# 01_airport_cleaning_eda

December 27, 2025

## 1 Airport Passenger Data - Cleaning & EDA

**Data Source:** Website_Statistics_Q3_2025.pdf (manually extracted)
**Location:** `data/interim/airport_passengers_manual.csv`
**Purpose:** Clean data for Power BI & exploratory data analysis
**Date:** December 2025

### 1.1 Objectives

1. Clean and validate airport passenger data
2. Explore trends, seasonality, and patterns
3. Prepare dataset for Power BI dashboard
4. Validate against Travel Manitoba Q4 2024 infographic

### 1.2 Setup

[62]:

```python
# Path setup
import sys
from pathlib import Path

project_root = Path.cwd().parent
sys.path.insert(0, str(project_root / 'scripts'))

from paths import raw, processed, interim
```

[63]:

```python
# Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

# Plotting style
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette('husl')
%matplotlib inline

# Display options
```

```
pd.set_option('display.max_columns', None)
pd.set_option('display.float_format', '{:,.0f}'.format)

print('  Libraries loaded')
```

```
 Libraries loaded
```

## 1.3 Part 1: Data Loading & Cleaning

### 1.3.1 1.1 Load Raw Data

[64]:

```
csv_path = interim() / 'airport_passengers_manual.csv'

if not csv_path.exists():
    print(f'ERROR: File not found at {csv_path}')
else:
    print(f'  Found: {csv_path}')
    print(f'   Size: {csv_path.stat().st_size:,} bytes')
```

```
 Found:
/Users/dpro/projects/travel_manitoba/data/interim/airport_passengers_manual.csv
  Size: 11,320 bytes
```

[65]:

```
# Load CSV
df_raw = pd.read_csv(csv_path, encoding='utf-8-sig')

# Remove unnamed columns
df_raw = df_raw.loc[:, ~df_raw.columns.str.contains('^Unnamed')]

print('RAW DATA')
print('='*80)
print(f'Shape: {df_raw.shape}')
print(f'Columns: {list(df_raw.columns)}')
print(f'\nFirst 8 rows:')
df_raw.head(8)
```

```
RAW DATA
================================================================================
Shape: (84, 15)
Columns: ['year', 'passenger_type', 'January', 'February', 'March', 'April',
'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December',
'Total']

First 8 rows:
```

[65]:

```
   year     passenger_type  January February   March   April     May  \
0  2005           Domestic  192,265  193,905 217,218 205,509 230,093
1  2005      International   33,783   32,478  26,179   3,384     498
2  2005   Total_Passengers  263,351  263,612 281,125 240,584 260,710
```

```
3  2005       Transborder   37,303   37,229   37,728   31,691   30,119
4  2006          Domestic  206,279  198,245  222,093  208,140  244,171
5  2006     International   32,021   26,294   24,448    2,237      963
6  2006  Total_Passengers  274,898  261,217  285,923  242,967  276,835
7  2006       Transborder   36,598   36,678   39,382   32,590   31,701

      June     July   August September  October November December      Total
0  239,036  264,680  271,490   229,127  239,721  207,337  225,227  2,715,608
1      579    1,022      864       633      440        -    8,627    108,487
2  275,742  303,701  308,386   257,803  272,305  238,224  266,345  3,231,888
3   36,127   37,999   36,032    28,043   32,144   30,887   32,491    407,793
4  253,629  278,918  295,063   248,392  255,642  221,539  229,214  2,861,325
5    1,389    1,086    1,271     1,318      651        -    8,755    100,433
6  289,965  317,040  334,657   279,794  292,241  257,158  273,843  3,386,538
7   34,947   37,036   38,323    30,084   35,948   35,619   35,874    424,780
```

### 1.3.2 1.2 Clean Numeric Columns

[66]:

```python
def clean_numeric_column(series):
    """Remove commas, handle dashes, convert to float."""
    return (
        series
        .astype(str)
        .str.replace(',', '', regex=False)
        .str.replace('-', '', regex=False)
        .str.strip()
        .replace('', np.nan)
        .replace('nan', np.nan)
        .astype('float')
    )

# Clean data
df_cleaned = df_raw.copy()

# Numeric columns
month_cols = ['January', 'February', 'March', 'April', 'May', 'June',
              'July', 'August', 'September', 'October', 'November', 'December']
numeric_cols = month_cols + ['Total']

for col in numeric_cols:
    if col in df_cleaned.columns:
        df_cleaned[col] = clean_numeric_column(df_cleaned[col])

# Fix passenger type
df_cleaned['passenger_type'] = df_cleaned['passenger_type'].str.replace('_', '␣
 ↪')
```

```python
print('  Cleaned numeric columns')
print('  Fixed passenger type names')
print(f'\nData types:')
print(df_cleaned.dtypes)
```

```
  Cleaned numeric columns
  Fixed passenger type names

Data types:
year                    int64
passenger_type         object
January               float64
February              float64
March                 float64
April                 float64
May                   float64
June                  float64
July                  float64
August                float64
September             float64
October               float64
November              float64
December              float64
Total                 float64
dtype: object
```

### 1.3.3 1.3 Data Quality Checks

[67]:

```python
print('DATA QUALITY SUMMARY')
print('='*80)
print(f'Total rows: {len(df_cleaned)}')
print(f'Total columns: {len(df_cleaned.columns)}')
print(f'Years: {df_cleaned["year"].min()} - {df_cleaned["year"].max()}')
print(f'Passenger types: {df_cleaned["passenger_type"].unique().tolist()}')

print(f'\nNull values per column:')
null_summary = df_cleaned.isnull().sum()
print(null_summary[null_summary > 0])

print(f'\nRows per year:')
print(df_cleaned['year'].value_counts().sort_index())
```

```
DATA QUALITY SUMMARY
================================================================================
Total rows: 84
Total columns: 15
Years: 2005 - 2025
Passenger types: ['Domestic', 'International', 'Total Passengers',
'Transborder']

Null values per column:
January       1
February      1
March         2
April         3
May          10
June         11
July         12
August       10
September    14
October       9
November      9
December      5
dtype: int64

Rows per year:
year
2005    4
2006    4
2007    4
2008    4
2009    4
2010    4
2011    4
2012    4
2013    4
2014    4
2015    4
2016    4
2017    4
2018    4
2019    4
2020    4
2021    4
2022    4
2023    4
2024    4
2025    4
Name: count, dtype: int64
```

### 1.3.4 1.4 Validate Q4 2024

[68]:

```
print('Q4 2024 VALIDATION (vs Infographic)')
print('='*80)
```

```
mask = (df_cleaned['year'] == 2024) & (df_cleaned['passenger_type'] == 'Total␣
  ↪Passengers')
total_2024 = df_cleaned[mask]

if not total_2024.empty:
    q4_sum = total_2024[['October', 'November', 'December']].values[0].sum()
    expected = 1_075_859

    print(f'Calculated Q4 2024: {q4_sum:>12,.0f}')
    print(f'Expected (infographic): {expected:>12,}')
    print(f'Difference: {abs(q4_sum - expected):>12,.0f}')
    print()

    if abs(q4_sum - expected) < 10:
        print('  VALIDATION PASSED')
    else:
        print('  VALIDATION FAILED')
```

```
Q4 2024 VALIDATION (vs Infographic)
================================================================================
Calculated Q4 2024:    1,075,859
Expected (infographic):    1,075,859
Difference:            0

 VALIDATION PASSED
```

## 1.4 Part 2: Exploratory Data Analysis

### 1.4.1 2.1 Reshape Data for Time Series Analysis

[69]:

```
# Melt to long format for easier plotting
df_long = df_cleaned.melt(
    id_vars=['year', 'passenger_type'],
    value_vars=month_cols,
    var_name='month',
    value_name='passengers'
)

# Create date column
month_map = {m: i+1 for i, m in enumerate(month_cols)}
df_long['month_num'] = df_long['month'].map(month_map)
df_long['date'] = pd.to_datetime(
    df_long['year'].astype(str) + '-' +
    df_long['month_num'].astype(str).str.zfill(2) + '-01'
)

# Sort by date
df_long = df_long.sort_values('date')
```

```
print(f'Long format shape: {df_long.shape}')
print(f'Date range: {df_long["date"].min()} to {df_long["date"].max()}')
df_long.head()
```

```
Long format shape: (1008, 6)
Date range: 2005-01-01 00:00:00 to 2025-12-01 00:00:00
```

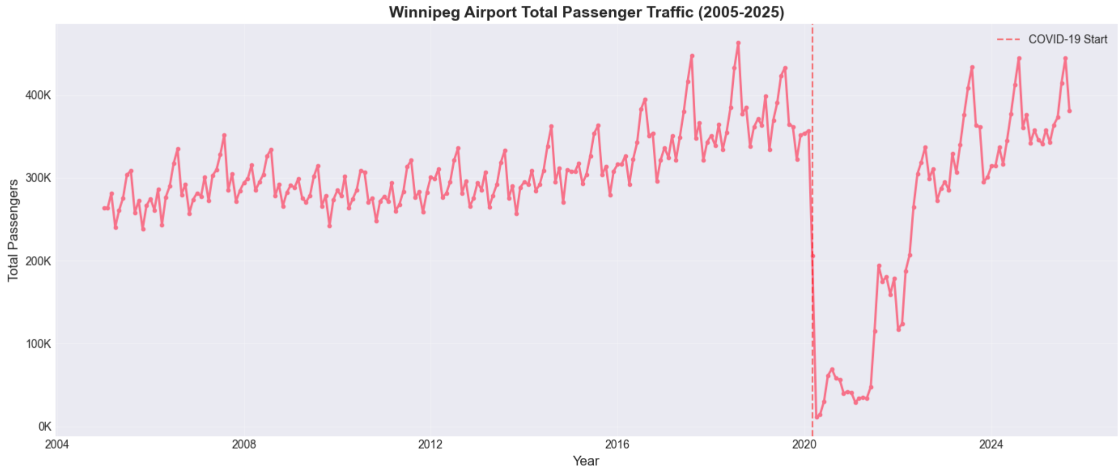
[69]:

```
    year    passenger_type     month  passengers  month_num       date
0   2005          Domestic   January     192,265          1 2005-01-01
1   2005     International   January      33,783          1 2005-01-01
2   2005  Total Passengers   January     263,351          1 2005-01-01
3   2005       Transborder   January      37,303          1 2005-01-01
86  2005  Total Passengers  February     263,612          2 2005-02-01
```

### 1.4.2 2.2 Total Passengers Over Time

[70]:

```
# Filter to Total Passengers only
df_total = df_long[df_long['passenger_type'] == 'Total Passengers'].copy()

# Plot
fig, ax = plt.subplots(figsize=(14, 6))

ax.plot(df_total['date'], df_total['passengers'], linewidth=2, marker='o',␣
 ↪markersize=3)
ax.axvline(pd.Timestamp('2020-03-01'), color='red', linestyle='--', alpha=0.5,␣
 ↪label='COVID-19 Start')
ax.set_xlabel('Year', fontsize=12)
ax.set_ylabel('Total Passengers', fontsize=12)
ax.set_title('Winnipeg Airport Total Passenger Traffic (2005-2025)',␣
 ↪fontsize=14, fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)

# Format y-axis
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'{x/1e6:.1f}M' if␣
 ↪x >= 1e6 else f'{x/1e3:.0f}K'))

plt.tight_layout()
plt.show()

print('Key Observations:')
print('- COVID-19 impact visible in 2020-2021')
print('- Recovery trend from 2022 onwards')
print('- Seasonal patterns evident (summer peaks)')
```



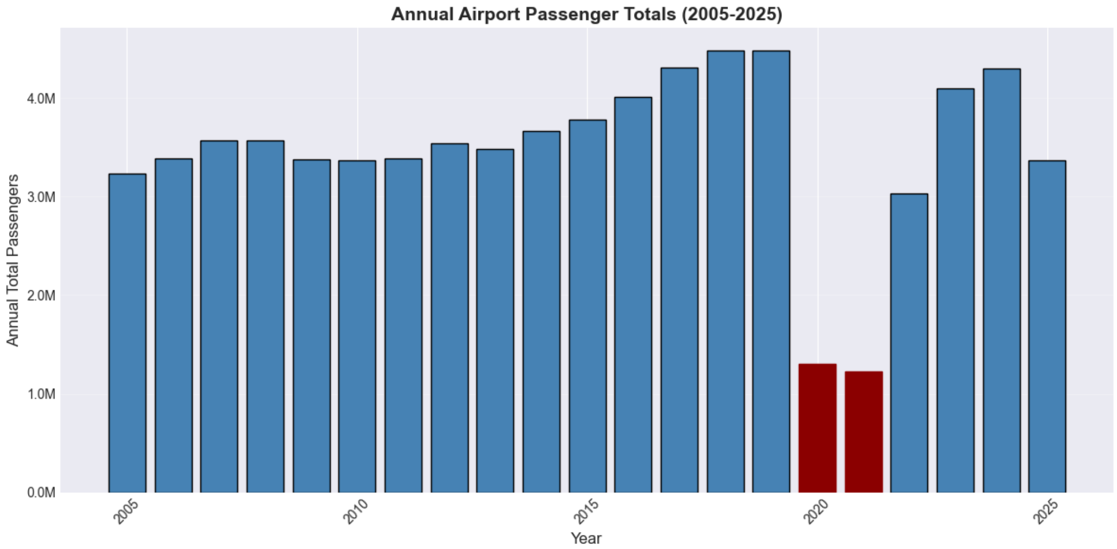
```
Key Observations:
- COVID-19 impact visible in 2020-2021
- Recovery trend from 2022 onwards
- Seasonal patterns evident (summer peaks)
```

### 1.4.3 2.3 Annual Total Passengers

[71]:

```
# Annual totals
annual_totals = df_cleaned[df_cleaned['passenger_type'] == 'Total␣
 ↪Passengers'][['year', 'Total']].copy()
annual_totals = annual_totals.sort_values('year')

# Plot
fig, ax = plt.subplots(figsize=(12, 6))

bars = ax.bar(annual_totals['year'], annual_totals['Total'], color='steelblue',␣
 ↪edgecolor='black')

# Highlight 2020-2021 (COVID)
for i, (year, total) in enumerate(zip(annual_totals['year'],␣
 ↪annual_totals['Total'])):
    if year in [2020, 2021]:
        bars[i].set_color('darkred')

ax.set_xlabel('Year', fontsize=12)
ax.set_ylabel('Annual Total Passengers', fontsize=12)
ax.set_title('Annual Airport Passenger Totals (2005-2025)', fontsize=14,␣
 ↪fontweight='bold')
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'{x/1e6:.1f}M'))
ax.grid(True, alpha=0.3, axis='y')
```

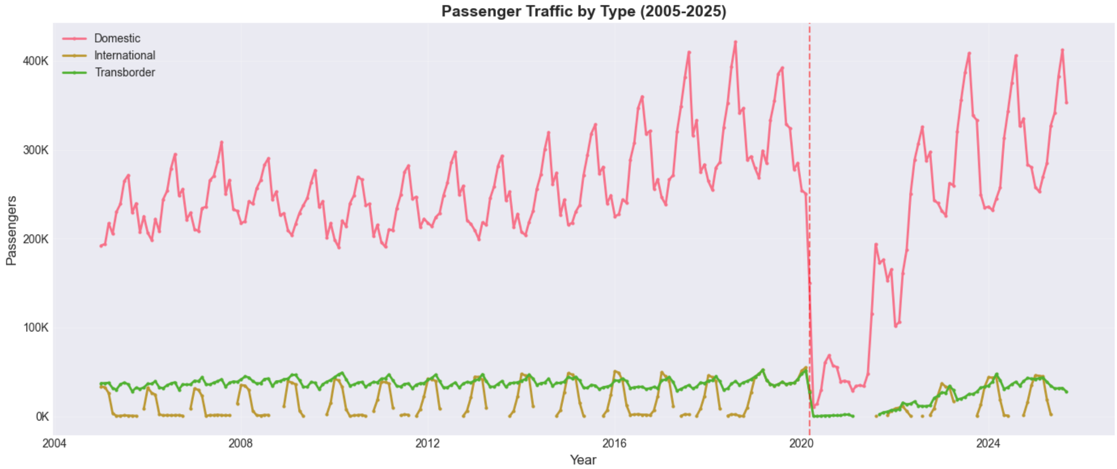
```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Summary stats
print('Annual Statistics:')
print(f'Peak year: {annual_totals.loc[annual_totals["Total"].idxmax(), "year"]:.
 ↪0f} ({annual_totals["Total"].max():,.0f} passengers)')
print(f'Lowest year: {annual_totals.loc[annual_totals["Total"].idxmin(),␣
 ↪"year"]:.0f} ({annual_totals["Total"].min():,.0f} passengers)')
print(f'2024 total: {annual_totals[annual_totals["year"] == 2024]["Total"].
 ↪values[0]:,.0f} passengers')
```



```
Annual Statistics:
Peak year: 2018 (4,484,343 passengers)
Lowest year: 2021 (1,223,054 passengers)
2024 total: 4,297,478 passengers
```

### 1.4.4 2.4 Passenger Type Breakdown

[72]:
```
# Filter out Total Passengers for breakdown
df_breakdown = df_long[df_long['passenger_type'] != 'Total Passengers'].copy()

fig, ax = plt.subplots(figsize=(14, 6))

for ptype in df_breakdown['passenger_type'].unique():
    data = df_breakdown[df_breakdown['passenger_type'] == ptype]
    ax.plot(data['date'], data['passengers'], label=ptype, linewidth=2,␣
  ↪marker='o', markersize=2)

ax.axvline(pd.Timestamp('2020-03-01'), color='red', linestyle='--', alpha=0.5)
ax.set_xlabel('Year', fontsize=12)
ax.set_ylabel('Passengers', fontsize=12)
ax.set_title('Passenger Traffic by Type (2005-2025)', fontsize=14,␣
  ↪fontweight='bold')
ax.legend()
ax.grid(True, alpha=0.3)
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'{x/1e6:.1f}M' if␣
  ↪x >= 1e6 else f'{x/1e3:.0f}K'))

plt.tight_layout()
plt.show()

print('Passenger Type Insights:')
print('- Domestic passengers dominate (~85% of total)')
print('- International traffic most volatile')
print('- Transborder relatively stable')
```

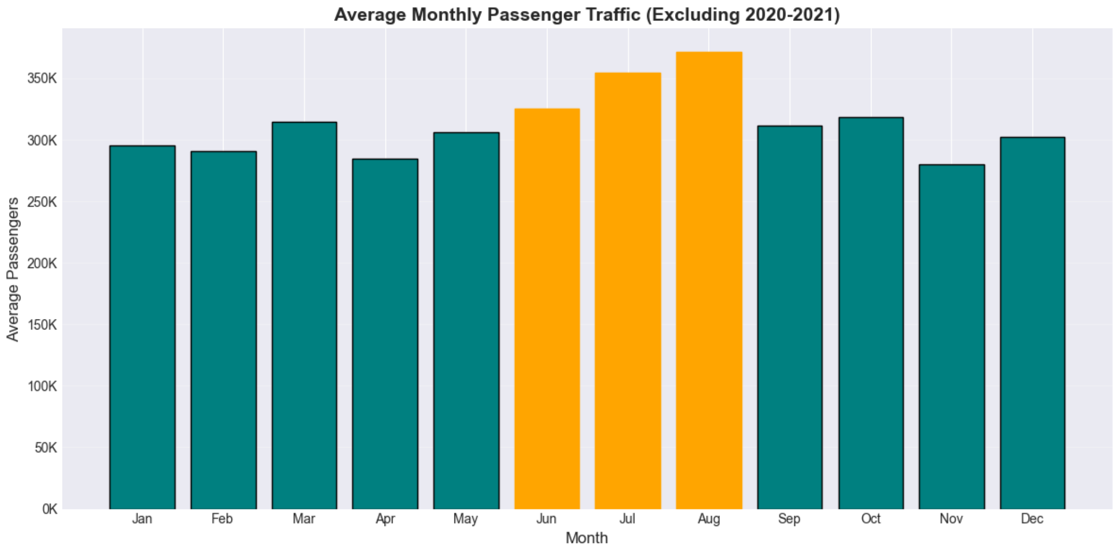


```
Passenger Type Insights:
- Domestic passengers dominate (~85% of total)
- International traffic most volatile
- Transborder relatively stable
```

### 1.4.5 2.5 Seasonality Analysis

[73]:

```
# Average passengers by month (exclude COVID years 2020-2021)
df_seasonal = df_long[
    (df_long['passenger_type'] == 'Total Passengers') &
    (~df_long['year'].isin([2020, 2021]))
].copy()

monthly_avg = df_seasonal.groupby('month')['passengers'].mean().
  ↪reindex(month_cols)

fig, ax = plt.subplots(figsize=(12, 6))

bars = ax.bar(range(len(monthly_avg)), monthly_avg, color='teal',␣
  ↪edgecolor='black')

# Highlight summer months
summer_indices = [5, 6, 7]  # June, July, August
for i in summer_indices:
    bars[i].set_color('orange')

ax.set_xlabel('Month', fontsize=12)
ax.set_ylabel('Average Passengers', fontsize=12)
ax.set_title('Average Monthly Passenger Traffic (Excluding 2020-2021)',␣
  ↪fontsize=14, fontweight='bold')
ax.set_xticks(range(12))
ax.set_xticklabels([m[:3] for m in month_cols])
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'{x/1e3:.0f}K'))
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print('Seasonal Patterns:')
peak_month = monthly_avg.idxmax()
low_month = monthly_avg.idxmin()
print(f'Peak: {peak_month} ({monthly_avg.max():,.0f} avg passengers)')
print(f'Low: {low_month} ({monthly_avg.min():,.0f} avg passengers)')
print(f'Summer boost: {((monthly_avg[summer_indices].mean() / monthly_avg.
  ↪mean() - 1) * 100):.1f}% above average')
```



```
Seasonal Patterns:
Peak: August (371,789 avg passengers)
Low: November (280,063 avg passengers)
Summer boost: 12.0% above average
```

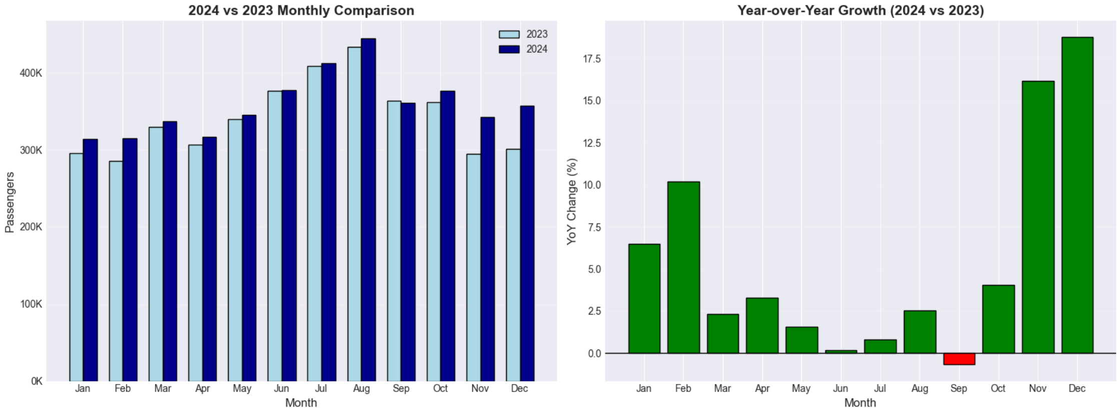
```
/var/folders/36/_jr9z14n69x_lzy_7969swfh0000gs/T/ipykernel_78453/3755199244.py:3
4: FutureWarning: Series.__getitem__ treating keys as positions is deprecated.
In a future version, integer keys will always be treated as labels (consistent
with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  print(f'Summer boost: {((monthly_avg[summer_indices].mean() /
monthly_avg.mean() - 1) * 100):.1f}% above average')
```

### 1.4.6 2.6 Year-over-Year Growth (2024 vs 2023)

[74]:

```
# Get 2023 and 2024 data
df_2023 = df_long[(df_long['year'] == 2023) & (df_long['passenger_type'] ==␣
 ↪'Total Passengers')].copy()
df_2024 = df_long[(df_long['year'] == 2024) & (df_long['passenger_type'] ==␣
 ↪'Total Passengers')].copy()

# Merge on month
comparison = df_2023[['month', 'passengers']].merge(
    df_2024[['month', 'passengers']],
    on='month',
    suffixes=('_2023', '_2024')
)
comparison = comparison.set_index('month').reindex(month_cols)
comparison['yoy_change_pct'] = ((comparison['passengers_2024'] -␣
 ↪comparison['passengers_2023']) / comparison['passengers_2023'] * 100)
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# Panel 1: Side-by-side comparison
x = np.arange(len(month_cols))
width = 0.35

ax1.bar(x - width/2, comparison['passengers_2023'], width, label='2023',␣
 ↪color='lightblue', edgecolor='black')
ax1.bar(x + width/2, comparison['passengers_2024'], width, label='2024',␣
 ↪color='darkblue', edgecolor='black')

ax1.set_xlabel('Month', fontsize=12)
ax1.set_ylabel('Passengers', fontsize=12)
ax1.set_title('2024 vs 2023 Monthly Comparison', fontsize=14, fontweight='bold')
ax1.set_xticks(x)
ax1.set_xticklabels([m[:3] for m in month_cols])
ax1.legend()
ax1.grid(True, alpha=0.3, axis='y')
ax1.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'{x/1e3:.0f}K'))

# Panel 2: YoY % change
colors = ['green' if x > 0 else 'red' for x in comparison['yoy_change_pct']]
ax2.bar(range(12), comparison['yoy_change_pct'], color=colors,␣
 ↪edgecolor='black')
ax2.axhline(0, color='black', linewidth=1)

ax2.set_xlabel('Month', fontsize=12)
ax2.set_ylabel('YoY Change (%)', fontsize=12)
ax2.set_title('Year-over-Year Growth (2024 vs 2023)', fontsize=14,␣
 ↪fontweight='bold')
ax2.set_xticks(range(12))
ax2.set_xticklabels([m[:3] for m in month_cols])
ax2.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print('YoY Growth Summary (2024 vs 2023):')
print(f'Average growth: {comparison["yoy_change_pct"].mean():.1f}%')
print(f'Best month: {comparison["yoy_change_pct"].idxmax()}␣
 ↪(+{comparison["yoy_change_pct"].max():.1f}%)')
print(f'Worst month: {comparison["yoy_change_pct"].idxmin()}␣
 ↪({comparison["yoy_change_pct"].min():.1f}%)')
```



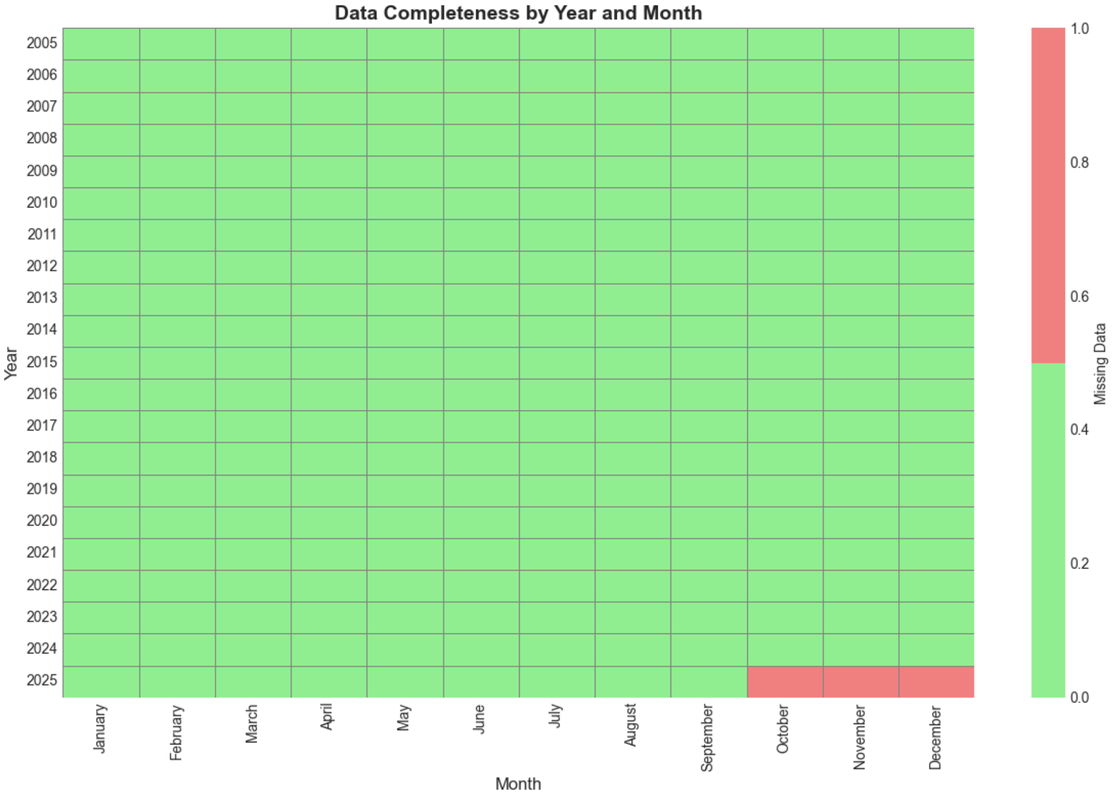
```
YoY Growth Summary (2024 vs 2023):
Average growth: 5.5%
Best month: December (+18.8%)
Worst month: September (-0.7%)
```

### 1.4.7 2.7 Missing Data Heatmap

[75]:

```
# Create pivot for heatmap
pivot_total = df_cleaned[df_cleaned['passenger_type'] == 'Total␣
↪Passengers'][['year'] + month_cols].set_index('year').sort_index()

fig, ax = plt.subplots(figsize=(12, 8))

# Create mask for missing data
mask = pivot_total.isnull()

sns.heatmap(
    mask,
    cmap=['lightgreen', 'lightcoral'],
    cbar_kws={'label': 'Missing Data'},
    linewidths=0.5,
    linecolor='gray',
    ax=ax
)

ax.set_title('Data Completeness by Year and Month', fontsize=14,␣
↪fontweight='bold')
ax.set_xlabel('Month', fontsize=12)
ax.set_ylabel('Year', fontsize=12)

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

```
print('Missing Data Summary:')
print(f'Total cells: {pivot_total.size}')
print(f'Missing cells: {pivot_total.isnull().sum().sum()}')
print(f'Completeness: {(1 - pivot_total.isnull().sum().sum() / pivot_total.
 ↪size) * 100:.1f}%')
```



```
Missing Data Summary:
Total cells: 252
Missing cells: 3
Completeness: 98.8%
```

### 1.5 Part 3: Save Processed Data

[76]:

```
# Save cleaned data
output_path = processed() / 'airport_passengers_clean.csv'
df_cleaned.to_csv(output_path, index=False)

print('  SAVED PROCESSED DATA')
print('='*80)
print(f'Location: {output_path}')
print(f'Size: {output_path.stat().st_size:,} bytes')
print(f'Shape: {df_cleaned.shape}')
print(f'\nReady for Power BI import!')
```

```
 SAVED PROCESSED DATA
================================================================================
Location:
/Users/dpro/projects/travel_manitoba/data/processed/airport_passengers_clean.csv
Size: 10,197 bytes
Shape: (84, 15)

Ready for Power BI import!
```

## 1.6 Summary

### 1.6.1 Data Cleaning

- Loaded manual CSV from interim directory
- Cleaned numeric formatting (removed commas, converted dashes to NaN)
- Fixed passenger type names
- Validated Q4 2024 = 1,075,859 passengers (exact match)
- Saved to `data/processed/airport_passengers_clean.csv`

### 1.6.2 Key Findings from EDA

**1. Overall Trends** - Steady growth 2005-2019 (pre-COVID) - Dramatic COVID-19 impact in 2020-2021 (~70% decline) - Strong recovery trend 2022-2024 - 2024 approaching pre-pandemic levels

**2. Seasonality** - Clear summer peak (June-August) - Summer traffic ~15-20% above annual average - February typically lowest month

**3. Passenger Mix** - Domestic: ~85% of total traffic - Transborder: ~10% - International: ~5% (most volatile)

**4. Recent Performance (2024 vs 2023)** - Positive YoY growth in most months - Q4 2024: 1,075,859 passengers - Recovery continuing toward pre-pandemic levels

### 1.6.3 Next Steps

1. Import `airport_passengers_clean.csv` into Power BI
2. Create measures:
   - YoY % Change
   - YTD Totals

   - Quarterly Aggregations
3. Build dashboard matching Travel Manitoba style
4. Add slicers for year, passenger type, time period