Taner Giray Sönmez / Enis Mert Kuzu / Ahmet Furkan Ün / Aydın Enes Bekar

GROUP 9

Introduction

The restaurant industry is one of the most competitive and rapidly evolving industries in the world. With an ever-increasing number of restaurants, customers have a wider range of options to choose from. To stay ahead of the competition, restaurants need to be able to predict their future sales accurately. Predicting future sales can help restaurant owners to make informed decisions about their inventory, staff scheduling, and marketing strategies.

In this project, we are tasked with predicting the future sales of different restaurants in Uppsala, Sweden, using historical data of their daily revenue. Our goal is to compare a number of algorithms and preprocessing methods to see how well they compare. To achieve this, we used a variety of techniques for feature extraction and pre-processing, including using information on public holidays, weather data, and different ways of converting date values to a format suitable for machine learning algorithms. After pre-processing the data and adding features such as public holiday information and weather data, we explored a variety of machine learning models to predict future sales for different restaurants in Uppsala. We began by experimenting with different models, including decision tree, random forest, linear regression, KNN, and gradient boosting algorithms.

One of the key challenges we faced was determining whether to train one model that predicts for any restaurant or to train a separate model for each restaurant. To make this decision, we performed an analysis of the data, examining the similarities and differences between each restaurant's sales patterns. We also trained different models and compared the performance for both approaches. We found that although there were some similarities between the restaurants, there were also significant differences that suggested that training separate models for each restaurant would be more effective.

With this in mind, we proceeded to train separate models for each restaurant, using a combination of decision trees and random forests. To ensure that our models were optimized for performance, we performed hyperparameter tuning, experimenting with different combinations of parameters such as learning rate, number of trees, and maximum depth of the decision trees. After

training our models, we evaluated their performance using mean squared error (MSE). We found that our models performed well, achieving low MSE scores. This suggested that our models were effective at predicting future sales for the different restaurants in Uppsala.

Feature Engineering

Holidays

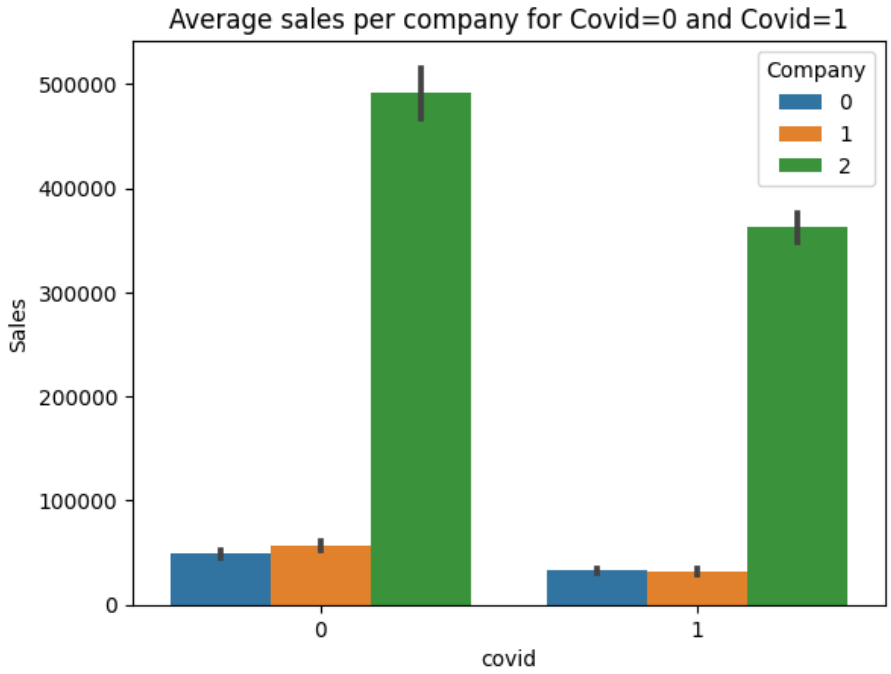
Since the only column in the data set we have is which restaurant sells how much, we had to add new features to our dataset before making forecasts using machine learning algorithms. In this context, we started by adding the holidays first. Adding holiday data to the sales data can be useful for improving the accuracy of the sales forecast. Holidays are special occasions that can have a significant impact on the sales of restaurants, as they can lead to changes in customer behavior, such as increased or decreased demand.

By including holiday data, the model can learn to recognize these patterns and adjust the forecasts accordingly. Holiday data can also be helpful in capturing seasonality, which is a common phenomenon in restaurant sales. Seasonality refers to the predictable patterns in sales that occur during different times of the year. Holidays can be an important factor in these seasonal patterns, as they can serve as indicators of when these patterns are likely to occur.

Covid-19

Marking the dates of Covid-19 in a restaurant's sales dataset can be extremely useful in several ways. Covid-19 has had a significant impact on the restaurant industry, and its effects can be seen in the sales data. By marking the dates of Covid-19 in the sales dataset, we can identify the impact of the pandemic on the restaurant's sales and make more accurate forecasts.

Firstly, Covid-19 has caused significant changes in customer behavior, which can be reflected in the restaurant's sales data. For example, during the pandemic, many customers refused to go out. By marking the dates of Covid-19 in the sales data, we can identify when these changes in customer behavior occurred and adjust our forecasts accordingly. Secondly, Covid-19 has caused significant disruptions in the supply chain, leading to shortages of certain food items. By marking the dates of Covid-19 in the sales data, we can identify when these shortages occurred and adjust our forecasts accordingly.



The plot above shows the average sales data for each restaurant during the Covid and non-Covid times. The plot indicates that sales during the Covid period were significantly lower than sales during the non-Covid period. This observation is in line with what we know about the impact of the pandemic on the restaurant industry. The pandemic has had a significant impact on the restaurant industry, and its effects are evident in the sales data. By including the Covid column in our forecasting algorithms, we can adjust our forecasts based on the impact of the pandemic on the restaurant's sales.

Date Conversion

Although date as a string makes sense for us humans, it doesn't make much sense for machine learning models. In this respect, we thought that we should take the string date and extract different features from it.

- The DayOfYear column can help capture any seasonal patterns in the data, as it represents the day of the year (e.g., December 31st would be day 365). This can help the model recognize any trends that occur around specific times of the year.
- The Quarter column can provide additional information on the seasonality of sales. This column represents the quarter of the year (1-4) in which a specific date falls, and can help the model identify patterns that occur during different parts of the year.
- The WeekOfYear and DayOfWeek columns can also provide insights into weekly patterns. The WeekOfYear column represents the week number of the year (1-52), and the DayOfWeek column represents the day of the week (0-6, with 0 being Monday and 6 being

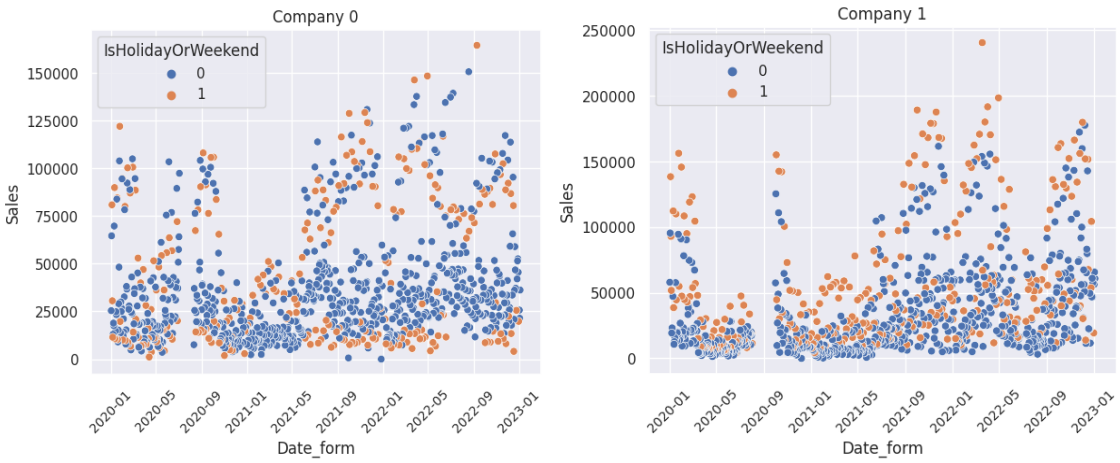
Sunday). This information can help the model recognize any weekly patterns that occur in the sales data.

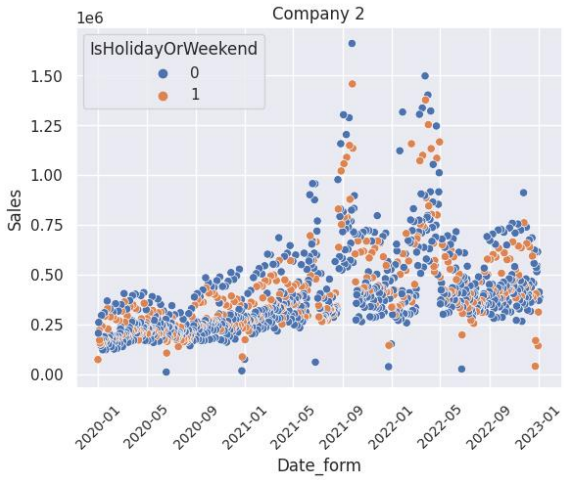
- The Day, Month, and Year columns can help the model identify any long-term trends in the data. For example, the Month column can help the model recognize any patterns that occur during specific months (e.g., sales may increase during the summer months).
- The IsWeekend column can be useful in capturing any changes in customer behavior that occur on weekends. This column is a binary indicator (1 for weekends and 0 for weekdays) and can help the model recognize any changes in demand that occur on weekends.

By including these columns, we believe that the models can capture the seasonality effects better.

Holiday Information Update

We observed that the weekends and the holidays are similar in terms of the sales. Therefore we decided to combine the IsHoliday column and the IsWeekend column to create the IsHolidayOrWeekend column. This column is created by taking the logical OR of the IsHoliday and IsWeekend columns, which results in a binary indicator for whether a particular date falls on a holiday or a weekend. This information can be useful in identifying periods of time when sales may be different from typical weekdays, such as weekends or holidays. The DataFrame is then modified by dropping the original IsHoliday and IsWeekend columns, which are no longer necessary since their information is now captured in the IsHolidayOrWeekend column.





As can be seen from the plots, the sales of the restaurants are below or above the normal values during the holidays, but mostly higher sales values were obtained during the holidays compared to the non-holiday days. This makes us think that this feature may play a critical role in our predictions.

Season

A new column that indicates the season of the year added to the DataFrame. This is achieved by iterating over the Month column and assigning a value of 1, 2, 3, or 4 to the Season column, depending on which season the corresponding month falls into. For example, if the month is March, April, or May, the value assigned to the Season column is 1, representing spring. Similarly, if the month is June, July, or August, the value assigned to the Season column is 2, representing summer.

The same approach is used to assign values to the Season column for autumn and winter months. This new column can be useful for the sales forecast model, as it can help capture any seasonality patterns that may exist in the data. For example, sales may be higher in the summer months due to warmer weather, or due to holiday events and people tend to go out. By adding the Season column, the model can more easily identify these patterns and adjust its forecasts accordingly.

As can be seen from the pie charts above, every restaurant has a different distribution of sales among different seasons. This makes us think that this feature may increase the performance of the models

Weather Data

Adding historical weather data to a restaurant sales forecasting model can be useful for several reasons:

- Weather can influence consumer behavior: The weather can affect people's moods, preferences, and decision-making processes. For example, on a hot summer day, people may be more likely to choose a cold beverage or a salad over a hot soup or stew. Similarly, on a rainy day, people may be less likely to dine out and may prefer to stay at home. By incorporating weather data, the model can capture these effects and make more accurate predictions.
- Seasonal trends: Weather patterns can be strongly correlated with seasonal trends in restaurant sales. For example, during the summer months, people may be more likely to dine out and enjoy outdoor seating, while during the winter months, people may prefer cozy indoor dining. By including weather data, the model can identify these seasonal patterns and adjust its predictions accordingly.
- Capacity planning: Restaurants often need to adjust their staffing and inventory levels based on expected sales volume. By incorporating weather data, the model can provide more accurate forecasts of expected customer traffic, allowing restaurants to better plan for staffing and inventory needs.
- Historical trends: Historical weather data can provide additional context for sales trends. For example, if sales were particularly high on a particular date, it may be useful to know that the weather was particularly nice on that day. This information can help restaurants make more informed decisions about marketing and promotions.

In order to combine the weather data with our data frame, we need to convert the date column to the format that we are using in our train dataframe.

Train-Test Split

We have used the train_test_split function from the sklearn.model_selection module to split our training data into training and validation sets. The function splits the data randomly into two subsets, with a specified test size, which in this case is 20%. The random state parameter is set to 123, ensuring that the random split is reproducible. The original training data had 2468 records and 12 features. After splitting, we have 80% of the data (1974 records) assigned to the training set and 20% (494 records) assigned to the validation set. Additionally, we have also split the labels into training and validation sets, resulting in 2468 training labels and 617 validation labels.

The purpose of splitting the data into training and validation sets is to evaluate the performance of our machine learning model. By training the model on a subset of the data and validating its performance on another subset, we can estimate how well the model will perform on new, unseen data. By splitting the data into training and validation sets, we can also prevent overfitting, which occurs when the model is too complex and memorizes the training data rather than learning general patterns that can be applied to new data. By using a validation set, we can monitor the model's performance on new data and adjust its complexity if needed.

Machine Learning Models

In our project, we conducted several experiments to explore various machine learning algorithms and identify the ones that would be most effective in solving our problem. This experimentation phase allowed us to gain insights into which models were worth pursuing further, rather than wasting time and resources on models that might not be well-suited to our needs.

Our first experiment involved the Linear Regression algorithm. We applied the base algorithm without any tuning to our validation set and evaluated its performance. Unfortunately, the mean squared error score of this model was quite high, coming in at around 155k. This result was not satisfactory, and it led us to question whether Linear Regression was the right approach for our specific problem.

For our second experiment, we chose to explore the KNN (K-Nearest Neighbors) algorithm. We began by training a model using the base version of the algorithm, and we then evaluated its performance using our validation set. Unfortunately, the results were not very promising. The mean squared error score we obtained was approximately 225k, which was significantly worse than the results we obtained with Linear Regression. While it is possible that we could have improved the performance of the KNN model through hyperparameter tuning, we decided to move on to other algorithms, as we felt that the potential of this model was limited. Although KNN is a commonly used algorithm for various machine learning problems, it might not be well-suited to our specific problem, and it is important to consider the suitability of the model for the specific task at hand.

In our quest to find the most suitable algorithm for our project, we decided to experiment with the Decision Tree algorithm. We started by training a model using the base version of the algorithm and then evaluated its performance on our validation set. We were pleased to find that the mean squared error score we obtained was approximately 118k, which was significantly lower than the scores we obtained with Linear Regression and KNN. This result was encouraging, and it suggested that the Decision Tree algorithm might be a promising candidate for our problem.

After achieving good results with the Decision Tree algorithm, we wanted to explore other algorithms that might perform even better. Since we saw ensemble learning in the lesson, we wanted to try some of these algorithms. One of the algorithms we decided to experiment with was the Random Forest algorithm. Random Forest is an ensemble learning algorithm that combines multiple decision trees to make more accurate predictions. We trained a model using the Random Forest algorithm and evaluated its performance on our validation set. We were pleased to find that the mean squared error score we obtained was approximately 72k, which was significantly lower than the scores we obtained with both Linear Regression and KNN, and even better than our previously successful Decision Tree algorithm. This result was very promising, and it suggested that the Random Forest algorithm was an even better candidate for our problem.

We continued our search for even better algorithms by exploring the XGBoost Regressor. XGBoost is a tree-based learning algorithm that combines multiple decision trees. These trees are used to predict the output variable using the features in the dataset. XGBoost builds each tree to minimize the prediction errors of the previous tree. This results in better forecasting performance as a result. So, XGBoost is a powerful ensemble learning algorithm that uses a gradient boosting framework to train models. We trained a model using the XGBoost Regressor algorithm and

evaluated its performance on our validation set. We were excited to see that the mean squared error score we obtained was approximately 59k, which was even lower than the scores we obtained with Random Forest. This result was very promising, and it suggested that the XGBoost Regressor algorithm was the best candidate for our problem so far.

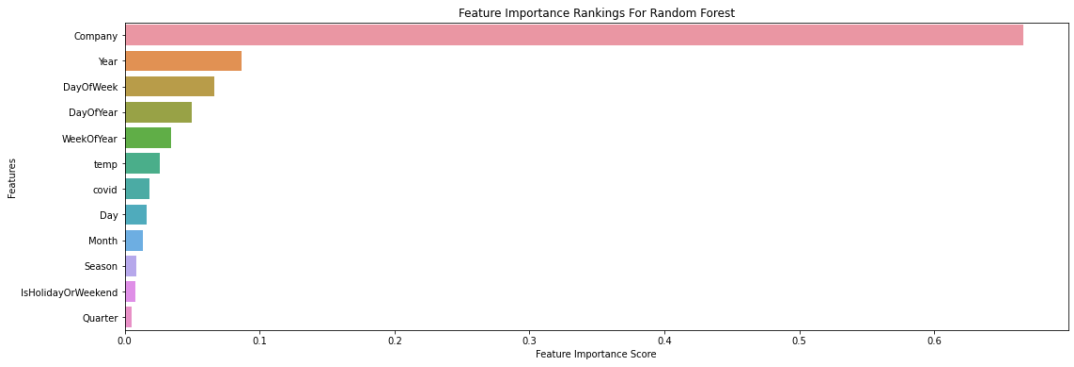
By testing different algorithms and analyzing their performance, we were able to make informed decisions about which models to focus on in our subsequent experiments. This process helped us save time and resources by avoiding models that were unlikely to yield satisfactory results. Ultimately, this allowed us to streamline our model development process and prioritize the models that were most likely to be successful in meeting our objectives. In the light of these results, we aimed to concentrate on the Random Forest and XGBoost algorithms and to achieve much better results by performing hyperparameter tuning on these two models.

Hyperparameter Tuning

Random Forest Regressor

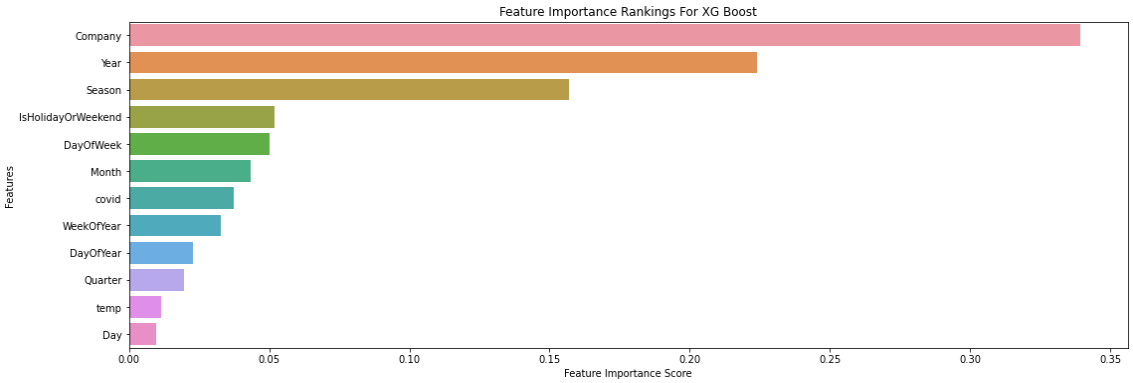
To perform the hyperparameter tuning, we used the RandomizedSearchCV function from scikit-learn. This function randomly selects a set of hyperparameters from a specified distribution and fits a model on the training data using each set of hyperparameters. It then evaluates the performance of each model using cross-validation on the validation set and selects the best performing model based on a specified scoring metric, in this case, negative mean squared error.

After performing hyperparameter tuning, we obtained the best model and best hyperparameters, which we then trained on the combined training and validation sets. Finally, we used this model to predict the sales for the test set and calculated the mean squared error (MSE) to evaluate its performance. Our achieved MSE score of 67k shows that our tuned model is more accurate than the original model without hyperparameter tuning. As a result, our feature importance graph was as follows for Random Forest Algorithm:



XG Boost Regressor

We were impressed with the performance of XG Boost Regressor and decided to fine-tune the hyperparameters to further improve the model's accuracy. Again, we used a randomized search approach to find the optimal combination of hyperparameters. The resulting MSE score was 60k. As a result, our feature importance graph was as follows for XG Boost Algorithm:



After that, we tested our algorithms powered by Hyperparameters on the Test dataset. We got a value like 49K MSE (Mean Squared error) for Random Forest and 51K MSE for XG Boost. However, we thought we could still develop these values. So, we wanted to explore what would happen if we developed a separate model for each company.

Separate ML algorithms for each company

Firstly, we separated both our test data and train data for each company. Then we re-run our Random Forest and XG Boost algorithms for each company. For a this approach, we divided our train set into train_0_x, train_1_x,train_2_x and train_0_y,train_1_y, and train_2_y. Also, we used the validation set for each company such as val_0_x, val_1_x, and val_2_x. For instance, the process we made for company number 1 was as follows in detail.

For the random forest model, code trains a random forest regression model on a training set (train_1_x, train_1_y), predicts the output for a validation set (val_1_x), calculates the mean squared error (MSE) between the predicted and actual values, and tunes the hyperparameters using randomized search cross-validation on the validation set. The best hyperparameters and best model are then used to predict the output for a test set (test_1_x), and the MSE between the predicted and actual values is calculated. Finally, the model is trained on the combined training and validation sets and used to predict the output for a separate test set (test_1_x).

Three random forest models were trained using the same methodology for each company, where the training and validation sets were split for each company, and hyperparameters were tuned using randomized search cross-validation on the validation sets. The best model and best hyperparameters were then used to predict the output for a separate test set for each company.

For the XGB model, code trains an XG Boost regression model on a training set (train_1_x, train_1_y), tunes the hyperparameters using randomized search cross-validation on the validation set, and evaluates the performance of the best model on the validation set by calculating the mean squared error (MSE) between the predicted and actual values. The best model and best hyperparameters are then used to predict the output for a separate test set (test_1_x). Finally, the model is trained on the combined training and validation sets and used to predict the output for a separate test set (test_1_x).

Three XGB models were trained using the same methodology for each company, where the training and validation sets were split for each company, and hyperparameters were tuned using randomized search cross-validation on the validation sets. The best model and best hyperparameters were then used to predict the output for a separate test set for each company.

After comparing the results and experimenting on the test set, we found that the XGBoost model was overfitting, and the Random Forest model was performing better. Therefore, we chose the Random Forest model as our final model and made predictions. Our model achieved an MSE of 32K, which placed us second in the Kaggle leaderboard.

Appendix:

You can find the all CSV and data files with our project python notebook using this link:
https://drive.google.com/drive/folders/139vZpcPTkj0U3Yvk1ntnKK_bkVuS_Khg?usp=sharing