Bicycle rental prediction

June 29, 2025

0.1 About the Author

ABIODUN OGUNTOLA
Data scientist passionate about solving real-world problems with data and machine learning. Always curious, always building.

Connect with me on [www.linkedin.com/in/abiodun-oguntola-811748224)

0.2 Bicycle Rental Demand Prediction

This project uses historical weather and calendar data to predict daily bicycle rental counts. It walks through data exploration, feature engineering, and linear regression modeling using Python and scikit-learn.

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
plt.rcParams['figure.figsize'] = [10,5]
```

0.3 Loading and Preparing the Dataset

We begin by loading the `day.csv` file, which contains daily bicycle rental data along with weather and calendar features. We also convert the date column into a proper datetime format for time-based operations.

[2]:
```
# Load the dataset
df = pd.read_csv("day.csv")

# Convert string date to datetime object
df['date'] = pd.to_datetime(df['dteday'])

# Display the first few rows
df.head()
```

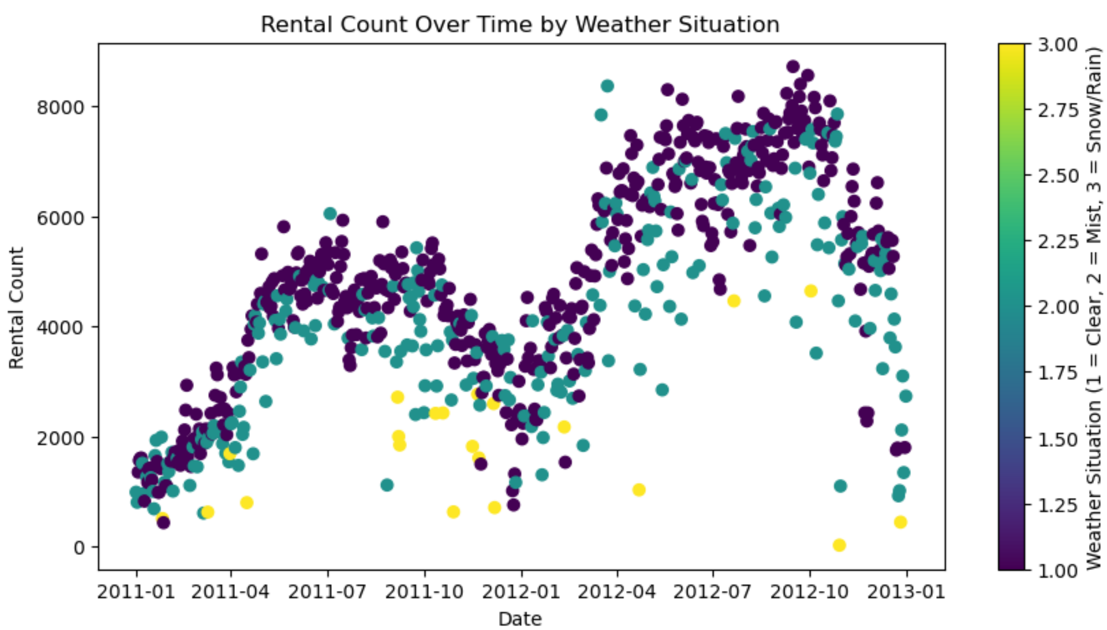
[2]:
```
   instant      dteday  season  yr  mnth  holiday  weekday  workingday  \
0        1  2011-01-01       1   0     1        0        6           0
1        2  2011-01-02       1   0     1        0        0           0
2        3  2011-01-03       1   0     1        0        1           1
```

```
3         4  2011-01-04       1    0      1         0         2            1
4         5  2011-01-05       1    0      1         0         3            1

   weathersit      temp     atemp       hum  windspeed  casual  registered  \
0           2  0.344167  0.363625  0.805833   0.160446     331         654
1           2  0.363478  0.353739  0.696087   0.248539     131         670
2           1  0.196364  0.189405  0.437273   0.248309     120        1229
3           1  0.200000  0.212122  0.590435   0.160296     108        1454
4           1  0.226957  0.229270  0.436957   0.186900      82        1518

    cnt        date
0   985  2011-01-01
1   801  2011-01-02
2  1349  2011-01-03
3  1562  2011-01-04
4  1600  2011-01-05
```

0.4 Exploratory Data Analysis (EDA)

In this section, we visualize patterns in rental behavior across dates and weather conditions. We'll explore how weather impacts rental count and examine key correlations between variables.

[3]:
```python
# Plot rentals over time colored by weather situation
plt.scatter(df['date'], df['cnt'], c=df['weathersit'], cmap='viridis')
plt.title("Rental Count Over Time by Weather Situation")
plt.xlabel("Date")
plt.ylabel("Rental Count")
plt.colorbar(label="Weather Situation (1 = Clear, 2 = Mist, 3 = Snow/Rain)")
plt.show()
```



0.4.1 Average Rentals by Weather Situation

We calculate the average number of rentals for each weather category to see how different weather conditions influence demand. Weather situations typically correspond to: - 1: Clear, Few clouds - 2: Mist + Cloudy - 3: Light Snow, Light Rain

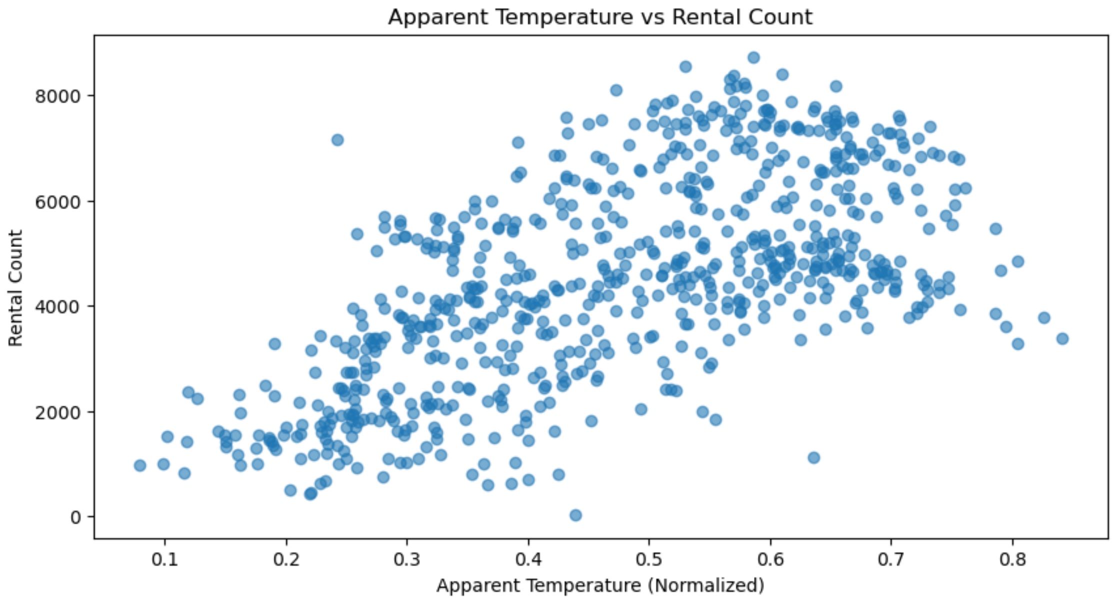
[4]:
```
# Calculate and print the average rental count for each weather situation
clear_avg = df[df['weathersit'] == 1]['cnt'].mean()
mist_avg = df[df['weathersit'] == 2]['cnt'].mean()
bad_weather_avg = df[df['weathersit'] == 3]['cnt'].mean()

print(f"Clear weather average rentals: {clear_avg:.2f}")
print(f"Mist/Cloudy average rentals: {mist_avg:.2f}")
print(f"Snow/Rain average rentals: {bad_weather_avg:.2f}")
```

```
Clear weather average rentals: 4876.79
Mist/Cloudy average rentals: 4035.86
Snow/Rain average rentals: 1803.29
```

0.4.2 Correlation Between Features and Rental Count

We'll visually and numerically explore how features like temperature, humidity, and windspeed relate to bike rental counts. Strong correlations can signal which variables are most influential.

[5]:
```
# Scatter plot: Apparent temperature vs rental count
plt.scatter(df['atemp'], df['cnt'], alpha=0.6)
plt.title("Apparent Temperature vs Rental Count")
plt.xlabel("Apparent Temperature (Normalized)")
plt.ylabel("Rental Count")
plt.show()
```



0.4.3 Numerical Correlation Analysis

We calculate Pearson correlation coefficients between `cnt` and other numeric features. This tells us how strongly each variable is linearly related to rental count — values close to 1 or –1 indicate strong relationships.

[6]:
```
# Correlation matrix between cnt and other features
correlation_matrix = df[['cnt', 'atemp', 'temp', 'hum', 'windspeed',␣
↪'weathersit']].corr()
correlation_matrix
```

[6]:

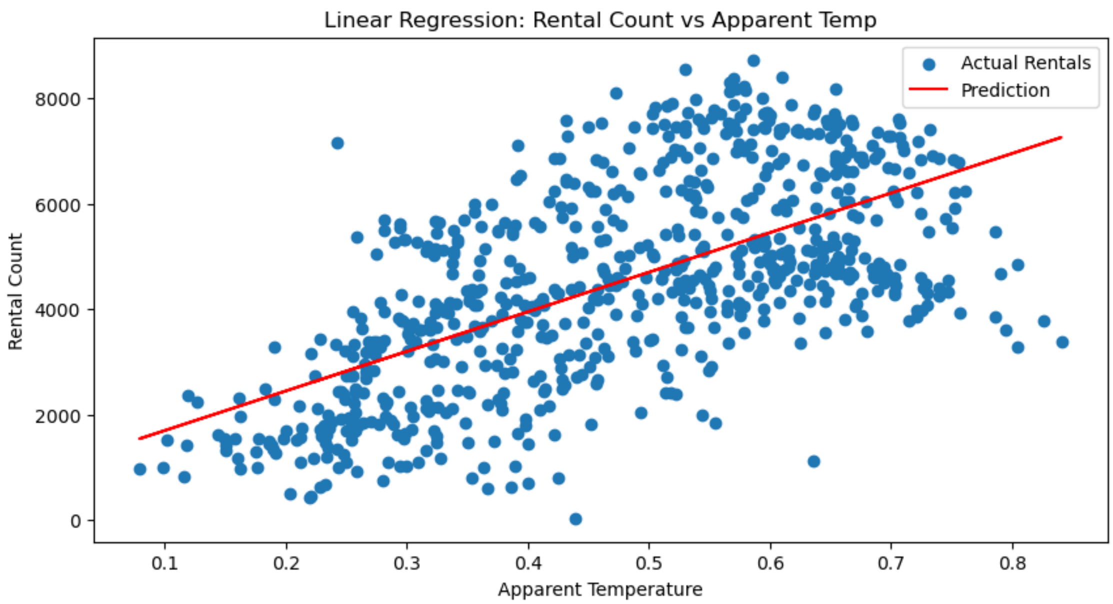
	cnt	atemp	temp	hum	windspeed	weathersit
cnt	1.000000	0.631066	0.627494	-0.100659	-0.234545	-0.297391
atemp	0.631066	1.000000	0.991702	0.139988	-0.183643	-0.121583
temp	0.627494	0.991702	1.000000	0.126963	-0.157944	-0.120602
hum	-0.100659	0.139988	0.126963	1.000000	-0.248489	0.591045
windspeed	-0.234545	-0.183643	-0.157944	-0.248489	1.000000	0.039511
weathersit	-0.297391	-0.121583	-0.120602	0.591045	0.039511	1.000000

0.5 Simple Linear Regression (cnt vs atemp)

We'll fit a basic linear regression model using only one feature: apparent temperature (`atemp`). This helps us understand how temperature alone predicts rental counts and gives us a baseline RMSE for comparison later.

[7]:
```
from sklearn.linear_model import LinearRegression
```

```
# Simple linear regression: cnt ~ atemp
lr = LinearRegression()
lr.fit(df['atemp'].values.reshape(-1,1), df['cnt'].values.reshape(-1,1))

# Plot actual data and regression line
plt.scatter(df['atemp'], df['cnt'], label='Actual Rentals')
plt.plot(df['atemp'], lr.predict(df['atemp'].values.reshape(-1,1)), c='red',␣
↪label='Prediction')
plt.title("Linear Regression: Rental Count vs Apparent Temp")
plt.xlabel("Apparent Temperature")
plt.ylabel("Rental Count")
plt.legend()
plt.show()
```



0.6 Train/Test Split Based on Date

We split the dataset into two parts: - **Training set**: data before June 1, 2012 - **Validation set**: data from June 1, 2012 onward

This ensures we respect the time-based nature of the data and avoid future data leakage.

[8]:
```
# Create training and validation sets based on date
training_set = df[df['date'] < '2012-06-01']
validation_set = df[df['date'] >= '2012-06-01']

# Select input features and output
features = ['atemp', 'workingday', 'hum', 'weathersit']
training_inputs = training_set[features].values
```

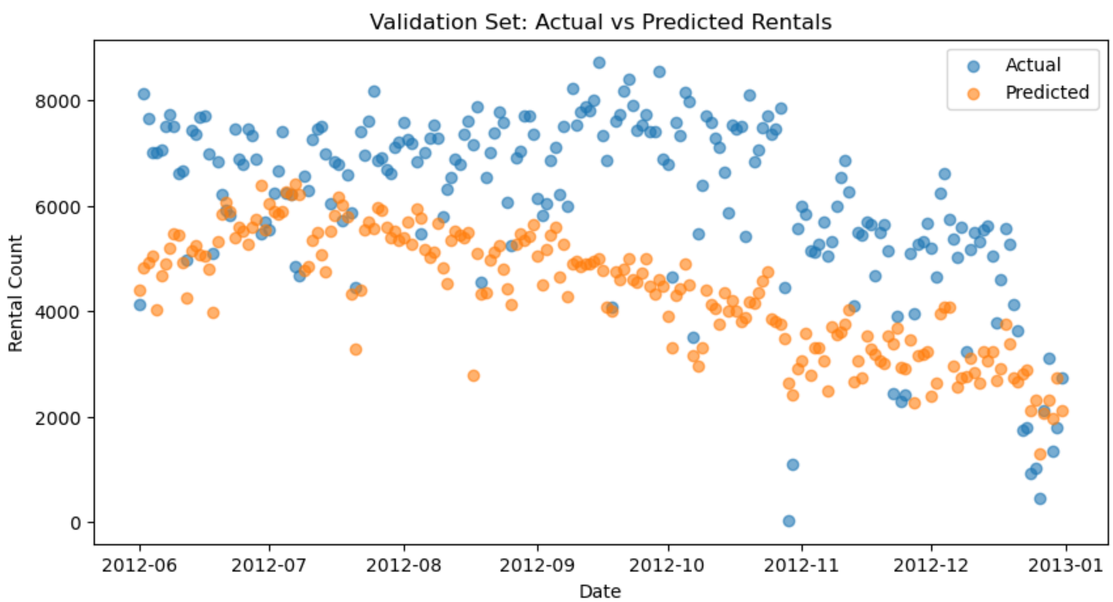
```
training_outputs = training_set[['cnt']].values

validation_inputs = validation_set[features].values
validation_outputs = validation_set[['cnt']].values

# Train linear regression model
lr = LinearRegression()
lr.fit(training_inputs, training_outputs)
```

[8]: LinearRegression()

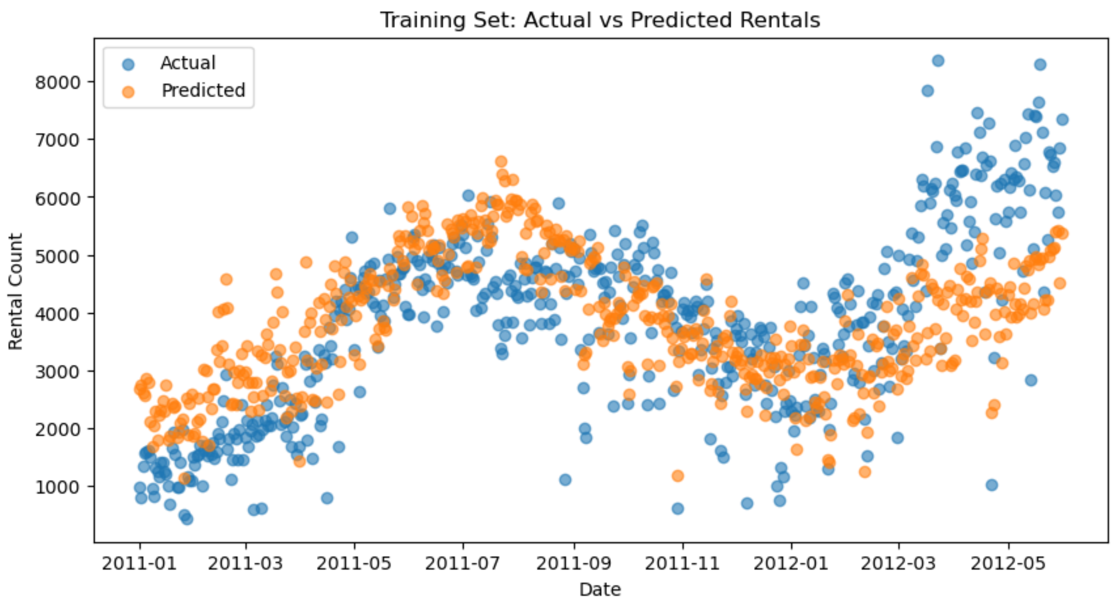
0.6.1 Model Predictions: Actual vs Predicted

We compare the model's predictions to the actual rental counts for both the training and validation periods. This helps us visually inspect how well the model learned and generalized.

[9]:
```
plt.scatter(validation_set['date'], validation_set['cnt'], label="Actual",␣
 ↪alpha=0.6)
plt.scatter(validation_set['date'], lr.predict(validation_inputs),␣
 ↪label="Predicted", alpha=0.6)
plt.title("Validation Set: Actual vs Predicted Rentals")
plt.xlabel("Date")
plt.ylabel("Rental Count")
plt.legend()
plt.show()
```



[10]:
```
plt.scatter(training_set['date'], training_set['cnt'], label="Actual", alpha=0.
↪6)
plt.scatter(training_set['date'], lr.predict(training_inputs),␣
↪label="Predicted", alpha=0.6)
plt.title("Training Set: Actual vs Predicted Rentals")
plt.xlabel("Date")
plt.ylabel("Rental Count")
plt.legend()
plt.show()
```



0.6.2 Model Evaluation: RMSE

To evaluate the model, we compute the Root Mean Squared Error (RMSE) between actual and predicted rental counts. Lower RMSE values indicate more accurate predictions — it's a measure of the average prediction error in real-world units.

[11]:
```
from sklearn.metrics import mean_squared_error

# Predict on the validation set
predictions = lr.predict(validation_inputs)

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(validation_outputs, predictions))
print(f"Validation RMSE: {rmse:.2f}")
```

```
Validation RMSE: 2186.29
```

0.7 Feature Engineering: Adding `last_week`

To account for weekly seasonality, we introduce a new feature called `last_week`, which represents the average number of rentals from exactly 7 days prior. This lag-based feature helps the model learn from recent trends in usage.

[12]:
```
# Create a rolling-style lag feature: average of same day last week
df['last_week'] = (df['cnt'].cumsum() - df['cnt'].cumsum().shift(7)) / 7

# Drop rows with missing values from shifting
df = df.dropna()
```

[13]:
```
df = df.sort_values('date')
```

0.8 Enhanced Linear Regression with `last_week`

Now that we've introduced the `last_week` feature, we'll include it — along with weather and calendar variables — in a new regression model. Our goal is to measure how much this feature improves prediction accuracy compared to earlier models.

[14]:
```
# Split the updated dataset again
training_set = df[df['date'] < '2012-06-01']
validation_set = df[df['date'] >= '2012-06-01']

# Define updated feature set
features = ['atemp', 'workingday', 'hum', 'weathersit', 'last_week',
 ↪'windspeed']

# Extract inputs and outputs
training_inputs = training_set[features].values
training_outputs = training_set[['cnt']].values

validation_inputs = validation_set[features].values
validation_outputs = validation_set[['cnt']].values

# Train a new linear regression model
lr = LinearRegression()
lr.fit(training_inputs, training_outputs)
```

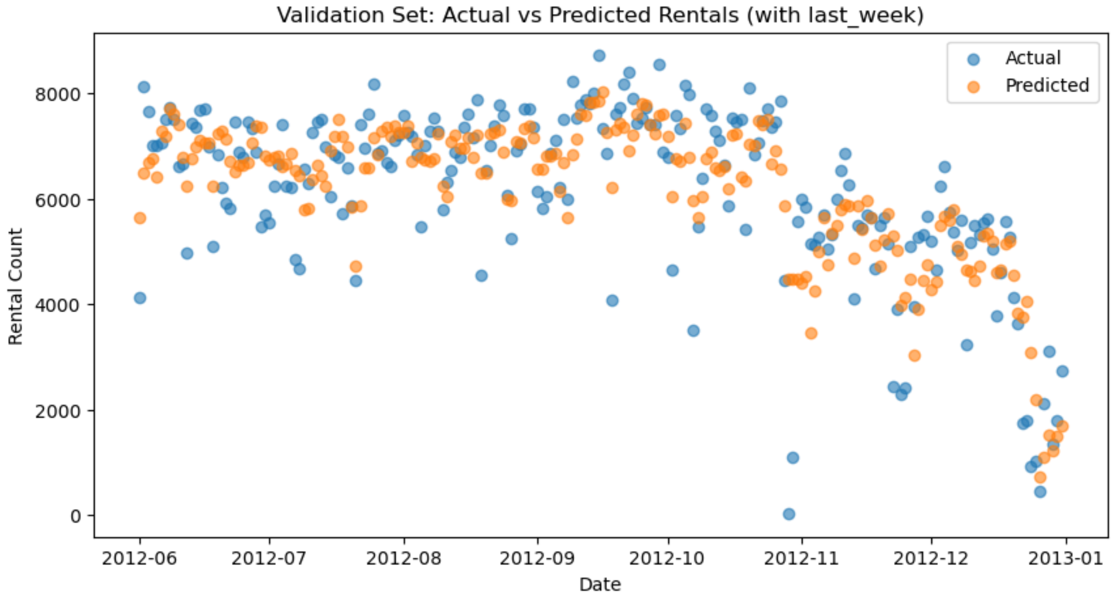
[14]:
```
LinearRegression()
```

0.8.1 Enhanced Model: Predictions and Evaluation

With our newly engineered `last_week` feature, we now assess how the model's predictions compare to actual rental counts. This step shows whether temporal trends have improved forecasting accuracy.

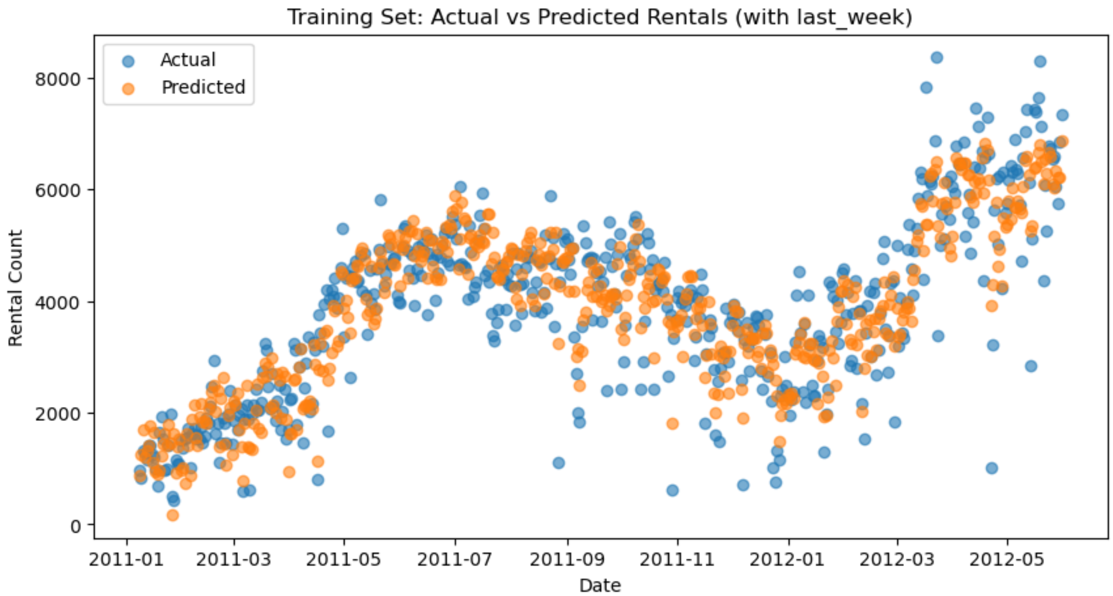
[15]:

```
plt.scatter(validation_set['date'], validation_set['cnt'], label="Actual",␣
↪alpha=0.6)
plt.scatter(validation_set['date'], lr.predict(validation_inputs),␣
↪label="Predicted", alpha=0.6)
plt.title("Validation Set: Actual vs Predicted Rentals (with last_week)")
plt.xlabel("Date")
plt.ylabel("Rental Count")
plt.legend()
plt.show()
```



[16]:

```
plt.scatter(training_set['date'], training_set['cnt'], label="Actual", alpha=0.
↪6)
plt.scatter(training_set['date'], lr.predict(training_inputs),␣
↪label="Predicted", alpha=0.6)
plt.title("Training Set: Actual vs Predicted Rentals (with last_week)")
plt.xlabel("Date")
plt.ylabel("Rental Count")
plt.legend()
plt.show()
```



0.8.2 Final Evaluation: RMSE with `last_week`

We now compute the Root Mean Squared Error (RMSE) for the improved model. Comparing this score with the baseline helps us quantify how much the additional `last_week` feature has boosted our prediction accuracy.

[17]:
```
# Calculate RMSE for enhanced model
from sklearn.metrics import mean_squared_error

final_predictions = lr.predict(validation_inputs)
final_rmse = np.sqrt(mean_squared_error(validation_outputs, final_predictions))
print(f"Final Validation RMSE with `last_week`: {final_rmse:.2f}")
```

```
Final Validation RMSE with `last_week`: 925.66
```

0.9 Conclusion

In this project, we successfully built and refined a linear regression model to predict daily bicycle rentals based on weather and calendar data.

0.9.1 Key Takeaways:

- **Apparent temperature (`atemp`)** showed a strong positive correlation with rental count.
- Initial models using basic features achieved an RMSE of approximately **2186**.
- Introducing the `last_week` feature — a temporal lag variable — reduced the RMSE to ~**926**, a substantial accuracy improvement.
- Visual analysis confirmed the enhanced model closely tracked rental demand trends in both the training and validation periods.

This project demonstrates how thoughtful feature engineering and time-aware modeling can significantly boost performance in real-world forecasting problems.

```
[ ]:
```