In [1]:
```
    # ---------- 1. IMPORTS & SETTINGS ----------
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, classification_report, confusion_matrix
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier, GradientBoostingRegressor, GradientBoosti

plt.rcParams.update({'figure.dpi': 100})
pd.options.display.max_columns = 50

# Why: loads essential libraries for cleaning, plotting, and modeling.
# Matplotlib is used for plotting. sklearn used for basic models and metrics.
```

In [2]:
```
# ---------- 2. PATH AND BASIC INFO ----------
file_path = r"C:\Users\ancha\Desktop\European Flights.csv"
print("File exists:", os.path.exists(file_path))
print("File path:", file_path)
# Why: confirm the file is where we expect it.
```

```
File exists: True
File path: C:\Users\ancha\Desktop\European Flights.csv
```

In [3]:
```
# ---------- 3. MEMORY-SAFE READ & CLEAN (CHUNKING) ----------
# Goal: create a cleaned CSV with added date features while never loading the whole huge file at once.
usecols_guess = ['FLT_DATE','APT_ICAO','APT_NAME','STATE_NAME','FLT_DEP_1','FLT_ARR_1','FLT_TOT_1',
                 'FLT_DEP_IFR_2','FLT_ARR_IFR_2','FLT_TOT_IFR_2']

# Find which of these columns are actually present to avoid errors:
existing_cols = pd.read_csv(file_path, nrows=1).columns.str.strip().tolist()
usecols = [c for c in usecols_guess if c in existing_cols]
print("Columns to use:", usecols)
#We're checking which columns exist in the CSV,and using only the ones that are actually there,so the program doesn't
```

```
Columns to use: ['FLT_DATE', 'APT_ICAO', 'APT_NAME', 'STATE_NAME', 'FLT_DEP_1', 'FLT_ARR_1', 'FLT_TOT_1', 'FLT_DEP_IF
R_2', 'FLT_ARR_IFR_2', 'FLT_TOT_IFR_2']
```

In [4]:
```
# We'll process in chunks to avoid memory errors
chunksize = 100000         # reduce if your laptop has less RAM
cleaned_csv = os.path.join(os.path.dirname(file_path), "European_Flights_cleaned.csv")
```

In [5]:
```
# Remove old cleaned file if exists
if os.path.exists(cleaned_csv):
    os.remove(cleaned_csv)
```

In [6]:
```
# Aggregators for EDA (so we don't need full file in memory)
country_totals = defaultdict(float)
airport_totals = defaultdict(float)
monthly_totals = defaultdict(float)
month_week_counts = defaultdict(float)
sample_parts = []
available_ifr_cols = set()
rows_processed = 0

print("Starting chunked read & cleaning... (this may take a minute)")

reader = pd.read_csv(file_path, usecols=usecols, parse_dates=['FLT_DATE'],
                     iterator=True, chunksize=chunksize, low_memory=True)
#We're setting up containers to store summaries and samples,and creating a reader that loads the giant CSV file in sm
```

```
Starting chunked read & cleaning... (this may take a minute)
```

In [7]:
```
first_write = True
for chunk in reader:
    # 3.1 Normalize column names
    chunk.columns = chunk.columns.str.strip()
    # 3.2 Coerce numeric columns we will use
    for c in ['FLT_DEP_1','FLT_ARR_1','FLT_TOT_1','FLT_DEP_IFR_2','FLT_ARR_IFR_2','FLT_TOT_IFR_2']:
        if c in chunk.columns:
            chunk[c] = pd.to_numeric(chunk[c], errors='coerce')

# 3.3 Drop rows without date; for plotting we require FLT_TOT_1 in many charts
    if 'FLT_TOT_1' in chunk.columns:
        chunk = chunk.dropna(subset=['FLT_DATE','FLT_TOT_1']).copy()
    else:
        chunk = chunk.dropna(subset=['FLT_DATE']).copy()
    # 3.4 Add date features for cleaned CSV
    chunk['month'] = chunk['FLT_DATE'].dt.month
```

```
    chunk['year'] = chunk['FLT_DATE'].dt.year
    chunk['weekday'] = chunk['FLT_DATE'].dt.day_name()
    # 3.5 Save chunk into cleaned CSV (append)
    chunk.to_csv(cleaned_csv, mode='a', index=False, header=first_write)
    first_write = False
    # 3.6 Aggregate for EDA
    if 'STATE_NAME' in chunk.columns and 'FLT_TOT_1' in chunk.columns:
        for k,v in chunk.groupby('STATE_NAME')['FLT_TOT_1'].sum().items():
            country_totals[k] += v
    if 'APT_NAME' in chunk.columns and 'FLT_TOT_1' in chunk.columns:
        for k,v in chunk.groupby('APT_NAME')['FLT_TOT_1'].sum().items():
            airport_totals[k] += v
    if 'FLT_TOT_1' in chunk.columns:
        # monthly totals (YYYY-MM)
        chunk['year_month'] = chunk['FLT_DATE'].dt.to_period('M').astype(str)
        for k,v in chunk.groupby('year_month')['FLT_TOT_1'].sum().items():
            monthly_totals[k] += v
        # month-week heat
        for (m,wd), s in chunk.groupby([chunk['month'], chunk['weekday']])['FLT_TOT_1'].sum().items():
            month_week_counts[(m,wd)] += s
        # sample for scatter/hist (keep small fraction)
        sample_parts.append(chunk[['FLT_TOT_1','FLT_DEP_1','FLT_ARR_1','APT_NAME']].sample(frac=0.05, random_state=42
    # record IFR availability
    for c in ['FLT_DEP_IFR_2','FLT_ARR_IFR_2','FLT_TOT_IFR_2']:
        if c in chunk.columns:
            available_ifr_cols.add(c)
    rows_processed += len(chunk)

print("Chunking done. Rows processed (approx):", rows_processed)
print("IFR columns available:", available_ifr_cols)
print("Cleaned CSV saved to:", cleaned_csv)

#We read the giant CSV in small chunks, clean each chunk, save it, compute useful totals, take small samples for char
#and keep track of which columns exist—without ever loading the whole dataset into memory.
```

```
C:\Users\ancha\AppData\Local\Temp\ipykernel_17728\996039007.py:31: UserWarning: Converting to PeriodArray/Index repre
sentation will drop timezone information.
  chunk['year_month'] = chunk['FLT_DATE'].dt.to_period('M').astype(str)
C:\Users\ancha\AppData\Local\Temp\ipykernel_17728\996039007.py:31: UserWarning: Converting to PeriodArray/Index repre
sentation will drop timezone information.
  chunk['year_month'] = chunk['FLT_DATE'].dt.to_period('M').astype(str)
C:\Users\ancha\AppData\Local\Temp\ipykernel_17728\996039007.py:31: UserWarning: Converting to PeriodArray/Index repre
sentation will drop timezone information.
  chunk['year_month'] = chunk['FLT_DATE'].dt.to_period('M').astype(str)
C:\Users\ancha\AppData\Local\Temp\ipykernel_17728\996039007.py:31: UserWarning: Converting to PeriodArray/Index repre
sentation will drop timezone information.
  chunk['year_month'] = chunk['FLT_DATE'].dt.to_period('M').astype(str)
C:\Users\ancha\AppData\Local\Temp\ipykernel_17728\996039007.py:31: UserWarning: Converting to PeriodArray/Index repre
sentation will drop timezone information.
  chunk['year_month'] = chunk['FLT_DATE'].dt.to_period('M').astype(str)
C:\Users\ancha\AppData\Local\Temp\ipykernel_17728\996039007.py:31: UserWarning: Converting to PeriodArray/Index repre
sentation will drop timezone information.
  chunk['year_month'] = chunk['FLT_DATE'].dt.to_period('M').astype(str)
Chunking done. Rows processed (approx): 688099
IFR columns available: {'FLT_ARR_IFR_2', 'FLT_TOT_IFR_2', 'FLT_DEP_IFR_2'}
Cleaned CSV saved to: C:\Users\ancha\Desktop\European_Flights_cleaned.csv
```

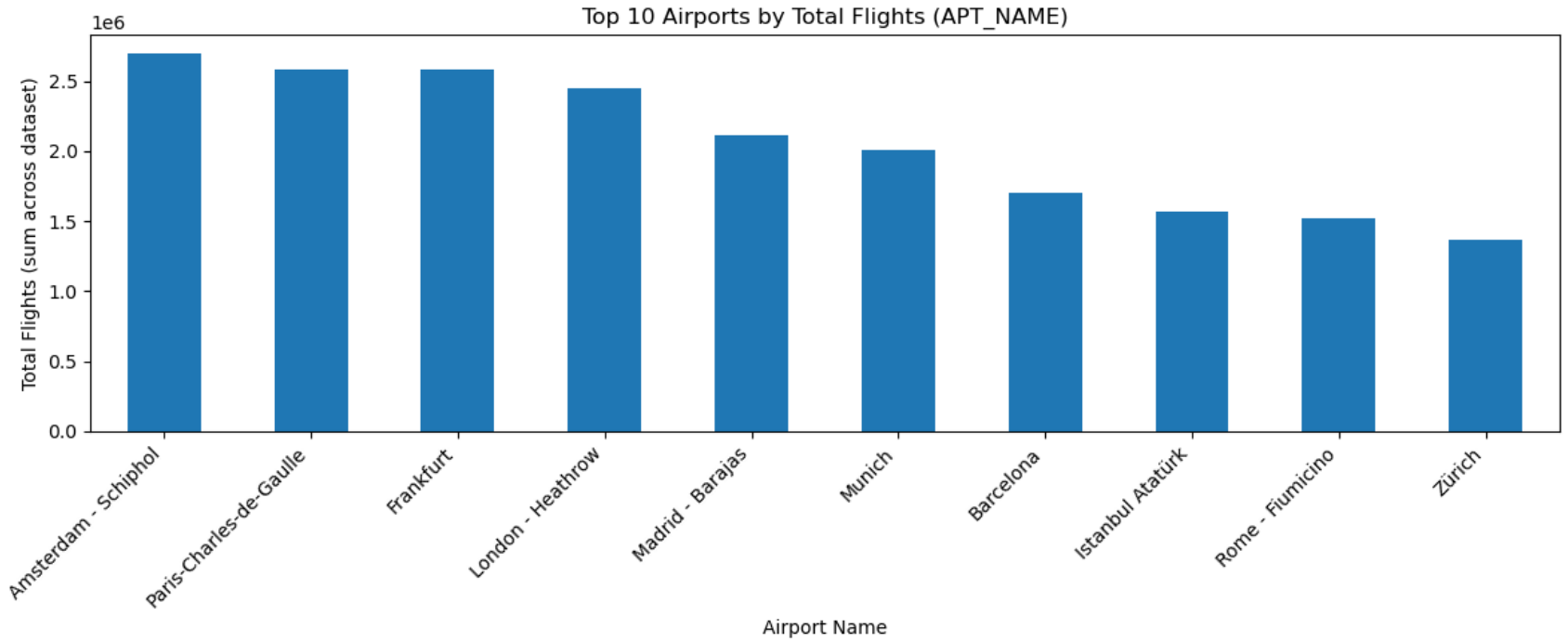
```
C:\Users\ancha\AppData\Local\Temp\ipykernel_17728\996039007.py:31: UserWarning: Converting to PeriodArray/Index repre
sentation will drop timezone information.
  chunk['year_month'] = chunk['FLT_DATE'].dt.to_period('M').astype(str)
```

In [8]:

```
# Combine sample
sample_df = pd.concat(sample_parts, ignore_index=True) if sample_parts else pd.DataFrame(columns=['FLT_TOT_1','FLT_DE

# Convert aggregators to pandas structures
country_s = pd.Series(country_totals).sort_values(ascending=False)
airport_s = pd.Series(airport_totals).sort_values(ascending=False)
monthly_index = sorted(monthly_totals.keys())
monthly_series = pd.Series([monthly_totals[k] for k in monthly_index], index=pd.to_datetime(monthly_index))

# Build month-week heat matrix
weekdays = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
heat_matrix = np.zeros((12,7))
for (m,wd), val in month_week_counts.items():
    try:
        heat_matrix[m-1, weekdays.index(wd)] = val
    except:
```

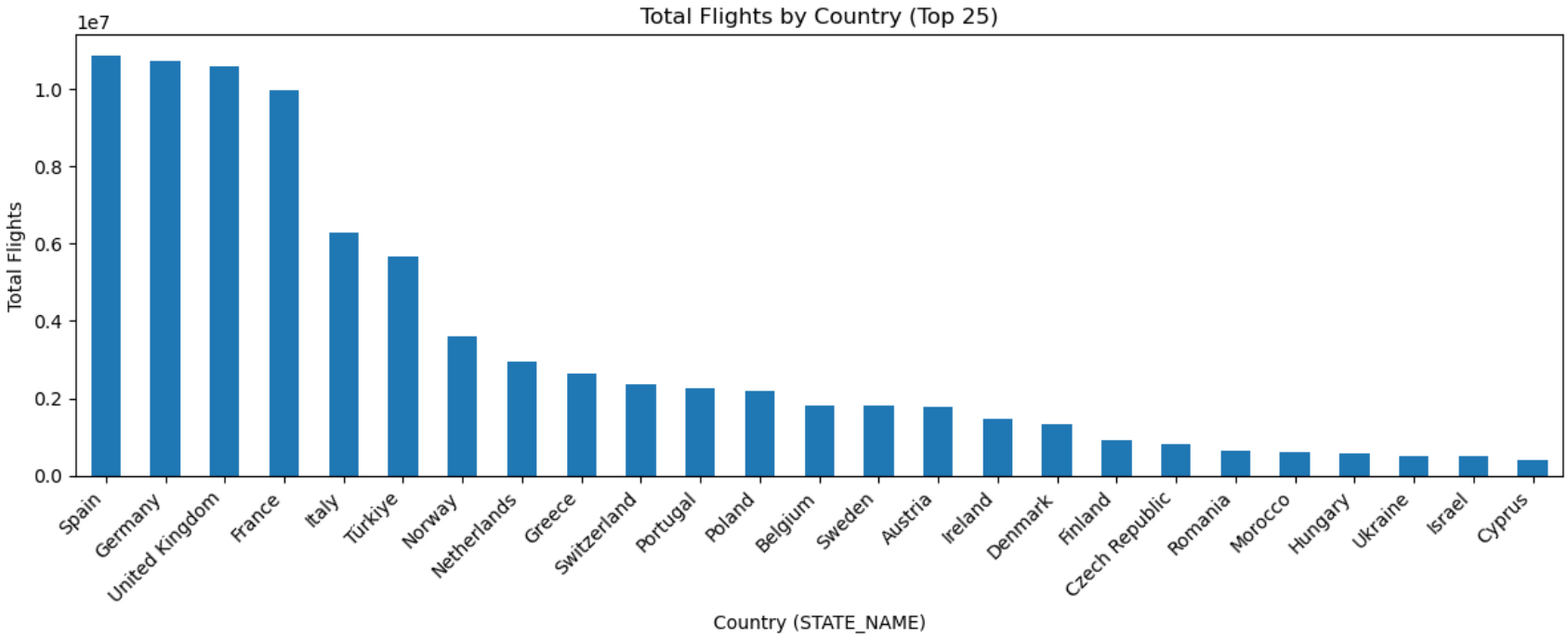
```
        pass
#We combine the sample data, convert our running totals into pandas objects, organize monthly totals,
#and build a heatmap matrix so we can easily make charts and visualizations later.

#We read a huge flight dataset in small pieces, cleaned it, saved a clean version, created useful new columns, remove
#collected samples for plotting, and calculated summaries (by country, airport, month, weekday).
#Everything was done chunk-by-chunk to avoid memory issues.
```

In [9]:
```
# ---------- 4. SIMPLE, HIGH-MEANING VISUALIZATIONS (EDA) ----------
# Each chart followed by a brief printed explanation.

# 4.1 Top 10 Airports by Total Flights (APT_NAME)
plt.figure(figsize=(12,5))
airport_s.head(10).plot(kind='bar')
plt.title("Top 10 Airports by Total Flights (APT_NAME)")
plt.xlabel("Airport Name")
plt.ylabel("Total Flights (sum across dataset)")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
print("Explanation: This shows busiest airports by total recorded flights. Use this to identify hubs and outliers.")
```



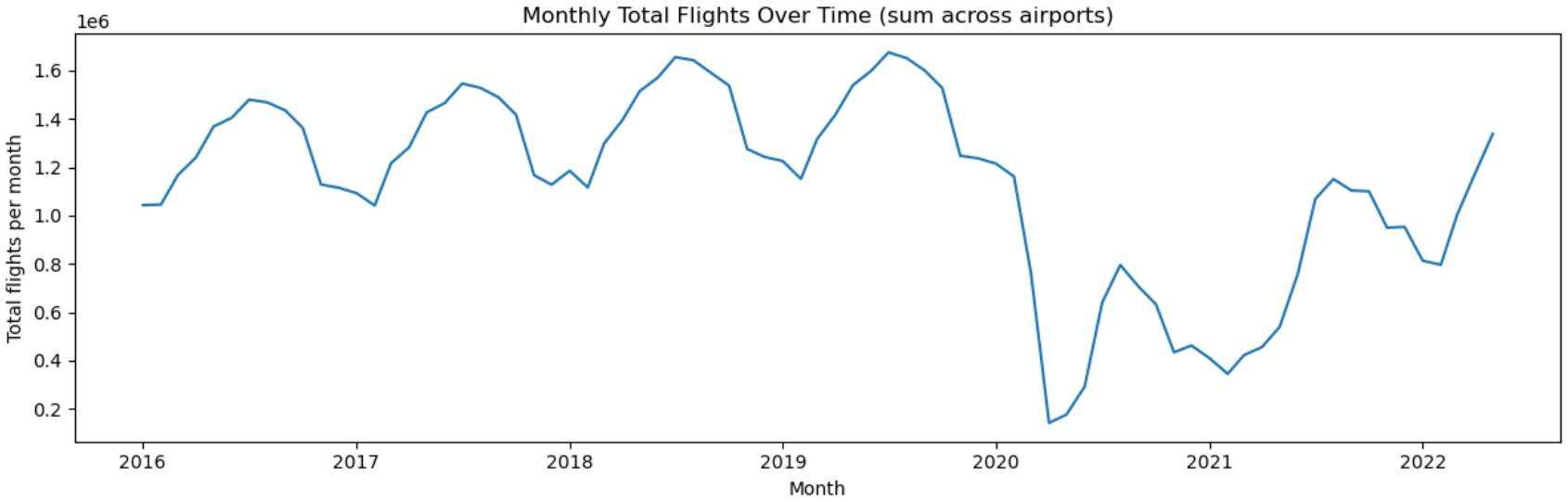
Explanation: This shows busiest airports by total recorded flights. Use this to identify hubs and outliers.

In [10]:
```
# 4.2 Total Flights by Country (Top 25)
plt.figure(figsize=(12,5))
country_s.head(25).plot(kind='bar')
plt.title("Total Flights by Country (Top 25)")
plt.xlabel("Country (STATE_NAME)")
plt.ylabel("Total Flights")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
print("Explanation: Shows which countries contribute most to flight volume. Useful for country-level discussion.")
```



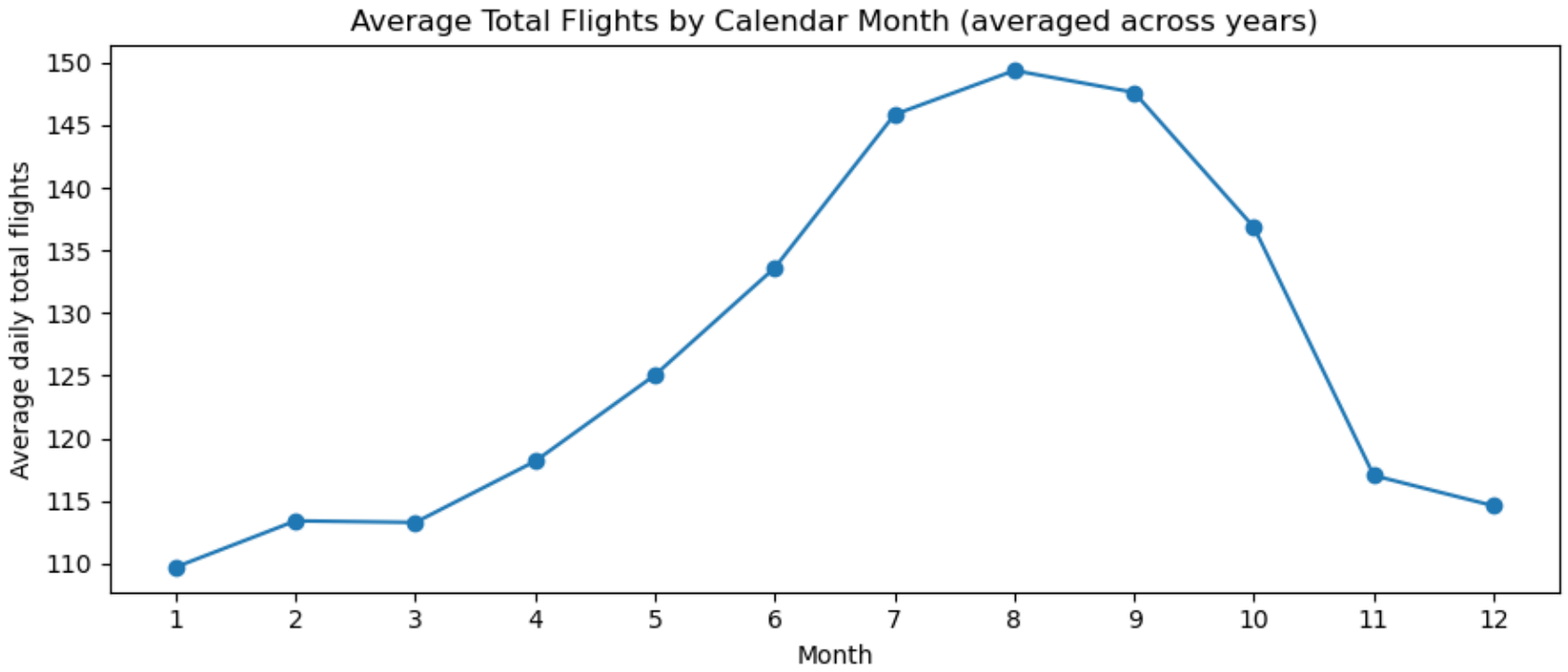
Explanation: Shows which countries contribute most to flight volume. Useful for country-level discussion.

In [11]:
```
# 4.3 Monthly total flights over time (trend & seasonality)
plt.figure(figsize=(12,4))
plt.plot(monthly_series.index, monthly_series.values)
plt.title("Monthly Total Flights Over Time (sum across airports)")
plt.xlabel("Month")
plt.ylabel("Total flights per month")
plt.tight_layout()
plt.show()
print("Explanation: Shows long-term trends and seasonal peaks/dips (e.g., summer peaks, sudden dips).")
#The chart shows yearly flight seasonality, strong pre-2020 growth, a dramatic COVID-19 collapse, and a slow recovery
#Flights rise during summer months (June–August)
#Flights drop during winter months (December–February)
```



Explanation: Shows long-term trends and seasonal peaks/dips (e.g., summer peaks, sudden dips).

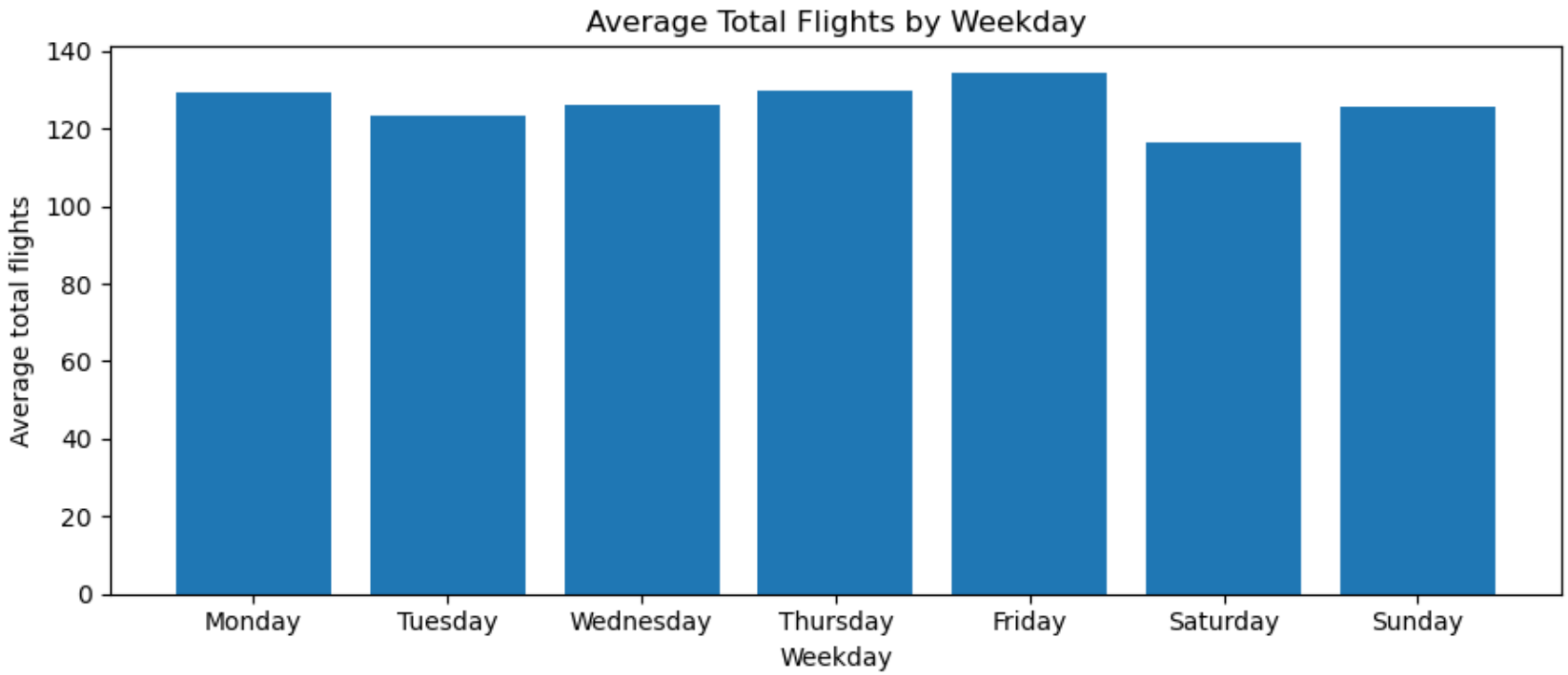
In [12]:
```
# 4.4 Average by calendar month (seasonality)
# compute average per calendar month safely using cleaned CSV
month_sum = defaultdict(float); month_count = defaultdict(int)
for chunk in pd.read_csv(cleaned_csv, parse_dates=['FLT_DATE'], iterator=True, chunksize=chunksize, low_memory=True):
    mgrp = chunk.groupby(chunk['FLT_DATE'].dt.month)['FLT_TOT_1'].agg(['sum','count'])
    for m,row in mgrp.iterrows():
        month_sum[m] += row['sum']; month_count[m] += row['count']
months_sorted = sorted(month_sum.keys())
monthly_avg = [month_sum[m]/month_count[m] for m in months_sorted]

plt.figure(figsize=(9,4))
plt.plot(months_sorted, monthly_avg, marker='o')
plt.title("Average Total Flights by Calendar Month (averaged across years)")
plt.xlabel("Month")
plt.ylabel("Average daily total flights")
plt.xticks(range(1,13))
plt.tight_layout()
plt.show()
print("Explanation: Average across months (1-12) shows expected seasonality. Good to mention summer tourist peaks.")
#The chart shows yearly seasonality: flights steadily rise from winter to summer, peak in July–August, and then decli
#during summers people have More leisure travel worldwide & its Summer vacation season
#Decline begins after September & dec is relatively slow
```



Explanation: Average across months (1-12) shows expected seasonality. Good to mention summer tourist peaks.

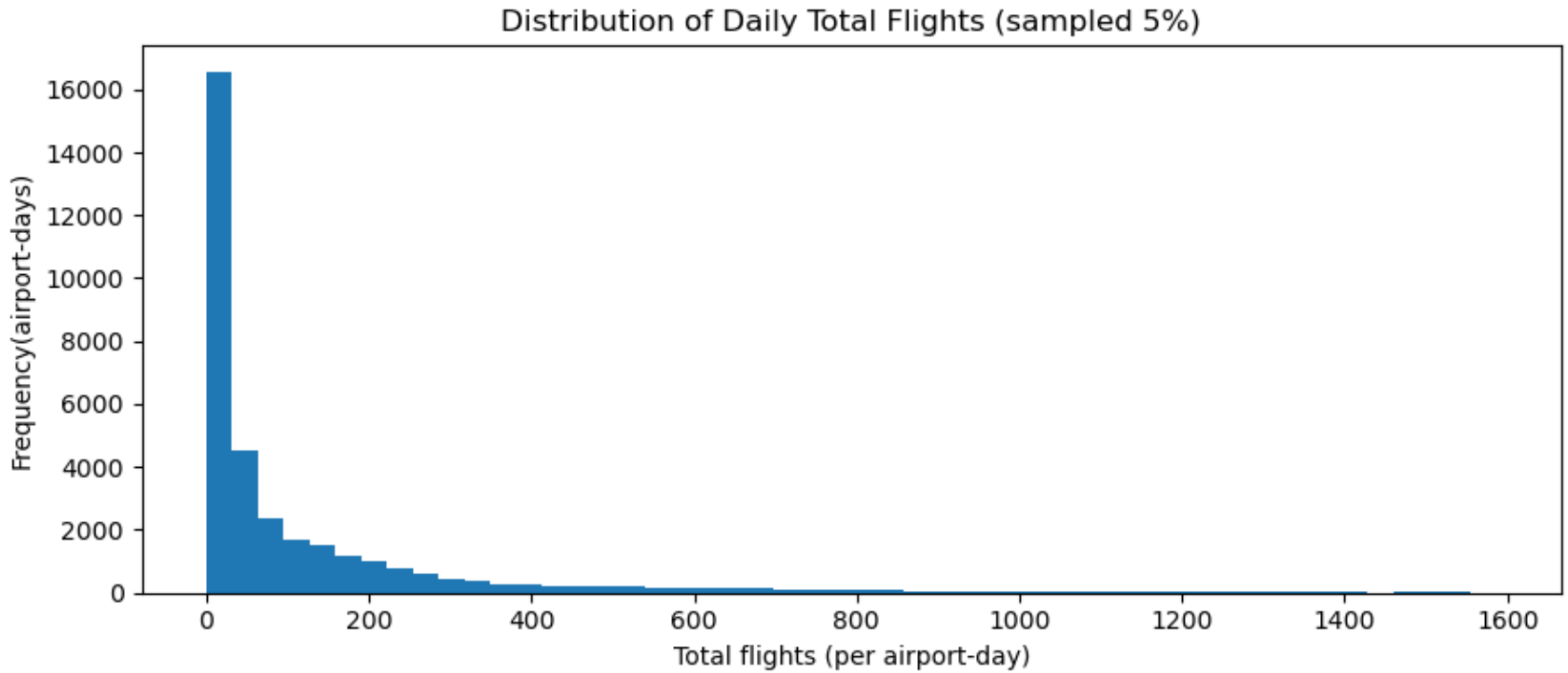
In [13]:
```
# 4.5 Weekday pattern (weekday vs weekend)
wd_sum = defaultdict(float); wd_count = defaultdict(int)
for chunk in pd.read_csv(cleaned_csv, parse_dates=['FLT_DATE'], iterator=True, chunksize=chunksize, low_memory=True):
    chunk['weekday'] = chunk['FLT_DATE'].dt.day_name()
    g = chunk.groupby('weekday')['FLT_TOT_1'].agg(['sum','count'])
    for wd,row in g.iterrows():
        wd_sum[wd] += row['sum']; wd_count[wd] += row['count']
wd_order = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']
wd_avg = [wd_sum[w]/wd_count[w] if wd_count[w]>0 else 0 for w in wd_order]

plt.figure(figsize=(9,4))
plt.bar(wd_order, wd_avg)
plt.title("Average Total Flights by Weekday")
plt.xlabel("Weekday")
plt.ylabel("Average total flights")
plt.tight_layout()
plt.show()
print("Monday, Thursday, and Friday have the highest average flights — around 130–135 flights per airport per day.")
```

```
#Average daily flights for that weekday, across all years and all airports.
```



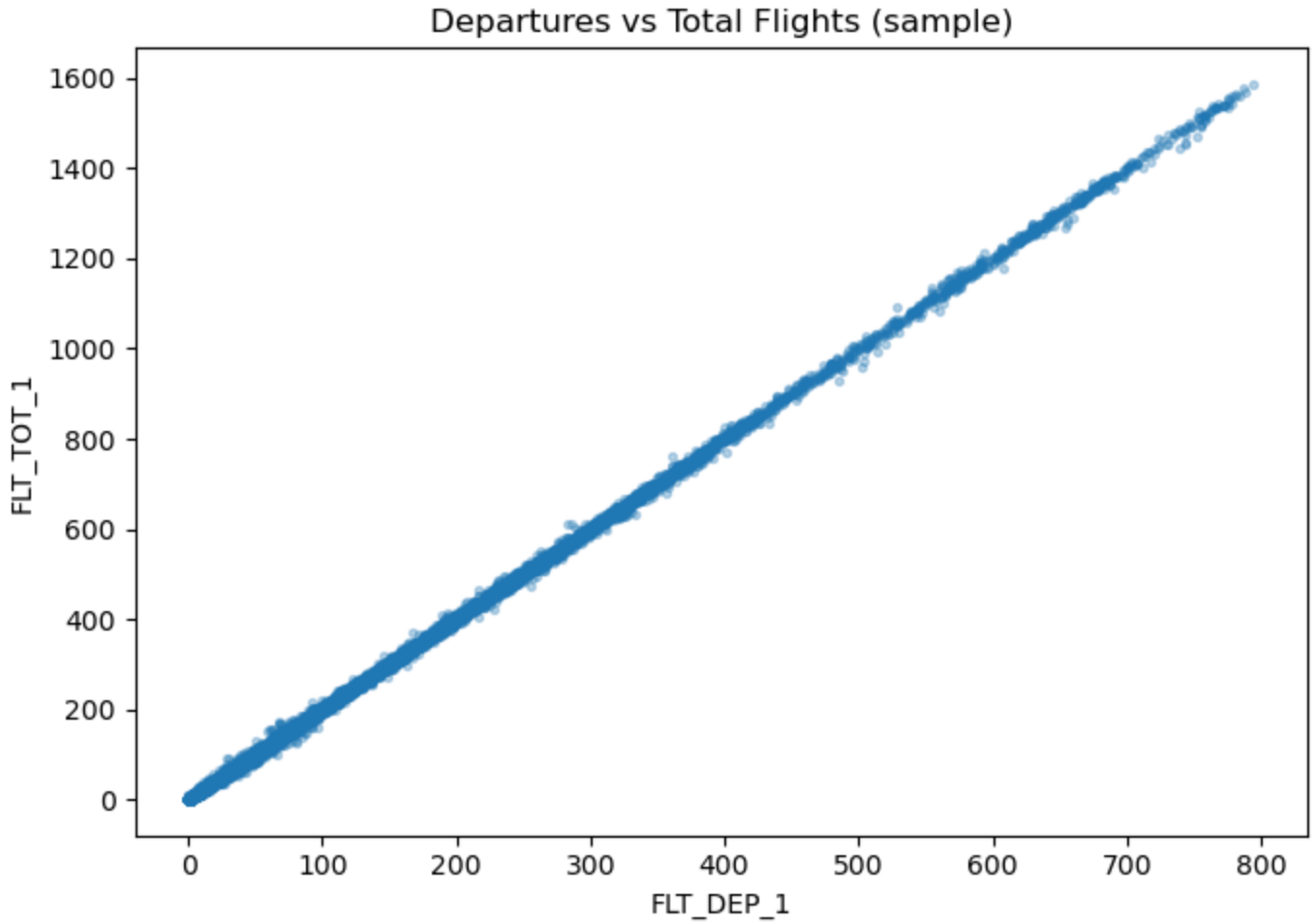
Monday, Thursday, and Friday have the highest average flights — around 130–135 flights per airport per day.

In [14]:
```
# 4.6 Distribution of FLT_TOT_1 (histogram, using sample)
plt.figure(figsize=(9,4))
plt.hist(sample_df['FLT_TOT_1'].dropna(), bins=50)
plt.title("Distribution of Daily Total Flights (sampled 5%)")
plt.xlabel("Total flights (per airport-day)")
plt.ylabel("Frequency(airport-days)")
plt.tight_layout()
plt.show()
print("The distribution is strongly right-skewed, indicating many low-volume days and a few extremely busy ones.")
#frequency: How many days had that many flights.
#Most airports are small → fewer flights per day → many days fall into the 0–100 range.
#There were 3000 airport-days with around 100 flights.eg
#There were 3000 airport-days where an airport had around 100 flights on that day.eg(just to understnd the graph)
```



The distribution is strongly right-skewed, indicating many low-volume days and a few extremely busy ones.

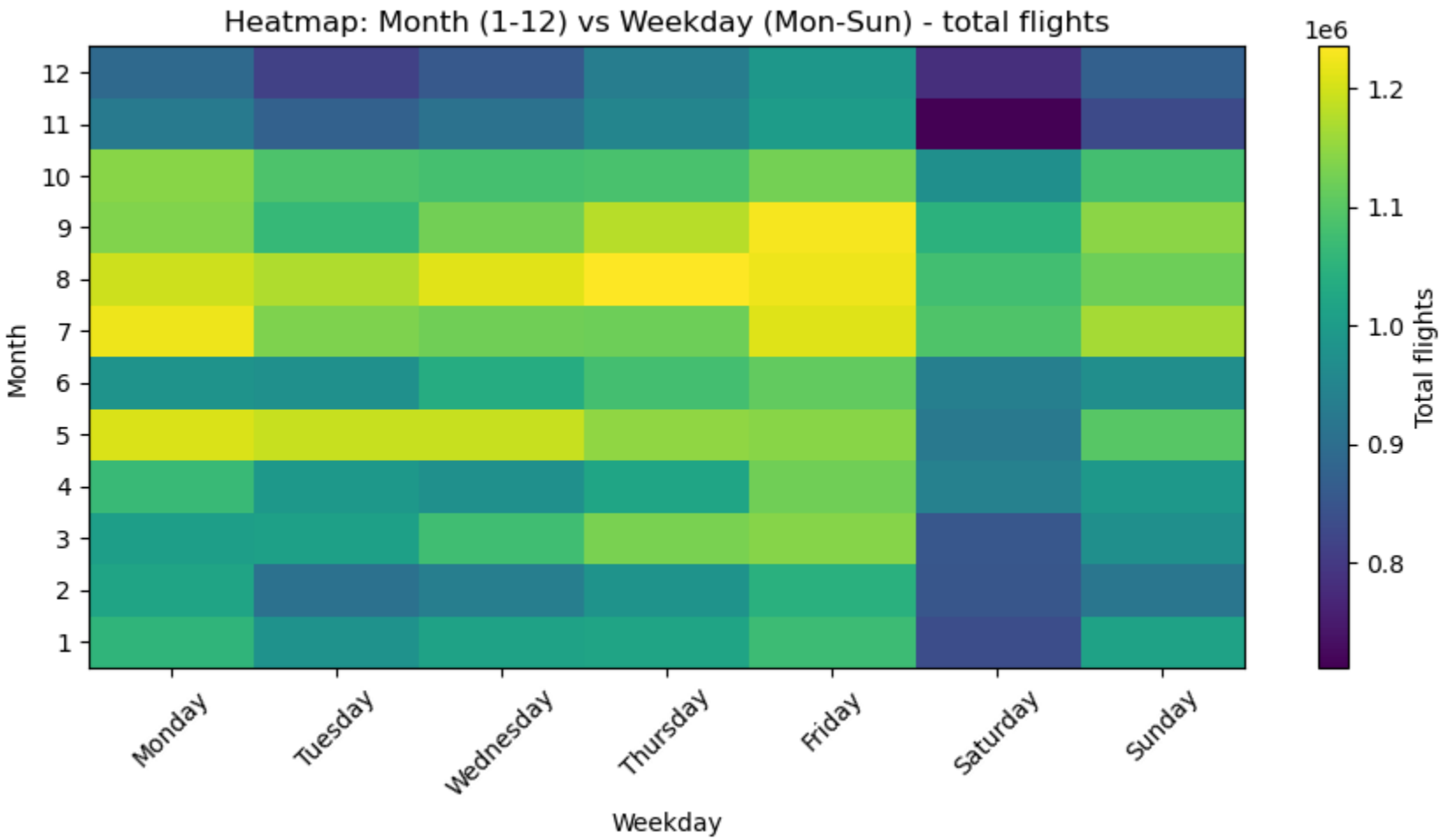
In [16]:
```
# 4.7 Departures vs Total flights scatter (sample)
plt.figure(figsize=(7,5))
plt.scatter(sample_df['FLT_DEP_1'], sample_df['FLT_TOT_1'], alpha=0.3, s=8)
plt.title("Departures vs Total Flights (sample)")
plt.xlabel("FLT_DEP_1")
plt.ylabel("FLT_TOT_1")
plt.tight_layout()
plt.show()
print("The tight upward line shows a very strong positive relationship—more departures always mean more total flights
```



The tight upward line shows a very strong positive relationship—more departures always mean more total flights.

In [17]:
```
# 4.8 Month vs Weekday heatmap (visual)
plt.figure(figsize=(9,5))
plt.imshow(heat_matrix, aspect='auto', origin='lower')
plt.colorbar(label='Total flights')
plt.yticks(ticks=range(12), labels=[str(i) for i in range(1,13)])
plt.xticks(ticks=range(7), labels=weekdays, rotation=45)
plt.title("Heatmap: Month (1-12) vs Weekday (Mon-Sun) - total flights")
plt.xlabel("Weekday")
plt.ylabel("Month")
plt.tight_layout()
plt.show()
```

```
print("Explanation: Shows hot spots (month-weekday combos) for traffic — great for visual storytelling.")
```

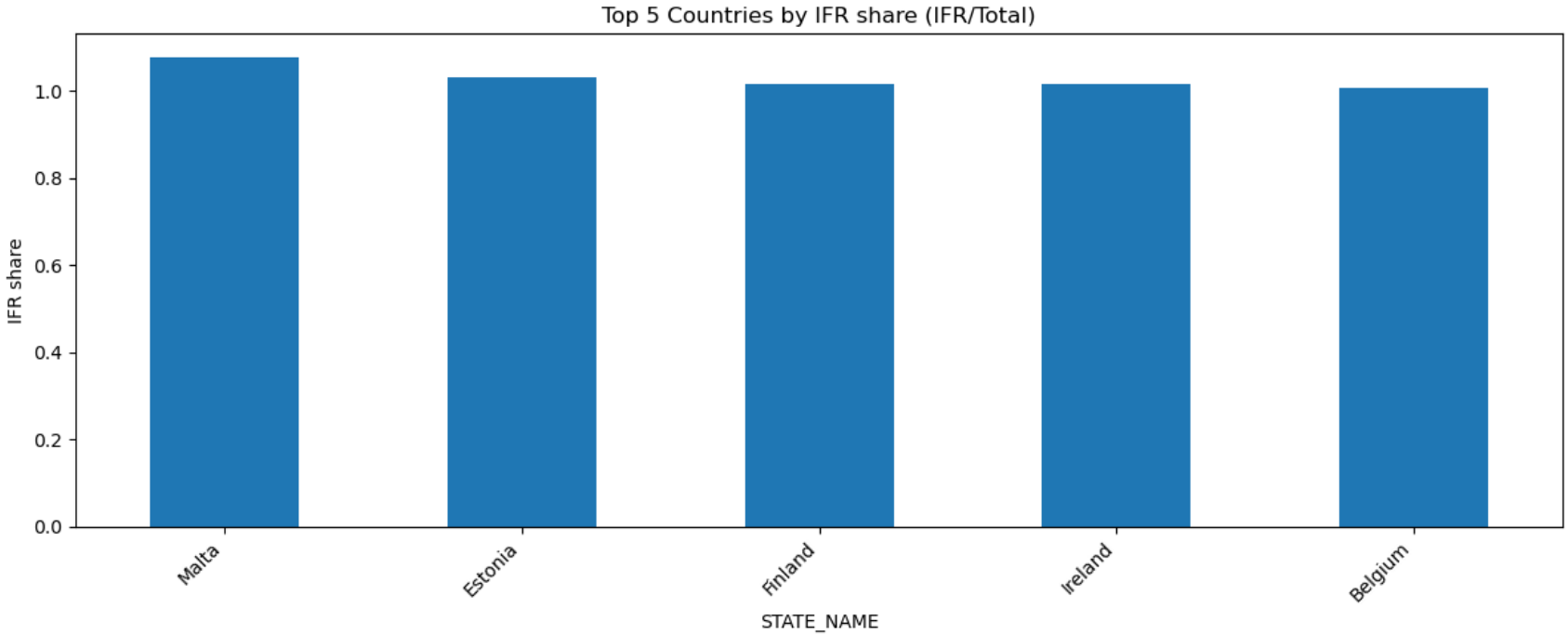



```
Explanation: Shows hot spots (month-weekday combos) for traffic — great for visual storytelling.
```

In [18]:
```
# 4.9 IFR proportion (if present)
if available_ifr_cols:
    print("IFR columns detected. Computing IFR share per country (this may take an extra moment).")
    # We'll compute IFR share per country using cleaned CSV if column exists
    if 'FLT_TOT_IFR_2' in pd.read_csv(cleaned_csv, nrows=1).columns:
        dfc = pd.read_csv(cleaned_csv, usecols=['STATE_NAME','FLT_TOT_1','FLT_TOT_IFR_2'])
        dfc = dfc.dropna(subset=['FLT_TOT_1','FLT_TOT_IFR_2'])
        country_ifr = dfc.groupby('STATE_NAME')[['FLT_TOT_1','FLT_TOT_IFR_2']].sum()
        country_ifr['IFR_share'] = country_ifr['FLT_TOT_IFR_2'] / country_ifr['FLT_TOT_1']
```

```
        country_ifr = country_ifr.sort_values('IFR_share', ascending=False)
        plt.figure(figsize=(12,5))
        country_ifr['IFR_share'].head(5).plot(kind='bar')
        plt.title("Top 5 Countries by IFR share (IFR/Total)")
        plt.ylabel("IFR share")
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()
        print("The top countries have nearly all IFR flights, showing they rely mostly on commercial, instrument-guid
else:
    print("No IFR columns in dataset; skipped IFR chart.")
```

```
IFR columns detected. Computing IFR share per country (this may take an extra moment).
```




```
The top countries have nearly all IFR flights, showing they rely mostly on commercial, instrument-guided aviation.
```

In [19]:
```
# Save quick summary tables (CSV) for report
country_csv = os.path.join(os.path.dirname(file_path), "top_50_countries_by_flights.csv")
airport_csv = os.path.join(os.path.dirname(file_path), "top_100_airports_by_flights.csv")
country_s.head(50).to_csv(country_csv, header=['total_flights'])
airport_s.head(100).to_csv(airport_csv, header=['total_flights'])
print("Saved summary CSVs:", country_csv, "and", airport_csv)
```

```
Saved summary CSVs: C:\Users\ancha\Desktop\top_50_countries_by_flights.csv and C:\Users\ancha\Desktop\top_100_airport
s_by_flights.csv
```

In [20]:
```
# ---------- 5. FEATURE ENGINEERING & MODEL PREP ----------
print("\n-- MODEL PREPARATION--")
```

```
-- MODEL PREPARATION--
```

In [21]:
```
df_model=pd.read_csv(cleaned_csv, parse_dates=['FLT_DATE'])
#df_model = pd.read_csv(cleaned_csv, parse_dates=['FLT_DATE'])
#dropping cols with null values
df_model=df_model.dropna(subset=['FLT_TOT_1']).copy()
#.copy() step is just preparing a working copy in memory for modeling (regression/classification). It doesn't overwri
```

In [22]:
```
# Create target for classification: high_traffic (1 if day's total > airport median)
# Create target for classification: high_traffic (1 if day's total > airport median)
df_model['median_by_airport']=df_model.groupby('APT_ICAO')['FLT_TOT_1'].transform('median')
df_model['high_traffic']=(df_model['FLT_TOT_1'] > df_model['median_by_airport']).astype(int)
```

In [23]:
```
# Create some features
df_model['month'] = df_model['FLT_DATE'].dt.month
df_model['weekday_num'] = df_model['FLT_DATE'].dt.dayofweek
# Ensure we do NOT use FLT_DEP_1 or FLT_ARR_1 as model inputs (they leak the target)
df_model.drop(columns=['FLT_DEP_1', 'FLT_ARR_1'], inplace=True, errors='ignore')
```

In [24]:
```
# Label encode categorical features
le_apt = LabelEncoder(); df_model['APT_code'] = le_apt.fit_transform(df_model['APT_ICAO'].astype(str))
le_name = LabelEncoder(); df_model['APTNAME_code'] = le_name.fit_transform(df_model['APT_NAME'].astype(str))
le_state = LabelEncoder(); df_model['STATE_code'] = le_state.fit_transform(df_model['STATE_NAME'].astype(str))
```

In [25]:
```
# Features & targets (NO-LEAK)
features = ['APT_code','APTNAME_code','STATE_code','month','weekday_num']
target_reg = 'FLT_TOT_1'
target_clf = 'high_traffic'
```

In [ ]:
```
# To avoid heavy compute, sample if dataset big
"""SAMPLE_ROWS = 200000   # reduce to e.g., 100k if laptop struggles
if len(df_model) > SAMPLE_ROWS:
    df_sample = df_model.sample(n=SAMPLE_ROWS, random_state=42)
else:
    df_sample = df_model.copy()
```

```
print("Modeling rows:", len(df_sample))"""
```

In [26]:
```
#Using the total dataset
df_sample = df_model.copy()
print("Modeling rows:", len(df_sample))
```

```
Modeling rows: 688099
```

In [27]:
```
X = df_sample[features]
y_reg = df_sample[target_reg]
y_clf = df_sample[target_clf]

# Train/test split
X_train, X_test, y_train_reg, y_test_reg = train_test_split(X, y_reg, test_size=0.2, random_state=42)
X_train_c, X_test_c, y_train_clf, y_test_clf = train_test_split(X, y_clf, test_size=0.2, random_state=42)
```

In [28]:
```
#print("Columns available:", df_sample.columns.tolist())

print("Features used:", features)
print("Columns in X:", X.columns.tolist())
```

```
Features used: ['APT_code', 'APTNAME_code', 'STATE_code', 'month', 'weekday_num']
Columns in X: ['APT_code', 'APTNAME_code', 'STATE_code', 'month', 'weekday_num']
```

In [ ]:
```
# ---------- 6. MODELING: REGRESSION (predict FLT_TOT_1) ----------
print("\n--- REGRESSION MODELS (predict total flights) ---")
```

In [29]:
```
# 6.1 Linear Regression (baseline)
lr = LinearRegression()
lr.fit(X_train, y_train_reg)
pred_lr = lr.predict(X_test)
rmse_lr = mean_squared_error(y_test_reg, pred_lr, squared=False)
r2_lr = r2_score(y_test_reg, pred_lr)
print(f"Linear Regression -> RMSE: {rmse_lr:.3f}, R2: {r2_lr:.3f}")
print("Why: baseline linear model. Good to compare against more complex models.")
```

```
Linear Regression -> RMSE: 220.739, R2: 0.013
Why: baseline linear model. Good to compare against more complex models.
```

In [30]:
```
# 6.2 Random Forest Regressor
rf = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train_reg)
```

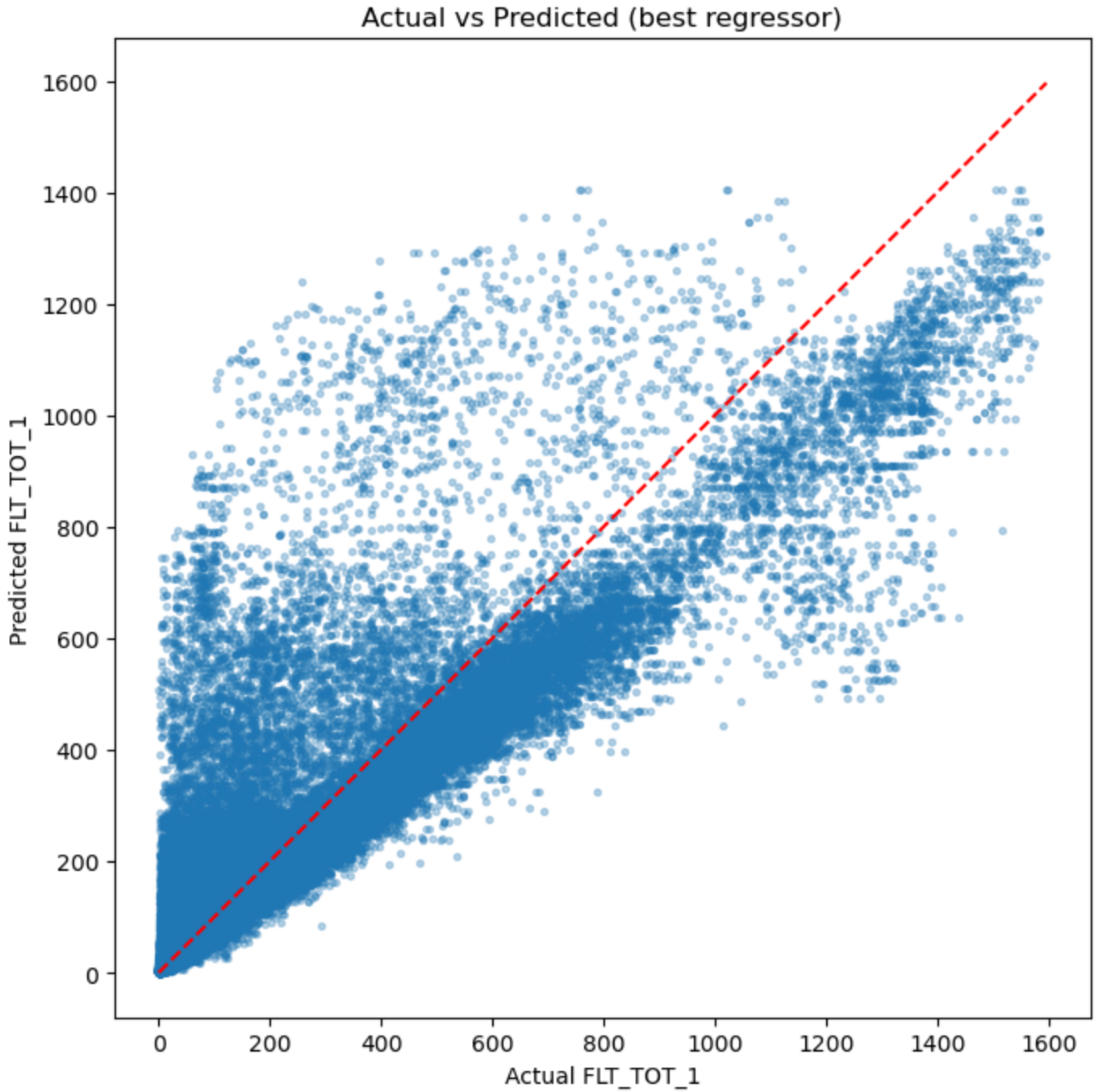
```
pred_rf = rf.predict(X_test)
rmse_rf = mean_squared_error(y_test_reg, pred_rf, squared=False)
r2_rf = r2_score(y_test_reg, pred_rf)
print(f"Random Forest -> RMSE: {rmse_rf:.3f}, R2: {r2_rf:.3f}")
print("Why: captures non-linear interactions (airport x month), often much better than linear.")
```

```
Random Forest -> RMSE: 96.048, R2: 0.813
Why: captures non-linear interactions (airport x month), often much better than linear.
```

In [31]:
```
# 6.3 Gradient Boosting Regressor
gb = GradientBoostingRegressor()
gb.fit(X_train, y_train_reg)
pred_gb = gb.predict(X_test)
rmse_gb = mean_squared_error(y_test_reg, pred_gb, squared=False)
r2_gb = r2_score(y_test_reg, pred_gb)
print(f"Gradient Boosting -> RMSE: {rmse_gb:.3f}, R2: {r2_gb:.3f}")
print("Why: boosting often gives best performance; use tuning to improve further.")
```

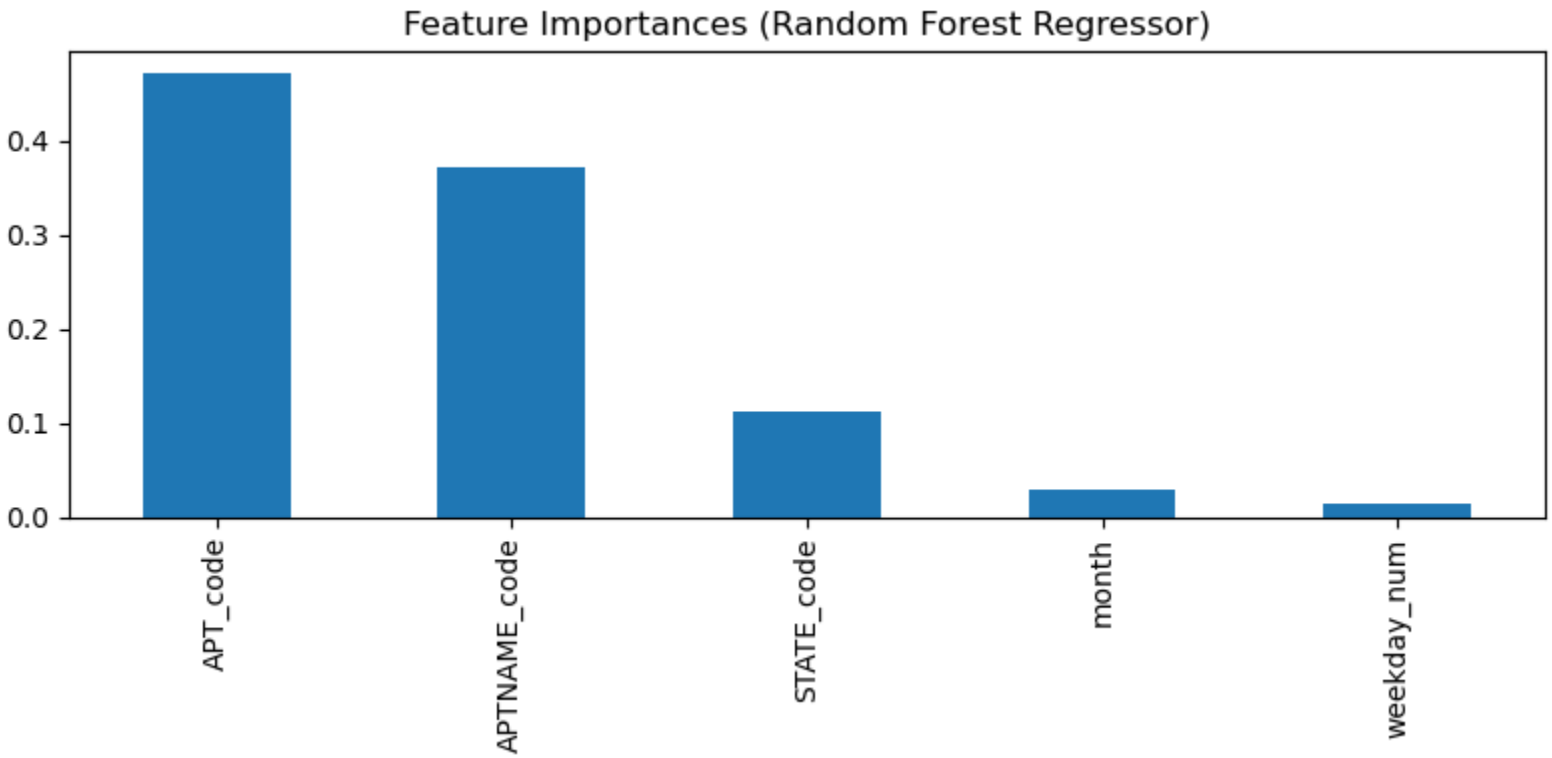
```
Gradient Boosting -> RMSE: 142.854, R2: 0.586
Why: boosting often gives best performance; use tuning to improve further.
```

In [32]:
```
# Plot Actual vs Predicted for best model (choose RF or GB by RMSE)
best_pred = pred_rf if rmse_rf <= rmse_gb else pred_gb
plt.figure(figsize=(7,7))
plt.scatter(y_test_reg, best_pred, alpha=0.3, s=8)
plt.plot([y_test_reg.min(), y_test_reg.max()], [y_test_reg.min(), y_test_reg.max()], 'r--')
plt.xlabel("Actual FLT_TOT_1")
plt.ylabel("Predicted FLT_TOT_1")
plt.title("Actual vs Predicted (best regressor)")
plt.tight_layout()
plt.show()
print("Explanation: Points close to the diagonal are good predictions. Look for bias or spread.")
```



Explanation: Points close to the diagonal are good predictions. Look for bias or spread.

In [33]:
```
# Feature importance (from Random Forest)
fi = pd.Series(rf.feature_importances_, index=features).sort_values(ascending=False)
plt.figure(figsize=(8,4))
fi.plot(kind='bar')
plt.title("Feature Importances (Random Forest Regressor)")
plt.tight_layout()
plt.show()
print("Why: shows which features drive model predictions. Airport and month are often top features.")
```




```
Why: shows which features drive model predictions. Airport and month are often top features.
```

In [ ]:
```
# ---------- 7. MODELING: CLASSIFICATION (predict high_traffic) ----------
```

In [34]:
```
print("\n--- CLASSIFICATION MODELS (predict high_traffic) ---")
```

```
--- CLASSIFICATION MODELS (predict high_traffic) ---
```

In [35]:
```
# 7.1 Logistic Regression (baseline)
logr = LogisticRegression(max_iter=300)
```

```
logr.fit(X_train_c, y_train_clf)
pred_logr = logr.predict(X_test_c)
acc_logr = accuracy_score(y_test_clf, pred_logr)
print("Logistic Regression -> Accuracy:", round(acc_logr,3))
print(classification_report(y_test_clf, pred_logr))
print("Why: quick baseline that gives probability outputs; easy to interpret.")
#This means Logistic Regression correctly predicted 56% of days.
#When it predicts low traffic → 57% correct
#When it predicts high traffic → 54% correct
#It's okay but not great — because it is a simple model.
```

```
Logistic Regression -> Accuracy: 0.562
              precision    recall  f1-score   support

           0       0.57      0.69      0.62     73252
           1       0.54      0.42      0.47     64368

    accuracy                           0.56    137620
   macro avg       0.56      0.55      0.55    137620
weighted avg       0.56      0.56      0.55    137620

Why: quick baseline that gives probability outputs; easy to interpret.
```

In [36]:
```
# 7.2 Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
rfc.fit(X_train_c, y_train_clf)
pred_rfc = rfc.predict(X_test_c)
acc_rfc = accuracy_score(y_test_clf, pred_rfc)
print("Random Forest Classifier -> Accuracy:", round(acc_rfc,3))
print(classification_report(y_test_clf, pred_rfc))
print("Why: robust classifier that handles non-linearities and interactions.")
```

```
Random Forest Classifier -> Accuracy: 0.701
              precision    recall  f1-score   support

           0       0.74      0.67      0.70     73252
           1       0.66      0.74      0.70     64368

    accuracy                           0.70    137620
   macro avg       0.70      0.70      0.70    137620
weighted avg       0.71      0.70      0.70    137620

Why: robust classifier that handles non-linearities and interactions.
```

In [37]:
```
# Confusion matrix for R.F.
cm = confusion_matrix(y_test_clf, pred_rfc)
print("Confusion matrix (rows=true, cols=pred):\n", cm)
```

```
Confusion matrix (rows=true, cols=pred):
 [[48961 24291]
 [16794 47574]]
```

In [38]:
```
# Precision/Recall tradeoff advice (text)
print("\nAdvice: If your operational goal is to catch busy days, prioritize recall for class=1 (high_traffic).")
print("If you want to avoid false alarms (not overstaffing), prioritize precision.")
```

```
Advice: If your operational goal is to catch busy days, prioritize recall for class=1 (high_traffic).
If you want to avoid false alarms (not overstaffing), prioritize precision.
```

In [44]:
```
# ---------- 8. SAVEING MODELING OUTPUTS (SAMPLE PREDICTIONS) ----------
out_pred = os.path.join(os.path.dirname(file_path), "model_predictions_sample.csv")
X_test_out = X_test.copy()
X_test_out['actual_FLT_TOT_1'] = y_test_reg.values
X_test_out['pred_RF'] = pred_rf
X_test_out['pred_LR'] = pred_lr
X_test_out.to_csv(out_pred, index=False)
```

```
print("Saved sample predictions to:", out_pred)
```

```
Saved sample predictions to: C:\Users\ancha\Desktop\model_predictions_sample.csv
```

In [42]:
```
# ---------- 9. SIMPLE INTERPRETATION ----------
print("\n--- REPORT SNIPPETS ---")
print("Regression interpretation example:")
print(f"- Best regressor RMSE = {min(rmse_rf, rmse_gb, rmse_lr):.2f} flights (lower is better).")
print(f"- Best R2 = {max(r2_rf, r2_gb, r2_lr):.3f} (closer to 1 is better).")

print("\nClassification interpretation example:")
print(f"- Random Forest accuracy = {acc_rfc:.3f}. Use classification_report above for precision/recall.")

print("\nEnd of notebook.")
```

```
--- REPORT SNIPPETS ---
Regression interpretation example:
- Best regressor RMSE = 96.05 flights (lower is better).
- Best R2 = 0.813 (closer to 1 is better).

Classification interpretation example:
- Random Forest accuracy = 0.701. Use classification_report above for precision/recall.

End of notebook.
```