



Data Science Essentials (CIA2C14)

Assignment 2

Fill in your personal data below

Name: Ho Wei Bin

Admin No:2202561C

Class:P03

Declaration of Originality

"I am the originator of this work and I have appropriately acknowledged all other original sources used as my reference for this work.

I understand that Plagiarism is the act of taking and using the whole or any part of another person's work, including work generated by AI, and presenting it as my own.

I understand that Plagiarism is an academic offence and if I am found to have committed or abetted the offence of plagiarism in relation to this submitted work, disciplinary action will be enforced."

Please sign your name or insert a picture of your signature in the markdown cell below.
(double click on the cell)

Ho wei Bin

1. This deliverable is worth 45% of the final marks for DSES. The total mark is 90. There are two submissions required- Part I and Part II The due dates are **10 Jul and 14 Aug 2023, 09:00 am.**

2. For Part I submission, you need to complete Task 1 Data Preparation and Task 2: Data Understanding.

3. For Part II submission, you need to complete Task 3: Data Modelling and Task 4: Recommendations & Conclusion.

4. Note that you should still keep your codes for Part I for tutor to evaluate Part II.

5. This is an individual deliverable. You are required to produce your own work.

6. You are allowed to use packages (libraries) not covered in the lecture slides or practicals (e.g. scikit-learn, seaborn, etc.)

7. Apart from using Markdown cells to include your description, takeaways, and comments, provide sufficient comments to your codes too.

8. Use random_state= Your Admin Number without the letter wherever needed.

e.g. random_state=2200212

9. Name your notebook according to the following format.

e.g. YourName_2200001A_Assignment2

10. You need to submit your deliverable, i.e. this Jupyter Notebook (.ipynb format) for marking.

Project Background & Aims

The project is regarding a bicycle rental company originating from Japan. The company name is Ohaiyo. Due to the spread of the COVID 19 pandemic, people have begin to rediscover places of beauty and interest in their backyard. This encourage people to use public transport and "slower methods" of travel. One of which is cycling. Knowing this, Ohaiyo wishes to take the opportunity to purchase more bicycle , increase its manpower and expand its business capabilities. However , such major decision that involves high amount of capital and other resources requires data driven evidence. As such, we are task to handle the data in the industry famous crisp-dm framework to hopefully give Ohaiyo and its stakeholder sufficient evidence to advise them on their next business decision. The project aims to breakdown the datam, vizualize and storytell to our audience and give them valuable insights as to advise them on what to do and where to allocate their assets. We also hope to use machine learning and data modelling to do predictive analysis to further solidify our evidence, all in the name to ensure that Ohaiyo and its stakeholder gains as much revenue as possible

Data Preparation

Data preparation will normally revolve around any steps to transform raw data into clean and useful data. These data will be ready to be send into a machine learning algorithm to build dashboard to highlight useful insights and build predictiv modelling. Step 1 is to do data wrangling and identify and fix certain issues such as Duplications, Inconsistencies, Missing values, data not tidy or in the correct type. For data that we know we will be using for machine learning that is called target. In this case it is the Rented_Bike_Count and hit sales column, we will not be altering values in

the column. we can only drop columns with missing values. Other method of filling missing values include replacing it with mean, mode and median. For a column that has consecutive and recurring values like time, I will be using Forward fill to remove the null value.



I have some missing data in a dataframe. it has got a sales column that states how many product was sold that day. and a few columns that provides the environmental information that may or may not affect the sales for the day. I have got some missing values for the sales column. this data will be use for future data modelling and machine learning. Do you have any suggestion on how I can deal with the rows with missing values?



When dealing with missing values in your sales column, there are several approaches you can consider. The choice of method depends on the nature of your data and the specific requirements of your analysis. Here are a few common strategies to handle missing values:

1. **Delete rows with missing values:** If the missing values are relatively small in number and randomly distributed, you may choose to remove the rows with missing sales values. However, be cautious with this approach as it can lead to loss of valuable information.
2. **Imputation with mean/median:** Replace the missing sales values with the mean or median of the available sales data. This approach assumes that the missing values are missing at random and the mean/median is a reasonable estimate for the missing values.
3. **Imputation with regression:** Use regression models to predict the missing sales values based on the available environmental information. Fit a regression model using the non-missing sales data as the dependent variable and the other environmental factors as predictors. Then, use this model to predict the missing sales values.
4. **Multiple imputation:** If you have multiple environmental factors that may affect sales, you can use multiple imputation techniques to fill in the missing values. Multiple imputation creates multiple plausible imputations based on the observed data and the relationships among variables, allowing for uncertainty estimation in subsequent analyses.
5. **Time series methods:** If your data has a time component, you can consider using time series methods to fill in missing sales values. Techniques like interpolation, seasonal decomposition, or ARIMA models can be applied to estimate missing values based on the temporal patterns in the data.

It's important to carefully consider the nature of your data, the amount of missing values, and the potential impact of imputation. Recommended to assess the impact of different imputation methods on your results and consider sensitivity analyses to evaluate

At first glance, the suggestion in this conversation looks perfectly usable and valid. However upon closer investigation, there is a couple of reasons why some of the suggestion is not valid here especially for certain columns. For example in point 3, imputation with regression. ChatGPT suggested using the model to predict the missing sales value. That is not viable as the sales value is the target for future data modelling. Target data cannot be altered or modified as to not affect the accuracy of machine learning. Other suggestions are slightly more valid such as the removal of rows as well as substitution of mean and median. Therefore this advise is yellow

Data Cleaning

Importing data

```
In [1]: import pandas as pd
import seaborn as sns

df = pd.read_csv("C:\\\\Users\\\\ciphe\\\\Downloads\\\\Assignment2_Dataset.csv", encoding=
assignment_dataframe=df #renaming the dataset.csv as "assignment_dataframe"
assignment_dataframe #print dataframe
```

Out[1]:

	S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visib
0	1	2018-11-23	JP	196.0	0.0	-1.7	79.0	0.5	
1	2	2018-11-23	JP	250.0	1.0	-1.6	79.0	0.6	
2	3	2018-11-23	JP	161.0	2.0	-1.1	79.0	0.7	
3	4	2018-11-23	JP	99.0	3.0	-0.6	76.0	0.5	
4	5	2018-11-23	JP	64.0	4.0	-0.3	76.0	0.9	
...	
8758	8759	2019-12-01	JP	1860.0	19.0	15.6	44.0	1.5	
8759	8760	2019-12-01	JP	1593.0	20.0	14.5	51.0	1.4	
8760	8761	2019-12-01	JP	1513.0	21.0	13.4	56.0	0.7	
8761	8762	2019-12-01	JP	1360.0	22.0	12.7	60.0	0.0	
8762	8763	2019-12-01	JP	916.0	23.0	12.0	59.0	0.4	

8763 rows × 18 columns

Identify missing values in dataset

```
In [2]: missing_values=assignment_dataframe.isna().sum() #command to know how many missing
missing_values #print command
```

```
Out[2]:      S/N          0
             Date         0
             Country       0
             Rented_Bike_Count 12
             Hour          24
             Temperature    24
             Humidity        12
             Windspeed       0
             Visibility       0
             Dewpoint_Temp    0
             Solar_Radiation   0
             Rainfall        0
             Snowfall        12
             Open            0
             Hit_Sales       20
             Region          0
             Latitude         0
             Longitude        0
             dtype: int64
```

Checking for duplicated values

```
In [3]: duplicate_record = assignment_dataframe.duplicated(subset=['Date', 'Hour'], keep=False)
duplicate_record = assignment_dataframe[duplicate_record]
duplicate_record
```

Out[3]:

	S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visib
788	789	2019-01-01	JP	206.0	20.0	-0.3	40.0	1.2	
2609	2610	2019-11-11	JP	111.0	5.0	6.8	83.0	0.6	
3532	3533	2019-01-01	JP	206.0	20.0	-0.3	40.0	1.2	
5222	5223	2019-07-06	JP	1216.0	NaN	25.7	57.0	1.9	
5223	5224	2019-07-06	JP	1210.0	NaN	26.2	53.0	2.1	
5224	5225	2019-07-06	JP	1209.0	NaN	27.0	52.0	2.1	
7155	7156	2019-09-25	JP	881.0	1.0	29.0	67.0	1.1	
8263	8264	2019-11-11	JP	111.0	5.0	6.8	83.0	0.6	
8384	8385	2019-09-25	JP	881.0	1.0	29.0	67.0	1.1	

Substituting the missing values in hours with ffill since it is in chronological order

```
In [4]: expected_hours = (assignment_dataframe.groupby('Date').cumcount() % 24).astype(int)
assignment_dataframe['Hour'].fillna(expected_hours, inplace=True)
```

Delete the remaining duplicate values. Only delete the records out of chronological order

```
In [5]: rows_to_drop = [3532, 8263, 8384]
assignment_dataframe = assignment_dataframe.drop(rows_to_drop)
```

check for duplicate record

```
In [6]: duplicate_record = assignment_dataframe.duplicated(subset=['Date', 'Hour'], keep=False)
duplicate_record = assignment_dataframe[duplicate_record]
duplicate_record
```

Out[6]: S/N Date Country Rented_Bike_Count Hour Temperature Humidity Windspeed Visibility |

S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility	

Checking if there is any invalid timing for hour

```
In [7]: Unique = assignment_dataframe['Hour'].value_counts()
Unique
```

```
Out[7]: 0.0    365
1.0    365
22.0   365
21.0   365
20.0   365
19.0   365
18.0   365
17.0   365
16.0   365
15.0   365
14.0   365
13.0   365
12.0   365
11.0   365
10.0   365
9.0    365
8.0    365
7.0    365
6.0    365
5.0    365
4.0    365
3.0    365
2.0    365
23.0   365
Name: Hour, dtype: int64
```

Filling in missing value for temperature based of the mean of the corresponding hour

```
In [8]: import pandas as pd

# Calculate the mean temperature for each hour
hourly_mean_temperature = assignment_dataframe.groupby('Hour')[['Temperature']].mean

# Function to fill missing values based on the mean temperature of the corresponding hour
def fill_missing_temperature(row):
    hour = row['Hour']
    if pd.isnull(row['Temperature']):
        return hourly_mean_temperature[hour]
    return row['Temperature']

# Apply the fill_missing_temperature function to fill in the missing values
assignment_dataframe['Temperature'] = assignment_dataframe.apply(fill_missing_temperature, axis=1)
```

Substituting missing value in column snowfall with mode

```
In [9]: # Calculate the mode of the 'snowfall' column
Snowfall_mode = assignment_dataframe['Snowfall'].mode().iloc[0]

# Fill missing values with the mode
assignment_dataframe['Snowfall'].fillna(Snowfall_mode, inplace=True)
```

Fixing humidity

```
In [10]: # Fill missing values with the mean
mean_humidity = assignment_dataframe['Humidity'].mean()
assignment_dataframe['Humidity'].fillna(mean_humidity, inplace=True)
```

Ensure that only missing values left is Rented bike count and hit sales

```
In [11]: missing_values=assignment_dataframe.isna().sum() #command to know how many missing
missing_values #print command
```

```
Out[11]: S/N          0
Date         0
Country      0
Rented_Bike_Count  12
Hour         0
Temperature   0
Humidity      0
Windspeed     0
Visibility    0
Dewpoint_Temp 0
Solar_Radiation 0
Rainfall      0
Snowfall      0
Open          0
Hit_Sales     20
Region        0
Latitude      0
Longitude     0
dtype: int64
```

Checking inconsistency

```
In [12]: Unique = assignment_dataframe['Open'].value_counts()
Unique
```

```
Out[12]: Yes      8455
No       290
n        5
yes      4
Y        4
yes      2
Name: Open, dtype: int64
```

Fixing inconsistencies

```
In [13]: assignment_dataframe['Open'] = assignment_dataframe['Open'].replace({'Y': 'Yes', 'N': 'No'})
print(assignment_dataframe['Open'].value_counts())
```

```
Out[13]: Yes      8465
No       295
Name: Open, dtype: int64
```

Identify row with the highest and lowest value. If it appears as a anomaly. Delete them.

```
In [14]: min_index = assignment_dataframe['Rented_Bike_Count'].idxmin()
lowest_value = assignment_dataframe.loc[min_index, 'Rented_Bike_Count']
max_index = assignment_dataframe['Rented_Bike_Count'].idxmax()
highest_value = assignment_dataframe.loc[max_index, 'Rented_Bike_Count']
print("Row with lowest Rented_Bike_Count:", min_index)
print("Lowest Rented_Bike_Count value:", lowest_value)
print("Row with Rented_Bike_Count:", max_index)
print("Highest Rented_Bike_Count value:", highest_value)
```

Row with lowest Rented_Bike_Count: 2259
 Lowest Rented_Bike_Count value: -2.0
 Row with Rented_Bike_Count: 4868
 Highest Rented_Bike_Count value: 34180.0

Drop column with the anomaly in rented bike count. row 4868 and row 2261 to prevent machine learning inaccuracy

```
In [15]: #Drop the row with index 4868
assignment_dataframe.drop(4868, inplace=True)
#Drop the row with index 2259
assignment_dataframe.drop(2259, inplace=True)
```

If temperature is greater than 56.7 degree which is the highest temperature ever recorded, substitute that value with the mean of the whole column

```
In [16]: import pandas as pd
import numpy as np

mean_value = assignment_dataframe['Temperature'].mean()
assignment_dataframe.loc[assignment_dataframe['Temperature'] > 56.7, 'Temperature'] = mean_value
```

```
In [17]: assignment_dataframe.describe()
```

	S/N	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	V
count	8758.000000	8746.000000	8758.000000	8758.000000	8758.000000	8758.000000	8758
mean	4381.383307	704.725131	11.500228	12.829934	58.234739	1.724732	3
std	2529.701650	644.344779	6.922428	12.071445	20.347739	1.036325	1
min	1.000000	0.000000	0.000000	-44.500000	0.000000	0.000000	1
25%	2190.250000	192.000000	6.000000	3.400000	42.000000	0.900000	3
50%	4381.500000	505.500000	11.500000	13.700000	57.000000	1.500000	4
75%	6571.750000	1066.000000	17.000000	22.500000	74.000000	2.300000	5
max	8763.000000	3556.000000	23.000000	39.400000	98.000000	7.400000	5

Feel free to insert more cells to continue reporting on your Data Cleaning.

Data Transformation (also known as Features Engineering)

Provide the description of how you have used ChatGPT or other internet sources to brainstorm for ideas.



given that in my dataframe. i have temperature, dewpoint temperature, rainfall, humidity, solar radiation, windspeed. what can I calculate with the factors I have?



With the factors you have in your dataframe (temperature, dewpoint temperature, rainfall, humidity, solar radiation, and windspeed), you can calculate various statistical measures and explore relationships between these variables. Here are some possibilities:

1. Descriptive Statistics: Calculate basic descriptive statistics such as mean, median, minimum, maximum, and standard deviation for each variable.
2. Correlation: Examine the correlation between pairs of variables to understand their relationships. You can use correlation coefficients such as Pearson's correlation coefficient or Spearman's rank correlation coefficient.
3. Heat Index: Calculate the heat index using the temperature and humidity values. The heat index is a measure of how hot it feels when relative humidity is factored in with the actual air temperature.
4. Dew Point: Calculate the dew point temperature using the temperature and dewpoint temperature values. The dew point is the temperature at which air becomes saturated and forms dew.
5. Rainfall Analysis: Analyze the rainfall data to calculate statistics such as total rainfall, average rainfall, or rainfall intensity over a specific period.
6. Solar Radiation Analysis: Analyze the solar radiation data to calculate statistics such as total solar radiation, average solar radiation, or the relationship between solar radiation and other variables.
7. Wind Chill: Calculate the wind chill index using the temperature and windspeed values. The wind chill index represents how cold it feels when wind is taken into account.

Send a message



To transform data, I aim to add new columns that can hopefully group or sum up some of the weather attribute to form more valuable insights. Through this insight, we might be able to identify more trends between the weather condition and the level of sales of Ohaiyo. However I won't be taking all of GPT suggestion as it is long and some information might be redundant. Therefore I will be trying out some of the solution and deciding which of its suggest is most suitable

Provide the description, justification and results of your technique(s) here.

Transformation 1

Column added: "Heat Index" Line 4 and 5 is used to calculate relative humidity and convert it to percentage

- **Description: Human body temperature is dependent on the air as it absorbs and removes moisture from our skin to cool us down. If the relative humidity is high, the amount of water evaporating from our skin is limited so we feel warm and stifled.**

- Reason: Through relative humidity, we are able to determine the heat index. Which is another good insight to the environment and it might play a role in the count of rented bike for that day and hour

```
In [18]: import numpy as np
import pandas as pd

# Calculate the relative humidity
assignment_dataframe['Relative_humidity'] = (assignment_dataframe['Humidity'] / assignment_dataframe['Temperature']) ** 0.16666666666666666

# Define the constants for the heat index calculation
c1 = -8.78469475556
c2 = 1.61139411
c3 = 2.33854883889
c4 = -0.14611605
c5 = -0.012308094
c6 = -0.0164248277778
c7 = 0.002211732
c8 = 0.00072546
c9 = -0.000003582

# Convert relative humidity to decimal
relative_humidity_decimal = assignment_dataframe['Relative_humidity'] / 100

# Calculate the heat index for each row
assignment_dataframe['Heat_Index'] = (
    c1 + c2 * assignment_dataframe['Temperature']
    + c3 * relative_humidity_decimal
    + c4 * assignment_dataframe['Temperature'] * relative_humidity_decimal
    + c5 * assignment_dataframe['Temperature'] ** 2
    + c6 * relative_humidity_decimal ** 2
    + c7 * assignment_dataframe['Temperature'] ** 2 * relative_humidity_decimal
    + c8 * assignment_dataframe['Temperature'] * relative_humidity_decimal ** 2
    + c9 * assignment_dataframe['Temperature'] ** 2 * relative_humidity_decimal **
)

# Print the resulting dataframe with the Heat_Index column
assignment_dataframe.head()
```

Out[18]:

	S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility
0	1	2018-11-23	JP	196.0	0.0	-1.7	79.0	0.5	3
1	2	2018-11-23	JP	250.0	1.0	-1.6	79.0	0.6	3
2	3	2018-11-23	JP	161.0	2.0	-1.1	79.0	0.7	3
3	4	2018-11-23	JP	99.0	3.0	-0.6	76.0	0.5	3
4	5	2018-11-23	JP	64.0	4.0	-0.3	76.0	0.9	3

Transformation 2

Column added: "Wind chill"

- **Description:** Wind Chill is a term used to describe what the air temperature feels like to the human skin due to the combination of cold temperatures and winds blowing on exposed skin.
- **Reason:** Using wind chill, we can join the windspeed and temperature together. we can see if both factors combined might be a compelling factor that cause a fluctuation of the sales of the rented bikes

In [19]:

```
import pandas as pd
```

```
# Calculate the wind chill index
assignment_dataframe['Windchill'] = 13.12 + 0.6215 * assignment_dataframe['Temperature']
assignment_dataframe.head()
```

Out[19]:

	S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility
0	1	2018-11-23	JP	196.0	0.0	-1.7	79.0	0.5	3
1	2	2018-11-23	JP	250.0	1.0	-1.6	79.0	0.6	3
2	3	2018-11-23	JP	161.0	2.0	-1.1	79.0	0.7	3
3	4	2018-11-23	JP	99.0	3.0	-0.6	76.0	0.5	3
4	5	2018-11-23	JP	64.0	4.0	-0.3	76.0	0.9	3

5 rows × 21 columns

Transformation 3

Create a new column called day to use it as an independent variable for further analysis

In [20]:

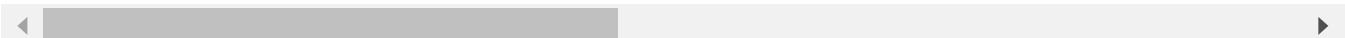
```
import pandas as pd
from datetime import datetime
```

```
assignment_dataframe['Date'] = pd.to_datetime(assignment_dataframe['Date'])
assignment_dataframe['Day'] = assignment_dataframe['Date'].apply(lambda x: x.strftime('%A'))
assignment_dataframe.head()
```

Out[20]:

	S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility
0	1	2018-11-23	JP	196.0	0.0	-1.7	79.0	0.5	3
1	2	2018-11-23	JP	250.0	1.0	-1.6	79.0	0.6	3
2	3	2018-11-23	JP	161.0	2.0	-1.1	79.0	0.7	3
3	4	2018-11-23	JP	99.0	3.0	-0.6	76.0	0.5	3
4	5	2018-11-23	JP	64.0	4.0	-0.3	76.0	0.9	3

5 rows × 22 columns



Transformation 4

Adjusting visibility column from numerical to nominal to ensure meaningful analysis during graph plotting

In [21]:

```
visibility_mapping = {5: 'Barely Visible', 4: 'Low Visible', 3: 'Visible', 2: 'Highly Visible', 1: 'Very High Visibility'}
assignment_dataframe['Visibility'] = assignment_dataframe['Visibility'].replace(visibility_mapping)
```

Pearson's Correlation Analysis

Just calculating the correlation between each of the individual between one factor and other

- **JUSTIFICATION:** Calculating the correlation matrix helps us understand the relationships between different variables in the dataset. This information can be useful for identifying patterns between variables. It provides insights into how changes in one variable may affect another and helps us identify the relationships between such variables
- **DESCRIPTION:** The result of running the code will be the correlation matrix, which shows the pairwise correlations between the selected columns. The matrix will be a square table with the selected columns as both row and column headers. Each cell in the matrix represents the correlation coefficient between the corresponding pair of variables.

In [22]:

```
import pandas as pd

# Select the columns of interest
columns = ['Rented_Bike_Count', 'Humidity', 'Windspeed', 'Dewpoint_Temp', 'Solar_Radiation']

# Create a subset DataFrame with the selected columns
subset_df = assignment_dataframe[columns]

# Calculate the Pearson correlation matrix
correlation_matrix = subset_df.corr(method='pearson')
```

```
# Print the correlation matrix
print(correlation_matrix)

      Rented_Bike_Count  Humidity  Windspeed  Dewpoint_Temp \
Rented_Bike_Count  1.000000 -0.199917  0.120405  0.379970
Humidity          -0.199917  1.000000 -0.336091  0.536379
Windspeed          0.120405 -0.336091  1.000000 -0.176615
Dewpoint_Temp      0.379970  0.536379 -0.176615  1.000000
Solar_Radiation   0.261422 -0.461554  0.332221  0.094315
Rainfall           -0.122729  0.236307 -0.019695  0.125619
Temperature        0.536543  0.157959 -0.040251  0.907939

      Solar_Radiation  Rainfall  Temperature
Rented_Bike_Count  0.261422 -0.122729  0.536543
Humidity          -0.461554  0.236307  0.157959
Windspeed          0.332221 -0.019695 -0.040251
Dewpoint_Temp      0.094315  0.125619  0.907939
Solar_Radiation   1.000000 -0.074198  0.350785
Rainfall           -0.074198  1.000000  0.050352
Temperature        0.350785  0.050352  1.000000
```

Other Method(s)

The provided code calculates the correlation matrix for the selected quantitative columns 'Temperature', 'Humidity', 'Windspeed', and 'Dewpoint_Temp' in the assignment_dataframe DataFrame. The correlation matrix is then visualized using a heatmap.

- JUSTIFICATION:** Visualizing the correlation matrix as a heatmap provides an intuitive and informative representation of the correlations.
- RESULT:** The result of running the code is a heatmap that displays the correlation matrix. The heatmap uses a color scale to represent the correlation coefficients, where blue indicates negative correlation, red indicates positive correlation, and lighter colors indicate stronger correlations. The annotations within each cell of the heatmap show the actual correlation coefficients, providing a quantitative measure of the relationships between the variables.

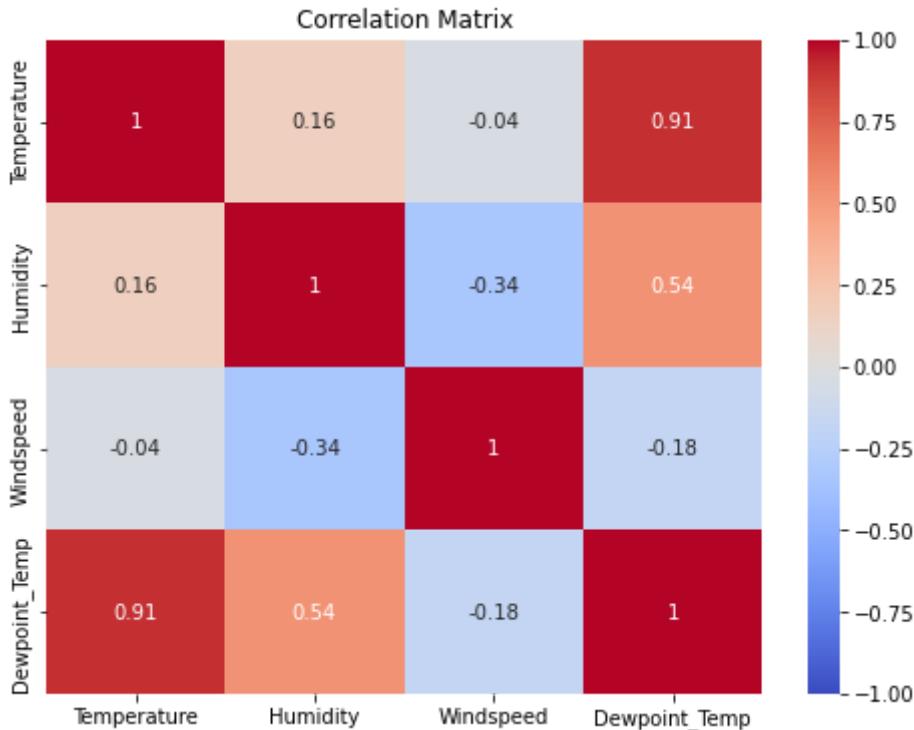
```
In [23]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Select the four quantitative columns of interest
columns = ['Temperature', 'Humidity', 'Windspeed', 'Dewpoint_Temp']

# Create a subset dataframe with the selected columns
subset_df = assignment_dataframe[columns]

# Calculate the correlation matrix
correlation_matrix = subset_df.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



Basically the same as the previous but with the remaining variable

In [24]:

```

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

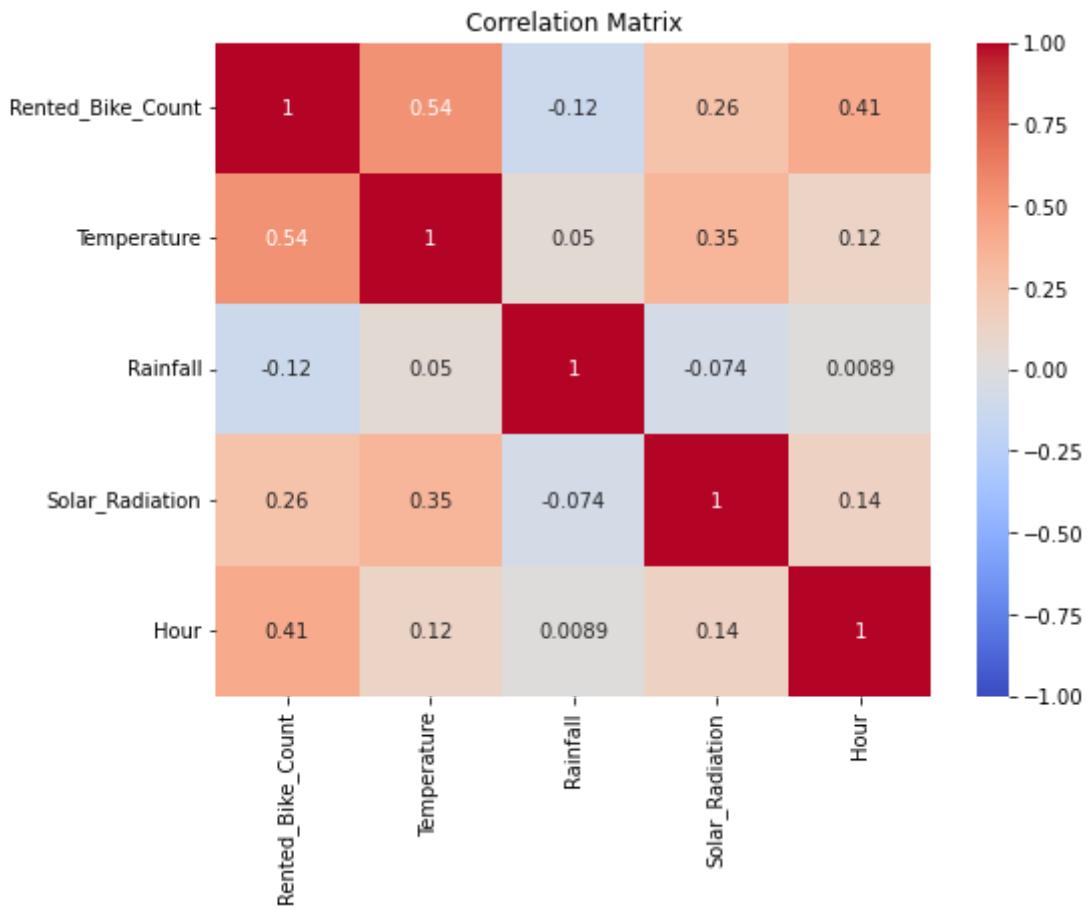
# Select the four quantitative columns of interest
columns = ['Rented_Bike_Count', 'Temperature', 'Rainfall', 'Solar_Radiation', 'Hour']

# Create a subset dataframe with the selected columns
subset_df = assignment_dataframe[columns]

# Calculate the correlation matrix
correlation_matrix = subset_df.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()

```



Description: This code performs a chi-square test of independence between the variables 'Visibility' and 'Snowfall'. It creates a contingency table to represent the observed frequencies of the two variables and then calculates the chi-square statistic and p-value using the `chi2_contingency` function from the `scipy.stats` module.

- **Justification:** The chi-square test of independence is commonly used to determine if there is a significant association between two categorical variables. In this case, we want to examine the relationship between 'Visibility' and 'Snowfall' to determine if they are dependent or independent variables.
- The code outputs the chi-square statistic and p-value. The chi-square statistic measures the discrepancy between the observed and expected frequencies in the contingency table. The p-value indicates the probability of observing the observed data or more extreme results under the assumption that the null hypothesis is true. But in this case since P-value is 0, there is no relation



is it possible to use pearson relation on nominal data?



The Pearson correlation coefficient is primarily used to measure the linear relationship between two continuous variables. It assumes that the data follows a bivariate normal distribution and measures the degree to which the relationship between the variables can be described by a straight line.



Nominal data, on the other hand, consists of categorical variables with no inherent numerical value or ordering. Since the Pearson correlation coefficient is based on the covariance and standard deviations of the variables, it is not applicable to nominal data as it does not capture the nature of categorical relationships.

For nominal data, other statistical measures such as chi-square test, Cramer's V, or contingency coefficient can be used to assess the association between variables. These measures are specifically designed for categorical data and provide insights into the strength and significance of the relationship between nominal variables.

I did end up using the suggestion specifically chi-square test. Cramer's V does not cater too well to nominal data and I therefore rejected it. However the general message and suggestion were valid and insightful

In [25]:

```
import pandas as pd
from scipy.stats import chi2_contingency

# Create a contingency table
contingency_table = pd.crosstab(assignment_dataframe['Visibility'], assignment_dataframe['Rented_Bike_Count'])

# Perform the chi-square test
chi2, p, _, _ = chi2_contingency(contingency_table)

# Print the results
print(f"Chi-square statistic: {chi2:.4f}")
print(f"P-value: {p:.4f}")
```

Chi-square statistic: 187.7308
P-value: 0.0000

Graphing

Using matplotlib bar chart to show relation between Rented_Bike_Count, Visibility and Day

- Purpose:**We are able to identify the average sales performance on the separate days and different level of visibility. That way stakeholders will be able to know if the following 2 factors actually affects the level of sales, and will then be able to adjust inventories and other assets accordingly. From the graph it is fair to say that the degree of visibility does not affect the rented bike count. We can also identify what day is the rented bike count at its highest

```
In [26]: import matplotlib.pyplot as plt
import numpy as np

# Select column we will be using
df = assignment_dataframe[['Rented_Bike_Count', 'Visibility', 'Day']]

# Define the color mapping for each day
color_mapping = {'Monday': 'green', 'Tuesday': 'blue', 'Wednesday': 'red', 'Thursday': 'orange', 'Friday': 'purple', 'Saturday': 'cyan', 'Sunday': 'yellow'}

# Group the data by 'Visibility' and calculate the mean 'Rented_Bike_Count' for each day
grouped_df = df.groupby(['Visibility', 'Day'])['Rented_Bike_Count'].mean().unstack()

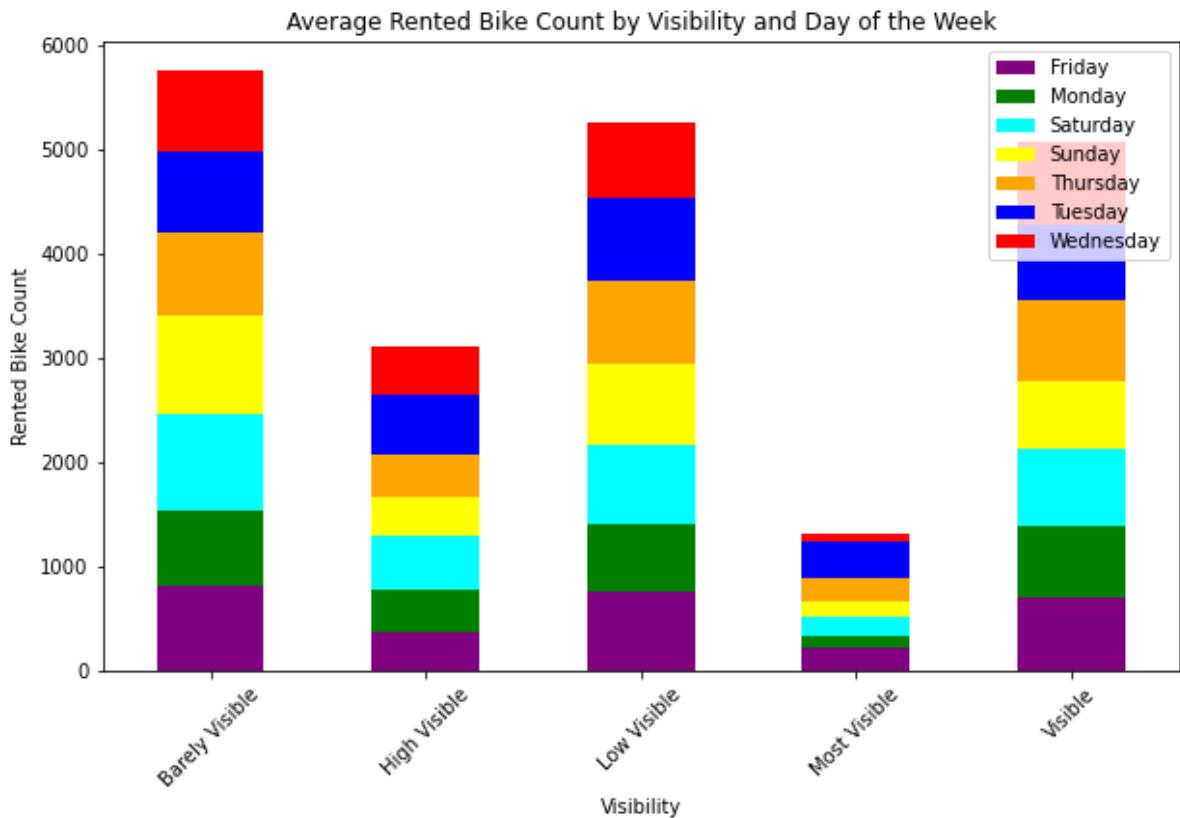
# Plotting the bar chart
fig, ax = plt.subplots(figsize=(10, 6))
grouped_df.plot(kind='bar', stacked=True, color=[color_mapping.get(day, 'gray') for day in grouped_df.columns], legend=True)

# Set the x-axis label and title
ax.set_xlabel('Visibility')
ax.set_ylabel('Rented Bike Count')
ax.set_title('Average Rented Bike Count by Visibility and Day of the Week')

# Set the legend
handles, labels = ax.get_legend_handles_labels()
labels = [label.capitalize() for label in labels]
ax.legend(handles, labels, loc='upper right')

# Rotate the x-axis tick labels for better visibility
plt.xticks(rotation=45)

# Display the plot
plt.show()
```



Using matplotlib lineplot to show relation between Average Rented Bike Count and hour as well as windchill

- **Purpose:** Aside from allowing us to observe how the average rental bike count varies throughout the day, we can also learn if there are specific hours where the bike count is higher or lower. The secondary y-axis representing the average windchill allows us to assess whether there is any correlation between windchill and bike rental. We can see if there are any patterns or trends in windchill that leads to changes in bike rental. On the stakeholders part, we can help provide insights for operational planning: The graph can offer valuable insights for bike rental operations. From this graph we can see that the there is a rather strong relationship between rented bike count and wind chill. We can also identify the hour where the rented bike count is lowest and highest.

In [27]:

```
import matplotlib.pyplot as plt

# Calculate the average of Rental_Bike_Count for each hour
average_rental_count = assignment_dataframe.groupby('Hour')[['Rented_Bike_Count']].mean()

# Calculate the average of Windchill for each hour
average_windchill = assignment_dataframe.groupby('Hour')[['Windchill']].mean()

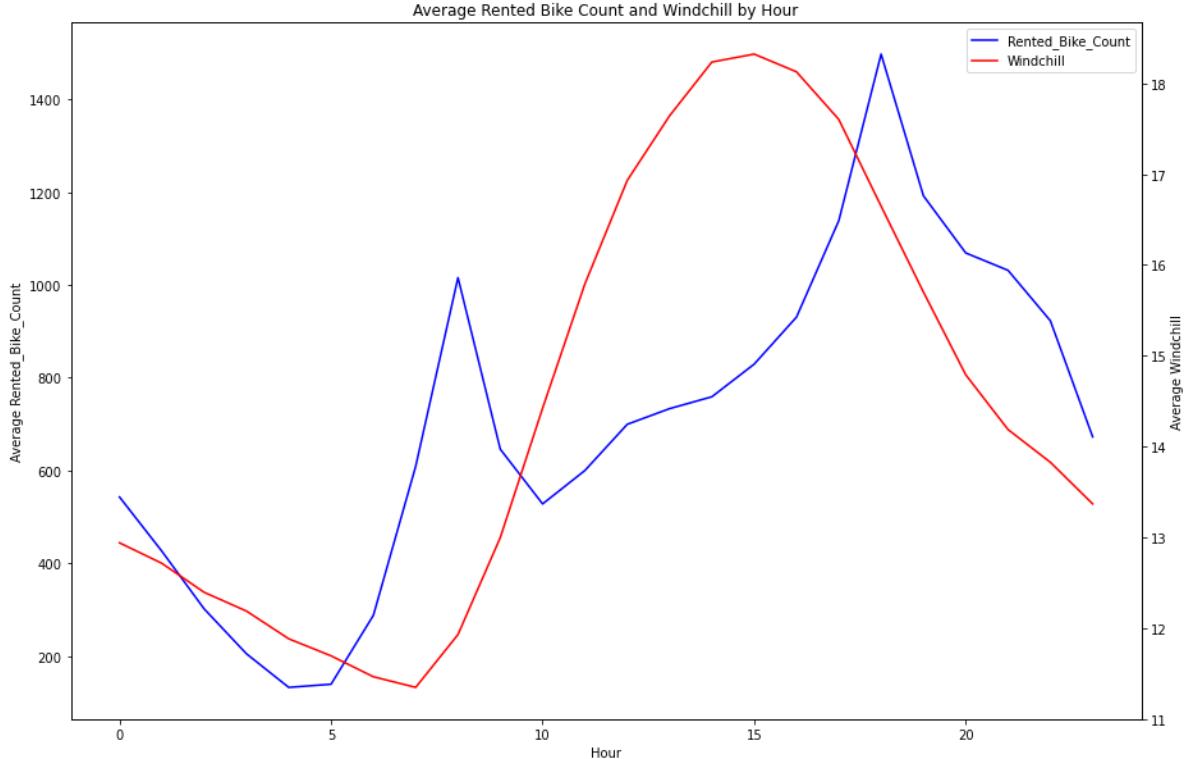
# Create the line chart with increased size
fig, ax1 = plt.subplots(figsize=(15, 10))

# Plot the line for Rental_Bike_Count
ax1.plot(average_rental_count.index, average_rental_count.values, color='blue', label='Average Rented Bike Count')
ax1.set_xlabel('Hour')
ax1.set_ylabel('Average Rented Bike Count')
ax1.set_title('Average Rented Bike Count and Windchill by Hour')

# Create the secondary y-axis for Windchill
ax2 = ax1.twinx()
ax2.plot(average_windchill.index, average_windchill.values, color='red', label='Average Windchill')
ax2.set_ylabel('Average Windchill')

# Combine the legends
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines + lines2, labels + labels2, loc='upper right')

# Display the chart
plt.show()
```



Using matplotlib line chart to show relation between hour , heat index and relation with rented bike count

- Purpose: Visualize the relationship between average heat index and average rented bike count as well as identify potential associations between heat index and bike rental by hour: By using different colors to represent different hours of the day, we can analyze if there are specific hours when the relationship between heat index and bike rental is stronger or weaker. Of course the big reason we also use scatter plot is to identify potential anomalies or outlier. This will allow for potential study or further cleaning for machine learning. For the stakeholders, the graph also enable business to optimize their operations, allocate resources, or implement targeted strategies during specific weather conditions. So we can learn from the general distribution of this graph that generally as the heat index increase, the general bike count rented will increase

```
In [28]: import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the average of Heat_Index for each hour
average_heat_index = assignment_dataframe.groupby('Hour')[['Heat_Index']].mean()

# Calculate the average of Rented_Bike_Count for each hour
average_rented_bike_count = assignment_dataframe.groupby('Hour')[['Rented_Bike_Count']].mean()

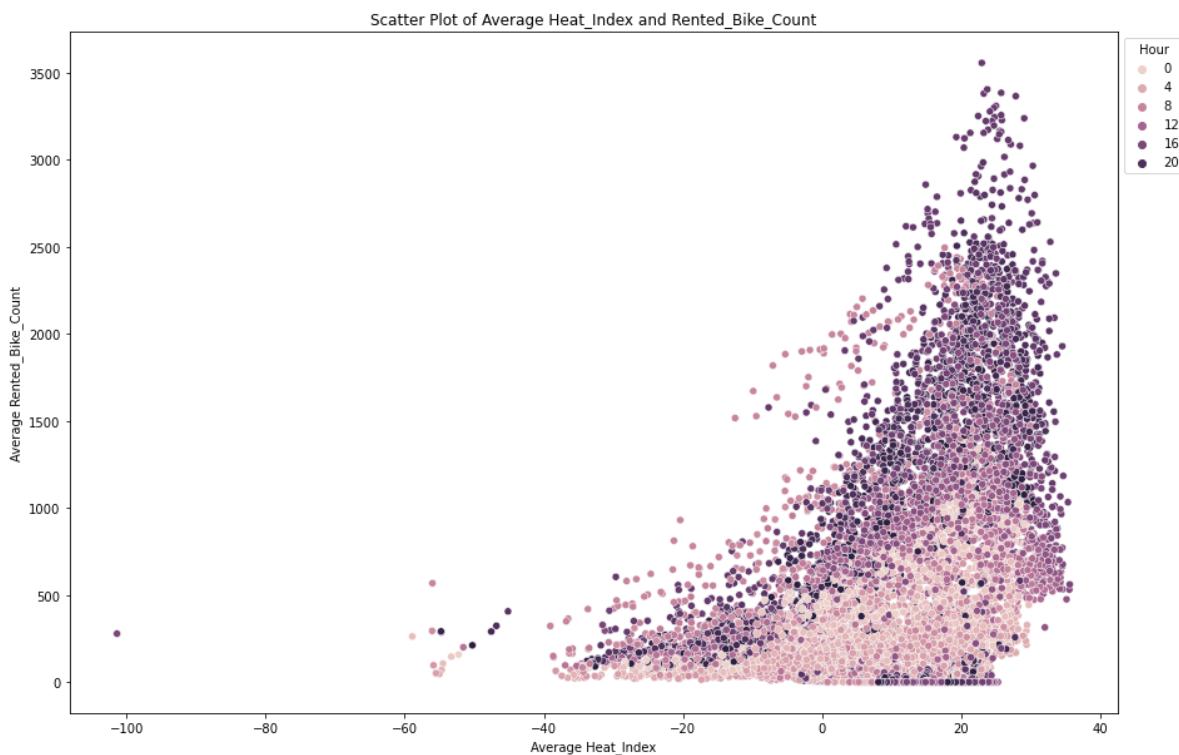
# Set the figure size
plt.figure(figsize=(15, 10))

# Create the scatter plot
sns.scatterplot(data=assignment_dataframe, x='Heat_Index', y='Rented_Bike_Count', color='blue')

# Add Labels and title
plt.xlabel('Average Heat_Index')
plt.ylabel('Average Rented_Bike_Count')
plt.title("Scatter Plot of Average Heat_Index and Rented_Bike_Count")
```

```
# Show the legend
plt.legend(title='Hour', bbox_to_anchor=(1, 1), loc='upper left')

# Display the plot
plt.show()
```



This graph is also using matplotlib to draw a line chart comparing factor of snowfall, rainfall and the Rented bike cout

- **Purpose:** we can visualize the relationship between average rainfall and rented bike count, identify the impact of different levels of snowfall on the relationship and as per usual on line chart, detect patterns or trends in the data. And as per usual on the stakeholders part, we hope that this information can be valuable for operational planning, resource allocation, or decision-making during different weather conditions. From this chart, we can see that the snowfall is generally low regardless of the rainfall and the number of rented bike count.

In [29]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the average of Rainfall for each category in Snowfall and Rented_Bike_Cout
average_rainfall = assignment_dataframe.groupby(['Snowfall', 'Rented_Bike_Count'])

# Set the figure size
plt.figure(figsize=(15, 10))

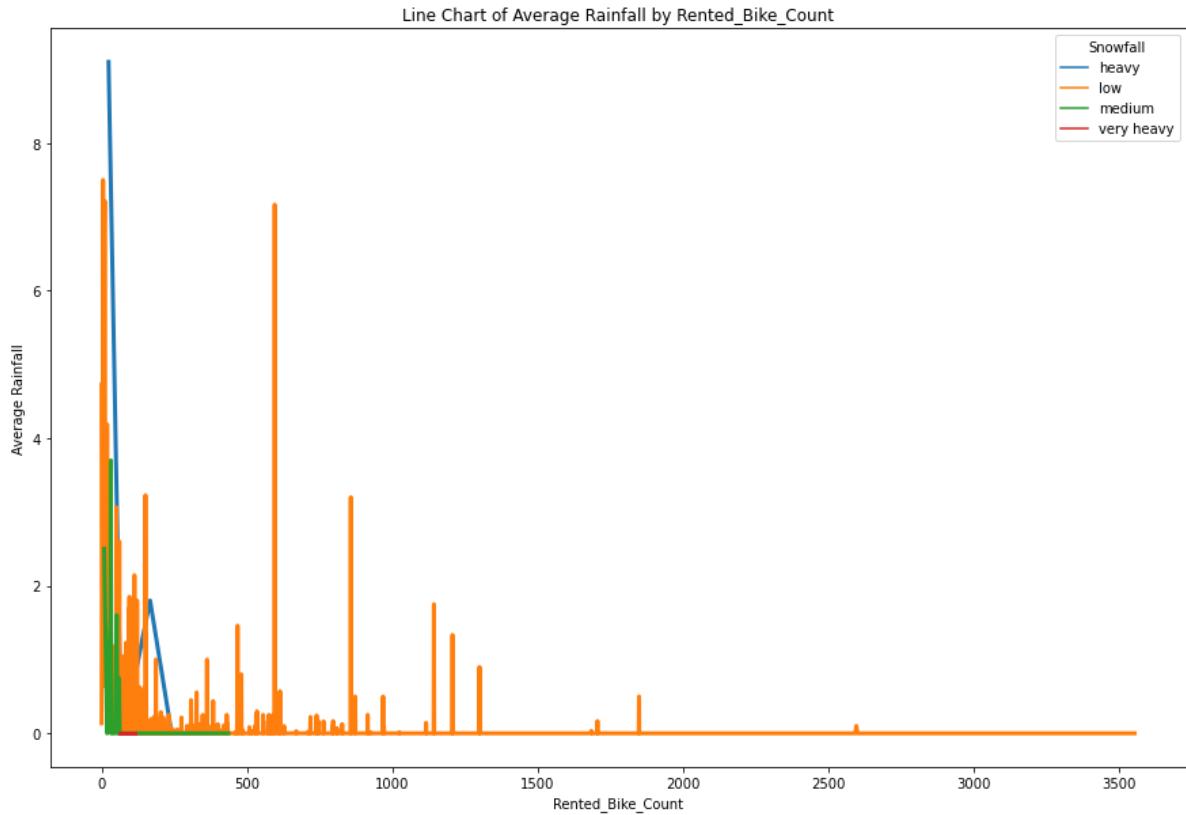
# Create the line chart with increased Linewidth for snowfall factor
sns.lineplot(data=average_rainfall, x='Rented_Bike_Count', y='Rainfall', hue='Snowfall')

# Add Labels and title
plt.xlabel('Rented_Bike_Count')
plt.ylabel('Average Rainfall')
plt.title('Line Chart of Average Rainfall by Rented_Bike_Count')

# Show the Legend
```

```
plt.legend(title='Snowfall')

# Display the plot
plt.show()
```



Using matplotlib to plot another line chart which pin rented bike count, dewpoint temperature and solar radiation against one another

- we can analyze the relationship between solar radiation and average rented bike count, explore the impact of solar radiation on bike rental and dewpoint temperature and as per usual identify trends and pattern between the variables and see if the variables associate themselves with one another. Just like the previous few graph of environmental data, we hope that it can help stakeholders understand the relationship between solar radiation, bike rental, and Dewpoint_Temp, and guide decision-making in terms of marketing strategies, pricing adjustments, or operational adaptations based on solar radiation levels. From this graph we can see that the Rented bike count and dewpoint temperature has little relation with one another. Its also fair to say that the solar radiation has no relation with the other 2 factor

```
In [30]: import matplotlib.pyplot as plt

# Calculate the average of Rented_Bike_Count for each solar radiation Level
average_rented_bike_count = assignment_dataframe.groupby('Solar_Radiation')['Rented_Bike_Count'].mean()

# Calculate the average of Dewpoint_Temp for each solar radiation Level
average_dewpoint_temp = assignment_dataframe.groupby('Solar_Radiation')['Dewpoint_Temp'].mean()

# Create the Line graph
fig, ax1 = plt.subplots(figsize=(15, 10))

# Plot the Line for Rented_Bike_Count
ax1.plot(average_rented_bike_count.index, average_rented_bike_count.values, color='blue')
```

```

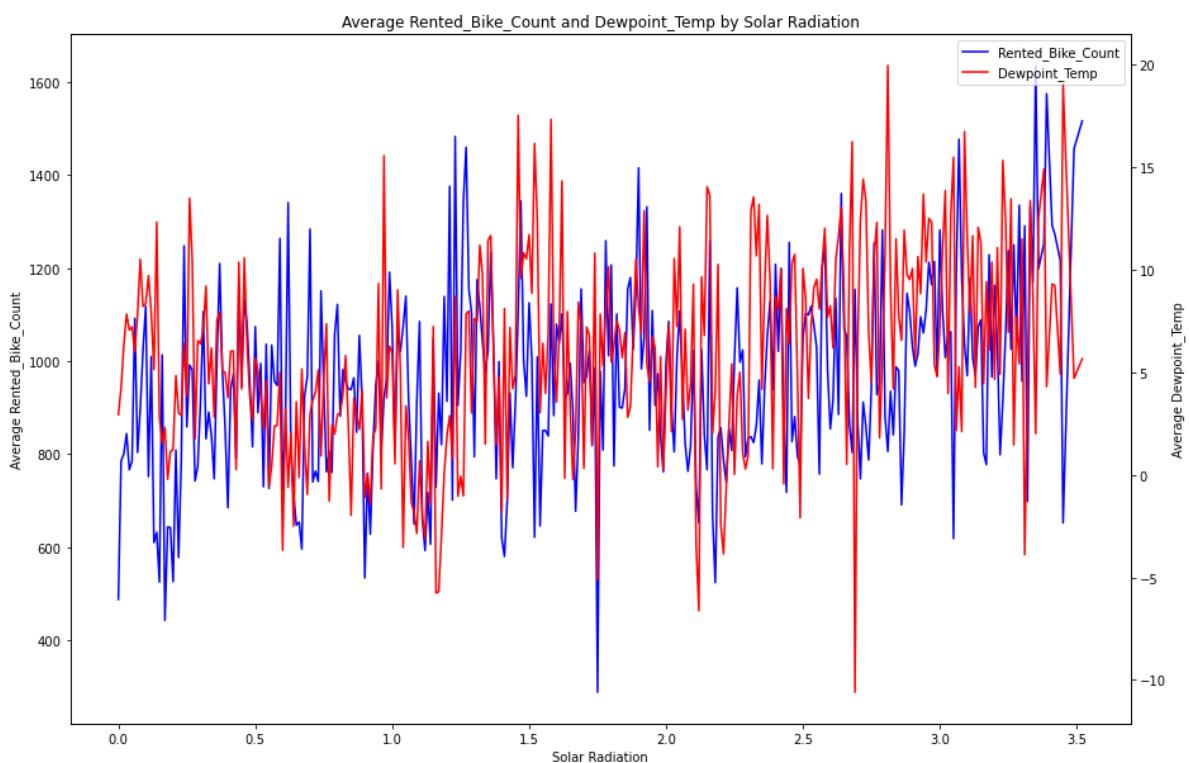
ax1.set_xlabel('Solar Radiation')
ax1.set_ylabel('Average Rented_Bike_Count')
ax1.set_title('Average Rented_Bike_Count and Dewpoint_Temp by Solar Radiation')

# Create the secondary y-axis for Dewpoint_Temp
ax2 = ax1.twinx()
ax2.plot(average_rented_bike_count.index, average_dewpoint_temp.values, color='red')
ax2.set_ylabel('Average Dewpoint_Temp')

# Combine the Legends
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines + lines2, labels + labels2, loc='upper right')

# Display the plot
plt.show()

```



In [31]: `#assignment_dataframe.to_csv('dseslmao.csv', index=False)`

Data Modelling

Linear Regression

Provide the description, justification and results of your technique(s) here.

In [32]:

```

import pandas as pd

# Replace 'Rented_Bike_Count' and 'Hit_Sales' with the actual column names in your
# Dropping rows with missing values in 'Rented_Bike_Count' and 'Hit_Sales' columns
assignment_dataframe.dropna(subset=['Rented_Bike_Count', 'Hit_Sales'], inplace=True)

# Optional: Resetting the index if you want a consecutive index after dropping rows
assignment_dataframe.reset_index(drop=True, inplace=True)

```

```
In [33]: # Convert 'Visibility' values to numerical values
visibility_values = ['Barely Visible', 'Low Visible', 'Visible', 'High Visible', 'Very High Visible']
assignment_dataframe['Visibility'] = assignment_dataframe['Visibility'].apply(lambda x: visibility_values.index(x))

# Convert 'Snowfall' values to numerical values
snowfall_values = ['low', 'medium', 'heavy', 'very heavy']
assignment_dataframe['Snowfall'] = assignment_dataframe['Snowfall'].apply(lambda x: snowfall_values.index(x))

# Convert 'Day' values to numerical values
day_mapping = {'Monday': 1, 'Tuesday': 2, 'Wednesday': 3, 'Thursday': 4, 'Friday': 5}
assignment_dataframe['Day'] = assignment_dataframe['Day'].replace(day_mapping)
```

```
In [34]: assignment_dataframe.head()
```

	S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility
0	1	2018-11-23	JP	196.0	0.0	-1.7	79.0	0.5	3
1	2	2018-11-23	JP	250.0	1.0	-1.6	79.0	0.6	3
2	3	2018-11-23	JP	161.0	2.0	-1.1	79.0	0.7	3
3	4	2018-11-23	JP	99.0	3.0	-0.6	76.0	0.5	3
4	5	2018-11-23	JP	64.0	4.0	-0.3	76.0	0.9	3

5 rows × 22 columns

```
In [35]: import pandas as pd

# Identify numeric columns (excluding datetime and non-numeric columns)
numeric_columns = assignment_dataframe.select_dtypes(include=['number']).columns

# Convert numeric columns to float
assignment_dataframe[numeric_columns] = assignment_dataframe[numeric_columns].astype(float)

# Print the data types after conversion
print(assignment_dataframe.dtypes)
```

```

S/N           float64
Date          datetime64[ns]
Country        object
Rented_Bike_Count float64
Hour           float64
Temperature    float64
Humidity       float64
Windspeed      float64
Visibility     float64
Dewpoint_Temp  float64
Solar_Radiation float64
Rainfall       float64
Snowfall       float64
Open           object
Hit_Sales      object
Region          object
Latitude        float64
Longitude       float64
Relative_humidity float64
Heat_Index      float64
Windchill       float64
Day             float64
dtype: object

```

```

In [36]: import numpy as np
corr_mat=assignment_dataframe.corr() #default is Pearson's
corr_mat.style.background_gradient(cmap='Purples')
#noticed that you don't have to physically drop those categorical attributes if they
#are in the form of string

```

Out[36]:

	S/N	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed
S/N	1.000000	0.342803	0.003025	0.575539	0.166223	-0.157186
Rented_Bike_Count	0.342803	1.000000	0.410300	0.536525	-0.200286	0.120653
Hour	0.003025	0.410300	1.000000	0.124007	-0.241540	0.285089
Temperature	0.575539	0.536525	0.124007	1.000000	0.157263	-0.039742
Humidity	0.166223	-0.200286	-0.241540	0.157263	1.000000	-0.336915
Windspeed	-0.157186	0.120653	0.285089	-0.039742	-0.336915	1.000000
Visibility	-0.123609	-0.200668	-0.090860	-0.036980	0.497641	-0.129523
Dewpoint_Temp	0.560379	0.379885	0.003758	0.907797	0.536314	-0.176344
Solar_Radiation	0.092059	0.261763	0.145352	0.351036	-0.462934	0.332897
Rainfall	0.037427	-0.122433	0.008703	0.049472	0.235318	-0.023354
Snowfall	-0.031677	-0.083292	-0.015070	-0.103469	0.110887	-0.024881
Latitude	-0.003900	0.001710	-0.002502	0.008077	-0.000595	0.001361
Longitude	0.013559	-0.000465	0.002486	0.011237	-0.000578	0.013873
Relative_humidity	0.166223	-0.200286	-0.241540	0.157263	1.000000	-0.336915
Heat_Index	0.593394	0.527066	0.114680	0.992336	0.194149	-0.062958
Windchill	0.581840	0.528125	0.105647	0.995116	0.176051	-0.104289
Day	0.007986	0.036210	0.000390	-0.003573	0.006608	0.022231

```
In [37]: import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Select the predictor variables
predictors = ["Temperature", "Humidity", "Windspeed", "Visibility", "Dewpoint_Temp",
              "Solar_Radiation", "Rainfall", "Snowfall", "Day", "Hour"]

# Calculate VIF for each predictor variable
X = assignment_dataframe[predictors]
vif_data = pd.DataFrame()
vif_data["Variable"] = predictors
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Print the VIF values
print(vif_data)
```

	Variable	VIF
0	Temperature	37.579682
1	Humidity	45.280708
2	Windspeed	4.745254
3	Visibility	8.307998
4	Dewpoint_Temp	25.129761
5	Solar_Radiation	2.837648
6	Rainfall	1.082462
7	Snowfall	60.275444
8	Day	4.979874
9	Hour	4.393914

```
In [38]: # Correlation with target
cor_target = abs(corr_mat['Rented_Bike_Count']) #interested in the strength, direction
# Selecting highly correlated attributes
relevant_features = cor_target[cor_target>0.6]
relevant_features
```

```
Out[38]: Rented_Bike_Count    1.0
Name: Rented_Bike_Count, dtype: float64
```

```
In [39]: from sklearn.model_selection import train_test_split #need to partition original (X, Y)
from sklearn.preprocessing import StandardScaler #optional for linear/logistic regression

scaler = StandardScaler() #define a placeholder (haven't insert the inputs yet)
Y = assignment_dataframe["Rented_Bike_Count"] #set this attribute as TARGET
X = assignment_dataframe[["Temperature", "Heat_Index", "Windchill"]] #set your features
X_std = scaler.fit_transform(X) #compare the magnitude of the beta coeff (weights)
X_train, X_val, Y_train, Y_val = train_test_split(X_std, Y, train_size=0.8, \
                                                 shuffle=True, random_state=220256)
#in your Assignment, random_state=Full admin no.
#set random_state for repeatability
```

```
In [40]: X_train.shape
```

```
Out[40]: (6980, 3)
```

```
In [41]: X_val.shape
```

```
Out[41]: (1746, 3)
```

```
In [42]: Y_train.shape
```

```
Out[42]: (6980, )
```

```
In [43]: from sklearn import linear_model
#define a placeholder
my_regr = linear_model.LinearRegression()
#fit the x and y data into the model
#this will give the best parameters to minimize the error term (find the 'best-fit'
my_regr.fit(X_train, Y_train)
```

Out[43]: LinearRegression()

```
In [44]: print("Intercept: \n", my_regr.intercept_)
print("Beta Coefficients: \n", my_regr.coef_)
```

```
Intercept:
705.954900693311
Beta Coefficients:
[ 919.50763172 -180.12133944 -393.88283426]
```

```
In [45]: coeff_df = pd.DataFrame(my_regr.coef_, X.columns, columns = ["Coefficients"])
coeff_df
```

	Coefficients
Temperature	919.507632
Heat_Index	-180.121339
Windchill	-393.882834

Predictive model

Rented_Bike_Count= 917.678347 Temperature +(-179.681163)Heat_Index + (-393.471156) * Windchill + 705.3941260744986

```
In [46]: from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

Y_pred=my_regr.predict(X_val) #use this predicted check again y_val
print("Mean Squared Error for val set: %0.2f" % mean_squared_error(Y_val, Y_pred))
print("Coefficient of Determination for val set: %0.2f" % r2_score(Y_val, Y_pred))

Mean Squared Error for val set: 284199.34
Coefficient of Determination for val set: 0.28
```

```
In [47]: # Note that the unit for Mean Squared Error is $ square.
# Let's convert into unit of $ to assess if the model built is 'good enough'
import math
print("Root Mean Squared Error for val set: %0.2f" % math.sqrt(mean_squared_error('

Root Mean Squared Error for val set: 533.10
```

```
In [48]: Y_pred2=my_regr.predict(X_train) #use this predicted check again y_val
print("Mean Squared Error for val set: %0.2f" % mean_squared_error(Y_train, Y_pred))
print("Coefficient of Determination for val set: %0.2f" % r2_score(Y_train, Y_pred))

Mean Squared Error for val set: 295418.73
Coefficient of Determination for val set: 0.29
```

```
In [49]: import statsmodels.api as sm
# use another package to provide the full summary of the model
X_train = sm.add_constant(X_train)
X_val=sm.add_constant(X_val)
my_regr2 = sm.OLS(Y_train, X_train).fit() #OLS means ordinary Least square
print(my_regr2.summary())
```

```
Y_pred2 = my_regr2.predict(X_val) #perform prediction
print(Y_pred2)
```

OLS Regression Results

Dep. Variable:	Rented_Bike_Count	R-squared:	0.294
Model:	OLS	Adj. R-squared:	0.294
Method:	Least Squares	F-statistic:	968.9
Date:	Sun, 13 Aug 2023	Prob (F-statistic):	0.00
Time:	12:13:43	Log-Likelihood:	-53865.
No. Observations:	6980	AIC:	1.077e+05
Df Residuals:	6976	BIC:	1.078e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	705.9549	6.508	108.479	0.000	693.198	718.712
x1	919.5076	79.665	11.542	0.000	763.340	1075.675
x2	-180.1213	53.692	-3.355	0.001	-285.374	-74.868
x3	-393.8828	69.149	-5.696	0.000	-529.436	-258.330

Omnibus:	732.825	Durbin-Watson:	1.994
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1061.584
Skew:	0.806	Prob(JB):	3.02e-231
Kurtosis:	4.026	Cond. No.	26.5

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[632.62534798 529.55462195 1161.57896583 ... 1176.22506143 937.28771213
1182.80196978]

```
In [50]: from sklearn.model_selection import train_test_split #need to partition original (X, Y)
from sklearn.preprocessing import StandardScaler #optional for linear/logistic regression

scaler = StandardScaler() #define a placeholder (haven't insert the inputs yet)
Y = assignment_dataframe["Rented_Bike_Count"] #set this attribute as TARGET
X = assignment_dataframe[["Humidity", "Hour", "Dewpoint_Temp"]].values #set your features (X)
X_std = scaler.fit_transform(X) #compare the magnitude of the beta coeff (weights)
X_train, X_val, Y_train, Y_val = train_test_split(X_std, Y, train_size=0.8, \
shuffle=True, random_state=22025)
#in your Assignment, random_state=Full admin no.
#set random_state for repeatability
```

```
In [51]: X_train.shape
```

```
Out[51]: (6980, 3)
```

```
In [52]: X_val.shape
```

```
Out[52]: (1746, 3)
```

```
In [53]: Y_train.shape
```

```
Out[53]: (6980,)
```

```
In [54]: from sklearn import linear_model
#define a placeholder
my_regr = linear_model.LinearRegression()
#fit the x and y data into the model
```

```
#this will give the best parameters to minimize the error term (find the 'best-fit'
my_regr.fit(X_train, Y_train)
```

Out[54]: LinearRegression()

In [55]:

```
print("Intercept: \n", my_regr.intercept_)
print("Beta Coefficients: \n", my_regr.coef_)
```

Intercept:
706.6177052477877
Beta Coefficients:
[-298.86974596 192.67032892 407.86452807]

In [56]:

```
coeff_df = pd.DataFrame(my_regr.coef_, X.columns, columns = ["Coefficients"])
coeff_df
```

Out[56]:

	Coefficients
Humidity	-298.869746
Hour	192.670329
Dewpoint_Temp	407.864528

Predictive model

Rented_Bike_Count= -298.869746Humidity + 192.670329Hour +407.864528 *
Dewpoint_Temp + 706.6177052477877

In [57]:

```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

Y_pred=my_regr.predict(X_val) #use this predicted check again y_val
print("Mean Squared Error for val set: %0.2f" % mean_squared_error(Y_val, Y_pred))
print("Coefficient of Determination for val set: %0.2f" % r2_score(Y_val, Y_pred))
```

Mean Squared Error for val set: 218318.20
Coefficient of Determination for val set: 0.45

In [58]:

```
# Note that the unit for Mean Squared Error is $ square.
# Let's convert into unit of $ to assess if the model built is 'good enough'
import math
print("Root Mean Squared Error for val set: %0.2f" % math.sqrt(mean_squared_error(
```

Root Mean Squared Error for val set: 467.25

In [59]:

```
Y_pred2=my_regr.predict(X_train) #use this predicted check again y_val
print("Mean Squared Error for val set: %0.2f" % mean_squared_error(Y_train, Y_pred))
print("Coefficient of Determination for val set: %0.2f" % r2_score(Y_train, Y_pred))
```

Mean Squared Error for val set: 228521.87
Coefficient of Determination for val set: 0.45

In [60]:

```
import statsmodels.api as sm
# use another package to provide the full summary of the model
X_train = sm.add_constant(X_train)
X_val=sm.add_constant(X_val)
my_regr2 = sm.OLS(Y_train, X_train).fit() #OLS means ordinary Least square
print(my_regr2.summary())
Y_pred2 = my_regr2.predict(X_val) #perform prediction
print(Y_pred2)
```

OLS Regression Results

Dep. Variable:	Rented_Bike_Count	R-squared:	0.454
Model:	OLS	Adj. R-squared:	0.454
Method:	Least Squares	F-statistic:	1933.
Date:	Sun, 13 Aug 2023	Prob (F-statistic):	0.00
Time:	12:13:43	Log-Likelihood:	-52969.
No. Observations:	6980	AIC:	1.059e+05
Df Residuals:	6976	BIC:	1.060e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	706.6177	5.724	123.458	0.000	695.398	717.838
x1	-298.8697	7.091	-42.146	0.000	-312.771	-284.969
x2	192.6703	5.980	32.217	0.000	180.947	204.394
x3	407.8645	6.870	59.373	0.000	394.398	421.331

Omnibus:	706.994	Durbin-Watson:	2.001
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1278.139
Skew:	0.692	Prob(JB):	2.85e-278
Kurtosis:	4.574	Cond. No.	1.96

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[879.97891423 620.9834076 859.35810085 ... 726.67098222 339.35451967
1390.08325061]

```
In [61]: from sklearn.model_selection import train_test_split #need to partition original (X, Y)
from sklearn.preprocessing import StandardScaler #optional for Linear/Logistic regression

scaler = StandardScaler() #define a placeholder (haven't insert the inputs yet)
Y = assignment_dataframe["Rented_Bike_Count"] #set this attribute as TARGET
X = assignment_dataframe[["Humidity", "Hour", "Solar_Radiation"]] #set your features
X_std = scaler.fit_transform(X) #compare the magnitude of the beta coeff (weights)
X_train, X_val, Y_train, Y_val = train_test_split(X_std, Y, train_size=0.8, \
                                                 shuffle=True, random_state=22025)
#in your Assignment, random_state=Full admin no.
#set random_state for repeatability
```

In [62]: X_train.shape

Out[62]: (6980, 3)

In [63]: X_val.shape

Out[63]: (1746, 3)

In [64]: Y_train.shape

Out[64]: (6980,)

```
In [65]: from sklearn import linear_model
#define a placeholder
my_regr = linear_model.LinearRegression()
#fit the x and y data into the model
#this will give the best parameters to minimize the error term (find the 'best-fit')
my_regr.fit(X_train, Y_train)
```

Out[65]: LinearRegression()

```
In [66]: print("Intercept: \n", my_regr.intercept_)
print("Beta Coefficients: \n", my_regr.coef_)
```

Intercept:
706.6730433275501
Beta Coefficients:
[-7.79727376 244.57067115 132.45735341]

```
In [67]: coeff_df = pd.DataFrame(my_regr.coef_, X.columns, columns = ["Coefficients"])
coeff_df
```

	Coefficients
Humidity	-7.797274
Hour	244.570671
Solar_Radiation	132.457353

Predictive model

Rented_Bike_Count= -7.797274Humidity + 244.570671Hour + 132.457353*Solar radiation+
706.6730433275501

```
In [68]: from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

Y_pred=my_regr.predict(X_val) #use this predicted check again y_val
print("Mean Squared Error for val set: %0.2f" % mean_squared_error(Y_val, Y_pred))
print("Coefficient of Determination for val set: %0.2f" % r2_score(Y_val, Y_pred))

Mean Squared Error for val set: 314478.43
Coefficient of Determination for val set: 0.21
```

```
In [69]: # Note that the unit for Mean Squared Error is $ square.
# Let's convert into unit of $ to assess if the model built is 'good enough'
import math
print("Root Mean Squared Error for val set: %0.2f" % math.sqrt(mean_squared_error(
```

Root Mean Squared Error for val set: 560.78

```
In [70]: Y_pred2=my_regr.predict(X_train) #use this predicted check again y_val
print("Mean Squared Error for val set: %0.2f" % mean_squared_error(Y_train, Y_pred))
print("Coefficient of Determination for val set: %0.2f" % r2_score(Y_train, Y_pred))

Mean Squared Error for val set: 330385.74
Coefficient of Determination for val set: 0.21
```

```
In [71]: import statsmodels.api as sm
# use another package to provide the full summary of the model
X_train = sm.add_constant(X_train)
X_val=sm.add_constant(X_val)
my_regr2 = sm.OLS(Y_train, X_train).fit() #OLS means ordinary Least square
print(my_regr2.summary())
Y_pred2 = my_regr2.predict(X_val) #perform prediction
print(Y_pred2)
```

OLS Regression Results

Dep. Variable:	Rented_Bike_Count	R-squared:	0.211
Model:	OLS	Adj. R-squared:	0.210
Method:	Least Squares	F-statistic:	620.3
Date:	Sun, 13 Aug 2023	Prob (F-statistic):	0.00
Time:	12:13:44	Log-Likelihood:	-54255.
No. Observations:	6980	AIC:	1.085e+05
Df Residuals:	6976	BIC:	1.085e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	706.6730	6.882	102.682	0.000	693.182	720.164
x1	-7.7973	7.929	-0.983	0.325	-23.341	7.746
x2	244.5707	7.104	34.425	0.000	230.644	258.498
x3	132.4574	7.813	16.954	0.000	117.142	147.773

Omnibus:	866.526	Durbin-Watson:	1.981
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1264.953
Skew:	0.929	Prob(JB):	2.08e-275
Kurtosis:	3.949	Cond. No.	1.73

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[934.23594744 636.69732994 315.28597956 ... 274.18731583 198.50763907
1184.45069176]

```
In [72]: from sklearn.model_selection import train_test_split #need to partition original (X)
from sklearn.preprocessing import StandardScaler #optional for Linear/Logistic regression

scaler = StandardScaler() #define a placeholder (haven't insert the inputs yet)
Y = assignment_dataframe["Rented_Bike_Count"] #set this attribute as TARGET
X = assignment_dataframe[["Temperature", "Humidity", "Windspeed", "Visibility", "De
    "Solar_Radiation", "Rainfall", "Snowfall", "Day", "Hour"]].drop("Rented_Bike_C
X_std = scaler.fit_transform(X) #compare the magnitude of the beta coeff (weights)
X_train, X_val, Y_train, Y_val = train_test_split(X_std, Y, train_size=0.8, \
                                                shuffle=True, random_state=220256
#in your Assignment, random_state=Full admin no.
#set random_state for repeatability
```

```
In [73]: from sklearn import linear_model
#define a placeholder
my_regr = linear_model.LinearRegression()
#fit the x and y data into the model
#this will give the best parameters to minimize the error term (find the 'best-fit')
my_regr.fit(X_train, Y_train)
```

Out[73]: LinearRegression()

```
In [74]: print("Intercept: \n", my_regr.intercept_)
print("Beta Coefficients: \n", my_regr.coef_)
coeff_df = pd.DataFrame(my_regr.coef_, X.columns, columns = ["Coefficients"])
coeff_df
```

Intercept:
706.6158009713646
Beta Coefficients:
[179.8009282 -232.67725467 8.63068507 -14.2164547 226.91728922
-58.72747846 -64.03022107 5.64291616 25.23908662 191.98442976]

Out[74]:

Coefficients	
Temperature	179.800928
Humidity	-232.677255
Windspeed	8.630685
Visibility	-14.216455
Dewpoint_Temp	226.917289
Solar_Radiation	-58.727478
Rainfall	-64.030221
Snowfall	5.642916
Day	25.239087
Hour	191.984430

Model formula:

Rented_Bike_Count = 179.800928 Temperature - 232.677255 Humidity + 8.630685
 Windspeed - 14.216455 Visibility + 226.917289 Dewpoint_Temp - 58.727478 Solar_Radiation
 - 64.030221 Rainfall + 5.642916 Snowfall + 25.239087 Day + 191.984430 Hour +
 706.6158009713646

In [75]:

```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

Y_pred=my_regr.predict(X_val) #use this predicted check again y_val
print("Mean Squared Error for val set: %0.2f" % mean_squared_error(Y_val, Y_pred))
print("Coefficient of Determination for val set: %0.2f" % r2_score(Y_val, Y_pred))

Mean Squared Error for val set: 209855.76
Coefficient of Determination for val set: 0.47
```

In [76]:

```
# Note that the unit for Mean Squared Error is $ square.
# Let's convert into unit of $ to assess if the model built is 'good enough'
import math
print("Root Mean Squared Error for val set: %0.2f" % math.sqrt(mean_squared_error(
```

Root Mean Squared Error for val set: 458.10

In [77]:

```
Y_pred2=my_regr.predict(X_train) #use this predicted check again y_val
print("Mean Squared Error for val set: %0.2f" % mean_squared_error(Y_train, Y_pred))
print("Coefficient of Determination for val set: %0.2f" % r2_score(Y_train, Y_pred))

Mean Squared Error for val set: 221393.29
Coefficient of Determination for val set: 0.47
```

In [78]:

```
import statsmodels.api as sm
# use another package to provide the full summary of the model
X_train = sm.add_constant(X_train)
X_val=sm.add_constant(X_val)
my_regr2 = sm.OLS(Y_train, X_train).fit() #OLS means ordinary Least square
print(my_regr2.summary())
Y_pred2 = my_regr2.predict(X_val) #perform prediction
print(Y_pred2)
```

OLS Regression Results

Dep. Variable:	Rented_Bike_Count	R-squared:	0.471			
Model:	OLS	Adj. R-squared:	0.470			
Method:	Least Squares	F-statistic:	620.5			
Date:	Sun, 13 Aug 2023	Prob (F-statistic):	0.00			
Time:	12:13:44	Log-Likelihood:	-52858.			
No. Observations:	6980	AIC:	1.057e+05			
Df Residuals:	6969	BIC:	1.058e+05			
Df Model:	10					
Covariance Type:	nonrobust					
coef	std err	t	P> t			
[0.025	0.975]					
const	706.6158	5.637	125.361	0.000	695.566	717.665
x1	179.8009	38.125	4.716	0.000	105.065	254.537
x2	-232.6773	19.771	-11.769	0.000	-271.434	-193.921
x3	8.6307	6.381	1.353	0.176	-3.878	21.139
x4	-14.2165	6.931	-2.051	0.040	-27.804	-0.629
x5	226.9173	43.991	5.158	0.000	140.681	313.154
x6	-58.7275	7.965	-7.373	0.000	-74.341	-43.114
x7	-64.0302	5.819	-11.003	0.000	-75.438	-52.623
x8	5.6429	5.637	1.001	0.317	-5.407	16.693
x9	25.2391	5.655	4.463	0.000	14.153	36.325
x10	191.9844	6.122	31.360	0.000	179.984	203.985
Omnibus:	685.253	Durbin-Watson:	1.995			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1270.302			
Skew:	0.667	Prob(JB):	1.44e-276			
Kurtosis:	4.610	Cond. No.	16.6			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[860.06126263 637.16959459 919.54721087 ... 787.30163426 208.24525583
1325.23389372]

```
In [79]: from sklearn.model_selection import train_test_split
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Load your assignment_dataframe and set Y and X as you've defined

# Define the scaler
scaler = StandardScaler()

# Set the target and features
Y = assignment_dataframe["Rented_Bike_Count"]
X = assignment_dataframe[["Temperature", "Humidity", "Windspeed", "Visibility", "D
    "Solar_Radiation", "Rainfall", "Snowfall", "Day", "Hour"]

# Standardize the features
X_std = scaler.fit_transform(X)

# Split the data into train and validation sets
X_train, X_val, Y_train, Y_val = train_test_split(X_std, Y, train_size=0.8, shuffle=True)

# Create and fit an Elastic Net model
elastic_net = ElasticNet(alpha=1.0, l1_ratio=0.5) # Adjust alpha and l1_ratio as needed
elastic_net.fit(X_train, Y_train)

# Make predictions using the Elastic Net model
```

```

Y_pred_train = elastic_net.predict(X_train)
Y_pred_val = elastic_net.predict(X_val)

# Calculate Mean Squared Error for training set
mse_train = mean_squared_error(Y_train, Y_pred_train)
print("Mean Squared Error for Training Set:", mse_train)

# Calculate Mean Squared Error for validation set
mse_val = mean_squared_error(Y_val, Y_pred_val)
print("Mean Squared Error for Validation Set:", mse_val)

# Calculate R-squared for training set
r2_train = r2_score(Y_train, Y_pred_train)
print("R-squared for Training Set:", r2_train)

# Calculate R-squared for validation set
r2_val = r2_score(Y_val, Y_pred_val)
print("R-squared for Validation Set:", r2_val)

```

Mean Squared Error for Training Set: 238041.30895437213
 Mean Squared Error for Validation Set: 224631.97418232166
 R-squared for Training Set: 0.4312256906463905
 R-squared for Validation Set: 0.4348307805638085

In [80]:

```

coefficients = elastic_net.coef_
intercept = elastic_net.intercept_
# Print the coefficients for each feature
feature_names = ["Temperature", "Humidity", "Windspeed", "Visibility", "Dewpoint_Temp", "Solar_Radiation", "Rainfall", "Snowfall", "Day", "Hour"]
for feature, coef in zip(feature_names, coefficients):
    print(f"{feature}: {coef}")

```

Temperature: 159.74905208654747
 Humidity: -83.0707097898659
 Windspeed: 13.970648686646763
 Visibility: -45.81223338193685
 Dewpoint_Temp: 106.60600122683489
 Solar_Radiation: 21.37515991494963
 Rainfall: -48.06058593078725
 Snowfall: -8.277291904713326
 Day: 15.068086020321111
 Hour: 142.70741715001876

Formula

Rented_Bike_Count = 706.72 + 159.75 Temperature - 83.07 Humidity + 13.97 Windspeed - 45.81 Visibility + 106.61 Dewpoint_Temp + 21.38 Solar_Radiation - 48.06 Rainfall - 8.28 Snowfall + 15.07 Day + 142.71 Hour

In [81]:

```

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures, StandardScaler

# Load your assignment_dataframe and set Y and X as you've defined

# Define the scaler
scaler = StandardScaler()

# Set the target and features
Y = assignment_dataframe["Rented_Bike_Count"]
X = assignment_dataframe[["Temperature", "Humidity", "Windspeed", "Visibility", "Dewpoint_Temp", "Solar_Radiation", "Rainfall", "Snowfall", "Day", "Hour"]]

```

```

# Standardize the features
X_std = scaler.fit_transform(X)

# Split the data into train and validation sets
X_train, X_val, Y_train, Y_val = train_test_split(X_std, Y, train_size=0.8, shuffle=True)

# Apply Polynomial Regression with degree 2
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_val_poly = poly.transform(X_val)

# Create and fit a Linear Regression model on the polynomial features
poly_regr = LinearRegression()
poly_regr.fit(X_train_poly, Y_train)

# Make predictions using the polynomial regression model
Y_pred_train = poly_regr.predict(X_train_poly)
Y_pred_val = poly_regr.predict(X_val_poly)

# Calculate Mean Squared Error for training set
mse_train = mean_squared_error(Y_train, Y_pred_train)
print("Mean Squared Error for Training Set:", mse_train)

# Calculate Mean Squared Error for validation set
mse_val = mean_squared_error(Y_val, Y_pred_val)
print("Mean Squared Error for Validation Set:", mse_val)

# Calculate R-squared for training set
r2_train = r2_score(Y_train, Y_pred_train)
print("R-squared for Training Set:", r2_train)

# Calculate R-squared for validation set
r2_val = r2_score(Y_val, Y_pred_val)
print("R-squared for Validation Set:", r2_val)

```

Mean Squared Error for Training Set: 174474.6338272048
 Mean Squared Error for Validation Set: 170955.69793563496
 R-squared for Training Set: 0.5831114784626981
 R-squared for Validation Set: 0.5698791380338766

Formula:

Rented_Bike_Count=706.6177052477877-298.869746×Humidity+407.864528×Temperature×Wi



Model Leaderboard (Descending)

Rank	Model Equation	Mean Square Error (Validation)	Mean Square Error (Training)	
---	----- ----- ----- ----- 1			
	Rented_Bike_Count = 706.6177 - 298.8697 * Humidity + 407.8645 * Temperature *			
	Windspeed 170955.70 174474.63 2 Rented_Bike_Count = 179.8009 * Temperature -			
	232.6773 * Humidity + 8.6307 * Windspeed - 14.2165 * Visibility + 226.9173 *			
	Dewpoint_Temp - 58.7275 * Solar_Radiation - 64.0302 * Rainfall + 5.6429 * Snowfall +			
	25.2391 * Day + 191.9844 * Hour + 706.6158 209855.76 221393.29 3			
	Rented_Bike_Count = -298.8697 * Humidity + 192.6703 * Hour + 407.8645 *			
	Dewpoint_Temp + 706.6177 218318.20 228521.87 4 Rented_Bike_Count = 706.72 +			
	159.75 * Temperature - 83.07 * Humidity + 13.97 * Windspeed - 45.81 * Visibility + 106.61 *			
	Dewpoint_Temp + 21.38 * Solar_Radiation - 48.06 * Rainfall - 8.28 * Snowfall + 15.07 * Day +			
	142.71 * Hour 238041.31 224631.97 5 Rented_Bike_Count = 917.6783 * Temperature +			

$$(-179.6812) * \text{Heat_Index} + (-393.4712) * \text{Windchill} + 705.3941 | 281499.34 | 295418.73 | | 6 | \\ \text{Rented_Bike_Count} = -7.7973 * \text{Humidity} + 244.5707 * \text{Hour} + 132.4574 * \text{Solar_Radiation} + \\ 706.6730 | 314478.43 | 330385.74 |$$

REPORT(LINEAR REGRESSION)

Methods & justification

Metric chosen: Mean square error(MSE)

Explanation: MSE measures how close the predicted values are to the actual values. It calculates the average of the squared differences between predicted and actual values. I have considered a more straightforward option which is R square is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable. Another option I have considered is negative mean square error. That is because the value for MSE is very high. By using the negative MSE, it makes it easy for further analysis and understanding. However I ran into the issue of some of my models have exact same result in terms of training and validation with another model. This occurs because the negative sign of the MSE is just a mathematical operation and doesn't affect the relative ranking of models' performance and I therefore rejected it. I am aware of how sensitive MSE is in terms of sensitivity to error which will cause fluctuation between the overall result but for its general use and reliability, I think it is a worthy tradeoff and judging by my different methods of optimizing my models, I think it is the better option in terms of its Differentiable and suitable.

Models used: [polynomial Linear regression type: 2], {Elastic net model}, (simple linear regression)

Explanation: The choice of a [polynomial degree of 2 allows the model] was used to capture curvature and non-linearity in the relationships between the features and the target variable. In this case, a degree of 2 is used, meaning that the model will include linear and quadratic terms. The quadratic terms created by the polynomial features allow the model to capture more complex patterns in the data. Polynomial features also enable the model to capture interactions between features. The choice of a {elastic net model} was used to overcome some limitations of Lasso (L1 regularization) and Ridge (L2 regularization) regression by combining their penalties. It can be particularly useful when dealing with high-dimensional data with potentially collinear features. This is the big reason why I chose elastic net rather than lasso and ridge since after trying both lasso and ridge separately, I realize that base of my MSE, it barely changed anything from a simple linear regression model. (simple linear regression) is chosen as the results of simple linear regression are easy to interpret. It is also chosen as it is the simplest to implement and the most efficient in terms of computing burden. However, after comparing the different variables, I realized that just manipulating simple linear regression would not be sufficient to create a somewhat reliable model and therefore I moved on

interpretation

(best) first model: The given linear regression equation represents a relationship between the number of rented bike counts (Rented_Bike_Count) and the predictor variables Humidity and the interaction between Temperature and Windspeed. As Humidity increases, the

expected bike count decreases. Additionally, the relationship between Temperature and Rented_Bike_Count is influenced by Windspeed, where higher values of the interaction term lead to higher expected bike counts as Temperature and Windspeed increase together. This is the binomial regression model.

Second model: Very much like the first linear regression equation, this simply shows how various predictor variables influence the number of rented bike counts. Each coefficient represents the change in the expected bike count associated with a one-unit change in the corresponding predictor variable, while holding all other variables constant. This model was developed as I tried to place all my variables into the model to see test the hypothesis: The higher the number of variables, the lower the MSE. The only way to test if this is true is just by using a simple linear regression model

Third model: this equation provides insights into how the predictor variables Humidity, Hour, and Dewpoint_Temp impact the number of rented bike counts. Each coefficient represents the change in the expected bike count associated with a one-unit change in the corresponding predictor variable, while holding all other variables constant.

Fourth model: The coefficient for Temperature is 159.75. This suggests that for every one-unit increase in Temperature, the expected Rented_Bike_Count increases by approximately 160 when all other variable are constant, The coefficient for Humidity is -83.07. This implies that for every one-unit increase in Humidity, the expected Rented_Bike_Count decreases by about 83 and so on. So the gist is that for every one unit increase in the variable, the coefficient is the increase or decrease on rented bike count. fifth model: This model was made using simple linear regression model. This one was the first one that I designed. This variables chosen was not taking multicollinearity at all and therefore it is a little underperforming. However the logic of the formula is the same like the other simple linear regression model. It is simply what effects the 3 variables has on target column Rented bike count. (worst) sixth model: This model very similar to the fifth one was created with only one aim, to avoid issue post by multicollinearity and place in a simple linear model. Through the performance of this model, we can see that it multicollinearity is one of the least impactful but still important aspect in terms of building a reliable linear regression model.

Conclusion: The base 2 aspect to consider when creating a model is its linear correlation and its multicollinearity. However we should also finely tune our variables selection as well as our model types. It is also important to select our metric wisely.

Provide the description, justification and results of your technique(s) here.

Logistic regression

```
In [82]: # Insert your codes here to generate the result.
```

```
In [83]: import pandas as pd

columns_to_copy = ['Hit_Sales', 'Day', 'Visibility', 'Snowfall']

# Create a new DataFrame 'Logistic_dataframe' by copying the specified columns
logistic_dataframe = assignment_dataframe[columns_to_copy].copy()
```

```
# Print the first few rows of the new DataFrame
print(logistic_dataframe.head())
```

	Hit_Sales	Day	Visibility	Snowfall
0	N	5.0	3.0	1.0
1	N	5.0	3.0	1.0
2	N	5.0	3.0	1.0
3	N	5.0	3.0	1.0
4	N	5.0	3.0	1.0

```
In [84]: # Convert numerical 'Visibility' values back to categorical values
# Perform one-hot encoding on the 'Visibility' column
visibility_encoded = pd.get_dummies(logistic_dataframe['Visibility'], prefix='Visi')

# Concatenate the one-hot encoded columns with the original dataframe
logistic_dataframe = pd.concat([logistic_dataframe, visibility_encoded], axis=1)

# Drop the original 'Visibility' column
logistic_dataframe.drop(columns=['Visibility'], inplace=True)

# Now you can continue with the rest of your code, using the updated Logistic_dataframe
# Make sure to adjust the input_columns list to include the new one-hot encoded columns

one_hot_snowfall = pd.get_dummies(logistic_dataframe['Snowfall'], prefix='Snowfall')
logistic_dataframe = pd.concat([logistic_dataframe, one_hot_snowfall], axis=1)
logistic_dataframe.drop(columns=['Snowfall'], inplace=True)

# Convert 'Day' values to one-hot encoded columns
encoded_days = pd.get_dummies(logistic_dataframe['Day'], prefix='Day')
logistic_dataframe = pd.concat([logistic_dataframe, encoded_days], axis=1)
logistic_dataframe = logistic_dataframe.drop(columns=['Day'])

# Convert 'Hit_Sales' values from 0 to 'N' and 1 to 'Y'
hit_sales_mapping = {0: 'N', 1: 'Y'}
logistic_dataframe['Hit_Sales'] = logistic_dataframe['Hit_Sales'].replace(hit_sales_mapping)

# Print the first few rows of the DataFrame with reverted categorical values
print(logistic_dataframe.head())
```

	Hit_Sales	Visibility_2.0	Visibility_3.0	Visibility_4.0	Visibility_5.0	Day_1.0	Day_2.0	Day_3.0	Day_4.0	Day_5.0	Day_6.0	Day_7.0
0	N	0	1	0	0	0	0	0	0	1	0	0
1	N	0	1	0	0	0	0	1	0	0	0	0
2	N	0	1	0	0	0	0	0	1	0	0	0
3	N	0	1	0	0	0	0	0	1	0	0	0
4	N	0	1	0	0	0	0	0	0	1	0	0

```
In [85]: import pandas as pd
import numpy as np
import scipy.stats as stats
```

```
# Replace 'Visibility' and 'Hit_Sales' with the actual column names in your DataFrame
# Contingency table for 'Visibility' and 'Hit_Sales'
contingency_table = pd.crosstab(assignment_dataframe['Day'], assignment_dataframe['Hit_Sales'])

# Computing the chi-squared statistic and p-value
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)

# Calculating Cramer's V
n = contingency_table.sum().sum()
cramer_v = np.sqrt(chi2 / (n * (min(contingency_table.shape) - 1)))

print(f"Cramer's V value: {cramer_v}")
```

Cramer's V value: 0.05021568392294905

```
In [86]: import pandas as pd
import numpy as np
import scipy.stats as stats

# Replace 'Visibility' and 'Hit_Sales' with the actual column names in your DataFrame
# Contingency table for 'Visibility' and 'Hit_Sales'
contingency_table = pd.crosstab(assignment_dataframe['Snowfall'], assignment_dataframe['Hit_Sales'])

# Computing the chi-squared statistic and p-value
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)

# Calculating Cramer's V
n = contingency_table.sum().sum()
cramer_v = np.sqrt(chi2 / (n * (min(contingency_table.shape) - 1)))

print(f"Cramer's V value: {cramer_v}")
```

Cramer's V value: 0.06341280816089144

```
In [87]: import pandas as pd
import numpy as np
import scipy.stats as stats

# Contingency table for 'Visibility' and 'Hit_Sales'
contingency_table = pd.crosstab(assignment_dataframe['Visibility'], assignment_dataframe['Hit_Sales'])

# Computing the chi-squared statistic and p-value
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)

# Calculating Cramer's V
n = contingency_table.sum().sum()
cramer_v = np.sqrt(chi2 / (n * (min(contingency_table.shape) - 1)))

print(f"Cramer's V value: {cramer_v}")
```

Cramer's V value: 0.18283161053976468

```
In [88]: assignment_dataframe.head()
```

Out[88]:

	S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility
0	1.0	2018-11-23	JP	196.0	0.0	-1.7	79.0	0.5	3.0
1	2.0	2018-11-23	JP	250.0	1.0	-1.6	79.0	0.6	3.0
2	3.0	2018-11-23	JP	161.0	2.0	-1.1	79.0	0.7	3.0
3	4.0	2018-11-23	JP	99.0	3.0	-0.6	76.0	0.5	3.0
4	5.0	2018-11-23	JP	64.0	4.0	-0.3	76.0	0.9	3.0

5 rows × 22 columns

In [89]:

```
import pandas as pd

assignment_dataframe['Hit_Sales'] = assignment_dataframe['Hit_Sales'].replace({'N': 0})

# Print the updated DataFrame
print(assignment_dataframe.head())
```

S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility	Dewpoint_Temp	... Snowfall	Open	Hit_Sales	
0	1.0	2018-11-23	JP	196.0	0.0	-1.7	79.0	0.5	3.0	-4.8 ...	1.0	Yes	0
1	2.0	2018-11-23	JP	250.0	1.0	-1.6	79.0	0.6	3.0	-4.7 ...	1.0	Yes	0
2	3.0	2018-11-23	JP	161.0	2.0	-1.1	79.0	0.7	3.0	-4.2 ...	1.0	Yes	0
3	4.0	2018-11-23	JP	99.0	3.0	-0.6	76.0	0.5	3.0	-4.3 ...	1.0	Yes	0
4	5.0	2018-11-23	JP	64.0	4.0	-0.3	76.0	0.9	3.0	-4.0 ...	1.0	Yes	0

Region	Latitude	Longitude	Relative_humidity	Heat_Index	Windchill	Day	
0	South	37.453024	127.718234	80.612245	-9.480569	1.283723	5.0
1	South	37.805615	127.766506	80.612245	-9.327687	1.063320	5.0
2	South	37.276916	127.736311	80.612245	-8.566436	1.285089	5.0
3	South	37.416781	128.262240	77.551020	-7.883928	2.357738	5.0
4	South	37.377350	127.895479	77.551020	-7.431513	1.636654	5.0

[5 rows × 22 columns]

In [90]:

```
import pandas as pd
import numpy as np

# Selecting the desired columns
selected_columns = ['Hit_Sales', 'Day', 'Snowfall', 'Visibility']
selected_dataframe = assignment_dataframe[selected_columns]

# Convert selected columns to a NumPy array and ensure int64 data type
selected_array = selected_dataframe.values.astype(np.int64)

selected_dataframe = pd.DataFrame(selected_array, columns=selected_columns)
```

```
# Print the first 5 rows of the DataFrame
print(selected_dataframe.head())
```

	Hit_Sales	Day	Snowfall	Visibility
0	0	5	1	3
1	0	5	1	3
2	0	5	1	3
3	0	5	1	3
4	0	5	1	3

```
In [91]: y=selected_dataframe['Hit_Sales'] #set categorical target because this is classification
x=selected_dataframe.drop(columns=['Hit_Sales']) #set your input/x/features/predictors
```

```
In [92]: x_train, x_val, y_train, y_val = train_test_split(x, y, train_size=0.8, \
                                                     shuffle=True, random_state=220256)
```

```
In [93]: from sklearn import linear_model
#define a placeholder
my_classifier = linear_model.LogisticRegression(max_iter=10000)
#fit/build the model
my_classifier.fit(x_train, y_train)
```

```
Out[93]: LogisticRegression(max_iter=10000)
```

```
In [94]: print("Intercept: \n", my_classifier.intercept_)
print("Beta Coefficients: \n", my_classifier.coef_[0])
```

```
Intercept:
[1.74155637]
Beta Coefficients:
[ 0.04477683 -2.03408461 -0.3883598 ]
```

```
In [95]: coeff_df2 = pd.DataFrame(my_classifier.coef_[0], x.columns, columns = ["Coefficients"])
coeff_df2
```

```
Out[95]:      Cefficients
              Day      0.044777
              Snowfall -2.034085
              Visibility -0.388360
```

```
In [96]: y_pred_train=my_classifier.predict(x_train) #use this predicted check again y_train
print("Accuracy of trained model: %0.2f" % accuracy_score(y_train, y_pred_train))
```

```
Accuracy of trained model: 0.72
```

```
In [97]: y_pred_val=my_classifier.predict(x_val) #use this predicted check again y_train
print("Accuracy of trained model: %0.2f" % accuracy_score(y_val, y_pred_val))
```

```
Accuracy of trained model: 0.72
```

formula:

$$\text{log-odds}(Y = 1) = 1.74155637 + (0.044777 \text{ Day}) - (2.034085 \text{ Snowfall}) - (0.388360 * \text{Visibility})$$

```
In [98]: y=selected_dataframe['Hit_Sales'] #set categorical target because this is classification
x=selected_dataframe.drop(columns=['Hit_Sales','Day','Visibility']) #set your input variables
```

```
In [99]: x_train, x_val, y_train, y_val = train_test_split(x, y, train_size=0.8, \
                                                     shuffle=True, random_state=22025)
```

```
In [100...]: from sklearn import linear_model
#define a placeholder
my_classifier = linear_model.LogisticRegression(max_iter=1000)
#fit/build the model
my_classifier.fit(x_train, y_train)
```

```
Out[100]: LogisticRegression(max_iter=1000)
```

```
In [101...]: print("Intercept: \n", my_classifier.intercept_)
print("Beta Coefficients: \n", my_classifier.coef_[0])
```

```
Intercept:
[1.40487445]
Beta Coefficients:
[-2.35818388]
```

```
In [102...]: coeff_df2 = pd.DataFrame(my_classifier.coef_[0], x.columns, columns = ["Coefficient"])
coeff_df2
```

```
Out[102]: Coefficients
```

	Coefficients
Snowfall	-2.358184

```
In [103...]: y_pred_train=my_classifier.predict(x_train) #use this predicted check again y_train
print("Accuracy of trained model: %0.2f" % accuracy_score(y_train, y_pred_train))
```

```
Accuracy of trained model: 0.72
```

```
In [104...]: y_pred_val=my_classifier.predict(x_val) #use this predicted check again y_train
print("Accuracy of trained model: %0.2f" % accuracy_score(y_val, y_pred_val))
```

```
Accuracy of trained model: 0.72
```

Formula

$$\text{log-odds} = 1.40487445 - 2.358184 \times \text{Snowfall}$$

```
In [105...]: from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pandas as pd
```

```
# Define the target and input features
target_column = 'Hit_Sales'
input_columns = ['Visibility_2.0', 'Visibility_3.0', 'Visibility_4.0', 'Visibility_5.0',
                 'Snowfall_1.0', 'Snowfall_2.0', 'Snowfall_3.0', 'Snowfall_4.0',
                 'Day_1.0', 'Day_2.0', 'Day_3.0', 'Day_4.0', 'Day_5.0', 'Day_6.0', 'Day_7.0']

# Extract the target and input data
X = logistic_dataframe[input_columns]
y = logistic_dataframe[target_column]

# Split the data into training and validation sets
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y, train_size=0.8, random_state=2202561)

# Create the Logistic regression model
my_classifier = LogisticRegression(max_iter=1000, random_state=2202561)

# Fit the model on the training data
my_classifier.fit(X_train, y_train)

# Print the intercept and beta coefficients
print("Intercept:", my_classifier.intercept_[0])
print("Beta Coefficients:", my_classifier.coef_[0])

# Predict on the training data
y_pred_train = my_classifier.predict(X_train)
accuracy_train = accuracy_score(y_train, y_pred_train)
print("Accuracy on Training Set:", accuracy_train)

# Predict on the validation data
y_pred_val = my_classifier.predict(X_val)
accuracy_val = accuracy_score(y_val, y_pred_val)
print("Accuracy on Validation Set:", accuracy_val)
```

Intercept: -2.1713019726106375
Beta Coefficients: [-0.11839083 -0.3587298 -1.2758249 -2.48988275 1.50640851 -1.13229221 -0.27307351 -0.11343399 -0.22966396 0.03899695 -0.0124507 0.046292 -0.06853137 0.09813422 0.11483167]
Accuracy on Training Set: 0.7243553008595989
Accuracy on Validation Set: 0.7216494845360825

Formula

$\text{log-odds} = -2.1713019726106375 - 0.11839083 \times \text{Visibility 2} - 0.3587298 \times \text{Visibility 3}$
 $- 1.2758249 \times \text{Visibility 4} - 2.48988275 \times \text{Visibility 5} + 1.50640851 \times \text{Snowfall}$
 $1 - 1.13229221 \times \text{Snowfall 2} - 0.27307351 \times \text{Snowfall 3} - 0.11343399 \times \text{Snowfall}$
 $4 + 0.03899695 \times \text{Day 1} + (-0.0124507) \times \text{Day 2} + 0.046292 \times \text{Day 3} + (-0.06853137) \times \text{Day}$
 $4 + 0.09813422 \times \text{Day 5} + 0.11483167 \times \text{Day 6} + \text{Day 7}$

In [106...]

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Define your target and input features
y = selected_dataframe['Hit_Sales']
x = selected_dataframe.drop(columns=['Hit_Sales'])

# Split the data into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(x, y, train_size=0.8, shuffle=True)

# Create a regularized logistic regression model with L2 regularization (Ridge)
my_classifier = LogisticRegression(penalty='l2', max_iter=100, random_state=220346)

# Fit the model on the training data
my_classifier.fit(x_train, y_train)

# Predict on the training data
y_pred_train = my_classifier.predict(x_train)
print("Accuracy of trained model on training data: %.2f" % accuracy_score(y_train))

# Predict on the validation data
```

```
y_pred_val = my_classifier.predict(x_val)
print("Accuracy of trained model on validation data: %0.2f" % accuracy_score(y_val))
```

Accuracy of trained model on training data: 0.72
 Accuracy of trained model on validation data: 0.72

```
In [107...]: print("Intercept: \n", my_classifier.intercept_)
print("Beta Coefficients: \n", my_classifier.coef_[0])
coeff_df2 = pd.DataFrame(my_classifier.coef_[0], x.columns, columns = ["Coefficient"])
coeff_df2
```

Intercept:
 [1.74155637]
 Beta Coefficients:
 [0.04477683 -2.03408461 -0.3883598]

Out[107]:

	Coefficients
Day	0.044777
Snowfall	-2.034085
Visibility	-0.388360

Formula

$$\text{log-odds} = \text{Intercept} + (\text{Day } 0.044777) + (\text{Snowfall } -2.034085) + (\text{Visibility} * -0.388360)$$

In [108...]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Define your target and input features
y = selected_dataframe['Hit_Sales']
x = selected_dataframe.drop(columns=['Hit_Sales'])

# Split the data into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(x, y, train_size=0.8, shuffle=True)

# Create an Elastic Net Logistic regression model
my_classifier = LogisticRegression(penalty='elasticnet', solver='saga', l1_ratio=0.5)

# Fit the model on the training data
my_classifier.fit(x_train, y_train)

# Predict on the training data
y_pred_train = my_classifier.predict(x_train)
print("Accuracy of trained model on training data: %0.2f" % accuracy_score(y_train))

# Predict on the validation data
y_pred_val = my_classifier.predict(x_val)
print("Accuracy of trained model on validation data: %0.2f" % accuracy_score(y_val))
```

Accuracy of trained model on training data: 0.72
 Accuracy of trained model on validation data: 0.72

In [109...]:

```
print("Intercept: \n", my_classifier.intercept_)
print("Beta Coefficients: \n", my_classifier.coef_[0])
coeff_df2 = pd.DataFrame(my_classifier.coef_[0], x.columns, columns = ["Coefficient"])
coeff_df2
```

```

Intercept: [1.95578318]
Beta Coefficients: [ 0.04465228 -2.24875241 -0.38776841]

Out[109]: 

|                   | Coefficients |
|-------------------|--------------|
| <b>Day</b>        | 0.044652     |
| <b>Snowfall</b>   | -2.248752    |
| <b>Visibility</b> | -0.387768    |


```

Formula

$\text{log-odds}(Y = 1) = 1.74155637 + (0.044777 \text{ Day}) - (2.034085 \text{ Snowfall}) - (0.388360 * \text{Visibility})$

Model Leaderboard (Descending)

Rank	Model Equation	Accuracy (Validation)	Accuracy (Training)
1	$\text{log-odds} = 1.74155637 + 0.044777 * \text{Day} - 2.034085 * \text{Snowfall} - 0.388360 * \text{Visibility}$	7.2	7.2
2	$\text{log-odds} = 1.40487445 - 2.358184 * \text{Snowfall}$	7.2	7.2
3	$\text{log-odds} = \text{Intercept} + (\text{Day} * 0.044777) + (\text{Snowfall} * -2.034085) + (\text{Visibility} * -0.388360)$	7.2	7.2
4	$\text{log-odds}(Y = 1) = 1.74155637 + (0.044777 * \text{Day}) - (2.034085 * \text{Snowfall}) - (0.388360 * \text{Visibility})$	7.2	7.2
5	$\text{log-odds} = -2.1713019726106375 - 0.11839083 * \text{Visibility}^2 - 0.3587298 * \text{Visibility}^3 - 1.2758249 * \text{Visibility}^4 - 2.48988275 * \text{Visibility}^5 + 1.50640851 * \text{Snowfall}^1 - 1.13229221 * \text{Snowfall}^2 - 0.27307351 * \text{Snowfall}^3 - 0.11343399 * \text{Snowfall}^4 + 0.03899695 * \text{Day}^1 - 0.0124507 * \text{Day}^2 + 0.046292 * \text{Day}^3 - 0.06853137 * \text{Day}^4 + 0.09813422 * \text{Day}^5 + 0.11483167 * \text{Day}^6 + \text{Day}^7$	7.2	7.2

Report(Logistics Regression)

Method & justification

Metric chosen: Accuracy

Reason: Accuracy is a straightforward and easily understandable metric. It represents the proportion of correctly classified instances out of the total instances in the dataset. This simplicity makes it easy to communicate the model's performance to non-technical stakeholders. Since our target column is hit sales which is a binary outcome target, accuracy also helps as in binary classification problems (where there are two possible classes), accuracy is a suitable metric to use, as it directly captures the correctness of both true positive and true negative predictions. With a combination of its accuracy and simplicity, it makes comparative analysis much easier. I am aware that there are other methods to vet through the logistics regression such as F1 score and ROC curve but due to time and simplicity I chose to stick with this. While accuracy gives a general idea of the model's performance, it doesn't provide insights into the model's ability to discriminate between classes, handle uncertainty, or generalize to new data. Additional metrics like precision,

recall, F1-score, and ROC curves can provide a more comprehensive understanding. I am also aware of how flawed the overall design is as it should be close to impossible that the accuracy score for all training and validation set across multiple types of is the exact same. But it is time to investigate further

Variables selected: The reason I only selected 4 variables which is Visibility, Snowfall, Day and Hit_Sales as it is the only categorical data or at least when encoded, they only have a few unique values. Using numeric data with multiple different types of values can lead to misinterpretation as it may undergo assumption violation, misinterpretation of data which is the coefficients represent the change in log-odds of the outcome for a one-unit change in the predictor variable. If the numeric attribute with mixed values does not have a consistent one-unit change across all types of values, the coefficient's interpretation becomes ambiguous. As always the more complicated the variables get, the more messy the model will be which may be because including such mixed-type numeric attributes can unnecessarily increase the complexity of the model, potentially leading to overfitting and reduced generalization performance.

Model selected: {Elastic Net logistic regression model} [Regularized logistic regression model] (Simple logistic regression)

{elastic net logistic regression model} was chosen for the similar reason I chose to use elastic net linear regression. So the usual usefulness when I want to address multicollinearity, perform feature selection, and control model complexity all in one model. It also helps when I suspect that there are groups of correlated features, some of which might be irrelevant. It strikes a balance between Ridge and Lasso, allowing me to control the balance between feature selection and coefficient size control. [Regularized logistic regression model] was chosen as it is useful when I want to control multicollinearity by penalizing the magnitudes of the coefficients. Ridge helps to stabilize the model by spreading the coefficients more evenly. Lasso helps with feature selection, as it can push the coefficients of irrelevant or weakly relevant features to zero, effectively excluding them from the model. This can lead to a more interpretable and sparse model. Just like elastic net, regularized is also simply utilizing lasso and ridge regularization just applied differently. Therefore their function is more or less the same. (simple logistic regression) is chosen as the base structure of logistic regression. It is simple, straightforward and serve as a guide to the manipulation of the model.

interpretation

Since the accuracy score of the models are the same, I decided to rate them by their ease of interpretation.

first model(best): The first model presents a clear interpretation where the log-odds of the target variable are influenced by the day, snowfall, and visibility. It's a good model with a relatively straightforward interpretation, providing insights into how these factors impact the outcome.

Second model: The second model's simplicity lies in its direct association between snowfall and the log-odds of the outcome. While it lacks complexity, its accuracy could be limited if the relationship between snowfall and the target isn't as dominant. Third model: The third model combines day, snowfall, and visibility, allowing for a more nuanced interpretation. Its use of multiple features enhances its accuracy and interpretability, making it a strong choice for predicting the outcome. Fourth model: The fourth model's formulation, incorporating day, snowfall, and visibility, results in accurate predictions. Its interpretation is consistent

with the first model, offering valuable insights into how these variables impact the target. Fifth model(worse): The fifth model's complexity arises from polynomial terms and various features. While its intricate structure might hinder direct interpretation, it captures non-linear relationships between the predictors and target. Its performance depends on the dataset's characteristics and the necessity for capturing intricate patterns, making it a potentially powerful choice. However due to its complexity, I ranked it the lowest since it seem to not fix my issue with the overall accuracy.

Addressing the big problem

After consulting chatgpt, I identified potential issues which cause my model to always produce the same accuracy score. Some of the more common answer I found were, simplistic model. Taking this into consideration, I started developing the elastic net logistic regression but it seem to failed. The next possible issue was that the data might be to lacking or detailed. therefore I created different types of model which compared different type of variable. Nominal , numerical as well as one hot encoded data. However that still seem to fail. But when I went back to revise my correlation in the form of cramer V, I notice the score were much lower than usual. After some research, I discovered that categorical variables with low Cramer's V values have limited predictive power. When these variables are used as features in a logistic regression model, they might not contribute significantly to explaining the variability in the target variable. This can result in suboptimal model performance and reduced accuracy. Since I wasnt exactly spoiled for choice in terms of catergorical data I can use, it would seem that the only way to explain this phenomenon is using domain knowledge. To begin, our target column which is the hit_sales has a vague definition base of the data dictionary. the data dictionary states: Whether the targeted rental number has been hit in binary value: Y (Yes) or N (No). Judging by how little hit sales there actually is, it is fair to assume that the target for hit sales is relatively high. Therefore, a lot of the variables setting is overlooked. It might even be fair to say that it is just pure luck whether or not the company will achieve its hit sales. To address the variables, tt is true that since it was given in the context that ohaiyo, since it is a popular hobby and a well known activity, it is fair to assume that the day has little influence wether or not there will be hit sales. And to be honest, japan is gorgeous especially during winter season therfore, I doubt whether or not the rate of snowfall would affect it and base of my cramer V statistic this is true. the only thing that can post a threat on whether ohaiyo will meet its hit sales is the degree of visibility. Which base of my CV stats is true as well, since it is the highest. But even so it is only at a low 0.18. In conclusion, we would hardly use columns with different value in logistic regression but our catergorical data is also

**lacking and more importantly the target column is ambiguous by itself
therefore modelling will be very very challenging.**

k-means clustering

Provide the description, justification and results of your technique(s) here.

In [110...]

```
# Insert your codes here to generate the result.
```

In [111...]

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
assignment_data_number=assignment_dataframe.drop(columns=['S/N', 'Date', 'Country'],
```

In [112...]

```
import pandas as pd

# Convert columns to desired data types
column_conversions = {
    'Rented_Bike_Count': 'float64',
    'Hour': 'float64',
    'Temperature': 'float64',
    'Humidity': 'float64',
    'Windspeed': 'float64',
    'Solar_Radiation': 'float64',
}

# Convert columns to the specified data types
assignment_data_number = assignment_data_number.astype(column_conversions)

# Print the data types after conversion
print(assignment_data_number.dtypes)
```

Rented_Bike_Count	float64
Hour	float64
Temperature	float64
Humidity	float64
Windspeed	float64
Solar_Radiation	float64
dtype:	object

In [113...]

```
import pandas as pd

def identify_data_types(dataframe):
    data_types = dataframe.dtypes
    return data_types

# Call the function to get data types of each column in the DataFrame
data_types = identify_data_types(assignment_data_number)

# Display the data types
print(data_types)
```

Rented_Bike_Count	float64
Hour	float64
Temperature	float64
Humidity	float64
Windspeed	float64
Solar_Radiation	float64
dtype:	object

In [114...]

```
#define a placeholder for the scaler
scaler=StandardScaler()
#apply the scaler to the df that only contains num attributes
scaled_assignment_dataframe=scaler.fit_transform(assignment_data_number)
```

In [115...]

```
scaled_assignment_dataframe=pd.DataFrame(scaled_assignment_dataframe, columns=assignment_data_number)
scaled_assignment_dataframe.head()
```

Out[115]:

	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Solar_Radiation
0	-0.789700	-1.661929	-1.203952	1.021001	-1.181144	-0.655639
1	-0.705806	-1.517399	-1.195667	1.021001	-1.084624	-0.655639
2	-0.844076	-1.372869	-1.154241	1.021001	-0.988103	-0.655639
3	-0.940399	-1.228339	-1.112815	0.873491	-1.181144	-0.655639
4	-0.994775	-1.083809	-1.087959	0.873491	-0.795062	-0.655639

In [116...]

```
#define the placeholder for the k-means algo
my_kmeans=KMeans(init='random', n_clusters=3, n_init=10, max_iter=300, random_state=2561)
#refer to the above markdown cell for justification
#need to set run how many sets through n_init
#need to set max_iter to stop infinite loop
#In your Project, random_state=YourLast4DigitsAdmn

#train/fit the model
my_kmeans.fit(scaled_assignment_dataframe)
```

Out[116]:

```
KMeans(init='random', n_clusters=3, random_state=2561)
```

In [117...]

```
print("The 3 cluster centroids are at: " , my_kmeans.cluster_centers_)
print()
print("The number of iterations required to converge: " , my_kmeans.n_iter_)
print()
print("The lowest SSE from the runs: " , my_kmeans.inertia_)
```

The 3 cluster centroids are at: [[-0.49934023 -0.75886684 -0.00254396 0.81157397
-0.58031176 -0.54443048]
[-0.57022121 0.37370089 -1.04833725 -0.70202431 0.47921842 -0.20629676]
[1.03969541 0.60066853 0.83082535 -0.40362224 0.30658082 0.80554574]]

The number of iterations required to converge: 11

The lowest SSE from the runs: 30685.86585290572

In [118...]

```
# Assign the found clusters for all the rows and append it in assignment_data_number
assignment_data_number = assignment_dataframe.assign(Cluster=lambda x: my_kmeans.labels_)

# View the first few rows of the assignment_data_number DataFrame
assignment_data_number.head()
```

Out[118]:

	S/N	Date	Country	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility
0	1.0	2018-11-23	JP	196.0	0.0	-1.7	79.0	0.5	3.0
1	2.0	2018-11-23	JP	250.0	1.0	-1.6	79.0	0.6	3.0
2	3.0	2018-11-23	JP	161.0	2.0	-1.1	79.0	0.7	3.0
3	4.0	2018-11-23	JP	99.0	3.0	-0.6	76.0	0.5	3.0
4	5.0	2018-11-23	JP	64.0	4.0	-0.3	76.0	0.9	3.0

5 rows × 23 columns

In [119...]

```

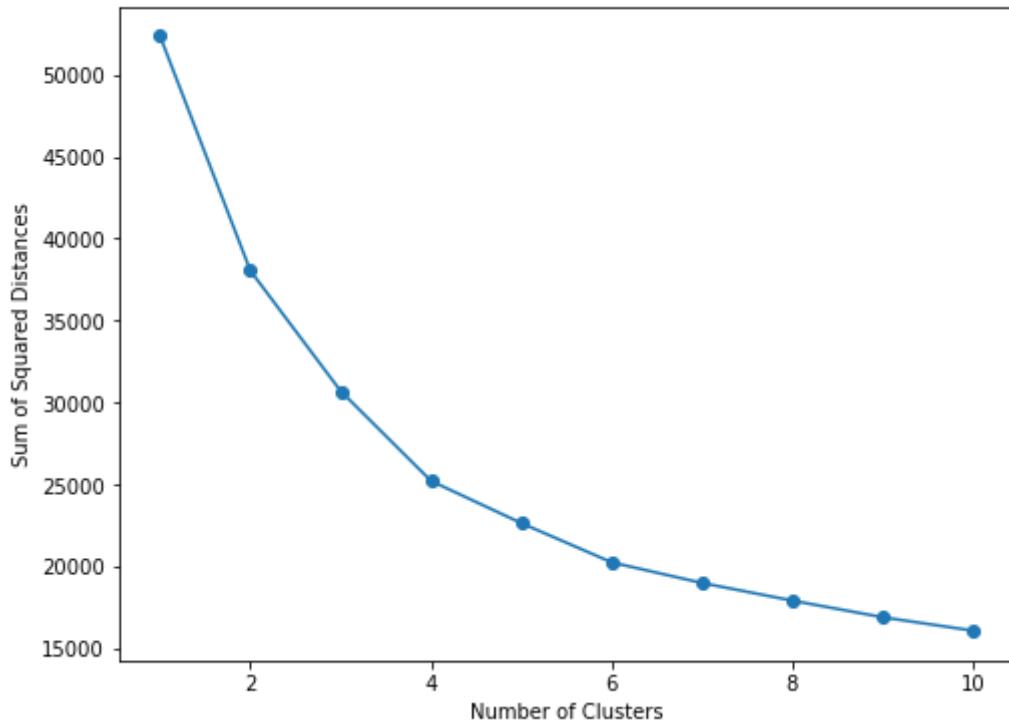
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Calculate the sum of squared distances for different numbers of clusters
sse = []
for k in range(1, 11): # Trying different numbers of clusters from 1 to 10
    kmeans = KMeans(n_clusters=k, init='random', n_init=10, max_iter=300, random_state=42)
    kmeans.fit(scaled_assignment_dataframe)
    sse.append(kmeans.inertia_)

# Plot the elbow plot
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), sse, marker='o')
plt.title('Elbow Plot')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Distances')
plt.show()

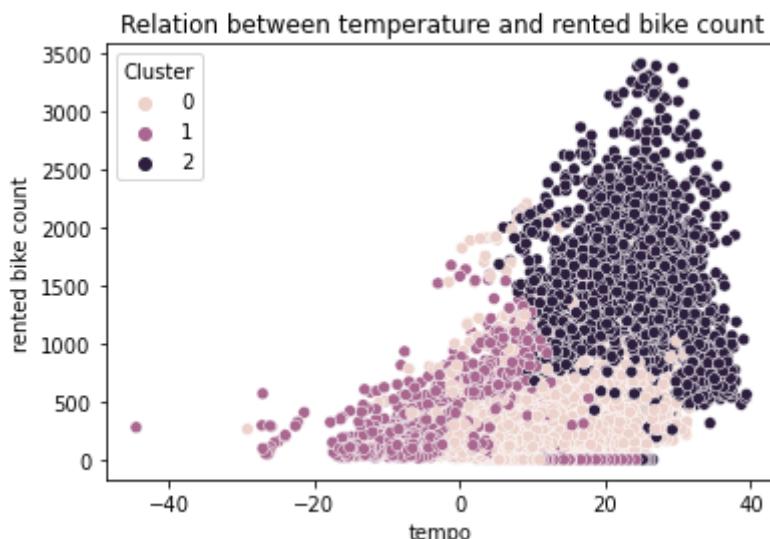
```

Elbow Plot



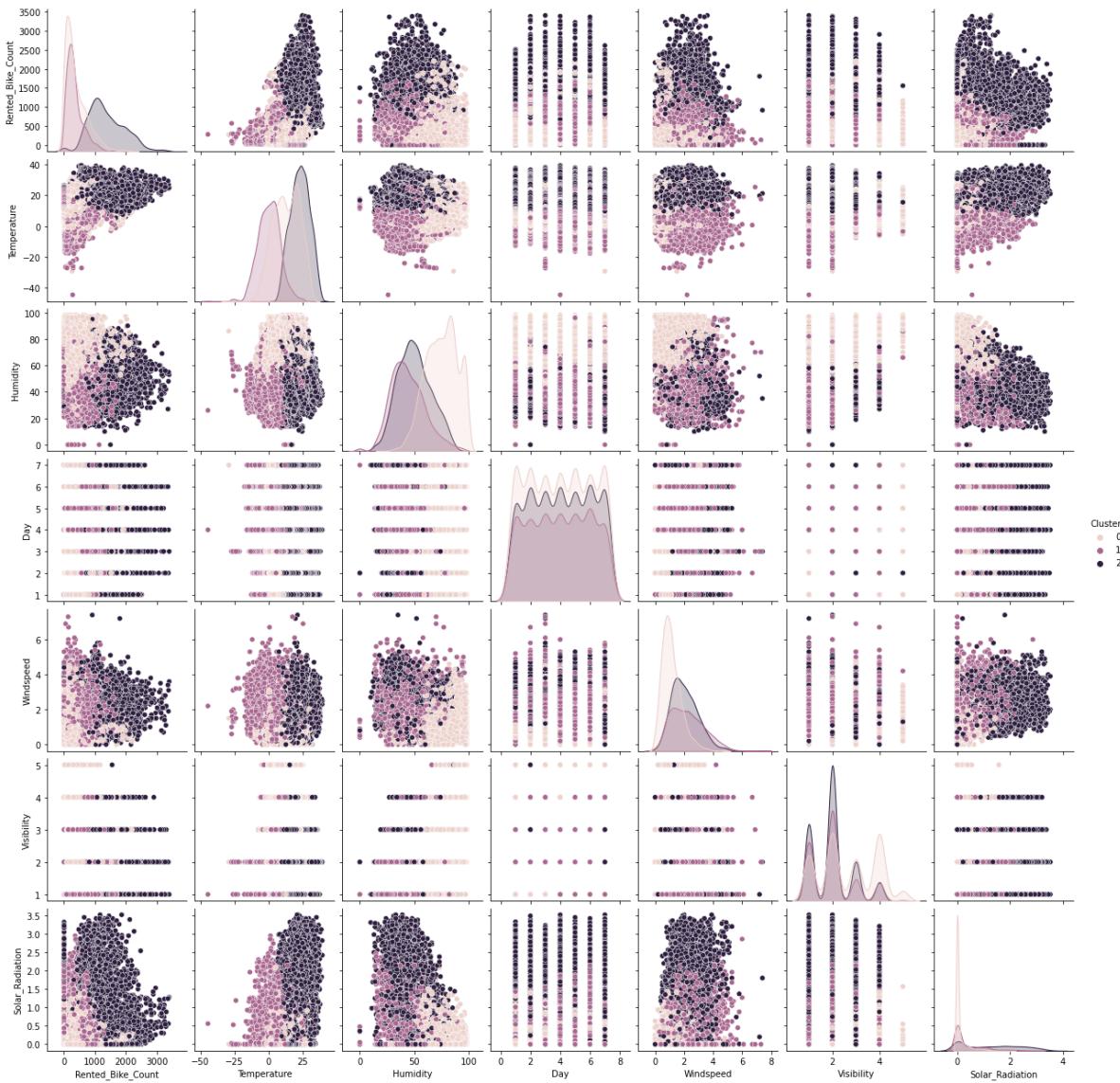
In [120...]

```
import seaborn as sns
sns.scatterplot(x='Temperature', y='Rented_Bike_Count', hue='Cluster', data=assignment_data_number)
plt.title("Relation between temperature and rented bike count")
plt.ylabel('rented bike count')
plt.xlabel('tempo')
plt.show()
```



In [121...]

```
sns.pairplot(assignment_data_number[['Rented_Bike_Count', 'Temperature', 'Humidity']])
plt.show()
```



Report(K-means)

While it is obvious that the model itself is flawed in the sense there are numerous occurrence of outlier and anomalies, I am suspecting it is due to the fact that feature with high variance can lead to outliers being generated, as they can pull cluster centroids towards extreme values. Other proposed potential flaw is that the number of clusters have not been refined or optimal however, my arguement is that the plotted elbow plot has shown rather high smoothness and effectiveness and also clusters might not align with domain-specific patterns. Lastly, I am aware that K-means is sensitive to outliers, as they can disproportionately influence the centroid of a cluster. Outliers can be misclassified as separate clusters or affect the cluster centroids, leading to anomalies in scatterplots.

Cluster 1: These bike rentals are rented in good weather conditions, such as sunny days with clear visibility and low windspeed. They are typically rented by people who are looking for a fun and healthy way to get around, especially on weekends.

Cluster 2: These bike rentals are rented in light rain conditions with moderate visibility and low windspeed. They are typically rented by people who are still willing to bike in light rain, but they may be looking for a more affordable option.

Cluster 3: These bike rentals are rented in heavy rain conditions with poor visibility and high

windspeed. They are typically rented by people who need to get around, but they may not have other options.

In [122...]

```
# Print the data types of columns in scaled_assignment_dataframe
print("Data Types of scaled_assignment_dataframe:")
print(scaled_assignment_dataframe.dtypes)

# Print the data types of columns in assignment_data_number
print("\nData Types of assignment_data_number:")
print(assignment_data_number.dtypes)
```

```
Data Types of scaled_assignment_dataframe:
Rented_Bike_Count      float64
Hour                   float64
Temperature            float64
Humidity               float64
Windspeed              float64
Solar_Radiation        float64
dtype: object
```

```
Data Types of assignment_data_number:
S/N                  float64
Date                 datetime64[ns]
Country              object
Rented_Bike_Count    float64
Hour                 float64
Temperature          float64
Humidity             float64
Windspeed             float64
Visibility            float64
Dewpoint_Temp         float64
Solar_Radiation       float64
Rainfall              float64
Snowfall              float64
Open                 object
Hit_Sales             int64
Region               object
Latitude              float64
Longitude             float64
Relative_humidity     float64
Heat_Index             float64
Windchill              float64
Day                  float64
Cluster               int32
dtype: object
```

In [123...]

```
# Columns to remove
columns_to_remove = ['S/N', 'Date', 'Country', 'Region', 'Longitude', 'Latitude']

# Remove the specified columns from scaled_assignment_dataframe
assignment_data_number = assignment_data_number.drop(columns=columns_to_remove)

# Print the updated dataframe
print(assignment_data_number.head())
```

	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility	\
0	196.0	0.0	-1.7	79.0	0.5	3.0	
1	250.0	1.0	-1.6	79.0	0.6	3.0	
2	161.0	2.0	-1.1	79.0	0.7	3.0	
3	99.0	3.0	-0.6	76.0	0.5	3.0	
4	64.0	4.0	-0.3	76.0	0.9	3.0	

	Dewpoint_Temp	Solar_Radiation	Rainfall	Snowfall	Open	Hit_Sales	\
0	-4.8	0.0	0.0	1.0	Yes	0	
1	-4.7	0.0	0.0	1.0	Yes	0	
2	-4.2	0.0	0.0	1.0	Yes	0	
3	-4.3	0.0	0.0	1.0	Yes	0	
4	-4.0	0.0	0.0	1.0	Yes	0	

	Relative_humidity	Heat_Index	Windchill	Day	Cluster
0	80.612245	-9.480569	1.283723	5.0	0
1	80.612245	-9.327687	1.063320	5.0	0
2	80.612245	-8.566436	1.285089	5.0	0
3	77.551020	-7.883928	2.357738	5.0	0
4	77.551020	-7.431513	1.636654	5.0	0

In [124...]

```

from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler

# Define a placeholder for the scaler
scaler = StandardScaler()

# Select only the numerical columns for scaling
numerical_columns = ['Rented_Bike_Count', 'Temperature', 'Humidity', 'Windspeed',
                     'Solar_Radiation']

# Apply the scaler to the DataFrame that only contains numerical attributes
scaled_assignment_dataframe = scaler.fit_transform(assignment_data_number[numerical_columns])
scaled_assignment_dataframe = pd.DataFrame(scaled_assignment_dataframe, columns=numerical_columns)

# Define the placeholder for the AgglomerativeClustering algorithm
n_clusters = 3 # Adjust this value
linkage = 'ward' # Adjust this value ('ward', 'complete', 'average', etc.)
my_agglomerative = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage)

# Fit the AgglomerativeClustering model
assignment_data_number['Cluster'] = my_agglomerative.fit_predict(scaled_assignment_dataframe)
# View the result
assignment_data_number.head()

```

Out[124]:

	Rented_Bike_Count	Hour	Temperature	Humidity	Windspeed	Visibility	Dewpoint_Temp	Sola
0	196.0	0.0	-1.7	79.0	0.5	3.0	-4.8	
1	250.0	1.0	-1.6	79.0	0.6	3.0	-4.7	
2	161.0	2.0	-1.1	79.0	0.7	3.0	-4.2	
3	99.0	3.0	-0.6	76.0	0.5	3.0	-4.3	
4	64.0	4.0	-0.3	76.0	0.9	3.0	-4.0	

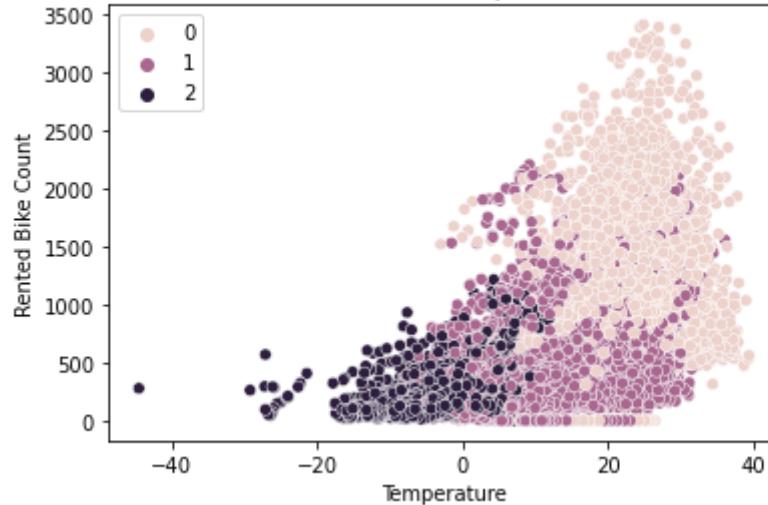
In [125...]

```

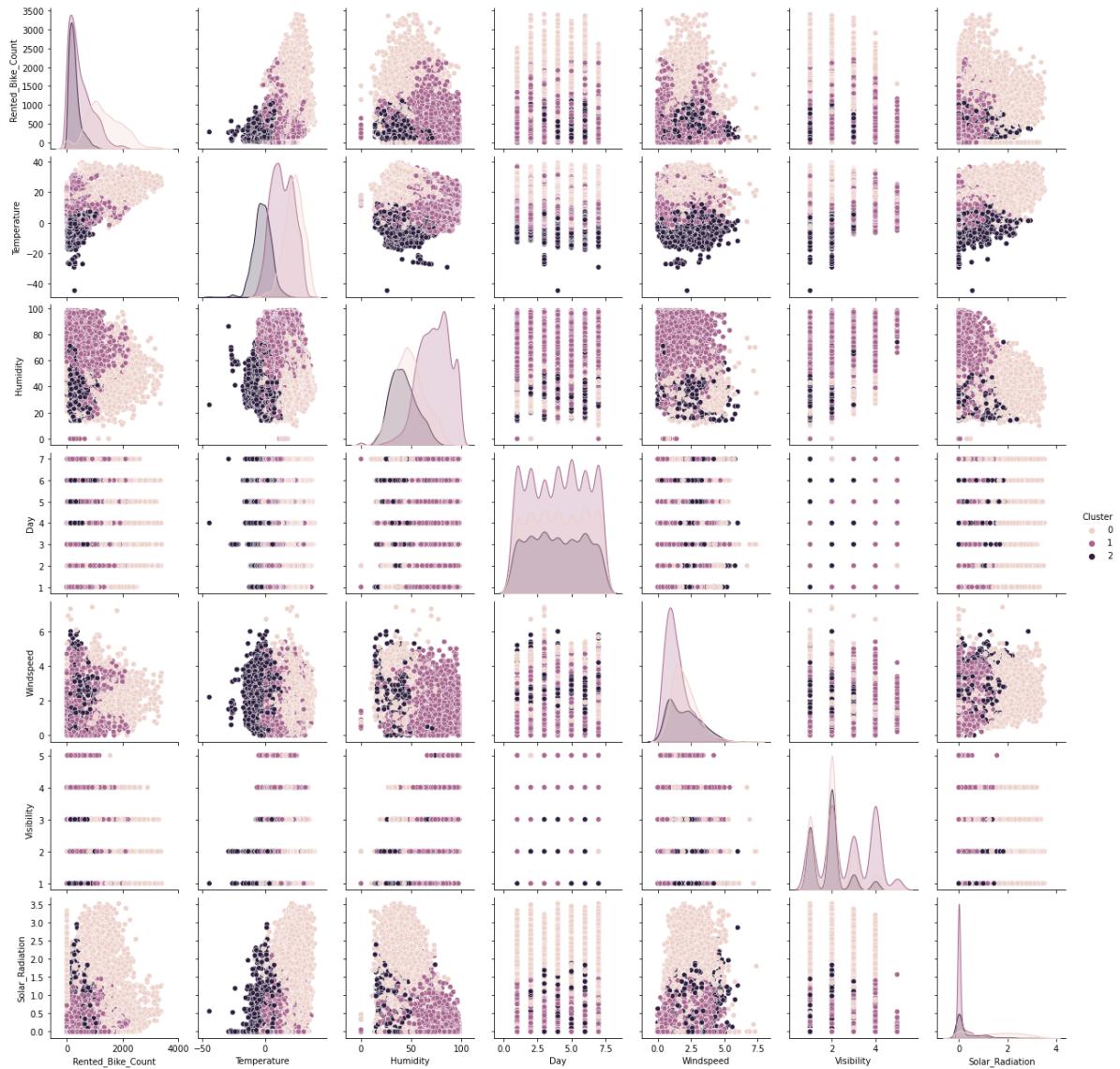
sns.scatterplot(x='Temperature', y='Rented_Bike_Count', hue='Cluster', data=assignment_data_number)
plt.title("DBSCAN Clusters: Relation between Temperature and Rented Bike Count")
plt.xlabel('Temperature')
plt.ylabel('Rented Bike Count')
plt.legend()
plt.show()

```

DBSCAN Clusters: Relation between Temperature and Rented Bike Count



```
In [126...  
sns.pairplot(assignment_data_number[['Rented_Bike_Count', 'Temperature', 'Humidity',  
plt.show()
```



Report(hierarchical clustering)

While there are several key difference between hierachial clustering and K means, some of its key differences are the different approach between the 2 algorithm, specification of numbers of clusters, centroids, cluster shapes etc. But the main reason I chose hierarchical clustering is because after doing some vizual inspection and analysis of I realize that regular k means is very very sensitive to outliers. Thanks to its linkage criteria, hierarchical clustering is better at dismissing outliers as evident in some of the pairplots. But the conclusion I have achieve is more or less the same.

Cluster 1: These bike rentals are rented in good weather conditions, such as sunny days with clear visibility and low windspeed. They are typically rented by people who are looking for a fun and healthy way to get around, especially on weekends.

Cluster 2: These bike rentals are rented in light rain conditions with moderate visibility and low windspeed. They are typically rented by people who are still willing to bike in light rain, but they may be looking for a more affordable option.

Cluster 3: These bike rentals are rented in heavy rain conditions with poor visibility and high windspeed. They are typically rented by people who need to get around, but they may not have other options.

Remember to highlight in the report - Why or why not you have adopted the suggestions provided by ChatGPT or other literatures - If adopted, indicate clearly where to find the supporting evidences or coding cells implementing them - Be specific in what tweak(s) is/are required to make the suggestions work or make predictive model(s) improve.

Recommendations & Conclusion

Conclusion

Throughout this entire assignment through numerous modelling and visualization and studying of the data itself. It is evident that yes there is relation between the environmental condition and ohaiyo rented bike count. However, the correlation is really not very impactful. From the strange fluctuations of the graph to the low readings on accuracy for the model despite numerous attempt to fine tune the factors and model type, I think it is safe to say, the conditions might not have as big of an impact on the rented bike count as I imagined. I am willing to say that this can be due to other factors. We all aware that cycling especially if someone is renting it, is mostly only use for recreation. Should a user want to use a bike for a prolong period of time, they might choose to invest and buy their own. Therefore, I think it is a legitimate suggestion to say that ohaiyo business normally flourish when it is during holidays as well as weekends. **BUT** there are severral occassions and variables that shows a stark relation to the rented bike count. From my vizualization modelling and other means of data study, it is fair to say a few variables play a relatively big role on how the data fluctuates. Some of which are: Temperature, Day, Hour, heat_index and windchill(Transform column). Now if we analysis the variables that I deem to have major impact on the rented bike count. We can see that they are rather "Big" variables. In the

sense that they are either multiple variables combining to form it, or they are just strong and informing variables individually. One of the biggest evidence of this is my heat index and windchill column. They are effectively combining the existing variable to give a more consolidated and informed variable. The reason why you don't see things like solar radiation, rainfall or snowfall is because they more or less contribute to the variable: Temperature. But more importantly, it just shows that it is safe to say that ohaiyo's customer do not really see the other variables as reason to stop or encouraging them from renting bikes to partake in as we suspected, recreational activities. Now that I have finished my conclusion, next we can discuss my recommendation.

Recommendations

Since we have established our few key variables that affect the business performance, I would suggest that depending on days and times, we extend or shortened operation hours. This way we can have more room for revenue to flow in while at the same time we will control expenses such as utilities as well as salaries expenditures. My next suggestion is to offer bikes suitable for different weather conditions. For example, offer electric bikes on hot days to help customers manage longer distances without feeling exhausted. Since we have established that temperature still plays a part in the overall performance of ohaiyo. It will be beneficial for the business to invest in bikes that still caters to the comfort of its customer regardless of the weather conditions. And since we also know things like rainfall and snowfall won't stop people from renting bikes, I think it is also good for the business to take into consideration the overall safety of the bikes and its user. In conclusion, from my study of ohaiyo dataset, I would suggest 2 things. 1: adjust the operation hours. When demand is high, cater to it. When demand is low, cut our losses. 2: Invest in different types of bike for the comfort and safety of our customers in any and all weather and environmental condition.

References

The content generated by AI tools are not retrievable except by the user who generated them, so they are considered non-recoverable sources. For non-recoverable sources:

- do not include in a reference list
- cite within the text as personal communications or correspondence

Based on APA 7th edition referencing format, we can adopt the format of (Communicator, personal communication, Month Day, year) E.g. (Paraphrase from OpenAI's ChatGPT AI language model, personal communication, March 8, 2023).

Enter your list here.

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

OpenAI. "Strategies and Aspects of Data Science for Enhancing Bike Rental Count." ChatGPT, a language model AI. August 1, 2023.

Bard, Google AI. "Can you explain why DBSCAN is harder to utilize compared to hierarchical clustering." Bard, Google AI, August 7. Accessed August 12

Declaration on the use of Generative AI tools for assignments

Describe how you have used Generative AI tools such as ChatGPT or Dall.E-2 in your assignment. Show snapshots of the conversations with the AI tool (i.e., the prompts you used and the response you get from the AI tool).

Drag and drop into the markdown cell(s) below.

Since I have already addressed my usage on AI tools for transformation and everything else before, I will now just address my use of AI for data modelling. (no particular order)

This is the conversation of me getting some suggestion on how to avoid a horrible elbow chart. And this is true. after removing leaving only my continuous numerical data, the curve

got a lot smoother



but why is it after 3, there is still a fluctuation of decrease till like point 10? it is a little janky after 3



If you observe a fluctuation in the decrease of the sum of squared distances (inertia) after the initial drop around 3 clusters, it could indicate that the data might not have a very clear and distinct cluster structure. There are a few reasons why this behavior might occur:

1. **Noise or Variability:** If the data has a lot of noise or variability, it might not exhibit a clear separation into distinct clusters. This can lead to fluctuations in the inertia values as you increase the number of clusters.
2. **Overfitting:** Adding more clusters can lead to overfitting, where the model captures noise in the data rather than meaningful patterns. This can result in clusters that are not well-defined and cause fluctuations in the elbow plot.
3. **Inherent Complexity:** Some datasets might have inherent complexities that make it difficult for the algorithm to find a clear separation into clusters. This can lead to a lack of stability in the elbow plot.
4. **Sparse Data:** If the data is sparse or high-dimensional, clustering algorithms might struggle to find meaningful patterns, leading to erratic behavior in the elbow plot.
5. **Algorithm Sensitivity:** K-means is sensitive to the initial placement of centroids. Different random initializations can lead to varying results, which might contribute to fluctuations.

Given the janky behavior you've observed after 3 clusters, it's a good idea to explore other methods and validate the stability of the clustering solution. You could consider trying different clustering algorithms, exploring lower-dimensional representations of the data, or seeking insights from domain experts to help make a more informed decision about the appropriate number of clusters. Remember that clustering can be challenging, and it's important to consider multiple factors and not rely solely on one method.



so... remove variables that I suspect to be affecting the model?



Yes, removing or modifying variables that you suspect might be affecting the cluster

Regenerate



Send a message

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. [ChatGPT August 3 Version](#)

despite using scikit learn to understand the difference between the different types of linear regression models, I still wanted to have a clear understanding the difference between

elastic net and regular linear regression models



what difference is there for elastic net linear regression and simple linear regression. is it more accurate?



Elastic Net regression is an extension of simple linear regression that combines both L1 (Lasso) and L2 (Ridge) regularization techniques. Simple linear regression aims to model the relationship between a dependent variable and one independent variable by fitting a straight line to the data points. Elastic Net regression, on the other hand, is used when you have multiple independent variables and want to prevent overfitting by incorporating both L1 and L2 regularization.



Here are the key differences between Elastic Net regression and simple linear regression:

1. **Regularization:** Simple linear regression does not involve any regularization. It aims to minimize the sum of squared differences between observed and predicted values. Elastic Net regression, on the other hand, introduces regularization terms (L1 and L2) to the objective function, which helps prevent overfitting and can lead to better generalization.
2. **Multiple Independent Variables:** Simple linear regression works with only one independent variable. Elastic Net regression is designed for situations where you have multiple independent variables. It can handle situations where there might be collinearity (high correlation) between these variables.
3. **L1 and L2 Regularization:** Elastic Net combines both Lasso (L1) and Ridge (L2) regularization. L1 regularization encourages sparsity in the model by driving some coefficient values to zero, effectively selecting a subset of important features. L2 regularization helps to control the magnitude of coefficient values. By combining these two types of regularization, Elastic Net can provide a balance between feature selection and controlling coefficient magnitudes.
4. **Parameter Tuning:** Elastic Net regression introduces two hyperparameters, alpha and the mixing parameter, which control the amount of L1 and L2 regularization applied. Tuning these hyperparameters is important to find the right balance between feature selection and coefficient magnitude control. In simple linear regression, you don't have these additional hyperparameters to tune.
5. **Accuracy:** Whether Elastic Net regression is more accurate than simple linear regression depends on the dataset and the nature of the problem. In cases where there is multicollinearity (high correlation among independent variables) or when you suspect that some variables are not relevant, Elastic Net might outperform simple linear regression by effectively handling these issues and leading to a more generalized model. However, in cases where you have a small number of independent variables and no multicollinearity, simple linear regression might be sufficient.

In summary, Elastic Net regression is more versatile and can be particularly useful when dealing with datasets with multiple correlated independent variables, as it combines the benefits of L1 and L2 regularization. Whether it is more accurate than simple linear regression depends on the specific characteristics of your data and the underlying relationship between variables.

after the success of elastic net linear regression, I wanted to implement the logistic regression version but I had to make sure



is elastic net linear regression have the same logic as elastic net logistic regression



The logic behind Elastic Net regularization is similar for both linear regression and logistic regression, but there are some differences due to the nature of the dependent variable and the underlying models.



1. Dependent Variable Type:

- Linear Regression: In linear regression, the dependent variable is continuous, representing a numeric value. The goal is to fit a linear equation that best represents the relationship between the independent variables and the continuous dependent variable.
- Logistic Regression: In logistic regression, the dependent variable is binary or categorical, representing classes or probabilities. The goal is to model the probability of an observation belonging to a certain class. The logistic function (sigmoid) is used to map the linear combination of independent variables to the range [0, 1].

2. Loss Function:

- Linear Regression: The loss function in linear regression is typically the mean squared error (MSE), which measures the difference between the predicted values and the actual values.
- Logistic Regression: The loss function in logistic regression is typically the log loss (also known as cross-entropy loss), which measures the difference between predicted probabilities and the actual binary outcomes.

3. Regularization:

- Elastic Net Regularization: As mentioned earlier, Elastic Net regularization combines L1 (Lasso) and L2 (Ridge) regularization. The regularization terms are added to the loss function to control the magnitude of coefficients and encourage feature selection.

The logic behind Elastic Net regularization remains similar in both linear and logistic regression. The goal is to strike a balance between fitting the data well and preventing overfitting by introducing regularization. The regularization terms add penalties to the loss function, influencing the coefficient values of the independent variables.

However, because logistic regression deals with binary or categorical outcomes and uses a different loss function, the implementation details and mathematical expressions for the regularization terms can differ slightly between linear and logistic regression. Nevertheless, the overall concept of finding a balance between fitting the data and regularization holds true for both types of regression when using Elastic Net regularization.

I wanted to utilize both regularized and elastic net linear regression



is regularized logistic regression more stable than elastic net logistic regression?



Regularized logistic regression and elastic net logistic regression serve similar purposes in terms of mitigating overfitting and improving model stability. However, whether one is more stable than the other can depend on the specific characteristics of your dataset and the degree of regularization you apply. Let's explore this further:

1. Regularized Logistic Regression:

- Regularized logistic regression includes either L1 (Lasso) or L2 (Ridge) regularization, but not both simultaneously. You choose either L1 or L2 regularization based on your problem's characteristics and goals.
- L1 regularization encourages sparsity by driving some coefficient values to exactly zero. This can lead to feature selection, meaning some features are effectively ignored by the model.
- L2 regularization controls the magnitude of coefficient values, preventing them from becoming too large.

2. Elastic Net Logistic Regression:

- Elastic net logistic regression combines both L1 and L2 regularization. It aims to provide a balance between feature selection (L1) and coefficient magnitude control (L2).
- The mixing parameter in elastic net (usually denoted as alpha) determines the proportion of L1 and L2 regularization applied. When alpha is 0, the model behaves like Ridge regression. When alpha is 1, it behaves like Lasso regression. Intermediate values balance the two.

In terms of stability:

- **Regularized Logistic Regression Stability:** Regularized logistic regression with either L1 or L2 regularization can improve model stability by preventing overfitting. However, if the dataset has a high degree of multicollinearity (strong correlation between features), L1 regularization (Lasso) might lead to instability as it selects only one of the correlated features while driving others to zero.
- **Elastic Net Logistic Regression Stability:** Elastic net logistic regression aims to address the potential instability caused by L1 regularization in the presence of multicollinearity. By combining L1 and L2 regularization, it can maintain feature stability while still promoting sparsity and controlling coefficient magnitudes.

In general, if you suspect multicollinearity is present in your dataset, elastic net logistic regression can be more stable than regularized logistic regression with L1 regularization alone. Elastic net strikes a balance between the strengths of L1 and L2 regularization, providing better stability and adaptability to various datasets.

However, it's important to note that the choice between regularized logistic regression and elastic net logistic regression ultimately depends on the nature of your data and the specific goals of your analysis. Experimenting with both approaches and tuning the hyperparameters (alpha and the regularization strength) is recommended to find the best model for your particular problem.