Importing Required Libraries and Mounting Google Drive

```
import pandas as pd
import numpy as np
import matplotlib as mlt
import matplotlib.pyplot as plt
import sklearn as skt
import seaborn as sns
import math
import datetime as dt
import random
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore') # to ignore warnings


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Loading All Dataset

```
df_Repeated_Orders= pd.read_csv(r"/content/drive/MyDrive/PROJECT: CUSTOMER RETENTION AND ACQUISITION/Organic Dec_23 Repeated Orders _ Ma
df_Repeated_Orders.head(2)
# Tip: Add encoding='utf-8' or 'ISO-8859-1' if you face character errors.
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_CITY
0	CRN1852411579	17833073	2023-12-01T00:32:14.792Z	2023-12-01T00:43:50.417Z	91	LCV	2.0	DEL
1	CRN1439293796	17833089	2023-12-01T00:43:57.228Z	2023-12-01T00:48:04.433Z	9	HCV	3.0	BLR

2 rows × 24 columns

```
df_All_Channels_Customers = pd.read_csv(r"/content/drive/MyDrive/PROJECT: CUSTOMER RETENTION AND ACQUISITION/All Channels Customers May_
df_All_Channels_Customers.head(2)
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_C
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	C
1	CRN1027469647	16770968	f702f3ad63211c33201a4e90f399546e	8e9709aa-cb4c-461c-9984-f0e63f9ed664	91	LCV	5.0	C

2 rows × 26 columns

```
df_Customer_Acq= pd.read_csv(r"/content/drive/MyDrive/PROJECT: CUSTOMER RETENTION AND ACQUISITION/Organic Dec_23 Customer Acq.csv")
df_Customer_Acq.head(2)
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_C
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	C
1	CRN1027469647	16770968	f702f3ad63211c33201a4e90f399546e	8e9709aa-cb4c-461c-9984-f0e63f9ed664	91	LCV	5.0	C

2 rows × 26 columns

✅ Validating Datasets for Similarity and **Differences**

```
Start coding or generate with AI.
```

```
df_All_Channels_Customers.equals(df_Customer_Acq)
```

```
True
```

```
(df_All_Channels_Customers == df_Customer_Acq).sum().sum()          # Total number of matching cells
(df_All_Channels_Customers != df_Customer_Acq).sum().sum()
```

```
np.int64(945600)
```

```
(df_All_Channels_Customers != df_Customer_Acq).sum().sum()
```

```
np.int64(945600)
```

```
(df_All_Channels_Customers != df_Customer_Acq).sum()
```

	0
ORDER_ID_FO	0
CUSTOMER_ID	0
ADID	0
UUID	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	2
GEO_REGION_CITY	2
MARKET_TYPE	2
CUSTOMER_FARE	0
FREQ	0
FREQ_SME_RETAILS	0
CAMPAIGN_NAME	157661
CITY_NAME_FROM_CAMPAIGN	157289
ADGROUP_NAME	157661
ADGROUP_FORMATED	157661
CREATIVE_NAME	157661
NETWORK	157661
MOBILE_NUMBER	0
REG_DATE	0
REG_DATE_FORMATED	0
ORDER_DATE	0
ORDER_DATE_FORMATED	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

```
(df_All_Channels_Customers == df_Customer_Acq)
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_CITY	MARKET_TYPE	CUSTOMER_FARE
0	True	True	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True	True	True	True
...	...	...	...	...	...	...	...	...	...	...
157656	True	True	True	True	True	True	True	True	True	True
157657	True	True	True	True	True	True	True	True	True	True
157658	True	True	True	True	True	True	True	True	True	True
157659	True	True	True	True	True	True	True	True	True	True
157660	True	True	True	True	True	True	True	True	True	True

157661 rows × 26 columns

```
Start coding or generate with AI.
```

```
print(df_Repeated_Orders.shape)
print(df_All_Channels_Customers.shape)
print(df_Customer_Acq.shape)
```

```
(708621, 24)
(157661, 26)
(157661, 26)
```

```
print(df_Repeated_Orders.columns)
print(df_All_Channels_Customers.columns)
print(df_Customer_Acq.columns)
```

```
Index(['ORDER_ID_RO_IN_WINDOW', 'CUSTOMER_ID', 'REG_DATE_CUSTOMERS',
       'ORDER_DATE', 'VEHICLE_ID', 'VEHICLE_TYPE', 'GEO_REGION_ID',
       'GEO_REGION_CITY', 'MARKET_TYPE', 'CUSTOMER_FARE', 'ADID', 'UUID',
       'REG_DATE', 'FREQ', 'FREQ_SME_RETAILS', 'CAMPAIGN_NAME',
       'CITY_NAME_FROM_CAMPAIGN', 'ADGROUP_NAME', 'MOBILE_NUMBER',
       'REG_MONTH_YEAR', 'ORDER_MONTH_YEAR', 'CREATIVE_NAME', 'NETWORK',
       'CONVERSION_WINDOW'],
      dtype='object')
Index(['ORDER_ID_FO', 'CUSTOMER_ID', 'ADID', 'UUID', 'VEHICLE_ID',
       'VEHICLE_TYPE', 'GEO_REGION_ID', 'GEO_REGION_CITY', 'MARKET_TYPE',
       'CUSTOMER_FARE', 'FREQ', 'FREQ_SME_RETAILS', 'CAMPAIGN_NAME',
       'CITY_NAME_FROM_CAMPAIGN', 'ADGROUP_NAME', 'ADGROUP_FORMATED',
       'CREATIVE_NAME', 'NETWORK', 'MOBILE_NUMBER', 'REG_DATE',
       'REG_DATE_FORMATED', 'ORDER_DATE', 'ORDER_DATE_FORMATED',
       'REG_MONTH_YEAR', 'ORDER_MONTH_YEAR', 'CONVERSION_WINDOW'],
      dtype='object')
Index(['ORDER_ID_FO', 'CUSTOMER_ID', 'ADID', 'UUID', 'VEHICLE_ID',
       'VEHICLE_TYPE', 'GEO_REGION_ID', 'GEO_REGION_CITY', 'MARKET_TYPE',
       'CUSTOMER_FARE', 'FREQ', 'FREQ_SME_RETAILS', 'CAMPAIGN_NAME',
       'CITY_NAME_FROM_CAMPAIGN', 'ADGROUP_NAME', 'ADGROUP_FORMATED',
       'CREATIVE_NAME', 'NETWORK', 'MOBILE_NUMBER', 'REG_DATE',
       'REG_DATE_FORMATED', 'ORDER_DATE', 'ORDER_DATE_FORMATED',
       'REG_MONTH_YEAR', 'ORDER_MONTH_YEAR', 'CONVERSION_WINDOW'],
      dtype='object')
```

```
df_Repeated_Orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 708621 entries, 0 to 708620
Data columns (total 24 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   ORDER_ID_RO_IN_WINDOW    708621 non-null  object
 1   CUSTOMER_ID              708621 non-null  int64
 2   REG_DATE_CUSTOMERS       708621 non-null  object
 3   ORDER_DATE               708621 non-null  object
 4   VEHICLE_ID               708621 non-null  int64
 5   VEHICLE_TYPE             708607 non-null  object
 6   GEO_REGION_ID            708616 non-null  float64
 7   GEO_REGION_CITY          708621 non-null  object
 8   MARKET_TYPE              708621 non-null  object
 9   CUSTOMER_FARE            708621 non-null  float64
 10  ADID                     708621 non-null  object
 11  UUID                     708621 non-null  object
 12  REG_DATE                 708621 non-null  object
 13  FREQ                     708621 non-null  int64
 14  FREQ_SME_RETAILS         708621 non-null  object
 15  CAMPAIGN_NAME            1636 non-null    object
 16  CITY_NAME_FROM_CAMPAIGN  1300 non-null    object
 17  ADGROUP_NAME             0 non-null       float64
 18  MOBILE_NUMBER            708621 non-null  int64
 19  REG_MONTH_YEAR           708621 non-null  object
 20  ORDER_MONTH_YEAR         708621 non-null  object
 21  CREATIVE_NAME            0 non-null       float64
 22  NETWORK                  0 non-null       float64
 23  CONVERSION_WINDOW        708621 non-null  int64
dtypes: float64(5), int64(5), object(14)
memory usage: 129.8+ MB
```

🧹 Data Cleaning and Preprocessing

```
df_Repeated_Orders.isnull().sum()
# sns.heatmap(df_Repeated_Orders.isnull(), cbar=False, yticklabels=False)
```

	0
ORDER_ID_RO_IN_WINDOW	0
CUSTOMER_ID	0
REG_DATE_CUSTOMERS	0
ORDER_DATE	0
VEHICLE_ID	0
VEHICLE_TYPE	14
GEO_REGION_ID	5
GEO_REGION_CITY	0
MARKET_TYPE	0
CUSTOMER_FARE	0
ADID	0
UUID	0
REG_DATE	0
FREQ	0
FREQ_SME_RETAILS	0
CAMPAIGN_NAME	706985
CITY_NAME_FROM_CAMPAIGN	707321
ADGROUP_NAME	708621
MOBILE_NUMBER	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CREATIVE_NAME	708621
NETWORK	708621
CONVERSION_WINDOW	0

dtype: int64

```
df_Repeated_Orders.drop(['ADGROUP_NAME','CREATIVE_NAME','NETWORK'], axis=1, inplace=True)
df_Repeated_Orders.head(1)
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_CITY
0	CRN1852411579	17833073	2023-12-01T00:32:14.792Z	2023-12-01T00:43:50.417Z	91	LCV	2.0	DEL

1 rows × 21 columns

```
df_All_Channels_Customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 157661 entries, 0 to 157660
Data columns (total 26 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   ORDER_ID_FO              157661 non-null  object
 1   CUSTOMER_ID              157661 non-null  int64
 2   ADID                     157661 non-null  object
 3   UUID                     157661 non-null  object
 4   VEHICLE_ID               157661 non-null  int64
 5   VEHICLE_TYPE             157661 non-null  object
 6   GEO_REGION_ID            157659 non-null  float64
 7   GEO_REGION_CITY          157659 non-null  object
 8   MARKET_TYPE              157659 non-null  object
 9   CUSTOMER_FARE            157661 non-null  float64
 10  FREQ                     157661 non-null  int64
 11  FREQ_SME_RETAILS         157661 non-null  object
 12  CAMPAIGN_NAME            0 non-null       float64
 13  CITY_NAME_FROM_CAMPAIGN  372 non-null     object
 14  ADGROUP_NAME             0 non-null       float64
 15  ADGROUP_FORMATED         0 non-null       float64
 16  CREATIVE_NAME            0 non-null       float64
 17  NETWORK                  0 non-null       float64
 18  MOBILE_NUMBER            157661 non-null  int64
```

```
 19  REG_DATE                157661 non-null  object
 20  REG_DATE_FORMATED       157661 non-null  object
 21  ORDER_DATE              157661 non-null  object
 22  ORDER_DATE_FORMATED     157661 non-null  object
 23  REG_MONTH_YEAR          157661 non-null  object
 24  ORDER_MONTH_YEAR        157661 non-null  object
 25  CONVERSION_WINDOW       157661 non-null  int64
dtypes: float64(7), int64(5), object(14)
memory usage: 31.3+ MB
```

```
df_All_Channels_Customers.drop(['CAMPAIGN_NAME','ADGROUP_NAME','ADGROUP_FORMATED','CREATIVE_NAME','NETWORK'], axis=1, inplace=True)
df_All_Channels_Customers.head(1)
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_C
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	C

1 rows × 21 columns

```
df_Customer_Acq.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 157661 entries, 0 to 157660
Data columns (total 26 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   ORDER_ID_FO              157661 non-null  object
 1   CUSTOMER_ID              157661 non-null  int64
 2   ADID                     157661 non-null  object
 3   UUID                     157661 non-null  object
 4   VEHICLE_ID               157661 non-null  int64
 5   VEHICLE_TYPE             157661 non-null  object
 6   GEO_REGION_ID            157659 non-null  float64
 7   GEO_REGION_CITY          157659 non-null  object
 8   MARKET_TYPE              157659 non-null  object
 9   CUSTOMER_FARE            157661 non-null  float64
 10  FREQ                     157661 non-null  int64
 11  FREQ_SME_RETAILS         157661 non-null  object
 12  CAMPAIGN_NAME            0 non-null       float64
 13  CITY_NAME_FROM_CAMPAIGN  372 non-null     object
 14  ADGROUP_NAME             0 non-null       float64
 15  ADGROUP_FORMATED         0 non-null       float64
 16  CREATIVE_NAME            0 non-null       float64
 17  NETWORK                  0 non-null       float64
 18  MOBILE_NUMBER            157661 non-null  int64
 19  REG_DATE                 157661 non-null  object
 20  REG_DATE_FORMATED        157661 non-null  object
 21  ORDER_DATE               157661 non-null  object
 22  ORDER_DATE_FORMATED      157661 non-null  object
 23  REG_MONTH_YEAR           157661 non-null  object
 24  ORDER_MONTH_YEAR         157661 non-null  object
 25  CONVERSION_WINDOW        157661 non-null  int64
dtypes: float64(7), int64(5), object(14)
memory usage: 31.3+ MB
```

```
df_Customer_Acq.drop(['CAMPAIGN_NAME','ADGROUP_NAME','ADGROUP_FORMATED','CREATIVE_NAME','NETWORK'], axis=1, inplace=True)
df_Customer_Acq.head(1)
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_C
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	C

1 rows × 21 columns

```
print(df_Repeated_Orders.shape)
print(df_All_Channels_Customers.shape)
print(df_Customer_Acq.shape)
```

```
(708621, 21)
(157661, 21)
(157661, 21)
```

```
# For remove duplicates
# Check unique counts per column
unique_counts = df_Repeated_Orders.nunique()
```

```
print(unique_counts)
# 708621
```

```
ORDER_ID_RO_IN_WINDOW      708620
CUSTOMER_ID                184833
REG_DATE_CUSTOMERS         184819
ORDER_DATE                 708562
VEHICLE_ID                     43
VEHICLE_TYPE                    3
GEO_REGION_ID                  20
GEO_REGION_CITY                20
MARKET_TYPE                     4
CUSTOMER_FARE              112092
ADID                       184834
UUID                       184833
REG_DATE                   184819
FREQ                            3
FREQ_SME_RETAILS                2
CAMPAIGN_NAME                 190
CITY_NAME_FROM_CAMPAIGN        88
MOBILE_NUMBER              184833
REG_MONTH_YEAR                  3
ORDER_MONTH_YEAR                6
CONVERSION_WINDOW             219
dtype: int64
```

```
df_Repeated_Orders_unique = df_Repeated_Orders.drop_duplicates(subset=['ORDER_ID_RO_IN_WINDOW'])
print(df_Repeated_Orders_unique.shape) # Access shape as an attribute
df_Repeated_Orders_unique.head(1)
```

```
(708620, 21)
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_CITY
0	CRN1852411579	17833073	2023-12-01T00:32:14.792Z	2023-12-01T00:43:50.417Z	91	LCV	2.0	DEL

1 rows × 21 columns

```
df_Repeated_Orders_unique.isnull().sum()
```

	0
ORDER_ID_RO_IN_WINDOW	0
CUSTOMER_ID	0
REG_DATE_CUSTOMERS	0
ORDER_DATE	0
VEHICLE_ID	0
VEHICLE_TYPE	14
GEO_REGION_ID	5
GEO_REGION_CITY	0
MARKET_TYPE	0
CUSTOMER_FARE	0
ADID	0
UUID	0
REG_DATE	0
FREQ	0
FREQ_SME_RETAILS	0
CAMPAIGN_NAME	706984
CITY_NAME_FROM_CAMPAIGN	707320
MOBILE_NUMBER	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

```
df_Repeated_Orders_unique['VEHICLE_ID'].dtype
```

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

```
df_Repeated_Orders_unique['VEHICLE_TYPE'].dtype
```

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

```
df_Repeated_Orders_unique['VEHICLE_TYPE'].nunique()
```

```
3
```

```
df_Repeated_Orders_unique['VEHICLE_TYPE'].unique()
```

```
array(['LCV', 'HCV', '2W', nan], dtype=object)
```

```
vehicle_mapping = df_Repeated_Orders_unique.dropna(subset=["VEHICLE_TYPE"]).drop_duplicates(subset=["VEHICLE_ID"])[['VEHICLE_ID', 'VEHI(
print(vehicle_mapping)
```

```
{91: 'LCV', 9: 'HCV', 1: 'LCV', 14: 'LCV', 97: '2W', 96: 'LCV', 8: 'HCV', 10: 'HCV', 125: 'LCV', 109: 'HCV', 3: 'HCV', 110: 'HCV', 9
```

Start coding or generate with AI.

```
df_Repeated_Orders_unique[df_Repeated_Orders_unique['VEHICLE_TYPE'].isna()]
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REG
581601	CRN1307010634	17856274	2023-12-02T04:45:36.022Z	2024-04-22T04:37:08.538Z	132	NaN	6.0	
586577	CRN1524420711	17923476	2023-12-05T10:03:20.771Z	2024-04-23T09:08:40.181Z	132	NaN	6.0	
611814	CRN1952018573	17856274	2023-12-02T04:45:36.022Z	2024-05-01T05:44:13.958Z	132	NaN	6.0	
639192	CRN1476964264	18398811	2023-12-29T07:35:57.702Z	2024-05-09T09:03:57.575Z	133	NaN	8.0	
639241	CRN1592525120	18217931	2023-12-20T05:46:15.443Z	2024-05-09T09:09:19.805Z	133	NaN	8.0	
651087	CRN1864681638	18043721	2023-12-11T08:48:49.429Z	2024-05-13T07:16:24.555Z	133	NaN	8.0	
656804	CRN1772851919	17948721	2023-12-06T14:17:18.019Z	2024-05-15T04:33:49.397Z	133	NaN	8.0	
674309	CRN1552976018	17875428	2023-12-02T17:52:16.513Z	2024-05-20T10:17:19.377Z	132	NaN	6.0	
690387	CRN1302405529	17938440	2023-12-06T07:17:48.398Z	2024-05-25T06:44:08.324Z	133	NaN	8.0	
690952	CRN1151579704	18398811	2023-12-29T07:35:57.702Z	2024-05-25T08:44:35.388Z	133	NaN	8.0	
693392	CRN1157634203	18027531	2023-12-10T10:56:14.851Z	2024-05-26T07:55:12.614Z	133	NaN	8.0	
695457	CRN1927674795	18398811	2023-12-29T07:35:57.702Z	2024-05-27T06:48:18.370Z	133	NaN	8.0	
699975	CRN1862184869	18324121	2023-12-25T10:18:55.890Z	2024-05-28T09:56:47.424Z	133	NaN	8.0	
[illegible]	22468039	18398811	2023-12-29T07:35:57.702Z	2024-05-29T12:02:08.023Z	133	NaN	8.0	

```
df_Repeated_Orders_unique['VEHICLE_TYPE'].value_counts()
```

	count
VEHICLE_TYPE	
2W	469049
LCV	207559
HCV	31998

```
df_Repeated_Orders_unique['VEHICLE_TYPE'] = df_Repeated_Orders_unique['VEHICLE_TYPE'].fillna('2W')
```

```
df_Repeated_Orders_unique.isnull().sum()
```

	0
ORDER_ID_RO_IN_WINDOW	0
CUSTOMER_ID	0
REG_DATE_CUSTOMERS	0
ORDER_DATE	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	5
GEO_REGION_CITY	0
MARKET_TYPE	0
CUSTOMER_FARE	0
ADID	0
UUID	0
REG_DATE	0
FREQ	0
FREQ_SME_RETAILS	0
CAMPAIGN_NAME	706984
CITY_NAME_FROM_CAMPAIGN	707320
MOBILE_NUMBER	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

```
df_Repeated_Orders_unique[df_Repeated_Orders_unique['GEO_REGION_ID'].isna()]
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION
237055	CRN1143317774	18396410	2023-12-29T06:17:30.314Z	2024-01-07T07:23:43.399Z	91	LCV	NaN	
246202	CRN1093936191	18348608	2023-12-26T14:27:58.506Z	2024-01-10T04:30:58.639Z	91	LCV	NaN	
289975	CRN1703774069	18227407	2023-12-20T11:25:37.361Z	2024-01-23T06:51:40.410Z	97	2W	NaN	
295559	CRN1778000552	18227407	2023-12-20T11:25:37.361Z	2024-01-25T05:34:43.462Z	97	2W	NaN	
471724	CRN1322442508	18396410	2023-12-29T06:17:30.314Z	2024-03-18T10:43:15.312Z	91	LCV	NaN	

5 rows × 21 columns

```
vehicle_mapping1 = df_Repeated_Orders_unique.dropna(subset=["GEO_REGION_ID"]).drop_duplicates(subset=["GEO_REGION_CITY"])[['GEO_REGION_(
print(vehicle_mapping1)
```

```
{'DEL': 2.0, 'BLR': 3.0, 'MUM': 1.0, 'CHN': 5.0, 'HYD': 4.0, 'JPR': 7.0, 'CHD': 15.0, 'KOC': 18.0, 'NGP': 14.0, 'AHM': 6.0, 'LKO': 1
```

```
vehicle_mapping1
```

```
{'DEL': 2.0,
 'BLR': 3.0,
 'MUM': 1.0,
 'CHN': 5.0,
```

```
    'HYD': 4.0,
    'JPR': 7.0,
    'CHD': 15.0,
    'KOC': 18.0,
    'NGP': 14.0,
    'AHM': 6.0,
    'LKO': 15.0,
    'PUN': 8.0,
    'CBE': 12.0,
    'KOL': 9.0,
    'IND': 13.0,
    'SRT': 10.0,
    'VDR': 4.0,
    'LUD': 2.0,
    'NSK': 4.0,
    'KNP': 20.0}
```

```
df_Repeated_Orders_unique['GEO_REGION_ID'] = df_Repeated_Orders_unique['GEO_REGION_CITY'].map(vehicle_mapping1)
df_Repeated_Orders_unique
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION
0	CRN1852411579	17833073	2023-12-01T00:32:14.792Z	2023-12-01T00:43:50.417Z	91	LCV	2.0	
1	CRN1439293796	17833089	2023-12-01T00:43:57.228Z	2023-12-01T00:48:04.433Z	9	HCV	3.0	
2	CRN1456514356	17833116	2023-12-01T01:02:22.553Z	2023-12-01T01:05:56.070Z	1	LCV	1.0	
3	CRN1900903855	17833075	2023-12-01T00:34:05.421Z	2023-12-01T01:07:05.982Z	91	LCV	3.0	
4	CRN1615991541	17833127	2023-12-01T01:07:06.978Z	2023-12-01T01:12:59.590Z	91	LCV	2.0	
...	...	...	...	...	...	...	...	...
708616	CRN1089676304	18194980	2023-12-18T21:46:22.938Z	2024-05-30T20:56:12.775Z	97	2W	2.0	
708617	CRN1178580600	17852815	2023-12-01T17:11:28.626Z	2024-05-30T21:27:36.280Z	97	2W	5.0	
708618	CRN1783197932	18157374	2023-12-16T18:36:53.552Z	2024-05-30T21:32:19.506Z	91	LCV	7.0	
708619	CRN1671189673	18395755	2023-12-29T05:54:04.834Z	2024-05-30T21:53:47.897Z	1	LCV	4.0	
708620	CRN1651194576	18175337	2023-12-17T21:03:44.231Z	2024-05-30T22:02:11.899Z	91	LCV	2.0	

708620 rows × 21 columns

```
df_Repeated_Orders_unique.isnull().sum()
# 708620
```

	0
ORDER_ID_RO_IN_WINDOW	0
CUSTOMER_ID	0
REG_DATE_CUSTOMERS	0
ORDER_DATE	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	0
GEO_REGION_CITY	0
MARKET_TYPE	0
CUSTOMER_FARE	0
ADID	0
UUID	0
REG_DATE	0
FREQ	0
FREQ_SME_RETAILS	0
CAMPAIGN_NAME	706984
CITY_NAME_FROM_CAMPAIGN	707320
MOBILE_NUMBER	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

```
df_Repeated_Orders_unique.iloc[237054]
```

	237055
ORDER_ID_RO_IN_WINDOW	CRN1143317774
CUSTOMER_ID	18396410
REG_DATE_CUSTOMERS	2023-12-29T06:17:30.314Z
ORDER_DATE	2024-01-07T07:23:43.399Z
VEHICLE_ID	91
VEHICLE_TYPE	LCV
GEO_REGION_ID	13.0
GEO_REGION_CITY	IND
MARKET_TYPE	Hyper Growth
CUSTOMER_FARE	457.28
ADID	af9cbf57c3975571befff20d9d129a86
UUID	b7df2496-898e-4671-b1d4-71c3c7439b6c
REG_DATE	2023-12-29T06:17:30.314Z
FREQ	4
FREQ_SME_RETAILS	SME
CAMPAIGN_NAME	NaN
CITY_NAME_FROM_CAMPAIGN	NaN
MOBILE_NUMBER	7869810398
REG_MONTH_YEAR	12-2023
ORDER_MONTH_YEAR	1-2024
CONVERSION_WINDOW	9

```
CAMPAIGN_NAME = df_Repeated_Orders_unique[df_Repeated_Orders_unique['CAMPAIGN_NAME'].notna()]
CAMPAIGN_NAME
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION
172	CRN1178975468	17834530	2023-12-01T04:05:15.474Z	2023-12-01T04:07:19.580Z	1	LCV	4.0	
337	CRN1493982186	17835141	2023-12-01T04:38:05.377Z	2023-12-01T04:50:43.432Z	91	LCV	5.0	
1247	CRN1736195431	17838607	2023-12-01T06:43:52.757Z	2023-12-01T07:21:17.324Z	91	LCV	1.0	
1385	CRN1182232390	17840200	2023-12-01T07:36:15.757Z	2023-12-01T07:40:55.462Z	97	2W	3.0	
1454	CRN2069277738	17840433	2023-12-01T07:44:06.031Z	2023-12-01T07:48:19.380Z	97	2W	3.0	
...	...	...	...	...	...	...	...	...
707177	CRN1138794840	18287633	2023-12-23T10:35:12.502Z	2024-05-30T10:24:17.876Z	97	2W	3.0	
707402	CRN1559962855	18085632	2023-12-13T09:42:18.828Z	2024-05-30T10:58:29.769Z	91	LCV	2.0	
707533	CRN1967785526	18093632	2023-12-13T15:30:14.557Z	2024-05-30T11:24:11.054Z	97	2W	2.0	
708109	CRN1229706526	18201138	2023-12-19T07:19:50.486Z	2024-05-30T13:24:35.010Z	97	2W	8.0	
708322	CRN1766328230	17976072	2023-12-08T05:33:55.758Z	2024-05-30T14:32:23.085Z	97	2W	5.0	

1636 rows × 21 columns

```
Start coding or generate with AI.
```

Drop CITY_NAME_FROM_CAMPAIGN and CAMPAIGN_NAME

```
df_Repeated_Orders.drop(['CAMPAIGN_NAME','CITY_NAME_FROM_CAMPAIGN'], axis=1, inplace=True)
df_Repeated_Orders.head(1)
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_CITY
0	CRN1852411579	17833073	2023-12-01T00:32:14.792Z	2023-12-01T00:43:50.417Z	91	LCV	2.0	DEL

```
df_Repeated_Orders_unique.to_csv('df_Repeated_Orders_unique.csv', index=False)
```

```
df_Repeated_Orders_unique_saved=pd.read_csv('df_Repeated_Orders_unique.csv')
```

```
Start coding or generate with AI.
```

df_Repeated_Orders_unique_saved **file cleaned**

```
Start coding or generate with AI.
```

```
df_All_Channels_Customers.head(2)
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_C
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	C
1	CRN1027469647	16770968	f702f3ad63211c33201a4e90f399546e	8e9709aa-cb4c-461c-9984-f0e63f9ed664	91	LCV	5.0	C

2 rows × 21 columns

```
unique_counts_acc = df_All_Channels_Customers.nunique()
print(unique_counts_acc)
# 157661
```

```
ORDER_ID_FO               157660
CUSTOMER_ID               157660
ADID                      157661
UUID                      157660
VEHICLE_ID                    40
VEHICLE_TYPE                   3
GEO_REGION_ID                 20
GEO_REGION_CITY               20
MARKET_TYPE                    4
CUSTOMER_FARE              72837
FREQ                           3
FREQ_SME_RETAILS               2
CITY_NAME_FROM_CAMPAIGN       84
MOBILE_NUMBER             157660
REG_DATE                  157650
REG_DATE_FORMATED             78
ORDER_DATE                157644
ORDER_DATE_FORMATED          109
REG_MONTH_YEAR                 3
ORDER_MONTH_YEAR               4
CONVERSION_WINDOW             31
dtype: int64
```

```
# Identify duplicate rows based on specific columns
duplicates = df_All_Channels_Customers[df_All_Channels_Customers.duplicated(subset=['ADID'])]
duplicates
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_CITY	MARKET_TYPE	CUSTOMER_FARE	...	FR

0 rows × 21 columns

```
df_All_Channels_Customers.isnull().sum()
# 157661
```

	0
ORDER_ID_FO	0
CUSTOMER_ID	0
ADID	0
UUID	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	2
GEO_REGION_CITY	2
MARKET_TYPE	2
CUSTOMER_FARE	0
FREQ	0
FREQ_SME_RETAILS	0
CITY_NAME_FROM_CAMPAIGN	157289
MOBILE_NUMBER	0
REG_DATE	0
REG_DATE_FORMATED	0
ORDER_DATE	0
ORDER_DATE_FORMATED	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

```
df_All_Channels_Customers[df_All_Channels_Customers['GEO_REGION_ID'].isna()]
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGI
153860	CRN1143317774	18396410	af9cbf57c3975571befff20d9d129a86	b7df2496-898e-4671-b1d4-71c3c7439b6c	91	LCV	NaN	
154873	CRN1093936191	18348608	c9b9fc45cffe25f707c74c911032514a	1d567456-b6a3-48e4-a8b9-a23b9f338b3d	91	LCV	NaN	

2 rows × 21 columns

```
df_All_Channels_Customers.drop(index=[153860, 154873], inplace=True)
```

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

```
df_All_Channels_Customers
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_RI
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	
1	CRN1027469647	16770968	f702f3ad63211c33201a4e90f399546e	8e9709aa-cb4c-461c-9984-f0e63f9ed664	91	LCV	5.0	
2	CRN1333160570	16772155	44c6a471a58b3983b77a04a3e10c3cef	87fb5327-803f-4759-a3d2-5f494a10896a	97	2W	2.0	
3	CRN1811055197	16857609	80ab70e4d90303b8e4c9e20432e968c1	5a6d3a2b-488b-4aeb-ab5a-3a6472da3662	1	LCV	9.0	
4	CRN1845847943	16859663	6b023fcb779e97181031edf048a874f1	3783abf3-ae0a-45fd-bb28-22c189091b0d	97	2W	3.0	
...	...	...	...	...	...	...	...	
157654	CRN1912544765	18443599	96756d7e26f69ee519585b440d00641f	21ff97d0-ead3-420b-a98e-cf2fafbb2722	97	2W	2.0	
157655	CRN2047526730	18441498	ed97f8b9fbaa51f3b49bfcefb2b7baf9	fcdbe255-8c91-4031-9749-ddf5f7550444	97	2W	10.0	
157656	CRN1130917095	18436754	22b5cceeca1df197cfc7355f3a035440	4216a809-6aa7-4378-8790-97886f07d2f1	1	LCV	3.0	
157657	CRN1833050727	18437113	1730ad8f903f61a387722bebbecd6165	86edaf5d-ff19-4530-9300-2463a0ddb8b1	97	2W	3.0	
157658	CRN1124737494	18452844	153f7bfecacb6f072aaea4dde39d0149	91b7f312-612c-40a7-ba8d-4c11f03eabdf	1	LCV	1.0	

157659 rows × 21 columns

```
df_All_Channels_Customers.isnull().sum()
```

	0
ORDER_ID_FO	0
CUSTOMER_ID	0
ADID	0
UUID	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	0
GEO_REGION_CITY	0
MARKET_TYPE	0
CUSTOMER_FARE	0
FREQ	0
FREQ_SME_RETAILS	0
CITY_NAME_FROM_CAMPAIGN	157287
MOBILE_NUMBER	0
REG_DATE	0
REG_DATE_FORMATED	0
ORDER_DATE	0
ORDER_DATE_FORMATED	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

Drop CITY_NAME_FROM_CAMPAIGN

```
df_All_Channels_Customers.drop(['CITY_NAME_FROM_CAMPAIGN'], axis=1, inplace=True)
df_All_Channels_Customers.isnull().sum()
```

	0
ORDER_ID_FO	0
CUSTOMER_ID	0
ADID	0
UUID	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	0
GEO_REGION_CITY	0
MARKET_TYPE	0
CUSTOMER_FARE	0
FREQ	0
FREQ_SME_RETAILS	0
MOBILE_NUMBER	0
REG_DATE	0
REG_DATE_FORMATED	0
ORDER_DATE	0
ORDER_DATE_FORMATED	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

Start coding or generate with AI.

```
df_All_Channels_Customers.to_csv('df_All_Channels_Customers.csv', index=False)
```

saved

```
df_All_Channels_Customers_saved=pd.read_csv('df_All_Channels_Customers.csv')
```

Start coding or generate with AI.

```
df_Customer_Acq.head(2)
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_C
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	C
1	CRN1027469647	16770968	f702f3ad63211c33201a4e90f399546e	8e9709aa-cb4c-461c-9984-f0e63f9ed664	91	LCV	5.0	C

2 rows × 21 columns

```
unique_counts_ca = df_Customer_Acq.nunique()
print(unique_counts_ca)
# 157661
```

```
ORDER_ID_FO               157660
CUSTOMER_ID               157660
ADID                      157661
UUID                      157660
VEHICLE_ID                    40
VEHICLE_TYPE                   3
GEO_REGION_ID                 20
GEO_REGION_CITY               20
MARKET_TYPE                    4
CUSTOMER_FARE              72837
```

```
FREQ                        3
FREQ_SME_RETAILS            2
CITY_NAME_FROM_CAMPAIGN    84
MOBILE_NUMBER          157660
REG_DATE               157650
REG_DATE_FORMATED          78
ORDER_DATE             157644
ORDER_DATE_FORMATED       109
REG_MONTH_YEAR              3
ORDER_MONTH_YEAR            4
CONVERSION_WINDOW          31
dtype: int64
```

```
# Identify duplicate rows based on specific columns
duplicates = df_Customer_Acq[df_Customer_Acq.duplicated(subset=['ADID'])]
duplicates
```

ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_CITY	MARKET_TYPE	CUSTOMER_FARE	...	FR

0 rows × 21 columns

```
df_Customer_Acq.isnull().sum()
# 157661
```

	0
ORDER_ID_FO	0
CUSTOMER_ID	0
ADID	0
UUID	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	2
GEO_REGION_CITY	2
MARKET_TYPE	2
CUSTOMER_FARE	0
FREQ	0
FREQ_SME_RETAILS	0
CITY_NAME_FROM_CAMPAIGN	157289
MOBILE_NUMBER	0
REG_DATE	0
REG_DATE_FORMATED	0
ORDER_DATE	0
ORDER_DATE_FORMATED	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

```
df_Customer_Acq[df_Customer_Acq['GEO_REGION_ID'].isna()]
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGI
153860	CRN1143317774	18396410	af9cbf57c3975571befff20d9d129a86	b7df2496-898e-4671-b1d4-71c3c7439b6c	91	LCV	NaN	
154873	CRN1093936191	18348608	c9b9fc45cffe25f707c74c911032514a	1d567456-b6a3-48e4-a8b9-a23b9f338b3d	91	LCV	NaN	

2 rows × 21 columns

```
df_Customer_Acq.drop(index=[153860, 154873], inplace=True)
```

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

```
df_Customer_Acq
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_RI
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	
1	CRN1027469647	16770968	f702f3ad63211c33201a4e90f399546e	8e9709aa-cb4c-461c-9984-f0e63f9ed664	91	LCV	5.0	
2	CRN1333160570	16772155	44c6a471a58b3983b77a04a3e10c3cef	87fb5327-803f-4759-a3d2-5f494a10896a	97	2W	2.0	
3	CRN1811055197	16857609	80ab70e4d90303b8e4c9e20432e968c1	5a6d3a2b-488b-4aeb-ab5a-3a6472da3662	1	LCV	9.0	
4	CRN1845847943	16859663	6b023fcb779e97181031edf048a874f1	3783abf3-ae0a-45fd-bb28-22c189091b0d	97	2W	3.0	
...	...	...	...	...	...	...	...	
157654	CRN1912544765	18443599	96756d7e26f69ee519585b440d00641f	21ff97d0-ead3-420b-a98e-cf2fafbb2722	97	2W	2.0	
157655	CRN2047526730	18441498	ed97f8b9fbaa51f3b49bfcefb2b7baf9	fcdbe255-8c91-4031-9749-ddf5f7550444	97	2W	10.0	
157656	CRN1130917095	18436754	22b5cceeca1df197cfc7355f3a035440	4216a809-6aa7-4378-8790-97886f07d2f1	1	LCV	3.0	
157657	CRN1833050727	18437113	1730ad8f903f61a387722bebbecd6165	86edaf5d-ff19-4530-9300-2463a0ddb8b1	97	2W	3.0	
157658	CRN1124737494	18452844	153f7bfecacb6f072aaea4dde39d0149	91b7f312-612c-40a7-ba8d-4c11f03eabdf	1	LCV	1.0	

157659 rows × 21 columns

```
df_Customer_Acq.isnull().sum()
```

	0
ORDER_ID_FO	0
CUSTOMER_ID	0
ADID	0
UUID	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	0
GEO_REGION_CITY	0
MARKET_TYPE	0
CUSTOMER_FARE	0
FREQ	0
FREQ_SME_RETAILS	0
CITY_NAME_FROM_CAMPAIGN	157287
MOBILE_NUMBER	0
REG_DATE	0
REG_DATE_FORMATED	0
ORDER_DATE	0
ORDER_DATE_FORMATED	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

Start coding or generate with AI.

Drop CITY_NAME_FROM_CAMPAIGN

```
df_Customer_Acq.drop(['CITY_NAME_FROM_CAMPAIGN'], axis=1, inplace=True)
df_Customer_Acq.isnull().sum()
```

	0
ORDER_ID_FO	0
CUSTOMER_ID	0
ADID	0
UUID	0
VEHICLE_ID	0
VEHICLE_TYPE	0
GEO_REGION_ID	0
GEO_REGION_CITY	0
MARKET_TYPE	0
CUSTOMER_FARE	0
FREQ	0
FREQ_SME_RETAILS	0
MOBILE_NUMBER	0
REG_DATE	0
REG_DATE_FORMATED	0
ORDER_DATE	0
ORDER_DATE_FORMATED	0
REG_MONTH_YEAR	0
ORDER_MONTH_YEAR	0
CONVERSION_WINDOW	0

```
df_Customer_Acq.to_csv('df_Customer_Acq.csv', index=False)
```

Saved

```
df_Customer_Acq=pd.read_csv('df_Customer_Acq.csv')
```

```
Start coding or generate with AI.
```

RFM Analysis for Customer Segmentation

Calculate Key Metrics

Create a summary DataFrame with the following metrics for each customer:

Recency: Days since the customer's last purchase.

Frequency: Total number of purchases made by the customer.

Monetary Value (Order Value): Total or average amount spent by the customer.

```
from datetime import datetime, timezone
import pandas as pd

# Current date for recency calculation
current_date = datetime.now(timezone.utc)  # Get current date with UTC timezone

# Convert 'ORDER_DATE' to datetime objects before calculating recency
df_Repeated_Orders_unique_saved['ORDER_DATE'] = pd.to_datetime(df_Repeated_Orders_unique_saved['ORDER_DATE'])

# Ensure 'ORDER_DATE' is timezone-aware (if not already) and convert to UTC
df_Repeated_Orders_unique_saved['ORDER_DATE'] = df_Repeated_Orders_unique_saved['ORDER_DATE'].dt.tz_localize(None).dt.tz_localize('UTC')


# Calculate Recency
recency = df_Repeated_Orders_unique_saved.groupby('CUSTOMER_ID')['ORDER_DATE'].max().apply(lambda x: (current_date - x).days)

# Calculate Frequency
frequency = df_Repeated_Orders_unique_saved.groupby('CUSTOMER_ID')['ORDER_DATE'].count()
```

```
# Calculate Monetary Value using 'CUSTOMER_FARE'
monetary = df_Repeated_Orders_unique_saved.groupby('CUSTOMER_ID')['CUSTOMER_FARE'].sum()

# Combine into a single DataFrame
rfm = pd.DataFrame({
    'Recency': recency,
    'Frequency': frequency,
    'MonetaryValue': monetary
})

rfm
```

CUSTOMER_ID	Recency	Frequency	MonetaryValue
16772155	540	1	104.92
16785502	365	5	1822.00
16803571	375	5	1260.07
16815351	539	1	235.87
16843891	371	7	910.73
...	...	...	...
18453188	451	4	1572.41
18453195	513	1	904.70
18453221	513	1	362.03
18453223	423	4	2712.29
18453227	458	2	1850.28

184833 rows × 3 columns

```
# Handle Recency, Frequency, and Monetary Value scoring
rfm['Recency_Score'] = pd.qcut(rfm['Recency'], 4, labels=[4, 3, 2, 1])  # Lower Recency is better

# Use pd.cut for Frequency if duplicate bin edges are an issue
max_frequency = rfm['Frequency'].max()
rfm['Frequency_Score'] = pd.cut(
    rfm['Frequency'],
    bins=[0, 1, 3, 5, max_frequency],  # Define bins
    labels=[1, 2, 3, 4],               # Higher Frequency is better
    include_lowest=True
)

# Use pd.qcut for MonetaryValue as it often has sufficient variation
rfm['Monetary_Score'] = pd.qcut(rfm['MonetaryValue'], 4, labels=[1, 2, 3, 4])  # Higher Monetary is better

# Combine scores into an RFM Score
rfm['RFM_Score'] = rfm['Recency_Score'].astype(int) + rfm['Frequency_Score'].astype(int) + rfm['Monetary_Score'].astype(int)

# Output the result
rfm
```

CUSTOMER_ID	Recency	Frequency	MonetaryValue	Recency_Score	Frequency_Score	Monetary_Score	RFM_Score
16772155	540	1	104.92	1	1	1	3
16785502	365	5	1822.00	4	3	4	11
16803571	375	5	1260.07	4	3	4	11
16815351	539	1	235.87	1	1	2	4
16843891	371	7	910.73	4	4	3	11
...	...	...	...	...	...	...	...
18453188	451	4	1572.41	3	3	4	10
18453195	513	1	904.70	3	1	3	7
18453221	513	1	362.03	3	1	2	6
18453223	423	4	2712.29	3	3	4	10
18453227	458	2	1850.28	3	2	4	9

184833 rows × 7 columns

```
def segment_customer(row):
    if row['RFM_Score'] >= 10:
        return 'Champion'
    elif row['RFM_Score'] >= 7:
        return 'Loyal Customer'
    elif row['RFM_Score'] >= 5:
        return 'Potential Loyalist'
    else:
        return 'At Risk'

rfm['Segment'] = rfm.apply(segment_customer, axis=1)
rfm
```

CUSTOMER_ID	Recency	Frequency	MonetaryValue	Recency_Score	Frequency_Score	Monetary_Score	RFM_Score	Segment
16772155	540	1	104.92	1	1	1	3	At Risk
16785502	365	5	1822.00	4	3	4	11	Champion
16803571	375	5	1260.07	4	3	4	11	Champion
16815351	539	1	235.87	1	1	2	4	At Risk
16843891	371	7	910.73	4	4	3	11	Champion
...	...	...	...	...	...	...	...	...
18453188	451	4	1572.41	3	3	4	10	Champion
18453195	513	1	904.70	3	1	3	7	Loyal Customer
18453221	513	1	362.03	3	1	2	6	Potential Loyalist
18453223	423	4	2712.29	3	3	4	10	Champion
18453227	458	2	1850.28	3	2	4	9	Loyal Customer

184833 rows x 8 columns

```
segment_counts = rfm['Segment'].value_counts()
print(segment_counts)
```

```
Segment
Potential Loyalist    55033
Loyal Customer        49614
At Risk               44373
Champion              35813
Name: count, dtype: int64
```

```
rfm.reset_index(drop=False, inplace=True)
rfm
```

	CUSTOMER_ID	Recency	Frequency	MonetaryValue	Recency_Score	Frequency_Score	Monetary_Score	RFM_Score	Segment
0	16772155	540	1	104.92	1	1	1	3	At Risk
1	16785502	365	5	1822.00	4	3	4	11	Champion
2	16803571	375	5	1260.07	4	3	4	11	Champion
3	16815351	539	1	235.87	1	1	2	4	At Risk
4	16843891	371	7	910.73	4	4	3	11	Champion
...	...	...	...	...	...	...	...	...	...
184828	18453188	451	4	1572.41	3	3	4	10	Champion
184829	18453195	513	1	904.70	3	1	3	7	Loyal Customer
184830	18453221	513	1	362.03	3	1	2	6	Potential Loyalist
184831	18453223	423	4	2712.29	3	3	4	10	Champion
184832	18453227	458	2	1850.28	3	2	4	9	Loyal Customer

184833 rows x 9 columns

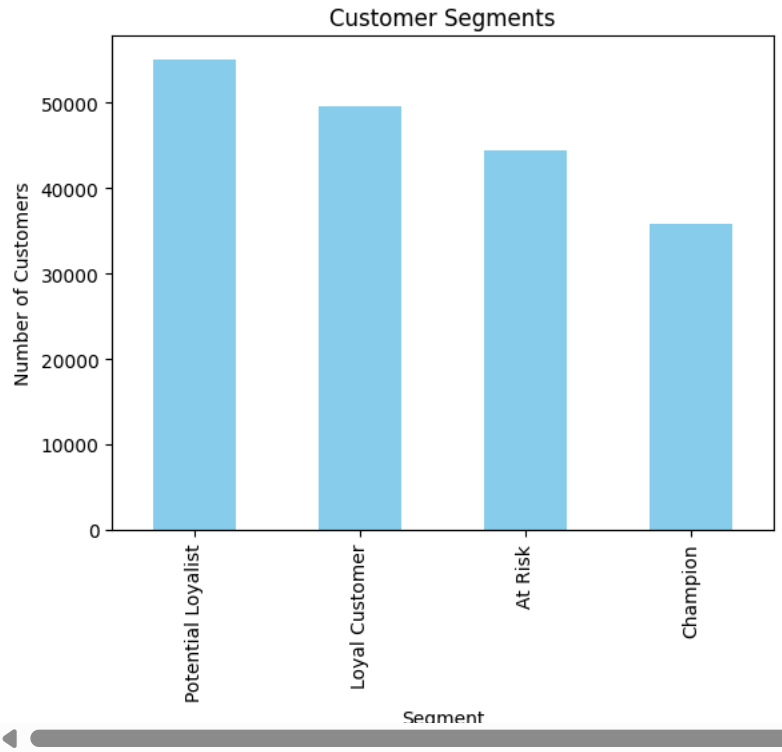
```
rfm.to_csv('rfm.csv', index=False)
```

```
# Filter rows where Segment is 'Champion'
champion_ids = rfm[rfm['Segment'] == 'Champion']['CUSTOMER_ID']

# Print all IDs of Champions
champion_ids
```

	CUSTOMER_ID
1	16785502
2	16803571
4	16843891
5	16852865
14	16911220
...	...
184804	18453072
184814	18453117
184826	18453183
184828	18453188
184831	18453223

35813 rows × 1 columns

```
import matplotlib.pyplot as plt

segment_counts.plot(kind='bar', color='skyblue', title='Customer Segments')
plt.xlabel('Segment')
plt.ylabel('Number of Customers')
plt.show()
```




```
rfm
```

	CUSTOMER_ID	Recency	Frequency	MonetaryValue	Recency_Score	Frequency_Score	Monetary_Score	RFM_Score	Segment
0	16772155	540	1	104.92	1	1	1	3	At Risk
1	16785502	365	5	1822.00	4	3	4	11	Champion
2	16803571	375	5	1260.07	4	3	4	11	Champion
3	16815351	539	1	235.87	1	1	2	4	At Risk
4	16843891	371	7	910.73	4	4	3	11	Champion
...	...	...	...	...	...	...	...	...	...
184828	18453188	451	4	1572.41	3	3	4	10	Champion
184829	18453195	513	1	904.70	3	1	3	7	Loyal Customer
184830	18453221	513	1	362.03	3	1	2	6	Potential Loyalist
184831	18453223	423	4	2712.29	3	3	4	10	Champion
184832	18453227	458	2	1850.28	3	2	4	9	Loyal Customer

184833 rows × 9 columns

Start coding or generate with AI.

Behavioral Segmentation

```
# df_Repeated_Orders
df_Repeated_Orders_unique_saved.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 708620 entries, 0 to 708619
Data columns (total 21 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   ORDER_ID_RO_IN_WINDOW    708620 non-null  object
 1   CUSTOMER_ID              708620 non-null  int64
 2   REG_DATE_CUSTOMERS       708620 non-null  object
 3   ORDER_DATE               708620 non-null  datetime64[ns, UTC]
 4   VEHICLE_ID               708620 non-null  int64
 5   VEHICLE_TYPE             708620 non-null  object
 6   GEO_REGION_ID            708620 non-null  float64
 7   GEO_REGION_CITY          708620 non-null  object
 8   MARKET_TYPE              708620 non-null  object
 9   CUSTOMER_FARE            708620 non-null  float64
 10  ADID                     708620 non-null  object
 11  UUID                     708620 non-null  object
 12  REG_DATE                 708620 non-null  object
 13  FREQ                     708620 non-null  int64
 14  FREQ_SME_RETAILS         708620 non-null  object
 15  CAMPAIGN_NAME            1636 non-null    object
 16  CITY_NAME_FROM_CAMPAIGN  1300 non-null    object
 17  MOBILE_NUMBER            708620 non-null  int64
 18  REG_MONTH_YEAR           708620 non-null  object
 19  ORDER_MONTH_YEAR         708620 non-null  object
 20  CONVERSION_WINDOW        708620 non-null  int64
dtypes: datetime64[ns, UTC](1), float64(2), int64(5), object(13)
memory usage: 113.5+ MB
```

```
df_Repeated_Orders_unique_saved.head(10)
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_
0	CRN1852411579	17833073	2023-12-01T00:32:14.792Z	2023-12-01 00:43:50.417000+00:00	91	LCV	2.0	
1	CRN1439293796	17833089	2023-12-01T00:43:57.228Z	2023-12-01 00:48:04.433000+00:00	9	HCV	3.0	
2	CRN1456514356	17833116	2023-12-01T01:02:22.553Z	2023-12-01 01:05:56.070000+00:00	1	LCV	1.0	
3	CRN1900903855	17833075	2023-12-01T00:34:05.421Z	2023-12-01 01:07:05.982000+00:00	91	LCV	3.0	
4	CRN1615991541	17833127	2023-12-01T01:07:06.978Z	2023-12-01 01:12:59.590000+00:00	91	LCV	2.0	
5	CRN2105568127	17833149	2023-12-01T01:18:35.101Z	2023-12-01 01:23:03.727000+00:00	91	LCV	3.0	
6	CRN1693867471	17833144	2023-12-01T01:15:32.514Z	2023-12-01 01:24:11.993000+00:00	14	LCV	5.0	
7	CRN1093365763	17833166	2023-12-01T01:24:54.686Z	2023-12-01 01:28:32.864000+00:00	97	2W	3.0	
8	CRN1465207187	17833180	2023-12-01T01:29:07.594Z	2023-12-01 01:31:59.490000+00:00	97	2W	3.0	
9	CRN2061756481	17833196	2023-12-01T01:34:16.115Z	2023-12-01 01:36:47.534000+00:00	97	2W	3.0	

10 rows × 21 columns

Start coding or generate with AI.

FOR CUSTOMER ID AND SME/RETAIL

```
# Filter rows where Freq == 4 and calculate the total CUSTOMER_FARE
total_customer_fare = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ'] == 4]['CUSTOMER_FARE'].sum()

print("Total CUSTOMER_FARE for Freq = 4:", total_customer_fare)
```

```
Total CUSTOMER_FARE for Freq = 4: 119016983.46999998
```

```
total_customer_fare = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ'] == 5]['CUSTOMER_FARE'].sum()

print("Total CUSTOMER_FARE for Freq = 5:", total_customer_fare)
```

```
Total CUSTOMER_FARE for Freq = 5: 101818927.09
```

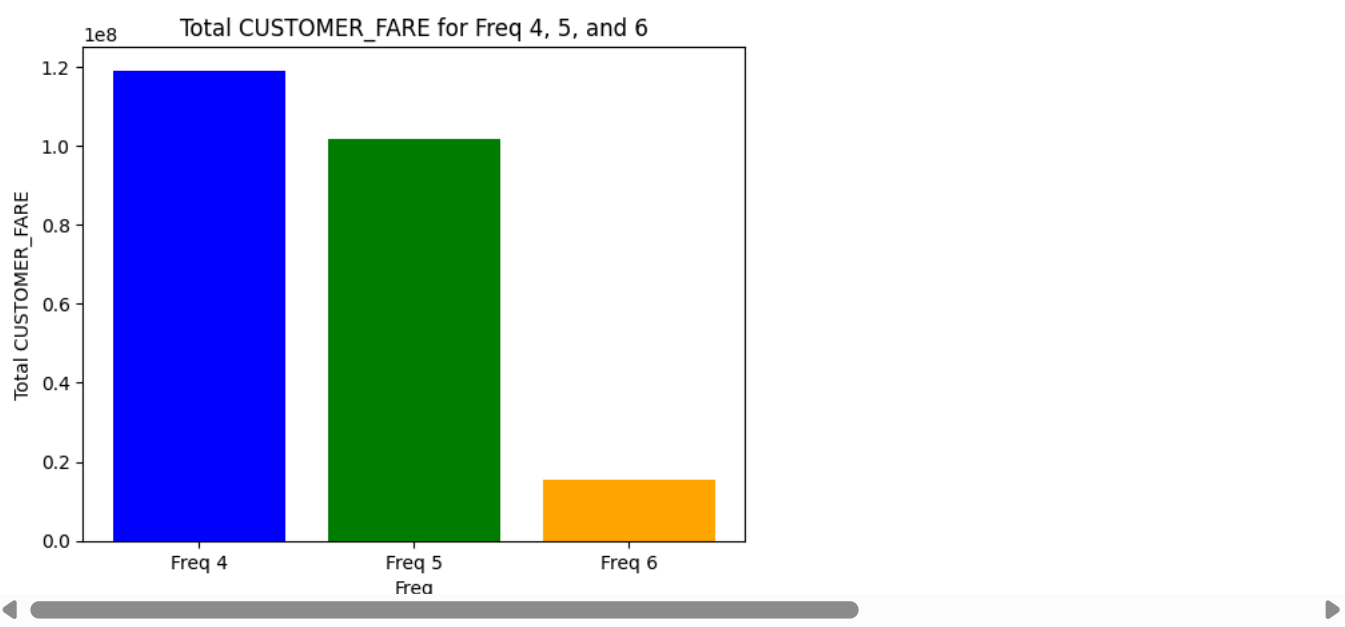
```
total_customer_fare = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ'] == 6]['CUSTOMER_FARE'].sum()

print("Total CUSTOMER_FARE for Freq = 6:", total_customer_fare)
```

```
Total CUSTOMER_FARE for Freq = 6: 15299622.950000001
```

```
# Filter rows where Freq is 4 or 5, then calculate the total CUSTOMER_FARE
total_customer_fare = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ'].isin([5, 6])]['CUSTOMER_FARE'].sum()

print("Total CUSTOMER_FARE for Freq = 6 and Freq = 5:", total_customer_fare)
```

```
Total CUSTOMER_FARE for Freq = 6 and Freq = 5: 117118550.04
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Filter the DataFrame to include only Freq = 4, 5, and 6
filtered_df = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ'].isin([4, 5, 6])]

# Group data by Freq and calculate the total CUSTOMER_FARE
total_fare_by_freq = filtered_df.groupby('FREQ')['CUSTOMER_FARE'].sum()

# Plotting the bar graph
plt.bar(total_fare_by_freq.index, total_fare_by_freq.values, color=['blue', 'green', 'orange'], tick_label=['Freq 4', 'Freq 5', 'Freq 6

# Adding labels and title
plt.xlabel('Freq')
plt.ylabel('Total CUSTOMER_FARE')
plt.title('Total CUSTOMER_FARE for Freq 4, 5, and 6')

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



Start coding or generate with AI.

Customer Lifetime Value (CLV) Analysis

```
# Convert dates to datetime format
df_Repeated_Orders_unique_saved['REG_DATE_CUSTOMERS'] = pd.to_datetime(df_Repeated_Orders_unique_saved['REG_DATE_CUSTOMERS'])
df_Repeated_Orders_unique_saved['ORDER_DATE'] = pd.to_datetime(df_Repeated_Orders_unique_saved['ORDER_DATE'])
```

```
# Group by CUSTOMER_ID to calculate key metrics
customer_metrics = df_Repeated_Orders_unique_saved.groupby('CUSTOMER_ID').agg(
    total_revenue=('CUSTOMER_FARE', 'sum'),
    order_count=('ORDER_ID_RO_IN_WINDOW', 'count'),
    first_order_date=('ORDER_DATE', 'min'),
    last_order_date=('ORDER_DATE', 'max'),
    reg_date=('REG_DATE_CUSTOMERS', 'min')
).reset_index()
customer_metrics
```

	CUSTOMER_ID	total_revenue	order_count	first_order_date	last_order_date	reg_date
0	16772155	104.92	1	2023-12-04 16:31:40.184000+00:00	2023-12-04 16:31:40.184000+00:00	2023-10-10 13:11:55.181000+00:00
1	16785502	1822.00	5	2023-12-08 08:41:05.948000+00:00	2024-05-27 14:22:07.309000+00:00	2023-10-11 08:51:44.805000+00:00
2	16803571	1260.07	5	2024-01-24 06:47:41.193000+00:00	2024-05-18 08:08:20.328000+00:00	2023-10-12 07:54:14.682000+00:00
3	16815351	235.87	1	2023-12-06 08:19:23.538000+00:00	2023-12-06 08:19:23.538000+00:00	2023-10-12 17:22:19.786000+00:00
4	16843891	910.73	7	2023-12-22 04:25:48.629000+00:00	2024-05-22 09:15:09.177000+00:00	2023-10-14 07:33:56.417000+00:00
...	...	...	...	...	...	...
184828	18453188	1572.41	4	2024-02-21 11:43:08.798000+00:00	2024-03-02 17:37:45.139000+00:00	2023-12-31 22:57:27.039000+00:00
184829	18453195	904.70	1	2023-12-31 23:19:10.929000+00:00	2023-12-31 23:19:10.929000+00:00	2023-12-31 23:04:54.055000+00:00

```
# Group by CUSTOMER_ID to calculate key metrics
customer_metrics = df_Repeated_Orders_unique_saved.groupby('CUSTOMER_ID').agg(
    total_revenue=('CUSTOMER_FARE', 'sum'),
    order_count=('ORDER_ID_RO_IN_WINDOW', 'count'),
    first_order_date=('ORDER_DATE', 'min'),
    last_order_date=('ORDER_DATE', 'max'),
    reg_date=('REG_DATE_CUSTOMERS', 'min')
).reset_index()
customer_metrics

# Calculate customer lifespan in months
customer_metrics['customer_lifespan_months'] = (
    (customer_metrics['last_order_date'] - customer_metrics['first_order_date']).dt.days / 30
).round()

# Avoid division by zero for avg_monthly_revenue
customer_metrics['avg_monthly_revenue'] = (
    customer_metrics['total_revenue'] / customer_metrics['customer_lifespan_months']
).where(customer_metrics['customer_lifespan_months'] > 0, 0)

# Calculate Historical CLV (set to 0 if lifespan is 0)
customer_metrics['Historical_clv/Predictive_clv'] = customer_metrics['avg_monthly_revenue'] * customer_metrics['customer_lifespan_months

customer_metrics['clv_segment'] = pd.qcut(
    customer_metrics['Historical_clv/Predictive_clv'],
    q=3,  # Reduce number of bins
    labels=['Low', 'Medium', 'High'],
    duplicates='drop'
)
bins = [0, 1, 1000, 2000, 50000, 150000, float('inf')]  # Define bin edges
labels = ['Low', 'Low_Medium', 'Medium', 'Medium_High', 'High', 'Extreme_High']  # Match labels

customer_metrics['clv_segment'] = pd.cut(
    customer_metrics['Historical_clv/Predictive_clv'],
    bins=bins,
    labels=labels,
    include_lowest=True
)

# Instead, directly fill NaN values using fillna
customer_metrics['clv_segment'] = customer_metrics['clv_segment'].fillna('Low')

# Format date columns to 'YYYY-MM-DD'
customer_metrics['first_order_date'] = customer_metrics['first_order_date'].dt.strftime('%Y-%m-%d')
customer_metrics['last_order_date'] = customer_metrics['last_order_date'].dt.strftime('%Y-%m-%d')
customer_metrics['reg_date'] = customer_metrics['reg_date'].dt.strftime('%Y-%m-%d')


# Verify no NaN or inf values
# print(customer_metrics[['avg_monthly_revenue', 'historical_clv', 'clv_segment']].isna().sum())
# print(customer_metrics[['avg_monthly_revenue', 'historical_clv']].replace([float('inf'), -float('inf')], 0).describe())
print(customer_metrics['clv_segment'].unique())
print(customer_metrics['clv_segment'].value_counts())

# Display the resulting DataFrame
customer_metrics
```

```
['Low', 'Medium', 'Low_Medium', 'Medium_High', 'High', 'Extreme_High']
Categories (6, object): ['Low' < 'Low_Medium' < 'Medium' < 'Medium_High' < 'High' < 'Extreme_High']
clv_segment
Low             116165
Low_Medium       32634
Medium_High      19865
Medium           15968
High               189
Extreme_High        12
Name: count, dtype: int64
```

	CUSTOMER_ID	total_revenue	order_count	first_order_date	last_order_date	reg_date	customer_lifespan_months	avg_monthly_
0	16772155	104.92	1	2023-12-04	2023-12-04	2023-10-10	0.0	
1	16785502	1822.00	5	2023-12-08	2024-05-27	2023-10-11	6.0	30
2	16803571	1260.07	5	2024-01-24	2024-05-18	2023-10-12	4.0	31
3	16815351	235.87	1	2023-12-06	2023-12-06	2023-10-12	0.0	
4	16843891	910.73	7	2023-12-22	2024-05-22	2023-10-14	5.0	18
...	...	...	...	...	...	...	...	
184828	18453188	1572.41	4	2024-02-21	2024-03-02	2023-12-31	0.0	
184829	18453195	904.70	1	2023-12-31	2023-12-31	2023-12-31	0.0	

```
# Make sure 'clv_segment' is treated as a category (optional but good practice)
import plotly.express as px
customer_metrics['clv_segment'] = customer_metrics['clv_segment'].astype('category')

# Create the pie chart
fig_pie = px.pie(
    customer_metrics,
    names='clv_segment',
    title='CLV Segment Distribution',
    hole=0.4,
    color_discrete_sequence=px.colors.qualitative.Set2
)

# Update trace for better readability
fig_pie.update_traces(textinfo='percent+label')

# Show the plot
fig_pie.show()
```


```
customer_metrics.to_csv('CLV-customer_metrics.csv', index=False)
```

```
Start coding or generate with AI.
```

```
Medium_clv= customer_metrics[customer_metrics['clv_segment'] == 'Medium']
Medium_clv
```

	CUSTOMER_ID	total_revenue	order_count	first_order_date	last_order_date	reg_date	customer_lifespan_months	avg_monthly_
1	16785502	1822.00	5	2023-12-08	2024-05-27	2023-10-11	6.0	303
2	16803571	1260.07	5	2024-01-24	2024-05-18	2023-10-12	4.0	315
21	16978769	1782.12	6	2024-02-26	2024-05-03	2023-10-20	2.0	89
27	17023484	1558.63	4	2024-01-01	2024-05-16	2023-10-23	4.0	389
42	17189819	1110.10	7	2024-03-04	2024-05-29	2023-10-31	3.0	370
...	...	...	...	...	...	...	...	
184765	18452860	1611.60	3	2024-01-13	2024-02-24	2023-12-31	1.0	161
184777	18452944	1571.88	5	2024-03-04	2024-05-11	2023-12-31	2.0	785

```
High_Predictive_CLV = customer_metrics[customer_metrics['clv_segment'] == 'High']
High_Predictive_CLV
```

	CUSTOMER_ID	total_revenue	order_count	first_order_date	last_order_date	reg_date	customer_lifespan_months	avg_monthly_
636	17834593	61257.64	63	2023-12-01	2024-03-13	2023-12-01	3.0	20419
924	17835445	54936.32	99	2023-12-01	2024-05-30	2023-12-01	6.0	9156
1135	17836074	54259.10	69	2023-12-21	2024-05-02	2023-12-01	4.0	13564
1423	17836971	91199.80	126	2023-12-01	2024-05-30	2023-12-01	6.0	15199
1658	17837625	61882.64	100	2023-12-01	2024-03-30	2023-12-01	4.0	1547(
...	...	...	...	...	...	...	...	
179262	18431171	69263.48	184	2024-01-08	2024-05-29	2023-12-30	5.0	13852
179710	18432971	114283.39	64	2024-01-08	2024-05-29	2023-12-30	5.0	22856

```
Start coding or generate with AI.
```

```
df_Repeated_Orders_unique_saved.columns
```

```
Index(['ORDER_ID_RO_IN_WINDOW', 'CUSTOMER_ID', 'REG_DATE_CUSTOMERS',
       'ORDER_DATE', 'VEHICLE_ID', 'VEHICLE_TYPE', 'GEO_REGION_ID',
       'GEO_REGION_CITY', 'MARKET_TYPE', 'CUSTOMER_FARE', 'ADID', 'UUID',
       'REG_DATE', 'FREQ', 'FREQ_SME_RETAILS', 'CAMPAIGN_NAME',
       'CITY_NAME_FROM_CAMPAIGN', 'MOBILE_NUMBER', 'REG_MONTH_YEAR',
       'ORDER_MONTH_YEAR', 'CONVERSION_WINDOW'],
      dtype='object')
```

```
customer_metrics
```

	CUSTOMER_ID	total_revenue	order_count	first_order_date	last_order_date	reg_date	customer_lifespan_months	avg_monthly_
0	16772155	104.92	1	2023-12-04	2023-12-04	2023-10-10	0.0	(
1	16785502	1822.00	5	2023-12-08	2024-05-27	2023-10-11	6.0	303
2	16803571	1260.07	5	2024-01-24	2024-05-18	2023-10-12	4.0	315
3	16815351	235.87	1	2023-12-06	2023-12-06	2023-10-12	0.0	(
4	16843891	910.73	7	2023-12-22	2024-05-22	2023-10-14	5.0	182
...	...	...	...	...	...	...	...	
184828	18453188	1572.41	4	2024-02-21	2024-03-02	2023-12-31	0.0	(
184829	18453195	904.70	1	2023-12-31	2023-12-31	2023-12-31	0.0	(

Start coding or generate with AI.

```
# Customers with high RFM & high CLV → “Loyal Premiums
```

CLV

Start coding or generate with AI.

```
df_Repeated_Orders_unique_saved
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_I
0	CRN1852411579	17833073	2023-12-01 00:32:14.792000+00:00	2023-12-01 00:43:50.417000+00:00	91	LCV	2.0	
1	CRN1439293796	17833089	2023-12-01 00:43:57.228000+00:00	2023-12-01 00:48:04.433000+00:00	9	HCV	3.0	
2	CRN1456514356	17833116	2023-12-01 01:02:22.553000+00:00	2023-12-01 01:05:56.070000+00:00	1	LCV	1.0	
3	CRN1900903855	17833075	2023-12-01 00:34:05.421000+00:00	2023-12-01 01:07:05.982000+00:00	91	LCV	3.0	
4	CRN1615991541	17833127	2023-12-01 01:07:06.978000+00:00	2023-12-01 01:12:59.590000+00:00	91	LCV	2.0	
...	...	...	...	...	...	...	...	
708615	CRN1089676304	18194980	2023-12-18 21:46:22.938000+00:00	2024-05-30 20:56:12.775000+00:00	97	2W	2.0	
708616	CRN1178580600	17852815	2023-12-01 17:11:28.626000+00:00	2024-05-30 21:27:36.280000+00:00	97	2W	5.0	
708617	CRN1783197932	18157374	2023-12-16 18:36:53.552000+00:00	2024-05-30 21:32:19.506000+00:00	91	LCV	7.0	
708618	CRN1671189673	18395755	2023-12-29 05:54:04.834000+00:00	2024-05-30 21:53:47.897000+00:00	1	LCV	4.0	
708619	CRN1651194576	18175337	2023-12-17 21:03:44.231000+00:00	2024-05-30 22:02:11.899000+00:00	91	LCV	2.0	

708620 rows × 21 columns

```
total_customer_fare_for_clv = df_Repeated_Orders_unique_saved['CUSTOMER_FARE'].sum()

print("Total Salary:", total_customer_fare_for_clv)
```

```
Total Salary: 236135533.5100001
```

```
119016983.46999998+117118550.04
```

```
236135533.51
```

```
Start coding or generate with AI.
```

```
total_customer_fare_for_sme = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ_SME_RETAILS'] == 'SME']['CUSTOMER_FA

print("Total CUSTOMER_FARE for sme:", total_customer_fare_for_sme)
```

```
Total CUSTOMER_FARE for sme: 119016983.46999998
```

```
total_customer_fare_for_Retail = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ_SME_RETAILS'] == 'Retail']['CUST(

print("Total CUSTOMER_FARE for retail:", total_customer_fare_for_Retail)
```

```
Total CUSTOMER_FARE for retail: 117118550.04
```

```
df_Customer_Acq.head(1)
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGION_C
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	C

```
total_customer_acq_fare_for_clv = df_Customer_Acq['CUSTOMER_FARE'].sum()

print("Total Salary:", total_customer_acq_fare_for_clv)
```

```
Total Salary: 64582293.49
```

```
total_Customer_Acq_fare_for_sme = df_Customer_Acq[df_Customer_Acq['FREQ_SME_RETAILS'] == 'SME']['CUSTOMER_FARE'].sum()

print("Total CUSTOMER_FARE for sme:", total_Customer_Acq_fare_for_sme)
```

```
Total CUSTOMER_FARE for sme: 19290267.2
```

```
total_Customer_Acq_fare_for_Retail = df_Customer_Acq[df_Customer_Acq['FREQ_SME_RETAILS'] == 'Retail']['CUSTOMER_FARE'].sum()

print("Total CUSTOMER_FARE for Retail:", total_Customer_Acq_fare_for_Retail)
```

```
Total CUSTOMER_FARE for Retail: 45292026.29
```

```
Start coding or generate with AI.
```

```
total_customer_fare_for_sme/total_Customer_Acq_fare_for_sme
```

```
np.float64(6.169794447948341)
```

```
# Sort by OrderDate and keep the first transaction for each CustomerID
first_order_table = df_All_Channels_Customers_saved.sort_values('ORDER_DATE').drop_duplicates('CUSTOMER_ID', keep='first')


first_order_table
```

	ORDER_ID_FO	CUSTOMER_ID	ADID	UUID	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_RI
0	CRN1854579779	16709077	2b5f69ad996c7a517d901e14fe6984f0	93dc12ed-e931-46fd-aea6-b5abc4343ccc	1	LCV	5.0	
1	CRN1027469647	16770968	f702f3ad63211c33201a4e90f399546e	8e9709aa-cb4c-461c-9984-f0e63f9ed664	91	LCV	5.0	
2	CRN1333160570	16772155	44c6a471a58b3983b77a04a3e10c3cef	87fb5327-803f-4759-a3d2-5f494a10896a	97	2W	2.0	
3	CRN1811055197	16857609	80ab70e4d90303b8e4c9e20432e968c1	5a6d3a2b-488b-4aeb-ab5a-3a6472da3662	1	LCV	9.0	
4	CRN1845847943	16859663	6b023fcb779e97181031edf048a874f1	3783abf3-ae0a-45fd-bb28-22c189091b0d	97	2W	3.0	
...	...	...	...	...	...	...	...	
157654	CRN1912544765	18443599	96756d7e26f69ee519585b440d00641f	21ff97d0-ead3-420b-a98e-cf2fafbb2722	97	2W	2.0	
157655	CRN2047526730	18441498	ed97f8b9fbaa51f3b49bfcefb2b7baf9	fcdbe255-8c91-4031-9749-ddf5f7550444	97	2W	10.0	
157656	CRN1130917095	18436754	22b5cceeca1df197cfc7355f3a035440	4216a809-6aa7-4378-8790-97886f07d2f1	1	LCV	3.0	
157657	CRN1833050727	18437113	1730ad8f903f61a387722bebbecd6165	86edaf5d-ff19-4530-9300-2463a0ddb8b1	97	2W	3.0	
157658	CRN1124737494	18452844	153f7bfecacb6f072aaea4dde39d0149	91b7f312-612c-40a7-ba8d-4c11f03eabdf	1	LCV	1.0	

157658 rows × 20 columns

```
first_order_table.to_csv('first_order_table.csv', index=False)
```

```
first_order_table_fare_for_clv = first_order_table['CUSTOMER_FARE'].sum()
```

```
print("Total Salary:", first_order_table_fare_for_clv)
```

```
Total Salary: 64581853.62
```

```
total_first_order_table_fare_for_sme = first_order_table[first_order_table['FREQ_SME_RETAILS'] == 'SME']['CUSTOMER_FARE'].sum()
```

```
print("Total first_order_table CUSTOMER_FARE for sme:", total_first_order_table_fare_for_sme)
```

```
Total first_order_table CUSTOMER_FARE for sme: 19289827.33
```

```
total_first_order_table_fare_for_Retail = first_order_table[first_order_table['FREQ_SME_RETAILS'] == 'Retail']['CUSTOMER_FARE'].sum()
```

```
print("Total first_order_table CUSTOMER_FARE for Retail:", total_first_order_table_fare_for_Retail)
```

```
Total first_order_table CUSTOMER_FARE for Retail: 45292026.29
```

```
clv_for_sme=total_customer_fare_for_sme/total_first_order_table_fare_for_sme
clv_for_sme
```

```
np.float64(6.169935139072082)
```

```
clv_for_retail=total_customer_fare_for_Retail/total_first_order_table_fare_for_Retail
clv_for_retail
```

```
np.float64(2.585853617811278)
```

Start coding or generate with AI.

```
# Step 1: Filter SME customers
sme_customers = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ_SME_RETAILS'] == 'SME']
sme_customers
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_I
0	CRN1852411579	17833073	2023-12-01 00:32:14.792000+00:00	2023-12-01 00:43:50.417000+00:00	91	LCV	2.0	
1	CRN1439293796	17833089	2023-12-01 00:43:57.228000+00:00	2023-12-01 00:48:04.433000+00:00	9	HCV	3.0	
6	CRN1693867471	17833144	2023-12-01 01:15:32.514000+00:00	2023-12-01 01:24:11.993000+00:00	14	LCV	5.0	
10	CRN1993516042	17833138	2023-12-01 01:11:12.012000+00:00	2023-12-01 01:37:59.404000+00:00	91	LCV	1.0	
14	CRN1129847023	17833279	2023-12-01 01:53:49.504000+00:00	2023-12-01 01:58:37.579000+00:00	97	2W	3.0	
...	...	...	...	...	...	...	...	
708609	CRN1375743548	17981297	2023-12-08 08:36:51.953000+00:00	2024-05-30 18:55:59.338000+00:00	91	LCV	2.0	
708611	CRN1645329111	18265266	2023-12-22 09:50:39.746000+00:00	2024-05-30 19:57:18.457000+00:00	97	2W	14.0	
708612	CRN1042745474	18197964	2023-12-19 05:28:06.537000+00:00	2024-05-30 20:11:58.266000+00:00	97	2W	2.0	
708613	CRN1208621716	17874082	2023-12-02 15:43:15.564000+00:00	2024-05-30 20:37:00.308000+00:00	9	HCV	3.0	
708616	CRN1178580600	17852815	2023-12-01 17:11:28.626000+00:00	2024-05-30 21:27:36.280000+00:00	97	2W	5.0	

334725 rows × 21 columns

```
# Count total CUSTOMER_IDs
total_customers = sme_customers['CUSTOMER_ID'].count()

# Count unique CUSTOMER_IDs
unique_customers = sme_customers['CUSTOMER_ID'].nunique()

# Count non-unique CUSTOMER_IDs (duplicates)
non_unique_customers = total_customers - unique_customers

print(f"Total CUSTOMER_IDs: {total_customers}")
print(f"Unique CUSTOMER_IDs: {unique_customers}")
print(f"Non-unique CUSTOMER_IDs: {non_unique_customers}")
```

```
Total CUSTOMER_IDs: 334725
Unique CUSTOMER_IDs: 49448
Non-unique CUSTOMER_IDs: 285277
```

```
# Step 2: Calculate Total Revenue for each SME customer
total_revenue_of_sme_customers = sme_customers.groupby('CUSTOMER_ID')['CUSTOMER_FARE'].sum()



# Step 3: Calculate Average Order Value (AOV)
total_orders_sme = sme_customers.groupby('CUSTOMER_ID')['ORDER_ID_RO_IN_WINDOW'].count()



aov = total_revenue_of_sme_customers / total_orders_sme
aov.mean()
```

```
np.float64(452.01303302364454)
```

```
# Step 4: Calculate Purchase Frequency (PF)
purchase_frequency = total_orders_sme.mean()
purchase_frequency
```

```
np.float64(6.769232324866526)
```

```
# Step 5: Calculate Customer Lifetime (CL)
sme_customers['ORDER_DATE'] = pd.to_datetime(sme_customers['ORDER_DATE'])
customer_lifetime = sme_customers.groupby('CUSTOMER_ID').apply(
    lambda x: (x['ORDER_DATE'].max() - x['ORDER_DATE'].min()).days / 365
).mean()
customer_lifetime
```

```
np.float64(0.1399994570191905)
```

```
# Convert ORDER_DATE to datetime if not already done
sme_customers['ORDER_DATE'] = pd.to_datetime(sme_customers['ORDER_DATE'])

# Initialize a list to store the customer lifetimes
customer_lifetimes = []

# Loop through each unique CUSTOMER_ID
for customer_id in sme_customers['CUSTOMER_ID'].unique():
    # Filter data for the current customer
    customer_data = sme_customers[sme_customers['CUSTOMER_ID'] == customer_id]

    # Find the first and last order dates
    first_order_date = customer_data['ORDER_DATE'].min()
    last_order_date = customer_data['ORDER_DATE'].max()

    # Calculate the customer lifetime in years
    lifetime_years = (last_order_date - first_order_date).days / 365

    # Append the lifetime to the list
    customer_lifetimes.append(lifetime_years)

# Calculate the average customer lifetime
average_customer_lifetime = sum(customer_lifetimes) / len(customer_lifetimes)

print(f"Average Customer Lifetime (in years): {average_customer_lifetime}")
```

```
Average Customer Lifetime (in years): 0.13999945701918884
```

```
# Step 6: Calculate CLV
clv = aov.mean() * purchase_frequency * customer_lifetime

print("Customer Lifetime Value (CLV) for SME customers:", clv)
```

```
Customer Lifetime Value (CLV) for SME customers: 428.3677114141546
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
import pandas as pd

# Step 1: Filter Retail customers

Retail_customers = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['FREQ_SME_RETAILS'] == 'Retail']


# Step 2: Calculate Total Revenue for each Retail customer
```

```
total_revenue_of_Retail_customers = Retail_customers.groupby('CUSTOMER_ID')['CUSTOMER_FARE'].sum()

# Step 3: Calculate Average Order Value (AOV)
total_orders_Retail = Retail_customers.groupby('CUSTOMER_ID')['ORDER_ID_RO_IN_WINDOW'].count()
aov = total_revenue_of_Retail_customers / total_orders_Retail

# Step 4: Calculate Purchase Frequency (PF)
purchase_frequency = total_orders_Retail.mean()

# Step 5: Calculate Customer Lifetime (CL)
Retail_customers['ORDER_DATE'] = pd.to_datetime(Retail_customers['ORDER_DATE'])
customer_lifetime = Retail_customers.groupby('CUSTOMER_ID').apply(
    lambda x: (x['ORDER_DATE'].max() - x['ORDER_DATE'].min()).days / 365
).mean()

# Step 6: Calculate CLV
clv = aov.mean() * purchase_frequency * customer_lifetime

print("Customer Lifetime Value (CLV) for Retail customers:", clv)
```

```
Customer Lifetime Value (CLV) for Retail customers: 94.48195654988737
```

```
Start coding or generate with AI.
```

```
import pandas as pd

# Step 1: Filter 2W customers
W_customers = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['VEHICLE_TYPE'] == '2W']


# Step 2: Calculate Total Revenue for each 2W customer
total_revenue = W_customers.groupby('CUSTOMER_ID')['CUSTOMER_FARE'].sum()

# Step 3: Calculate Average Order Value (AOV)
total_orders = W_customers.groupby('CUSTOMER_ID')['ORDER_ID_RO_IN_WINDOW'].count()
aov = total_revenue / total_orders

# Step 4: Calculate Purchase Frequency (PF)
purchase_frequency = total_orders.mean()

# Step 5: Calculate Customer Lifetime (CL)
W_customers['ORDER_DATE'] = pd.to_datetime(W_customers['ORDER_DATE'])
customer_lifetime = W_customers.groupby('CUSTOMER_ID').apply(
    lambda x: (x['ORDER_DATE'].max() - x['ORDER_DATE'].min()).days / 365
).mean()

# Step 6: Calculate CLV
clv = aov.mean() * purchase_frequency * customer_lifetime

print("Customer Lifetime Value (CLV) for 2W customers:", clv)
```

```
Customer Lifetime Value (CLV) for 2W customers: 80.99732695783634
```

```
HCV_and_LCV_customers = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['VEHICLE_TYPE'].isin(['HCV', 'LCV'])]
HCV_and_LCV_customers.head()
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_REGIOI
0	CRN1852411579	17833073	2023-12-01 00:32:14.792000+00:00	2023-12-01 00:43:50.417000+00:00	91	LCV	2.0	
1	CRN1439293796	17833089	2023-12-01 00:43:57.228000+00:00	2023-12-01 00:48:04.433000+00:00	9	HCV	3.0	
2	CRN1456514356	17833116	2023-12-01 01:02:22.553000+00:00	2023-12-01 01:05:56.070000+00:00	1	LCV	1.0	
3	CRN1900903855	17833075	2023-12-01 00:34:05.421000+00:00	2023-12-01 01:07:05.982000+00:00	91	LCV	3.0	
4	CRN1615991541	17833127	2023-12-01 01:07:06.978000+00:00	2023-12-01 01:12:59.590000+00:00	91	LCV	2.0	

5 rows × 21 columns

```
import pandas as pd

# Step 1: Filter SME customers
HCV_and_LCV_customers = df_Repeated_Orders_unique_saved[df_Repeated_Orders_unique_saved['VEHICLE_TYPE'].isin(['HCV', 'LCV'])]

# Step 2: Calculate Total Revenue for each SME customer
total_revenue_HCV_and_LCV_customers = HCV_and_LCV_customers.groupby('CUSTOMER_ID')['CUSTOMER_FARE'].sum()

# Step 3: Calculate Average Order Value (AOV)
total_orders_HCV_and_LCV_customers = HCV_and_LCV_customers.groupby('CUSTOMER_ID')['ORDER_ID_RO_IN_WINDOW'].count()
aov = total_revenue_HCV_and_LCV_customers / total_orders_HCV_and_LCV_customers

# Step 4: Calculate Purchase Frequency (PF)
purchase_frequency = total_orders_HCV_and_LCV_customers.mean()

# Step 5: Calculate Customer Lifetime (CL)
HCV_and_LCV_customers['ORDER_DATE'] = pd.to_datetime(HCV_and_LCV_customers['ORDER_DATE'])
customer_lifetime = HCV_and_LCV_customers.groupby('CUSTOMER_ID').apply(
    lambda x: (x['ORDER_DATE'].max() - x['ORDER_DATE'].min()).days / 365
).mean()

# Step 6: Calculate CLV
clv = aov.mean() * purchase_frequency * customer_lifetime

print("Customer Lifetime Value (CLV) for HCV AND LCV customers:", clv)
```

```
Customer Lifetime Value (CLV) for HCV AND LCV customers: 118.35889962526915
```

📈 Customer Lifetime Value (CLV) Analysis Dashboard

```
!pip install statsmodels
```

```
Collecting statsmodels
  Downloading statsmodels-0.14.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.2 kB)
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.0.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.15.3)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.2.2)
Collecting patsy>=0.5.6 (from statsmodels)
  Downloading patsy-1.0.1-py2.py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (25.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmoc
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (26
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4
Downloading statsmodels-0.14.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (10.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.8/10.8 MB 102.9 MB/s eta 0:00:00
Downloading patsy-1.0.1-py2.py3-none-any.whl (232 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.9/232.9 kB 16.3 MB/s eta 0:00:00
Installing collected packages: patsy, statsmodels
Successfully installed patsy-1.0.1 statsmodels-0.14.4
```

```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

# Load customer_metrics file
df = pd.read_csv("CLV-customer_metrics.csv")

# Convert date columns back to datetime (if needed)
date_cols = ["first_order_date", "last_order_date", "reg_date"]
for col in date_cols:
    df[col] = pd.to_datetime(df[col], errors='coerce')

# Clean column names
df.columns = df.columns.str.strip()

# 1. CLV Segment Distribution (Pie Chart)
fig_pie = px.pie(
    df,
    names='clv_segment',
    title='CLV Segment Distribution',
    hole=0.4,
    color_discrete_sequence=px.colors.qualitative.Set2
)
fig_pie.update_traces(textinfo='percent+label')

# 2. CLV vs Customer Lifespan (Scatter)
fig_scatter_lifespan = px.scatter(
    df,
    x='customer_lifespan_months',
    y='Historical_clv/Predictive_clv',
    color='clv_segment',
    title='CLV vs Customer Lifespan',
    labels={
        'customer_lifespan_months': 'Lifespan (Months)',
        'Historical_clv/Predictive_clv': 'CLV'
    },
    size=df['total_revenue'].apply(lambda x: x if x > 0 else 1),  # ✅ fixed size
    color_discrete_sequence=px.colors.qualitative.Set1
)


# 3. CLV vs Order Count (Scatter with Trend)
fig_scatter_orders = px.scatter(
    df,
    x='order_count',
    y='Historical_clv/Predictive_clv',
    color='clv_segment',
    trendline='ols',
    title='CLV vs Order Count',
    labels={
        'order_count': 'Order Count',
        'Historical_clv/Predictive_clv': 'CLV'
    },
    color_discrete_sequence=px.colors.qualitative.Dark2
)

# 3. Bar Chart: Avg Revenue per CLV Segment
segment_avg = df.groupby('clv_segment')['total_revenue'].mean().reset_index()
fig_bar = px.bar(
    segment_avg,
    x='clv_segment',
    y='total_revenue',
    title='Average Revenue by CLV Segment',
    color='clv_segment',
    color_discrete_sequence=px.colors.sequential.Teal
)

# 4. Customer Lifespan Distribution (Box Plot)
fig_lifespan = px.box(
    df,
    y='customer_lifespan_months',
    title='Customer Lifespan Distribution (Months)',
    points='outliers',
    color_discrete_sequence=['#636EFA']
)

# Save all to one HTML file
with open("clv_dashboard.html", "w") as f:
    f.write(fig_pie.to_html(full_html=False, include_plotlyjs='cdn'))
    f.write(fig_scatter_lifespan.to_html(full_html=False, include_plotlyjs=False))
    f.write(fig_scatter_orders.to_html(full_html=False, include_plotlyjs=False))
    f.write(fig_bar.to_html(full_html=False, include_plotlyjs=False))
```

```
    f.write(fig_lifespan.to_html(full_html=False, include_plotlyjs=False))

print("✅ Dashboard saved as: clv_dashboard.html")
```

```
✅ Dashboard saved as: clv_dashboard.html
```

Promotional Effectiveness

```
customer_metrics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 184833 entries, 0 to 184832
Data columns (total 10 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   CUSTOMER_ID                   184833 non-null  int64
 1   total_revenue                 184833 non-null  float64
 2   order_count                   184833 non-null  int64
 3   first_order_date              184833 non-null  object
 4   last_order_date               184833 non-null  object
 5   reg_date                      184833 non-null  object
 6   customer_lifespan_months      184833 non-null  float64
 7   avg_monthly_revenue           184833 non-null  float64
 8   Historical_clv/Predictive_clv  184833 non-null  float64
 9   clv_segment                   184833 non-null  category
dtypes: category(1), float64(4), int64(2), object(3)
memory usage: 12.9+ MB
```

```
df_Repeated_Orders_unique_saved.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 708620 entries, 0 to 708619
Data columns (total 21 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   ORDER_ID_RO_IN_WINDOW   708620 non-null  object
 1   CUSTOMER_ID             708620 non-null  int64
 2   REG_DATE_CUSTOMERS      708620 non-null  datetime64[ns, UTC]
 3   ORDER_DATE              708620 non-null  datetime64[ns, UTC]
 4   VEHICLE_ID              708620 non-null  int64
 5   VEHICLE_TYPE            708620 non-null  object
 6   GEO_REGION_ID           708620 non-null  float64
 7   GEO_REGION_CITY         708620 non-null  object
 8   MARKET_TYPE             708620 non-null  object
 9   CUSTOMER_FARE           708620 non-null  float64
 10  ADID                    708620 non-null  object
 11  UUID                    708620 non-null  object
 12  REG_DATE                708620 non-null  object
 13  FREQ                    708620 non-null  int64
 14  FREQ_SME_RETAILS        708620 non-null  object
 15  CAMPAIGN_NAME           1636 non-null    object
 16  CITY_NAME_FROM_CAMPAIGN  1300 non-null    object
 17  MOBILE_NUMBER           708620 non-null  int64
 18  REG_MONTH_YEAR          708620 non-null  object
 19  ORDER_MONTH_YEAR        708620 non-null  object
 20  CONVERSION_WINDOW       708620 non-null  int64
dtypes: datetime64[ns, UTC](2), float64(2), int64(5), object(12)
memory usage: 113.5+ MB
```

```
Start coding or generate with AI.
```

```
# from sklearn.cluster import KMeans
# # Example: K-means clustering
# X = customer_metrics[['total_revenue', 'order_count', 'customer_lifespan_months']]
# kmeans = KMeans(n_clusters=5, random_state=42)
# customer_metrics['customer_segment'] = kmeans.fit_predict(X)


# Example: Filter campaigns by customer segments
import pandas as pd

# Merge campaign data with customer data based on CUSTOMER_ID
df_Repeated_Orders_unique_saved = pd.merge(df_Repeated_Orders_unique_saved, customer_metrics, on='CUSTOMER_ID', how='inner')

# Group by campaign name and customer segment to calculate total revenue
campaign_summary = df_Repeated_Orders_unique_saved.groupby(['CAMPAIGN_NAME',])['total_revenue'].sum().reset_index()

# Evaluate campaign performance by segment
campaign_summary
```

	CAMPAIGN_NAME	total_revenue
0	any app to collect docs in same city	67720.94
1	app for courier delivery	274.58
2	app for intercity item delivery	9669.24
3	app for parcel delivery	209.36
4	app for sending items from one place to another	971.25
...	...	...
185	urgent material transfer app download	194.84
186	within city courier like dunzo in pune	193.15
187	पोटर	555551.00
188	पोर्टर	4809.95
189	पोर्टर एप	356.19

190 rows × 2 columns

```
df_Repeated_Orders_unique_saved
```

	ORDER_ID_RO_IN_WINDOW	CUSTOMER_ID	REG_DATE_CUSTOMERS	ORDER_DATE	VEHICLE_ID	VEHICLE_TYPE	GEO_REGION_ID	GEO_I
0	CRN1852411579	17833073	2023-12-01 00:32:14.792000+00:00	2023-12-01 00:43:50.417000+00:00	91	LCV	2.0	
1	CRN1439293796	17833089	2023-12-01 00:43:57.228000+00:00	2023-12-01 00:48:04.433000+00:00	9	HCV	3.0	
2	CRN1456514356	17833116	2023-12-01 01:02:22.553000+00:00	2023-12-01 01:05:56.070000+00:00	1	LCV	1.0	
3	CRN1900903855	17833075	2023-12-01 00:34:05.421000+00:00	2023-12-01 01:07:05.982000+00:00	91	LCV	3.0	
4	CRN1615991541	17833127	2023-12-01 01:07:06.978000+00:00	2023-12-01 01:12:59.590000+00:00	91	LCV	2.0	
...	...	...	...	...	...	...	...	
708615	CRN1089676304	18194980	2023-12-18 21:46:22.938000+00:00	2024-05-30 20:56:12.775000+00:00	97	2W	2.0	
708616	CRN1178580600	17852815	2023-12-01 17:11:28.626000+00:00	2024-05-30 21:27:36.280000+00:00	97	2W	5.0	
708617	CRN1783197932	18157374	2023-12-16 18:36:53.552000+00:00	2024-05-30 21:32:19.506000+00:00	91	LCV	7.0	
708618	CRN1671189673	18395755	2023-12-29 05:54:04.834000+00:00	2024-05-30 21:53:47.897000+00:00	1	LCV	4.0	
708619	CRN1651194576	18175337	2023-12-17 21:03:44.231000+00:00	2024-05-30 22:02:11.899000+00:00	91	LCV	2.0	

708620 rows × 30 columns

```
df_Repeated_Orders_unique_saved.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 708620 entries, 0 to 708619
Data columns (total 30 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   ORDER_ID_RO_IN_WINDOW          708620 non-null  object
 1   CUSTOMER_ID                    708620 non-null  int64
 2   REG_DATE_CUSTOMERS             708620 non-null  datetime64[ns, UTC]
 3   ORDER_DATE                     708620 non-null  datetime64[ns, UTC]
 4   VEHICLE_ID                     708620 non-null  int64
 5   VEHICLE_TYPE                   708620 non-null  object
 6   GEO_REGION_ID                  708620 non-null  float64
 7   GEO_REGION_CITY                708620 non-null  object
 8   MARKET_TYPE                    708620 non-null  object
 9   CUSTOMER_FARE                  708620 non-null  float64
 10  ADID                           708620 non-null  object
 11  UUID                           708620 non-null  object
 12  REG_DATE                       708620 non-null  object
 13  FREQ                           708620 non-null  int64
 14  FREQ_SME_RETAILS               708620 non-null  object
 15  CAMPAIGN_NAME                  1636 non-null    object
 16  CITY_NAME_FROM_CAMPAIGN        1300 non-null    object
 17  MOBILE_NUMBER                  708620 non-null  int64
 18  REG_MONTH_YEAR                 708620 non-null  object
```

```
 19  ORDER_MONTH_YEAR              708620 non-null  object
 20  CONVERSION_WINDOW             708620 non-null  int64
 21  total_revenue                 708620 non-null  float64
 22  order_count                   708620 non-null  int64
 23  first_order_date              708620 non-null  object
 24  last_order_date               708620 non-null  object
 25  reg_date                      708620 non-null  object
 26  customer_lifespan_months      708620 non-null  float64
 27  avg_monthly_revenue           708620 non-null  float64
 28  Historical_clv/Predictive_clv 708620 non-null  float64
 29  clv_segment                   708620 non-null  category
dtypes: category(1), datetime64[ns, UTC](2), float64(6), int64(6), object(15)
memory usage: 157.5+ MB
```

```
df_Repeated_Orders_unique_saved.to_csv('df_Repeated_Orders_unique_saved.csv', index=False)
```

```
# Example: Analyze promotion effectiveness
promotion_effectiveness = df_Repeated_Orders_unique_saved.groupby(['CAMPAIGN_NAME']).agg({
    'total_revenue': 'sum',
    'order_count': 'sum',
    'CONVERSION_WINDOW': 'mean'
}).reset_index()

# Sort promotions by effectiveness
promotion_effectiveness = promotion_effectiveness.sort_values(by='total_revenue', ascending=False)
promotion_effectiveness.head(11)
```

	CAMPAIGN_NAME	total_revenue	order_count	CONVERSION_WINDOW
119	porter apk download	10325588.50	24025	64.451613
120	porter app	4214131.85	14496	69.734797
187	पोटर	555551.00	1373	64.615385
121	porter app download	376064.15	1175	73.942857
117	porter	303126.34	1034	45.354369
182	traspot app	89943.91	49	6.714286
148	porter service	69347.73	87	31.444444
0	any app to collect docs in same city	67720.94	361	96.526316
111	portal driver app	50466.46	121	40.000000
150	porter service in delhi	38973.37	202	64.444444
92	parcel delivery app	31538.84	111	88.210526

```
promotion_effectiveness.to_csv('promotion_effectiveness.csv', index=False)
```

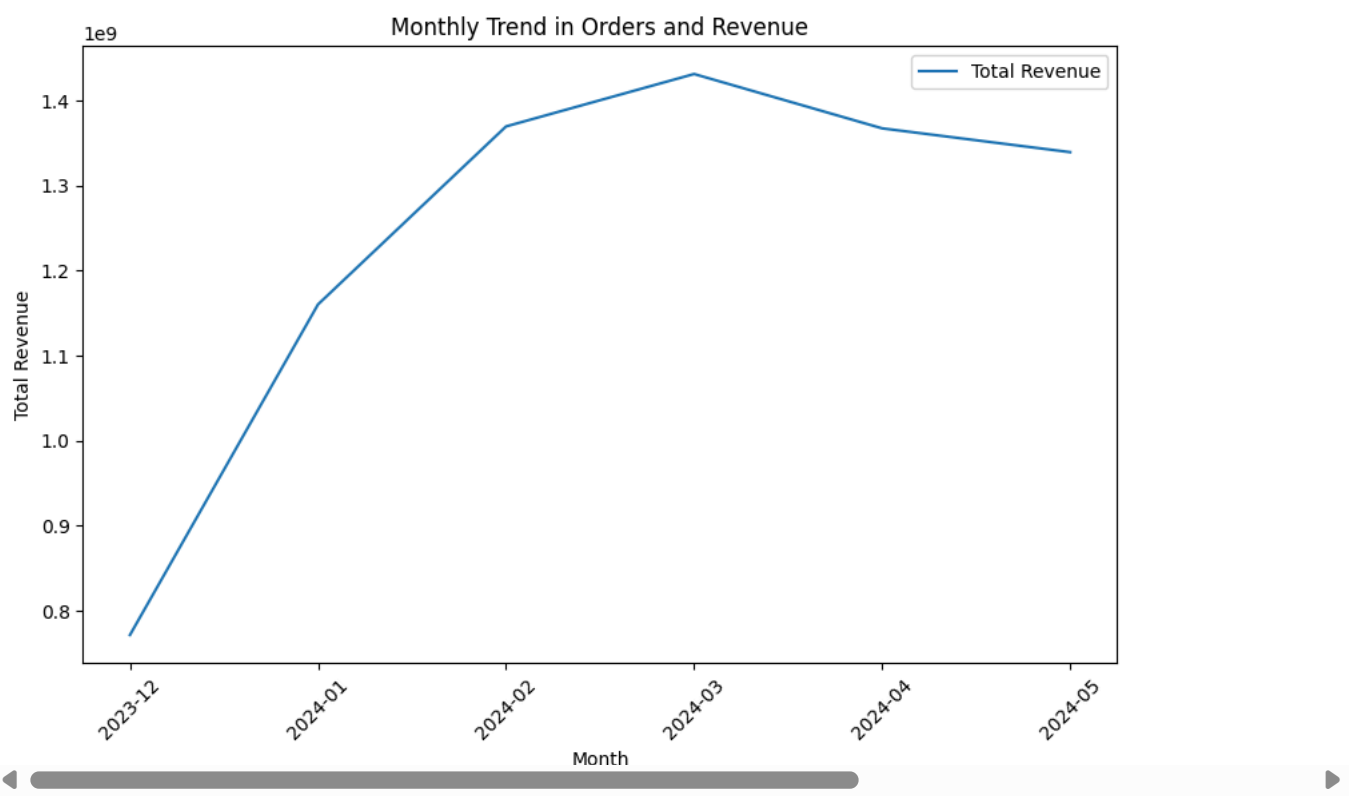
📈 Pattern Analysis and Forecasting

```
df_Repeated_Orders_unique_saved.columns
```

```
Index(['ORDER_ID_RO_IN_WINDOW', 'CUSTOMER_ID', 'REG_DATE_CUSTOMERS',
       'ORDER_DATE', 'VEHICLE_ID', 'VEHICLE_TYPE', 'GEO_REGION_ID',
       'GEO_REGION_CITY', 'MARKET_TYPE', 'CUSTOMER_FARE', 'ADID', 'UUID',
       'REG_DATE', 'FREQ', 'FREQ_SME_RETAILS', 'CAMPAIGN_NAME',
       'CITY_NAME_FROM_CAMPAIGN', 'MOBILE_NUMBER', 'REG_MONTH_YEAR',
       'ORDER_MONTH_YEAR', 'CONVERSION_WINDOW', 'total_revenue', 'order_count',
       'first_order_date', 'last_order_date', 'reg_date',
       'customer_lifespan_months', 'avg_monthly_revenue',
       'Historical_clv/Predictive_clv', 'clv_segment'],
      dtype='object')
```

```
# Convert ORDER_DATE to datetime if it is not already
df_Repeated_Orders_unique_saved['ORDER_DATE'] = pd.to_datetime(df_Repeated_Orders_unique_saved['ORDER_DATE'])

# Aggregate data by month or any other time period (e.g., weekly, daily)
df_monthly_orders = df_Repeated_Orders_unique_saved.groupby(df_Repeated_Orders_unique_saved['ORDER_DATE'].dt.to_period('M')).agg({
    'ORDER_ID_RO_IN_WINDOW': 'count',
    'total_revenue': 'sum'
}).reset_index()

# Plotting the trends
plt.figure(figsize=(10, 6))
plt.plot(df_monthly_orders['ORDER_DATE'].astype(str), df_monthly_orders['total_revenue'], label='Total Revenue')
plt.xlabel('Month')
```

```
plt.ylabel('Total Revenue')
plt.title('Monthly Trend in Orders and Revenue')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```




```
df_monthly_orders
```

	ORDER_DATE	ORDER_ID_RO_IN_WINDOW	total_revenue
0	2023-12	213041	7.711819e+08
1	2024-01	104216	1.160177e+09
2	2024-02	98026	1.369694e+09
3	2024-03	98142	1.431405e+09
4	2024-04	97984	1.367449e+09
5	2024-05	97211	1.339418e+09

```
df_monthly_orders_dataframe = pd.DataFrame(df_monthly_orders)
print(df_monthly_orders_dataframe)
```

```
  ORDER_DATE  ORDER_ID_RO_IN_WINDOW  total_revenue
0    2023-12                 213041   7.711819e+08
1    2024-01                 104216   1.160177e+09
2    2024-02                  98026   1.369694e+09
3    2024-03                  98142   1.431405e+09
4    2024-04                  97984   1.367449e+09
5    2024-05                  97211   1.339418e+09
```

```
df_monthly_orders_dataframe= df_monthly_orders_dataframe.rename(columns={'ORDER_DATE': 'Months'})
```

```
df_monthly_orders_dataframe = df_monthly_orders_dataframe[['Months', 'total_revenue']]
df_monthly_orders_dataframe
```

	Months	total_revenue
0	2023-12	7.711819e+08
1	2024-01	1.160177e+09
2	2024-02	1.369694e+09
3	2024-03	1.431405e+09
4	2024-04	1.367449e+09
5	2024-05	1.339418e+09

```
df_Repeated_Orders_trend = df_Repeated_Orders_unique_saved.groupby(df_Repeated_Orders_unique_saved['ORDER_DATE']).agg({
    'total_revenue': 'sum'
}).reset_index()
```

```
df_Repeated_Orders_trend
```

	ORDER_DATE	total_revenue
0	2023-12-01 00:43:50.417000+00:00	3475.65
1	2023-12-01 00:48:04.433000+00:00	1361.28
2	2023-12-01 01:05:56.070000+00:00	3551.74
3	2023-12-01 01:07:05.982000+00:00	530.42
4	2023-12-01 01:12:59.590000+00:00	4073.26
...	...	...
708557	2024-05-30 20:56:12.775000+00:00	1777.06
708558	2024-05-30 21:27:36.280000+00:00	4272.62
708559	2024-05-30 21:32:19.506000+00:00	2045.72
708560	2024-05-30 21:53:47.897000+00:00	2667.47
708561	2024-05-30 22:02:11.899000+00:00	7739.02

708562 rows × 2 columns

```
df_Repeated_Orders_unique_saved.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 708620 entries, 0 to 708619
Data columns (total 30 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   ORDER_ID_RO_IN_WINDOW           708620 non-null  object
 1   CUSTOMER_ID                     708620 non-null  int64
 2   REG_DATE_CUSTOMERS              708620 non-null  datetime64[ns, UTC]
 3   ORDER_DATE                      708620 non-null  datetime64[ns, UTC]
 4   VEHICLE_ID                      708620 non-null  int64
 5   VEHICLE_TYPE                    708620 non-null  object
 6   GEO_REGION_ID                   708620 non-null  float64
 7   GEO_REGION_CITY                 708620 non-null  object
 8   MARKET_TYPE                     708620 non-null  object
 9   CUSTOMER_FARE                   708620 non-null  float64
 10  ADID                            708620 non-null  object
 11  UUID                            708620 non-null  object
 12  REG_DATE                        708620 non-null  object
 13  FREQ                            708620 non-null  int64
 14  FREQ_SME_RETAILS                708620 non-null  object
 15  CAMPAIGN_NAME                   1636 non-null    object
 16  CITY_NAME_FROM_CAMPAIGN         1300 non-null    object
 17  MOBILE_NUMBER                   708620 non-null  int64
 18  REG_MONTH_YEAR                  708620 non-null  object
 19  ORDER_MONTH_YEAR                708620 non-null  object
 20  CONVERSION_WINDOW               708620 non-null  int64
 21  total_revenue                   708620 non-null  float64
 22  order_count                     708620 non-null  int64
 23  first_order_date                708620 non-null  object
 24  last_order_date                 708620 non-null  object
 25  reg_date                        708620 non-null  object
 26  customer_lifespan_months        708620 non-null  float64
 27  avg_monthly_revenue             708620 non-null  float64
 28  Historical_clv/Predictive_clv   708620 non-null  float64
 29  clv_segment                     708620 non-null  category
dtypes: category(1), datetime64[ns, UTC](2), float64(6), int64(6), object(15)
memory usage: 157.5+ MB
```

```
df_Repeated_Orders_trend['Date'] = df_Repeated_Orders_trend['ORDER_DATE'].dt.strftime('%Y-%m-%d')
df_Repeated_Orders_trend = df_Repeated_Orders_trend[['Date', 'total_revenue']]
df_Repeated_Orders_trend
# 708620
```

	Date	total_revenue
0	2023-12-01	3475.65
1	2023-12-01	1361.28
2	2023-12-01	3551.74
3	2023-12-01	530.42
4	2023-12-01	4073.26
...	...	...
708557	2024-05-30	1777.06
708558	2024-05-30	4272.62
708559	2024-05-30	2045.72
708560	2024-05-30	2667.47
708561	2024-05-30	7739.02

708562 rows × 2 columns

```
df_Repeated_Orders_trend = df_Repeated_Orders_trend.groupby('Date')['total_revenue'].sum()
df_Repeated_Orders_trend
```

Date	total_revenue
2023-12-01	6439986.09
2023-12-02	9480567.91
2023-12-03	7683181.61
2023-12-04	8420819.14
2023-12-05	10043229.09
...	...
2024-05-26	19676398.78
2024-05-27	45235721.69
2024-05-28	46914781.21
2024-05-29	49339660.66
2024-05-30	44654283.59

182 rows × 1 columns

```
df_Repeated_Orders_trend=df_Repeated_Orders_trend.reset_index()
df_Repeated_Orders_trend
```

	Date	total_revenue
0	2023-12-01	6439986.09
1	2023-12-02	9480567.91
2	2023-12-03	7683181.61
3	2023-12-04	8420819.14
4	2023-12-05	10043229.09
...	...	...
177	2024-05-26	19676398.78
178	2024-05-27	45235721.69
179	2024-05-28	46914781.21
180	2024-05-29	49339660.66
181	2024-05-30	44654283.59

182 rows × 2 columns

```
Start coding or generate with AI.
```

```
df_Repeated_Orders_trend.head(11)
```

	Date	total_revenue
0	2023-12-01	6439986.09
1	2023-12-02	9480567.91
2	2023-12-03	7683181.61
3	2023-12-04	8420819.14
4	2023-12-05	10043229.09
5	2023-12-06	13833619.06
6	2023-12-07	17705935.28
7	2023-12-08	22466389.85
8	2023-12-09	22569463.01
9	2023-12-10	10267617.94
10	2023-12-11	21678364.13

```
plt.figure(figsize=(40, 6))
plt.plot(df_Repeated_Orders_trend['Date'], df_Repeated_Orders_trend['total_revenue'])
plt.xlabel('Date')
plt.ylabel('Total Revenue')
plt.title('Total Revenue Over Time')
# Rotate x-axis labels
plt.xticks(rotation=90)
plt.show()
```


```
df_Repeated_Orders_trend
```

	Date	total_revenue
0	2023-12-01	6439986.09
1	2023-12-02	9480567.91
2	2023-12-03	7683181.61
3	2023-12-04	8420819.14
4	2023-12-05	10043229.09
...	...	...
177	2024-05-26	19676398.78
178	2024-05-27	45235721.69
179	2024-05-28	46914781.21
180	2024-05-29	49339660.66
181	2024-05-30	44654283.59

182 rows × 2 columns

```
df = df_Repeated_Orders_trend.reset_index()
df
```

	index	Date	total_revenue
0	0	2023-12-01	6439986.09
1	1	2023-12-02	9480567.91
2	2	2023-12-03	7683181.61
3	3	2023-12-04	8420819.14
4	4	2023-12-05	10043229.09
...	...	...	...
177	177	2024-05-26	19676398.78
178	178	2024-05-27	45235721.69
179	179	2024-05-28	46914781.21
180	180	2024-05-29	49339660.66
181	181	2024-05-30	44654283.59

182 rows x 3 columns

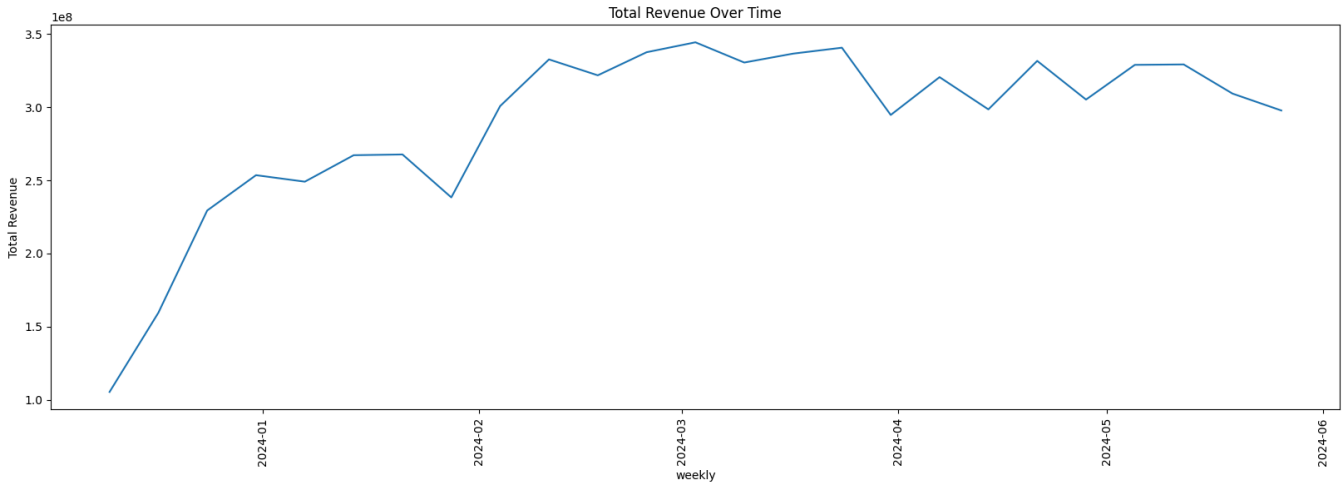
```
import pandas as pd

# Assuming your DataFrame is named 'df'
# Convert 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Resample to weekly frequency (W-SUN = weeks ending on Sunday)
weekly_df = df.resample('W-SUN', on='Date')['total_revenue'].sum().reset_index()

# Rename columns for clarity
weekly_df.columns = ['Week_Ending', 'Total_Revenue']

# Convert 'Total_Revenue' to integer type
weekly_df['Total_Revenue'] = weekly_df['Total_Revenue'].astype(int)

# Ensure 'Week_Ending' remains in datetime format
weekly_df['Week_Ending'] = pd.to_datetime(weekly_df['Week_Ending'])

# Show results
print(weekly_df)
```

```
   Week_Ending  Total_Revenue
0   2023-12-03       23603735
1   2023-12-10      105307073
2   2023-12-17      159454363
3   2023-12-24      229334384
4   2023-12-31      253482341
5   2024-01-07      249037988
6   2024-01-14      267129689
7   2024-01-21      267643213
8   2024-01-28      238284074
9   2024-02-04      300709013
10  2024-02-11      332608704
11  2024-02-18      321736085
12  2024-02-25      337521336
13  2024-03-03      344301596
14  2024-03-10      330478246
15  2024-03-17      336563128
16  2024-03-24      340618011
17  2024-03-31      294645888
18  2024-04-07      320493535
19  2024-04-14      298454206
20  2024-04-21      331602620
21  2024-04-28      305161601
22  2024-05-05      328891910
23  2024-05-12      329174636
24  2024-05-19      309246696
25  2024-05-26      297696649
26  2024-06-02      186144447
```

```
weekly_df=weekly_df.iloc[1:-1]

weekly_df
```

	Week_Ending	Total_Revenue
1	2023-12-10	105307073
2	2023-12-17	159454363
3	2023-12-24	229334384
4	2023-12-31	253482341
5	2024-01-07	249037988
6	2024-01-14	267129689
7	2024-01-21	267643213
8	2024-01-28	238284074
9	2024-02-04	300709013
10	2024-02-11	332608704
11	2024-02-18	321736085
12	2024-02-25	337521336
13	2024-03-03	344301596
14	2024-03-10	330478246
15	2024-03-17	336563128
16	2024-03-24	340618011
17	2024-03-31	294645888
18	2024-04-07	320493535
19	2024-04-14	298454206
20	2024-04-21	331602620
21	2024-04-28	305161601
22	2024-05-05	328891910
23	2024-05-12	329174636
24	2024-05-19	309246696
25	2024-05-26	297696649

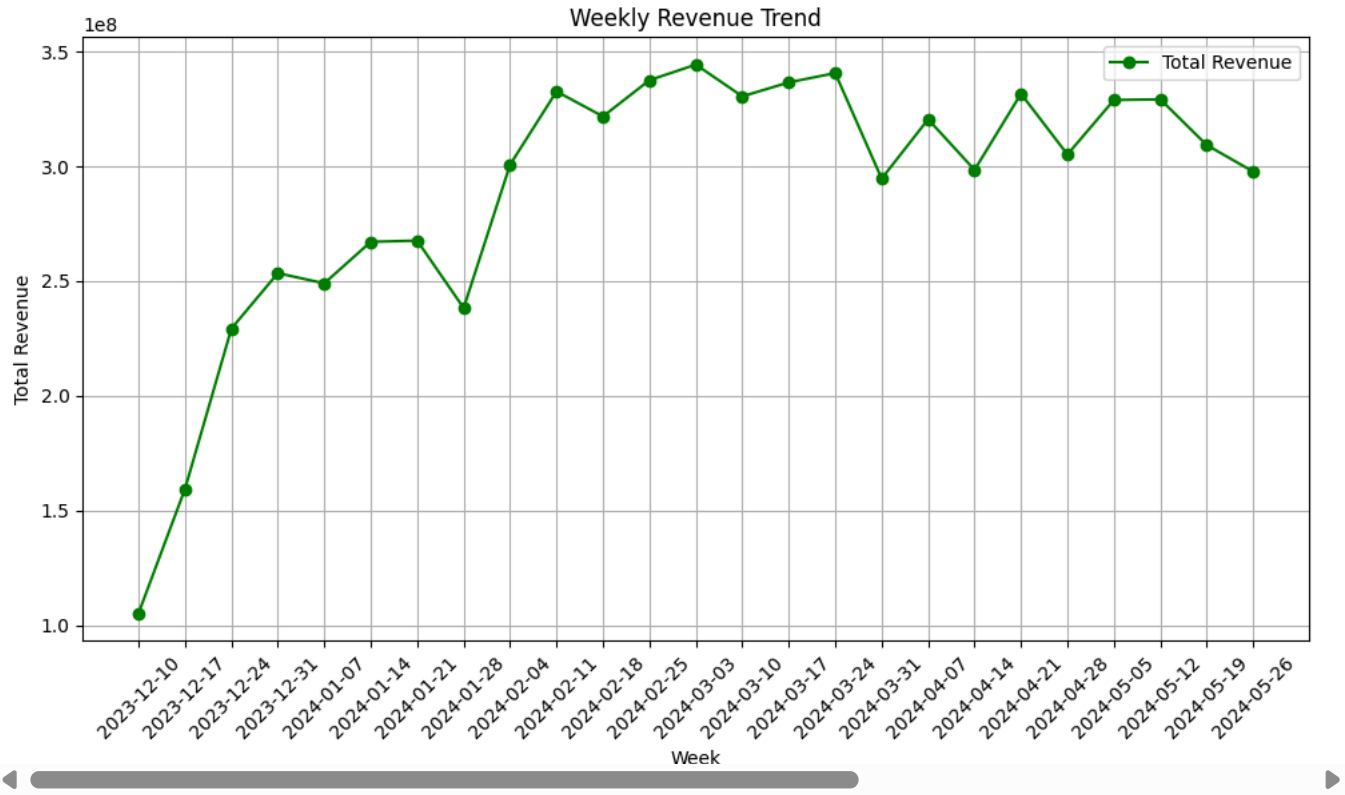
```
plt.figure(figsize=(20, 6))
plt.plot(weekly_df['Week_Ending'], weekly_df['Total_Revenue'])
plt.xlabel('weekly')
plt.ylabel('Total Revenue')
plt.title('Total Revenue Over Time')
# Rotate x-axis labels
plt.xticks(rotation=90)
plt.show()
```




```
df=weekly_df
df
```

	Week_Ending	Total_Revenue
1	2023-12-10	105307073
2	2023-12-17	159454363
3	2023-12-24	229334384
4	2023-12-31	253482341
5	2024-01-07	249037988
6	2024-01-14	267129689
7	2024-01-21	267643213
8	2024-01-28	238284074
9	2024-02-04	300709013
10	2024-02-11	332608704
11	2024-02-18	321736085
12	2024-02-25	337521336
13	2024-03-03	344301596
14	2024-03-10	330478246
15	2024-03-17	336563128
16	2024-03-24	340618011
17	2024-03-31	294645888
18	2024-04-07	320493535
19	2024-04-14	298454206
20	2024-04-21	331602620
21	2024-04-28	305161601
22	2024-05-05	328891910
23	2024-05-12	329174636
24	2024-05-19	309246696
25	2024-05-26	297696649

```
# Plotting the weekly revenue trend
plt.figure(figsize=(10, 6))
plt.plot(df['Week_Ending'].astype(str), df['Total_Revenue'], marker='o', linestyle='-', color='g', label='Total Revenue')
plt.xlabel('Week')
plt.ylabel('Total Revenue')
plt.title('Weekly Revenue Trend')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
df['Week_Ending'] = pd.to_datetime(df['Week_Ending'])
```

```
df['Week_Ending'].dtype
```

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

```
! pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.11/dist-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.0.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.15.3)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.2.2)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (25.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmod
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (20
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4
```

```
!pip install statsmodels
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Assuming 'df' is the DataFrame with your data
# Split the data into training and test sets (last 6 weeks as the test set)
split_point = len(df) - 6

# Split the data
train = df['Total_Revenue'][:split_point]  # First 80% for training
```

```
test = df['Total_Revenue'][split_point:]   # Remaining 20% for testing
# Print the results
print("Training data:")
print(train)
print("\nTesting data:")
print(test)

# Set a smaller seasonal_periods value (e.g., 4 for quarterly data, or a value less than len(train) // 2)
seasonal_periods = 4  # Example: Quarterly seasonality

# Check if seasonal_periods is valid based on the length of the training data
if seasonal_periods > len(train) // 2:
    seasonal_periods = len(train) // 2  # Limit seasonal_periods to half the training data length
    print(f"Seasonal periods adjusted to {seasonal_periods} due to insufficient training data.")

# Fit the Additive Holt-Winters model
additive_model = ExponentialSmoothing(train, trend='add', seasonal='add', seasonal_periods=seasonal_periods).fit()
additive_forecast = additive_model.forecast(len(test))

# Fit the Multiplicative Holt-Winters model
multiplicative_model = ExponentialSmoothing(train, trend='add', seasonal='mul', seasonal_periods=seasonal_periods).fit()
multiplicative_forecast = multiplicative_model.forecast(len(test))

# Calculate RMSE and MAE for the Additive model
additive_rmse = np.sqrt(mean_squared_error(test, additive_forecast))
additive_mae = mean_absolute_error(test, additive_forecast)

# Calculate RMSE and MAE for the Multiplicative model
multiplicative_rmse = np.sqrt(mean_squared_error(test, multiplicative_forecast))
multiplicative_mae = mean_absolute_error(test, multiplicative_forecast)

# Print the results
print(f"Additive Model RMSE: {additive_rmse:.4f}")
print(f"Additive Model MAE: {additive_mae:.4f}")
print(f"Multiplicative Model RMSE: {multiplicative_rmse:.4f}")
print(f"Multiplicative Model MAE: {multiplicative_mae:.4f}")

# Automatically select the better model based on RMSE and MAE
if additive_rmse < multiplicative_rmse and additive_mae < multiplicative_mae:
    print("Using Additive model for forecasting")
    selected_trend = 'add'
    selected_seasonal = 'add'
else:
    print("Using Multiplicative model for forecasting")
    selected_trend = 'add'
    selected_seasonal = 'mul'

# Convert the 'Week_Ending' column to datetime with the correct format if it's not already datetime
# df['Week_Ending'] = pd.to_datetime(df['Week_Ending']) # No need to convert if it's already datetime

# Reset the index to bring 'Week_Ending' back as a column
df = df.reset_index()

# Convert to datetime if not already
df['Week_Ending'] = pd.to_datetime(df['Week_Ending'])

# Set 'Week_Ending' as the index
df.set_index('Week_Ending', inplace=True)

# Step 1: Fit the selected Holt-Winters model
model = ExponentialSmoothing(
    df['Total_Revenue'],
    trend=selected_trend,           # Selected trend
    seasonal=selected_seasonal,     # Selected seasonality
    seasonal_periods=seasonal_periods            # 12 months in a year
).fit()

# Step 2: Forecast the next 12 weeks (adjust if needed)
forecast = model.forecast(steps=12)

# Step 3: Create a new DataFrame to store results
forecast_index = pd.date_range(start=df.index[-1] + pd.DateOffset(weeks=1), periods=12, freq='W-SUN')
forecast_df = pd.DataFrame({'Total_Revenue': forecast}, index=forecast_index)

# Step 4: Plot original data and forecast
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Total_Revenue'], label='Historical Data', marker='o')
plt.plot(forecast_df.index, forecast_df['Total_Revenue'], label='Forecast', marker='o', color='orange')
plt.title('Holt-Winters Forecast')
plt.xlabel('Week')  # Changed to 'Week' for weekly data
plt.ylabel('Total Revenue')
plt.legend()
```

```
plt.grid()
plt.show()

# Step 5: Display forecasted values
print("Forecasted Values:")
print(forecast_df)
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.11/dist-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.0.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.15.3)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.2.2)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (25.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->stats
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=
Training data:
1     105307073
2     159454363
3     229334384
4     253482341
5     249037988
6     267129689
7     267643213
8     238284074
9     300709013
10    332608704
11    321736085
12    337521336
13    344301596
14    330478246
15    336563128
16    340618011
17    294645888
18    320493535
19    298454206
Name: Total_Revenue, dtype: int64

Testing data:
20    331602620
21    305161601
22    328891910
23    329174636
24    309246696
25    297696649
Name: Total_Revenue, dtype: int64
Additive Model RMSE: 29678179.0720
Additive Model MAE: 24234232.0908
Multiplicative Model RMSE: 30423092.9721
Multiplicative Model MAE: 25123188.8939
Using Additive model for forecasting
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/holtwinters/model.py:918: ConvergenceWarning:

Optimization failed to converge. Check mle_retvals.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/holtwinters/model.py:918: ConvergenceWarning:

Optimization failed to converge. Check mle_retvals.

/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning:

No frequency information was provided, so inferred frequency W-SUN will be used.
```

```
# Step 6: Save historical data and forecast to CSV
# Concatenate historical data and forecast data
combined_df = pd.concat([df, forecast_df])

# Save to CSV
combined_df.to_csv('total_revenue_forecast.csv')
print("Data saved to 'total_revenue_forecast.csv'")
```

```
Data saved to 'total_revenue_forecast.csv'
```

Start coding or generate with AI.

Start coding or generate with AI.


```
from scipy.stats import zscore

# Compute Z-scores for CUSTOMER_FARE (order value equivalent)
df_Repeated_Orders['Fare_Z_Score'] = zscore(df_Repeated_Orders['CUSTOMER_FARE'])

# Define threshold for anomaly detection (values above 3 are considered outliers)
threshold = 3
```

```
# Identify anomalies
anomalies = df_Repeated_Orders[df_Repeated_Orders['Fare_Z_Score'].abs() > threshold]

# Display anomalies
print("Anomalous Orders:")
print(anomalies[['CUSTOMER_ID', 'CUSTOMER_FARE', 'Fare_Z_Score']])
```

```
Forecasted Values:
Anomalous Orders:
            Total Revenue
            CUSTOMER_ID  CUSTOMER_FARE  Fare_Z_Score
2024-06-02  3.179191e+08
4           17833127     1784.08        3.484436
2024-06-09  3.212176e+08
70          17833772     1736.43        3.369997
2024-06-16  3.183409e+08
74          17833841     2656.63        5.580001
2024-06-23  3.202432e+08
105         17834111     1987.73        3.973533
2024-06-30  3.355712e+08
147         17834397     2624.88        5.503748
2024-07-07  3.388697e+08
...                      ...            ...
2024-07-14  3.359930e+08
708053      18039769     4618.00        10.290537
2024-07-21  3.378953e+08
708142      17950019     1788.00        3.493850
2024-07-28  3.532233e+08
708200      18088645     1757.00        3.419399
2024-08-04  3.562218e+08
708355      17962326     1832.00        3.599523
2024-08-11  3.536452e+08
708601      17951662     2122.00        4.296003
2024-08-18  3.555474e+08
[11738 rows x 3 columns]
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

🔥 Cohort Analysis and Visualization

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset (Assuming df_Repeated_Orders_unique_saved contains 'CUSTOMER_ID' and 'ORDER_DATE')
df = df_Repeated_Orders_unique_saved.copy()

# Convert ORDER_DATE to datetime format
df['ORDER_DATE'] = pd.to_datetime(df['ORDER_DATE'])

# Extract Cohort Month (first purchase month of each customer)
df['Cohort_Month'] = df.groupby('CUSTOMER_ID')['ORDER_DATE'].transform('min').dt.to_period('M')

# Extract Order Year and Month
df['Order_Year'] = df['ORDER_DATE'].dt.year
df['Order_Month'] = df['ORDER_DATE'].dt.month

# Create Cohort Index (time difference from first purchase)
df['Cohort_Index'] = (df['Order_Year'] - df['Cohort_Month'].dt.year) * 12 + (df['Order_Month'] - df['Cohort_Month'].dt.month) +

# Create Cohort Table (Pivot)
cohort_counts = df.groupby(['Cohort_Month', 'Cohort_Index'])['CUSTOMER_ID'].nunique().unstack()

# Calculate Retention Rates
cohort_sizes = cohort_counts.iloc[:, 0]  # First column (month 1) is the cohort size
retention_matrix = cohort_counts.divide(cohort_sizes, axis=0)

# Calculate Churn Rates (1 - Retention Rate)
churn_matrix = 1 - retention_matrix

# 1️⃣ Heatmap for Retention Trends
plt.figure(figsize=(12, 6))
sns.heatmap(retention_matrix, annot=True, fmt=".0%", cmap="Blues")
plt.title("Customer Retention by Cohort")
plt.ylabel("Cohort Month")
plt.xlabel("Cohort Index (Months)")
plt.show()

# 2️⃣ Line Plot for Retention Rate Trends
plt.figure(figsize=(10, 5))
for index in retention_matrix.index:
    plt.plot(retention_matrix.columns, retention_matrix.loc[index], marker="o", label=str(index))

plt.title("Customer Retention Over Time")
plt.xlabel("Cohort Index (Months)")
```

```
plt.ylabel("Retention Rate")
plt.legend(title="Cohort Month", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```