💻 Question No: 03

⚙️ Setup

- Ensure the Python kernel has the necessary libraries: `pandas`, `seaborn`, `numpy`, `kmeans`, `matplotlib` and `lets-plot`
- Ensure the `online_retail.csv` file is in the `data` folder.

In [151...
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

In [152...
```
# Load the dataset
df = pd.read_excel('D:/Data Science for Marketing-I/dataset/Online Retail.xlsx')
```

In [153...
```
df.head()
```

Out[153…

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Un Kingc
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	Un Kingc
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingc

i. Why is data cleaning important in the dataset, and what impact does filtering out transactions with Quantity <= 0 and UnitPrice <= 0, missing CustomerID, and removing observations from December 2011 in the InvoiceDate column have on the dataset's validity for analysis?

In [154…

```
df = df.dropna(subset=["CustomerID"])  # Remove missing CustomerID
```

💡 Filtering out transactions with Quantity <= 0 and UnitPrice <= 0: This removes erroneous or refund transactions that can distort the sales data.

In [155…

```
df = df[(df["Quantity"] > 0) & (df["UnitPrice"] > 0)]  # Filter out invalid transac
```

💡 Removing missing CustomerID: Transactions without CustomerID cannot be assigned to a customer, making them useless for customer behavior analysis.

In [156…

```
df = df[~df["InvoiceDate"].astype(str).str.startswith("2011-12")] # Remove Dec 2011
```

💡 Removing December 2011 transactions: If these transactions are incomplete, they could bias the analysis, especially in time-based predictions.

ii. What role does the newly created Sales based on Quantity and UnitPrice columns play in customer behavior analysis?

In [157…

```
df["Sales"] = df["Quantity"] * df["UnitPrice"]
```

In [158…

```
df.head()
```

Out[158…

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	Un Kingd
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingd
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	Un Kingd
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingd
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	Un Kingd

💡 The Sales column (calculated as Quantity * UnitPrice) helps in understanding:

- Which customers generate the highest revenue
- How much customers typically spend in each transaction
- Trends in customer spending over time.

iii. Create orderDF by grouping transactions based on CustomerID and InvoiceNo, and visualize customer behavior based on purchase frequency and total sales.

In [159…

```
orders_df = df.groupby(['CustomerID', 'InvoiceNo']).agg({
    'Sales': 'sum',  # Use string instead of built-in function
```

```
    'InvoiceDate': 'max'  # Use string instead of built-in function
}).reset_index()
```

In [160…

```
orders_df
```

Out[160…

	CustomerID	InvoiceNo	Sales	InvoiceDate
0	12346.0	541431	77183.60	2011-01-18 10:01:00
1	12347.0	537626	711.79	2010-12-07 14:57:00
2	12347.0	542237	475.39	2011-01-26 14:30:00
3	12347.0	549222	636.25	2011-04-07 10:43:00
4	12347.0	556201	382.52	2011-06-09 13:01:00
...	...	...	...	...
17749	18283.0	578262	313.65	2011-11-23 13:27:00
17750	18283.0	579673	223.61	2011-11-30 12:59:00
17751	18287.0	554065	765.28	2011-05-22 10:39:00
17752	18287.0	570715	1001.32	2011-10-12 10:23:00
17753	18287.0	573167	70.68	2011-10-28 09:29:00

17754 rows × 4 columns

In [161…

```
def groupby_mean(x):
    return x.mean()

def groupby_count(x):
    return x.count()

def purchase_duration(x):
    return (x.max() - x.min()).days

def avg_frequency(x):
    return (x.max() - x.min()).days/x.count()

groupby_mean.__name__ = 'avg'
groupby_count.__name__ = 'count'
purchase_duration.__name__ = 'purchase_duration'
avg_frequency.__name__ = 'purchase_frequency'
```

In [162…

```
summary_df = orders_df.reset_index().groupby('CustomerID').agg({
    'Sales': ['min','max', 'sum', groupby_mean, groupby_count],
    'InvoiceDate': ['min', 'max', purchase_duration, avg_frequency]
})
```

In [163…

```
summary_df
```

Out[163...

	Sales							
	min	max	sum	avg	count	min	max	purchas
CustomerID								
12346.0	77183.60	77183.60	77183.60	77183.600000	1	2011-01-18 10:01:00	2011-01-18 10:01:00	
12347.0	382.52	1294.32	4085.18	680.863333	6	2010-12-07 14:57:00	2011-10-31 12:25:00	
12348.0	227.44	892.80	1797.24	449.310000	4	2010-12-16 19:09:00	2011-09-25 13:13:00	
12349.0	1757.55	1757.55	1757.55	1757.550000	1	2011-11-21 09:51:00	2011-11-21 09:51:00	
12350.0	334.40	334.40	334.40	334.400000	1	2011-02-02 16:01:00	2011-02-02 16:01:00	
...	...	...	...	...	...	...	...	
18280.0	180.60	180.60	180.60	180.600000	1	2011-03-07 09:52:00	2011-03-07 09:52:00	
18281.0	80.82	80.82	80.82	80.820000	1	2011-06-12 10:53:00	2011-06-12 10:53:00	
18282.0	100.21	100.21	100.21	100.210000	1	2011-08-05 13:35:00	2011-08-05 13:35:00	
18283.0	1.95	313.65	1886.88	125.792000	15	2011-01-06 14:14:00	2011-11-30 12:59:00	
18287.0	70.68	1001.32	1837.28	612.426667	3	2011-05-22 10:39:00	2011-10-28 09:29:00	

4297 rows × 9 columns

In [164...

```
summary_df.columns = ['_'.join(col) for col in summary_df.columns]
```

In [165...

```
summary_df
```

Out[165…

	Sales_min	Sales_max	Sales_sum	Sales_avg	Sales_count	InvoiceDate_min
CustomerID						
12346.0	77183.60	77183.60	77183.60	77183.600000	1	2011-01-18 10:01:00
12347.0	382.52	1294.32	4085.18	680.863333	6	2010-12-07 14:57:00
12348.0	227.44	892.80	1797.24	449.310000	4	2010-12-16 19:09:00
12349.0	1757.55	1757.55	1757.55	1757.550000	1	2011-11-21 09:51:00
12350.0	334.40	334.40	334.40	334.400000	1	2011-02-02 16:01:00
...	...	...	...	...	...	...
18280.0	180.60	180.60	180.60	180.600000	1	2011-03-07 09:52:00
18281.0	80.82	80.82	80.82	80.820000	1	2011-06-12 10:53:00
18282.0	100.21	100.21	100.21	100.210000	1	2011-08-05 13:35:00
18283.0	1.95	313.65	1886.88	125.792000	15	2011-01-06 14:14:00
18287.0	70.68	1001.32	1837.28	612.426667	3	2011-05-22 10:39:00

4297 rows × 9 columns

In [166…

```
summary_df.shape
```

Out[166…

```
(4297, 9)
```

In [167…

```
summary_df = summary_df.loc[summary_df['InvoiceDate_purchase_duration'] > 0]
```

In [168…

```
summary_df.shape
```

Out[168…

```
(2692, 9)
```

In [169…

```
summary_df
```

Out[169…

CustomerID	Sales_min	Sales_max	Sales_sum	Sales_avg	Sales_count	InvoiceDate_min
12347.0	382.52	1294.32	4085.18	680.863333	6	2010-12-07 14:57:00
12348.0	227.44	892.80	1797.24	449.310000	4	2010-12-16 19:09:00
12352.0	120.33	840.30	2506.04	313.255000	8	2011-02-16 12:33:00
12356.0	58.35	2271.62	2811.43	937.143333	3	2011-01-18 09:50:00
12359.0	547.50	2876.85	6372.58	1593.145000	4	2011-01-12 12:43:00
...	...	...	...	...	...	...
18270.0	111.95	171.20	283.15	141.575000	2	2011-03-18 12:41:00
18272.0	340.72	753.66	2710.70	542.140000	5	2011-04-07 09:35:00
18273.0	51.00	102.00	153.00	76.500000	2	2011-03-27 11:22:00
18283.0	1.95	313.65	1886.88	125.792000	15	2011-01-06 14:14:00
18287.0	70.68	1001.32	1837.28	612.426667	3	2011-05-22 10:39:00

2692 rows × 9 columns

In [170…

```
summary_df.groupby('Sales_count').count()['Sales_min']
```

Out[170…

	Sales_count
2	745
3	512
4	380
5	227
6	171
7	132
8	97
9	60
10	45
11	54
12	47
13	29
14	19
15	24
16	13
17	15
18	11
19	15
20	10
21	10
22	4
23	5
24	4
25	8
26	3
27	5
28	5
29	4
30	3
31	2
32	1
33	1
34	1
35	2
36	1
37	3
38	1
39	1
40	1
43	1
45	2
47	3
51	1
53	1
54	1
55	1
57	1
59	1
63	1
70	1
84	1
88	1
91	1
93	1
120	1

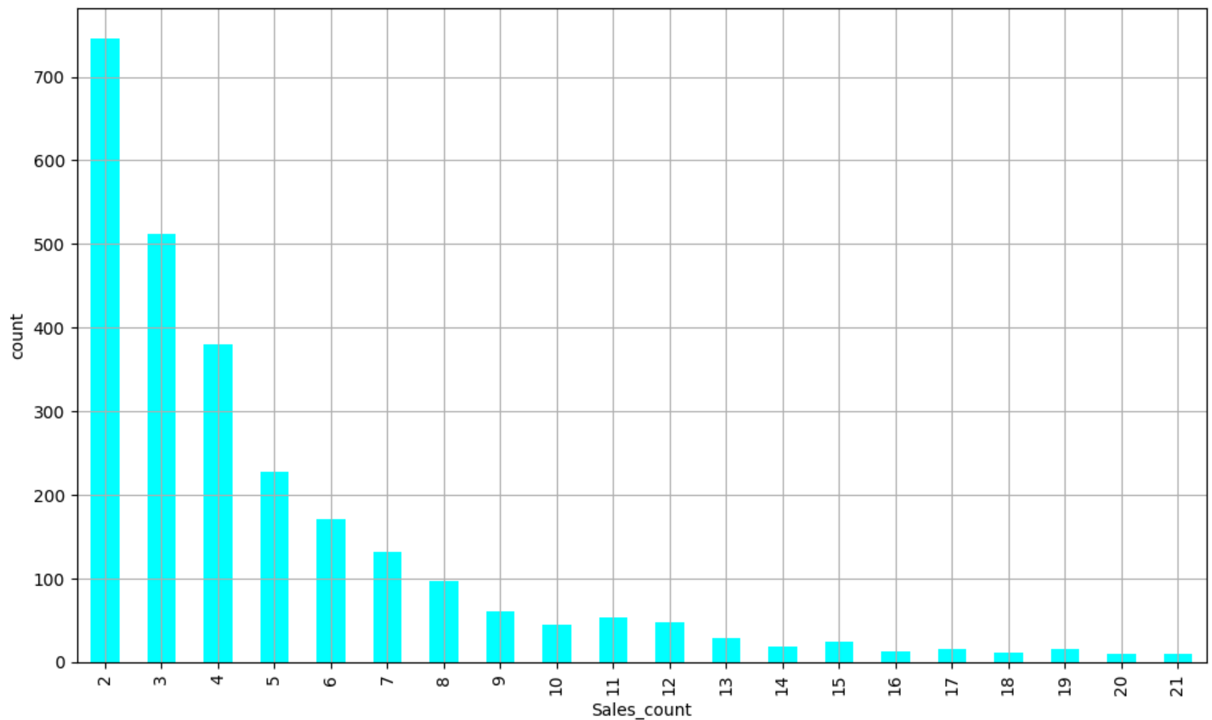
```
192      1
200      1
Name: Sales_min, dtype: int64
```

In [171…

```
ax = summary_df.groupby('Sales_count').count()['Sales_avg'][:20].plot(
    kind='bar',
    color='cyan',
    figsize=(12,7),
    grid=True
)

ax.set_ylabel('count')

plt.show()
```



In [172…

```
summary_df['Sales_count'].describe()
```

Out[172…

```
count    2692.000000
mean        5.969911
std         8.867340
min         2.000000
25%         2.000000
50%         4.000000
75%         6.000000
max       200.000000
Name: Sales_count, dtype: float64
```

In [173…

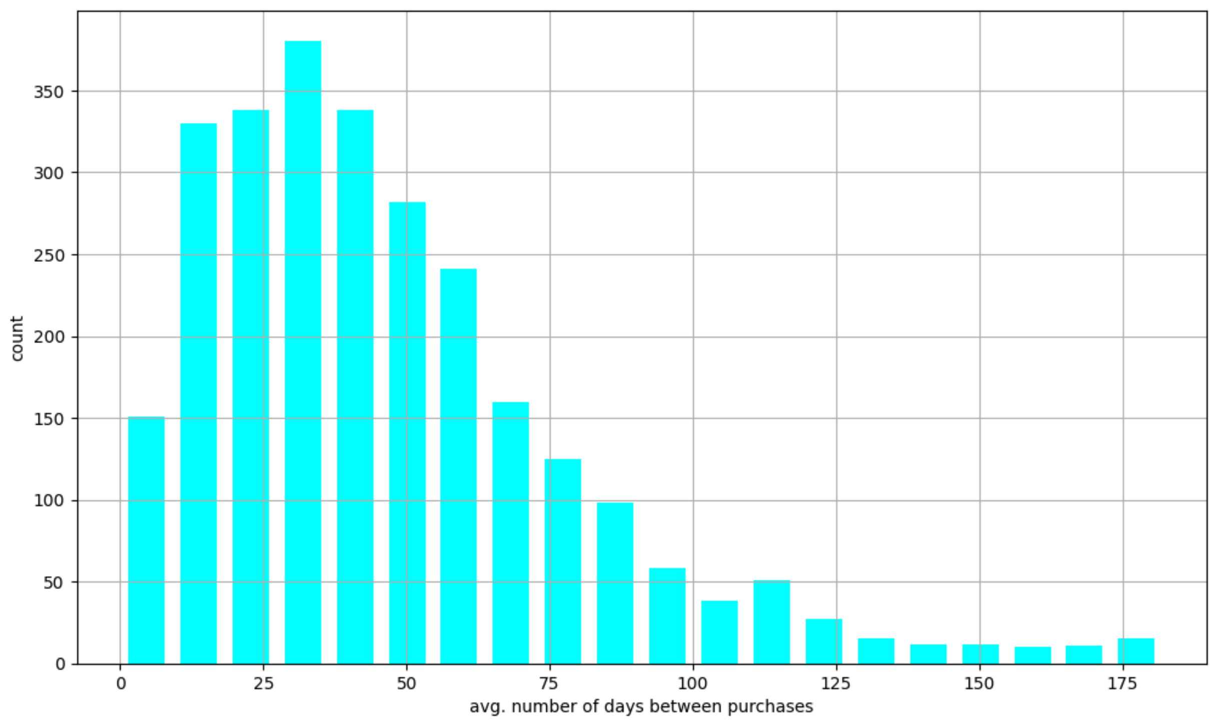
```
summary_df['Sales_avg'].describe()
```

Out[173…
```
count     2692.000000
mean       391.494935
std        465.724765
min          3.450000
25%        197.802708
50%        306.043333
75%        444.524000
max      14844.766667
Name: Sales_avg, dtype: float64
```

In [174…
```python
ax = summary_df['InvoiceDate_purchase_frequency'].hist(
    bins=20,
    color='cyan',
    rwidth=0.7,
    figsize=(12,7)
)

ax.set_xlabel('avg. number of days between purchases')
ax.set_ylabel('count')

plt.show()
```



In [175…
```python
summary_df['InvoiceDate_purchase_frequency'].describe()
```

Out[175…
```
count    2692.000000
mean       47.003043
std        32.395136
min         0.029412
25%        23.500000
50%        40.500000
75%        62.333333
max       182.000000
Name: InvoiceDate_purchase_frequency, dtype: float64
```

In [176...
```
summary_df['InvoiceDate_purchase_duration'].describe()
```

Out[176...
```
count    2692.000000
mean      199.720282
std       107.816559
min         1.000000
25%       107.000000
50%       209.000000
75%       296.000000
max       364.000000
Name: InvoiceDate_purchase_duration, dtype: float64
```

💡

- Grouping by CustomerID & InvoiceNo helps understand purchase frequency and spending behavior.
- Bar charts show how often customers buy.
- Histograms reveal how much time passes between purchases

iv. Prepare data for model building by creating five quarters, using four quarters as features and the latest quarter as the response variable.

In [177...
```
clv_freq = '3ME'
```

In [215...
```
data_df = orders_df.reset_index().groupby([
    'CustomerID',
    pd.Grouper(key='InvoiceDate', freq=clv_freq)
]).agg({
    'Sales': ['sum', groupby_mean, groupby_count],
})
```

In [179...
```
data_df.columns = ['_'.join(col) for col in data_df.columns]
data_df.columns
```

Out[179...
```
Index(['Sales_sum', 'Sales_avg', 'Sales_count'], dtype='object')
```

In [180...
```
data_df = data_df.reset_index()
```

In [181...
```
data_df.head(10)
```

Out[181…

	CustomerID	InvoiceDate	Sales_sum	Sales_avg	Sales_count
0	12346.0	2011-03-31	77183.60	77183.600	1
1	12347.0	2010-12-31	711.79	711.790	1
2	12347.0	2011-03-31	475.39	475.390	1
3	12347.0	2011-06-30	1018.77	509.385	2
4	12347.0	2011-09-30	584.91	584.910	1
5	12347.0	2011-12-31	1294.32	1294.320	1
6	12348.0	2010-12-31	892.80	892.800	1
7	12348.0	2011-03-31	227.44	227.440	1
8	12348.0	2011-06-30	367.00	367.000	1
9	12348.0	2011-09-30	310.00	310.000	1

In [182…

```
date_month_map = {
    str(x)[:10]: 'M_%s' % (i+1) for i, x in enumerate(
        sorted(data_df.reset_index()['InvoiceDate'].unique(), reverse=True)
    )
}
```

In [183…

```
data_df['M'] = data_df['InvoiceDate'].apply(lambda x: date_month_map[str(x)[:10]])
```

In [184…

```
date_month_map
```

Out[184…

```
{'2011-12-31': 'M_1',
 '2011-09-30': 'M_2',
 '2011-06-30': 'M_3',
 '2011-03-31': 'M_4',
 '2010-12-31': 'M_5'}
```

In [185…

```
data_df.head(10)
```

Out[185…

	CustomerID	InvoiceDate	Sales_sum	Sales_avg	Sales_count	M
0	12346.0	2011-03-31	77183.60	77183.600	1	M_4
1	12347.0	2010-12-31	711.79	711.790	1	M_5
2	12347.0	2011-03-31	475.39	475.390	1	M_4
3	12347.0	2011-06-30	1018.77	509.385	2	M_3
4	12347.0	2011-09-30	584.91	584.910	1	M_2
5	12347.0	2011-12-31	1294.32	1294.320	1	M_1
6	12348.0	2010-12-31	892.80	892.800	1	M_5
7	12348.0	2011-03-31	227.44	227.440	1	M_4
8	12348.0	2011-06-30	367.00	367.000	1	M_3
9	12348.0	2011-09-30	310.00	310.000	1	M_2

- Building Sample Set

In [186…

```
features_df = pd.pivot_table(
    data_df.loc[data_df['M'] != 'M_1'],
    values=['Sales_sum', 'Sales_avg', 'Sales_count'],
    columns='M',
    index='CustomerID'
)
```

In [187…

```
features_df.columns = ['_'.join(col) for col in features_df.columns]
```

In [188…

```
features_df.shape
```

Out[188…

```
(3616, 12)
```

In [189…

```
features_df.head(10)
```

Out[189...

CustomerID	Sales_avg_M_2	Sales_avg_M_3	Sales_avg_M_4	Sales_avg_M_5	Sales_count_M_2
12346.0	NaN	NaN	77183.600	NaN	NaN
12347.0	584.91	509.385	475.390	711.79	1.0
12348.0	310.00	367.000	227.440	892.80	1.0
12350.0	NaN	NaN	334.400	NaN	NaN
12352.0	316.25	NaN	312.362	NaN	2.0
12353.0	NaN	89.000	NaN	NaN	NaN
12354.0	NaN	1079.400	NaN	NaN	NaN
12355.0	NaN	459.400	NaN	NaN	NaN
12356.0	NaN	481.460	2271.620	NaN	NaN
12358.0	484.86	NaN	NaN	NaN	1.0

In [190...

```
features_df = features_df.fillna(0)
```

In [191...

```
features_df.head()
```

Out[191...

CustomerID	Sales_avg_M_2	Sales_avg_M_3	Sales_avg_M_4	Sales_avg_M_5	Sales_count_M_2
12346.0	0.00	0.000	77183.600	0.00	0.0
12347.0	584.91	509.385	475.390	711.79	1.0
12348.0	310.00	367.000	227.440	892.80	1.0
12350.0	0.00	0.000	334.400	0.00	0.0
12352.0	316.25	0.000	312.362	0.00	2.0

In [192...

```
response_df = data_df.loc[
    data_df['M'] == 'M_1',
    ['CustomerID', 'Sales_sum']
]
```

In [193...

```
response_df.columns = ['CustomerID', 'CLV_'+clv_freq]
```

In [194...

```
response_df.shape
```

Out[194...

```
(2406, 2)
```

In [195…

```
response_df.head(10)
```

Out[195…

	CustomerID	CLV_3ME
5	12347.0	1294.32
10	12349.0	1757.55
14	12352.0	311.73
20	12356.0	58.35
21	12357.0	6207.67
25	12359.0	2876.85
28	12360.0	1043.78
33	12362.0	2119.85
37	12364.0	299.06
41	12370.0	739.28

In [196…

```
sample_set_df = features_df.merge(
    response_df,
    left_index=True,
    right_on='CustomerID',
    how='left'
)
```

In [197…

```
sample_set_df.shape
```

Out[197…

```
(3616, 14)
```

In [198…

```
sample_set_df.head(10)
```

Out[198…

	Sales_avg_M_2	Sales_avg_M_3	Sales_avg_M_4	Sales_avg_M_5	Sales_count_M_2	Sales
NaN	0.00	0.000	77183.600	0.00	0.0	
5.0	584.91	509.385	475.390	711.79	1.0	
NaN	310.00	367.000	227.440	892.80	1.0	
NaN	0.00	0.000	334.400	0.00	0.0	
14.0	316.25	0.000	312.362	0.00	2.0	
NaN	0.00	89.000	0.000	0.00	0.0	
NaN	0.00	1079.400	0.000	0.00	0.0	
NaN	0.00	459.400	0.000	0.00	0.0	
20.0	0.00	481.460	2271.620	0.00	0.0	
NaN	484.86	0.000	0.000	0.00	1.0	

In [199…

```
sample_set_df = sample_set_df.fillna(0)
```

In [200…

```
sample_set_df.head()
```

Out[200…

	Sales_avg_M_2	Sales_avg_M_3	Sales_avg_M_4	Sales_avg_M_5	Sales_count_M_2	Sales
NaN	0.00	0.000	77183.600	0.00	0.0	
5.0	584.91	509.385	475.390	711.79	1.0	
NaN	310.00	367.000	227.440	892.80	1.0	
NaN	0.00	0.000	334.400	0.00	0.0	
14.0	316.25	0.000	312.362	0.00	2.0	

In [201…

```
sample_set_df['CLV_'+clv_freq].describe()
```

Out[201…

```
count     3616.000000
mean       511.558520
std       2371.743293
min          0.000000
25%          0.000000
50%          0.000000
75%        458.662500
max      68012.350000
Name: CLV_3ME, dtype: float64
```

💡 split data into five quarters, using four quarters as predictors and the last quarter as the target. This helps model future customer spending (CLV prediction).

v. Build a regression model using the linear regression algorithm and interpret the model summary.

In [202…

```
from sklearn.model_selection import train_test_split
```

In [203…

```
target_var = 'CLV_'+clv_freq
all_features = [x for x in sample_set_df.columns if x not in ['CustomerID', target_
```

In [204…

```
x_train, x_test, y_train, y_test = train_test_split(
    sample_set_df[all_features],
    sample_set_df[target_var],
    test_size=0.3
)
```

- Linear Regression Model

In [205…

```
from sklearn.linear_model import LinearRegression

# Try these models as well
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
```

In [206…

```
reg_fit = LinearRegression()
```

In [207…

```
reg_fit.fit(x_train, y_train)
```

Out[207…

```
▾ LinearRegression
LinearRegression()
```

In [208…

```
reg_fit.intercept_
```

Out[208…

```
np.float64(25.916656930902832)
```

In [209…

```
coef = pd.DataFrame(list(zip(all_features, reg_fit.coef_)))
coef.columns = ['feature', 'coef']

coef
```

Out[209…

	feature	coef
0	Sales_avg_M_2	0.370523
1	Sales_avg_M_3	-0.493296
2	Sales_avg_M_4	-0.198953
3	Sales_avg_M_5	-0.439027
4	Sales_count_M_2	96.603223
5	Sales_count_M_3	10.416535
6	Sales_count_M_4	-105.026721
7	Sales_count_M_5	-11.839619
8	Sales_sum_M_2	0.207873
9	Sales_sum_M_3	0.355720
10	Sales_sum_M_4	0.231876
11	Sales_sum_M_5	0.899390

💡 Linear Regression to predict future customer value. The model learns relationships between past purchases and future spending.

vi.Predict values for the training and test data, and evaluate its performance using all possible metrics.

In [210…
```
from sklearn.metrics import r2_score, median_absolute_error
```

In [211…
```
train_preds =  reg_fit.predict(x_train)
test_preds = reg_fit.predict(x_test)
```

- R-Squared

In [212…
```
print('In-Sample R-Squared: %0.4f' % r2_score(y_true=y_train, y_pred=train_preds))
print('Out-of-Sample R-Squared: %0.4f' % r2_score(y_true=y_test, y_pred=test_preds)
```

```
In-Sample R-Squared: 0.7627
Out-of-Sample R-Squared: 0.2238
```

- Median Absolute Error

In [213…
```
print('In-Sample MAE: %0.4f' % median_absolute_error(y_true=y_train, y_pred=train_p
print('Out-of-Sample MSE: %0.4f' % median_absolute_error(y_true=y_test, y_pred=test
```

```
In-Sample MAE: 211.1019
Out-of-Sample MSE: 211.0535
```

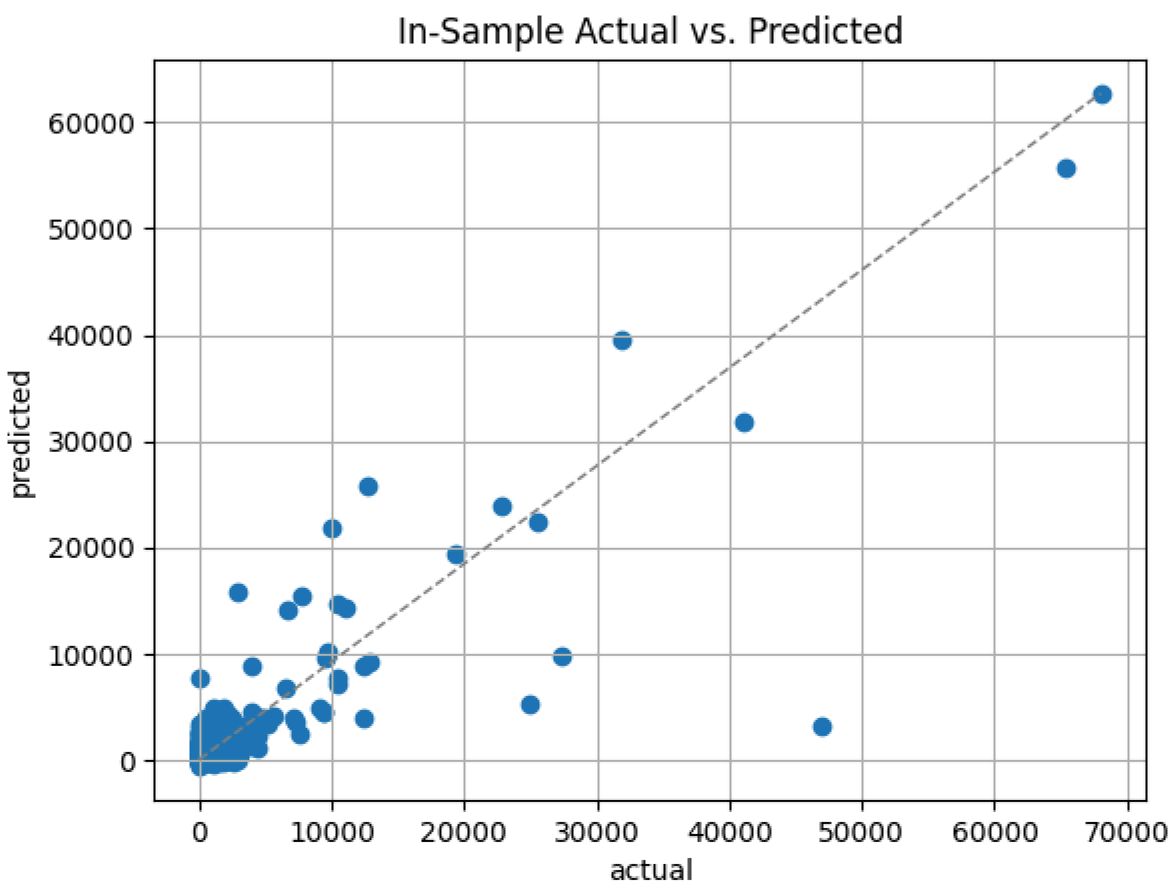
- Scatter Plot

In [214…

```
plt.scatter(y_train, train_preds)
plt.plot([0, max(y_train)], [0, max(train_preds)], color='gray', lw=1, linestyle='-

plt.xlabel('actual')
plt.ylabel('predicted')
plt.title('In-Sample Actual vs. Predicted')
plt.grid()

plt.show()
```



💡

- R^2 Score: Measures how well predictions match actual values.
- MAE (Median Absolute Error): Measures average prediction error.
- Scatter Plot: Compares actual vs. predicted values.