```
# 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
# Read the hour.csv file
bike_data = pd.read_csv('/content/hour.csv')
print the rst ve rows of dataset
bike_data.head()
import matplotlib.pyplot as plt
import seaborn as sns
# Create a histogram of the 'hr' column
plt.figure(figsize=(10, 6))
sns.histplot(bike_data['hr'], bins=24, kde=False)
plt.title('Distribution of Bike Sharing by Hour')
plt.xlabel('Hour of the Day')
plt.ylabel('Count')
plt.xticks(range(24))
plt.show()
import matplotlib.pyplot as plt
import seaborn as sns
# Create a histogram of the 'cnt' column
plt.figure(figsize=(10, 6))
```

```
sns.histplot(bike_data['cnt'], kde=True)  # kde=True to show the density curve
plt.title('Distribution of Bike Sharing Count')
plt.xlabel('Total Bike Rentals (cnt)')
plt.ylabel('Frequency')
plt.show()
 import matplotlib.pyplot as plt
import seaborn as sns
# Create a histogram of the 'casual' column
plt.figure(figsize=(10, 6))
sns.histplot(bike_data['casual'], kde=True)  # kde=True to show the density curve
plt.title('Distribution of Casual Bike Rentals')
plt.xlabel('Number of Casual Rentals (casual)')
plt.ylabel('Frequency')
plt.show()
import matplotlib.pyplot as plt
import seaborn as sns
# Create a histogram of the 'registered' column
plt.figure(figsize=(10, 6))
sns.histplot(bike_data['registered'], kde=True)  # kde=True to show the density curve
plt.title('Distribution of Registered Bike Rentals')
plt.xlabel('Number of Registered Rentals (registered)')
plt.ylabel('Frequency')
plt.show()
import matplotlib.pyplot as plt
import seaborn as sns
# Create a histogram of the 'registered' column
plt.figure(figsize=(10, 6))
sns.histplot(bike_data['registered'], kde=True)  # kde=True to show the density curve
plt.title('Distribution of Registered Bike Rentals')
plt.xlabel('Number of Registered Rentals (registered)')
```

```
plt.ylabel('Frequency')
plt.show()
import matplotlib.pyplot as plt
import pandas as pd
# Filter data for 2011
bike_data_2011 = bike_data[pd.to_datetime(bike_data['dteday']).dt.year == 2011]
# Group data by month and sum casual and registered rentals
monthly_counts =
bike_data_2011.groupby(pd.to_datetime(bike_data_2011['dteday']).dt.month)[['casual',
'registered']].sum()
# Create stacked bar chart
monthly_counts.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Monthly Bike Rentals in 2011 (Casual vs. Registered)')
plt.xlabel('Month')
plt.ylabel('Total Rentals')
plt.xticks(range(12), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec'])
plt.legend(loc='upper left')
plt.show()
 import matplotlib.pyplot as plt
import pandas as pd
# Filter data for 2012
bike_data_2012 = bike_data[pd.to_datetime(bike_data['dteday']).dt.year == 2012]
# Group data by month and sum casual and registered rentals
monthly_counts_2012 =
bike_data_2012.groupby(pd.to_datetime(bike_data_2012['dteday']).dt.month)[['casual',
'registered']].sum()
# Create stacked bar chart
monthly_counts_2012.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Monthly Bike Rentals in 2012 (Casual vs. Registered)')
plt.xlabel('Month')
```

```
plt.ylabel('Total Rentals')
plt.xticks(range(12), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.legend(loc='upper left')
plt.show()
 import matplotlib.pyplot as plt
import seaborn as sns
# Calculate correlation matrix, excluding non-numeric columns
correlation_matrix = bike_data.select_dtypes(include=np.number).corr()
# Create heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of Bike Sharing Features')
plt.show()
import matplotlib.pyplot as plt
import seaborn as sns
# Create box plots for casual and registered variables
plt.figure(figsize=(10, 6))
sns.boxplot(data=bike_data[['casual', 'registered']])
plt.title('Box Plots of Casual and Registered Rentals')
plt.ylabel('Number of Rentals')
plt.show()
 bike_data = bike_data.drop(['instant', 'dteday', 'casual', 'registered'], axis=1)
categorical_vars = ['season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersit']
continuous_vars = ['temp', 'atemp', 'hum', 'windspeed', 'cnt']
from sklearn.preprocessing import MinMaxScaler
# Create a MinMaxScaler object
scaler = MinMaxScaler()
# Fit the scaler to the continuous variables
```

```
scaler.fit(bike_data[continuous_vars[:-1]])  # Exclude 'cnt' (target) from scaling
# Transform the continuous variables
bike_data[continuous_vars[:-1]] = scaler.transform(bike_data[continuous_vars[:-1]])
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
# Create a OneHotEncoder object
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore') # sparse=False for dense output
# Fit the encoder to the categorical variables
encoder.fit(bike_data[categorical_vars])
# Transform the categorical variables
encoded_data = encoder.transform(bike_data[categorical_vars])
# Get feature names for the encoded columns
encoded_feature_names = encoder.get_feature_names_out(categorical_vars)
# Create a DataFrame for the encoded data
encoded_df = pd.DataFrame(encoded_data, columns=encoded_feature_names, index=bike_data.index)
# Concatenate the encoded data with the original DataFrame
bike_data = pd.concat([bike_data, encoded_df], axis=1)
# Drop the original categorical columns
bike_data = bike_data.drop(categorical_vars, axis=1)
# Specify features (all columns except 'cnt')
features = bike_data.drop('cnt', axis=1)
# Specify target ('cnt' column)
target = bike_data['cnt']
 from sklearn.model_selection import train_test_split
# Features (all columns except 'cnt')
features = bike_data.drop('cnt', axis=1)
# Target ('cnt' column)
target = bike_data['cnt']
```

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
import numpy as np
from scipy.linalg import lstsq
# Add a column of ones to X_train for the intercept term
X_train_with_intercept = np.c_[np.ones(X_train.shape[0]), X_train]
# Calculate coefficients using the normal equation
coefficients, _, _, _ = lstsq(X_train_with_intercept, y_train)
# Print the coefficients
print("Coefficients:", coefficients)
 import numpy as np
# Add a column of ones to X_train for the intercept term
X_train_with_intercept = np.c_[np.ones(X_train.shape[0]), X_train]
# Initialize coefficients randomly
coefficients = np.random.rand(X_train_with_intercept.shape[1])
# Set hyperparameters
learning_rate = 0.01
iterations = 1000
# Batch gradient descent loop
for _ in range(iterations):
# Calculate predictions
predictions = X_train_with_intercept @ coefficients
# Calculate error
error = predictions - y_train
# Update coefficients
coefficients = coefficients - learning_rate * (X_train_with_intercept.T @ error) / len(y_train)
# Print the coefficients
print("Coefficients:", coefficients)
```

```
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
# Create an SGDRegressor object
sgd_regressor = SGDRegressor(max_iter=1000, tol=1e-3, random_state=42)  # Adjust parameters as needed
# Fit the model to the training data
sgd_regressor.fit(X_train, y_train)
# Predict on the test data
y_pred = sgd_regressor.predict(X_test)
# Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
# Print the MSE
print("Mean Squared Error:", mse)
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# Create a LinearRegression object
linear_regressor = LinearRegression()
# Fit the model to the training data
linear_regressor.fit(X_train, y_train)
# Predict on the test data
y_pred = linear_regressor.predict(X_test)
# Calculate the Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
# Print the MSE
print("Mean Squared Error:", mse)
from sklearn.metrics import r2_score
# Assuming you have y_test (actual values) and y_pred (predicted values)
r2 = r2_score(y_test, y_pred)
# Print the R-squared value
print("R-squared:", r2)
```

```
import matplotlib.pyplot as plt
import numpy as np
# Get feature names and coefficients
feature_names = X_train.columns
coefficients = linear_regressor.coef_
# Sort features by absolute coefficient value
sorted_indices = np.argsort(np.abs(coefficients))[::-1]  # Sort in descending order
sorted_features = feature_names[sorted_indices]
sorted_coefficients = coefficients[sorted_indices]
# Create bar chart
plt.figure(figsize=(12, 6))
plt.bar(sorted_features, sorted_coefficients)
plt.xticks(rotation=90)
plt.title('Feature Importance based on Linear Regression Coefficients')
plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.tight_layout()
plt.show()
# setting up alpha_values
alpha_values = [0.0001, 0.001,0.01, 0.1, 1, 10, 100]
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
# Initialize variables to store results
best_alpha = None
best_mse = float('inf')  # Initialize with a very large value
# Iterate through alpha values
for alpha in alpha_values:
  # Create Lasso model
  lasso_model = Lasso(alpha=alpha, random_state=42)
  # Fit the model
```

```
  lasso_model.fit(X_train, y_train)
  # Predict on test data
  y_pred = lasso_model.predict(X_test)
  # Calculate MSE
  mse = mean_squared_error(y_test, y_pred)
  # Update best alpha and MSE if current MSE is lower
  if mse < best_mse:
     best_mse = mse
     best_alpha = alpha
# Print the best alpha and MSE
print("Best alpha:", best_alpha)
print("Best MSE:", best_mse)
# Calculate R-squared for the best model
best_lasso_model = Lasso(alpha=best_alpha, random_state=42)
best_lasso_model.fit(X_train, y_train)
y_pred_best = best_lasso_model.predict(X_test)
r2 = r2_score(y_test, y_pred_best)
print("R-squared:", r2)
 from sklearn.linear_model import Ridge
# Initialize variables to store results
best_alpha = None
best_mse = float('inf')  # Initialize with a very large value
# Iterate through alpha values
for alpha in alpha_values:
  # Create Ridge model
  ridge_model = Ridge(alpha=alpha, random_state=42)
  # Fit the model
  ridge_model.fit(X_train, y_train)
  # Predict on test data
  y_pred = ridge_model.predict(X_test)
```

```
    # Calculate MSE
    mse = mean_squared_error(y_test, y_pred)
  # Update best alpha and MSE if current MSE is lower
    if mse < best_mse:
        best_mse = mse
        best_alpha = alpha
# Print the best alpha and MSE
print("Best alpha:", best_alpha)
print("Best MSE:", best_mse)
# Calculate R-squared for the best model
best_ridge_model = Ridge(alpha=best_alpha, random_state=42)
best_ridge_model.fit(X_train, y_train)
y_pred_best = best_ridge_model.predict(X_test)
r2 = r2_score(y_test, y_pred_best)
print("R-squared:", r2)
from sklearn.linear_model import ElasticNet
# Initialize variables to store results
best_alpha = None
best_mse = float('inf')  # Initialize with a very large value
# Iterate through alpha values
for alpha in alpha_values:
    # Create ElasticNet model
    elasticnet_model = ElasticNet(alpha=alpha, random_state=42)
    # Fit the model
    elasticnet_model.fit(X_train, y_train)
    # Predict on test data
    y_pred = elasticnet_model.predict(X_test)
    # Calculate MSE
    mse = mean_squared_error(y_test, y_pred)
    # Update best alpha and MSE if current MSE is lower
```

```
    if mse < best_mse:
        best_mse = mse
        best_alpha = alpha
# Print the best alpha and MSE
print("Best alpha:", best_alpha)
print("Best MSE:", best_mse)
# Calculate R-squared for the best model
best_elasticnet_model = ElasticNet(alpha=best_alpha, random_state=42)
best_elasticnet_model.fit(X_train, y_train)
y_pred_best = best_elasticnet_model.predict(X_test)
r2 = r2_score(y_test, y_pred_best)
print("R-squared:", r2)
```