Universidad Autónoma de Querétaro
DIVISIÓN DE INVESTIGACIÓN Y POSGRADO

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

Faculty of Engineering
MCIA

Final Exam

Pricing Analytics

Student:
Juan Manuel
Aviña Muñoz

Professor:
Dr. Marco
Aceves Fernandez

December 1st, 2023

Contents

1 Objectives

The primary objective of the "*Pricing Analytics*" project is to implement a combination of Random Forest and a custom-built Decision Tree model for predicting product prices using a comprehensive sales history dataset. This dataset encompasses various features, including historical sales data, product categories, and other pertinent sales-related parameters. Each instance in the dataset represents a unique combination of these features, associated with the actual sales price of the product.

Our goal is to construct a predictive model using Random Forest and a handmade Decision Tree. Random Forest, a robust ensemble learning method comprising multiple decision trees, is known for its high accuracy, ability to run on large databases, and handle thousands of input variables without variable deletion. This model is particularly effective for our task because it can capture complex relationships within the data and provide reliable price predictions.

In addition to the Random Forest model, we will implement a custom-built Decision Tree. This approach allows for a deeper understanding of the decision-making logic and offers a degree of interpretability that can be invaluable in understanding the pricing dynamics.

By integrating these two models, we aim to develop a comprehensive predictive framework that can accurately forecast product prices. This framework will enable businesses to make data-driven pricing decisions, providing insights into optimal pricing strategies and contributing to enhanced revenue management and competitive market positioning, ultimately seeking to transform how businesses approach pricing, leading to well-informed decisions and improved financial outcomes.

2 Introduction

In the ever-evolving landscape of business, the strategic importance of effective pricing strategies cannot be overstated. This project is poised at the forefront of this domain, leveraging advanced machine learning techniques to revolutionize the approach businesses take towards pricing their products.

The Random Forest model, an ensemble learning method renowned for its accuracy and robustness, forms the backbone of our predictive analysis; by harnessing the collective power of multiple decision trees, this model offers a nuanced understanding of the complex relationships hidden within the data, leading to reliable and precise price predictions.

Complementing the Random Forest is our custom-designed Decision Tree, this model provides an intuitive and interpretable structure, allowing for a transparent view into the decision-making process. This clarity is instrumental in deciphering the intricate dynamics of product pricing.

The synergy of these two models aims to unlock unprecedented insights into optimal pricing strategies. This endeavor is not just a step towards enhanced revenue management and competitive positioning in the market; it's a stride towards fundamentally transforming the way businesses make pricing decisions.

3 Theoretical Foundation

3.1 Decision Trees

Decision Trees are non-parametric supervised learning methods used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

3.1.1 Key Concepts

- **Node Structure:** The root node represents the entire dataset, which gets divided into two or more homogeneous sets, at each node, the tree splits based on a condition on a single feature, aiming to separate the data such that similar response values end up in the same set after the split.

- **Splitting Criteria:** *Entropy* is a measure of disorder or uncertainty, the goal in a decision tree is to reduce entropy after each split, Equation 1.

  $$Entropy(S) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{1}$$

  where p_i is the proportion of data points in category i in set S.

  Information Gain measures the reduction in entropy or surprise by splitting a dataset according to a particular feature. A decision tree algorithm will try to maximize information gain, Equation 2.

  $$IG(S, A) = S - \sum_{t \in T} p(t) * Entropy(t) \tag{2}$$

  where T are subsets created from splitting set S by attribute A.

- **Tree Construction:** We begin by doing a recursive binary splitting, which is the process of starting at the root and recursively finding the best split for all of our nodes.

  Decision trees can become susceptible to overfitting, particularly when they are allowed to grow extensively in depth. Overfitting occurs when the tree starts to model the random noise in the data rather than the underlying pattern, to prevent this, it is crucial to employ pruning techniques which effectively trim the tree to an optimal size, there are two strategies:

  - **Pre-pruning:** Stopping the tree before it fully develops.
  - **Post-pruning:** Allowing the tree to fully develop and then removing nodes.

- **Variability:** A small change in the data can lead to a very different series of splits, making decision trees somewhat unstable; ensemble methods like Random Forests help mitigate this issue by averaging multiple decision trees.

- **Handling Different Types of Data:** Decision trees can handle categorical and numerical data, however, the handling of numerical data might involve discretization into categories.

3.1.2 Practical Applications

Random Forest Regressor has a wide range of practical applications across various domains due to its versatility and ability to handle complex tasks, here are some of it's uses:

- **Classification Tasks:** From diagnosing diseases to categorizing customers based on purchasing behaviors.
- **Regression Tasks:** Such as predicting housing prices or forecasting sales.
- **Feature Selection:** Decision trees can be used for identifying the most effective variables in a dataset.

Gaining a thorough understanding of these fundamental concepts is crucial not only for effectively implementing decision trees but also for recognizing their wide-ranging applicability and inherent limitations in diverse machine learning scenarios. Here are some practical real-world applications:

- **Finance:** Generally used for predicting stock prices or financial market trends, credit scoring and risk assessment for loan approvals and fraud detections in financial transactions.
- **Healthcare:** It demonstrates proficiency in predicting patient outcomes based on medical data, identifying disease risk factors and treatment response and medical image analysis for diagnosis and prognosis.
- **Energy:** The model is adept at predicting energy consumption for resource planning, fault detection in energy infrastructure and renewable energy forecasting.
- **Manufacturing:** Employing predictive maintenance to minimize equipment downtime, ensuring quality control and detecting defects in manufacturing processes, and optimizing supply chain operations alongside effective inventory management.

These are just a few examples, and the versatility of Random Forest Regressor allows it to be applied to a wide range of problems in different industries. It's ability to handle both regression and classification tasks makes it a popular choice for various real-world applications where accurate predictions are crucial [1, 2].

3.1.3 Advantages

- Random Forest generally provides high accuracy in predicting outcomes compared to single decision trees. It's less prone to overfitting due to the averaging effect of multiple trees.

- It is robust to outliers in the data because it aggregates predictions from multiple trees, reducing the impact of individual outliers.
- Random Forest can handle missing values in the dataset. It imputes missing values during training and handles them during predictions.
- It provides a feature importance score, which can help in understanding the relative significance of different features in making predictions.
- The use of multiple trees reduces the risk of overfitting, especially when the dataset has noise or irrelevant features.
- Random Forest can be used for both regression and classification tasks, making it a versatile algorithm.
- Training of individual trees in a Random Forest can be done in parallel, leading to faster training times on large datasets.

3.1.4 Disadvantages

- The model itself can be difficult to interpret compared to a single decision tree. Understanding the relationship between features and predictions might not be straightforward.
- Random Forests can be resource-intensive, especially in terms of memory and computational power, which might be a limitation on large datasets.
- Like many ensemble models, Random Forest is considered a "black box" model, meaning it's challenging to understand the internal workings of the algorithm.
- In classification tasks with imbalanced class distributions, Random Forests may be biased toward the dominant class.
- On noisy datasets with a high level of randomness, Random Forest might not perform as well compared to other algorithms.
- While individual trees can be trained in parallel, the overall training time might be longer compared to simpler models like linear regression.
- Random Forest may not perform well on very small datasets as it requires a sufficient amount of data to capture meaningful patterns.

3.2 Data Analytics and Business Analytics

Data Analytics and Business Analytics play a crucial role in aiding organizations in decision-making. These analytics provide insights that help businesses understand market trends, customer behavior, operational efficiency, and more, enabling informed and strategic decision-making [3].

- **Data Analytics:** Involves the process of examining data sets to draw conclusions about the information they contain. Data analytic techniques enable you to take raw data and uncover patterns to extract valuable insights.
- **Business Analytics:** Focuses more on using statistical analysis and predictive modeling to identify trends, manage risk, and optimize operations within a business context.

3.2.1 Data Analytics in Decision-Making

- By analyzing customer data, companies can segment their customers into distinct groups based on purchasing behavior, preferences, or demographics; which enables targeted marketing, personalized product offerings and improved customer engagement strategies.
- Data analytics can predict equipment failures before they occur by analyzing historical operation data, this predictive maintenance saves costs and downtime, ensuring optimal operational efficiency.
- Analyzing data from various points in the supply chain can help identify inefficiencies and bottlenecks; by optimizing these areas, businesses can reduce costs, improve delivery times and enhance overall supply chain resilience.
- Data analytics helps assessing financial health by analyzing cash flows, expenditures and revenue streams, it's also crucial in risk assessment, helping businesses to mitigate potential financial risks and fraud.
- Analyzing market trends enables businesses to adapt their strategies according to changing market conditions, stay competitive and capitalize on emerging opportunities.

3.2.2 Business Analytics in Decision-Making

- Business analytics provides insights into market dynamics, competition and internal operations, aiding leaders in making strategic decisions like market entry, product launches and pricing strategies.
- Through KPIs and metrics analysis, organizations can measure performance across departments, this helps in identifying areas of success and those needing improvement.
- Analyzing customer interaction data helps in understanding customer satisfaction and loyalty; insights gained can drive improvements in customer service and retention strategies.
- Business analytics in HR can help in talent acquisition, understanding employee turnover and optimizing workforce management for better productivity.
- By analyzing internal processes, organizations can identify inefficiencies and areas for process automation, leading to cost savings and enhanced productivity.

3.2.3 Real-World Examples

- A retail company uses customer purchasing data to personalize marketing campaigns, resulting in increased sales and customer loyalty.
- A logistics company employs predictive analytics to optimize routes and delivery schedules, reducing fuel costs and improving delivery times.
- A financial institution implements advanced analytics for credit scoring, reducing the risk of loan defaults.

These examples demonstrate how leveraging data analytics and business analytics can lead to more informed, data-driven decisions and ultimately contributing to the operational effectiveness and strategic success of organizations.

4 Methods and Materials

4.1 Dataset

The dataset, provided by Dr. Aceves, encompasses comprehensive sales records of a company spanning from 1996 to 2023. The data is meticulously organized, with each year's data segregated into separate sheets, and within these sheets, the sales information is broken down on a weekly basis.

In total, the dataset comprises 1440 instances, each representing weekly sales data across various years, and 20 distinct features, where each instance corresponds to the sales data of a particular week across the years and every feature corresponds to a particular product. However, it's noteworthy that there are some instances of missing data within this extensive collection.

Here are the key features and details of the dataset:

- **Week Overview:** Denotes the sales week within a given year.
- **Product Data:** Illustrates the average sales figures for each week.
- **Yearly Perspective:** Represents the sales year under consideration.

4.1.1 Preprocessing

To properly utilize the provided database, a series of meticulous data preprocessing protocols were employed. These steps were crucial in creating a new, refined database while striving to maintain the original distribution of the data values as faithfully as possible. The following procedures were meticulously executed:

- This step involved identifying and eliminating rows that contained average values, which could skew the analysis by introducing redundancy or bias.
- To ensure the uniqueness of temporal data, any instances of duplicate records across time were removed, thereby preserving the integrity of the time series data.
- Rows that lacked pertinent information or contributed no meaningful insight to the analysis were systematically removed. This action was taken to enhance the quality and relevance of the data.
- Dates were uniformly formatted to the ISO standard (MM/DD/YYYY), ensuring consistency and facilitating easier manipulation and analysis of temporal data.
- In instances of missing data, cells were populated with zeroes as a placeholder. This approach was adopted as a preliminary step to maintain data integrity before applying more sophisticated imputation techniques.
- Post data arrangement, an extensive visualization was conducted. This included graphically representing the data to observe and analyze its distribution patterns.

- Various imputation techniques were applied to address missing or incomplete data. This included linear regression imputations and mean imputation, aimed at preserving the underlying trends and characteristics of the data.
- The dataset was also subjected to other imputation strategies to observe behavioral changes. This comparative analysis helped in determining the most suitable imputation method that minimally distorts the original data distribution.
- In our custom modeling approach, upon importing the dataset, we observed that it is in the form of a Pandas DataFrame, however, our model requires the data specifically as a NumPy array, for optimal functionality. Consequently, it's essential to transform the DataFrame into a NumPy array before proceeding with the modeling process.

Through these meticulous preprocessing steps, the database was refined and transformed into a resource more suitable for in-depth analysis, ensuring both the reliability and integrity of subsequent data interpretations and insights.

4.1.2 Metrics

Mean Squared Error (*MSE*) is the average of the squares of the errorsthat is, the average squared difference between the estimated values and the actual value, Equation 3.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{3}$$

- where n is the number of data points
- Y_i is the actual value for the $i-th$ data point.
- $\hat{Y}_i$ is the predicted value for the ii-th data point.

Mean Absolute Error (*MAE*) is the average of the absolute differences between the predicted values and observed values, Equation 4.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}Y_i - \hat{Y}_i \tag{4}$$

- where n is the number of data points
- Y_i is the actual value for the $i-th$ data point.
- $\hat{Y}_i$ is the predicted value for the $i-th$ data point.

R-squared (*Coefficient of Determination*) represents the proportion of the variance for the dependent variable that's explained by the independent variables in a regression model, Equation 5.

$$R^2 = 1\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \overline{Y}_i)^2} \tag{5}$$

- where n is the number of data points
- Y_i is the actual value for the $i - th$ data point.
- $\hat{Y}_i$ is the predicted value for the ii-th data point.
- $\overline{Y}_i$ is the mean of the actual values Y.

The R-squared value ranges from 0 to 1, where a higher value indicates a better fit of the model to the data.

4.2 Resources

Equipment:

- Intel i7-9750H @ 2.60 GHz processor.
- 16 GB RAM DDR4 @ 2666 MHz.
- Linux Mint 21.1 x86-64 OS
- GeForce GTX 1660Ti @ 1455 MHz 6 GB VRAM GDDR6.

4.3 Libraries

The following libraries were utilized for this activity:

- **NumPy** is a fundamental package for scientific computing in Python as it provides support for large, multi-dimensional arrays and matrices, along with a variety of mathematical functions to operate on these arrays. NumPy is a core library in the data science ecosystem and is used for tasks like numerical computations, linear algebra operations, random number generation, and more.
- **Pandas** is a powerful library for data manipulation and analysis in Python. It provides data structures (primarily Series and DataFrame) to efficiently handle and manipulate structured data. Pandas enables tasks such as loading data from various file formats (CSV, Excel, SQL databases, etc), cleaning and preprocessing data, filtering, merging, transforming, and summarizing data. It's widely used for data wrangling and exploration.

- **Seaborn** is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn simplifies the process of creating complex visualizations like scatter plots, bar plots, histograms, box plots, and more. It also supports color palettes, themes, and additional statistical plotting capabilities.

- **Matplotlib** is a versatile 2D plotting library in Python. It provides a wide range of functions to create static, interactive, and animated visualizations. While it can be used to create basic plots, it can also be customized to create complex visualizations. Matplotlib is the foundation for many other visualization libraries and tools.

- **Scikit Learn** is a machine learning library.
  - **Metrics:** These functions are used to evaluate the performance of machine learning models.
  - **Model Selection:** This function used for splitting datasets into random train and test subsets. Generally used to easily split a dataset into a training set and a testing set, which is essential for training and evaluating machine learning models.
  - **RandomForestRegressor:** This class from the "*ensemble*" module is used for regression tasks, where the goal is to predict continuous outcomes based on input features. Random Forest is an ensemble learning method that operates by constructing multiple decision trees during the training phase and outputs the average prediction of these trees for regression tasks.

- **Fancyimpute:** An imputation library used to handle missing values in datasets. The library *IterativeImputer* is a flexible imputer that uses various regression models to estimate missing values iteratively. It is particularly useful for complex imputation tasks and is based on the idea of Multiple Imputation by Chained Equations (MICE).

These libraries form a comprehensive toolkit for data preprocessing, analysis, visualization, machine learning modeling, and evaluation, making them indispensable in the field of data science and analytics.

5 Results

Figure 1 shows the price trends for every product over the years.

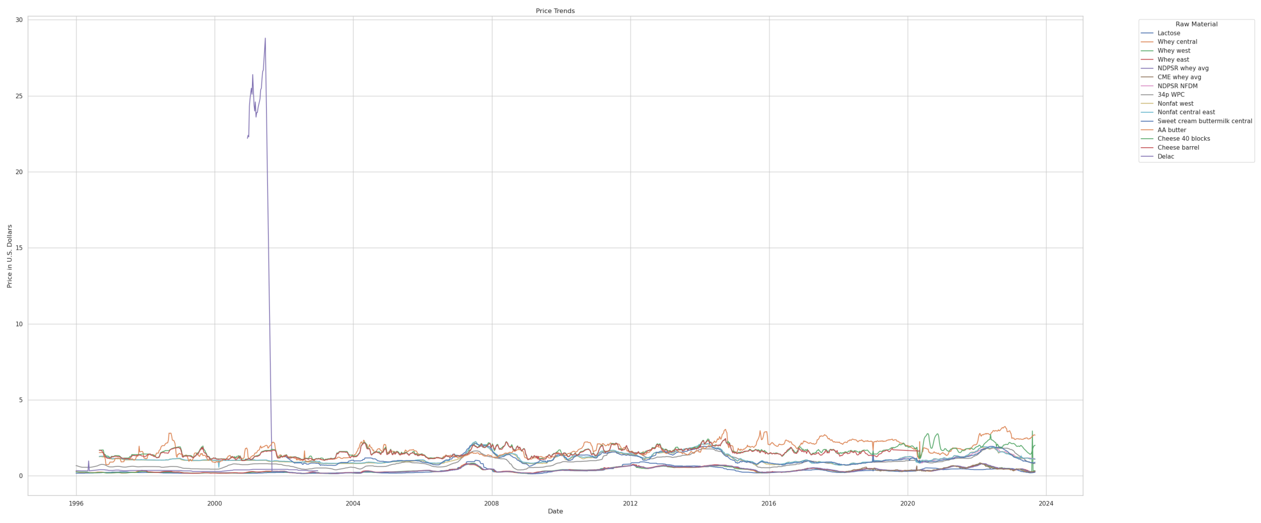


Figure 1: *`Product trends by year.`*

Figure 2 shows the price behavior of the "*Lactose*" feature across the years.

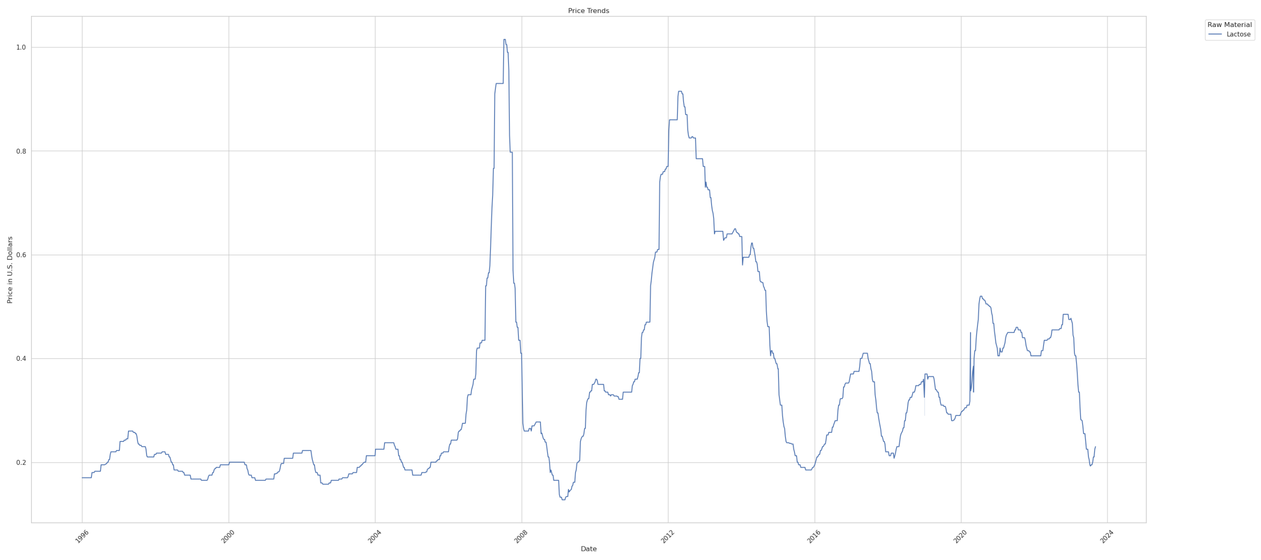


Figure 2: *`"Lactose" feature trend by year.`*

Figure 3 show the "*Lactose*" feature weekly price tendency across the years.

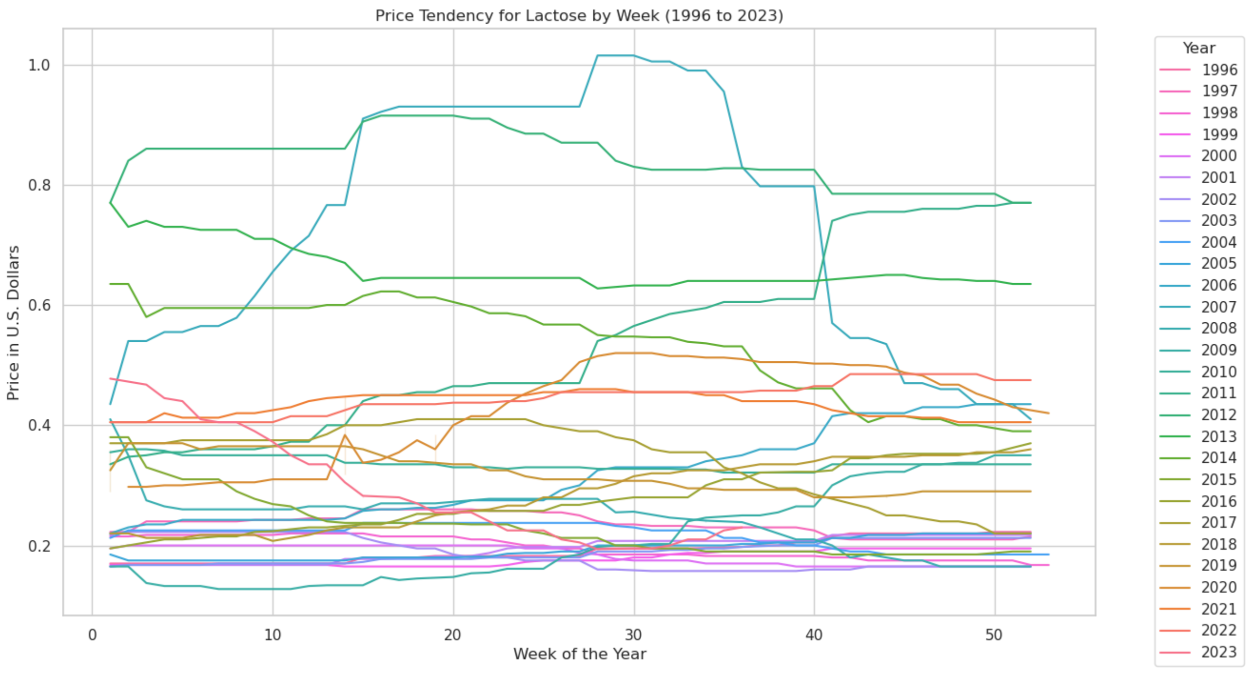


Figure 3: `"Lactose" feature trend by week (1996-2023).`

In order to deal with our outliers, shown in Figures 4 and 5 try to remove all outliers for all attributes and impute with the median of each attribute.

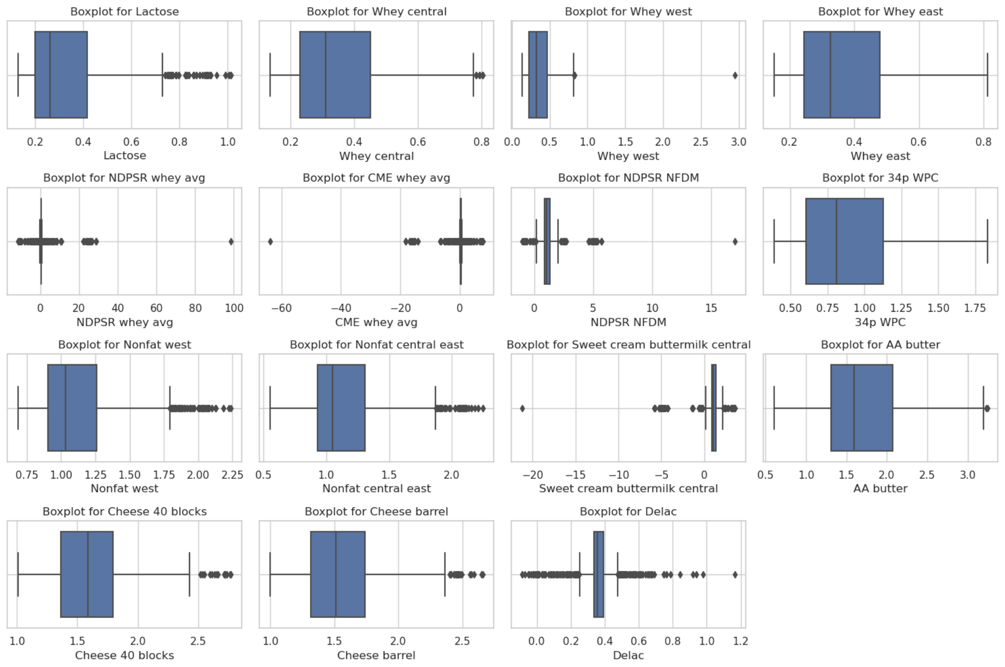


Figure 4: *Outliers before processing.*

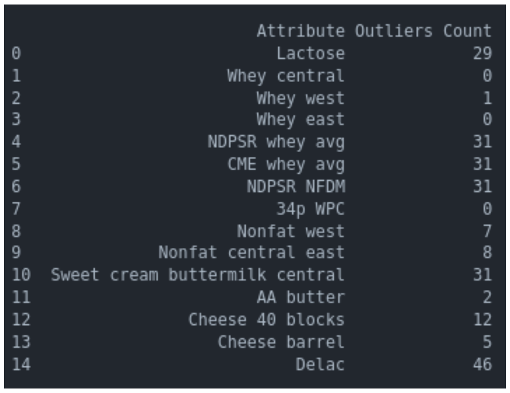

```
                      Attribute Outliers Count
0                       Lactose             29
1                  Whey central              0
2                     Whey west              1
3                     Whey east              0
4                NDPSR whey avg             31
5                  CME whey avg             31
6                   NDPSR NFDM              31
7                      34p WPC               0
8                  Nonfat west               7
9           Nonfat central east              8
10  Sweet cream buttermilk central          31
11                    AA butter              2
12             Cheese 40 blocks             12
13                Cheese barrel              5
14                        Delac             46
```

Figure 5: *Outliers before processing.*

Figures 6 and 7 show the outliers from our dataset after processing.

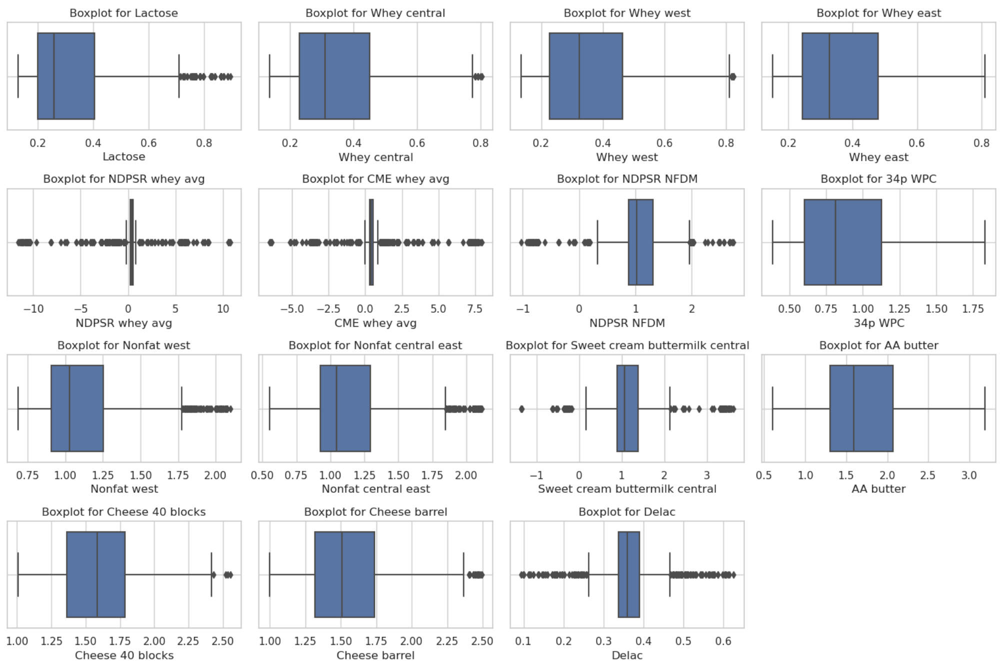


Figure 6: *Outliers after processing.*

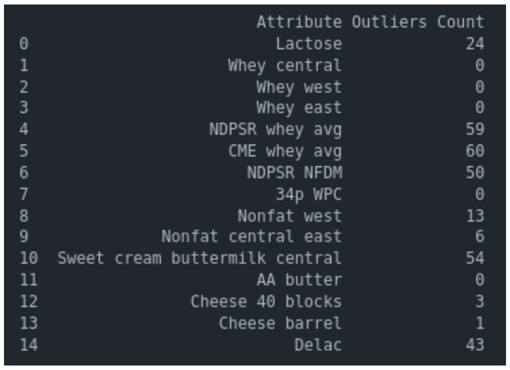
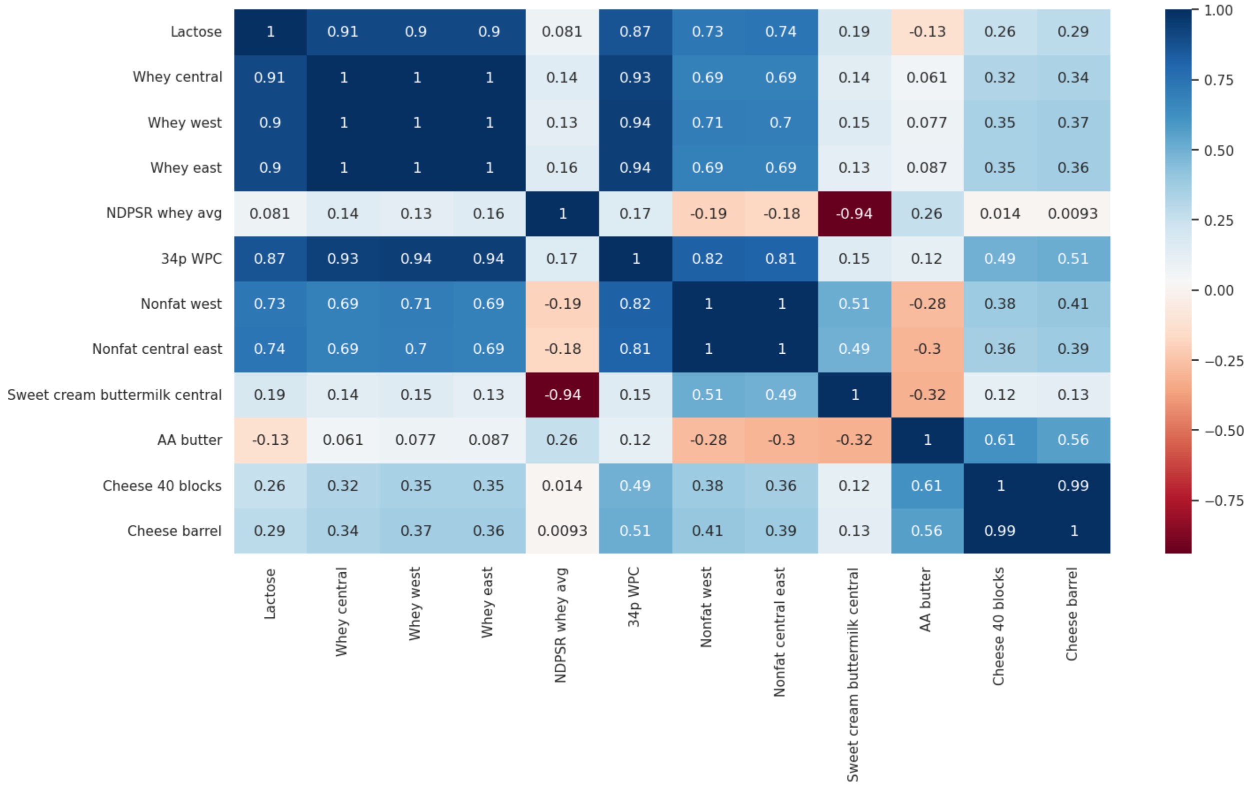

```
                         Attribute  Outliers Count
0                          Lactose              24
1                     Whey central               0
2                        Whey west               0
3                        Whey east               0
4                   NDPSR whey avg              59
5                     CME whey avg              60
6                       NDPSR NFDM              50
7                         34p WPC                0
8                      Nonfat west              13
9              Nonfat central east               6
10  Sweet cream buttermilk central              54
11                       AA butter               0
12                Cheese 40 blocks               3
13                   Cheese barrel               1
14                           Delac              43
```

Figure 7: *Outliers after processing.*

In order to select the best features for our models, we use a Correlation Matrix, shown in Figure 8.

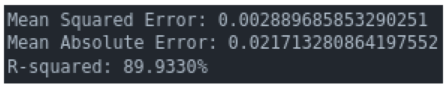


	Lactose	Whey central	Whey west	Whey east	NDPSR whey avg	34p WPC	Nonfat west	Nonfat central east	Sweet cream buttermilk central	AA butter	Cheese 40 blocks	Cheese barrel
Lactose	1	0.91	0.9	0.9	0.081	0.87	0.73	0.74	0.19	-0.13	0.26	0.29
Whey central	0.91	1	1	1	0.14	0.93	0.69	0.69	0.14	0.061	0.32	0.34
Whey west	0.9	1	1	1	0.13	0.94	0.71	0.7	0.15	0.077	0.35	0.37
Whey east	0.9	1	1	1	0.16	0.94	0.69	0.69	0.13	0.087	0.35	0.36
NDPSR whey avg	0.081	0.14	0.13	0.16	1	0.17	-0.19	-0.18	-0.94	0.26	0.014	0.0093
34p WPC	0.87	0.93	0.94	0.94	0.17	1	0.82	0.81	0.15	0.12	0.49	0.51
Nonfat west	0.73	0.69	0.71	0.69	-0.19	0.82	1	1	0.51	-0.28	0.38	0.41
Nonfat central east	0.74	0.69	0.7	0.69	-0.18	0.81	1	1	0.49	-0.3	0.36	0.39
Sweet cream buttermilk central	0.19	0.14	0.15	0.13	-0.94	0.15	0.51	0.49	1	-0.32	0.12	0.13
AA butter	-0.13	0.061	0.077	0.087	0.26	0.12	-0.28	-0.3	-0.32	1	0.61	0.56
Cheese 40 blocks	0.26	0.32	0.35	0.35	0.014	0.49	0.38	0.36	0.12	0.61	1	0.99
Cheese barrel	0.29	0.34	0.37	0.36	0.0093	0.51	0.41	0.39	0.13	0.56	0.99	1

Figure 8: *Correlation Matrix for the Dataset.*

To assess our model's performance, we will utilize the following features *'Whey central', 'Whey west', 'Whey east', '34p WPC', 'Nonfat west', 'Nonfat central east', 'Sweet cream buttermilk central'*. Our target variable for predictions will be *'Lactose'*. The dataset has been divided into a training set and a test set with a ratio of 70/30, respectively.

First, we will use the "*RandomForestRegressor*" model from Scikit-Learn ensemble library with 100 estimators (trees). This model gives the following metrics, Figures 9 and 10.



```
Mean Squared Error: 0.002889685853290251
Mean Absolute Error: 0.021713280864197552
R-squared: 89.9330%
```

Figure 9: *Results for the prediction in the Scikit-Learn model.*

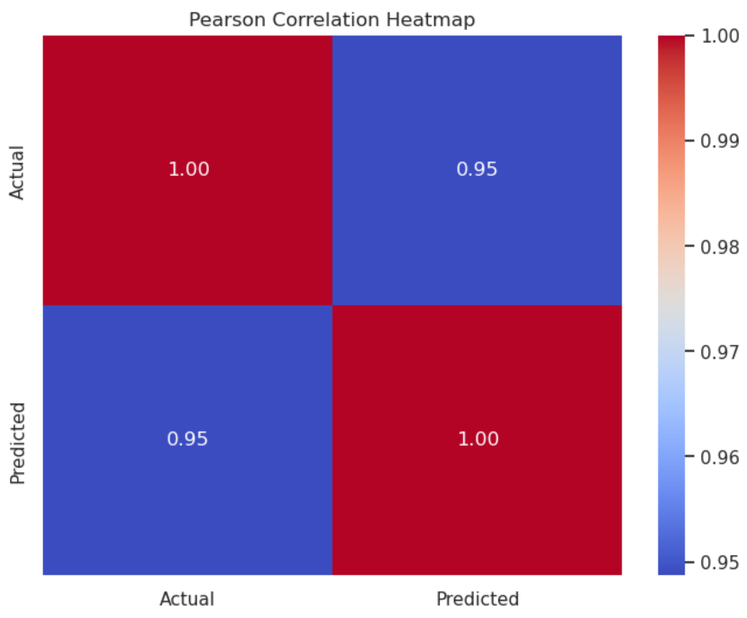
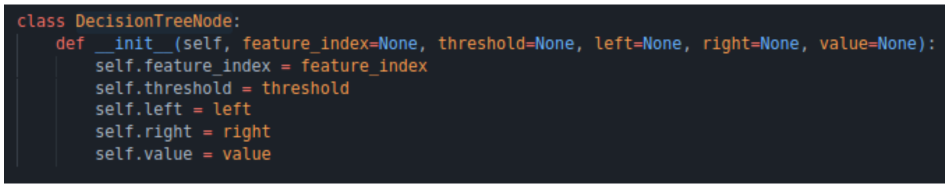


Figure 10: *Pearson Correlation Heatmap for the prediction in the Scikit-Learn model.*

The value of 0.95 in Figure 10 indicates a very high positive correlation between the "Actual" and "Predicted" variables. This suggests that the predictions made by the model are highly correlated with the actual values.

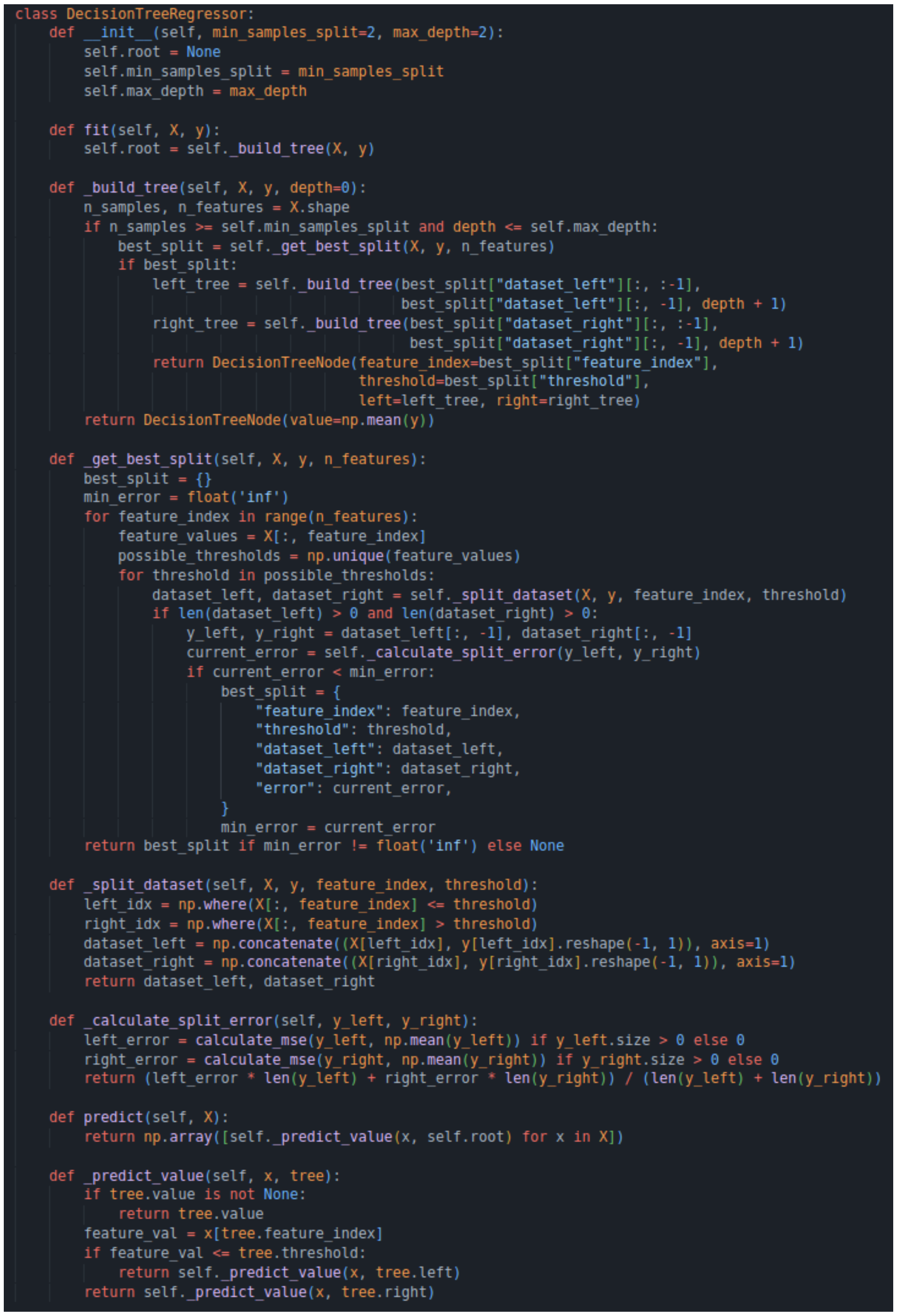
For our custom model, a "*DecissionTreeNode*" class was created which represents a single node in our decision tree structure, Figure 11.



```
class DecisionTreeNode:
    def __init__(self, feature_index=None, threshold=None, left=None, right=None, value=None):
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
```

Figure 11: *DecisionTreeNode code snippet.*

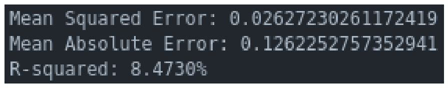
Next, the model was created, Figure 12 giving it a minimum sample split of 2 controlling the minimum number of samples required to split an internal node; a max depth of 2 allowing the tree to grow deeper to capture more details; minimum sample leafs of 1 which specifies the minimum number of samples required to be at a leaf node.



```python
class DecisionTreeRegressor:
    def __init__(self, min_samples_split=2, max_depth=2):
        self.root = None
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth

    def fit(self, X, y):
        self.root = self._build_tree(X, y)

    def _build_tree(self, X, y, depth=0):
        n_samples, n_features = X.shape
        if n_samples >= self.min_samples_split and depth <= self.max_depth:
            best_split = self._get_best_split(X, y, n_features)
            if best_split:
                left_tree = self._build_tree(best_split["dataset_left"][:, :-1],
                                             best_split["dataset_left"][:, -1], depth + 1)
                right_tree = self._build_tree(best_split["dataset_right"][:, :-1],
                                              best_split["dataset_right"][:, -1], depth + 1)
                return DecisionTreeNode(feature_index=best_split["feature_index"],
                                        threshold=best_split["threshold"],
                                        left=left_tree, right=right_tree)
        return DecisionTreeNode(value=np.mean(y))

    def _get_best_split(self, X, y, n_features):
        best_split = {}
        min_error = float('inf')
        for feature_index in range(n_features):
            feature_values = X[:, feature_index]
            possible_thresholds = np.unique(feature_values)
            for threshold in possible_thresholds:
                dataset_left, dataset_right = self._split_dataset(X, y, feature_index, threshold)
                if len(dataset_left) > 0 and len(dataset_right) > 0:
                    y_left, y_right = dataset_left[:, -1], dataset_right[:, -1]
                    current_error = self._calculate_split_error(y_left, y_right)
                    if current_error < min_error:
                        best_split = {
                            "feature_index": feature_index,
                            "threshold": threshold,
                            "dataset_left": dataset_left,
                            "dataset_right": dataset_right,
                            "error": current_error,
                        }
                        min_error = current_error
        return best_split if min_error != float('inf') else None

    def _split_dataset(self, X, y, feature_index, threshold):
        left_idx = np.where(X[:, feature_index] <= threshold)
        right_idx = np.where(X[:, feature_index] > threshold)
        dataset_left = np.concatenate((X[left_idx], y[left_idx].reshape(-1, 1)), axis=1)
        dataset_right = np.concatenate((X[right_idx], y[right_idx].reshape(-1, 1)), axis=1)
        return dataset_left, dataset_right

    def _calculate_split_error(self, y_left, y_right):
        left_error = calculate_mse(y_left, np.mean(y_left)) if y_left.size > 0 else 0
        right_error = calculate_mse(y_right, np.mean(y_right)) if y_right.size > 0 else 0
        return (left_error * len(y_left) + right_error * len(y_right)) / (len(y_left) + len(y_right))

    def predict(self, X):
        return np.array([self._predict_value(x, self.root) for x in X])

    def _predict_value(self, x, tree):
        if tree.value is not None:
            return tree.value
        feature_val = x[tree.feature_index]
        if feature_val <= tree.threshold:
            return self._predict_value(x, tree.left)
        return self._predict_value(x, tree.right)
```

Figure 12: `DecisionTreeRegressor code snippet.`

This model gives the following results, Figure 13 and 14.



```
Mean Squared Error: 0.02627230261172419
Mean Absolute Error: 0.1262252757352941
R-squared: 8.4730%
```

Figure 13: *Results for the prediction in the Custom model.*

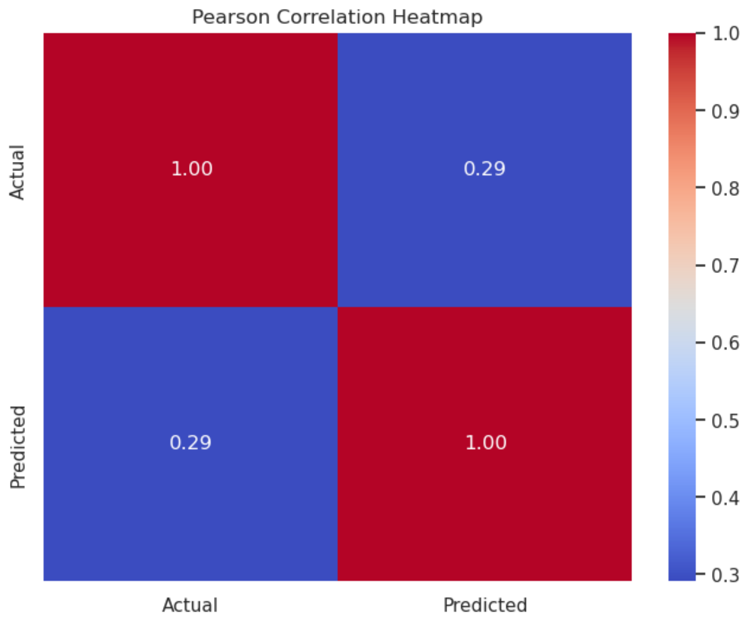


Figure 14: *Pearson Correlation Heatmap for the prediction in the Custom model.*

The value of 0.29 in Figure 14 indicates a positive but weak correlation between the actual and predicted values. This suggests that there is some linear relationship between the actual and predicted values, but it's not strong.

6 Discussion

Predicting company sales is a complex task that involves understanding market trends, consumer behavior, and numerous other factors. The choice of the best model often depends on the specific characteristics of the dataset, the nature of the sales process, and the business context. However, several models are commonly used due to their effectiveness in handling such tasks:

- **Random Forest Regressor:** Handles non-linear data well, is robust to outliers, and can model complex relationships without extensive data preprocessing. It's ability to handle a mix of categorical and numerical data, provide feature importance insights, and its inherent mechanism to avoid overfitting make it a strong choice for sales prediction.
- **Linear Regression and its Variants (Ridge, Lasso):** Simple to moderately complex datasets where relationships between features and sales are linear. Easy to implement and interpret, useful for understanding the influence of each predictor on sales.
- **Time Series Models (ARIMA, SARIMA):** It's best suited for data with strong temporal components, like seasonal sales patterns. These models are specifically designed to handle time-based patterns and can be very effective in forecasting short-term sales.
- **Gradient Boosting Machines (GBM) and XGBoost:** Often provide superior predictive accuracy by sequentially correcting errors from previous models but require careful tuning of parameters but can outperform many models if configured correctly.

The choice of the model is influenced by the specific business context, the nature of the sales data, and the desired balance between predictive accuracy and model interpretability. Random Forest is often chosen for its robustness, versatility, and balance between complexity and performance.

Using Scikit-Learn's RandomForestRegressor library offers several advantages over building a model from scratch:

- **Efficiency and Reliability:** Scikit-Learns implementations are highly optimized and tested. They offer efficient computation and robust performance out-of-the-box, which is hard to match with custom-built models.
- **Ease of Use:** Scikit-Learn provides a user-friendly interface, making it easier to implement, train, and evaluate models without delving into the complexities of algorithmic implementation.
- **Community and Support:** Scikit-Learn has a large community and extensive documentation, offering valuable support, examples, and best practices.
- **Integrated Tools and Compatibility:** Scikit-Learn models integrate seamlessly with other Python libraries and tools, making it easier to preprocess data, tune models, and evaluate results.

- **Regular Updates and Maintenance:** Being an open-source library, Scikit-Learn is regularly updated with the latest research findings, new features, and performance improvements.

In the context of a predictive model, a Pearson correlation coefficient of 0.29 would generally be considered low, indicating that the model's predictions are not very accurate in terms of matching the actual values. In other words, the model has limited effectiveness for the task it is trying to predict. For improving the model, one might look into better feature selection, more data, feature engineering, different modeling techniques, or hyperparameter tuning.

In summary, machine learning and specifically models like Random Forest Regressor implemented in Scikit-Learn, provide businesses with advanced tools to accurately predict sales prices. These predictions enable companies to make data-driven decisions, optimize pricing strategies and maintain competitive advantages in the market. The choice of Scikit-Learn's RandomForestRegressor ensures efficiency, reliability, and ease of use, crucial for practical applications in a business setting.

7 Conclusions

Machine learning algorithms have become an essential tool for predicting sales prices in businesses due to their ability to handle complex, multifaceted data and extract meaningful patterns. These algorithms can analyze vast amounts of historical sales data, considering various factors that influence pricing such as market trends, customer behavior, seasonality and economic indicators. Unlike traditional statistical methods, machine learning models can automatically adapt to changes in market dynamics, providing more accurate and timely predictions, this adaptability is crucial in fast-paced business environments where pricing strategies can significantly impact revenue and competitiveness.

The Random Forest Regressor model was chosen for several reasons:

- It is particularly effective in capturing complex, non-linear relationships between features without the need for extensive manual feature engineering.
- By being an ensemble of decision trees, is less prone to overfitting compared to individual decision trees. This robustness is crucial for achieving reliable predictions.
- It provides insights into the importance of each feature in predicting sales prices, aiding in understanding which factors most significantly impact pricing.
- It works well with a mix of numerical and categorical data and typically requires less hyperparameter tuning to achieve high performance.

Handling Complex Interactions:

References

[1] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining.* Pearson, 2005.

[2] K. P. Murphy, *Machine Learning: a Probabilistic Perspective.* MIT Press, 2012.

[3] R. Mohammed, *The Art of Pricing, New Edition: How to Find the Hidden Profits to Grow Your Business*, 2017.

[4] L. Pierson, *Data Science for Dummies.* Wiley, 2015.

[5] P. H. Winston, *Artificial Intelligence, Third Edition.* Addison-Wesley, 1992.