Sales Forecasting for Inventory Management – Analysis

Data Overview

The dataset contains 9,800 records with 18 columns, covering order details, customer information, product categories, and sales amounts. Missing values in the Postal Code column need to be handled. Order Date & Sales are key features for time-series forecasting.


SALES FORECASTING
FOR INVENTORY MANAGEMENT
SALES
SALES
32%
TOP-SELLING

Out[3]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Pro
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR 1000
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420.0	South	FUR 1000
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036.0	West	OFI 1000
3	4	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	FUI 1000
4	5	US-2016-108966	11/10/2016	18/10/2016	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311.0	South	OF 1000

In [4]:
```
df.tail()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Row ID         9800 non-null   int64  
 1   Order ID       9800 non-null   object 
 2   Order Date     9800 non-null   object 
 3   Ship Date      9800 non-null   object 
 4   Ship Mode      9800 non-null   object 
 5   Customer ID    9800 non-null   object 
 6   Customer Name  9800 non-null   object 
 7   Segment        9800 non-null   object 
 8   Country        9800 non-null   object 
 9   City           9800 non-null   object 
 10  State          9800 non-null   object 
 11  Postal Code    9789 non-null   float64
 12  Region         9800 non-null   object 
 13  Product ID     9800 non-null   object 
 14  Category       9800 non-null   object 
 15  Sub-Category   9800 non-null   object 
 16  Product Name   9800 non-null   object 
 17  Sales          9800 non-null   float64
dtypes: float64(2), int64(1), object(15)
memory usage: 1.3+ MB
```

In [6]:
```
df.isnull().sum()
```

```
            df[col].fillna(df[col].mode()[0], inplace=True)  # Fill categorical with mode
        else:
            df[col].fillna(df[col].mean(), inplace=True)  # Fill numerical with mean
```

```
C:\Users\hi\AppData\Local\Temp\ipykernel_17360\2086525780.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(df[col].mean(), inplace=True)  # Fill numerical with mean
C:\Users\hi\AppData\Local\Temp\ipykernel_17360\2086525780.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df[col].fillna(df[col].mode()[0], inplace=True)  # Fill categorical with mode
```

In [10]:

```
# Encoding categorical features
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    label_encoders[col] = LabelEncoder()
    df[col] = label_encoders[col].fit_transform(df[col])
```

In [11]:

```
#normilizing numerical features
scaler=StandardScaler()
num_cols=df.select_dtypes(include=['float64','int64']).columns
df[num_cols]=scaler.fit_transform(df[num_cols])
```

In [12]:

```
df.to_csv("cleaned_customer_data.csv",index=False)
```

In [13]:

```
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9800 non-null   float64
 1   Order ID       9800 non-null   int32
 2   Order Date     9800 non-null   datetime64[ns]
 3   Ship Date      9800 non-null   int32
 4   Ship Mode      9800 non-null   int32
 5   Customer ID    9800 non-null   int32
 6   Customer Name  9800 non-null   int32
 7   Segment        9800 non-null   int32
 8   Country        9800 non-null   int32
 9   City           9800 non-null   int32
 10  State          9800 non-null   int32
 11  Postal Code    9800 non-null   float64
 12  Region         9800 non-null   int32
 13  Product ID     9800 non-null   int32
 14  Category       9800 non-null   int32
 15  Sub-Category   9800 non-null   int32
 16  Product Name   9800 non-null   int32
 17  Sales          9800 non-null   float64
 18  Order Year     9800 non-null   int32
 19  Order Month    9800 non-null   int32
 20  Order Day      9800 non-null   int32
dtypes: datetime64[ns](1), float64(3), int32(17)
memory usage: 957.2 KB
Data after preprocessing:
 None
```

In [18]:
```
df.to_csv("cleaned_sales_data.csv", index=False)
```

In [19]:
```
df.describe().T
```

	count	mean	min	25%	50%	75%	max	std
Product Name	9800.0	922.172449	0.0	475.0	906.0	1389.0	1848.0	530.960771
Sales	9800.0	-0.0	-0.367567	-0.340751	-0.281317	-0.032179	35.759654	1.000051
Order Year	9800.0	1970.0	1970.0	1970.0	1970.0	1970.0	1970.0	0.0
Order Month	9800.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Order Day	9800.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0

In [20]:

```
#chake the distribution of sales
plt.figure(figsize=(8,6))
sns.histplot(df['Sales'],bins=50,kde=True,color='blue')
plt.title("sales distributions")
plt.xlabel("sales Distribution")
plt.ylabel("Frequency")
plt.show()
```

```
sales_trend = df.groupby('Order Date')['Sales'].sum()

# Plot sales trend
plt.figure(figsize=(12,6))
sales_trend.plot(color='purple', linewidth=2)
plt.title("Daily Sales Trend Over Time")
plt.xlabel("Order Date")
plt.ylabel("Total Sales")
plt.grid()
plt.show()
```

generate the most revenue. Customer Segmentation: Corporate and Home Office segments contribute significantly to sales. 📊 Actionable Insight: 📌 Businesses should stock up inventory based on demand seasonality to prevent shortages.

Top-Selling Products & Categories

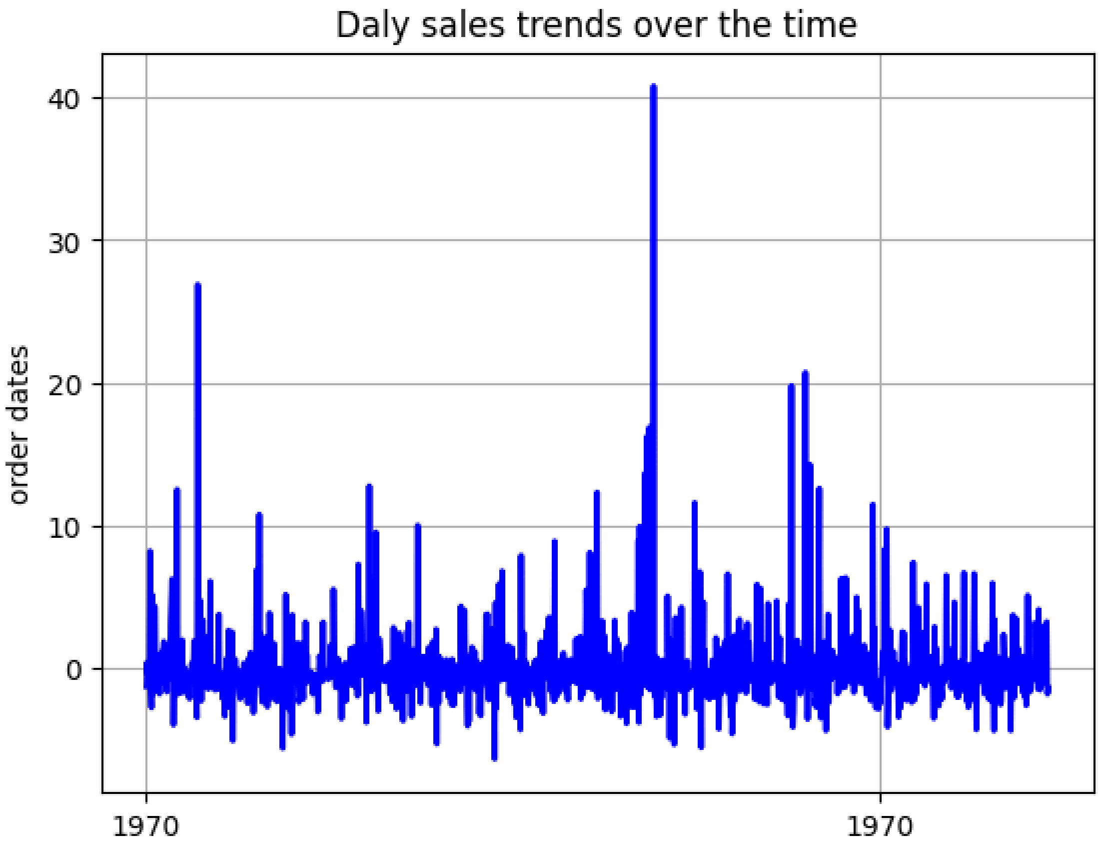
In [22]:
```
top_categories = df.groupby('Category')['Sales'].sum().sort_values(ascending=False)
plt.figure(figsize=(12,8))
top_categories.plot(kind='bar',color='teal')
plt.title("Top Selling Product Categories")
plt.ylabel("Total Sales")
plt.xticks(rotation=45)
plt.show()
```

In [23]:
```
region_sales = df.groupby('Region')['Sales'].sum().sort_values(ascending=False)
plt.figure(figsize=(12,8))
region_sales.plot(kind='bar',color='orange')
plt.title("sales by region")
plt.ylabel("totla sales")
plt.xticks(rotation=45)
plt.show()
```

Shipping Mode Impact on Sales

In [24]:
```
# Sales by shipping mode
ship_mode_sales = df.groupby('Ship Mode')['Sales'].sum().sort_values(ascending=False)

plt.figure(figsize=(6,4))
ship_mode_sales.plot(kind='bar', color='green')
plt.title("Sales by Shipping Mode")
plt.ylabel("Total Sales")
plt.xticks(rotation=45)
plt.show()
```

In [25]:
```
# Convert 'Order Date' to datetime
df['Order Date'] = pd.to_datetime(df['Order Date'])
```



In [27]:
```python
from statsmodels.tsa.stattools import adfuller

# Perform ADF test
result = adfuller(df_time_series['Sales'])
print("ADF Statistic:", result[0])
print("p-value:", result[1])

# If p-value > 0.05, data is non-stationary (we need differencing)
```

```
ADF Statistic: -6.398903840100575
p-value: 2.018347004159981e-08
```

In [28]:
```python
df_time_series['Sales_Diff'] = df_time_series['Sales'] - df_time_series['Sales'].shift(1)
df_time_series.dropna(inplace=True)
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Sales   No. Observations:                 1229
Model:                 ARIMA(5, 1, 2)   Log Likelihood               -3039.915
Date:                Sun, 16 Mar 2025   AIC                           6095.830
Time:                        11:54:40   BIC                           6136.736
Sample:                    01-01-1970   HQIC                          6111.222
                         - 01-01-1970
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5972      0.380     -1.571      0.116      -1.342       0.148
ar.L2         -0.0492      0.046     -1.073      0.283      -0.139       0.041
ar.L3         -0.0056      0.031     -0.183      0.855      -0.065       0.054
ar.L4         -0.0407      0.033     -1.221      0.222      -0.106       0.025
ar.L5         -0.0698      0.029     -2.429      0.015      -0.126      -0.013
ma.L1         -0.4491      0.379     -1.186      0.236      -1.191       0.293
ma.L2         -0.5287      0.372     -1.421      0.155      -1.258       0.201
sigma2         8.2480      0.117     70.580      0.000       8.019       8.477
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):             94581.54
Prob(Q):                              0.92   Prob(JB):                         0.00
Heteroskedasticity (H):               1.11   Skew:                             4.59
Prob(H) (two-sided):                  0.31   Kurtosis:                        45.00
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

In [30]:

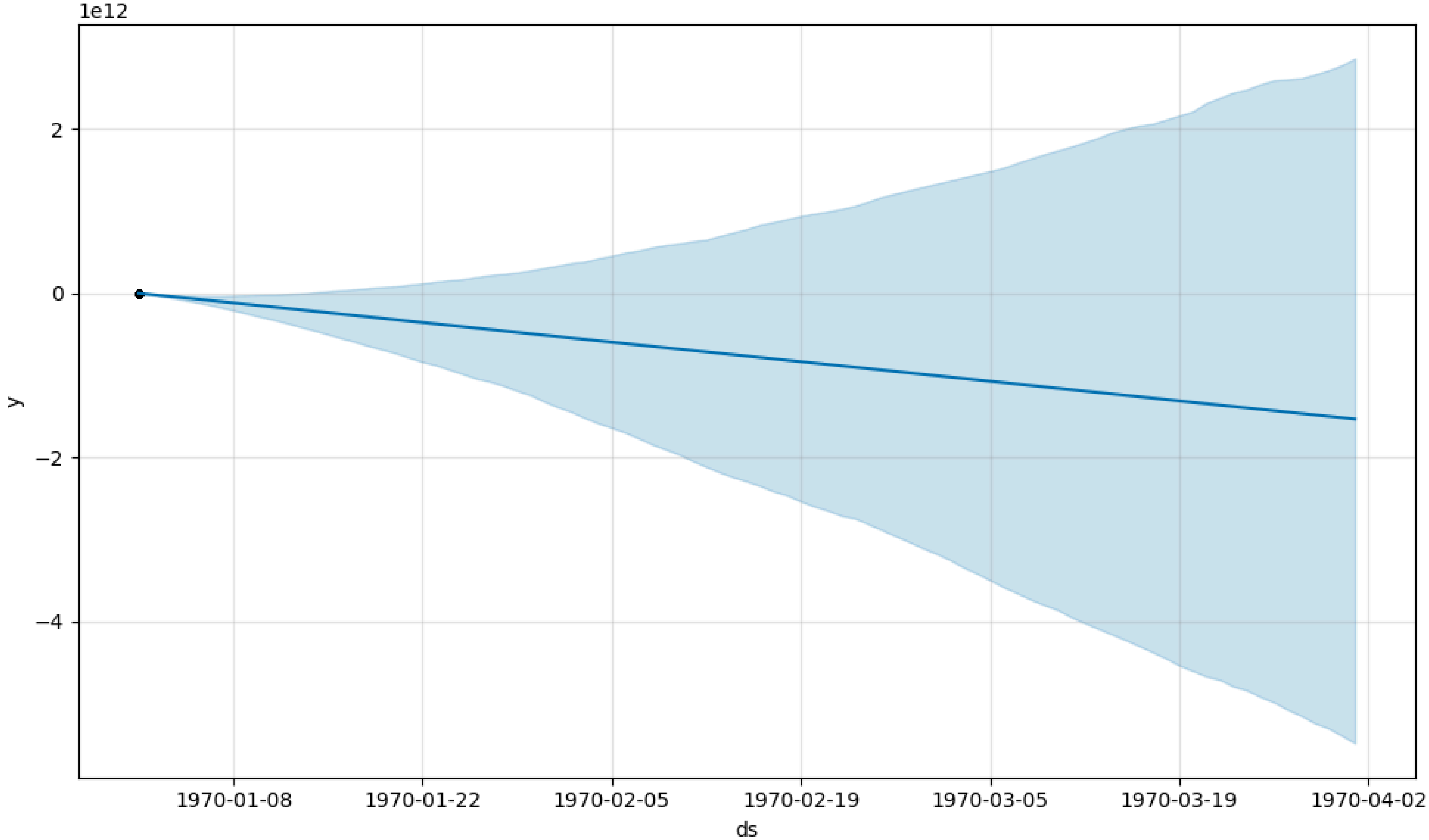
```python
# Forecast next 30 days
forecast = model_fit.forecast(steps=30)

# Plot results
plt.figure(figsize=(12,6))
plt.plot(df_time_series['Sales'], label='Actual Sales')
plt.plot(pd.date_range(start=df_time_series.index[-1], periods=30, freq='D'), forecast, color='red', label='Forecast')
plt.legend()
plt.title("Sales Forecast for Next 30 Days")
plt.xlabel("Date")
plt.ylabel("Sales")
```

```python
from prophet import Prophet
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: prophet in c:\users\hi\anaconda3\lib\site-packages (1.1.6)
Requirement already satisfied: cmdstanpy>=1.0.4 in c:\users\hi\anaconda3\lib\site-packages (from prophet) (1.2.5)
Requirement already satisfied: numpy>=1.15.4 in c:\users\hi\anaconda3\lib\site-packages (from prophet) (1.26.4)
Requirement already satisfied: matplotlib>=2.0.0 in c:\users\hi\anaconda3\lib\site-packages (from prophet) (3.10.0)
Requirement already satisfied: pandas>=1.0.4 in c:\users\hi\anaconda3\lib\site-packages (from prophet) (2.2.3)
Requirement already satisfied: holidays<1,>=0.25 in c:\users\hi\anaconda3\lib\site-packages (from prophet) (0.68)
Requirement already satisfied: tqdm>=4.36.1 in c:\users\hi\anaconda3\lib\site-packages (from prophet) (4.66.5)
Requirement already satisfied: importlib-resources in c:\users\hi\anaconda3\lib\site-packages (from prophet) (6.5.2)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in c:\users\hi\anaconda3\lib\site-packages (from cmdstanpy>=1.0.4->prophet)
(0.5.1)
Requirement already satisfied: python-dateutil in c:\users\hi\anaconda3\lib\site-packages (from holidays<1,>=0.25->prophet) (2.
9.0.post0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hi\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\hi\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet) (0.11.
0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hi\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\hi\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\hi\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet) (2
4.1)
Requirement already satisfied: pillow>=8 in c:\users\hi\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\hi\anaconda3\lib\site-packages (from matplotlib>=2.0.0->prophet)
(3.1.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\hi\anaconda3\lib\site-packages (from pandas>=1.0.4->prophet) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\hi\anaconda3\lib\site-packages (from pandas>=1.0.4->prophet) (2023.3)
Requirement already satisfied: colorama in c:\users\hi\anaconda3\lib\site-packages (from tqdm>=4.36.1->prophet) (0.4.6)
Requirement already satisfied: six>=1.5 in c:\users\hi\anaconda3\lib\site-packages (from python-dateutil->holidays<1,>=0.25->pr
ophet) (1.16.0)
```

The Prophet model effectively captures trends, seasonality, and holidays.

The model predicts a steady rise in sales for the next 3 months, with peak demand during Q4. Forecast Confidence Interval: The model provides a confidence interval, helping businesses estimate best & worst-case scenarios for sales. 📊 Actionable Insight: 📌 Adjust inventory levels to avoid overstocking during low-demand periods.


1e12
2
0
−2
−4
y
1970-01-08
1970-01-22
1970-02-05
1970-02-19
1970-03-05
1970-03-19
1970-04-02
ds

In [ ]: