Modelo predictivo de averías en aeronaves

Optimización del mantenimiento preventivo

Víctor Caballero Cañamero


¿Es posible crear un modelo que prediga las averías?
AIRPORT

Los datos han sido obtenidos del EDA:
“Análisis de Retrasos por causas técnicas”

Identificamos:
Averías más recurrentes
y las estaciones con
mayor impacto.

Las averías en aeronaves generan costos elevados, retrasos operativos y riesgos de seguridad.
Este proyecto busca predecir dichas averías utilizando Machine Learning, mejorando así la eficiencia operativa y reduciendo costos.

Datos y preparación

Eventos con retrasos

	ID	Occurred	At	AC	Station	Headline	FlightNo	ATA	Minutos
0	12185	2021-01-02	18:59:00	AviÃ³n 1	PIQ	98254/1	NT421	77-00	38.0
1	12194	2021-01-03	20:15:00	AviÃ³n 2	PDQ	0097630/1	NT312	33-49	15.0
2	12200	2021-01-04	06:50:00	AviÃ³n 3	PIQ	MAINTENANCE ACTION WITHOUT PILOT REPORT - OPER...	414	25-65	3.0
3	12210	2021-01-04	14:13:00	AviÃ³n 3	PDQ	0096673/1	NT158	52-42	12.0
4	12217	2021-01-05	07:33:00	AviÃ³n 4	PIQ	OTHERS / NOT SPECIFIED - OPERATIONAL	NaN	33-00	8.0
...	...	...	...	...	...	...	...	...	...
4519	68953	2024-07-19	19:00:00	AviÃ³n 39	PDQ	2 ANOT: PARASOL CM1 Y ROLLER IZQ CM1	NT273	25-13	3.0
4520	68980	2024-07-20	06:25:00	AviÃ³n 4	PDQ	MTTO SIN INFO	NT201	52-31	5.0
4521	69000	2024-07-20	14:27:00	AviÃ³n 25	PIQ	SALTA AL ARRANCAR AVIONIC SINGLE SWITCH	NT141	42-11	30.0
4522	69007	2024-07-20	18:48:00	AviÃ³n 25	PDQ	TEMPERATURA ALTA DE COMBUSTIBLE ENG #1	NT178	73-15	40.0
4523	69045	2024-07-21	10:40:00	AviÃ³n 4	PDQ	OIL PRESS N1 FLUCTUACIONES + 3 REPORTES	NT4073	79-33	29.0

Histórico de vuelos

	Fecha	Numero vuelo	Aeropuerto Despegue Real	Aeropuerto Aterrizaje Real	Matricula Real	Flota Real
0	2021-01-01	105	PDQ	PIQ	AviÃ³n 18	ATR72
1	2021-01-01	106	PIQ	PDQ	AviÃ³n 18	ATR72
2	2021-01-01	110	PIQ	PDQ	AviÃ³n 25	ATR72
3	2021-01-01	128	CVU	PDQ	AviÃ³n 20	ATR72
4	2021-01-01	145	PDQ	PIQ	AviÃ³n 25	ATR72

Tratamiento de nulos usando la frecuencia proporcional

Eliminación de columnas: AT, ID Aeropuerto Aterrizaje Real,..

Se reorganizaron la columnas de ambos para facilitar el análisis y compresión de los datos

Se creó un nuevo dataset por medio de sqlite3 para afrontar el miniEDA con una mayor claridad

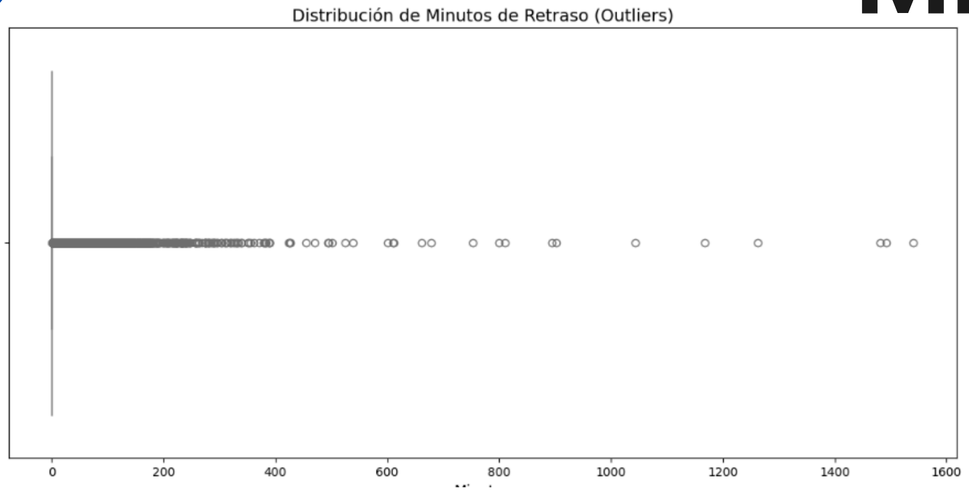
MiniEDA

El dataset tiene 203,300 registros y 6 columnas con la siguiente estructura:

* Fecha: Fecha del registro (tipo object).
* Numero_vuelo: Número de vuelo (tipo object).
* Station: Estación relacionada (tipo object).
* AC: Identificación del avión (tipo object) con valores faltantes.
* Minutos: Duración en minutos (tipo float64).
* ATA: Tipo de avería, que será nuestra variable objetivo.

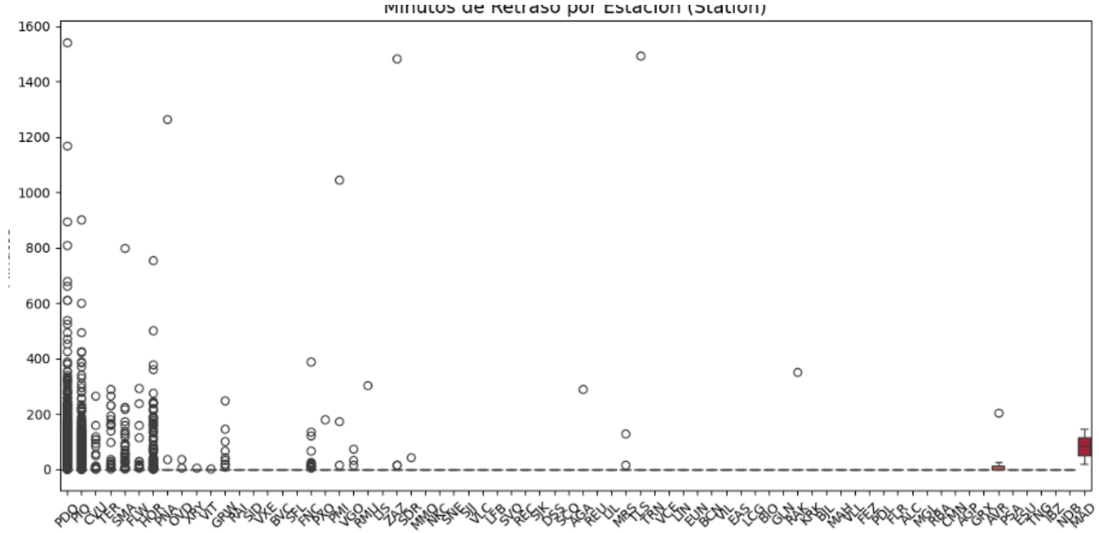
	Fecha	Numero_vuelo	Station	AC	Minutos	ATA
0	2021-01-01	105	PDQ	avion 18	0.0	99
1	2021-01-01	106	PIQ	avion 18	0.0	99
2	2021-01-01	110	PIQ	avion 25	0.0	99
3	2021-01-01	128	CVU	avion 20	0.0	99
4	2021-01-01	145	PDQ	avion 25	0.0	99

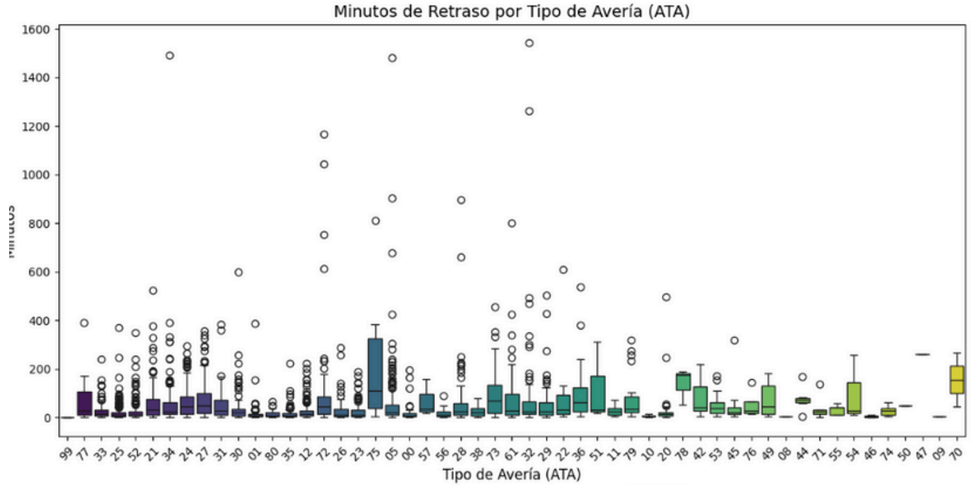
MiniEDA



Se detectan varios outliers en la distribución de minutos, con valores muy alejados del rango

ALgunas estaciones tienen outliers más frecuentes que en otras.
Destacando las bases principales PDQ y PIQ.

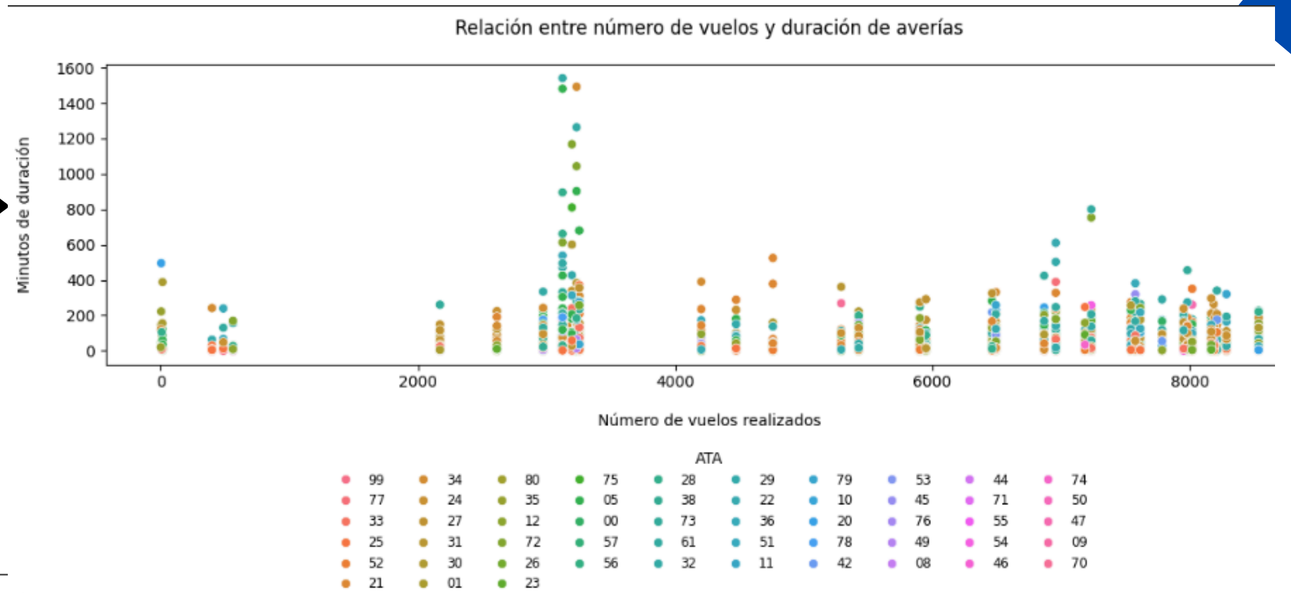


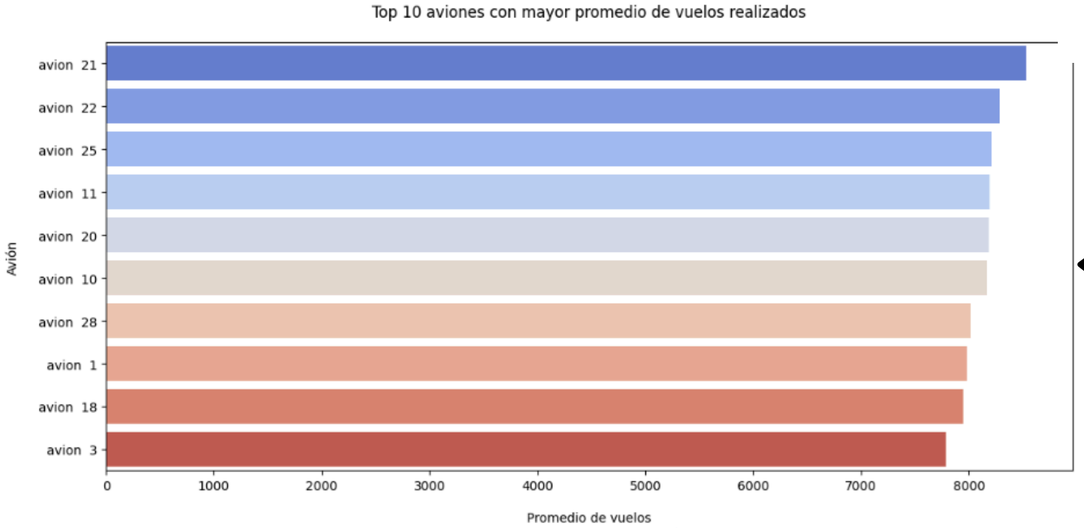


Los outliers están más concentrados en algunas averías. No es uniforme.
Además los rangos son variados.

MiniEDA

Los aviones que realizan más vuelos tienden a acumular más minutos relacionados con averías, aunque también hay mucha dispersión.



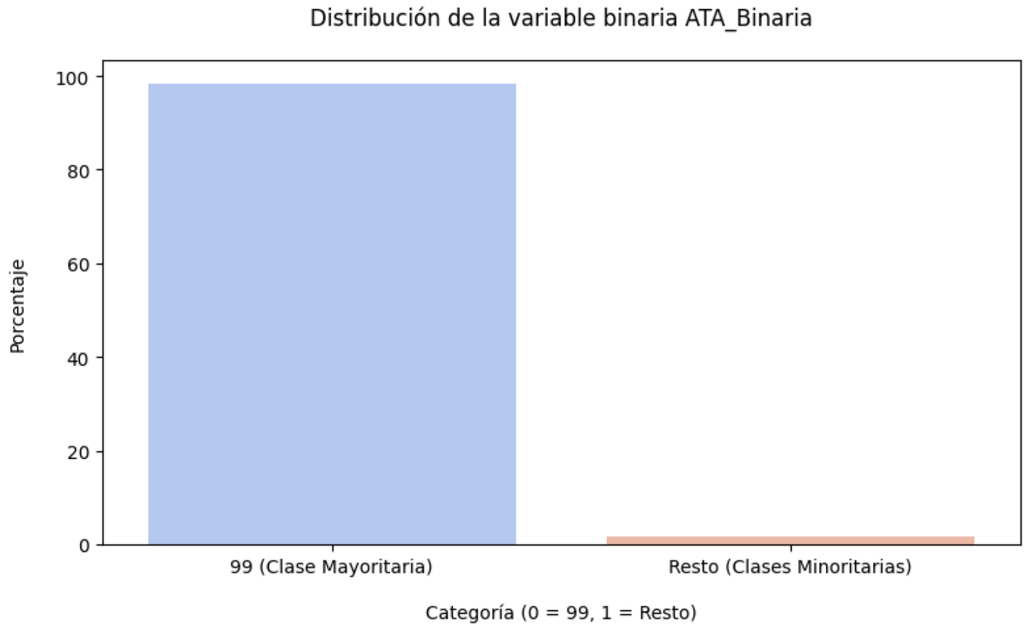


Un gráfico de barras que resalta los 10 aviones con el mayor promedio de vuelos realizados, ayudando a identificar cuáles están sometidos a mayor carga operativa.

MiniEDA

Hemos categorizado el tarjet como:
Sin avería 0 (ATA 99) y con avería 1 (el resto).
Igualmente sigue desbalanceado pero antes teníamos una gran variedad con un porcentajemuco menor.

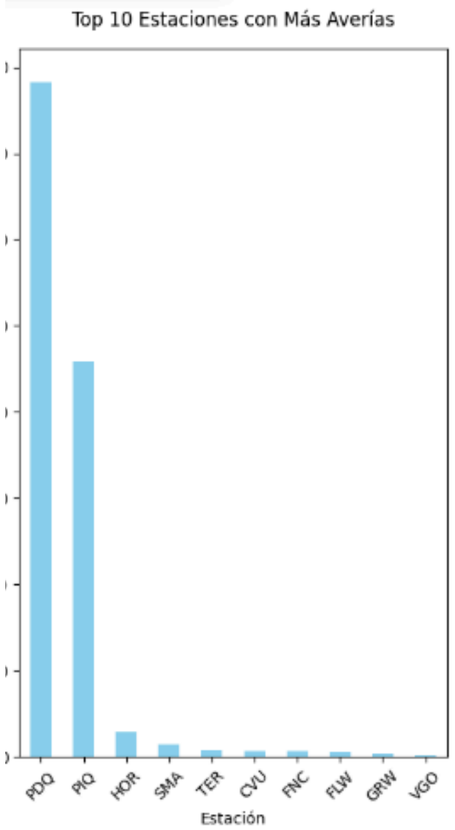
Al seguir el tarjet tan desbalanceado vamos a optar por convertirla en una variable binaria



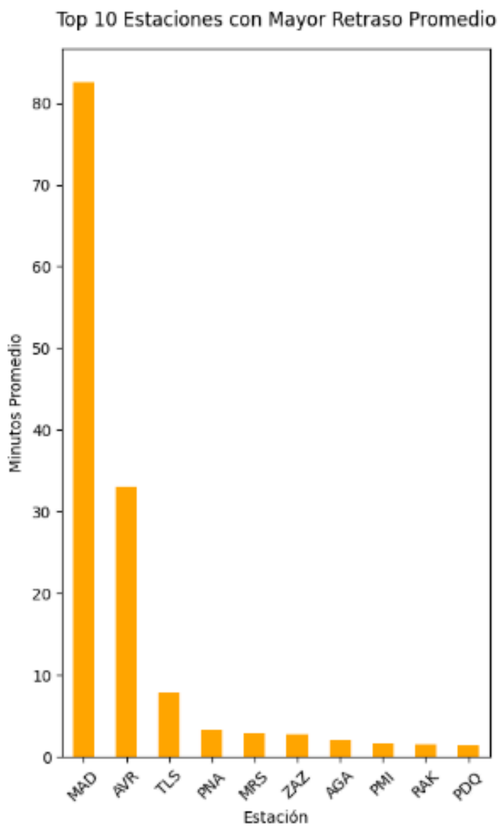
***Contras*:**
Se conserva menos información del tipo especifico de avería.
Se podría identificar relaciones más concretas entre las variables.

***Pros*:**
Requiere menos cantidad de tiempo. Se intentó realizar la primera opción pero se descarto por falta del mismo.
Modelo más simple y directo.
Los resultados seran más sencillos de entender.

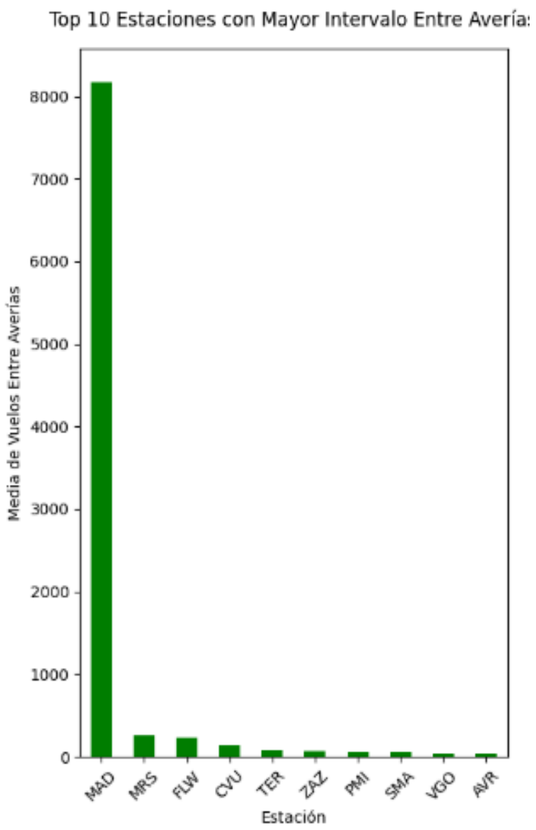
MiniEDA



Las estaciones con más retrasos es donde están los 2 centro de mantenimiento

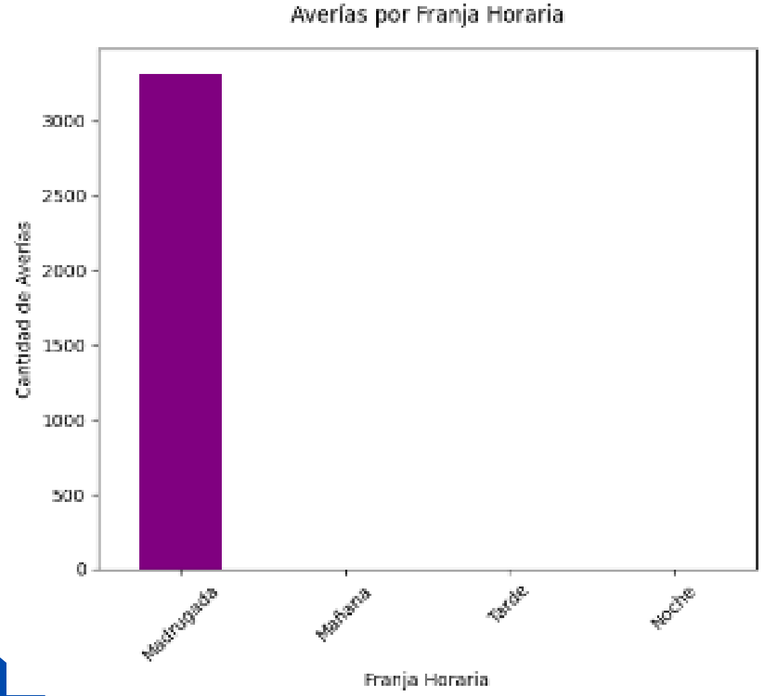


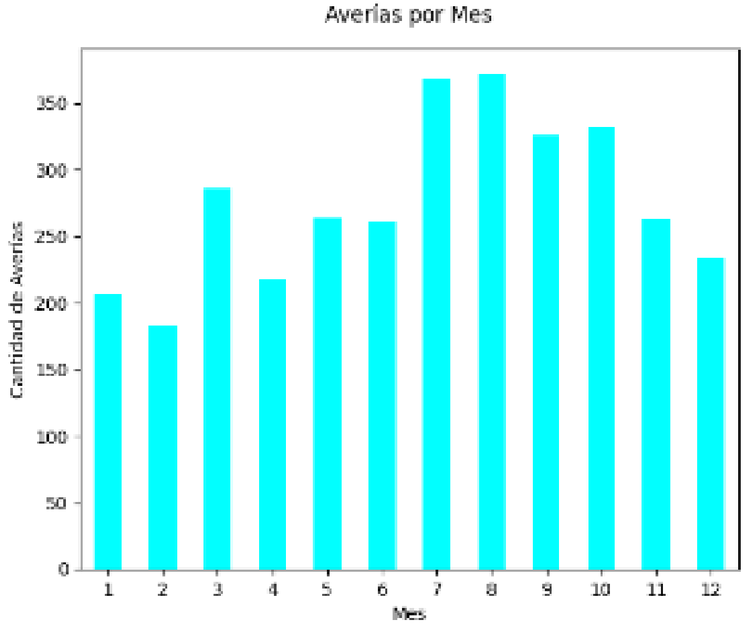
Algunas estaciones, aunque no lideran en cantidad de averías, tienen un tiempo de retraso bastante alto.

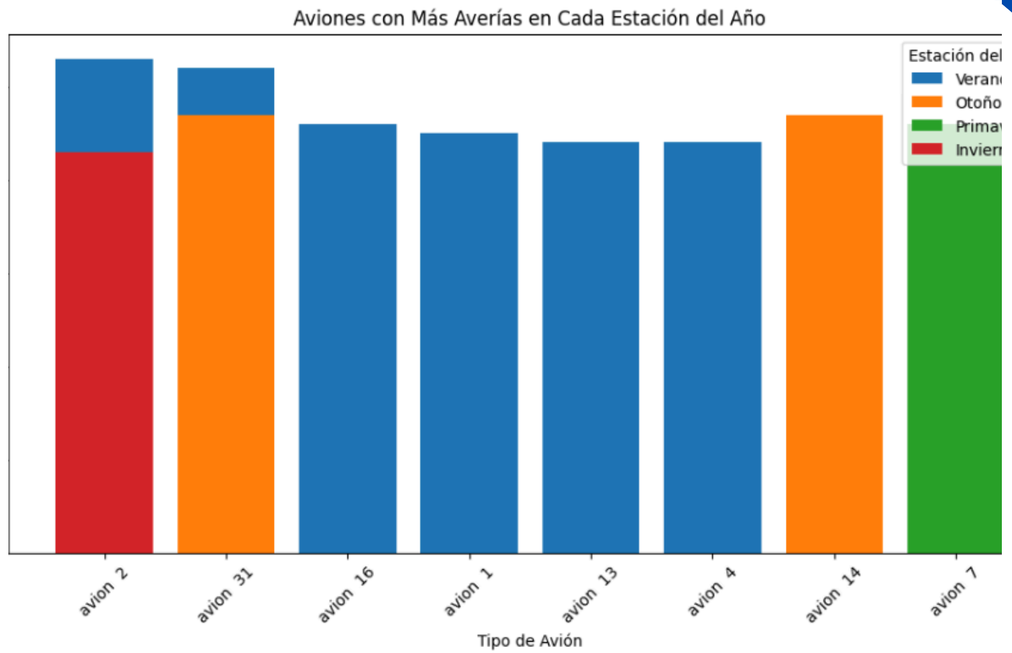


Hay una estación donde las averías son mucho más frecuentes.

Avería por franja horaria y mes







- Las averías suelen concentrarse más de madrugada.
- Algunos meses presentan un mayor número de averías.

Factores como estacionalidad y tipo de avión son determinantes para predecir averías.

Las bases principales PDQ y PIQ lideran en cantidad de averías.

Temporalidad de las averías.
Hay ciertos meses que predominan más.

Conclusiones MiniEDA

Tipo de avión:
Algunos modelos de aviones concentran más averías, indicando posibles problemas de mantenimiento o diseño.

Frecuencia elevada de averías en ciertas estaciones podría reflejar un mantenimiento ineficaz u otros problemas por definir.

Modelado

- Identificamos y separamos las variables por categoricas y numéricas.
- Variables numéricas: escalamos utilizando StandardScaler (media 0, desviación estándar 1)
- Variables categóricas: codificamos mediante OneHotEncoder
- Seleccionamos 3 modelos para realizar las pruebas:
  - Logistic Regression.
  - Random Forest.
  - Gradient Boosting.

```
# Análisis de tipos de variables
categorical_columns = X.select_dtypes(include=['object']).columns
numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns
print(f'Categóricas: {list(categorical_columns)}')
print(f'Numéricas: {list(numeric_columns)}')
```

✓ 0.0s

```
Categóricas: ['Franja_Horaria', 'AC', 'Estacion_Ano']
Numéricas: ['Minutos', 'Total_Vuelos', 'Hora']
```

```
# Preprocesamiento para Logistic Regression (escalado numérico)
# La regresión logítisca es sensible a las escalas de las variable, p
# En este caso nos interesa porque trabajando con diferentes escalas,
numeric_transformer_lr = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')# con
```

```
 Prueba de los distintos modelos
odels = {
   "Logistic Regression": (LogisticRegression(max_iter=1000, random_state=42), preprocesso
   "Random Forest": (RandomForestClassifier(n_estimators=100, random_state=42), preprocess
   "Gradient Boosting": (GradientBoostingClassifier(random_state=42), preprocessor_trees)
```

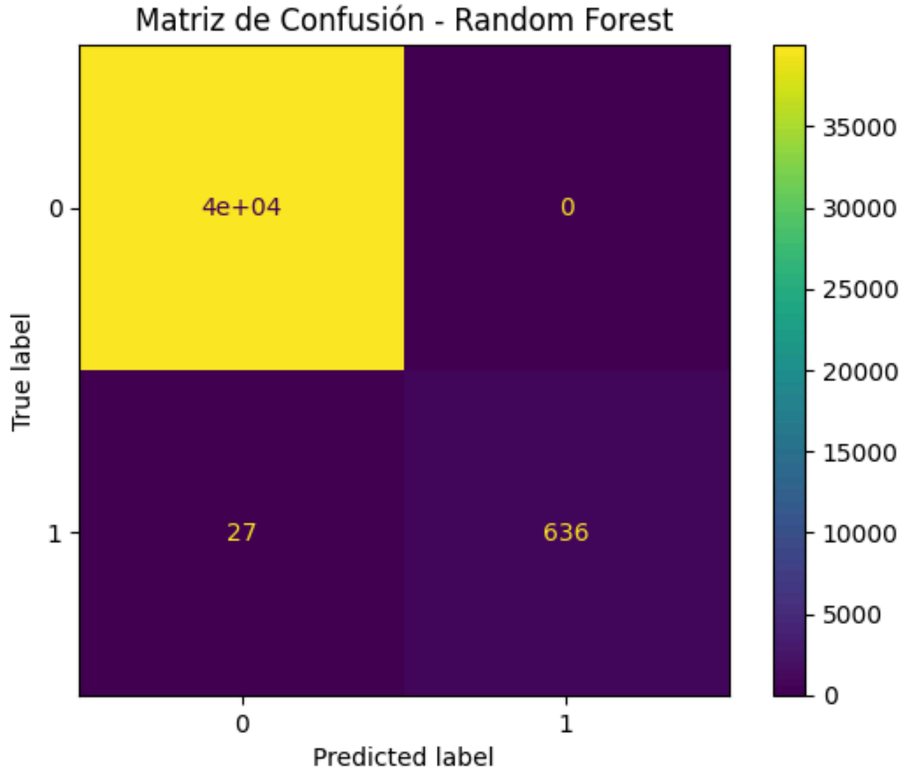
Modelado

- Usaremos un bucle para iterar sobre cada modelo de forma consecutiva:
  - Pipeline para asegurar que el preprocesamiento de los datos sea solo en train.
  - Validación cruzada.
  - Métrica principal Roc-Auc.
  - Matriz de confusión.
  - Reporte de clasificación para cada modelo.

```
# Probar modelos uno por uno
for model_name, (model, preprocessor) in models.items():
    print(f"\nProcesando modelo: {model_name}")
    # Crear pipeline
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor), #Transforma las variables categóricas y numéricas según el m
        ('classifier', model) # El modelo ML se ajustará y evaluará.
    ])
    # Validación cruzada
    print("Realizando validación cruzada...")
    cv_score = cross_val_score(pipeline, X_train, y_train, cv=5, scoring='roc_auc') # hacemos 5 subc
    print(f"Resultados de Validación Cruzada (ROC-AUC) para {model_name}: {cv_score.mean():.4f}") #

    # Entrenar y evaluar el modelo
    print("Entrenando el modelo y generando predicciones...")
    pipeline.fit(X_train, y_train) # Ajusta el modelo con el conjunto de entrenamiento (X_train, y_t
    y_pred = pipeline.predict(X_test) # Genera predicciones para el conjunto de prueba (X_test).

    # Matriz de confusión
    print("Generando matriz de confusión...")
    cm = confusion_matrix(y_test, y_pred) #Genera una matriz que compara las etiquetas reales (y_tes
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1]) # Visualiza la matriz
    disp.plot(cmap="viridis")
    plt.title(f"Matriz de Confusión - {model_name}")
    plt.show()

    # Métricas detalladas
    print(f"Reporte de clasificación para {model_name}:\n")
    print(classification_report(y_test, y_pred))
'''
```

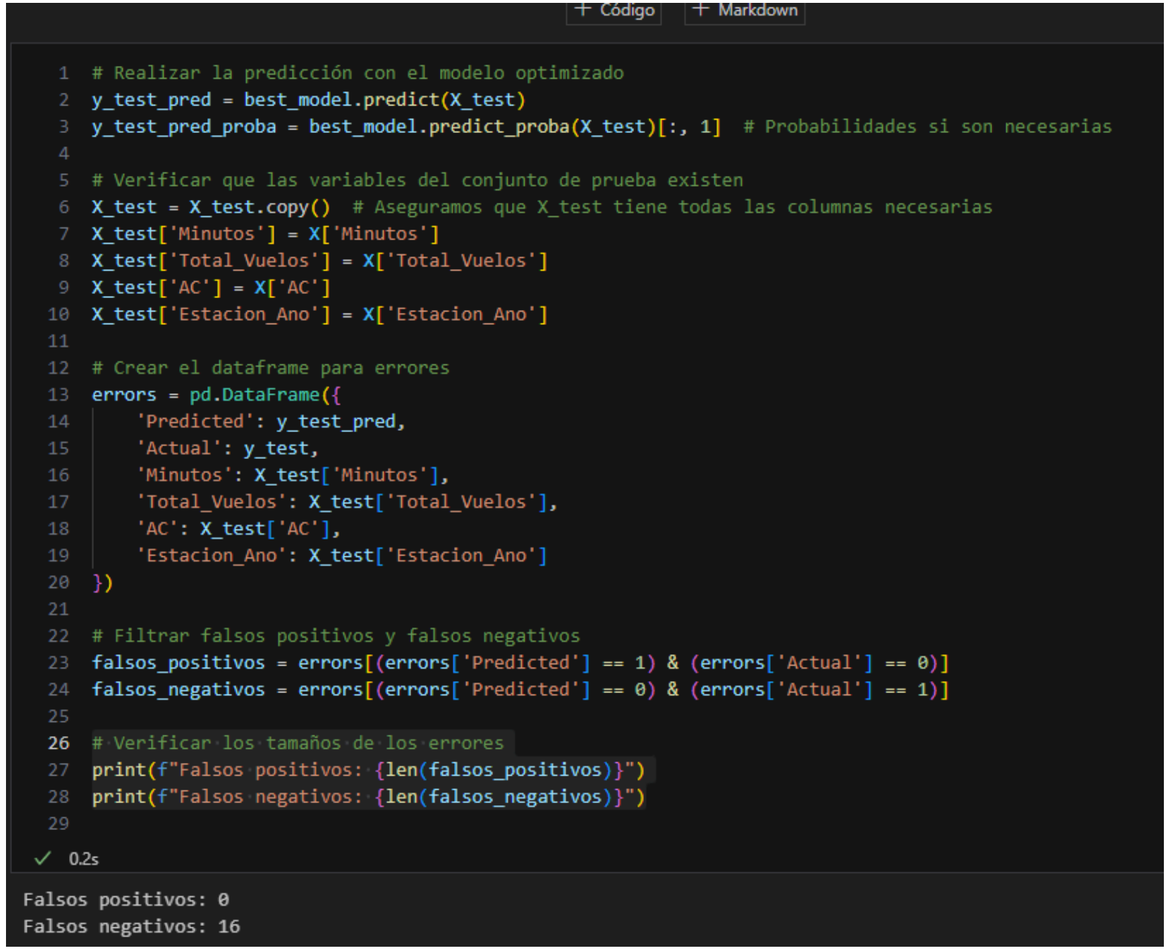
Selección del modelo

- El random forest es el elegido porque entra mútliples árboles de decisión en subconjutnos y combina resutlado para evitar sobreajuste.
- Y en comparación el gradient es más preciso pero requiere mas tiempo al ser más precio, lento y sensible a los hiperparametros



Optimización del modelo

- Previo a la optimización daba 0 falsos positivos y 27 falsos negativos.
- Usamos gridsearch para ajustar los hiperparametros, los óptimos fueron:
  - n_estimators = 100
  - max_depth = 20
  - min_samples_leaf = 2
- El modelo optimizado mostró un mejor equilibrio entre precisión y recall, con un ROC-AUC elevado tanto en validación como en prueba.
- Tras la optimización conseguimos bajar los falsos positivos a 16.



+ Código + Markdown

```python
# Realizar la predicción con el modelo optimizado
y_test_pred = best_model.predict(X_test)
y_test_pred_proba = best_model.predict_proba(X_test)[:, 1]  # Probabilidades si son necesarias

# Verificar que las variables del conjunto de prueba existen
X_test = X_test.copy()  # Aseguramos que X_test tiene todas las columnas necesarias
X_test['Minutos'] = X['Minutos']
X_test['Total_Vuelos'] = X['Total_Vuelos']
X_test['AC'] = X['AC']
X_test['Estacion_Ano'] = X['Estacion_Ano']

# Crear el dataframe para errores
errors = pd.DataFrame({
    'Predicted': y_test_pred,
    'Actual': y_test,
    'Minutos': X_test['Minutos'],
    'Total_Vuelos': X_test['Total_Vuelos'],
    'AC': X_test['AC'],
    'Estacion_Ano': X_test['Estacion_Ano']
})

# Filtrar falsos positivos y falsos negativos
falsos_positivos = errors[(errors['Predicted'] == 1) & (errors['Actual'] == 0)]
falsos_negativos = errors[(errors['Predicted'] == 0) & (errors['Actual'] == 1)]

# Verificar los tamaños de los errores
print(f"Falsos positivos: {len(falsos_positivos)}")
print(f"Falsos negativos: {len(falsos_negativos)}")
```

✓ 0.2s

```
Falsos positivos: 0
Falsos negativos: 16
```

Evolución de los hiperparámetros

1. Antes de la optimización de hiperparámetros

- **Clase `0` (Sin Avería):**
  - **Precision:** 1.00 (100%)
    - Todas las predicciones de "sin avería" son correctas.
  - **Recall:** 1.00 (100%)
    - Detecta absolutamente todos los casos de "sin avería".
  - **F1-Score:** 1.00 (100%)
    - Como la precision y el recall son perfectos, el F1-score también lo es.
- **Clase `1` (Con Avería):**
  - **Precision:** 1.00 (100%)
    - Todas las predicciones de "avería" son correctas.
  - **Recall:** 0.92 (92%)
    - El modelo detecta el 92% de los casos reales de "avería".
    - Esto significa que un 8% de las averías reales no fueron detectadas (falsos negativos).
  - **F1-Score:** 0.96 (96%)
    - Un buen equilibrio entre precision y recall, aunque puede mejorar.

2. Después de la optimización de hiperparámetros

- **Clase `0` (Sin Avería):**
  - **Precision:** 1.00 (100%)
  - **Recall:** 1.00 (100%)
  - **F1-Score:** 1.00 (100%)
    - Igual que antes, el modelo sigue manejando perfectamente los casos "sin avería".
- **Clase `1` (Con Avería):**
  - **Precision:** 1.00 (100%)
  - **Recall:** 0.96 (96%)
    - Después de la optimización, el modelo mejora el recall, detectando un mayor porcentaje de averías reales (del 92% al 96%).
    - Ahora solo un 4% de las averías reales no son detectadas, lo que reduce los falsos negativos.
  - **F1-Score:** 0.98 (98%)
    - La mejora en el recall eleva el F1-Score, indicando un mejor balance entre precision y recall.

- **El modelo maneja muy bien la clase `0` (sin avería), pero tiene un margen de mejora en el recall de la clase `1` (averías), donde algunos casos no son detectados.**
- **Esto podría tener un impacto crítico en el negocio si se pierden alertas de averías reales.**

- **El modelo optimizado mejora el manejo de la clase `1` (averías), detectando más casos reales con menos falsos negativos.**
- **La optimización de los hiperparámetros fue efectiva para reforzar el rendimiento global, manteniendo el excelente desempeño en la clase `0`.**

Conclusiones modelo predicción Random Forest

El modelo logró una precisión del 98.28%, muy buen balance entre las métricas de precisión, recall y F1-score para ambas clases (sin avería y con avería).

No se identificaron falsos positivos en las predicciones.

El modelo es robusto, generaliza bien con nuevos datos, proporciona predicciones confiables.

Es apto para entornos reales donde se necesita anticipar averías, optimizando el mantenimiento y reduciendo retrasos.

Las métricas muestran un balance adecuado entre precisión y recall, especialmente en la clase crítica de "Con avería"