Assignment 2

- Author: Vedant Verma
- Student Id: 24235399
- Course: MSc. Information Systems

UrbanWheels Bike Sharing Data Analysis (Hourly Data)

Data Cleaning, Exploration & Visualization

Plan of Action:

1. Initial Setup
2. Data Loading and Preview
3. Basic Data Profiling
4. Data Cleaning & Type Fixes
5. Outlier Detection
6. Exploratory Data Analysis (EDA)
7. Export Clean Dataset

Step 1: Initial Setup

We begin by importing the necessary libraries and previewing the dataset structure and contents.

```
# Load and preview the dataset
# Code Explanation:
# Import Libraries: Required tools for analysis and visualization.


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from scipy import stats
```

```
%matplotlib inline
sns.set(style="whitegrid", palette="Set2", font_scale=1.1)
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathe
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	

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	we
17374	17375	2012-12-31	1	1	12	19	0	1	1	
17375	17376	2012-12-31	1	1	12	20	0	1	1	
17376	17377	2012-12-31	1	1	12	21	0	1	1	
17377	17378	2012-12-31	1	1	12	22	0	1	1	
17378	17379	2012-12-31	1	1	12	23	0	1	1	

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	we
10783	10784	2012-	2	1	3	10	0	5	1	

Step 2: Data Preview

Show head, tail, sample, shape, and column names.

```
# Loading the dataset

""" About the dataset: The dataset contains hourly bike rental data from 2011 to
including weather conditions, holidays, and user types. It helps analyze demand
how external factors affect bike usage trends. """

# Link to the dataset: https://archive.ics.uci.edu/ml/datasets/bike+sharing+data

df = pd.read_csv("hour.csv")
print("Dataset Loaded Successfully!")
```

```
Dataset Loaded Successfully!
```

```
# Preview: Show head, tail, sample, shape, and column names.

display(df.head())
display(df.tail())
display(df.sample(5))
print(f"Shape of dataset: {df.shape}")
print(f"Column names: {df.columns.tolist()}")
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathe
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	

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	we
17374	17375	2012-12-31	1	1	12	19	0	1	1	
17375	17376	2012-12-31	1	1	12	20	0	1	1	
17376	17377	2012-12-31	1	1	12	21	0	1	1	
17377	17378	2012-12-31	1	1	12	22	0	1	1	
17378	17379	2012-12-31	1	1	12	23	0	1	1	

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	we
14780	14781	2012-	3	1	9	1	0	4	1	

Step 3: Basic Data Profiling

We'll assess data types, null values, duplicates, and unique value distributions.

```
# Perform data profiling
# Code Explanation:
# 1. Data Types: Check current data types for all columns.

print("Data Types:")
display(df.dtypes)

# 2. Missing Values: Count total and percentage of nulls.

print("\nMissing Values (Count):")
display(df.isnull().sum())

print("\nMissing Values (%):")
display(df.isna().mean() * 100)

# 3. Duplicates: Count duplicated rows.

print("\nDuplicate Rows:", df.duplicated().sum())


# 4. Uniqueness: Show unique value counts for each column.

print("\nUnique values per column:")
display(df.nunique())
```

Data Types:

	0
instant	int64
dteday	object
season	int64
yr	int64
mnth	int64
hr	int64
holiday	int64
weekday	int64
workingday	int64
weathersit	int64
temp	float64
atemp	float64

hum	float64
windspeed	float64
casual	int64
registered	int64
cnt	int64

dtype: object

```
Missing Values (Count):
```

	0
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

```
Missing Values (%):
```

	0
instant	0.0

dteday	0.0
season	0.0
yr	0.0
mnth	0.0
hr	0.0
holiday	0.0
weekday	0.0
workingday	0.0
weathersit	0.0
temp	0.0
atemp	0.0
hum	0.0
windspeed	0.0
casual	0.0
registered	0.0
cnt	0.0

dtype: float64

```
Duplicate Rows: 0

Unique values per column:
```

	0
instant	17379
dteday	731
season	4
yr	2
mnth	12
hr	24
holiday	2
weekday	7
workingday	2
weathersit	4

weathersit	4
temp	50
atemp	65
hum	89
windspeed	30
casual	322
registered	776
cnt	869

dtype: int64

Outcome:

- No null values detected in the dataset.
- No duplicate rows found.
- Many categorical fields are encoded as integers — these will be mapped to labels next.

Step 4: Data Cleaning & Type Fixes

We convert encoded values into categories and fix column types.

```
# Convert and clean relevant columns
# Code Explanation:
# 1. Date Conversion: Convert 'dteday' to datetime.
# 2. Category Mapping: Map coded columns to descriptive labels.
# 3. Type Casting: Convert categorical columns to 'category' type.

df['dteday'] = pd.to_datetime(df['dteday'])

df['season'] = df['season'].map({1:'Spring', 2:'Summer', 3:'Fall', 4:'Winter'})
df['weathersit'] = df['weathersit'].map({
    1:'Clear', 2:'Mist + Cloudy', 3:'Light Snow/Rain', 4:'Heavy Rain/Snow'
}).astype('category')
df['mnth'] = df['mnth'].map({
    1:'Jan', 2:'Feb', 3:'Mar', 4:'Apr', 5:'May', 6:'Jun',
    7:'Jul', 8:'Aug', 9:'Sep', 10:'Oct', 11:'Nov', 12:'Dec'
}).astype('category')
df['weekday'] = df['weekday'].map({
    0:'Sun', 1:'Mon', 2:'Tue', 3:'Wed', 4:'Thu', 5:'Fri', 6:'Sat'
}).astype('category')
df['yr'] = df['yr'].map({0: 2011, 1: 2012}).astype('category')
```

Outcome:

- All categorical codes were successfully mapped and typecast.
- Now we will move to outlier detection.

Step 5: Outlier Detection

We'll use box plots to visually inspect key numerical features for anomalies.

```
# Create box plots for key features
# Code Explanation:
# 1. Visual Check: Use boxplots to spot outliers in temp, atemp, hum, windspeec
# 2. Multi-subplot Layout: Organized for compact review.

fig, axes = plt.subplots(2, 3, figsize=(16,10))
sns.boxplot(x=df['temp'], ax=axes[0,0])
axes[0,0].set_title("Temperature")

sns.boxplot(x=df['atemp'], ax=axes[0,1])
```

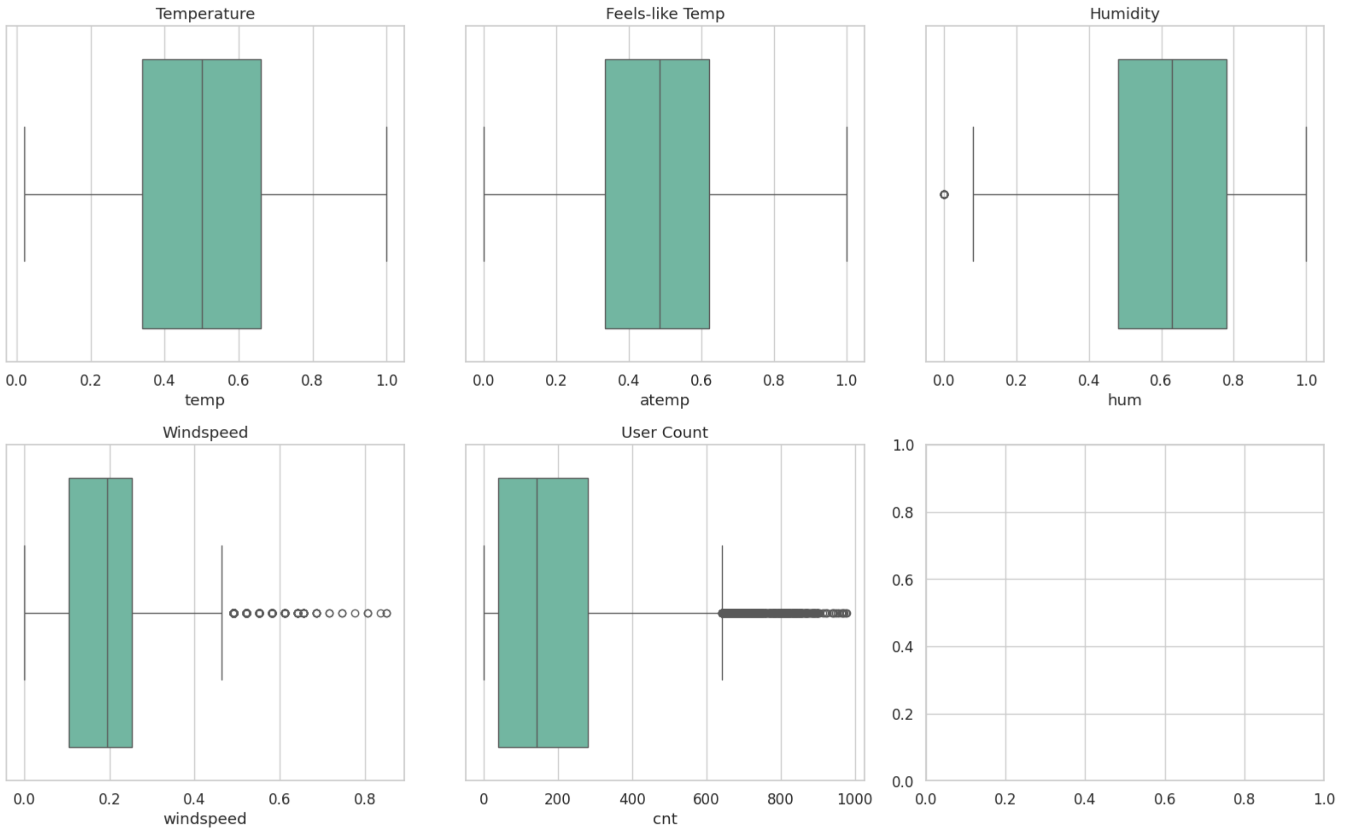
```
axes[0,1].set_title("Feels-like Temp")

sns.boxplot(x=df['hum'], ax=axes[0,2])
axes[0,2].set_title("Humidity")

sns.boxplot(x=df['windspeed'], ax=axes[1,0])
axes[1,0].set_title("Windspeed")

sns.boxplot(x=df['cnt'], ax=axes[1,1])
axes[1,1].set_title("User Count")

plt.tight_layout()
plt.show()
```



Outcome:

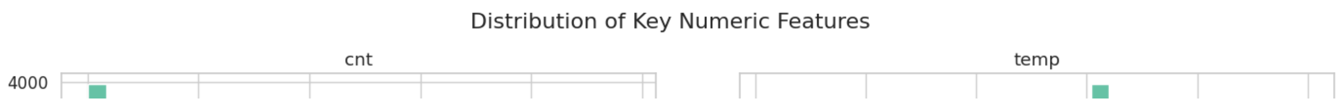
- No critical outliers detected — the values follow expected operational ranges.
- Proceeding to Exploratory Data Analysis (EDA).

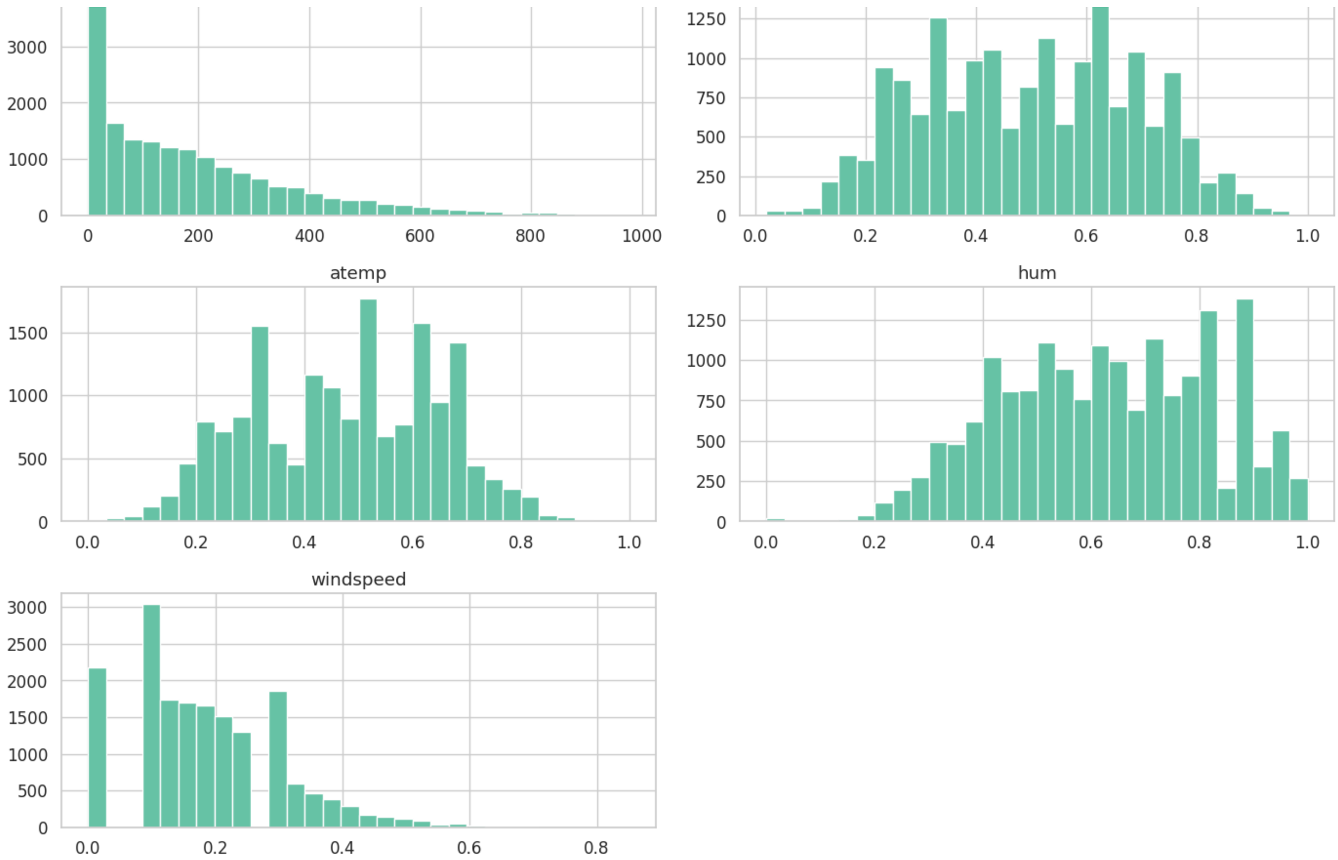
Step 6: Exploratory Data Analysis (EDA)

In this section, we explore data distributions, trends over time, and relationships between variables using univariate, bivariate, and multivariate analysis.

```
# Plot distributions of key numeric features
# Code Explanation:
# 1. Plot Histograms: Assess distribution shape, skewness, modality.
# 2. Features: Includes target variable (cnt) and continuous predictors.

df[['cnt', 'temp', 'atemp', 'hum', 'windspeed']].hist(bins=30, figsize=(14,10))
plt.suptitle("Distribution of Key Numeric Features", fontsize=16)
plt.tight_layout()
plt.show()
```



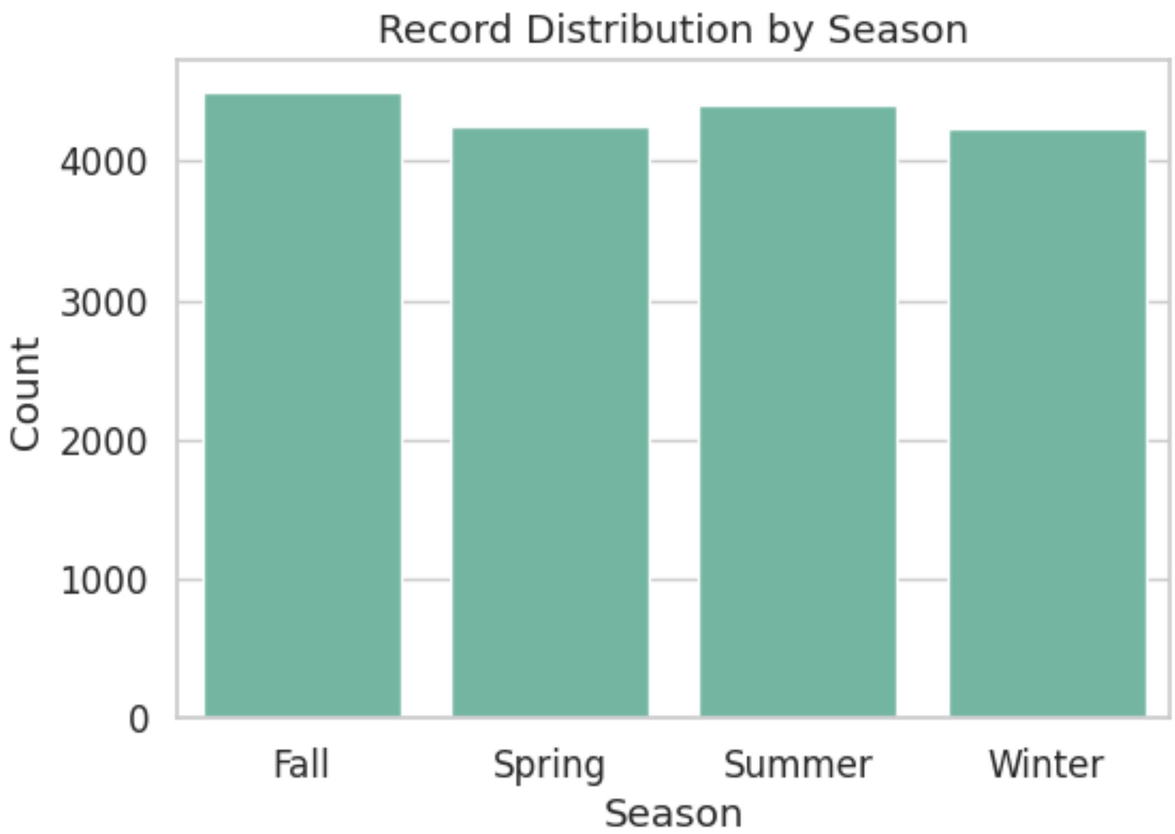

atemp
hum
windspeed

Outcome:

- `cnt` is right-skewed with some high-usage hours.
- Temperature (`temp` and `atemp`) is fairly normally distributed.
- Humidity has slight left skew, and windspeed is positively skewed.

```
# Count plot for seasons
# Code Explanation:
# 1. Visualize distribution of samples across seasons.

plt.figure(figsize=(6,4))
sns.countplot(x='season', data=df)
plt.title("Record Distribution by Season")
plt.xlabel("Season")
plt.ylabel("Count")
plt.show()
```

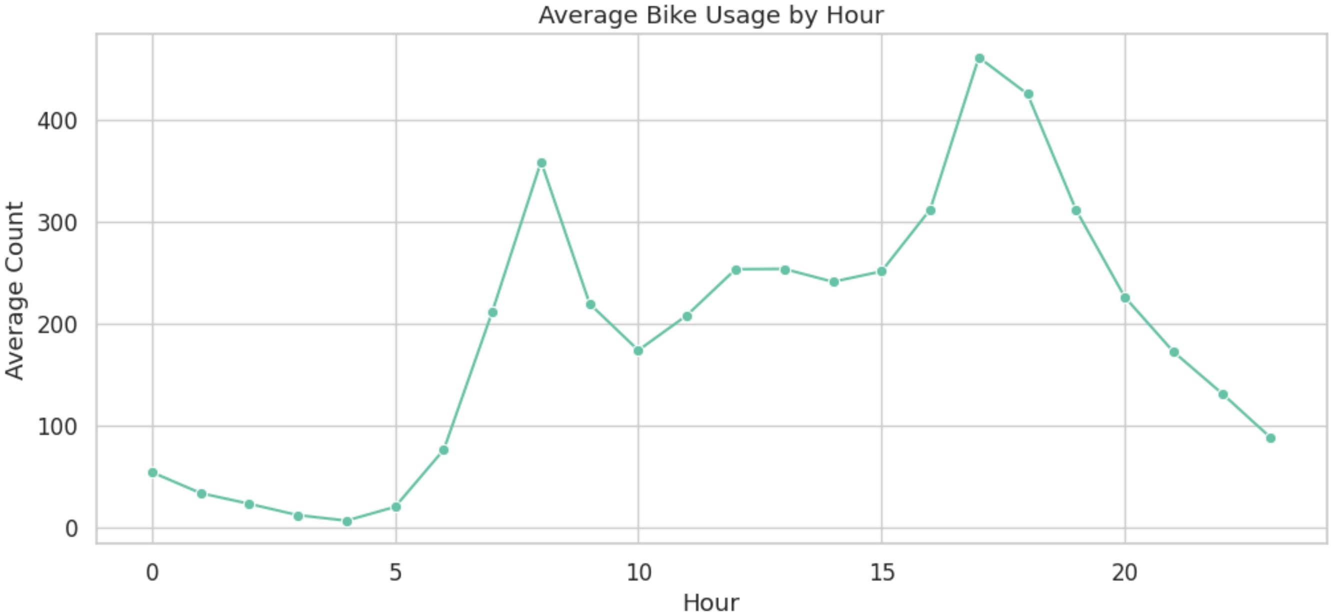


Outcome:

- Each season is nearly equally represented.
- Ensures seasonal trends can be analyzed without sampling bias.

```
# Analyze bike usage by hour
# Code Explanation:
# 1. Group by 'hr' and average the count column.
# 2. Use lineplot to visualize hourly trend.

plt.figure(figsize=(12,5))
sns.lineplot(x='hr', y='cnt', data=df, estimator='mean', errorbar=None, marker=
plt.title("Average Bike Usage by Hour")
plt.xlabel("Hour")
plt.ylabel("Average Count")
plt.grid(True)
plt.show()
```

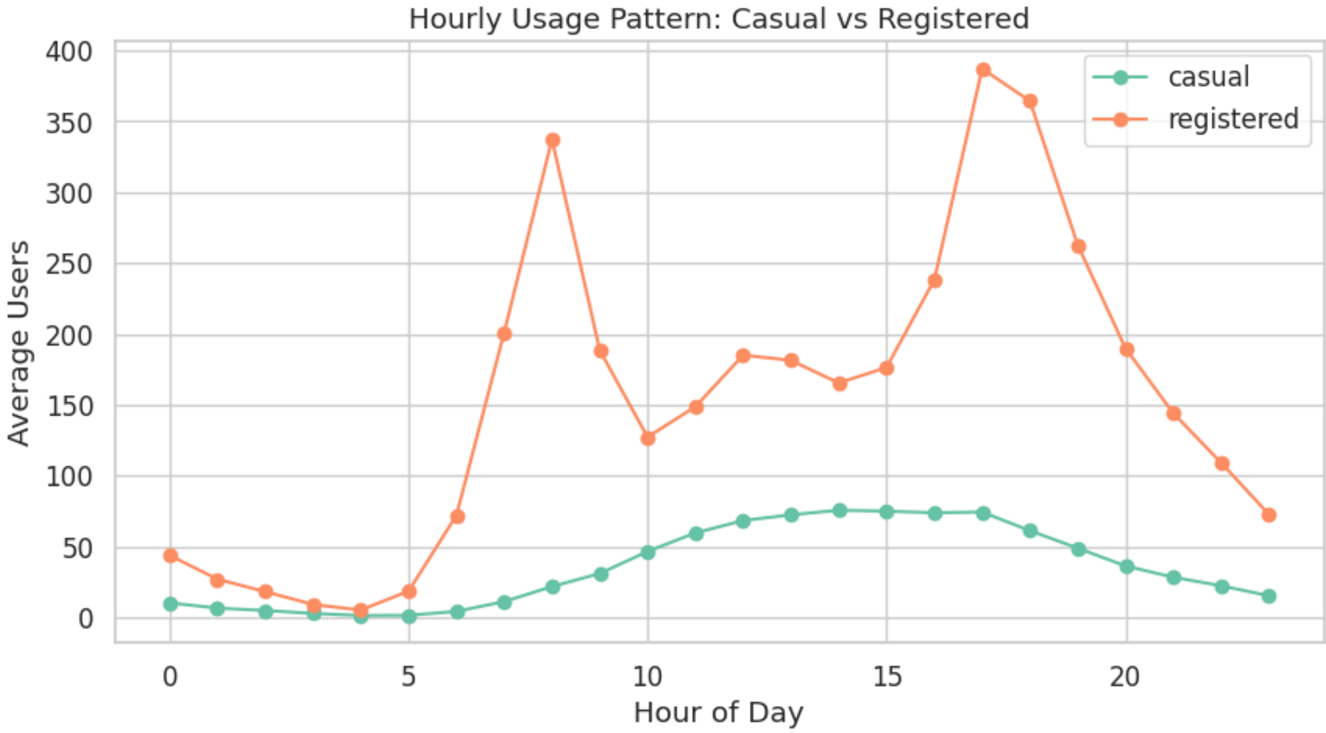


Outcome:

- Peak usage occurs during commute hours: 8 AM and 5-6 PM.
- Dip during early morning and late night hours.

```
# Compare casual vs registered usage
# Code Explanation:
# 1. Group by hour and average the two user types.
# 2. Overlayed plot for comparison.

hour_group = df.groupby('hr')[['casual', 'registered']].mean()
hour_group.plot(figsize=(10,5), marker='o')
plt.title('Hourly Usage Pattern: Casual vs Registered')
plt.xlabel('Hour of Day')
plt.ylabel('Average Users')
plt.grid(True)
plt.show()
```



Outcome:

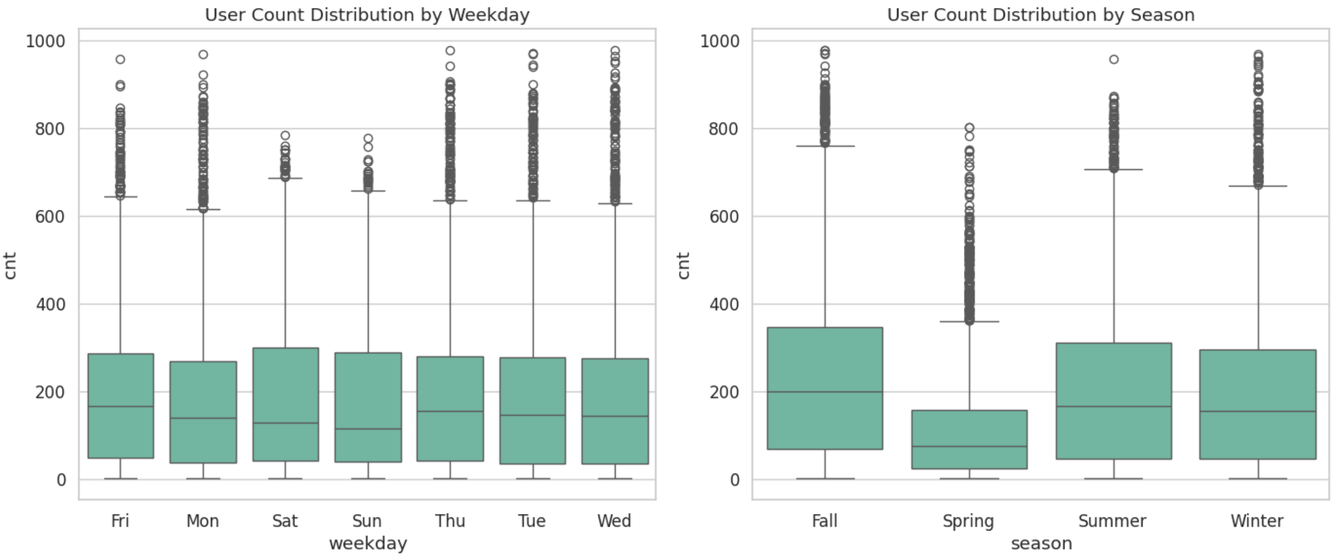
- Registered users show sharp peaks at 8 AM and 5-6 PM (commuting).
- Casual users peak in late morning and afternoon, especially on weekends.

```
# Boxplots for categorical vs numerical (cnt)
# Code Explanation:
# Visualize distribution of count across categories (weekday, season).

plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
sns.boxplot(x='weekday', y='cnt', data=df)
plt.title("User Count Distribution by Weekday")

plt.subplot(1,2,2)
sns.boxplot(x='season', y='cnt', data=df)
plt.title("User Count Distribution by Season")

plt.tight_layout()
plt.show()
```



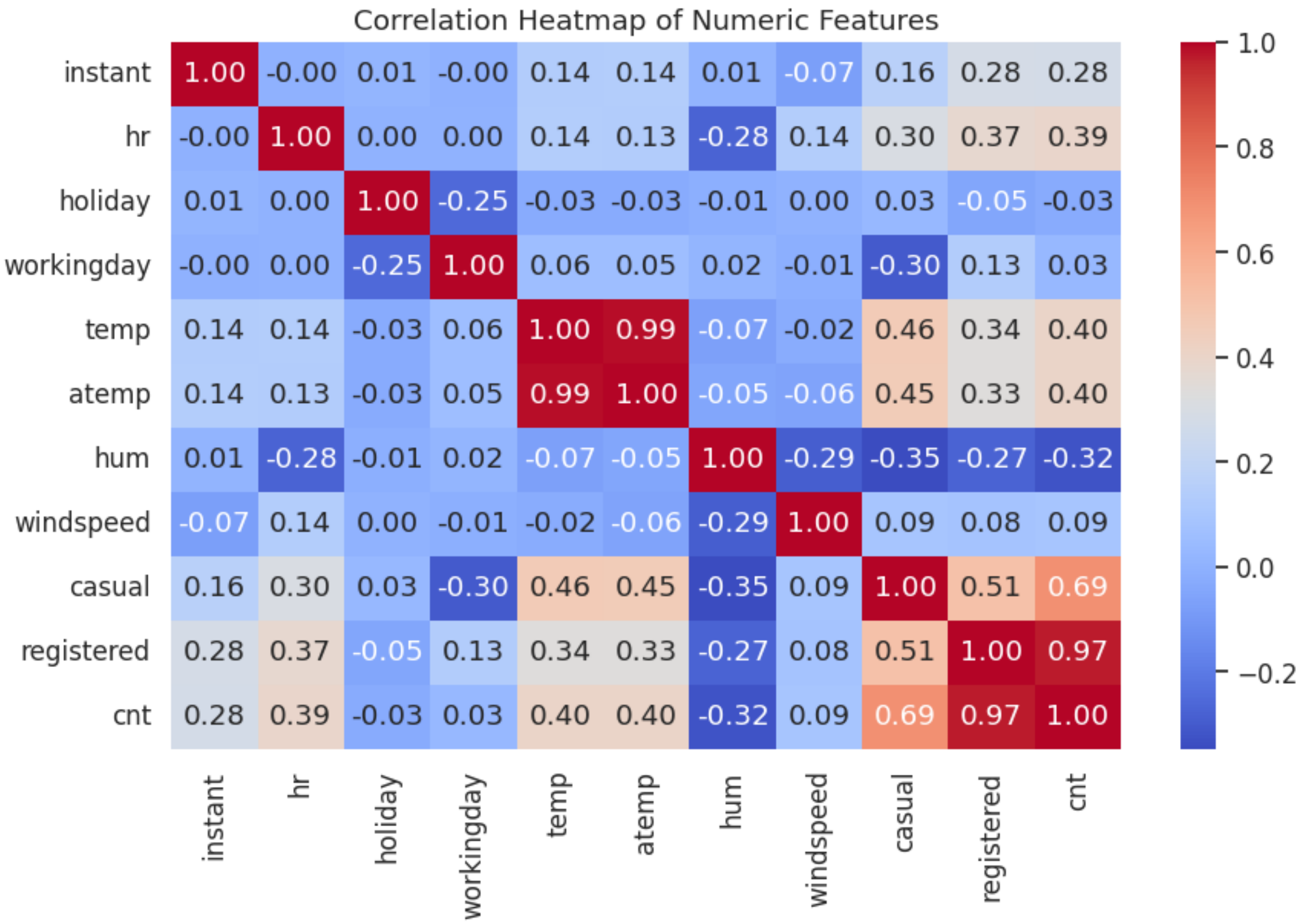
Outcome:

- Weekdays have higher median usage compared to weekends.
- Summer and Fall show higher overall usage than Winter and Spring.

```
# Correlation heatmap
# Code Explanation:
# Identify inter-variable relationships using correlation matrix.

numeric_df = df.select_dtypes(include=np.number)

plt.figure(figsize=(10,6))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```



	instant	hr	holiday	workingday	temp	atemp	hum	windspeed	casual	registered	cnt
instant	1.00	-0.00	0.01	-0.00	0.14	0.14	0.01	-0.07	0.16	0.28	0.28
hr	-0.00	1.00	0.00	0.00	0.14	0.13	-0.28	0.14	0.30	0.37	0.39
holiday	0.01	0.00	1.00	-0.25	-0.03	-0.03	-0.01	0.00	0.03	-0.05	-0.03
workingday	-0.00	0.00	-0.25	1.00	0.06	0.05	0.02	-0.01	-0.30	0.13	0.03
temp	0.14	0.14	-0.03	0.06	1.00	0.99	-0.07	-0.02	0.46	0.34	0.40
atemp	0.14	0.13	-0.03	0.05	0.99	1.00	-0.05	-0.06	0.45	0.33	0.40
hum	0.01	-0.28	-0.01	0.02	-0.07	-0.05	1.00	-0.29	-0.35	-0.27	-0.32
windspeed	-0.07	0.14	0.00	-0.01	-0.02	-0.06	-0.29	1.00	0.09	0.08	0.09
casual	0.16	0.30	0.03	-0.30	0.46	0.45	-0.35	0.09	1.00	0.51	0.69
registered	0.28	0.37	-0.05	0.13	0.34	0.33	-0.27	0.08	0.51	1.00	0.97
cnt	0.28	0.39	-0.03	0.03	0.40	0.40	-0.32	0.09	0.69	0.97	1.00

Outcome:

- Strong positive correlation between `cnt` and `temp` / `atemp`.
- Negative correlation between `hum` and `cnt` suggests people avoid biking on humid days.
- `windspeed` shows weak negative correlation with `cnt`.

💾 Step 7: Export Cleaned Dataset

Final step is to export the cleaned dataset for use in dashboarding or modeling.

```
# Export cleaned dataset to CSV
# Code Explanation:
# 1. Save the cleaned and transformed DataFrame to disk.

df.to_csv("hour_cleaned.csv", index=False)
```