In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
#we will do a comphresnisve analysis on "insurance" dataset.
```

In [3]:
```
df=pd.read_csv("C:\\Users\\rautu\\OneDrive\\Desktop\\insurance dataset.csv")
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 21 columns):
 #   Column                Non-Null Count    Dtype  
---  ------                --------------    -----  
 0   id                    1048575 non-null  int64  
 1   Age                   1032254 non-null  float64
 2   Gender                1048575 non-null  object 
 3   Annual Income         1009366 non-null  float64
 4   Marital Status        1032269 non-null  object 
 5   Number of Dependents  952871 non-null   float64
 6   Education Level       1048575 non-null  object 
 7   Occupation            735638 non-null   object 
 8   Health Score          983942 non-null   float64
 9   Location              1048575 non-null  object 
 10  Policy Type           1048575 non-null  object 
 11  Previous Claims       730583 non-null   float64
 12  Vehicle Age           1048570 non-null  float64
 13  Credit Score          928164 non-null   float64
 14  Insurance Duration    1048574 non-null  float64
 15  Policy Start Date     1048575 non-null  object 
 16  Customer Feedback     980702 non-null   object 
 17  Smoking Status        1048575 non-null  object 
 18  Exercise Frequency    1048575 non-null  object 
 19  Property Type         1048575 non-null  object 
 20  Premium Amount        1048575 non-null  int64  
dtypes: float64(8), int64(2), object(11)
memory usage: 168.0+ MB
```

In [5]:
```
#STEPS I WILL FOLLOW IN THIS ANALYSIS:::
# 1. EDA/PREPROCESSING
# 2. DESCRIPTIVE ANALYTICS
# 3. DIAGNOSTIC  ANALYTICS
# 4. PREDICTIVE ANALYTICS
# 5. PRESCRIPTIVE ANALYTICS
```

DATA CLEANING/PREPROCESSING

In [7]:
```
df.describe() #the count for each features says that there are missing data.
```

Out[7]:

	id	Age	Annual Income	Number of Dependents	Health Score	Prev Cl
count	1.048575e+06	1.032254e+06	1.009366e+06	952871.000000	983942.000000	730583.00
mean	5.242870e+05	4.114057e+01	3.276203e+04	2.009702	25.610221	1.00
std	3.026977e+05	1.353652e+01	3.220290e+04	1.417424	12.203206	0.98
min	0.000000e+00	1.800000e+01	1.000000e+00	0.000000	2.012237	0.00
25%	2.621435e+05	3.000000e+01	7.991000e+03	1.000000	15.919120	0.00
50%	5.242870e+05	4.100000e+01	2.393400e+04	2.000000	24.573981	1.00
75%	7.864305e+05	5.300000e+01	4.463600e+04	3.000000	34.520860	2.00
max	1.048574e+06	6.400000e+01	1.499970e+05	4.000000	58.975914	9.00

In [8]:

```
# we are going to make a function that checks for null values and displays the tota
```

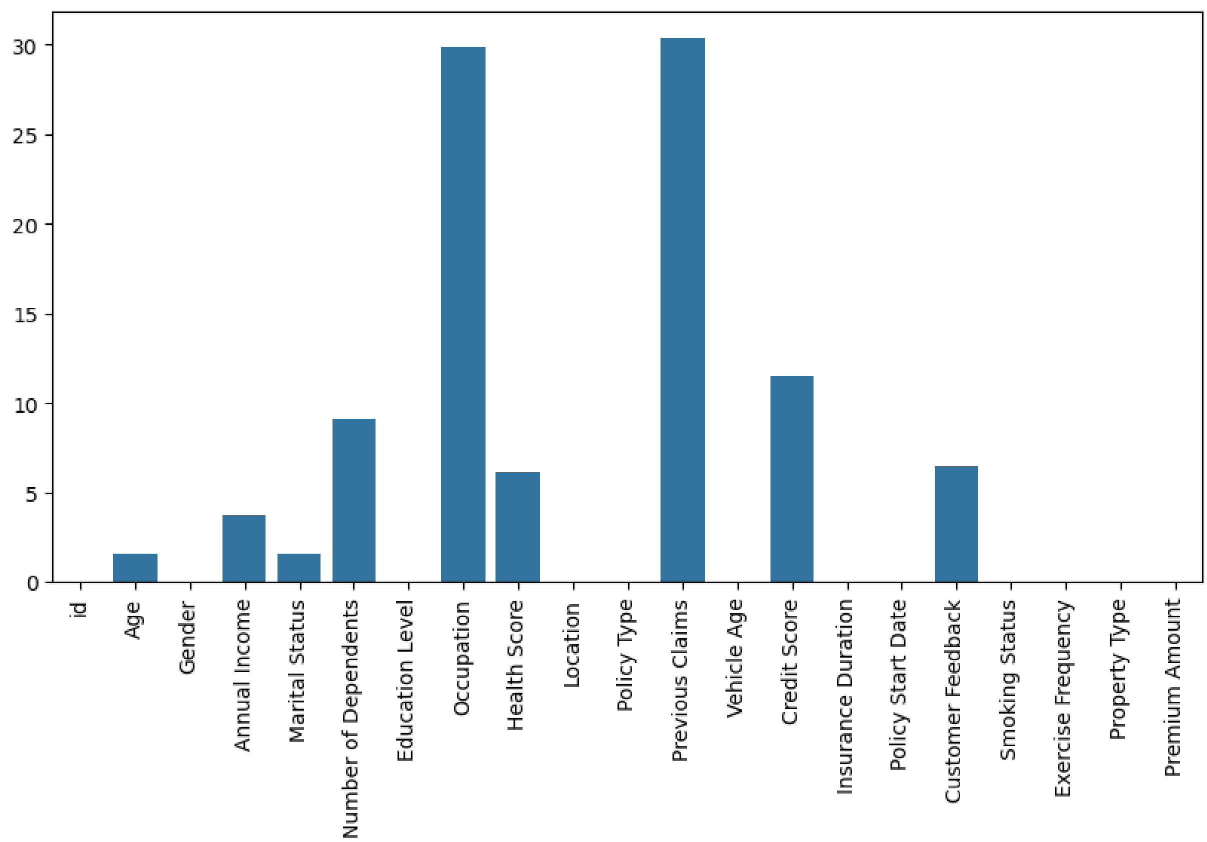
In [9]:

```
def check_null(data):
    null={}
    data=data.replace(["",""],np.nan)
    for cols in data.columns:
        total_null=data[cols].isnull().sum()
        null[cols]=(total_null/len(data))*100
    plt.figure(figsize=(10,5))
    sns.barplot(x=list(null.keys()),y=list(null.values()))
    plt.xticks(rotation=90)
    return null
```

In [10]:

```
check_null(df) #it shows almost 30% data are missing in "occupation" and "previous
               #tells me that data doesn't have entire missing rows.
               #now we are going to impute those null values and call the same func
```

Out[10]:
```
{'id': 0.0,
 'Age': 1.5564933361943591,
 'Gender': 0.0,
 'Annual Income': 3.7392651932384426,
 'Marital Status': 1.5550628233555064,
 'Number of Dependents': 9.12705338197077,
 'Education Level': 0.0,
 'Occupation': 29.84402641680376,
 'Health Score': 6.163889087571228,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 30.32610924349713,
 'Vehicle Age': 0.00047683761295090957,
 'Credit Score': 11.483298762606395,
 'Insurance Duration': 9.536752259018191e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 6.472879860763417,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0}
```



In [11]:
```
#using conditional filtering to check if we have same nan values for 4 major missin
#i am considering dropping them all. dropping less than 1% of data would not have s
#in the next cell, i will drop them.
df[(df['Occupation'].isnull())&(df['Previous Claims'].isnull())&(df['Credit Score']
```

Out[11]:

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation
678	678	62.0	Male	NaN	Single	NaN	Master's	NaN
1764	1764	64.0	Male	27539.0	Single	NaN	PhD	NaN
3501	3501	41.0	Female	2586.0	Single	NaN	PhD	NaN
3853	3853	19.0	Female	NaN	Single	NaN	Bachelor's	NaN
7578	7578	64.0	Female	4418.0	Single	NaN	PhD	NaN
...	...	...	...	...	...	...	...	...
1044141	1044141	20.0	Female	9064.0	Single	NaN	Bachelor's	NaN
1044742	1044742	31.0	Male	147539.0	Married	NaN	Master's	NaN
1044877	1044877	58.0	Female	17082.0	Divorced	NaN	Bachelor's	NaN
1048067	1048067	64.0	Female	86517.0	Married	NaN	Bachelor's	NaN
1048362	1048362	20.0	Male	132327.0	Divorced	NaN	Bachelor's	NaN

998 rows × 21 columns

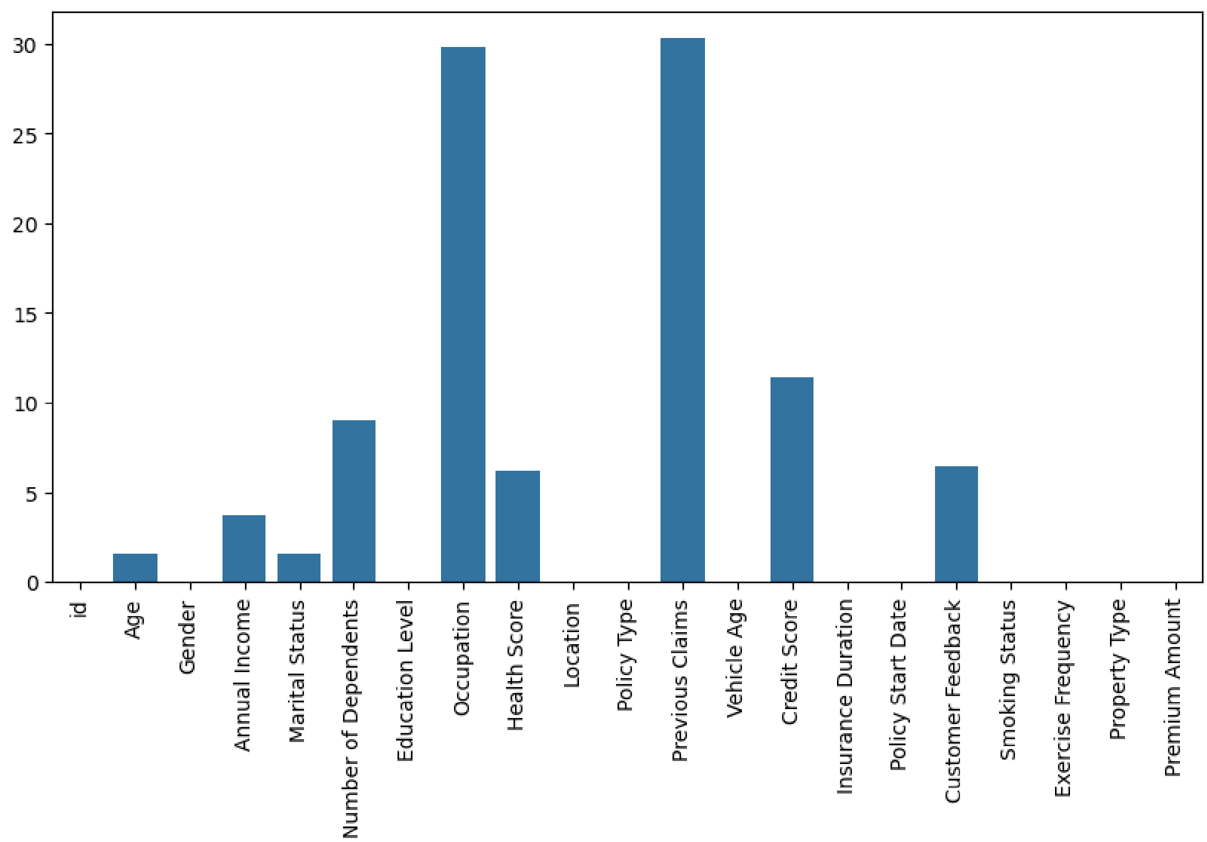
DROPPING NAN VALUES

In [13]:
```
index=df[(df['Occupation'].isnull())&(df['Previous Claims'].isnull())&(df['Credit S
```

In [14]:
```
df=df.drop(index).reset_index(drop=True) #later we will calculate the percent of dr
```

In [15]:
```
check_null(df)
#in next step, i will drop the rows with same nan values for minor columns.
```

Out[15]:
```
{'id': 0.0,
 'Age': 1.5555897084414798,
 'Gender': 0.0,
 'Annual Income': 3.739868286531682,
 'Marital Status': 1.5548260414270263,
 'Number of Dependents': 9.040481033852405,
 'Education Level': 0.0,
 'Occupation': 29.77719060269555,
 'Health Score': 6.164511057421077,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 30.259732697453266,
 'Vehicle Age': 0.0004772918840333455,
 'Credit Score': 11.398971149614779,
 'Insurance Duration': 9.545837680666911e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 6.471791572361746,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0}
```



In [16]:
```
df[(df['Age'].isnull())&(df['Marital Status'].isnull())]
```

Out[16]:

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation
1886	1888	NaN	Female	4947.0	NaN	0.0	High School	Employed
2535	2537	NaN	Female	4544.0	NaN	2.0	High School	NaN
6278	6282	NaN	Female	132579.0	NaN	2.0	High School	Employed
14416	14427	NaN	Female	18644.0	NaN	3.0	Bachelor's	NaN
19422	19437	NaN	Female	34693.0	NaN	0.0	High School	Self-Employed
...	...	...	...	...	...	...	...	...
1035796	1036783	NaN	Male	5365.0	NaN	1.0	High School	Employed
1041340	1042331	NaN	Female	138855.0	NaN	2.0	Master's	NaN
1041857	1042848	NaN	Female	1208.0	NaN	4.0	PhD	NaN
1045248	1046244	NaN	Female	24294.0	NaN	2.0	Master's	NaN
1046382	1047378	NaN	Female	54722.0	NaN	4.0	High School	Unemployed

259 rows × 21 columns

In [17]:

```
index_2=df[(df['Age'].isnull())&(df['Marital Status'].isnull())].index
```

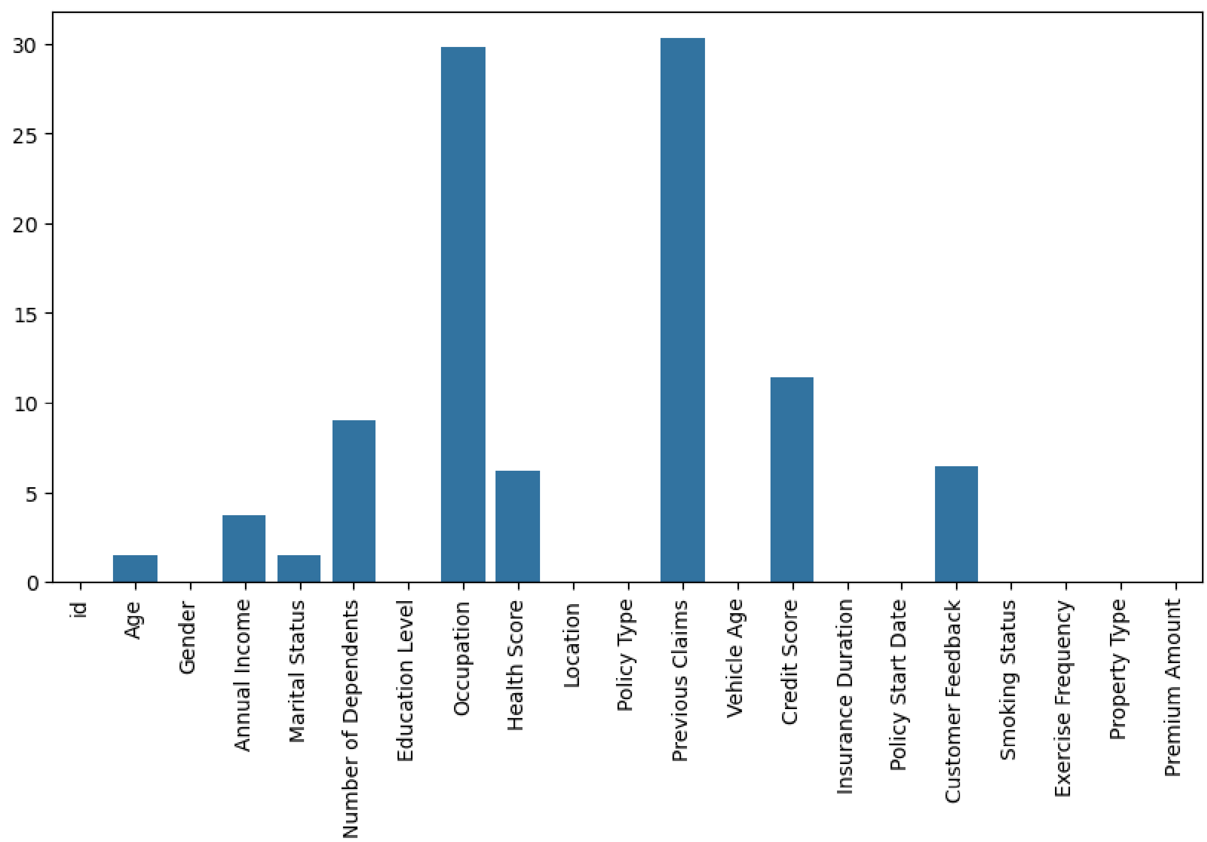
In [18]:

```
df=df.drop(index_2).reset_index(drop=True)
```

In [19]:

```
check_null(df)
```

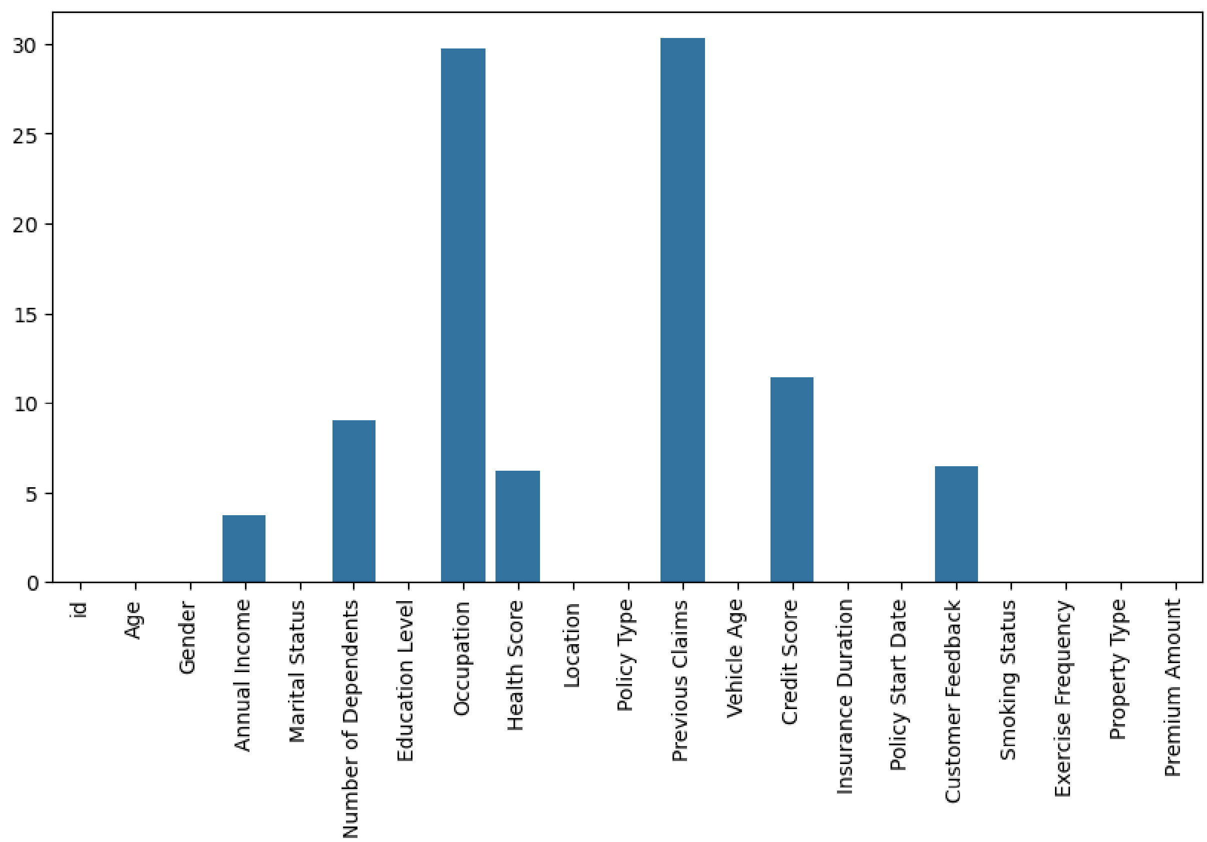
Out[19]:
```
{'id': 0.0,
 'Age': 1.5312445694621881,
 'Gender': 0.0,
 'Annual Income': 3.739551883955017,
 'Marital Status': 1.5304807135941518,
 'Number of Dependents': 9.040711608126664,
 'Education Level': 0.0,
 'Occupation': 29.777393303657533,
 'Health Score': 6.164698782986639,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 30.259672802338926,
 'Vehicle Age': 0.0004774099175226627,
 'Credit Score': 11.39835274482058,
 'Insurance Duration': 9.548198350453254e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 6.471864323920719,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0}
```



In [20]:
```
df=df.dropna(subset=['Age','Marital Status','Vehicle Age']).reset_index(drop=True)
```

In [21]:
```
check_null(df) #in next step, we will work on filling values.
```

Out[21]:
```
{'id': 0.0,
 'Age': 0.0,
 'Gender': 0.0,
 'Annual Income': 3.710919608725758,
 'Marital Status': 0.0,
 'Number of Dependents': 9.02076046518729,
 'Education Level': 0.0,
 'Occupation': 29.768716381333803,
 'Health Score': 6.190020753570313,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 30.280611516212314,
 'Vehicle Age': 0.0,
 'Credit Score': 11.38846014812159,
 'Insurance Duration': 9.849819797546804e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 6.441683649397634,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0}
```



FILLING NAN VALUES

WE WILL PERFORM EDA ON FEATURE RELATIONSHIP WHILE FILLING NAN.

trying to fill occupation value based on annual income and vice-versa

In [24]:
```
df.groupby('Occupation')['Annual Income'].describe()
#interesting that the salary of unemployed is higher than others.
#that may be the result of nan values present in this column
```

Out[24]:

Occupation	count	mean	std	min	25%	50%	75%	ma
Employed	231036.0	32665.389104	32146.664790	5.0	7982.0	23906.0	44456.0	149996.(
Self-Employed	229962.0	32716.875410	32215.924732	2.0	7984.0	23861.0	44641.0	149997.(
Unemployed	225403.0	32766.184075	32230.377323	2.0	7983.0	23947.0	44672.0	149997.(

In [25]:
```
df['Occupation'].value_counts()
```

Out[25]:
```
Occupation
Employed         239816
Self-Employed    239029
Unemployed       234176
Name: count, dtype: int64
```

In [26]:
```
df[df['Occupation'].isnull()]['Annual Income'].describe()
#the below output is fundamental for the function I am going to create.
#since there are nan present in income as well, so we will classify them as 'Unempl
```

Out[26]:
```
count    291171.000000
mean      32475.133914
std       31826.927586
min           1.000000
25%        8071.000000
50%       23891.000000
75%       44127.000000
max      149996.000000
Name: Annual Income, dtype: float64
```

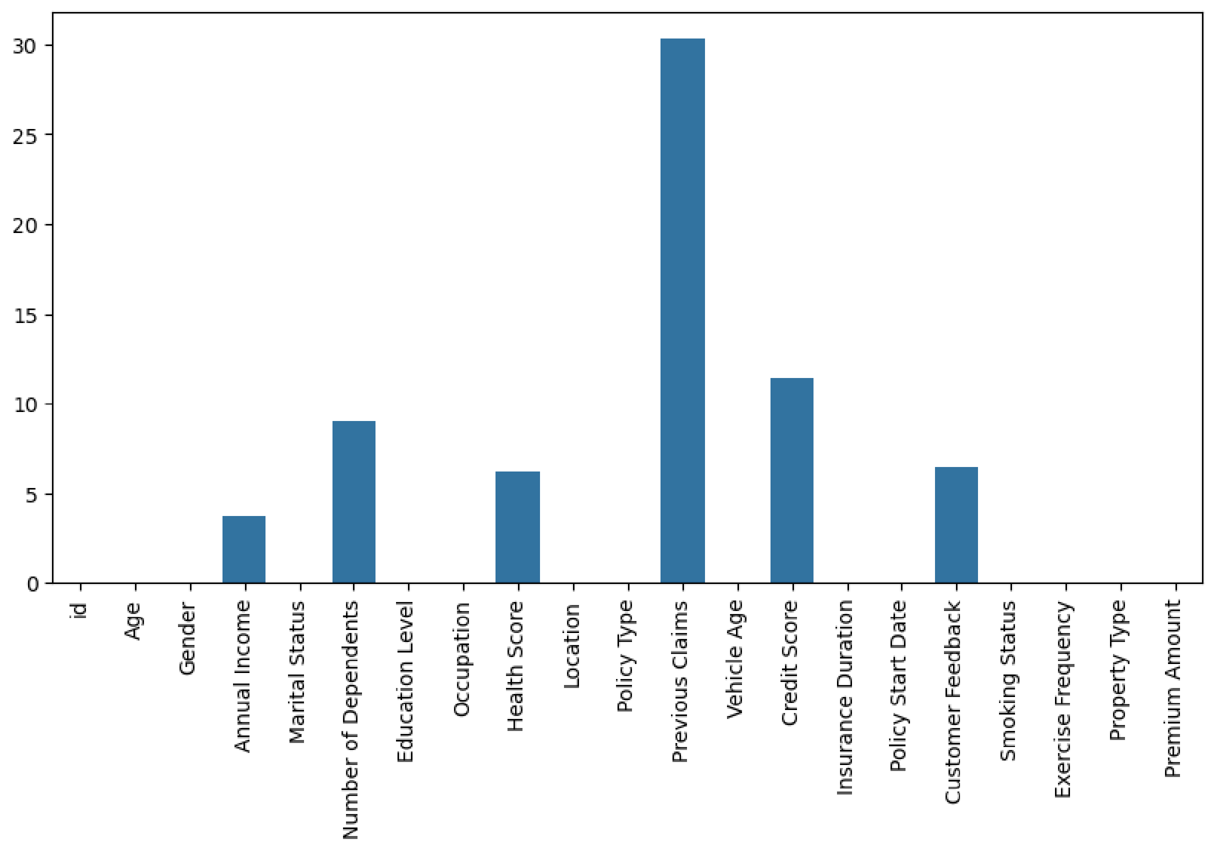
In [27]:
```
def fill_occupation(Income):
    if pd.isna(Income):
        return 'Unemployed'
    if Income<=8071:
        return 'Unemployed'
    elif Income<=44127:
        return 'Self-Employed'
    else:
        return 'Employed'
```

In [28]:
```
df['Occupation']=np.where(df['Occupation'].isna(),
                         np.vectorize(fill_occupation)(df['Annual Income']),
```

```
                     df['Occupation'])
#WE WILL CHECK THIS WTIH OUR EARLIER FUNCTION
```

In [29]:
```
check_null(df)
#since we have already dropped the rows with same nan values in both columns before
```

Out[29]:
```
{'id': 0.0,
 'Age': 0.0,
 'Gender': 0.0,
 'Annual Income': 3.710919608725758,
 'Marital Status': 0.0,
 'Number of Dependents': 9.02076046518729,
 'Education Level': 0.0,
 'Occupation': 0.0,
 'Health Score': 6.190020753570313,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 30.280611516212314,
 'Vehicle Age': 0.0,
 'Credit Score': 11.38846014812159,
 'Insurance Duration': 9.849819797546804e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 6.441683649397634,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0}
```



In [30]:
```
df.groupby('Occupation')['Annual Income'].describe()
#filling this column shows significant impact on 25 percentile.
```

Out[30]:

Occupation	count	mean	std	min	25%	50%	75%	ma
Employed	303827.0	43467.494252	36797.263557	5.0	12982.0	35667.0	64438.0	149996
Self-Employed	375518.0	29435.411197	26309.885152	2.0	12228.0	23864.0	37875.0	149997
Unemployed	298227.0	25657.660869	30705.735024	1.0	3872.0	14000.0	36992.0	149997

In [31]:
```
#now are are doing same for Annual Income
```

In [32]:
```
df[df['Annual Income'].isnull()]
#looks like we have a lot of unemployed whose income are missing.
#i am still considering droppig rows if they all share same nan values.
```

Out[32]:

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation
22	22	22.0	Male	NaN	Divorced	4.0	PhD	Unemployed
36	36	41.0	Female	NaN	Married	3.0	PhD	Self-Employed
66	67	45.0	Male	NaN	Married	3.0	High School	Self-Employed
84	86	37.0	Male	NaN	Single	1.0	Bachelor's	Unemployed
85	87	52.0	Male	NaN	Married	2.0	PhD	Unemployed
...	...	...	...	...	...	...	...	...
1015089	1048413	27.0	Female	NaN	Divorced	1.0	Master's	Self-Employed
1015131	1048456	22.0	Female	NaN	Single	0.0	Bachelor's	Employed
1015164	1048489	56.0	Female	NaN	Divorced	0.0	Bachelor's	Employed
1015186	1048511	36.0	Male	NaN	Single	4.0	PhD	Unemployed
1015232	1048559	33.0	Female	NaN	Single	1.0	PhD	Employed

37675 rows × 21 columns

In [33]:
```
index_3=df[(df['Annual Income'].isnull())&(df['Credit Score'].isnull())&(df['Health
df=df.drop(index_3).reset_index(drop=True)
```

In [34]:
```
#for filling annual income,i am following below strategies.
#25 percentile value for unemployed
#50 percentile for self-employed and employed.
```

In [35]:
```
df.loc[df['Occupation']=='Unemployed','Annual Income']
```

Out[35]:
```
8           1733.0
10          8054.0
16         28266.0
22             NaN
25         72482.0
            ...
1015201    28785.0
1015202     7058.0
1015203     5876.0
1015207     2557.0
1015214    35330.0
Name: Annual Income, Length: 318042, dtype: float64
```

In [36]:
```
q1_umemp=df.loc[df['Occupation']=='Unemployed']['Annual Income'].quantile(0.25)
```

In [37]:
```
q2_self=df.loc[df['Occupation']=='Self-Employed']['Annual Income'].quantile(0.5)
```

In [38]:
```
q2_emp=df.loc[df['Occupation']=='Employed']['Annual Income'].quantile(0.5)
```

In [39]:
```
occupation_map={'Unemployed':q1_umemp,
               'Employed':q2_emp,
               'Self-Employed': q2_self}
```

In [40]:
```
df['Annual Income']=df['Annual Income'].fillna(df['Occupation'].map(occupation_map))
```

In [41]:
```
df.dropna(subset=['Occupation', 'Annual Income'], how='all', inplace=True)
```

In [42]:
```
df['Occupation'].value_counts()
```

Out[42]:
```
Occupation
Self-Employed    384575
Unemployed       318042
Employed         312603
Name: count, dtype: int64
```

In [43]:
```
df['Annual Income'].isna
```

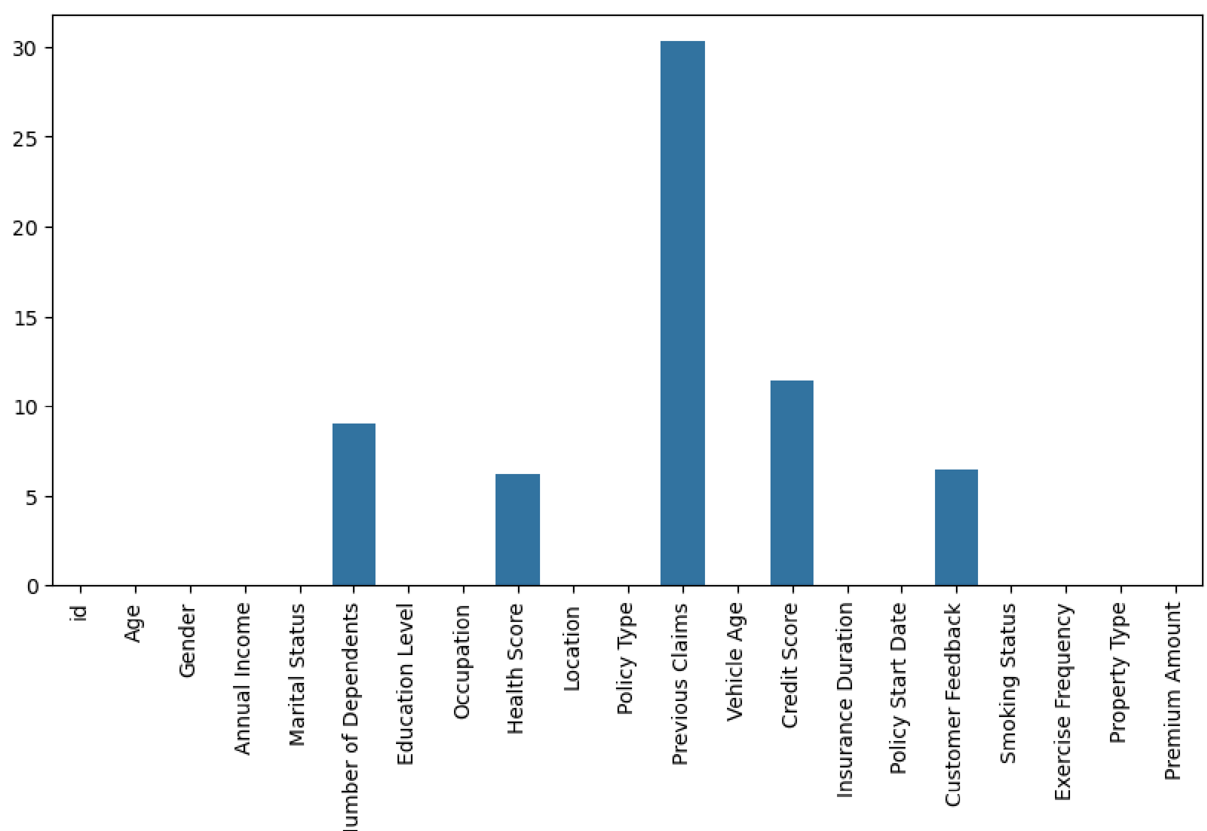
Out[43]:
```
<bound method Series.isna of 0           10049.0
1           31678.0
2           25602.0
3          141855.0
4           39651.0
             ...
1015215     39766.0
1015216     44715.0
1015217     11928.0
1015218     80123.0
1015219     27048.0
Name: Annual Income, Length: 1015220, dtype: float64>
```

In [44]:
```
index_4=df[df['Occupation'] == 'nan'].index
```

In [45]:
```
df=df.drop(index_4).reset_index(drop=True)
#dropping those with same nan between annual income and occupation
```

In [46]:
```
check_null(df)
```

Out[46]:
```
{'id': 0.0,
 'Age': 0.0,
 'Gender': 0.0,
 'Annual Income': 0.0,
 'Marital Status': 0.0,
 'Number of Dependents': 9.020704871850436,
 'Education Level': 0.0,
 'Occupation': 0.0,
 'Health Score': 6.1875258564646085,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 30.280628829219282,
 'Vehicle Age': 0.0,
 'Credit Score': 11.386103504659088,
 'Insurance Duration': 9.850081755678572e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 6.439195445322196,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0}
```



In [47]:
```
#now filling number of dependents based on Maritial Status
df.groupby('Marital Status')['Number of Dependents'].describe()
```

Out[47]:

Marital Status	count	mean	std	min	25%	50%	75%	max
Divorced	306213.0	2.010238	1.416649	0.0	1.0	2.0	3.0	4.0
Married	307718.0	2.012138	1.416502	0.0	1.0	2.0	3.0	4.0
Single	309709.0	2.007688	1.418593	0.0	1.0	2.0	3.0	4.0

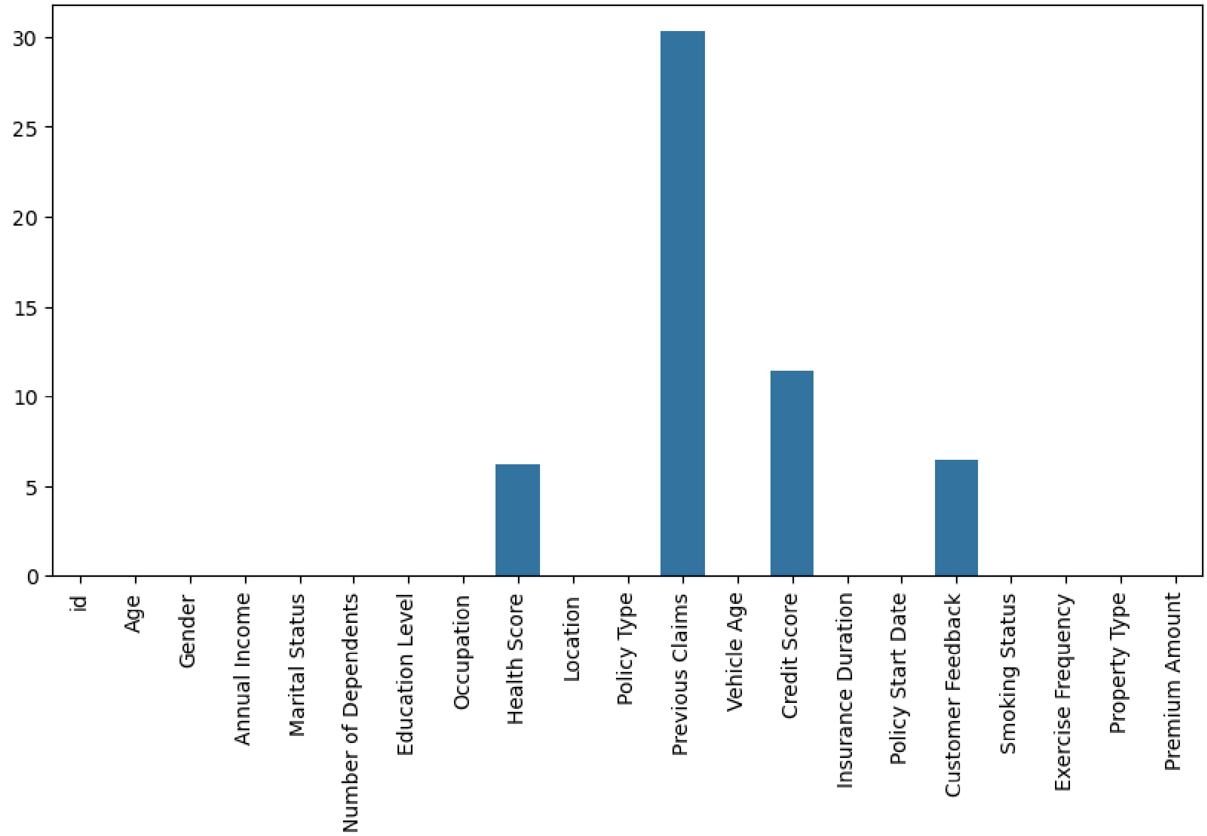
In [48]:
```
df.loc[(df['Number of Dependents'].isna()) & (df['Age'] <= 20), 'Number of Dependen

#for this specific condition, i would like to put 0 for number of dependents.
#for the rest, i will be using the same idea to impute values.
```

In [49]:
```
df.loc[(df['Number of Dependents'].isna()) & (df['Age'].between(21,30)), 'Number of
```

In [50]:
```
df.loc[(df['Number of Dependents'].isna()) & (df['Age'] > 30), 'Number of Dependent
```

In [51]:
```
check_null(df)
```

Out[51]:
```
{'id': 0.0,
 'Age': 0.0,
 'Gender': 0.0,
 'Annual Income': 0.0,
 'Marital Status': 0.0,
 'Number of Dependents': 0.0,
 'Education Level': 0.0,
 'Occupation': 0.0,
 'Health Score': 6.1875258564646085,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 30.280628829219282,
 'Vehicle Age': 0.0,
 'Credit Score': 11.386103504659088,
 'Insurance Duration': 9.850081755678572e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 6.439195445322196,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0}
```



In [52]:
```
#in next step, i will impute health score based on smoking and exercise frequency
```

In [53]:
```
df.groupby('Exercise Frequency')['Health Score'].describe()
# this shows that regardless of exercise frequency, the mean health score is same
#we will check if there is any relatioship with smoking pattern.
```

Out[53]:

Exercise Frequency	count	mean	std	min	25%	50%	75%	
Daily	233214.0	25.631039	12.181886	2.024415	15.936263	24.592459	34.547187	58.8
Monthly	238428.0	25.563935	12.214367	2.064241	15.924942	24.482855	34.479105	57.5
Rarely	237587.0	25.616466	12.194310	2.012237	15.918658	24.582641	34.496852	57.9
Weekly	243174.0	25.598099	12.203136	2.053458	15.879229	24.582544	34.480114	58.9

In [54]:
```
df['Smoking Status'].value_counts()
```

Out[54]:
```
Smoking Status
Yes    508998
No     506222
Name: count, dtype: int64
```

In [55]:
```
df.groupby('Smoking Status')['Health Score'].describe()
#Again it seems that the health score is same for smoking status as well.
#now, i am filling all those nan with its mean value.
```

Out[55]:

Smoking Status	count	mean	std	min	25%	50%	75%	ı
No	475213.0	25.623937	12.208699	2.024415	15.933706	24.566223	34.547667	58.975
Yes	477190.0	25.580541	12.188417	2.012237	15.895357	24.566341	34.452887	57.988

In [56]:
```
df['Health Score']=df['Health Score'].fillna(value=df['Health Score'].mean())
```

In [57]:
```
#in the next step, i will fill "previous claims" based on the policy start date.
```

In [58]:
```
df['Policy Start Date'].value_counts()
```

Out[58]:
```
Policy Start Date
21:39.2    494964
21:39.1    359665
21:39.3    160591
Name: count, dtype: int64
```

In [59]:
```
df['Policy Start Date'] = pd.to_datetime(df['Policy Start Date'], errors='coerce',u
```

```
C:\Users\rautu\AppData\Local\Temp\ipykernel_13192\3231855590.py:1: UserWarning: Coul
d not infer format, so each element will be parsed individually, falling back to `da
teutil`. To ensure parsing is consistent and as-expected, please specify a format.
  df['Policy Start Date'] = pd.to_datetime(df['Policy Start Date'], errors='coerce',
utc=True)
```

In [60]:
```
df['Policy Start Date']=df['Policy Start Date'].dt.time
```

In [61]:
```
df['Previous Claims'].corr(df['Gender'].map({'Male':0,'Female':1}))
```

Out[61]:
```
-8.73237127822894e-06
```

In [62]:
```
df.groupby('Insurance Duration')['Previous Claims'].describe()
```

Out[62]:

Insurance Duration	count	mean	std	min	25%	50%	75%	max
1.0	79718.0	0.997755	0.982521	0.0	0.0	1.0	2.0	8.0
2.0	77351.0	0.992618	0.974774	0.0	0.0	1.0	2.0	8.0
3.0	77988.0	1.000705	0.985091	0.0	0.0	1.0	2.0	8.0
4.0	78004.0	0.998154	0.975632	0.0	0.0	1.0	2.0	7.0
5.0	77920.0	0.998024	0.984731	0.0	0.0	1.0	2.0	8.0
6.0	77926.0	1.010856	0.988148	0.0	0.0	1.0	2.0	7.0
7.0	78710.0	1.009389	0.986435	0.0	0.0	1.0	2.0	7.0
8.0	79043.0	1.005642	0.983296	0.0	0.0	1.0	2.0	7.0
9.0	81144.0	0.998620	0.976902	0.0	0.0	1.0	2.0	9.0

In [63]:

```
df.select_dtypes(include='number').corrwith(df['Previous Claims'])
#since ther are no strong correlation between other features. i have decided to imp
```

Out[63]:

```
id                      0.000045
Age                     0.002193
Annual Income           0.038894
Number of Dependents   -0.004346
Health Score            0.002209
Previous Claims         1.000000
Vehicle Age            -0.000694
Credit Score            0.036761
Insurance Duration      0.003140
Premium Amount          0.047976
dtype: float64
```

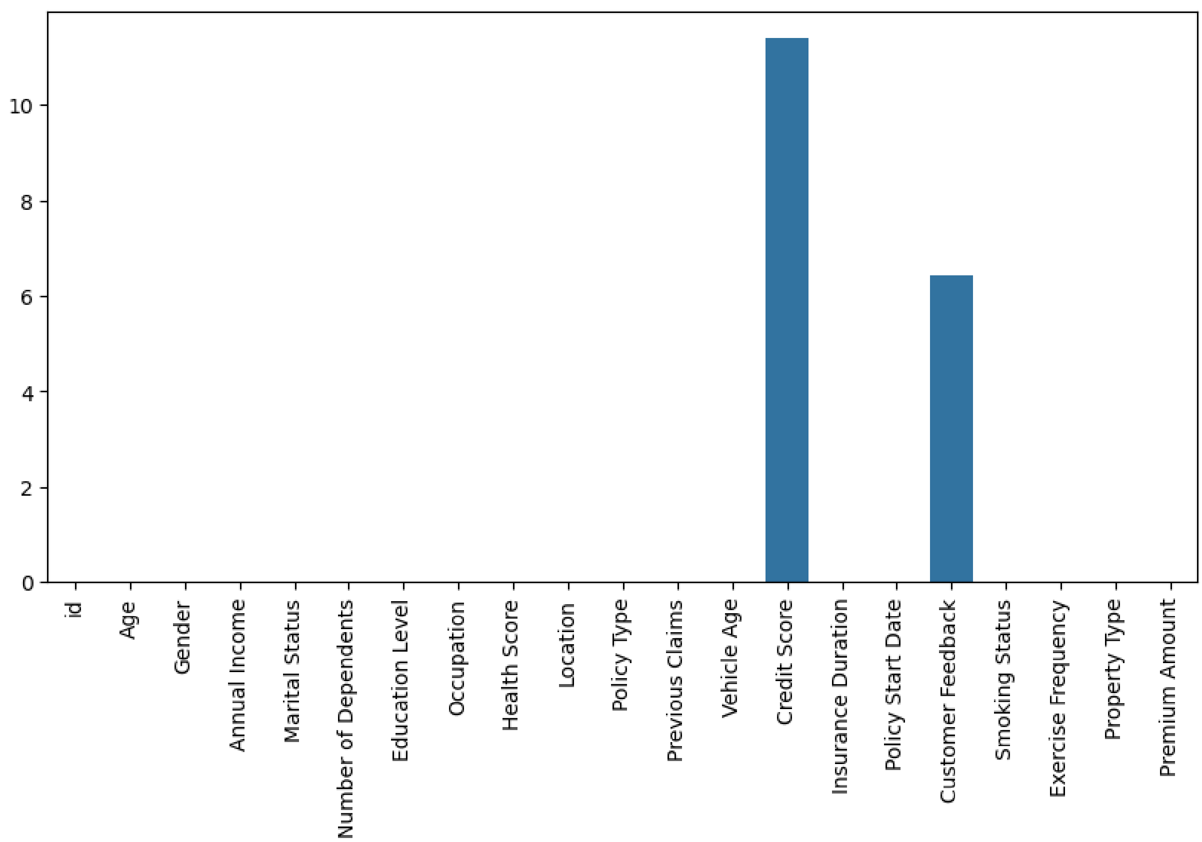
In [64]:

```
df['Previous Claims']=df['Previous Claims'].fillna(df['Previous Claims'].median())
```

In [65]:

```
check_null(df)
```

Out[65]:
```
{'id': 0.0,
 'Age': 0.0,
 'Gender': 0.0,
 'Annual Income': 0.0,
 'Marital Status': 0.0,
 'Number of Dependents': 0.0,
 'Education Level': 0.0,
 'Occupation': 0.0,
 'Health Score': 0.0,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 0.0,
 'Vehicle Age': 0.0,
 'Credit Score': 11.386103504659088,
 'Insurance Duration': 9.850081755678572e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 6.439195445322196,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0}
```



In [66]:
```
#quick correlation check with numeric features.
df.select_dtypes(include='number').corrwith(df['Credit Score'])
#though negatively correlated with annual income, i will use this column as a basel
#i will do this using binnig approach where i will impute the median credit value b
```

Out[66]:
```
id                        0.001355
Age                       0.003555
Annual Income            -0.185481
Number of Dependents     -0.002711
Health Score              0.011990
Previous Claims           0.030592
Vehicle Age               0.000694
Credit Score              1.000000
Insurance Duration        0.000772
Premium Amount           -0.027711
dtype: float64
```

In [188…
```
df[df['Credit Score'].isnull()]['Annual Income'].describe()
```

Out[188…
```
count    107711.000000
mean      27350.919665
std       26855.383669
min          11.000000
25%        7857.000000
50%       20287.000000
75%       37282.000000
max      149995.000000
Name: Annual Income, dtype: float64
```

In [196…
```
bins = [0, 10000, 30000, 60000, 100000, np.inf]
labels = ['Very Low', 'Low', 'Medium', 'High', 'Very High']
df['Income Bin']=pd.cut(df['Annual Income'],bins, labels)
```

In [210…
```
median_value=df.groupby('Income Bin')['Credit Score'].median()
```

```
C:\Users\rautu\AppData\Local\Temp\ipykernel_13192\2152537781.py:1: FutureWarning: Th
e default of observed=False is deprecated and will be changed to True in a future ve
rsion of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  median_value=df.groupby('Income Bin')['Credit Score'].median()
```

In [212…
```
df['Credit Score']=df['Credit Score'].fillna(df['Income Bin'].map(median_value))
```

In [198…
```
df['Income Bin']
```

Out[198…
```
0            (10000.0, 30000.0]
1            (30000.0, 60000.0]
2            (10000.0, 30000.0]
3               (100000.0, inf]
4            (30000.0, 60000.0]
                    ...
1007332      (30000.0, 60000.0]
1007333      (30000.0, 60000.0]
1007334      (10000.0, 30000.0]
1007335     (60000.0, 100000.0]
1007336      (10000.0, 30000.0]
Name: Income Bin, Length: 1007337, dtype: category
Categories (5, interval[float64, right]): [(0.0, 10000.0] < (10000.0, 30000.0] <
(30000.0, 60000.0] < (60000.0, 100000.0] < (100000.0, inf]]
```

In [142…
```
df['Customer Feedback'].value_counts()
```

Out[142…
```
Customer Feedback
Average    319862
Poor       317954
Good       312032
Name: count, dtype: int64
```

Doing Correlation check with numeric features and Chi-Squared test with categorical features for 'customer feedback'

In [149…
```
df.select_dtypes(include='number').corrwith(df['Customer Feedback'].map({'Poor':0,'
#shows extreme weak correlation with numeric features
```

Out[149…
```
id                        0.000609
Age                       0.001167
Annual Income             0.001059
Number of Dependents      0.000556
Health Score              0.001598
Previous Claims           0.001782
Vehicle Age              -0.001083
Credit Score             -0.001688
Insurance Duration       -0.000759
Premium Amount           -0.001474
dtype: float64
```

In [153…
```
from scipy.stats import chi2_contingency
```

In [166…
```
for col in df.select_dtypes(include='object'):
    if col!='Customer Feedback':
        crosstab=pd.crosstab(df[col],df['Customer Feedback'])
        chi2,P,dof,exp=chi2_contingency(crosstab)
        print(f"{col}: chi2={chi2}, P_value={P}")
#shows that EXERCISE FREQUENCY AND PROPERTY TYPE ARE SIGNIFICANT TO THIS FEATURE.
```

```
Gender: chi2=3.203317562043628, P_value=0.20156189349334974
Marital Status: chi2=8.572407158097867, P_value=0.07272258730566684
Education Level: chi2=4.074168939688087, P_value=0.6663988814247323
Occupation: chi2=3.825189613207832, P_value=0.43018048034597667
Location: chi2=6.923264641262666, P_value=0.13999933163991324
Policy Type: chi2=1.5775887561679696, P_value=0.8128143746621408
Policy Start Date: chi2=3.7216375043004604, P_value=0.44498669714794326
Smoking Status: chi2=1.1196120441980522, P_value=0.5713198765312958
Exercise Frequency: chi2=30.123839688327806, P_value=3.7233716830366516e-05
Property Type: chi2=11.899945866168412, P_value=0.018111011273456105
```

In [67]:
```
index_6=df[(df['Customer Feedback'].isna())&(df['Credit Score'].isna())].index
```

In [70]:
```
#i am dropping those rows with same missing data between these columns.
df=df.drop(index_6).reset_index(drop=True)
```

In [218…

```
pd.crosstab(df['Customer Feedback'], df['Exercise Frequency'])
#since the count for customer feedback in all groups of exercise frequency are quit
#so i will do the random sampling from overall CUSTOMER FEEDBACK data to fill the n
```

Out[218…

Exercise Frequency	Daily	Monthly	Rarely	Weekly
Customer Feedback				
Average	78331	80279	79595	81657
Good	76958	76988	78109	79977
Poor	77532	80208	79367	80847

In [242…

```
# Get observed distribution
probs = df['Customer Feedback'].value_counts(normalize=True)
```

In [244…

```
probs
```

Out[244…

```
Customer Feedback
Average    0.336751
Poor       0.334742
Good       0.328507
Name: proportion, dtype: float64
```

In [263…

```
index_7=df[df['Customer Feedback'].isna()].index
```

In [275…

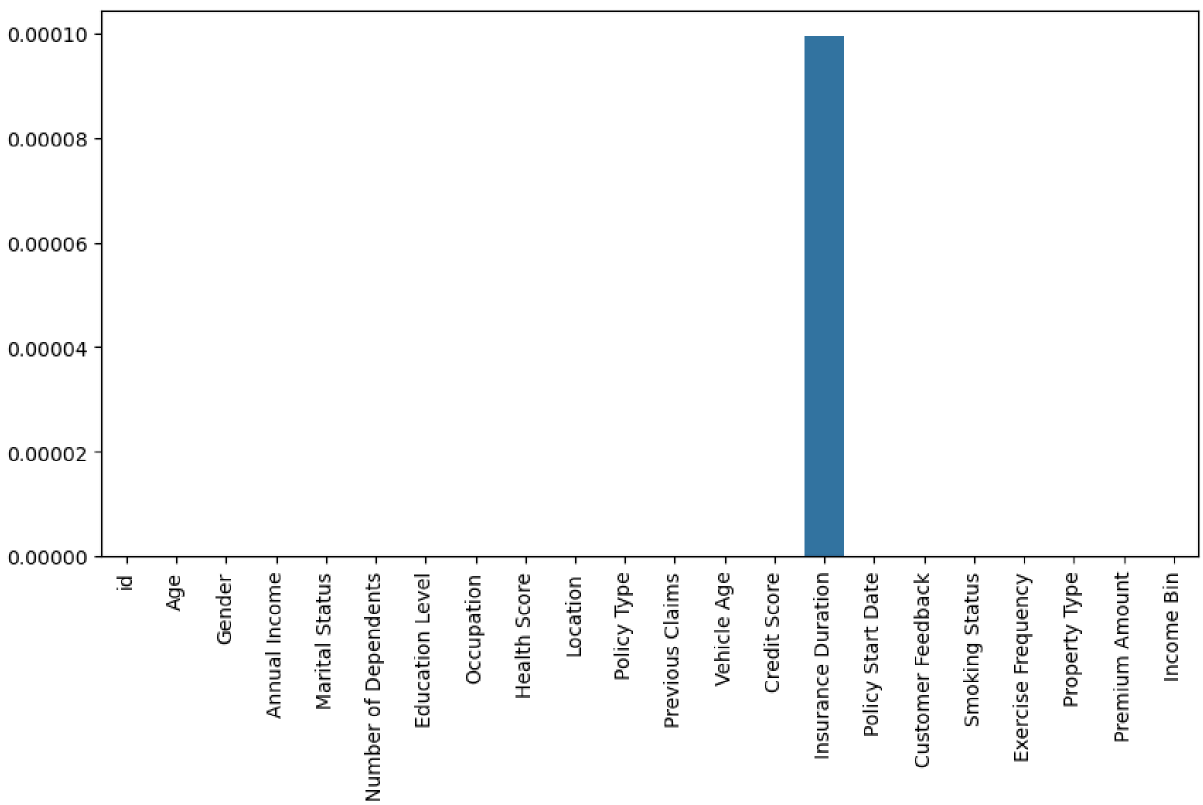
```
df.loc[index_7,'Customer Feedback']= np.random.choice(
    probs.index,
    size=len(index_7),
    p=probs.values
)
```

In [281…

```
check_null(df)
```

Out[281…

```
{'id': 0.0,
 'Age': 0.0,
 'Gender': 0.0,
 'Annual Income': 0.0,
 'Marital Status': 0.0,
 'Number of Dependents': 0.0,
 'Education Level': 0.0,
 'Occupation': 0.0,
 'Health Score': 0.0,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 0.0,
 'Vehicle Age': 0.0,
 'Credit Score': 0.0,
 'Insurance Duration': 9.927164394835096e-05,
 'Policy Start Date': 0.0,
 'Customer Feedback': 0.0,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0,
 'Income Bin': 0.0}
```



In [283…

```
df['Insurance Duration'].value_counts()
```

Out[283...

```
Insurance Duration
9.0    115650
1.0    113084
8.0    112483
7.0    112219
4.0    111096
6.0    111072
5.0    110938
3.0    110771
2.0    110023
Name: count, dtype: int64
```

In [279...

```
df[df['Insurance Duration'].isnull()]
```

Out[279...

	id	Age	Gender	Annual Income	Marital Status	Number of Dependents	Education Level	Occupation	H
683376	711358	64.0	Male	30206.0	Married	3.0	Master's	Employed	49.5:

1 rows × 22 columns

In [285...

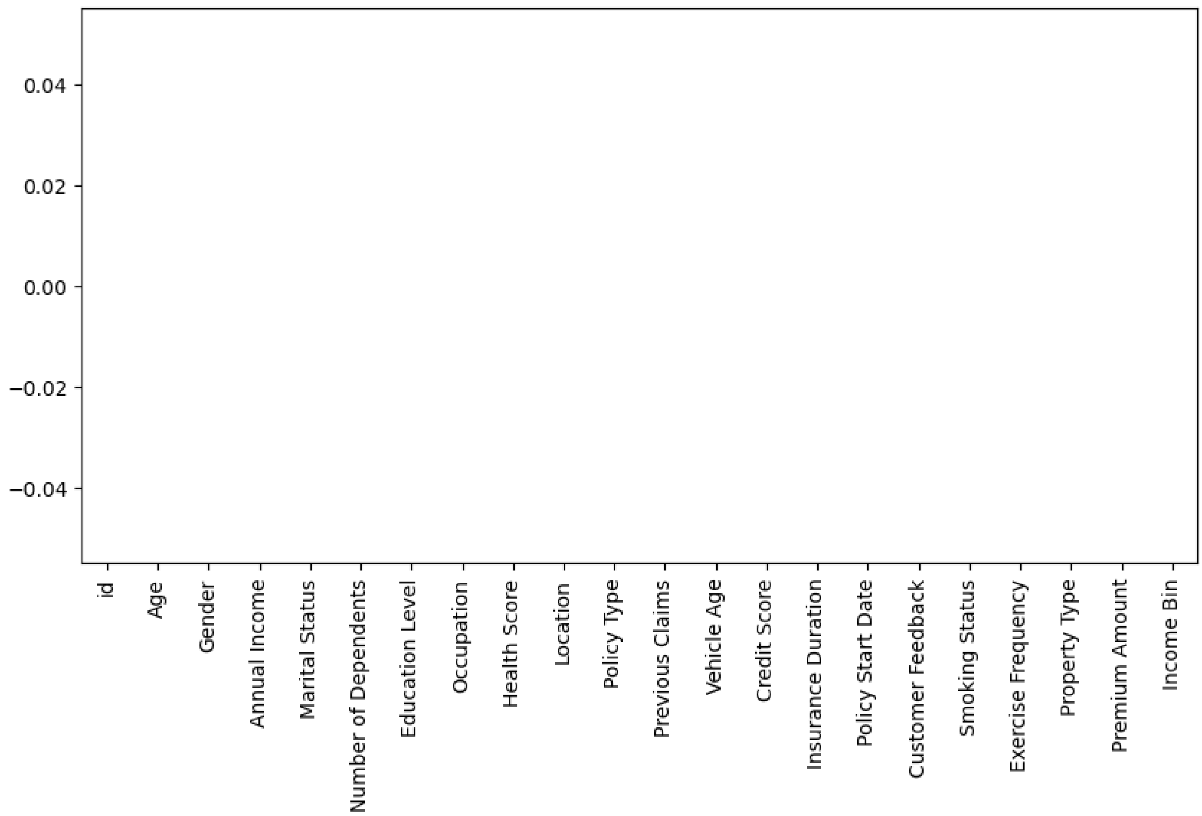
```
df.at[683376,'Insurance Duration']=5
```

In [287...

```
check_null(df)
#we have finally finished preprocessing step.
```

Out[287...

```
{'id': 0.0,
 'Age': 0.0,
 'Gender': 0.0,
 'Annual Income': 0.0,
 'Marital Status': 0.0,
 'Number of Dependents': 0.0,
 'Education Level': 0.0,
 'Occupation': 0.0,
 'Health Score': 0.0,
 'Location': 0.0,
 'Policy Type': 0.0,
 'Previous Claims': 0.0,
 'Vehicle Age': 0.0,
 'Credit Score': 0.0,
 'Insurance Duration': 0.0,
 'Policy Start Date': 0.0,
 'Customer Feedback': 0.0,
 'Smoking Status': 0.0,
 'Exercise Frequency': 0.0,
 'Property Type': 0.0,
 'Premium Amount': 0.0,
 'Income Bin': 0.0}
```



SO FAR HAVE COMPLETED DATA PREPROCESSING WITH EDA...

ALONG THE WAY, WE HAVE USED SEVERAL STRATEGIES TO FILL THE MISSING VALUES.

WE HAVE ALSO CHECKED THE CORRELATION BETWEEN THE FEATURES AND PERFORM HYPOTHESIS TESTING USING CHI SQUARED TEST.

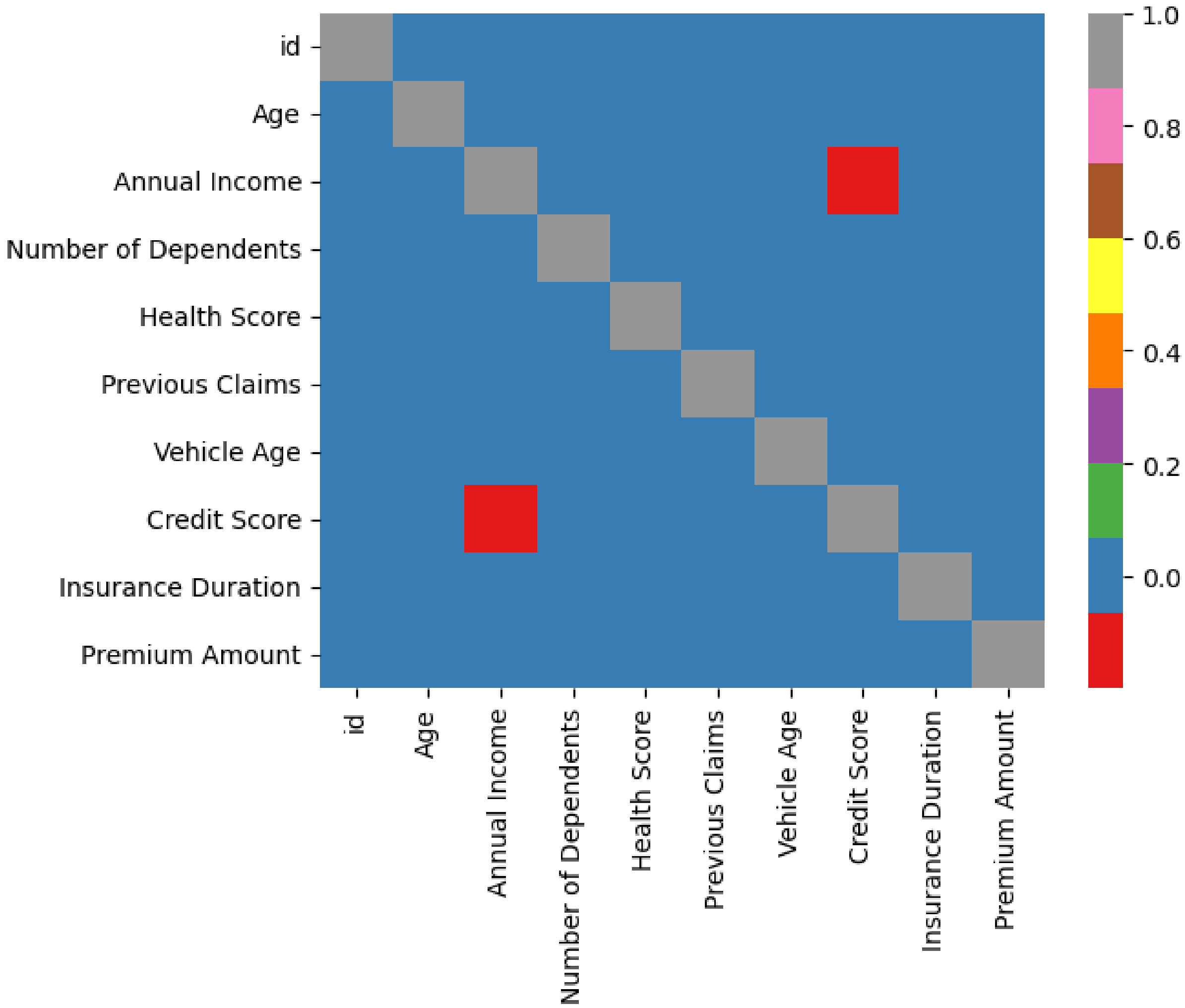
IN THE CELL BELOW WE WILL FOCUS ON MORE DETAIL RELATIONSHIP BETWEEN FEATURES.

In [681…
```
numeric_data=df.select_dtypes(include='number')
```

In [683…
```
cat_data=df.select_dtypes(include='object')
```

In [685…
```
sns.heatmap(data=numeric_data.corr(),cmap='Set1')
#it shows that there are no strong correlation between features.
# in fact those red distinct block represents some negative correlation.
```

Out[685…
```
<Axes: >
```



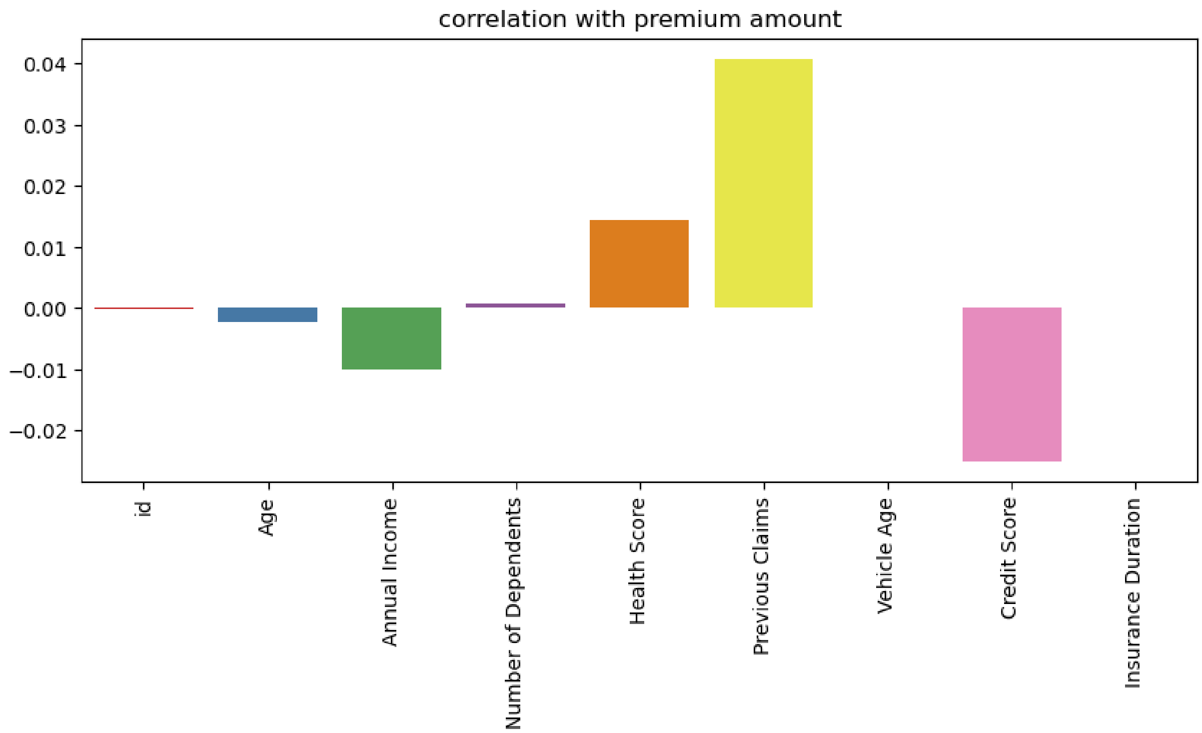
In [689…

```
#checking if there is any correlation with our label "premium amount"
plt.figure(figsize=(10,4))
sns.barplot(data=numeric_data.corrwith(numeric_data['Premium Amount'].transpose())[
plt.xticks(rotation=90);
plt.title('correlation with premium amount')
```

```
C:\Users\rautu\AppData\Local\Temp\ipykernel_13192\514606500.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=numeric_data.corrwith(numeric_data['Premium Amount'].transpose())
[:-1],palette='Set1')
```

Out[689…

```
Text(0.5, 1.0, 'correlation with premium amount')
```



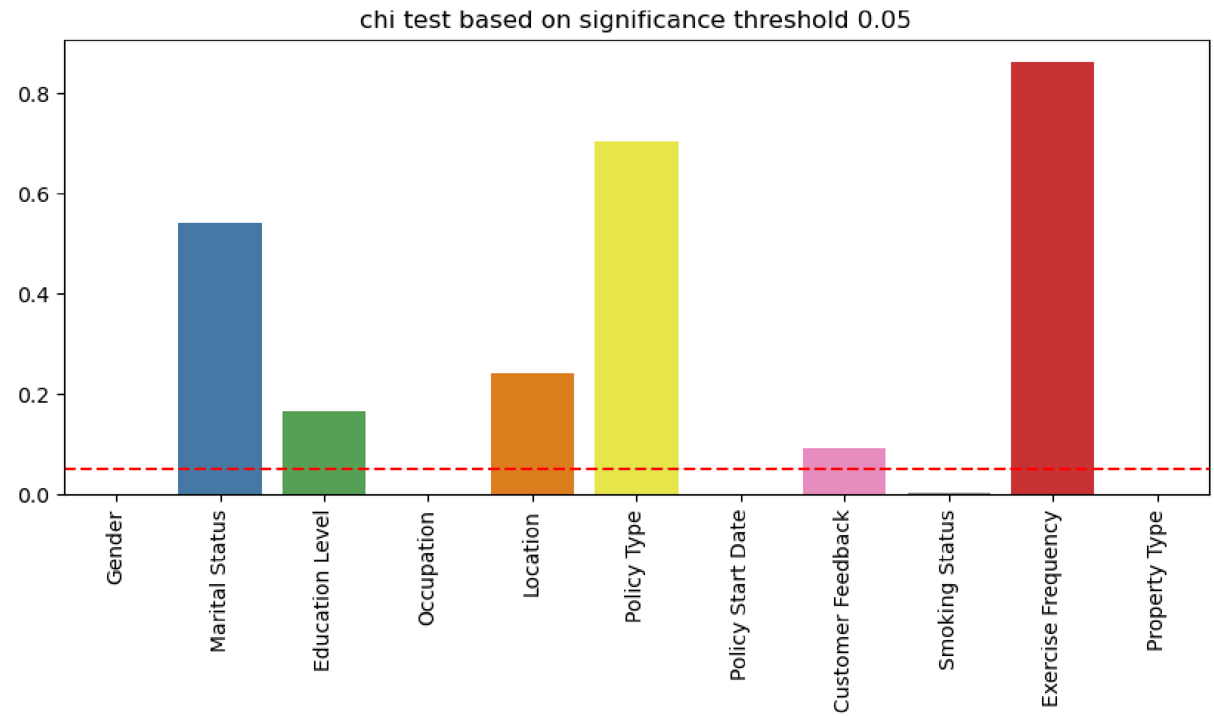
In [691…

```
#WE WILL ALSO PERFORM CHI-SQUARED TEST TO FIND ANY RELATIONSHIP BETWEEN INSURANCE P
# since we need all categorical features for this test, so we are going to create b
numeric_data['Premium Bin']=pd.qcut(numeric_data['Premium Amount'],q=5,duplicates='
chi={}
for cols in cat_data.columns:
    table_2=pd.crosstab(cat_data[cols],numeric_data['Premium Bin'])
    chi2,P,dof,ex=chi2_contingency(table_2)
    chi[cols]=P
plt.figure(figsize=(10,4))
sns.barplot(x=chi.keys(),y=chi.values(),palette='Set1')
plt.xticks(rotation=90)
plt.axhline(y=0.05,ls="--",color='red')
plt.title('chi test based on significance threshold 0.05')
chi
```

```
C:\Users\rautu\AppData\Local\Temp\ipykernel_13192\725875300.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=chi.keys(),y=chi.values(),palette='Set1')
```

Out[691…

```
{'Gender': 0.0003185899624851237,
 'Marital Status': 0.5432534388444026,
 'Education Level': 0.16425299004825217,
 'Occupation': 0.0,
 'Location': 0.2427720728781098,
 'Policy Type': 0.7038724835961978,
 'Policy Start Date': 7.207644933596629e-27,
 'Customer Feedback': 0.09076314383207312,
 'Smoking Status': 0.0028405237067245705,
 'Exercise Frequency': 0.8632641645859609,
 'Property Type': 0.00029962548456309053}
```



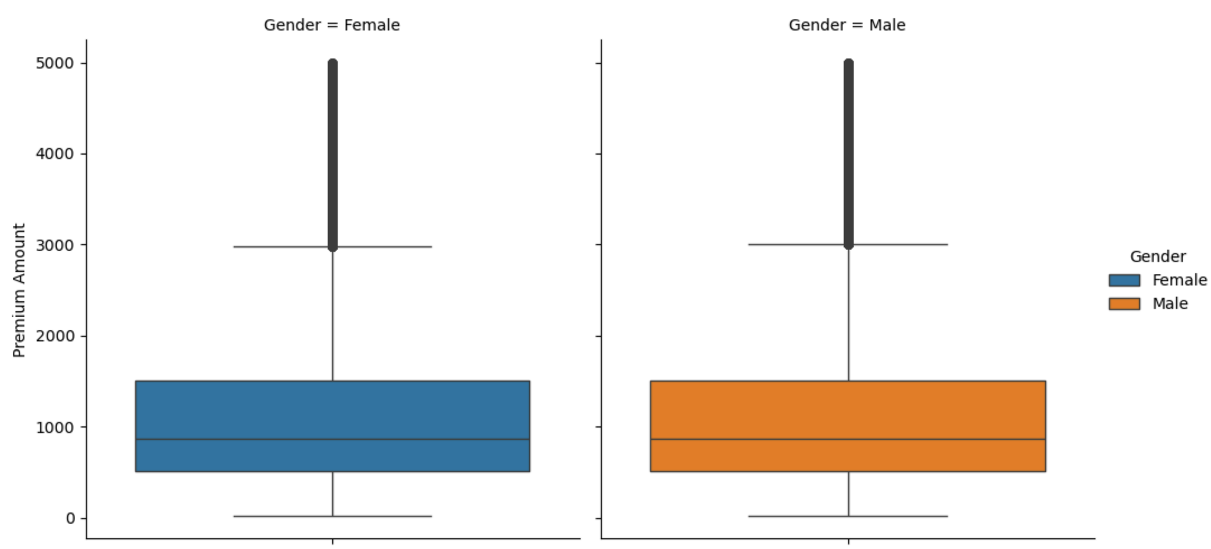
based on above chart, it seems some categorical feature have relationship with Premium amount

In [355…

```
sns.catplot(data=df,y='Premium Amount',col='Gender',kind='box', hue='Gender')
#looks like the both male and female show a similiar distribution of premium amount
##precisely speaking, more data points lies above median, suggesting some policy ho
### the data is skewed towards right: more outliers as high payment amount.
```

Out[355…

```
<seaborn.axisgrid.FacetGrid at 0x231b5be65d0>
```



In [361…

```
numeric_data.columns
```

Out[361…

```
Index(['id', 'Age', 'Annual Income', 'Number of Dependents', 'Health Score',
       'Previous Claims', 'Vehicle Age', 'Credit Score', 'Insurance Duration',
       'Premium Amount'],
      dtype='object')
```

In [393…

```
cat_data.columns
```

Out[393…

```
Index(['Gender', 'Marital Status', 'Education Level', 'Occupation', 'Location',
       'Policy Type', 'Policy Start Date', 'Customer Feedback',
       'Smoking Status', 'Exercise Frequency', 'Property Type'],
      dtype='object')
```

In [395…

```
df['Policy Type'].value_counts()
```

Out[395…

```
Policy Type
Premium          337026
Comprehensive    335572
Basic            334739
Name: count, dtype: int64
```

In [498…

```
stats=df.groupby('Location')[['Premium Amount','Annual Income']].agg(['min','mean',
#gives me the multiindex columns.
#i am going to flat the multiindex and then plot the chart to check the relationshi
```

In [500…

```
stats.columns=stats.columns = stats.columns.to_flat_index().map('_'.join)
```

In [502…

```
stats=stats.reset_index()
```

In [504…

```
stats.columns
```

Out[504…

```
Index(['Location', 'Premium Amount_min', 'Premium Amount_mean',
       'Premium Amount_median', 'Premium Amount_max', 'Annual Income_min',
       'Annual Income_mean', 'Annual Income_median', 'Annual Income_max'],
      dtype='object')
```

In [506…

```
stats
```

Out[506…

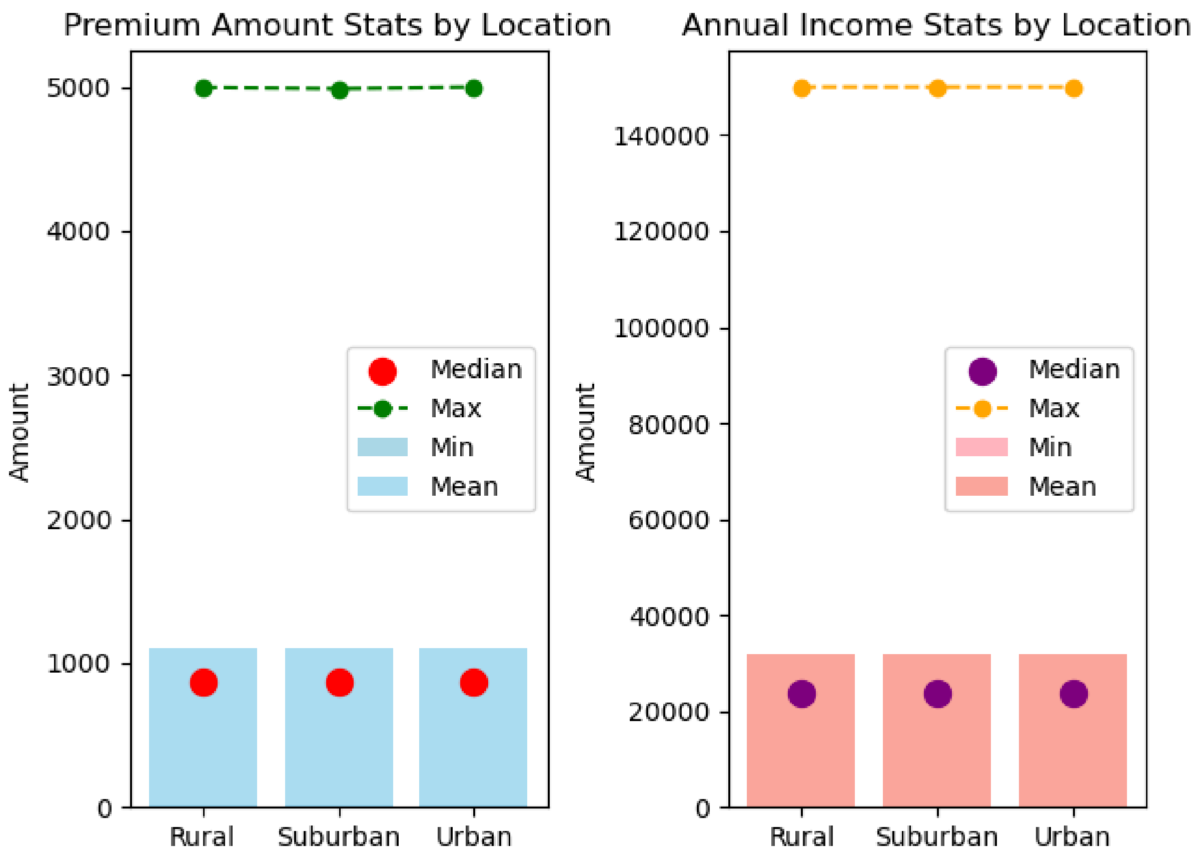
	Location	Premium Amount_min	Premium Amount_mean	Premium Amount_median	Premium Amount_max	Annual Income_min	In
0	Rural	20	1098.422601	869.0	4997	2.0	3
1	Suburban	20	1099.924692	868.0	4988	2.0	3
2	Urban	20	1102.085458	872.0	4999	1.0	3

In [514…

```
fig=plt.figure(figsize=(10,4))
fig,axes=plt.subplots(nrows=1,ncols=2)
axes[0].bar(stats['Location'], stats['Premium Amount_min'], color='lightblue', labe
axes[0].bar(stats['Location'], stats['Premium Amount_mean'], color='skyblue', label
axes[0].scatter(stats['Location'], stats['Premium Amount_median'], color='red', lab
axes[0].plot(stats['Location'], stats['Premium Amount_max'], color='green', marker=
axes[0].set_title('Premium Amount Stats by Location')
axes[0].set_ylabel('Amount')
axes[0].legend()
axes[1].bar(stats['Location'], stats['Annual Income_min'], color='lightpink', label
axes[1].bar(stats['Location'], stats['Annual Income_mean'], color='salmon', label='
axes[1].scatter(stats['Location'], stats['Annual Income_median'], color='purple', l
axes[1].plot(stats['Location'], stats['Annual Income_max'], color='orange', marker=
```

```
axes[1].set_title('Annual Income Stats by Location')
axes[1].set_ylabel('Amount')
axes[1].legend()
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x400 with 0 Axes>
```

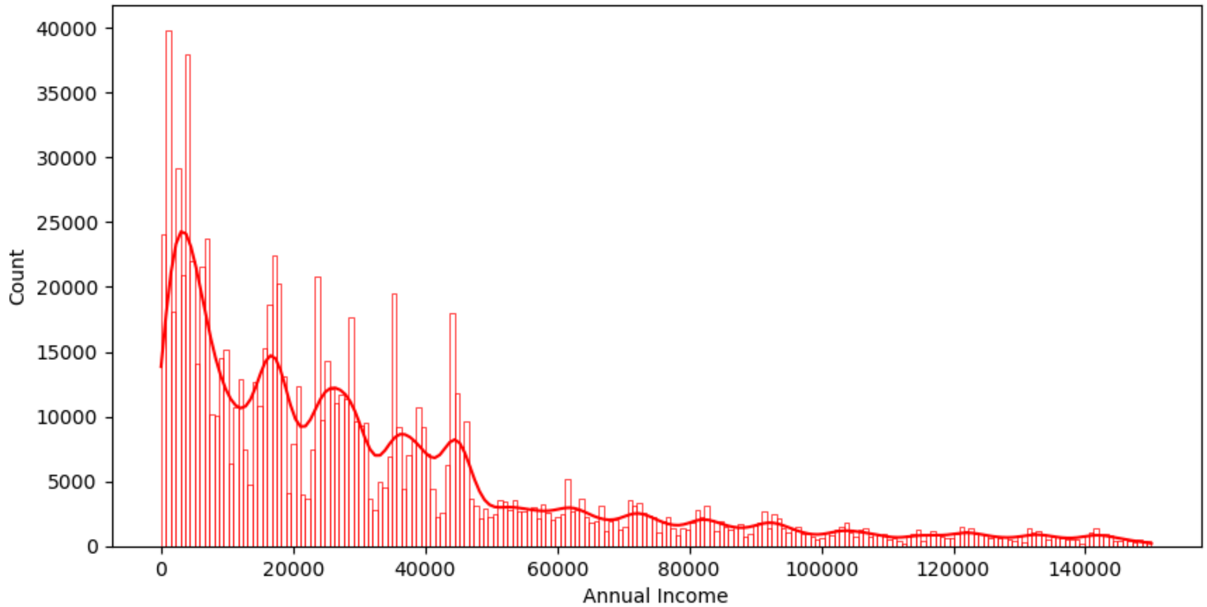


In [544…

```
plt.figure(figsize=(10,5))
sns.histplot(df['Annual Income'], color='red', kde=True,fill=False)
```

Out[544…

```
<Axes: xlabel='Annual Income', ylabel='Count'>
```



so far we have seen that the dataset is randomly dispersed without any lienar relationship within features

IN THE NEXT STEP WE WILL HEAD TO OUR REGRESSION TASK

WE WILL BEGIN BY DROPPING THE OUTLIERS

NOTE THAT THIS DATASET IS OVER A MILLION ROWS, TRAINING THE MODEL ON THE ENTIRE DATA IS COMPUTATIONALLY INFEASIBLE CONSIDERING THE RESOURCE I HAVE. THEREFORE, i WILL ADOPT A RANDOM SAMPLING STRATEGY THAT PRESERVES THE UNDERLYING DISTRIBUTION AND VARIANCE OF THE ORIGINAL DATASET, ENSURING MODEL IS TRAINED ON A REPRESNTATIVE SUBSET WHILE MAINTAINING PERFORMANCE AND RELIABILITY

In [581...
```
Q3=df['Premium Amount'].quantile(0.75)
```

In [583...
```
Q2=df['Premium Amount'].quantile(0.5)
```

In [585...
```
Q1=df['Premium Amount'].quantile(0.25)
```

In [587...
```
IQR=Q3-Q1
```

In [589...
```
UPPER_WHISKER=Q3+IQR
```

In [591...
```
LOWER_WHISKER=Q1-IQR
```

In [595...
```
Outliers=df[(df['Premium Amount']>=UPPER_WHISKER)|(df['Premium Amount']<=LOWER_WHIS
```

In [599...
```
clean_df=df.drop(Outliers.index).reset_index(drop=True)
```

In [612...
```
100-(len(clean_df)/1048575*100) #we have dropped almost 12 percentage of total data
```

Out[612… 11.888038528479129

preprocessing for model training

In [626…
```
clean_df=clean_df.drop('Income Bin',axis=1)
```

In [630…
```
model_data=clean_df.sample(n=300000, random_state=101) #roughly 33 percent of total
```

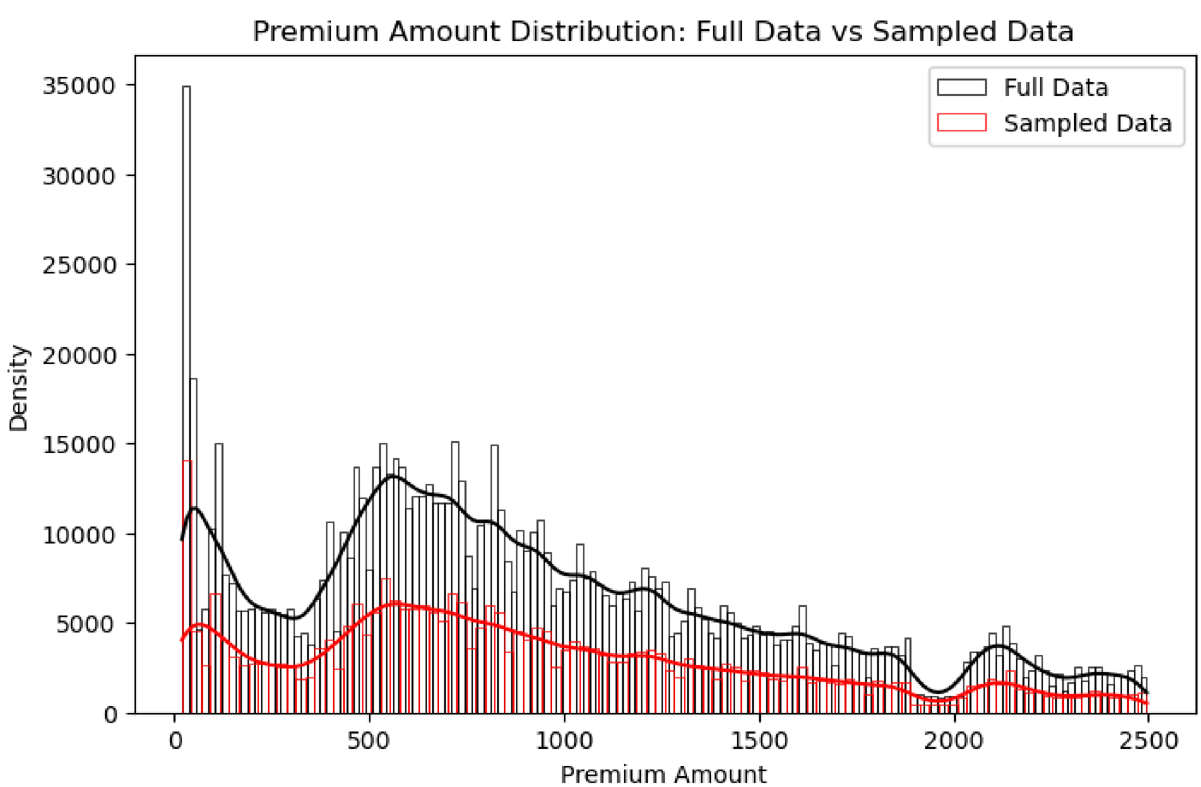
In [632…
```
len(model_data)
```

Out[632… 300000

To validate our claim, we are going to plot the 'Premium Amount' and check the dispersion.

In [655…
```
plt.figure(figsize=(8,5))

sns.histplot(clean_df['Premium Amount'], color='black', kde=True, fill=False, label
sns.histplot(model_data['Premium Amount'], color='red', kde=True, fill=False, label

plt.title("Premium Amount Distribution: Full Data vs Sampled Data")
plt.xlabel("Premium Amount")
plt.ylabel("Density")
plt.legend()
```

Out[655… <matplotlib.legend.Legend at 0x231bc599160>



The above plot verifies that random sampling allows data sampling preserving data variance.

In next step, we will focus on steps of trainig, testing and evaluating several model.

In [775…
```
from sklearn.model_selection import train_test_split,RandomizedSearchCV
from sklearn.preprocessing import StandardScaler,OneHotEncoder,OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import GradientBoostingRegressor,RandomForestRegressor
from xgboost import XGBRegressor
```

In [795…
```
model_data.columns
```

Out[795…
```
Index(['id', 'Age', 'Gender', 'Annual Income', 'Marital Status',
       'Number of Dependents', 'Education Level', 'Occupation', 'Health Score',
       'Location', 'Policy Type', 'Previous Claims', 'Vehicle Age',
       'Credit Score', 'Insurance Duration', 'Policy Start Date',
       'Customer Feedback', 'Smoking Status', 'Exercise Frequency',
       'Property Type', 'Premium Amount'],
      dtype='object')
```

In [797…
```
X=model_data.drop(['Premium Amount','id'],axis=1)
y=model_data['Premium Amount']
```

In [799…
```
X_columns=X.select_dtypes(include='object').columns
```

In [801…
```
numeric_X=X.select_dtypes(include='number').columns
```

In [803…
```
X_columns
```

Out[803…
```
Index(['Gender', 'Marital Status', 'Education Level', 'Occupation', 'Location',
       'Policy Type', 'Policy Start Date', 'Customer Feedback',
       'Smoking Status', 'Exercise Frequency', 'Property Type'],
      dtype='object')
```

In [805…
```
nominal_cols = ['Gender', 'Marital Status', 'Smoking Status','Policy Start Date']
ordinal_cols = ['Customer Feedback','Education Level', 'Occupation', 'Location', 'P
```

splitting the data in train/validate/test split

60% training (50% of remaining as validate and other 50% as final holdout test set)

In [808…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_st
```

In [810…
```
X_val, X_test, y_val, y_test = train_test_split(X_test,y_test, test_size=0.50, rand
```

In [812...
```
transformer=ColumnTransformer(transformers=[('nominal',OneHotEncoder(handle_unknown
                                            ('ordinal',OrdinalEncoder(), ordinal_col
                                            ('num',StandardScaler(),numeric_X)])
```

In [814...
```
def my_model(model,X_train,y_train,X_val,y_val):
    pipe=Pipeline(steps=[('col_transformer',transformer),
                         ('used_model',model)])
    pipe.fit(X_train,y_train)
    pred=pipe.predict(X_val)
    mae=mean_absolute_error(y_val,pred)
    rmse=np.sqrt(mean_squared_error(y_val,pred))
    r_score=r2_score(y_val,pred)
    print(f"Used model is: {model}")
    print(f"The Mean Absolute Error: {mae:.4f}")
    print(f"THE RMSE is: {rmse:.4f}")
    print(f"R squared score is:{r_score}")
```

In [761...
```
my_model(model=RandomForestRegressor(n_estimators=200),
         X_train=X_train,
         y_train=y_train,
         X_val=X_val,
         y_val=y_val
        )
```

```
Used model is: RandomForestRegressor(n_estimators=200)
The Mean Absolute Error: 479.6721
THE RMSE is: 600.2831
R squared score is:0.048251554990720646
```

In [816...
```
my_model(model=XGBRegressor(n_estimators=100,learning_rate=0.05,max_depth=6,subsamp
         X_train=X_train,
         y_train=y_train,
         X_val=X_val,
         y_val=y_val
        )
```

```
Used model is: XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.05, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=100,
             n_jobs=-1, num_parallel_tree=None, ...)
The Mean Absolute Error: 475.6411
THE RMSE is: 597.3336
R squared score is:0.05758124589920044
```

In [820…

```
my_model(model=CatBoostRegressor(
    iterations=600,
    depth=8,
    learning_rate=0.05,
    l2_leaf_reg=3,
    random_seed=42,
    verbose=100),
         X_train=X_train,
         y_train=y_train,
         X_val=X_val,
         y_val=y_val
        )
```

```
0:      learn: 614.1804609      total: 46.4ms   remaining: 27.8s
100:    learn: 597.3235825      total: 4.82s    remaining: 23.8s
200:    learn: 593.6675726      total: 12.7s    remaining: 25.2s
300:    learn: 589.7937734      total: 17.9s    remaining: 17.8s
400:    learn: 586.6558952      total: 23.1s    remaining: 11.4s
500:    learn: 583.6303917      total: 29.9s    remaining: 5.92s
599:    learn: 580.7173813      total: 34.8s    remaining: 0us
Used model is: <catboost.core.CatBoostRegressor object at 0x00000231A8CBA150>
The Mean Absolute Error: 474.5964
THE RMSE is: 596.9703
R squared score is:0.05872735242580229
```

In [759…

```
model_data['Premium Amount'].mean()
```

Out[759…

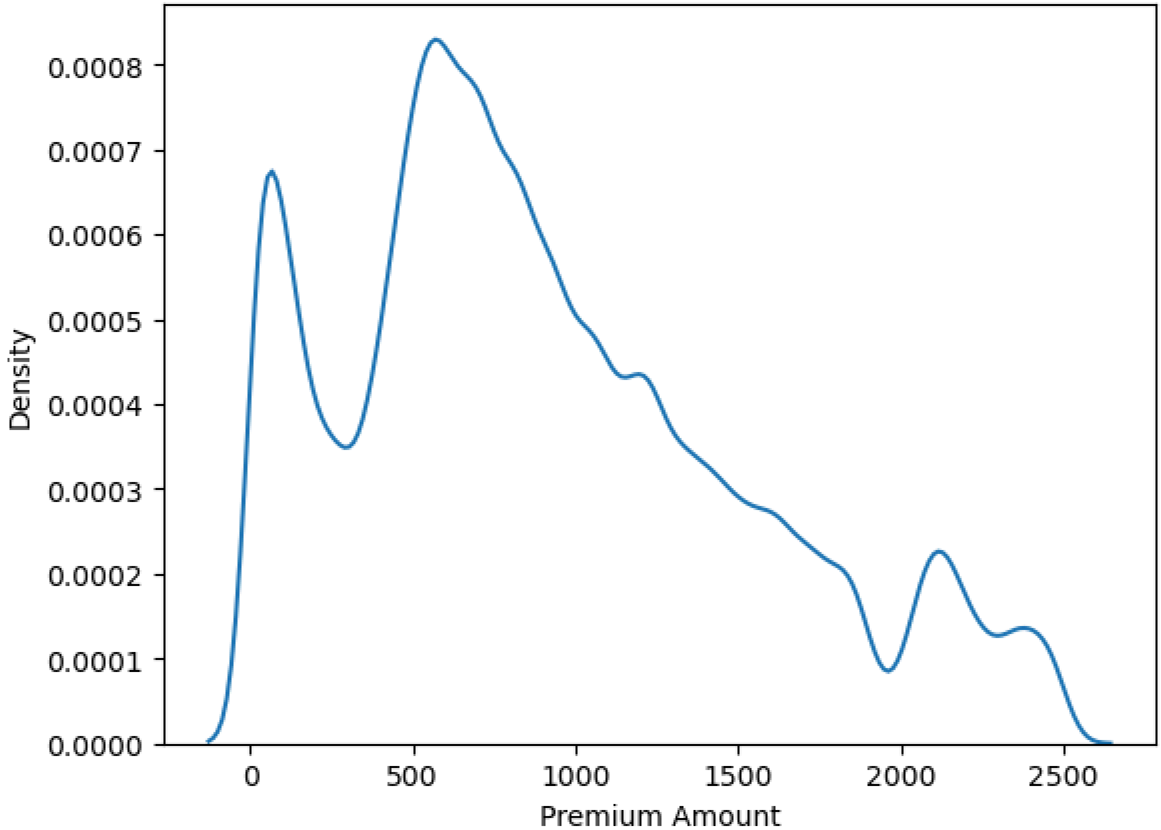
```
914.7659
```

In [767…

```
sns.kdeplot(data=model_data,x='Premium Amount')
```

Out[767…

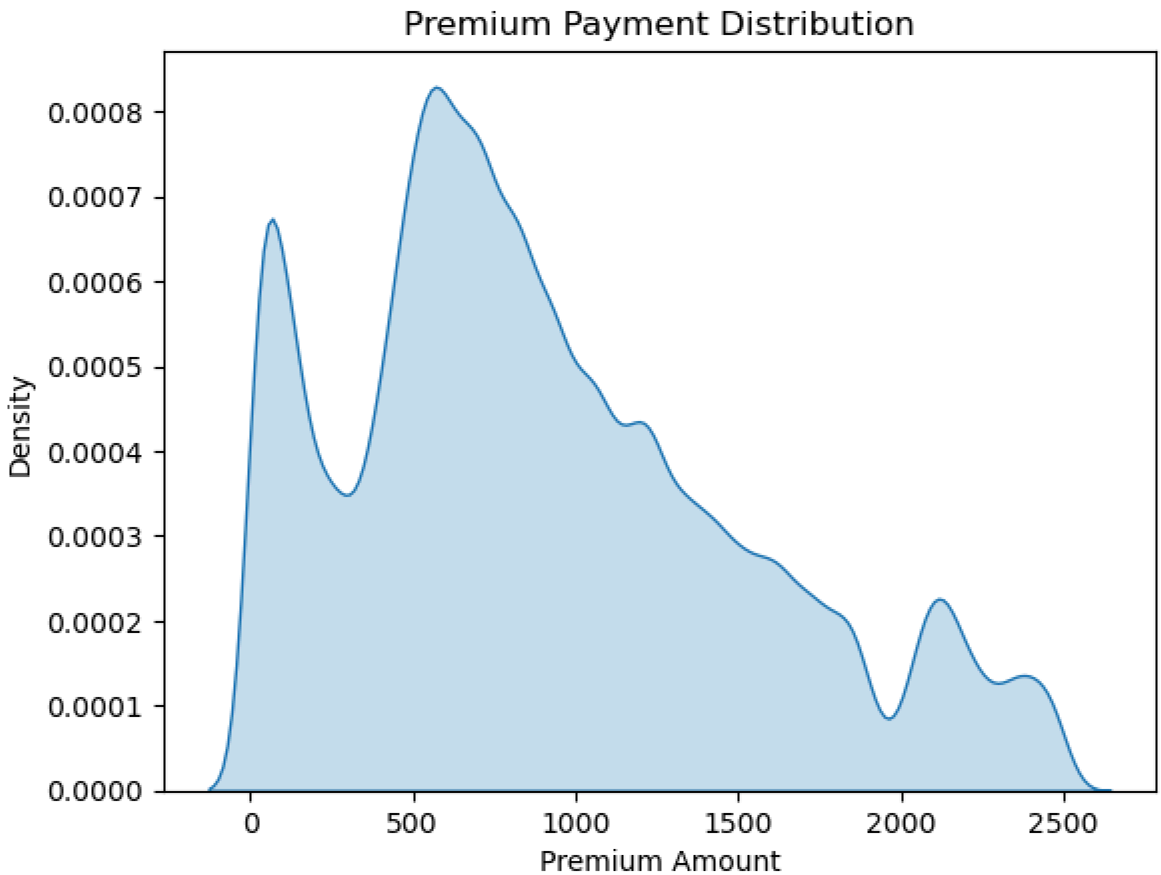
```
<Axes: xlabel='Premium Amount', ylabel='Density'>
```



In [829…

```
sns.kdeplot(model_data['Premium Amount'], shade=True)
plt.title("Premium Payment Distribution")
plt.show()
```

```
C:\Users\rautu\AppData\Local\Temp\ipykernel_13192\3505316733.py:1: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(model_data['Premium Amount'], shade=True)
```


Premium Payment Distribution
Density
0.0000
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
0.0008
0
500
1000
1500
2000
2500
Premium Amount

In [ ]: