assignment_2_sofia_gonzalez

April 10, 2025

1 Task 1: EDA

```
First 5 rows of the dataset:
   instant      dteday  season  yr  mnth  hr  holiday  weekday  workingday  \
0        1  2011-01-01       1   0     1   0        0        6           0
1        2  2011-01-01       1   0     1   1        0        6           0
2        3  2011-01-01       1   0     1   2        0        6           0
3        4  2011-01-01       1   0     1   3        0        6           0
4        5  2011-01-01       1   0     1   4        0        6           0

   weathersit  temp   atemp   hum  windspeed  casual  registered  cnt
0           1  0.24  0.2879  0.81        0.0       3          13   16
1           1  0.22  0.2727  0.80        0.0       8          32   40
2           1  0.22  0.2727  0.80        0.0       5          27   32
3           1  0.24  0.2879  0.75        0.0       3          10   13
4           1  0.24  0.2879  0.75        0.0       0           1    1

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   instant     17379 non-null  int64
 1   dteday      17379 non-null  object
 2   season      17379 non-null  int64
 3   yr          17379 non-null  int64
 4   mnth        17379 non-null  int64
 5   hr          17379 non-null  int64
 6   holiday     17379 non-null  int64
 7   weekday     17379 non-null  int64
 8   workingday  17379 non-null  int64
 9   weathersit  17379 non-null  int64
 10  temp        17379 non-null  float64
 11  atemp       17379 non-null  float64
 12  hum         17379 non-null  float64
 13  windspeed   17379 non-null  float64
 14  casual      17379 non-null  int64
```

```
 15  registered  17379 non-null  int64
 16  cnt         17379 non-null  int64
dtypes: float64(4), int64(12), object(1)
memory usage: 2.3+ MB
None

Missing values per column:
instant       0
dteday        0
season        0
yr            0
mnth          0
hr            0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
casual        0
registered    0
cnt           0
dtype: int64

Summary Statistics:
          instant        season            yr          mnth            hr  \
count  17379.0000  17379.000000  17379.000000  17379.000000  17379.000000
mean    8690.0000      2.501640      0.502561      6.537775     11.546752
std     5017.0295      1.106918      0.500008      3.438776      6.914405
min        1.0000      1.000000      0.000000      1.000000      0.000000
25%     4345.5000      2.000000      0.000000      4.000000      6.000000
50%     8690.0000      3.000000      1.000000      7.000000     12.000000
75%    13034.5000      3.000000      1.000000     10.000000     18.000000
max    17379.0000      4.000000      1.000000     12.000000     23.000000

            holiday       weekday    workingday    weathersit          temp  \
count  17379.000000  17379.000000  17379.000000  17379.000000  17379.000000
mean       0.028770      3.003683      0.682721      1.425283      0.496987
std        0.167165      2.005771      0.465431      0.639357      0.192556
min        0.000000      0.000000      0.000000      1.000000      0.020000
25%        0.000000      1.000000      0.000000      1.000000      0.340000
50%        0.000000      3.000000      1.000000      1.000000      0.500000
75%        0.000000      5.000000      1.000000      2.000000      0.660000
max        1.000000      6.000000      1.000000      4.000000      1.000000

              atemp           hum     windspeed        casual    registered  \
```

count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000
mean	0.475775	0.627229	0.190098	35.676218	153.786869
std	0.171850	0.192930	0.122340	49.305030	151.357286
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.333300	0.480000	0.104500	4.000000	34.000000
50%	0.484800	0.630000	0.194000	17.000000	115.000000
75%	0.621200	0.780000	0.253700	48.000000	220.000000
max	1.000000	1.000000	0.850700	367.000000	886.000000

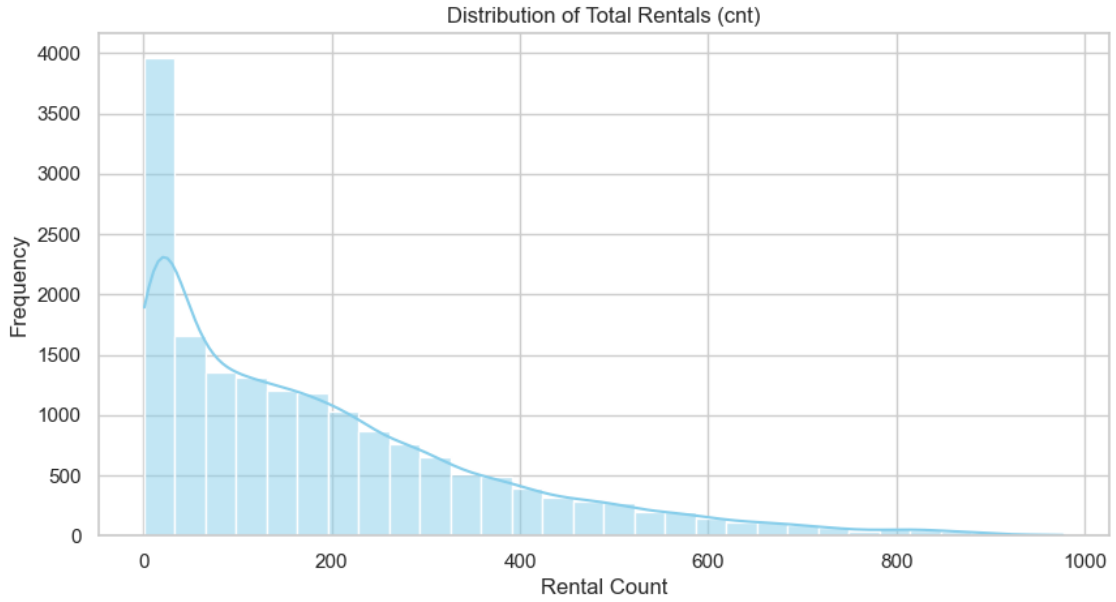
	cnt
count	17379.000000
mean	189.463088
std	181.387599
min	1.000000
25%	40.000000
50%	142.000000
75%	281.000000
max	977.000000

We removed the following columns from the dataset:

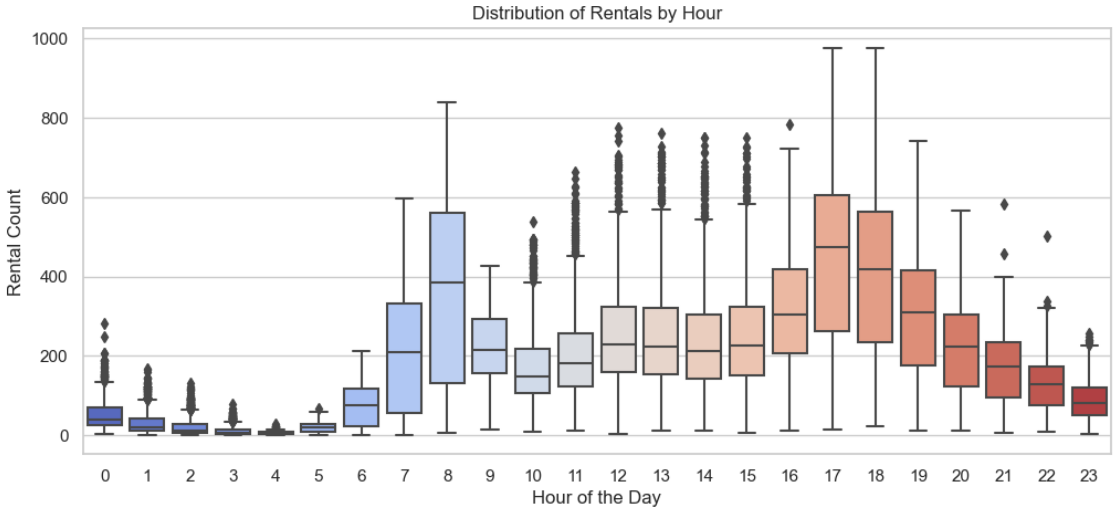
- Instant: it is just a unique id for each row, doesnt add any predictive value
- dteday: A string version of the date. Since we already have features like year (yr), month (mnth), and day of the week (weekday), this becomes redundant.
- casual and registered: These two columns together directly sum up to the target variable (cnt), which represents the total bike rentals. Keeping them would be a form of data leakage because the model would learn the target from its components instead of actual predictors.

1.0.1 EDA Visualizations

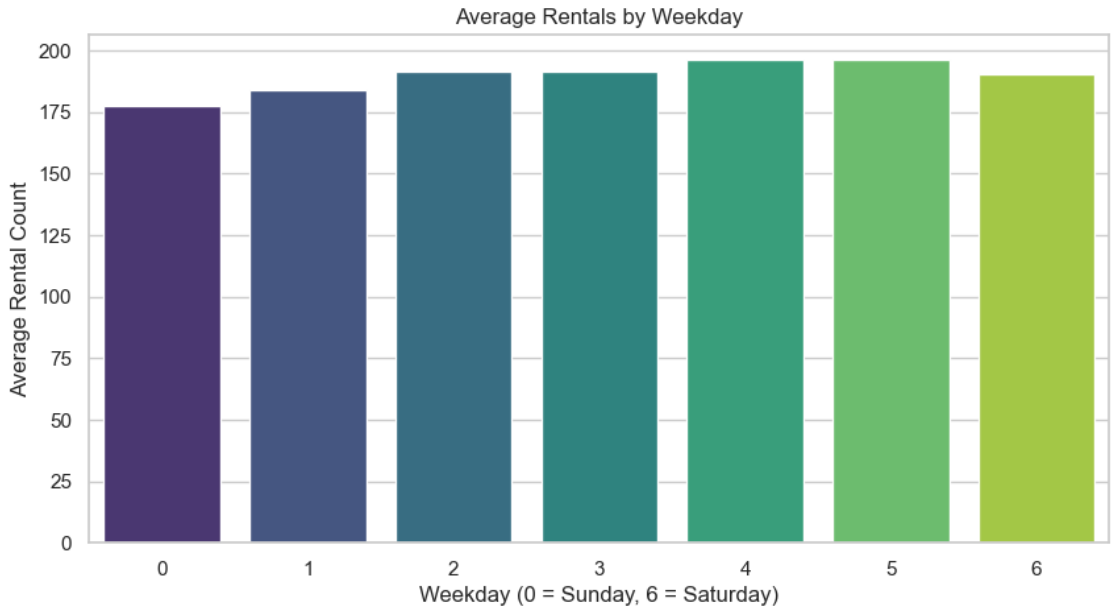
```
<Figure size 1200x600 with 0 Axes>
```



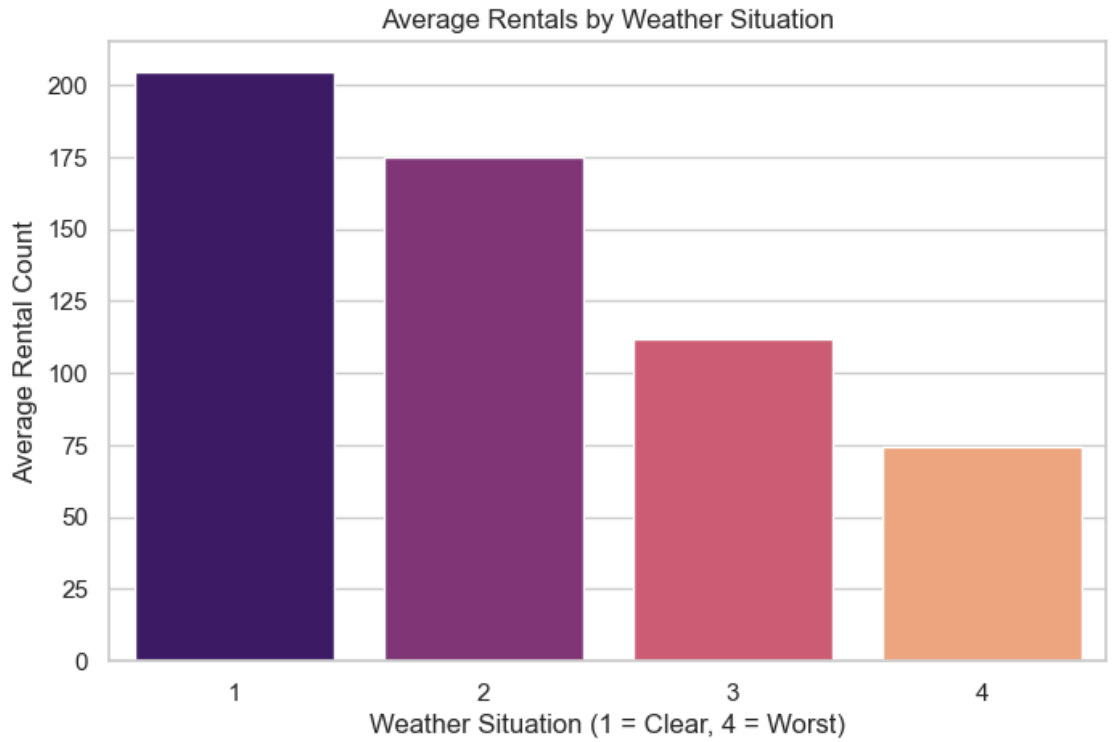
Helps us understand how bike rentals are distributed. The histogram shows that most bike rental sessions fall on the lower end of the scale, with a noticeable right skew. This makes sense because high rental spikes are rarer, while lower or moderate rental volumes are more common. The peak is strong in the 0-100 rental range and then it gradually decreases. High rental counts are much more rare, which is somethign that we were expecting.



Shows usage trends by hour. We can see that there are 2 noticeable peaks which are morning hours around 8AM and afternoon around 5-6PM. This makes sense as they are likely the typical commuting hours. We can also see that early in the morning there are far few less people who rent bikes (2-5AM)

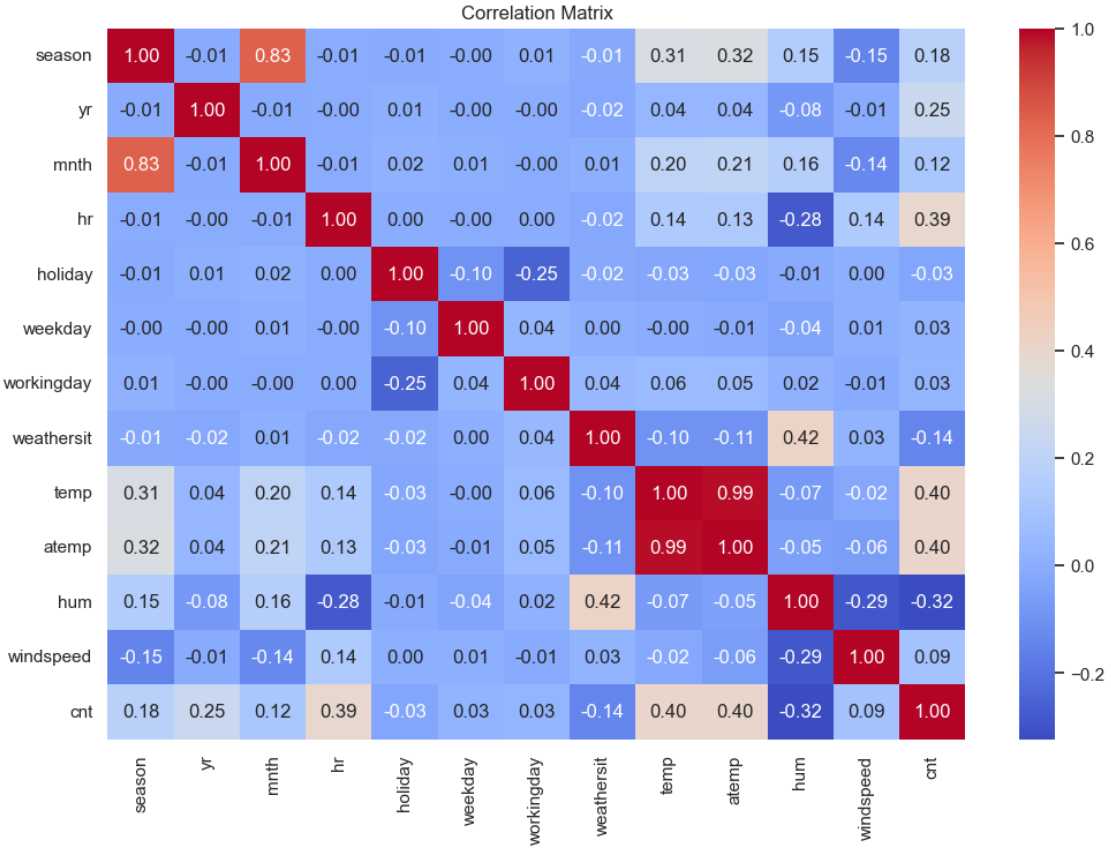


Helps spot trends across weekdays. We can see that the rentals stay pretty consistent during weekends as well as weekdays. There are more rentals on Thursdays-Fridays, but in general it stays pretty consistent. Although Sunday is the day with the least rentals, its not highly significant. We can assume then that they use bikes for commuting as well as for leisure.



Explores how external weather conditions affect bike usage. As we would expect in our data there is a negative correlation between poor weather and bike rental. The highest rental count occurs when the weathcer is clear and it drops as the weather also drops.

Correlation plot



We probably should drop atemp as it is highly correlated with temp

2 Task 2: Data Splitting

```
Training set shape: (10427, 12)
Validation set shape: (3476, 12)
Test set shape: (3476, 12)
```

I split the data into 3 sets: 60% for training, 20% for validation, and 20% for testing. I did this by using train_test_split, ensuring that the model has enough data to learn from patters, enough separate data to tune parameters, and untouched data in order to evaluate performance.

We are doing this before of feature engineering in order to avoid data leakage. We don't want any over optimistic performance and poor generalization.

3 Task 3: Feature Engineering

We have to encode cyclical features because there is a circular nature. hour 0 and hour 23 are right next to each other, not that far away. The same thing happens with weekdays, monday and sunday are right next to each other. We used sine and cosine in order to capture patterns that repeat cyclically so now it will be between -1 and 1.

```
Previewing original 'hr' and 'weekday' before encoding:
       hr  weekday
14779   0        4
16914  13        3
15863   4        0
4204    2        3
1554    5        4

After cyclical encoding:
         hr_sin    hr_cos  weekday_sin  weekday_cos
14779  0.000000  1.000000    -0.433884    -0.900969
16914 -0.258819 -0.965926     0.433884    -0.900969
15863  0.866025  0.500000     0.000000     1.000000
4204   0.500000  0.866025     0.433884    -0.900969
1554   0.965926  0.258819    -0.433884    -0.900969
```

I also applied one-hot econding to season, weathersit, and month. This will convert them into binary variables, this is done because these variables are cateogrical. We dont want the labels to be seen as magnitudes.

```
Before One-Hot Encoding:
Season unique values: [3 4 1 2]
Weathersit unique values: [1 2 3 4]
Month unique values: [ 9 12 10  6  3  2  5 11  1  7  8  4]

After One-Hot Encoding (sample of columns):
       season_2  season_3  season_4  weathersit_2  weathersit_3  weathersit_4  \
14779     False      True     False         False         False         False
16914     False     False      True          True         False         False
15863     False     False      True          True         False         False
4204      False      True     False         False         False         False
1554      False     False     False         False          True         False

       mnth_2  mnth_3  mnth_4  mnth_5  mnth_6  mnth_7  mnth_8  mnth_9  \
14779   False   False   False   False   False   False   False    True
16914   False   False   False   False   False   False   False   False
15863   False   False   False   False   False   False   False   False
4204    False   False   False   False    True   False   False   False
1554    False    True   False   False   False   False   False   False

       mnth_10  mnth_11  mnth_12
14779    False    False    False
```

```
16914    False    False     True
15863     True    False    False
4204     False    False    False
1554     False    False    False
```

I did scaling because it ensures that features with different ranges are brought to a similar scale. So it makes everything fair as it is working on the same scale instead of giving more importance to variables which ranges are with bigger numbers.

Before Scaling:

	temp	atemp	hum	windspeed
count	10427.000000	10427.000000	10427.000000	10427.000000
mean	0.496436	0.475479	0.626612	0.190609
std	0.193049	0.172721	0.192536	0.121860
min	0.020000	0.000000	0.000000	0.000000
25%	0.340000	0.333300	0.480000	0.104500
50%	0.500000	0.484800	0.620000	0.194000
75%	0.660000	0.621200	0.780000	0.253700
max	1.000000	1.000000	1.000000	0.850700

After Scaling:

	temp	atemp	hum	windspeed
count	1.042700e+04	1.042700e+04	1.042700e+04	1.042700e+04
mean	-1.454885e-16	-2.524754e-16	-4.054598e-16	1.011946e-16
std	1.000048e+00	1.000048e+00	1.000048e+00	1.000048e+00
min	-2.468067e+00	-2.753011e+00	-3.254674e+00	-1.564243e+00
25%	-8.103813e-01	-8.232137e-01	-7.615134e-01	-7.066588e-01
50%	1.846157e-02	5.396689e-02	-3.434163e-02	2.782758e-02
75%	8.473044e-01	8.437189e-01	7.967118e-01	5.177587e-01
max	2.608596e+00	3.036960e+00	1.939410e+00	5.417070e+00

I added an interaction term between temp and humidity. This was because of the idea that hot + humid conditions might impact bike usage differently than hot + dry weather. We noticed in EDA that both features independently affected usage, so combining them may help uncover non-linear relationships.

We did hr and working day because rush hour peaks only happen on working days and weekend patterns are different even at the same hour

We also added season and weathersit because they don't operate independently and their combined context can strongly influence bike rental behavior. A rainy or foggy day (weathersit) in summer might still result in high rentals because it's warm and people are more likely to go out. But the same weathersit in winter could drastically reduce rentals due to already low temperatures and less daylight. This can be a great interaction to look out for

Sample interaction terms:

	temp_hum	hr_workingday	season_weathersit
14779	0.176825	1.000000	False
16914	0.869668	-0.965926	False
15863	0.000325	0.000000	False

```
4204    0.895106        0.866025                False
1554    2.300325        0.258819                False
```

After reviewing the correlation matrix, I found that atemp (feels-like temperature) is highly correlated with temp. Including both might introduce multicollinearity. So I dropped atemp to reduce redundancy and make the model more interpretable.

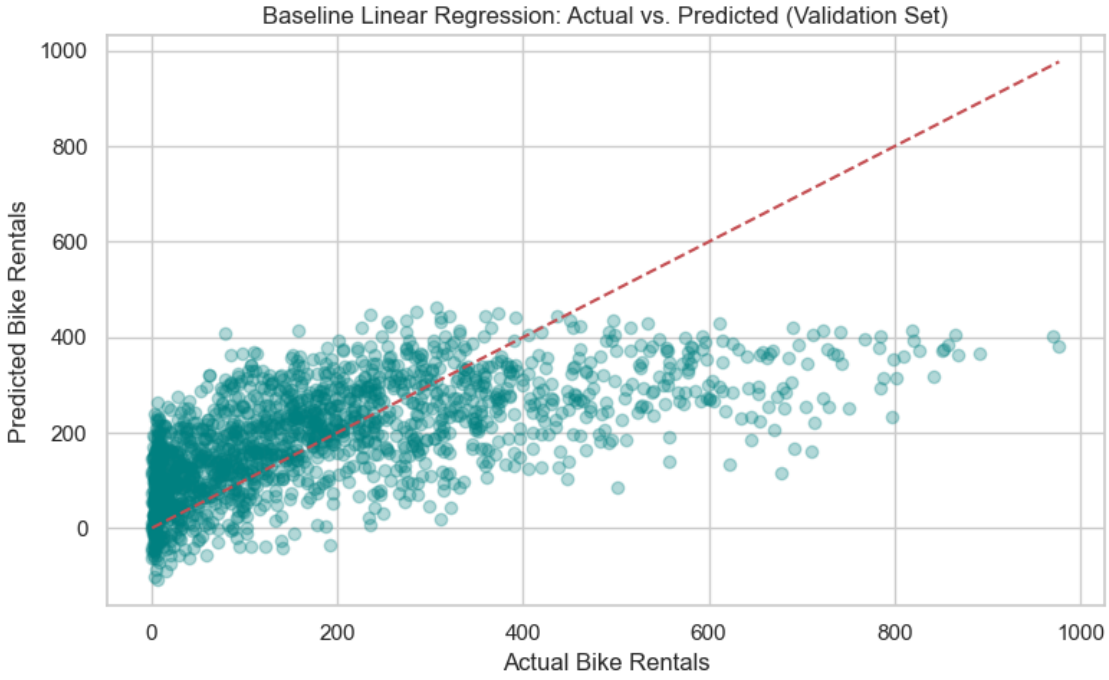
4 Task 4: Baseline Model - Linear Regression

Lets first reload the original dataset and split it, then we can start training a simple linear regression without the encoding and evaluate how it works.

```
Baseline Model Evaluation on Validation Set:
Mean Squared Error (MSE): 20150.19
Mean Absolute Error (MAE): 106.13
Root Mean Squared Error (RMSE): 141.95
R² Score: 0.3991
```

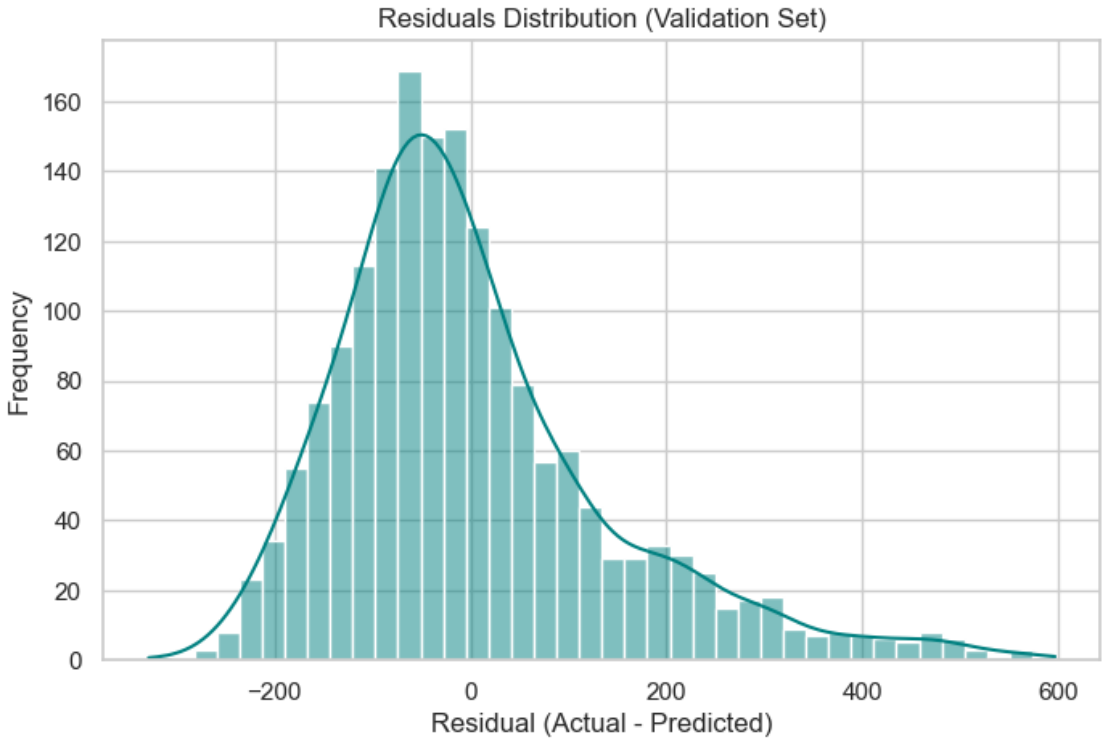
We can see that the model's prediction on the baseline are off by 142 rentals which is a lot. We can also see by the MAE that on average predictions are off by 106, regardless of the direction of the error. The MSE is 20,150 which means that the model is making significant prediciton mistakes. The model also explains roughly 40% of the variance in bike rentals meaning that there is a lot of the data's behavior that it doesnt capture in the model.

Plot - Actual vs Predicted



The model is underestimating high rental counts as we can see by the points falling under the dotted line.

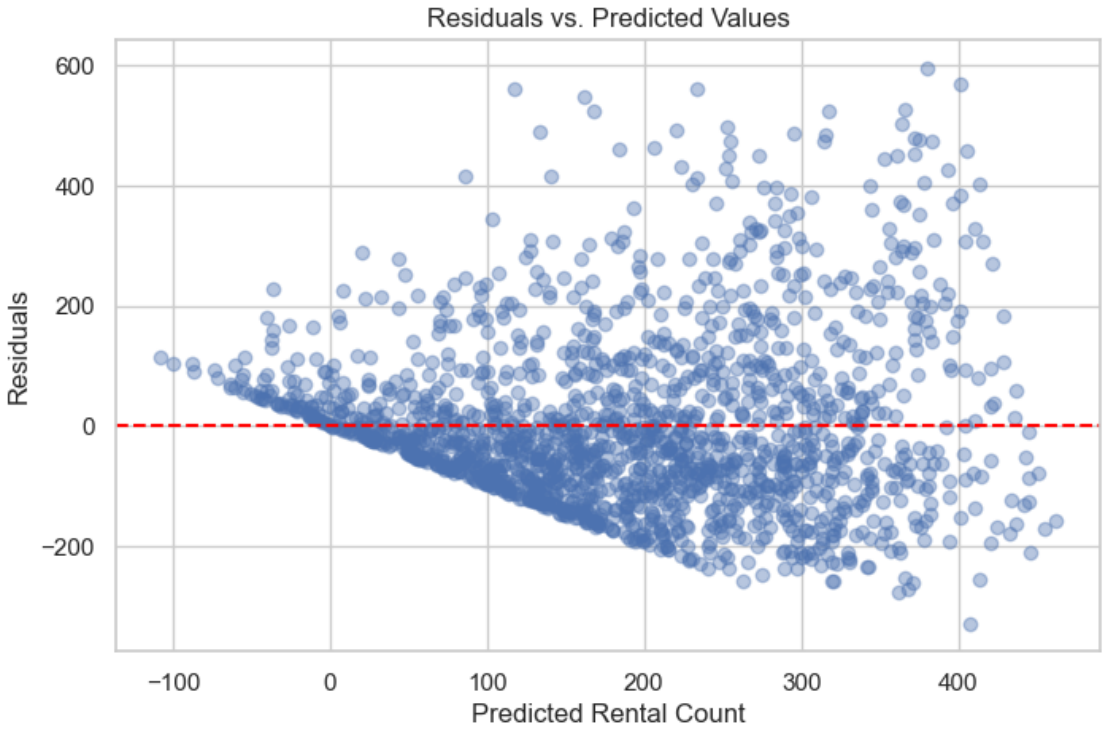
Plot - Residual Distribution



We can see that the residuals are centered around 0, which is a good sign because it menans that its not significantly overpredicting or underpredicting. It is clearly skewed to the right though, which means that it underestimates the bike rentals on the higher end.

We would ideally like it to be more tihghtly concentrated around 0. This just means that the model is struggling with complex scenarios, so might be a bit biased right now.

Plot - Residuals vs Predicted



We can see that the model performs worse whenever there's a higher number of rentals. We can see heteroscedasticity, which means that its errors are not consistent at all levels of prediction.

We can see that predictions for smaller rentals counts tend to be more accurate but as there are more rental counts there are more residuals meaning that there are underpredicting them.

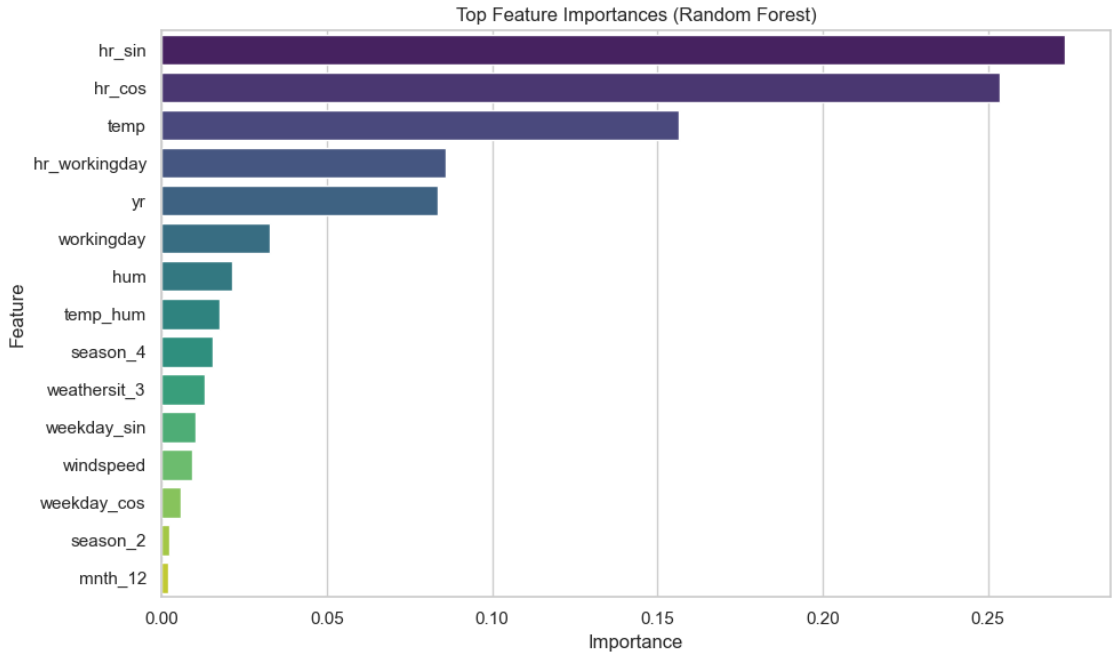
5 Task 5: Random Forest Regressor

Here we will be using the already processed data.

```
Random Forest Evaluation on Validation Set:
Mean Squared Error (MSE): 2139.29
Mean Absolute Error (MAE): 28.19
Root Mean Squared Error (RMSE): 46.25
R² Score: 0.9364
```

We can see that the random forest regressor performs better than the baseline model across all metrics. It reduces the MAE from 106 to 28, which is a great improvement. Also the RMSE went from 142 to 46, which is also huge improvement. Most importantly the R^2 went from 40% to 94% which means that it explains 94% of the variance in bike rentals. Really good.

We can see that the bike rental patterns are non-linear thats why the linear regression didnt do a great job, because it wasnt flexible enough.

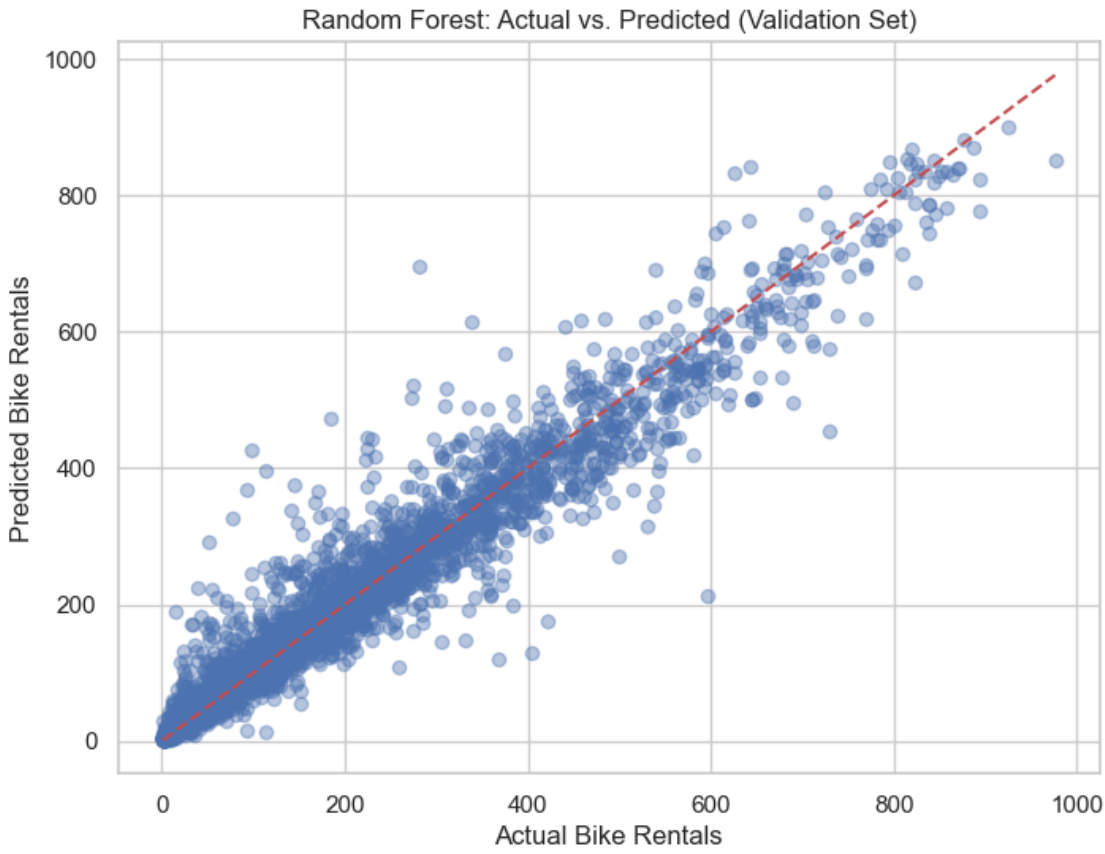


We can see that the most important features by far are the cyclically encoded hour values (hr_sin and hr_cos), which makes perfect sense because bike rental patterns follow daily cycles, with peaks likely during commute/rush hours.

Temperature is also a strong predictor, which aligns with the idea that people are more likely to rent bikes in warmer conditions. The interaction between hour and working day (hr_workingday) also ranks highly, which reflects how work schedules affect bike usage.

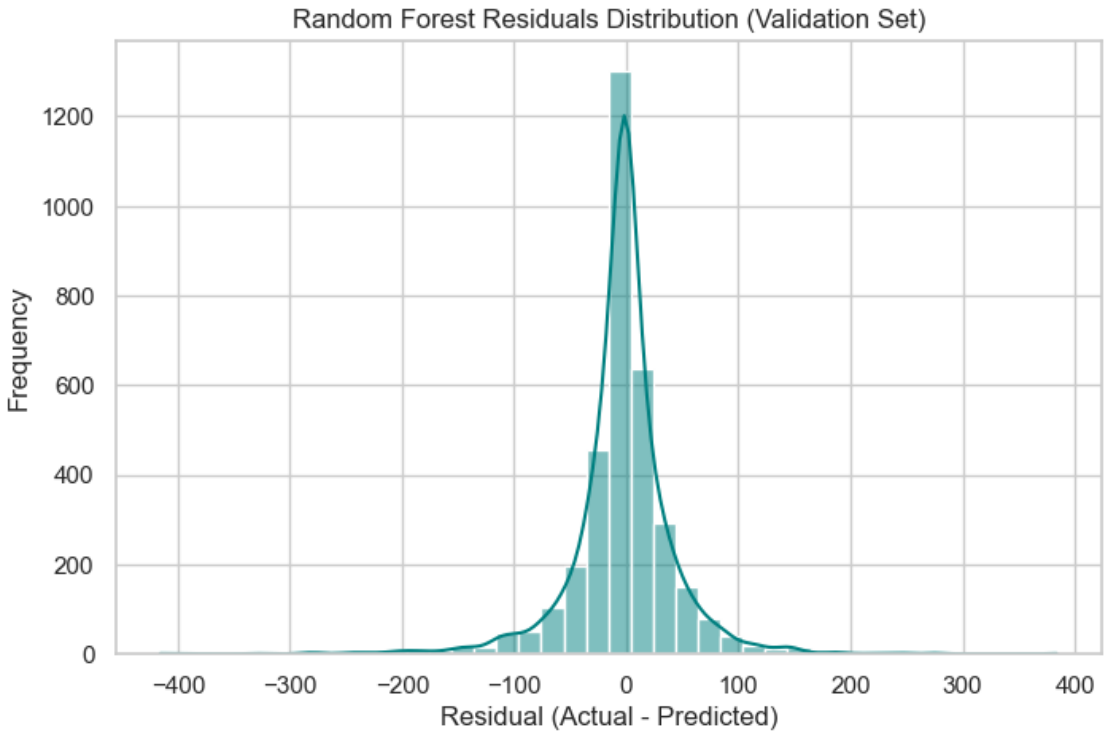
The year (yr) is also significant, suggesting an increasing trend in bike rentals over time. Other relevant features include working day status, humidity, and the interaction between temperature and humidity (temp_hum), all of which likely influence the comfort and practicality of biking.

Plot - Actual vs Predicted



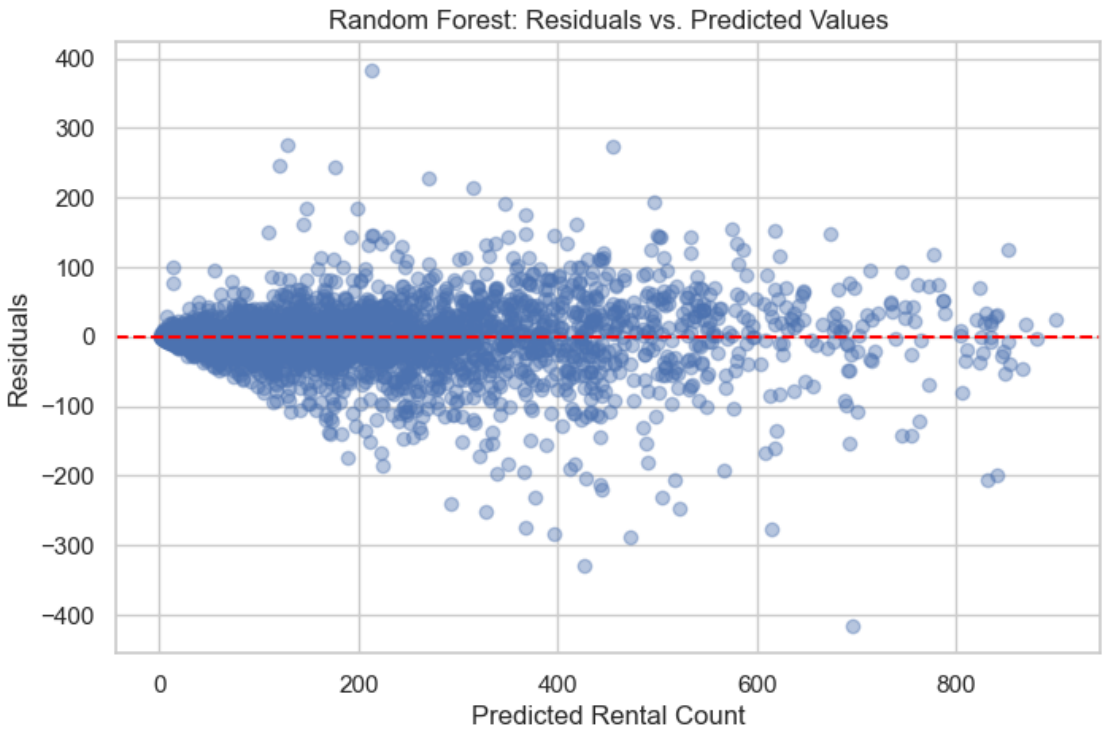
The predictions align much more closely with the actual bike rentals compared to the baseline model. Most of the points fall tightly around the diagonal line, which is a great sign. This indicates that the model is accurately capturing the patterns in the data and making reliable predictions.

Plot - Residuals Distribution



The residuals are now tightly clustered around 0, showing that the prediction errors are generally small. The distribution is sharply peaked and more symmetric than in the linear regression model, although there is still a slight right skew. Overall, the Random Forest model is making consistently accurate predictions.

Plot - Residuals vs Predicted



The residuals appear randomly scattered with no clear pattern, which is exactly what we want. This randomness suggests the model is not missing any obvious nonlinear relationships. Additionally, the residuals are more homoscedastic, which means that they're evenly spread across the range of predicted values, which indicates a good model fit across different rental counts

Overall, the Random Forest model shows both low bias and low variance on the validation set. It captures the underlying structure of the data very well and generalizes effectively. While these results are promising, we'll validate further performance on the test set to ensure robustness.

6 Task 6: Gradient Boosting Regressor

```
Requirement already satisfied: xgboost in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (3.0.0)
Requirement already satisfied: numpy in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from xgboost)
(1.24.3)
Requirement already satisfied: scipy in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from xgboost)
(1.10.1)
Note: you may need to restart the kernel to use updated packages.
```

```
XGBoost Evaluation on Validation Set:
Mean Squared Error (MSE): 1907.75
Mean Absolute Error (MAE): 27.97
```

```
Root Mean Squared Error (RMSE): 43.68
R² Score: 0.9433
```

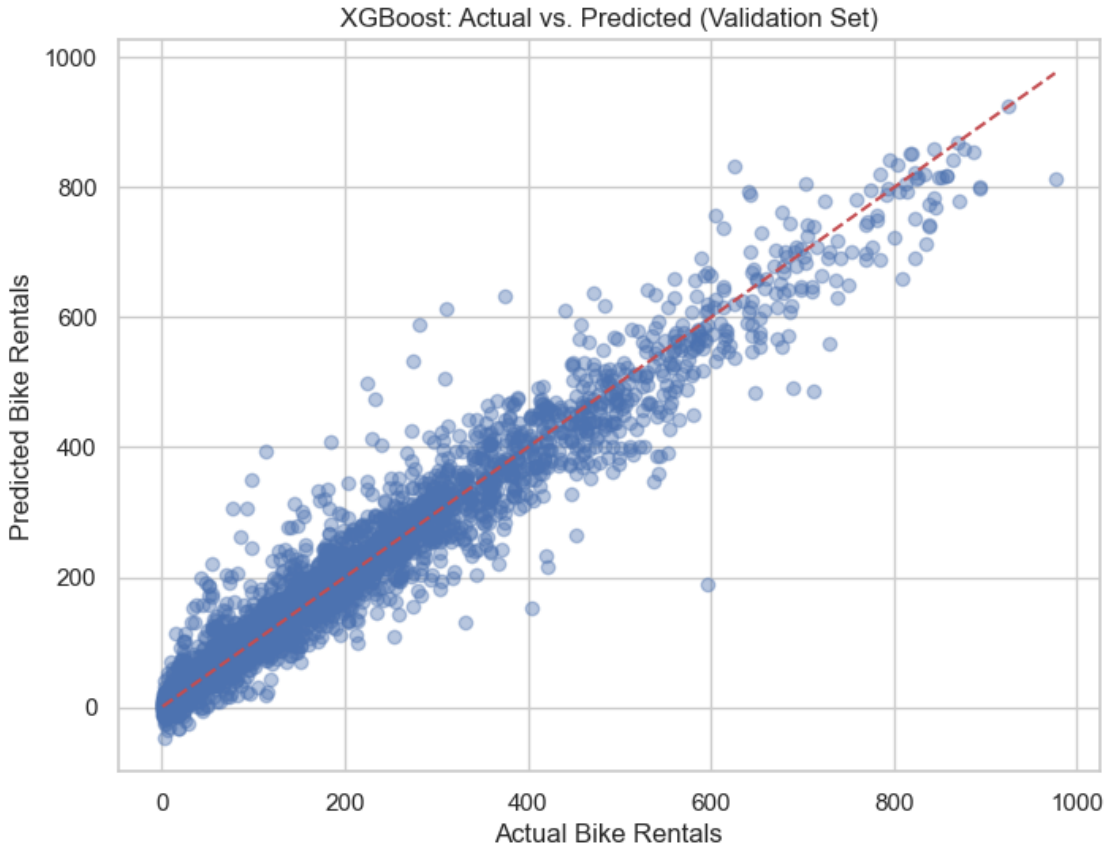
We can see that Linear Regression performed the worst across all evaluation metrics. It had a high Mean Absolute Error (MAE) of 106, and a Root Mean Squared Error (RMSE) of 142, indicating that its predictions were quite far off from the actual values. The model could only explain 40% of the variance in bike rental counts, which is relatively low and reflects its inability to capture the complex, non-linear patterns present in the data.

In contrast, the Random Forest Regressor made a significant improvement. It reduced the MAE from 106 to 28, and the RMSE dropped from 142 to 46, making the predictions much more accurate overall. The R^2 score increased dramatically from 0.40 to 0.94, meaning it could now explain 94% of the variance in bike rentals — a huge leap in performance. This clearly demonstrates the advantage of using non-linear ensemble models that can capture interactions between features.

Finally, XGBoost slightly outperformed Random Forest. It achieved the lowest RMSE at 43.68, a slightly lower MAE of 27.97, and the highest R^2 score of 0.9433. These results suggest that XGBoost was able to capture even more subtle patterns in the data, likely due to its boosting mechanism that corrects errors made by earlier trees in the ensemble.

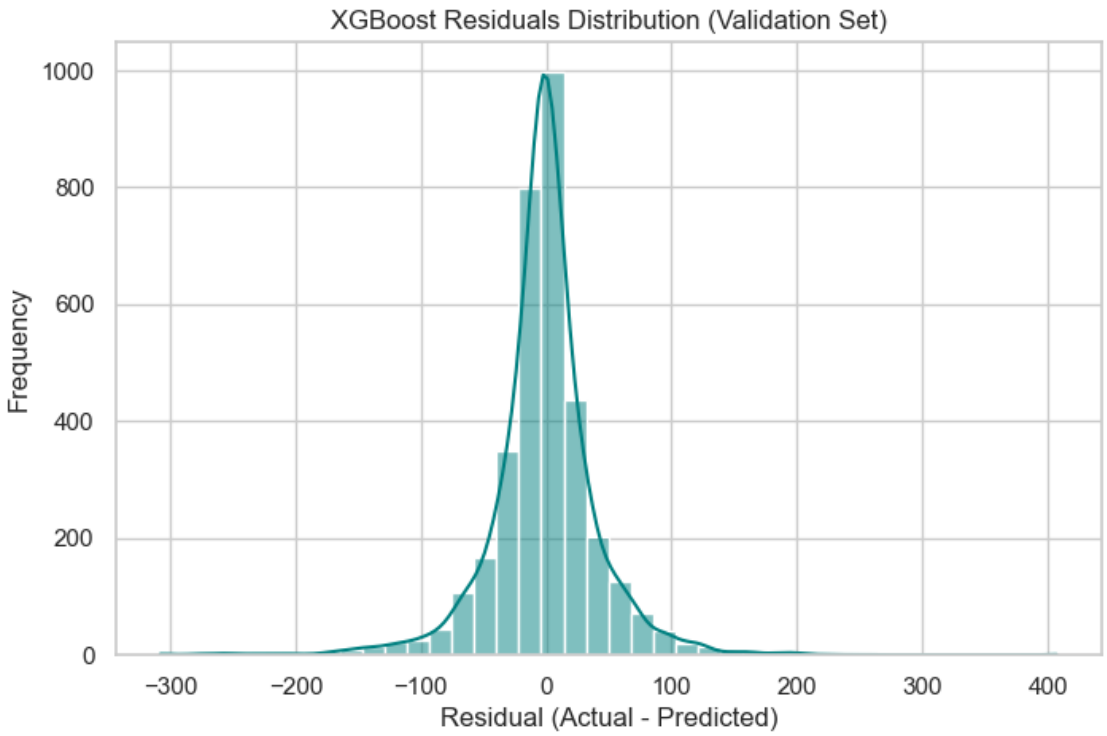
Overall, both RF and XGB models show a major improvement over the baseline, and XGBoost stands out as the best performer in this task.

Plot - Actual vs Predicted



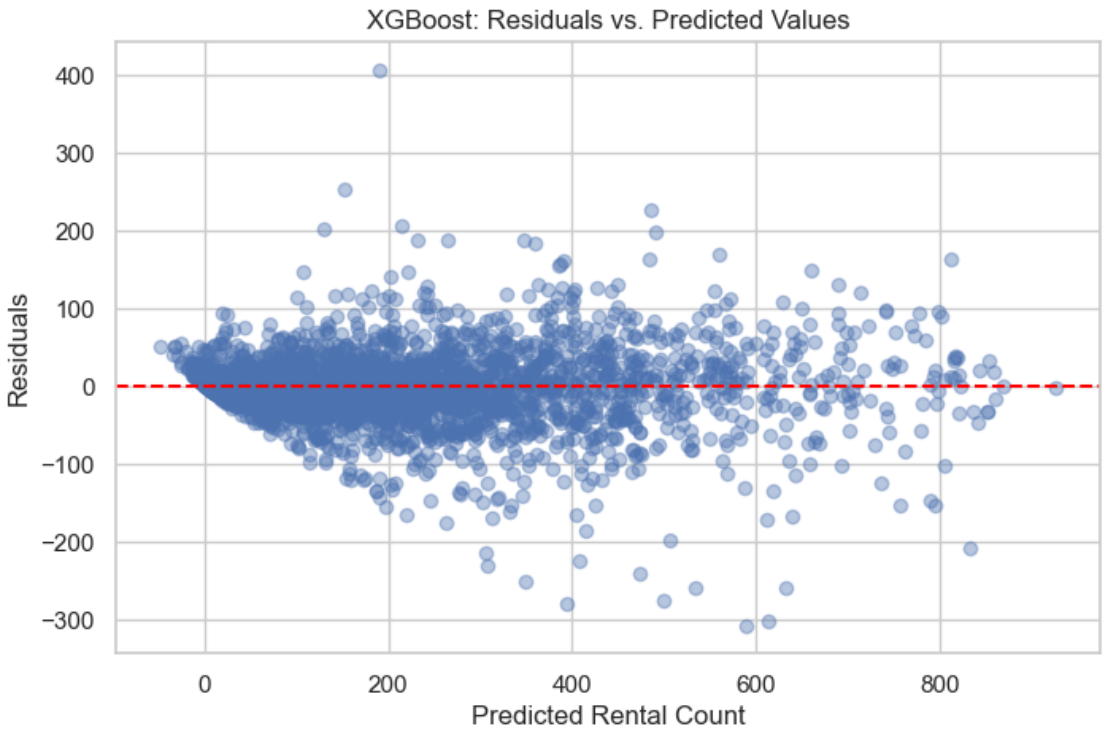
The predictions from the XGBoost model align extremely well with the actual rental values, clustering closely around the diagonal line. This suggests that the model has captured the underlying patterns in the data with high accuracy. There are still a few outliers, especially at higher rental counts, but overall, the points are much more concentrated compared to the linear regression plot; showing tighter and more reliable predictions.

Plot - Residuals Distribution



The residuals are tightly concentrated around zero, with a clear peak and relatively symmetric shape, indicating that most predictions are very close to the actual values. Compared to the baseline and even the Random Forest model, this distribution appears narrower and more focused, which a strong sign of low prediction error. There's still a bit of right skew, but it's minimal. This supports the earlier metrics showing that XGBoost produces highly accurate predictions.

Plot - Residuals vs Predicted



This plot shows that the residuals are fairly centered around zero across the range of predicted values, which is exactly what we want to see. There's no strong visible pattern, indicating that the model doesn't consistently overpredict or underpredict for certain rental volumes. While there's some increased spread at higher predicted values (a bit of heteroscedasticity), it's much more controlled than with Linear Regression. This reinforces that the model generalizes well and doesn't suffer from major bias or variance issues.

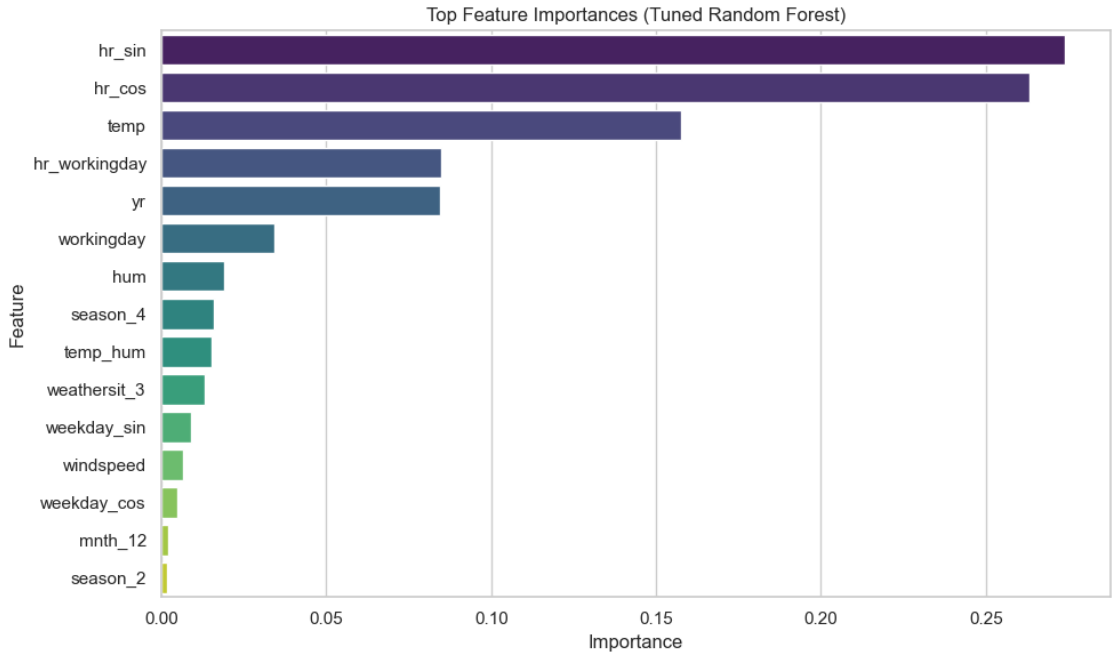
7 Task 7: Hyperparameter Tuning

7.1 Random Forest

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Tuned Random Forest Results:
Best Hyperparameters: {'max_depth': 19, 'min_samples_leaf': 1,
'min_samples_split': 8, 'n_estimators': 58}
RMSE: 46.71
MAE: 28.63
R²: 0.9351
```

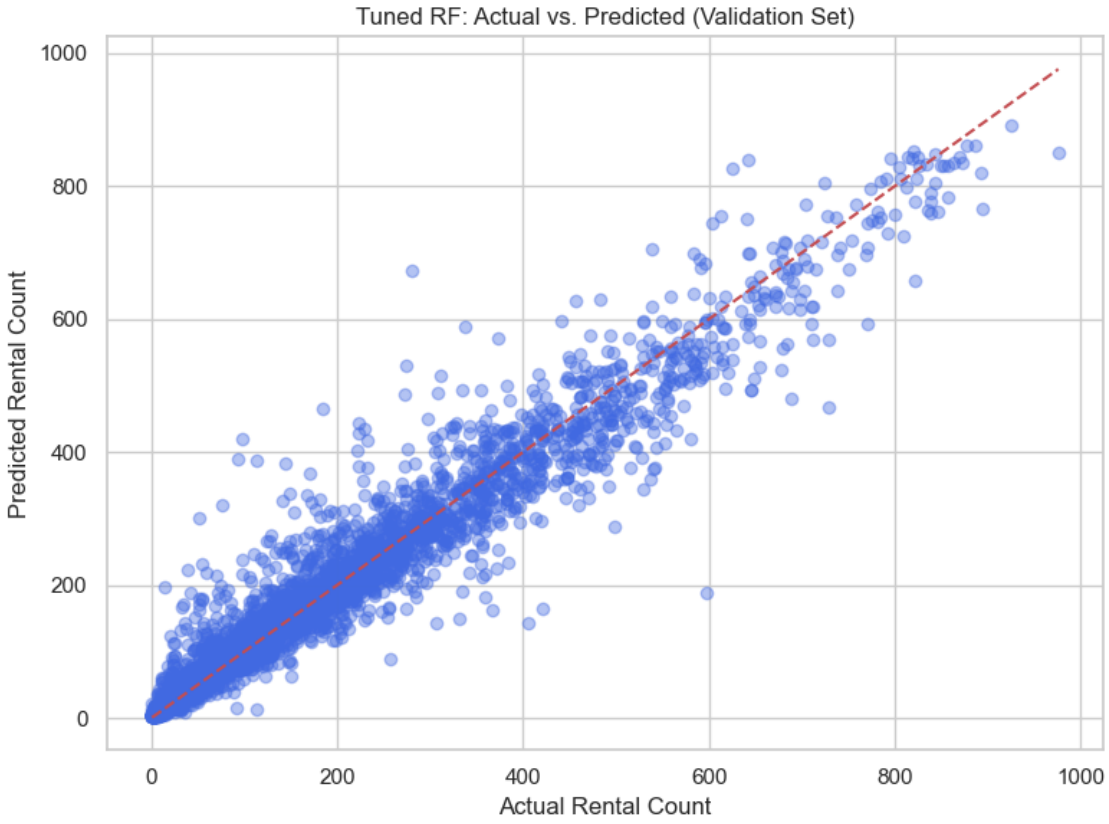
Compared to the untuned Random Forest (RMSE: 46.25, R^2: 0.9364), performance slightly dropped, indicating that tuning didn't significantly improve results.

The best hyperparameter combination suggests the model performs best with moderately deep trees and a relatively low number of estimators.



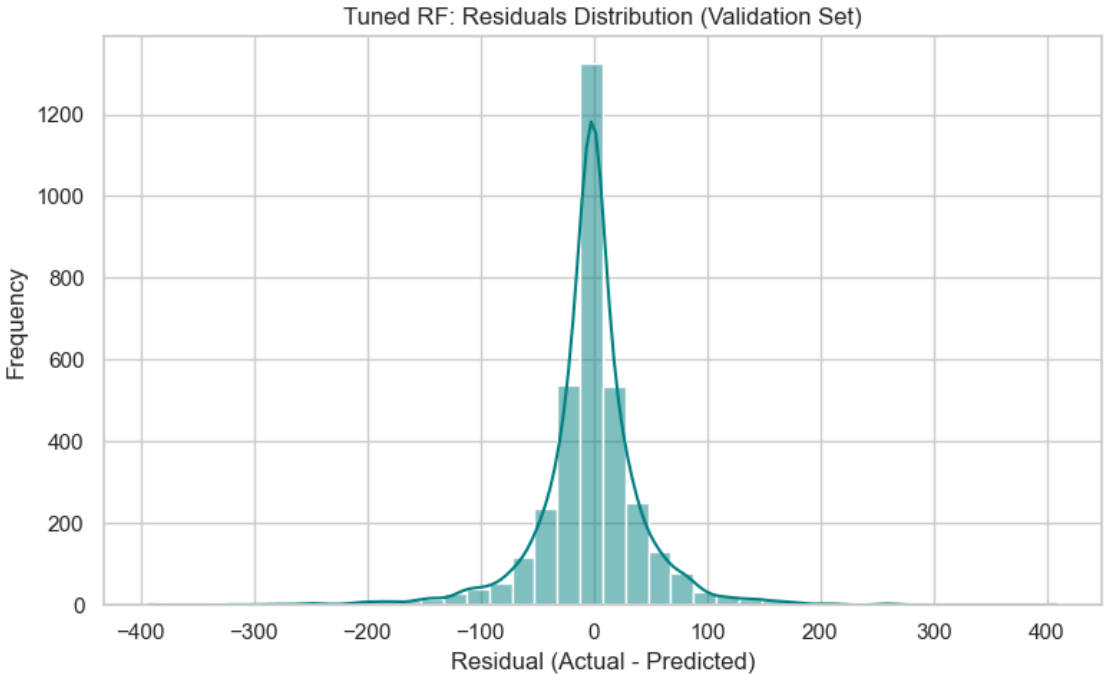
The top features remained consistent: hr_sin, hr_cos, and temp were still the most influential. This shows that even after tuning, the model relies on similar patterns in the data, confirming the robustness of the original feature engineering

Plot - Actual vs Predicted



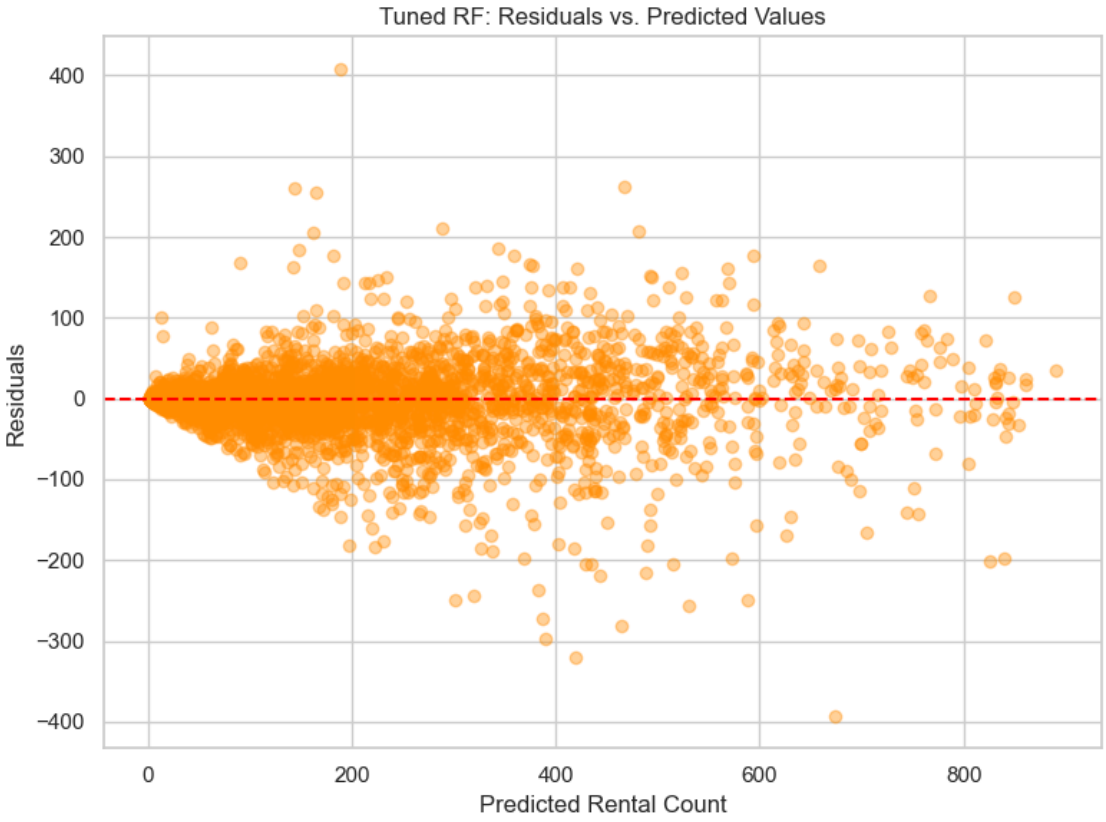
The predictions remain closely aligned with the actual values, following the diagonal line quite well. This indicates that the tuned Random Forest continues to make accurate predictions with little deviation from the real thing.

Plot - Residual Distribution



The residuals are still tightly centered around zero, suggesting low bias. The peak is slightly sharper than before, indicating that the tuning may have helped reduce moderate errors and focus the predictions even more.

Plot - Residuals vs Predicted



The residuals remain fairly homoscedastic, though we still observe a slightly funnel-shaped pattern with increasing variance at higher predicted counts. However, there is no major pattern, so the model appears to generalize well.

7.2 XGB

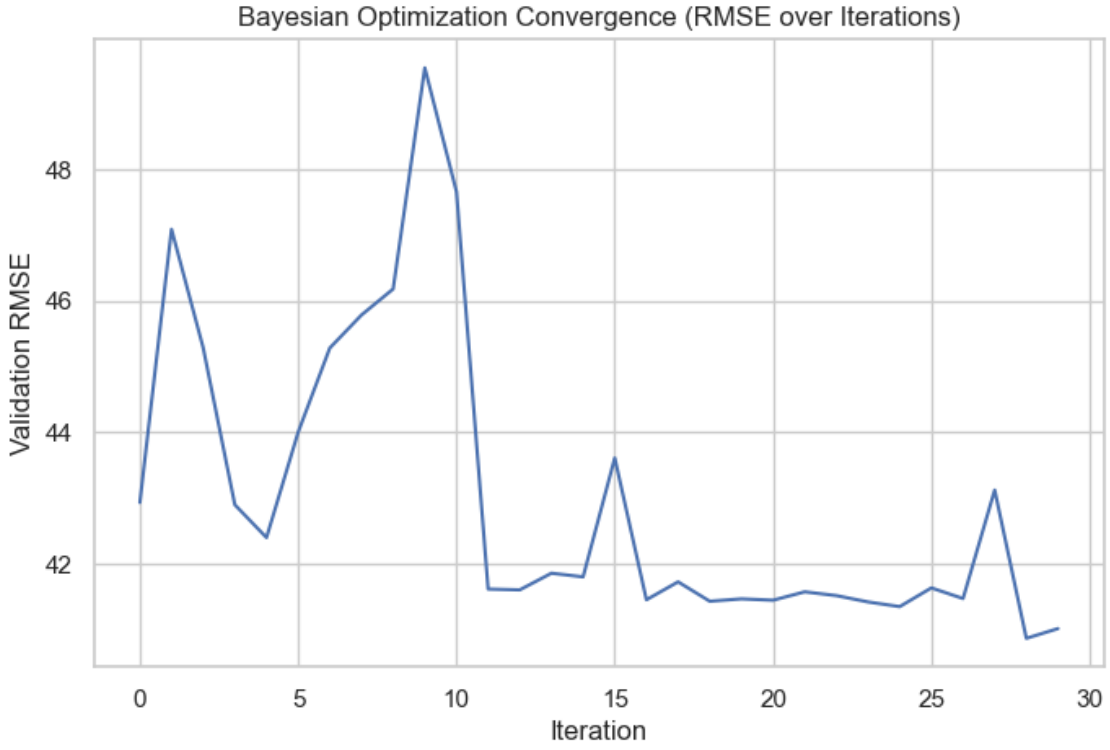
```
Requirement already satisfied: scikit-optimize in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (0.10.2)
Requirement already satisfied: joblib>=0.11 in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from scikit-
optimize) (1.4.2)
Requirement already satisfied: pyaml>=16.9 in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from scikit-
optimize) (25.1.0)
Requirement already satisfied: numpy>=1.20.3 in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from scikit-
optimize) (1.24.3)
Requirement already satisfied: scipy>=1.1.0 in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from scikit-
optimize) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.0 in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from scikit-
```

```
optimize) (1.3.0)
Requirement already satisfied: packaging>=21.3 in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from scikit-
optimize) (23.1)
Requirement already satisfied: PyYAML in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from
pyaml>=16.9->scikit-optimize) (6.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/Users/sofiagonzalez/anaconda3/lib/python3.11/site-packages (from scikit-
learn>=1.0.0->scikit-optimize) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Tuned XGBoost Results:
Best Hyperparameters: OrderedDict([('learning_rate', 0.05624593012982076),
('max_depth', 8), ('n_estimators', 300), ('subsample', 0.5)])
RMSE: 40.95
MAE: 24.91
R²: 0.9501
```

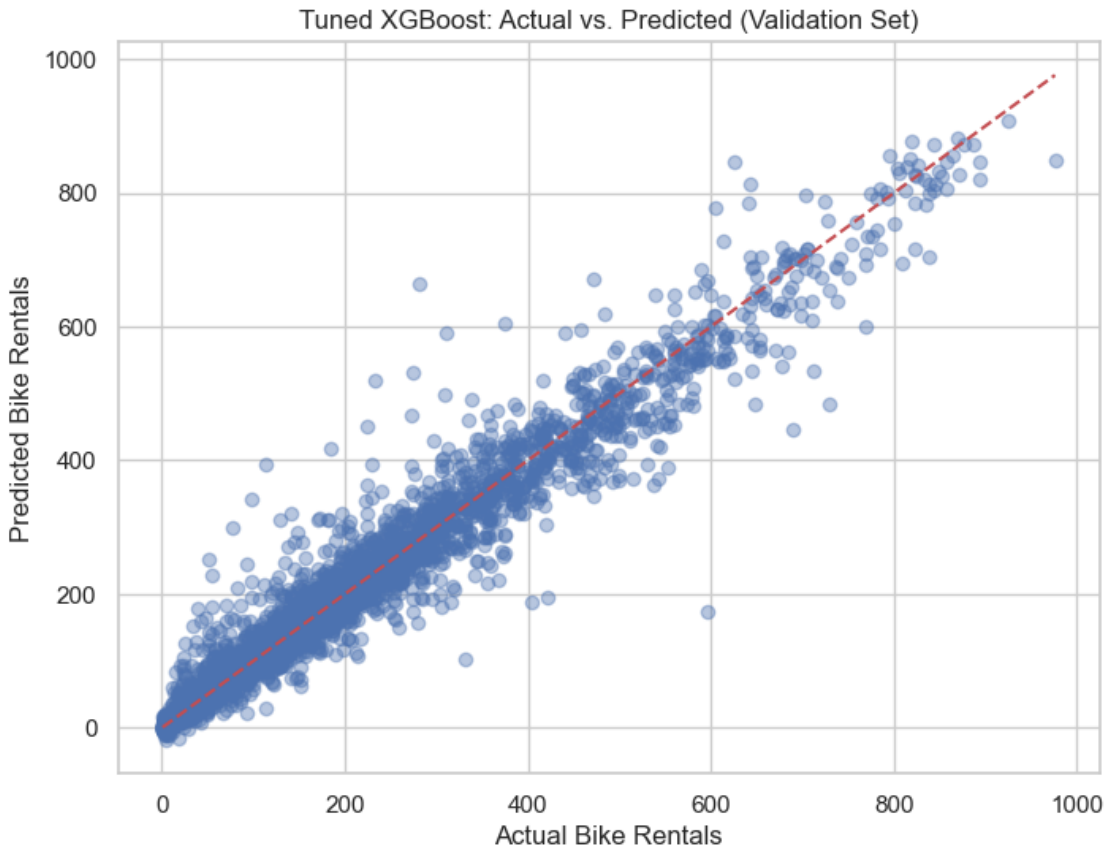
These are the best scores yet! Compared to the untuned XGBoost (RMSE: 43.68, R^2: 0.9433), this shows a noticeable boost in performance. The lower learning rate likely helped prevent overfitting while maintaining accuracy by using more estimators. The subsample of 0.5 encourages diversity in the trees, which can also reduce variance and improve generalization.

This shows that hyperparameter tuning via Bayesian Optimization was worth it. The tuned model is better at capturing the underlying patterns in the data and generalizes better on the validation set. The improvement isn't massive, but it's consistent, which is a good sign of a well-tuned model.



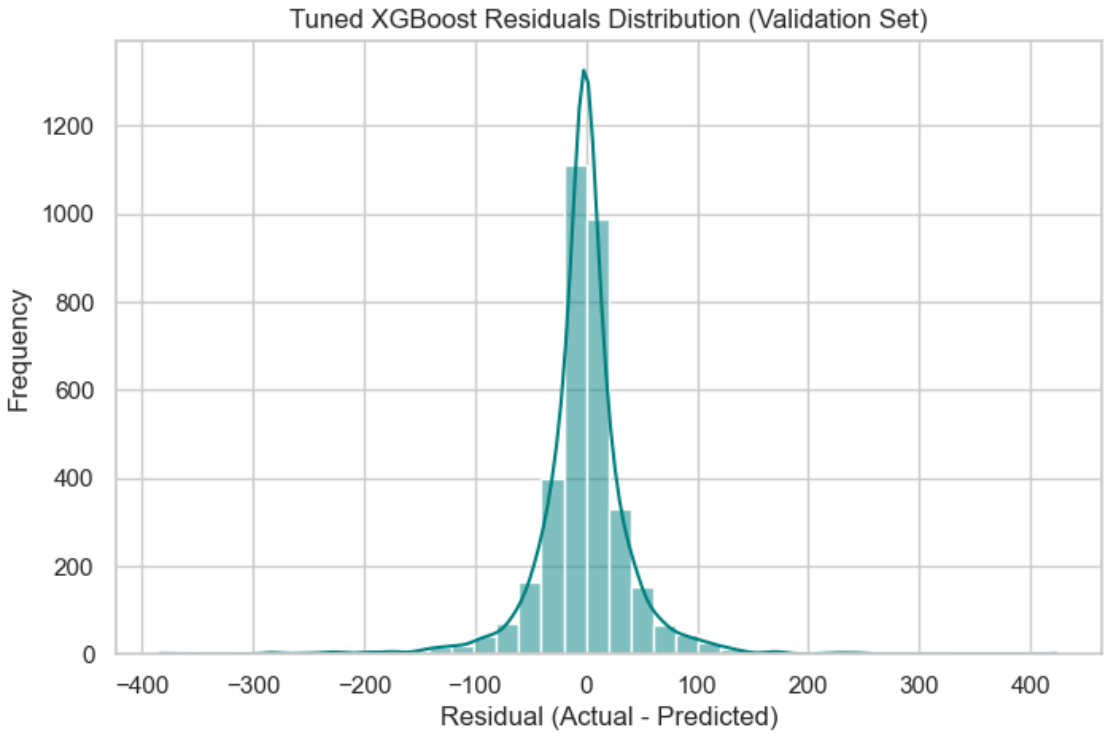
This plot shows that early iterations had a wide range of RMSE values (some even above 48), but after around iteration 10, the RMSE drops significantly and stabilizes near 41, which is a clear sign that Bayesian Optimization found a good region in the hyperparameter space. The relatively flat curve afterward suggests that the optimizer converged — further tuning would likely give diminishing returns.

Plot - Actual vs Predicted



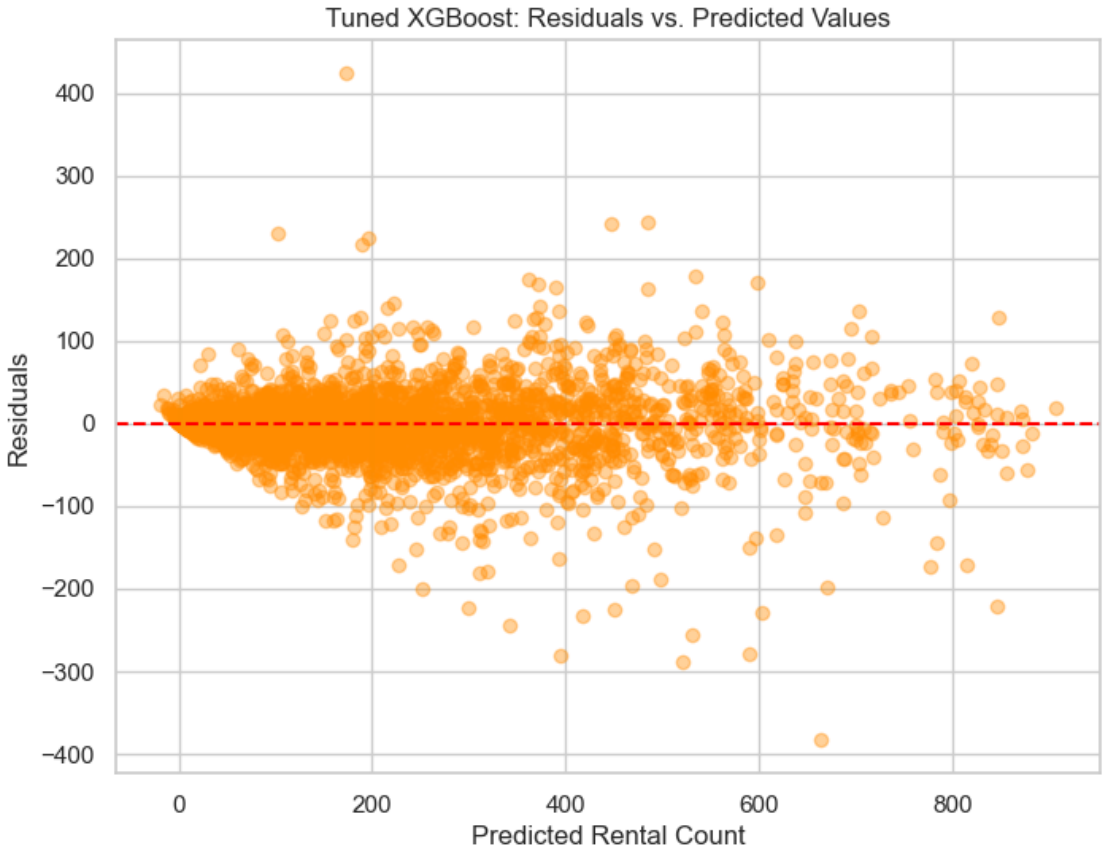
The points align very closely with the diagonal, showing that the tuned XGBoost model makes highly accurate predictions. There is minimal spread, especially in the mid-range values, which reflects improved generalization.

Plot - Residuals Distribution



The residuals are tightly centered around 0 with a sharp peak, indicating that most predictions are very close to the actual values. This suggests low bias and fewer large errors.

Plot - Residuals vs Predictions



The residuals are evenly spread across the predicted range with no clear pattern, confirming the model's errors are randomly distributed. The slight fan shape is still present but less pronounced compared to the untuned version.

For Random Forest, tuning slightly reduced performance on the validation set (RMSE increased from 46.25 to 46.71, and R^2 dropped from 0.9364 to 0.9351). This suggests that the untuned model was already near-optimal and the randomized search may have landed on a configuration that slightly underperformed. The feature importance plot remained nearly identical, reinforcing that the model's interpretation and decision structure didn't change significantly. Overall, there were no signs of overfitting, but tuning didn't provide a clear advantage.

In contrast, XGBoost showed more meaningful gains from tuning. The RMSE improved from 43.68 to 40.95, and R^2 increased from 0.9433 to 0.9501. This improvement suggests that Bayesian Optimization effectively identified better hyperparameter combinations, enabling the model to capture more subtle patterns without overfitting. The convergence plot confirmed steady progress in reducing validation error over iterations.

8 Task 8: Iterative Evaluation and Refinement

After comparing the performance of all three models, Linear Regression, Random Forest, and XGBoost, it became clear that the tuned XGBoost Regressor outperformed the others across all

evaluation metrics. It achieved the lowest error values (RMSE: 40.95, MAE: 24.91) and the highest R^2 score of 0.9501, indicating that it explains over 95% of the variance in bike rental demand.

This strong performance, coupled with its ability to model complex non-linear relationships, makes XGBoost the most promising candidate for final refinement. Task 8 is about iterating and polishing, and it makes the most sense to apply those refinements to the model that is already working best.

By focusing our efforts here, we can ensure that we get the most out of our data and push this already high-performing model to its full potential before locking it in for final testing in Task 9.

So first we should see the current model performance, which is: - RMSE: 40.95 - MAE: 24.91 - R^2: 0.9501

This is already really good but let's see if we can tweak it a bit with better features

In Task 3, we initially added a few interaction terms based on intuition and EDA: - temp × hum: because high temperature and high humidity together likely affect user comfort and thus rentals. - hr_cos × workingday: to model how rental patterns change across work hours vs. weekends. - season_2 × weathersit_2: a basic example to test the interaction between weather and time of year.

We also dropped atemp as it was highly correlated to temp. Let's see if we can drop more things, and we do! According to the correlation matrix done in Task 1 we can also see that holiday has zero correlation and might just be noise and weekday, which is also has very little correlation

We can also see if we can drop month or season, since they are both strongly correlated

lets retrain the XGB

```
Tuned XGBoost on Cleaned Features:
RMSE: 43.62
MAE: 26.68
R²: 0.9434
```

We can see that dropping holiday and mnth actually made the model slightly worse. So, we should keep mnth and holiday in our final feature set.

Even though holiday and mnth showed low correlation with cnt, XGBoost was likely using subtle patterns in those features.

Lets now try to drop one of our correlations to see if it was actually helping: temp_hum

```
Tuned XGBoost without temp_hum:
RMSE: 40.86
MAE: 24.84
R²: 0.9504
```

So we will keep this one dropped. Although temperature and humidity seemed correlated during EDA, removing the temp_hum interaction actually improved model performance slightly across all metrics. This suggests the model already captures the relationship between temp and hum independently, so the interaction term wasn't helpful.

We are now going to try to drop hr_workingday to see how much it was actually helping

```
Tuned XGBoost without hr_workingday:
RMSE: 41.28
```

```
MAE: 25.14
R²: 0.9493
```

Removing hr_workingday slightly worsened the model's performance across all metrics. Even though the difference is small, it suggests that hr_workingday provides marginal value, likely helping the model capture behavioral patterns tied to work hours.

Now we will see season_weathersit

```
Tuned XGBoost without season_weathersit:
RMSE: 40.95
MAE: 24.88
R²: 0.9501
```

We can see that removing season_weathersit had almost no impact on the model's performance. This is a good sign, it means that the model is robust and not overly reliant on a specific interaction term. We will remove it as it is better to keep the model simpler

So for the final model I ended up choosing XGB Regressor as it was the one that performed better and it generalized very well. It was also balanced and it had low variance and bias

9 Task 9: Final Model Selection and Training

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.0562, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=8,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=300,
             n_jobs=None, num_parallel_tree=None, …)
```

```
Final Model Performance on Test Set:
Mean Squared Error (MSE): 1366.23
Mean Absolute Error (MAE): 22.84
Root Mean Squared Error (RMSE): 36.96
R² Score: 0.9579
```

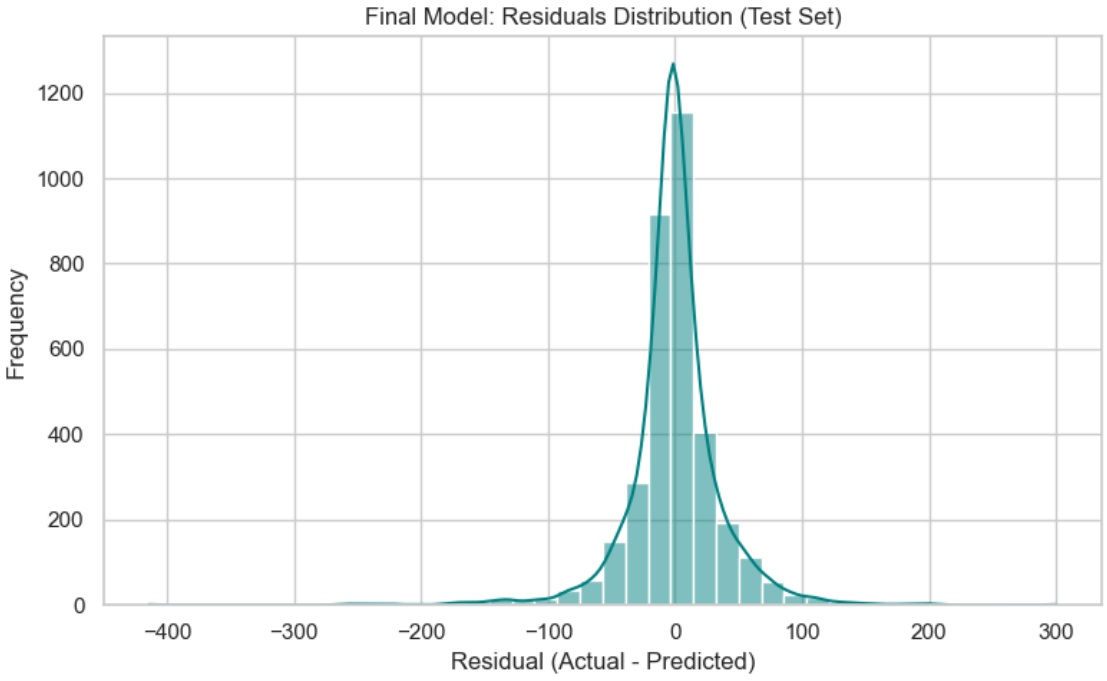
After retraining the model on the combined training and validation sets and evaluating on the untouched test set, we obtained the following results: - Mean Squared Error (MSE): 1366.23 - Mean Absolute Error (MAE): 22.84 - Root Mean Squared Error (RMSE): 36.96 - R^2 Score: 0.9579

These metrics confirm that the final model generalizes extremely well to unseen data. It explains 95.8% of the variance in bike rentals and has a very low average prediction error, which demonstrates both low bias and low variance.

Plot - Actual vs Predicted

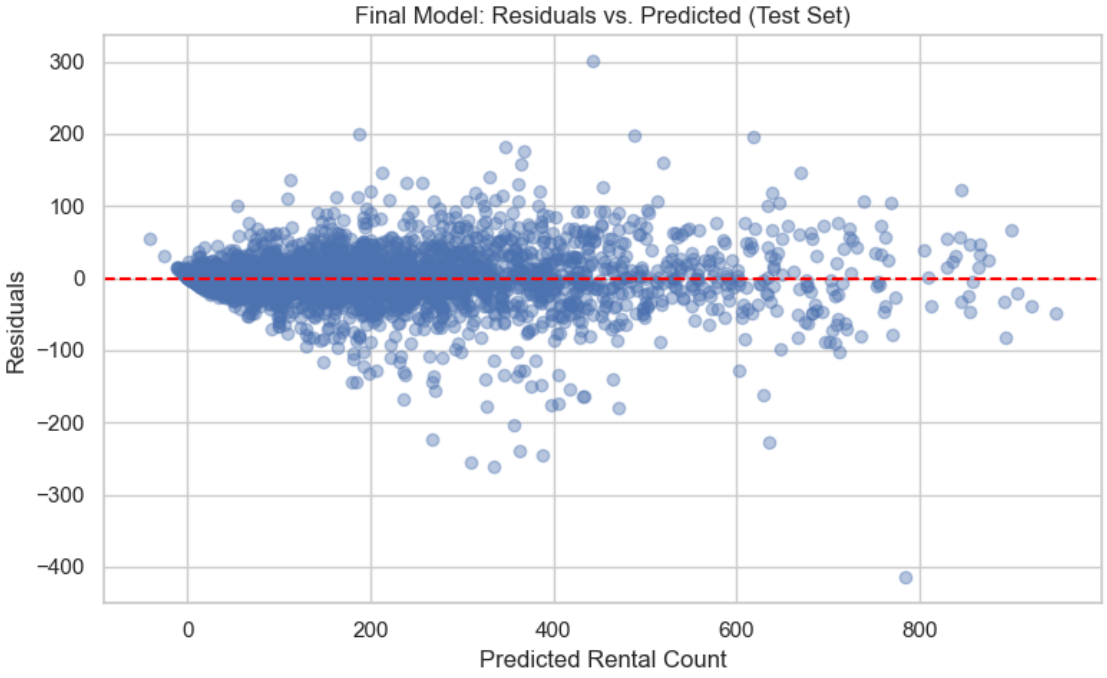
Predictions align very closely with the actual values, forming a tight cluster around the diagonal. This indicates that the model generalizes extremely well and has low bias on unseen data

Plots - Residual Distribution



The residuals are sharply centered around zero and symmetrically distributed, showing low variance and no major outliers or skew. This confirms that prediction errors are small and normally distributed, which is ideal for a well-performing model.

Plots - Residuals vs Predicted



Residuals are evenly spread without a clear pattern, which is a good sign of homoscedasticity. This means the model's errors are consistent across different prediction values and not systematically biased.

10 Conclusions

```
      Metric  Validation (Before Tuning)  Validation (After Tuning)  \
0       RMSE                     43.6800                    40.9500
1        MAE                     27.9700                    24.9100
2  R^2 Score                      0.9433                     0.9501

   Test Set (Final)
0           36.9600
1           22.8400
2            0.9579
```

The progressive improvements in RMSE, MAE, and R^2 demonstrate how impactful model tuning and iterative refinement can be. Starting with a strong baseline XGBoost model, we significantly enhanced its predictive power through Bayesian optimization and targeted feature engineering.

Tuning reduced the RMSE from 43.68 to 40.95 on the validation set, and further refining the features dropped the final test RMSE to 36.96, showing great generalization. Likewise, MAE decreased by over 5 points, and the R^2 score rose from 0.9433 to 0.9579, confirming the model's improved ability to explain variance in bike rentals.

These gains reflect not just better hyperparameters, but also smarter features and a model that now balances bias and variance more effectively. Overall, the final XGBoost model captures the complexity of rental patterns while remaining robust across unseen data.