

★ Member-only story

A Unified Machine Learning Framework for Time Series Forecasting



Shenggang Li · Following

Published in Data Science Collective · 13 min read · 1 day ago



318



4



Harness Diverse Algorithms to Improve Predictive Accuracy from Transactional Data

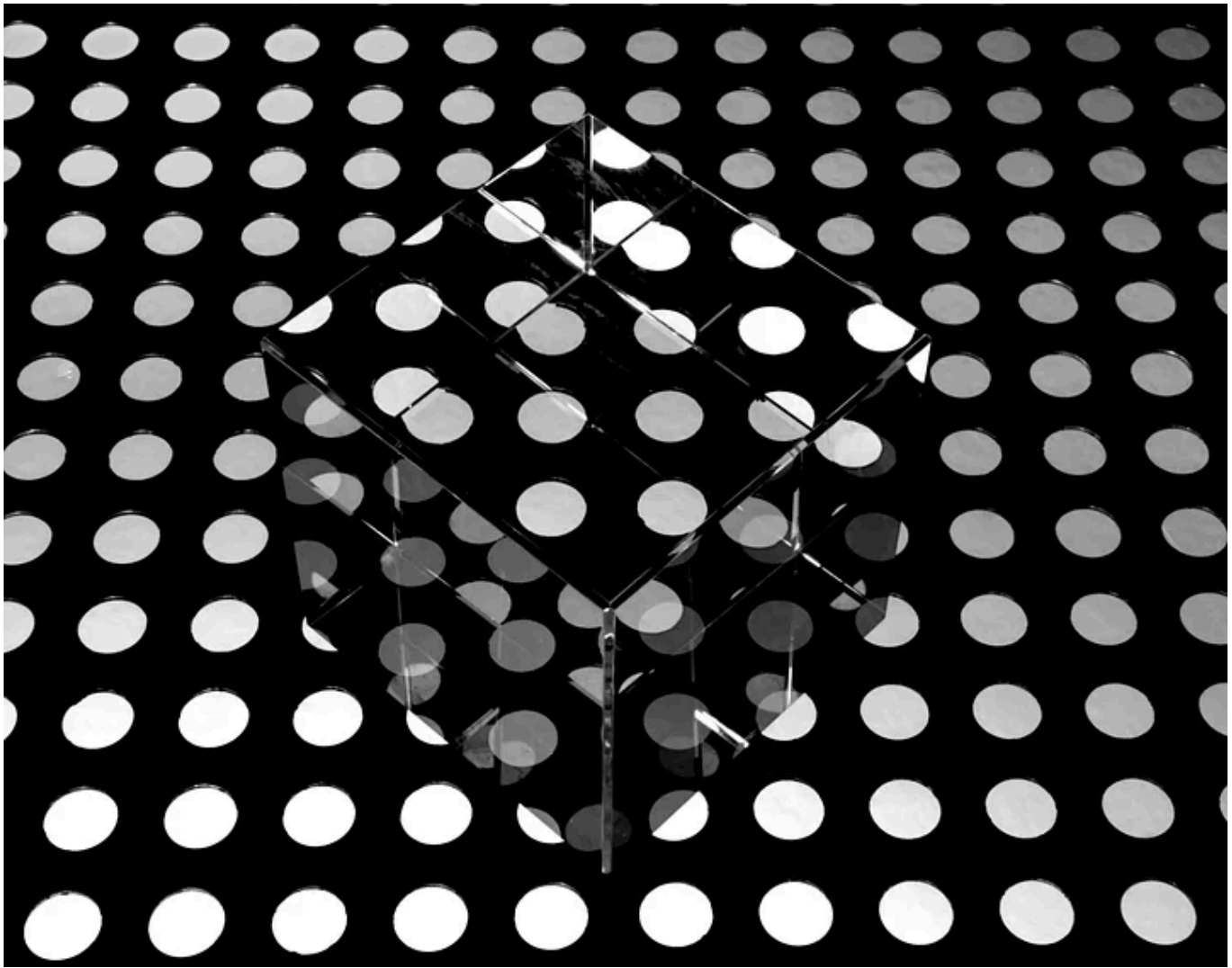


Photo by [Michael Dziedzic](#) on [Unsplash](#)

Introduction

Imagine you're running an online store, and you need to predict next month's sales. Should you stock up on inventory, hire more staff, or prepare for a slow period? That's where time series forecasting comes in — it helps businesses make smarter decisions by predicting future trends based on past data. Whether it's forecasting demand in supply chains, predicting stock market movements, or planning energy consumption, time series forecasting is a crucial tool for data scientists.

But here's the challenge: traditional forecasting models like *ARIMA* and Exponential Smoothing struggle in real-world scenarios. These structured

models assume that time series data follows predictable patterns — like a straight line with trends and seasonality — but reality is often far messier. External shocks (e.g., economic crises, sudden spikes in demand) and complex interactions (e.g., the impact of marketing campaigns on sales) can break these models.

Take *ARIMA*, for example — it works well when data has a clear trend and seasonality, but it crumbles when patterns shift unpredictably. Exponential Smoothing is great for short-term forecasts but fails when handling long-term dependencies. Even hybrid approaches, like *SARIMAX*, still rely on strict mathematical assumptions that don't always hold up in volatile environments.

That's why machine learning-based forecasting is gaining traction. In a previous post, *Time Series Forecasting: A Comparative Analysis of SARIMAX, RNN, LSTM, Prophet, and Transformer*, I explored different forecasting methods. However, I left out machine learning approaches because the post was already packed with information. So, in this post, I'm filling that gap.

We'll dive into different time series models, explore how they work, and discuss why I developed a generalized machine learning framework for forecasting transactional data. While applying machine learning to time series forecasting isn't new, this framework is designed to tackle real-world forecasting challenges — handling data irregularities, sudden shocks, and feature-driven predictions across multiple scenarios. Whether you're predicting e-commerce sales, customer demand, or financial trends, this approach offers greater flexibility and adaptability.

Let's get started!

Forecasting vs. Prediction: What's the Difference?

Out of Sample vs. Out of Time

People often use “forecasting” and “prediction” interchangeably, but in the world of time series analysis, they mean different things. Think of it this way: forecasting is like looking ahead on a road trip using past travel speed and road conditions to estimate when you'll reach your destination.

Prediction, on the other hand, is like guessing how a new car model will perform based on past data from similar cars.

In simple terms, forecasting is all about time — it predicts future data points based on historical trends. This is why models that estimate future values in a sequence are called time series forecasting models. Traditional prediction models, like linear regression, don't necessarily account for time. Instead, they predict outcomes within the same timeframe but for new or unseen samples.

Let's break it down with an example:

- **Forecasting Example:** Predicting next year's sales using five years of monthly data — an out-of-time prediction based on trends and seasonality.
- **Prediction Example:** Estimating a new product's first-month sales using past sales of similar products — an out-of-sample prediction within the same timeframe.

In sort, time-series forecasting considers the sequence of past values to make future projections, while traditional predictive models focus on finding patterns within a dataset without necessarily considering time dependencies. Forecasting is often more complex because real-world data can have trends, seasonality, and sudden shifts — things a simple regression model might miss.

Machine Learning for Time Series Forecasting

Building time series forecasting and machine learning models takes different approaches since they handle data differently. But machine learning can still be a powerful tool for time series forecasting.

Organize Data into Temporal Patterns

- Turn the dependent variable (e.g., sales) into an independent one using lagged values.
- Add seasonal dummies for patterns (e.g., months, weekdays, holidays).
- Include trend variables to track long-term shifts.
- Use recent data (e.g., last 100 days) to keep things relevant.
- Transform the target variable as needed for better accuracy.

Model Validation

1. Do not use random splits due to temporal dependencies.
2. Use backtesting to sequentially test the model, identifying when it works and fails within specific time ranges.

Handling Outliers in Time Series

Outliers act as disruptions in time series data, often caused by external events. Here's how to manage them effectively:

- Adjust for interventions by redefining variables (e.g., adding new slope variables).
- Plan ahead for known events like promotions or holidays in forecasts.
- Consider data relationships to smooth out sudden spikes.

Retail Sales Example:

- Trend: Sales gradually increase over the month.
- Seasonality: Sales peak every Saturday.
- Interventions: Market promotions cause sudden sales spikes.
- Lagged Predictors: High sales one month often mean high sales the next.
- Partial Data: Using only recent data ensures the model captures the latest trends.

By handling outliers smartly, forecasts stay accurate and reflect real-world shifts.

A Generalized Machine Learning Framework

Think of time series data as an item's transaction history. Using machine learning for time series forecasting is just like building a predictive model for future

transactions — it's all about analyzing past data to predict what's next.

Let's use the following transactional data format for this study:

date	sales
2022-01-01	258
2022-01-02	218
2022-01-03	212
2022-01-04	218
2022-01-05	195
...	...
2023-08-19	243
2023-08-20	190

Transactional Sales Data

The time series forecasting model with a generalized transformation of sales and restricted to a specific time range can be represented as:

$$y_t = G(\text{trend}_t, \text{seasonal}_t, \text{event}_t, \text{lag}_t, \text{mov_avg}_t) + \varepsilon_t$$

Where:

- y_t is the forecasted transformed sales at time t , where $y_t = f(\text{sales}_t)$ and $f(\cdot)$ is a transformation function (e.g., $\log(\cdot)$, $\text{sqrt}(\cdot)$, $\text{logistic}(\cdot)$).
- $G(\cdot)$ represents a machine learning algorithm (such as linear regression, *LightGBM*, *XGBoost*, or Random Forest).

- $trend_t$ is the trend component at time t .
- $seasonal_t$ is the dummy variable (e.g., weekday, month, weekend flag).
- $event_t$ is the event component (e.g., holidays, promotions, special events).
- lag_t is the lagged sales values (e.g., sales at $n = 1, 2$, lag orders, etc.).
- mov_avg_t is the moving average of sales over a specified window.
- ϵ_t is the noise or error term.

To understand the rationale behind the machine learning (*ML*) framework, it's important to understand the *ARIMA* model, a fundamental method in time series analysis. *ARIMA* (AutoRegressive Integrated Moving Average) effectively models temporal dependencies and captures underlying patterns in time series data:

$$y_t = \mu + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \dots + \theta_q \epsilon_{t-q} + \epsilon_t$$

The above model can be extended to *SARIMA*, which explicitly includes seasonal components.

This machine learning-based time series model builds on *ARIMA* but goes further by:

- Using *ML* to capture non-linearity and interactions while keeping time series fundamentals.
- Focusing on recent data to stay relevant and maintain stationarity.
- Adding event markers for external impacts like promotions or holidays.

- Transforming the target variable to manage dispersion and non-stationarity.

Code and Interpretation

Here, I provide a detailed explanation of the code for implementing the above framework.

Data Loading and Preprocessing

First, I built the Python code using simulated data to predict sales. Then, I tested it with both generated data and the publicly available [Electric Production tm.csv](#) dataset to fully validate the results.

```
file = 'simulation.csv'
DF = pd.read_csv(file)
DF.columns = ['date', 'sales']
DF = DF.sort_values(['date'])
```

Feature Engineering

A function *create_data* is defined to extract date-related features and create lagged variables and moving averages. This function is important for capturing seasonality, trends, events, and various lag effects in the sales data.

```
# Feature creation and engineering
def create_date(df, x, lag_n):
    df = df.copy()
    df['date'] = pd.to_datetime(df['date'])
```

```

df['weekday'] = df['date'].dt.day_name()
df['month'] = df['date'].dt.month_name()

df = pd.get_dummies(df, columns=['weekday', 'month'], prefix='is')

holidays = cal.holidays(start=df['date'].min(), end=df['date'].max())
df['is_holiday'] = df['date'].isin(holidays).astype(int)

df['t'] = (df['date'] - df['date'].min()).dt.days
df['t_lg'] = np.log1p(df['t'])

for lag in range(1, lag_n + 1):
    df[f'lag{lag}'] = df[x].shift(lag)

df['rolling_mean_7'] = df[x].rolling(window=7).mean()
df['rolling_std_7'] = df[x].rolling(window=7).std()
df['rolling_mean_30'] = df[x].rolling(window=30).mean()
df['rolling_std_30'] = df[x].rolling(window=30).std()

df = df.fillna(0)

new_DF = create_date(DF, 'sales', 3)

```

Transformation Functions

The following functions can handle the transformation of the target variable (sales) to stabilize variance and improve model performance.

```

# For logistic transformation
max_y = new_DF['sales'].max()

def transform_target(y, transform_fun):
    if transform_fun == 'log':
        return np.log(y)
    elif transform_fun == 'sqrt':
        return np.sqrt(y)
    elif transform_fun == 'logistic':
        return 1 / (1 + np.exp(-y/max_y))
    else:
        raise ValueError("Unsupported transformation function. Choose from 'log'

def inverse_transform_target(y, transform_fun):
    if transform_fun == 'log':

```

```
        return np.exp(y)
    elif transform_fun == 'sqrt':
        return np.square(y)
    elif transform_fun == 'logistic':
        return -np.log((1 / y) - 1)*max_y
    else:
        raise ValueError("Unsupported transformation function. Choose from 'log'
```

Model Training

We define the function to train various machine learning models, including *LightGBM*, *XGBoost*, Linear Regression, and Random Forest.

```
def train_ml_model(X_train, y_train, X_valid, y_valid, model_type='lightgbm', transform_fun=None):
    if transform_fun is not None:
        y_train = transform_target(y_train, transform_fun)
        y_valid = transform_target(y_valid, transform_fun)
    if model_type == 'lightgbm':
        train_data = lgb.Dataset(X_train, label=y_train)
        valid_data = lgb.Dataset(X_valid, label=y_valid)
        params = {'objective': 'regression', 'metric': 'rmse', 'verbosity': -1}
        model = lgb.train(params=params, train_set=train_data, valid_sets=[valid_data])
    elif model_type == 'xgboost':
        model = xgb.XGBRegressor(objective='reg:linear', n_estimators=150)
        model.fit(X_train, y_train, early_stopping_rounds=10, eval_set=[(X_valid, y_valid)])
    elif model_type == 'linear_regression':
        model = LinearRegression()
        model.fit(X_train, y_train)
    elif model_type == 'random_forest':
        model = RandomForestRegressor(n_estimators=150)
        model.fit(X_train, y_train)
    else:
        raise ValueError("Unsupported model type. Choose from 'lightgbm', 'xgboost', 'linear_regression', 'random_forest'")
    return model
```

Feature Selection

I select the most relevant features for the machine learning model based on their correlation (absolute values) and threshold with the target (sales) variable.

```
# Function to create correlation (absolute values) with the target
def getcorr_cut(Y, df, varnamelist, thresh):
    corr = []
    meanv = []
    for vname in varnamelist:
        X = df[vname]
        C = np.corrcoef(X, Y)
        beta=np.round(C[1, 0],4)
        corr.append(beta)
        avg = round(X.mean(), 6)
        meanv.append(avg)
    corrdf = pd.DataFrame({'varname': varnamelist, 'correlation': corr, 'mean':
    corrdf['abscorr'] = np.abs(corrdf['correlation'])
    corrdf.sort_values(['abscorr'], ascending=False, inplace=True)
    corrdf['order'] = range(1,len(corrdf)+1)
    corrdf['meanabs'] = np.abs(corrdf['mean'])
    corrdf['abscorr'] = corrdf['abscorr'].fillna(0.0)
    corrdf = corrdf[(corrdf.abscorr >= thresh)]
    return corrdf

X = df[features]
y = df['sales']
cor_df = getcorr_cut(y, X, features, 0.05)
features = list(cor_df.varname)
X = df[features]
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, shuff
```

Rolling Window Validation

I also created a function with a rolling window approach to test the model — just like in real life, where new data keeps coming in over time.

```
# Use backtesting to sequentially test the model
def rolling_window_validation(Df, target_col, lag_n, train_size, test_size, feat
```

```

predictions = []
actuals = []
test_indices = []
# You can start testing from other points
for start in range(0, len(DF) - train_size - test_size + 1, test_size):
    train = DF[start:start + train_size]
    test = DF[start + train_size:start + train_size + test_size]
    test_indices.extend(test.index)
    X_train = train.drop(columns=[target_col, 'date'])
    y_train = train[target_col]
    X_test = test.drop(columns=[target_col, 'date'])
    y_test = test[target_col]
    X_train = X_train[features]
    X_test = X_test[features]
    model = train_ml_model(X_train, y_train, X_test, y_test, model_type=model_type)
    y_pred = model.predict(X_test)
    if transform_fun is not None:
        y_pred = inverse_transform_target(y_pred, transform_fun)
    predictions.extend(y_pred)
    actuals.extend(y_test)

# Use MAPE % to represent the error term
mape_rt = mean_absolute_percentage_error(actuals, predictions)
return predictions, actuals, test_indices, model, mape_rt

# Perform rolling window validation
train_size = 340
test_size = 50
predictions, actuals, test_indices, model, mape_rt = rolling_window_validation(n

```

Next, I will introduce the forecasting code to forecast future sales using the proposed machine learning framework.

Forecasting the Next Day's Sales

The *forecast_next_1_day* function predicts the sales for the next day by creating the necessary features and applying the trained model. This function is important because forecasting future sales requires iterative predictions, where the forecasted result of the current day serves as an input for the next day's forecast.

```

def forecast_next_1_day(model, DF, features):
    # Ensure the data frame is sorted by date
    DF = DF.sort_values('date')

    # Create the next day's date
    next_date = DF['date'].max() + pd.Timedelta(days=1)

    # Create a new row for the next day's features
    new_row = pd.DataFrame(index=[0])

    # Fill the new row with feature values
    new_row['date'] = next_date
    new_row['t'] = (next_date - DF['date'].min()).days
    new_row['t_lg'] = np.log(new_row['t'] + 1)

    # Calculate lag features
    for lag in range(1, 4): # Assuming lag_n=3 from the provided create_date fu
        new_row[f'lag{lag}'] = DF['sales'].shift(lag).iloc[-1]

    # Calculate rolling statistics
    new_row['rolling_mean_7'] = DF['sales'].rolling(window=7).mean().iloc[-1]
    new_row['rolling_std_7'] = DF['sales'].rolling(window=7).std().iloc[-1]
    new_row['rolling_mean_30'] = DF['sales'].rolling(window=30).mean().iloc[-1]
    new_row['rolling_std_30'] = DF['sales'].rolling(window=30).std().iloc[-1]

    # Extract day of the week and month for dummy variables
    new_row['weekday'] = next_date.day_name()
    new_row['month'] = next_date.month_name()

    dummies_wkd = pd.get_dummies(new_row['weekday'], prefix='is')
    new_row = pd.concat([new_row, dummies_wkd], axis=1)

    dummies_mon = pd.get_dummies(new_row['month'], prefix='is')
    new_row = pd.concat([new_row, dummies_mon], axis=1)

    holidays = cal.holidays(start=DF['date'].min(), end=DF['date'].max())
    new_row['is_holiday'] = next_date in holidays

    # Fill missing dummy variables with 0 if they do not exist
    missing_columns = set(features) - set(new_row.columns)
    for col in missing_columns:
        new_row[col] = 0

    # Ensure the new row contains only the required features
    new_row = new_row[features + ['date']]

    # Predict the sales for the next day
    sales_prediction = model.predict(new_row[features])[0]

```

```
return sales_prediction, new_row
```

Forecasting the Next 7 Days' Sales

The *forecast_next_7_days* function defined below uses the previous function *forecast_next_1_day* iteratively to forecast sales for the next 7 days. By updating the DataFrame with each new prediction, this function ensures that each day's forecasted result is used as input for the subsequent day's forecast.

```
def forecast_next_7_days(model, DF, features):  
    predictions = []  
    for _ in range(7):  
        sales_prediction, new_row = forecast_next_1_day(model, DF, features)  
        predictions.append(sales_prediction)  
  
        # Append the prediction to the DataFrame to update lags and rolling stat  
        new_row['sales'] = sales_prediction  
        DF = pd.concat([DF, new_row[['date', 'sales'] + features]], ignore_index=True)  
  
    return predictions, DF
```

Applying the Forecasting Function

We then use the function *forecast_next_7_days* function to forecast the sales for the next seven days and update the DataFrame with the forecasts.

```
# Predict the next 7 days' sales and update the DataFrame  
DF = new_DF.copy()  
seven_day_preds, updated_DF = forecast_next_7_days(model, DF, features)
```

Integrating Predictions with Historical Data

We can add the predictions to the original DataFrame and prepares it for visualization:

```
# Add the 'pred' column for testing data predictions
new_DF['pred'] = np.nan
new_DF.loc[test_indices, 'pred'] = predictions

# Add the new forecasted sales data to new_DF
forecast_dates = pd.date_range(start=new_DF['date'].max() + pd.Timedelta(days=1)
forecast_df = pd.DataFrame({'date': forecast_dates, 'pred': seven_day_preds, 'sa
new_DF = pd.concat([new_DF, forecast_df], ignore_index=True)

# Display the updated DataFrame with predictions
print(new_DF.tail(15))

new_DF[new_DF.pred.isnull()==False][['date', 'sales', 'pred']].head(100)
new_DF[new_DF.pred.isnull()==False][['date', 'sales', 'pred']].tail(20)
```

Visualization

Finally, we can plot the training and testing data along with the forecasted sales for the next 7 days.

```
# Plot 1: Time series plot with training and testing data
plt.figure(figsize=(14, 7))
plt.scatter(new_DF['date'][:train_size], new_DF['sales'][:train_size], color='b')
plt.scatter(new_DF['date'][train_size:train_size + test_size], new_DF['sales'][t
plt.scatter(new_DF['date'][train_size:train_size + test_size], new_DF['pred'][tr
plt.xlabel('Date')
```

[Open in app](#) ↗

Medium

🔍 Search

✍ Write



```
# Plot 2: Last 25 points in current data and next 7 days forecasted sales
plt.figure(figsize=(14, 7))
```



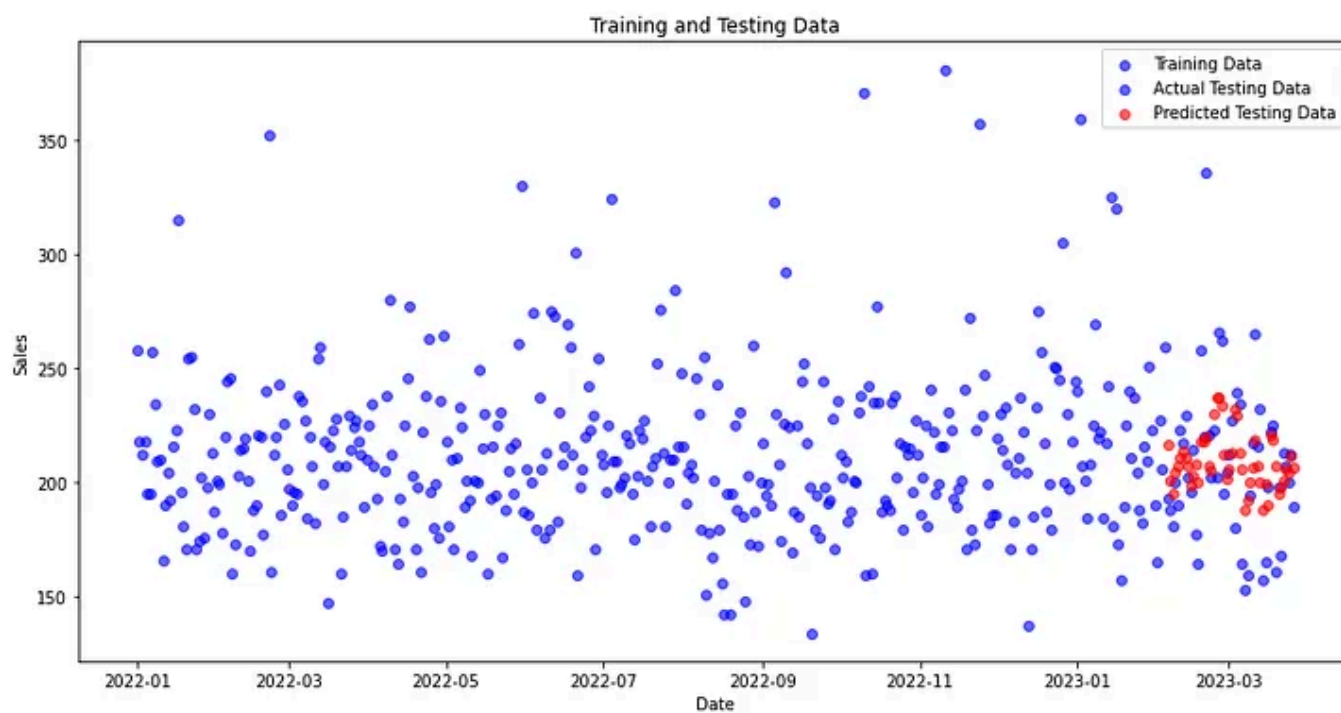
```

last_25_data = new_DF[-(25 + 7):-7]
forecast_data = new_DF[-7:]

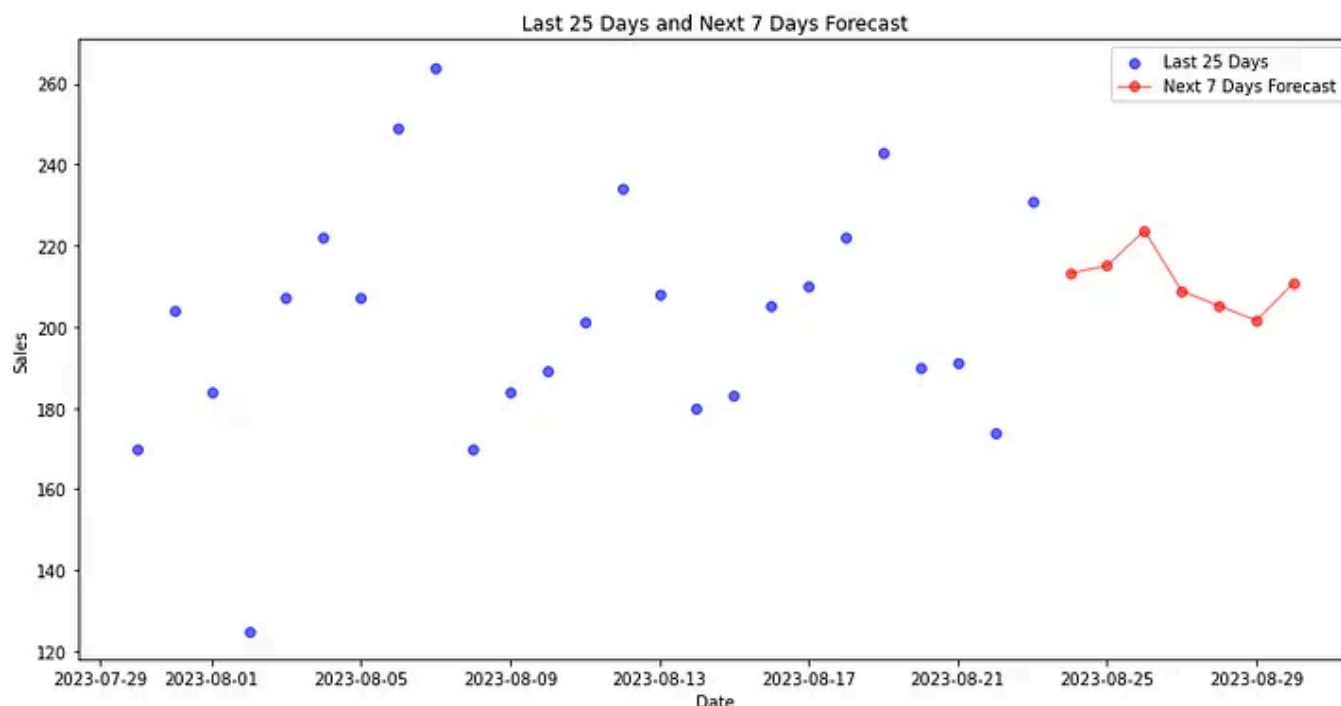
plt.scatter(last_25_data['date'], last_25_data['sales'], color='blue', label='La
plt.plot(forecast_data['date'], forecast_data['pred'], color='red', linestyle='--

plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Last 25 Days and Next 7 Days Forecast')
plt.legend()
plt.show()

```



Training and Testing Data in Model Training Stage



Training and Future Forecasted Data in Model Forecasting Stage

Comparative Analysis

I tested the code from the previous section using two datasets: one is the Electric Production dataset from Kaggle, which analyzes industry energy consumption trends over time (see my other paper: [Time Series Forecasting: A Comparative Analysis of SARIMAX, RNN, LSTM, Prophet, and Transformer](#)). The other dataset consists of my generated data for sales transactions.

I chose these datasets because the first one is a typical time series dataset, and the second one is generated by myself to weaken the autoregressive and moving average effects while emphasizing other factors influencing sales, such as holidays, weekends, and specific days like Tuesdays. These factors are common in the retail business. By doing so, I can conduct a comparative

analysis of *SARIMAX* and machine learning approaches across different data conditions and scenarios.

Here are the testing results (*MAPE* Ratio) for the Electric Production dataset:

Algorithm	None	Sqrt	Log	Logistic
LightGBM	3.10%	3.10%	3.10%	3.10%
Linear Regression	4.60%	5.20%	6.80%	5.70%
XGBoost	3.80%	3.60%	3.20%	3.60%
Random Forest	3.40%	3.40%	3.30%	3.40%
SARIMAX	44.20%	4.30%	4.20%	4.50%

Testing Results for Electric Production

We can see that *LightGBM* consistently shows the lowest error across all transformations, indicating its robustness and reliability for this dataset. *XGBoost* also performs well, particularly with the log transformation. Random Forest demonstrates stable performance across all transformations. In contrast, Linear Regression shows increased error rates with transformations, especially the log transformation. *SARIMAX* benefits from transformations but still lags behind machine learning models. Overall, machine learning models, particularly *LightGBM* and *XGBoost*, are more effective for time series forecasting in the Electric Production dataset. Target (sales) transformations can improve *SARIMAX*'s (Arima) performance.

Now let's see the testing results (*MAPE* Ratio) for the generated dataset:

Algorithm	None	Sqrt	Log	Logistic
LightGBM	9.20%	9.40%	9.20%	9.50%
Linear Regression	8.20%	8.10%	8.10%	8.10%
XGBoost	8.60%	8.30%	8.50%	8.50%
Random Forest	8.10%	8.00%	8.00%	8.10%
SARIMAX	15.90%	16.30%	16.60%	16.00%

Testing Results for Simulated Data

For the simulated sales transaction data, Linear Regression and Random Forest models reach the lowest *MAPE* ratios, indicating high performance and stability across different transformation functions. *XGBoost* performs well, particularly with the sqrt transformation. *LightGBM*, although effective, shows slightly higher errors than Linear Regression and Random Forest. *SARIMAX* consistently has the highest error rates, indicating it is less suitable for this dataset.

If the data lacks features of autoregressive items (lags) and moving average factors, traditional *ARIMA*-based methods like *SARIMAX* struggle to provide accurate forecasts. In contrast, machine learning models are more flexible as they can easily capture various features such as holidays, time events, and nonlinear trends.

The Python scripts and data are available in my GitHub repository at:

https://github.com/datalev001/tsa_ml

Final Thoughts & Next Steps

We built a **machine-learning framework for time series forecasting** using feature engineering, improving on ARIMA by capturing trends, holidays, and nonlinear patterns with models like LightGBM, XGBoost, and random forests. This flexible approach handles external variables and real-world complexities better than traditional methods.

To further enhance forecasting:

- Fine-tune models with hyperparameter optimization.
- Update in real time as new data arrives.
- Blend models (e.g., XGBoost + LSTM) for short- and long-term patterns.
- Use deep learning for complex temporal dependencies.
- Add domain-specific features like promotions or weather.

With continuous improvements, we can make forecasts even smarter and more adaptable!

About me

With over 20 years of experience in software and database management and 25 years teaching IT, math, and statistics, I am a Data Scientist with extensive expertise across multiple industries.

You can connect with me at:

Email: datalev@gmail.com | [LinkedIn](#) | [X/Twitter](#)

Timeseries

Python

Forecasting

AI

Machine Learning



Published in Data Science Collective

833K Followers · Last published 1 day ago

Follow

Advice, insights, and ideas from the Medium data science community



Written by Shenggang Li

2K Followers · 75 Following

Following

Responses (4)



Alex Mylnikov

What are your thoughts?



Paolo Perrone

1 day ago



Great article Shenggang!



3

[Reply](#)**Vinicious Marketing**

11 hours ago



This so thoughtfull 🧐 but you still been having low engagement on your article, so you need to get your article promoted on Fiverr to increase your engagement, Check this out... [more](#)



1

[Reply](#)**Stéphane Chalon**

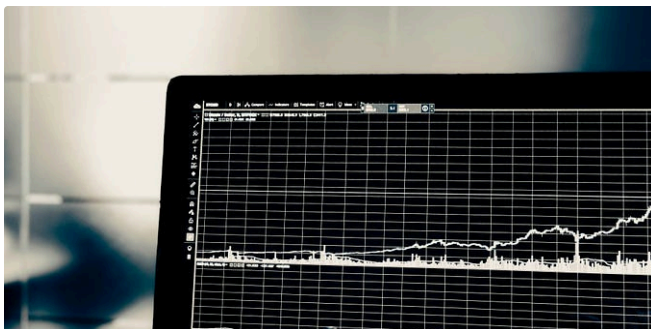
57 mins ago (edited)



In similar studies (MLR, SVR, LSTM) I found interesting to filter input data using Wavelets (Haar or Daubechies) as they allow to remove noise from raw data. They have advantage to keep timestamp

[Reply](#)[See all responses](#)

More from Shenggang Li and Data Science Collective



In Towards AI by Shenggang Li



In Data Science Collective by Paul Iusztin



Beyond Buy-and-Hold: Dynamic Strategies for Unlocking Long-...

Harnessing Survival Analysis and Markov Decision Processes to Surpass Static ETF...

★

Feb 7

👤

317

💬

7

🔖

⋮

				populated exclusively by single socks. Further research is needed to establish inter-dimensional laundry protocols.
The Aerodynamics of Toast: Why it Always Lands Butter-Side Down	2345.12345	Jane Doe	05/21/1998	Through 500 controlled toast-dropping experiments, we confirmed a 96% probability of butter-side-down landings. Analysis reveals a complex interplay of toast rotation, butter weight distribution, and Murphy's Law. We propose a novel "anti-butter" coating to mitigate breakfast-related tragedies.
The Acoustic Properties of	1111.22222	Joe Schmo	11/05/2008	This study explores the phenomenon of snack acoustics,

DSC

In Data Science Collective by Emily Gibbs

How to Set Up and Use a Vector DB In Less Than 10 Minutes

Speed up your vector searches when high latency isn't an option

★

Feb 14

👤

250

💬

2

🔖

⋮

Building a TikTok-like recommender

Scaling a personalized recommender to millions of items in real-time

1d ago

👤

78

💬

3

🔖

⋮



TN

In Towards AI by Shenggang Li

Practical Guide to Distilling Large Models into Small Models: A Nove...

Comparing Traditional and Enhanced Step-by-Step Distillation: Adaptive Learning,...

★

Mar 3

👤

120

💬

2

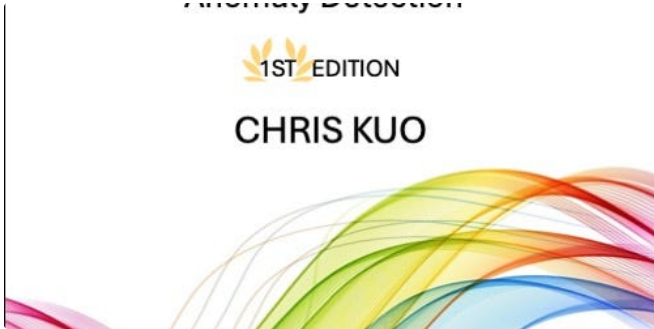
🔖

⋮

See all from Shenggang Li

See all from Data Science Collective

Recommended from Medium

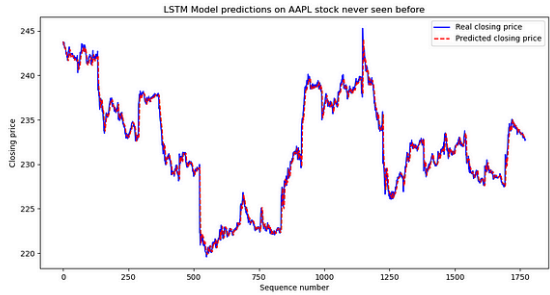


 In Dataman in AI by Chris Kuo/Dr. Dataman

Monte Carlo Simulation for Time Series Probabilistic Forecasting

Its application on stock market prices


★ Mar 14, 2024  644  6  



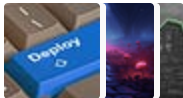
 edgard hall

I Tried Combining LSTM with Mean Reversion for Stock Prediction — ...

How AI can extract noise from the market , allowing traders to trade on it

Mar 2  51  6  

Lists



Predictive Modeling w/ Python

20 stories · 1853 saves



Practical Guides to Machine Learning

10 stories · 2221 saves



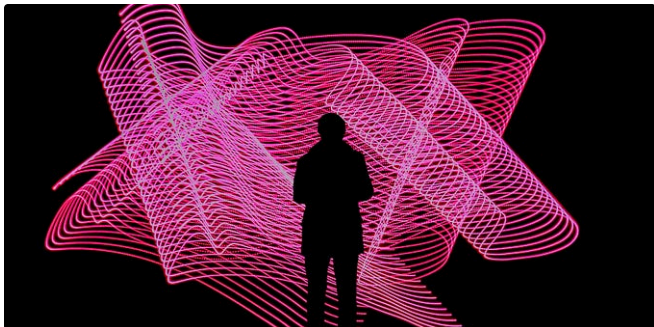
Coding & Development


11 stories · 1030 saves

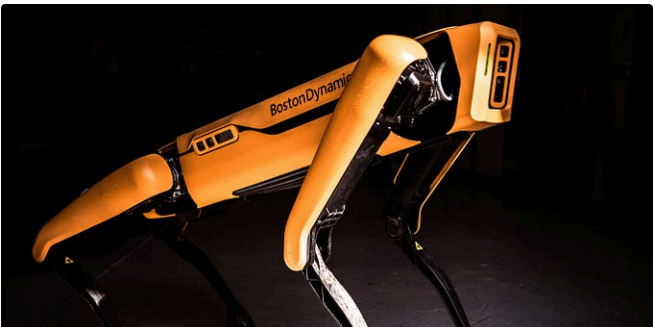


Natural Language Processing

1973 stories · 1617 saves



 In Data Science Collective by Rohith Teja

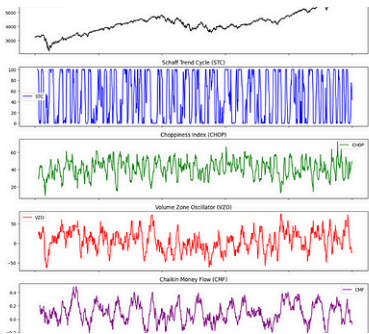


 In Towards AI by Shenggang Li

Let’s Build and Test a Simple Spatiotemporal Graph Neural...

How to build a spatiotemporal graph and preprocess it?

Feb 19 3



Kridtapon P.

4 Must-Have Technical Indicators for Algorithmic Trading

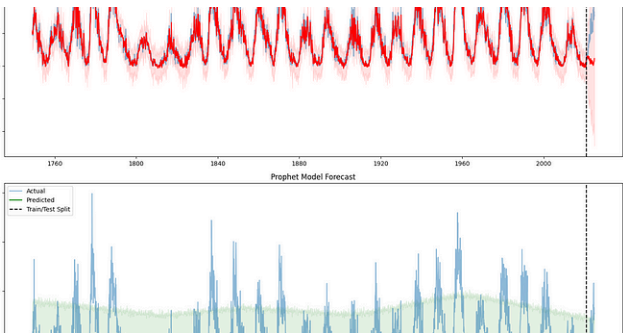
A practical guide to using STC, CHOP, VZO, and CMF to improve trend analysis and...

Feb 25 70 1

Dynamic Time Series Model Updating

Enhancing Forecast Accuracy with Incremental Adjustments Without Rebuildin...

Sep 11, 2024 282 1



Kyle Jones

Bayesian Time Series Forecasting using Orbit-ML and Prophet in...

Orbit is an open-source Python package developed by Uber for Bayesian time series...

Jan 31 60 3

See more recommendations