



UNIVERSIDAD DE LAS AMÉRICAS

INTELIGENCIA ARTIFICIAL I

Tema de proyecto: Estimación de costos por colisiones aéreas con aves

Autores: Moncayo Jorge, Montalvo Hayland & Palacios Gorky

BIRD STRIKES: Análisis DATASET

Los golpes de aves, o colisiones entre aves y aeronaves, ocurren cuando las aves entran en contacto con las aeronaves, representando un riesgo significativo para la seguridad en la aviación. Estos eventos pueden ir desde inconvenientes menores hasta accidentes catastróficos, dependiendo de la gravedad de la colisión y las circunstancias de la aeronave.

Los golpes de aves pueden producirse en cualquier etapa del vuelo, pero son más frecuentes durante el despegue y el aterrizaje, cuando la aeronave opera a altitudes más bajas. Estos incidentes pueden provocar fallos en los motores, daños estructurales o accidentes, con posibles lesiones o pérdidas humanas. A medida que el tráfico aéreo crece, también lo hacen los riesgos por golpes de aves, convirtiéndolos en un área crítica de atención para los profesionales de seguridad aérea.

Causas y Factores de Riesgo

Las colisiones con aves se producen por varias razones, y los factores de riesgo pueden variar. Las causas más comunes incluyen:

- Patrones de Migración:** Las aves siguen rutas migratorias que pueden coincidir con rutas de vuelo. Durante las temporadas de migración, el número de aves en el aire aumenta, incrementando la probabilidad de colisiones.
- Proximidad de Hábitats:** Los aeropuertos suelen ubicarse cerca de cuerpos de agua, humedales o campos abiertos, zonas que atraen aves. Esta proximidad incrementa la actividad de aves alrededor de las pistas.

3. **Velocidad y Altitud de la Aeronave:** Los golpes de aves son más frecuentes a bajas altitudes (menos de 3.000 pies), especialmente en despegues y aterrizajes. La velocidad relativa entre el ave y la aeronave también influye en la gravedad del impacto.
4. **Especies de Aves:** Las aves de gran tamaño, como gansos canadienses o buitres, suponen un mayor riesgo debido a su tamaño y peso. Sin embargo, aves más pequeñas pueden causar daños significativos, especialmente si impactan componentes críticos como motores o parabrisas.
5. **Variación Estacional:** La frecuencia de golpes de aves suele correlacionarse con cambios estacionales. Los picos ocurren en primavera y otoño, cuando grandes cantidades de aves migran a través de países y continentes.

Impactos de los Golpes de Aves

1. **Daño o Falla del Motor:** Uno de los resultados más peligrosos de un golpe de aves es cuando un ave es ingerida por el motor. Esto puede dañar las palas de la turbina, provocando falla del motor o, en casos raros, la pérdida completa de potencia.
2. **Daño al Parabrisas:** Un golpe directo al parabrisas de la cabina puede afectar la visibilidad del piloto y generar situaciones peligrosas. Aunque los aviones modernos están diseñados para soportar impactos, el efecto psicológico en la tripulación es considerable.
3. **Daño Estructural:** Un golpe de aves puede causar abolladuras, grietas o agujeros en las alas, fuselaje o cola de la aeronave. Este tipo de daño puede comprometer la integridad estructural, especialmente a altas velocidades.
4. **Lesiones o Fatalidades:** Aunque relativamente raro, un golpe de aves puede causar lesiones o la muerte de pasajeros o tripulación. En casos de golpes catastróficos con aves grandes, como gansos canadienses, puede haber graves consecuencias.
5. **Costos Financieros:** El costo económico incluye reparaciones, investigación y retrasos, sin contar posibles responsabilidades legales. La industria aeronáutica gasta millones cada año en prevención y mitigación de riesgos.

Estadísticas de Golpes de Aves

- Según la **FAA (Administración Federal de Aviación)**, más de **13.000** golpes de aves fueron reportados en EE. UU. en 2021, aunque la cifra real podría ser mayor debido al subregistro.
- Los golpes con **aviones comerciales** han aumentado en frecuencia durante las últimas décadas, aunque la gravedad de los incidentes ha disminuido gracias a mejoras en el diseño y protocolos de seguridad.
- Las especies de aves involucradas varían ampliamente, pero aves acuáticas grandes como gansos son de las más peligrosas. Otras especies problemáticas incluyen gaviotas,

halcones y cuervos.

Importación de Librerías

```
In [36]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

Visión General de los Datos

El conjunto de datos proporciona información detallada sobre incidentes de golpes de aves en EE. UU. de 2000 a 2011, recopilada por la FAA. Incluye una amplia variedad de variables que describen las circunstancias e impactos de estos incidentes, enfocándose en la seguridad, costos económicos y patrones de ocurrencia.

```
In [4]: data = pd.read_csv('Bird_strikes.csv')
data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25429 entries, 0 to 25428
Data columns (total 26 columns):
 #   Column           Non-Null Count Dtype  
 ---  -- 
 0   RecordID        25429 non-null  int64   
 1   AircraftType    25429 non-null  object  
 2   AirportName     25429 non-null  object  
 3   AltitudeBin     25429 non-null  object  
 4   MakeModel       25429 non-null  object  
 5   NumberStruck    25429 non-null  object  
 6   NumberStruckActual 25429 non-null  int64   
 7   Effect          2078 non-null  object  
 8   FlightDate      25429 non-null  object  
 9   Damage          25429 non-null  object  
 10  Engines          25195 non-null  object  
 11  Operator         25429 non-null  object  
 12  OriginState     24980 non-null  object  
 13  FlightPhase     25429 non-null  object  
 14  ConditionsPrecipitation 2015 non-null  object  
 15  RemainsCollected? 25429 non-null  bool    
 16  RemainsSentToSmithsonian 25429 non-null  bool    
 17  Remarks          20668 non-null  object  
 18  WildlifeSize     25429 non-null  object  
 19  ConditionsSky    25429 non-null  object  
 20  WildlifeSpecies  25429 non-null  object  
 21  PilotWarned      25429 non-null  object  
 22  Cost              25429 non-null  object  
 23  Altitude         25429 non-null  int64   
 24  PeopleInjured    25429 non-null  int64   
 25  IsAircraftLarge? 25429 non-null  object  
dtypes: bool(2), int64(4), object(20)
memory usage: 4.7+ MB

```

```
In [6]: # Check for missing data
missing_df = data.isnull().sum().rename('missing').reset_index()
missing_df['missing_percentage'] = missing_df['missing']/data.shape[0]*100
missing_df[missing_df['missing']>0].style.bar(subset='missing_percentage',color='red')
```

	index	missing	missing_percentage
7	Effect	23351	91.828228
10	Engines	234	0.920209
12	OriginState	449	1.765701
14	ConditionsPrecipitation	23414	92.075976
17	Remarks	4761	18.722718

```
In [7]: # Dropping unnecessary features
data.drop(['RecordID','AltitudeBin','NumberStruck','Effect','ConditionsPrecipitatio
# Correcting data types
data.FlightDate = pd.DatetimeIndex(data.FlightDate)
```

```

data.Cost = data.Cost.str.replace(',', '')
data.Cost = data.Cost.astype(float)
data.Engines = data.Engines.replace('C', '2')
data.Engines = data.Engines.fillna(int(data.Engines.mode())).astype(int)
data.OriginState.fillna('NA', inplace=True)
data.FlightPhase = data.FlightPhase.replace({'Descent': 'Others', 'Taxi': 'Others', 'Pa

```

In [8]: `data.describe().style.background_gradient()`

	NumberStruckActual	FlightDate	Engines	Cost	Altitud
count	25429.000000	25429	25429.000000	25429.000000	25429.000000
mean	2.699634	2007-01-22 13:20:40.017303040	2.003933	5566.368241	799.02843
min	1.000000	2000-01-02 00:00:00	1.000000	0.000000	0.000000
25%	1.000000	2004-06-17 00:00:00	2.000000	0.000000	0.000000
50%	1.000000	2007-07-29 00:00:00	2.000000	0.000000	50.000000
75%	1.000000	2009-11-01 00:00:00	2.000000	0.000000	700.000000
max	942.000000	2011-12-31 00:00:00	4.000000	12397751.000000	18000.000000
std	12.825804	nan	0.363270	122238.789137	1740.07984

In [9]: `data.describe(include='O')`

	AirportName	MakeModel	Damage	Operator	OriginState	FlightPhase	Wildlif
count	25429	25429	25429	25429	25429	25429	25429
unique	1109	324	2	292	61	5	2
top	DALLAS/FORT WORTH INTL ARPT	B-737-700	No damage	SOUTHWEST AIRLINES	California	Approach	2
freq	803	2488	22975	4628	2499	10382	1

In [10]: `data.head()`

Out[10]:

	AirportName	MakeModel	NumberStruckActual	FlightDate	Damage	Engines	Operat
0	LAGUARDIA NY	B-737-400	859	2000-11-23	Caused damage	2	AIRWA
1	DALLAS/FORT WORTH INTL ARPT	MD-80	424	2001-07-25	Caused damage	2	AMERIC AIRLIN
2	LAKEFRONT AIRPORT	C-500	261	2001-09-14	No damage	2	BUSINE
3	SEATTLE- TACOMA INTL	B-737-400	806	2002-09-05	No damage	2	ALAS AIRLIN
4	NORFOLK INTL	CL- RJ100/200	942	2003-06-23	No damage	2	COM AIRLIN

Análisis Exploratorio de Datos

In [11]:

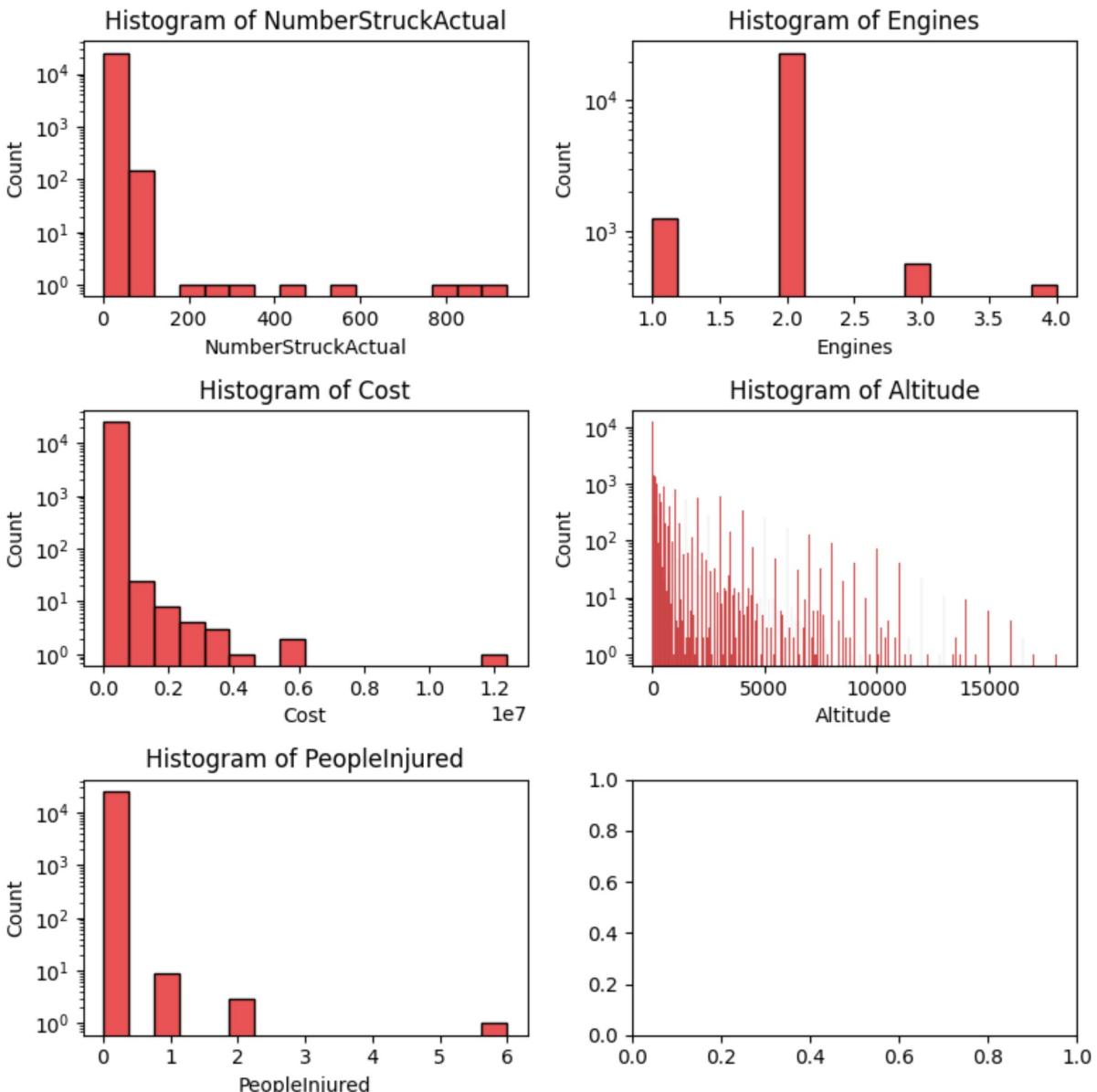
```

sns.set_palette('Set1')
fig,ax = plt.subplots(3,2,figsize=(8,8))
ax = ax.flatten()

for i,feat in enumerate(data.select_dtypes(include='number').columns):
    sns.histplot(data[feat], ax=ax[i])
    ax[i].set_yscale('log') # Set logarithmic scale on x-axis
    ax[i].set_title(f"Histogram of {feat}")

plt.tight_layout()
plt.show()

```



Tendencias

```
In [12]: cmap = cm.get_cmap('rocket_r')
fig,ax = plt.subplots(figsize=(8,4))

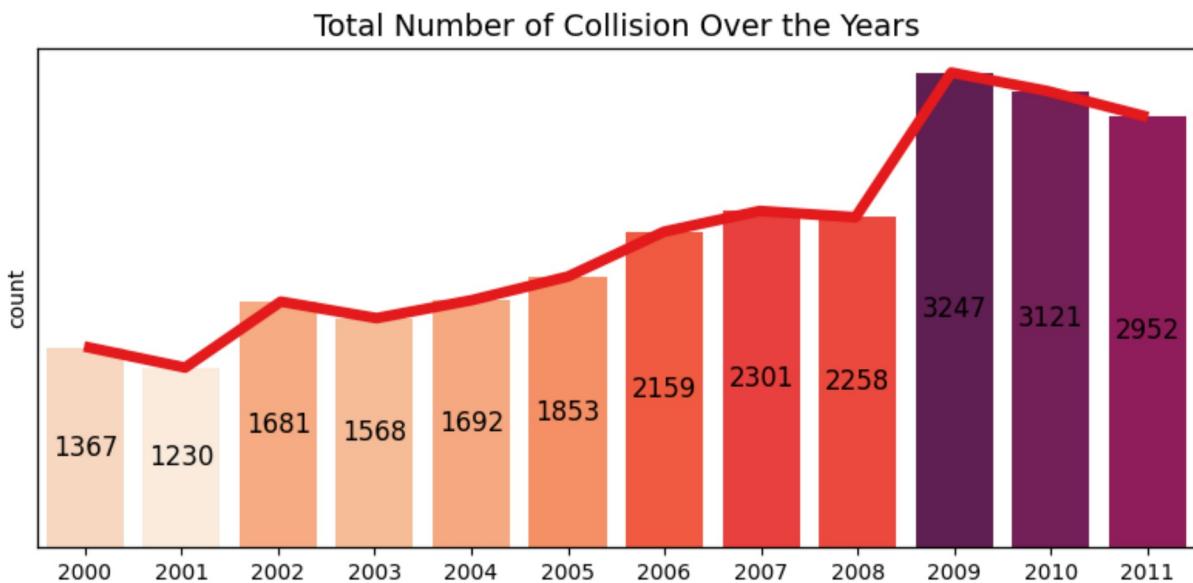
agg_data = data.FlightDate.dt.year.value_counts()
norm = plt.Normalize(agg_data.min(), agg_data.max())
bars = sns.barplot(x=agg_data.index,y=agg_data,ax=ax,)
bars.bar_label(bars.containers[0], fontsize=12, label_type='center')

for bar in bars.patches:
    height = bar.get_height()
    bar.set_facecolor(cmap(norm(height)*0.75))

ax2 = ax.twiny()
sns.lineplot(x=agg_data.index,y=agg_data,ax=ax2,linewidth=5,palette='b')
```

```
plt.title('Total Number of Collision Over the Years', fontsize=14)
plt.xticks([])
plt.yticks([])
ax.set_xlabel('')
plt.xlabel('')

plt.tight_layout()
plt.show()
```



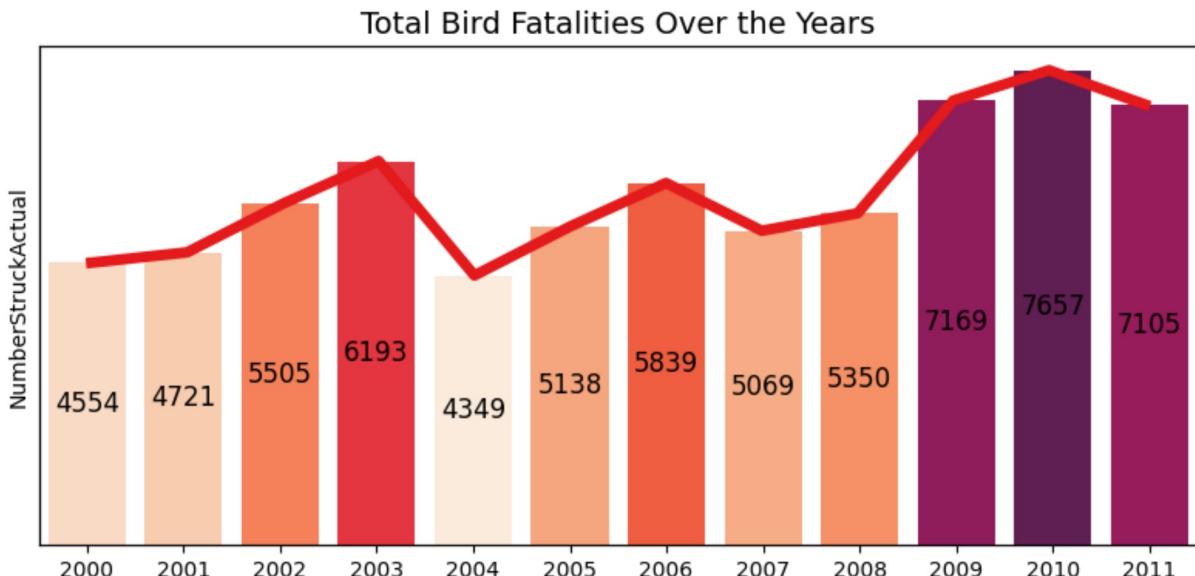
```
In [13]: fig,ax = plt.subplots(figsize=(8,4))

agg_data = data.groupby(data.FlightDate.dt.year)[ 'NumberStruckActual'].sum()
norm = plt.Normalize(agg_data.min(), agg_data.max())
bars = sns.barplot(x=agg_data.index,y=agg_data,ax=ax,)
bars.bar_label(bars.containers[0], fontsize=12, label_type='center')

for bar in bars.patches:
    height = bar.get_height()
    bar.set_facecolor(cmap(norm(height)*0.75))

ax2 = ax.twiny()
sns.lineplot(x=agg_data.index,y=agg_data, ax=ax2, linewidth=5)
plt.title('Total Bird Fatalities Over the Years', fontsize=14)
plt.xticks([])
plt.yticks([])
ax.set_xlabel('')
plt.xlabel('')

plt.tight_layout()
plt.show()
```

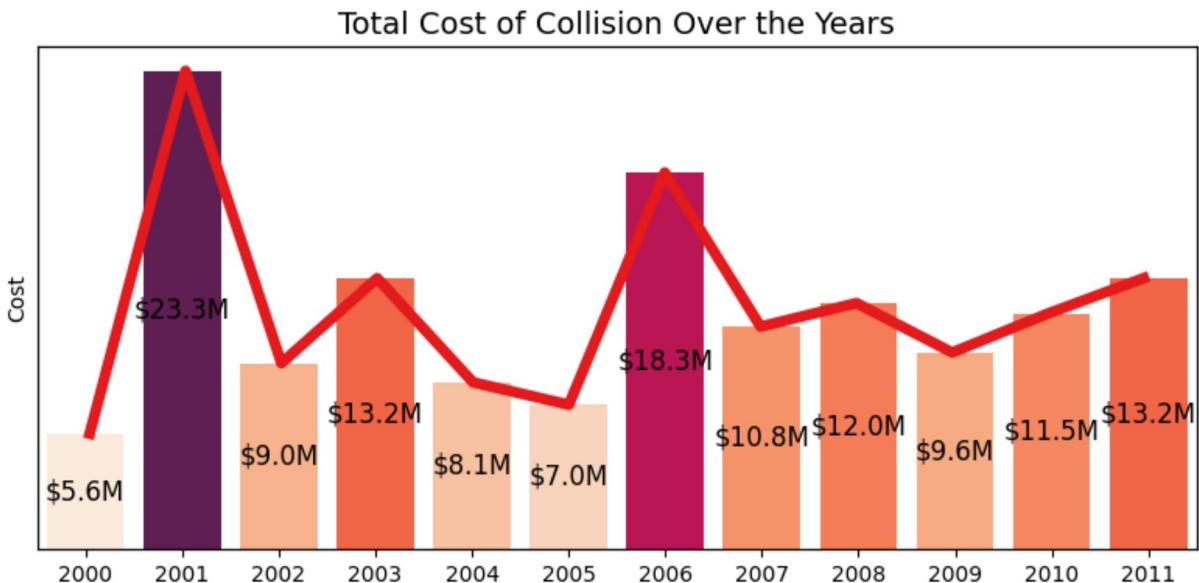


```
In [14]: fig,ax = plt.subplots(figsize=(8,4))

agg_data = data.groupby(data.FlightDate.dt.year)[ 'Cost'].sum()
norm = plt.Normalize(agg_data.min(), agg_data.max())
bars = sns.barplot(x=agg_data.index,y=agg_data,ax=ax,)
bars.bar_label(bars.containers[0], fontsize=12,label_type='center',labels=[f'${x/1e
for bar in bars.patches:
    height = bar.get_height()
    bar.set_facecolor(cmap(norm(height)*0.75))

ax2 = ax.twiny()
sns.lineplot(x=agg_data.index,y=agg_data, ax=ax2,linewidth=5)
plt.title('Total Cost of Collision Over the Years',fontsize=14)
plt.xticks([])
plt.yticks([])
ax.set_xlabel('')
plt.xlabel('')

plt.tight_layout()
plt.show()
```



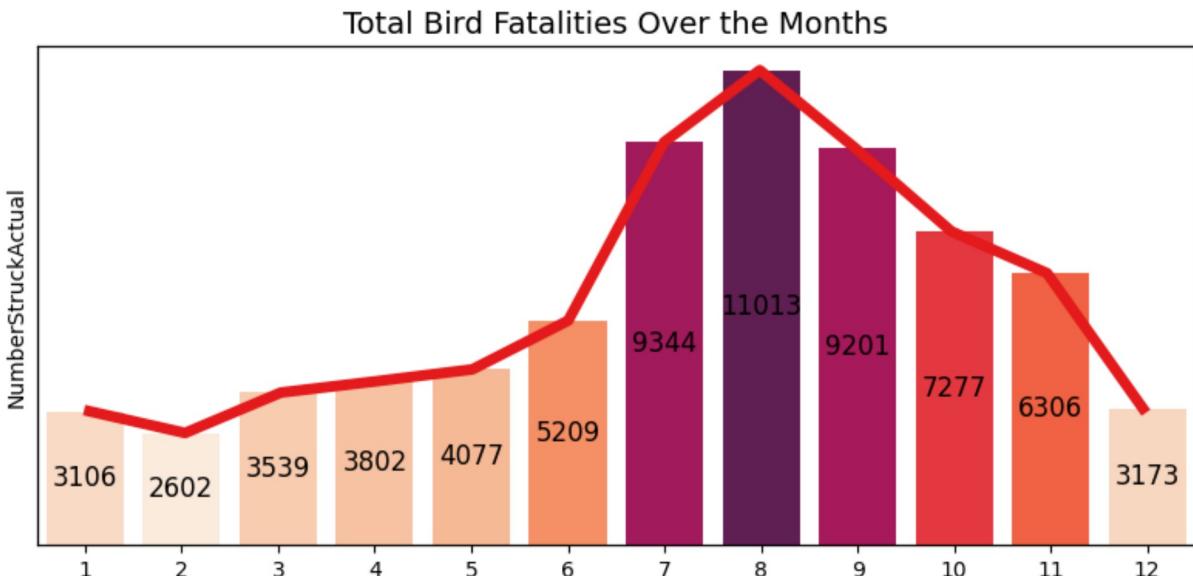
```
In [15]: fig,ax = plt.subplots(figsize=(8,4))

agg_data = data.groupby(data.FlightDate.dt.month)[['NumberStruckActual']].sum()
norm = plt.Normalize(agg_data.min(), agg_data.max())
bars = sns.barplot(x=agg_data.index,y=agg_data,ax=ax,)
bars.bar_label(bars.containers[0], fontsize=12,label_type='center')

for bar in bars.patches:
    height = bar.get_height()
    bar.set_facecolor(cmap(norm(height)*0.75))

ax2 = ax.twiny()
sns.lineplot(x=agg_data.index,y=agg_data, ax=ax2, linewidth=5)
plt.title('Total Bird Fatalities Over the Months', fontsize=14)
plt.xticks([])
plt.yticks([])
ax.set_xlabel('')
plt.xlabel('')

plt.tight_layout()
plt.show()
```



```
In [22]: fig, ax = plt.subplots(1, 2, figsize=(8, 4))
ax = ax.flatten()

# Pie chart
agg_data_pie = data.groupby('WildlifeSize').size().sort_values()
plt.sca(ax[0])
plt.pie(
    agg_data_pie,
    autopct='%.2f%%',
    labels=agg_data_pie.index,
    colors=sns.color_palette("rocket_r", len(agg_data_pie)))
)
plt.pie([1], radius=0.5, colors=['white'])
plt.title('Percentage of Collision by Bird Size', fontsize=14)

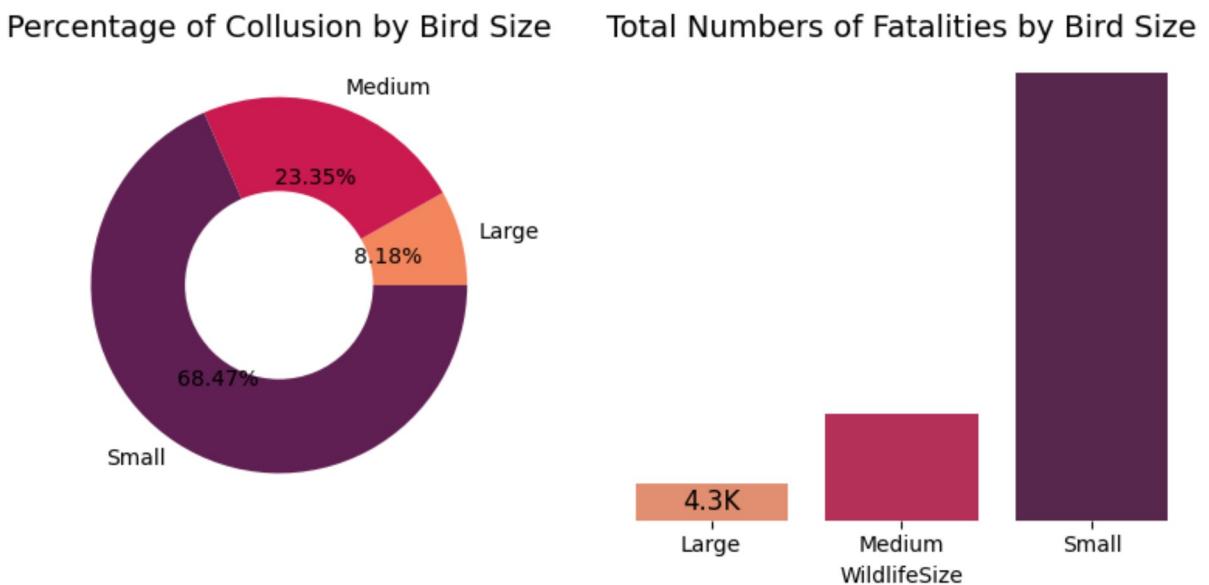
# Bar plot
agg_data_bar = data.groupby('WildlifeSize')[['NumberStruckActual']].sum().dropna()
bars = sns.barplot(x=agg_data_bar.index, y=agg_data_bar.values, ax=ax[1], palette=''

# Etiquetar barras solo si son válidas
def safe_bar_label(ax, container, labels, **kwargs):
    from matplotlib.text import Text
    bars = [bar for bar in container if bar is not None and bar.get_height() > 0]
    # Solo etiquetar barras válidas, con sus respectivas etiquetas
    for bar, label in zip(bars, labels):
        x = bar.get_x() + bar.get_width() / 2
        y = bar.get_height()
        ax.text(x, y / 2, label, ha='center', va='center', fontsize=kwargs.get('font

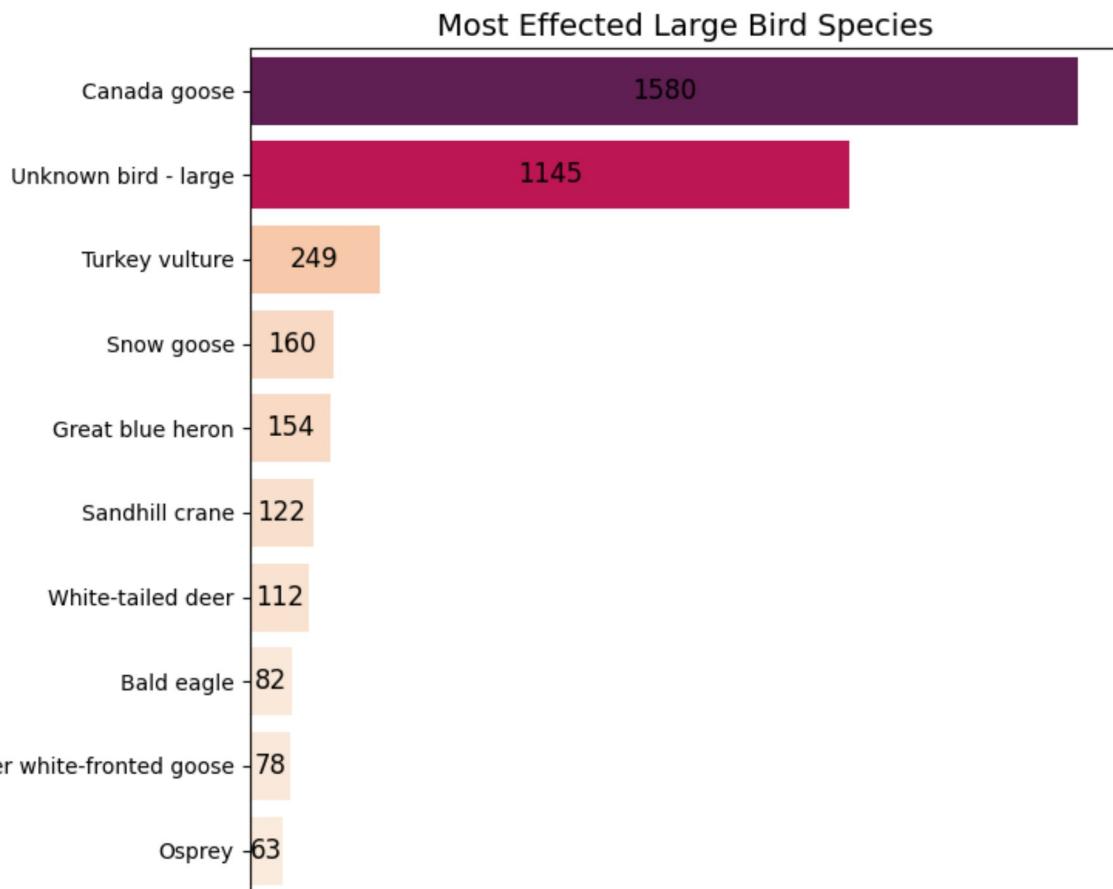
if bars.containers:
    safe_bar_label(ax[1], bars.containers[0], [f'{x/1e3:.1f}K' for x in agg_data_ba

plt.sca(ax[1])
plt.title('Total Numbers of Fatalities by Bird Size', fontsize=14)
plt.yticks([])
plt.ylabel('')
plt.box(False)
```

```
plt.tight_layout()  
plt.show()
```



```
In [23]: fig,ax = plt.subplots(figsize=(8,6))  
  
agg_data = data[data.WildlifeSize=='Large'].groupby('WildlifeSpecies')['NumberStruc  
norm = plt.Normalize(agg_data.min(), agg_data.max())  
bars = sns.barplot(y=agg_data.index,x=agg_data,ax=ax,orient='h')  
bars.bar_label(bars.containers[0], fontsize=12, label_type='center')  
  
for bar in bars.patches:  
    width = bar.get_width()  
    bar.set_facecolor(cmap(norm(width)*0.75))  
  
plt.title('Most Effected Large Bird Species', fontsize=14)  
plt.xticks([])  
plt.ylabel('')  
plt.xlabel('')  
  
plt.tight_layout()  
plt.show()
```



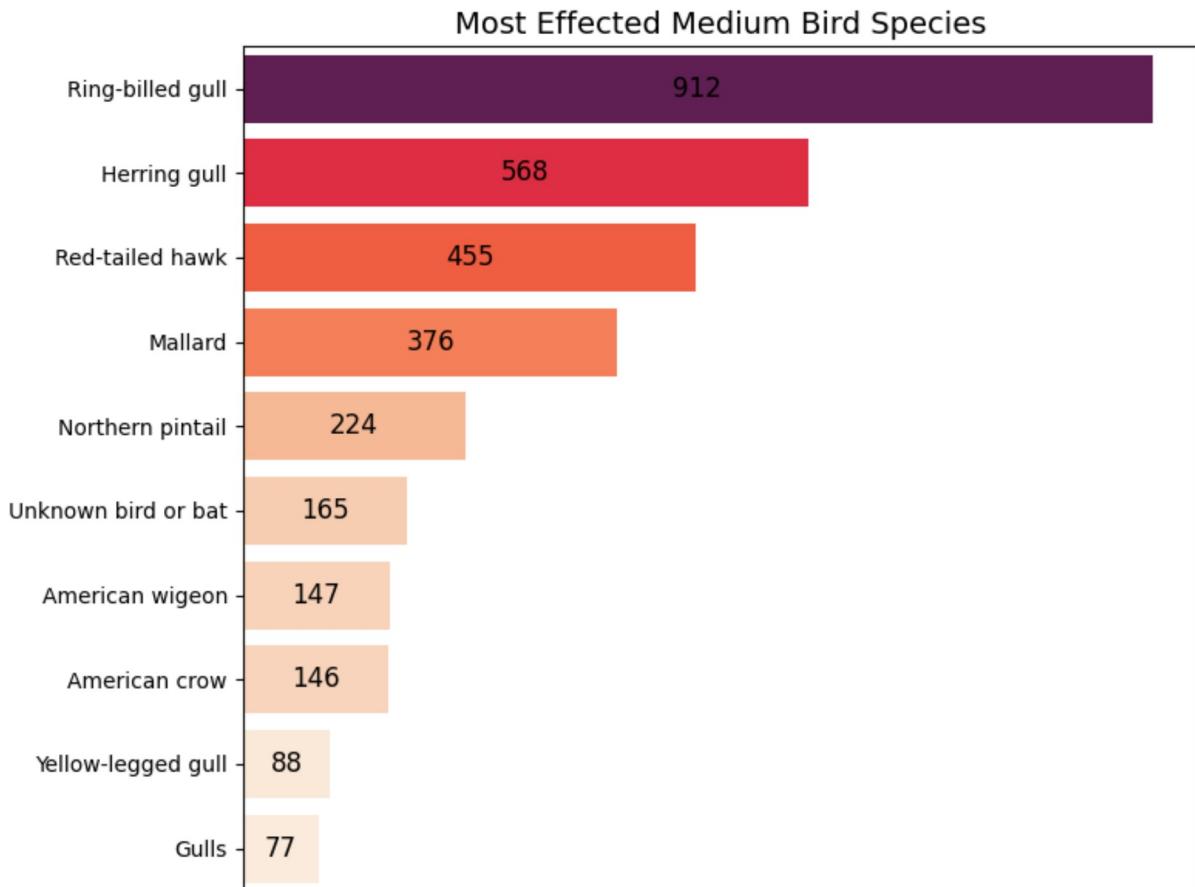
```
In [24]: fig,ax = plt.subplots(figsize=(8,6))

agg_data = data[data.WildlifeSize=='Medium'].groupby('WildlifeSpecies')[['NumberStru
norm = plt.Normalize(agg_data.min(), agg_data.max())
bars = sns.barplot(y=agg_data.index,x=agg_data,ax=ax,orient='h')
bars.bar_label(bars.containers[0], fontsize=12, label_type='center')

for bar in bars.patches:
    width = bar.get_width()
    bar.set_facecolor(cmap(norm(width)*0.75))

plt.title('Most Effect Medium Bird Species', fontsize=14)
plt.xticks([])
plt.ylabel('')
plt.xlabel('')

plt.tight_layout()
plt.show()
```



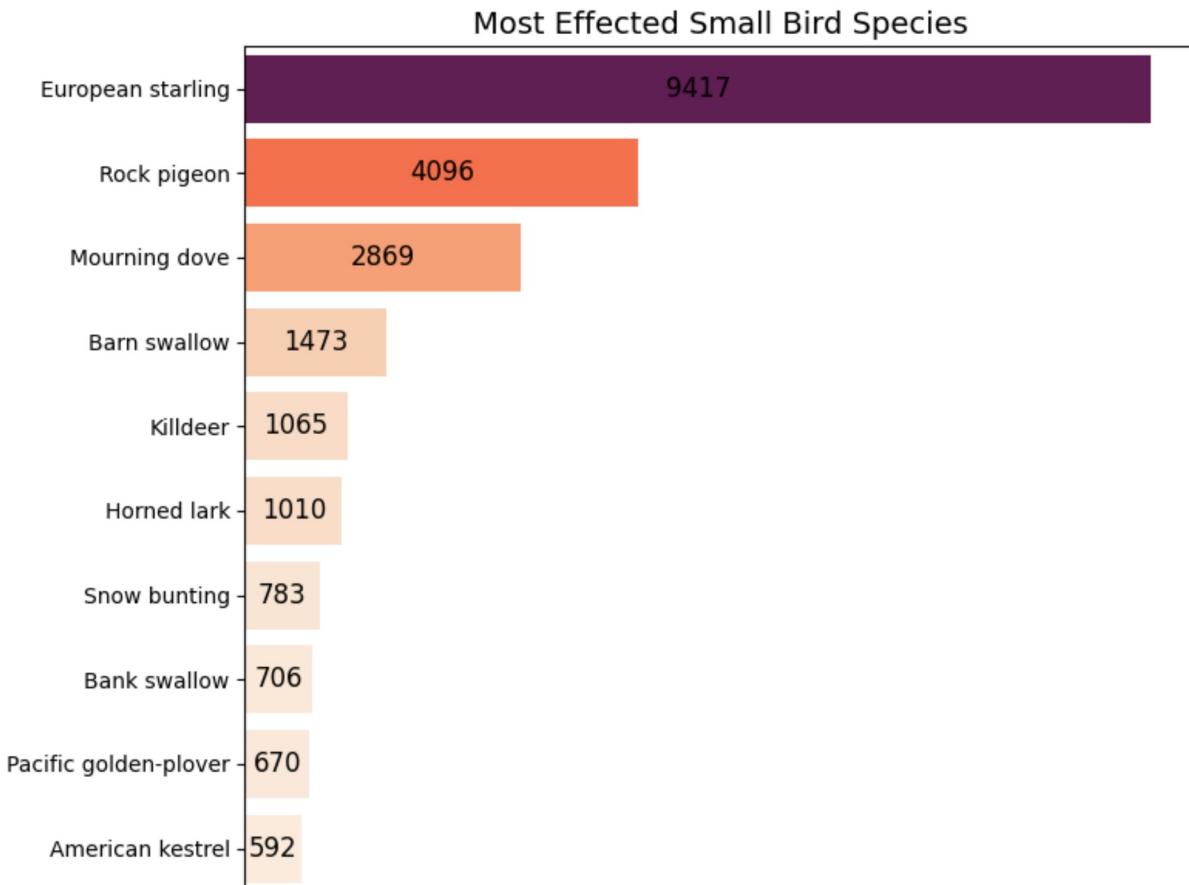
```
In [25]: fig,ax = plt.subplots(figsize=(8,6))

agg_data = data[data.WildlifeSize=='Small'].groupby('WildlifeSpecies')['NumberStruc
norm = plt.Normalize(agg_data.min(), agg_data.max())
bars = sns.barplot(y=agg_data.index,x=agg_data,ax=ax,orient='h')
bars.bar_label(bars.containers[0], fontsize=12, label_type='center')

for bar in bars.patches:
    width = bar.get_width()
    bar.set_facecolor(cmap(norm(width)*0.75))

plt.title('Most Effect Small Bird Species',fontsize=14)
plt.xticks([])
plt.ylabel('')
plt.xlabel('')

plt.tight_layout()
plt.show()
```



```
In [31]: ig, ax = plt.subplots(1, 2, figsize=(12, 5))
ax = ax.flatten()

# Gráfico 1: Pie chart
agg_data = data.groupby('FlightPhase').size().sort_values()
plt.sca(ax[0])
plt.pie(
    agg_data,
    autopct='%.2f%%',
    labels=agg_data.index,
    colors=sns.color_palette("rocket_r", len(agg_data)))
)
plt.pie([1], radius=0.5, colors=['white'])
plt.title('Percentage of Collision by Flight Phase', fontsize=12)

# Gráfico 2: Barplot con etiquetas manuales
agg_data_bar = data.groupby('FlightPhase')[['NumberStruckActual']].sum().sort_values()
norm = plt.Normalize(agg_data_bar.min(), agg_data_bar.max())
cmap = cm.get_cmap('rocket_r')

bars = sns.barplot(
    x=agg_data_bar.index,
    y=agg_data_bar.values,
    ax=ax[1],
    palette='rocket_r'
)

# Agregar etiquetas manualmente
```

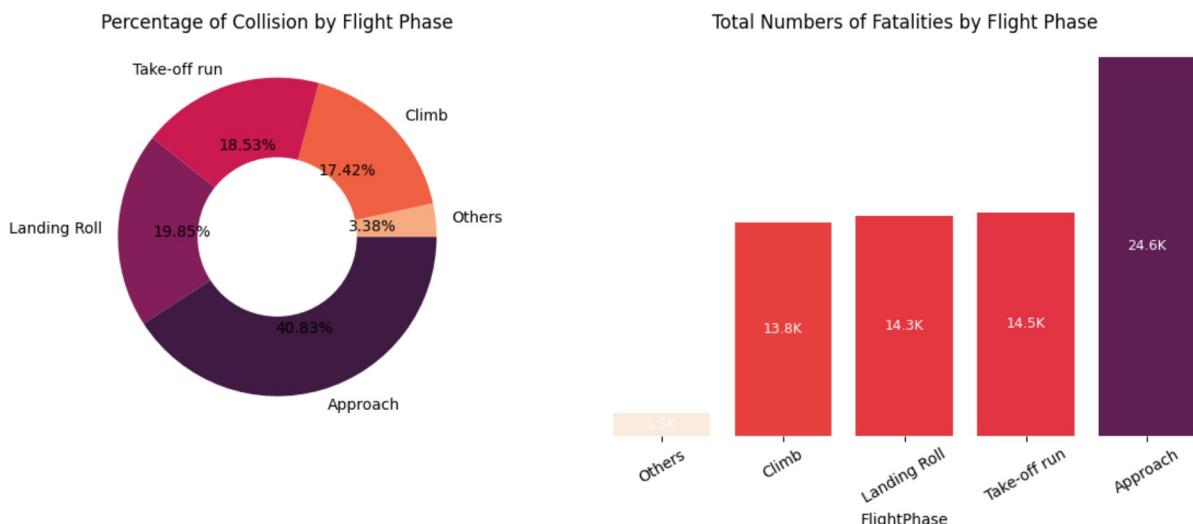
```

for bar, value in zip(bars.patches, agg_data_bar.values):
    height = bar.get_height()
    bars.text(
        bar.get_x() + bar.get_width() / 2,
        height / 2,
        f'{value / 1e3:.1f}K',
        ha='center',
        va='center',
        fontsize=9,
        color='white'
    )
    bar.set_facecolor(cmap(norm(height)*0.75))

plt.sca(ax[1])
plt.title('Total Numbers of Fatalities by Flight Phase', fontsize=12)
plt.yticks([])
plt.ylabel('')
plt.xticks(rotation=30)
plt.box(False)

plt.tight_layout()
plt.show()

```



```

In [32]: fig, ax = plt.subplots(figsize=(8,4))

bin_alt = pd.qcut(data[data.Altitude > 5]['Altitude'], q=20, duplicates='drop')
agg_data = data.groupby(bin_alt)['NumberStruckActual'].sum()
norm = plt.Normalize(agg_data.min(), agg_data.max())
cmap = cm.get_cmap('rocket_r')

bars = sns.barplot(x=agg_data.index, y=agg_data, ax=ax, palette='rocket_r')

# Añadir etiquetas manualmente
for bar, value in zip(bars.patches, agg_data):
    height = bar.get_height()
    bars.text(
        bar.get_x() + bar.get_width() / 2,
        height / 2,
        f'{value / 1e3:.1f}K',
        ha='center',
        va='center',
        fontsize=9,
        color='white'
    )

```

```
        ha='center',
        va='center',
        fontsize=9,
        color='white'
    )
    bar.set_facecolor(cmap(norm(height)*0.75))

plt.title('Fatalities by Altitude', fontsize=12)
plt.yticks([])
plt.xticks(rotation=30)
plt.xlabel('')
plt.tight_layout()
plt.show()

fig, ax = plt.subplots(figsize=(8,4))

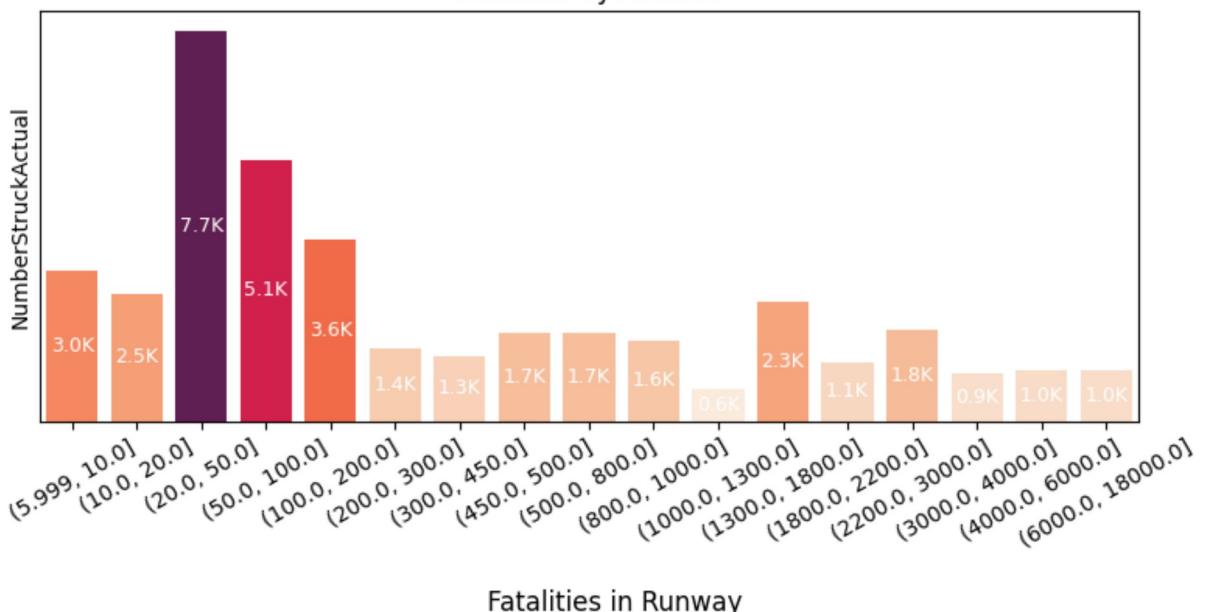
agg_data = data[data.Altitude <= 5].groupby('FlightPhase')[['NumberStruckActual']].sum()
norm = plt.Normalize(agg_data.min(), agg_data.max())
cmap = cm.get_cmap('rocket_r')

bars = sns.barplot(x=agg_data.index, y=agg_data, ax=ax, palette='rocket_r')

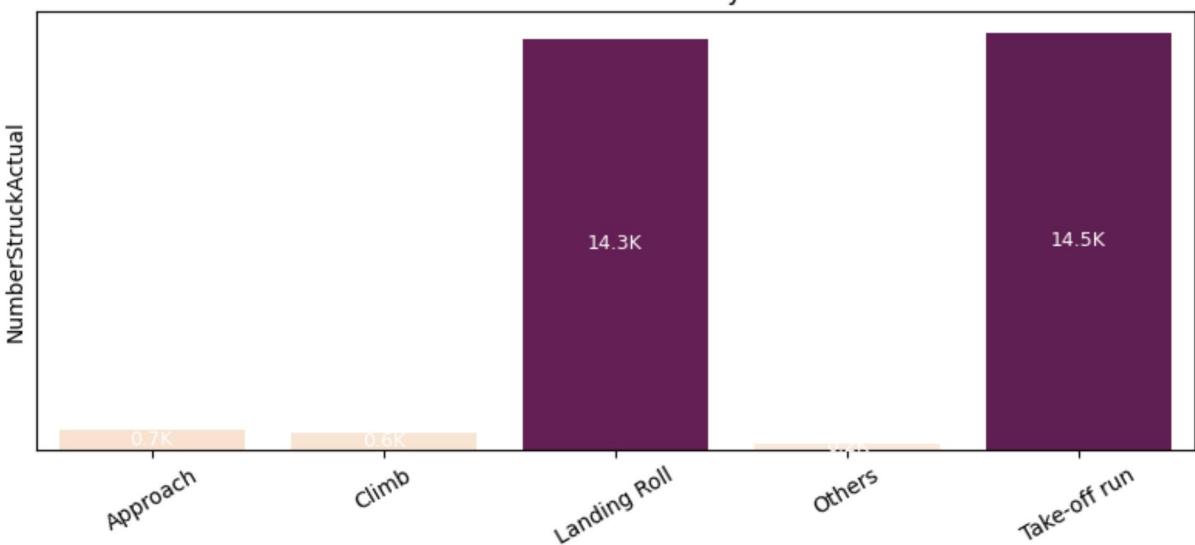
# Añadir etiquetas manualmente
for bar, value in zip(bars.patches, agg_data):
    height = bar.get_height()
    bars.text(
        bar.get_x() + bar.get_width() / 2,
        height / 2,
        f'{value / 1e3:.1f}K',
        ha='center',
        va='center',
        fontsize=9,
        color='white'
    )
    bar.set_facecolor(cmap(norm(height)*0.75))

plt.title('Fatalities in Runway', fontsize=12)
plt.yticks([])
plt.xticks(rotation=30)
plt.xlabel('')
plt.tight_layout()
plt.show()
```

Fatalities by Altitude



Fatalities in Runway

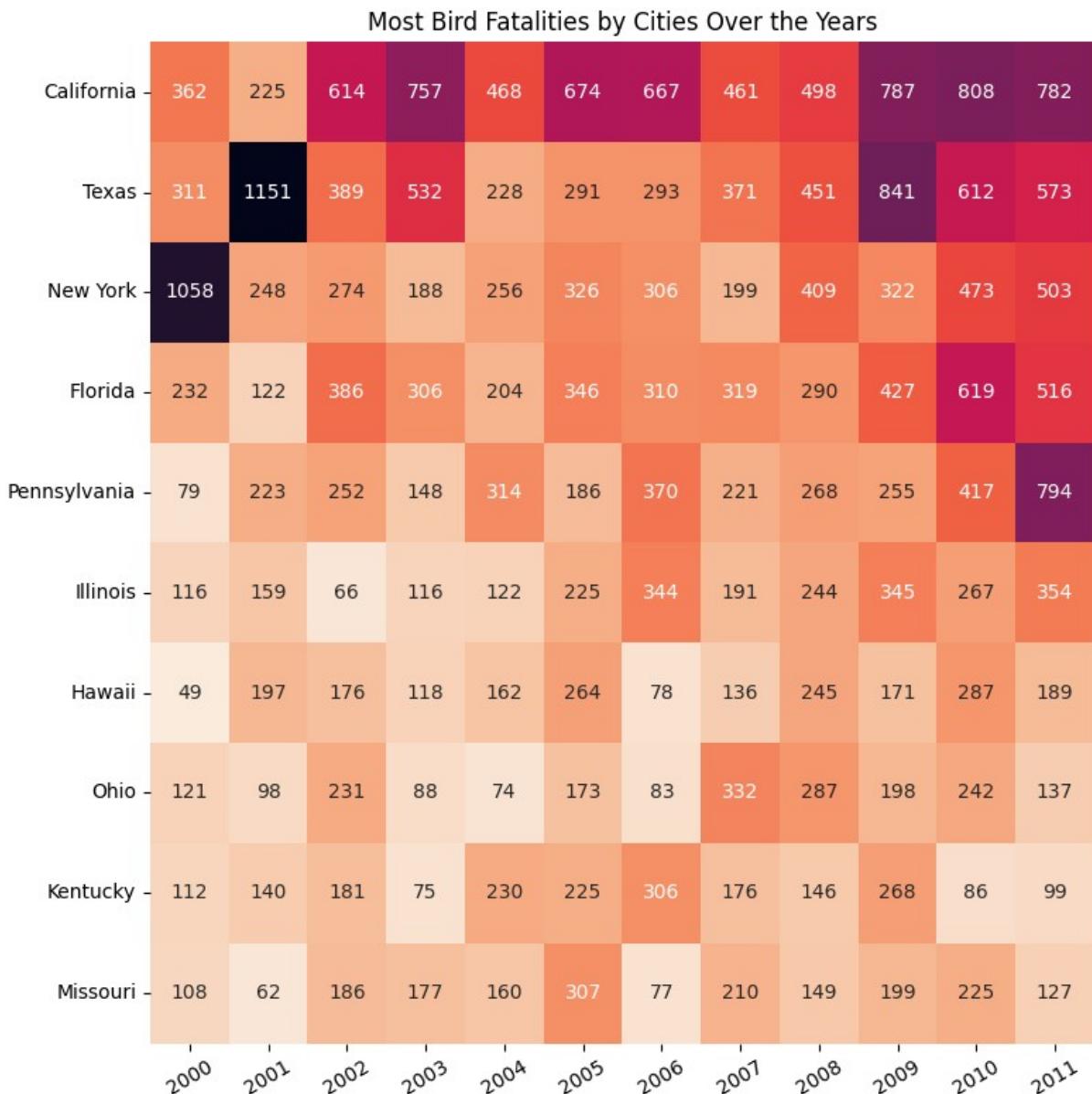


```
In [33]: fig,ax = plt.subplots(figsize=(8,8))

agg_data = data.groupby(['OriginState',data.FlightDate.dt.year])['NumberStruckActual'].sum().sort_values(ascending=False).index[:10]
sns.heatmap(agg_data.loc[idx],ax=ax,annot=True,cmap='rocket_r',cbar=False,fmt='g')

plt.title('Most Bird Fatalities by Cities Over the Years',fontsize=12)
plt.xticks(rotation=30)
plt.xlabel('')
plt.ylabel('')

plt.tight_layout()
plt.show()
```



Modelo de Clasificación para predecir daño

Modelo de Clasificación para Predecir Daño

Esta sección muestra la implementación de un modelo de clasificación (Regresión Logística) para predecir la variable categórica `Damage` (daño causado o no), considerando variables como `NumberStruckActual`, `IsAircraftLarge?`, `Engines`, `WildlifeSize`, `FlightPhase` y `OriginState`. Se incluyen codificación de variables, división de datos, entrenamiento y evaluación del modelo.

```
In [60]: # 1. Preparar datos
X_cls = data[['NumberStruckActual', 'IsAircraftLarge?', 'Engines', 'WildlifeSize',
y_cls = data['Damage'].map({'No damage': 0, 'Caused damage': 1})

# 2. Codificar variables categóricas
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
cat_features = ['IsAircraftLarge?', 'WildlifeSize', 'FlightPhase', 'OriginState']
ct_cls = ColumnTransformer([('enc', OneHotEncoder(drop='first'), cat_features)], re
X_encoded_cls = ct_cls.fit_transform(X_cls)

# 3. División de datos
from sklearn.model_selection import train_test_split
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(X_encoded_cls,

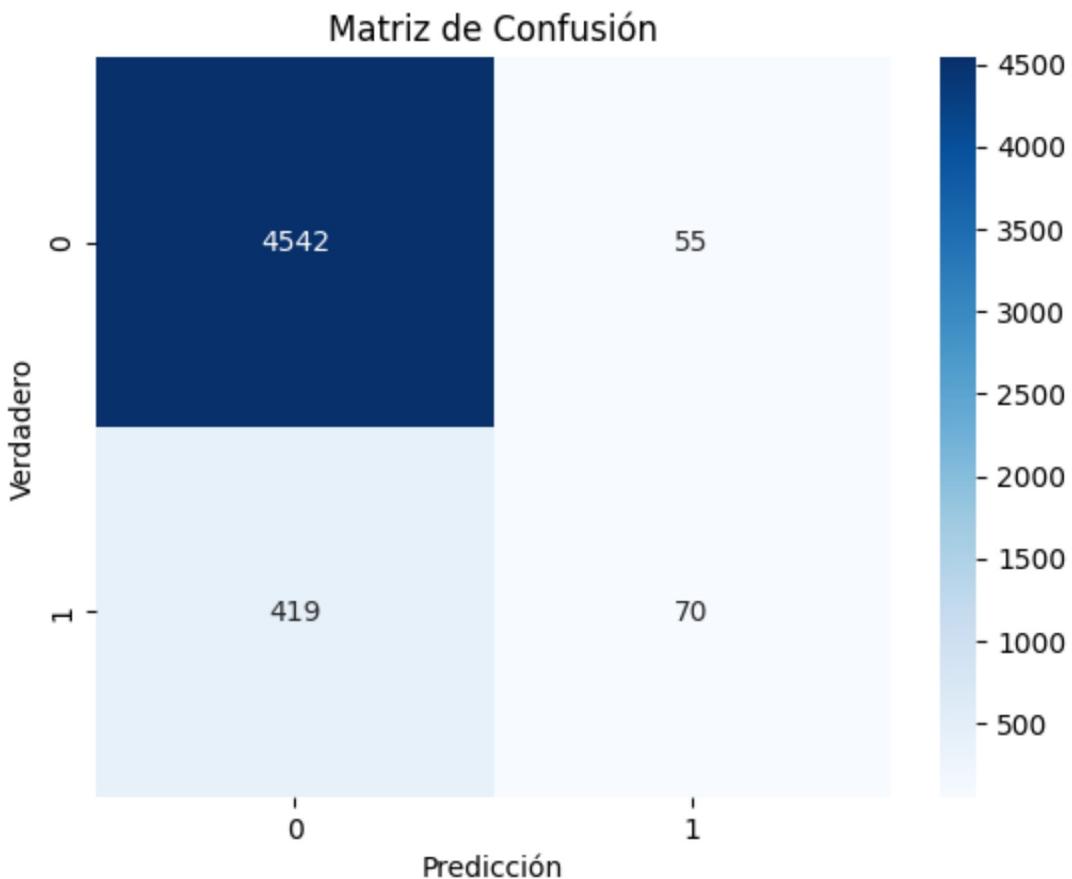
# 4. Entrenamiento con Regresión Logística
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train_cls, y_train_cls)

# 5. Predicción y evaluación
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
y_pred_cls = clf.predict(X_test_cls)
print("Accuracy:", accuracy_score(y_test_cls, y_pred_cls))
print(classification_report(y_test_cls, y_pred_cls))

# Matriz de confusión
import seaborn as sns
import matplotlib.pyplot as plt
conf_mat = confusion_matrix(y_test_cls, y_pred_cls)
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
plt.title('Matriz de Confusión')
plt.xlabel('Predicción')
plt.ylabel('Verdadero')
plt.show()
```

Accuracy: 0.9068029885961463

	precision	recall	f1-score	support
0	0.92	0.99	0.95	4597
1	0.56	0.14	0.23	489
accuracy			0.91	5086
macro avg	0.74	0.57	0.59	5086
weighted avg	0.88	0.91	0.88	5086



Mejoras del Modelo

Para mejorar el desempeño en la clase minoritaria (Damage = 1), se pueden aplicar técnicas como balanceo de datos (SMOTE), ajuste de pesos de clase, exploración de modelos más complejos (Random Forest, XGBoost) y búsqueda de hiperparámetros.

```
In [63]: from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Aplicar SMOTE
sm = SMOTE(random_state=0)
X_res, y_res = sm.fit_resample(X_encoded_cls, y_cls)

# División de datos balanceados
X_train_sm, X_test_sm, y_train_sm, y_test_sm = train_test_split(X_res, y_res, test_size=0.2, random_state=42)

# Definir RandomForest con peso de clases balanceado
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
param_grid = {'n_estimators': [100, 200], 'max_depth': [None, 10, 20]}
grid = GridSearchCV(rf, param_grid, cv=3, scoring='f1', n_jobs=-1)
grid.fit(X_train_sm, y_train_sm)
```

```
# Evaluar mejor modelo
best_rf = grid.best_estimator_
y_pred_rf = best_rf.predict(X_test_sm)
print('Classification Report - RandomForest SMOTE')
print(classification_report(y_test_sm, y_pred_rf))
cm = confusion_matrix(y_test_sm, y_pred_rf)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Matriz de Confusión - RandomForest SMOTE')
plt.show()

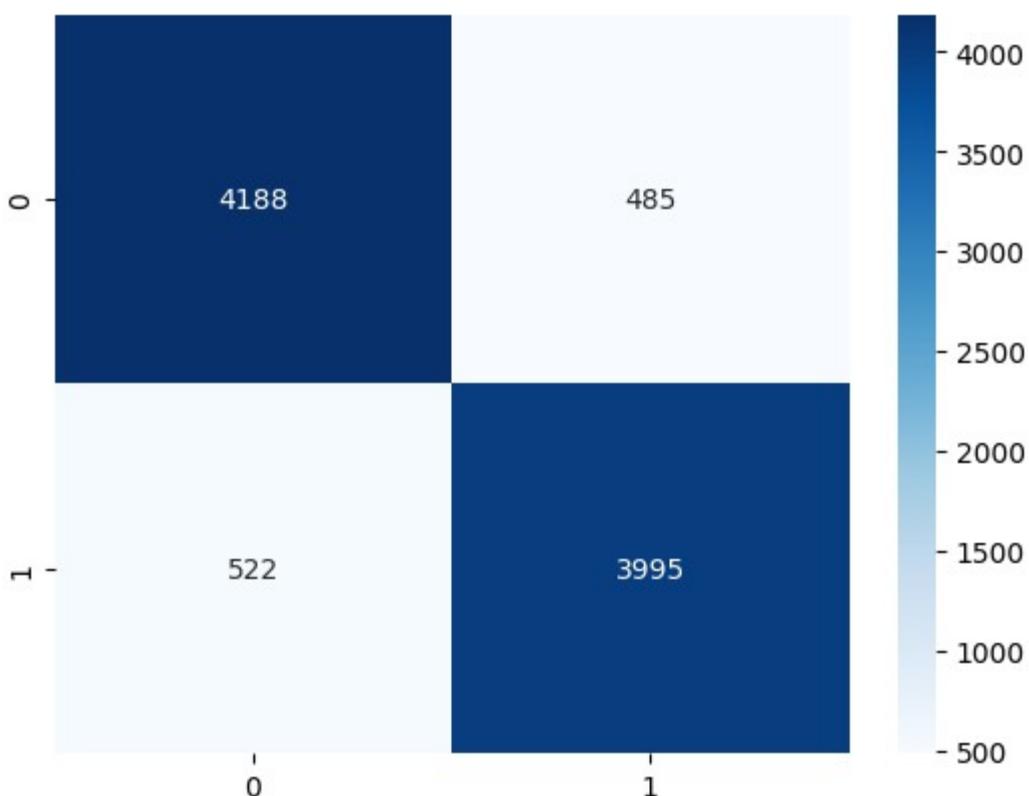
# Imprimir y graficar métricas adicionales para el modelo RandomForest SMOTE
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score,
                           accuracy_score, confusion_matrix, classification_report

# Precision, Recall, F1-score
precision = precision_score(y_test_sm, y_pred_rf)
recall = recall_score(y_test_sm, y_pred_rf)
f1 = f1_score(y_test_sm, y_pred_rf)
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"F1-score: {f1:.3f}")

# ROC AUC
if hasattr(best_rf, "predict_proba"):
    y_proba = best_rf.predict_proba(X_test_sm)[:,1]
    roc_auc = roc_auc_score(y_test_sm, y_proba)
    print(f"ROC AUC: {roc_auc:.3f}")
    # Curva ROC
    fpr, tpr, _ = roc_curve(y_test_sm, y_proba)
    plt.figure()
    plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0,1],[0,1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Curva ROC')
    plt.legend()
    plt.show()
    # Curva Precision-Recall
    precision_vals, recall_vals, _ = precision_recall_curve(y_test_sm, y_proba)
    pr_auc = auc(recall_vals, precision_vals)
    print(f"PR AUC: {pr_auc:.3f}")
    plt.figure()
    plt.plot(recall_vals, precision_vals, label=f'PR curve (AUC = {pr_auc:.2f})')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Curva Precision-Recall')
    plt.legend()
    plt.show()
else:
    print("El modelo no soporta predict_proba para curvas ROC/PR.")
```

Classification Report - RandomForest SMOTE				
	precision	recall	f1-score	support
0	0.89	0.90	0.89	4673
1	0.89	0.88	0.89	4517
accuracy			0.89	9190
macro avg	0.89	0.89	0.89	9190
weighted avg	0.89	0.89	0.89	9190

Matriz de Confusión - RandomForest SMOTE



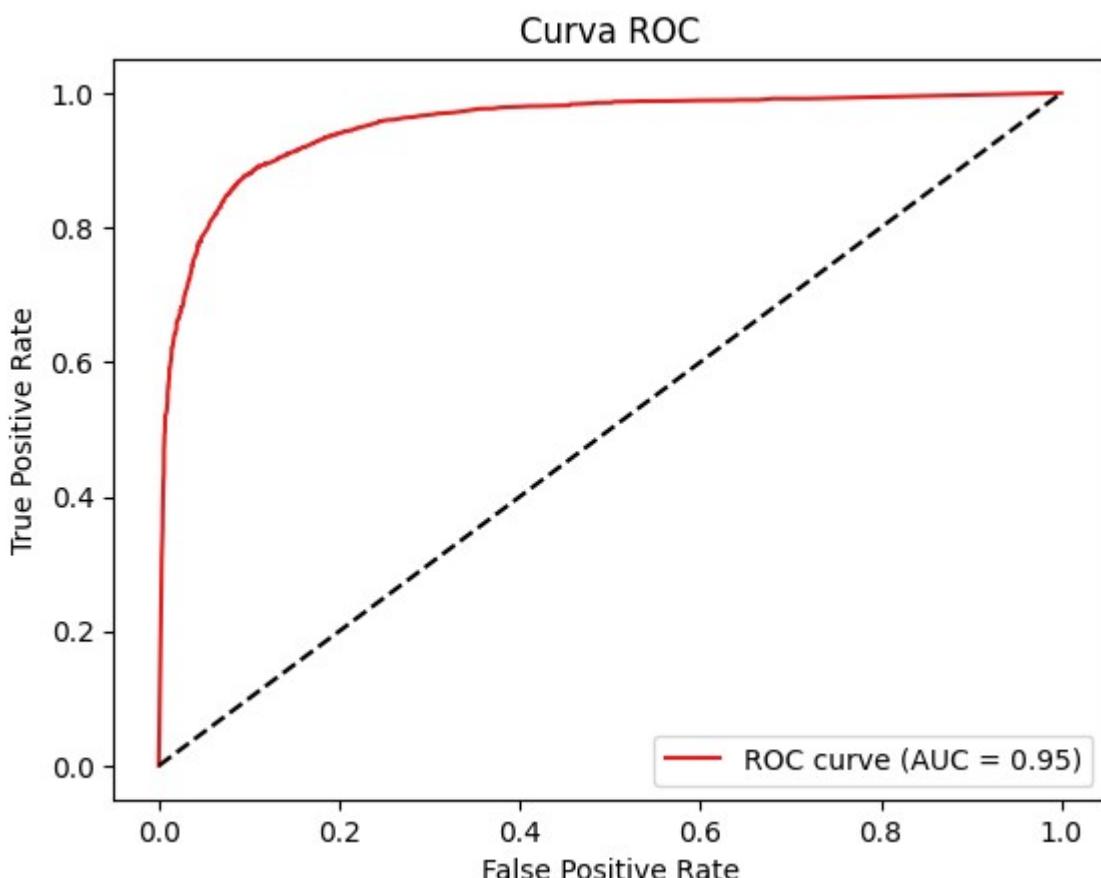
Precision: 0.892

Recall: 0.884

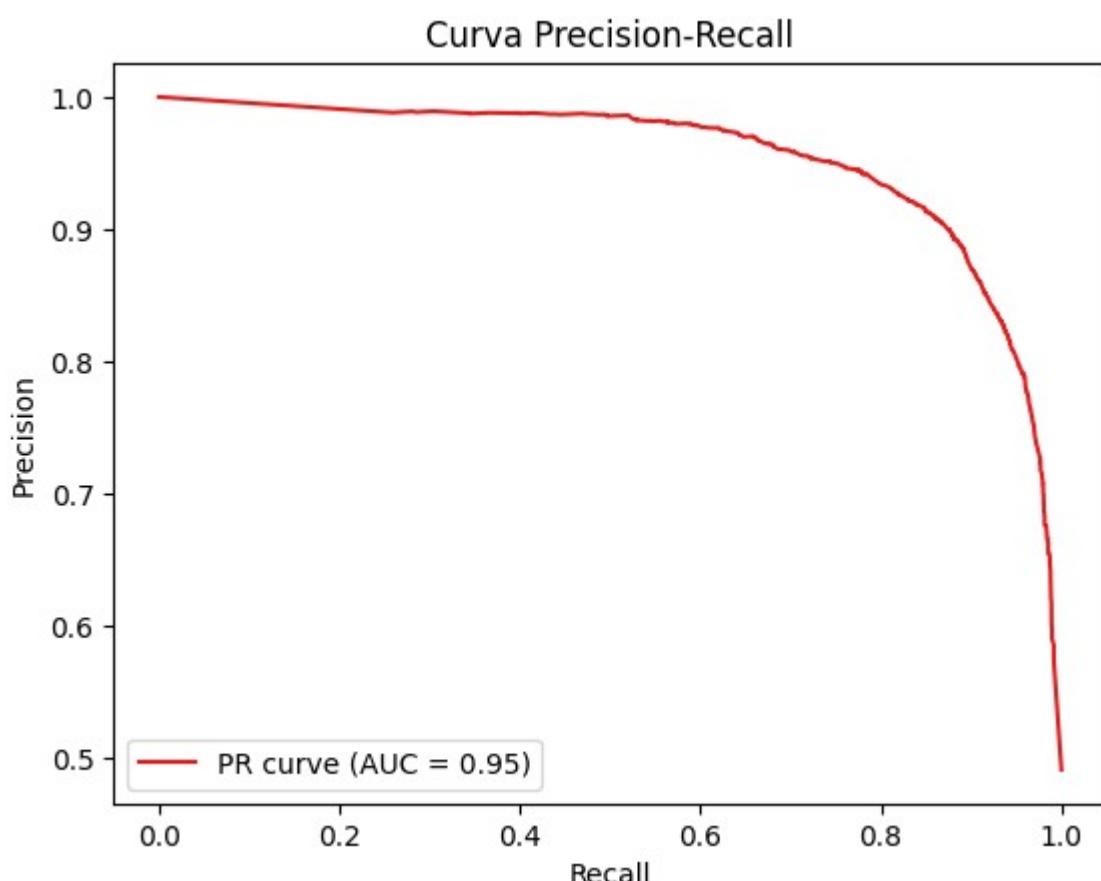
F1-score: 0.888

ROC AUC: 0.953

ROC AUC: 0.953



PR AUC: 0.953



Análisis de Resultados de los Modelos de Clasificación

A continuación se presenta un análisis comparativo de los resultados obtenidos con los dos enfoques de clasificación utilizados para predecir el daño causado por colisiones con aves:

1. Regresión Logística (sin balanceo)

- **Accuracy:** 0.91
- **Precision clase 1 (daño):** 0.56
- **Recall clase 1 (daño):** 0.14
- **F1-score clase 1 (daño):** 0.23

Interpretación:

- El modelo tiene un buen accuracy general, pero esto se debe a que la clase mayoritaria (sin daño) domina el dataset.
- La precisión para la clase de daño es baja, pero el recall es especialmente bajo (solo identifica correctamente el 14% de los casos de daño real), lo que indica que el modelo no es útil para detectar incidentes con daño.
- El F1-score bajo para la clase minoritaria confirma que el modelo no es adecuado para este objetivo.

2. RandomForest con SMOTE (balanceo de clases)

- **Accuracy:** 0.89
- **Precision clase 1 (daño):** 0.89
- **Recall clase 1 (daño):** 0.88
- **F1-score clase 1 (daño):** 0.89
- **ROC AUC:** 0.953

Interpretación:

- El accuracy general es ligeramente menor, pero ahora el modelo es mucho más equilibrado y efectivo para ambas clases.
- La precisión y el recall para la clase de daño son muy altos y balanceados, lo que significa que el modelo identifica correctamente la mayoría de los casos de daño y comete pocos falsos positivos.
- El F1-score alto indica un excelente balance entre precisión y recall.
- El área bajo la curva ROC (ROC AUC) de 0.95 muestra que el modelo tiene una gran capacidad para distinguir entre incidentes con y sin daño.

Conclusión General

- El uso de técnicas de balanceo como SMOTE y modelos más robustos como RandomForest mejora drásticamente la capacidad del modelo para detectar incidentes de daño, que es el objetivo principal en este contexto.
- Para problemas con clases desbalanceadas, no basta con observar el accuracy: es fundamental analizar métricas como precision, recall, F1-score y ROC AUC, especialmente para la clase minoritaria.
- El modelo RandomForest con SMOTE es claramente superior para la predicción de daños en colisiones aéreas con aves y es la opción recomendada para aplicaciones prácticas.