TASK -01 Time Series Analysis Research and Development Report

So to build a forecasting model for predicting the number of units sold for each item on Amazon, we can follow a structured approach, involving data preprocessing, feature engineering, model training, and evaluation.

Github: https://github.com/lakshmancodes/Anarix_task

1. Introduction :

Objective: The goal of this time series analysis is to forecast unit sales using historical sales data. Accurate forecasting helps in inventory management, sales planning, and optimizing supply chain operations.

Dataset: The dataset contains daily sales data with columns such as date, units sold, ad spend, and unit price. The primary target variable for this analysis is the number of units sold.

2. Time Series Analysis Overview

Definition: Time series analysis involves analyzing data points collected or recorded at specific time intervals. The key purpose is to identify patterns and make forecasts.

Components of Time Series:
Trend: Long-term movement in the data.
Seasonality: Regular patterns repeating at fixed intervals.
Cyclic: Long-term oscillations unrelated to seasonality.
Irregular: Random noise or variability in the data.

3. Exploratory Data Analysis (EDA)

Initial Data Inspection: Initial steps involved checking for missing values, data types, and basic statistics.

Visualization:

Line Plot: A line plot of units sold over time revealed clear seasonal patterns and trends.
Correlation Heatmap: A correlation heatmap helped understand the relationships between various features.

Insights:
The sales data showed seasonal patterns with peaks at regular intervals.
Some features, like ad spend and unit price, were correlated with units sold.

```
# Initial inspection
print(data.head())
print(data.info())

# Plotting sales over time
plt.figure(figsize=(12, 6))
sns.lineplot(x='date', y='units', data=data)
plt.title('Units Sold Over Time')
plt.xlabel('Date')
plt.ylabel('Units Sold')
plt.show()

# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

4. Feature Engineering

Date Features: Extracted features from the date to capture temporal information:

Year, Month, Day, Day of the Week, Week of the Year
Lag Features: Created lag features to incorporate past sales information:

Lag 1, Lag 7, Lag 30
Rolling Mean Features: Created rolling mean features to smooth out short-term fluctuations:

Rolling Mean 7, Rolling Mean 30
These features help the model understand temporal dependencies and patterns in the data.

```
# Extract date features
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day'] = data['date'].dt.day
data['dayofweek'] = data['date'].dt.dayofweek
data['weekofyear'] = data['date'].dt.isocalendar().week.astype(int)

# Lag features
data['lag_1'] = data['units'].shift(1)
data['lag_7'] = data['units'].shift(7)
data['lag_30'] = data['units'].shift(30)
```

```
# Rolling mean features
data['rolling_mean_7'] = data['units'].rolling(window=7).mean()
data['rolling_mean_30'] = data['units'].rolling(window=30).mean()

# Drop rows with NaN values created by shifting and rolling
data = data.dropna()
```

5. Model Selection

Initial Model Choice: The XGBoost model was chosen for its efficiency and performance in handling time series data.

XGBoost Overview: XGBoost is an ensemble learning method that uses gradient boosting frameworks. It is known for its speed and performance, handling missing data well, and preventing overfitting.
Model Training: The initial model was trained with default hyperparameters.

Evaluation Metric: Mean Squared Error (MSE) was used to evaluate model performance.

```
# Select features and target variable

features = ['year', 'month', 'day', 'dayofweek', 'weekofyear', 'lag_1', 'lag_7',
'lag_30', 'rolling_mean_7', 'rolling_mean_30', 'ad_spend', 'unit_price']
target = 'units'

X = data[features]
y = data[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False)

# Initialize and train the XGBoost model
xgb_model = XGBRegressor(objective='reg:squarederror', n_estimators=100)
xgb_model.fit(X_train, y_train)

# Predictions
y_pred = xgb_model.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

6. Hyperparameter Tuning

Parameter Grid: Defined a grid of hyperparameters to search for the best combination.

Parameters: Number of estimators, learning rate, and maximum depth.
Cross-Validation: Used TimeSeriesSplit for cross-validation to ensure the model is validated on consecutive time intervals.

Grid Search: Performed a grid search to find the best hyperparameters.

Best Model: Identified the best hyperparameters and retrained the model.

```
# Define the parameter grid
param_grid = {
   'n_estimators': [100, 200, 300],
   'learning_rate': [0.01, 0.1, 0.3],
   'max_depth': [3, 5, 7]
}

# Time series split for cross-validation
tscv = TimeSeriesSplit(n_splits=5)

# Grid search
grid_search =
GridSearchCV(estimator=XGBRegressor(objective='reg:squarederror'),
param_grid=param_grid, cv=tscv, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Best parameters and model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Predictions with best model
y_pred_best = best_model.predict(X_test)

# Evaluation
mse_best = mean_squared_error(y_test, y_pred_best)
print(f'Best Model Mean Squared Error: {mse_best}')
```

7. Results and Discussion

Model Performance: The best model showed significant improvement over the initial model based on MSE.

Visual Comparison: A plot comparing actual vs. predicted units sold showed the model's effectiveness in capturing the sales patterns.

Insights: The tuned model was better at capturing the seasonal patterns and trends in the sales data, leading to more accurate forecasts.

```
# Plot actual vs predicted

plt.figure(figsize=(12, 6))
plt.plot(data['date'][-len(y_test):], y_test, label='Actual')
plt.plot(data['date'][-len(y_test):], y_pred_best, label='Predicted')
plt.legend()
plt.title('Actual vs Predicted Units Sold')
plt.xlabel('Date')
plt.ylabel('Units Sold')
plt.show()
```

8. Conclusion:

Summary: This report outlines the process of forecasting unit sales using time series analysis. Key steps included data preprocessing, feature engineering, model selection, and hyperparameter tuning. The XGBoost model, after tuning, provided accurate forecasts.

Future Work: Future improvements could include:

Incorporating more external features like holidays or economic indicators.
Experimenting with other time series models like ARIMA, Prophet, or LSTM.
Implementing more advanced hyperparameter tuning techniques like Bayesian optimization.

This script outlines the feature engineering process, training a LightGBM model, and preparing the submission file. Ensure to adjust parameters and add more features if necessary based on further analysis.