[Bike Sharing Demand]

[Yingshuang Yang]

[DSCI570]

[Nov 1 2024]

[Zibo Wang]

Introduction

Currently, bike layouts often mismatch demand patterns, resulting in unavailability during peak hours and underutilization at other times.

My model predicts the number of bikes people will rent based on different factors like time of day(e.g., peak commuting hours versus off-peak times), season(e.g., higher demand in spring and summer), and temperature(e.g., fewer rentals on extreme temperature days). This solution addresses the critical challenge faced by bike-sharing companies and government: ensuring bikes are available during peak hours while avoiding waste during low-demand periods.

For bike-sharing companies, our predictive data enable precise resource allocation, they can optimize bike placement and scheduling strategies, significantly reducing operational costs and improving user experience. For government departments, our data supports informed decision-making in urban planning, such as optimizing bike lane placement and parking infrastructure, ultimately reducing traffic congestion.

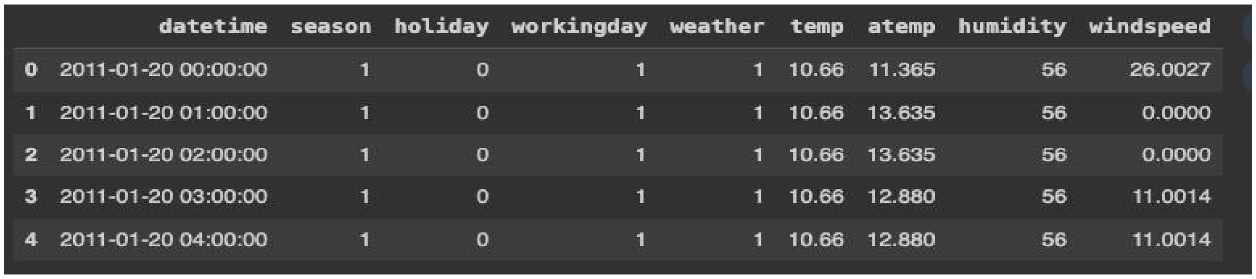
Our solution improves on current methods by giving more accurate and timely information, helping companies and cities manage bike-sharing systems more effectively.

Technical Details

Dataset

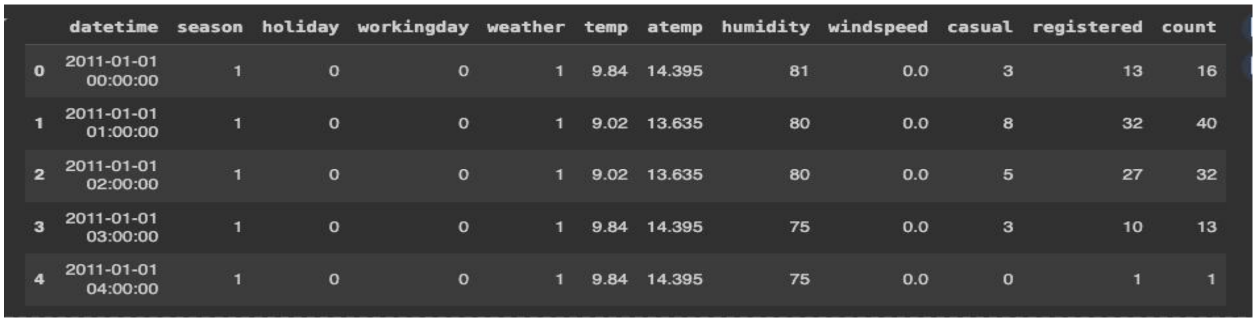
The datasets are provided in CSV format, containing hourly rental data spanning two years. The training set comprises data from the first 19 days of each month, while the test set includes data from the 20th day to the end of each month.

The training data set consists of 10,886 data points with 9 feature columns and 3 target columns. The testing data set contains 6,493 data points with only 9 feature columns. These datasets include features such as the datetime of renting, season, holiday, working, weather, temperature, 'feels like' temperature, humidity and windspeed. And 3 columns of targets are number of non-registered user rentals initiated, number of registered user rentals initiated, and number of total rentals, shown in Figure1 and2. The main prediction is total count, so I just remove 'casual' and 'registered'.



	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014

Fig. 1. The first 5 rows of training data

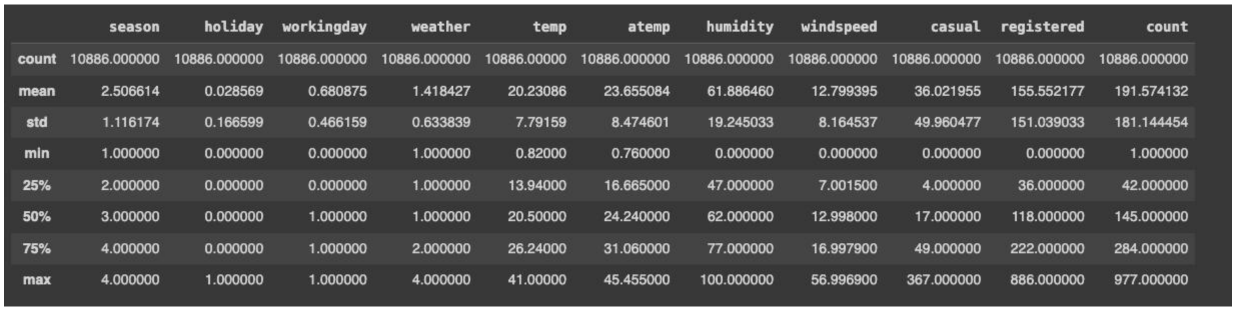


	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

Fig. 2. The first 5 rows of testing data

Data Wrangling & Cleaning (if applicable)

As you can see from the description, shown in Figure3, the first 4 columns have been Label Encoded, and the box plot, shown in Figure4, shows that the humidity and wind speed have some outliers that need to be removed using the IOR method.



	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
std	1.116174	0.166599	0.466159	0.633839	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454
min	1.000000	0.000000	0.000000	1.000000	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	0.000000	1.000000	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	3.000000	0.000000	1.000000	1.000000	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	4.000000	0.000000	1.000000	2.000000	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	4.000000	1.000000	1.000000	4.000000	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000

Fig. 3. The describe of training data

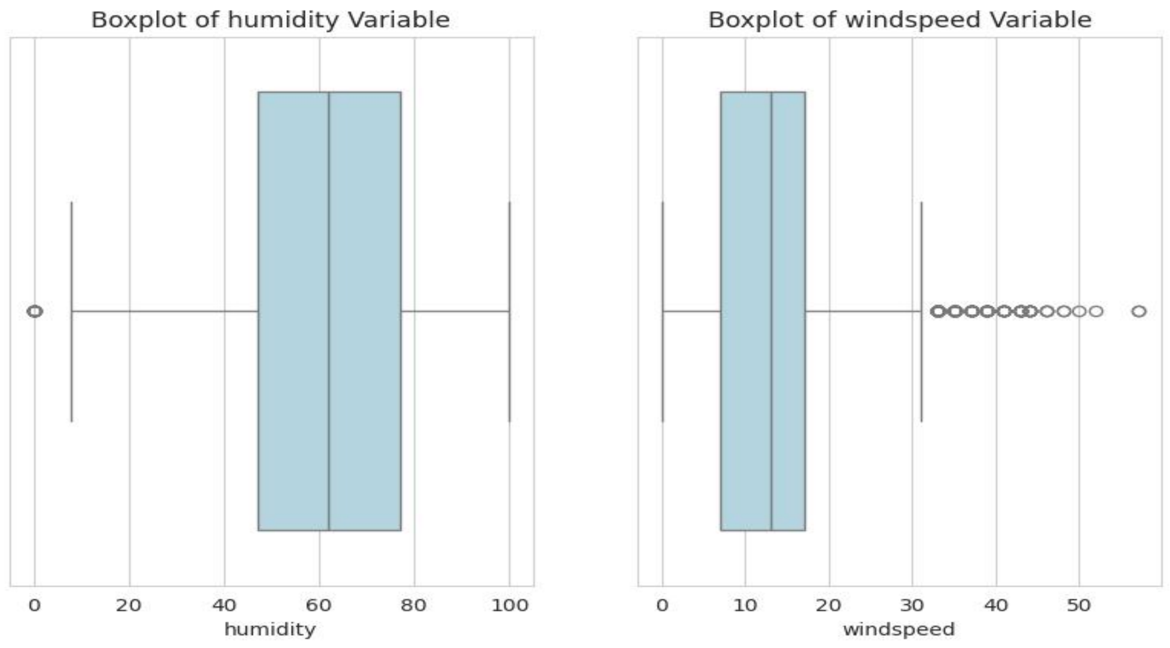


Fig. 4. The boxplot of humidity and windspeed

I created additional features from the datetime field, including the hour of rental, week, month, and day of the week. Additionally, I categorized temperature into three levels: ‘1’ for temperatures from the lowest degree to 10 degrees, ‘2’ for temperatures from 10 to 30 degrees, and ‘3’ for temperatures from 30 degrees to the highest degree.

At the end of the data preprocessing, I standardized the data. Although standardization was not necessary for the random forest model, the final evaluation showed a 0.001 reduction in RMSLE after standardization.

Model(s)

I used a random forest regression model to predict the target variable. Random Forest is an integrated algorithm based on multiple decision trees, which improves the overall prediction performance. The final parameters of the model are as follows：

n_estimators=200: 200 trees were used to construct the random forest.

max_depth=10: the maximum depth of each tree is 10.

min_samples_split=8: the minimum number of samples needed to split a node is 8.

min_samples_leaf=3: the minimum number of samples per leaf node is 3.

random_state=0: set the random seed to ensure reproducible results.

Here is a visualization of the first decision tree in a random forest, shown in Figure5.

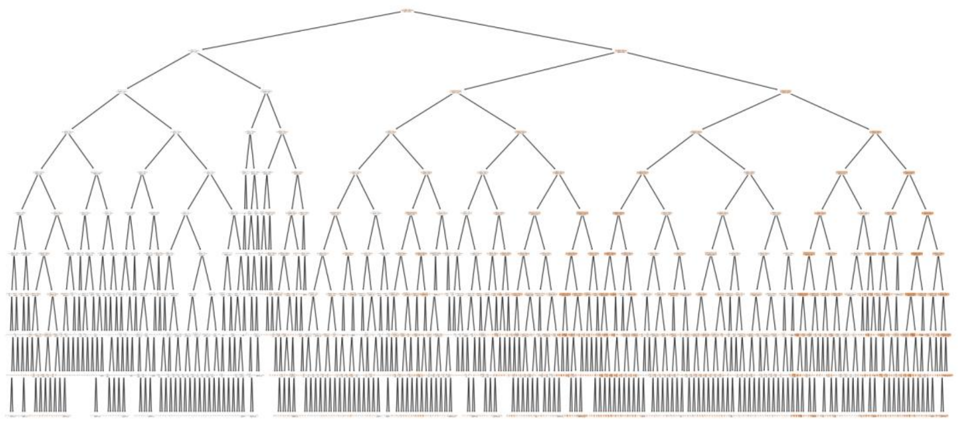


Fig. 5. The first decision tree in a random forest.

I did not use feature selection alone because even after data engineering,the dataset contains only 14 features, and the random forest model automatically selects the most important features during training, shown in Figure6.

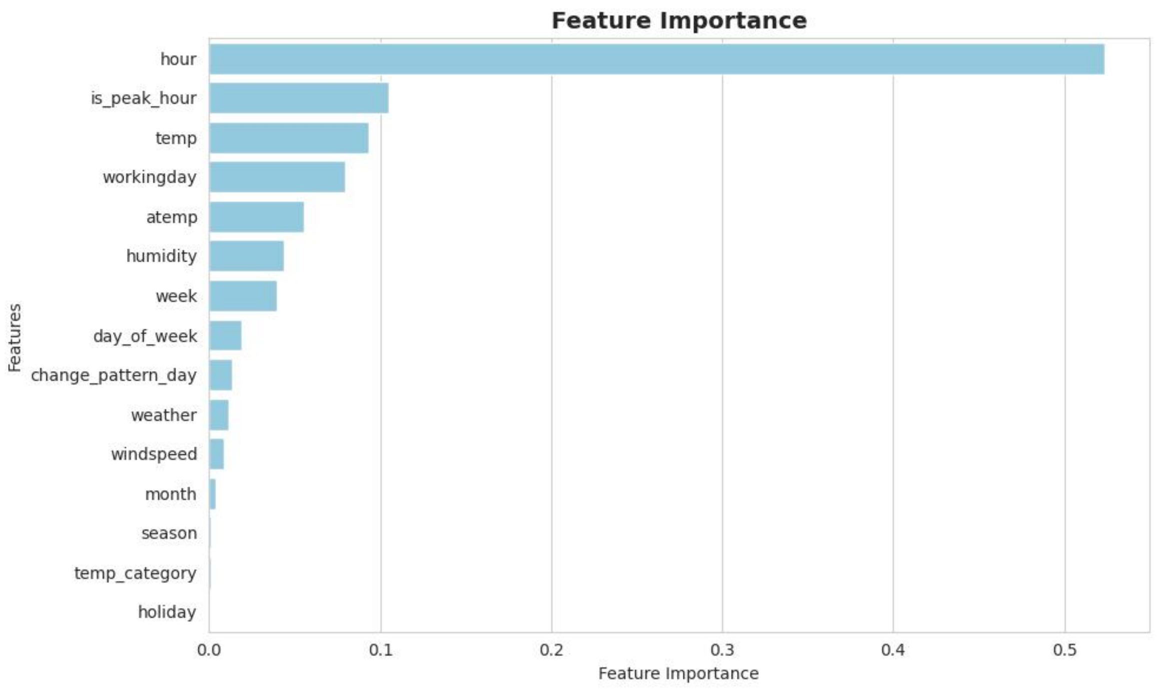


Fig. 6. The feature importance

To avoid overfitting, I segmented and cross-validated the dataset. Due to the limited amount of data, I decided to allocate more data to training process. The dataset was divided into 80% training set and 20% validation set to monitor the model performance during training. 5-fold cross-validation (k=5) is used during training, which is the training data divided into 5 subsets, the model is trained on 4 subsets and validated on the remaining subset.

Performance

My evaluation metric is RMSLE, as required by Kaggle. This metric applies a log transformation to the difference between the actual and predicted values, making it suitable for data where the target variable has a skewed distribution, as shown in Figure 7.

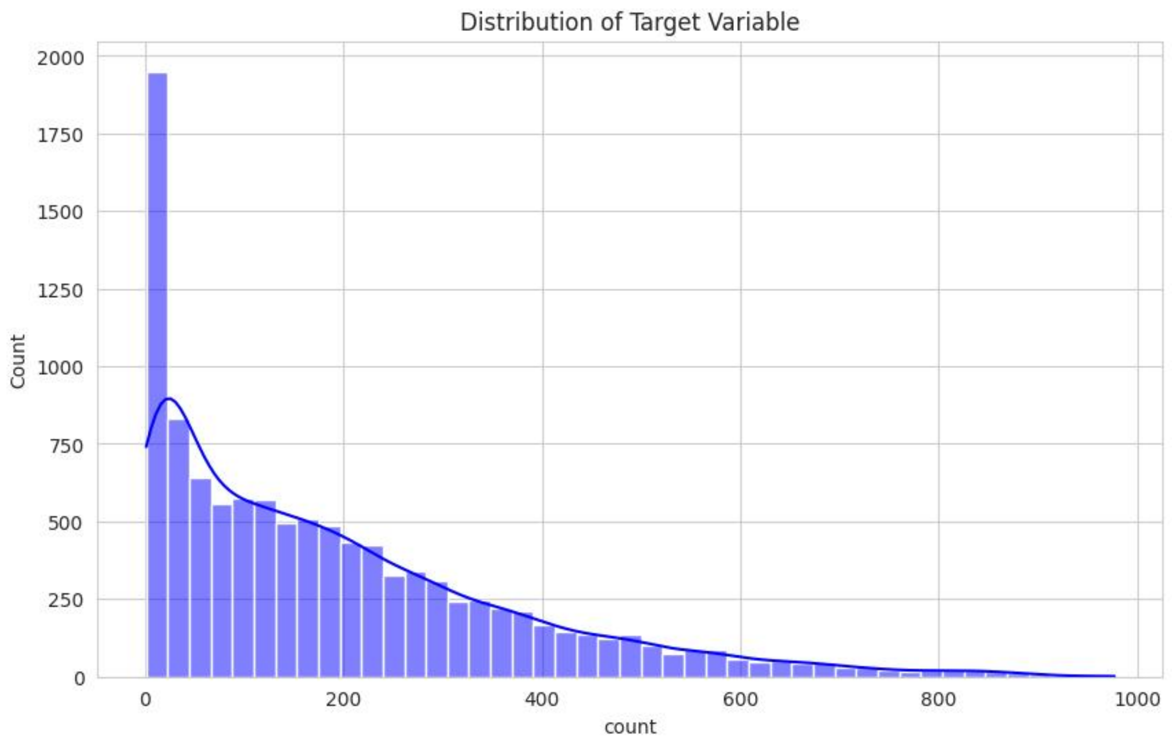


Fig. 7. The distribution of target variable

A visualization of the results is provided in Figure 8.

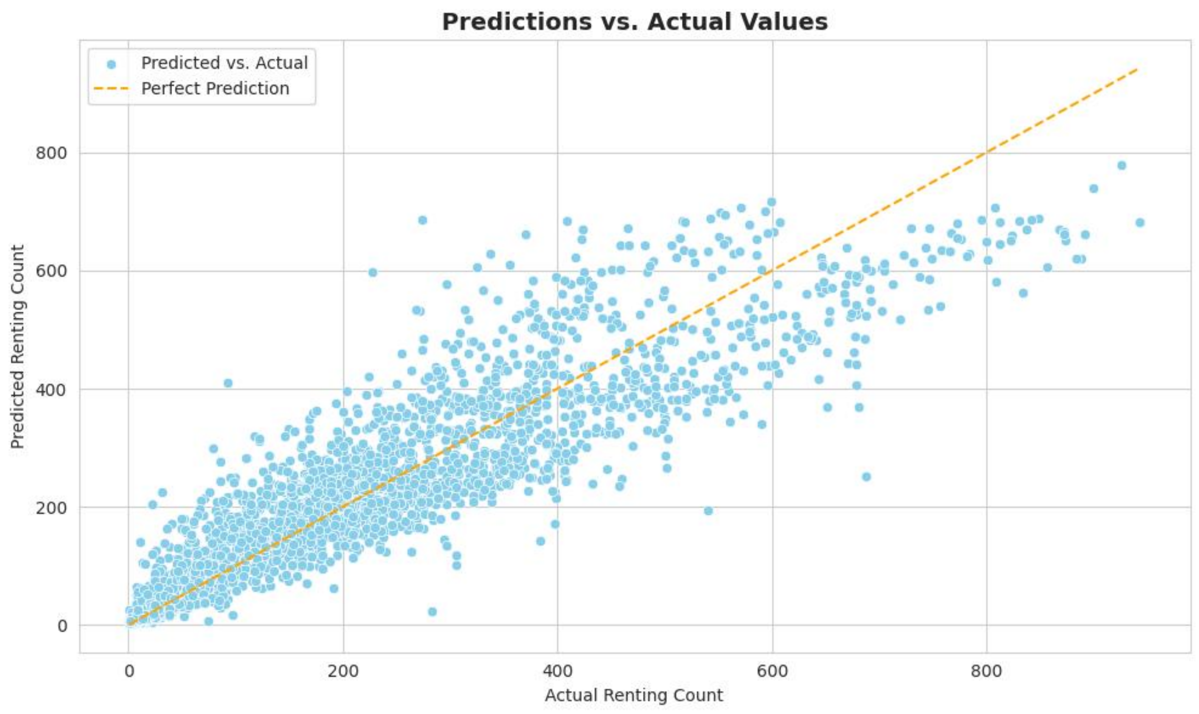


Fig. 8. The line plot of predicted versus actual values

For the initial model, I used the default parameters for Random Forest, SVR, and Ridge Regression. The performance of each model is shown in Figure 9. It is evident that Random Forest achieved the best performance, but the model was overfitted, with a training RMSLE of 0.1790 and a testing RMSLE of 0.4085.

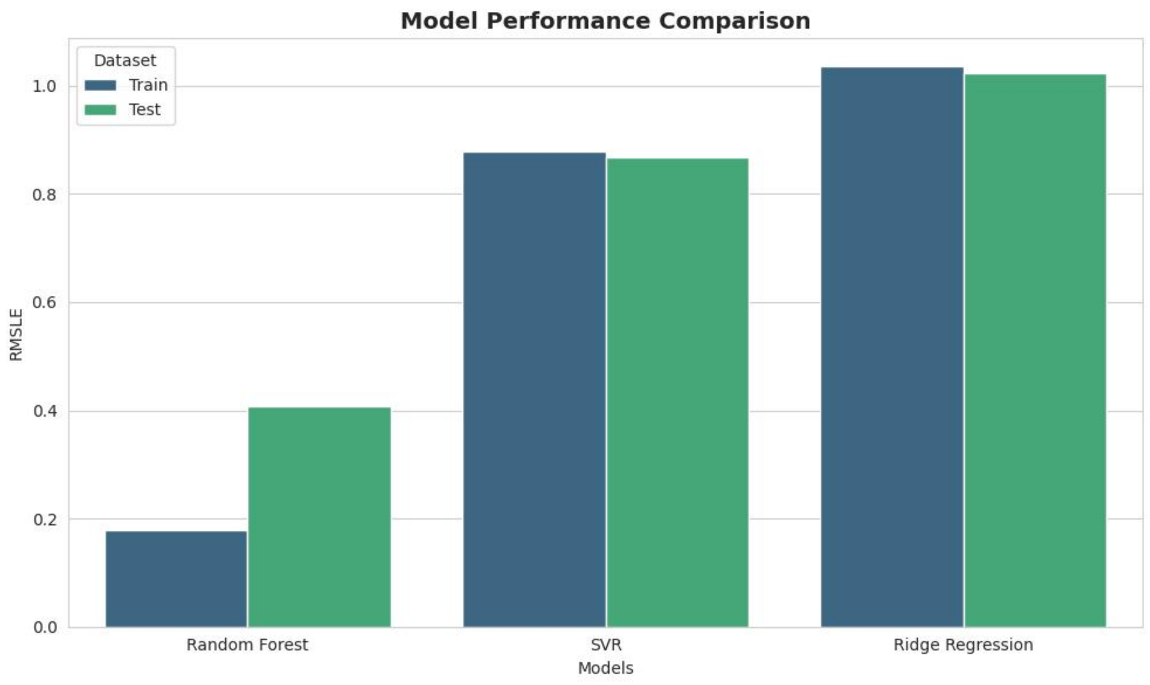


Fig. 9. The performance of each model

In the second attempt, I used GridSearchCV to find the optimal parameters, resulting in min_samples_leaf=2, min_samples_split=2, and n_estimators=300. I also applied 5-fold cross-validation to help mitigate overfitting. The results were a training RMSLE of 0.2242 and a testing RMSLE of 0.4084. Although this improved the model, overfitting persisted.

In the third attempt, I manually adjusted the parameters to avoid overfitting. The parameters are min_samples_leaf=3, min_samples_split=8, max_depth=10, and n_estimators=500. This yielded a training RMSLE of 0.3454 and a testing RMSLE of 0.4252. While the test RMSLE increased slightly, the degree of overfitting was reduced.

In the fourth attempt, I performed additional feature engineering. I created the features is_peak_hour and renting change pattern of day based on hourly rental patterns, as shown in Figure 10. In addition, I added some rolling statistics features and difference features.

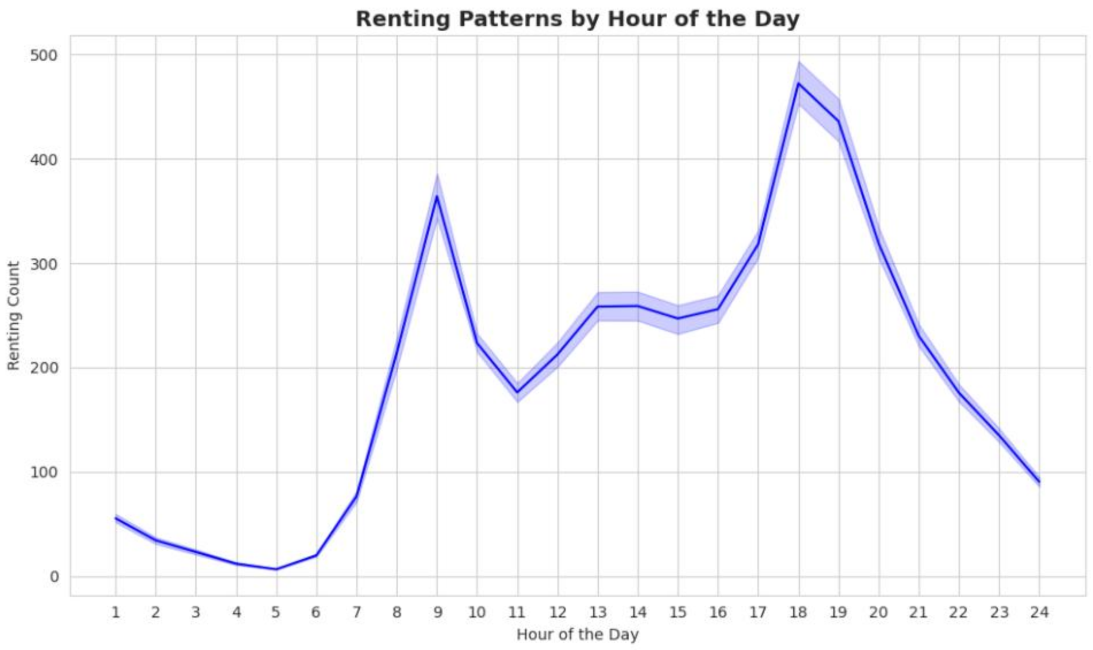


Fig. 10. Renting Patterns by Hour of the Day

With these new features, the model achieved a training RMSLE of 0.3231 and a testing RMSLE of 0.4144. Although there was some improvement, overfitting remained an issue.

In the final attempt, I thought that the random selection of training and testing data might have caused the model to 'see' some test data in advance, leading to overfitting. Therefore, I applied a time-series split to prevent information leakage. The first 80% of the data was used as the training set, and the remaining 20% as the test set, respecting the chronological order of the dataset. However, this resulted in a training RMSLE of 0.813 and a testing RMSLE of 0.4915, indicating worse performance and increased overfitting.

There’s another some a little adjust that I do not record. As a result I choose the fourth attempt as my model.

Attempt	Train RMSLE	Test RMSLE	Overfitting (Difference)
Default Parameters	0.179	0.4085	0.2295
GridSearchCV Optimization	0.2242	0.4084	0.1842
Manual Parameter Adjustment	0.3454	0.4252	0.0798
Additional Feature Engineering	0.3231	0.4144	0.0913
Time Series Split	0.2813	0.4913	0.2100

Challenges and Conclusion

One of the main challenges I encountered during the modeling process was the difficulty in identifying which step caused poor model performance. I could only modify one step at a time, yet the final results were not always as expected. For instance, I initially focused on adjusting various parameters, only to realize later that redoing the feature engineering was necessary. However, adding more features doesn't always lead to better results; for example, I tried increasing the feature set to 32, but the outcome was not as good as with the current model. At one point, I suspected that data leakage was the reason for overfitting, but that turned out not to be the case.

I came close to achieving the desired performance, but I believe the feature engineering was still insufficient. In the end, I realized that restructuring the data was crucial for optimizing the model's performance. Given more time, I would explore additional feature engineering and selection methods and experiment with neural network-based models.

Although this particular approach was not very effective for my model, I learned about the TimeSeriesSplit method for handling chronological data and gained a better understanding of how to explore and prepare the data from the start.

Acknowledgment and References

I would like to express my gratitude to the following individuals and resources that significantly contributed to this project:

- **Professor Zibo**: For providing valuable insights into feature engineering through proposal feedback, which guided the initial steps of this project.
- **Fanaee-T, Hadi, and Gama, Joao**: For providing and hosting the dataset, which served as the foundation for the analysis.

References

1. Fanaee-T, Hadi, and Gama, Joao. *Event Labeling Combining Ensemble Detectors and Background Knowledge*, Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg.
2. Will Cukierski. *Bike Sharing Demand.* https://kaggle.com/competitions/bike-sharing-demand, 2014. Kaggle.