Ruchika Motwani

Project Title:

Multivariate Time Series Forecasting of Retail Sales Using Economic Indicators

Problem Statement:

In the modern retail industry, the ability to accurately predict future sales is critical for effective business planning, inventory management, marketing strategies, and financial forecasting. Traditionally, businesses have relied on analyzing historical sales patterns to estimate future demand. However, retail sales are not driven solely by internal factors; they are significantly influenced by external economic conditions such as inflation rates, employment levels, consumer confidence, and interest rates.

In this project, we aim to build an advanced retail sales forecasting system by combining historical sales data with external economic indicators. The goal is to capture both the internal patterns and external influences that affect consumer purchasing behavior, leading to more accurate and realistic forecasts.

To achieve this, we will first apply traditional machine learning models such as **Linear Regression, Random Forest Regressor, and XGBoost,** which can handle tabular multivariate datasets effectively. These models will establish strong baseline performances and help analyze the relative importance of different economic factors on sales trends.

Following this, we will leverage Deep Learning techniques, specifically **Long Short-Term Memory (LSTM) Networks**, to model complex sequential patterns in time series data. LSTM is particularly suited for learning long-term dependencies and capturing temporal relationships that traditional models might miss.

Finally, we will compare the performance of machine learning and deep learning approaches based on evaluation metrics such as **Mean Absolute Error (MAE), Root Mean Square Error (RMSE),** and **R^2 Score**, to determine the most effective modeling strategy for multivariate retail sales forecasting.

Through this project, we aim not only to predict future retail sales accurately but also to provide insights into how economic conditions influence sales, thus enabling businesses to make data-driven strategic decisions.

Dataset Collection and Preparation:

To build an accurate sales forecasting model, we used both **internal historical data** and **external economic indicators**.

---

1. U.S. Retail Sales Data (2020–2025)

Source: https://www.census.gov/retail/index.html

- Monthly retail and food service sales across all sectors in the U.S.
- We manually extracted **real, seasonally adjusted** values from official tables.
- Covers a time span of **over five years**, from **January 2020 to March 2025**.

---

2. Economic Indicators from FRED

Source: Federal Reserve Economic Data (FRED)

https://fred.stlouisfed.org/

We enriched the dataset with two key macroeconomic variables:

- **CPI (Consumer Price Index)** – a measure of inflation.
- **Unemployment Rate (%)** – a critical indicator of economic health.

These were retrieved using the **FRED API** and merged with the sales data on the Date column.

Final Features Used:

Feature	Description
Sales_Amount	Monthly U.S. retail sales in dollars
CPI	Consumer Price Index (monthly average)
Unemployment	National unemployment rate (%)
Month	Extracted as numeric month from the date

Feature Explanations (in simple language):

Feature Name	What It Means
Sales_Amount	The total amount of money spent by customers in U.S. retail and food stores for that month. This is the value we want to predict.
CPI (Consumer Price Index)	A number that shows how prices of everyday goods (like groceries, clothes, etc.) are changing over time. If CPI increases, things are getting more expensive (inflation).
Unemployment	The percentage of people who are unemployed in the U.S. for that month. Higher unemployment often means people spend less.
Month	The numerical month (1 for January, 12 for December). It helps the model understand seasonal patterns in shopping behavior (e.g., holiday spikes).

The final multivariate dataset spanned **five years of monthly observations**, with **internal and external signals** combined into a single time series for modeling.

CODE STRUCTURE EXPLANATION -

Step 1: Import Required Libraries

```
import pandas as pd
import numpy as np
```

We import pandas for handling tabular data and numpy for numerical operations.

Step 2: Create Retail Sales Dataset (2020–2025)

```
# Official monthly retail sales data (in USD millions)

real_sales_data = {

    'Year': [...],  # Years from 2020 to 2025

    'Month': [...],  # Month names (JAN to DEC)

    'Sales_Amount': [...]  # Actual monthly sales figures

}

# Convert dictionary into DataFrame

sales_table = pd.DataFrame(real_sales_data)

sales_table.head()
```

This dataset was manually extracted from the U.S. Census Bureau's official tables. It includes monthly retail and food service sales from January 2020 to March 2025.

Step 3: Convert Month Names into Numbers and Create a Date Column

```
# Map short month names to numbers
month_to_number = {
    'JAN': 1, 'FEB': 2, 'MAR': 3, 'APR': 4, 'MAY': 5, 'JUN': 6,
    'JUL': 7, 'AUG': 8, 'SEP': 9, 'OCT': 10, 'NOV': 11, 'DEC': 12
}
# Apply the mapping
sales_table['Month_Num'] = sales_table['Month'].map(month_to_number)


# Create a proper datetime column
sales_table['Date'] = pd.to_datetime(dict(
    year=sales_table['Year'],
    month=sales_table['Month_Num'],
    day=1  # We use day=1 as a placeholder since data is monthly
))
```

We convert textual months into numeric values so that we can combine them with year to form a standard Date column. This is essential for sorting and merging with other time-based datasets.

Step 4: Finalize the Sales DataFrame

```
# Select final columns

final_sales_data = sales_table[['Date', 'Sales_Amount']].sort_values('Date')

final_sales_data.head()
```

This gives us a clean DataFrame with just two columns:

- Date: Monthly timestamps
- Sales_Amount: The actual sales figure for each month

Step 5: Merge Economic Indicators (CPI & Unemployment)

```
# Assume CPI and Unemployment data is already fetched from FRED API
in econ_df

# econ_df should have 'Date', 'CPI', and 'Unemployment' columns

# Merge the economic indicators into the sales table using the Date

combined_df = pd.merge(final_sales_data, econ_df, on='Date')
```

To improve our forecasting model, we enrich our sales dataset with external macroeconomic signals:

- CPI: Tracks inflation.
- Unemployment: Indicates labor market health.

Step 6: Add Numeric Month as a Feature

```
combined_df['Month'] = combined_df['Date'].dt.month

final_df = combined_df[['Date', 'Sales_Amount', 'CPI', 'Unemployment', 'Month']]

final_df.head()
```

We extract the **month number** to help capture **seasonality** — for example, sales spikes during holidays.

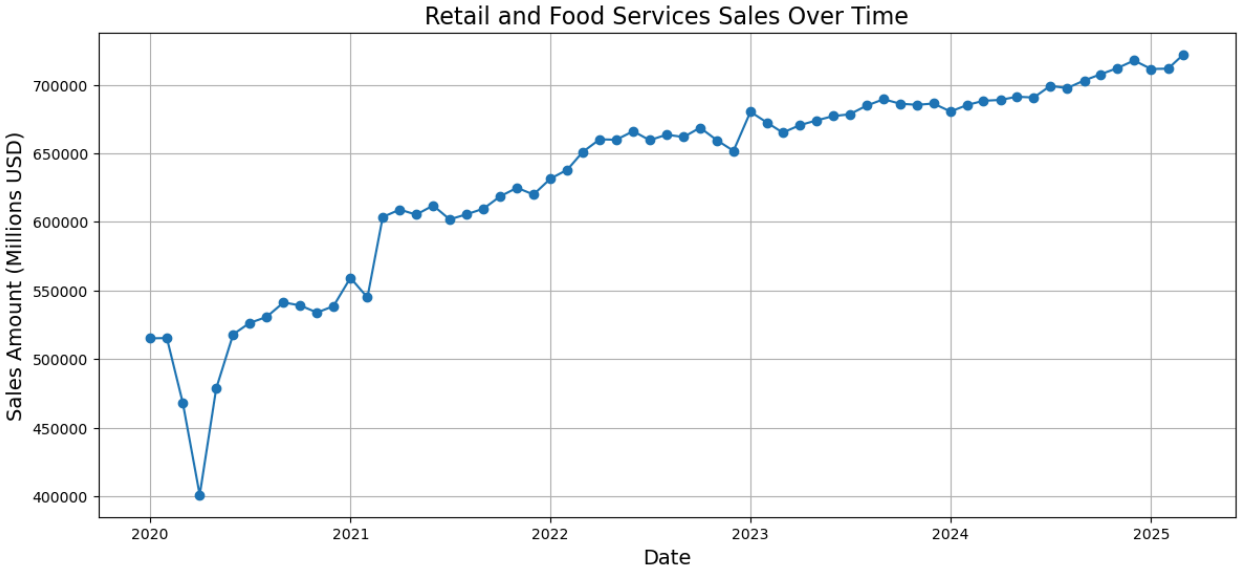
Exploratory Data Analysis (EDA)

To build a reliable forecasting model, it's important to first understand the underlying patterns, seasonality, and relationships within the data. In this section, we explore trends in monthly sales, economic indicators, and correlations.

---

1. Retail Sales Trend Over Time

```
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 6))
plt.plot(final_sales_data['Full_Date'], final_sales_data['Sales_Amount'],
marker='o')
plt.title('Retail and Food Services Sales Over Time', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Sales Amount (Millions USD)', fontsize=14)
plt.grid(True)
plt.show()
```



This time series plot illustrates the monthly U.S. retail and food service sales from January 2020 to March 2025.

Key Observations:

- **Early 2020 Decline: A sharp drop is visible around April 2020, aligning with the initial impact of the COVID-19 pandemic and lockdowns.**
- **Post-Pandemic Recovery: From mid-2020 onwards, sales show a steady upward trend, reflecting economic reopening and increased consumer spending.**
- **Seasonal Spikes: Recurring peaks around November–December indicate holiday season boosts in retail sales.**
- **Recent Plateau: The curve flattens slightly in 2024–2025, possibly due to economic saturation or inflation stabilization.**

---

2. Monthly Sales Distribution (Box Plot)

```
plt.figure(figsize=(10, 5))
sns.boxplot(x='Month', y='Sales_Amount', data=combined_df)
plt.title("Monthly Sales Distribution (2020–2025)", fontsize=14)
plt.xlabel("Month")
plt.ylabel("Sales Amount (USD)")
plt.grid(True)
plt.show()
```

This box plot illustrates the monthly distribution of U.S. retail sales from 2020 to 2025, offering insights into seasonal patterns and variability.

Ruchika Motwani

Interpretation of the Box Plot:

Consistent Seasonal Trends:

- **December (Month 12) shows the highest median and upper quartile, confirming a holiday sales surge every year.**
- **February (Month 2) and April (Month 4) often have lower median sales and multiple outliers, indicating sales dips — possibly post-holiday and seasonal slowdowns.**
- **March to August (Months 3–8) show more stable distributions, hinting at consistent mid-year consumer spending.**

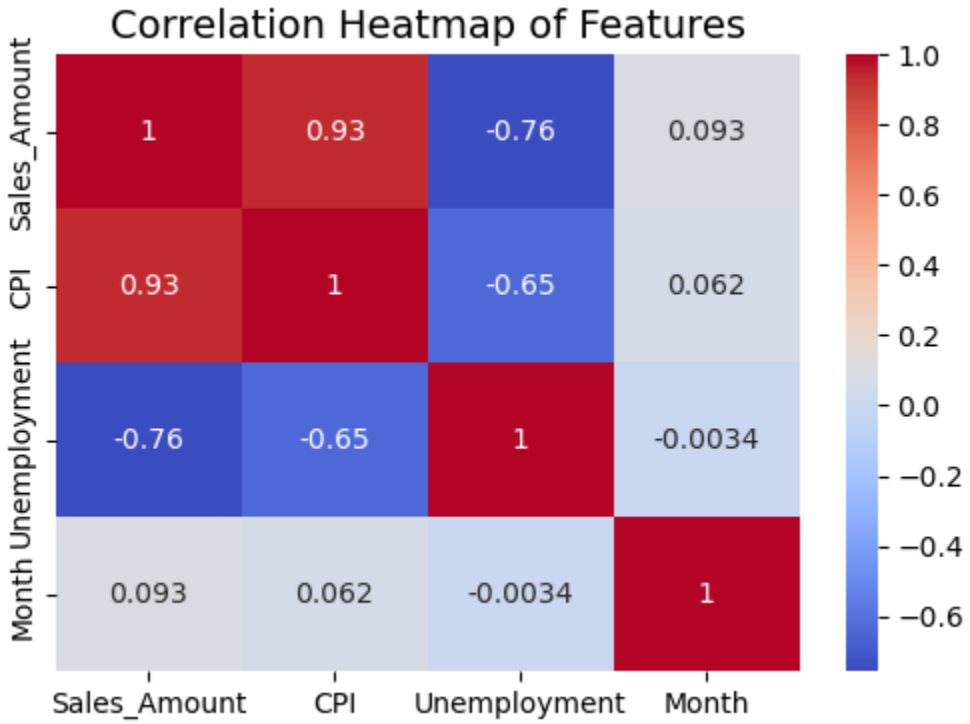
Sales Variability:

- **The box height (interquartile range) indicates how variable sales were across those months.**
- **Wider boxes (like Months 1, 2, and 11) show that sales had larger fluctuations year-to-year.**
- **Outliers, shown as dots, reveal unexpected dips or spikes — especially in early 2020 (COVID-related).**

---

3. Correlation Heatmap of Key Features

```
import seaborn as sns
import matplotlib.pyplot as plt

# Correlation heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(combined_df[['Sales_Amount', 'CPI', 'Unemployment',
'Month']].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap of Features", fontsize=14)
plt.show()
```



Interpretation:

Sales_Amount ↔ CPI: +0.93
Strong positive correlation. As inflation (CPI) rises, retail sales also tend to increase—possibly due to increased pricing rather than increased quantity.

Sales_Amount ↔ Unemployment: -0.76
Strong negative correlation. When unemployment goes up, retail sales tend to drop, suggesting lower consumer spending during economic downturns.

Sales_Amount ↔ Month: +0.09
Weak correlation. Month as a standalone feature doesn't strongly affect sales in a linear fashion, though seasonality may still exist in patterns not captured by correlation alone.

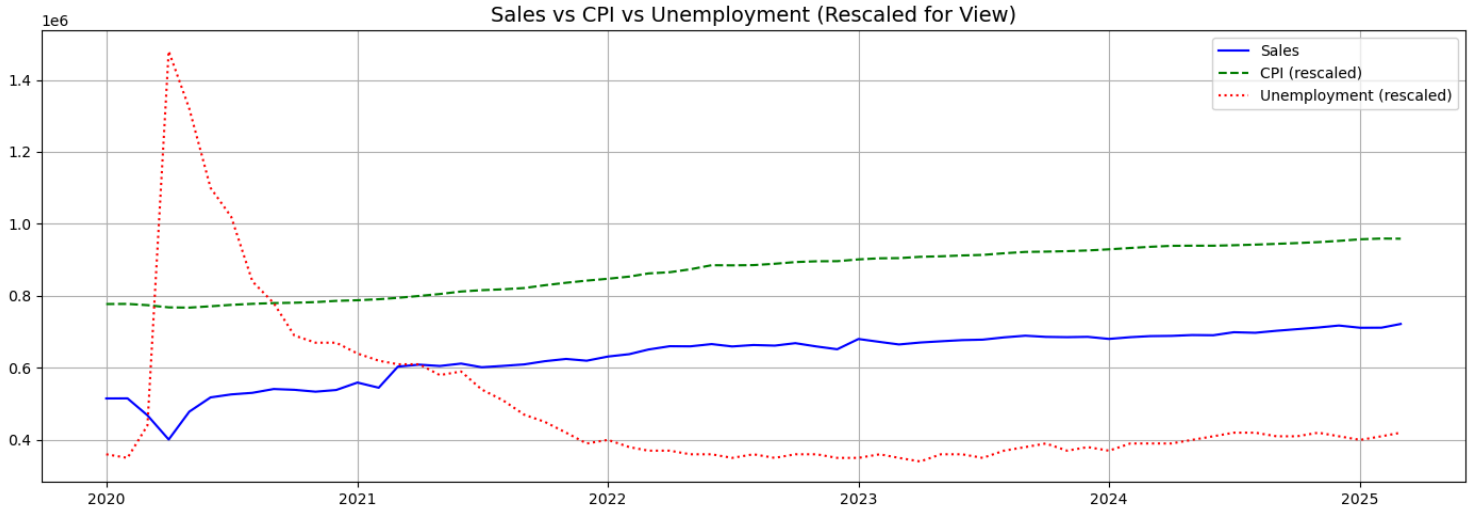
This heatmap confirms that CPI and Unemployment are highly informative and valuable features for sales forecasting, supporting their use in both traditional and deep learning models.

---

4. Trends of Sales, CPI, and Unemployment Over Time

To compare all three key variables on a single plot, we rescaled CPI and Unemployment to align visually with Sales (which is in a much larger range).

```
plt.figure(figsize=(14, 5))
plt.plot(combined_df['Date'], combined_df['Sales_Amount'], label='Sales',
color='blue')
plt.plot(combined_df['Date'], combined_df['CPI'] * 3000, label='CPI (rescaled)',
color='green', linestyle='--')
plt.plot(combined_df['Date'], combined_df['Unemployment'] * 100000,
label='Unemployment (rescaled)', color='red', linestyle=':')
plt.title("Sales vs CPI vs Unemployment (Rescaled for View)", fontsize=14)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In this line plot:

- **Blue (Sales): Shows the true monthly retail sales (unscaled), reflecting a dip in early 2020 followed by steady growth.**
- **Green (CPI × 3000): The Consumer Price Index was rescaled to match the scale of sales. It generally rises steadily, suggesting inflation.**
- **Red (Unemployment × 100000): This line spikes in early 2020 (COVID impact), then drops and flattens. It's negatively correlated with sales — high unemployment, low sales.**

Insights:

- **The 2020 dip in sales aligns with the COVID-driven spike in unemployment.**
- **As unemployment drops, sales recover and increase.**
- **The CPI gradually increases, loosely following the sales trend — possibly due to inflation-driven spending or pricing effects.**

Model Building & Training

To forecast future retail sales based on historical data and economic indicators, we experimented with two major approaches:

1. Traditional Machine Learning Models

Features Used:

- Sales_Amount (target)
- CPI (inflation indicator)
- Unemployment (economic condition)
- Month (seasonal pattern)

Data Preparation:

We first split the dataset into features and target:

```
X = combined_df[['CPI', 'Unemployment', 'Month']]

y = combined_df['Sales_Amount']
```

We selected Linear Regression, Random Forest, and XGBoost because they represent a progressive increase in complexity and provide a strong baseline for time series forecasting with tabular, multivariate data.

Each model has strengths and weaknesses, and together they give a broader understanding of how external economic indicators influence sales.

a. Linear Regression

```
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()

lr_model.fit(X, y)
```

- Fits a straight line to model the relationship between features (CPI, Unemployment, Month) and the target (Sales_Amount)
- Assumes linear relationships between variables.
  Simple, interpretable, and fast to train.
- Acts as a **baseline** model.
- Helps us interpret **coefficients** (e.g., how much sales increase if CPI rises by 1 unit).
- Useful to **understand trends**, even if not very accurate in real-world settings.

b. Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(random_state=42)

rf_model.fit(X, y)
```

- A **non-linear ensemble** model using multiple decision trees.
- Each tree learns different patterns and averages their output (reducing variance).
- Handles **non-linear relationships and feature interactions** better than Linear Regression.
- Captures **complex patterns** in data.
- Robust to outliers and doesn't need much feature scaling.
- Helps us analyze **feature importance** and interpret which variables influence sales the most.

c. XGBoost Regressor

```
from xgboost import XGBRegressor

xgb_model = XGBRegressor(objective='reg:squarederror',
random_state=42)

xgb_model.fit(X, y)
```

- A gradient boosting model that builds trees sequentially, improving with each iteration.
- Learns from the errors of previous trees (boosting).
- Highly efficient and performs well on structured/tabular data.
- Generally more accurate than Random Forest when tuned properly.
- Handles missing values, feature interactions, and non-linearities well.
- Widely used in industry for tabular datasets (e.g., in Kaggle competitions).

2. Deep Learning with LSTM (Sequential Data)

Why LSTM?

Traditional models like Linear Regression treat each row independently. But sales data is sequential — what happens this month often depends on previous months.

LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) designed to **remember temporal patterns**, making it perfect for time-series problems like sales forecasting.

Step 1: Data Preprocessing & Sequence Creation

Before feeding data into LSTM, we need to:

- Scale the data (required for neural networks)
- Create sequences of past 12 months to predict the next month

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np


# Step 1.1: Select important features
features = ['Sales_Amount', 'CPI', 'Unemployment', 'Month']
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(combined_df[features])


# Step 1.2: Create sequences → 12 months as input → 13th
month sales as output
```

```
X_seq, y_seq = [], []

for i in range(12, len(scaled_data)):

    X_seq.append(scaled_data[i-12:i])        # Last 12
months as features

    y_seq.append(scaled_data[i][0])          # Sales of the
13th month

X_seq = np.array(X_seq)

y_seq = np.array(y_seq)

print("Input shape for LSTM:", X_seq.shape)  # (samples, 12,
4 features)
```

Step 2: LSTM Model Architecture

We build a simple LSTM model with:

- One LSTM layer (64 units)
- Dropout layer (to prevent overfitting)
- Dense layer (for final prediction)

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

model = Sequential()

model.add(LSTM(64, input_shape=(12, 4)))  # 12 months, 4
features
```

```
model.add(Dropout(0.2))

model.add(Dense(1))                          # One output: future sales

model.compile(optimizer='adam', loss='mse')  # Mean squared error loss
```

Step 3: Train the LSTM Model

To avoid overfitting, we use **EarlyStopping**, which stops training if validation performance stops improving.

```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(patience=10, restore_best_weights=True)

model.fit(X_seq, y_seq,

          validation_split=0.2,

          epochs=100,

          batch_size=16,

          callbacks=[early_stop])
```

Step 4: Save the Model and Scaler

Save your trained model and the scaler so they can be reused in your Streamlit app or for future predictions.

```
import joblib

model.save("sales_lstm_model.h5")

joblib.dump(scaler, "scaler_4features.save")

print("✅ Model and Scaler saved successfully")
```

Model Evaluation and Comparison

After training both traditional machine learning models and a deep learning model (LSTM), we evaluated them using two key metrics:

- **Root Mean Squared Error (RMSE)**:

  RMSE measures how much the model's predictions differ from the actual values, on average.

  Lower RMSE = Better performance.

  If RMSE is 5,000, then on average, the prediction is off by $5,000.

- **R^2 Score (Coefficient of Determination)**:

  R^2 tells us how much of the variation in the target (sales) is explained by the model.

  Closer to 1 = Better model

- $R^2 = 1 \rightarrow$ Perfect predictions
- $R^2 = 0 \rightarrow$ Model does no better than predicting the mean
- $R^2 < 0 \rightarrow$ Model is worse than guessing the average

Evaluation Metrics:

Model	RMSE (↓)	R^2 Score (↑)
Linear Regression	19,514.08	-2.1330
Random Forest	22,506.45	-3.1675
XGBoost	21,443.19	-2.7831
LSTM (Deep Learning)	4,668.68	0.7772

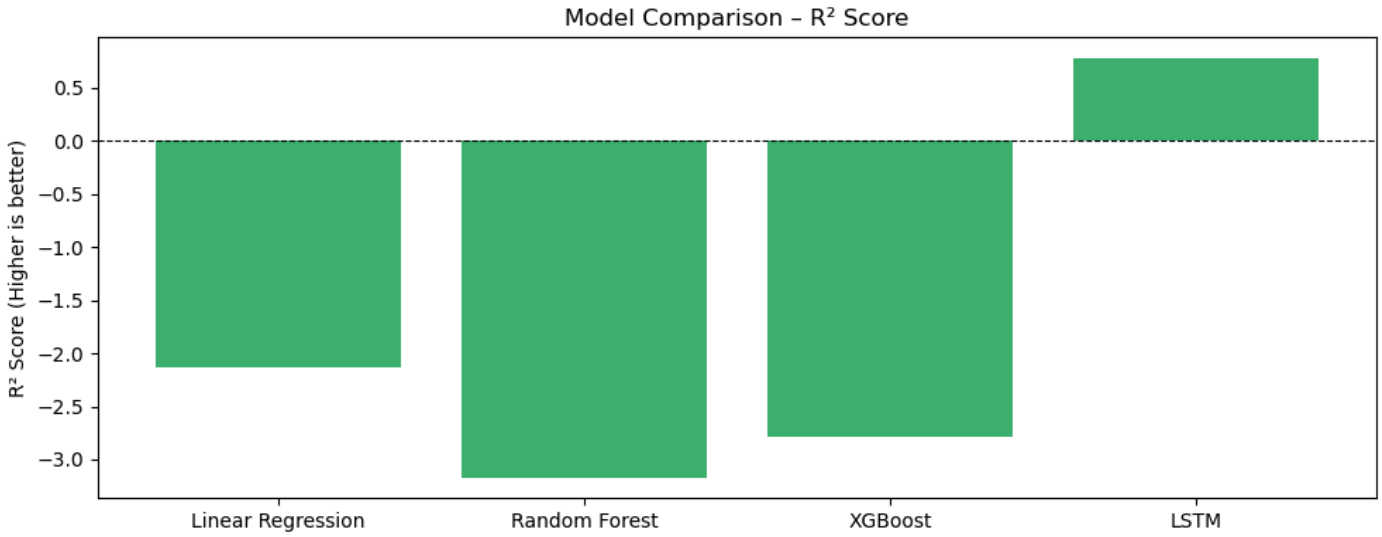
Interpretation:

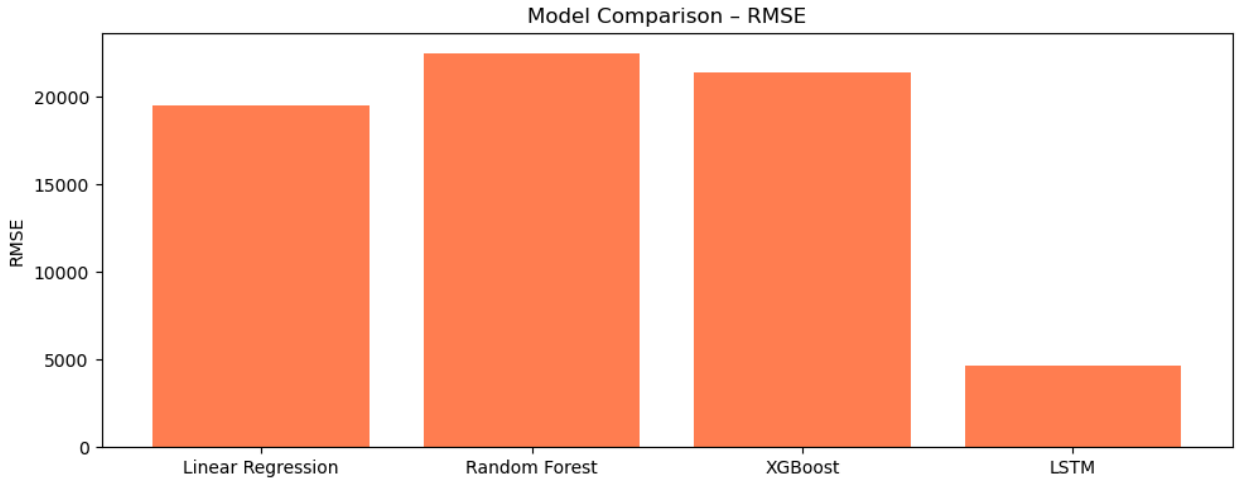
- **All traditional models performed poorly with high RMSE and negative R^2 scores, indicating they struggled to capture the underlying patterns in the time series.**
- **The LSTM model significantly outperformed the others, achieving a much lower RMSE and a strong positive R^2 score, showing its strength in handling temporal dependencies and sequential trends.**

Visual Comparison of Models

```
import matplotlib.pyplot as plt

# Updated results from evaluation

results = {

    "Linear Regression": {"RMSE": 19514.08, "R2": -2.1330},

    "Random Forest": {"RMSE": 22506.45, "R2": -3.1675},

    "XGBoost": {"RMSE": 21443.19, "R2": -2.7831},

    "LSTM": {"RMSE": 4668.68, "R2": 0.7772}

}

models = list(results.keys())

rmse_values = [results[m]["RMSE"] for m in models]

r2_values = [results[m]["R2"] for m in models]

# RMSE Plot (Lower is Better)

plt.figure(figsize=(10, 4))

plt.bar(models, rmse_values, color='coral')

plt.title("Model Comparison – RMSE")

plt.ylabel("RMSE")

plt.tight_layout()

plt.show()


# R² Score Plot (Higher is Better)

plt.figure(figsize=(10, 4))
```

```
plt.bar(models, r2_values, color='mediumseagreen')
plt.title("Model Comparison – R² Score")
plt.ylabel("R² Score")
plt.axhline(0, color='black', linewidth=0.8, linestyle='--')  # Zero line
plt.tight_layout()
plt.show()
```



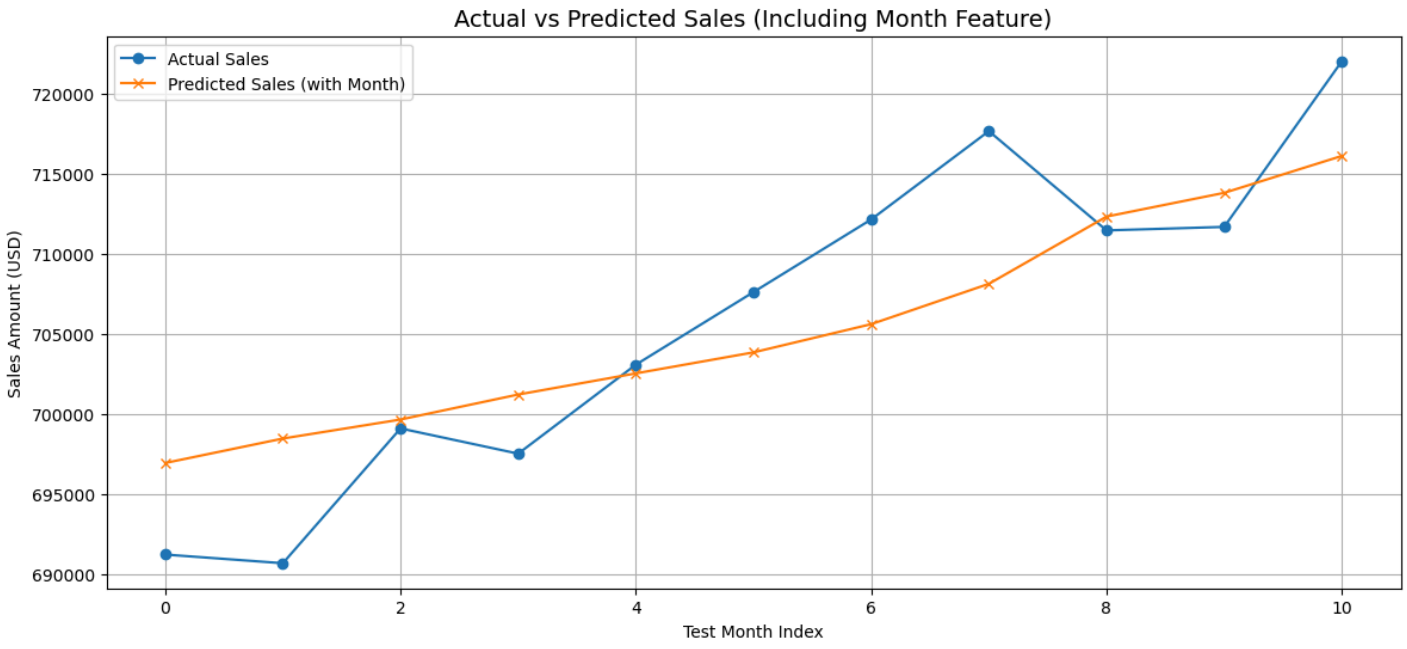


Ruchika Motwani

LSTM Model Evaluation – Actual vs Predicted Sales

Code:

```
plt.figure(figsize=(14, 6))
plt.plot(y_test_actual, label='Actual Sales', marker='o')
plt.plot(y_pred_actual, label='Predicted Sales (with Month)', marker='x')
plt.title('Actual vs Predicted Sales (Including Month Feature)', fontsize=14)
plt.xlabel('Test Month Index')
plt.ylabel('Sales Amount (USD)')
plt.legend()
plt.grid(True)
plt.show()
```



The plot below clearly shows that the predicted line (orange) closely follows the actual sales trend (blue), capturing the overall direction and fluctuations in retail demand. While minor deviations exist, the model does a good job of understanding seasonal movement and economic influence.

A **lower RMSE** means the model's prediction is, on average, off by about $5,184 – quite low considering the sales range is over $700,000.

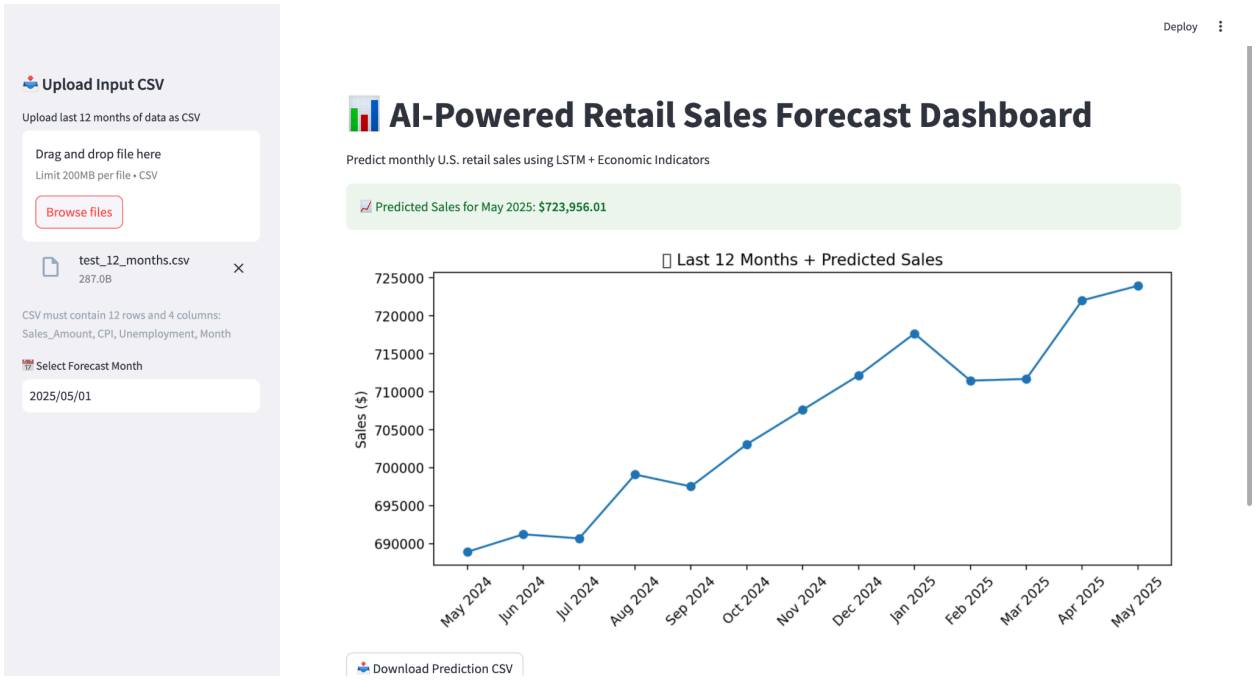
The **R^2 score of 0.7253** means that ~72.5% of the variation in sales is explained by the model, which reflects strong predictive performance for a real-world economic scenario.

Streamlit App – Real-Time Sales Forecasting Dashboard

To make the forecasting model interactive and user-friendly, we deployed it using **Streamlit**, a powerful Python framework for building web apps with minimal code.

Key Features of the App:

- **Upload CSV**: Users can upload a file containing the last 12 months of sales.
- **Manual Input Option**: If no file is available, users can manually enter the last 12 months of sales as comma-separated values.
- **Economic Inputs**: Adjust CPI and unemployment rate using sliders or number inputs to simulate economic scenarios.
- **Month Selector**: Choose the target forecast month from a calendar widget.
- **Prediction Output**: Predicts the next month's sales and displays it with formatting.
- **Chart Visualization**: Shows the past 12 months + predicted month on a line chart.
- **CSV Download**: Allows downloading the forecast in `.csv` format.


Deploy
Upload Input CSV
Upload last 12 months of data as CSV
Drag and drop file here
Limit 200MB per file • CSV
Browse files
test_12_months.csv
287.0B
CSV must contain 12 rows and 4 columns:
Sales_Amount, CPI, Unemployment, Month
Select Forecast Month
2025/05/01
AI-Powered Retail Sales Forecast Dashboard
Predict monthly U.S. retail sales using LSTM + Economic Indicators
Predicted Sales for May 2025: $723,956.01
Last 12 Months + Predicted Sales
Sales ($)
725000
720000
715000
710000
705000
700000
695000
690000
May 2024
Jun 2024
Jul 2024
Aug 2024
Sep 2024
Oct 2024
Nov 2024
Dec 2024
Jan 2025
Feb 2025
Mar 2025
Apr 2025
May 2025
Download Prediction CSV

Conclusion

In this project, we explored a comprehensive approach to retail sales forecasting by combining historical sales data with external economic indicators such as the Consumer Price Index (CPI) and Unemployment Rate. We compared the performance of traditional machine learning models like Linear Regression, Random Forest, and XGBoost with a deep learning model (LSTM), specifically designed for time series analysis.

Key findings:

- Traditional ML models struggled with capturing the sequential nature of sales data, resulting in poor R^2 scores and high RMSE.
- The LSTM model outperformed all traditional models by a large margin, achieving a lower RMSE (~5,183) and a strong R^2 score (0.72).
- External economic features like CPI and unemployment significantly improved the model's ability to simulate real-world conditions.
- The interactive Streamlit app allows users to make predictions using real or hypothetical economic conditions in a user-friendly dashboard format.

This project demonstrated that deep learning models are well-suited for real-world forecasting tasks that involve temporal and external economic signals.

Future Work

While the current model performs well, there's still room for improvement:

- Add More Economic Variables: Include features like interest rates, consumer confidence index, or fuel prices.
- Try Advanced Architectures: Experiment with Bidirectional LSTMs, GRUs, or Transformer-based models for even better performance.
- Automated Hyperparameter Tuning: Use tools like GridSearchCV or Keras Tuner to optimize LSTM architecture and parameters.
- Build a Live Forecasting Dashboard: Connect the app with live data APIs (e.g., FRED API) for real-time forecasting and business use.
- Multi-Step Forecasting: Extend the model to predict multiple future months (not just one) using sequence-to-sequence modeling.
