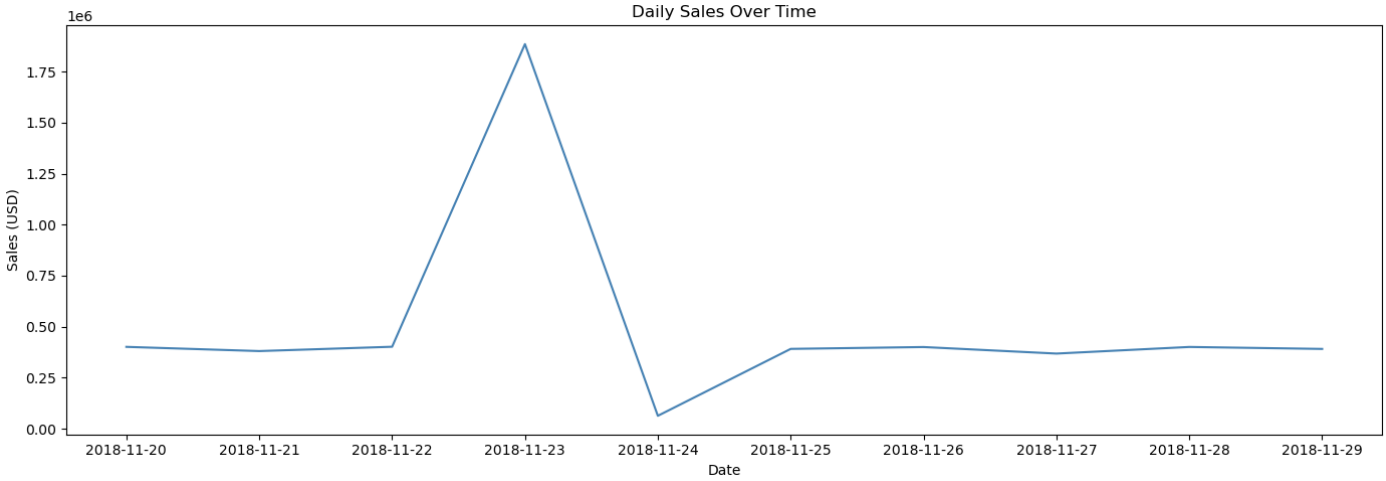
Report 4

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'")
```

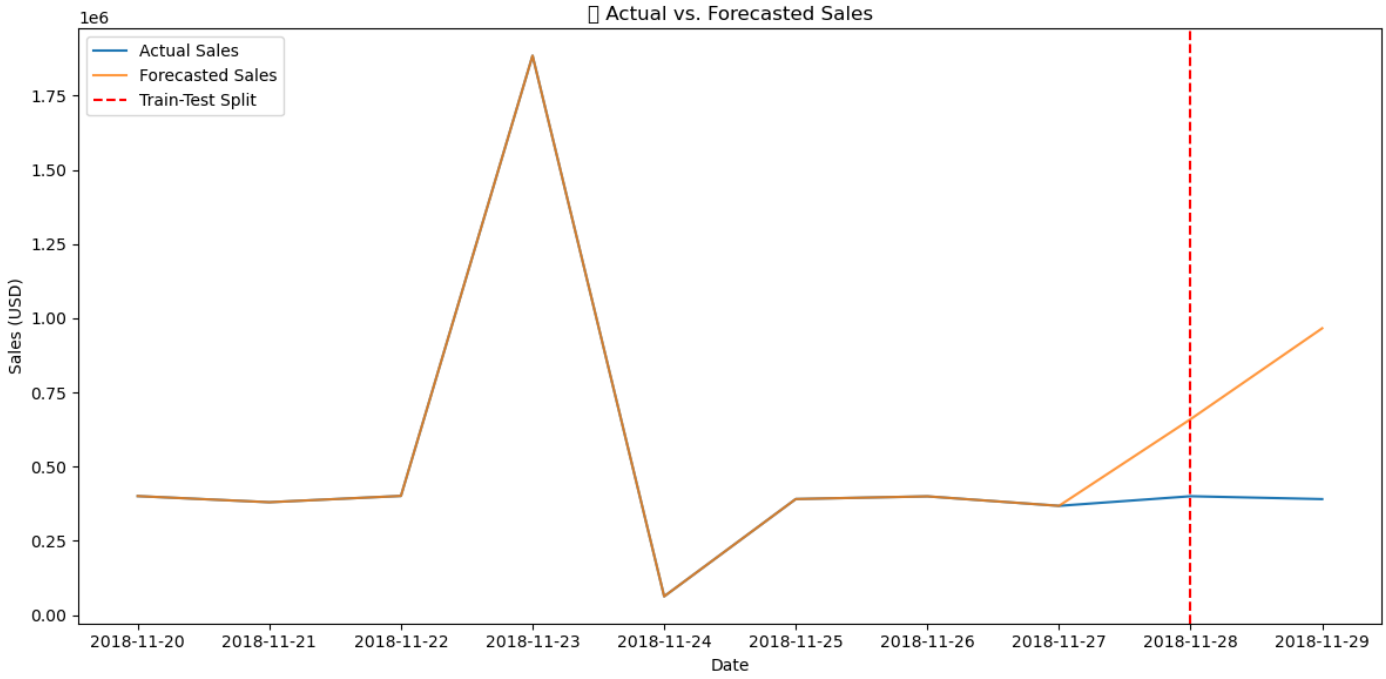


✅ Time series data is prepared and saved as 'processed_time_series_sales.csv'

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

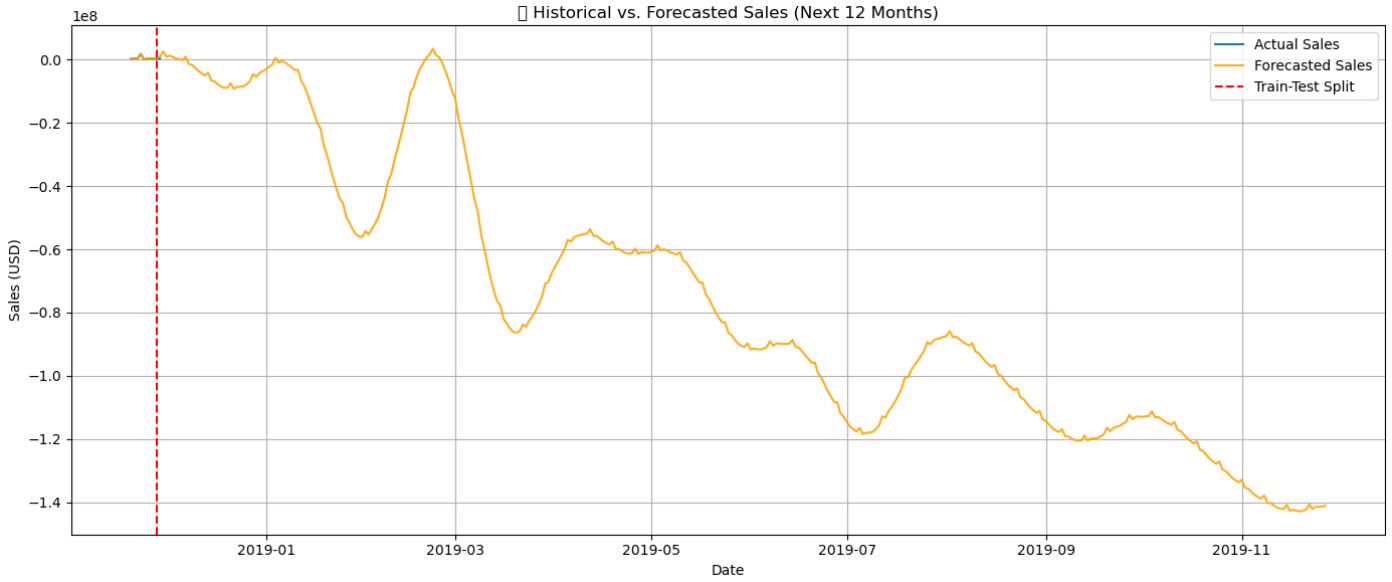
📊 RMSE: 446346.29

📊 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%