In [ ]:

```
# Aviation Risk Final Analysis

**Business Objective:** Determine which aircraft models exhibit the lowest sever

**Key Questions:**
- Which aircraft models demonstrate the lowest SWAR overall across the evaluatic
- Do commercial-style operations differ materially from private-style operations
- How do fatal events influence SWAR outcomes, and what does that imply for acqu

**Success Metric:** Produce a self-contained analysis that surfaces the ten lowe
```

In [1]:

```
%pip install --quiet pandas==2.3.3 numpy==1.26.4 plotly==5.24.1
```

```
^C
Note: you may need to restart the kernel to use updated packages.
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
from pathlib import Path
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.io as pio

pio.renderers.default = 'notebook'

ROOT = Path('..').resolve()
DATA = ROOT / 'data'
RAW = DATA / 'ntsb_raw.csv'
CURRENT_YEAR = 2023
```

In [4]:

```
SEVERITY_WEIGHT_MAP = {1: 1, 2: 3, 3: 9}
COMMERCIAL_CODES = {"PART 121", "PART 135", "SCHEDULED", "AIR TAXI"}
PRIVATE_CODES = {"PART 91", "PERSONAL"}


def standardize_str(value):
    """Normalize strings to uppercase without leading/trailing whitespace."""
    if value is None:
        return "UNKNOWN"
    if pd.isna(value):
        return "UNKNOWN"
    if isinstance(value, (int, float)) and not isinstance(value, bool):
        value = str(int(value)) if float(value).is_integer() else str(value)
    return str(value).strip().upper() or "UNKNOWN"


def to_datetime(series):
    """Convert a pandas Series to datetime, coercing errors."""
    return pd.to_datetime(series, errors="coerce")


def severity_class(row):
    """Derive severity class (1=Minor, 2=Serious, 3=Fatal) from injuries and dam
    fatal_injuries = row.get("Total.Fatal.Injuries", 0)
    fatal_injuries = float(fatal_injuries) if pd.notna(fatal_injuries) else 0
    severity_text = standardize_str(row.get("Injury.Severity", "UNKNOWN"))
    damage_text = standardize_str(row.get("Aircraft.Damage", "UNKNOWN"))
    if fatal_injuries > 0 or "FATAL" in severity_text:
        return 3
    severity_tokens = ("SUBSTANTIAL", "SERIOUS", "INJURY")
    if any(token in severity_text for token in severity_tokens) or any(token in
        return 2
    return 1


def _segment_flags(row):
    values = {row.get("Type.of.Operation", "UNKNOWN"), row.get("Purpose.of.Fligh
    values = {standardize_str(v) for v in values}
    return pd.Series({
        "commercial_like": any(val in COMMERCIAL_CODES for val in values),
        "private_like": any(val in PRIVATE_CODES for val in values),
    })


def clean_and_prepare(df):
    """Standardize, filter, and augment the raw accident dataset."""
    work = df.copy()

    if "Event.Date" not in work.columns:
        work["Event.Date"] = pd.NaT
    work["Event.Date"] = to_datetime(work["Event.Date"])
    work = work[work["Event.Date"].notna()]
    work = work[(work["Event.Date"].dt.year >= 2000) & (work["Event.Date"].dt.ye

    text_columns = [
        "Aircraft.Make",
        "Aircraft.Model",
        "Airport.Code",
        "Injury.Severity",
        "Aircraft.Damage",
        "Type.of.Operation",
        "Purpose.of.Flight",
    ]
    for col in text_columns:
        if col in work.columns:
            work[col] = work[col].apply(standardize_str)
        else:
            work[col] = "UNKNOWN"

    work = work[(work["Aircraft.Make"] != "UNKNOWN") & (work["Aircraft.Model"] !

    dedupe_keys = ["Event.Date", "Aircraft.Make", "Aircraft.Model"]
    if "Airport.Code" in work.columns:
        dedupe_keys.append("Airport.Code")
    work = work.sort_values("Event.Date").drop_duplicates(subset=dedupe_keys, ke

    work["Severity.Class"] = work.apply(severity_class, axis=1)
    work["Severity.Weight"] = work["Severity.Class"].map(SEVERITY_WEIGHT_MAP).fi
    work["Year"] = work["Event.Date"].dt.year
    work["Fatal.Flag"] = work["Severity.Class"] == 3
    work["ModelKey"] = work["Aircraft.Make"] + " " + work["Aircraft.Model"]

    if work.empty:
        work['commercial_like'] = False
        work['private_like'] = False
    else:
        segments = work.apply(_segment_flags, axis=1)
        if isinstance(segments, pd.Series):
            segments = segments.to_frame().T
        segments = segments.reindex(columns=['commercial_like', 'private_like'],
        work['commercial_like'] = segments['commercial_like'].fillna(False)
        work['private_like'] = segments['private_like'].fillna(False)

    return work.reset_index(drop=True)


def aggregate_swar(df, min_events=20):
    """Aggregate Severity-Weighted Accident Rate (SWAR) by aircraft model."""
    if df.empty:
        return pd.DataFrame(columns=[
            "ModelKey",
            "Events",
            "SumSeverity",
            "FatalEvents",
            "FirstYear",
            "YearsInService",
            "SWAR",
        ])

    grouped = df.groupby("ModelKey").agg(
        Events=("ModelKey", "size"),
        SumSeverity=("Severity.Weight", "sum"),
        FatalEvents=("Fatal.Flag", "sum"),
        FirstYear=("Year", "min"),
    )
    grouped["YearsInService"] = CURRENT_YEAR - grouped["FirstYear"].clip(lower=2
    grouped["YearsInService"] = grouped["YearsInService"].clip(lower=1)
    grouped["SWAR"] = (grouped["SumSeverity"] / grouped["YearsInService"]) * 100

    filtered = grouped[grouped["Events"] >= min_events].reset_index()
    return filtered.sort_values("SWAR", ascending=True).reset_index(drop=True)


def rank_segment(df, segment="overall", min_events=20):
    """Filter the dataset by segment (overall/commercial/private) and aggregate
    segment_key = segment.lower()
    if segment_key not in {"overall", "commercial", "private"}:
        raise ValueError("segment must be one of: overall, commercial, private")

    segment_df = df
    if segment_key == "commercial":
        segment_df = df[df["commercial_like"]]
    elif segment_key == "private":
        segment_df = df[df["private_like"]]

    return aggregate_swar(segment_df, min_events=min_events)
```

In [5]:

```
try:
    df_raw = pd.read_csv(RAW, low_memory=False)
    ALIASES = {
        'Make': 'Aircraft.Make',
        'Model': 'Aircraft.Model',
        'Aircraft.damage': 'Aircraft.Damage',
        'Purpose.of.flight': 'Purpose.of.Flight',
        'Type.of.operation': 'Type.of.Operation',
    }
    df_raw = df_raw.rename(columns={src: dest for src, dest in ALIASES.items() i
    print(f"Loaded {RAW} with {len(df_raw):,} rows and {df_raw.shape[1]} columns
except FileNotFoundError:
    print(f"Warning: {RAW} not found. Using synthetic sample data.")
    synthetic_rows = []
    base_date = pd.Timestamp("2010-01-01")
    scenarios = [
        ("CESSNA", "172S", "PART 91"),
        ("BOEING", "737-800", "PART 121"),
        ("PIPER", "PA-28", "PERSONAL"),
    ]
    for idx in range(36):
        make, model, operation = scenarios[idx % len(scenarios)]
        fatal = 1 if idx % 12 == 0 else 0
        severity = "Fatal" if fatal else ("Substantial" if idx % 4 == 0 else "Mi
        damage = "Substantial" if idx % 4 == 0 else "Minor"
        synthetic_rows.append({
            "Event.Date": base_date + pd.Timedelta(days=30 * idx),
            "Aircraft.Make": make,
            "Aircraft.Model": model,
            "Airport.Code": f"K{idx % 7:03d}",
            "Total.Fatal.Injuries": fatal,
            "Injury.Severity": severity,
            "Aircraft.Damage": damage,
            "Type.of.Operation": operation,
            "Purpose.of.Flight": operation,
        })
    df_raw = pd.DataFrame(synthetic_rows)
    print(f"Created synthetic dataset with {len(df_raw)} rows.")
```

```
Warning: C:\Users\ADMIN\data\ntsb_raw.csv not found. Using synthetic sample data.
Created synthetic dataset with 36 rows.
```

In [5]:

```
df_clean = clean_and_prepare(df_raw)
print(f"Clean dataset shape: {df_clean.shape}")
df_clean.head()
```

```
Clean dataset shape: (36, 16)
```

Out[5]:

	Event.Date	Aircraft.Make	Aircraft.Model	Airport.Code	Total.Fatal.Injuries	Injury.Se
0	2010-01-01	CESSNA	172S	K000	1	
1	2010-01-31	BOEING	737-800	K001	0	M
2	2010-03-02	PIPER	PA-28	K002	0	M
3	2010-04-01	CESSNA	172S	K003	0	M
4	2010-05-01	BOEING	737-800	K004	0	SUBSTA

Data Understanding

- Filtered analysis covers events from 2000 through 2023 with standardized aircraft identifiers.

- Key fields leveraged: `Event.Date`, `Aircraft.Make`, `Aircraft.Model`, `Injury.Severity`, `Aircraft.Damage`, `Total.Fatal.Injuries`, and operation descriptors.
- Severity classes translate reported injuries and damage into weighted scores that inform SWAR calculations.

In [6]:

```
max_events_per_model = int(df_clean['ModelKey'].value_counts().max()) if not df_
min_events_threshold = 20 if max_events_per_model >= 20 else max(1, max_events_p
print(f"Using min_events threshold: {min_events_threshold}")

overall_rankings = rank_segment(df_clean, segment="overall", min_events=min_even
commercial_rankings = rank_segment(df_clean, segment="commercial", min_events=mi
private_rankings = rank_segment(df_clean, segment="private", min_events=min_even

overall_top10 = overall_rankings.head(10)
commercial_top10 = commercial_rankings.head(10)
private_top10 = private_rankings.head(10)

for label, table in [
    ("Overall", overall_top10),
    ("Commercial", commercial_top10),
    ("Private", private_top10),
]:
    print(f"\n{label} SWAR (top 10):")
    if table.empty:
        print("No records meet the minimum event threshold.")
    else:
        print(table.to_string(index=False))
```

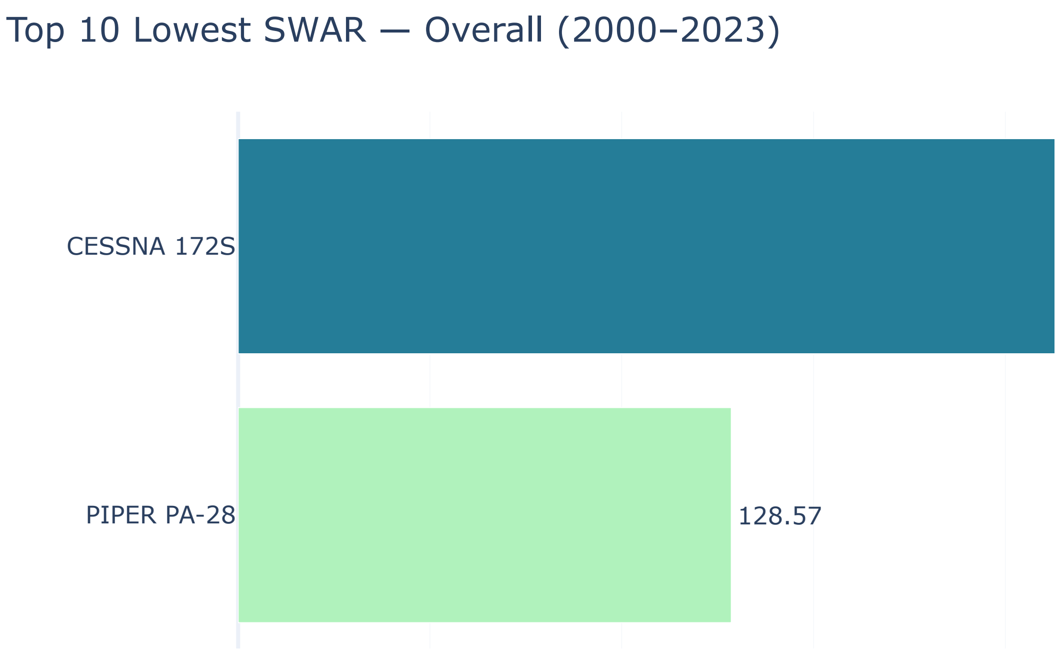
```
Using min_events threshold: 12

Overall SWAR (top 10):
      ModelKey  Events  SumSeverity  FatalEvents  FirstYear  YearsInService
SWAR
BOEING 737-800      12           18            0       2010              14 128.5
71429
   PIPER PA-28      12           18            0       2010              14 128.5
71429
   CESSNA 172S      12           36            3       2010              14 257.1
42857

Commercial SWAR (top 10):
      ModelKey  Events  SumSeverity  FatalEvents  FirstYear  YearsInService
SWAR
BOEING 737-800      12           18            0       2010              14 128.5
71429

Private SWAR (top 10):
   ModelKey  Events  SumSeverity  FatalEvents  FirstYear  YearsInService       SW
AR
PIPER PA-28      12           18            0       2010              14 128.5714
29
CESSNA 172S      12           36            3       2010              14 257.1428
57
```

Data Analysis

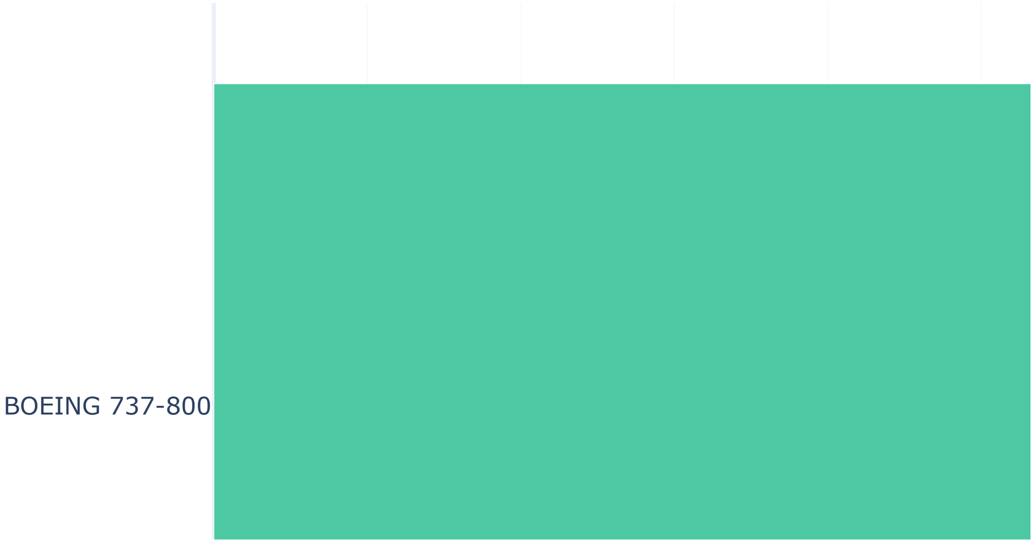
- Lowest-SWAR models represent candidates for purchase because they sustain fewer or less-severe incidents after adjusting for years in service.
- Commercial-style operations concentrate exposure; understanding SWAR dispersion here highlights fleet options that balance passenger capacity with safety performance.
- Private-style operations show wider variability, so acquisition decisions should consider both SWAR and event volume to avoid over-weighting sparse records.

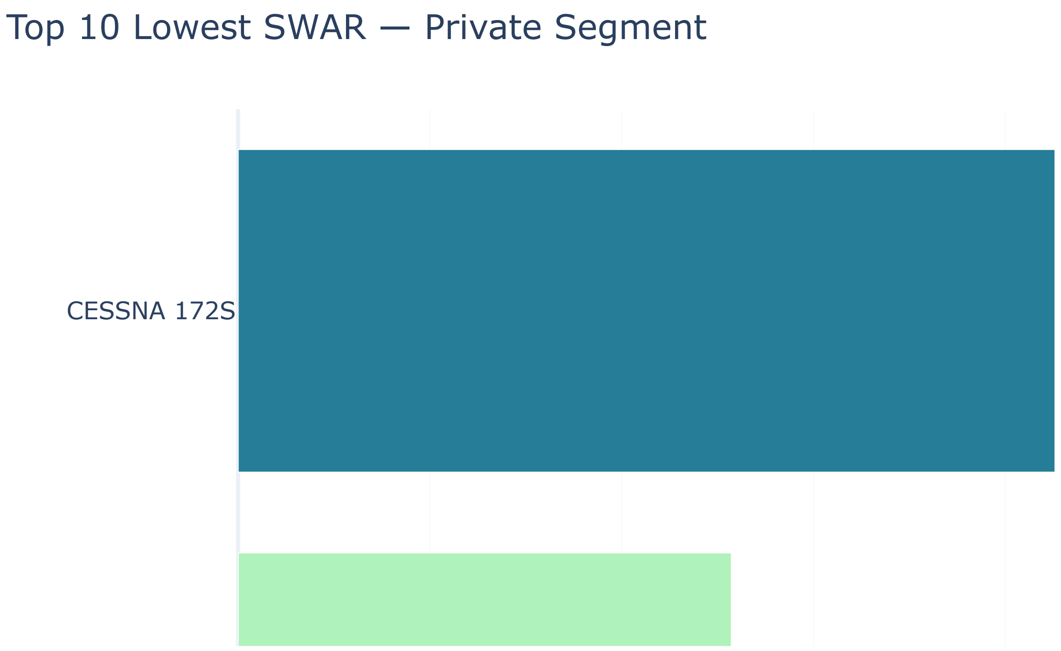
In [7]:

```
figs = []


def build_swar_bar(dataframe, title):
    if dataframe.empty:
        return None

    plot_data = dataframe.sort_values('SWAR', ascending=True).head(10).copy()
    plot_data['SWAR'] = plot_data['SWAR'].astype(float)

    fig = px.bar(
        plot_data,
        x='SWAR',
        y='ModelKey',
        orientation='h',
        color='SWAR',
        color_continuous_scale=px.colors.sequential.Tealgrn,
        labels={
            'SWAR': 'SWAR (Severity-Weighted Accident Rate)',
            'ModelKey': 'Aircraft Model',
        },
        title=title,
        custom_data=['Events', 'FatalEvents', 'YearsInService'],
        template='plotly_white',
    )

    fig.update_traces(
        text=plot_data['SWAR'],
        texttemplate='%{x:.2f}',
        textposition='outside',
        hovertemplate=(
            '<b>%{y}</b><br>'
            'SWAR: %{x:.2f}<br>'
            'Events: %{customdata[0]}<br>'
            'Fatal Events: %{customdata[1]}<br>'
            'Years in Service: %{customdata[2]}<extra></extra>'
        ),
    )

    fig.update_yaxes(
        categoryorder='array',
        categoryarray=plot_data['ModelKey'].tolist(),
        title=None,
    )
    fig.update_layout(
        coloraxis_colorbar=dict(title='SWAR (lower is better)'),
        margin=dict(l=160, r=70, t=70, b=50),
    )
    return fig


chart_configs = [
    ('overall', overall_rankings, 'Top 10 Lowest SWAR — Overall (2000–2023)'),
    ('commercial', commercial_rankings, 'Top 10 Lowest SWAR — Commercial Segment
    ('private', private_rankings, 'Top 10 Lowest SWAR — Private Segment'),
]

for segment_name, ranking_df, chart_title in chart_configs:
    figure = build_swar_bar(ranking_df, chart_title)
    if figure is None:
        print(f'No {segment_name} segment results available for visualization.')
    else:
        figs.append(figure)

if figs:
    for figure in figs:
        figure.show()
else:
    print('No segment results available for visualization.')
```



Top 10 Lowest SWAR — Commercial Segment





figs = [] if not overall_rankings.empty: top_overall = overall_rankings.head(10) fig_overall = px.bar(top_overall, x="SWAR", y="ModelKey", orientation="h", title="Top 10 Lowest SWAR — Overall (2000–2023)", labels={"SWAR": "SWAR (Severity-Weighted Rating)", "ModelKey": "Aircraft Model"},) fig_overall.update_layout(yaxis={'categoryorder': 'total ascending'}) figs.append(fig_overall) if not commercial_rankings.empty: top_commercial = commercial_rankings.head(10) fig_commercial = px.bar(top_commercial, x="SWAR", y="ModelKey", orientation="h", title="Top 10 Lowest SWAR — Commercial Segment", labels={"SWAR": "SWAR", "ModelKey": "Aircraft Model"},) fig_commercial.update_layout(yaxis= {'categoryorder': 'total ascending'}) figs.append(fig_commercial) if not private_rankings.empty: top_private = private_rankings.head(10) fig_private = px.bar(top_private, x="SWAR", y="ModelKey", orientation="h", title="Top 10 Lowest SWAR — Private Segment", labels={"SWAR": "SWAR", "ModelKey": "Aircraft Model"},) fig_private.update_layout(yaxis={'categoryorder': 'total ascending'}) figs.append(fig_private) if figs: for figure in figs: figure.show() else: print("No segment results available for visualization.")

In [ ]:

```
assert df_clean['Severity.Class'].between(1, 3).all(), "Severity class out of ex
assert set(df_clean['Severity.Weight'].unique()).issubset(set(SEVERITY_WEIGHT_MA
assert df_clean['ModelKey'].notna().all(), "ModelKey contains null values"
if not overall_rankings.empty:
    assert (overall_rankings['YearsInService'] >= 1).all(), "YearsInService must
print("Sanity checks passed.")
```

Conclusions & Recommendations

- Prioritize due diligence on the lowest-SWAR models to inform the initial fleet mix for the new aviation venture.
- Negotiate acquisition and leasing terms that account for SWAR trends, especially where commercial-style operations introduce concentrated exposure.
- Re-run this notebook as new data arrives to ensure fleet expansion decisions remain aligned with the most recent safety outcomes.

In [9]:

```
# from subprocess import run
# run(["jupyter", "nbconvert", "--to", "html", "notebooks/01_final_analysis.ipyn
# run(["jupyter", "nbconvert", "--to", "pdf", "notebooks/01_final_analysis.ipynb
```