LONDON METROPOLITAN UNIVERSITY

Islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CC5067NI-Smart Data Discovery

Assessment Weightage & Type

60% Individual Coursework

Year and Semester

2022-23 Spring

Student Name: Himanshu Yadav

London Met ID: 21049513

College ID: NP01CP4A210166

Assignment Due Date: Thursday, May 4, 2023

Assignment Submission Date: Wednesday, May 3, 2023

Word Count: 2579

I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

Table of Contents

Table of Tables

Table of Tables

1. Data Understanding

The data set contains information related to the sales analysis of the ABC company in 2019. It includes several attributes such as order ID, product, quantity ordered, price per item, order date, and purchase address.

The Order ID attribute represents a unique identifier for each order placed by the customers of the ABC company. The product attribute provides information about the type of product that was ordered. The quantity ordered attribute indicates the quantity of each product that was ordered by the customers. The Price Each attribute represents the unit price of each product ordered. The Order Date attribute provides information about the date on which the order was placed. The Purchase Address attribute represents the address of the customer who placed the order.

This data set can be used for various purposes, such as analyzing the sales trends of the ABC company during the year 2019, identifying the top-selling products, and understanding customer behavior and preferences. By analyzing this data set, the company can make informed decisions regarding its inventory management, pricing strategies, and marketing campaigns to increase sales and revenue.

S.No.	Column Name	Description	Data Type
1.	Order ID	Unique identifier for each order placed by customers	Float64
2.	Product	Type of product that was ordered	Object
3.	Quantity Ordered	Quantity of each product that was ordered by customers	Float64
4.	Price Each	Unit price of each product ordered	Float64
5.	Order Date	Date on which the order was placed	Object
6.	Purchase Address	Address of the customer who placed the order	Object

Table 1: Table with descriptions and data type of each column of the data set

2. Data Preparation

A. Write a python program to merge data from each month into one CSV and read in updated dataframe.

```
import os
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import calendar
```

Figure 1: While importing libraries

A. Write a python program to merge data from each month into one CSV and read in updated dataframe.

```
master_df = pd.DataFrame()
data_frames = []
for file in os.listdir(os.getcwd()):
    if file.endswith('.csv'):
        data_frames.append(pd.read_csv(file))
master_df = pd.concat(data_frames)
```

Figure 2: Merging data set into one CSV and reading it part 1

```
master_df.to_csv('Sales_All_2019.csv',index=False)
```

```
df=pd.read_csv('Sales_All_2019.csv')
df
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558.0	USB-C Charging Cable	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559.0	Bose SoundSport Headphones	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215
3	176560.0	Google Phone	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560.0	Wired Headphones	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
...	...	...	...	...	...	...
186845	259353.0	AAA Batteries (4-pack)	3.0	2.99	9/17/2019 20:56	840 Highland St, Los Angeles, CA 90001
186846	259354.0	iPhone	1.0	700.00	9/1/2019 16:00	216 Dogwood St, San Francisco, CA 94016
186847	259355.0	iPhone	1.0	700.00	9/23/2019 7:39	220 12th St, San Francisco, CA 94016
186848	259356.0	34in Ultrawide Monitor	1.0	379.99	9/19/2019 17:30	511 Forest St, San Francisco, CA 94016
186849	259357.0	USB-C Charging Cable	1.0	11.95	9/30/2019 0:18	250 Meadow St, San Francisco, CA 94016

186850 rows × 6 columns

Figure 3: Merging data set into one CSV and reading it part 2

This code imports the required libraries, including os for managing file directories, pandas for data manipulation and analysis, matplotlib for data visualisation, seaborn for statistical data visualisation, and calendar for working with dates and calendars.

The code creates an empty master_df Pandas DataFrame and an empty data_frames list. The programme then iterates through each file in the current

working directory, selecting only those with the.csv extension. For each of these selected files, a Pandas DataFrame is created and added to the data_frames list. The code then concatenates and stores the result in the master_df DataFrame. The combined DataFrame is then written to a new.csv file named 'Sales_All_2019.csv'.

The code then loads the 'Sales_All_2019.csv' file into a new Pandas DataFrame named df and outputs its contents.

B. Write a python program to remove the NaN missing values from updated dataframe.

B. Write a python program to remove the NaN missing values from updated dataframe.

```
# NaN stands for Not A Number and is one of the common ways to represent the missing value in the data.
# It is a special floating-point value and cannot be converted to any other type than float.
# NaN value is one of the major problems in Data Analysis.
# It is very essential to deal with NaN in order to get the desired results.
```

```
df.isnull().sum()
```

```
Order ID            900
Product             900
Quantity Ordered    900
Price Each          900
Order Date          900
Purchase Address    900
dtype: int64
```

```
df.head()
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558.0	USB-C Charging Cable	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559.0	Bose SoundSport Headphones	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215
3	176560.0	Google Phone	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560.0	Wired Headphones	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001

Figure 4: Removing NaN missing values part 1

```
sns.heatmap(df.isnull())
plt.title('Heatmap of the Data Frame \n before removing NaN missing values')
plt.show()
```

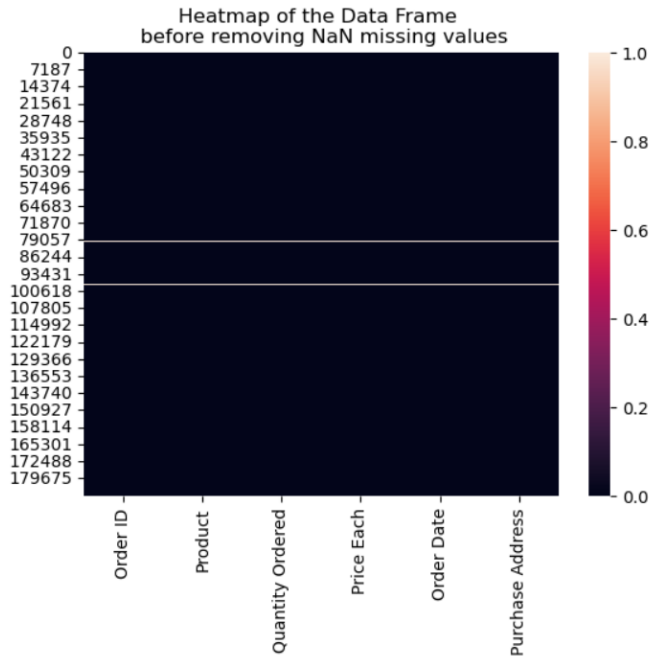


Figure 5: Removing NaN missing values part 2

```
df.dropna(axis=0,how='all',inplace=True)
```

```
df.isnull().sum()
```

```
Order ID            0
Product             0
Quantity Ordered    0
Price Each          0
Order Date          0
Purchase Address    0
dtype: int64
```

```
df.head()
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558.0	USB-C Charging Cable	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001
2	176559.0	Bose SoundSport Headphones	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215
3	176560.0	Google Phone	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560.0	Wired Headphones	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
5	176561.0	Wired Headphones	1.0	11.99	4/30/2019 9:27	333 8th St, Los Angeles, CA 90001

Figure 6: Removing NaN missing values part 3

```
sns.heatmap(df.isnull())
plt.title('Heatmap of the Data Frame \n after removing NaN missing values')
plt.show()
```

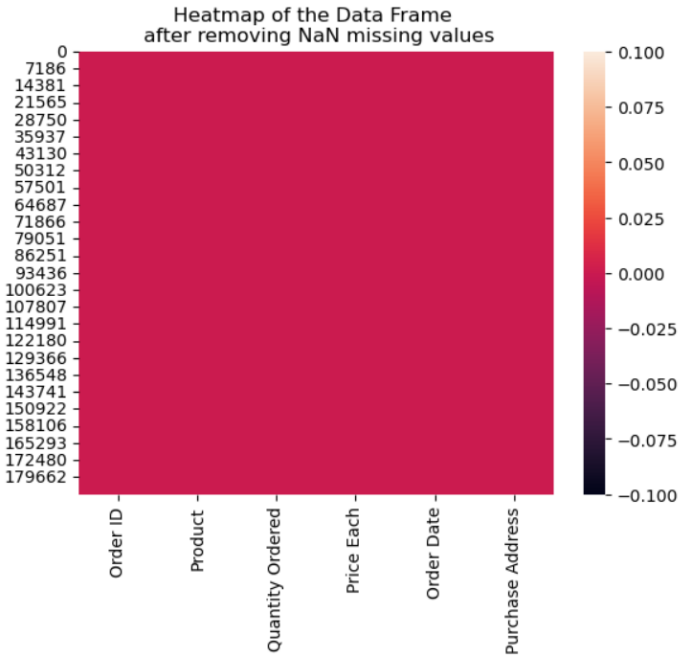


Figure 7: Removing NaN missing values part 4

```
df.reset_index(drop=True,inplace=True)
```

```
df.head()
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558.0	USB-C Charging Cable	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001
1	176559.0	Bose SoundSport Headphones	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215
2	176560.0	Google Phone	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
3	176560.0	Wired Headphones	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
4	176561.0	Wired Headphones	1.0	11.99	4/30/2019 9:27	333 8th St, Los Angeles, CA 90001

Figure 8: Removing NaN missing values part 5

df.isnull() is used to determine whether the DataFrame df contains any missing values.sum() returns the number of missing values for each column. The missing values in the DataFrame are then visually represented using the sns.heatmap(df.isnull()) function. Then, it removes rows with all missing values using df.dropna(axis=0,how='all',inplace=True) and generates a new heatmap to reflect the modifications. After removing rows with missing values, the DataFrame index is reset using df.reset_index(drop=True,inplace=True) to ensure that the index is sequential.

This Python code checks for and handles missing values in a DataFrame using the pandas and seaborn libraries. It ensures that the data are accurate and suitable for analysis.

C. Write a python program to convert Quantity Ordered and Price Each to numeric.

C. Write a python program to convert Quantity Ordered and Price Each to numeric

```
df.columns
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address'],
      dtype='object')
```

```
# The exact numeric types are INTEGER , BIGINT , DECIMAL , NUMERIC , NUMBER , and MONEY.
# Approximate numeric types, values where the precision needs to be preserved and the scale can be floating.
# The approximate numeric types are DOUBLE PRECISION , FLOAT , and REAL.
```

```
# Convertig the data type of 'Quantity Order' from 'float64' to 'int64'
```

```
df['Quantity Ordered'].dtypes
```

```
dtype('float64')
```

```
df['Quantity Ordered'] = df['Quantity Ordered'].astype('int64')
```

```
df['Quantity Ordered'].dtypes
```

```
dtype('int64')
```

```
# Convertig the data type of 'Price Each' from 'float64' to 'int64'
```

```
df['Price Each'].dtypes
```

```
dtype('float64')
```

```
df['Price Each'] = df['Price Each'].astype('int64')
```

```
df['Price Each'].dtypes
```

```
dtype('int64')
```

Figure 9: Converting Quantity Ordered and Price Each to numeric

The preceding code initially outputs the column names of a given DataFrame, df. Using the dtypes attribute, it verifies the data type of the Quantity Ordered column, which returns object. The astype() method is then used to convert this column to the int64 data type so that mathematical operations can be performed on it. Price Each is also converted to int64 using the same method. The dtypes attribute is utilised once more to confirm that both columns have been converted to the desired data type.

D. Create a new column named Month from Ordered Date of updated dataframe and convert it to integer as data type.

D. Create a new column named Month from Ordered Date of updated dataframe and convert it to integer as data type.

```
df['Order Date'].dtypes
```

```
dtype('O')
```

```
df['Order Date'] = df['Order Date'].apply(pd.to_datetime)
```

```
df['Order Date'].dtypes
```

```
dtype('<M8[ns]')
```

```
# dtype('<M8[ns]') is the new data type of a 'Order Date' column in a pandas DataFrame.
# The < indicates little-endian byte order, the M8 indicates that it is a datetime type with a precision of 8 bytes,
# and the [ns] indicates that the precision is in nanoseconds.
# In simpler terms, dtype('<M8[ns]') represents a datetime type with nanosecond precision.
# This data type is used in pandas to represent datetime objects with nanosecond precision.
```

```
df.head()
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558.0	USB-C Charging Cable	2	11	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001
1	176559.0	Bose SoundSport Headphones	1	99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215
2	176560.0	Google Phone	1	600	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001
3	176560.0	Wired Headphones	1	11	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001
4	176561.0	Wired Headphones	1	11	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001

Figure 10: Creating a new column 'Month' part 1

```
df
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month
0	176558.0	USB-C Charging Cable	2	11	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4
1	176559.0	Bose SoundSport Headphones	1	99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4
2	176560.0	Google Phone	1	600	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4
3	176560.0	Wired Headphones	1	11	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4
4	176561.0	Wired Headphones	1	11	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4
...	...	...	...	...	...	...	...
185945	259353.0	AAA Batteries (4-pack)	3	2	2019-09-17 20:56:00	840 Highland St, Los Angeles, CA 90001	9
185946	259354.0	iPhone	1	700	2019-09-01 16:00:00	216 Dogwood St, San Francisco, CA 94016	9
185947	259355.0	iPhone	1	700	2019-09-23 07:39:00	220 12th St, San Francisco, CA 94016	9
185948	259356.0	34in Ultrawide Monitor	1	379	2019-09-19 17:30:00	511 Forest St, San Francisco, CA 94016	9
185949	259357.0	USB-C Charging Cable	1	11	2019-09-30 00:18:00	250 Meadow St, San Francisco, CA 94016	9

185950 rows × 7 columns

```
df['Month'].dtypes
```

```
dtype('int64')
```

Figure 11: Creating a new column 'Month' part 2

This code begins by validating the data type of the 'Order Date' column in the DataFrame 'df'. We then convert the data type of the 'Order Date' column to datetime using the 'apply' function of Pandas. After this, the data type of the 'Order Date' column is double-checked to ensure that it has been converted to datetime. Then, we construct a new column named 'Month' by extracting the month component from

the 'Order Date' column using the 'dt.month' function of Pandas. Finally, we examine the data type of the 'Month' column in the 'df' DataFrame.

E. Create a new column named City from Purchase Address based on the value in updated dataframe.

E. Create a new column named City from Purchase Address based on the value in updated dataframe.

```
1 df['Purchase Address'].dtypes
```

```
dtype('O')
```

```
1 df['City'] = df['Purchase Address'].str.split(',').str[1]
```

```
1 df
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City
0	176558.0	USB-C Charging Cable	2	11	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4	Dallas
1	176559.0	Bose SoundSport Headphones	1	99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4	Boston
2	176560.0	Google Phone	1	600	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4	Los Angeles
3	176560.0	Wired Headphones	1	11	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4	Los Angeles
4	176561.0	Wired Headphones	1	11	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4	Los Angeles
...	...	...	...	...	...	...	...	...
185945	259353.0	AAA Batteries (4-pack)	3	2	2019-09-17 20:56:00	840 Highland St, Los Angeles, CA 90001	9	Los Angeles
185946	259354.0	iPhone	1	700	2019-09-01 16:00:00	216 Dogwood St, San Francisco, CA 94016	9	San Francisco
185947	259355.0	iPhone	1	700	2019-09-23 07:39:00	220 12th St, San Francisco, CA 94016	9	San Francisco
185948	259356.0	34in Ultrawide Monitor	1	379	2019-09-19 17:30:00	511 Forest St, San Francisco, CA 94016	9	San Francisco
185949	259357.0	USB-C Charging Cable	1	11	2019-09-30 00:18:00	250 Meadow St, San Francisco, CA 94016	9	San Francisco

185950 rows × 8 columns

Figure 12: Creating a new column 'City'

The code constructs a new column named 'City' in the dataframe 'df' by separating the values in the 'Purchase Address' column with a comma and extracting the second element. This is accomplished by utilising the'str.split()' method of the Pandas series, which divides strings according to the specified separator and returns a list. The list is indexed with [1] to retrieve the second element, which is the city name. The city name is then allocated to the dataframe's new 'City' column. This is beneficial for city-based sales data analysis because it enables us to group and compare sales data for various cities.

3. Data Analysis

A. Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of any chosen variable.

A. Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of any chosen variable.

```
df.columns
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month', 'City'],
      dtype='object')
```

```
Summary = df['Price Each'].describe()
```

```
# Skewness and kurtosis are statistical measures that describe the shape of a probability distribution.
# Skewness measures the degree of asymmetry of the distribution, while kurtosis measures the degree of peakedness.
# These measures are important because they provide information about the tail behavior and the peak of the distribution,
# and can help identify non-normal distributions and determine whether a given dataset requires a specific type of
# statistical analysis or transformation.
```

```
Summary['skewness'] = df['Price Each'].skew()
```

```
Summary['kurtosis'] = df['Price Each'].kurtosis()
```

```
Summary
```

```
count       185950.000000
mean           183.653014
std            332.941600
min              2.000000
25%             11.000000
50%             14.000000
75%            150.000000
max           1700.000000
skewness         2.870662
kurtosis         9.086667
Name: Price Each, dtype: float64
```

Figure 13: Statistics of sum, mean, standard deviation, skewness, and kurtosis of 'Price Each' variable.

The provided Python code first retrieves the column labels/names of a DataFrame df. Then, it creates a new Pandas Series called Summary by calling the describe() method on the 'Price Each' column of the DataFrame df. This method generates a summary of the central tendency, dispersion, and shape of the distribution of the data in the column. Next, the code adds two new items to the Summary Series, called 'skewness' and 'kurtosis', which represent the skewness and kurtosis of the 'Price Each' column, respectively. Skewness is a measure of the asymmetry of the distribution of the data around its mean, while kurtosis is a measure of the degree of peakedness or flatness of the distribution of the data. Finally, the code returns the Summary Series, which contains a summary of the 'Price Each' column along with its skewness and kurtosis values. This code can be useful for getting a quick overview of the distribution and shape of the 'Price Each' variable in the DataFrame.

B. Write a Python program to calculate and show correlation of all variables.

B. Write a Python program to calculate and show correlation of all variables.

```
1 df.corr()
```

	Order ID	Quantity Ordered	Price Each	Month
Order ID	1.000000	0.000702	-0.002861	0.993063
Quantity Ordered	0.000702	1.000000	-0.148335	0.000791
Price Each	-0.002861	-0.148335	1.000000	-0.003379
Month	0.993063	0.000791	-0.003379	1.000000

Figure 14: correlation of all variables part 1

```
1 sns.heatmap(df.corr(), cmap='Blues', annot=True)
2 plt.title('Correlation of all variables')
3 plt.show()
```

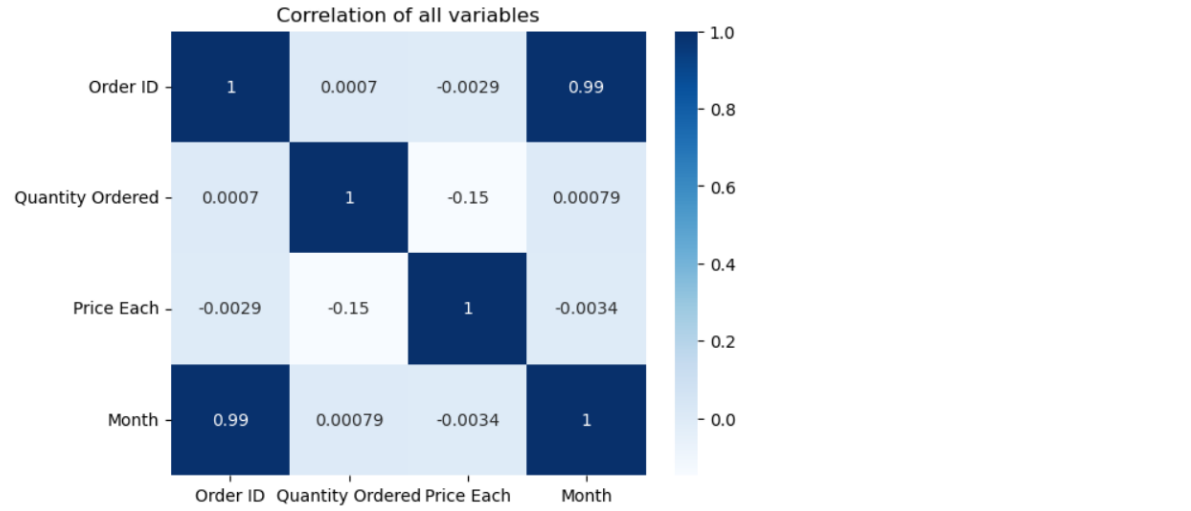


Figure 15: correlation of all variables part 2

The code generates a heatmap depicting the correlation between all variables of the dataframe. The df.corr() function calculates the correlation between every pair of variables and returns a correlation matrix. Using this correlation matrix, the sns.heatmap() function from the seaborn library is used to generate a heatmap. The cmap parameter is used to define the heatmap's colour map, the annot parameter is set to True to display the correlation coefficient values, and the plt.title() function is used to add a title to the heatmap. The heatmap is then displayed using the plt.show() function. The heatmap aids in identifying variables that are highly correlated and can aid in selecting the most pertinent variables for analysis.

4. Data Exploration

A. Which Month has the best sales? and how much was the earning in that month? Make a bar graph of sales as well.

A. Which Month has the best sales? and how much was the earning in that month? Make a bar graph of sales as well. ¶

```
df.columns
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month', 'City'],
      dtype='object')
```

```
df['Price Each'].dtypes
```

```
dtype('int64')
```

```
df['Quantity Ordered'].dtypes
```

```
dtype('int64')
```

```
# Add a Total Sales column
```

```
df['Total Sales'] = df['Quantity Ordered'] * df['Price Each']
df.head(5)
```

Figure 16: Best sales month and earning in that month part 1.

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	City	Total Sales
0	176558.0	USB-C Charging Cable	2	11	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4	Dallas	22
1	176559.0	Bose SoundSport Headphones	1	99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4	Boston	99
2	176560.0	Google Phone	1	600	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4	Los Angeles	600
3	176560.0	Wired Headphones	1	11	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4	Los Angeles	11
4	176561.0	Wired Headphones	1	11	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4	Los Angeles	11

```
# Group the data by month and calculate the total sales for each month
```

```
monthly_sales = df.groupby('Month')['Total Sales'].sum()
```

```
# Find the month with the highest total sales
```

```
best_month = monthly_sales.idxmax()
```

```
# Calculate the earnings in the best month
```

```
best_month_earnings = monthly_sales.loc[best_month]
```

```
# Print the result
```

```
print("The best sales were in", best_month, "with total earnings of $", best_month_earnings)
```

```
The best sales were in 12 with total earnings of $ 4591824
```

Figure 17: Best sales month and earning in that month part 2.

```
print("December has the best sales and the earning in that month was $4,591,824.")
```

```
December has the best sales and the earning in that month was $4,591,824.
```

Figure 18: Best sales month and earning in that month part 3.

```
# Create a bar graph of monthly sales
```

```
plt.bar(monthly_sales.index, monthly_sales.values)
plt.title('Total Sales by Month')
plt.xlabel('Month')
plt.ylabel('Total Sales (in USD)')
plt.xticks(range(1,13), [calendar.month_name[i] for i in range(1,13)],  rotation=45, ha='right')
plt.ticklabel_format(style='plain', axis='y')
plt.tight_layout()
plt.show()
```

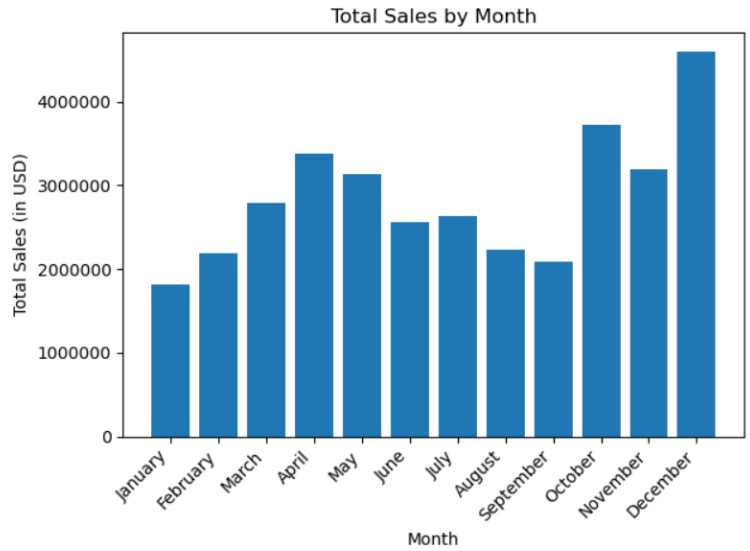


Figure 19: Best sales month and earning in that month part 4.

This code uses the 'df' dataframe to calculate total sales and the top month for sales. The 'Quantity Ordered' and 'Price Each' columns are multiplied to create a new column titled 'Total Sales'. The monthly sales are then calculated by clustering the 'Total Sales' column by 'Month' and summing the results. The code determines the greatest month for sales by using the 'idxmax' function to identify the index (i.e., the month) corresponding to the highest value in the'monthly_sales' series, and then the 'loc' function to identify the value (i.e., the total earnings) corresponding to that index. The code then prints the result and generates a bar plot of the monthly sales, with the x-axis labelled with the month names and the y-axis labelled with the total sales in USD.

B. Which city has sold the highest product?

B. Which city has sold the highest product?

```
df.columns
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month', 'City', 'Total Sales'],
      dtype='object')
```

```
city_highproduct = df.groupby('City')['Quantity Ordered'].sum().sort_values()
city_highproduct
```

```
City
 Austin          11153
 Portland        14053
 Seattle         16553
 Atlanta         16602
 Dallas          16730
 Boston          22528
 New York City   27932
 Los Angeles     33289
 San Francisco   50239
Name: Quantity Ordered, dtype: int64
```

```
City = city_highproduct.idxmax()
```

```
Number_of_product = city_highproduct.loc[City]
```

```
print(City,'has sold the highest product. \n They sold', Number_of_product, 'number of products')
```

```
San Francisco has sold the highest product.
They sold 50239 number of products
```

Figure 20: Highest product sold by city part 1.

```
plt.barh(city_highproduct.index,city_highproduct.values)
plt.title('City vs product sold')
plt.xlabel('Number of product sold')
plt.ylabel('City')
plt.show()
```

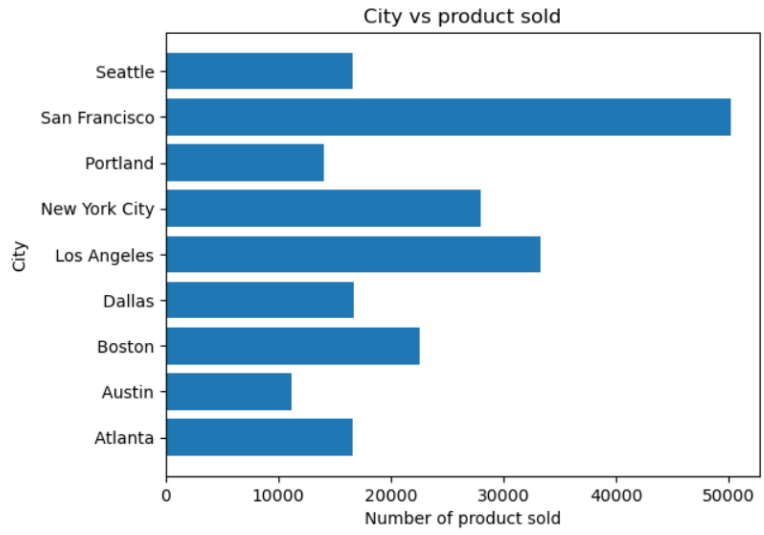


Figure 21: Highest product sold by city part 2.

This Python code analyzes the sales performance of different cities in the DataFrame df. It first groups the DataFrame by the 'City' column and calculates the

sum of 'Quantity Ordered' for each city. Then, it identifies the city with the highest sales using the idxmax() method and stores the name in the City variable.

The code then finds the number of products sold in the highest selling city using the loc() method and stores the value in the Number_of_product variable. It prints out a message stating which city has sold the highest product and how many products they sold.

Finally, the code creates a horizontal bar chart where the x-axis represents the number of products sold, and the y-axis represents the city names. It displays the chart using Matplotlib's show() method.

This code is useful for gaining insights into the sales performance of different cities and identifying the top-performing city in terms of sales.

C. Which product was sold the most in overall? Illustrate it through bar graph.

C. Which product was sold the most in overall? Illustrate it through bar graph.

```
df.columns
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month', 'City', 'Total Sales'],
      dtype='object')
```

```
highest_sold_product = df.groupby('Product')['Quantity Ordered'].sum()
highest_sold_product
```

```
Product
20in Monitor                  4129
27in 4K Gaming Monitor        6244
27in FHD Monitor              7550
34in Ultrawide Monitor        6199
AA Batteries (4-pack)        27635
AAA Batteries (4-pack)       31017
Apple Airpods Headphones     15661
Bose SoundSport Headphones   13457
Flatscreen TV                 4819
Google Phone                  5532
LG Dryer                       646
LG Washing Machine             666
Lightning Charging Cable     23217
Macbook Pro Laptop            4728
ThinkPad Laptop               4130
USB-C Charging Cable         23975
Vareebadd Phone               2068
Wired Headphones             20557
iPhone                        6849
Name: Quantity Ordered, dtype: int64
```

```
Product = highest_sold_product.idxmax()
```

```
print("The highest product sold was", Product)
```

```
The highest product sold was AAA Batteries (4-pack)
```

Figure 22: Most product sold in overall part 1

```
plt.barh(highest_sold_product.index,highest_sold_product.values, color='Purple')
plt.title('Highest Sold Product', fontsize=16)
plt.xlabel('Number of Product Sold', fontsize=12)
plt.ylabel('Product', fontsize=12)
plt.show()
```

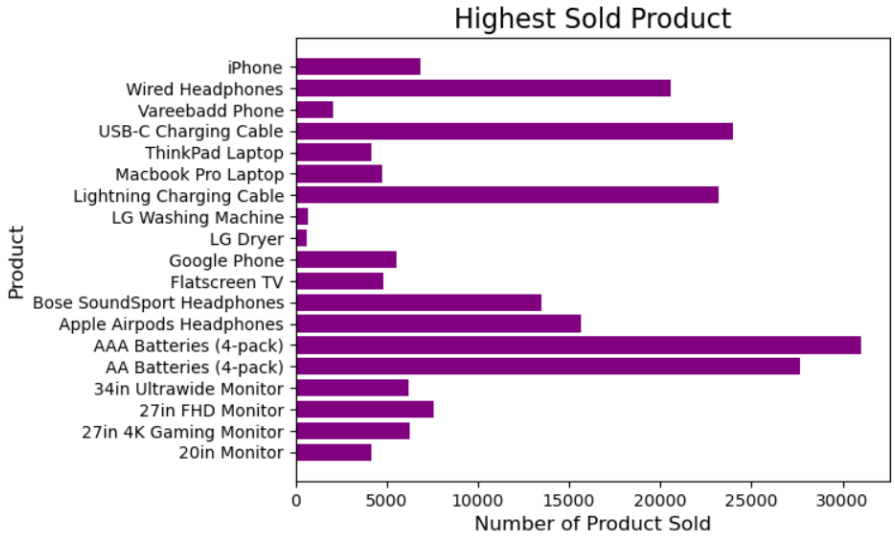


Figure 23: Most product sold in overall part 2

This Python code finds the product that had the highest sales in the DataFrame df and creates a horizontal bar chart to visualize the sales performance of all products.

It first groups the DataFrame by the 'Product' column and calculates the sum of 'Quantity Ordered' for each product. It then identifies the product with the highest sales using the idxmax() method and stores the name in the Product variable.

The code then creates a horizontal bar chart where the x-axis represents the number of products sold, and the y-axis represents the product names. It displays the chart using Matplotlib's show() method.

This code can be useful for analyzing sales performance and identifying the top-selling product.

D. Write a Python program to show histogram plot of any chosen variables. Use proper labels in the graph.

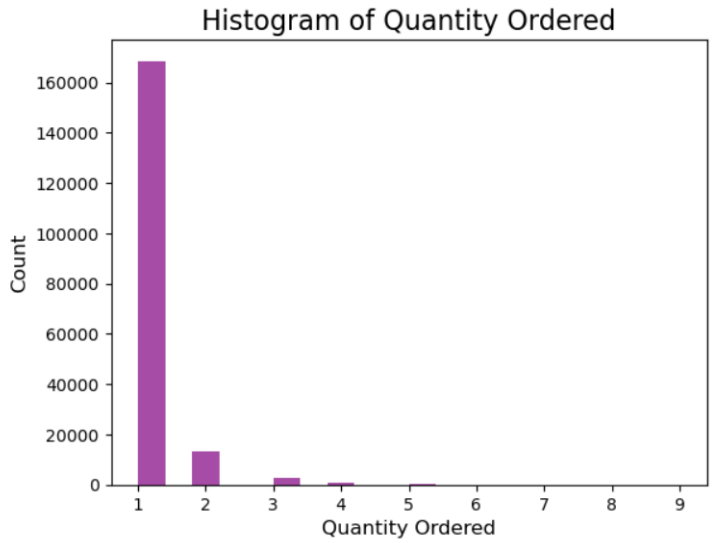
D. Write a Python program to show histogram plot of any chosen variables. Use proper labels in the graph.

```
df.columns
```

```
Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date',
       'Purchase Address', 'Month', 'City', 'Total Sales'],
      dtype='object')
```

Figure 24: Histogram of 'Quantity Ordered' variable part 1

```
plt.hist(df['Quantity Ordered'], bins=20, color='purple', alpha=0.7)
plt.title('Histogram of ' + 'Quantity Ordered', fontsize=16)
plt.xlabel('Quantity Ordered', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.show()
```




```
#The bar line in a histogram plot is not symmetric to odd numbers.
#This is because the bins in a histogram plot are defined by ranges, not by individual numbers.
#For example, if the bin width is set to 2, the bins will cover the ranges 1-2, 3-4, 5-6, etc.
#Therefore, if the data falls on an odd number, it will be included in the bin that starts with that odd number.
#As a result, the bars in the histogram will not be centered on the odd numbers.
```

Figure 25: Histogram of 'Quantity Ordered' variable part 2

This code generates a histogram depicting the distribution of the "Quantity Ordered" column within the dataframe. It creates 20 bins for the range of values in this column and makes the bars faintly transparent by colouring them purple with an alpha of 0.7. "Distribution of Quantity Ordered" is the title of the histogram. The x-axis is labelled "Quantity Ordered", while the y-axis is labelled "Count". The resulting scatterplot depicts the distribution of the number of items purchased by customers, revealing the typical number of items ordered at once.