5. Customer Segmentation Dataset (Unsupervised Learning - Clustering)

- TC Antony

Datasets:

Customer_ID	Age	Income	Location	Number_of_Active_Policies	Total_Premium_Paid	Claim_Frequency	Policy_Upgrades
bdd640fb-0667-4ad1-9c80-317fa3b1799d	58	40039.59	Johnberg	3	12999.70	2	0
bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	61	153320.19	New Roberttown	5	5260.00	6	0
815ef6d1-3b8f-4a18-b7f8-a88b17fc695a	19	36865.14	New Jamesside	2	25762.41	0	1
96da1dac-72ff-4d2a-b86e-cbe06b65a6a4	63	136979.20	Lisatown	5	21556.47	7	2
b2b9437a-28df-4ec4-8e4a-2bbdc241330b	69	176474.06	Petersonberg	2	35208.83	5	2

[6]
```
df5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Customer_ID               1000 non-null   object 
 1   Age                       1000 non-null   int64  
 2   Income                    1000 non-null   float64
 3   Location                  1000 non-null   object 
 4   Number_of_Active_Policies  1000 non-null   int64  
 5   Total_Premium_Paid        1000 non-null   float64
 6   Claim_Frequency           1000 non-null   int64  
 7   Policy_Upgrades           1000 non-null   int64  
dtypes: float64(2), int64(4), object(2)
memory usage: 62.6+ KB
```

[7]
```
# Identify categorical and numerical columns
categorical_columns = df5.select_dtypes(include=['object']).columns.tolist()
numerical_columns = df5.select_dtypes(include=['int64', 'float64']).columns.tolist()

print("Categorical Columns:", categorical_columns)
print("Numerical Columns:", numerical_columns)
```

```
Categorical Columns: ['Customer_ID', 'Location']
Numerical Columns: ['Age', 'Income', 'Number_of_Active_Policies', 'Total_Premium_Paid', 'Claim_Frequency', 'Policy_Upgrades']
```

	0
Customer_ID	1000
Age	63
Income	1000
Location	963
Number_of_Active_Policies	5
Total_Premium_Paid	1000
Claim_Frequency	11
Policy_Upgrades	4

dtype: int64

Primary vs. Cross-Check Columns

Primary Metrics:

Age, Income, Total_Premium_Paid, Claim_Frequency, Policy_Upgrades.

Cross-Check Analysis:

Income vs. Total_Premium_Paid:

1.Do higher-income customers pay more in premiums?

2.Age vs. Claim_Frequency: Do older customers file more claims?

3.Number_of_Active_Policies vs. Total_Premium_Paid: Do more policies result in higher payments?

4.Claim_Frequency vs. Policy_Upgrades: Are frequent claimers upgrading their policies more?

1) Histogram Distribution of

Income,

Claim_Frequency,

Total_Premium_Paid

Identifies spread and skewness.

2) Boxplot Outlier detection in

Income and

Claim_Frequency

Detects anomalies.

3) Scatter Plot

Income vs. Total_Premium_Paid,

Age vs. Claim_Frequency

Shows **correlation between numerical variables**.

4) Bar Chart

Number_of_Active_Policies vs. Policy_Upgrades

Highlights policy trends.

5) Line Chart

Income vs. Total_Premium_Paid

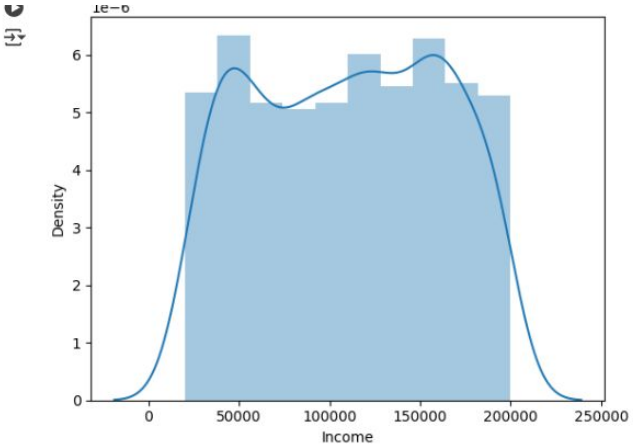
over time Identifies payment trends

6) Heatmap Correlation among all numerical variables Highlights strong relationships.

1) Histogram Distribution

Use Histplot or Distplot to Identify the Spread to Check Normal/Gausian Distribution : to befor model train, machine understand a Data is Normal Distribution

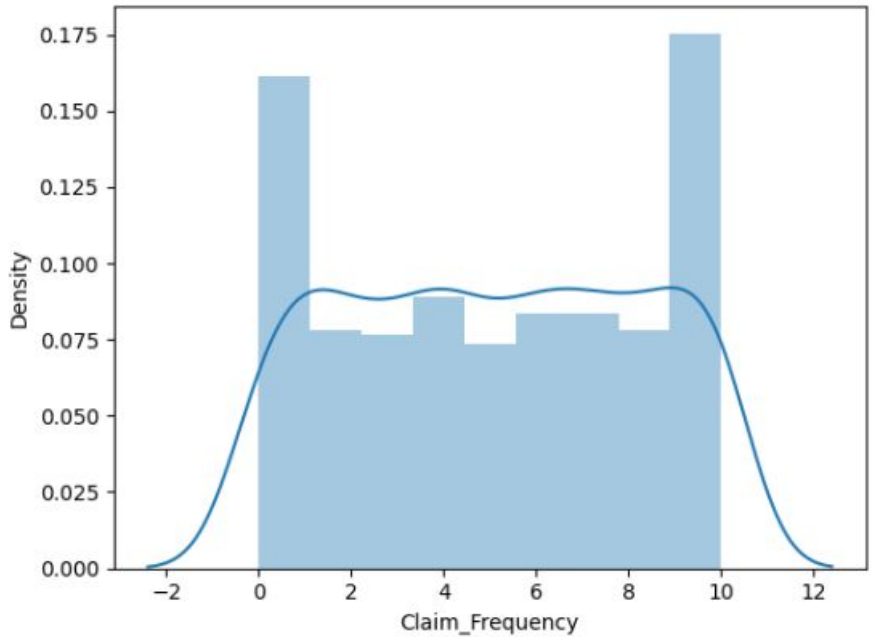
i) Income



Skewness is -0.028644209702153704 its almost near to 0 and its negative value -> its Almost a Normal Distribution

ii) Claim_Frequency

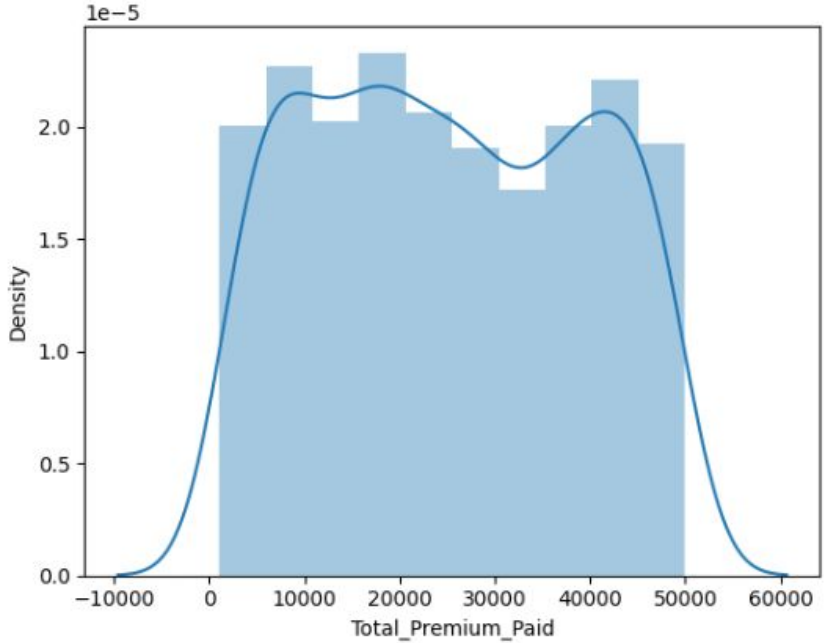
```
-0.014915399033445515
```



Skewness is -0.014915399033445515 its almost near to Zero its Negative Value almost Flaten Bell Curve its a Normal Distribution

iii) Total_Premium_Paid

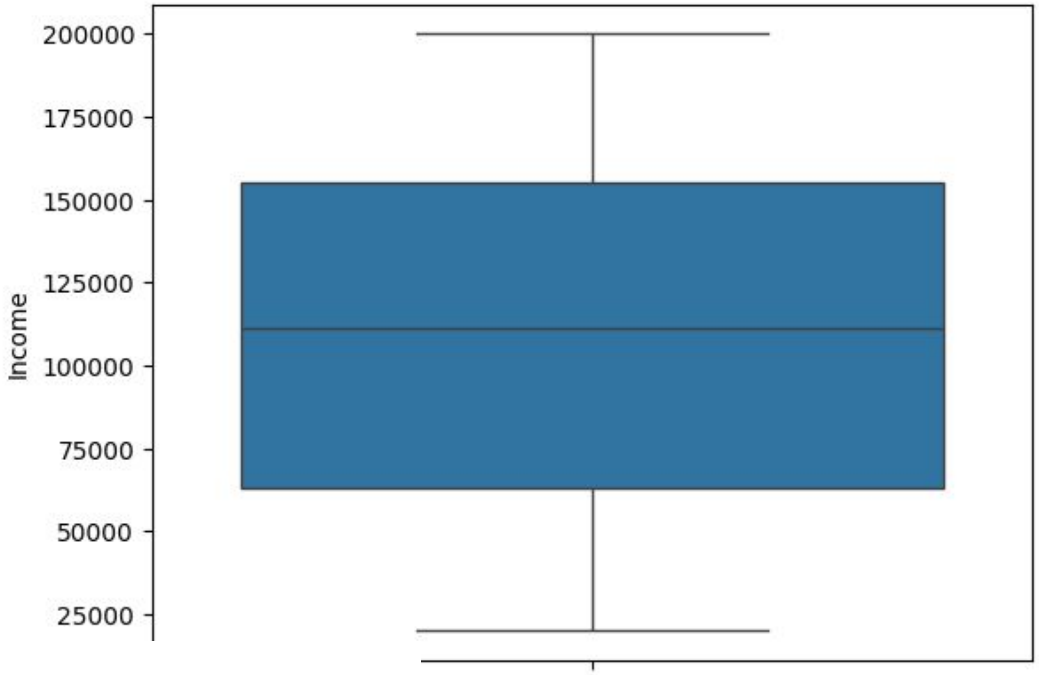
```
sns.distplot(df5['Total_Premium_Paid'])
0.060513725074797 65
```



Skewness is 0.060513725074797 65 and its a Almost Normal Distribution

2) Boxplot

i) Income



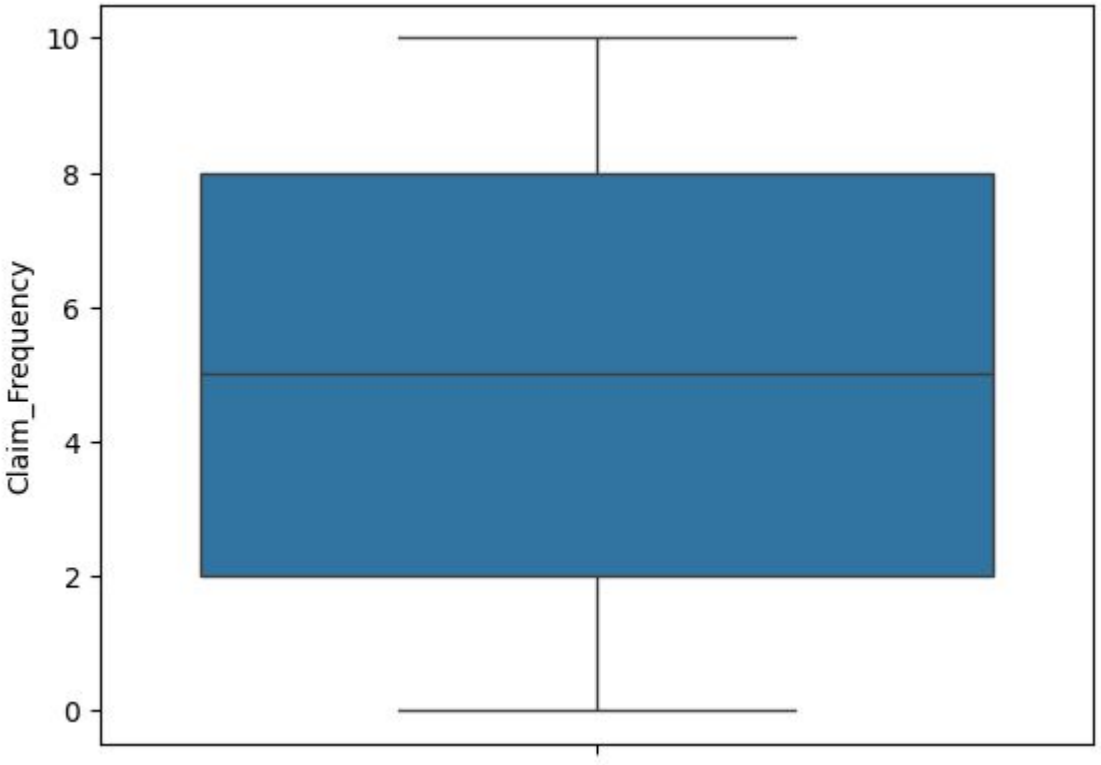
No Outlier Detected :-

Minimum is almost 70k and Maximum is 155k

and Lower Boundary is 25k and Upper Boundary is 200k

>>>>Most of the Income is 70 to 155K

ii) Claim_Frequency



No Outlier Detected:

Minimum Claim Frequency is 2

and Maximum Claim Frequency is 8

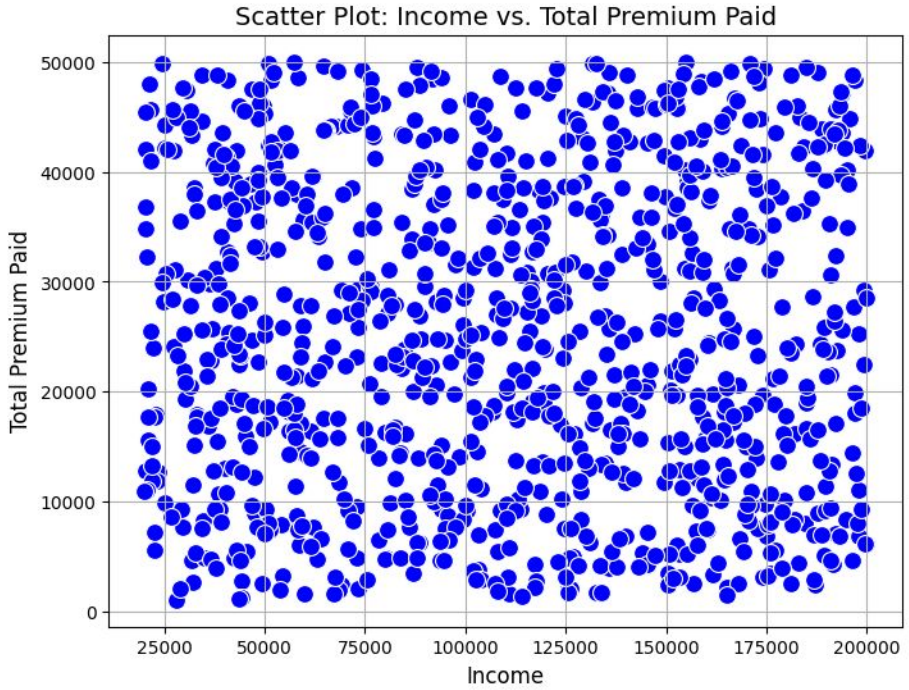
Lower Boundary is 0 and

Upper Boundary is 10

Most of the Claim is 2 to 8

3) Scatter Plot

i) Income vs. Total_Premium_Paid

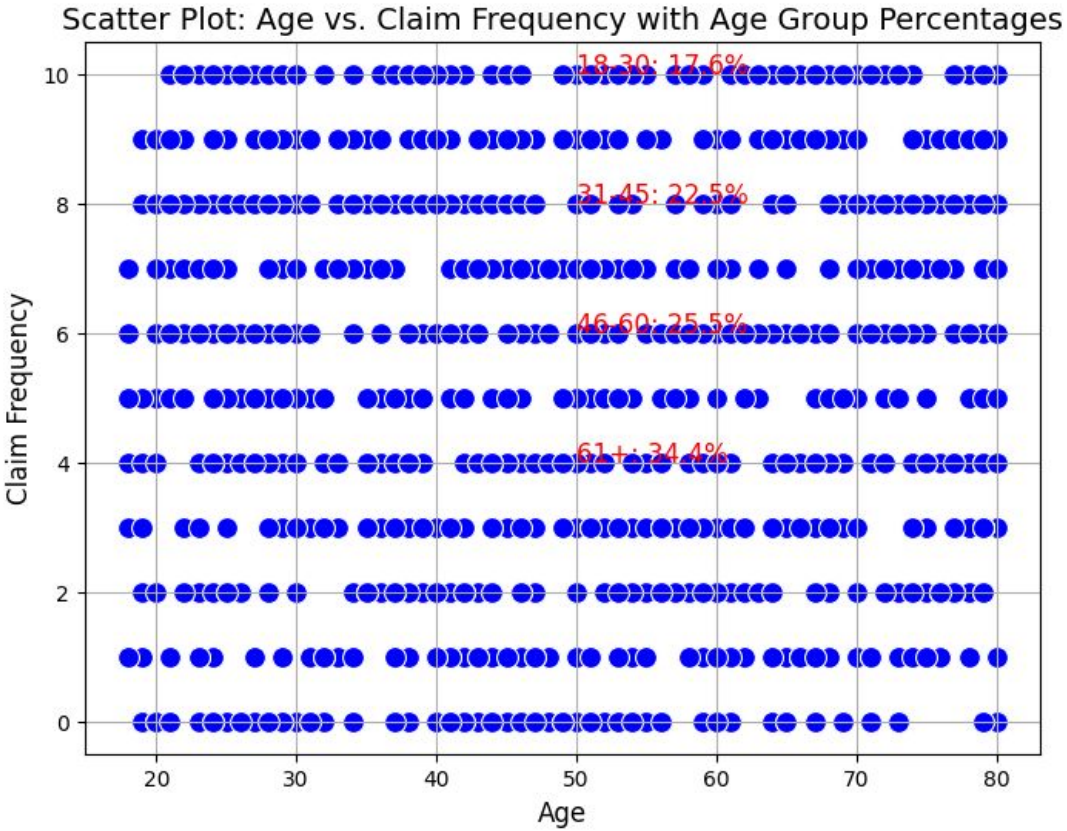


Insights:-

Scatter Plot Represent there is **no correlation between Income and Total Premium Paid**,

Low Income Persons also Uses Maximum Premium Amount and **High Income Persons also Uses Minimum Premium Amount**

ii) Age vs. Claim_Frequency



B/W Age Groups and Claim Frequency Slightly NO Correlation because all Age groups Cover All Frequency

Age groups:

1) 18 - 30 Ages -> 17% Claims,

2) 31 - 45 Ages -> 22% Claims,

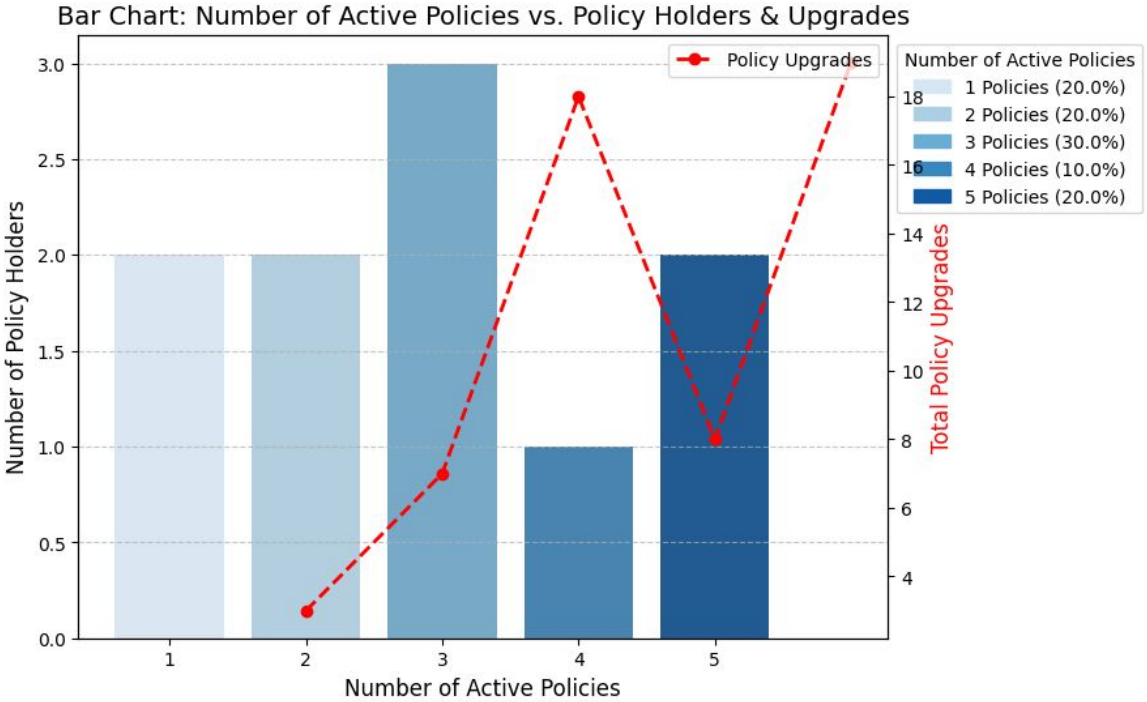
3) 46 - 60 Ages -> 25% Claims,

4) 60+ Ages -> 34% Claims

60+ Ages was Highest Claims Happens

4) Bar Chart

i) Number_of_Active_Policies vs. Policy_Upgrades



Insights:

1) 3 Policies Holders almost 30% -> 3 Policies Holders are Highest Combare to other

2) 4 Policies Holders are Very Low 10% only they hold

3) 1 to 3 Policies Holders are almost 70%

Policies Upgrade:

1) Policies Upgrades Start From 2 Policies holders

2) some 2 Policies Holders upgrader to 3 Policies, 4 Policies and 5 Policies

Recommendations:

1) 4 Policies Holders are very Low so Focus on 3 Policies Holder to give Offers to Upgrade to 4 Plicies Holders

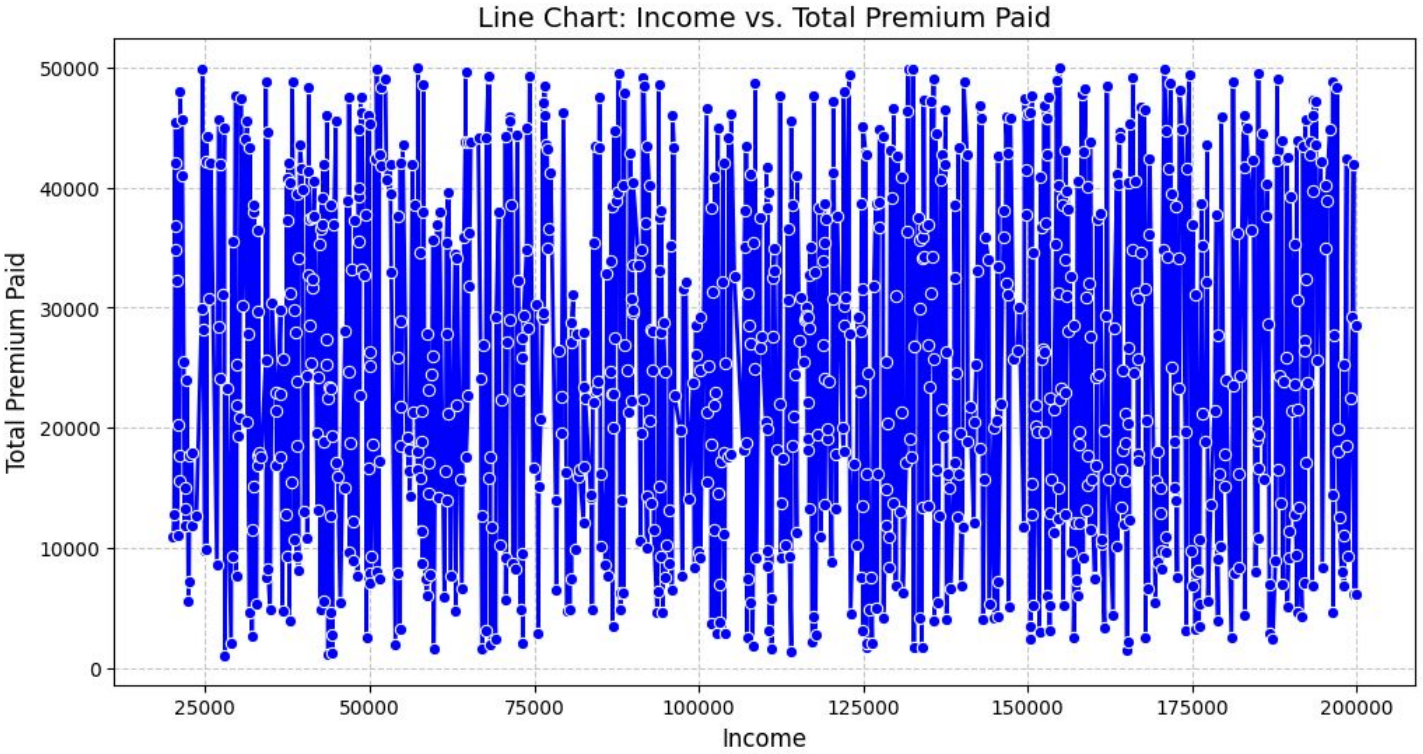
2) 1 Policy Holders does not upgrade to 2 Policies Holders

3) so Focusing on 1 Policy Holder to give best plan and Offers to Upgrade to Multiple Policy Holders

4) Most of the 2 Policies Holders are Upgrades to 3 Policies Holders so get Ideas form 2 to 3 Holders

5) Line Chart

Income vs. Total_Premium_Paid



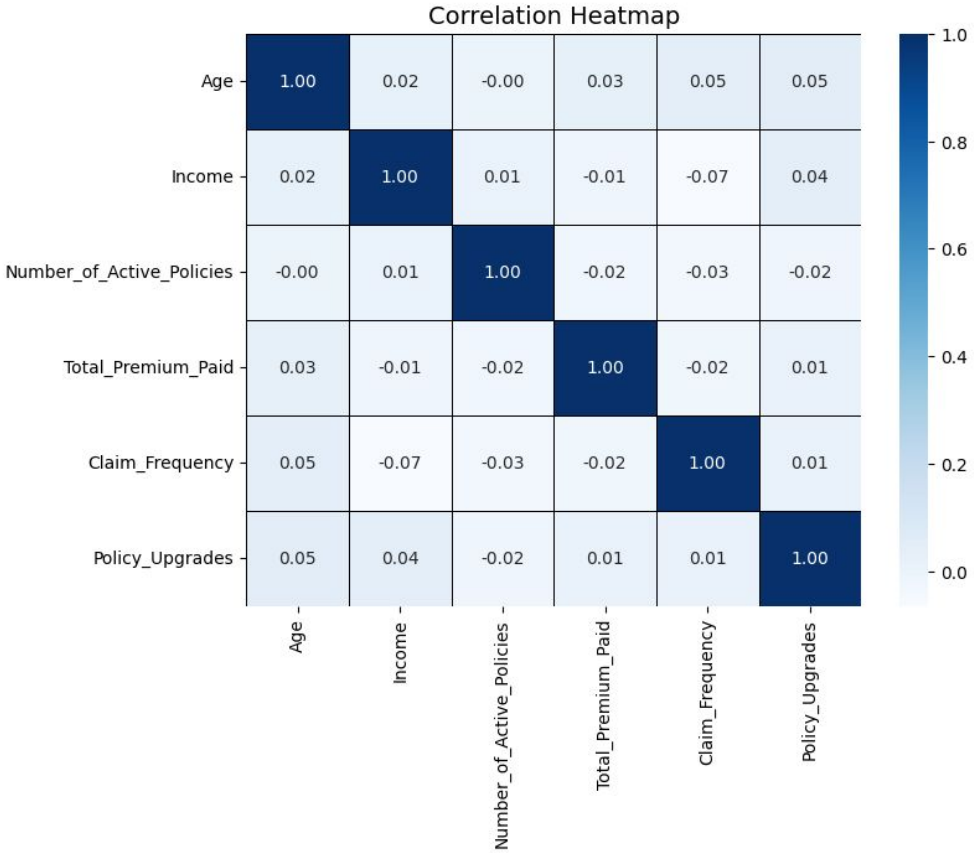
Positive Correlation:

There appears to be a generally positive correlation between income and total premium paid. As income increases, the total premium paid also tends to increase. This is logical, as higher-income individuals typically have more disposable income to allocate towards insurance or other premium-based services.

Potential Outliers:

There are a **few instances where individuals with relatively lower incomes are paying very high premiums**, and conversely, **some high-income individuals are paying lower premiums**. These could be outliers due to unique circumstances or might warrant further investigation.

6) Heatmap for All Numeric Columns



All Columns are No Correlated:

Some Columns are Slightly Posative and Negative Correlated

SQL

SQL - Create Table

```python
import sqlite3
import pandas as pd

# File name of the uploaded CSV
csv_filename = "/content/df5-customer_segmentation_dataset.csv"  # Replace this with the uploaded file name

# Load CSV into a pandas DataFrame
df = pd.read_csv(csv_filename)

# Connect to SQLite database (or create a new one)
conn = sqlite3.connect("example.db")
cursor = conn.cursor()

# Write DataFrame to SQLite table
table_name = "df5"  # Specify your table name
df.to_sql(table_name, conn, if_exists="replace", index=False)

print(f"Table '{table_name}' created in SQLite database.")

```

```
Table 'df5' created in SQLite database.
```

1) Retrieve All Data

```
query="Select * From df5"

result = pd.read_sql_query(query,conn)
result
```

	Unnamed: 0	Customer_ID	Age	Income	Location	Number_of_Active_Policies	Total_Premium_Paid	Claim_Frequency	Policy_Upgrades
0	0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	58	40039.59	Johnberg	3	12999.70	2	0
1	1	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	61	153320.19	New Roberttown	5	5260.00	6	0
2	2	815ef6d1-3b8f-4a18-b7f8-a88b17fc695a	19	36865.14	New Jamesside	2	25762.41	0	1
3	3	96da1dac-72ff-4d2a-b86e-cbe06b65a6a4	63	136979.20	Lisatown	5	21556.47	7	2
4	4	b2b9437a-28df-4ec4-8e4a-2bbdc241330b	69	176474.06	Petersonberg	2	35208.83	5	2
...	...	...	...	...	...	...	...	...	...

2) Find Customers with More than 3 Active Policies

```
query="Select Customer_ID,Age,Location,Number_of_Active_Policies From df5 where Number_of_Active_Policies>=3;"

result = pd.read_sql_query(query,conn)
result
```

	Customer_ID	Age	Location	Number_of_Active_Policies
0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	58	Johnberg	3
1	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	61	New Roberttown	5
2	96da1dac-72ff-4d2a-b86e-cbe06b65a6a4	63	Lisatown	5
3	562b0f79-c374-49ee-b50b-ea63371ecd7b	27	North Noahstad	3
4	43b7a3a6-9a8d-4a03-980d-7b71d8f56413	40	Lake Stephenville	5
...	...	...	...	...
613	e83bcc84-a609-4ad0-b114-64aa52f69081	67	Lake Jessica	3
614	730bc657-b4cb-4613-9d39-e9c217584f23	71	Christopherfurt	3
615	e98965f2-2bbf-4e35-84ac-b57d0b3d6dc9	58	Arianaberg	5

Total 618 Users Have More than 3 Policies Out of 1000 Users

3) Find Customers Who Paid More Than $25000 in Total Premium

```
Select
Customer_ID,Age,Location,Number_of_Active_Policies,Total_Premium_P
aid From df5 where Total_Premium_Paid>=25000 Order By
Total_Premium_Paid ASC;
```

	Customer_ID	Age	Location	Number_of_Active_Policies	Total_Premium_Paid
0	3a6c7da6-57f7-471f-a978-cc731440adec	57	Loganport	5	25107.80
1	5168bbf2-97af-4bde-bc8e-9d782d14e9f4	80	Robintown	4	25161.02
2	aa1d3fc6-5f31-473d-9519-830c40198303	59	North Randy	2	25249.50

Almost 490 Users are Paid More than $25000 Are Premium Out of 1000 Users

4) Calculate Average Income by Age Group

```
query="Select Age,avg(Income) From df5 Group by Age;"

result = pd.read_sql_query(query,conn)
result
```

	Age	avg(Income)
0	18	68140.772857
1	19	115301.455000
2	20	106333.218125
3	21	86633.578571
4	22	102859.938571
...	...	...
58	76	104706.482500
59	77	121370.102000
60	78	117810.505294

5) Find Customers Who Have High Claim Frequency (>8)

```
query="Select Customer_ID,Age,Claim_Frequency From df5 where Claim_Frequency>=8;"

result = pd.read_sql_query(query,conn)
result
```

	Customer_ID	Age	Claim_Frequency
0	35a240ae-5af3-4553-9ec4-2e0829a3b2e9	28	10
1	61b1cd22-6280-4c45-9043-5a1098ae4334	42	8
2	ae270da7-02f0-4b90-b143-262fdc5c0eed	61	10
3	e037e5ed-b8db-4672-b42d-47cc00d4af59	50	8
4	deda4e16-1b3d-4d5c-a9a1-fa6f81f76d1c	27	8
...	...	...	...
277	8ec57d47-0a0c-44e5-bc34-1b4ffbdc532e	77	10
278	74246baa-2dca-4c82-8660-a3e9736d417c	34	8
279	e83bcc84-a609-4ad0-b114-64aa52f69081	67	9

Above Claim Frequency Above 8 has 288 Cutomers its Very Bad

Highest Claim Frequency Customers are Almost Fraud

6) Rank Customers Based on Policy Upgrades (Highest First)

```
query="Select Customer_ID,Age,Policy_Upgrades From df5 Where Policy_Upgrades>0 Order by Policy_Upgrades DESC;"

result = pd.read_sql_query(query,conn)
result
```

	Customer_ID	Age	Policy_Upgrades
0	43b7a3a6-9a8d-4a03-980d-7b71d8f56413	40	3
1	88bd6407-2bcf-4e01-a28d-efe39bf00273	56	3
2	f16287e4-e9c3-49e0-b602-f8ac10f1bc81	76	3
3	89fa6a68-8fb5-427b-beb7-99193f22faf8	65	3
4	00257ad1-eb22-43dd-87c5-421eec24a3c5	33	3

Almost 755 Users are Upgraded Policies Out of 1000 Customers

7) Calculate Total Premium Paid Per Location

```
query="Select Location,sum(Total_Premium_Paid) as Total_Paid From df5 Group by Location Order by Total_Paid DESC;"

result = pd.read_sql_query(query,conn)
result
```

	Location	Total_Paid
0	Matthewfurt	127035.01
1	South Andrea	100386.12
2	Port Christopher	94080.38
3	Silvatown	91841.10
4	Hallfurt	89163.75

Preprocessing

1) Missing Values

```
df5.isnull().sum()
```

	0
Customer_ID	0
Age	0
Income	0
Location	0
Number_of_Active_Policies	0
Total_Premium_Paid	0
Claim_Frequency	0
Policy_Upgrades	0

dtype: int64

No Missing Values Found

If Missing Values have this Datasets.

Step 1:

check % of Missing Values if more than 70% Missing Values -> Delete that Columns

Step 2:

Check this Columns have Normal Distribution and No Outliers - Use Mean Imputation

Else use Median Imputation

Step 3:

Check that Columns Have Categorical Columns Use Mode

Step 4:

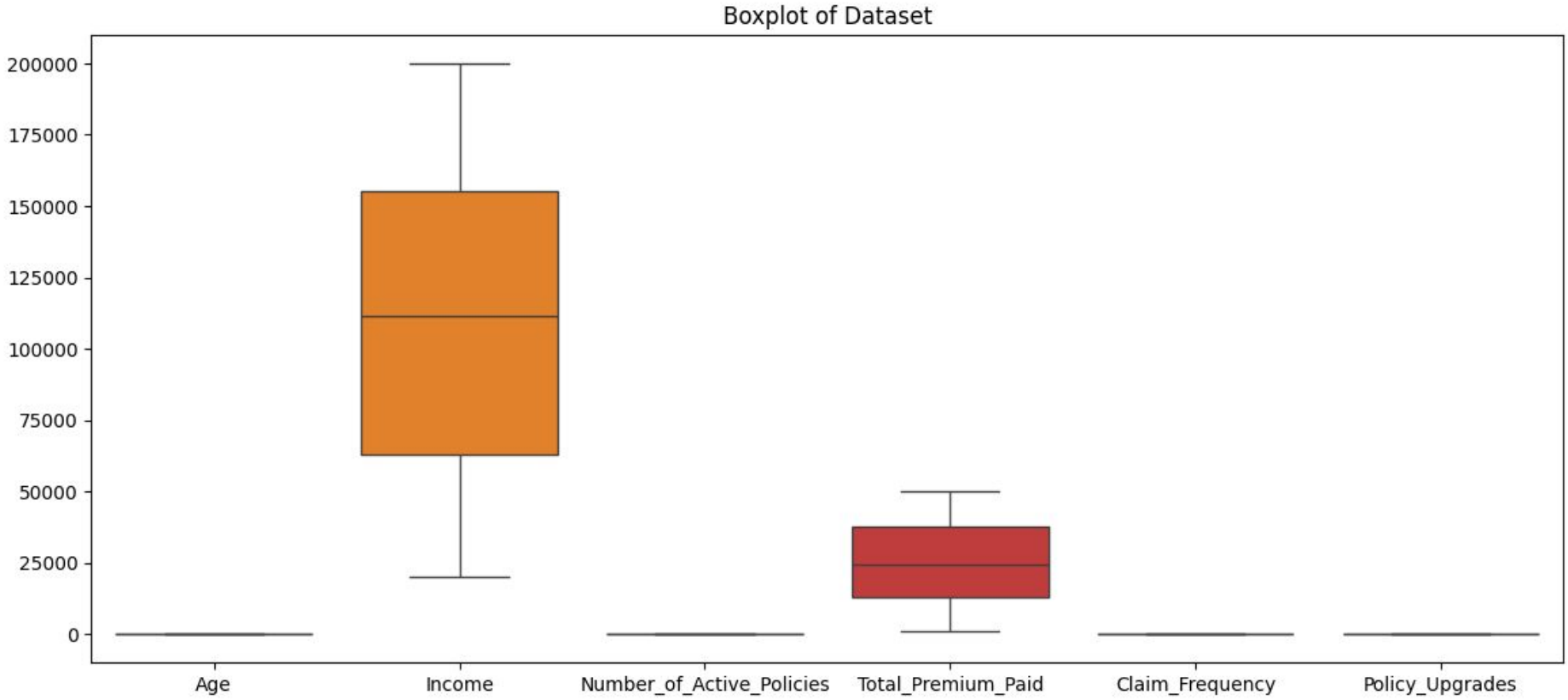
If That Dataset have Times series Use Forward fill or Backward Fill

Step 5:

If that Missing Values was Voluntarily Happen Based On Confidential else any other Situation

Client or SME or BA Give us Custom values fill us Cutom method to fill Missing Values

2) Outliers


Boxplot of Dataset
200000
175000
150000
125000
100000
75000
50000
25000
0
Age
Income
Number_of_Active_Policies
Total_Premium_Paid
Claim_Frequency
Policy_Upgrades

```python
import pandas as pd

# Detect outliers using IQR for all numeric columns
outliers = {}

for col in df5.select_dtypes(include=['number']).columns:
    Q1 = df5[col].quantile(0.25)
    Q3 = df5[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers[col] = df5[(df5[col] < lower_bound) | (df5[col] > upper_bound)][col].tolist()

# Print outliers for each column
for col, out in outliers.items():
    print(f"Outliers in {col}: {out}")

```

```
Outliers in Age: []
Outliers in Income: []
Outliers in Number_of_Active_Policies: []
Outliers in Total_Premium_Paid: []
Outliers in Claim_Frequency: []
Outliers in Policy_Upgrades: []
```

No Outlier Found in this Datasets

No Outlier Found in this Datasets

If Outlier Found in this Datasets:-

Step 1: Understand the Outliers

Check Data Entry Errors: Typos or incorrect values? Fix them if possible.

Assess Context: Are these extreme values realistic or expected in your data domain?

Step 2: Choose an Outlier Treatment Approach:

1) Removal Methods:

Delete Outliers: If they're definitely errors or irrelevant.

df_clean = df1[~((df1['A'] < lower_bound) | (df1['A'] > upper_bound))]

2) Imputation Methods:

Replace with Mean/Median: Works well when outliers aren't extreme.

median = df1['A'].median() df1['A'] = np.where((df1['A'] < lower_bound) | (df1['A'] > upper_bound), median, df1['A'])

Use Mode: For categorical data with outliers

3) Transformation Methods:

Log Transformation: Reduces effect of right-skewed outliers.

df1['A_log'] = np.log1p(df1['A'])

Square Root or Box-Cox: For data with different types of skew.

4) Capping or Clipping

Winsorization: Limit extreme values to percentiles.

from scipy.stats.mstats import winsorize df1['A_winsorized'] = winsorize(df1['A'], limits=[0.05, 0.05]) # Caps at 5th & 95th percentile

Cap at Boundaries: Replace outliers with IQR bounds.

df1['A'] = np.clip(df1['A'], lower_bound, upper_bound)

3) Encoding

No need Encoding. All Feature Columns has Numerical

```
[ ]  1 ######################################################################
```

3) Feature Scaling

Apply StandardScaler for numerical variables.

```
1 df5.head()
```

	Customer_ID	Age	Income	Location	Number_of_Active_Policies	Total_Premium_Paid	Claim_Frequency	Policy_Upgrades
0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	58	40039.59	Johnberg	3	12999.70	2	0
1	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	61	153320.19	New Roberttown	5	5260.00	6	0
2	815ef6d1-3b8f-4a18-b7f8-a88b17fc695a	19	36865.14	New Jamesside	2	25762.41	0	1
3	96da1dac-72ff-4d2a-b86e-cbe06b65a6a4	63	136979.20	Lisatown	5	21556.47	7	2
4	b2b9437a-28df-4ec4-8e4a-2bbdc241330b	69	176474.06	Petersonberg	2	35208.83	5	2

So i took Numerical Columns from **[Income & Total_Premium_Paid]**

Reason:

this columns have 4 digit values, 5 digit values and 6 digit values machine very tough to upderstand or high variance create so i scale this columns take as Means = 0 and take standard deviation as 1 : to scale the columns as standardization.

Z-Score method

```
[ ]  1 from sklearn.preprocessing import StandardScaler
     2
     3 # Initialize the scaler
     4 scaler = StandardScaler()
     5
     6 # Select the columns to standardize
     7 columns_to_scale = ['Income', 'Total_Premium_Paid']
     8
     9 # Apply the scaler and transform the columns
    10 df5[columns_to_scale] = scaler.fit_transform(df5[columns_to_scale])
    11
    12 # Check the results
    13 print(df5.head())
```

```
                            Customer_ID  Age    Income        Location  \
0  bdd640fb-0667-4ad1-9c80-317fa3b1799d   58 -1.347477        Johnberg
1  bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9   61  0.835905  New Roberttown
2  815ef6d1-3b8f-4a18-b7f8-a88b17fc695a   19 -1.408661   New Jamesside
3  96da1dac-72ff-4d2a-b86e-cbe06b65a6a4   63  0.520947        Lisatown
4  b2b9437a-28df-4ec4-8e4a-2bbdc241330b   69  1.282176    Petersonberg

   Number_of_Active_Policies  Total_Premium_Paid  Claim_Frequency  \
0                          3           -0.861332                2
1                          5           -1.408809                6
2                          2            0.041455                0
3                          5           -0.256058                7
4                          2            0.709659                5

   Policy_Upgrades
0                0
1                0
2                1
3                2
4                2
```

```
1 df5.head()
```

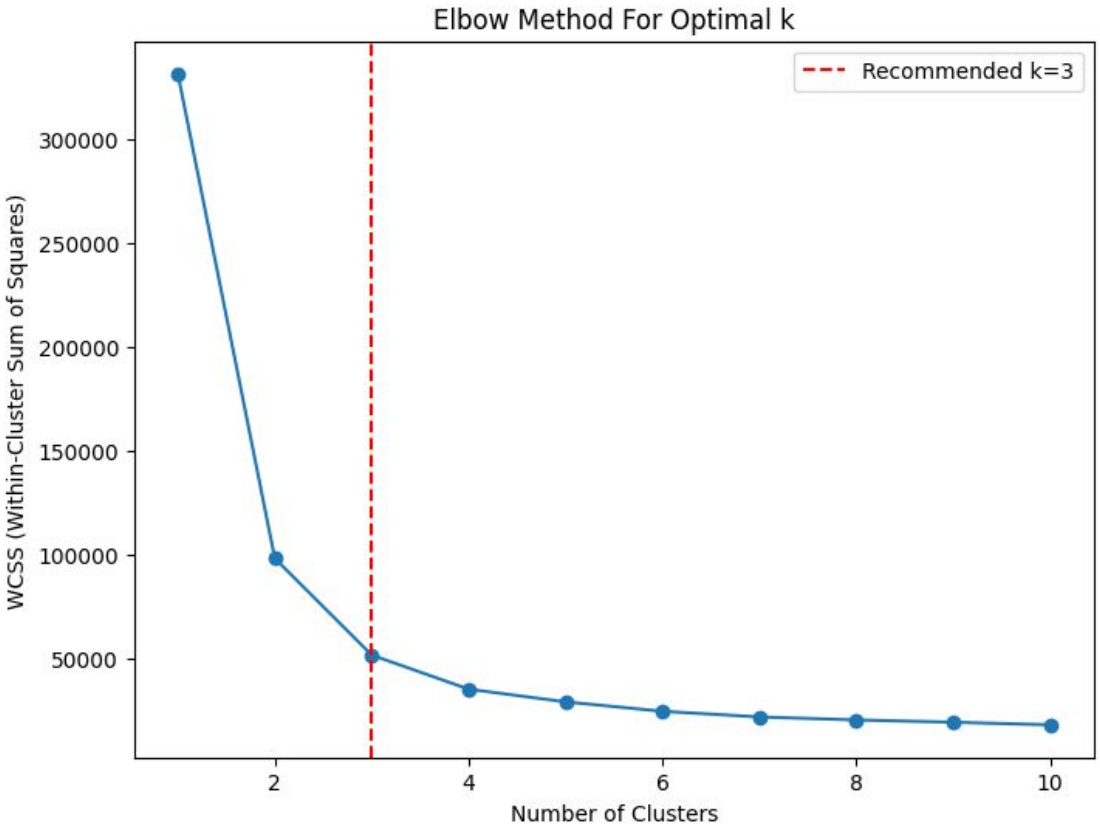
	Customer_ID	Age	Income	Location	Number_of_Active_Policies	Total_Premium_Paid	Claim_Frequency	Policy_Upgrades
0	bdd640fb-0667-4ad1-9c80-317fa3b1799d	58	-1.347477	Johnberg	3	-0.861332	2	0
1	bd9c66b3-ad3c-4d6d-9a3d-1fa7bc8960a9	61	0.835905	New Roberttown	5	-1.408809	6	0
2	815ef6d1-3b8f-4a18-b7f8-a88b17fc695a	19	-1.408661	New Jamesside	2	0.041455	0	1
3	96da1dac-72ff-4d2a-b86e-cbe06b65a6a4	63	0.520947	Lisatown	5	-0.256058	7	2
4	b2b9437a-28df-4ec4-8e4a-2bbdc241330b	69	1.282176	Petersonberg	2	0.709659	5	2

```
[ ] 1 df5.to_csv('df5_scaled.csv')
```

Model Training

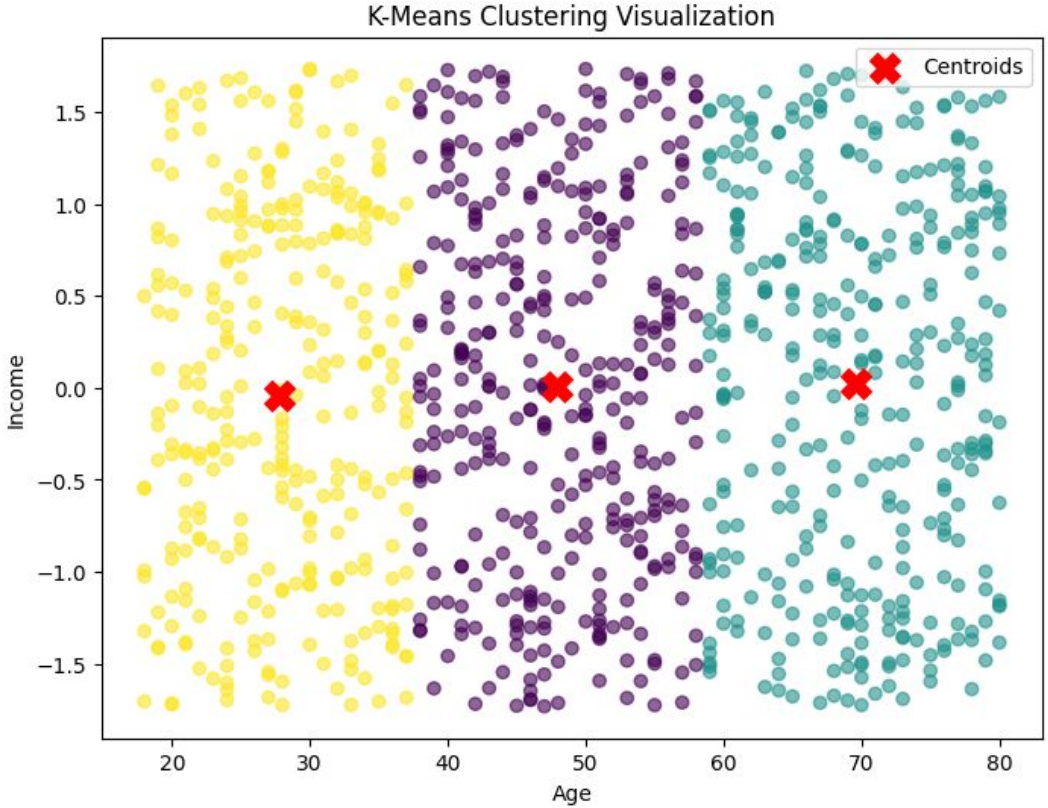
Clustering Algorithm Selection

1) k-Means Clustering



Using Elbow Method to Find K Value that Means Number of Cluster Value

Recommended number of clusters: 3



	Age	Income	Number_of_Active_Policies	Total_Premium_Paid	\
0	58	-1.347477	3	-0.861332	
1	61	0.835905	5	-1.408809	
2	19	-1.408661	2	0.041455	
3	63	0.520947	5	-0.256058	
4	69	1.282176	2	0.709659	

	Claim_Frequency	Policy_Upgrades	Cluster
0	2	0	0
1	6	0	1
2	0	1	2
3	7	2	1
4	5	2	1

Silhouette Score for k=3: 0.50

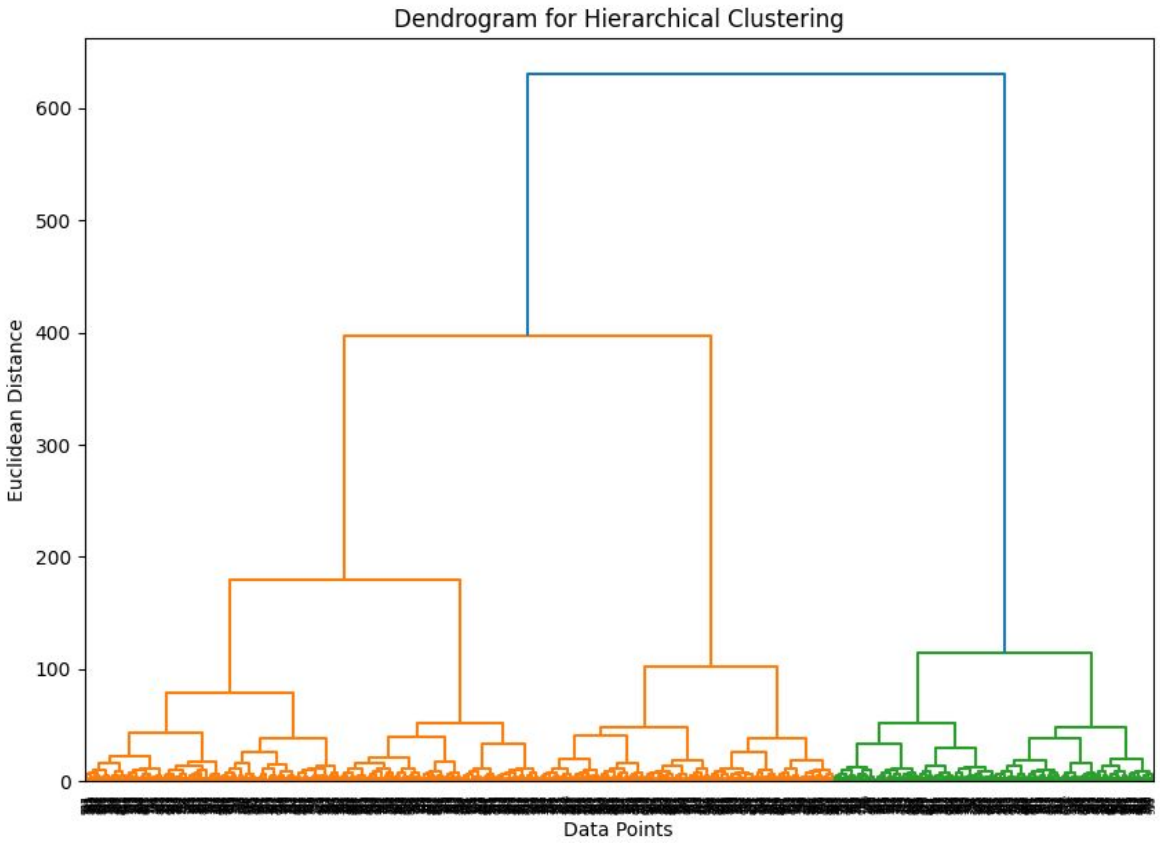
Number of data points in each cluster:

Cluster	
0	356
1	347
2	297

Name: count, dtype: int64

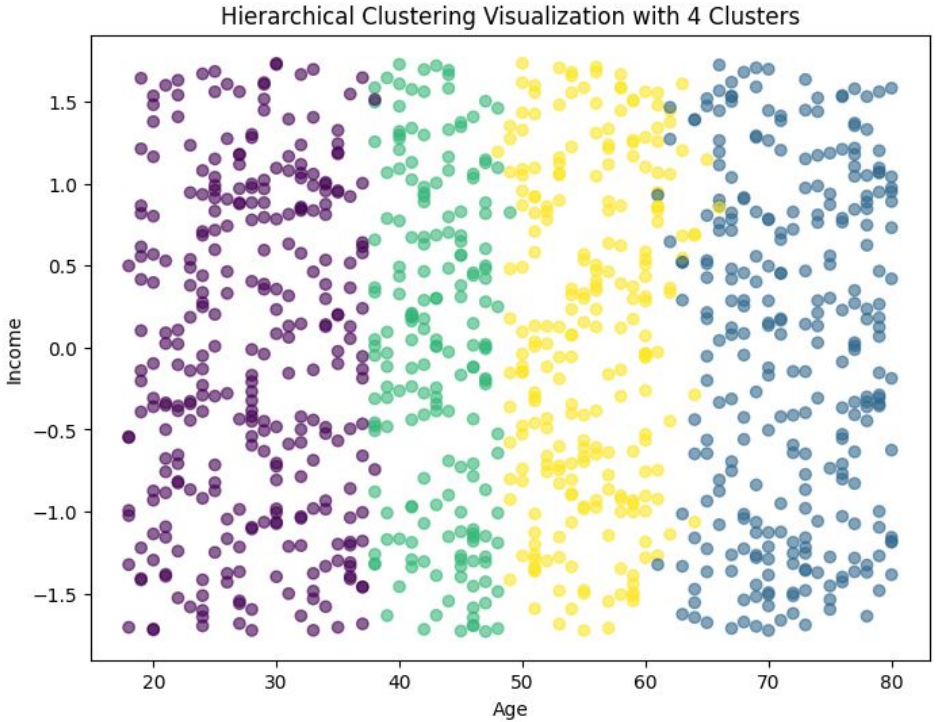
Silhouette Score is 0.50

2) Hierarchical [agglomerative clustering]



Using Dendrogram Using to Find Number of Clusters in Hierarchical

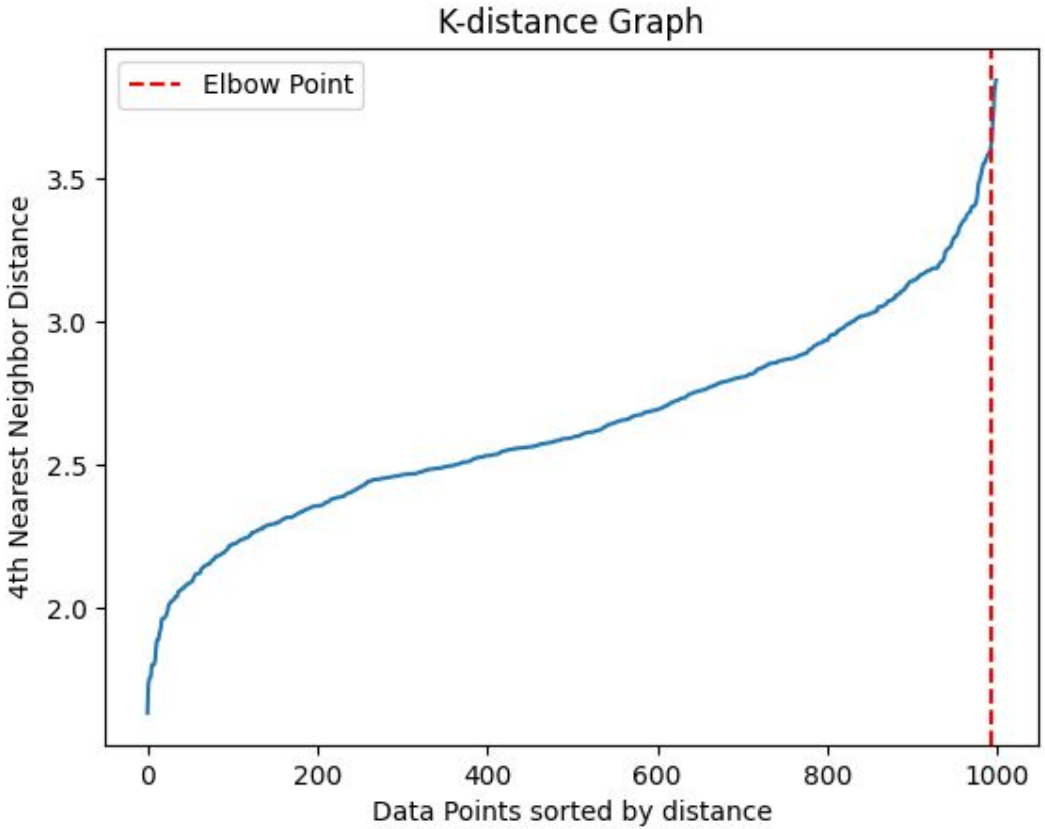
Actual Clusters 600 above based on Chart but am giving 4



```
Recommended number of clusters based on dendrogram: 4
Based on the dendrogram, enter the suggested number of clusters: 4
Silhouette Score for 4 clusters: 0.42

Number of data points in each cluster:
   Cluster  Count
0        1    299
1        2    276
2        4    239
3        3    186
```

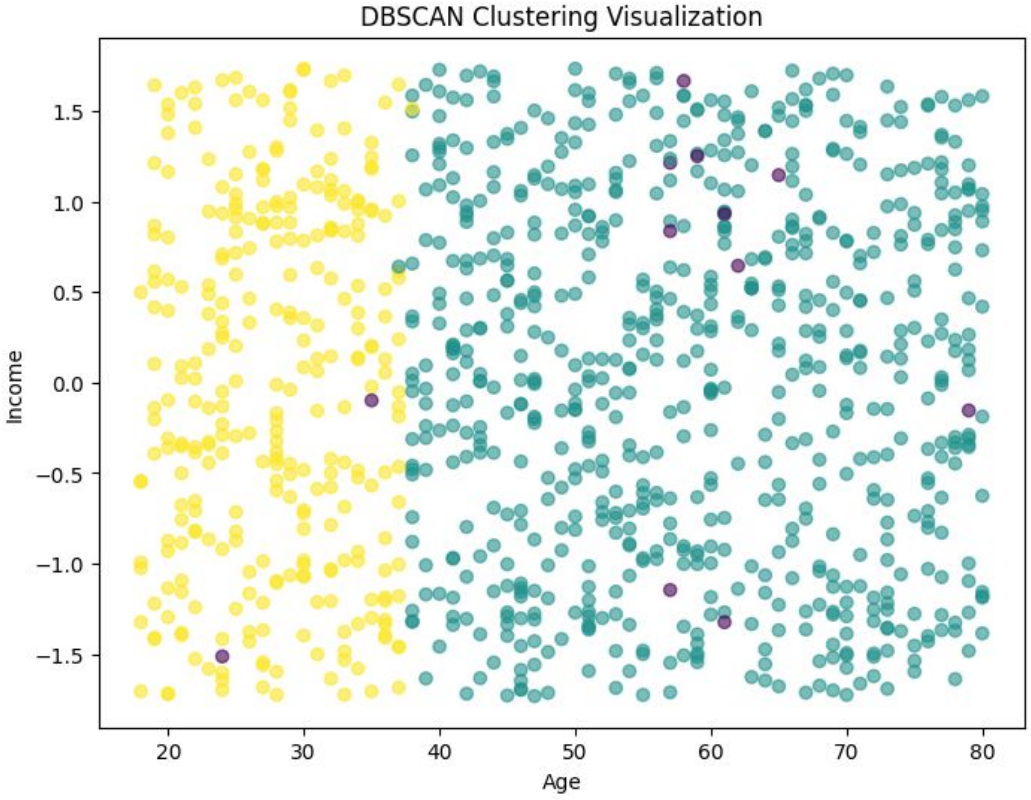
Silhouette Score is 0.42 is Low Compared to K-means

3) DBScan Algorithm



Using K-distance Graph to Find a EPS (Radius of Circle) and Find Number of Samples Per Circle

```
Recommended eps: 3.61
Recommended min_samples: 16
```



```
Enter the recommended eps value based on the elbow point: 3.61
Enter the recommended min_samples value: 16
Silhouette Score: 0.49

Number of data points in each cluster:
DBSCAN_Cluster
 0    693
 1    295
-1     12
Name: count, dtype: int64
```

Silhouette Score : 0.49

	Age	Income	Number_of_Active_Policies	Total_Premium_Paid	\
0	58	-1.347477	3	-0.861332	
1	61	0.835905	5	-1.408809	
2	19	-1.408661	2	0.041455	
3	63	0.520947	5	-0.256058	
4	69	1.282176	2	0.709659	

	Claim_Frequency	Policy_Upgrades	Cluster	Hierarchical_Cluster	\
0	2	0	0	4	
1	6	0	1	4	
2	0	1	2	1	
3	7	2	1	2	
4	5	2	1	2	

	DBSCAN_Cluster
0	0
1	0
2	1
3	0
4	0

Silhouette Score For K- Means Clustering -> 0.50

Silhouette Score For Hierarchical Clustering -> 0.16

Silhouette Score For DBSCAN Clustering -> 0.49

Silhouette Score Near to 1 is Good - K-Means Clustering is 0.50 is Higher Compare to Other Clustering

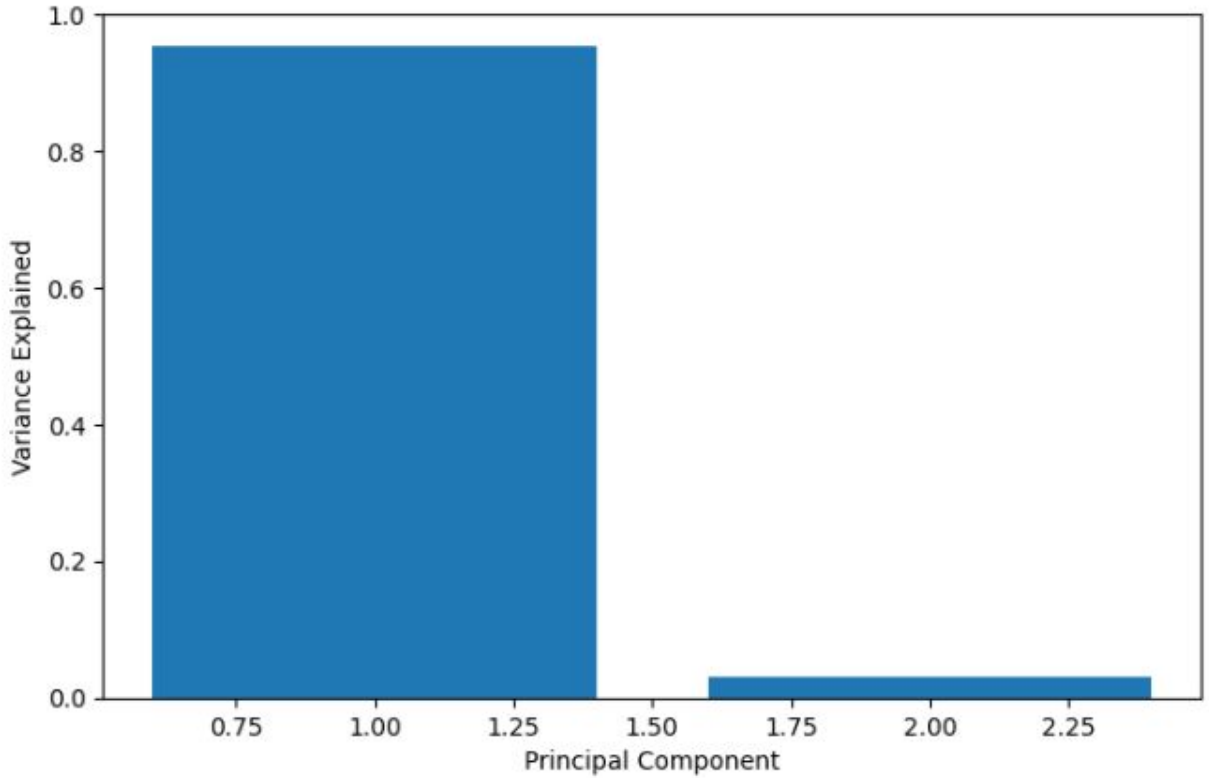
✅ **Feature Selection**

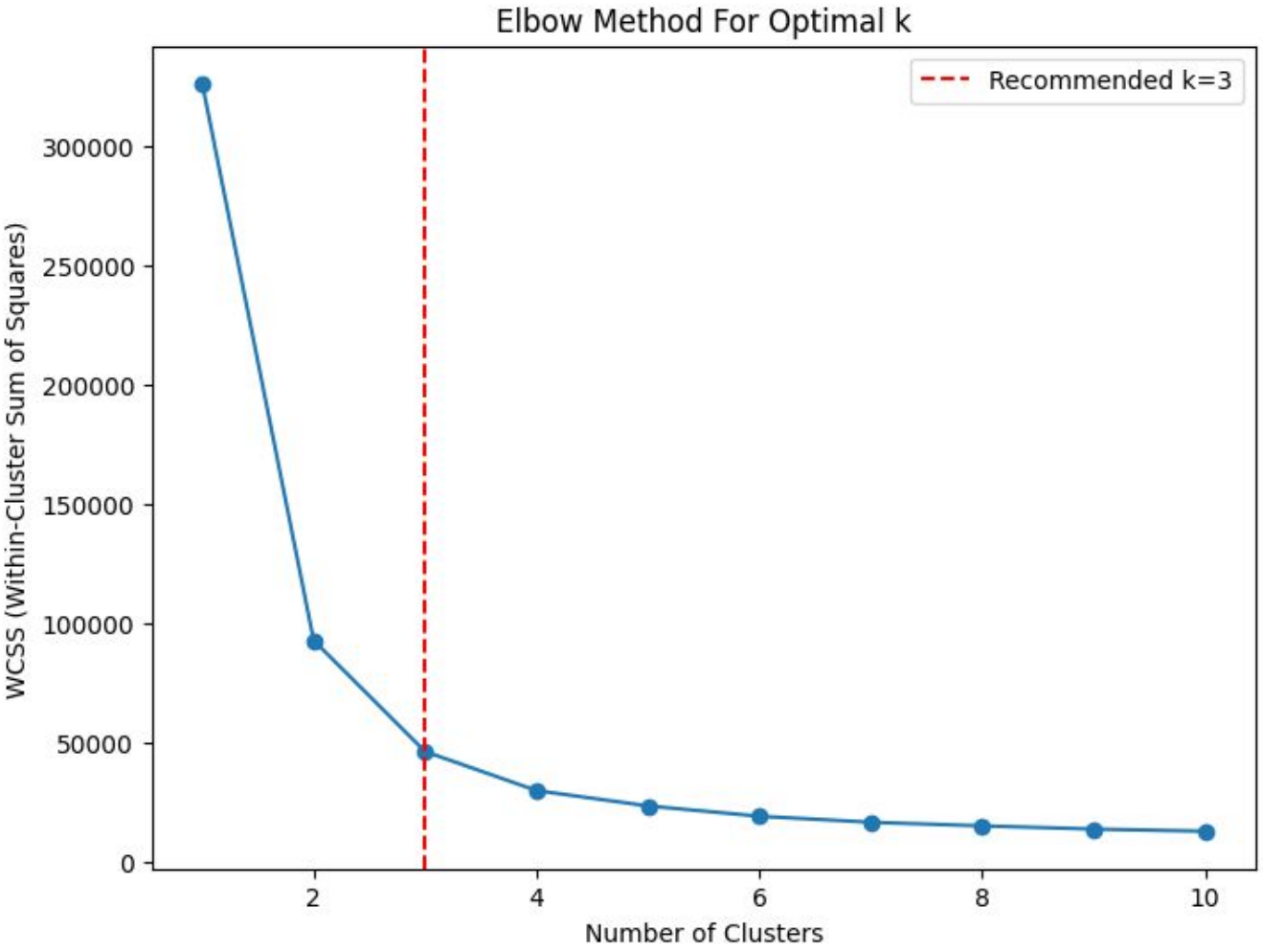
- Use **Principal Component Analysis (PCA)** to reduce dimensionality before clustering.

K-Means Clustering Selected

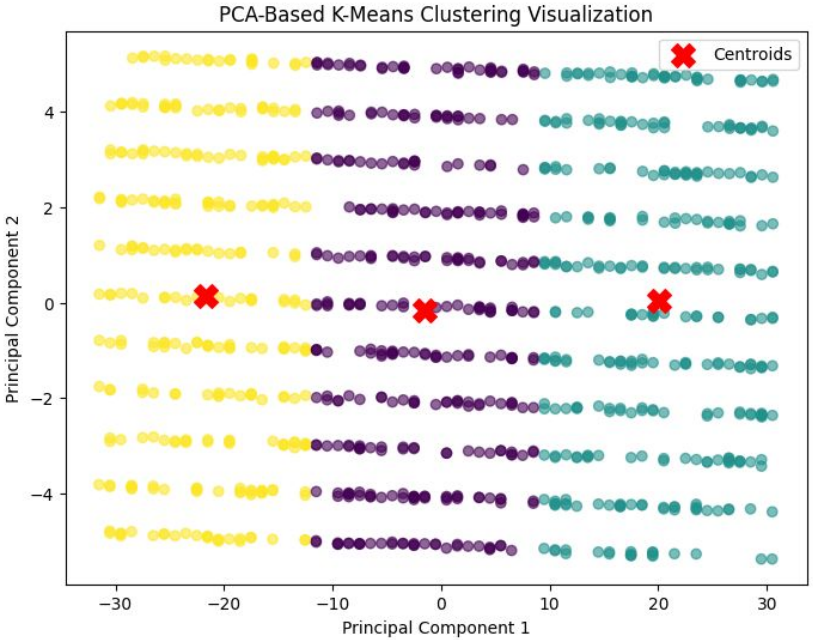
PCA - K-Means

```
            Age    Income  Number_of_Active_Policies  Total_Premium_Paid  \
PC1    0.999956  0.001144                  -0.000235            0.001522
PC2   -0.008529 -0.023155                  -0.015502           -0.007245

     Claim_Frequency  Policy_Upgrades
PC1         0.008554         0.003202
PC2         0.999542         0.003785
```




Elbow Method For Optimal k
Recommended k=3
WCSS (Within-Cluster Sum of Squares)
300000
250000
200000
150000
100000
50000
0
2
4
6
8
10
Number of Clusters

```
Recommended number of clusters: 3
Silhouette Score for k=3: 0.52

Number of data points in each cluster:
Cluster
0    356
1    347
2    297
Name: count, dtype: int64
```



	PC1	PC2	Cluster
0	8.494422	-3.134971	0
1	11.529703	0.760016	1
2	-30.516246	-4.787262	2
3	13.545968	1.749012	1
4	19.531646	-0.279362	1

Claim_Frequency - Policy_Upgrades

PC1 - 0.008554 - 0.003202

PC2 - 0.999542 - 0.003785

1️⃣ Interpret the Clusters

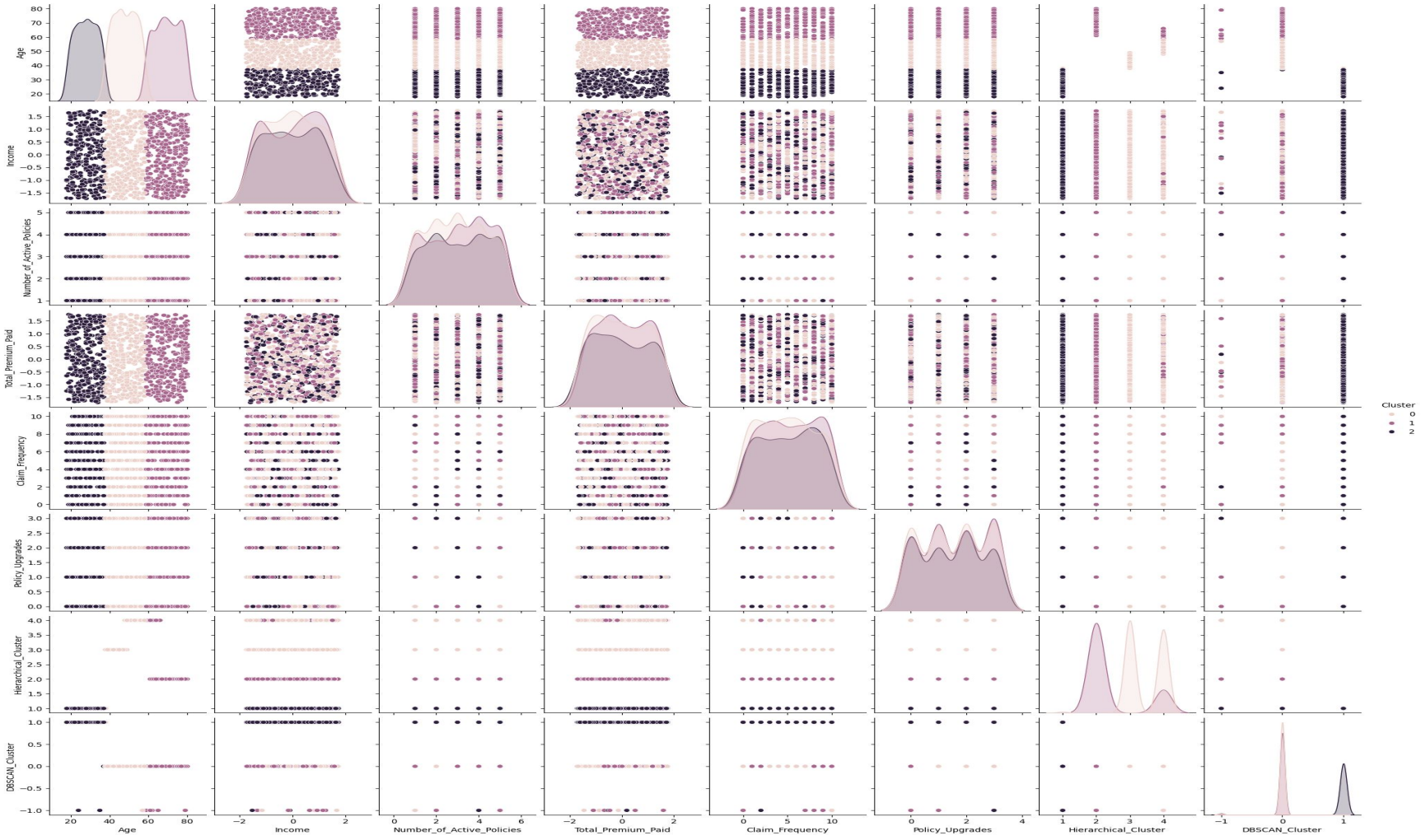
Understand the meaning of each cluster:

Check the average values of original features for each cluster to see what defines them

```
1 cluster_summary = df.groupby('Cluster').mean()
```

```
1 cluster_summary
```

Cluster	Age	Income	Number_of_Active_Policies	Total_Premium_Paid	Claim_Frequency	Policy_Upgrades
0	47.952247	0.008523	2.960674	-0.022918	4.918539	1.525281
1	69.547550	0.027331	3.097983	0.045963	5.317003	1.576369
2	27.835017	-0.042148	3.080808	-0.026230	5.050505	1.461279


Age
Income
Number_of_Active_Policies
Total_Premium_Paid
Claim_Frequency
Policy_Upgrades
Hierarchical_Cluster
DBSCAN_Cluster
Cluster
0
1
2

```
1 print(f'K-Means Inertia: {kmeans.inertia_}')
```

```
K-Means Inertia: 46237.590486579116
```

🚀 Step-by-Step Cluster Analysis & Insights Extraction

Now that your clustering is complete, let's analyze each cluster's characteristics. Here's what we will do

✅ **1. Find Cluster Centroids** → Understand key feature values for each cluster.

✅ **2. Compare Feature Distributions** → Identify patterns across clusters.

✅ **3. Generate Business Insights** → Apply findings to fraud detection or customer segmentation.

1. Get Cluster Centroids

Average Feature Values for Each Cluster:

```
[ ]
```

```python
import pandas as pd

# Convert centroids to a DataFrame for better readability
centroid_df = pd.DataFrame(kmeans.cluster_centers_, columns=['PC1', 'PC2'])

# Add cluster labels
centroid_df.index = [f'Cluster {i}' for i in range(len(centroid_df))]

print("Cluster Centroids:\n", centroid_df)
```

```
Cluster Centroids:
                 PC1       PC2
Cluster 0  -1.520206 -0.163161
Cluster 1  20.077823  0.048065
Cluster 2 -21.635728  0.139416
```

This will show how different each cluster is based on PCA components.

2. Compare Feature Distributions per Cluster

Since PCA transformed features, we need to map clusters back to original features:

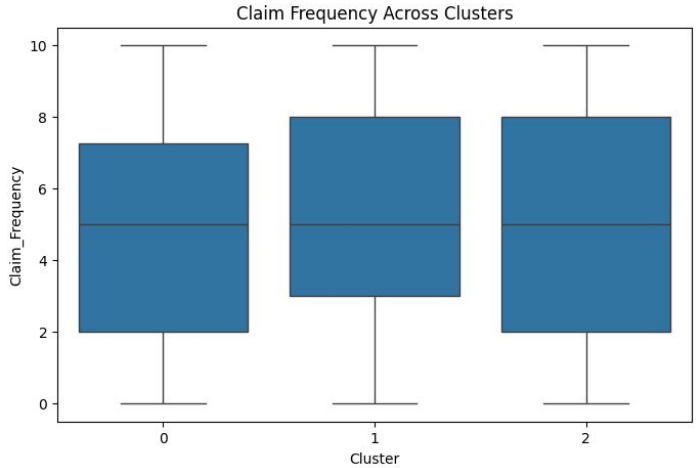
```
[ ]
# Add cluster labels to the original dataset
df['Cluster'] = kmeans.labels_

# Group by clusters and calculate feature means
cluster_summary = df.groupby('Cluster').mean()

print("Cluster-wise Feature Averages:\n", cluster_summary)

```

```
Cluster-wise Feature Averages:
                Age    Income  Number_of_Active_Policies  Total_Premium_Paid  \
Cluster
0         47.952247  0.008523                   2.960674           -0.022918
1         69.547550  0.027331                   3.097983            0.045963
2         27.835017 -0.042148                   3.080808           -0.026230

          Claim_Frequency  Policy_Upgrades  Hierarchical_Cluster  \
Cluster
0                4.918539         1.525281              3.460674
1                5.317003         1.576369              2.409222
2                5.050505         1.461279              1.000000
```

Cluster 1 has a Highest Income, Highest Claim Frequency, High Number of Policy Upgrades, Claim Frequency

This will help us answer:

- **Which cluster has higher income?**
- **Which cluster has more active policies?**
- **Which cluster has high claim frequency (potential fraud risk)?**



Large Number of Claim Frequency Happens on Cluster 2

Cluster	Key Features	Insights & Actions
🟢 **Cluster 0**	Low claim frequency, high income, many policies	**Loyal Customers** → Offer premium policies
🟡 **Cluster 1**	Medium claim frequency, moderate income	**Standard Customers** → Regular engagement
🔴 **Cluster 2**	High claim frequency, low income, few policies	**High-Risk (Possible Fraud)** → Investigate claims

🚀 Identifying High-Risk (Fraud-Prone) Clusters

Now, let's **identify clusters with high fraud risk** by analyzing claim frequency, income levels, and other key factors.

🔹 1. Find the High-Risk Cluster

We assume fraud-prone customers tend to have:

✅ **High Claim Frequency**

✅ **Low Income**

✅ **Few Active Policies**

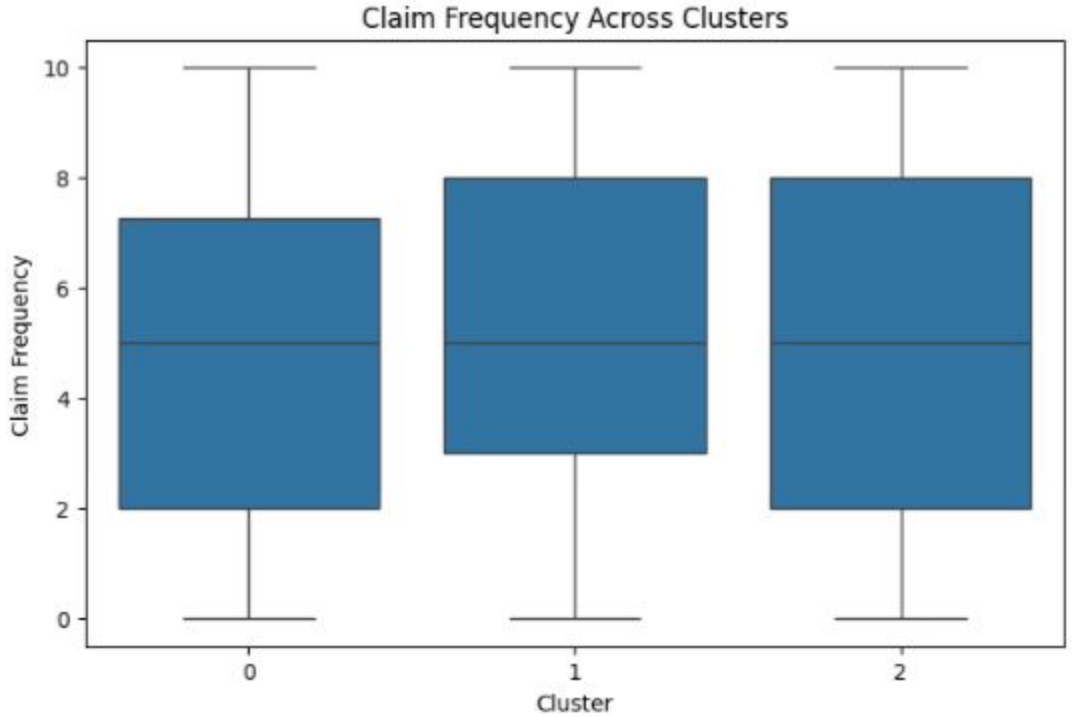
```
[ ]  1 # Sort clusters by claim frequency (descending)
     2 high_risk_cluster = df.groupby('Cluster')[['Claim_Frequency', 'Income', 'Number_of_Active_Policies']].mean().sort_values(by='Claim_Frequency', ascending=False)
     3
     4 print("Cluster Characteristics (Sorted by Claim Frequency):\n", high_risk_cluster)
```

```
Cluster Characteristics (Sorted by Claim Frequency):
          Claim_Frequency    Income  Number_of_Active_Policies
Cluster
1                5.317003  0.027331                   3.097983
2                5.050505 -0.042148                   3.080808
0                4.918539  0.008523                   2.960674
```

High Claim Frequency and High Number of Active Policies Consider as a Cluster 1

Low Income Group Consider as Class 0

- **The top cluster** in this list is likely the **high-risk** one.
- If a cluster has **high claims + low income**, it might indicate **potential fraud**.



A cluster with high median & many outliers is likely fraud-prone.

🔥 Scatter Plot of Income vs Claim Frequency

If a cluster has low income & high claim frequency, it's a fraud-risk group

3. Extracting High-Risk Customers

```python
# Replace '2' with the actual high-risk cluster ID from analysis
high_risk_customers = df[df['Cluster'] == 1]

print("Number of High-Risk Customers:", len(high_risk_customers))
high_risk_customers.head()

```

```
Number of High-Risk Customers: 347
```

	Age	Income	Number_of_Active_Policies	Total_Premium_Paid	Claim_Frequency	Policy_Upgrades	Cluster
1	61	0.835905	5	-1.408809	6	0	1
3	63	0.520947	5	-0.256058	7	2	1
4	69	1.282176	2	0.709659	5	2	1
14	76	1.537551	3	-0.973166	7	3	1
15	74	1.439154	4	-1.214942	2	1	1

- Flag them for **manual review**
- Send data to a **fraud detection model** for further risk scoring

Conclusion:

The K-Means clustering approach successfully segmented the dataset into distinct groups based on policy-related features. With a silhouette score of **0.50**, K-Means outperformed Hierarchical and DBSCAN clustering, indicating well-defined clusters. PCA helped reduce dimensionality while preserving key information. This clustering can be used for customer segmentation, risk assessment, and targeted policy recommendations.

Thank You