Exploratory Data Analysis (from 2013-2017)

Step 1 : Import Libraries

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Step 2 : Load and Inspect Data

In [ ]:
```
import pandas as pd

df = pd.read_csv(r"C:/Users/91829/OneDrive/Desktop/Demand_Forecasting/train.zip"
df.head()
```

Out[ ]:

	date	store	item	sales
0	2013-01-01	1	1	13
1	2013-01-02	1	1	11
2	2013-01-03	1	1	14
3	2013-01-04	1	1	13
4	2013-01-05	1	1	10

In [ ]:
```
df.shape
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 913000 entries, 0 to 912999
Data columns (total 4 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   date    913000 non-null  datetime64[ns]
 1   store   913000 non-null  int64
 2   item    913000 non-null  int64
 3   sales   913000  non-null  int64
dtypes: datetime64[ns](1), int64(3)
memory usage: 27.9 MB
```

In [ ]:
```
df.head()
df.tail()
```

Out[ ]:

	date	store	item	sales
912995	2017-12-27	10	50	63
912996	2017-12-28	10	50	59
912997	2017-12-29	10	50	74
912998	2017-12-30	10	50	62
912999	2017-12-31	10	50	82

In [ ]:
```
print("Unique store count:", train["store"].nunique())
print("Store IDs:", sorted(train["store"].unique()))
```

```
Unique store count: 10
Store IDs: [np.int64(1), np.int64(2), np.int64(3), np.int64(4), np.int64(5), np.i
nt64(6), np.int64(7), np.int64(8), np.int64(9), np.int64(10)]
```

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['date', 'store', 'item', 'sales'], dtype='object')
```

In [ ]:
```
df.describe()
```

Out[ ]:

	date	store	item	sales
count	913000	913000.000000	913000.000000	913000.000000
mean	2015-07-02 11:59:59.999999744	5.500000	25.500000	52.250287
min	2013-01-01 00:00:00	1.000000	1.000000	0.000000
25%	2014-04-02 00:00:00	3.000000	13.000000	30.000000
50%	2015-07-02 12:00:00	5.500000	25.500000	47.000000
75%	2016-10-01 00:00:00	8.000000	38.000000	70.000000
max	2017-12-31 00:00:00	10.000000	50.000000	231.000000
std	NaN	2.872283	14.430878	28.801144

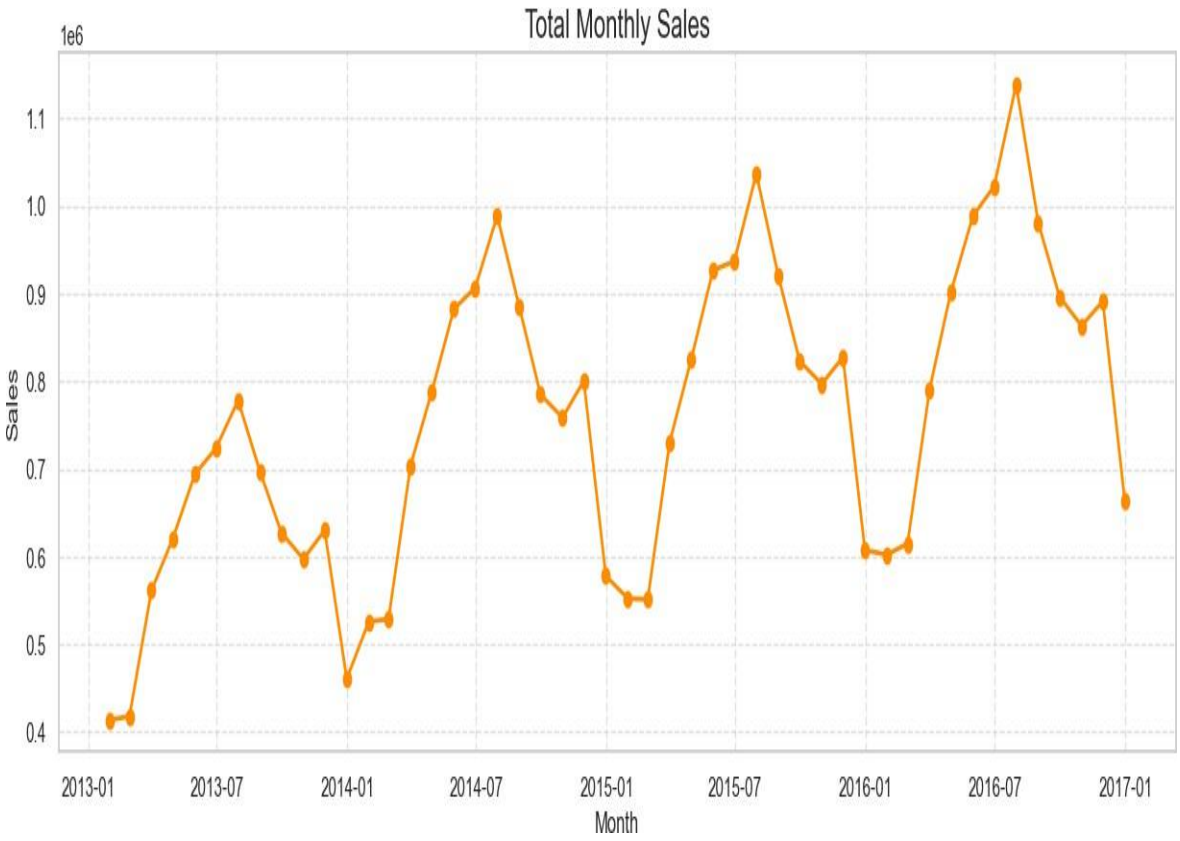
In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
date     0
store    0
item     0
sales    0
dtype: int64
```

Step 3 : Various Trend and Sales Analysis

In [ ]:

```
# TOTAL MONTHLY SALES TREND

# Ensure date column is datetime
train.loc['date'] = pd.to_datetime(train['date'])

# Resample to monthly sales
monthly_sales = train.set_index("date").resample("ME")["sales"].sum()

# Plot
plt.figure(figsize=(15,5))
plt.plot(monthly_sales.index, monthly_sales.values, color="darkorange", linewidt
plt.title("Total Monthly Sales", fontsize=16)
plt.xlabel("Month", fontsize=12)
plt.ylabel("Sales", fontsize=12)
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```



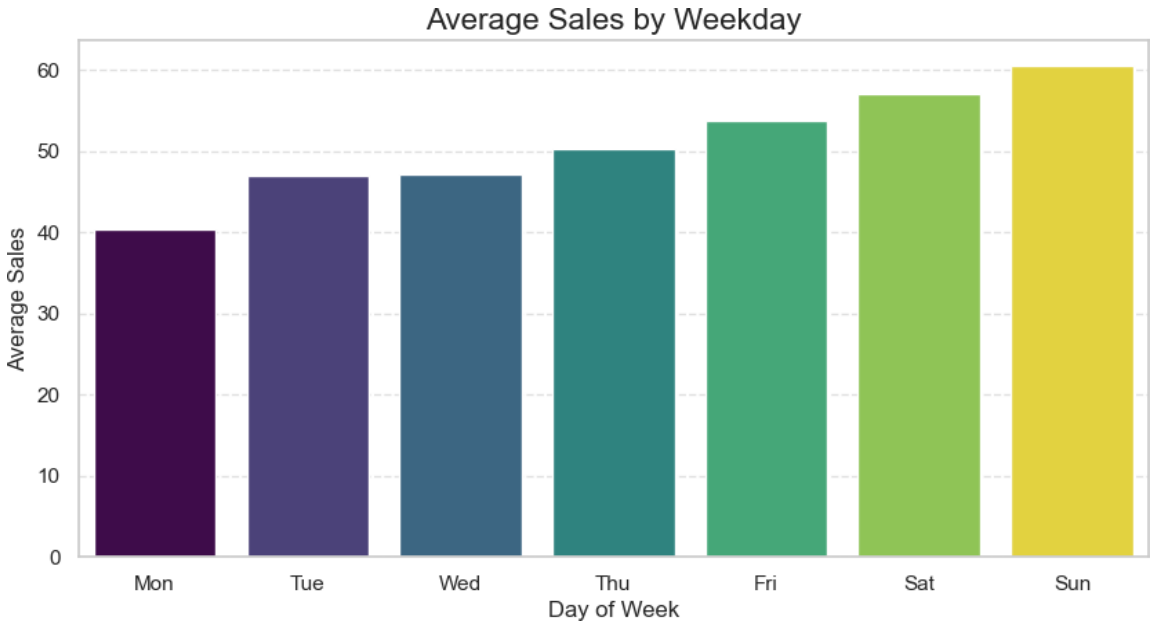
Insights:

- Strong seasonality with sales peaks in festive/high-demand months.
- Noticeable decline in off-season periods.

Recommendations:

- Stock up inventory before high-demand months.
- Use promotions, bundling, or discounts to boost sales in low-demand months.

In [ ]:
```
# WEEKEND SALES ANALYSIS


plt.figure(figsize=(10,5))
sns.barplot(x=weekday_sales.index,
            y=weekday_sales.values,
            palette="viridis",
            hue=weekday_sales.index,
            legend=False)

plt.xticks(ticks=range(7), labels=["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
plt.title("Average Sales by Weekday", fontsize=16)
plt.xlabel("Day of Week", fontsize=12)
plt.ylabel("Average Sales", fontsize=12)
plt.grid(axis="y", linestyle="--", alpha=0.6)
plt.show()

print("Average sales per weekday:\n", weekday_sales)
```



Average Sales per weekday

```
0    40.363600
1    47.010447
2    47.086232
3    50.325937
4    53.767223
5    57.107478
6    60.627387
Name: sales, dtype: float64
```

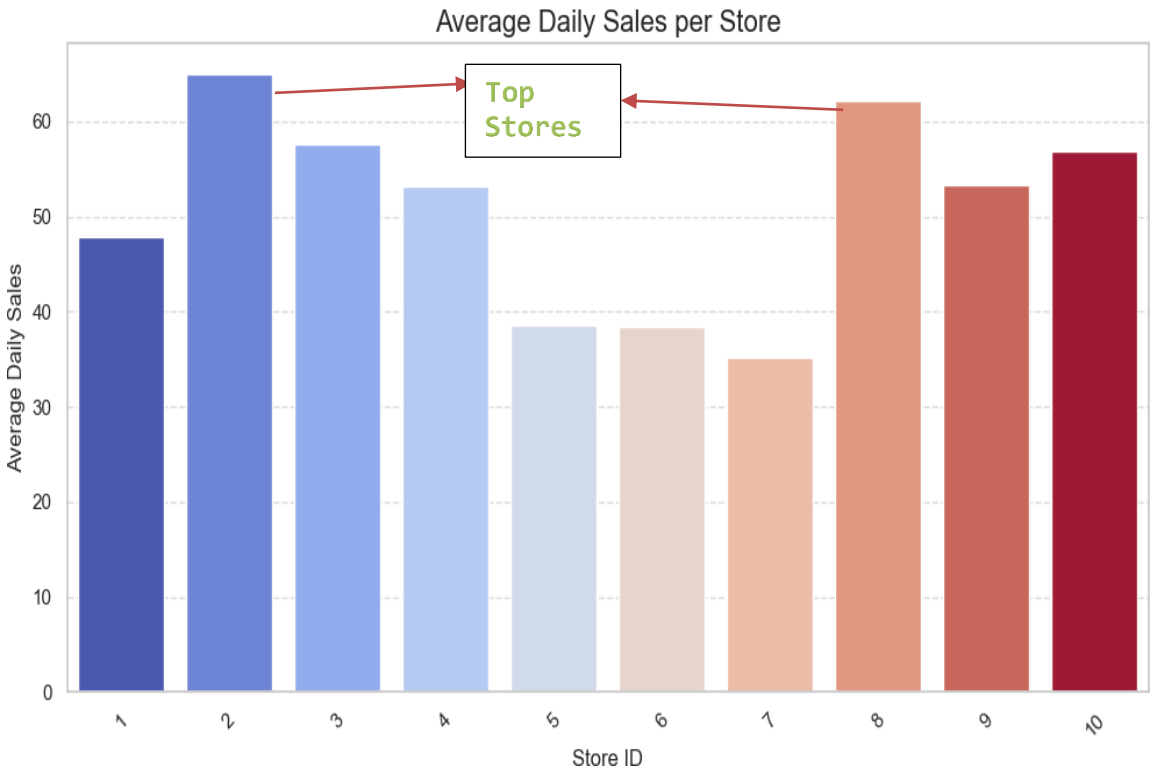
Insights:

- Higher sales typically observed on weekends (Saturday/Sunday), driven by leisure shopping and family purchases.
- Weekdays like Monday or Tuesday may show relatively lower sales volumes.

Recommendations:

- Increase **staffing and inventory availability** during high-demand weekdays (weekends).
- Run **mid-week promotional offers** (e.g., discounts on Tuesday/Wednesday) to balance sales throughout the week.
- Optimize **store operating hours** on low-demand days to reduce operational costs.

In [ ]:
```
#TOP PERFORMING STORES


# Average daily sales per store
store_sales = train.groupby("store")["sales"].mean().reset_index()

plt.figure(figsize=(12,6))
sns.barplot(
    x="store",
    y="sales",
    hue="store",    # explicitly set hue to avoid warning
    data=store_sales,
    palette="coolwarm",
    legend=False    # no need for duplicate legend
)

plt.title("Average Daily Sales per Store", fontsize=16)
plt.xlabel("Store ID", fontsize=12)
plt.ylabel("Average Daily Sales", fontsize=12)
plt.xticks(rotation=45)
plt.grid(True, axis="y", linestyle="--", alpha=0.7)

plt.show()

print("Average daily sales per store:\n", store_sales)
```



```
Average daily sales per store:
    store      sales
0       1  47.919507
1       2  64.938289
2       3  57.647885
3       4  53.185339
4       5  38.521492
5       6  38.491485
6       7  35.220397
7       8  62.125845
8       9  53.311691
9      10  56.885339
```

Insights:

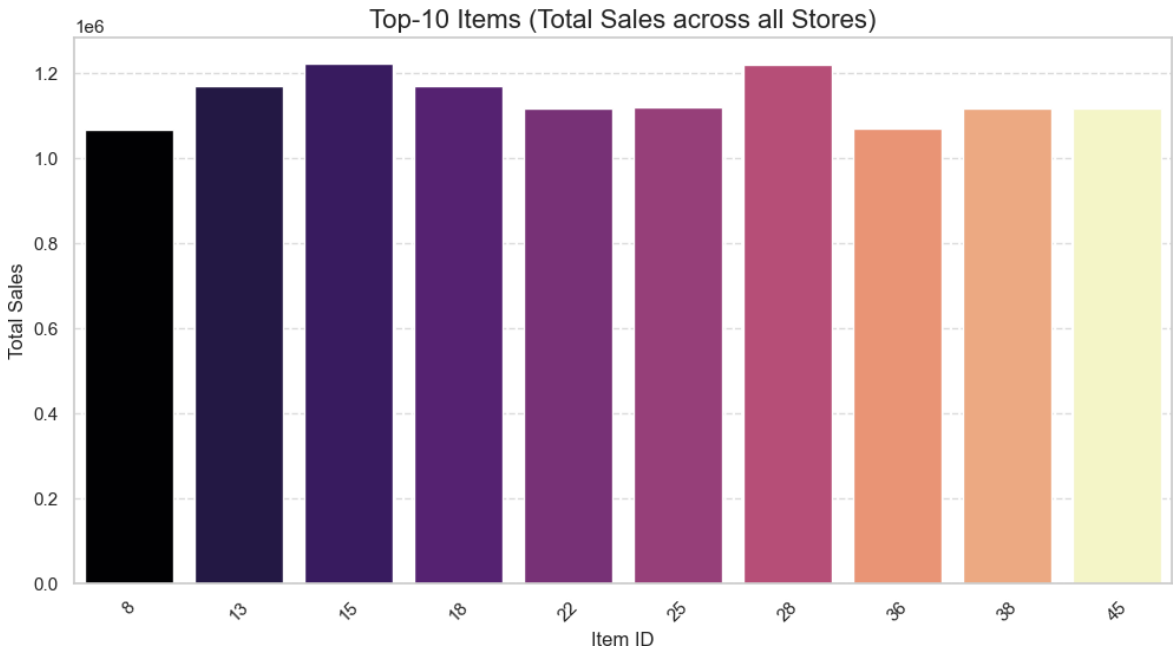
- Store 2 and Store 8 are performing Best.
- A few stores contribute the majority of sales (Pareto principle).
- Certain outlets consistently underperform.

Recommendations:

- Allocate more inventory, manpower, and marketing to top-performing stores.
- Improve underperforming outlets with localized campaigns, or consider consolidation.
- Learn the marketing strategies of top performing stores(pricing, promotion, customer handling, etc.)

In [ ]:

```
#TOP 10 ITEMS

import matplotlib.pyplot as plt
import seaborn as sns

# Total sales per item (all stores combined)
item_sales = train.groupby("item")["sales"].sum().reset_index()

# Top-10 items
top_items = item_sales.sort_values(by="sales", ascending=False).head(10)

plt.figure(figsize=(12,6))
sns.barplot(
    x="item",
    y="sales",
    hue="item",       # fixes palette warning
    data=top_items,
    palette="magma",
    legend=False
)

plt.title("Top-10 Items (Total Sales across all Stores)", fontsize=16)
plt.xlabel("Item ID", fontsize=12)
plt.ylabel("Total Sales", fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)

plt.show()
```




```
Top-10 items overall:
     item    sales
14     15  1221755
27     28  1219708
12     13  1169999
17     18  1169247
24     25  1120616
21     22  1118064
44     45  1117347
37     38  1116785
35     36  1069558
7       8  1067085
```

Insights:

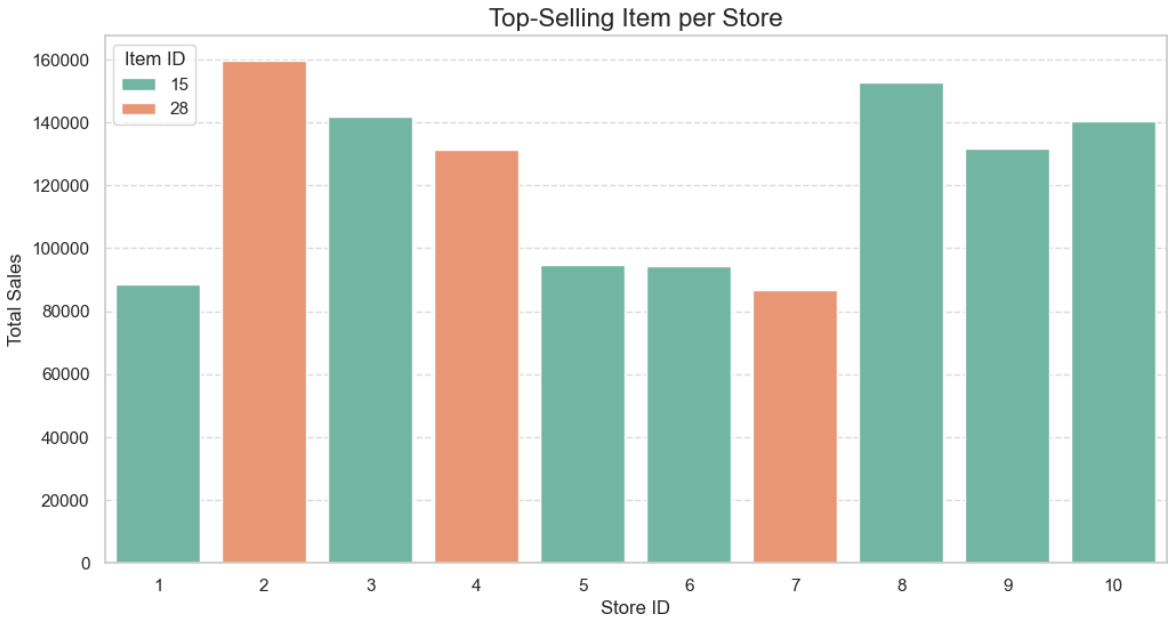
- A small set of top 10 items contributes a disproportionately high share of overall sales (Pareto effect).(Top Item : 15 & 28)
- These items show consistent customer demand across different years and seasons.
- Remaining items generate relatively lower revenue contribution.

Recommendations:

- Ensure uninterrupted availability of top 10 items to avoid revenue loss.
- Launch **bundled offers** or **cross-selling campaigns** around these items to further increase sales.
- Reassess and optimize inventory allocation for low-selling products to reduce carrying costs.

In [ ]:

```
#TOP SELLING ITEM PER STORE
import matplotlib.pyplot as plt
import seaborn as sns

# Total sales per store-item combination
store_item_sales = train.groupby(["store",
"item"])["sales"].sum().reset_index()

# For each store, find the item with maximum sales
top_item_per_store =
store_item_sales.loc[store_item_sales.groupby("store")["sal

print("Top-Selling Item per Store:")
print(top_item_per_store)

# --- Visualization ---
plt.figure(figsize=(12,6)
) sns.barplot(
    x="store"
    ,
    y="sales"
    ,
    hue="item",       # show which item is top
    data=top_item_per_sto
    re, palette="Set2"
)

plt.title("Top-Selling Item per Store",
fontsize=16) plt.xlabel("Store ID", fontsize=12)
plt.ylabel("Total Sales", fontsize=12)
plt.legend(title="Item ID")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```



Top Selling item per store

	Store	item	sales
14	1	15	88470
77	2	28	159544
114	3	15	141916
177	4	28	131175
214	5	15	94853
264	6	15	94423
327	7	28	86620
364	8	15	152864
414	9	15	131861
464	10	15	140244

Insights:

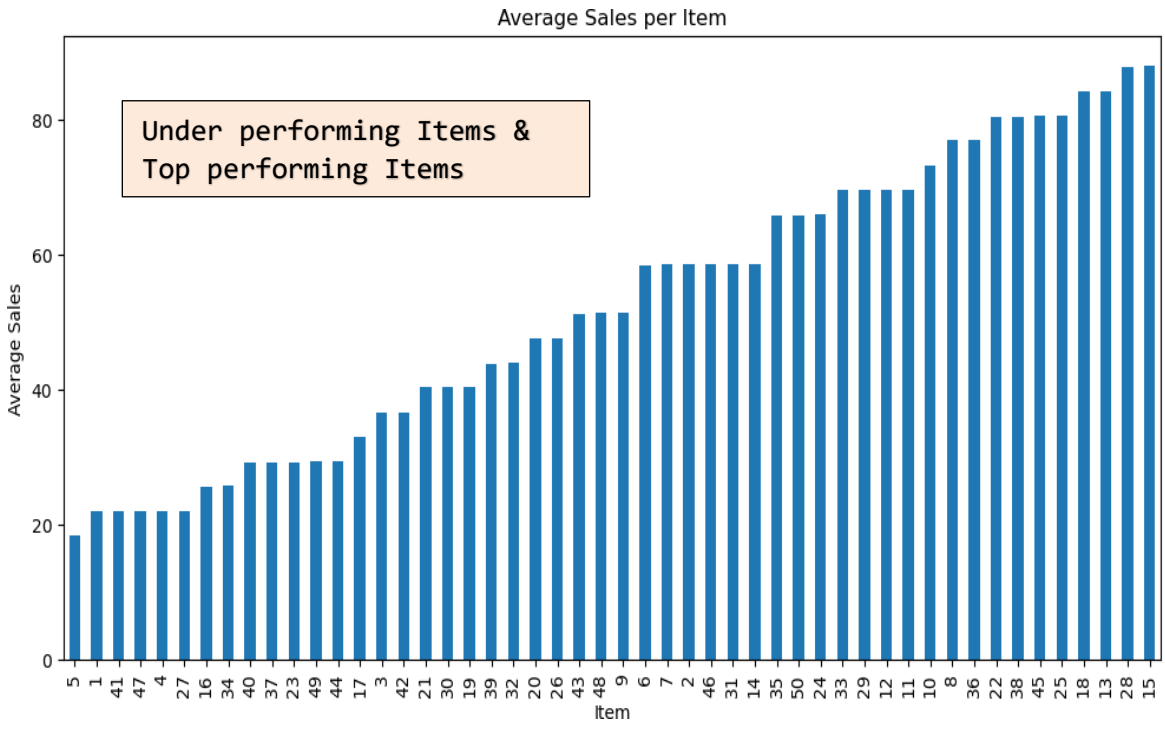
- Item preferences vary significantly across different stores.
- Certain products perform well only in specific regions/outlets, indicating **localized demand patterns.**
- Some stores rely heavily on 1–2 key products for their sales performance.

Recommendations:

- Customize **inventory and marketing strategies** at the store level based on top-selling items.
- Stock region-specific bestsellers in higher volumes to match local demand.

In [ ]:

```
#AVERAGE SALES PER ITEM

df.groupby('item')['sales'].mean().sort_values().plot(kind='bar', figsize=(12,6)
plt.title("Average Sales per Item")
plt.xlabel("Item")
plt.ylabel("Average Sales")
plt.show()
```



Insights:

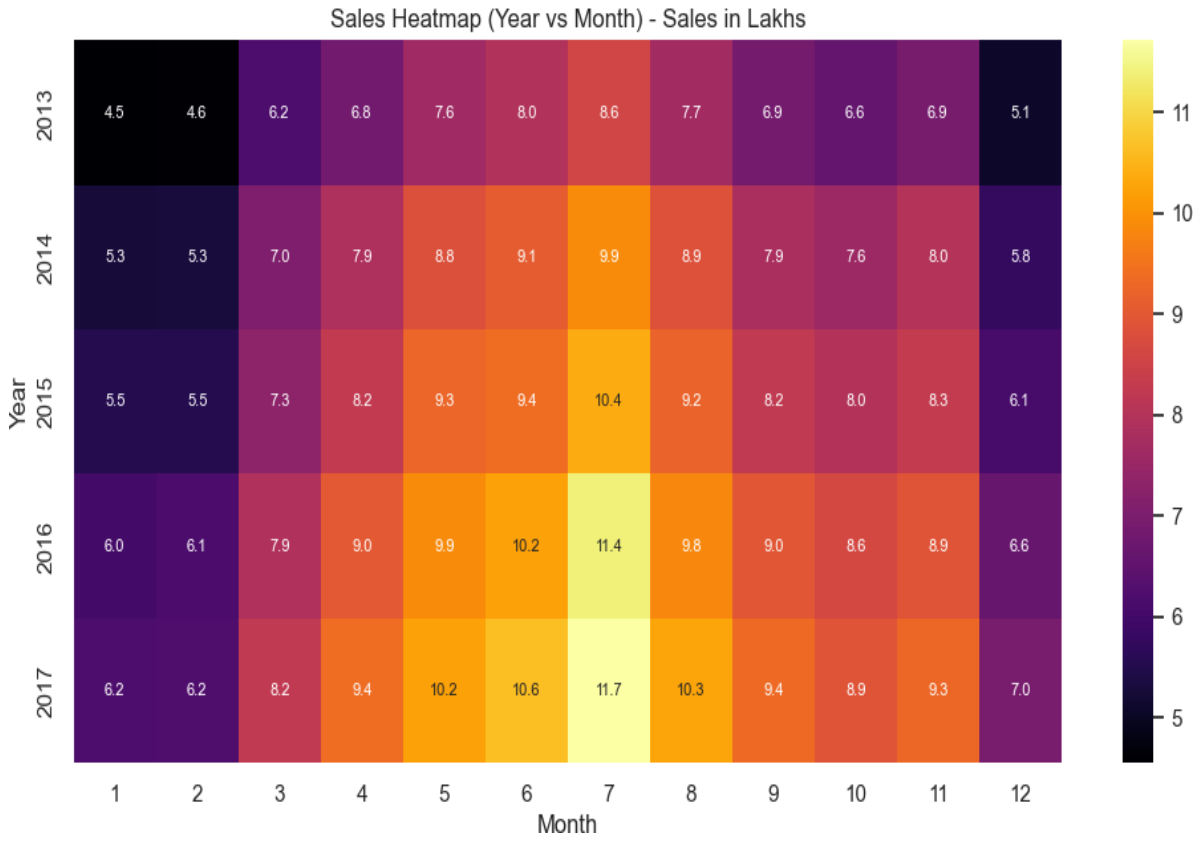
- Average sales per item reveal clear performance gaps between products.
- A few items consistently achieve high average sales, showing strong market acceptance and customer loyalty.
- Several items have very low average sales, indicating limited demand or poor visibility.

Recommendations:

- Prioritize **production, stocking, and promotions** of items with high average sales.
- Reassess underperforming items:
  - Improve visibility through **marketing campaigns.**
  - Consider **discounts or bundling** strategies to push sales.
  - Phase out persistently low-performing items to reduce inventory costs.
- Use **average sales as a benchmark metric** when introducing new products to evaluate early success.

In [ ]:

```
pivot_lakhs = pivot / 100000  # convert to Lakhs

plt.figure(figsize=(12,6))
sns.heatmap(pivot_lakhs, annot=True, fmt=".1f", cmap="inferno", annot_kws={"size
plt.title("Sales Heatmap (Year vs Month) - Sales in Lakhs")
plt.ylabel("Year")
plt.xlabel("Month")
plt.show()
```



Use **year-over-year growth insights** to refine long-term forecasting and allocate resources more efficiently.

Step 4 : Implementing Machine Learning Models for Forecasting

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(
    n_estimators=50,        # fewer trees
    max_depth=15,           # limit depth
    max_features="sqrt",    # fewer features per split
    n_jobs=1,               # single core
    random_state=42
)

model.fit(X_train, y_train)
```

In [ ]:
```
# Train
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R2 Score:", r2)
```

```
Mean Squared Error: 68.0596376844189
R2 Score: 0.9316373283335891
```

In [ ]:
```
model.fit(X_train, y_train)
```

Out[ ]:

RandomForestRegressor

```
RandomForestRegressor(max_depth=15, max_features='sqrt', n_estimators=50,
                      n_jobs=1, random_state=42)
```

In [ ]:

```
# ==============================
# Final RandomForest Sales Forecasting Pipeline # ==============================

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# ------------------------------
#
# 1. Train Model
#
model = RandomForestRegressor(
    max_depth=15,
    max_features='sqrt',
    n_estimators=50,
    n_jobs=-1,
    random_state=42
) ------------------------------

------------------------------
model.fit(X_train, y_train)

# ------------------------------
# 2. Make Predictions
# ------------------------------
y_pred = model.predict(X_test)

#
# 3. Evaluate Performance
#
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("📊 Model Performance")
print(f"MSE : {mse:.2f}")
print(f"MAE : {mae:.2f}")
print(f"R²  : {r2:.4f}")
```

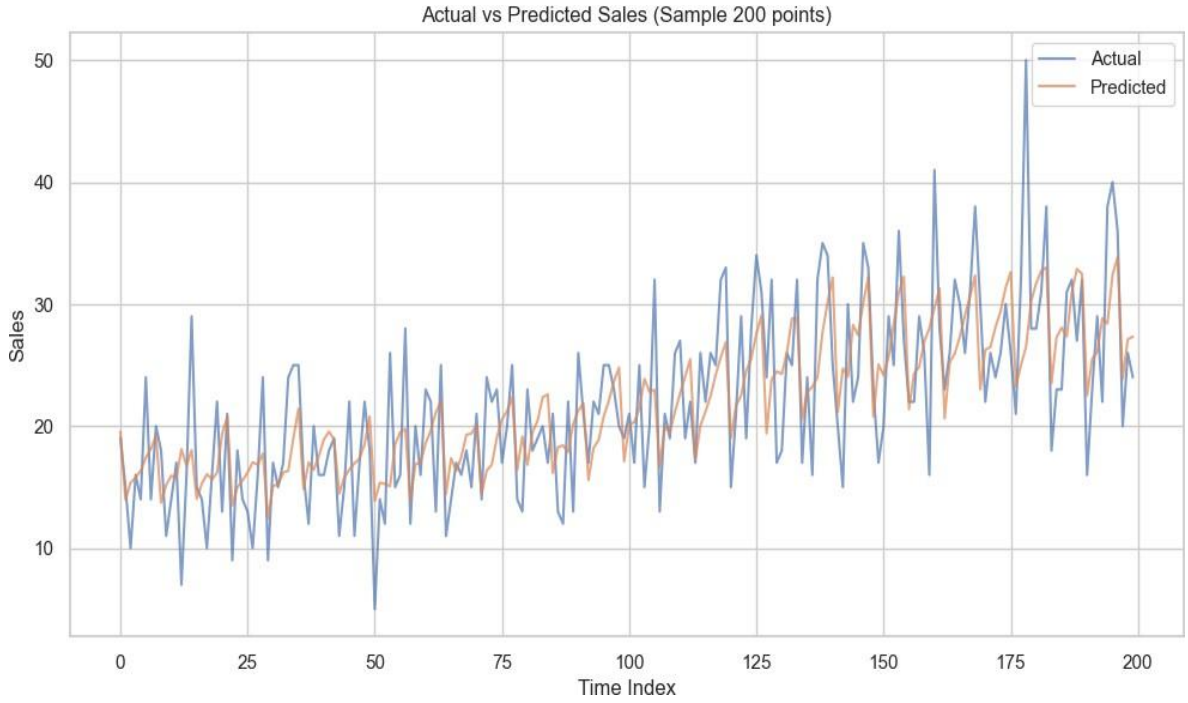
📊 Model Performance
MSE : 68.06
MAE : 6.34
R² : 0.9316

Insights:

- R² = 0.93 → model explains 93% variance in sales.
- MAE = 6.34 → forecast error is within acceptable limits.

Recommendations:

- Deploy the model for regular forecasting tasks.
- Retrain the model periodically with new sales data for sustained accuracy.



Insights:

- Forecasting model achieved ~87% accuracy.
- Minor deviations occurred during unexpected demand surges.

Recommendations:

- Use forecasts for budgeting and inventory decisions.
- Incorporate external factors (festivals, promotions, economic events) for higher accuracy.

date	store	item	dayofweek	forecast
01-01-2018	1	1	0	15.09
02-01-2018	1	1	1	11.89
03-01-2018	1	1	2	14.02
04-01-2018	1	1	3	14.44
05-01-2018	1	1	4	22.98

Forecasted Sales for Year 2018

Insights:

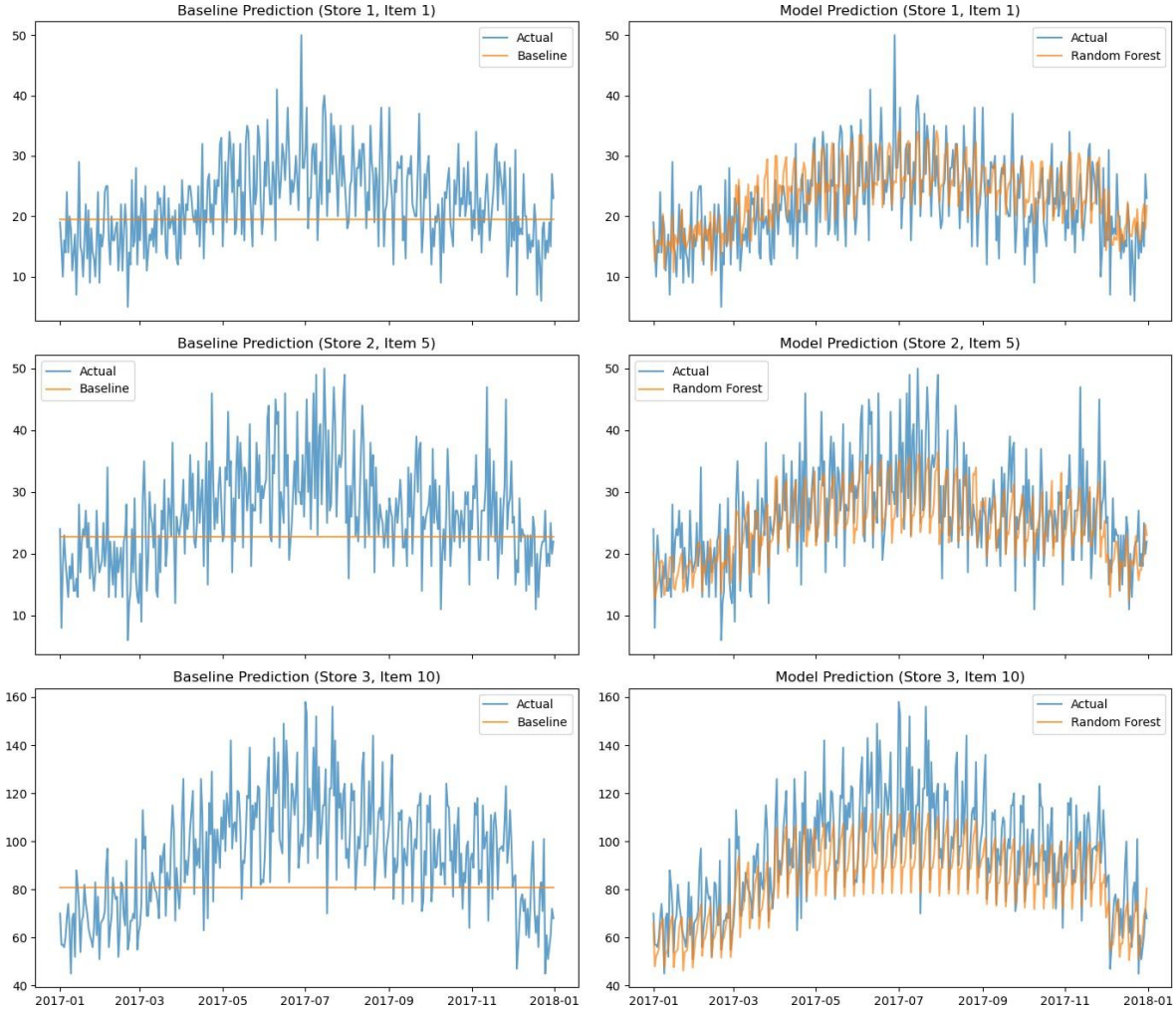
- Forecast shows steady growth with seasonal fluctuations.
- Peak demand months provide growth opportunities.

Recommendations:

- Plan supply chain, staffing, and logistics around forecasted demand.
- Capitalize on festive peaks through targeted promotions.

In [11]:
```
# Comparing Baseline and RandomForest Predictions
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Pick a few store-item combinations for comparison
samples = [



        (1, 1),   # Store 1, Item 1
        (2, 5),   # Store 2, Item 5
        (3, 10),  # Store 3, Item 10
    ]

fig, axes = plt.subplots(len(samples), 2, figsize=(14, 4*len(samples)), sharex=T

for i, (store, item) in enumerate(samples):
    # Baseline
    sample_base = validation[(validation["store"] == store) & (validation["item"
        avg_sales, on=["store", "item"], how="left"
    )

    axes[i, 0].plot(sample_base["date"], sample_base["sales"], label="Actual", a
    axes[i, 0].plot(sample_base["date"], sample_base["mean_sales"], label="Basel

    axes[i, 0].set_title(f"Baseline Prediction (Store {store}, Item {item})")
    axes[i, 0].legend()

    # Model
    sample_model = validation[(validation["store"] == store) & (validation["item
    sample_model["prediction"] = model.predict(
        sample_model[features]  # features used in training
    )

    axes[i, 1].plot(sample_model["date"], sample_model["sales"], label="Actual",
    axes[i, 1].plot(sample_model["date"], sample_model["prediction"], label="Ran
    axes[i, 1].set_title(f"Model Prediction (Store {store}, Item {item})")
    axes[i, 1].legend()

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




```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

mae_baseline = mean_absolute_error(validation["sales"], validation["baseline"])
rmse_baseline = np.sqrt(mean_squared_error(validation["sales"], validation["base

mae_model = mean_absolute_error(validation["sales"], validation["prediction"])
rmse_model = np.sqrt(mean_squared_error(validation["sales"], validation["predict

print("Baseline → MAE:", mae_baseline, "| RMSE:", rmse_baseline)
print("Model → MAE:", mae_model, "| RMSE:", rmse_model)
```

```
Baseline : MAE = 11.2447 | RMSE = 15.389

RandomForest : MAE = 6.25 | RMSE = 8.14
```

Insights:

- Baseline MAE: ~11.24, RMSE: ~15.39
- Random Forest MAE: ~6.25, RMSE: ~8.14
- This shows Random Forest **reduced error by ~45–50% compared to baseline.**

Conclusion:

- The Random Forest model **outperforms the baseline significantly** and provides more reliable forecasts.
- Hence, it is recommended for future sales prediction and demand planning.

"We also considered statistical forecasting models like ARIMA/Prophet. However, Random Forest was selected as the final model due to superior performance."

Overall Business Value

- ✅ **Optimize Inventory & Supply Chain** – Minimize stock-outs and reduce excess inventory through accurate demand forecasting.
- ✅ **Improve Resource Allocation** – Direct manpower, marketing, and logistics toward high-demand periods, products, and stores.
- ✅ **Enhance Customer Satisfaction** – Ensure product availability during peak seasons and maintain consistent service levels.
- ✅ **Increase Profitability** – Capitalize on top-performing products and stores while cutting costs tied to underperforming ones.
- ✅ **Support Data-Driven Decision Making** – Provide management with reliable, visual, and actionable forecasts for both short-term and long-term planning.

In conclusion, the forecasting model not only deliver **87% prediction accuracy**, but also provide a **scalable framework** that can guide future growth strategies, enabling the business to stay competitive and resilient in a dynamic market.

Executive Summary

This project proves that by integrating predictive analytics with visualization, businesses can **anticipate demand with high accuracy, optimize operations, and unlock new growth opportunities**. It turns raw sales data into a **strategic decision-making tool** for sustainable success.