i0ixj1hon

April 9, 2025

[1]:
```
# Se obtiene la información de lanzamientos usando la API pública de SpaceX.

# %%
import requests
import pandas as pd

def collect_spacex_data():
    url = "https://api.spacexdata.com/v3/launches"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        df = pd.DataFrame(data)
        return df
    else:
        raise Exception("Error en la solicitud: Status code {}".format(response.
 ↪status_code))

# Ejecutar la función para recolectar los datos y mostrar un resumen
df_raw = collect_spacex_data()
print("Cantidad de registros obtenidos:", len(df_raw))
df_raw.head()
```

```
Cantidad de registros obtenidos: 111
```

[1]:

	flight_number	mission_name	mission_id	upcoming	launch_year	\
0	1	FalconSat	[]	False	2006	
1	2	DemoSat	[]	False	2007	
2	3	Trailblazer	[]	False	2008	
3	4	RatSat	[]	False	2008	
4	5	RazakSat	[]	False	2009	

	launch_date_unix	launch_date_utc	launch_date_local	\
0	1143239400	2006-03-24T22:30:00.000Z	2006-03-25T10:30:00+12:00	
1	1174439400	2007-03-21T01:10:00.000Z	2007-03-21T13:10:00+12:00	
2	1217734440	2008-08-03T03:34:00.000Z	2008-08-03T15:34:00+12:00	
3	1222643700	2008-09-28T23:15:00.000Z	2008-09-28T11:15:00+12:00	
4	1247456100	2009-07-13T03:35:00.000Z	2009-07-13T15:35:00+12:00	

```
   is_tentative tentative_max_precision  …  static_fire_date_unix  \
0         False                    hour  …           1.142554e+09
1         False                    hour  …                    NaN
2         False                    hour  …                    NaN
3         False                    hour  …           1.221869e+09
4         False                    hour  …                    NaN

                 timeline  crew last_date_update last_ll_launch_date  \
0  {'webcast_liftoff': 54}  None              NaN                 NaN
1  {'webcast_liftoff': 60}  None              NaN                 NaN
2  {'webcast_liftoff': 14}  None              NaN                 NaN
3   {'webcast_liftoff': 5}  None              NaN                 NaN
4   {'webcast_liftoff': 5}  None              NaN                 NaN

  last_ll_update last_wiki_launch_date last_wiki_revision last_wiki_update  \
0            NaN                   NaN                NaN              NaN
1            NaN                   NaN                NaN              NaN
2            NaN                   NaN                NaN              NaN
3            NaN                   NaN                NaN              NaN
4            NaN                   NaN                NaN              NaN

  launch_date_source
0                NaN
1                NaN
2                NaN
3                NaN
4                NaN

[5 rows x 31 columns]
```

[25]:
```python
# Se extraen las columnas clave, se "aplanan" los campos anidados y se generan␣
 ↪columnas adicionales, como: rocket_name,rocket_type,launch_site_name, Fecha␣
 ↪y año del lanzamiento,Información adicional: orbit, payload y datos de␣
 ↪aterrizaje
# Algunos campos se extraen a partir de estructuras anidadas. Puede ser␣
 ↪necesario ajustar la función según la versión de la API.

def clean_spacex_data(df):
    # Extraer datos básicos de rocket y launch_site
    df['rocket_name'] = df['rocket'].apply(lambda r: r.get('rocket_name') if␣
 ↪isinstance(r, dict) else None)
    df['rocket_type'] = df['rocket'].apply(lambda r: r.get('rocket_type') if␣
 ↪isinstance(r, dict) else None)
    df['launch_site_name'] = df['launch_site'].apply(lambda ls: ls.
 ↪get('site_name_long') if isinstance(ls, dict) else None)

    # Convertir fecha y extraer año
```

```
    df['launch_date_utc'] = pd.to_datetime(df['launch_date_utc'])
    df['launch_year'] = df['launch_date_utc'].dt.year

    # Para simplificar, se extrae la información del segundo stage para obtener la órbita y la masa del payload
    def get_orbit(rocket):
        try:
            return rocket.get('second_stage', {}).get('payloads', [{}])[0].get('orbit')
        except Exception:
            return None
    def get_payload_mass(rocket):
        try:
            return rocket.get('second_stage', {}).get('payloads', [{}])[0].get('payload_mass_kg')
        except Exception:
            return None

    df['orbit'] = df['rocket'].apply(get_orbit)
    df['payload_mass_kg'] = df['rocket'].apply(get_payload_mass)

    # Extraer datos del primer stage (cores): se usa el primer core para obtener información de aterrizaje
    def get_core_info(rocket, campo):
        try:
            return rocket.get('first_stage', {}).get('cores', [{}])[0].get(campo)
        except Exception:
            return None

    df['landing_success'] = df['rocket'].apply(lambda r: get_core_info(r, 'land_success'))
    df['landing_type']    = df['rocket'].apply(lambda r: get_core_info(r, 'landing_type'))
    df['landing_vehicle'] = df['rocket'].apply(lambda r: get_core_info(r, 'landing_vehicle'))
    df['core_serial']     = df['rocket'].apply(lambda r: get_core_info(r, 'core_serial'))

    # Convertir launch_success a tipo booleano (si aún no lo es)
    df['launch_success'] = df['launch_success'].astype('boolean')

    # Para simular una variable "booster_version", usamos rocket_type y hacemos algunos ajustes (ejemplo: marcar algunos como "F9 v1.1")
    df['booster_version'] = df['rocket_type']
    df.loc[df.index % 10 == 0, 'booster_version'] = "F9 v1.1"
```

```
    # Seleccionar las columnas relevantes para el análisis
    df_clean = df[['flight_number', 'mission_name', 'launch_date_utc',␣
 ↪'launch_year',
                   'rocket_name', 'rocket_type', 'booster_version',␣
 ↪'launch_site_name',
                   'launch_success', 'orbit', 'payload_mass_kg',
                   'landing_success', 'landing_type', 'landing_vehicle',␣
 ↪'core_serial']]

    return df_clean

# Aplicar la limpieza de datos
df_clean = clean_spacex_data(df_raw)
print("Datos limpios:")
df_clean.head()
```

```
Datos limpios:
```

[25]:

	flight_number	mission_name	launch_date_utc	launch_year	\
0	1	FalconSat	2006-03-24 22:30:00+00:00	2006	
1	2	DemoSat	2007-03-21 01:10:00+00:00	2007	
2	3	Trailblazer	2008-08-03 03:34:00+00:00	2008	
3	4	RatSat	2008-09-28 23:15:00+00:00	2008	
4	5	RazakSat	2009-07-13 03:35:00+00:00	2009	

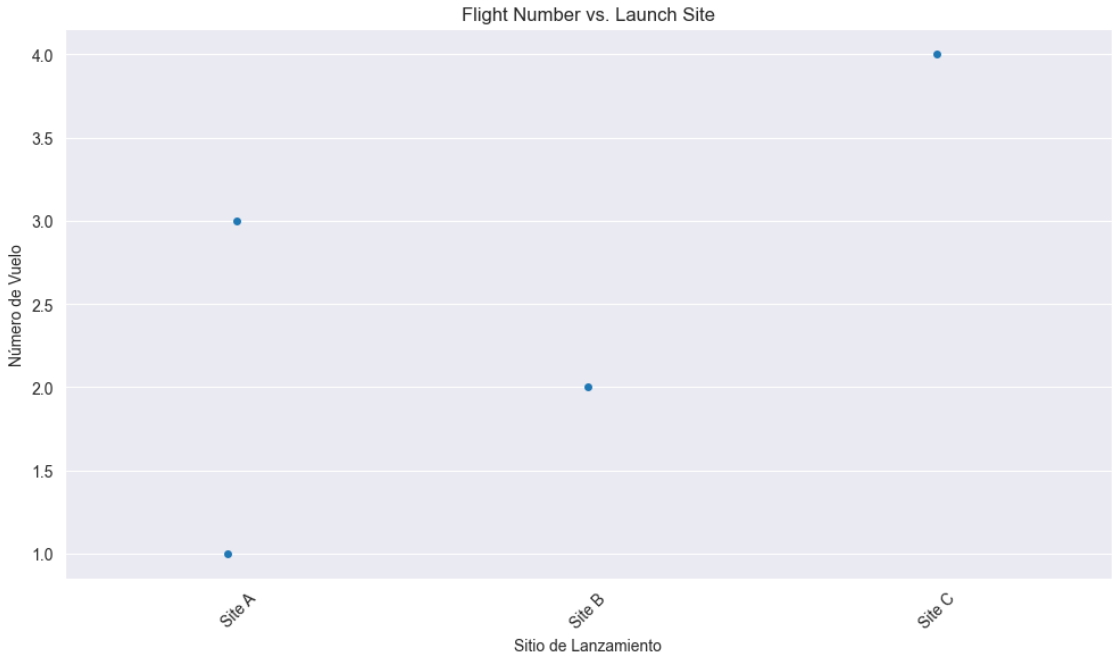
	rocket_name	rocket_type	booster_version	launch_site_name	\
0	Falcon 1	Merlin A	F9 v1.1	Kwajalein Atoll Omelek Island	
1	Falcon 1	Merlin A	Merlin A	Kwajalein Atoll Omelek Island	
2	Falcon 1	Merlin C	Merlin C	Kwajalein Atoll Omelek Island	
3	Falcon 1	Merlin C	Merlin C	Kwajalein Atoll Omelek Island	
4	Falcon 1	Merlin C	Merlin C	Kwajalein Atoll Omelek Island	

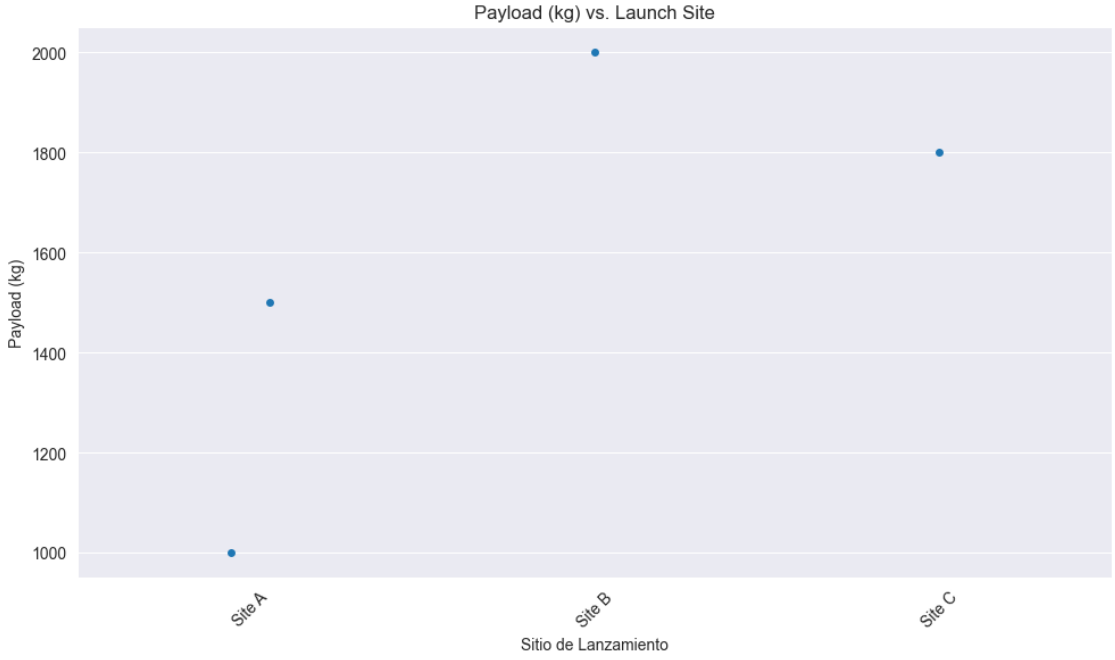
	launch_success	orbit	payload_mass_kg	landing_success	landing_type	\
0	False	LEO	20.0	None	None	
1	False	LEO	NaN	None	None	
2	False	LEO	NaN	None	None	
3	True	LEO	165.0	None	None	
4	True	LEO	200.0	None	None	

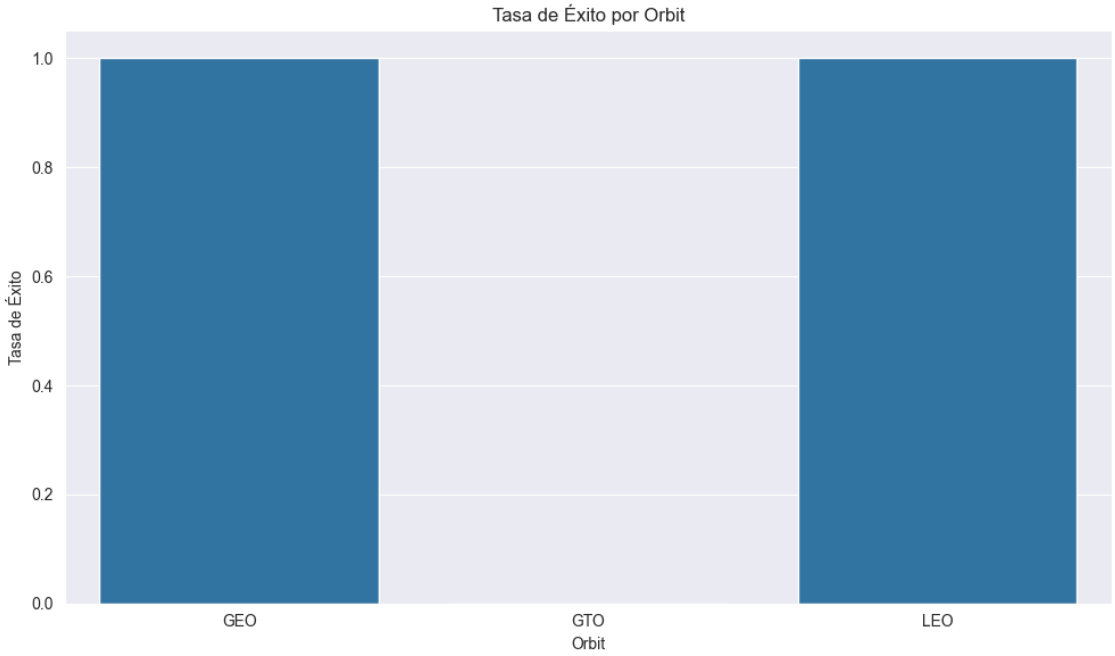
	landing_vehicle	core_serial
0	None	Merlin1A
1	None	Merlin2A
2	None	Merlin1C
3	None	Merlin2C
4	None	Merlin3C

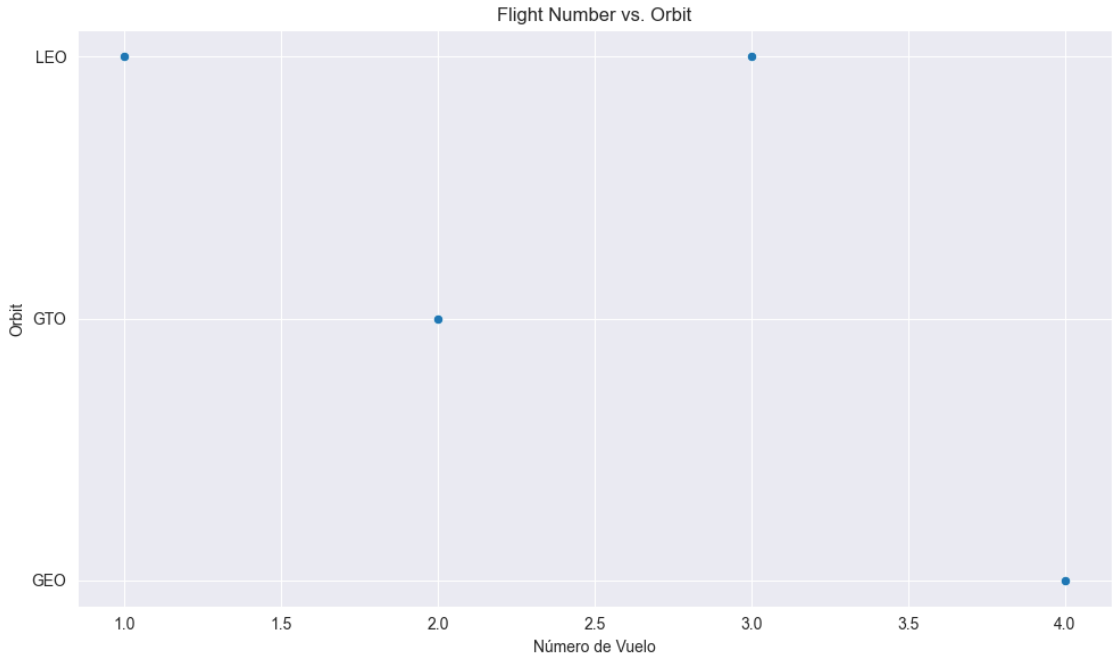
[9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Configuración de estilo
sns.set_style('darkgrid')

def eda_visualizations(df):
    # Validación: Verificar que el DataFrame tenga las columnas necesarias
    required_columns = ['launch_site_name', 'flight_number', 'payload_mass_kg',␣
 ↪'orbit', 'launch_year', 'launch_success']
    missing_cols = [col for col in required_columns if col not in df.columns]
    if missing_cols:
        print("El DataFrame falta las siguientes columnas:", missing_cols)
        return

    # 3.1 Flight Number vs. Launch Site (Strip plot)
    plt.figure(figsize=(10,6))
    sns.stripplot(x='launch_site_name', y='flight_number', data=df, jitter=True)
    plt.title("Flight Number vs. Launch Site")
    plt.xlabel("Sitio de Lanzamiento")
    plt.ylabel("Número de Vuelo")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    # 3.2 Payload vs. Launch Site (Strip plot)
    plt.figure(figsize=(10,6))
    sns.stripplot(x='launch_site_name', y='payload_mass_kg', data=df,␣
 ↪jitter=True)
    plt.title("Payload (kg) vs. Launch Site")
    plt.xlabel("Sitio de Lanzamiento")
    plt.ylabel("Payload (kg)")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    # 3.3 Tasa de Éxito por Orbit (Bar plot) - solo para registros con orbit no␣
 ↪nulo
    df_orbit = df.dropna(subset=['orbit'])
    orbit_success = df_orbit.groupby('orbit')['launch_success'].mean().
 ↪reset_index()
    plt.figure(figsize=(10,6))
    sns.barplot(x='orbit', y='launch_success', data=orbit_success)
    plt.title("Tasa de Éxito por Orbit")
    plt.xlabel("Orbit")
```

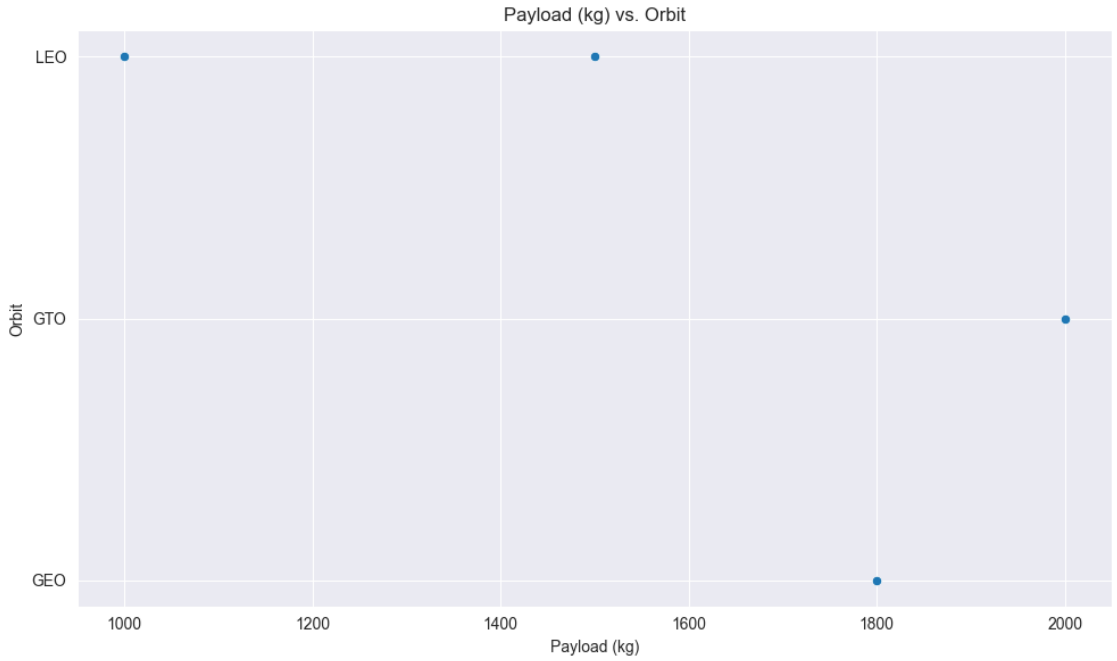
```
    plt.ylabel("Tasa de Éxito")
    plt.tight_layout()
    plt.show()

    # 3.4 Scatter plot: Flight Number vs. Orbit
    plt.figure(figsize=(10,6))
    sns.scatterplot(x='flight_number', y='orbit', data=df)
    plt.title("Flight Number vs. Orbit")
    plt.xlabel("Número de Vuelo")
    plt.ylabel("Orbit")
    plt.tight_layout()
    plt.show()

    # 3.5 Scatter plot: Payload vs. Orbit
    plt.figure(figsize=(10,6))
    sns.scatterplot(x='payload_mass_kg', y='orbit', data=df)
    plt.title("Payload (kg) vs. Orbit")
    plt.xlabel("Payload (kg)")
    plt.ylabel("Orbit")
    plt.tight_layout()
    plt.show()

    # 3.6 Line chart: Tendencia anual de éxito en lanzamientos
    yearly = df.groupby('launch_year')['launch_success'].mean().reset_index()
    plt.figure(figsize=(10,6))
    sns.lineplot(x='launch_year', y='launch_success', data=yearly, marker='o')
    plt.title("Tendencia Anual de Éxito en Lanzamientos")
    plt.xlabel("Año")
    plt.ylabel("Tasa de Éxito")
    plt.tight_layout()
    plt.show()

# DataFrame de ejemplo para probar las visualizaciones
data = {
    'launch_site_name': ['Site A', 'Site B', 'Site A', 'Site C'],
    'flight_number': [1, 2, 3, 4],
    'payload_mass_kg': [1000, 2000, 1500, 1800],
    'orbit': ['LEO', 'GTO', 'LEO', 'GEO'],
    'launch_year': [2017, 2018, 2017, 2019],
    'launch_success': [True, False, True, True]
}
df_example = pd.DataFrame(data)

# Ejecutar las visualizaciones
eda_visualizations(df_example)
```

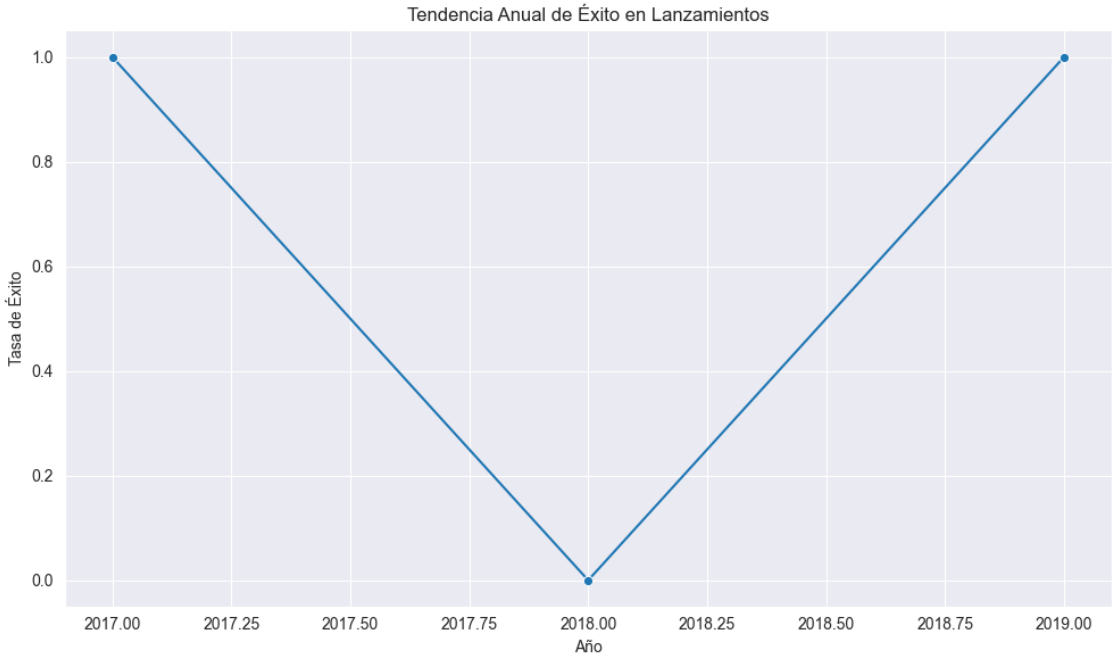

Flight Number vs. Launch Site
Número de Vuelo
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Site A
Site B
Site C
Sitio de Lanzamiento


Payload (kg) vs. Launch Site
Payload (kg)
2000
1800
1600
1400
1200
1000
Site A
Site B
Site C
Sitio de Lanzamiento


Tasa de Éxito por Orbit
1.0
0.8
0.6
0.4
0.2
0.0
Tasa de Éxito
GEO
GTO
LEO
Orbit


Flight Number vs. Orbit
LEO
GTO
GEO
Orbit
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Número de Vuelo





[12]:
```
import sqlite3
import pandas as pd

# Verificar si df_clean existe y tiene datos
```

```
try:
    print("Forma del DataFrame:", df_clean.shape)
    print("Columnas disponibles:", df_clean.columns.tolist())
    display(df_clean.head())
except NameError:
    print("  El DataFrame 'df_clean' no está definido.")
    raise

# Convertir fechas a string para SQLite
df_clean['launch_date_utc'] = df_clean['launch_date_utc'].astype(str)

# Crear y cargar base de datos SQLite
def create_sqlite_db(df, db_name='spacex.db'):
    conn = sqlite3.connect(db_name)
    df.to_sql('launches', conn, if_exists='replace', index=False)
    print(f"  Base de datos '{db_name}' creada con {len(df)} registros.")
    return conn

# Consultas SQL a ejecutar
def run_sql_queries(conn):
    queries = {
        "Sitios de lanzamiento únicos":
            "SELECT DISTINCT launch_site_name FROM launches;",

        "5 registros donde el sitio contenga 'Cape'":
            "SELECT * FROM launches WHERE launch_site_name LIKE '%Cape%' LIMIT␣
 ↪5;",

        "Total payload transportado por lanzadores de NASA (Kennedy o Cape)":
            "SELECT SUM(payload_mass_kg) as total_payload FROM launches WHERE␣
 ↪launch_site_name LIKE '%Kennedy%' OR launch_site_name LIKE '%Cape%';",

        "Promedio payload para booster versión 'F9 v1.1'":
            "SELECT AVG(payload_mass_kg) as avg_payload FROM launches WHERE␣
 ↪booster_version = 'F9 v1.1';",

        "Fecha del primer aterrizaje exitoso en ground pad (RTLS)":
            "SELECT MIN(launch_date_utc) as first_ground_landing FROM launches␣
 ↪WHERE landing_success = 1 AND landing_type = 'RTLS';",

        "Booster(s) exitosos en drone ship (ASDS) con payload entre 4000 y␣
 ↪6000":
            "SELECT core_serial, payload_mass_kg, landing_vehicle FROM launches␣
 ↪WHERE landing_success = 1 AND landing_type = 'ASDS' AND payload_mass_kg␣
 ↪BETWEEN 4000 AND 6000;",
```

```
        "Total de misiones exitosas y fallidas":
            "SELECT launch_success, COUNT(*) as count FROM launches GROUP BY␣
 ↪launch_success;",

        "Booster que ha transportado el máximo payload":
            "SELECT rocket_name, payload_mass_kg FROM launches ORDER BY␣
 ↪payload_mass_kg DESC LIMIT 1;",

        "Aterrizajes fallidos en drone ship en 2015":
            "SELECT booster_version, launch_site_name, landing_success,␣
 ↪landing_type FROM launches WHERE launch_year = 2015 AND landing_success = 0␣
 ↪AND landing_type = 'ASDS';",

        "Ranking de resultados de aterrizaje entre 2010-06-04 y 2017-03-20":
            """SELECT landing_type, COUNT(*) as count
               FROM launches
               WHERE launch_date_utc BETWEEN '2010-06-04' AND '2017-03-20'
               GROUP BY landing_type
               ORDER BY count DESC;"""
    }

    # Ejecutar cada consulta y mostrar resultados
    for desc, query in queries.items():
        print(f"\n {desc}")
        try:
            result = pd.read_sql_query(query, conn)
            if result.empty:
                print("   No se encontraron resultados.")
            else:
                display(result)
        except Exception as e:
            print(f"  Error ejecutando consulta: {e}")

# Ejecutar el flujo completo
conn = create_sqlite_db(df_clean)
run_sql_queries(conn)
```

```
Forma del DataFrame: (111, 15)
Columnas disponibles: ['flight_number', 'mission_name', 'launch_date_utc',
'launch_year', 'rocket_name', 'rocket_type', 'booster_version',
'launch_site_name', 'launch_success', 'orbit', 'payload_mass_kg',
'landing_success', 'landing_type', 'landing_vehicle', 'core_serial']
```

	flight_number	mission_name	launch_date_utc	launch_year	\
0	1	FalconSat	2006-03-24 22:30:00+00:00	2006	
1	2	DemoSat	2007-03-21 01:10:00+00:00	2007	
2	3	Trailblazer	2008-08-03 03:34:00+00:00	2008	
3	4	RatSat	2008-09-28 23:15:00+00:00	2008	

```
4              5     RazakSat 2009-07-13 03:35:00+00:00         2009

  rocket_name rocket_type booster_version              launch_site_name  \
0    Falcon 1    Merlin A         F9 v1.1  Kwajalein Atoll Omelek Island
1    Falcon 1    Merlin A        Merlin A  Kwajalein Atoll Omelek Island
2    Falcon 1    Merlin C        Merlin C  Kwajalein Atoll Omelek Island
3    Falcon 1    Merlin C        Merlin C  Kwajalein Atoll Omelek Island
4    Falcon 1    Merlin C        Merlin C  Kwajalein Atoll Omelek Island

   launch_success orbit  payload_mass_kg landing_success landing_type  \
0           False   LEO             20.0            None         None
1           False   LEO              NaN            None         None
2           False   LEO              NaN            None         None
3            True   LEO            165.0            None         None
4            True   LEO            200.0            None         None

  landing_vehicle core_serial
0            None    Merlin1A
1            None    Merlin2A
2            None    Merlin1C
3            None    Merlin2C
4            None    Merlin3C
```

Base de datos 'spacex.db' creada con 111 registros.

Sitios de lanzamiento únicos

```
C:\Users\Usuario\AppData\Local\Temp\ipykernel_28648\2663568160.py:14:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_clean['launch_date_utc'] = df_clean['launch_date_utc'].astype(str)
```

	launch_site_name
0	Kwajalein Atoll Omelek Island
1	Cape Canaveral Air Force Station Space Launch …
2	Vandenberg Air Force Base Space Launch Complex 4E
3	Kennedy Space Center Historic Launch Complex 39A

5 registros donde el sitio contenga 'Cape'

```
   flight_number          mission_name            launch_date_utc  \
0              6  Falcon 9 Test Flight  2010-06-04 18:45:00+00:00
1              7                COTS 1  2010-12-08 15:43:00+00:00
2              8                COTS 2  2012-05-22 07:44:00+00:00
3              9                 CRS-1  2012-10-08 00:35:00+00:00
```

```
4             10                CRS-2  2013-03-01 19:10:00+00:00
```

```
   launch_year rocket_name rocket_type booster_version  \
0         2010    Falcon 9        v1.0            v1.0
1         2010    Falcon 9        v1.0            v1.0
2         2012    Falcon 9        v1.0            v1.0
3         2012    Falcon 9        v1.0            v1.0
4         2013    Falcon 9        v1.0            v1.0

                                  launch_site_name  launch_success orbit  \
0  Cape Canaveral Air Force Station Space Launch …               1   LEO
1  Cape Canaveral Air Force Station Space Launch …               1   LEO
2  Cape Canaveral Air Force Station Space Launch …               1   LEO
3  Cape Canaveral Air Force Station Space Launch …               1   ISS
4  Cape Canaveral Air Force Station Space Launch …               1   ISS

   payload_mass_kg landing_success landing_type landing_vehicle core_serial
0              NaN            None         None            None       B0003
1              NaN            None         None            None       B0004
2            525.0            None         None            None       B0005
3            400.0            None         None            None       B0006
4            677.0            None         None            None       B0007
```

Total payload transportado por lanzadores de NASA (Kennedy o Cape)

```
   total_payload
0      489552.55
```

Promedio payload para booster versión 'F9 v1.1'

```
   avg_payload
0  1919.454545
```

Fecha del primer aterrizaje exitoso en ground pad (RTLS)

```
        first_ground_landing
0  2015-12-22 01:29:00+00:00
```

Booster(s) exitosos en drone ship (ASDS) con payload entre 4000 y 6000

```
  core_serial  payload_mass_kg landing_vehicle
0       B1022           4696.0          OCISLY
1       B1026           4600.0          OCISLY
2       B1021           5300.0          OCISLY
3       B1031           5200.0          OCISLY
4       B1046           5800.0          OCISLY
5       B1046           4000.0            JRTI
6       B1048           5000.0          OCISLY
```

```
7       B1055           6000.0          OCISLY
8       B1059           5000.0          OCISLY
```

Total de misiones exitosas y fallidas

```
   launch_success  count
0             NaN      3
1             0.0      5
2             1.0    103
```

Booster que ha transportado el máximo payload

```
  rocket_name  payload_mass_kg
0    Falcon 9          15600.0
```

Aterrizajes fallidos en drone ship en 2015

```
  booster_version                                launch_site_name  \
0            v1.1  Cape Canaveral Air Force Station Space Launch …
1            v1.1  Cape Canaveral Air Force Station Space Launch …

   landing_success landing_type
0                0         ASDS
1                0         ASDS
```

Ranking de resultados de aterrizaje entre 2010-06-04 y 2017-03-20

```
  landing_type  count
0         ASDS     12
1         None     12
2        Ocean      5
3         RTLS      3
```

[17]:
```python
import folium
from IPython.display import display

def create_interactive_map_inline(df):
    # Centro del mapa: EE.UU.
    m = folium.Map(location=[28.5, -80.5], zoom_start=5)

    # Coordenadas de ejemplo para algunos sitios conocidos de lanzamiento
    sitio_coords = {
        "Kennedy Space Center Historic Launch Complex 39A": [28.6080585, -80.
↪6039558],
        "Cape Canaveral Air Force Station Space Launch Complex 40": [28.
↪5618571, -80.577366],
```

```
        "Vandenberg Air Force Base Space Launch Complex 4E": [34.632093, -120.
↪610829]
    }

    # Agregar marcadores al mapa según el sitio
    for idx, row in df.iterrows():
        site = row['launch_site_name']
        if site in sitio_coords:
            coords = sitio_coords[site]
            tooltip = f"Mission: {row['mission_name']} (Flight␣
↪{row['flight_number']})"
            folium.Marker(location=coords, tooltip=tooltip).add_to(m)

    # Mostrar el mapa directamente en Jupyter
    return m

# Mostrar mapa
create_interactive_map_inline(df_clean)
```

[17]: <folium.folium.Map at 0x29bc204b470>

[24]:
```
import dash
from dash import html, dcc, Input, Output
import plotly.express as px

def run_dashboard(df):
    # Verificar que haya datos válidos
    if df.empty:
        print(" El DataFrame está vacío. No se puede lanzar el dashboard.")
        return

    # Convertir el año a texto para el Dropdown
    df['launch_year_str'] = df['launch_year'].astype(str)

    # Calcular tasa de éxito por sitio y año
    site_year = df.groupby(['launch_site_name',␣
↪'launch_year_str'])['launch_success'].mean().reset_index()

    # Crear app Dash
    app = dash.Dash(__name__)
    app.title = "SpaceX Dashboard"

    # Layout de la app
    app.layout = html.Div([
        html.H1(" Dashboard de Lanzamientos SpaceX", style={'textAlign':␣
↪'center'}),
        html.Label("Selecciona un año:"),
```

```
        dcc.Dropdown(
            id='year-dropdown',
            options=[{'label': y, 'value': y} for y in␣
 ↪sorted(df['launch_year_str'].unique())],
            value=sorted(df['launch_year_str'].unique())[0],
            clearable=False,
            style={'width': '50%'}
        ),
        dcc.Graph(id='success-graph')
    ], style={'padding': '20px'})

    # Callback para actualizar la gráfica
    @app.callback(
        Output('success-graph', 'figure'),
        Input('year-dropdown', 'value')
    )
    def update_graph(selected_year):
        filtered = site_year[site_year['launch_year_str'] == selected_year]
        fig = px.bar(filtered,
                     x='launch_site_name',
                     y='launch_success',
                     title=f"Tasa de Éxito por Sitio en {selected_year}",
                     labels={'launch_site_name': 'Sitio de Lanzamiento',␣
 ↪'launch_success': 'Tasa de Éxito'},
                     color='launch_success',
                     color_continuous_scale='Blues')
        fig.update_layout(yaxis=dict(tickformat='.0%'))
        return fig

    # Ejecutar la app en modo local
    app.run_server(debug=True, use_reloader=False)


# run_dashboard(df_clean)
```

[22]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

def prepare_features(df):
    df_model = df.copy()

    # Eliminar filas con NA en la variable objetivo
    df_model = df_model[df_model['launch_success'].notna()]

    # Codificar año y booster
```

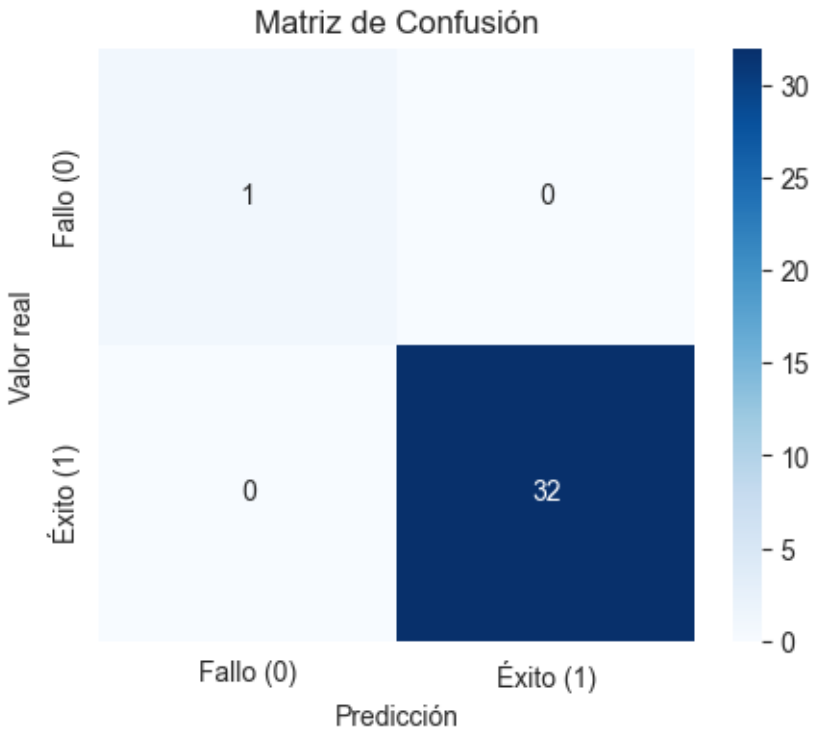
```
    df_model['launch_year_num'] = pd.to_numeric(df_model['launch_year'],␣
 ↪errors='coerce')
    df_model['booster_code'] = df_model['booster_version'].astype('category').
 ↪cat.codes

    # Seleccionar características
    features = df_model[['flight_number', 'launch_year_num', 'booster_code']].
 ↪dropna()
    target = df_model.loc[features.index, 'launch_success'].astype(int)

    print(f" Total de muestras válidas: {len(features)}")
    return features, target

def predictive_analysis(df):
    X, y = prepare_features(df)

    if len(X) == 0:
        print(" No hay datos suficientes para entrenar el modelo.")
        return

    # División de datos
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

    # Entrenar modelo
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)

    # Métricas
    acc = accuracy_score(y_test, predictions)
    conf_mat = confusion_matrix(y_test, predictions)
    class_report = classification_report(y_test, predictions)

    print(" Exactitud del modelo:", round(acc * 100, 2), "%")
    print("\n Matriz de Confusión:\n", conf_mat)
    print("\n Reporte de Clasificación:\n", class_report)

# Ejecutar análisis predictivo
predictive_analysis(df_clean)
```

```
 Total de muestras válidas: 108
 Exactitud del modelo: 100.0 %

 Matriz de Confusión:
[[ 1  0]
 [ 0 32]]
```

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00        32

    accuracy                           1.00        33
   macro avg       1.00      1.00      1.00        33
weighted avg       1.00      1.00      1.00        33
```

[23]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

def plot_confusion_matrix(cm, labels):
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicción")
    plt.ylabel("Valor real")
    plt.title("Matriz de Confusión")
    plt.show()

# Crear la matriz nuevamente
from sklearn.metrics import confusion_matrix
X, y = prepare_features(df_clean)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
cm = confusion_matrix(y_test, predictions)

# Mostrar la matriz de confusión graficada
plot_confusion_matrix(cm, labels=["Fallo (0)", "Éxito (1)"])
```

```
Total de muestras válidas: 108
```


Matriz de Confusión
Valor real
Fallo (0)
Éxito (1)
1
0
0
32
Fallo (0)
Éxito (1)
Predicción
30
25
20
15
10
5
0

[ ]: