Task

predict insurance premiums based on various factors.

Here is all the data you need: "sample_submission.csv" "test.csv" "train.csv"

Data loading

Subtask:

Load the provided CSV files into pandas DataFrames.

Reasoning: Load the three CSV files into pandas DataFrames.

```
import pandas as pd

df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
df_submission = pd.read_csv('sample_submission.csv')

display(df_train.head())
display(df_test.head())
display(df_submission.head())
```

th re	Location	...	Previous Claims	Vehicle Age	Credi Scor
61	Urban	...	2.0	17.0	372
31	Rural	...	1.0	12.0	694
49	Suburban	...	1.0	14.0	Na
44	Rural	...	1.0	0.0	367
94	Rural	...	0.0	8.0	598

Health Score	Location	Policy Type	Previous Claims
'.657981	Rural	Basic	NaN
3.381379	Suburban	Premium	NaN
1.354527	Urban	Comprehensive	NaN
5.136225	Suburban	Comprehensive	1.0
1.844155	Suburban	Premium	NaN

Data exploration

Subtask:

Explore the training dataset (`df_train`) to understand its characteristics.

Reasoning: Explore the training dataset (`df_train`) by examining its shape, data types, descriptive statistics, missing values, and data distributions of numerical and categorical features. This will provide initial insights into the data's characteristics, potential outliers, and relationships between variables.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Data Shape and Info
print("Shape of df_train:", df_train.shape)
print("\nInfo of df_train:")
display(df_train.info())

# Descriptive Statistics
print("\nDescriptive Statistics of Numerical Features:")
display(df_train.describe())

# Missing Values
print("\nMissing Values:")
missing_values = df_train.isnull().sum()
missing_percentage = (missing_values / len(df_train)) * 100
missing_df = pd.DataFrame({'Missing Values': missing_values, 'Percentage':
```

```
missing_percentage})
display(missing_df)

# Visualization of missing data pattern (heatmap)
plt.figure(figsize=(12, 6))
sns.heatmap(df_train.isnull(), cbar=False, yticklabels=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()

# Data Distribution
numerical_features = df_train.select_dtypes(include=['number']).columns
categorical_features = df_train.select_dtypes(exclude=['number']).columns

# Histograms for numerical features
for col in numerical_features:
  plt.figure(figsize=(8, 4))
  sns.histplot(df_train[col], kde=True)
  plt.title(f'Distribution of {col}')
  plt.show()

# Value counts for categorical features
for col in categorical_features:
  print(f"\nValue counts for {col}:")
  display(df_train[col].value_counts())

# Relationships between numerical features and target variable
for col in numerical_features:
  if col != 'Premium Amount' and col != 'id':
    plt.figure(figsize=(8, 4))
    sns.scatterplot(x=col, y='Premium Amount', data=df_train)
    plt.title(f'Relationship between {col} and Premium Amount')
```

```
    plt.show()

# Relationships between categorical features and target variable
for col in categorical_features:
  if col != 'Policy Start Date':  # Exclude date column for boxplot
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=col, y='Premium Amount', data=df_train)
    plt.title(f'Relationship between {col} and Premium Amount')
    plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
    plt.show()

# Outlier detection (using boxplots)
for col in numerical_features:
  if col != 'Premium Amount' and col != 'id':
      plt.figure(figsize=(8, 4))
      sns.boxplot(x=col, data=df_train)
      plt.title(f'Boxplot of {col}')
      plt.show()
```

```
Shape of df_train: (1200000, 21)

Info of df_train:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200000 entries, 0 to 1199999
Data columns (total 21 columns):
 #   Column                Non-Null Count    Dtype  
---  ------                --------------    -----  
 0   id                    1200000 non-null  int64  
 1   Age                   1181295 non-null  float64
 2   Gender                1200000 non-null  object 
 3   Annual Income         1155051 non-null  float64
 4   Marital Status        1181471 non-null  object 
 5   Number of Dependents  1090328 non-null  float64
 6   Education Level       1200000 non-null  object 
 7   Occupation            841925 non-null   object 
 8   Health Score          1125924 non-null  float64
 9   Location              1200000 non-null  object 
 10  Policy Type           1200000 non-null  object 
 11  Previous Claims       835971 non-null   float64
 12  Vehicle Age           1199994 non-null  float64
 13  Credit Score          1062118 non-null  float64
 14  Insurance Duration    1199999 non-null  float64
 15  Policy Start Date     1200000 non-null  object 
 16  Customer Feedback     1122176 non-null  object 
 17  Smoking Status        1200000 non-null  object 
 18  Exercise Frequency    1200000 non-null  object 
 19  Property Type         1200000 non-null  object 
 20  Premium Amount        1200000 non-null  float64
dtypes: float64(9), int64(1), object(11)
memory usage: 192.3+ MB
None

Descriptive Statistics of Numerical Features:
```

	id	Age	Annual Income	Number of Dependents	Health Score	Previous Claims	Vehicle Age	Credit Score	Ins Du
count	1.200000e+06	1.181295e+06	1.155051e+06	1.090328e+06	1.125924e+06	835971.000000	1.199994e+06	1.062118e+06	1.1999
mean	5.999995e+05	4.114556e+01	3.274522e+04	2.009934e+00	2.561391e+01	1.002689	9.569889e+00	5.929244e+02	5.0182
std	3.464103e+05	1.353995e+01	3.217951e+04	1.417338e+00	1.220346e+01	0.982840	5.776189e+00	1.499819e+02	2.5943
min	0.000000e+00	1.800000e+01	1.000000e+00	0.000000e+00	2.012237e+00	0.000000	0.000000e+00	3.000000e+02	1.0000

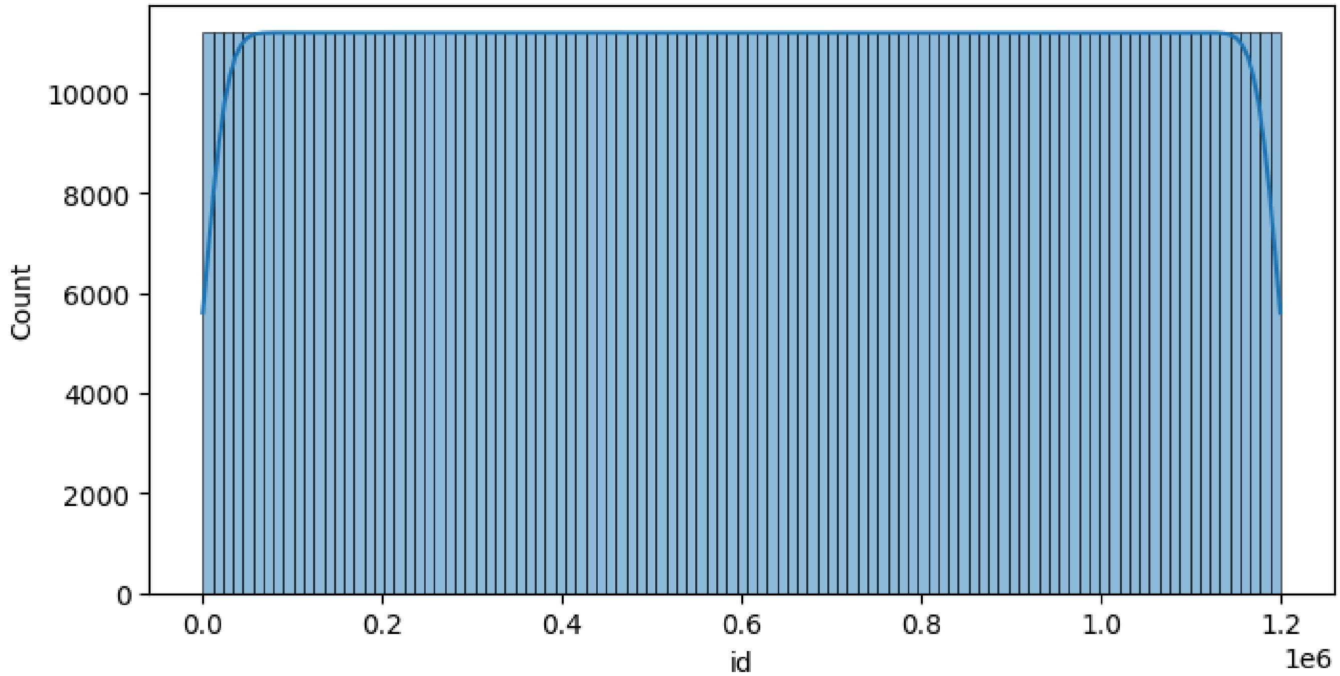
25%	2.999998e+05	3.000000e+01	8.001000e+03	1.000000e+00	1.591896e+01	0.000000	5.000000e+00	4.680000e+02	3.0000
50%	5.999995e+05	4.100000e+01	2.391100e+04	2.000000e+00	2.457865e+01	1.000000	1.000000e+01	5.950000e+02	5.0000
75%	8.999992e+05	5.300000e+01	4.463400e+04	3.000000e+00	3.452721e+01	2.000000	1.500000e+01	7.210000e+02	7.0000
max	1.199999e+06	6.400000e+01	1.499970e+05	4.000000e+00	5.897591e+01	9.000000	1.900000e+01	8.490000e+02	9.0000

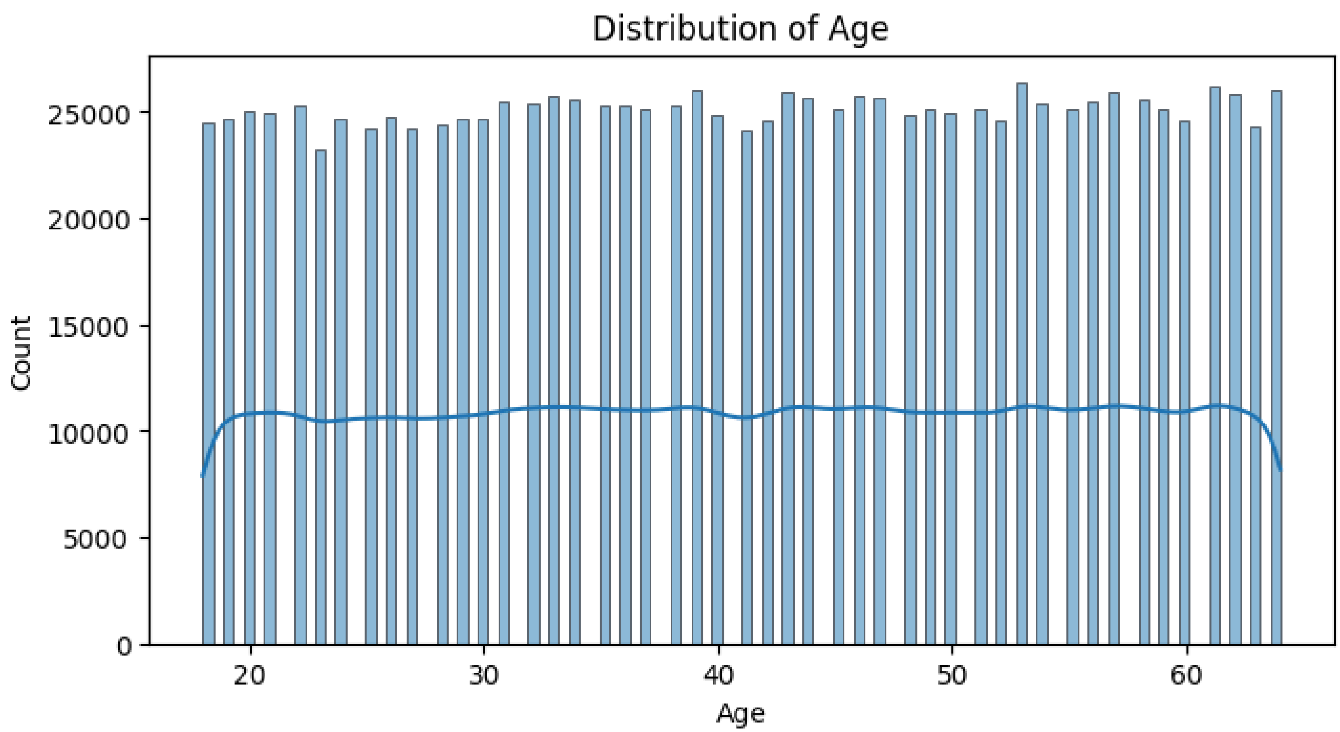
```
Missing Values:
```

	Missing Values	Percentage
id	0	0.000000
Age	18705	1.558750
Gender	0	0.000000
Annual Income	44949	3.745750
Marital Status	18529	1.544083
Number of Dependents	109672	9.139333
Education Level	0	0.000000
Occupation	358075	29.839583
Health Score	74076	6.173000
Location	0	0.000000
Policy Type	0	0.000000
Previous Claims	364029	30.335750
Vehicle Age	6	0.000500
Credit Score	137882	11.490167
Insurance Duration	1	0.000083
Policy Start Date	0	0.000000
Customer Feedback	77824	6.485333
Smoking Status	0	0.000000
Exercise Frequency	0	0.000000

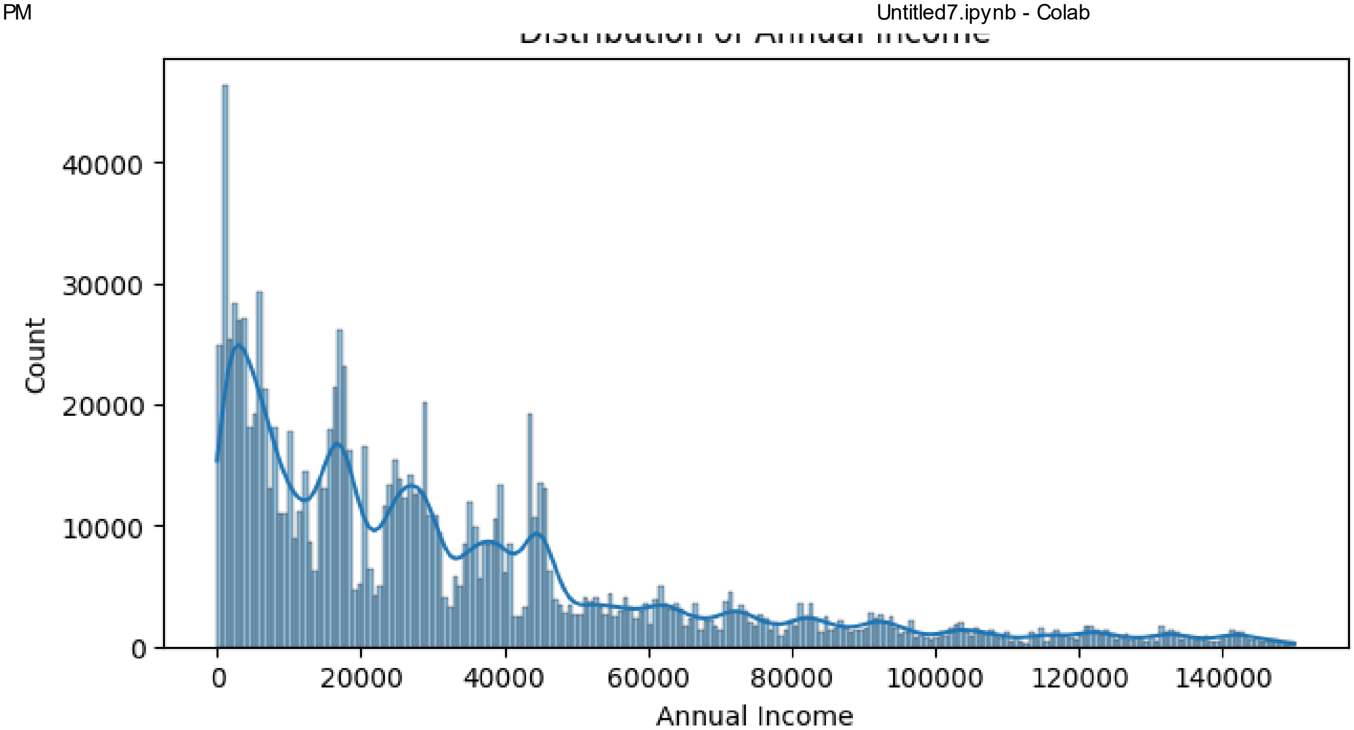
Property Type	0	0.000000
Premium Amount	0	0.000000

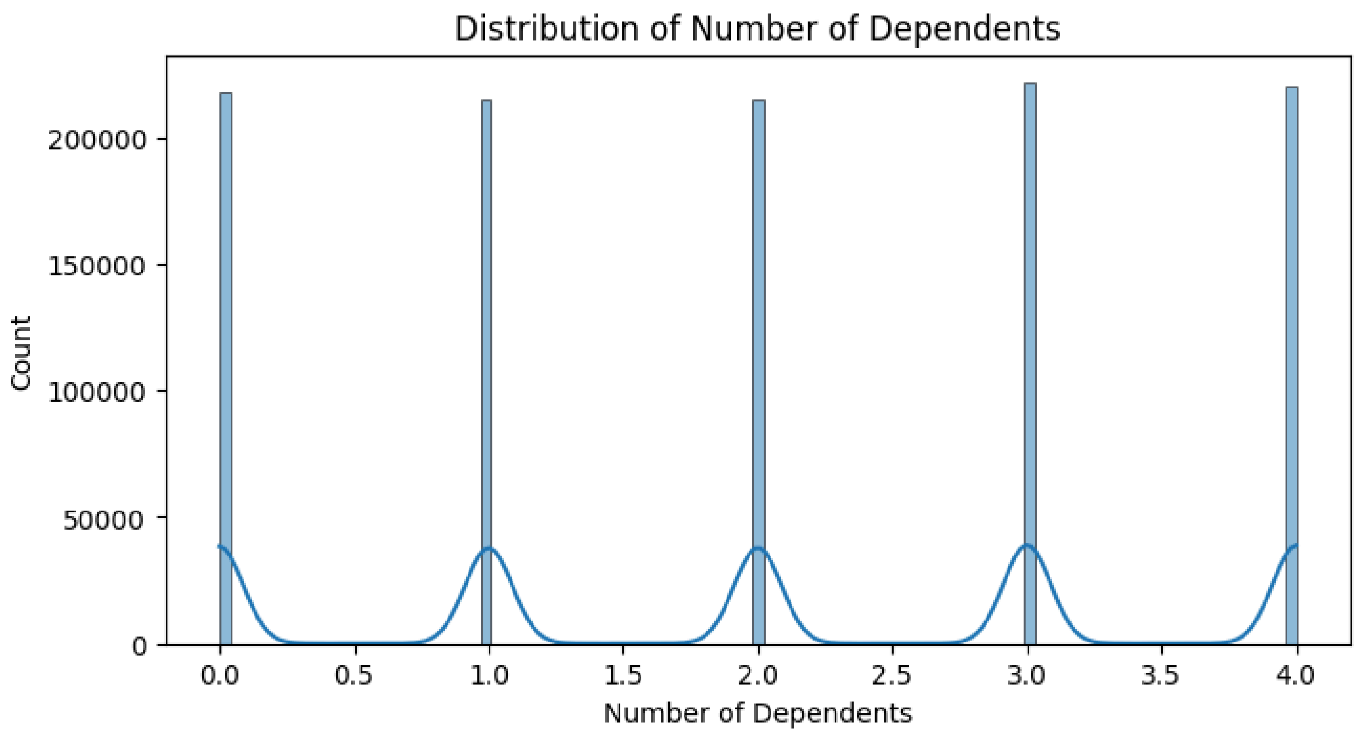
Distribution of id

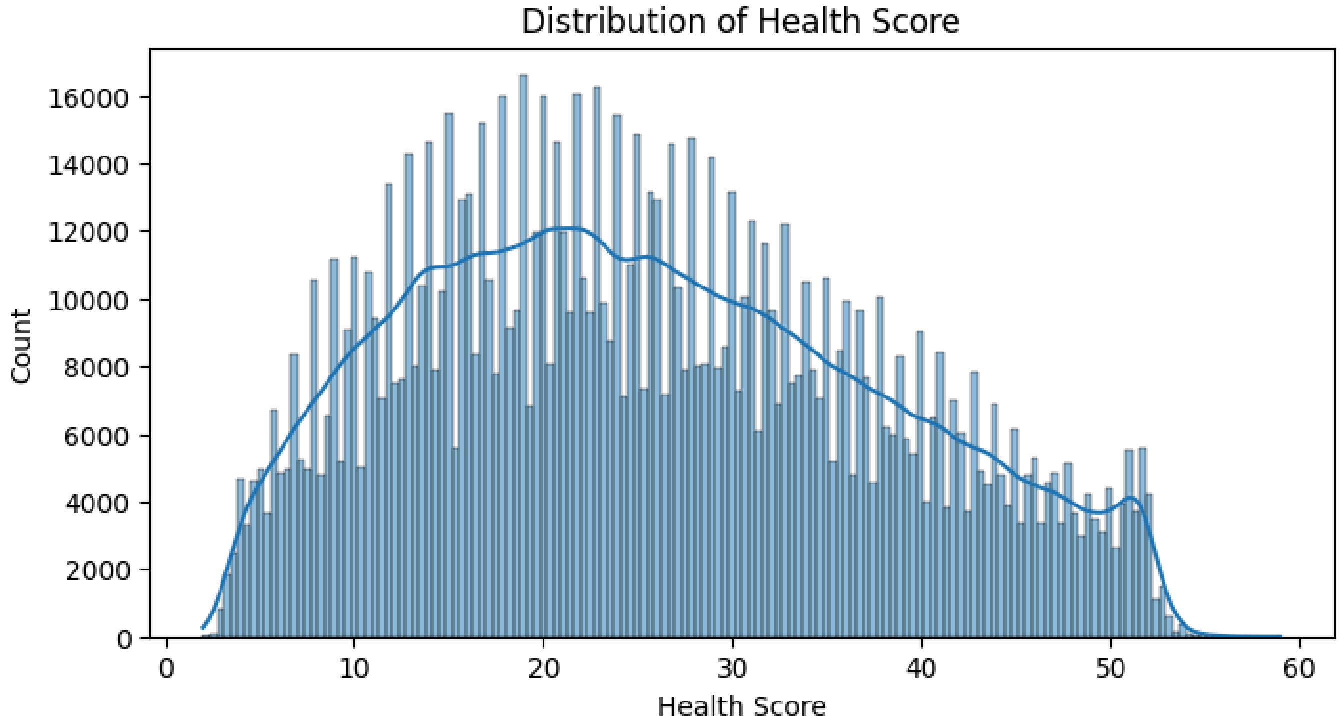


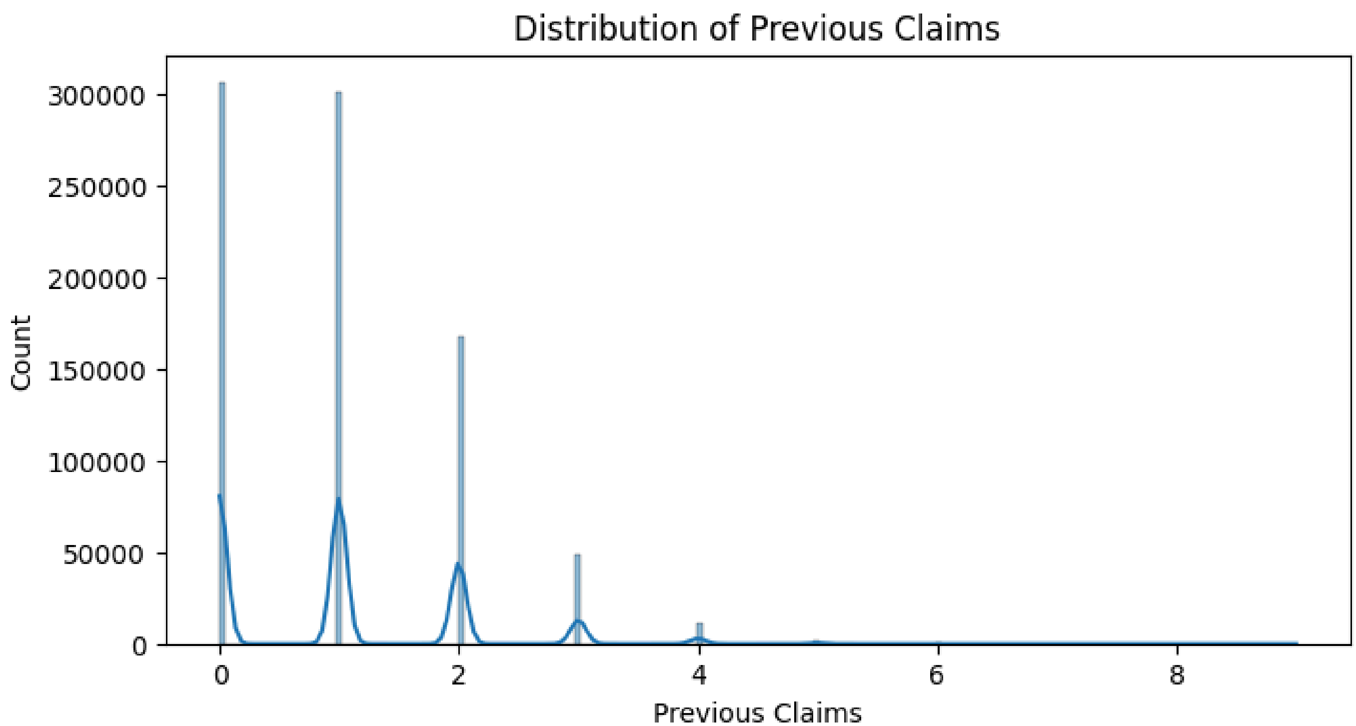


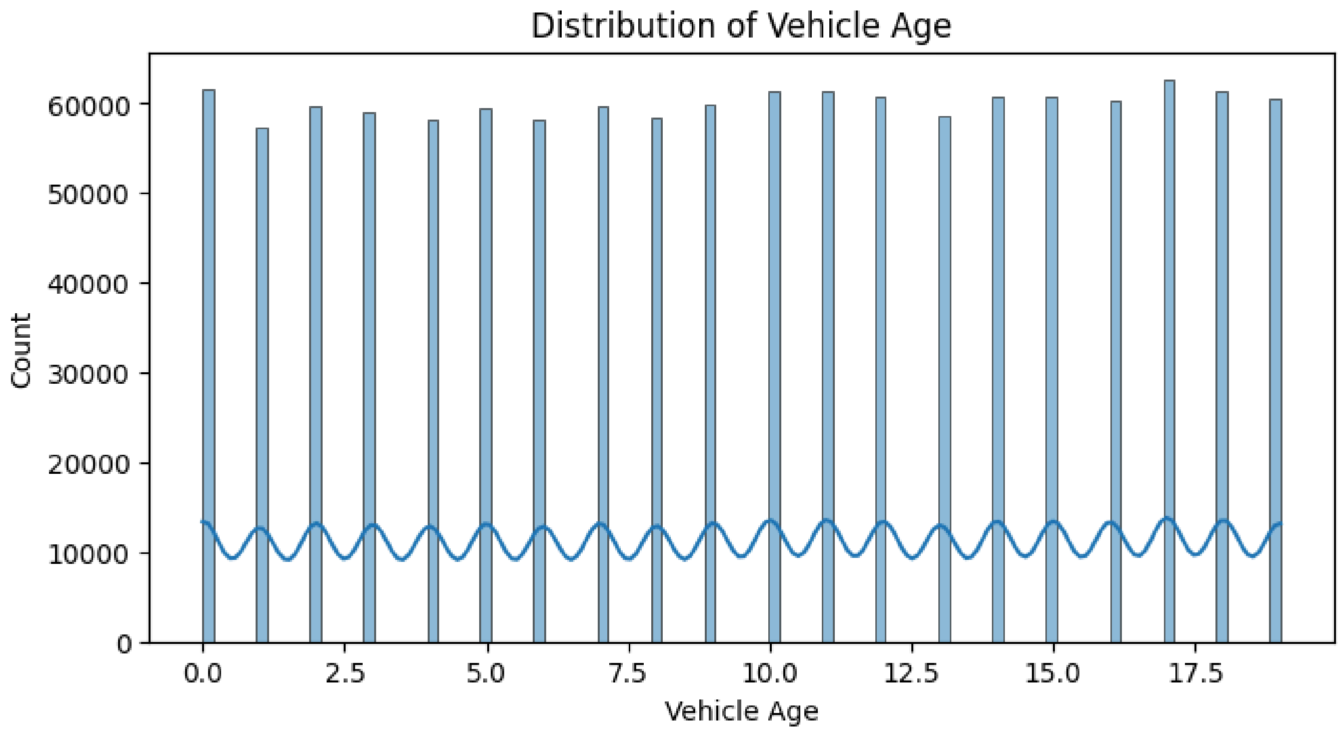
Distribution of Annual Income

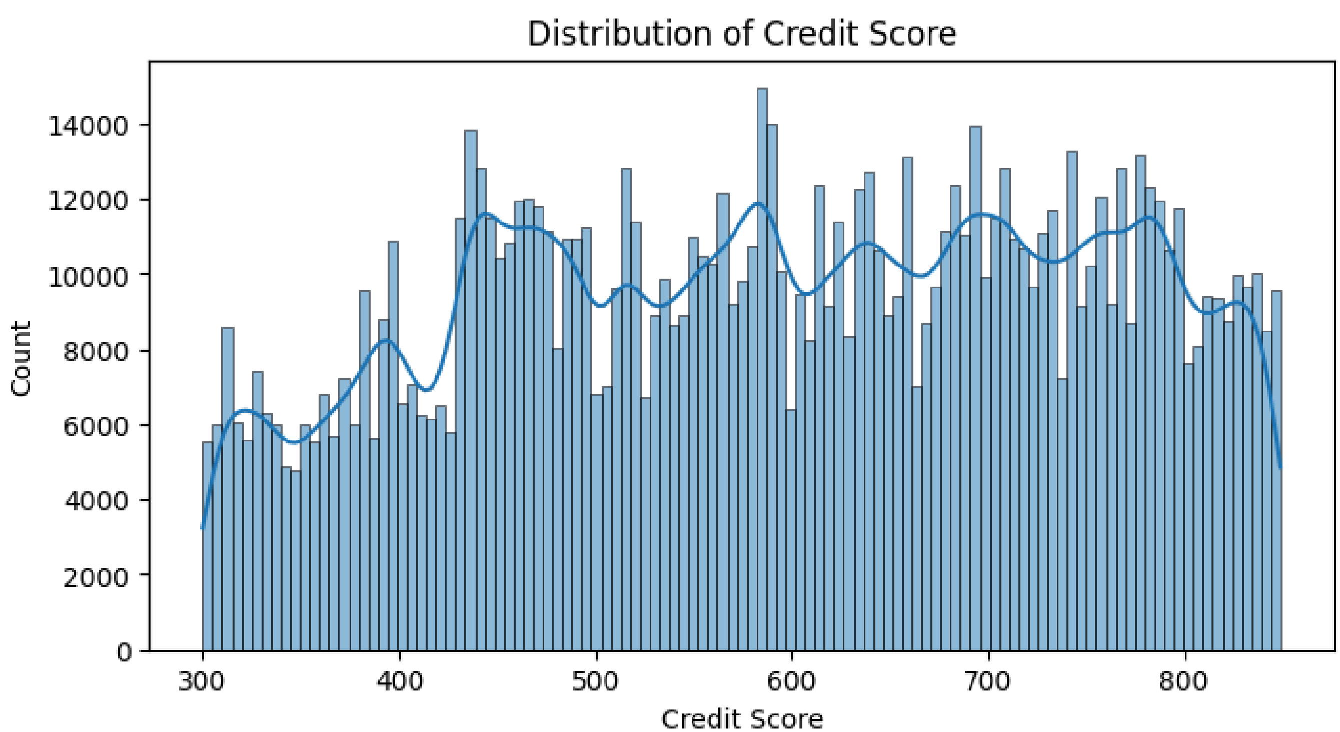

Count
Annual Income
0
10000
20000
30000
40000
20000
40000
60000
80000
100000
120000
140000

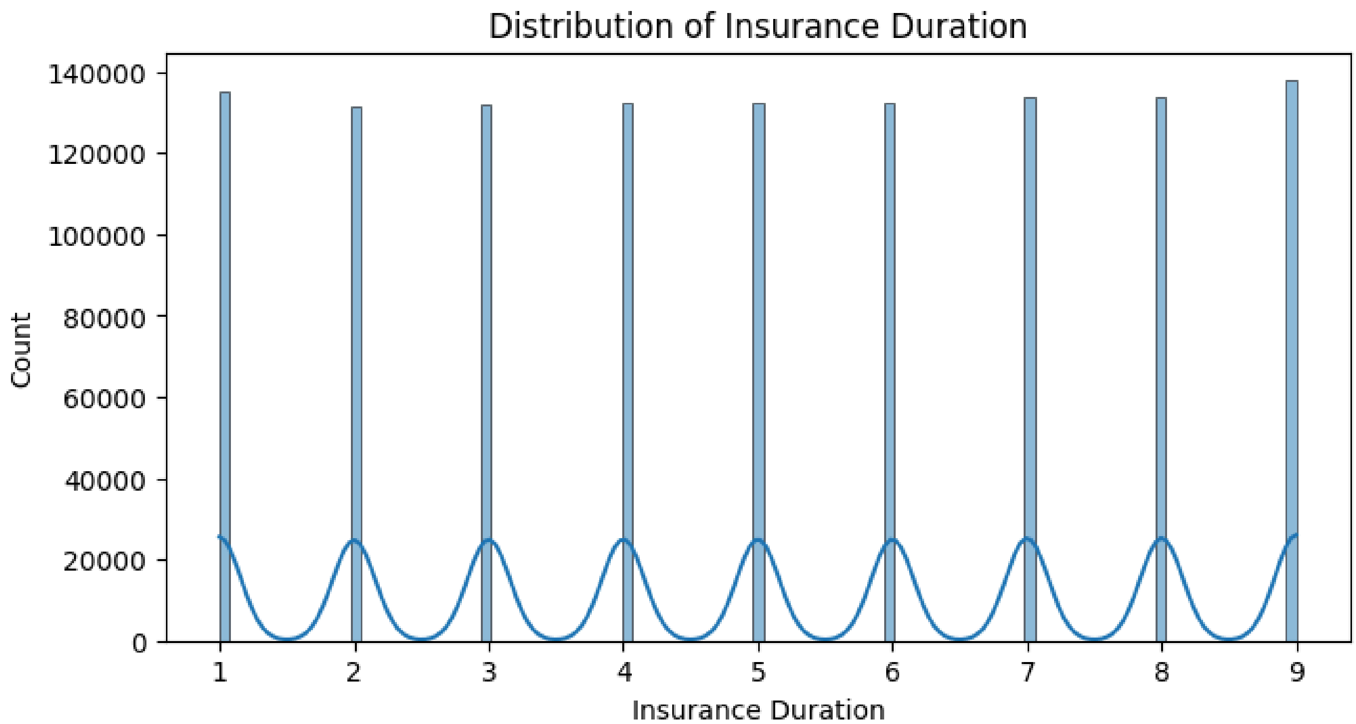

Distribution of Number of Dependents
Count
Number of Dependents
0
50000
100000
150000
200000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

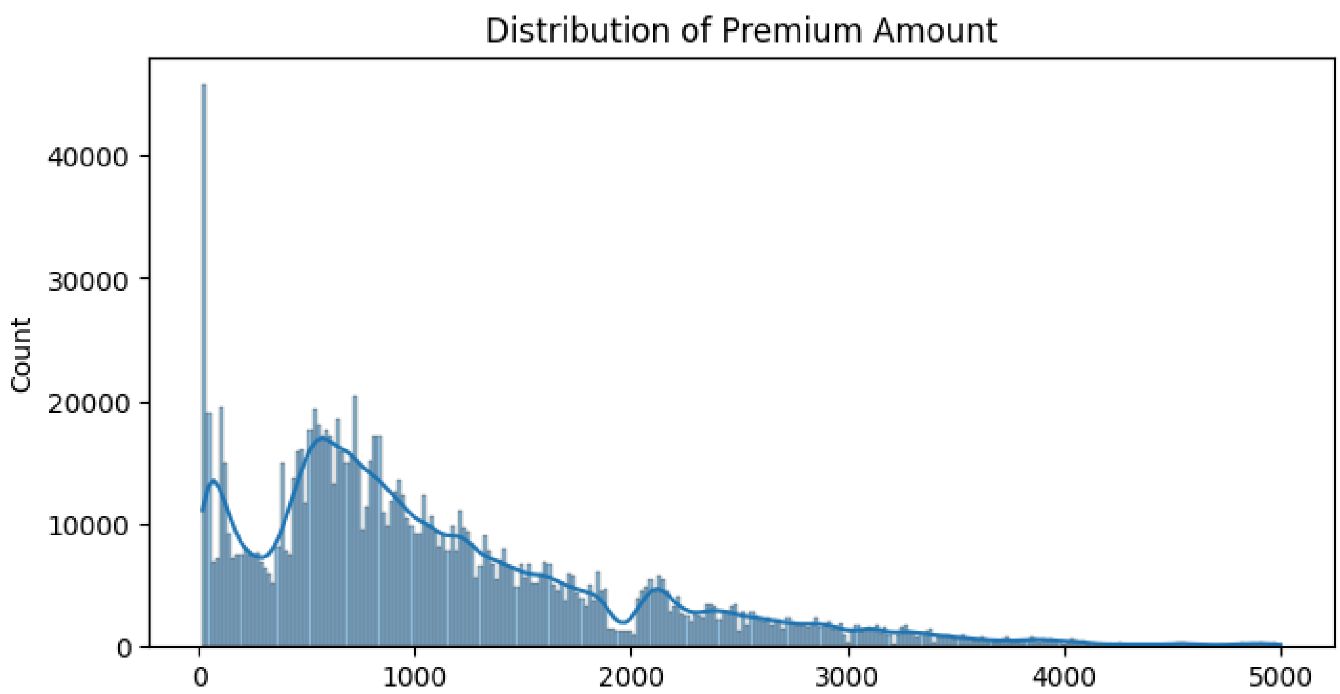

Distribution of Health Score
Count
16000
14000
12000
10000
8000
6000
4000
2000
0
0
10
20
30
40
50
60
Health Score


Distribution of Previous Claims
Count
300000
250000
200000
150000
100000
50000
0
0
2
4
6
8
Previous Claims


Distribution of Vehicle Age
Count
Vehicle Age
0
10000
20000
30000
40000
50000
60000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Distribution of Credit Score
Count
Credit Score
0
2000
4000
6000
8000
10000
12000
14000
300
400
500
600
700
800


Distribution of Insurance Duration
Count
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
8
9
Insurance Duration


Distribution of Premium Amount
Count
40000
30000
20000
10000
0
0
1000
2000
3000
4000
5000

Premium Amount

```
Value counts for Gender:
```

Gender	count
Male	602571
Female	597429

dtype: int64

```
Value counts for Marital Status:
```

Marital Status	count
Single	395391
Married	394316
Divorced	391764

dtype: int64

```
Value counts for Education Level:
```

Education Level	count
Master's	303818
PhD	303507
Bachelor's	303234
High School	289441

dtype: int64

```
Value counts for Occupation:
```

count

Occupation	
Employed	282750
Self-Employed	282645
Unemployed	276530

dtype: int64

```
Value counts for Location:
```

Location	count
Suburban	401542
Rural	400947
Urban	397511

dtype: int64

```
Value counts for Policy Type:
```

Policy Type	count
Premium	401846
Comprehensive	399600
Basic	398554

dtype: int64

```
Value counts for Policy Start Date:
```

Policy Start Date	count
2020-02-08 15:21:39.134960	142
2023-08-13 15:21:39.155231	137

2022-02-02 15:21:39.134960	137
2022-08-30 15:21:39.134960	134
2023-11-02 15:21:39.134960	118
...	...
2021-06-07 15:21:39.104139	1
2024-07-19 15:21:39.233998	1
2019-12-14 15:21:39.110557	1
2020-07-23 15:21:39.217387	1
2020-10-19 15:21:39.118178	1

167381 rows × 1 columns

dtype: int64

```
Value counts for Customer Feedback:
```

Customer Feedback	count
Average	377905
Poor	375518
Good	368753

dtype: int64

```
Value counts for Smoking Status:
```

Smoking Status	count
Yes	601873
No	598127

dtype: int64

```
Value counts for Exercise Frequency:
```

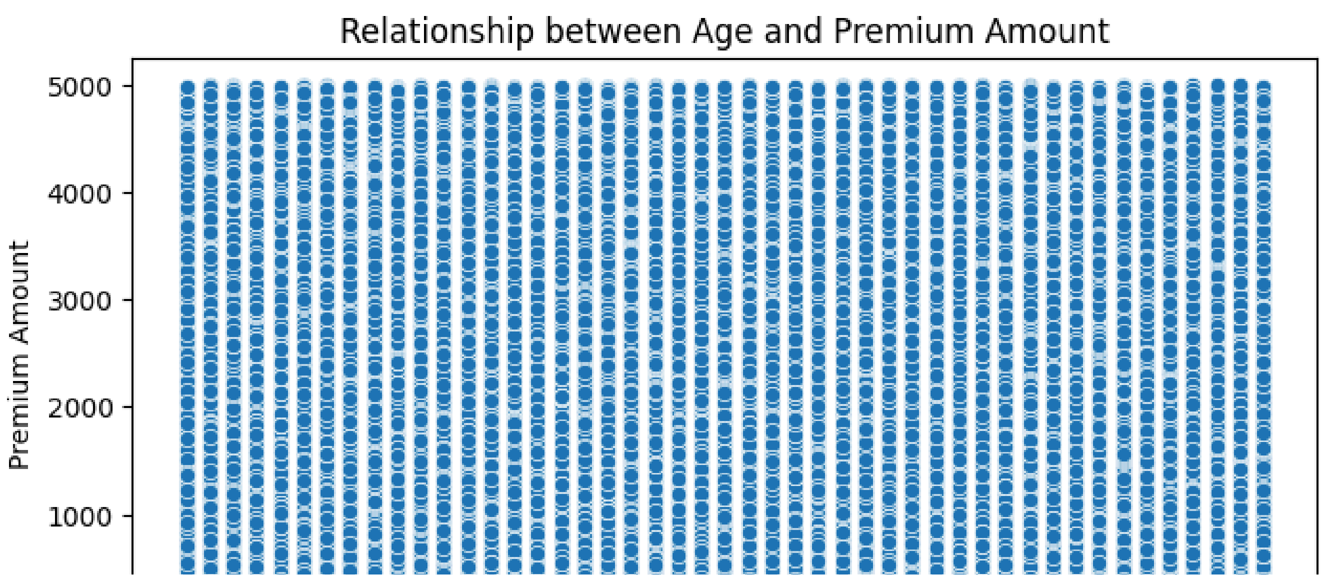
Exercise Frequency	count
Weekly	306179
Monthly	299830
Rarely	299420
Daily	294571

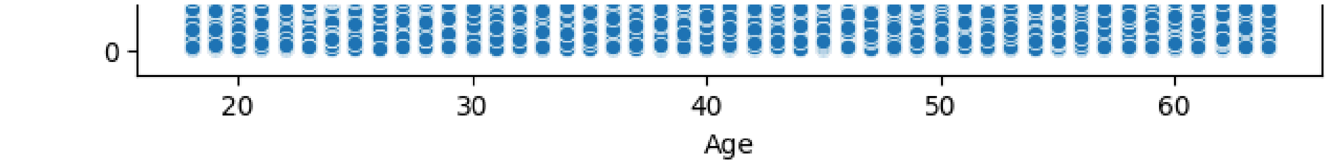
dtype: int64

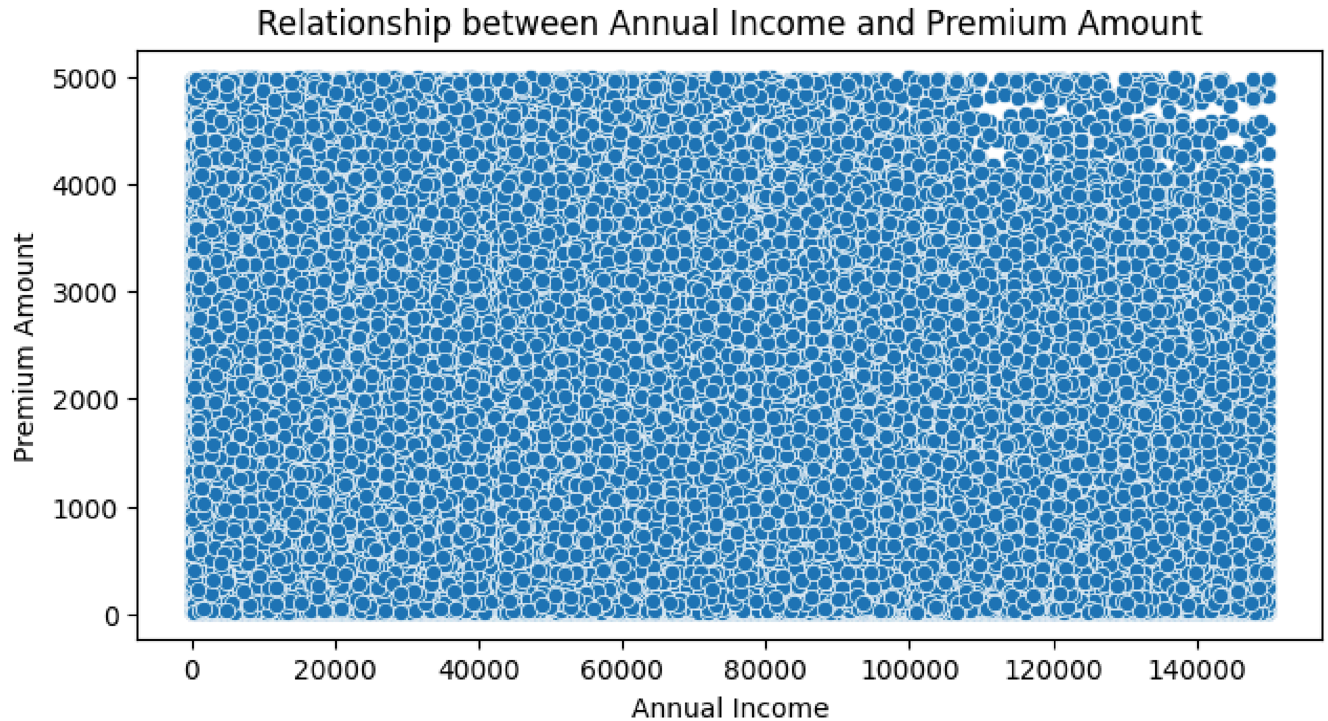
```
Value counts for Property Type:
```

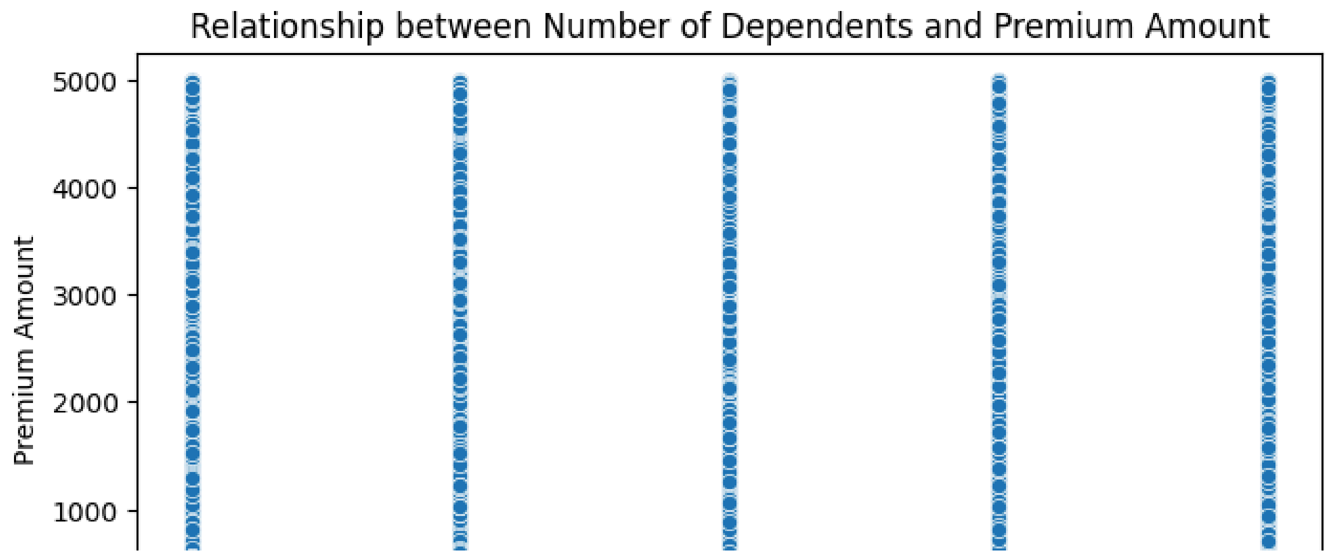
Property Type	count
House	400349
Apartment	399978
Condo	399673

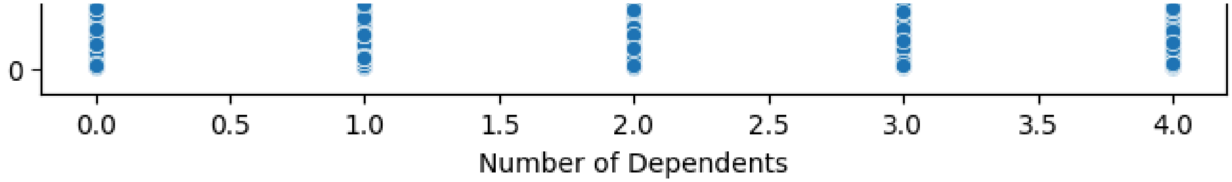
dtype: int64



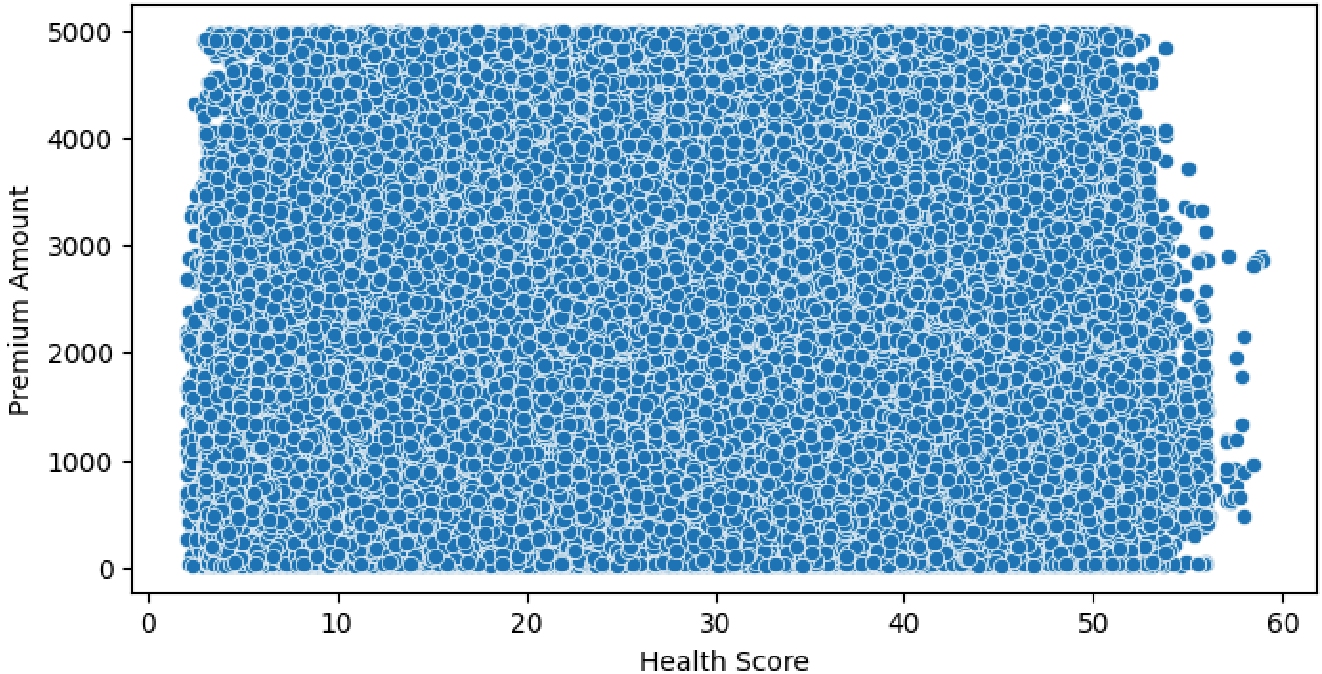

0
20
30
40
50
60
Age


Relationship between Annual Income and Premium Amount
Premium Amount
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000
100000
120000
140000
Annual Income

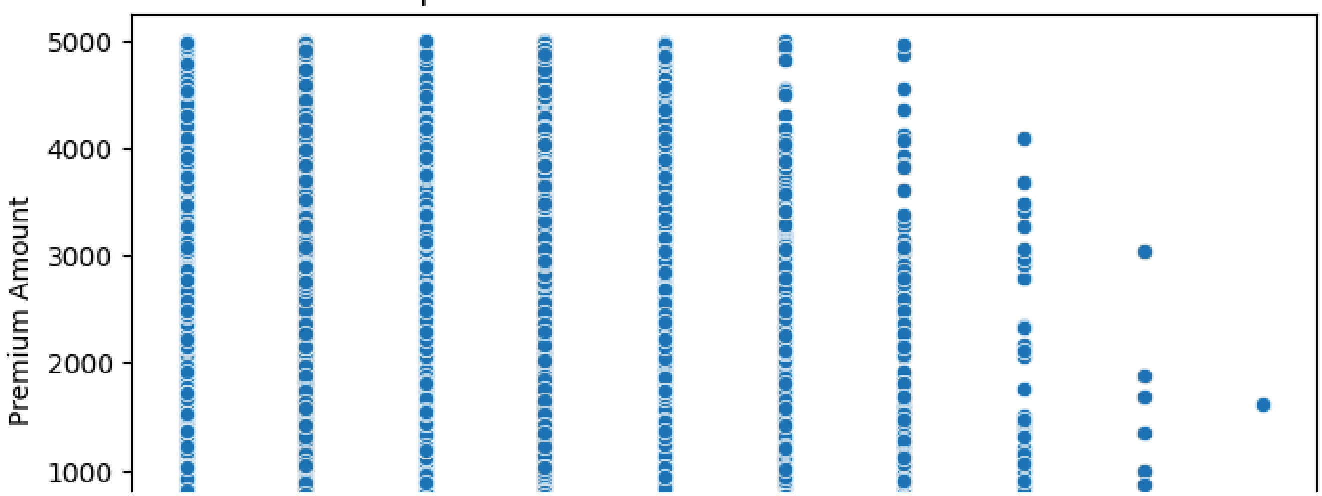

Relationship between Number of Dependents and Premium Amount
Premium Amount
5000
4000
3000
2000
1000

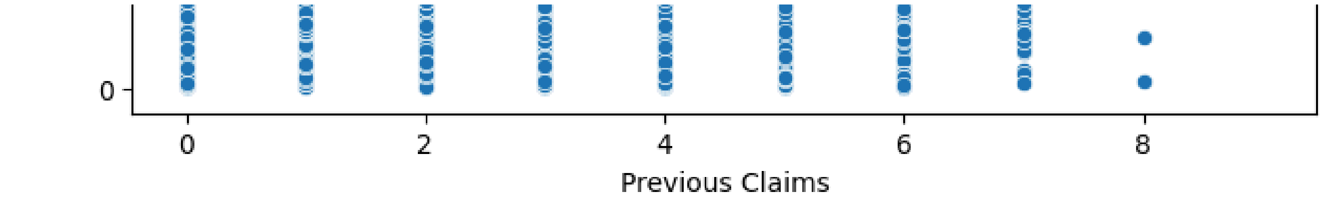


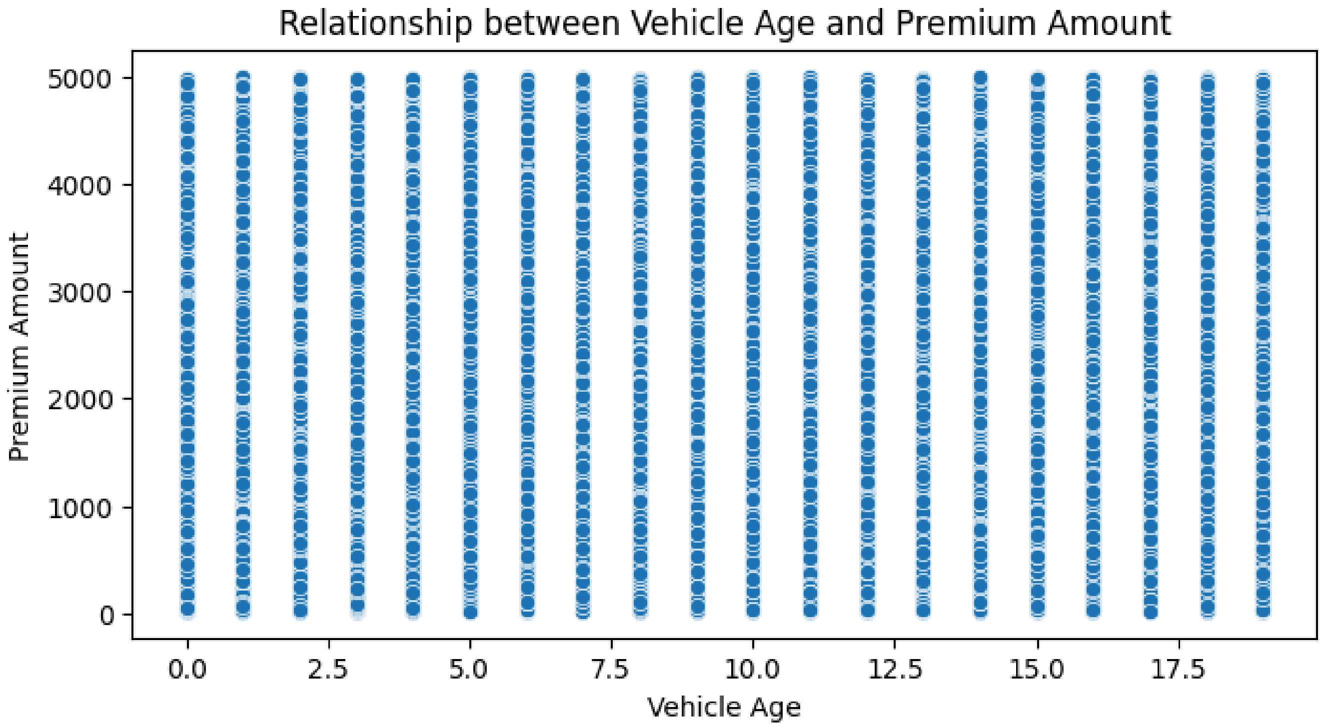
Relationship between Health Score and Premium Amount

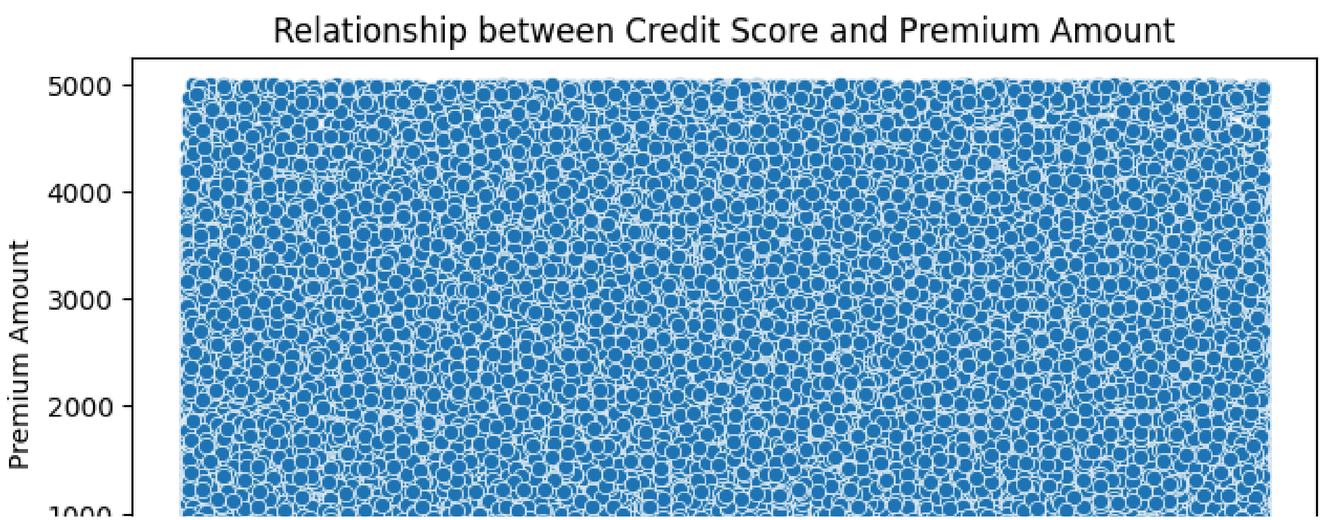


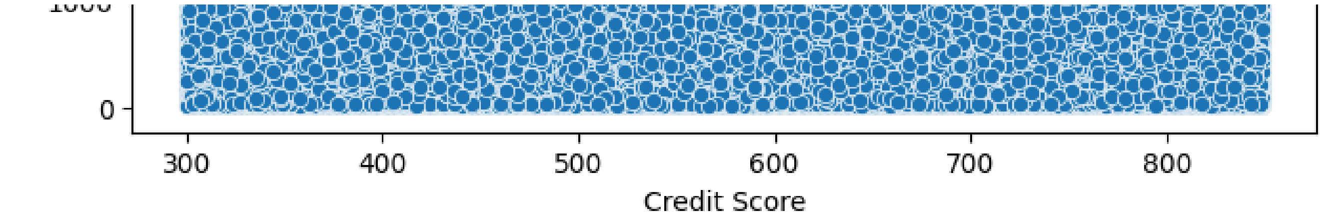
Relationship between Previous Claims and Premium Amount

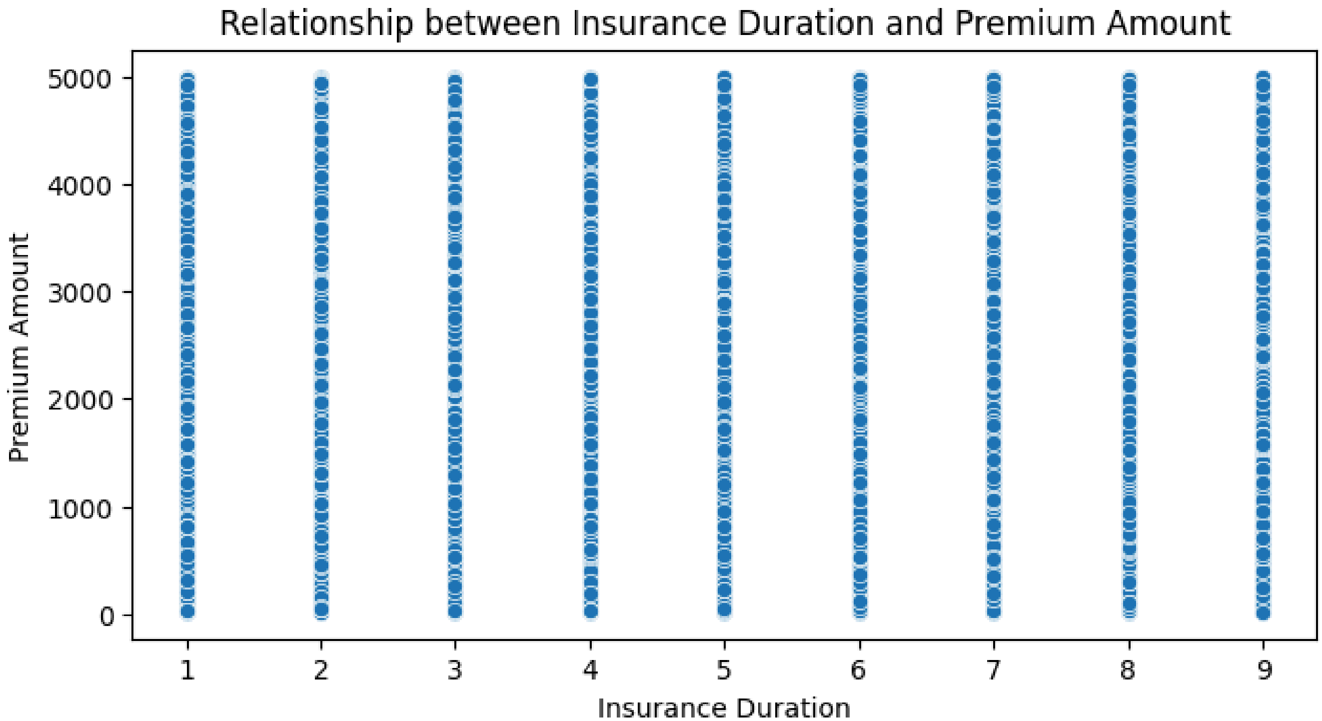


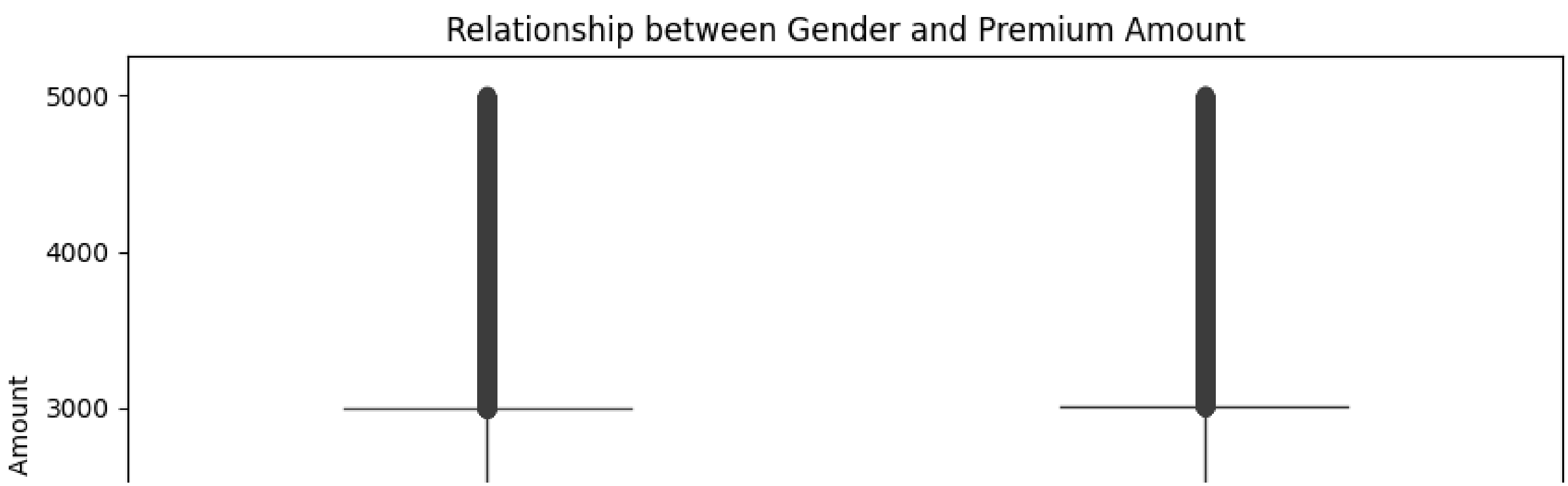

0
0
2
4
6
8
Previous Claims

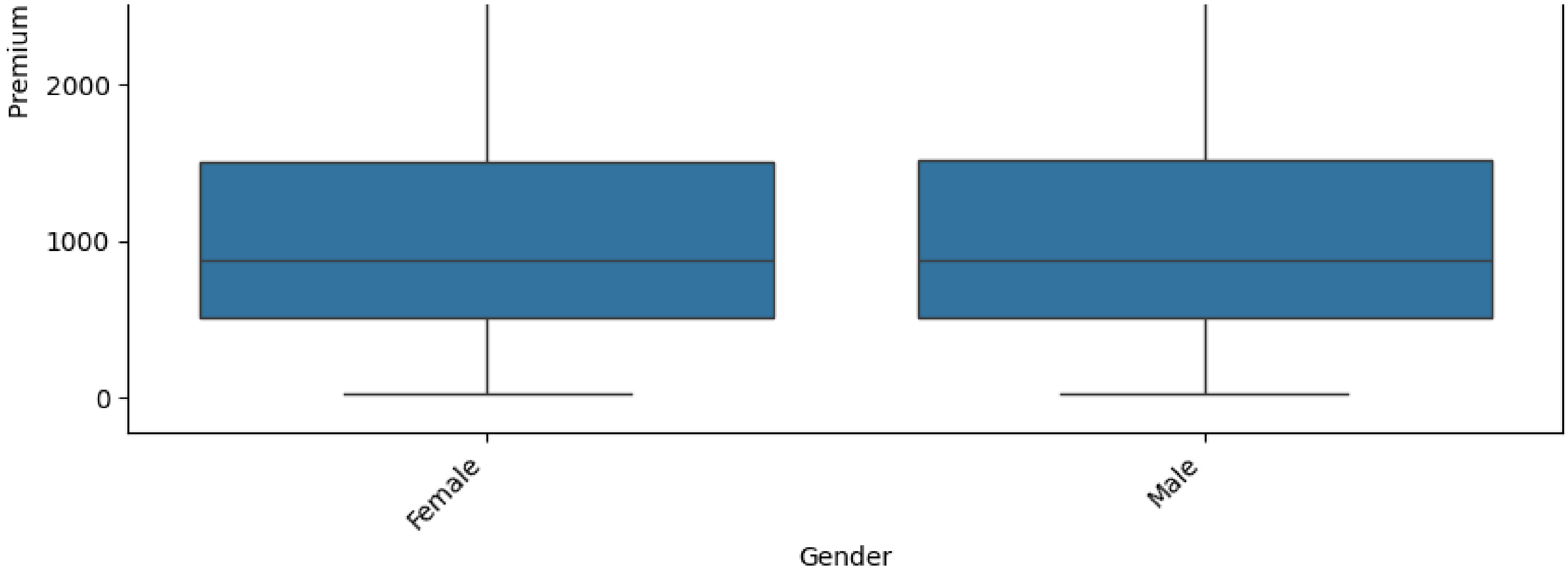

Relationship between Vehicle Age and Premium Amount
Premium Amount
5000
4000
3000
2000
1000
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Vehicle Age

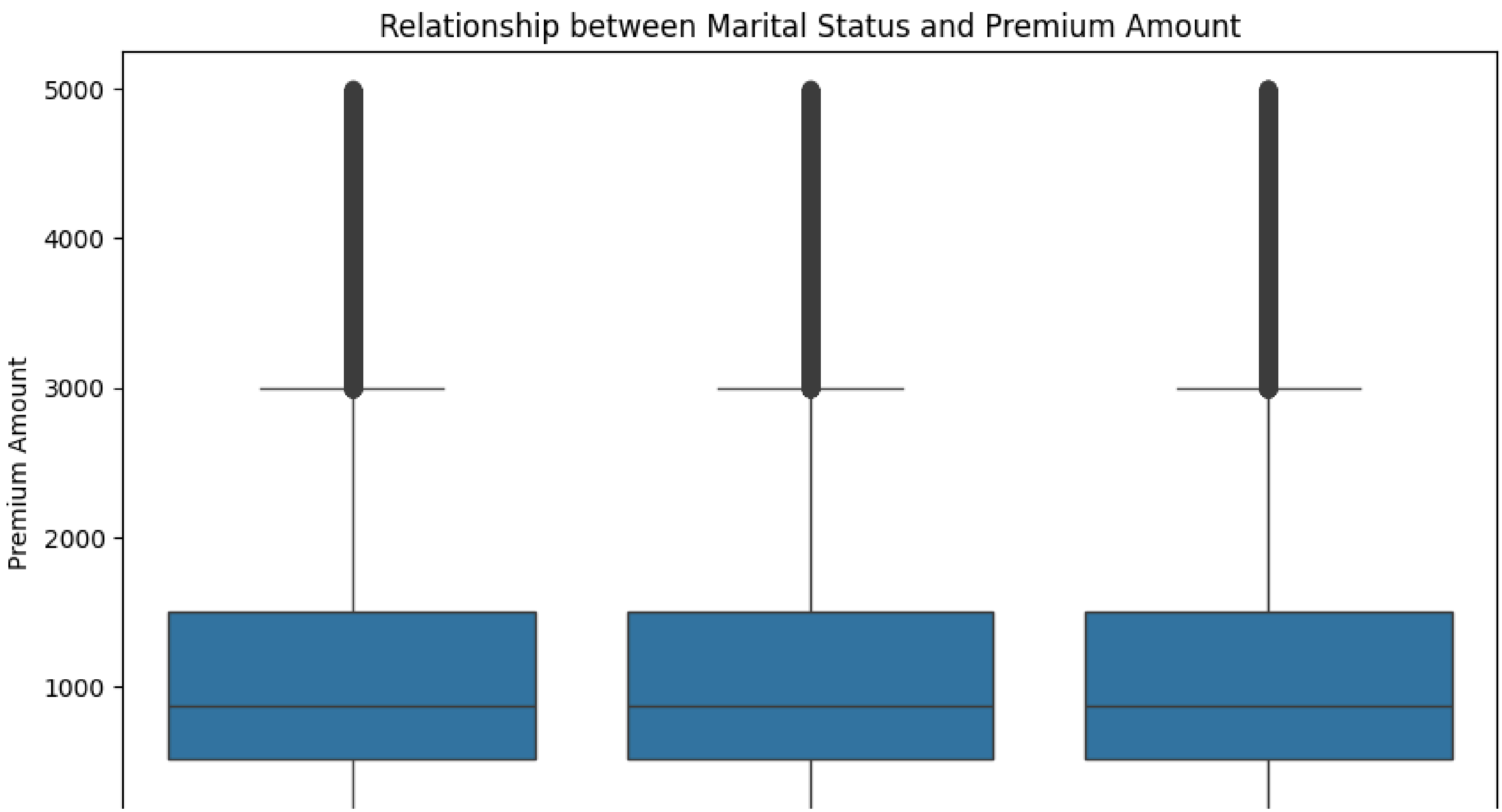

Relationship between Credit Score and Premium Amount
Premium Amount
5000
4000
3000
2000

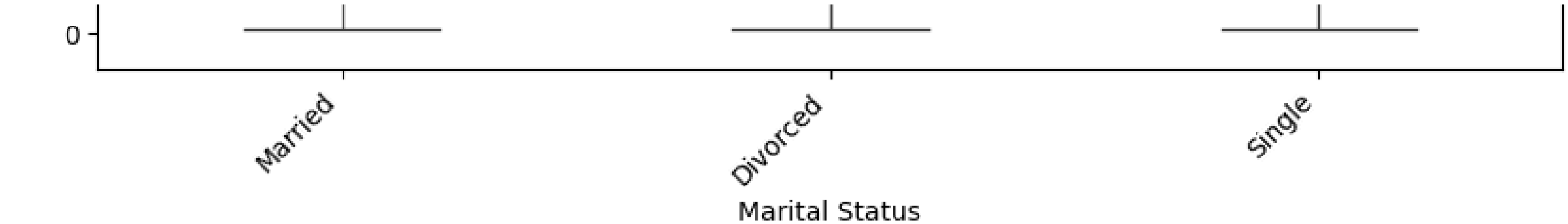

1000
0
300
400
500
600
700
800
Credit Score


Relationship between Insurance Duration and Premium Amount
5000
4000
3000
2000
1000
0
Premium Amount
1
2
3
4
5
6
7
8
9
Insurance Duration

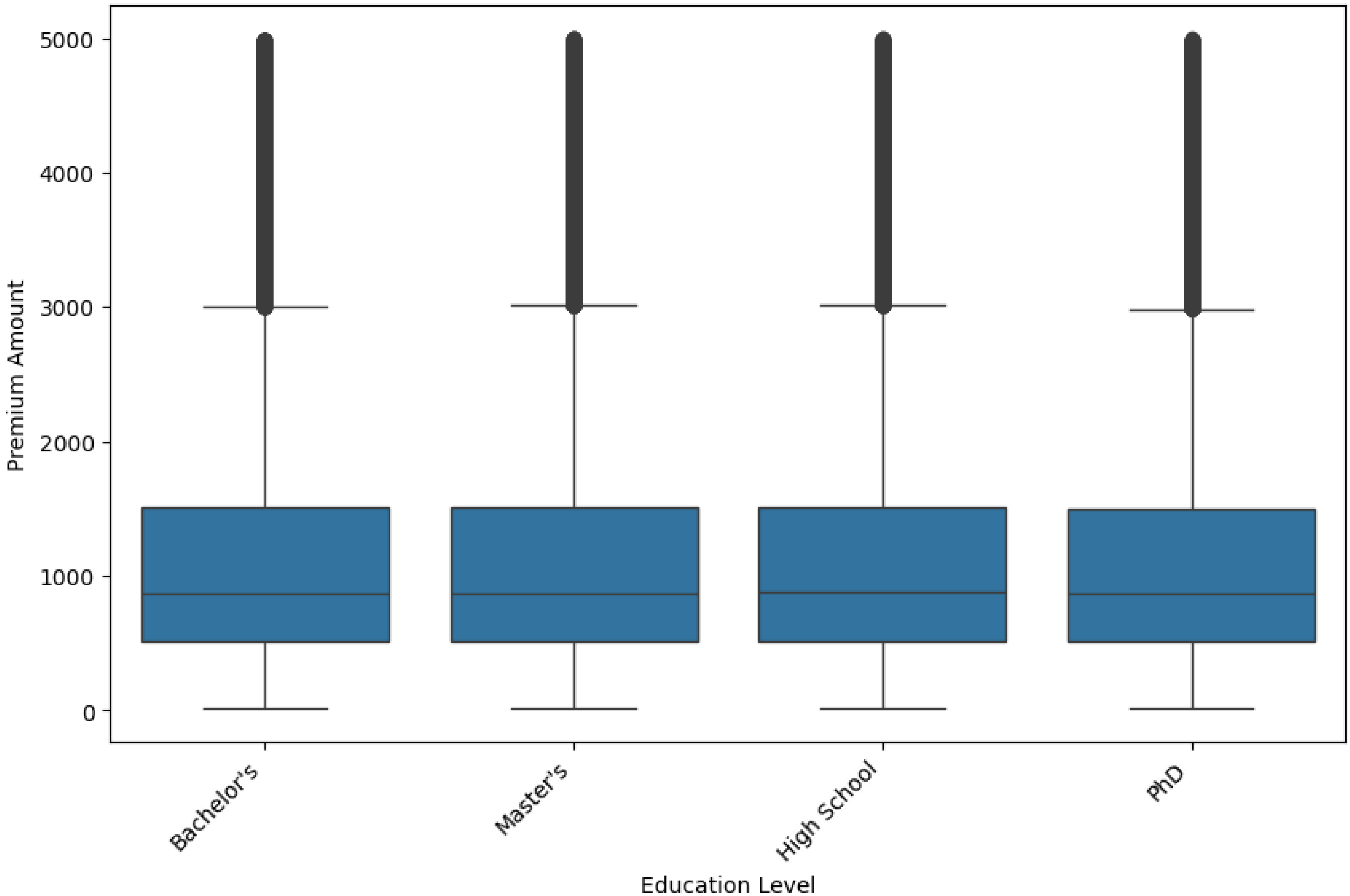

Relationship between Gender and Premium Amount
5000
4000
3000
Amount


Premium
2000
1000
0
Female
Male
Gender

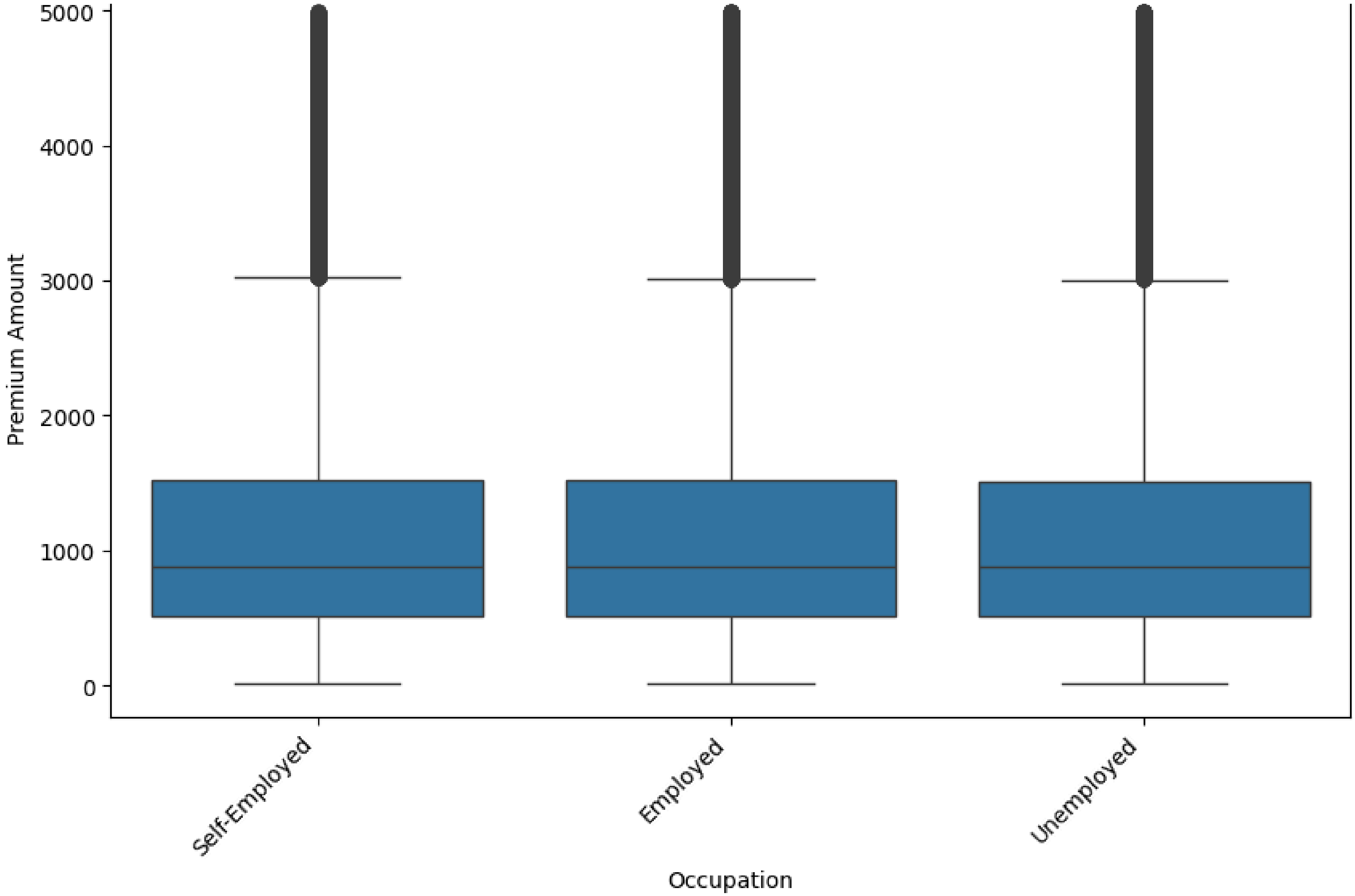

Relationship between Marital Status and Premium Amount
5000
4000
3000
2000
1000
Premium Amount



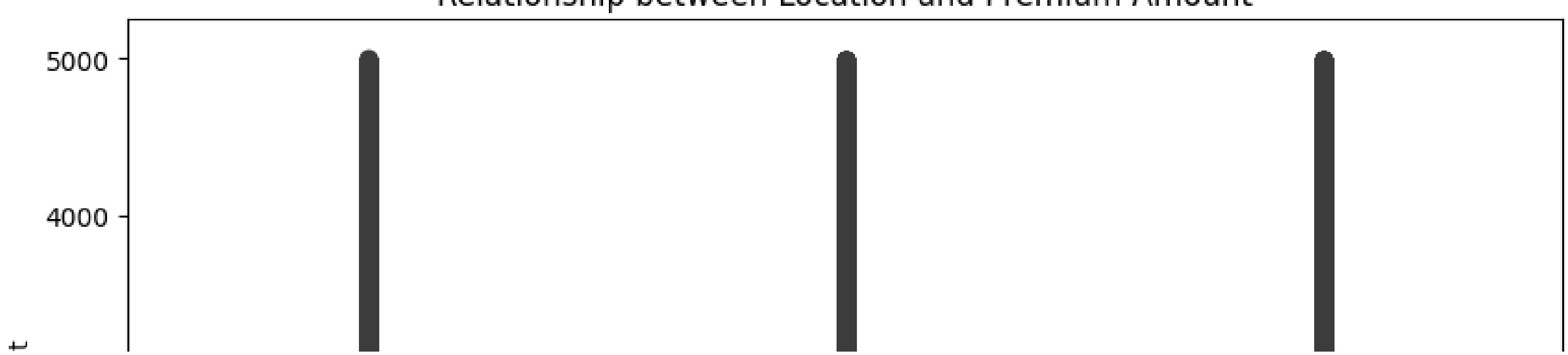
Relationship between Education Level and Premium Amount

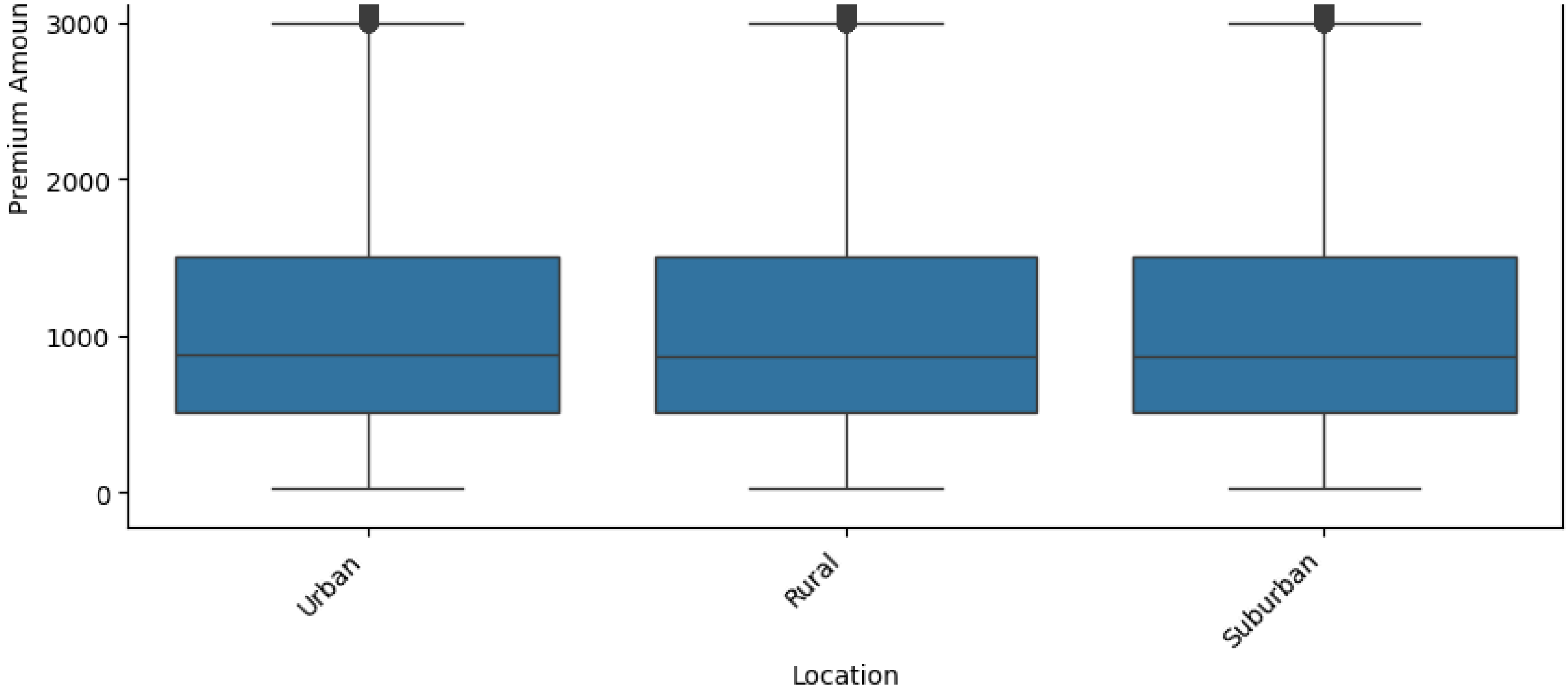


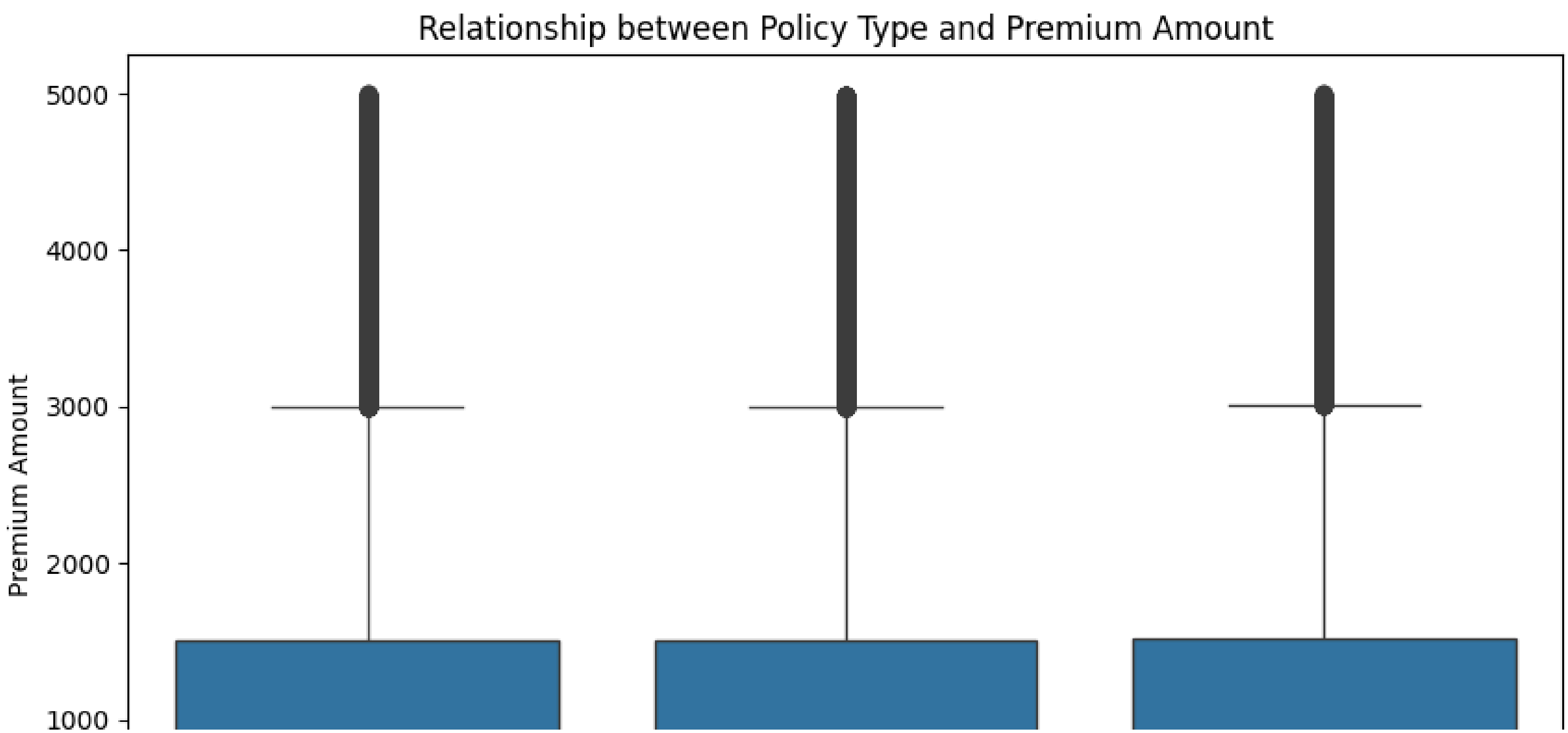
Relationship between Occupation and Premium Amount

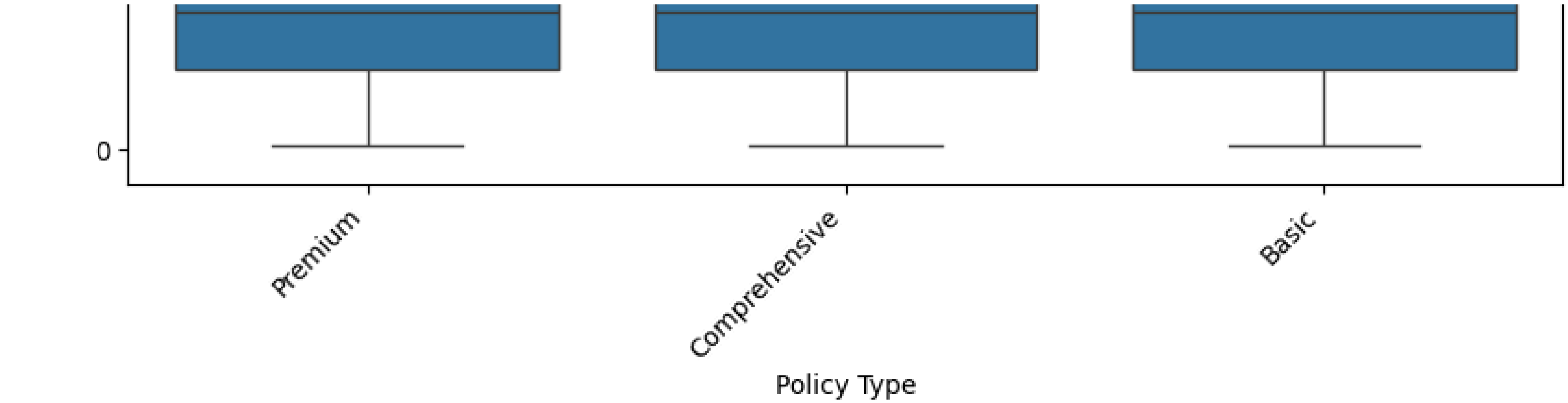


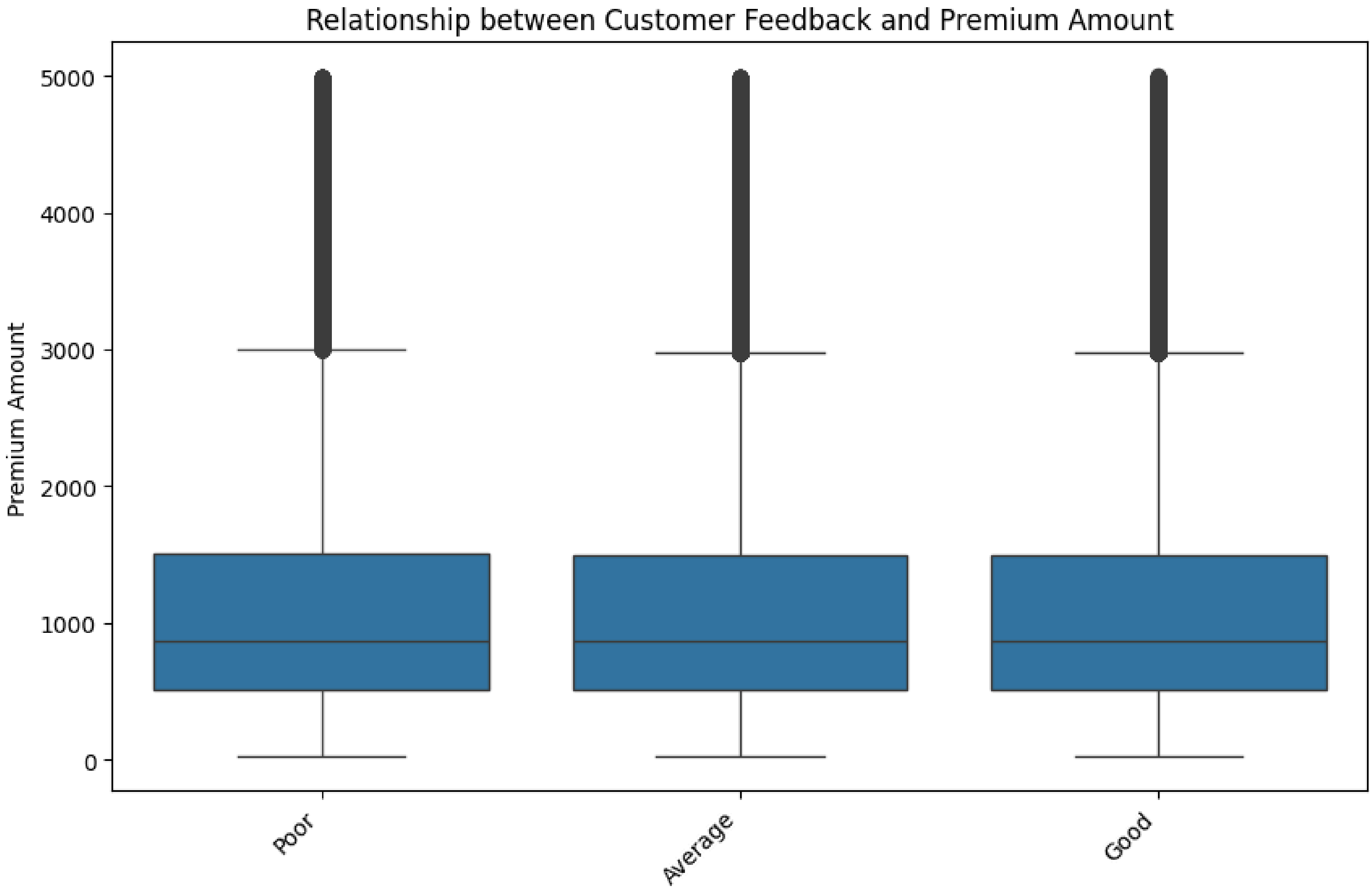
Relationship between Location and Premium Amount



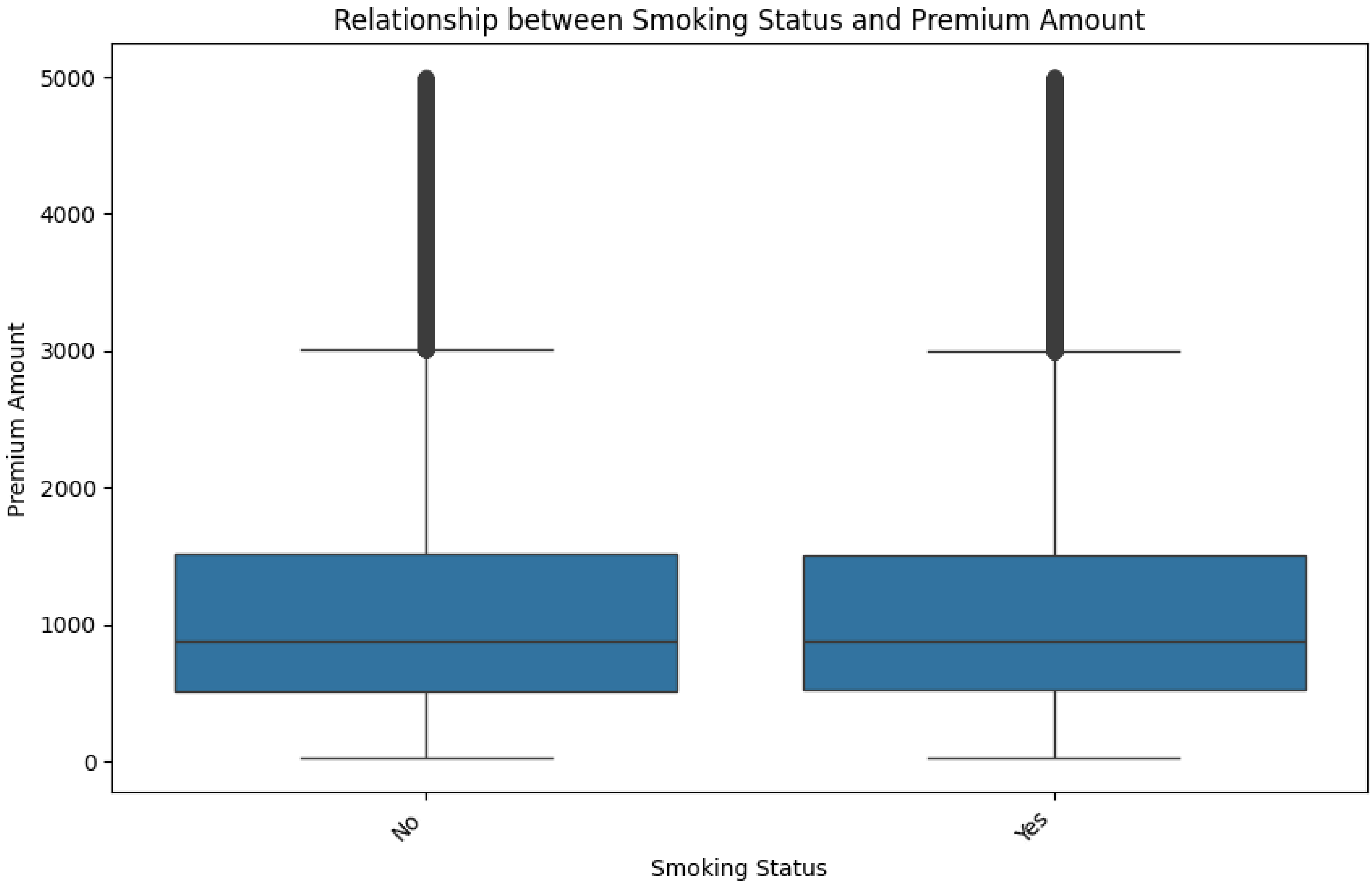

Premium Amoun
3000
2000
1000
0
Urban
Rural
Suburban
Location

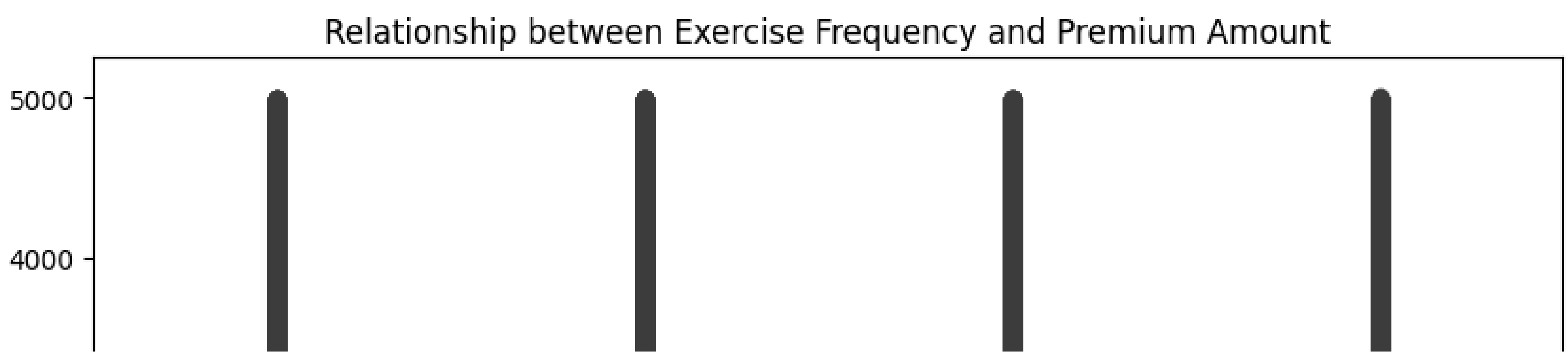

Relationship between Policy Type and Premium Amount
Premium Amount
5000
4000
3000
2000
1000

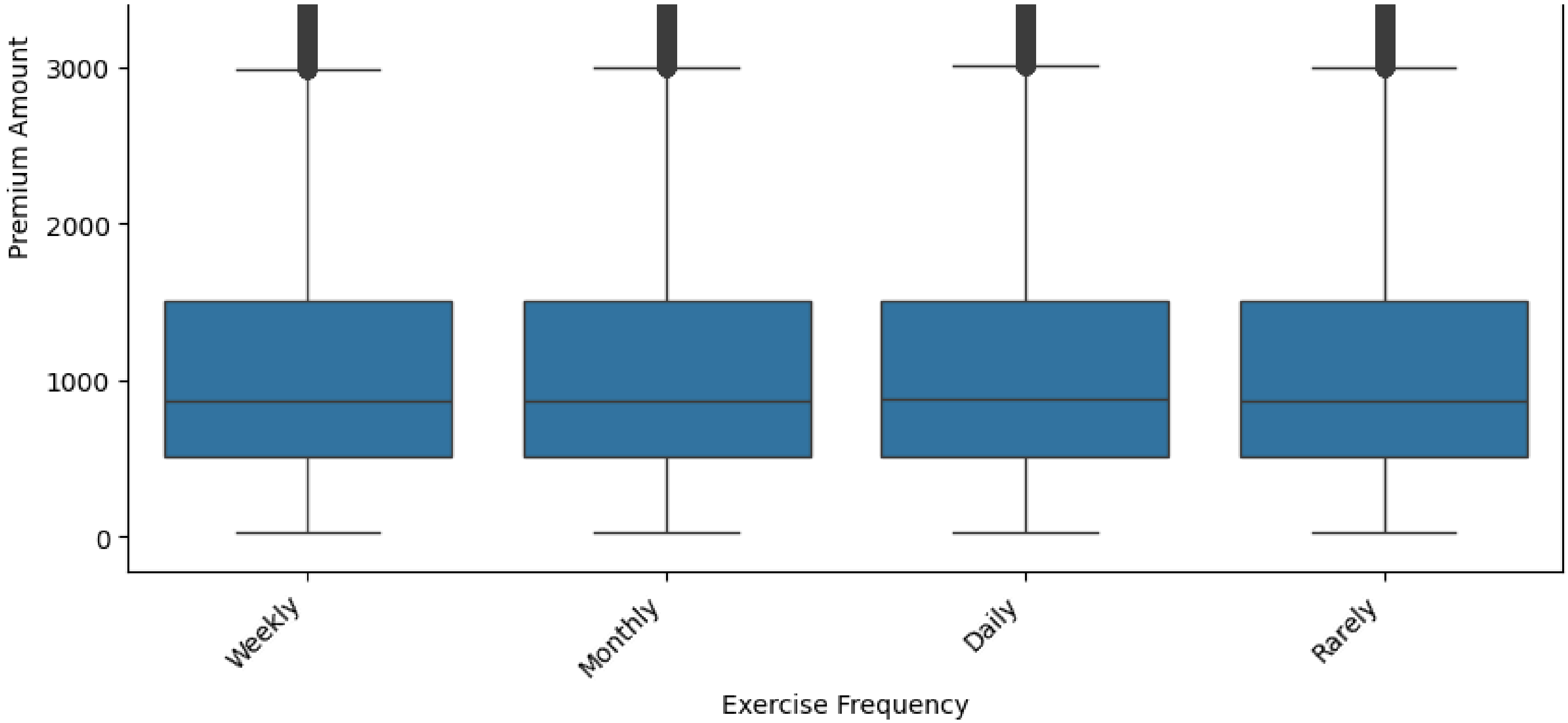

0
Premium
Comprehensive
Basic
Policy Type


Relationship between Customer Feedback and Premium Amount
Premium Amount
5000
4000
3000
2000
1000
0
Poor
Average
Good

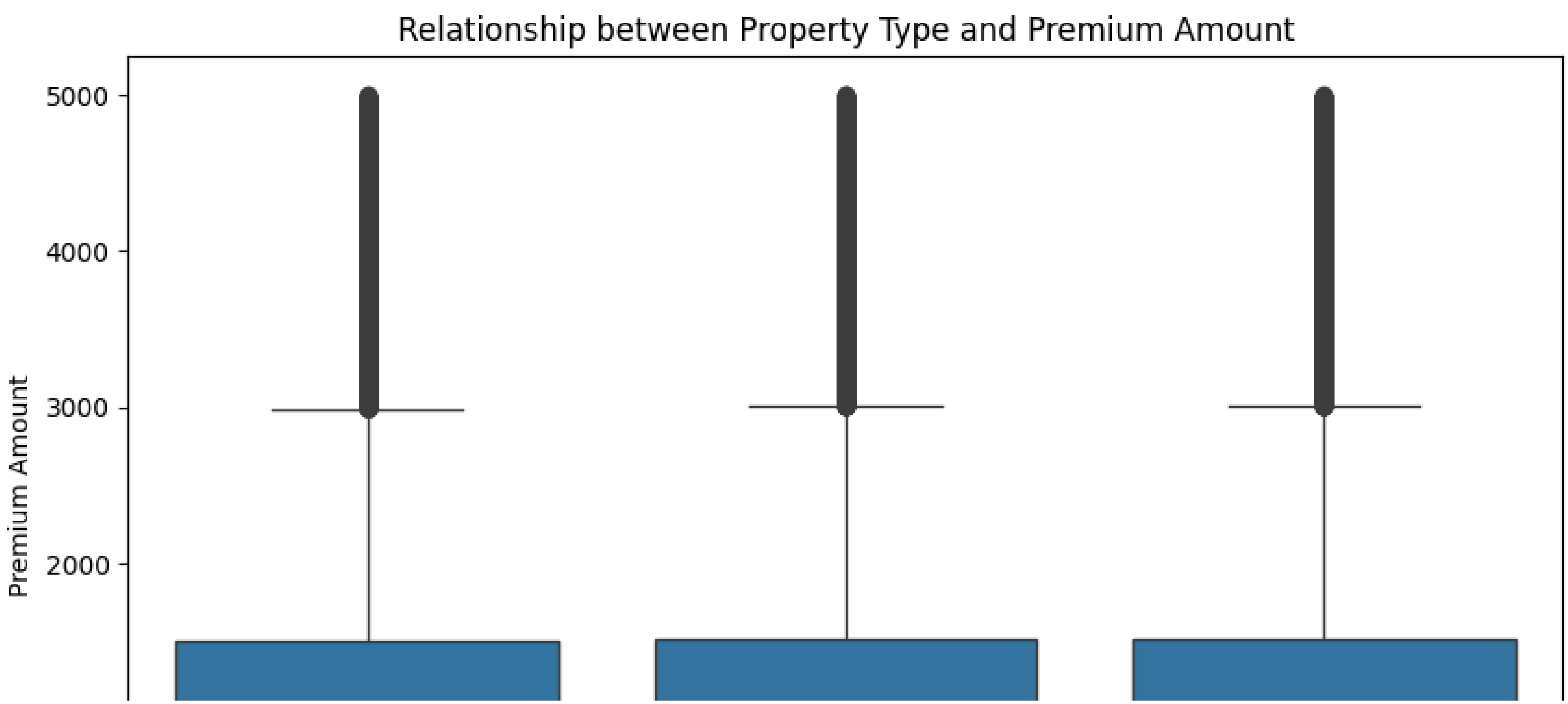
Customer Feedback



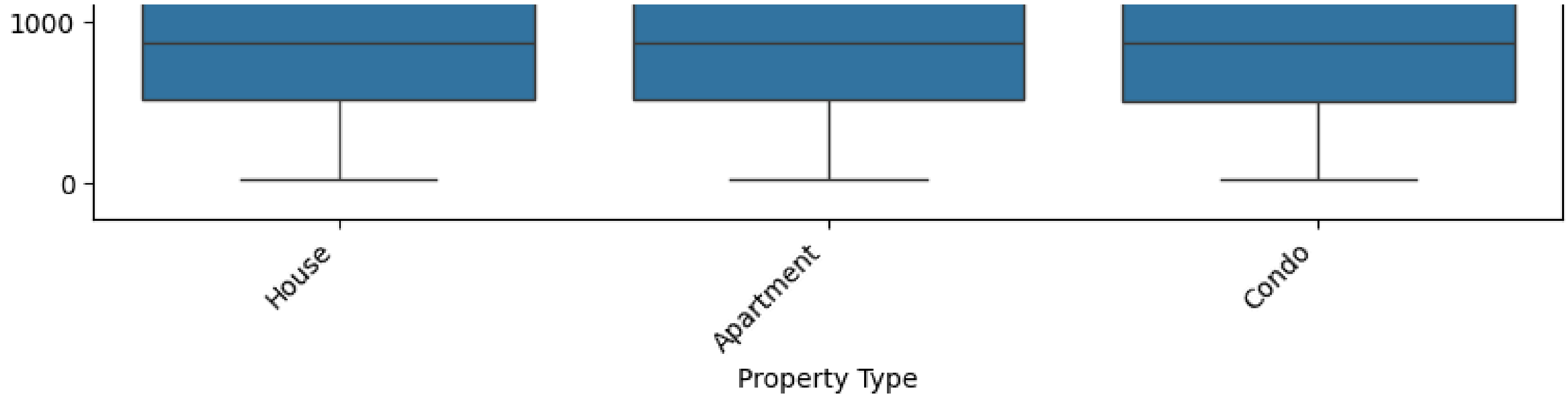


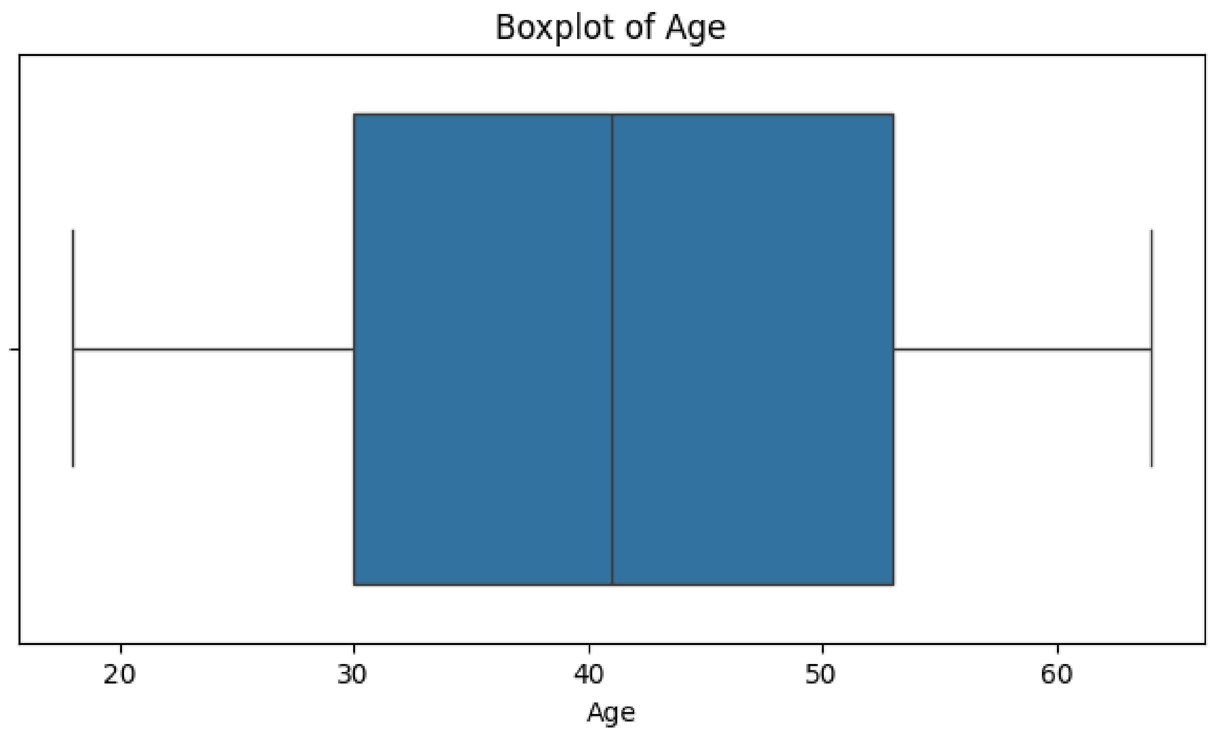


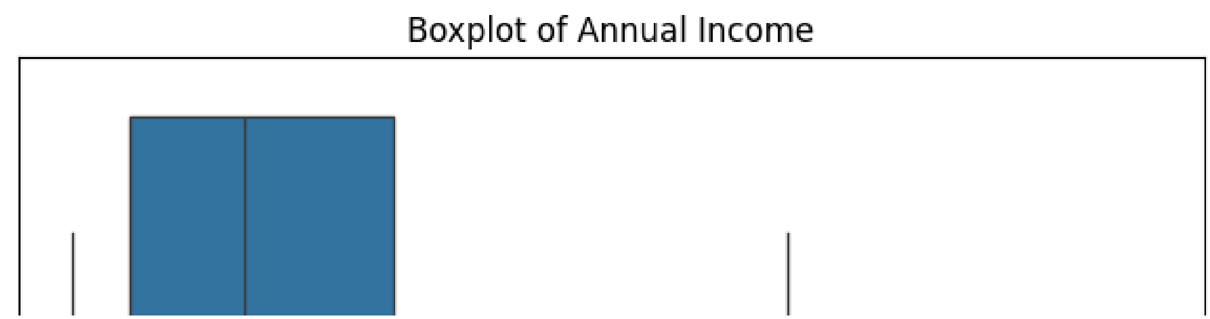
Premium Amount
3000
2000
1000
0
Weekly
Monthly
Daily
Rarely
Exercise Frequency

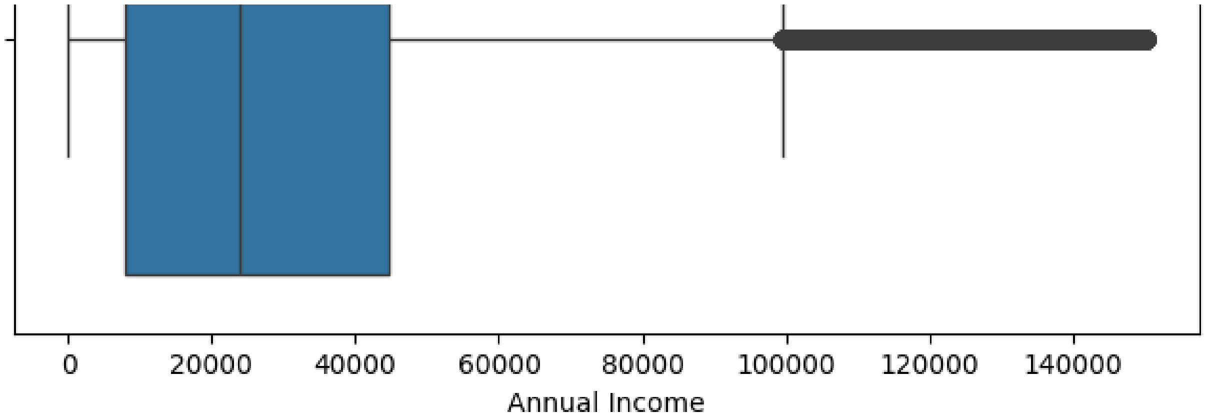


Relationship between Property Type and Premium Amount
Premium Amount
5000
4000
3000
2000

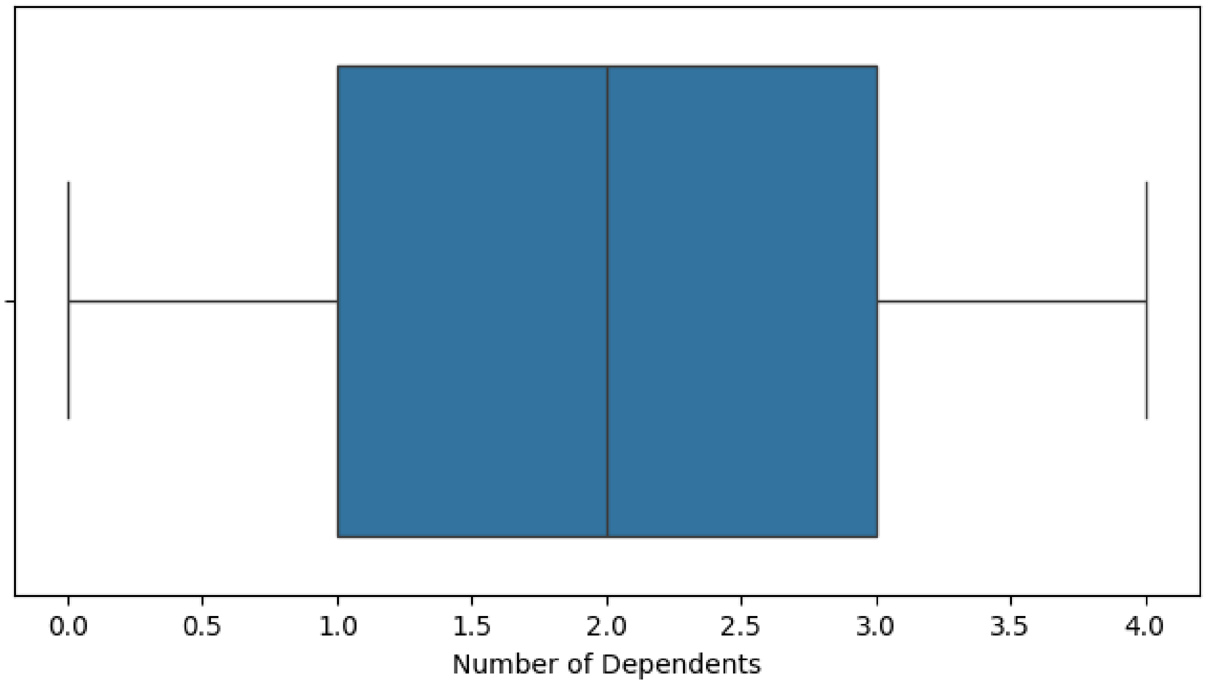

1000
0
House
Apartment
Condo
Property Type


Boxplot of Age
20
30
40
50
60
Age

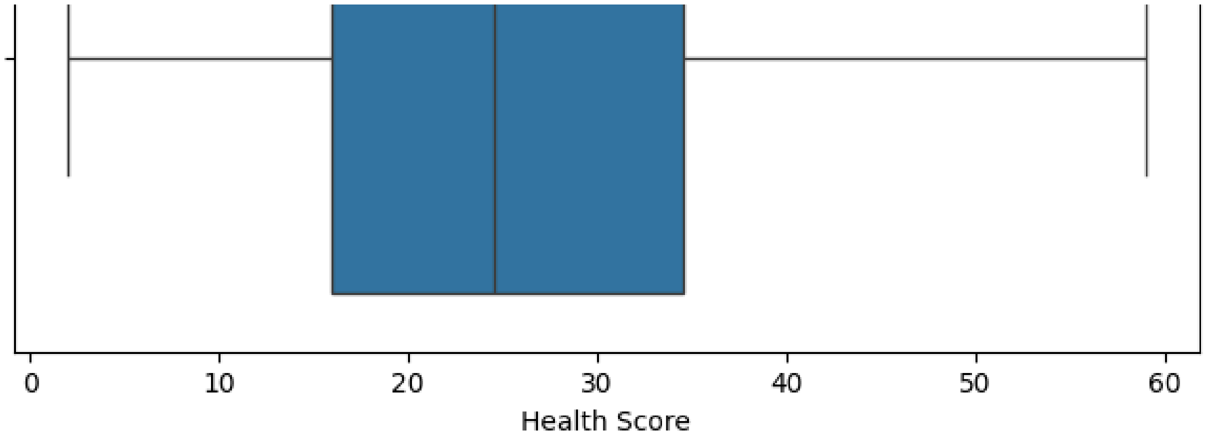

Boxplot of Annual Income

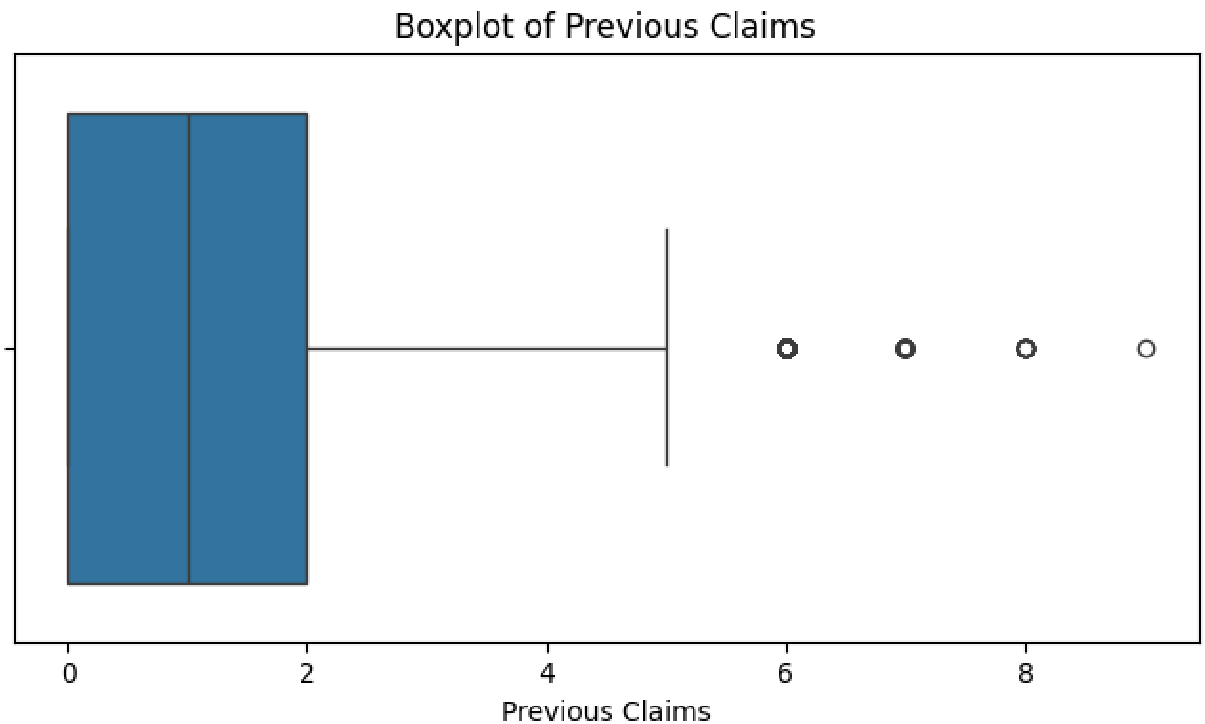


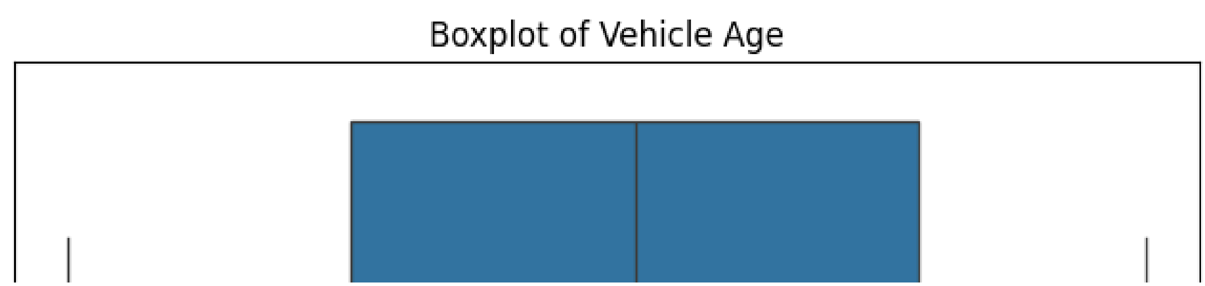
Boxplot of Number of Dependents

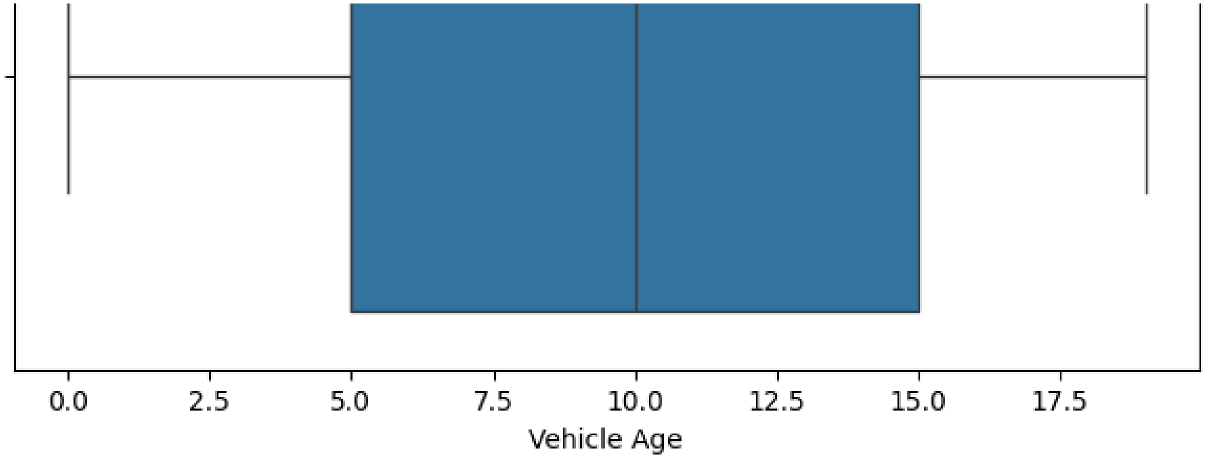


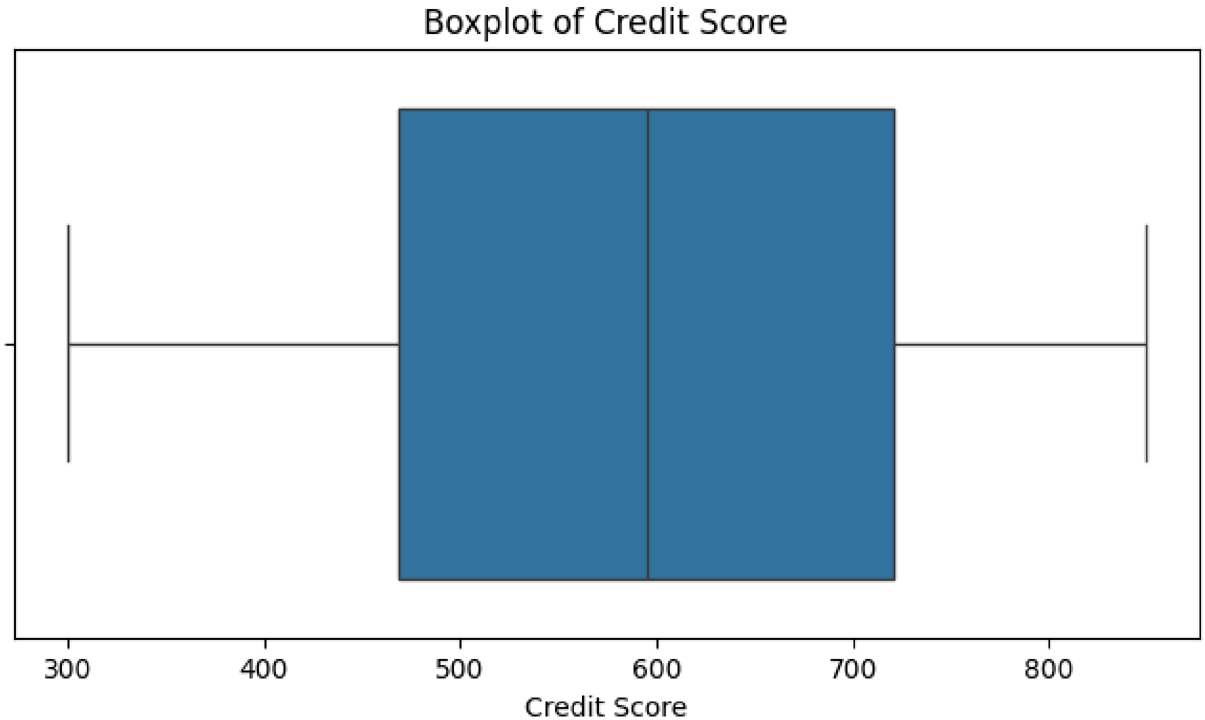
Boxplot of Health Score

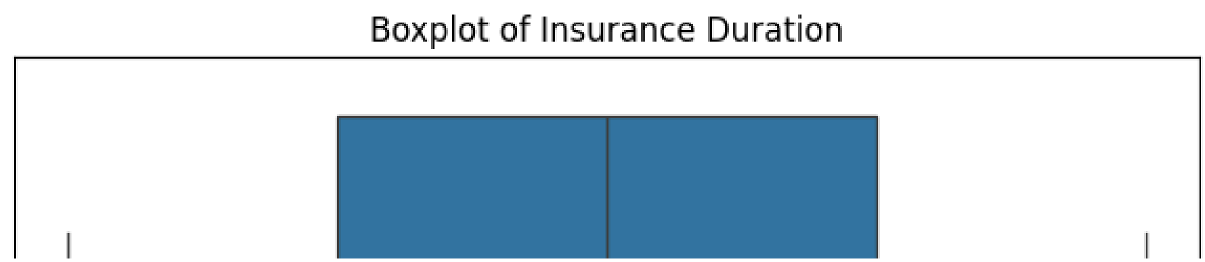

0
10
20
30
40
50
60
Health Score

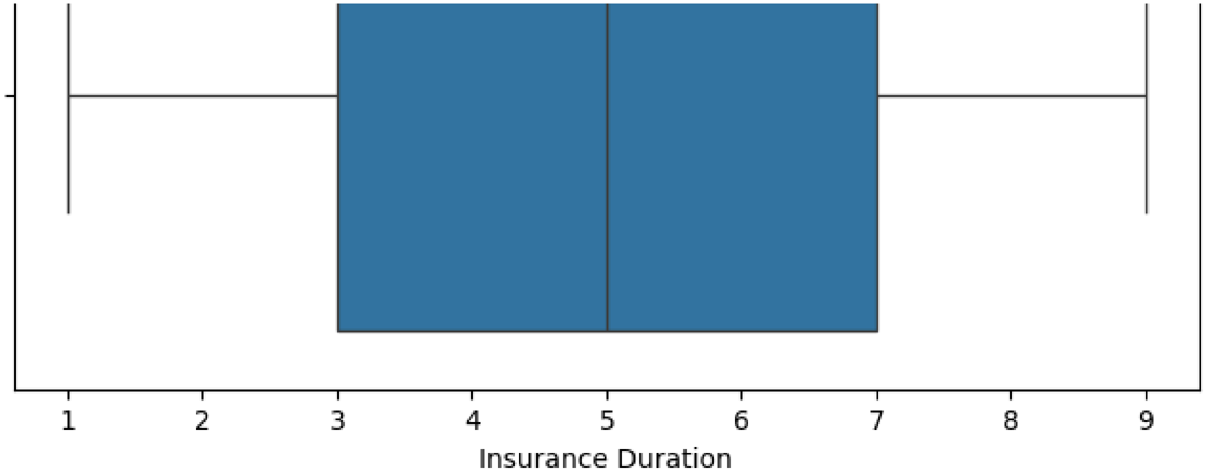

Boxplot of Previous Claims
0
2
4
6
8
Previous Claims


Boxplot of Vehicle Age


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Vehicle Age


Boxplot of Credit Score
300
400
500
600
700
800
Credit Score


Boxplot of Insurance Duration

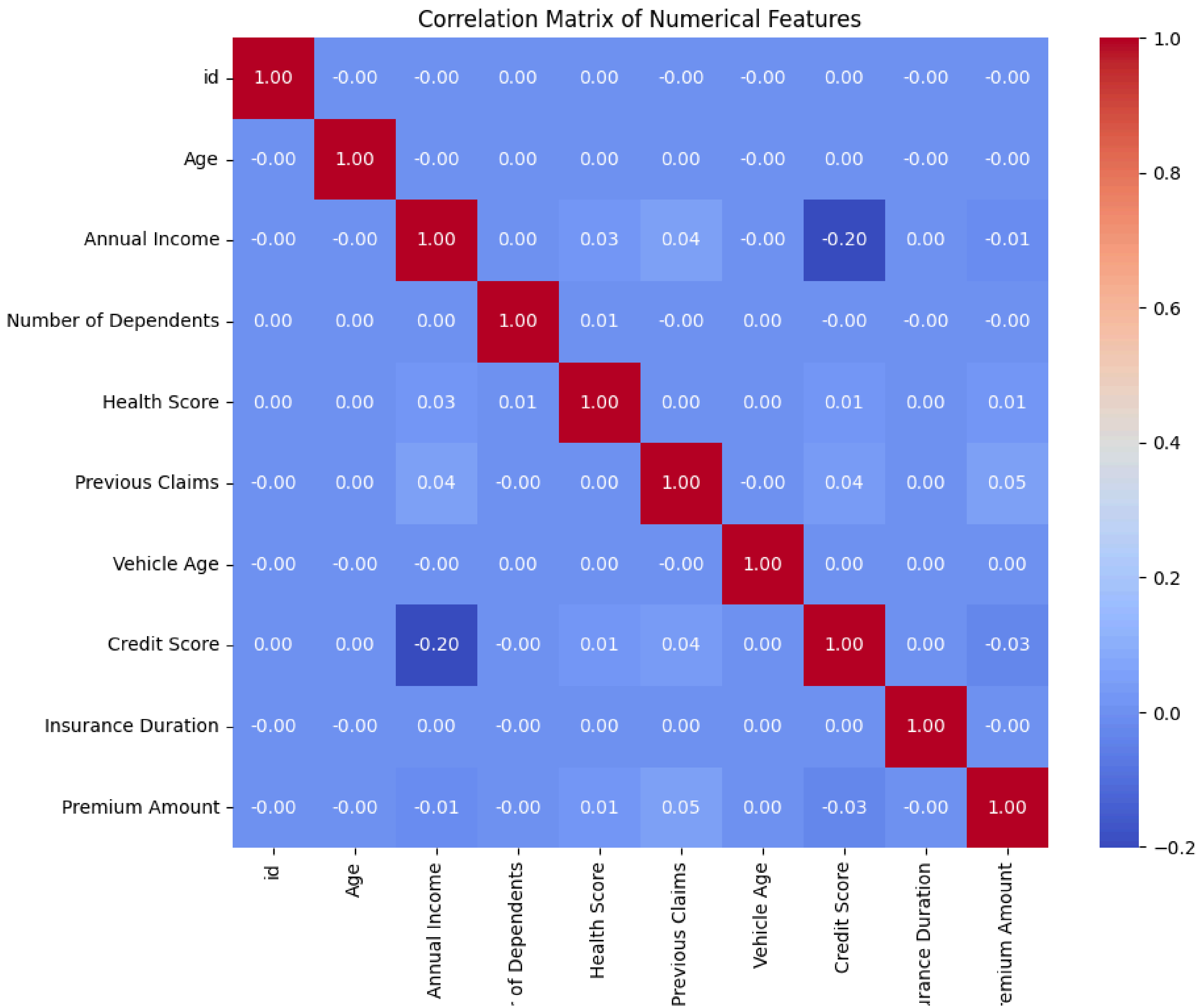


Next steps: Generate code with `missing_df` View recommended plots New interactive sheet

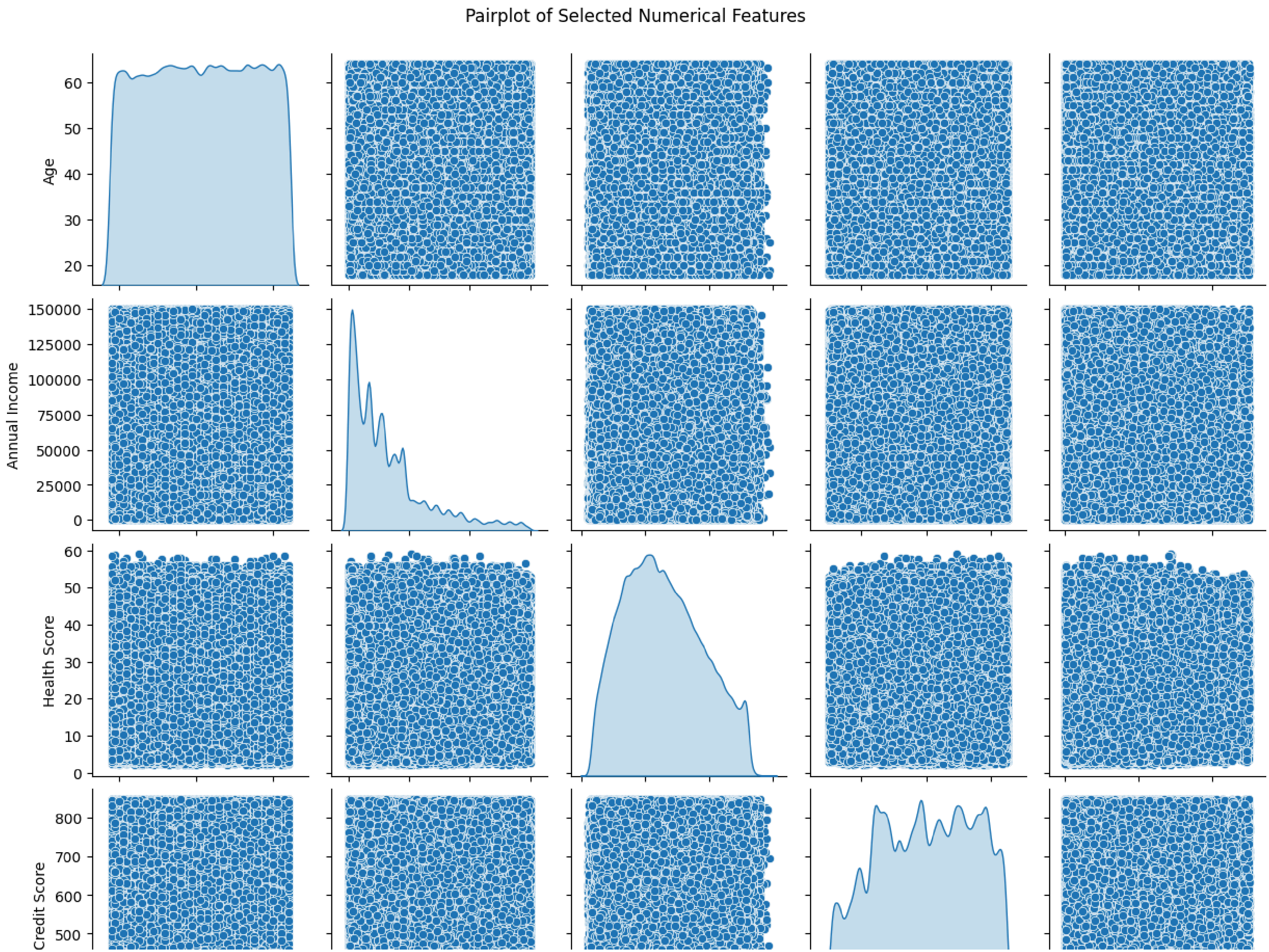
Reasoning: The previous code block generated a large number of plots, many of which are difficult to interpret or irrelevant. I will refine the code to focus on more informative visualizations and statistical summaries related to the target variable "Premium Amount" and key features, addressing missing values and outliers effectively.

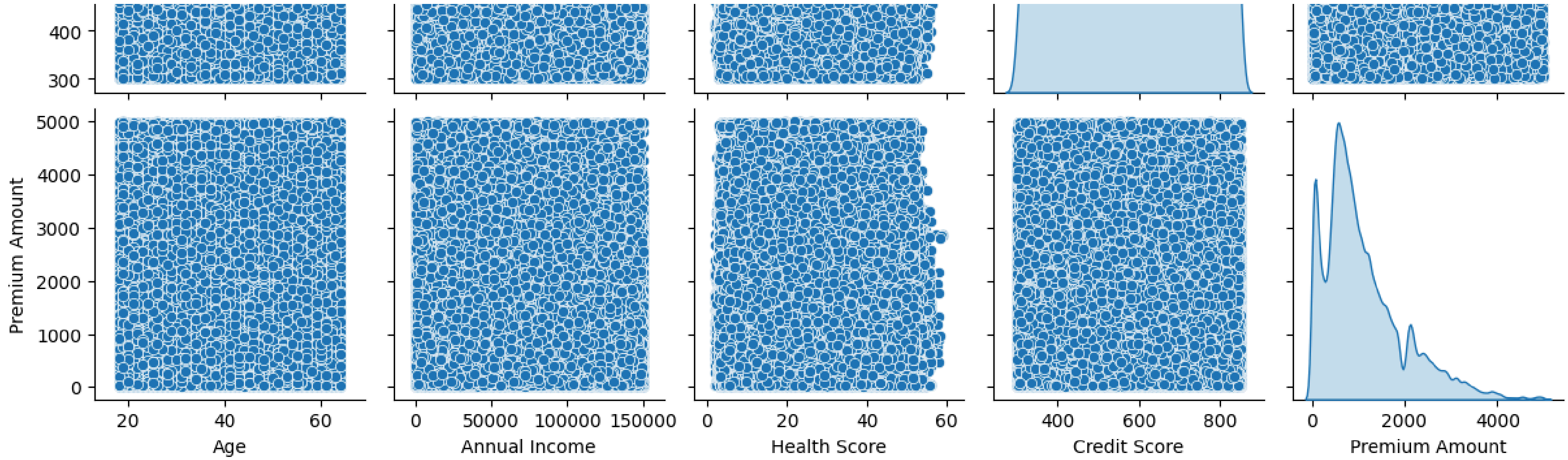
```
import matplotlib.pyplot as plt
import seaborn as sns

# Focus on relationships with 'Premium Amount'
numerical_features = df_train.select_dtypes(include=['number']).columns
categorical_features = df_train.select_dtypes(exclude=['number']).columns

# Correlation matrix (heatmap) for numerical features
plt.figure(figsize=(10, 8))
correlation_matrix = df_train[numerical_features].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()


# Pairplot for selected numerical features and 'Premium Amount'
selected_numerical_features = ['Age', 'Annual Income', 'Health Score', 'Credit Score',
sns.pairplot(df_train[selected_numerical_features], diag_kind='kde')
plt.suptitle('Pairplot of Selected Numerical Features', y=1.02)
plt.show()

# Box plots for categorical features vs. 'Premium Amount'
for col in categorical_features:
    if col not in ['Policy Start Date', 'id']:
        plt.figure(figsize=(10, 6))
        sns.boxplot(x=col, y='Premium Amount', data=df_train)
```

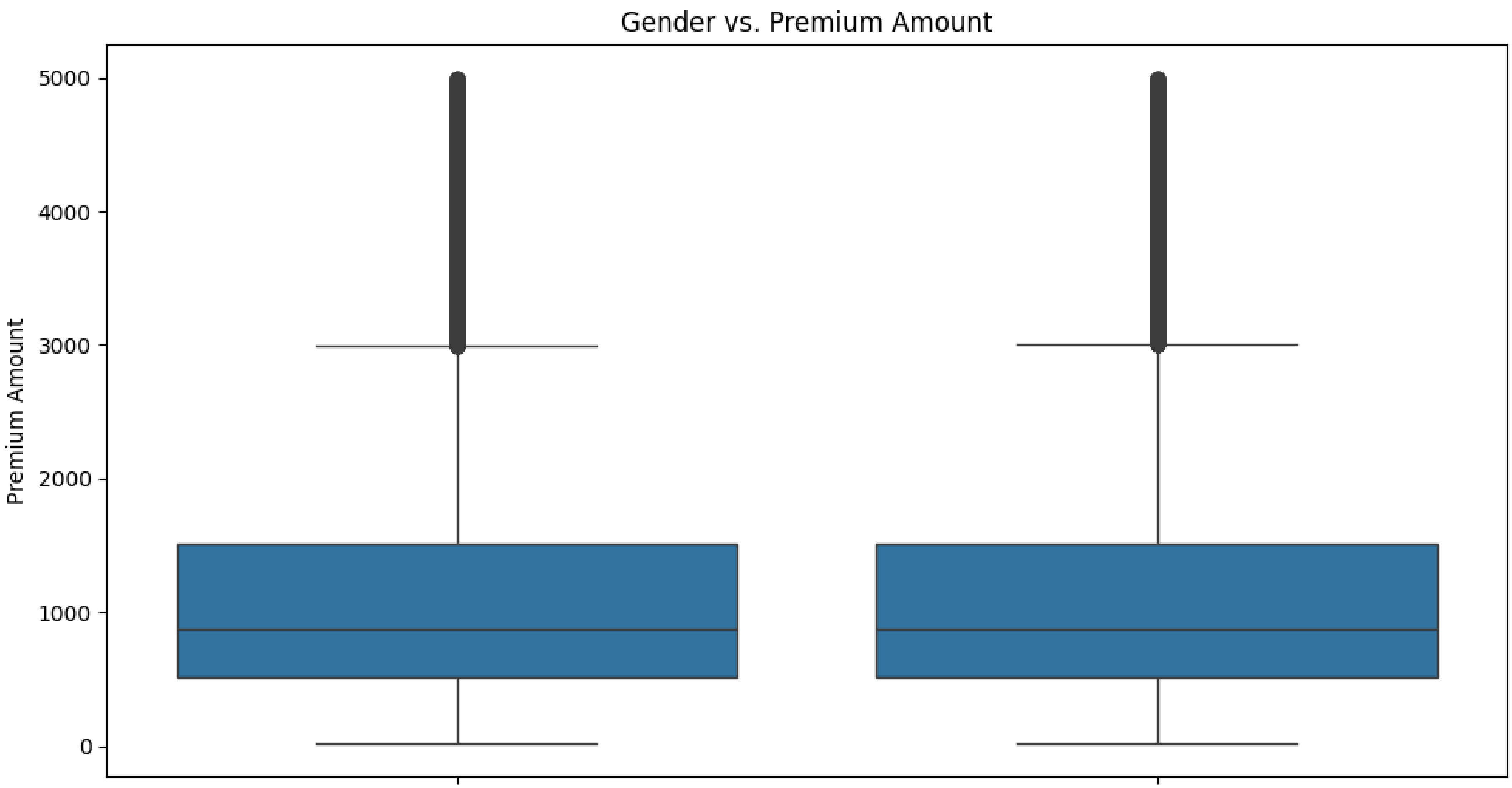
```
        plt.title(f'{col} vs. Premium Amount')
        plt.xticks(rotation=45, ha='right')
        plt.tight_layout()
        plt.show()

# Missing Value Analysis for crucial features
missing_cols = ['Age', 'Annual Income', 'Health Score', 'Credit Score','Previous Claim
for col in missing_cols:
    plt.figure(figsize=(8, 4))
    sns.histplot(df_train[col], kde=True)
    plt.title(f"Distribution of {col} with Missing Values")
    plt.show()

    plt.figure(figsize=(8, 4))
    sns.boxplot(y=col, data=df_train)
    plt.title(f"Box Plot for {col} with Missing Values")
    plt.show()
```


Correlation Matrix of Numerical Features
id
Age
Annual Income
Number of Dependents
Health Score
Previous Claims
Vehicle Age
Credit Score
Insurance Duration
Premium Amount
1.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00
-0.00 1.00 -0.00 0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00
-0.00 -0.00 1.00 0.00 0.03 0.04 -0.00 -0.20 0.00 -0.01
0.00 0.00 0.00 1.00 0.01 -0.00 0.00 -0.00 -0.00 -0.00
0.00 0.00 0.03 0.01 1.00 0.00 0.00 0.01 0.00 0.01
-0.00 0.00 0.04 -0.00 0.00 1.00 -0.00 0.04 0.00 0.05
-0.00 -0.00 -0.00 0.00 0.00 -0.00 1.00 0.00 0.00 0.00
0.00 0.00 -0.20 -0.00 0.01 0.04 0.00 1.00 0.00 -0.03
-0.00 -0.00 0.00 -0.00 0.00 0.00 0.00 0.00 1.00 -0.00
-0.00 -0.00 -0.01 -0.00 0.01 0.05 0.00 -0.03 -0.00 1.00
id
Age
Annual Income
of Dependents
Health Score
Previous Claims
Vehicle Age
Credit Score
rance Duration
emium Amount
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

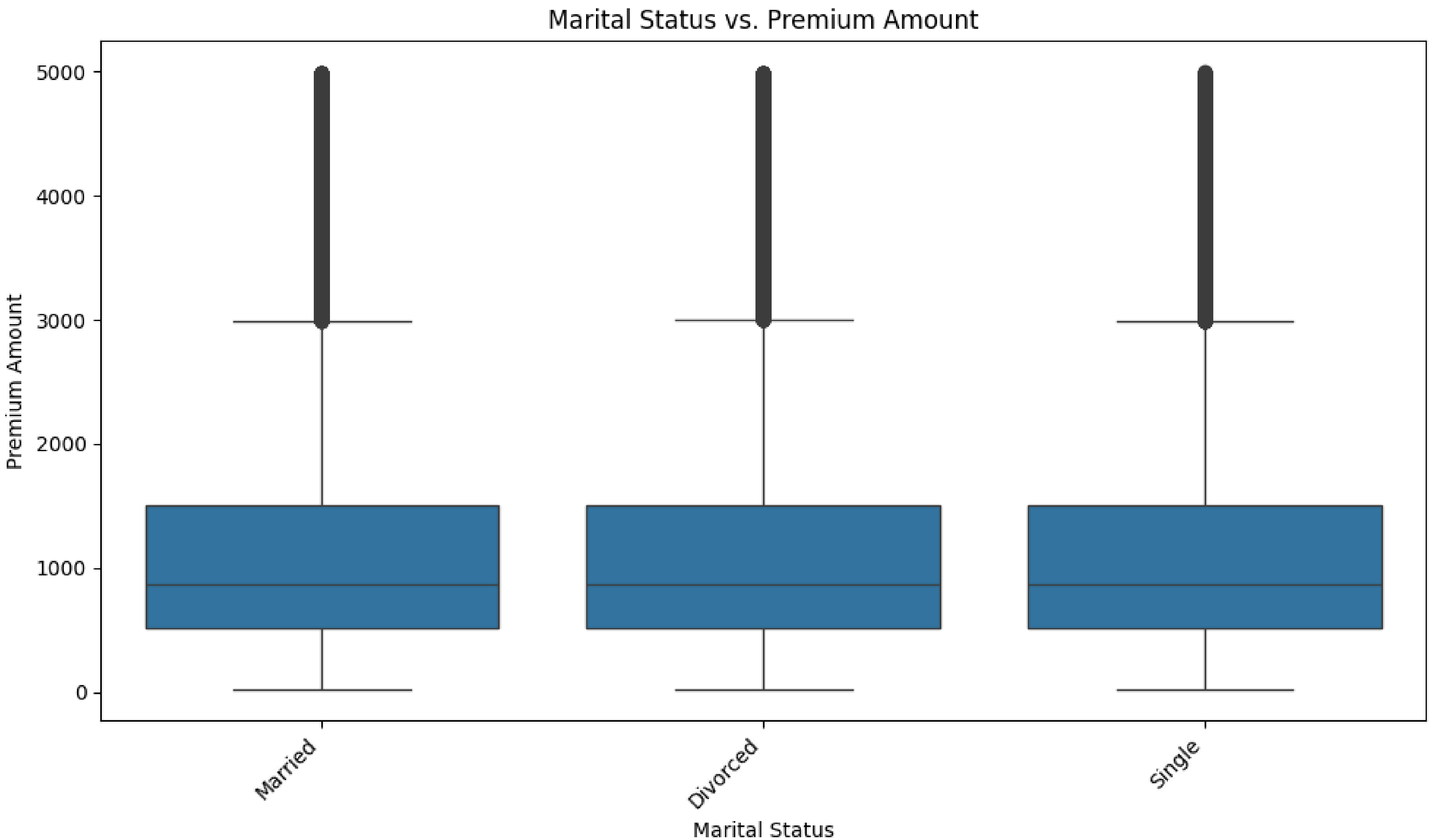
Numbei Inst Pi

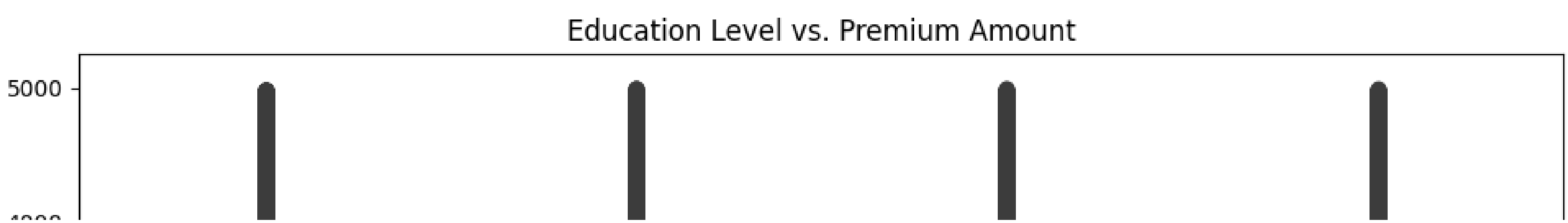


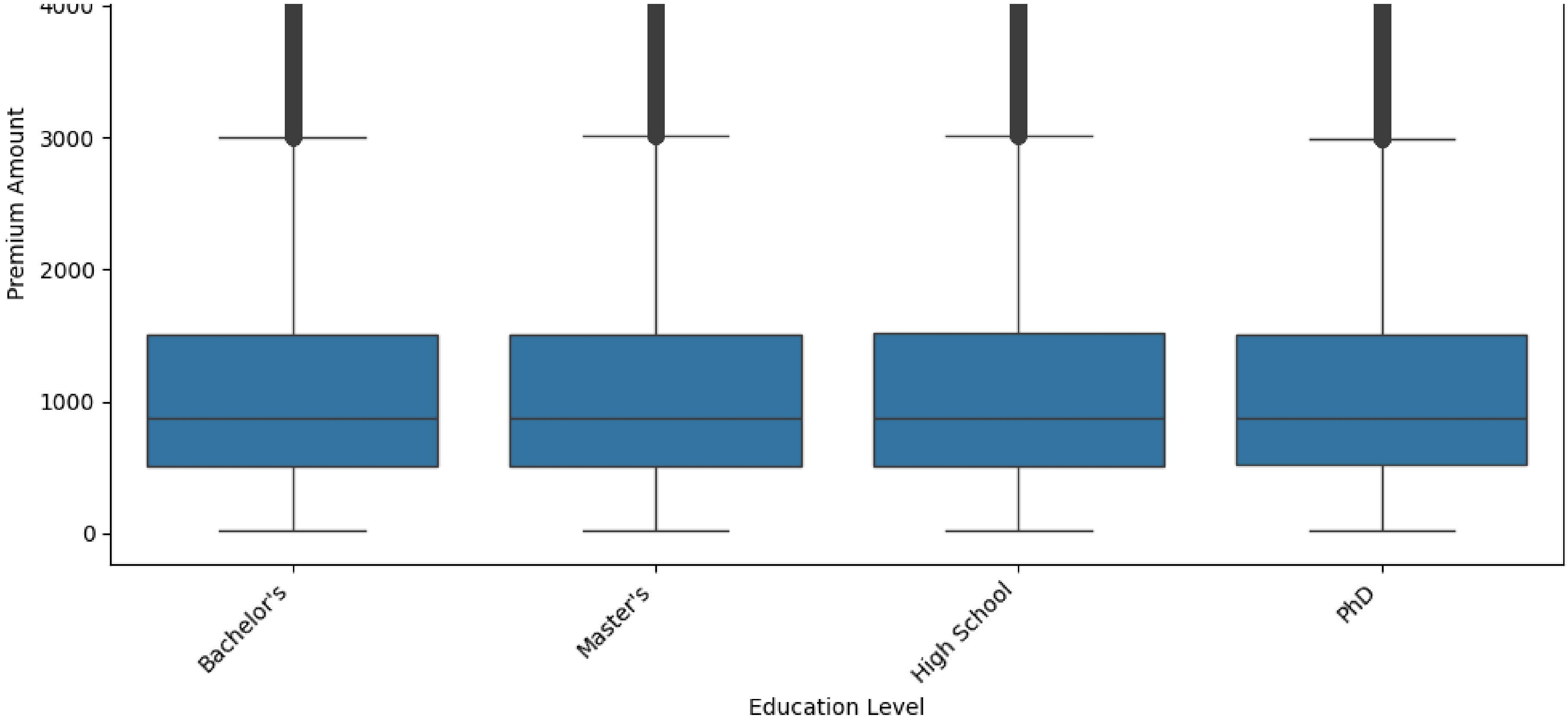

Premium Amount
Age
Annual Income
Health Score
Credit Score
Premium Amount

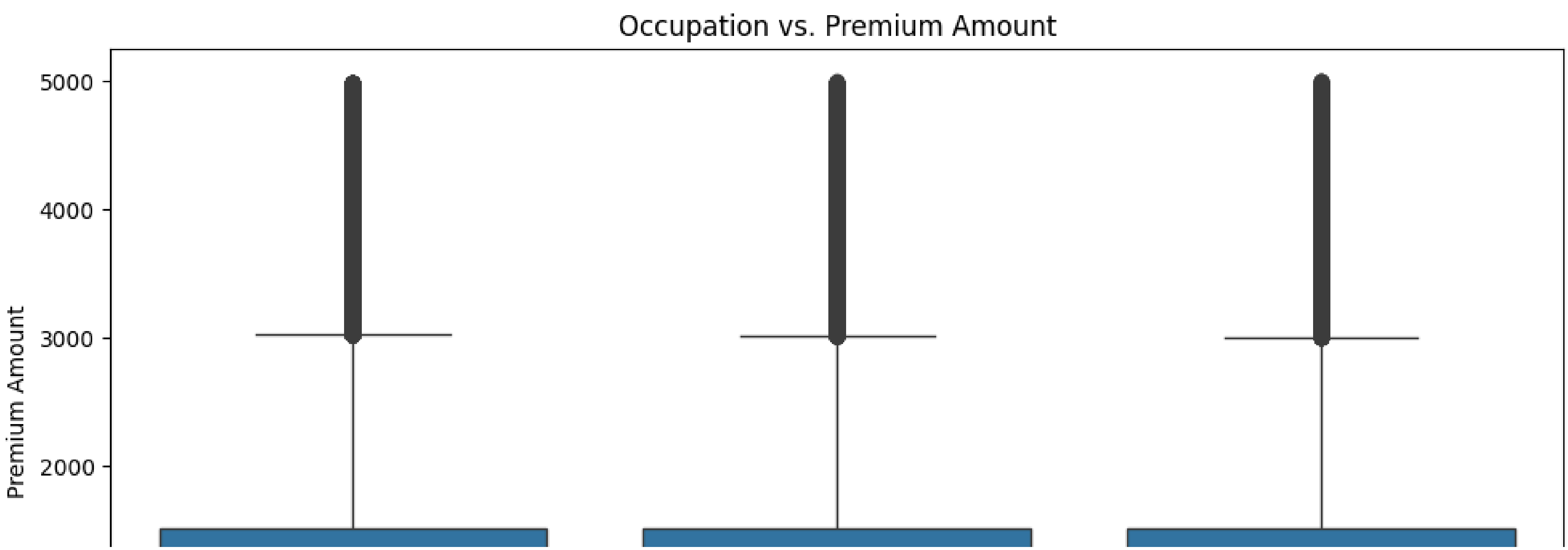

Gender vs. Premium Amount
Premium Amount

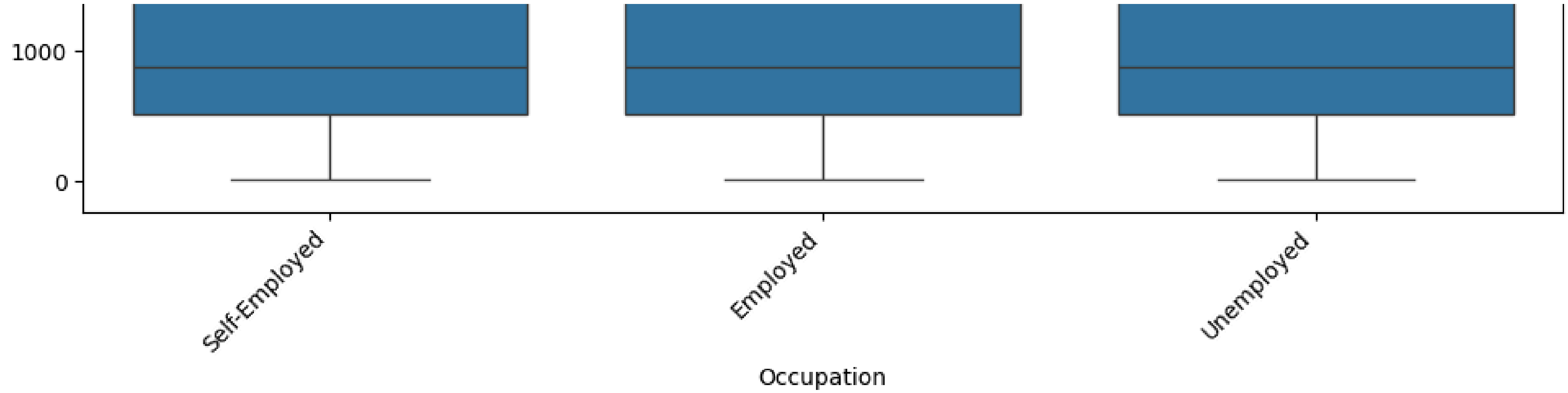

Female
Male
Gender

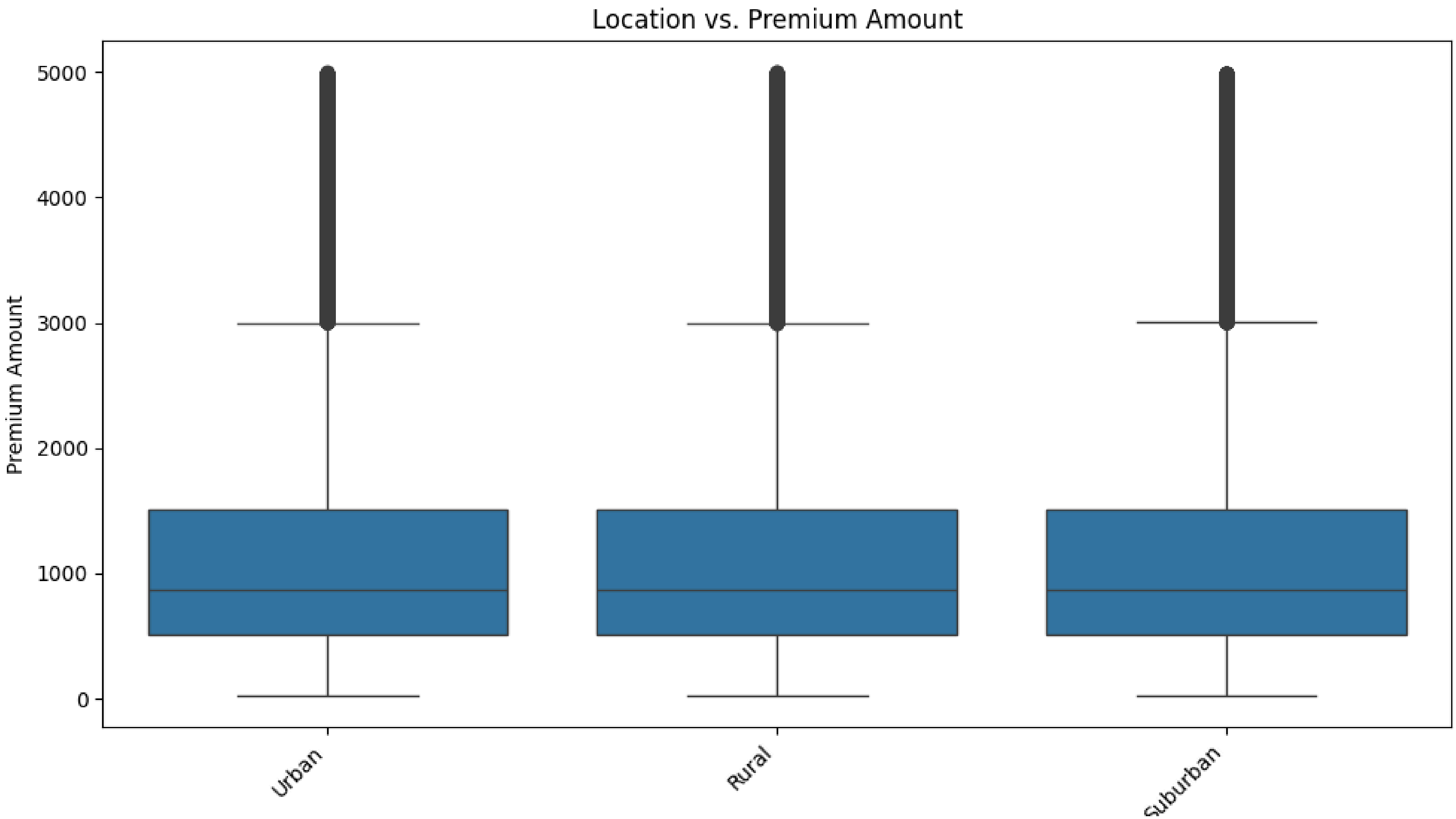

Marital Status vs. Premium Amount
5000
4000
3000
2000
1000
0
Premium Amount
Married
Divorced
Single
Marital Status


Education Level vs. Premium Amount
5000

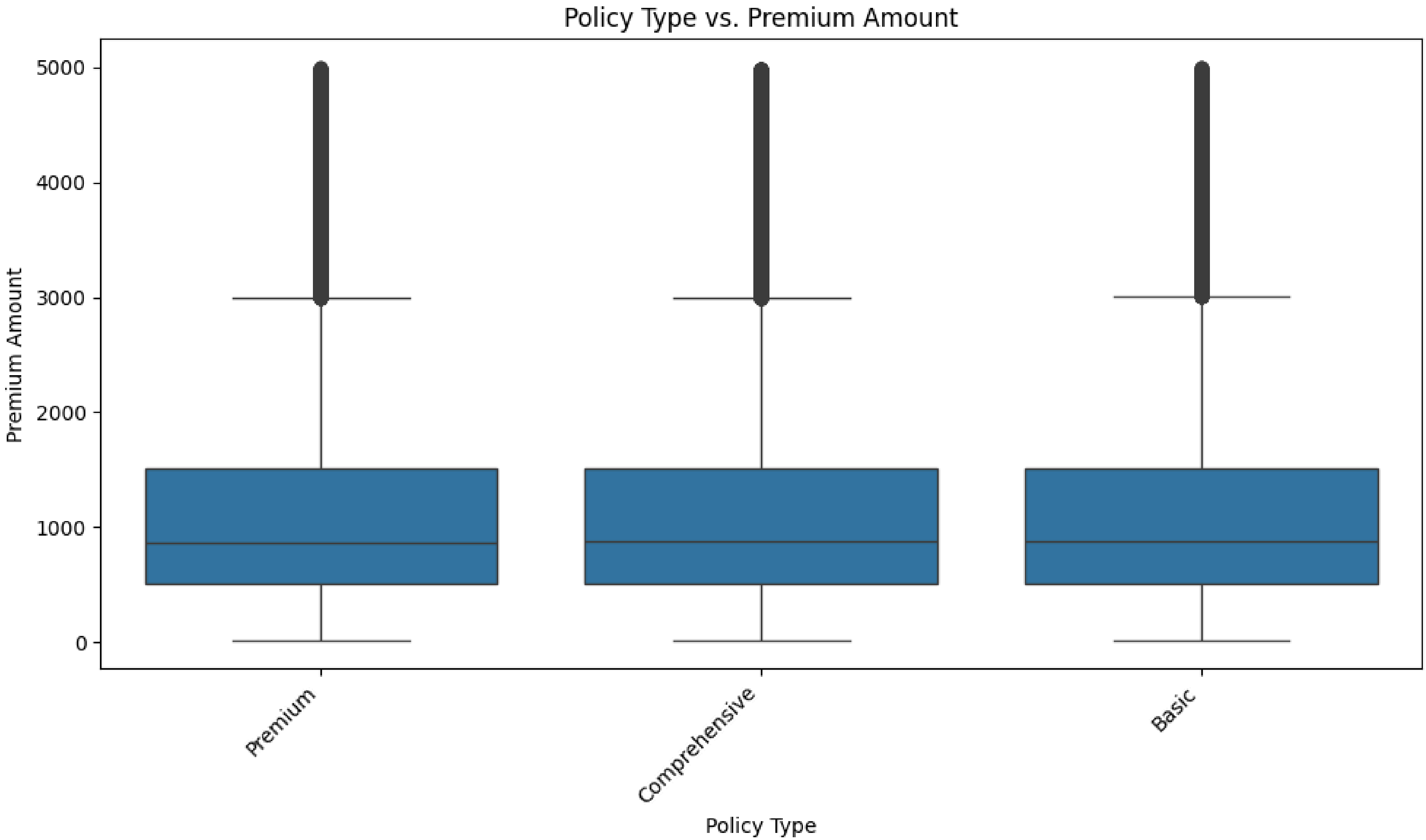

4000
3000
2000
1000
0
Premium Amount
Bachelor's
Master's
High School
PhD
Education Level

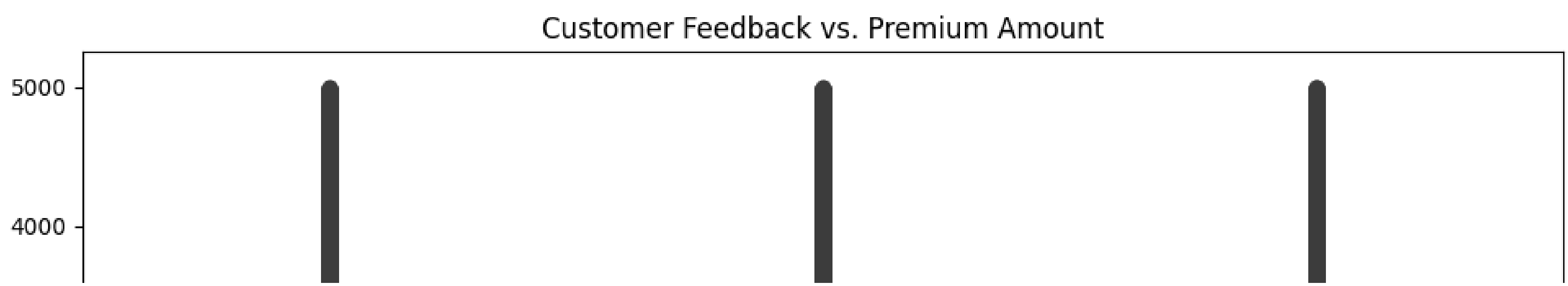

Occupation vs. Premium Amount
5000
4000
3000
2000
Premium Amount

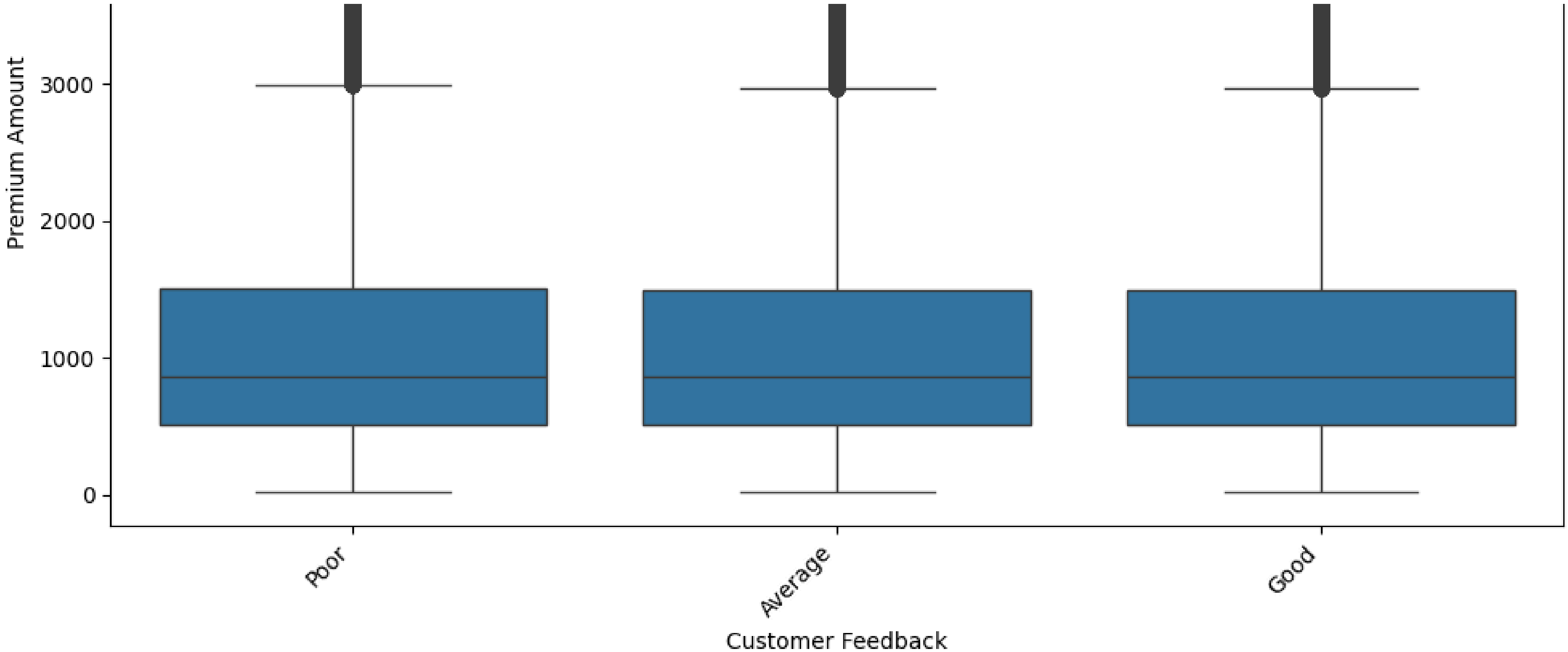

1000
0
Self-Employed
Employed
Unemployed
Occupation

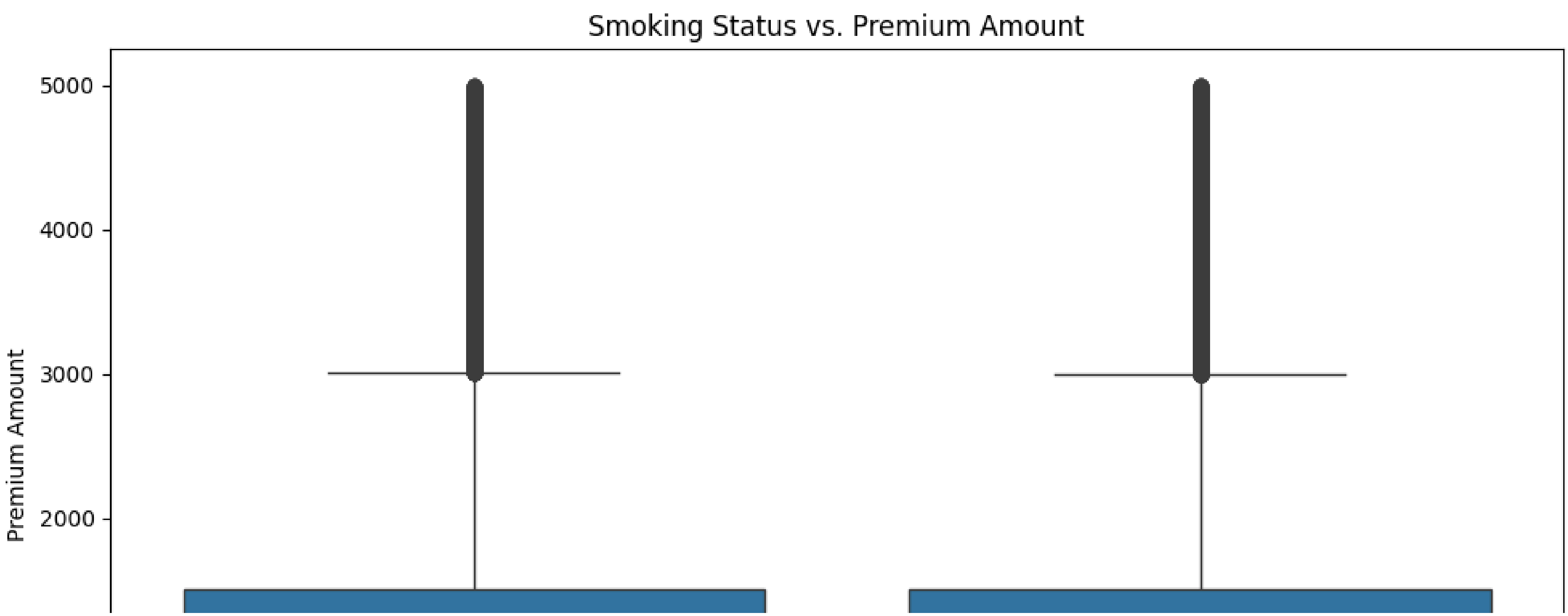

Location vs. Premium Amount
5000
4000
3000
2000
1000
0
Premium Amount
Urban
Rural
Suburban

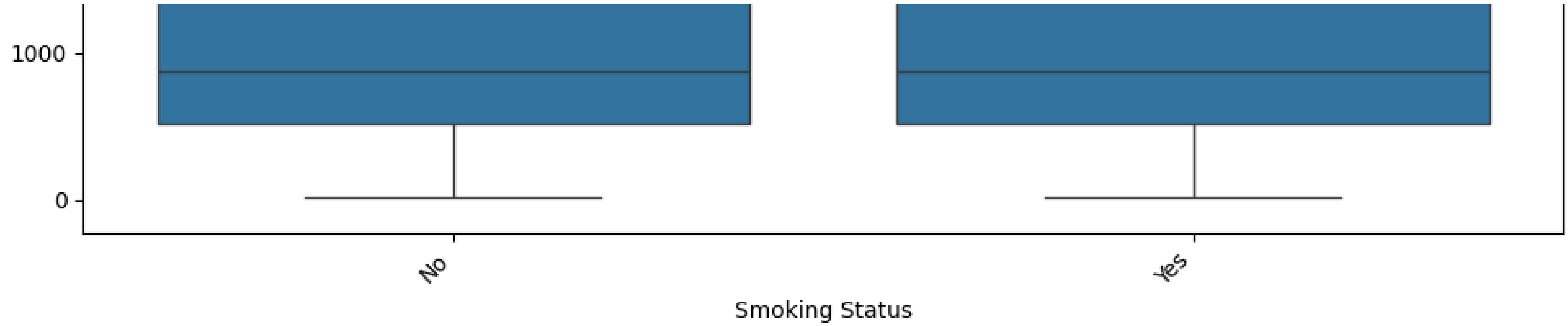
Location

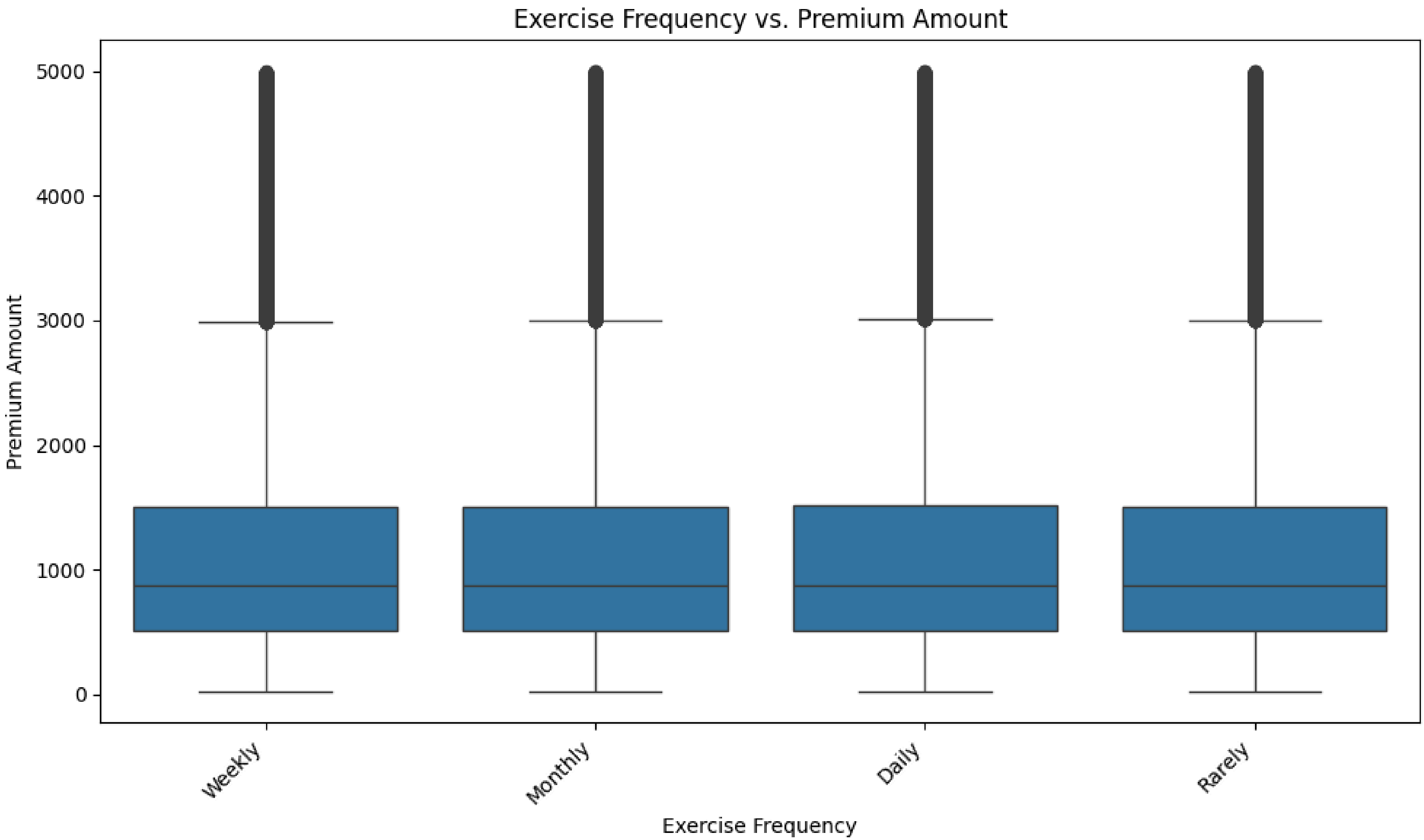

Policy Type vs. Premium Amount
Premium Amount
5000
4000
3000
2000
1000
0
Premium
Comprehensive
Basic
Policy Type

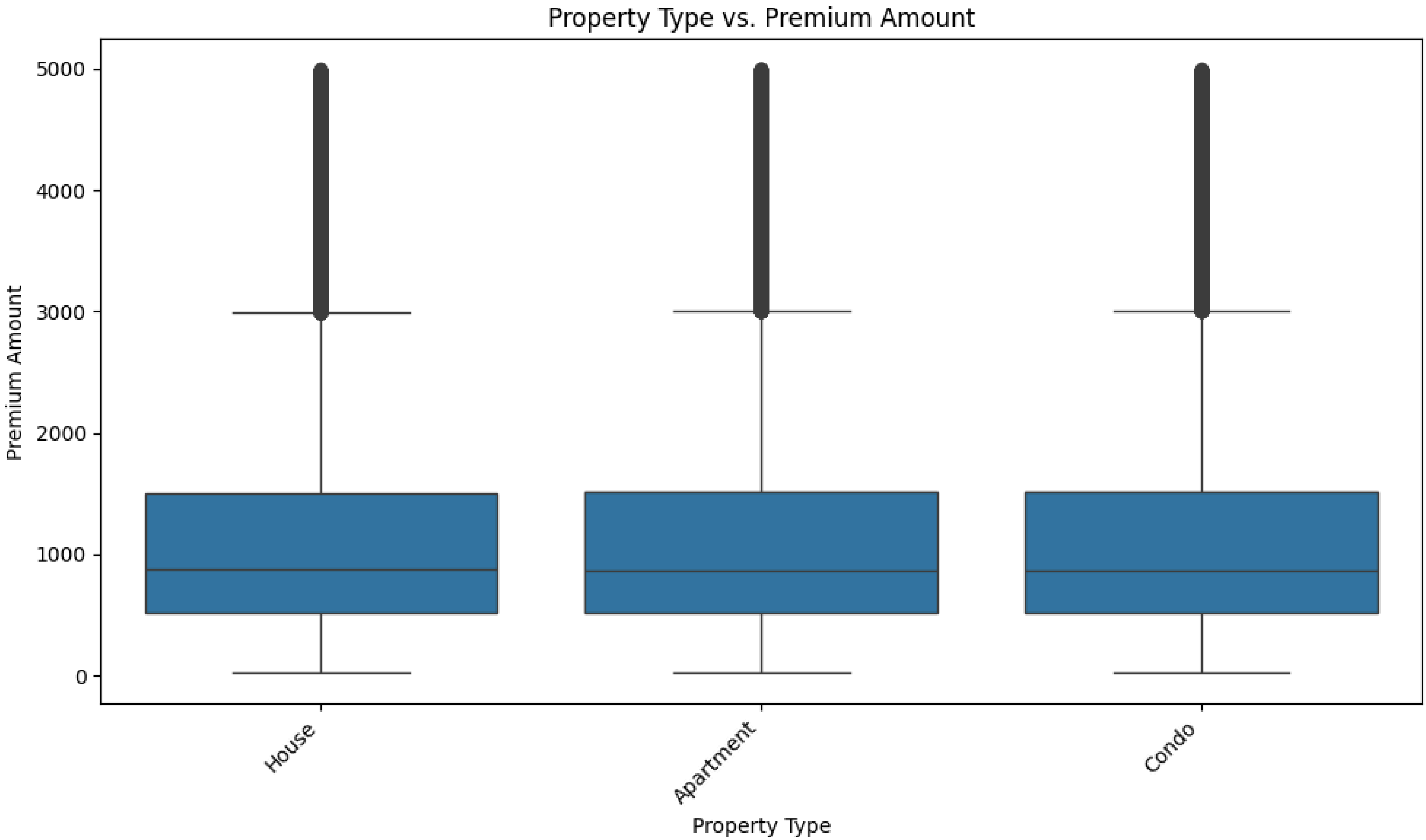

Customer Feedback vs. Premium Amount
5000
4000

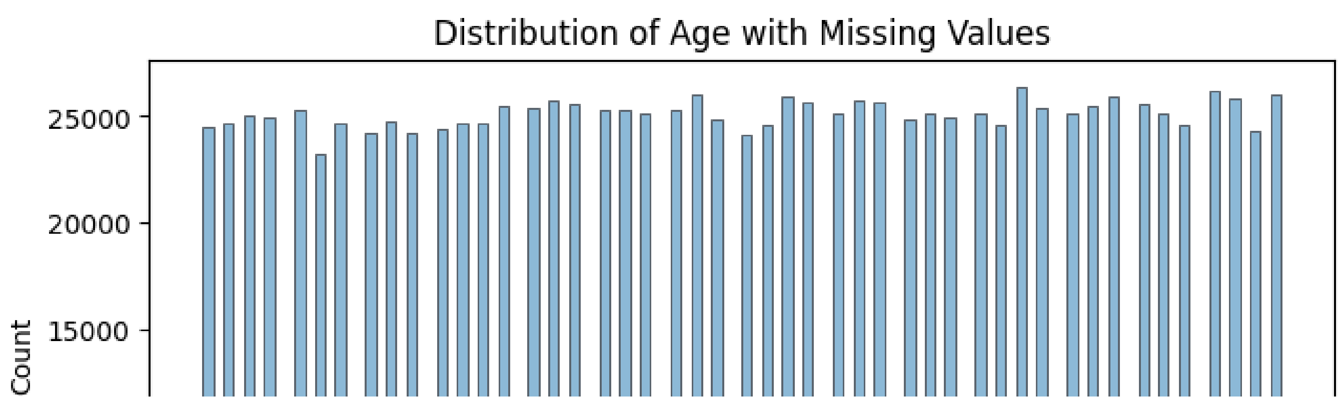

Premium Amount
3000
2000
1000
0
Poor
Average
Good
Customer Feedback

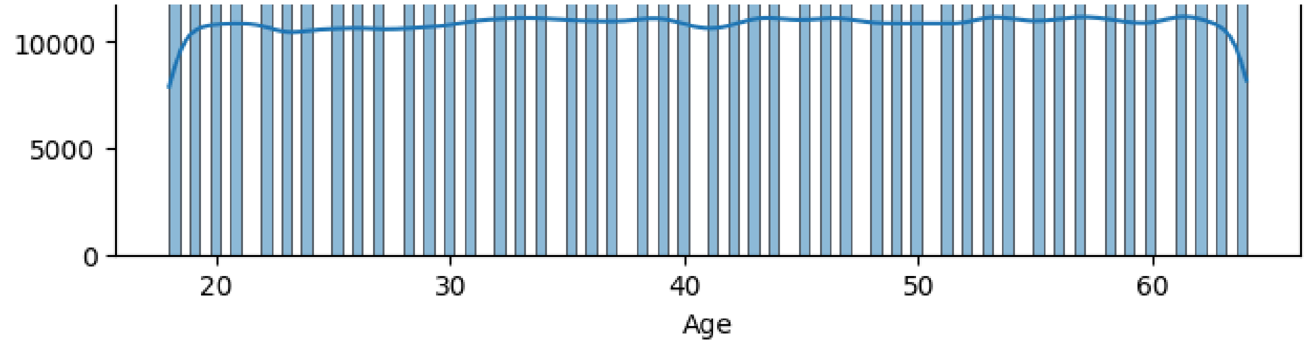

Smoking Status vs. Premium Amount
Premium Amount
5000
4000
3000
2000

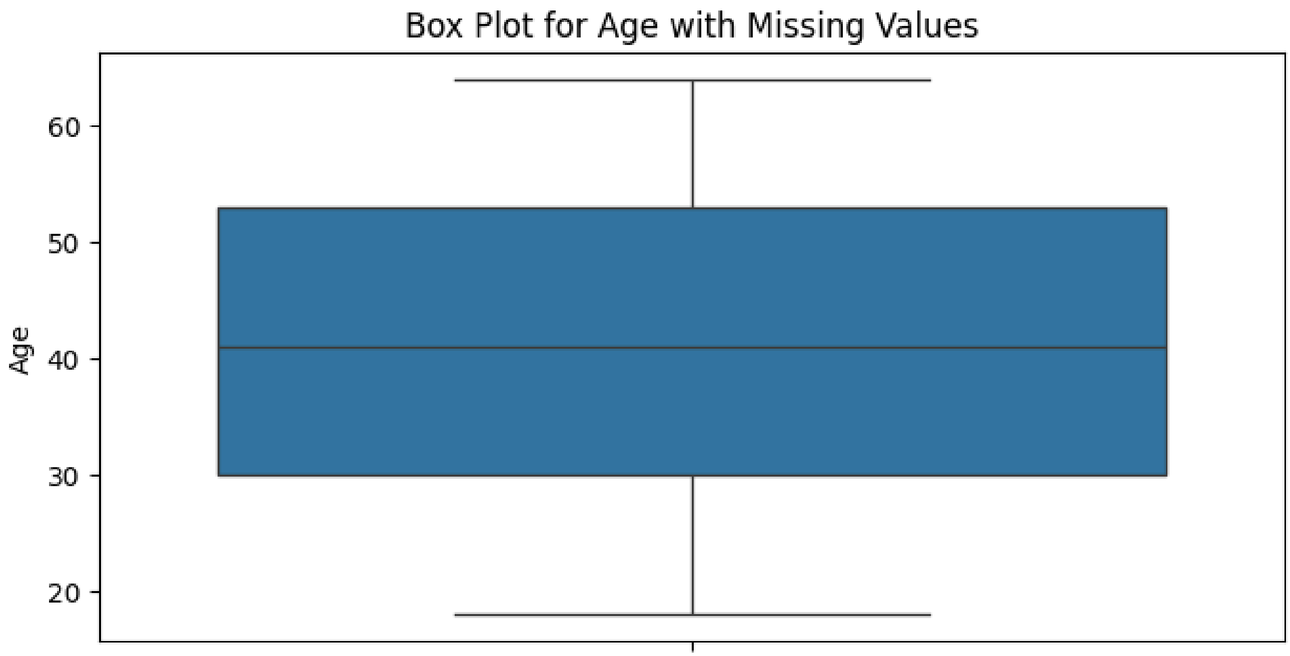

1000
0
No
Yes
Smoking Status

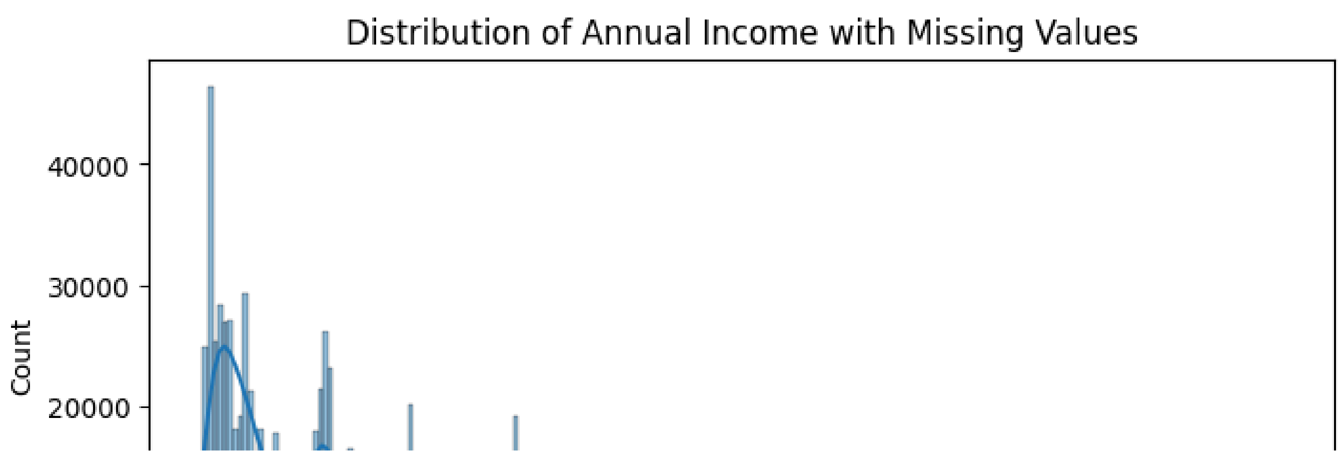

Exercise Frequency vs. Premium Amount
Premium Amount
5000
4000
3000
2000
1000
0
Weekly
Monthly
Daily
Rarely
Exercise Frequency

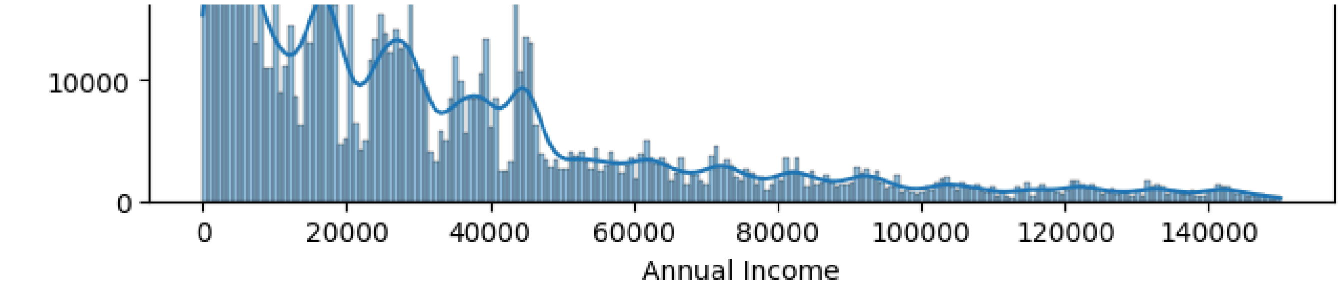

Property Type vs. Premium Amount
Premium Amount
5000
4000
3000
2000
1000
0
House
Apartment
Condo
Property Type


Distribution of Age with Missing Values
Count
25000
20000
15000

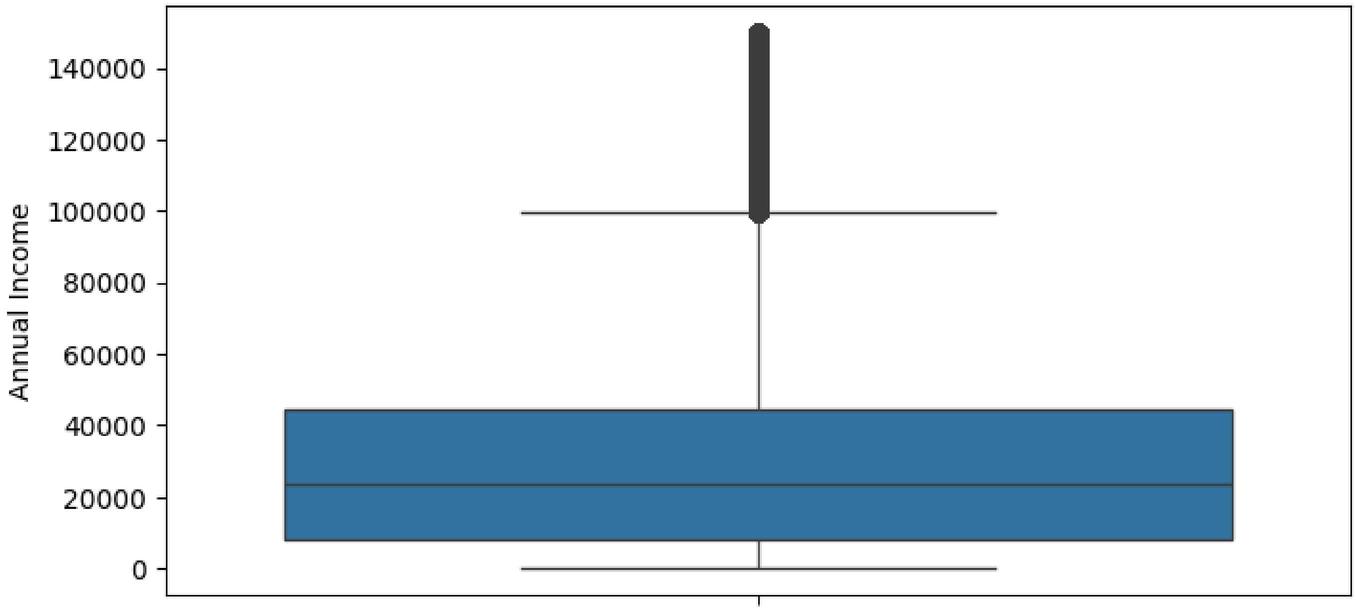

10000
5000
0
20
30
40
50
60
Age


Box Plot for Age with Missing Values
60
50
40
30
20
Age

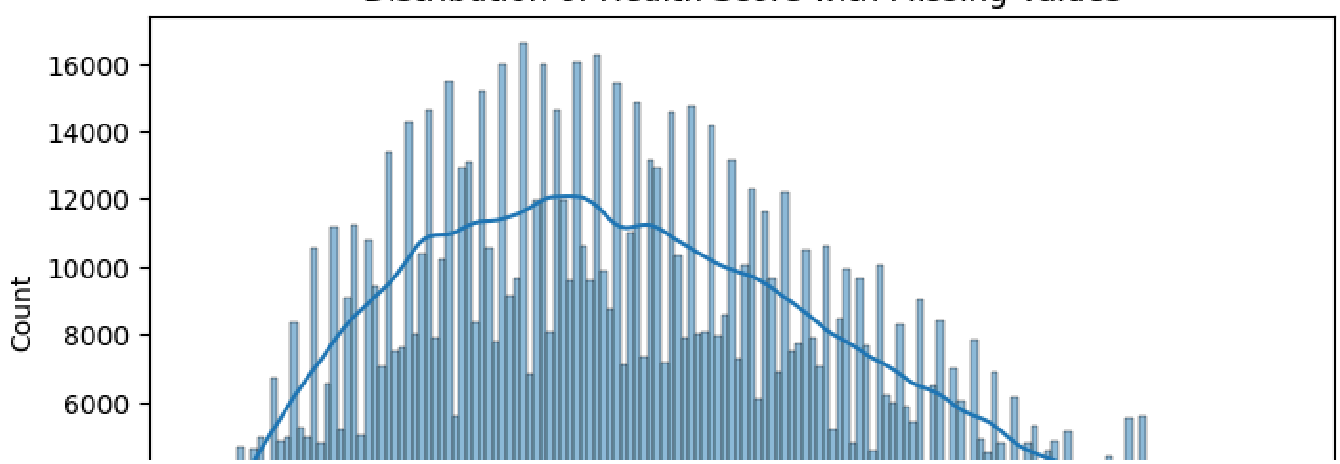

Distribution of Annual Income with Missing Values
40000
30000
20000
Count

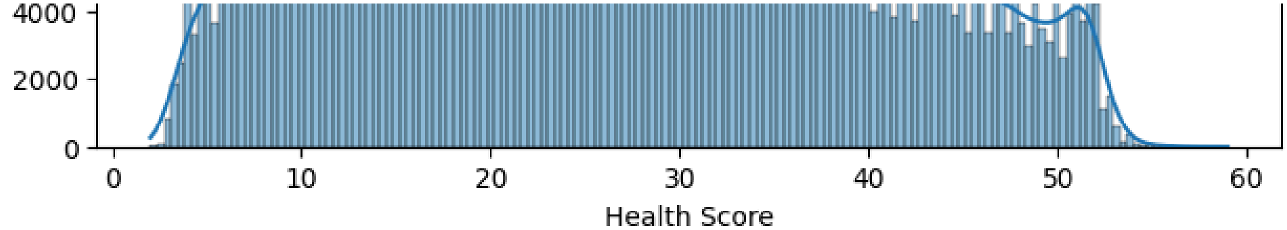


Box Plot for Annual Income with Missing Values

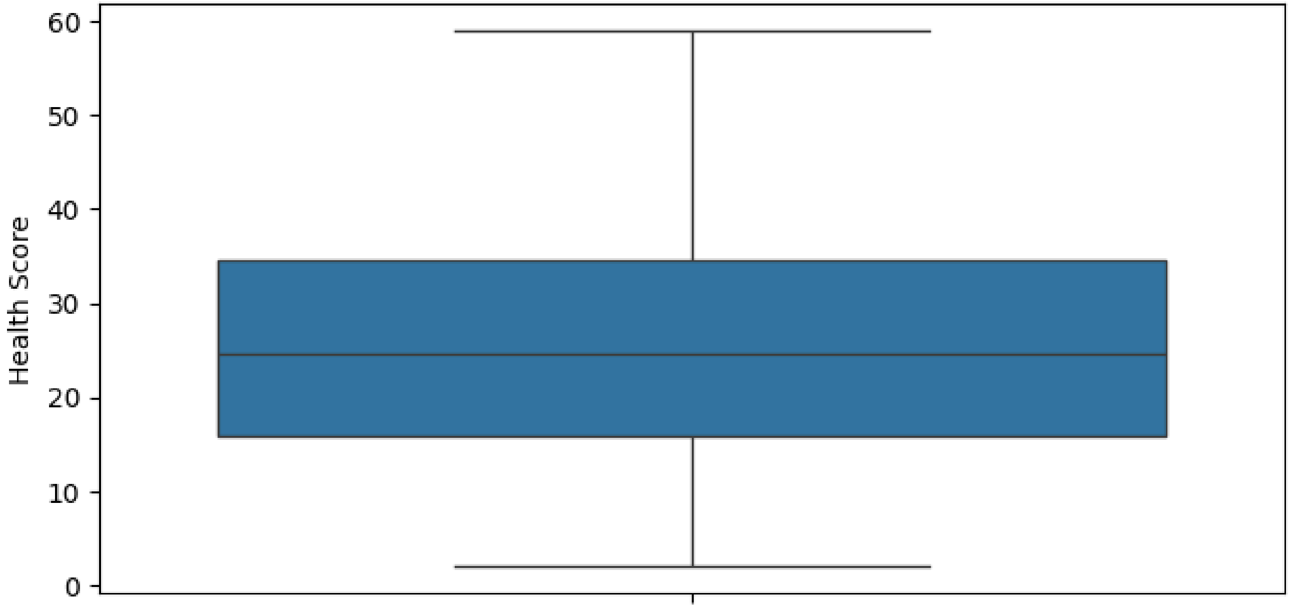


Distribution of Health Score with Missing Values

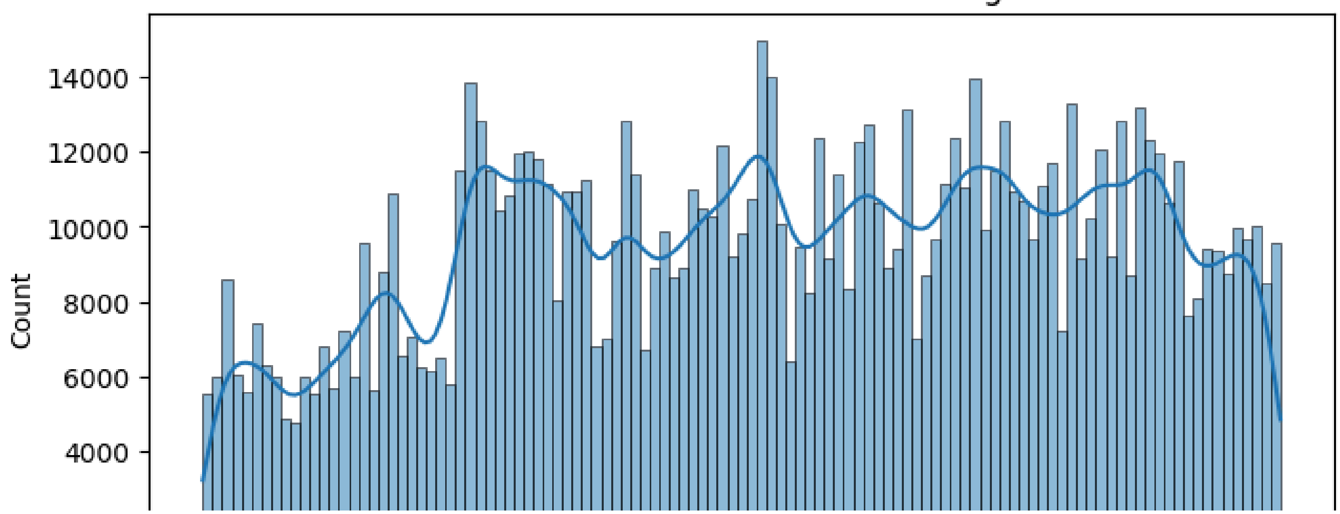


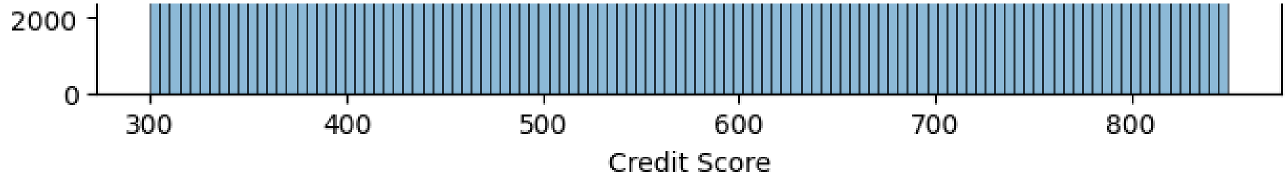


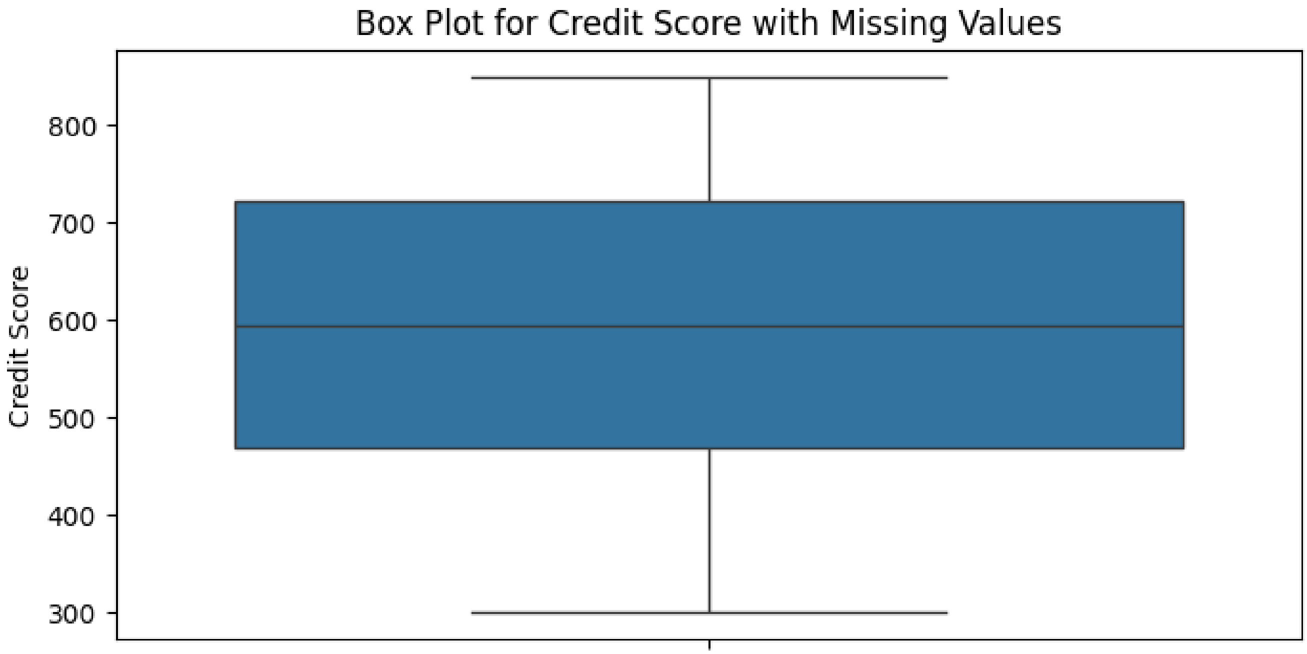
Box Plot for Health Score with Missing Values

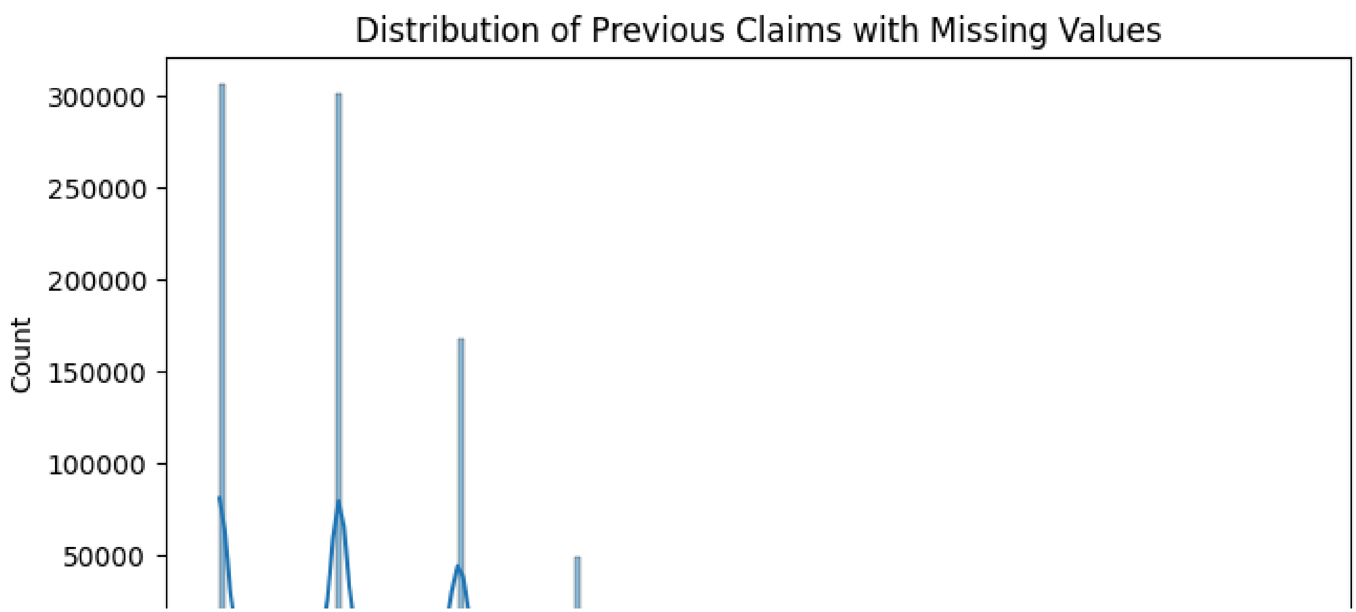


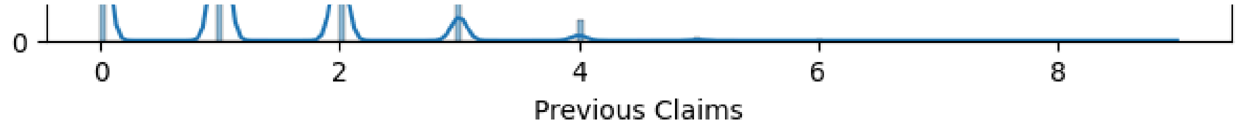
Distribution of Credit Score with Missing Values



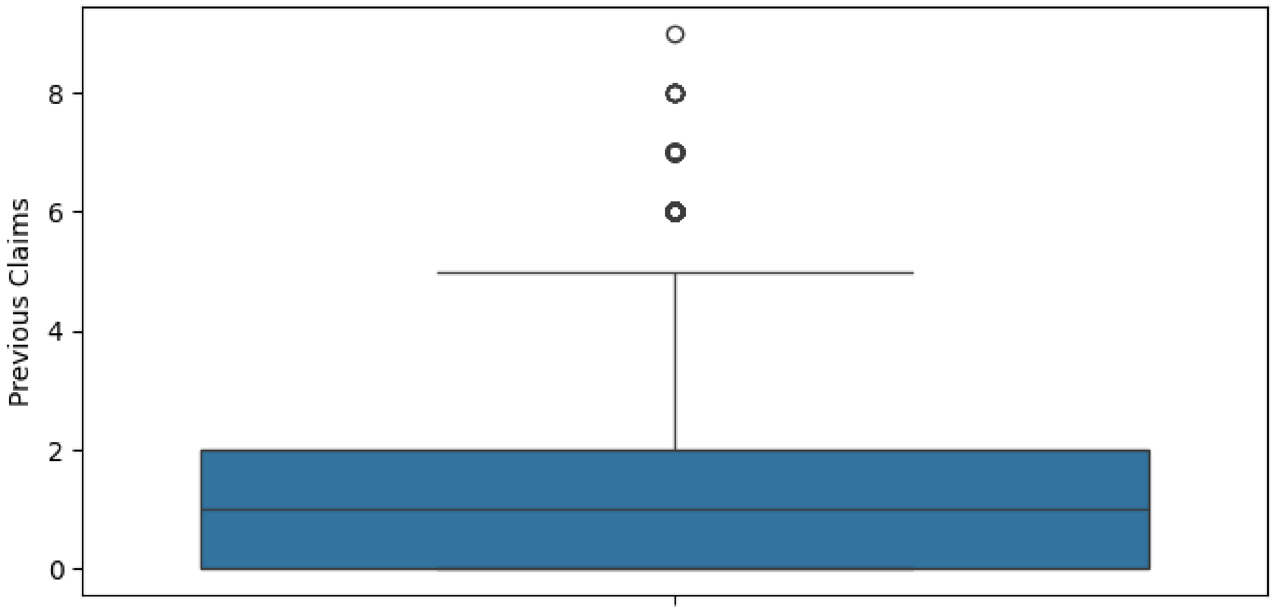

2000
0
300
400
500
600
700
800
Credit Score


Box Plot for Credit Score with Missing Values
800
700
600
500
400
300
Credit Score

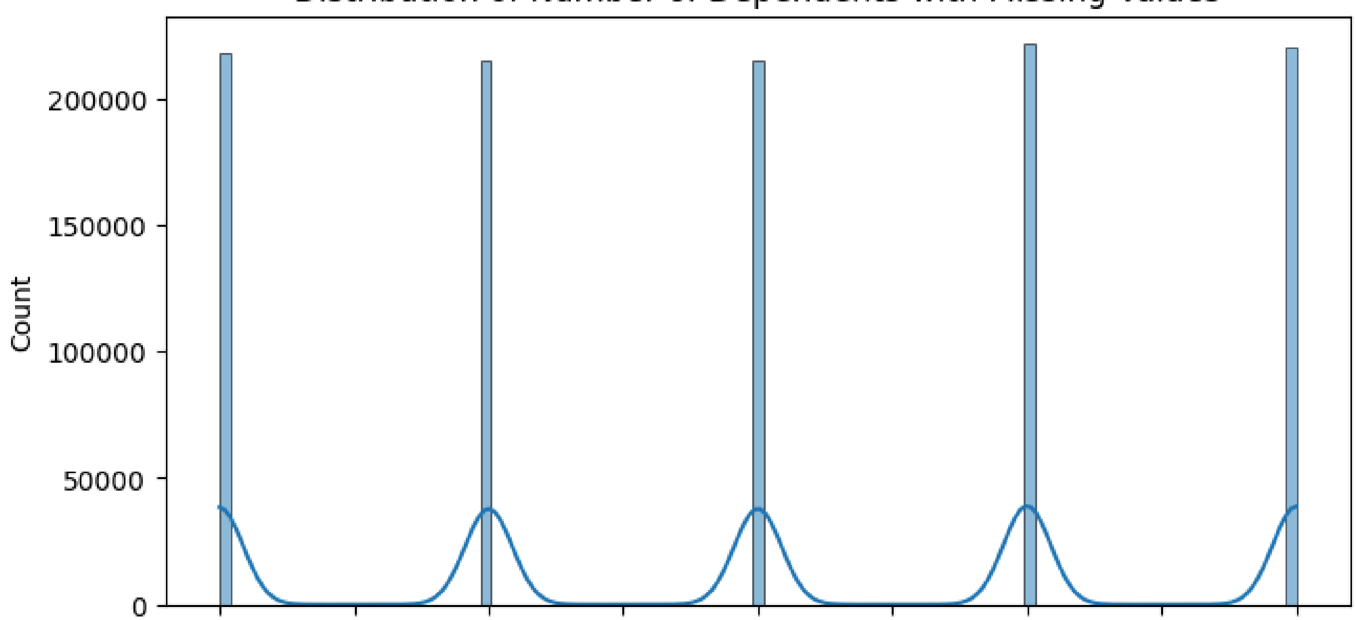

Distribution of Previous Claims with Missing Values
300000
250000
200000
150000
100000
50000
Count



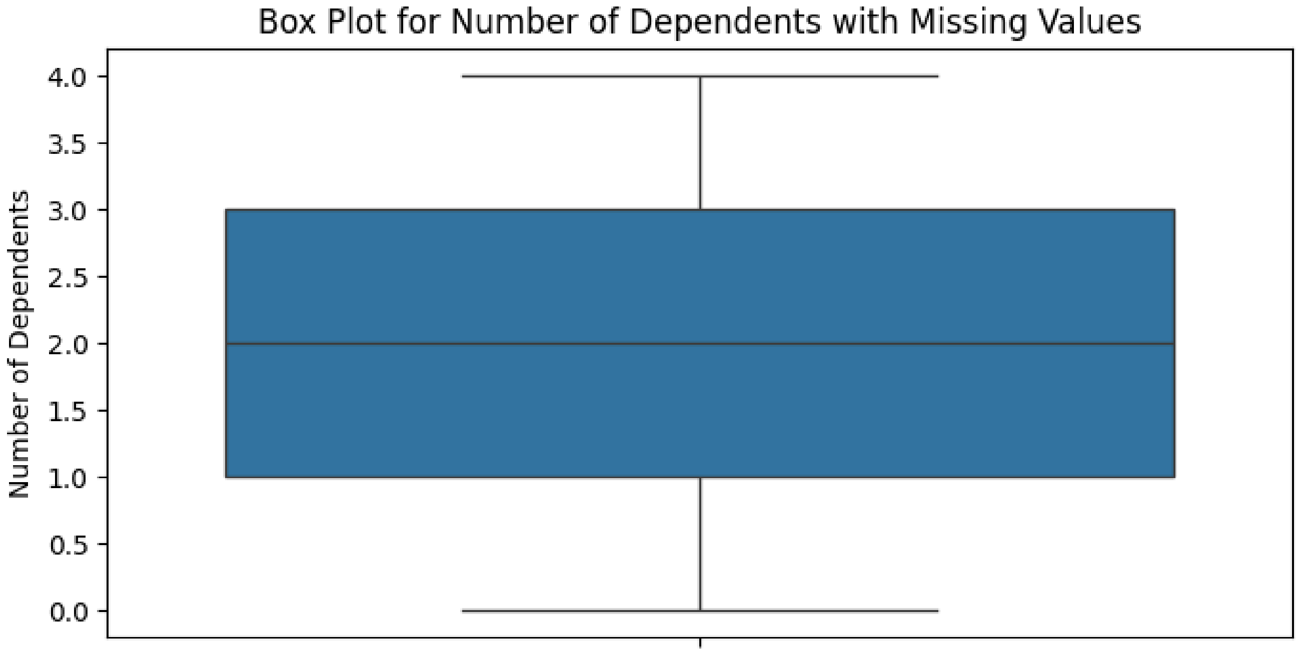
Box Plot for Previous Claims with Missing Values

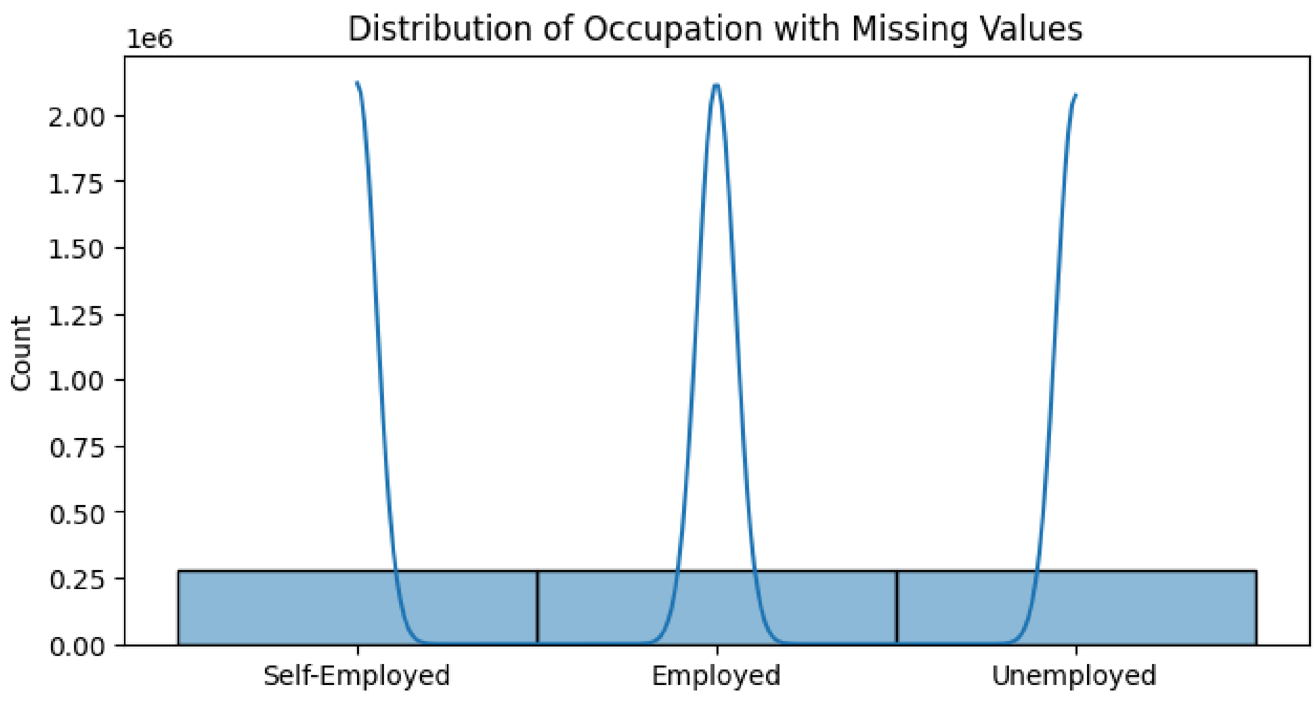


Distribution of Number of Dependents with Missing Values

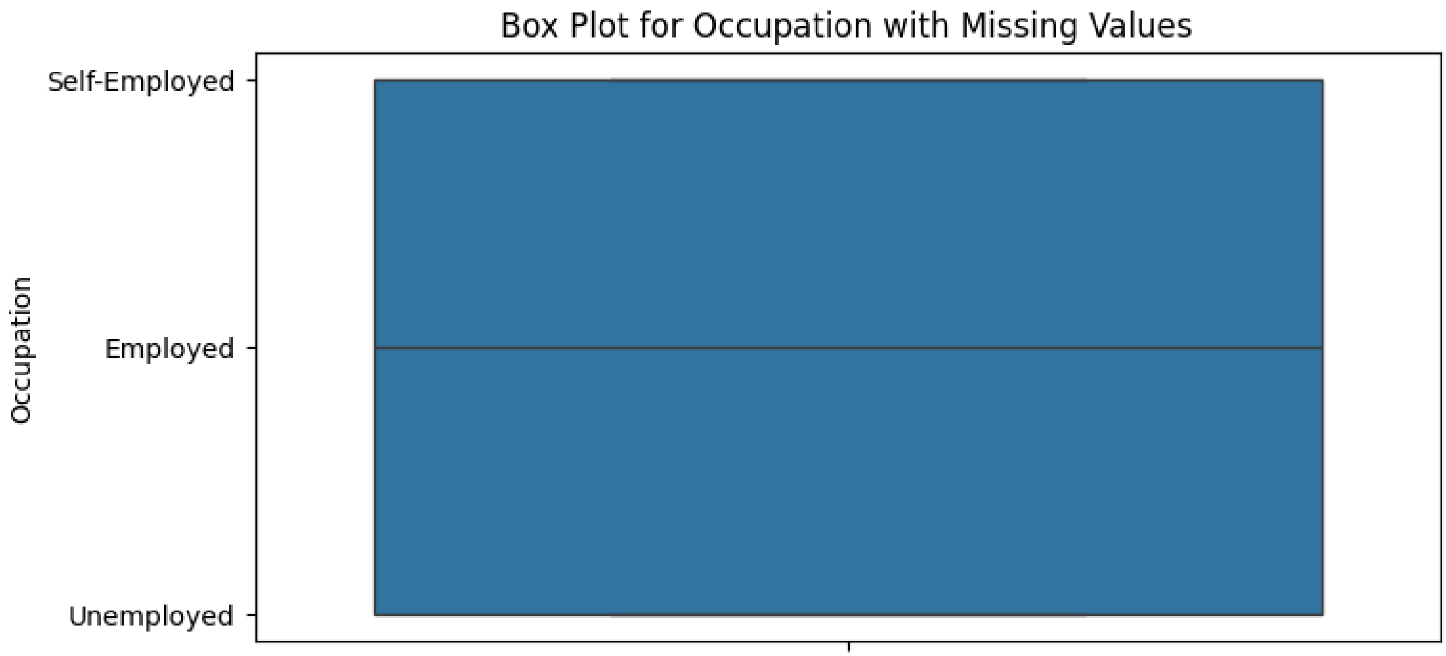



0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Number of Dependents


Box Plot for Number of Dependents with Missing Values
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Number of Dependents


Distribution of Occupation with Missing Values
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Count
Self-Employed
Employed
Unemployed

Occupation


Box Plot for Occupation with Missing Values
Self-Employed
Employed
Unemployed
Occupation

Data cleaning

Subtask:

Clean the training and testing datasets (`df_train` and `df_test`)

Reasoning: Impute missing values and handle outliers in both `df_train` and `df_test` datasets. Use the mean for numerical features and mode for categorical features for imputation. Cap outliers at the 95th percentile. Apply the same strategy to both datasets.

```
import pandas as pd
import numpy as np

# Identify numerical and categorical columns
numerical_cols = df_train.select_dtypes(include=np.number).columns
categorical_cols = df_train.select_dtypes(exclude=np.number).columns

# Impute missing numerical values with the mean
for col in numerical_cols:
    if df_train[col].isnull().any():
        mean_val = df_train[col].mean()
        df_train[col].fillna(mean_val, inplace=True)
        df_test[col].fillna(mean_val, inplace=True)

# Impute missing categorical values with the mode
for col in categorical_cols:
    if df_train[col].isnull().any():
        mode_val = df_train[col].mode()[0]
        df_train[col].fillna(mode_val, inplace=True)
        if col in df_test.columns:  # Check if the column exists in df_test
            df_test[col].fillna(mode_val, inplace=True)
```

```

# Outlier handling (capping at the 95th percentile)
for col in numerical_cols:
    if col != 'id' and col != 'Premium Amount':  # Exclude ID and target
        upper_limit = df_train[col].quantile(0.95)
        df_train[col] = np.where(df_train[col] > upper_limit, upper_limit, df_train[co
        df_test[col] = np.where(df_test[col] > upper_limit, upper_limit, df_test[col])

display(df_train.head())
display(df_test.head())
```

```
<ipython-input-4-c81664d17b69>:12: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c


  df_train[col].fillna(mean_val, inplace=True)
<ipython-input-4-c81664d17b69>:13: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c


  df_test[col].fillna(mean_val, inplace=True)
<ipython-input-4-c81664d17b69>:19: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c


  df_train[col].fillna(mode_val, inplace=True)
<ipython-input-4-c81664d17b69>:21: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c


  df_test[col].fillna(mode_val, inplace=True)
```

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation	Health Score	Location	...	Previous Claims	Vehicle Age	Cr S
0	0	19.0	Female	10049.0	Married	1.0	Bachelor's	Self-Employed	22.598761	Urban	...	2.0	17.0	372.(
1	1	39.0	Female	31678.0	Divorced	3.0	Master's	Employed	15.569731	Rural	...	1.0	12.0	694.(
2	2	23.0	Male	25602.0	Divorced	3.0	High School	Self-Employed	47.177549	Suburban	...	1.0	14.0	592.9
3	3	21.0	Male	103528.0	Married	2.0	Bachelor's	Employed	10.938144	Rural	...	1.0	0.0	367.(
4	4	21.0	Male	39651.0	Single	1.0	Bachelor's	Self-Employed	20.376094	Rural	...	0.0	8.0	598.(

5 rows × 21 columns

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation	Health Score	Location	Policy Type	Previous Claims
0	1200000	28.0	Female	2310.0	Single	4.0	Bachelor's	Self-Employed	7.657981	Rural	Basic	1.002689
1	1200001	31.0	Female	103528.0	Married	2.0	Master's	Self-Employed	13.381379	Suburban	Premium	1.002689
2	1200002	47.0	Female	17092.0	Divorced	0.0	PhD	Unemployed	24.354527	Urban	Comprehensive	1.002689
3	1200003	28.0	Female	30424.0	Divorced	3.0	PhD	Self-Employed	5.136225	Suburban	Comprehensive	1.000000
4	1200004	24.0	Male	10863.0	Divorced	2.0	High School	Unemployed	11.844155	Suburban	Premium	1.002689

Reasoning: Fix the chained assignment warnings from the previous code block and re-run the data cleaning steps.

```
import pandas as pd
import numpy as np

# Identify numerical and categorical columns
numerical_cols = df_train.select_dtypes(include=np.number).columns
categorical_cols = df_train.select_dtypes(exclude=np.number).columns

# Impute missing numerical values with the mean
for col in numerical_cols:
    if df_train[col].isnull().any():
        mean_val = df_train[col].mean()
        df_train[col] = df_train[col].fillna(mean_val)
        df_test[col] = df_test[col].fillna(mean_val)

# Impute missing categorical values with the mode
for col in categorical_cols:
    if df_train[col].isnull().any():
        mode_val = df_train[col].mode()[0]
        df_train[col] = df_train[col].fillna(mode_val)
        if col in df_test.columns:  # Check if the column exists in df_test
            df_test[col] = df_test[col].fillna(mode_val)

# Outlier handling (capping at the 95th percentile)
for col in numerical_cols:
    if col != 'id' and col != 'Premium Amount':  # Exclude ID and target
        upper_limit = df_train[col].quantile(0.95)
        df_train[col] = np.where(df_train[col] > upper_limit, upper_limit, df_train[co
        df_test[col] = np.where(df_test[col] > upper_limit, upper_limit, df_test[col])
```

```

display(df_train.head())
display(df_test.head())
```

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation	Health Score	Location	...	Previous Claims	Vehicle Age	Cr
0	0	19.0	Female	10049.0	Married	1.0	Bachelor's	Self-Employed	22.598761	Urban	...	2.0	17.0	372.(
1	1	39.0	Female	31678.0	Divorced	3.0	Master's	Employed	15.569731	Rural	...	1.0	12.0	694.(
2	2	23.0	Male	25602.0	Divorced	3.0	High School	Self-Employed	47.177549	Suburban	...	1.0	14.0	592.9
3	3	21.0	Male	103528.0	Married	2.0	Bachelor's	Employed	10.938144	Rural	...	1.0	0.0	367.(
4	4	21.0	Male	39651.0	Single	1.0	Bachelor's	Self-Employed	20.376094	Rural	...	0.0	8.0	598.(

5 rows × 21 columns

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation	Health Score	Location	Policy Type	Previous Claims
0	1200000	28.0	Female	2310.0	Single	4.0	Bachelor's	Self-Employed	7.657981	Rural	Basic	1.002689
1	1200001	31.0	Female	103528.0	Married	2.0	Master's	Self-Employed	13.381379	Suburban	Premium	1.002689
2	1200002	47.0	Female	17092.0	Divorced	0.0	PhD	Unemployed	24.354527	Urban	Comprehensive	1.002689
3	1200003	28.0	Female	30424.0	Divorced	3.0	PhD	Self-Employed	5.136225	Suburban	Comprehensive	1.000000
4	1200004	24.0	Male	10863.0	Divorced	2.0	High School	Unemployed	11.844155	Suburban	Premium	1.002689

Feature engineering

Subtask:

Engineer new features from existing ones in both the training (`df_train`) and testing (`df_test`) datasets to potentially improve model performance.

Reasoning: Apply one-hot encoding to categorical features, create interaction and polynomial features, and then scale numerical features using standardization. This addresses all parts of the subtask instructions in one go.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# 1. Categorical Encoding
categorical_cols = ['Gender', 'Marital Status', 'Education Level', 'Occupation', 'Loca
df_train = pd.get_dummies(df_train, columns=categorical_cols, drop_first=True)
df_test = pd.get_dummies(df_test, columns=categorical_cols, drop_first=True)

# Align columns in df_test with df_train
missing_cols = set(df_train.columns) - set(df_test.columns)
for c in missing_cols:
    df_test[c] = 0
df_test = df_test[df_train.columns] # Ensure same order
df_test = df_test.drop(columns=['Premium Amount'])


# 2. Interaction Features (Example: Age * Annual Income)
df_train['Age_AnnualIncome'] = df_train['Age'] * df_train['Annual Income']
df_test['Age_AnnualIncome'] = df_test['Age'] * df_test['Annual Income']

# 3. Polynomial Features (Example: Age^2)
df_train['Age_squared'] = df_train['Age'] ** 2
df_test['Age_squared'] = df_test['Age'] ** 2

```

```
# 4. Feature Scaling
numerical_cols = ['Age', 'Annual Income', 'Health Score', 'Number of Dependents', 'Pre
scaler = StandardScaler()
df_train[numerical_cols] = scaler.fit_transform(df_train[numerical_cols])
df_test[numerical_cols] = scaler.transform(df_test[numerical_cols])

display(df_train.head())
display(df_test.head())
```

Data splitting

Subtask:

Split the training data (`df_train`) into training and validation sets.

Reasoning: Split the training data into training and validation sets, stratifying by the target variable 'Premium Amount'.

```
from sklearn.model_selection import train_test_split

# Separate features (X) and target variable (y)
X = df_train.drop(columns=['Premium Amount', 'Policy Start Date', 'id'])
y = df_train['Premium Amount']

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-fe444ea34ee4> in <cell line: 0>()
      6
      7 # Split data into training and validation sets
----> 8 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

                                   3 frames

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_split.py in _iter_indices(self, X, y, groups)
   2316         class_counts = np.bincount(y_indices)
   2317         if np.min(class_counts) < 2:
-> 2318             raise ValueError(
   2319                 "The least populated class in y has only 1"
   2320                 " member, which is too few. The minimum"

ValueError: The least populated class in y has only 1 member, which is too few. The minimum number of groups for any
class cannot be less than 2.
```

Next steps: Explain error

Reasoning: The error message indicates that stratification is failing because at least one class in the target variable 'Premium Amount' has only one member. Since stratification is used to maintain the class distribution in the training and validation sets, and this is not possible with a single member class, try splitting without stratification.

```
from sklearn.model_selection import train_test_split

# Separate features (X) and target variable (y)
X = df_train.drop(columns=['Premium Amount', 'Policy Start Date', 'id'])
y = df_train['Premium Amount']

# Split data into training and validation sets without stratification
```