1. Importing necessary libraries

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

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

In [3]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [4]:
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train_te
```

2. DATA Loading

In [5]:
```
df_m = pd.read_excel("premiums.xlsx")
df_m.head(10)
```

Out[5]:

	Age	Gender	Region	Marital_status	Number Of Dependants	BMI_Category	Smoking_
0	26	Male	Northwest	Unmarried	0	Normal	No S
1	29	Female	Southeast	Married	2	Obesity	
2	49	Female	Northeast	Married	2	Normal	No S
3	30	Female	Southeast	Married	3	Normal	No S
4	18	Male	Northeast	Unmarried	0	Overweight	
5	56	Male	Northeast	Married	3	Obesity	Occ
6	33	Male	Southeast	Married	3	Normal	
7	43	Male	Northeast	Married	3	Overweight	
8	59	Female	Southeast	Unmarried	0	Overweight	No S
9	22	Female	Northwest	Unmarried	0	Underweight	No S

In [6]:

```
df_m.shape
```

Out[6]:

```
(50000, 13)
```

In [7]:

```
df_m.columns
```

Out[7]:

```
Index(['Age', 'Gender', 'Region', 'Marital_status', 'Number Of Dependants',
       'BMI_Category', 'Smoking_Status', 'Employment_Status', 'Income_Level',
       'Income_Lakhs', 'Medical History', 'Insurance_Plan',
       'Annual_Premium_Amount'],
      dtype='object')
```

In [8]:

```
# giving the column names a more proper structure

df_m.columns = df_m.columns.str.replace(" ","_").str.lower()
df_m.head(5)
```

Out[8]:

	age	gender	region	marital_status	number_of_dependants	bmi_category
0	26	Male	Northwest	Unmarried	0	Normal
1	29	Female	Southeast	Married	2	Obesity
2	49	Female	Northeast	Married	2	Normal
3	30	Female	Southeast	Married	3	Normal
4	18	Male	Northeast	Unmarried	0	Overweight

3. Exploratory Data Analysis and Data Cleaning

3A - Handling the NA Values

In [9]:
```
df_m.isna().sum()
```

Out[9]:
```
age                      0
gender                   0
region                   0
marital_status           0
number_of_dependants     0
bmi_category             0
smoking_status          11
employment_status        2
income_level            13
income_lakhs             0
medical_history          0
insurance_plan           0
annual_premium_amount    0
dtype: int64
```

In [10]:
```
df_m.dropna(inplace=True)
```

In [11]:
```
df_m.shape
```

Out[11]:
```
(49976, 13)
```

In [12]:
```
df_m.isna().sum()
```

Out[12]:
```
age                      0
gender                   0
region                   0
marital_status           0
number_of_dependants     0
bmi_category             0
smoking_status           0
employment_status        0
income_level             0
income_lakhs             0
medical_history          0
insurance_plan           0
annual_premium_amount    0
dtype: int64
```

3B - Handling the Duplicates

df_m.duplicated().sum()

In [13]:
```
df_m.drop_duplicates(inplace=True)
df_m.duplicated().sum()
```

Out[13]: 0

In [14]:
```
df_m.describe()
```

Out[14]:

	age	number_of_dependants	income_lakhs	annual_premium_am
count	49976.000000	49976.000000	49976.000000	49976.00
mean	34.591764	1.711842	23.021150	15766.81
std	15.000378	1.498195	24.221794	8419.99
min	18.000000	-3.000000	1.000000	3501.00
25%	22.000000	0.000000	7.000000	8607.75
50%	31.000000	2.000000	17.000000	13928.00
75%	45.000000	3.000000	31.000000	22273.50
max	356.000000	5.000000	930.000000	43471.00

number_of_dependents, the min. value is -3 **(wrong)**

3C - Data Cleaning

In [15]:
```
df_m[df_m['number_of_dependants']<0]['number_of_dependants'].unique()
```

Out[15]:
```
array([-3, -1], dtype=int64)
```

We can see some negative values in number_of_dependants. We can replace them with positive numbers

In [16]:
```
df_m['number_of_dependants'] = df_m['number_of_dependants'].abs()

df_m.describe()
```

Out[16]:

	age	number_of_dependants	income_lakhs	annual_premium_am
count	49976.000000	49976.000000	49976.000000	49976.00
mean	34.591764	1.717284	23.021150	15766.81
std	15.000378	1.491953	24.221794	8419.99
min	18.000000	0.000000	1.000000	3501.00
25%	22.000000	0.000000	7.000000	8607.75
50%	31.000000	2.000000	17.000000	13928.00
75%	45.000000	3.000000	31.000000	22273.50
max	356.000000	5.000000	930.000000	43471.00

3D - Numeric Columns

3D(i) - Univariate Analysis : Numeric Column

WHY ??

I use **select_dtypes** to dynamically identify numeric features, which helps streamline preprocessing steps like scaling, correlation analysis, and feature selection in regression models.

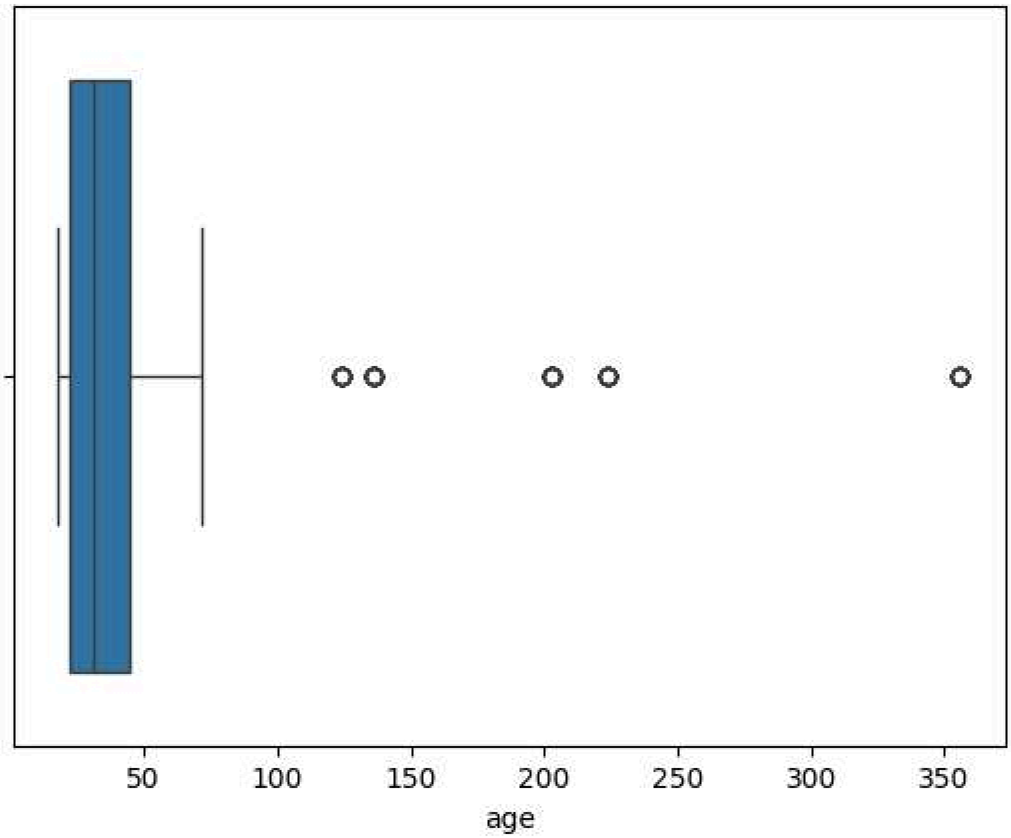
In [17]:
```
numeric_columns = df_m.select_dtypes(include=['float64', 'int64']).columns
numeric_columns

# It automatically finds all numeric columns (integers + decimals) from df_m a
```

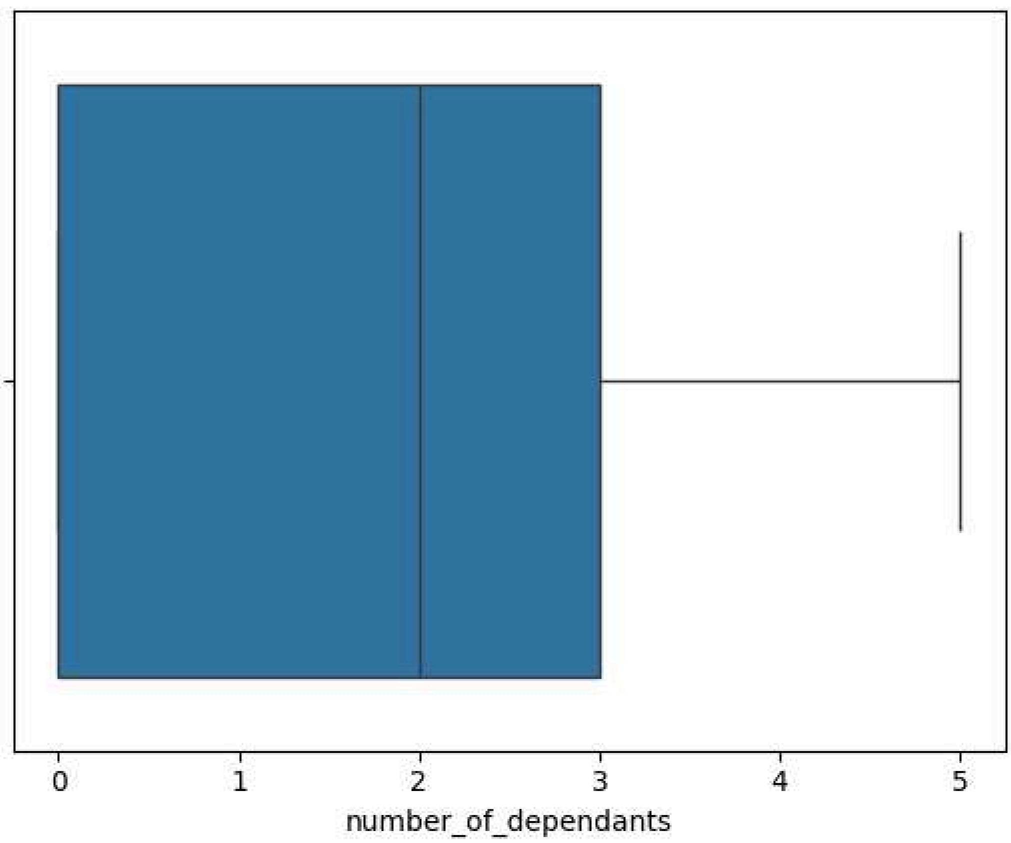
Out[17]:
```
Index(['age', 'number_of_dependants', 'income_lakhs', 'annual_premium_amoun
t'], dtype='object')
```

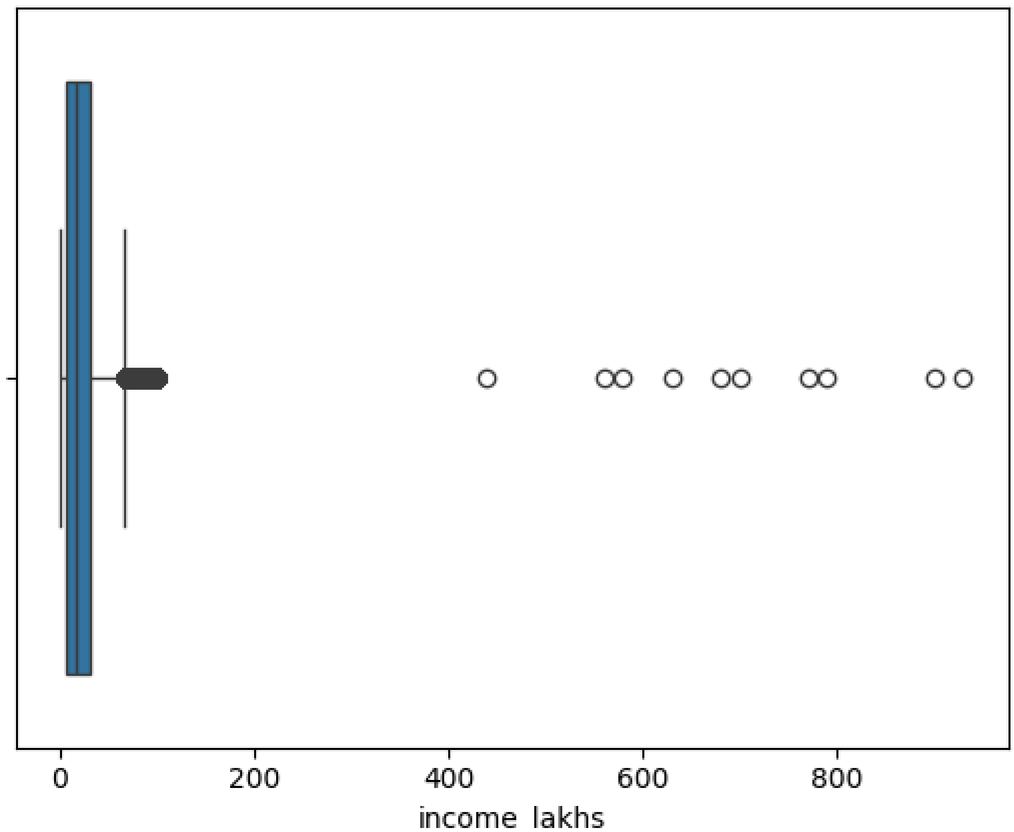
3D(ii) - Box plots for numeric columns

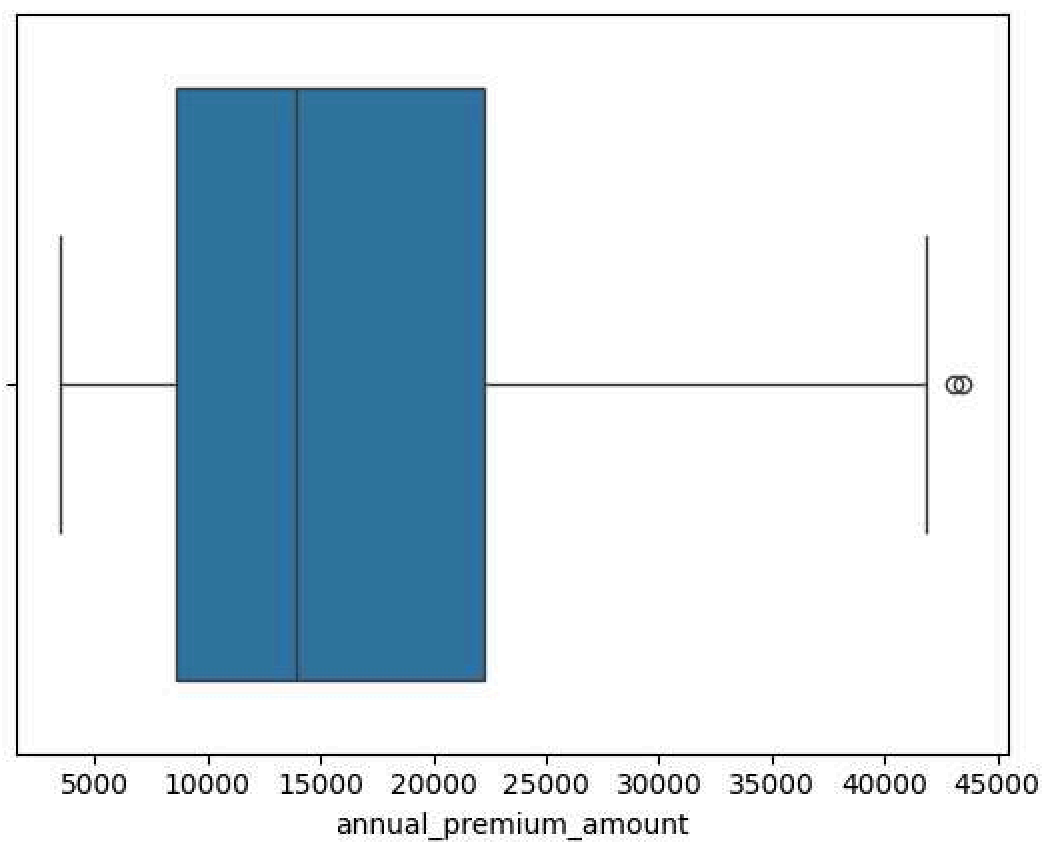
In [18]:
```
for col in numeric_columns:
    sns.boxplot(x = df_m[col])
```

```
plt.show()
```






0
200
400
600
800
income_lakhs


5000
10000
15000
20000
25000
30000
35000
40000
45000
annual_premium_amount

3D(iii) - Outlier Treatment

For 'Age' column

In [19]:
```
df_m[df_m['age']>100]['age'].unique()
```

Out[19]:
```
array([224, 124, 136, 203, 356], dtype=int64)
```

In [20]:
```
df1 = df_m[df_m.age <= 100]
df1.age.describe()
```

Out[20]:
```
count    49918.000000
mean        34.401839
std         13.681600
min         18.000000
25%         22.000000
50%         31.000000
75%         45.000000
max         72.000000
Name: age, dtype: float64
```

For 'Income' column

In [21]:
```
def get_iqr_bounds(col):
    Q1, Q3 = col.quantile([0.25, 0.75])
    IQR = Q3-Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return lower_bound, upper_bound

lower, upper = get_iqr_bounds(df1['income_lakhs'])
lower, upper

# It calculates the lower and upper limits beyond which values in a column are
```

Out[21]:
```
(-29.0, 67.0)
```

In [22]:
```
df1[df1.income_lakhs > upper].shape
```

Out[22]:
```
(3559, 13)
```

There are many legitimate records (3559) that we will get rid of if we use IQR bounds method. Hence after discussion with business we decided to use a simple quantile bound.

In [23]:
```
quantile_thresold = df1.income_lakhs.quantile(0.999)
quantile_thresold
```

```
# It calculates the 99.9th percentile of the income_lakhs column and stores th
```

Out[23]:

```
100.0
```

For skewed variables like income, use high-percentile capping (e.g., 99.9%) to control extreme values while preserving data volume and business realism.

In [24]:

```
df1[df1.income_lakhs > quantile_thresold].shape

# It tells you how many rows have income values greater than the 99.9% quantil
```

Out[24]:

```
(10, 13)
```

In [25]:

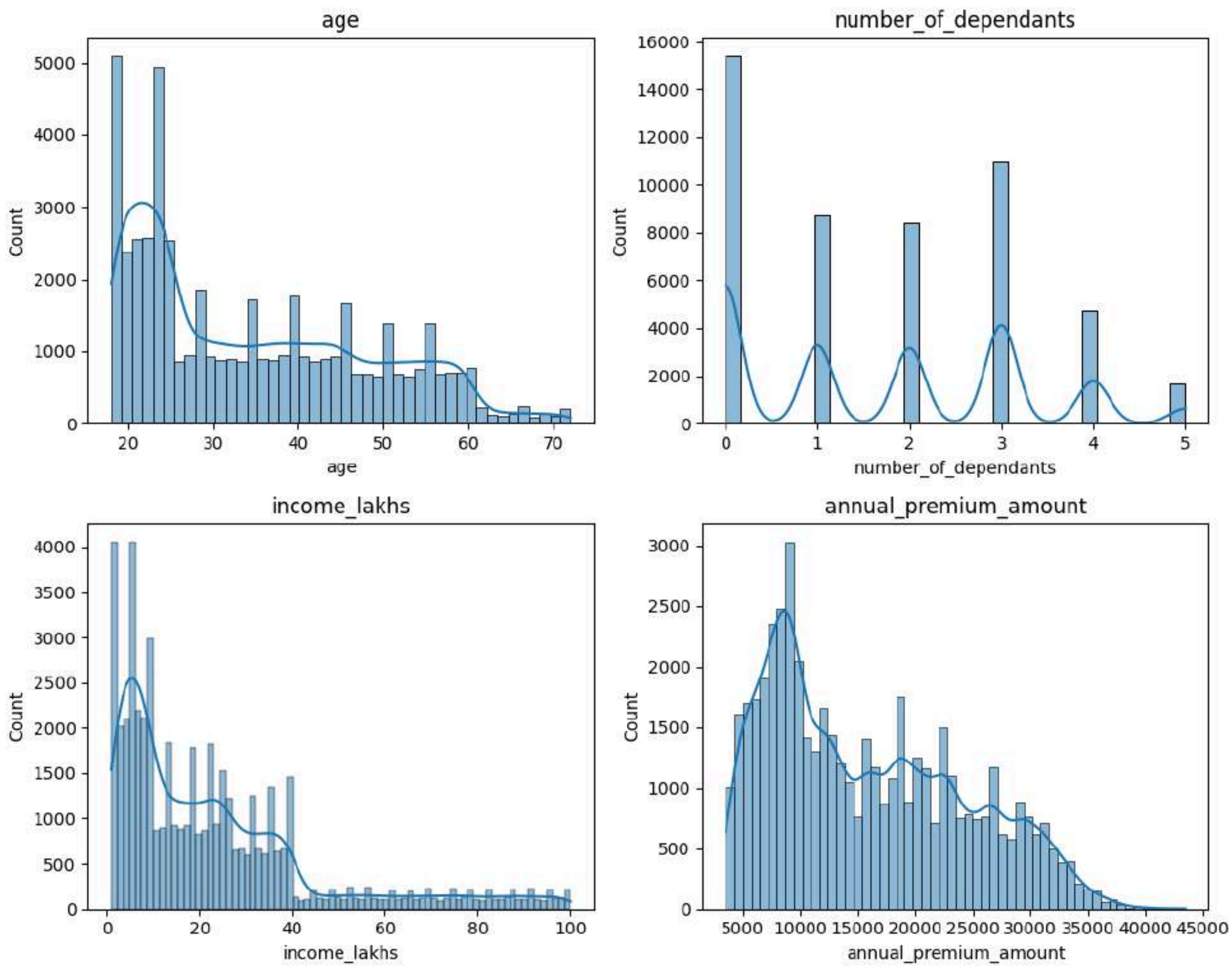
```
df2 = df1[df1.income_lakhs <= quantile_thresold].copy()
df2.describe()
```

Out[25]:

	age	number_of_dependants	income_lakhs	annual_premium_am
count	49908.000000	49908.000000	49908.000000	49908.00
mean	34.401579	1.717640	22.889897	15765.73
std	13.681604	1.492032	22.170699	8418.67
min	18.000000	0.000000	1.000000	3501.00
25%	22.000000	0.000000	7.000000	8608.00
50%	31.000000	2.000000	17.000000	13928.00
75%	45.000000	3.000000	31.000000	22270.50
max	72.000000	5.000000	100.000000	43471.00

In [26]:

```
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))  # Adjust the size

for i, column in enumerate(numeric_columns):

    ax = axs[i//2, i%2]  # Determines the position of the subplot in the grid

    sns.histplot(df2[column], kde=True, ax=ax)

    ax.set_title(column) # Each subplot gets its own column name as title

plt.tight_layout() # Prevents titles and plots from overlapping
plt.show()

# The above code creates a 2×2 grid of plots and draws a histogram (with KDE)
```



3E - Categorical Columns

In [27]:
```
categorical_cols = ['gender', 'region', 'marital_status', 'bmi_category', 'smo
for col in categorical_cols:
    print(col, ":", df2[col].unique())
```

```
gender : ['Male' 'Female']
region : ['Northwest' 'Southeast' 'Northeast' 'Southwest']
marital_status : ['Unmarried' 'Married']
bmi_category : ['Normal' 'Obesity' 'Overweight' 'Underweight']
smoking_status : ['No Smoking' 'Regular' 'Occasional' 'Smoking=0' 'Does Not Smo
ke'
 'Not Smoking']
employment_status : ['Salaried' 'Self-Employed' 'Freelancer']
income_level : ['<10L' '10L - 25L' '> 40L' '25L - 40L']
medical_history : ['Diabetes' 'High blood pressure' 'No Disease'
 'Diabetes & High blood pressure' 'Thyroid' 'Heart disease'
 'High blood pressure & Heart disease' 'Diabetes & Thyroid'
 'Diabetes & Heart disease']
insurance_plan : ['Bronze' 'Silver' 'Gold']
```

The category for **smoking_status** is not uniform.

No Smoking; Smoking=0; Does not smoke; Not Smoking - all are SAME

In [28]:
```
# giving the uniform values to the smoking_status

df2['smoking_status'].replace(
    {
    'Not Smoking': 'No Smoking',
    'Does Not Smoke': 'No Smoking',
    'Smoking=0': 'No Smoking'
    }, inplace=True)
```

In [29]:
```
df2['smoking_status'].unique()
```

Out[29]:
```
array(['No Smoking', 'Regular', 'Occasional'], dtype=object)
```

3E(i) - Univariate Analysis

In [30]:
```
df2.head(5)
```

Out[30]:

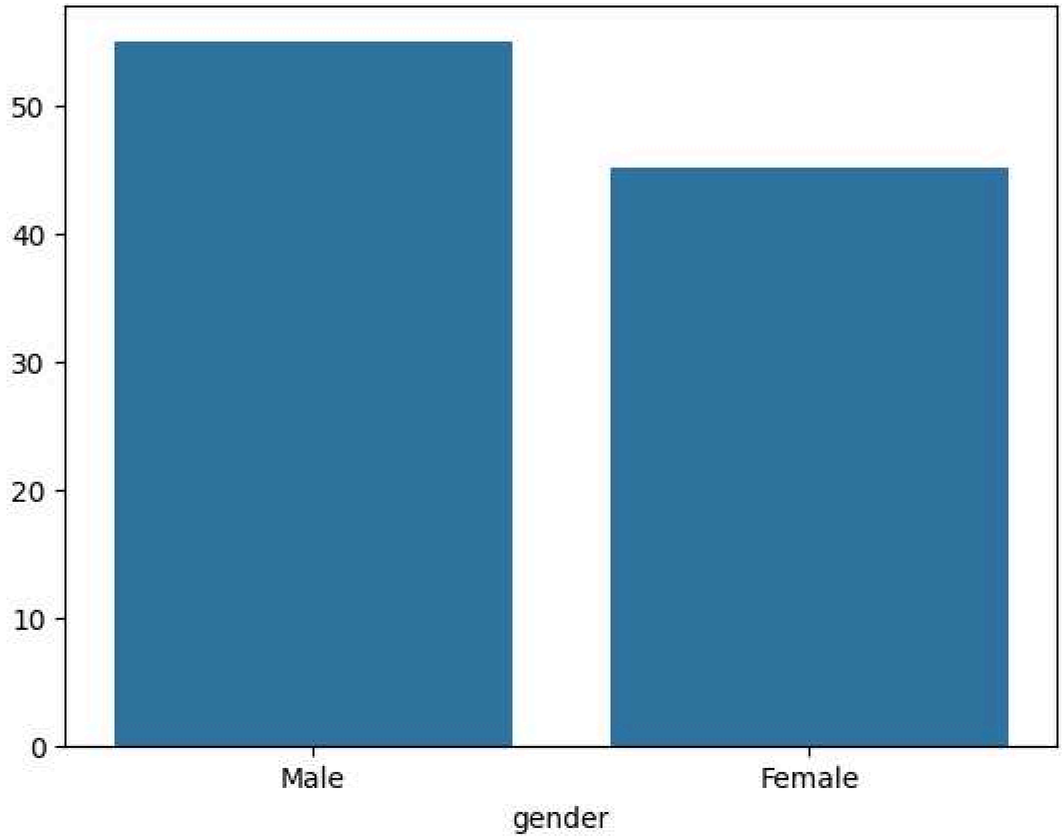
	age	gender	region	marital_status	number_of_dependants	bmi_category
0	26	Male	Northwest	Unmarried	0	Normal
1	29	Female	Southeast	Married	2	Obesity
2	49	Female	Northeast	Married	2	Normal
3	30	Female	Southeast	Married	3	Normal
4	18	Male	Northeast	Unmarried	0	Overweight

In [31]:
```
# percentage count of the two genders

pct_count = df2['gender'].value_counts(normalize=True)*100
pct_count
```

Out[31]:
```
gender
Male      54.963132
Female    45.036868
Name: proportion, dtype: float64
```

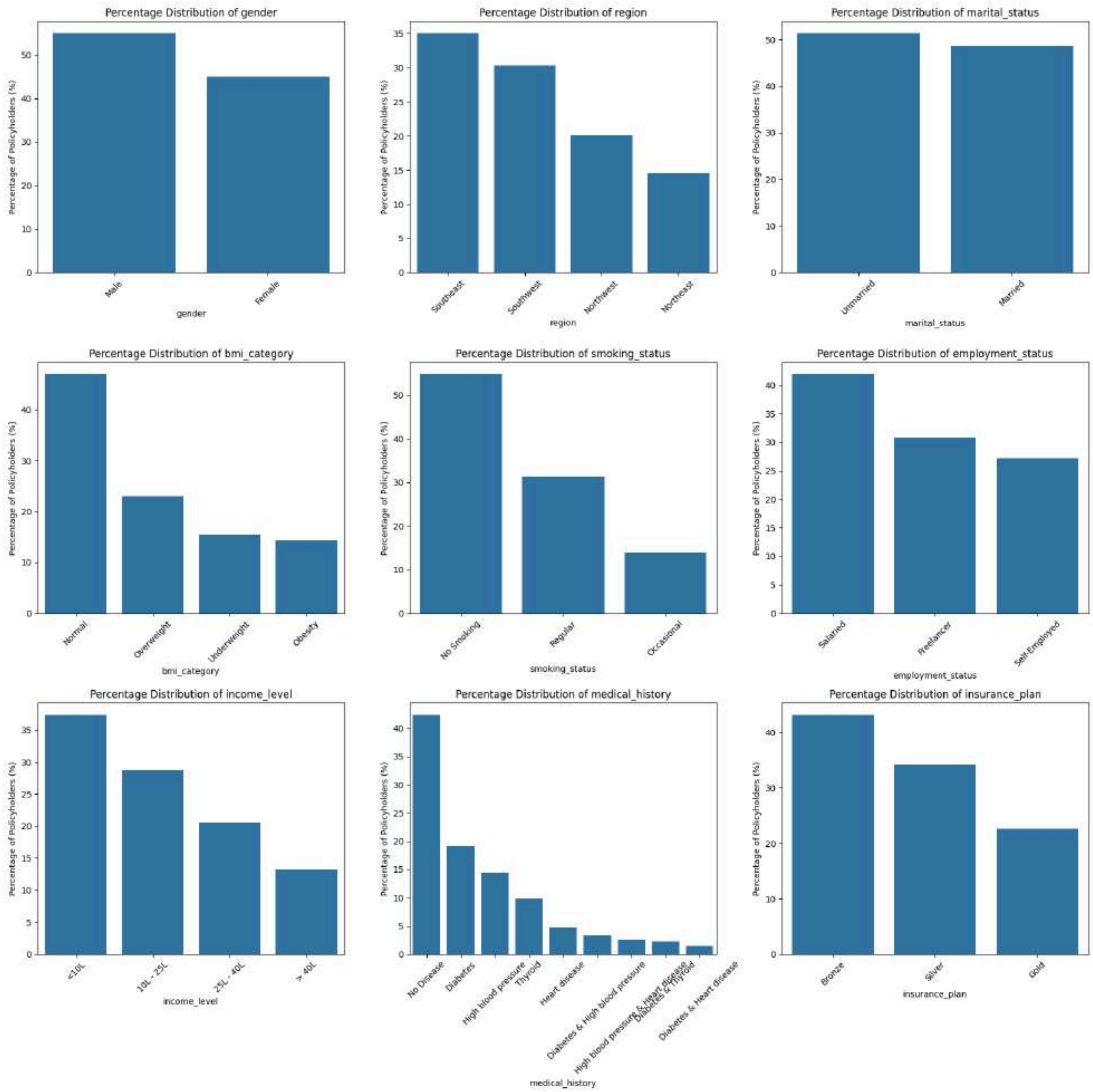
In [32]:
```
sns.barplot(x = pct_count.index, y = pct_count.values)
```

Out[32]:
```
<Axes: xlabel='gender'>
```



In [33]:

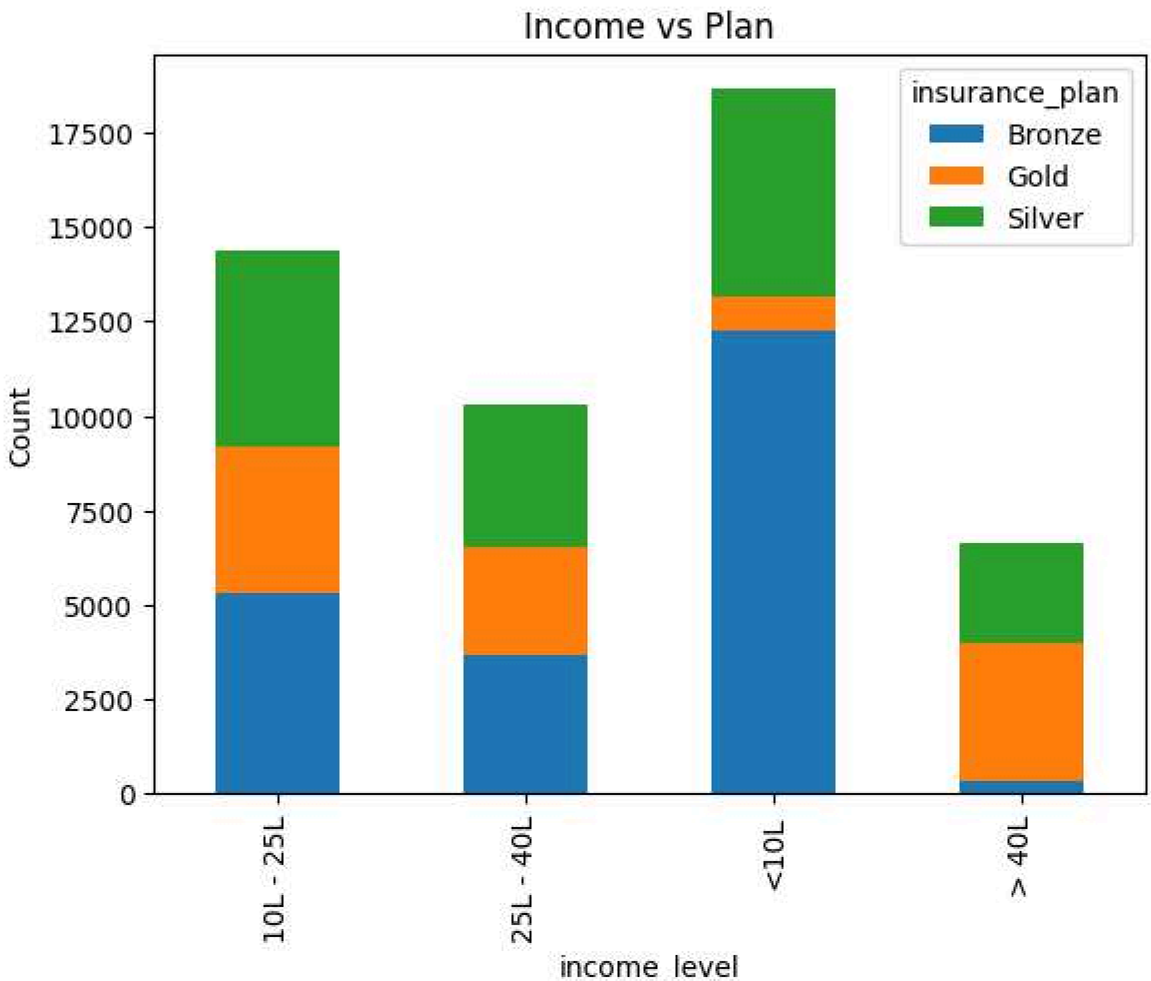
```
# Checking the category-wise distribution of each category column

fig, axes = plt.subplots(3, 3, figsize=(18, 18))
axes = axes.flatten()  # Flattening the 2D array of axes into 1D for easier it

for ax, column in zip(axes, categorical_cols):

    # Calculate the percentage distribution of each category
    category_counts = df2[column].value_counts(normalize=True) * 100  # normal

    # Plotting the distribution using barplot
    sns.barplot(x = category_counts.index, y = category_counts.values, ax=ax)
    ax.set_title(f'Percentage Distribution of {column}')
    ax.set_ylabel('Percentage of Policyholders (%)')
    ax.set_xlabel(column)  # Set xlabel to the column name for clarity

    ax.tick_params(axis='x', rotation=45)

plt.tight_layout()  # Adjusts plot parameters for better fit in the figure wir
plt.show()
```



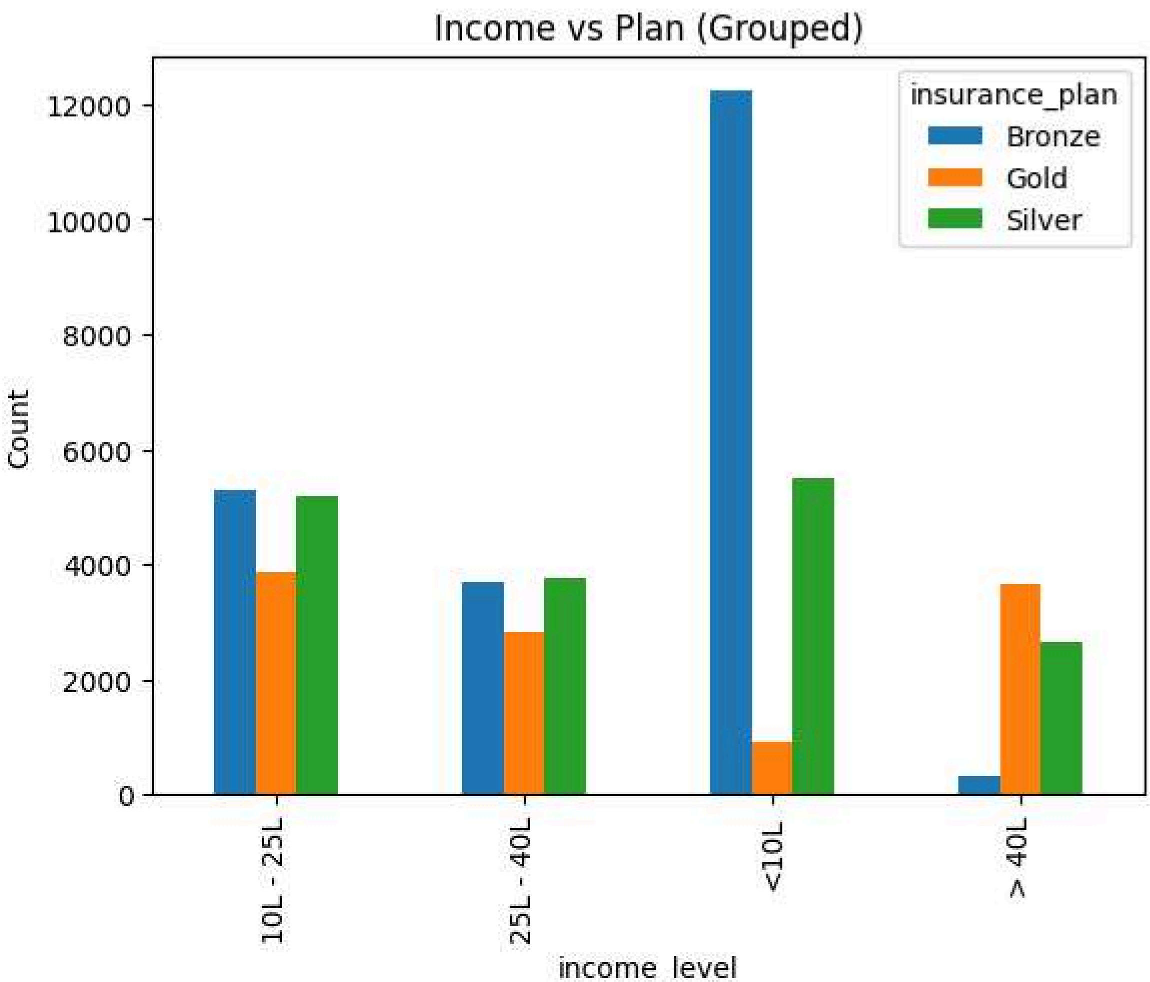
3E(ii) - Bivariate Analysis

In [34]:
```python
# Cross-tabulation of gender and smoking status

crosstab = pd.crosstab(df2['income_level'], df2['insurance_plan'])
print(crosstab)

# Plotting the crosstab
crosstab.plot(kind='bar', stacked=True)
plt.title('Income vs Plan')
plt.ylabel('Count')
plt.show()
```

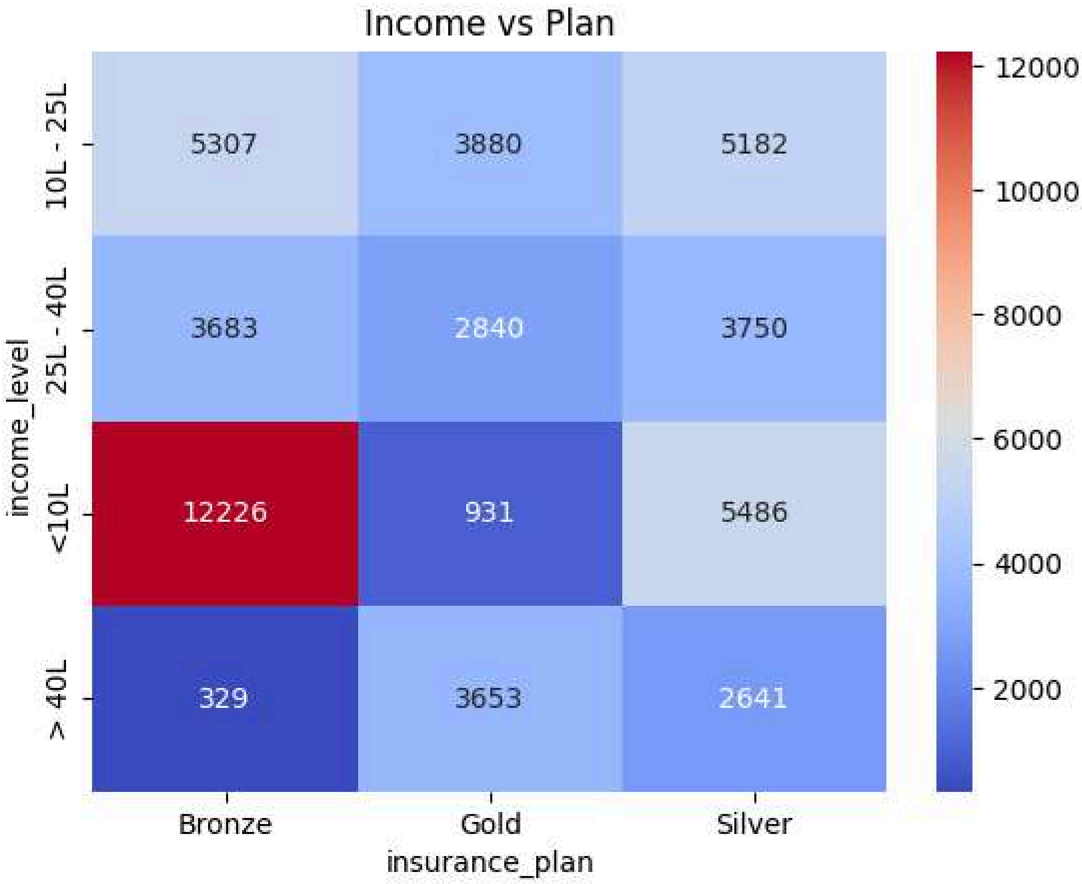
```
insurance_plan  Bronze  Gold  Silver
income_level
10L - 25L         5307  3880    5182
25L - 40L         3683  2840    3750
<10L             12226   931    5486
> 40L              329  3653    2641
```



In [35]:
```
crosstab.plot(kind='bar')
plt.title('Income vs Plan (Grouped)')
plt.ylabel('Count')
plt.show()
```



In [36]:
```
sns.heatmap(crosstab, annot=True, cmap='coolwarm',fmt="d")
plt.title('Income vs Plan')
plt.show()
```



4. Feature Engineering

In [37]:
```
df2.head(25)
```

Out[37]:

	age	gender	region	marital_status	number_of_dependants	bmi_categor
0	26	Male	Northwest	Unmarried	0	Norm
1	29	Female	Southeast	Married	2	Obesit
2	49	Female	Northeast	Married	2	Norm
3	30	Female	Southeast	Married	3	Norm
4	18	Male	Northeast	Unmarried	0	Overweigh
5	56	Male	Northeast	Married	3	Obesit
6	33	Male	Southeast	Married	3	Norm
7	43	Male	Northeast	Married	3	Overweigh
8	59	Female	Southeast	Unmarried	0	Overweigh
9	22	Female	Northwest	Unmarried	0	Underweigh
10	21	Female	Southeast	Unmarried	0	Norm
11	46	Female	Southeast	Married	4	Norm
12	68	Female	Southwest	Married	1	Norm
13	59	Female	Northeast	Married	2	Obesit
14	60	Male	Northwest	Married	5	Underweigh
15	27	Male	Southwest	Married	4	Norm
16	25	Male	Southeast	Unmarried	0	Norm
17	36	Male	Northwest	Unmarried	0	Norm
18	29	Male	Northwest	Married	2	Obesit
19	20	Male	Southeast	Unmarried	2	Overweigh
20	28	Female	Southwest	Unmarried	0	Overweigh
21	22	Female	Southwest	Unmarried	0	Underweigh
22	32	Male	Northwest	Married	4	Underweigh
23	19	Male	Southwest	Unmarried	0	Norm
24	55	Male	Southeast	Married	4	Norm

In [38]:

```
df2.medical_history.unique()
```

Out[38]:
```
array(['Diabetes', 'High blood pressure', 'No Disease',
       'Diabetes & High blood pressure', 'Thyroid', 'Heart disease',
       'High blood pressure & Heart disease', 'Diabetes & Thyroid',
       'Diabetes & Heart disease'], dtype=object)
```

Need to convert the above **text** data into **numeric** data

4A - Risk Score

It is a numerical representation of the likelihood or severity of an adverse outcome, calculated using historical data, business rules, or machine learning models.

In [39]:
```
# Defining the risk scores for each condition

risk_scores = {
    "diabetes": 6,
    "heart disease": 8,
    "high blood pressure":6,
    "thyroid": 5,
    "no disease": 0,
    "none":0
}

df2[['disease1', 'disease2']] = df2['medical_history'].str.split(" & ", expand
df2.head(10)
```

Out[39]:

	age	gender	region	marital_status	number_of_dependants	bmi_category
0	26	Male	Northwest	Unmarried	0	Normal
1	29	Female	Southeast	Married	2	Obesity
2	49	Female	Northeast	Married	2	Normal
3	30	Female	Southeast	Married	3	Normal
4	18	Male	Northeast	Unmarried	0	Overweight
5	56	Male	Northeast	Married	3	Obesity
6	33	Male	Southeast	Married	3	Normal
7	43	Male	Northeast	Married	3	Overweight
8	59	Female	Southeast	Unmarried	0	Overweight
9	22	Female	Northwest	Unmarried	0	Underweight

In [40]:
```
# Calculating total_risk_score

df2['disease1'].fillna('none', inplace=True)
df2['disease2'].fillna('none', inplace=True)
df2['total_risk_score'] = 0

for disease in ['disease1', 'disease2']:
    df2['total_risk_score'] += df2[disease].map(risk_scores)

max_score = df2['total_risk_score'].max()     # Normalizing the risk score to
min_score = df2['total_risk_score'].min()
df2['normalized_risk_score'] = (df2['total_risk_score'] - min_score) / (max_sc

df2.head(10)
```

Out[40]:

	age	gender	region	marital_status	number_of_dependants	bmi_category
0	26	Male	Northwest	Unmarried	0	Normal
1	29	Female	Southeast	Married	2	Obesity
2	49	Female	Northeast	Married	2	Normal
3	30	Female	Southeast	Married	3	Normal
4	18	Male	Northeast	Unmarried	0	Overweight
5	56	Male	Northeast	Married	3	Obesity
6	33	Male	Southeast	Married	3	Normal
7	43	Male	Northeast	Married	3	Overweight
8	59	Female	Southeast	Unmarried	0	Overweight
9	22	Female	Northwest	Unmarried	0	Underweight

In [41]:
```
# Encoding the other 'text' columns

df2['insurance_plan'] = df2['insurance_plan'].map({'Bronze': 1, 'Silver': 2, '
df2.head(10)
```

Out[41]:

	age	gender	region	marital_status	number_of_dependants	bmi_category
0	26	Male	Northwest	Unmarried	0	Normal
1	29	Female	Southeast	Married	2	Obesity
2	49	Female	Northeast	Married	2	Normal
3	30	Female	Southeast	Married	3	Normal
4	18	Male	Northeast	Unmarried	0	Overweight
5	56	Male	Northeast	Married	3	Obesity
6	33	Male	Southeast	Married	3	Normal
7	43	Male	Northeast	Married	3	Overweight
8	59	Female	Southeast	Unmarried	0	Overweight
9	22	Female	Northwest	Unmarried	0	Underweight

In [42]:

```
df2['income_level'] = df2['income_level'].map({'<10L':1, '10L - 25L': 2, '25L
df2.head(10)
```

Out[42]:

	age	gender	region	marital_status	number_of_dependants	bmi_category
0	26	Male	Northwest	Unmarried	0	Normal
1	29	Female	Southeast	Married	2	Obesity
2	49	Female	Northeast	Married	2	Normal
3	30	Female	Southeast	Married	3	Normal
4	18	Male	Northeast	Unmarried	0	Overweight
5	56	Male	Northeast	Married	3	Obesity
6	33	Male	Southeast	Married	3	Normal
7	43	Male	Northeast	Married	3	Overweight
8	59	Female	Southeast	Unmarried	0	Overweight
9	22	Female	Northwest	Unmarried	0	Underweight

The other columns are 'nominal', therefore need to do One-Hot-Encoding

In [43]:

```
# Applying One-Hot Encoding

nominal_cols = ['gender', 'region', 'marital_status', 'bmi_category', 'smoking
df3 = pd.get_dummies(df2, columns=nominal_cols, drop_first=True, dtype=int)

df3.head(3)
```

Out[43]:

	age	number_of_dependants	income_level	income_lakhs	medical_history	ins
0	26	0	1	6	Diabetes	
1	29	2	1	6	Diabetes	
2	49	2	2	20	High blood pressure	

3 rows × 23 columns

What pd.get_dummies() does :

- Creates new binary columns for each category

- Replaces text with 0 or 1

In [44]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 49908 entries, 0 to 49999
Data columns (total 23 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   age                              49908 non-null  int64  
 1   number_of_dependants             49908 non-null  int64  
 2   income_level                     49908 non-null  int64  
 3   income_lakhs                     49908 non-null  int64  
 4   medical_history                  49908 non-null  object 
 5   insurance_plan                   49908 non-null  int64  
 6   annual_premium_amount            49908 non-null  int64  
 7   disease1                         49908 non-null  object 
 8   disease2                         49908 non-null  object 
 9   total_risk_score                 49908 non-null  int64  
 10  normalized_risk_score            49908 non-null  float64
 11  gender_Male                      49908 non-null  int32  
 12  region_Northwest                 49908 non-null  int32  
 13  region_Southeast                 49908 non-null  int32  
 14  region_Southwest                 49908 non-null  int32  
 15  marital_status_Unmarried         49908 non-null  int32  
 16  bmi_category_Obesity             49908 non-null  int32  
 17  bmi_category_Overweight          49908 non-null  int32  
 18  bmi_category_Underweight         49908 non-null  int32  
 19  smoking_status_Occasional        49908 non-null  int32  
 20  smoking_status_Regular           49908 non-null  int32  
 21  employment_status_Salaried       49908 non-null  int32  
 22  employment_status_Self-Employed  49908 non-null  int32  
dtypes: float64(1), int32(12), int64(7), object(3)
memory usage: 6.9+ MB
```

4B - Feature Selection

In [45]:
```
# Dropping the unwanted columns

df4 = df3.drop(['medical_history','disease1', 'disease2', 'total_risk_score'],
df4.head(10)
```

Out[45]:

	age	number_of_dependants	income_level	income_lakhs	insurance_plan	an
0	26	0	1	6	1	
1	29	2	1	6	1	
2	49	2	2	20	2	
3	30	3	4	77	3	
4	18	0	4	99	2	
5	56	3	2	14	1	
6	33	3	1	4	2	
7	43	3	4	46	3	
8	59	0	2	21	3	
9	22	0	1	3	2	

4C - Scaling

I scale selected numeric features using MinMaxScaler to normalize their ranges, ensuring fair contribution to the regression model and stable optimization.

In [46]:

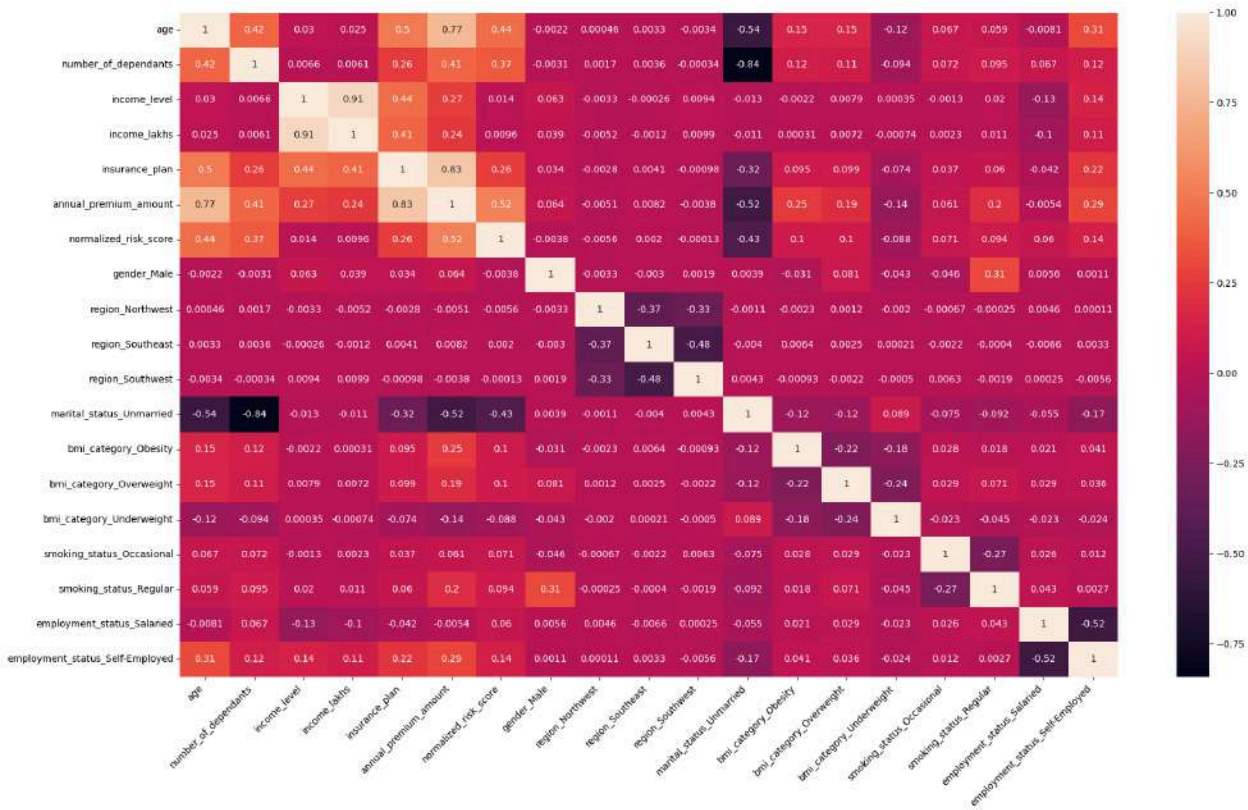
```
df4.columns
```

Out[46]:

```
Index(['age', 'number_of_dependants', 'income_level', 'income_lakhs',
       'insurance_plan', 'annual_premium_amount', 'normalized_risk_score',
       'gender_Male', 'region_Northwest', 'region_Southeast',
       'region_Southwest', 'marital_status_Unmarried', 'bmi_category_Obesit
y',
       'bmi_category_Overweight', 'bmi_category_Underweight',
       'smoking_status_Occasional', 'smoking_status_Regular',
       'employment_status_Salaried', 'employment_status_Self-Employed'],
      dtype='object')
```

In [47]:

```
cm = df4.corr()

plt.figure(figsize=(20,12))
sns.heatmap(cm, annot=True)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



In [48]:
```
X = df4.drop('annual_premium_amount', axis='columns')  # independent variables
y = df4['annual_premium_amount']                       # target variable (what

from sklearn.preprocessing import MinMaxScaler         # MinMaxScaler rescales

cols_to_scale = ['age','number_of_dependants', 'income_level',  'income_lakhs'
scaler = MinMaxScaler()

X[cols_to_scale] = scaler.fit_transform(X[cols_to_scale])
X.describe()
```

Out[48]:

	age	number_of_dependants	income_level	income_lakhs	insura
count	49908.000000	49908.000000	49908.000000	49908.000000	4990
mean	0.303733	0.343528	0.365900	0.221110	
std	0.253363	0.298406	0.349711	0.223946	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.074074	0.000000	0.000000	0.060606	
50%	0.240741	0.400000	0.333333	0.161616	
75%	0.500000	0.600000	0.666667	0.303030	
max	1.000000	1.000000	1.000000	1.000000	

4D - Calculating VIF for Multicollinearity

In [49]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(data):
    vif_df = pd.DataFrame()
    vif_df['Column'] = data.columns
    vif_df['VIF'] = [variance_inflation_factor(data.values,i) for i in range(c
    return vif_df
```

In [50]:
```
X.isna().sum().sort_values(ascending=False).head(20)
```

Out[50]:
```
age                                 0
number_of_dependants                0
employment_status_Salaried          0
smoking_status_Regular              0
smoking_status_Occasional           0
bmi_category_Underweight            0
bmi_category_Overweight             0
bmi_category_Obesity                0
marital_status_Unmarried            0
region_Southwest                    0
region_Southeast                    0
region_Northwest                    0
gender_Male                         0
normalized_risk_score               0
insurance_plan                      0
income_lakhs                        0
income_level                        0
employment_status_Self-Employed     0
dtype: int64
```

In [51]:
```
calculate_vif(X)
```

Out[51]:

	Column	VIF
0	age	4.567634
1	number_of_dependants	4.534650
2	income_level	12.450675
3	income_lakhs	11.183367
4	insurance_plan	3.584752
5	normalized_risk_score	2.687610
6	gender_Male	2.421496
7	region_Northwest	2.102556
8	region_Southeast	2.922414
9	region_Southwest	2.670666
10	marital_status_Unmarried	3.411185
11	bmi_category_Obesity	1.352806
12	bmi_category_Overweight	1.549922
13	bmi_category_Underweight	1.302886
14	smoking_status_Occasional	1.272745
15	smoking_status_Regular	1.777089
16	employment_status_Salaried	2.382134
17	employment_status_Self-Employed	2.137753

VIF for **income_level** and **income_lakhs** are pretty high.

Since VIF should be <10, therefore we need to drop **one(or both)** to get the VIFs of the remaining columns within the range

In [52]:

```
# Starting with 'income_level'

calculate_vif(X.drop('income_level', axis="columns"))
```

Out[52]:

	Column	VIF
0	age	4.545825
1	number_of_dependants	4.526598
2	income_lakhs	2.480563
3	insurance_plan	3.445682
4	normalized_risk_score	2.687326
5	gender_Male	2.409980
6	region_Northwest	2.100789
7	region_Southeast	2.919775
8	region_Southwest	2.668314
9	marital_status_Unmarried	3.393718
10	bmi_category_Obesity	1.352748
11	bmi_category_Overweight	1.549907
12	bmi_category_Underweight	1.302636
13	smoking_status_Occasional	1.272744
14	smoking_status_Regular	1.777024
15	employment_status_Salaried	2.374628
16	employment_status_Self-Employed	2.132810

Since now the VIF is <5 for all columns, therefore, no need to drop 'income_lakhs'

In [53]:

```
X_reduced = X.drop('income_level', axis="columns")

X_reduced.head(10)
```

Out[53]:

	age	number_of_dependants	income_lakhs	insurance_plan	normalized_r
0	0.148148	0.0	0.050505	0.0	
1	0.203704	0.4	0.050505	0.0	
2	0.574074	0.4	0.191919	0.5	
3	0.222222	0.6	0.767677	1.0	
4	0.000000	0.0	0.989899	0.5	
5	0.703704	0.6	0.131313	0.0	
6	0.277778	0.6	0.030303	0.5	
7	0.462963	0.6	0.454545	1.0	
8	0.759259	0.0	0.202020	1.0	
9	0.074074	0.0	0.020202	0.5	

5. Model Training

In [54]:

```
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size =


# shape of the X_train, X_test, y_train, y_test features

print("x train: ",X_train.shape)
print("x test: ",X_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)
```

```
x train:  (34935, 17)
x test:  (14973, 17)
y train:  (34935,)
y test:  (14973,)
```

5A - Linear Regression Model

In [55]:

```
model_lr = LinearRegression()    # Initializes a Linear Regression model
model_lr.fit(X_train, y_train)   # It finds the best-fit line that minimizes e

test_score = model_lr.score(X_test, y_test)
train_score = model_lr.score(X_train, y_train)

train_score, test_score
```

Out[55]: (0.9282143576916762, 0.9280547230217837)

In [56]:
```python
# The below code checks how accurate your model’s predictions are by comparing

y_pred = model_lr.predict(X_test)

mse_lr = mean_squared_error(y_test, y_pred)
rmse_lr = np.sqrt(mse_lr)
print("Linear Regression ==> MSE: ", mse_lr, "RMSE: ", rmse_lr)
```

```
Linear Regression ==> MSE:  5165611.913027982 RMSE:  2272.798256121291
```

In [57]:
```python
X_test.shape
```

Out[57]:
```
(14973, 17)
```

In [58]:
```python
np.set_printoptions(suppress=True, precision=6)
model_lr.coef_
```

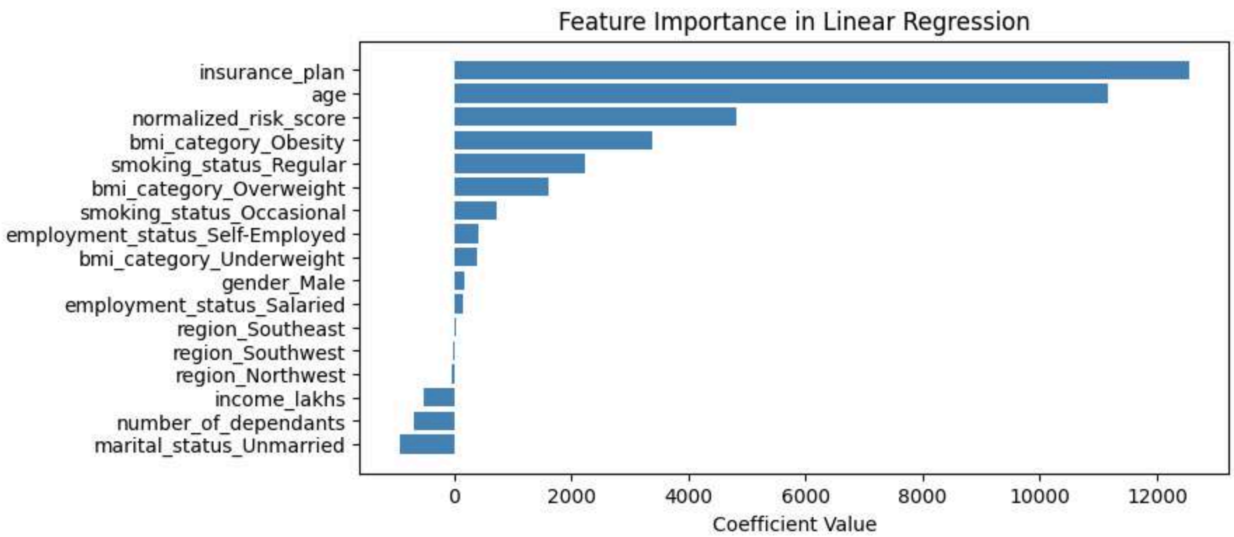
Out[58]:
```
array([11160.926462,  -676.443991,  -514.229816, 12557.012936,
        4810.357702,   168.646662,   -35.719292,    39.96513 ,
         -24.652929,  -935.760611,  3387.911455,  1599.362268,
         391.171304,   735.912278,  2234.804712,   155.984674,
         415.903973])
```

The above code tells Python how to display numbers, and then shows how much each feature affects the prediction in your Linear Regression model.

For ex :

Beacuse of **suppress=True** → the output don’t show scientific notation (Shows 0.000123 instead of 1.23e-4)

In [59]:
```python
feature_importance = model_lr.coef_

# Create a DataFrame for easier handling
coef_df = pd.DataFrame(feature_importance, index=X_train.columns, columns=['Co

# Sort the coefficients for better visualization
coef_df = coef_df.sort_values(by='Coefficients', ascending=True)

# Plotting
plt.figure(figsize=(8, 4))
plt.barh(coef_df.index, coef_df['Coefficients'], color='steelblue')
plt.xlabel('Coefficient Value')
plt.title('Feature Importance in Linear Regression')
plt.show()
```



The above code shows which features influence the prediction the most by turning model coefficients into a clear horizontal bar chart.

Since we have around 92% of model accuracy, we need to get it higher. Therefore, we need to look into other ways to do so.

Out of which, first will be **Ridge Regression Model**.

5B - Ridge Regression Model

In [60]:
```
model_rg = Ridge(alpha=1)

model_rg.fit(X_train, y_train)
test_score = model_rg.score(X_test, y_test)
train_score = model_rg.score(X_train, y_train)
train_score, test_score
```

Out[60]: (0.9282143198366275, 0.9280541644640345)

In [61]:
```
model_rg = Ridge(alpha=10)

model_rg.fit(X_train, y_train)
test_score_rg = model_rg.score(X_test, y_test)
train_score_rg = model_rg.score(X_train, y_train)

train_score_rg, test_score_rg
```

Out[61]: (0.9282106074563636, 0.9280459054997704)

In [62]:
```
model_rg = Ridge(alpha=30)

model_rg.fit(X_train, y_train)
```

```
test_score = model_rg.score(X_test, y_test)
train_score = model_rg.score(X_train, y_train)
train_score, test_score
```

Out[62]: (0.9281812886908575, 0.9280073438795297)

Not a very noticeable change, therefore, switching to another : **XG Boost**

5C - XG Boost

In [63]:
```
from xgboost import XGBRegressor

model_xgb = XGBRegressor(n_estimators=20, max_depth=3)
model_xgb.fit(X_train, y_train)
model_xgb.score(X_test, y_test)
```

Out[63]: 0.9782300591468811

In [64]:
```
y_pred = model_rg.predict(X_test)

mse_lr = mean_squared_error(y_test, y_pred)
rmse_lr = np.sqrt(mse_lr)
print("Ridge Regression ==> MSE: ", mse_lr, "RMSE: ", rmse_lr)
```

```
Ridge Regression ==> MSE:  5169013.69660484 RMSE:  2273.546501966661
```

In [65]:
```
y_pred = model_xgb.predict(X_test)

mse_lr = mean_squared_error(y_test, y_pred)
rmse_lr = np.sqrt(mse_lr)
print("XGBoost Regression ==> MSE: ", mse_lr, "RMSE: ", rmse_lr)
```

```
XGBoost Regression ==> MSE:  1563064.1356043513 RMSE:  1250.2256338774819
```

The above comparison shows that : On average, **Ridge Regression's** predictions are off by about ₹2,273, but **XGBoost's** predictions are off by about ₹1,250.

That means : **XGBoost reduces error by ~₹1,000 per prediction and that's a big improvement in real terms.**

WHY go with XG Boost ??

As we compared Ridge Regression and XGBoost using RMSE, we see that XGBoost reduced prediction error by nearly 45% compared to Ridge, indicating that the relationship between features and premium is non-linear and better captured by tree-based models.

5D - Setting up Randomized Search

WHAT

I'm using **RandomizedSearchCV to tune XGBoost** hyperparameters by testing multiple configurations with cross-validation and selecting the model with the best R^2 score.

In [66]:
```
model_xgb = XGBRegressor()            # creates a basic XGBoost model
param_grid = {                        # Try different combinations of these ar
    'n_estimators': [20, 40, 50],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
}
random_search = RandomizedSearchCV(model_xgb, param_grid, n_iter=10, cv=3, sco
random_search.fit(X_train, y_train)
random_search.best_score_            # The best average R² score achieved duri
```

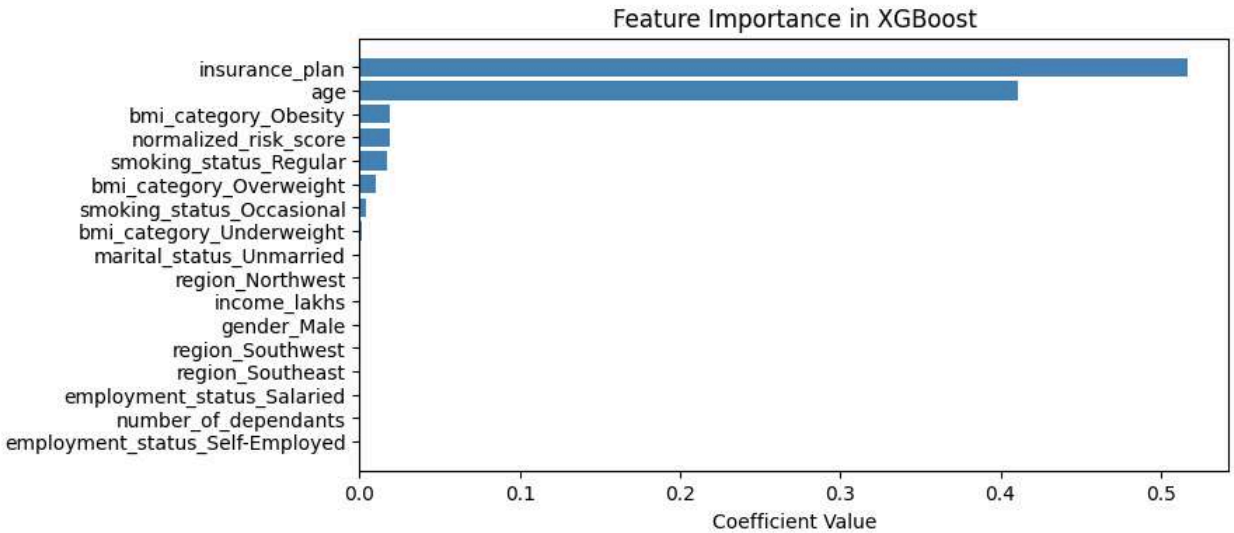
Out[66]:
```
0.9809474547704061
```

In [67]:
```
random_search.best_params_      # it gives the best possible combination to ha
```

Out[67]:
```
{'n_estimators': 50, 'max_depth': 5, 'learning_rate': 0.1}
```

In [68]:
```
best_model = random_search.best_estimator_
```

In [69]:
```
feature_importance = best_model.feature_importances_

# Create a DataFrame for easier handling
coef_df = pd.DataFrame(feature_importance, index=X_train.columns, columns=['Co

# Sort the coefficients for better visualization
coef_df = coef_df.sort_values(by='Coefficients', ascending=True)

# Plotting
plt.figure(figsize=(8, 4))
plt.barh(coef_df.index, coef_df['Coefficients'], color='steelblue')
plt.xlabel('Coefficient Value')
plt.title('Feature Importance in XGBoost')
plt.show()
```



Feature Importance in **XG Boost** may not be as straightforward to interpret as Feature Importance in **Linear Regression**, since XG Boost is a **Tree-Based Model.**

5E - Error Analysis

In [70]:
```
y_pred = best_model.predict(X_test)

residuals = y_pred - y_test
residuals_pct = (residuals / y_test) * 100

results_df = pd.DataFrame({
    'actual': y_test,
    'predicted': y_pred,
    'diff': residuals,
    'diff_pct': residuals_pct
})
results_df.head(10)
```

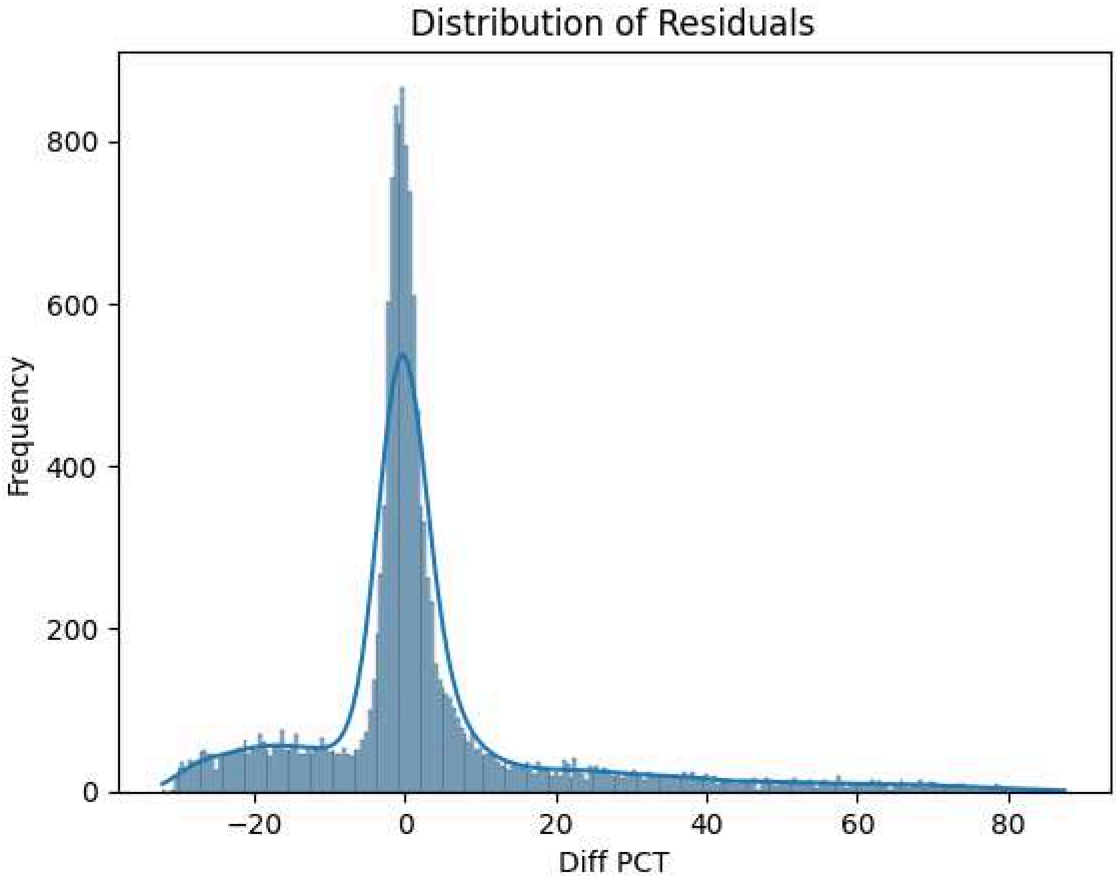
Out[70]:

	actual	predicted	diff	diff_pct
3598	20554	20334.953125	-219.046875	-1.065714
35794	29647	29378.779297	-268.220703	-0.904714
43608	20560	20618.185547	58.185547	0.283004
42730	5018	7352.829590	2334.829590	46.529087
18936	8929	8203.291992	-725.708008	-8.127540
45416	9892	10230.427734	338.427734	3.421227
20029	5140	6670.849121	1530.849121	29.783057
4294	9631	7053.477539	-2577.522461	-26.762771
39145	18777	18649.619141	-127.380859	-0.678388
3152	14536	13996.699219	-539.300781	-3.710104

We can see above that for '42730' and '20029', the diff_pct is higher, which is not good

In [71]:

```
# Visualizing the diff_pct

sns.histplot(results_df['diff_pct'], kde=True)
plt.title('Distribution of Residuals')
plt.xlabel('Diff PCT')
plt.ylabel('Frequency')
plt.show()
```



There are differences that shows error upto 80%

In [72]:
```
X_test.shape
```

Out[72]: (14973, 17)

In [73]:
```
# Taking out the records having 'diff_pct' >10

extreme_error_threshold = 10

extreme_results_df = results_df[np.abs(results_df['diff_pct']) > extreme_error
extreme_results_df.head(10)
```

Out[73]:

	actual	predicted	diff	diff_pct
42730	5018	7352.829590	2334.829590	46.529087
20029	5140	6670.849121	1530.849121	29.783057
4294	9631	7053.477539	-2577.522461	-26.762771
44419	4687	6670.849121	1983.849121	42.326629
6707	8826	10047.326172	1221.326172	13.837822
11728	4796	6565.852051	1769.852051	36.902670
15740	9045	10047.326172	1002.326172	11.081550
35065	5929	6820.886230	891.886230	15.042777
9654	5927	8212.278320	2285.278320	38.557083
22679	8482	7058.687988	-1423.312012	-16.780382

In [74]:

```
extreme_results_df.shape     # total records having having 'diff_pct' >10
```

Out[74]: (4487, 4)

In [75]:

```
results_df.shape
```

Out[75]: (14973, 4)

In [76]:

```
extreme_errors_pct = extreme_results_df.shape[0]*100/X_test.shape[0]
round(extreme_errors_pct, 3)
```

Out[76]: 29.967

30% of the predictions are extreme error (very wrong).

That means, for 30% customers, we will either overcharge or undercharge by 10% or more.

In [77]:

```
extreme_results_df[abs(extreme_results_df.diff_pct)>50].sort_values("diff_pct"
```

Out[77]:

	actual	predicted	diff	diff_pct
36269	3501	6565.852051	3064.852051	87.542189
48801	3516	6565.852051	3049.852051	86.742095
42342	3521	6565.852051	3044.852051	86.476911
18564	3523	6565.852051	3042.852051	86.371049
7988	3527	6565.852051	3038.852051	86.159684
...	...	...	...	...
32671	4656	6994.980957	2338.980957	50.235845
14798	4371	6565.852051	2194.852051	50.213957
13736	4371	6565.852051	2194.852051	50.213957
10107	4710	7073.240723	2363.240723	50.174962
16908	4699	7053.477539	2354.477539	50.105928

549 rows × 4 columns

There will be about 549 customers whom we will overcharge or underchage by more than 50%

5F - More Analysis

In [78]:

```
extreme_errors_df = X_test.loc[extreme_results_df.index]    # rows that have h
extreme_errors_df.head(5)
```

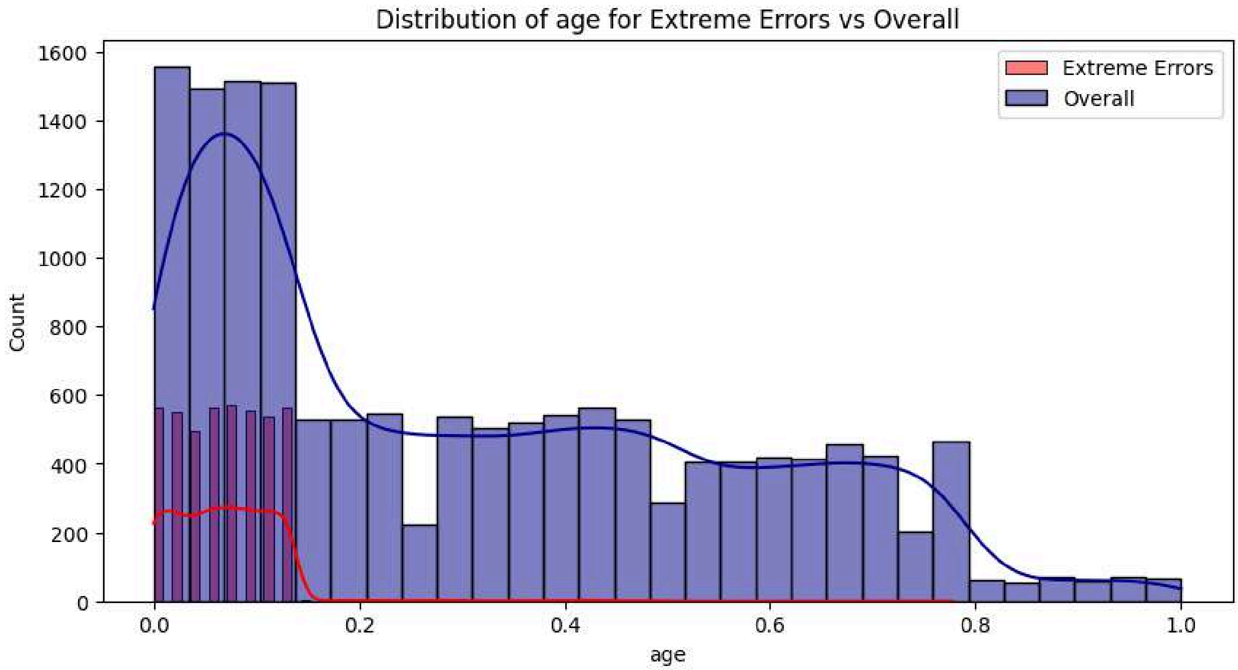
Out[78]:

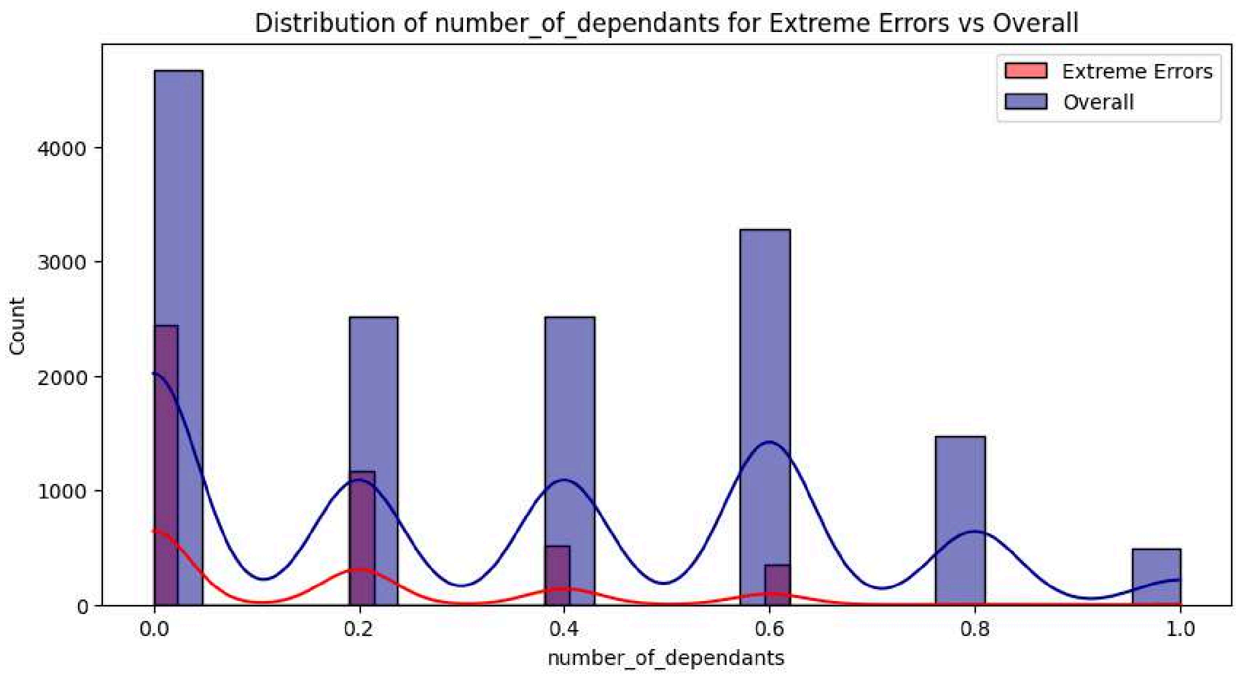
	age	number_of_dependants	income_lakhs	insurance_plan	normali
42730	0.092593	0.2	0.131313	0.0	
20029	0.018519	0.2	0.030303	0.0	
4294	0.000000	0.2	0.020202	0.0	
44419	0.055556	0.0	0.242424	0.0	
6707	0.111111	0.2	0.070707	0.5	

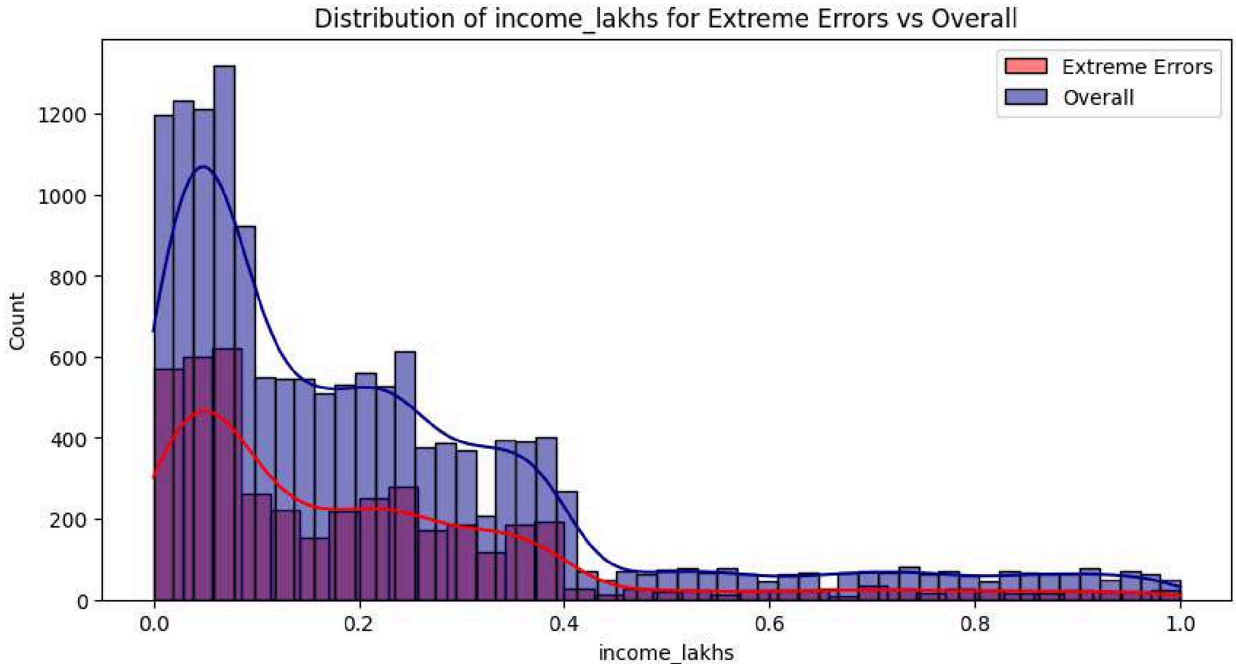
In [79]:

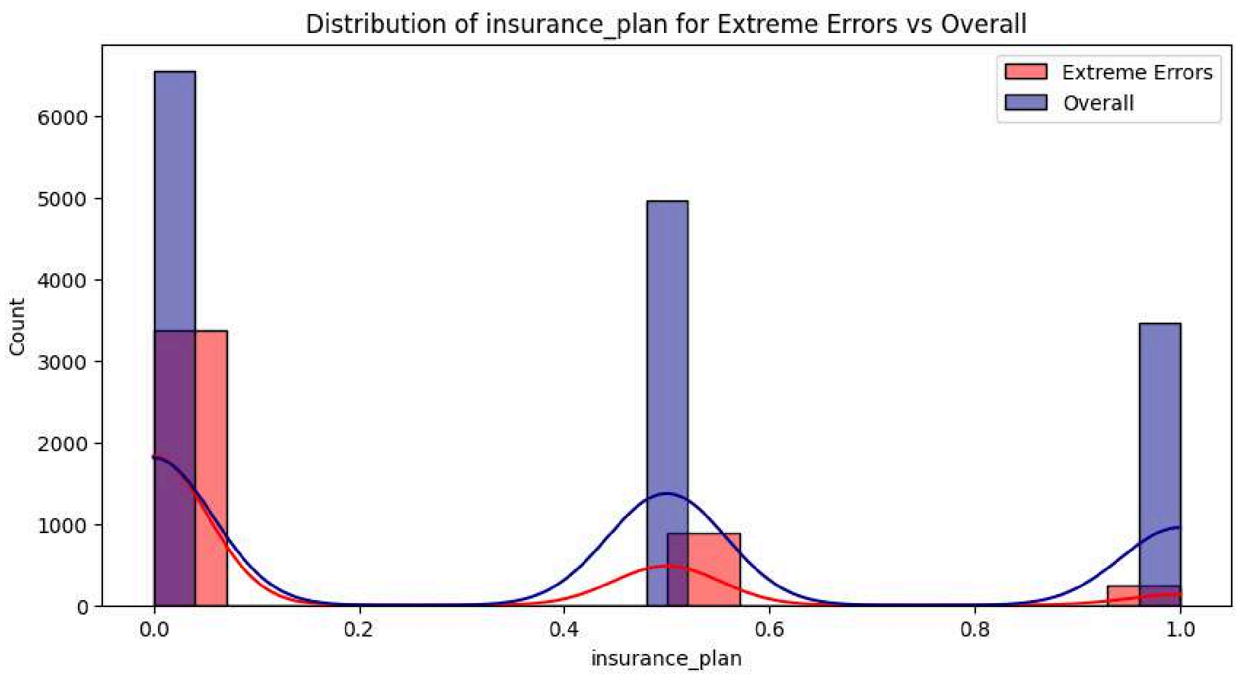
```
for feature in X_test.columns:
    plt.figure(figsize=(10, 5))

    sns.histplot(extreme_errors_df[feature], color='red', label='Extreme Error
    sns.histplot(X_test[feature], color='darkblue', label='Overall', alpha=0.5

    plt.legend()
```

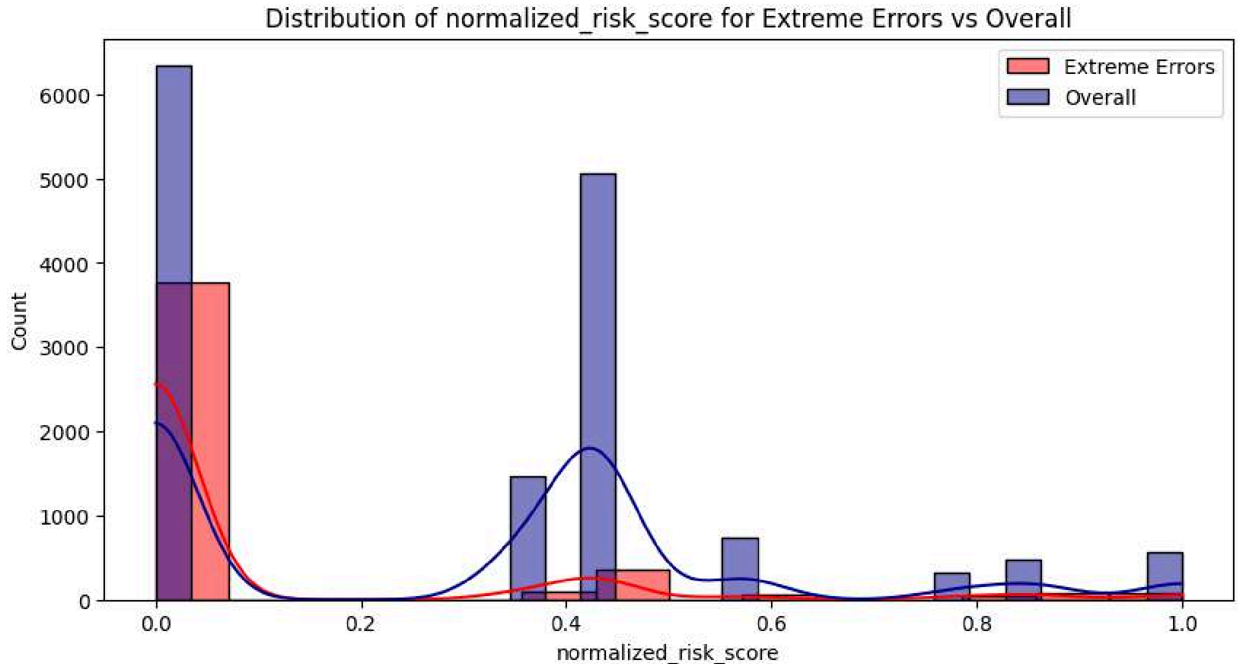
```
    plt.title(f'Distribution of {feature} for Extreme Errors vs Overall')
    plt.show()
```

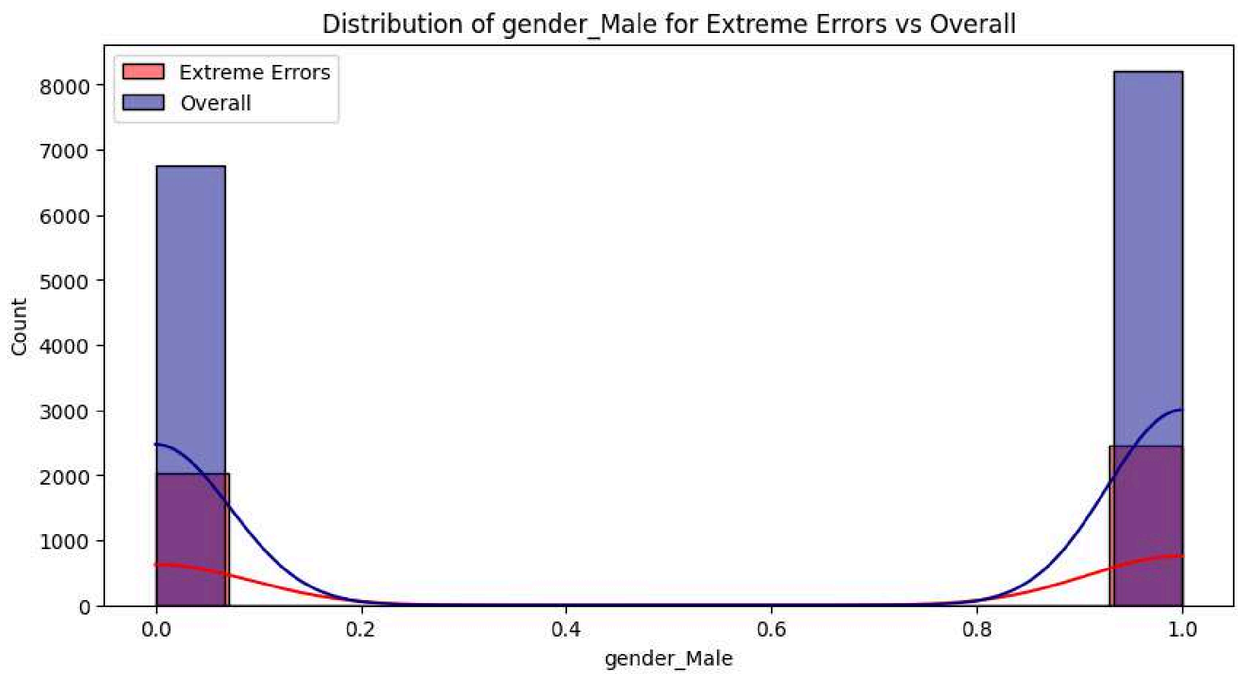


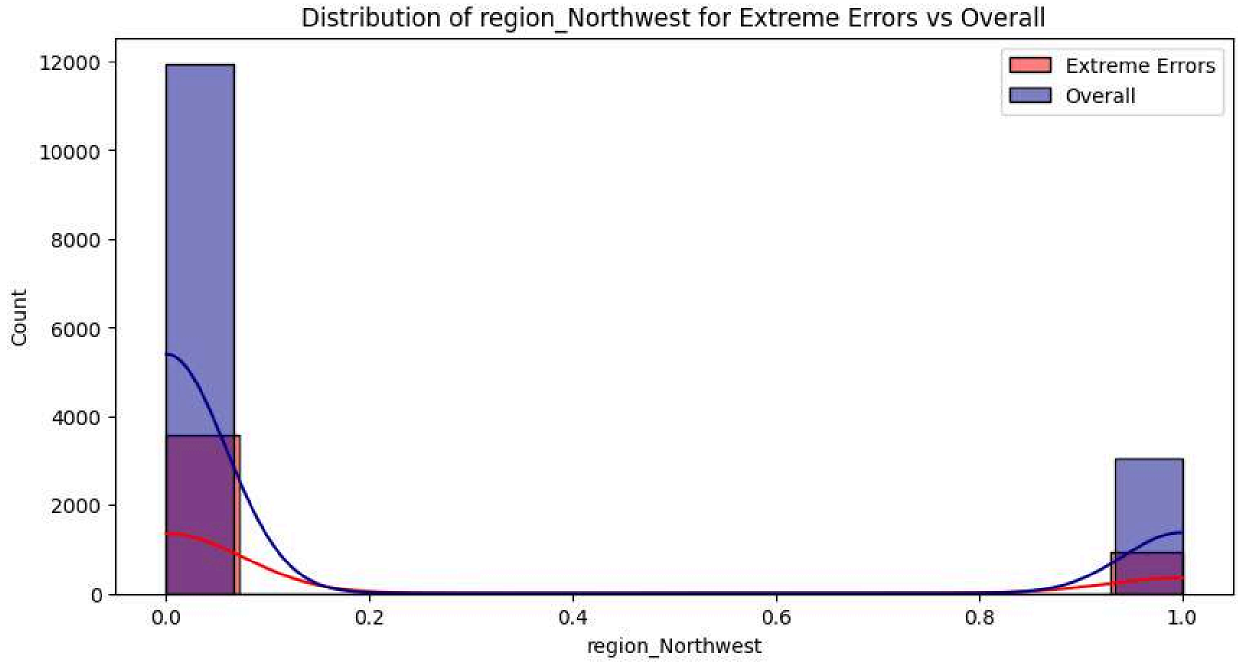


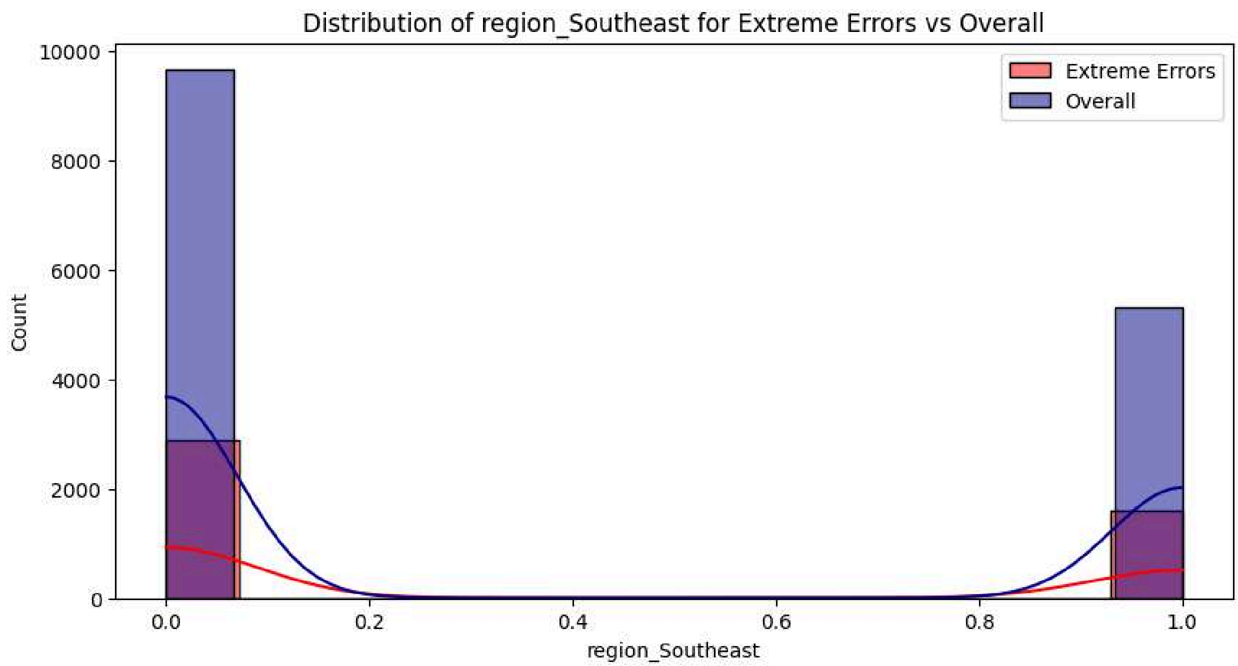

Distribution of income_lakhs for Extreme Errors vs Overall
Extreme Errors
Overall
Count
income_lakhs
0
200
400
600
800
1000
1200
0.0
0.2
0.4
0.6
0.8
1.0

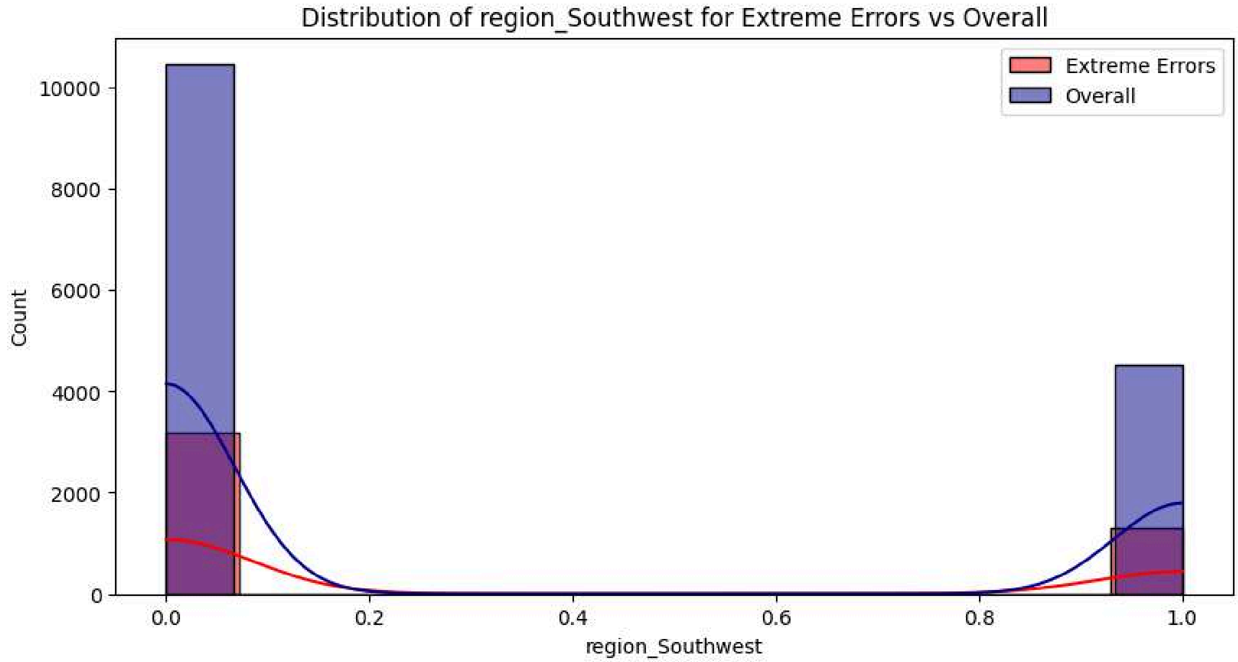

Distribution of insurance_plan for Extreme Errors vs Overall
Extreme Errors
Overall
Count
insurance_plan
0
1000
2000
3000
4000
5000
6000
0.0
0.2
0.4
0.6
0.8
1.0

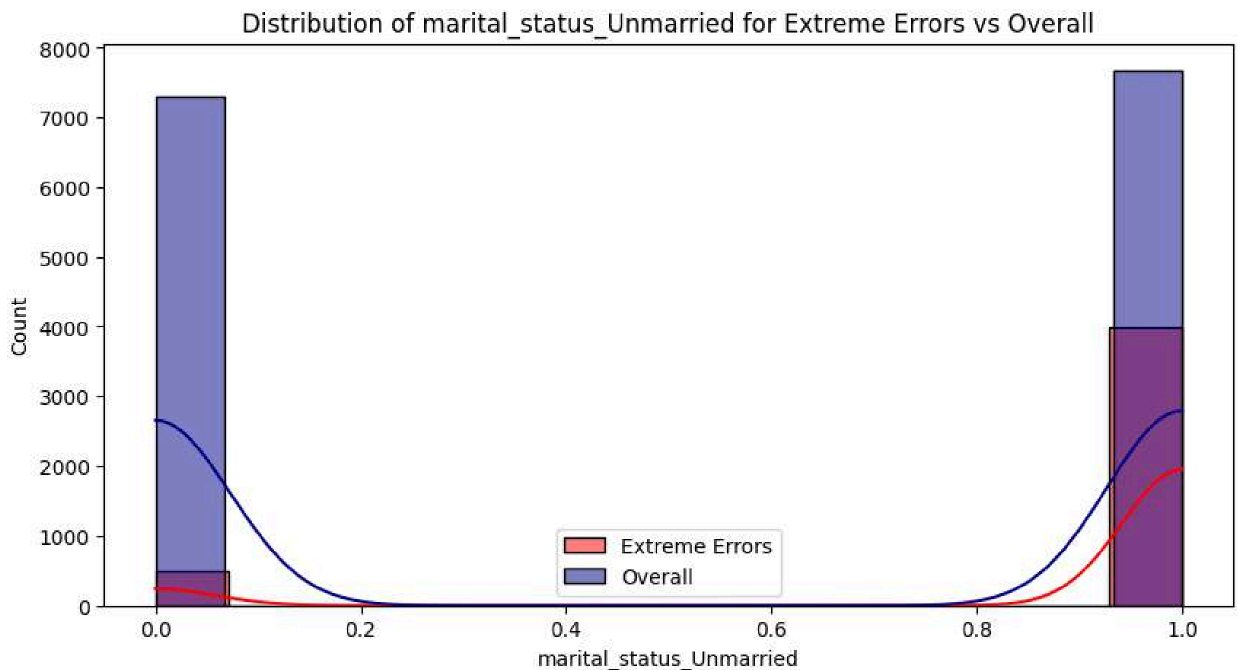

Distribution of normalized_risk_score for Extreme Errors vs Overall
Extreme Errors
Overall
Count
normalized_risk_score
0
1000
2000
3000
4000
5000
6000
0.0
0.2
0.4
0.6
0.8
1.0

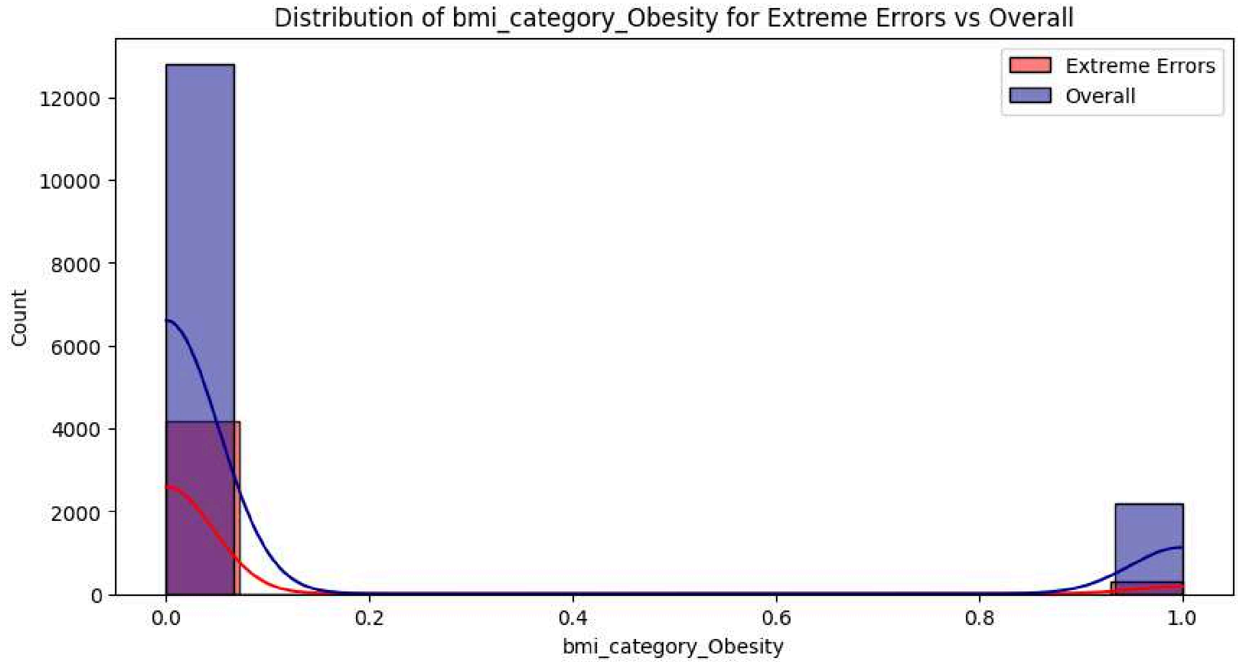

Distribution of gender_Male for Extreme Errors vs Overall
Extreme Errors
Overall
Count
gender_Male
0
1000
2000
3000
4000
5000
6000
7000
8000
0.0
0.2
0.4
0.6
0.8
1.0

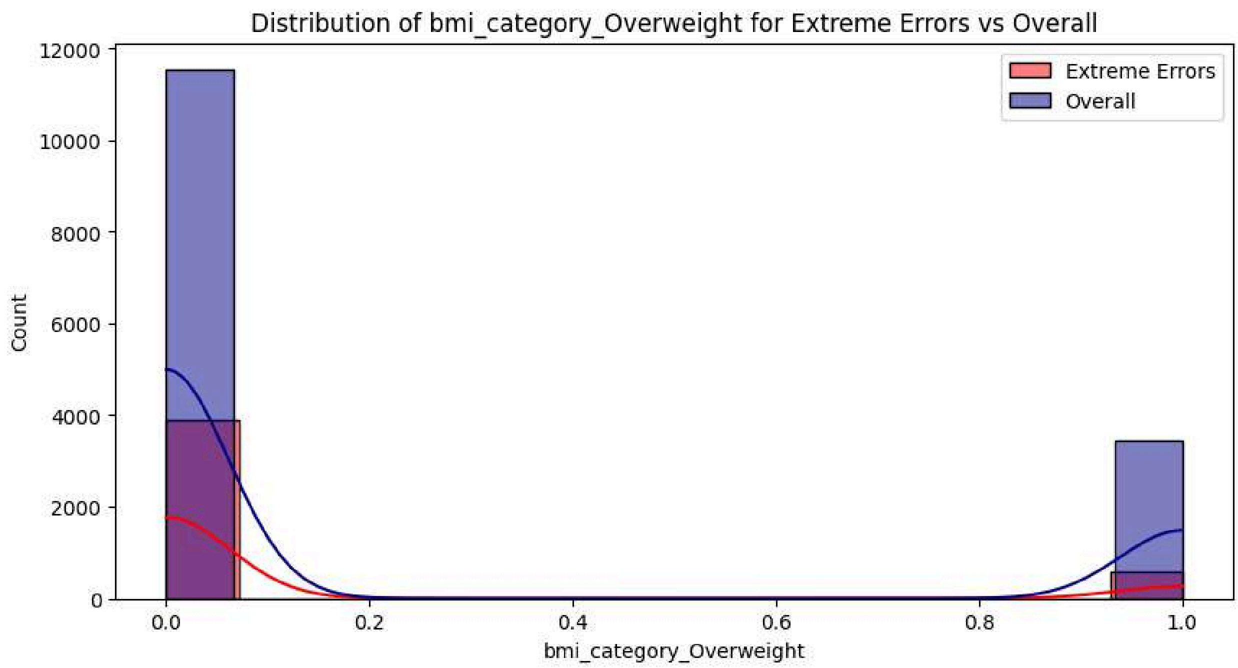

Distribution of region_Northwest for Extreme Errors vs Overall
Extreme Errors
Overall
Count
region_Northwest
0
2000
4000
6000
8000
10000
12000
0.0
0.2
0.4
0.6
0.8
1.0

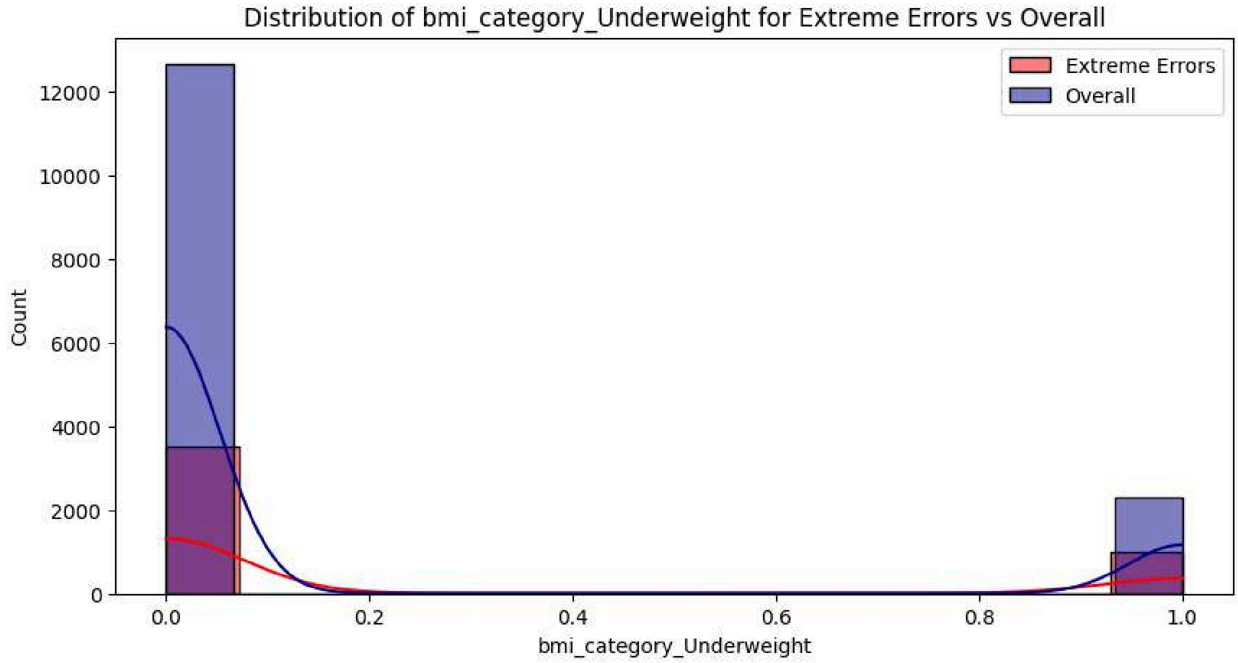

Distribution of region_Southeast for Extreme Errors vs Overall
Extreme Errors
Overall
Count
region_Southeast
0
2000
4000
6000
8000
10000
0.0
0.2
0.4
0.6
0.8
1.0

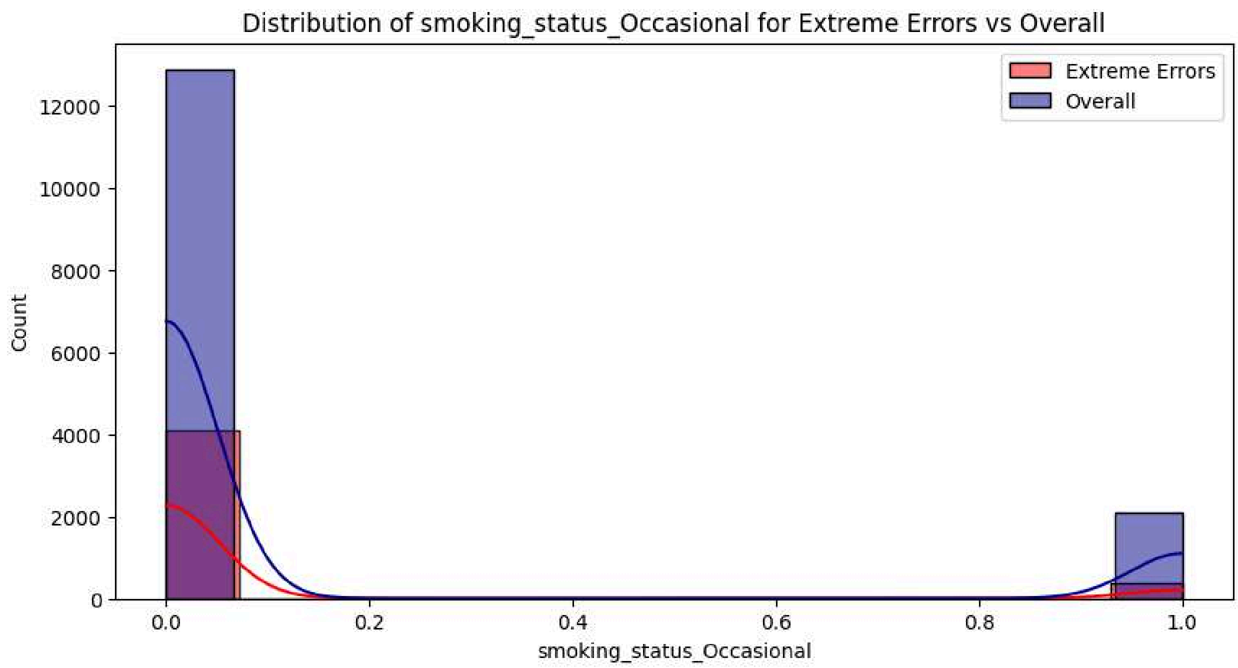

Distribution of region_Southwest for Extreme Errors vs Overall
Extreme Errors
Overall
Count
0
2000
4000
6000
8000
10000
0.0
0.2
0.4
0.6
0.8
1.0
region_Southwest

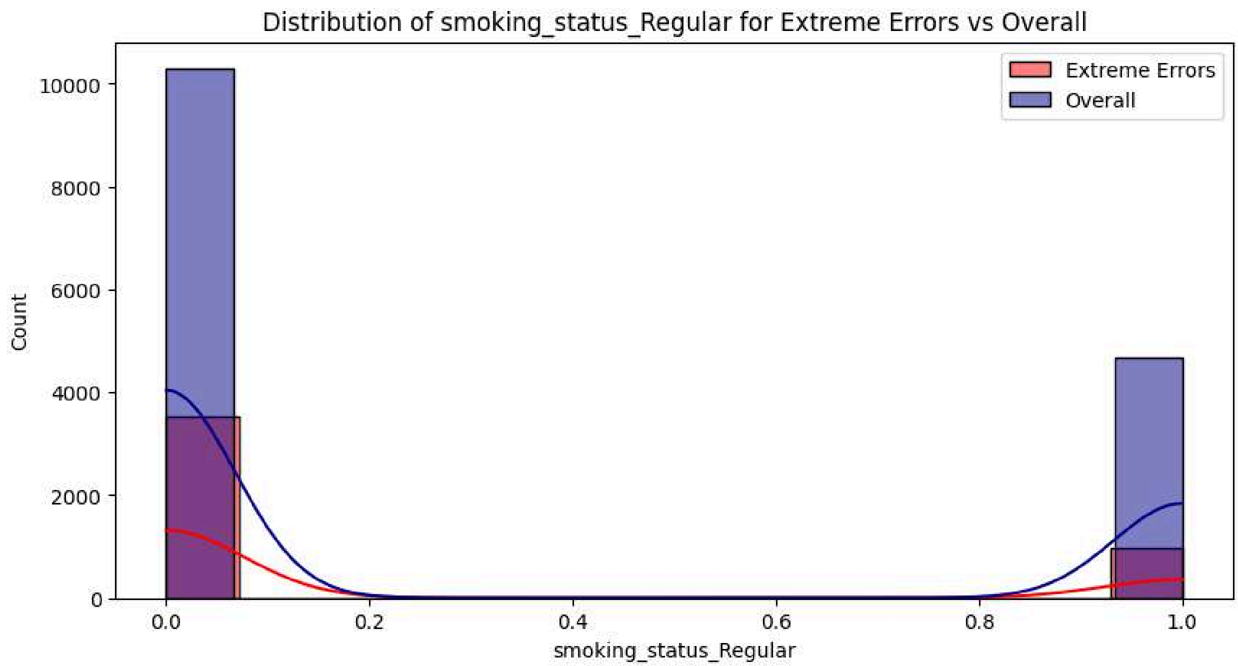

Distribution of marital_status_Unmarried for Extreme Errors vs Overall
Extreme Errors
Overall
Count
0
1000
2000
3000
4000
5000
6000
7000
8000
0.0
0.2
0.4
0.6
0.8
1.0
marital_status_Unmarried

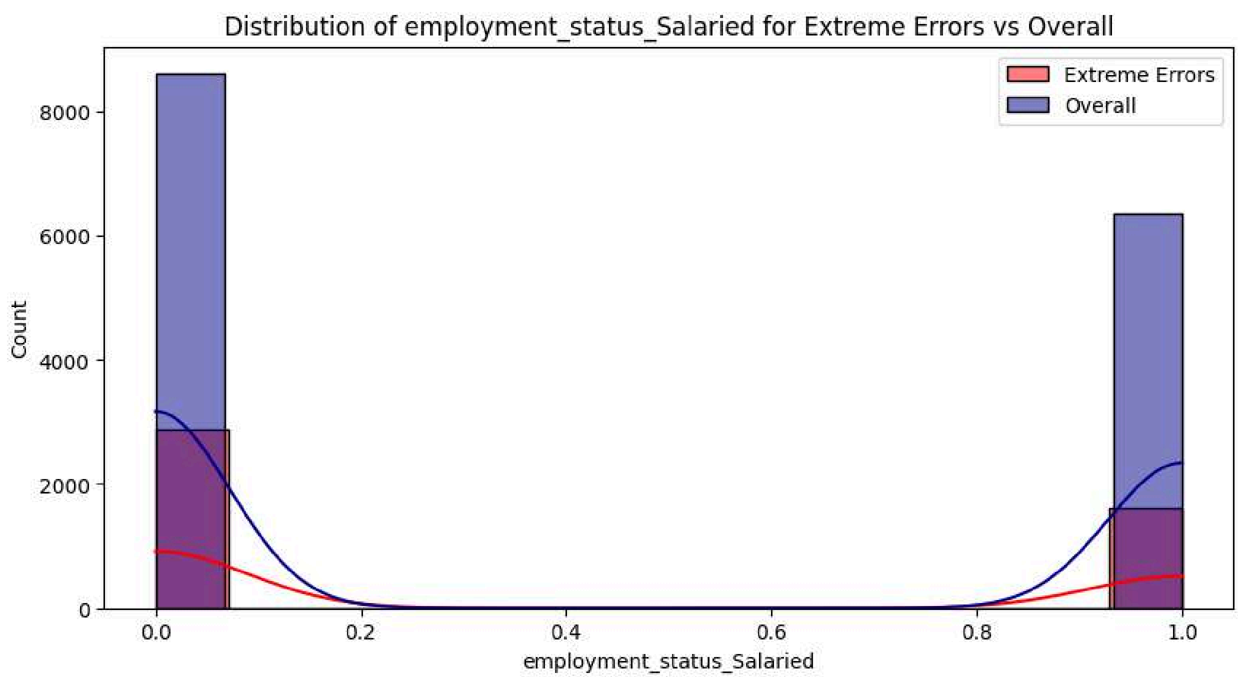

Distribution of bmi_category_Obesity for Extreme Errors vs Overall
Extreme Errors
Overall
Count
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
bmi_category_Obesity

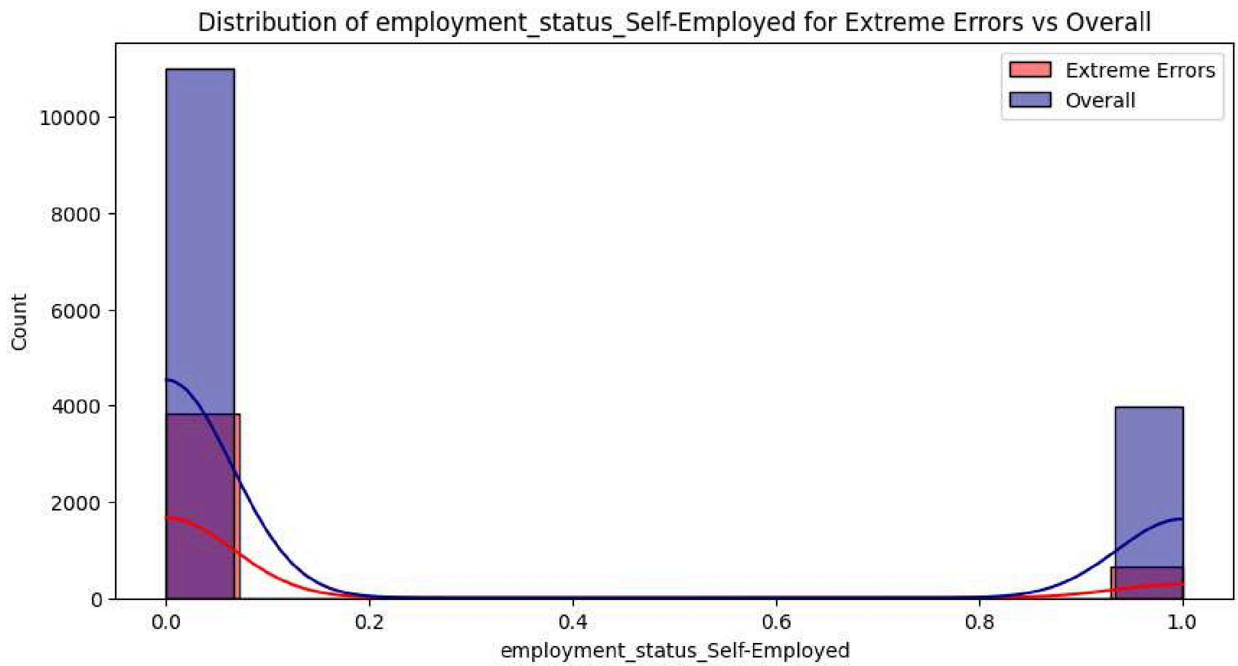

Distribution of bmi_category_Overweight for Extreme Errors vs Overall
Extreme Errors
Overall
Count
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
bmi_category_Overweight


Distribution of bmi_category_Underweight for Extreme Errors vs Overall
Extreme Errors
Overall
Count
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
bmi_category_Underweight


Distribution of smoking_status_Occasional for Extreme Errors vs Overall
Extreme Errors
Overall
Count
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
smoking_status_Occasional


Distribution of smoking_status_Regular for Extreme Errors vs Overall
Extreme Errors
Overall
Count
0
2000
4000
6000
8000
10000
0.0
0.2
0.4
0.6
0.8
1.0
smoking_status_Regular


Distribution of employment_status_Salaried for Extreme Errors vs Overall
Extreme Errors
Overall
Count
0
2000
4000
6000
8000
0.0
0.2
0.4
0.6
0.8
1.0
employment_status_Salaried



Why this analysis is IMPORTANT ??

This tells you:

- Where the model breaks
- Which features cause trouble
- What to fix next

Looking into the above **feature** insights, we can see that the distribution (here it is path of red & blue line) is almost similar in every features except the **age**, and to be specific - **lower range of age**.

5G - Reverse Scaling

In [80]:
```
extreme_errors_df['income_level']=-1    # since we dropped this columns previc
```

In [81]:
```
df_reversed = pd.DataFrame()
df_reversed[cols_to_scale] = scaler.inverse_transform(extreme_errors_df[cols_t
df_reversed.head(10)
```

Out[81]:

	age	number_of_dependants	income_level	income_lakhs	insurance_plan
0	23.0	1.0	-2.0	14.0	1.0
1	19.0	1.0	-2.0	4.0	1.0
2	18.0	1.0	-2.0	3.0	1.0
3	21.0	0.0	-2.0	25.0	1.0
4	24.0	1.0	-2.0	8.0	2.0
5	19.0	0.0	-2.0	7.0	1.0
6	22.0	1.0	-2.0	9.0	2.0
7	20.0	1.0	-2.0	28.0	1.0
8	25.0	0.0	-2.0	8.0	1.0
9	25.0	0.0	-2.0	8.0	1.0

In [82]:

```python
df_reversed.describe()
```

Out[82]:

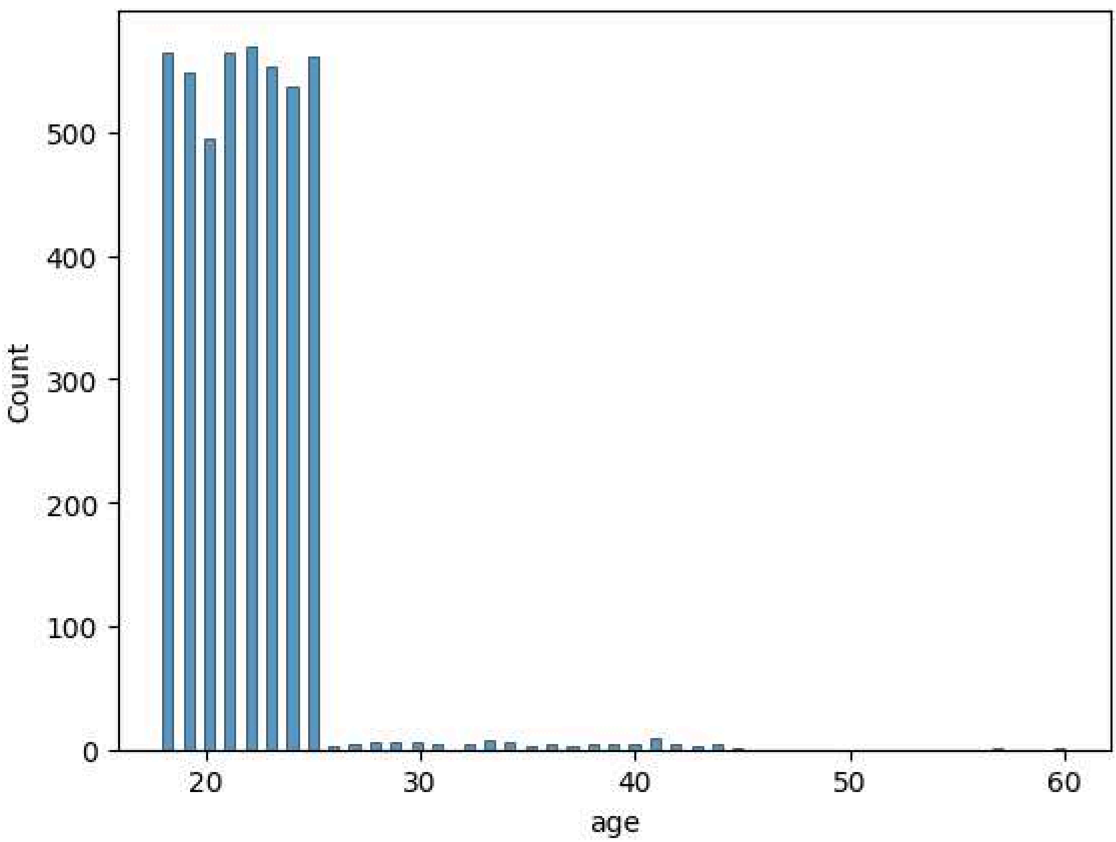
	age	number_of_dependants	income_level	income_lakhs	insurar
count	4487.000000	4487.000000	4487.0	4487.000000	448
mean	21.804992	0.739247	-2.0	21.182527	1
std	3.172355	0.968855	0.0	20.598596	0
min	18.000000	0.000000	-2.0	1.000000	1
25%	20.000000	0.000000	-2.0	6.000000	1
50%	22.000000	0.000000	-2.0	15.000000	1
75%	24.000000	1.000000	-2.0	30.000000	1
max	60.000000	5.000000	-2.0	100.000000	3

We can see above that 75% of 'age' is <= 24

In [83]:

```python
sns.histplot(df_reversed.age)
```

Out[83]: <Axes: xlabel='age', ylabel='Count'>



In [84]:
```
df_reversed['age'].quantile(0.97)
```

Out[84]:
```
25.0
```

This shows that majority (around 97%) of the **extreme errors** are coming from **young age group (i.e. <=25 years of age)**.

We need to may be build a separate model for this segment.

6. Model Segmentation

```
df_youth = df_m[df_m.Age<=25]

df_mature = df_m[df_m.Age>25]

df_youth.to_excel("prem_youth.xlsx", index = False)

df_mature.to_excel("prem_mature.xlsx", index = False)
```

7. Model Retraining

- Divided the data into 2 parts : **Youth**(age <= 25) and **Mature**(age > 25)
- Added the additional data that displays **Genetic Risk**, for better model training
- Created an artifact folder that has **model & scaler** joblib file for the **youth & mature**

8. Building The App

Why Streamlit is popular ??

- Extremely easy
- Fast to build
- Perfect for ML demos
- Recruiter-friendly
- No web dev skills needed

That’s why it’s the #1 tool for ML portfolios.

What Streamlit is commonly used for ??

- ML model demos
- Data dashboards
- What-if analysis tools
- Internal business tools

9. Deployed The App

https://insurance-premium-aiml.streamlit.app/