Kaggle Titanic – Machine Learning from Disaster



*“God himself could not sink this ship.” — Cal*

**Fecha:** 06/10/2023  
**Equipo formativo:**  
Pablo Marián Vasco  
Alejandro Jiménez  
Jorge Alba  
Daniel Gómez  
Alejandro Díez

# **Enunciado**

**Aplicación Frontend con Python y Tkinter para visualizar KPIs del Titanic**

**Objetivo:** Desarrollar una aplicación frontend utilizando Python, Tkinter, Pandas y Matplotlib para visualizar KPIs clave relacionados con el dataset del Titanic de Kaggle.

**Requerimientos:**

1. **Página Principal:** La aplicación debe iniciar mostrando una ventana principal con un título que diga "KPIs del Titanic" y un botón que diga "Ver KPIs".
2. **Visualización de KPIs:** Al hacer clic en el botón "Ver KPIs", se debe mostrar una nueva ventana que contenga gráficos generados con Matplotlib relacionados con el dataset del Titanic. Algunos KPIs sugeridos incluyen:
   * Porcentaje de supervivientes por género.
   * Porcentaje de supervivientes por clase de cabina.
   * Distribución de edades de los pasajeros.
3. **Interacción:** La ventana de visualización debe permitir al usuario regresar a la página principal o cerrar la aplicación.
4. **Manejo de Datos:** Debes utilizar Pandas para cargar y procesar el dataset en formato CSV del Titanic.

**Arquitectura de la Aplicación:**

1. **Frontend:** Se utilizará Tkinter para crear la interfaz gráfica de la aplicación, mostrando ventanas y botones al usuario.
2. **Procesamiento de Datos:** Se utilizará Pandas para cargar el archivo CSV, procesar los datos y calcular los KPIs.
3. **Visualización:** Con Matplotlib, se generarán los gráficos basados en los KPIs calculados, que luego se mostrarán en la interfaz de Tkinter.

**Instrucciones:**

1. Configurar el entorno de desarrollo instalando las bibliotecas necesarias: Tkinter, Pandas y Matplotlib.
2. Cargar el dataset del Titanic en formato CSV utilizando Pandas.
3. Diseñar la ventana principal de la aplicación con Tkinter, agregando el título y el botón "Ver KPIs".
4. Al hacer clic en "Ver KPIs", generar gráficos basados en el dataset usando **Matplotlib**. Estos gráficos deben representar los KPIs mencionados anteriormente u otros que consideres relevantes.
5. Mostrar estos gráficos en una nueva ventana de Tkinter.
6. Asegurarse de que la aplicación permita al usuario regresar a la página principal o cerrarla completamente.
7. Realizar pruebas para asegurarse de que la aplicación funciona correctamente y los KPIs se muestran de manera adecuada.

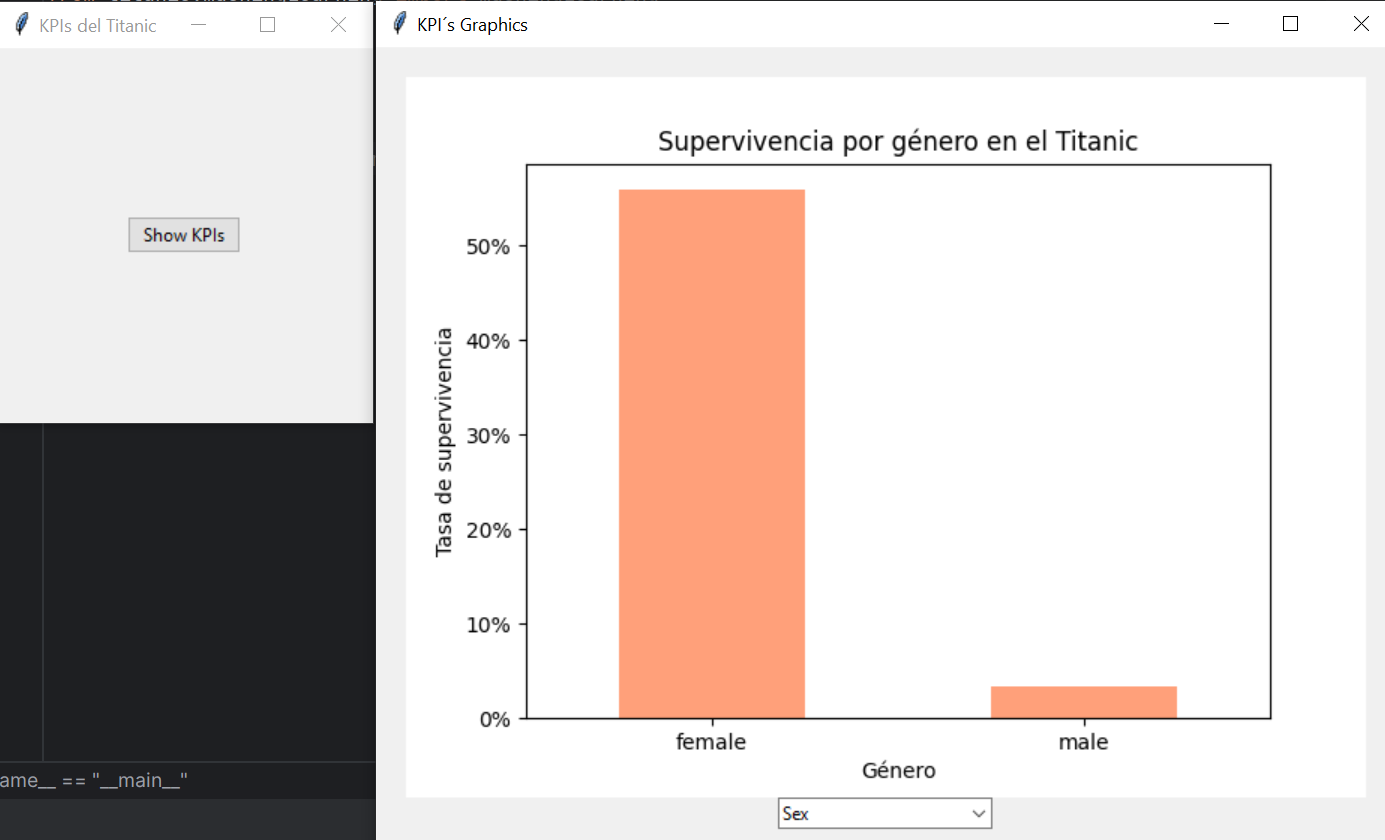
**Entrega:** Presentar el código fuente de la aplicación, junto con capturas de pantalla que demuestren su correcto funcionamiento y un breve informe sobre los KPIs visualizados y cualquier desafío o aspecto interesante que hayáis encontrado durante el desarrollo. Enviármelo a mi correo [daniel.exposito@habber.com](mailto:daniel.exposito@habber.com) en un zip. Y si no os deja, lo subís a OneDrive y lo compartís conmigo.

## **KPIs Visualizados**

### **Supervivencia por género**

La mayoría de supervivientes en el Titanic fueron mujeres, por lo cual el género (Sex) es un valor clave a la hora de analizar los datos. La fórmula para calcular este KPI es la siguiente:

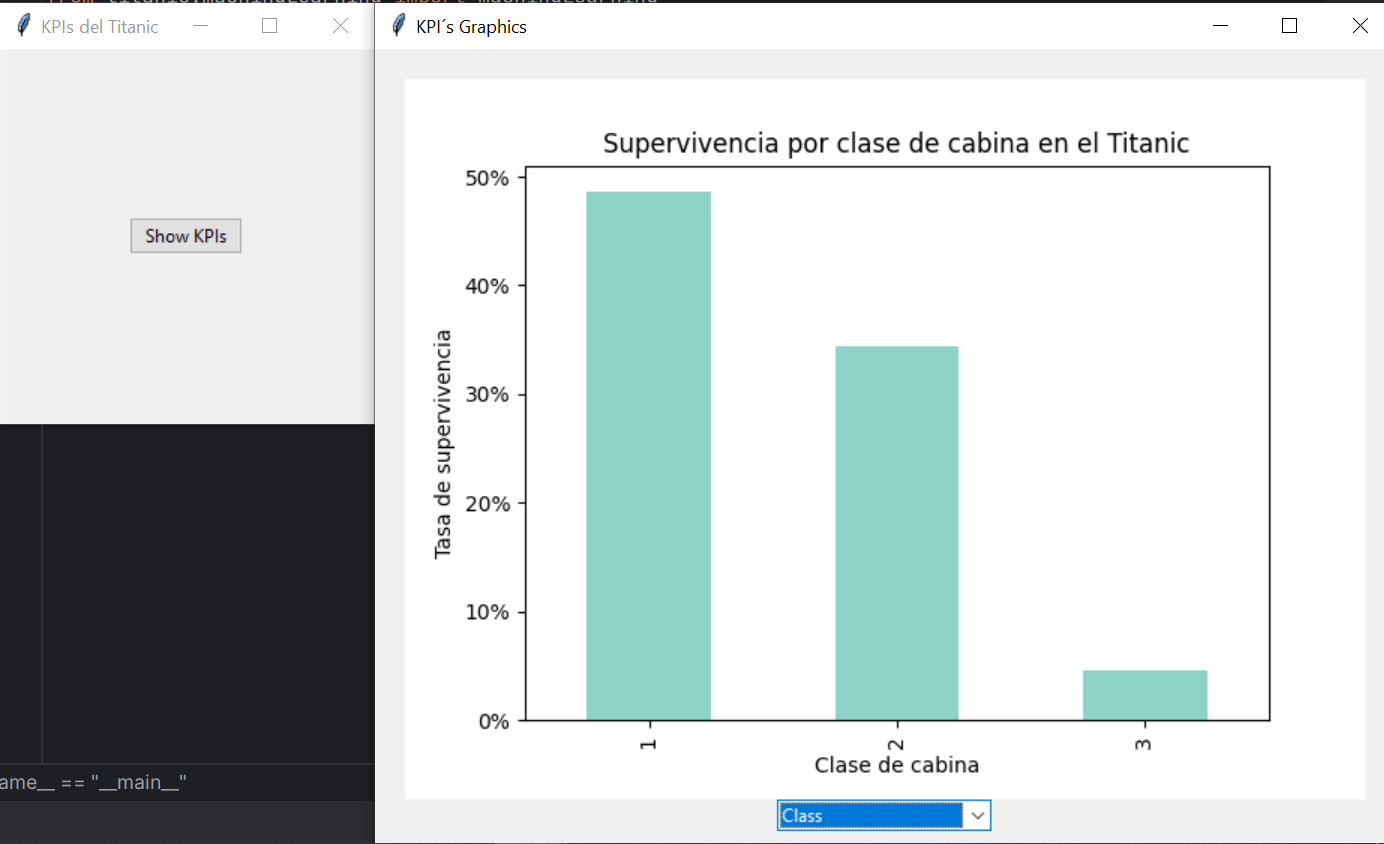
En este caso:



### **Supervivencia por clase de cabina**

Al igual que el género la clase de cabina en la que viajaba el pasajero (Pclass) es crucial para determinar si un pasajero sobrevivió o no, ya que a mayor clase social y mayor poder adquisitivo, mayor es la prioridad a la hora de salvarse. Se puede observar una gran diferencia entre la 1ª y la 3ª clase en la siguiente gráfica. La fórmula para calcular este KPI es la siguiente:

Un ejemplo en este caso:

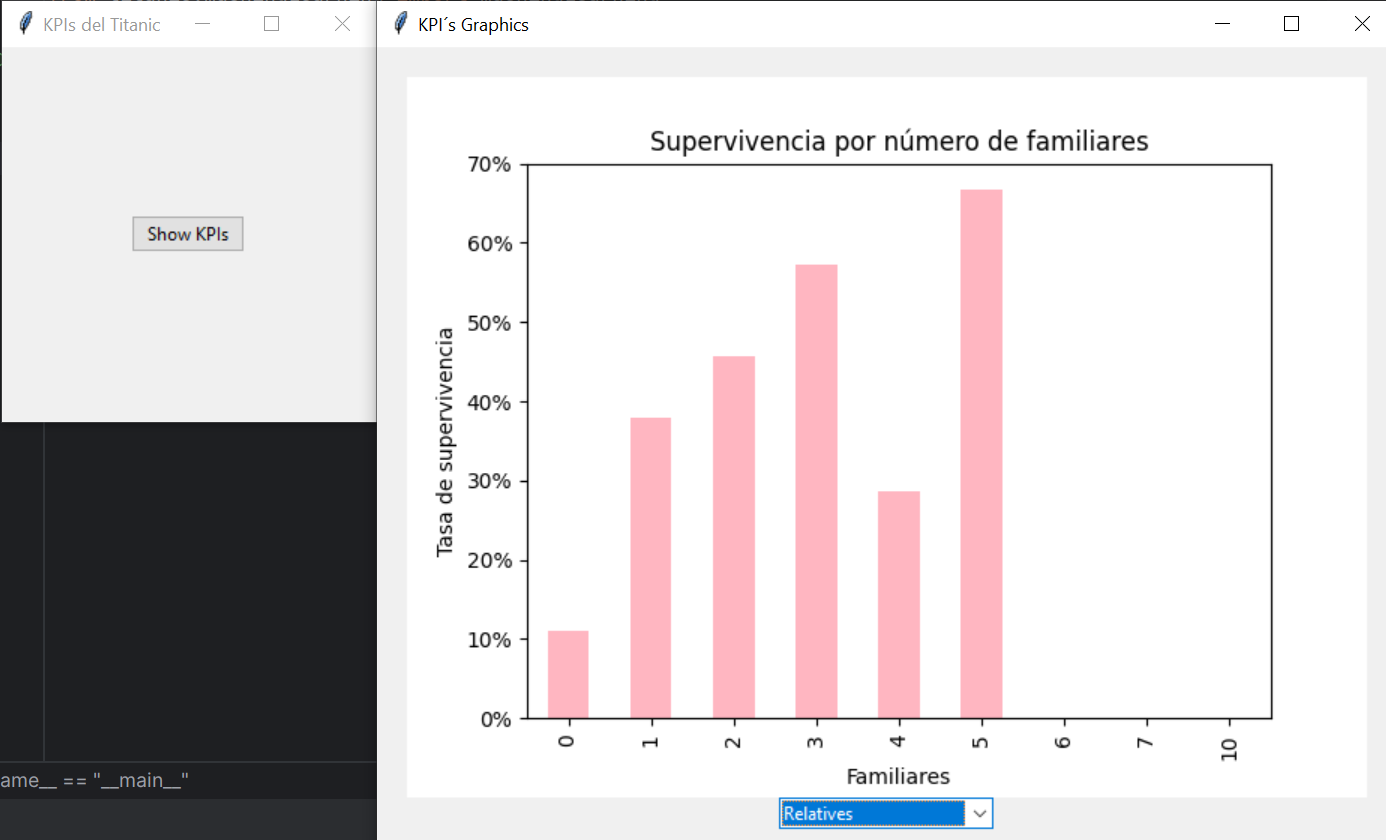


### **Supervivencia por número de familiares**

Hemos pensado que sería interesante investigar la correlación que existe entre SibSp (Siblings/Spouses), número de hermanos/cónyuges a bordo; y Parch (Parents/Children), número de padres/hijos a bordo; comparada con la supervivencia de cada pasajero en concreto; es decir, la supervivencia de un pasajero en concreto en función del número de familiares a bordo del Titanic. Dicho mal y pronto, si las familias se salvan. La fórmula utilizada para realizar esta investigación ha sido la siguiente:

Un ejemplo para este caso:

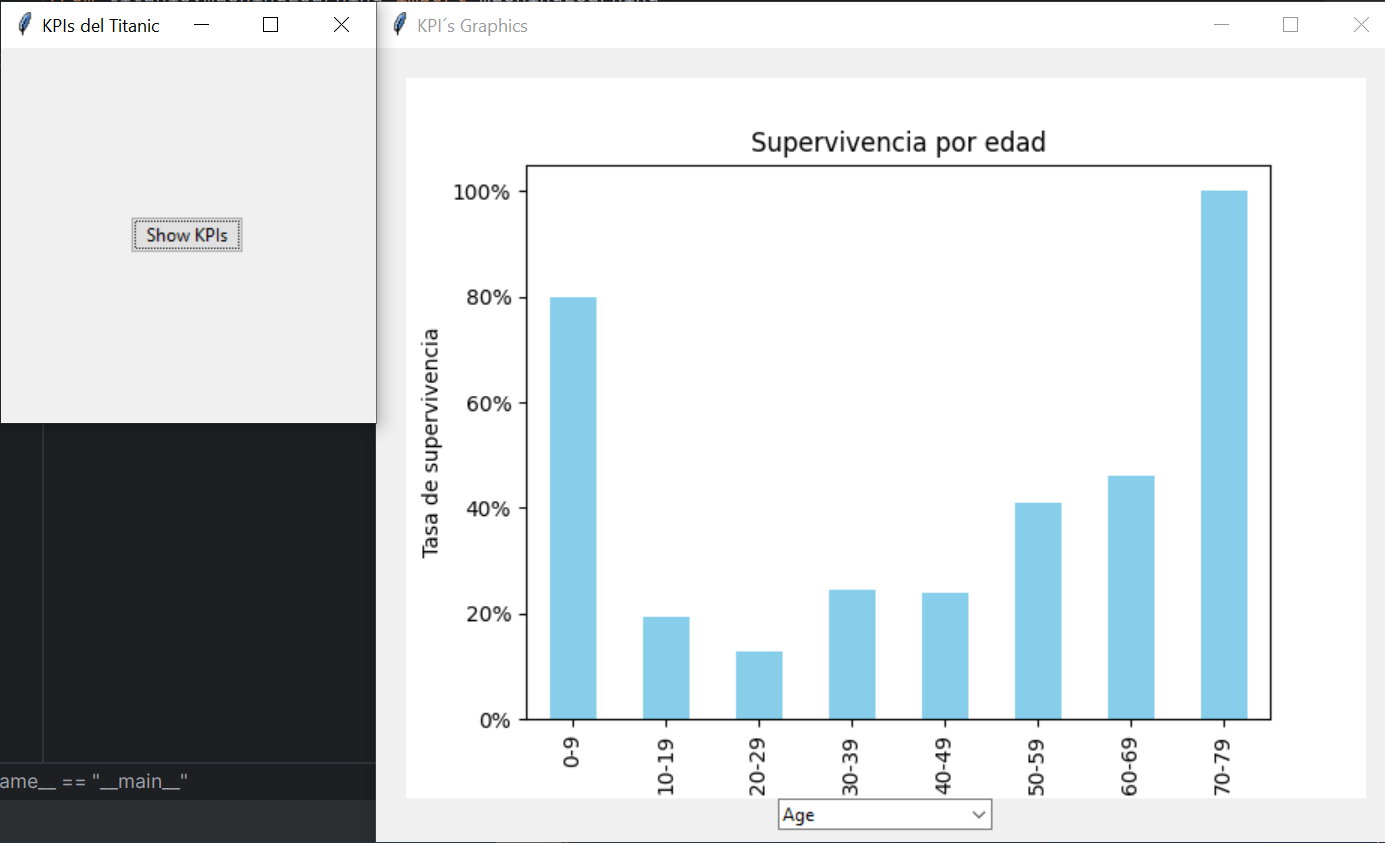
No consideramos este indicador un KPI, ya que necesitaríamos tratar estos datos más a fondo. Pensamos que a la hora del hundimiento se dio prioridad a ancianos, mujeres e hijos; esto se puede ver claramente en los KPI de género y edades. Al parecer tener familiares o ser una familia a bordo a priori no parece tan relevante y parece estar englobado por los KPIs anteriormente mencionados.



### **Supervivencia por edades**

En el siguiente gráfico podemos ver como la edad es un atributo clave para poder predecir la supervivencia de los pasajeros. Existe un alto porcentaje de supervivencia en niños (0-9 años) y en personas mayores (70-79 años) mientras que el porcentaje más bajo de supervivencia lo tienen las personas de entre 10-30 años. La fórmula para calcular este KPI es la siguiente:

Un ejemplo para este caso:



## **Desafíos o aspectos interesantes en el desarrollo**

### **Frontend**

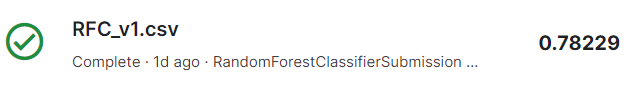
El desarrollo del frontend se ha realizado haciendo uso de la librería Tkinter, para toda la parte de interfaz, y Matplotlib, para las gráficas.

Podemos destacar dos desafíos a lo largo del proyecto:

* Visualización de las gráficas sobre la interfaz de Tkinter. El principal problema lo tuvimos para configurar la ventana creada en Tkinter y que no nos cortara parte de la gráfica.
* Al cerrar la ventana secundaria (la que mostraba la gráfica), no mataba el proceso, por lo que cuando cerrábamos la ventana principal el programa no finalizaba del todo. El error lo encontramos a la hora de crear la gráfica como figura, para poder usarla con Tkinter, no la cerrábamos y por lo tanto mantenía la ventana en “ejecución”.

### **Backend**

Hemos jugado con varios modelos de machine learning para resolver el problema de la competición de Titanic, entre los usados se encuentran: Gradient Boosting, Support Vector Machine (SVM), Logistic Regresion, K-Nearest Neighbors (KNN) y finalmente nos decantamos Random Forest ser el que mejor score nos aportaba:



Uno de los problemas ha sido la familiarización con Python y entender la estructura de código de Machine Learning (ML), limpiar los datos, usar pandas para leer ficheros, separar datos de prueba y test y predecir.

Con especial enfoque en la limpieza de datos que sin ser lo más complicado de escribir respecto al código, es la parte que más tiempo y razonamiento requiere ya que es quizá la parte más fundamental para que la IA aprenda patrones y por lo tanto obtener una predicción más precisa.

Otra parte interesante ha sido jugar con los hiper parámetros y la validación cruzada, para conseguir un sobreajuste del 100%, por ejemplo en el Random Forest, cambiamos el número de árboles y la profundidad y los multiplicamos por 100, quedando el modelo de la siguiente manera:

rfc\_model = RandomForestClassifier(n\_estimators=10000, max\_depth=500, random\_state=1)

Para los datos de entrenamiento conseguimos un sobreajuste del 100%:



Sin embargo, al probar con los datos de test de Kaggle, la precisión del modelo disminuye:

