

week1_premium_estimator_main

June 23, 2025

```
[6]: import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split, KFold, \
    cross_val_score, cross_validate, RandomizedSearchCV
from sklearn.metrics import mean_squared_error, r2_score, root_mean_squared_error
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from time import time
```

0.1 Data Import & Exploration

```
[7]: # Switching to the source directory

parent_dir = os.path.abspath(os.path.join(os.getcwd(), "../.."))
parent_dir
print(parent_dir)
```

C:\Users\91948\Downloads\BKs\Projects\personalized_health_insurance_premium_estimator

```
[8]: # Reading the data

df = pd.read_excel(fr'{parent_dir}\data\premiums.xlsx', sheet_name='Sheet1')
```

```
[9]: # Displaying the data

df.head()
```

```
[9]:   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

	Smoking_Status	Employment_Status	Income_Level	Income_Lakhs	\
0	No Smoking	Salaried	<10L	6	
1	Regular	Salaried	<10L	6	
2	No Smoking	Self-Employed	10L - 25L	20	
3	No Smoking	Salaried	> 40L	77	
4	Regular	Self-Employed	> 40L	99	

	Medical History	Insurance_Plan	Annual_Premium_Amount
0	Diabetes	Bronze	9053
1	Diabetes	Bronze	16339
2	High blood pressure	Silver	18164
3	No Disease	Gold	20303
4	High blood pressure	Silver	13365

```
[10]: df.shape
```

```
[10]: (50000, 13)
```

```
[11]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 13 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Age                                   50000 non-null  int64
1   Gender                               50000 non-null  object
2   Region                               50000 non-null  object
3   Marital_status                       50000 non-null  object
4   Number Of Dependants                 50000 non-null  int64
5   BMI_Category                         50000 non-null  object
6   Smoking_Status                       49989 non-null  object
7   Employment_Status                   49998 non-null  object
8   Income_Level                         49987 non-null  object
9   Income_Lakhs                        50000 non-null  int64
10  Medical History                      50000 non-null  object
11  Insurance_Plan                       50000 non-null  object
12  Annual_Premium_Amount                50000 non-null  int64
dtypes: int64(4), object(9)
memory usage: 5.0+ MB
```

0.2 Data Cleaning

The following steps were performed during data cleaning:

- Handling missing values

- Removing duplicated rows
- Formatting numeric and categorical values
- Treating outliers to improve data quality

0.2.1 Column Formatter

```
[12]: # Renaming the columns with proper formatter
# Eg: Number Of Dependants -> number_of_dependants

df.columns = df.columns.str.replace(' ', '_').str.lower()
```

```
[13]: df.head()
```

```
[13]:   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
```

```
      smoking_status  employment_status  income_level  income_lakhs \
0      No Smoking          Salaried      <10L             6
1      Regular          Salaried      <10L             6
2      No Smoking      Self-Employed    10L - 25L            20
3      No Smoking          Salaried      > 40L            77
4      Regular      Self-Employed      > 40L            99
```

```
      medical_history  insurance_plan  annual_premium_amount
0      Diabetes      Bronze          9053
1      Diabetes      Bronze          16339
2  High blood pressure      Silver          18164
3      No Disease      Gold          20303
4  High blood pressure      Silver          13365
```

0.2.2 Handling Missing Values

```
[14]: # Finding the columns with NA values

df.isna().sum()
```

```
[14]: 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

```

```
[15]: # Dropping the Rows where it has NA value
```

```

# Before Dropping
df[df['smoking_status'].isna()].head()

```

```

[15]:      age  gender      region marital_status  number_of_dependants \
177    26  Female  Southwest      Married              3
15648  47   Male  Southwest      Married              4
16324  45   Male  Northwest      Married              4
16941  34   Male  Southwest      Married              5
16975  23   Male  Southwest      Unmarried             0

      bmi_category smoking_status employment_status income_level \
177    Underweight          NaN      Salaried      > 40L
15648      Normal          NaN      Freelancer  25L - 40L
16324  Overweight          NaN      Salaried  10L - 25L
16941      Normal          NaN  Self-Employed  25L - 40L
16975      Normal          NaN      Freelancer    <10L

      income_lakhs      medical_history insurance_plan \
177             69             Diabetes      Gold
15648            32             Diabetes      Gold
16324            16  High blood pressure & Heart disease  Silver
16941            35  High blood pressure & Heart disease    Gold
16975             3             No Disease    Bronze

      annual_premium_amount
177             22605
15648            26100
16324            21881
16941            25865
16975             6001

```

Since the count is low, the rows containing at least one missing value were dropped.

```

[16]: # Dropping
df.dropna(how='any',inplace = True)

```

```

[17]: # After Dropping
df[df['smoking_status'].isna()]

```

```
[17]: Empty DataFrame
      Columns: [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]
      Index: []
```

```
[18]: df.isna().sum()
```

```
[18]: 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
```

```
[19]: df.reset_index(inplace=True,drop=True)
```

0.2.3 Handling Duplicated Rows

```
[20]: # Duplicated rows

      df[df.duplicated()]
```

```
[20]: Empty DataFrame
      Columns: [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]
      Index: []
```

Although there are no duplicated rows, they are being dropped as a precautionary measure.

```
[21]: # Dropping the duplicated rows

      df.drop_duplicates(inplace=True)
```

0.2.4 Formatting Values - Numeric Columns

```
[22]: df.dtypes
```

```
[22]: age                int64
      gender            object
      region            object
      marital_status    object
      number_of_dependants  int64
      bmi_category      object
      smoking_status    object
      employment_status object
      income_level      object
      income_lakhs      int64
      medical_history    object
      insurance_plan     object
      annual_premium_amount  int64
      dtype: object
```

```
[23]: # Selecting only the columns with numerical values
```

```
numeric_columns = df.select_dtypes(include=['int64']).columns
numeric_columns
```

```
[23]: Index(['age', 'number_of_dependants', 'income_lakhs', 'annual_premium_amount'],
      dtype='object')
```

```
[24]: # Printing the unique values of each numeric columns to identify the values
      ↪with improper formats
```

```
for col in numeric_columns:
    print(f'{col}:\n',df[col].unique())
    print('*'*100)
```

age:

```
[ 26  29  49  30  18  56  33  43  59  22  21  46  68  60  27  25  36  20
  28  32  19  55  35  52  40  23  50  41  67  37  24  34  54  42  45  44
  57  38  31  58  48  51 224  47  39  53  66  64  65  62  61  70  72  69
  71 124  63 136 203 356]
```

```
*****
*****
```

number_of_dependants:

```
[ 0  2  3  4  1  5 -3 -1]
```

```
*****
*****
```

income_lakhs:

```
[  6  20  77  99  14   4  46  21   3  97   1  27  15  18   7  37  30  13
   8  83  19  29   5  70  11  33  23  40  84  22   9  71  59  38  35  28
  39  57  25  12  36  92   2  24  16  34  93  78  26  49  68  52  62  31
  90  50  32  10  88  54  86  41  95  64  85  81  79  56  80  17  98  89
  82 100  44  66  53  75  94  69  58  74  65  91  42  61  87  96  51  67
  43  73  63  55  48  45  47  72  60 560  76 440 630 900 930 580 700 790]
```

```

770 680]
*****
*****
annual_premium_amount:
[ 9053 16339 18164 ... 26370 10957 27076]
*****
*****

```

The negative values in `number_of_dependents` should be handled by converting them to their absolute values.

[25]: *# Before formatting*

```
df['number_of_dependants'].unique()
```

[25]: array([0, 2, 3, 4, 1, 5, -3, -1])

[26]: `df['number_of_dependants'] = abs(df['number_of_dependants'])`

[27]: *# After formatting*

```
df['number_of_dependants'].unique()
```

[27]: array([0, 2, 3, 4, 1, 5])

0.2.5 Formatting Values - Categorical Columns

[28]: *# Selecting only the columns with categorical values*

```
cat_columns = df.select_dtypes(include=['object']).columns
cat_columns
```

[28]: Index(['gender', 'region', 'marital_status', 'bmi_category', 'smoking_status',
'employment_status', 'income_level', 'medical_history',
'insurance_plan'],
dtype='object')

[29]: *# Printing the unique values of each categorical columns to identify the values, ↵
↵with improper formats*

```
for col in cat_columns:
    print(f'{col}:\n',list(df[col].unique()))
    print('*'*100)
```

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 Smoke', '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']
*****
*****

```

As indicated by the above result, the `smoking_status` column contains multiple values with the same meaning; therefore, this column will be formatted.

```
[30]: # Before formatting
```

```
df['smoking_status'].unique()
```

```
[30]: array(['No Smoking', 'Regular', 'Occasional', 'Smoking=0',
        'Does Not Smoke', 'Not Smoking'], dtype=object)
```

```
[31]: # Replacing values with desired formats
```

```
df['smoking_status'] = df['smoking_status'].replace(
```



```
{
    'Smoking=0' : 'No Smoking',
    'Does Not Smoke' : 'No Smoking',
    'Not Smoking' : 'No Smoking',
}
```

```
[32]: # After formatting
```

```
df['smoking_status'].unique()
```

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

0.2.6 Outlier Treatment

```
[33]: df.describe()
```

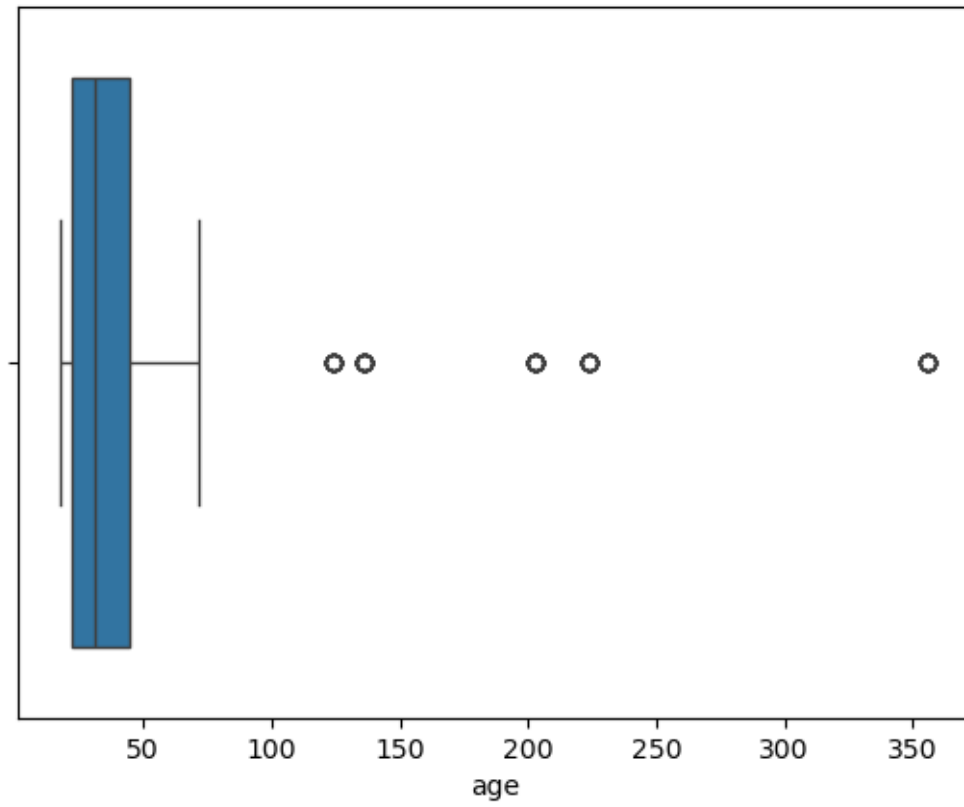
```
[33]:
```

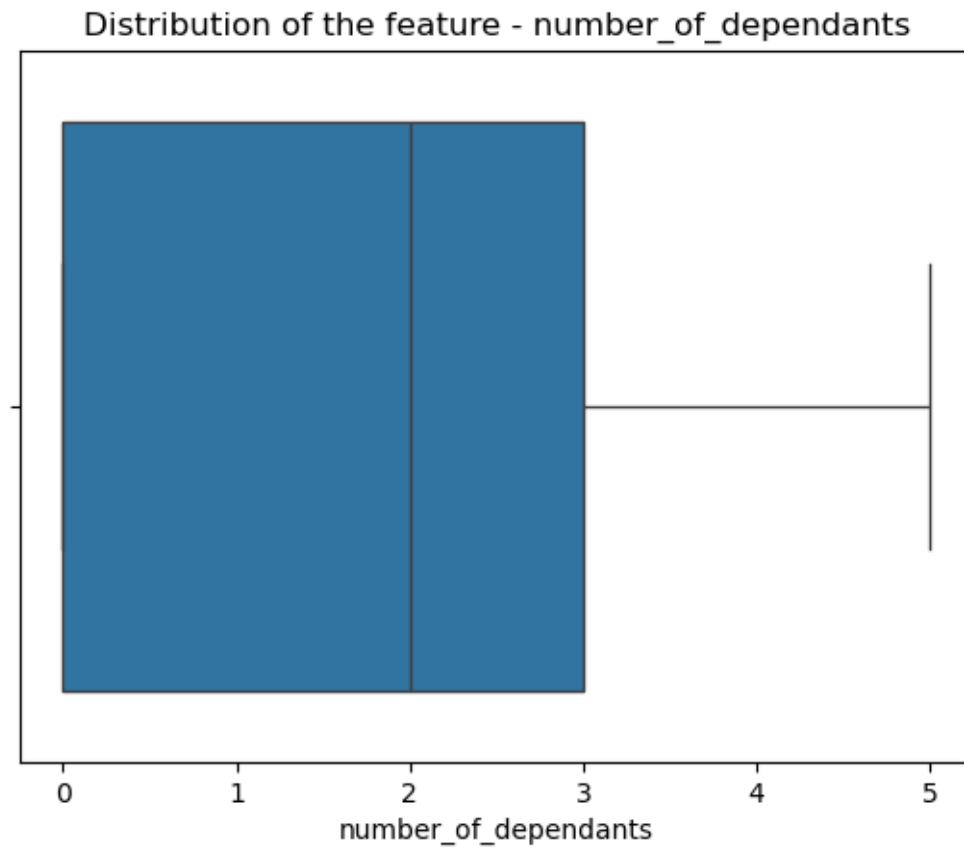
	age	number_of_dependants	income_lakhs	annual_premium_amount
count	49976.000000	49976.000000	49976.000000	49976.000000
mean	34.591764	1.717284	23.021150	15766.810189
std	15.000378	1.491953	24.221794	8419.995271
min	18.000000	0.000000	1.000000	3501.000000
25%	22.000000	0.000000	7.000000	8607.750000
50%	31.000000	2.000000	17.000000	13928.000000
75%	45.000000	3.000000	31.000000	22273.500000
max	356.000000	5.000000	930.000000	43471.000000

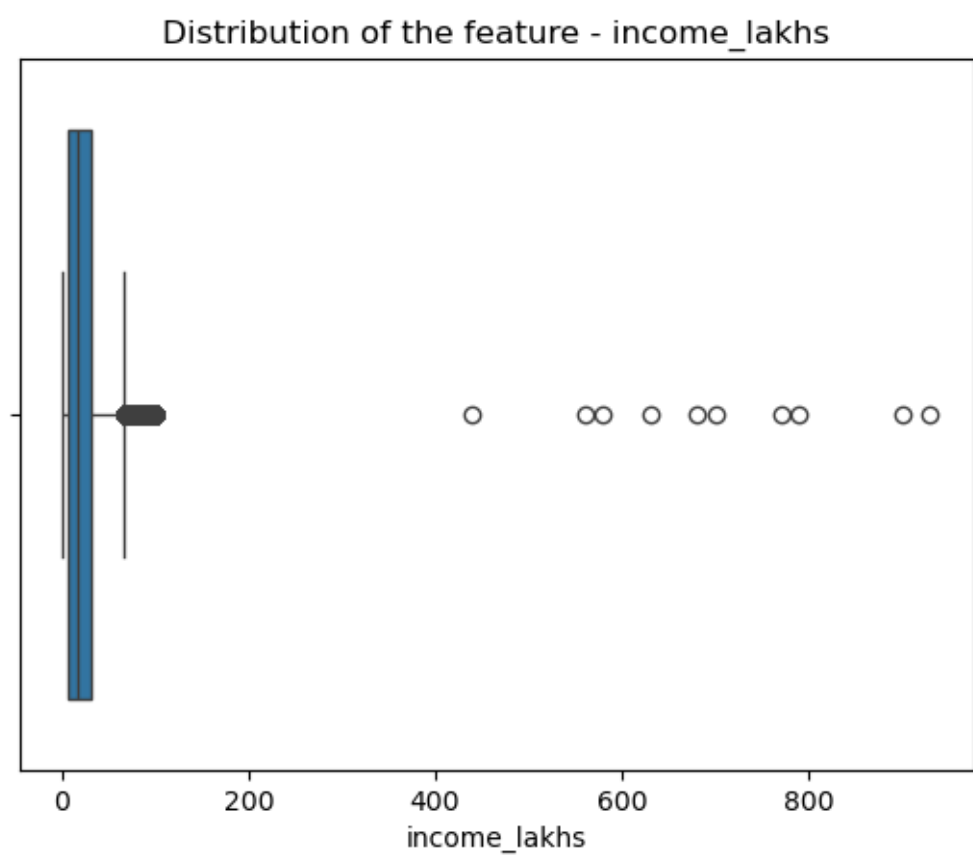
```
[34]: # Plotting box plot for each numerical columns to detect outliers
```

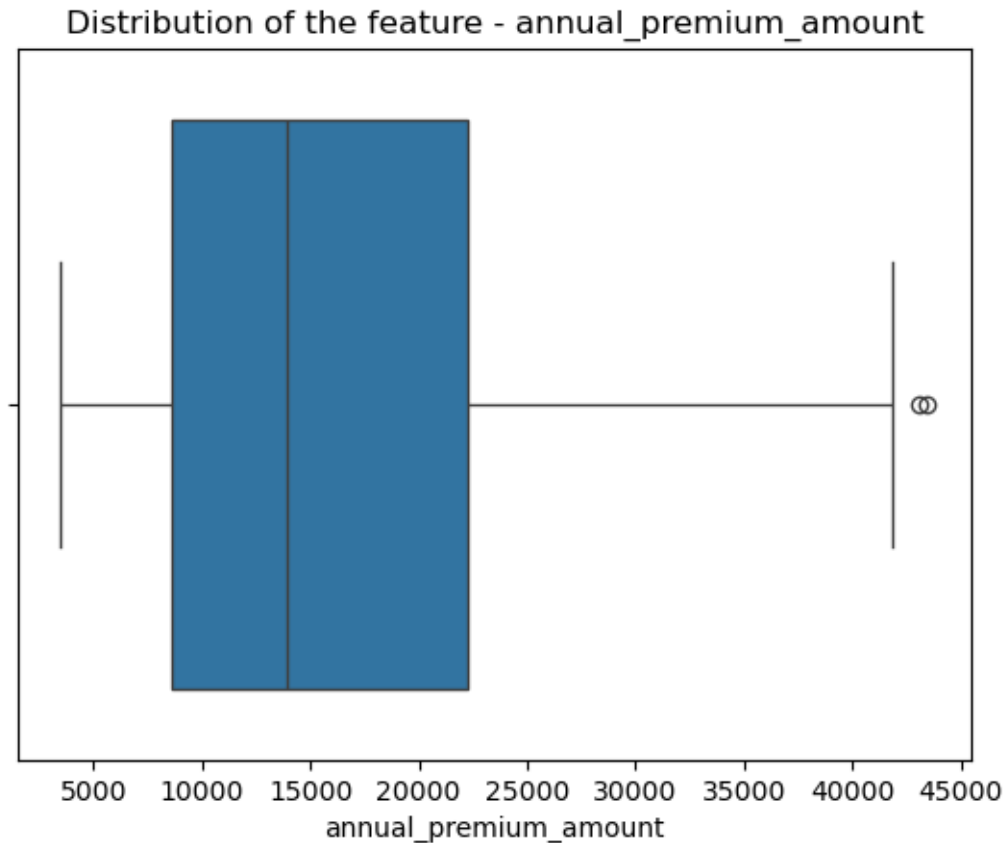
```
for col in numeric_columns:
    sns.boxplot(data=df,x=col)
    plt.title(f'Distribution of the feature - {col}')
    plt.show()
```

Distribution of the feature - age









As observed above, outliers are present in the `age` and `income_lakhs` columns. This is also illustrated by the box plot shown above.

Age

```
[35]: # Before
      df['age'].unique()
```

```
[35]: array([ 26,  29,  49,  30,  18,  56,  33,  43,  59,  22,  21,  46,  68,
          60,  27,  25,  36,  20,  28,  32,  19,  55,  35,  52,  40,  23,
          50,  41,  67,  37,  24,  34,  54,  42,  45,  44,  57,  38,  31,
          58,  48,  51, 224,  47,  39,  53,  66,  64,  65,  62,  61,  70,
          72,  69,  71, 124,  63, 136, 203, 356])
```

```
[36]: # Selecting only the rows where age <= 100. Because age > 100 is outlier

      df1 = df[df['age'] <= 100]
      df1.reset_index(inplace=True, drop=True)
```

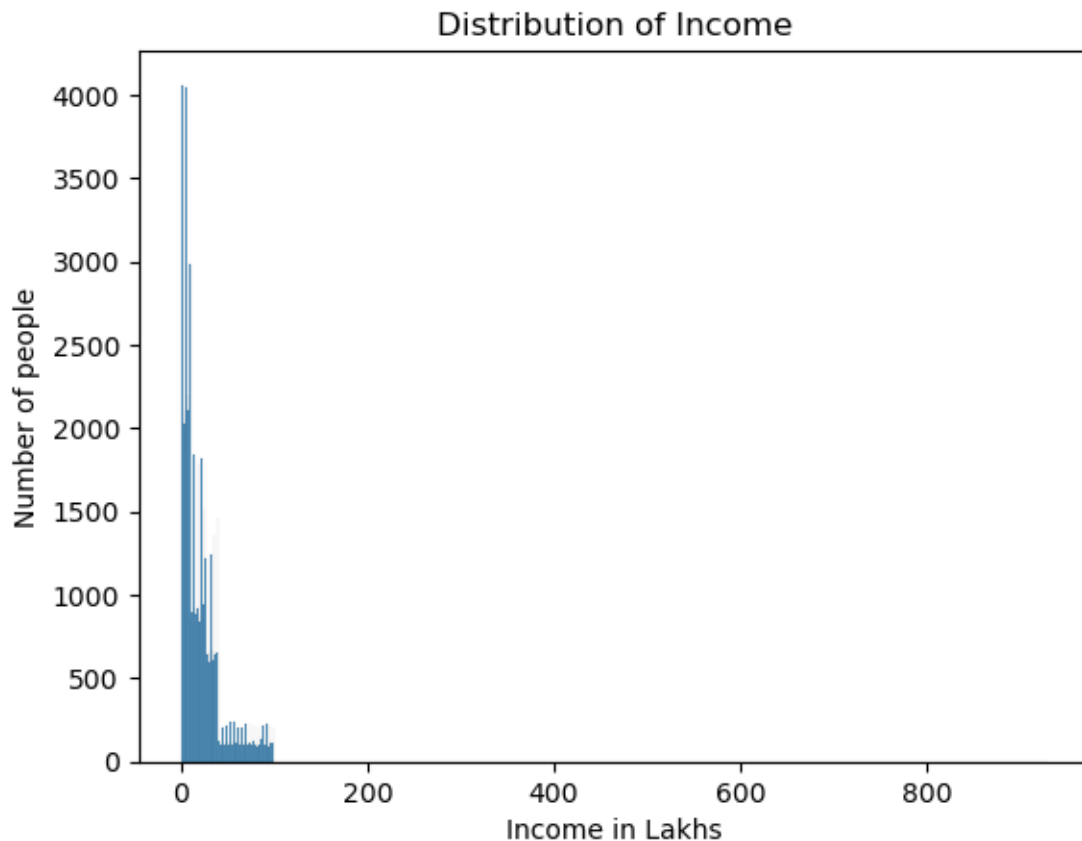
```
[37]: # After
      df1['age'].unique()
```

```
[37]: array([26, 29, 49, 30, 18, 56, 33, 43, 59, 22, 21, 46, 68, 60, 27, 25, 36,
          20, 28, 32, 19, 55, 35, 52, 40, 23, 50, 41, 67, 37, 24, 34, 54, 42,
          45, 44, 57, 38, 31, 58, 48, 51, 47, 39, 53, 66, 64, 65, 62, 61, 70,
          72, 69, 71, 63])
```

Income

```
[38]: # Distribution of Income using Histogram
```

```
sns.histplot(data=df1,x='income_lakhs')
plt.title('Distribution of Income')
plt.xlabel('Income in Lakhs')
plt.ylabel('Number of people')
plt.show()
```



```
[39]: # Determining Lower Salary Boundary and Upper Salary Boundary using IQR Method
```

```
# Quartile 1 and Quartile 3
```

```
q1 = np.percentile(df1['income_lakhs'],25)
q3 = np.percentile(df1['income_lakhs'],75)
```

```

print(f'Q1 -> {q1}\nQ3 -> {q3}')

# Inter Quartile Range

iqr = q3-q1
print('IQR ->',iqr)

# Lower and Upper boundary using IQR

lower_boundary = q1 - (iqr * 1.5)
upper_boundary = q3 + (iqr * 1.5)
print(f'Lower Boundary -> {lower_boundary}\nUpper Boundary -> {upper_boundary}')

```

```

Q1 -> 7.0
Q3 -> 31.0
IQR -> 24.0
Lower Boundary -> -29.0
Upper Boundary -> 67.0

```

NOTE:

The current upper boundary is too low. Therefore, consultation with the business team will be conducted to determine the optimal upper boundary for identifying outliers in the income_lakhs column.

It has been decided that the upper boundary will be set at 1 crore (i.e., 100 lakhs). Any values exceeding this threshold will be considered outliers.

```

[40]: # Selecting only the rows where income <= 100 lakhs. Because income > 100 lakhs
      ↳ is outlier

```

```

income_threshold = 100
df2 = df1[df1['income_lakhs'] <= income_threshold]

```

```

[41]: df2.reset_index(drop=True,inplace=True)

```

```

[42]: df2

```

```

[42]:
   age  gender  region marital_status  number_of_dependants  \
0    26   Male  Northwest      Unmarried                0
1    29  Female  Southeast      Married                2
2    49  Female  Northeast      Married                2
3    30  Female  Southeast      Married                3
4    18   Male  Northeast      Unmarried                0
...   ...   ...   ...      ...      ...      ...
49903  24  Female  Northwest      Unmarried                0
49904  47  Female  Southeast      Married                2
49905  21   Male  Northwest      Unmarried                0
49906  18   Male  Northwest      Unmarried                2

```

49907	48	Female	Southwest	Married	3
-------	----	--------	-----------	---------	---

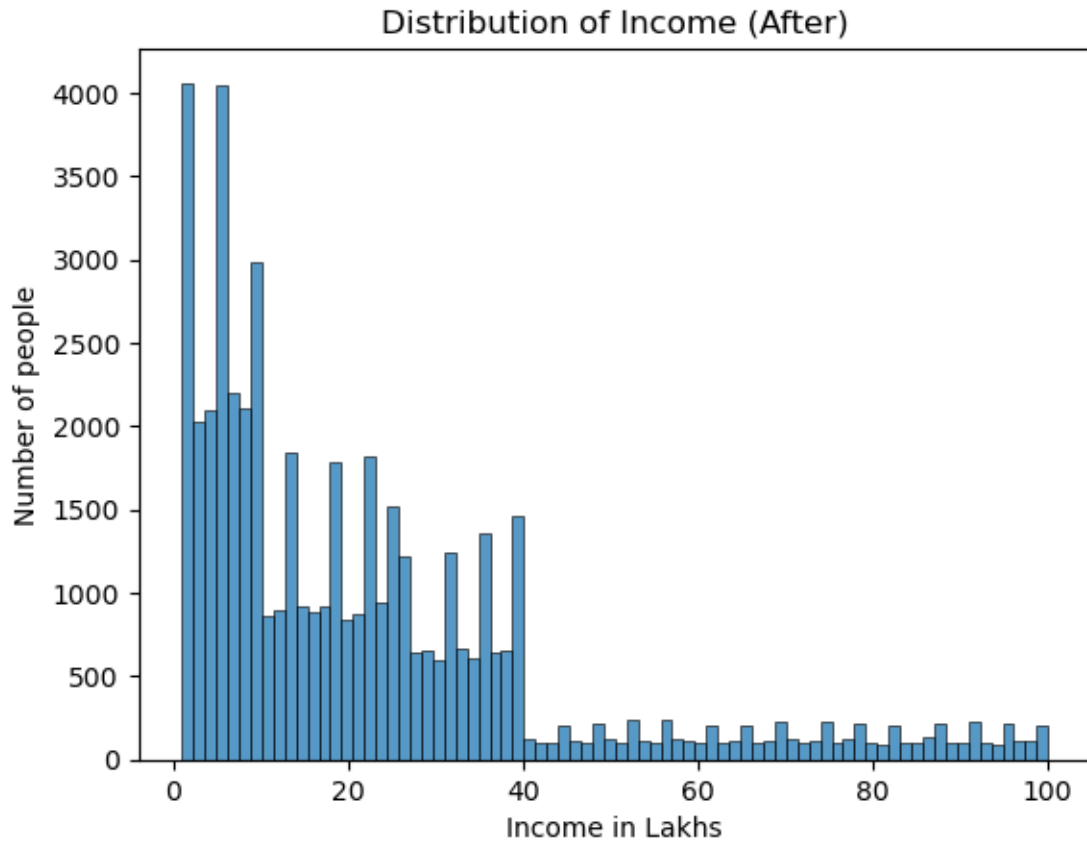
	bmi_category	smoking_status	employment_status	income_level \
0	Normal	No Smoking	Salaried	<10L
1	Obesity	Regular	Salaried	<10L
2	Normal	No Smoking	Self-Employed	10L - 25L
3	Normal	No Smoking	Salaried	> 40L
4	Overweight	Regular	Self-Employed	> 40L
...
49903	Underweight	No Smoking	Self-Employed	25L - 40L
49904	Normal	No Smoking	Salaried	> 40L
49905	Normal	Regular	Freelancer	25L - 40L
49906	Normal	No Smoking	Salaried	10L - 25L
49907	Normal	Occasional	Self-Employed	<10L

	income_lakhs	medical_history	insurance_plan	annual_premium_amount
0	6	Diabetes	Bronze	9053
1	6	Diabetes	Bronze	16339
2	20	High blood pressure	Silver	18164
3	77	No Disease	Gold	20303
4	99	High blood pressure	Silver	13365
...
49903	35	No Disease	Bronze	9111
49904	82	Thyroid	Gold	27076
49905	32	No Disease	Bronze	8564
49906	20	No Disease	Bronze	9490
49907	7	Diabetes	Silver	19730

```
[49908 rows x 13 columns]
```

```
[43]: # After Treating outliers in 'income_lakhs' columns
```

```
sns.histplot(data=df2,x='income_lakhs')
plt.title('Distribution of Income (After)')
plt.xlabel('Income in Lakhs')
plt.ylabel('Number of people')
plt.show()
```

0.3 EDA

Exploratory Data Analysis (EDA) is performed as follows:

- Univariate analysis on numeric columns
- Univariate analysis on categorical columns
- Bivariate analysis on numeric columns
- Bivariate analysis on categorical columns

0.3.1 Univariate Analysis - Numeric Columns

The distribution of numerical columns will be plotted using histograms.

```
[44]: numeric_columns
```

```
[44]: Index(['age', 'number_of_dependants', 'income_lakhs', 'annual_premium_amount'],
dtype='object')
```

```
[45]: # Plotting Distribution for all numerical columns
```

```
fig , ax = plt.subplots(2,2,figsize=(8,8))
```

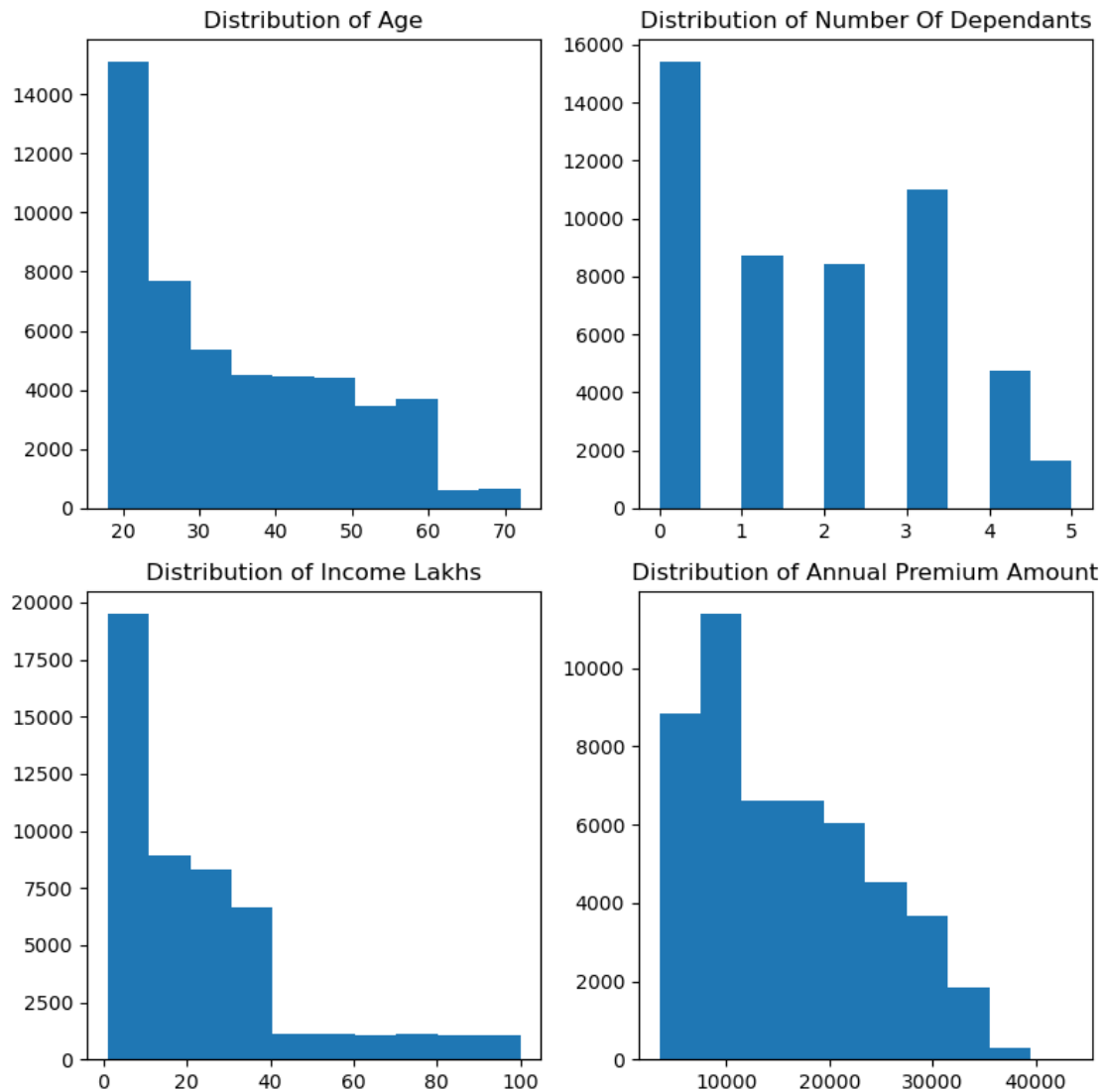
```

# Iterator initiation to retrieve one value at a time
it = iter(numeric_columns)

for i in range(2):
    for j in range(2):
        # To retrieve one value at a time
        col = next(it)
        # Format column name for display: capitalize first letter of each word
        ↪and replace underscores with spaces
        x_ax = col.title().replace('_', ' ')
        ax[i,j].hist(x=df2[col])
        ax[i,j].set_title(f'Distribution of {x_ax}')

# To adjust subplot parameters to give specified padding and prevent overlap of
↪labels
plt.tight_layout()
plt.show()

```



0.3.2 Univariate Analysis - Categorical Columns

The distribution of categorical columns will be plotted using barcharts.

```
[46]: df2[cat_columns]
```

```
[46]:
```

	gender	region	marital_status	bmi_category	smoking_status	\
0	Male	Northwest	Unmarried	Normal	No Smoking	
1	Female	Southeast	Married	Obesity	Regular	
2	Female	Northeast	Married	Normal	No Smoking	
3	Female	Southeast	Married	Normal	No Smoking	
4	Male	Northeast	Unmarried	Overweight	Regular	
...	

49903	Female	Northwest	Unmarried	Underweight	No Smoking
49904	Female	Southeast	Married	Normal	No Smoking
49905	Male	Northwest	Unmarried	Normal	Regular
49906	Male	Northwest	Unmarried	Normal	No Smoking
49907	Female	Southwest	Married	Normal	Occasional

	employment_status	income_level	medical_history	insurance_plan
0	Salaried	<10L	Diabetes	Bronze
1	Salaried	<10L	Diabetes	Bronze
2	Self-Employed	10L - 25L	High blood pressure	Silver
3	Salaried	> 40L	No Disease	Gold
4	Self-Employed	> 40L	High blood pressure	Silver
...
49903	Self-Employed	25L - 40L	No Disease	Bronze
49904	Salaried	> 40L	Thyroid	Gold
49905	Freelancer	25L - 40L	No Disease	Bronze
49906	Salaried	10L - 25L	No Disease	Bronze
49907	Self-Employed	<10L	Diabetes	Silver

[49908 rows x 9 columns]

In previous runs, the x-axis labels for the `medical_history` column were misaligned in the plots. To resolve this issue, the values will be mapped using the dictionary `ds_dict` as shown below.

```
[47]: ds_dict = {'No Disease': 'NO DISEASE',
'Diabetes': 'DB',
'High blood pressure': 'HBP',
'Thyroid': 'THY',
'Heart disease': 'HD',
'Diabetes & High blood pressure': 'DB & HBP',
'High blood pressure & Heart disease': 'HBP & HD',
'Diabetes & Thyroid': 'DB & THY',
'Diabetes & Heart disease': 'DB & HD'}
```

ds_dict

```
[47]: {'No Disease': 'NO DISEASE',
'Diabetes': 'DB',
'High blood pressure': 'HBP',
'Thyroid': 'THY',
'Heart disease': 'HD',
'Diabetes & High blood pressure': 'DB & HBP',
'High blood pressure & Heart disease': 'HBP & HD',
'Diabetes & Thyroid': 'DB & THY',
'Diabetes & Heart disease': 'DB & HD'}
```

[48]: *# Showing how mapping works*

```
print('Without Mapping:')
print(list(df2['medical_history'].value_counts(normalize=True).index))
print('\n','*' * 100, '\n')
print('With Mapping:')
print(list(df2['medical_history'].value_counts(normalize=True).index.
↪map(ds_dict)))
```

Without Mapping:

```
['No Disease', 'Diabetes', 'High blood pressure', 'Thyroid', 'Heart disease',
'Diabetes & High blood pressure', 'High blood pressure & Heart disease',
'Diabetes & Thyroid', 'Diabetes & Heart disease']
```

```
*****
*****
```

With Mapping:

```
['NO DISEASE', 'DB', 'HBP', 'THY', 'HD', 'DB & HBP', 'HBP & HD', 'DB & THY', 'DB
& HD']
```

[49]: *# Plotting Distribution for all categorical columns*

```
fig, ax = plt.subplots(3,3,figsize=(16,22))

# Iterator initiation to retrieve one value at a time
it = iter(cat_columns)

for i in range(3):
    for j in range(3):
        # To retrieve one value at a time
        col = next(it)

        # Format column name for display: capitalize first letter of each word
        ↪and replace underscores with spaces
        x_ax = col.title().replace('_', ' ')

        # Map the 'medical_history' column using ds_dict to ensure consistent
        ↪and clean labels for plotting

        if col == 'medical_history':
            x_val = df2[col].value_counts(normalize=True).index.map(ds_dict)
        else:
            x_val = df2[col].value_counts(normalize=True).index

        h_val = df2[col].value_counts(normalize=True).values * 100
        ax[i,j].bar(x=x_val,height=h_val)
```

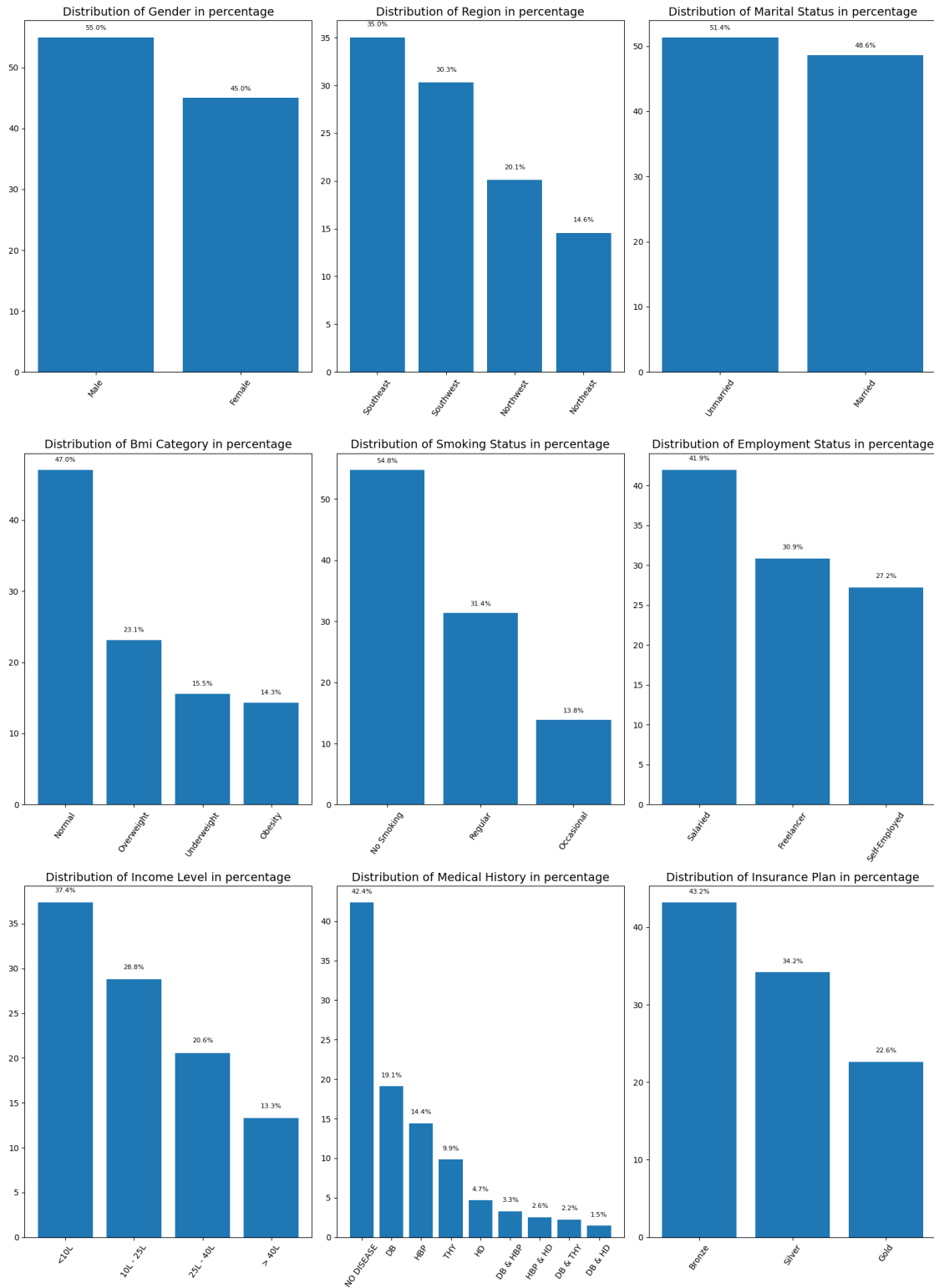
```

ax[i,j].set_title(f'Distribution of {x_ax} in percentage',fontsize=14)
ax[i,j].tick_params(axis='x', rotation=55)

# Annotate each bar with its corresponding percentage value for better
↳ interpretability
    for idx, val in enumerate(h_val):
        ax[i,j].text(idx, val + 1, f'{round(val,1)}%', ha='center',
        ↳va='bottom', fontsize=8)

plt.tight_layout()
plt.show()

```



0.3.3 Bivariate Analysis - Numeric Columns

The relationship between each numerical column (excluding the `annual_premium_amount` column itself) and the `annual_premium_amount` column will be plotted to analyze potential correlations.

```
[50]: df2[numeric_columns]
```

```
[50]:
```

	age	number_of_dependants	income_lakhs	annual_premium_amount
0	26	0	6	9053
1	29	2	6	16339
2	49	2	20	18164
3	30	3	77	20303
4	18	0	99	13365
...
49903	24	0	35	9111
49904	47	2	82	27076
49905	21	0	32	8564
49906	18	2	20	9490
49907	48	3	7	19730

```
[49908 rows x 4 columns]
```

```
[51]: # Plotting the scatter plots
```

```
fig, ax = plt.subplots(1,3,figsize=(20,8))
```

```
for i in range(3):
```

```
    col = numeric_columns[i]
```

```
    # Format column names for display: capitalize first letter of each word and
    ↪replace underscores with spaces
```

```
    x_ax = col.title().replace('_', ' ')
```

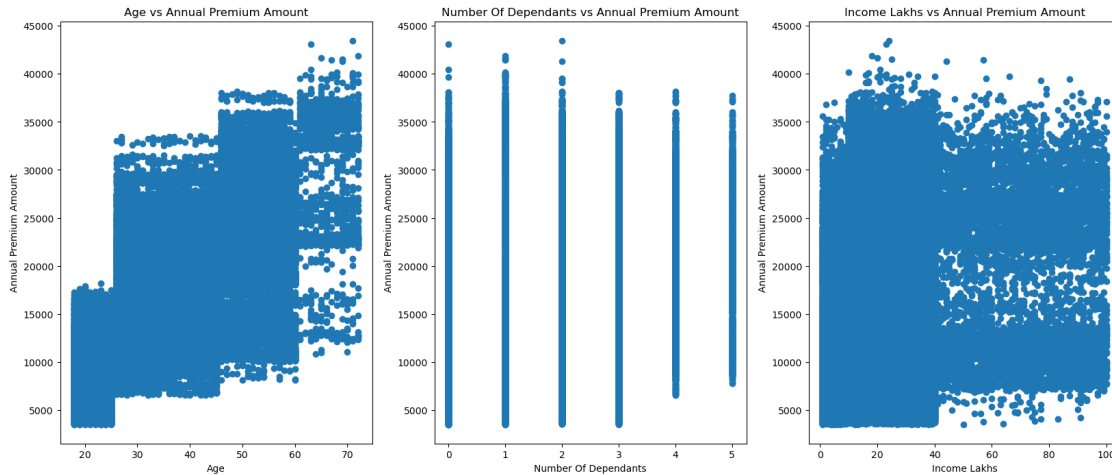
```
    y_ax = 'annual_premium_amount'.title().replace('_', ' ')
```

```
    ax[i].scatter(x=df2[col],y=df2['annual_premium_amount'])
```

```
    ax[i].set_title(f'{x_ax} vs {y_ax}')
```

```
    ax[i].set_xlabel(x_ax)
```

```
    ax[i].set_ylabel(y_ax)
```

0.3.4 Bivariate Analysis - Categorical

The relationship between the categorical columns `income_level` and `insurance_plan` will be explored using multiple approaches:

- Cross-tabulation
- Grouped Bar Chart
- Stacked Bar Chart
- Heatmap

```
[52]: df2.head()
```

```
[52]:   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
```

```
   smoking_status  employment_status  income_level  income_lakhs  \
0   No Smoking      Salaried      <10L           6
1   Regular      Salaried      <10L           6
2   No Smoking  Self-Employed  10L - 25L          20
3   No Smoking      Salaried      > 40L          77
4   Regular  Self-Employed      > 40L          99
```

```
   medical_history  insurance_plan  annual_premium_amount
0      Diabetes      Bronze      9053
1      Diabetes      Bronze     16339
2  High blood pressure      Silver     18164
3      No Disease      Gold      20303
```

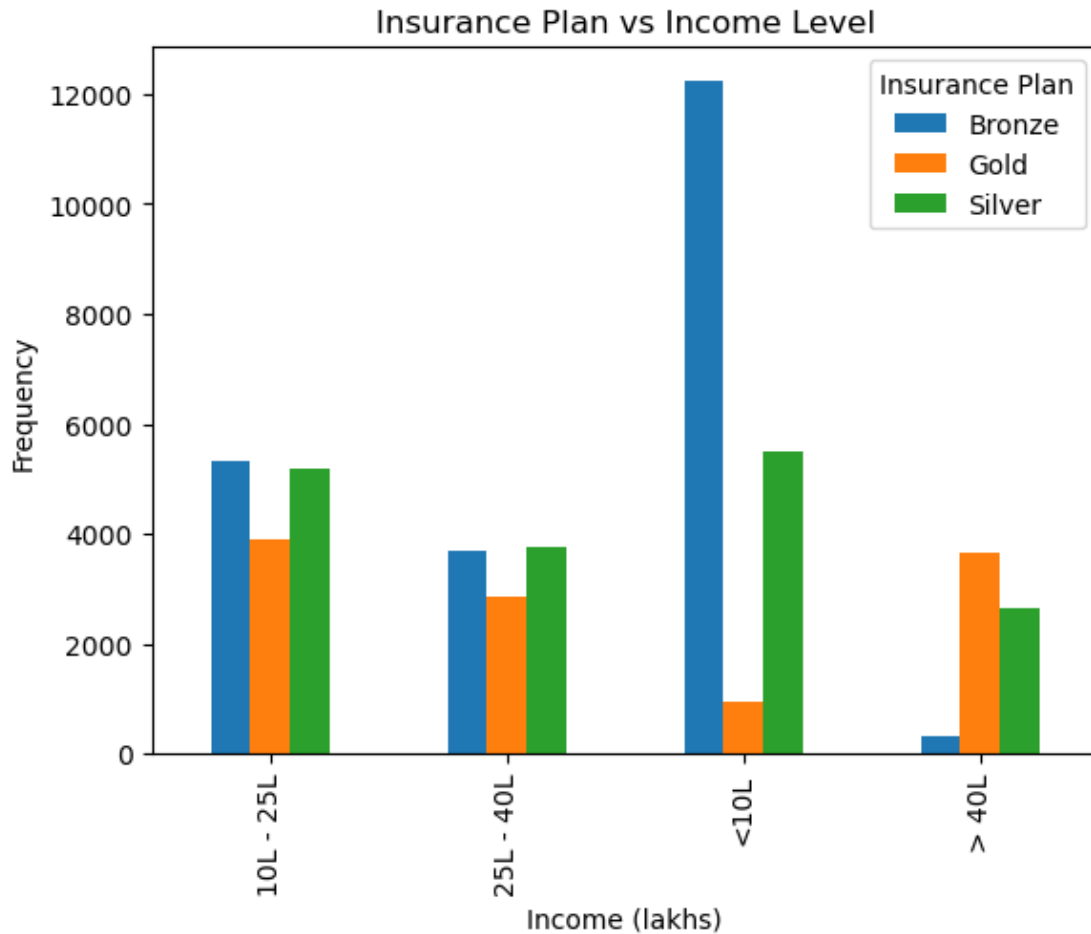
Cross-tabulation The frequency distribution between `income_level` and `insurance_plan` will be examined using a cross-tabulation, which will display the number of individuals in each combination of income level and insurance plan.

```
[53]: ct = pd.crosstab(df['income_level'],df['insurance_plan'])
      ct
```

```
[53]: insurance_plan  Bronze  Gold  Silver
      income_level
      10L - 25L      5314  3886   5189
      25L - 40L      3686  2844   3753
      <10L          12239   931   5495
      > 40L           330  3662   2647
```

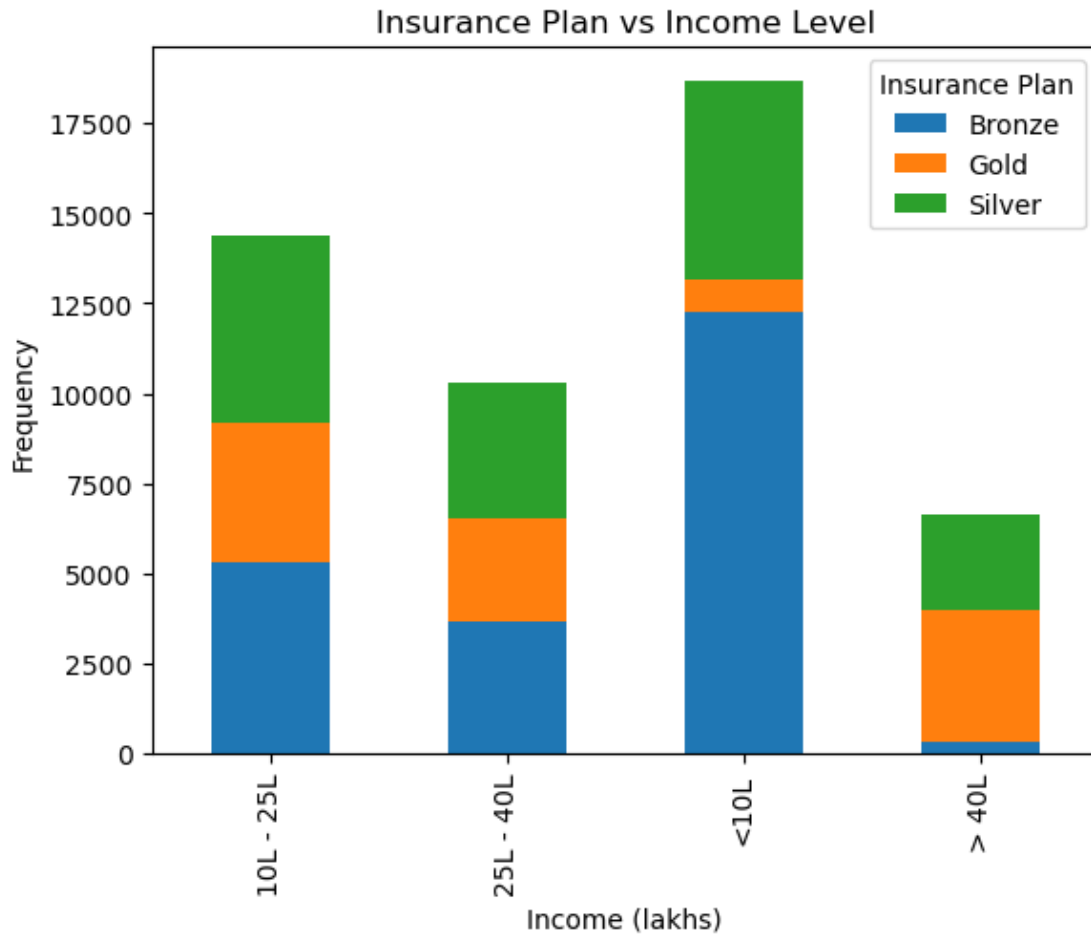
Grouped Bar Chart The count of each `insurance_plan` across different `income_level` categories will be visualized using grouped bar charts, where each income level will have side-by-side bars representing the different insurance plans for easy comparison.

```
[54]: ct.plot(kind='bar')
      plt.title('Insurance Plan vs Income Level')
      plt.xlabel('Income (lakhs)')
      plt.ylabel('Frequency')
      plt.legend(title='Insurance Plan')
      plt.show()
```



Stacked Bar Chart The distribution of `insurance_plan` within each `income_level` will be represented in a stacked bar chart format, helping to understand the proportion of each plan type within the income categories.

```
[55]: ct.plot(kind='bar',stacked=True)
plt.title('Insurance Plan vs Income Level')
plt.xlabel('Income (lakhs)')
plt.ylabel('Frequency')
plt.legend(title='Insurance Plan')
plt.show()
```



Heatmap The intensity of the relationship between `insurance_plan` and `income_level` will be visualized using a heatmap, where the cells are color-coded based on frequency to highlight patterns and concentrations in the data.

```
[56]: sns.heatmap(ct,annot=True,fmt='0')
plt.title('Insurance Plan vs Income Level')
plt.xlabel('Insurance Plan')
plt.ylabel('Income Level')
plt.show()
```

