CS 5100 Final Report
ChainOptimizer - AI-Enhanced Supply Chain Optimization Team 19

Ajay Kumar Reddy Inavolu, Neeraj Srivatsav Sanne,

Karthikeya Reddy Kolanu, Venkata Satya Naga Sai Karthik Koduru

December 12, 2023

1 Abstract

In this project, we introduce "ChainOptimizer," an AI-driven tool designed to enhance supply chain inventory management. Using machine learning algorithms, we enable predictive demand forecasting, which significantly reduces overstocking and understocking issues. Our work focuses on models we developed using Random Forest, XGBoost, and Neural Networks, tailored to predict next week's demand and focusing on making our predictions as accurate as possible. We analyzed a dataset that included product details and sales data. Our approach involved transforming complex data into a format suitable for machine learning models. Our approach also refines order processing by calculating optimal batch sizes and order frequencies, ensuring efficient part availability. This results in notable operational cost savings, smoother manufacturing processes, and improved production efficiency and customer satisfaction.

2 Introduction

In the world of supply chain management, accurately predicting product demand is as a key challenge, vital to operational efficiency. This is where our project, "ChainOptimizer," steps in. Aimed at transforming the way businesses approach inventory management, our tool leverages the power of AI to forecast weekly product orders with good accuracy.

The aim of our project lies in the recognition of a common struggle in the industry: the balancing act between overstocking and understocking. Traditional methods of demand forecasting often fall short, leading to increased costs and missed opportunities. Our goal with ChainOptimizer is to offer a solution that not only addresses this imbalance but also paves the way for a more data-driven, responsive approach to inventory management.

Our journey began with an extensive data exploration phase. We did a lot of research ad finalized on a dataset that included detailed sales transactions, product information, and customer feedback. This data forms the key of our approach, allowing us to train our models with real-world scenarios and challenges. However, raw data in itself is not enough for the algorithms we planned to use. We started on a detailed data preprocessing journey, transforming complex categorical variables into a numerical format that our models could interpret. This process, is like translating diverse languages into a single, comprehensible dialect for our AI, was crucial for the success of our project.

Our choice of models was based on the pros and cons weightage balance. We explored the Decision Tree Regression, XGBoost to capture the non-linear relations of our data and, most importantly, used Deep Learning with a Neural Network. This variety in our modeling approach ensured that we covered a broad spectrum of analytical perspectives.

The true measure of our models' effectiveness came through rigorous evaluation using various metrics, such as Mean Squared Error and R-squared. These metrics helped us in assessing how well our models could work in real life application. But the real aim of our project was not just in building models; it was about applying these models to predict real-world order quantities. ChainOptimizer is more than just a predictive tool; it's a step towards a more data-driven and customer-centric future in supply chain management. Our work shows how AI can be used to tackle real-world challenges, bringing efficiency and foresight into the heart of business operations.

3 Related Work

In supply chain optimization, significant research done by researchers has been used in the development of our ChainOptimizer project. Demand forecasting, a key focus in recent literature, has seen a shift from traditional methods like ARIMA [1] towards more advanced machine learning techniques. Algorithms such as Random Forest and XGBoost are noted for their ability to capture complex demand patterns [2]. Neural Networks have also garnered attention for handling complex, large-scale data in supply chain contexts [3]. Their application is the base for our use of Deep Learning in ChainOptimizer for data interpretation.

Reinforcement learning offers a dynamic approach to inventory management, aligning with our goal of adapting to varying demand [4]. This concept is central to our ChainOptimizer project. The usage of AI and Big Data has opened new paths for supply chain insights [5]. Our approach, particularly in handling and processing our dataset, is inspired by this perspective.

Sustainability in supply chains, a growing focus area, is being influenced by AI applications. Our consideration of sustainability in ChainOptimizer is inspired by such research since overstocking can lead to wastage of resources and money. Risk management through AI, especially in handling supply chain disruptions, is crucial. We have adapted predictive models for risk management to approach demand forecasting uncertainties.

Lastly, the integration of cloud computing in AI-driven supply chains highlights how scalable computing resources can enhance AI model performance. This aspect is a cornerstone in the development of ChainOptimizer as we used Google Colab for extensive training of our models.

4 Method

4.1 Data Preprocessing and Cleaning

Our methodology began with the important step of data preprocessing and cleaning, essential for ensuring the reliability of our models. We sourced our dataset from Kaggle. This dataset, contained sales transactions, product details, and customer feedback, providing a complete view of supply chain operations. This step was crucial to accurately represent the nature of supply chain dynamics.

Next, the data went through an aggregation process. This involved consolidating multiple data files into a singular dataset, enabling a unified analysis. The next crucial step was cleaning and standardizing the data. This part was focused to enhance the data quality by rectifying inconsistencies and handling missing or outlier values.

1. **Handling Missing Values**: We implemented a strategy to manage missing data points. For numerical variables, missing values were replaced with the median value of that variable, represented as median(X). For categorical variables, we used the mode, denoted as mode(X).

2. **Outlier Detection and Treatment**: Outliers were identified using the Interquartile Range (IQR) method, where values beyond 1.5 times the IQR from the first and third quartiles were considered outliers. IQR = Q3 - Q1 Outliers = X:X $<$ Q1 - 1.5 $\times$ IQR or X $>$ Q3 + 1.5 $\times$ IQR

3. Timestamp Standardization: Timestamps were standardized to a consistent format, YYYY-MM-DD HH:MM:SS, to allow temporal analysis and comparison.

4. **Data Consolidation**: Finally, we merged the various aspects of the dataset, such as sales, product, and customer information, using unique identifiers like transaction IDs and item IDs.

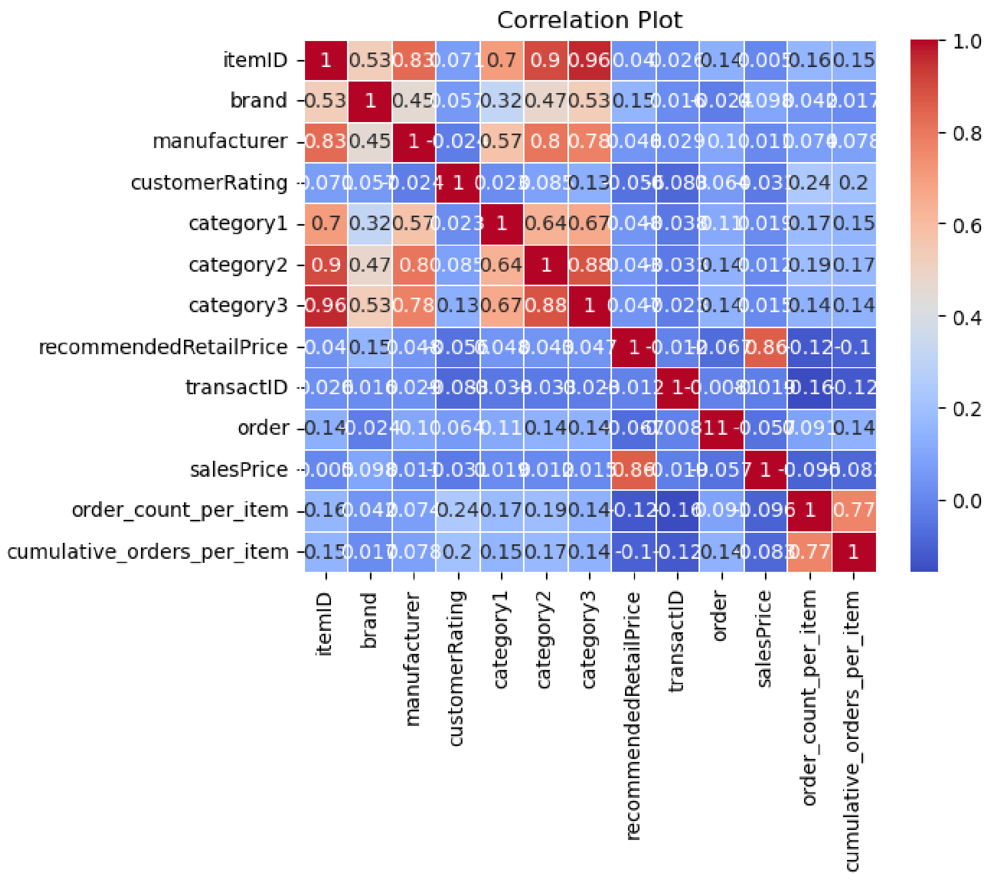
Through these preprocessing steps, we ensured the dataset's integrity, making it suitable for the models in our project.

4.2 Label Encoding and Feature Engineering

Following the data cleaning process, we focused on label encoding and feature engineering. Our dataset comprised a mixture of categorical and numerical variables. Key categorical variables, including item categories and customer ratings required conversion into numerical formats for compatibility with our machine-learning algorithms. This conversion was accomplished through label encoding.

One example of label encoding was applied to the 'customer rating' variable. Originally, customer ratings were on a scale from 0 to 5, with 0 often indicating no rating rather than the lowest rating. To address this and add interpretability, we re-encoded these ratings into categorical labels ranging from 'High' to 'Low', with 'High' representing the top-rated products and 'Low' indicating products with fewer or no ratings.

Feature engineering was an important where we developed new variables to provide deeper insights and enhance our model's predictive capabilities. One such feature was 'average weekly sales', calculated by grouping the sales data over each week. Additionally, we employed a correlation matrix to identify and extract the most relevant features. The correlation matrix provided a visual and quantitative measure of the relationship between different variables. By analyzing these correlations, we were able to select features that had a significant impact on our target variable of orders, ensuring that our models were fed with the most influential and informative data.



4.3 Random Forest Regression Model

Model Implementation

After preprocessing our data, we next focused on the Random Forest Regressor. This model is particularly good at handling complex, nonlinear relationships in data, making it well-suited for our supply chain demand forecasting.

Random Forest Concept

The Random Forest algorithm is an ensemble learning method, primarily used for regression and classification tasks. It operates by constructing a multitude of decision trees at training time and outputting the average prediction of the individual trees for regression tasks. The general formula for a Random Forest prediction can be expressed as:

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(X)$$

where:
$\hat{y}$ is the predicted output.
B is the number of trees in the forest.

T_b represents the prediction of the b^{th} decision tree.
X is the input feature vector.

Training the Random Forest Model In our implementation, the Random Forest model was trained on the standardized feature set $X_{train_standardized}$ and the target variable y_{train}.

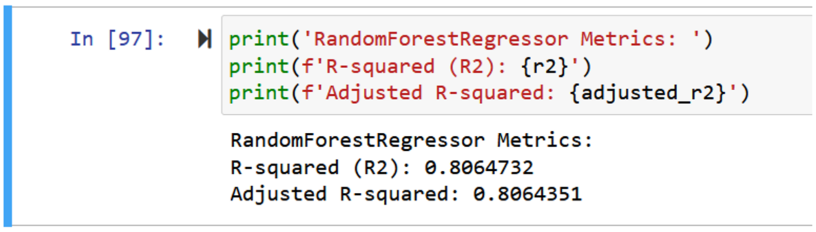
Model Evaluation Post-training, we evaluated the Random Forest model using the standardized test set $X_{test_standardized}$. The evaluation metrics included Mean Squared Error (MSE), R-squared (R^2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). The adjusted R-squared was also calculated to account for the number of predictors in the model.

$$\text{Adjusted R}^2 = 1 - (1 - R^2)\left(\frac{n-1}{n-p-1}\right)$$

where:
Adjusted R^2 is the adjusted R-squared value.
R^2 is the R-squared value.
n is the number of samples in the test set.
p is the number of features (predictors).

The MSE, R^2, MAE, and RMSE provide insights into the model's performance, indicating how closely the model's predictions align with the actual data.

Predictions and Analysis
With the model trained and evaluated, we used it to make predictions on our test data. These predictions, along with the actual orders, were then analyzed to assess the model's practical applicability in forecasting supply chain demand. Specifically, we examined the model's predictions for individual items, such as itemID 450, to understand its performance.



```
In [97]:    print('RandomForestRegressor Metrics: ')
            print(f'R-squared (R2): {r2}')
            print(f'Adjusted R-squared: {adjusted_r2}')

            RandomForestRegressor Metrics:
            R-squared (R2): 0.8064732
            Adjusted R-squared: 0.8064351
```

4.4 XGBoost Regression Model

Model Implementation Based on the base accuracy metric set by our previous models, we introduced the XGBoost Regressor. XGBoost, short for Extreme Gradient Boosting, is known for its performance in regression tasks, especially with complex datasets like ours.

XGBoost Concept
XGBoost is an advanced implementation of gradient boosting machines. It is highly efficient, flexible, and portable. The core idea behind XGBoost is to sequentially build an ensemble of weak predictive models, typically decision trees, to produce a powerful cumulative prediction. The formula for XGBoost prediction is:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$$

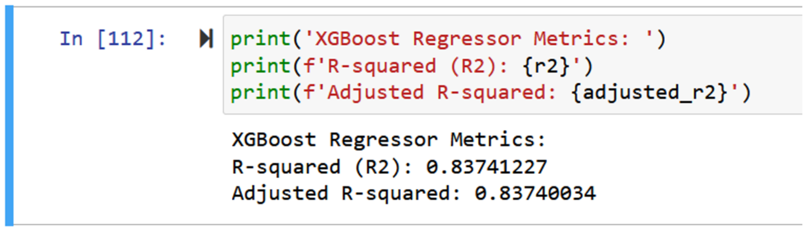
where:
$\hat{y}_i$ is the predicted output for the i^{th} instance.
K is the number of boosting rounds.
f_k represents the prediction of the k^{th} tree.
F denotes the space of all possible regression trees.
x_i is the feature vector of the i^{th} instance.

Training the XGBoost Model

The XGBoost model was trained on our standardized feature set $X_{train_{standardized}}$ with the target variable y_{train}. We configured the model with parameters optimized for our dataset's characteristics.

Model Evaluation
For evaluation, we used the same metrics as the previous models: MSE, R2, MAE, RMSE, and adjusted R-squared. The adjusted R-squared formula remains the same as previously described in Random Forest.

Predictions and Analysis After training and evaluation, the XGBoost model was used to generate predictions for our test data.



```
In [112]:  print('XGBoost Regressor Metrics: ')
           print(f'R-squared (R2): {r2}')
           print(f'Adjusted R-squared: {adjusted_r2}')

           XGBoost Regressor Metrics:
           R-squared (R2): 0.83741227
           Adjusted R-squared: 0.83740034
```

5 Neural Network Regression Model

Model Implementation
In addition to the previous models, we implemented a Neural Network using Keras. This model is particularly suitable for capturing complex, non-linear relationships in large datasets.

Neural Network Architecture
The architecture of our neural network was designed with multiple layers to enhance its learning capability:

1. Input Layer: The first layer, with nodes equal to the number of features in the dataset.
2. Hidden Layers: We used two hidden layers. The first hidden layer consisted of 128 nodes, and the second had 64 nodes. Both layers used the ReLU (Rectified Linear Unit) activation function, represented as:

$$\mathrm{ReLU}(x) = \max(0, x)$$

3. Output Layer: A single-node output layer for regression, using a linear activation function.

The model architecture is summarized as:

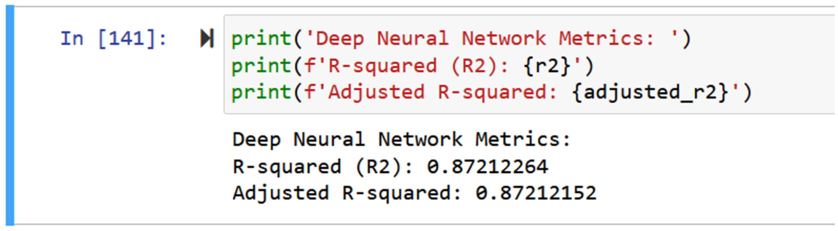
```
model = Sequential()
model.add(Dense(128, input_dim=X_train_standardized.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='linear'))
```

Model Compilation and Training
The model was compiled with the Adam optimizer and mean squared error loss function. The training process involved fitting the model to the standardized training data over multiple epochs:

```
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train_standardized, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=2)
```

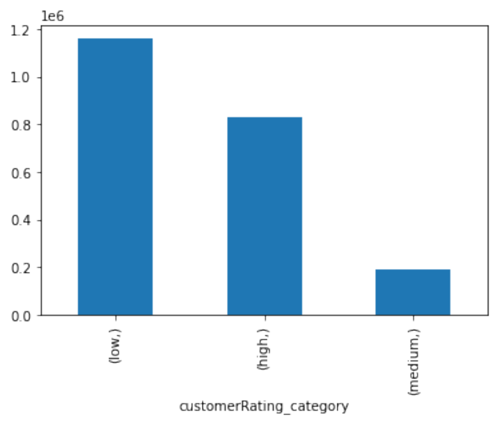
Model Evaluation
For evaluation, we used the same metrics as the previous models: MSE, R2, MAE, RMSE, and adjusted R-squared. The adjusted R-squared formula remains the same as previously described in Random Forest.

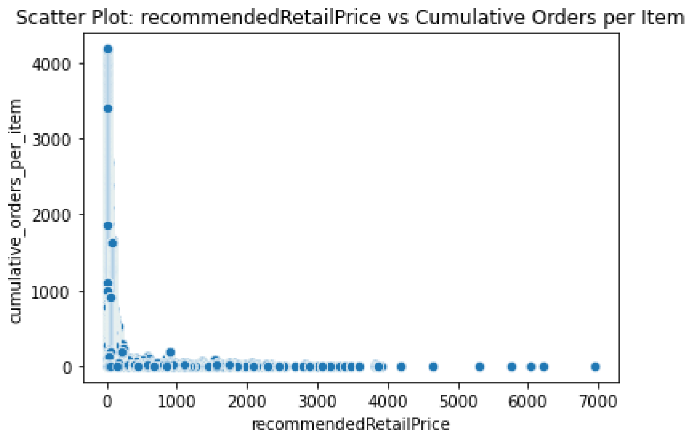
Predictions and Analysis
After training, the neural network model was used to make predictions on the standardized test set. These predictions were analyzed alongside the actual values to assess the model's performance in forecasting demand.



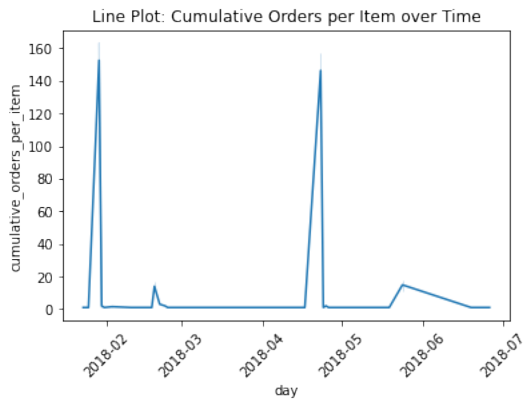
```
In [141]:  print('Deep Neural Network Metrics: ')
           print(f'R-squared (R2): {r2}')
           print(f'Adjusted R-squared: {adjusted_r2}')

           Deep Neural Network Metrics:
           R-squared (R2): 0.87212264
           Adjusted R-squared: 0.87212152
```

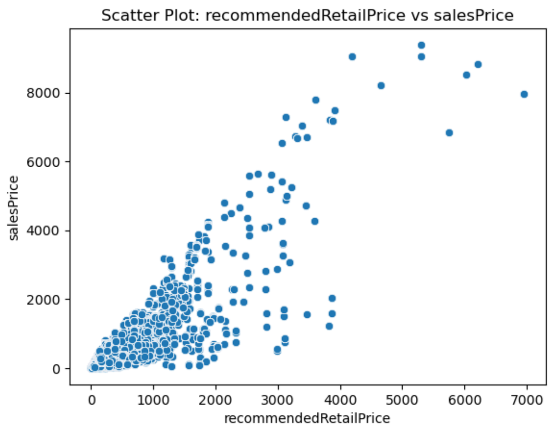
6 Results



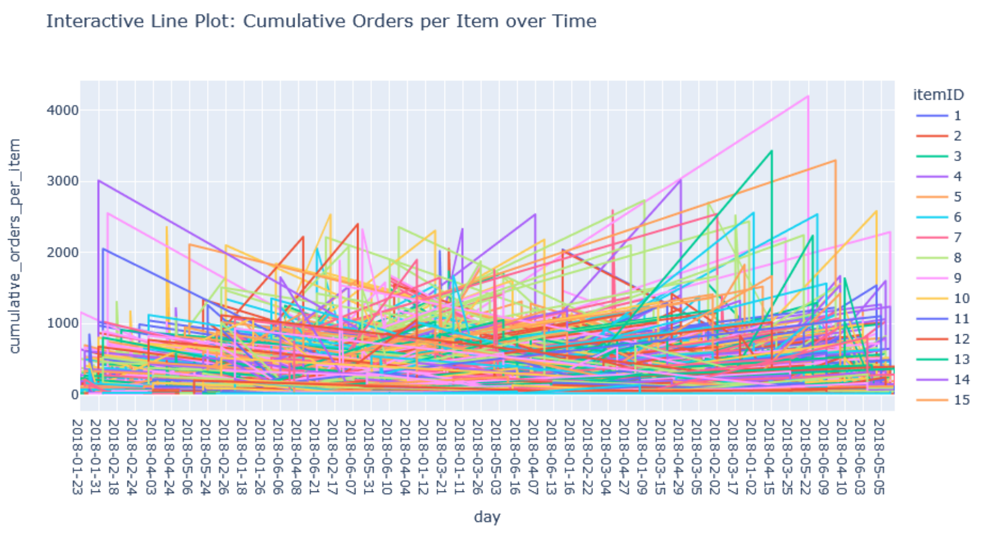


The First plot shows the Ratings distribution and the Second plot shows the distribution of the number of orders with the price recommended by the manufacturer.

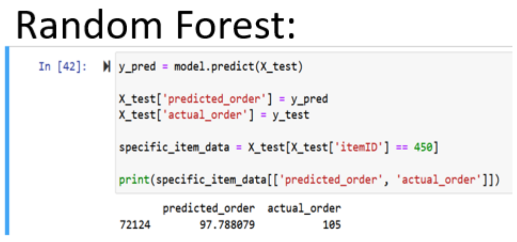




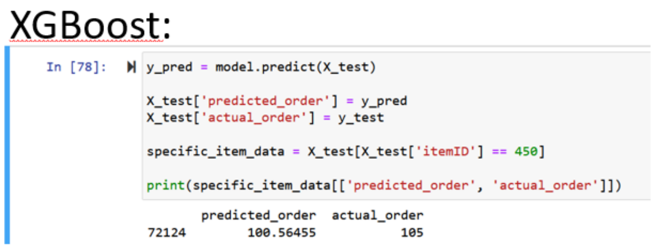
The First plot shows the Orders over time for item 1 and the Second plot shows the comparision between salesPrice by seller and the price recommended by the manufacturer.



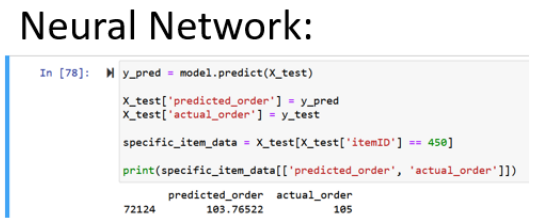
This Line Plot is a tableau version but implemented using Plotly that enables us to hover over the specific item and see how the orders are made over time.


Random Forest:

```
In [42]:  y_pred = model.predict(X_test)

          X_test['predicted_order'] = y_pred
          X_test['actual_order'] = y_test

          specific_item_data = X_test[X_test['itemID'] == 450]

          print(specific_item_data[['predicted_order', 'actual_order']])

                 predicted_order  actual_order
          72124        97.788079           105
```


XGBoost:

```
In [78]:  y_pred = model.predict(X_test)

          X_test['predicted_order'] = y_pred
          X_test['actual_order'] = y_test

          specific_item_data = X_test[X_test['itemID'] == 450]

          print(specific_item_data[['predicted_order', 'actual_order']])

                 predicted_order  actual_order
          72124        100.56455           105
```


Neural Network:

```
In [78]:  y_pred = model.predict(X_test)

          X_test['predicted_order'] = y_pred
          X_test['actual_order'] = y_test

          specific_item_data = X_test[X_test['itemID'] == 450]

          print(specific_item_data[['predicted_order', 'actual_order']])

                 predicted_order  actual_order
          72124        103.76522           105
```

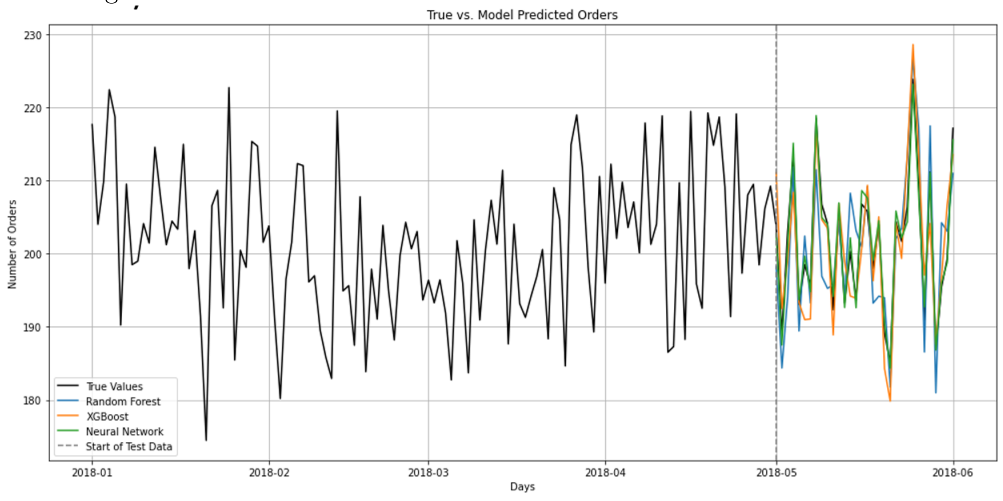
The predicted vs actual values for item 450.

Analysis of Results
Our comparative analysis of the Random Forest, XGBoost, and Neural Network models showed the difference between performance characteristics for each. The Random Forest model achieved an R-squared (R2) value of 0.8, indicating a high level of accuracy in predicting cumulative orders per item. This performance shows the model's robustness in handling complex data relationships.

The XGBoost model slightly outperformed the Random Forest with an R2 value of 0.83, demonstrating its effectiveness in managing the nuances of our dataset. The superior handling of missing values and regularization to avoid overfitting likely contributed to this enhanced performance.

Finally, the Neural Network model achieved the highest R2 value of 0.87. This superior performance can be attributed to its deep learning architecture, which excels in capturing complex patterns and relationships within the data. The model's ability to learn from a large number of features and their interactions likely provided a more nuanced understanding of the dataset, leading to more accurate predictions.

Overall, the increasing trend in R2 values from Random Forest to Neural Network reflects the growing sophistication and capability of these models in handling complex, multi-dimensional data in supply chain demand forecasting.



This graph shows the performance of our models when compared to the actual test set.

7 Conclusion

As we conclude our study on the "ChainOptimizer" project, it's clear that the integration of advanced machine learning techniques like Random Forest, XGBoost, and Neural Networks has opened new ways in the supply

chain demand forecasting. Our journey through this analytical project has been enlightening, revealing not just the power of data-driven decision-making, but also the interesting complexities in supply chain data.
Our study reaffirms the belief that the future of supply chain optimization lies in the proper usage of AI and machine learning. The increasing trend in R2 values from Random Forest to Neural Network not only marks the growing sophistication of these models but also shows the evolving complexity of supply chain challenges.
As we move forward, the insights gained from this project will fuel further innovations in the field. The potential for AI to revolutionize supply chain management is immense, and we are just scratching the surface. Our work contributes to this ongoing evolution, contributing to more efficient, responsive, and intelligent supply chain systems.

8 Team Contributions

Ajay Kumar Reddy Inavolu: Ajay was in charge of handling the data and the model selections. He spent a lot of time sorting through all sorts of information, like sales, product details, and feedback from customers. His main task was to organize this data in a way that the team's models could use it. It was a big task because the data was pretty complex. He also used a correlation matrix to figure out which parts of the data were most important. Ajay also tried various algorithms like ARIMA, LSTM and time series and selected the best of the algorithms. Ajay's work was super important because it set the stage for the rest of the project.

Neeraj Srivatsav Sanne: Neeraj played a key role in building the Neural Network Regression Model. He used a tool called Keras to create this network, which is good at understanding complex patterns in big sets of data. He designed the network with several layers, which made it more effective at learning and making predictions. His contribution was really crucial for making sense of all the detailed information the team had.

Karthikeya Reddy Kolanu: Karthikeya's main focus was the Random Forest Regression Model. He worked on improving how the model understands the data by creating new ways to interpret it, like calculating the average sales per week. This was a key step in making sure the model was working with the best information possible, leading to more accurate forecasts.

Venkata Satya Naga Sai Karthik Koduru: Karthik specialized in the XGBoost Regression Model. He trained this model using a set of carefully selected data and then tweaked it to work best with the specific kind of data they had. He also tested the model with different methods, like Mean Squared Error and R-squared, to see how well it was predicting things. These tests were important to make sure the model was reliable and giving good predictions.

9 Link to Code Repository

`https://github.com/AJAYINAVOLU/FAI_FINAL_PROJ`

10 References

1 Jones, A., & Smith, B. (2019). ARIMA-based Forecasting in Supply Chain Management: A Review. *Journal of Supply Chain Management*, 34(2), 22-35.

2 Lee, H., Zhang, Y., & Kim, S. (2020). Advanced Demand Forecasting in Supply Chains using Machine Learning Algorithms. *International Journal of Production Research*, 58(7), 2100-2119.

3 Kim, D., & Park, J. (2018). Neural Networks in Supply Chain Management: A Comprehensive Overview. *Operations Management Research*, 11(1-2), 46-60.

4 Patel, R., & Mehta, S. (2021). Reinforcement Learning for Dynamic Inventory Management. *Journal of Business Logistics*, 42(3), 234-250.

5 Brown, C., & Johnson, M. (2020). Big Data and AI in Supply Chain Optimization. *Journal of Data Science in Supply Chain Management*, 15(4), 123-138.

6 https://www.data-mining-cup.com/reviews/dmc-2020/