tipo de dato de cada atributo, categórico o numérico, se realiza dos tipos de comprobaciones:

- Categóricos: Se comprueba que el valor de la columna es uno de los esperados.
- Numéricos: Se castea el tipo de dato a numérico.

Para la modificación de valores de la columna c10 (*Local time of the accident/incident*) se tiene datos categóricos, esto se debe a que, si fuera de otro modo, el valor para la hora 10 sería menor que para la hora siguiente 11, cobrando más importancia esta última, pero conceptualmente, una hora de diferencia no ha de suponer una mayor importancia para uno u otro valor. Dicho esto, se modifica los valores del atributo para que vayan desde las 00:00 hasta las 23:00, con lo que finalmente se obtiene un atributo de 24 valores.

```
def get_hour(value):
    """
    Convertir la hora en HHMM a HH
    """
    try:
        if value in[None, 'nan']:
            return np.NaN

        # Extraer la hora y minuto
        hour = int(value[:2])
        # Comprobar que la hora sea correcta y no sea un valor
        # ilogico
        if hour >= 24:
            return np.NaN
        minute = int(value[2:])
        hour += 1 if minute > 30 else 0
        # Cambiar la hora a 00 para aquellas que sumen 1 y sea las 23
        hour = 0 if hour == 24 else hour

        return str(hour).zfill(2)

    except (TypeError, Exception):
        return np.NaN

# Utilizar compresion de listas y aplicar el filtrado de valores
hours = [get_hour(value) for value in df['c10']]
df['c10'] = hours

df.c10.unique()
```

*Script 17 Conversión valores horarios*

### Imputación de valores

Tras el proceso anterior, se consigue un Dataset con los valores esperados para los atributos y en su defecto, el valor NaN. En esta situación, es posible realizar lo que se conoce como las imputaciones.

Este método consiste en sustituir los valores NaN por aquellos significativos para cada atributo con el fin de poder tener el mayor número posible de registros válidos en el Dataset.

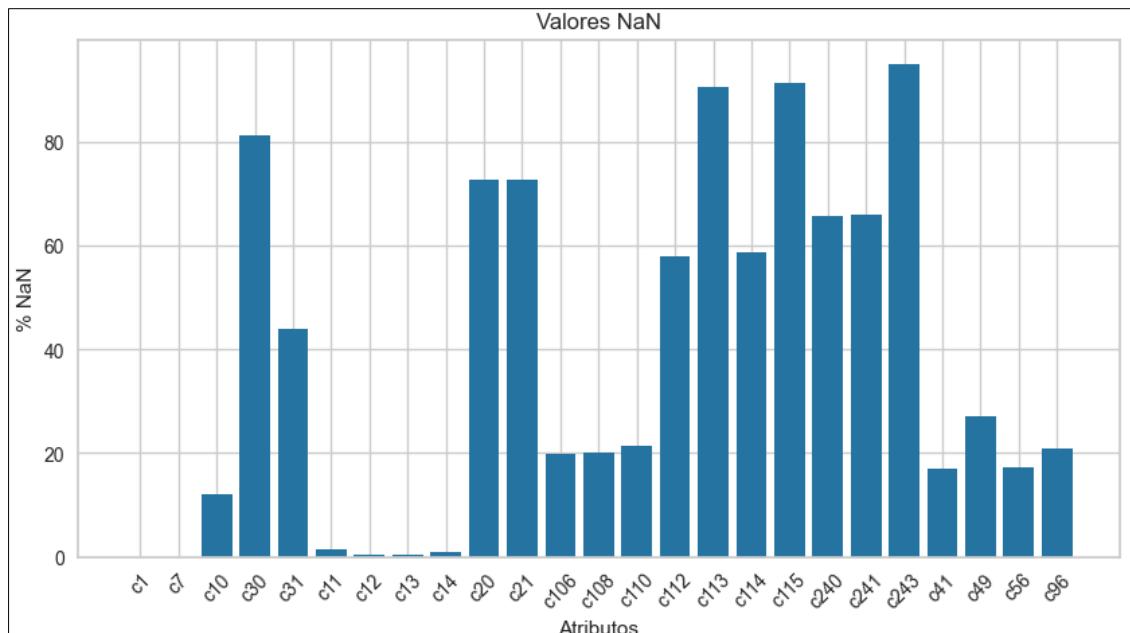


Ilustración 13 Valores NaN

En la Ilustración 13 se observa que existe 10 atributos con más del 50% de sus valores como NaN y el resto en torno al 20% o bien un porcentaje 0% o muy cercano a este.

Para realizar la imputación se tiene en cuenta la relación de los atributos. Es decir, como los valores de los atributos están relacionados entre ellos. Esto no se realiza teniendo en cuenta la correlación de las columnas, pues dicho proceso ya se utiliza para la selección de características, sino que se realiza teniendo en cuenta el concepto de cada uno de los atributos. Tras realizar este previo análisis de relación lógico de atributos, la selección de relaciones queda del siguiente modo:

- Localización del incidente

La localización del suceso está relacionada con los atributos que hacen referencia a la región, el distrito, estado y ciudad del suceso.

- Hora del incidente

Se relaciona la hora del incidente con la luminosidad en el momento del suceso por indicar el momento del día de este.

- Estado del cielo

La situación atmosférica se relaciona con los diferentes códigos de visibilidad, condición lumínica, restricción de visibilidad y estado del cielo. Además, se añade la altura de las nubes en el momento del suceso.

Para realizar las imputaciones teniendo en cuenta los valores NaN, se comprueba cuál de los atributos tiene un mayor número de estos datos nulos, de modo que sea este el primero sobre el que se realice las imputaciones y poder permitir así imputar el mayor número posible de datos.

Como norma general, para el proceso de imputaciones se realiza los siguientes pasos:

1. **Descripción de los datos del atributo:** Permite conocer qué tipo de dato existe en la columna y que cantidad de NaN existe en este. Este proceso se acompaña de una representación gráfica de los datos, bien sea datos numéricos o categóricos independientemente.
2. **Análisis de datos:** Como dato estadístico representativo para la imputación se tiene en cuenta el sesgo. Con el sesgo se puede conocer si los datos están desplazados hacia uno de los extremos en la gráfica de distribución o bien se encuentran centrado.
3. **Agrupación de atributos:** Este es un punto que se realiza en aquellos atributos que se encuentran relacionados como se menciona al inicio de este punto. Este proceso consiste en agrupar el conjunto de datos y crear un nuevo Dataframe cuyo índice sea fruto de los atributos agrupados. Para ello, la agrupación se realiza de menor a mayor número de valores, es decir, como primer nivel de la agrupación se escoge el atributo con menor número de valores únicos.

Como ejemplo se muestra la agrupación para la imputación del lugar del accidente. Esta agrupación da como resultado 28249 grupos.

			c20	c21	
		c11 c13	c14		
		AB	BONNYVILLE	53.916667	
AL	AK	ACKERMAN LAKE		NaN	
		ADAK ISLAND		NaN	
		ADMIRALTY ISLAN		NaN	
		AFOGNAK ISL		NaN	
		...		...	
		MAJURO MARSHALL		-0.016667 164.716667	
		OSAKA JAPAN		34.433333 135.233333	
WP	ZZ	PARDUBICE		50.000000 15.733333	
		SALTILO		NaN	
		TOKYO		36.033333 14.166667	
28249 rows × 2 columns					

Ilustración 14 Agrupación de ejemplo

4. **Imputación de valores:** La imputación de valores se realiza mediante la media o la mediana. Para ello, se tiene en cuenta el valor de sesgo obtenido, con el que se puede realizar una imputación por mediana si el sesgo es reducido o si por el contrario es elevado, mediante la media. La idea de este punto, es tratar de reducir lo máximo posible el sesgo en la distribución de los datos de los atributos.

En esta sección, una vez se realiza todas las imputaciones, se procede a una reimputación de los datos para aquellos atributos que están relacionados unos con otros. Esto se debe a que, al haber imputado valores en todos ellos, una reagrupación puede añadir nuevos valores.

En la reimputación, además se considera crear nuevas características en los atributos. Esto se realiza para permitir crear nuevos grupos que antes no se han creado y así, realizar un mayor número de imputaciones.

Tras el proceso descrito, como muestra la Ilustración 15, se consigue que únicamente el atributo c115 tenga más del 50% de sus valores faltantes y que el resto, a excepción de c243 y c30, estén por debajo del 20% de valores NaN.

En base a los resultados que se obtiene de este proceso, se realiza:

- Eliminar la columna c243 (*Gust speed in miles per hour*) ya que la columna c241 (*Wind speed in miles per hours*) tiene la misma información y posee un menor número de NaN.
- Eliminar la columna c240 (*Wind direction*) ya que, pese a que haya realizado imputaciones y sus datos NaN estén en torno al 20%, sus valores oscilan entre un rango de 0 a 90 grados, lo cual no se considera lógico y se entiende que esa columna posee datos anómalos.
- Eliminar la columna c30 (*Airworthiness class code of the aircraft*) ya que sus datos podrían aportar ruido al modelo y además existe un gran número de datos faltantes.
- Eliminar la columna c31 (*Airframe hours of the aircraft*) puesto que se ha imputado casi la mitad de sus datos, y su información no aporta mucho valor al caso de estudio.
- Eliminar las columnas c11, c12, c13 y c14 ya que, su existencia hasta este punto es de comodín para la imputación de los datos de c20 y c21.

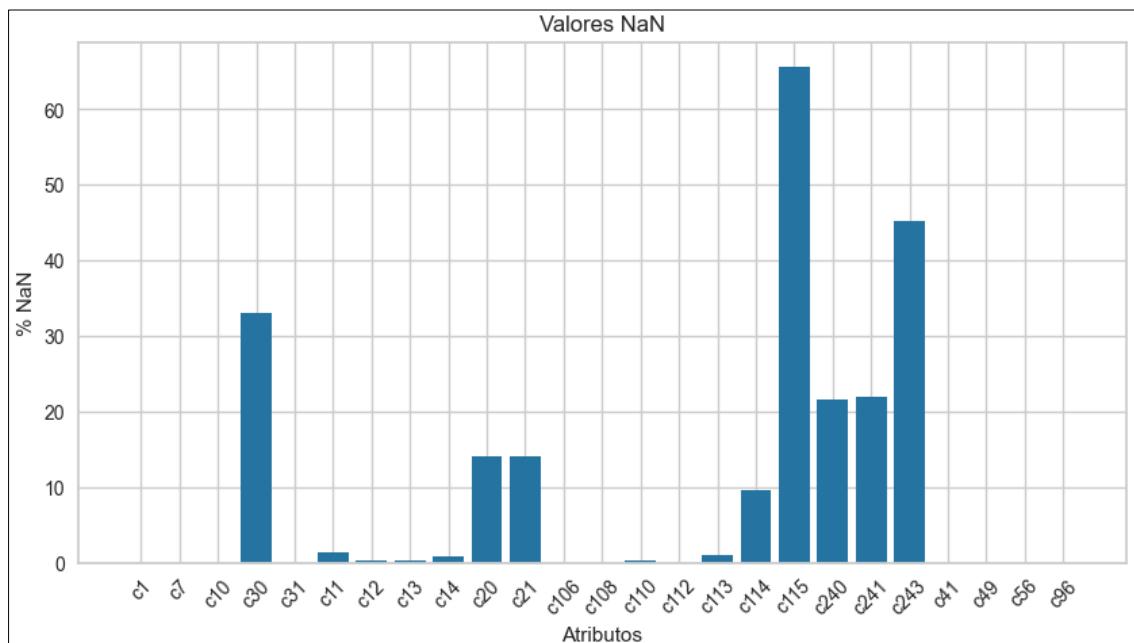


Ilustración 15 Valores NaN

Como paso final, se elimina todos los registros que contengan algún valor NaN.

### Conversión de datos

Para conseguir un correcto modelado de los datos, es necesario que todos los atributos contengan los datos en el tipo de datos esperado. En este punto, se realiza el mapeado de los datos al tipo de dato esperado.

Los cambios a realizar son:

- El atributo c7 (*Month the accident/incident happened*) contiene el tipo de datos numérico, no obstante, sus datos han de ser categóricos ya que el valor numérico del mes no aporta mayor peso sobre otro.
- El atributo c96 contiene una gran variedad de características, no obstante, se crea nuevas características para este atributo, se modo que se englobe los datos actuales del atributo en los siguientes:
  - **AIRBNE:** Airborne o en vuelo.
  - **T/O:** Takeoff o despegando.
  - **LDG:** Landing o aterrizando.
  - **GND:** Ground o en tierra.
  - **SPL:** Special o maniobras de vuelo especiales.
  - **OTH:** Others u otros.

#### 1.2.4 Feature Engineering (FE)

En este punto, se realiza las diferentes operaciones necesarias para poder crear un modelo de predicción y que, además, este sea creado con el mejor rendimiento posible.

#### Codificación de datos categóricos

Para poder realizar un modelo de predicción, se necesita trabajar con datos numéricos. Para ello, los atributos categóricos deben ser transformados a numéricos mediante el uso de las diferentes técnicas existentes:

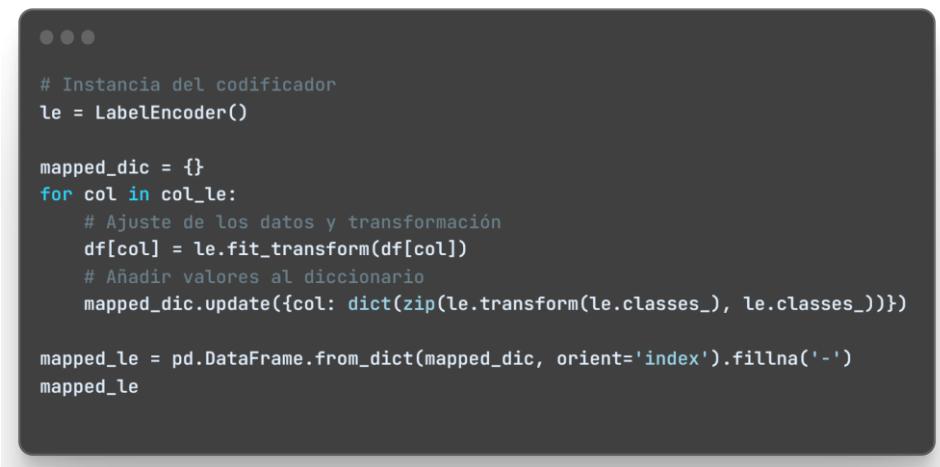
- **OrdinalEncoder:** Se emplea en codificar variables categóricas que no tienen una relación de orden natural. Devuelve una única columna que va de 1 a n\_caracteristicas-1.
- **LabelEncoder:** Se emplea en codificar variables categóricas que tienen una relación de orden natural. Devuelve una columna de 1 a n\_caracteristicas-1. Su uso se realiza principalmente para transformar la variable dependiente.
- **OneHotEncoder:** Se utiliza para codificar variables categóricas que no tienen una relación de orden natural. Devuelve una columna para cada atributo codificado, indicando 0 o 1 según la presencia de la característica para cada registro.

En base a los métodos existentes, se realiza los siguientes cambios en los diferentes atributos:

- Para aquellos atributos categóricos cuyos valores vienen representados como numéricos, simplemente se les aplica un cambio de tipo de datos a entero para poder ser interpretado por el modelo.
- Para la variable dependiente u objetivo, c1, así como para los atributos de entrada cuyos datos tienen un orden de relación, se emplea LabelEncoder.
- Para los atributos que poseen pocas clases y además tienen una mayor importancia en el significado del estudio, se emplea OneHotEncoder.
- Para el resto de columnas, se realiza un OrdinalEncoder.

Los atributos que reciben un mapeado de los datos, aquellas con las que se emplea LabelEncoder u OrdinalEncoder, es necesario almacenar la relación de las características con su valor numérico tomado. Para ello, se crea un diccionario con la información del mapeo devuelta por el framework scikitlearn, empleado en la transformación de los valores, que posteriormente es

almacenado en formato .csv y así posteriormente poder ser cargado y poder realizar las conversiones necesarias previas a la predicción de resultados. Véase el *Script 18*



```
# Instancia del codificador
le = LabelEncoder()

mapped_dic = {}
for col in col_le:
    # Ajuste de los datos y transformación
    df[col] = le.fit_transform(df[col])
    # Añadir valores al diccionario
    mapped_dic.update({col: dict(zip(le.classes_, le.classes_))})

mapped_le = pd.DataFrame.from_dict(mapped_dic, orient='index').fillna(' ')
mapped_le
```

*Script 18 Diccionario de mapeos*

## Normalización atributos numéricos

Para que los diferentes atributos numéricos estén dentro de la misma escala y el modelo sea capaz de interpretar los valores dentro del mismo rango en importancia, se realiza la normalización de los valores.

Para ello, se realiza el escalado entre 0 y 1 de los valores o bien el normalizado de los mismos atendiendo a su desviación estándar.

### 1.2.5 Data Modeling

Para entrenar el modelo, es necesario realizar una serie de segmentaciones en los datos que se tiene. En concreto, se realiza un total de tres particiones de los datos, siendo éstas las siguientes:

- **Data:** 95% de los datos con los que se realiza el modelo. Sus valores son generalmente del 70% para el entrenamiento del mismo y el 30% para la fase de test.
- **Data unseen:** 5% de los datos que se reserva para la validación del modelo. Estos son empleados para realizar predicciones con el modelo ya generado y testeado.

## 1.3 Crew

### 1.3.1 Obtención de datos

En primer lugar, se obtienen los datos y se carga el modelo de la fuente (U.S. Department of Transportation, Federal Aviation Administration, s.f.)

```
...  
  
# Directorio de los datos  
data_dir = 'Datasets/PHA/'  
pha_file = 'PHA_Data.parquet'  
legend_file = 'column_info.csv'  
model_info_dir = 'Crew/'  
# Lectura del fichero .parquet creado todo el conjunto de datos  
data_df = pd.read_parquet(data_dir + pha_file, engine='pyarrow')  
  
# Dataframe y diccionario leyenda de las columnas  
df_legend = pd.read_csv(data_dir + legend_file)  
legend_dict = dict(zip(df_legend['Column_name'], df_legend['Description']))
```

Script 19 Carga de datos

### 1.3.2 Exploratory data Analysis (EDA)

#### Reducción de dimensionalidad y multicolinealidad:

Dado que el número de columnas es elevado y alguna de estas no aporta información a nuestro caso de estudio, se realiza primero la selección de los atributos de estudio dejando así un Dataframe que sea más reducido en cuanto a sus atributos de partida para poder trabajar el análisis de datos con mayor facilidad.

Si las variables predictoras tienen una alta correlación entre sí, puede haber problemas de multicolinealidad. La multicolinealidad ocurre cuando hay una fuerte relación lineal entre las variables independientes, lo que puede dificultar la interpretación de los coeficientes del modelo y conducir a estimaciones inestables. En tales casos, puede ser deseable eliminar una de las variables correlacionadas o combinarlas para crear una nueva variable.

También es importante la interpretación y simplicidad; una alta correlación entre las variables predictoras puede dificultar la interpretación de los resultados. En ese caso, puede ser preferible trabajar con variables menos correlacionadas para facilitar la comprensión de cómo cada una afecta a la variable objetivo.

Como primer estudio para ver la relación de los datos y ayudarnos a tomar la decisión de los atributos a seleccionar, se observa la relación de las columnas de forma gráfica. Esto es, la dependencia que tiene una columna con respecto a las otras si estas varían.

Para este estudio, se emplea el método *corr* de Pandas. Pero, previamente se requiere codificar las variables no numéricas ya que esta función, únicamente actúa sobre las columnas numéricas, y es este momento, queremos ver la relación de todas las columnas. Para ello, se hace uso del módulo Preprocessing de scikitlearn. Con este módulo se puede hacer uso de métodos que nos permitirá codificar los atributos categóricos a numéricos.

Se utiliza el *encoder* OrdinalEncoder que transforma el valor en el ordinal para la columna, dando como resultado valores numéricos de 0 a n-1 característica.

```
# Instancia del codificador
oe = OrdinalEncoder()

# Ajuste del modelo (fit) y codificación de los datos (transform)
oe.fit(data_df)
data_encoded = oe.transform(data_df)

# Crear un dataframe auxiliar con los atributos y valores transformados
names = oe.get_feature_names_out()
df_encoded = pd.DataFrame(data_encoded, columns=names)
df_encoded
```

*Script 20 Codificación de los atributos*

```
corr = df_encoded.corr()

# Eliminar la diagonal principal y los valores debajo de esta, manteniendo solo los valores
superiores
# para ver solo valores únicos diferentes de 1 (relación con diferentes atributos)
corr_filter = corr.where(np.triu(np.ones(corr.shape), k=1).astype(np.bool_))

# Ordenar de mayor a menor
corr_sort = corr_filter.stack().sort_values(ascending=False).to_frame()

# Añadir nombres a las columnas para entender las relaciones
corr_sort = corr_sort.rename(columns=leyend_dict, index=leyend_dict)

corr_sort.head(60)
```

*Script 21 Correlación de atributos*

Si todas las variables predictoras están altamente correlacionadas entre sí pero también tienen una fuerte correlación con la variable objetivo, esto puede ser beneficioso. Significaría que todas las variables están capturando información relevante para predecir la variable objetivo. Sin embargo, si algunas variables no están correlacionadas con la variable objetivo, pero sí entre sí, es posible que esas variables no estén aportando información adicional al modelo y podrían eliminarse.

```
# Verificar los valores NaN en el dataframe
print(df_encoded.isna().sum())

# Separar las variables predictoras y la variable objetivo
predictors = df_encoded.drop('c1', axis=1)
target = df_encoded['c1']

# Inicializar y ajustar SimpleImputer para rellenar los valores faltantes
imputer = SimpleImputer(strategy='mean') # Elige una estrategia ('mean', 'median',
'most_frequent', etc.)
imputed_X = pd.DataFrame(imputer.fit_transform(predictors), columns=predictors.columns)

# Verificar si se han rellenado los valores NaN
print(imputed_X.isna().sum())

# Aplicar la selección de características en el conjunto de datos imputado
k = 50
selector = SelectKBest(score_func=f_regression, k=k)
selector.fit(imputed_X, target)

# Obtener las puntuaciones y características seleccionadas
feature_scores = pd.DataFrame({'Feature': imputed_X.columns, 'Score': selector.scores_})
feature_scores = feature_scores.sort_values(by='Score', ascending=False).reset_index(drop=True)
selected_features = feature_scores.nlargest(k, 'Score')['Feature'].values

# Mostrar las características seleccionadas y sus puntuaciones
print("Características seleccionadas:")
for i, feature in enumerate(selected_features):
    score = feature_scores.loc[feature_scores['Feature'] == feature, 'Score'].values[0]
    print(f"{i+1}. {feature} - Score: {score}")
```

Script 22 Selección de atributos

Hay variables con alta correlación con la variable objetivo por lo que podemos tener una idea de que variables son potencialmente interesantes.

### Selección de atributos

Si las variables están altamente correlacionadas y una de ellas falta o se modifica, es más probable que otras variables correlacionadas puedan proporcionar información similar. Esto puede hacer que tu modelo sea más robusto y estable en presencia de cambios o ruido en los datos.

De estas 50 variables con alta relación con la variable objetivo se escogerá teniendo en cuenta que se requiere realizar una predicción en relación con los datos de la tripulación, pasajeros y algunas características adicionales relacionadas.

Esta lista de variables con gran correlación sobre la variable objetivo se debe filtrar las variables que tienen una alta correlación entre sí pero no con la variable objetivo ya que como se menciona anteriormente pueden ser redundantes y no aporta ningún beneficio extra.

### Análisis conjunto de datos

Para tener un contexto de los datos que se tiene, se realiza una serie de visualización de datos.

La primera de estas acciones consiste en describir los datos que tenemos, en su tipo y cuántos de estos datos son válidos, o al menos, no desconocidos (NaN).

```
#Pasamos valores extraños a np.nan
cols=["c1","c7","c144","c106","c108",
       "c101","c109","c151","c35",
       "c117","c128","c156","c41","c30","c49","c51",
       "c62","c65","c10","c31","c56"]

df.replace(['<NA>', '<NaN>', 'nan ', ' nan', '$!', "$", "!"], np.nan, inplace=True)

for col in cols:
    df[col] = df[col].fillna(np.nan)
```

*Script 23 Análisis de datos*

De forma gráfica, y desde el concepto de porcentaje de valores NaN en las columnas, se muestra el siguiente gráfico.

```
def plot_nan(df):
    """
    Imprimir el porcentaje de valores faltantes
    """
    def df_info_perc(df) -> dict:
        """
        Obtener el porcentaje de datos faltantes para plotear
        """
        # Diccionario con datos
        info_dic = {}
        # Número de datos faltantes para las diferentes columnas
        rows = len(df)
        # print('\tCOLUMN\tNAN\tPERC')
        for col in df:
            nan_values = df[col].isna().sum()
            perc = round((nan_values/rows) * 100, 2)
            # print(f'[+]\t{col}\t{nan_values}\t{perc}%')

            # Almacenar datos descriptivos
            info_dic.update({col: perc})

        return info_dic
    # Describir los datos faltantes
    info = df_info_perc(df)

    # Mostrar información
    plt.figure(figsize=(10, 5))
    plt.bar(info.keys(), info.values())
    plt.xlabel('Atributos')
    plt.xticks(rotation=45)
    plt.ylabel('% NaN')
    plt.title('Valores NaN')

    plt.show()

plot_nan(df)
```

*Script 24 Ploteo de atributos NaN*

Se puede observar que en algunas columnas hay bastantes datos faltantes.

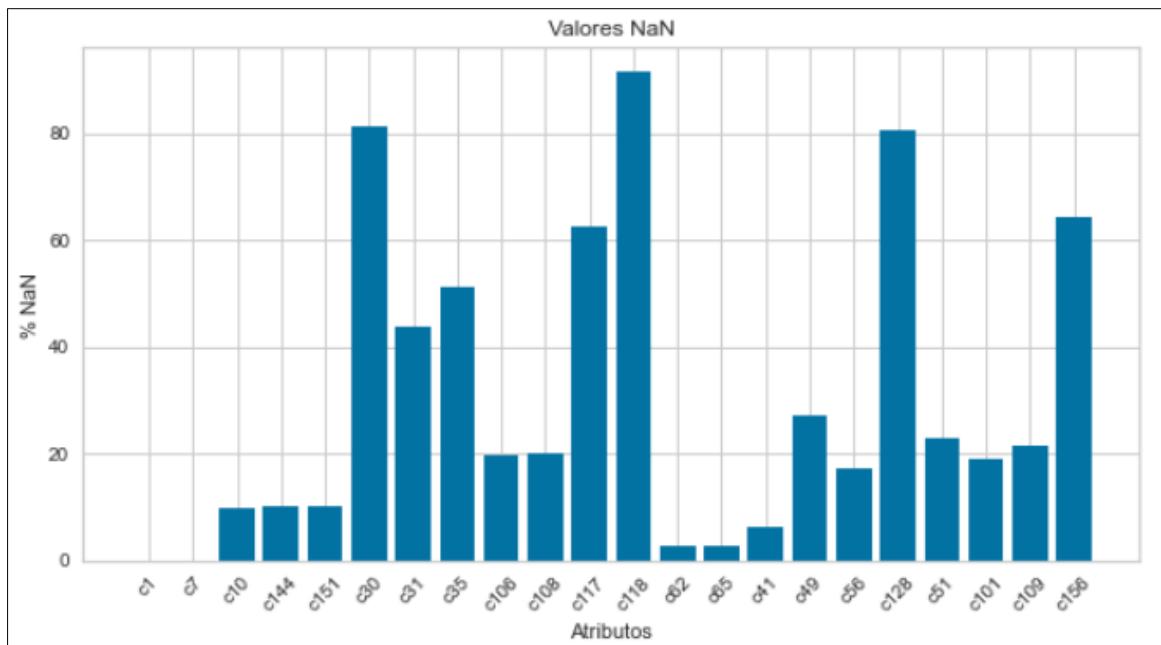
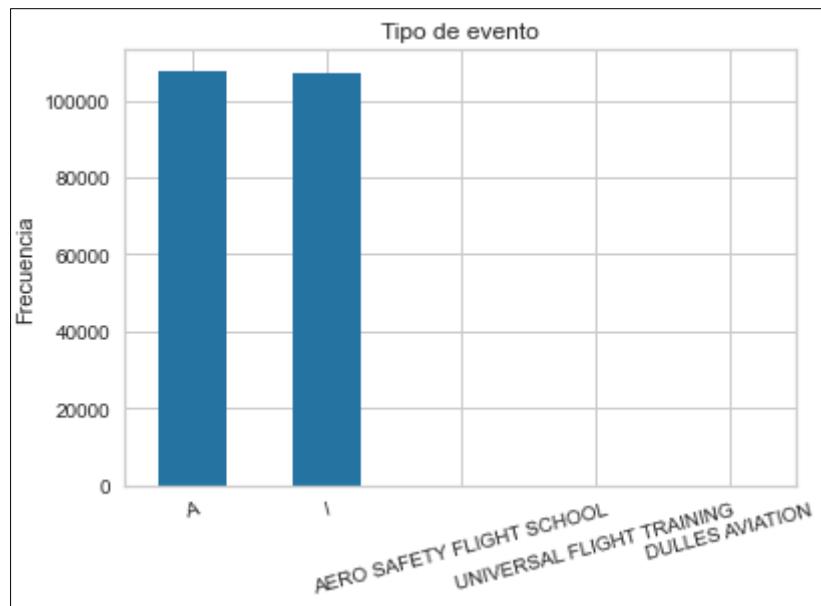


Ilustración 16 Valores NaN

El modelo que se desea realizar, debe de ser capaz de predecir el tipo de incidente que ocurrirá. Para ver si tenemos unos datos balanceados, a continuación, se muestra una gráfica de la predicción a realizar, donde podemos observar que si existe un balance de datos.

```
...  
ax = df.c1.value_counts().plot(kind='bar', title='Tipo de evento')  
  
ax.set_ylabel('Frecuencia')  
  
# Rotar nombres de atributos  
for label in ax.xaxis.get_ticklabels():  
    label.set_rotation(15)  
  
plt.show()
```

Script 25 Balanceo de clases

*Ilustración 17 Balanceo de clases*

A continuación, se muestra el balanceo de datos para la columna del mes del incidente. Teniendo en cuenta el tipo de análisis a realizar, se realiza una previa visualización de la distribución de los sucesos por esta característica.

```
# Obtener la serie de conteos y ordenar los indices alfabéticamente
serie_counts = df.c7.value_counts().sort_index()

# Reordenar los valores de la serie según el nuevo orden de los indices
serie_counts = serie_counts.reindex(serie_counts.index.sort_values())

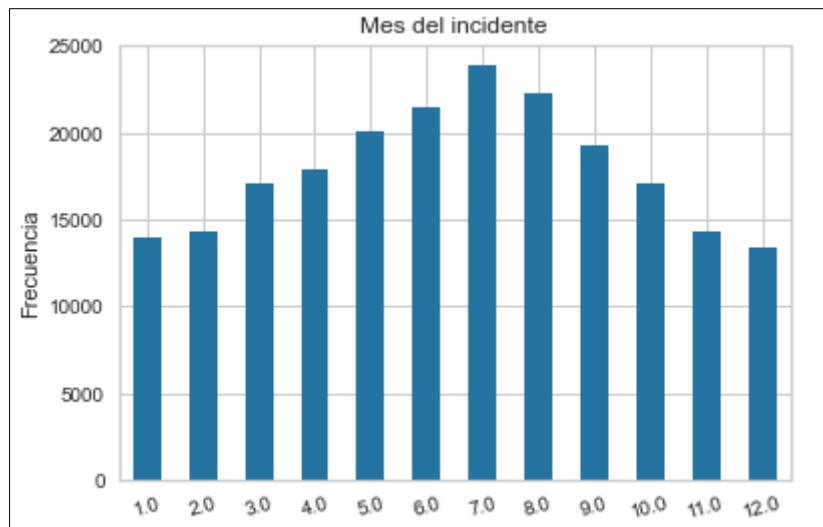
# Generar el gráfico de barras
ax = serie_counts.plot(kind='bar', title='Mes del incidente')

ax.set_ylabel('Frecuencia')

# Rotar nombres de atributos
for label in ax.xaxis.get_ticklabels():
    label.set_rotation(15)

plt.show()
```

*Script 26 Balanceo de clases*



Script 27 Balanceo de clases

### Descripción de valores

Dado el elevado número de datos faltantes y sabiendo que existe valores que están en columnas diferentes por el estudio realizado para obtener el Dataset de trabajo. A continuación, se detalla los valores que existe para cada uno de los atributos del modelo de datos obtenido.

Como se puede observar, existe la mitad de columnas que contiene más del 50% de datos nulos o vacíos que habrá que tener en cuenta a la hora de la selección de características para nuestro estudio. Además de valores atípicos para su columna por encontrarse desplazados.

Se observa que las clases de estudio (Accidente o Incidente) están balanceadas

Por último, analizando el tipo de datos para el Dataframe, se observa que la gran mayoría de atributos hace referencia a tipo objeto. Se debe a que determinados de estos atributos, tiene datos categóricos que mezcla números y letras en sus datos. No obstante, se analiza más adelante cuál de estas son realmente este tipo de columnas y cuáles son fruto de un error en los datos.

#### 1.3.3 Data Preprocessing(DP)

Con las conclusiones extraídas en el apartado anterior, en este apartado se realiza las correcciones necesarias para corregir estos errores.

### Eliminar características

La columna c118 tiene demasiados valores nulos, por lo que una imputación de valores podría afectar negativamente a la predicción.

### Valores desplazados o atípicos

A continuación, se define los valores que ha de tener cada una de las columnas categóricas para encontrar en que otras columnas puede existir y así poder relocalizarlas. Esta búsqueda se

realiza sobre todo el dataset, de este modo, se podrá importar esos valores a las columnas del dataset de estudio.

## Análisis

Vamos a contar los valores de las columnas que a priori parecen categóricas para poder identificar algunos valores que no deberían estar en determinadas columnas debido a un desplazamiento, generando un valor atípico.

```
● ● ●

# Lista de columnas categoricas a recorrer
columnas = ["c1","c7","c144",
            "c106","c108",
            "c101","c109","c151","c35","c49","c51",
            "c117","c128",
            "c156","c41","c30"] # Agrega aquí todas las columnas que deseas recorrer

for columna in columnas:
    # Obtener los valores repetidos y contar su frecuencia
    repeated_values = df[columna].value_counts()

    # Crear un DataFrame con los valores repetidos y su frecuencia
    repeated_values_df = repeated_values.to_frame()
    repeated_values_df.columns = ['Frequency']

    # Ordenar por las categorías menos frecuentes
    repeated_values_df = repeated_values_df.sort_values(by='Frequency')

    # Imprimir el nombre de la columna
    print("Columna:", columna)

    # Imprimir el DataFrame
    print(repeated_values_df)
    print() # Agregar una linea en blanco después de cada columna
```

Script 28 Análisis de valores

## c41

c42 parece tener valores mal introducidos que hay que solventar

```
● ● ●

df['c41'] = df['c41'].replace({'9': '09', '3': '03', '1': '01', '0': '00'})
conteo_valores = df['c41'].value_counts()
conteo_valores
```

Script 29 Análisis c41

## c1, c106, c108

c1, c106 y c108 presenta valores que son poco comunes ya que solo se repiten una vez y esos valores no concuerdan con el resto por lo que procedemos a la eliminación.

```
...  
  
df['c41'] = df['c41'].replace({'9': '09', '3': '03', '1': '01', '0': '00'})  
conteo_valores = df['c41'].value_counts()  
conteo_valores
```

Script 30 Análisis c42

### c144, c35, c122, c128

Estas columnas requieren de observación más detallada ya que su lista de valores es de mayor tamaño.

```
...  
  
# Lista de columnas a recorrer  
columnas = ["c144","c35","c128"] # Agrega aquí todas las columnas que deseas recorrer  
  
for columna in columnas:  
    # Obtener los valores repetidos y contar su frecuencia  
    repeated_values = df[columna].value_counts()  
  
    # Crear un DataFrame con los valores repetidos y su frecuencia  
    repeated_values_df = repeated_values.to_frame()  
    repeated_values_df.columns = ['Frequency']  
  
    # Ordenar por las categorías menos frecuentes  
    repeated_values_df = repeated_values_df.sort_values(by='Frequency').head(50)  
  
    # Imprimir el nombre de la columna  
    print("Columna:", columna)  
  
    # Imprimir el DataFrame  
    print(repeated_values_df)  
    print() # Agregar una línea en blanco después de cada columna
```

Script 31 Análisis c144, c35, c122, c128

Tras ver los resultados, se puede apreciar que estas columnas no presentan ninguna alteración.

### Acciones

Tras poderse comprobar que c1, c106 y c108 presentan valores atípicos se procede a borrar las filas que alteran la variable ya que son pocos registros los afectados y no alteraría la predicción.

### Variables numéricas.

Tras la comprobación no aparenta tener datos atípicos o registros desplazados

```

● ● ●

# Lista de columnas a recorrer
columnas = ["c62","c65","c10","c31","c56"]

for columna in columnas:
    # Obtener los valores repetidos y contar su frecuencia
    repeated_values = df[columna].value_counts()

    # Crear un DataFrame con los valores repetidos y su frecuencia
    repeated_values_df = repeated_values.to_frame()
    repeated_values_df.columns = ['Frequency']

    # Ordenar por las categorías menos frecuentes
    repeated_values_df = repeated_values_df.sort_values(by='Frequency').head(50)

    # Imprimir el nombre de la columna
    print("Columna:", columna)

    # Imprimir el DataFrame
    print(repeated_values_df)
    print() # Agregar una línea en blanco después de cada columna

```

*Script 32 Análisis c62, c65, c10, c31, c56*

## Formato de datos

Ya tenemos comprobado el desplazamiento y valores atípicos, por lo que el siguiente paso sería la verificación y adecuación de los datos para que pueda ser interpretado correctamente por el algoritmo de predicción.

**NOTA:** Los datos aquí indicados como categóricos o numéricos, no son necesariamente de dicha índole para el estudio del modelo, esta agrupación se ha realizado por tipo de datos contenido, pero no por significado para el modelo final.

### c7 Month the accident/incident happened

```

● ● ●

col = 'c7'

# Observar los valores de la columna
print(f"[+] {col}\t{col_name(col)}")
print(f"VALUES\t{df[col].unique()}\n")
type(df['c7'].unique()[12])

df[col].replace(pd.NaT, np.nan, inplace=True)
print(f"VALUES\t{df[col].unique()}\n")

```

*Script 33 Formateo c7*

c10 Local time of the accident/incident

```
...  
  
col = 'c10'  
  
# Observar los valores de la columna  
print(f"[+] {col}\t{col_name(col)}")  
print(f"VALUES\t{df[col].unique()}\n")
```

Script 34 Formateo c10

La columna c10 contiene los datos de la hora del incidente, pero para poder lidiar con todos los datos, se convierte los valores dados en formato HHMM en HH. Para ello, se recorre todos los valores y se modifica su valor, además, si los minutos son más de 30, se sumará una hora al resultado.

Se toma esta decisión para no sobreponer al modelo de características para este atributo.

```
...  
  
def get_hour(value):  
    """  
    Convertir la hora en HHMM a HH  
    """  
    try:  
        if value in[None, 'nan']:  
            return np.NaN  
  
        # Extraer la hora y minuto  
        hour = int(value[:2])  
        # Comprobar que la hora sea correcta y no sea un valor  
        # ilógico  
        if hour >= 24:  
            return np.NaN  
        minute = int(value[2:])  
        hour += 1 if minute > 30 else 0  
        # Cambiar la hora a 00 para aquellas que sumen 1 y sea las 23  
        hour = 0 if hour == 24 else hour  
  
        return str(hour).zfill(2)  
  
    except (TypeError, Exception):  
        return np.NaN  
  
    # Utilizar compresión de listas y aplicar el filtrado de valores  
hours = [get_hour(value) for value in df['c10']]  
df['c10'] = hours  
  
df.c10.unique()
```

Script 35 Conversión horaria

**Conversión de columnas mal identificadas como string a numéricas.**

Iterar las columnas y mostrar los valores únicos y ver si son numéricas o string.

Podemos apreciar la cantidad de valores únicos de cada columna.

Vamos a establecer que los que tengan más de 100 valores únicos son numéricos y los restantes categóricos.

Separar variables tipo float y almacenar variables categóricas en una lista.

```
# Separar variables tipo float y almacenar variables categóricas en una lista
str_cols = []
num_cols = []

columnas = ["c1","c7","c144",
            "c106","c108",
            "c101","c109","c151","c35",
            "c117","c128",
            "c156","c41","c49","c51",
            "c30","c62","c65","c10","c31","c56"]

for columna in columnas:
    if df[columna].dtype == 'object':
        str_cols.append(columna)
    else:
        num_cols.append(columna)

print("Columnas tipo string:", str_cols)
print("Columnas tipo numérica:", num_cols)
```

*Script 36 Clasificación de atributos*

Se va establecer como criterio de clasificación las que superen 100 caracteres únicos serán numéricas las restantes serán categóricas

```

● ● ●

#Vamos a establecer como criterio de clasificación las que superen 100 caracteres únicos
#serán numéricas
#las restantes serán categóricas

# Definimos una lista para almacenar las columnas numéricas
num_cols_cat = []

# Definimos una lista para almacenar las columnas categóricas
num_cols_ncat = []

# Verificamos la cardinalidad de cada columna
for col in num_cols:
    unique_vals = df[col].nunique()
    if unique_vals >= 100:
        num_cols_ncat.append(col)
    else:
        num_cols_cat.append(col)

# Imprimimos las columnas numéricas y categóricas
print("Columnas numéricas no categóricas:", num_cols_ncat)
print("Columnas categóricas numéricas:", num_cols_cat)
print("Columnas categóricas string:", str_cols)

```

*Script 37 Clasificación de atributos*

La columna c65 debe ser numérica no categórica, debido a su naturaleza, es decir, hace referencia a la cantidad de tripulantes.

### Imputación de valores

Primeramente, debe de aislarse la variable objetivo de los datos a imputar ya que puede perjudicar gravemente a la predicción ya que los datos imputados pueden generar una relación muy fuerte con la variable generando ruido que el algoritmo de predicción puede mal interpretar.

```

● ● ●

df = df.dropna(subset=['c1'])
#Extracción de variable objetivo para no interferir en la imputación
df_raw = df
df_raw = df_raw.drop('c1', axis=1)

```

*Script 38 Eliminación clases objetivo*

Imputamos con IterativeImputer de valores de columnas numéricas

```
...  
  
# Imputación para columnas numéricas no categóricas  
num_imputer = IterativeImputer()  
  
# Imputar las columnas numéricas no categóricas  
df_raw[num_cols_ncat] = num_imputer.fit_transform(df_raw[num_cols_ncat])
```

*Script 39 Imputación iterativa*

Ahora se va a imputar las columnas tipo string para ello se utilizará KnnImputer y se codificará los datos mediante LabelEncoder, almacenando los valores originales para poder decodificarlos tras la imputación.

```
...  
  
#df_raw = df  
str_cols=['c10','c144', 'c106', 'c108', 'c101', 'c109', 'c35', 'c117','c128', 'c156',  
'c41', 'c30','c49','c51']  
# Eliminar la columna c1 de df_raw  
#df_raw = df_raw.drop('c1', axis=1)  
  
# 1st possibility  
label_encoder = LabelEncoder() # Crear una instancia del codificador  
  
# Ajustar el codificador a los datos  
df_temp = df_raw.astype("str").apply(label_encoder.fit_transform)  
df_final = df_temp.where(~df_raw.isna(), df_raw)
```

*Script 40 Preparación datos*

```
...  
  
# Imputación de valores faltantes para las columnas categóricas mediante KNNImputer  
imputer = KNNImputer(n_neighbors=1)  
df_final[str_cols] = imputer.fit_transform(df_final[str_cols])
```

*Script 41 Imputación KNN*

Ahora tras la imputación, se realiza la decodificación.

```
...  
  
# Realizar la decodificación manualmente  
df_decoded = df_final.copy()  
for column in df_decoded.columns:  
    mapping = {label: original for label, original in zip(df_temp[column].unique(),  
    df_raw[column].unique())}  
    df_decoded[column] = df_decoded[column].map(mapping)
```

*Script 42 Decodificación de valores*

Los valores que no han podido imputarse se eliminarán dada la baja repercusión que tendrían sobre el Dataset. Además, al Dataset imputado se le debe de volver a unir la variable objetivo.

```
...  
  
# Find rows with null values  
null_rows = df_decoded[df_decoded.isnull().any(axis=1)]  
  
# Print rows with null values  
print(null_rows)  
  
# Unir la columna c1 al DataFrame resultante  
df_decoded['c1'] = df['c1']
```

Script 43 Añadir clase objetivo

Tras la unión c1 tendrá registros que no se han borrado y que deben borrarse para que no tenga valores NaN.

```
...  
  
df = df_decoded.dropna()  
# Find rows with null values  
null_rows = df[df.isnull().any(axis=1)]  
str_cols=['c1','c10','c144', 'c106', 'c108', 'c101', 'c109', 'c35', 'c117','c128', 'c156',  
'c41', 'c30']
```

Script 44 Eliminación NaN

## Conversión de datos

Antes de continuar después de la imputación debemos de formatear la columna c7 como string.

```
...  
  
# Datos de la columna  
df['c7'].unique()  
# Convertir los datos a entero y posteriormente a tipo string.  
df['c7'] = df['c7'].astype(int).astype(str)  
df['c7'].unique()
```

Script 45 Formateo c7

### 1.3.4 Feature Engineering (FE)

A continuación, se realiza una serie de modificaciones en los datos para que estos puedan ser interpretados por el modelo.

## Codificación de datos categóricos

Existe una serie de columnas con datos categóricos que se deberá de codificar para poder tratar sus datos y realizar nuestra predicción del modelo.

Hasta ahora se ha trabajado con las columnas categóricas y numéricas en base a sus datos, pero, para realizar la codificación necesaria, se va a realizar sobre todas las columnas que necesariamente serán categóricas para realizar el modelo.

Existen tres técnicas de Data Encoding dependiendo de las características de los atributos:

-OrdinalEncoder: Se utiliza para codificar variables categóricas que no tienen una relación de orden natural. Devuelve una única columna que va de 1 a n\_caracteristicas-1.

-LabelEncoder: Se utiliza para codificar variables categóricas que tienen una relación de orden natural. Devuelve una columna de 1 a n\_caracteristicas-1. Su uso se realiza principalmente para transformar la variable dependiente.

-OneHotEncoder: Se utiliza para codificar variables categóricas que no tienen una relación de orden natural. Devuelve una columna para cada atributo codificado, indicando 0 o 1 según la presencia de la característica para cada registro.

A continuación, se muestra los tipos de datos categóricos del Dataset para seleccionar que técnica emplear con cada atributo.

Para la variable dependiente u objetivo, c1, así como para los atributos de entrada cuyos datos tienen un orden de relación, se emplea LabelEncoder, col\_le. Para los atributos cuyas clases no están relacionadas, se emplea OrdinalEncoder. Para el resto de columnas se ha realizado un OrdinalEncoder, col\_oe.

```
df['c65'] = df['c65'].astype(int)
df['c151'] = df['c151'].astype(int)
num_cols_ncat= ['c62', 'c31', 'c56','c65']
str_cols=['c1', 'c10','c7', 'c151','c144', 'c106', 'c108',  'c101', 'c109', 'c35',
'c117','c128', 'c156', 'c41', 'c30','c49','c51']
# LabelEncoder
col_le = ["c1"]
# OrdinalEncoder
col_oe = ['c7', 'c151','c10','c144', 'c106', 'c108', 'c101', 'c109', 'c35', 'c117','c128',
'c156', 'c41', 'c30','c49','c51']
```

Script 46 Clasificación codificadores

### LabelEncoder

Para la transformación de los datos, se guarda el diccionario de los datos que se va a mapear para poder emplearlo en la entrada de nuevos datos.

```
# Instancia del codificador
le = LabelEncoder()

mapped_dic = {}
for col in col_le:
    # Ajuste de los datos y transformación
    df[col] = le.fit_transform(df[col])
    # Añadir valores al diccionario
    mapped_dic.update({col: dict(zip(le.classes_, le.classes_))})

mapped_le = pd.DataFrame.from_dict(mapped_dic, orient='index').fillna('-')
mapped_le
```

Script 47 LabelEncoder

### OrdinalEncoder

```
# Instancia del codificador
oe = OrdinalEncoder()

cat_cols_mapped = []
for col in col_oe:
    df[col] = oe.fit_transform(df[[col]])
    categories = list(oe.categories_[0])

    # Añadir valores al diccionario
    mapped_dic.update({col: {idx: cat for idx, cat in enumerate(categories)}})

    # Añadir columnas a lista de columnas transformadas
    cat_cols_mapped.append(col)

mapped_df = pd.DataFrame.from_dict(mapped_dic, orient='index').fillna('-')
mapped_df
```

Script 48 OrdinalEncoder

```
# Almacenar diccionario en .csv para su posterior uso
mapped_df.to_csv(f'{model_info_dir}mapped_dictionary.csv', index=True)
```

Script 49 Guardado csv

### Normalización atributos numéricos

En base a la desviación estándar de determinadas columnas, que son elevadas, se realiza el normalizado de los datos y el escalado entre 0 y 1.

```
# Columnas a aplicar diferentes métodos
std_cols = ['c62', 'c31', 'c56','c65']
#min_max_cols = ['c20', 'c240', 'c241']

# Instancia de los normalizados
std_scaler = StandardScaler(with_mean=False)
#min_max_scaler = MinMaxScaler()

# Ajuste de las instancias a los datos y guardado de estos
std_scaler.fit(df[std_cols])
#min_max_scaler.fit(df[min_max_cols])

joblib.dump(std_scaler, f'{model_info_dir}scaler_{"_".join(std_cols)}.joblib')
#joblib.dump(min_max_scaler, f'{model_info_dir}scaler_{"_".join(min_max_cols)}.joblib')
#scaler_new = joblib.load('scaler.joblib')
#df['c31'] = scaler_new.transform(df[['c31']])

# Transformación de los datos
df[std_cols] = std_scaler.transform(df[std_cols])
#df[min_max_cols] = min_max_scaler.transform(df[min_max_cols])

df.describe()
```

Script 50 Estandarizado valores

### 1.3.5 Data Modeling

Crear el modelo de predicción con el conjunto de datos obtenido.

#### División de los datos

Para el modelado, se emplea unos datos de entrenamiento (train) y unos de prueba (test). Una vez el modelo está realizado, existe los datos de validación, estos no son vistos por el modelo durante el train o test.

A continuación, se parte los datos en:

**data**: datos con los que se entrenará y testeará el modelo.

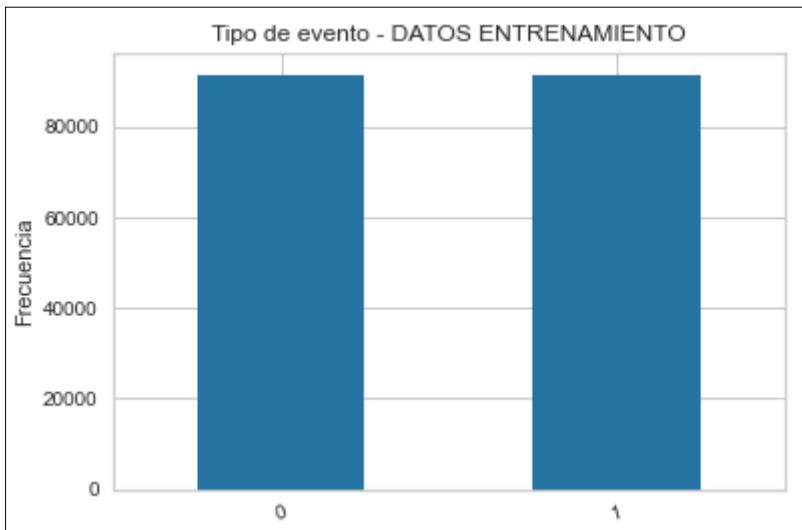
**data\_unseen**: datos que se emplearán para la validación (5% de los datos).

```
## sample devuelve una muestra aleatoria de un eje del objeto
data = df.sample(frac=0.95, random_state=786)
data
```

Script 51 Creación data

```
...  
  
ax = data.c1.value_counts().plot(kind='bar', title='Tipo de evento - DATOS ENTRENAMIENTO')  
  
ax.set_ylabel('Frecuencia')  
  
# Rotar nombres de atributos  
for label in ax.xaxis.get_ticklabels():  
    label.set_rotation(15)  
  
plt.show()
```

Script 52 Distribución de los datos



Script 53 Distribución de los datos

```
...  
  
# eliminamos del conjunto de datos original estos datos aleatorios para la validación  
data_unseen = df.drop(data.index)  
data_unseen
```

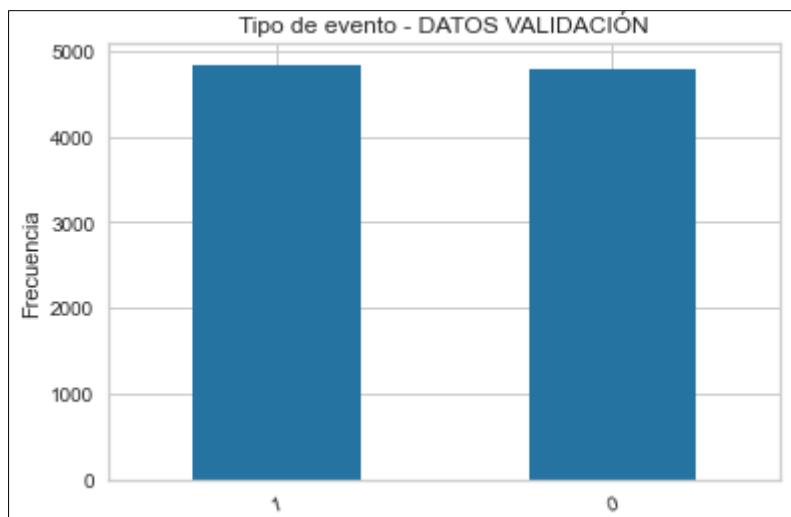
Script 54 DataUnseen

```
# Balanceo de los datos para la validación
ax = data_unseen.c1.value_counts().plot(kind='bar', title='Tipo de evento - DATOS
VALIDACIÓN')

ax.set_ylabel('Frecuencia')

# Rotar nombres de atributos
for label in ax.xaxis.get_ticklabels():
    label.set_rotation(15)

plt.show()
```

*Script 55 Distribución de los datos**Script 56 Distribución de los datos*

#### 1.4 Airplane(PABLO)

##### 1.4.1 Obtención de datos

##### 1.4.2 Exploratory Data Analysis (EDA)

##### 1.4.3 Data Preprocessing (DP)

##### 1.4.4 Feature Engineering (FE)

##### 1.4.5 Data Modeling

## 2 Generar el modelo

### 2.1 Forecast

#### 2.1.1 PyCaret SETUP

Para la creación del modelo, se hace uso del framework PyCaret por su sencillez y potencia de uso una vez ya realizado el preprocesado de los datos.

El primero paso a realizar con el framework es indicar la configuración de este para la creación del modelo. Esta configuración se realiza de acuerdo con la documentación del propio framework, (PyCaret, s.f.)

- **fold\_strategy:** Permite seleccionar una técnica de validación cruzada.
- **preprocess:** Existe la posibilidad de introducir datos ya preprocesados y por lo tanto indicar únicamente el pipeline de transformaciones que se desea (valor a False). O, por el contrario, que PyCaret preprocese los datos para realizar imputaciones, codificación de características...
- **pca:** Se permite el uso de reducción de dimensionalidad con el uso de Principal Component Analysis.
- **low\_variance\_threshold:** Ignorar varianzas bajas en aquellos casos en los que existe una característica dominante sobre otras relacionadas y no existe mucha variación en la información de estas características.
- **remove\_multicollinearity:** Permite lidiar con las características que están fuertemente ligadas con otras del Dataset.
- **fix\_imbalance:** Cuando existe un conjunto de datos desbalanceado, se permite balancear este mediante diferentes técnicas.
- **remove\_outliers:** Eliminación de outliers en el conjunto de los datos.

Además, es necesario indicar también cual es la variable objetivo y de los valores de entrada, o inputs, cuáles de ellos se debe considerar como numéricos y cuáles como categóricos.

Teniendo en cuenta la correlación de los datos y la distribución de los valores para el conjunto de datos de entrenamiento y test mencionado previamente, la configuración final para la creación del modelo es la siguiente:

```

model_setup = setup(
    use_gpu=True,
    target='c1',
    verbose=True,
    n_jobs=4,
    session_id=2023051,
    # Validación cruzada
    fold_strategy = 'kfold', fold = 10, fold_shuffle=True,
    # Indicar datos
    data=data,
    categorical_features=cat_cols_mapped,
    numeric_features=num_cols_ncat,
    # Pipeline de transformaciones
    preprocess=False, # custom_pipeline=transformer_pipeline,
    # Análisis de componentes principales
    pca = True, pca_method='incremental', pca_components=20,
    # Ignorar varianzas bajas
    low_variance_threshold=None,
    # Correlación de variables
    remove_multicollinearity=True, # multicollinearity_threshold=0.95,
    # Desbalanceo
    fix_imbalance=False,
    # Outliers
    remove_outliers=False)

```

Script 57 Configuración PyCaret Forecast

	Description	Value
0	Session id	2023051
1	Target	c1
2	Target type	Binary
3	Original data shape	(182876, 21)
4	Transformed data shape	(182876, 21)
5	Transformed train set shape	(128013, 21)
6	Transformed test set shape	(54863, 21)
7	Numeric features	4
8	Categorical features	16

Ilustración 18 Datos de configuración

### 2.1.2 Comparación de modelos

El siguiente paso con el framework de PyCaret consiste en la creación del modelo, pero previo a ello, se puede indicar que modelo se quiere emplear, de este modo, se toma mayor control sobre la creación automatizada del modelo.

Para ello, se realiza una comparación de una lista de modelos disponibles y compatibles con el conjunto de datos indicados en la configuración. Es decir, de forma automática PyCaret selecciona los modelos para el tipo de predicción a realizar.

Igual que con la configuración, es posible indicar una serie de parámetros para realizar las comparaciones, pues este proceso, no deja de ser un paso en el que PyCaret entrena y evalúa un modelo.

```
compare_models(
    cross_validation=True,
    include=test_models_id,
    probability_threshold=0.5,
    sort='Recall')
```

*Script 58 Comparación de modelos*

Como parámetros de configuración, se indica una lista de modelos que ha sido creada previamente con los modelos disponibles que tienen la característica de ejecutarse en modo *Turbo*, además, con el parámetro **sort**, los resultados se ordenan en base a la estadística “*Recall*” (*Sensibilidad*).

La ordenación en función a este parámetro se debe a que, debido a la importancia de la predicción, es necesario escoger un modelo cuya proporción de verdaderos positivos en relación con el total de muestras positivas en el conjunto de datos sea lo más elevada posible.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
<b>xgboost</b>	Extreme Gradient Boosting	0.8142	0.9078	0.8352	0.8011	0.8178	0.6284	0.6290	1.8720
<b>rf</b>	Random Forest Classifier	0.8094	0.9035	0.8349	0.7940	0.8139	0.6188	0.6196	6.3120
<b>lightgbm</b>	Light Gradient Boosting Machine	0.8099	0.9038	0.8300	0.7974	0.8134	0.6197	0.6203	2.2660
<b>et</b>	Extra Trees Classifier	0.8020	0.8960	0.8245	0.7885	0.8061	0.6040	0.6047	5.3200
<b>gbc</b>	Gradient Boosting Classifier	0.7897	0.8867	0.8100	0.7779	0.7936	0.5794	0.5799	16.4440
<b>ridge</b>	Ridge Classifier	0.7571	0.0000	0.7877	0.7418	0.7640	0.5143	0.5153	1.4880
<b>lda</b>	Linear Discriminant Analysis	0.7571	0.8309	0.7877	0.7418	0.7640	0.5143	0.5153	1.6050
<b>knn</b>	K Neighbors Classifier	0.7452	0.8203	0.7740	0.7313	0.7520	0.4904	0.4912	2.5460
<b>ada</b>	Ada Boost Classifier	0.7639	0.8578	0.7705	0.7599	0.7652	0.5278	0.5279	5.5330
<b>dt</b>	Decision Tree Classifier	0.7514	0.7514	0.7514	0.7509	0.7511	0.5028	0.5029	2.3500
<b>lr</b>	Logistic Regression	0.7139	0.7960	0.7116	0.7143	0.7129	0.4277	0.4278	5.2370
<b>svm</b>	SVM - Linear Kernel	0.6842	0.0000	0.6990	0.7258	0.6656	0.3688	0.4130	4.7750
<b>qda</b>	Quadratic Discriminant Analysis	0.7063	0.8328	0.5016	0.8479	0.6303	0.4122	0.4515	1.4460
<b>nb</b>	Naive Bayes	0.6902	0.7858	0.4717	0.8362	0.6032	0.3799	0.4220	1.4840
<b>dummy</b>	Dummy Classifier	0.5008	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	1.4100

*Script 59 Resultados de la comparación*

### 2.1.3 Creación del modelo

Tras los resultados obtenidos en la comparación de modelos, el que finalmente es escogido para crear un único modelo y ajustarlo posteriormente para mejorar su rendimiento, es el modelo **Extreme Gradient Boosting**.

Para la creación, se realiza varios modelos cuya tasa de aprendizaje sea variable, de este modo, se permite elegir cuál es el que mejor comportamiento puede tener.

```

models = []
results = []

for i in np.arange(0.1,1,0.1):
    model = create_model('xgboost',
                          cross_validation=True,
                          learning_rate = i,
                          verbose=False)
    model_results = pull().loc[['Mean']]
    models.append(model)
    results.append(model_results)

results = pd.concat(results, axis=0)
results.index = np.arange(0.1,1,0.1)
results.plot()

```

*Script 60 Comparación tasas de aprendizaje*

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
<b>0.100000</b>	0.806400	0.901600	0.827900	0.793200	0.810200	0.612700	0.613300
<b>0.200000</b>	0.812900	0.907000	0.835000	0.799200	0.816700	0.625700	0.626400
<b>0.300000</b>	0.814200	0.907800	0.835200	0.801100	0.817800	0.628400	0.629000
<b>0.400000</b>	0.814900	0.908100	0.832600	0.803700	0.817900	0.629800	0.630200
<b>0.500000</b>	0.813600	0.906800	0.831500	0.802300	0.816700	0.627200	0.627600
<b>0.600000</b>	0.810400	0.904600	0.826400	0.800400	0.813200	0.620900	0.621200
<b>0.700000</b>	0.809500	0.902600	0.825900	0.799200	0.812300	0.619100	0.619400
<b>0.800000</b>	0.806200	0.900000	0.821400	0.796700	0.808900	0.612400	0.612700
<b>0.900000</b>	0.805400	0.898000	0.818500	0.797200	0.807700	0.610800	0.611000

*Ilustración 19 Resultados de creación de modelos*

En base a los resultados obtenidos, Ilustración 19, se realiza la creación del modelo con el de mejor rendimiento.

#### 2.1.4 Ajustar el modelo

Para mejorar el rendimiento del modelo, es importante ajustar el mayor número de parámetros posibles. Con el método ***tune\_model*** se permite realizar esta acción de manera automática.

En este caso, se realiza el ajuste del modelo en base a la Precisión. Ya que el modelo se escoge en base a la sensibilidad, se persigue ahora mejorar la Precisión del modelo. Además, se añade el parámetro *choose\_better* de modo que se devuelva el mejor de los modelos en todos los ajustes que se realiza.

```
● ● ●

tuned_model = tune_model(model,
                         optimize='Accuracy',
                         n_iter=20,
                         choose_better=True
)
```

Script 61 Ajuste del modelo

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Fold							
<b>0</b>	0.7696	0.9016	0.9645	0.6935	0.8068	0.5395	0.5858
<b>1</b>	0.7735	0.9021	0.9623	0.6971	0.8085	0.5480	0.5917
<b>2</b>	0.7701	0.8973	0.9589	0.6984	0.8082	0.5384	0.5816
<b>3</b>	0.7727	0.9001	0.9656	0.6984	0.8105	0.5441	0.5899
<b>4</b>	0.7677	0.8994	0.9595	0.6907	0.8033	0.5373	0.5817
<b>5</b>	0.7653	0.9007	0.9644	0.6866	0.8021	0.5331	0.5808
<b>6</b>	0.7651	0.8970	0.9613	0.6914	0.8043	0.5294	0.5757
<b>7</b>	0.7671	0.8999	0.9655	0.6912	0.8056	0.5343	0.5820
<b>8</b>	0.7630	0.8964	0.9607	0.6907	0.8036	0.5241	0.5708
<b>9</b>	0.7592	0.8966	0.9581	0.6824	0.7971	0.5206	0.5673
<b>Mean</b>	0.7673	0.8991	0.9621	0.6920	0.8050	0.5349	0.5807
<b>Std</b>	0.0042	0.0020	0.0026	0.0049	0.0037	0.0080	0.0073

Ilustración 20 Resultados del modelo

Con el modelo ajustado, creado y finalizado, se realiza una predicción de este con el 5% de los datos del Dataframe que no han sido utilizados en todo el proceso de creación del modelo, es decir, con los “***data\_unseen***” que se menciona al principio del capítulo. *Ilustración 21*

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.8211	0.9127	0.8352	0.8140	0.8245	0.6421	0.6423

*Ilustración 21 Resultados predicción data\_unseen*

### 2.1.5 Guardar el modelo

Finalmente, se guarda el modelo en un archivo “.pkl” que se puede importar más adelante por la GUI de Streamlit y así poder realizar las predicciones en un entorno de producción con la introducción de nuevos datos que predecir.

```
● ● ●

# Registrar la hora actual para el guardado del modelo
date = datetime.now()
date = date.strftime('%Y%m%d_%H%M')

# Guardado del modelo
model_name = 'Forecast_model'
save_model(final_model, f'{model_info_dir}{model_name}_{date}')
```

*Script 62 Guardado del modelo*

## 2.2 Crew

### 2.2.1 PyCaret SETUP

Para la configuración de nuestro clasificador, se emplea el método `setup()` de PyCaret, en el cual se le indica aquellos parámetros necesarios para realizar los modelos:

**fold\_strategy**: Permite seleccionar una técnica de validación cruzada.

**preprocess**: Existe la posibilidad de introducir datos ya procesados y por lo tanto indicar únicamente el pipeline de transformaciones que se desea (valor a False). O, por el contrario, que PyCaret procese los datos para realizar imputaciones, codificación de características... (valor a True).

**pca**: Se permite el uso de reducción de dimensionalidad con el uso de Principal Component Analysis.

**low\_variance\_threshold**: Ignorar varianzas bajas en aquellos casos en los que existe una característica dominante sobre otras relacionadas y no existe mucha variación en la información de estas características.

**remove\_multicollinearity**: Permite lidiar con las características que están fuertemente ligadas con otras del Dataset.

**fix\_imbalance**: Cuando existe un conjunto de datos desbalanceado, se permite balancear este mediante diferentes técnicas.

**remove\_outliers**: Eliminación de outliers en el conjunto de los datos.

```
● ● ●

model_setup = setup(
    use_gpu=True,
    data=data, target='c1',
    verbose=True, n_jobs=4, session_id=2023051,
    # Validación cruzada
    fold_strategy = 'kfold', fold = 10, fold_shuffle=True,
    # Indicar datos
    categorical_features=cat_cols_mapped, numeric_features=num_cols_ncat,
    # Pipeline de transformaciones
    preprocess=False, # custom_pipeline=transformer_pipeline,
    # Análisis de componentes principales
    pca = True, pca_method='incremental', pca_components=20,
    # Ignorar varianzas bajas
    low_variance_threshold=None,
    # Correlación de variables
    remove_multicollinearity=True, # multicollinearity_threshold=0.95,
    # Desbalanceo
    fix_imbalance=False, # Cambiar a false
    # Outliers
    remove_outliers=False,
    #class_weight={0: 0.3, 1: 0.7}
)
```

Script 63 Configuración PyCaret

	Description	Value
0	Session id	2023051
1	Target	c1
2	Target type	Binary
3	Original data shape	(182876, 21)
4	Transformed data shape	(182876, 21)
5	Transformed train set shape	(128013, 21)
6	Transformed test set shape	(54863, 21)
7	Numeric features	4
8	Categorical features	16

Ilustración 22 Configuración PyCaret

### 2.2.2 Comparación de modelos

```
...  
  
test_models = models()  
test_models_id = [i[0] for i in test_models.iterrows() if i[1]["Turbo"] == True]  
print(test_models_id)  
test_models
```

*Script 64 Modelos PyCaret*

ID	Name	Reference	Turbo
lr	Logistic Regression	sklearn.linear_model.logistic.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors_classification.KNeighborsCl...	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True
dt	Decision Tree Classifier	sklearn.tree._classes.DecisionTreeClassifier	True
svm	SVM - Linear Kernel	sklearn.linear_model._stochastic_gradient.SGDC...	True
rbfsvm	SVM - Radial Kernel	sklearn.svm._classes.SVC	False
gpc	Gaussian Process Classifier	sklearn.gaussian_process._gpc.GaussianProcessC...	False
mlp	MLP Classifier	sklearn.neural_network._multilayer_perceptron....	False
ridge	Ridge Classifier	sklearn.linear_model._ridge.RidgeClassifier	True
rf	Random Forest Classifier	sklearn.ensemble._forest.RandomForestClassifier	True
qda	Quadratic Discriminant Analysis	sklearn.discriminant_analysis.QuadraticDiscrim...	True
ada	Ada Boost Classifier	sklearn.ensemble._weight_boosting.AdaBoostClas...	True
gbc	Gradient Boosting Classifier	sklearn.ensemble._gb.GradientBoostingClassifier	True
lda	Linear Discriminant Analysis	sklearn.discriminant_analysis.LinearDiscrimina...	True
et	Extra Trees Classifier	sklearn.ensemble._forest.ExtraTreesClassifier	True
xgboost	Extreme Gradient Boosting	xgboost.sklearn.XGBClassifier	True
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMClassifier	True
dummy	Dummy Classifier	sklearn.dummy.DummyClassifier	True

*Ilustración 23 Modelos PyCaret*

De los modelos disponibles, se realiza una comprobación de sus predicciones ordenados por la Sensibilidad (Recall) para poder compararlos y escoger el mejor de ellos.

Dada la importancia de la predicción, es importante escoger un modelo cuya proporción de verdaderos positivos en relación con el total de muestras positivas en el conjunto de datos sea lo más elevada posible.

```
...  
  
compare_models(cross_validation=True, include=test_models_id, probability_threshold=0.5,  
sort='Recall')
```

*Script 65 Comparación modelos*

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.8142	0.9078	0.8352	0.8011	0.8178	0.6284	0.6290	1.8720
rf	Random Forest Classifier	0.8094	0.9035	0.8349	0.7940	0.8139	0.6188	0.6196	6.3120
lightgbm	Light Gradient Boosting Machine	0.8099	0.9038	0.8300	0.7974	0.8134	0.6197	0.6203	2.2660
et	Extra Trees Classifier	0.8020	0.8960	0.8245	0.7885	0.8061	0.6040	0.6047	5.3200
gbc	Gradient Boosting Classifier	0.7897	0.8867	0.8100	0.7779	0.7936	0.5794	0.5799	16.4440
ridge	Ridge Classifier	0.7571	0.0000	0.7877	0.7418	0.7640	0.5143	0.5153	1.4880
lda	Linear Discriminant Analysis	0.7571	0.8309	0.7877	0.7418	0.7640	0.5143	0.5153	1.6050
knn	K Neighbors Classifier	0.7452	0.8203	0.7740	0.7313	0.7520	0.4904	0.4912	2.5460
ada	Ada Boost Classifier	0.7639	0.8578	0.7705	0.7599	0.7652	0.5278	0.5279	5.5330
dt	Decision Tree Classifier	0.7514	0.7514	0.7514	0.7509	0.7511	0.5028	0.5029	2.3500
lr	Logistic Regression	0.7139	0.7960	0.7116	0.7143	0.7129	0.4277	0.4278	5.2370
svm	SVM - Linear Kernel	0.6842	0.0000	0.6990	0.7258	0.6656	0.3688	0.4130	4.7750
qda	Quadratic Discriminant Analysis	0.7063	0.8328	0.5016	0.8479	0.6303	0.4122	0.4515	1.4460
nb	Naive Bayes	0.6902	0.7858	0.4717	0.8362	0.6032	0.3799	0.4220	1.4840
dummy	Dummy Classifier	0.5008	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	1.4100

Ilustración 24 Comparación modelos

Teniendo en cuenta los resultados obtenidos, en cuanto a métricas y tiempos de ejecución, se decide escoger un top 3 para crear los modelos, poder ajustarlos y finalmente realizar el modelo con el mejor de ellos.

A continuación, se muestra la comparación de los modelos, esta vez, empleando la validación cruzada con 10 folds como se ha indicado en los ajustes.

```
...  
compare_models(cross_validation=True, include=['xgboost', 'et', 'rf'],  
probability_threshold=0.5, sort='Recall')
```

Script 66 Comparación modelos

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.8142	0.9078	0.8352	0.8011	0.8178	0.6284	0.6290	1.4360
rf	Random Forest Classifier	0.8094	0.9035	0.8349	0.7940	0.8139	0.6188	0.6196	1.8570
et	Extra Trees Classifier	0.8020	0.8960	0.8245	0.7885	0.8061	0.6040	0.6047	2.1660

Ilustración 25 Comparación modelos

### 2.2.3 Creación del modelo

En base a los resultados anteriores, se escoge el modelo Extreme Gradient Boosting para realizar las predicciones.

```

models = []
results = []

for i in np.arange(0.1,1,0.1):
    model = create_model('xgboost', cross_validation=True, learning_rate = i,
    verbose=False)
    model_results = pull().loc[['Mean']]
    models.append(model)
    results.append(model_results)

results = pd.concat(results, axis=0)
results.index = np.arange(0.1,1,0.1)
results.plot()

```

Script 67 Creación modelo

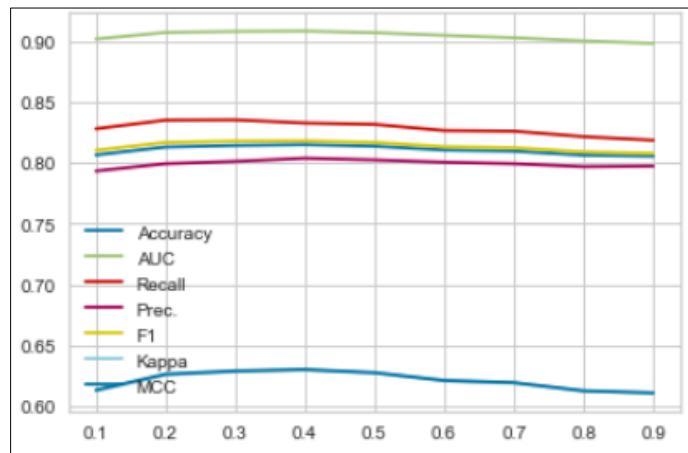


Ilustración 26 Comparación modelos

Tras la creación del modelo teniendo en cuenta su tasa de aprendizaje, se establece esta en 0.5.

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0.100000	0.806400	0.901600	0.827900	0.793200	0.810200	0.612700	0.613300
0.200000	0.812900	0.907000	0.835000	0.799200	0.816700	0.625700	0.626400
0.300000	0.814200	0.907800	0.835200	0.801100	0.817800	0.628400	0.629000
0.400000	0.814900	0.908100	0.832600	0.803700	0.817900	0.629800	0.630200
0.500000	0.813600	0.906800	0.831500	0.802300	0.816700	0.627200	0.627600
0.600000	0.810400	0.904600	0.826400	0.800400	0.813200	0.620900	0.621200
0.700000	0.809500	0.902600	0.825900	0.799200	0.812300	0.619100	0.619400
0.800000	0.806200	0.900000	0.821400	0.796700	0.808900	0.612400	0.612700
0.900000	0.805400	0.898000	0.818500	0.797200	0.807700	0.610800	0.611000

Ilustración 27 Creación modelo

```
...  
model = create_model('xgboost', cross_validation=True, learning_rate = 0.5)
```

Script 68 Creación modelo

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8171	0.9101	0.8356	0.8052	0.8202	0.6343	0.6348
1	0.8142	0.9086	0.8295	0.8032	0.8161	0.6286	0.6289
2	0.8100	0.9055	0.8220	0.8058	0.8138	0.6199	0.6201
3	0.8139	0.9081	0.8306	0.8058	0.8180	0.6277	0.6280
4	0.8131	0.9060	0.8280	0.8006	0.8141	0.6262	0.6266
5	0.8129	0.9058	0.8318	0.7975	0.8143	0.6260	0.6265
6	0.8156	0.9065	0.8343	0.8055	0.8196	0.6312	0.6316
7	0.8147	0.9079	0.8361	0.8017	0.8185	0.6294	0.6300
8	0.8117	0.9049	0.8318	0.8024	0.8168	0.6231	0.6236
9	0.8127	0.9052	0.8358	0.7954	0.8151	0.6257	0.6265
Mean	0.8136	0.9068	0.8315	0.8023	0.8167	0.6272	0.6276
Std	0.0019	0.0016	0.0041	0.0034	0.0022	0.0038	0.0039

Ilustración 28 Creación modelo

## 2.2.4 Ajustar el modelo

```
XGBClassifier(base_score=None, booster='gbtree', callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric=None, feature_types=None,
               gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=0.5, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=None, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               n_estimators=100, n_jobs=4, num_parallel_tree=None,
               objective='binary:logistic', predictor=None, ...)
```

Ilustración 29 Datos modelo

```
...  
tuned_model = tune_model(model, optimize='Accuracy', n_iter=20, choose_better=True) # ,  
custom_grid=param_grid  
tuned_model
```

Script 69 Ajuste modelo

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7696	0.9016	0.9645	0.6935	0.8068	0.5395	0.5858
1	0.7735	0.9021	0.9623	0.6971	0.8085	0.5480	0.5917
2	0.7701	0.8973	0.9589	0.6984	0.8082	0.5384	0.5816
3	0.7727	0.9001	0.9656	0.6984	0.8105	0.5441	0.5899
4	0.7677	0.8994	0.9595	0.6907	0.8033	0.5373	0.5817
5	0.7653	0.9007	0.9644	0.6866	0.8021	0.5331	0.5808
6	0.7651	0.8970	0.9613	0.6914	0.8043	0.5294	0.5757
7	0.7671	0.8999	0.9655	0.6912	0.8056	0.5343	0.5820
8	0.7630	0.8964	0.9607	0.6907	0.8036	0.5241	0.5708
9	0.7592	0.8966	0.9581	0.6824	0.7971	0.5206	0.5673
Mean	0.7673	0.8991	0.9621	0.6920	0.8050	0.5349	0.5807
Std	0.0042	0.0020	0.0026	0.0049	0.0037	0.0080	0.0073

*Ilustración 30 Ajuste de modelo*

Con el mejor de los modelos seleccionado y ajustado, se ajusta el modelo al conjunto de todos los datos, incluyendo test y train. Realizando finalmente una predicción sobre el conjunto de datos que se extrajo en un principio y no se han usado durante todo el proceso de creación del modelo.

Ya con el modelo creado, pasamos a realizar las predicciones con los datos que se ha reservado para ello.

Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0 Extreme Gradient Boosting	0.8486	0.9351	0.8657	0.8367	0.8509	0.6972	0.6976

*Ilustración 31 Predicciones*

## 2.2.5 Guardar el modelo

Tras todo el proceso de modelado y ajuste del mismo, para poder utilizar el modelo en un entorno de producción, es necesario guardar el mismo. Para ello, se emplea las propias herramientas de PyCaret.

```
# Registrar la hora actual para el guardado del modelo
date = datetime.now()
date = date.strftime('%Y%m%d_%H%M')

# Guardado del modelo
model_name = 'crew_model'
save_model(final_model, f'{model_info_dir}{model_name}_{date}')
```

*Script 70 Guardar modelo*

Ya estaría preparado para cargar y usar nuestro modelo.



*Script 71 Cargar modelo*

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.8211	0.9127	0.8352	0.8140	0.8245	0.6421	0.6423

*Ilustración 32 Predicción*

- 2.3 Airplane(PABLO)
  - 2.3.1 PyCaret SETUP
  - 2.3.2 Comparación de modelos
  - 2.3.3 Creación del modelo
  - 2.3.4 Ajustar el modelo
  - 2.3.5 Guardar el modelo

### 3 GUI: Streamlit

Para las acciones de usuario, es empleado el framework de Python **Streamlit**. Este permite crear servidores que presenten la información deseada en forma de páginas, permitiendo al usuario navegar por ellas e interactuar con los datos.

Este framework es escogido como el idóneo para crear cualquier script que trabaje con datos (Machine Learning, BigData...) en aplicaciones web fácilmente configurables y de rápido desarrollo.

Streamlit, permite realizar un desarrollo con un servidor ejecutado en local o bien en la nube gracias a su “Community Cloud”. Un servidor en la nube que, de forma completamente gratuita, permite vincular repositorios de GitHub y que las aplicaciones web creadas, estén disponibles para cualquier usuario.

#### 3.1 Directories of the Streamlit application

Para poder crear el servidor, es necesario crear un directorio con una estructura de carpetas definida.

Lo primero de todo, es necesario crear la página principal del proyecto. Esta página es la que se muestra al usuario en el momento de acceder a la aplicación. La manera de crear esta es mediante un módulo Python que se encuentra en la carpeta raíz del directorio. Para este proyecto, la página principal es el módulo “**0\_AAP\_Home.py**”.

Para poder navegar por las diferentes páginas del proyecto, Streamlit permite hacerlo de dos modos diferentes: Creación de apps o Apps multipágina.

La creación de apps, ya se ha mencionado, y sería la de crear un módulo, como en este caso *O\_AAP\_Home.py* en la que se junten diferentes objetos de vistas de Streamlit. Y todo, en un único modulo, por lo que, para aplicaciones grandes, resulta ser muy difícil poder gestionar las diferentes vistas o páginas de la aplicación. Para lidiar con ello, existe las **Multipage Apps**. Estas se organizan con, la app principal que servirá de fichero de entrada a la app y las diferentes apps organizadas en el directorio **pages/** del directorio raíz.

Para parametrizar diferentes configuraciones de la aplicación, existe el directorio **.streamlit/**, el cual incluye un fichero *config.toml*.

Finalmente, existe un directorio con los módulos para las páginas de cada modelo predictivo y un directorio común **source/** que contiene las fuentes necesarias para desarrollar cada una de las predicciones y auxiliares para la creación de las diferentes páginas.

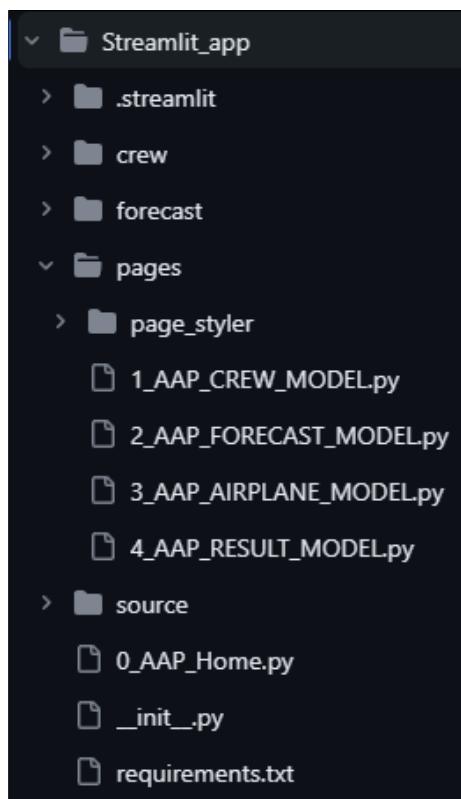


Ilustración 33 Directorios Streamlit

### 3.1.1 Directorios de modelos

Como se ha mencionado, existe los directorios para cada uno de los modelos con los módulos necesarios para el desarrollo de predicciones y muestra de datos. Estos son **airplane/**, **crew/** y **forecast/**.

Para explicar su contenido, se emplea el directorio **forecast/** ya que el resto de directorios tienen la misma estructura y fin.

Dentro de cada directorio se encuentra:

- Módulo **constants.py**

- Módulo ***model\_name.py***
- Json ***help\_columns.json***

### 3.1.1.1 Constants.py

Este módulo contiene las constantes que se emplean en el módulo del modelo.

Estas constantes son:

- SOURCE\_DIRECTORY: Ruta al directorio streamlit. Es necesario configurar esta constante ya que la ejecución del servidor en local o desde el Cloud de Streamlit, toman paths de ejecución diferente y es necesario crear una ejecución agnóstica al entorno de ejecución.
- HELP\_INFO: Ruta al json del mismo directorio.
- COL\_NAME\_DICT: Ruta al diccionario que contiene la información de las columnas, o leyenda del Dataset.
- RESULTS: Ruta al json de resultados que recibe la predicción de cada modelo para representar posteriormente de forma conjunta.
- MODEL: Ruta al modelo creado y guardado.

Adicionalmente, se añade todas aquellas constantes que sean necesarias, como las rutas a los archivos para el escalado de datos numéricos o diccionario de atributos categóricos codificados.

### 3.1.1.2 Help\_columns.json

Se trata de un fichero json el cual contiene los datos informativos, o de leyenda, para cada uno de los atributos del modelo. De esto modo, se permite un pop up de ayuda para introducir los valores correctos en los campos.



Ilustración 34 Pop-up ayuda atributos

### 3.1.1.3 Model\_name.py

Se trata del módulo principal en cada uno de los directorios para los modelos. Este, contiene la lógica para la creación de nuevas predicciones.

Para la interacción del usuario con el modelo predictivo, existe diferentes *tabs* que agrupan las características de los datos de entrada y una última *tab* que para crear una nueva predicción.

DATOS CLIMATOLOGICOS	DATOS TRIPULACIÓN	DATOS VUELO	RESULTADOS
c113 Cloud ceiling <input type="text" value="0"/>	?	c114 Visibility code <input type="text" value="1"/>	?
c241 Wind speed in miles per hours <input type="text" value="0"/>	?	c112 Sky condition code <input type="text" value="BRKN"/>	?
	- +		
		c115 Visibility restriction code. <input type="text" value="BLS"/>	?

Ilustración 35 Agrupamiento de datos de entrada y resultados

Como está estructurado el modulo es mediante el uso de una clase que se usa como instancia para crear e interactuar con los distintos elementos del framework.

La manera de interactuar con la instancia de las clases creadas, es llamar a los métodos públicos, los cuales son los que crean los campos de entrada de un modo dinámico, mediante previa parametrización de las listas de columnas numéricas y categóricas, ya que, dependiendo del tipo de dato a introducir, se necesita añadir un campo u otro de streamlit. Todas las instancias, tienen en común la tabla **Resultados** que es la encargada de evaluar los datos introducidos y generar la predicción final. Estos resultados son almacenados en un json bajo de la ruta de la constante RESULTS que posteriormente será leído por la vista de resultados de Streamlit para representar todos los modelos de forma gráfica, explicado en 3.1.2.3.1Representación gráfica de predicciones.

Existe además funciones externas a la clase que sirven para la carga de archivos de datos necesarios para la aplicación o bien realizar la predicción. Esto se debe a que dichas funciones, por trabajar con un gran volumen de datos y requerir tiempos de ejecución más altos que un proceso normal, hacen uso de cache ofrecido por streamlit con su decorador **@streamlit.cache\_data** o **@streamlit.cache\_resource**

### 3.1.2 Directorio pages

Como se presenta anteriormente, la aplicación desarrollada en el framework es una *Multi page*. La característica de estas arquitecturas es que las diferentes pestañas, se controlan en módulos independientes, y el propio streamlit los genera en su barra lateral como accesos a los diferentes módulos.



*Ilustración 36 Enlaces a diferentes páginas del proyecto*

Como refleja la Ilustración 36, existe cinco accesos en el proyecto, siendo Home la principal y el resto las secundarias. Son estas, las secundarias, las que se ha de incluir en el directorio **pages/** del proyecto. Además, el nombre a mostrar en Streamlit, es el que se le indica como nombre del módulo, como se puede apreciar. No obstante, existe una serie de consideraciones a tener en cuenta para las nomenclaturas. (Streamlit, Streamlit Documentation - Multipage apps, s.f.).

### 3.1.2.1 Subdirectorio page\_styler

Para que todas las páginas adquieran el mismo formato, este directorio contiene el modulo **page\_style.py** que contiene las clases para cada una de las vistas genéricas de la app.

Estas clases son las que configuran el texto del **sidebar** así como el texto mostrado en el contenido de la vista.



*Ilustración 37 Ajustes page styler*

### 3.1.2.2 Vistas de modelo

Para las vistas de los campos streamlit de cada modelo, existe los módulos numerados del 1 al 3.

La estructura de estos módulos es de mera pasarela con la creación de los campos, que como se menciona previamente, son creados en *3.1.1.3 Model\_name.py*.

El módulo se estructura en dos partes. Una primera que crea la vista genérica de la vista al invocarse la clase de *page\_styler*.

```
● ● ●

from pages.page_styler.page_style import ForecastSetup

# Setup de los datos de la página
ForecastSetup()
```

Script 72 Creación vista genérica

Una segunda que crea las diferentes *tabs* que se configura en el módulo de clase de cada modelo. Es decir, aquellas pestañas que sirven para interactuar con el modelo y crear nuevas predicciones.

```
● ● ●

import streamlit as st
from forecast.forecast import ForecastModel

# Instancia del modelo
forecast_model = ForecastModel()

# Tabs con las secciones para interacción con el modelo
list_of_tabs = ["DATOS CLIMATOLOGICOS", "DATOS TRIPULACIÓN",
                 "DATOS VUELO", ":blue[RESULTADOS]"]
tabs = st.tabs(list_of_tabs)

with tabs[0]:
    forecast_model.data_forecast()

with tabs[1]:
    forecast_model.data_crew()

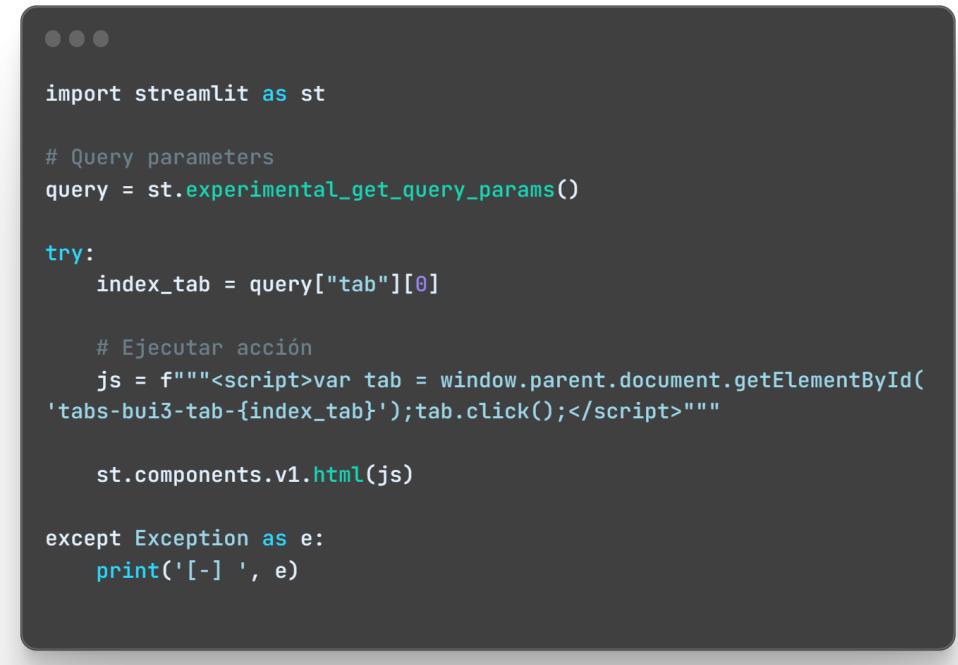
with tabs[2]:
    forecast_model.data_flight()

with tabs[3]:
    forecast_model.res_data()
```

Script 73 Creación tabs de interacción con modelo

Por último, se crea una gestión de la request de cada vista. Al tratarse de vistas independientes, estas tienen su propia url por lo que pueden recibir sus propios parámetros en la request como son por ejemplo los **queryStringParameters**.

En este punto, se evalúa la existencia del parámetro **tab** por query y con el que se indica que tab es la que se desea activar mediante la ejecución de un código en java con el módulo components de streamlit. Esta ejecución de código y activación, se realiza para poder crear enlaces a diferentes secciones de la vista desde otras vistas de la aplicación como se verá más adelante.



```
import streamlit as st

# Query parameters
query = st.experimental_get_query_params()

try:
    index_tab = query["tab"][0]

    # Ejecutar acción
    js = f"""<script>var tab = window.parent.document.getElementById(
'tabs-bui3-tab-{index_tab}');tab.click();</script>"""

    st.components.v1.html(js)

except Exception as e:
    print('[-] ', e)
```

Script 74 Gestión de queryStringParameters

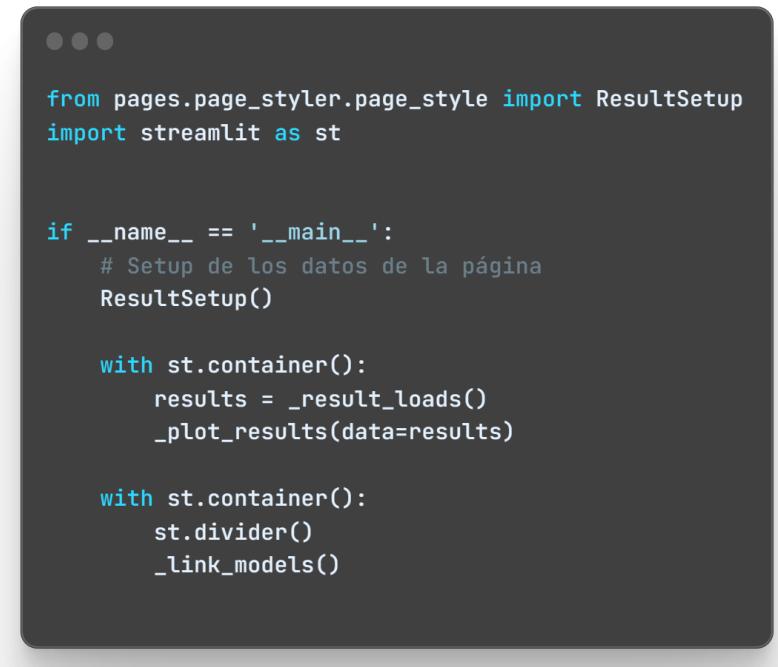
### 3.1.2.3 Vista de resultados

Este módulo es empleado para crear un gráfico de barras de los resultados de predicción de los modelos y el enlace a las pestañas de resultados de cada uno de estos, indicando el queryStringParameter “tab” como se menciona en Script 74.

Dado que se trata de una pestaña común y su función es la de agrupar los resultados, su lógica es reservada para agrupar y mostrar estos campos de streamlit, por lo que toda su lógica de creación de frontend se reduce al propio módulo.

Siguiendo una lógica de programación lineal, se detalla las partes principales del código en los siguientes scripts y subapartados.

Con la activación de la vista desde la app, en el Script 75 se crea la vista genérica al importar la clase **ResultSetup** de page\_styler como ya se ha visto previamente. Además, se añade dos elementos **container** de streamlit para agrupar en ellos el gráfico de los resultados y en el segundo, el enlace a los resultados de las predicciones.



```
from pages.page_styler.page_style import ResultSetup
import streamlit as st

if __name__ == '__main__':
    # Setup de los datos de la página
    ResultSetup()

    with st.container():
        results = _result_loads()
        _plot_results(data=results)

    with st.container():
        st.divider()
        _link_models()
```

Script 75 Creación de diseño y containers vista Results

#### 3.1.2.3.1 Representación gráfica de predicciones

En el primer container, se ejecuta dos funciones del módulo. Una primera para cargar los resultados de un json en el que cada uno de las predicciones es almacenado y posteriormente a la representación gráfica de este con el uso de matplotlib.

La lectura de las predicciones del Script 76 tiene dos funciones, una primera **\_result\_loads** que se le añade el decorador de streamlit **cache\_data** ya que se trata de una operación que, aunque no consume elevados recursos de la máquina, no se ejecutara si el resultado de las predicciones no cambia. La segunda función **\_map\_value** de este proceso, tiene la función de convertir los valores alfabéticos con los que es almacenada la información de la predicción a valores numéricos de 0 ó 1 para poder ser representado en el gráfico.

```
import json

def _map_value(value: str) -> int:
    """
    Obtener los resultados para cada uno de los modelos en formato numérico.
    Donde 1 es Accidente y 0 no accidente
    :return:
    """
    return 1 if value == 'A' else 0

@st.cache_data
def _result_loads() -> dict:
    """
    Obtener las predicciones mapeadas a valores numéricos
    :return:
    """
    with open(RESULTS, 'r') as f:
        res = json.load(f)

    # Devolver valores en valor numérico
    return {key: _map_value(value) for key, value in res.items()}
```

Script 76 Carga de predicciones

Las predicciones previamente cargadas, son el parámetro de entrada para la función **\_plot\_results** ya que esta es la encargada de ejecutar las acciones para la representación gráfica de estos resultados.

La gráfica de barras adquiere un color predefinido en función del número de accidentes obtenido en las predicciones. Siendo 3 el mayor de los riesgos y 0 la situación ideal. Este color es almacenado en el diccionario **accidente\_data** junto con el texto a mostrar al usuario que explica el riesgo existente o no. No obstante, este color es modificable ya que se emplea en el método **\_set\_plot\_color** el elemento de streamlit **color\_picker** que permite al usuario seleccionar un color mediante una rueda de selección de color como se muestra en el Script 78.



```
def _plot_results(data: dict) -> None:
    """
    Representar los datos en formato de gráfico
    :param data:
    :return:
    """
    accident_data = _get_accident_prob(results_=data)
    color = _set_plot_color(predefined_color=accident_data["plot_color"])
    _plot_results_do(data=data, color=color)
    _text_result(data=accident_data)
```

Script 77 Representación gráfica de resultados

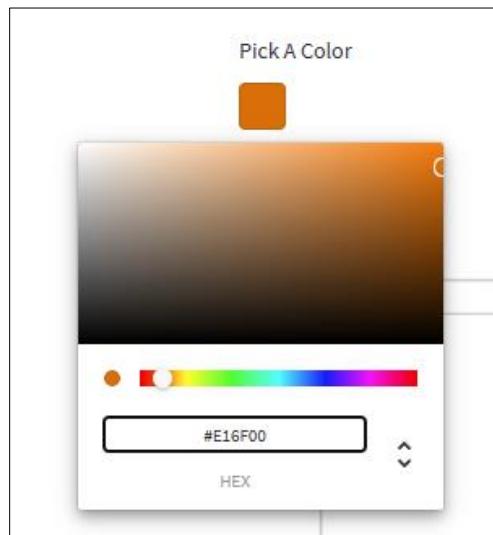


Ilustración 38 Selección de color de resultados

```
def _get_accident_prob(results_: dict):
    """
    Obtener la probabilidad de accidente
    :param results_:
    :return:
    """
    res = {"msg": '', "plot_color": ''}
    accident_counter = sum([v for k, v in results_.items()])

    if accident_counter == 0:
        res["msg"] = 'No existe probabilidad de accidente'
        res["plot_color"] = '#317f43'

    elif accident_counter == 1:
        res["msg"] = 'Probabilidad de accidente baja'
        res["plot_color"] = '#65b3ff'

    elif accident_counter == 2:
        res["msg"] = 'Probabilidad de accidente media'
        res["plot_color"] = '#e16f00'

    else:
        res["msg"] = 'Probabilidad de accidente alta'
        res["plot_color"] = '#aa1c0d'

    return res

def _set_plot_color(predefined_color: str) -> str:
    """
    Selecciona el color del grafico a gusto del usuario
    :param predefined_color:
    :return:
    """
    return st.color_picker('Pick A Color', predefined_color)
```

*Script 78 Obtención del color del gráfico*

Por último, el método `_plot_result_do` crea el gráfico con la librería matplotlib con los datos de entrada el diccionario de predicciones por modelo y el color para el gráfico. Esta función, por tener que consumir recursos de la máquina para realizar el gráfico, añade el decorador de streamlit para almacenado del cache, de modo que, si ninguno de los parámetros de entrada cambia, no se vuelve a ejecutar la función para el ploteado. Para generar el gráfico, se emplea el método `pyplot` de streamlit. Script 79.

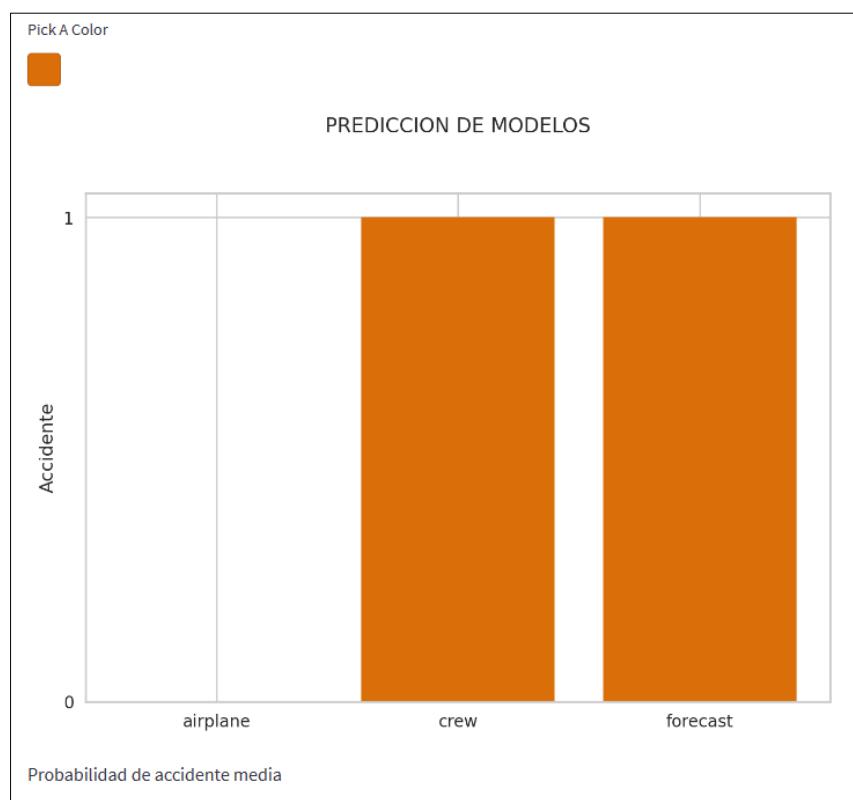
```
import streamlit as st
import matplotlib.pyplot as plt

@st.cache_resource
def _plot_results_do(data: dict, color: str) -> None:
    """
    Genera el grafico de barras con los resultados de las predicciones
    :param data:
    :param probability:
    :return:
    """
    values = [value for key, value in data.items()]

    fig, ax = plt.subplots()
    ax.bar(list(data.keys()), values, color=color)

    ax.set_ylabel("Accidente")
    ax.set_yticks([0, 1])

    ax.set_title("PREDICCION DE MODELOS", y=1.1)
    st.pyplot(fig)
```

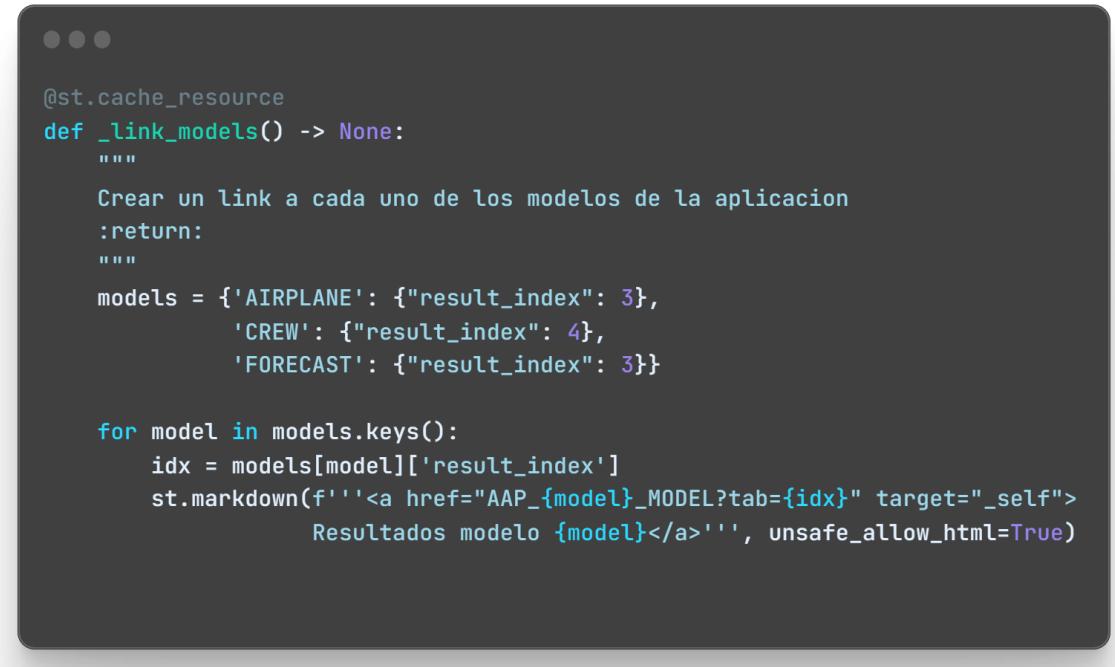
*Script 79 Creación de gráfico**Ilustración 39 Ejemplo gráfico*

### 3.1.2.3.2 Enlace a vista de resultados

Como segundo contenedor en la vista de resultados, existe los tres hipervínculos a cada uno de los resultados de los modelos.

Para ello, existe la función `_link_models` que muestra un texto HTML en pantalla. Este texto, incluye como texto “Resultados modelo <modelo>” y añade además el hipervínculo al modelo indicándolo el queryStringParameter “tab” de valor el índice de esta en la vista del modelo.

Esta función, incluye el decorador de memoria cache de streamlit ya visto. El motivo de ello es que, una vez se genera la vista de Resultado, al tratarse de datos estáticos, no es necesario volver a ejecutar la función.



```
@st.cache_resource
def _link_models() -> None:
    """
    Crear un link a cada uno de los modelos de la aplicacion
    :return:
    """
    models = {'AIRPLANE': {"result_index": 3},
              'CREW': {"result_index": 4},
              'FORECAST': {"result_index": 3}}

    for model in models.keys():
        idx = models[model]['result_index']
        st.markdown(f'''<a href="AAP_{model}_MODEL?tab={idx}" target="_self">
                    Resultados modelo {model}</a>''', unsafe_allow_html=True)
```

Script 80 Hipervínculo a predicciones

### 3.1.3 Directorio Source

En este directorio se incluye los archivos que son empleados por los módulos previamente mencionados.

En la raíz del directorio se encuentra **column\_info.csv** y **results.json**, que incluyen la leyenda de las columnas y el resultado de las predicciones respectivamente.

También se encuentra un directorio por cada modelo, en los que se incluye los archivos de estos propiamente dichos, como el modelo de predicción, diccionarios de codificación de atributos o modelos de escalado y normalizado.

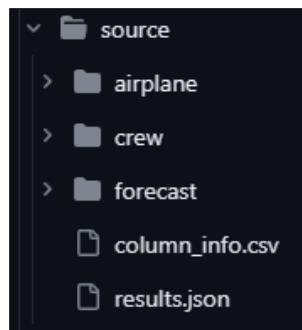


Ilustración 40 Directorio source

### 3.2 StreamlitCloud

Más allá de un framework, Streamlit es también un servicio de Cloud gratuito en el que poder almacenar las aplicaciones y gestionarlas. Para ello, se debe vincular el repositorio de la aplicación, este debe ser público, y ya está todos los pasos hechos, pues la integración se muestra muy sencilla.

Para esta aplicación de streamlit, existe dos aplicaciones gestionadas en StreamlitCloud para un mismo repositorio GitHub. Esto se debe a que el repositorio está gestionado con diferentes ramas, en las que existe las ramas *main\_new* y *feat/streamlit*, entre otras. Con esto, es posible dirigir la fuente de cada aplicación al de una rama, y de este modo, se consigue tener una aplicación en un entorno de producción y otra en un entorno de pruebas, que es de uso privado, respectivamente.

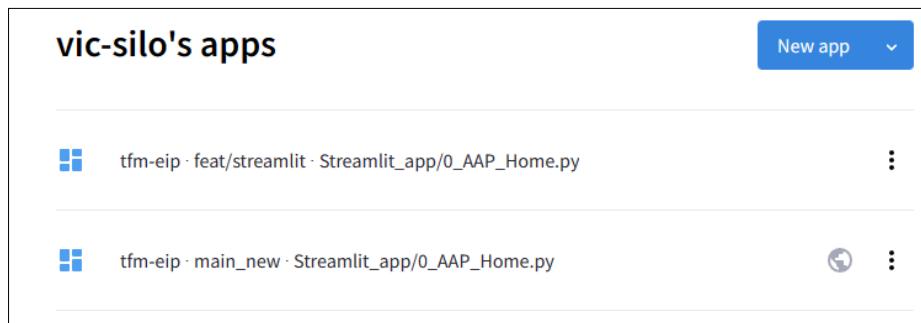


Ilustración 41 Aplicaciones streamlit

### Deploy an app

Repository [Paste GitHub URL](#)  
vic-silo/repo

Branch  
master

Main file path  
streamlit\_app.py

App URL (Optional)  
.streamlit.app

[Advanced settings...](#)

**Deploy!**

Ilustración 42 Creación de nueva aplicación

#### 4 Unión de las partes y prueba en local

## Evaluación de los resultados

En primer lugar, se realizó un análisis exploratorio detallado de los datos de accidentes de aviones. A través de este análisis, se identificaron patrones y tendencias significativas en los datos, lo que permitió comprender mejor la naturaleza de los accidentes y los factores que podrían contribuir a su ocurrencia.

Posteriormente, se procedió al preprocesamiento de los datos, abordando cuestiones como la limpieza de datos, el manejo de valores faltantes y la normalización de variables. Este proceso fue fundamental para garantizar la calidad y consistencia de los datos utilizados en los modelos de predicción.

Se construyeron modelos de predicción individuales para cada una de las agrupaciones de columnas de datos (tripulación, aeronave y datos climatológicos). Se aplicaron diversos algoritmos de aprendizaje automático para entrenar y ajustar estos modelos, con el objetivo de predecir la probabilidad de accidentes de aviones.

A través de la implementación de una interfaz de usuario utilizando Streamlit, se permitió a los usuarios introducir los datos relevantes en cada uno de los modelos de predicción. Esto facilitó la interacción con el sistema y brindó una experiencia amigable al usuario.

Al integrar los resultados de los modelos individuales y realizar un análisis global, se obtuvieron conclusiones significativas sobre la probabilidad de ocurrencia de accidentes de aviones. Se pudo establecer una visión más precisa y confiable al considerar la combinación de los resultados individuales. Esto permitió identificar patrones comunes y factores de riesgo que influyen en la probabilidad de accidentes.

En resumen, los resultados obtenidos de nuestro TFM demostraron la eficacia de los modelos de predicción construidos utilizando técnicas de aprendizaje automático y datos relevantes. La integración de estos modelos y el análisis global de los resultados proporcionaron una comprensión más profunda de la probabilidad de ocurrencia de accidentes de aviones, lo que contribuye a la mejora de la seguridad en la industria de la aviación. Los aspectos destacados de esta evaluación incluyen la identificación de patrones, la validación de los modelos y la identificación de factores clave para la prevención de accidentes. Estos hallazgos son fundamentales para tomar medidas preventivas y mejorar la seguridad en el sector aeronáutico.

## Conclusiones

Durante la investigación, hemos desarrollado con éxito un sistema de predicción de accidentes aéreos utilizando métodos de aprendizaje automático y datos relevantes. El sistema es una herramienta eficaz para evaluar la probabilidad de accidentes y ayuda a mejorar la seguridad de la industria de la aviación.

Realizamos un análisis exploratorio detallado de los datos de accidentes de aeronaves para identificar patrones y tendencias importantes. Esto ha llevado a una mejor comprensión de los factores que influyen en los accidentes, proporcionando una base sólida para medidas de prevención más eficaces.

El preprocesamiento completo de los datos es esencial para garantizar la calidad y la consistencia de los datos. Mediante el uso de tareas de limpieza, manejo de valores perdidos y normalización de variables, mejoramos la precisión del modelo predicho y aumentamos la confiabilidad de los resultados obtenidos. Se ha demostrado que los modelos predictivos que construimos para cada conjunto de columnas de datos (tripulación, aeronave y datos meteorológicos) son efectivos para predecir la probabilidad de un accidente de aeronave individual.

Este enfoque permite una comprensión detallada de los factores específicos que pueden influir en la ocurrencia de accidentes, lo cual es esencial para la prevención y la toma de decisiones informadas. La implementación de una interfaz de usuario intuitiva con Streamlit mejora la interacción con el sistema y facilita su uso a los usuarios. Esto facilita el ingreso de los datos apropiados en cada modelo de predicción y mejora la experiencia general del usuario al usar nuestro sistema.

Al integrar los resultados individuales de los modelos de predicción y analizarlos globalmente, obtenemos una comprensión más precisa y confiable de la probabilidad de accidentes aéreos. Esta evaluación conjunta tiene en cuenta la combinación de resultados individuales, lo que nos permite determinar la importancia de múltiples factores y obtener una evaluación más precisa de los riesgos asociados.

Con el TFM hemos desarrollado con éxito un sistema de predicción de accidentes aéreos que ayuda a mejorar la seguridad en la industria aeronáutica. Nuestros resultados confirman la validez de los modelos predictivos, la relevancia de los datos utilizados y la importancia de aplicar técnicas de aprendizaje automático para tomar decisiones informadas en seguridad aérea.

## Referencias

- DRUMOND, C. (n.d.). *Agile Project Management - What is it and how to get started?* Retrieved from Agile Project Management - What is it and how to get started?: <https://www.atlassian.com/agile/project-management>
- PyCaret. (n.d.). *PyCaret docs.* Retrieved from <https://pycaret.gitbook.io/docs/>
- Streamlit. (n.d.). *Streamlit Cloud.* Retrieved from <https://streamlit.io/cloud>
- Streamlit. (n.d.). *Streamlit Documentation - Multipage apps.* Retrieved from Streamlit Documentation - Multipage apps: <https://docs.streamlit.io/library/get-started/multipage-apps#how-pages-are-labeled-and-sorted-in-the-ui>
- U.S. Department of Transportation. (n.d.). *Federal Aviation Administration.* Retrieved from [https://av-info.faa.gov/dd\\_sublevel.asp?Folder=%5CAID](https://av-info.faa.gov/dd_sublevel.asp?Folder=%5CAID)
- U.S. Department of Transportation. (n.d.). *Federal Aviation Administration.* Retrieved from <https://av-info.faa.gov/data/AID/AIDCODES.DOC>

**Anexos**

Información complementaria al TFM:

Anexo A ..... AIDCODES.doc

Anexo B ..... <https://tfm-eip-aap-models.streamlit.app/>

Anexo C ..... <https://github.com/Vic-silo/TFM-EIP.git>