DataFlow 2025

Mastering the Data Waves

Team Project First Round

FORECASTING BUSINESS PERFORMANCE [VN]

Members:

Pham Duy Anh (leader)	anhpd.23BI14023@usth.edu.vn
Le Quoc Anh	anhlq.23BI14025@usth.edu.vn
Nguyen Thai Nhat Anh	anhntn.23BI14024@usth.edu.vn
Nguyen Lam Tung	tungnl.23BI14446@usth.edu.vn

Team No Name

2025/2026

Abstract

All of our Documents will be contained here: Link
A quick Overview of what is in our Github Repository

- The **Data** folder contains:
  - The Original Dataset stored in .xlsx file, which contains 3 sheets
  - Extracted sheets into .csv files and the Sales data already split into 'train' and 'test'
  - A .sav file that is extracted from Data Cleaning and Feature Engineering to run Statistical Analysis on SPSS
- The **Model** folder contains:
  - The Implementation and Training of ARIMA Model
  - The Implementation and Training of LSTM/RNN Model
  - The Implementation and Training of XGBoost and Tree-Based Model
- The **Preprocessing** folder contain:
  - A file that specifies how we cleaned and transformed the data into 1 final table
  - A file that compresses all of the cleaning and transforming part into 1 block and export into 1 file for model training
  - The Exported .csv file for Model training
- The **Report** folder contains:
  - The .pbix file which shows all of our Visualizations and Graphical Analysis implemented on PowerBI.

Contents

List of Figures

1 Introduction

1.1 Project Background and Scope

1.1.1 Background

A major fashion company in the United States operates across a vast market, generating sales in over 14,000 cities across the Central, Western, and Eastern regions. As a Data Analyst in the company's Data Analysis department, your task is to analyze past index, forecast the company's future sales and provide data-driven suggestions to increase the company performance.

1.1.2 Project Scope

- Analyze sales data from the years 2010 to 2020.
- Build a forecasting model for the years 2021-2022 to assess its accuracy and make predictions for 2023.

1.2 Dataset Introduction

The original data is an .xlsx file consisting of three sheets (SalesData, Product, Geography), with the Fact table being Sales, containing 976,243 rows and 6 columns.

1.3 Cleaning and Transforming the Data

With the 3 datasets: 1 Fact and 2 Dimensions table, we can clean and merging thm into 1 main table used for training and testing models

The code of each step will be given below:

```
#Split into 3 different DataFrames
sales = df['SalesFact']
geo = df['Geography']
product = df['Product']

#Convert Date column into datetime format
sales['Date'] = pd.to_datetime(sales['Date'])

#Handle null in Revenue
sales['Revenue'] = sales['Revenue'].replace(np.nan, sales['Revenue'].median())

#Add column profit
sales['Profit'] = sales['Revenue'] - sales['COGS']

#Split City in Geography
geo['City'] = geo['City'].str.split(',').str[0]

#Split District in Geography
geo['District'] = geo['District'].str.split('#').str[1]

#Join Sales and Geo into 1
sales = pd.merge(sales, geo, on = 'Zip', how = 'left')

#Delete some columns in Sales table
sales = sales.drop(['State', 'District', 'Zip'], axis = 1)

#Join Sals and Product
sales = pd.merge(sales, product, on = 'ProductID', how = 'left')

#Get product brand
sales['Product_Brand'] = sales['Product'].str.split(' ').str[0]
```

```
#Delete some columns
sales = sales.drop(['Product', 'ProductID','Unnamed: 6'], axis = 1)

sales['Month'] = sales['Date'].dt.month
sales['Year'] = sales['Date'].dt.year
sales['Day'] = sales['Date'].dt.day

#Split sales into train and test
train = sales[sales['Date'].dt.year <= 2020]
test = sales [sales['Date'].dt.year >= 2021]
```

After cleaning and transforming the data, it was merged into a single table with 976,243 rows and 14 columns.

	ID	Date	Units	Revenue	COGS	Profit	City	Region	Category	Segment	Product_Brand	Month	Year	Day
0	0	2013-07-31	12	19648.44	12309.747660	7338.692340	Austin	Central	Urban	Convenience	Pirum	7	2013	31
1	1	2014-03-12	16	20351.52	13497.128064	6854.391936	Torrance	West	Rural	Productivity	Natura	3	2014	12
2	2	2013-11-29	26	111367.62	91488.499830	19879.120170	Salem	East	Urban	Convenience	Currus	11	2013	29
3	3	2018-08-29	12	36280.44	21967.806420	14312.633580	Lithia Springs	East	Rural	Select	Pirum	8	2018	29
4	4	2013-04-27	14	55557.18	48645.866808	6911.313192	Troup	Central	Urban	Convenience	Natura	4	2013	27
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
976238	976238	2013-08-12	1	1070.37	749.794185	320.575815	Mckeesport	East	Rural	Productivity	Aliqui	8	2013	12
976239	976239	2011-12-04	1	1070.37	749.794185	320.575815	Littleton	Central	Rural	Productivity	Aliqui	12	2011	4
976240	976240	2011-05-29	1	1070.37	749.794185	320.575815	Cumming	East	Rural	Productivity	Aliqui	5	2011	29
976241	976241	2014-04-01	1	1070.37	749.794185	320.575815	Rocheport	Central	Rural	Productivity	Aliqui	4	2014	1
976242	976242	2012-12-22	1	1070.37	749.794185	320.575815	Hurricane	East	Rural	Productivity	Aliqui	12	2012	22

976243 rows × 14 columns

Figure 1: Overview of Final Data

2 Data Descriptive Analysis (EDA & Visualize).

2.1 Overall Sale Index

Chronologically

Below is a chart illustrating the company's revenue distribution over 10 years from 2010 to 2020. It clearly shows a declining revenue trend since 2014. The company's highest revenue periods typically fall in Q2, particularly in the months of April, June, and August each year.

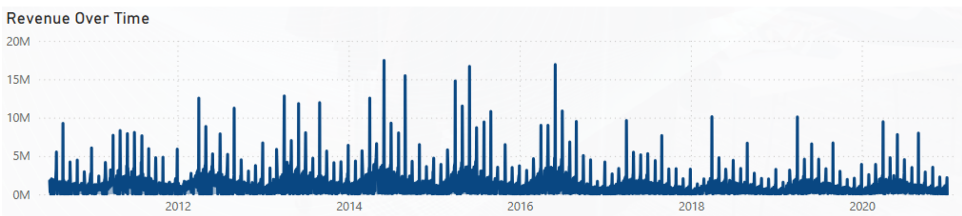


Figure 2: Revenue across time

In reality, after analyzing individual cities, the Revenue decline can largely be attributed to the fact that more than 4,000 cities recorded sales for only a short period (less than 5 days) before disappearing. This phenomenon occurred between 2013 and 2017, significantly contributing to the surge in total revenue during that time. Meanwhile, the revenue of major cities remained stable and did not show a noticeable decline throughout the timeline.

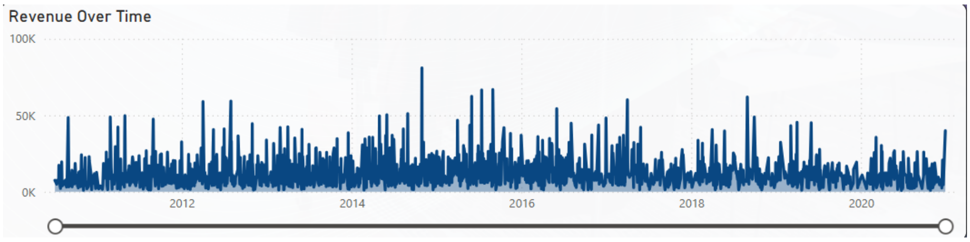


Figure 3: Revenue of a top-performing city.

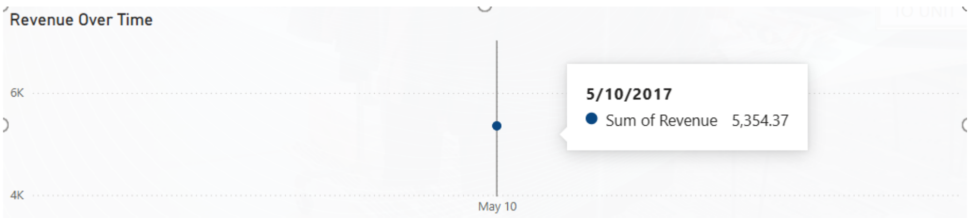


Figure 4: Example of revenue appearing once and vanished

```python
city_counts = df['City'].value_counts()

# Find the number of cities that appear only once
unique_city_count = (city_counts < 5).sum()
print(f'City appears less than 5 times: {unique_city_count} cities')
```

```
✓ 0.1s
City appears less than 5 times: 4127 cities
```

Figure 5: Total number of cities recording revenue for less than 5 days.

```python
city_counts = df['City'].value_counts()

# Get cities that appear < 5 times
cities_less_than_5 = city_counts[city_counts < 5].index

# Filter the original DataFrame
filtered_df = df[df['City'].isin(cities_less_than_5)]

# Count distinct cities per year
yearly_city_count = filtered_df.groupby('Year')['City'].nunique().reset_index()


# Sort the result
yearly_city_count = yearly_city_count.sort_values(by='Year')

print(yearly_city_count)
```

```
✓ 0.2s
    Year  City
0   2010   329
1   2011   754
2   2012   793
3   2013   833
4   2014   877
5   2015   824
6   2016   761
7   2017   444
8   2018   398
9   2019   375
10  2020   421
11  2021   379
12  2022   241
```

Figure 6: Number of cities recording revenue for less than 5 days each year.

- It can be observed that every year, there are cities with recorded revenue for less than 5 days. However, from 2012 to 2016, this number increased significantly—doubling compared to other years—directly explaining the unusually high revenue during this period.

Another reason that may explain the decline in revenue in recent years is the emergence of the COVID-19 pandemic, which has impacted people's lifestyles and consumption habits.

About Geographical Distribution

The states with high revenue distribution (Light → Dark, Blue → Green) are mostly located in the Southeastern part of the Central Region, with the highest concentration in California.

This is understandable due to the nature of high industrialization and quality of living in these states

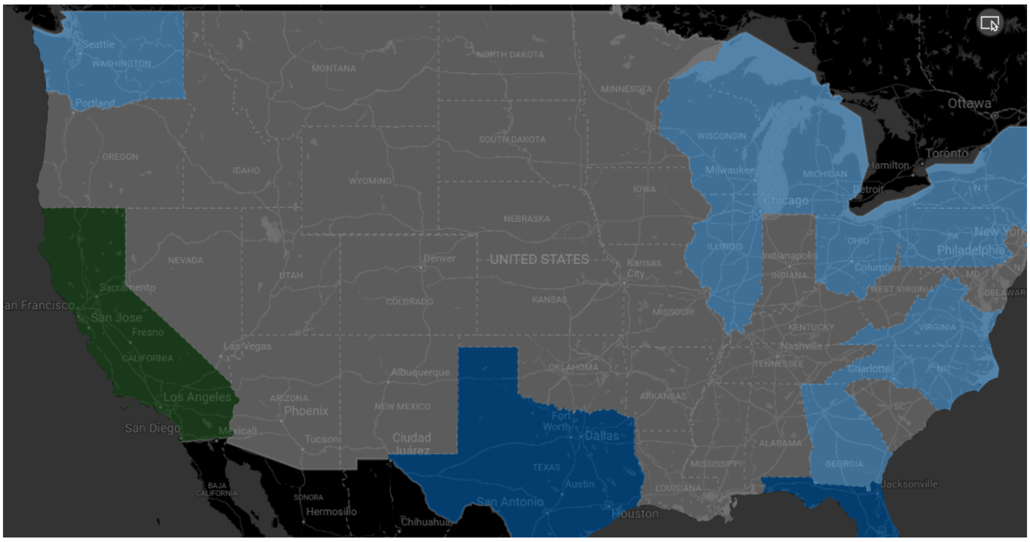


Figure 7: Sales map of Central Region

Some cities account for the company's main revenue share:

City	Revenue	Units Sold	Profit
Miami	26,472,570.39	4690	4157662.46
Houston	25,115,716.08	4190	4354357.80
Las Vegas	24,190,889.94	4268	4300447.70
San Diego	21,560,166.81	3992	3912525.64
Jacksonville	21,108,853.71	3425	3664535.67

Figure 8: Top 5 City with highest Sales

2.2 Detailed Sale Performance

June 1st (Children's Day and summer sale), August 30th (Back to school), and April 1st (Spring sale) record peak revenue each year.

Date	Units	Total Revenue
6/1/2014	3451	17,472,120.75
6/1/2016	3092	16,918,014.33
6/1/2015	3229	16,680,175.47
8/30/2014	3112	15,476,679.54
4/1/2015	2585	14,792,681.61
4/1/2013	2705	12,841,044.93
4/1/2014	2494	12,552,624.00
4/1/2012	2671	12,549,316.50
8/30/2013	2801	11,953,167.66
6/1/2013	2651	11,840,924.97

Figure 9: Top Dates with highest sales.

The Top Sales recorded date here may mostly be affected by the unexpected surge of sudden sale within cities, as listed above. The reason why they mainly focus on these 3 days will require more thorough discoveries

Sales index of Brands.

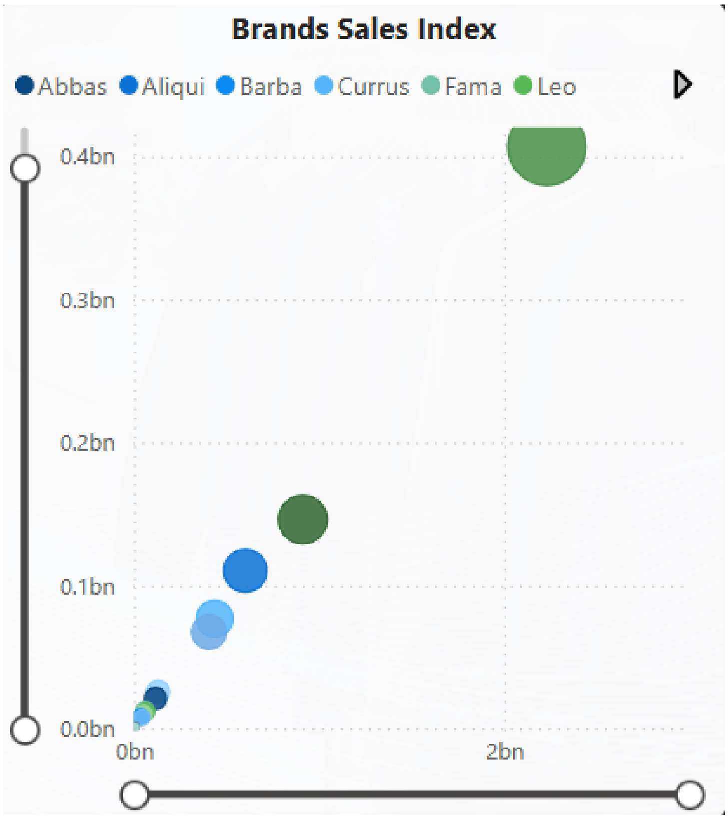


Figure 10: Revenue and Profit Growth of each Brand

The Maximus brand has been the best-selling brand over the years, surpassing all other brands. It is followed by Natura and Aliqui, whose sales are less than 50% of Maximus.

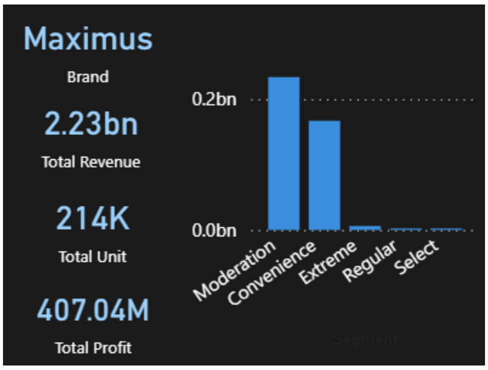


Figure 11: Sales index of Top 1 Brand

The best-selling Category and Segment over the years.

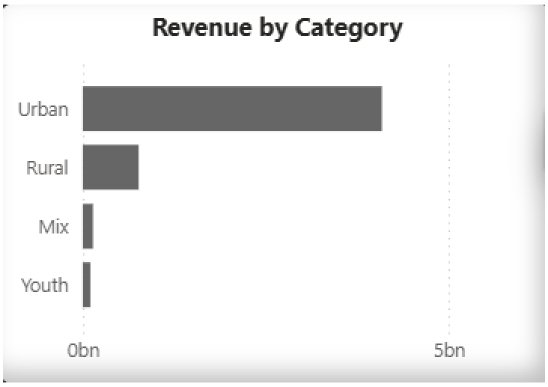


Figure 12: Revenue by Category

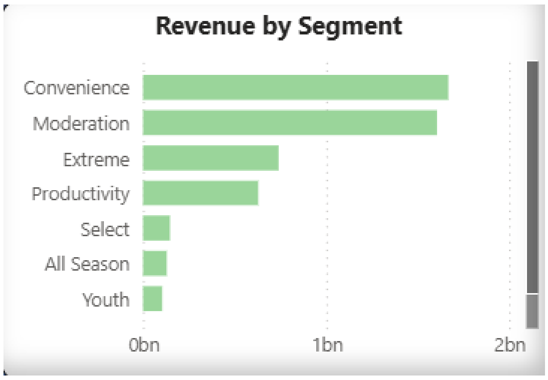


Figure 13: Revenue by Segment

Note: These are only the overall index of Categories and Segments, these will change depending on which Brands, timestamps or City you specifically search for

3 Predictive Models

- The presentation will use three predictive models with increasing accuracy: ARIMA → RNN/LSTM → XGBoost.
- Evaluation metrics: R-Square (R^2), Mean Absolute Percentage Error (MAPE), RMSE.

3.1 The AutoRegression Integrated Moving Average (ARIMA) Model

3.1.1 Prerequisites and Problems

Before running the ARIMA model, we need to perform clean and feature engineering 1 more time to get all relevant data to the same distinct timestamp

```
df_copy = df.groupby('Date').agg({
'Revenue': 'sum',
'Units': 'sum',
'COGS':'sum',
'City': lambda x: x.mode()[0],  # Most frequent value (mode)
'Segment': lambda x: x.mode()[0],
'Category': lambda x: x.mode()[0],
})
```

And save this as a new csv file

```
df_copy.to_csv('analyse_data.csv')
```

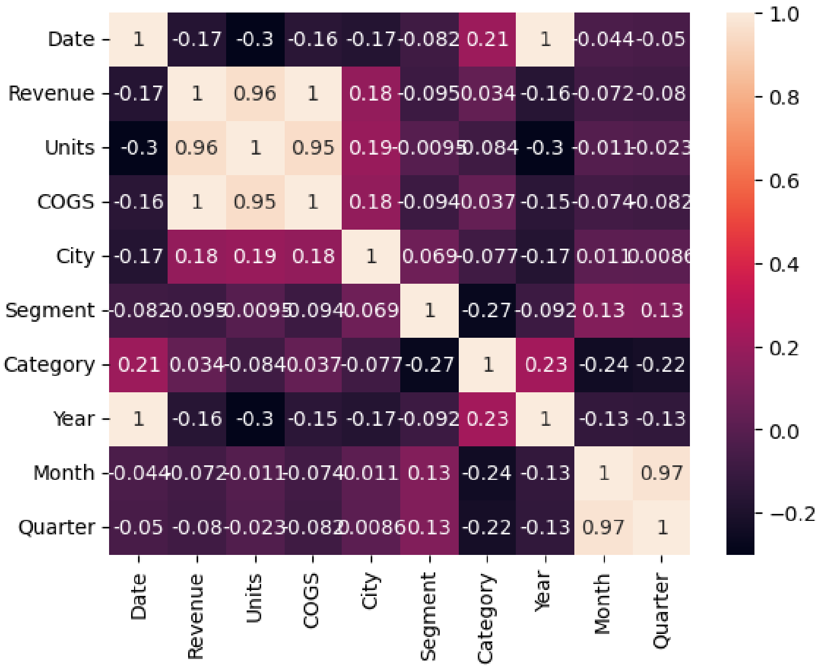
Now this step is to ensure that we bring every index into the same time structure without needing to create a rather frustrating hierarchical structure for each day, the new data will make sure that Date is distinct, Profit, COGS and Revenue are summed up and other available categorical data is taken as mode

	Date	Revenue	Units	COGS	City	Segment	Category	Year	Month	Quarter
0	2010-07-04	1765391.67	252	1.340364e+06	974	1	2	2010	7	3
1	2010-07-05	1425986.10	208	1.112580e+06	494	1	2	2010	7	3
2	2010-07-06	302463.00	33	2.373014e+05	779	1	2	2010	7	3
3	2010-07-07	1047787.65	181	8.425688e+05	727	1	2	2010	7	3
4	2010-07-08	771811.74	121	6.028101e+05	1118	1	2	2010	7	3
...	...	...	...	...	...	...	...	...	...	...
4248	2022-06-27	1625947.47	217	1.434694e+06	1080	3	2	2022	6	2
4249	2022-06-28	2042916.75	263	1.763163e+06	1015	3	2	2022	6	2
4250	2022-06-29	2284500.33	282	1.971644e+06	488	3	2	2022	6	2
4251	2022-06-30	700389.90	67	6.056699e+05	1060	3	2	2022	6	2
4252	2022-07-01	5312183.31	713	4.605905e+06	573	3	2	2022	7	3

4253 rows × 10 columns

Figure 14: New aggregated data

The new data after LabelEncoding categorical data and adding more detailed time intervals



By running the Correlation Heatmap, we can see a high multicollinearity between Revenue - Units - COGS, which is understandable, also the Time Hierarchy shows a high multicollinearity between its segments. In the final input data, only Revenue and Date will remain as Target and Timestamps, along with encoded City, Segment and Category as Exogenous features

```
decomposed = seasonal_decompose(main_train['Revenue'], model='additive',
    period=365)
decomposed.plot()
plt.show()
```

We will do one last thing is to check for Seasonality for SARIMAX model

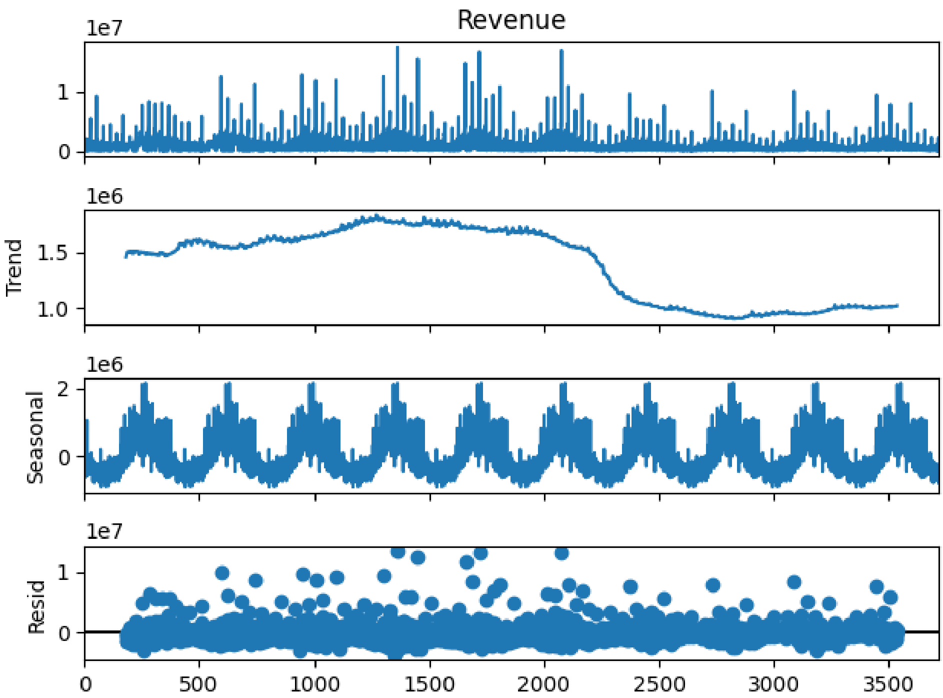


Figure 15: Data Seasonality

The clear, repeated fluctuations indicate a strong seasonal effect, where seasonal or recurrent monthly trends might affect how the data is distributed. Also the residuals shows unexpected features, indicating that external factors might occur

As the result, we will successively run ARIMA, SARIMA and SARIMAX model to further explore the data

3.1.2 Determine the parameters p,d,q

- Use the Augmented Dickey-Fuller (ADFuller) test to determine the d parameter.

```
def find_d(dataset):
    i = 0
    temp = dataset
    while True:
        result = adfuller(temp['Revenue'].dropna())
        if result[1] < 0.05:
            print(f"Number of difference: {i}")
            print('ADF Statistic: %f' % result[0])
            print('p-value: %f' % result[1])
            break
        else:
            temp = temp.diff().dropna()
            i = i+1

            if temp["Revenue"].isna().all():  # Prevent infinite loop
                print("All values became NaN after differencing. Check
                    data.")
                break
```

```
find_d(train)
✓ 0.9s

Number of difference: 1
ADF Statistic: -29.458019
p-value: 0.000000
```

It is ideal that differencing stops at 1, but that will not sufficient to clearly reduce seasonality

- Plot AutoCorrelation (ACF) and Partial AutoCorrelation (PACF) to determine the p and q parameters.

```
def findpq(i, dataset):
  if i == 1:
    X_traindiff = dataset.diff().dropna().dropna()
  if i == 2:
    X_traindiff = dataset.diff().dropna().diff().dropna().dropna()

  fig, axes = plt.subplots(1, 2, figsize=(14, 5))

  # ACF plot (for q selection)
  plot_acf(X_traindiff, lags = 15, ax=axes[0])
  axes[0].set_title("Autocorrelation Function (ACF)")

  # PACF plot (for p selection)
  plot_pacf(X_traindiff,lags = 15, ax=axes[1])
  axes[1].set_title("Partial Autocorrelation Function (PACF)")

  plt.show()
```

To run the ACF and PACF, we will do one more extra step to reset Date into index and use only target column 'Revenue' for the graph

```
train.set_index('Date', inplace = True)
test.set_index('Date', inplace = True)

findpq(1, train['Revenue'])
```

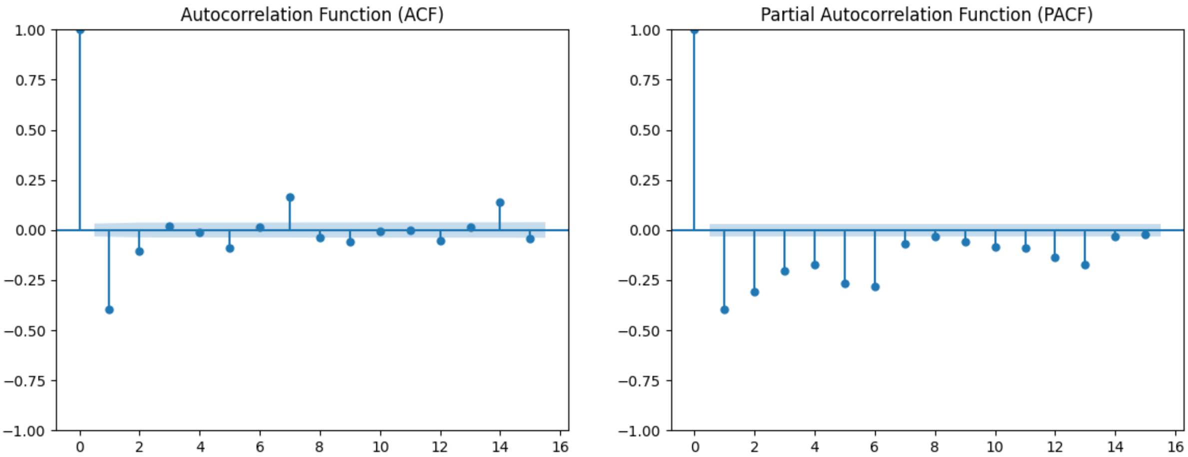


Figure 16: ACF and PACF

Take note that the Lag occure right before the sudden drop of value is the ideal lag for parameter, this case visually the optimal p and q are 1(7) and 1(7) respectively

- Use auto_arima to determine the optimal p, d, q parameters for the ARIMA model.

```
def detect(dataset):
    auto_model = auto_arima(dataset["Revenue"], seasonal=False, trace=True)
    print(auto_model.summary())
```

```
Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 94.542 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 3718
Model:               SARIMAX(2, 1, 2)   Log Likelihood              -57470.681
Date:                Sun, 23 Feb 2025   AIC                         114951.362
Time:                        11:15:54   BIC                         114982.466
Sample:                             0   HQIC                        114962.429
                               - 3718
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.0859      0.013     84.749      0.000       1.061       1.111
ar.L2         -0.1891      0.015    -12.939      0.000      -0.218      -0.160
ma.L1         -1.9325      0.009   -212.328      0.000      -1.950      -1.915
ma.L2          0.9394      0.009    107.173      0.000       0.922       0.957
sigma2      1.709e+12   6.01e-15   2.84e+26      0.000    1.71e+12    1.71e+12
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):            214538.82
Prob(Q):                              0.89   Prob(JB):                         0.00
Heteroskedasticity (H):               0.47   Skew:                             4.43
...
```

This is the demo run of how auto_arima can be used to automatically track the optimal parameter, it is also surprising that the parameter p, q clearly differ from visual intuition

- Use seasonal_compose to check for seasonality and auto_arima to optimize the parameters P,D,Q,m for the SARIMA model.

```
def detect_parameter(train_data, p, d, q):
    auto_arima(train_data['Revenue'], seasonal=True,
    p=p, d=d, q=q,  # We already know p,d,q
    P=None, D=None, Q=None,  # Let auto_arima find seasonal orders
    m=7,  # Time period
    trace=True)
```

Keep in mind that the parameter m = 7 indicates the time interval that the model will work on, which is each week, this remains unsure whether the data will rather follow a monthly/ seasonal or yearly trend, which sets the parameter to 30, 120 and 365. That will pose a huge burden on the runtime performance and require stronger computer processor cores.

```
Best model:  ARIMA(2,1,2)(2,0,2)[7]
Total fit time: 455.545 seconds
```

Tuned parameter (temporary):

$$(p, d, q, P, D, Q, m) = (2, 1, 2, 2, 0, 2, 7)$$

3.1.3 Run the ARIMA and SARIMA Models with given parameter

```
def arima(p,d,q, dataset, test_data):

  model = ARIMA(dataset['Revenue'], order=(p,d,q))
  model_fit = model.fit()
  print(model_fit.summary())

  start = 3031
  pred = model_fit.predict(start = start, end= start + len(test_data) - 1, typ = 'levels')

  residuals = test_data['Revenue'] - pred

  rmse = root_mean_squared_error(test_data['Revenue'], pred)
  mape = mean_absolute_percentage_error(test_data['Revenue'], pred)
  r2 = r2_score(test_data['Revenue'], pred)

  print(f"RMSE: {rmse:.2f}")
  print(f"MAPE: {mape * 100:.2f}%")
  print(f"R  : {r2:.4f}")
```

```
def sarimax(p, d, q, P, D, Q, s, train_data, test_data):

    # Fit SARIMAX model
    model = SARIMAX(train_data['Revenue'],
                    order=(p, d, q),
                    seasonal_order=(P, D, Q, s), mle_regression = False)
    model_fit = model.fit(disp=False)

    # Forecast for the length of test data
    start = 3031
    pred = model_fit.predict(start = start, end= start + len(test_data) - 1, typ = 'levels')


    # Ensure prediction and test data are aligned
    pred = pred[:len(test_data)]

    print(model_fit.summary())

    # Calculate evaluation metrics
    rmse = root_mean_squared_error(test_data['Revenue'], pred)
    mape = mean_absolute_percentage_error(test_data['Revenue'], pred)
    r2 = r2_score(test_data['Revenue'], pred)

    # Display metrics
    print(f"RMSE: {rmse:.2f}")
    print(f"MAPE: {mape * 100:.2f}%")
    print(f"R  : {r2:.4f}")

    # Plot predictions vs actual values
    plt.figure(figsize=(10, 5))
    plt.plot(train_data.index, train_data['Revenue'], label='Train Data')
    plt.plot(test_data.index, test_data['Revenue'], label='Actual Test Data')
    plt.plot(test_data.index, pred, label='Forecast', linestyle='--')
    plt.legend()
```

```
    plt.title('SARIMAX Forecast vs Actual')
    plt.show()

    return model_fit
```

One more time, keep in mind that the start = 3031 part in the Model code is obtained through a series of test - compare where I have to set a threshold R^2 for each step to find out the best start position. This part doesn't necessarily reflect the actual accuracy of the model but visually it does.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                 3718
Model:                 ARIMA(2, 1, 2)   Log Likelihood               -4954.417
Date:                Sun, 23 Feb 2025   AIC                           9918.835
Time:                        13:49:53   BIC                           9949.938
Sample:                             0   HQIC                          9929.901
                               - 3718
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.0860      0.012     90.209      0.000       1.062       1.110
ar.L2         -0.1894      0.013    -14.112      0.000      -0.216      -0.163
ma.L1         -1.9325      0.008   -227.921      0.000      -1.949      -1.916
ma.L2          0.9394      0.008    115.175      0.000       0.923       0.955
sigma2         0.8412      0.005    182.315      0.000       0.832       0.850
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):            214895.21
Prob(Q):                              0.90   Prob(JB):                         0.00
Heteroskedasticity (H):               0.47   Skew:                             4.43
Prob(H) (two-sided):                  0.00   Kurtosis:                        39.18
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
RMSE: 0.69
MAPE: 164.67%
R²: 0.1164
```

Figure 17: ARIMA model Result

3.1.4 Predict Sales for 2023

```
def sarimax_forecast(p, d, q, P, D, Q, s, train_data, test_data,
        future_periods=12):

    # Fit SARIMAX model
    model = SARIMAX(train_data['Revenue'],
                    order=(p, d, q),
                    seasonal_order=(P, D, Q, s),
                    mle_regression=False)
    model_fit = model.fit(disp=False)

    # Forecast for test data
    start = 3031
    pred_test = model_fit.predict(start=start, end=start + len(test_data) - 1,
        typ='levels')

    # Forecast for future periods
    future_index = pd.date_range(start=test_data.index[-1],
        periods=future_periods + 1, freq='M')[1:]
```

```
    pred_future = model_fit.predict(start=start + len(test_data), end=start +
        len(test_data) + future_periods - 1, typ='levels')
    #pred_future = scaler.inverse_transform([pred_future])


    # Calculate evaluation metrics on test data
    rmse = root_mean_squared_error(test_data['Revenue'], pred_test)
    mape = mean_absolute_percentage_error(test_data['Revenue'], pred_test)
    r2 = r2_score(test_data['Revenue'], pred_test)

    # Display metrics
    print(model_fit.summary())
    print(f"RMSE: {rmse:.2f}")
    print(f"MAPE: {mape * 100:.2f}%")
    print(f"R  : {r2:.4f}")

    # Plot predictions vs actual values
    plt.figure(figsize=(12, 6))
    plt.plot(train_data.index, train_data['Revenue'], label='Train Data')
    plt.plot(test_data.index, test_data['Revenue'], label='Actual Test Data')
    plt.plot(test_data.index, pred_test, label='Test Forecast', linestyle='--')
    plt.plot(future_index, pred_future, label='Future Forecast (2023)',
        linestyle='dotted', color='red')
    plt.legend()
    plt.title('SARIMAX Forecast vs Actual')
    plt.show()

    future = scaler.inverse_transform([pred_future])*1000
    f = pd.DataFrame({'Month': np.arange(1,13), 'Predicted Revenue':
        future.flatten()})

    return model_fit, f
```

I've added up both SARIMA and SARIMAX into 1 final Result as the Exogenous 'City', 'Segment' and 'Category' don't bring high predictive power but rather reduce the accuracy.

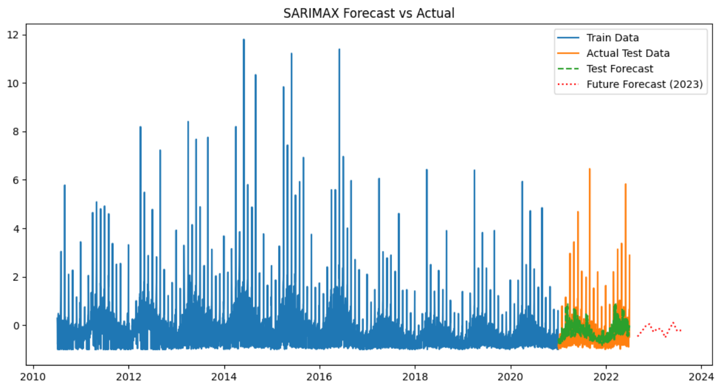


Figure 18: Visualized Prediction

Month	Predicted Revenue
1	435641.557910
2	498564.981042
3	572076.255626
4	601715.841162
5	496638.456666
6	540848.192542
7	533724.080097
8	421735.676016
9	530639.008547
10	620318.290729
11	508112.582527
12	518004.285986)

Figure 19: Detailed Prediction for 2023

3.2 The Recurrent Neural Network (RNN) and Long-Short Term Memory (LSTM) Model.

3.2.1 Preprocessing and defining parameters

- Create variables for training RNN and LSTM models, including `day_of_week`, `day_of_month`, and `month`.

- Convert date-related variables into sine and cosine representations to capture cyclic patterns. Additionally, apply the logarithm function to the `Revenue` variable to reduce the impact of large values.

- Use `MinMaxScaler` to normalize input variables `(Revenue_log, dow_sin, dow_cos, month_sin, month_cos, dom_sin, dom_cos)`. Split the data into two parts: Training set: Data before 2021.Test set: Data from 2021 and 2022.

- Set the `lookback window` to 30 days to create time-series samples for the model, ensuring it has enough past information to predict future values.

3.2.2 Model Training and Evaluation

- The LSTM model consists of 300 units, uses the ReLU activation function, the Adam optimizer, and optimizes based on MSE.

- Both models were trained on the training set with 38 epochs and a batch size of 20.

- The results of both models are shown in Table 1, where LSTM outperforms RNN in terms of R^2, indicating that it explains the variance of the dependent variable (`Revenue`) better than RNN.

- The MAPE (Mean Absolute Percentage Error) and RMSE (Root Mean Squared Error) values are lower, demonstrating smaller forecast errors and less fluctuation, which reflects the stability of the model.

After training the model on data from 2010 to 2020, the data from 2021–2022 was used for validation and hyperparameter tuning.

Once the optimal accuracy was achieved, the model was retrained on the entire dataset (2010–2022) to make predictions for 2023.

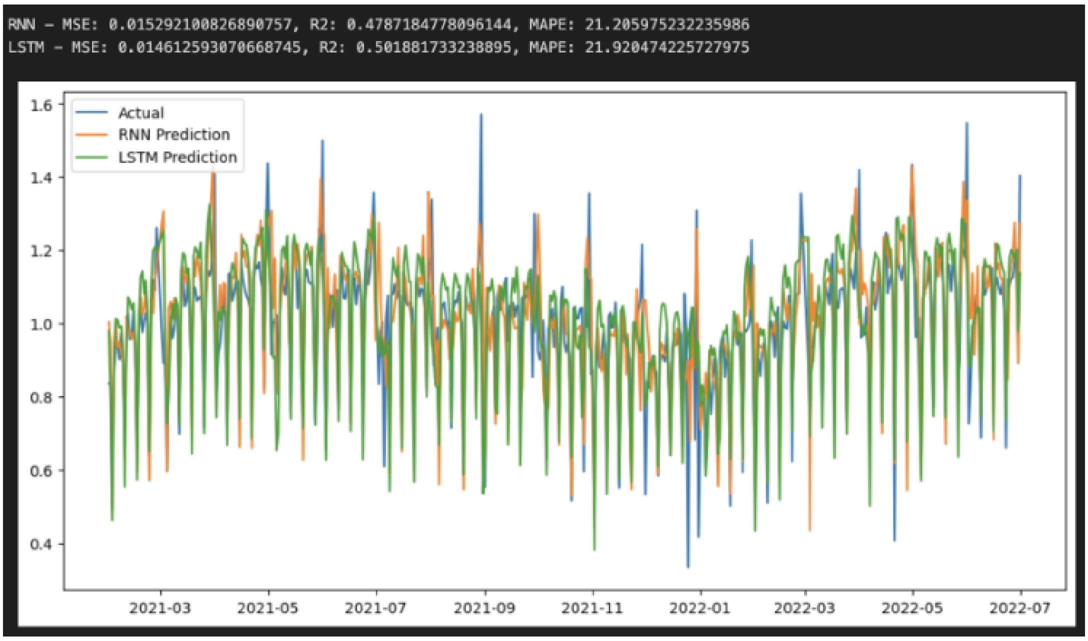


Figure 20: Testing and Training data

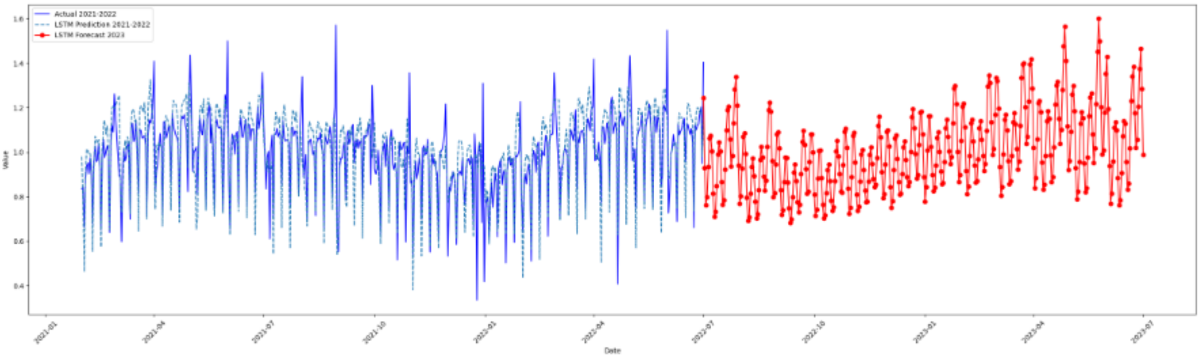


Figure 21: LSTM prediction for 2023

3.3 The Extreme Gradient Boosting (XGBoost) Model

3.3.1 Preprocessing

Data preprocessing for XGBoost includes:

- Converting the 'Date' column to datetime format and creating time-based features (Year, Month, Quarter, DayOfWeek).
- Creating important features such as ProfitMargin, RevenuePerUnit, and lag features to prevent data leakage.
- Encoding categorical variables using LabelEncoder.
- Selecting important features using a correlation matrix.
- Normalizing data with RobustScaler to reduce the impact of outliers.

3.3.2 Hyperparameter Tuning

After Running the RandomizedSearchCV to loop through predefined parameters, the best params are listed below:

$$
\mathbf{XG} = \begin{bmatrix} \text{n\_estimators} & 50 \\ \text{learning\_rate} & 0.07 \\ \text{max\_depth} & 1 \\ \text{subsample} & 0.6 \\ \text{colsample\_bytree} & 0.6 \\ \text{min\_child\_weight} & 15 \\ \text{gamma} & 1.0 \\ \text{reg\_lambda} & 2.0 \\ \text{reg\_alpha} & 0.5 \\ \text{random\_state} & 42 \end{bmatrix}
$$

Index:

- Number of trees (n_estimators = 50): Specifies the total number of boosting rounds.
- Learning rate (learning_rate = 0.07): Limits the depth of each tree to prevent overfitting.
- Tree depth (max_depth = 1): Limits the depth of each tree to prevent overfitting.
- Row sampling rate (subsample = 0.6): Randomly samples 60% of the rows for each tree, improving model generalization.
- Feature sampling rate (colsample_bytree = 0.6): Randomly selects 60% of features for each tree, reducing feature correlation.
- Minimum weight for node splitting (min_child _weight = 15): Prevents splitting on small variations, reducing overfitting.
- New split threshold (gamma = 1.0): Requires a higher gain threshold before creating new branches, promoting simpler trees.

3.3.3 Model Training and Testing

The XGBoost model was trained using the optimized hyperparameters.

Performance Evaluation: XGBoost achieved higher accuracy compared to ARIMA and LSTM, thanks to its ability to effectively capture non-linear patterns in the data.

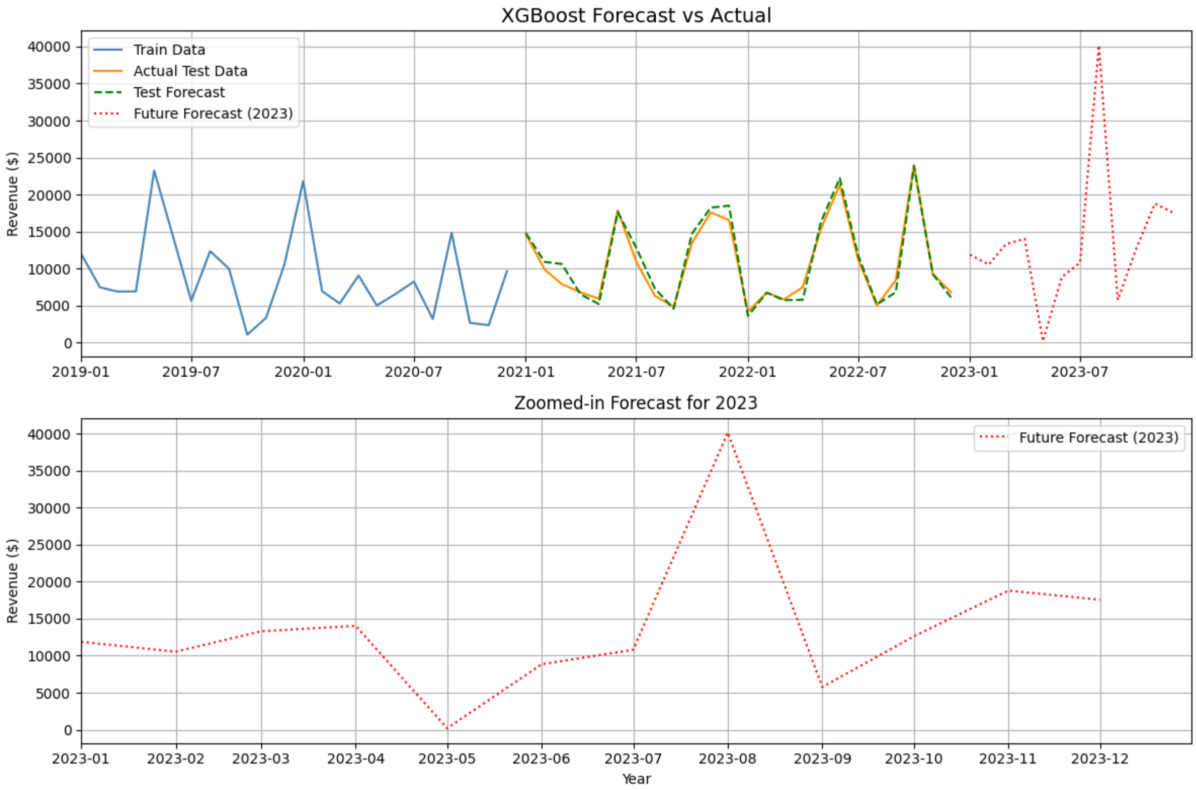


Figure 22: XGBoost Training - Testing - Forecasting

4 Results, Evaluation, and Strategy.

4.1 Model Evaluation and Prediction for 2023 Revenue

Along with the declining revenue trend in previous years, 2023 is projected to continue this downward trajectory, with estimated revenue falling within the range of:

4.2 Propose upcoming investment strategies

- Strengthen investment in Texas, Washington, Florida, and the Southeastern region (cities with historically high and stable revenue), including boosting production and launching new products.
- Continue investing in top-performing brands such as Maximus, Natura, and Aliqui, along with best-selling products and new offerings aligned with customer preferences.
- Expand and enhance online sales platforms while developing additional marketing campaigns to increase brand awareness.
- Address key factors affecting revenue in recent years (inflation, COVID-19, emerging trends, reduced clothing expenditure) and develop strategic measures to mitigate these challenges.

References

[1] Neusser, K. *Time Series Econometrics*. Springer, 2016.

[2] Hilmer, C. E., and Hilmer, M. J. *Practical Econometrics: Data Collection, Analysis, and Application*. McGraw-Hill, 2014.

[3] Auffarth, B. *Machine Learning for Time-Series with Python: Forecast, Predict, and Detect Anomalies with State-of-the-Art Machine Learning Methods*. Packt Publishing, 2021.

[4] YouTube Video: Forecasting with Machine Learning.

[5] YouTube Video: Time Series Analysis Tutorial.

[6] StatQuest Playlist: Time Series and Forecasting Techniques.

[7] Brownlee, J. "XGBoost for Time Series Forecasting." *Machine Learning Mastery*, 2024. Available at: https://machinelearningmastery.com/xgboost-for-time-series-forecasting/.

[8] Analytics Vidhya. "XGBoost for Time Series Forecasting." *Analytics Vidhya*, 2024. Available at: https://www.analyticsvidhya.com/blog/2024/01/xgboost-for-time-series-forecasting/.

[9] GeeksforGeeks. "XGBoost Overview." *GeeksforGeeks*, 2024. Available at: https://www.geeksforgeeks.org/xgboost/.

Model	Performance Metrics			Problem Involved	Future Improvement
	R2	MAPE	RMSE		
ARIMA	0.1071	163.61%	0.7	Sensitive to large datasets and has not fully optimized parameters due to high computational cost	Tune parameters further, explore data aggregation, and incorporate exogenous variables.
RNN	0.4787	21.2%	0.1236	Takes advantage of time-series but struggles to store long-term relationships. Often encounters the vanishing gradient problem, which can prevent the model from fully learning long-term trends in the data.	Adjust the number of neurons, activation functions, or add more RNN layers.
LSTM	0.5018	21.92%	0.1208	It overcomes RNNs drawback of gradually forgetting information thanks to the gating mechanism (forget, input, output). This helps the model retain long-term relationships better, leading to more favorable R^2 and RMSE metrics compared to RNN.	Apply regularization mechanisms (Dropout, L2, etc.) and adjust the architecture (multi-layer LSTM, Bi-LSTM) to enhance model generalization. Additionally, experiment with hybrid models.
XGBoost	0.9268	14.19%	0.1046	Multiple hyperparameter tuning iterations were required to prevent overfitting, as XGBoost does not inherently capture temporal relationships. Therefore, manual feature engineering, such as Lag Features, was necessary. Additionally, XGBoost is not optimized for long-term forecasting and is highly sensitive to hyperparameters.	Optimizing learning rate and max depth, along with feature selection, helps reduce noise. Combining XGBoost with other models to create a hybrid model can further improve forecasting accuracy..

Table 1: Summary of Model Evaluation