Medicare Savings

August 16, 2025

[1]:
```
# Cell 1: Setup and Imports
# Medicare Part D: Generic vs Brand Name Savings Analysis
# Case Study: How much could Medicare save by increasing generic drug adoption?

import pandas as pd
import requests
import numpy as np
import sqlite3
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

print("MEDICARE PART D: GENERIC vs BRAND NAME SAVINGS ANALYSIS")
print("=" * 80)
print(f"Analysis Date: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
print("Data Source: CMS Medicare Part D Spending by Drug API")
print("API Endpoint: https://data.cms.gov/data-api/v1/dataset/
  ↪7e0b4365-fd63-4a29-8f5e-e0ac9f66a81b/data")
print("Business Question: How much could Medicare save with generic adoption?")
print("=" * 80)
```

```
MEDICARE PART D: GENERIC vs BRAND NAME SAVINGS ANALYSIS
================================================================================
Analysis Date: 2025-08-16 14:05:40
Data Source: CMS Medicare Part D Spending by Drug API
API Endpoint: https://data.cms.gov/data-
api/v1/dataset/7e0b4365-fd63-4a29-8f5e-e0ac9f66a81b/data
Business Question: How much could Medicare save with generic adoption?
================================================================================
```

[2]:
```
# Functions to fetch data

def fetch_medicare_part_d_data(limit=5000, offset=0):
    """
    Fetch Medicare Part D data from CMS API

    Parameters:
    limit (int): Number of records to fetch (API appears to max at 5000)
```

```
    offset (int): Starting record number for pagination

    Returns:
    pandas.DataFrame: Medicare Part D drug spending data
    """

    base_url = "https://data.cms.gov/data-api/v1/dataset/
↪7e0b4365-fd63-4a29-8f5e-e0ac9f66a81b/data"

    # Try different parameter combinations that might work with this API
    params = {
        'size': limit,
        'offset': offset
    }

    # Alternative parameter names to try if the above doesn't work
    alt_params = [
        {'limit': limit, 'skip': offset},
        {'$limit': limit, '$offset': offset},
        {'per_page': limit, 'page': offset // limit + 1}
    ]

    try:
        print(f"Fetching Medicare Part D data (limit={limit}, offset={offset})..
↪.")

        # Try main parameters first
        response = requests.get(base_url, params=params, timeout=30)
        response.raise_for_status()

        data = response.json()
        df = pd.DataFrame(data)

        print(f"Successfully fetched {len(df)} records")

        # If we got fewer records than expected, let's check the response␣
↪headers for pagination info
        if len(df) < limit and offset == 0:
            print(f"Response headers (for debugging):")
            for key, value in response.headers.items():
                if any(word in key.lower() for word in ['total', 'count',␣
↪'page', 'limit']):
                    print(f"      {key}: {value}")

        return df

    except requests.exceptions.RequestException as e:
```

```
        print(f"Error fetching data: {e}")

        # Try alternative parameter combinations
        for i, alt_param in enumerate(alt_params):
            try:
                print(f"   Trying alternative parameter set {i+1}: {alt_param}")
                response = requests.get(base_url, params=alt_param, timeout=30)
                response.raise_for_status()
                data = response.json()
                df = pd.DataFrame(data)
                print(f"   Alternative method worked! Fetched {len(df)}␣
↪records")
                return df
            except:
                continue

        print(f"   All parameter combinations failed")
        return pd.DataFrame()

def fetch_all_medicare_data(batch_size=5000, target_records=14309):
    """Fetch all available Medicare Part D data using proper pagination"""

    all_data = []
    offset = 0

    print(f"Fetching Medicare Part D data (target: {target_records:,} records)..
↪.")
    print(f"   API appears to limit responses to {batch_size:,} records per␣
↪request")

    while True:
        batch = fetch_medicare_part_d_data(limit=batch_size, offset=offset)

        if batch.empty:
            print(f"   No more data returned at offset {offset:,}")
            break

        all_data.append(batch)

        # Calculate running total
        total_so_far = sum(len(df) for df in all_data)
        print(f"   Batch {len(all_data)}: {len(batch):,} records | Total:␣
↪{total_so_far:,}")

        # If we got less than batch_size, we've reached the end
        if len(batch) < batch_size:
```

```
            print(f"   Reached end of data (received {len(batch):,} <␣
↪{batch_size:,})")
            break

        # If we've reached our target, we can stop (optional safety check)
        if total_so_far >= target_records:
            print(f"   Reached target of {target_records:,} records")
            break

        # Increment offset for next batch
        offset += len(batch)

        # Add a small delay to be respectful to the API
        import time
        time.sleep(0.1)

    if all_data:
        combined_df = pd.concat(all_data, ignore_index=True)
        expected_vs_actual = f"Expected: {target_records:,} | Actual:␣
↪{len(combined_df):,}"

        if len(combined_df) == target_records:
            print(f"SUCCESS: Fetched all {len(combined_df):,} records!")
        elif len(combined_df) < target_records:
            print(f"PARTIAL: Fetched {len(combined_df):,} of {target_records:,}␣
↪records")
            print(f"   Possible reasons: API limits, data changes, or network␣
↪issues")
        else:
            print(f"UNEXPECTED: Fetched {len(combined_df):,} records (more than␣
↪expected {target_records:,})")

        print(f"   {expected_vs_actual}")
        return combined_df
    else:
        print("No data retrieved from any batch")
        return pd.DataFrame()
```

[3]:
```
# Extract Data from CMS API

print("\nSTEP 1: Extracting All Medicare Part D Data from CMS API")
df = fetch_all_medicare_data(batch_size=5000, target_records=14309)

# Display basic info about the dataset
if not df.empty:
    print(f"\nDataset Overview:")
    print(f"   Total Drug Records: {len(df):,}")
```

```
    print(f"   Columns: {list(df.columns)}")
    print(f"   Memory Usage: {df.memory_usage(deep=True).sum() / 1024**2:.2f}␣
 ↪MB")

    # Show sample data
    print(f"\nSample Data (First 3 Records):")
    print(df.head(3).to_string())
else:
    print("No data retrieved. Please check API connection.")
```

```
STEP 1: Extracting All Medicare Part D Data from CMS API
Fetching Medicare Part D data (target: 14,309 records)…
   API appears to limit responses to 5,000 records per request
Fetching Medicare Part D data (limit=5000, offset=0)…
Successfully fetched 5000 records
   Batch 1: 5,000 records | Total: 5,000
Fetching Medicare Part D data (limit=5000, offset=5000)…
Successfully fetched 5000 records
   Batch 2: 5,000 records | Total: 10,000
Fetching Medicare Part D data (limit=5000, offset=10000)…
Successfully fetched 4309 records
   Batch 3: 4,309 records | Total: 14,309
   Reached end of data (received 4,309 < 5,000)
SUCCESS: Fetched all 14,309 records!
   Expected: 14,309 | Actual: 14,309

Dataset Overview:
   Total Drug Records: 14,309
   Columns: ['Brnd_Name', 'Gnrc_Name', 'Tot_Mftr', 'Mftr_Name',
'Tot_Spndng_2019', 'Tot_Dsg_Unts_2019', 'Tot_Clms_2019', 'Tot_Benes_2019',
'Avg_Spnd_Per_Dsg_Unt_Wghtd_2019', 'Avg_Spnd_Per_Clm_2019',
'Avg_Spnd_Per_Bene_2019', 'Outlier_Flag_2019', 'Tot_Spndng_2020',
'Tot_Dsg_Unts_2020', 'Tot_Clms_2020', 'Tot_Benes_2020',
'Avg_Spnd_Per_Dsg_Unt_Wghtd_2020', 'Avg_Spnd_Per_Clm_2020',
'Avg_Spnd_Per_Bene_2020', 'Outlier_Flag_2020', 'Tot_Spndng_2021',
'Tot_Dsg_Unts_2021', 'Tot_Clms_2021', 'Tot_Benes_2021',
'Avg_Spnd_Per_Dsg_Unt_Wghtd_2021', 'Avg_Spnd_Per_Clm_2021',
'Avg_Spnd_Per_Bene_2021', 'Outlier_Flag_2021', 'Tot_Spndng_2022',
'Tot_Dsg_Unts_2022', 'Tot_Clms_2022', 'Tot_Benes_2022',
'Avg_Spnd_Per_Dsg_Unt_Wghtd_2022', 'Avg_Spnd_Per_Clm_2022',
'Avg_Spnd_Per_Bene_2022', 'Outlier_Flag_2022', 'Tot_Spndng_2023',
'Tot_Dsg_Unts_2023', 'Tot_Clms_2023', 'Tot_Benes_2023',
'Avg_Spnd_Per_Dsg_Unt_Wghtd_2023', 'Avg_Spnd_Per_Clm_2023',
'Avg_Spnd_Per_Bene_2023', 'Outlier_Flag_2023', 'Chg_Avg_Spnd_Per_Dsg_Unt_22_23',
'CAGR_Avg_Spnd_Per_Dsg_Unt_19_23']
   Memory Usage: 40.32 MB
```

Sample Data (First 3 Records):

	Brnd_Name	Gnrc_Name	Tot_Mftr	Mftr_Name	Tot_Spndng_2019	Tot_Dsg_Unts_2019	Tot_Clms_2019	Tot_Benes_2019	Avg_Spnd_Per_Dsg_Unt_Wghtd_2019	Avg_Spnd_Per_Clm_2019	Avg_Spnd_Per_Bene_2019	Outlier_Flag_2019	Tot_Spndng_2020	Tot_Dsg_Unts_2020	Tot_Clms_2020	Tot_Benes_2020	Avg_Spnd_Per_Dsg_Unt_Wghtd_2020	Avg_Spnd_Per_Clm_2020	Avg_Spnd_Per_Bene_2020	Outlier_Flag_2020	Tot_Spndng_2021	Tot_Dsg_Unts_2021	Tot_Clms_2021	Tot_Benes_2021	Avg_Spnd_Per_Dsg_Unt_Wghtd_2021	Avg_Spnd_Per_Clm_2021	Avg_Spnd_Per_Bene_2021	Outlier_Flag_2021	Tot_Spndng_2022	Tot_Dsg_Unts_2022	Tot_Clms_2022	Tot_Benes_2022	Avg_Spnd_Per_Dsg_Unt_Wghtd_2022	Avg_Spnd_Per_Clm_2022	Avg_Spnd_Per_Bene_2022	Outlier_Flag_2022	Tot_Spndng_2023	Tot_Dsg_Unts_2023	Tot_Clms_2023	Tot_Benes_2023	Avg_Spnd_Per_Dsg_Unt_Wghtd_2023	Avg_Spnd_Per_Clm_2023	Avg_Spnd_Per_Bene_2023	Outlier_Flag_2023	Chg_Avg_Spnd_Per_Dsg_Unt_22_23	CAGR_Avg_Spnd_Per_Dsg_Unt_19_23
0	1st Tier Unifine Pentips	Pen Needle, Diabetic	1	Overall	139201.68	642471	5392	1878	0.2167879244	25.816335312	74.122300319	0	118923.24	547006	4457	1595	0.217701046	26.682351357	74.560025078	0	102280.76	459384	3708	1313	0.2230012539	27.583807983	77.898522468	0	70039.61	310304	2501	1147	0.225873987	28.004642143	61.063304272	0	44355.04	195672	1613	699	0.2271618646	27.498474892	63.454992847	0	0.0057017529	0.0117543595
1	1st Tier Unifine Pentips	Pen Needle, Diabetic	1	Owen Mumford Us	139201.68	642471	5392	1878	0.2167879244	25.816335312	74.122300319	0	118923.24	547006	4457	1595	0.217701046	26.682351357	74.560025078	0	102280.76	459384	3708	1313	0.2230012539	27.583807983	77.898522468	0	70039.61	310304	2501	1147	0.225873987	28.004642143	61.063304272	0	44355.04	195672	1613	699	0.2271618646	27.498474892	63.454992847	0	0.0057017529	0.0117543595
2	1st Tier Unifine Pentips Plus	Pen Needle, Diabetic	1	Overall	343031.42	1830596	14581	5319	0.1873888035	23.525918661	64.491712728	0	210217.15	1046616	8408	3905	0.200851195	25.002039724	53.832816901	0	131927.33	566872	4564	1766	0.2328115409	28.906075811	74.704037373	0	114601.54	486206	3846	1474	0.2357078344	29.797592304	77.748670285	0	97951.18	406617	3269	1267	0.2409322287	29.963652493	77.309534333	0		

0.0221647035 0.0648484791

[4]:
```
# Data cleaning

def clean_and_classify_drug_data(df):
    """Clean Medicare Part D data and classify Generic vs Brand drugs"""

    if df.empty:
        return None, None

    # Create a copy to avoid modifying original
    clean_df = df.copy()

    print(f"Analyzing column structure...")

    # Display available columns to understand the data structure
    print(f"Available columns ({len(clean_df.columns)}):")
    for i, col in enumerate(clean_df.columns):
        print(f"   {i+1:2d}. {col}")

    # Key columns we need for analysis (adjust based on actual API response)
    key_columns = {
        'brand_name': ['Brnd_Name', 'brnd_name', 'brand_name'],
        'generic_name': ['Gnrc_Name', 'gnrc_name', 'generic_name'],
        'total_spending_2023': ['Tot_Spndng_2023', 'tot_spndng_2023',␣
↪'total_spending_2023'],
        'total_claims_2023': ['Tot_Clms_2023', 'tot_clms_2023',␣
↪'total_claims_2023'],
        'total_beneficiaries_2023': ['Tot_Benes_2023', 'tot_benes_2023',␣
↪'total_beneficiaries_2023'],
        'avg_spending_per_claim_2023': ['Avg_Spnd_Per_Clm_2023',␣
↪'avg_spnd_per_clm_2023'],
        'avg_spending_per_unit_2023': ['Avg_Spnd_Per_Dsg_Unt_Wghtd_2023',␣
↪'avg_spnd_per_dsg_unt_wghtd_2023'],
        'manufacturer': ['Mftr_Name', 'mftr_name', 'manufacturer_name']
    }

    # Map columns to standardized names
    column_mapping = {}
    for standard_name, possible_names in key_columns.items():
        for possible_name in possible_names:
            if possible_name in clean_df.columns:
                column_mapping[possible_name] = standard_name
                break

    # Apply column mapping
    clean_df = clean_df.rename(columns=column_mapping)
```

```
    print(f"\nColumn mapping applied:")
    for old_name, new_name in column_mapping.items():
        print(f"   {old_name} -> {new_name}")

    # Check which key columns we have
    available_key_columns = [col for col in key_columns.keys() if col in␣
↪clean_df.columns]
    print(f"\nAvailable key columns for analysis: {available_key_columns}")

    # Classify Generic vs Brand drugs
    # Logic: If brand_name == generic_name (ignoring case), it's likely generic
    if 'brand_name' in clean_df.columns and 'generic_name' in clean_df.columns:

        # Clean brand and generic names
        clean_df['brand_name'] = clean_df['brand_name'].astype(str).str.strip().
↪str.upper()
        clean_df['generic_name'] = clean_df['generic_name'].astype(str).str.
↪strip().str.upper()

        # Classify drug type
        def classify_drug_type(row):
            brand = str(row['brand_name']).upper().strip()
            generic = str(row['generic_name']).upper().strip()

            # If brand name equals generic name, it's likely a generic drug
            if brand == generic:
                return 'Generic'
            else:
                return 'Brand'

        clean_df['drug_type'] = clean_df.apply(classify_drug_type, axis=1)

        # Display classification results
        drug_type_counts = clean_df['drug_type'].value_counts()
        print(f"\nDrug Classification Results:")
        for drug_type, count in drug_type_counts.items():
            percentage = (count / len(clean_df)) * 100
            print(f"   {drug_type}: {count:,} drugs ({percentage:.1f}%)")

    # Convert spending columns to numeric
    numeric_columns = ['total_spending_2023', 'total_claims_2023',␣
↪'total_beneficiaries_2023',
                       'avg_spending_per_claim_2023',␣
↪'avg_spending_per_unit_2023']

    for col in numeric_columns:
```

```
        if col in clean_df.columns:
            # Remove any currency symbols and convert to numeric
            clean_df[col] = pd.to_numeric(clean_df[col].astype(str).str.
↪replace(r'[$,]', '', regex=True),
                                          errors='coerce')

    # Remove rows with missing critical data
    critical_columns = ['brand_name', 'generic_name']
    if all(col in clean_df.columns for col in critical_columns):
        initial_count = len(clean_df)
        clean_df = clean_df.dropna(subset=critical_columns)
        final_count = len(clean_df)
        print(f"\nRemoved {initial_count - final_count:,} rows with missing␣
↪critical data")
        print(f"   Final dataset: {final_count:,} drug records")

    # Create SQLite database for analysis
    print(f"\nSetting up SQLite database for SQL analysis...")
    conn = sqlite3.connect(':memory:')

    # Load data into SQLite
    clean_df.to_sql('medicare_drugs', conn, if_exists='replace', index=False)

    # Create indexes for better performance
    cursor = conn.cursor()
    try:
        cursor.execute("CREATE INDEX idx_drug_type ON␣
↪medicare_drugs(drug_type)")
        cursor.execute("CREATE INDEX idx_generic_name ON␣
↪medicare_drugs(generic_name)")
        cursor.execute("CREATE INDEX idx_brand_name ON␣
↪medicare_drugs(brand_name)")
        print(f"SQLite database created with indexes")
        print(f"   Table: medicare_drugs")
        print(f"   Records: {final_count:,}")
        print(f"   Indexes: drug_type, generic_name, brand_name")
    except Exception as e:
        print(f"Index creation warning: {e}")

    return clean_df, conn

print("\nSTEP 2: Data Cleaning")
df_clean, sql_conn = clean_and_classify_drug_data(df)
```

```
STEP 2: Data Cleaning
Analyzing column structure…
```

```
Available columns (46):
    1. Brnd_Name
    2. Gnrc_Name
    3. Tot_Mftr
    4. Mftr_Name
    5. Tot_Spndng_2019
    6. Tot_Dsg_Unts_2019
    7. Tot_Clms_2019
    8. Tot_Benes_2019
    9. Avg_Spnd_Per_Dsg_Unt_Wghtd_2019
   10. Avg_Spnd_Per_Clm_2019
   11. Avg_Spnd_Per_Bene_2019
   12. Outlier_Flag_2019
   13. Tot_Spndng_2020
   14. Tot_Dsg_Unts_2020
   15. Tot_Clms_2020
   16. Tot_Benes_2020
   17. Avg_Spnd_Per_Dsg_Unt_Wghtd_2020
   18. Avg_Spnd_Per_Clm_2020
   19. Avg_Spnd_Per_Bene_2020
   20. Outlier_Flag_2020
   21. Tot_Spndng_2021
   22. Tot_Dsg_Unts_2021
   23. Tot_Clms_2021
   24. Tot_Benes_2021
   25. Avg_Spnd_Per_Dsg_Unt_Wghtd_2021
   26. Avg_Spnd_Per_Clm_2021
   27. Avg_Spnd_Per_Bene_2021
   28. Outlier_Flag_2021
   29. Tot_Spndng_2022
   30. Tot_Dsg_Unts_2022
   31. Tot_Clms_2022
   32. Tot_Benes_2022
   33. Avg_Spnd_Per_Dsg_Unt_Wghtd_2022
   34. Avg_Spnd_Per_Clm_2022
   35. Avg_Spnd_Per_Bene_2022
   36. Outlier_Flag_2022
   37. Tot_Spndng_2023
   38. Tot_Dsg_Unts_2023
   39. Tot_Clms_2023
   40. Tot_Benes_2023
   41. Avg_Spnd_Per_Dsg_Unt_Wghtd_2023
   42. Avg_Spnd_Per_Clm_2023
   43. Avg_Spnd_Per_Bene_2023
   44. Outlier_Flag_2023
   45. Chg_Avg_Spnd_Per_Dsg_Unt_22_23
   46. CAGR_Avg_Spnd_Per_Dsg_Unt_19_23
```

```
Column mapping applied:
   Brnd_Name -> brand_name
   Gnrc_Name -> generic_name
   Tot_Spndng_2023 -> total_spending_2023
   Tot_Clms_2023 -> total_claims_2023
   Tot_Benes_2023 -> total_beneficiaries_2023
   Avg_Spnd_Per_Clm_2023 -> avg_spending_per_claim_2023
   Avg_Spnd_Per_Dsg_Unt_Wghtd_2023 -> avg_spending_per_unit_2023
   Mftr_Name -> manufacturer

Available key columns for analysis: ['brand_name', 'generic_name',
'total_spending_2023', 'total_claims_2023', 'total_beneficiaries_2023',
'avg_spending_per_claim_2023', 'avg_spending_per_unit_2023', 'manufacturer']

Drug Classification Results:
   Brand: 9,485 drugs (66.3%)
   Generic: 4,824 drugs (33.7%)

Removed 0 rows with missing critical data
   Final dataset: 14,309 drug records

Setting up SQLite database for SQL analysis…
SQLite database created with indexes
   Table: medicare_drugs
   Records: 14,309
   Indexes: drug_type, generic_name, brand_name
```

[5]:
```python
# SQL Helper Functions

def execute_sql(conn, query, description=""):
    """Execute SQL query and return results as DataFrame"""

    if description:
        print(f"\n{description}")

    print(f"SQL Query:")
    print("-" * 100)
    print(query)
    print("-" * 100)

    try:
        result = pd.read_sql_query(query, conn)
        print(f"Query executed successfully. Returned {len(result)} rows.")
        return result

    except Exception as e:
        print(f"SQL Error: {e}")
```

```
        return pd.DataFrame()

def display_sql_results(result_df, title="Query Results", show_all=True):
    """Display SQL query results in a formatted way"""

    print(f"\n{title}:")
    print("=" * 200)

    if not result_df.empty:
        if show_all or len(result_df) <= 20:
            print(result_df.to_string(index=False, float_format='%.2f'))
        else:
            print("Top 20 results:")
            print(result_df.head(20).to_string(index=False, float_format='%.
 ↪2f'))
            print(f"\n... and {len(result_df) - 20} more rows")
    else:
        print("No results returned")

    print("=" * 200)
```

[6]:
```
# SQL analysis - generic vs brand comparison

def run_generic_vs_brand_analysis(conn):
    """Run SQL queries to analyze Generic vs Brand savings opportunities"""

    if not conn:
        print("No database connection available")
        return

    print("\n" + "="*80)
    print("SQL QUERY 1: Overall Generic vs Brand Summary")
    print("="*200)

    # Basic comparison
    sql_query_1 = """
    SELECT
        drug_type,
        COUNT(*) as drug_count,
        ROUND(100.0 * COUNT(*) / (SELECT COUNT(*) FROM medicare_drugs), 2) as␣
 ↪percentage_of_drugs,
        COALESCE(SUM(total_spending_2023), 0) as total_spending,
        COALESCE(SUM(total_claims_2023), 0) as total_claims,
        COALESCE(SUM(total_beneficiaries_2023), 0) as total_beneficiaries,
        COALESCE(AVG(avg_spending_per_claim_2023), 0) as avg_spending_per_claim,
        COALESCE(AVG(avg_spending_per_unit_2023), 0) as avg_spending_per_unit
    FROM medicare_drugs
```

```
    WHERE drug_type IS NOT NULL
    GROUP BY drug_type
    ORDER BY total_spending DESC;
    """

    result1 = execute_sql(conn, sql_query_1, "Generic vs Brand overall␣
 ↪comparison")
    display_sql_results(result1, "Generic vs Brand Summary Statistics")

    return result1

if sql_conn:
    result1 = run_generic_vs_brand_analysis(sql_conn)
```

```
================================================================================
SQL QUERY 1: Overall Generic vs Brand Summary
================================================================================
================================================================================
========================================

Generic vs Brand overall comparison
SQL Query:
--------------------------------------------------------------------------------
--------------------

    SELECT
        drug_type,
        COUNT(*) as drug_count,
        ROUND(100.0 * COUNT(*) / (SELECT COUNT(*) FROM medicare_drugs), 2) as
percentage_of_drugs,
        COALESCE(SUM(total_spending_2023), 0) as total_spending,
        COALESCE(SUM(total_claims_2023), 0) as total_claims,
        COALESCE(SUM(total_beneficiaries_2023), 0) as total_beneficiaries,
        COALESCE(AVG(avg_spending_per_claim_2023), 0) as avg_spending_per_claim,
        COALESCE(AVG(avg_spending_per_unit_2023), 0) as avg_spending_per_unit
    FROM medicare_drugs
    WHERE drug_type IS NOT NULL
    GROUP BY drug_type
    ORDER BY total_spending DESC;

--------------------------------------------------------------------------------
--------------------
Query executed successfully. Returned 2 rows.

Generic vs Brand Summary Statistics:
================================================================================
================================================================================
```

```
========================================
drug_type  drug_count  percentage_of_drugs  total_spending  total_claims
total_beneficiaries  avg_spending_per_claim  avg_spending_per_unit
    Brand        9485                66.29 500182808153.60    1433665398
498048328.00                 2937.43                 426.65
  Generic        4824                33.71  51666232948.15    1801607502
568002987.00                  568.59                  29.17
================================================================================
================================================================================
========================================
```

[7]:
```
# SQL analysis - savings opportunities

def analyze_savings_opportunities(conn):
    """Analyze potential savings from generic adoption"""

    print("\n" + "="*80)
    print("SQL QUERY 2: Potential Savings Analysis by Generic Name")
    print("   (Compare Brand vs Generic prices for same drug)")
    print("="*200)

    # Savings potential analysis
    sql_query_2 = """
    WITH drug_comparison AS (
        SELECT
            generic_name,
            MAX(CASE WHEN drug_type = 'Brand' THEN avg_spending_per_claim_2023␣
 ↪END) as brand_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Generic' THEN␣
 ↪avg_spending_per_claim_2023 END) as generic_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_claims_2023 END) as␣
 ↪brand_claims,
            MAX(CASE WHEN drug_type = 'Generic' THEN total_claims_2023 END) as␣
 ↪generic_claims,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_spending_2023 END) as␣
 ↪brand_spending,
            MAX(CASE WHEN drug_type = 'Generic' THEN total_spending_2023 END)␣
 ↪as generic_spending
        FROM medicare_drugs
        WHERE drug_type IS NOT NULL
        AND avg_spending_per_claim_2023 IS NOT NULL
        AND total_claims_2023 IS NOT NULL
        GROUP BY generic_name
        HAVING COUNT(DISTINCT drug_type) = 2  -- Must have both brand and␣
 ↪generic
    ),
    savings_calculation AS (
```

```
        SELECT
            generic_name,
            brand_cost_per_claim,
            generic_cost_per_claim,
            brand_claims,
            generic_claims,
            brand_spending,
            generic_spending,
            (brand_cost_per_claim - generic_cost_per_claim) as␣
 ↪cost_difference_per_claim,
            ROUND(100.0 * (brand_cost_per_claim - generic_cost_per_claim) /␣
 ↪brand_cost_per_claim, 2) as percent_savings,
            ROUND(brand_claims * (brand_cost_per_claim -␣
 ↪generic_cost_per_claim), 0) as potential_annual_savings
        FROM drug_comparison
        WHERE brand_cost_per_claim > generic_cost_per_claim
        AND brand_cost_per_claim > 0
        AND generic_cost_per_claim > 0
    )
    SELECT
        generic_name,
        ROUND(brand_cost_per_claim, 2) as brand_cost_per_claim,
        ROUND(generic_cost_per_claim, 2) as generic_cost_per_claim,
        ROUND(cost_difference_per_claim, 2) as savings_per_claim,
        percent_savings,
        brand_claims,
        ROUND(potential_annual_savings, 0) as potential_annual_savings
    FROM savings_calculation
    ORDER BY potential_annual_savings DESC
    LIMIT 20;
    """

    result2 = execute_sql(conn, sql_query_2, "Potential savings if brand users␣
 ↪switched to generic")
    display_sql_results(result2, "Top 20 Drugs with Highest Savings Potential")

    return result2

if sql_conn:
    result2 = analyze_savings_opportunities(sql_conn)
```

```
================================================================================
SQL QUERY 2: Potential Savings Analysis by Generic Name
   (Compare Brand vs Generic prices for same drug)
================================================================================
================================================================================
```

```
======================================

Potential savings if brand users switched to generic
SQL Query:
--------------------------------------------------------------------------------
--------------------

    WITH drug_comparison AS (
        SELECT
            generic_name,
            MAX(CASE WHEN drug_type = 'Brand' THEN avg_spending_per_claim_2023
END) as brand_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Generic' THEN avg_spending_per_claim_2023
END) as generic_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_claims_2023 END) as
brand_claims,
            MAX(CASE WHEN drug_type = 'Generic' THEN total_claims_2023 END) as
generic_claims,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_spending_2023 END) as
brand_spending,
            MAX(CASE WHEN drug_type = 'Generic' THEN total_spending_2023 END) as
generic_spending
        FROM medicare_drugs
        WHERE drug_type IS NOT NULL
        AND avg_spending_per_claim_2023 IS NOT NULL
        AND total_claims_2023 IS NOT NULL
        GROUP BY generic_name
        HAVING COUNT(DISTINCT drug_type) = 2  -- Must have both brand and
generic
    ),
    savings_calculation AS (
        SELECT
            generic_name,
            brand_cost_per_claim,
            generic_cost_per_claim,
            brand_claims,
            generic_claims,
            brand_spending,
            generic_spending,
            (brand_cost_per_claim - generic_cost_per_claim) as
cost_difference_per_claim,
            ROUND(100.0 * (brand_cost_per_claim - generic_cost_per_claim) /
brand_cost_per_claim, 2) as percent_savings,
            ROUND(brand_claims * (brand_cost_per_claim -
generic_cost_per_claim), 0) as potential_annual_savings
        FROM drug_comparison
        WHERE brand_cost_per_claim > generic_cost_per_claim
        AND brand_cost_per_claim > 0
```

```
        AND generic_cost_per_claim > 0
    )
    SELECT
        generic_name,
        ROUND(brand_cost_per_claim, 2) as brand_cost_per_claim,
        ROUND(generic_cost_per_claim, 2) as generic_cost_per_claim,
        ROUND(cost_difference_per_claim, 2) as savings_per_claim,
        percent_savings,
        brand_claims,
        ROUND(potential_annual_savings, 0) as potential_annual_savings
    FROM savings_calculation
    ORDER BY potential_annual_savings DESC
    LIMIT 20;

--------------------------------------------------------------------------------
--------------------
Query executed successfully. Returned 20 rows.

Top 20 Drugs with Highest Savings Potential:
================================================================================
================================================================================
========================================
        generic_name  brand_cost_per_claim  generic_cost_per_claim
savings_per_claim  percent_savings  brand_claims  potential_annual_savings
       METFORMIN HCL               8035.09                2668.56
5366.53            66.79       9101305            48842469096.00
       BUPROPION HCL               4202.69                  41.62
4161.06            99.01       6162383            25642069045.00
     VENLAFAXINE HCL               1372.61                  35.44
1337.17            97.42       5232598             6996895148.00
          NIFEDIPINE                724.99                  81.80
643.19             88.72       3524646            2267027764.00
      INSULIN LISPRO               1558.85                 107.01
1451.85            93.14       1363952             1980247840.00
 OXYBUTYNIN CHLORIDE                733.05                 267.69
465.36             63.48       3360126            1563657749.00
      INSULIN ASPART               1007.96                 286.08
721.88             71.62       2118734            1529480105.00
    INSULIN DEGLUDEC               1238.37                 157.08
1081.29            87.32       1000189             1081492812.00
        ARIPIPRAZOLE               5484.99                 833.98
4651.00            84.80        232162             1079786236.00
   DIVALPROEX SODIUM                759.41                 233.40
526.02             69.27       1613313             848629712.00
       OXYCODONE HCL               1590.49                 248.51
1341.98            84.38        535789              719019718.00
 QUETIAPINE FUMARATE               1150.30                  31.39
1118.91            97.27        494442              553233758.00
```

```
   LITHIUM CARBONATE               1311.97                   13.95
1298.02            98.94        410632              533008628.00
TOLTERODINE TARTRATE                844.34                  107.48
736.86             87.27        621750              458142592.00
         LAMOTRIGINE                3288.96                   93.03
3195.94            97.17        140986              450582223.00
        LENALIDOMIDE               17852.36                16041.79
1810.56            10.14        216207              391456764.00
      LURASIDONE HCL                1596.75                  561.03
1035.73            64.86        285089              295273984.00
  TIOTROPIUM BROMIDE                 826.94                  636.34
190.60             23.05       1313138              250280371.00
       CARBAMAZEPINE                 591.14                  111.90
479.24             81.07        499929              239584276.00
         RISPERIDONE                3129.06                   88.06
3041.01            97.19         63970              194533102.00
================================================================================
================================================================================
========================================
```

[8]:
```
# SQL analysis - high impact opportunities

def analyze_high_impact_opportunities(conn):
    """Focus on drugs with high volume and high cost difference"""

    print("\n" + "="*80)
    print("SQL QUERY 3: High-Impact Savings Opportunities")
    print("   (Focus on drugs with high volume and high cost difference)")
    print("="*200)

    # High-impact opportunities
    sql_query_3 = """
    WITH drug_comparison AS (
        SELECT
            generic_name,
            MAX(CASE WHEN drug_type = 'Brand' THEN avg_spending_per_claim_2023␣
 ↪END) as brand_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Generic' THEN␣
 ↪avg_spending_per_claim_2023 END) as generic_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_claims_2023 END) as␣
 ↪brand_claims,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_beneficiaries_2023␣
 ↪END) as brand_beneficiaries,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_spending_2023 END) as␣
 ↪brand_total_spending
        FROM medicare_drugs
        WHERE drug_type IS NOT NULL
```

```
        AND avg_spending_per_claim_2023 IS NOT NULL
        AND total_claims_2023 IS NOT NULL
        GROUP BY generic_name
        HAVING COUNT(DISTINCT drug_type) = 2
    )
    SELECT
        generic_name,
        ROUND(brand_cost_per_claim, 2) as brand_cost_per_claim,
        ROUND(generic_cost_per_claim, 2) as generic_cost_per_claim,
        brand_claims,
        brand_beneficiaries,
        ROUND(brand_total_spending, 0) as current_brand_spending,
        ROUND((brand_cost_per_claim - generic_cost_per_claim), 2) as␣
↪savings_per_claim,
        ROUND(100.0 * (brand_cost_per_claim - generic_cost_per_claim) /␣
↪brand_cost_per_claim, 1) as percent_savings,
        ROUND(brand_claims * (brand_cost_per_claim - generic_cost_per_claim),␣
↪0) as total_potential_savings
    FROM drug_comparison
    WHERE brand_cost_per_claim > generic_cost_per_claim
    AND brand_claims > 10000  -- Focus on high-volume drugs
    AND (brand_cost_per_claim - generic_cost_per_claim) > 50  -- Significant␣
↪cost difference
    ORDER BY total_potential_savings DESC
    LIMIT 15;
    """

    result3 = execute_sql(conn, sql_query_3, "High-impact savings␣
↪opportunities")
    display_sql_results(result3, "High-Impact Generic Adoption Opportunities")

    return result3

if sql_conn:
    result3 = analyze_high_impact_opportunities(sql_conn)
```

```
================================================================================
SQL QUERY 3: High-Impact Savings Opportunities
   (Focus on drugs with high volume and high cost difference)
================================================================================
================================================================================
========================================

High-impact savings opportunities
SQL Query:
--------------------------------------------------------------------------------
```

```
--------------------

    WITH drug_comparison AS (
        SELECT
            generic_name,
            MAX(CASE WHEN drug_type = 'Brand' THEN avg_spending_per_claim_2023
END) as brand_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Generic' THEN avg_spending_per_claim_2023
END) as generic_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_claims_2023 END) as
brand_claims,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_beneficiaries_2023 END)
as brand_beneficiaries,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_spending_2023 END) as
brand_total_spending
        FROM medicare_drugs
        WHERE drug_type IS NOT NULL
        AND avg_spending_per_claim_2023 IS NOT NULL
        AND total_claims_2023 IS NOT NULL
        GROUP BY generic_name
        HAVING COUNT(DISTINCT drug_type) = 2
    )
    SELECT
        generic_name,
        ROUND(brand_cost_per_claim, 2) as brand_cost_per_claim,
        ROUND(generic_cost_per_claim, 2) as generic_cost_per_claim,
        brand_claims,
        brand_beneficiaries,
        ROUND(brand_total_spending, 0) as current_brand_spending,
        ROUND((brand_cost_per_claim - generic_cost_per_claim), 2) as
savings_per_claim,
        ROUND(100.0 * (brand_cost_per_claim - generic_cost_per_claim) /
brand_cost_per_claim, 1) as percent_savings,
        ROUND(brand_claims * (brand_cost_per_claim - generic_cost_per_claim), 0)
as total_potential_savings
    FROM drug_comparison
    WHERE brand_cost_per_claim > generic_cost_per_claim
    AND brand_claims > 10000  -- Focus on high-volume drugs
    AND (brand_cost_per_claim - generic_cost_per_claim) > 50  -- Significant
cost difference
    ORDER BY total_potential_savings DESC
    LIMIT 15;

--------------------------------------------------------------------------------
--------------------
Query executed successfully. Returned 15 rows.

High-Impact Generic Adoption Opportunities:
```

```
================================================================================
================================================================================
=======================================
        generic_name  brand_cost_per_claim  generic_cost_per_claim  brand_claims
brand_beneficiaries  current_brand_spending  savings_per_claim  percent_savings
total_potential_savings
       METFORMIN HCL               8035.09                2668.56       9101305
2498132.00            191118411.00           5366.53            66.80
48842469096.00
       BUPROPION HCL               4202.69                  41.62       6162383
1325901.00            197171943.00           4161.06            99.00
25642069045.00
     VENLAFAXINE HCL               1372.61                  35.44       5232598
954551.00            151208248.00           1337.17            97.40
6996895148.00
          NIFEDIPINE                724.99                  81.80       3524646
849024.00            146639465.00            643.19            88.70
2267027764.00
      INSULIN LISPRO               1558.85                 107.01       1363952
418431.00           1208385334.00           1451.85            93.10
1980247840.00
 OXYBUTYNIN CHLORIDE                733.05                 267.69       3360126
881680.00            113070736.00            465.36            63.50
1563657749.00
      INSULIN ASPART               1007.96                 286.08       2118734
588526.00           1875605627.00            721.88            71.60
1529480105.00
    INSULIN DEGLUDEC               1238.37                 157.08       1000189
258451.00           1047670731.00           1081.29            87.30
1081492812.00
       ARIPIPRAZOLE               5484.99                 833.98        232162
29378.00            636857891.00           4651.00            84.80
1079786236.00
   DIVALPROEX SODIUM                759.41                 233.40       1613313
232066.00            120375972.00            526.02            69.30
848629712.00
       OXYCODONE HCL               1590.49                 248.51        535789
65061.00            390573654.00           1341.98            84.40
719019718.00
 QUETIAPINE FUMARATE               1150.30                  31.39        494442
76394.00             50996191.00           1118.91            97.30
553233758.00
   LITHIUM CARBONATE               1311.97                  13.95        410632
61548.00              7483309.00           1298.02            98.90
533008628.00
TOLTERODINE TARTRATE                844.34                 107.48        621750
160195.00             91643656.00            736.86            87.30
458142592.00
```

```
         LAMOTRIGINE                3288.96                    93.03        140986
23679.00            111836411.00            3195.94             97.20
450582223.00
================================================================================
================================================================================
========================================
```

[9]:
```python
# SQL analysis - total savings calculation

def calculate_total_savings(conn):
    """Calculate total Medicare savings potential"""

    print("\n" + "="*100)
    print("SQL QUERY 4: Total Medicare Savings Potential")
    print("="*100)

    # Total savings calculation
    sql_query_4 = """
    WITH drug_comparison AS (
        SELECT
            generic_name,
            MAX(CASE WHEN drug_type = 'Brand' THEN avg_spending_per_claim_2023␣
 ↪END) as brand_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Generic' THEN␣
 ↪avg_spending_per_claim_2023 END) as generic_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_claims_2023 END) as␣
 ↪brand_claims,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_spending_2023 END) as␣
 ↪brand_spending
        FROM medicare_drugs
        WHERE drug_type IS NOT NULL
        AND avg_spending_per_claim_2023 IS NOT NULL
        AND total_claims_2023 IS NOT NULL
        GROUP BY generic_name
        HAVING COUNT(DISTINCT drug_type) = 2
    )
    SELECT
        COUNT(*) as drugs_with_savings_potential,
        SUM(brand_claims) as total_brand_claims,
        ROUND(SUM(brand_spending), 0) as total_current_brand_spending,
        ROUND(SUM(brand_claims * generic_cost_per_claim), 0) as␣
 ↪cost_if_all_generic,
        ROUND(SUM(brand_claims * (brand_cost_per_claim -␣
 ↪generic_cost_per_claim)), 0) as total_potential_savings,
        ROUND(100.0 * SUM(brand_claims * (brand_cost_per_claim -␣
 ↪generic_cost_per_claim)) / SUM(brand_spending), 2) as␣
 ↪percent_savings_potential
```

```
    FROM drug_comparison
    WHERE brand_cost_per_claim > generic_cost_per_claim
    AND brand_cost_per_claim > 0
    AND generic_cost_per_claim > 0;
    """

    result4 = execute_sql(conn, sql_query_4, "Total Medicare savings potential␣
 ↪calculation")
    display_sql_results(result4, "TOTAL MEDICARE SAVINGS POTENTIAL")

    return result4

if sql_conn:
    result4 = calculate_total_savings(sql_conn)
```

```
================================================================================
====================
SQL QUERY 4: Total Medicare Savings Potential
================================================================================
====================

Total Medicare savings potential calculation
SQL Query:
--------------------------------------------------------------------------------
--------------------

    WITH drug_comparison AS (
        SELECT
            generic_name,
            MAX(CASE WHEN drug_type = 'Brand' THEN avg_spending_per_claim_2023
END) as brand_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Generic' THEN avg_spending_per_claim_2023
END) as generic_cost_per_claim,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_claims_2023 END) as
brand_claims,
            MAX(CASE WHEN drug_type = 'Brand' THEN total_spending_2023 END) as
brand_spending
        FROM medicare_drugs
        WHERE drug_type IS NOT NULL
        AND avg_spending_per_claim_2023 IS NOT NULL
        AND total_claims_2023 IS NOT NULL
        GROUP BY generic_name
        HAVING COUNT(DISTINCT drug_type) = 2
    )
    SELECT
        COUNT(*) as drugs_with_savings_potential,
        SUM(brand_claims) as total_brand_claims,
```

```
        ROUND(SUM(brand_spending), 0) as total_current_brand_spending,
        ROUND(SUM(brand_claims * generic_cost_per_claim), 0) as
cost_if_all_generic,
        ROUND(SUM(brand_claims * (brand_cost_per_claim -
generic_cost_per_claim)), 0) as total_potential_savings,
        ROUND(100.0 * SUM(brand_claims * (brand_cost_per_claim -
generic_cost_per_claim)) / SUM(brand_spending), 2) as percent_savings_potential
    FROM drug_comparison
    WHERE brand_cost_per_claim > generic_cost_per_claim
    AND brand_cost_per_claim > 0
    AND generic_cost_per_claim > 0;

--------------------------------------------------------------------------------
--------------------
Query executed successfully. Returned 1 rows.

TOTAL MEDICARE SAVINGS POTENTIAL:
================================================================================
================================================================================
========================================
 drugs_with_savings_potential  total_brand_claims  total_current_brand_spending
cost_if_all_generic  total_potential_savings  percent_savings_potential
                         322            46115718                17175111432.00
35316483211.00           99868022524.00                     581.47
================================================================================
================================================================================
========================================
```

[10]:
```
# SQL analysis - manufacturer analysis

def analyze_manufacturers(conn):
    """Analyze manufacturers by drug type"""

    print("\n" + "="*80)
    print("SQL QUERY 5: Manufacturer Analysis")
    print("   (Which manufacturers have the largest brand vs generic price␣
 ↪gaps)")
    print("="*80)

    # Manufacturer analysis
    sql_query_5 = """
    SELECT
        manufacturer,
        drug_type,
        COUNT(*) as drug_count,
        ROUND(AVG(avg_spending_per_claim_2023), 2) as avg_cost_per_claim,
        ROUND(SUM(total_spending_2023), 0) as total_spending,
```

```
        ROUND(SUM(total_claims_2023), 0) as total_claims
    FROM medicare_drugs
    WHERE manufacturer IS NOT NULL
    AND drug_type IS NOT NULL
    AND avg_spending_per_claim_2023 IS NOT NULL
    AND total_spending_2023 IS NOT NULL
    GROUP BY manufacturer, drug_type
    HAVING COUNT(*) >= 3  -- Only manufacturers with multiple drugs
    ORDER BY total_spending DESC
    LIMIT 20;
    """

    result5 = execute_sql(conn, sql_query_5, "Manufacturer analysis by drug␣
 ↪type")
    display_sql_results(result5, "Manufacturer Analysis: Brand vs Generic")

    return result5

if sql_conn:
    result5 = analyze_manufacturers(sql_conn)
```

```
================================================================================
SQL QUERY 5: Manufacturer Analysis
   (Which manufacturers have the largest brand vs generic price gaps)
================================================================================

Manufacturer analysis by drug type
SQL Query:
--------------------------------------------------------------------------------
--------------------

    SELECT
        manufacturer,
        drug_type,
        COUNT(*) as drug_count,
        ROUND(AVG(avg_spending_per_claim_2023), 2) as avg_cost_per_claim,
        ROUND(SUM(total_spending_2023), 0) as total_spending,
        ROUND(SUM(total_claims_2023), 0) as total_claims
    FROM medicare_drugs
    WHERE manufacturer IS NOT NULL
    AND drug_type IS NOT NULL
    AND avg_spending_per_claim_2023 IS NOT NULL
    AND total_spending_2023 IS NOT NULL
    GROUP BY manufacturer, drug_type
    HAVING COUNT(*) >= 3  -- Only manufacturers with multiple drugs
    ORDER BY total_spending DESC
    LIMIT 20;
```

```
--------------------------------------------------------------------------------
--------------------
Query executed successfully. Returned 20 rows.

Manufacturer Analysis: Brand vs Generic:
================================================================================
================================================================================
========================================
   manufacturer drug_type  drug_count  avg_cost_per_claim  total_spending
total_claims
        Overall     Brand        2997             4474.07 250091404077.00
716832699.00
        Overall   Generic         601              925.13  25833116474.00
900803751.00
BMS Primarycare     Brand          18             6774.71  19594167559.00
21372231.00
   Novo Nordisk     Brand          43              841.86  18927065140.00
17220629.00
Eli Lilly & Co.     Brand          54             3363.65  15886397838.00
13764072.00
Boehringer Ing.     Brand          27             4143.90  15717354549.00
14916961.00
Glaxosmithkline     Brand          45             2072.99  12554619721.00
27841963.00
    Astrazeneca     Brand          35             5161.05  12239890905.00
11465462.00
  Abbvie US LLC     Brand          34             6768.00  11540973575.00
6537037.00
 Janssen Pharm.     Brand          20             3379.78   9551407797.00
7977171.00
 Sanofi-Aventis     Brand          30             9018.68   7885001830.00
8364594.00
       Novartis     Brand          55             9518.43   7747270369.00
3768124.00
Merck Sharp & D     Brand          37             2889.45   6156624699.00
6203331.00
Pfizer US Pharm     Brand          64             5395.02   6150857851.00
2796037.00
Gilead Sciences     Brand          22             9604.54   6121792400.00
1441619.00
  Allergan Inc.     Brand          36              907.22   6044801891.00
7544352.00
    Celgene/BMS     Brand          10            18572.08   5915783717.00
314602.00
          Amgen     Brand          31             5530.64   5458827516.00
1489733.00
Astellas Pharma     Brand           9             4911.16   5213902466.00
```

```
3861722.00
Viiv Healthcare     Brand          16             2785.56    2394006655.00
720316.00
================================================================================
================================================================================
======================================
```

[11]:
```python
# Executive summary generation

def generate_savings_executive_summary(conn):
    """Generate executive summary of Generic vs Brand savings analysis"""

    print("\n\nEXECUTIVE SUMMARY: GENERIC vs BRAND SAVINGS OPPORTUNITY")
    print("=" * 70)

    if not conn:
        print("Cannot generate summary - no database connection")
        return

    # Get summary statistics
    summary_sql = """
    SELECT
        drug_type,
        COUNT(*) as drug_count,
        COALESCE(SUM(total_spending_2023), 0) as total_spending,
        COALESCE(SUM(total_claims_2023), 0) as total_claims,
        COALESCE(AVG(avg_spending_per_claim_2023), 0) as avg_cost_per_claim
    FROM medicare_drugs
    WHERE drug_type IS NOT NULL
    GROUP BY drug_type;
    """

    summary_result = pd.read_sql_query(summary_sql, conn)

    if not summary_result.empty:
        print(f"\nKEY FINDINGS:")

        for _, row in summary_result.iterrows():
            drug_type = row['drug_type']
            drug_count = int(row['drug_count'])
            total_spending = row['total_spending']
            total_claims = int(row['total_claims'])
            avg_cost = row['avg_cost_per_claim']

            print(f"   {drug_type} Drugs:")
            print(f"     - Count: {drug_count:,} drugs")
            print(f"     - Total Spending: ${total_spending:,.0f}")
```

```
            print(f"     - Total Claims: {total_claims:,}")
            print(f"     - Avg Cost per Claim: ${avg_cost:.2f}")

        # Calculate savings potential
        savings_sql = """
        WITH drug_comparison AS (
            SELECT
                generic_name,
                MAX(CASE WHEN drug_type = 'Brand' THEN␣
↪avg_spending_per_claim_2023 END) as brand_cost,
                MAX(CASE WHEN drug_type = 'Generic' THEN␣
↪avg_spending_per_claim_2023 END) as generic_cost,
                MAX(CASE WHEN drug_type = 'Brand' THEN total_claims_2023 END)␣
↪as brand_claims
            FROM medicare_drugs
            WHERE drug_type IS NOT NULL
            AND avg_spending_per_claim_2023 IS NOT NULL
            GROUP BY generic_name
            HAVING COUNT(DISTINCT drug_type) = 2
        )
        SELECT
            COUNT(*) as switchable_drugs,
            ROUND(SUM(brand_claims * (brand_cost - generic_cost)), 0) as␣
↪total_potential_savings,
            ROUND(AVG(100.0 * (brand_cost - generic_cost) / brand_cost), 1) as␣
↪avg_percent_savings
        FROM drug_comparison
        WHERE brand_cost > generic_cost;
        """

        savings_result = pd.read_sql_query(savings_sql, conn)

        if not savings_result.empty and len(savings_result) > 0:
            savings_data = savings_result.iloc[0]
            switchable_drugs = int(savings_data['switchable_drugs'])
            total_savings = savings_data['total_potential_savings']
            avg_savings_pct = savings_data['avg_percent_savings']

            print(f"\nSAVINGS OPPORTUNITY:")
            print(f"   Drugs with Both Brand & Generic Available:␣
↪{switchable_drugs}")
            print(f"   Potential Annual Savings: ${total_savings:,.0f}")
            print(f"   Average Savings per Drug: {avg_savings_pct}%")

            print(f"\nRECOMMENDATIONS:")
```

```
            print(f"   1. Prioritize generic adoption for high-volume,␣
 ↪high-cost drugs")
            print(f"   2. Implement generic substitution policies")
            print(f"   3. Educate prescribers about cost-effective␣
 ↪alternatives")
            print(f"   4. Monitor brand vs generic pricing trends")
            print(f"   5. Focus on drugs with >50% potential savings")

    print(f"\n" + "=" * 40)
    print(f"Report Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"Analysis Method: SQL queries on live CMS Medicare Part D data")
    print(f"Data Source: CMS Medicare Part D Spending by Drug API")
    print("=" * 75)

if sql_conn:
    generate_savings_executive_summary(sql_conn)
```

```
EXECUTIVE SUMMARY: GENERIC vs BRAND SAVINGS OPPORTUNITY
======================================================================

KEY FINDINGS:
   Brand Drugs:
     - Count: 9,485 drugs
     - Total Spending: $500,182,808,154
     - Total Claims: 1,433,665,398
     - Avg Cost per Claim: $2937.43
   Generic Drugs:
     - Count: 4,824 drugs
     - Total Spending: $51,666,232,948
     - Total Claims: 1,801,607,502
     - Avg Cost per Claim: $568.59

SAVINGS OPPORTUNITY:
   Drugs with Both Brand & Generic Available: 322
   Potential Annual Savings: $99,868,022,524
   Average Savings per Drug: 66.3%

RECOMMENDATIONS:
   1. Prioritize generic adoption for high-volume, high-cost drugs
   2. Implement generic substitution policies
   3. Educate prescribers about cost-effective alternatives
   4. Monitor brand vs generic pricing trends
   5. Focus on drugs with >50% potential savings

========================================
```

```
Report Generated: 2025-08-16 14:05:44
Analysis Method: SQL queries on live CMS Medicare Part D data
Data Source: CMS Medicare Part D Spending by Drug API
================================================================================
```

[12]:
```python
# Export data

def export_results(df_clean, sql_conn):
    """Export analysis results and clean up"""

    print("\nSAVING ANALYSIS RESULTS")
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

    try:
        # Save full dataset
        filename = f"medicare_generic_vs_brand_analysis_{timestamp}.csv"
        df_clean.to_csv(filename, index=False)
        print(f"Analysis dataset saved: {filename}")

        # Save high-impact opportunities
        high_impact_sql = """
        WITH drug_comparison AS (
            SELECT
                generic_name,
                MAX(CASE WHEN drug_type = 'Brand' THEN␣
↪avg_spending_per_claim_2023 END) as brand_cost_per_claim,
                MAX(CASE WHEN drug_type = 'Generic' THEN␣
↪avg_spending_per_claim_2023 END) as generic_cost_per_claim,
                MAX(CASE WHEN drug_type = 'Brand' THEN total_claims_2023 END)␣
↪as brand_claims
            FROM medicare_drugs
            WHERE drug_type IS NOT NULL
            GROUP BY generic_name
            HAVING COUNT(DISTINCT drug_type) = 2
        )
        SELECT
            generic_name,
            brand_cost_per_claim,
            generic_cost_per_claim,
            brand_claims,
            (brand_cost_per_claim - generic_cost_per_claim) as␣
↪savings_per_claim,
            ROUND(brand_claims * (brand_cost_per_claim -␣
↪generic_cost_per_claim), 0) as total_potential_savings
        FROM drug_comparison
        WHERE brand_cost_per_claim > generic_cost_per_claim
        AND brand_claims > 5000
```

```
        ORDER BY total_potential_savings DESC;
        """

        high_impact_df = pd.read_sql_query(high_impact_sql, sql_conn)
        if not high_impact_df.empty:
            savings_filename = f"high_impact_generic_opportunities_{timestamp}.
↪csv"
            high_impact_df.to_csv(savings_filename, index=False)
            print(f"High-impact opportunities saved: {savings_filename}")

    except Exception as e:
        print(f"Error saving results: {e}")

    # Close database connection
    if sql_conn:
        sql_conn.close()
        print(f"Database connection closed")

print("\nGENERIC vs BRAND SAVINGS ANALYSIS COMPLETE!")
print("=" * 50)
print("SQL Queries Executed:")
print("1. Generic vs Brand overall comparison")
print("2. Potential savings analysis by generic name")
print("3. High-impact savings opportunities")
print("4. Total Medicare savings potential")
print("5. Manufacturer analysis by drug type")
print("=" * 50)

if sql_conn and df_clean is not None:
    export_results(df_clean, sql_conn)
```

```
GENERIC vs BRAND SAVINGS ANALYSIS COMPLETE!
==================================================
SQL Queries Executed:
1. Generic vs Brand overall comparison
2. Potential savings analysis by generic name
3. High-impact savings opportunities
4. Total Medicare savings potential
5. Manufacturer analysis by drug type
==================================================

SAVING ANALYSIS RESULTS
Analysis dataset saved: medicare_generic_vs_brand_analysis_20250816_140544.csv
High-impact opportunities saved:
high_impact_generic_opportunities_20250816_140544.csv
Database connection closed
```