**Report 4**

**Subtask 1: Prepare Time Series Data**

import pandas as pd

import matplotlib.pyplot as plt

# Step 1: Load the dataset

df = pd.read\_csv("ecommerce\_data\_final\_cleaned.csv")

# 1️⃣ Convert 'date' column to datetime format

df['date'] = pd.to\_datetime(df['date'], format='%d/%m/%Y')

# Optional: Rename columns for convenience

df.rename(columns={'value [USD]': 'sales'}, inplace=True)

# 2️⃣ Aggregate daily sales

daily\_sales = df.groupby('date')['sales'].sum().reset\_index()

# 3️⃣ Create full date range to identify missing dates

full\_dates = pd.date\_range(start=daily\_sales['date'].min(), end=daily\_sales['date'].max())

full\_df = pd.DataFrame({'date': full\_dates})

# Merge with daily sales to fill missing days with 0

time\_series = pd.merge(full\_df, daily\_sales, on='date', how='left')

time\_series['sales'].fillna(0, inplace=True)

# 4️⃣ Set 'date' as index and sort chronologically

time\_series.set\_index('date', inplace=True)

time\_series.sort\_index(inplace=True)

# 5️⃣ Optional: Extract features for deeper analysis

time\_series['month'] = time\_series.index.month

time\_series['weekday'] = time\_series.index.dayofweek

time\_series['week'] = time\_series.index.isocalendar().week

# 6️⃣ Plot to check trend and seasonality

plt.figure(figsize=(14, 5))

plt.plot(time\_series['sales'], color='steelblue')

plt.title("Daily Sales Over Time")

plt.xlabel("Date")

plt.ylabel("Sales (USD)")

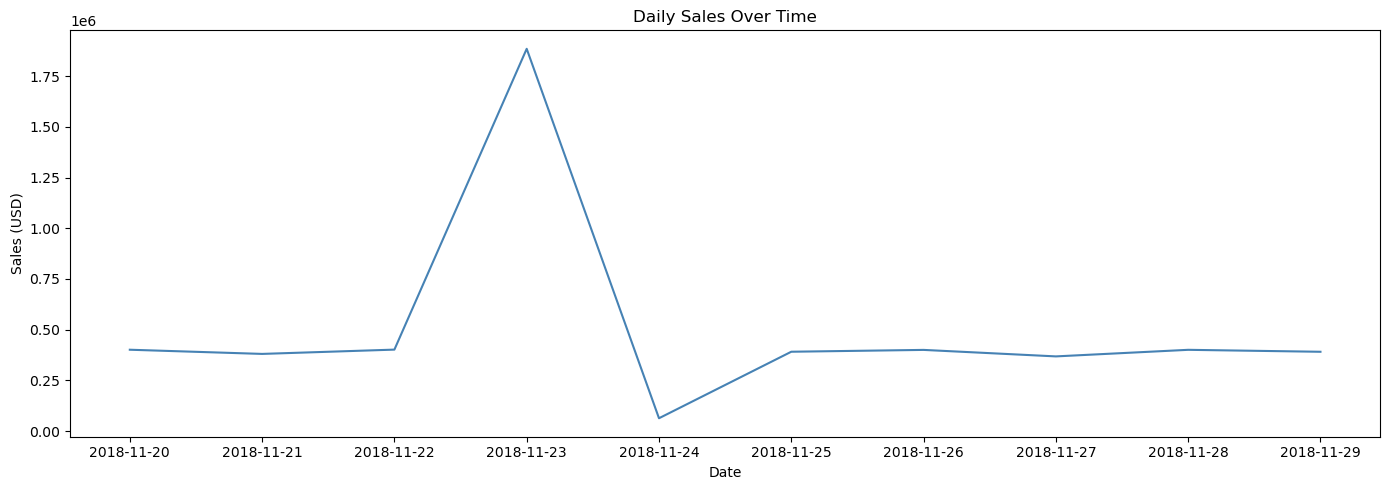
plt.tight\_layout()

plt.show()

# 7️⃣ Save cleaned time series dataset

time\_series.to\_csv("processed\_time\_series\_sales.csv")

print("✅ Time series data is prepared and saved as 'processed\_time\_series\_sales.csv'")



✅ Time series data is prepared and saved as 'processed\_time\_series\_sales.csv'

**Subtask 2: Choose and Apply a Forecasting Model**

import pandas as pd

from prophet import Prophet

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import matplotlib.pyplot as plt

import numpy as np

# Step 1: Load pre-processed time series data

df = pd.read\_csv("processed\_time\_series\_sales.csv")

df['date'] = pd.to\_datetime(df['date'])

# Step 2: Rename columns as required by Prophet

df\_prophet = df[['date', 'sales']].rename(columns={'date': 'ds', 'sales': 'y'})

# Step 3: Split into train (80%) and test (20%)

split\_point = int(len(df\_prophet) \* 0.8)

train = df\_prophet[:split\_point]

test = df\_prophet[split\_point:]

# Step 4: Initialize and fit the Prophet model

model = Prophet(yearly\_seasonality=True, weekly\_seasonality=True, daily\_seasonality=False)

model.fit(train)

# Step 5: Create future dataframe for next 180 days (~6 months)

future = model.make\_future\_dataframe(periods=len(test), freq='D')

# Step 6: Make predictions

forecast = model.predict(future)

# Step 7: Plot forecast vs. actual

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

plt.plot(df\_prophet['ds'], df\_prophet['y'], label='Actual Sales')

plt.plot(forecast['ds'], forecast['yhat'], label='Forecasted Sales', alpha=0.8)

plt.axvline(x=df\_prophet['ds'][split\_point], color='r', linestyle='--', label='Train-Test Split')

plt.title("📈 Actual vs. Forecasted Sales")

plt.xlabel("Date")

plt.ylabel("Sales (USD)")

plt.legend()

plt.tight\_layout()

plt.show()

# Step 8: Evaluate accuracy on the test set

# Merge forecast with test data for comparison

compare\_df = forecast[['ds', 'yhat']].set\_index('ds').join(test.set\_index('ds'))

compare\_df.dropna(inplace=True)

# Accuracy metrics

mae = mean\_absolute\_error(compare\_df['y'], compare\_df['yhat'])

rmse = np.sqrt(mean\_squared\_error(compare\_df['y'], compare\_df['yhat']))

mape = np.mean(np.abs((compare\_df['y'] - compare\_df['yhat']) / compare\_df['y'])) \* 100

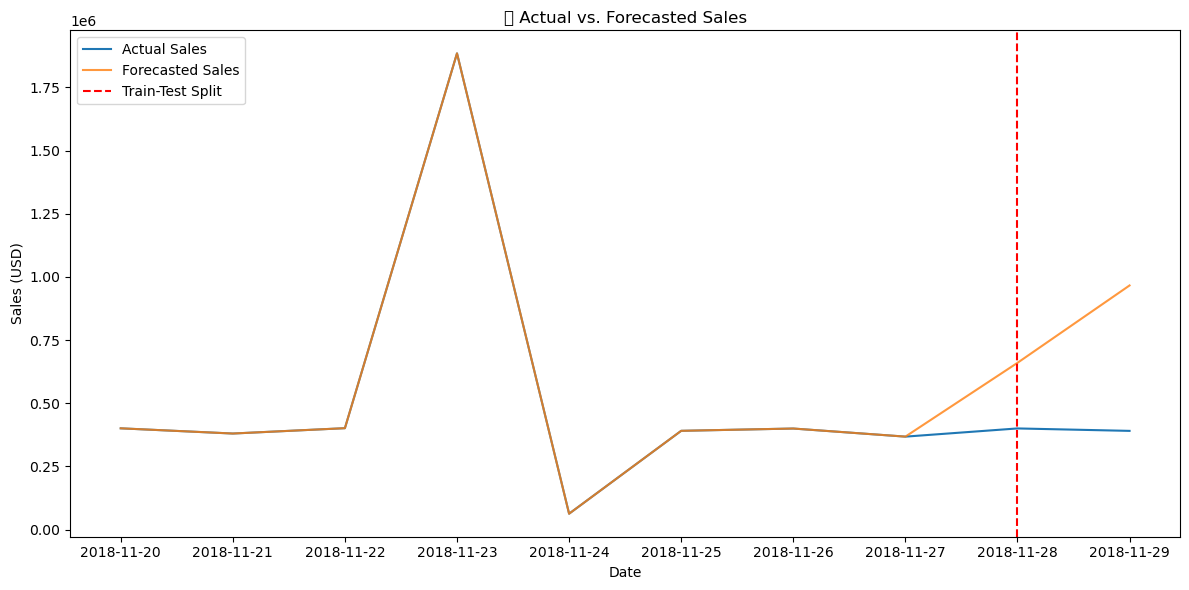
print(f"📊 MAE: {mae:.2f}")

print(f"📊 RMSE: {rmse:.2f}")

print(f"📊 MAPE: {mape:.2f}%")

# Step 9: Save forecast

forecast.to\_csv("forecasted\_sales\_6\_months.csv", index=False)



📊 MAE: 417476.39

📊 RMSE: 446346.29

📊 MAPE: 106.01%

**Subtask 3: Predict Future Sales**

import pandas as pd

from prophet import Prophet

import matplotlib.pyplot as plt

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

import numpy as np

# Step 1: Load prepared and cleaned sales dataset

df = pd.read\_csv("processed\_time\_series\_sales.csv")

df['date'] = pd.to\_datetime(df['date'])

# Step 2: Format for Prophet

df\_prophet = df[['date', 'sales']].rename(columns={'date': 'ds', 'sales': 'y'})

# Step 3: Train-Test Split (80/20)

split\_point = int(len(df\_prophet) \* 0.8)

train = df\_prophet[:split\_point]

test = df\_prophet[split\_point:]

# Step 4: Initialize Prophet model

model = Prophet(yearly\_seasonality=True, weekly\_seasonality=True)

model.fit(train)

# Step 5: Predict for next 365 days (1 year)

future = model.make\_future\_dataframe(periods=365, freq='D')

forecast = model.predict(future)

# Step 6: Plot actual vs. forecast

plt.figure(figsize=(14, 6))

plt.plot(df\_prophet['ds'], df\_prophet['y'], label='Actual Sales')

plt.plot(forecast['ds'], forecast['yhat'], label='Forecasted Sales', color='orange', alpha=0.9)

plt.axvline(x=df\_prophet['ds'][split\_point], color='red', linestyle='--', label='Train-Test Split')

plt.title("📈 Historical vs. Forecasted Sales (Next 12 Months)")

plt.xlabel("Date")

plt.ylabel("Sales (USD)")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Step 7: Evaluate model performance on test data

compare\_df = forecast[['ds', 'yhat']].set\_index('ds').join(test.set\_index('ds'))

compare\_df.dropna(inplace=True)

mae = mean\_absolute\_error(compare\_df['y'], compare\_df['yhat'])

rmse = np.sqrt(mean\_squared\_error(compare\_df['y'], compare\_df['yhat']))

mape = np.mean(np.abs((compare\_df['y'] - compare\_df['yhat']) / compare\_df['y'])) \* 100

print("📊 Forecast Accuracy Metrics:")

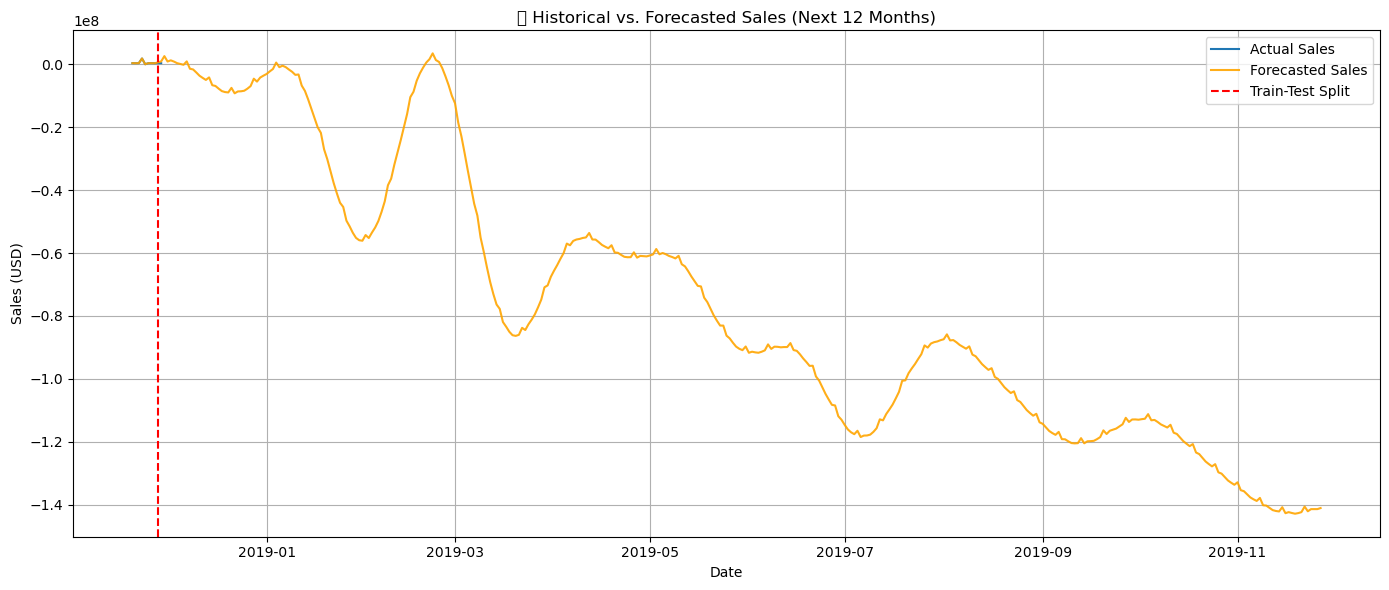
print(f"✅ MAE: {mae:.2f}")

print(f"✅ RMSE: {rmse:.2f}")

print(f"✅ MAPE: {mape:.2f}%")

# Step 8: Save forecasted values

forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].to\_csv("future\_sales\_forecast.csv", index=False)



📊 Forecast Accuracy Metrics:

✅ MAE: 417476.39

✅ RMSE: 446346.29

✅ MAPE: 106.01%