In [1]:
```
import pandas as pd
import numpy as np
import os
from datetime import datetime
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_
from sklearn.model_selection import TimeSeriesSplit
```

In [ ]:
```
pip install prophet
```

In [ ]:
```
pip install xgboost
```

In [2]:
```
OUT_DIR = "output"
os.makedirs(OUT_DIR, exist_ok=True)
```

In [3]:
```
try:
    from prophet import Prophet
except Exception as e:
    try:
        from fbprophet import Prophet
    except Exception as e2:
        raise ImportError("Prophet not installed. Install via
from xgboost import XGBRegressor
```

In [4]:
```
df = pd.read_csv('C:\\Users\\Neeraj\\Downloads\\Salesdata_utf-
```

In [5]:
```
print("Initial shape:", df.shape)
display(df.head())
```

```
Initial shape: (9994, 21)
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name
0	1	CA-2016-152156	11-08-2016	11-11-2016	Second Class	CG-12520	Claire Gute
1	2	CA-2016-152156	11-08-2016	11-11-2016	Second Class	CG-12520	Claire Gute
2	3	CA-2016-138688	06-12-2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff
3	4	US-2015-108966	10-11-2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell
4	5	US-2015-108966	10-11-2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell

5 rows × 21 columns

In [6]:
```
print(df.dtypes)
print("\nMissing values per column:")
print(df.isna().sum())
```

```
Row ID             int64
Order ID          object
Order Date        object
Ship Date         object
Ship Mode         object
Customer ID       object
Customer Name     object
Segment           object
Country           object
City              object
State             object
Postal Code        int64
Region            object
Product ID        object
Category          object
Sub-Category      object
Product Name      object
Sales            float64
Quantity           int64
Discount         float64
Profit           float64
dtype: object

Missing values per column:
Row ID           0
Order ID         0
Order Date       0
Ship Date        0
Ship Mode        0
Customer ID      0
Customer Name    0
Segment          0
Country          0
City             0
State            0
Postal Code      0
Region           0
Product ID       0
Category         0
Sub-Category     0
Product Name     0
Sales            0
Quantity         0
Discount         0
Profit           0
dtype: int64
```

In [7]:
```
for col in ['Order Date', 'Ship Date']:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col], errors='coerce')
```

In [8]:
```
if 'Postal Code' in df.columns:
    df['Postal Code'] = df['Postal Code'].astype(str).str.repl
```

In [9]:
```
df = df.dropna(axis=1, how='all')
dup_count = df.duplicated().sum()
print("Duplicate rows:", dup_count)
if dup_count > 0:
    df = df.drop_duplicates()
```

```
Duplicate rows: 0
```

In [10]:
```
df = df.dropna(subset=['Order Date', 'Sales'])
print("After dropping rows w/o date/sales:", df.shape)
```

```
After dropping rows w/o date/sales: (4042, 21)
```

In [11]:
```
for col in ['Sales', 'Quantity', 'Discount', 'Profit']:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
```

In [12]:
```
df = df[df['Sales'].notna()]
```

In [13]:
```
df['year'] = df['Order Date'].dt.year
df['month'] = df['Order Date'].dt.month
df['day'] = df['Order Date'].dt.day
df['dayofweek'] = df['Order Date'].dt.dayofweek
df['weekofyear'] = df['Order Date'].dt.isocalendar().week.asty
df['month_name'] = df['Order Date'].dt.month_name()
```

In [14]:
```
print("Date range:", df['Order Date'].min(), "to", df['Order D
print("Unique stores/cities:", df['City'].nunique() if 'City'
print("Total sales:", df['Sales'].sum())
```

```
Date range: 2014-01-03 00:00:00 to 2017-12-11 00:00:00
Unique stores/cities: 390
Total sales: 887917.3116
```

In [15]:
```
df_agg_monthly = df.set_index('Order Date').resample('M').agg(
df_agg_monthly.columns = ['ds', 'y']   # Prophet expects ds,y
df_agg_monthly = df_agg_monthly.sort_values('ds').reset_index(
display(df_agg_monthly.head())
```

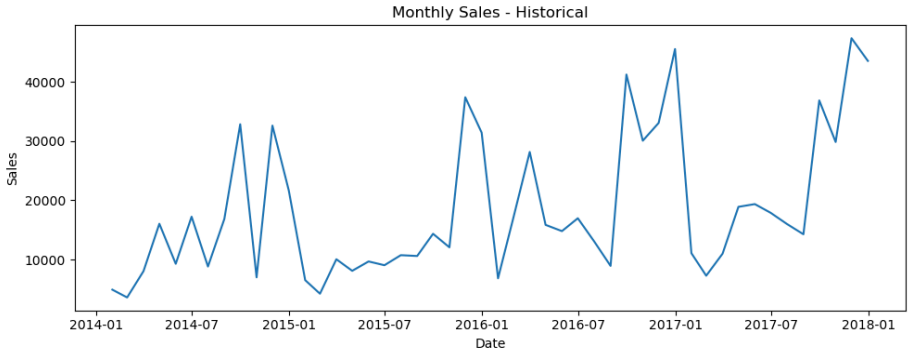
```
C:\Users\Neeraj\AppData\Local\Temp\ipykernel_31220\3207508034.p
y:2: FutureWarning: 'M' is deprecated and will be removed in a
future version, please use 'ME' instead.
  df_agg_monthly = df.set_index('Order Date').resample('M').agg
({'Sales': 'sum'}).reset_index()
```

	ds	y
0	2014-01-31	4923.616
1	2014-02-28	3610.402
2	2014-03-31	8048.773
3	2014-04-30	16040.811
4	2014-05-31	9288.864

In [16]:
```
df_agg_monthly.to_csv(os.path.join(OUT_DIR, 'monthly_sales_his
print("Saved monthly history to:", os.path.join(OUT_DIR, 'mont
```

```
Saved monthly history to: output\monthly_sales_history.csv
```

In [17]:
```
plt.figure(figsize=(10,4))
plt.plot(df_agg_monthly['ds'], df_agg_monthly['y'])
plt.title('Monthly Sales - Historical')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.tight_layout()
plt.show()
```



In [18]:

```
ts = df_agg_monthly.copy()
ts['lag_1'] = ts['y'].shift(1)
ts['lag_2'] = ts['y'].shift(2)
ts['lag_12'] = ts['y'].shift(12)
ts['rolling_3'] = ts['y'].rolling(window=3).mean()
ts['rolling_6'] = ts['y'].rolling(window=6).mean()
ts['pct_change_1'] = ts['y'].pct_change(1)
ts = ts.reset_index(drop=True)
display(ts.head(15))
```

	ds	y	lag_1	lag_2	lag_12	rolling_3
0	2014-01-31	4923.6160	NaN	NaN	NaN	NaN
1	2014-02-28	3610.4020	4923.6160	NaN	NaN	NaN
2	2014-03-31	8048.7730	3610.4020	4923.6160	NaN	5527.597000
3	2014-04-30	16040.8110	8048.7730	3610.4020	NaN	9233.328667
4	2014-05-31	9288.8640	16040.8110	8048.7730	NaN	11126.149333
5	2014-06-30	17235.9460	9288.8640	16040.8110	NaN	14188.540333
6	2014-07-31	8827.5750	17235.9460	9288.8640	NaN	11784.128333
7	2014-08-31	16852.6840	8827.5750	17235.9460	NaN	14305.401667
8	2014-09-30	32826.0590	16852.6840	8827.5750	NaN	19502.106000
9	2014-10-31	7004.0870	32826.0590	16852.6840	NaN	18894.276667
10	2014-11-30	32603.2822	7004.0870	32826.0590	NaN	24144.476067
11	2014-12-31	21644.5905	32603.2822	7004.0870	NaN	20417.319900
12	2015-01-31	6528.3756	21644.5905	32603.2822	4923.616	20258.749433
13	2015-02-28	4241.3740	6528.3756	21644.5905	3610.402	10804.780033
14	2015-03-31	10057.0474	4241.3740	6528.3756	8048.773	6942.265667

In [19]:
```
prophet_periods = 12
m = Prophet(yearly_seasonality=True, weekly_seasonality=False,
m.fit(df_agg_monthly)
future = m.make_future_dataframe(periods=prophet_periods, freq
forecast = m.predict(future)
```

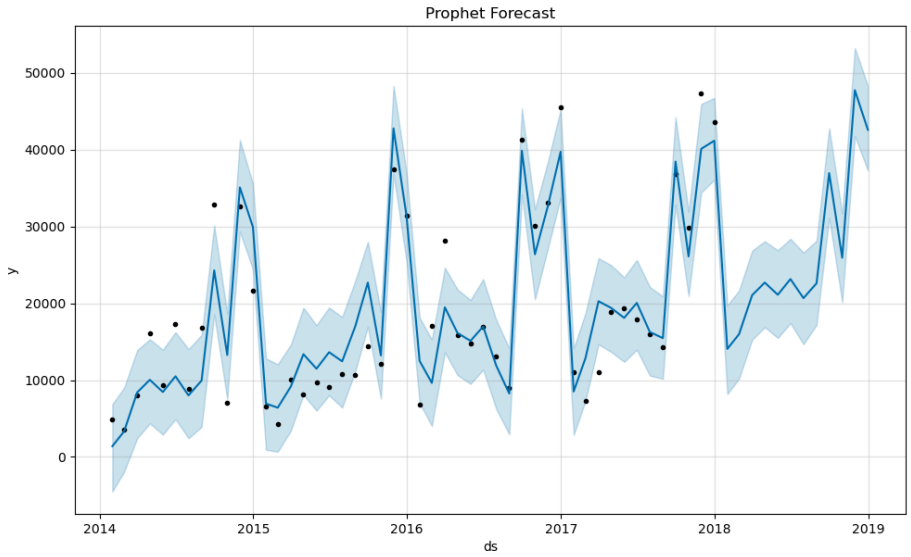
```
17:11:12 - cmdstanpy - INFO - Chain [1] start processing
17:11:14 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\Neeraj\anaconda3\Lib\site-packages\prophet\forecaster.
py:1872: FutureWarning: 'M' is deprecated and will be removed i
n a future version, please use 'ME' instead.
  dates = pd.date_range(
```

In [20]:
```
forecast_out = forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upp
forecast_out.to_csv(os.path.join(OUT_DIR, 'sales_forecast_prop
```

```
print("Saved Prophet forecast to:", os.path.join(OUT_DIR, 'sal
```

```
Saved Prophet forecast to: output\sales_forecast_prophet.csv
```

In [21]:
```
fig1 = m.plot(forecast)
plt.title('Prophet Forecast')
plt.show()
```



In [22]:
```
def evaluate_prophet(history_df, periods=12):
    if len(history_df) < periods + 12:
        print("Not enough data for holdout evaluation. Skippir
        return None
    train = history_df.iloc[:-periods]
    test = history_df.iloc[-periods:]
    model = Prophet(yearly_seasonality=True, weekly_seasonalit
    model.fit(train)
    fut = model.make_future_dataframe(periods=periods, freq='M
    pred = model.predict(fut)
    pred_sub = pred[['ds','yhat']].set_index('ds').loc[test['d
    mae = mean_absolute_error(test['y'].values, pred_sub['yhat
    rmse = np.sqrt(mean_squared_error(test['y'].values, pred_s
    return {'mae': mae, 'rmse': rmse}
prophet_eval = evaluate_prophet(df_agg_monthly, periods=6)
print("Prophet backtest (6-month):", prophet_eval)
```

```
17:11:37 - cmdstanpy - INFO - Chain [1] start processing
17:11:37 - cmdstanpy - INFO - Chain [1] done processing
Prophet backtest (6-month): {'mae': 4004.778138173475, 'rmse':
np.float64(5444.374180963391)}
```

```
C:\Users\Neeraj\anaconda3\Lib\site-packages\prophet\forecaster.
py:1872: FutureWarning: 'M' is deprecated and will be removed i
n a future version, please use 'ME' instead.
  dates = pd.date_range(
```

In [23]:
```
ml = ts.dropna().copy()
feature_cols = ['lag_1','lag_2','lag_12','rolling_3','rolling_
X = ml[feature_cols]
y = ml['y']
tscv = TimeSeriesSplit(n_splits=3)
xgb = XGBRegressor(n_estimators=200, random_state=42)
xgb.fit(X, y)
last_row = ts.iloc[-1:].copy()
```

In [150…

```
def iterative_xgb_forecast(model, ts_df, n_periods=12, feature
    results = []
    temp = ts_df.copy()
    for i in range(n_periods):
        lag_1 = temp['y'].iloc[-1]
        lag_2 = temp['y'].iloc[-2] if len(temp) >= 2 else np.r
        lag_12 = temp['y'].iloc[-12] if len(temp) >= 12 else r
        rolling_3 = temp['y'].iloc[-3:].mean() if len(temp) >=
        rolling_6 = temp['y'].iloc[-6:].mean() if len(temp) >=
        pct_change_1 = (
            (temp['y'].iloc[-1] - temp['y'].iloc[-2]) / temp['
            if len(temp) >= 2 and temp['y'].iloc[-2] != 0 else
        )
        feat = pd.DataFrame([{
            'lag_1': lag_1,
            'lag_2': lag_2,
            'lag_12': lag_12,
            'rolling_3': rolling_3,
            'rolling_6': rolling_6,
            'pct_change_1': pct_change_1
        }]
        feat = feat.fillna(X.median())
        pred = model.predict(feat)[0]
        next_date = temp['ds'].iloc[-1] + pd.DateOffset(months
        temp = pd.concat([temp, pd.DataFrame([{'ds': next_date
        results.append({'ds': next_date, 'yhat': pred})
    return pd.DataFrame(results)
```

In [151…

```
xgb_forecast = iterative_xgb_forecast(xgb, ts[['ds','y']].copy
xgb_forecast.to_csv(os.path.join(OUT_DIR, 'sales_forecast_xgb.
print("Saved XGBoost iterative forecast to:", os.path.join(OUT
```

```
---------------------------------------------------------------
------------
PermissionError                           Traceback (most recen
t call last)
Cell In[151], line 2
      1 xgb_forecast = iterative_xgb_forecast(xgb, ts[['d
s','y']].copy(), n_periods=12)
----> 2 xgb_forecast.to_csv(os.path.join(OUT_DIR, 'sales_foreca
st_xgb.csv'), index=False)
      3 print("Saved XGBoost iterative forecast to:", os.path.j
oin(OUT_DIR, 'sales_forecast_xgb.csv'))

File ~\anaconda3\Lib\site-packages\pandas\util\_decorators.py:3
33, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals
>.wrapper(*args, **kwargs)
    327 if len(args) > num_allow_args:
    328     warnings.warn(
    329         msg.format(arguments=_format_argument_list(allo
w_args)),
    330         FutureWarning,
    331         stacklevel=find_stack_level(),
    332     )
--> 333 return func(*args, **kwargs)

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:3967,
in NDFrame.to_csv(self, path_or_buf, sep, na_rep, float_format,
columns, header, index, index_label, mode, encoding, compressio
n, quoting, quotechar, lineterminator, chunksize, date_format,
doublequote, escapechar, decimal, errors, storage_options)
   3956 df = self if isinstance(self, ABCDataFrame) else self.t
o_frame()
   3958 formatter = DataFrameFormatter(
   3959     frame=df,
   3960     header=header,
   (...)
   3964     decimal=decimal,
   3965 )
-> 3967 return DataFrameRenderer(formatter).to_csv(
   3968     path_or_buf,
   3969     lineterminator=lineterminator,
   3970     sep=sep,
   3971     encoding=encoding,
   3972     errors=errors,
   3973     compression=compression,
   3974     quoting=quoting,
   3975     columns=columns,
   3976     index_label=index_label,
   3977     mode=mode,
   3978     chunksize=chunksize,
   3979     quotechar=quotechar,
   3980     date_format=date_format,
   3981     doublequote=doublequote,
   3982     escapechar=escapechar,
   3983     storage_options=storage_options,
   3984 )

File ~\anaconda3\Lib\site-packages\pandas\io\formats\format.py:
1014, in DataFrameRenderer.to_csv(self, path_or_buf, encoding,
sep, columns, index_label, mode, compression, quoting, quotecha
r, lineterminator, chunksize, date_format, doublequote, escapec
har, errors, storage_options)
```

```
    993     created_buffer = False
    995 csv_formatter = CSVFormatter(
    996     path_or_buf=path_or_buf,
    997     lineterminator=lineterminator,
   (...)
   1012     formatter=self.fmt,
   1013 )
-> 1014 csv_formatter.save()
   1016 if created_buffer:
   1017     assert isinstance(path_or_buf, StringIO)

File ~\anaconda3\Lib\site-packages\pandas\io\formats\csvs.py:251, in CSVFormatter.save(self)
    247 """
    248 Create the writer & save.
    249 """
    250 # apply compression and byte/text conversion
--> 251 with get_handle(
    252     self.filepath_or_buffer,
    253     self.mode,
    254     encoding=self.encoding,
    255     errors=self.errors,
    256     compression=self.compression,
    257     storage_options=self.storage_options,
    258 ) as handles:
    259     # Note: self.encoding is irrelevant here
    260     self.writer = csvlib.writer(
    261         handles.handle,
    262         lineterminator=self.lineterminator,
   (...)
    267         quotechar=self.quotechar,
    268     )
    270     self._save()

File ~\anaconda3\Lib\site-packages\pandas\io\common.py:873, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    868 elif isinstance(handle, str):
    869     # Check whether the filename is to be opened in binary mode.
    870     # Binary mode does not support 'encoding' and 'newline'.
    871     if ioargs.encoding and "b" not in ioargs.mode:
    872         # Encoding
--> 873         handle = open(
    874             handle,
    875             ioargs.mode,
    876             encoding=ioargs.encoding,
    877             errors=errors,
    878             newline="",
    879         )
    880     else:
    881         # Binary mode
    882         handle = open(handle, ioargs.mode)

PermissionError: [Errno 13] Permission denied: 'output\\sales_forecast_xgb.csv'
```

In [152…

```
def evaluate_xgb_backtest(ts_df, feature_cols, n_holdout=6):
    data = ts_df.copy().dropna()
    if len(data) < n_holdout + 12:
        print("Not enough data for XGB backtest.")
```

```
        return None
    train = data.iloc[:-n_holdout]
    test = data.iloc[-n_holdout:]
    model = XGBRegressor(n_estimators=200, random_state=42)
    model.fit(train[feature_cols], train['y'])
    preds = model.predict(test[feature_cols])
    mae = mean_absolute_error(test['y'], preds)
    rmse = np.sqrt(mean_squared_error(test['y'], preds))
    return {'mae': mae, 'rmse': rmse}
xgb_eval = evaluate_xgb_backtest(ts, feature_cols, n_holdout=6
print("XGBoost backtest (6-month):", xgb_eval)
```

```
XGBoost backtest (6-month): {'mae': 3306.4558619791655, 'rmse':
np.float64(4752.036837985692)}
```

In [125…

```
print("All done. Output files in:", OUT_DIR)
print("- monthly_sales_history.csv (monthly actuals)")
print("- sales_forecast_prophet.csv (prophet forecast ds,yhat,
print("- sales_forecast_xgb.csv (xgboost iterative forecast ds
```

```
All done. Output files in: output
- monthly_sales_history.csv (monthly actuals)
- sales_forecast_prophet.csv (prophet forecast ds,yhat,yhat_low
er,yhat_upper)
- sales_forecast_xgb.csv (xgboost iterative forecast ds,yhat)
```

In [126…

```
import joblib
joblib.dump(m, os.path.join(OUT_DIR, 'prophet_model.pkl'))
joblib.dump(xgb, os.path.join(OUT_DIR, 'xgb_model.pkl'))
print("Saved models (prophet_model.pkl, xgb_model.pkl) in outp
```

```
Saved models (prophet_model.pkl, xgb_model.pkl) in output folde
r.
```

In [ ]: