Project: Linear Regression

Name: Melody Goldanloo

This Jupyter Notebook Starter File provides a basic outline for your solutions. For detailed instructions, please refer to the assignment on Canvas. Complete all your work for this project in this same Jupyter Notebook file, which you will submit:

- Code:
  - Insert your code where you see #Insert Code Here.
  - Ensure all code is well-commented and easy to understand.
  - Use clear and descriptive variable names.
- Questions:
  - You will be provided guided questions in a separate assignment vs. here in the code to give you the opportunity to demonstrate a deep understanding of the concepts through thorough explanations and critical thinking.

In [1]:
```
#Example of supress warnings for Numpy version out of range (optional)
import warnings
warnings.filterwarnings("ignore", category=Warning)
warnings.simplefilter(action='ignore', category=FutureWarning)

#Some recommended libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
import requests
from io import BytesIO
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score
```

The Dataset

In [2]:
```
# URL of the Bike Sharing Dataset zip file
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00275/Bike-

# Fetch the zip file from the URL
response = requests.get(url)
```

```
zip_file = zipfile.ZipFile(BytesIO(response.content))

# List files in the zip
print(zip_file.namelist())

# Load the day.csv file into a DataFrame
with zip_file.open('day.csv') as file:
    df = pd.read_csv(file)
    print(df.head())
```

```
['Readme.txt', 'day.csv', 'hour.csv']
   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
0   985
1   801
2  1349
3  1562
4  1600
```

Data Preprocessing

Loading Data: Load the Bike Sharing dataset using Pandas.

Data Exploration and Visualization: Produce some visualizations, statistics, etc. to gain an understanding for the dataset and what it contains.

Handling Missing Values: Check for and handle any missing values in the dataset (if any).

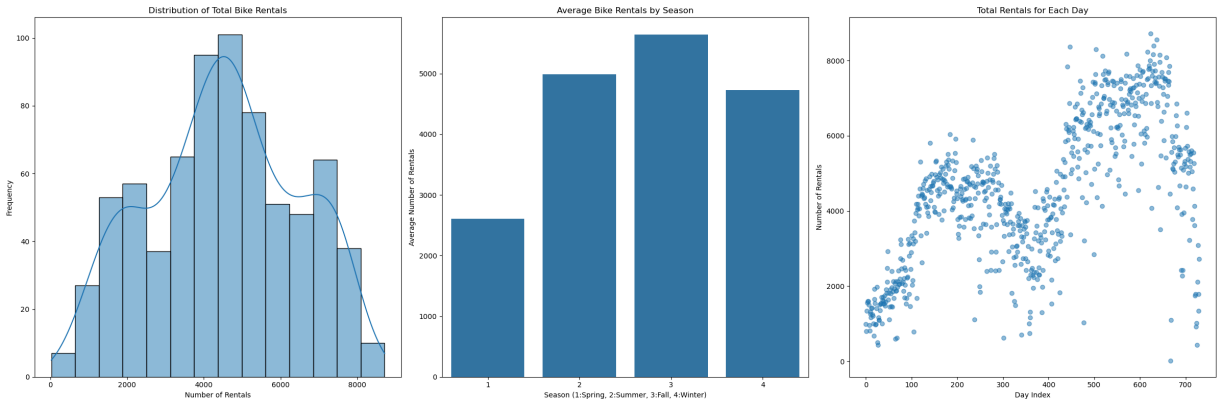
Encoding Categorical Variables: Convert categorical variables to numerical values using techniques like one-hot encoding (if any you see that need this).

Feature Engineering: Create new features from the date column (e.g., day of the week, month, hour) to capture temporal patterns.

Standardization: Standardize the features to have a mean of 0 and a standard deviation of 1 for consistent training. This step helps ensure that the model is not biased towards features with larger scales.

Train-Test Split: Split the dataset into training and testing sets to evaluate the model's performance on unseen data.

In [3]:
```
df = pd.read_csv('day.csv')


# Data Preprocessing
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24, 8))

sns.histplot(df['cnt'], kde=True, ax=ax1)
ax1.set_title('Distribution of Total Bike Rentals')
ax1.set_xlabel('Number of Rentals')
ax1.set_ylabel('Frequency')

season_avg = df.groupby('season')['cnt'].mean().reset_index()
sns.barplot(x='season', y='cnt', data=season_avg, ax=ax2)
ax2.set_title('Average Bike Rentals by Season')
ax2.set_xlabel('Season (1:Spring, 2:Summer, 3:Fall, 4:Winter)')
ax2.set_ylabel('Average Number of Rentals')

ax3.scatter(df.index, df['cnt'], alpha=0.5)
ax3.set_title('Total Rentals for Each Day')
ax3.set_xlabel('Day Index')
ax3.set_ylabel('Number of Rentals')

plt.tight_layout()
plt.show()


df.describe(), df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   instant     731 non-null    int64  
 1   dteday      731 non-null    object 
 2   season      731 non-null    int64  
 3   yr          731 non-null    int64  
 4   mnth        731 non-null    int64  
 5   holiday     731 non-null    int64  
 6   weekday     731 non-null    int64  
 7   workingday  731 non-null    int64  
 8   weathersit  731 non-null    int64  
 9   temp        731 non-null    float64
 10  atemp       731 non-null    float64
 11  hum         731 non-null    float64
 12  windspeed   731 non-null    float64
 13  casual      731 non-null    int64  
 14  registered  731 non-null    int64  
 15  cnt         731 non-null    int64  
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

Out[3]:
```
(          instant      season          yr        mnth     holiday     week
day  \
 count  731.000000  731.000000  731.000000  731.000000  731.000000  731.000
000
 mean   366.000000    2.496580    0.500684    6.519836    0.028728    2.997
264
 std    211.165812    1.110807    0.500342    3.451913    0.167155    2.004
787
 min      1.000000    1.000000    0.000000    1.000000    0.000000    0.000
000
 25%    183.500000    2.000000    0.000000    4.000000    0.000000    1.000
000
 50%    366.000000    3.000000    1.000000    7.000000    0.000000    3.000
000
 75%    548.500000    3.000000    1.000000   10.000000    0.000000    5.000
000
 max    731.000000    4.000000    1.000000   12.000000    1.000000    6.000
000

        workingday  weathersit        temp       atemp         hum   windsp
eed  \
 count  731.000000  731.000000  731.000000  731.000000  731.000000  731.000
000
 mean     0.683995    1.395349    0.495385    0.474354    0.627894    0.190
486
 std      0.465233    0.544894    0.183051    0.162961    0.142429    0.077
498
 min      0.000000    1.000000    0.059130    0.079070    0.000000    0.022
392
 25%      0.000000    1.000000    0.337083    0.337842    0.520000    0.134
950
 50%      1.000000    1.000000    0.498333    0.486733    0.626667    0.180
975
 75%      1.000000    2.000000    0.655417    0.608602    0.730209    0.233
214
 max      1.000000    3.000000    0.861667    0.840896    0.972500    0.507
463

             casual   registered          cnt
 count   731.000000   731.000000   731.000000
 mean    848.176471  3656.172367  4504.348837
 std     686.622488  1560.256377  1937.211452
 min       2.000000    20.000000    22.000000
 25%     315.500000  2497.000000  3152.000000
 50%     713.000000  3662.000000  4548.000000
 75%    1096.000000  4776.500000  5956.000000
 max    3410.000000  6946.000000  8714.000000  ,
 None)
```

In [4]:
```
# Checking for missing values
df.isnull().sum()
# no missing values

# I don't think it would make sense to encode the categorical variable (dtec

# Feature Engineering
```

```
df_processed = df.copy()
df_processed['dteday'] = pd.to_datetime(df_processed['dteday'])


df_processed['day_of_week'] = df_processed['dteday'].dt.dayofweek
df_processed['day_of_month'] = df_processed['dteday'].dt.day

# Drop the original 'dteday' column
df_processed = df_processed.drop('dteday', axis=1)

# Display the first few rows to verify the changes
print(df_processed.head())
print(df_processed.info())
```

```
   instant  season  yr  mnth  holiday  weekday  workingday  weathersit  \
0        1       1   0     1        0        6           0           2
1        2       1   0     1        0        0           0           2
2        3       1   0     1        0        1           1           1
3        4       1   0     1        0        2           1           1
4        5       1   0     1        0        3           1           1

       temp     atemp       hum  windspeed  casual  registered   cnt  \
0  0.344167  0.363625  0.805833   0.160446     331         654   985
1  0.363478  0.353739  0.696087   0.248539     131         670   801
2  0.196364  0.189405  0.437273   0.248309     120        1229  1349
3  0.200000  0.212122  0.590435   0.160296     108        1454  1562
4  0.226957  0.229270  0.436957   0.186900      82        1518  1600

   day_of_week  day_of_month
0            5             1
1            6             2
2            0             3
3            1             4
4            2             5
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   instant        731 non-null    int64
 1   season         731 non-null    int64
 2   yr             731 non-null    int64
 3   mnth           731 non-null    int64
 4   holiday        731 non-null    int64
 5   weekday        731 non-null    int64
 6   workingday     731 non-null    int64
 7   weathersit     731 non-null    int64
 8   temp           731 non-null    float64
 9   atemp          731 non-null    float64
 10  hum            731 non-null    float64
 11  windspeed      731 non-null    float64
 12  casual         731 non-null    int64
 13  registered     731 non-null    int64
 14  cnt            731 non-null    int64
 15  day_of_week    731 non-null    int32
 16  day_of_month   731 non-null    int32
dtypes: float64(4), int32(2), int64(11)
memory usage: 91.5 KB
None
```

In [5]:
```
df_standardized = df_processed.copy()

scaler = StandardScaler()

df_standardized[['cnt_standardized', 'registered_standardized']] = scaler.fi

print(df_standardized.head())
print(df_standardized.info())
print(df_standardized[['cnt_standardized', 'registered_standardized']].descr
```

```
   instant  season  yr  mnth  holiday  weekday  workingday  weathersit  \
0        1       1   0     1        0        6           0           2
1        2       1   0     1        0        0           0           2
2        3       1   0     1        0        1           1           1
3        4       1   0     1        0        2           1           1
4        5       1   0     1        0        3           1           1

       temp     atemp       hum  windspeed  casual  registered   cnt  \
0  0.344167  0.363625  0.805833   0.160446     331         654   985
1  0.363478  0.353739  0.696087   0.248539     131         670   801
2  0.196364  0.189405  0.437273   0.248309     120        1229  1349
3  0.200000  0.212122  0.590435   0.160296     108        1454  1562
4  0.226957  0.229270  0.436957   0.186900      82        1518  1600

   day_of_week  day_of_month  cnt_standardized  registered_standardized
0            5             1         -1.817953                -1.925471
1            6             2         -1.912999                -1.915209
2            0             3         -1.629925                -1.556689
3            1             4         -1.519898                -1.412383
4            2             5         -1.500269                -1.371336
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   instant                 731 non-null    int64
 1   season                  731 non-null    int64
 2   yr                      731 non-null    int64
 3   mnth                    731 non-null    int64
 4   holiday                 731 non-null    int64
 5   weekday                 731 non-null    int64
 6   workingday              731 non-null    int64
 7   weathersit              731 non-null    int64
 8   temp                    731 non-null    float64
 9   atemp                   731 non-null    float64
 10  hum                     731 non-null    float64
 11  windspeed               731 non-null    float64
 12  casual                  731 non-null    int64
 13  registered              731 non-null    int64
 14  cnt                     731 non-null    int64
 15  day_of_week             731 non-null    int32
 16  day_of_month            731 non-null    int32
 17  cnt_standardized        731 non-null    float64
 18  registered_standardized  731 non-null    float64
dtypes: float64(6), int32(2), int64(11)
memory usage: 102.9 KB
None
       cnt_standardized  registered_standardized
count      7.310000e+02             7.310000e+02
mean      -1.166418e-16             7.776117e-17
std        1.000685e+00             1.000685e+00
min       -2.315399e+00            -2.332092e+00
25%       -6.985684e-01            -7.434458e-01
50%        2.254842e-02             3.737606e-03
75%        7.498640e-01             7.185324e-01
max        2.174535e+00             2.109961e+00
```

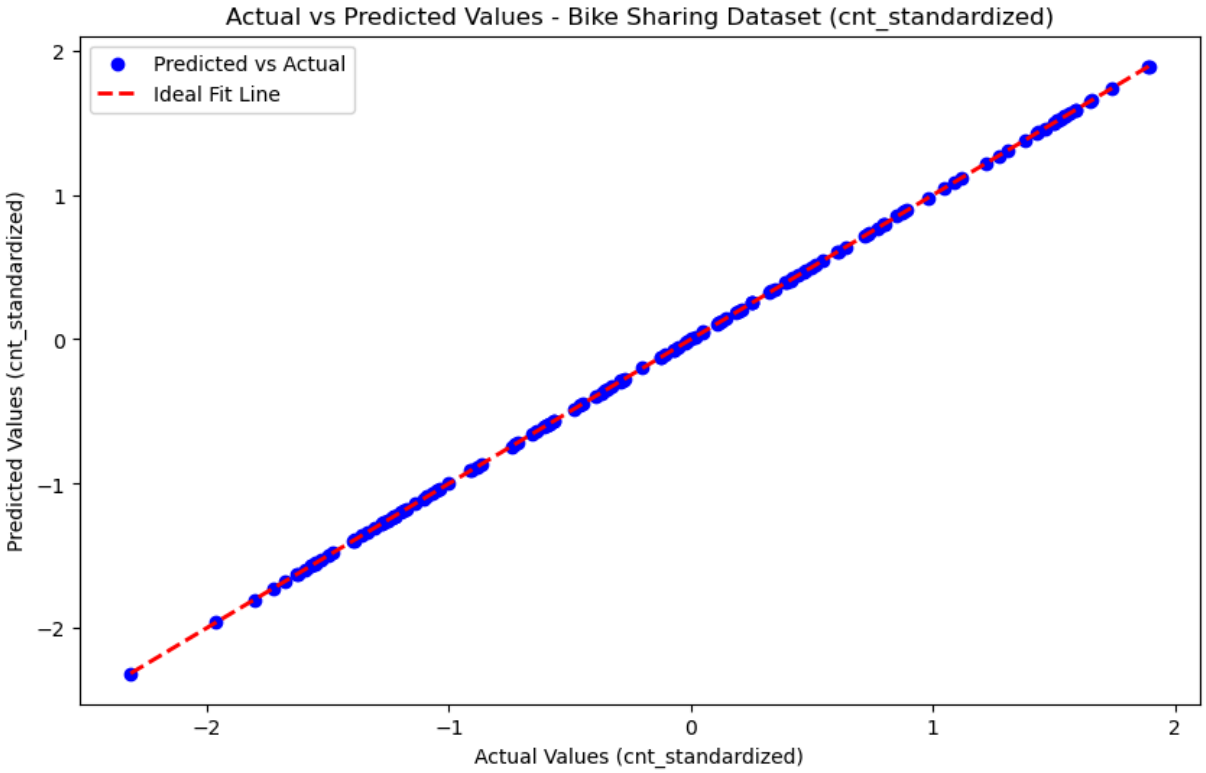
In [6]:
```
#Train-Test Split: Split the dataset into training and testing sets to evalu

X = df_standardized.drop(columns=['cnt_standardized', 'registered_standardiz
y = df_standardized[['cnt_standardized', 'registered_standardized']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

Building the Linear Regression Model

Model Initialization:

Choosing Hyperparameters: Explain key hyperparameters such as the fit_intercept and normalize. Linear Regression Initialization: Use the LinearRegression class from Scikit-learn, specifying the chosen hyperparameters. Model Fitting: Fit the Linear Regression model to the standardized Bike Sharing dataset.

In [7]:
```
model = LinearRegression()
model.fit(X_train, y_train)

#Make predictions on the testing set
y_hat = model.predict(X_test)

# Plotting actual vs. predicted values
y_test_single = y_test['cnt_standardized']
y_hat_single = y_hat[:, 0]  # Assuming 'cnt_standardized' is the first colum

# Plotting actual vs. predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test_single, y_hat_single, color='blue', label='Predicted vs A
plt.plot([y_test_single.min(), y_test_single.max()], [y_test_single.min(), y
plt.xlabel('Actual Values (cnt_standardized)')
plt.ylabel('Predicted Values (cnt_standardized)')
plt.title('Actual vs Predicted Values - Bike Sharing Dataset (cnt_standardiz
plt.legend()
plt.show()
```



Evaluating the Model

Performance Metrics: Calculate metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2) to evaluate the model. These metrics provide insights into the model's predictive performance and its ability to generalize.

In [8]:
```
mae = mean_absolute_error(y_test, y_hat)
mse = mean_squared_error(y_test, y_hat)
r2 = r2_score(y_test, y_hat)

print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {np.sqrt(mse)}")
print(f"R-squared: {r2}")
```

```
Mean Absolute Error: 6.855095493903137e-16
Mean Squared Error: 6.955860071299784e-31
Root Mean Squared Error: 8.34017989692056e-16
R-squared: 1.0
```

For the Model Selection Project, you will STOP HERE!

During Units 4, 5, and 6, we will explore and learn additional techniques, and then revisit these projects to apply the below:

- Model evaluation discussion and parameter tuning