Libraries

```
pip install pandas numpy matplotlib seaborn plotly openpyxl
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages (5.24.1)
Requirement already satisfied: openpyxl in /usr/local/lib/python3.12/dist-packages (3.1.5)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.59.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from plotly) (8.5.0)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.12/dist-packages (from openpyxl) (2.0.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
pip install ydata-profiling
```

```
Collecting ydata-profiling
  Downloading ydata_profiling-4.16.1-py2.py3-none-any.whl.metadata (22 kB)
Collecting scipy<1.16,>=1.4.1 (from ydata-profiling)
  Downloading scipy-1.15.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 2.7 MB/s eta 0:00:00
Requirement already satisfied: pandas!=1.4.0,<3.0,>1.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.2.2)
Requirement already satisfied: matplotlib<=3.10,>=3.5 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.10.0)
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.11.7)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (6.0.2)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (3.1.6)
Collecting visions<0.8.2,>=0.7.5 (from visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling)
  Downloading visions-0.8.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: numpy<2.2,>=1.16.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.0.2)
Collecting htmlmin==0.1.12 (from ydata-profiling)
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Collecting phik<0.13,>=0.11.1 (from ydata-profiling)
  Downloading phik-0.12.5-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (5.6 kB)
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (2.32.4)
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.67.1)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.13.2)
Collecting multimethod<2,>=1.4 (from ydata-profiling)
  Downloading multimethod-1.12-py3-none-any.whl.metadata (9.6 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.14.5)
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.4.4)
Collecting imagehash==4.3.1 (from ydata-profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: wordcloud>=1.9.3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.9.4)
Collecting dacite>=1.8 (from ydata-profiling)
  Downloading dacite-1.9.2-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: numba<=0.61,>=0.56.0 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.60.0)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.12/dist-packages (from imagehash==4.3.1->ydata-profiling) (1.9.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (from imagehash==4.3.1->ydata-profiling) (11.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling) (3.0.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (4.59.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (25.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata-profiling) (2.9.0.post0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba<=0.61,>=0.56.0->ydata-profiling) (0.43.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas!=1.4.0,<3.0,>1.1->ydata-profiling) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas!=1.4.0,<3.0,>1.1->ydata-profiling) (2025.2)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.12/dist-packages (from phik<0.13,>=0.11.1->ydata-profiling) (1.5.1)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2->ydata-profiling) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2->ydata-profiling) (2.33.2)
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2->ydata-profiling) (4.15.0)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2->ydata-profiling) (0.4.1)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.24.0->ydata-profiling) (2025.8.3)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.12/dist-packages (from statsmodels<1,>=0.13.2->ydata-profiling) (1.0.1)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling) (25.3.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling) (3.5)
Collecting puremagic (from visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling)
  Downloading puremagic-1.30-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib<=3.10,>=3.5->ydata-profiling) (1.17.0)
Downloading ydata_profiling-4.16.1-py2.py3-none-any.whl (400 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.1/400.1 kB 10.9 MB/s eta 0:00:00
Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 22.1 MB/s eta 0:00:00
Downloading dacite-1.9.2-py3-none-any.whl (16 kB)
Downloading multimethod-1.12-py3-none-any.whl (10 kB)
Downloading phik-0.12.5-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (679 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 679.7/679.7 kB 26.2 MB/s eta 0:00:00
Downloading scipy-1.15.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (37.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.3/37.3 MB 36.8 MB/s eta 0:00:00
Downloading visions-0.8.1-py3-none-any.whl (105 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 105.4/105.4 kB 7.3 MB/s eta 0:00:00
Downloading puremagic-1.30-py3-none-any.whl (43 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.3/43.3 kB 2.9 MB/s eta 0:00:00
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) ... done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=27081 sha256=267a40758d7a00822f40681d1f35c28b2531d1d2264b39606b608f86d14c3ef7
  Stored in directory: /root/.cache/pip/wheels/5f/d4/d7/4189b07b5902ee9f3ce0dbb14909fbe8037c39d6c63ffd49c9
Successfully built htmlmin
Installing collected packages: puremagic, htmlmin, scipy, multimethod, dacite, imagehash, visions, phik, ydata-profiling
  Attempting uninstall: scipy
    Found existing installation: scipy 1.16.1
    Uninstalling scipy-1.16.1:
      Successfully uninstalled scipy-1.16.1
Successfully installed dacite-1.9.2 htmlmin-0.1.12 imagehash-4.3.1 multimethod-1.12 phik-0.12.5 puremagic-1.30 scipy-1.15.3 visions-0.8.1 ydata-profiling-4.16.1
WARNING: The following packages were previously imported in this runtime:
  [scipy]
You must restart the runtime in order to use newly installed versions.
```

You must restart the runtime in order to use newly installed versions.

RESTART SESSION

Inventory

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')

# Set style for better visualizations
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")

def load_and_clean_data(file_path):
    """Load and clean the inventory data"""
    try:
        # Read the Excel file
        df = pd.read_excel("/content/drive/MyDrive/CAPSTONE/Inventory.xlsx")

        # Convert date columns to datetime
        date_columns = ['last_order_date', 'expiry_date']
        for col in date_columns:
            if col in df.columns:
                df[col] = pd.to_datetime(df[col], errors='coerce')

        # Clean column names (remove extra spaces)
        df.columns = df.columns.str.strip()

        return df
    except Exception as e:
        print(f"Error loading data: {e}")
        return None

def basic_info_analysis(df):
    """Perform basic information analysis"""
    print("="*60)
    print("BASIC DATA INFORMATION")
    print("="*60)

    print(f"Dataset Shape: {df.shape}")
    print(f"Total Records: {df.shape[0]}")
    print(f"Total Features: {df.shape[1]}")

    print("\nColumn Names and Data Types:")
    print("-" * 40)
    for col in df.columns:
        print(f"{col}: {df[col].dtype}")

    print("\nMissing Values:")
    print("-" * 20)
    missing_data = df.isnull().sum()
    missing_percentage = (missing_data / len(df)) * 100
    missing_df = pd.DataFrame({
        'Missing Count': missing_data,
        'Percentage': missing_percentage.round(2)
    })
    print(missing_df[missing_df['Missing Count'] > 0])

    print("\nBasic Statistics for Numerical Columns:")
    print("-" * 45)
    numerical_cols = df.select_dtypes(include=[np.number]).columns
    print(df[numerical_cols].describe().round(2))

def stock_status_analysis(df):
    """Analyze stock status distribution"""
    print("\n" + "="*60)
    print("STOCK STATUS ANALYSIS")
    print("="*60)

    if 'stock_status' in df.columns:
        # Stock status distribution
```

```
        status_counts = df['stock_status'].value_counts()
        status_percentage = df['stock_status'].value_counts(normalize=True) * 100

        print("Stock Status Distribution:")
        print("-" * 30)
        for status, count in status_counts.items():
            percentage = status_percentage[status]
            print(f"{status}: {count} ({percentage:.1f}%)")

        # Visualize stock status
        plt.figure(figsize=(12, 5))

        plt.subplot(1, 2, 1)
        status_counts.plot(kind='bar', color='skyblue', edgecolor='black')
        plt.title('Stock Status Distribution')
        plt.xlabel('Stock Status')
        plt.ylabel('Count')
        plt.xticks(rotation=45)

        plt.subplot(1, 2, 2)
        plt.pie(status_counts.values, labels=status_counts.index, autopct='%1.1f%%', startangle=90)
        plt.title('Stock Status Percentage')

        plt.tight_layout()
        plt.show()

    def inventory_value_analysis(df):
        """Analyze inventory values"""
        print("\n" + "="*60)
        print("INVENTORY VALUE ANALYSIS")
        print("="*60)

        if 'stock_value' in df.columns:
            total_value = df['stock_value'].sum()
            avg_value = df['stock_value'].mean()
            median_value = df['stock_value'].median()

            print(f"Total Inventory Value: ${total_value:,.2f}")
            print(f"Average Item Value: ${avg_value:,.2f}")
            print(f"Median Item Value: ${median_value:,.2f}")

            # Value by stock status
            if 'stock_status' in df.columns:
                print("\nValue Distribution by Stock Status:")
                print("-" * 40)
                value_by_status = df.groupby('stock_status')['stock_value'].agg(['sum', 'mean', 'count'])
                print(value_by_status.round(2))

            # Visualize inventory values
            plt.figure(figsize=(15, 5))

            plt.subplot(1, 3, 1)
            plt.hist(df['stock_value'], bins=30, color='lightgreen', edgecolor='black', alpha=0.7)
            plt.title('Distribution of Stock Values')
            plt.xlabel('Stock Value ($)')
            plt.ylabel('Frequency')

            plt.subplot(1, 3, 2)
            df.boxplot(column='stock_value', ax=plt.gca())
            plt.title('Stock Value Box Plot')
            plt.ylabel('Stock Value ($)')

            if 'stock_status' in df.columns:
                plt.subplot(1, 3, 3)
                df.groupby('stock_status')['stock_value'].sum().plot(kind='bar', color='coral')
                plt.title('Total Value by Stock Status')
                plt.xlabel('Stock Status')
                plt.ylabel('Total Value ($)')
                plt.xticks(rotation=45)

            plt.tight_layout()
            plt.show()

    def stock_level_analysis(df):
        """Analyze current stock levels"""
        print("\n" + "="*60)
        print("STOCK LEVEL ANALYSIS")
        print("="*60)
```

```
        if 'current_stock' in df.columns:
            print(f"Total Current Stock Units: {df['current_stock'].sum():,.0f}")
            print(f"Average Stock Level: {df['current_stock'].mean():.2f}")
            print(f"Median Stock Level: {df['current_stock'].median():.2f}")

            # Reorder level analysis
            if 'reorder_level' in df.columns:
                below_reorder = df[df['current_stock'] <= df['reorder_level']]
                print(f"\nItems Below Reorder Level: {len(below_reorder)} ({len(below_reorder)/len(df)*100:.1f}%)")

                if len(below_reorder) > 0:
                    print("\nItems Requiring Immediate Attention:")
                    print("-" * 40)
                    critical_items = below_reorder[['inventory_id', 'sku_id', 'current_stock', 'reorder_level', 'stock_status']].head(10)
                    print(critical_items.to_string(index=False))

            # Visualize stock levels
            plt.figure(figsize=(15, 10))

            plt.subplot(2, 2, 1)
            plt.hist(df['current_stock'], bins=30, color='lightblue', edgecolor='black', alpha=0.7)
            plt.title('Distribution of Current Stock Levels')
            plt.xlabel('Current Stock')
            plt.ylabel('Frequency')

            plt.subplot(2, 2, 2)
            if 'reorder_level' in df.columns:
                plt.scatter(df['current_stock'], df['reorder_level'], alpha=0.6, color='orange')
                plt.plot([0, df['current_stock'].max()], [0, df['current_stock'].max()], 'r--', label='Equal Line')
                plt.title('Current Stock vs Reorder Level')
                plt.xlabel('Current Stock')
                plt.ylabel('Reorder Level')
                plt.legend()

            if 'stock_status' in df.columns:
                plt.subplot(2, 2, 3)
                df.boxplot(column='current_stock', by='stock_status', ax=plt.gca())
                plt.title('Stock Levels by Status')
                plt.suptitle('')

            if 'turnover_rate' in df.columns:
                plt.subplot(2, 2, 4)
                plt.scatter(df['current_stock'], df['turnover_rate'], alpha=0.6, color='purple')
                plt.title('Stock Level vs Turnover Rate')
                plt.xlabel('Current Stock')
                plt.ylabel('Turnover Rate')

            plt.tight_layout()
            plt.show()

    def vendor_analysis(df):
        """Analyze vendor performance"""
        print("\n" + "="*60)
        print("VENDOR ANALYSIS")
        print("="*60)

        if 'vendor_id' in df.columns:
            vendor_stats = df.groupby('vendor_id').agg({
                'inventory_id': 'count',
                'stock_value': ['sum', 'mean'],
                'current_stock': 'sum',
                'turnover_rate': 'mean'
            }).round(2)

            vendor_stats.columns = ['Items_Count', 'Total_Value', 'Avg_Value', 'Total_Stock', 'Avg_Turnover']
            vendor_stats = vendor_stats.sort_values('Total_Value', ascending=False)

            print("Top 10 Vendors by Total Value:")
            print("-" * 35)
            print(vendor_stats.head(10))

            # Visualize vendor analysis
            plt.figure(figsize=(15, 5))

            plt.subplot(1, 3, 1)
            top_vendors = vendor_stats.head(10)
            top_vendors['Total_Value'].plot(kind='bar', color='lightcoral')
            plt.title('Top 10 Vendors by Total Value')
            plt.xlabel('Vendor ID')
```

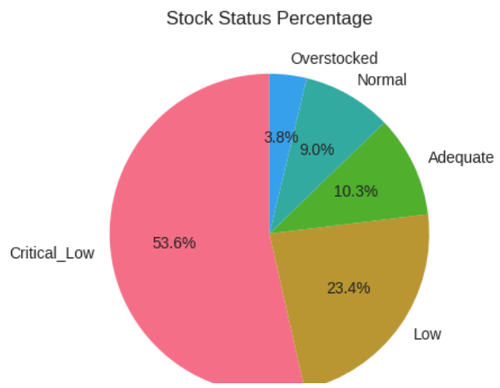
```
        plt.ylabel('Total Value ($)')
        plt.xticks(rotation=45)

        plt.subplot(1, 3, 2)
        top_vendors['Items_Count'].plot(kind='bar', color='lightgreen')
        plt.title('Top 10 Vendors by Item Count')
        plt.xlabel('Vendor ID')
        plt.ylabel('Number of Items')
        plt.xticks(rotation=45)

        plt.subplot(1, 3, 3)
        top_vendors['Avg_Turnover'].plot(kind='bar', color='gold')
        plt.title('Top 10 Vendors by Avg Turnover')
        plt.xlabel('Vendor ID')
        plt.ylabel('Average Turnover Rate')
        plt.xticks(rotation=45)

        plt.tight_layout()
        plt.show()

def expiry_analysis(df):
    """Analyze expiry dates and days to expiry"""
    print("\n" + "="*60)
    print("EXPIRY ANALYSIS")
    print("="*60)

    if 'days_to_expiry' in df.columns:
        # Items expiring soon (within 30 days)
        expiring_soon = df[df['days_to_expiry'] <= 30]
        expired_items = df[df['days_to_expiry'] <= 0]

        print(f"Items Expiring Within 30 Days: {len(expiring_soon)} ({len(expiring_soon)/len(df)*100:.1f}%)")
        print(f"Already Expired Items: {len(expired_items)} ({len(expired_items)/len(df)*100:.1f}%)")

        if len(expiring_soon) > 0:
            expiring_value = expiring_soon['stock_value'].sum()
            print(f"Value of Items Expiring Soon: ${expiring_value:,.2f}")

        # Expiry categories
        def categorize_expiry(days):
            if days <= 0:
                return 'Expired'
            elif days <= 30:
                return 'Expiring Soon (≤30 days)'
            elif days <= 90:
                return 'Short Term (≤90 days)'
            elif days <= 365:
                return 'Medium Term (≤1 year)'
            else:
                return 'Long Term (>1 year)'

        df['expiry_category'] = df['days_to_expiry'].apply(categorize_expiry)
        expiry_distribution = df['expiry_category'].value_counts()

        print("\nExpiry Distribution:")
        print("-" * 25)
        for category, count in expiry_distribution.items():
            percentage = (count / len(df)) * 100
            print(f"{category}: {count} ({percentage:.1f}%)")

        # Visualize expiry analysis
        plt.figure(figsize=(15, 5))

        plt.subplot(1, 3, 1)
        plt.hist(df['days_to_expiry'], bins=50, color='salmon', edgecolor='black', alpha=0.7)
        plt.title('Distribution of Days to Expiry')
        plt.xlabel('Days to Expiry')
        plt.ylabel('Frequency')
        plt.axvline(x=30, color='red', linestyle='--', label='30 Days')
        plt.axvline(x=0, color='darkred', linestyle='-', label='Expiry Date')
        plt.legend()

        plt.subplot(1, 3, 2)
        expiry_distribution.plot(kind='bar', color='lightpink')
        plt.title('Items by Expiry Category')
        plt.xlabel('Expiry Category')
        plt.ylabel('Count')
        plt.xticks(rotation=45)
```

```
            plt.subplot(1, 3, 3)
            expiry_value = df.groupby('expiry_category')['stock_value'].sum()
            expiry_value.plot(kind='bar', color='orange')
            plt.title('Value by Expiry Category')
            plt.xlabel('Expiry Category')
            plt.ylabel('Total Value ($)')
            plt.xticks(rotation=45)

            plt.tight_layout()
            plt.show()

    def turnover_analysis(df):
        """Analyze inventory turnover rates"""
        print("\n" + "="*60)
        print("TURNOVER ANALYSIS")
        print("="*60)

        if 'turnover_rate' in df.columns:
            avg_turnover = df['turnover_rate'].mean()
            median_turnover = df['turnover_rate'].median()

            print(f"Average Turnover Rate: {avg_turnover:.2f}")
            print(f"Median Turnover Rate: {median_turnover:.2f}")

            # Categorize turnover rates
            def categorize_turnover(rate):
                if rate >= 50:
                    return 'Very High (≥50)'
                elif rate >= 20:
                    return 'High (20-49)'
                elif rate >= 10:
                    return 'Medium (10-19)'
                elif rate >= 5:
                    return 'Low (5-9)'
                else:
                    return 'Very Low (<5)'

            df['turnover_category'] = df['turnover_rate'].apply(categorize_turnover)
            turnover_distribution = df['turnover_category'].value_counts()

            print("\nTurnover Rate Distribution:")
            print("-" * 30)
            for category, count in turnover_distribution.items():
                percentage = (count / len(df)) * 100
                print(f"{category}: {count} ({percentage:.1f}%)")

            # Slow-moving items (low turnover)
            slow_moving = df[df['turnover_rate'] < 5]
            print(f"\nSlow-Moving Items (Turnover < 5): {len(slow_moving)} items")
            if len(slow_moving) > 0:
                slow_moving_value = slow_moving['stock_value'].sum()
                print(f"Value Tied in Slow-Moving Items: ${slow_moving_value:,.2f}")

            # Visualize turnover analysis
            plt.figure(figsize=(15, 5))

            plt.subplot(1, 3, 1)
            plt.hist(df['turnover_rate'], bins=30, color='lightsteelblue', edgecolor='black', alpha=0.7)
            plt.title('Distribution of Turnover Rates')
            plt.xlabel('Turnover Rate')
            plt.ylabel('Frequency')

            plt.subplot(1, 3, 2)
            turnover_distribution.plot(kind='bar', color='mediumpurple')
            plt.title('Items by Turnover Category')
            plt.xlabel('Turnover Category')
            plt.ylabel('Count')
            plt.xticks(rotation=45)

            plt.subplot(1, 3, 3)
            if 'stock_status' in df.columns:
                df.boxplot(column='turnover_rate', by='stock_status', ax=plt.gca())
                plt.title('Turnover Rate by Stock Status')
                plt.suptitle('')

            plt.tight_layout()
            plt.show()

    def correlation_analysis(df):
```

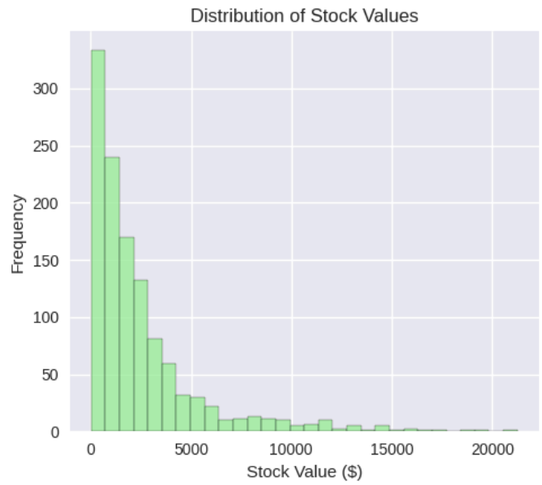
```
    """Analyze correlations between numerical variables"""
    print("\n" + "="*60)
    print("CORRELATION ANALYSIS")
    print("="*60)

    # Select numerical columns
    numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()

    # Remove ID columns for correlation analysis
    id_cols = [col for col in numerical_cols if 'id' in col.lower()]
    analysis_cols = [col for col in numerical_cols if col not in id_cols]

    if len(analysis_cols) > 1:
        correlation_matrix = df[analysis_cols].corr()

        print("Correlation Matrix:")
        print("-" * 20)
        print(correlation_matrix.round(3))

        # Find strong correlations (>0.7 or <-0.7)
        strong_corr = []
        for i in range(len(correlation_matrix.columns)):
            for j in range(i+1, len(correlation_matrix.columns)):
                corr_val = correlation_matrix.iloc[i, j]
                if abs(corr_val) > 0.7:
                    strong_corr.append((
                        correlation_matrix.columns[i],
                        correlation_matrix.columns[j],
                        corr_val
                    ))

        if strong_corr:
            print("\nStrong Correlations (|r| > 0.7):")
            print("-" * 35)
            for var1, var2, corr in strong_corr:
                print(f"{var1} - {var2}: {corr:.3f}")

        # Visualize correlation matrix
        plt.figure(figsize=(10, 8))
        sns.heatmap(correlation_matrix, annot=True, cmap='RdYlBu_r', center=0,
                    square=True, linewidths=0.5, cbar_kws={"shrink": .5})
        plt.title('Correlation Matrix Heatmap')
        plt.tight_layout()
        plt.show()

def generate_summary_report(df):
    """Generate a comprehensive summary report"""
    print("\n" + "="*60)
    print("EXECUTIVE SUMMARY REPORT")
    print("="*60)

    total_items = len(df)
    total_value = df['stock_value'].sum() if 'stock_value' in df.columns else 0

    print(f"📊 INVENTORY OVERVIEW")
    print(f"   • Total Items: {total_items:,}")
    print(f"   • Total Value: ${total_value:,.2f}")

    if 'stock_status' in df.columns:
        critical_low = len(df[df['stock_status'] == 'Critical_Low'])
        low_stock = len(df[df['stock_status'] == 'Low'])
        print(f"\n🚨 STOCK ALERTS")
        print(f"   • Critical Low: {critical_low} items ({critical_low/total_items*100:.1f}%)")
        print(f"   • Low Stock: {low_stock} items ({low_stock/total_items*100:.1f}%)")

    if 'days_to_expiry' in df.columns:
        expiring_soon = len(df[df['days_to_expiry'] <= 30])
        expired = len(df[df['days_to_expiry'] <= 0])
        print(f"\n⏰ EXPIRY ALERTS")
        print(f"   • Expiring ≤30 days: {expiring_soon} items ({expiring_soon/total_items*100:.1f}%)")
        print(f"   • Already Expired: {expired} items ({expired/total_items*100:.1f}%)")

    if 'turnover_rate' in df.columns:
        slow_moving = len(df[df['turnover_rate'] < 5])
        fast_moving = len(df[df['turnover_rate'] >= 20])
        print(f"\n🔄 TURNOVER INSIGHTS")
        print(f"   • Fast Moving (≥20): {fast_moving} items ({fast_moving/total_items*100:.1f}%)")
        print(f"   • Slow Moving (<5): {slow_moving} items ({slow_moving/total_items*100:.1f}%)")
```

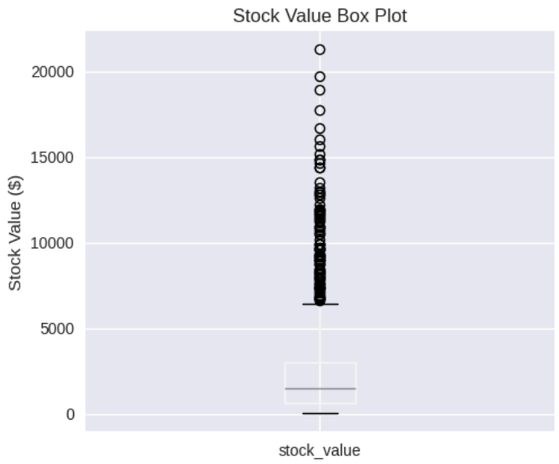
```
        if 'vendor_id' in df.columns:
            unique_vendors = df['vendor_id'].nunique()
            top_vendor = df['vendor_id'].value_counts().index[0]
            top_vendor_count = df['vendor_id'].value_counts().iloc[0]
            print(f"\n🏭 VENDOR INSIGHTS")
            print(f"   • Total Vendors: {unique_vendors}")
            print(f"   • Top Vendor: {top_vendor} ({top_vendor_count} items)")

    def main():
        """Main function to run the complete EDA"""
        # Specify your Excel file path here
        file_path = input("Enter the path to your Excel file: ").strip('"')

        # Load data
        print("Loading data...")
        df = load_and_clean_data(file_path)

        if df is not None:
            print(f"Data loaded successfully! Shape: {df.shape}")

            # Run all analyses
            basic_info_analysis(df)
            stock_status_analysis(df)
            inventory_value_analysis(df)
            stock_level_analysis(df)
            vendor_analysis(df)
            expiry_analysis(df)
            turnover_analysis(df)
            correlation_analysis(df)
            generate_summary_report(df)

            print("\n" + "="*60)
            print("EDA COMPLETED SUCCESSFULLY!")
            print("="*60)

        else:
            print("Failed to load data. Please check the file path and format.")

    if __name__ == "__main__":
        main()
```

```
Enter the path to your Excel file: /content/drive/MyDrive/CAPSTONE/Data/Inventory.xlsx
Loading data...
Data loaded successfully! Shape: (1197, 16)
============================================================
BASIC DATA INFORMATION
============================================================
Dataset Shape: (1197, 16)
Total Records: 1197
Total Features: 16

Column Names and Data Types:
----------------------------------------
inventory_id: object
hospital_id: object
sku_id: object
vendor_id: object
current_stock: float64
reorder_level: float64
max_stock_level: float64
stock_status: object
days_of_stock: float64
estimated_daily_consumption: float64
last_order_date: datetime64[ns]
expiry_date: datetime64[ns]
days_to_expiry: int64
batch_number: object
stock_value: float64
turnover_rate: float64

Missing Values:
--------------------
Empty DataFrame
Columns: [Missing Count, Percentage]
Index: []

Basic Statistics for Numerical Columns:
----------------------------------------------
       current_stock  reorder_level  max_stock_level  days_of_stock  \
count        1197.00        1197.00          1197.00        1197.00
mean           42.86         114.93           373.10          15.07
std            22.07          96.58           320.11          15.28
min             5.06           5.14            17.00           0.50
25%            24.10          43.64           140.74           5.30
50%            43.48          85.02           268.66          10.20
75%            61.90         161.72           512.30          19.30
max            79.99         606.92          2103.14         112.20

       estimated_daily_consumption  days_to_expiry  stock_value  turnover_rate
count                      1197.00         1197.00      1197.00        1197.00
mean                          4.64         1274.72      2455.55          63.66
std                           3.18          521.56      2863.47          85.19
min                           0.47          367.00        19.32           3.25
25%                           2.30          873.00       660.22          18.87
50%                           3.73         1244.00      1521.29          35.76
75%                           6.19         1623.00      3012.78          68.83
max                          16.89         2543.00     21253.44         793.03

============================================================
STOCK STATUS ANALYSIS
============================================================
Stock Status Distribution:
------------------------------
Critical_Low: 641 (53.6%)
Low: 280 (23.4%)
Adequate: 123 (10.3%)
Normal: 108 (9.0%)
Overstocked: 45 (3.8%)
```

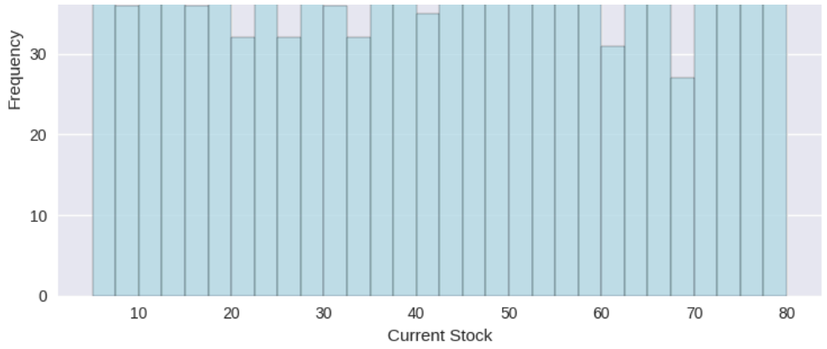


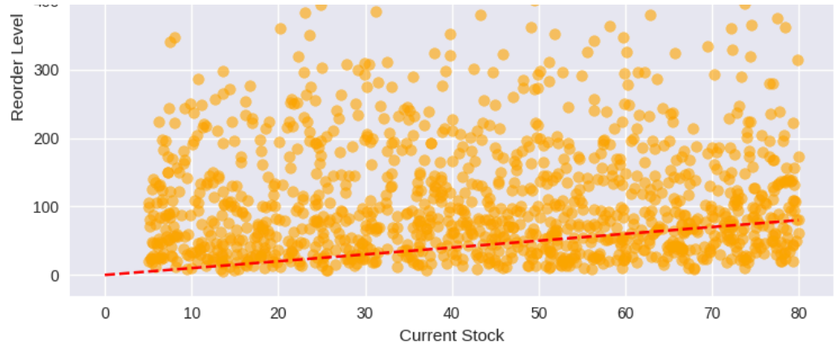

```
============================================================
INVENTORY VALUE ANALYSIS
============================================================
Total Inventory Value: $2,939,287.86
Average Item Value: $2,455.55
Median Item Value: $1,521.29

Value Distribution by Stock Status:
----------------------------------------
                      sum     mean  count
stock_status
Adequate       331914.11  2698.49    123
Critical_Low  1297540.39  2024.24    641
Low            883359.74  3154.86    280
Normal         316134.63  2927.17    108
Overstocked    110338.99  2451.98     45
```

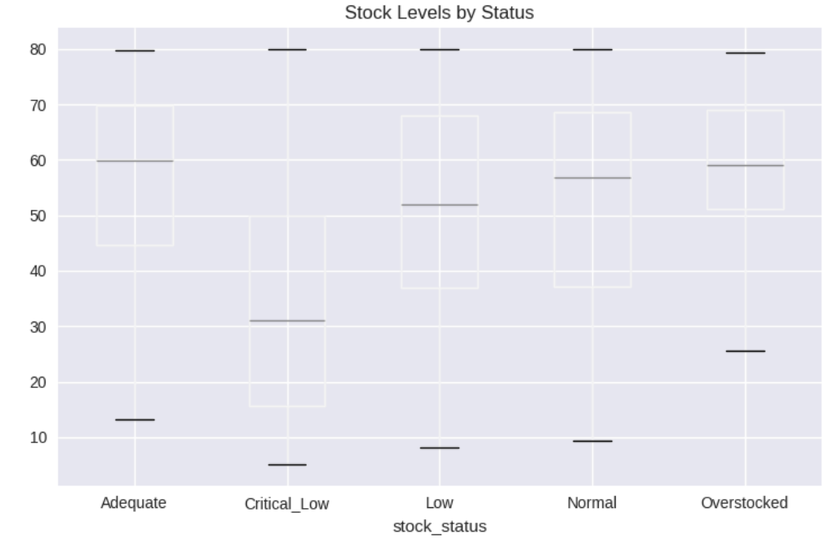


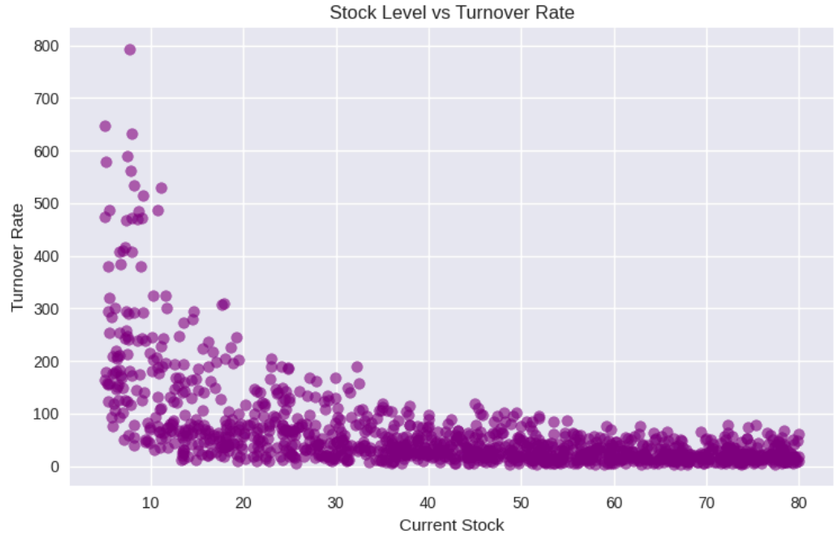



```
============================================================
STOCK LEVEL ANALYSIS
============================================================
Total Current Stock Units: 51,305
Average Stock Level: 42.86
Median Stock Level: 43.48

Items Below Reorder Level: 921 (76.9%)

Items Requiring Immediate Attention:
----------------------------------------
inventory_id   sku_id  current_stock  reorder_level stock_status
   INV000001 SKU00001          48.82         187.39 Critical_Low
   INV000005 SKU00005          74.11         223.28 Critical_Low
   INV000007 SKU00007          25.62          44.34          Low
   INV000008 SKU00008          44.65          73.73          Low
   INV000009 SKU00009          31.17          64.85 Critical_Low
   INV000010 SKU00010          62.20         164.40 Critical_Low
   INV000012 SKU00012          28.34          55.09          Low
   INV000015 SKU00015          57.13          94.27          Low
   INV000018 SKU00018          10.74          46.76 Critical_Low
   INV000019 SKU00019          13.41          87.65 Critical_Low
```

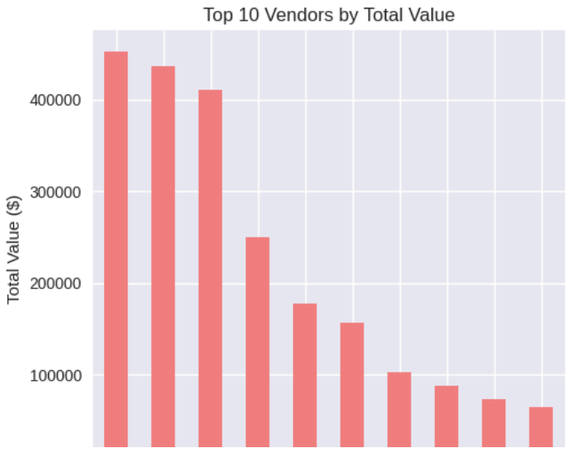


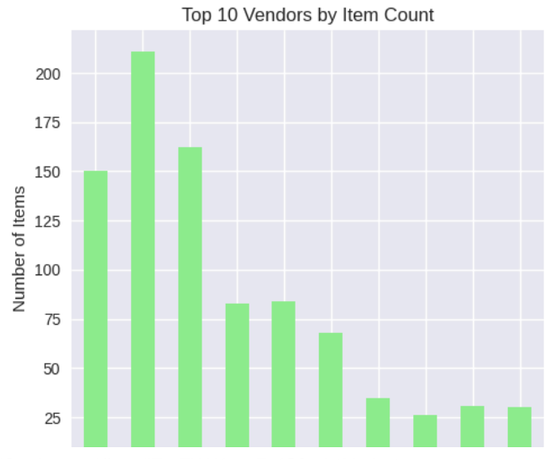


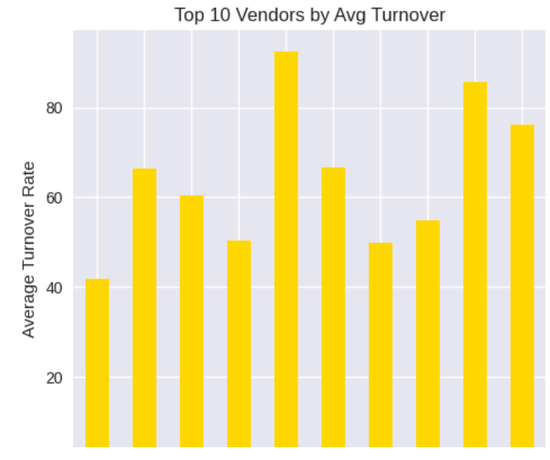




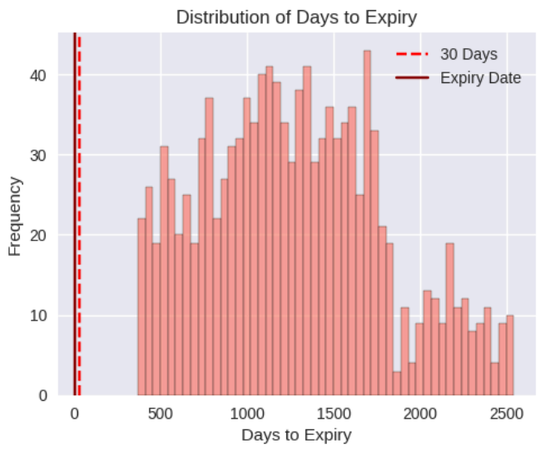

```
============================================================
VENDOR ANALYSIS
============================================================
Top 10 Vendors by Total Value:
----------------------------------
           Items_Count  Total_Value  Avg_Value  Total_Stock  Avg_Turnover
vendor_id
V006               150    452811.27    3018.74      6586.86         41.85
V001               211    436959.24    2070.90      8850.51         66.32
V004               162    410644.94    2534.85      6862.78         60.39
V003                83    250011.25    3012.18      3564.91         50.46
V002                84    177635.98    2114.71      3931.32         92.53
V008                68    156207.46    2297.17      2920.25         66.76
V020                35    102252.88    2921.51      1642.93         49.82
V010                26     87519.90    3366.15      1289.44         54.84
V009                31     72797.85    2348.32      1315.64         85.72
V024                30     64473.40    2149.11      1173.23         76.12
```

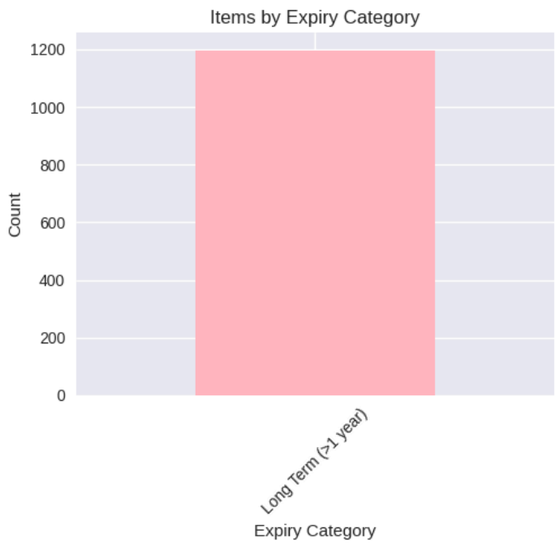


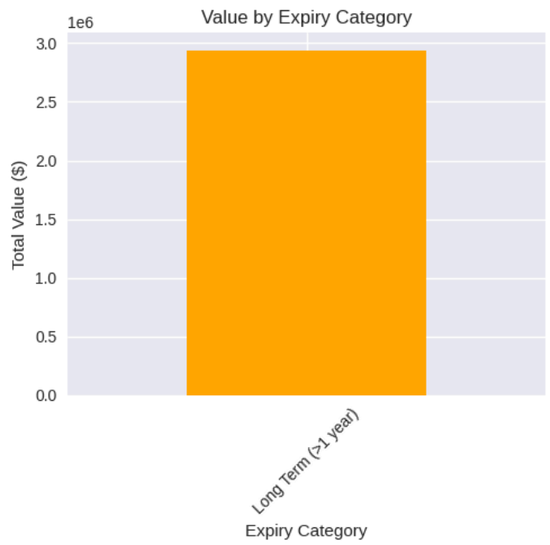




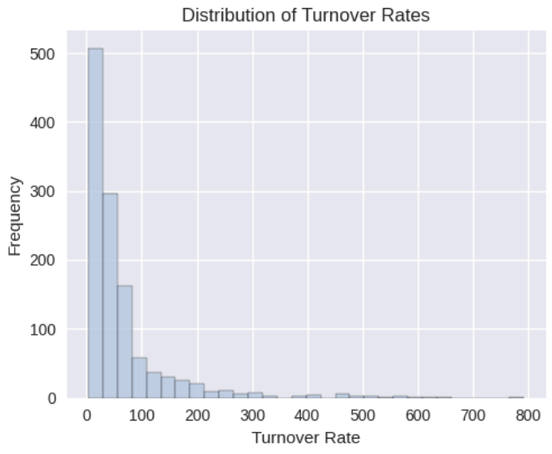

```
============================================================
EXPIRY ANALYSIS
============================================================
Items Expiring Within 30 Days: 0 (0.0%)
Already Expired Items: 0 (0.0%)

Expiry Distribution:
-------------------------
Long Term (>1 year): 1197 (100.0%)
```

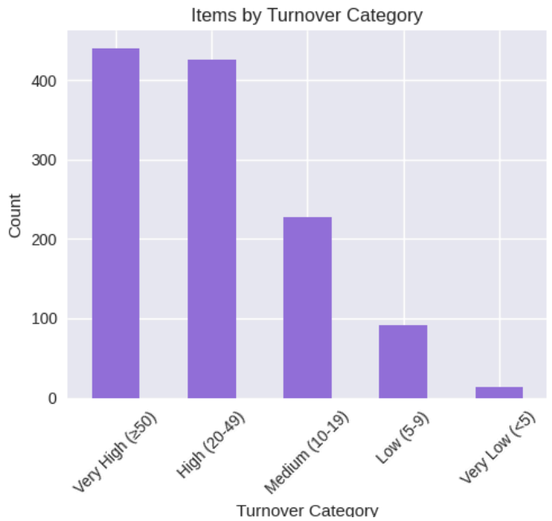


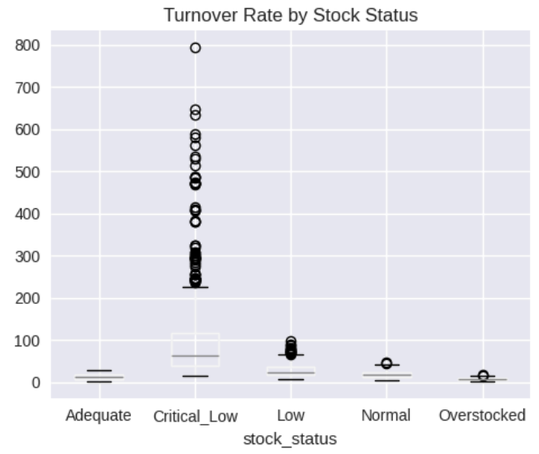




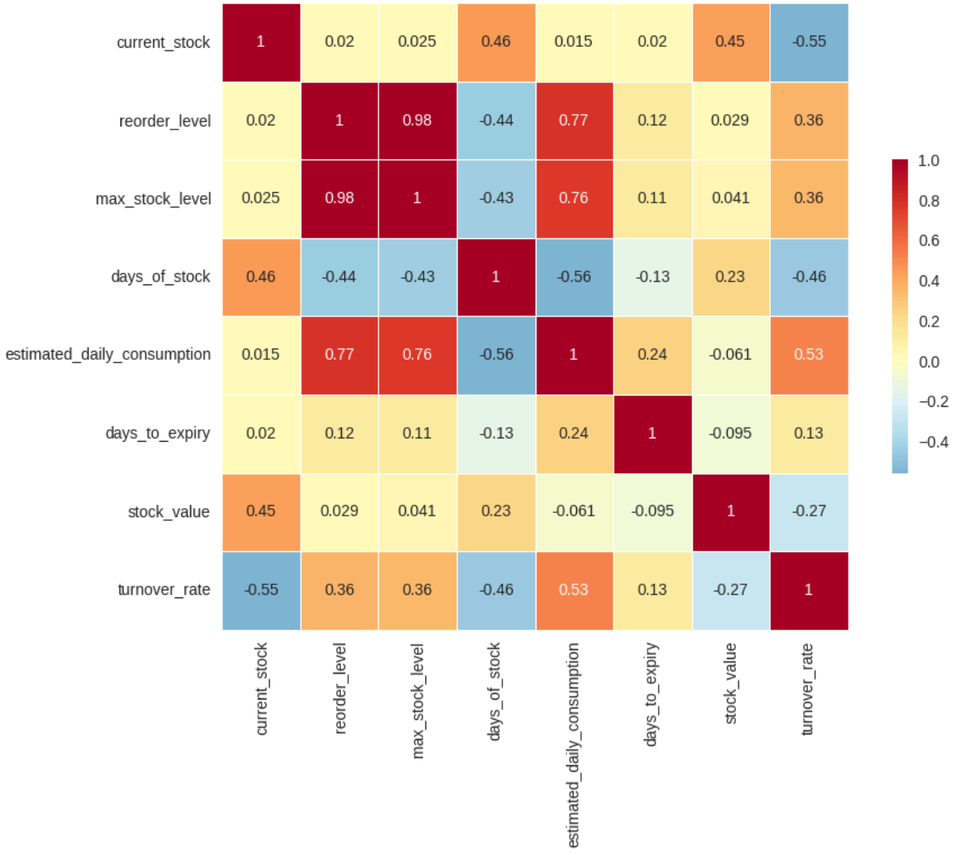

```
============================================================
TURNOVER ANALYSIS
============================================================
Average Turnover Rate: 63.66
Median Turnover Rate: 35.76

Turnover Rate Distribution:
------------------------------
Very High (≥50): 440 (36.8%)
High (20-49): 425 (35.5%)
Medium (10-19): 228 (19.0%)
Low (5-9): 91 (7.6%)
Very Low (<5): 13 (1.1%)

Slow-Moving Items (Turnover < 5): 13 items
Value Tied in Slow-Moving Items: $73,106.42
```







```
============================================================
CORRELATION ANALYSIS
============================================================
Correlation Matrix:
--------------------
                           current_stock  reorder_level  max_stock_level  \
current_stock                      1.000          0.020            0.025
reorder_level                      0.020          1.000            0.980
max_stock_level                    0.025          0.980            1.000
days_of_stock                      0.458         -0.445           -0.433
estimated_daily_consumption        0.015          0.774            0.761
days_to_expiry                     0.020          0.116            0.114
stock_value                        0.445          0.029            0.041
turnover_rate                     -0.551          0.363            0.358

                           days_of_stock  estimated_daily_consumption  \
current_stock                      0.458                        0.015
reorder_level                     -0.445                        0.774
max_stock_level                   -0.433                        0.761
days_of_stock                      1.000                       -0.557
estimated_daily_consumption       -0.557                        1.000
days_to_expiry                    -0.134                        0.240
stock_value                        0.229                       -0.061
turnover_rate                     -0.457                        0.528

                           days_to_expiry  stock_value  turnover_rate
current_stock                       0.020        0.445         -0.551
reorder_level                       0.116        0.029          0.363
max_stock_level                     0.114        0.041          0.358
days_of_stock                      -0.134        0.229         -0.457
estimated_daily_consumption         0.240       -0.061          0.528
days_to_expiry                      1.000       -0.095          0.126
stock_value                        -0.095        1.000         -0.272
turnover_rate                       0.126       -0.272          1.000

Strong Correlations (|r| > 0.7):
----------------------------------
reorder_level - max_stock_level: 0.980
reorder_level - estimated_daily_consumption: 0.774
max_stock_level - estimated_daily_consumption: 0.761
```



```
============================================================
EXECUTIVE SUMMARY REPORT
============================================================
📊 INVENTORY OVERVIEW
   • Total Items: 1,197
   • Total Value: $2,939,287.86

🚨 STOCK ALERTS
   • Critical Low: 641 items (53.6%)
   • Low Stock: 280 items (23.4%)

⏰ EXPIRY ALERTS
   • Expiring ≤30 days: 0 items (0.0%)
   • Already Expired: 0 items (0.0%)

🔄 TURNOVER INSIGHTS
   • Fast Moving (≥20): 865 items (72.3%)
   • Slow Moving (<5): 13 items (1.1%)

🏭 VENDOR INSIGHTS
   • Total Vendors: 49
   • Top Vendor: V001 (211 items)

============================================================
EDA COMPLETED SUCCESSFULLY!
============================================================
```

Start coding or generate with AI.

Hospital Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from datetime import datetime

# Set style and suppress warnings
plt.style.use('default')
warnings.filterwarnings('ignore')

class HospitalDataEDA:
    def __init__(self, file_path):
        """Initialize the EDA class with the Excel file path"""
        self.file_path = file_path
        self.sheets_data = {}
        self.load_data()

    def load_data(self):
        """Load all sheets from the Excel file"""
        try:
            excel_file = pd.ExcelFile(self.file_path)
            print(f"Loading data from: {self.file_path}")
            print(f"Available sheets: {excel_file.sheet_names}")

            for sheet_name in excel_file.sheet_names:
                self.sheets_data[sheet_name] = pd.read_excel(self.file_path, sheet_name=sheet_name)
                print(f"Loaded sheet '{sheet_name}' with shape: {self.sheets_data[sheet_name].shape}")

        except Exception as e:
            print(f"Error loading data: {str(e)}")

    def basic_info(self):
        """Display basic information about all datasets"""
        print("\n" + "="*80)
        print("BASIC DATA INFORMATION")
        print("="*80)

        for sheet_name, df in self.sheets_data.items():
            print(f"\nSHEET: {sheet_name.upper()}")
            print("-" * 50)
            print(f"Shape: {df.shape}")
            print(f"Columns: {list(df.columns)}")
            print("\nData Types:")
            print(df.dtypes)
            print("\nMissing Values:")
            missing = df.isnull().sum()
            if missing.sum() > 0:
                print(missing[missing > 0])
            else:
                print("No missing values found!")

            print("\nFirst 3 rows:")
            print(df.head(3))
            print("\n" + "-"*50)

    def hospital_analysis(self):
        """Analyze hospital basic information"""
        hospitals_df = None

        # Find hospital data
        for sheet_name, df in self.sheets_data.items():
            if 'hospital_name' in df.columns or 'hospital_id' in df.columns:
                hospitals_df = df
                break

        if hospitals_df is None:
            print("Hospital data not found")
            return

        print("\n" + "="*80)
        print("HOSPITAL ANALYSIS")
```

```
        print("="*80)

        # Basic statistics
        if 'bed_capacity' in hospitals_df.columns:
            print(f"\nBed Capacity Statistics:")
            print(hospitals_df['bed_capacity'].describe())

        if 'annual_budget' in hospitals_df.columns:
            print(f"\nAnnual Budget Statistics:")
            print(hospitals_df['annual_budget'].describe())

        # Visualizations
        fig, axes = plt.subplots(2, 2, figsize=(15, 10))
        fig.suptitle('Hospital Overview Analysis', fontsize=16)

        # Hospital type distribution
        if 'hospital_type' in hospitals_df.columns:
            type_counts = hospitals_df['hospital_type'].value_counts()
            axes[0, 0].pie(type_counts.values, labels=type_counts.index, autopct='%1.1f%%')
            axes[0, 0].set_title('Hospital Type Distribution')

        # Country distribution
        if 'country' in hospitals_df.columns:
            country_counts = hospitals_df['country'].value_counts()
            axes[0, 1].bar(country_counts.index, country_counts.values)
            axes[0, 1].set_title('Hospitals by Country')
            plt.setp(axes[0, 1].xaxis.get_majorticklabels(), rotation=45)

        # Bed capacity distribution
        if 'bed_capacity' in hospitals_df.columns:
            axes[1, 0].hist(hospitals_df['bed_capacity'], bins=20, edgecolor='black', alpha=0.7)
            axes[1, 0].set_title('Bed Capacity Distribution')
            axes[1, 0].set_xlabel('Bed Capacity')
            axes[1, 0].set_ylabel('Frequency')

        # Establishment year
        if 'establishment_year' in hospitals_df.columns:
            axes[1, 1].hist(hospitals_df['establishment_year'], bins=15, edgecolor='black', alpha=0.7)
            axes[1, 1].set_title('Hospital Establishment Year')
            axes[1, 1].set_xlabel('Year')
            axes[1, 1].set_ylabel('Count')

        plt.tight_layout()
        plt.show()

    def department_analysis(self):
        """Analyze department data"""
        dept_df = None
        for sheet_name, df in self.sheets_data.items():
            if 'dept_name' in df.columns or 'department' in df.columns:
                dept_df = df
                break

        if dept_df is None:
            print("Department data not found")
            return

        print("\n" + "="*80)
        print("DEPARTMENT ANALYSIS")
        print("="*80)

        # Department statistics
        if 'dept_name' in dept_df.columns:
            print(f"\nTotal Departments: {dept_df['dept_name'].nunique()}")
            print("Department Distribution:")
            print(dept_df['dept_name'].value_counts())

        if 'bed_count' in dept_df.columns:
            print(f"\nBed Count Statistics by Department:")
            bed_stats = dept_df.groupby('dept_name')['bed_count'].agg(['sum', 'mean', 'count']).round(2)
            print(bed_stats)

        # Visualizations
        fig, axes = plt.subplots(2, 2, figsize=(15, 10))
        fig.suptitle('Department Analysis', fontsize=16)

        # Department type distribution
        if 'dept_type' in dept_df.columns:
            type_counts = dept_df['dept_type'].value_counts()
```

```
            axes[0, 0].pie(type_counts.values, labels=type_counts.index, autopct='%1.1f%%')
            axes[0, 0].set_title('Department Type Distribution')

        # Bed count by department
        if 'dept_name' in dept_df.columns and 'bed_count' in dept_df.columns:
            dept_beds = dept_df.groupby('dept_name')['bed_count'].sum().sort_values(ascending=False).head(10)
            axes[0, 1].barh(dept_beds.index, dept_beds.values)
            axes[0, 1].set_title('Top 10 Departments by Total Beds')
            axes[0, 1].set_xlabel('Total Beds')

        # Monthly budget distribution
        if 'monthly_budget' in dept_df.columns:
            axes[1, 0].hist(dept_df['monthly_budget'], bins=20, edgecolor='black', alpha=0.7)
            axes[1, 0].set_title('Monthly Budget Distribution')
            axes[1, 0].set_xlabel('Monthly Budget')
            axes[1, 0].set_ylabel('Frequency')

        # Budget vs Bed Count correlation
        if 'monthly_budget' in dept_df.columns and 'bed_count' in dept_df.columns:
            axes[1, 1].scatter(dept_df['bed_count'], dept_df['monthly_budget'], alpha=0.6)
            axes[1, 1].set_title('Budget vs Bed Count Correlation')
            axes[1, 1].set_xlabel('Bed Count')
            axes[1, 1].set_ylabel('Monthly Budget')

            # Add correlation coefficient
            corr = dept_df['bed_count'].corr(dept_df['monthly_budget'])
            axes[1, 1].text(0.05, 0.95, f'Correlation: {corr:.3f}',
                            transform=axes[1, 1].transAxes, fontsize=10,
                            bbox=dict(boxstyle='round', facecolor='wheat'))

        plt.tight_layout()
        plt.show()

    def physician_analysis(self):
        """Analyze physician data"""
        physician_df = None
        for sheet_name, df in self.sheets_data.items():
            if 'physician_id' in df.columns or 'specialty' in df.columns:
                physician_df = df
                break

        if physician_df is None:
            print("Physician data not found")
            return

        print("\n" + "="*80)
        print("PHYSICIAN ANALYSIS")
        print("="*80)

        # Basic statistics
        print(f"\nTotal Physicians: {len(physician_df)}")

        if 'specialty' in physician_df.columns:
            print(f"Specialties: {physician_df['specialty'].nunique()}")
            print("\nTop 10 Specialties:")
            print(physician_df['specialty'].value_counts().head(10))

        if 'experience_years' in physician_df.columns:
            print(f"\nExperience Statistics:")
            print(physician_df['experience_years'].describe())

        # Visualizations
        fig, axes = plt.subplots(2, 3, figsize=(18, 10))
        fig.suptitle('Physician Analysis Dashboard', fontsize=16)

        # Specialty distribution
        if 'specialty' in physician_df.columns:
            specialty_counts = physician_df['specialty'].value_counts().head(8)
            axes[0, 0].pie(specialty_counts.values, labels=specialty_counts.index, autopct='%1.1f%%')
            axes[0, 0].set_title('Top Specialties Distribution')

        # Experience distribution
        if 'experience_years' in physician_df.columns:
            axes[0, 1].hist(physician_df['experience_years'], bins=20, edgecolor='black', alpha=0.7)
            axes[0, 1].set_title('Experience Years Distribution')
            axes[0, 1].set_xlabel('Years of Experience')
            axes[0, 1].set_ylabel('Count')

        # Nationality distribution
```

```
        if 'nationality' in physician_df.columns:
            nat_counts = physician_df['nationality'].value_counts().head(8)
            axes[0, 2].bar(nat_counts.index, nat_counts.values)
            axes[0, 2].set_title('Physician Nationality Distribution')
            plt.setp(axes[0, 2].xaxis.get_majorticklabels(), rotation=45)

        # Employment type
        if 'employment_type' in physician_df.columns:
            emp_counts = physician_df['employment_type'].value_counts()
            axes[1, 0].pie(emp_counts.values, labels=emp_counts.index, autopct='%1.1f%%')
            axes[1, 0].set_title('Employment Type Distribution')

        # Prescribing preference
        if 'prescribing_preference' in physician_df.columns:
            pref_counts = physician_df['prescribing_preference'].value_counts()
            axes[1, 1].bar(pref_counts.index, pref_counts.values)
            axes[1, 1].set_title('Prescribing Preference')
            plt.setp(axes[1, 1].xaxis.get_majorticklabels(), rotation=45)

        # Cost consciousness vs Prescription volume
        if 'cost_consciousness' in physician_df.columns and 'prescription_volume' in physician_df.columns:
            cost_vol = physician_df.groupby('cost_consciousness')['prescription_volume'].count()
            axes[1, 2].bar(cost_vol.index, cost_vol.values)
            axes[1, 2].set_title('Count by Cost Consciousness')
            plt.setp(axes[1, 2].xaxis.get_majorticklabels(), rotation=45)

        plt.tight_layout()
        plt.show()

    def pharmacy_analysis(self):
        """Analyze pharmacy/medication data"""
        pharmacy_df = None
        for sheet_name, df in self.sheets_data.items():
            if 'sku_id' in df.columns or 'sku_name' in df.columns:
                pharmacy_df = df
                break

        if pharmacy_df is None:
            print("Pharmacy data not found")
            return

        print("\n" + "="*80)
        print("PHARMACY ANALYSIS")
        print("="*80)

        # Basic statistics
        print(f"\nTotal SKUs: {len(pharmacy_df)}")

        if 'category' in pharmacy_df.columns:
            print(f"Categories: {pharmacy_df['category'].nunique()}")
            print(pharmacy_df['category'].value_counts())

        if 'standard_cost' in pharmacy_df.columns:
            print(f"\nCost Statistics:")
            print(pharmacy_df['standard_cost'].describe())

        # Visualizations
        fig, axes = plt.subplots(2, 3, figsize=(18, 10))
        fig.suptitle('Pharmacy & Medication Analysis', fontsize=16)

        # Category distribution
        if 'category' in pharmacy_df.columns:
            cat_counts = pharmacy_df['category'].value_counts()
            axes[0, 0].pie(cat_counts.values, labels=cat_counts.index, autopct='%1.1f%%')
            axes[0, 0].set_title('Medication Categories')

        # Brand vs Generic
        if 'brand_generic_flag' in pharmacy_df.columns:
            brand_counts = pharmacy_df['brand_generic_flag'].value_counts()
            axes[0, 1].bar(brand_counts.index, brand_counts.values)
            axes[0, 1].set_title('Brand vs Generic Distribution')

        # Cost distribution
        if 'standard_cost' in pharmacy_df.columns:
            axes[0, 2].hist(pharmacy_df['standard_cost'], bins=30, edgecolor='black', alpha=0.7)
            axes[0, 2].set_title('Standard Cost Distribution')
            axes[0, 2].set_xlabel('Cost')
            axes[0, 2].set_ylabel('Frequency')
```

```
        # Shelf life analysis
        if 'shelf_life_days' in pharmacy_df.columns:
            axes[1, 0].hist(pharmacy_df['shelf_life_days'], bins=25, edgecolor='black', alpha=0.7)
            axes[1, 0].set_title('Shelf Life Distribution')
            axes[1, 0].set_xlabel('Days')
            axes[1, 0].set_ylabel('Count')

        # Storage requirements
        if 'storage_requirements' in pharmacy_df.columns:
            storage_counts = pharmacy_df['storage_requirements'].value_counts()
            axes[1, 1].pie(storage_counts.values, labels=storage_counts.index, autopct='%1.1f%%')
            axes[1, 1].set_title('Storage Requirements')

        # Controlled substance analysis
        if 'controlled_substance_flag' in pharmacy_df.columns:
            controlled_counts = pharmacy_df['controlled_substance_flag'].value_counts()
            axes[1, 2].bar(controlled_counts.index, controlled_counts.values)
            axes[1, 2].set_title('Controlled Substance Distribution')

        plt.tight_layout()
        plt.show()

    def vendor_analysis(self):
        """Analyze vendor data"""
        vendor_df = None
        for sheet_name, df in self.sheets_data.items():
            if 'vendor_id' in df.columns or 'vendor_type' in df.columns:
                vendor_df = df
                break

        if vendor_df is None:
            print("Vendor data not found")
            return

        print("\n" + "="*80)
        print("VENDOR ANALYSIS")
        print("="*80)

        # Basic statistics
        print(f"\nTotal Vendors: {len(vendor_df)}")

        if 'vendor_type' in vendor_df.columns:
            print("Vendor Types:")
            print(vendor_df['vendor_type'].value_counts())

        if 'market_share_percent' in vendor_df.columns:
            print(f"\nMarket Share Statistics:")
            print(vendor_df['market_share_percent'].describe())

        # Visualizations
        fig, axes = plt.subplots(2, 3, figsize=(18, 10))
        fig.suptitle('Vendor Performance Analysis', fontsize=16)

        # Vendor type distribution
        if 'vendor_type' in vendor_df.columns:
            type_counts = vendor_df['vendor_type'].value_counts()
            axes[0, 0].pie(type_counts.values, labels=type_counts.index, autopct='%1.1f%%')
            axes[0, 0].set_title('Vendor Type Distribution')

        # Performance category
        if 'performance_category' in vendor_df.columns:
            perf_counts = vendor_df['performance_category'].value_counts()
            axes[0, 1].bar(perf_counts.index, perf_counts.values)
            axes[0, 1].set_title('Performance Categories')

        # Quality rating distribution
        if 'quality_rating' in vendor_df.columns:
            axes[0, 2].hist(vendor_df['quality_rating'], bins=20, edgecolor='black', alpha=0.7)
            axes[0, 2].set_title('Quality Rating Distribution')
            axes[0, 2].set_xlabel('Rating')
            axes[0, 2].set_ylabel('Count')

        # Price volatility vs Delivery reliability
        if 'price_volatility' in vendor_df.columns and 'delivery_reliability' in vendor_df.columns:
            axes[1, 0].scatter(vendor_df['price_volatility'], vendor_df['delivery_reliability'], alpha=0.6)
            axes[1, 0].set_title('Price Volatility vs Delivery Reliability')
            axes[1, 0].set_xlabel('Price Volatility')
            axes[1, 0].set_ylabel('Delivery Reliability')
```

```
            # Fill rate percentage
            if 'fill_rate_percentage' in vendor_df.columns:
                axes[1, 1].hist(vendor_df['fill_rate_percentage'], bins=20, edgecolor='black', alpha=0.7)
                axes[1, 1].set_title('Fill Rate Percentage Distribution')
                axes[1, 1].set_xlabel('Fill Rate %')
                axes[1, 1].set_ylabel('Count')

            # Market share analysis
            if 'market_share_percent' in vendor_df.columns:
                axes[1, 2].hist(vendor_df['market_share_percent'], bins=20, edgecolor='black', alpha=0.7)
                axes[1, 2].set_title('Market Share Distribution')
                axes[1, 2].set_xlabel('Market Share %')
                axes[1, 2].set_ylabel('Count')

            plt.tight_layout()
            plt.show()

        def correlation_analysis(self):
            """Perform correlation analysis for numerical columns"""
            print("\n" + "="*80)
            print("CORRELATION ANALYSIS")
            print("="*80)

            for sheet_name, df in self.sheets_data.items():
                print(f"\nCorrelations for {sheet_name.upper()}:")

                # Select only numerical columns
                numerical_cols = df.select_dtypes(include=[np.number]).columns

                if len(numerical_cols) > 1:
                    correlation_matrix = df[numerical_cols].corr()

                    # Plot correlation heatmap
                    plt.figure(figsize=(10, 8))
                    mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
                    sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='coolwarm',
                                center=0, fmt='.2f', square=True)
                    plt.title(f'Correlation Matrix - {sheet_name.upper()}')
                    plt.tight_layout()
                    plt.show()

                    # Show strongest correlations
                    print("Strongest correlations found:")
                    for i in range(len(correlation_matrix.columns)):
                        for j in range(i+1, len(correlation_matrix.columns)):
                            corr_value = correlation_matrix.iloc[i, j]
                            if abs(corr_value) > 0.5:  # Only show strong correlations
                                print(f"  {correlation_matrix.columns[i]} <-> {correlation_matrix.columns[j]}: {corr_value:.3f}")
                else:
                    print("Not enough numerical columns for correlation analysis")

        def generate_summary_insights(self):
            """Generate key business insights"""
            print("\n" + "="*80)
            print("KEY BUSINESS INSIGHTS")
            print("="*80)

            insights = []

            for sheet_name, df in self.sheets_data.items():
                if 'hospital' in sheet_name.lower():
                    if 'bed_capacity' in df.columns:
                        avg_capacity = df['bed_capacity'].mean()
                        insights.append(f"Average hospital bed capacity: {avg_capacity:.0f} beds")

                    if 'annual_budget' in df.columns:
                        total_budget = df['annual_budget'].sum()
                        insights.append(f"Total annual budget across hospitals: ${total_budget:,.0f}")

                elif 'dept' in sheet_name.lower():
                    if 'monthly_budget' in df.columns:
                        highest_budget_dept = df.loc[df['monthly_budget'].idxmax(), 'dept_name']
                        highest_budget = df['monthly_budget'].max()
                        insights.append(f"Highest budget department: {highest_budget_dept} (${highest_budget:,.0f}/month)")

                elif 'physician' in sheet_name.lower():
                    if 'experience_years' in df.columns:
                        avg_experience = df['experience_years'].mean()
                        insights.append(f"Average physician experience: {avg_experience:.1f} years")
```

```
            # Print insights
            for i, insight in enumerate(insights, 1):
                print(f"{i}. {insight}")

        def run_complete_eda(self):
            """Run the complete EDA process"""
            print("STARTING COMPREHENSIVE HOSPITAL DATA EDA")
            print("="*80)

            # Basic information
            self.basic_info()

            # Individual analyses
            self.hospital_analysis()
            self.department_analysis()
            self.physician_analysis()
            self.pharmacy_analysis()
            self.vendor_analysis()

            # Advanced analysis
            self.correlation_analysis()
            self.generate_summary_insights()

            print("\nEDA COMPLETE!")
            print("="*80)

    # Simple usage function
    def run_eda(file_path):
        """Simple function to run EDA"""
        try:
            eda = HospitalDataEDA(file_path)
            eda.run_complete_eda()
        except Exception as e:
            print(f"Error: {e}")

    # Quick data preview function
    def preview_data(file_path):
        """Quick preview of the Excel file"""
        try:
            excel_file = pd.ExcelFile(file_path)
            print(f"File: {file_path}")
            print(f"Sheets: {excel_file.sheet_names}")

            for sheet_name in excel_file.sheet_names:
                df = pd.read_excel(file_path, sheet_name=sheet_name)
                print(f"\nSheet: {sheet_name}")
                print(f"Size: {df.shape}")
                print(f"Columns: {list(df.columns)}")
                print("First 3 rows:")
                print(df.head(3))
                print("-" * 50)

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

    # Main execution
    if __name__ == "__main__":
        # UPDATE THIS PATH TO YOUR EXCEL FILE
        file_path = "/content/drive/MyDrive/CAPSTONE/Data/Hospital_Dataset.xlsx"  # Change this to your file path

        print("Hospital Data EDA System")
        print("Choose an option:")
        print("1. Quick Preview")
        print("2. Complete EDA")

        # For quick preview, uncomment the line below:
        preview_data(file_path)

        # For complete EDA, uncomment the line below:
        run_eda(file_path)

        print("\nTo run the analysis:")
        print("1. Update 'file_path' variable with your Excel file location")
        print("2. Uncomment either preview_data(file_path) or run_eda(file_path)")
        print("3. Run the script")
```

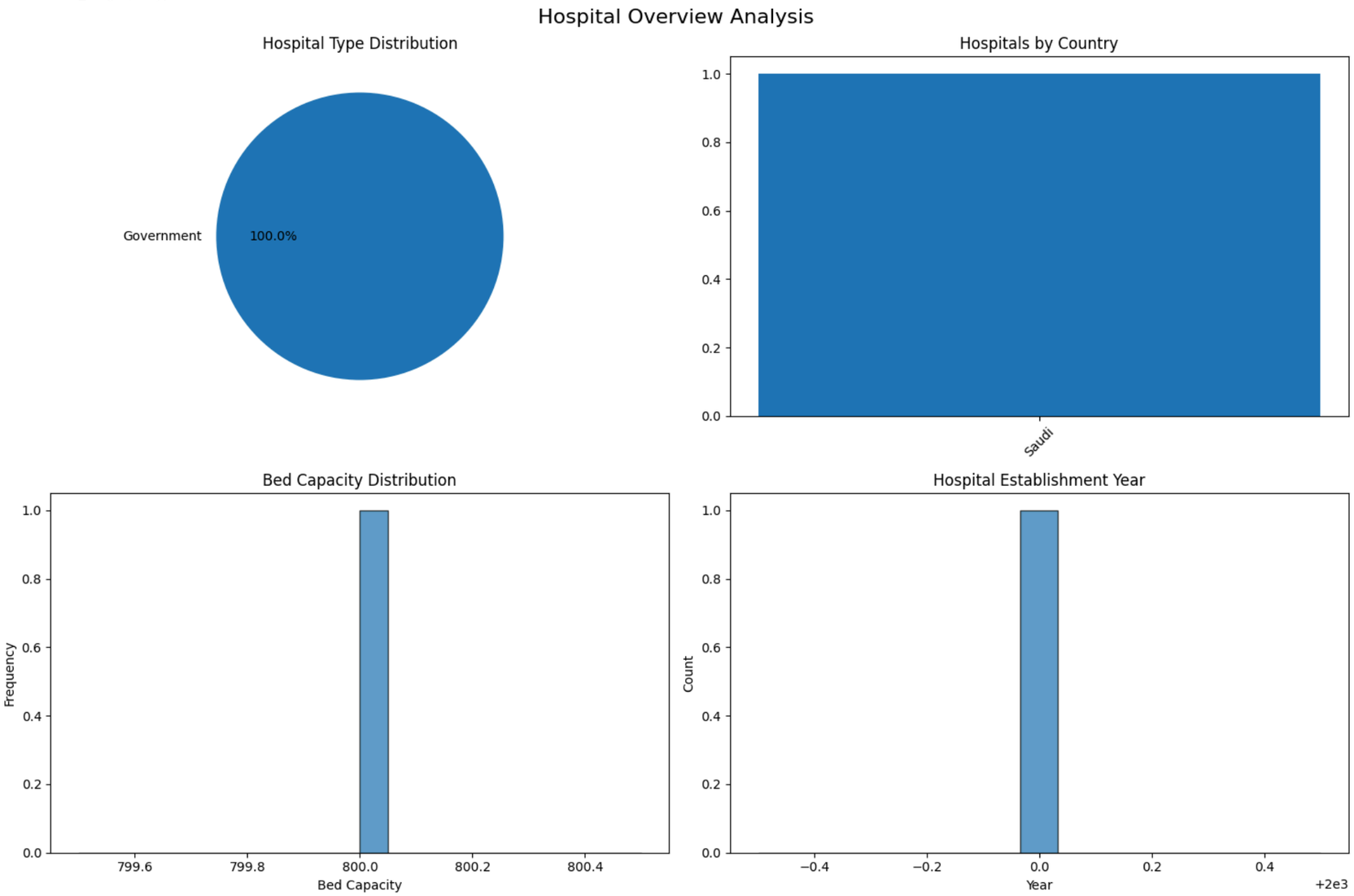
```
Hospital Data EDA System
Choose an option:
1. Quick Preview
2. Complete EDA
File: /content/drive/MyDrive/CAPSTONE/Data/Hospital_Dataset.xlsx
Sheets: ['Hospital', 'Departments', 'Physicians', 'SKUs', 'Vendors']

Sheet: Hospital
Size: (1, 8)
Columns: ['hospital_id', 'hospital_name', 'hospital_type', 'country', 'city', 'bed_capacity', 'annual_budget', 'establishment_year']
First 3 rows:
  hospital_id hospital_name hospital_type country    city  bed_capacity  \
0        H001          KAMC    Government   Saudi  Riyadh           800

   annual_budget  establishment_year
0      960000000                2000
--------------------------------------------------

Sheet: Departments
Size: (10, 7)
Columns: ['dept_id', 'hospital_id', 'dept_name', 'dept_type', 'bed_count', 'monthly_budget', 'head_physician_id']
First 3 rows:
  dept_id hospital_id           dept_name dept_type  bed_count  \
0    D001        H001  Emergency Medicine  Clinical         80
1    D002        H001   Internal Medicine  Clinical        144
2    D003        H001          Cardiology  Clinical         96

   monthly_budget head_physician_id
0         1440000             P0002
1         1728000             P0011
2         2112000             P0019
--------------------------------------------------

Sheet: Physicians
Size: (120, 11)
Columns: ['physician_id', 'hospital_id', 'primary_dept_id', 'specialty', 'experience_years', 'nationality', 'employment_type', 'prescribing_preference', 'formulary_adherence_score', 'cost_consciousness', 'prescription_volume']
First 3 rows:
  physician_id hospital_id primary_dept_id           specialty  \
0        P0001        H001            D001  Emergency Medicine
1        P0002        H001            D001  Emergency Medicine
2        P0003        H001            D001  Emergency Medicine

   experience_years nationality employment_type prescribing_preference  \
0                 3      Indian      Specialist                Generic
1                26       Saudi      Consultant                Generic
2                 4      Indian      Specialist                  Brand

   formulary_adherence_score cost_consciousness prescription_volume
0                       0.90              High                High
1                       0.67               Low              Medium
2                       0.76              High                 Low
--------------------------------------------------

Sheet: SKUs
Size: (1197, 17)
Columns: ['sku_id', 'sku_name', 'category', 'sub_category', 'therapeutic_area', 'brand_generic_flag', 'unit_of_measure', 'standard_cost', 'shelf_life_days', 'storage_requirements', 'controlled_substance_flag', 'halal_certified', 'base_drug_name', 'consumption_volatility', 'c
First 3 rows:
     sku_id                              sku_name  category  \
0  SKU00001      Gliclazide 25mg Injection (Brand)  Pharmacy
1  SKU00002  Insulin Glargine 25mg Syrup (Generic)  Pharmacy
2  SKU00003       Gliclazide 100mg Cream (Generic)  Pharmacy

          sub_category     therapeutic_area brand_generic_flag  \
0  Diabetes Management  Diabetes Management              Brand
1  Diabetes Management  Diabetes Management            Generic
2  Diabetes Management  Diabetes Management            Generic

  unit_of_measure  standard_cost  shelf_life_days storage_requirements  \
0            Pack         222.98             1223              Freezer
1            Pack          26.85             1161         Refrigerated
2            Pack           6.21             1171              Freezer

   controlled_substance_flag  halal_certified    base_drug_name  \
0                      False             True        Gliclazide
1                      False             True  Insulin Glargine
2                      False             True        Gliclazide

   consumption_volatility criticality_level formulary_status  \
0                    0.15            Medium       Restricted
1                    0.15          Standard    Non-Preferred
2                    0.15          Standard        Preferred

   price_sensitivity
0           0.659908
1           0.606927
2           0.158763
--------------------------------------------------
```

```
Sheet: Vendors
Size: (53, 14)
Columns: ['vendor_id', 'vendor_type', 'performance_category', 'specialization', 'price_volatility', 'delivery_reliability', 'country_of_origin', 'average_lead_time_days', 'lead_time_variance', 'fill_rate_percentage', 'quality_rating', 'payment_terms', 'established_year', 'ma
First 3 rows:
  vendor_id vendor_type performance_category specialization  price_volatility  \
0      V001       Tier1            excellent       general              0.05
1      V002       Tier1            excellent      surgical              0.05
2      V003       Tier1                 good      pharmacy              0.10

   delivery_reliability country_of_origin  average_lead_time_days  \
0                  0.95             Saudi                       9
1                  0.95             Saudi                      10
2                  0.88             Saudi                      13

   lead_time_variance  fill_rate_percentage  quality_rating  payment_terms  \
0                   5                  96.1             4.6             60
1                   6                  96.1             4.8             60
2                   8                  91.0             4.1             30

   established_year  market_share_percent
0              1994                  9.84
1              1991                 11.62
2              1996                 11.21
--------------------------------------------------
Loading data from: /content/drive/MyDrive/CAPSTONE/Data/Hospital_Dataset.xlsx
Available sheets: ['Hospital', 'Departments', 'Physicians', 'SKUs', 'Vendors']
Loaded sheet 'Hospital' with shape: (1, 8)
Loaded sheet 'Departments' with shape: (10, 7)
Loaded sheet 'Physicians' with shape: (120, 11)
Loaded sheet 'SKUs' with shape: (1197, 17)
Loaded sheet 'Vendors' with shape: (53, 14)
STARTING COMPREHENSIVE HOSPITAL DATA EDA
================================================================================

================================================================================
BASIC DATA INFORMATION
================================================================================

SHEET: HOSPITAL
--------------------------------------------------
Shape: (1, 8)
Columns: ['hospital_id', 'hospital_name', 'hospital_type', 'country', 'city', 'bed_capacity', 'annual_budget', 'establishment_year']

Data Types:
hospital_id           object
hospital_name         object
hospital_type         object
country               object
city                  object
bed_capacity           int64
annual_budget          int64
establishment_year     int64
dtype: object

Missing Values:
No missing values found!

First 3 rows:
  hospital_id hospital_name hospital_type country    city  bed_capacity  \
0        H001          KAMC    Government   Saudi  Riyadh           800

   annual_budget  establishment_year
0      960000000                2000

--------------------------------------------------

SHEET: DEPARTMENTS
--------------------------------------------------
Shape: (10, 7)
Columns: ['dept_id', 'hospital_id', 'dept_name', 'dept_type', 'bed_count', 'monthly_budget', 'head_physician_id']

Data Types:
dept_id              object
hospital_id          object
dept_name            object
dept_type            object
bed_count             int64
monthly_budget        int64
head_physician_id    object
dtype: object

Missing Values:
No missing values found!

First 3 rows:
  dept_id hospital_id           dept_name dept_type  bed_count  \
```

```
  dept_id hospital_id           dept_name dept_type  bed_count  \
0    D001        H001  Emergency Medicine  Clinical         80
1    D002        H001   Internal Medicine  Clinical        144
2    D003        H001          Cardiology  Clinical         96

   monthly_budget head_physician_id
0         1440000             P0002
1         1728000             P0011
2         2112000             P0019

--------------------------------------------------

SHEET: PHYSICIANS
--------------------------------------------------
Shape: (120, 11)
Columns: ['physician_id', 'hospital_id', 'primary_dept_id', 'specialty', 'experience_years', 'nationality', 'employment_type', 'prescribing_preference', 'formulary_adherence_score', 'cost_consciousness', 'prescription_volume']

Data Types:
physician_id                object
hospital_id                 object
primary_dept_id             object
specialty                   object
experience_years             int64
nationality                 object
employment_type             object
prescribing_preference      object
formulary_adherence_score  float64
cost_consciousness          object
prescription_volume         object
dtype: object

Missing Values:
No missing values found!

First 3 rows:
  physician_id hospital_id primary_dept_id          specialty  \
0        P0001        H001            D001  Emergency Medicine
1        P0002        H001            D001  Emergency Medicine
2        P0003        H001            D001  Emergency Medicine

   experience_years nationality employment_type prescribing_preference  \
0                 3      Indian      Specialist                Generic
1                26       Saudi      Consultant                Generic
2                 4      Indian      Specialist                  Brand

   formulary_adherence_score cost_consciousness prescription_volume
0                       0.90               High                High
1                       0.67                Low              Medium
2                       0.76               High                 Low

--------------------------------------------------

SHEET: SKUS
--------------------------------------------------
Shape: (1197, 17)
Columns: ['sku_id', 'sku_name', 'category', 'sub_category', 'therapeutic_area', 'brand_generic_flag', 'unit_of_measure', 'standard_cost', 'shelf_life_days', 'storage_requirements', 'controlled_substance_flag', 'halal_certified', 'base_drug_name', 'consumption_volatility', 'c

Data Types:
sku_id                       object
sku_name                     object
category                     object
sub_category                 object
therapeutic_area             object
brand_generic_flag           object
unit_of_measure              object
standard_cost               float64
shelf_life_days               int64
storage_requirements         object
controlled_substance_flag       bool
halal_certified                 bool
base_drug_name               object
consumption_volatility      float64
criticality_level            object
formulary_status             object
price_sensitivity           float64
dtype: object

Missing Values:
therapeutic_area      420
brand_generic_flag    420
base_drug_name        420
formulary_status      420
dtype: int64

First 3 rows:
     sku_id                              sku_name  category  \
0  SKU00001      Gliclazide 25mg Injection (Brand)  Pharmacy
1  SKU00002  Insulin Glargine 25mg Syrup (Generic)  Pharmacy
```

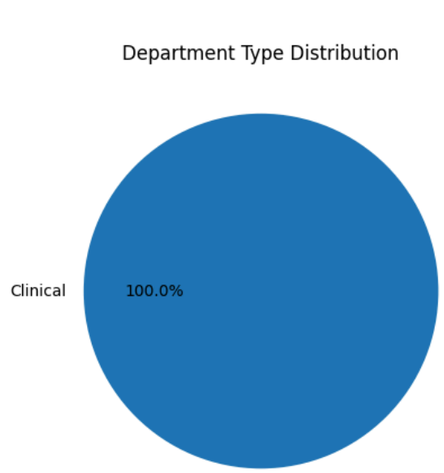
```
2  SKU00003       Gliclazide 100mg Cream (Generic)  Pharmacy

          sub_category     therapeutic_area brand_generic_flag  \
0  Diabetes Management  Diabetes Management              Brand
1  Diabetes Management  Diabetes Management            Generic
2  Diabetes Management  Diabetes Management            Generic

  unit_of_measure  standard_cost  shelf_life_days storage_requirements  \
0            Pack         222.98             1223              Freezer
1            Pack          26.85             1161         Refrigerated
2            Pack           6.21             1171              Freezer

   controlled_substance_flag  halal_certified    base_drug_name  \
0                      False             True        Gliclazide
1                      False             True  Insulin Glargine
2                      False             True        Gliclazide

   consumption_volatility criticality_level formulary_status  \
0                    0.15            Medium       Restricted
1                    0.15          Standard    Non-Preferred
2                    0.15          Standard        Preferred

   price_sensitivity
0           0.659908
1           0.606927
2           0.158763

--------------------------------------------------

SHEET: VENDORS
--------------------------------------------------
Shape: (53, 14)
Columns: ['vendor_id', 'vendor_type', 'performance_category', 'specialization', 'price_volatility', 'delivery_reliability', 'country_of_origin', 'average_lead_time_days', 'lead_time_variance', 'fill_rate_percentage', 'quality_rating', 'payment_terms', 'established_year', 'ma

Data Types:
vendor_id                object
vendor_type              object
performance_category     object
specialization           object
price_volatility        float64
delivery_reliability    float64
country_of_origin        object
average_lead_time_days    int64
lead_time_variance        int64
fill_rate_percentage    float64
quality_rating          float64
payment_terms             int64
established_year          int64
market_share_percent    float64
dtype: object

Missing Values:
No missing values found!

First 3 rows:
  vendor_id vendor_type performance_category specialization  price_volatility  \
0      V001       Tier1            excellent        general              0.05
1      V002       Tier1            excellent       surgical              0.05
2      V003       Tier1                 good       pharmacy              0.10

   delivery_reliability country_of_origin  average_lead_time_days  \
0                  0.95             Saudi                       9
1                  0.95             Saudi                      10
2                  0.88             Saudi                      13

   lead_time_variance  fill_rate_percentage  quality_rating  payment_terms  \
0                   5                  96.1             4.6             60
1                   6                  96.1             4.8             60
2                   8                  91.0             4.1             30

   established_year  market_share_percent
0              1994                  9.84
1              1991                 11.62
2              1996                 11.21

--------------------------------------------------

================================================================================
HOSPITAL ANALYSIS
================================================================================

Bed Capacity Statistics:
count      1.0
mean     800.0
std        NaN
min      800.0
25%      800.0
```

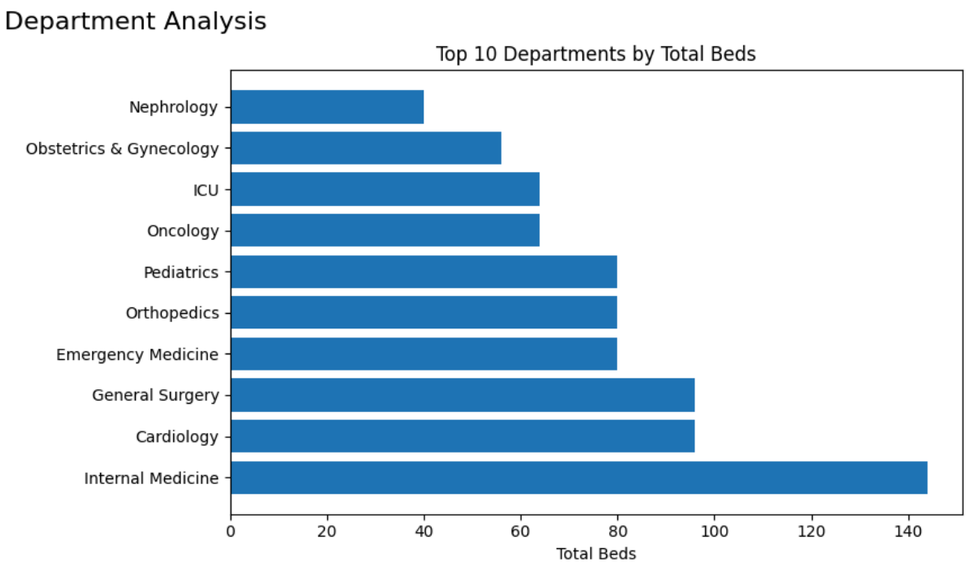
```
50%      800.0
75%      800.0
max      800.0
Name: bed_capacity, dtype: float64

Annual Budget Statistics:
count           1.0
mean     960000000.0
std              NaN
min      960000000.0
25%      960000000.0
50%      960000000.0
75%      960000000.0
max      960000000.0
Name: annual_budget, dtype: float64
```

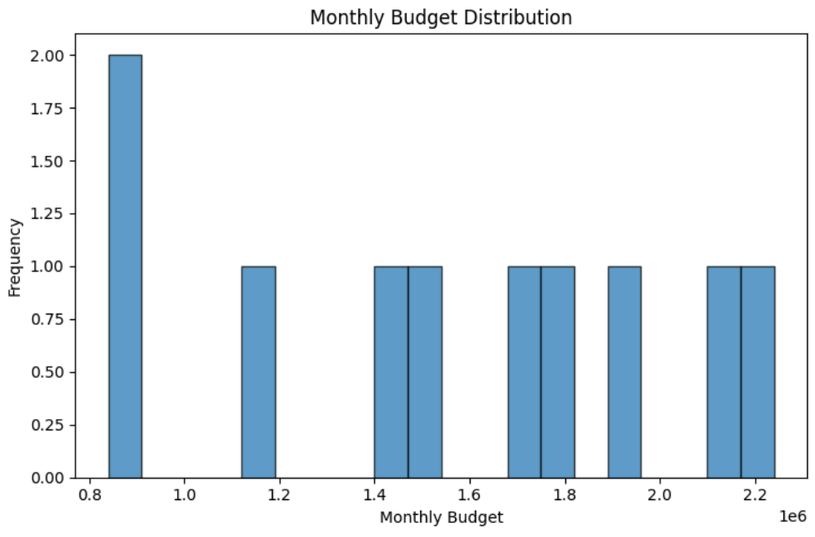


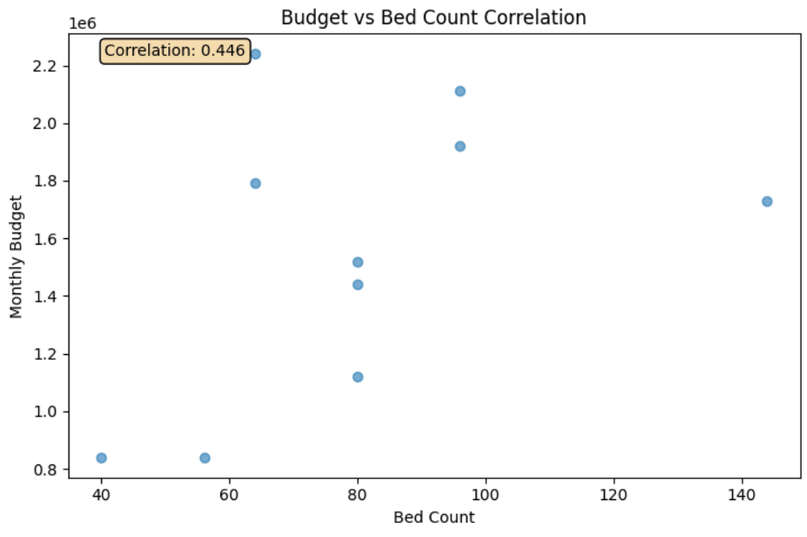

```
============================================================================
DEPARTMENT ANALYSIS
============================================================================

Total Departments: 10
Department Distribution:
dept_name
Emergency Medicine         1
Internal Medicine          1
Cardiology                 1
Orthopedics                1
ICU                        1
General Surgery            1
Pediatrics                 1
```

```
Oncology                  1
Obstetrics & Gynecology   1
Nephrology                1
Name: count, dtype: int64

Bed Count Statistics by Department:
                          sum   mean  count
dept_name
Cardiology                 96   96.0      1
Emergency Medicine         80   80.0      1
General Surgery            96   96.0      1
ICU                        64   64.0      1
Internal Medicine         144  144.0      1
Nephrology                 40   40.0      1
Obstetrics & Gynecology    56   56.0      1
Oncology                   64   64.0      1
Orthopedics                80   80.0      1
Pediatrics                 80   80.0      1
```



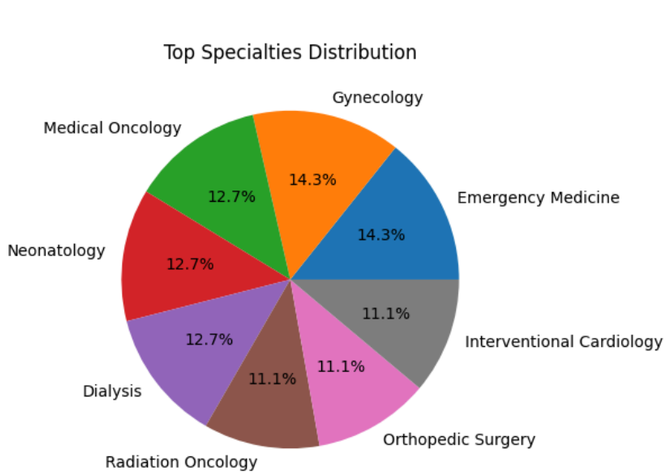


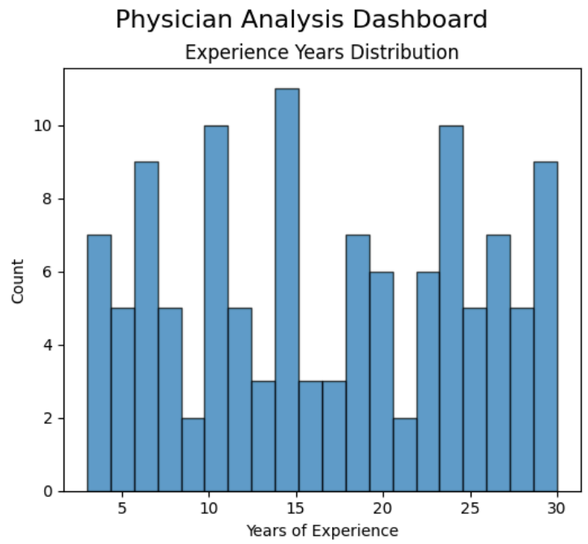


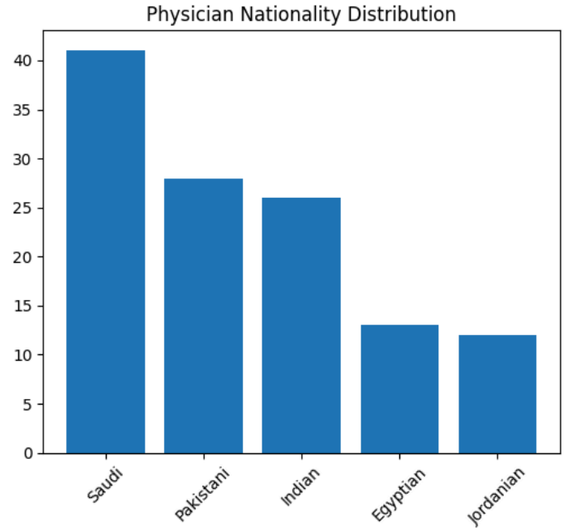


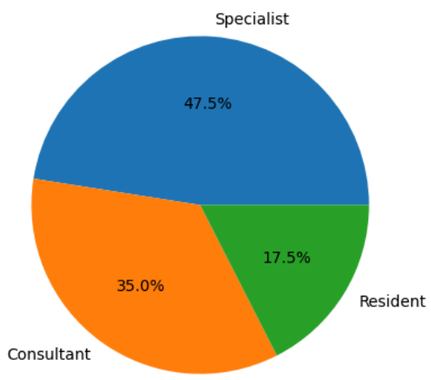

```
================================================================================
PHYSICIAN ANALYSIS
================================================================================

Total Physicians: 120
Specialties: 19

Top 10 Specialties:
specialty
Emergency Medicine          9
Gynecology                  9
```

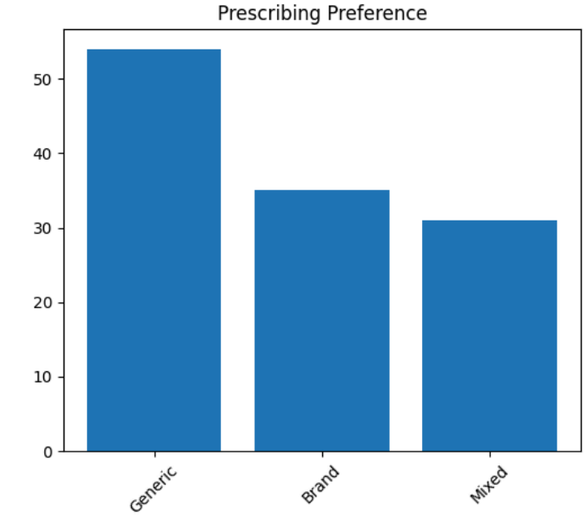
```
Medical Oncology            8
Neonatology                 8
Dialysis                    8
Radiation Oncology          7
Orthopedic Surgery          7
Interventional Cardiology   7
Nephrology                  7
General Surgery             7
Name: count, dtype: int64

Experience Statistics:
count    120.000000
mean      16.750000
std        8.142894
min        3.000000
25%       10.000000
50%       16.500000
75%       24.000000
max       30.000000
Name: experience_years, dtype: float64
```

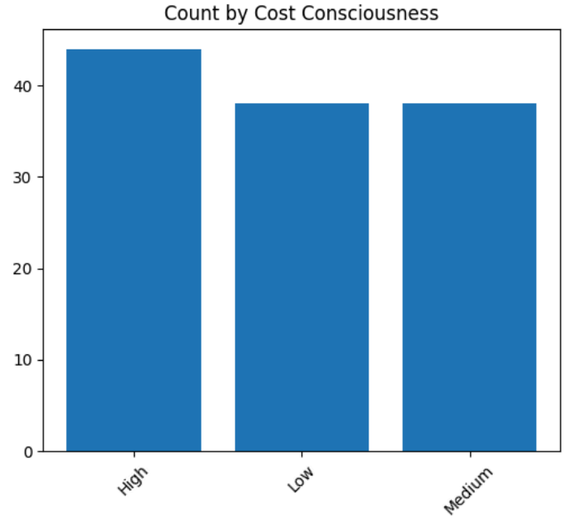






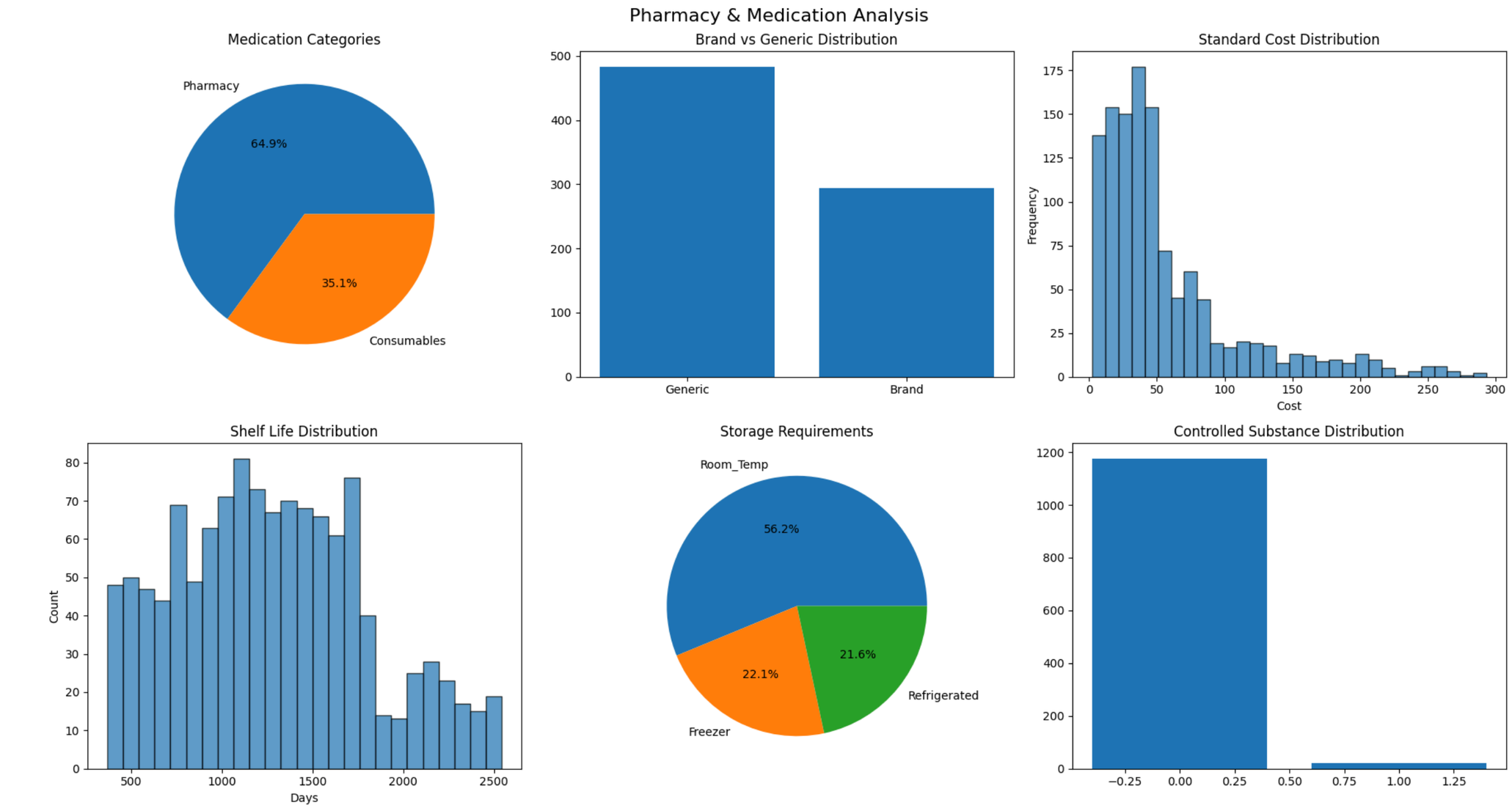






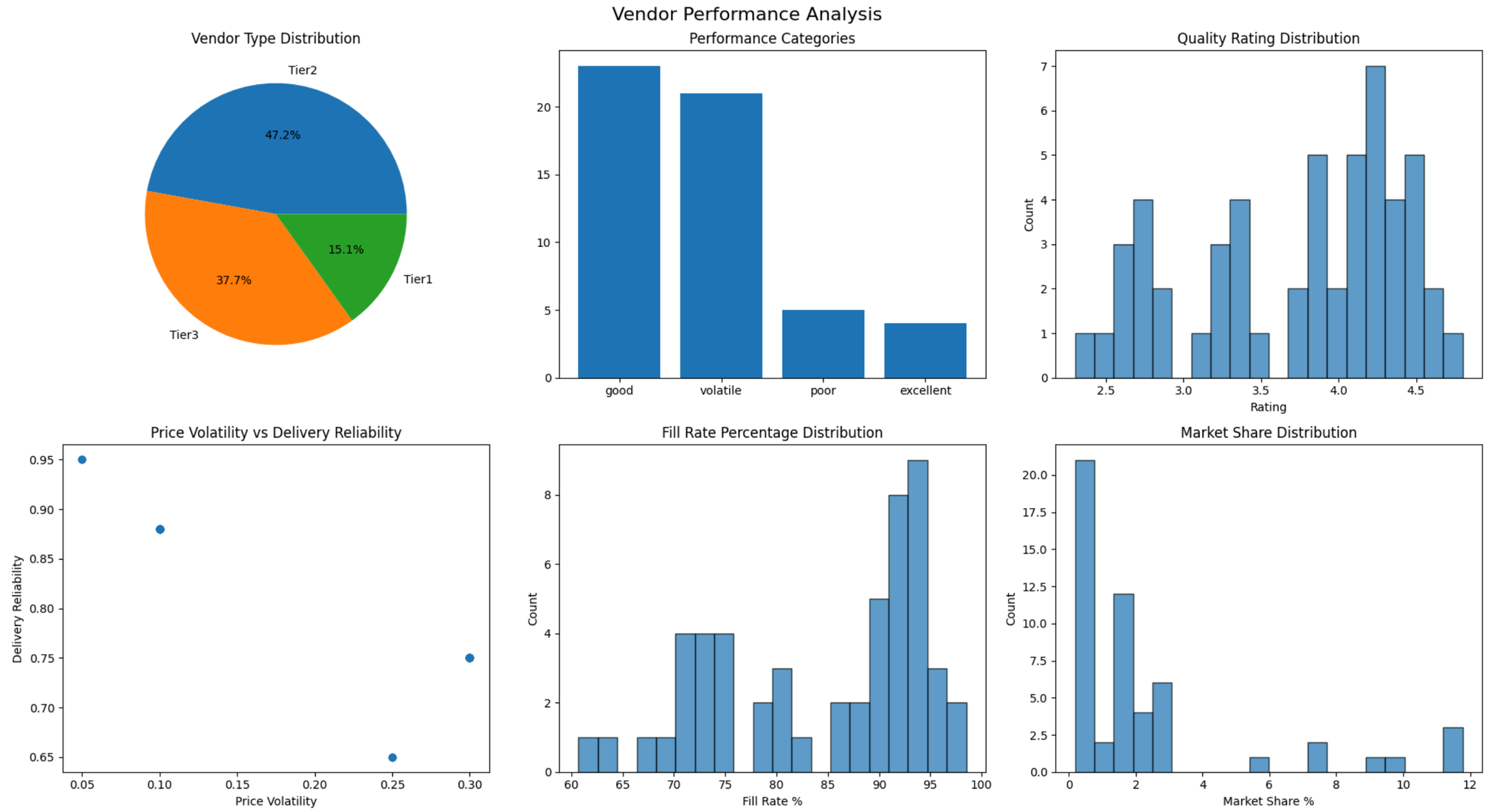

```
================================================================================
PHARMACY ANALYSIS
================================================================================

Total SKUs: 1197
Categories: 2
category
Pharmacy       777
Consumables    420
Name: count, dtype: int64
```

```
Cost Statistics:
count    1197.000000
mean       57.005856
std        53.689199
min         2.100000
25%        22.200000
50%        39.760000
75%        71.510000
max       293.530000
Name: standard_cost, dtype: float64
```

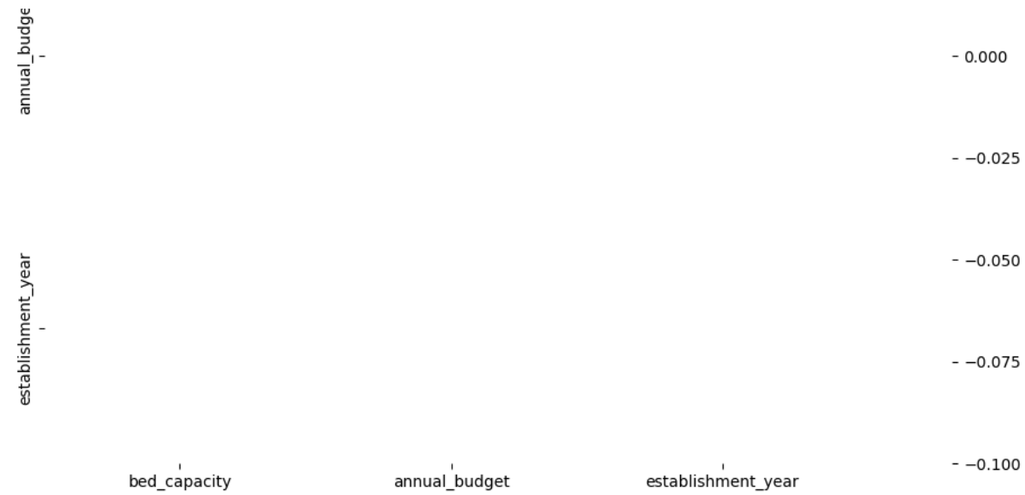



```
================================================================================
VENDOR ANALYSIS
================================================================================

Total Vendors: 53
Vendor Types:
vendor_type
Tier2    25
Tier3    20
Tier1     8
Name: count, dtype: int64

Market Share Statistics:
count    53.000000
mean      2.487925
std       3.113223
min       0.180000
25%       0.630000
```

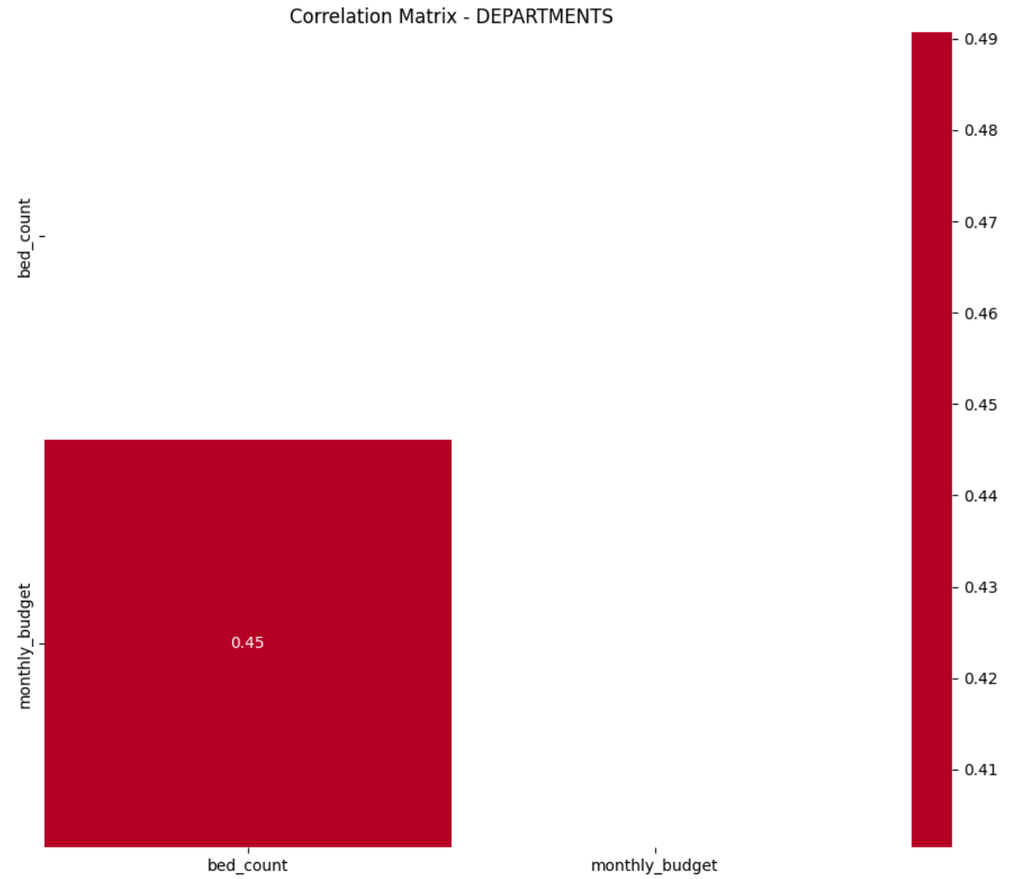
```
50%       1.600000
75%       2.500000
max      11.780000
Name: market_share_percent, dtype: float64
```



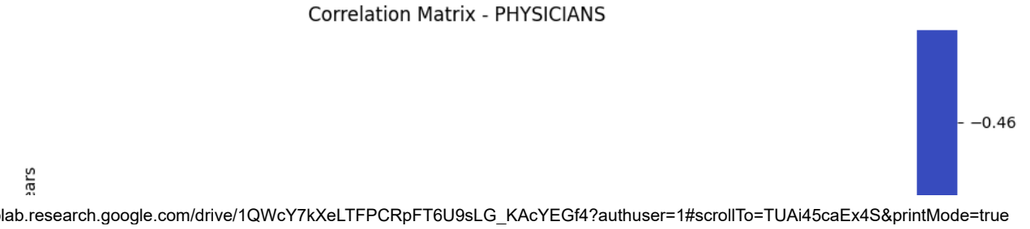

```
================================================================================
CORRELATION ANALYSIS
================================================================================

Correlations for HOSPITAL:
```

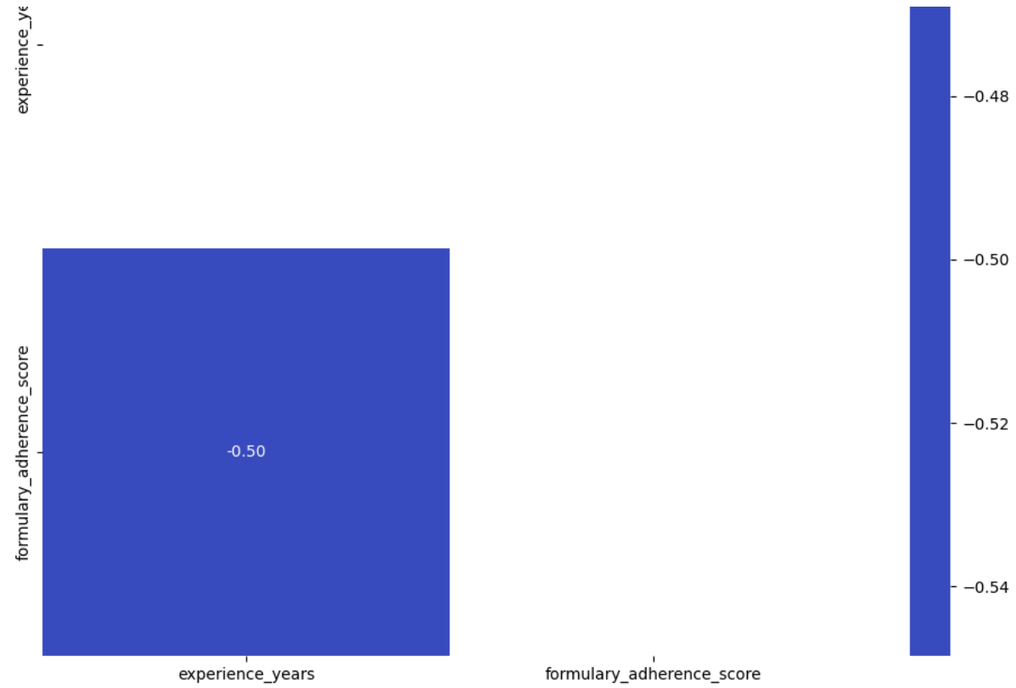



```
Strongest correlations found:

Correlations for DEPARTMENTS:
```

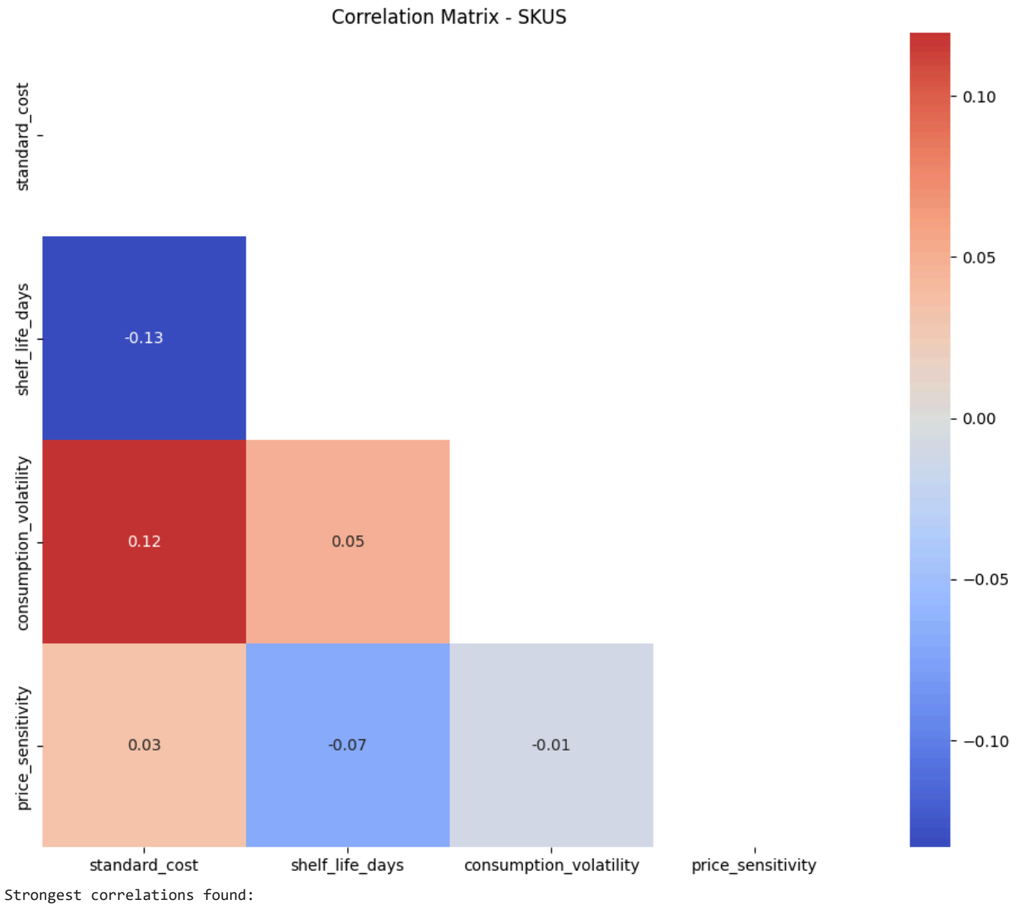



```
Strongest correlations found:

Correlations for PHYSICIANS:
```






```
Strongest correlations found:

Correlations for SKUS:
```




```
Strongest correlations found:

Correlations for VENDORS:
```

```
Strongest correlations found:
  price_volatility <-> delivery_reliability: -0.891
  price_volatility <-> average_lead_time_days: 0.616
  price_volatility <-> fill_rate_percentage: -0.811
  price_volatility <-> quality_rating: -0.803
  price_volatility <-> market_share_percent: -0.588
  delivery_reliability <-> average_lead_time_days: -0.733
  delivery_reliability <-> fill_rate_percentage: 0.887
  delivery_reliability <-> quality_rating: 0.869
  delivery_reliability <-> market_share_percent: 0.623
  average_lead_time_days <-> fill_rate_percentage: -0.581
  average_lead_time_days <-> quality_rating: -0.636
  fill_rate_percentage <-> quality_rating: 0.830
  fill_rate_percentage <-> market_share_percent: 0.559
  quality_rating <-> market_share_percent: 0.550

================================================================================
KEY BUSINESS INSIGHTS
================================================================================
1. Average hospital bed capacity: 800 beds
2. Total annual budget across hospitals: $960,000,000
3. Average physician experience: 16.8 years

EDA COMPLETE!
================================================================================

To run the analysis:
1. Update 'file_path' variable with your Excel file location
2. Uncomment either preview_data(file_path) or run_eda(file_path)
3. Run the script
```

Patient

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from datetime import datetime, timedelta
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Set style and suppress warnings
plt.style.use('default')
warnings.filterwarnings('ignore')
sns.set_palette("husl")

class PatientDataEDA:
    def __init__(self, file_path):
        """Initialize the Patient EDA class with the Excel file path"""
        self.file_path = file_path
        self.sheets_data = {}
        self.load_data()

    def load_data(self):
        """Load all sheets from the Excel file"""
        try:
            excel_file = pd.ExcelFile(self.file_path)
            print(f"📊 Loading patient data from: {self.file_path}")
            print(f"📋 Available sheets: {excel_file.sheet_names}")

            for sheet_name in excel_file.sheet_names:
                self.sheets_data[sheet_name] = pd.read_excel(self.file_path, sheet_name=sheet_name)
                print(f"✅ Loaded sheet '{sheet_name}' with shape: {self.sheets_data[sheet_name].shape}")

        except Exception as e:
            print(f"❌ Error loading data: {str(e)}")

    def preprocess_data(self):
        """Preprocess patient data - handle dates, categorize age groups, etc."""
        print("\n" + "="*80)
        print("🔄 PREPROCESSING PATIENT DATA")
        print("="*80)

        for sheet_name, df in self.sheets_data.items():
            print(f"\n📊 Processing sheet: {sheet_name}")

            # Convert date columns
            date_columns = ['admission_date', 'discharge_date']
            for col in date_columns:
                if col in df.columns:
                    try:
                        df[col] = pd.to_datetime(df[col])
                        print(f"✅ Converted {col} to datetime")
                    except:
                        print(f"⚠️ Could not convert {col} to datetime")

            # Calculate length of stay if we have both dates
            if 'admission_date' in df.columns and 'discharge_date' in df.columns:
                df['length_of_stay'] = (df['discharge_date'] - df['admission_date']).dt.days
                print("✅ Calculated length_of_stay")

            # Categorize age groups if age_group is not already categorical
            if 'age_group' in df.columns:
                print(f"✅ Age groups found: {df['age_group'].unique()}")

            # Extract admission year/month for trend analysis
            if 'admission_date' in df.columns:
                df['admission_year'] = df['admission_date'].dt.year
                df['admission_month'] = df['admission_date'].dt.month
                df['admission_day_of_week'] = df['admission_date'].dt.day_name()
                print("✅ Created time-based features")

            self.sheets_data[sheet_name] = df

    def basic_info(self):
```

```
        """Display basic information about all datasets"""
        print("\n" + "="*80)
        print("📈 BASIC PATIENT DATA INFORMATION")
        print("="*80)

        for sheet_name, df in self.sheets_data.items():
            print(f"\n📋 SHEET: {sheet_name.upper()}")
            print("-" * 50)
            print(f"Shape: {df.shape}")
            print(f"Columns: {list(df.columns)}")
            print(f"\nData Types:")
            print(df.dtypes)

            print(f"\nMissing Values:")
            missing = df.isnull().sum()
            if missing.sum() > 0:
                missing_df = pd.DataFrame({
                    'Column': missing.index,
                    'Missing_Count': missing.values,
                    'Missing_Percentage': (missing.values / len(df) * 100).round(2)
                })
                missing_df = missing_df[missing_df['Missing_Count'] > 0]
                print(missing_df.to_string(index=False))
            else:
                print("No missing values found!")

            print(f"\nSample Data (first 3 rows):")
            print(df.head(3).to_string())
            print("\n" + "-"*50)

    def patient_demographics_analysis(self):
        """Analyze patient demographics"""
        patient_df = None

        # Find patient data
        for sheet_name, df in self.sheets_data.items():
            if 'patient_id' in df.columns:
                patient_df = df
                break

        if patient_df is None:
            print("❌ Patient data not found")
            return

        print("\n" + "="*80)
        print("👥 PATIENT DEMOGRAPHICS ANALYSIS")
        print("="*80)

        # Basic statistics
        print(f"\n📊 Total Patients: {len(patient_df):,}")

        if 'age_group' in patient_df.columns:
            print(f"\n👶 Age Group Distribution:")
            age_dist = patient_df['age_group'].value_counts().sort_index()
            print(age_dist)

        if 'patient_type' in patient_df.columns:
            print(f"\n🏥 Patient Type Distribution:")
            print(patient_df['patient_type'].value_counts())

        if 'socioeconomic_level' in patient_df.columns:
            print(f"\n💰 Socioeconomic Level Distribution:")
            print(patient_df['socioeconomic_level'].value_counts())

        # Visualizations
        fig, axes = plt.subplots(2, 3, figsize=(18, 12))
        fig.suptitle('Patient Demographics Analysis', fontsize=16, fontweight='bold')

        # Age group distribution
        if 'age_group' in patient_df.columns:
            age_counts = patient_df['age_group'].value_counts()
            axes[0, 0].pie(age_counts.values, labels=age_counts.index, autopct='%1.1f%%')
            axes[0, 0].set_title('Age Group Distribution')

        # Patient type distribution
        if 'patient_type' in patient_df.columns:
            type_counts = patient_df['patient_type'].value_counts()
            axes[0, 1].bar(type_counts.index, type_counts.values, color=['skyblue', 'lightcoral', 'lightgreen'])
            axes[0, 1].set_title('Patient Type Distribution')
```

```
            axes[0, 1].tick_params(axis='x', rotation=45)

        # Insurance type analysis
        if 'insurance_type' in patient_df.columns:
            insurance_counts = patient_df['insurance_type'].value_counts()
            axes[0, 2].pie(insurance_counts.values, labels=insurance_counts.index, autopct='%1.1f%%')
            axes[0, 2].set_title('Insurance Type Distribution')

        # Socioeconomic level
        if 'socioeconomic_level' in patient_df.columns:
            socio_counts = patient_df['socioeconomic_level'].value_counts()
            # Use valid matplotlib colors
            axes[1, 0].bar(socio_counts.index, socio_counts.values, color=['gold', 'silver', 'sienna'])
            axes[1, 0].set_title('Socioeconomic Level Distribution')

        # Chronic conditions analysis
        if 'chronic_conditions' in patient_df.columns:
            chronic_counts = patient_df['chronic_conditions'].value_counts()
            axes[1, 1].pie(chronic_counts.values, labels=chronic_counts.index, autopct='%1.1f%%')
            axes[1, 1].set_title('Chronic Conditions Distribution')

        # Length of stay distribution
        if 'length_of_stay' in patient_df.columns:
            los_data = patient_df['length_of_stay'].dropna()
            axes[1, 2].hist(los_data, bins=30, edgecolor='black', alpha=0.7, color='lightblue')
            axes[1, 2].set_title('Length of Stay Distribution')
            axes[1, 2].set_xlabel('Days')
            axes[1, 2].set_ylabel('Count')

            # Add statistics
            mean_los = los_data.mean()
            median_los = los_data.median()
            axes[1, 2].axvline(mean_los, color='red', linestyle='--', label=f'Mean: {mean_los:.1f}')
            axes[1, 2].axvline(median_los, color='green', linestyle='--', label=f'Median: {median_los:.1f}')
            axes[1, 2].legend()

        plt.tight_layout()
        plt.show()

    def admission_patterns_analysis(self):
        """Analyze admission patterns and trends"""
        patient_df = None

        for sheet_name, df in self.sheets_data.items():
            if 'admission_date' in df.columns:
                patient_df = df
                break

        if patient_df is None:
            print("❌ Admission data not found")
            return

        print("\n" + "="*80)
        print("📅 ADMISSION PATTERNS ANALYSIS")
        print("="*80)

        # Time-based statistics
        if 'admission_year' in patient_df.columns:
            print(f"\n📈 Admissions by Year:")
            yearly_admissions = patient_df['admission_year'].value_counts().sort_index()
            print(yearly_admissions)

        if 'admission_day_of_week' in patient_df.columns:
            print(f"\n📅 Admissions by Day of Week:")
            daily_admissions = patient_df['admission_day_of_week'].value_counts()
            print(daily_admissions)

        # Visualizations
        fig, axes = plt.subplots(2, 2, figsize=(16, 12))
        fig.suptitle('Admission Patterns Analysis', fontsize=16, fontweight='bold')

        # Monthly admission trends
        if 'admission_month' in patient_df.columns:
            monthly_counts = patient_df['admission_month'].value_counts().sort_index()
            month_names = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                           'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
            axes[0, 0].plot(monthly_counts.index, monthly_counts.values, marker='o', linewidth=2, markersize=8)
            axes[0, 0].set_title('Monthly Admission Trends')
            axes[0, 0].set_xlabel('Month')
```

```
        axes[0, 0].set_ylabel('Number of Admissions')
        axes[0, 0].set_xticks(range(1, 13))
        axes[0, 0].set_xticklabels(month_names, rotation=45)
        axes[0, 0].grid(True, alpha=0.3)

    # Day of week patterns
    if 'admission_day_of_week' in patient_df.columns:
        day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
        day_counts = patient_df['admission_day_of_week'].value_counts().reindex(day_order, fill_value=0)
        axes[0, 1].bar(day_counts.index, day_counts.values, color='lightcoral')
        axes[0, 1].set_title('Admissions by Day of Week')
        axes[0, 1].set_ylabel('Number of Admissions')
        plt.setp(axes[0, 1].xaxis.get_majorticklabels(), rotation=45)

    # Patient type by age group
    if 'age_group' in patient_df.columns and 'patient_type' in patient_df.columns:
        type_age_crosstab = pd.crosstab(patient_df['age_group'], patient_df['patient_type'])
        type_age_crosstab.plot(kind='bar', stacked=True, ax=axes[1, 0])
        axes[1, 0].set_title('Patient Type by Age Group')
        axes[1, 0].set_ylabel('Count')
        axes[1, 0].legend(title='Patient Type')
        plt.setp(axes[1, 0].xaxis.get_majorticklabels(), rotation=45)

    # Complexity score distribution
    if 'complexity_score' in patient_df.columns:
        axes[1, 1].hist(patient_df['complexity_score'], bins=25, edgecolor='black', alpha=0.7, color='lightgreen')
        axes[1, 1].set_title('Case Complexity Score Distribution')
        axes[1, 1].set_xlabel('Complexity Score')
        axes[1, 1].set_ylabel('Frequency')

        # Add mean and median lines
        mean_complexity = patient_df['complexity_score'].mean()
        median_complexity = patient_df['complexity_score'].median()
        axes[1, 1].axvline(mean_complexity, color='red', linestyle='--', label=f'Mean: {mean_complexity:.2f}')
        axes[1, 1].axvline(median_complexity, color='blue', linestyle='--', label=f'Median: {median_complexity:.2f}')
        axes[1, 1].legend()

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

```
def medical_conditions_analysis(self):
    """Analyze medical conditions and treatment patterns"""
    patient_df = None

    for sheet_name, df in self.sheets_data.items():
        if 'chronic_conditions' in df.columns:
            patient_df = df
            break

    if patient_df is None:
        print("❌ Medical conditions data not found")
        return

    print("\n" + "="*80)
    print("🏥 MEDICAL CONDITIONS ANALYSIS")
    print("="*80)

    # Chronic conditions analysis
    if 'chronic_conditions' in patient_df.columns:
        print(f"\n🩺 Chronic Conditions Distribution:")
        chronic_dist = patient_df['chronic_conditions'].value_counts()
        print(chronic_dist)

        # Calculate percentage with chronic conditions
        total_patients = len(patient_df)
        patients_with_chronic = len(patient_df[patient_df['chronic_conditions'] != 'None'])
        chronic_percentage = (patients_with_chronic / total_patients) * 100
        print(f"\n📊 Patients with chronic conditions: {patients_with_chronic:,} ({chronic_percentage:.1f}%)")

    # Primary department analysis
    if 'primary_dept_id' in patient_df.columns:
        print(f"\n🏥 Primary Department Distribution:")
        dept_dist = patient_df['primary_dept_id'].value_counts().head(10)
        print(dept_dist)

    # Visualizations
    fig, axes = plt.subplots(2, 3, figsize=(18, 12))
    fig.suptitle('Medical Conditions & Treatment Analysis', fontsize=16, fontweight='bold')
```

```
            # Chronic conditions pie chart
            if 'chronic_conditions' in patient_df.columns:
                chronic_counts = patient_df['chronic_conditions'].value_counts()
                colors = ['lightblue', 'lightcoral', 'lightgreen', 'gold', 'plum']
                axes[0, 0].pie(chronic_counts.values, labels=chronic_counts.index, autopct='%1.1f%%', colors=colors[:len(chronic_counts)])
                axes[0, 0].set_title('Chronic Conditions Distribution')

            # Patient type vs chronic conditions
            if 'patient_type' in patient_df.columns and 'chronic_conditions' in patient_df.columns:
                condition_type_crosstab = pd.crosstab(patient_df['chronic_conditions'], patient_df['patient_type'])
                condition_type_crosstab.plot(kind='bar', ax=axes[0, 1])
                axes[0, 1].set_title('Chronic Conditions by Patient Type')
                axes[0, 1].set_ylabel('Count')
                plt.setp(axes[0, 1].xaxis.get_majorticklabels(), rotation=45)

            # Age group vs chronic conditions
            if 'age_group' in patient_df.columns and 'chronic_conditions' in patient_df.columns:
                age_condition_crosstab = pd.crosstab(patient_df['age_group'], patient_df['chronic_conditions'])
                age_condition_crosstab.plot(kind='bar', stacked=True, ax=axes[0, 2])
                axes[0, 2].set_title('Chronic Conditions by Age Group')
                axes[0, 2].set_ylabel('Count')
                plt.setp(axes[0, 2].xaxis.get_majorticklabels(), rotation=45)

            # Complexity score by chronic conditions
            if 'complexity_score' in patient_df.columns and 'chronic_conditions' in patient_df.columns:
                complexity_by_condition = patient_df.groupby('chronic_conditions')['complexity_score'].mean().sort_values(ascending=False)
                axes[1, 0].bar(complexity_by_condition.index, complexity_by_condition.values, color='salmon')
                axes[1, 0].set_title('Average Complexity Score by Condition')
                axes[1, 0].set_ylabel('Average Complexity Score')
                plt.setp(axes[1, 0].xaxis.get_majorticklabels(), rotation=45)

            # Length of stay by patient type
            if 'length_of_stay' in patient_df.columns and 'patient_type' in patient_df.columns:
                patient_df.boxplot(column='length_of_stay', by='patient_type', ax=axes[1, 1])
                axes[1, 1].set_title('Length of Stay by Patient Type')
                axes[1, 1].set_xlabel('Patient Type')
                axes[1, 1].set_ylabel('Length of Stay (Days)')

            # Primary department distribution
            if 'primary_dept_id' in patient_df.columns:
                dept_counts = patient_df['primary_dept_id'].value_counts().head(10)
                axes[1, 2].barh(dept_counts.index.astype(str), dept_counts.values, color='lightsteelblue')
                axes[1, 2].set_title('Top 10 Primary Departments')
                axes[1, 2].set_xlabel('Number of Patients')

        plt.tight_layout()
        plt.show()

    def insurance_and_financial_analysis(self):
        """Analyze insurance patterns and financial aspects"""
        patient_df = None

        for sheet_name, df in self.sheets_data.items():
            if 'insurance_type' in df.columns:
                patient_df = df
                break

        if patient_df is None:
            print("❌ Insurance data not found")
            return

        print("\n" + "="*80)
        print("🏧 INSURANCE & FINANCIAL ANALYSIS")
        print("="*80)

        # Insurance statistics
        if 'insurance_type' in patient_df.columns:
            print(f"\n🏧 Insurance Type Distribution:")
            insurance_dist = patient_df['insurance_type'].value_counts()
            print(insurance_dist)

            # Calculate percentages
            for insurance_type, count in insurance_dist.items():
                percentage = (count / len(patient_df)) * 100
                print(f"   {insurance_type}: {count:,} patients ({percentage:.1f}%)")

        # Socioeconomic analysis
        if 'socioeconomic_level' in patient_df.columns:
            print(f"\n💰 Socioeconomic Distribution:")
```

```
            socio_dist = patient_df['socioeconomic_level'].value_counts()
            print(socio_dist)

        # Visualizations
        fig, axes = plt.subplots(2, 2, figsize=(14, 10))
        fig.suptitle('Insurance & Financial Analysis', fontsize=16, fontweight='bold')

        # Insurance type distribution
        if 'insurance_type' in patient_df.columns:
            insurance_counts = patient_df['insurance_type'].value_counts()
            axes[0, 0].pie(insurance_counts.values, labels=insurance_counts.index, autopct='%1.1f%%')
            axes[0, 0].set_title('Insurance Type Distribution')

        # Insurance vs Patient Type
        if 'insurance_type' in patient_df.columns and 'patient_type' in patient_df.columns:
            insurance_patient_crosstab = pd.crosstab(patient_df['insurance_type'], patient_df['patient_type'])
            insurance_patient_crosstab.plot(kind='bar', ax=axes[0, 1])
            axes[0, 1].set_title('Insurance Type vs Patient Type')
            axes[0, 1].set_ylabel('Count')
            plt.setp(axes[0, 1].xaxis.get_majorticklabels(), rotation=45)

        # Socioeconomic vs Age Group
        if 'socioeconomic_level' in patient_df.columns and 'age_group' in patient_df.columns:
            socio_age_crosstab = pd.crosstab(patient_df['socioeconomic_level'], patient_df['age_group'])
            socio_age_crosstab.plot(kind='bar', stacked=True, ax=axes[1, 0])
            axes[1, 0].set_title('Socioeconomic Level by Age Group')
            axes[1, 0].set_ylabel('Count')
            plt.setp(axes[1, 0].xaxis.get_majorticklabels(), rotation=45)

        # Length of stay by insurance type
        if 'length_of_stay' in patient_df.columns and 'insurance_type' in patient_df.columns:
            patient_df.boxplot(column='length_of_stay', by='insurance_type', ax=axes[1, 1])
            axes[1, 1].set_title('Length of Stay by Insurance Type')
            axes[1, 1].set_xlabel('Insurance Type')
            axes[1, 1].set_ylabel('Length of Stay (Days)')

        plt.tight_layout()
        plt.show()

    def temporal_trends_analysis(self):
        """Analyze trends over time"""
        patient_df = None

        for sheet_name, df in self.sheets_data.items():
            if 'admission_date' in df.columns:
                patient_df = df
                break

        if patient_df is None:
            print("❌ Temporal data not found")
            return

        print("\n" + "="*80)
        print("📈 TEMPORAL TRENDS ANALYSIS")
        print("="*80)

        # Create time-based aggregations
        if 'admission_date' in patient_df.columns:
            # Daily admissions trend
            daily_admissions = patient_df.groupby(patient_df['admission_date'].dt.date).size()

            # Monthly admissions trend
            monthly_admissions = patient_df.groupby(patient_df['admission_date'].dt.to_period('M')).size()

            print(f"📊 Date range: {patient_df['admission_date'].min()} to {patient_df['admission_date'].max()}")
            print(f"📊 Average daily admissions: {daily_admissions.mean():.2f}")
            print(f"📊 Peak admission day: {daily_admissions.idxmax()} ({daily_admissions.max()} admissions)")

        # Visualizations
        fig, axes = plt.subplots(2, 2, figsize=(16, 12))
        fig.suptitle('Temporal Trends Analysis', fontsize=16, fontweight='bold')

        # Daily admission trend
        if 'admission_date' in patient_df.columns:
            daily_admissions.plot(ax=axes[0, 0], color='blue', alpha=0.7)
            axes[0, 0].set_title('Daily Admission Trends')
            axes[0, 0].set_xlabel('Date')
            axes[0, 0].set_ylabel('Number of Admissions')
            axes[0, 0].grid(True, alpha=0.3)
```

```
        # Monthly trend
        if 'admission_date' in patient_df.columns:
            monthly_admissions.plot(kind='bar', ax=axes[0, 1], color='green')
            axes[0, 1].set_title('Monthly Admission Trends')
            axes[0, 1].set_xlabel('Month')
            axes[0, 1].set_ylabel('Number of Admissions')
            plt.setp(axes[0, 1].xaxis.get_majorticklabels(), rotation=45)

        # Complexity score trend over time
        if 'admission_date' in patient_df.columns and 'complexity_score' in patient_df.columns:
            monthly_complexity = patient_df.groupby(patient_df['admission_date'].dt.to_period('M'))['complexity_score'].mean()
            monthly_complexity.plot(ax=axes[1, 0], marker='o', color='red')
            axes[1, 0].set_title('Average Complexity Score Trend')
            axes[1, 0].set_xlabel('Month')
            axes[1, 0].set_ylabel('Average Complexity Score')
            axes[1, 0].grid(True, alpha=0.3)

        # Length of stay trend
        if 'admission_date' in patient_df.columns and 'length_of_stay' in patient_df.columns:
            monthly_los = patient_df.groupby(patient_df['admission_date'].dt.to_period('M'))['length_of_stay'].mean()
            monthly_los.plot(ax=axes[1, 1], marker='s', color='purple')
            axes[1, 1].set_title('Average Length of Stay Trend')
            axes[1, 1].set_xlabel('Month')
            axes[1, 1].set_ylabel('Average Length of Stay (Days)')
            axes[1, 1].grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()

    def correlation_analysis(self):
        """Perform correlation analysis for numerical columns"""
        print("\n" + "="*80)
        print("🔗 CORRELATION ANALYSIS")
        print("="*80)

        for sheet_name, df in self.sheets_data.items():
            print(f"\nCorrelations for {sheet_name.upper()}:")

            # Select only numerical columns
            numerical_cols = df.select_dtypes(include=[np.number]).columns

            if len(numerical_cols) > 1:
                correlation_matrix = df[numerical_cols].corr()

                # Plot correlation heatmap
                plt.figure(figsize=(12, 8))
                mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
                sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='coolwarm',
                           center=0, fmt='.2f', square=True)
                plt.title(f'Correlation Matrix - {sheet_name.upper()}')
                plt.tight_layout()
                plt.show()

                # Show strongest correlations
                print("Strongest correlations (>0.5 or <-0.5):")
                for i in range(len(correlation_matrix.columns)):
                    for j in range(i+1, len(correlation_matrix.columns)):
                        corr_value = correlation_matrix.iloc[i, j]
                        if abs(corr_value) > 0.5:
                            print(f"  {correlation_matrix.columns[i]} <-> {correlation_matrix.columns[j]}: {corr_value:.3f}")
            else:
                print("Not enough numerical columns for correlation analysis")

    def patient_risk_analysis(self):
        """Analyze patient risk factors and outcomes"""
        patient_df = None

        for sheet_name, df in self.sheets_data.items():
            if 'complexity_score' in df.columns:
                patient_df = df
                break

        if patient_df is None:
            print("❌ Risk analysis data not found")
            return

        print("\n" + "="*80)
        print("⚠️ PATIENT RISK ANALYSIS")
```

```
        print("="*80)

        # Risk categorization based on complexity score
        if 'complexity_score' in patient_df.columns:
            # Define risk categories
            patient_df['risk_category'] = pd.cut(patient_df['complexity_score'],
                                                 bins=[0, 1, 1.5, 2, float('inf')],
                                                 labels=['Low', 'Medium', 'High', 'Critical'])

            print(f"\n⚠️ Risk Category Distribution:")
            risk_dist = patient_df['risk_category'].value_counts()
            print(risk_dist)

        # High-risk patient analysis
        if 'risk_category' in patient_df.columns:
            high_risk_patients = patient_df[patient_df['risk_category'].isin(['High', 'Critical'])]
            print(f"\n🚨 High-risk patients: {len(high_risk_patients):,} ({len(high_risk_patients)/len(patient_df)*100:.1f}%)")

            if len(high_risk_patients) > 0:
                print("\nHigh-risk patient characteristics:")
                if 'age_group' in high_risk_patients.columns:
                    print("Age groups:", high_risk_patients['age_group'].value_counts().to_dict())
                if 'chronic_conditions' in high_risk_patients.columns:
                    print("Chronic conditions:", high_risk_patients['chronic_conditions'].value_counts().to_dict())

        # Visualizations
        fig, axes = plt.subplots(2, 2, figsize=(14, 10))
        fig.suptitle('Patient Risk Analysis', fontsize=16, fontweight='bold')

        # Risk category distribution
        if 'risk_category' in patient_df.columns:
            risk_counts = patient_df['risk_category'].value_counts()
            colors = ['green', 'yellow', 'orange', 'red']
            axes[0, 0].pie(risk_counts.values, labels=risk_counts.index, autopct='%1.1f%%', colors=colors[:len(risk_counts)])
            axes[0, 0].set_title('Patient Risk Category Distribution')

        # Risk by age group
        if 'risk_category' in patient_df.columns and 'age_group' in patient_df.columns:
            risk_age_crosstab = pd.crosstab(patient_df['age_group'], patient_df['risk_category'])
            risk_age_crosstab.plot(kind='bar', stacked=True, ax=axes[0, 1], color=['green', 'yellow', 'orange', 'red'])
            axes[0, 1].set_title('Risk Categories by Age Group')
            axes[0, 1].set_ylabel('Count')
            plt.setp(axes[0, 1].xaxis.get_majorticklabels(), rotation=45)

        # Length of stay by risk category
        if 'risk_category' in patient_df.columns and 'length_of_stay' in patient_df.columns:
            patient_df.boxplot(column='length_of_stay', by='risk_category', ax=axes[1, 0])
            axes[1, 0].set_title('Length of Stay by Risk Category')
            axes[1, 0].set_xlabel('Risk Category')
            axes[1, 0].set_ylabel('Length of Stay (Days)')

        # Risk vs chronic conditions
        if 'risk_category' in patient_df.columns and 'chronic_conditions' in patient_df.columns:
            risk_chronic_crosstab = pd.crosstab(patient_df['chronic_conditions'], patient_df['risk_category'])
            risk_chronic_crosstab.plot(kind='bar', ax=axes[1, 1], color=['green', 'yellow', 'orange', 'red'])
            axes[1, 1].set_title('Risk Categories by Chronic Conditions')
            axes[1, 1].set_ylabel('Count')
            plt.setp(axes[1, 1].xaxis.get_majorticklabels(), rotation=45)

        plt.tight_layout()
        plt.show()

    def hospital_performance_metrics(self):
        """Analyze hospital performance metrics"""
        patient_df = None

        for sheet_name, df in self.sheets_data.items():
            if 'hospital_id' in df.columns:
                patient_df = df
                break

        if patient_df is None:
            print("❌ Hospital performance data not found")
            return

        print("\n" + "="*80)
        print("🏥 HOSPITAL PERFORMANCE METRICS")
        print("="*80)
```

```
        # Hospital-level metrics
        if 'hospital_id' in patient_df.columns:
            hospital_metrics = patient_df.groupby('hospital_id').agg({
                'patient_id': 'count',  # Total patients
                'length_of_stay': ['mean', 'median', 'std'],
                'complexity_score': ['mean', 'median'],
                'admission_date': ['min', 'max']
            }).round(2)

            hospital_metrics.columns = ['_'.join(col).strip() for col in hospital_metrics.columns.values]
            hospital_metrics = hospital_metrics.rename(columns={'patient_id_count': 'total_patients'})

            print("Hospital Performance Summary:")
            print(hospital_metrics)

        # Visualizations
        fig, axes = plt.subplots(2, 2, figsize=(14, 10))
        fig.suptitle('Hospital Performance Analysis', fontsize=16, fontweight='bold')

        # Patient volume by hospital
        if 'hospital_id' in patient_df.columns:
            hospital_volume = patient_df['hospital_id'].value_counts()
            axes[0, 0].bar(hospital_volume.index.astype(str), hospital_volume.values, color='lightblue')
            axes[0, 0].set_title('Patient Volume by Hospital')
            axes[0, 0].set_ylabel('Number of Patients')
            axes[0, 0].set_xlabel('Hospital ID')

        # Average length of stay by hospital
        if 'hospital_id' in patient_df.columns and 'length_of_stay' in patient_df.columns:
            hospital_los = patient_df.groupby('hospital_id')['length_of_stay'].mean()
            axes[0, 1].bar(hospital_los.index.astype(str), hospital_los.values, color='lightcoral')
            axes[0, 1].set_title('Average Length of Stay by Hospital')
            axes[0, 1].set_ylabel('Average Length of Stay (Days)')
            axes[0, 1].set_xlabel('Hospital ID')

        # Complexity score by hospital
        if 'hospital_id' in patient_df.columns and 'complexity_score' in patient_df.columns:
            hospital_complexity = patient_df.groupby('hospital_id')['complexity_score'].mean()
            axes[1, 0].bar(hospital_complexity.index.astype(str), hospital_complexity.values, color='lightgreen')
            axes[1, 0].set_title('Average Complexity Score by Hospital')
            axes[1, 0].set_ylabel('Average Complexity Score')
            axes[1, 0].set_xlabel('Hospital ID')

        # Patient type distribution by hospital
        if 'hospital_id' in patient_df.columns and 'patient_type' in patient_df.columns:
            hospital_patient_type = pd.crosstab(patient_df['hospital_id'], patient_df['patient_type'], normalize='index') * 100
            hospital_patient_type.plot(kind='bar', stacked=True, ax=axes[1, 1])
            axes[1, 1].set_title('Patient Type Distribution by Hospital (%)')
            axes[1, 1].set_ylabel('Percentage')
            axes[1, 1].set_xlabel('Hospital ID')
            plt.setp(axes[1, 1].xaxis.get_majorticklabels(), rotation=45)

        plt.tight_layout()
        plt.show()

    def generate_business_insights(self):
        """Generate key business insights and recommendations"""
        print("\n" + "="*80)
        print("💡 KEY BUSINESS INSIGHTS & RECOMMENDATIONS")
        print("="*80)

        insights = []
        recommendations = []

        for sheet_name, df in self.sheets_data.items():
            if 'patient_id' in df.columns:

                # Patient volume insights
                total_patients = len(df)
                insights.append(f"📊 Total patients analyzed: {total_patients:,}")

                # Age group insights
                if 'age_group' in df.columns:
                    elderly_patients = len(df[df['age_group'].isin(['51-65', '65+'])])
                    elderly_percentage = (elderly_patients / total_patients) * 100
                    insights.append(f"👴 Elderly patients (51+): {elderly_patients:,} ({elderly_percentage:.1f}%)")

                    if elderly_percentage > 40:
                        recommendations.append("🎯 Consider specialized geriatric care programs")
```

```
        # Chronic conditions insights
        if 'chronic_conditions' in df.columns:
            chronic_patients = len(df[df['chronic_conditions'].fillna('None') != 'None'])
            chronic_percentage = (chronic_patients / total_patients) * 100
            insights.append(f"🩺 Patients with chronic conditions: {chronic_patients:,} ({chronic_percentage:.1f}%)")

            if chronic_percentage > 50:
                recommendations.append("💊 Implement chronic disease management programs")

        # Length of stay insights
        if 'length_of_stay' in df.columns:
            avg_los = df['length_of_stay'].mean()
            long_stay_patients = len(df[df['length_of_stay'] > 7])
            insights.append(f"🛏️ Average length of stay: {avg_los:.1f} days")
            insights.append(f"⏰ Long-stay patients (>7 days): {long_stay_patients:,}")

            if avg_los > 5:
                recommendations.append("⚡ Review discharge planning processes to reduce length of stay")

        # Insurance insights
        if 'insurance_type' in df.columns:
            private_patients = len(df[df['insurance_type'] == 'Private'])
            private_percentage = (private_patients / total_patients) * 100
            insights.append(f"💳 Private insurance patients: {private_patients:,} ({private_percentage:.1f}%)")

        # Complexity insights
        if 'complexity_score' in df.columns:
            avg_complexity = df['complexity_score'].mean()
            high_complexity = len(df[df['complexity_score'] > 2])
            insights.append(f"📈 Average complexity score: {avg_complexity:.2f}")
            insights.append(f"🚨 High-complexity cases (>2.0): {high_complexity:,}")

            if high_complexity / total_patients > 0.2:
                recommendations.append("🏥 Consider establishing specialized high-acuity units")

        # Admission patterns
        if 'admission_day_of_week' in df.columns:
            weekend_admissions = len(df[df['admission_day_of_week'].isin(['Saturday', 'Sunday'])])
            weekend_percentage = (weekend_admissions / total_patients) * 100
            insights.append(f"🗓️ Weekend admissions: {weekend_admissions:,} ({weekend_percentage:.1f}%)")

            if weekend_percentage > 25:
                recommendations.append("👥 Ensure adequate weekend staffing levels")

    # Print insights
    print("\n🔍 KEY INSIGHTS:")
    for i, insight in enumerate(insights, 1):
        print(f"{i}. {insight}")

    print("\n💡 STRATEGIC RECOMMENDATIONS:")
    for i, recommendation in enumerate(recommendations, 1):
        print(f"{i}. {recommendation}")

    # Risk factors assessment
    print("\n⚠️ RISK FACTORS IDENTIFIED:")
    risk_factors = []

    for sheet_name, df in self.sheets_data.items():
        if 'patient_id' in df.columns:
            # High complexity concentration
            if 'complexity_score' in df.columns:
                high_complexity_rate = len(df[df['complexity_score'].fillna(0) > 2]) / len(df)
                if high_complexity_rate > 0.3:
                    risk_factors.append("🔴 High concentration of complex cases may strain resources")

            # Elderly patient concentration
            if 'age_group' in df.columns:
                elderly_rate = len(df[df['age_group'].isin(['51-65', '65+'])]) / len(df)
                if elderly_rate > 0.6:
                    risk_factors.append("🟡 High elderly patient ratio requires specialized care planning")

            # Emergency admissions
            if 'patient_type' in df.columns:
                emergency_rate = len(df[df['patient_type'] == 'Emergency']) / len(df)
                if emergency_rate > 0.4:
                    risk_factors.append("🟠 High emergency admission rate indicates potential capacity issues")

    if risk_factors:
```

```
            for i, risk in enumerate(risk_factors, 1):
                print(f"{i}. {risk}")
        else:
            print("✅ No major risk factors identified")

    def export_summary_report(self):
        """Export a comprehensive summary report"""
        print("\n" + "="*80)
        print("📄 GENERATING COMPREHENSIVE SUMMARY REPORT")
        print("="*80)

        try:
            with pd.ExcelWriter('patient_eda_summary_report.xlsx', engine='openpyxl') as writer:

                # Overall summary
                summary_data = {}

                for sheet_name, df in self.sheets_data.items():
                    if 'patient_id' in df.columns:

                        # Basic statistics
                        basic_stats = {
                            'Total_Patients': len(df),
                            'Date_Range_Start': df['admission_date'].min() if 'admission_date' in df.columns else 'N/A',
                            'Date_Range_End': df['admission_date'].max() if 'admission_date' in df.columns else 'N/A',
                            'Avg_Length_of_Stay': df['length_of_stay'].mean() if 'length_of_stay' in df.columns else 'N/A',
                            'Avg_Complexity_Score': df['complexity_score'].mean() if 'complexity_score' in df.columns else 'N/A'
                        }

                        summary_data[sheet_name] = basic_stats

                        # Demographic summary
                        if 'age_group' in df.columns:
                            age_summary = df['age_group'].value_counts()
                            age_summary.to_excel(writer, sheet_name=f'{sheet_name}_age_distribution')

                        if 'patient_type' in df.columns:
                            type_summary = df['patient_type'].value_counts()
                            type_summary.to_excel(writer, sheet_name=f'{sheet_name}_patient_types')

                        if 'chronic_conditions' in df.columns:
                            condition_summary = df['chronic_conditions'].value_counts()
                            condition_summary.to_excel(writer, sheet_name=f'{sheet_name}_conditions')

                        # Numerical summaries
                        numerical_cols = df.select_dtypes(include=[np.number]).columns
                        if len(numerical_cols) > 0:
                            numerical_summary = df[numerical_cols].describe()
                            numerical_summary.to_excel(writer, sheet_name=f'{sheet_name}_numerical_stats')

                # Overall summary sheet
                if summary_data:
                    summary_df = pd.DataFrame(summary_data).T
                    summary_df.to_excel(writer, sheet_name='Overall_Summary')

            print("✅ Summary report exported to 'patient_eda_summary_report.xlsx'")

        except Exception as e:
            print(f"❌ Error exporting report: {str(e)}")

    def run_complete_patient_eda(self):
        """Run the complete patient data EDA process"""
        print("🚀 STARTING COMPREHENSIVE PATIENT DATA EDA")
        print("="*80)

        # Preprocess data
        self.preprocess_data()

        # Basic information
        self.basic_info()

        # Demographic analysis
        self.patient_demographics_analysis()

        # Admission patterns
        self.admission_patterns_analysis()

        # Medical conditions analysis
        self.medical_conditions_analysis()
```

```
            # Insurance and financial analysis
            self.insurance_and_financial_analysis()

            # Temporal trends
            self.temporal_trends_analysis()

            # Risk analysis
            self.patient_risk_analysis()

            # Hospital performance
            self.hospital_performance_metrics()

            # Correlation analysis
            self.correlation_analysis()

            # Business insights
            self.generate_business_insights()

            # Export report
            self.export_summary_report()

            print("\n🎉 PATIENT DATA EDA COMPLETE!")
            print("="*80)

    # ======================================================================
    # USAGE FUNCTIONS
    # ======================================================================

    def run_patient_eda(file_path):
        """Simple function to run complete patient EDA"""
        try:
            # Test 1: Basic data loading
            eda = PatientDataEDA(FILE_PATH)
            print("✅ Data loaded successfully!")

            # Test 2: Run preprocessing
            eda.preprocess_data()
            print("✅ Data preprocessing completed!")

            # Test 3: Basic info (no visualizations)
            eda.basic_info()
            print("✅ Basic info generated!")

            # Test 4: Run complete EDA
            eda.run_complete_patient_eda()
            print("✅ Complete EDA analysis finished!")

        except Exception as e:
            print(f"❌ Error: {e}")
            print("\n🔧 Troubleshooting steps:")
            print("1. Check if the file path is correct")
            print("2. Ensure the Excel file exists")
            print("3. Verify all required packages are installed")


    def preview_patient_data(file_path):
        """Quick preview of the patient Excel file"""
        try:
            excel_file = pd.ExcelFile(file_path)
            print(f"📁 File: {file_path}")
            print(f"📊 Sheets: {excel_file.sheet_names}")
            print("="*60)

            for sheet_name in excel_file.sheet_names:
                df = pd.read_excel(file_path, sheet_name=sheet_name)
                print(f"\n📋 Sheet: {sheet_name}")
                print(f"   Size: {df.shape[0]:,} rows × {df.shape[1]} columns")
                print(f"   Columns: {list(df.columns)}")

                # Show data types
                print("   Data Types:")
                for col in df.columns:
                    print(f"     {col}: {df[col].dtype}")

                print(f"\n   Sample Data (first 3 rows):")
                print(df.head(3).to_string(index=False))
                print("-" * 60)
```

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

    def analyze_specific_metrics(file_path):
        """Analyze specific patient metrics"""
        eda = PatientDataEDA(file_path)
        eda.preprocess_data()


        print("🎯 SPECIFIC METRICS ANALYSIS")
        print("="*50)

        # Choose specific analyses
        print("Available analyses:")
        print("1. Demographics only")
        print("2. Medical conditions only")
        print("3. Temporal trends only")
        print("4. Risk analysis only")

        # Run specific analysis (example)
        eda.patient_demographics_analysis()
        eda.medical_conditions_analysis()

    # ======================================================================
    # INTERACTIVE DASHBOARD (OPTIONAL)
    # ======================================================================

    def create_patient_dashboard(file_path):
        """Create an interactive patient dashboard using Plotly"""
        try:
            import plotly.express as px
            import plotly.graph_objects as go
            from plotly.subplots import make_subplots

            eda = PatientDataEDA(file_path)
            eda.preprocess_data()

            # Get patient data
            patient_df = None
            for sheet_name, df in eda.sheets_data.items():
                if 'patient_id' in df.columns:
                    patient_df = df
                    break

        if patient_df is None:
            print("❌ Patient data not found for dashboard")
            return

        # Create dashboard
        fig = make_subplots(
            rows=3, cols=2,
            subplot_titles=('Age Distribution', 'Patient Type Distribution',
                            'Admission Trends', 'Length of Stay Distribution',
                            'Risk Categories', 'Insurance Types'),
            specs=[[{"type": "pie"}, {"type": "bar"}],
                   [{"type": "scatter"}, {"type": "histogram"}],
                   [{"type": "pie"}, {"type": "pie"}]]
        )

        # Age distribution
        if 'age_group' in patient_df.columns:
            age_counts = patient_df['age_group'].value_counts()
            fig.add_trace(
                go.Pie(labels=age_counts.index, values=age_counts.values, name="Age Groups"),
                row=1, col=1
            )

        # Patient type distribution
        if 'patient_type' in patient_df.columns:
            type_counts = patient_df['patient_type'].value_counts()
            fig.add_trace(
                go.Bar(x=type_counts.index, y=type_counts.values, name="Patient Types"),
                row=1, col=2
            )

        # Admission trends over time
        if 'admission_date' in patient_df.columns:
            daily_admissions = patient_df.groupby(patient_df['admission_date'].dt.date).size()
            fig.add_trace(
```

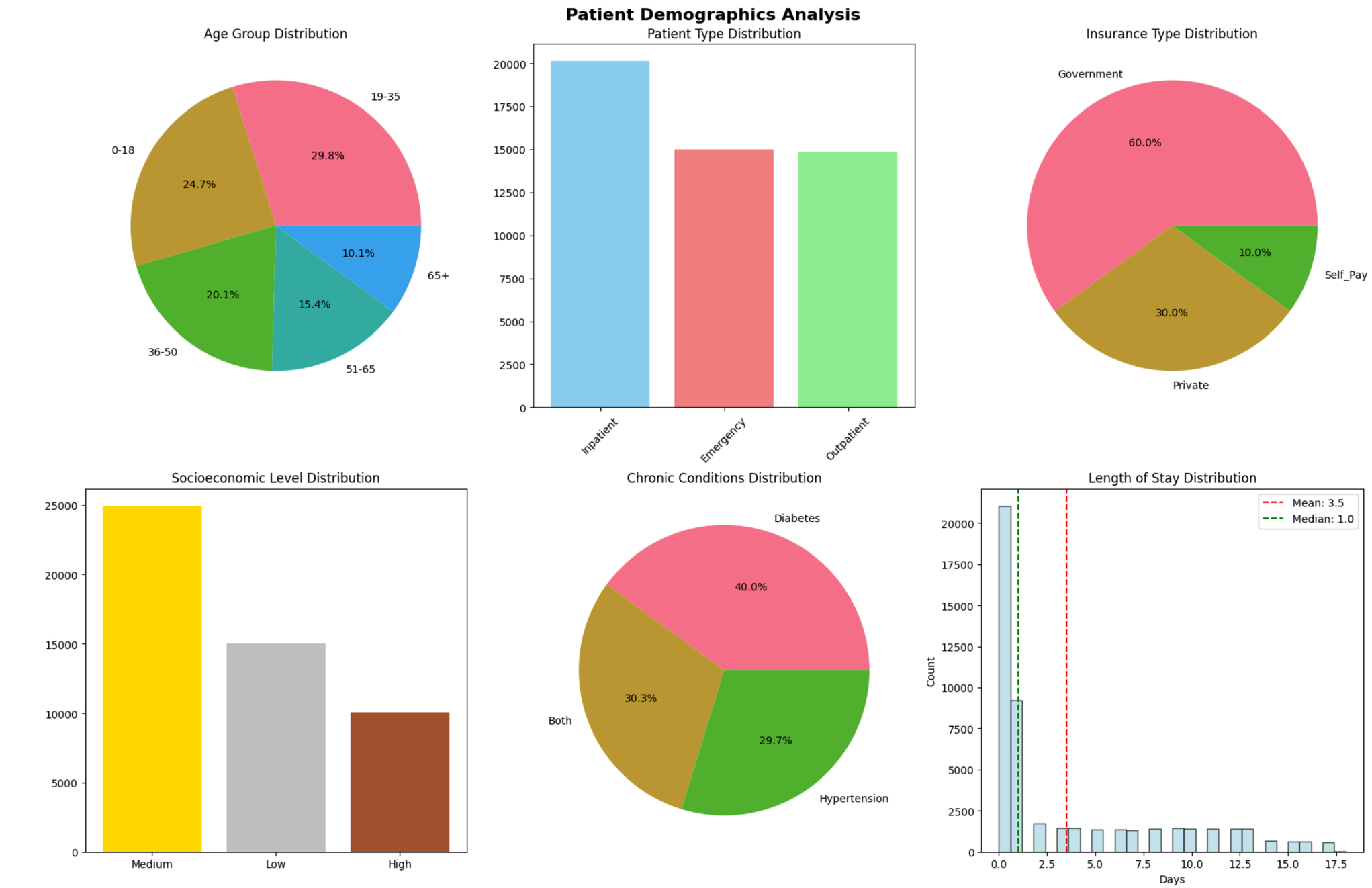
```
            go.Scatter(x=daily_admissions.index, y=daily_admissions.values,
                       mode='lines+markers', name="Daily Admissions"),
            row=2, col=1
        )

    # Length of stay distribution
    if 'length_of_stay' in patient_df.columns:
        fig.add_trace(
            go.Histogram(x=patient_df['length_of_stay'], name="Length of Stay"),
            row=2, col=2
        )

    # Risk categories (if created)
    if 'complexity_score' in patient_df.columns:
        patient_df['risk_category'] = pd.cut(patient_df['complexity_score'],
                                             bins=[0, 1, 1.5, 2, float('inf')],
                                             labels=['Low', 'Medium', 'High', 'Critical'])
        risk_counts = patient_df['risk_category'].value_counts()
        fig.add_trace(
            go.Pie(labels=risk_counts.index, values=risk_counts.values, name="Risk Categories"),
            row=3, col=1
        )

    # Insurance types
    if 'insurance_type' in patient_df.columns:
        insurance_counts = patient_df['insurance_type'].value_counts()
        fig.add_trace(
            go.Pie(labels=insurance_counts.index, values=insurance_counts.values, name="Insurance Types"),
            row=3, col=2
        )

    # Update layout
    fig.update_layout(
        height=1200,
        title_text="Patient Data Interactive Dashboard",
        title_font_size=20,
        showlegend=False
    )

    fig.show()

except ImportError:
    print("⚠️ Plotly not installed. Install with: pip install plotly")
except Exception as e:
    print(f"❌ Error creating dashboard: {e}")

# ================================================================
# MAIN EXECUTION
# ================================================================

if __name__ == "__main__":
    """
    Main execution block - Update the file path and choose your analysis
    """

    # 🎯 UPDATE THIS PATH TO YOUR EXCEL FILE
    FILE_PATH = "/content/drive/MyDrive/CAPSTONE/Data/Patients.xlsx"  # ← CHANGE THIS TO YOUR FILE PATH

    print("🏥 PATIENT DATA EDA SYSTEM")
    print("="*50)
    print("Available Options:")
    print("1. Quick Preview of Data")
    print("2. Complete EDA Analysis")
    print("3. Specific Metrics Analysis")
    print("4. Interactive Dashboard")

    print(f"\n📁 Current file path: {FILE_PATH}")
    print("\n🚀 To run analysis:")
    print("1. Update FILE_PATH with your Excel file location")
    print("2. Uncomment one of the function calls below")
    print("3. Run the script")

    # UNCOMMENT ONE OF THESE TO RUN:

    # Option 1: Quick preview
    preview_patient_data(FILE_PATH)

    # Option 2: Complete EDA (Recommended)
    run_patient_eda(FILE_PATH)
```

```
    # Option 3: Specific analysis
    #analyze_specific_metrics(FILE_PATH)

    # Option 4: Interactive dashboard
    #create_patient_dashboard(FILE_PATH)

# ================================================================
# INSTALLATION REQUIREMENTS
# ================================================================

print("\n🎯 FEATURES INCLUDED:")
print("✅ Patient demographics analysis")
print("✅ Admission patterns and trends")
print("✅ Medical conditions analysis")
print("✅ Insurance and financial patterns")
print("✅ Risk assessment and categorization")
print("✅ Hospital performance metrics")
print("✅ Temporal trends analysis")
print("✅ Business insights and recommendations")
print("✅ Automated report generation")
print("✅ Interactive dashboard (optional)")
```

```
🏥 PATIENT DATA EDA SYSTEM
==================================================
Available Options:
1. Quick Preview of Data
2. Complete EDA Analysis
3. Specific Metrics Analysis
4. Interactive Dashboard

📝 Current file path: /content/drive/MyDrive/CAPSTONE/Data/Patients.xlsx

🚀 To run analysis:
1. Update FILE_PATH with your Excel file location
2. Uncomment one of the function calls below
3. Run the script
📁 File: /content/drive/MyDrive/CAPSTONE/Data/Patients.xlsx
📊 Sheets: ['Sheet1']
============================================================

📋 Sheet: Sheet1
   Size: 50,000 rows × 12 columns
   Columns: ['patient_id', 'hospital_id', 'age_group', 'patient_type', 'chronic_conditions', 'insurance_type', 'socioeconomic_level', 'admission_date', 'discharge_date', 'length_of_stay', 'primary_dept_id', 'complexity_score']
   Data Types:
     patient_id: object
     hospital_id: object
     age_group: object
     patient_type: object
     chronic_conditions: object
     insurance_type: object
     socioeconomic_level: object
     admission_date: datetime64[ns]
     discharge_date: datetime64[ns]
     length_of_stay: float64
     primary_dept_id: object
     complexity_score: float64

   Sample Data (first 3 rows):
patient_id hospital_id age_group patient_type chronic_conditions insurance_type socioeconomic_level admission_date      discharge_date  length_of_stay primary_dept_id  complexity_score
  PT000001        H001     19-35    Inpatient               NaN     Government                Low     2024-01-02 2024-01-06 04:48:00             4.2            D004               1.0
  PT000002        H001     19-35   Outpatient              Both     Government               High     2024-11-03 2024-11-03 04:48:00             0.2            D003               1.6
  PT000003        H001      0-18    Emergency               NaN        Private               High     2023-11-18 2023-11-18 09:36:00             0.4            D001               0.8
------------------------------------------------------------
📊 Loading patient data from: /content/drive/MyDrive/CAPSTONE/Data/Patients.xlsx
📋 Available sheets: ['Sheet1']
✅ Loaded sheet 'Sheet1' with shape: (50000, 12)
✅ Data loaded successfully!

================================================================================
🔄 PREPROCESSING PATIENT DATA
================================================================================

📊 Processing sheet: Sheet1
✅ Converted admission_date to datetime
✅ Converted discharge_date to datetime
✅ Calculated length_of_stay
✅ Age groups found: ['19-35' '0-18' '51-65' '65+' '36-50']
✅ Created time-based features
✅ Data preprocessing completed!

================================================================================
📈 BASIC PATIENT DATA INFORMATION
================================================================================

🏥 SHEET: SHEET1
--------------------------------------------------
Shape: (50000, 15)
Columns: ['patient_id', 'hospital_id', 'age_group', 'patient_type', 'chronic_conditions', 'insurance_type', 'socioeconomic_level', 'admission_date', 'discharge_date', 'length_of_stay', 'primary_dept_id', 'complexity_score', 'admission_year', 'admission_month', 'admission_day

Data Types:
patient_id                      object
hospital_id                     object
age_group                       object
patient_type                    object
chronic_conditions              object
insurance_type                  object
socioeconomic_level             object
admission_date          datetime64[ns]
discharge_date          datetime64[ns]
length_of_stay                   int64
primary_dept_id                 object
complexity_score               float64
admission_year                   int32
admission_month                  int32
admission_day_of_week           object
dtype: object

Missing Values:
            Column  Missing Count  Missing Percentage
```

```
                Column  Missing_Count  Missing_Percentage
chronic_conditions          25007              50.01

Sample Data (first 3 rows):
  patient_id hospital_id age_group patient_type chronic_conditions insurance_type socioeconomic_level admission_date      discharge_date  length_of_stay primary_dept_id  complexity_score  admission_year  admission_month admission_day_of_week
0   PT000001        H001     19-35    Inpatient               NaN     Government                Low     2024-01-02 2024-01-06 04:48:00               4            D004               1.0            2024                1               Tuesday
1   PT000002        H001     19-35   Outpatient              Both     Government               High     2024-11-03 2024-11-03 04:48:00               0            D003               1.6            2024               11                Sunday
2   PT000003        H001      0-18    Emergency               NaN        Private               High     2023-11-18 2023-11-18 09:36:00               0            D001               0.8            2023               11              Saturday

--------------------------------------------------
✅ Basic info generated!
🚀 STARTING COMPREHENSIVE PATIENT DATA EDA
================================================================================

================================================================================
🔄 PREPROCESSING PATIENT DATA
================================================================================

📊 Processing sheet: Sheet1
✅ Converted admission_date to datetime
✅ Converted discharge_date to datetime
✅ Calculated length_of_stay
✅ Age groups found: ['19-35' '0-18' '51-65' '65+' '36-50']
✅ Created time-based features

================================================================================
📈 BASIC PATIENT DATA INFORMATION
================================================================================

📋 SHEET: SHEET1
--------------------------------------------------
Shape: (50000, 15)
Columns: ['patient_id', 'hospital_id', 'age_group', 'patient_type', 'chronic_conditions', 'insurance_type', 'socioeconomic_level', 'admission_date', 'discharge_date', 'length_of_stay', 'primary_dept_id', 'complexity_score', 'admission_year', 'admission_month', 'admission_day

Data Types:
patient_id                    object
hospital_id                   object
age_group                     object
patient_type                  object
chronic_conditions            object
insurance_type                object
socioeconomic_level           object
admission_date        datetime64[ns]
discharge_date        datetime64[ns]
length_of_stay                 int64
primary_dept_id               object
complexity_score             float64
admission_year                 int32
admission_month                int32
admission_day_of_week         object
dtype: object

Missing Values:
                Column  Missing_Count  Missing_Percentage
chronic_conditions          25007              50.01

Sample Data (first 3 rows):
  patient_id hospital_id age_group patient_type chronic_conditions insurance_type socioeconomic_level admission_date      discharge_date  length_of_stay primary_dept_id  complexity_score  admission_year  admission_month admission_day_of_week
0   PT000001        H001     19-35    Inpatient               NaN     Government                Low     2024-01-02 2024-01-06 04:48:00               4            D004               1.0            2024                1               Tuesday
1   PT000002        H001     19-35   Outpatient              Both     Government               High     2024-11-03 2024-11-03 04:48:00               0            D003               1.6            2024               11                Sunday
2   PT000003        H001      0-18    Emergency               NaN        Private               High     2023-11-18 2023-11-18 09:36:00               0            D001               0.8            2023               11              Saturday

--------------------------------------------------

================================================================================
👥 PATIENT DEMOGRAPHICS ANALYSIS
================================================================================

📊 Total Patients: 50,000

👶 Age Group Distribution:
age_group
0-18     12331
19-35    14909
36-50    10042
51-65     7677
65+       5041
Name: count, dtype: int64

📋 Patient Type Distribution:
patient_type
Inpatient     20125
Emergency     15010
Outpatient    14865
Name: count, dtype: int64

💰 Socioeconomic Level Distribution:
socioeconomic_level
```

```
Medium    24940
Low       14986
High      10074
Name: count, dtype: int64
```

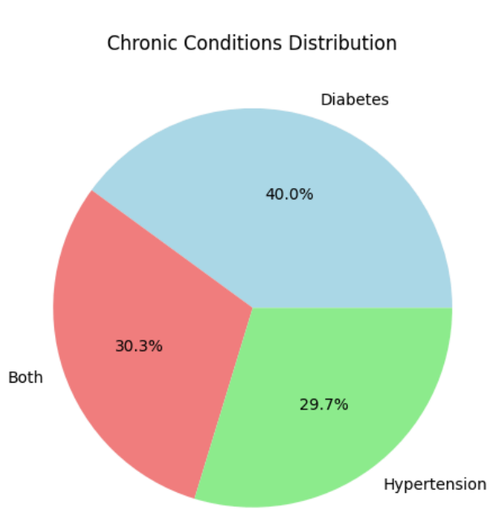


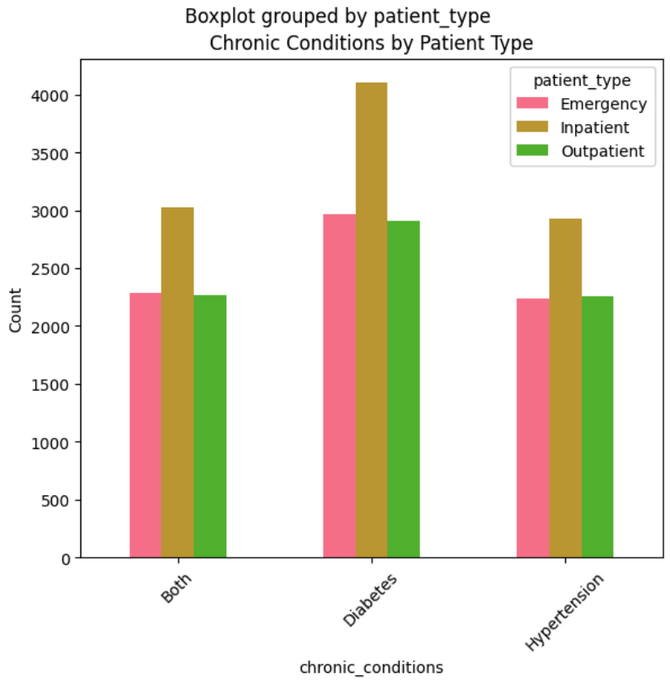

```
================================================================================
📅 ADMISSION PATTERNS ANALYSIS
================================================================================

📈 Admissions by Year:
admission_year
2023    24880
2024    25120
Name: count, dtype: int64

📅 Admissions by Day of Week:
admission_day_of_week
Friday       7293
Tuesday      7230
```

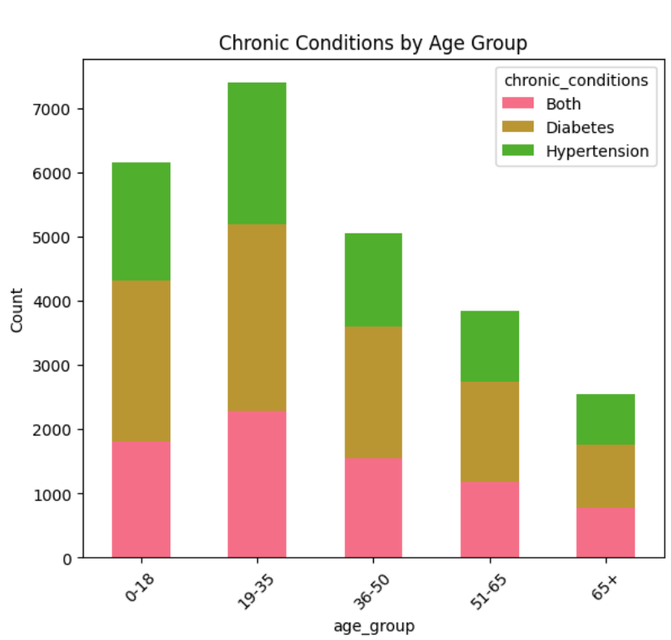
```
Saturday     7162
Monday       7130
Sunday       7095
Wednesday    7064
Thursday     7026
Name: count, dtype: int64
```

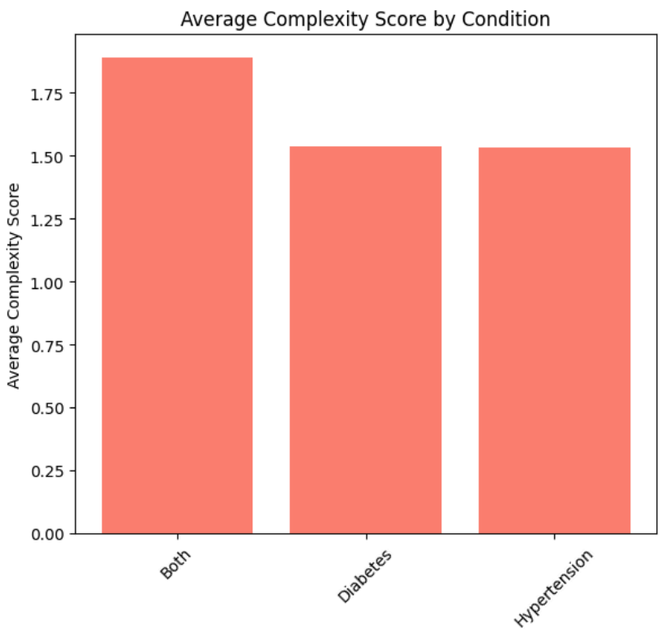

```
================================================================================
🏥 MEDICAL CONDITIONS ANALYSIS
================================================================================

🩺 Chronic Conditions Distribution:
chronic_conditions
Diabetes        9986
Both            7580
Hypertension    7427
Name: count, dtype: int64

📊 Patients with chronic conditions: 50.000 (100.0%)
```

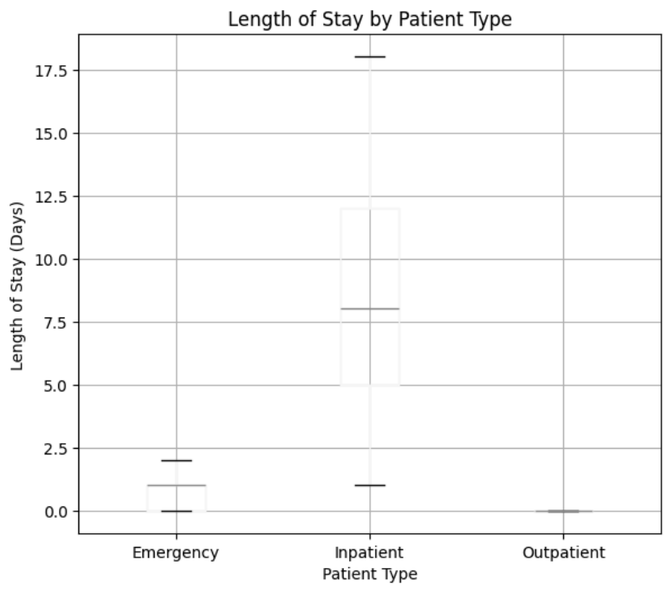
```
🏥 Primary Department Distribution:
primary_dept_id
D001    16732
D003    12209
D002     8794
D008     1816
D005     1783
D004     1763
D007     1758
D006     1732
D009     1712
D010     1701
Name: count, dtype: int64
```

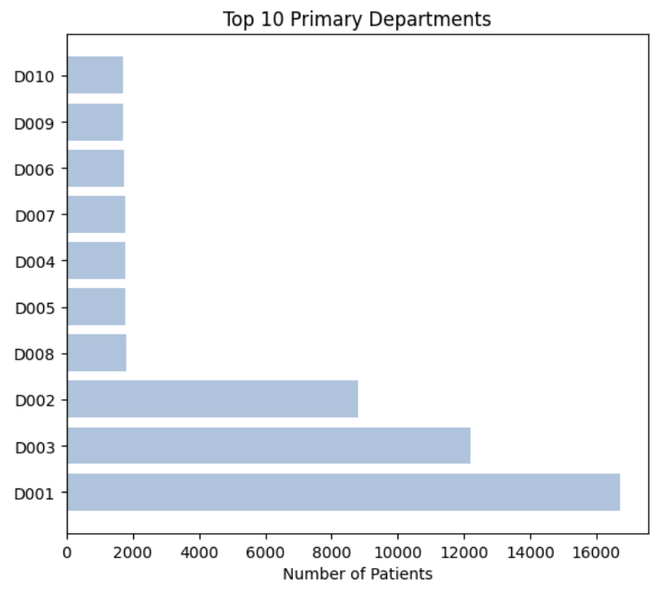






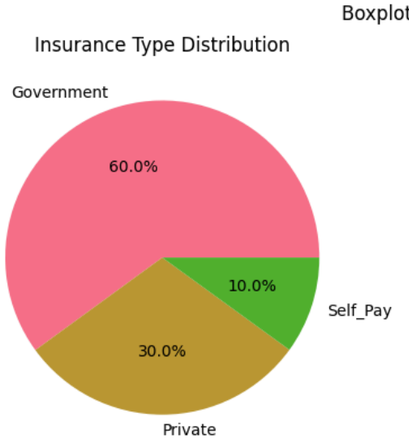


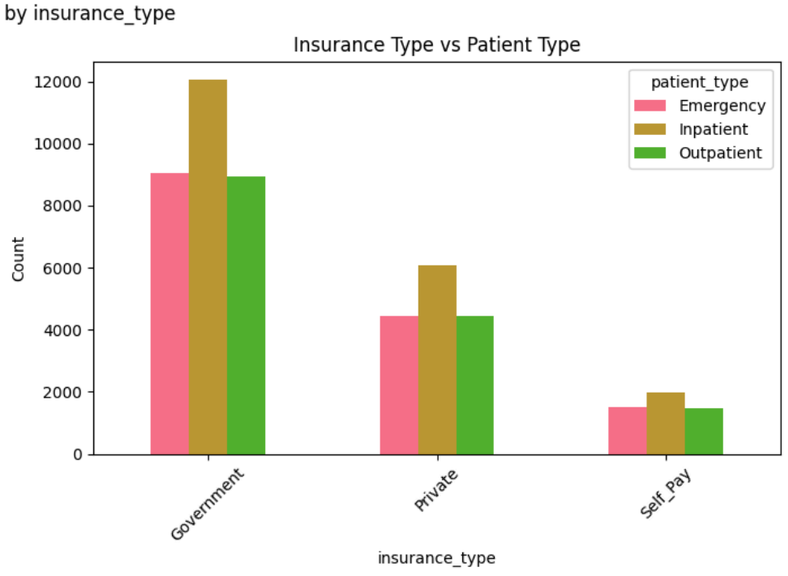


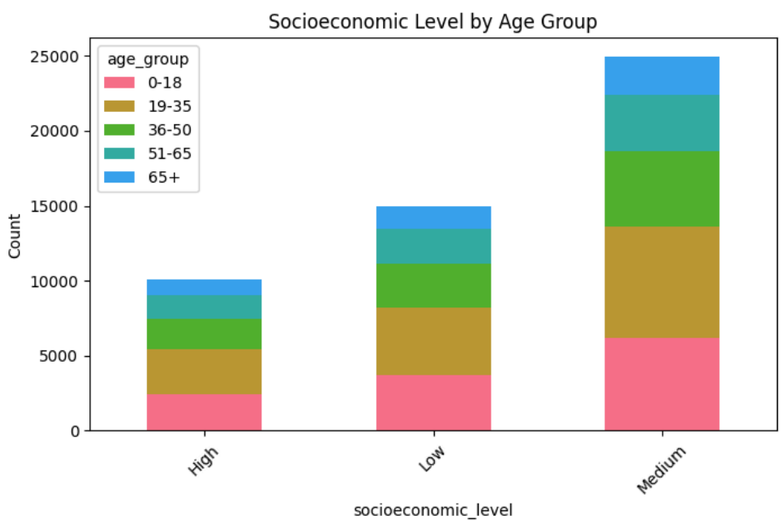


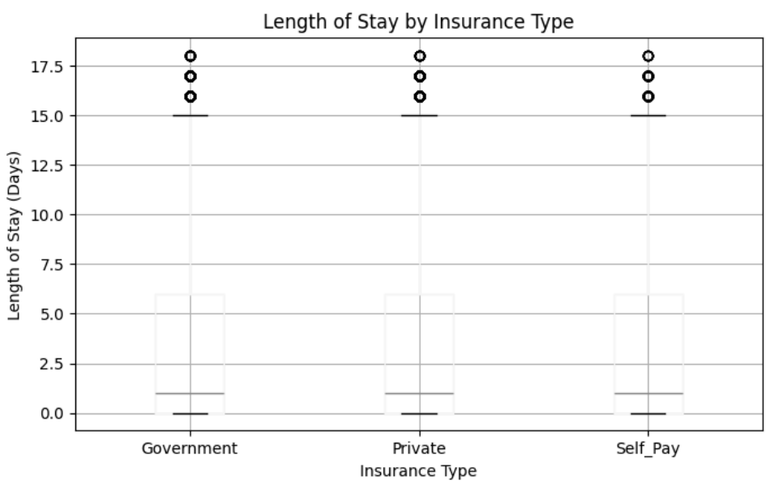

```
================================================================================
💳 INSURANCE & FINANCIAL ANALYSIS
================================================================================
```

```
🏧 Insurance Type Distribution:
insurance_type
Government    30013
Private       14992
Self_Pay       4995
Name: count, dtype: int64
   Government: 30,013 patients (60.0%)
   Private: 14,992 patients (30.0%)
   Self_Pay: 4,995 patients (10.0%)

💰 Socioeconomic Distribution:
socioeconomic_level
Medium    24940
Low       14986
High      10074
Name: count, dtype: int64
```

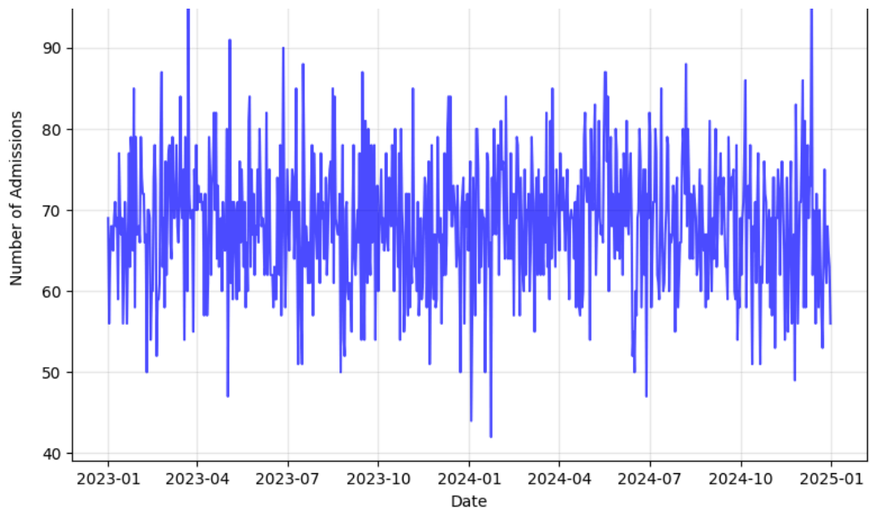


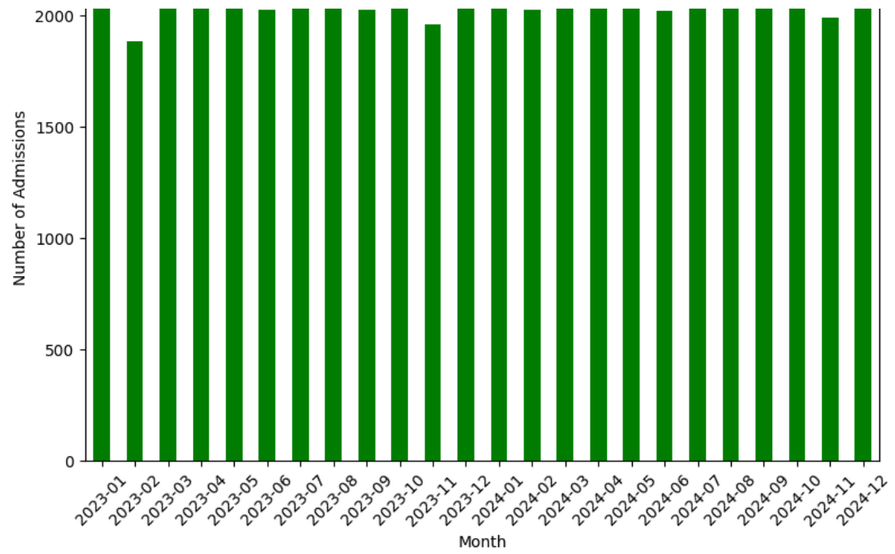


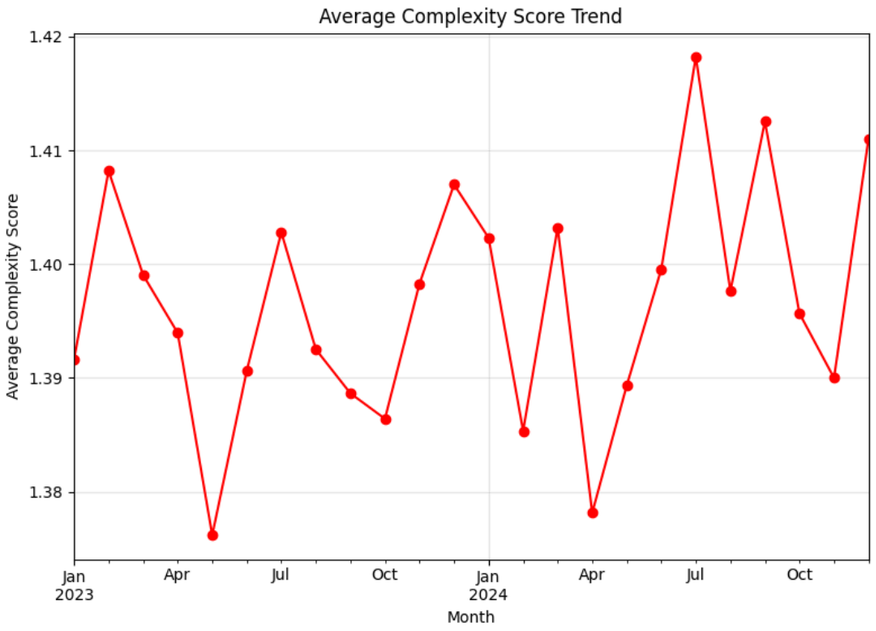


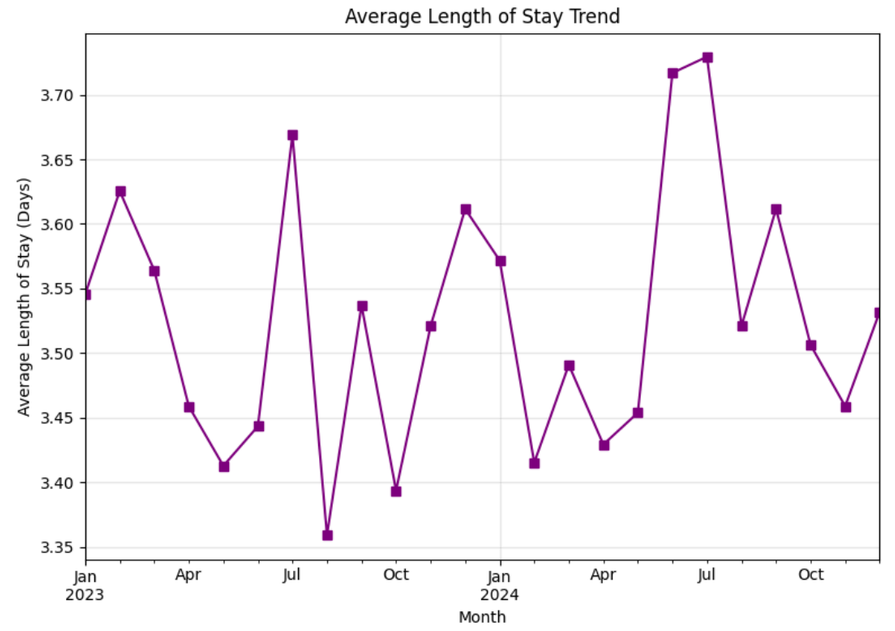



```
================================================================================
📈 TEMPORAL TRENDS ANALYSIS
================================================================================
📊 Date range: 2023-01-01 00:00:00 to 2024-12-31 00:00:00
📊 Average daily admissions: 68.40
📊 Peak admission day: 2024-12-12 (101 admissions)
```

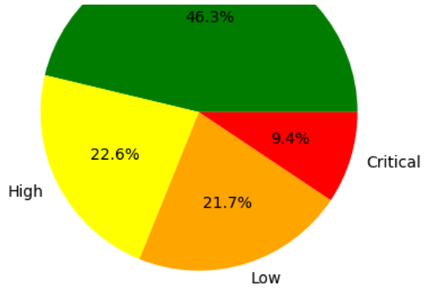


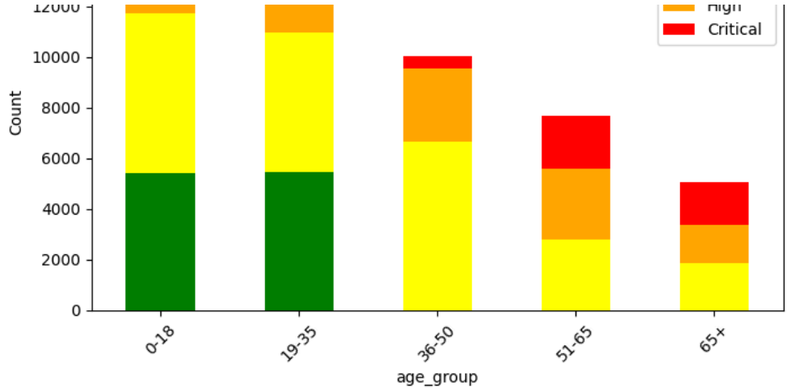


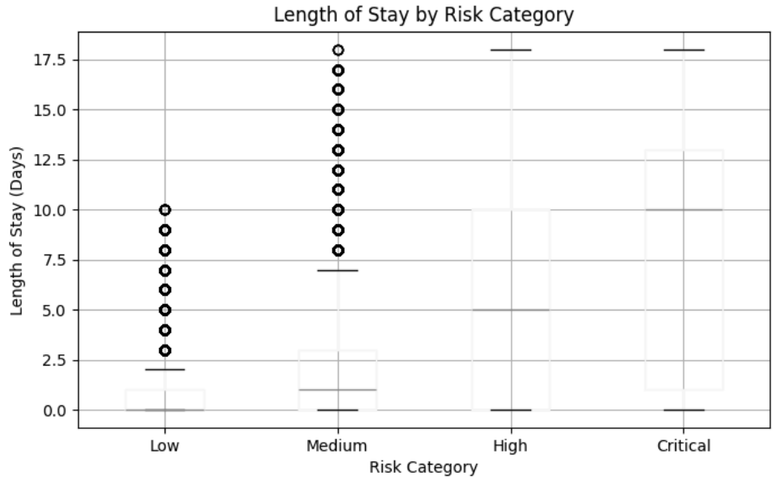


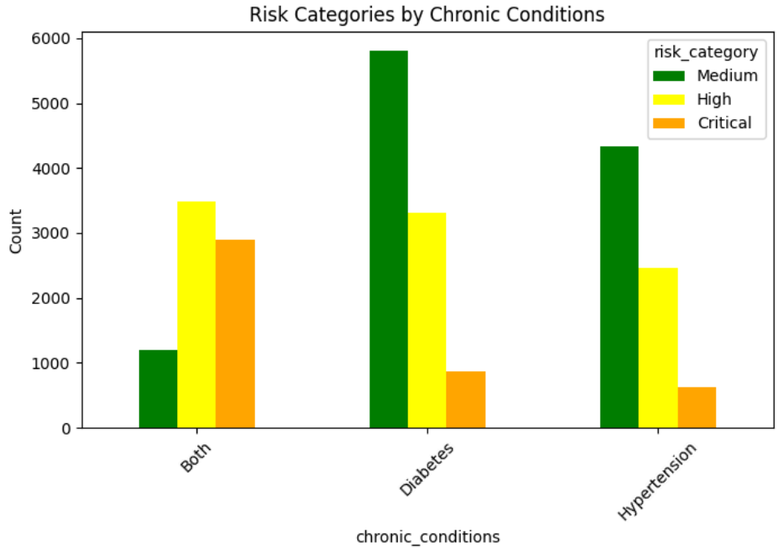



```
================================================================================
⚠️ PATIENT RISK ANALYSIS
================================================================================

⚠️ Risk Category Distribution:
risk_category
Medium      23127
High        11311
Low         10874
Critical     4688
Name: count, dtype: int64

🚨 High-risk patients: 15,999 (32.0%)

High-risk patient characteristics:
Age groups: {'51-65': 4890, '19-35': 3928, '36-50': 3381, '65+': 3197, '0-18': 603}
Chronic conditions: {'Both': 6378, 'Diabetes': 4172, 'Hypertension': 3092}
```

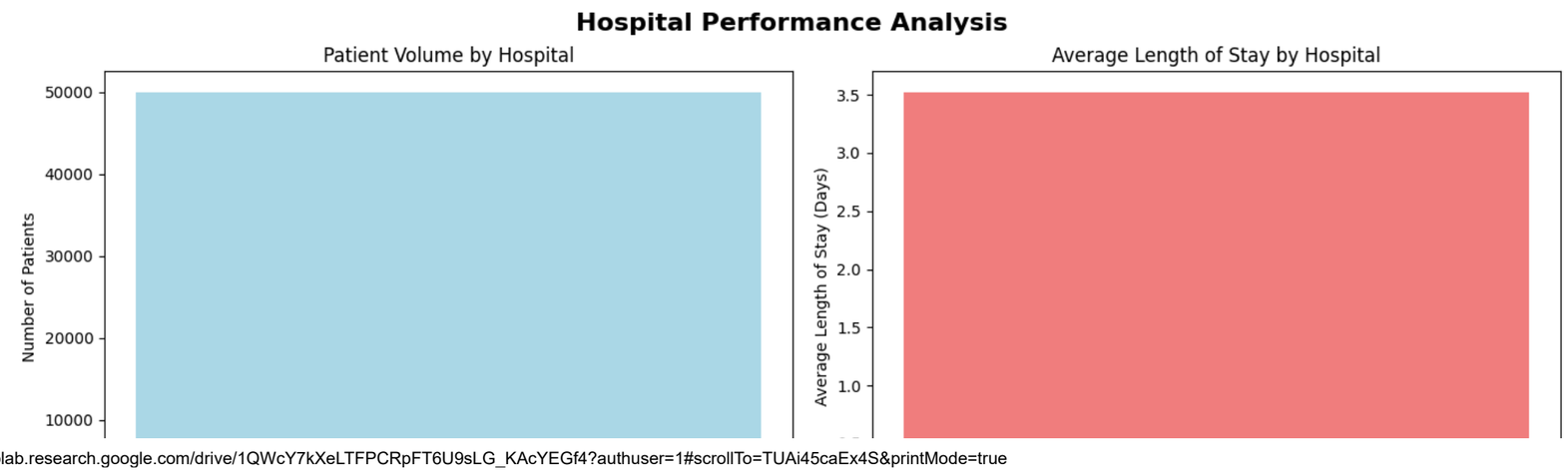


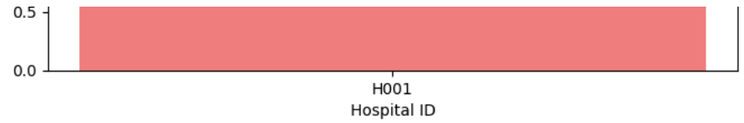


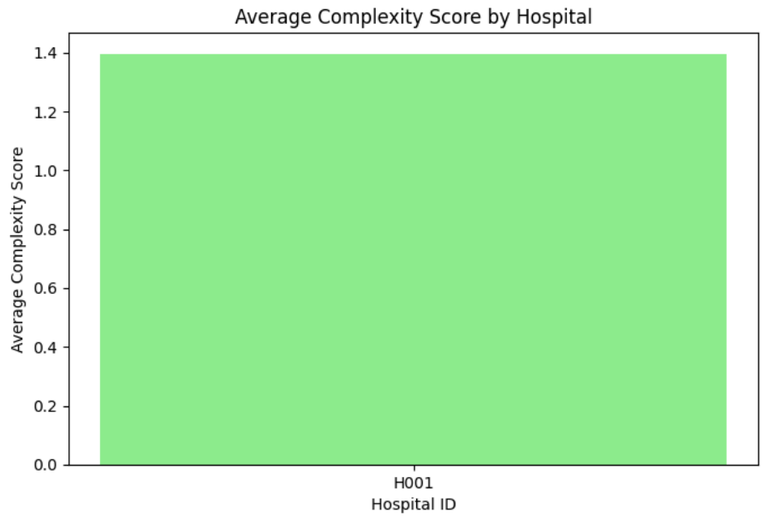


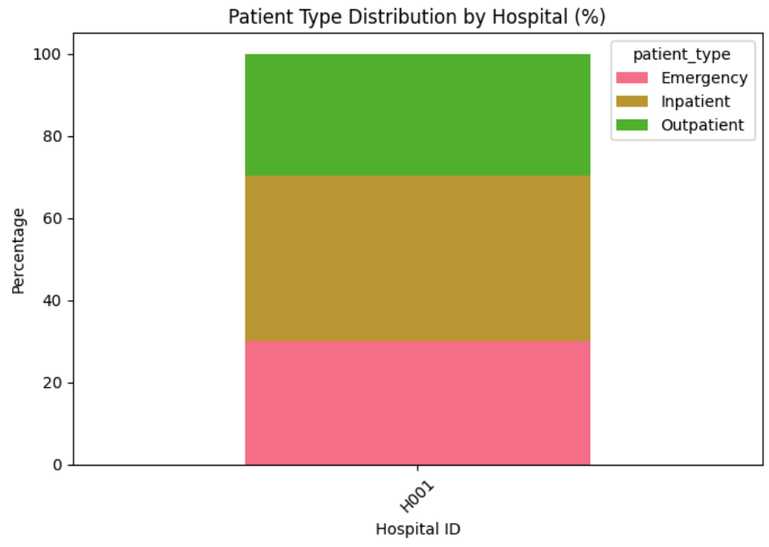



```
================================================================================
🏥 HOSPITAL PERFORMANCE METRICS
================================================================================
Hospital Performance Summary:
            total_patients  length_of_stay_mean  length_of_stay_median  \
hospital_id
H001                 50000                 3.52                    1.0

            length_of_stay_std  complexity_score_mean  \
hospital_id
H001                      4.82                    1.4

            complexity_score_median admission_date_min admission_date_max
hospital_id
H001                            1.3         2023-01-01         2024-12-31
```

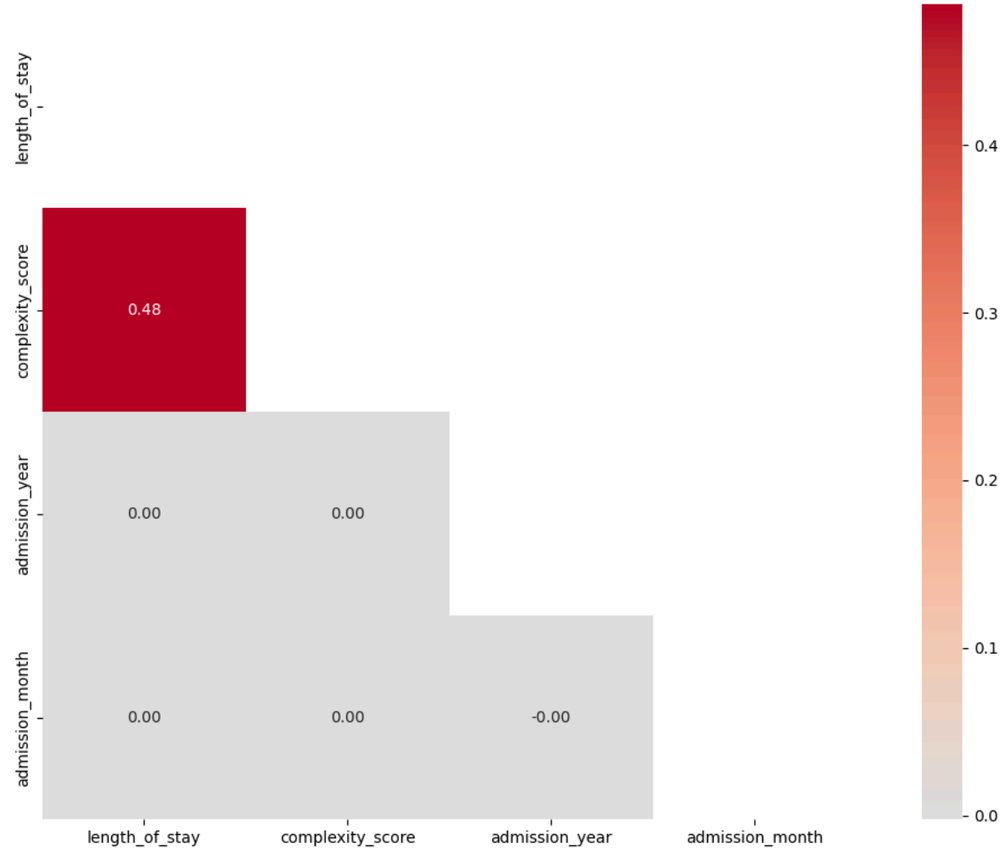









```
================================================================================
🔗 CORRELATION ANALYSIS
================================================================================

Correlations for SHEET1:
```




```
Strongest correlations (>0.5 or <-0.5):
```

```
================================================================================
💡 KEY BUSINESS INSIGHTS & RECOMMENDATIONS
================================================================================

🔍 KEY INSIGHTS:
1. 📊 Total patients analyzed: 50,000
2. 👴 Elderly patients (51+): 12,718 (25.4%)
3. 🩺 Patients with chronic conditions: 24,993 (50.0%)
4. 🛏️ Average length of stay: 3.5 days
5. ⏰ Long-stay patients (>7 days): 11,084
6. 💳 Private insurance patients: 14,992 (30.0%)
7. 📈 Average complexity score: 1.40
8. 🚨 High-complexity cases (>2.0): 4,688
9. 📅 Weekend admissions: 14,257 (28.5%)

💡 STRATEGIC RECOMMENDATIONS:
1. 👥 Ensure adequate weekend staffing levels

⚠️ RISK FACTORS IDENTIFIED:
✅ No major risk factors identified

================================================================================
📄 GENERATING COMPREHENSIVE SUMMARY REPORT
================================================================================
✅ Summary report exported to 'patient_eda_summary_report.xlsx'

🎉 PATIENT DATA EDA COMPLETE!
================================================================================
✅ Complete EDA analysis finished!

🎯 FEATURES INCLUDED:
✅ Patient demographics analysis
✅ Admission patterns and trends
✅ Medical conditions analysis
✅ Insurance and financial patterns
✅ Risk assessment and categorization
✅ Hospital performance metrics
✅ Temporal trends analysis
✅ Business insights and recommendations
✅ Automated report generation
✅ Interactive dashboard (optional)
```

Supply_Chain

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# Set plotting style
plt.style.use('default')
sns.set_palette("husl")

class SupplyChainEDA:
    def __init__(self, file_path):
        """
        Initialize the EDA class with Excel file path
        """
        self.file_path = file_path
        self.sheets_data = {}
        self.load_data()

    def load_data(self):
        """
        Load all sheets from Excel file
        """
        try:
            # Read all sheets from Excel file
            excel_file = pd.ExcelFile(self.file_path)
            print(f"Available sheets: {excel_file.sheet_names}")

            for sheet_name in excel_file.sheet_names:
                self.sheets_data[sheet_name] = pd.read_excel(self.file_path, sheet_name=sheet_name)
                print(f"Loaded sheet '{sheet_name}' with shape: {self.sheets_data[sheet_name].shape}")

        except Exception as e:
            print(f"Error loading data: {e}")

    def basic_info(self):
        """
        Display basic information about all sheets
        """
        print("="*80)
        print("BASIC DATA INFORMATION")
        print("="*80)

        for sheet_name, df in self.sheets_data.items():
            print(f"\n--- {sheet_name.upper()} SHEET ---")
            print(f"Shape: {df.shape}")
            print(f"Columns: {list(df.columns)}")
            print(f"Data types:\n{df.dtypes}")
            print(f"Missing values:\n{df.isnull().sum()}")
            print(f"Duplicate rows: {df.duplicated().sum()}")
            print("-" * 60)

    def convert_datetime_columns(self):
        """
        Convert date columns to datetime format
        """
        date_columns = ['po_date', 'requested_delivery_date', 'delivery_date', 'expiry_date']

        for sheet_name, df in self.sheets_data.items():
            for col in date_columns:
                if col in df.columns:
                    try:
                        df[col] = pd.to_datetime(df[col])
                        print(f"Converted {col} to datetime in {sheet_name}")
                    except:
                        print(f"Could not convert {col} to datetime in {sheet_name}")

    def purchase_orders_analysis(self):
        """
        Analyze purchase orders data (first sheet)
        """
        if not self.sheets_data:
            print("No data loaded")
```

```
        return

    # Get the first sheet (purchase orders)
    po_data = list(self.sheets_data.values())[0]
    sheet_name = list(self.sheets_data.keys())[0]

    print("="*80)
    print("PURCHASE ORDERS ANALYSIS")
    print("="*80)

    # Basic statistics
    print("\nBASIC STATISTICS:")
    numeric_cols = po_data.select_dtypes(include=[np.number]).columns
    print(po_data[numeric_cols].describe())

    # Create visualizations
    fig, axes = plt.subplots(2, 3, figsize=(18, 12))
    fig.suptitle('Purchase Orders Analysis', fontsize=16, fontweight='bold')

    # 1. Order quantity distribution
    axes[0,0].hist(po_data['ordered_quantity'], bins=30, alpha=0.7, color='skyblue', edgecolor='black')
    axes[0,0].set_title('Distribution of Ordered Quantities')
    axes[0,0].set_xlabel('Ordered Quantity')
    axes[0,0].set_ylabel('Frequency')

    # 2. Total value distribution
    axes[0,1].hist(po_data['total_value'], bins=30, alpha=0.7, color='lightgreen', edgecolor='black')
    axes[0,1].set_title('Distribution of Total Values')
    axes[0,1].set_xlabel('Total Value')
    axes[0,1].set_ylabel('Frequency')

    # 3. Vendor selection reasons
    vendor_reasons = po_data['vendor_selection_reason'].value_counts()
    axes[0,2].pie(vendor_reasons.values, labels=vendor_reasons.index, autopct='%1.1f%%')
    axes[0,2].set_title('Vendor Selection Reasons')

    # 4. Orders by vendor
    vendor_counts = po_data['vendor_id'].value_counts()
    axes[1,0].bar(vendor_counts.index, vendor_counts.values, color='orange', alpha=0.7)
    axes[1,0].set_title('Orders by Vendor')
    axes[1,0].set_xlabel('Vendor ID')
    axes[1,0].set_ylabel('Number of Orders')

    # 5. Urgency levels
    urgency_counts = po_data['urgency_level'].value_counts()
    axes[1,1].bar(urgency_counts.index, urgency_counts.values, color='red', alpha=0.7)
    axes[1,1].set_title('Orders by Urgency Level')
    axes[1,1].set_xlabel('Urgency Level')
    axes[1,1].set_ylabel('Count')

    # 6. PO Status
    status_counts = po_data['po_status'].value_counts()
    axes[1,2].bar(status_counts.index, status_counts.values, color='purple', alpha=0.7)
    axes[1,2].set_title('Purchase Order Status')
    axes[1,2].set_xlabel('PO Status')
    axes[1,2].set_ylabel('Count')

    plt.tight_layout()
    plt.show()

    # Key insights
    print("\nKEY INSIGHTS - PURCHASE ORDERS:")
    print(f"• Total number of purchase orders: {len(po_data)}")
    print(f"• Total value of all orders: ${po_data['total_value'].sum():,.2f}")
    print(f"• Average order value: ${po_data['total_value'].mean():,.2f}")
    print(f"• Most common vendor: {po_data['vendor_id'].mode()[0]}")
    print(f"• Most common urgency level: {po_data['urgency_level'].mode()[0]}")
    print(f"• Most common vendor selection reason: {po_data['vendor_selection_reason'].mode()[0]}")

def delivery_analysis(self):
    """
    Analyze delivery data (second sheet)
    """
    if len(self.sheets_data) < 2:
        print("Delivery data not available")
        return

    delivery_data = list(self.sheets_data.values())[1]
```

```
        print("="*80)
        print("DELIVERY ANALYSIS")
        print("="*80)

        # Basic statistics
        print("\nDELIVERY STATISTICS:")
        numeric_cols = delivery_data.select_dtypes(include=[np.number]).columns
        print(delivery_data[numeric_cols].describe())

        # Create visualizations
        fig, axes = plt.subplots(2, 3, figsize=(18, 12))
        fig.suptitle('Delivery Performance Analysis', fontsize=16, fontweight='bold')

        # 1. Delivered vs Accepted quantities
        axes[0,0].scatter(delivery_data['delivered_quantity'], delivery_data['accepted_quantity'],
                          alpha=0.6, color='blue')
        axes[0,0].plot([delivery_data['delivered_quantity'].min(), delivery_data['delivered_quantity'].max()],
                       [delivery_data['delivered_quantity'].min(), delivery_data['delivered_quantity'].max()],
                       'r--', label='Perfect Match')
        axes[0,0].set_title('Delivered vs Accepted Quantities')
        axes[0,0].set_xlabel('Delivered Quantity')
        axes[0,0].set_ylabel('Accepted Quantity')
        axes[0,0].legend()

        # 2. Rejected quantities
        axes[0,1].hist(delivery_data['rejected_quantity'], bins=20, alpha=0.7, color='red', edgecolor='black')
        axes[0,1].set_title('Distribution of Rejected Quantities')
        axes[0,1].set_xlabel('Rejected Quantity')
        axes[0,1].set_ylabel('Frequency')

        # 3. Price variance analysis
        axes[0,2].hist(delivery_data['price_variance_pct'], bins=30, alpha=0.7, color='green', edgecolor='black')
        axes[0,2].set_title('Price Variance Percentage Distribution')
        axes[0,2].set_xlabel('Price Variance %')
        axes[0,2].set_ylabel('Frequency')

        # 4. Rejection reasons
        rejection_reasons = delivery_data['rejection_reason'].value_counts().head(10)
        axes[1,0].barh(rejection_reasons.index, rejection_reasons.values, color='coral')
        axes[1,0].set_title('Top 10 Rejection Reasons')
        axes[1,0].set_xlabel('Count')

        # 5. Vendor performance (by rejection rate)
        vendor_performance = delivery_data.groupby('vendor_id').agg({
            'rejected_quantity': 'sum',
            'delivered_quantity': 'sum'
        })
        vendor_performance['rejection_rate'] = (vendor_performance['rejected_quantity'] /
                                                vendor_performance['delivered_quantity'] * 100)
        top_vendors = vendor_performance.nlargest(10, 'rejection_rate')

        axes[1,1].bar(range(len(top_vendors)), top_vendors['rejection_rate'], color='orange', alpha=0.7)
        axes[1,1].set_title('Top 10 Vendors by Rejection Rate')
        axes[1,1].set_xlabel('Vendor (Index)')
        axes[1,1].set_ylabel('Rejection Rate %')

        # 6. Actual vs Expected unit price
        axes[1,2].scatter(delivery_data['actual_unit_price'], delivery_data['price_variance_pct'],
                          alpha=0.6, color='purple')
        axes[1,2].set_title('Unit Price vs Price Variance')
        axes[1,2].set_xlabel('Actual Unit Price')
        axes[1,2].set_ylabel('Price Variance %')

        plt.tight_layout()
        plt.show()

        # Key insights
        print("\nKEY INSIGHTS - DELIVERIES:")
        print(f"• Total deliveries: {len(delivery_data)}")
        print(f"• Average delivery acceptance rate: {(delivery_data['accepted_quantity'].sum() / delivery_data['delivered_quantity'].sum() * 100):.2f}%")
        print(f"• Average price variance: {delivery_data['price_variance_pct'].mean():.2f}%")
        print(f"• Most common rejection reason: {delivery_data['rejection_reason'].mode()[0] if not delivery_data['rejection_reason'].isnull().all() else 'No rejections'}")

    def batch_quality_analysis(self):
        """
        Analyze batch and quality data (third sheet)
        """
        if len(self.sheets_data) < 3:
            print("Batch quality data not available")
            return
```

```
        batch_data = list(self.sheets_data.values())[2]

        print("="*80)
        print("BATCH QUALITY ANALYSIS")
        print("="*80)

        # Basic statistics
        print("\nBATCH QUALITY STATISTICS:")
        numeric_cols = batch_data.select_dtypes(include=[np.number]).columns
        print(batch_data[numeric_cols].describe())

        # Create visualizations
        fig, axes = plt.subplots(2, 3, figsize=(18, 12))
        fig.suptitle('Batch Quality and Market Analysis', fontsize=16, fontweight='bold')

        # 1. Quality rating distribution
        axes[0,0].hist(batch_data['quality_rating'], bins=20, alpha=0.7, color='gold', edgecolor='black')
        axes[0,0].set_title('Quality Rating Distribution')
        axes[0,0].set_xlabel('Quality Rating')
        axes[0,0].set_ylabel('Frequency')

        # 2. Days late distribution
        axes[0,1].hist(batch_data['days_late'], bins=30, alpha=0.7, color='tomato', edgecolor='black')
        axes[0,1].set_title('Days Late Distribution')
        axes[0,1].set_xlabel('Days Late')
        axes[0,1].set_ylabel('Frequency')

        # 3. Delivery status
        status_counts = batch_data['delivery_status'].value_counts()
        axes[0,2].pie(status_counts.values, labels=status_counts.index, autopct='%1.1f%%')
        axes[0,2].set_title('Delivery Status Distribution')

        # 4. Market conditions
        market_counts = batch_data['market_condition'].value_counts()
        axes[1,0].bar(market_counts.index, market_counts.values, color='lightblue', alpha=0.7)
        axes[1,0].set_title('Market Conditions')
        axes[1,0].set_xlabel('Market Condition')
        axes[1,0].set_ylabel('Count')
        axes[1,0].tick_params(axis='x', rotation=45)

        # 5. Waste value vs Quality rating
        axes[1,1].scatter(batch_data['quality_rating'], batch_data['waste_value'],
                          alpha=0.6, color='brown')
        axes[1,1].set_title('Quality Rating vs Waste Value')
        axes[1,1].set_xlabel('Quality Rating')
        axes[1,1].set_ylabel('Waste Value')

        # 6. Expected waste quantity
        axes[1,2].hist(batch_data['expected_waste_qty'], bins=25, alpha=0.7, color='darkgreen', edgecolor='black')
        axes[1,2].set_title('Expected Waste Quantity Distribution')
        axes[1,2].set_xlabel('Expected Waste Quantity')
        axes[1,2].set_ylabel('Frequency')

        plt.tight_layout()
        plt.show()

        # Key insights
        print("\nKEY INSIGHTS - BATCH QUALITY:")
        print(f"• Total batches analyzed: {len(batch_data)}")
        print(f"• Average quality rating: {batch_data['quality_rating'].mean():.2f}")
        print(f"• Average days late: {batch_data['days_late'].mean():.2f}")
        print(f"• Total waste value: ${batch_data['waste_value'].sum():,.2f}")
        print(f"• On-time delivery rate: {(batch_data['days_late'] == 0).sum() / len(batch_data) * 100:.2f}%")

    def comprehensive_analysis(self):
        """
        Perform comprehensive cross-sheet analysis
        """
        print("="*80)
        print("COMPREHENSIVE SUPPLY CHAIN ANALYSIS")
        print("="*80)

        # Time series analysis if date columns exist
        self.time_series_analysis()

        # Vendor performance analysis
        self.vendor_performance_analysis()
```

```
        # Cost and efficiency analysis
        self.cost_efficiency_analysis()

        # Quality and delivery correlation
        self.quality_delivery_correlation()

    def time_series_analysis(self):
        """
        Analyze trends over time
        """
        print("\nTIME SERIES ANALYSIS:")

        for sheet_name, df in self.sheets_data.items():
            date_cols = [col for col in df.columns if 'date' in col.lower()]

            if date_cols:
                print(f"\n--- {sheet_name} Time Analysis ---")

                # Convert to datetime if not already
                for col in date_cols:
                    if not pd.api.types.is_datetime64_any_dtype(df[col]):
                        try:
                            df[col] = pd.to_datetime(df[col])
                        except:
                            continue

                # Plot time series for the first date column
                if len(date_cols) > 0:
                    date_col = date_cols[0]
                    df_sorted = df.sort_values(date_col)

                    plt.figure(figsize=(12, 6))

                    # Plot orders/deliveries over time
                    monthly_counts = df_sorted.groupby(df_sorted[date_col].dt.to_period('M')).size()

                    plt.subplot(1, 2, 1)
                    monthly_counts.plot(kind='line', marker='o', color='blue', alpha=0.7)
                    plt.title(f'{sheet_name} - Monthly Trends')
                    plt.xlabel('Month')
                    plt.ylabel('Count')
                    plt.xticks(rotation=45)

                    # Plot value trends if total_value exists
                    if 'total_value' in df.columns:
                        plt.subplot(1, 2, 2)
                        monthly_values = df_sorted.groupby(df_sorted[date_col].dt.to_period('M'))['total_value'].sum()
                        monthly_values.plot(kind='line', marker='s', color='green', alpha=0.7)
                        plt.title(f'{sheet_name} - Monthly Value Trends')
                        plt.xlabel('Month')
                        plt.ylabel('Total Value')
                        plt.xticks(rotation=45)

                    plt.tight_layout()
                    plt.show()

    def vendor_performance_analysis(self):
        """
        Analyze vendor performance across sheets
        """
        print("\nVENDOR PERFORMANCE ANALYSIS:")

        # Combine vendor data from multiple sheets
        vendor_metrics = {}

        for sheet_name, df in self.sheets_data.items():
            if 'vendor_id' in df.columns:
                print(f"\n--- {sheet_name} Vendor Analysis ---")

                # Count orders per vendor
                vendor_counts = df['vendor_id'].value_counts()
                print(f"Top 5 vendors by order count:")
                print(vendor_counts.head())

                # Average values if available
                if 'total_value' in df.columns:
                    vendor_avg_values = df.groupby('vendor_id')['total_value'].mean().sort_values(ascending=False)
                    print(f"\nTop 5 vendors by average order value:")
                    print(vendor_avg_values.head())
```

```
                # Plot vendor distribution
                plt.figure(figsize=(12, 6))
                vendor_counts.head(10).plot(kind='bar', color='steelblue', alpha=0.7)
                plt.title(f'{sheet_name} - Top 10 Vendors by Order Count')
                plt.xlabel('Vendor ID')
                plt.ylabel('Number of Orders')
                plt.xticks(rotation=45)
                plt.tight_layout()
                plt.show()

    def cost_efficiency_analysis(self):
        """
        Analyze cost efficiency and price variations
        """
        print("\nCOST EFFICIENCY ANALYSIS:")

        for sheet_name, df in self.sheets_data.items():
            if any(col in df.columns for col in ['total_value', 'unit_price', 'price_variance_pct']):
                print(f"\n--- {sheet_name} Cost Analysis ---")

                plt.figure(figsize=(15, 5))

                # Plot 1: Unit price distribution
                if 'unit_price' in df.columns:
                    plt.subplot(1, 3, 1)
                    plt.hist(df['unit_price'], bins=30, alpha=0.7, color='green', edgecolor='black')
                    plt.title('Unit Price Distribution')
                    plt.xlabel('Unit Price')
                    plt.ylabel('Frequency')

                # Plot 2: Price variance if available
                if 'price_variance_pct' in df.columns:
                    plt.subplot(1, 3, 2)
                    plt.hist(df['price_variance_pct'], bins=30, alpha=0.7, color='orange', edgecolor='black')
                    plt.title('Price Variance Distribution')
                    plt.xlabel('Price Variance %')
                    plt.ylabel('Frequency')

                # Plot 3: Total value trends
                if 'total_value' in df.columns:
                    plt.subplot(1, 3, 3)
                    plt.boxplot(df['total_value'])
                    plt.title('Total Value Box Plot')
                    plt.ylabel('Total Value')

                plt.tight_layout()
                plt.show()

    def quality_delivery_correlation(self):
        """
        Analyze correlation between quality and delivery metrics
        """
        print("\nQUALITY-DELIVERY CORRELATION ANALYSIS:")

        # Find sheets with both quality and delivery metrics
        for sheet_name, df in self.sheets_data.items():
            quality_cols = [col for col in df.columns if 'quality' in col.lower()]
            delivery_cols = [col for col in df.columns if any(word in col.lower() for word in ['late', 'delivery', 'days'])]

            if quality_cols and delivery_cols:
                print(f"\n--- {sheet_name} Quality-Delivery Correlation ---")

                # Create correlation matrix
                relevant_cols = quality_cols + delivery_cols + ['waste_value'] if 'waste_value' in df.columns else quality_cols + delivery_cols
                corr_data = df[relevant_cols].select_dtypes(include=[np.number])

                if len(corr_data.columns) > 1:
                    plt.figure(figsize=(10, 8))
                    correlation_matrix = corr_data.corr()
                    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0,
                                square=True, fmt='.2f')
                    plt.title(f'{sheet_name} - Quality-Delivery Correlation Matrix')
                    plt.tight_layout()
                    plt.show()

                    # Print strong correlations
                    print("Strong correlations (|r| > 0.5):")
                    for i in range(len(correlation_matrix.columns)):
                        for j in range(i+1, len(correlation_matrix.columns)):
```

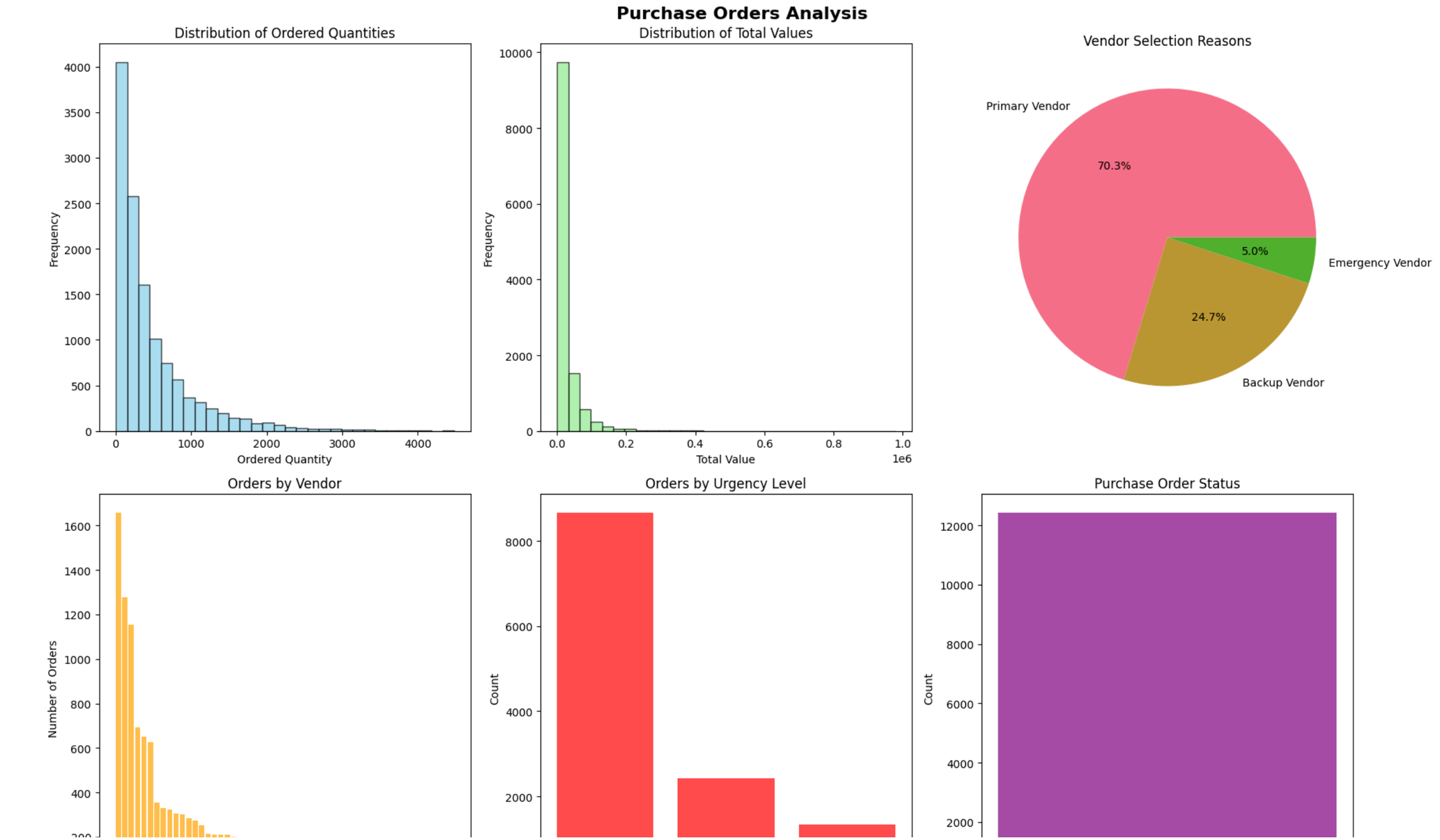
```
                        corr_val = correlation_matrix.iloc[i, j]
                        if abs(corr_val) > 0.5:
                            print(f"  {correlation_matrix.columns[i]} vs {correlation_matrix.columns[j]}: {corr_val:.3f}")

    def generate_summary_report(self):
        """
        Generate a comprehensive summary report
        """
        print("="*80)
        print("EXECUTIVE SUMMARY REPORT")
        print("="*80)

        total_sheets = len(self.sheets_data)
        total_records = sum(len(df) for df in self.sheets_data.values())

        print(f"\nDATA OVERVIEW:")
        print(f"• Total sheets analyzed: {total_sheets}")
        print(f"• Total records across all sheets: {total_records}")

        # Calculate key metrics from each sheet
        for sheet_name, df in self.sheets_data.items():
            print(f"\n{sheet_name.upper()} SUMMARY:")
            print(f"  - Records: {len(df)}")
            print(f"  - Columns: {len(df.columns)}")
            print(f"  - Missing data: {df.isnull().sum().sum()} cells")

            # Sheet-specific metrics
            if 'total_value' in df.columns:
                print(f"  - Total value: ${df['total_value'].sum():,.2f}")
                print(f"  - Average value: ${df['total_value'].mean():,.2f}")

            if 'quality_rating' in df.columns:
                print(f"  - Average quality rating: {df['quality_rating'].mean():.2f}")

            if 'days_late' in df.columns:
                on_time_rate = (df['days_late'] == 0).sum() / len(df) * 100
                print(f"  - On-time delivery rate: {on_time_rate:.2f}%")

        print("\n" + "="*80)
        print("RECOMMENDATIONS:")
        print("="*80)
        print("1. Focus on vendors with high rejection rates")
        print("2. Investigate price variance patterns")
        print("3. Improve delivery time performance")
        print("4. Monitor quality ratings trends")
        print("5. Optimize inventory based on market conditions")

    def run_complete_eda(self):
        """
        Run complete EDA analysis
        """
        print("STARTING COMPREHENSIVE SUPPLY CHAIN EDA")
        print("="*80)

        # Convert datetime columns
        self.convert_datetime_columns()

        # Basic information
        self.basic_info()

        # Individual sheet analysis
        self.purchase_orders_analysis()
        self.delivery_analysis()
        self.batch_quality_analysis()

        # Comprehensive analysis
        self.comprehensive_analysis()

        # Summary report
        self.generate_summary_report()

# USAGE EXAMPLE:
# Replace 'your_file_path.xlsx' with your actual file path
def main():
    """
    Main function to run the EDA
    """
    # Example file path - replace with your actual path
    file_path = "/content/drive/MyDrive/CAPSTONE/Data/Supply_Chain.xlsx"  # Update this path
```

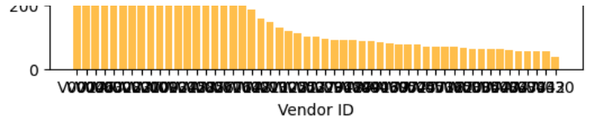
```
        try:
            # Initialize EDA
            eda = SupplyChainEDA(file_path)

            # Run complete analysis
            eda.run_complete_eda()

        except FileNotFoundError:
            print(f"File not found: {file_path}")
            print("Please update the file_path variable with your Excel file location")
        except Exception as e:
            print(f"Error during analysis: {e}")

    # ALTERNATIVE: Load from Google Drive (requires authentication)
    def load_from_google_drive():
        """
        Alternative method to load from Google Drive
        Requires Google Drive API setup
        """
        try:
            from google.colab import drive
            drive.mount('/content/drive')

            # Update this path to your file location in Google Drive
            file_path = "/content/drive/MyDrive/your_supply_chain_file.xlsx"

            eda = SupplyChainEDA(file_path)
            eda.run_complete_eda()

        except ImportError:
            print("Google Colab drive not available. Use local file path instead.")

    if __name__ == "__main__":
        main()

        # Uncomment the line below if running in Google Colab
        # load_from_google_drive()

    # ADDITIONAL UTILITY FUNCTIONS:

    def quick_sheet_preview(file_path, max_rows=5):
        """
        Quick preview of all sheets in the Excel file
        """
        try:
            excel_file = pd.ExcelFile(file_path)

            for sheet_name in excel_file.sheet_names:
                print(f"\n--- {sheet_name} PREVIEW ---")
                df = pd.read_excel(file_path, sheet_name=sheet_name)
                print(f"Shape: {df.shape}")
                print(df.head(max_rows))
                print("-" * 60)

        except Exception as e:
            print(f"Error previewing file: {e}")

    def export_analysis_results(eda_instance, output_path="supply_chain_analysis_results.xlsx"):
        """
        Export analysis results to Excel file
        """
        try:
            with pd.ExcelWriter(output_path, engine='openpyxl') as writer:
                for sheet_name, df in eda_instance.sheets_data.items():
                    # Basic statistics
                    numeric_df = df.select_dtypes(include=[np.number])
                    if not numeric_df.empty:
                        stats_df = numeric_df.describe()
                        stats_df.to_excel(writer, sheet_name=f"{sheet_name}_stats")

                    # Missing values summary
                    missing_df = pd.DataFrame({
                        'Column': df.columns,
                        'Missing_Count': df.isnull().sum(),
                        'Missing_Percentage': (df.isnull().sum() / len(df) * 100).round(2)
                    })
                    missing_df.to_excel(writer, sheet_name=f"{sheet_name}_missing", index=False)

            print(f"Analysis results exported to: {output_path}")
```

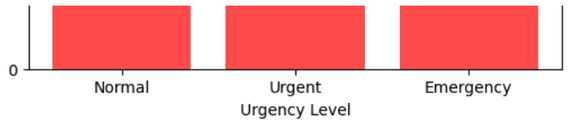
```
        except Exception as e:
            print(f"Error exporting results: {e}")

    # USAGE INSTRUCTIONS:
    print("""
    USAGE INSTRUCTIONS:
    ==================
    1. Update the file_path variable in main() function with your Excel file path
    2. Run the script: python supply_chain_eda.py
    3. For Google Colab: Uncomment the load_from_google_drive() call

    FEATURES:
    =========
    ✓ Multi-sheet Excel file support
    ✓ Automatic data type detection and conversion
    ✓ Comprehensive statistical analysis
    ✓ Multiple visualization types
    ✓ Vendor performance analysis
    ✓ Quality and delivery correlation
    ✓ Time series analysis
    ✓ Executive summary report
    ✓ Export results to Excel
    """)
```

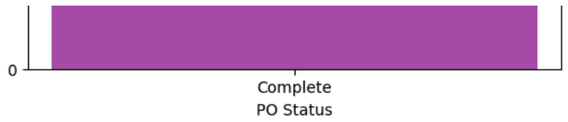
```
Available sheets: ['Purchase_Orders', 'Deliveries']
Loaded sheet 'Purchase_Orders' with shape: (12428, 12)
Loaded sheet 'Deliveries' with shape: (12428, 21)
STARTING COMPREHENSIVE SUPPLY CHAIN EDA
================================================================================
Converted po_date to datetime in Purchase_Orders
Converted requested_delivery_date to datetime in Purchase_Orders
Converted delivery_date to datetime in Deliveries
Converted expiry_date to datetime in Deliveries
================================================================================
BASIC DATA INFORMATION
================================================================================

--- PURCHASE_ORDERS SHEET ---
Shape: (12428, 12)
Columns: ['po_id', 'hospital_id', 'vendor_id', 'sku_id', 'po_date', 'requested_delivery_date', 'ordered_quantity', 'unit_price', 'total_value', 'po_status', 'urgency_level', 'vendor_selection_reason']
Data types:
po_id                            object
hospital_id                      object
vendor_id                        object
sku_id                           object
po_date                  datetime64[ns]
requested_delivery_date  datetime64[ns]
ordered_quantity                float64
unit_price                      float64
total_value                     float64
po_status                        object
urgency_level                    object
vendor_selection_reason          object
dtype: object
Missing values:
po_id                      0
hospital_id                0
vendor_id                  0
sku_id                     0
po_date                    0
requested_delivery_date    0
ordered_quantity           0
unit_price                 0
total_value                0
po_status                  0
urgency_level              0
vendor_selection_reason    0
dtype: int64
Duplicate rows: 0
------------------------------------------------------------

--- DELIVERIES SHEET ---
Shape: (12428, 21)
Columns: ['delivery_id', 'po_id', 'hospital_id', 'vendor_id', 'sku_id', 'delivery_date', 'delivered_quantity', 'accepted_quantity', 'rejected_quantity', 'rejection_reason', 'actual_unit_price', 'price_variance_pct', 'batch_number', 'expiry_date', 'delivery_status', 'days_lat
Data types:
delivery_id                 object
po_id                       object
hospital_id                 object
vendor_id                   object
sku_id                      object
delivery_date       datetime64[ns]
delivered_quantity         float64
accepted_quantity          float64
rejected_quantity          float64
rejection_reason            object
actual_unit_price          float64
price_variance_pct         float64
batch_number                object
expiry_date         datetime64[ns]
delivery_status             object
days_late                    int64
quality_rating             float64
expected_waste_qty         float64
waste_value                float64
market_condition            object
delivery_notes              object
dtype: object
Missing values:
delivery_id              0
po_id                    0
hospital_id              0
vendor_id                0
sku_id                   0
delivery_date            0
delivered_quantity       0
accepted_quantity        0
rejected_quantity        0
rejection_reason     12027
actual_unit_price        0
price_variance_pct       0
batch_number             0
```

```
batch_number              0
expiry_date               0
delivery_status           0
days_late                 0
quality_rating            0
expected_waste_qty        0
waste_value               0
market_condition          0
delivery_notes            0
dtype: int64
Duplicate rows: 0
------------------------------------------------------------
================================================================================
PURCHASE ORDERS ANALYSIS
================================================================================

BASIC STATISTICS:
       ordered_quantity    unit_price    total_value
count      12428.000000  12428.000000   12428.000000
mean         469.018978     56.532292   25424.139868
std          549.054330     52.672464   47165.524148
min            2.890000      2.100000      38.720000
25%          111.547500     22.270000    3600.825000
50%          270.195000     39.850000   10019.885000
75%          611.030000     70.390000   27466.510000
max         4481.210000    293.530000  979022.730000
```

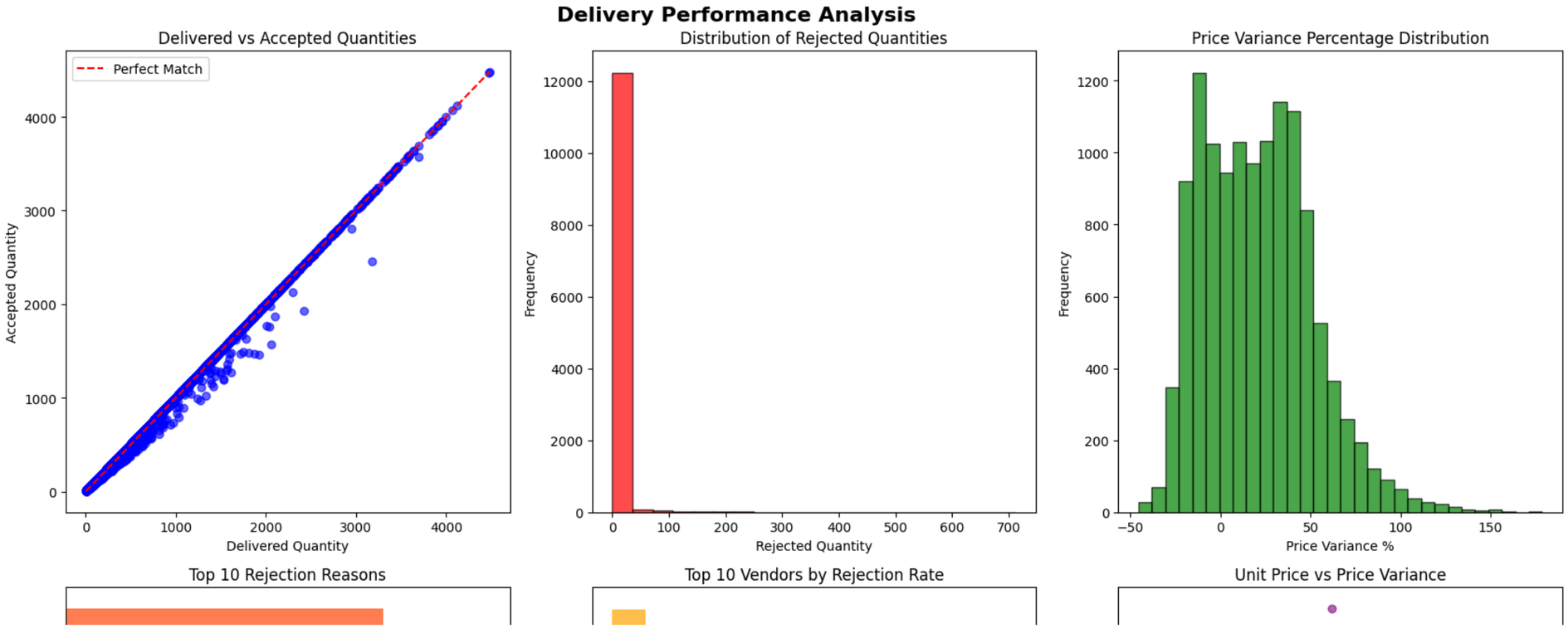


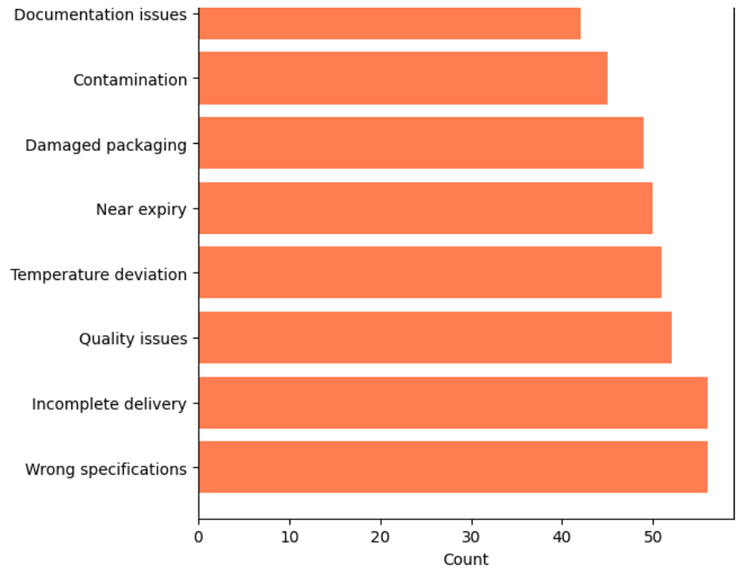


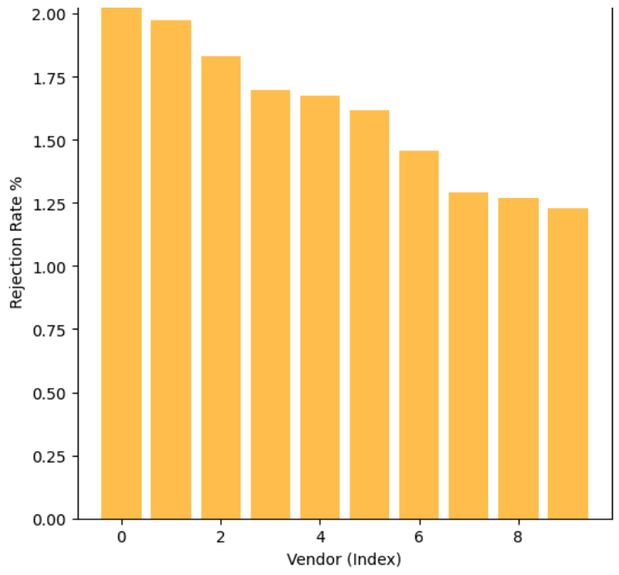


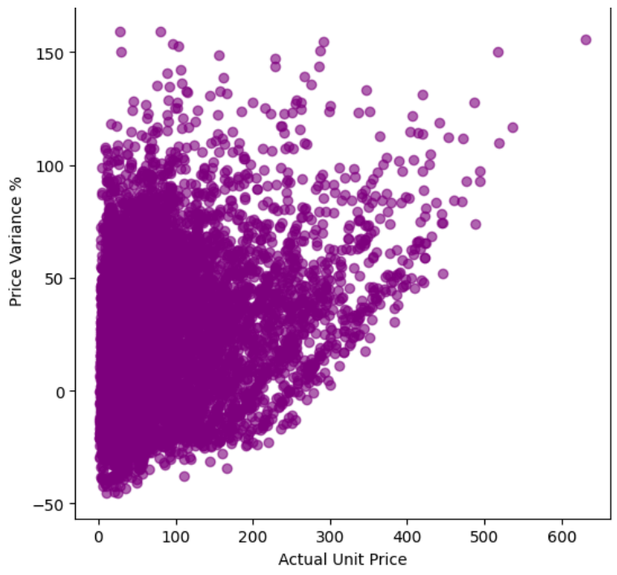



```
KEY INSIGHTS - PURCHASE ORDERS:
• Total number of purchase orders: 12428
• Total value of all orders: $315,971,210.28
• Average order value: $25,424.14
• Most common vendor: V001
• Most common urgency level: Normal
• Most common vendor selection reason: Primary Vendor
================================================================================
DELIVERY ANALYSIS
================================================================================

DELIVERY STATISTICS:
       delivered_quantity  accepted_quantity  rejected_quantity  \
count        12428.000000       12428.000000       12428.000000
mean           459.655824         457.641885           2.013939
std            539.939736         538.013125          18.877999
min              2.890000           2.890000           0.000000
25%            108.517500         108.202500           0.000000
50%            264.895000         263.845000           0.000000
75%            598.510000         594.962500           0.000000
max           4481.210000        4481.210000         712.500000

       actual_unit_price  price_variance_pct     days_late  quality_rating  \
count       12428.000000        12428.000000  12428.000000    12428.000000
mean           69.238204           20.625161      7.940457        4.119673
std            70.528506           30.270973     11.725633        0.648726
min             1.660000          -45.500000      0.000000        1.000000
25%            25.350000           -4.600000      1.000000        3.900000
50%            45.870000           19.200000      4.000000        4.300000
75%            83.635000           40.400000      9.000000        4.600000
max           631.500000          178.700000    156.000000        5.000000

       expected_waste_qty    waste_value
count        12428.000000   12428.000000
mean            35.930251    2027.907391
std             57.608470    5187.443004
min              0.000000       0.030000
25%              5.230000     171.017500
50%             15.465000     581.025000
75%             41.650000    1818.762500
max            753.600000  179577.310000
```

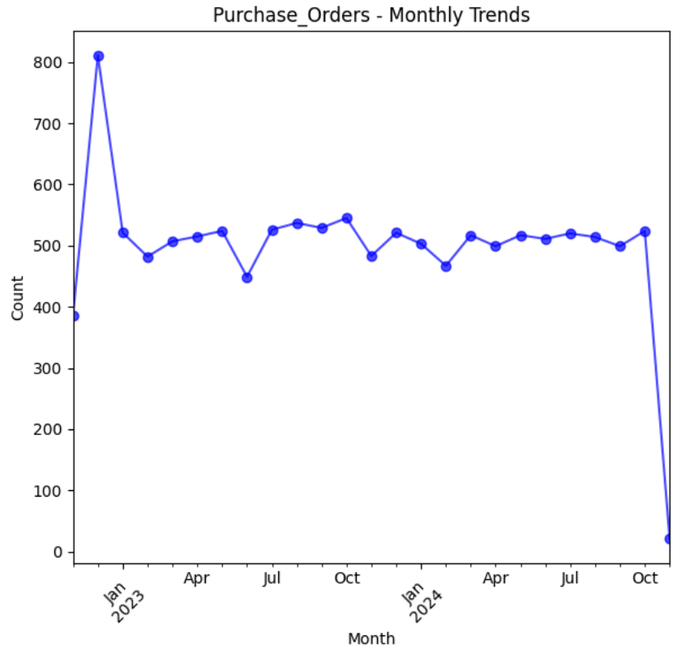


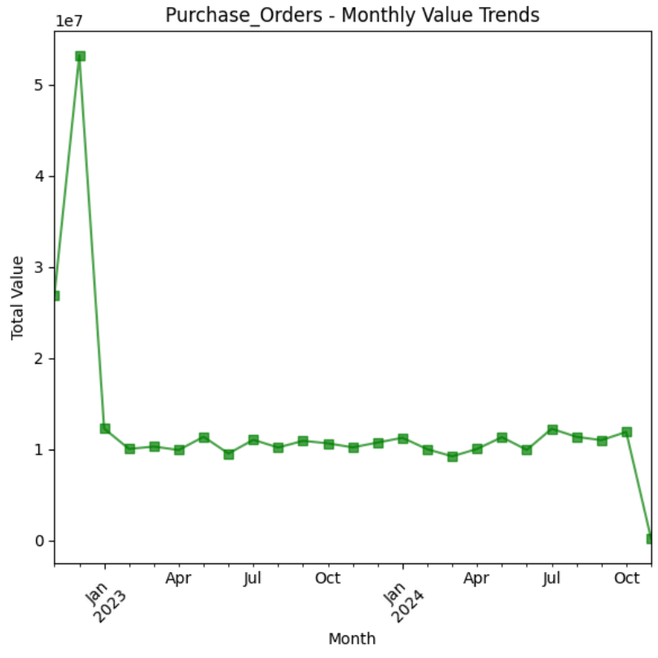




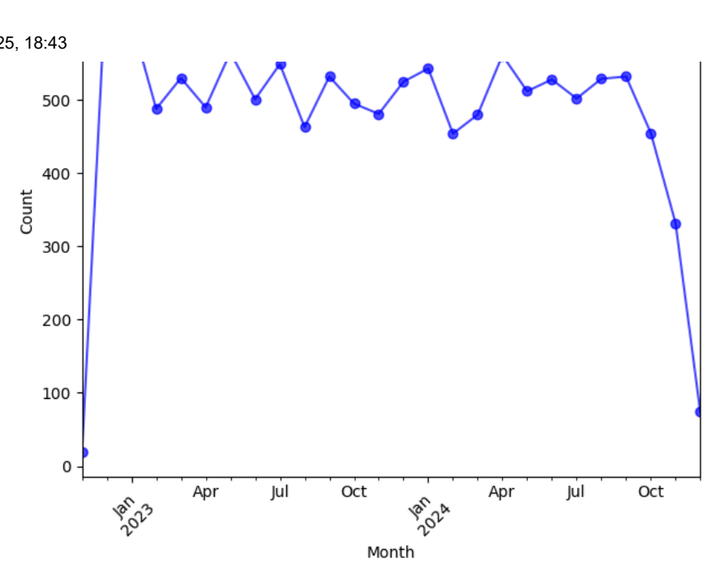



```
KEY INSIGHTS - DELIVERIES:
• Total deliveries: 12428
• Average delivery acceptance rate: 99.56%
• Average price variance: 20.63%
• Most common rejection reason: Incomplete delivery
Batch quality data not available
================================================================================
COMPREHENSIVE SUPPLY CHAIN ANALYSIS
================================================================================

TIME SERIES ANALYSIS:

--- Purchase_Orders Time Analysis ---
```





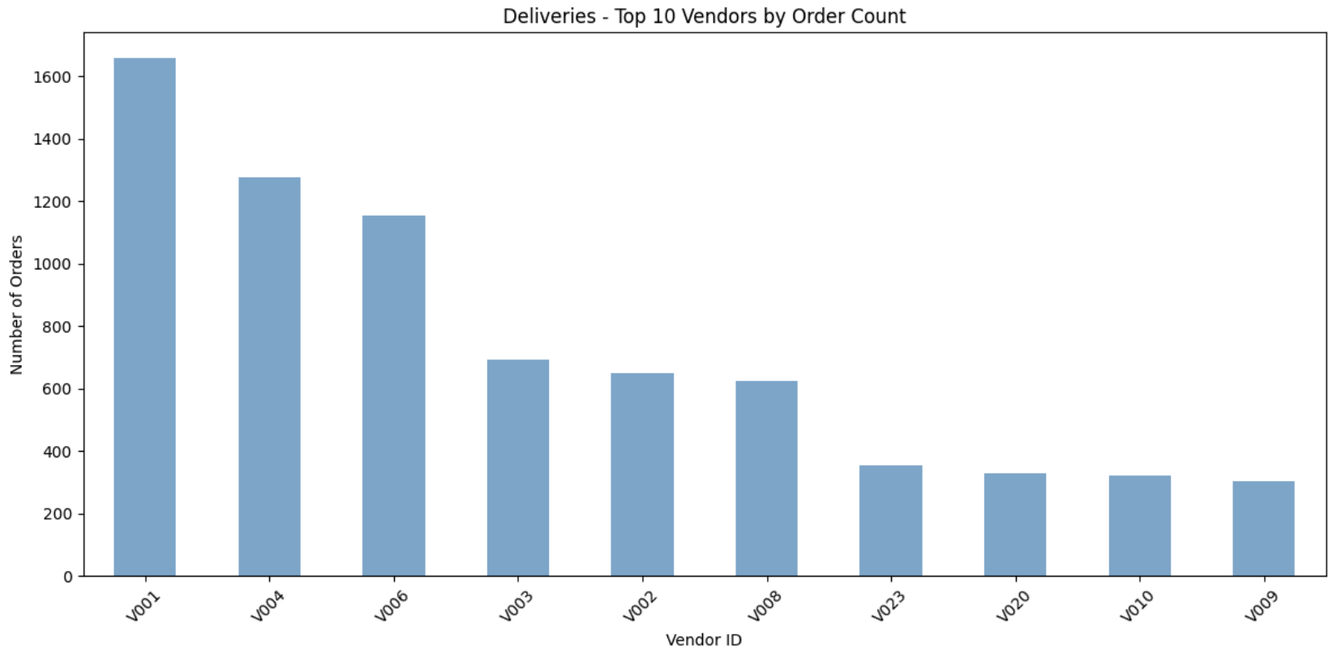

```
--- Deliveries Time Analysis ---
```



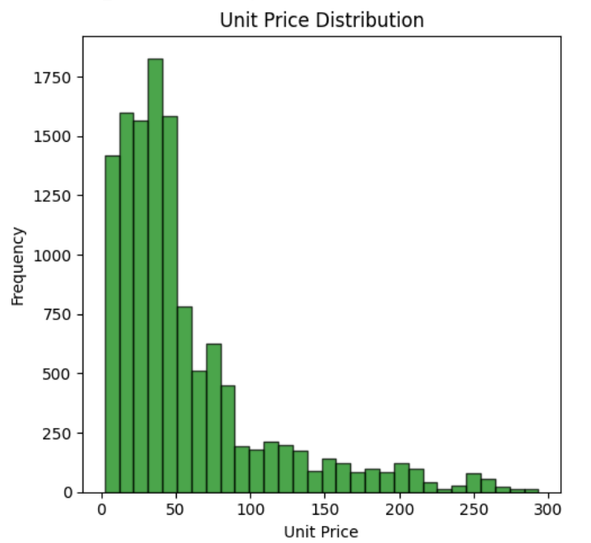

```
VENDOR PERFORMANCE ANALYSIS:

--- Purchase_Orders Vendor Analysis ---
Top 5 vendors by order count:
vendor_id
V001    1658
V004    1279
V006    1153
V003     693
V002     652
Name: count, dtype: int64

Top 5 vendors by average order value:
vendor_id
V018    42441.288333
V014    33702.943266
V044    33440.763175
V009    32187.370163
V028    30247.405810
Name: total_value, dtype: float64
```

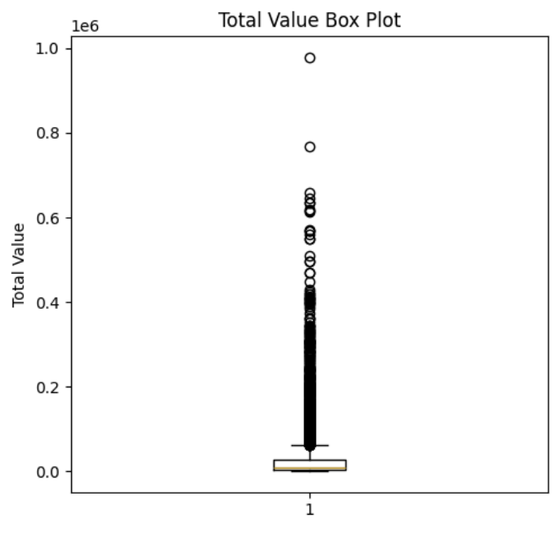

```
--- Deliveries Vendor Analysis ---
Top 5 vendors by order count:
vendor_id
V001    1658
V004    1279
```

```
V006    1153
V003     693
V002     652
Name: count, dtype: int64
```

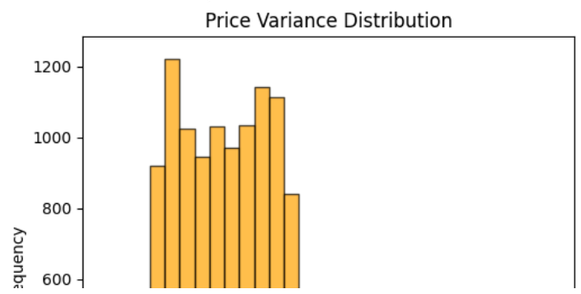


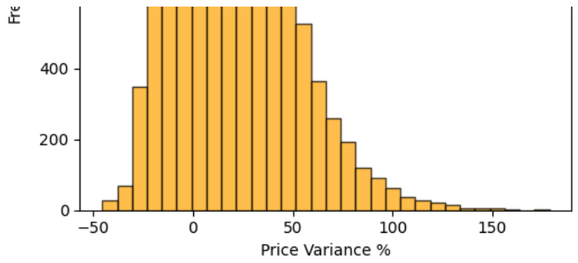

```
COST EFFICIENCY ANALYSIS:

--- Purchase_Orders Cost Analysis ---
```



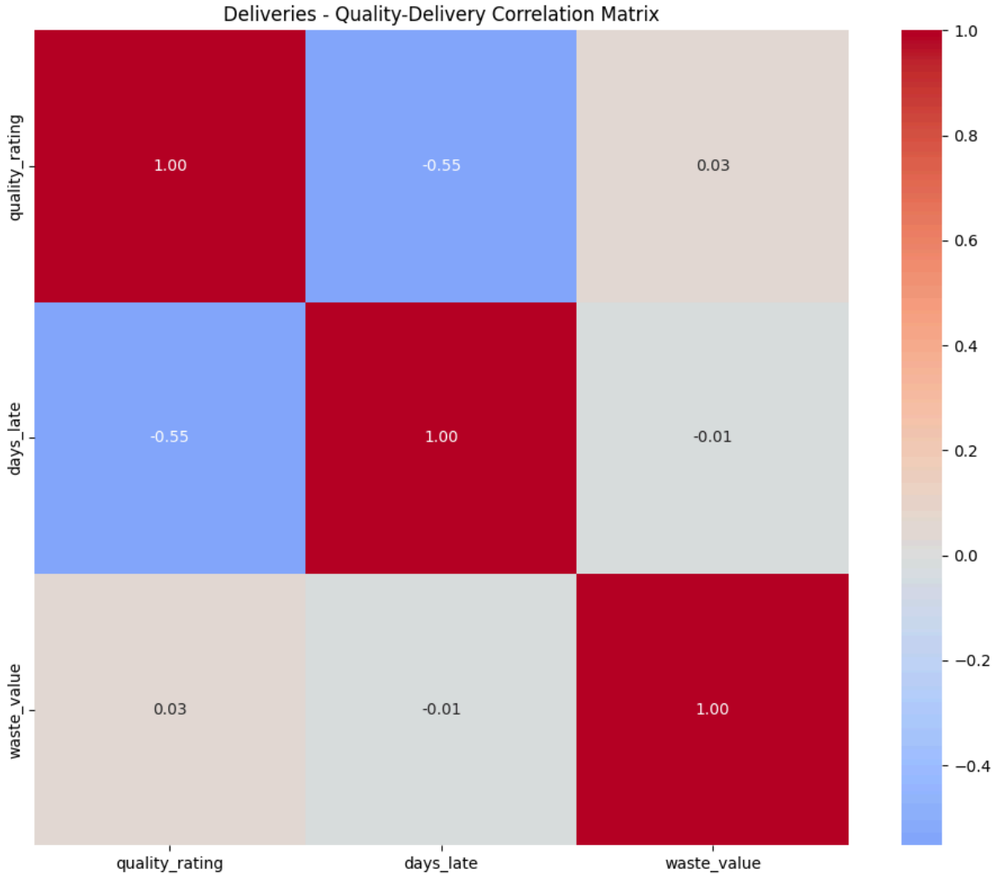



```
--- Deliveries Cost Analysis ---
```






```
QUALITY-DELIVERY CORRELATION ANALYSIS:

--- Deliveries Quality-Delivery Correlation ---
```




```
Strong correlations (|r| > 0.5):
  quality_rating vs days_late: -0.552
================================================================================
EXECUTIVE SUMMARY REPORT
================================================================================

DATA OVERVIEW:
• Total sheets analyzed: 2
• Total records across all sheets: 24856

PURCHASE_ORDERS SUMMARY:
  - Records: 12428
  - Columns: 12
  - Missing data: 0 cells
  - Total value: $315,971,210.28
  - Average value: $25,424.14

DELIVERIES SUMMARY:
  - Records: 12428
  - Columns: 21
  - Missing data: 12027 cells
  - Average quality rating: 4.12
  - On-time delivery rate: 17.90%
```

```
================================================================================
RECOMMENDATIONS:
================================================================================
1. Focus on vendors with high rejection rates
2. Investigate price variance patterns
3. Improve delivery time performance
4. Monitor quality ratings trends
5. Optimize inventory based on market conditions

USAGE INSTRUCTIONS:
==================
1. Update the file_path variable in main() function with your Excel file path
2. Run the script: python supply_chain_eda.py
3. For Google Colab: Uncomment the load_from_google_drive() call

FEATURES:
=========
✓ Multi-sheet Excel file support
✓ Automatic data type detection and conversion
✓ Comprehensive statistical analysis
✓ Multiple visualization types
✓ Vendor performance analysis
✓ Quality and delivery correlation
✓ Time series analysis
✓ Executive summary report
✓ Export results to Excel
```

Transaction 2023

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# Set up plotting style
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)

class HospitalDataEDA:
    def __init__(self, file_path):
        """
        Initialize the EDA class with the Excel file path
        """
        self.file_path = file_path
        self.data = {}
        self.combined_data = None

    def load_data(self):
        """
        Load data from Excel file with multiple sheets
        """
        print("Loading data from Excel file...")
        try:
            # Read all sheets from Excel file
            excel_file = pd.ExcelFile(self.file_path)
            print(f"Available sheets: {excel_file.sheet_names}")

            # Load all sheets
            for sheet_name in excel_file.sheet_names:
                self.data[sheet_name] = pd.read_excel(self.file_path, sheet_name=sheet_name)
                print(f"Loaded sheet '{sheet_name}' with shape: {self.data[sheet_name].shape}")

            # If multiple sheets exist, combine them (assuming same structure)
            if len(self.data) > 1:
                self.combined_data = pd.concat(self.data.values(), ignore_index=True)
                print(f"Combined data shape: {self.combined_data.shape}")
            else:
                self.combined_data = list(self.data.values())[0]

        except Exception as e:
            print(f"Error loading data: {e}")
            return None

    def basic_info(self):
        """
        Display basic information about the dataset
        """
        print("="*60)
        print("BASIC DATASET INFORMATION")
        print("="*60)

        df = self.combined_data

        print(f"Dataset Shape: {df.shape}")
        print(f"Number of Rows: {df.shape[0]:,}")
        print(f"Number of Columns: {df.shape[1]}")
        print("\nColumn Names and Data Types:")
        print("-" * 40)

        for col in df.columns:
            dtype = df[col].dtype
            null_count = df[col].isnull().sum()
            null_pct = (null_count / len(df)) * 100
            # Convert dtype to string before formatting
            print(f"{col:<20} | {str(dtype):<10} | Nulls: {null_count:>6} ({null_pct:>5.1f}%)")


        print(f"\nMemory Usage: {df.memory_usage(deep=True).sum() / 1024**2:.2f} MB")
```

```
    def data_quality_check(self):
        """
        Perform data quality checks
        """
        print("\n" + "="*60)
        print("DATA QUALITY ANALYSIS")
        print("="*60)

        df = self.combined_data

        # Missing values analysis
        print("\nMissing Values Analysis:")
        print("-" * 30)
        missing_data = df.isnull().sum()
        missing_pct = (missing_data / len(df)) * 100
        missing_df = pd.DataFrame({
            'Missing_Count': missing_data,
            'Missing_Percentage': missing_pct
        }).sort_values('Missing_Percentage', ascending=False)

        print(missing_df[missing_df['Missing_Count'] > 0])

        # Duplicate records
        duplicates = df.duplicated().sum()
        print(f"\nDuplicate Records: {duplicates:,} ({(duplicates/len(df))*100:.2f}%)")

        # Data type consistency
        print("\nData Type Issues:")
        print("-" * 20)

        # Check for date columns
        date_columns = ['transaction_date', 'transaction_time']
        for col in date_columns:
            if col in df.columns:
                try:
                    pd.to_datetime(df[col])
                    print(f"✓ {col}: Valid date format")
                except:
                    print(f"✗ {col}: Invalid date format detected")

        # Check for negative values in cost columns
        cost_columns = ['unit_cost', 'total_cost', 'revenue_lost']
        for col in cost_columns:
            if col in df.columns:
                negative_count = (df[col] < 0).sum()
                if negative_count > 0:
                    print(f"⚠ {col}: {negative_count} negative values found")
                else:
                    print(f"✓ {col}: No negative values")

    def descriptive_statistics(self):
        """
        Generate descriptive statistics for numerical columns
        """
        print("\n" + "="*60)
        print("DESCRIPTIVE STATISTICS")
        print("="*60)

        df = self.combined_data

        # Numerical columns
        numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()

        if numerical_cols:
            print("\nNumerical Variables Summary:")
            print("-" * 40)
            desc_stats = df[numerical_cols].describe()
            print(desc_stats.round(2))

        # Categorical columns
        categorical_cols = df.select_dtypes(include=['object']).columns.tolist()

        if categorical_cols:
            print(f"\nCategorical Variables Summary:")
            print("-" * 40)
            for col in categorical_cols[:10]:  # Show first 10 categorical columns
                unique_count = df[col].nunique()
                most_common = df[col].mode().iloc[0] if len(df[col].mode()) > 0 else 'N/A'
                print(f"{col:<25} | Unique: {unique_count:>6} | Most Common: {most_common}")
```

```
    def transaction_analysis(self):
        """
        Analyze transaction patterns
        """
        print("\n" + "="*60)
        print("TRANSACTION PATTERN ANALYSIS")
        print("="*60)

        df = self.combined_data.copy()

        # Convert transaction_date to datetime if not already
        if 'transaction_date' in df.columns:
            df['transaction_date'] = pd.to_datetime(df['transaction_date'])
            df['month'] = df['transaction_date'].dt.month
            df['day_of_week'] = df['transaction_date'].dt.day_name()
            df['hour'] = pd.to_datetime(df['transaction_time'], format='%H:%M', errors='coerce').dt.hour

        # Transaction volume analysis
        print("\nTransaction Volume by Month:")
        if 'month' in df.columns:
            monthly_volume = df['month'].value_counts().sort_index()
            print(monthly_volume)

        print("\nTransaction Volume by Day of Week:")
        if 'day_of_week' in df.columns:
            daily_volume = df['day_of_week'].value_counts()
            print(daily_volume)

        # Transaction type analysis
        if 'transaction_type' in df.columns:
            print(f"\nTransaction Types:")
            type_counts = df['transaction_type'].value_counts()
            print(type_counts)
            print(f"\nTransaction Type Percentages:")
            print((type_counts / len(df) * 100).round(2))

        # Department analysis
        if 'dept_id' in df.columns:
            print(f"\nTop 10 Departments by Transaction Volume:")
            dept_volume = df['dept_id'].value_counts().head(10)
            print(dept_volume)

    def financial_analysis(self):
        """
        Analyze financial metrics
        """
        print("\n" + "="*60)
        print("FINANCIAL ANALYSIS")
        print("="*60)

        df = self.combined_data

        # Cost analysis
        if 'total_cost' in df.columns:
            total_revenue = df['total_cost'].sum()
            avg_transaction = df['total_cost'].mean()
            median_transaction = df['total_cost'].median()

            print(f"Total Revenue: ${total_revenue:,.2f}")
            print(f"Average Transaction Value: ${avg_transaction:,.2f}")
            print(f"Median Transaction Value: ${median_transaction:,.2f}")

            # Revenue by transaction type
            if 'transaction_type' in df.columns:
                print(f"\nRevenue by Transaction Type:")
                revenue_by_type = df.groupby('transaction_type')['total_cost'].agg(['sum', 'mean', 'count'])
                revenue_by_type['sum'] = revenue_by_type['sum'].apply(lambda x: f"${x:,.2f}")
                revenue_by_type['mean'] = revenue_by_type['mean'].apply(lambda x: f"${x:,.2f}")
                print(revenue_by_type)

        # Revenue lost analysis
        if 'revenue_lost' in df.columns:
            total_lost = df['revenue_lost'].sum()
            print(f"\nTotal Revenue Lost: ${total_lost:,.2f}")

            if 'bounce_reason' in df.columns:
                print(f"\nRevenue Lost by Bounce Reason:")
                lost_by_reason = df.groupby('bounce_reason')['revenue_lost'].sum().sort_values(ascending=False)
```

```
                for reason, amount in lost_by_reason.head(5).items():
                    print(f"{reason}: ${amount:,.2f}")

    def adherence_analysis(self):
        """
        Analyze medication/treatment adherence
        """
        print("\n" + "="*60)
        print("ADHERENCE ANALYSIS")
        print("="*60)

        df = self.combined_data

        if 'formulary_adherent' in df.columns:
            adherence_rate = (df['formulary_adherent'] == True).mean() * 100
            print(f"Overall Formulary Adherence Rate: {adherence_rate:.2f}%")

            # Adherence by urgency level
            if 'urgency_level' in df.columns:
                print(f"\nAdherence Rate by Urgency Level:")
                adherence_by_urgency = df.groupby('urgency_level')['formulary_adherent'].mean() * 100
                for level, rate in adherence_by_urgency.items():
                    print(f"{level}: {rate:.2f}%")

        if 'adherence_impact_pct' in df.columns:
            avg_impact = df['adherence_impact_pct'].mean()
            print(f"\nAverage Adherence Impact: {avg_impact:.2f}%")

            # High impact transactions
            high_impact = df[df['adherence_impact_pct'] > 30]
            print(f"High Impact Transactions (>30%): {len(high_impact):,} ({len(high_impact)/len(df)*100:.2f}%)")

    def create_visualizations(self):
        """
        Create comprehensive visualizations
        """
        print("\n" + "="*60)
        print("GENERATING VISUALIZATIONS")
        print("="*60)

        df = self.combined_data.copy()

        # Convert date column
        if 'transaction_date' in df.columns:
            df['transaction_date'] = pd.to_datetime(df['transaction_date'])
            df['month'] = df['transaction_date'].dt.month
            df['day_of_week'] = df['transaction_date'].dt.day_name()

        # Create subplots
        fig = plt.figure(figsize=(20, 15))

        # 1. Transaction Volume Over Time
        if 'transaction_date' in df.columns:
            plt.subplot(3, 3, 1)
            daily_volume = df['transaction_date'].dt.date.value_counts().sort_index()
            plt.plot(daily_volume.index, daily_volume.values, linewidth=2)
            plt.title('Daily Transaction Volume', fontsize=14, fontweight='bold')
            plt.xticks(rotation=45)
            plt.ylabel('Number of Transactions')
            plt.grid(True, alpha=0.3)

        # 2. Transaction Type Distribution
        if 'transaction_type' in df.columns:
            plt.subplot(3, 3, 2)
            type_counts = df['transaction_type'].value_counts()
            colors = plt.cm.Set3(np.linspace(0, 1, len(type_counts)))
            plt.pie(type_counts.values, labels=type_counts.index, autopct='%1.1f%%', colors=colors)
            plt.title('Transaction Type Distribution', fontsize=14, fontweight='bold')

        # 3. Total Cost Distribution
        if 'total_cost' in df.columns:
            plt.subplot(3, 3, 3)
            plt.hist(df['total_cost'], bins=50, alpha=0.7, color='skyblue', edgecolor='black')
            plt.title('Total Cost Distribution', fontsize=14, fontweight='bold')
            plt.xlabel('Total Cost ($)')
            plt.ylabel('Frequency')
            plt.grid(True, alpha=0.3)

        # 4. Urgency Level Analysis
```

```
        if 'urgency_level' in df.columns:
            plt.subplot(3, 3, 4)
            urgency_counts = df['urgency_level'].value_counts()
            plt.bar(urgency_counts.index, urgency_counts.values, color='coral', alpha=0.8)
            plt.title('Cases by Urgency Level', fontsize=14, fontweight='bold')
            plt.xlabel('Urgency Level')
            plt.ylabel('Number of Cases')
            plt.grid(True, alpha=0.3)

        # 5. Department Performance
        if 'dept_id' in df.columns and 'total_cost' in df.columns:
            plt.subplot(3, 3, 5)
            dept_revenue = df.groupby('dept_id')['total_cost'].sum().sort_values(ascending=False).head(10)
            plt.barh(range(len(dept_revenue)), dept_revenue.values, color='lightgreen', alpha=0.8)
            plt.yticks(range(len(dept_revenue)), dept_revenue.index)
            plt.title('Top 10 Departments by Revenue', fontsize=14, fontweight='bold')
            plt.xlabel('Total Revenue ($)')
            plt.grid(True, alpha=0.3)

        # 6. Adherence Rate Analysis
        if 'formulary_adherent' in df.columns:
            plt.subplot(3, 3, 6)
            adherence_counts = df['formulary_adherent'].value_counts()
            labels = ['Non-Adherent', 'Adherent']
            plt.pie(adherence_counts.values, labels=labels, autopct='%1.1f%%',
                   colors=['lightcoral', 'lightblue'])
            plt.title('Formulary Adherence Rate', fontsize=14, fontweight='bold')

        # 7. Cost vs Quantity Relationship
        if 'quantity' in df.columns and 'total_cost' in df.columns:
            plt.subplot(3, 3, 7)
            # Sample data for better visualization if dataset is large
            sample_df = df.sample(n=min(1000, len(df)))
            plt.scatter(sample_df['quantity'], sample_df['total_cost'], alpha=0.6, color='purple')
            plt.title('Cost vs Quantity Relationship', fontsize=14, fontweight='bold')
            plt.xlabel('Quantity')
            plt.ylabel('Total Cost ($)')
            plt.grid(True, alpha=0.3)

        # 8. Bounce Reason Analysis
        if 'bounce_reason' in df.columns:
            plt.subplot(3, 3, 8)
            bounce_reasons = df['bounce_reason'].value_counts().head(5)
            plt.barh(range(len(bounce_reasons)), bounce_reasons.values, color='salmon', alpha=0.8)
            plt.yticks(range(len(bounce_reasons)), bounce_reasons.index)
            plt.title('Top 5 Bounce Reasons', fontsize=14, fontweight='bold')
            plt.xlabel('Frequency')
            plt.grid(True, alpha=0.3)

        # 9. Monthly Revenue Trend
        if 'month' in df.columns and 'total_cost' in df.columns:
            plt.subplot(3, 3, 9)
            monthly_revenue = df.groupby('month')['total_cost'].sum()
            plt.plot(monthly_revenue.index, monthly_revenue.values, marker='o', linewidth=3, markersize=8)
            plt.title('Monthly Revenue Trend', fontsize=14, fontweight='bold')
            plt.xlabel('Month')
            plt.ylabel('Total Revenue ($)')
            plt.grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()

    def correlation_analysis(self):
        """
        Analyze correlations between numerical variables
        """
        print("\n" + "="*60)
        print("CORRELATION ANALYSIS")
        print("="*60)

        df = self.combined_data

        # Select numerical columns
        numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()

        if len(numerical_cols) > 1:
            # Calculate correlation matrix
            corr_matrix = df[numerical_cols].corr()
```

```
            # Create heatmap
            plt.figure(figsize=(12, 10))
            mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
            sns.heatmap(corr_matrix, mask=mask, annot=True, cmap='RdYlBu_r', center=0,
                        square=True, linewidths=0.5, cbar_kws={"shrink": .8})
            plt.title('Correlation Matrix of Numerical Variables', fontsize=16, fontweight='bold')
            plt.tight_layout()
            plt.show()

            # Strong correlations
            print("\nStrong Correlations (|r| > 0.7):")
            print("-" * 35)
            strong_corr = []
            for i in range(len(corr_matrix.columns)):
                for j in range(i+1, len(corr_matrix.columns)):
                    corr_val = corr_matrix.iloc[i, j]
                    if abs(corr_val) > 0.7:
                        strong_corr.append((corr_matrix.columns[i], corr_matrix.columns[j], corr_val))

            if strong_corr:
                for var1, var2, corr_val in strong_corr:
                    print(f"{var1} <-> {var2}: {corr_val:.3f}")
            else:
                print("No strong correlations found")

    def advanced_insights(self):
        """
        Generate advanced business insights
        """
        print("\n" + "="*60)
        print("ADVANCED BUSINESS INSIGHTS")
        print("="*60)

        df = self.combined_data

        # Patient behavior analysis
        if 'patient_id' in df.columns:
            print("\nPatient Behavior Analysis:")
            print("-" * 30)

            # Patient transaction frequency
            patient_freq = df['patient_id'].value_counts()
            print(f"Average transactions per patient: {patient_freq.mean():.2f}")
            print(f"Patients with single transaction: {(patient_freq == 1).sum():,}")
            print(f"Most active patient transactions: {patient_freq.max()}")

            # High-value patients
            if 'total_cost' in df.columns:
                patient_value = df.groupby('patient_id')['total_cost'].sum().sort_values(ascending=False)
                print(f"\nTop 5 Patients by Total Spending:")
                for i, (patient, value) in enumerate(patient_value.head(5).items(), 1):
                    print(f"{i}. Patient {patient}: ${value:,.2f}")

        # Physician performance analysis
        if 'physician_id' in df.columns:
            print(f"\nPhysician Performance Analysis:")
            print("-" * 35)

            physician_stats = df.groupby('physician_id').agg({
                'total_cost': ['sum', 'mean', 'count'],
                'formulary_adherent': 'mean' if 'formulary_adherent' in df.columns else 'count'
            }).round(2)

            print("Top 5 Physicians by Total Revenue:")
            top_physicians = physician_stats['total_cost']['sum'].sort_values(ascending=False).head(5)
            for physician, revenue in top_physicians.items():
                print(f"Physician {physician}: ${revenue:,.2f}")

        # Seasonal patterns
        if 'seasonal_complexity_factor' in df.columns:
            print(f"\nSeasonal Complexity Analysis:")
            print("-" * 30)
            seasonal_avg = df['seasonal_complexity_factor'].mean()
            print(f"Average Seasonal Complexity Factor: {seasonal_avg:.3f}")

            high_complexity = df[df['seasonal_complexity_factor'] > seasonal_avg]
            print(f"High Complexity Transactions: {len(high_complexity):,} ({len(high_complexity)/len(df)*100:.2f}%)")

    def create_dashboard_visualizations(self):
```

```
        """
        Create additional dashboard-style visualizations
        """
        print("\n" + "="*60)
        print("CREATING DASHBOARD VISUALIZATIONS")
        print("="*60)

        df = self.combined_data.copy()

        # Convert date if needed
        if 'transaction_date' in df.columns:
            df['transaction_date'] = pd.to_datetime(df['transaction_date'])
            df['month'] = df['transaction_date'].dt.month
            df['day_of_week'] = df['transaction_date'].dt.day_name()

        fig, axes = plt.subplots(2, 2, figsize=(16, 12))

        # 1. Revenue Trend by Department
        if 'dept_id' in df.columns and 'total_cost' in df.columns:
            dept_revenue = df.groupby('dept_id')['total_cost'].sum().sort_values(ascending=False).head(8)
            axes[0, 0].bar(range(len(dept_revenue)), dept_revenue.values, color='steelblue', alpha=0.8)
            axes[0, 0].set_xticks(range(len(dept_revenue)))
            axes[0, 0].set_xticklabels(dept_revenue.index, rotation=45)
            axes[0, 0].set_title('Revenue by Department', fontsize=14, fontweight='bold')
            axes[0, 0].set_ylabel('Revenue ($)')
            axes[0, 0].grid(True, alpha=0.3)

        # 2. Adherence Impact Distribution
        if 'adherence_impact_pct' in df.columns:
            axes[0, 1].hist(df['adherence_impact_pct'], bins=30, color='orange', alpha=0.7, edgecolor='black')
            axes[0, 1].set_title('Adherence Impact Distribution', fontsize=14, fontweight='bold')
            axes[0, 1].set_xlabel('Adherence Impact (%)')
            axes[0, 1].set_ylabel('Frequency')
            axes[0, 1].grid(True, alpha=0.3)

        # 3. Bounced vs Non-Bounced Revenue
        if 'bounced' in df.columns and 'total_cost' in df.columns:
            bounce_revenue = df.groupby('bounced')['total_cost'].sum()
            axes[1, 0].bar(['Non-Bounced', 'Bounced'], bounce_revenue.values,
                           color=['lightgreen', 'lightcoral'], alpha=0.8)
            axes[1, 0].set_title('Revenue: Bounced vs Non-Bounced', fontsize=14, fontweight='bold')
            axes[1, 0].set_ylabel('Total Revenue ($)')
            axes[1, 0].grid(True, alpha=0.3)

        # 4. Urgency Level vs Average Cost
        if 'urgency_level' in df.columns and 'total_cost' in df.columns:
            urgency_cost = df.groupby('urgency_level')['total_cost'].mean()
            axes[1, 1].bar(urgency_cost.index, urgency_cost.values, color='mediumpurple', alpha=0.8)
            axes[1, 1].set_title('Average Cost by Urgency Level', fontsize=14, fontweight='bold')
            axes[1, 1].set_xlabel('Urgency Level')
            axes[1, 1].set_ylabel('Average Cost ($)')
            axes[1, 1].grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()

    def generate_summary_report(self):
        """
        Generate a comprehensive summary report
        """
        print("\n" + "="*80)
        print("EXECUTIVE SUMMARY REPORT")
        print("="*80)

        df = self.combined_data

        print(f"\n📊 DATASET OVERVIEW")
        print(f"   • Total Records: {len(df):,}")
        print(f"   • Time Period: {df['transaction_date'].min().strftime('%Y-%m-%d')} to {df['transaction_date'].max().strftime('%Y-%m-%d')}")
        print(f"   • Unique Patients: {df['patient_id'].nunique():,}")
        print(f"   • Unique Physicians: {df['physician_id'].nunique():,}")
        print(f"   • Departments: {df['dept_id'].nunique():,}")

        if 'total_cost' in df.columns:
            print(f"\n💰 FINANCIAL METRICS")
            print(f"   • Total Revenue: ${df['total_cost'].sum():,.2f}")
            print(f"   • Average Transaction: ${df['total_cost'].mean():,.2f}")
            print(f"   • Revenue Range: ${df['total_cost'].min():.2f} - ${df['total_cost'].max():,.2f}")
```

```
            if 'revenue_lost' in df.columns:
                total_lost = df['revenue_lost'].sum()
                print(f"   • Total Revenue Lost: ${total_lost:,.2f}")
                print(f"   • Revenue Loss Rate: {(total_lost/df['total_cost'].sum())*100:.2f}%")

        if 'formulary_adherent' in df.columns:
            adherence_rate = (df['formulary_adherent'] == True).mean() * 100
            print(f"\n🏥 CLINICAL METRICS")
            print(f"   • Formulary Adherence Rate: {adherence_rate:.2f}%")

            if 'urgency_level' in df.columns:
                emergency_rate = (df['urgency_level'] == 'Emergency').mean() * 100
                print(f"   • Emergency Cases: {emergency_rate:.2f}%")

        if 'bounced' in df.columns:
            bounce_rate = (df['bounced'] == True).mean() * 100
            print(f"\n⚠️  OPERATIONAL METRICS")
            print(f"   • Transaction Bounce Rate: {bounce_rate:.2f}%")

            if 'bounce_reason' in df.columns:
                top_bounce_reason = df['bounce_reason'].mode().iloc[0] if len(df['bounce_reason'].mode()) > 0 else 'N/A'
                print(f"   • Top Bounce Reason: {top_bounce_reason}")

        print(f"\n🔍 KEY INSIGHTS")
        insights = []

        # High-value transaction insight
        if 'total_cost' in df.columns:
            high_value_threshold = df['total_cost'].quantile(0.9)
            high_value_count = (df['total_cost'] > high_value_threshold).sum()
            insights.append(f"10% of transactions (${high_value_threshold:.2f}+) represent high-value cases")

        # Adherence insight
        if 'formulary_adherent' in df.columns and 'total_cost' in df.columns:
            adherent_avg_cost = df[df['formulary_adherent'] == True]['total_cost'].mean()
            non_adherent_avg_cost = df[df['formulary_adherent'] == False]['total_cost'].mean()
            if adherent_avg_cost != non_adherent_avg_cost:
                cost_diff = abs(adherent_avg_cost - non_adherent_avg_cost)
                insights.append(f"Cost difference between adherent and non-adherent cases: ${cost_diff:.2f}")

        # Department insight
        if 'dept_id' in df.columns:
            busiest_dept = df['dept_id'].mode().iloc[0]
            dept_volume = df['dept_id'].value_counts().iloc[0]
            insights.append(f"Busiest department ({busiest_dept}) handles {dept_volume:,} transactions")

        for i, insight in enumerate(insights, 1):
            print(f"   {i}. {insight}")

        print("\n" + "="*80)

    def run_complete_eda(self):
        """
        Run the complete EDA pipeline
        """
        print("🏥 HOSPITAL TRANSACTION DATA - EXPLORATORY DATA ANALYSIS")
        print("="*80)

        # Load data
        self.load_data()

        if self.combined_data is None:
            print("❌ Failed to load data. Please check the file path.")
            return

        # Run all analyses
        self.basic_info()
        self.data_quality_check()
        self.descriptive_statistics()
        self.transaction_analysis()
        self.financial_analysis()
        self.adherence_analysis()
        self.correlation_analysis()
        self.create_visualizations()
        self.create_dashboard_visualizations()
        self.advanced_insights()
        self.generate_summary_report()

        print("\n✅ EDA Complete! All analyses and visualizations have been generated.")
```

```
# USAGE INSTRUCTIONS
print("""
🚀 USAGE INSTRUCTIONS:

1. Update the file path below to point to your Excel file
2. Run the analysis

Example usage:
""")

# Example usage - UPDATE THIS PATH TO YOUR EXCEL FILE
if __name__ == "__main__":
    # CHANGE THIS PATH TO YOUR EXCEL FILE LOCATION
    file_path = r"/content/drive/MyDrive/CAPSTONE/Data/Transactions_2023.xlsx"  # Update this path

    # Create EDA instance and run analysis
    eda = HospitalDataEDA(file_path)
    eda.run_complete_eda()

# ALTERNATIVE USAGE FOR GOOGLE COLAB/DRIVE:
"""
For Google Colab with Google Drive:

from google.colab import drive
drive.mount('/content/drive')

# Then use the mounted drive path:
file_path = '/content/drive/MyDrive/CAPSTONE/Data/Transactions_2023.xlsx'
eda = HospitalDataEDA(file_path)
eda.run_complete_eda()
"""

# ADDITIONAL ANALYSIS FUNCTIONS
def custom_analysis(file_path):
    """
    Run specific custom analysis based on your business needs
    """
    eda = HospitalDataEDA(file_path)
    eda.load_data()
    df = eda.combined_data

    print("\n" + "="*60)
    print("CUSTOM BUSINESS ANALYSIS")
    print("="*60)

    # 1. Peak Hours Analysis
    if 'transaction_time' in df.columns:
        df['hour'] = pd.to_datetime(df['transaction_time'], format='%H:%M', errors='coerce').dt.hour
        peak_hours = df['hour'].value_counts().sort_index()

        plt.figure(figsize=(12, 6))
        plt.subplot(1, 2, 1)
        plt.plot(peak_hours.index, peak_hours.values, marker='o', linewidth=2, markersize=6)
        plt.title('Transaction Volume by Hour', fontweight='bold')
        plt.xlabel('Hour of Day')
        plt.ylabel('Number of Transactions')
        plt.grid(True, alpha=0.3)

        # 2. Cost Efficiency Analysis
        if 'unit_cost' in df.columns and 'total_cost' in df.columns:
            plt.subplot(1, 2, 2)
            efficiency_ratio = df['total_cost'] / (df['unit_cost'] * df['quantity'])
            plt.hist(efficiency_ratio, bins=30, alpha=0.7, color='lightcoral', edgecolor='black')
            plt.title('Cost Efficiency Distribution', fontweight='bold')
            plt.xlabel('Efficiency Ratio')
            plt.ylabel('Frequency')
            plt.grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()

    # 3. Risk Assessment
    print(f"\nRisk Assessment Metrics:")
    print("-" * 25)

    if 'bounced' in df.columns:
        high_risk_patients = df[df['bounced'] == True]['patient_id'].nunique()
        print(f"High-risk patients (with bounced transactions): {high_risk_patients:,}")
```

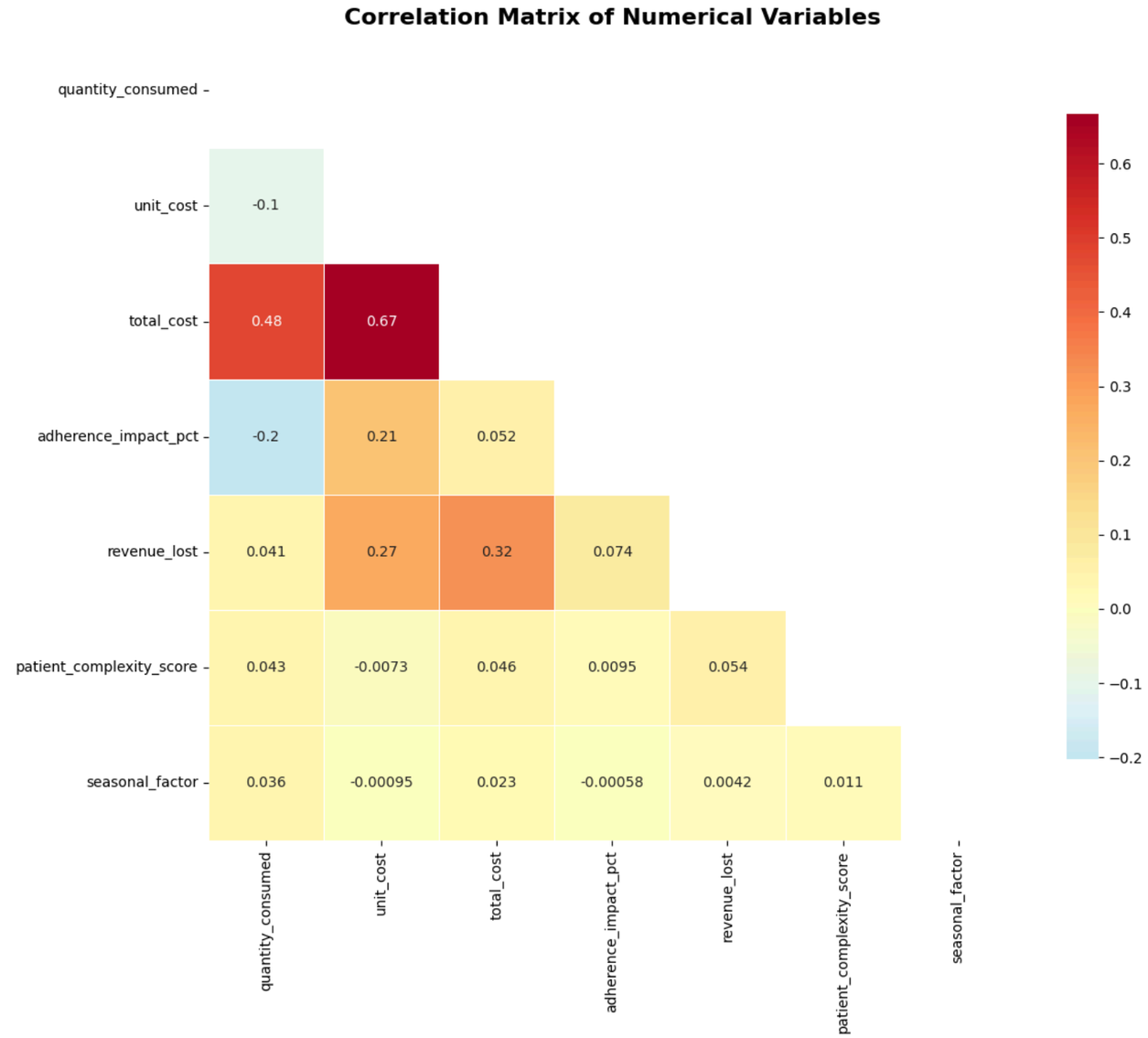
```
        if 'adherence_impact_pct' in df.columns:
            critical_adherence = df[df['adherence_impact_pct'] > 40]
            print(f"Critical adherence cases (>40% impact): {len(critical_adherence):,}")

        if 'urgency_level' in df.columns:
            emergency_cases = df[df['urgency_level'] == 'Emergency']
            print(f"Emergency cases requiring immediate attention: {len(emergency_cases):,}")

    # PERFORMANCE OPTIMIZATION FOR LARGE DATASETS
    def optimized_eda_for_large_data(file_path, sample_size=10000):
        """
        Optimized EDA for very large datasets using sampling
        """
        print(f"\n🔧 OPTIMIZED EDA FOR LARGE DATASETS (Sample Size: {sample_size:,})")
        print("="*70)

        # Read file in chunks for memory efficiency
        chunk_list = []
        chunk_size = 5000

        try:
            excel_file = pd.ExcelFile(file_path)
            for sheet_name in excel_file.sheet_names:
                sheet_data = pd.read_excel(file_path, sheet_name=sheet_name)
                chunk_list.append(sheet_data)

            # Combine all sheets
            full_data = pd.concat(chunk_list, ignore_index=True)

            # Sample data if too large
            if len(full_data) > sample_size:
                sampled_data = full_data.sample(n=sample_size, random_state=42)
                print(f"📊 Sampled {sample_size:,} records from {len(full_data):,} total records")
            else:
                sampled_data = full_data
                print(f"📊 Using full dataset: {len(full_data):,} records")

            # Run EDA on sampled data
            eda = HospitalDataEDA("")
            eda.combined_data = sampled_data
            eda.basic_info()
            eda.descriptive_statistics()
            eda.financial_analysis()

        except Exception as e:
            print(f"Error in optimized EDA: {e}")

    # EXPORT FUNCTIONS
    def export_insights_to_excel(eda_instance, output_path="hospital_eda_insights.xlsx"):
        """
        Export key insights to Excel file for sharing with stakeholders
        """
        df = eda_instance.combined_data

        with pd.ExcelWriter(output_path, engine='openpyxl') as writer:
            # Summary statistics
            numerical_cols = df.select_dtypes(include=[np.number]).columns.tolist()
            if numerical_cols:
                df[numerical_cols].describe().to_excel(writer, sheet_name='Summary_Stats')

            # Department performance
            if 'dept_id' in df.columns and 'total_cost' in df.columns:
                dept_performance = df.groupby('dept_id').agg({
                    'total_cost': ['sum', 'mean', 'count'],
                    'formulary_adherent': 'mean' if 'formulary_adherent' in df.columns else 'count'
                })
                dept_performance.to_excel(writer, sheet_name='Department_Performance')

            # Patient analysis
            if 'patient_id' in df.columns:
                patient_summary = df.groupby('patient_id').agg({
                    'total_cost': 'sum',
                    'transaction_id': 'count'
                }).sort_values('total_cost', ascending=False)
                patient_summary.to_excel(writer, sheet_name='Patient_Analysis')

            # Quality metrics
            quality_metrics = {
```

```
                'Total_Records': len(df),
                'Missing_Data_Percentage': (df.isnull().sum().sum() / (len(df) * len(df.columns))) * 100,
                'Duplicate_Records': df.duplicated().sum(),
                'Adherence_Rate': (df['formulary_adherent'] == True).mean() * 100 if 'formulary_adherent' in df.columns else None,
                'Bounce_Rate': (df['bounced'] == True).mean() * 100 if 'bounced' in df.columns else None
            }

            quality_df = pd.DataFrame([quality_metrics])
            quality_df.to_excel(writer, sheet_name='Quality_Metrics', index=False)

        print(f"✅ Insights exported to: {output_path}")

print("""
📋 ADDITIONAL FEATURES:

1. Custom Analysis:
   custom_analysis('/content/drive/MyDrive/CAPSTONE/Data/Transactions_2023.xlsx')

2. Large Dataset Optimization:
   optimized_eda_for_large_data('/content/drive/MyDrive/CAPSTONE/Data/Transactions_2023.xlsx', sample_size=5000)

3. Export Insights:
   eda = HospitalDataEDA('/content/drive/MyDrive/CAPSTONE/Data/Transactions_2023.xlsx')
   eda.run_complete_eda()
   export_insights_to_excel(eda, 'insights_report.xlsx')

📈 WHAT THIS EDA PROVIDES:

✓ Data Quality Assessment
✓ Missing Value Analysis
✓ Transaction Pattern Analysis
✓ Financial Performance Metrics
✓ Patient Behavior Insights
✓ Physician Performance Analysis
✓ Adherence Rate Analysis
✓ Seasonal Trend Analysis
✓ Correlation Analysis
✓ Risk Assessment
✓ Interactive Visualizations
✓ Executive Summary Report

🎯 KEY BUSINESS INSIGHTS:
- Revenue optimization opportunities
- Patient adherence patterns
- Department performance comparison
- Peak operational hours
- Risk factor identification
- Cost efficiency analysis

💡 NEXT STEPS:
1. Update the file path to your Excel file
2. Run: python hospital_eda.py
3. Review generated visualizations and insights
4. Export results for stakeholder presentation
""")
```

```
🚀 USAGE INSTRUCTIONS:

1. Update the file path below to point to your Excel file
2. Run the analysis

Example usage:

🏥 HOSPITAL TRANSACTION DATA - EXPLORATORY DATA ANALYSIS
================================================================================
Loading data from Excel file...
Available sheets: ['Sheet1']
Loaded sheet 'Sheet1' with shape: (536826, 20)
============================================================
BASIC DATASET INFORMATION
============================================================
Dataset Shape: (536826, 20)
Number of Rows: 536,826
Number of Columns: 20

Column Names and Data Types:
----------------------------------------
transaction_id       | object     | Nulls:      0 (  0.0%)
hospital_id          | object     | Nulls:      0 (  0.0%)
dept_id              | object     | Nulls:      0 (  0.0%)
physician_id         | object     | Nulls:      0 (  0.0%)
patient_id           | object     | Nulls:      0 (  0.0%)
sku_id               | object     | Nulls:      0 (  0.0%)
transaction_date     | datetime64[ns] | Nulls:      0 (  0.0%)
transaction_time     | object     | Nulls:      0 (  0.0%)
quantity_consumed    | float64    | Nulls:      0 (  0.0%)
unit_cost            | float64    | Nulls:      0 (  0.0%)
total_cost           | float64    | Nulls:      0 (  0.0%)
transaction_type     | object     | Nulls:      0 (  0.0%)
formulary_adherent   | bool       | Nulls:      0 (  0.0%)
adherence_impact_pct | float64    | Nulls:      0 (  0.0%)
bounced              | bool       | Nulls:      0 (  0.0%)
bounce_reason        | object     | Nulls: 479388 ( 89.3%)
revenue_lost         | float64    | Nulls:      0 (  0.0%)
patient_complexity_score | float64    | Nulls:      0 (  0.0%)
seasonal_factor      | float64    | Nulls:      0 (  0.0%)
urgency_level        | object     | Nulls:      0 (  0.0%)
Memory Usage: 308.65 MB

============================================================
DATA QUALITY ANALYSIS
============================================================

Missing Values Analysis:
------------------------------
               Missing_Count  Missing_Percentage
bounce_reason         479388           89.300444

Duplicate Records: 0 (0.00%)

Data Type Issues:
--------------------
✓ transaction_date: Valid date format
✓ transaction_time: Valid date format
✓ unit_cost: No negative values
✓ total_cost: No negative values
✓ revenue_lost: No negative values

============================================================
DESCRIPTIVE STATISTICS
============================================================

Numerical Variables Summary:
----------------------------------------
       quantity_consumed  unit_cost  total_cost  adherence_impact_pct  \
count          536826.00  536826.00   536826.00             536826.00
mean                4.29      64.76      255.46                 11.12
std                 3.40      65.45      312.59                 14.94
min                 0.27       2.10        1.93                  0.00
25%                 1.98      23.57       60.25                  0.00
50%                 3.28      42.14      139.11                  0.00
75%                 5.52      74.30      328.05                 23.40
max                25.00     425.00     4102.07                 45.00

       revenue_lost  patient_complexity_score  seasonal_factor
count     536826.00                536826.00        536826.00
mean          31.14                     1.77             1.01
std          147.26                     0.54             0.09
min            0.00                     0.80             0.90
25%            0.00                     1.40             0.95
50%            0.00                     1.79             1.05
```

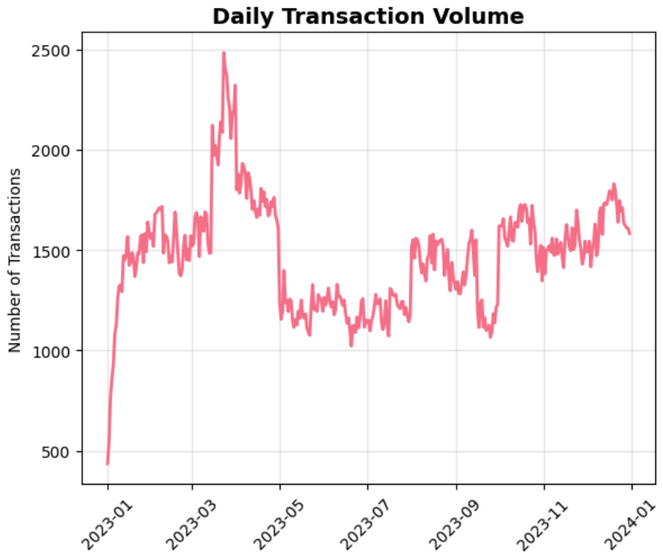
```
50%            0.00                      2.10            1.15
75%            0.00                      2.10            1.15
max         4003.16                      3.36            1.15

Categorical Variables Summary:
---------------------------------------
transaction_id           | Unique: 536826 | Most Common: T00000018
hospital_id              | Unique:      1 | Most Common: H001
dept_id                  | Unique:     10 | Most Common: D003
physician_id             | Unique:    120 | Most Common: P0028
patient_id               | Unique:  24880 | Most Common: PT043004
sku_id                   | Unique:   1197 | Most Common: SKU00373
transaction_time         | Unique:   1440 | Most Common: 20:22
transaction_type         | Unique:      2 | Most Common: Prescription
bounce_reason            | Unique:      4 | Most Common: High Cost
urgency_level            | Unique:      3 | Most Common: Normal

============================================================
TRANSACTION PATTERN ANALYSIS
============================================================

Transaction Volume by Month:
month
1     40913
2     43374
3     59093
4     52820
5     37287
6     35880
7     37223
8     45436
9     38553
10    49576
11    45606
12    51065
Name: count, dtype: int64

Transaction Volume by Day of Week:
day_of_week
Friday       77927
Sunday       77610
Wednesday    77012
Thursday     76714
Saturday     76380
Tuesday      76078
Monday       75105
Name: count, dtype: int64

Transaction Types:
transaction_type
Prescription    417348
Procedure       119478
Name: count, dtype: int64

Transaction Type Percentages:
transaction_type
Prescription    77.74
Procedure       22.26
Name: count, dtype: float64

Top 10 Departments by Transaction Volume:
dept_id
D003    207284
D002    135534
D001     81979
D007     16647
D004     16356
D010     16297
D009     16088
D008     16084
D005     15285
D006     15272
Name: count, dtype: int64

============================================================
FINANCIAL ANALYSIS
============================================================
Total Revenue: $137,135,167.09
Average Transaction Value: $255.46
Median Transaction Value: $139.11

Revenue by Transaction Type:
                              sum     mean   count
transaction_type
Prescription      $100,148,518.50  $239.96  417348
Procedure          $36,986,648.59  $309.57  119478

Total Revenue Lost: $16,714,573.38
```

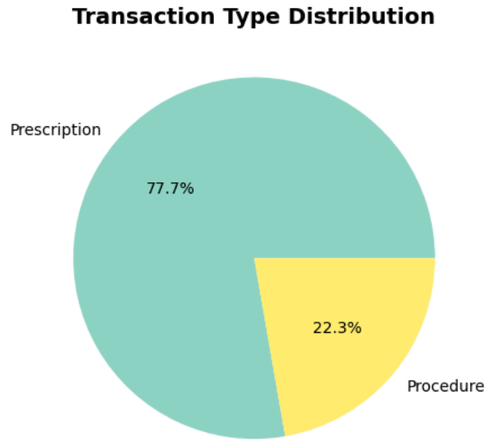
```
Revenue Lost by Bounce Reason:
High Cost: $6,582,804.23
Insurance Issues: $4,268,414.22
Patient Declined: $3,403,486.80
Alternative Prescribed: $2,459,868.13

============================================================
ADHERENCE ANALYSIS
============================================================
Overall Formulary Adherence Rate: 59.57%

Adherence Rate by Urgency Level:
Emergency: 44.16%
Normal: 61.19%
Urgent: 59.46%

Average Adherence Impact: 11.12%
High Impact Transactions (>30%): 92,575 (17.24%)

============================================================
CORRELATION ANALYSIS
============================================================
```

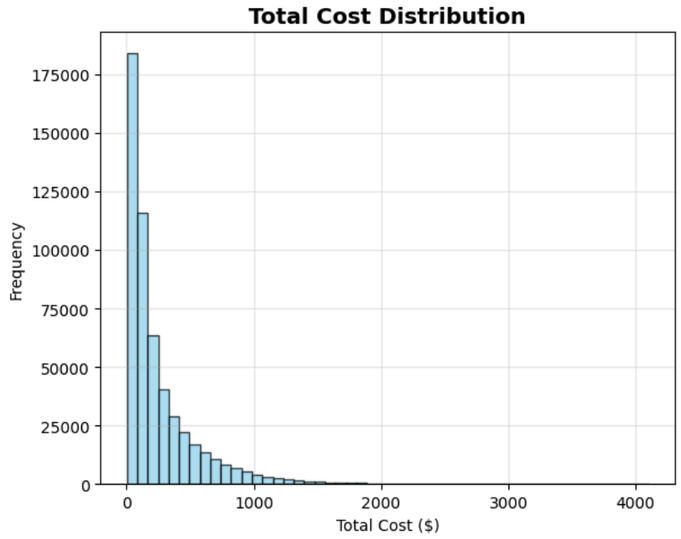


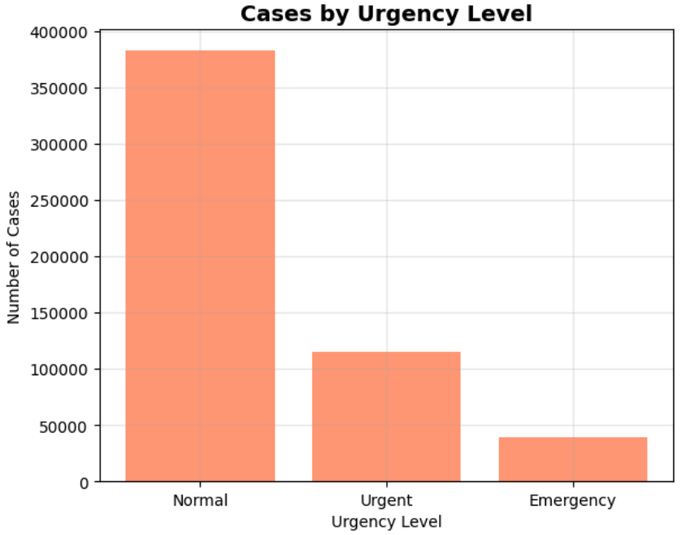

```
Strong Correlations (|r| > 0.7):
----------------------------------
No strong correlations found
```

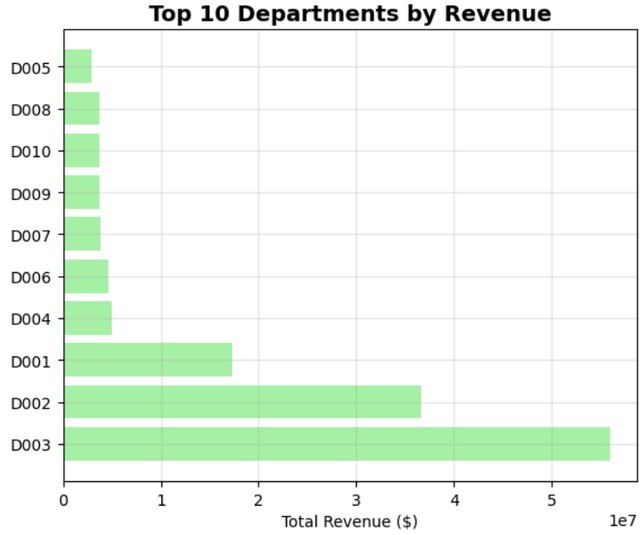
```
============================================================
GENERATING VISUALIZATIONS
============================================================
```

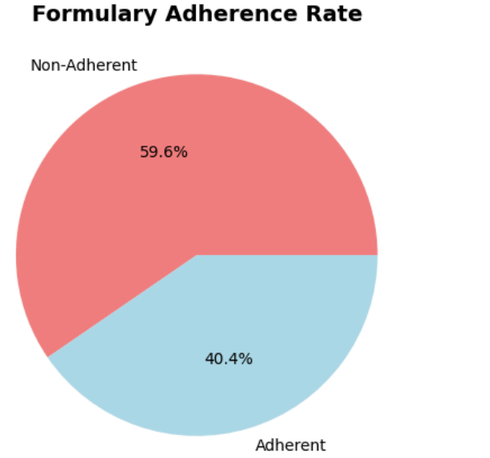


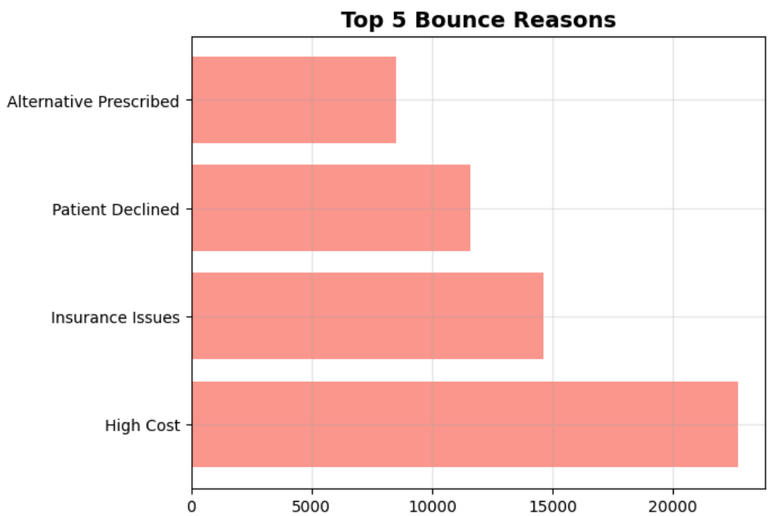


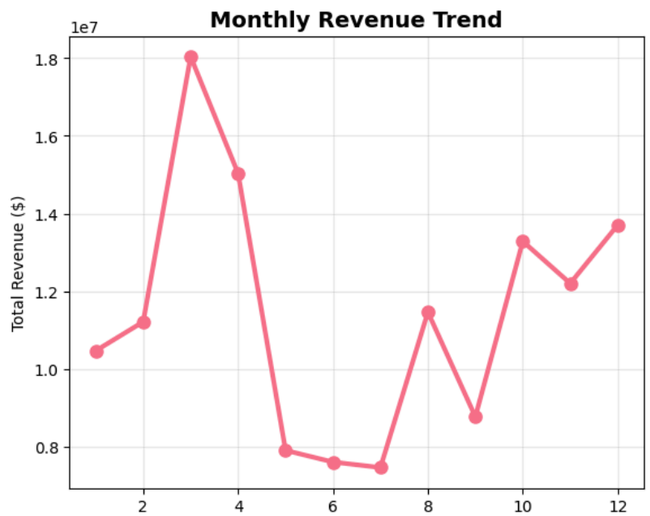






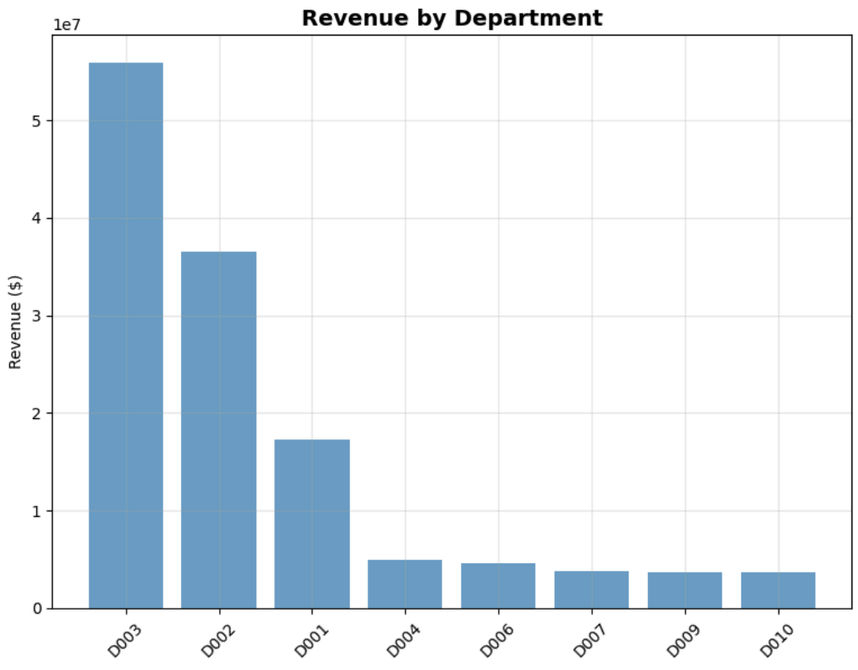


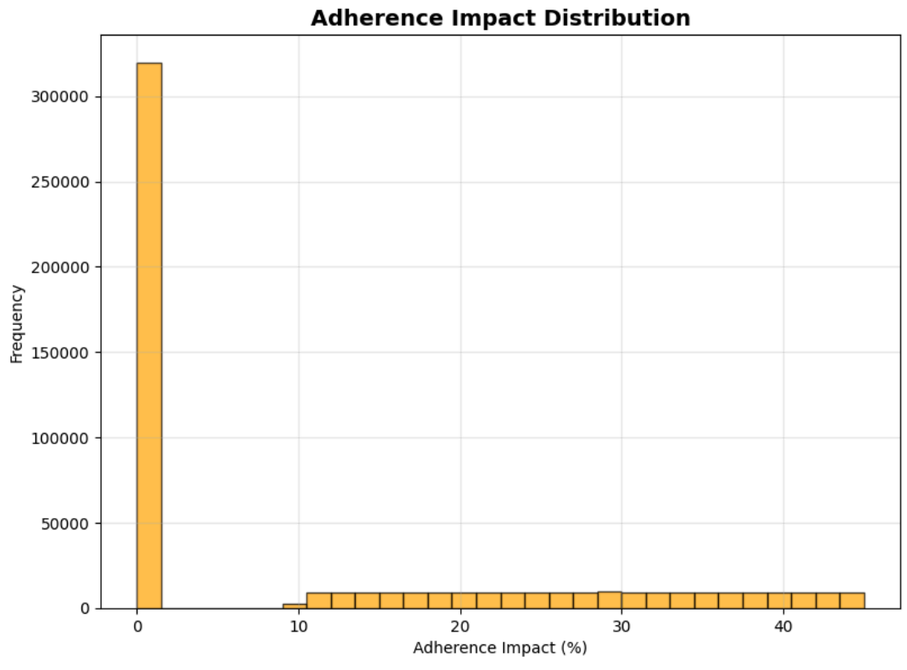


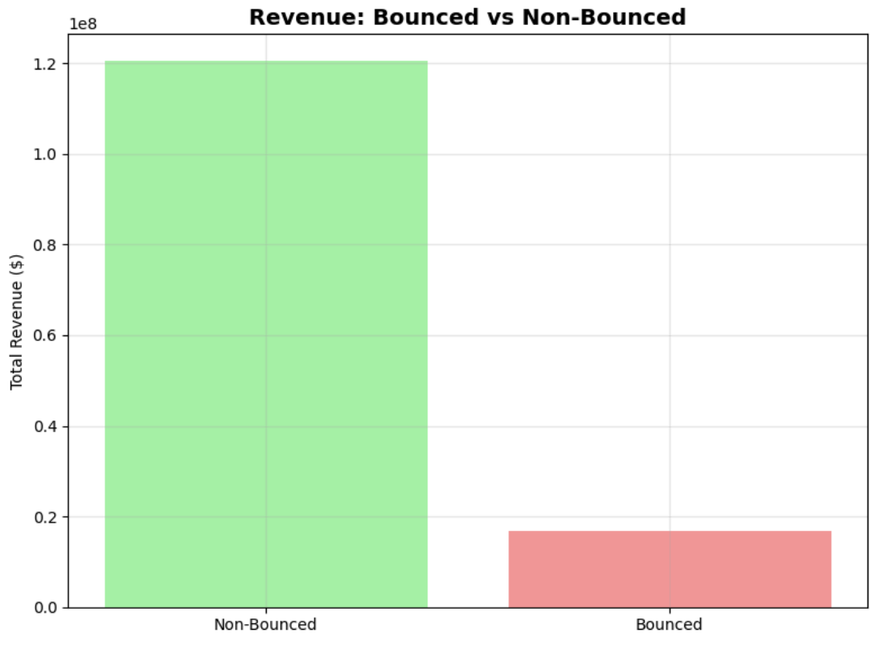


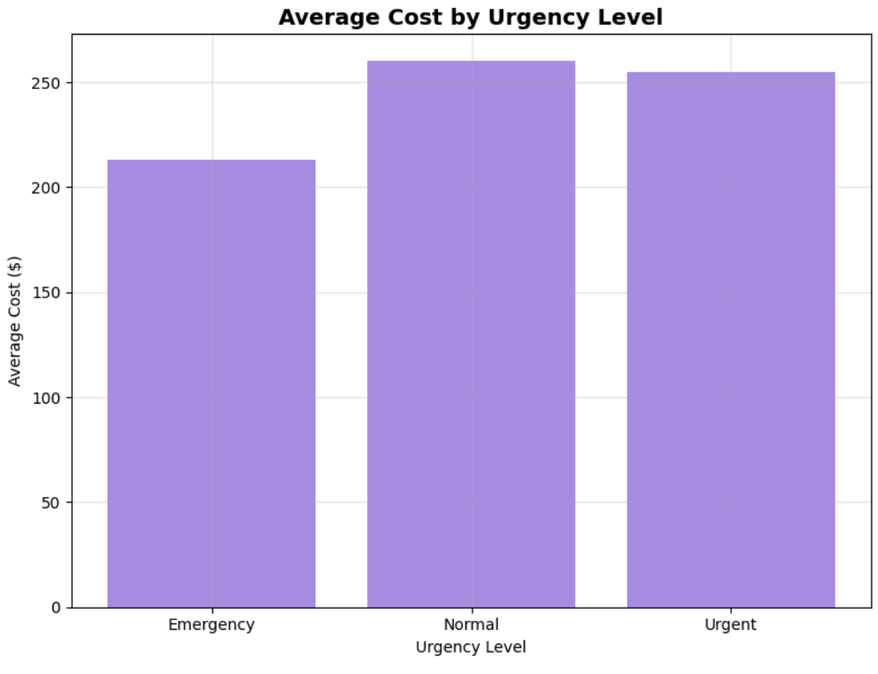
Frequency Month

```
============================================================
CREATING DASHBOARD VISUALIZATIONS
============================================================
```










```
============================================================
ADVANCED BUSINESS INSIGHTS
============================================================

Patient Behavior Analysis:
------------------------------
Average transactions per patient: 21.58
Patients with single transaction: 146
Most active patient transactions: 188

Top 5 Patients by Total Spending:
1. Patient PT044193: $83,122.79
```

```
2. Patient PT036812: $75,622.02
3. Patient PT013522: $73,071.26
4. Patient PT049762: $71,189.86
5. Patient PT043991: $68,321.71

Physician Performance Analysis:
---------------------------------
Top 5 Physicians by Total Revenue:
Physician P0023: $4,438,671.52
Physician P0019: $4,408,837.01
Physician P0022: $4,353,993.96
Physician P0027: $4,335,965.05
Physician P0021: $4,335,457.68

================================================================================
EXECUTIVE SUMMARY REPORT
================================================================================

📊 DATASET OVERVIEW
   • Total Records: 536,826
   • Time Period: 2023-01-01 to 2023-12-31
   • Unique Patients: 24,880
   • Unique Physicians: 120
   • Departments: 10

💰 FINANCIAL METRICS
   • Total Revenue: $137,135,167.09
   • Average Transaction: $255.46
   • Revenue Range: $1.93 - $4,102.07
   • Total Revenue Lost: $16,714,573.38
   • Revenue Loss Rate: 12.19%

🩺 CLINICAL METRICS
   • Formulary Adherence Rate: 59.57%
   • Emergency Cases: 7.33%

⚠️  OPERATIONAL METRICS
   • Transaction Bounce Rate: 10.70%
   • Top Bounce Reason: High Cost

🔍 KEY INSIGHTS
   1. 10% of transactions ($642.28+) represent high-value cases
   2. Cost difference between adherent and non-adherent cases: $23.18
   3. Busiest department (D003) handles 207,284 transactions

================================================================================

✅ EDA Complete! All analyses and visualizations have been generated.

📋 ADDITIONAL FEATURES:

1. Custom Analysis:
   custom_analysis('/content/drive/MyDrive/CAPSTONE/Data/Transactions_2023.xlsx')

2. Large Dataset Optimization:
   optimized_eda_for_large_data('/content/drive/MyDrive/CAPSTONE/Data/Transactions_2023.xlsx', sample_size=5000)

3. Export Insights:
   eda = HospitalDataEDA('/content/drive/MyDrive/CAPSTONE/Data/Transactions_2023.xlsx')
   eda.run_complete_eda()
   export_insights_to_excel(eda, 'insights_report.xlsx')

📈 WHAT THIS EDA PROVIDES:

✓ Data Quality Assessment
✓ Missing Value Analysis
✓ Transaction Pattern Analysis
✓ Financial Performance Metrics
✓ Patient Behavior Insights
✓ Physician Performance Analysis
✓ Adherence Rate Analysis
✓ Seasonal Trend Analysis
✓ Correlation Analysis
✓ Risk Assessment
✓ Interactive Visualizations
✓ Executive Summary Report

🎯 KEY BUSINESS INSIGHTS:
- Revenue optimization opportunities
- Patient adherence patterns
- Department performance comparison
- Peak operational hours
- Risk factor identification
- Cost efficiency analysis

💡 NEXT STEPS:
1. Update the file path to your Excel file
```

```
2. Run: python hospital_eda.py
3. Review generated visualizations and insights
4. Export results for stakeholder presentation
```

Transaction 2024

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# Set up plotting style
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 8)

class HospitalDataEDA:
    def __init__(self, file_path):
        """
        Initialize the EDA class with Excel file path
        """
        self.file_path = file_path
        self.data = {}
        self.sheet_names = []

    def load_data(self):
        """
        Load all sheets from Excel file
        """
        try:
            # Get all sheet names
            excel_file = pd.ExcelFile(self.file_path)
            self.sheet_names = excel_file.sheet_names
            print(f"Found {len(self.sheet_names)} sheets: {self.sheet_names}")

            # Load all sheets
            for sheet in self.sheet_names:
                self.data[sheet] = pd.read_excel(self.file_path, sheet_name=sheet)
                print(f"Loaded sheet '{sheet}' with shape: {self.data[sheet].shape}")

        except Exception as e:
            print(f"Error loading data: {e}")

    def basic_info(self, sheet_name=None):
        """
        Display basic information about the dataset
        """
        if sheet_name is None:
            sheets_to_analyze = self.sheet_names
        else:
            sheets_to_analyze = [sheet_name]

        for sheet in sheets_to_analyze:
            df = self.data[sheet]
            print(f"\n{'='*50}")
            print(f"BASIC INFO FOR SHEET: {sheet}")
            print(f"{'='*50}")

            print(f"Dataset Shape: {df.shape}")
            print(f"Memory Usage: {df.memory_usage(deep=True).sum() / 1024**2:.2f} MB")

            print(f"\nColumn Names and Types:")
            print(df.dtypes)

            print(f"\nMissing Values:")
            missing = df.isnull().sum()
            missing_pct = (missing / len(df)) * 100
            missing_df = pd.DataFrame({
                'Missing Count': missing,
                'Missing Percentage': missing_pct
            })
            print(missing_df[missing_df['Missing Count'] > 0])

            print(f"\nDuplicate Rows: {df.duplicated().sum()}")

    def data_preprocessing(self, sheet_name):
        """
```

```
        Clean and preprocess the data
        """
        df = self.data[sheet_name].copy()

        print(f"\nPREPROCESSING SHEET: {sheet_name}")
        print("="*40)

        # Convert transaction_date to datetime if it exists
        if 'transaction_date' in df.columns:
            df['transaction_date'] = pd.to_datetime(df['transaction_date'])
            df['year'] = df['transaction_date'].dt.year
            df['month'] = df['transaction_date'].dt.month
            df['day_of_week'] = df['transaction_date'].dt.day_name()
            df['hour'] = df['transaction_date'].dt.hour

        # Convert boolean columns and handle potential missing values
        bool_columns = ['formulary_adherent', 'bounced']
        for col in bool_columns:
            if col in df.columns:
                # Convert to string, then map, then fill missing with False
                df[col] = df[col].astype(str).map({'TRUE': True, 'FALSE': False}).fillna(False)


        # Convert numeric columns
        numeric_columns = ['quantity_consumed', 'unit_cost', 'total_cost', 'adherence_impact_pct', 'revenue_lost', 'seasonal_factor', 'urgency_level']
        for col in numeric_columns:
            if col in df.columns:
                df[col] = pd.to_numeric(df[col], errors='coerce')

        self.data[f"{sheet_name}_processed"] = df
        print(f"Preprocessing completed. New columns added: year, month, day_of_week, hour")
        return df

    def univariate_analysis(self, sheet_name):
        """
        Perform univariate analysis
        """
        df = self.data[f"{sheet_name}_processed"]

        print(f"\nUNIVARIATE ANALYSIS FOR: {sheet_name}")
        print("="*50)

        # Numerical columns analysis
        numerical_cols = df.select_dtypes(include=[np.number]).columns

        if len(numerical_cols) > 0:
            print("\nNUMERICAL VARIABLES SUMMARY:")
            print(df[numerical_cols].describe())

            # Plot histograms for key numerical variables
            fig, axes = plt.subplots(2, 3, figsize=(18, 12))
            axes = axes.ravel()

            key_numeric_cols = ['total_cost', 'unit_cost', 'quantity_consumed', 'adherence_impact_pct', 'revenue_lost', 'urgency_level']

            for i, col in enumerate(key_numeric_cols):
                if col in df.columns and i < 6:
                    df[col].hist(bins=30, ax=axes[i], alpha=0.7, edgecolor='black')
                    axes[i].set_title(f'Distribution of {col}')
                    axes[i].set_xlabel(col)
                    axes[i].set_ylabel('Frequency')

            plt.tight_layout()
            plt.show()

        # Categorical columns analysis
        categorical_cols = df.select_dtypes(include=['object', 'bool']).columns

        if len(categorical_cols) > 0:
            print(f"\nCATEGORICAL VARIABLES:")

            categorical_to_plot = ['transaction_type', 'formulary_adherent', 'bounced', 'bounce_reason', 'day_of_week']

            fig, axes = plt.subplots(2, 3, figsize=(18, 12))
            axes = axes.ravel()

            for i, col in enumerate(categorical_to_plot):
                if col in df.columns and i < 6:
                    value_counts = df[col].value_counts().head(10)
```

```
                    # Check if value_counts is not empty before plotting
                    if not value_counts.empty:
                        value_counts.plot(kind='bar', ax=axes[i], alpha=0.7)
                        axes[i].set_title(f'Distribution of {col}')
                        axes[i].set_xlabel(col)
                        axes[i].set_ylabel('Count')
                        axes[i].tick_params(axis='x', rotation=45)
                    else:
                        axes[i].set_title(f'No data to plot for {col}')
                        axes[i].text(0.5, 0.5, 'No data', horizontalalignment='center', verticalalignment='center', transform=axes[i].transAxes)


            plt.tight_layout()
            plt.show()

    def temporal_analysis(self, sheet_name):
        """
        Analyze temporal patterns in the data
        """
        df = self.data[f"{sheet_name}_processed"]

        if 'transaction_date' not in df.columns:
            print("No transaction_date column found for temporal analysis")
            return

        print(f"\nTEMPORAL ANALYSIS FOR: {sheet_name}")
        print("="*40)

        # Daily transaction volume
        daily_transactions = df.groupby(df['transaction_date'].dt.date).size()

        # Monthly patterns
        monthly_stats = df.groupby(['year', 'month']).agg({
            'total_cost': ['sum', 'mean', 'count'],
            'transaction_id': 'count'
        }).round(2)

        print("Monthly Transaction Summary:")
        print(monthly_stats)

        # Time series plots
        fig, axes = plt.subplots(2, 2, figsize=(16, 12))

        # Daily transaction volume
        daily_transactions.plot(ax=axes[0,0], title='Daily Transaction Volume')
        axes[0,0].set_xlabel('Date')
        axes[0,0].set_ylabel('Number of Transactions')

        # Monthly total cost
        monthly_cost = df.groupby(['year', 'month'])['total_cost'].sum()
        monthly_cost.plot(kind='bar', ax=axes[0,1], title='Monthly Total Cost')
        axes[0,1].set_xlabel('Year-Month')
        axes[0,1].set_ylabel('Total Cost')

        # Day of week pattern
        day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
        dow_counts = df['day_of_week'].value_counts().reindex(day_order)
        dow_counts.plot(kind='bar', ax=axes[1,0], title='Transactions by Day of Week')
        axes[1,0].set_xlabel('Day of Week')
        axes[1,0].set_ylabel('Count')

        # Hourly pattern
        if 'hour' in df.columns:
            hourly_counts = df['hour'].value_counts().sort_index()
            hourly_counts.plot(kind='line', ax=axes[1,1], title='Transactions by Hour of Day', marker='o')
            axes[1,1].set_xlabel('Hour')
            axes[1,1].set_ylabel('Count')

        plt.tight_layout()
        plt.show()

    def financial_analysis(self, sheet_name):
        """
        Analyze financial aspects of the data
        """
        df = self.data[f"{sheet_name}_processed"]

        print(f"\nFINANCIAL ANALYSIS FOR: {sheet_name}")
        print("="*40)
```

```
# Overall financial metrics
total_revenue = df['total_cost'].sum()
total_revenue_lost = df['revenue_lost'].sum() if 'revenue_lost' in df.columns else 0
avg_transaction_value = df['total_cost'].mean()

print(f"Total Revenue: ${total_revenue:,.2f}")
print(f"Total Revenue Lost: ${total_revenue_lost:,.2f}")
print(f"Average Transaction Value: ${avg_transaction_value:,.2f}")
print(f"Revenue Loss Rate: {(total_revenue_lost/total_revenue)*100:.2f}%")

# Transaction type analysis
if 'transaction_type' in df.columns:
    type_analysis = df.groupby('transaction_type').agg({
        'total_cost': ['sum', 'mean', 'count'],
        'revenue_lost': 'sum' if 'revenue_lost' in df.columns else 'count'
    }).round(2)

    print(f"\nTransaction Type Analysis:")
    print(type_analysis)

# Cost distribution analysis
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Total cost distribution
df['total_cost'].hist(bins=50, ax=axes[0,0], alpha=0.7, edgecolor='black')
axes[0,0].set_title('Total Cost Distribution')
axes[0,0].set_xlabel('Total Cost')
axes[0,0].set_ylabel('Frequency')

# Cost by transaction type
if 'transaction_type' in df.columns:
    df.boxplot(column='total_cost', by='transaction_type', ax=axes[0,1])
    axes[0,1].set_title('Cost Distribution by Transaction Type')
    axes[0,1].set_xlabel('Transaction Type')
    axes[0,1].set_ylabel('Total Cost')

# Revenue lost analysis
if 'revenue_lost' in df.columns:
    df['revenue_lost'].hist(bins=30, ax=axes[1,0], alpha=0.7, edgecolor='black')
    axes[1,0].set_title('Revenue Lost Distribution')
    axes[1,0].set_xlabel('Revenue Lost')
    axes[1,0].set_ylabel('Frequency')

# Unit cost vs quantity
if 'unit_cost' in df.columns and 'quantity_consumed' in df.columns:
    df.plot.scatter(x='quantity_consumed', y='unit_cost', ax=axes[1,1], alpha=0.6)
    axes[1,1].set_title('Unit Cost vs Quantity Consumed')
    axes[1,1].set_xlabel('Quantity Consumed')
    axes[1,1].set_ylabel('Unit Cost')

plt.tight_layout()
plt.show()

def patient_analysis(self, sheet_name):
    """
    Analyze patient-related patterns
    """
    df = self.data[f"{sheet_name}_processed"]

    print(f"\nPATIENT ANALYSIS FOR: {sheet_name}")
    print("="*40)

    # Patient transaction patterns
    patient_stats = df.groupby('patient_id').agg({
        'total_cost': ['sum', 'mean', 'count'],
        'transaction_date': ['min', 'max'] if 'transaction_date' in df.columns else 'count'
    }).round(2)

    print("Patient Transaction Statistics (Top 10 by total cost):")
    top_patients = patient_stats.sort_values(('total_cost', 'sum'), ascending=False).head(10)
    print(top_patients)

    # Adherence analysis
    if 'formulary_adherent' in df.columns:
        adherence_stats = df.groupby('formulary_adherent').agg({
            'total_cost': ['sum', 'mean', 'count'],
            'adherence_impact_pct': 'mean' if 'adherence_impact_pct' in df.columns else 'count'
        }).round(2)
```

```
        print(f"\nAdherence Analysis:")
        print(adherence_stats)

    # Plotting patient analysis
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))

    # Patient cost distribution
    patient_total_costs = df.groupby('patient_id')['total_cost'].sum()
    patient_total_costs.hist(bins=30, ax=axes[0,0], alpha=0.7, edgecolor='black')
    axes[0,0].set_title('Patient Total Cost Distribution')
    axes[0,0].set_xlabel('Total Cost per Patient')
    axes[0,0].set_ylabel('Number of Patients')

    # Transaction frequency per patient
    patient_transaction_count = df.groupby('patient_id').size()
    patient_transaction_count.hist(bins=20, ax=axes[0,1], alpha=0.7, edgecolor='black')
    axes[0,1].set_title('Transaction Frequency per Patient')
    axes[0,1].set_xlabel('Number of Transactions')
    axes[0,1].set_ylabel('Number of Patients')

    # Adherence impact
    if 'formulary_adherent' in df.columns:
        adherence_counts = df['formulary_adherent'].value_counts()
        # Check if value_counts is not empty before plotting
        if not adherence_counts.empty:
            adherence_counts.plot(kind='pie', ax=axes[1,0], autopct='%1.1f%%')
            axes[1,0].set_title('Formulary Adherence Distribution')
        else:
            axes[1,0].set_title('No data to plot for Formulary Adherence')
            axes[1,0].text(0.5, 0.5, 'No data', horizontalalignment='center', verticalalignment='center', transform=axes[1,0].transAxes)


    # Bounce analysis
    if 'bounced' in df.columns:
        bounce_counts = df['bounced'].value_counts()
        # Check if value_counts is not empty before plotting
        if not bounce_counts.empty:
            bounce_counts.plot(kind='pie', ax=axes[1,1], autopct='%1.1f%%')
            axes[1,1].set_title('Transaction Bounce Rate')
        else:
            axes[1,1].set_title('No data to plot for Bounce Rate')
            axes[1,1].text(0.5, 0.5, 'No data', horizontalalignment='center', verticalalignment='center', transform=axes[1,1].transAxes)


    plt.tight_layout()
    plt.show()

def correlation_analysis(self, sheet_name):
    """
    Analyze correlations between numerical variables
    """
    df = self.data[f"{sheet_name}_processed"]

    print(f"\nCORRELATION ANALYSIS FOR: {sheet_name}")
    print("="*50)

    # Select numerical columns for correlation
    numerical_cols = df.select_dtypes(include=[np.number]).columns

    if len(numerical_cols) > 1:
        correlation_matrix = df[numerical_cols].corr()

        # Plot correlation heatmap
        plt.figure(figsize=(12, 10))
        mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
        sns.heatmap(correlation_matrix,
                    mask=mask,
                    annot=True,
                    cmap='coolwarm',
                    center=0,
                    square=True,
                    fmt='.2f')
        plt.title('Correlation Matrix of Numerical Variables')
        plt.tight_layout()
        plt.show()

        # Print strong correlations
        print("\nStrong Correlations (|r| > 0.5):")
```

```
                for i in range(len(correlation_matrix.columns)):
                    for j in range(i+1, len(correlation_matrix.columns)):
                        corr_val = correlation_matrix.iloc[i, j]
                        if abs(corr_val) > 0.5:
                            print(f"{correlation_matrix.columns[i]} - {correlation_matrix.columns[j]}: {corr_val:.3f}")

    def advanced_insights(self, sheet_name):
        """
        Generate advanced insights and business intelligence
        """
        df = self.data[f"{sheet_name}_processed"]

        print(f"\nADVANCED INSIGHTS FOR: {sheet_name}")
        print("="*50)

        # Revenue impact analysis
        if 'bounce_reason' in df.columns:
            bounce_impact = df.groupby('bounce_reason').agg({
                'total_cost': 'sum',
                'revenue_lost': 'sum' if 'revenue_lost' in df.columns else 'count',
                'transaction_id': 'count' # Use transaction_id for count
            }).round(2)

            print("Revenue Impact by Bounce Reason:")
            print(bounce_impact.sort_values('total_cost', ascending=False))

        # Physician performance analysis
        if 'physician_id' in df.columns:
            physician_stats = df.groupby('physician_id').agg({
                'total_cost': ['sum', 'mean', 'count'],
                'formulary_adherent': lambda x: (x == True).sum() / len(x) * 100 if 'formulary_adherent' in df.columns else 0
            }).round(2)

            print(f"\nTop 10 Physicians by Transaction Volume:")
            top_physicians = physician_stats.sort_values(('total_cost', 'count'), ascending=False).head(10)
            print(top_physicians)

        # Department analysis
        if 'dept_id' in df.columns:
            dept_analysis = df.groupby('dept_id').agg({
                'total_cost': ['sum', 'mean', 'count'],
                # Check if urgency_level is numeric before taking mean
                'urgency_level': 'mean' if 'urgency_level' in df.columns and pd.api.types.is_numeric_dtype(df['urgency_level']) else 'count'
            }).round(2)

            print(f"\nDepartment Performance:")
            print(dept_analysis.sort_values(('total_cost', 'sum'), ascending=False))

        # Seasonal analysis
        if 'seasonal_factor' in df.columns and 'month' in df.columns:
            seasonal_analysis = df.groupby('month').agg({
                'total_cost': 'sum',
                'seasonal_factor': 'mean',
                'transaction_id': 'count' # Use transaction_id for count
            }).round(2)

            print(f"\nSeasonal Patterns:")
            print(seasonal_analysis)

            # Plot seasonal trends
            fig, axes = plt.subplots(1, 2, figsize=(16, 6))

            seasonal_analysis['total_cost'].plot(kind='line', ax=axes[0], marker='o')
            axes[0].set_title('Monthly Revenue Trends')
            axes[0].set_xlabel('Month')
            axes[0].set_ylabel('Total Cost')

            seasonal_analysis['seasonal_factor'].plot(kind='line', ax=axes[1], marker='o', color='orange')
            axes[1].set_title('Seasonal Factor by Month')
            axes[1].set_xlabel('Month')
            axes[1].set_ylabel('Seasonal Factor')

            plt.tight_layout()
            plt.show()

    def outlier_analysis(self, sheet_name):
        """
        Identify and analyze outliers
        """
```

```
        df = self.data[f"{sheet_name}_processed"]

        print(f"\nOUTLIER ANALYSIS FOR: {sheet_name}")
        print("="*40)

        numerical_cols = ['total_cost', 'unit_cost', 'quantity_consumed']

        for col in numerical_cols:
            if col in df.columns:
                Q1 = df[col].quantile(0.25)
                Q3 = df[col].quantile(0.75)
                IQR = Q3 - Q1
                lower_bound = Q1 - 1.5 * IQR
                upper_bound = Q3 + 1.5 * IQR

                outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]

                print(f"\n{col} Outliers:")
                print(f"Number of outliers: {len(outliers)} ({len(outliers)/len(df)*100:.2f}%)")
                print(f"Outlier range: < {lower_bound:.2f} or > {upper_bound:.2f}")

                if len(outliers) > 0:
                    print("Top 5 outlier transactions:")
                    # Use transaction_id instead of transaction_id
                    print(outliers.nlargest(5, col)[['transaction_id', col, 'patient_id', 'transaction_type']])

        # Box plots for outlier visualization
        numerical_cols_available = [col for col in numerical_cols if col in df.columns]
        if numerical_cols_available:
            fig, axes = plt.subplots(1, len(numerical_cols_available), figsize=(6*len(numerical_cols_available), 6))
            if len(numerical_cols_available) == 1:
                axes = [axes]

            for i, col in enumerate(numerical_cols_available):
                df.boxplot(column=col, ax=axes[i])
                axes[i].set_title(f'{col} - Outlier Detection')
                axes[i].set_ylabel(col)

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

```
    def generate_summary_report(self, sheet_name):
        """
        Generate a comprehensive summary report
        """
        df = self.data[f"{sheet_name}_processed"]

        print(f"\n{'='*60}")
        print(f"EXECUTIVE SUMMARY REPORT FOR: {sheet_name}")
        print(f"{'='*60}")

        # Key metrics
        total_transactions = len(df)
        unique_patients = df['patient_id'].nunique()
        date_range = f"{df['transaction_date'].min()} to {df['transaction_date'].max()}" if 'transaction_date' in df.columns else "Date range not available"
        total_revenue = df['total_cost'].sum()

        print(f"📊 OVERVIEW:")
        print(f"   • Total Transactions: {total_transactions:,}")
        print(f"   • Unique Patients: {unique_patients:,}")
        print(f"   • Date Range: {date_range}")
        print(f"   • Total Revenue: ${total_revenue:,.2f}")

        # Transaction types breakdown
        if 'transaction_type' in df.columns:
            type_breakdown = df['transaction_type'].value_counts()
            print(f"\n💊 TRANSACTION TYPES:")
            for trans_type, count in type_breakdown.items():
                percentage = (count / total_transactions) * 100
                print(f"   • {trans_type}: {count:,} ({percentage:.1f}%)")

        # Quality metrics
        if 'formulary_adherent' in df.columns:
            adherence_rate = (df['formulary_adherent'] == True).sum() / len(df) * 100
            print(f"\n📈 QUALITY METRICS:")
            print(f"   • Formulary Adherence Rate: {adherence_rate:.1f}%")

        if 'bounced' in df.columns:
            bounce_rate = (df['bounced'] == True).sum() / len(df) * 100
```

```
            print(f"   • Transaction Bounce Rate: {bounce_rate:.1f}%")

        # Top insights
        print(f"\n🔍 KEY INSIGHTS:")

        # Highest cost transaction
        max_cost_row = df.loc[df['total_cost'].idxmax()]
        print(f"   • Highest cost transaction: ${max_cost_row['total_cost']:.2f} ({max_cost_row['transaction_type']})")

        # Most frequent patient
        most_frequent_patient = df['patient_id'].value_counts().head(1)
        print(f"   • Most frequent patient: {most_frequent_patient.index[0]} with {most_frequent_patient.iloc[0]} transactions")

        # Peak transaction day
        if 'day_of_week' in df.columns:
            peak_day = df['day_of_week'].value_counts().head(1)
            print(f"   • Peak transaction day: {peak_day.index[0]} ({peak_day.iloc[0]} transactions)")

    def run_complete_eda(self, sheet_name=None):
        """
        Run complete EDA pipeline
        """
        if sheet_name is None:
            # Run for the first sheet by default
            sheet_name = self.sheet_names[0]

        print(f"Starting Complete EDA for sheet: {sheet_name}")

        # Step 1: Basic Info
        self.basic_info(sheet_name)

        # Step 2: Data Preprocessing
        self.data_preprocessing(sheet_name)

        # Step 3: Univariate Analysis
        self.univariate_analysis(sheet_name)

        # Step 4: Temporal Analysis
        self.temporal_analysis(sheet_name)

        # Step 5: Financial Analysis
        self.financial_analysis(sheet_name)

        # Step 6: Patient Analysis
        self.patient_analysis(sheet_name)

        # Step 7: Correlation Analysis
        self.correlation_analysis(sheet_name)

        # Step 8: Outlier Analysis
        self.outlier_analysis(sheet_name)

        # Step 9: Summary Report
        self.generate_summary_report(sheet_name)

        print(f"\n✅ Complete EDA finished for {sheet_name}!")

# USAGE EXAMPLE
def main():
    """
    Main function to run the EDA
    """
    # Replace with your actual file path
    file_path = "/content/drive/MyDrive/CAPSTONE/Data/Transactions_2024.xlsx"  # Update this path

    # Initialize EDA object
    eda = HospitalDataEDA(file_path)

    # Load data
    eda.load_data()

    # Run complete EDA for each sheet
    for sheet in eda.sheet_names:
        print(f"\n🔄 Processing sheet: {sheet}")
        eda.run_complete_eda(sheet)
        print(f"\n" + "="*80 + "\n")

    # If you want to analyze a specific sheet only:
    # eda.run_complete_eda("Sheet1")  # Replace "Sheet1" with your actual sheet name
```

```
if __name__ == "__main__":
    main()

# ADDITIONAL ANALYSIS FUNCTIONS

def compare_sheets(eda_object):
    """
    Compare metrics across different sheets if multiple sheets exist
    """
    if len(eda_object.sheet_names) > 1:
        print("\n📊 CROSS-SHEET COMPARISON")
        print("="*40)

        comparison_data = []

        for sheet in eda_object.sheet_names:
            if f"{sheet}_processed" in eda_object.data:
                df = eda_object.data[f"{sheet}_processed"]

                metrics = {
                    'Sheet': sheet,
                    'Total_Transactions': len(df),
                    'Unique_Patients': df['patient_id'].nunique(),
                    'Total_Revenue': df['total_cost'].sum(),
                    'Avg_Transaction_Value': df['total_cost'].mean(),
                    'Date_Range_Days': (df['transaction_date'].max() - df['transaction_date'].min()).days if 'transaction_date' in df.columns else 0
                }

                comparison_data.append(metrics)

        comparison_df = pd.DataFrame(comparison_data)
        print(comparison_df)

        return comparison_df

def export_summary_to_excel(eda_object, output_path="hospital_eda_summary.xlsx"):
    """
    Export EDA summary to Excel file
    """
    with pd.ExcelWriter(output_path, engine='openpyxl') as writer:

        for sheet in eda_object.sheet_names:
            if f"{sheet}_processed" in eda_object.data:
                df = eda_object.data[f"{sheet}_processed"]

                # Basic statistics
                summary_stats = df.describe()
                summary_stats.to_excel(writer, sheet_name=f"{sheet}_Summary")

                # Missing values analysis
                missing_analysis = pd.DataFrame({
                    'Missing_Count': df.isnull().sum(),
                    'Missing_Percentage': (df.isnull().sum() / len(df)) * 100
                })
                missing_analysis.to_excel(writer, sheet_name=f"{sheet}_Missing", startrow=0)

                # Top patients by revenue
                if 'patient_id' in df.columns:
                    top_patients = df.groupby('patient_id')['total_cost'].sum().sort_values(ascending=False).head(20)
                    top_patients.to_excel(writer, sheet_name=f"{sheet}_TopPatients")

    print(f"EDA summary exported to: {output_path}")

# CUSTOM ANALYSIS FOR SPECIFIC USE CASES

def pharmacy_specific_analysis(df):
    """
    Pharmacy-specific analysis for prescription data
    """
    print("\n💊 PHARMACY-SPECIFIC ANALYSIS")
    print("="*40)

    if 'transaction_type' in df.columns:
        # Prescription vs Procedure analysis
        prescription_data = df[df['transaction_type'] == 'Prescription']
        procedure_data = df[df['transaction_type'] == 'Procedure']

        print(f"Prescription transactions: {len(prescription_data)} ({len(prescription_data)/len(df)*100:.1f}%)")
```

```
            print(f"Procedure transactions: {len(procedure_data)} ({len(procedure_data)/len(df)*100:.1f}%)")

            # Average costs comparison
            if len(prescription_data) > 0:
                print(f"Average prescription cost: ${prescription_data['total_cost'].mean():.2f}")
            if len(procedure_data) > 0:
                print(f"Average procedure cost: ${procedure_data['total_cost'].mean():.2f}")

        # SKU analysis
        if 'sku_id' in df.columns:
            sku_performance = df.groupby('sku_id').agg({
                'total_cost': ['sum', 'mean', 'count'],
                'quantity_consumed': 'sum'
            }).round(2)

            print(f"\nTop 10 SKUs by Revenue:")
            top_skus = sku_performance.sort_values(('total_cost', 'sum'), ascending=False).head(10)
            print(top_skus)

        # Urgency level impact
        if 'urgency_level' in df.columns:
            urgency_analysis = df.groupby('urgency_level').agg({
                'total_cost': ['sum', 'mean', 'count'],
                'formulary_adherent': lambda x: (x == True).sum() / len(x) * 100 if 'formulary_adherent' in df.columns else 0
            }).round(2)

            print(f"\nUrgency Level Analysis:")
            print(urgency_analysis)

    def cost_optimization_insights(df):
        """
        Generate cost optimization insights
        """
        print("\n💰 COST OPTIMIZATION INSIGHTS")
        print("="*40)

        # High-cost, low-adherence transactions
        if 'formulary_adherent' in df.columns:
            high_cost_low_adherence = df[
                (df['total_cost'] > df['total_cost'].quantile(0.75)) &
                (df['formulary_adherent'] == False)
            ]

            potential_savings = high_cost_low_adherence['total_cost'].sum()
            print(f"High-cost, non-adherent transactions: {len(high_cost_low_adherence)}")
            print(f"Potential cost optimization: ${potential_savings:,.2f}")

        # Bounce analysis for cost reduction
        if 'bounced' in df.columns and 'bounce_reason' in df.columns:
            bounced_transactions = df[df['bounced'] == True]
            bounce_cost_impact = bounced_transactions.groupby('bounce_reason')['total_cost'].sum().sort_values(ascending=False)

            print(f"\nBounce Cost Impact by Reason:")
            for reason, cost in bounce_cost_impact.head(5).items():
                print(f"   • {reason}: ${cost:,.2f}")

        # Revenue recovery opportunities
        if 'revenue_lost' in df.columns:
            total_revenue_lost = df['revenue_lost'].sum()
            recoverable_revenue = df[df['bounced'] == True]['revenue_lost'].sum() if 'bounced' in df.columns else 0

            print(f"\nRevenue Recovery Opportunities:")
            print(f"   • Total Revenue Lost: ${total_revenue_lost:,.2f}")
            print(f"   • Potentially Recoverable: ${recoverable_revenue:,.2f}")

    def patient_segmentation_analysis(df):
        """
        Segment patients based on transaction patterns
        """
        print("\n👥 PATIENT SEGMENTATION ANALYSIS")
        print("="*40)

        # Create patient segments based on total cost and frequency
        patient_metrics = df.groupby('patient_id').agg({
            'total_cost': 'sum',
            'transaction_id': 'count', # Use transaction_id for count
            'formulary_adherent': lambda x: (x == True).sum() / len(x) * 100 if 'formulary_adherent' in df.columns else 0
        }).round(2)
```

```
# Define segments using quartiles
cost_q75 = patient_metrics['total_cost'].quantile(0.75)
freq_q75 = patient_metrics['transaction_id'].quantile(0.75)

def segment_patient(row):
    if row['total_cost'] >= cost_q75 and row['transaction_id'] >= freq_q75:
        return 'High-Value Frequent'
    elif row['total_cost'] >= cost_q75:
        return 'High-Value Infrequent'
    elif row['transaction_id'] >= freq_q75:
        return 'Low-Value Frequent'
    else:
        return 'Low-Value Infrequent'

patient_metrics['Segment'] = patient_metrics.apply(segment_patient, axis=1)

segment_summary = patient_metrics.groupby('Segment').agg({
    'total_cost': ['sum', 'mean'],
    'transaction_id': ['sum', 'mean'] # Use transaction_id for sum and mean
}).round(2)

print("Patient Segments:")
print(segment_summary)

# Visualize segments
plt.figure(figsize=(12, 8))
colors = {'High-Value Frequent': 'red', 'High-Value Infrequent': 'orange',
          'Low-Value Frequent': 'blue', 'Low-Value Infrequent': 'green'}

for segment in patient_metrics['Segment'].unique():
    segment_data = patient_metrics[patient_metrics['Segment'] == segment]
    plt.scatter(segment_data['transaction_id'], segment_data['total_cost'],
                label=segment, alpha=0.6, c=colors.get(segment, 'gray'))

plt.xlabel('Transaction Frequency')
plt.ylabel('Total Cost')
plt.title('Patient Segmentation: Cost vs Frequency')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

```
# COMPLETE WORKFLOW FUNCTION
def run_comprehensive_hospital_eda(file_path):
    """
    Complete workflow function that runs all analyses
    """
    print("🏥 HOSPITAL DATA EDA - COMPREHENSIVE ANALYSIS")
    print("="*60)

    # Initialize and load data
    eda = HospitalDataEDA(file_path)
    eda.load_data()

    # Process each sheet
    for sheet in eda.sheet_names:
        print(f"\n🔄 ANALYZING SHEET: {sheet}")
        print("="*50)

        # Run standard EDA
        eda.run_complete_eda(sheet)

        # Run specialized analyses
        processed_df = eda.data[f"{sheet}_processed"]

        pharmacy_specific_analysis(processed_df)
        cost_optimization_insights(processed_df)
        patient_segmentation_analysis(processed_df)

    # Cross-sheet comparison if multiple sheets
    if len(eda.sheet_names) > 1:
        comparison_df = compare_sheets(eda)

    # Export summary
    export_summary_to_excel(eda, f"hospital_eda_summary_{datetime.now().strftime('%Y%m%d_%H%M%S')}.xlsx")

    return eda

# QUICK START INSTRUCTIONS
"""
```

```
SAMPLE OUTPUT INCLUDES:
======================
✓ Basic dataset information and data quality
✓ Missing values and duplicate analysis
✓ Temporal patterns (daily, monthly, hourly trends)
✓ Financial analysis (revenue, costs, losses)
✓ Patient behavior patterns and segmentation
✓ Transaction type analysis
✓ Correlation analysis between variables
✓ Outlier detection and analysis
✓ Pharmacy-specific insights
✓ Cost optimization opportunities
✓ Executive summary report
✓ Visual charts and graphs for all analyses
✓ Exported Excel summary file
"""
```

```
Found 1 sheets: ['Sheet1']
Loaded sheet 'Sheet1' with shape: (550159, 20)

🔄 Processing sheet: Sheet1
Starting Complete EDA for sheet: Sheet1

==================================================
BASIC INFO FOR SHEET: Sheet1
==================================================
Dataset Shape: (550159, 20)
Memory Usage: 316.31 MB

Column Names and Types:
transaction_id                      object
hospital_id                         object
dept_id                             object
physician_id                        object
patient_id                          object
sku_id                              object
transaction_date            datetime64[ns]
transaction_time                    object
quantity_consumed                  float64
unit_cost                          float64
total_cost                         float64
transaction_type                    object
formulary_adherent                    bool
adherence_impact_pct               float64
bounced                               bool
bounce_reason                       object
revenue_lost                       float64
patient_complexity_score           float64
seasonal_factor                    float64
urgency_level                       object
dtype: object

Missing Values:
               Missing Count  Missing Percentage
bounce_reason         491477           89.333629

Duplicate Rows: 0

PREPROCESSING SHEET: Sheet1
========================================
Preprocessing completed. New columns added: year, month, day_of_week, hour

UNIVARIATE ANALYSIS FOR: Sheet1
==================================================

NUMERICAL VARIABLES SUMMARY:
       quantity_consumed      unit_cost     total_cost  adherence_impact_pct  \
count      550159.000000  550159.000000  550159.000000         550159.000000
mean            4.259331      64.932102     254.155645             11.152538
std             3.372505      65.621187     310.553262             14.962219
min             0.290000       2.100000       1.720000              0.000000
25%             1.970000      23.570000      60.000000              0.000000
50%             3.260000      42.208067     138.380000              0.000000
75%             5.430000      74.710000     326.790000             23.400000
max            25.000000     425.466133    4117.350000             45.000000

        revenue_lost  patient_complexity_score  seasonal_factor  \
count  550159.000000             550159.000000    550159.000000
mean       31.036452                  1.784423         1.015447
std       146.591511                  0.557252         0.097979
min         0.000000                  0.800000         0.900000
25%         0.000000                  1.400000         0.900000
50%         0.000000                  1.790000         1.050000
75%         0.000000                  2.100000         1.150000
max      3924.380000                  3.360000         1.150000

       urgency_level      year          month      hour
count            0.0  550159.0  550159.000000  550159.0
mean             NaN    2024.0       6.467727       0.0
std              NaN       0.0       3.535900       0.0
```