

✓ Problem Statement

Predict the bike-sharing counts per hour based on the features including weather, day, time, humidity, wind speed, season e.t.c.

✓ Learning Objectives

At the end of the mini-project, you will be able to :

- perform data exploration and visualization
- implement linear regression using sklearn and optimization
- apply regularization on regression using Lasso, Ridge and Elasticnet techniques
- calculate and compare the MSE value of each regression technique
- analyze the features that are best contributing to the target

✓ Dataset

The dataset chosen for this mini-project is [Bike Sharing Dataset](#). This dataset contains the hourly and daily count of rental bikes between the years 2011 and 2012 in the capital bike share system with the corresponding weather and seasonal information. This dataset consists of 17389 instances of 16 features.

Bike sharing systems are a new generation of traditional bike rentals where the whole process from membership, rental and return has become automatic. Through these systems, the user can easily rent a bike from a particular position and return to another position. Currently, there are about over 500 bike-sharing programs around the world which is composed of over 500 thousand bicycles. Today, there exists great interest in these systems due to their important role in traffic, environmental and health issues.

Apart from interesting real world applications of bike sharing systems, the characteristics of data being generated by these systems make them attractive for the research. As opposed to other transport services such as bus or subway, the duration of travel, departure and arrival position are explicitly recorded in these systems. This feature turns bike sharing system into a virtual sensor network that can be used for sensing mobility in the city. Hence, it is expected that the most important events in the city could be detected via monitoring these data.



Data Fields

- dteday - hourly date
- season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
- hr - hour
- holiday - whether the day is considered a holiday
- workingday - whether the day is neither a weekend nor holiday
- weathersit -
 - 1 - Clear, Few clouds, Partly cloudy, Partly cloudy
 - 2 - Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
 - 3 - Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
 - 4 - Heavy Rain + Ice Pellets + Thunderstorm + Mist, Snow + Fog
- temp - temperature in Celsius
- atemp - "feels like" temperature in Celsius
- humidity - relative humidity
- windspeed - wind speed
- casual - number of non-registered user rentals initiated

- registered - number of registered user rentals initiated
- cnt - number of total rentals

Information

Regularization: It is a form of regression that shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, to avoid the risk of overfitting. A simple relation for linear regression looks like this.

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X).

If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

Below are the Regularization techniques:

- Ridge Regression
- Lasso Regression
- Elasticnet Regression

Grading = 10 Points

Download the dataset

```
#@title Download the dataset
!wget -qq https://cdn.iisc.talentsprint.com/CDS/MiniProjects/Bike_Sharing_Dataset.zip
!unzip Bike_Sharing_Dataset.zip
```

📁 Archive: Bike_Sharing_Dataset.zip
 inflating: Readme.txt
 inflating: day.csv
 inflating: hour.csv

Importing Necessary Packages

```
# Loading the Required Packages
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import r2_score
```

Data Loading

```
# Read the hour.csv file
# YOUR CODE HERE
df = pd.read_csv('hour.csv')
```

print the first five rows of dataset

```
# YOUR CODE HERE
print(df.head())
```

```
📄 instant      dteday  season  yr  mnth  hr  holiday  weekday  workingday  \
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

   weathersit  temp  atemp  hum  windspeed  casual  registered  cnt
0          1   0.24  0.2879  0.81         0.0         3         13   16
1          1   0.22  0.2727  0.80         0.0         8         32   40
2          1   0.22  0.2727  0.80         0.0         5         27   32
```

3	1	0.24	0.2879	0.75	0.0	3	10	13
4	1	0.24	0.2879	0.75	0.0	0	1	1

print the datatypes of the columns

```
# YOUR CODE HERE
print(df.dtypes)
```

```
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
```

Task flow with respect to feature processing and model training

- Explore and analyze the data
- Identify continuous features and categorical features
- Apply scaling on continuous features and one-hot encoding on categorical features
- Separate the features, targets and split the data into train and test
- Find the coefficients of the features using normal equation and find the cost (error)
- Apply batch gradient descent technique and find the best coefficients
- Apply SGD Regressor using sklearn
- Apply linear regression using sklearn
- Apply Lasso, Ridge, Elasticnet Regression

✓ EDA & Visualization (2 points)

✓ Visualize the hour (hr) column with an appropriate plot and find the busy hours of bike sharing

```
# YOUR CODE HERE
plt.figure(figsize=(10, 6))
sns.countplot(x='hr', data=df, palette='viridis')
plt.title('Distribution of Bike Sharing by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Bike Shares')
plt.xticks(range(0, 24)) # Ensure all hours from 0 to 23 are labeled
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

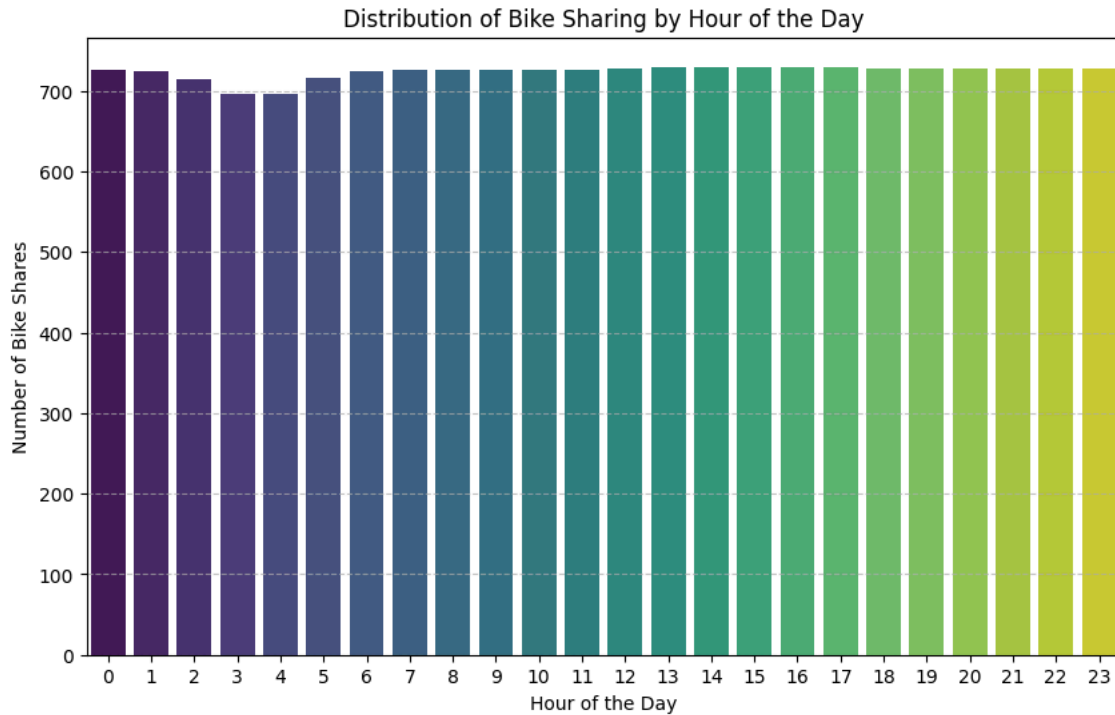
# Optional: Calculate the busiest hours by counting occurrences
busy_hours = df['hr'].value_counts().sort_values(ascending=False)

# Print the busiest hours
print("Busiest hours of bike sharing (sorted by count):")
print(busy_hours)
```

```
<ipython-input-6-e6b28808bcd6>:3: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

```
sns.countplot(x='hr', data=df, palette='viridis')
```



Busiest hours of bike sharing (sorted by count):

```
hr
17    730
16    730
13    729
15    729
14    729
12    728
22    728
21    728
20    728
19    728
18    728
23    728
 8    727
 7    727
11    727
 9    727
10    727
 0    726
 6    725
 1    724
 5    717
 2    715
 4    697
 3    697
```

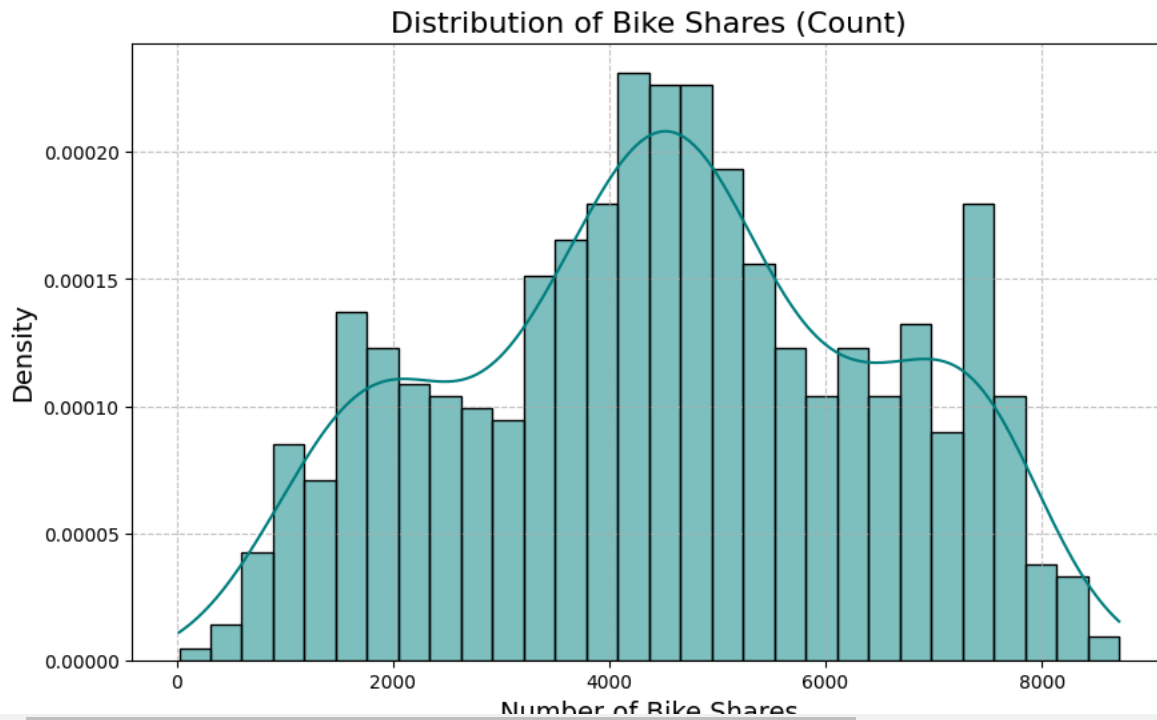
Visualize the distribution of count, casual and registered variables

```
# YOUR CODE HERE for distribuion of count variable
df1 = pd.read_csv('day.csv')

# Visualize the distribution of the 'count' variable (number of bike shares)
plt.figure(figsize=(10, 6))

# Plot the histogram of the 'count' column
sns.histplot(df1['cnt'], kde=True, bins=30, color='teal', stat='density')

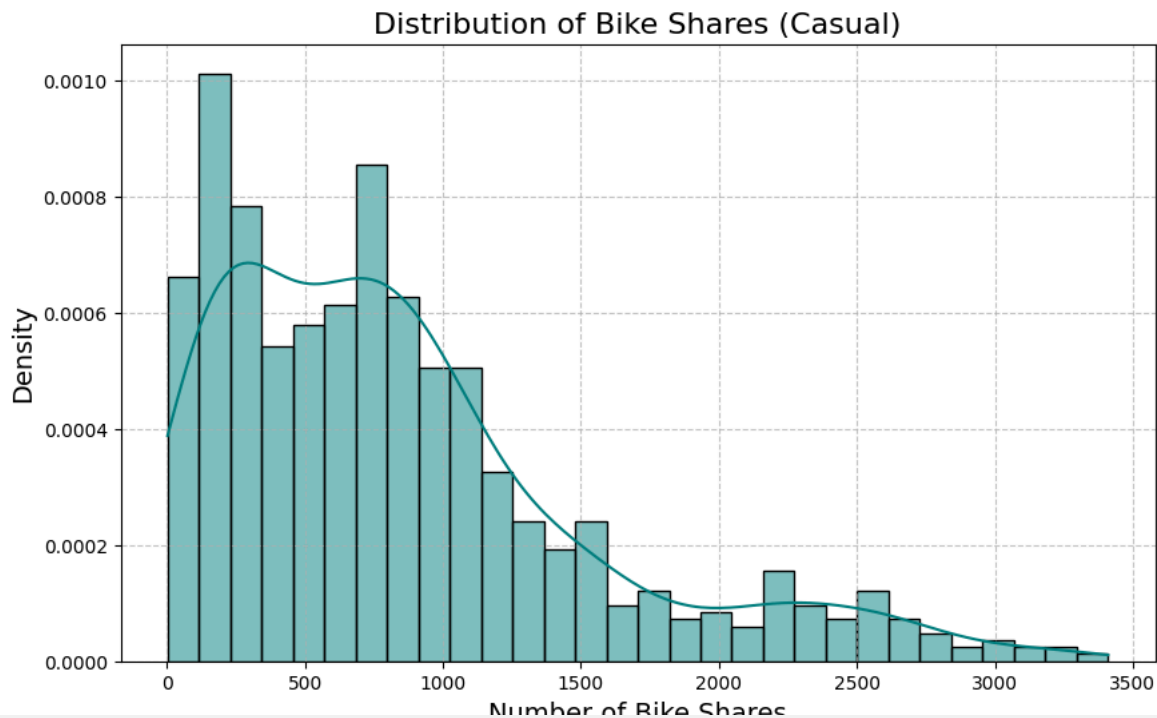
# Customize plot appearance
plt.title('Distribution of Bike Shares (Count)', fontsize=16)
plt.xlabel('Number of Bike Shares', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```



```
# YOUR CODE HERE for distribuion of casual variable
plt.figure(figsize=(10, 6))

# Plot the histogram of the 'count' column
sns.histplot(df1['casual'], kde=True, bins=30, color='teal', stat='density')

# Customize plot appearance
plt.title('Distribution of Bike Shares (Casual)', fontsize=16)
plt.xlabel('Number of Bike Shares', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

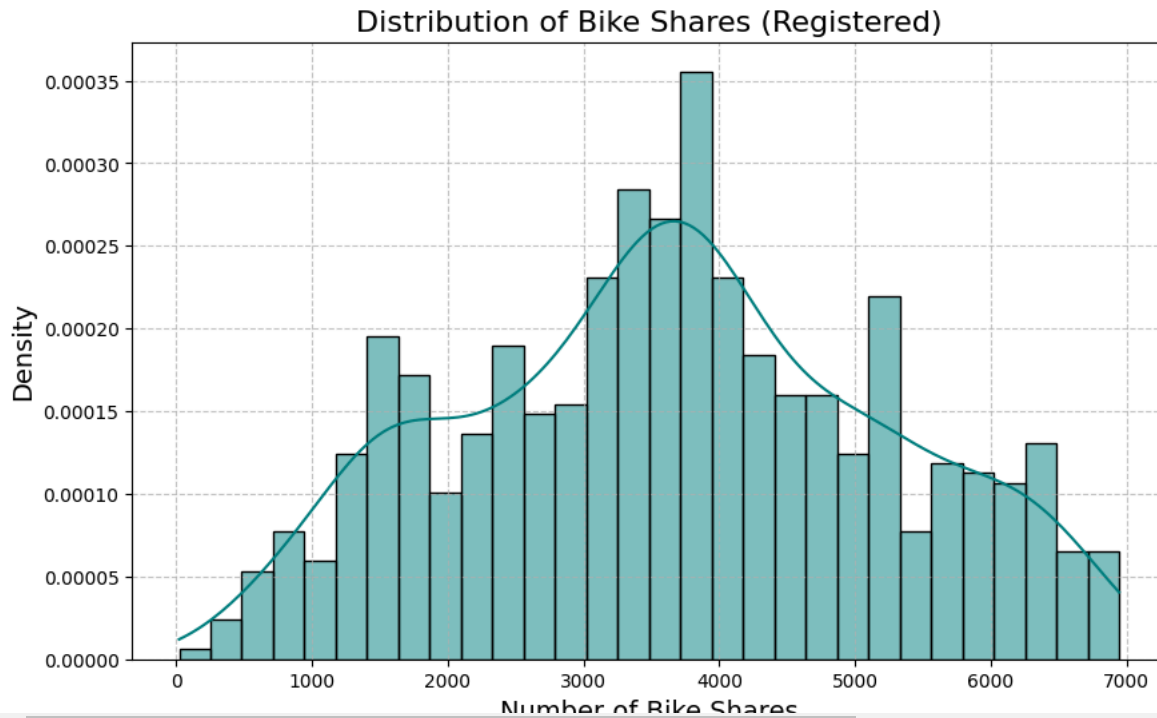


```
# YOUR CODE HERE for distribuion of registered variable
plt.figure(figsize=(10, 6))

# Plot the histogram of the 'count' column
sns.histplot(df1['registered'], kde=True, bins=30, color='teal', stat='density')

# Customize plot appearance
plt.title('Distribution of Bike Shares (Registered)', fontsize=16)
```

```
plt.xlabel('Number of Bike Shares', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```



Describe the relation of weekday, holiday and working day

YOUR CODE HERE

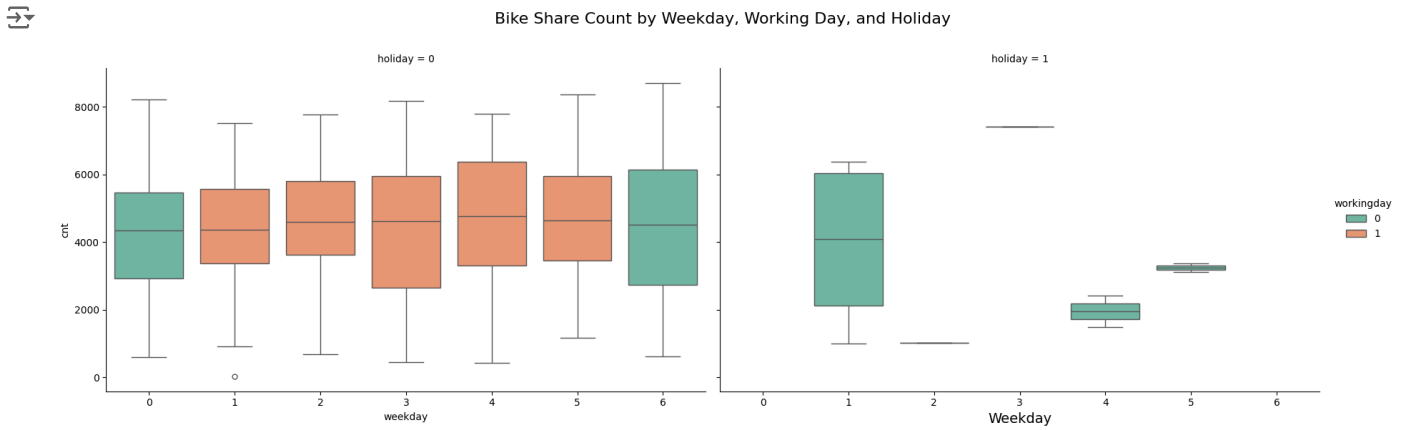
```
sns.catplot(
    data=df1,
    x='weekday',
    y='cnt',
    hue='workingday', # color by working day
    col='holiday', # create separate plots for holidays (0/1)
    kind='box', # use boxplot for distribution
    palette='Set2',
    height=6,
    aspect=1.5
)
```

Customize the plot

```
plt.subplots_adjust(top=0.85)
plt.suptitle('Bike Share Count by Weekday, Working Day, and Holiday', fontsize=16)
plt.xlabel('Weekday', fontsize=14)
plt.ylabel('Number of Bike Shares', fontsize=14)
```

Show the plot

```
plt.show()
```



✓ Visualize the month wise count of both casual and registered for the year 2011 and 2012 separately.

Hint: Stacked barchart

```
# stacked bar chart for year 2011
# YOUR CODE HERE
df1['dteday'] = pd.to_datetime(df1['dteday'], format='%Y-%m-%d', errors='coerce')

# Filter the data for the year 2011
df_2011 = df1[df1['dteday'].dt.year == 2011]

# Extract the month from the 'datetime' column (no time needed)
df_2011['month'] = df_2011['dteday'].dt.month

# Group by month and aggregate the counts for 'casual' and 'registered' users
monthly_counts = df_2011.groupby('month')[['casual', 'registered']].sum()

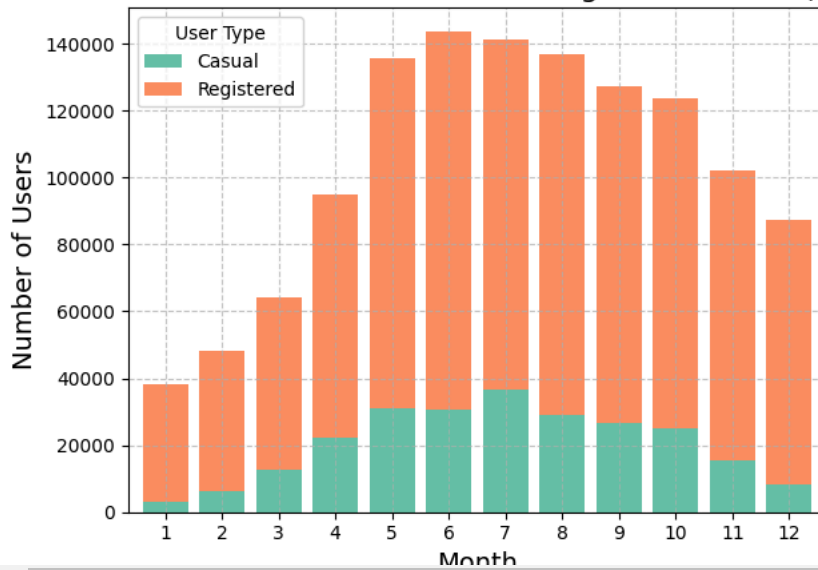
# Plot the stacked bar chart
plt.figure(figsize=(10, 6))
monthly_counts.plot(kind='bar', stacked=True, color=['#66c2a5', '#fc8d62'], width=0.8)

# Customize the plot
plt.title('Month-wise Count of Casual and Registered Users (2011)', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Users', fontsize=14)
plt.xticks(rotation=0)
plt.legend(title='User Type', labels=['Casual', 'Registered'], loc='upper left')
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
<ipython-input-11-41bd89731078>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-df_2011\['month'\] = df_2011\['dteday'\].dt.month](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-df_2011['month'] = df_2011['dteday'].dt.month)
<Figure size 1000x600 with 0 Axes>

Month-wise Count of Casual and Registered Users (2011)



```
# stacked bar chart for year 2012
# YOUR CODE HERE
df1['dteday'] = pd.to_datetime(df1['dteday'], format='%Y-%m-%d', errors='coerce')

# Filter the data for the year 2011
df_2012 = df1[df1['dteday'].dt.year == 2012]

# Extract the month from the 'datetime' column (no time needed)
df_2012['month'] = df_2012['dteday'].dt.month

# Group by month and aggregate the counts for 'casual' and 'registered' users
monthly_counts = df_2012.groupby('month')[['casual', 'registered']].sum()

# Plot the stacked bar chart
plt.figure(figsize=(10, 6))
monthly_counts.plot(kind='bar', stacked=True, color=['#66c2a5', '#fc8d62'], width=0.8)

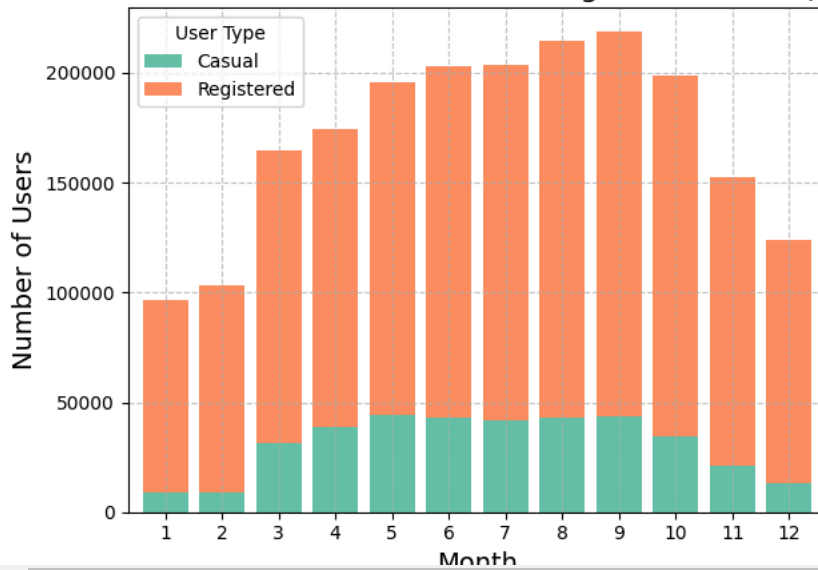
# Customize the plot
plt.title('Month-wise Count of Casual and Registered Users (2012)', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Users', fontsize=14)
plt.xticks(rotation=0)
plt.legend(title='User Type', labels=['Casual', 'Registered'], loc='upper left')
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



```
<ipython-input-12-57b4c99cd619>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-df_2012\['month'\] = df_2012\['dteday'\].dt.month](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-df_2012['month'] = df_2012['dteday'].dt.month)
<Figure size 1000x600 with 0 Axes>

Month-wise Count of Casual and Registered Users (2012)



▼ Analyze the correlation between features with heatmap

```
# YOUR CODE HERE
correlation_matrix = df1.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 8))

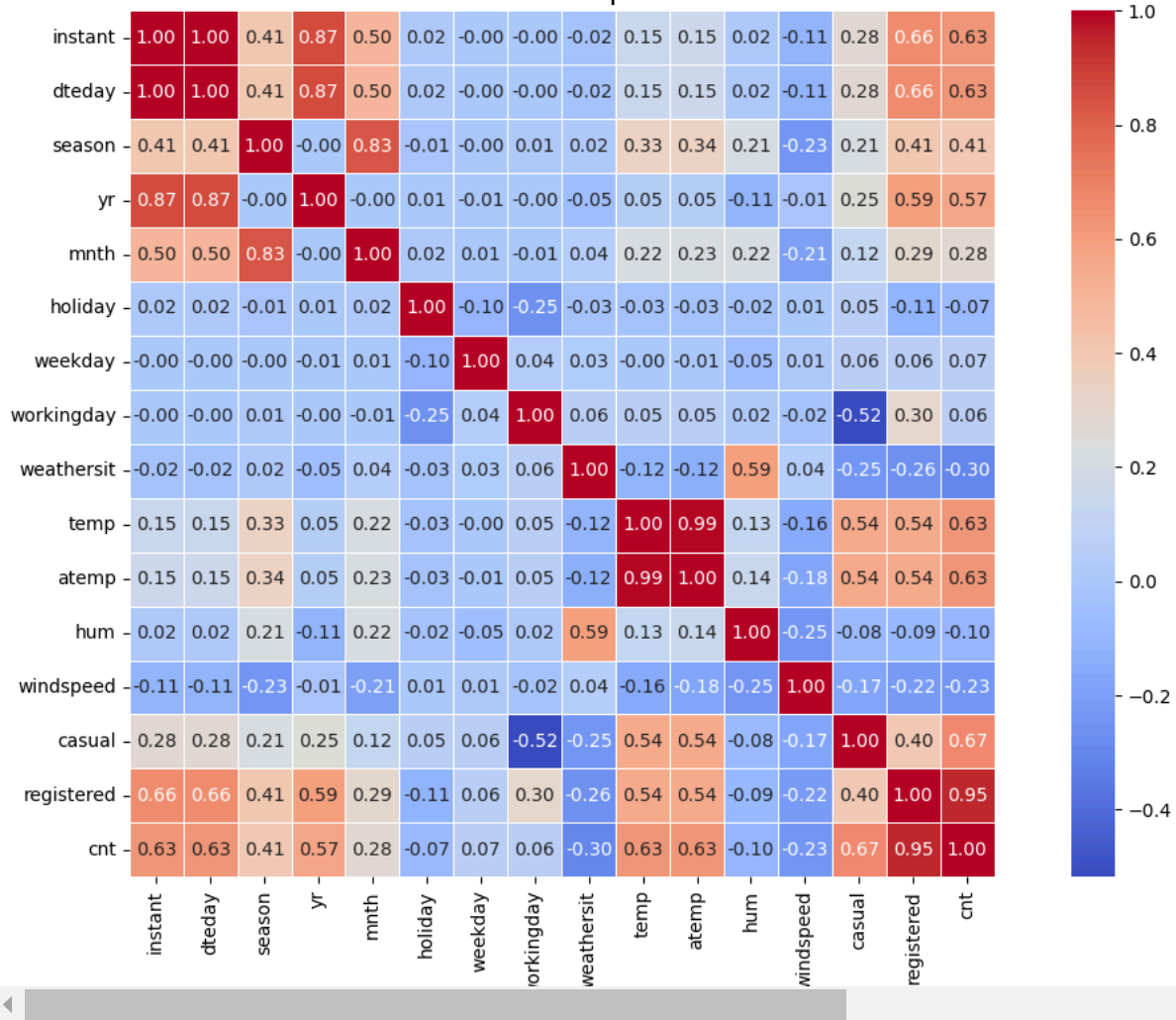
# Create the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', cbar=True, square=True, linewidths=0.5)

# Add a title
plt.title('Correlation Heatmap of Features', fontsize=16)

# Show the plot
plt.tight_layout()
plt.show()
```



Correlation Heatmap of Features



Visualize the box plot of casual and registered variables to check the outliers

```
# YOUR CODE HERE
plt.figure(figsize=(10, 6))

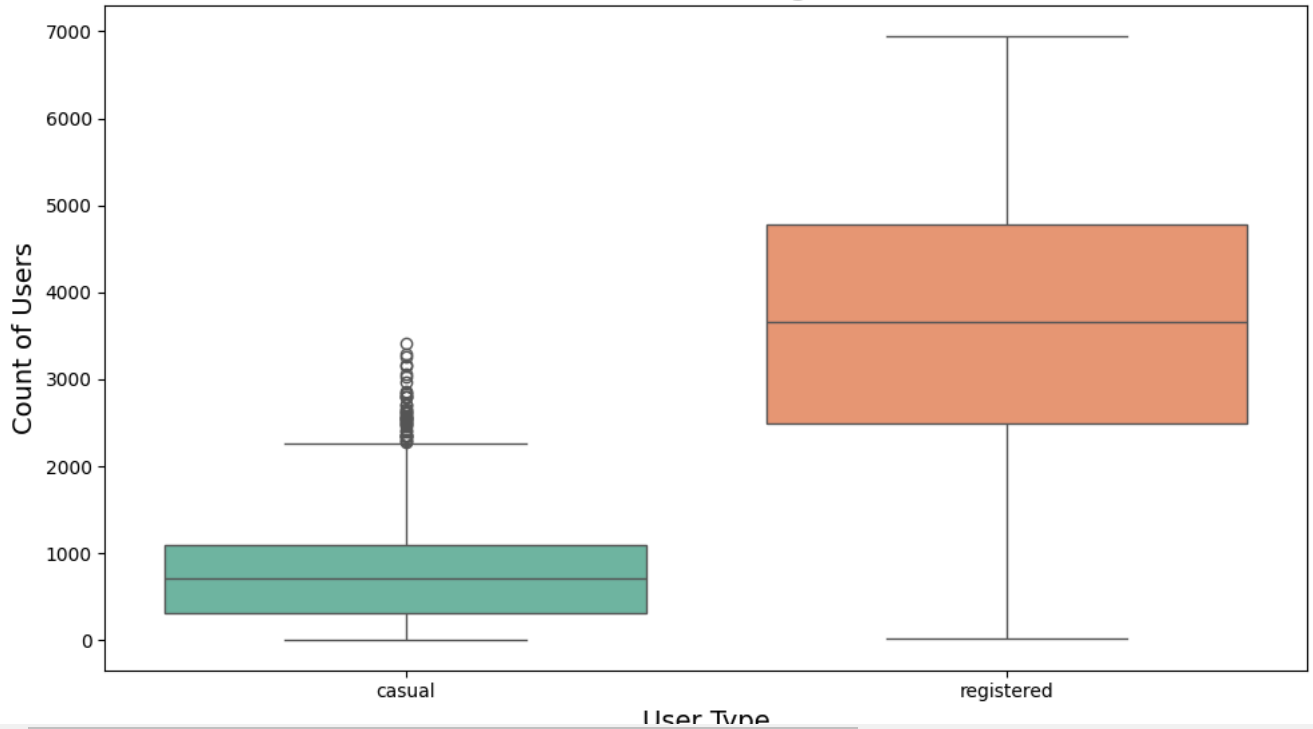
# Create a boxplot for casual and registered users
sns.boxplot(data=df1[['casual', 'registered']], palette="Set2")

# Add a title and labels
plt.title('Box Plot of Casual and Registered Users', fontsize=16)
plt.xlabel('User Type', fontsize=14)
plt.ylabel('Count of Users', fontsize=14)

# Show the plot
plt.tight_layout()
plt.show()
```



Box Plot of Casual and Registered Users



✓ Pre-processing and Data Engineering (1 point)

✓ Drop unwanted columns

```
# YOUR CODE HERE
columns_ = ['instant', 'casual', 'registered']
df1 = df1.drop(columns = columns_)
```

✓ Identify categorical and continuous variables

```
# YOUR CODE HERE
print(df1.dtypes)

# Identify categorical variables: typically 'object', 'category', 'bool'
categorical_vars = df1.select_dtypes(include=['object', 'category', 'bool']).columns.tolist()

# Identify continuous variables: typically 'int64', 'float64'
continuous_vars = df1.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Print the results
print("Categorical Variables:", categorical_vars)
print("Continuous Variables:", continuous_vars)
```

```
dteday      datetime64[ns]
season      int64
yr          int64
mnth        int64
holiday      int64
weekday      int64
workingday   int64
weathersit    int64
temp         float64
atemp        float64
hum          float64
windspeed    float64
cnt          int64
dtype: object
Categorical Variables: []
Continuous Variables: ['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed']
```

✓ Feature scaling

Feature scaling is essential for machine learning algorithms, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance. Apply scaling on the continuous variables on the given data.

Hint: MinMaxScaler or StandardScaler

```
# YOUR CODE HERE
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
df_minmax_scaled = df1.copy()
df_minmax_scaled[continuous_vars] = min_max_scaler.fit_transform(df1[continuous_vars])
```

✓ Apply one-hot encode on the categorical data

One-hot encoding is applied on the categorical variables, which should not have a different weight or order attached to them, it is presumed that all categorical variables have equivalent "values". This means that you cannot simply order them from zero to the number of categories as this would imply that the earlier categories have less "value" than later categories.

Hint: sklearn.preprocessing.OneHotEncoder

```
# YOUR CODE HERE
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, drop='first')
encoded_array = encoder.fit_transform(df1[categorical_vars])
encoded_df = pd.DataFrame(encoded_array, columns=encoder.get_feature_names_out(categorical_vars))
```

✓ Specify features and targets after applying scaling and one-hot encoding

```
# YOUR CODE HERE
target = 'cnt'

# Separate features (X) and target (y) from the one-hot encoded DataFrame
X = df_minmax_scaled.drop(columns=[target])
y = df_minmax_scaled[target]

# Optional: Convert to NumPy arrays if required by your ML algorithm
X_array = X.values
y_array = y.values
```

✓ Implement the linear regression by finding the coefficients using below approaches (2 points)

- Find the coefficients using normal equation
- (Optional) Implement batch gradient descent
- (Optional) SGD Regressor from sklearn

✓ Select the features and target and split the dataset

As there are 3 target variables, choose the count (cnt) variable.

```
# YOUR CODE HERE
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print the shapes of the resulting splits (optional)
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (584, 12)
X_test shape: (147, 12)
y_train shape: (584,)
y_test shape: (147,)
```

✓ Implementation using Normal Equation

$$\theta = (X^T X)^{-1} \cdot (X^T Y)$$

θ is the hypothesis parameter that defines the coefficients

X is the input feature value of each instance

Y is Output value of each instance

For performing Linear Regression Using the Normal Equation refer [here](#).

To solve the normal equation compute least-squares solution by using `scipy.linalg`

Hint: [scipy.linalg.lstsq](#)

```
# YOUR CODE HERE
import numpy as np
from scipy.linalg import lstsq

# Drop non-numeric columns (like datetime) and ensure numeric types
X_train_numeric = X_train.select_dtypes(include=[np.number]).values
X_test_numeric = X_test.select_dtypes(include=[np.number]).values
y_train_numeric = y_train.values.astype(float)
y_test_numeric = y_test.values.astype(float)

# Add a bias column (intercept term) to the feature matrices
X_train_with_bias = np.c_[np.ones(X_train_numeric.shape[0]), X_train_numeric]
X_test_with_bias = np.c_[np.ones(X_test_numeric.shape[0]), X_test_numeric]

# Compute the least-squares solution using the normal equation
theta, residuals, rank, singular_values = lstsq(X_train_with_bias, y_train_numeric)


# Print the coefficients
print("Coefficients (theta):", theta)

# Predict on the training and testing sets
y_train_pred = X_train_with_bias @ theta
y_test_pred = X_test_with_bias @ theta

# Evaluate performance
from sklearn.metrics import mean_squared_error, r2_score

mse_train = mean_squared_error(y_train_numeric, y_train_pred)
mse_test = mean_squared_error(y_test_numeric, y_test_pred)
r2_train = r2_score(y_train_numeric, y_train_pred)
r2_test = r2_score(y_test_numeric, y_test_pred)

print(f"Training MSE: {mse_train}, R²: {r2_train}")
print(f"Testing MSE: {mse_test}, R²: {r2_test}")
```

 Coefficients (theta): [0.1648602 0.18110534 0.23285752 -0.04865293 -0.04504726 0.05034768
0.01850033 -0.1456181 0.19363992 0.3057159 -0.09682925 -0.11610797]
Training MSE: 0.01013647389681229, R²: 0.7910851850465888
Testing MSE: 0.009146619530416827, R²: 0.8276670090367211

✓ (Optional) Implementing Linear regression using batch gradient descent

Initialize the random coefficients and optimize the coefficients in the iterative process by calculating cost and finding the gradient.

Hint: [gradient descent](#)

```
# YOUR CODE HERE
import numpy as np

# Ensure X_train and X_test contain only numeric data
X_train_numeric = X_train.select_dtypes(include=[np.number]).values
X_test_numeric = X_test.select_dtypes(include=[np.number]).values
y_train_numeric = y_train.values.astype(float)
y_test_numeric = y_test.values.astype(float)

# Add a bias column (intercept term) to the feature matrix
X_train_with_bias = np.c_[np.ones(X_train_numeric.shape[0]), X_train_numeric]
X_test_with_bias = np.c_[np.ones(X_test_numeric.shape[0]), X_test_numeric]

# Initialize parameters
m, n = X_train_with_bias.shape
theta = np.random.randn(n) # Random initialization of coefficients
learning_rate = 0.01 # Set a learning rate
iterations = 1000 # Number of iterations

# Gradient Descent
for i in range(iterations):
    # Predict values
    y_pred = X_train_with_bias @ theta
```

```

# Calculate cost (Mean Squared Error)
cost = np.mean((y_pred - y_train_numeric) ** 2)

# Compute the gradient
gradient = (2 / m) * X_train_with_bias.T @ (y_pred - y_train_numeric)

# Update coefficients
theta -= learning_rate * gradient

# Print the cost for every 100 iterations
if i % 100 == 0:
    print(f"Iteration {i}, Cost: {cost}")

# Print the optimized coefficients
print("Optimized Coefficients (theta):", theta)

# Prediction for the test set
y_test_pred = X_test_with_bias @ theta

# Evaluate performance
from sklearn.metrics import mean_squared_error, r2_score

mse_test = mean_squared_error(y_test_numeric, y_test_pred)
r2_test = r2_score(y_test_numeric, y_test_pred)

print(f"Testing MSE: {mse_test}, R²: {r2_test}")

```

```

↗ Iteration 0, Cost: 2.091573825857619
Iteration 100, Cost: 0.8512474092999089
Iteration 200, Cost: 0.3979582520525887
Iteration 300, Cost: 0.21206298866601167
Iteration 400, Cost: 0.13057394928704552
Iteration 500, Cost: 0.0913097988845747
Iteration 600, Cost: 0.07009864767744078
Iteration 700, Cost: 0.057248436191998875
Iteration 800, Cost: 0.04867799421166707
Iteration 900, Cost: 0.04254205131906318
Optimized Coefficients (theta): [ 0.09284185 -0.36843371  0.19478935  0.74054565  0.16820174  0.12456054
  0.05801933 -0.36595524 -0.08766348  0.62970131 -0.37893903  0.22520934]
Testing MSE: 0.053567637408704516, R²: -0.009276830940597947

```

✓ (Optional) SGD Regressor

Scikit-learn API provides the SGDRegressor class to implement SGD method for regression problems. The SGD regressor applies regularized linear model with SGD learning to build an estimator. A regularizer is a penalty (L1, L2, or Elastic Net) added to the loss function to shrink the model parameters.

- Import SGDRegressor from sklearn and fit the data
- Predict the test data and find the error

Hint: [SGDRegressor](#)

```

# YOUR CODE HERE
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Ensure X_train and X_test contain only numeric data
X_train_numeric = X_train.select_dtypes(include=[np.number])
X_test_numeric = X_test.select_dtypes(include=[np.number])
y_train_numeric = y_train.astype(float)
y_test_numeric = y_test.astype(float)

# Initialize the SGDRegressor
sgd_regressor = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)

# Fit the model to the training data
sgd_regressor.fit(X_train_numeric, y_train_numeric)

# Predict on the test set
y_test_pred = sgd_regressor.predict(X_test_numeric)

# Calculate the error metrics
mse_test = mean_squared_error(y_test_numeric, y_test_pred)
r2_test = r2_score(y_test_numeric, y_test_pred)

# Print the error metrics
print(f"Testing Mean Squared Error (MSE): {mse_test}")

```

```
print(f"Testing R2 Score: {r2_test}")
```

```
Testing Mean Squared Error (MSE): 0.014566134940735512
Testing R2 Score: 0.7255570112254053
```

Linear regression using sklearn (3 points)

Implement the linear regression model using sklearn

- Import Linear Regression and fit the train data
- Predict the test data and find the error

Hint: [LinearRegression](#)

```
# YOUR CODE HERE
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Ensure X_train and X_test contain only numeric data
X_train_numeric = X_train.select_dtypes(include=[np.number])
X_test_numeric = X_test.select_dtypes(include=[np.number])
y_train_numeric = y_train.astype(float)
y_test_numeric = y_test.astype(float)

# Initialize the Linear Regression model
linear_regressor = LinearRegression()

# Fit the model to the training data
linear_regressor.fit(X_train_numeric, y_train_numeric)

# Predict on the test set
y_test_pred = linear_regressor.predict(X_test_numeric)

# Calculate the error metrics
mse_test = mean_squared_error(y_test_numeric, y_test_pred)

# Print the error metrics
print(f"Testing Mean Squared Error (MSE): {mse_test}")
```

```
Testing Mean Squared Error (MSE): 0.009146619530416823
```

Calculate the R^2 (coefficient of determination) of the actual and predicted data

```
# YOUR CODE HERE
from sklearn.metrics import r2_score

# Predict on the test set using the fitted model (assuming linear_regressor is fitted)
y_test_pred = linear_regressor.predict(X_test_numeric)

# Calculate the R2 (coefficient of determination)
r2_value = r2_score(y_test_numeric, y_test_pred)

# Print the R2 value
print(f"R2 Score: {r2_value}")
```

```
R2 Score: 0.8276670090367212
```

Summarize the importance of features

Prediction is the weighted sum of the input values e.g. linear regression. Regularization, such as ridge regression and the elastic net, find a set of coefficients to use in the weighted sum to make a prediction. These coefficients can be used directly as a crude type of feature importance score. This assumes that the input variables have the same scale or have been scaled prior to fitting a model.

Use the coefficients obtained through the sklearn Linear Regression implementation and create a bar chart of the coefficients.

```
# YOUR CODE HERE
import matplotlib.pyplot as plt
import pandas as pd

# Extract coefficients from the fitted Linear Regression model
```

```

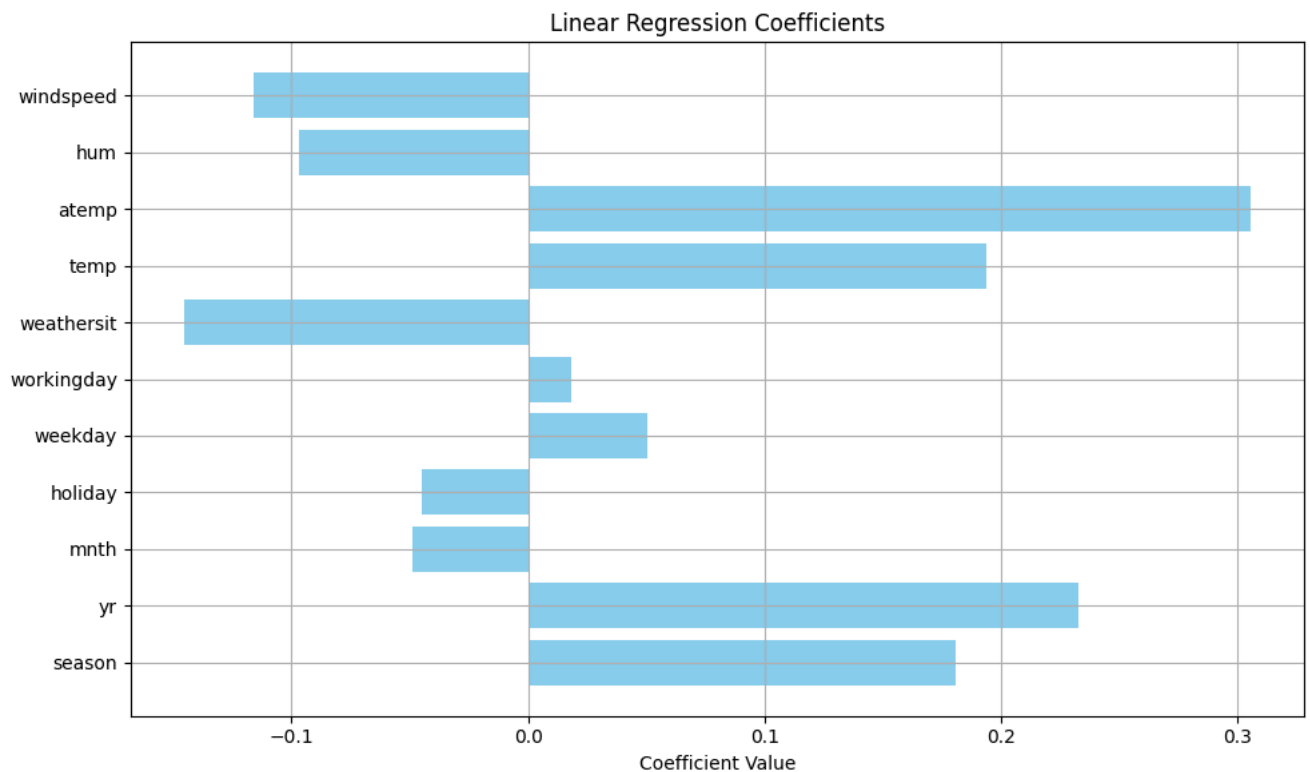
coefficients = linear_regressor.coef_

# Feature names for labeling (excluding the intercept)
feature_names = X_train_numeric.columns

# Create a DataFrame for easier plotting
coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})

# Plotting the coefficients as a bar chart
plt.figure(figsize=(10, 6))
plt.barh(coef_df['Feature'], coef_df['Coefficient'], color='skyblue')
plt.xlabel('Coefficient Value')
plt.title('Linear Regression Coefficients')
plt.grid(True)
plt.tight_layout()
plt.show()

```



✓ Regularization methods (2 points)

✓ Apply Lasso regression

- Apply Lasso regression with different alpha values given below and find the best alpha that gives the least error.
- Calculate the metrics for the actual and predicted

Hint: [Lasso](#)

```

# setting up alpha
alpha_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]

# YOUR CODE HERE
from sklearn.linear_model import Lasso
best_alpha = None
best_mse = float('inf')
best_r2 = -float('inf')
results = []

# Iterate over different alpha values
for alpha in alpha_values:
    # Initialize the Lasso model with the current alpha
    lasso_regressor = Lasso(alpha=alpha, random_state=42)

    # Fit the model to the training data
    lasso_regressor.fit(X_train_numeric, y_train_numeric)

    # Predict on the test set

```



```

y_test_pred = lasso_regressor.predict(X_test_numeric)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test_numeric, y_test_pred)

# Calculate R2 Score
r2 = r2_score(y_test_numeric, y_test_pred)


# Track the best alpha
if mse < best_mse:
    best_mse = mse
    best_alpha = alpha
    best_r2 = r2

# Store results
results.append((alpha, mse, r2))

# Print the best alpha and its corresponding metrics
print(f"Best Alpha: {best_alpha}")
print(f"Minimum MSE: {best_mse}")
print(f"Maximum R2 Score: {best_r2}")

# Display all results
for alpha, mse, r2 in results:
    print(f"Alpha: {alpha}, MSE: {mse}, R2 Score: {r2}")

```

 Best Alpha: 0.0001
 Minimum MSE: 0.009196120983312834
 Maximum R² Score: 0.8267343438694177
 Alpha: 0.0001, MSE: 0.009196120983312834, R² Score: 0.8267343438694177
 Alpha: 0.001, MSE: 0.009918109127188999, R² Score: 0.8131312442914325
 Alpha: 0.01, MSE: 0.014584642497105026, R² Score: 0.7252083072551601
 Alpha: 0.1, MSE: 0.054124898830033394, R² Score: -0.019776286741366
 Alpha: 1, MSE: 0.054124898830033394, R² Score: -0.019776286741366
 Alpha: 10, MSE: 0.054124898830033394, R² Score: -0.019776286741366
 Alpha: 100, MSE: 0.054124898830033394, R² Score: -0.019776286741366

▼ Apply Ridge regression

- Apply Ridge regression with different alpha values given and find the best alpha that gives the least error.
- Calculate the metrics for the actual and predicted

Hint: [Ridge](#)

```

# YOUR CODE HERE
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

# Alpha values to test
alpha_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]

# Initialize variables to track the best model
best_alpha = None
best_mse = float('inf')
best_r2 = -float('inf')
results = []

# Iterate over different alpha values
for alpha in alpha_values:
    # Initialize the Ridge model with the current alpha
    ridge_regressor = Ridge(alpha=alpha, random_state=42)

    # Fit the model to the training data
    ridge_regressor.fit(X_train_numeric, y_train_numeric)

```