AI Systems and Infrastructure Mini project

1st Anders Jensen Klynge
Dept. of Architecture, Design and Media Technology
Aalborg, Denmark
aklyng23@student.aau.dk

2nd Jacob Koreny Selbo
Dept. of Architecture, Design and Media Technology
Aalborg, Denmark
jselbo23@student.aau.dk

3rd Martin Tran Pham
Dept. of Architecture, Design and Media Technology
Aalborg, Denmark
mpham22@student.aau.dk

***Abstract*—In this mini project we designed and deployed a machine learning solution that can forecast the quantity of pizzas that a pizza restaurant sells. Based on a data analysis, time-based features, such as hour, day of week, and month, were extracted for use in the model training. To determine the best pipeline, we experimented with lag features and removal of outliers on multiple machine learning models: Elastic Net, SVM, and XGBRegressor. Based on the experiment results, the best model was XGBRegressor without lagged features and outlier-removal. Having determined the best pipeline, we trained and tested XGBRegressor, which achieved a mean RMSE and MAE of 6.55 and 4.95, respectively. With the model trained, it was deployed for inference. This was done using Docker, FastAPI, and Uvicorn, allowing the system to be run on nearly any system.**

I. Introduction

In this mini project, we were tasked to improve the sales of a pizza restaurant. In order to discover a way to achieve this, we had to gain insight into the provided data. This necessitated the use of data analysis. With the use of the knowledge gained during data analysis, we were able to find a way to increase sales, choose potential machine learning models, and extract relevant features. In accordance to the mini project requirements, we also created a machine learning solution and deployed the model.

The provided dataset contains detailed information about pizza orders, such as: pizza styles, quantities, pricing, dates, and times.

II. Data

This section covers our data analysis and used pre-processing methods. The provided dataset includes the following features: Pizza ID, Order ID, Pizza name ID, Quantity, Order date, Order time, Unit price, Total price, Pizza size, Pizza category, Pizza ingredients, and Pizza name. Additionally, the dataset is formatted as a Comma Separated Values (CSV) file. The dataset does not contain any missing values.

A. Exploratory Data Analysis (EDA)

Before we could design a machine learning solution to improve sales, we had to gain data insights by performing an EDA. To analyze the data, we imported and manipulated the data using the Python library PySpark as it allows for large-scale data processing. Moreover, the libraries Matplotlib and Seaborn was also used to plot and visualize the data. As PySpark DataFrames do not allow for data visualizations with Matplotlib or Seaborn, Pandas' DataFrame was used as a medium. Furthermore, Pandas' datetime-function was used for time conversions.

B. Line plots

Due to the pizza restaurant being open between 10 and 23, there will naturally be a variance in sales, both hourly and per weekday. To gain insight in this, a line plot was used to visualize the total quantity of sales per hour for each weekday. This shows where sales potentially spiked and on which weekdays pizzas were sold the most.

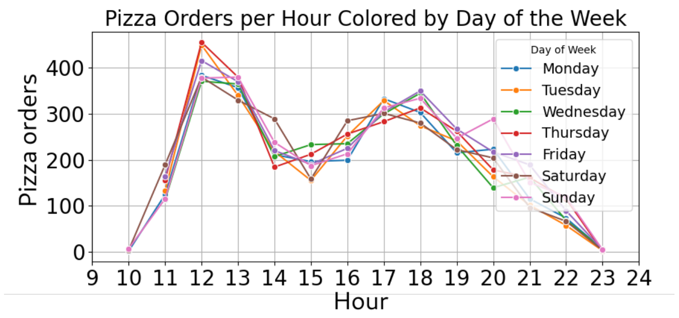


Fig. 1. Line plot of total sales per hour for each weekday. Regardless of weekday, there seems to be a pattern between the hour of the day and sold pizzas.

As seen in Figure 1, the total sales of pizzas did not depend on the weekday but rather the hour of the day. Furthermore, the hourly sales per weekday did not vary significantly.

We also wanted to gain insight into whether any specific pizza size was preferred at any point during the day. This was done with another line plot of quantity of pizza sizes during the day.
As seen in Figure 2, all pizza sizes follow the same pattern over the course of an average day. However, the pizza size L was, in general, ordered more than the other sizes.

C. Bar chart

The pizzas have a variety of different ingredients and toppings. It could therefore be relevant to analyze which ingredients were the most commonly used. This was done by creating a

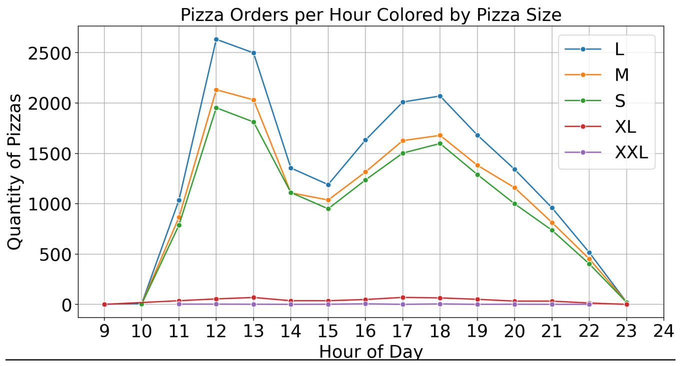


Fig. 2. Line plot with quantity of pizza sizes over the course of the day. There does not seem be a change in trend between the sizes. Rather, only the total amount differs.

bar chart. As the restaurant offered many ingredients, only ingredients with a quantity larger than 6000 has been visualized, see Figure 3.

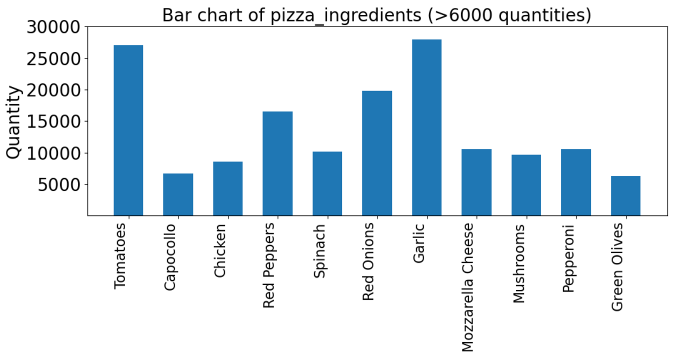


Fig. 3. Bar chart of the different ingredients and the quantity. Only ingredients with a quantity over 6000 are shown.

As seen in Figure 3, the four most popular ingredients were (from most popular to least popular): garlic, tomatoes, red onions, and red peppers. Moreover, the difference in quantity between garlic and tomatoes was insignificant.

As the quantity of sold pizzas are directly linked to the sales, it was relevant to explore if there was a difference in the distribution of sold pizza sizes per month. For example, if large pizzas were the most commonly sold pizza size during December. Figure 4 shows a bar chart of the sold pizza sizes and their respective quantities per month.

As seen in Figure 4, there were some minor trends in the quantity of sold pizza sizes per month. Some months, such as May, had higher disparity between L and M/S than other months, such as July. Furthermore, May and December had the lowest quantity of pizzas sold, but since the dataset only contains data of one year, it is unknown whether this is a general trend for those months. Moreover, the large pizzas were the most common pizzas in every month, followed by medium, small, XL, and then XXL.

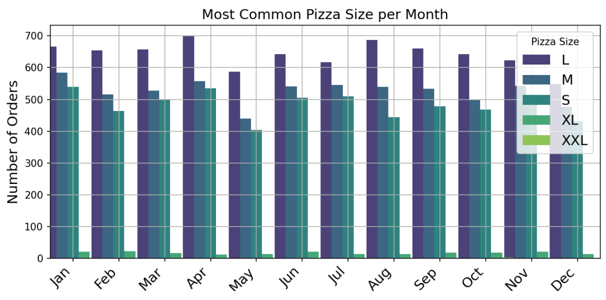


Fig. 4. Bar chart of the sold pizza sizes per month and their respective quantity. There only appears to be minor trends in the plot.

D. Pie Charts

As the restaurant served different pizza sizes: small, medium, large, extra large, and XXL, a pie chart was used to visualize the sales share by the different sizes. XL and XXL were combined because XXL only accounted for 0.1% of the total sales.

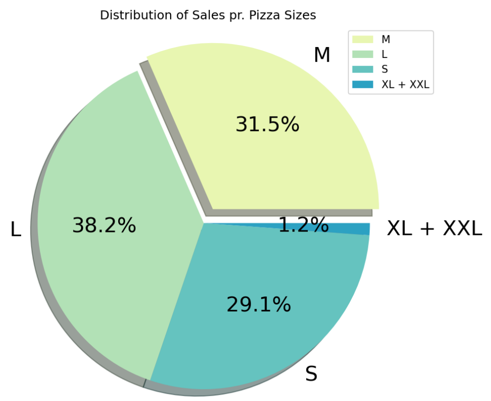


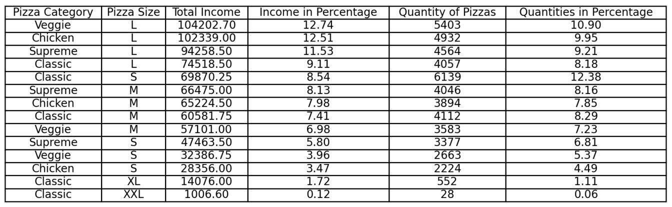
Fig. 5. Pie chart of the different sizes and their respective sales share. Large pizzas are the most common, followed by medium, small, and then XL+XLL.

Based on Figure 5, the pizza sizes from most ordered to least ordered were: L (38.2%), M (31.5%), S (29.1%), and XL combined with XXL (1.2%). Moreover, there was almost an even distribution between L, M, and S in terms of sales share, within 9.1%, and only a few customers order the XL or XXL pizza sizes.

E. Pivot table

The restaurant served four different categories of pizzas: Chicken, Classic, Veggie, and Supreme. All four categories of pizza could be served as one of the aforementioned sizes. A pivot table was used to compare each combination of pizza size and category to see their sales income and quantities sold. The income and quantity in percentage per combination was also added, see Figure 6.

As seen in Figure 6, all four types of pizza categories with an L pizza size contributed more to the total income than S Classic pizza, even though S Classic pizzas were sold in larger quantities. Moreover, for all categories with a size of S



Pizza Category	Pizza Size	Total Income	Income in Percentage	Quantity of Pizzas	Quantities in Percentage
Veggie	L	104202.70	12.74	5403	10.90
Chicken	L	102339.00	12.51	4932	9.95
Supreme	L	94258.50	11.53	4564	9.21
Classic	L	74518.50	9.11	4057	8.18
Classic	S	69870.25	8.54	6139	12.38
Supreme	M	66475.00	8.13	4046	8.16
Chicken	M	65224.50	7.98	3894	7.85
Classic	M	60581.75	7.41	4112	8.29
Veggie	M	57101.00	6.98	3583	7.23
Supreme	S	47463.50	5.80	3377	6.81
Veggie	S	32386.75	3.96	2663	5.37
Chicken	S	28356.00	3.47	2224	4.49
Classic	XL	14076.00	1.72	552	1.11
Classic	XXL	1006.60	0.12	28	0.06

Fig. 6. Pivot table of income and quantity of pizzas for each combination of pizza category and size. The most popular pizza category seems to depend on the pizza size.

or M, other than Chicken M, the income in percentage was less than the quantity in percentage. Furthermore, quantity of sold pizzas per category types depended heavily on pizza size, e.g. Veggie was the most sold L size pizza but the least sold M size pizza.

F. Conclusion of EDA

The main take-aways from the EDA are presented below:

- There was a pattern between sold pizzas and hour of the day. Most pizzas were sold around 12 o'clock, followed by 18 o'clock. However, the day of week had no significant influence on the sales.
- The main ingredients, used in their pizzas, were tomatoes, garlic, red onions, and red peppers. The rest were used in substantially lower quantities.
- The disparity between L and M/S size pizzas sold was different depending on the month. Furthermore, some months had considerably higher pizza sales for all sizes. Large pizzas were consistently the most sold pizza size per month.
- The most ordered pizza sizes (from most to least) were: large, medium, small, and XL + XXL. Moreover, XXL only accounted for 0.1% of the sales.
- The quantity of sold pizza types depended on the size, indicating that they correlated.
- In general, the pizza sizes L, XL, and XXL had a higher income in percentage than quantities in percentage and vise versa for S and M.

Based on these main take-aways, we could conclude that the number of pizza orders were time-dependent. We therefore decided to forecast the quantity of pizzas the restaurant will make, with the aim of improving the pizza sales. By forecasting the amount of pizzas they will make, the restaurant can optimize their storage, purchase of ingredients, and the number of workers required for the day.

III. Pre-processing

Certain data specific quirks were discovered during data analysis. Therefore, we did the following pre-processing steps:

- Extracted time-based features: hour, day of week, and month.
- Total quantity was set as target-class. Total quantity was created by combining the total quantity of orders per hour.
- Lag features of sales, quantity, unique pizza sizes, and unique pizza types were made with a lag of one hour, three hours, and five hours.

Time-based features were extracted because of the clear time dependency from the other features. These time-based features included: hour, day of week, and month. Other than time-based features, different lag features of pizza sales, pizza quantity, pizza sizes and pizza types were added as they could be correlated to the prediction, potentially improving the prediction of the quantity of pizza sales. After pre-processing, a new CSV file containing the extracted/added features was created and sorted chronologically.

A. Train/Test Split

To split the time series data, the TimeSeriesSplit class from scikit-learn was used. It allows for the data to be split into non-overlapping, chronological splits of train and test data, which can be iterated over to extract the performance for each split, indicating how well the model is able to generalize over time. We chose to split the data into five even splits, resulting in the last split being a 80/20 train-test split of the entire dataset.

IV. Experiments

Experiments were performed on the pre-processed dataset before the final test. The aim of the experiments were to identify the best pipeline to use for the final test. These experiments evaluated the influence of lag features (across multiple machine learning models) and outlier-removal. The pipeline for the final test was chosen based on the best experiment results. In the experiments, the quantitative metrics Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) were used. The chosen prediction feature was set to be total quantity. Furthermore, all the experiments were conducted on the first four splits to ensure that the last split was untouched for the final test.

To evaluate the influence of lag features, three models were used: Elastic Net, SVR, and XGBRegressor. They were trained and evaluated on three different sets of lag features: 1. No lag features, 2. Lagged sales and quantity, and 3. Lagged sales, quantity, unique sizes, and unique types. Since pre-processing combined sales per hour, all lag features received three versions: one hour lag, three hour lag, and five hour lag. Table I presents the RMSE / MAE results. The best results are marked in **bold**.

Lag features	Elastic Net	SVR	XGBRegressor
None	7.67 / 5.89	7.81 / 5.58	**6.74 / 5.01**
Sales and quantity	7.57 / 5.77	7.54 / 5.48	6.80 / 5.07
Sales, quantity, unique sizes and unique types	7.57 / 5.78	7.52 / 5.57	6.78 / 5.09

TABLE I
TABLE SHOWING THE RMSE / MAE PER MODEL WITH THE USE OF DIFFERENT LAG FEATURES. ALL RMSE / MAE VALUES WERE ACHIEVED IN SPLIT FOUR.

As presented in Table I, XGBRegressor outperformed both Elastic Net and SVR. Furthermore, although Elastic Net and SVR achieved slightly better RMSE and MAE when using sales and quantity as lag features, XGBRegressor performed best without lag features. Based on these results, we chose XGBRegressor, with no lag features, as our model. The XGBRegressor was then evaluated when outliers were removed.

To evaluate the effect of outliers, an IQR-based outlier threshold was used with a weight of 1.5, see Figure 7. On the fourth split, the XGBRegressor yielded an RMSE and MAE of 6.83 and 5.04, respectively, when outliers were removed. When outliers were not removed, the XGBRegressor achieved an RMSE and MAE of 6.74 and 5.01, respectively. Based on this, removing outliers negatively impacted the performance of the model. This might be because XGBRegressor is an ensemble model made up of decision trees; it may be more robust to outliers. For this reason, outliers were not removed in the final pipeline.

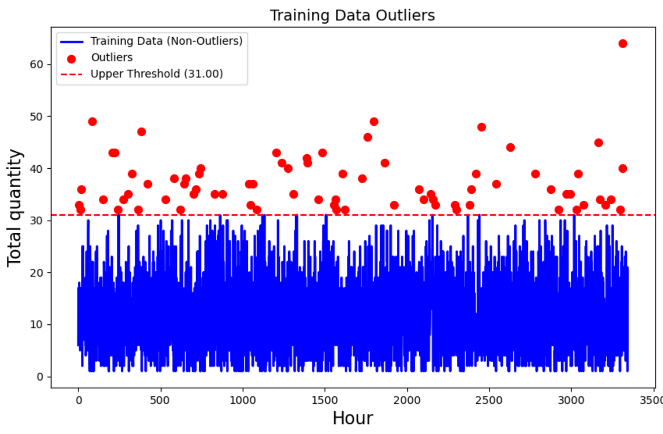


Fig. 7. Plot of the training data and its IQR-based outliers. The outliers are marked as red circles.

V. Method

On the basis of the experimental results, the chosen machine learning model was XGBRegressor without lagged features and without outlier-removal. XGB stands for eXtreme Gradient Boosting [4]. The gradient boosting machine learning algorithm is an ensemble model of decision trees. Each tree is a weak learner, only slightly better than random chance. Using the residual error from the current ensemble, the algorithm optimizes new trees with a differentiable loss function through gradient decent, i.e. every weak learner is trained to predict the residuals of the current ensemble, and the residual-predictions minimize the loss. The tree is then scaled by a learning rate and added to the ensemble. XGB is a variation of gradient boosting, where, for example, L1 and L2 regularization are added, yielding better performance and efficiency. XGBRegressor is specialized for regression and is therefore a suitable choice for our time series forecasting problem. Since XGBRegressor is tree-based, it is not necessary to normalize the features.

VI. Final Results

For each split in TimeSeriesSplit, the RMSE and MAE scores were saved, see Table II. To visualize the model's predictions, two plots were set up for the last split: predicted vs. actual values and residuals, which can be seen on Figures 8 and 9, respectively. For better visibility, only the last 100 data examples are visualized when comparing the predicted values to the ground truth, but the plot of the entire split can be seen in the Appendix.

Split	RMSE	MAE
1	6.84	5.12
2	6.00	4.55
3	6.33	4.87
4	6.74	5.01
5	6.82	5.22

TABLE II
RMSE and MAE scores for each split

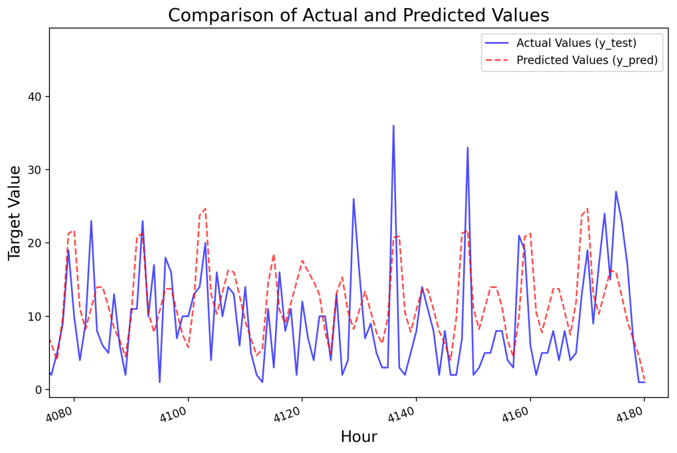


Fig. 8. Plot of the last 1̃00 actual values (blue) and predicted values (red). Observe the seemingly continuous prediction range

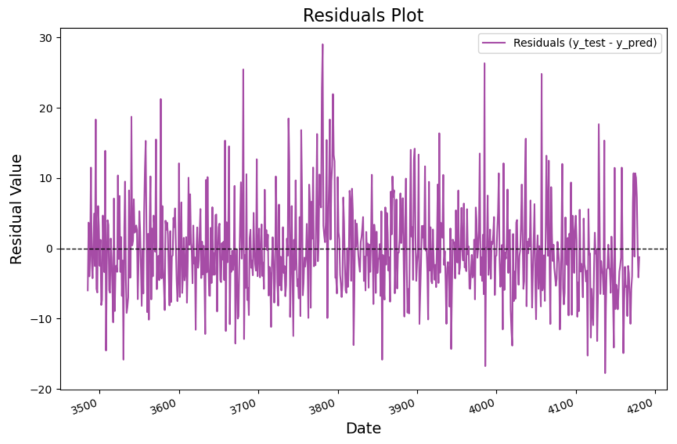


Fig. 9. Plot of residuals calculated with y_test and y_pred. Note the large residual variation

Based on Table II the mean RMSE and MAE were 6.55 and 4.95, respectively. This shows that the model is off by approximately 5 pizzas when predicting the total quantity of pizzas sold per hour.

Based on Figure 8, the model can predict the overall trend well. However, the model is not effective at predicting sudden spikes. This trend can also be seen on the full test set, as shown on Figure 10. Additionally, a spike in residuals can be observed around 3800 hours, see Figure 9. This can possibly indicate that some external event happened, which is not specified in the data. This event can also be observed on Figure 10.

VII. Deployment

In accordance with the curriculum, the machine learning solution has to get deployed. To fulfill this requirement, we packed the system, made a container for the package, and connected the container to an endpoint. This enables real-time or batch inference with the model. The project made use of a monolithic architecture, i.e. the entire application is one unit, since the problem is fairly small. The following Sections explains the deployment in details.

A. Model Packaging

After the machine learning model was trained and evaluated, it was necessary to package the different software artifacts, e.g. the model and script(s), to ensure portability. That is, the artifacts can be moved, as one or more files, and replicated in a different environment for testing, production, and/or deployment [1]. Doing this helps with model inference, where the model can process data in real-time, and interoperability, where two or more models or components can exchange information and learn from each other.
We chose to package our artifacts using the ONNX format, which uses the serialization method [1] [2]. Serialization converts data structures into a format that can be stored, e.g. in a file, and then transmitted to a different environment. The data structure is then reconstructed in the new environment. Although our model only used the scikit-learn framework, we still chose to use the ONNX format, ensuring interoperability between models with different frameworks, such as TensorFlow. This enables transfer learning, where a pre-trained model can be retrained using a different framework.

B. Container

When the artifacts have been packaged, the system was deployed. This can be done in a lot of ways, but is typically done using Docker images and later containers [1]. Containers are software units that composes code and its dependencies, thus standardizing it and allowing it to be used in different computing environments. This makes them very versatile and allows them be deployed on nearly all environments, given they support Docker. Due to their nature, they are both highly portable and interoperable, making them ideal for the system type that was created for this mini project. We used Docker and Docker Desktop to compile the Docker image into a container and Docker Desktop to manage the image and container, respectively. Docker was chosen due to its platform invariance and development simplicity, ensuring the system can be run on any system along with minimizing problems during development [1].

C. Endpoints

The last step in our deployment was endpoint management using an API. The endpoints are the interfaces through which users and other systems can interact with the deployed system [1]. For example: the "/predict" endpoint does model inference and the "/status" endpoint checks the system's status. The API facilitates this communication and provides access to the system's data and functionalities. An API can have several endpoints. We created a web service using the FastAPI framework as the API and Uvicorn as the server implementation. The web service enables sending requests and receiving responses from the system. This means that after deployment, a user can send an HTTP request for model inference to an endpoint on a server (that uses an API) and receive the result from a JSON response [3].

VIII. Conclusion

Based on the insights gained during data analysis, it was decided to predict the total quantity of pizzas sold per hour. As such, time-based features were extracted: hour, day of week, and month. Most of the trends seen during data analysis were time-depended. However, there were some minor correlations between some of the other features, such as pizza type and size. Based on our experimental results, XGBRegressor was chosen as our machine learning model. Furthermore, adding lag features or removing outliers worsened the model performance. In the final test, the XGBRegressor achieved a mean RMSE and MAE of 6.55 and 4.95, respectively, indicating that our model was, on average, off by approximately five pizzas. The trained XGBRegressor was then deployed using Docker, FastAPI, and Uvicorn.

References

[1] E. Raj, "Engineering MLOps", April 2021
[2] Microsoft, Baidu, and Amazon, "ONNX', 2024
[3] Sebastián Ramírez, "FastAPI', 2024, url: https://fastapi.tiangolo.com/features/. [Accessed: 11/19/2024]
[4] Chen, Tianqi and Guestrin, Carlos, "XGBoost: A Scalable Tree Boosting System", 2016

APPENDIX

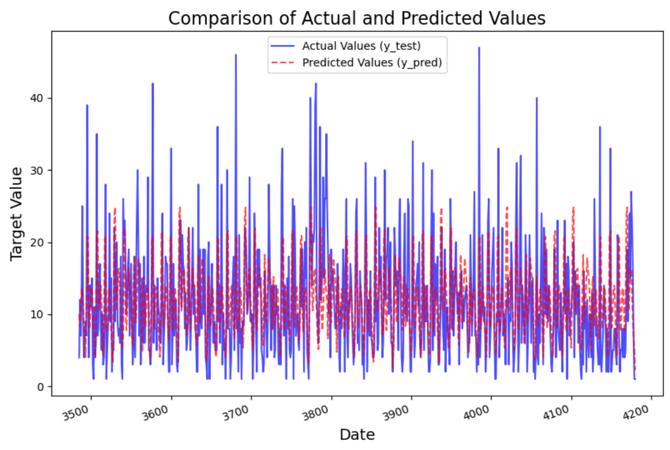


Fig. 10. Full plot of actual values (blue) and predicted values (red). Note the similar continuous predictions as seen on Figure 8