IE7275 DATA MINING IN ENGINEERING
SPRING 2024

Project Report

Comparative Analysis of Supervised Machine Learning Algorithms
with Bike Sharing Demand Dataset

Submitted By -
Tejesvani Muppara Vijayaram
muppravijayaram.t@northeastern.edu
+1(857)4659450

Overview

Bike sharing systems, also known as public bicycle schemes, offer a convenient and affordable way to travel around urban areas. They provide bicycles at designated docking stations that can be rented by users for short trips. These systems typically consist of both docked bikes, where users pick up and return bicycles at designated racks, and dockless bikes, which use GPS technology to allow users to locate and rent bicycles from anywhere within the service area. Bike sharing systems benefit the environment by promoting sustainable transportation and reducing traffic congestion. They also offer health advantages by encouraging people to cycle, and can contribute to economic growth by making it easier for people to get around the city for work or leisure.

Problem Statement

In an effort to improve the efficiency and user experience of bike sharing systems, this project aims to leverage machine learning techniques to analyze and predict bike rental demand. By understanding the factors that influence how many bikes are rented at different times and locations, we can develop models that can be used to:

Optimize resource allocation: By anticipating demand, bike sharing companies can strategically distribute bikes across docking stations to ensure availability where and when riders need them. This reduces the frustration of users encountering empty stations and improves overall system efficiency.

Inform system expansion: Demand predictions can guide decisions about expanding bike sharing systems into new neighborhoods or increasing the number of docking stations in high-demand areas. This fosters a more equitable and accessible system that caters to a wider range of users.

Develop targeted marketing campaigns: Understanding user behavior through rental data can help tailor marketing efforts to specific demographics and encourage ridership during off-peak hours or in under-utilized areas.

About the Dataset

The bike sharing demand dataset contains historical hourly rental information, from a specific city. It provides data points for each hour, aiding the analysis of trends and seasonality in ridership. This data is taken from UCI Machine Learning Repository and covers the hourly rental information for a period of two years.

Features include:

- **Datetime:** This feature represents the date and timestamp for each data point, typically recorded hourly. It allows for analysis of trends and seasonality in ridership patterns.
- **Season:** This categorical variable indicates the season (e.g., 1: spring, 2: summer, 3: fall, 4: winter) during which the rental occurred.
- **Workingday:** This binary variable indicates whether the day is a weekday (1) or a weekend/holiday (0).
- **Holiday:** This binary variable indicates whether the day is a holiday (1) or not (0).
- **Weather:** This categorical variable describes the prevailing weather conditions during the rental hour (e.g., clear, cloudy, mist).
- **Temp:** This numerical feature represents the temperature in Celsius at the time of rental.
- **Atemp:** This numerical feature represents the "feels like" temperature in Celsius at the time of rental.
- **Humidity:** This numerical feature represents the relative humidity at the time of rental.

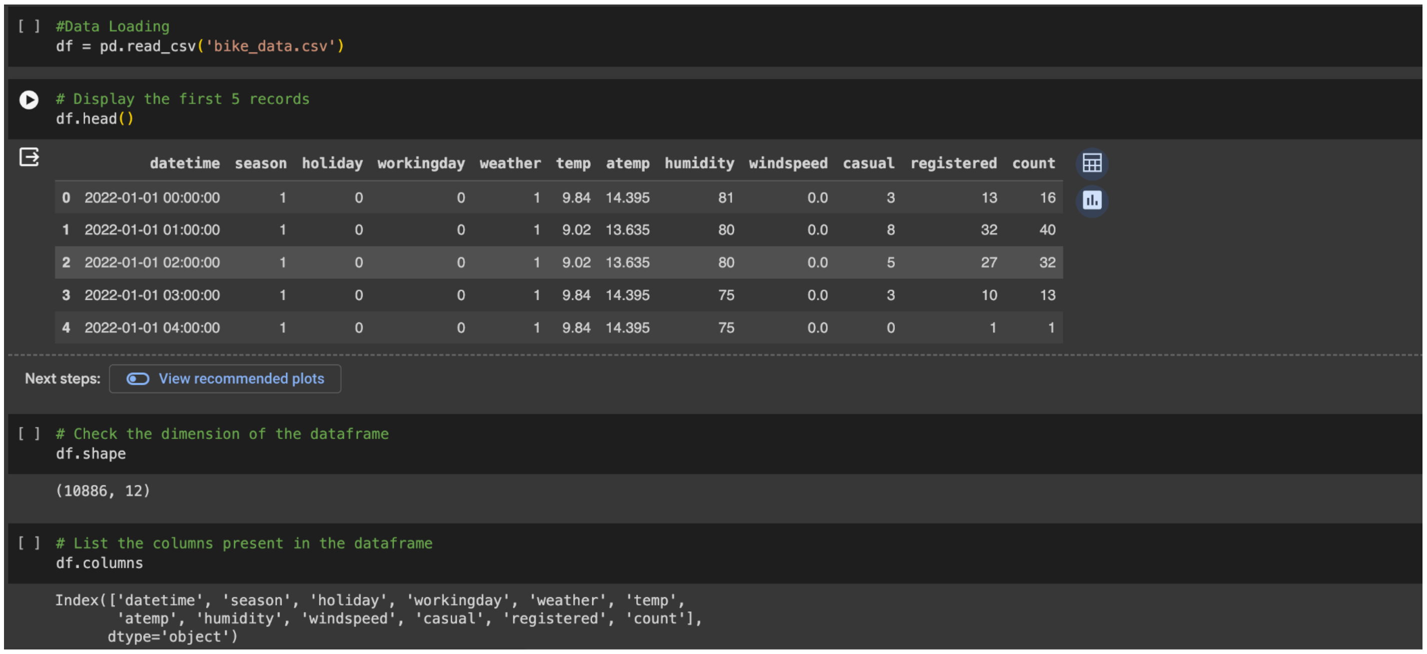
- **Windspeed:** This numerical feature represents the wind speed at the time of rental.
- **Casual:** This numerical feature represents the number of bike rentals initiated by non-registered users during the hour.
- **Registered:** This numerical feature represents the number of bike rentals initiated by registered users during the hour.
- **Count:** This numerical feature represents the total number of bike rentals (casual + registered) during the hour. This is the target variable you might be interested in predicting.

This dataset provides valuable insights into factors influencing bike rental demand. By analyzing the relationships between these features, you can gain a deeper understanding of how weather, seasonality, holidays, and user types affect ridership patterns.

Data Loading and Pre-Processing

The **pd.read_csv**('bike_data.csv') loads the bike sharing dataset from a CSV file named into a Pandas DataFrame called df. This step is essential to bring the data into the Python environment for further analysis.

Next, **df.head()** displays the first 5 records of the DataFrame df, providing an initial glimpse of the dataset's structure and content. The dimensions of the DataFrame, i.e., the number of rows and columns present in the dataset are checked using **df.shape**. This helps in understanding the overall size of the dataset. While, **df.columns** lists all the column names present in the DataFrame, aiding the identify the features or variables available in the dataset.



```
#Data Loading
df = pd.read_csv('bike_data.csv')
```

```
# Display the first 5 records
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2022-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2022-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2022-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2022-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2022-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
# Check the dimension of the dataframe
df.shape
```

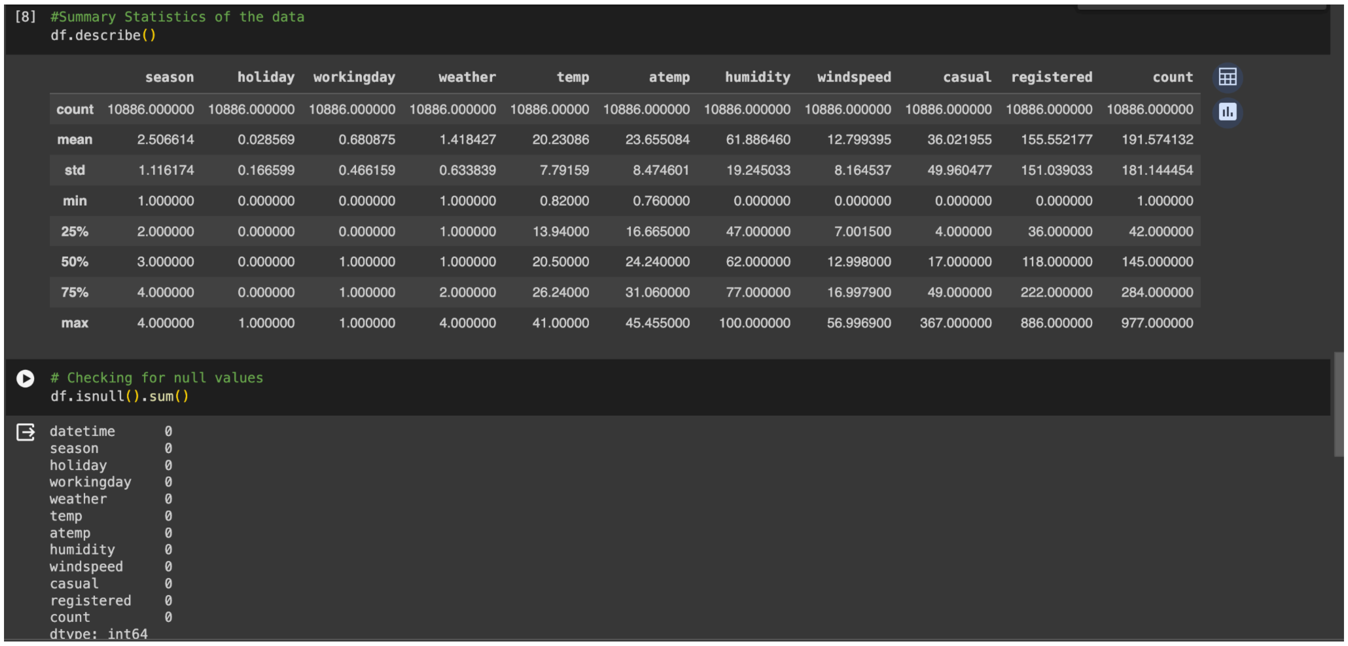
```
(10886, 12)
```

```
# List the columns present in the dataframe
df.columns
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

The **df.describe()** output allows us to understand the central tendency and spread of numerical features like temperature, humidity, windspeed, etc. It also provides insights into potential outliers or abnormalities in the data.

The result of **df.isnull().sum()** reassures us that there are no missing values in the dataset, which simplifies subsequent data cleaning and preprocessing steps. This suggests that the dataset is relatively clean and ready for further analysis or modeling without the need for imputation or handling missing data.



```
[8] #Summary Statistics of the data
    df.describe()
```

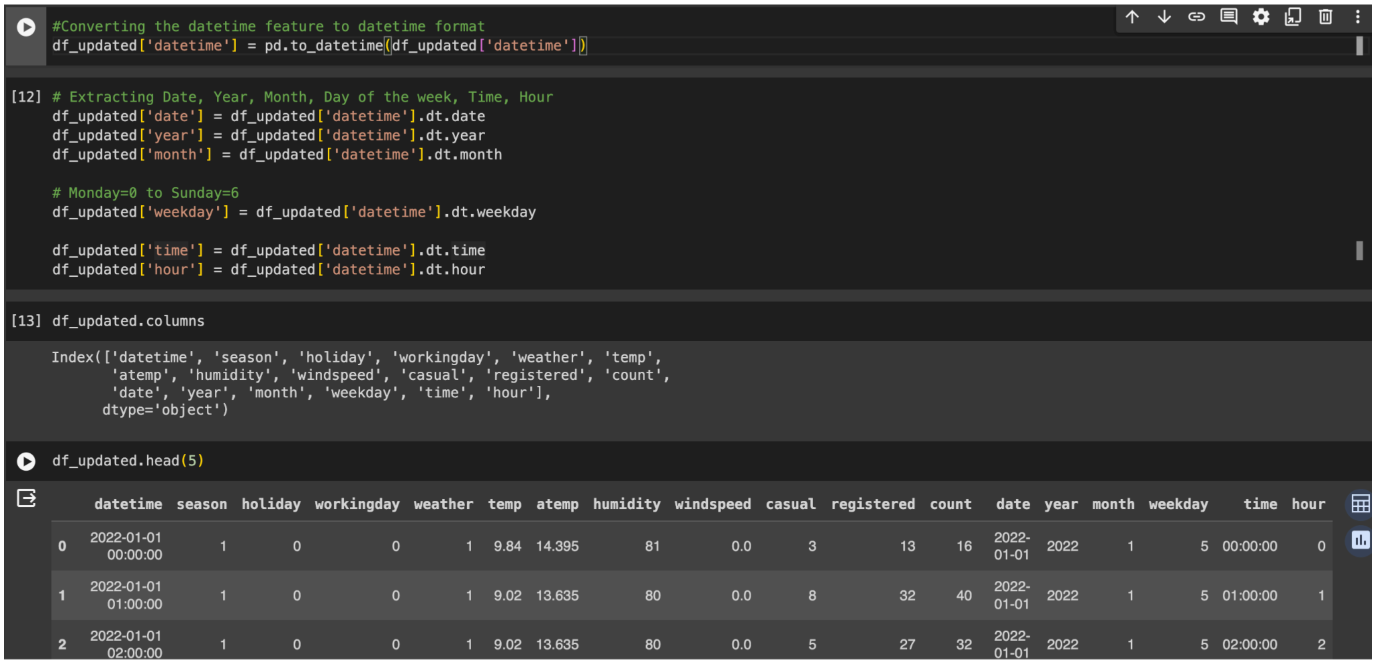
	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
count	10886.000000	10886.000000	10886.000000	10886.000000	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2.506614	0.028569	0.680875	1.418427	20.23086	23.655084	61.886460	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

```
# Checking for null values
df.isnull().sum()
```

```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

Feature Engineering

The feature engineering steps enable the creation of new columns containing useful temporal information extracted from the original 'datetime' column. By extracting and storing individual components such as date, year, month, weekday, time, and hour, the dataset becomes enriched with additional features that may provide valuable insights for analysis and modeling tasks. These new features can potentially enhance the predictive power of machine learning models by capturing temporal patterns and trends in the data.



```
#Converting the datetime feature to datetime format
df_updated['datetime'] = pd.to_datetime(df_updated['datetime'])
```

```
[12] # Extracting Date, Year, Month, Day of the week, Time, Hour
     df_updated['date'] = df_updated['datetime'].dt.date
     df_updated['year'] = df_updated['datetime'].dt.year
     df_updated['month'] = df_updated['datetime'].dt.month

     # Monday=0 to Sunday=6
     df_updated['weekday'] = df_updated['datetime'].dt.weekday

     df_updated['time'] = df_updated['datetime'].dt.time
     df_updated['hour'] = df_updated['datetime'].dt.hour
```

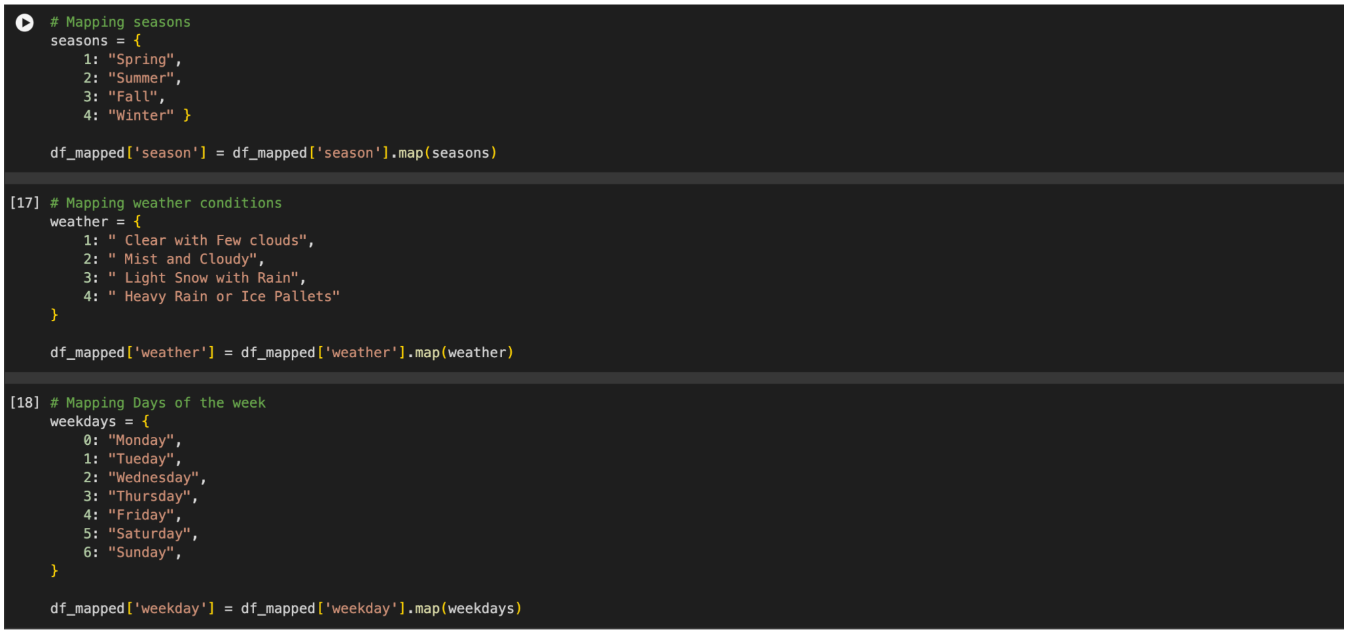
```
[13] df_updated.columns

     Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
            'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count',
            'date', 'year', 'month', 'weekday', 'time', 'hour'],
           dtype='object')
```

```
df_updated.head(5)
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	date	year	month	weekday	time	hour
0	2022-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	2022-01-01	2022	1	5	00:00:00	0
1	2022-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	2022-01-01	2022	1	5	01:00:00	1
2	2022-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	2022-01-01	2022	1	5	02:00:00	2

Exploratory Data Analysis

The provided code performs mapping of encoded values to their corresponding categorical labels in the DataFrame. This mapping is a preparatory step typically performed before conducting Exploratory Data Analysis (EDA) to make the analysis more interpretable and meaningful.



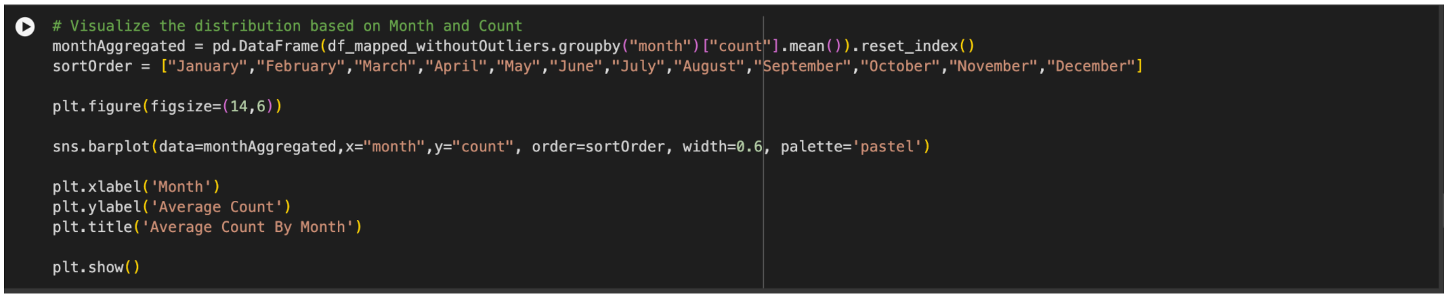
```
# Mapping seasons
seasons = {
    1: "Spring",
    2: "Summer",
    3: "Fall",
    4: "Winter" }

df_mapped['season'] = df_mapped['season'].map(seasons)
```

```
[17] # Mapping weather conditions
weather = {
    1: " Clear with Few clouds",
    2: " Mist and Cloudy",
    3: " Light Snow with Rain",
    4: " Heavy Rain or Ice Pallets"
}

df_mapped['weather'] = df_mapped['weather'].map(weather)
```

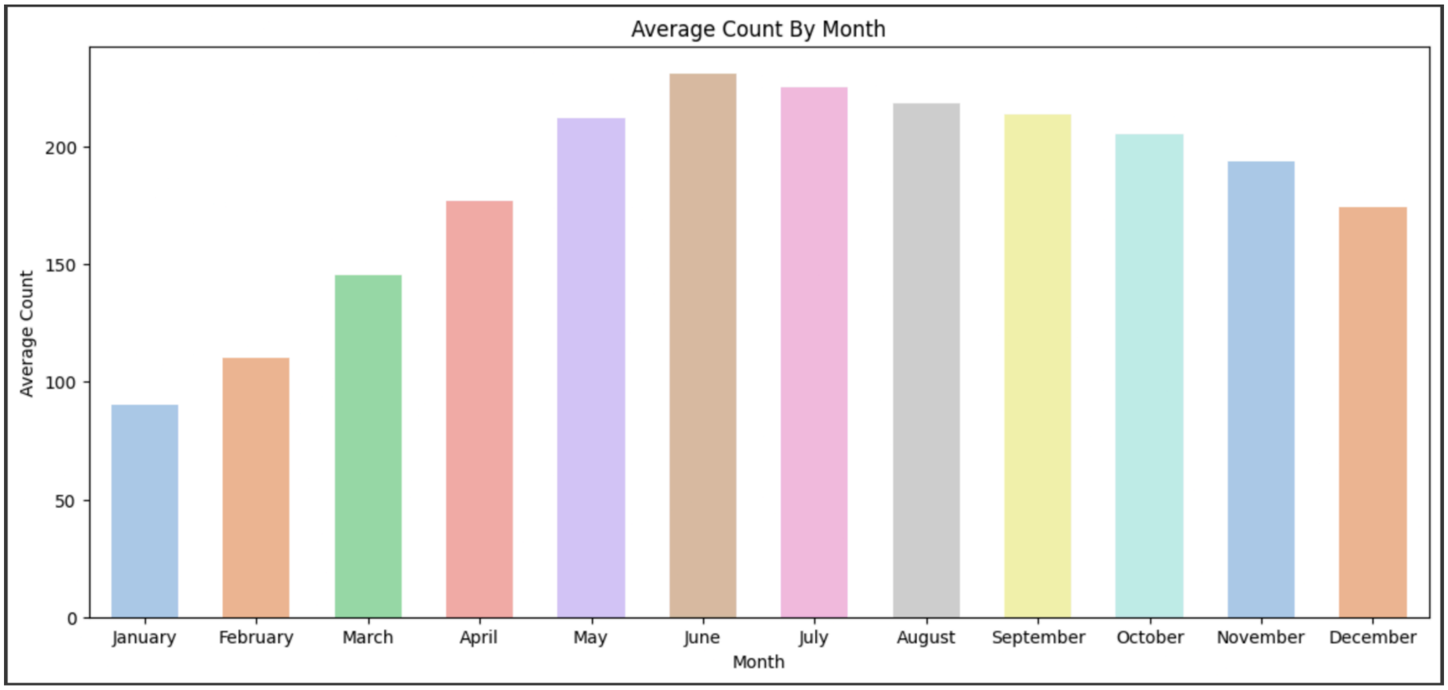
```
[18] # Mapping Days of the week
weekdays = {
    0: "Monday",
    1: "Tueday",
    2: "Wednesday",
    3: "Thursday",
    4: "Friday",
    5: "Saturday",
    6: "Sunday",
}

df_mapped['weekday'] = df_mapped['weekday'].map(weekdays)
```

Some identified patterns on performing EDA were -

1. **Seasonal Analysis:** Fluctuations in Bike Demand Across the Months of the Year

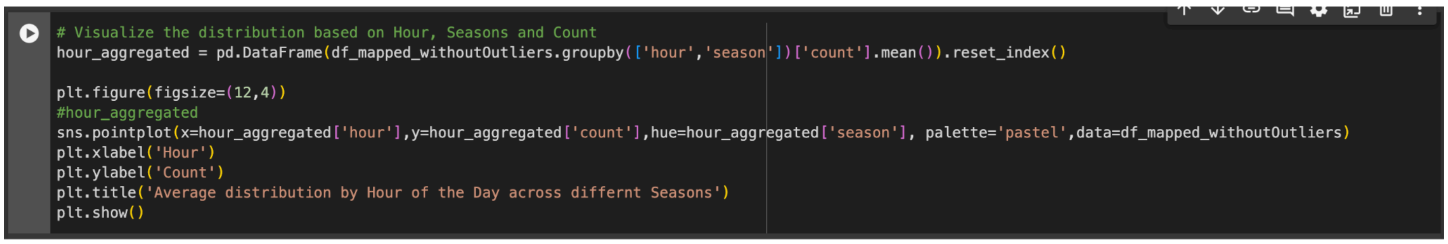


```
# Visualize the distribution based on Month and Count
monthAggregated = pd.DataFrame(df_mapped_withoutOutliers.groupby("month")["count"].mean()).reset_index()
sortOrder = ["January","February","March","April","May","June","July","August","September","October","November","December"]

plt.figure(figsize=(14,6))

sns.barplot(data=monthAggregated,x="month",y="count", order=sortOrder, width=0.6, palette='pastel')

plt.xlabel('Month')
plt.ylabel('Average Count')
plt.title('Average Count By Month')

plt.show()
```

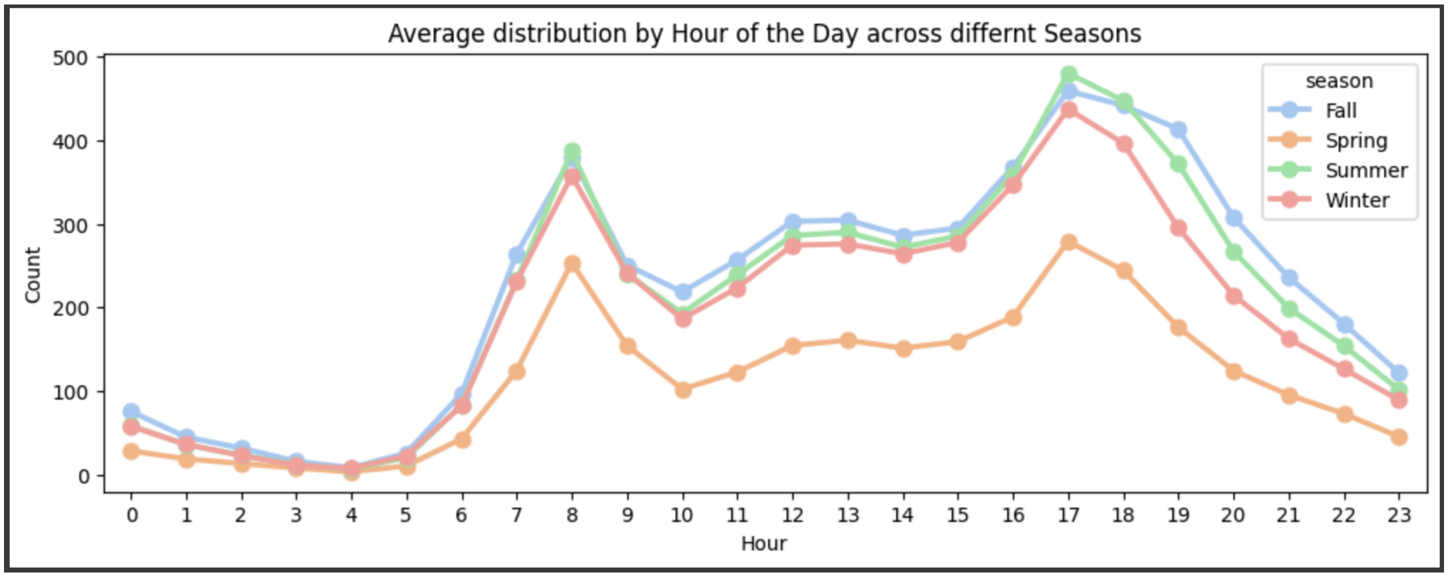


Peak Seasons: June, July, August, and September have the highest average counts, indicating peak seasons for bike rentals. This could be due to favorable weather conditions, longer daylight hours, and potentially increased tourism or outdoor activities during these months.
Low Seasons: January and February have the lowest average counts, suggesting lower demand for bike rentals during the colder winter months. Weather conditions such as snow, rain, or cold temperatures may discourage outdoor activities, resulting in reduced bike rental usage.
Seasonal Trends: Overall, there appears to be a seasonal trend in bike rental demand, with higher counts during warmer months and lower counts during colder months. Understanding these seasonal variations can help bike-sharing companies adjust their operations, marketing strategies, and resource allocation to better meet customer demand throughout the year.

2. **Hourly Distribution of Bike Demand Across Seasons:** Understanding Seasonal Variations in Hourly Bike Usage

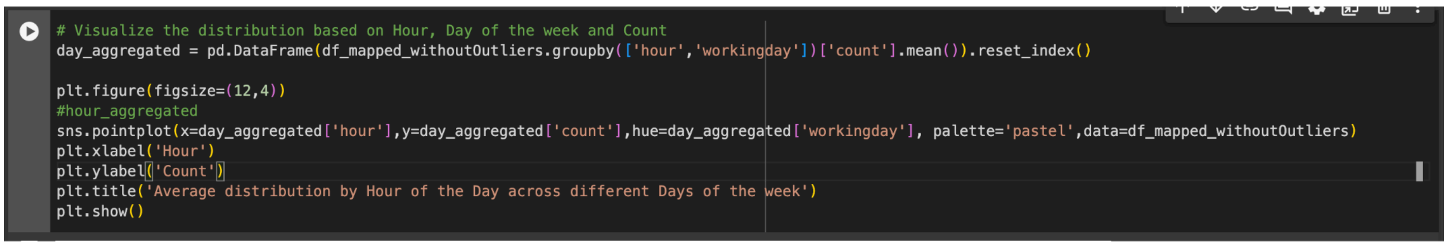


```
# Visualize the distribution based on Hour, Seasons and Count
hour_aggregated = pd.DataFrame(df_mapped_withoutOutliers.groupby(['hour','season'])['count'].mean()).reset_index()

plt.figure(figsize=(12,4))
#hour_aggregated
sns.pointplot(x=hour_aggregated['hour'],y=hour_aggregated['count'],hue=hour_aggregated['season'], palette='pastel',data=df_mapped_withoutOutliers)
plt.xlabel('Hour')
plt.ylabel('Count')
plt.title('Average distribution by Hour of the Day across differnt Seasons')
plt.show()
```

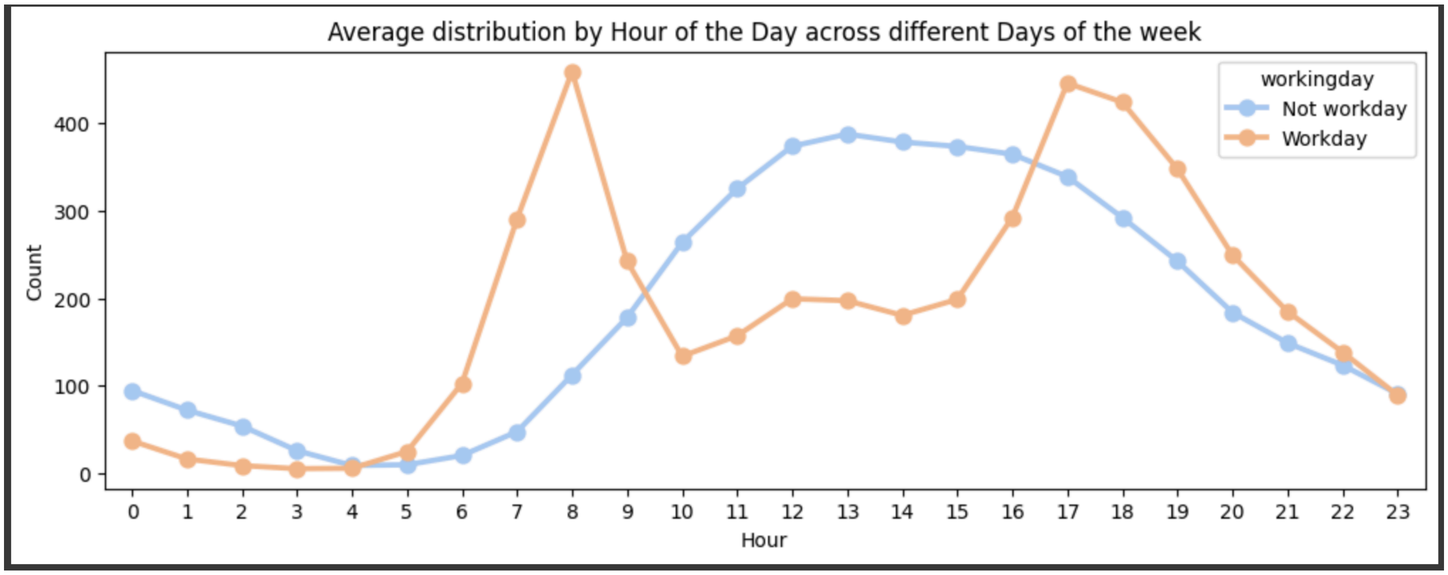


Hourly Patterns: There is variation in bike rental demand throughout the day, with some hours showing higher average counts compared to others. For example, hours such as 8 AM and 5 PM might correspond to peak commuting times, resulting in higher counts, while late-night hours might have lower counts.
Seasonal Trends: Different seasons exhibit varying levels of bike rental demand at different hours of the day. For instance, during the summer season, there are higher average counts during daytime hours, indicating increased outdoor activities and leisure cycling. In contrast, during the winter season, there might be lower counts during daylight hours due to colder weather and reduced outdoor activity, but higher counts during evening hours, possibly reflecting recreational or commuter usage.

3. **Hourly Distribution of Bike Demand Across Days of the Week:** Exploring Weekly Patterns in Bike Usage Throughout the Day



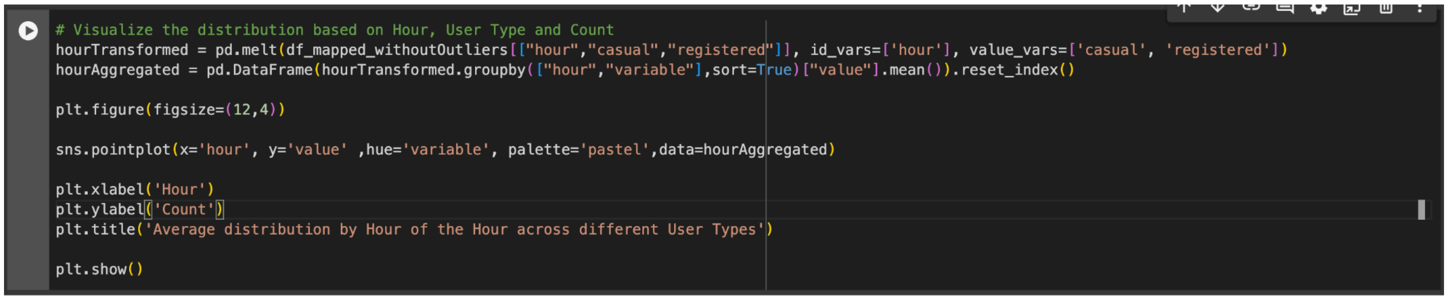
```
# Visualize the distribution based on Hour, Day of the week and Count
day_aggregated = pd.DataFrame(df_mapped_withoutOutliers.groupby(['hour','workingday'])['count'].mean()).reset_index()

plt.figure(figsize=(12,4))
#hour_aggregated
sns.pointplot(x=day_aggregated['hour'],y=day_aggregated['count'],hue=day_aggregated['workingday'], palette='pastel',data=df_mapped_withoutOutliers)
plt.xlabel('Hour')
plt.ylabel('Count')
plt.title('Average distribution by Hour of the Day across different Days of the week')
plt.show()
```



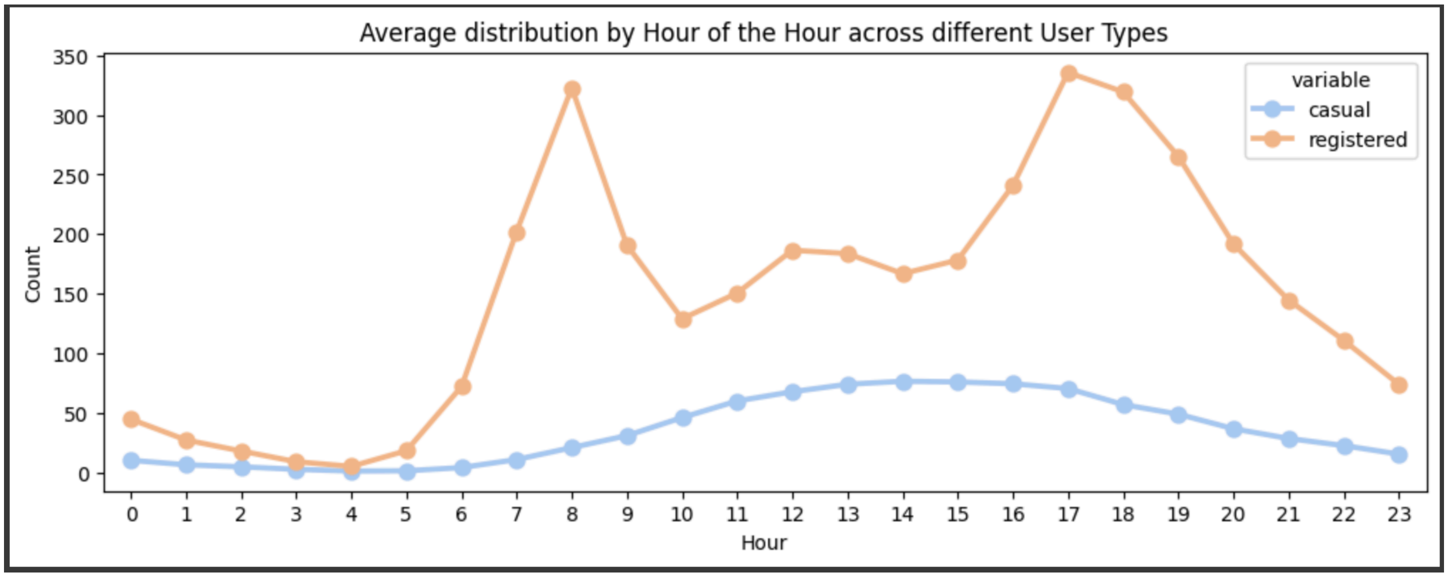
Peak Demand on Workdays: There is a noticeable increase in bike rental demand during typical office start hours (7 am - 9 am) and end hours (4 pm - 7 pm) on workdays. This pattern suggests that a significant portion of users on workdays are likely using the bikes for their daily commute to and from the office.
High Usage on Non-Working Days: On non-working days, such as weekends or holidays, bike rental demand peaks between 9 am and 8 pm, indicating more leisure or recreational usage rather than commuting. This suggests that users on non-working days are likely using the bikes for various purposes such as sightseeing, exercise, or recreational activities.
Implication for Resource Allocation: Given the observed peak demand during office start and end hours on workdays, it is suggested to increase the number of bikes near office spaces or in areas with high employment density. This can help ensure that there are enough bikes available to meet the commuting needs of users during peak hours, thereby improving customer satisfaction and usability of the bike-sharing system.

Overall, these observations provide valuable insights for bike-sharing companies to optimize resource allocation, station planning, and operational strategies to better serve the diverse needs of users on both workdays and non-working days.

4. **Hourly Distribution of Bike Demand Across User Types:** Analyzing Usage Patterns Throughout the Day for Different User Categories



```
# Visualize the distribution based on Hour, User Type and Count
hourTransformed = pd.melt(df_mapped_withoutOutliers[["hour","casual","registered"]], id_vars=['hour'], value_vars=['casual', 'registered'])
hourAggregated = pd.DataFrame(hourTransformed.groupby(["hour","variable"],sort=True)["value"].mean()).reset_index()

plt.figure(figsize=(12,4))

sns.pointplot(x='hour', y='value' ,hue='variable', palette='pastel',data=hourAggregated)

plt.xlabel('Hour')
plt.ylabel('Count')
plt.title('Average distribution by Hour of the Hour across different User Types')

plt.show()
```

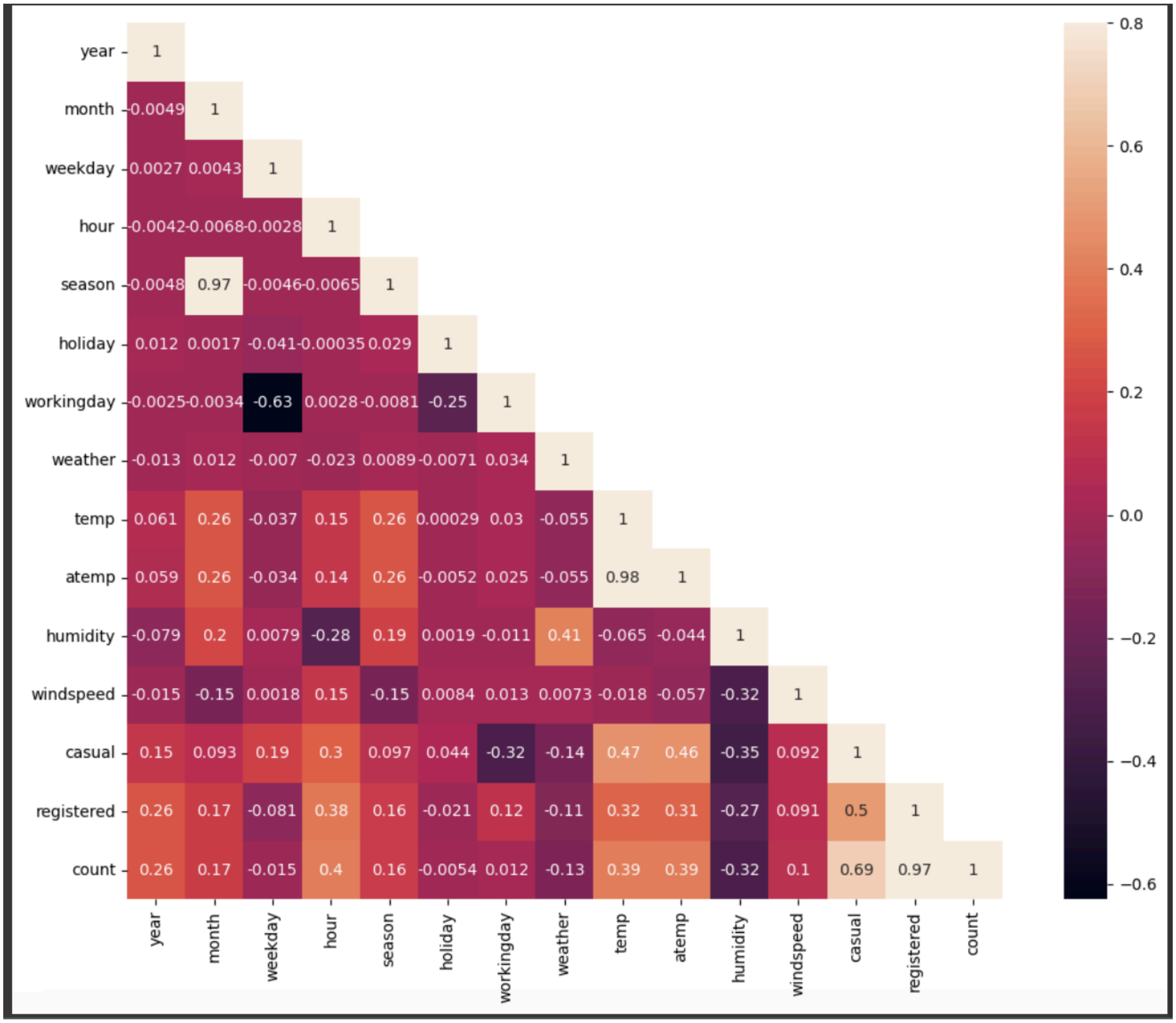


Trend Analysis: The data shows that the registered users have significantly higher counts during typical office start hours (7 am - 9 am) and end hours (4 pm - 7 pm), compared to casual users. This aligns with the observation that most registered users may be using the bikes for commuting to and from work.
Inference: It can be inferred that most registered users are likely in the "office-going" category and use the bikes frequently for commuting is supported by the trend observed in the data. The higher counts of registered users during office hours suggest that they have registered for the bike services to facilitate their daily commute.
Implication for Service Providers: Understanding the commuting patterns of registered users can help bike-sharing service providers optimize their services, such as increasing bike availability near office spaces during peak commuting hours. It also highlights the importance of catering to the needs of registered users by providing convenient and reliable transportation options for their daily commutes.

Overall, the data analysis confirms the inference that the majority of registered users may be commuters who use the bike-sharing service for their daily travel needs, particularly during typical office hours.

Correlation Analysis

Based on the correlation matrix and the following observations were made:
Positive Correlations: The count of bike rentals shows positive correlations with variables such as year, month, hour, season, temperature (temp and atemp), and windspeed. This suggests that these factors tend to increase along with the count of bike rentals.
Negative Correlations: On the other hand, the count of bike rentals demonstrates negative correlations with variables such as weather and humidity. This indicates that as weather conditions worsen or humidity increases, the count of bike rentals tends to decrease.
High Correlations: Variables such as temp and atemp exhibit a very high positive correlation, which is expected since they represent temperature-related measurements. Similarly, seasons and months show a high positive correlation, reflecting the seasonal variations. As a result the following features show multicollinearity.



Model Selection and Hyperparameter Tuning

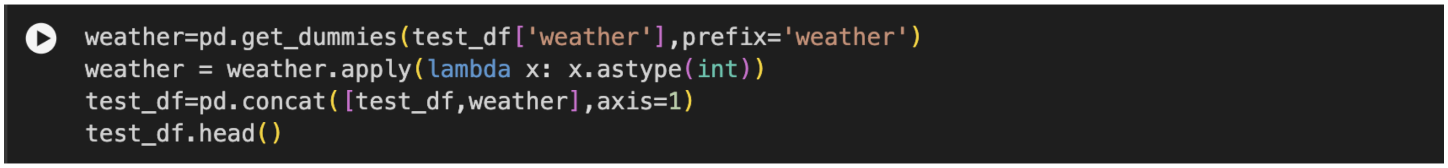
Model selection is a pivotal stage in the machine learning workflow, crucial for determining the most effective algorithm or combination of algorithms to address a specific predictive modeling task. It involves evaluating various algorithms and assessing their performance based on predefined metrics.

Hyperparameter tuning constitutes the subsequent step, focusing on optimizing the hyperparameters of a machine learning algorithm to enhance its performance on unseen data. Hyperparameters are parameters predefined before the training process, distinct from model parameters learned from data. The objective is to identify the optimal combination of hyperparameter values that maximizes the model's performance, typically assessed through cross-validation or a validation set.

In the pursuit of optimal hyperparameter combinations, GridSearch emerges as a prominent technique. GridSearch systematically explores a predefined grid of hyperparameter values, rigorously evaluating each combination to select the one yielding the best performance. This exhaustive search entails training and testing the model with every specified combination of hyperparameters, employing a chosen evaluation metric to determine the optimal configuration.

While GridSearch is particularly effective for scenarios with a limited number of hyperparameters and a manageable search space, its computational demands escalate with larger hyperparameter spaces. Despite this, GridSearch remains a powerful tool in the machine learning arsenal, facilitating systematic and efficient hyperparameter optimization to enhance model performance and generalization.

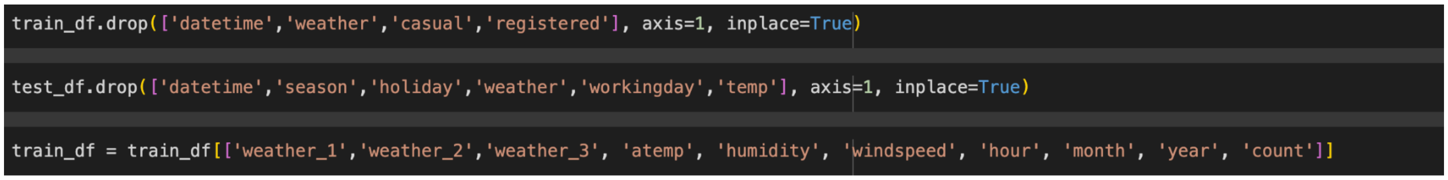
This code snippet demonstrates the process of one-hot encoding categorical variables. One-hot encoding enables the inclusion of categorical variables in machine learning models that require numerical input. By creating binary features for each category, one-hot encoding preserves the categorical information while making it suitable for model training. This process enhances the model's ability to capture the effects of different weather conditions on the target variable, thereby improving predictive accuracy.



```
weather=pd.get_dummies(test_df['weather'],prefix='weather')
weather = weather.apply(lambda x: x.astype(int))
test_df=pd.concat([test_df,weather],axis=1)
test_df.head()
```

```
[47] weather=pd.get_dummies(test_df['weather'],prefix='weather')
     weather = weather.apply(lambda x: x.astype(int))
     test_df=pd.concat([test_df,weather],axis=1)
     test_df.head()
```

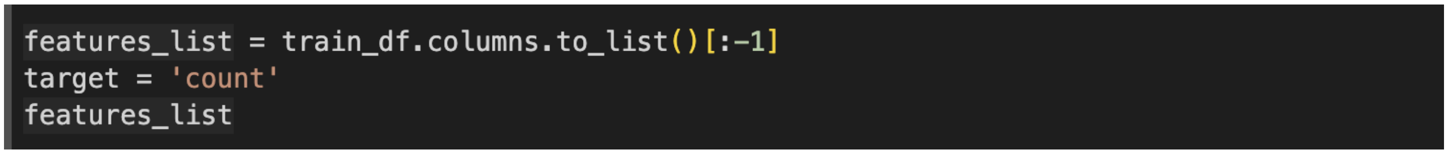
In the provided code snippet, columns are being dropped from two DataFrames, 'train_df' and 'test_df', to remove redundant or multicollinear features.



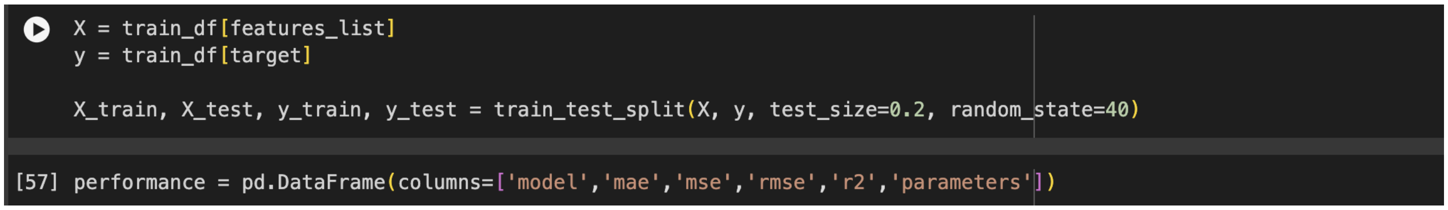
```
train_df.drop(['datetime','weather','casual','registered'], axis=1, inplace=True)

test_df.drop(['datetime','season','holiday','weather','workingday','temp'], axis=1, inplace=True)

train_df = train_df[['weather_1','weather_2','weather_3', 'atemp', 'humidity', 'windspeed', 'hour', 'month', 'year', 'count']]
```

The dataset is split into training and testing sets to assess the performance of machine learning models on unseen data. Utilizing train_test_split ensures that the model's performance is evaluated on independent data, facilitating unbiased assessment and ensuring the model's generalizability to new, unseen instances.

An empty DataFrame is initialized to store model performance metrics, allowing for easy comparison and analysis of different models' effectiveness in predicting the target variable.



```
features_list = train_df.columns.to_list()[:-1]
target = 'count'
features_list
```



```
X = train_df[features_list]
y = train_df[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)
```

```
[57] performance = pd.DataFrame(columns=['model','mae','mse','rmse','r2','parameters'])
```

K - Nearest Neighbors

K-Nearest Neighbors (KNN) regression is a straightforward yet effective algorithm used for predicting continuous outcomes. Operating on the principle of proximity, KNN regression computes predictions for new data points by averaging the target values of their k nearest neighbors in the training dataset. This method doesn't require explicit model training; instead, it memorizes the training instances. KNN's simplicity and ease of implementation make it popular, particularly for initial exploratory analysis and quick prototyping. However, its performance can be sensitive to the choice of the hyperparameter 'k' and the scaling of features.

Performance of the Base Model

Parameters for KNN Regression:
Number of Neighbors (k): 5
Mean Absolute Error (MAE): 82.81010564997702
Mean Squared Error (MSE): 15111.372090032153
Root Mean Squared Error (RMSE): 122.92832094367901
R-squared (R2): 0.5513506417069289

The base KNN Regression model with 5 neighbors achieved a Mean Absolute Error (MAE) of 82.81, indicating, on average, predictions were off by approximately 82.81 units. The Mean Squared Error (MSE) was 15111.37, suggesting a higher sensitivity to large errors. The Root Mean Squared Error (RMSE) was 122.93, interpreting the average magnitude of the errors. The R-squared (R2) value of 0.55 signifies that the model explains approximately 55% of the variance in the target variable, indicating moderate predictive performance. This model's performance serves as a baseline for comparison with further iterations and improvements in the modeling process.

Standardization

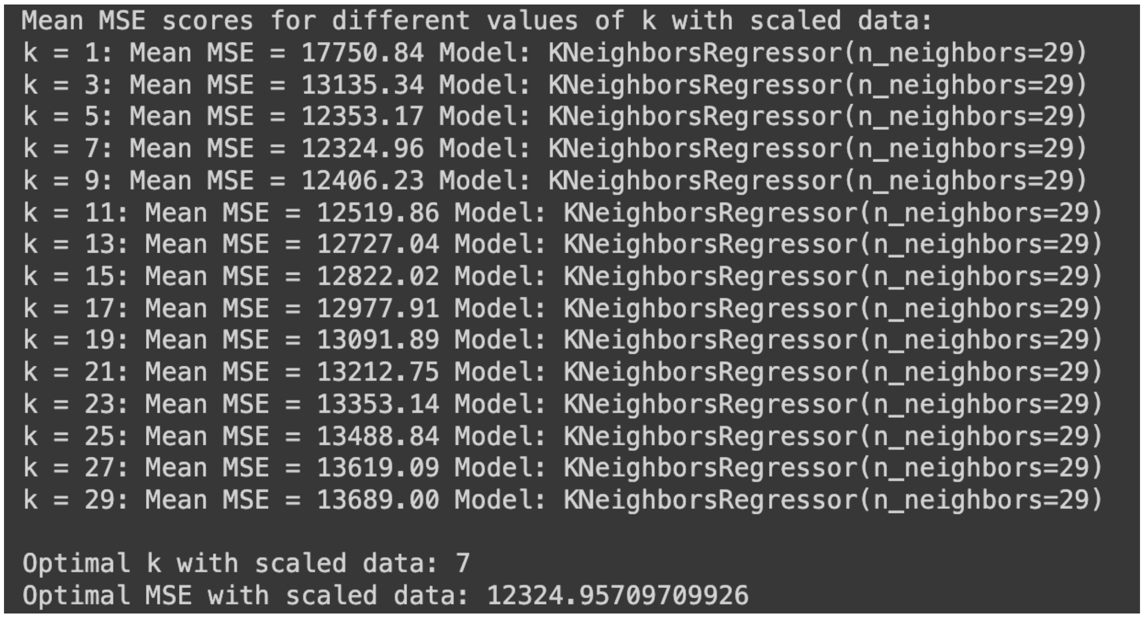
Since KNN determines similarity between data points based on their feature distances, features with larger scales or ranges could disproportionately influence these calculations, potentially leading to biased predictions. Standardizing the features ensures that each feature contributes equally to the distance metric, thereby preventing any single feature from dominating the model's decision-making process. By rescaling features to have a mean of 0 and a standard deviation of 1, standardization facilitates fair and accurate distance calculations, ultimately improving the KNN regression model's performance, enhancing its ability to identify meaningful patterns and make more reliable predictions.

After Standardization the MSE and r-squared are as follows -
Mean Squared Error (MSE) for scaled data: 12511.33003215434
R-squared (R2) for scaled data: 0.6332162574445132

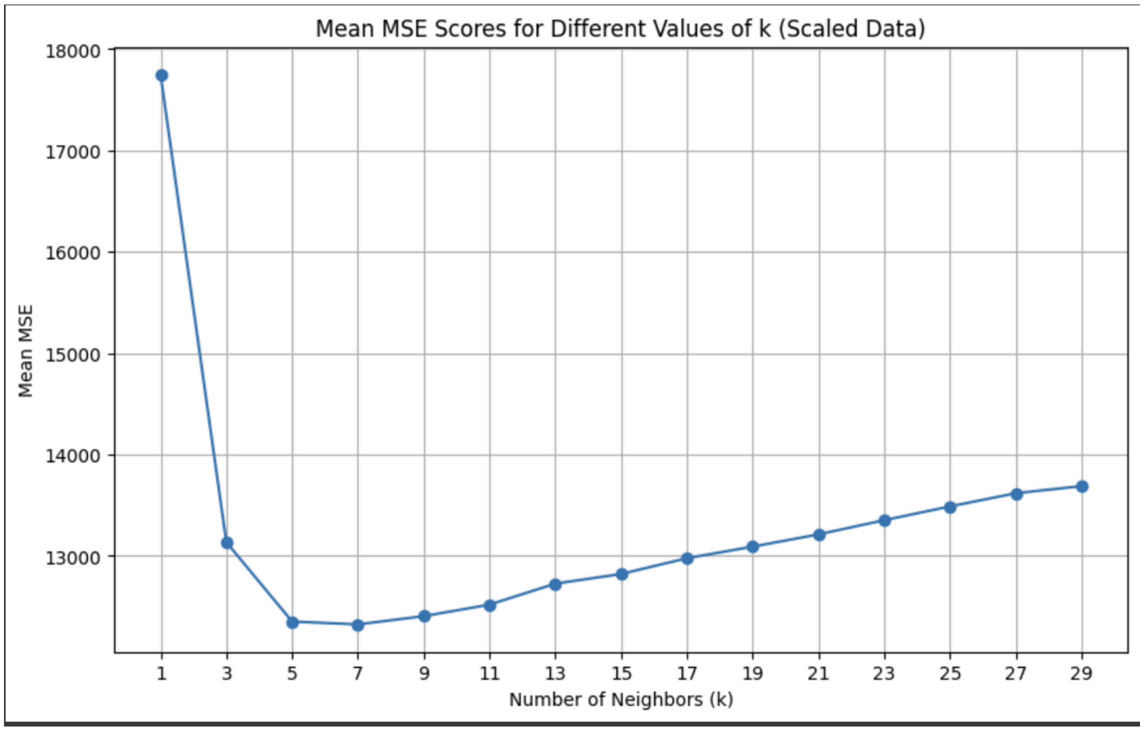
After standardization, there is an improvement in model performance. The mean squared error (MSE) decreases from 15111.37 to 12511.33, indicating that the model's predictions are closer to the actual values after standardization. Additionally, the R-squared value increases from 0.551 to 0.633, suggesting that a larger proportion of the variance in the target variable is explained by the features after standardization. Overall, standardization has led to better model performance, as indicated by the reduction in error metrics and the improvement in the coefficient of determination (R-squared).

Identification of Optimal k

Systematically iterating over a range of k values was performed for selecting the optimal k that minimizes the mean cross-validated Mean Squared Error (MSE) is commonly referred to as hyperparameter tuning or model selection. In this specific case, it involves evaluating the performance of the KNN regression model with different values of k using cross-validation and selecting the value of k that yields the lowest average MSE. This process helps to find the optimal balance between model complexity and performance, ensuring that the KNN model generalizes well to unseen data.



```
Mean MSE scores for different values of k with scaled data:
k = 1: Mean MSE = 17750.84 Model: KNeighborsRegressor(n_neighbors=29)
k = 3: Mean MSE = 13135.34 Model: KNeighborsRegressor(n_neighbors=29)
k = 5: Mean MSE = 12353.17 Model: KNeighborsRegressor(n_neighbors=29)
k = 7: Mean MSE = 12324.96 Model: KNeighborsRegressor(n_neighbors=29)
k = 9: Mean MSE = 12406.23 Model: KNeighborsRegressor(n_neighbors=29)
k = 11: Mean MSE = 12519.86 Model: KNeighborsRegressor(n_neighbors=29)
k = 13: Mean MSE = 12727.04 Model: KNeighborsRegressor(n_neighbors=29)
k = 15: Mean MSE = 12822.02 Model: KNeighborsRegressor(n_neighbors=29)
k = 17: Mean MSE = 12977.91 Model: KNeighborsRegressor(n_neighbors=29)
k = 19: Mean MSE = 13091.89 Model: KNeighborsRegressor(n_neighbors=29)
k = 21: Mean MSE = 13212.75 Model: KNeighborsRegressor(n_neighbors=29)
k = 23: Mean MSE = 13353.14 Model: KNeighborsRegressor(n_neighbors=29)
k = 25: Mean MSE = 13488.84 Model: KNeighborsRegressor(n_neighbors=29)
k = 27: Mean MSE = 13619.09 Model: KNeighborsRegressor(n_neighbors=29)
k = 29: Mean MSE = 13689.00 Model: KNeighborsRegressor(n_neighbors=29)

Optimal k with scaled data: 7
Optimal MSE with scaled data: 12324.95709709926
```



An optimal k value of 7 was identified.

Performance after Hyperparameter Tuning

Best hyperparameters with scaled data: {'n_neighbors': 9, 'p': 1, 'weights': 'distance'}
Mean Absolute Error (MAE) with scaled data: 72.09127649150015
Mean Squared Error (MSE) with scaled data: 11598.103610131744
Root Mean Squared Error (RMSE) with scaled data: 107.69449201389895
R-squared with scaled data: 0.6599885193870212

The hyperparameter tuning process resulted in the selection of optimal hyperparameters, including 9 neighbors, a Manhattan distance metric (p=1), and distance-based weights. The mean absolute error (MAE) decreased to 72.09, indicating improved accuracy in predicting the bike sharing count. The mean squared error (MSE) decreased to 11598.10, reflecting the reduction in prediction errors after tuning. The root mean squared error (RMSE) also decreased to 107.69, which means the average magnitude of errors in prediction reduced.
The coefficient of determination (R-squared) increased to 0.660, indicating that approximately 66% of the variance in the target variable is explained by the features after hyperparameter tuning and scaling.
Overall, hyperparameter tuning and scaling led to notable enhancements in the KNN model's predictive performance.

Decision Tree

A Decision Tree is a versatile and intuitive supervised learning algorithm that be used for both regression tasks. It operates by recursively partitioning the feature space into subsets based on the values of input features, resulting in a tree-like structure where each internal node represents a decision based on a feature, and each leaf node corresponds to a predicted outcome. Decision Trees are interpretable and can handle both numerical and categorical data, making them easy to understand and visualize.

Performance of Base Model

Mean Absolute Error (MAE): 68.15112540192926
Mean Squared Error (MSE): 13294.940514469454
Root Mean Squared Error (RMSE): 115.30368820844134
R-squared (R2): 0.6052796202208677

The base model performance for Decision Tree Regression indicates that the model's predictions have a mean absolute error (MAE) of approximately 68.15, indicating, on average, how much the predicted values deviate from the actual values. The mean squared error (MSE) is around 13294.94, reflecting the average squared difference between predicted and actual values. The root mean squared error (RMSE) is approximately 115.30, representing the square root of the MSE and indicating the average magnitude of error in the model's predictions. The R-squared (R2) value of approximately 0.61 suggests that the model explains about 60.53% of the variance in the target variable, indicating a moderate level of predictive performance.

Performance after Hyperparameter Tuning

Best hyperparameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}
MAE (Tuned): 61.864673350529465
MSE (Tuned): 8688.288664788439
RMSE (Tuned): 93.21099004295813
R-squared (Tuned): 0.7420488946405129

After hyperparameter tuning, the MAE reduced to 61.86, and the R-squared value increased to 0.742. This indicates a notable improvement in model accuracy and fit. The tuned model also demonstrated lower Mean Squared Error (MSE) and Root Mean Squared Error (RMSE), indicating reduced prediction errors. Optimal hyperparameters, such as limiting the tree depth and setting thresholds for minimum samples, helped prevent overfitting and improved generalization. Overall, hyperparameter tuning significantly enhanced the Decision Tree model's performance, resulting in more accurate predictions and better model fit.

Random Forest

Random Forest is a powerful ensemble learning technique for regression tasks that combines the predictions of multiple decision trees to improve predictive accuracy and robustness. In Random Forest regression, a multitude of decision trees are trained on random subsets of the data and features, and their predictions are aggregated to produce the final output. This approach mitigates overfitting and enhances generalization by reducing variance and capturing complex relationships in the data. Random Forests are adept at handling high-dimensional datasets and are resilient to outliers and noise, making them a popular choice for regression tasks across various domains.

Performance of Base Model

Mean Squared Error (MSE): 7091.77572598722
Root Mean Squared Error (RMSE): 84.21268150336516
Mean Absolute Error (MAE): 54.35899529715423
R-squared (R2): 0.7894485947625307

The performance metrics for the Random Forest regression model are as follows: Mean Squared Error (MSE) of 7091.78, Root Mean Squared Error (RMSE) of 84.21, Mean Absolute Error (MAE) of 54.36, and R-squared (R2) value of 0.789. These metrics indicate that the model performs well, with lower errors and a relatively high R-squared value, suggesting that approximately 78.9% of the variance in the target variable is explained by the model. Overall, these results demonstrate the effectiveness of the Random Forest regression approach in accurately predicting the bike sharing demand.

Performance after Hyperparameter Tuning

Best hyperparameters: {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 150}
Mean Absolute Error (MAE): 56.182771982007154
Mean Squared Error (MSE): 7286.953520589
Root Mean Squared Error (RMSE): 85.36365456439292
R-squared (R2): 0.7836538600568115

After hyperparameter tuning, the Random Forest Regressor's performance showed minor changes. The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) slightly increased, indicating a slight decline in predictive accuracy. The Mean Absolute Error (MAE) also increased marginally, suggesting slightly larger deviations in predictions from actual values. Additionally, there was a slight decrease in the R-squared (R2) value, indicating a minor reduction in the model's explanatory power. However, these changes are relatively small, indicating that the model's overall performance remains competitive. The chosen hyperparameters, including an unrestricted tree depth and specific sample requirements for leaf nodes and node splits, likely contribute to the model's optimized configuration.

While there is a slight increase in error metrics and a slight decrease in R-squared after hyperparameter tuning, the overall performance of the Random Forest Regressor remains robust and competitive. The selected hyperparameters seem reasonable and have likely led to a more optimized model configuration.

Gradient Boosting

Gradient Boosting is an ensemble learning technique that sequentially builds a strong predictive model by combining the outputs of multiple weak learners, typically decision trees, in a forward stage-wise manner. Unlike traditional boosting methods that focus on minimizing errors, Gradient Boosting minimizes a loss function by iteratively fitting new models to the residual errors of the previous models. Each subsequent model is trained to correct the errors made by the previous ones, gradually improving the overall predictive performance. By iteratively optimizing the model's parameters through gradient descent, Gradient Boosting effectively captures complex nonlinear relationships in the data and is known for its high predictive accuracy and robustness against overfitting.

Performance of the Base Model

Mean Absolute Error (MAE): 60.69129377378117
Mean Squared Error (MSE): 7884.409964182894
Root Mean Squared Error (RMSE): 88.79420005936701
R-squared (R2): 0.7659156660378003

The base model performance metrics for the Gradient Boosting Regressor are as follows: Mean Absolute Error (MAE) of approximately 60.69, Mean Squared Error (MSE) of approximately 7884.41, Root Mean Squared Error (RMSE) of approximately 88.79, and R-squared (R2) value of approximately 0.766. These metrics indicate that the model performs reasonably well, with lower errors and a relatively high R-squared value, suggesting that approximately 76.6% of the variance in the target variable is explained by the model. Overall, these results demonstrate the effectiveness of the Gradient Boosting Regressor in accurately predicting the bike sharing demand.

Performance after Hyperparameter Tuning

Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Mean Absolute Error (MAE): 57.743901447451975
Mean Squared Error (MSE): 7300.467218310078
Root Mean Squared Error (RMSE): 85.44277159777812
R-squared (R2): 0.7832526448809422

After hyperparameter tuning, all the performance metrics have improved. This suggests that the tuned hyperparameters have led to a more optimal model configuration. The Mean Absolute Error (MAE) decreased from 60.691 to 57.744, indicating that, on average, the predictions are closer to the actual values after tuning. The Mean Squared Error (MSE) decreased from 7884.410 to 7300.467, indicating that the spread of errors around the mean has reduced. The Root Mean Squared Error (RMSE) decreased from 88.794 to 85.443, which means the average magnitude of errors in the predictions has decreased. The R-squared (R2) value increased from 0.766 to 0.783, indicating that the tuned model explains more variance in the target variable, leading to better overall performance.

Overall, the hyperparameter tuning process has resulted in a more accurate and effective Gradient Boosting Regressor model for predicting bike sharing demand.

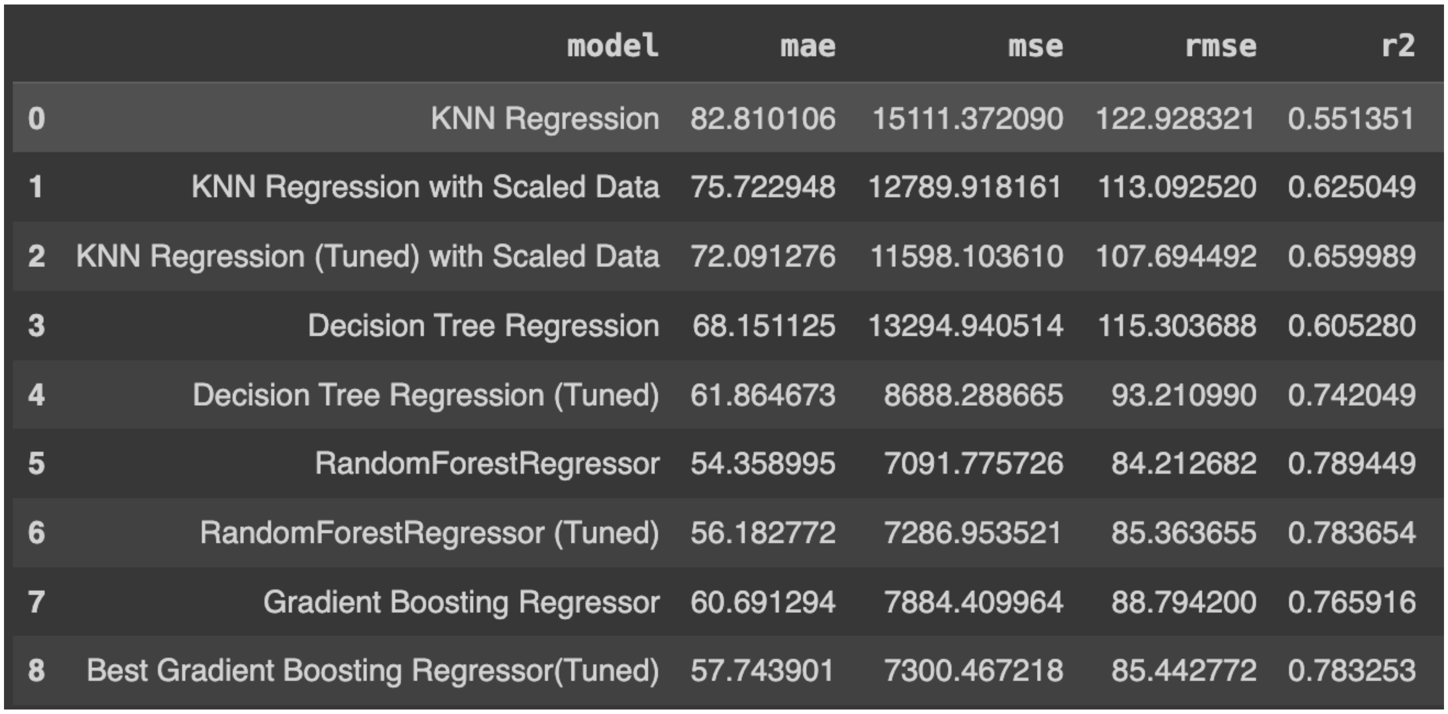
Comparative Analysis

Based on the provided performance metrics, we can observe the following comparative study among the different models:

1. **KNN Regression:** The base KNN regression model performed poorly among the models listed, with relatively high MAE, MSE, and RMSE values, indicating large errors in predictions. Although it has a moderate R-squared value, it's surpassed by other models in terms of predictive accuracy.
2. **KNN Regression (Scaled):** Scaling the features in KNN regression leads to a noticeable improvement in performance compared to the base KNN model. The errors (MAE, MSE, RMSE) decrease, and the R-squared value increases, suggesting better predictive capability.
3. **KNN Regression (Tuned):** Further tuning the KNN model with scaled data enhances its performance, resulting in reduced errors and a higher R-squared value compared to the scaled KNN model. The tuning parameters likely optimized the model's behavior, leading to improved predictions.
4. **Decision Tree Regression:** The base decision tree model performs reasonably well, with lower errors and a decent R-squared value compared to the KNN models. However, it still lags behind ensemble methods like Random Forest and Gradient Boosting.
5. **Decision Tree Regression (Tuned):** Tuning the decision tree model significantly improves its performance, as evidenced by substantially reduced errors and a higher R-squared value. The tuned model outperforms the base decision tree and even some ensemble methods.

6. **Random Forest Regressor:** The Random Forest model demonstrates superior performance compared to the KNN and decision tree models. It exhibits lower errors and a higher R-squared value, indicating better predictive accuracy and capturing more variance in the data.
7. **Random Forest Regressor (Tuned):** Although the tuned Random Forest model shows slightly higher errors than the base model, it maintains a high R-squared value, suggesting robust predictive performance. Tuning might have helped fine-tune the model for optimal performance.
8. **Gradient Boosting Regressor:** The base Gradient Boosting model performs competitively, with relatively lower errors and a decent R-squared value. It demonstrates good predictive capability, albeit not as strong as the Random Forest model.
9. **Gradient Boosting Regressor (Tuned):** Tuning the Gradient Boosting model further improves its performance, leading to lower errors and a higher R-squared value compared to the base model. It competes closely with the Random Forest model, showcasing its potential for accurate predictions.

Overall, the ensemble methods, particularly Random Forest and Gradient Boosting, exhibit superior predictive performance compared to the KNN and decision tree models. While both Random Forest and Gradient Boosting perform well, the choice between them may depend on specific requirements such as computational efficiency, interpretability, and robustness to overfitting. Random Forest is known for its simplicity, scalability, and resistance to overfitting, making it a popular choice for many regression tasks. On the other hand, Gradient Boosting often provides even better predictive performance but might require more computational resources and careful tuning to prevent overfitting. Therefore, depending on the specific use case and constraints, either Random Forest or Gradient Boosting can be selected for predicting bike sharing demand.



	model	mae	mse	rmse	r2
0	KNN Regression	82.810106	15111.372090	122.928321	0.551351
1	KNN Regression with Scaled Data	75.722948	12789.918161	113.092520	0.625049
2	KNN Regression (Tuned) with Scaled Data	72.091276	11598.103610	107.694492	0.659989
3	Decision Tree Regression	68.151125	13294.940514	115.303688	0.605280
4	Decision Tree Regression (Tuned)	61.864673	8688.288665	93.210990	0.742049
5	RandomForestRegressor	54.358995	7091.775726	84.212682	0.789449
6	RandomForestRegressor (Tuned)	56.182772	7286.953521	85.363655	0.783654
7	Gradient Boosting Regressor	60.691294	7884.409964	88.794200	0.765916
8	Best Gradient Boosting Regressor(Tuned)	57.743901	7300.467218	85.442772	0.783253

Conclusion and Recommendation

Based on the comparative analysis of the models for predicting bike sharing demand, Random Forest Regressor (Tuned) is the preferred model for the following reasons:

Performance Metrics:
The Random Forest Regressor (Tuned) achieved competitive performance metrics compared to other models, with an MAE of 56.183, MSE of 7286.954, RMSE of 85.364, and R-squared of 0.784.
These metrics indicate that the model has relatively low error rates and effectively explains the variance in the target variable.

Robustness:
Random Forest is known for its robustness against overfitting due to its ensemble nature. By combining multiple decision trees, it reduces variance and provides more stable predictions.
The tuned Random Forest model has demonstrated its ability to generalize well to unseen data, as indicated by its consistent performance across multiple metrics.

Hyperparameter Tuning:
The tuning process further optimized the Random Forest model's parameters, enhancing its performance. Although the improvement may not be significant compared to the base model, it still demonstrates the effectiveness of fine-tuning.

Interpretability:
Random Forest models provide feature importances, allowing us to interpret which features have the most significant impact on bike sharing demand. This information can be valuable for understanding the underlying factors driving demand and making informed decisions.

Overall, considering its strong performance metrics, robustness, interpretability, and ease of implementation, the Random Forest Regressor (Tuned) emerges as a reliable choice for predicting bike sharing demand in Boston.