Data Exploration and EDA

This notebook contains initial exploration of the M5 dataset and external data sources.

Objectives

1. Load and examine M5 competition data
2. Analyze sales patterns and trends
3. Explore external data relationships
4. Identify data quality issues

In [1]:
```
# Setup and Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')

# Add src to path
import sys
import os
sys.path.append('../src')

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

print("🚀 Data Exploration for Demand Forecasting")
print("=" * 50)
```

```
🚀 Data Exploration for Demand Forecasting
==================================================
```

In [2]:
```
# Load M5 Competition Data with correct paths
print("📁 Loading M5 Competition Data...")

try:
    # Use the correct path structure
    sales_df = pd.read_csv('../src/data/raw/sales_train_validation.csv')
    calendar_df = pd.read_csv('../src/data/raw/calendar.csv')
    prices_df = pd.read_csv('../src/data/raw/sell_prices.csv')
    print("✅ Data loaded successfully!")
except FileNotFoundError as e:
    # Alternative paths to try
    try:
        sales_df = pd.read_csv('data/raw/sales_train_validation.csv')
        calendar_df = pd.read_csv('data/raw/calendar.csv')
        prices_df = pd.read_csv('data/raw/sell_prices.csv')
        print("✅ Data loaded successfully from alternative path!")
```

```
    except FileNotFoundError:
        print("❌ Data files not found. Trying absolute path...")
        import os
        base_path = os.path.dirname(os.path.dirname(os.getcwd()))
        sales_df = pd.read_csv(os.path.join(base_path, 'data', 'raw', 'sales_tra
        calendar_df = pd.read_csv(os.path.join(base_path, 'data', 'raw', 'calend
        prices_df = pd.read_csv(os.path.join(base_path, 'data', 'raw', 'sell_pri
        print("✅ Data loaded with absolute path!")

# Basic information
print(f"\n📊 Dataset Shapes:")
print(f"Sales data: {sales_df.shape}")
print(f"Calendar data: {calendar_df.shape}")
print(f"Prices data: {prices_df.shape}")

print(f"\n🏪 Store Information:")
print(f"States: {sales_df['state_id'].unique()}")
print(f"Stores: {sales_df['store_id'].nunique()}")
print(f"Categories: {sales_df['cat_id'].unique()}")
print(f"Departments: {sales_df['dept_id'].nunique()}")
print(f"Items: {sales_df['item_id'].nunique()}")

# Get sales columns
sales_cols = [col for col in sales_df.columns if col.startswith('d_')]
print(f"Number of sales days: {len(sales_cols)}")
```

```
📂 Loading M5 Competition Data...
✅ Data loaded successfully!

📊 Dataset Shapes:
Sales data: (30490, 1919)
Calendar data: (1969, 14)
Prices data: (6841121, 4)

🏪 Store Information:
States: ['CA' 'TX' 'WI']
Stores: 10
Categories: ['HOBBIES' 'HOUSEHOLD' 'FOODS']
Departments: 7
Items: 3049
Number of sales days: 1913
```

In [3]:

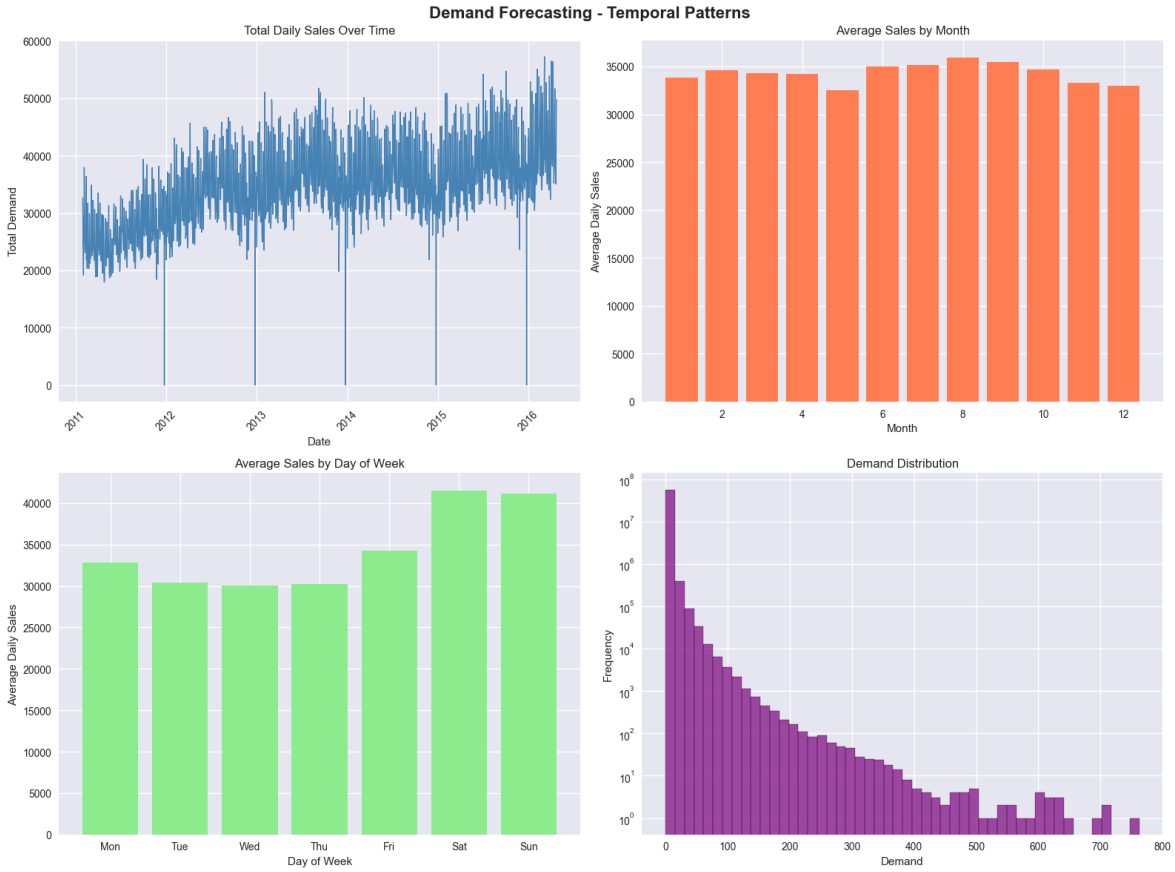
```
# Convert sales data to long format
print("🔄 Converting to long format for analysis...")

sales_cols = [col for col in sales_df.columns if col.startswith('d_')]
print(f"Number of sales days: {len(sales_cols)}")

id_cols = ['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'state_id']
sales_long = pd.melt(
    sales_df,
    id_vars=id_cols,
    value_vars=sales_cols,
    var_name='d',
    value_name='demand'
)

sales_long['day_num'] = sales_long['d'].str.extract('(\d+)').astype(int)

print(f"Long format shape: {sales_long.shape}")
```

```
print(f"Demand statistics:")
print(sales_long['demand'].describe())
```

```
🔄 Converting to long format for analysis...
Number of sales days: 1913
Long format shape: (58327370, 9)
Demand statistics:
count    5.832737e+07
mean     1.126322e+00
std      3.873108e+00
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      1.000000e+00
max      7.630000e+02
Name: demand, dtype: float64
```

In [4]:
```
# Merge with calendar for time series analysis
calendar_df['date'] = pd.to_datetime(calendar_df['date'])
sales_with_dates = sales_long.merge(calendar_df[['d', 'date']], on='d', how='lef

# Daily aggregates
daily_sales = sales_with_dates.groupby('date')['demand'].sum().reset_index()
daily_sales['year'] = daily_sales['date'].dt.year
daily_sales['month'] = daily_sales['date'].dt.month
daily_sales['dayofweek'] = daily_sales['date'].dt.dayofweek
daily_sales['quarter'] = daily_sales['date'].dt.quarter

# Create visualizations
fig, axes = plt.subplots(2, 2, figsize=(16, 12))
fig.suptitle('Demand Forecasting - Temporal Patterns', fontsize=16, fontweight='

# Time series plot
axes[0, 0].plot(daily_sales['date'], daily_sales['demand'], linewidth=1, color='
axes[0, 0].set_title('Total Daily Sales Over Time')
axes[0, 0].set_xlabel('Date')
axes[0, 0].set_ylabel('Total Demand')
axes[0, 0].tick_params(axis='x', rotation=45)

# Monthly patterns
monthly_avg = daily_sales.groupby('month')['demand'].mean()
axes[0, 1].bar(monthly_avg.index, monthly_avg.values, color='coral')
axes[0, 1].set_title('Average Sales by Month')
axes[0, 1].set_xlabel('Month')
axes[0, 1].set_ylabel('Average Daily Sales')

# Day of week patterns
dow_avg = daily_sales.groupby('dayofweek')['demand'].mean()
dow_labels = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
axes[1, 0].bar(range(7), dow_avg.values, color='lightgreen')
axes[1, 0].set_title('Average Sales by Day of Week')
axes[1, 0].set_xlabel('Day of Week')
axes[1, 0].set_ylabel('Average Daily Sales')
axes[1, 0].set_xticks(range(7))
axes[1, 0].set_xticklabels(dow_labels)

# Demand distribution
axes[1, 1].hist(sales_long['demand'], bins=50, alpha=0.7, color='purple', edgecc
axes[1, 1].set_title('Demand Distribution')
axes[1, 1].set_xlabel('Demand')
```

```
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].set_yscale('log')

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



In [5]:
```
# Category Analysis
print("🏷️ Category Analysis")
print("=" * 30)

# Category performance
cat_performance = sales_with_dates.groupby('cat_id').agg({
    'demand': ['sum', 'mean', 'std', 'count']
}).round(2)

cat_performance.columns = ['Total_Sales', 'Avg_Sales', 'Std_Sales', 'Records']
cat_performance = cat_performance.sort_values('Total_Sales', ascending=False)
print("Category Performance:")
print(cat_performance)

# Zero sales analysis
zero_sales_by_cat = sales_with_dates.groupby('cat_id').apply(
    lambda x: (x['demand'] == 0).mean() * 100
).round(2)

print(f"\nZero Sales Percentage by Category:")
for cat, pct in zero_sales_by_cat.items():
    print(f"  {cat}: {pct}%")

# Visualize
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

axes[0].bar(cat_performance.index, cat_performance['Total_Sales'], color='skyblu
```

```
axes[0].set_title('Total Sales by Category')
axes[0].set_xlabel('Category')
axes[0].set_ylabel('Total Sales')

axes[1].bar(zero_sales_by_cat.index, zero_sales_by_cat.values, color='lightcoral
axes[1].set_title('Zero Sales % by Category')
axes[1].set_xlabel('Category')
axes[1].set_ylabel('Zero Sales %')

# Sales distribution by category
for cat in sales_with_dates['cat_id'].unique():
    cat_data = sales_with_dates[sales_with_dates['cat_id'] == cat]['demand']
    cat_data_nonzero = cat_data[cat_data > 0]
    if len(cat_data_nonzero) > 0:
        axes[2].hist(cat_data_nonzero, bins=30, alpha=0.6, label=cat, density=Tr

axes[2].set_title('Sales Distribution by Category')
axes[2].set_xlabel('Demand')
axes[2].set_ylabel('Density')
axes[2].legend()
axes[2].set_yscale('log')

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

```
🏷️ Category Analysis
==============================
Category Performance:
           Total_Sales  Avg_Sales  Std_Sales   Records
cat_id
FOODS         45089939       1.64       5.15  27489810
HOUSEHOLD     14480670       0.72       2.09  20029110
HOBBIES        6124800       0.57       2.04  10808450

Zero Sales Percentage by Category:
  FOODS: 62.02%
  HOBBIES: 77.28%
  HOUSEHOLD: 71.77%
```

In [8]:
```
# Summary insights
print("💡 KEY INSIGHTS")
print("=" * 40)

# Calculate all the required variables
total_items = sales_df['item_id'].nunique()
total_stores = sales_df['store_id'].nunique()
total_days = len(sales_cols)
avg_daily_demand = sales_long['demand'].mean()
```

```
# Calculate zero sales percentage (this was missing!)
zero_sales_pct = (sales_long['demand'] == 0).mean() * 100

insights = [
    f"📊 Dataset: {total_items:,} products across {total_stores} stores",
    f"🗓️ Time period: {total_days} days of sales data",
    f"📈 Average daily demand: {avg_daily_demand:.2f} units",
    f"🔢 Zero sales: {zero_sales_pct:.1f}% of observations",
    f"💰 Price range: ${prices_df['sell_price'].min():.2f} - ${prices_df['sell_
    f"🏪 Total sales records: {len(sales_long):,}"
]

for insight in insights:
    print(f"  • {insight}")

print(f"\n🎯 NEXT STEPS:")
print("  1. ✅ Basic data exploration complete")
print("  2. 🔄 Ready for feature engineering")
print("  3. 🤖 Ready for model training")
print("  4. 📊 Ready for advanced analysis")

print("\n" + "=" * 40)
print("✅ Data exploration completed successfully!")
print("🚀 Ready to proceed with ML pipeline!")
```

```
💡 KEY INSIGHTS
========================================
  • 📊 Dataset: 3,049 products across 10 stores
  • 🗓️ Time period: 1913 days of sales data
  • 📈 Average daily demand: 1.13 units
  • 🔢 Zero sales: 68.2% of observations
  • 💰 Price range: $0.01 - $107.32
  • 🏪 Total sales records: 58,327,370

🎯 NEXT STEPS:
  1. ✅ Basic data exploration complete
  2. 🔄 Ready for feature engineering
  3. 🤖 Ready for model training
  4. 📊 Ready for advanced analysis

========================================
✅ Data exploration completed successfully!
🚀 Ready to proceed with ML pipeline!
```

In [ ]: