Internet Based Data Repositories: Accessing, Storage, Manipulation and Analysis

By student 21025647

# Introduction

In the era of digitalisation, becoming an expert in data management (which includes data access, storage, manipulation, and analysis) has become crucial for companies in many different industries. These crucial processes, which form the basis for well-informed decision-making and strategic planning, are clarified by this thorough investigation. When we explore the complex world of data, it becomes clear that the value of data lies not only in its acquisition but also in the methodical way in which it is managed.

Accessing data starts with the first problem that organisations encounter: finding pertinent data from online resources in an effective and morally sound manner. This step is crucial since the insights and results that may be drawn from the data are directly influenced by the quality and relevancy of the data that can be accessed. Next, data storage takes the stage, emphasising the value of scalable, secure, and effective data storage options. Whether it is relational databases, cloud storage, or sophisticated data warehousing, choosing the right storage technology is essential to preserving data integrity and enabling seamless data retrieval. The process of transforming data to make it ready for analysis is handled by manipulating the data. To do this, the data must be meticulously cleaned, organised, and enhanced. These processes call for a thorough understanding of the underlying patterns and any abnormalities in the data.

The process ends when the prepared data is carefully analysed to yield insights that can be put into practice. This phase reveals how data might impact organisational plans and operations using statistical analysis, machine learning models, and data visualisation tools. Collectively, these phases establish an all-encompassing structure for managing data, highlighting the vital part that methodical data handling plays in using information as a tactical advantage.

# Accessing and Storing Data

Accessing started by retrieving data from an online database, utilising a dataset supplied by a CarSharing service. The dataset is kept as an Excel CSV file and contains both numerical and categorical data. The data was gathered on an hourly basis. The data included temporal variables such as the day, hour, and season, along with meteorological details such as weather conditions, temperature, humidity, and wind speed. The 'demand' column in the dataset measures the extent to which customers are inclined to hire a car at various periods, with higher demand suggesting a stronger readiness to rent.

After accessing the data, the attention turned to storing the data. The CarSharing dataset may be kept in many repositories, such as relational databases, cloud storage, and data warehouses, each providing distinct benefits. Relational databases are well-suited for managing operational data because they provide for organised storage and the ability to perform detailed queries. Cloud storage provides the flexibility to easily adjust the size and capacity of storage, as well as the ability to access data from another location. These features are particularly important for businesses that operate in various locations. Data warehouses consolidate data from several sources into a central repository, facilitating complex analytics and extensive reporting. For this purpose, the SQLite3 relational database system, integrated with Python, was selected due to its ability to both store and query data, which were extensively used in the research.

# Data Manipulation

Data manipulation begins with the initial stage of data cleansing. This involves the process of recognising and rectifying (or eliminating) mistakes and discrepancies, including anomalies that deviate from the established pattern of the remaining data, duplicate entries that might distort outcomes, and absent values that can result in inaccurate analysis. Arranging the data is another crucial element of data manipulation. This involves organising the data in a manner that is consistent with the objectives of the analysis. After the data has been cleaned and organised, it frequently requires transformation or enrichment to be entirely valuable for analysis. This may include generating new variables that are more informative for the specific questions under consideration.

Data manipulation frequently includes the process of normalisation or standardisation, especially when preparing data for machine learning models. These techniques standardise the scales of the data to ensure comparability across different variables. Feature engineering is a crucial aspect of data manipulation. It entails generating new variables based on the available data to aid in predicting or explaining the specific aberration of interest. Efficiently manipulating data leads to more precise discovery and forecasting, with the ultimate objective of transforming raw data into a valuable strategic asset.

# Data Analysis

In the Data Analytics process, hypothesis testing served was used for exploring the relationships between weather conditions (temperature, feels-like temperature, humidity, windspeed) and car-sharing demand. Employing Pearson's correlation coefficient and ANOVA facilitated the identification of significant relationships: temperature and feels-like temperature exhibited a positive correlation with demand, suggesting an increased preference for car-sharing during warmer conditions. Conversely, humidity displayed a negative correlation, indicating a decreased demand in more humid conditions, while windspeed's positive correlation pointed to nuanced user behaviour influenced by weather factors. These statistical tests revealed the dynamics between environmental conditions and consumer behaviour, offering the Car sharing service valuable insights into demand.

The dataset was used to uncover seasonal and cyclic patterns, particularly for the year 2017. Seasonal decomposition of time series data revealed pronounced trends and fluctuations, with temperature demonstrating a clear seasonal variation that mirrored conventional expectations—increased demand during warmer months and a decline during cooler periods. This pattern further revealed the impact of temperature on demand, unlocking Car user behaviour trends. In contrast, humidity and windspeed showed varied impacts, suggesting a complex interplay of factors influencing demand.

Predictive modelling was also used and further enriched the analysis process. An ARIMA model forecasted weekly average demand rates with commendable accuracy, highlighting the model's use in capturing time series trends and fluctuations. Additionally, the comparative analysis of a Random Forest regressor and a Deep Neural Network, using the Mean Squared Error metric, showed the strengths of simpler models in certain contexts. Despite its sophistication, the Neural Network's performance was eclipsed by the Random Forest model, suggesting that complexity does not always equate to superior predictive power, especially in scenarios with limited data.

Classification and clustering techniques used further outlined the dataset's characteristics. Demand rates were bifurcated relative to the average, and models like Random Forest, Logistic Regression, and SVM were deployed to predict these categories. Random Forest's superior accuracy revealed its effectiveness in managing the data's nonlinearities. Meanwhile, clustering temperature data via KMeans revealed a tripartite seasonal variation, offering an in-depth understanding of the temperature's impact on car-sharing demand.

Collectively, these analytical methods gave an extensive understanding of the data. Each technique unravelled different dimensions of the dataset.

# Task 1 : Data Management

1. An SQLite database was imported to house the CarSharing data. Following the importation of data into a table named "CarSharing", a backup table was created to safeguard against data loss, upholding fundamental practice in data management to ensure information integrity and reliability.

### 2. A new column, "temp\_category", was introduced to classify the "feels-like" temperature into "Cold", "Mild", and "Hot" categories. This categorisation assisted with analysing how temperature influences demand, causing a clearer understanding of customer behaviour across different weather conditions.

3. A dedicated table named "temperature" was created to focus on temperature-related data, which involved selecting and isolating temperature and category information. Removing the raw temperature data from the main CarSharing table streamlined the dataset.

4. To facilitate a more systematic analysis of weather conditions on service usage, distinct weather conditions were encoded with specific numerical values, added to the database as "weather\_code".

5. The creation of a separate "weather" table to house both weather conditions and their corresponding codes assists with strategic data organisation. By removing the original "weather" column from the CarSharing table, the database structure was optimised for a more focused analysis.

6. A new table named was established, categorising each row's timestamp into hour, weekday, and month. This structuring was crucial for temporal analysis, allowing for an in-depth exploration of demand patterns across different timescales.

7A. The analysis discovered the exact date and time of peak demand in 2017.

7B. By comparing average demand across weekdays, months, and seasons, patterns were revealed, and peak demand times could be pinpointed.

7C. Focusing on specific weekdays, hourly demand rates were analysed, this helps assist with finding peak times for the Car Sharing company.

7D. The examination of weather conditions throughout 2017 and their impact on demand assists in the detection of correlation between the two.

7E. By isolating the month with the highest average demand for a detailed comparison, the analysis provided a better view of how specific conditions affect service usage.

# Task 2: Data Analysis

1. The CarSharing dataset was imported into a CSV format for preprocessing, a crucial step for ensuring data quality before analysis. The process involved dropping duplicate rows to eliminate redundancy and handling null values by imputing numerical columns with the mean and categorical columns with the mode, ensuring no loss of data integrity.

2. hypothesis testing was used to ascertain the relationship between several features (temperature, feels-like temperature, humidity, windspeed) and the demand rate. Pearson's correlation coefficient and ANOVA were the chosen statistical methods. The results were significant for all tests as both temperature and feels-like temperature showed a positive correlation with demand, indicating higher demand at warmer temperatures. Humidity had a negative correlation, implying lower demand at higher humidity levels and windspeed also demonstrated a positive correlation, suggesting that demand is indeed influenced by these weather-related factors.

3. Analysis of the 2017 data revealed clear seasonal and cyclic patterns in temperature, humidity, windspeed, and demand. Seasonal decomposition of time series data indicated the presence of underlying trends and seasonal fluctuations. The temperature data exhibited a clear seasonal trend, with higher temperatures peaking during the mid-year months, reflective of summer, and dipping during the start and end of the year, indicative of the cooler winter months. This annual temperature cycle is typical in many regions and corresponds with general user behaviours, where the use of car-sharing services may increase during warmer periods due to a preference for travel.

Humidity trends were not too similar to temperature, with higher humidity levels often present during the cooler months. This could be due to various climatic factors such as increased rainfall during colder periods. The data showed less pronounced cyclicality on a daily or weekly basis, suggesting that humidity's influence on demand is more complex and less direct than temperature. Windspeed fluctuations did not display as clear a seasonal pattern as temperature. Demand data reflected the combined influence of these variables, with several interesting patterns. An overall seasonal trend mirrored that of the temperature, with demand increasing during warmer months and decreasing when it was cooler.

4. An ARIMA model was used to predict the weekly average demand rate, with 30% of the data reserved for testing. The model selection process included the assessment of stationarity, and differencing was employed to stabilise the time series. The model indicated an acceptable level of predictive capability

5. Both Random Forest regressor and a Deep Neural Network was compared using the Mean Squared Error (MSE) metric. The Random Forest model yielded a lower MSE, indicating better performance. The Neural Network, while more complex, did not outperform the simpler Random Forest, which could be due to its greater need for more fine-tuning. Random Forest Regressor can handle smaller datasets well, whereas the Deep Neural Network needs lots of data to get reliable results.

6. Demand rates were categorised into two groups relative to the average demand rate and labelled accordingly. Three classifiers - Random Forest, Logistic Regression, and SVM - were then employed to predict these labels. The Random Forest Classifier achieved the highest accuracy, demonstrating better effectiveness in handling the nonlinearities and interaction effects present in the data.

7.The task of clustering temperature data from 2017 using KMeans explored the uniformity of clusters at different values of k (2, 3, 4, and 12). Uniformity was assessed based on the variance of sample counts in each cluster. The analysis concluded that k=3 produced the most uniform clusters, highlighting a potential tripartite seasonal variation in temperature.

# Conclusion

This thorough examination the data management procedures, (include data access, storage, manipulation, and analysis) highlights the importance of these activities when using information as a strategic resource across different sectors. This study showed the fundamental importance of obtaining accurate data from a CarSharing service's file and the subsequent actions needed to effectively use its potential. The use of SQLite3 for data storage emphasised the need of scalable, secure, and efficient storage solutions, a necessity for preserving data integrity and enabling smooth access.

Data manipulation, which involved cleaning, organising, and transforming the data, was used in preparing the dataset for thorough analysis. This phase guaranteed that the data was not only precise and free of irregularities, but also organised it in a manner that fits with the analytical goals outlined. The manipulation process established the foundation for data analysis, where hypothesis testing and statistical techniques such as Pearson's correlation coefficient and ANOVA uncovered meaningful connections between meteorological conditions and the demand for car-sharing. The results, in addition to the analysis of temporal patterns and the successful use of predictive modelling, provided detailed insights into consumer behaviour and factors influencing demand.

In addition, the study's methodical approach to data organisation, achieved through the development of specialised tables and categories, enabled a more targeted investigation of the impact of temperature and weather conditions on service use. The structured data environment facilitated the use of advanced analytical methods such as classification, clustering, and time series analysis. These approaches collectively enhanced the understanding of the complexities inside the dataset.

To summarise, this investigation on online repository data management and analysis demonstrated the essential need of organised data manipulation when looking to obtain valuable insights from detailed datasets. The findings provided a comprehensive analysis of the aspects that influence car-sharing demand in connection to environmental conditions and much more.

The effective implementation of data management principles serves as a strong confirmation of the significant impact that data has in guiding well-informed and evidence-based initiatives in the current digital age.

# Appendix

**DATA MANAGEMENT TASK**

**Code 1.1**

################################## DATA MANAGEMENT ##################################

######QUESTION 1- Create an SQLite database and import the data into a table named “CarSharing”. Create

#a backup table and copy the whole table into it. ######

import pandas as pd

import sqlite3

# Loading necessary Libaries and modules

df = pd.read\_csv(r'CarSharing.csv')

# Connecting to a new SQLite database

conn = sqlite3.connect('car\_sharing.db')

# Creating a cursor object using the connection

cur = conn.cursor()

# Importing data into a table named "CarSharing"

df.to\_sql('CarSharing', conn, if\_exists='replace', index=False)

# Querying ancd checking the "CarSharing" table and print the first 5 rows

cur.execute('SELECT \* FROM CarSharing LIMIT 5;')

rows = cur.fetchall()

print("The first 5 rows of the 'CarSharing' table:")

for row in rows:

print(row)

# Create a backup table and copy the whole table into it

df.to\_sql('CarSharing\_backup', conn, if\_exists='replace', index=False)

# Query the "CarSharing\_backup" table and print the first 5 rows

cur.execute('SELECT \* FROM CarSharing\_backup LIMIT 5;')

backup\_rows = cur.fetchall()

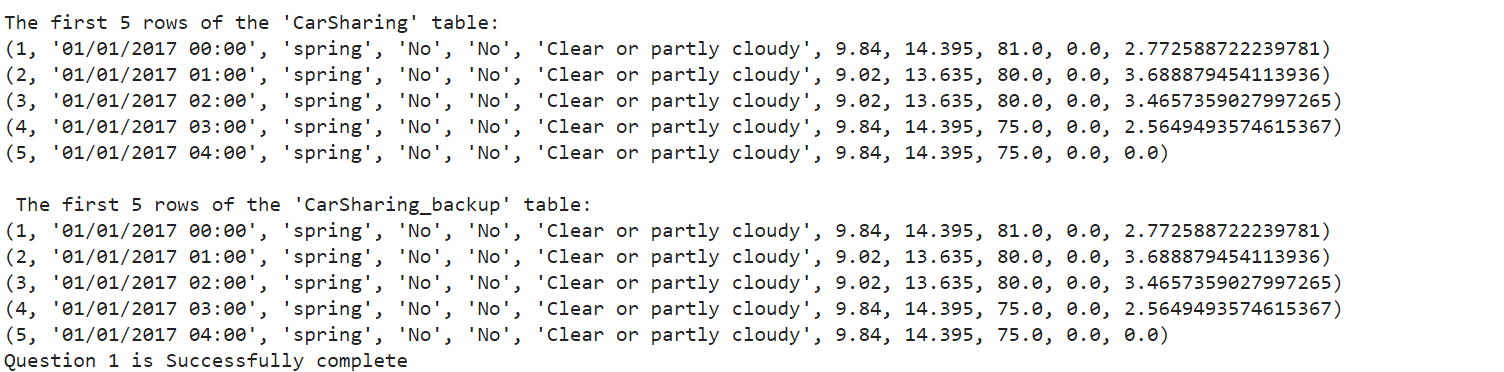
print("\n The first 5 rows of the 'CarSharing\_backup' table:")

for row in backup\_rows:

print(row)

print ("Question 1 is Successfully complete")

**Result 1.1**

**Code1.2**

######QUESTION 2- Add a column to the CarSharing table named “temp\_category”. This column should

#contain three string values. If the “feels-like” temperature is less than 10 then the

#corresponding value in the temp\_category column should be “Cold”, if the feels-like

#temperature is between 10 and 25, the value should be “Mild”, and if the feels-like

#temperature is greater than 25, then the value should be “Hot”. ######

# Adding a column for the temperature category

conn.execute('''

ALTER TABLE CarSharing

ADD COLUMN temp\_category TEXT;

''')

# Updating the column based on weather conditions

conn.execute('''

UPDATE CarSharing

SET temp\_category = CASE

WHEN temp\_feel < 10 THEN 'Cold'

WHEN temp\_feel BETWEEN 10 AND 25 THEN 'Mild'

WHEN temp\_feel > 25 THEN 'Hot'

END;

''')

# Querying the "CarSharing" table exists then printing the first 5 rows to verify the 'temp\_category' update is there

cur.execute('SELECT \* FROM CarSharing LIMIT 5;')

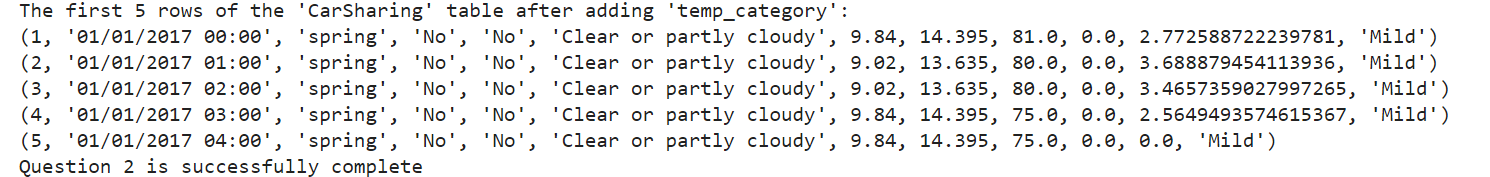
rows = cur.fetchall()

print("The first 5 rows of the 'CarSharing' table after adding 'temp\_category':")

for row in rows:

print(row)

print ("Question 2 is successfully complete")

**Result 1.2**

**Code 1.3**

######QUESTION 3 - Create another table named “temperature” by selecting the temp, temp\_feel, and

#temp\_category columns. Then drop the temp and temp\_feel columns from the CarSharing

#table. ######

# Creating another table named "temperature" with selected columns as requested

conn.execute('''

CREATE TABLE temperature AS

SELECT id, temp, temp\_feel, temp\_category

FROM CarSharing;

''')

# Querying the "temperature" table and print the first 5 rows to verify the new table has been created

print("First 5 rows of the 'temperature' table:")

cur.execute('SELECT \* FROM temperature LIMIT 5;')

rows = cur.fetchall()

for row in rows:

print(row)

# Dropping the "temp" and "temp\_feel" columns from the "CarSharing" table (SQLite does not support dropping columns directly so recreated the table without these column)

conn.execute('''

CREATE TABLE CarSharing\_new AS

SELECT id, timestamp, season, holiday, workingday, weather, humidity, windspeed, demand, temp\_category

FROM CarSharing;

''')

# Checking the modified "CarSharing" table and printing the first 5 rows to verify the column removals

print("\nFirst 5 rows of the modified 'CarSharing' table:")

cur.execute('SELECT \* FROM CarSharing LIMIT 5;')

rows = cur.fetchall()

for row in rows:

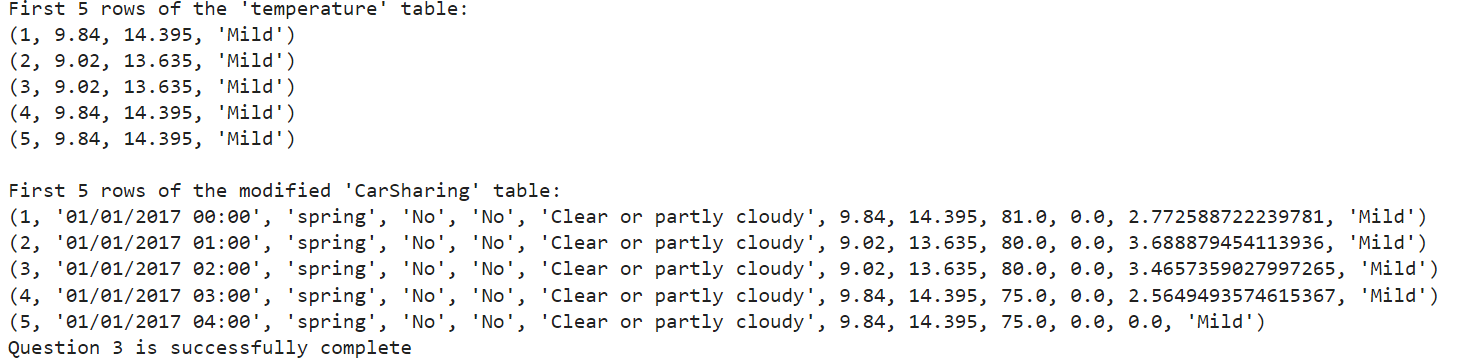
print(row)

conn.execute('DROP TABLE CarSharing;')

conn.execute('ALTER TABLE CarSharing\_new RENAME TO CarSharing;')

print ("Question 3 is successfully complete")

**Result 1.3**



**Code 1.4**

######QUESTION 4 - Find the distinct values of the weather column and assign a number to each value. Add

#another column named “weather\_code” to the table containing each row’s assigned

#weather code. #######

# Createing a cursor object using the connection

cur = conn.cursor()

# Finding distinct weather values as requested

cur.execute("SELECT DISTINCT weather FROM CarSharing;")

distinct\_weather = cur.fetchall()

# Assigning a number to each distinct weather value

weather\_code = {weather[0]: i for i, weather in enumerate(distinct\_weather)}

# Adding a new column for weather\_code

cur.execute("ALTER TABLE CarSharing ADD COLUMN weather\_code INTEGER;")

# Updating the weather\_code in the CarSharing table based on weather

for weather, code in weather\_code.items():

cur.execute("UPDATE CarSharing SET weather\_code = ? WHERE weather = ?", (code, weather))

# Checking the CarSharing table and printing the first 5 rows to verify the weather\_code column has been added and populated

print("\nFirst 5 rows of the 'CarSharing' table after adding weather\_code:")

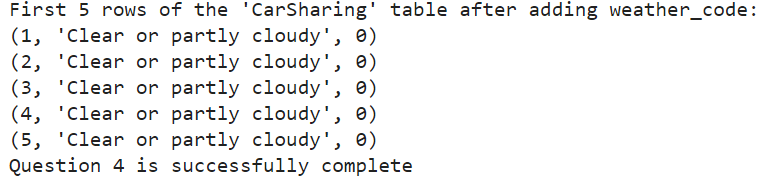
cur.execute("SELECT id, weather, weather\_code FROM CarSharing LIMIT 5;")

rows = cur.fetchall()

for row in rows:

print(row)

print ("Question 4 is successfully complete")

**Result 1.4**

**Code 1.5**

######QUESTION 5 - Create a table called “weather” and copy the columns “weather” and “weather\_code” to

#this table. Then drop the weather column from the CarSharing table. ######

# Creating the weather table

cur.execute('''

CREATE TABLE IF NOT EXISTS weather (

weather TEXT UNIQUE,

weather\_code INTEGER PRIMARY KEY

);

''')

# Putting both weather and weather\_code into the weather table

cur.execute('''

INSERT OR IGNORE INTO weather (weather, weather\_code)

SELECT DISTINCT weather, weather\_code FROM CarSharing;

''')

# Creating a new table that mirrors CarSharing without the weather column as requested

cur.execute('''CREATE TABLE CarSharing\_new (

id INTEGER PRIMARY KEY,

timestamp TEXT,

season INTEGER,

holiday INTEGER,

workingday INTEGER,

weather\_code INTEGER,

humidity REAL,

windspeed REAL,

demand INTEGER

);''')

# Copying data from CarSharing into CarSharing\_new

cur.execute('''INSERT INTO CarSharing\_new (id, timestamp, season, holiday, workingday, weather\_code, humidity, windspeed, demand)

SELECT id, timestamp, season, holiday, workingday, weather\_code, humidity, windspeed, demand FROM CarSharing;''')

# Dropping the original CarSharing table

cur.execute('DROP TABLE IF EXISTS CarSharing;')

# Renaming CarSharing\_new to CarSharing

cur.execute('ALTER TABLE CarSharing\_new RENAME TO CarSharing;')

# Verify the results by querying and checking the CarSharing table and the weather table

print("\nVerifying the CarSharing table:")

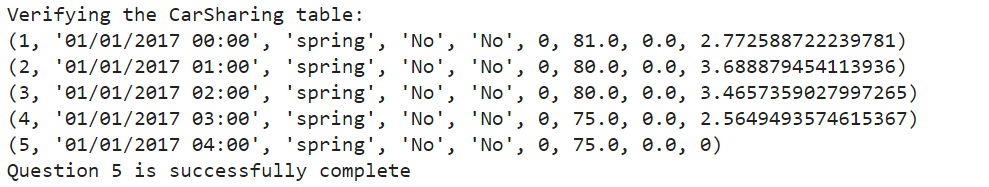
cur.execute("SELECT \* FROM CarSharing LIMIT 5;")

for row in cur.fetchall():

print(row)

print ("Question 5 is successfully complete")

**Result 1.5**



**Code 1.6**

######QUESTION 6 - Create a table called time with four columns containing each row’s timestamp, hour,

#weekday name, and month name (Hint: you can use the strftime() function for this

#purpose). ######

# Createing the "time" table with the desired columns

create\_time\_table\_sql = """

CREATE TABLE IF NOT EXISTS time (

timestamp TEXT,

hour INTEGER,

weekday TEXT,

month TEXT

);

"""

# Inserting data into the "time" table

insert\_into\_time\_sql = """

INSERT INTO time (timestamp, hour, weekday, month)

SELECT

timestamp as timestamp,

CAST(strftime('%H', timestamp) AS INTEGER) AS hour,

CASE strftime('%w', timestamp)

WHEN '0' THEN 'Sunday'

WHEN '1' THEN 'Monday'

WHEN '2' THEN 'Tuesday'

WHEN '3' THEN 'Wednesday'

WHEN '4' THEN 'Thursday'

WHEN '5' THEN 'Friday'

WHEN '6' THEN 'Saturday'

END AS weekday,

CASE strftime('%m', timestamp)

WHEN '01' THEN 'January'

WHEN '02' THEN 'February'

WHEN '03' THEN 'March'

WHEN '04' THEN 'April'

WHEN '05' THEN 'May'

WHEN '06' THEN 'June'

WHEN '07' THEN 'July'

WHEN '08' THEN 'August'

WHEN '09' THEN 'September'

WHEN '10' THEN 'October'

WHEN '11' THEN 'November'

WHEN '12' THEN 'December'

END AS month

FROM CarSharing;

"""

# Executing the SQL commands

cur.execute(create\_time\_table\_sql)

cur.execute(insert\_into\_time\_sql)

print("The 'time' table has successfully been created and populated.")

# Verifying the results by querying and checking (preview) the "time" table

print("\nVerifying the 'time' table:")

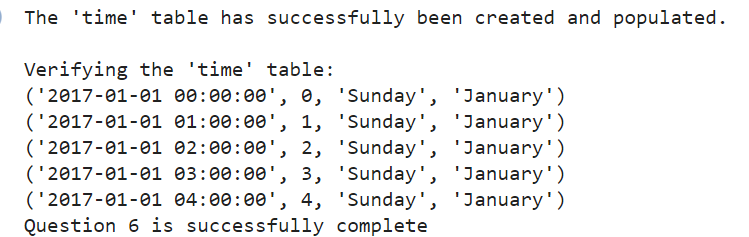
cur.execute("SELECT \* FROM time LIMIT 5;")

for row in cur.fetchall():

print(row)

print ("Question 6 is successfully complete")

**Result 1.6**



**Code 1.7**

######QUESTION 7 - Assume it’s the first day you have started working at this company and your boss Linda

#sends you an email as follows: “ Hello, welcome to the team. I hope you enjoy working at this company. Could you

#please give me a report containing the following information:

#A) Please tell me which date and time we had the highest demand rate in 2017.

#B) Give me a table containing the name of the weekday, month, and season in which we had the highest and lowest average demand rates throughout 2017.

#Please include the calculated average demand values as well.

#C) For the weekday selected in (b), please give me a table showing the average demand rate we had at different hours of that weekday throughout 2017.

#Please sort the results in descending order based on the average demand rates.

# D)Please tell me what the weather was like in 2017. Was it mostly cold, mild, or hot? Which weather condition (shown in the weather column) was the most

# prevalent in 2017? What was the average, highest, and lowest wind speed and humidity for each month in 2017? Please organise this information in two tables

#for the wind speed and humidity. Please also give me a table showing the average demand rate for each cold, mild, and hot weather in 2017 sorted in

#descending order based on their average demand rates. [4%]

#e) Give me another table showing the information requested in (d) for the month we had the highest average demand rate in 2017 so that I can compare it with

#other months.

# A

# Querying and checking to find the date and time with the highest demand rate in the year 2017

cur.execute('''

SELECT timestamp, demand

FROM CarSharing

WHERE strftime('%Y', timestamp) = '2017'

ORDER BY demand DESC

LIMIT 1;

''')

highest\_demand\_2017 = cur.fetchone()

print (highest\_demand\_2017)

if highest\_demand\_2017:

print(f" The highest demand in 2017 was at {highest\_demand\_2017[0]} with a demand of {highest\_demand\_2017[1]}")

else:

print("No data found for 2017.")

print ("7A is successfully complete")

#B

# The high average demand

cur.execute('''

SELECT

strftime('%w', timestamp) AS weekday,

strftime('%m', timestamp) AS month,

AVG(demand) AS avg\_demand

FROM CarSharing

WHERE strftime('%Y', timestamp) = '2017'

GROUP BY weekday, month

ORDER BY avg\_demand DESC

LIMIT 1;

''')

highest\_avg\_demand = cur.fetchone()

weekday\_names = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

month\_names = ["", "January", "February", "March", "April", "May", "June",

"July", "August", "September", "October", "November", "December"]

#The low average demand

cur.execute('''

SELECT

strftime('%w', timestamp) AS weekday,

strftime('%m', timestamp) AS month,

AVG(demand) AS avg\_demand

FROM CarSharing

WHERE strftime('%Y', timestamp) = '2017'

GROUP BY weekday, month

ORDER BY avg\_demand ASC

LIMIT 1;

''')

print ("7B is successfully complete")

#C

# Defining weekday and month names list

weekday\_names = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

month\_names = ["", "January", "February", "March", "April", "May", "June",

"July", "August", "September", "October", "November", "December"]

# Finding the highest average demand's weekday and month

cur.execute('''

SELECT

strftime('%w', timestamp) AS weekday,

strftime('%m', timestamp) AS month,

AVG(demand) AS avg\_demand

FROM CarSharing

WHERE strftime('%Y', timestamp) = '2017'

GROUP BY weekday, month

ORDER BY avg\_demand DESC

LIMIT 1;

''')

highest\_avg\_demand = cur.fetchone()

highest\_weekday = highest\_avg\_demand[0]

# Providing a table that displays the average demand rate for different hours on that day over the course of 2017

cur.execute(f'''

SELECT strftime('%H', timestamp) AS hour, AVG(demand) AS avg\_demand

FROM CarSharing

WHERE strftime('%Y', timestamp) = '2017' AND strftime('%w', timestamp) = '{highest\_weekday}'

GROUP BY hour

ORDER BY avg\_demand DESC;

''')

hourly\_demand\_2017 = cur.fetchall()

# Print the results

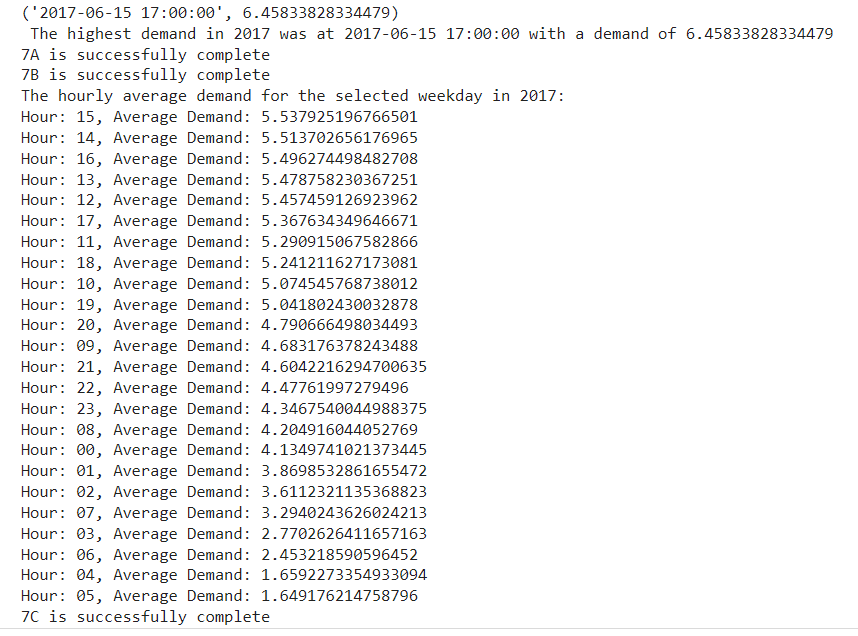
print(f"The hourly average demand for the selected weekday in 2017:")

for hour\_demand in hourly\_demand\_2017:

print(f"Hour: {hour\_demand[0]}, Average Demand: {hour\_demand[1]}")

print ("7C is successfully complete")

**Result1.7**



**Code 1.7b**

#7D

#Importing necessary libaries and modules

import sqlite3

from datetime import datetime

conn = sqlite3.connect('car\_sharing.db')

# Creating a cursor object using the connection

cur = conn.cursor()

# D1

# Createing a new table to analyze weather conditions alongside temperature categories Considering both the weather condition and temp\_category in the grouping

cur.execute('''

CREATE TABLE IF NOT EXISTS WeatherTempAnalysis AS

SELECT cb.weather AS WeatherCondition, c.temp\_category AS TemperatureCategory, COUNT(\*) AS count

FROM CarSharing\_backup AS cb

JOIN CarSharing AS c ON c.id = cb.id

WHERE strftime('%Y', c.timestamp) LIKE '2017%'

GROUP BY WeatherCondition, TemperatureCategory

ORDER BY count DESC

''')

# Retrieving and displaying the most common weather condition, considering both weather and temperature category

cur.execute("SELECT \* FROM WeatherTempAnalysis")

records = cur.fetchall()

print(f"\nAnalysis Result: \nWhen considering both the weather conditions and temperature categories,\n"

f"the most frequently occurring scenario is: {records[0]}")

print ("1st question in 7D has been answered")

# 7D2

# Createing a table to summarize weather conditions from the dataset, Considering only the weather condition in the grouping

cur.execute('''

CREATE TABLE IF NOT EXISTS WeatherSummary AS

SELECT cb.weather AS WeatherCondition, COUNT(\*) AS count

FROM CarSharing\_backup AS cb

JOIN CarSharing AS c ON c.id = cb.id

WHERE strftime('%Y', c.timestamp) LIKE '2017%'

GROUP BY WeatherCondition

ORDER BY count DESC

''')

# Retrieving and displaying the most common weather condition from the summary

cur.execute("SELECT \* FROM WeatherSummary")

records = cur.fetchall()

print(f"\nWeather Analysis Result: \nWhen focusing solely on the weather condition,\n"

f"the most frequently recorded weather condition is: {records[0]}")

print ("2nd question in 7D has been answered")

# 7D3.

# Creating a table to summarize average, highest, and lowest wind speed for each month in 2017

cur.execute("""

CREATE TABLE IF NOT EXISTS WindSpeedMonthlySummary AS

SELECT strftime('%m', timestamp) AS Month,

AVG(windspeed) AS AverageWindSpeed,

MAX(windspeed) AS MaxWindSpeed,

MIN(windspeed) AS MinWindSpeed

FROM CarSharing

WHERE strftime('%Y', timestamp) = '2017'

AND windspeed IS NOT NULL

GROUP BY Month

ORDER BY Month ASC

""")

# Retrieving and displaying the wind speed summary for each month

cur.execute("SELECT \* FROM WindSpeedMonthlySummary")

windSpeedSummary = cur.fetchall()

print("\n The Wind Speed Summary for Each Month in 2017:")

print("Month | Average Wind Speed | Maximum Wind Speed | Minimum Wind Speed")

for row in windSpeedSummary:

print(f"{row[0]} | {row[1]:.2f} km/h | {row[2]:.2f} km/h | {row[3]:.2f} km/h")

print ("3rd question in 7D has been answered")

# 7D4.

# Table for average, highest and lowest humidity for each month in 2017 and establishing a table to gather average, maximum, and minimum humidity values for each month of 2017

cur.execute("""

CREATE TABLE IF NOT EXISTS HumidityMonthlySummary AS

SELECT strftime('%m', timestamp) AS Month,

AVG(humidity) AS AverageHumidity,

MAX(humidity) AS MaxHumidity,

MIN(humidity) AS MinHumidity

FROM CarSharing

WHERE strftime('%Y', timestamp) = '2017'

AND humidity IS NOT NULL

GROUP BY Month

ORDER BY Month ASC

""")

# Retrieving the summarized humidity data for each month

cur.execute("SELECT \* FROM HumidityMonthlySummary")

humiditySummary = cur.fetchall()

print("\n The Humidity Summary for Each Month in 2017:")

print("Month | Average Humidity | Maximum Humidity | Minimum Humidity")

for row in humiditySummary:

month = int(row[0]) # Convert month from string to integer to remove any leading zeros

print(f"{month:02d} | {row[1]:.2f}% | {row[2]:.2f}% | {row[3]:.2f}%")

print ("4th question in 7D has been answered")

# 7D5.

# Create table showing the average demand rate for each cold, mild and hot weather in 2017

# sorted in descending order based on their average demand rates.

# And creating a table to analyze average demand rates across different temperature categories in 2017

cur.execute("""

CREATE TABLE IF NOT EXISTS TemperatureDemandSummary AS

SELECT temp\_category AS TemperatureCategory, AVG(demand) AS AverageDemand

FROM CarSharing

WHERE strftime('%Y', timestamp) = '2017'

GROUP BY TemperatureCategory

ORDER BY AverageDemand DESC

""")

# Retrieving and displaying the average demand rate for each temperature category

cur.execute("SELECT \* FROM TemperatureDemandSummary")

temperatureDemandSummary = cur.fetchall()

print("\nAverage Demand Rate by Temperature Category in 2017:")

print("Temperature Category | Average Demand Rate")

for row in temperatureDemandSummary:

print(f"{row[0]} | {row[1]:.2f}")

print ("5th question in 7D has been answered")

print ("7D is successfully complete")

#7E

# 7e.

# Create table showing the information in 7d for the month with the highest average demand rate

# in 2017 and compare it with other months

# 7ei.

# Createing the table to show the information summary for all the months in year 2017.

cur.execute("""

CREATE TABLE IF NOT EXISTS info\_summary\_demand2017 AS

SELECT c.temp\_category, cb.weather, cb.humidity, cb.windspeed, AVG(c.demand) AS avg\_demand,

CASE strftime('%m', c.timestamp)

WHEN '01' THEN 'January'

WHEN '02' THEN 'February'

WHEN '03' THEN 'March'

WHEN '04' THEN 'April'

WHEN '05' THEN 'May'

WHEN '06' THEN 'June'

WHEN '07' THEN 'July'

WHEN '08' THEN 'August'

WHEN '09' THEN 'September'

WHEN '10' THEN 'October'

WHEN '11' THEN 'November'

WHEN '12' THEN 'December'

END AS month

FROM CarSharing\_backup cb

JOIN CarSharing c

ON c.id = cb.id

WHERE strftime('%Y', c.timestamp) LIKE '2017%'

GROUP BY month

ORDER BY avg\_demand DESC

""")

# Fetching the month of the highest demand rate throughout 2017 and printing

highest\_month = datetime.strptime(highest\_demand\_2017[0], '%Y-%m-%d %H:%M:%S').strftime('%B')

print("\nQ7e: Month with highest demand rate in 2017 ", highest\_month)

# Fetching the table to show the information summary for all the months in year 2017 for comparison.

cur.execute("SELECT \* FROM info\_summary\_demand2017 WHERE month = ?", (highest\_month,))

highest\_month\_summary\_demand2017 = cur.fetchall()

print("\nQ7e: The Information summary for the month with highest demand in year 2017.")

for row in highest\_month\_summary\_demand2017:

print(row)

# Fetching the table to show the information summary for all the months in year 2017 for comparison.

cur.execute("SELECT \* FROM info\_summary\_demand2017")

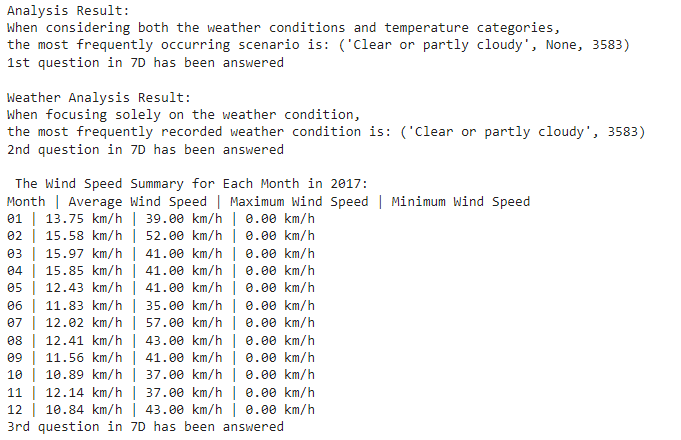
info\_summary\_demand2017 = cur.fetchall()

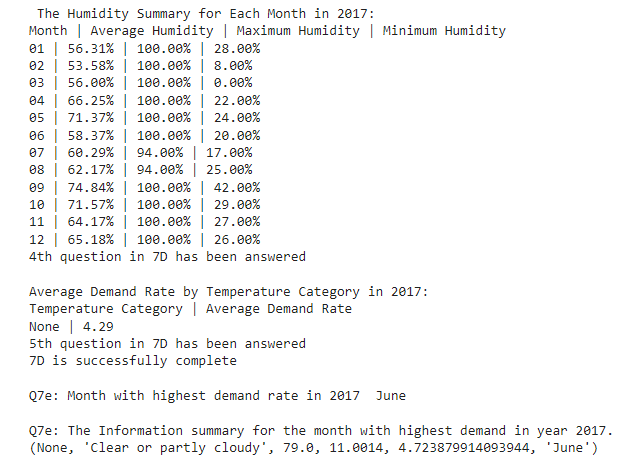
print("\nQ7e: The Information summary for all the months in year 2017.")

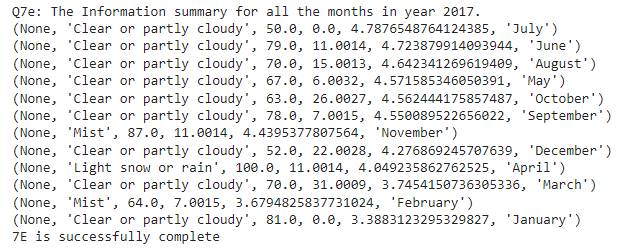
for row in info\_summary\_demand2017:

print(row)

**Result 1.7b**







**DATA MANAGEMENT TASK ENDS**

**------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**DATA ANALYTICS TASK**

**Code 2.1**

############################# TASK 2 - DATA ANALYTICS ###########################

#### QUESTION 1 - Import the CarSharing table into a CSV file and preprocess it with python. You need to

#drop duplicate rows and deal with null values using appropriate methods ####

# Importing pandas and csv

import pandas as pd

import csv

# Loading the CarSharing CSV file into a DataFrame

car\_sharing\_df = pd.read\_csv(r'CarSharing.csv')

# Displaying the first few rows of the Carsharing csv for a quick preview of the table

print(car\_sharing\_df.head())

# Preprocessing a summary before cleaning

print(f"Initial DataFrame contains: {car\_sharing\_df.shape[0]} rows, {car\_sharing\_df.shape[1]} columns")

# Droping any duplicate rows

car\_sharing\_df.drop\_duplicates(inplace=True)

print(f"After dropping any duplicates: {car\_sharing\_df.shape[0]} rows")

# Dealing with any possible null values and filling in any missing numerical values with the mean of their respective columns

numeric\_cols = car\_sharing\_df.select\_dtypes(include=['float64', 'int64']).columns

car\_sharing\_df[numeric\_cols] = car\_sharing\_df[numeric\_cols].fillna(car\_sharing\_df[numeric\_cols].mean())

# Filling in any missing categorical columns with the most frequent value (AKA mode) of their respective columns For categorical columns: fill missing values with the mode (most frequent value) of their respective columns

categorical\_cols = car\_sharing\_df.select\_dtypes(include=['object']).columns

for col in categorical\_cols:

car\_sharing\_df[col] = car\_sharing\_df[col].fillna(car\_sharing\_df[col].mode()[0])

# Checking one last time for any remaining null values

print("\nFinal Null Value Check:\n", car\_sharing\_df.isnull().sum())

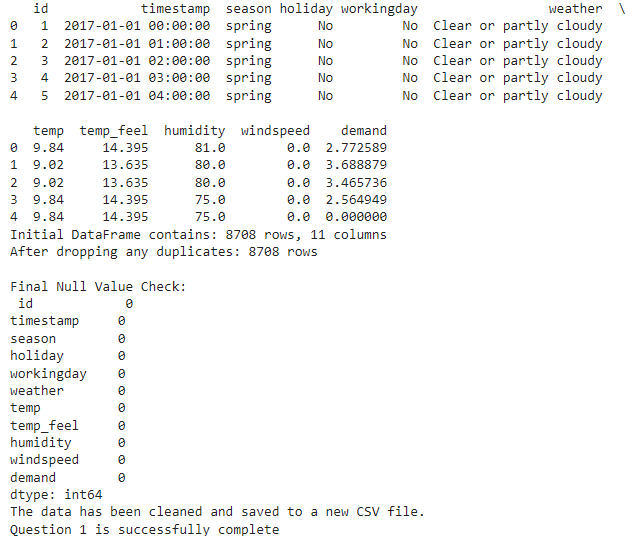
# Saving the cleaned dataset

car\_sharing\_df.to\_csv(r'CarSharingCleaned.csv')

print("The data has been cleaned and saved to a new CSV file.")

print("Question 1 is successfully complete")

**Result 2.1**



**Code 2.2**

#### QUESTION 2 - Using appropriate hypothesis testing, determine if there is a significant relationship

#between each column (except the timestamp column) and the demand rate. Report the

#tests’ results####

# Loading necessary Libaries and modules

from scipy.stats import pearsonr, f\_oneway

from statsmodels.formula.api import ols

# Loading the processed and cleaned dataset

car\_sharing\_df = pd.read\_csv(r'CarSharingCleaned.csv')

# Analyzing Correlations among numerical columns with Pearson's Method

numerical\_columns = ["temp", "temp\_feel", "humidity", "windspeed"]

print("###### Numerical Columns Analysis with Pearson's Method ######")

for col in numerical\_columns:

correlation, p\_value = pearsonr(car\_sharing\_df[col], car\_sharing\_df["demand"])

print(f"{col}: Pearson correlation coefficient = {correlation}, p-value = {p\_value}")

# Analyzing Correlations among categorical columns with ANOVA

categorical\_columns = ["season", "holiday", "workingday", "weather"]

print("\n###### Categorical Columns Analysis with ANOVA ######")

for col in categorical\_columns:

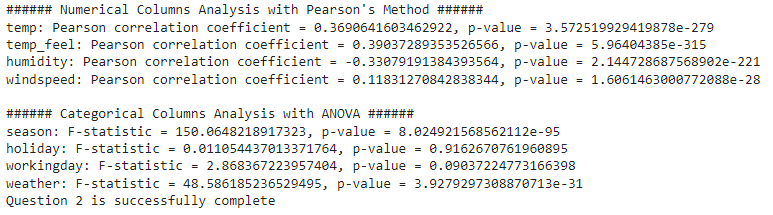
groups = car\_sharing\_df.groupby(col)["demand"].apply(list)

f\_stat, p\_value = f\_oneway(\*groups)

print(f"{col}: F-statistic = {f\_stat}, p-value = {p\_value}")

print("Question 2 is successfully complete")

**Result 2.2**



**Code 2.3**

#### QUESTION 3 - Please describe if you see any seasonal or cyclic pattern in the temp, humidity, windspeed,

#or demand data in 2017. Describe your answers. ####

# Loading necessary Libaries and modules

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal\_decompose

# Loading the dataset, Choosing a specific datetime format for the 'timestamp' column

car\_sharing\_df = pd.read\_csv( r'CarSharingCleaned.csv',

parse\_dates=['timestamp'],

# date\_format='%d-%m-%Y %H:%M'

dayfirst=True

)

# Setting the 'timestamp' column as the DataFrame's index

car\_sharing\_df.set\_index('timestamp', inplace=True)

# Filtering for data within the year of 2017

df\_2017 = car\_sharing\_df.loc[car\_sharing\_df.index.year == 2017]

# Plotting and seasonal decomposition of each specified column

columns = ['temp', 'humidity', 'windspeed', 'demand']

for column in columns:

# # Seasonal decomposition

# # Assuming data is hourly, the frequency is 24 (hours in a day)

# # The ⁠ model='additive' ⁠ should be selected based on the nature of the data

data = df\_2017[column].dropna()

decomposition = seasonal\_decompose(data, model='additive', period=24)

# decomposition.plot()

# # plt.show(block=False)

# Plot the decomposition

plt.figure(figsize=(12, 10))

plt.suptitle(column.capitalize() + ' in 2017') # Add a title for the entire figure

# Top left subplot

plt.subplot(2, 2, 1)

plt.plot(data, label='Original')

plt.legend(loc='upper left')

# Top right subplot

plt.subplot(2, 2, 2)

plt.plot(decomposition.trend, label='Trend')

plt.legend(loc='upper right')

# Bottom left subplot

plt.subplot(2, 2, 3)

plt.plot(decomposition.seasonal, label='Seasonal')

plt.legend(loc='lower left')

# Bottom right subplot

plt.subplot(2, 2, 4)

plt.plot(decomposition.resid, label='Residual')

plt.legend(loc='lower right')

plt.tight\_layout()

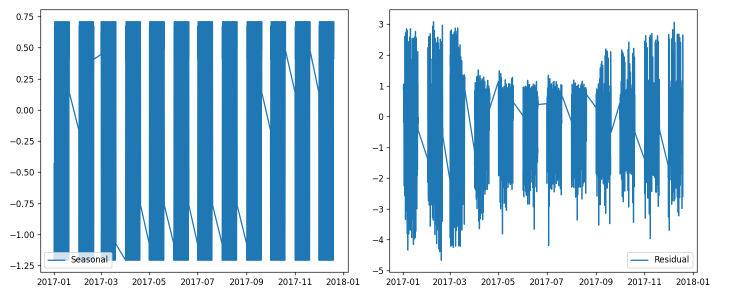
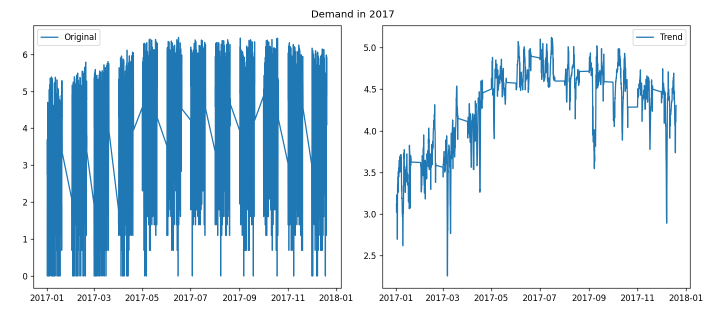
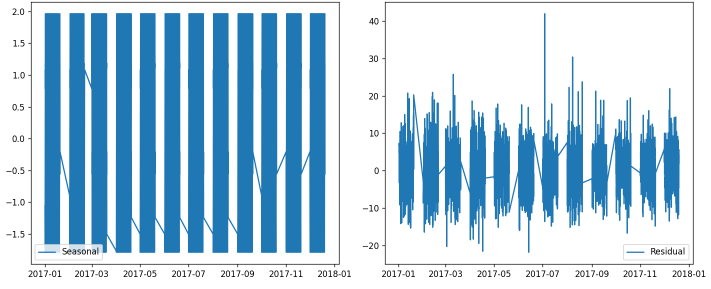
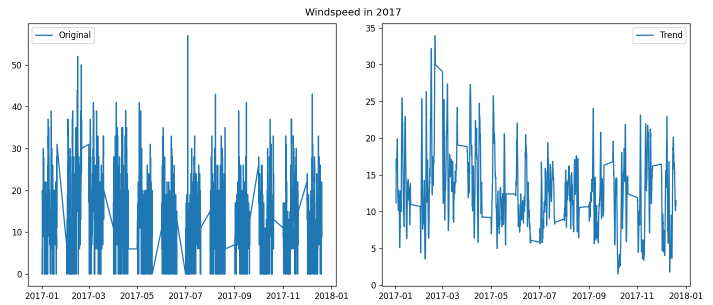
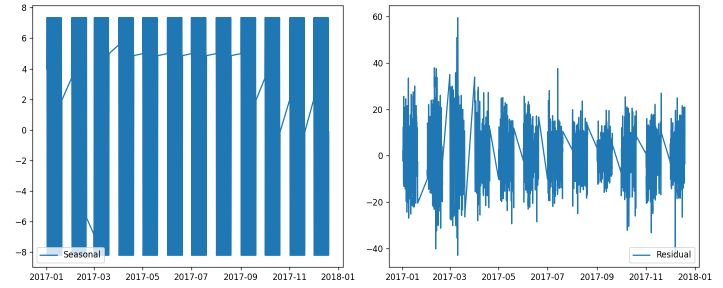
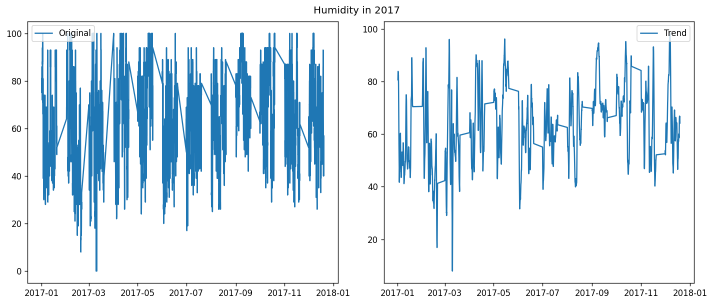
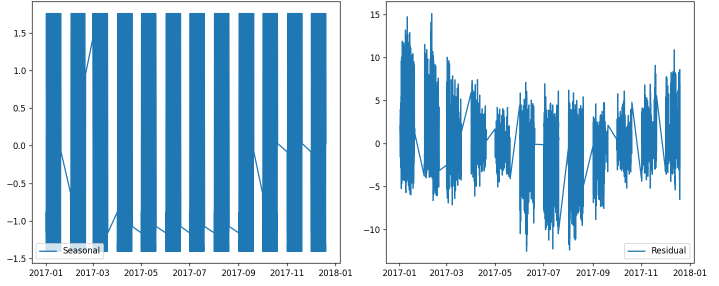
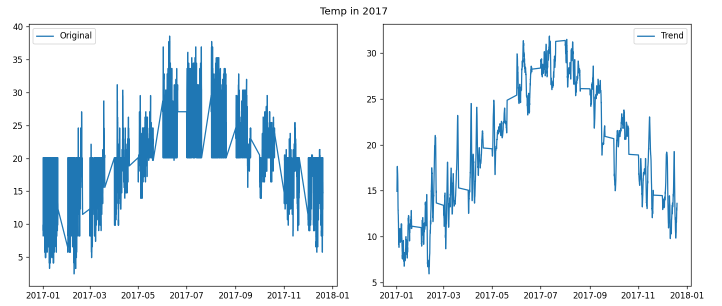
plt.show()

# Pause briefly to display the plot

plt.pause(0.01)

print ("Question 3 is successfully complete")

**Result 2.3**



**Code 2.4**

#### QUESTION 4 - Use an ARIMA model to predict the weekly average demand rate. Consider 30 percent of

#data for testing. ####

# Loading necessary Libaries and modules

from statsmodels.tsa.stattools import adfuller

import matplotlib.pyplot as plt

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from statsmodels.tsa.arima.model import ARIMA

from sklearn.metrics import mean\_squared\_error

import numpy as np

# Ensureing the 'timestamp' column is in the correct datetime format

car\_sharing\_df = pd.read\_csv(r'CarSharingCleaned.csv', parse\_dates=['timestamp'])

car\_sharing\_df.set\_index('timestamp', inplace=True)

# Selecting the 'demand' series for stationarity check and modeling

series = car\_sharing\_df['demand']

# Checking if the time series is stationary

result = adfuller(series.dropna())

print('ADF Statistic: %f' % result[0])

print('p-value: %f' % result[1])

# Applying differencing to achieve stationarity in the Time Series and Determining the 'd' Parameter for the ARIMA Model

result = adfuller(series.diff().dropna())

print('After applying 1st Differencing - ADF Statistic: %f' % result[0])

print('p-value: %f' % result[1])

# Since your series is likely to become stationary after differencing, `d=1` is a good starting point.

df = pd.DataFrame(series.values, columns=['value'])

plt.rcParams.update({'figure.figsize':(9,3), 'figure.dpi':120})

#Partial Autocorrelation Function (PACF) Plot

fig, axes = plt.subplots(1, 2, sharex=True)

axes[0].plot(df.diff().value)

axes[0].set\_title('1st Order Differencing')

plot\_pacf(df.diff().value.dropna(), ax=axes[1])

plt.show()

#Autocorrelation Function (ACF) Plot

fig, axes = plt.subplots(1, 2, sharex=True)

axes[0].plot(df.diff().value)

axes[0].set\_title('1st Order Differencing')

plot\_acf(df.diff().value.dropna(), ax=axes[1])

plt.show()

# Splitting the Data into Training and Testing Sets

X = series.values

size = int(len(X) \* 0.7)

train, test = X[0:size], X[size:len(X)]

#Fitting the ARIMA Model

model = ARIMA(train, order=(1,1,1)) # Use determined p, d, q values

model\_fit = model.fit()

print(model\_fit.summary())

# Plotting and visualising residual errors

residuals = pd.DataFrame(model\_fit.resid)

fig, ax = plt.subplots(1,2)

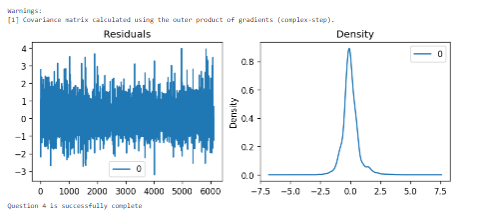
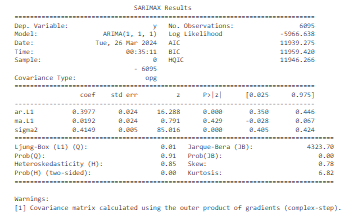
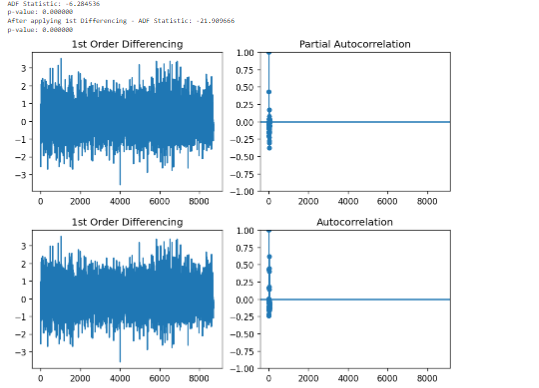
residuals.plot(title="Residuals", ax=ax[0])

residuals.plot(kind='kde', title='Density', ax=ax[1])

plt.show()

print("Question 4 is successfully complete")

**Result 2.4**



**Code 2.5**

#### QUESTION 5 - Use a random forest regressor and a deep neural network to predict the demand rate and

#report the minimum square error for each model. Which one is working better? Why? ####

## Loading necessary Libaries and modules

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Loading the processed and cleaned dataset

df = pd.read\_csv(r'CarSharingCleaned.csv')

# Encode categorical values for training

df = pd.get\_dummies(df, columns=['season', 'weather','holiday','workingday'])

# Preparing the data for modelling

X = df.drop(columns=['demand', 'timestamp'])

y = df['demand']

# Splitting the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Using Random Forest Regressor to predict the demand rate and reporting the minimum square error

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

rf\_predictions = rf\_model.predict(X\_test)

rf\_mse = mean\_squared\_error(y\_test, rf\_predictions)

print("Random Forest MSE:", rf\_mse)

# Using Deep Neural Network to predict the demand rate and reporting the minimum square error

dnn\_model = Sequential()

dnn\_model.add(Dense(64, activation='relu', input\_dim=X\_train.shape[1]))

dnn\_model.add(Dense(32, activation='relu'))

dnn\_model.add(Dense(1))

dnn\_model.compile(optimizer='adam', loss='mean\_squared\_error')

dnn\_model.fit(X\_train, y\_train, epochs=100, batch\_size=32, verbose=0)

dnn\_predictions = dnn\_model.predict(X\_test)

dnn\_mse = mean\_squared\_error(y\_test, dnn\_predictions.flatten())

print("DNN MSE:", dnn\_mse)

# Comparing both of the previously used models

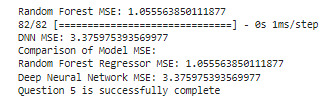
print("Comparison of Model MSE:")

print(f"Random Forest Regressor MSE: {rf\_mse}")

print(f"Deep Neural Network MSE: {dnn\_mse}")

print("Question 5 is successfully complete")

**Result 2.5**



**Code 2.6**

#### QUESTION 6 - Categorize the demand rate into the following two groups: demand rates greater than the

#average demand rate and demand rates less than the average demand rate. Use labels 1

#and 2 for the first and the second groups, respectively. Now, use three different classifiers

#to predict the demand rates’ labels and report the accuracy of all models. Use 30 percent

#of data for testing. ####

## Loading necessary Libaries and modules

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# Loading the processed and cleaned dataset

df = pd.read\_csv(r'CarSharingCleaned.csv')

# Encode categorical values for modelling

df = pd.get\_dummies(df, columns=['season', 'weather','holiday','workingday'])

# Calculating the average demand rate and categorizing the demand rate into the requested groups

average\_demand = df['demand'].mean()

df['demand\_group'] = df['demand'].apply(lambda x: 1 if x > average\_demand else 2)

# Preparing the data for modeling

X = df.drop(columns=['demand', 'demand\_group', 'timestamp']) # Assuming 'timestamp' is excluded

y = df['demand\_group']

# Splitting the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Standardizing the features

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Initializing classifiers

classifiers = {

"Random Forest Classifier": RandomForestClassifier(n\_estimators=100, random\_state=42),

"Logistic Regression": LogisticRegression(random\_state=42),

"SVM": SVC(random\_state=42)

}

# Training and reflecting on each classifier

for name, clf in classifiers.items():

clf.fit(X\_train\_scaled, y\_train)

predictions = clf.predict(X\_test\_scaled)

accuracy = accuracy\_score(y\_test, predictions)

print(f"{name} Accuracy is: {accuracy:.4f}")

print("Question 6 is successfully complete")

**Result 2.6**



**Code 2.7**

# 7. To determine which k value gives the most uniform clusters when clustering

# the temp data in 2017 using 2 different methods, you can follow these steps:

# Import the necessary libraries for clustering

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import numpy as np

df = pd.read\_csv(r'CarSharingCleaned.csv', parse\_dates=['timestamp'])

# Filter the dataframe for the year 2017 and extract 'temp' data

df\_2017 = df[df['timestamp'].dt.year == 2017]['temp']

# Standardize the temp data

scaler = StandardScaler()

X = scaler.fit\_transform(df\_2017.values.reshape(-1, 1))

# Define the range of k values to evaluate for KMeans

k\_values = [2, 3, 4, 12]

print("Evaluating KMeans for different k values:")

for k in k\_values:

# Apply KMeans clustering

model\_kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)

labels\_kmeans = model\_kmeans.fit\_predict(X)

# Calculate the number of samples in each cluster

cluster\_counts = [list(labels\_kmeans).count(i) for i in range(k)]

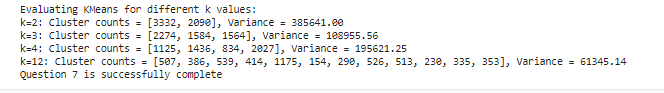
# Determine uniformity by the variance of the cluster sizes

variance = np.var(cluster\_counts)

print(f"k={k}: Cluster counts = {cluster\_counts}, Variance = {variance:.2f}")

print("Question 7 is successfully complete")

**Results 2.7**



**DATA ANALYTICS TASK ENDS**

**------------------------------------------------------------------------------------------------------------------------------------------------------------------------**