CST 2130 Data Management and Business Intelligence

Coursework 2: Machine Learning

Dr. Jaspreet Singh Sethi

Sanjar Rozyyev – M00912071

Lebohang Khasipe – M00860241

Md Ashraful Islam – M00853622

Abubakar Adamu Sani – M00853235

# Introduction

Coursework 2: Machine Learning

Balancing bike-sharing supply and demand is crucial for the success of any public bicycle-renting company. In London, the Santander Cycles has experienced rapid growth, presenting challenges in ensuring enough bikes are available when and where they are needed. This project aims to solve this challenge by trying to predict daily bike usage using the data from Transport for London (TfL).

To achieve this, we utilized two powerful machine learning models: Logistic Regression and Random Forest Classifier. Logistic Regression is a linear model that estimates the probability of a specific outcome (bike_rented in this case) based on independent variables like weather and time. Its interpretability allows us to understand which factors most influence bike usage. On the other hand, Random Forest creates an ensemble of decision trees, making it resistant to outliers and complex data. Its strength lies in its ability to capture even non-linear patterns, potentially leading to more accurate predictions.

By comparing the performance of these models through k-fold cross-validation, we aim to identify the most effective model for predicting bike usage. This information can be helpful for TfL to optimize bike distribution and improve user experience.

# Logistic Regression

In [307]:

```
# improting libraries required for data manipulation
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
%matplotlib inline
```

In [308]:

```
bike_data = pd.read_csv('London_bike_data.csv')
## firstly to get a better understanding of the data at hand we use built in functions fr
om pandas to display the data of london bike, we will then look at the data and look for
any relations between the data
#displaying the data in a table format see the varaibles and data we are dealing with
bike_data.head()
```

Out[308]:

| | id | date | hour | season | is_weekend | is_holiday | temperature | temperature_feels | humidity | wind_speed | weather_code |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 8650 | 2016-01-01 | 6 | 3 | 0 | 1 | 3.0 | 0.0 | 87.0 | 10.0 | 1 |
| 1 | 9383 | 2016-01-31 | 19 | 3 | 1 | 0 | 14.0 | 14.0 | 77.0 | 35.0 | 3 |
| 2 | 12036 | 2016-05-22 | 8 | 0 | 1 | 0 | 14.5 | 14.5 | 65.0 | 6.5 | 1 |
| 3 | 2404 | 2015-04-14 | 11 | 0 | 0 | 0 | 18.0 | 18.0 | 54.0 | 21.5 | 1 |
| 4 | 7406 | 2015-11-09 | 21 | 2 | 0 | 0 | 15.0 | 15.0 | 82.0 | 31.5 | 4 |

In [309]:

```
# displaying the bike_data information using built in function .info() from pandas and ge
t an understanding of the datatype of each column, as you can see bike_rented is a catego
rical string as well as date. Will need to remove ID as it has no relation to anything. W
ill also drop more varaiables such as is_holdiay, season, and weather_code. We will ponte
itnailly also drop date to see if it have any effect on the model accuracy
bike_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13060 entries, 0 to 13059
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 13060 non-null  int64  
 1   date               13060 non-null  object 
 2   hour               13060 non-null  int64  
 3   season             13060 non-null  int64  
 4   is_weekend         13060 non-null  int64  
 5   is_holiday         13060 non-null  int64  
 6   temperature        13060 non-null  float64
 7   temperature_feels  13060 non-null  float64
 8   humidity           13060 non-null  float64
 9   wind_speed         13060 non-null  float64
 10  weather_code       13060 non-null  int64  
 11  bike_rented        13060 non-null  object 
dtypes: float64(4), int64(6), object(2)
memory usage: 1.2+ MB
```

In [310]:

```
# displaying the Bike rented group's occurance count
bike_data['bike_rented'].value_counts()
```

Out[310]:

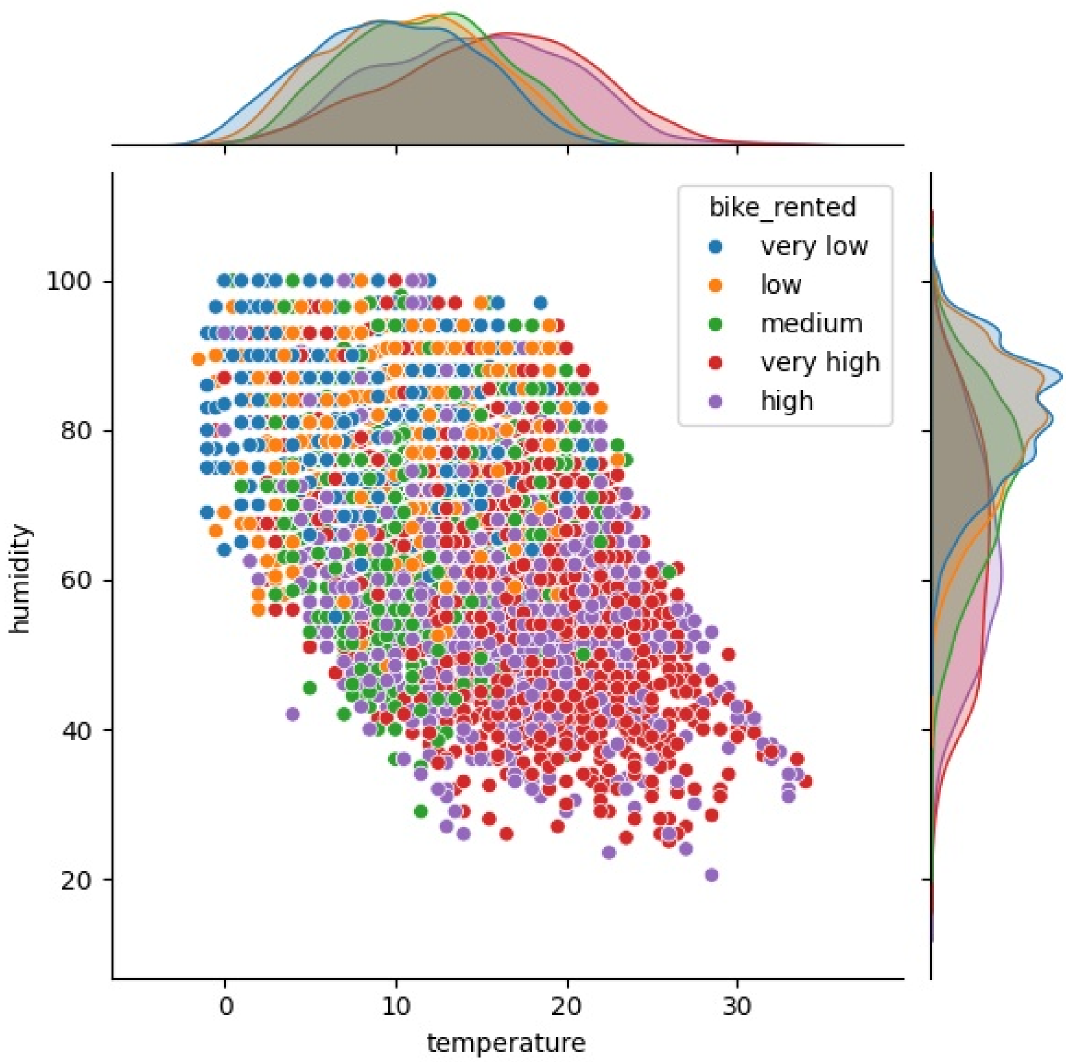
```
low          2642
very low     2629
high         2620
very high    2592
medium       2577
Name: bike_rented, dtype: int64
```

In [319]:

```
#Exploratory Analysis

# showing the relathionship between the temperature and humidity colour coded on the leve
l of bikes rented.
sns.jointplot(x="temperature", y='humidity', hue='bike_rented', data=bike_data)


# As you can see from the jointplot you can see that their is a higher increase of bikes
rented when the humidty levels are low and the temperature is high.
```

Out[319]:

```
<seaborn.axisgrid.JointGrid at 0x7f973c2c6dc0>
```



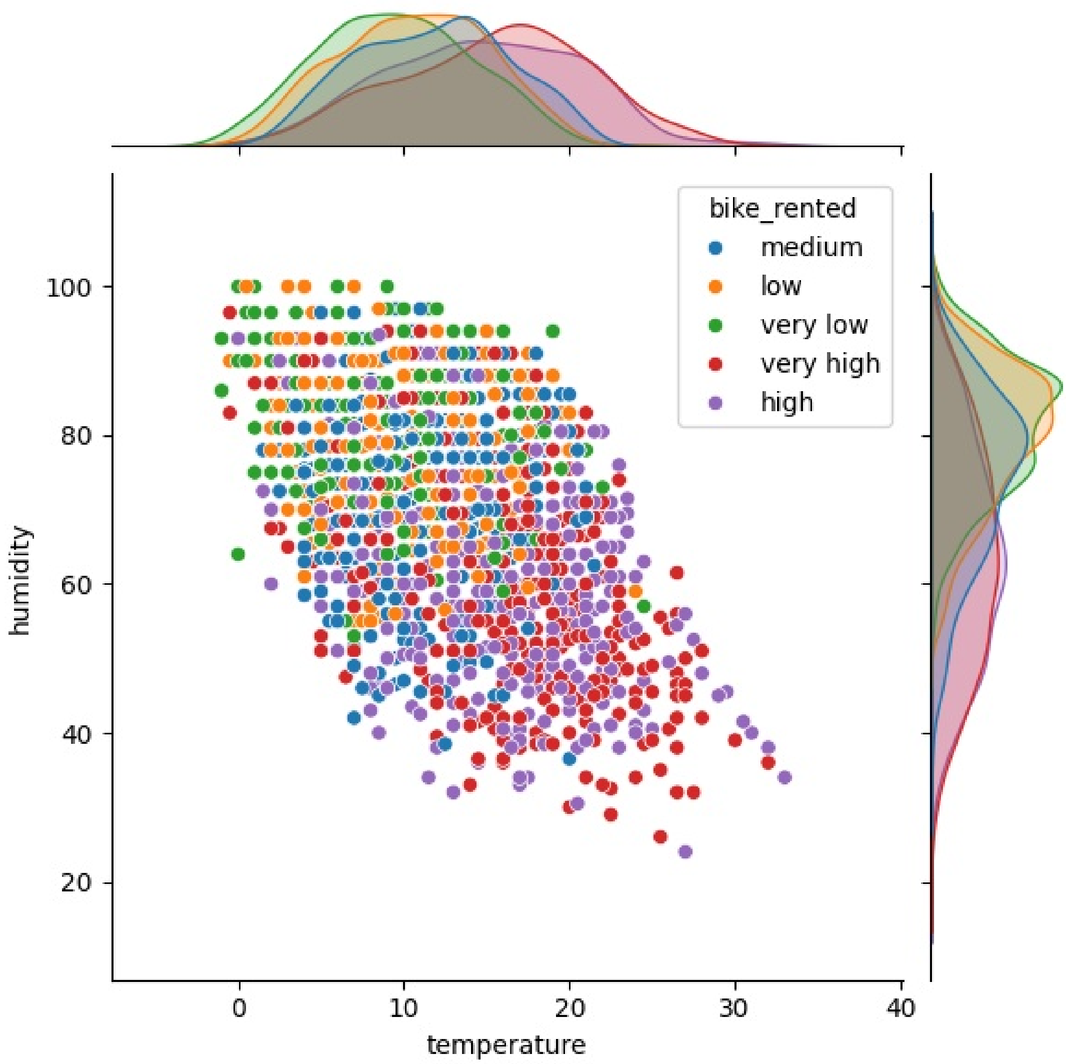

In [312]:

```
# Sample a fraction of the data (20%)
sampled_data = bike_data.sample(frac=0.2)
```

```
sns.jointplot(x="temperature", y='humidity', hue='bike_rented', data=sampled_data)

#sampling data to improve the performance of the visualization to make plots more readabl
e and provide a clearer representation of trends or patterns in the data. as u can see he
re after taking about 20% of the data we can see that the same conclusion of the relation
ship still stands.
```

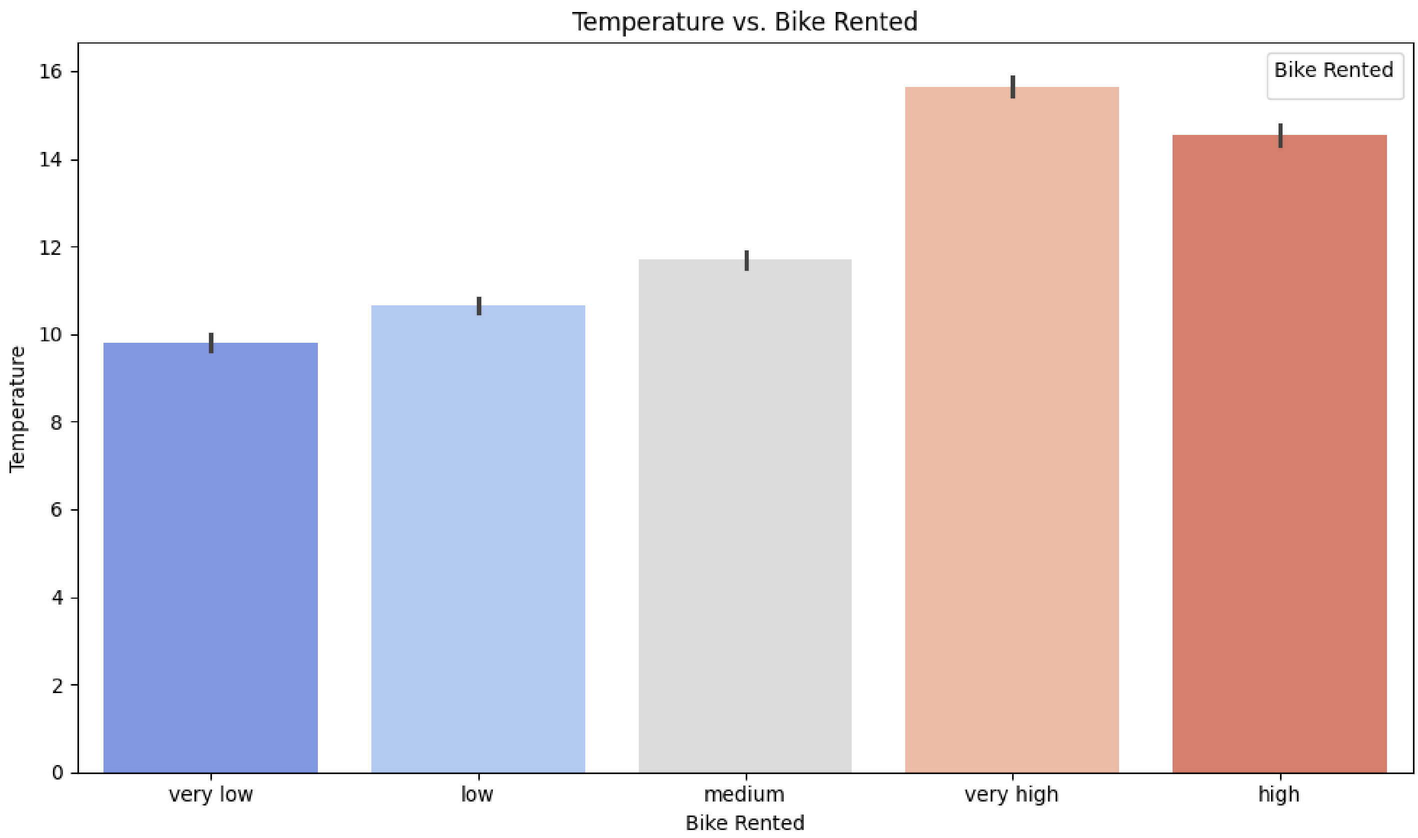
Out[312]:

```
<seaborn.axisgrid.JointGrid at 0x7f973bfc1640>
```




In [313]:

```
# Creating a bar chart visualiztion to see the affect of temperature on the amount of bik
es rented
#setting the figure size 10 by 6
plt.figure(figsize=(10, 6))
#built in function barplot setting x and y varaibles color coded by bike_rented setting p
alette to coolwarm
sns.barplot(x='bike_rented', y='temperature', hue='bike_rented', data=bike_data, palette=
'coolwarm')

# Setting the plot titles and labels
plt.title('Temperature vs. Bike Rented')
plt.xlabel('Bike Rented')
plt.ylabel('Temperature')

# Show the plot
plt.tight_layout()
plt.legend(title='Bike Rented')  # Add legend with title
plt.show()

## as you can see from the bargraph below, we can see a trend in the increase of temperat
ure leading to an increase in bike_rented. So we can state that when the temperature incr
eases, the level of bikes rented is very high compared to when tmperatures are lower. Inc
onclusion they should increase their level of stocks during hotter months.
```




In [314]:

```
# Next we start to build the model, splitting the data into training and testing, fitting
our model using logistic regression, retreiving the predictions, and display the classifi
cation report, and evulate with k-cross validation



## model building

# dropping unneccesary features as they are unneccarry to the output we are trying to ach
eive
# we drop id, is_holiday, weather_code, and is_weekend to see the affects of the accuracy
by dropping unnecssary features from the model
bike_data.drop(['id', 'is_holiday', 'weather_code', 'season'], axis=1, inplace=True)
#feature variables selected are temperature, is_weekend, and humidty
bike_data.info()
X = bike_data[['temperature', 'is_weekend', 'humidity', 'wind_speed']]  #adding is_weeke
nd increased the avg accuraxy of the model by 1 percent.
# dynamically setting target varaible to be the last column in the dataset after dropping
unnesccary data from it
y = bike_data['bike_rented']

# splitting the dataset using train and test method( 80% for training and 20% for testing
)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10
1)



logclf = LogisticRegression()
logclf.fit(X_train, y_train)  # Use the training data for fitting using logisitc regressi
on



## Predictions and evaluations

predictions = logclf.predict(X_test)
```

```
## classification and report for model

# high recall false
from sklearn.metrics import classification_report
print("############Classification Report#########")
print(classification_report(y_test, predictions))

## K fold cross validation to evalute model performance

from sklearn.model_selection import KFold


# creating k fold object with 5 splits
kf = KFold(n_splits=5)

# creating empty array accuracy
accuracy = []

# initializing logistic regression model
model = LogisticRegression()




# for loop to iterate through the kfold cross validation splits
print("#############K FOLD CROSS VALIDATION#########")
for train_index, test_index in kf.split(X):
    # fetching X and y from train/test
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    # fitting the model on the training data
    model.fit(X_train, y_train)
    # storing the predicitons of test data in variable perdicitons
    predictions = model.predict(X_test)
    # calculating and printing the metrics

    print('Accuracy score:', model.score(X_test, y_test))
    accuracy.append(model.score(X_test, y_test))


# Calculate and round the average accuracy to 2 decimal places
avg_accuracy = np.mean(accuracy)
rounded_avg_accuracy = round(avg_accuracy, 2)

# Print the rounded average accuracy
print("Average Accuracy:", rounded_avg_accuracy)

# As you can see within the classification report we got an average accuracy score of 36%
. From here we will try and see if adding hour and day as a feature variable will have an
y significant impact on the model accuracy. Inaddition,  we will increase the number of k
splits to pontetially give us a higher accuracy score.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13060 entries, 0 to 13059
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   date               13060 non-null  object 
 1   hour               13060 non-null  int64  
 2   is_weekend         13060 non-null  int64  
 3   temperature        13060 non-null  float64
 4   temperature_feels  13060 non-null  float64
 5   humidity           13060 non-null  float64
 6   wind_speed         13060 non-null  float64
 7   bike_rented        13060 non-null  object 
dtypes: float64(4), int64(2), object(2)
memory usage: 816.4+ KB
############Classification Report#########
              precision    recall  f1-score   support

        high       0.28      0.21      0.24       538
         low       0.41      0.42      0.42       519
```

```
      medium       0.26      0.17      0.21       529
   very high       0.41      0.48      0.44       498
    very low       0.33      0.47      0.39       528

    accuracy                           0.35      2612
   macro avg       0.34      0.35      0.34      2612
weighted avg       0.34      0.35      0.34      2612

#############K FOLD CROSS VALIDATION#########
Accuracy score: 0.3518376722817764
Accuracy score: 0.36294027565084225
Accuracy score: 0.36408882082695254
Accuracy score: 0.3610260336906585
Accuracy score: 0.36408882082695254
Average Accuracy: 0.36
```

In [315]:

```
# Converting date to datetime
bike_data['timestamp'] = pd.to_datetime(bike_data['date'])
# Creating new columns
# Creating months of year, and days of month to use for prediction
# As months and days are repeated, these can be an indicator for bike rented
# Defining month, day, is_weekend as category as they cannot be expressed as a fraction t
o show a transition between two groups
bike_data['month'] = bike_data['timestamp'].dt.month.astype('category')
bike_data['day'] = bike_data['timestamp'].dt.day.astype('category')
bike_data['is_weekend'] = bike_data['is_weekend'].astype('category')
```

In [316]:

```
# as we have created a bargraph to see the feature importance of each variable, I have de
cided to add day and hour as a feature variable to see if it will increase the accuracy o
f the model. We will also increase the number of splits to 10 to see if that will have an
y affect. Firstly we will see the affect of the addition of day as one of the feature var
iables to the model

## model building

# dropping unneccesary features as they are unneccessary to the output we are trying to a
cheive
bike_data.drop(['date', 'month'], axis=1, inplace=True)
#feature variables selected are temperature, is_weekend, 'wind_speed', day, and humidity
bike_data.info()
X = bike_data[['temperature', 'is_weekend', 'humidity', 'wind_speed', 'day']]
#target varaiable bike rented
y = bike_data['bike_rented']

# splitting the dataset using train and test method( 80% for training and 20% for testing
)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10
1)



logclf = LogisticRegression()
logclf.fit(X_train, y_train)  # Use the training data for fitting using logisitc regressi
on



## Predictions and evaluations

predictions = logclf.predict(X_test)



## classification and report for model

# high recall false
from sklearn.metrics import classification_report
```

```python
print("############Classification Report########")
print(classification_report(y_test, predictions))

## K fold cross validation to evalute model performance

from sklearn.model_selection import KFold


# creating k fold object with 5 splits
kf = KFold(n_splits=5)

# creating empty array accuracy
accuracy = []

# initializing logistic regression model
model = LogisticRegression()




# for loop to iterate through the kfold cross validation splits
print("############K FOLD CROSS VALIDATION########")
for train_index, test_index in kf.split(X):
    # fetching X and y from train/test
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    # fitting the model on the training data
    model.fit(X_train, y_train)
    # storing the predicitons of test data in variable perdicitons
    predictions = model.predict(X_test)
    # calculating and printing the metrics

    print('Accuracy score:', model.score(X_test, y_test))
    accuracy.append(model.score(X_test, y_test))


# Calculate and round the average accuracy to 2 decimal places
avg_accuracy = np.mean(accuracy)
rounded_avg_accuracy = round(avg_accuracy, 2)

# Print the rounded average accuracy
print("Average Accuracy:", rounded_avg_accuracy)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13060 entries, 0 to 13059
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   hour               13060 non-null  int64
 1   is_weekend         13060 non-null  category
 2   temperature        13060 non-null  float64
 3   temperature_feels  13060 non-null  float64
 4   humidity           13060 non-null  float64
 5   wind_speed         13060 non-null  float64
 6   bike_rented        13060 non-null  object
 7   timestamp          13060 non-null  datetime64[ns]
 8   day                13060 non-null  category
dtypes: category(2), datetime64[ns](1), float64(4), int64(1), object(1)
memory usage: 741.3+ KB
############Classification Report########
              precision    recall  f1-score   support

        high       0.32      0.12      0.18       538
         low       0.41      0.43      0.42       519
      medium       0.27      0.21      0.24       529
   very high       0.37      0.56      0.45       498
    very low       0.33      0.45      0.39       528

    accuracy                           0.35      2612
   macro avg       0.34      0.35      0.33      2612
weighted avg       0.34      0.35      0.33      2612
```

```
#############K FOLD CROSS VALIDATION#########
Accuracy score: 0.3568147013782542
Accuracy score: 0.35911179173047475
Accuracy score: 0.3633231240428790З
Accuracy score: 0.36217457886676874
Accuracy score: 0.36791730474732004
Average Accuracy: 0.36
```

In [317]:

```
# Next we will implement hour as one of the feature variables to see if it will increase
the accuracy of the model and see its affect on the model as it has the highest feature i
mportance as shown in the feature importance graph.

## model building



#feature variables selected are temperature, is_weekend, and humidty
bike_data.info()
X = bike_data[['temperature', 'is_weekend', 'humidity', 'wind_speed', 'day', 'hour']]
# setting target variable bike_rented
y = bike_data['bike_rented']

# splitting the dataset using train and test method( 80% for training and 20% for testing
)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10
1)



logclf = LogisticRegression()
logclf.fit(X_train, y_train)  # Use the training data for fitting using logisitc regressi
on



## Predictions and evaluations

predictions = logclf.predict(X_test)



## classification and report for model

# high recall false
from sklearn.metrics import classification_report
print("############Classification Report#########")
print(classification_report(y_test, predictions))

## K fold cross validation to evalute model performance

from sklearn.model_selection import KFold


# creating k fold object with 5 splits
kf = KFold(n_splits=5)

# creating empty array accuracy
accuracy = []

# initializing logistic regression model
model = LogisticRegression()




# for loop to iterate through the kfold cross validation splits
print("#############K FOLD CROSS VALIDATION#########")
for train_index, test_index in kf.split(X):
    # fetching X and y from train/test
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
```

```
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    # fitting the model on the training data
    model.fit(X_train, y_train)
    # storing the predicitons of test data in variable perdicitons
    predictions = model.predict(X_test)
    # calculating and printing the metrics

    print('Accuracy score:', model.score(X_test, y_test))
    accuracy.append(model.score(X_test, y_test))


# Calculate and round the average accuracy to 2 decimal places
avg_accuracy = np.mean(accuracy)
rounded_avg_accuracy = round(avg_accuracy, 2)

# Print the rounded average accuracy
print("Average Accuracy:", rounded_avg_accuracy)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13060 entries, 0 to 13059
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   hour               13060 non-null  int64
 1   is_weekend         13060 non-null  category
 2   temperature        13060 non-null  float64
 3   temperature_feels  13060 non-null  float64
 4   humidity           13060 non-null  float64
 5   wind_speed         13060 non-null  float64
 6   bike_rented        13060 non-null  object
 7   timestamp          13060 non-null  datetime64[ns]
 8   day                13060 non-null  category
dtypes: category(2), datetime64[ns](1), float64(4), int64(1), object(1)
memory usage: 741.3+ KB
#############Classification Report#########
              precision    recall  f1-score   support

        high       0.34      0.19      0.24       538
         low       0.29      0.28      0.29       519
      medium       0.37      0.36      0.37       529
   very high       0.45      0.51      0.48       498
    very low       0.59      0.82      0.68       528

    accuracy                           0.43      2612
   macro avg       0.41      0.43      0.41      2612
weighted avg       0.41      0.43      0.41      2612

#############K FOLD CROSS VALIDATION#########
Accuracy score: 0.41883614088820825
Accuracy score: 0.4333843797856049
Accuracy score: 0.4375957120980092
Accuracy score: 0.43606431852986216
Accuracy score: 0.4375957120980092
Average Accuracy: 0.43
```

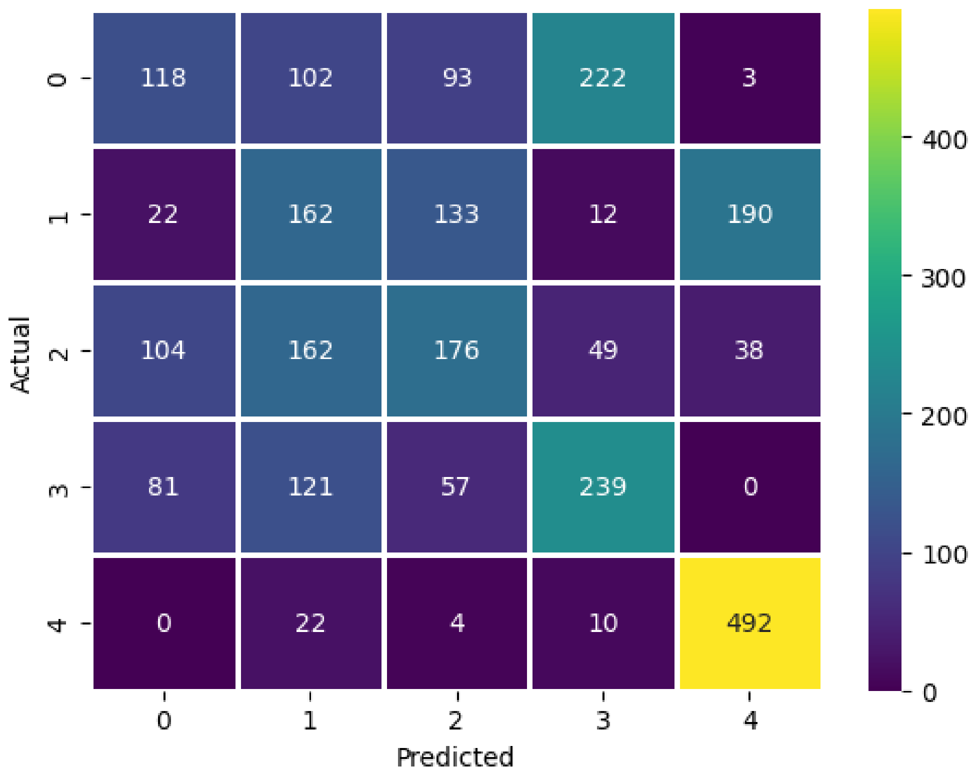
In [318]:

```
#Finally we will see if increasing the number of k splits to 10 instead of 5 and we will
remove is_weekend as one of the feature_varaibles as it has the least feature importance
compared to the other varaibles and see the impact it has on the accuracy of the model



## model building

#feature variables selected are temperature, is_weekend, and humidty
bike_data.info()
X = bike_data[['temperature', 'humidity', 'wind_speed', 'day', 'hour']]  #removing is_we
ekend increased the  accuracy of the model by 2 percent.
# dynamically setting target varaible to be the last column in the dataset after dropping
unnesccary data from it
y = bike_data['bike_rented']
```

```
# splitting the dataset using train and test method( 80% for training and 20% for testing
)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10
1)



logclf = LogisticRegression()
logclf.fit(X_train, y_train)  # Use the training data for fitting using logisitc regressi
on



## Predictions and evaluations

predictions = logclf.predict(X_test)

# Printing the confusion matrix to understand how accurate the model predictions are
confusion_matrix(y_test, predictions)

cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot=True, cmap='viridis', fmt='d', linewidth = 1)
plt.xlabel('Predicted')
plt.ylabel('Actual')
print("##########Confusion Matrix###########")
plt.show()



## classification and report for model

# high recall false

print("############Classification Report########")
print(classification_report(y_test, predictions))

## K fold cross validation to evalute model performance

from sklearn.model_selection import KFold


# creating k fold object with 5 splits
kf = KFold(n_splits=10)

# creating empty array accuracy
accuracy = []

# initializing logistic regression model
model = LogisticRegression()




# for loop to iterate through the kfold cross validation splits
print("############K FOLD CROSS VALIDATION########")
for train_index, test_index in kf.split(X):
    # fetching X and y from train/test
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    # fitting the model on the training data
    model.fit(X_train, y_train)
    # storing the predicitons of test data in variable perdicitons
    predictions = model.predict(X_test)
    # calculating and printing the metrics

    print('Accuracy score:', model.score(X_test, y_test))
    accuracy.append(model.score(X_test, y_test))


# Calculate and round the average accuracy to 2 decimal places
avg_accuracy = np.mean(accuracy)
```

```python
rounded_avg_accuracy = round(avg_accuracy, 2)

# Print the rounded average accuracy
print("Average Accuracy:", rounded_avg_accuracy)

#Inconclusion, When we included day as a feature varaible and dropping any unncessary dat
a such as months to our model it had no impact on the accuracy of our model. However, wit
h the addition of hour as a feature varaible we were able to increase the accuracy model
from 36% to 43%. This brings us back to the feature importance graph showing that hour ha
s the higest feature importance affect our model accuracy by 7%.

#Furthermore, changing the number of splits from 5 to 10 also had no affect on the averag
e accuracy of the model but when we removed is_weekend as a feature variable, The accurac
y went from 43% to 45% with an average accuracy of 44%. A two percent increase in accurac
y.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13060 entries, 0 to 13059
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   hour               13060 non-null  int64
 1   is_weekend         13060 non-null  category
 2   temperature        13060 non-null  float64
 3   temperature_feels  13060 non-null  float64
 4   humidity           13060 non-null  float64
 5   wind_speed         13060 non-null  float64
 6   bike_rented        13060 non-null  object
 7   timestamp          13060 non-null  datetime64[ns]
 8   day                13060 non-null  category
dtypes: category(2), datetime64[ns](1), float64(4), int64(1), object(1)
memory usage: 741.3+ KB
###########Confusion Matrix############
```




```
############Classification Report#########
              precision    recall  f1-score   support

        high       0.36      0.22      0.27       538
         low       0.28      0.31      0.30       519
      medium       0.38      0.33      0.35       529
   very high       0.45      0.48      0.46       498
    very low       0.68      0.93      0.79       528

    accuracy                           0.45      2612
   macro avg       0.43      0.46      0.44      2612
weighted avg       0.43      0.45      0.43      2612
```

```
#############K FOLD CROSS VALIDATION#########
Accuracy score: 0.41577335375191427
Accuracy score: 0.4234303215926493
Accuracy score: 0.43644716692189894
Accuracy score: 0.43721286370597245
Accuracy score: 0.45099540581929554
Accuracy score: 0.445635528330781
Accuracy score: 0.43797856049004597
Accuracy score: 0.442572741194487
Accuracy score: 0.46018376722817766
Accuracy score: 0.4555895865237366
Average Accuracy: 0.44
```

# Random Forest Classifier

# CST2130_CW2_1

February 23, 2024

[1]:
```
# This section of the coursework will focus on analysing the dataset and␣
↪implementing the Random Forest Classifier Model
#
#
# We will analyse the features, create a Random Forest Classifier Model,␣
↪implement it, and evaluate it
#
#
# Importing the libraries that will be used initially for data reading,␣
↪visualisation, and modelling
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
# Reading the London_bike_data.csv and storing it in df
df = pd.read_csv("London_bike_data.csv")
# My hypothesis is that the hour, day of the week, and the temperature will␣
↪have the highest impact on the bike_rented variable
```

[3]:
```
# Outputting the number of rows and columns in the dataset
df.shape
```

[3]:
```
(13060, 12)
```

[4]:
```
# The head function is giving us more insight on what the FIRST records within␣
↪the dataset are and how they are represented
df.head()
```

[4]:
```
      id        date  hour  season  is_weekend  is_holiday  temperature  \
0   8650  2016-01-01     6       3           0           1          3.0
1   9383  2016-01-31    19       3           1           0         14.0
2  12036  2016-05-22     8       0           1           0         14.5
3   2404  2015-04-14    11       0           0           0         18.0
4   7406  2015-11-09    21       2           0           0         15.0

   temperature_feels  humidity  wind_speed  weather_code bike_rented
0                0.0      87.0        10.0             1    very low
```

```
1                14.0      77.0        35.0                3         low
2                14.5      65.0         6.5                1         low
3                18.0      54.0        21.5                1      medium
4                15.0      82.0        31.5                4      medium
```

[5]:
```python
# The tail function is giving us more insight on what the LAST records within␣
↪the dataset are and how they are represented
df.tail()
```

[5]:
```
          id        date  hour  season  is_weekend  is_holiday  temperature  \
13055   5876  2015-09-06    10       2           1           0         14.0
13056   5541  2015-08-23    11       1           1           0         22.0
13057  10575  2016-03-21    19       0           0           0          9.0
13058   5126  2015-08-06     4       1           0           0         18.0
13059   1048  2015-02-16    17       3           0           0          8.0

       temperature_feels  humidity  wind_speed  weather_code bike_rented
13055               14.0      63.0         8.0             1        high
13056               22.0      63.0        10.0             3   very high
13057                7.5      60.0        10.0             3        high
13058               18.0      64.0        10.0             1    very low
13059                7.5      87.0         5.0             7        high
```

[6]:
```python
# Convert the 'date' column to datetime format to enable easy extraction of␣
↪year, month, and day
df['date'] = pd.to_datetime(df['date'])

# Extract the year from the 'date' column and create a new column 'year'
# This is useful for analyzing data on a yearly basis, such as identifying␣
↪yearly trends and to make possible predictions
df['year'] = df['date'].dt.year

# Extract the month from the 'date' column and create a new column 'month'
# This allows for monthly analysis, which can be important for understanding␣
↪seasonal or monthly patterns
df['month'] = df['date'].dt.month

# Extract the day from the 'date' column and create a new column 'day'
# Analyzing data on a daily basis can help in identifying daily patterns or␣
↪outliers
df['day'] = df['date'].dt.day

# Drop the original 'date' column as it's no longer necessary after extracting␣
↪year, month, and day
# This helps in simplifying the dataset and focusing on the extracted features␣
↪for analysis
df.drop(columns=['date'], inplace=True)
```

[7]:
```
# Now that we've extracted the new columns, we need to recheck the columns for␣
↪the number of tuples, their datatypes, and values.
# The info function will give us more insight on these questions
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13060 entries, 0 to 13059
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  13060 non-null  int64
 1   hour                13060 non-null  int64
 2   season              13060 non-null  int64
 3   is_weekend          13060 non-null  int64
 4   is_holiday          13060 non-null  int64
 5   temperature         13060 non-null  float64
 6   temperature_feels   13060 non-null  float64
 7   humidity            13060 non-null  float64
 8   wind_speed          13060 non-null  float64
 9   weather_code        13060 non-null  int64
 10  bike_rented         13060 non-null  object
 11  year                13060 non-null  int32
 12  month               13060 non-null  int32
 13  day                 13060 non-null  int32
dtypes: float64(4), int32(3), int64(6), object(1)
memory usage: 1.2+ MB
```

[8]:
```
# We can now see that all of the independent variables/columns in this dataset␣
↪are numerical.
# To gain more statistical analysis of the values, we can use describe()␣
↪function
# It will show the count, mean, standard deviation, min and max values, 25th,␣
↪50th, and 75th percentiles
# We can create boxplots for the continuous numerical data columns␣
↪(temperature, temperature_feels, humidity, wind_speed) to
# visualise the data below
#
df.describe()
```

[8]:

| | id | hour | season | is_weekend | is_holiday \ |
|---|---|---|---|---|---|
| count | 13060.000000 | 13060.000000 | 13060.000000 | 13060.000000 | 13060.000000 |
| mean | 8699.206891 | 11.497320 | 1.488974 | 0.285835 | 0.021822 |
| std | 5008.757529 | 6.920567 | 1.118707 | 0.451828 | 0.146109 |
| min | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 4365.500000 | 6.000000 | 0.000000 | 0.000000 | 0.000000 |
| 50% | 8697.500000 | 12.000000 | 1.000000 | 0.000000 | 0.000000 |
| 75% | 13038.250000 | 18.000000 | 2.000000 | 1.000000 | 0.000000 |

```
max    17414.000000     23.000000      3.000000      1.000000      1.000000

        temperature  temperature_feels      humidity    wind_speed  \
count  13060.000000       13060.000000  13060.000000  13060.000000
mean      12.460784          11.512506     72.403407     15.885094
std        5.573487           6.618541     14.264575      7.883711
min       -1.500000          -6.000000     20.500000      0.000000
25%        8.000000           6.000000     63.000000     10.000000
50%       12.500000          12.500000     75.000000     15.000000
75%       16.000000          16.000000     83.000000     20.500000
max       34.000000          34.000000    100.000000     56.500000

       weather_code          year         month           day
count  13060.000000  13060.000000  13060.000000  13060.000000
mean       2.716309   2015.506738      6.516233     15.746861
std        2.348494      0.507574      3.442588      8.783426
min        1.000000   2015.000000      1.000000      1.000000
25%        1.000000   2015.000000      4.000000      8.000000
50%        2.000000   2016.000000      7.000000     16.000000
75%        3.000000   2016.000000     10.000000     23.000000
max       26.000000   2017.000000     12.000000     31.000000
```

[9]:

```python
# Checking if the null values are present in the dataset
df.isnull()
```

[9]:

```
          id   hour  season  is_weekend  is_holiday  temperature  \
0      False  False   False       False       False        False
1      False  False   False       False       False        False
2      False  False   False       False       False        False
3      False  False   False       False       False        False
4      False  False   False       False       False        False
…        …     …      …          …           …            …
13055  False  False   False       False       False        False
13056  False  False   False       False       False        False
13057  False  False   False       False       False        False
13058  False  False   False       False       False        False
13059  False  False   False       False       False        False

       temperature_feels  humidity  wind_speed  weather_code  bike_rented  \
0                  False     False       False         False        False
1                  False     False       False         False        False
2                  False     False       False         False        False
3                  False     False       False         False        False
4                  False     False       False         False        False
…                    …         …           …             …            …
13055              False     False       False         False        False
13056              False     False       False         False        False
```

```
13057               False     False       False         False        False
13058               False     False       False         False        False
13059               False     False       False         False        False

        year  month    day
0      False  False  False
1      False  False  False
2      False  False  False
3      False  False  False
4      False  False  False
…        …     …      …
13055  False  False  False
13056  False  False  False
13057  False  False  False
13058  False  False  False
13059  False  False  False

[13060 rows x 14 columns]
```

[10]:
```
# As the dataset is too large to manually check for null values, we use .sum()␣
↪to add all counts of null values
# We can see that the dataset includes only non-null values
df.isnull().sum()
```

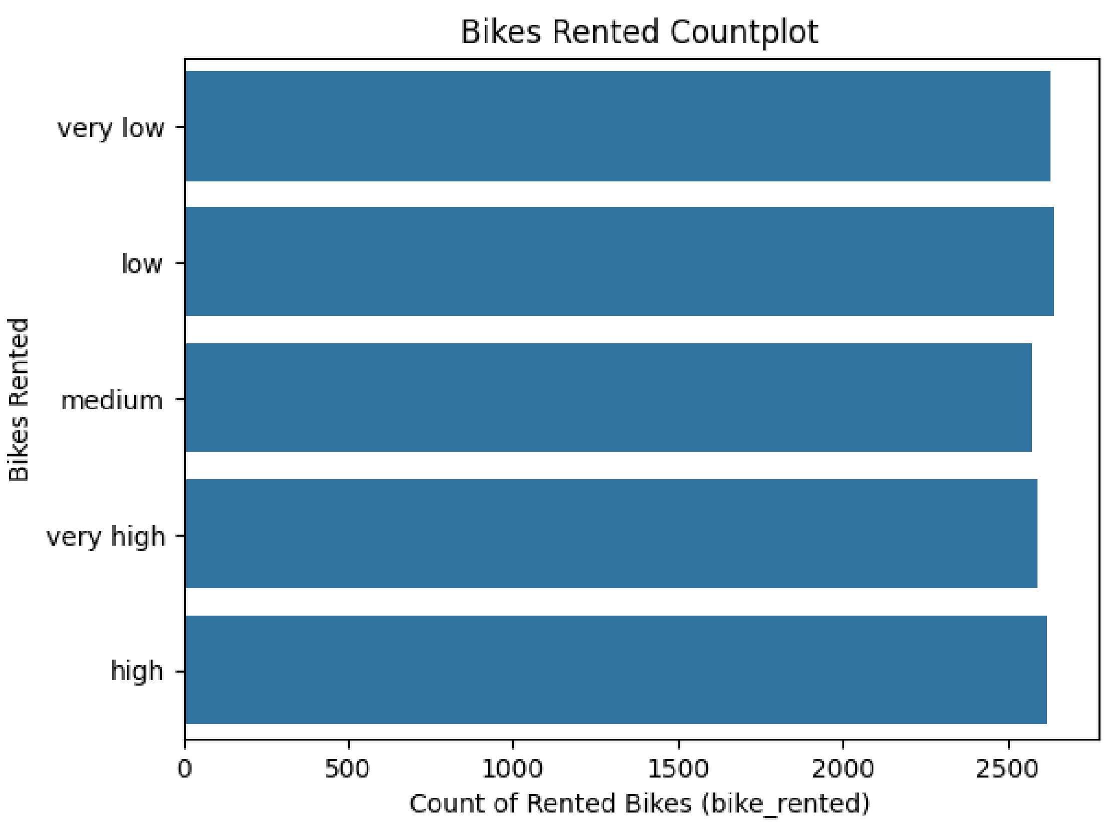
[10]:
```
id                   0
hour                 0
season               0
is_weekend           0
is_holiday           0
temperature          0
temperature_feels    0
humidity             0
wind_speed           0
weather_code         0
bike_rented          0
year                 0
month                0
day                  0
dtype: int64
```

[11]:
```
# Now, as we learned more about the dataset, we should gain more understanding␣
↪about the target variable 'bike_rented'
# This is a categorical column, consisting of 5 categories: low, very low,␣
↪high, very high, medium
# Lets look at the counts of each category in the bike_rented variable
# Setting up the countplot or a bar chart, will further help us understand the␣
↪distribution of the bike_rented table
```

```
df["bike_rented"].value_counts()
```

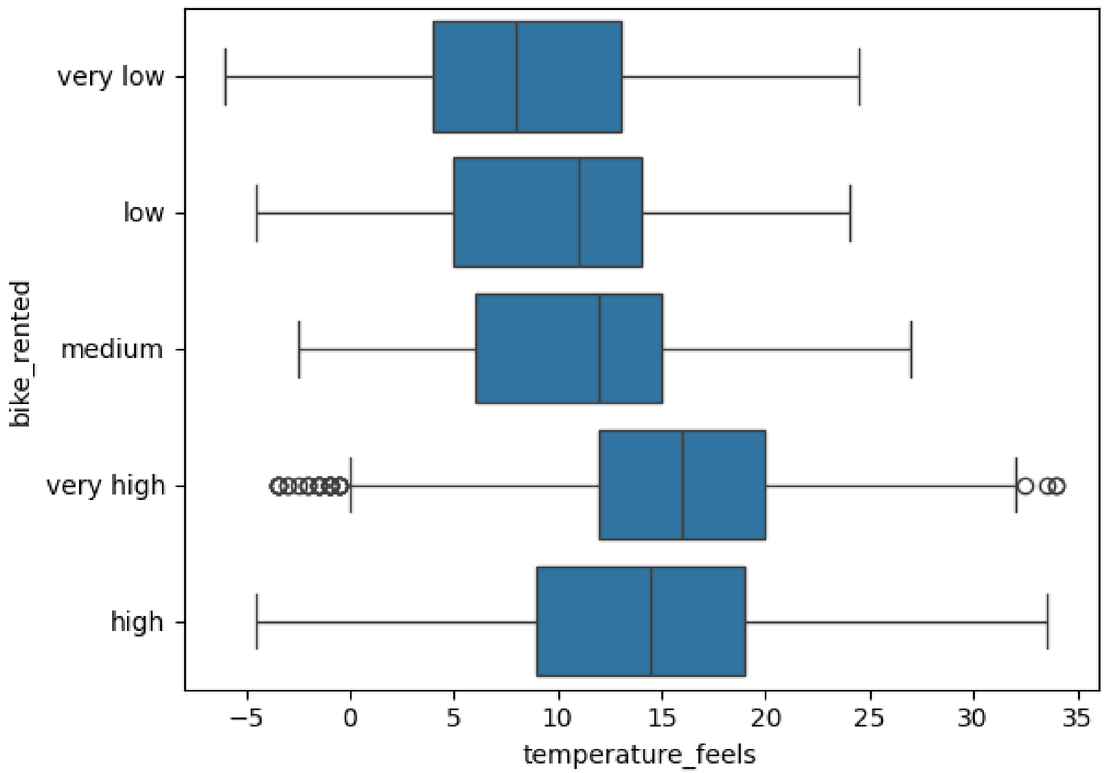
[11]:
```
bike_rented
low          2642
very low     2629
high         2620
very high    2592
medium       2577
Name: count, dtype: int64
```

[12]:
```
#Setting up the countplot of the bike_rented that displays the value of counts
sns.countplot(df["bike_rented"])
plt.ylabel("Bikes Rented")
plt.xlabel("Count of Rented Bikes (bike_rented)")
plt.title("Bikes Rented Countplot")
plt.show()
# As we can see from the countplot, there is a uniform/even distribution of␣
↪categories
# This means, that the bikes are being rented out at a close rate for each␣
↪category
# We can now try to find some correlations between continuous independent␣
↪variables and bikes_rented by using the boxplots
```



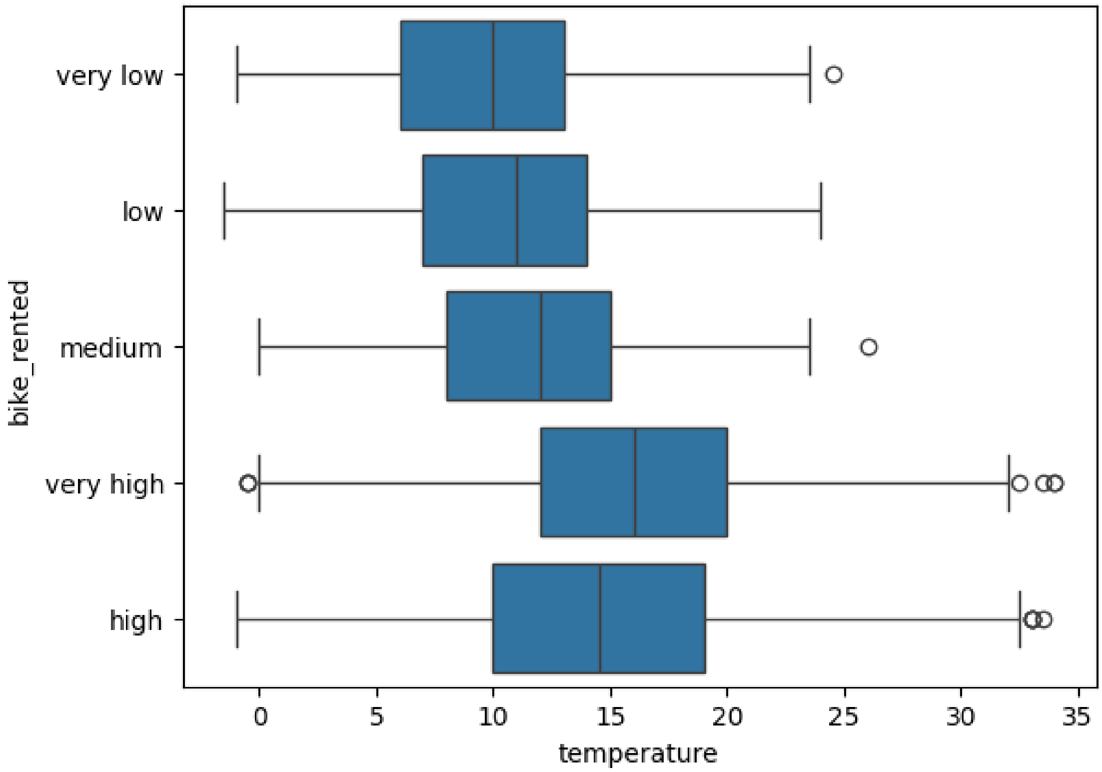
```
[13]: sns.boxplot(y=df["bike_rented"], x=df["temperature_feels"])
# After graphing this boxplot, we can conclude that there is a positive correlation between temperature feels and the number of
# bikes rented
# However, as we know, we cannot imply causation just because there is correlation, so we have to analyze more data
# Therefore, we can graph boxplots for temperature, humidity, and wind_speed
```

```
[13]: <Axes: xlabel='temperature_feels', ylabel='bike_rented'>
```



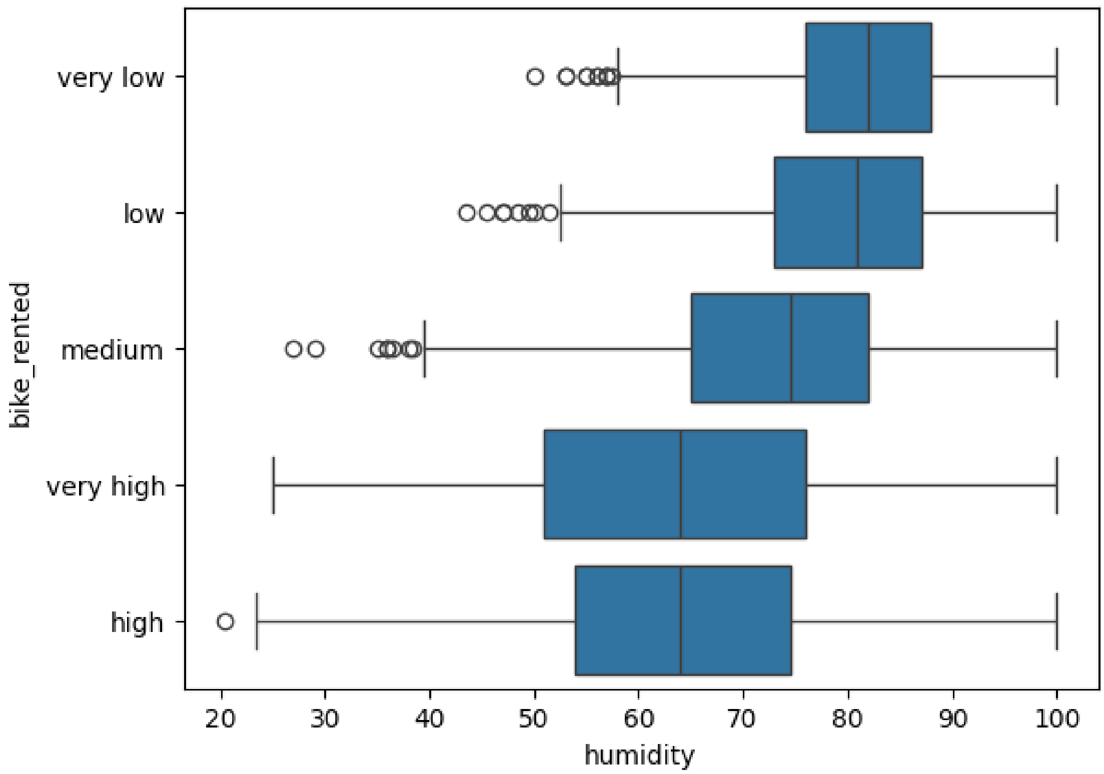

```
[14]: sns.boxplot(y=df["bike_rented"], x=df["temperature"])
# After graphing this boxplot, we can see similar results compared to the previous graph.
# This is happening because temperature_feels values are caused with the actual temperature values
# Let's make more boxplots, to gain more insights
```

```
[14]: <Axes: xlabel='temperature', ylabel='bike_rented'>
```



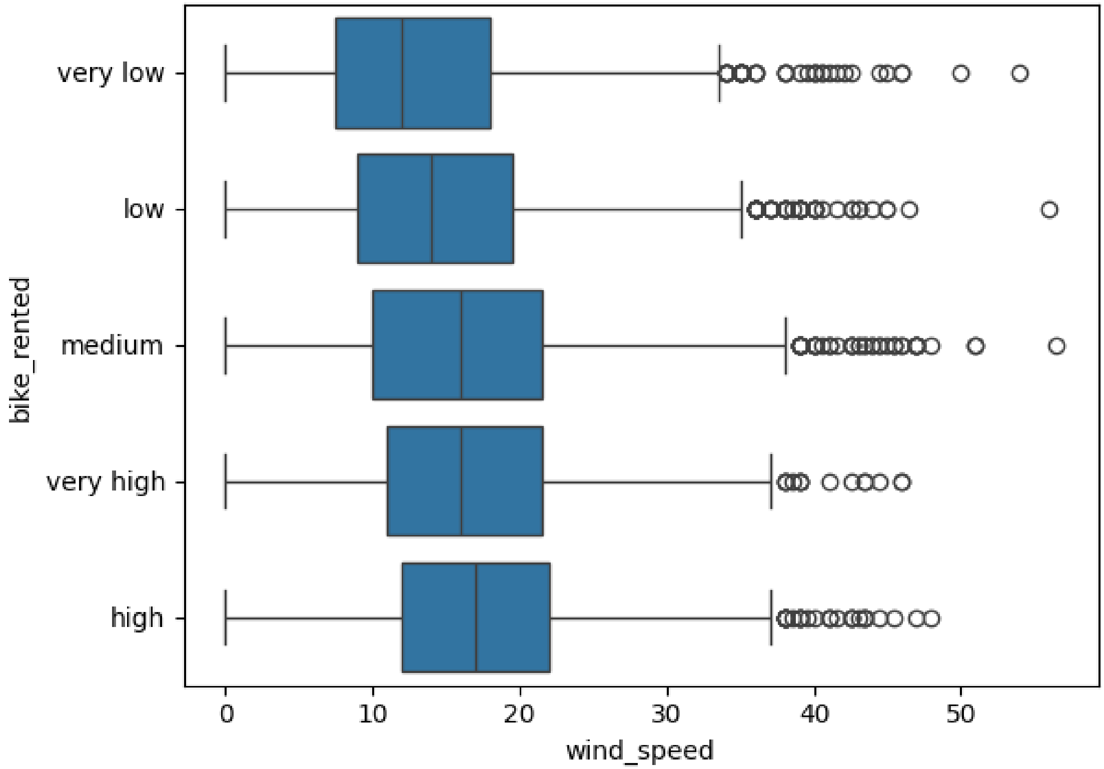

[15]:
```
sns.boxplot(y=df["bike_rented"], x=df["humidity"])
# This boxplot shows that there are more bikes rented, when the humidity is␣
↪lower
# There are some outliers for the very low/low/medium categories, however these␣
↪numbers do not significantly affect the other categories
# The humidity still shows a strong correlation, thus still making it an␣
↪important feature
# Let's move on to graph the wind_speed x bike_rented boxplot
```

[15]:
```
<Axes: xlabel='humidity', ylabel='bike_rented'>
```

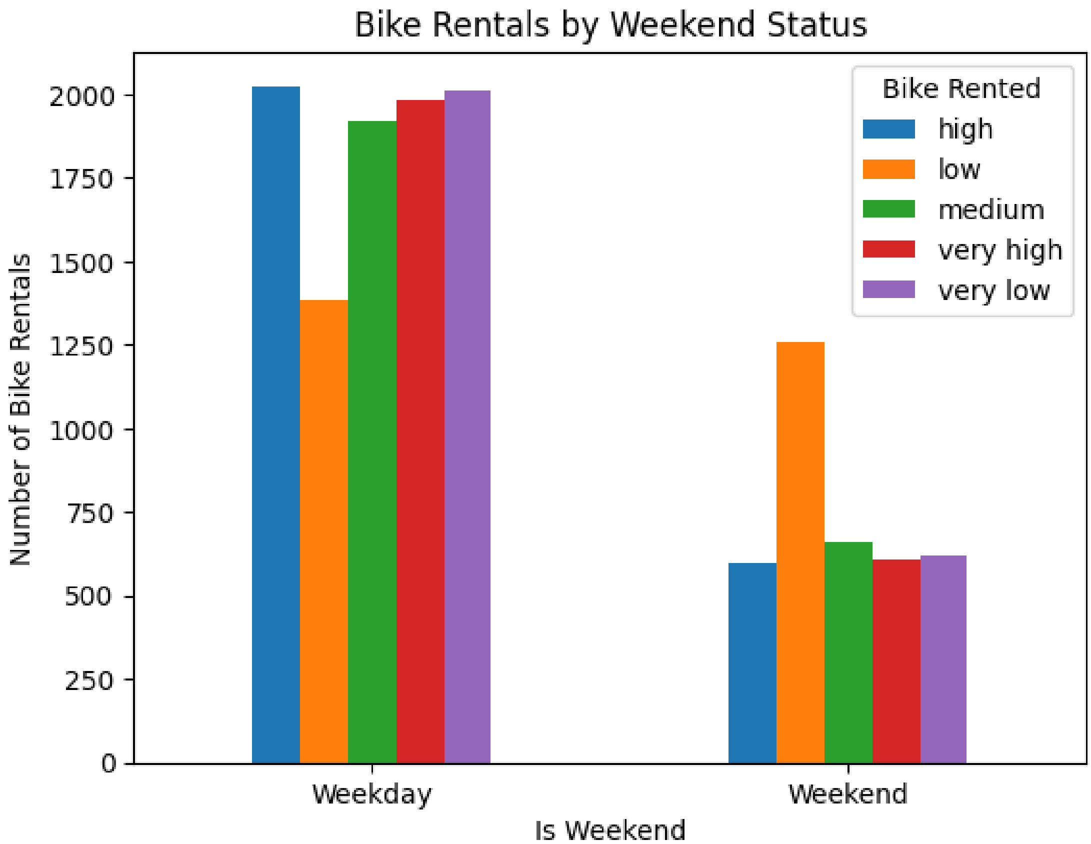



[16]:
```
sns.boxplot(y=df['bike_rented'], x=df['wind_speed'])
# There is still correlation in the graph.
# It is a positive correlation, with higher wind speeds indicating higher␣
↪number of bikes rented
# This boxplot has a very high number of outliers, which we cannot ignore
# It might have a strong negative impact on the accuracy of the model.
# Because of those reasons, this column might be dropped later after we feature␣
↪the importances and try to improve the accuracy of
# the model
# Let's now move on to create
```

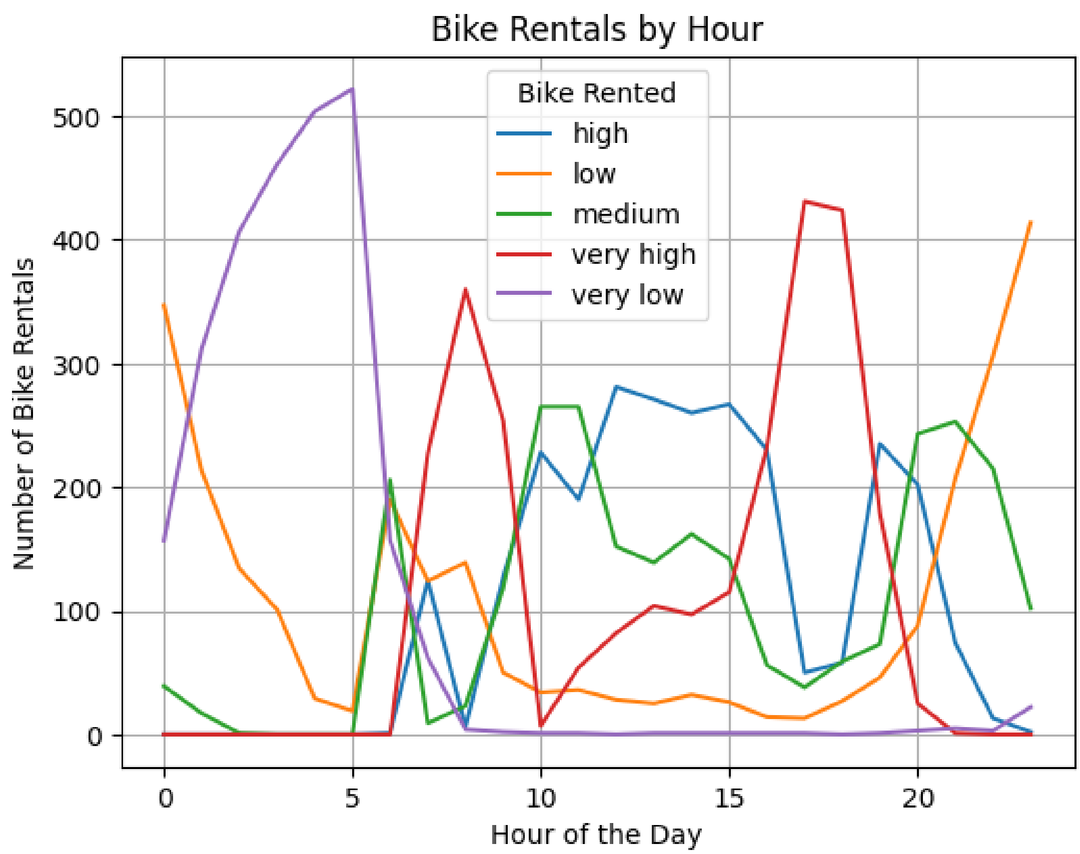
[16]: <Axes: xlabel='wind_speed', ylabel='bike_rented'>




[17]:
```
# Creating a new dataframe that aggregates bike rentals by the 'is_weekend'␣
↪attribute
bike_rentals_weekend = df.groupby('is_weekend')['bike_rented'].value_counts().
↪unstack(fill_value=0)

# Plotting the bar chart
bike_rentals_weekend.plot(kind='bar', stacked=False)
plt.title('Bike Rentals by Weekend Status')
plt.xlabel('Is Weekend')
plt.ylabel('Number of Bike Rentals')
plt.xticks(ticks=[0, 1], labels=['Weekday', 'Weekend'], rotation=0)
plt.legend(title='Bike Rented')
plt.show()
# Now we can see that that bike_rented are low on the weekends, and tend to be␣
↪rented more
```




[18]:
```
# Creating a line plot for bike_rented value counts by hour
bike_rentals_hourly = df.groupby('hour')['bike_rented'].value_counts().
 ↪unstack(fill_value=0)

bike_rentals_hourly.plot(kind='line')
plt.title('Bike Rentals by Hour')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Bike Rentals')
plt.legend(title='Bike Rented')
plt.grid(True)
plt.show()

# We can see that the highest number of bikes are rented out approximately␣
 ↪between 7 AM - 8 AM and 4 PM - 6 PM
# This could be like that because people are likely to use bikes during peak␣
 ↪hours to commute to jobs
# We can see that the hours have a very high importance on the bike_rented value
```




[19]:
```
# Splitting the dataframe 'df' into features (X) and target variable (y)

# Selecting features for the model
# The first part, df.iloc[:, :10], selects all rows and the first 10 columns of␣
↪the dataframe 'df'
# The second part, df.iloc[:, 11:], selects all rows and columns starting from␣
↪the 12th column to the end
# These two parts are then joined together to form the features dataframe 'X',␣
↪excluding the 'bike_rented' column (which is the 11th column)
X = df.iloc[:, :10].join(df.iloc[:, 11:])

# Selecting the target variable for the model
# df.iloc[:, 10] selects all rows and only the 11th column ('bike_rented'),␣
↪which is our target variable
# This column is stored in 'y', which will be used as the target variable for␣
↪model training
y = df.iloc[:, 10]
```

```
# The 'X' dataframe now contains the features that will be used to predict the␣
↪target variable 'y'
# The 'y' series contains the labels that our model will try to predict
# This setup is a common practice in supervised learning tasks, where 'X' is␣
↪used for training/testing the model, and 'y' is what we aim to predict
```

[20]:
```
#Checking the X.shape to confirm the number of rows and columns to ensure that␣
↪the data was split correctly
X.shape
```

[20]:
```
(13060, 13)
```

[21]:
```
#Checking the y.shape to confirm the number of rows and columns to ensure that␣
↪the data was split correctly
y.shape
```

[21]:
```
(13060,)
```

[22]:
```
# Importing the train_test_split function from the sklearn library
from sklearn.model_selection import train_test_split

# Splitting the dataset into training and testing sets
# X represents the features, y represents the target variable
# random_state is set to ensure reproducibility of the results
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 99)
```

[23]:
```
# Importing the RandomForestClassifier class from the sklearn.ensemble module
from sklearn.ensemble import RandomForestClassifier

# Creating an instance of the RandomForestClassifier Model with specific these␣
↪parameters:
# criterion = "gini" specifies the function to measure the quality of a split.
# max_depth = 8 limits the maximum depth of the tree to prevent overfitting.
# min_samples_split = 10 requires at least 10 samples to split an internal node.
# random_state = 99 ensures that the splits that you generate are reproducible.␣
↪Specific random states ensure the same results are generated each time the␣
↪code is run.
clf = RandomForestClassifier(criterion = "gini",
                             max_depth = 8,
                             min_samples_split = 10,
                             random_state = 99)
```

[24]:
```
# Now fit the model
clf.fit(X_train, y_train)
```

[24]:
```
RandomForestClassifier(max_depth=8, min_samples_split=10, random_state=99)
```

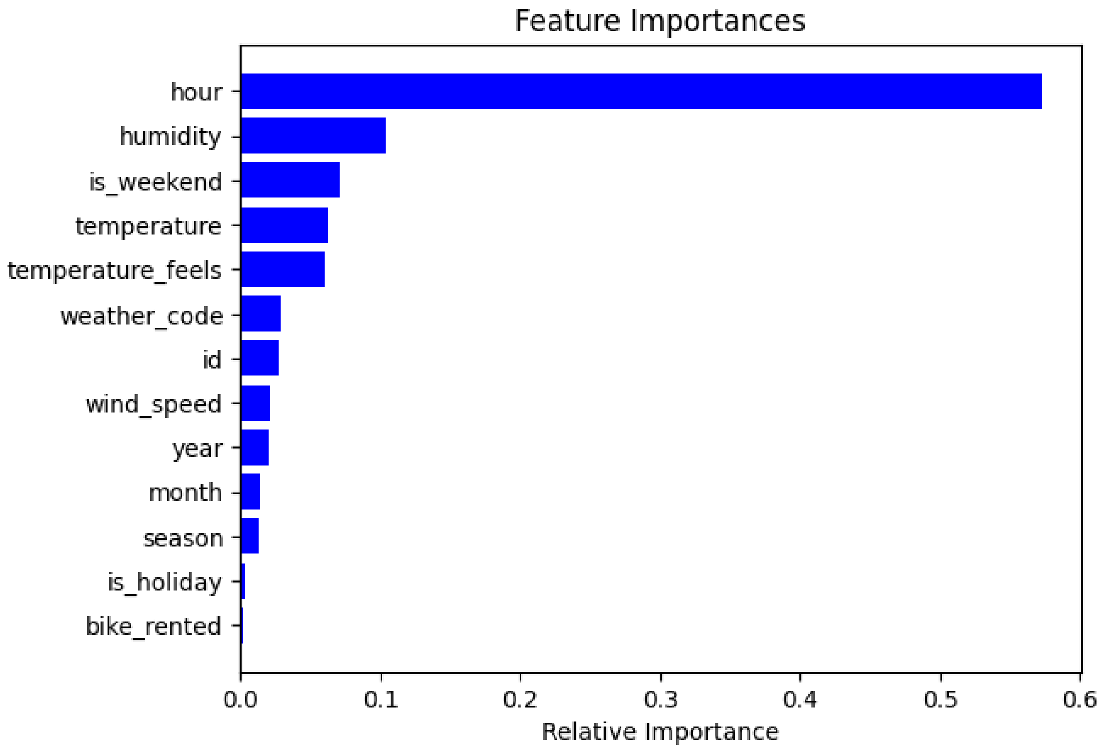
[25]:
```
# Now we are accessing the feature importances of the trained␣
↪RandomForestClassifier model
# This attribute returns an array of values indicating the importance of each␣
↪feature in the model
# Higher values indicate more important features in predicting the target␣
↪variable
clf.feature_importances_
```

[25]:
```
array([0.02726125, 0.57253004, 0.01287229, 0.07176153, 0.00282735,
       0.06262975, 0.060253  , 0.10398649, 0.02104123, 0.02875959,
       0.00233618, 0.02003345, 0.01370785])
```

[26]:
```
# Displaying the columns in an array format.
# However, a better visualisation would be more helpful
df.columns
```

[26]:
```
Index(['id', 'hour', 'season', 'is_weekend', 'is_holiday', 'temperature',
       'temperature_feels', 'humidity', 'wind_speed', 'weather_code',
       'bike_rented', 'year', 'month', 'day'],
      dtype='object')
```

[27]:
```
# Extract column names from the dataframe
features = df.columns
# Get the feature importances from the classifier
importances = clf.feature_importances_
# Get the indices that would sort the importances array
indices = np.argsort(importances)

# Set the title of the plot
plt.title('Feature Importances')
# Create a horizontal bar chart to display the feature importances
plt.barh(range(len(indices)), importances[indices], color = 'b', align =␣
↪'center')
# Set the y-ticks to be the feature names, in the order of their importance
plt.yticks(range(len(indices)), [features[i] for i in indices])
# Label the x-axis as "Relative Importance"
plt.xlabel("Relative Importance")
# Display the plot
plt.show()
```




[28]:
```
# Making predictions based on unseen X_test data with the trained clf model. It␣
↪includes the 20% of the total data,
# as the training was done on the other 80%, which is usually a standard
y_pred = clf.predict(X_test)
```

[29]:
```
# Outputting the predictions in an array
y_pred
```

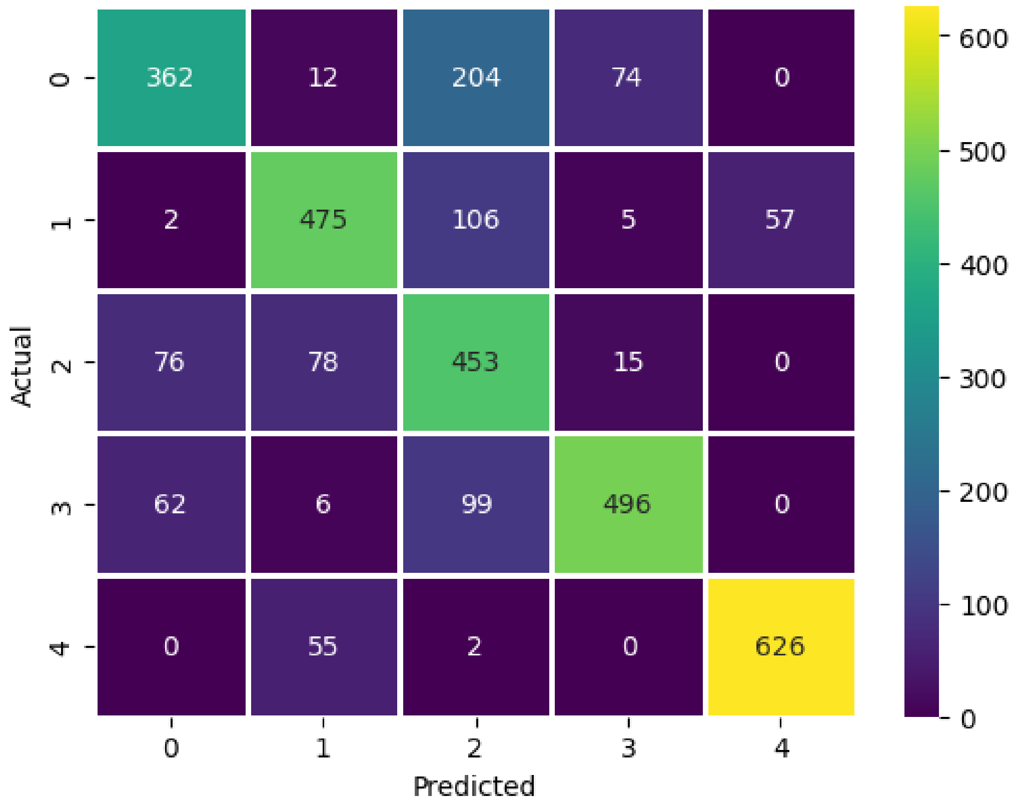
[29]:
```
array(['very low', 'medium', 'medium', …, 'very high', 'very high',
       'medium'], dtype=object)
```

[30]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

[30]:
```
array([[362,  12, 204,  74,   0],
       [  2, 475, 106,   5,  57],
       [ 76,  78, 453,  15,   0],
       [ 62,   6,  99, 496,   0],
       [  0,  55,   2,   0, 626]], dtype=int64)
```

[31]:
```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='viridis', fmt='d', linewidths=1)
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.show()
# The confusion matrix visually shows the performance of the classification␣
↪model.
# Each cell in the matrix represents the number of predictions for each actual␣
↪and predicted class combination.
# The diagonal cells represent correct predictions, while off-diagonal cells␣
↪indicate misclassifications.
```




[32]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

[32]:
```
0.7387442572741194
```

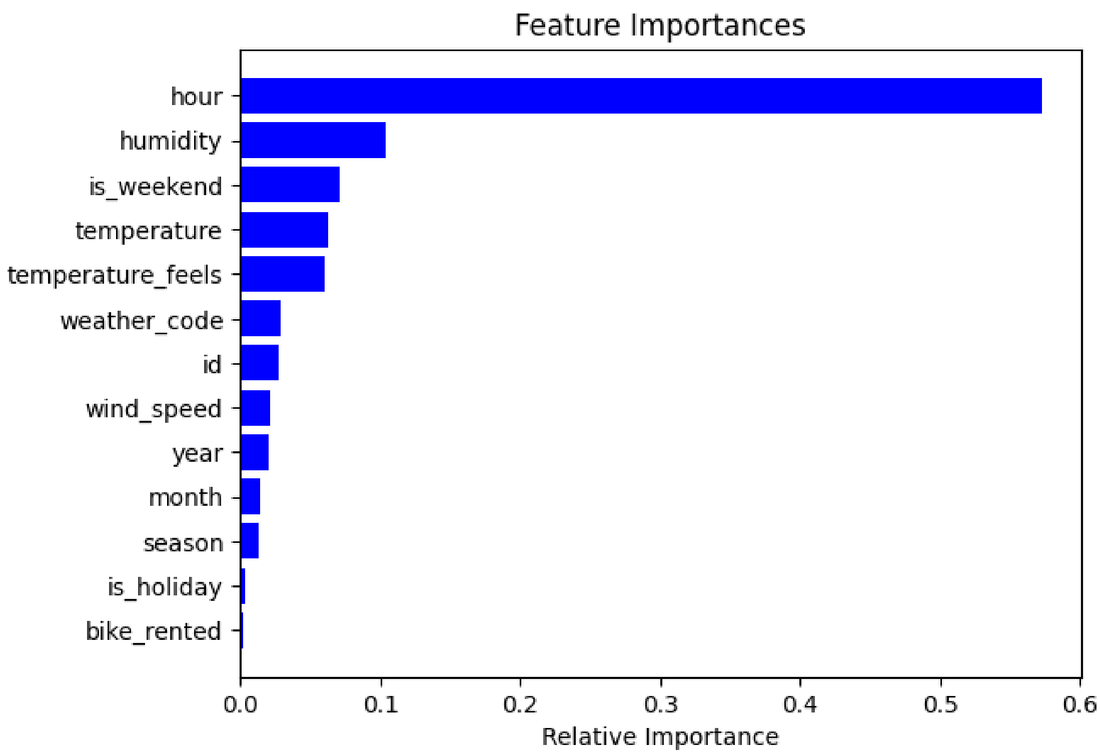
[33]:
```
from sklearn.model_selection import cross_val_score
# We are doing the K-Fold Cross Validation with 10 folds, as it is recommended␣
↪for the larger datasets (more than 10,000 entries)
# This would reduce the impact of individual folds on the model accuracy.
# This line of code, prints the cross validation scores for each fold.
cross_val_score(clf, X_train, y_train, cv=10)
```

[33]: array([0.76428571, 0.73979592, 0.70102041, 0.72346939, 0.73469388,
        0.69969356, 0.72727273, 0.72318693, 0.73033708, 0.73953013])

[34]:
```python
from sklearn.metrics import classification_report
# Here we get the classification report for our model
# We can see that the weighted avg accuracy is 0.75, meaning that the model␣
↪correctly predicted the class in 3/4 cases based
# on the attributes
print(classification_report(y_pred, y_test))
# The model was very precise and with a high recall in predicting the low␣
↪category of bikes rented in London in each day and hour
```

```
              precision    recall  f1-score   support

        high       0.56      0.72      0.63       502
         low       0.74      0.76      0.75       626
      medium       0.73      0.52      0.61       864
   very high       0.75      0.84      0.79       590
    very low       0.92      0.92      0.92       683

    accuracy                           0.74      3265
   macro avg       0.74      0.75      0.74      3265
weighted avg       0.75      0.74      0.74      3265
```

[35]:
```python
# Extract column names from the dataframe
features = df.columns
# Get the feature importances from the classifier
importances = clf.feature_importances_
# Get the indices that would sort the importances array
indices = np.argsort(importances)

# Set the title of the plot
plt.title('Feature Importances')
# Create a horizontal bar chart to display the feature importances
plt.barh(range(len(indices)), importances[indices], color = 'b', align =␣
↪'center')
# Set the y-ticks to be the feature names, in the order of their importance
plt.yticks(range(len(indices)), [features[i] for i in indices])
# Label the x-axis as "Relative Importance"
plt.xlabel("Relative Importance")
# Display the plot
plt.show()
```




[36]:
```
# Although, we were happy with the results, we want to experiment on the model
↪by dropping low importance attributes and then using a random forest
↪classifier model.
df.drop(columns = ['weather_code', 'id', 'wind_speed', 'year', 'month',
↪'season', 'is_holiday'], inplace=True)
```

[37]:
```
# As we can see, the columns were dropped
# Let's construct a Random Forest Classifier Model
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13060 entries, 0 to 13059
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   hour               13060 non-null  int64
 1   is_weekend         13060 non-null  int64
 2   temperature        13060 non-null  float64
 3   temperature_feels  13060 non-null  float64
 4   humidity           13060 non-null  float64
 5   bike_rented        13060 non-null  object
 6   day                13060 non-null  int32
dtypes: float64(3), int32(1), int64(2), object(1)
```

```
memory usage: 663.3+ KB
```

[38]:
```
X = df.iloc[:, :5].join(df.iloc[:, 6:])
y = df.iloc[:, 5]
# The 'X' dataframe now contains the features that will be used to predict the␣
↪target variable 'y'
# The 'y' series contains the labels that our model will try to predict
```

[39]:
```
# Splitting the dataset into training and testing sets
# X represents the features, y represents the target variable
# random_state is set to ensure reproducibility of the results
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 99)
```

[40]:
```
# Now fit the new model
clf.fit(X_train, y_train)
```

[40]:
```
RandomForestClassifier(max_depth=8, min_samples_split=10, random_state=99)
```

[41]:
```
# Making predictions with a new model
y_pred = clf.predict(X_test)
```

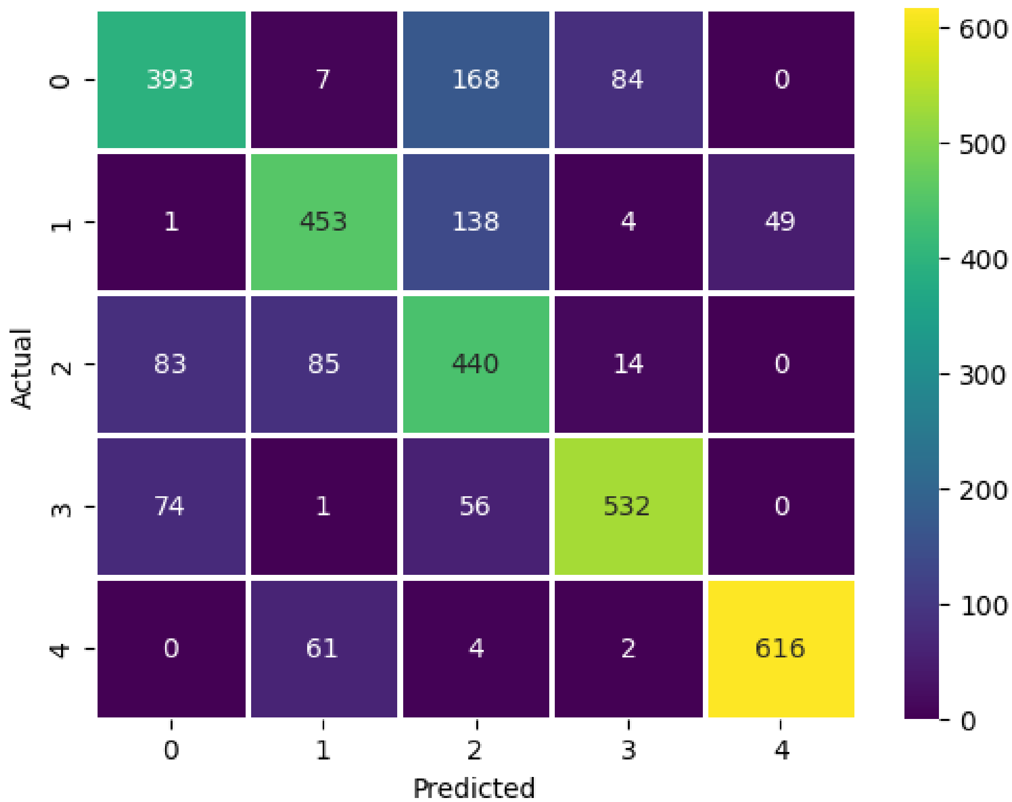
[42]:
```
# Printing the predictions of new model
y_pred
```

[42]:
```
array(['very low', 'very high', 'medium', …, 'very high', 'very high',
       'low'], dtype=object)
```

[43]:
```
# Printing the confusion matrix of the new model
# The differences with the initial confusion matrix will be discussed along␣
↪with the proper visualisation
confusion_matrix(y_test, y_pred)
```

[43]:
```
array([[393,   7, 168,  84,   0],
       [  1, 453, 138,   4,  49],
       [ 83,  85, 440,  14,   0],
       [ 74,   1,  56, 532,   0],
       [  0,  61,   4,   2, 616]], dtype=int64)
```

[44]:
```
# Visualising the new confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='viridis', fmt='d', linewidths=1)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
# When comparing to the initial confusion matrix, we can see that the model␣
↪became more accurate in predicting the
# values for very high and high categories, while the predictions became less␣
↪precise for low and very low categories
```




```
[45]: accuracy_score(y_test, y_pred)
      # After printing the accuracy score, we see that dropping non-important␣
       ↪features did not make a signicant impact on the
      # accuracy of the model
```

```
[45]: 0.7454823889739663
```

```
[46]: # Checking the cross validation scores for the new model.
      # No significant differences from the initial model
      cross_val_score(clf, X_train, y_train, cv=10)
```

```
[46]: array([0.77040816, 0.77653061, 0.72244898, 0.7255102 , 0.73877551,
             0.73135853, 0.76404494, 0.75280899, 0.75280899, 0.75383044])
```

```
[47]: # In the classification report, we can see some minor changes in predictions␣
       ↪for each class
      # For example, the precision for high and very high classes, has increased,
      # while, the precision for very low, low, and medium classes has dropped.
      # Overall accuracy did not significantly change
      print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support
```

| | | | | |
|---|---|---|---|---|
| high | 0.60 | 0.71 | 0.65 | 551 |
| low | 0.70 | 0.75 | 0.72 | 607 |
| medium | 0.71 | 0.55 | 0.62 | 806 |
| very high | 0.80 | 0.84 | 0.82 | 636 |
| very low | 0.90 | 0.93 | 0.91 | 665 |
| accuracy | | | 0.75 | 3265 |
| macro avg | 0.74 | 0.75 | 0.75 | 3265 |
| weighted avg | 0.75 | 0.75 | 0.74 | 3265 |

Conclusion: The hour, day of the week, and temperature had the most impact on the prediction model. Therefore, we confirmed our hypothesis. In addition to that, we discovered that the Random Forest Classifier model is working efficiently on larger (categorical) datasets and gained valuable insights about the database itself. It was also discovered, that dropping non-important features in this given case did not significantly impact the prediction results of the model. Therefore, we can conclude that in this case, the outliers and features with low correlation values did not have a strong impact on the model and that the model was resistant to these kind of flaws in the database.

# Conclusion

In conclusion, our findings of Logistic Regression and Random Forest Classifiers for predicting bike usage in London's Santander Cycles program yielded valuable insights about the number of bikes rented each hour daily. By leveraging k-fold cross-validation, we evaluated the performance of both models and assessed their suitability for this task.

Logistic regression and random forest classification revealed valuable insights into predicting bike usage in London's Santander Cycles program. Adding "hour" as a feature in Logistic Regression yielded a significant 7% accuracy increase, highlighting its crucial role. Other features, like "day," did not have a strong impact. Dropping the feature is_weekend increased the accuracy of the model by 2%. Random Forest confirmed the predicted importance of "hour," "day of week," and "temperature," highlighting its effectiveness with potentially large datasets. It also demonstrated resilience to complex data, as dropping features with low importance did not significantly impact the precision and accuracy. Overall, both models offer promising potential. Logistic Regression provides interpretability, while Random Forest offers higher accuracy and robustness. Further exploration could involve comparing additional metrics, analysing other potential features, and fine-tuning hyperparameters for better performance. These findings contribute to building a robust prediction system for London's Santander Cycles program, ultimately aiding optimized bike distribution and a more efficient transportation system.

This knowledge can empower Tfl to make informed decisions regarding bike rebalancing strategies, ensuring cyclists a comfortable experience while optimizing resource allocation for the company. Additionally, the interpretability of Logistic Regression and precision of the Random Forest Classifier highlights the key factors influencing bike usage and rent.