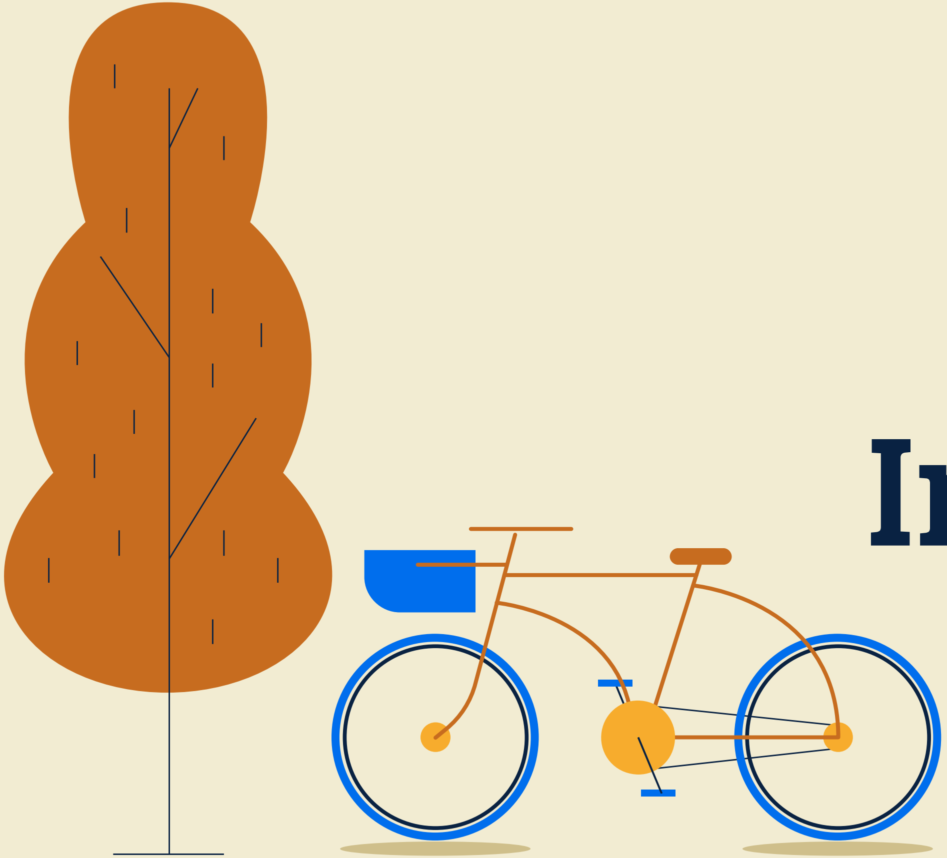
Bike Sharing

Wajahat Khan
Harpreet Gujral
Bryan Aguirre
Alan Senoff
Jaeik Lee
Zachary Feldmar

Table of contents



1

Introduction

The Problem

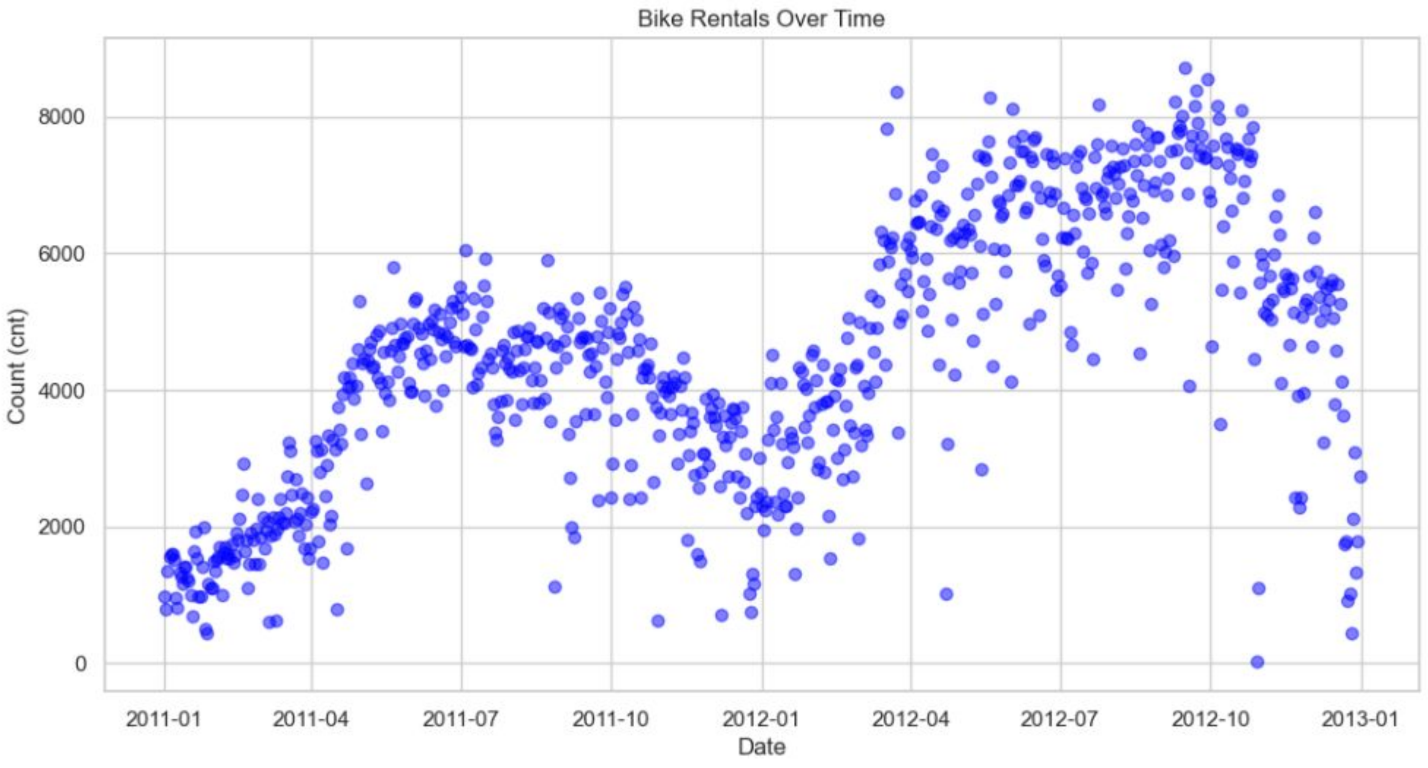
Stakeholders

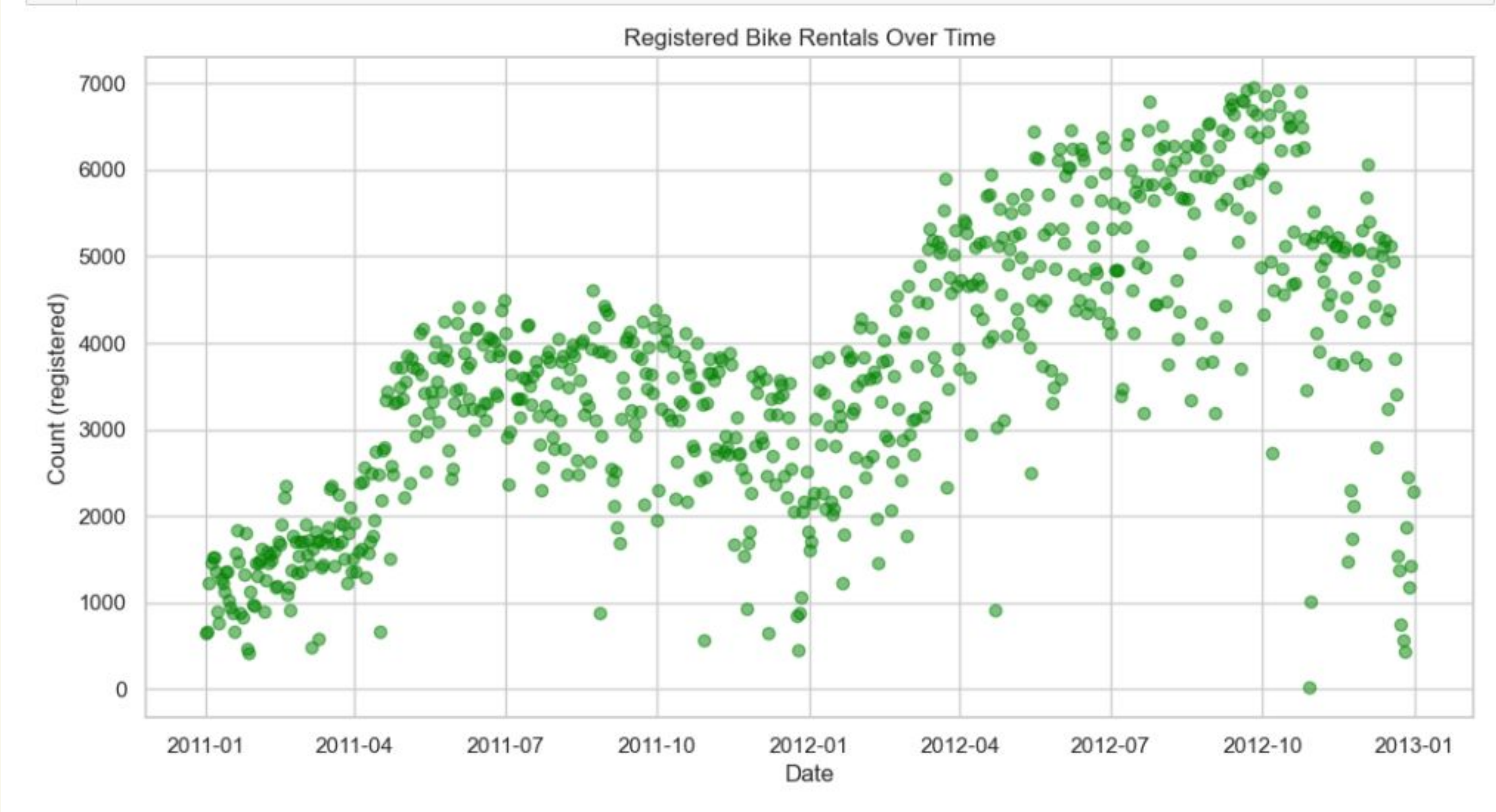
2 Dataset

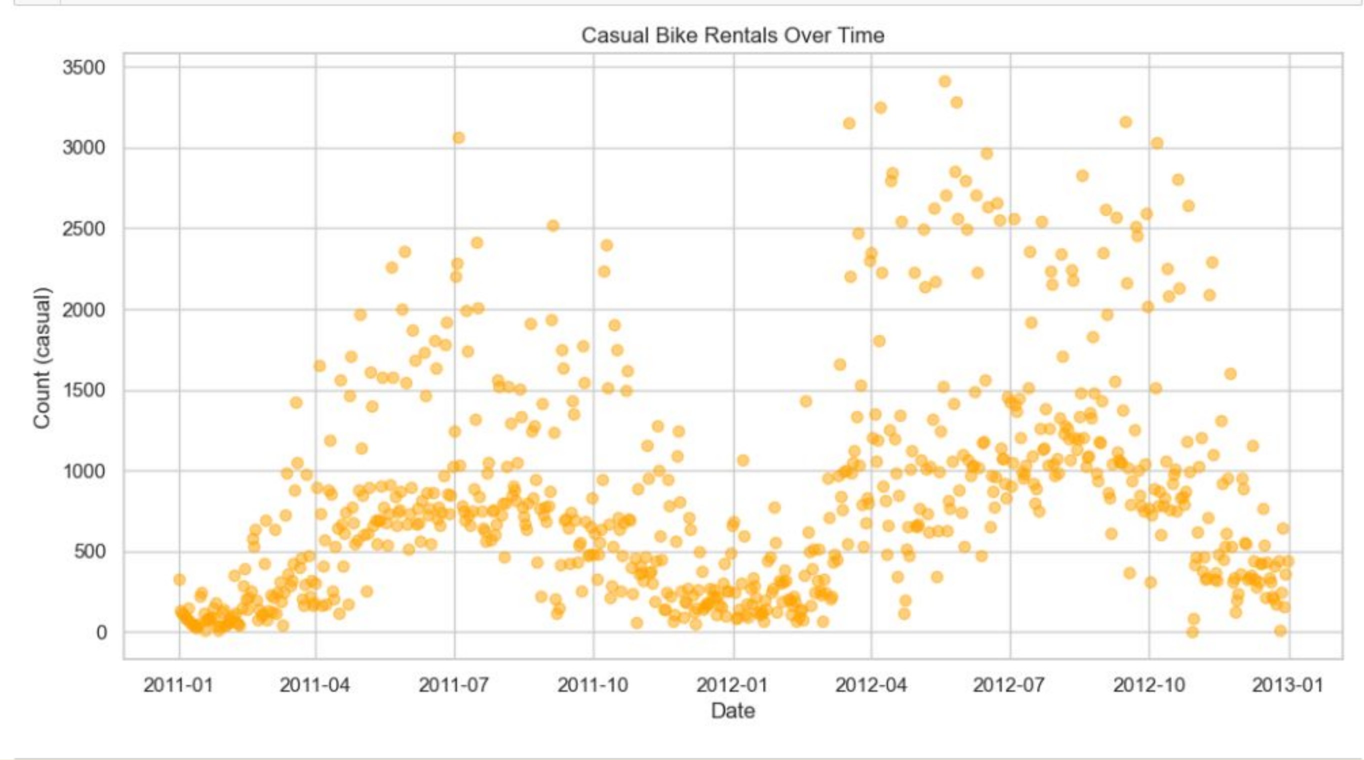
	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
726	727	2012-12-27	1	1	12	0	4	1	2	0.254167	0.226642	0.652917	0.350133	247	1867	2114
727	728	2012-12-28	1	1	12	0	5	1	2	0.253333	0.255046	0.590000	0.155471	644	2451	3095
728	729	2012-12-29	1	1	12	0	6	0	2	0.253333	0.242400	0.752917	0.124383	159	1182	1341
729	730	2012-12-30	1	1	12	0	0	0	1	0.255833	0.231700	0.483333	0.350754	364	1432	1796
730	731	2012-12-31	1	1	12	0	1	1	2	0.215833	0.223487	0.577500	0.154846	439	2290	2729

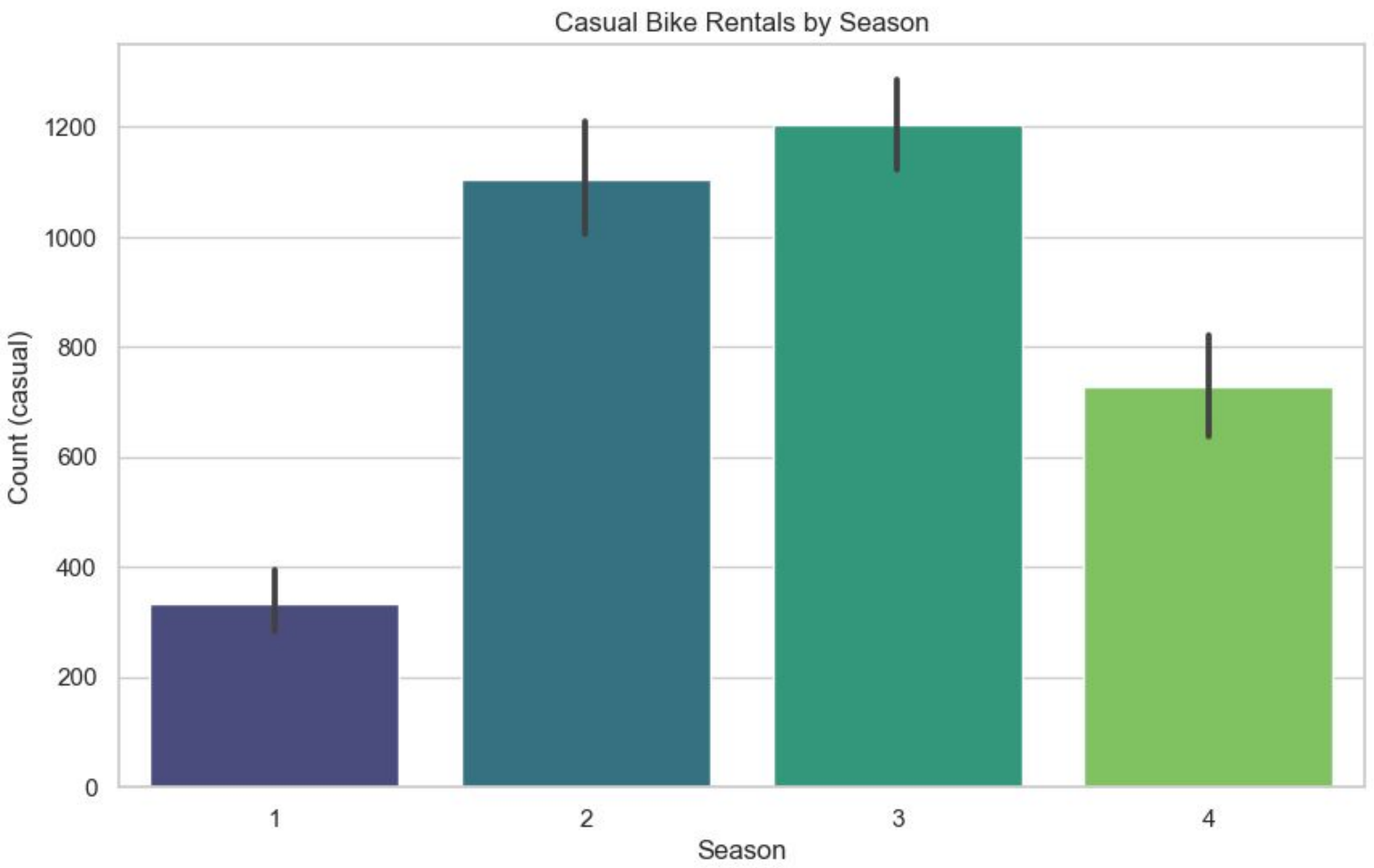
731 rows × 16 columns

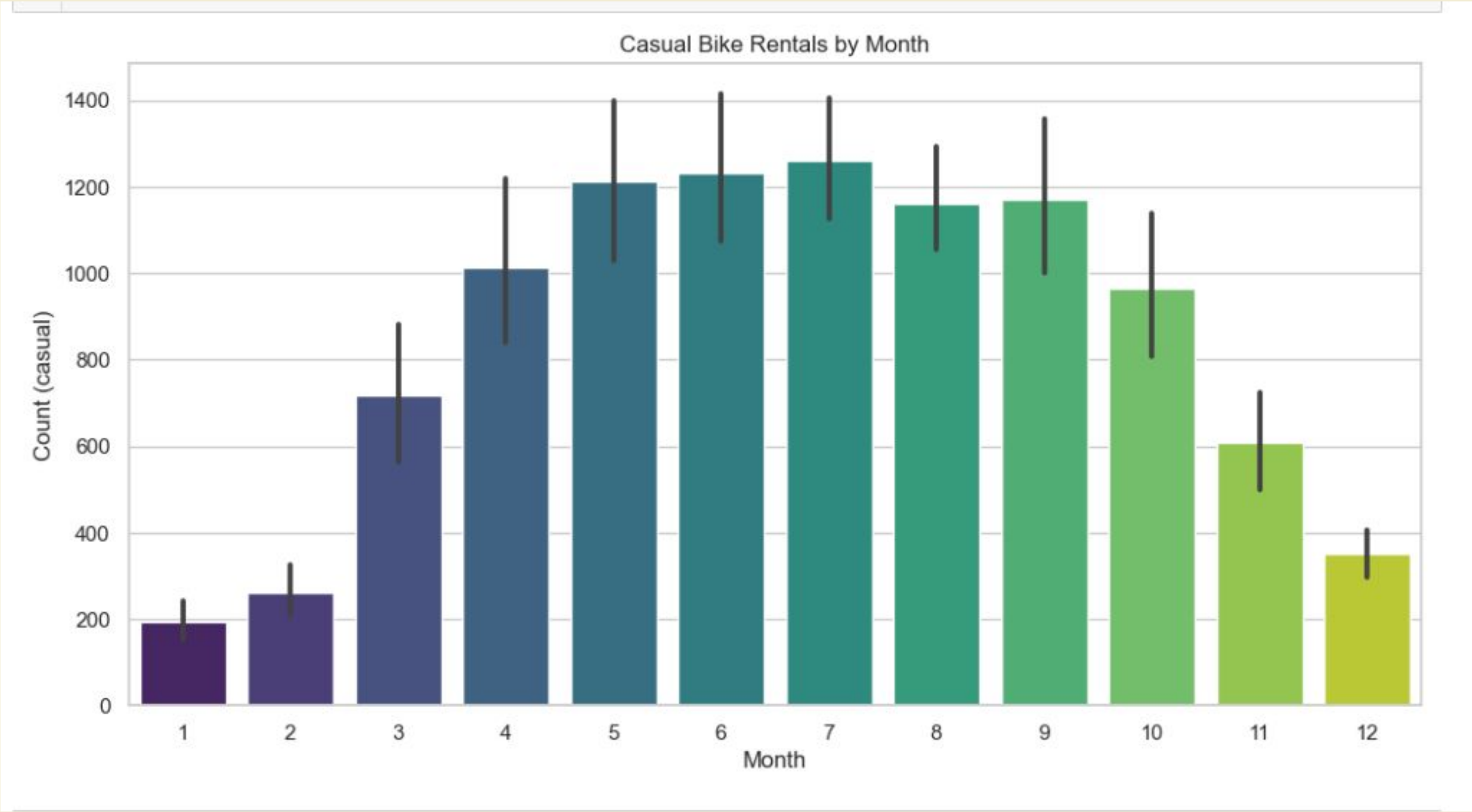
```python
import matplotlib.pyplot as plt
import pandas as pd

plt.figure(figsize=(12, 6))
day_Bike['date'] = pd.to_datetime(day_Bike[['year', 'month', 'day']])
plt.scatter(day_Bike['date'], day_Bike['cnt'], color='blue', alpha=0.5)
plt.title('Bike Rentals Over Time')
plt.xlabel('Date')
plt.ylabel('Count (cnt)')
plt.grid(True)
plt.show()
```

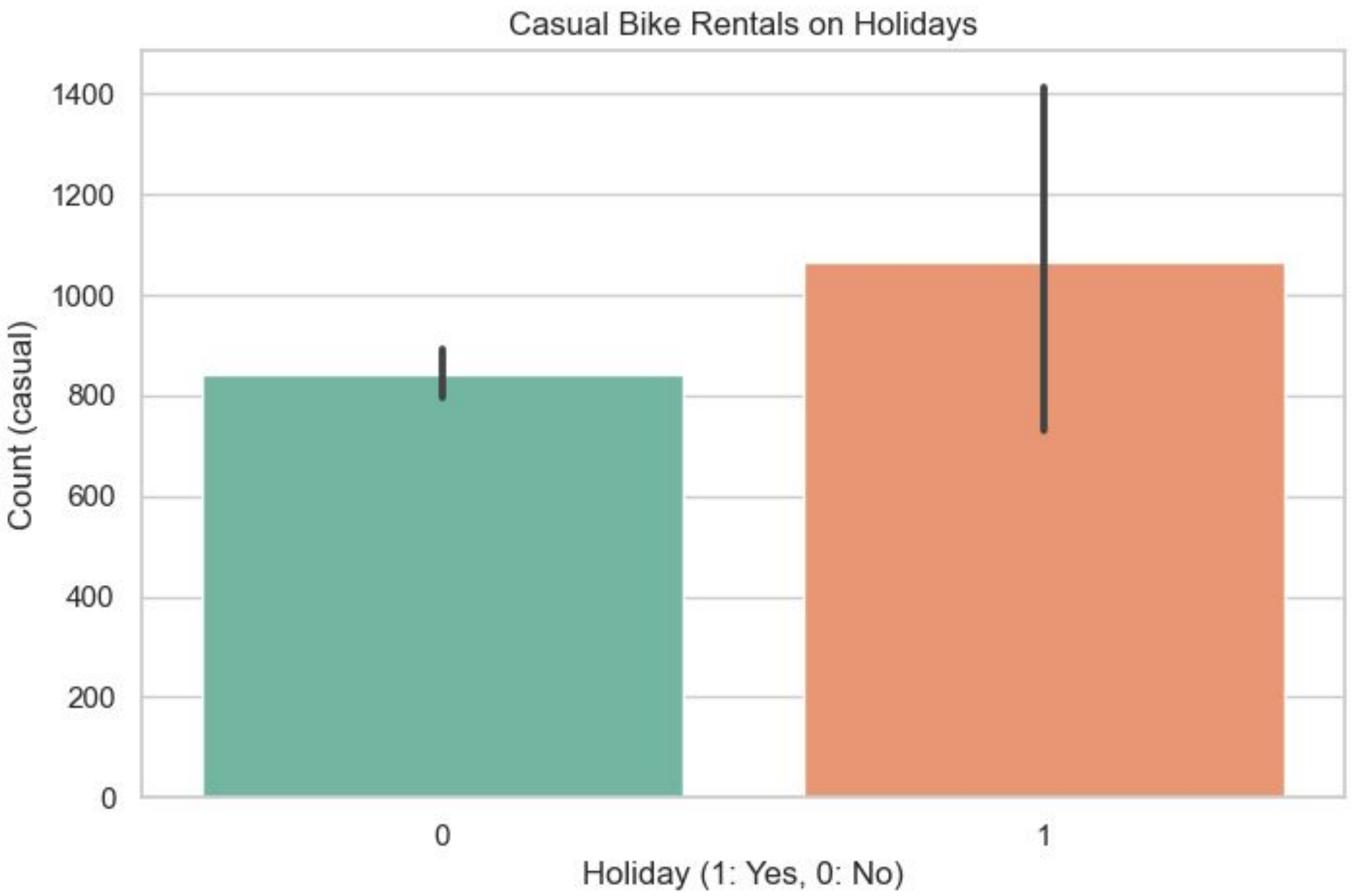



Registered Bike Rentals Over Time
Count (registered)
7000
6000
5000
4000
3000
2000
1000
0
2011-01
2011-04
2011-07
2011-10
2012-01
2012-04
2012-07
2012-10
2013-01
Date


Casual Bike Rentals Over Time
Count (casual)
3500
3000
2500
2000
1500
1000
500
0
2011-01
2011-04
2011-07
2011-10
2012-01
2012-04
2012-07
2012-10
2013-01
Date


Casual Bike Rentals by Season
Count (casual)
0
200
400
600
800
1000
1200
1
2
3
4
Season


Casual Bike Rentals by Month
Count (casual)
0
200
400
600
800
1000
1200
1400
1
2
3
4
5
6
7
8
9
10
11
12
Month


Casual Bike Rentals on Holidays
1400
1200
1000
800
600
400
200
0
Count (casual)
0
1
Holiday (1: Yes, 0: No)

3

Methods

Models Used

Linear Regression

LSTM

Arima

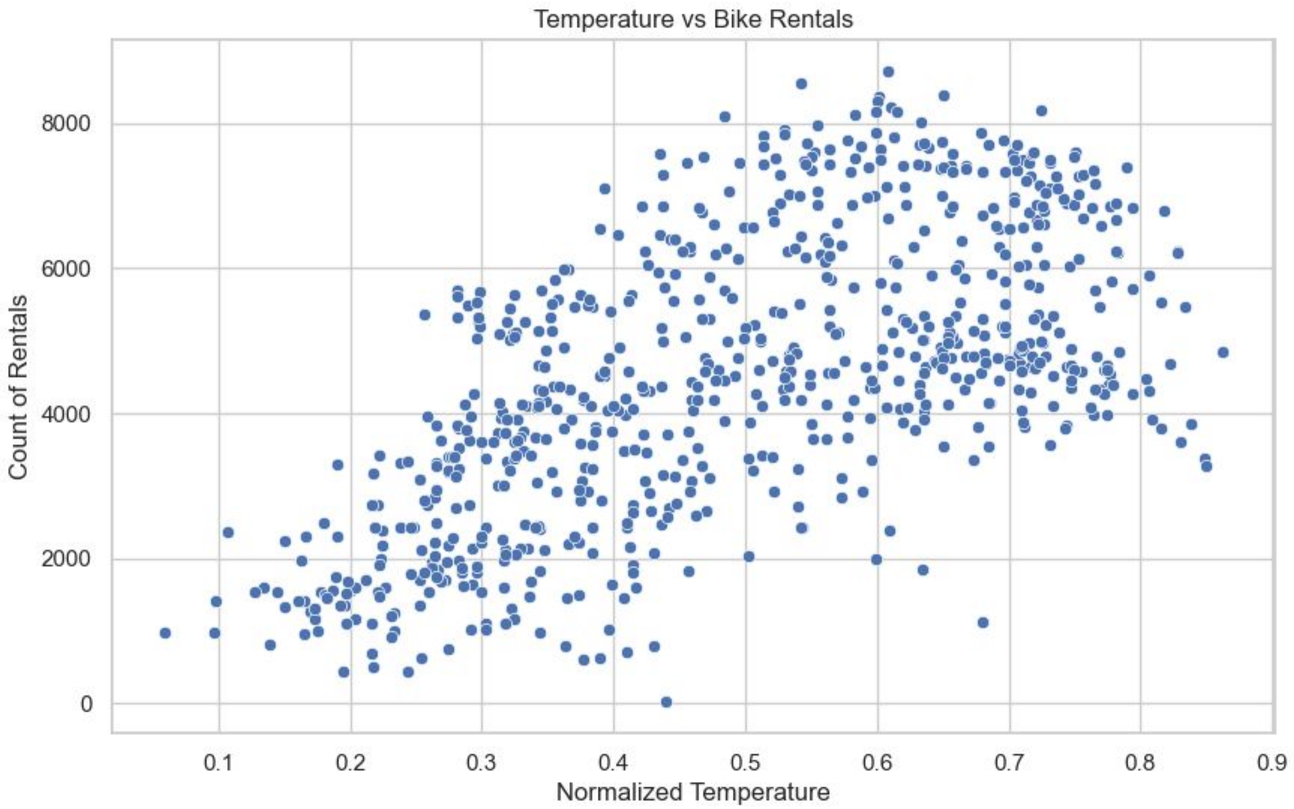
Random Forest Classifier

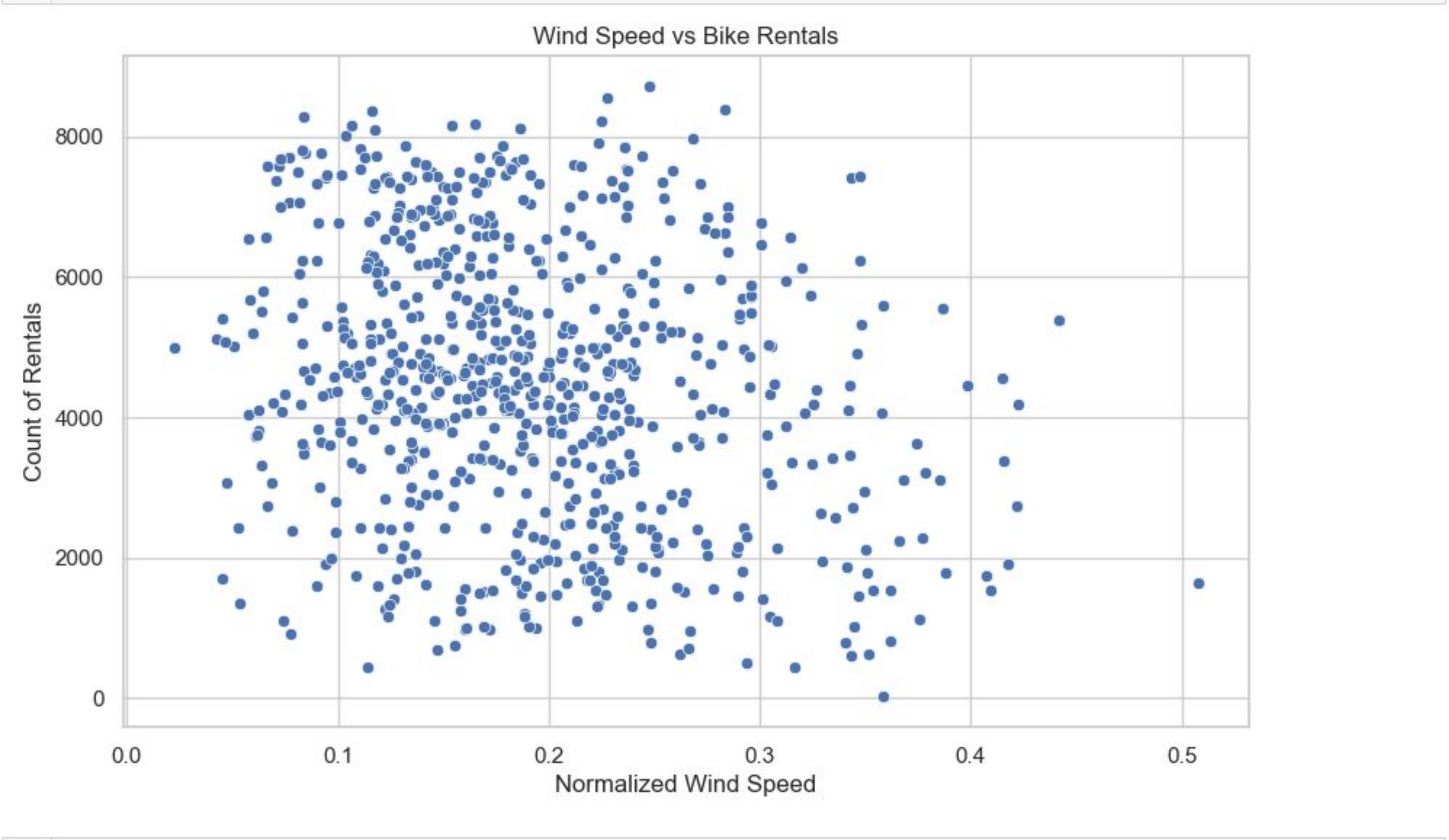
Simple Linear Regression

- Equation:
  - $Y = \beta0 + \beta1X + \varepsilon$
  - Y: Dependent variable
  - $\beta0$: Y-intercept
  - $\beta1$: Coefficient of independent variable X
  - ε: Error term
- Key Metrics:
  - R-Squared (R^2):
    - Measures the proportion/coorelation of variance in the dependent variable that can be explained by the independent variable. Ranges from 0 to 1, with higher values indicating a better fit.
  - Mean Squared Error (MSE):
    - Represents the average of the squares of the errors (difference between observed and predicted values). Lower MSE indicates a better fit of the model.
  - P-Value:
    - Tests the significance of the regression coefficient ($\beta1$). A low p-value (< 0.05) suggests that the independent variable has a statistically significant impact on the dependent variable.

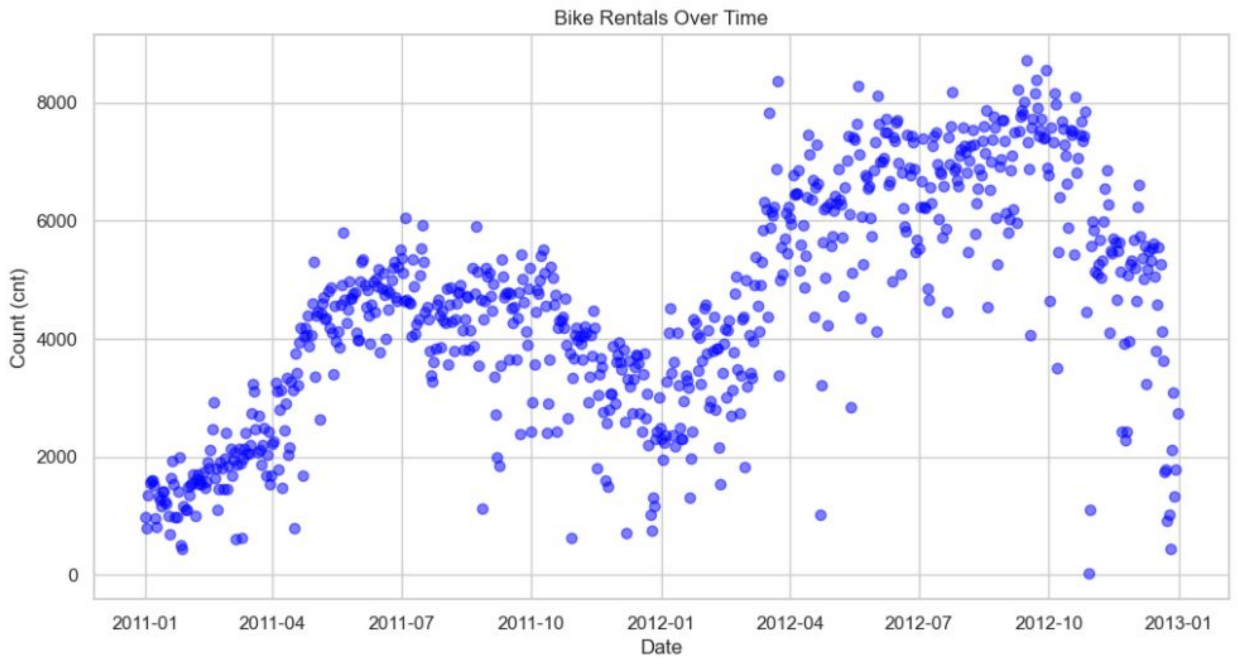
In [9]:

```
plot_relationship('temp', 'cnt', title='Temperature vs Bike Rentals',
                  xlabel='Normalized Temperature', ylabel='Count of Rentals')
```



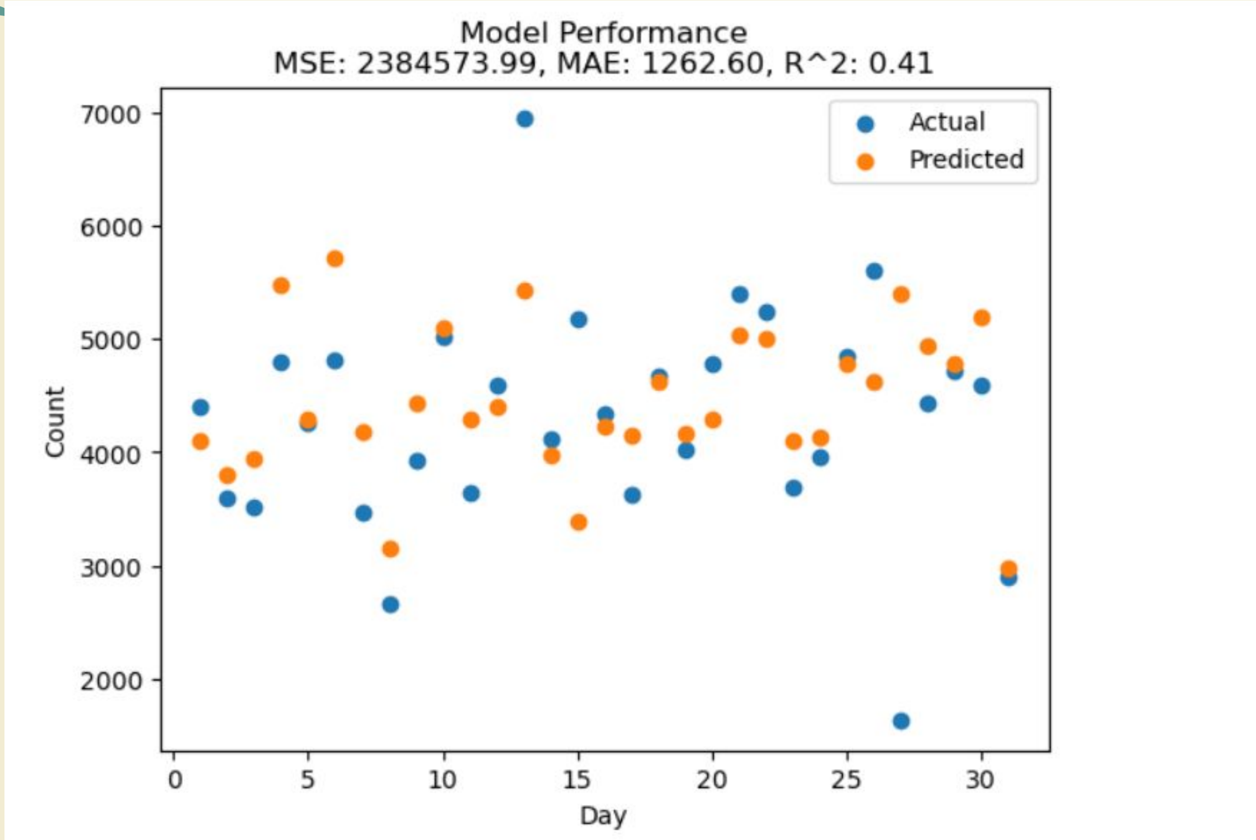

Wind Speed vs Bike Rentals
Count of Rentals
Normalized Wind Speed
0
2000
4000
6000
8000
0.0
0.1
0.2
0.3
0.4
0.5

```python
import matplotlib.pyplot as plt
import pandas as pd

plt.figure(figsize=(12, 6))
day_Bike['date'] = pd.to_datetime(day_Bike[['year', 'month', 'day']])
plt.scatter(day_Bike['date'], day_Bike['cnt'], color='blue', alpha=0.5)
plt.title('Bike Rentals Over Time')
plt.xlabel('Date')
plt.ylabel('Count (cnt)')
plt.grid(True)
plt.show()
```

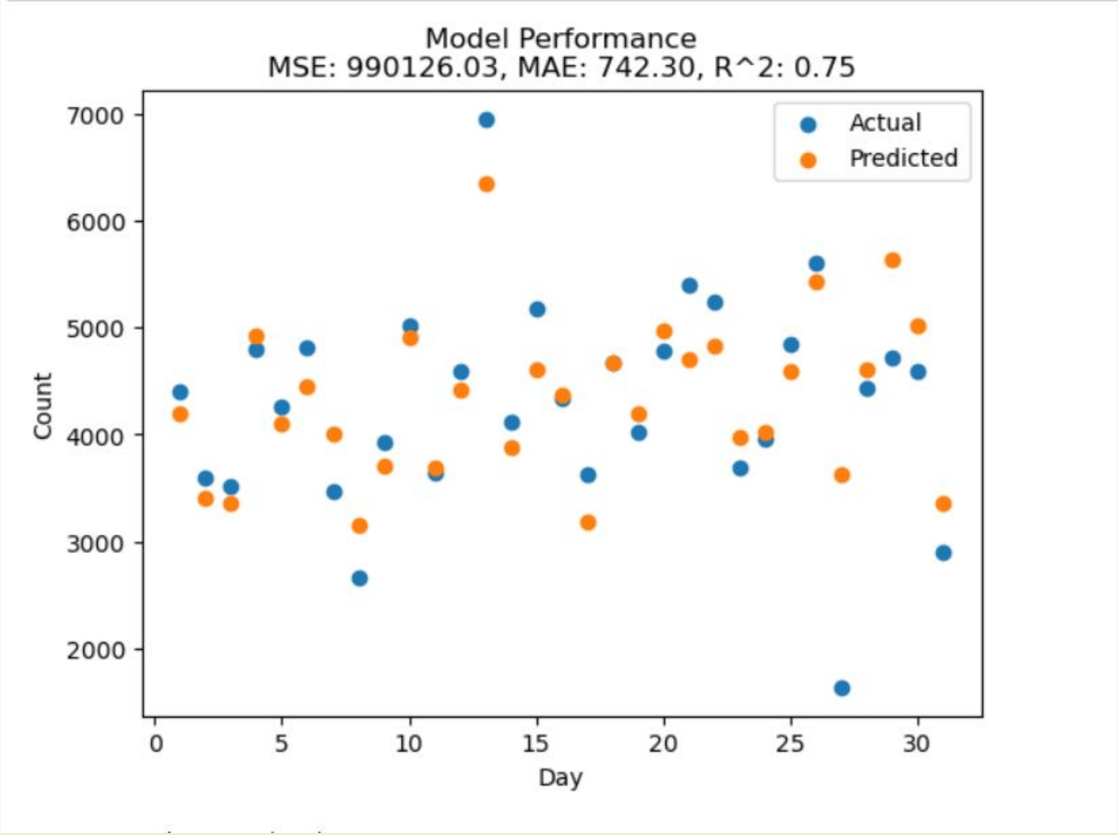


- Shows how in worse weather conditions there is a clear decline in bike rentals over time.

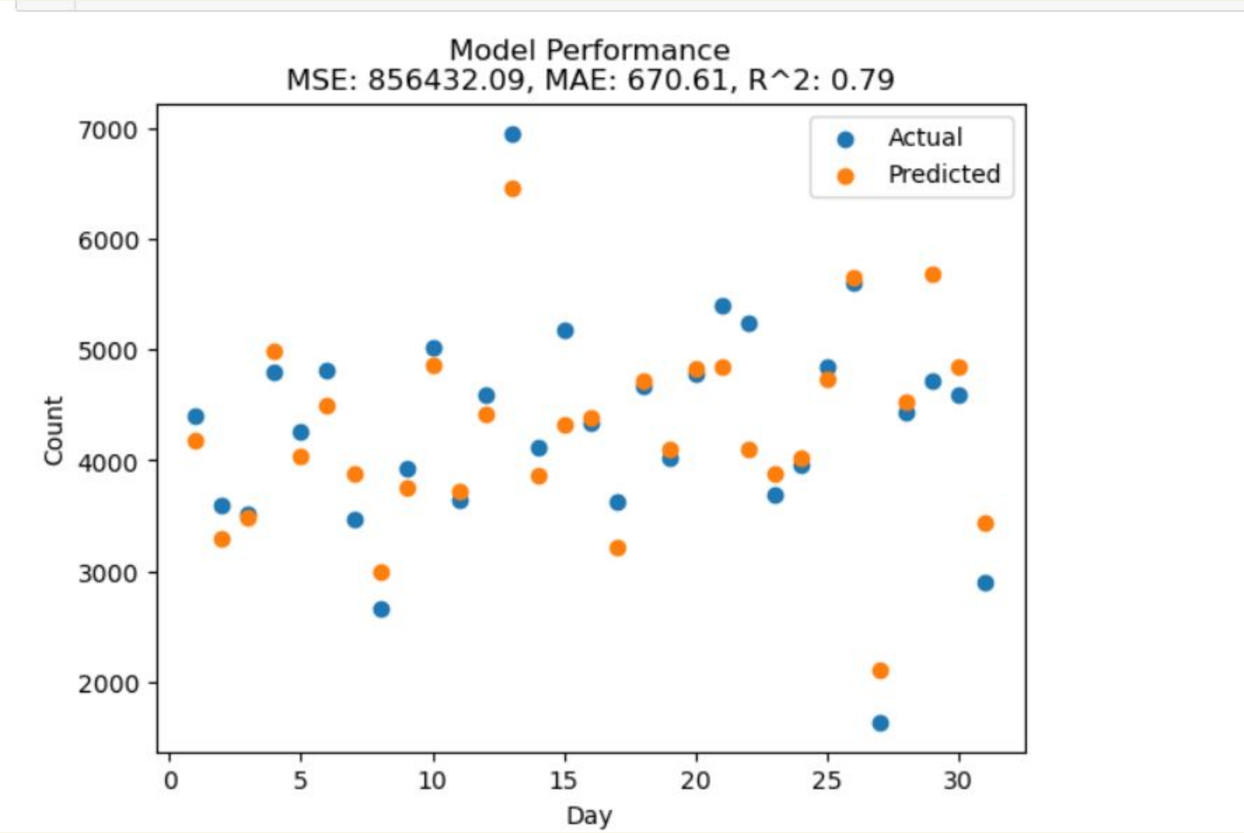
Day_Bike[['day', 'month', 'year']]



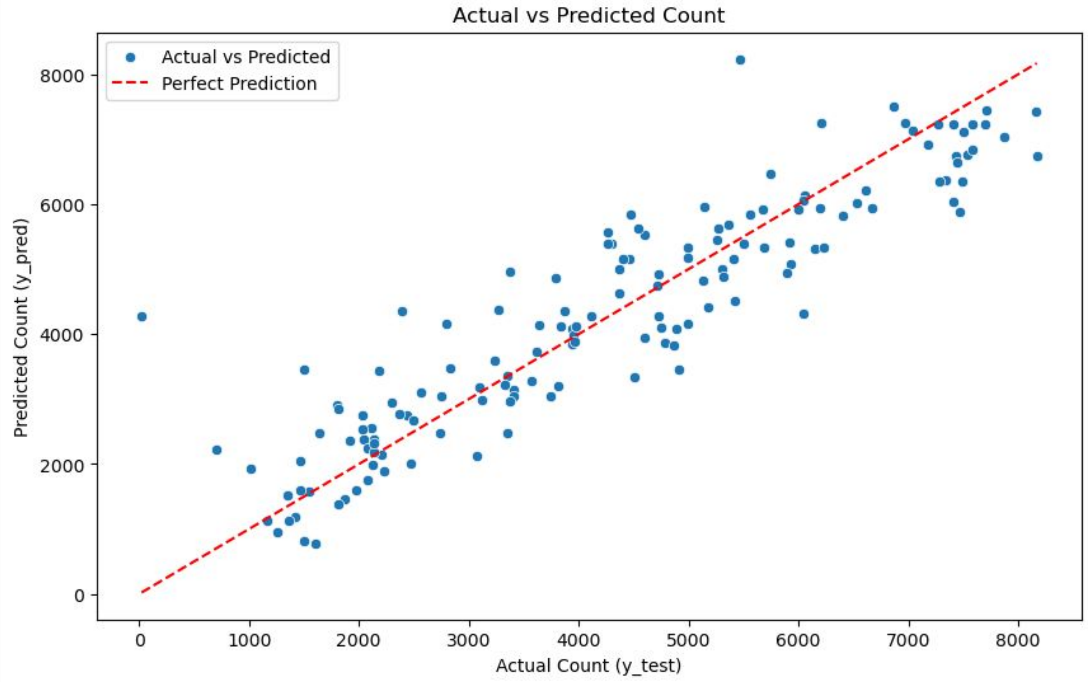
selected_features = ['day', 'month', 'year', 'temp', 'hum']



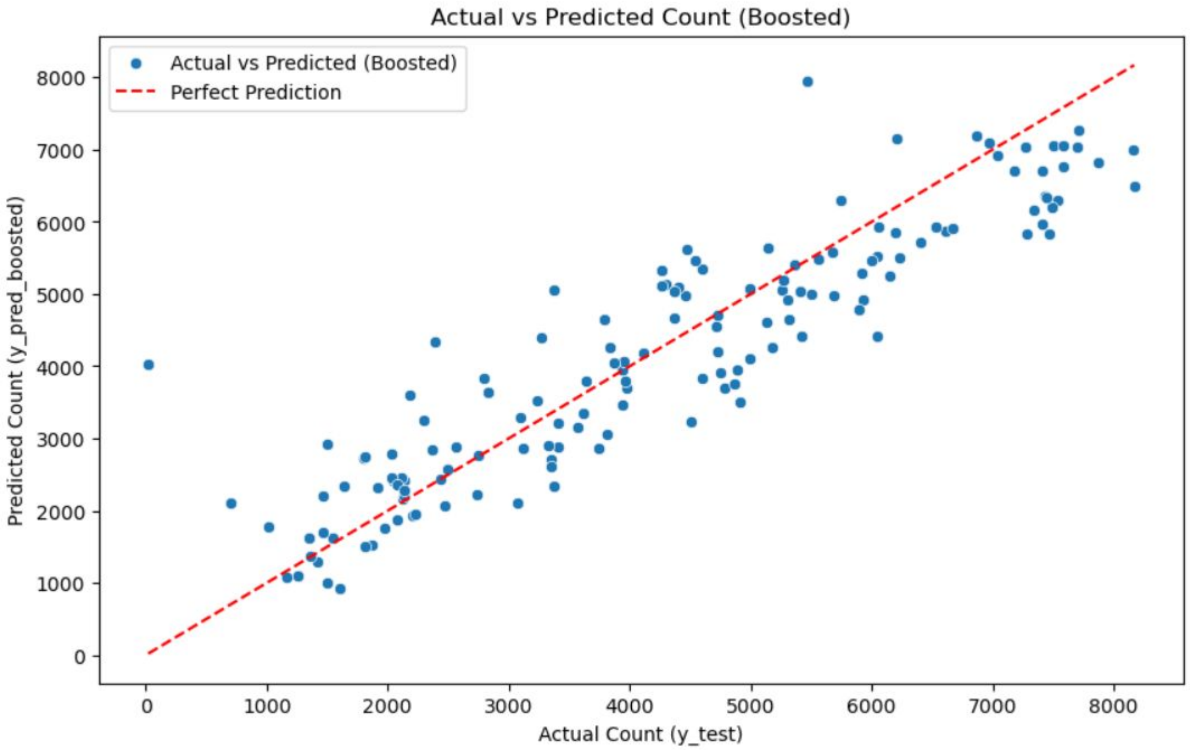
selected_features = ['day', 'month', 'year', 'temp', 'hum',"season"]



selected_features = ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed']

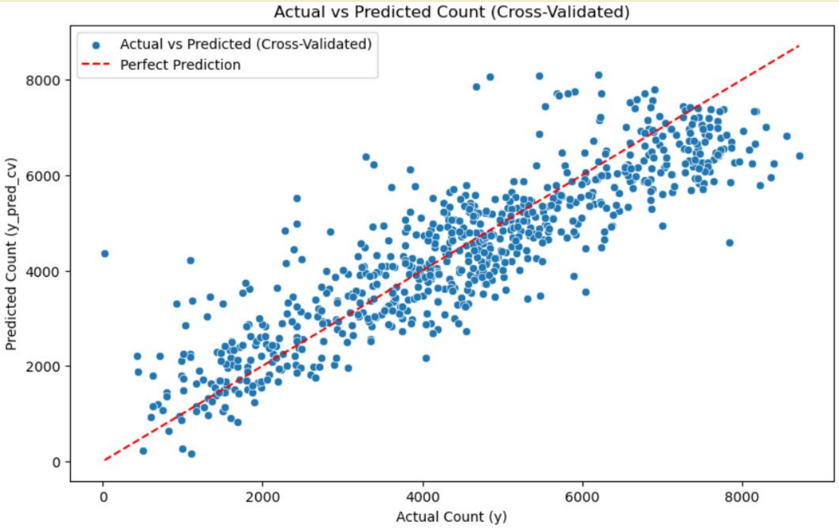



```
Mean Squared Error (MSE): 691035.01
Mean Absolute Error (MAE): 617.39
R-squared (R^2): 0.83
```

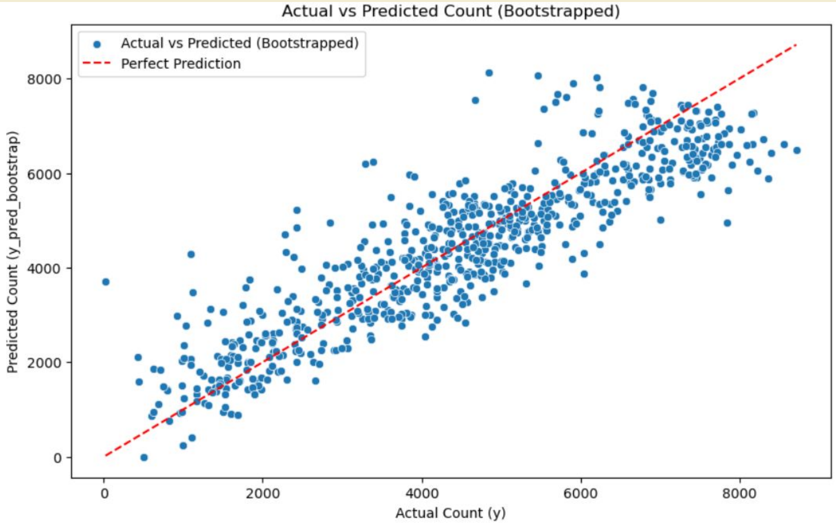



```
Metrics for Boosted Model:
Mean Squared Error (MSE): 709242.80
Mean Absolute Error (MAE): 649.23
R-squared (R^2): 0.82
```

- Boosting actually led to a decrease in R-Squared




```
Metrics for Cross-Validated Boosted Model:
Mean Squared Error (MSE): 832991.32
Mean Absolute Error (MAE): 694.58
R-squared (R^2): 0.78
```




```
Metrics for Bootstrapped Boosted Model:
Mean Squared Error (MSE): 766825.70
Mean Absolute Error (MAE): 671.29
R-squared (R^2): 0.80
```

LSTM

Long Short-Term Memory Neural Network

captures long-term dependencies and relationships in information over time.

Simple Terms:
imagine it as a smart system that can learn from past experiences and use that knowledge to make predictions or decisions about what might happen next.

A type of Recurrent Neural Network (RNN) that is specifically designed to handle sequential data, such as time series

Processing, predicting, and classifying on the basis of time-series data.

Benefits of LSTM

- Robust to Noisy Data
- Feature Extraction:
- Handling Long-Term Dependencies:
- Memory Cell
- Non-Linearity
- Better Performance

Models for Prediction

I. Season/Weather
II. Days of the Week, Season/Weather/Temperature

Features for Training

```
# Load the data
day_Bike = pd.read_csv(r'C:\Users\jackf\OneDrive\Desktop\Bike_Sharing\UCI_Bike_Sharing\data\day.csv')

# Extract features and target
features = day_Bike[['season',  'weathersit',
                     'temp', 'atemp', 'hum', 'windspeed']]


target = day_Bike['cnt']
```

Model 1
Features

```python
# Extract features and target
features = day_Bike[['season',  'weathersit',
                     'temp', 'atemp', 'hum', 'windspeed']]


target = day_Bike['cnt']

# Normalize features
scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target.values, test_size=0.2, random_state=42)

# Reshape data for LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'))
model.add(Dropout(0.2))  # Adding dropout for regularization
model.add(Dense(1))

# Using the Adam optimizer with a lower learning rate
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mean_squared_error')

# Adding early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), callbacks=[early_stopping])

# Evaluate the model
mse = model.evaluate(X_test, y_test)
print(f'Mean Squared Error on Test Data: {mse}')

# Make predictions
predictions = model.predict(X_test)

# Extract values from predictions
predicted_values = np.squeeze(predictions)

# Now 'predicted_values' contains the predicted values from your LSTM model
print(predicted_values)
```

- Load and Process Clean Data
- Split Data
- Reshape Data for LSTM
- Build the LSTM Model
- Train the Model
- Evaluate the Model

```python
# Assuming you have new data in the same format as your training data
# Replace 'new_data' with your actual new data
new_data = pd.DataFrame({
    'season': [1],  'weathersit': [1], 'temp': [0.215833], 'atemp': [0.223487],
    'hum': [0.577500], 'windspeed': [0.154846]
})

# Normalize new features using the same scaler
new_features_scaled = scaler.transform(new_data)

# Reshape data for LSTM
new_features_scaled = new_features_scaled.reshape((new_features_scaled.shape[0], 1, new_features_scaled.shape[1]))

# Use the trained model to make predictions
new_predictions = model.predict(new_features_scaled)

# Extract values from predictions
predicted_cnt_values = np.squeeze(new_predictions)

# Now 'predicted_cnt_values' contains the predicted "cnt" values for the new data
print(predicted_cnt_values)
```

```
1/1 [==============================] - 0s 14ms/step
1090.5117
```

- Create a new data set
- Normalized new features
- Reshaped for LSTM
- Print Predictions

Model 2

Model 2 Features

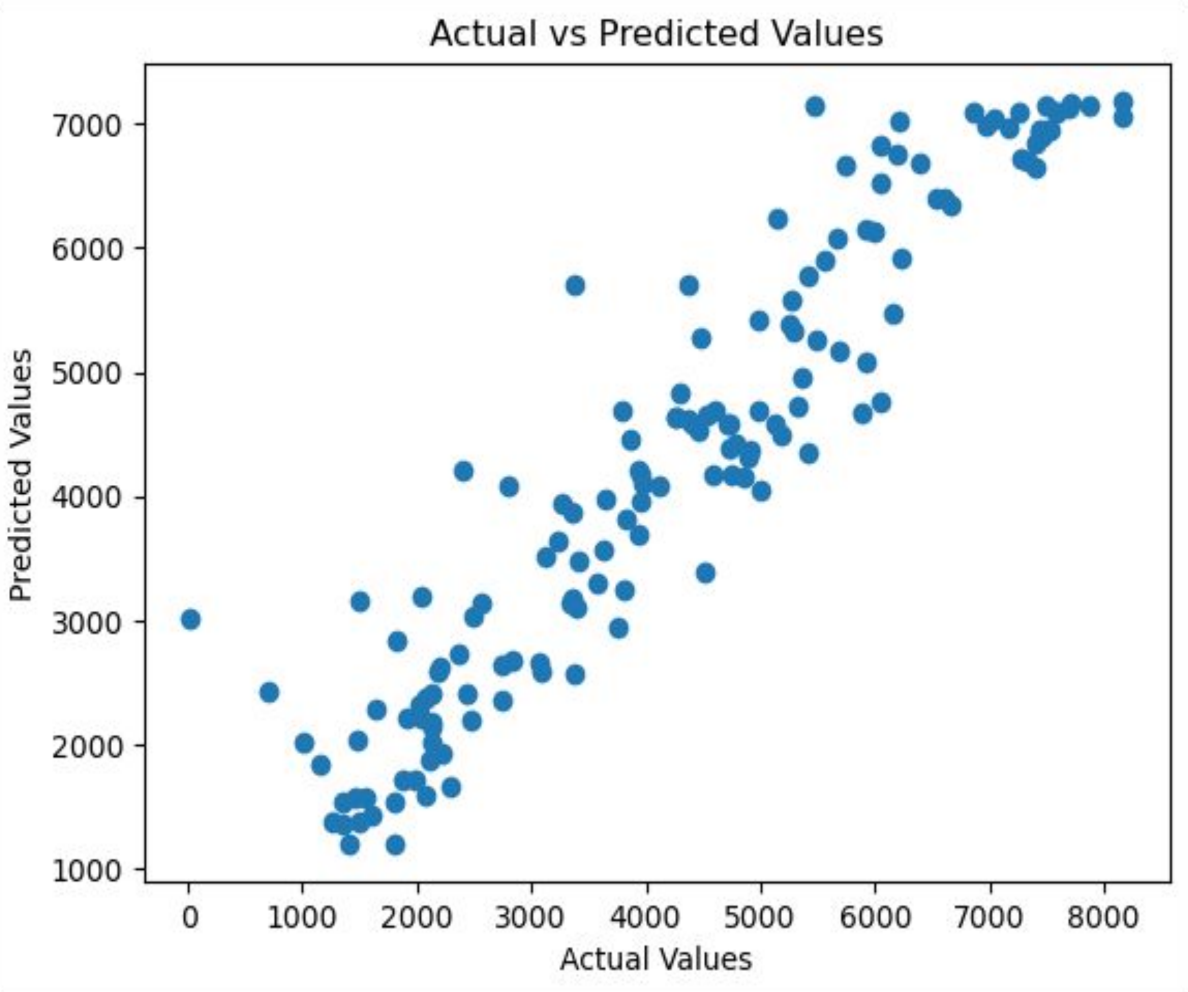
```
# Load the data
day_Bike = pd.read_csv(r'C:\Users\jackf\OneDrive\Desktop\Bike_Sharing\UCI_Bike_Sharing\data\day.csv')  # Replace 'your_data

# Extract features and target
features = day_Bike[['season', 'yr', 'mnth', 'holiday',
                     'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed']]
target = day_Bike['cnt']
```

```
target = day_Bike['cnt']

# Normalize features
scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_scaled, target.values, test_size=0.2, random_state=42)

# Reshape data for LSTM
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Build the LSTM model
model = Sequential()
model.add(LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2]), activation='relu'))
model.add(Dropout(0.2))  # Adding dropout for regularization
model.add(Dense(1))

# Using the Adam optimizer with a lower learning rate
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mean_squared_error')

# Adding early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), callbacks=[early_stopping])

# Evaluate the model
mse = model.evaluate(X_test, y_test)
print(f'Mean Squared Error on Test Data: {mse}')

# Make predictions
predictions = model.predict(X_test)

# Extract values from predictions
predicted_values = np.squeeze(predictions)

# Now 'predicted_values' contains the predicted values from your LSTM model
print(predicted_values)
```

Same methodology and architecture as the first one only adding more variety in our features

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
726	727	2012-12-27	1	1	12	0	4	1	2	0.254167	0.226642	0.652917	0.350133	247	1867	2114
727	728	2012-12-28	1	1	12	0	5	1	2	0.253333	0.255046	0.590000	0.155471	644	2451	3095
728	729	2012-12-29	1	1	12	0	6	0	2	0.253333	0.242400	0.752917	0.124383	159	1182	1341
729	730	2012-12-30	1	1	12	0	0	0	1	0.255833	0.231700	0.483333	0.350754	364	1432	1796
730	731	2012-12-31	1	1	12	0	1	1	2	0.215833	0.223487	0.577500	0.154846	439	2290	2729

731 rows × 16 columns



Actual vs Predicted Values
Predicted Values
Actual Values
1000
2000
3000
4000
5000
6000
7000
0
1000
2000
3000
4000
5000
6000
7000
8000

```
# We want the value near 2729

# both 2626.3992
# just weather 1090.8662
```

Overall Target Goal was to reach a prediction of 2729 avg

Model 1 Performance: 1090.8862

Model 2 Performance: 2626.3992 ✓

Arima

Auto Regressive Integrated Moving Average (ARIMA)

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA

# Assuming your date_Bike dataframe has a datetime index
day_Bike['dteday'] = pd.to_datetime(day_Bike['dteday'])
day_Bike.set_index('dteday', inplace=True)

# Replace 'your_column' with the column you want to predict (e.g., 'cnt')
y = day_Bike['cnt']

# Fit ARIMA model
arima_model = ARIMA(y, order = (5, 1, 0))  # Example order, you may need to tune this
arima_result = arima_model.fit()

# Forecast future values
future_steps = 30  # Adjust as needed
forecast = arima_result.get_forecast(steps = future_steps)

# Print the forecasted values
(forecast.predicted_mean)
```

The ARIMA model is powerful and widely used for time series forecasting.

but its effectiveness depends on the characteristics of the data being analyzed.

In practice, the selection of the appropriate values for p, d, and q requires careful analysis and often involves experimentation.

```python
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA

# Assuming your date_Bike dataframe has a datetime index
day_Bike['dteday'] = pd.to_datetime(day_Bike['dteday'])
day_Bike.set_index('dteday', inplace=True)

# Replace 'your_column' with the column you want to predict (e.g., 'cnt')
y = day_Bike['cnt']

# Fit ARIMA model
arima_model = ARIMA(y, order=(5, 1, 0))  # Example order, you may need to tune this
arima_result = arima_model.fit()

# Forecast future values for the year 2100
future_steps = 365 * (2100 - 2013)  # Assuming a daily frequency
future_dates = pd.date_range(start='2013-01-01', periods=future_steps)
forecast = arima_result.get_forecast(steps=future_steps)

# Add forecasted values to a new DataFrame
forecast_df = pd.DataFrame(index=future_dates)
forecast_df['predicted_mean'] = forecast.predicted_mean

# Access the forecasted value for January 5, 2100
specific_date = pd.to_datetime('2099-01-05')
forecast_value = forecast_df.loc[specific_date, 'predicted_mean']

# Print or use the forecasted value
print(f'Predicted value for 2100-01-05: {forecast_value:.2f}')
```

```
C:\Users\jackf\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was p
rovided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
C:\Users\jackf\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was p
rovided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
C:\Users\jackf\anaconda3\Lib\site-packages\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency information was p
rovided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
Predicted value for 2100-01-05: 2171.05
```

Fitting ARIMA model and Forecasting (First Block):
Fitting ARIMA model(Second Block)

- Fitted an ARIMA model to your bike-sharing data using the order (5, 1, 0).
- Forecasted future values for each day from 2013 to 2100 (with a daily frequency).
- Created a DataFrame (`forecast_df`) to store the forecasted mean values.

Accessing Specific Forecasted Value (Third Block):

- Accessed and printed the forecasted mean value for January 5, 2100.

Random Forest

- Random Forest is a method where multiple decision trees (like a group of experts) each make a prediction based on different parts of the data, and their combined decisions lead to a more accurate and reliable final result.
- For instance, in a medical diagnosis application of Random Forest, multiple decision trees are generated, each considering different patient features like age, symptoms, and medical history, and the ensemble of trees combines their outputs to provide a more accurate prediction of a patient's likelihood of having a particular disease.
- Random Forest method offers high accuracy, reduced overfitting, and robustness, making it suitable for large datasets, providing feature importance insights, and requiring minimal tuning. It excels in classification and regression tasks, is parallelizable, interpretable, and effective with imbalanced data.

```python
# Drop non-numeric columns before splitting

from sklearn.ensemble import RandomForestRegressor
X = day_Bike.drop(['cnt', 'casual', 'registered', "dteday"], axis=1)
y = day_Bike['cnt']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)

# Create a Random Forest Regressor model
rf_model = RandomForestRegressor(random_state = 42)

# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Calculate R-squared on the test set
r_squared_rf = r2_score(y_test, y_pred_rf)
print(f'R-squared on the test set (Random Forest): {r_squared_rf}')
```

[6]

```
R-squared on the test set (Random Forest): 0.8716122813755863
```

Boosting with Random Forest

```python
# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.ensemble import GradientBoostingRegressor

# Drop non-numeric columns before splitting
X = day_Bike.drop(['cnt', 'casual', 'registered', "dteday"], axis=1)
y = day_Bike['cnt']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Gradient Boosting Regressor model
gb_model = GradientBoostingRegressor(random_state=42)

# Fit the model to the training data
gb_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_gb = gb_model.predict(X_test)

# Calculate R-squared on the test set
r_squared_gb = r2_score(y_test, y_pred_gb)
print(f'R-squared on the test set (Gradient Boosting): {r_squared_gb}')
```

```
R-squared on the test set (Gradient Boosting): 0.8988810786043309
```

Boosting and Bootstrapping

```
# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.ensemble import GradientBoostingRegressor, BaggingRegressor

# Drop non-numeric columns before splitting
X = day_Bike.drop(['cnt', 'casual', 'registered', "dteday"], axis=1)
y = day_Bike['cnt']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Gradient Boosting Regressor model
base_gb_model = GradientBoostingRegressor(random_state=42)

# Wrap the base model with BaggingRegressor for bootstrapping
gb_model_with_bootstrap = BaggingRegressor(base_gb_model, n_estimators=50, random_state=42)

# Fit the model to the training data
gb_model_with_bootstrap.fit(X_train, y_train)

# Make predictions on the test set
y_pred_gb_bootstrap = gb_model_with_bootstrap.predict(X_test)

# Calculate R-squared on the test set
r_squared_gb_bootstrap = r2_score(y_test, y_pred_gb_bootstrap)
print(f'R-squared on the test set (Gradient Boosting with Bootstrapping): {r_squared_gb_bootstrap}')
```

```
R-squared on the test set (Gradient Boosting with Bootstrapping): 0.900799578231797
```

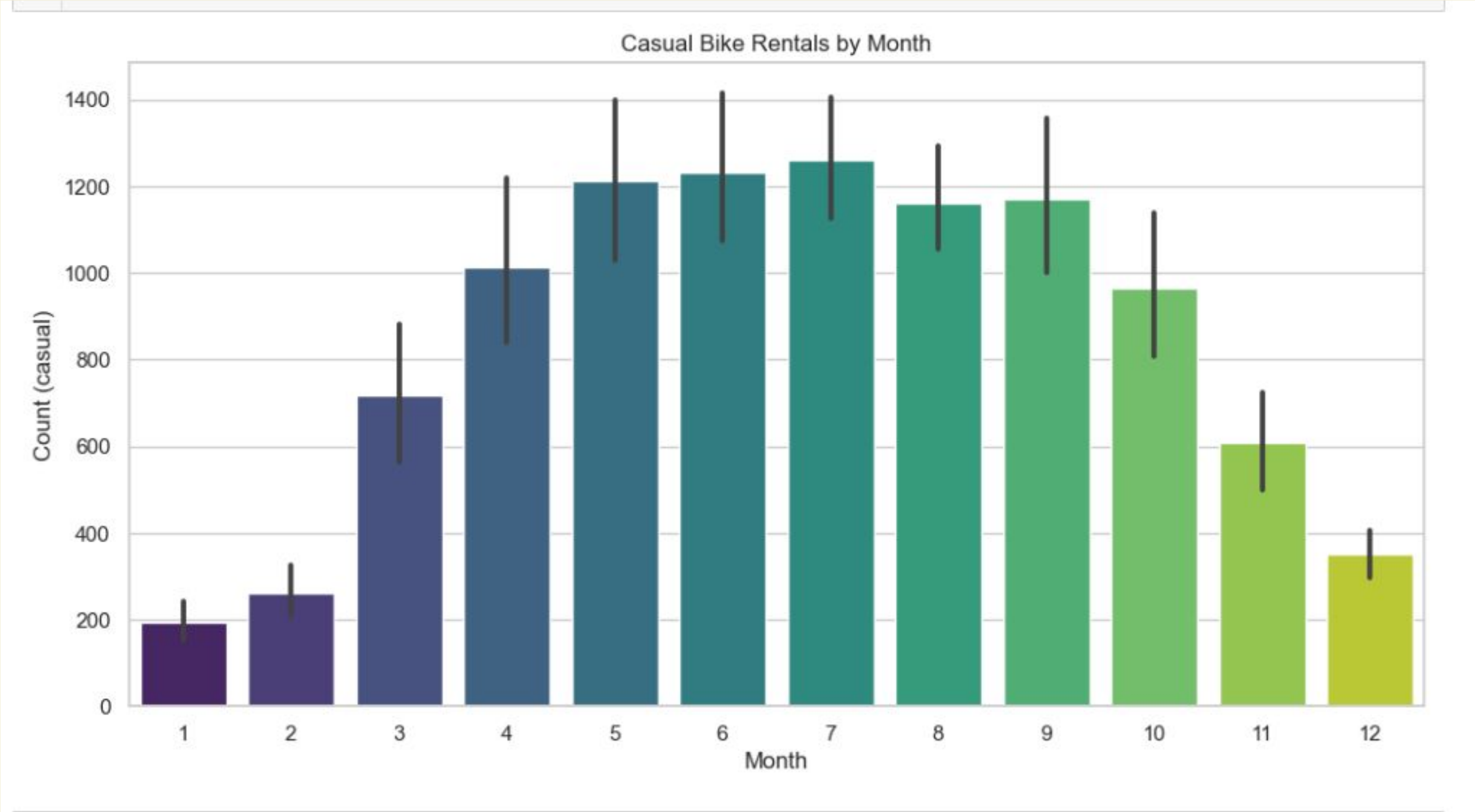
Cross Validation, Boosting, and bootstrapping.

```
# Import necessary modules
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import r2_score
from sklearn.ensemble import GradientBoostingRegressor, BaggingRegressor

# Drop non-numeric columns before splitting
X = day_Bike.drop(['cnt', 'casual', 'registered', "dteday"], axis=1)
y = day_Bike['cnt']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Gradient Boosting Regressor model
base_gb_model = GradientBoostingRegressor(random_state=42)

# Wrap the base model with BaggingRegressor for bootstrapping
gb_model_with_bootstrap = BaggingRegressor(base_gb_model, n_estimators=50, random_state=42)

# Perform cross-validation on the entire dataset
cv_scores = cross_val_score(gb_model_with_bootstrap, X, y, cv=5, scoring='r2')

# Print the cross-validation scores
print(f'Cross-Validation R-squared scores: {cv_scores}')

# Fit the model to the training data
gb_model_with_bootstrap.fit(X_train, y_train)

# Make predictions on the test set
y_pred_gb_bootstrap = gb_model_with_bootstrap.predict(X_test)

# Calculate R-squared on the test set
r_squared_gb_bootstrap = r2_score(y_test, y_pred_gb_bootstrap)
print(f'R-squared on the test set (Gradient Boosting with Bootstrapping): {r_squared_gb_bootstrap}')
```

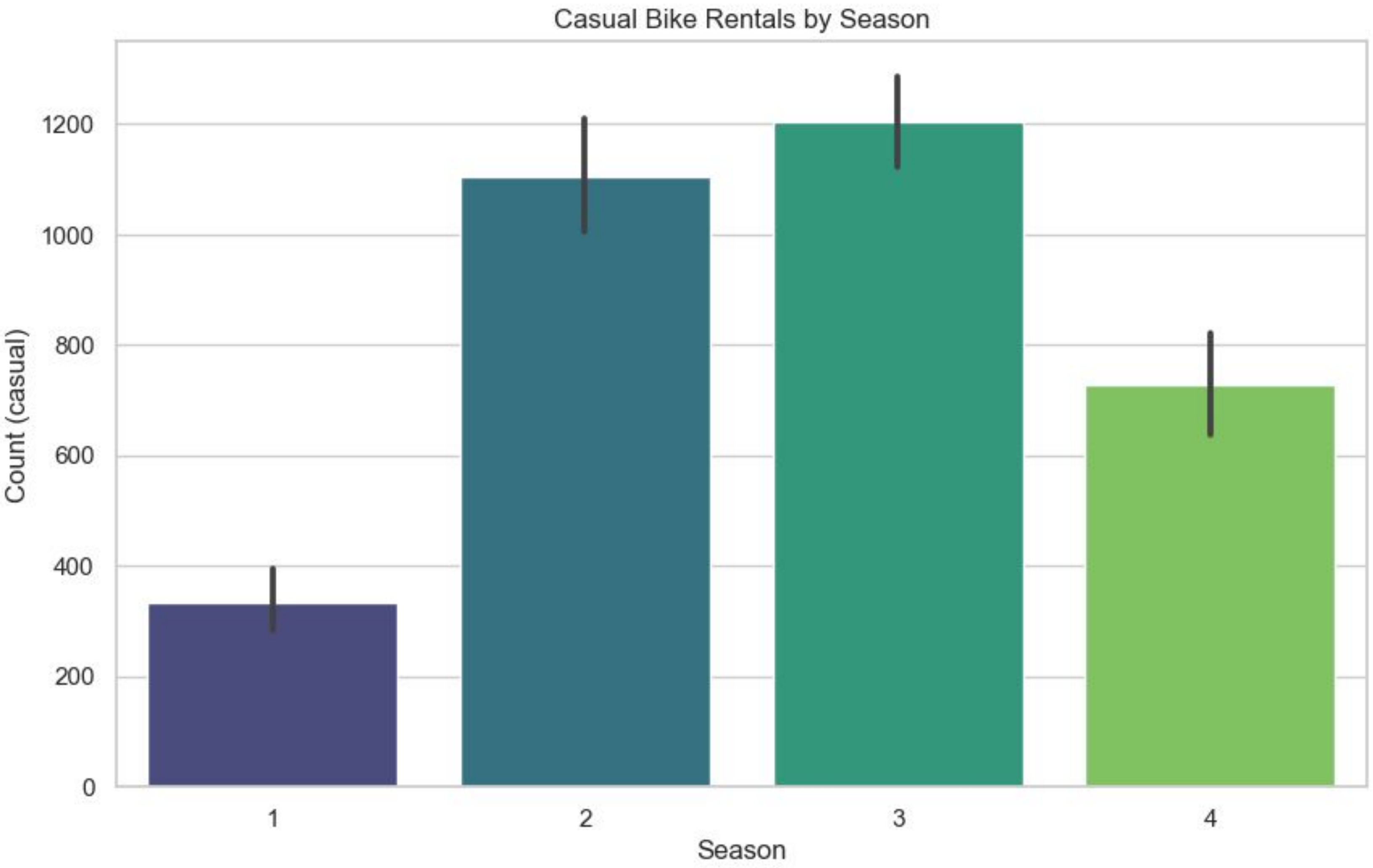
[9]

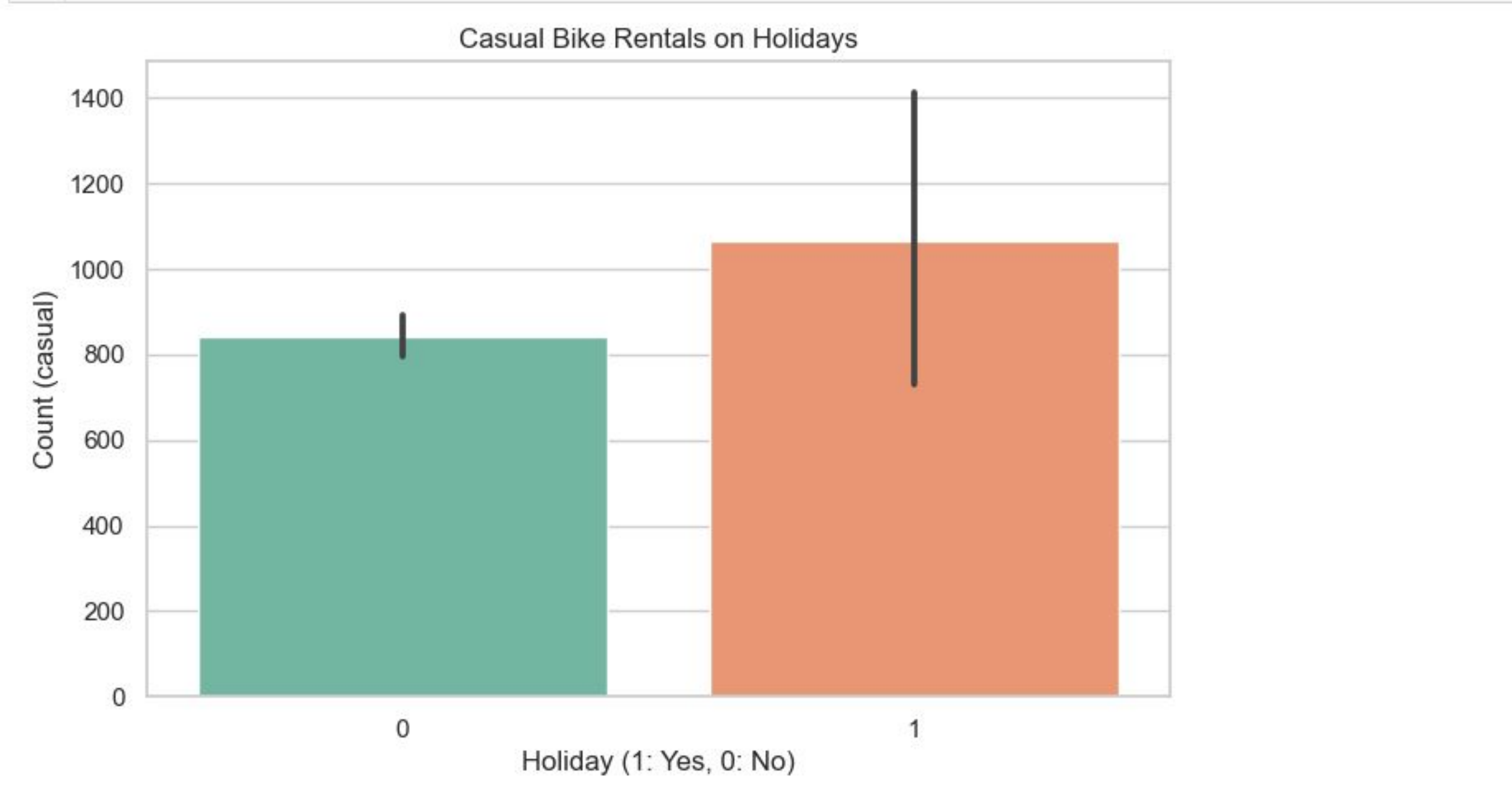
```
Cross-Validation R-squared scores: [-0.04569916  0.54852738  0.20664622  0.52739073  0.69939849]
R-squared on the test set (Gradient Boosting with Bootstrapping): 0.900799578231797
```

Dataset isn't big enough for Cross Validation

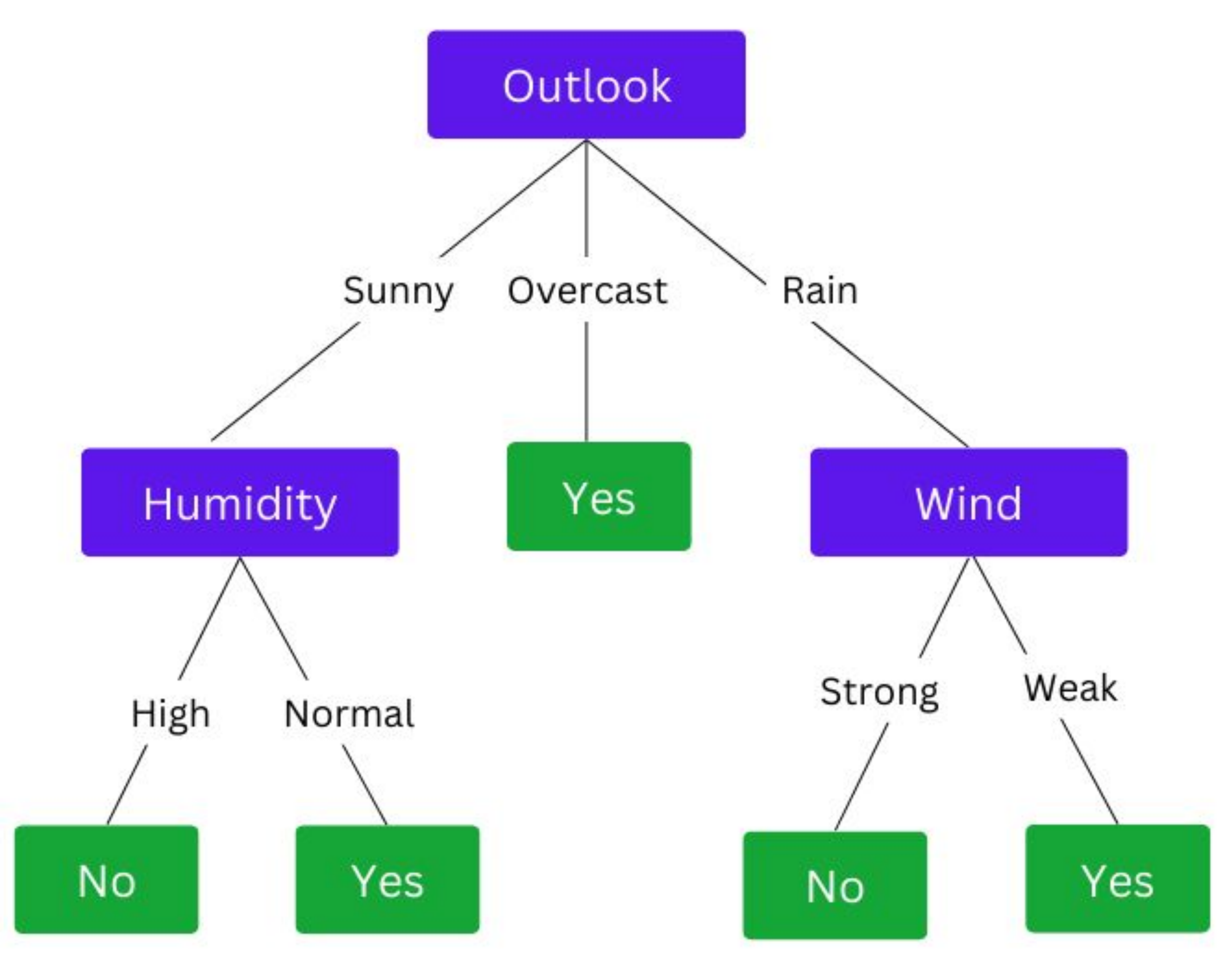


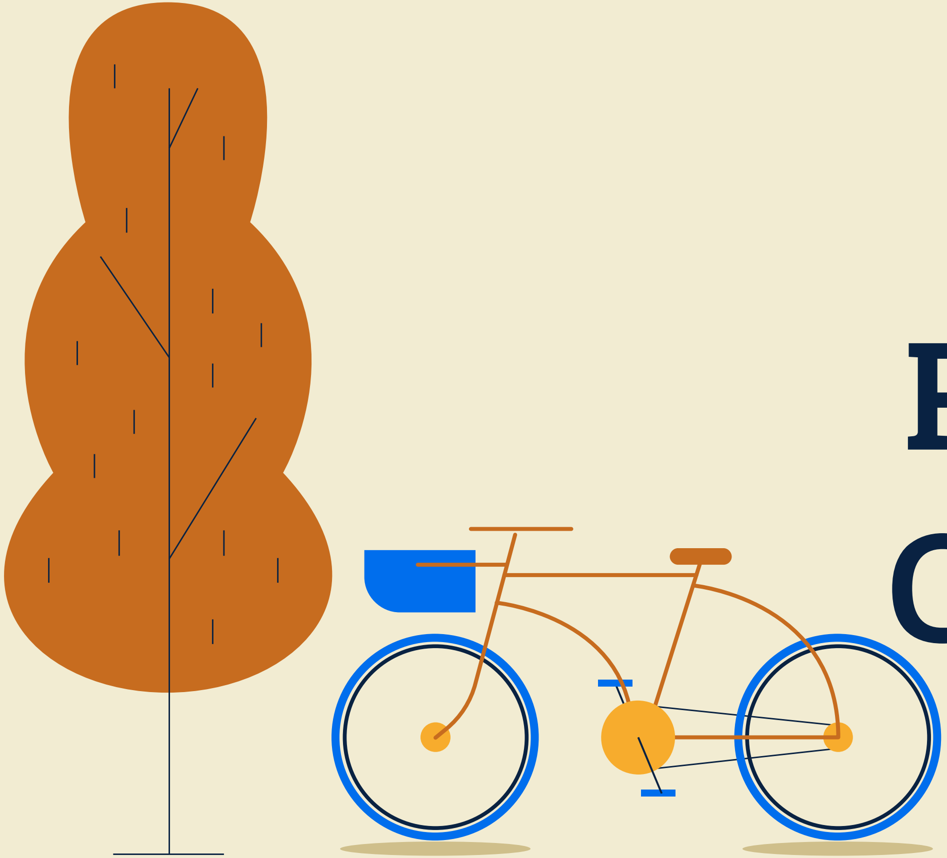
Casual Bike Rentals by Month
Count (casual)
0
200
400
600
800
1000
1200
1400
1
2
3
4
5
6
7
8
9
10
11
12
Month


Casual Bike Rentals by Season
Count (casual)
0
200
400
600
800
1000
1200
1
2
3
4
Season



Casual Bike Rentals on Holidays
1400
1200
1000
800
600
400
200
0
Count (casual)
0
1
Holiday (1: Yes, 0: No)


Outlook
Sunny
Overcast
Rain
Humidity
Yes
Wind
High
Normal
Strong
Weak
No
Yes
No
Yes



4

Results and Conclusions

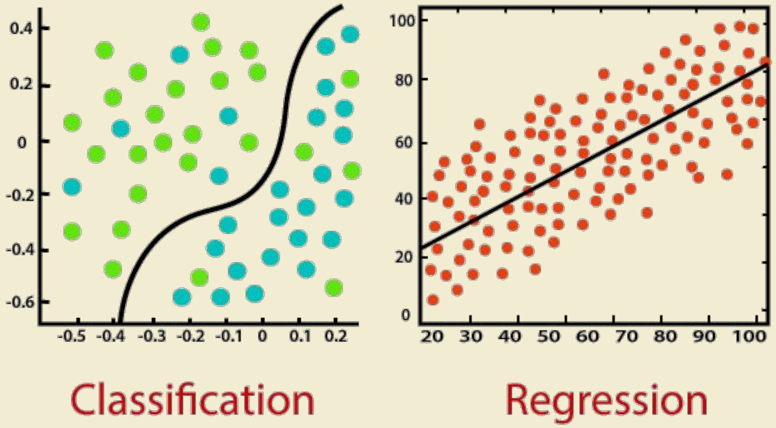
Wrapping Up

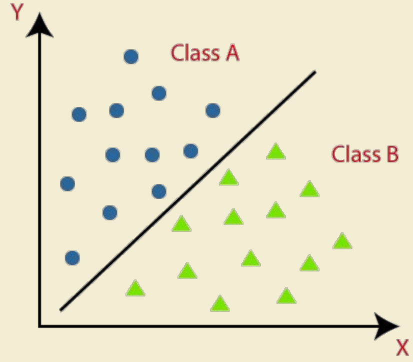
Models that we didn't do:

- Logistic Regression
- Decision Tree Classifier
- linear discriminant analysis

Models We forgot to test:

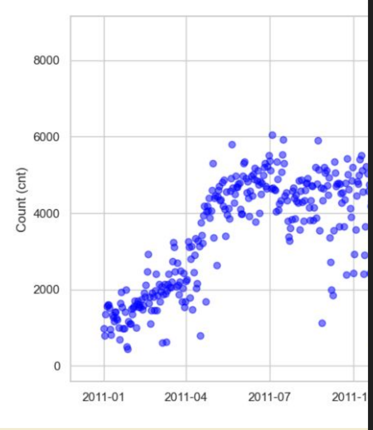
- Bagging
- Stacking





Linear Regression

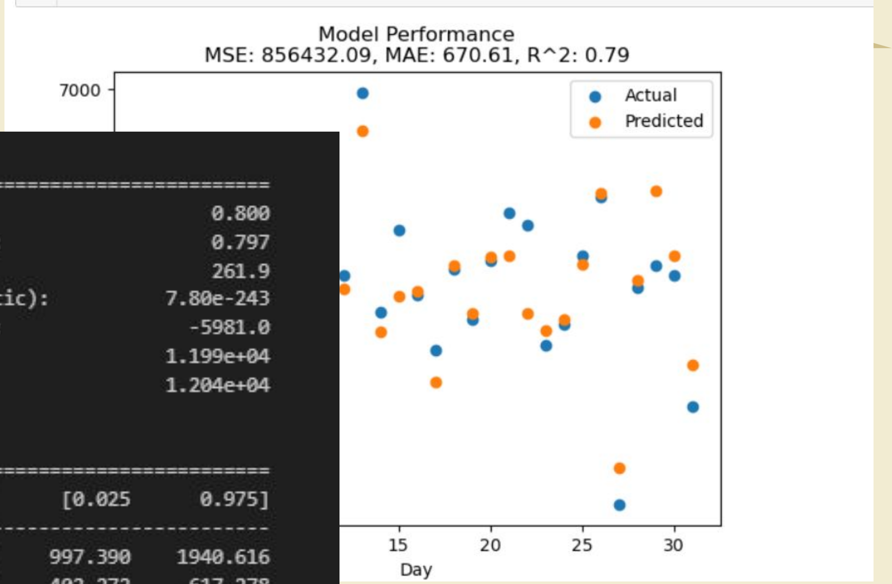
```python
import matplotlib.pyplot as plt
import pandas as pd

plt.figure(figsize=(12, 6))
day_Bike['date'] = pd.to_datetime(day_Bike[['year', 'month', 'day']])
plt.scatter(day_Bike['date'], day_Bike['cnt'], color='blue', alpha=0.5)
plt.title('Bike Rentals Over Time')
plt.xlabel('Date')
plt.ylabel('Count (cnt)')
plt.grid(True)
plt.show()
```



OLS Regression Results

Dep. Variable:	cnt	R-squared:	0.800
Model:	OLS	Adj. R-squared:	0.797
Method:	Least Squares	F-statistic:	261.9
Date:	Mon, 04 Dec 2023	Prob (F-statistic):	7.80e-243
Time:	20:29:26	Log-Likelihood:	-5981.0
No. Observations:	731	AIC:	1.199e+04
Df Residuals:	719	BIC:	1.204e+04
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1469.0031	240.218	6.115	0.000	997.390	1940.616
season	509.7752	54.757	9.310	0.000	402.272	617.278
yr	2040.7034	65.185	31.306	0.000	1912.727	2168.680
mnth	-38.9796	17.079	-2.282	0.023	-72.510	-5.449
holiday	-518.9919	201.040	-2.582	0.010	-913.688	-124.296
weekday	69.0622	16.299	4.237	0.000	37.063	101.061
workingday	120.3570	72.007	1.671	0.095	-21.013	261.727
weathersit	-610.9870	78.363	-7.797	0.000	-764.835	-457.139
temp	2028.9161	1403.671	1.445	0.149	-726.867	4784.699
atemp	3573.2743	1589.389	2.248	0.025	452.877	6693.671
hum	-1018.8616	313.995	-3.245	0.001	-1635.318	-402.405
...						



Random Forest

```
# Drop non-numeric columns before splitting

from sklearn.ensemble import RandomForestRegressor
X = day_Bike.drop(['cnt', 'casual', 'registered', "dteday"], axis=1)
y = day_Bike['cnt']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)

# Create a Random Forest Regressor model
rf_model = RandomForestRegressor(random_state = 42)

# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)

# Calculate R-squared on the test set
r_squared_rf = r2_score(y_test, y_pred_rf)
print(f'R-squared on the test set (Random Forest): {r_squared_rf}')
```

```
quared on the test set (Random Forest): 0.8716122813755863
```

- **Normal r2 Score: 0.87**
- **Gradient Boosting: 0.89**
- **Gradient Boosting & bootstrapping: 0.90**
- **Cross validation :** [-0.04569916 0.54852738 0.20664622 0.52739073 0.69939849]

ARIMA and LTSM

ARIMA wasn't a good Model

```
MAE: nan
MSE: nan
RMSE: nan
```

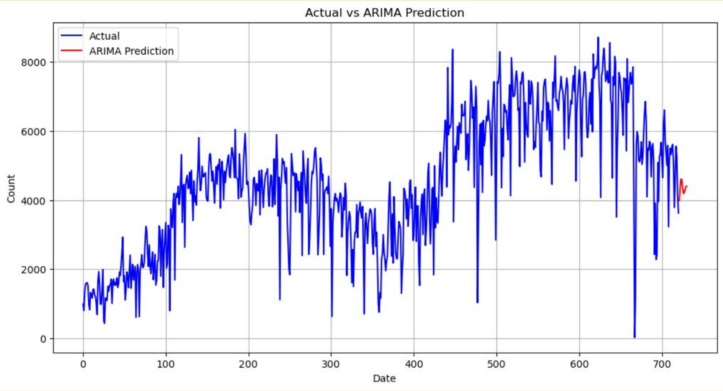
LTSM

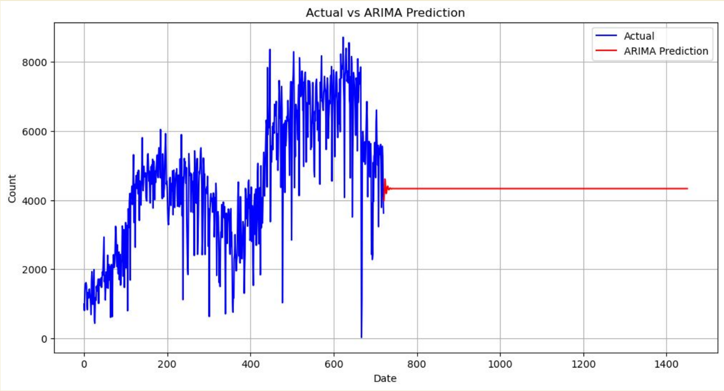
 # We want the value near 2729

 # both 2626.3992

 # without weather 2188.2783

 # just weather 1090.8662





Best Model

LTSM:

- **Lower MSE**
- **Comparable Accuracy Score to other models**

Random Forest:

- **Unsure if the implementation was correct for the large dataset, because it kept giving r2 of 1.0**

Thank You!

Bicycle icon pack

Alternative resources