```
# 2. Complex DateTime Operations**
#Convert all dates to UTC datetime

df['OrderDate'] = pd.to_datetime(df['OrderDate'], errors='coerce', utc=True)
#Calculate order processing time (assume returns happen 7 days after order)
df['ReturnDate'] = df['OrderDate'] + pd.Timedelta(days=7)
#Find weekly sales trends for electronics vs home goods
df['Week'] = df['OrderDate'].dt.to_period('W')
weekly_sales = df.groupby(['Week',
'Category'])['Price'].sum().unstack().fillna(0)
#Identify customers with >2 orders in any 14-day window
df = df.sort_values(by='OrderDate')
df['OrderDate'] = df['OrderDate'].dt.date
customer_orders =
df.groupby('CustomerID')['OrderDate'].apply(list).reset_index()
customer_orders['OrderCount'] = customer_orders['OrderDate'].apply(lambda x:
len(x))
customer_orders['HasMoreThan2OrdersIn14Days'] =
customer_orders['OrderDate'].apply(
   lambda dates: any(len(list(g)) > 2 for k, g in groupby(dates, key=lambda
date: date - pd.Timedelta(days=date.weekday())) if len(list(g)) > 2)
)

#3. Advanced Collections & Optimization**
# Build nested dictionary: `{CustomerID: {"total_spent": X,
"favorite_category": Y}}`
customer_summary = df.groupby('CustomerID').agg(
   total_spent=('Price', 'sum'),
   favorite_category=('Category', lambda x: x.mode().iloc[0] if not
x.mode().empty else None)
).to_dict(orient='index')
#Use `defaultdict` to track return rates by region: `{"North": 0.25, ...}`
from collections import defaultdict

return_rates = defaultdict(lambda: {'returns': 0, 'total': 0})
for _, row in df.iterrows():
   return_rates[row['Region']]['total'] += 1
   if row['ReturnFlag'] == 1:
       return_rates[row['Region']]['returns'] += 1

return_rate_dict = {region: data['returns'] / data['total'] for region, data in
return_rates.items()}
#Find most common promo code sequence using `itertools` and `Counter`
from itertools import groupby
from collections import Counter

promo_sequences = df.groupby('CustomerID')['PromoCode'].apply(lambda x:
list(x.dropna())).tolist()
```

```
promo_sequence_counter = Counter(tuple(group) for sublist in promo_sequences
for key, group in groupby(sublist))
most_common_promo_sequence = promo_sequence_counter.most_common(1)
#Optimize memory usage by downcasting numerical columns
df['Quantity'] = pd.to_numeric(df['Quantity'], downcast='integer')
df['Price'] = pd.to_numeric(df['Price'], downcast='float')

#4. Bonus (If Time Permits)
#Create UDF to flag "suspicious orders" (multiple returns + high value)
def flag_suspicious_orders(row):
   return row['ReturnFlag'] > 1 and row['Price'] > 500

df['SuspiciousOrder'] = df.apply(flag_suspicious_orders, axis=1)
#Calculate rolling 7-day average sales using efficient windowing
df['OrderDate'] = pd.to_datetime(df['OrderDate'])
df = df.set_index('OrderDate')
rolling_avg_sales = df['Price'].rolling('7D').mean()
```