Insurance Customer Classification

100498877

Abstract

The report gives a comprehensive analysis of the Insurance Dataset. The analysis performs categorization and segmentation of customers based on their socio-demographic factors like Income, Number of houses etc. This study involves data exploration, data preprocessing, data encoding, feature engineering, feature selection, normalization. The data has 7 categorical variables and 76 numerical features.

1 Introduction

Understanding customer profiles is a fundamental aspect of strategic decision-making in the insurance industry. The center goal of this analysis is to classify customers into defined subtypes using different machine learning techniques. They are categorized into 7 subtypes; Rural & Low-income, Middle-Class Families, Young & Low-income, Seniors & Retired, Wealthy & Affluent. The dataset contains 75 numerical features which is the base for performing different machine learning algorithms. The project is a pipeline of several phases: **Data Exploration, Data Preprocessing, Supervised Learning, Unsupervised Learning, Evaluation**.

	Column Name	Data Type	Field Size	Description	Example
0	Customer_Type	object	5521	Group the customer belongs to based on their l...	0 [Rural & Low-income, Rural & Low-incom...
1	Number_of_Houses	int64	5521	How many houses the customer owns.	0 [1, 1, 1] 1 [1, 1, 1] 2 [1...
2	Avg_Household_Size	int64	5521	Average number of people in the household.	0 [3, 2, 2] 1 [3, 2, 2] 2 [3...
3	Avg_Age	object	5521	Average age of people in the household.	0 [30-40 years, 30-40 years, 30-40 years...
4	Household_Profile	object	5521	Type of household the customer lives in.	0 [Family with Grown-Ups, Family with Gr...
...	...	...	...	...	...

Figure 1: Data Dictionary

2 Data Exploration and Visualization

We analyzed all the records in the dataset, including the 'Customer Type'. This goal is to explore the dataset containing customer information. This gives an

understanding of the characteristics of each customer group and how insurance company can cater to different customer needs. Some columns contain numerical datatypes while others had categorical data. The numerical data were basically 'income', 'Number of houses', 'Avg Age', etc. A few columns had missing values, with some missing data having more significance in certain features ('Insurance Contributions'). We observed a few extreme values, particularly in income and contributions fields. In **Fig: 2**, the distribution among classes varies significantly. The plot shows that 'Rural and Low Income' has the largest number of data points, while the other categories exhibit imbalanced data.

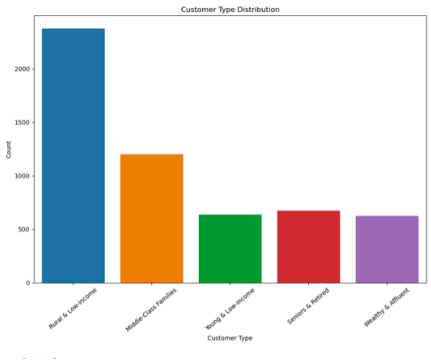


Figure 2: Visualization of Class Distribution of Customer Type

Fig: 3, shows a heatmap describing the correlation between certain features. This chart helps to quickly identify which numeric factors might influence each other. The correlations shown are those set with a threshold of 0.5 and -0.5. The The bright pink in diagonal line represents a perfect correlation with itself. Many insurance contributions features like 'Fire Insurance Contribution', 'Boat Insurance Contribution' along with 'Number Boat Insurance', 'Number Fire Insurance' are strongly correlated, that people who pay more towards a particular insurance are likely to hold more policies of that type. Features like 'Income 75K to 122K' or 'Average Income' show positive correlation with 'Owns One Car' and negative correlation with 'Owns No Car'. A cluster of vehicle related policies are correlated with each other. This shows that higher income individuals are more likely to own a car. People who own homes tend to own more than one as 'Number of Houses' and 'Home Owner' show strong positive correlation.

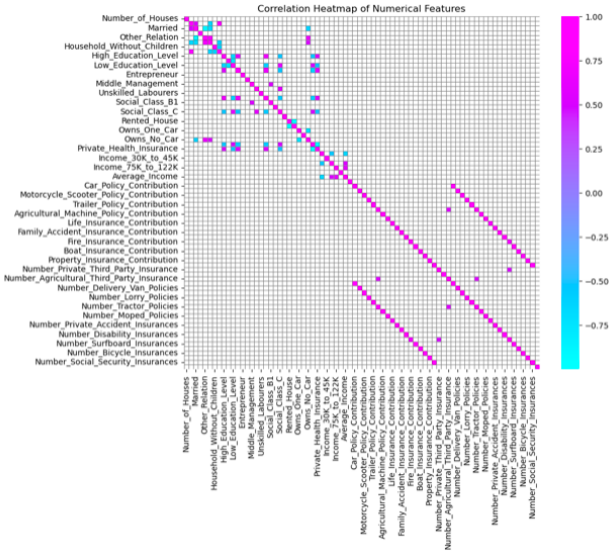


Figure 3: Heatmap summarizing correlation between features

2.1 Initial Findings

The exploration reveals that a number of features like 'Car Policy Contribution', 'Number Life Insurance' etc exhibit skewness, with more than 50 percent of zeros with high sparsity. Some features like 'Avg Age', 'Income Above 123K', 'Business Third Party Insurance Contribution', 'Delivery Van Policy Contribution', etc also contains missing values.

3 Data Preprocessing

This involves the following tasks:

- Handling Missing Values
- Outlier Treatment
- Feature Engineering
- Feature Selection
- Train-Test Split

3.0.1 Handling Missing Values

Having missing values while training and testing can lead to biased model accuracy, potential errors during model execution. Missing values in categorical columns like 'Avg Age', 'Private Third Party Insurance Contribution' were filled with the label 'Unknown'. Missing values in numerical columns like 'Number Property Insurances' were filled with the median of that column.

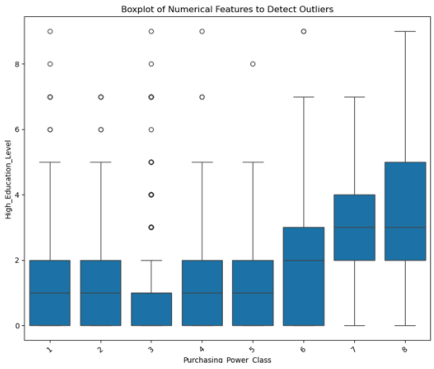


Figure 5: Box plot for Outlier Detection in correlation between Power Class and High Education level

```
Columns with Null Values:
 Customer_Type                                    0
Avg_Age                                          14
Household_Profile                                 0
Private_Third_Party_Insurance_Contribution        9
Business_Third_Party_Insurance_Contribution       0
Agricultural_Third_Party_Insurance_Contribution   0
Mobile_Home_Policies                             65
dtype: int64
```

Figure 4: Null Values within Categorical Columns

3.0.2 Outlier Treatment

The presence of extreme values in numerical columns like 'Car Policy Contribution', 'Income' etc can distort the statistical measures and model performance. Normalizing the data can get more consistent patterns. **Boxplot** was used for visualization and detection of outliers within the dataset. **log1p** was applied for normalizing the skewed distribution. It is worth noting that not a lot of the features had extreme outliers, they just exibited outliers of different densities, generally lying between 0-9.

3.0.3 Feature Engineering

The Insurance dataset contains upto 7 categorical features that cannot be processed by machine learning models. Features like 'Customer Type', 'Avg Age', 'Middle Class Families', etc needs to be encoded for further processing, and for this, label encoder was used to assign a number ranging between 0 - 9 to each type in the target variable. Models like KNN, Decision Tree Classifier, SVC require numerical inputs for computations. One hot encoding was the first choice, but it tends to increase the number of dimensions in the dataset, which was not fit for this analysis.

	Avg_Age_Encoded	Household_Profile_Encoded	Private_Third_Party_Insurance_Contribution_Encoded
0	1	5	0
1	1	5	2
2	1	5	2
3	2	0	0
4	1	6	0
...	...	...	...

Figure 6: Some of the Encoded Data from the Insurance Dataset

3.0.4 Feature Selection

A significant number of features in the dataset consist of more than 50 percent of zeros which is not helpful for predicting or classifying the right values among the classes. So to tackle that issue, we used Mutual Info Classif followed by Dimensionality Reduction using Principal Component Analysis (PCA). The Mutual Information Classifier was used to indicate the relationship of each variable with the target variable. Variables like Household Profile Encoded shows a score of 1.33 which says that it has comparatively the highest relation with Customer Type, whereas Avg Age Encoded, Private_Third_Party_Insurance_Contribution_Encoded, Agricultural Third Party Insurance Contribution Encoded has a moderate relation. It also includes features with a zero MI score, indicating that they have no effect on the predictability or classification of customers with respect to customer type. Such features are considered for removal.

```
mi_series #Mutual Info Classif with scores
```

```
Avg_Age_Encoded                                          0.085267
Household_Profile_Encoded                                1.336244
Private_Third_Party_Insurance_Contribution_Encoded       0.007906
Business_Third_Party_Insurance_Contribution_Encoded      0.000000
Agricultural_Third_Party_Insurance_Contribution_Encoded  0.018037
                                                           ...
Number_Boat_Insurances                                   0.001788
Number_Bicycle_Insurances                                0.000000
Number_Property_Insurances                               0.000000
Number_Social_Security_Insurances                        0.000000
Number_Mobile_Home_Policies                              0.000000
Length: 82, dtype: float64
```

Figure 7: Mutual Info Classification and Corresponding Scores

3.0.5 Train Test Split

For building a model to customer types, we first split the dataset into train and test data to 80-20 percent. This splitting has different effects on various models. Like SVC model revealed that 93 percent, slightly overfit, but simplifying the data increased the accuracy to 95 percent. The Decision Tree Classifier was overfit yet gaining a percentile value of 92 percent after tuning and our Neural Network achieved an impressive 96.6 percent accuracy. This is has been occurring across other models like KNN, Random Forest Classifier etc for precise classification.

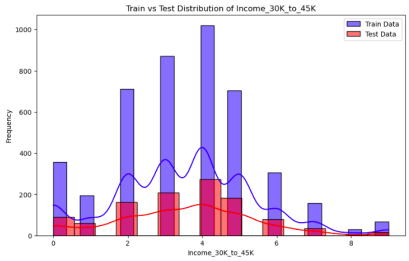


Figure 8: Histogram for Visualizing Income 30K to 45K Train-Test Distribution

Two types of learning was performed on this dataset: Supervised learning and Unsupervised learning

4 Supervised Learning

4.1 Models Used:

- Decision Tree Classifier
- K-Nearest Neighbors (KNN)
- Support Vector Classifier (SVC)

4.1.1 Decision Tree Classifier

Performing the Decision Tree Classifier initially gave us an accuracy of 98.8 percent which clearly shows signs of overfitting. As an improvement, we induced Dimensionality reduction using PCA with components having a variance of 0.95, approx. 23 features involved. This is done via a pipeline for a well structured approach. This time we can say the DT accuracy was 88.7 percent which shows less overfitting in the training data, also proving better classification than earlier.

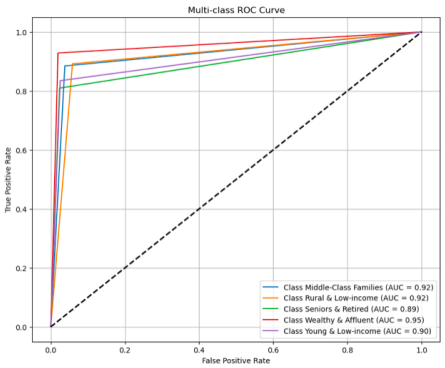


Figure 9: ROC curve of a Pipeline Decision Tree Classifier

4.1.2 K-Nearest Neighbors

Without Dimensionality Reduction, the KNN model works good in identifying middle and wealthy classes, but sometimes mixes up with other income groups

such as rural/low income with middle classes caused by overlapping categories. When enhanced with data preprocessing techniques like Dimension Reduction using PCA via pipeline, the accuracy improves upto 88.6 percent, working particularly well for middle and rural/low income groups. It is comparatively less accurate for young/low income classes with only 79 percent while wealthy and senior classes shows 90 percent classification. These improvements helps distinguish between overlapping categories.

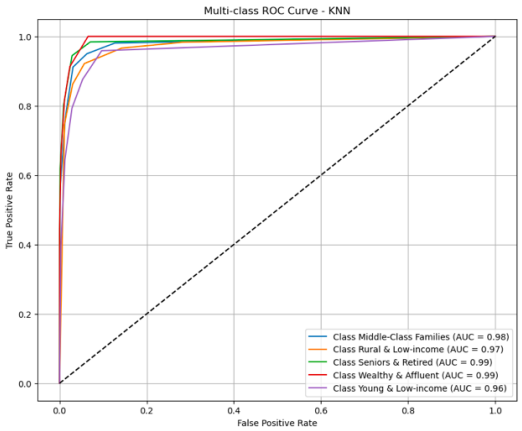


Figure 10: ROC curve for Pipeline KNN

4.1.3 Support Vector Classifier

For better classification, we used SVC with and without Dimensionality Reduction. Initially the model performed well, achieving an accuracy of 93.76 percent, strongly identifying middle class and rural customers. It did show a slight lower performance with other categories. We simplified the data using a Mutual Info Classification, along with PCA, reducing unnecessary details. The accuracy increased upto 95.3 percent, thus the model became more balanced across all classes.

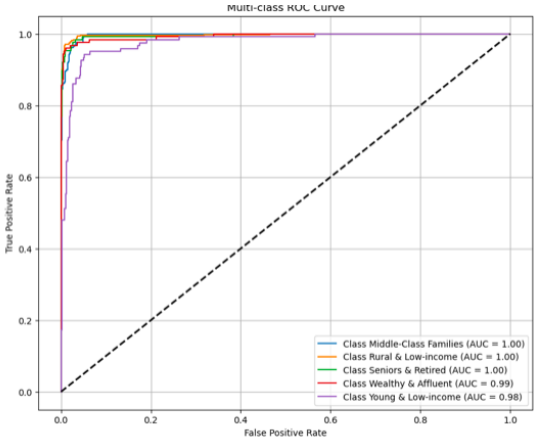


Figure 11: ROC curve of a Pipeline SVC

4.2 Hyper Parameter Tuning

We used a Decision Tree Classifier, Support Vector Classifier, and a Random Forest Classifier with Dimensionality Reduction dataset with an overall variance of 0.95, via pipeline. An accuracy of 91.7 percent for DT classifier, categorizing most of the customer types, especially the middle class and rural groups. Then we tested the same with a Tuned SVC and made improvements to it, than the earlier model. This model gave us more accurate results, with a percentile value of 96.38 percent, which proves that it worked well across all the customer types. We also tested the performance of a Random Forest Classifier model as it tends to work well outliers. Yet RF classifier gives a similar average precision to that of a Tuned SVC model, 0.98.

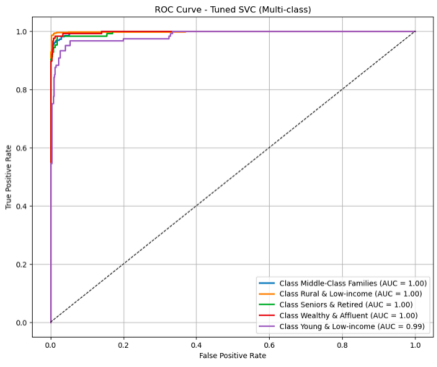


Figure 12: ROC curve of a Tuned SVC model

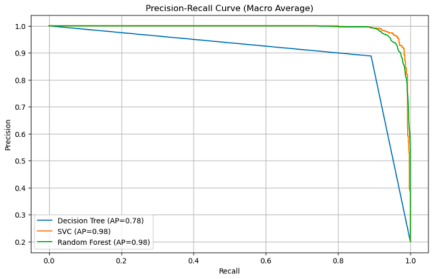


Figure 13: Precision-Recall Curve showing comparison between Tuned Decision Tree, Tuned SVC and a Random Forest Classifier

5 Unsupervised Learning

5.1 Models Used:

- K-Means Clustering
- Agglomerative Clustering

5.1.1 K-Means Clustering

We employed an unsupervised learning model to categorize the classes in the Customer Type. Using Dimensionality Reduction, from 83 features to 2 principal components for clear clustering and visualization of data. The Elbow Method plotted signifies 5 clusters, that align with the known Customer Type. From the confusion matrix after mapping the class names, it proves that one dominant cluster (1735 customers) were identified. Yet there were some overlaps between other segments. After mapping the True Labels with the Clusters labels, it gave an accuracy of 0.59. Encoded labels: ['Middle-Class Families' 'Rural & Low-income' 'Seniors & Retired' 'Wealthy & Affluent' 'Young & Low-income']. **Fig 17** shows that even though few clusters are visible, mathematically these clusters are not accurate and may consists of overlapping (Forced Circular Clusters).

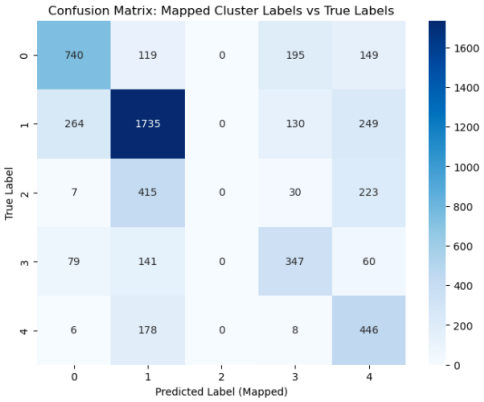


Figure 14: Confusion Matrix of K-Means Clustering

5.1.2 Agglomerative Clustering

Hierarchical Clustering was another model chosen for clustering over K-Means model. Compared to K-Means, Agg. model doesn't show much of an improvement. Out of 4 linkages used, Ward Linkage showed better categorization (5), whereas the rest indicates strong **overlapping** across other clusters. A low **silhouette score** of 0.07, shows weak clustering with only 44 percent accuracy compared to Kmeans that has approx. 59 percent. Rural Classes shows high identification. Along with Middle Class and young class show a balanced proportion. The other classes have low classification, meaning overlapping is significant within these classes (Senior Clusters, Wealthy Classes), they are completely mismatched.

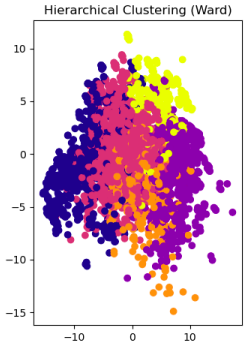


Figure 15: Visualization of Hierarchical Clustering with 5 clusters

6 Deep Learning with Neural Network

The neural network performs much better than most of the traditional models used for analysis. It achieved an accuracy of 96.6 percent. It learns hidden patterns and built in safeguards to prevent from overfitting. The system identified

5 customer types, with precision of 92 percent showing a slight difficulty with one group. Overall, this approach is more accurate than other models.

	precision	recall	f1-score	support
0	0.92	0.99	0.96	260
1	1.00	0.99	0.99	472
2	0.93	0.98	0.96	126
3	0.97	0.92	0.94	126
4	0.99	0.86	0.92	121
accuracy			0.97	1105
macro avg	0.96	0.95	0.95	1105
weighted avg	0.97	0.97	0.97	1105

Figure 16: Precision-Recall info of DL model

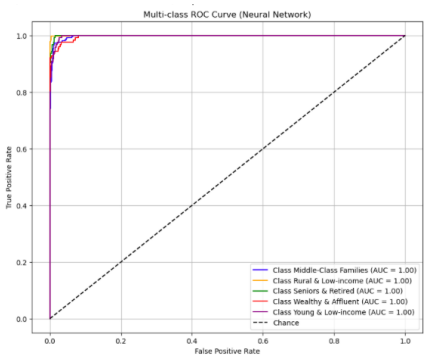


Figure 17: ROC curve for Classification using Deep Learning

7 Compare Clustering vs Classification

While Classification models achieved an accuracy of (88, 95.2, 96.6 percent). The dataset is well suited for supervised learning algorithms, with predefined classes. Yet prone to overfitting, but improved with regularizations. Clustering on the other hand, like only displayed an accuracy of 59 percent, which just proves the dataset is not fit for unsupervised learning algorithms.

8 Conclusion

The analysis efficiently classify customer groups with an accuracy of 96 percent, while also uncovering customer segments using clustering techniques. This gave us some valuable insights into the different customer groups and marketing strategies. Personalized marketing can be done by tailoring the data available of each customer type. Moreover, businesses can leverage this information to identify at-risk customers. This study does point some issues. Such as Data Imbalance, that could be addressed in the further study, and also high skewness towards the 0s, which could be improved using further feature engineering. We could try exploring other clustering methods like DBSCAN and apply models to a larger dataset to prevent overfitting in the future works.

9 Summary

The purpose of the analysis was to give an understanding of the current records in the insurance dataset. We examine over 5500 records, focusing on factors like income, status, education, average age, number of insurance contributions, household demographics etc. Due to the presence of unbalanced number of values in some of the classes, we had to clean and refine the dataset. We then tried to identify the distinct customer patterns and relations that offer strong business value. Customers with higher status and highly educated hold multiple insurance policies. Yet some of the records were mostly skewed towards 0 that provided us with not much significant patterns. So for proper classification of customer types (*Middle-Class Families, Rural & Low-income, Young & Low-income, Seniors & Retired,* and *Wealthy & Affluent*), we relied on most of the customer features. From some of the visualizations done across different features like Income and Education, it is clear that they are related with deeper patterns whereas values like Number of insurance contributions and skill are not closely related comparatively. Yet there are records like Skills against Income which indicates they are not entirely related like it seemed to. The analysis employs several machine learning models to assess and classify the customers based on the given records which shows varying accuracy depending on the models being used. Like Deep Learning models, RandomForestClassifier, Tuned Pipeline SVC shows higher accuracy rate (96-98 percent) in classifying the customers compared to other traditional models used. Middle - Class families (approx. 38 percent of the base) have balanced needs across the market whereas the rest of the classes are comparatively stable with only marginal differences that classify each other. Based on this review, the Insurance agency can afford to establish a strategy that are personalized towards their customers depending on some of the extracted information. The analysis also addresses the probability of overfitting due to limited number of records that were used to work with yet we have tried several pre processing techniques like scaling using PCA and labelencoding to get past those flaws. The complete analysis is available for further examination along with this report. This segmentation framework gives us the unprecedented abiility to move from a reactive to predictive customer interaction.

APPENDIX

Data Analysis Project

- This project analyzes an insurance dataset ('Insurance_Data.csv') to classify and segment customers based on various attributes like income, education, and insurance contributions. We will apply supervised learning for classification and unsupervised learning for clustering to explore customer subtypes and natural groupings.

Introduction

- Understanding customer profiles is essential in the insurance industry for developing tailored services and predicting customer behavior. This project focuses on analyzing an insurance dataset to classify and segment customers based on various attributes. The dataset, 'Insurance_Data.csv,' contains 5,521 observations and 83 variables, including the target variable `Customer_Type`.

- The dataset includes information such as household size, education level, income, social class, and insurance contributions across different policy types like car, life, disability, and property insurance. The primary objective of this analysis is to classify customers into relevant subtypes and identify natural groupings using clustering techniques.

- This project involves both supervised learning for classification and unsupervised learning for segmentation. The analysis will include data exploration, cleaning, feature engineering, model training, and evaluation. Finally, we will perform clustering to explore how customer types group naturally in the dataset and compare the results with the classification outcomes.

Load The Dataset

- Importing the dataset 'Insurance_Data.CSV' for exploration.
- Summarize the information in the dataset

In [58]:
```
import pandas as pd
import numpy as np
import os
import shutil

print(os.listdir())
```

```
['PandasPractice.ipynb', 'SupervisedClassification2_Practice_advance_Sols.
ipynb', 'Lab 7_Deep Learning with TensorFlow.ipynb', 'CleaningPractice_Sol
s.ipynb', 'Neural Networks with Scikit-learn.ipynb', 'FeaturesAdultPractic
eSols.ipynb', 'SupervisedClassification2_Examples.ipynb', 'PandasPractice_
Sols.ipynb', 'CleaningExamples (1).ipynb', 'DataMiningAssignment.ipynb', '
Machine Learning Pipeline.ipynb', 'DL with TF Google.ipynb', 'UnSupervised
ClusteringExamples.ipynb', 'Insurance_Data.csv', '.ipynb_checkpoints', 'DL
with TF.ipynb', 'ClassificationExamples.ipynb']
```

In [100…
```
path = 'Insurance_Data.csv'
Insurance_df = pd.read_csv(path)
```

Read the Dataset

- First, we obtained the dataset, and now we need to analyze its contents, focusing on the features and attributes it contains.

We used the info() method to gain an overview of the dataset. This includes details about the number of entries, the data types of each feature, and the count of non-null values for each attribute. This helps us understand the structure of the dataset and identify any potential issues, such as missing data

In [710…
```
Insurance_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5521 entries, 0 to 5520
Data columns (total 83 columns):
 #   Column                                  Non-Null Count  Dtyp
e
---  ------                                  --------------  ----
-
 0   Customer_Type                           5521 non-null   obje
ct
 1   Number_of_Houses                        5521 non-null   int6
4
 2   Avg_Household_Size                      5521 non-null   int6
4
 3   Avg_Age                                 5507 non-null   obje
ct
 4   Household_Profile                       5521 non-null   obje
ct
 5   Married                                 5521 non-null   int6
4
 6   Living_Together                         5521 non-null   int6
4
 7   Other_Relation                          5521 non-null   int6
4
 8   Singles                                 5521 non-null   int6
4
 9   Household_Without_Children              5521 non-null   int6
4
```

```
 10  Household_With_Children                 5521 non-null   int6
4
 11  High_Education_Level                    5521 non-null   int6
4
 12  Medium_Education_Level                  5521 non-null   int6
4
 13  Low_Education_Level                     5521 non-null   int6
4
 14  High_Status                             5521 non-null   int6
4
 15  Entrepreneur                            5521 non-null   int6
4
 16  Farmer                                  5521 non-null   int6
4
 17  Middle_Management                       5521 non-null   int6
4
 18  Skilled_Labourers                       5521 non-null   int6
4
 19  Unskilled_Labourers                     5521 non-null   int6
4
 20  Social_Class_A                          5521 non-null   int6
4
 21  Social_Class_B1                         5521 non-null   int6
4
 22  Social_Class_B2                         5521 non-null   int6
4
 23  Social_Class_C                          5521 non-null   int6
4
 24  Social_Class_D                          5521 non-null   int6
4
 25  Rented_House                            5521 non-null   int6
4
 26  Home_Owner                              5521 non-null   int6
4
 27  Owns_One_Car                            5521 non-null   int6
4
 28  Owns_Two_Cars                           5521 non-null   int6
4
 29  Owns_No_Car                             5521 non-null   int6
4
 30  National_Health_Insurance               5521 non-null   int6
4
 31  Private_Health_Insurance                5521 non-null   int6
4
 32  Income_Less_Than_30K                    5521 non-null   int6
4
 33  Income_30K_to_45K                       5521 non-null   int6
4
 34  Income_45K_to_75K                       5521 non-null   int6
4
 35  Income_75K_to_122K                      5521 non-null   int6
4
 36  Income_Above_123K                       5521 non-null   int6
```

```
4
 37  Average_Income                                  5521 non-null   int6
4
 38  Purchasing_Power_Class                          5521 non-null   int6
4
 39  Private_Third_Party_Insurance_Contribution      5512 non-null   obje
ct
 40  Business_Third_Party_Insurance_Contribution     5521 non-null   obje
ct
 41  Agricultural_Third_Party_Insurance_Contribution 5521 non-null   obje
ct
 42  Car_Policy_Contribution                         5521 non-null   int6
4
 43  Delivery_Van_Policy_Contribution                5502 non-null   floa
t64
 44  Motorcycle_Scooter_Policy_Contribution          5521 non-null   int6
4
 45  Lorry_Policy_Contribution                       5521 non-null   int6
4
 46  Trailer_Policy_Contribution                     5502 non-null   floa
t64
 47  Tractor_Policy_Contribution                     5521 non-null   int6
4
 48  Agricultural_Machine_Policy_Contribution        5521 non-null   int6
4
 49  Moped_Policy_Contribution                       5469 non-null   floa
t64
 50  Life_Insurance_Contribution                     5461 non-null   floa
t64
 51  Private_Accident_Insurance_Contribution         5469 non-null   floa
t64
 52  Family_Accident_Insurance_Contribution          5469 non-null   floa
t64
 53  Disability_Insurance_Contribution               5449 non-null   floa
t64
 54  Fire_Insurance_Contribution                     5466 non-null   floa
t64
 55  Surfboard_Insurance_Contribution                5469 non-null   floa
t64
 56  Boat_Insurance_Contribution                     5469 non-null   floa
t64
 57  Bicycle_Insurance_Contribution                  5469 non-null   floa
t64
 58  Property_Insurance_Contribution                 5469 non-null   floa
t64
 59  Social_Security_Insurance_Contribution          5469 non-null   floa
t64
 60  Number_Private_Third_Party_Insurance            5469 non-null   floa
t64
 61  Number_Business_Third_Party_Insurance           5469 non-null   floa
t64
 62  Number_Agricultural_Third_Party_Insurance       5469 non-null   floa
t64
```

```
 63  Number_Car_Policies                          5521 non-null   int6
4
 64  Number_Delivery_Van_Policies                 5521 non-null   int6
4
 65  Number_Motorcycle_Scooter_Policies           5521 non-null   int6
4
 66  Number_Lorry_Policies                        5521 non-null   int6
4
 67  Number_Trailer_Policies                      5489 non-null   floa
t64
 68  Number_Tractor_Policies                      5489 non-null   floa
t64
 69  Number_Agricultural_Machine_Policies         5489 non-null   floa
t64
 70  Number_Moped_Policies                        5489 non-null   floa
t64
 71  Number_Life_Insurances                       5489 non-null   floa
t64
 72  Number_Private_Accident_Insurances           5489 non-null   floa
t64
 73  Number_Family_Accident_Insurances            5489 non-null   floa
t64
 74  Number_Disability_Insurances                 5521 non-null   int6
4
 75  Number_Fire_Insurances                       5521 non-null   int6
4
 76  Number_Surfboard_Insurances                  5521 non-null   int6
4
 77  Number_Boat_Insurances                       5521 non-null   int6
4
 78  Number_Bicycle_Insurances                    5521 non-null   int6
4
 79  Number_Property_Insurances                   5472 non-null   floa
t64
 80  Number_Social_Security_Insurances            5472 non-null   floa
t64
 81  Number_Mobile_Home_Policies                  5447 non-null   floa
t64
 82  Mobile_Home_Policies                         5456 non-null   obje
ct
dtypes: float64(26), int64(50), object(7)
memory usage: 3.5+ MB
```

We used the describe() method to generate a statistical summary of the dataset. This provides key metrics such as the mean, standard deviation, minimum, maximum, and quartiles for numerical features.

In [713…

```
Insurance_df.describe()
```

Out[713…

	Number_of_Houses	Avg_Household_Size	Married	Living_Together	Ot
count	5521.000000	5521.000000	5521.000000	5521.000000	
mean	1.111393	2.681217	6.188372	0.883354	
std	0.410128	0.790448	1.902710	0.967486	
min	1.000000	1.000000	0.000000	0.000000	
25%	1.000000	2.000000	5.000000	0.000000	
50%	1.000000	3.000000	6.000000	1.000000	
75%	1.000000	3.000000	7.000000	1.000000	
max	10.000000	5.000000	9.000000	7.000000	

8 rows × 76 columns

In [715…
```
Insurance_df.columns
```

Out[715…
```
Index(['Customer_Type', 'Number_of_Houses', 'Avg_Household_Size', 'Avg_A
ge',
       'Household_Profile', 'Married', 'Living_Together', 'Other_Relatio
n',
       'Singles', 'Household_Without_Children', 'Household_With_Childre
n',
       'High_Education_Level', 'Medium_Education_Level', 'Low_Education_
Level',
       'High_Status', 'Entrepreneur', 'Farmer', 'Middle_Management',
       'Skilled_Labourers', 'Unskilled_Labourers', 'Social_Class_A',
       'Social_Class_B1', 'Social_Class_B2', 'Social_Class_C',
       'Social_Class_D', 'Rented_House', 'Home_Owner', 'Owns_One_Car',
       'Owns_Two_Cars', 'Owns_No_Car', 'National_Health_Insurance',
       'Private_Health_Insurance', 'Income_Less_Than_30K', 'Income_30K_t
o_45K',
       'Income_45K_to_75K', 'Income_75K_to_122K', 'Income_Above_123K',
       'Average_Income', 'Purchasing_Power_Class',
       'Private_Third_Party_Insurance_Contribution',
       'Business_Third_Party_Insurance_Contribution',
       'Agricultural_Third_Party_Insurance_Contribution',
       'Car_Policy_Contribution', 'Delivery_Van_Policy_Contribution',
       'Motorcycle_Scooter_Policy_Contribution', 'Lorry_Policy_Contribut
ion',
       'Trailer_Policy_Contribution', 'Tractor_Policy_Contribution',
       'Agricultural_Machine_Policy_Contribution', 'Moped_Policy_Contrib
ution',
       'Life_Insurance_Contribution',
       'Private_Accident_Insurance_Contribution',
       'Family_Accident_Insurance_Contribution',
       'Disability_Insurance_Contribution', 'Fire_Insurance_Contributio
n',
       'Surfboard_Insurance_Contribution', 'Boat_Insurance_Contributio
```

```
n',
       'Bicycle_Insurance_Contribution', 'Property_Insurance_Contributio
n',
       'Social_Security_Insurance_Contribution',
       'Number_Private_Third_Party_Insurance',
       'Number_Business_Third_Party_Insurance',
       'Number_Agricultural_Third_Party_Insurance', 'Number_Car_Policie
s',
       'Number_Delivery_Van_Policies', 'Number_Motorcycle_Scooter_Polici
es',
       'Number_Lorry_Policies', 'Number_Trailer_Policies',
       'Number_Tractor_Policies', 'Number_Agricultural_Machine_Policie
s',
       'Number_Moped_Policies', 'Number_Life_Insurances',
       'Number_Private_Accident_Insurances',
       'Number_Family_Accident_Insurances', 'Number_Disability_Insurance
s',
       'Number_Fire_Insurances', 'Number_Surfboard_Insurances',
       'Number_Boat_Insurances', 'Number_Bicycle_Insurances',
       'Number_Property_Insurances', 'Number_Social_Security_Insurance
s',
       'Number_Mobile_Home_Policies', 'Mobile_Home_Policies'],
      dtype='object')
```

Seperate the Categorical Columns and Numerical Columns

In [102…
```
Categorical_Columns = Insurance_df.select_dtypes(include = 'object').colu
```

In [104…
```
Numerical_Columns = Insurance_df.select_dtypes(include = 'number').column
```

In [15]:
```
print(Insurance_df.shape)
```

```
(5521, 83)
```

In [186…
```
Insurance_df[Categorical_Columns].nunique()
```

Out[186…
```
Customer_Type                                    5
Avg_Age                                          6
Household_Profile                               10
Private_Third_Party_Insurance_Contribution       4
Business_Third_Party_Insurance_Contribution      7
Agricultural_Third_Party_Insurance_Contribution  4
Mobile_Home_Policies                             2
dtype: int64
```

In [120…
```
# To view the first 5 rows from the dataset, to have an idea of the value
Insurance_df.head()
```

Out[120…

	Customer_Type	Number_of_Houses	Avg_Household_Size	Avg_Age	Household_
0	Rural & Low-income	1	3	30-40 years	Family with
1	Rural & Low-income	1	2	30-40 years	Family with
2	Rural & Low-income	1	2	30-40 years	Family with
3	Middle-Class Families	1	3	40-50 years	Averag
4	Rural & Low-income	1	4	30-40 years	

5 rows × 83 columns

In [120…

```
insurance_df.info()
```

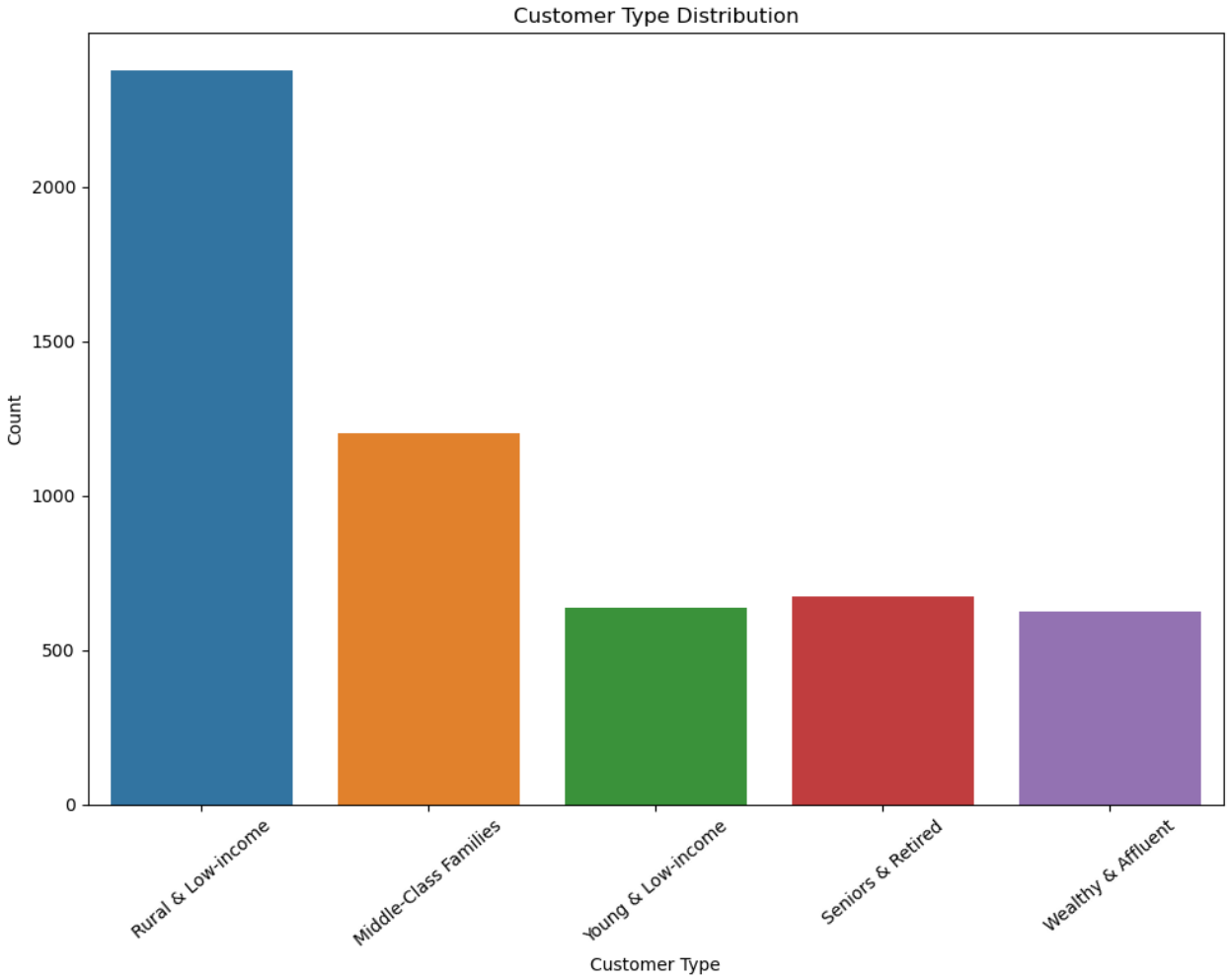
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5521 entries, 0 to 5520
Data columns (total 83 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Customer_Type                           5521 non-null   object
 1   Number_of_Houses                        5521 non-null   int64
 2   Avg_Household_Size                      5521 non-null   int64
 3   Avg_Age                                 5507 non-null   object
 4   Household_Profile                       5521 non-null   object
 5   Married                                 5521 non-null   int64
 6   Living_Together                         5521 non-null   int64
 7   Other_Relation                          5521 non-null   int64
 8   Singles                                 5521 non-null   int64
 9   Household_Without_Children              5521 non-null   int64
 10  Household_With_Children                 5521 non-null   int64
 11  High_Education_Level                    5521 non-null   int64
 12  Medium_Education_Level                  5521 non-null   int64
```

```
 13  Low_Education_Level                          5521 non-null   int6
4
 14  High_Status                                  5521 non-null   int6
4
 15  Entrepreneur                                 5521 non-null   int6
4
 16  Farmer                                       5521 non-null   int6
4
 17  Middle_Management                            5521 non-null   int6
4
 18  Skilled_Labourers                            5521 non-null   int6
4
 19  Unskilled_Labourers                          5521 non-null   int6
4
 20  Social_Class_A                               5521 non-null   int6
4
 21  Social_Class_B1                              5521 non-null   int6
4
 22  Social_Class_B2                              5521 non-null   int6
4
 23  Social_Class_C                               5521 non-null   int6
4
 24  Social_Class_D                               5521 non-null   int6
4
 25  Rented_House                                 5521 non-null   int6
4
 26  Home_Owner                                   5521 non-null   int6
4
 27  Owns_One_Car                                 5521 non-null   int6
4
 28  Owns_Two_Cars                                5521 non-null   int6
4
 29  Owns_No_Car                                  5521 non-null   int6
4
 30  National_Health_Insurance                    5521 non-null   int6
4
 31  Private_Health_Insurance                     5521 non-null   int6
4
 32  Income_Less_Than_30K                         5521 non-null   int6
4
 33  Income_30K_to_45K                            5521 non-null   int6
4
 34  Income_45K_to_75K                            5521 non-null   int6
4
 35  Income_75K_to_122K                           5521 non-null   int6
4
 36  Income_Above_123K                            5521 non-null   int6
4
 37  Average_Income                               5521 non-null   int6
4
 38  Purchasing_Power_Class                       5521 non-null   int6
4
 39  Private_Third_Party_Insurance_Contribution   5512 non-null   obje
```

```
ct
 40  Business_Third_Party_Insurance_Contribution      5521 non-null   obje
ct
 41  Agricultural_Third_Party_Insurance_Contribution  5521 non-null   obje
ct
 42  Car_Policy_Contribution                          5521 non-null   int6
4
 43  Delivery_Van_Policy_Contribution                 5502 non-null   floa
t64
 44  Motorcycle_Scooter_Policy_Contribution           5521 non-null   int6
4
 45  Lorry_Policy_Contribution                        5521 non-null   int6
4
 46  Trailer_Policy_Contribution                      5502 non-null   floa
t64
 47  Tractor_Policy_Contribution                      5521 non-null   int6
4
 48  Agricultural_Machine_Policy_Contribution         5521 non-null   int6
4
 49  Moped_Policy_Contribution                        5469 non-null   floa
t64
 50  Life_Insurance_Contribution                      5461 non-null   floa
t64
 51  Private_Accident_Insurance_Contribution          5469 non-null   floa
t64
 52  Family_Accident_Insurance_Contribution           5469 non-null   floa
t64
 53  Disability_Insurance_Contribution                5449 non-null   floa
t64
 54  Fire_Insurance_Contribution                      5466 non-null   floa
t64
 55  Surfboard_Insurance_Contribution                 5469 non-null   floa
t64
 56  Boat_Insurance_Contribution                      5469 non-null   floa
t64
 57  Bicycle_Insurance_Contribution                   5469 non-null   floa
t64
 58  Property_Insurance_Contribution                  5469 non-null   floa
t64
 59  Social_Security_Insurance_Contribution           5469 non-null   floa
t64
 60  Number_Private_Third_Party_Insurance             5469 non-null   floa
t64
 61  Number_Business_Third_Party_Insurance            5469 non-null   floa
t64
 62  Number_Agricultural_Third_Party_Insurance        5469 non-null   floa
t64
 63  Number_Car_Policies                              5521 non-null   int6
4
 64  Number_Delivery_Van_Policies                     5521 non-null   int6
4
 65  Number_Motorcycle_Scooter_Policies               5521 non-null   int6
4
```

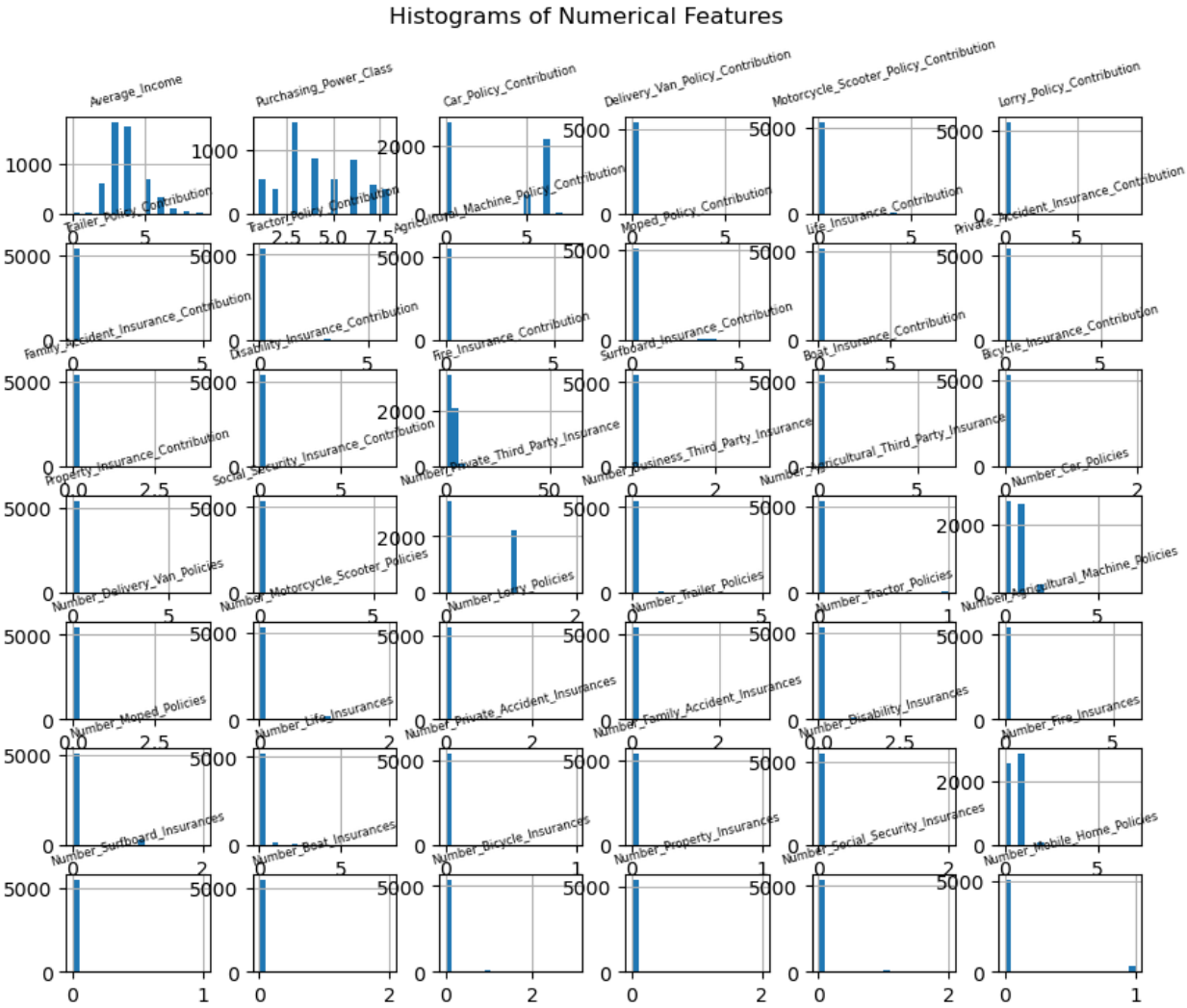
```
 66  Number_Lorry_Policies                       5521 non-null   int6
4
 67  Number_Trailer_Policies                     5489 non-null   floa
t64
 68  Number_Tractor_Policies                     5489 non-null   floa
t64
 69  Number_Agricultural_Machine_Policies        5489 non-null   floa
t64
 70  Number_Moped_Policies                       5489 non-null   floa
t64
 71  Number_Life_Insurances                      5489 non-null   floa
t64
 72  Number_Private_Accident_Insurances          5489 non-null   floa
t64
 73  Number_Family_Accident_Insurances           5489 non-null   floa
t64
 74  Number_Disability_Insurances                5521 non-null   int6
4
 75  Number_Fire_Insurances                      5521 non-null   int6
4
 76  Number_Surfboard_Insurances                 5521 non-null   int6
4
 77  Number_Boat_Insurances                      5521 non-null   int6
4
 78  Number_Bicycle_Insurances                   5521 non-null   int6
4
 79  Number_Property_Insurances                  5472 non-null   floa
t64
 80  Number_Social_Security_Insurances           5472 non-null   floa
t64
 81  Number_Mobile_Home_Policies                 5447 non-null   floa
t64
 82  Mobile_Home_Policies                        5456 non-null   obje
ct
dtypes: float64(26), int64(50), object(7)
memory usage: 3.5+ MB
```

In [24]:
```
# Display the count of distribution of each value in Customer Type

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (10, 8))
sns.countplot(x = 'Customer_Type', hue = 'Customer_Type', data = Insuranc
plt.title("Customer Type Distribution")
plt.xlabel('Customer Type')
plt.xticks(rotation = 40)
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

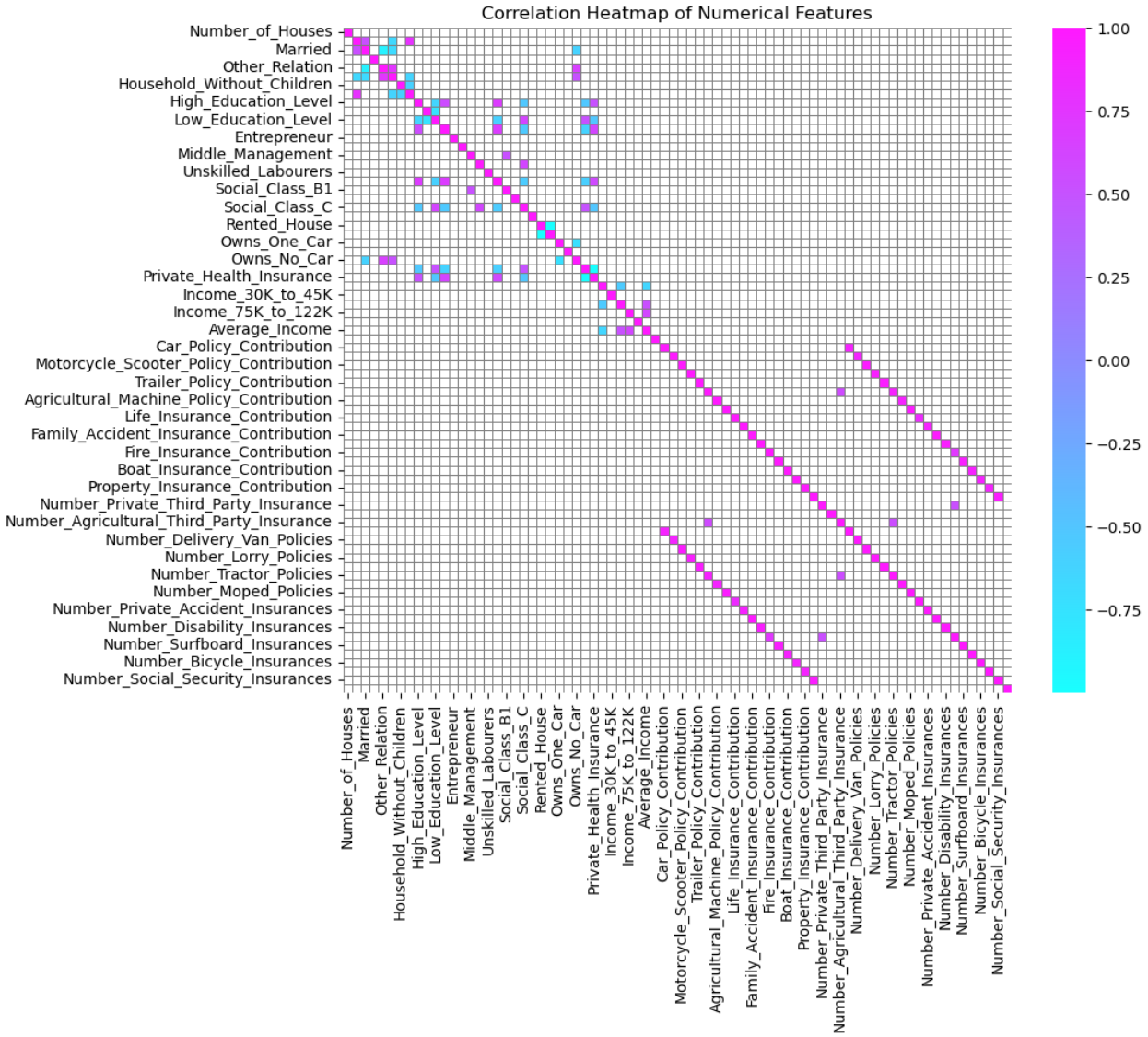


In [120…

```
Insurance_df[Numerical_Columns].iloc[:,34:].hist(figsize = (10, 8), bins
for i, ax in enumerate(plt.gcf().axes):
    ax.set_title(Insurance_df[Numerical_Columns].columns[34+i], rotation=

plt.suptitle("Histograms of Numerical Features")
plt.show()
```

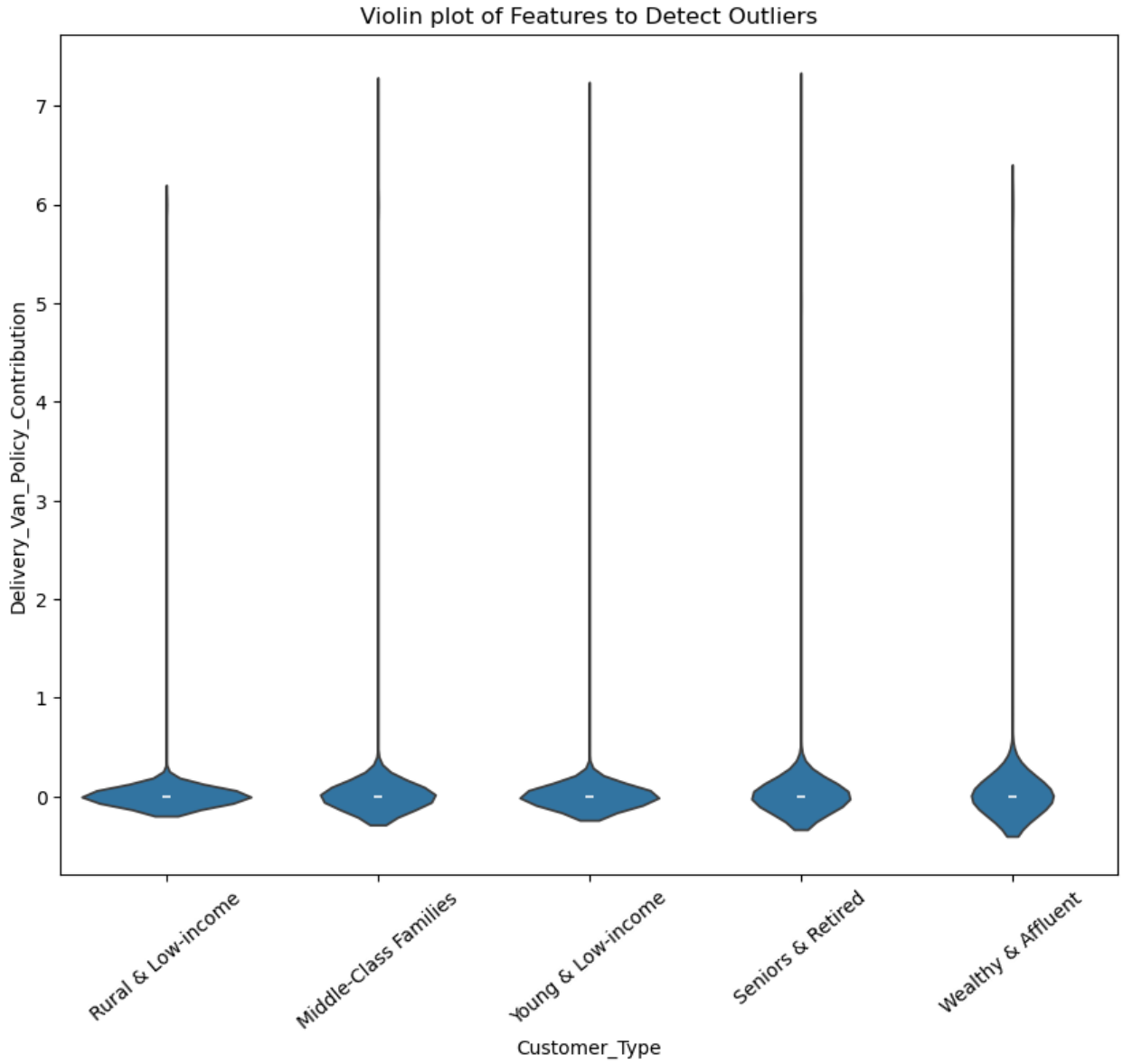


In [55]:
```
# To identify the relation between the numeric features within the datase
import seaborn as sns

#for readability, a threshold of 0.5 was used for indicating the correlat
corr_matrix = Insurance_df[Numerical_Columns].corr()
strong_matrix = corr_matrix[(corr_matrix > 0.5)|(corr_matrix < -0.5)]

plt.figure(figsize=(10, 8))
sns.heatmap(strong_matrix, cmap = 'cool',annot = False ,alpha = 0.9, line
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



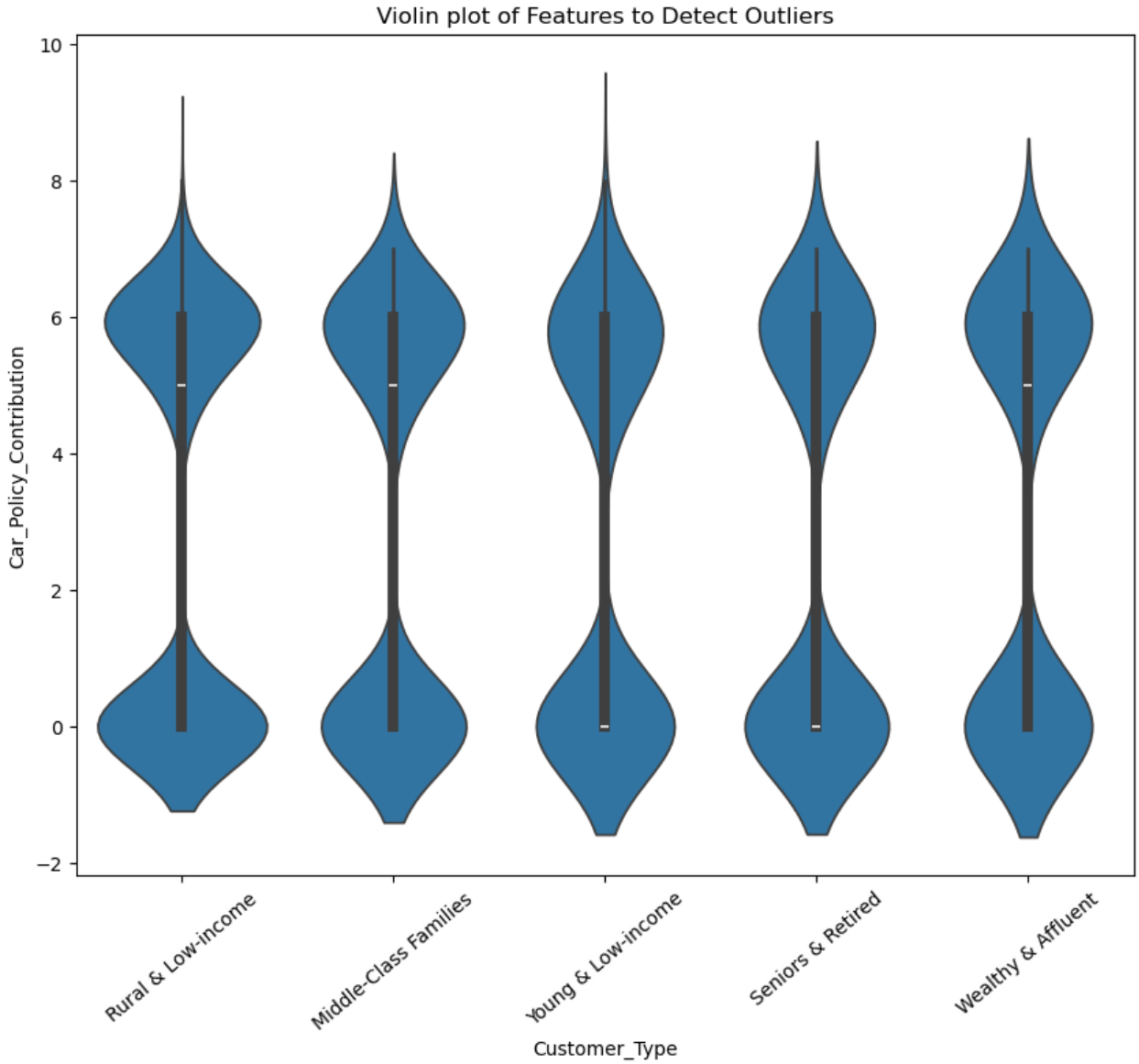
In [598…

```
plt.figure(figsize=(10,8))
sns.violinplot(x = Insurance_df['Customer_Type'], y = Insurance_df['Deliv
plt.xlabel('Customer_Type')
plt.ylabel('Delivery_Van_Policy_Contribution')
plt.xticks(rotation = 40)
plt.title("Violin plot of Features to Detect Outliers")
plt.show()
```



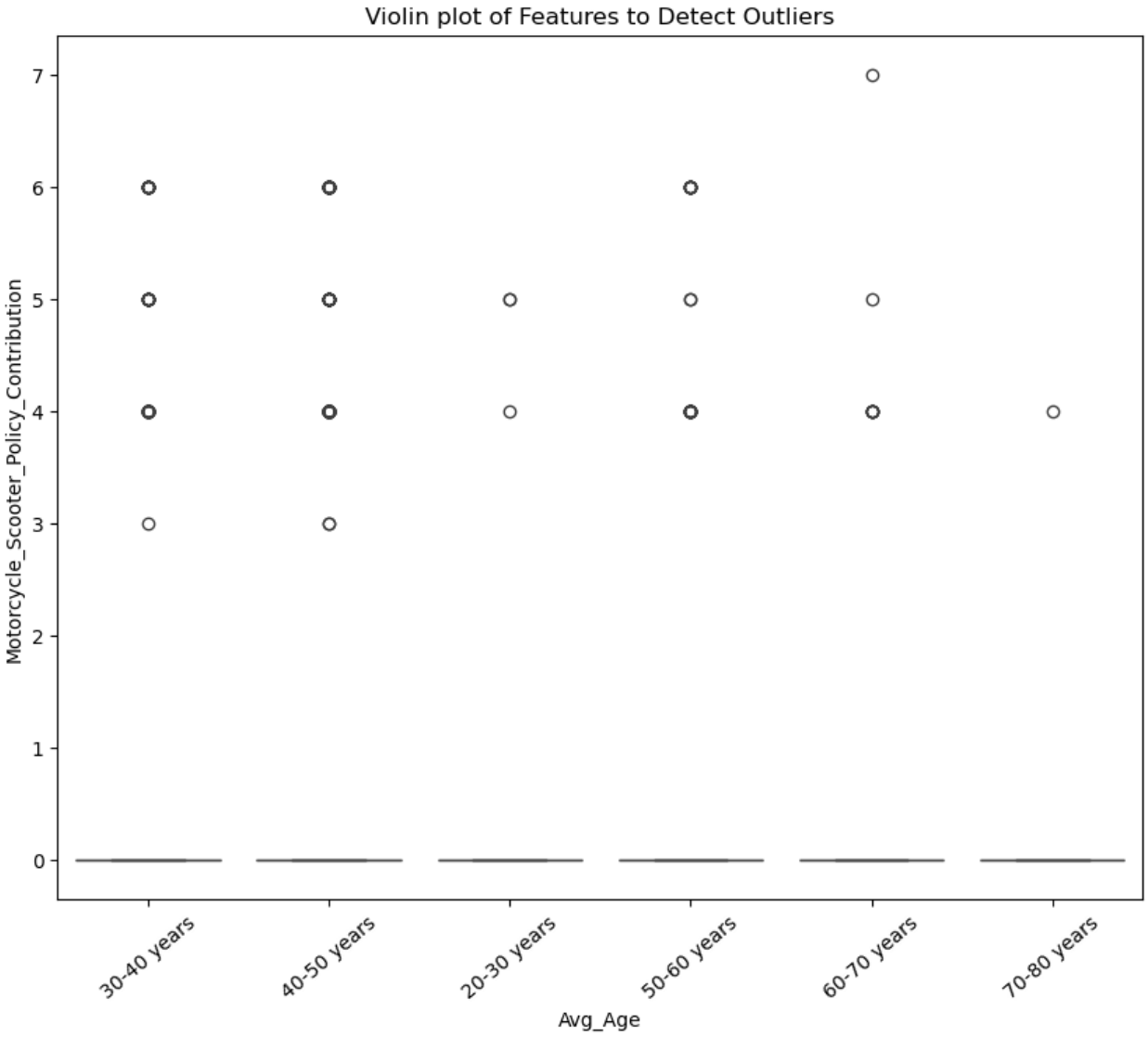
In [596…

```
plt.figure(figsize=(10,8))
sns.violinplot(x = Insurance_df['Customer_Type'], y = Insurance_df['Car_P
plt.xlabel('Customer_Type')
plt.ylabel('Car_Policy_Contribution')
plt.xticks(rotation = 40)
plt.title("Violin plot of Features to Detect Outliers")
plt.show()
```



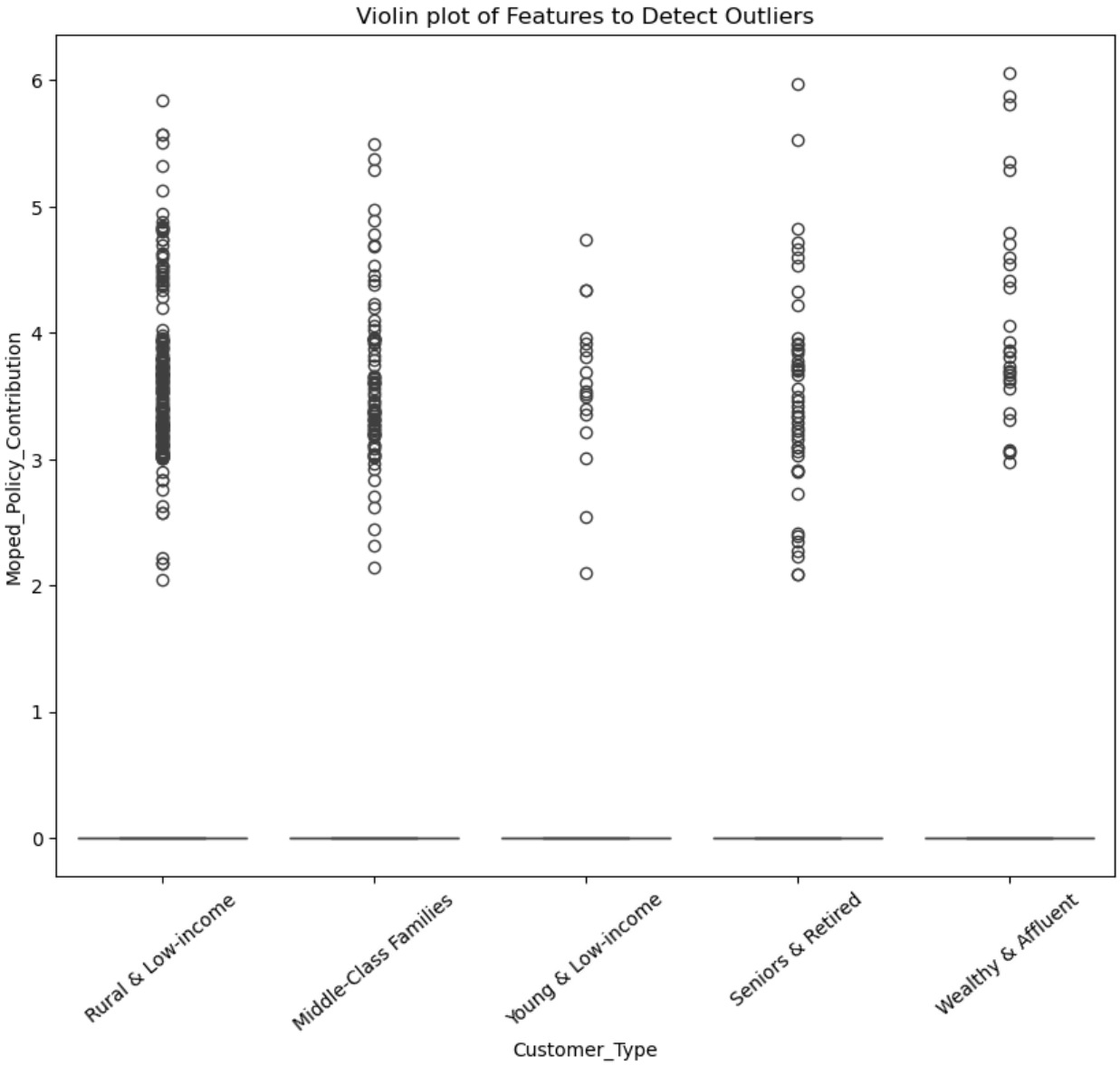
In [775…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Avg_Age'], y = Insurance_df['Motorcycle_Sco
plt.xlabel('Avg_Age')
plt.ylabel('Motorcycle_Scooter_Policy_Contribution')
plt.xticks(rotation = 40)
plt.title("Violin plot of Features to Detect Outliers")
plt.show()
```



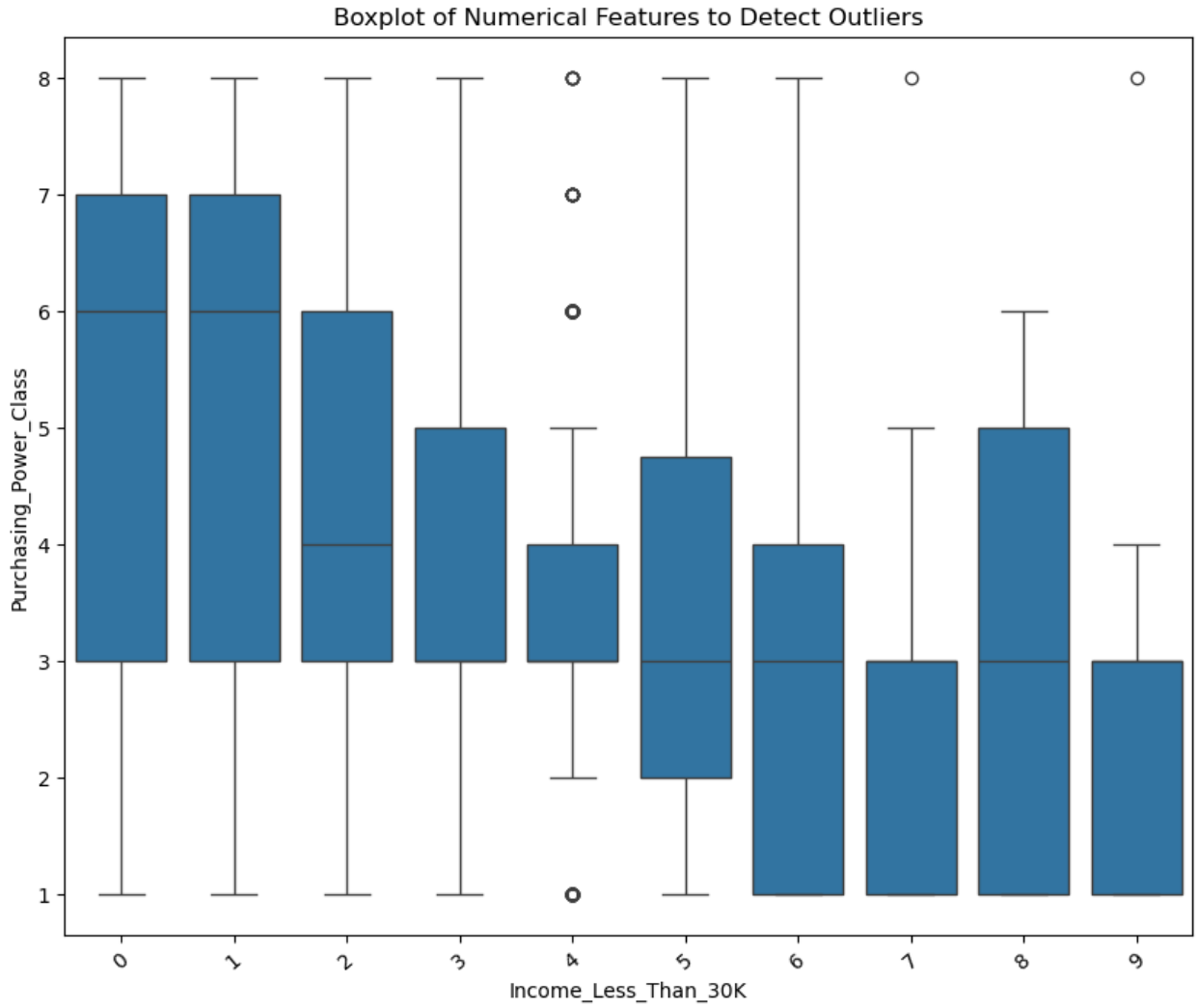
In [777…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Customer_Type'], y = Insurance_df['Moped_Po
plt.xlabel('Customer_Type')
plt.ylabel('Moped_Policy_Contribution')
plt.xticks(rotation = 40)
plt.title("Violin plot of Features to Detect Outliers")
plt.show()
```

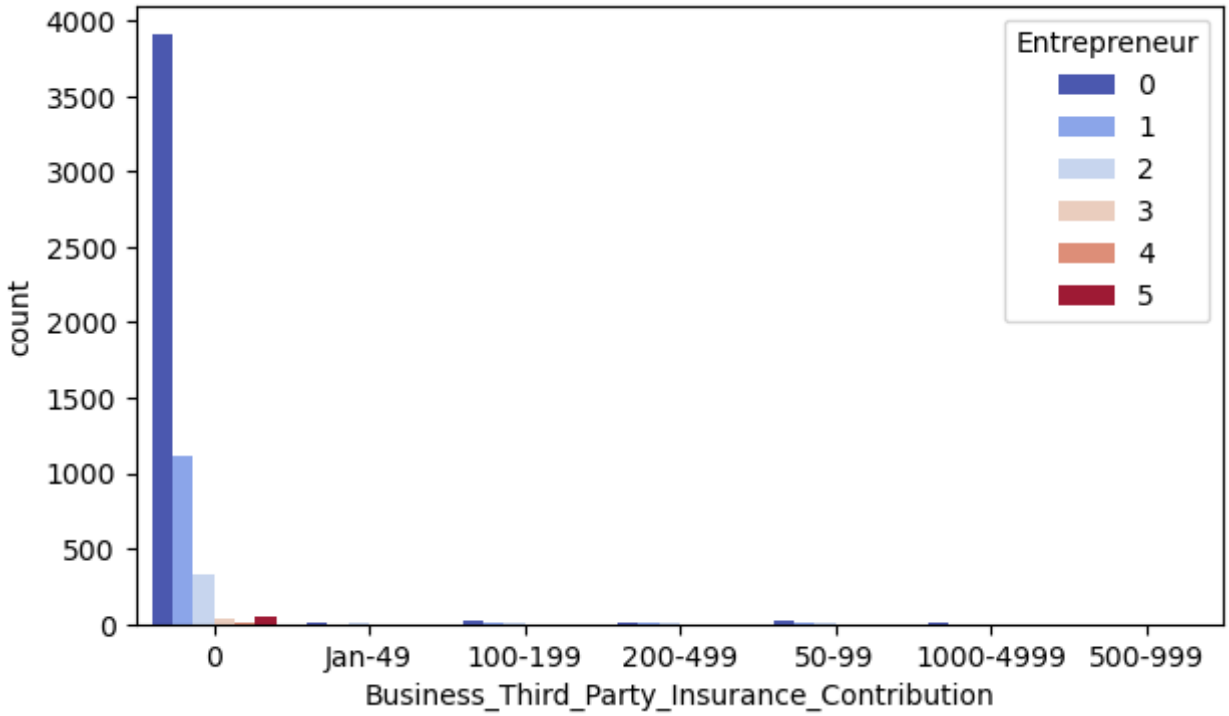


In [600…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Income_Less_Than_30K'], y = Insurance_df['P
plt.xlabel('Income_Less_Than_30K')
plt.ylabel('Purchasing_Power_Class')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```

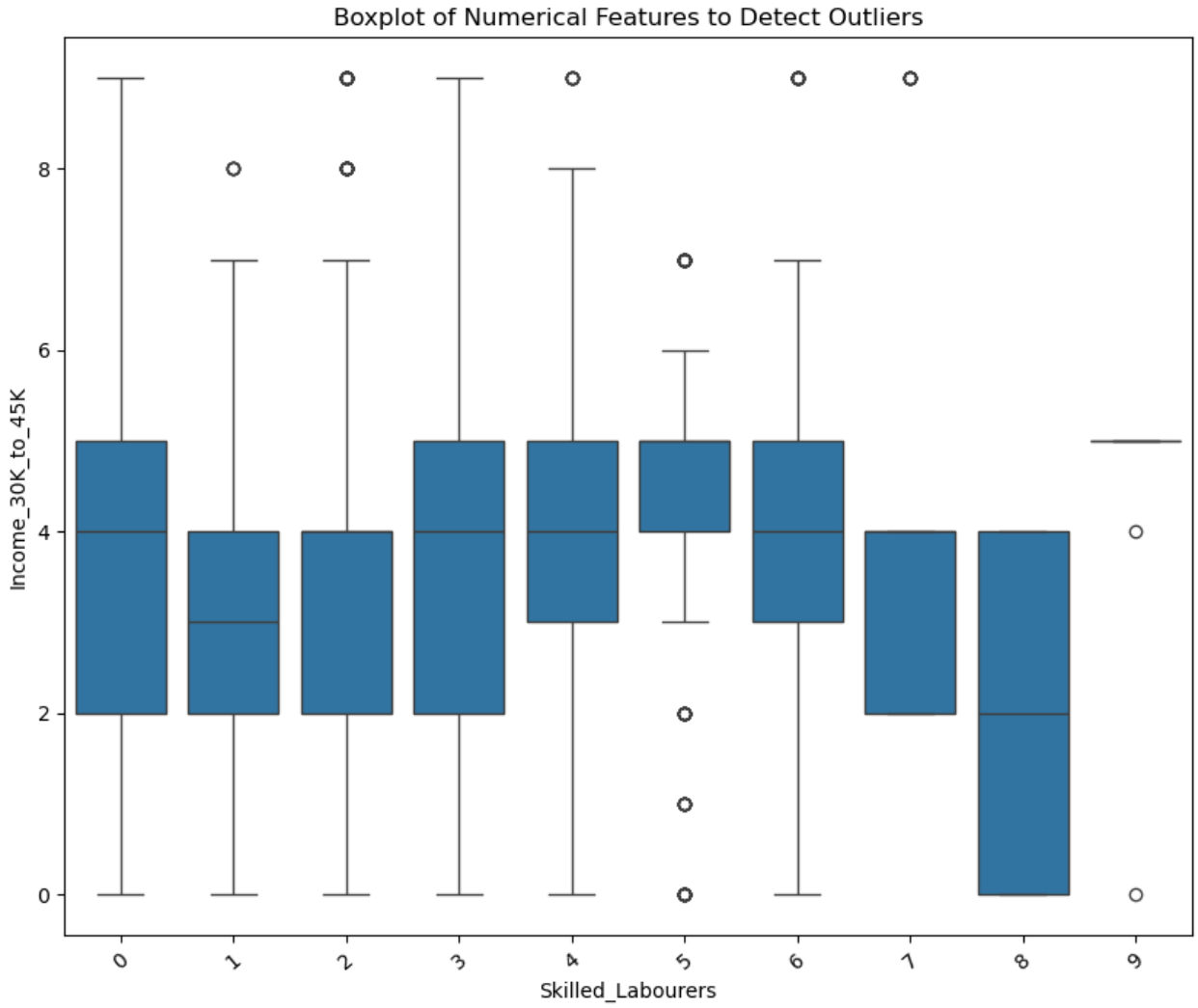


In [860…
```
plt.figure(figsize=(7,4))
sns.countplot(x="Business_Third_Party_Insurance_Contribution", hue='Entre
plt.show()
```



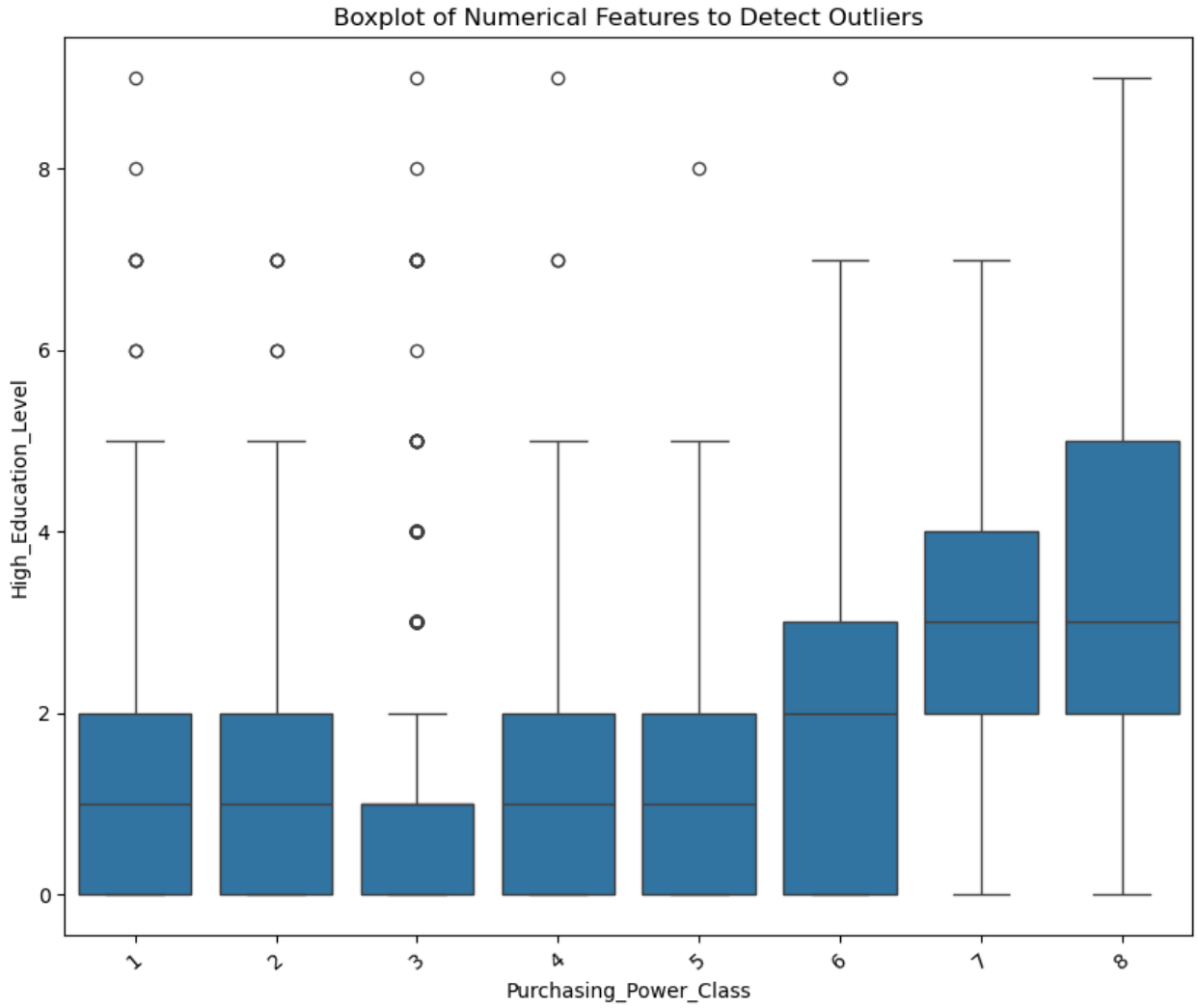
In [650…

```
plt.figure(figsize=(10,8))
sns.boxplot(x=Insurance_df['Skilled_Labourers'], y = Insurance_df['Income
plt.xlabel('Skilled_Labourers')
plt.ylabel('Income_30K_to_45K')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



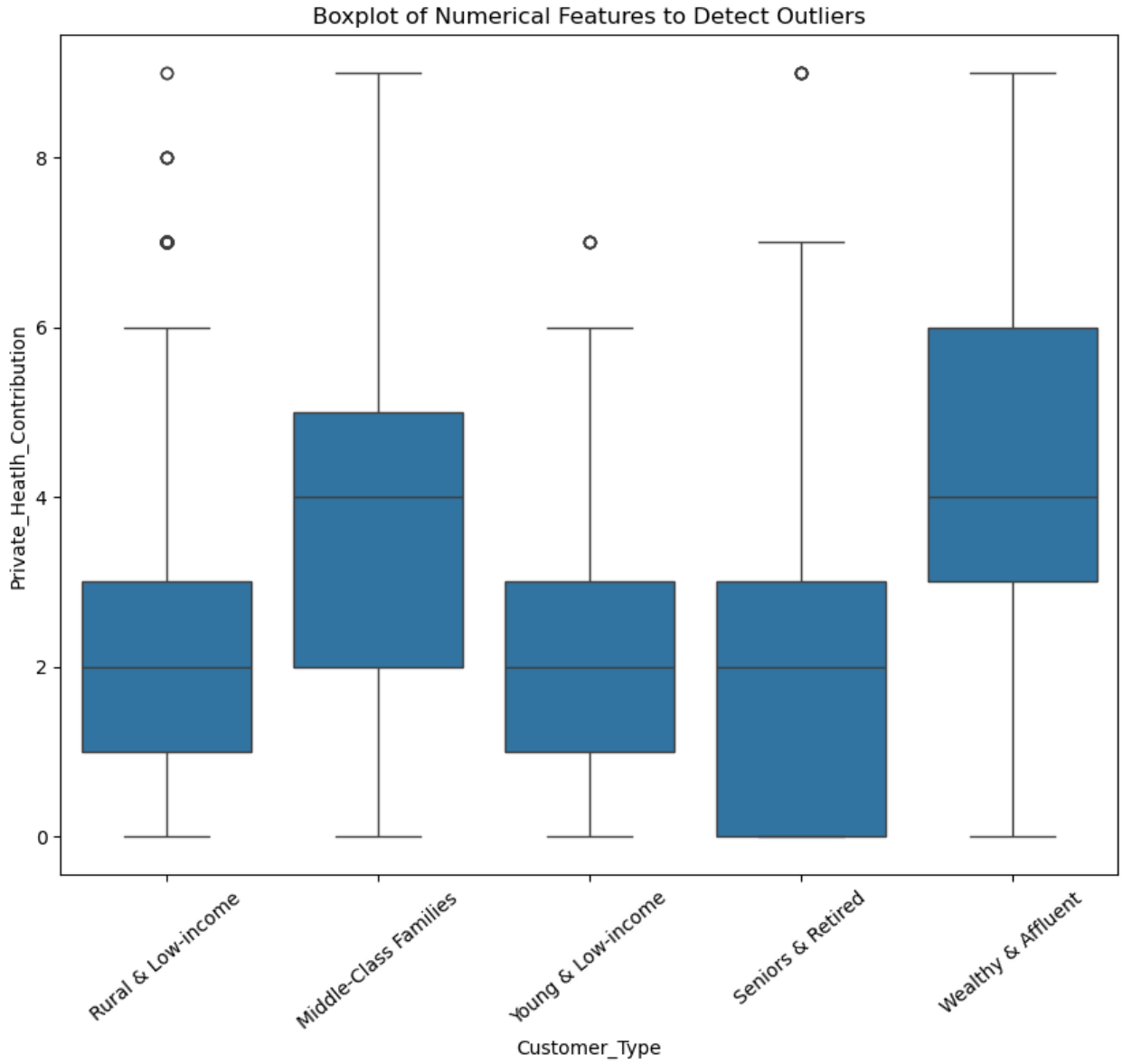
In [690…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Purchasing_Power_Class'], y = Insurance_df[
plt.ylabel('High_Education_Level')
plt.xlabel('Purchasing_Power_Class')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



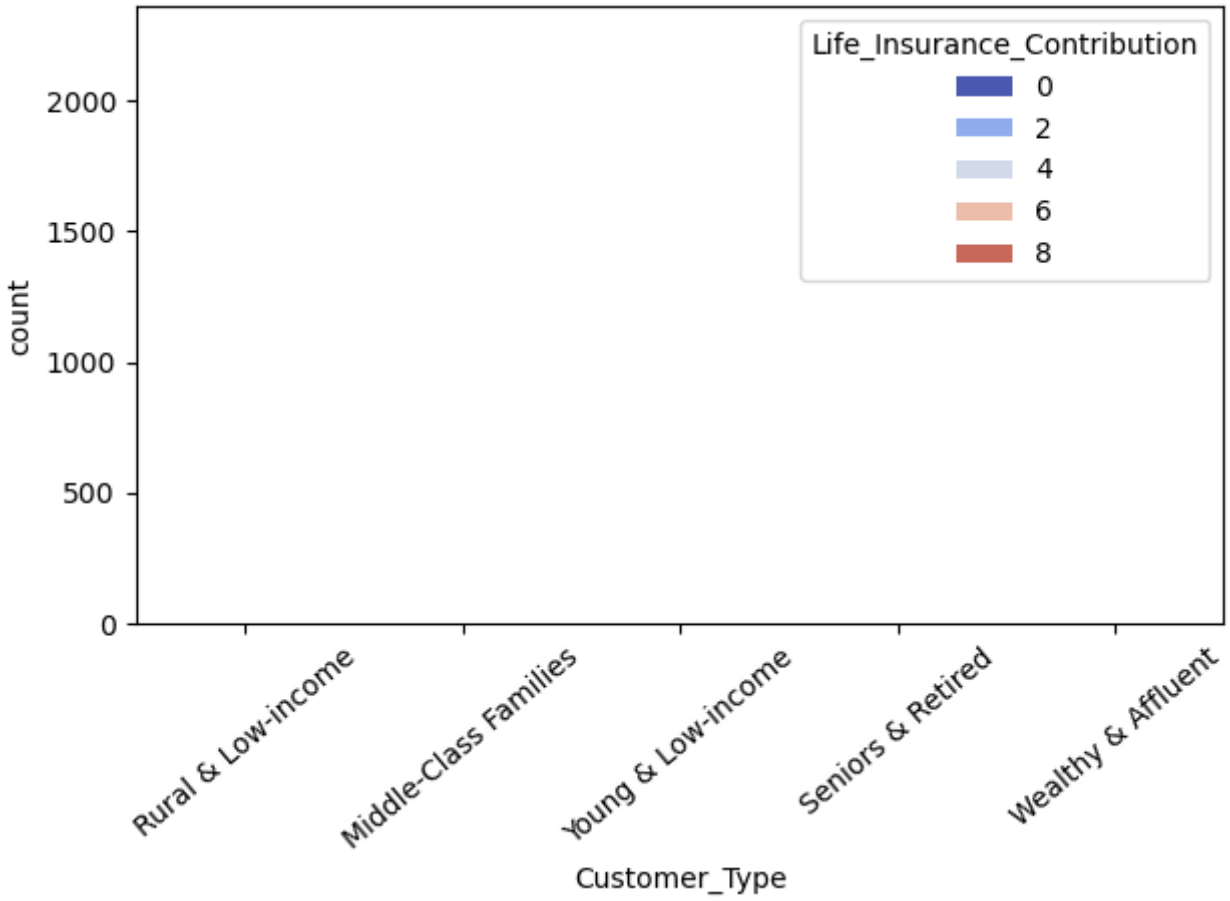
In [692…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Customer_Type'], y = Insurance_df['Private_
plt.xlabel('Customer_Type')
plt.ylabel('Private_Heatlh_Contribution')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



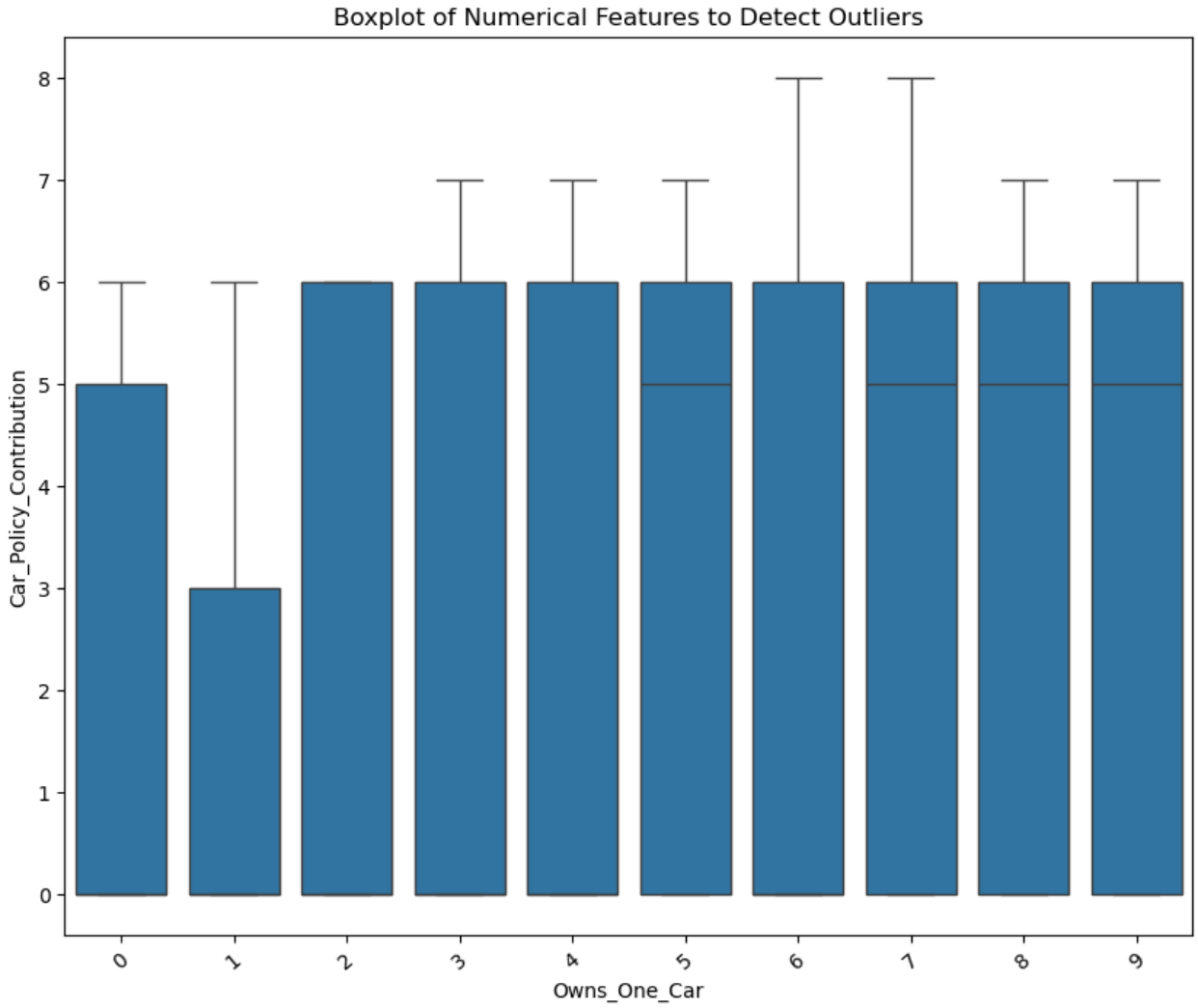
In [868…

```
plt.figure(figsize=(7,4))
sns.countplot(x="Customer_Type", hue='Life_Insurance_Contribution', palet
plt.xticks(rotation = 40)
plt.show()
```



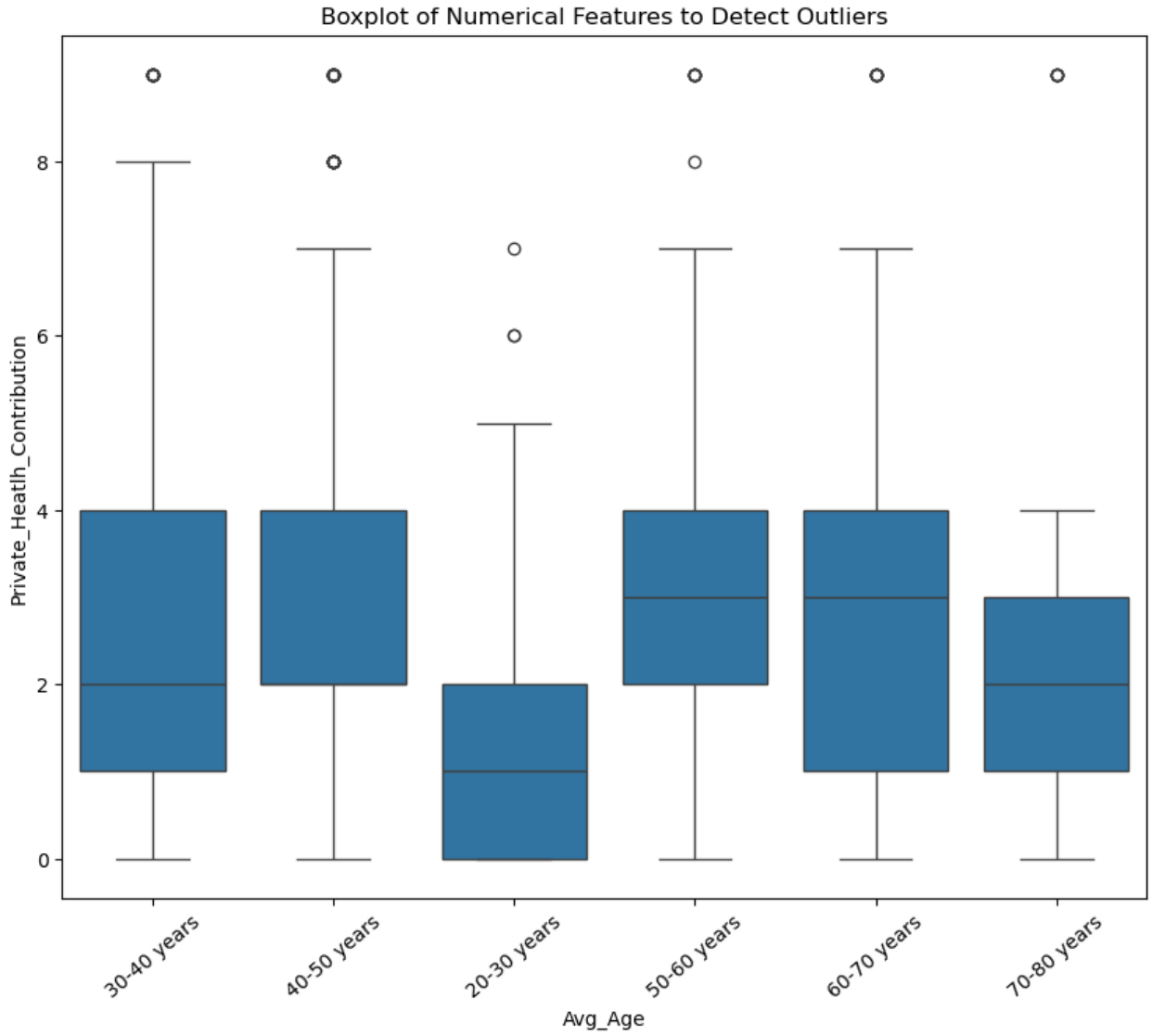
In [721…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Owns_One_Car'], y = Insurance_df['Car_Polic
plt.xlabel('Owns_On_Car')
plt.ylabel('Car_Policy_Contribution')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



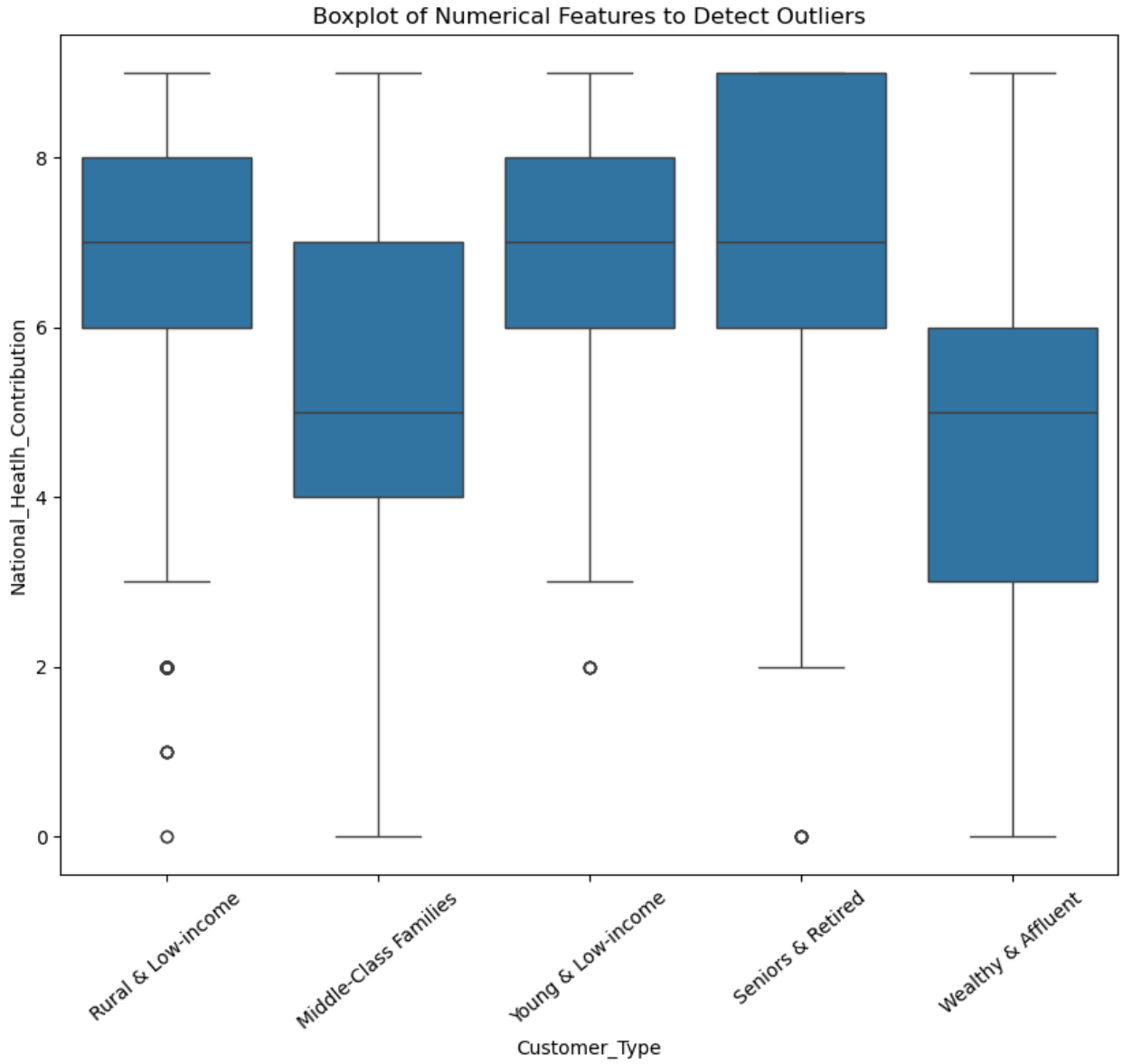
In [672…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Avg_Age'], y = Insurance_df['Private_Health
plt.xlabel('Avg_Age')
plt.ylabel('Private_Heatlh_Contribution')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



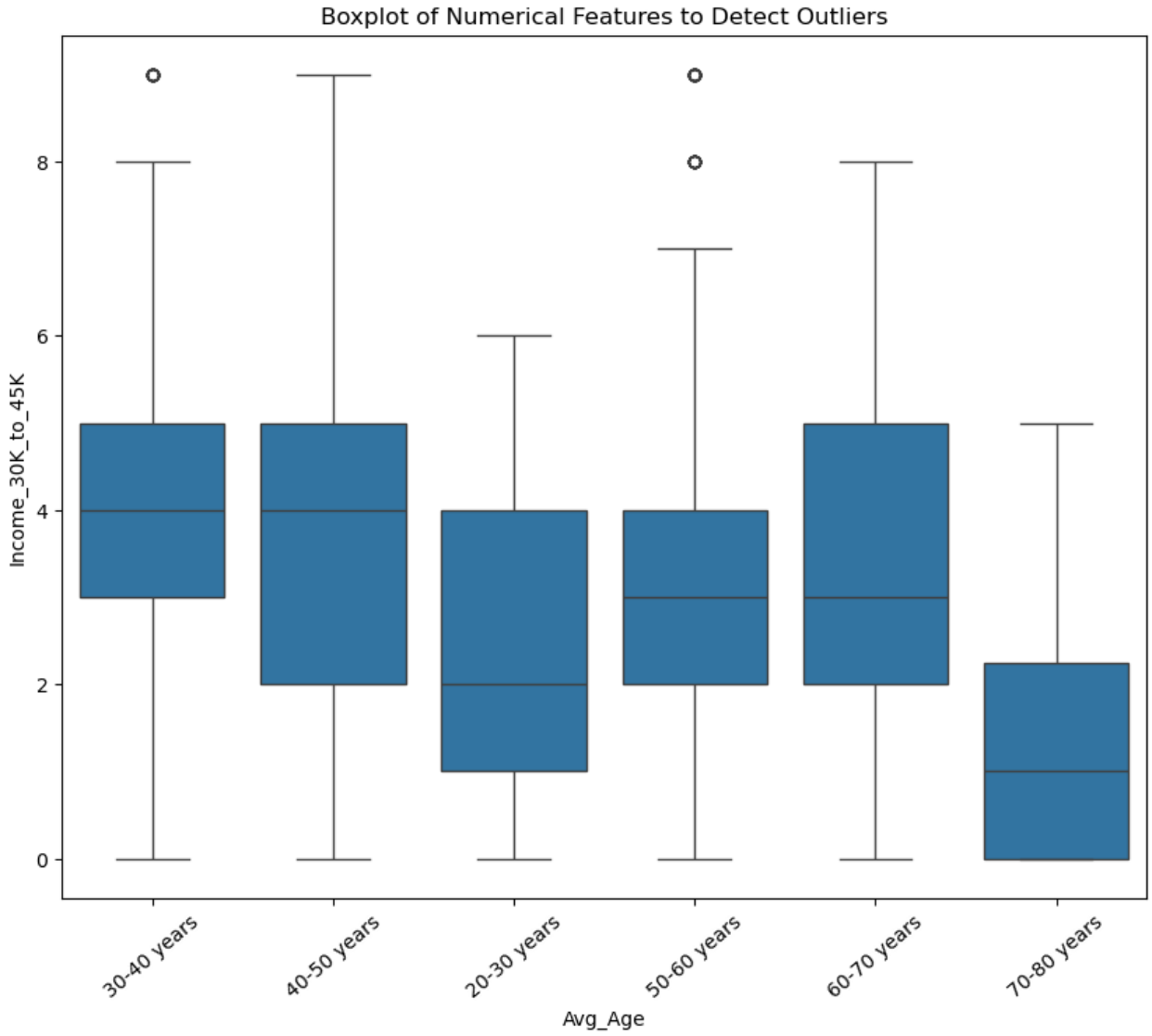
In [704…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Customer_Type'], y = Insurance_df['National
plt.xlabel('Customer_Type')
plt.ylabel('National_Heatlh_Contribution')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



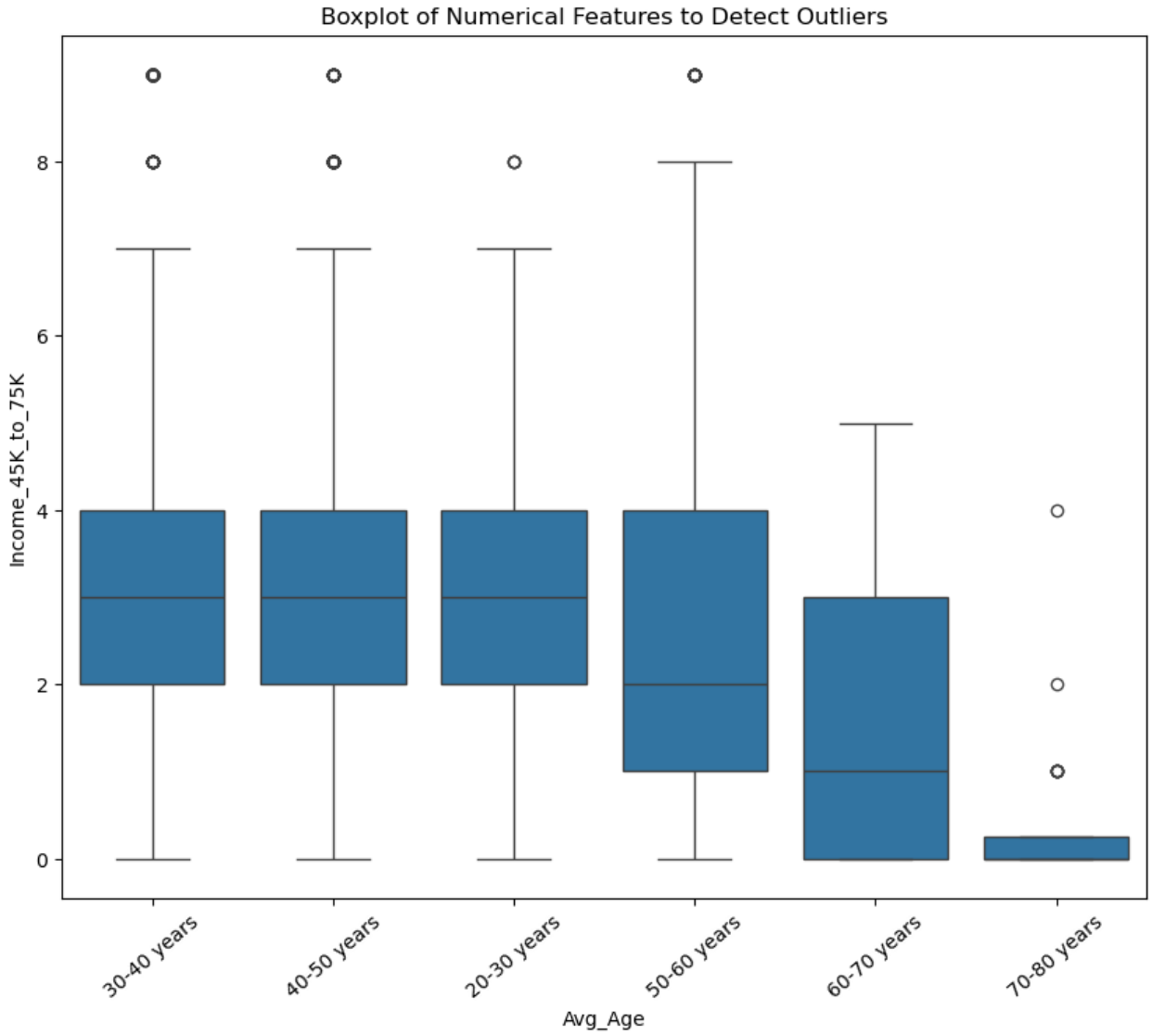
In [ ]:

In [727…
```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Avg_Age'], y = Insurance_df['Income_30K_to_
plt.xlabel('Avg_Age')
plt.ylabel('Income_30K_to_45K')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



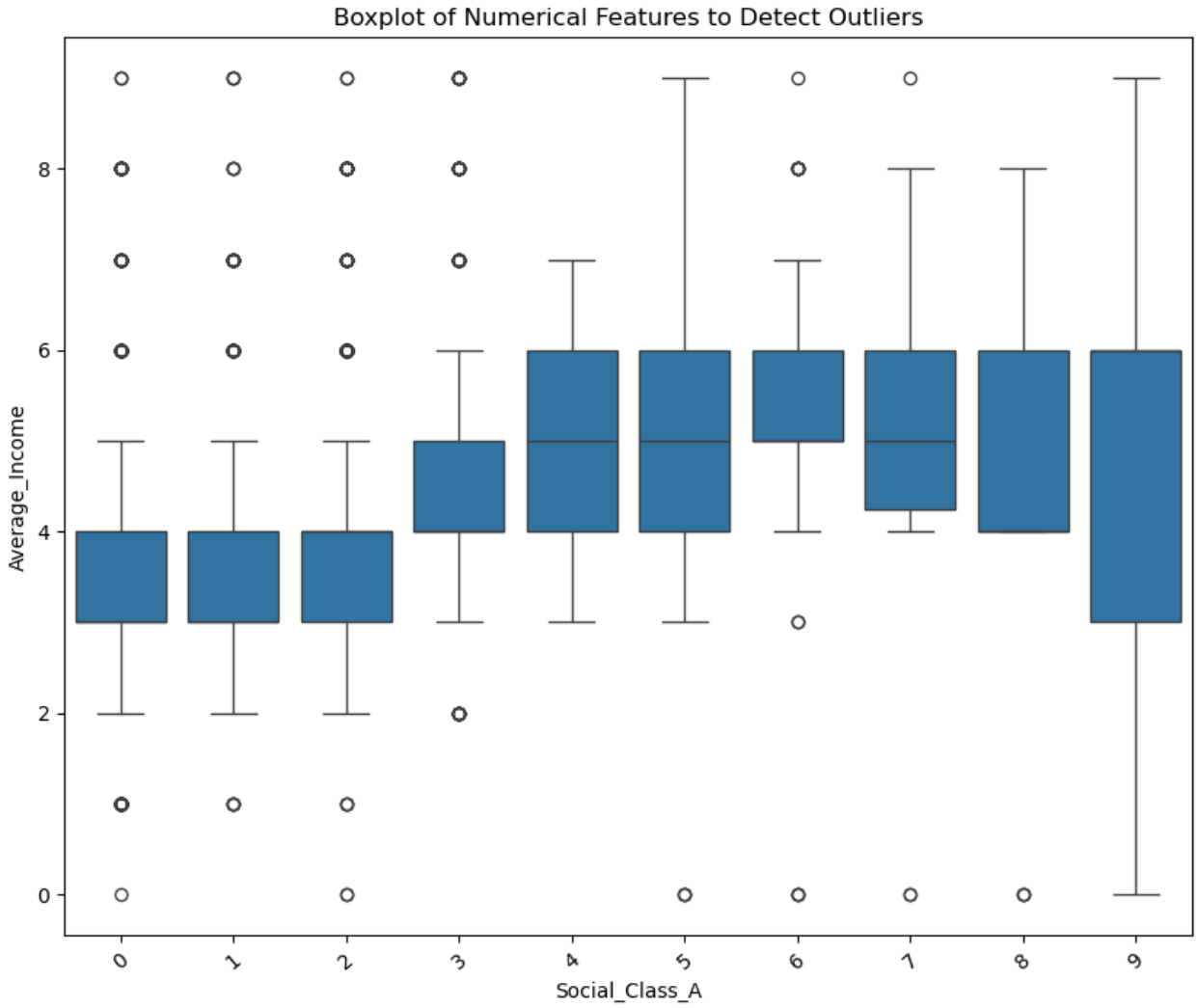
In [725…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Avg_Age'], y = Insurance_df['Income_45K_to_
plt.xlabel('Avg_Age')
plt.ylabel('Income_45K_to_75K')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



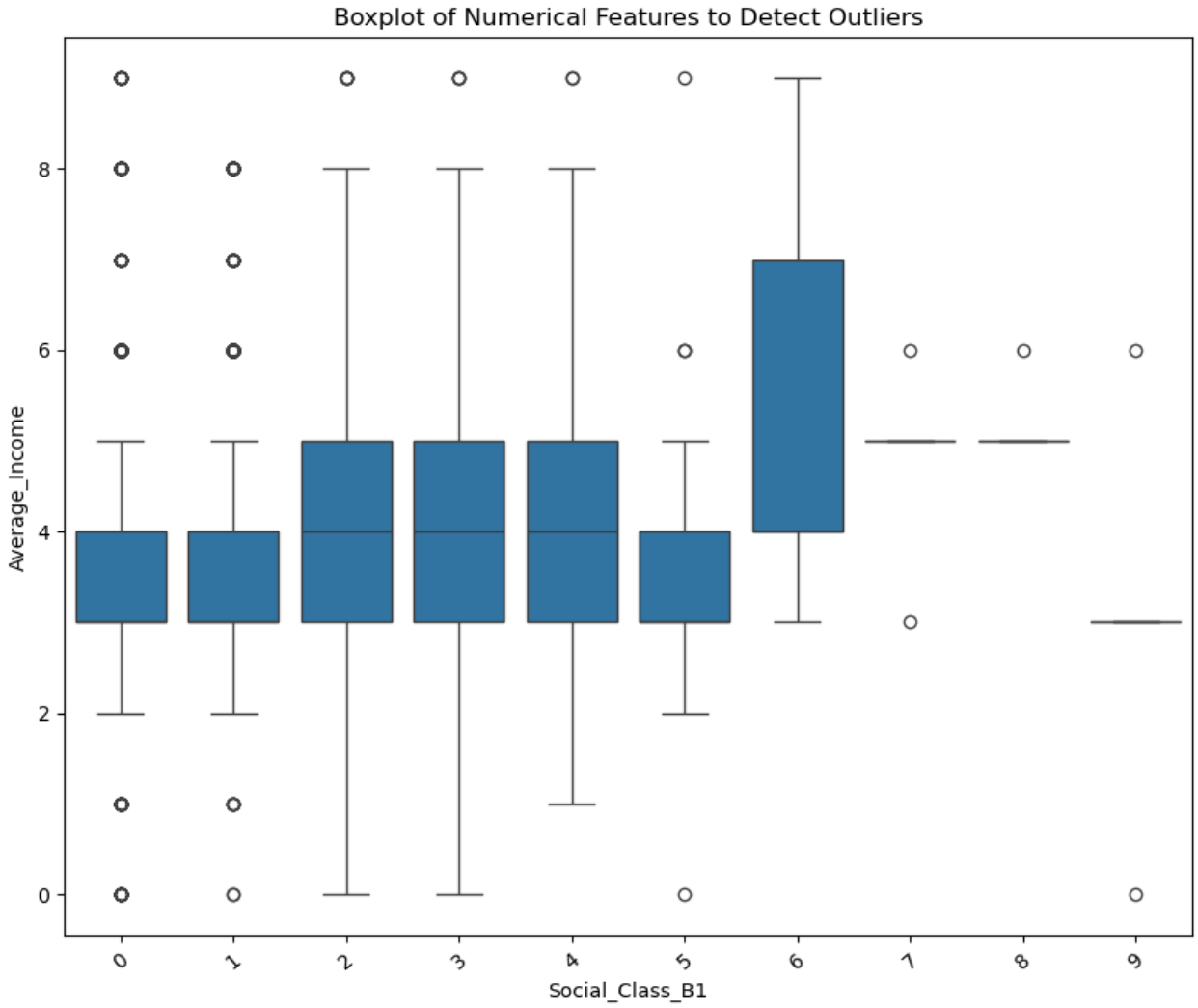
In [737…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Social_Class_A'], y = Insurance_df['Average
plt.xlabel('Social_Class_A')
plt.ylabel('Average_Income')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



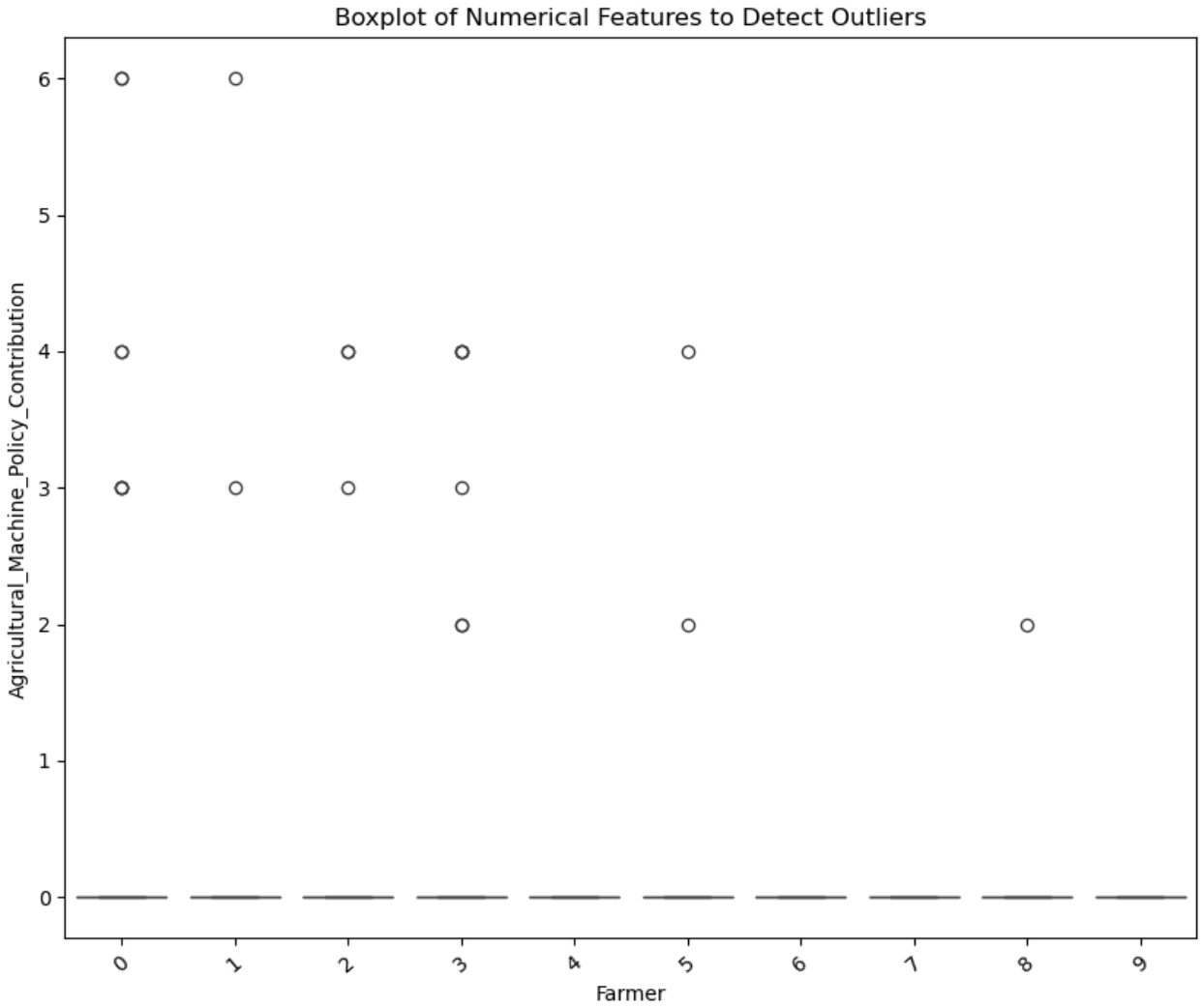
In [735…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Social_Class_B1'], y = Insurance_df['Averag
plt.xlabel('Social_Class_B1')
plt.ylabel('Average_Income')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



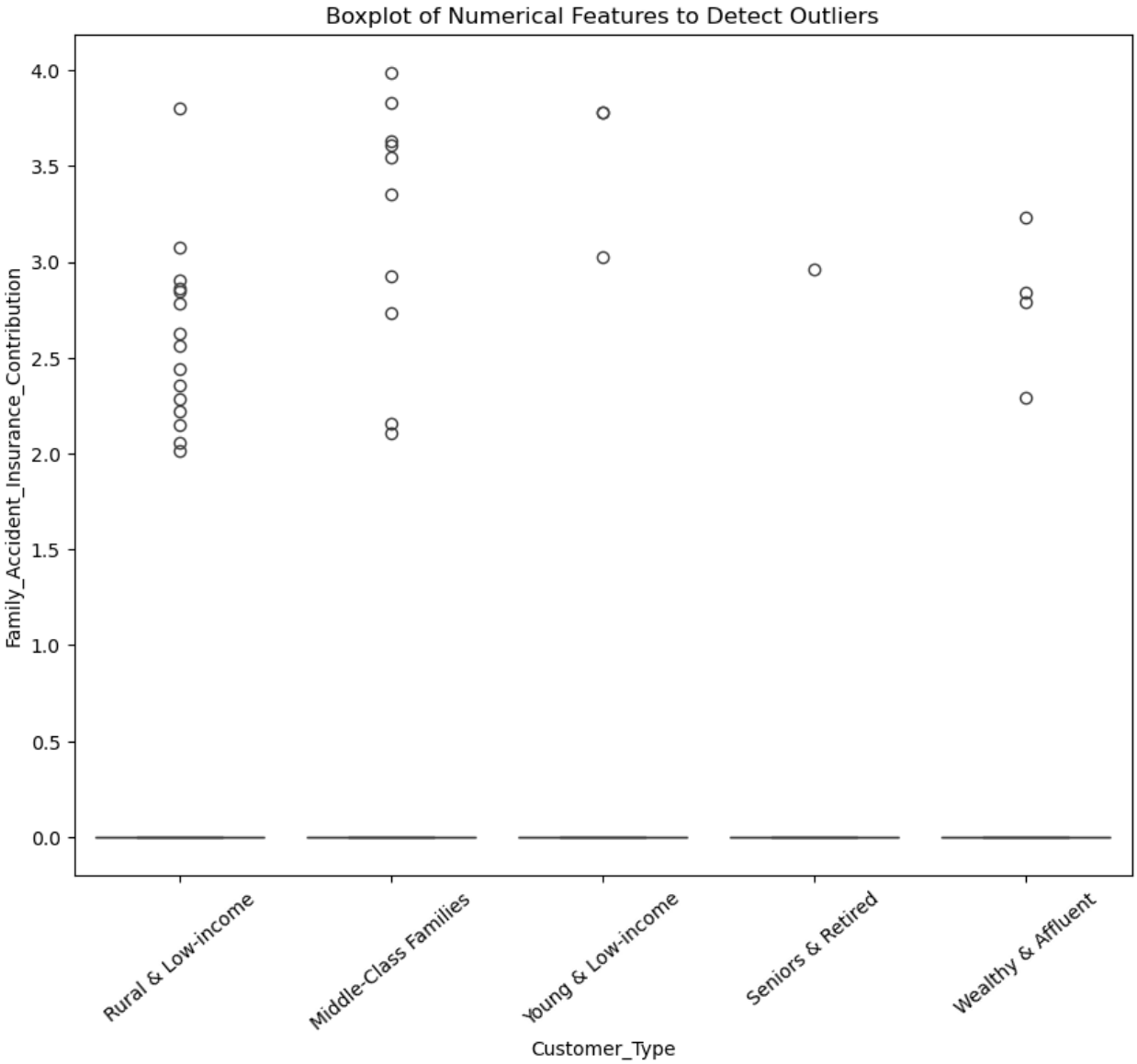
In [949…

```
plt.figure(figsize=(10,8))
sns.boxplot(x=Insurance_df['Farmer'],
                y=Insurance_df['Agricultural_Machine_Policy_Contribution'
plt.xlabel('Farmer')
plt.ylabel('Agricultural_Machine_Policy_Contribution')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



In [749…

```
plt.figure(figsize=(10,8))
sns.boxplot(x = Insurance_df['Customer_Type'], y = Insurance_df['Family_A
plt.xlabel('Customer_Type')
plt.ylabel('Family_Accident_Insurance_Contribution')
plt.xticks(rotation = 40)
plt.title("Boxplot of Numerical Features to Detect Outliers")
plt.show()
```



In [164…
```
#To Verify whether the Numerical Featuers in the Dataset are skewed towar

check_columns_for_zeros_in_object = Insurance_df.iloc[:, 40:42]
check_columns_for_zeros_in_numeric = Insurance_df.iloc[:, 42:83]

check_columns_for_zeros_in_object.eq('0').sum()
```

Out[164…
```
Business_Third_Party_Insurance_Contribution        5442
Agricultural_Third_Party_Insurance_Contribution    5403
dtype: int64
```

In [165…
```
check_columns_for_zeros_in_numeric.eq(0).sum() #number of zeros within ea
```

Out[165…

```
Car_Policy_Contribution                       2690
Delivery_Van_Policy_Contribution              5454
Motorcycle_Scooter_Policy_Contribution        5312
Lorry_Policy_Contribution                     5512
Trailer_Policy_Contribution                   5442
Tractor_Policy_Contribution                   5382
Agricultural_Machine_Policy_Contribution      5500
Moped_Policy_Contribution                     5096
Life_Insurance_Contribution                   5182
Private_Accident_Insurance_Contribution       5440
Family_Accident_Insurance_Contribution        5436
Disability_Insurance_Contribution             5427
Fire_Insurance_Contribution                   2497
Surfboard_Insurance_Contribution              5466
Boat_Insurance_Contribution                   5440
Bicycle_Insurance_Contribution                5326
Property_Insurance_Contribution               5425
Social_Security_Insurance_Contribution        5392
Number_Private_Third_Party_Insurance          3265
Number_Business_Third_Party_Insurance         5391
Number_Agricultural_Third_Party_Insurance     5353
Number_Car_Policies                           2690
Number_Delivery_Van_Policies                  5473
Number_Motorcycle_Scooter_Policies            5312
Number_Lorry_Policies                         5512
Number_Trailer_Policies                       5429
Number_Tractor_Policies                       5350
Number_Agricultural_Machine_Policies          5468
Number_Moped_Policies                         5117
Number_Life_Insurances                        5211
Number_Private_Accident_Insurances            5460
Number_Family_Accident_Insurances             5456
Number_Disability_Insurances                  5498
Number_Fire_Insurances                        2530
Number_Surfboard_Insurances                   5518
Number_Boat_Insurances                        5491
Number_Bicycle_Insurances                     5377
Number_Property_Insurances                    5428
Number_Social_Security_Insurances             5395
Number_Mobile_Home_Policies                   5119
Mobile_Home_Policies                             0
dtype: int64
```

In [106…

```
nan_count = Insurance_df[Categorical_Columns].isnull().sum()  # Count NAN

print('Columns with Null Values:\n',nan_count)
```

```
Columns with Null Values:
 Customer_Type                                        0
Avg_Age                                              14
Household_Profile                                     0
Private_Third_Party_Insurance_Contribution            9
Business_Third_Party_Insurance_Contribution           0
Agricultural_Third_Party_Insurance_Contribution       0
Mobile_Home_Policies                                 65
dtype: int64
```

In [187…
```
# Check for columns in Numerical_Columns with no null values
null_numerical_columns = Insurance_df[Numerical_Columns].isnull().sum()
zero_numerical_columns = Insurance_df[Numerical_Columns].eq(0).sum()
# Display the columns with no null values
print("Numerical columns with zeros values:")
print(zero_numerical_columns)
print("\n")
print("Numerical columns with Null Values:")
print(null_numerical_columns)
```

```
Numerical columns with zeros values:
Number_of_Houses                        0
Avg_Household_Size                      0
Married                                61
Living_Together                      2320
Other_Relation                       1105
                                     ...
Number_Boat_Insurances               5491
Number_Bicycle_Insurances            5377
Number_Property_Insurances           5428
Number_Social_Security_Insurances    5395
Number_Mobile_Home_Policies          5119
Length: 76, dtype: int64


Numerical columns with Null Values:
Number_of_Houses                       0
Avg_Household_Size                     0
Married                                0
Living_Together                        0
Other_Relation                         0
                                      ..
Number_Boat_Insurances                 0
Number_Bicycle_Insurances              0
Number_Property_Insurances            49
Number_Social_Security_Insurances     49
Number_Mobile_Home_Policies           74
Length: 76, dtype: int64
```

In [187…
```
Insurance_df['Business_Third_Party_Insurance_Contribution'].unique()
```

Out[187…
```
array(['0', 'Jan-49', '100-199', '200-499', '50-99', '1000-4999',
       '500-999'], dtype=object)
```

Data Dictionary

- Displays the summary of the features present in the dataset
- Statistical Summary
- The dictionary helps the developer to get an idea of the information of the columns

In [ ]:
```
data_dict = []
for col in Insurance_df.columns:
   data_dict.append({
      "Column Name": col,
      "Data Type": Insurance_df[col].dtype,
      "Field Size": Insurance_df[col].size,
      "Description": "",
      "Example": Insurance_df[col].apply(
          lambda x: Insurance_df[col].dropna().astype(str).head(3).tolist
      )
    })
```

In [43]:
```
descriptions = {
    "Customer_Type": "Group the customer belongs to based on their lifest
    "Number_of_Houses": "How many houses the customer owns.",
    "Avg_Household_Size": "Average number of people in the household.",
    "Avg_Age": "Average age of people in the household.",
    "Household_Profile": "Type of household the customer lives in.",
    "Married": "Is the customer married?",
    "Living_Together": "Is the customer living with a partner?",
    "Other_Relation": "Lives with relatives or others?",
    "Singles": "Lives alone or is single?",
    "Household_Without_Children": "No children in the household?",
    "Household_With_Children": "Children are part of the household?",
    "High_Education_Level": "Customer has a high education level?",
    "Medium_Education_Level": "Customer has a medium education level?",
    "Low_Education_Level": "Customer has a low education level?",
    "High_Status": "Customer has a high social or financial status?",
    "Entrepreneur": "Customer owns or runs a business?",
    "Farmer": "Customer works as a farmer?",
    "Middle_Management": "Customer works in middle management?",
    "Skilled_Labourers": "Customer has a skilled job?",
    "Unskilled_Labourers": "Customer has an unskilled job?",
    "Social_Class_A": "Belongs to upper social class (A)?",
    "Social_Class_B1": "Belongs to social class B1?",
    "Social_Class_B2": "Belongs to social class B2?",
    "Social_Class_C": "Belongs to social class C?",
    "Social_Class_D": "Belongs to social class D?",
    "Rented_House": "Lives in a rented house?",
    "Home_Owner": "Owns their home?",
    "Owns_One_Car": "Owns one car?",
    "Owns_Two_Cars": "Owns two cars?",
    "Owns_No_Car": "Doesn't own a car?",
```

```
    "National_Health_Insurance": "Has government health insurance?",
    "Private_Health_Insurance": "Has private health insurance?",
    "Income_Less_Than_30K": "Income is under 30,000?",
    "Income_30K_to_45K": "Income is between 30,000 and 45,000?",
    "Income_45K_to_75K": "Income is between 45,000 and 75,000?",
    "Income_75K_to_122K": "Income is between 75,000 and 122,000?",
    "Income_Above_123K": "Income is more than 123,000?",
    "Average_Income": "Average yearly income of the customer.",
    "Purchasing_Power_Class": "How strong their buying power is.",
    "Private_Third_Party_Insurance_Contribution": "Money paid into privat
    "Business_Third_Party_Insurance_Contribution": "Money paid into busin
    "Agricultural_Third_Party_Insurance_Contribution": "Money paid into f
    "Car_Policy_Contribution": "Money paid into car insurance.",
    "Delivery_Van_Policy_Contribution": "Money paid for delivery van insu
    "Motorcycle_Scooter_Policy_Contribution": "Money paid for motorbike o
    "Lorry_Policy_Contribution": "Money paid for truck insurance.",
    "Trailer_Policy_Contribution": "Money paid for trailer insurance.",
    "Tractor_Policy_Contribution": "Money paid for tractor insurance.",
    "Agricultural_Machine_Policy_Contribution": "Money paid for farm mach
    "Moped_Policy_Contribution": "Money paid for moped insurance.",
    "Life_Insurance_Contribution": "Money paid into life insurance.",
    "Private_Accident_Insurance_Contribution": "Money paid into personal
    "Family_Accident_Insurance_Contribution": "Money paid into family acc
    "Disability_Insurance_Contribution": "Money paid into disability insu
    "Fire_Insurance_Contribution": "Money paid into fire insurance.",
    "Surfboard_Insurance_Contribution": "Money paid into surfboard insura
    "Boat_Insurance_Contribution": "Money paid into boat insurance.",
    "Bicycle_Insurance_Contribution": "Money paid into bicycle insurance.
    "Property_Insurance_Contribution": "Money paid into home/property ins
    "Social_Security_Insurance_Contribution": "Money paid into social sec
    "Number_Private_Third_Party_Insurance": "Number of private third-part
    "Number_Business_Third_Party_Insurance": "Number of business third-pa
    "Number_Agricultural_Third_Party_Insurance": "Number of farm-related
    "Number_Car_Policies": "Number of car insurance policies.",
    "Number_Delivery_Van_Policies": "Number of delivery van insurance pol
    "Number_Motorcycle_Scooter_Policies": "Number of motorcycle or scoote
    "Number_Lorry_Policies": "Number of lorry (truck) insurance policies.
    "Number_Trailer_Policies": "Number of trailer insurance policies.",
    "Number_Tractor_Policies": "Number of tractor insurance policies.",
    "Number_Agricultural_Machine_Policies": "Number of agricultural machi
    "Number_Moped_Policies": "Number of moped insurance policies.",
    "Number_Life_Insurances": "Number of life insurance policies.",
    "Number_Private_Accident_Insurances": "Number of personal accident in
    "Number_Family_Accident_Insurances": "Number of family accident insur
    "Number_Disability_Insurances": "Number of disability insurance polic
    "Number_Fire_Insurances": "Number of fire insurance policies.",
    "Number_Surfboard_Insurances": "Number of surfboard insurance policie
    "Number_Boat_Insurances": "Number of boat insurance policies.",
    "Number_Bicycle_Insurances": "Number of bicycle insurance policies.",
    "Number_Property_Insurances": "Number of property insurance policies.
    "Number_Social_Security_Insurances": "Number of social security insur
    "Number_Mobile_Home_Policies": "Number of mobile home policies.",
    "Mobile_Home_Policies": "Money or count related to mobile home insura
```

```
}
```

In [45]:
```
data_dict["Description"] = data_dict["Column Name"].map(descriptions).fil

data_dict = pd.DataFrame(data_dict)
data_dict
```

Out[45]:

	Column Name	Data Type	Field Size	Description	Example
0	Customer_Type	object	5521	Group the customer belongs to based on their l...	0 [Rural & Low-income, Rural & Low-incom...
1	Number_of_Houses	int64	5521	How many houses the customer owns.	0 [1, 1, 1] 1 [1, 1, 1] 2 [1...
2	Avg_Household_Size	int64	5521	Average number of people in the household.	0 [3, 2, 2] 1 [3, 2, 2] 2 [3...
3	Avg_Age	object	5521	Average age of people in the household.	0 [30-40 years, 30-40 years, 30-40 years...
4	Household_Profile	object	5521	Type of household the customer lives in.	0 [Family with Grown-Ups, Family with Gr...
...	...	...	...	...	...
78	Number_Bicycle_Insurances	int64	5521	Number of bicycle insurance policies.	0 [0, 0, 0] 1 [0, 0, 0] 2 [0...
79	Number_Property_Insurances	float64	5521	Number of property insurance policies.	0 [0.0, 0.0, 0.0] 1 [0.0, 0.0, 0.0...
80	Number_Social_Security_Insurances	float64	5521	Number of social security insurance policies.	0 [0.0, 0.0, 0.0] 1 [0.0, 0.0, 0.0...
81	Number_Mobile_Home_Policies	float64	5521	Number of mobile home	0 [0.0, 0.0, 0.0] 1 [0.0,

					policies.	0.0, 0.0...
82	Mobile_Home_Policies	object	5521	Money or count related to mobile home insurance.	0 [No Policy, No Policy, No Policy] 1 ...	

83 rows × 5 columns

In [ ]:
```
bb
```

In [ ]:
```

```

The number of zeros in the dataset are more than 50 percent. It just shows that we will have to analyse the dataset based on the

Preprocessing and Cleaning

- Once getting the dataset, check for the missing values within the features.
- Remove the features that does not add much into the analysis

Replace the NAN values in the features

- Some of the features from their visualization indicates that some of the features are skewed towards 0. So for more precise classification, we will have to reduce it's dimensionality.

In [114…
```
#To Fill in all the Nan values among the Categorical Features with Unknow
Insurance_df[Categorical_Columns] = Insurance_df[Categorical_Columns].app
```

In [116…
```
Categorical_Columns.remove('Customer_Type')
Insurance_df[Categorical_Columns].isnull().sum()
```

Out[116…
```
Avg_Age                                          0
Household_Profile                                0
Private_Third_Party_Insurance_Contribution       0
Business_Third_Party_Insurance_Contribution      0
Agricultural_Third_Party_Insurance_Contribution  0
Mobile_Home_Policies                             0
dtype: int64
```

In [21]:
```
# There is a unique value within the feature that needs to be replaced ap

valid_categories = ['0', '100-199', '200-499', '50-99', '1000-4999', '500

Insurance_df['Business_Third_Party_Insurance_Contribution'] = Insurance_d
```

In [23]:
```
Insurance_df['Business_Third_Party_Insurance_Contribution'].unique()
```

Out[23]:
```
array(['0', 'Other', '100-199', '200-499', '50-99', '1000-4999',
       '500-999'], dtype=object)
```

In [21]:
```
Insurance_df.columns
```

Out[21]:
```
Index(['Customer_Type', 'Number_of_Houses', 'Avg_Household_Size', 'Avg_A
ge',
       'Household_Profile', 'Married', 'Living_Together', 'Other_Relatio
n',
       'Singles', 'Household_Without_Children', 'Household_With_Childre
n',
       'High_Education_Level', 'Medium_Education_Level', 'Low_Education_
Level',
       'High_Status', 'Entrepreneur', 'Farmer', 'Middle_Management',
       'Skilled_Labourers', 'Unskilled_Labourers', 'Social_Class_A',
       'Social_Class_B1', 'Social_Class_B2', 'Social_Class_C',
       'Social_Class_D', 'Rented_House', 'Home_Owner', 'Owns_One_Car',
       'Owns_Two_Cars', 'Owns_No_Car', 'National_Health_Insurance',
       'Private_Health_Insurance', 'Income_Less_Than_30K', 'Income_30K_t
o_45K',
       'Income_45K_to_75K', 'Income_75K_to_122K', 'Income_Above_123K',
       'Average_Income', 'Purchasing_Power_Class',
       'Private_Third_Party_Insurance_Contribution',
       'Business_Third_Party_Insurance_Contribution',
       'Agricultural_Third_Party_Insurance_Contribution',
       'Car_Policy_Contribution', 'Delivery_Van_Policy_Contribution',
       'Motorcycle_Scooter_Policy_Contribution', 'Lorry_Policy_Contribut
ion',
       'Trailer_Policy_Contribution', 'Tractor_Policy_Contribution',
       'Agricultural_Machine_Policy_Contribution', 'Moped_Policy_Contrib
ution',
       'Life_Insurance_Contribution',
       'Private_Accident_Insurance_Contribution',
       'Family_Accident_Insurance_Contribution',
       'Disability_Insurance_Contribution', 'Fire_Insurance_Contributio
n',
       'Surfboard_Insurance_Contribution', 'Boat_Insurance_Contributio
n',
       'Bicycle_Insurance_Contribution', 'Property_Insurance_Contributio
n',
       'Social_Security_Insurance_Contribution',
       'Number_Private_Third_Party_Insurance',
       'Number_Business_Third_Party_Insurance',
       'Number_Agricultural_Third_Party_Insurance', 'Number_Car_Policie
s',
       'Number_Delivery_Van_Policies', 'Number_Motorcycle_Scooter_Polici
es',
       'Number_Lorry_Policies', 'Number_Trailer_Policies',
       'Number_Tractor_Policies', 'Number_Agricultural_Machine_Policie
s',
       'Number_Moped_Policies', 'Number_Life_Insurances',
```

```
       'Number_Private_Accident_Insurances',
       'Number_Family_Accident_Insurances', 'Number_Disability_Insurance
s',
       'Number_Fire_Insurances', 'Number_Surfboard_Insurances',
       'Number_Boat_Insurances', 'Number_Bicycle_Insurances',
       'Number_Property_Insurances', 'Number_Social_Security_Insurance
s',
       'Number_Mobile_Home_Policies', 'Mobile_Home_Policies'],
      dtype='object')
```

In [24]:
```
Insurance_df[Numerical_Columns].isnull().sum()
```

Out[24]:
```
Number_of_Houses                    0
Avg_Household_Size                  0
Married                             0
Living_Together                     0
Other_Relation                      0
                                   ..
Number_Boat_Insurances              0
Number_Bicycle_Insurances           0
Number_Property_Insurances         49
Number_Social_Security_Insurances  49
Number_Mobile_Home_Policies        74
Length: 76, dtype: int64
```

In [108…
```
Insurance_df[Numerical_Columns] = Insurance_df[Numerical_Columns].apply(l
```

In [110…
```
Insurance_df[Numerical_Columns].isnull().sum()
```

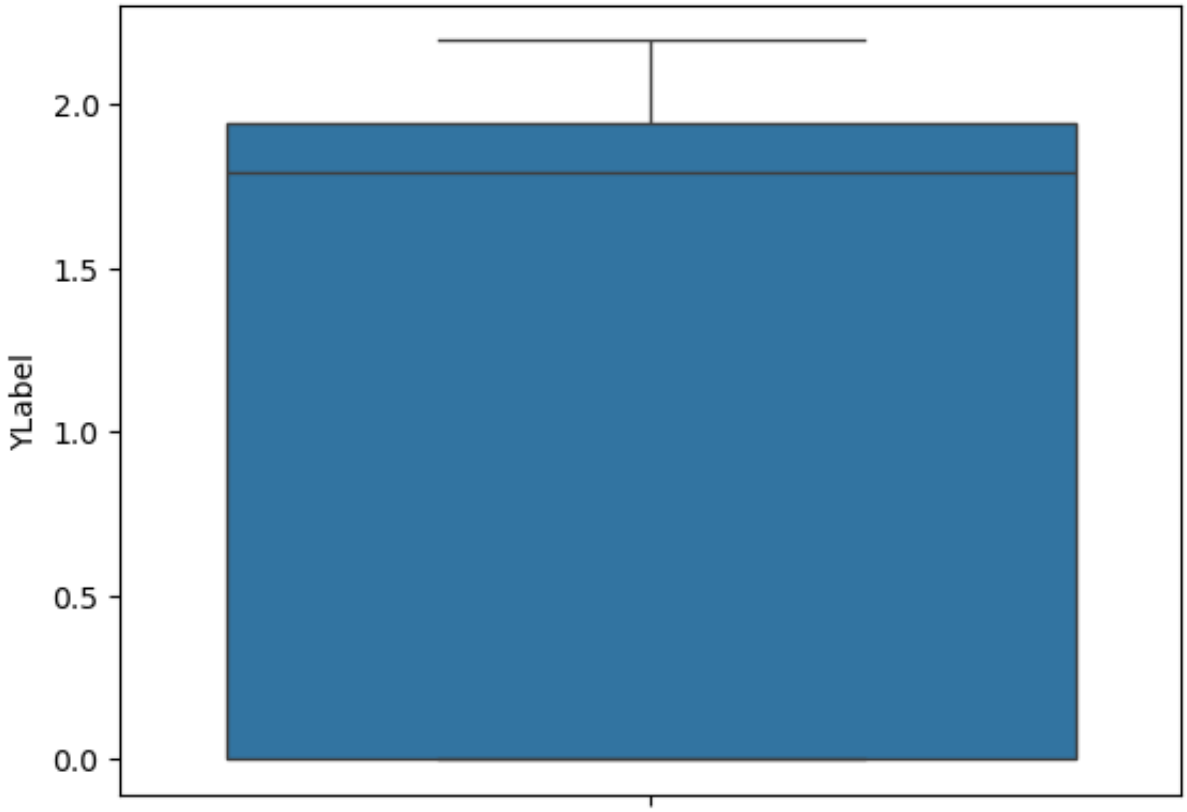
Out[110…
```
Number_of_Houses                    0
Avg_Household_Size                  0
Married                             0
Living_Together                     0
Other_Relation                      0
                                   ..
Number_Boat_Insurances              0
Number_Bicycle_Insurances           0
Number_Property_Insurances          0
Number_Social_Security_Insurances   0
Number_Mobile_Home_Policies         0
Length: 76, dtype: int64
```

Cleaning the Dataset Before Train_test_split

- To remove all the extreme outliers before actual analysis

In [176…
```
# Boxplot was used to detect the Outliers within the Dataset, as the valu
#Yet I used log1p ] log(1+x) for normalizing the values within

Insurance_df['Car_Policy_Contribution'] = np.log1p(Insurance_df['Car_Poli
sns.boxplot(y = Insurance_df['Car_Policy_Contribution'], data = Insurance
plt.ylabel('YLabel')
```

```
plt.show()
```



Encoding the Dataset

- With the help of feature Selection we'd able to find which ever columns/attributes would be helpful for classifying the information according to target_variable

In [111…
```
#!pip install category_encoders
```

In [118…
```
from sklearn.preprocessing import OrdinalEncoder

Insurance_Encoded_df = pd.DataFrame()
Insurance_df['Avg_Age'].unique()
```

Out[118…
```
array(['30-40 years', '40-50 years', '20-30 years', '50-60 years',
       '60-70 years', 'Unknown', '70-80 years'], dtype=object)
```

In [120…
```
Categorical_Columns = list(Insurance_df.select_dtypes(include = 'object')
Categorical_Columns.remove('Customer_Type')
```

In [122…
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for col in Categorical_Columns:
    Insurance_Encoded_df[col+'_Encoded'] = le.fit_transform(Insurance_df[
```

In [124…
```
Insurance_Encoded_df
```

Out[124…

	Avg_Age_Encoded	Household_Profile_Encoded	Private_Third_Party_Insuranc
0	1	5	
1	1	5	
2	1	5	
3	2	0	
4	1	6	
...	...	...	
5516	1	5	
5517	3	5	
5518	3	5	
5519	1	5	
5520	2	5	

5521 rows × 6 columns

In [126…
```
Numerical_Columns = list(Insurance_df.select_dtypes(include = 'number'))

Insurance_Encoded_df = pd.concat([Insurance_Encoded_df, Insurance_df[Nume
```

Train and Test Split

- After encoding the feature dataset. Split the train_test_split

In [128…
```
allInputs = Insurance_Encoded_df
target_variable = Insurance_df[['Customer_Type']]
```

In [92]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

X_train, X_test, y_train, y_test = train_test_split(Insurance_Encoded_df,
```

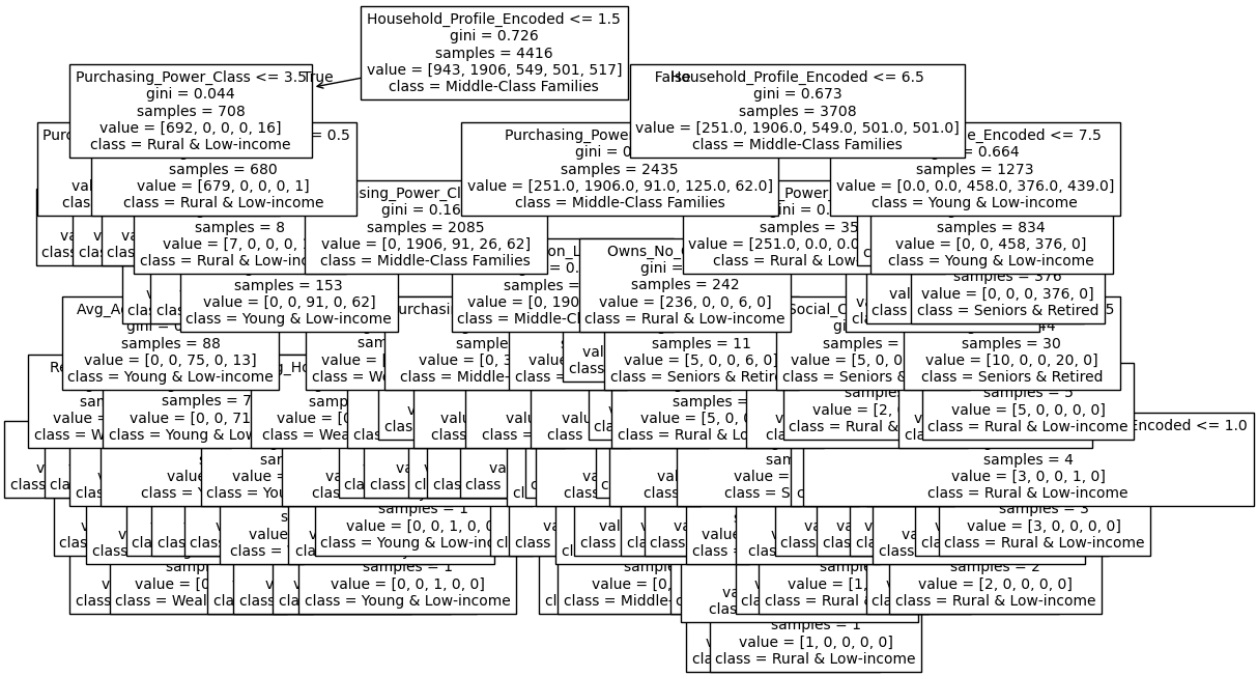
Decision Tree Classification

- The classification is done using Decision Tree
- The features as input is encoded using Label Encoder
- No Dimensionality Reduction

In [94]:
```
DT_classifier = DecisionTreeClassifier(random_state=42)

DT_classifier.fit(X_train, y_train)
y_pred = DT_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")


# Make predictions on both training and testing sets
y_pred_train_decision_tree = DT_classifier.predict(X_train)
y_pred_test_decision_tree = DT_classifier.predict(X_test)

# Evaluate the model's performance on both sets
accuracy_train_decision_tree = accuracy_score(y_train, y_pred_train_decis
accuracy_test_decision_tree = accuracy_score(y_test, y_pred_test_decision

# Assess overfitting or underfitting
if accuracy_train_decision_tree  > accuracy_test_decision_tree :
    print("The Decision Tree model might be overfitted.")
elif accuracy_train_decision_tree  < accuracy_test_decision_tree :
    print("The Decision Tree model might be underfitted.")
else:
    print("The Decision Tree model seems well-fitted.")
```

```
Accuracy: 98.82%
The Decision Tree model might be overfitted.
```

In [268…
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize = (12, 8))
#target_names = list(Insurance_df['Customer_Type'].unique())
plot_tree(DT_classifier, filled=False, feature_names = Insurance_Encoded_
plt.show()
```



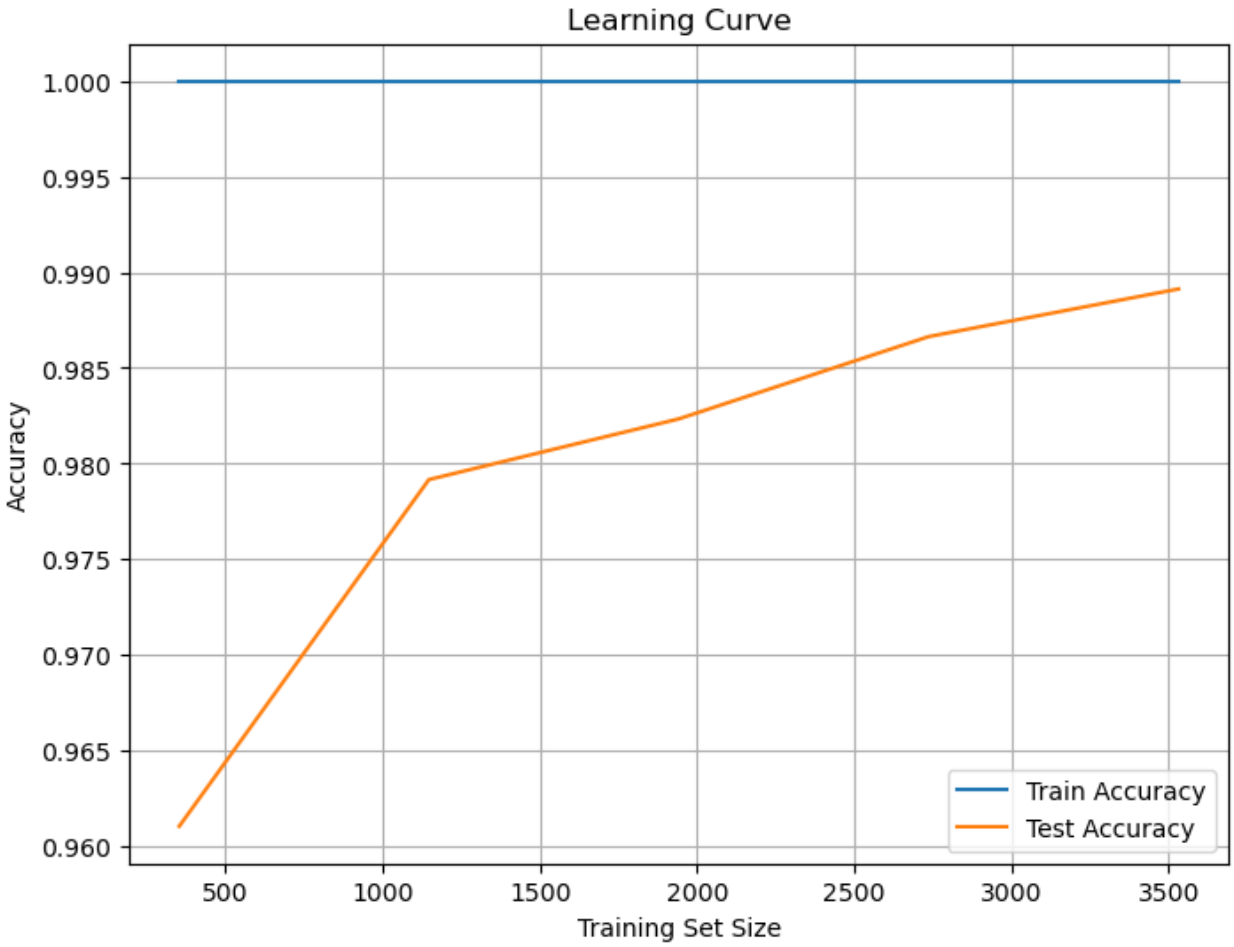
In [527…

```
print(classification_report(y_test, y_pred, target_names=target_names))
```

	precision	recall	f1-score	support
Rural & Low-income	0.99	0.98	0.99	260
Middle-Class Families	1.00	1.00	1.00	472
Young & Low-income	0.97	0.98	0.98	126
Seniors & Retired	0.97	0.99	0.98	126
Wealthy & Affluent	0.98	0.95	0.97	121
accuracy			0.99	1105
macro avg	0.98	0.98	0.98	1105
weighted avg	0.99	0.99	0.99	1105

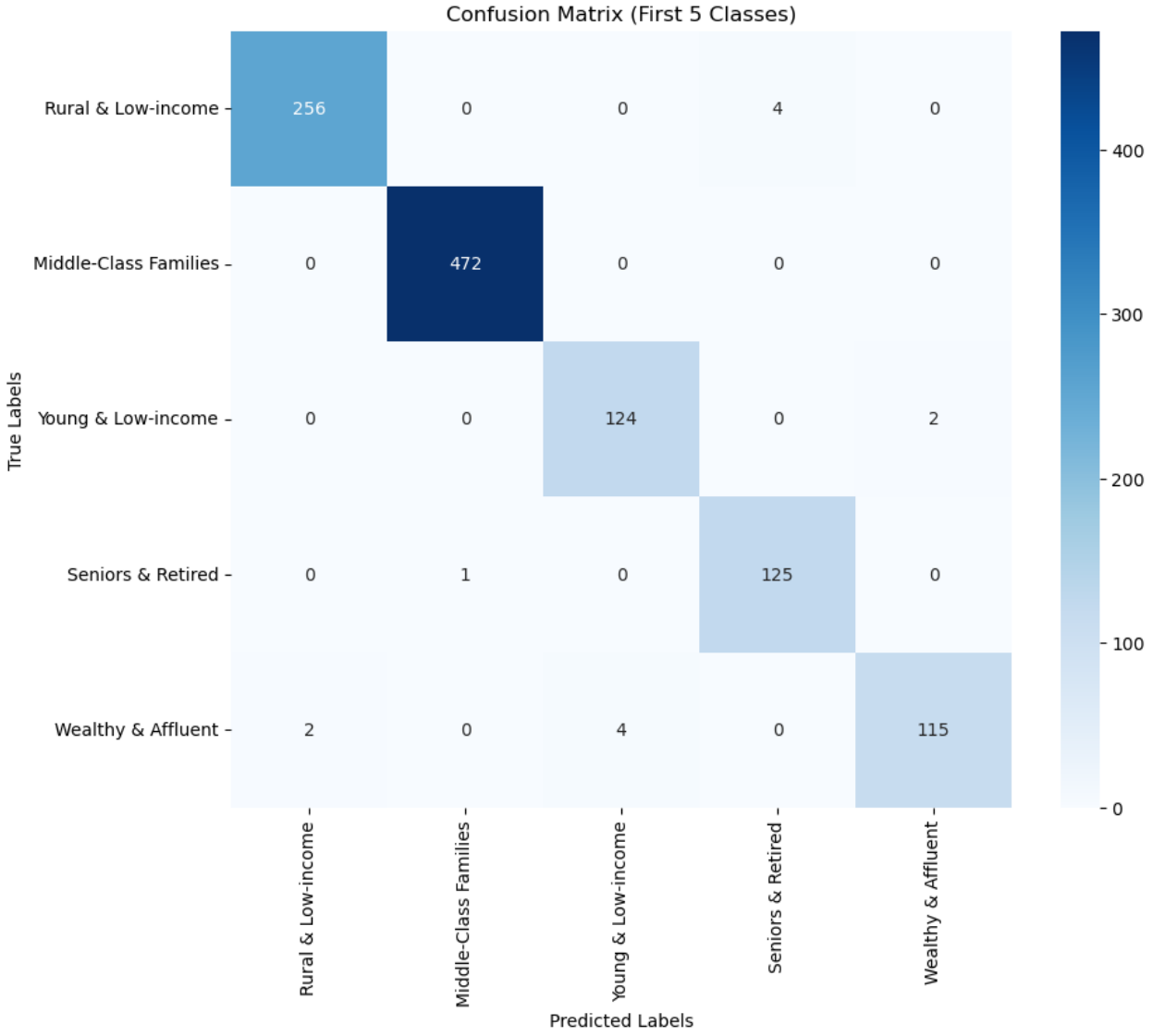
In [272…

```
from sklearn.model_selection import learning_curve

# Calculate the learning curve
train_sizes, train_scores, test_scores = learning_curve(DT_classifier, X_

# Plot the learning curve
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_scores.mean(axis=1), label='Train Accuracy')
plt.plot(train_sizes, test_scores.mean(axis=1), label='Test Accuracy')
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Learning Curve')
plt.legend()
plt.grid(True)
plt.show()
```



In [274…

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Get the predictions
y_pred = DT_classifier.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Select only the first 5 columns for visualization


# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cm_5, annot=True, fmt="d", cmap="Blues", xticklabels=target_n
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix (First 5 Classes)')
plt.show()
```



K - NN Classifier

- Without Dimensionality Reduction

In [529…

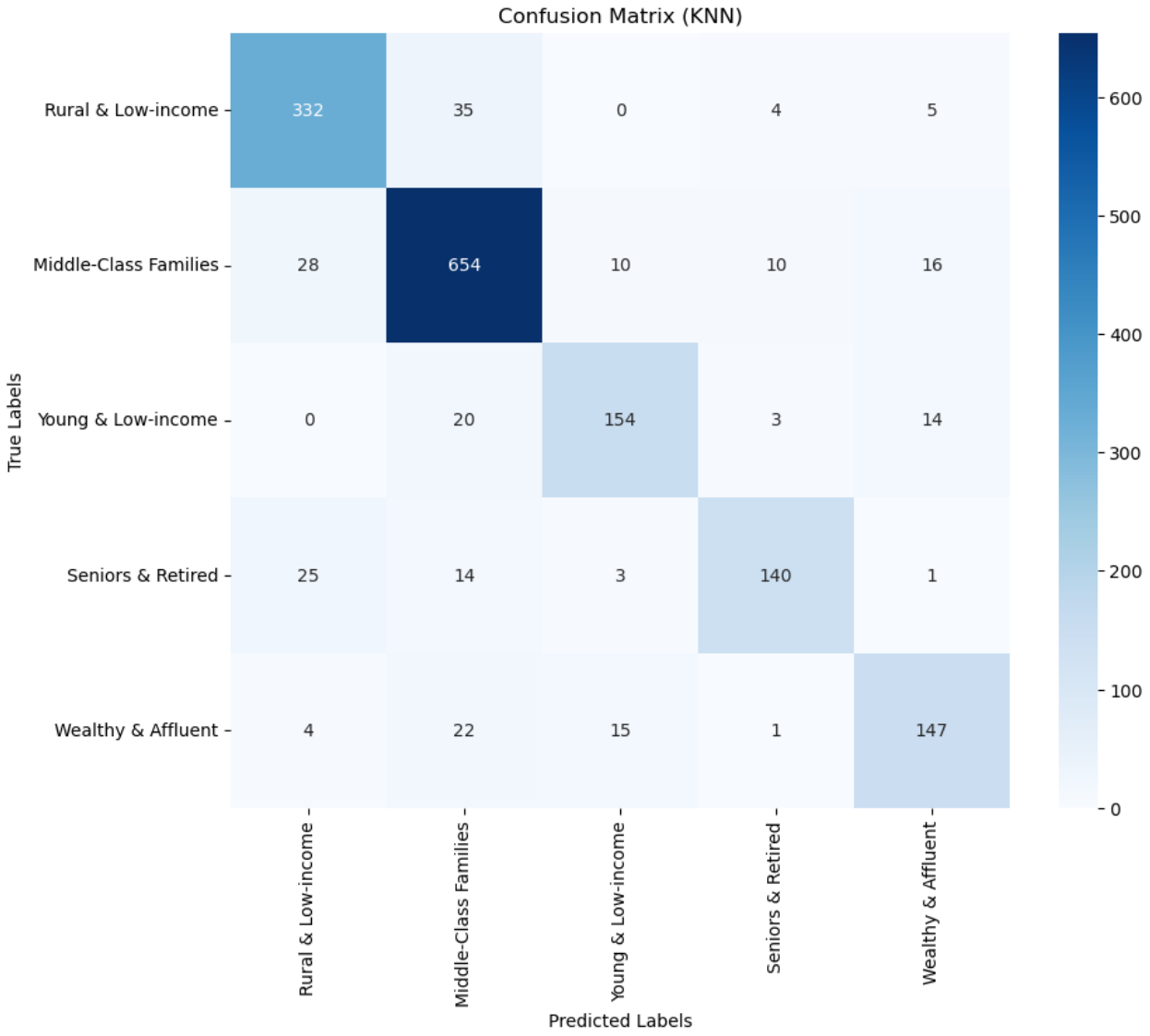
```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

y_1D = Insurance_df[['Customer_Type']].values.ravel()

X_train, X_test, y_train, y_test = train_test_split(Insurance_Encoded_df,


# Create KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can tune the number of n

# Train the model
knn.fit(X_train, y_train)

# Make predictions
```

```
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:\n", classification_report(y_test, y_pred)
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)

# Make predictions on both training and testing sets
y_pred_train_decision_tree = knn.predict(X_train)
y_pred_test_decision_tree = knn.predict(X_test)

# Evaluate the model's performance on both sets
accuracy_train_decision_tree = accuracy_score(y_train, y_pred_train_decis
accuracy_test_decision_tree = accuracy_score(y_test, y_pred_test_decision

# Assess overfitting or underfitting
if accuracy_train_decision_tree  > accuracy_test_decision_tree :
    print("The Decision Tree model might be overfitted.")
elif accuracy_train_decision_tree  < accuracy_test_decision_tree :
    print("The Decision Tree model might be underfitted.")
else:
    print("The Decision Tree model seems well-fitted.")
```

```
Accuracy: 86.12%

Classification Report:
                        precision    recall  f1-score   support

Middle-Class Families       0.85      0.88      0.87       376
   Rural & Low-income       0.88      0.91      0.89       718
    Seniors & Retired       0.85      0.81      0.83       191
   Wealthy & Affluent       0.89      0.77      0.82       183
   Young & Low-income       0.80      0.78      0.79       189

             accuracy                           0.86      1657
            macro avg       0.85      0.83      0.84      1657
         weighted avg       0.86      0.86      0.86      1657


Confusion Matrix:
The Decision Tree model might be overfitted.
```

In [278…

```
# Plot the confusion matrix using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=target_nam
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix (KNN)')
plt.show()
```



Apply Mutual Information And Principal Component Analysis

- As the dataset has 76 features the accuracy reduces
- So from now on we would only be considering only the important features

KNN with Pipeline (ENSEMBLE)

In [161…

```
# Mutual Info

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confus
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Mutual Info Feature Selection
from sklearn.feature_selection import mutual_info_classif

y = Insurance_df[['Customer_Type']].values.ravel()

mi = mutual_info_classif(Insurance_Encoded_df, y)
mi_series = pd.Series(mi, index=Insurance_Encoded_df.columns)
X_selected = Insurance_Encoded_df[mi_series[mi_series > 0].index]
```

In [136…
```
mi_series #Mutual Info Classif with scores
```

Out[136…
```
Avg_Age_Encoded                                          0.085267
Household_Profile_Encoded                                1.336244
Private_Third_Party_Insurance_Contribution_Encoded        0.007906
Business_Third_Party_Insurance_Contribution_Encoded       0.000000
Agricultural_Third_Party_Insurance_Contribution_Encoded   0.018037
                                                             ...
Number_Boat_Insurances                                   0.001788
Number_Bicycle_Insurances                                0.000000
Number_Property_Insurances                               0.000000
Number_Social_Security_Insurances                        0.000000
Number_Mobile_Home_Policies                              0.000000
Length: 82, dtype: float64
```

In [132…
```
X_selected
```

Out[132…

	Avg_Age_Encoded	Household_Profile_Encoded	Private_Third_Party_Insuranc
0	1	5	
1	1	5	
2	1	5	
3	2	0	
4	1	6	
...	...	...	
5516	1	5	
5517	3	5	
5518	3	5	
5519	1	5	
5520	2	5	

5521 rows × 63 columns

In [ ]:
```
```

In [138…
```
X_selected
```

Out[138…

	Avg_Age_Encoded	Household_Profile_Encoded	Private_Third_Party_Insuranc
0	1	5	
1	1	5	
2	1	5	
3	2	0	
4	1	6	
...	...	...	
5516	1	5	
5517	3	5	
5518	3	5	
5519	1	5	
5520	2	5	

5521 rows × 63 columns

In [ ]:
```
```

In [163…
```
# Split before pipeline

X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_s

# Create pipeline
pipeline = Pipeline([
    ('pca', PCA(n_components=0.95)),
    ('knn', KNeighborsClassifier(n_neighbors=5))
])

# Fit and predict
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

# Make predictions on both training and testing sets
y_pred_train_KNN = pipeline.predict(X_train)
y_pred_test_KNN = pipeline.predict(X_test)

# Evaluate the model's performance on both sets
accuracy_train_KNN = accuracy_score(y_train, y_pred_train_decision_tree)
accuracy_test_KNN = accuracy_score(y_test, y_pred_test_decision_tree)

# Assess overfitting or underfitting
if accuracy_train_KNN  > accuracy_test_KNN :
    print("The KNN model might be overfitted.")
```

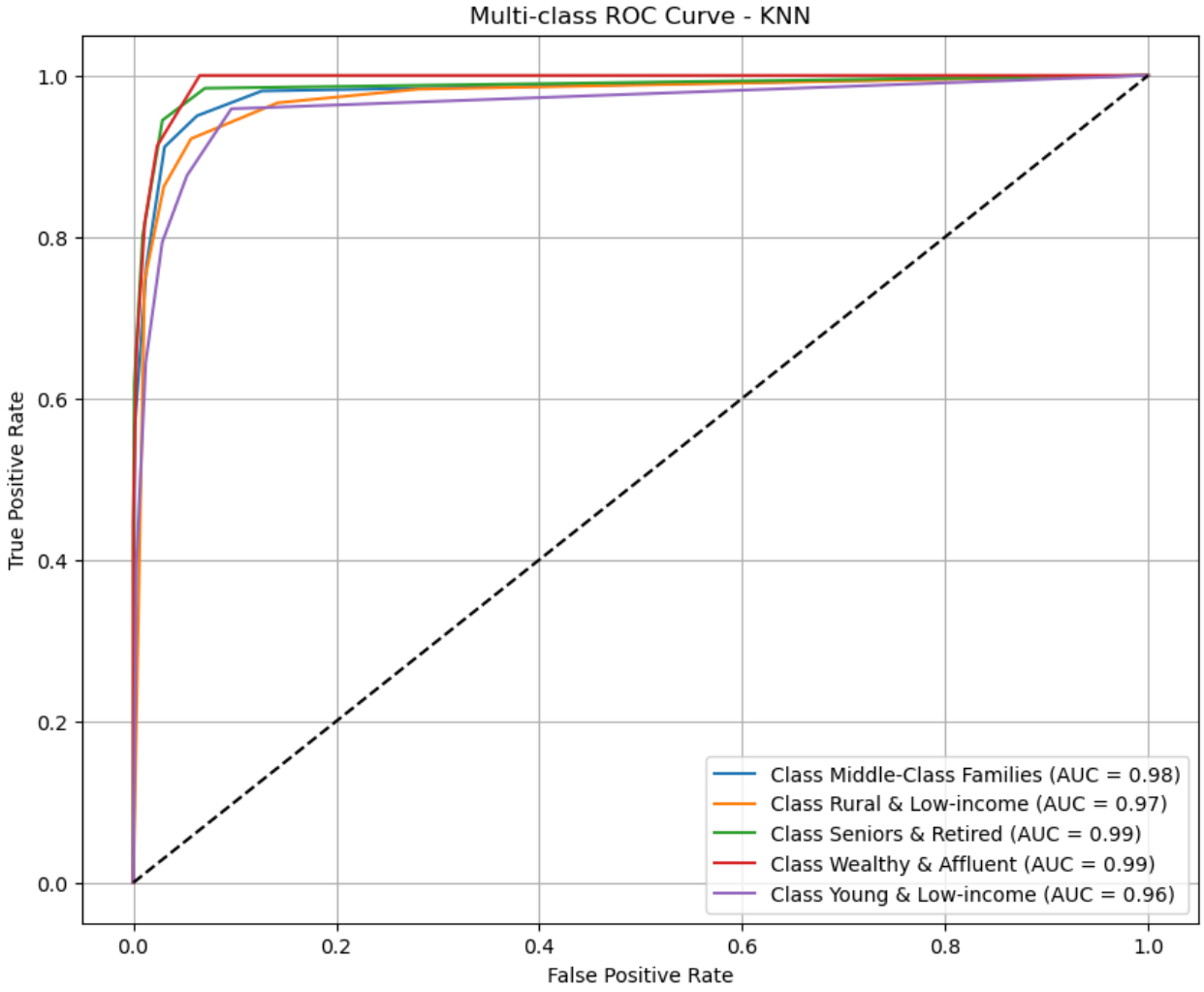
```
elif accuracy_train_KNN  < accuracy_test_KNN :
    print("The KNN model might be underfitted.")
else:
    print("The KNN model seems well-fitted.")
```

```
The KNN model might be overfitted.
```

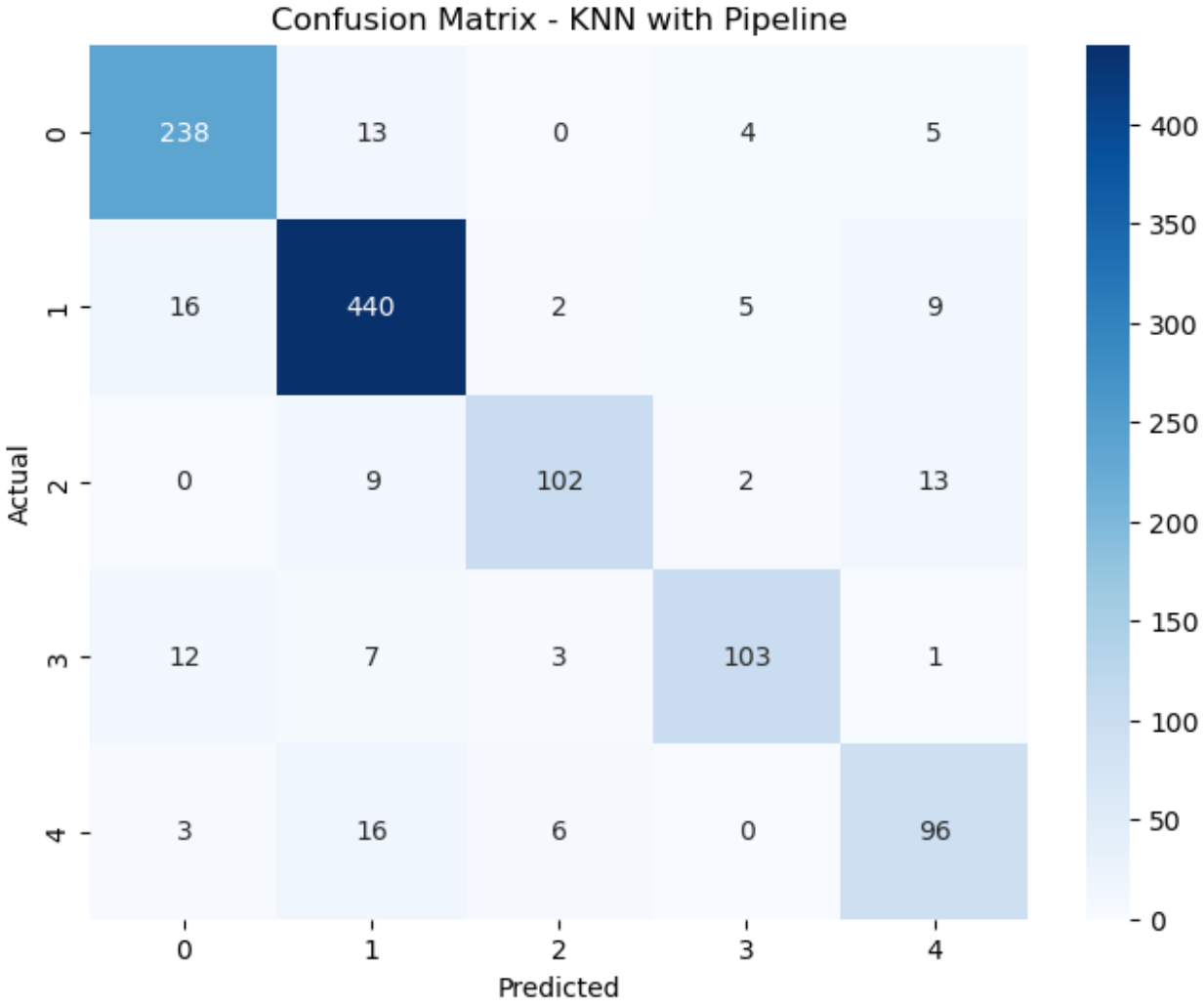
In [695…

```
# ROC Curve with KNN with Pipeline

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Binarize y for multi-class ROC
classes = np.unique(y)

y_test_bin = label_binarize(y_test, classes=classes)
y_train_bin = label_binarize(y_train, classes=classes)

# OneVsRest for multi-class support
knn_ovr = OneVsRestClassifier(pipeline)
knn_ovr.fit(X_train, y_train_bin)
y_score = knn_ovr.predict_proba(X_test)

# Calculate FPR, TPR for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot
plt.figure(figsize=(10, 8))
for i in range(len(classes)):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i

plt.plot([0, 1], [0, 1], 'k--', lw=1.5)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multi-class ROC Curve - KNN')
plt.legend()
plt.grid(True)
plt.show()

# Predict the final labels from probabilities
y_pred_bin = knn_ovr.predict(X_test)
y_pred_labels = np.argmax(y_pred_bin, axis=1)
y_test_labels = np.argmax(y_test_bin, axis=1)
```

```
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'KNN Classifier Accuracy: {accuracy * 100:.2f}%')

# Classification Report
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - KNN with Pipeline')
plt.show()
```



```
KNN Classifier Accuracy: 88.60%
Classification Report:
                        precision    recall  f1-score   support

Middle-Class Families       0.88      0.92      0.90       260
   Rural & Low-income       0.91      0.93      0.92       472
    Seniors & Retired       0.90      0.81      0.85       126
   Wealthy & Affluent       0.90      0.82      0.86       126
   Young & Low-income       0.77      0.79      0.78       121

             accuracy                           0.89      1105
            macro avg       0.87      0.85      0.86      1105
         weighted avg       0.89      0.89      0.89      1105
```



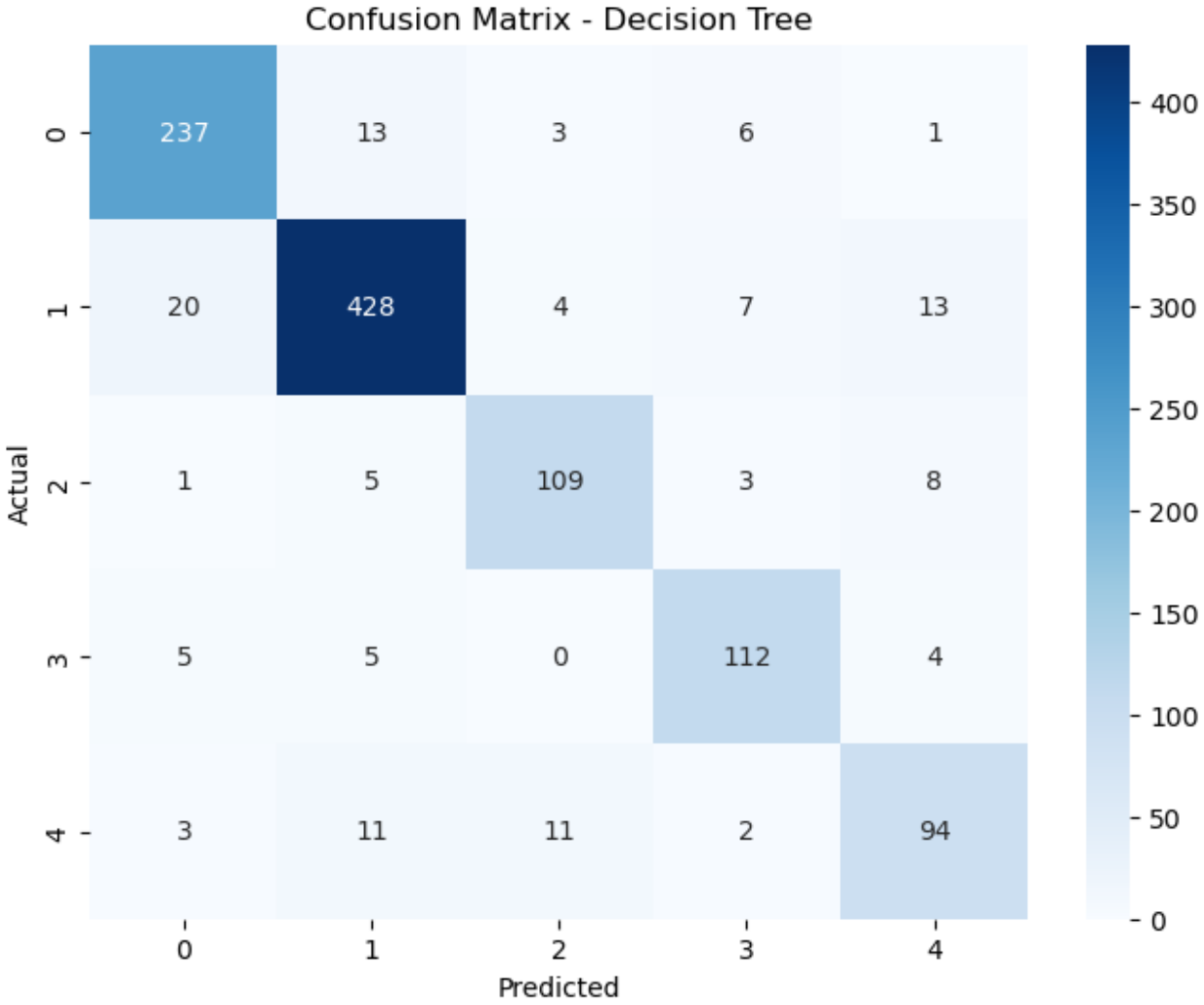
Decision Tree Classifier after Dimensional Reduction (ENSEMBLE)

In [703…

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confus
import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Step 1: Mutual Info Feature Selection (done outside the pipeline)
y = Insurance_df[['Customer_Type']].values.ravel()

mi = mutual_info_classif(Insurance_Encoded_df, y)
mi_series = pd.Series(mi, index=Insurance_Encoded_df.columns)
X_selected = Insurance_Encoded_df[mi_series[mi_series > 0].index]

# Step 2: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_s

# Step 3: Pipeline
pipeline = Pipeline([
   # ('scaler', StandardScaler()), 68.69%
    ('pca', PCA(n_components=0.95)),
    ('dt', DecisionTreeClassifier(random_state=42))
])

# Step 4: Fit and Predict
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

# Step 5: Evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

print("Classification Report:\n", classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Decision Tree')
plt.show()


# Make predictions on both training and testing sets
y_pred_train_decision_tree = pipeline.predict(X_train)
y_pred_test_decision_tree = pipeline.predict(X_test)

# Evaluate the model's performance on both sets
accuracy_train_decision_tree = accuracy_score(y_train, y_pred_train_decis
accuracy_test_decision_tree = accuracy_score(y_test, y_pred_test_decision

# Assess overfitting or underfitting
if accuracy_train_decision_tree  > accuracy_test_decision_tree :
    print("The Decision Tree model might be overfitted.")
elif accuracy_train_decision_tree  < accuracy_test_decision_tree :
    print("The Decision Tree model might be underfitted.")
else:
    print("The Decision Tree model seems well-fitted.")
```

```
Accuracy: 88.69%
Classification Report:
                        precision    recall  f1-score   support

Middle-Class Families       0.89      0.91      0.90       260
   Rural & Low-income       0.93      0.91      0.92       472
    Seniors & Retired       0.86      0.87      0.86       126
   Wealthy & Affluent       0.86      0.89      0.88       126
   Young & Low-income       0.78      0.78      0.78       121

             accuracy                           0.89      1105
            macro avg       0.86      0.87      0.87      1105
         weighted avg       0.89      0.89      0.89      1105
```

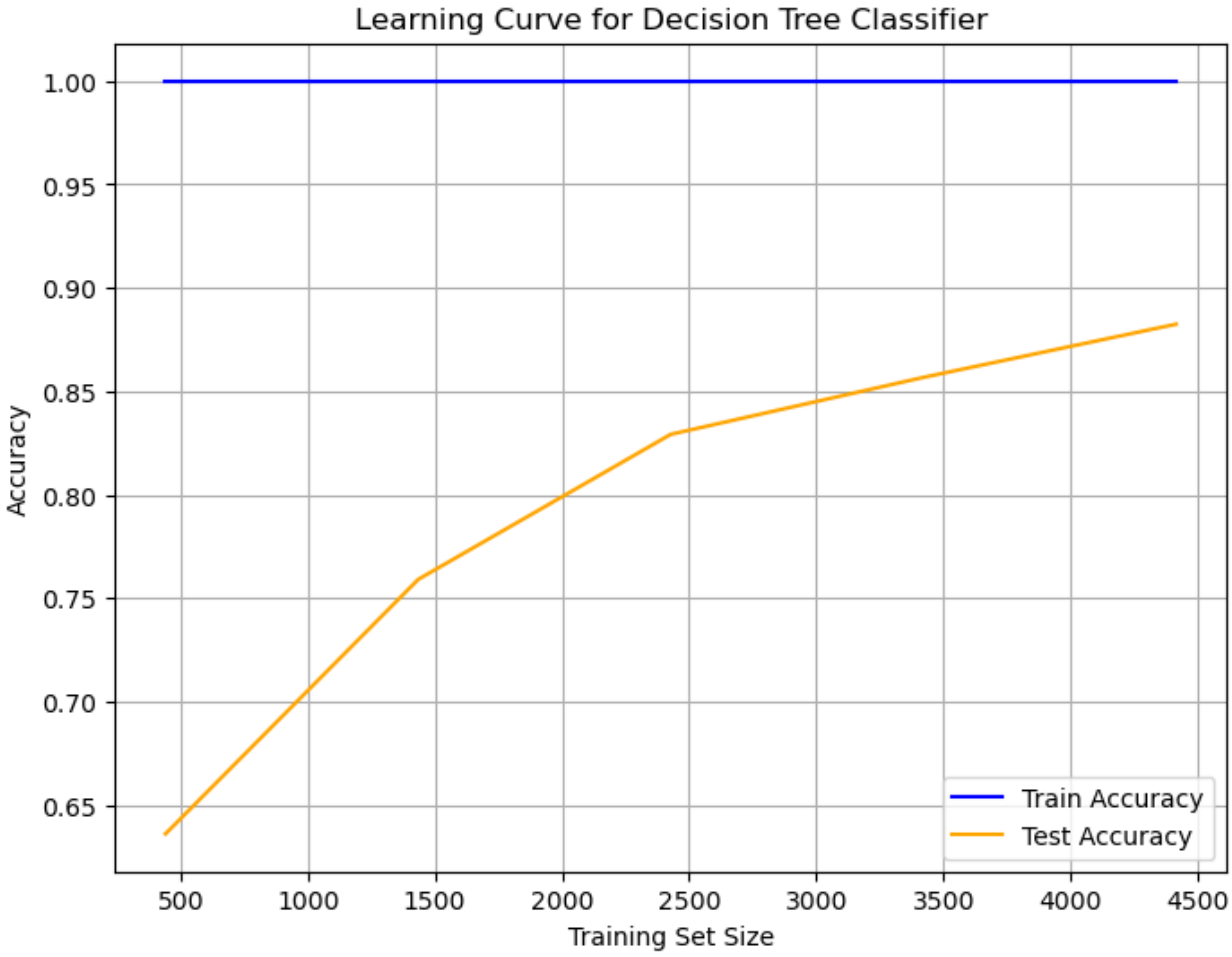


The Decision Tree model might be overfitted.

In [701…

```
# Learning Curve with DTC
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(pipeline, X_selec

# Plot the learning curve
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_scores.mean(axis = 1), label = "Train Accurac
```

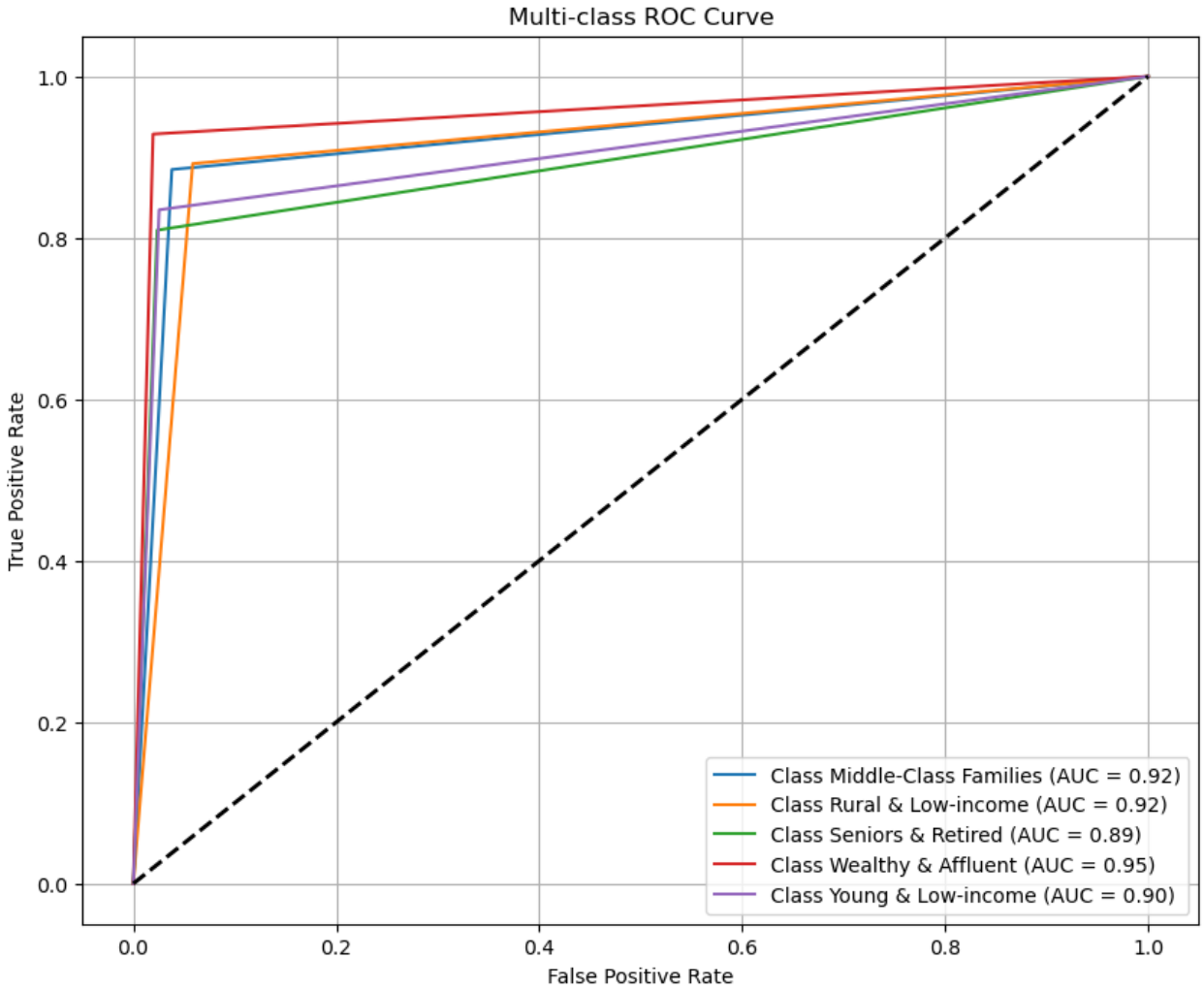
```
plt.plot(train_sizes, test_scores.mean(axis = 1), label = "Test Accuracy"
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Learning Curve for Decision Tree Classifier')
plt.legend()
plt.grid(True)
plt.show()
```



In [705…

```
# ROC curve

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc, classification_report, confus
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Assuming X_pca and y are already defined
X_selected_train, X_selected_test, y_train, y_test = train_test_split(X_s

# Binarize the output for ROC (multi-class)
classes = np.unique(y)
y_test_bin = label_binarize(y_test, classes=classes)
```

```
y_train_bin = label_binarize(y_train, classes=classes)

# Train the classifier using OneVsRest
Dt = OneVsRestClassifier(pipeline)
Dt.fit(X_selected_train, y_train_bin)
y_score = Dt.predict_proba(X_selected_test)

from sklearn.metrics import roc_auc_score

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(10, 8))

for i in range(len(classes)):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multi-class ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```



Support Vector Classifier

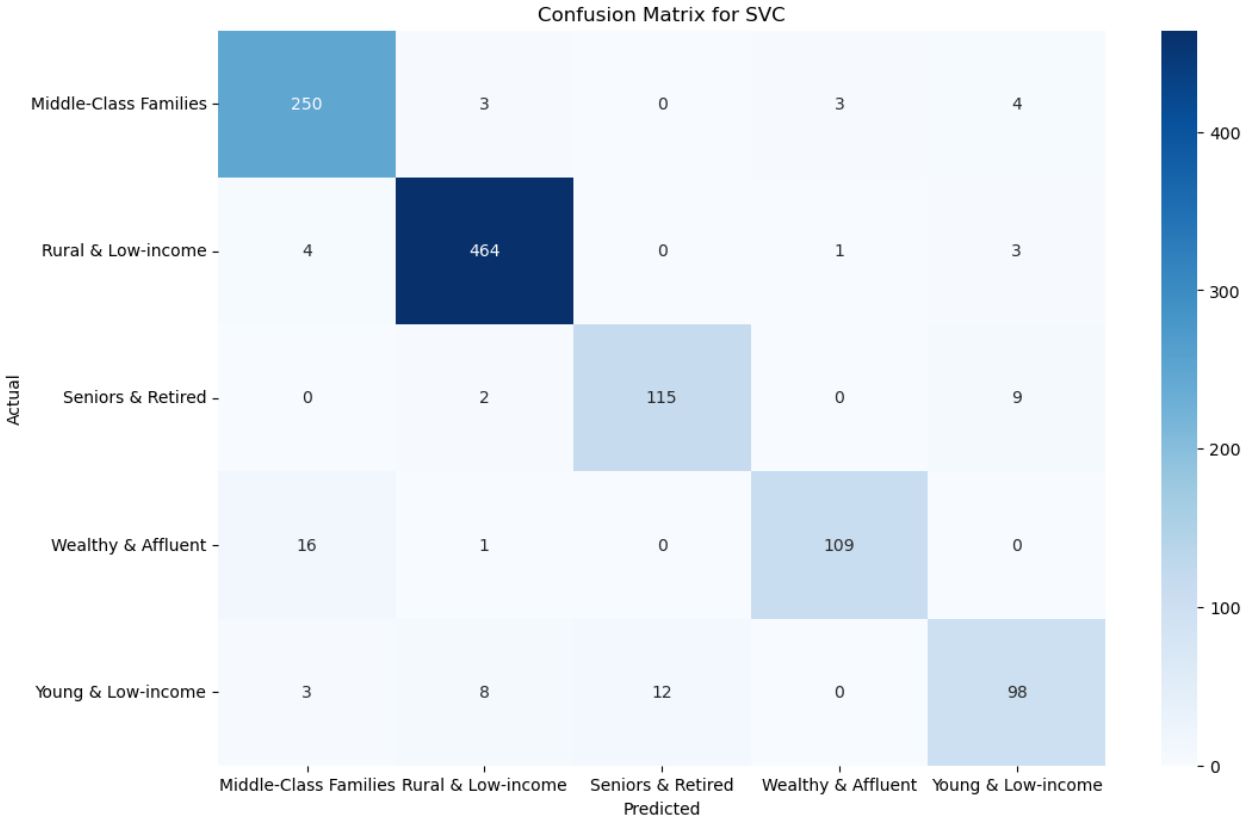
- Without Dimensional Reductions

In [147…
```
y = Insurance_df[['Customer_Type']].values.ravel()
X_train, X_test, y_train, y_test = train_test_split(Insurance_Encoded_df,
```

In [159…
```
from sklearn.svm import SVC

svc_classifier = SVC(random_state = 42)
svc_classifier.fit(X_train, y_train)
y_pred = svc_classifier.predict(X_test)

accuracy = svc_classifier.score(X_test, y_test)
print(f'SVC Classifier Accuracy: {accuracy * 100:.2f}%')


print("Classification Report: \n", classification_report(y_test, y_pred))

cm_svc = confusion_matrix(y_test, y_pred)

plt.figure(figsize = (12, 8))
```

```
sns.heatmap(cm_svc, annot = True, fmt = 'd', cmap = 'Blues', xticklabels
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVC')
plt.show()


# Make predictions on both training and testing sets
y_pred_train_SVC = svc_classifier.predict(X_train)
y_pred_test_SVC = svc_classifier.predict(X_test)

# Evaluate the model's performance on both sets
accuracy_train_SVC = accuracy_score(y_train, y_pred_train_SVC)
accuracy_test_SVC= accuracy_score(y_test, y_pred_test_SVC)

# Assess overfitting or underfitting
if accuracy_train_SVC  > accuracy_test_SVC :
    print("The SVC model might be overfitted.")
elif accuracy_train_SVC  < accuracy_test_SVC :
    print("The SVC model might be underfitted.")
else:
    print("The SVC model seems well-fitted.")
```

```
SVC Classifier Accuracy: 93.76%
Classification Report:
                        precision    recall  f1-score   support

Middle-Class Families       0.92      0.96      0.94       260
   Rural & Low-income       0.97      0.98      0.98       472
    Seniors & Retired       0.91      0.91      0.91       126
   Wealthy & Affluent       0.96      0.87      0.91       126
   Young & Low-income       0.86      0.81      0.83       121

             accuracy                           0.94      1105
            macro avg       0.92      0.91      0.91      1105
         weighted avg       0.94      0.94      0.94      1105
```

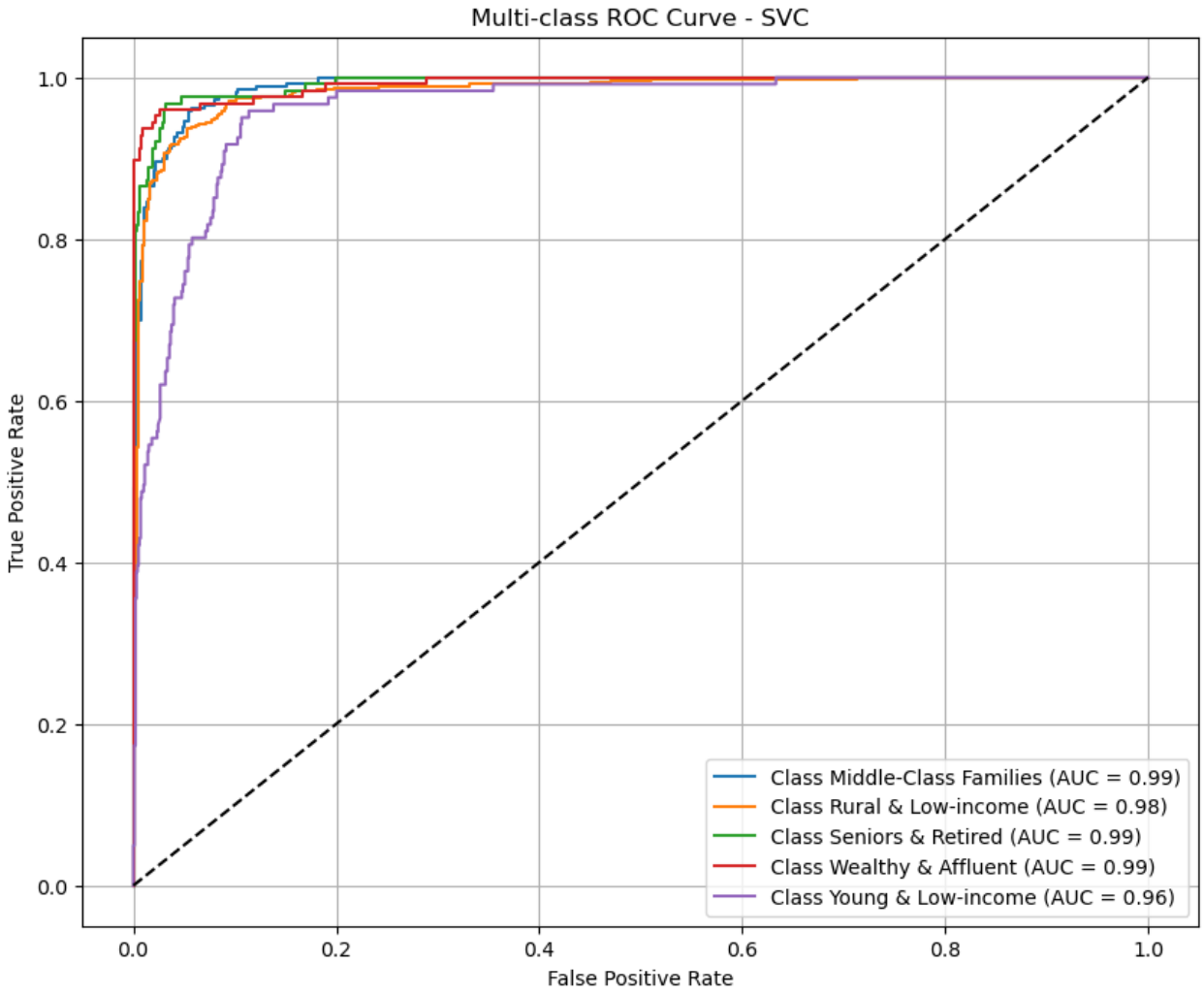


The SVC model might be overfitted.

In [715…

```
#ROC Curve for Multi Class Target Variable (Without Ensemble)

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Binarize y for multi-class ROC
classes = np.unique(y)
y_test_bin = label_binarize(y_test, classes=classes)
y_train_bin = label_binarize(y_train, classes=classes)

# OneVsRest for multi-class support
svc_ovr = OneVsRestClassifier(SVC(random_state = 42, probability = True))
svc_ovr.fit(X_train, y_train_bin)
y_score = svc_ovr.predict_proba(X_test)

# Calculate FPR, TPR for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
# Plot
plt.figure(figsize=(10, 8))
for i in range(len(classes)):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i

plt.plot([0, 1], [0, 1], 'k--', lw=1.5)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multi-class ROC Curve - SVC')
plt.legend()
plt.grid(True)
plt.show()

# Predict the final labels from probabilities
y_pred_bin = svc_ovr.predict(X_test)
y_pred_labels = np.argmax(y_pred_bin, axis=1)
y_test_labels = np.argmax(y_test_bin, axis=1)
```



SVC with Dimensionality Reduction (Ensemble)

In [167…

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confus
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# === Mutual Info Selection (outside the pipeline) ===
from sklearn.feature_selection import mutual_info_classif
from sklearn.decomposition import PCA

# Feature selection
mi = mutual_info_classif(Insurance_Encoded_df, y)
mi_series = pd.Series(mi, index=Insurance_Encoded_df.columns) #Convert Fe
X_selected = Insurance_Encoded_df[mi_series[mi_series > 0].index]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_s
X_selected.columns
```

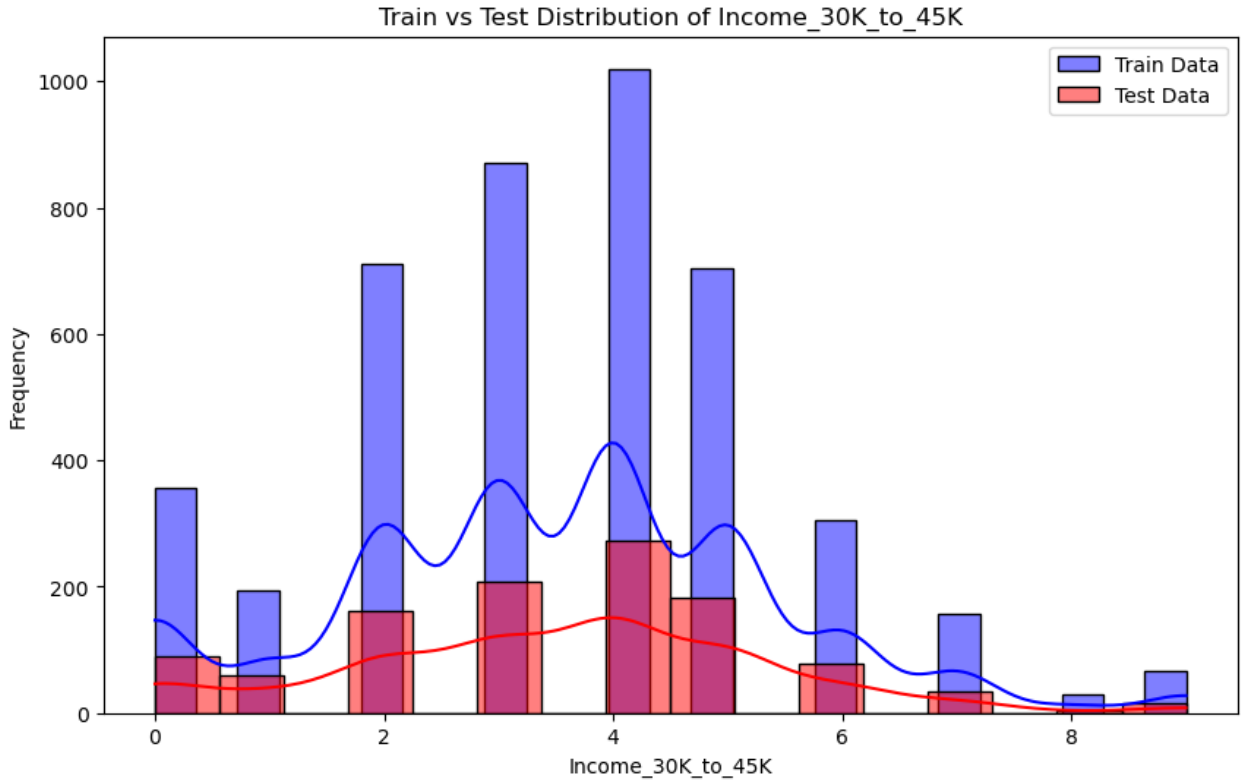
Out[167…

```
Index(['Avg_Age_Encoded', 'Household_Profile_Encoded',
       'Private_Third_Party_Insurance_Contribution_Encoded',
       'Agricultural_Third_Party_Insurance_Contribution_Encoded',
       'Number_of_Houses', 'Avg_Household_Size', 'Married', 'Living_Toge
ther',
       'Other_Relation', 'Singles', 'Household_Without_Children',
       'Household_With_Children', 'High_Education_Level',
       'Medium_Education_Level', 'Low_Education_Level', 'High_Status',
       'Entrepreneur', 'Farmer', 'Middle_Management', 'Skilled_Labourer
s',
       'Unskilled_Labourers', 'Social_Class_A', 'Social_Class_B1',
       'Social_Class_B2', 'Social_Class_C', 'Social_Class_D', 'Rented_Ho
use',
       'Home_Owner', 'Owns_One_Car', 'Owns_Two_Cars', 'Owns_No_Car',
       'National_Health_Insurance', 'Private_Health_Insurance',
       'Income_Less_Than_30K', 'Income_30K_to_45K', 'Income_45K_to_75K',
       'Income_75K_to_122K', 'Income_Above_123K', 'Average_Income',
       'Purchasing_Power_Class', 'Car_Policy_Contribution',
       'Trailer_Policy_Contribution', 'Moped_Policy_Contribution',
       'Life_Insurance_Contribution', 'Disability_Insurance_Contributio
n',
       'Fire_Insurance_Contribution', 'Boat_Insurance_Contribution',
       'Bicycle_Insurance_Contribution',
       'Number_Private_Third_Party_Insurance',
       'Number_Agricultural_Third_Party_Insurance', 'Number_Car_Policie
s',
       'Number_Motorcycle_Scooter_Policies', 'Number_Lorry_Policies',
       'Number_Trailer_Policies', 'Number_Tractor_Policies',
       'Number_Agricultural_Machine_Policies', 'Number_Moped_Policies',
       'Number_Life_Insurances', 'Number_Private_Accident_Insurances',
       'Number_Disability_Insurances', 'Number_Surfboard_Insurances'],
      dtype='object')
```

In [169…

```
# Extract the variable from the training and testing datasets
train_variable = X_train['Income_30K_to_45K']
```

```
test_variable = X_test['Income_30K_to_45K']

# Create the plot
plt.figure(figsize=(10, 6))

# Plot the train data
sns.histplot(train_variable, color='blue', label='Train Data', kde=True)

# Plot the test data
sns.histplot(test_variable, color='red', label='Test Data', kde=True)

# Add labels and legend
plt.xlabel('Income_30K_to_45K')
plt.ylabel('Frequency')
plt.legend()

# Display the plot
plt.title('Train vs Test Distribution of Income_30K_to_45K')
plt.show()
```

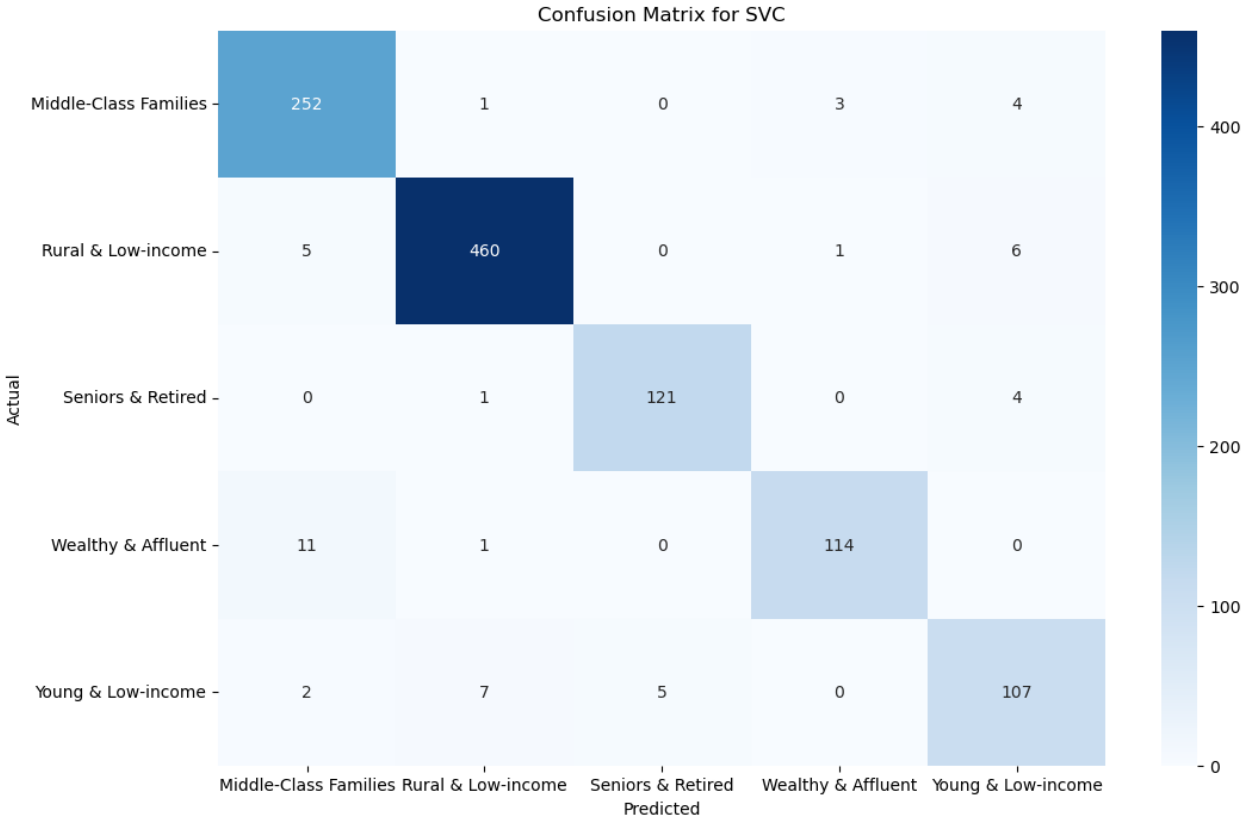


In [171…

```
# === Pipeline ===
pipeline = Pipeline([
    ('pca', PCA(n_components=0.95)),
    ('svc', SVC(random_state=42))
])

# Fit and predict
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

# === Evaluation ===
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'SVC Classifier Accuracy: {accuracy * 100:.2f}%')
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(12, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=pipeline.n
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVC')
plt.show()



# Make predictions on both training and testing sets
y_pred_train_SVC = pipeline.predict(X_train)
y_pred_test_SVC = pipeline.predict(X_test)

# Evaluate the model's performance on both sets
accuracy_train_SVC = accuracy_score(y_train, y_pred_train_SVC)
accuracy_test_SVC = accuracy_score(y_test, y_pred_test_SVC)

# Assess overfitting or underfitting
if accuracy_train_SVC  > accuracy_test_SVC :
    print("The SVC model might be overfitted.")
elif accuracy_train_SVC  < accuracy_test_SVC :
    print("The SVC model might be underfitted.")
else:
    print("The SVC model seems well-fitted.")
```

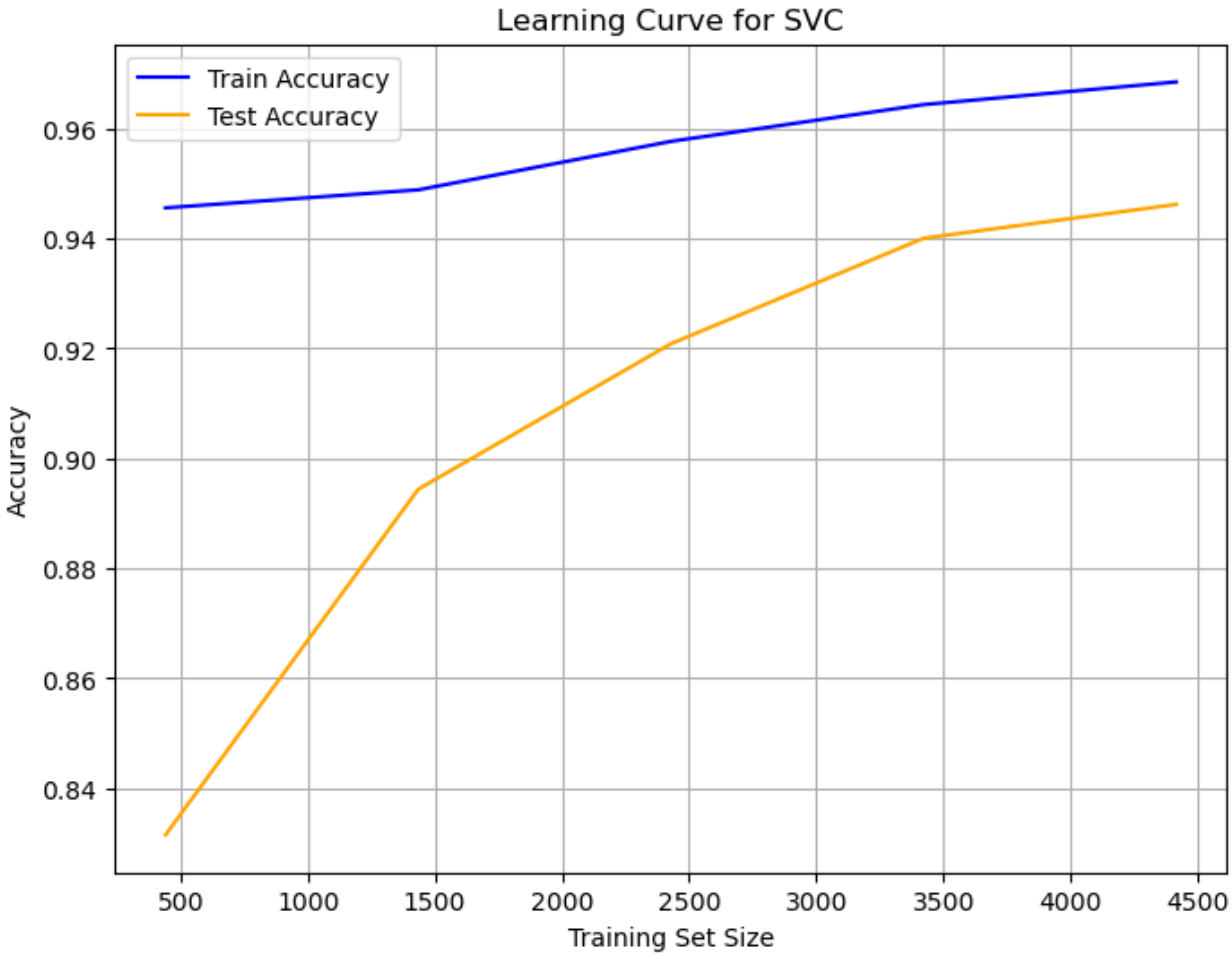
```
SVC Classifier Accuracy: 95.38%
Classification Report:
                        precision    recall  f1-score   support

Middle-Class Families       0.93      0.97      0.95       260
   Rural & Low-income       0.98      0.97      0.98       472
    Seniors & Retired       0.96      0.96      0.96       126
   Wealthy & Affluent       0.97      0.90      0.93       126
   Young & Low-income       0.88      0.88      0.88       121

             accuracy                           0.95      1105
            macro avg       0.94      0.94      0.94      1105
         weighted avg       0.95      0.95      0.95      1105
```



The SVC model might be overfitted.

In [173…

```
# Learning Curve with SVC
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(pipeline, X_selec

# Plot the learning curve
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_scores.mean(axis = 1), label = "Train Accurac
plt.plot(train_sizes, test_scores.mean(axis = 1), label = "Test Accuracy"
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Learning Curve for SVC')
plt.legend()
plt.grid(True)
plt.show()
```

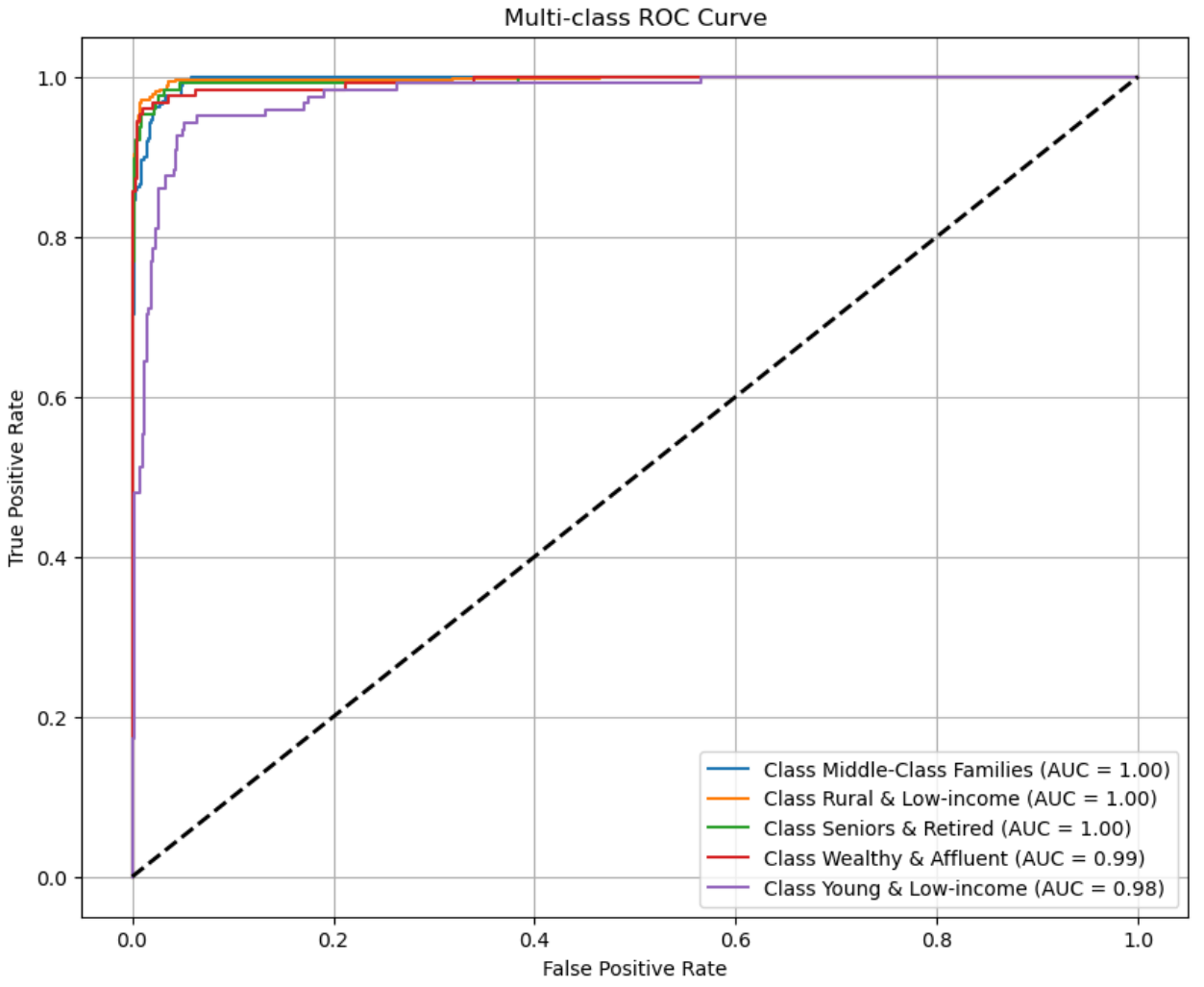


In [721…

```
# ROC curve

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc, classification_report, confus
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Assuming X_pca and y are already defined
X_selected_train, X_selected_test, y_train, y_test = train_test_split(X_s

# Binarize the output for ROC (multi-class)
classes = np.unique(y)
y_test_bin = label_binarize(y_test, classes=classes)
y_train_bin = label_binarize(y_train, classes=classes)

# Train the classifier using OneVsRest
svc = OneVsRestClassifier(pipeline)
svc.fit(X_selected_train, y_train_bin)
y_score = svc.decision_function(X_selected_test)
```

In [723…

```
from sklearn.metrics import roc_auc_score

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(10, 8))

for i in range(len(classes)):
    plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Multi-class ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```



Hyperparameter Tuning

Tuned model - Decision Tree Classifier

In [727…

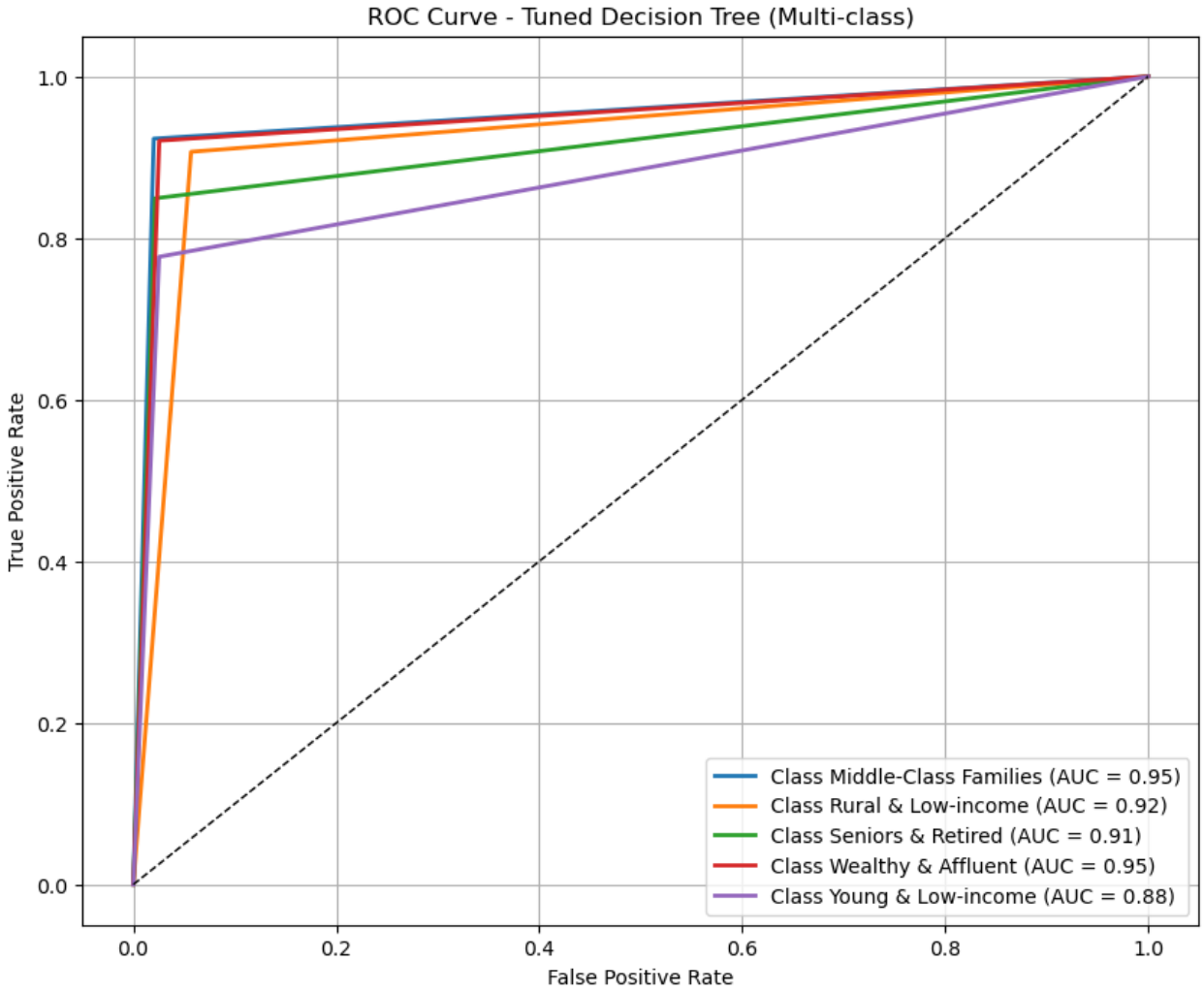
```
# Classification of inputs using Decision Tree
from sklearn.model_selection import GridSearchCV

classifier_decision_tree = DecisionTreeClassifier(random_state = 42)

param_grid = {
    'dt__criterion': ['gini','entropy'],
    'dt__max_depth': [None, 10, 20, 30],
    'dt__min_samples_split': [2, 5, 10],
    'dt__min_samples_leaf': [1, 2, 4]
}

pipeline_classifier_dt = Pipeline([
    ('pca', PCA(n_components = 0.95)),
    ('dt', DecisionTreeClassifier(random_state = 42))
])

y = Insurance_df[['Customer_Type']]
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_s

grid_search_DT = GridSearchCV(estimator = pipeline_classifier_dt, param_g
grid_search_DT.fit(X_train, y_train)

best_parameters_decision_tree = grid_search_DT.best_params_
print('Best Parameters For Decision Tree', best_parameters_decision_tree)

tuned_pipeline_decision_tree = grid_search_DT.best_estimator_
tuned_pipeline_decision_tree.fit(X_train, y_train)

y_pred = tuned_pipeline_decision_tree.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy : {accuracy * 100:.4f}")

report_tuned_decision_tree = classification_report(y_test, y_pred)
print(f"Classification Report: {report_tuned_decision_tree}")
```

```
Best Parameters For Decision Tree {'dt__criterion': 'gini', 'dt__max_dept
h': 20, 'dt__min_samples_leaf': 1, 'dt__min_samples_split': 2}
Accuracy : 91.6742
Classification Report:                        precision    recall  f1-scor
e   support

Middle-Class Families       0.92      0.94      0.93       260
   Rural & Low-income       0.93      0.93      0.93       472
    Seniors & Retired       0.93      0.90      0.91       126
   Wealthy & Affluent       0.88      0.89      0.88       126
   Young & Low-income       0.88      0.87      0.88       121

             accuracy                           0.92      1105
            macro avg       0.91      0.90      0.91      1105
         weighted avg       0.92      0.92      0.92      1105
```

In [729…

```python
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np

# Binarize the labels for multi-class ROC
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
y_train_bin = label_binarize(y_train, classes=classes)

# Wrap your pipeline in a OneVsRest strategy
dt_multiclass = OneVsRestClassifier(tuned_pipeline_decision_tree)
dt_multiclass.fit(X_train, y_train_bin)

# Get probabilities
y_score = dt_multiclass.predict_proba(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(10, 8))
for i in range(len(classes)):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {classes[i]} (AUC = {roc

plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Tuned Decision Tree (Multi-class)')
plt.legend(loc='lower right')
```

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

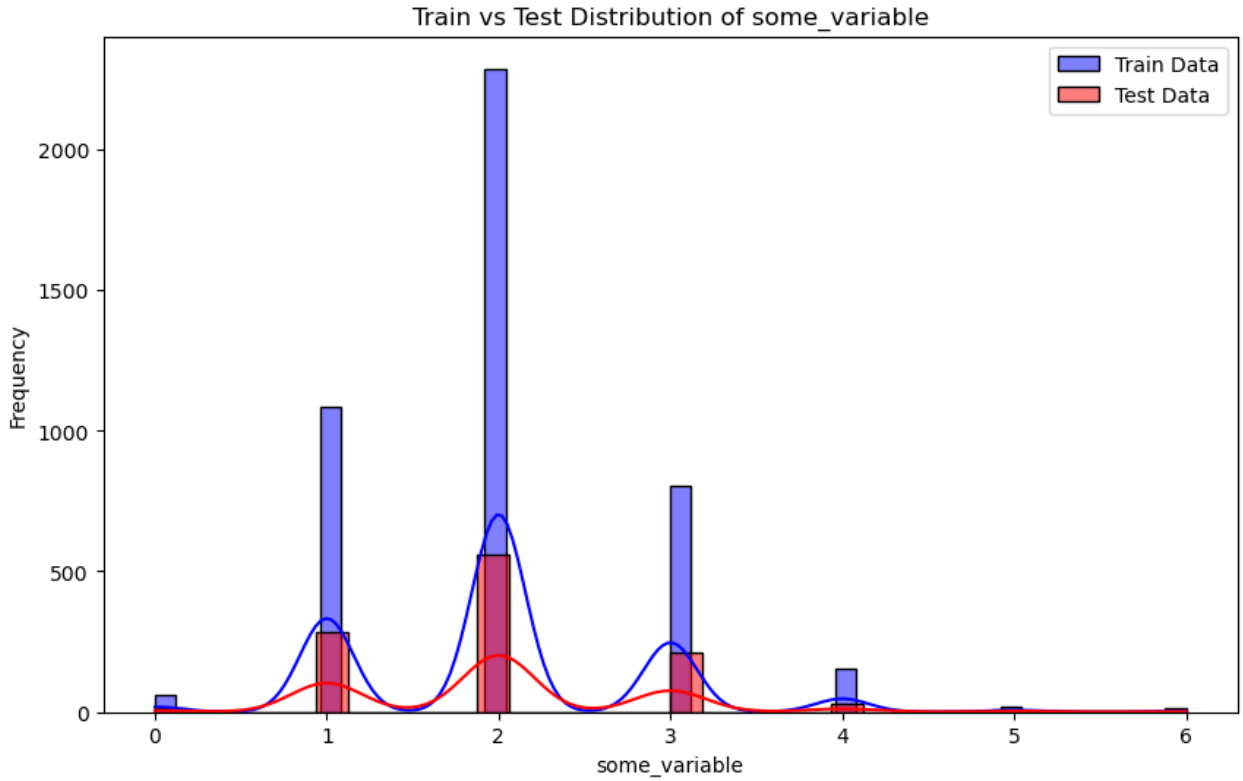


Tuned Model - SVC

In [ ]:
```
# Classification of inputs using Decision Tree
from sklearn.model_selection import GridSearchCV

classifier_svc = SVC(random_state = 42)
```

In [ ]:
```
param_grid = {
     'classifier_svc__C': [0.1, 1, 10], #decision boundary
     'classifier_svc__kernel': ['linear', 'rbf', 'poly'], #line seperatin
     'classifier_svc__gamma': ['scale','auto']
}

pipeline_classifier_svc = Pipeline([
    ('pca', PCA(n_components = 0.95)),
    ('classifier_svc', classifier_svc)
])

y = Insurance_df[['Customer_Type']].values.ravel()
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_s
```

In [158…
```
# Extract the variable from the training and testing datasets
train_variable = X_train['Avg_Age_Encoded']  # Replace 'some_variable' wi
test_variable = X_test['Avg_Age_Encoded']

# Create the plot
plt.figure(figsize=(10, 6))

# Plot the train data
sns.histplot(train_variable, color='blue', label='Train Data', kde=True)

# Plot the test data
sns.histplot(test_variable, color='red', label='Test Data', kde=True)

# Add labels and legend
plt.xlabel('some_variable')
plt.ylabel('Frequency')
plt.legend()

# Display the plot
plt.title('Train vs Test Distribution of some_variable')
plt.show()
```



In [ ]:
```
grid_search_SVC = GridSearchCV(estimator = pipeline_classifier_svc, param
grid_search_SVC.fit(X_train, y_train)

best_parameters_svc = grid_search_SVC.best_params_
print('Best Parameters For SVC', best_parameters_svc)

tuned_pipeline_svc = grid_search_SVC.best_estimator_
tuned_pipeline_svc.fit(X_train, y_train)
```

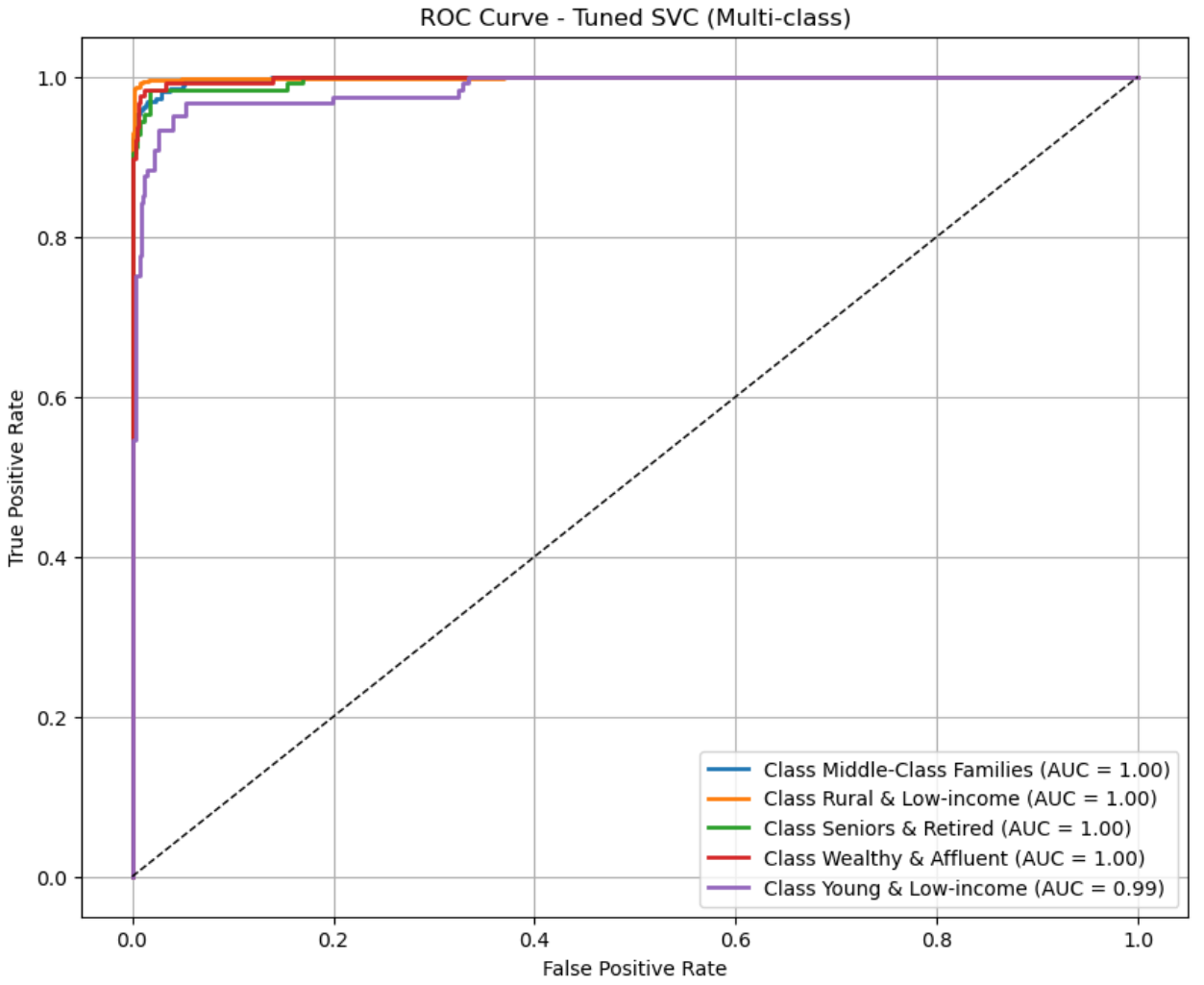
```
y_pred = tuned_pipeline_svc.predict(X_test)
```

In [ ]:
```
# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy : {accuracy * 100:.4f}")

report_tuned_svc = classification_report(y_test, y_pred)
print(f"Classification Report: {report_tuned_svc}")
```

```
Best Parameters For SVC {'classifier_svc__C': 10, 'classifier_svc__gamma':
'scale', 'classifier_svc__kernel': 'rbf'}
Accuracy : 96.3801
Classification Report:                        precision    recall  f1-scor
e   support

Middle-Class Families       0.96      0.98      0.97       260
   Rural & Low-income       0.99      0.97      0.98       472
    Seniors & Retired       0.95      0.94      0.95       126
   Wealthy & Affluent       0.95      0.98      0.96       126
   Young & Low-income       0.91      0.90      0.90       121

             accuracy                           0.96      1105
            macro avg       0.95      0.95      0.95      1105
         weighted avg       0.96      0.96      0.96      1105
```

In [733…
```
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_curve, auc, classification_report, accura
import matplotlib.pyplot as plt
import numpy as np

# Binarize output for multi-class ROC
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
y_train_bin = label_binarize(y_train, classes=classes)

# Wrap your tuned pipeline in OneVsRestClassifier
svc_multiclass = OneVsRestClassifier(tuned_pipeline_svc)
svc_multiclass.fit(X_train, y_train_bin)

# Get decision scores
y_score = svc_multiclass.decision_function(X_test)

# Calculate ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve
```

```
plt.figure(figsize=(10, 8))
for i in range(len(classes)):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {classes[i]} (AUC = {roc

plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Tuned SVC (Multi-class)')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



In [ ]:

Compare the models Decision Tree, SVC and Random Forest

In [ ]:

In [735…

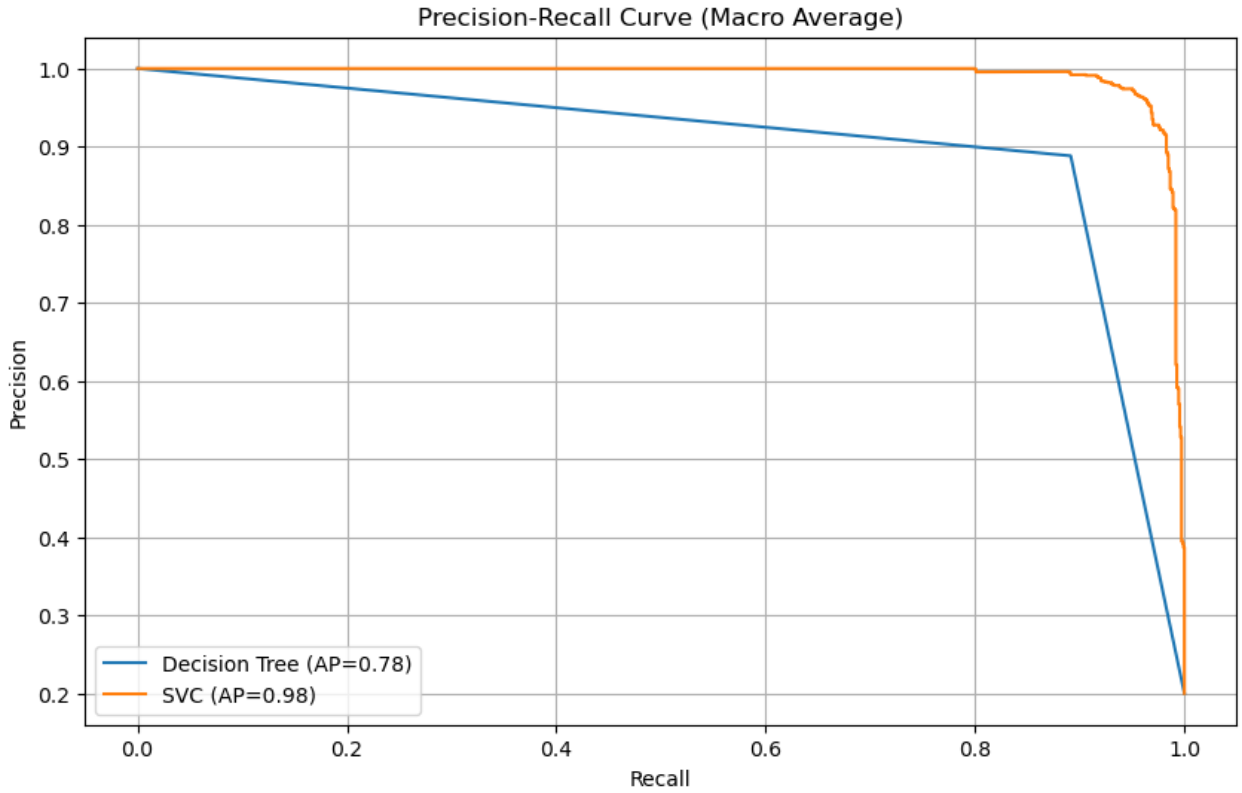
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

classifier_random_forest = RandomForestClassifier(random_state = 3)
```

```
pipeline = {
    ('pca', PCA(n_components = 0.95)),
    ('classifier_random_forest', classifier_random_forest)
}

pipeline_random_forest.fit(X_train, y_train)

y_pred_random_forest = pipeline_random_forest.predict(X_test)
```

In [ ]:
```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import precision_recall_curve, average_precision_sco
from sklearn.multiclass import OneVsRestClassifier

# Binarize y_test and y_train
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
y_train_bin = label_binarize(y_train, classes=classes)

# Wrap both models in OneVsRest strategy
dt_ovr = OneVsRestClassifier(tuned_pipeline_decision_tree)
svc_ovr = OneVsRestClassifier(tuned_pipeline_svc)
rf_ovr = OneVsRestClassifier(pipeline_random_forest)

dt_ovr.fit(X_train, y_train_bin)
svc_ovr.fit(X_train, y_train_bin)
rf_ovr.fit(X_train, y_train_bin)

# Get prediction probabilities / decision scores
probas_decision_tree = dt_ovr.predict_proba(X_test)
probas_svc = svc_ovr.decision_function(X_test)
probas_random_forest = rf_ovr.predict_proba(X_test)
```

In [ ]:
```
# Compute average precision (macro-average over all classes)
average_precision_decision_tree = average_precision_score(y_test_bin, pro
average_precision_svc = average_precision_score(y_test_bin, probas_svc, a

# Precision-Recall Curve (macro-average)
precision_dt, recall_dt, _ = precision_recall_curve(y_test_bin.ravel(), p
precision_svc, recall_svc, _ = precision_recall_curve(y_test_bin.ravel(),
```

In [ ]:
```
# Plot
plt.figure(figsize=(10, 6))
plt.plot(recall_dt, precision_dt, label=f'Decision Tree (AP={average_prec
plt.plot(recall_svc, precision_svc, label=f'SVC (AP={average_precision_sv
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (Macro Average)')
plt.legend()
plt.grid(True)
plt.show()
```



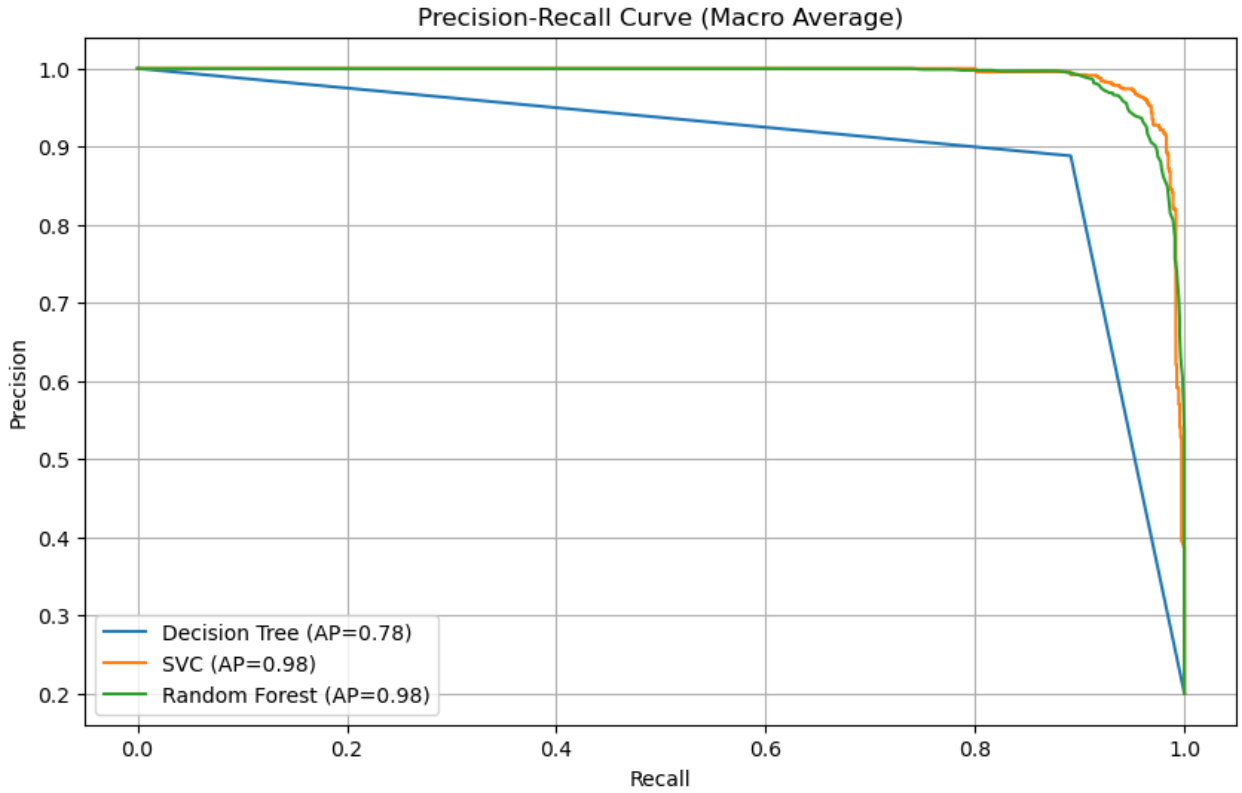
In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import label_binarize
from sklearn.metrics import precision_recall_curve, average_precision_sco
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt
import numpy as np

# Initialize Random Forest classifier
classifier_random_forest = RandomForestClassifier(random_state=3)
```

In [ ]:
```
# Pipeline for Random Forest
pipeline_random_forest = Pipeline([
    ('pca', PCA(n_components=0.95)),  # PCA step
    ('classifier_random_forest', classifier_random_forest)  # Random Fore
])

# Fit the Random Forest model
pipeline_random_forest.fit(X_train, y_train)

# Predict probabilities for Random Forest
probas_random_forest = pipeline_random_forest.predict_proba(X_test)

# Binarize y_test for multi-class to work with precision-recall
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)

# OneVsRest classification for Decision Tree, SVC, and Random Forest
dt_ovr = OneVsRestClassifier(tuned_pipeline_decision_tree)
svc_ovr = OneVsRestClassifier(tuned_pipeline_svc)
rf_ovr = OneVsRestClassifier(pipeline_random_forest)
```

```
dt_ovr.fit(X_train, y_train_bin)
svc_ovr.fit(X_train, y_train_bin)
rf_ovr.fit(X_train, y_train_bin)

# Predict probabilities for each model (Decision Tree, SVC, and Random Fo
probas_decision_tree = dt_ovr.predict_proba(X_test)
probas_svc = svc_ovr.decision_function(X_test)
probas_rf = rf_ovr.predict_proba(X_test)
```

In [ ]:
```
# Compute average precision scores for each model
average_precision_decision_tree = average_precision_score(y_test_bin, pro
average_precision_svc = average_precision_score(y_test_bin, probas_svc, a
average_precision_rf = average_precision_score(y_test_bin, probas_rf, ave

# Compute Precision-Recall curve for each model
precision_dt, recall_dt, _ = precision_recall_curve(y_test_bin.ravel(), p
precision_svc, recall_svc, _ = precision_recall_curve(y_test_bin.ravel(),
precision_rf, recall_rf, _ = precision_recall_curve(y_test_bin.ravel(), p
```

In [ ]:
```
# Plot Precision-Recall curves
plt.figure(figsize=(10, 6))

# Plot each model's Precision-Recall curve
plt.plot(recall_dt, precision_dt, label=f'Decision Tree (AP={average_prec
plt.plot(recall_svc, precision_svc, label=f'SVC (AP={average_precision_sv
plt.plot(recall_rf, precision_rf, label=f'Random Forest (AP={average_prec

# Plot settings
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve (Macro Average)')
plt.legend()
plt.grid(True)
plt.show()
```



Unsupervised Learning

- Without using the labels from the target variable
- Display the plot for KMeans()

Kmeans Clustering

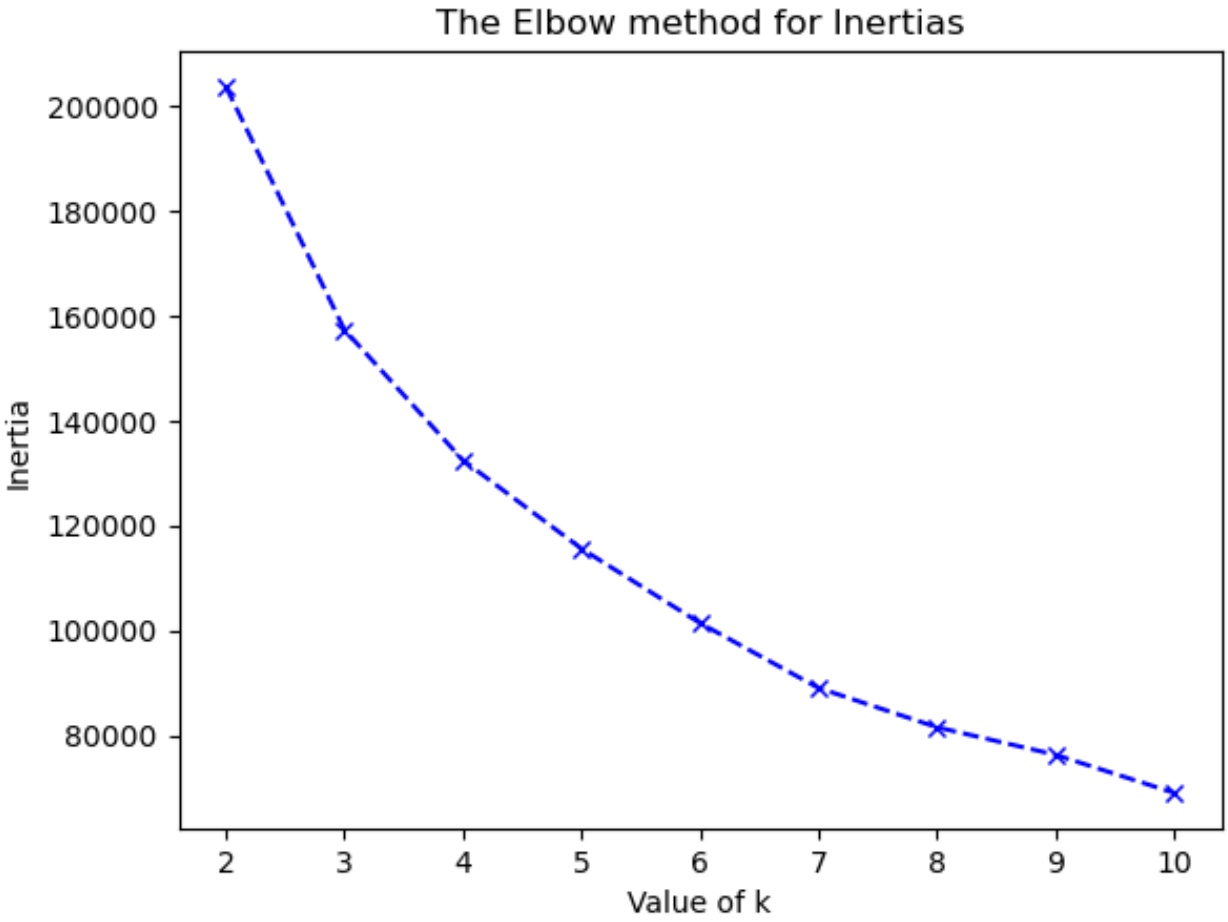
In [184…

```
# Silhouette Score method is used for deciding the right number of cluste
# Clusters range starting from 2, 11 for pattern recognition

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import davies_bouldin_score

inertias = []
silhouette_scores = []

db_scores = []
for k in range(2, 11):
    km = KMeans(n_clusters = k, random_state = 42).fit(X_selected)
    inertias.append(km.inertia_)
    score = silhouette_score(X_selected, km.labels_)
    silhouette_scores.append(score)
    score = davies_bouldin_score(X_selected, km.labels_)
    db_scores.append(score)
```

Number of clusters =3 considered will be as there is a significant drop after the first

point

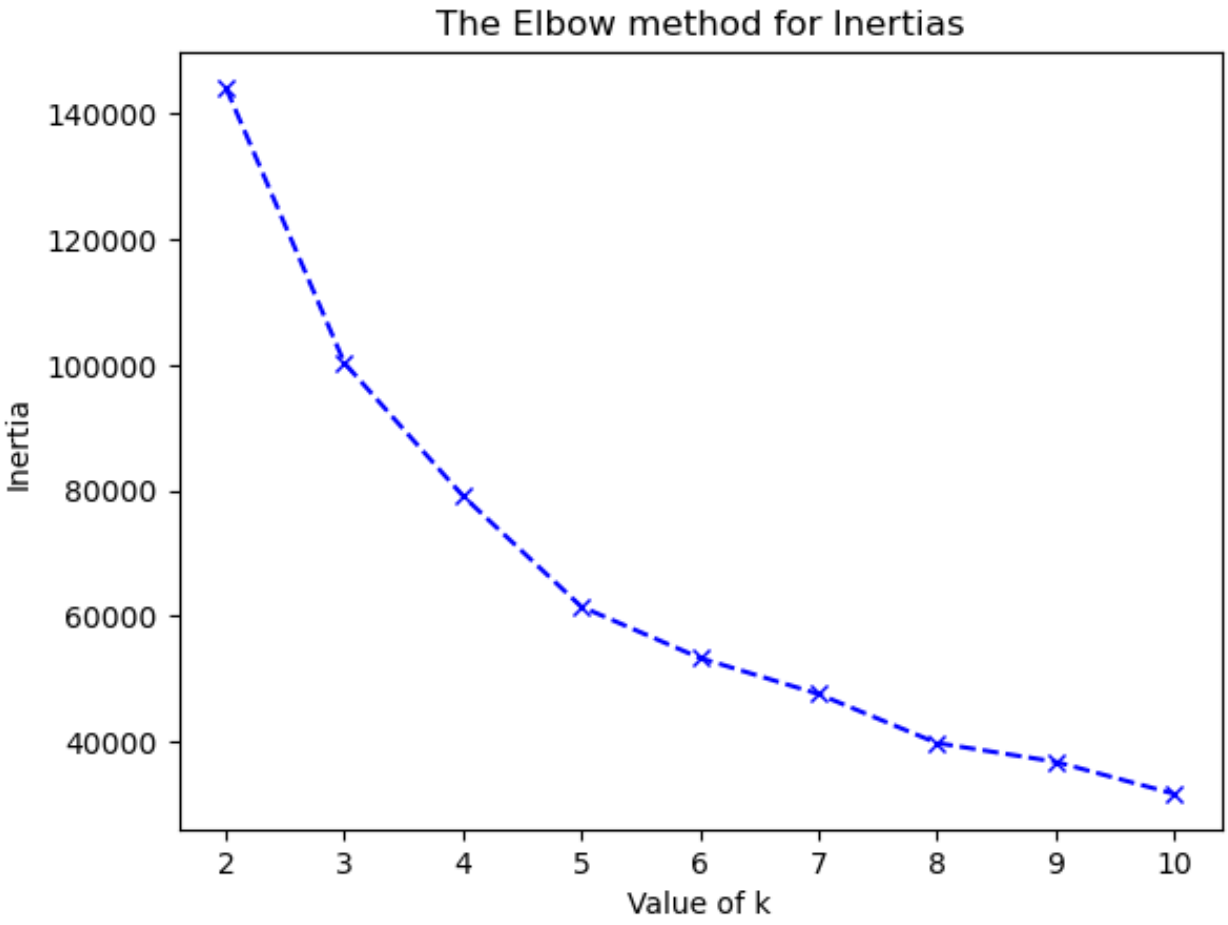
In [130…
```
# Elbow method for KMeans

plt.plot(range(2, 11), inertias, 'bx--')
plt.xlabel('Value of k')
plt.ylabel('Inertia')
plt.title('The Elbow method for Inertias') # Elbow Method
plt.show()
```



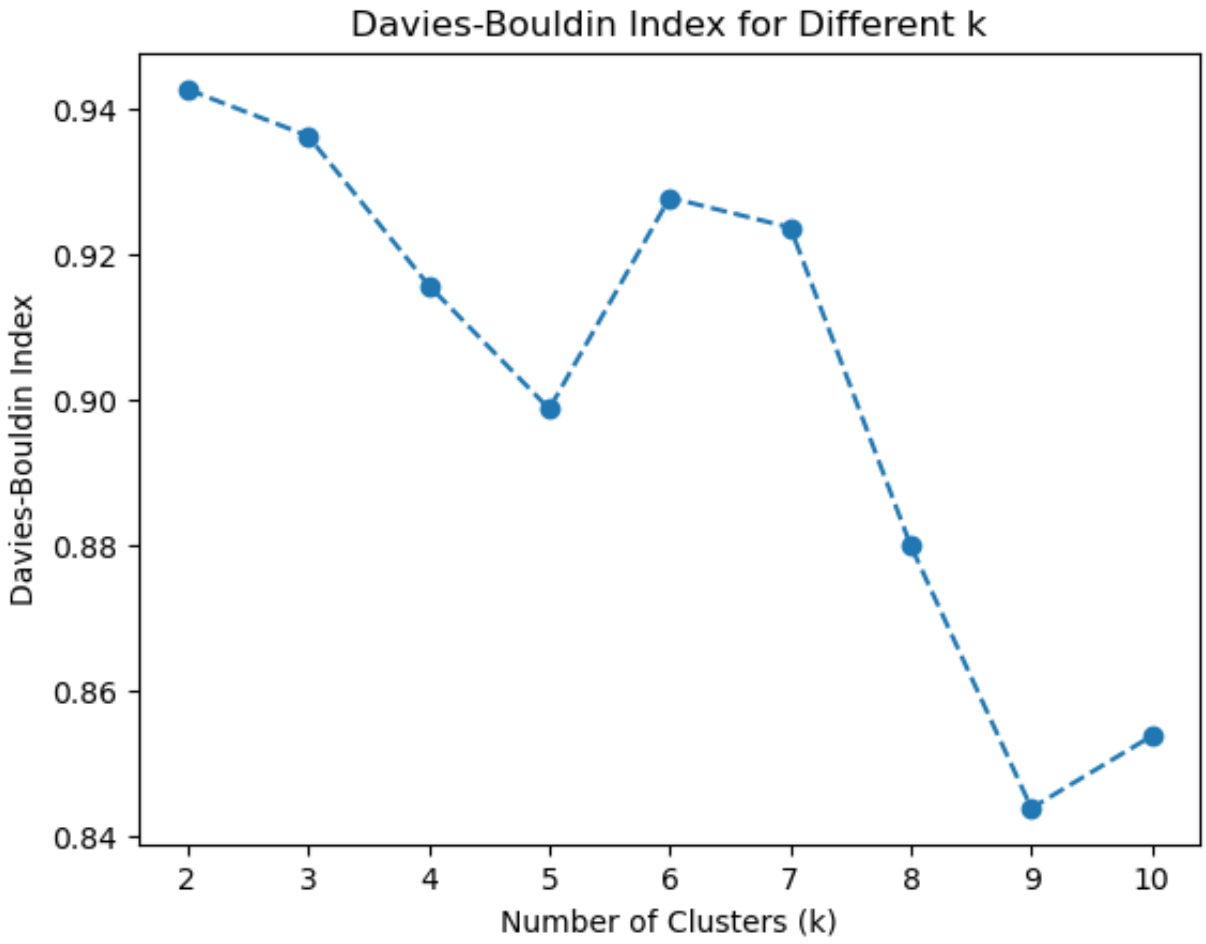
In [186…
```
pca = PCA(n_components = 2)
X_pca_selected = pca.fit_transform(X_selected)
```

In [188…
```
# PCA and Elbow method for KMeans
silhouette_scores = []
inertias = []
db_scores = []
for i in range(2, 11):
    Kmeans = KMeans(n_clusters = i, random_state = 42).fit(X_pca_selected
    inertias.append(Kmeans.inertia_)
    score = silhouette_score(X_pca_selected, Kmeans.labels_)
    silhouette_scores.append(score)
    score = davies_bouldin_score(X_pca_selected, Kmeans.labels_)
    db_scores.append(score)
```

In [189…
```
# Elbow method for KMeans
plt.plot(range(2, 11), inertias, 'bx--')
plt.xlabel('Value of k')
plt.ylabel('Inertia')
plt.title('The Elbow method for Inertias') # Elbow Method
plt.show()
```

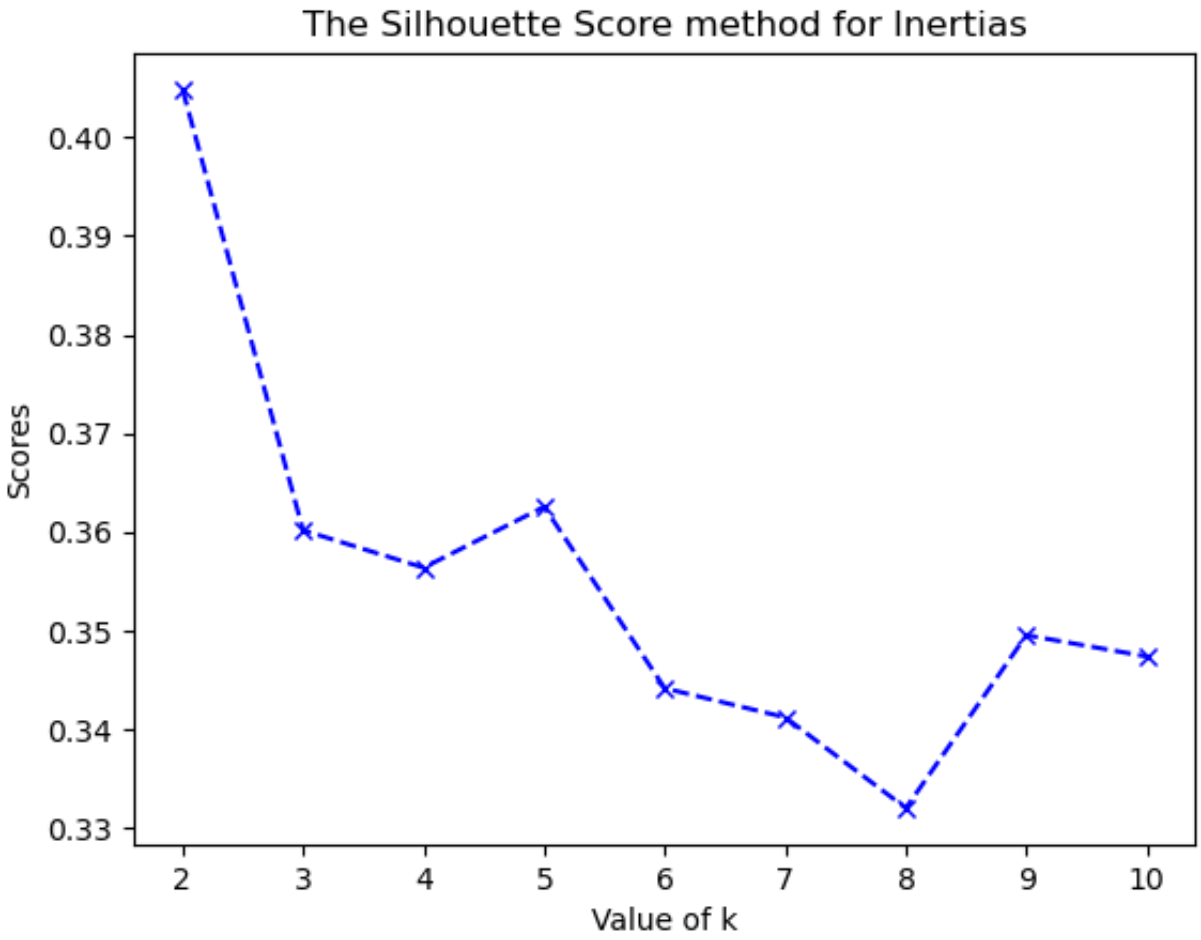


In [131…
```
plt.plot(range(2, 11), db_scores, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Davies-Bouldin Index')
plt.title('Davies-Bouldin Index for Different k')
plt.show()
```



In [130…

```
# Silhouette Score plot

plt.plot(range(2, 11), silhouette_scores, 'bx--')
plt.xlabel('Value of k')
plt.ylabel('Scores')
plt.title('The Silhouette Score method for Inertias')
plt.show()
```



In [130…
```
print(inertias) # k = 5
print(silhouette_scores) # k = 2;
print(db_scores) # k = 9;
```

```
[144063.92148416856, 100079.87277462729, 79028.13475408519, 61414.22115127
732, 54332.73803596271, 47184.68702901757, 42158.507767929346, 35006.27574
554211, 31992.07121294181]
[0.40487133817694215, 0.3601505460472625, 0.3563574999744461, 0.3625403583
058281, 0.34413121496448273, 0.34117553077355156, 0.33197608646210824, 0.3
495145566290804, 0.34735057716388124]
[0.9426688961850108, 0.9362741905417948, 0.9157680544952987, 0.89879094022
6451, 0.927799784336914, 0.9237010010994436, 0.8799690003504868, 0.8439786
248295464, 0.8538977658025605]
```

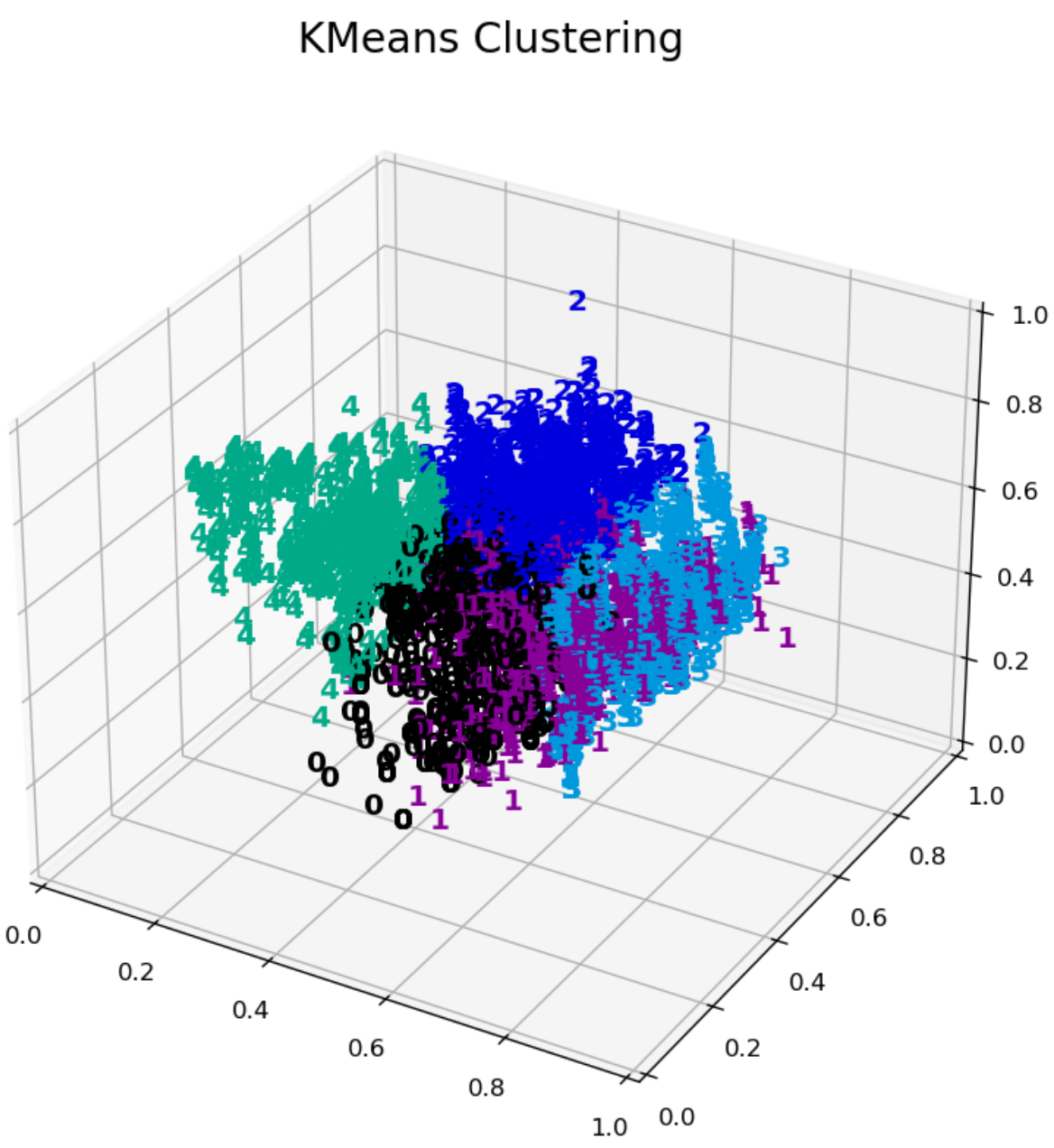
In this case we prefer Elbow Method as better metric. It finds the optimal cluster w.r.t the Supervised Learning which indicates 5 classes.

In [133…
```
# Kmeans model for predicting 5 Clusters

Kmeans = KMeans(n_clusters = 5, random_state = 42).fit(X_pca_selected)
Kmeans_labels = Kmeans.fit_predict(X_pca_selected)
```

3d plot of KMeans model after Dimensionality Reduction

In [132…
```
# 3d plot for KMeans Clustering
```

```
def plot_clustering(X_data, labels, title = None, kx = None):
    x_min, x_max = np.min(X_data, axis = 0), np.max(X_data, axis = 0)
    X_data = (X_data - x_min)/(x_max - x_min)

    for i in range(X_data.shape[0]):
        kx.text(X_data[i, 0], X_data[i, 1], X_data[i, 2], str(labels[i]),
               color = plt.cm.nipy_spectral(labels[i]/10),
               fontweight='bold', fontsize=12)
        if title is not None:
            kx.set_title(title, size = 17)

fig = plt.figure(figsize = (10, 8), dpi = 120)
kx = fig.add_subplot(1,1,1, projection = '3d')


plot_clustering(X_pca_selected, Kmeans.labels_, 'KMeans Clustering', kx)
```

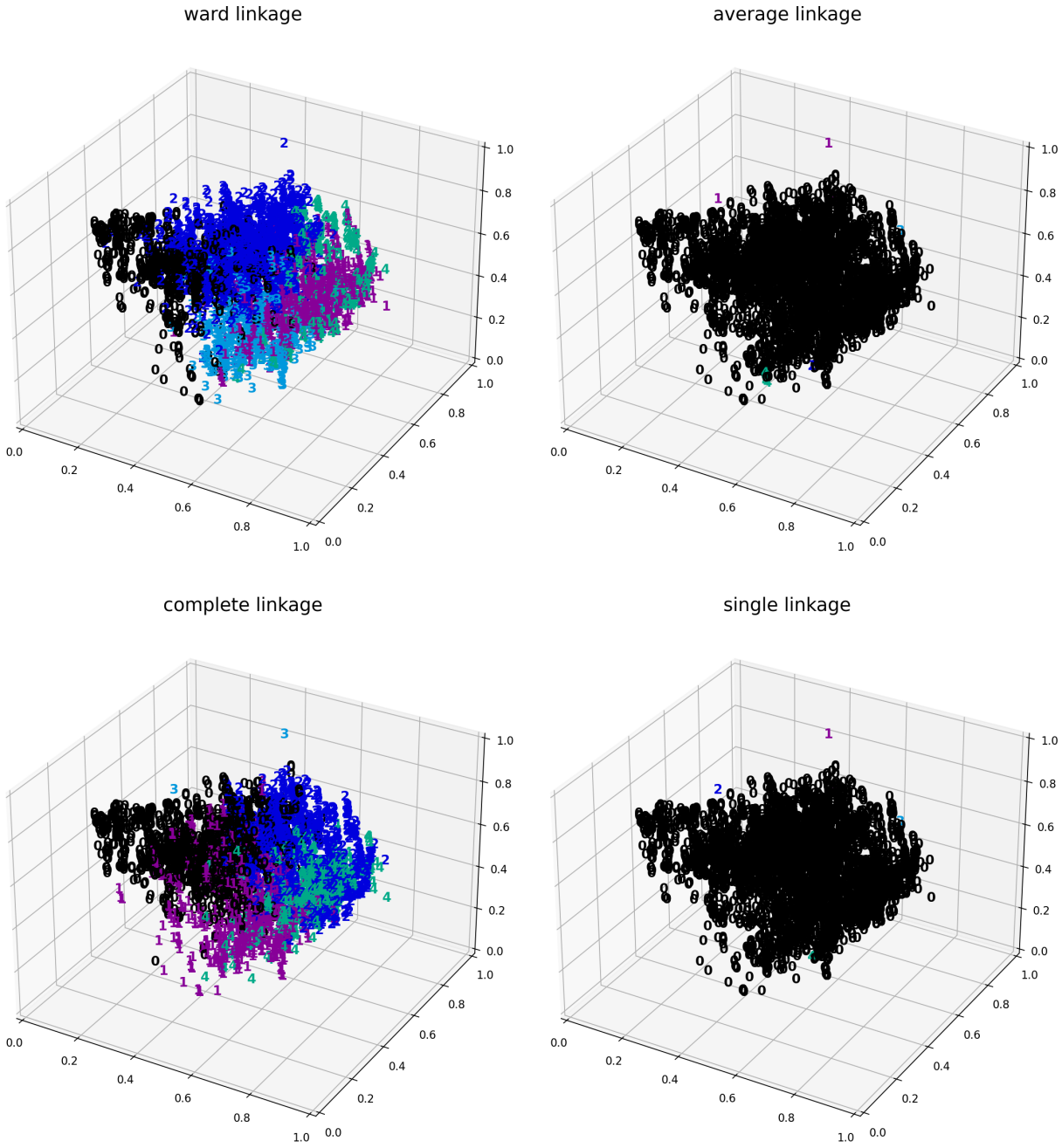


Agglomerative Clustering

In [206…
```
pca = PCA(n_components = 0.95)
X_pca_selected = pca.fit_transform(X_selected)
```

3d plot of Agglomerative Clustering

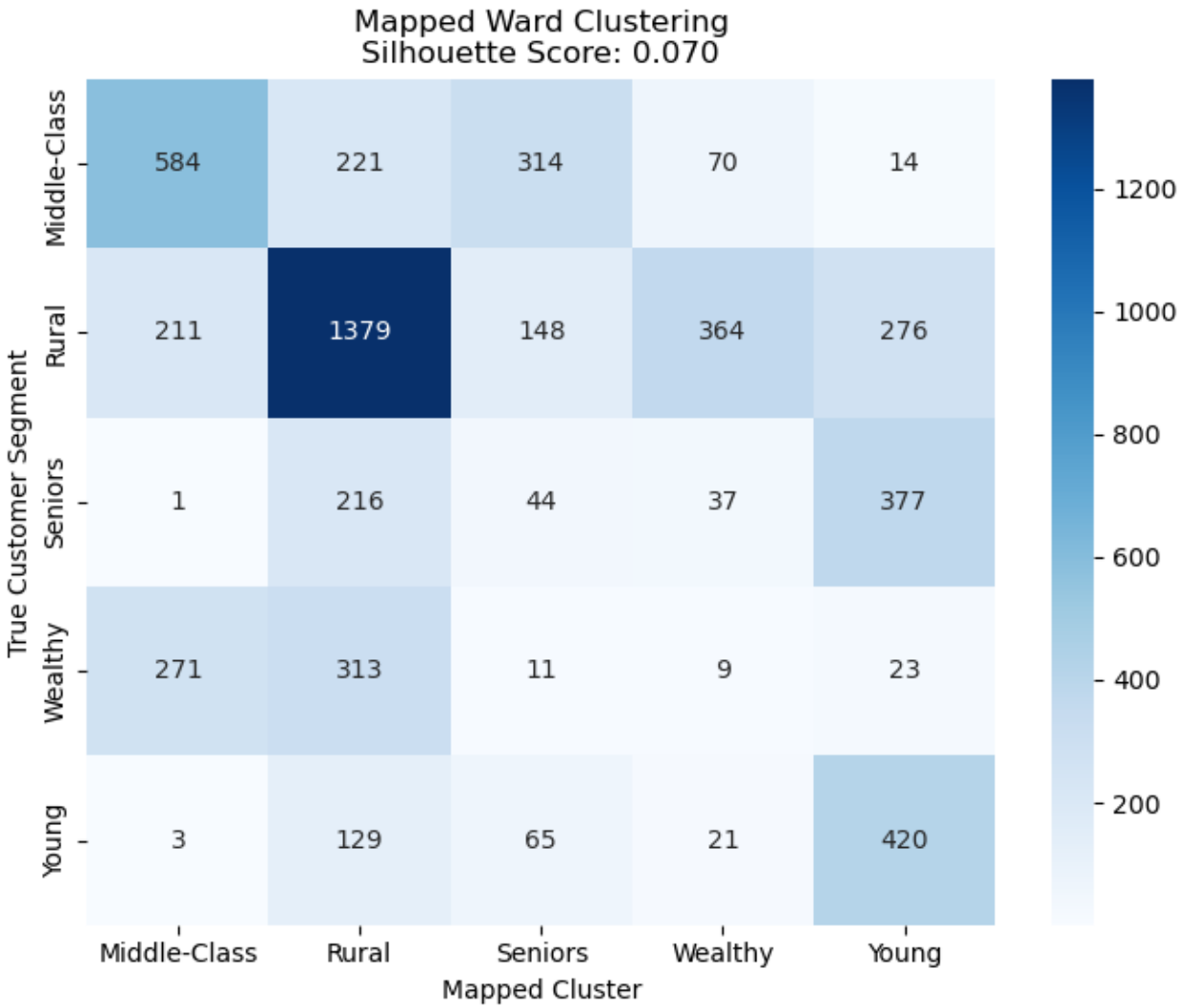
In [208…
```
def plot_clustering(X_data, labels, title=None, ax=None):
    x_min, x_max = np.min(X_data, axis=0), np.max(X_data, axis=0)
    X_data = (X_data - x_min) / (x_max - x_min)

    for i in range(X_data.shape[0]):
        ax.text(X_data[i, 0], X_data[i, 1], X_data[i,2], str(labels[i]),
                color=plt.cm.nipy_spectral(labels[i] / 10),
                fontdict={'weight': 'bold', 'size': 12})

    if title is not None:
        ax.set_title(title, size=17)

from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(14, 18), dpi=160)
ax = fig.add_subplot(2,2,1, projection='3d')
ax1 = fig.add_subplot(2,2,2, projection='3d')
ax2 = fig.add_subplot(2,2,3, projection='3d')
ax3 = fig.add_subplot(2,2,4, projection='3d')

fig.tight_layout(rect=[0, 0.03, 1, 0.95])

from sklearn.cluster import AgglomerativeClustering

for ax_, linkage in zip( (ax,ax1,ax2,ax3),('ward', 'average', 'complete',
    ward_clustering = AgglomerativeClustering(linkage=linkage, n_clusters
    ward_clustering.fit(X_pca_selected)
    plot_clustering(X_pca_selected, ward_clustering.labels_, "%s linkage"
    score = silhouette_score(X_pca_selected, ward_clustering.labels_)
    silhouette_scores.append(score)
```



In [232…

```
from sklearn.metrics import confusion_matrix, adjusted_rand_score
from scipy.optimize import linear_sum_assignment
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Perform Ward linkage clustering
ward_clustering = AgglomerativeClustering(linkage='ward', n_clusters=5)
ward_labels = ward_clustering.fit_predict(X_pca_selected)

# Calculate silhouette score
ward_score = silhouette_score(X_pca_selected, ward_labels)

y_true = true_labels

# Build confusion matrix
```

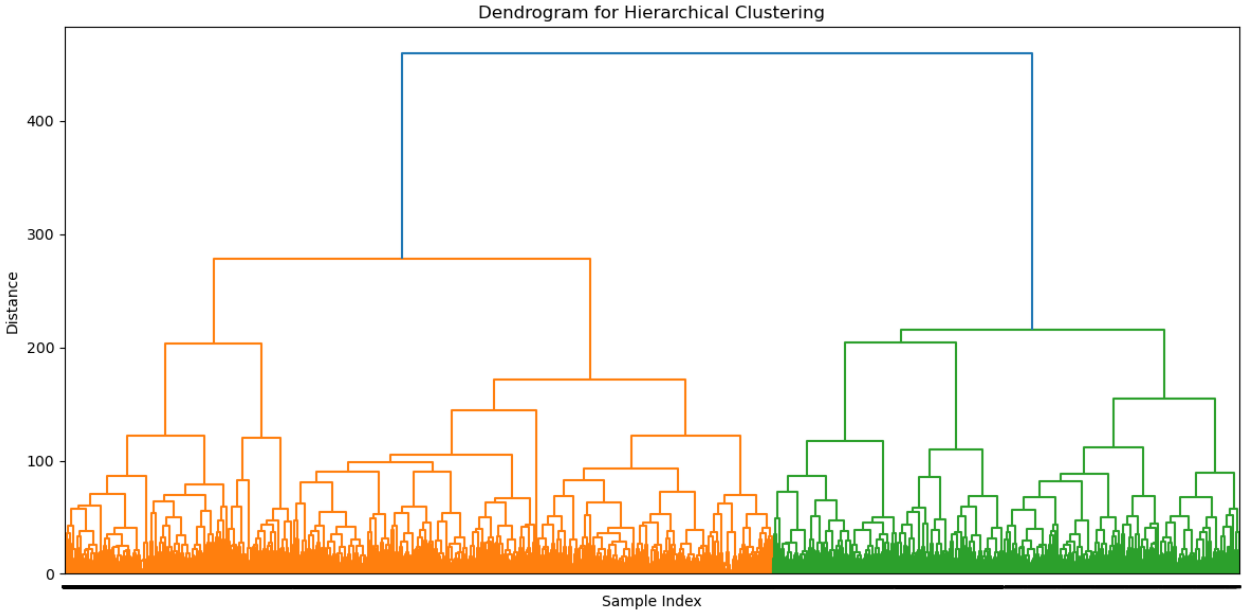
```
cm = confusion_matrix(y_true, ward_labels)

row_ind, col_ind = linear_sum_assignment(-cm)

mapping = dict(zip(col_ind, row_ind))

# Relabel ward_labels using the mapping
mapped_labels = np.array([mapping[label] for label in ward_labels])

# Create new confusion matrix after mapping
cm_mapped = confusion_matrix(y_true, mapped_labels)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm_mapped, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Middle-Class', 'Rural', 'Seniors', 'Wealthy', '
            yticklabels=['Middle-Class', 'Rural', 'Seniors', 'Wealthy', '
plt.title(f'Mapped Ward Clustering\nSilhouette Score: {ward_score:.3f}')
plt.ylabel('True Customer Segment')
plt.xlabel('Mapped Cluster')
plt.show()

accuracy = np.mean(mapped_labels == y_true)
print(f"Post-mapped Accuracy: {accuracy:.4f}")
```




```
Post-mapped Accuracy: 0.4412
```

In [222…

```python
# Dendrogram diagram for Aggolomerative Clustering

from scipy.cluster.hierarchy import linkage, dendrogram
import pandas as pd

linked = linkage(X_pca_selected, method='ward')  # or 'single', 'complete

plt.figure(figsize=(12, 6))
dendrogram(linked, labels = ward_clustering.labels_,
           orientation='top',
           distance_sort='descending',
           show_leaf_counts=True)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.tight_layout()
plt.show()
```

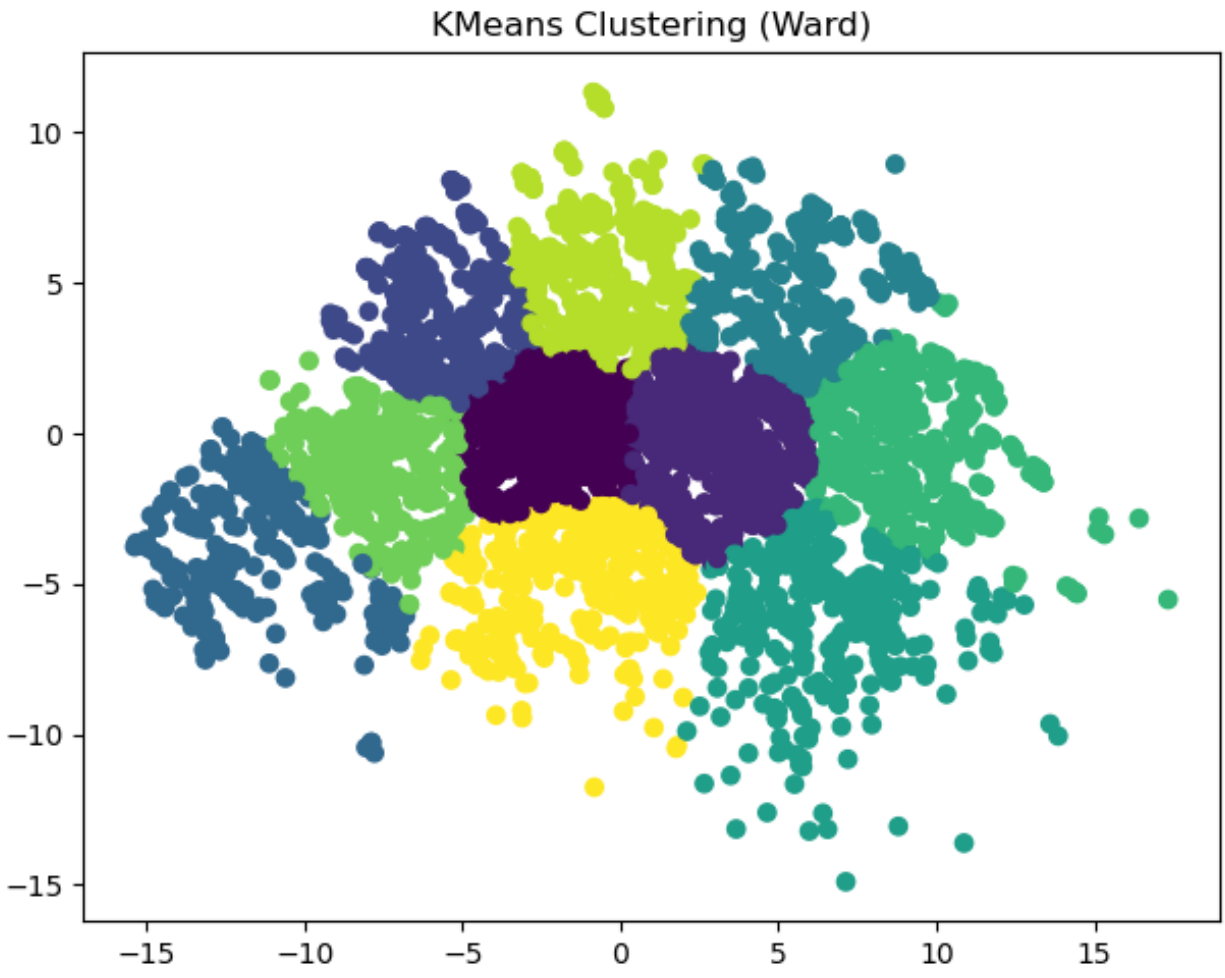


Comparison between KMeans and Hierarchical Clustering

- KMeans is better than Hierarchical Clustering for this dataset

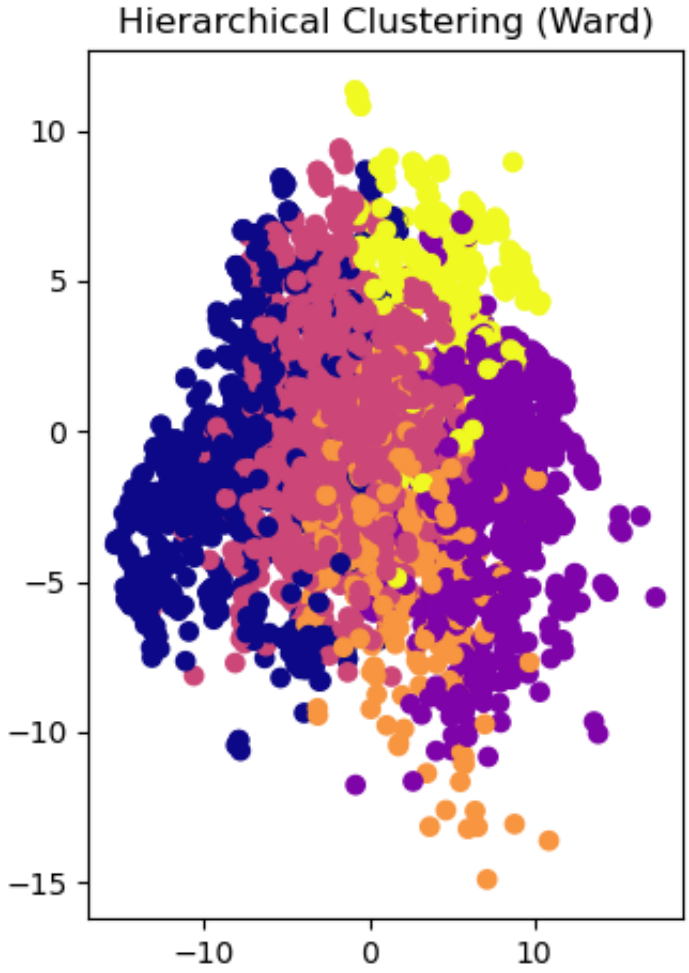
In [220…

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X_pca_selected[:, 0], X_pca_selected[:, 1], c=Kmeans.labels_,
plt.title("KMeans Clustering (Ward)")

plt.tight_layout()
plt.show()
```



In [218…

```
plt.subplot(1, 2, 2)
plt.scatter(X_pca_selected[:, 0], X_pca_selected[:, 1], c=ward_clustering
plt.title("Hierarchical Clustering (Ward)")

plt.tight_layout()
plt.show()
```



Applying Deep Learning using Tensorflow

In [116…
```
!pip install tensorflow
```

```
Collecting tensorflow
  Downloading tensorflow-2.16.2-cp312-cp312-macosx_10_15_x86_64.whl.metada
ta (4.1 kB)
Collecting absl-py>=1.0.0 (from tensorflow)
  Downloading absl_py-2.2.2-py3-none-any.whl.metadata (2.6 kB)
Collecting astunparse>=1.6.0 (from tensorflow)
  Downloading astunparse-1.6.3-py2.py3-none-any.whl.metadata (4.4 kB)
Collecting flatbuffers>=23.5.26 (from tensorflow)
  Downloading flatbuffers-25.2.10-py2.py3-none-any.whl.metadata (875 byte
s)
Collecting gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 (from tensorflow)
  Downloading gast-0.6.0-py3-none-any.whl.metadata (1.3 kB)
Collecting google-pasta>=0.1.1 (from tensorflow)
  Downloading google_pasta-0.2.0-py3-none-any.whl.metadata (814 bytes)
Requirement already satisfied: h5py>=3.10.0 in /opt/anaconda3/lib/python3.
12/site-packages (from tensorflow) (3.11.0)
Collecting libclang>=13.0.0 (from tensorflow)
  Downloading libclang-18.1.1-py2.py3-none-macosx_10_9_x86_64.whl.metadata
(5.2 kB)
```

```
Collecting ml-dtypes~=0.3.1 (from tensorflow)
  Downloading ml_dtypes-0.3.2-cp312-cp312-macosx_10_9_universal2.whl.metad
ata (20 kB)
Collecting opt-einsum>=2.3.2 (from tensorflow)
  Downloading opt_einsum-3.4.0-py3-none-any.whl.metadata (6.3 kB)
Requirement already satisfied: packaging in /opt/anaconda3/lib/python3.12/
site-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.
3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /opt/anaconda3/lib/python3.12/si
te-packages (from tensorflow) (4.25.3)
Requirement already satisfied: requests<3,>=2.21.0 in /opt/anaconda3/lib/p
ython3.12/site-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.1
2/site-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /opt/anaconda3/lib/python3.1
2/site-packages (from tensorflow) (1.16.0)
Collecting termcolor>=1.1.0 (from tensorflow)
  Downloading termcolor-3.0.1-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/anaconda3/
lib/python3.12/site-packages (from tensorflow) (4.11.0)
Requirement already satisfied: wrapt>=1.11.0 in /opt/anaconda3/lib/python
3.12/site-packages (from tensorflow) (1.14.1)
Collecting grpcio<2.0,>=1.24.3 (from tensorflow)
  Downloading grpcio-1.71.0-cp312-cp312-macosx_10_14_universal2.whl.metada
ta (3.8 kB)
Collecting tensorboard<2.17,>=2.16 (from tensorflow)
  Downloading tensorboard-2.16.2-py3-none-any.whl.metadata (1.6 kB)
Collecting keras>=3.0.0 (from tensorflow)
  Downloading keras-3.9.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: numpy<2.0.0,>=1.26.0 in /opt/anaconda3/lib/
python3.12/site-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /opt/anaconda3/lib/py
thon3.12/site-packages (from astunparse>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: rich in /opt/anaconda3/lib/python3.12/site-
packages (from keras>=3.0.0->tensorflow) (13.7.1)
Collecting namex (from keras>=3.0.0->tensorflow)
  Downloading namex-0.0.8-py3-none-any.whl.metadata (246 bytes)
Collecting optree (from keras>=3.0.0->tensorflow)
  Downloading optree-0.15.0-cp312-cp312-macosx_10_13_universal2.whl.metada
ta (48 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/
lib/python3.12/site-packages (from requests<3,>=2.21.0->tensorflow) (3.3.
2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.
12/site-packages (from requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/py
thon3.12/site-packages (from requests<3,>=2.21.0->tensorflow) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/py
thon3.12/site-packages (from requests<3,>=2.21.0->tensorflow) (2025.1.31)
Requirement already satisfied: markdown>=2.6.8 in /opt/anaconda3/lib/pytho
n3.12/site-packages (from tensorboard<2.17,>=2.16->tensorflow) (3.4.1)
Collecting tensorboard-data-server<0.8.0,>=0.7.0 (from tensorboard<2.17,>=
2.16->tensorflow)
```

```
  Downloading tensorboard_data_server-0.7.2-py3-none-macosx_10_9_x86_64.wh
l.metadata (1.1 kB)
Requirement already satisfied: werkzeug>=1.0.1 in /opt/anaconda3/lib/pytho
n3.12/site-packages (from tensorboard<2.17,>=2.16->tensorflow) (3.0.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/anaconda3/lib/pyt
hon3.12/site-packages (from werkzeug>=1.0.1->tensorboard<2.17,>=2.16->tens
orflow) (2.1.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in /opt/anaconda3/li
b/python3.12/site-packages (from rich->keras>=3.0.0->tensorflow) (2.2.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /opt/anaconda3/l
ib/python3.12/site-packages (from rich->keras>=3.0.0->tensorflow) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in /opt/anaconda3/lib/python3.1
2/site-packages (from markdown-it-py>=2.2.0->rich->keras>=3.0.0->tensorflo
w) (0.1.0)
Downloading tensorflow-2.16.2-cp312-cp312-macosx_10_15_x86_64.whl (259.7 M
B)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 259.7/259.7 MB 8.7 MB/s eta 0:
00:0000:0100:01
Downloading absl_py-2.2.2-py3-none-any.whl (135 kB)
Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Downloading flatbuffers-25.2.10-py2.py3-none-any.whl (30 kB)
Downloading gast-0.6.0-py3-none-any.whl (21 kB)
Downloading google_pasta-0.2.0-py3-none-any.whl (57 kB)
Downloading grpcio-1.71.0-cp312-cp312-macosx_10_14_universal2.whl (11.3 M
B)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.3/11.3 MB 9.2 MB/s eta 0:0
0:00ta 0:00:01
Downloading keras-3.9.2-py3-none-any.whl (1.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 6.1 MB/s eta 0:00:0
0
Downloading libclang-18.1.1-py2.py3-none-macosx_10_9_x86_64.whl (26.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.5/26.5 MB 9.4 MB/s eta 0:0
0:00:00:0100:01
Downloading ml_dtypes-0.3.2-cp312-cp312-macosx_10_9_universal2.whl (393 k
B)
Downloading opt_einsum-3.4.0-py3-none-any.whl (71 kB)
Downloading tensorboard-2.16.2-py3-none-any.whl (5.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 9.4 MB/s eta 0:00:0
0ta 0:00:01
Downloading termcolor-3.0.1-py3-none-any.whl (7.2 kB)
Downloading tensorboard_data_server-0.7.2-py3-none-macosx_10_9_x86_64.whl
(4.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.8/4.8 MB 8.2 MB/s eta 0:00:0
0a 0:00:01m
Downloading namex-0.0.8-py3-none-any.whl (5.8 kB)
Downloading optree-0.15.0-cp312-cp312-macosx_10_13_universal2.whl (639 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 639.5/639.5 kB 8.6 MB/s eta 0:
00:00
Installing collected packages: namex, libclang, flatbuffers, termcolor, te
nsorboard-data-server, optree, opt-einsum, ml-dtypes, grpcio, google-past
a, gast, astunparse, absl-py, tensorboard, keras, tensorflow
Successfully installed absl-py-2.2.2 astunparse-1.6.3 flatbuffers-25.2.10
gast-0.6.0 google-pasta-0.2.0 grpcio-1.71.0 keras-3.9.2 libclang-18.1.1 ml
```

```
-dtypes-0.3.2 namex-0.0.8 opt-einsum-3.4.0 optree-0.15.0 tensorboard-2.16.
2 tensorboard-data-server-0.7.2 tensorflow-2.16.2 termcolor-3.0.1
```

In [240…
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras import models, layers, regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
```

In [236…
```
from sklearn.preprocessing import LabelEncoder

X = X_selected
y = Insurance_df[['Customer_Type']].values.ravel()

# Assuming X_selected is your feature matrix and y is your target variabl
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,


# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the labels in the training set
y_train_encoded = label_encoder.fit_transform(y_train)

# Transform the labels in the test set
y_test_encoded = label_encoder.transform(y_test)
```

In [242…
```
# Build a Neural Network

# Define the model architecture
model = models.Sequential()

# Input layer (assumes X_train has raw features)
model.add(layers.InputLayer(shape=(X_train.shape[1],)))

model.add(layers.Dense(128, activation='relu', kernel_regularizer=regular
model.add(layers.Dropout(0.2))
model.add(layers.Dense(64, activation='relu', kernel_regularizer=regulari

#output layer with 5 inputs
model.add(layers.Dense(5, activation='softmax'))  # for multi-class class
```

In [244…
```
# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',  # for integer labe
              metrics=['accuracy'])

# Train the model
```

```
history = model.fit(X_train, y_train_encoded,
                    epochs=50,
                    batch_size=32,
                    validation_data=(X_test, y_test_encoded))
```

```
Epoch 1/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.5094 - loss: 2.6617
- val_accuracy: 0.8199 - val_loss: 1.3852
Epoch 2/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7953 - loss: 1.3471
- val_accuracy: 0.8534 - val_loss: 1.0266
Epoch 3/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8503 - loss: 0.9881
- val_accuracy: 0.8679 - val_loss: 0.8453
Epoch 4/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8654 - loss: 0.8317
- val_accuracy: 0.8959 - val_loss: 0.7028
Epoch 5/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9010 - loss: 0.6868
- val_accuracy: 0.9077 - val_loss: 0.6164
Epoch 6/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8981 - loss: 0.6099
- val_accuracy: 0.9231 - val_loss: 0.5319
Epoch 7/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9190 - loss: 0.5377
- val_accuracy: 0.9176 - val_loss: 0.5014
Epoch 8/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9083 - loss: 0.4996
- val_accuracy: 0.9385 - val_loss: 0.4370
Epoch 9/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9301 - loss: 0.4457
- val_accuracy: 0.9367 - val_loss: 0.4027
Epoch 10/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9391 - loss: 0.4012
- val_accuracy: 0.9475 - val_loss: 0.3868
Epoch 11/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9329 - loss: 0.3883
- val_accuracy: 0.9403 - val_loss: 0.3731
Epoch 12/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9383 - loss: 0.3722
- val_accuracy: 0.9367 - val_loss: 0.3402
Epoch 13/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9386 - loss: 0.3515
- val_accuracy: 0.9520 - val_loss: 0.3245
Epoch 14/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9447 - loss: 0.3332
- val_accuracy: 0.9538 - val_loss: 0.3105
Epoch 15/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9437 - loss: 0.3221
- val_accuracy: 0.9484 - val_loss: 0.3121
Epoch 16/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9471 - loss: 0.3123
- val_accuracy: 0.9566 - val_loss: 0.2861
Epoch 17/50
```

```
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9371 - loss: 0.3090
- val_accuracy: 0.9593 - val_loss: 0.2870
Epoch 18/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9530 - loss: 0.2833
- val_accuracy: 0.9475 - val_loss: 0.2802
Epoch 19/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9467 - loss: 0.2868
- val_accuracy: 0.9439 - val_loss: 0.2875
Epoch 20/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9497 - loss: 0.2827
- val_accuracy: 0.9566 - val_loss: 0.2695
Epoch 21/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9457 - loss: 0.2906
- val_accuracy: 0.9511 - val_loss: 0.2762
Epoch 22/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9480 - loss: 0.2780
- val_accuracy: 0.9602 - val_loss: 0.2717
Epoch 23/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9508 - loss: 0.2713
- val_accuracy: 0.9575 - val_loss: 0.2545
Epoch 24/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9448 - loss: 0.2805
- val_accuracy: 0.9584 - val_loss: 0.2512
Epoch 25/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9588 - loss: 0.2555
- val_accuracy: 0.9520 - val_loss: 0.2589
Epoch 26/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9529 - loss: 0.2612
- val_accuracy: 0.9665 - val_loss: 0.2373
Epoch 27/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9523 - loss: 0.2589
- val_accuracy: 0.9602 - val_loss: 0.2563
Epoch 28/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9572 - loss: 0.2505
- val_accuracy: 0.9602 - val_loss: 0.2497
Epoch 29/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9557 - loss: 0.2405
- val_accuracy: 0.9656 - val_loss: 0.2363
Epoch 30/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9468 - loss: 0.2634
- val_accuracy: 0.9620 - val_loss: 0.2363
Epoch 31/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9418 - loss: 0.2659
- val_accuracy: 0.9602 - val_loss: 0.2377
Epoch 32/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9515 - loss: 0.2469
- val_accuracy: 0.9638 - val_loss: 0.2352
Epoch 33/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9652 - loss: 0.2329
- val_accuracy: 0.9575 - val_loss: 0.2537
Epoch 34/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9483 - loss: 0.2512
- val_accuracy: 0.9538 - val_loss: 0.2352
```

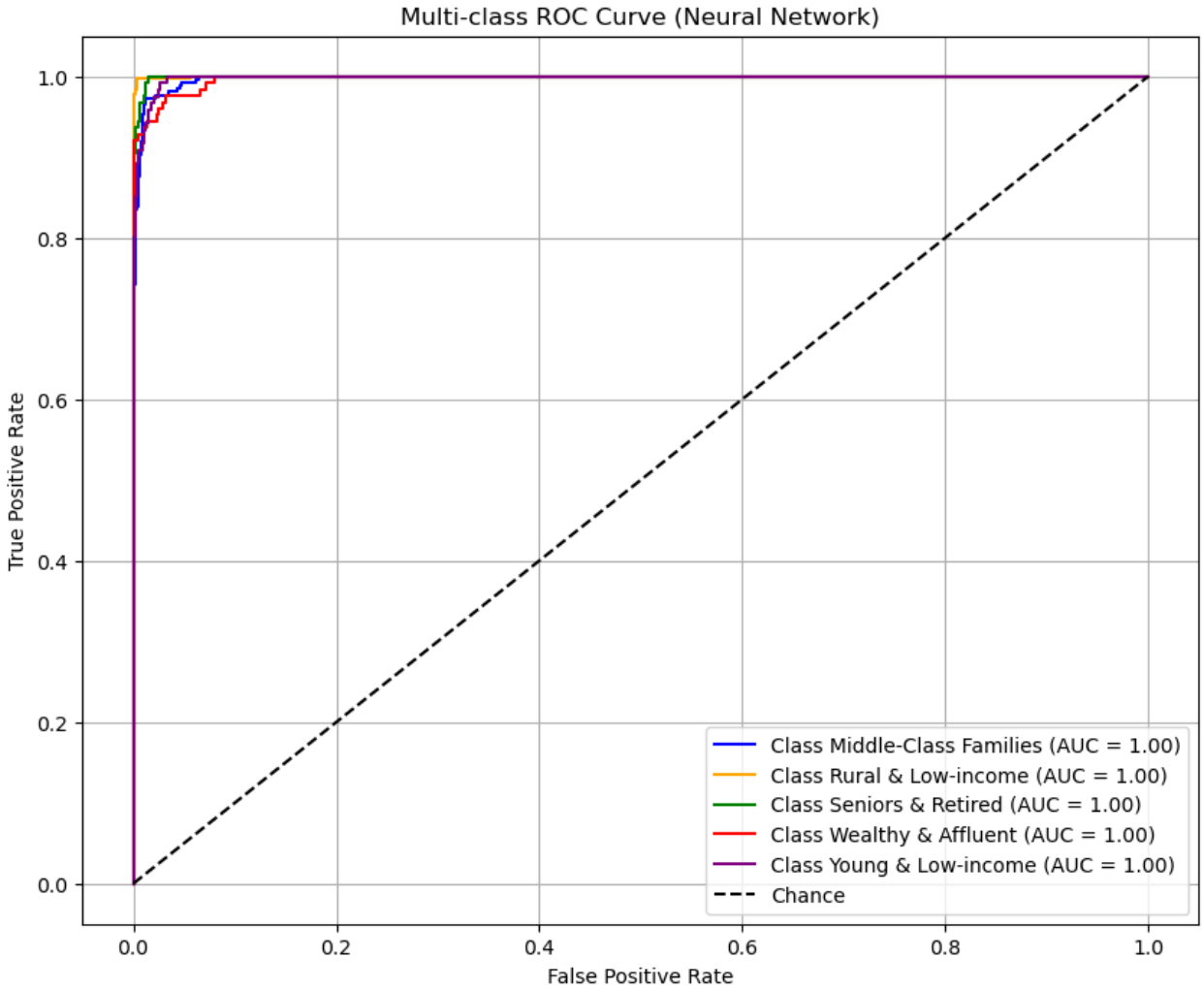
```
Epoch 35/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9598 - loss: 0.2293
- val_accuracy: 0.9511 - val_loss: 0.2492
Epoch 36/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9607 - loss: 0.2344
- val_accuracy: 0.9674 - val_loss: 0.2237
Epoch 37/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9652 - loss: 0.2176
- val_accuracy: 0.9548 - val_loss: 0.2332
Epoch 38/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9591 - loss: 0.2309
- val_accuracy: 0.9620 - val_loss: 0.2311
Epoch 39/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9606 - loss: 0.2192
- val_accuracy: 0.9502 - val_loss: 0.2429
Epoch 40/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9595 - loss: 0.2214
- val_accuracy: 0.9674 - val_loss: 0.2161
Epoch 41/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9562 - loss: 0.2278
- val_accuracy: 0.9448 - val_loss: 0.2626
Epoch 42/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9553 - loss: 0.2300
- val_accuracy: 0.9240 - val_loss: 0.2895
Epoch 43/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9553 - loss: 0.2241
- val_accuracy: 0.9638 - val_loss: 0.2115
Epoch 44/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9615 - loss: 0.2091
- val_accuracy: 0.9502 - val_loss: 0.2368
Epoch 45/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9551 - loss: 0.2269
- val_accuracy: 0.9059 - val_loss: 0.2989
Epoch 46/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9535 - loss: 0.2235
- val_accuracy: 0.9611 - val_loss: 0.2206
Epoch 47/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9655 - loss: 0.1988
- val_accuracy: 0.9548 - val_loss: 0.2486
Epoch 48/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9585 - loss: 0.2223
- val_accuracy: 0.9584 - val_loss: 0.2183
Epoch 49/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9611 - loss: 0.2125
- val_accuracy: 0.9611 - val_loss: 0.2132
Epoch 50/50
138/138 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9647 - loss: 0.1988
- val_accuracy: 0.9593 - val_loss: 0.2040
```

In [248…

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np
```

```
# 1. Get predicted probabilities from the model
y_pred_probs = model.predict(X_test)  # shape: (n_samples, n_classes)

# 2. Binarize the true labels
n_classes = len(np.unique(y_train_encoded))  # should be 5
y_test_bin = label_binarize(y_test_encoded, classes=[0, 1, 2, 3, 4])

# 3. Compute ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# 4. Plot all ROC curves
plt.figure(figsize=(10, 8))
colors = ['blue', 'orange', 'green', 'red', 'purple']
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i],
             label=f'Class {label_encoder.inverse_transform([i])[0]} (AUC

plt.plot([0, 1], [0, 1], 'k--', label='Chance')
plt.title('Multi-class ROC Curve (Neural Network)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

```
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```



In [122…

```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test_encoded)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")

# Get predictions
predictions = model.predict(X_test)
predicted_classes = np.argmax(predictions, axis=1)

# Print the classification report
print(classification_report(y_test_encoded, predicted_classes))
```

```
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9690 - loss: 0.2290
Test Loss: 0.2309829294681549
Test Accuracy: 0.9656108617782593
35/35 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

           0       0.92      0.99      0.96       260
           1       1.00      0.99      0.99       472
           2       0.93      0.98      0.96       126
           3       0.97      0.92      0.94       126
           4       0.99      0.86      0.92       121

    accuracy                           0.97      1105
   macro avg       0.96      0.95      0.95      1105
weighted avg       0.97      0.97      0.97      1105
```

In [152…
```
pca = PCA(n_components = 0.95)
X_pca_selected = pca.fit_transform(X_selected)
```

Accuracy after Mapping the clusters of KMeans

- Seems like KMeans and Hierarchical Clustering
- does not seem to accurately cluster

In [ ]:
```
ca = PCA(n_components = 0.95)
X_pca_selected = pca.fit_transform(X_selected)
```

In [152…
```
Kmeans = KMeans(n_clusters = 5, random_state = 42).fit(X_pca_selected)
score = silhouette_score(X_pca_selected, Kmeans.labels_)
```

In [147…
```
score = silhouette_score(X_pca_selected, clustering.labels_)
```

In [202…
```
y = Insurance_df['Customer_Type'].values.ravel()

from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to the 'Customer_Type' column
y_encoded = label_encoder.fit_transform(Insurance_df['Customer_Type'])
true_labels = y_encoded
# Check the unique encoded labels
print("Encoded labels:", label_encoder.classes_)
print("Encoded labels array:", y_encoded)

dataframe = pd.DataFrame(true_labels, columns = ['y encoded'])
dataframe['y encoded'].unique()
```

```
Encoded labels: ['Middle-Class Families' 'Rural & Low-income' 'Seniors & R
etired'
 'Wealthy & Affluent' 'Young & Low-income']
Encoded labels array: [1 1 1 ... 1 1 1]
```

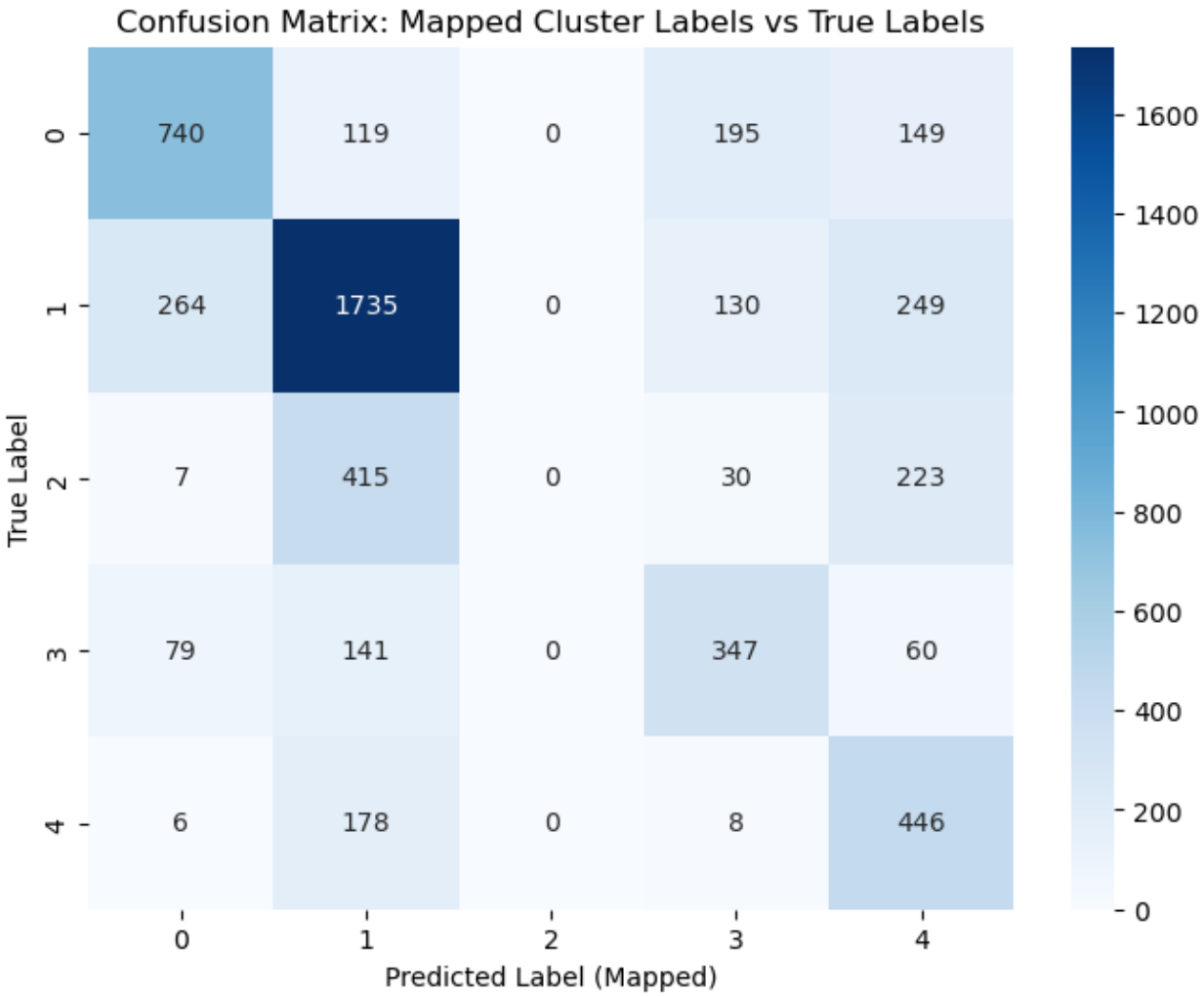
Out[202… array([1, 0, 4, 2, 3])

In [155…
```python
from sklearn.metrics import adjusted_rand_score

# True labels (encoded labels)
true_labels = y_encoded  # Replace with actual true labels

# Cluster labels from KMeans or another clustering algorithm
cluster_labels = Kmeans.labels_

from scipy.stats import mode
import numpy as np

def map_clusters_to_labels(clusters, true_labels):
    labels = np.zeros_like(clusters)
    for i in np.unique(clusters):
        mask = clusters == i
        labels[mask] = mode(true_labels[mask], keepdims=True)[0]
    return labels

mapped_preds = map_clusters_to_labels(cluster_labels, true_labels)
accuracy = np.mean(mapped_preds == true_labels)
print(f"Post-mapped Accuracy: {accuracy:.4f}")
```

```
Post-mapped Accuracy: 0.5919
```

Confusion Matrix of KMeans model with mapped labels

In [155…
```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the confusion matrix
cm = confusion_matrix(true_labels, mapped_preds)

# Plot it
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(true_labels),
            yticklabels=np.unique(true_labels))
plt.xlabel("Predicted Label (Mapped)")
plt.ylabel("True Label")
plt.title("Confusion Matrix: Mapped Cluster Labels vs True Labels")
plt.show()
```

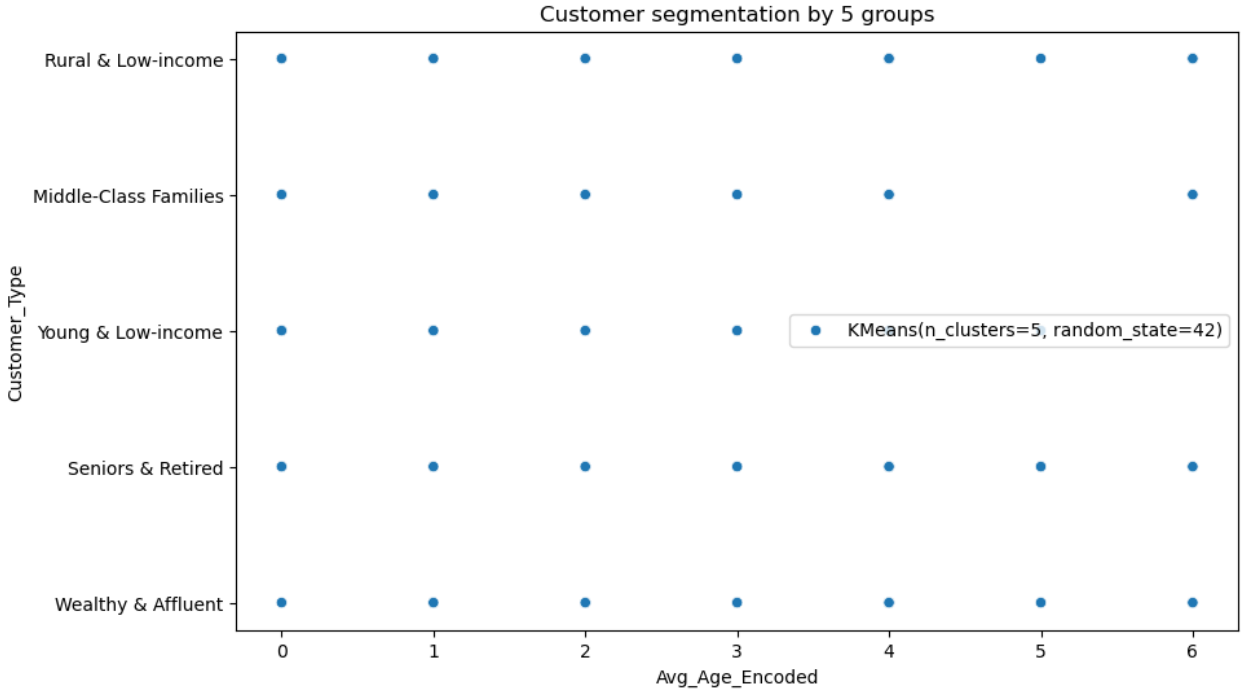


Scatter plot showing the Relation

- After fitting and predictions we use the model to plot the relation between two variables

In [156…

```
plt.figure(figsize=(10,6))
sns.scatterplot(data=X_selected, x='Avg_Age_Encoded', y=Insurance_df['Cus
plt.title('Customer segmentation by 5 groups')
plt.show()
```



In [156…
```
X_selected.columns
```

Out[156…

```
Index(['Avg_Age_Encoded', 'Household_Profile_Encoded',
       'Private_Third_Party_Insurance_Contribution_Encoded',
       'Agricultural_Third_Party_Insurance_Contribution_Encoded',
       'Number_of_Houses', 'Avg_Household_Size', 'Married', 'Living_Toge
ther',
       'Other_Relation', 'Singles', 'Household_Without_Children',
       'Household_With_Children', 'High_Education_Level',
       'Medium_Education_Level', 'Low_Education_Level', 'High_Status',
       'Entrepreneur', 'Farmer', 'Middle_Management', 'Skilled_Labourer
s',
       'Unskilled_Labourers', 'Social_Class_A', 'Social_Class_B1',
       'Social_Class_B2', 'Social_Class_C', 'Social_Class_D', 'Rented_Ho
use',
       'Home_Owner', 'Owns_One_Car', 'Owns_Two_Cars', 'Owns_No_Car',
       'National_Health_Insurance', 'Private_Health_Insurance',
       'Income_Less_Than_30K', 'Income_30K_to_45K', 'Income_45K_to_75K',
       'Income_75K_to_122K', 'Income_Above_123K', 'Average_Income',
       'Purchasing_Power_Class', 'Delivery_Van_Policy_Contribution',
       'Lorry_Policy_Contribution', 'Tractor_Policy_Contribution',
       'Private_Accident_Insurance_Contribution',
       'Fire_Insurance_Contribution', 'Surfboard_Insurance_Contributio
n',
       'Bicycle_Insurance_Contribution',
       'Social_Security_Insurance_Contribution',
       'Number_Private_Third_Party_Insurance',
       'Number_Business_Third_Party_Insurance', 'Number_Car_Policies',
       'Number_Motorcycle_Scooter_Policies', 'Number_Tractor_Policies',
       'Number_Agricultural_Machine_Policies', 'Number_Moped_Policies',
       'Number_Life_Insurances', 'Number_Family_Accident_Insurances',
       'Number_Disability_Insurances', 'Number_Surfboard_Insurances',
       'Number_Bicycle_Insurances', 'Number_Property_Insurances',
       'Number_Social_Security_Insurances'],
      dtype='object')
```

In [ ]: