Data Cleaning

The initial data cleaning process was crucial to ensure that the dataset was reliable and suitable for analysis. First, I addressed any missing values by removing rows with NaN values, specifically targeting the **Temperature** feature and the feature that is created in the feature engineering section, **1-Day Lagged Price** (only first row was NaN) columns. This allowed me to maintain completeness across all features without introducing bias or gaps in the data. Then, I examined the **Price** and **Volume** columns. In particular, I removed any rows that contained negative values because their existence in the data would be unrealistic in the context of energy trading prices and transaction volumes. Furthermore, I checked the consistency of the use of decimal separator, whether there are any empty values in any of the features and whether there are any duplicate rows/records.

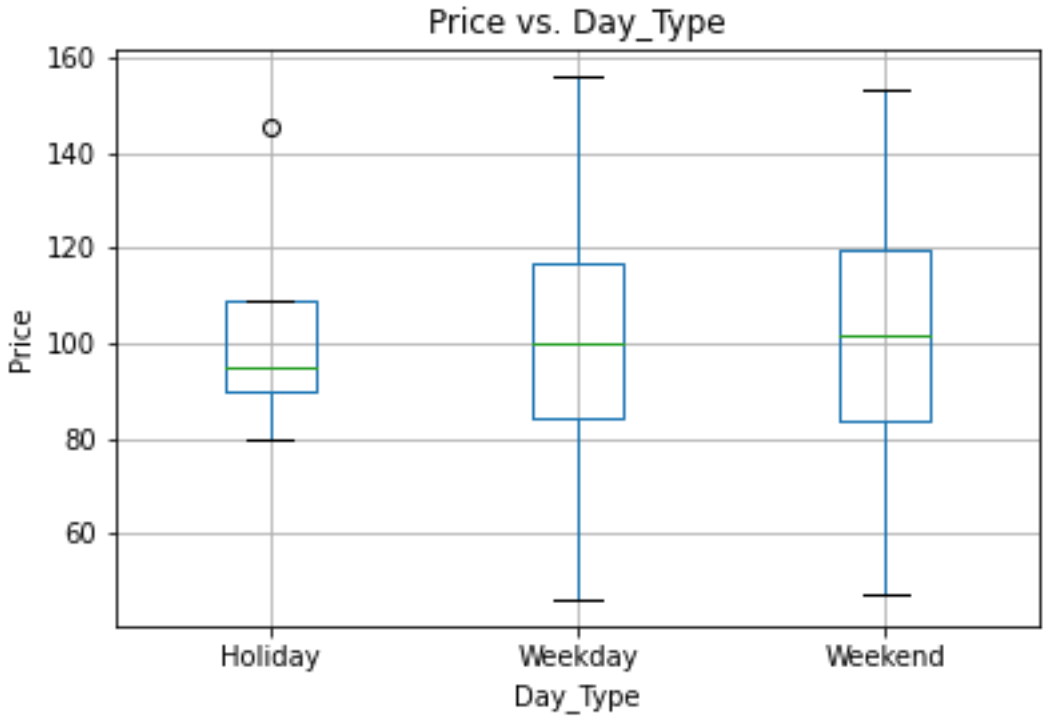
Next, **Date** column was converted to datetime64 format which is the conventional format of a date object. This allowed me to extract time-based information efficiently during the feature engineering phase. I also encoded many features as categorical variables. This prevented the model from misinterpreting these features as having an inherent ranking specific to each of them. As mentioned before, I checked for and removed any duplicate values based on the **Date** column (which is enough to detect any duplicate), hence treating it as a **primary key**. This step ensured that each entry in the dataset was unique.

Feature Engineering

Feature engineering enriched the dataset and the model captured patterns and trends more accurately, displaying improved performance. After plotting the dependent variable against the original independent variables, I observed very apparent relationships between my dependent variable Price and the independent variables Volume and Date. Thus, I began by extracting time-based features from the **Date** column. This included the **Season** feature, representing broader seasonal patterns (Winter, Spring, Summer, Autumn) that could influence energy demand and prices. I also created the **Day_of_Month** feature, which provided a numeric representation of the day within each month and may have captured any recurring patterns within a month, such as monthly billing or inventory cycles. Moreover, I modified the **Day_Type** feature so that it would take a value that may range from a weekday, weekend or holiday. In other words, I dissected the “Weekday” value into the 5 days of the weekday after observing significantly more price volatility and transaction volume in weekdays compared to “Holiday” and “Weekend”:

```
Variation of Price and Volume within each Day_Type category:
              Price       Volume
Day_Type
Holiday   28.580780    87.618122
Weekday   59.600678  1232.812724
Weekend   25.486220   971.336296
```

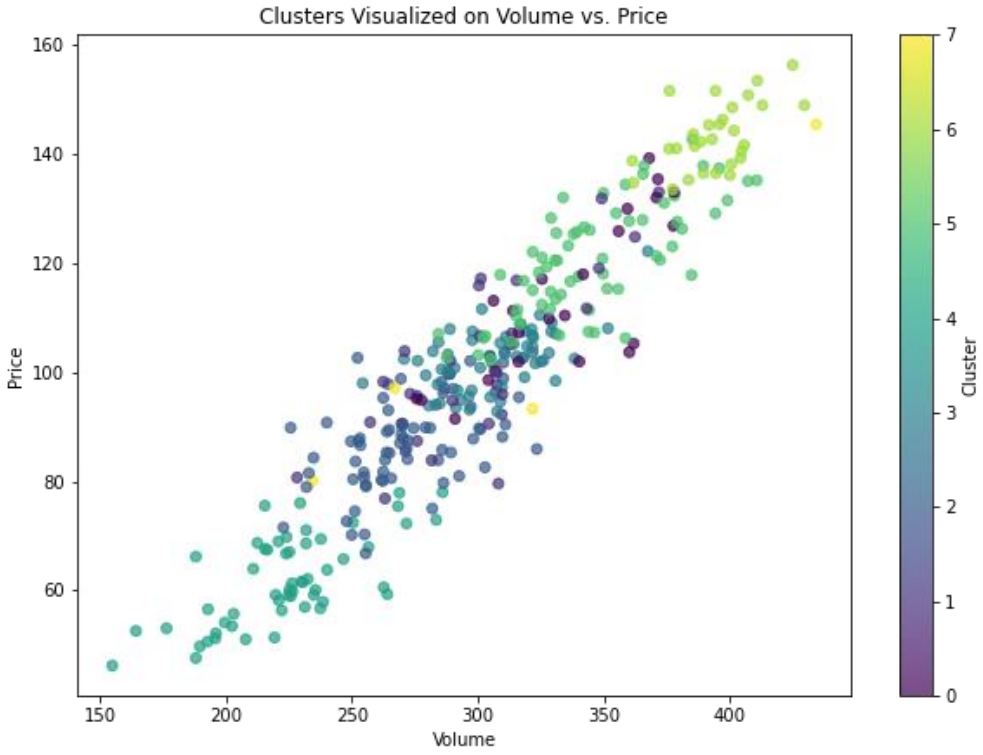
Another reason for considering each weekday individually was the similarity of the box plots of both “Weekday” and “Weekend”, which was discovered during the exploratory analysis stage. As can be observed, the median price, interquartile ranges (IQR), and the maximum and minimum values of them are quite similar albeit the fact that price volatility is generally much higher on weekdays (59.60) than on weekends (25.49).



This contradictory pair of findings directly implied that weekdays **must** have distinctive Interquartile Ranges (the spread of the middle 50% of the data), which was another reason for analyzing each weekday individually.

Additionally, I introduced lagged and rolling features to incorporate temporal dependencies. The **1-Day Lagged Price** feature brought in the previous day's price as a predictor, which can be crucial in markets where prices often follow recent trends. To capture short-term in a more general and comprehensive manner, I added a **7-Day Rolling Average of Price**, which reduced noise from daily fluctuations and highlighted recent pricing trends.

One of the more advanced steps was the **clustering analysis** using K-Means clustering. By examining all of the features, I identified clusters that represented distinct patterns or market conditions in the data. I determined the optimal number of clusters (8) using the **Silhouette Score (see Appendix for the methodology)** and labeled each data point with its cluster assignment. This **Cluster** feature provided the model with valuable context, enabling it to recognize broader market conditions that might affect prices. Overall, these engineered features enriched the dataset so as to capture temporal patterns, interactions, and segment-specific nuances.

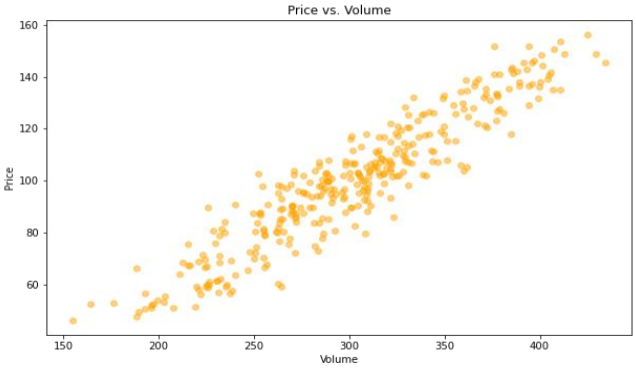


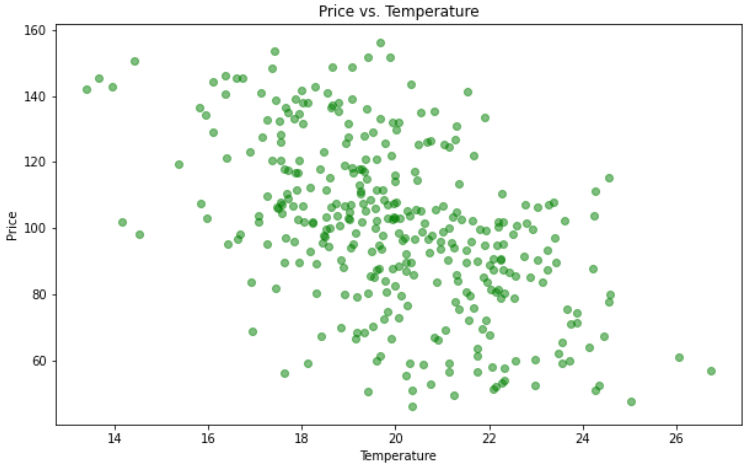
Model Selection

Given the nature of the data and the relationships observed, I selected **Support Vector Regression (SVR) with a linear kernel** as our primary predictive model. The exploratory analysis had shown that **Price** had predominantly linear relationships with key features such as **Volume** and **Time**. On the other hand, **Temperature** feature seemed to have a somewhat ambigious relationship with **Price** taking the shape of a mild negative correlation**.** This made SVR with a linear kernel a logical choice, as it captures linear dependencies effectively while offering flexibility through regularization. The results of the exploratory analysis can be observed below:

```
Pearson Correlation (linear relationships):
                Price    Volume  Temperature
Price        1.000000  0.936573    -0.514155
Volume       0.936573  1.000000    -0.498390
Temperature -0.514155 -0.498390     1.000000

Spearman Correlation (nonlinear monotonic relationships):
                Price    Volume  Temperature
Price        1.000000  0.929015    -0.513516
Volume       0.929015  1.000000    -0.493402
Temperature -0.513516 -0.493402     1.000000
```





As an alternative method and to make sure my interpretation of relationships between my features are fairly accurate, I considered SVR with a Radial Basis Function (RBF) kernel to test for potential non-linear patterns. However, the RBF kernel performed poorly, confirming that the relationships are primarily linear. This validation underscored the suitability of the linear kernel, which balances simplicity with flexibility and is well-suited for data with linear relationships. Further, the linear SVR model also offers regularization through the **C** parameter, helping to manage overfitting by balancing error reduction with model complexity. Additionally, We used **GridSearchCV** to optimize the SVR model with a linear kernel by systematically testing combinations of key hyperparameters, specifically **C** and **epsilon** (see Appendix for detailed explanation of GridSearchCV).

This approach was chosen over simpler linear models (e.g., as linear regression) and other more complex (and more computationally intensive) machine learning model because SVR with a linear kernel is robust, handles high-dimensional data well, and performs consistently in structured datasets with clear linear relationships **without** taking too much time to be trained as in the case of training a Random Forest Regressor.

Model Evaluation

For the evaluation, I used **Mean Absolute Error (MAE)**, **Mean Squared Error (MSE)**, and **R-squared (R^2)** metrics to assess the model's accuracy and generalizability. **MAE** provides an interpretable measure of average prediction error in real units of **Price**, helping us understand how closely the model's predictions matched the actual values. **MSE**, which penalizes larger errors, helps detect any significant deviations that could indicate outliers or instability in the predictions. Finally, **R^2** measures the proportion of variance in **Price** explained by the model, giving an indication of overall fit.

The tuned SVR model with a linear kernel yielded very positive results. The model achieved a **Test MAE** of 3.32, meaning that, on average, predictions were within 3.32 units of the actual price. The **Test MSE** of 18.03 confirmed minimal large errors[1], which is crucial for accurate pricing models. Finally, the **Test R^2** was 0.975, indicating that the model explained 97.5% of the variance in **Price**, a strong indication that the model captured the data's structure effectively.

This high level of accuracy, along with minimal overfitting, confirms that the SVR with a linear kernel was well-suited for this dataset. With a balance of interpretability, precision, and robustness, the model is now ready for reliable price predictions on unseen data.

[1] Since the Test MAE is 3.32 this implies there are at least a few absolute errors larger than 3.32. When one takes the square of numbers just slightly higher than 3.32, such as that of 4 (16) and of 5 (25), one can observe that these numbers are much higher than the square of 3.32 (11.02). Given this difference of the squares, we can conclude that the Test MSE is not too high, which implies the deviation of the errors from the mean error (3.32) is often smaller than 2.

APPENDIX

Understanding the Silhouette Score in Clustering

The Silhouette Score is a metric that quantifies the quality of a clustering solution by assessing how well each data point fits within its assigned cluster compared to other clusters. This metric provides insight into both cohesion (how similar data points within the same cluster are to each other) and separation (how different each cluster is from other clusters). A high Silhouette Score indicates that clusters are well-defined and distinct from one another.

Intuition Behind the Silhouette Score

For each data point, the Silhouette Score combines two distances:

1. Intra-cluster Distance (‘a’): The average distance between a data point and all other points in the same cluster. This measures how well a point fits within its cluster. Lower values of ‘a’ indicate high cohesion.
2. Inter-cluster Distance (‘b’): The average distance between a data point and all points in the nearest neighboring cluster (i.e., the closest cluster to which the point does not belong). This measures how distinct a cluster is from other clusters, with higher values indicating better separation.

The Silhouette Score for a single data point i is calculated as:
$s(i) = (b(i) - a(i)) / \max(a(i), b(i))$

Where:
- s(i) is the Silhouette Score for point i.
- a(i) is the average distance from point i to all other points in its cluster.
- b(i) is the average distance from point i to all points in the nearest neighboring cluster.

Interpretation of the Silhouette Score

The formula for the Silhouette Score captures the balance between a and b:
- If ‘a’ is much smaller than ‘b’ (i.e., the point is close to its own cluster and far from others), s(i) will be closer to 1, indicating a well-clustered point.
- If ‘a’ is similar to ‘b’, s(i) will be close to 0, suggesting that the point lies ambiguously between clusters.
- If ‘a’ is larger than ‘b’ (i.e., the point is closer to another cluster than its own), s(i) will be negative, indicating that the point may have been misclassified.

Overall Silhouette Score for a Clustering Solution

To compute the overall Silhouette Score for a clustering solution, you take the average Silhouette Score across all data points:

Silhouette Score = (1/N) * sum(s(i)) for i = 1 to N

where N is the total number of data points. This gives a single value between -1 and 1:
- Closer to 1: Well-defined clusters with high cohesion and separation.
- Around 0: Ambiguous clusters with overlapping points.
- Closer to -1: Poor clustering where many points might be assigned to the wrong cluster.

GridSearchCV

GridSearchCV allows us to efficiently explore multiple settings to identify the parameter combination that yields the best performance based on cross-validation. Here's an overview of how it worked and why it was beneficial for our SVR model.

Purpose of GridSearchCV

The SVR model's performance depends heavily on the choice of its hyperparameters:

1. **C** (Regularization Parameter): This parameter controls the trade-off between achieving a low error on the training data and maintaining a simple model. A higher value of **C** makes the model fit the training data more closely, potentially capturing more intricate patterns. However, too high a **C** value can lead to overfitting.
2. **epsilon** (Epsilon-Insensitive Loss): This parameter defines a margin within which errors are **not penalized** in SVR. A smaller **epsilon** allows the model to fit closer to the data points, while a larger **epsilon** creates a wider margin, thus making the model more tolerant to small deviations and less likely to overfit.

By using GridSearchCV, we were able to systematically search through a predefined set of values for **C** and **epsilon** and determine the optimal combination of these parameters based on the model's performance.

How GridSearchCV Works

1. **Parameter Grid**: We specified a range of values for both **C** and **epsilon** to explore. The parameter grid included values of C like [0.1, 1, 10, 100] and epsilon values such as [0.01, 0.1, 0.5, 1].
2. **Cross-Validation**: GridSearchCV performed **5-fold cross-validation** on each combination of **C** and **epsilon**. This means the training data was split into 5 subsets (folds), and the model was trained and validated 5 times for each parameter combination. Each time, the model was trained on 4 folds and validated on the remaining fold, cycling through all folds.

3. **Evaluation Metric**: I used **negative**[2] **mean absolute error (MAE)** as the scoring metric in GridSearchCV. Since the main goal of SVR is to minimize prediction error, this metric allowed me to directly and consistently evaluate the effectiveness of each parameter combination by its impact on the same metric.
4. **Selection of Best Parameters**: After completing the cross-validation process for all combinations, GridSearchCV selected the **C** and **epsilon** values that achieved the lowest average MAE across the folds. This combination of parameters was used to train the final model on the entire training set.

Advantages of GridSearchCV in Our Model

- **Systematic Exploration**: GridSearchCV ensured that all possible parameter combinations within the specified ranges were tested, providing confidence that I found the best possible settings.
- **Cross-Validation**: By using 5-fold cross-validation, I minimized the risk of overfitting, as each combination was evaluated on different subsets of data, providing a robust measure of performance.
- **Objective Optimization**: The selection of parameters was data-driven, based on the model's performance, rather than subjective guessing. This objective approach undoubtedly increases the reliability of the chosen parameters.

[2] The **negative Mean Absolute Error (MAE)** is used because GridSearchCV, by default, aims to **maximize** the scoring metric. Needless to say, **MAE** is a measure of error and thus lower values indicate better model performance. However, if MAE were used directly, GridSearchCV would have attempted to maximize it, which would mean selecting higher error values (the opposite of my goal). This approach is just a workaround to minimize the actual MAE through maximizing the negative MAE.