PROJECT PHASE - 1

Bhavani Kiran Kukunoor
Surya Venkata Rohit Moganti

PREDICTION MODEL FOR LEAD HEALTH INSURANCE

Problem Statement

The Health Insurance sector encounters huge challenges in accurately identifying possible customers. Not concentrating well enough on the potential customers leads to poor conversion rates. Despite having enormous client data, many organizations struggle to analyze and predict leads. The health insurance companies lack precision in lead identification, which results in loss in income and market spendings. The goal of our model is to discover and target people who are most likely to get health insurance by using predictive analysis.

Health Insurance benefits individuals and families by improving their access to health services and providing financial protection against expenses both high and modest, that are still unaffordable for many. Having health insurance shields you from unforeseen, expensive medical bills. The objective of the model is to forecast the number of individuals who will choose to opt for health insurance. An individual is categorized as lead, once they complete the insurance form for the policy suggested to them on the website, this is perceived as a positive outcome. Failure to do so is classified as a negative outcome.

Obtaining leads efficiently is crucial for business success, in the highly competitive field of the Health insurance sector. The key challenge is to track user behavior and anticipate lead conversions from website visitors. Our model aims to improve lead prediction performance with higher precision.

Growth of the company’s business is directly proportional to customer satisfaction. Volume of sales significantly affects the earning potential of the company. Various insurance policies are offered to different people based on how well they match. The project helps accurately identify candidates who have high chances of becoming customers. It eventually helps the business to gather information about events and plan ways to go around any obstacles that may arise. The project helps insurers to better understand customer preferences, which leads to customizing their services and offerings to meet the demands of the present market. Thus, fostering long term relationships with the customers. A better return on investment can be obtained by implementing targeted marketing strategies that are based on the insights obtained from the project. Insurers can optimize their marketing budget by concentrating their efforts on individuals who are more likely to convert. The project's output can be properly analyzed, allowing insurers to effectively streamline their marketing strategies. This contribution plays a vital role in providing a competitive advantage to insurers in a high demand market.

Data Source -

https://www.kaggle.com/datasets/owaiskhan9654/health-insurance-lead-prediction-raw-data

The dataset is obtained from Kaggle. Our model is based on lead generation for health insurance, which is a binary classification problem. Main goal of the project is to predict if an individual will become a customer as in "lead" or not. This can assist insurers in optimizing their marketing strategies.

The dataset comprises of multiple columns of client data -

- ID - Id is a unique identifier given to a customer
- City_Code - Each city is represented with a city code.
- Region_Code - Each region is represented with a region code
- Accommodation_Type - Describes type of accommodation of the customer, either owned or rented.
- Reco_Insurance_Type - Describes type of insurance that is recommended to a customer.
- Upper_Age - Indicates Upper age limit of a customer
- Lower_Age - Indicates lower age limit of a customer
- Is_Spouse - Indicates if an individual is married or not
- Health_Indicator - Describes health of a customer
- Holding_Policy_Duration - Describes duration of policy held by the customer
- Holding_Policy_Type - Describes the type of policy that is currently being held by a customer.
- Reco_Policy_Category - Policy category that is recommended to a customer
- Reco_Policy_Premium - Describes recommended premium amount for the policy suggested.
- Response - Is the target output. It suggests if an individual converted as lead or not.

Data Cleaning

1. Removing spaces in Column Name -

Unwanted spaces in a column name can reduce readability and clarity of data presentation. For the column name “Health Indicator”, space was replaced by an underscore “Health_Indicator”. Such uniformity in naming simplifies code writing and debugging.. Spaces in column names can cause parsing issues, typographical errors, incompatibility issues. To avoid such errors, removal of spaces from column names is performed.

Performed on: Health_Indicator

```
# The name of the column is with a space and it might be a problem further
# Adding underscore to the column name [Health_Indicator]


Dataset = Dataset.rename(columns={'Health Indicator': 'Health_Indicator'})
Dataset.head()
```

ısurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health_Indicator	Holding_Policy_Duration	H
Individual	36	36	No	X1	14+	
Joint	75	22	No	X2	NaN	
Individual	32	32	No	NaN	1.0	
Joint	52	48	No	X1	14+	
Individual	44	44	No	X2	3.0	

2. Forward filling -

This data cleaning step allows us to make observations, by filling in appropriate recent available values in missing entries. This helps to maintain the overall integrity of the dataset. This method avoids bias that can result from imputing missing values, it can not be seen while performing mode which might change how the data were originally distributed.

Performed on: Health_Indicator

```
# Replacing the missing values with mode will make the coloumn biased to one category
# Here we are including forward fill concept where the missing value replaces with its forward value in the column
Dataset['Health_Indicator'] = Dataset['Health_Indicator'].ffill()
print(Dataset['Health_Indicator'].value_counts())
```

```
Health_Indicator
X1    17020
X2    13309
X3     8722
X4     7471
X5     2242
X6     1683
X7      258
X8      104
X9       73
Name: count, dtype: int64
```

3. One hot encoding -

It is performed on categorical variables to convert them into binary representation. It allows different algorithms to easily interpret and process the data, without making incorrect assumptions.

Performed on: Health_Indicator

```
#Performing One Hot Encoding for this Health_Indicator column as it is a categorical column.
Health_Indicator_One_Hot = pd.get_dummies(Dataset['Health_Indicator'],prefix = 'HI_',dtype = 'int')


#Health_Indicator_One_Hot.head()
#len(Health_Indicator_One_Hot)


#Concating this new Dataframe to the existing Dataset
Dataset = pd.concat([Dataset, Health_Indicator_One_Hot], axis=1)
#Dataset = Dataset.drop('Health_Indicator', axis = 1)
```

```
Dataset.head()
```

e	Lower_Age	Is_Spouse	Health_Indicator	Holding_Policy_Duration	...	Response	HI__X1	HI__X2	HI__X3	HI__X4	HI__X5	HI__X6	HI__X7	HI__X8	HI__X9
3	36	No	X1	14+	...	0	1	0	0	0	0	0	0	0	0
5	22	No	X2	NaN	...	0	0	1	0	0	0	0	0	0	0
2	32	No	X2	1.0	...	1	0	1	0	0	0	0	0	0	0
2	48	No	X1	14+	...	0	1	0	0	0	0	0	0	0	0
4	44	No	X2	3.0	...	0	0	1	0	0	0	0	0	0	0

4. Identifying Inconsistent Data -

Data cleaning contributes to the accuracy and consistency of the dataset by locating and resolving inconsistencies. If inaccurate or inconsistent data is not handled, it might result in incorrect conclusions and judgments.

Performed on : Holding policy duration

```
# Dealing with Holding Policy Duration Column
```

```
# Handling Missing Values in Holding_Policy_Indicator
print(Dataset['Holding_Policy_Duration'].value_counts())
# There are 20251 missing values in this column
print(Dataset['Holding_Policy_Duration'].dtype)
```

```
Holding_Policy_Duration
1.0     4499
14+     4335
2.0     4260
3.0     3586
```

Here 14+ is an inconsistent data, so we change such values to 15

```
#There is a string datatype in this column saying 14+
#Now Change this value to numeric Form
# 14+ here might indicate that the term is greater than 14 hence let us keep the value as 15 which is greater than 14

Dataset['Holding_Policy_Duration'] = Dataset['Holding_Policy_Duration'].replace('14+', 15.0)
print(Dataset['Holding_Policy_Duration'].value_counts())
```

```
Holding_Policy_Duration
1.0     4499
15.0    4335
2.0     4260
3.0     3586
4.0     2771
5.0     2362
6.0     1894
7.0     1645
8.0     1316
9.0     1114
10.0     813
11.0     546
12.0     513
13.0     511
14.0     466
Name: count, dtype: int64
```

5. Identifying Inconsistent Data Types -

Data columns should have a standardized consistent format. Inconsistent data types should be converted to data types that align with the column's datatype. By addressing such issues, we can facilitate accurate analysis.

Performed on: Holding_Policy_Duration

```
#The Column here is comprised to inconsistent datatypes
#Let us change all the values to float datatype

Dataset['Holding_Policy_Duration'] = Dataset['Holding_Policy_Duration'].astype(float)
```

6. Replacing missing values with mean -

A straightforward yet effective method is to use the mean function to replace any missing inputs. It is simple to use and does not require complex calculations. Imputing mean ensures that the statistics of the entire column are not altered.

Performed on: Holding_Policy_Duration

```
#In order to compute Mean of the column let us first change the Nan values to 0
Dataset['Holding_Policy_Duration'] = Dataset['Holding_Policy_Duration'].fillna(0)
```

```
#Since the column here is an Numeric Column
#We can replace the missing values with the mean of the remaining values


Mean_of_Column = Dataset['Holding_Policy_Duration'].mean()
Mean_of_Column = float(math.ceil(Mean_of_Column))
print(Mean_of_Column)


#Fill the Mean value to all the missing values in the column
Dataset['Holding_Policy_Duration'] = Dataset['Holding_Policy_Duration'].replace(0,Mean_of_Column)
```

```
4.0
```

7. Replace NaN value with 0 in a column -

0 is represented as a placeholder for missing value. Which does not disrupt the flow of the data.

Performed on: Holding_Policy_type

```
print(Dataset['Holding_Policy_Type'].value_counts())
```

```
Holding_Policy_Type
3.0    13279
1.0     8173
2.0     5005
4.0     4174
Name: count, dtype: int64
```

```
#There are four different cateogries in this column and we are replacing and new values to [ 0.0 ]
Dataset['Holding_Policy_Type'] = Dataset['Holding_Policy_Type'].fillna(0.0)
# By doing this we clearly state that the value 0.0 is assigned for a missing value
```

8. Binary classification -

This method classifies a column into two classes based on specific criteria. In our case there are three binary classification problems, which have text data in them. These columns are converted to numeric data with zeros and ones. It helps simplify things and makes it easier to distinguish between various data groupings,which makes our mathematical model perform well.
Performed on: Accommodation_Type , Reco_Insurance_Type

```
# These columns are binary classification column and hence we can change these values to 0 or 1

Dataset['Accomodation_Type'] = Dataset['Accomodation_Type'].replace('Rented',0)
Dataset['Accomodation_Type'] = Dataset['Accomodation_Type'].replace('Owned',1)

Dataset['Reco_Insurance_Type'] = Dataset['Reco_Insurance_Type'].replace('Individual',0)
Dataset['Reco_Insurance_Type'] = Dataset['Reco_Insurance_Type'].replace('Joint',1)

Dataset['Is_Spouse'] = Dataset['Is_Spouse'].replace('No',0)
Dataset['Is_Spouse'] = Dataset['Is_Spouse'].replace('Yes',1)
```

9. Multiple classification -

Similar to binary classification, multiple classification allows for the grouping of a column into numerous classes. Multi-class classification is involved. This was accomplished via label encoding, which gives each class a unique integer after converting categorical labels into numerical values. Label encoding is efficient for large datasets, It also uses less memory because the categorical variable is represented by a single column.
Performed on: City_Code

```
# There are different categories in the City_Code Column lets convert it to Numeric Data
# We can perform this by Label Encoding concept
Dataset['City_Code'] = Dataset['City_Code'].astype('category')
Dataset.dtypes
```

```
City_Code                  category
Region_Code                   int64
Accomodation_Type             int64
Reco_Insurance_Type           int64
Upper_Age                     int64
Lower_Age                     int64
Is_Spouse                     int64
Holding_Policy_Duration     float64
Holding_Policy_Type         float64
Reco_Policy_Cat               int64
Reco_Policy_Premium         float64
Response                      int64
Health_Indicator__X1          int64
Health_Indicator__X2          int64
Health_Indicator__X3          int64
Health_Indicator__X4          int64
Health_Indicator__X5          int64
Health_Indicator__X6          int64
Health_Indicator__X7          int64
Health_Indicator__X8          int64
Health_Indicator__X9          int64
dtype: object
```

```
# Dataset['Health_Indicator'].head()
Dataset['City_Code_encoded'] = Dataset['City_Code'].cat.codes
```

```
Dataset[['City_Code','City_Code_encoded']]
```

	City_Code	City_Code_encoded
0	C3	22
1	C5	31
2	C5	31
3	C24	16
4	C8	34

10. Dropping a column -

Feature elimination occurs when a specific column is not useful for the task at hand, such irrelevant or unnecessary features can simply be removed by drop method.
Performed on: ID column.

```
Dataset[Dataset.duplicated('ID')]
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_A

0 rows × 23 columns

```
# Since there are no Duplicates We can drop this Column
Dataset = Dataset.drop('ID', axis = 1)
```

```
# Checking for entire row duplicates in the Dataset

duplicated_rows = Dataset[Dataset.duplicated()]

duplicated_rows
```

	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age

0 rows × 22 columns

11. Outliers -

Data points that fall far away from the rest of the data in a dataset are considered as outliers. These occur in continuous data. The model might predict wrong values due to the presence of outliers.

We have few columns with continuous data in it and we are checking the outliers using Interquartile Range - Measure of statistical dispersion, denoted as IQR.

IQR is a list of

IQR = Q3 - Q1

We then will get a list which states

[Q1 - (1.5 * IQR) , Q3 + (1.5 * IQR)],

Any value in the column which is not a part of this list can be considered as a potential Outlier.

Performed on: Reco_Policy_Premium

```
# Handling Outliers For Numeric Data using InterQuartile Range
# Checking the Outliers for there continous data column [Upper_Age,Lower_Age,Reco_Policy_Premium]
```

```
# Create function which takes a column as input and returns its IQR

def InterQuartileRange(Column):
    Quartile_1 = Dataset[Column].quantile(0.25)
    Quartile_3 = Dataset[Column].quantile(0.75)
    Inter_Quartile_Range = Quartile_3 - Quartile_1
    return [Quartile_1 - (1.5 * Inter_Quartile_Range) , Quartile_3 + (1.5 * Inter_Quartile_Range)]

#print(InterQuartileRange('Upper_Age'))
```

```
Outlier_Column_List = ['Upper_Age','Lower_Age','Reco_Policy_Premium']

for Column in Outlier_Column_List:
    Outlier_count = 0
    Range = Range = InterQuartileRange(Column)
    print(f'Inter Quartile Range for {Column} Column is : {Range}')

    for i in Dataset[Column]:
        if i < Range[0] or i > Range[1]:
            Outlier_count += 1
    print(f'There are {Outlier_count} Outliers in the given column \n')
```

```
Inter Quartile Range for Upper_Age Column is : [-18.5, 105.5]
There are 0 Outliers in the given column

Inter Quartile Range for Lower_Age Column is : [-18.0, 102.0]
There are 0 Outliers in the given column
```

There are 821 Outliers in Reco_Policy_premium Column based on the IQR of that column.

```
Range = InterQuartileRange('Reco_Policy_Premium')[1]

'''
This piece of code here means that There are 679 entires where the premium is high and the person lives in an own house and wants a joint
Insurance Account hence these outliers have meaning.
All the other outliers can be removed
'''
count = 0
for index, row in Dataset.iterrows():
    if row['Reco_Policy_Premium'] > Range:
        if row['Accomodation_Type'] == 1 and row['Reco_Insurance_Type'] == 1:
            count += 1
            #print(row)
print('Number of Rows that meet the above codition : ',count)
```

```
Number of Rows that meet the above codition :  679
```

```
# Total number of Outliers
Range = InterQuartileRange('Reco_Policy_Premium')[1]
count = 0
for i in Dataset['Reco_Policy_Premium']:
    if i > Range:
        count += 1
print(f'There are {count} outliers in this Column')

'''
The outliers who are having a rented house or an individual account can be removed as they
have high insurance premium but live in a rented home or taking an individual account
'''

Condition = (Dataset['Reco_Policy_Premium'] > Range) & ((Dataset['Accomodation_Type'] == 0) | (Dataset['Reco_Insurance_Type'] == 0))
removed_outliers = len(Dataset[Condition])
Dataset_temp = Dataset[~Condition]
```

```
There are 821 outliers in this Column
```

```
print(f'Number of outlier rows dropped from the table : {removed_outliers}')
Dataset = Dataset_temp
print('\n Dataset with new rows is of length :',len(Dataset))
```

```
Number of outlier rows dropped from the table : 142

 Dataset with new rows is of length : 50740
```

There can be valuable information even in the Outliers, So we have further filtered the Outliers and categorized them as people who are having accommodation type as rented and Insurance type as individuals with high premium charges and we got 142 such entries and we removed them.

12. Scaling data in column -

There are a wide range of values in the Region_Code column and they might make our Machine Learning model slow in computation. We followed the concept of MIN-MAX Normalization and scaled the values down.

```
# This wide range of value might make our Machine Learning model slow in computation
# Now we are Scaling down the values using Normalization (specifically min-max Normalization)
Column_min = Dataset['Region_Code'].min()
Column_max = Dataset['Region_Code'].max()

temp_dataset = Dataset.copy()

#temp_dataset.head()

temp_dataset['Region_Code'] = (Dataset['Region_Code'] - Column_min) / (Column_max - Column_min)
```

```
Dataset = temp_dataset.copy()

print(f'The value in this column range from {Dataset['Region_Code'].min()} and {Dataset['Region_Code'].max()}')
```

```
The value in this column range from 0.0 and 1.0
```

13. Rearranging Columns -

After all the data cleaning steps all columns are scattered and it is highly necessary for them to be visually organized for easy access and analysis.

Re Arrange Columns

```
Dataset = Dataset[['City_Code','City_Code_encoded','Region_Code','Accomodation_Type',
                   'Reco_Insurance_Type','Upper_Age','Lower_Age','Is_Spouse','Holding_Policy_Duration',
                   'Holding_Policy_Type','Reco_Policy_Cat','Reco_Policy_Premium','Health_Indicator',
                   'Health_Indicator__X1','Health_Indicator__X2','Health_Indicator__X3',
                   'Health_Indicator__X4','Health_Indicator__X5','Health_Indicator__X6',
                   'Health_Indicator__X7','Health_Indicator__X8','Health_Indicator__X9','Response']]
```

Exploratory Data Analysis

Exploratory Data Analysis is a technique used to analyze the datasets, the primary objective of EDA is to understand data before using expensive data methodology. It often uses visual representation of the data to identify patterns and trends. Applying Exploratory data analysis, helps identifying data quality issues and provides valuable insights on data characteristics.

There are two methods for analyzing data: univariate analysis and multivariate analysis.

Univariate Analysis -
Only one variable is taken into account at a time, determining the distribution of individual variables in the dataset is the primary objective. Techniques such as histograms, pie charts and, barcharts are used.

Multivariate Analysis -
Examines several variables at once in order to determine their relationship to one another. The primary goal is to identify patterns and dependencies between variables. Techniques such as scatter plots, cluster analysis are included.

1. Extracting Information from continuous columns -
We have a couple of continuous columns in our dataset and it is necessary for us to understand the characteristics of those columns.

```
# Describing the Dataset
```

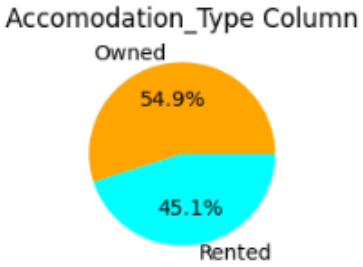
```
Dataset[['Holding_Policy_Duration','Upper_Age','Lower_Age','Reco_Policy_Premium']].describe()
```

	Holding_Policy_Duration	Upper_Age	Lower_Age	Reco_Policy_Premium
count	50740.000000	50740.000000	50740.000000	50740.000000
mean	5.293339	44.786894	42.709342	14127.213039
std	3.836449	17.281844	17.312282	6509.909218
min	1.000000	18.000000	16.000000	2280.000000
25%	4.000000	28.000000	27.000000	9232.000000
50%	4.000000	44.000000	40.000000	13156.000000
75%	6.000000	59.000000	57.000000	18048.000000
max	15.000000	75.000000	75.000000	43350.400000

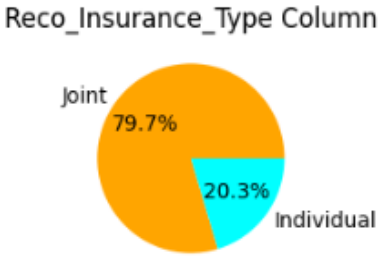
2. Pie plots for binary classification of columns -

Pie plots give easy-to-understand visual representation, the circular form and segmented layout of the chart facilitate a clear interpretation of the bigger and smaller sections.

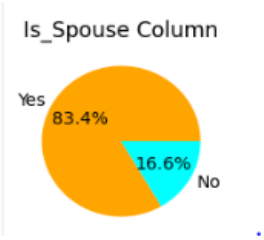
```
# Plotting pie plots for the binary classification columns in the dataframe
# Accomodation Type, Reco Insurance Type , Is Spouse


Pie_Chart_list = {'Accomodation_Type':['Owned','Rented'],'Reco_Insurance_Type':['Joint','Individual'],'Is_Spouse':['Yes','No']}


for Column in Pie_Chart_list.keys():
    Range_of_values = Dataset[Column].value_counts()

    plt.figure(figsize=(2, 2))
    plt.pie(Range_of_values, labels=Pie_Chart_list[Column], autopct='%1.1f%%',colors=['orange','cyan'])
    plt.title(f'{Column} Column')
    plt.show()
```



The pie chart represents the accommodation type of the customers, we have 54.9% owned houses and 45.1% of customers in rented houses.



The chart indicates whether insurance is for an individual or a group, as shown in the pie chart as follows: 20.3% is taken as an individual, and 79.7% is taken as a joint.



The chart indicates whether the insured individual has coverage for their spouse under their insurance plan, 83.4% as yes, 16.6% no.

3. Relation between Accomodation Type, Reco Insurance Type , Is Spouse together and Response

Now that we have seen how the values are ranging in the above columns let us check how these values are depending on the final target i.e Response Column.

```
#How the Columns Accomodation_Type,Reco_Insurance_Type,Is_Spouse have effect on Response Column
grouped_data = Dataset.groupby(['Accomodation_Type', 'Reco_Insurance_Type', 'Is_Spouse'])['Response'].value_counts().unstack()
grouped_data
```

		Response	0	1
Accomodation_Type	**Reco_Insurance_Type**	**Is_Spouse**		
Owned	**Individual**	**No**	15376	4886
	Joint	**No**	930	335
		Yes	4882	1542
Rented	**Individual**	**No**	15520	4754
	Joint	**No**	477	182
		Yes	1488	510

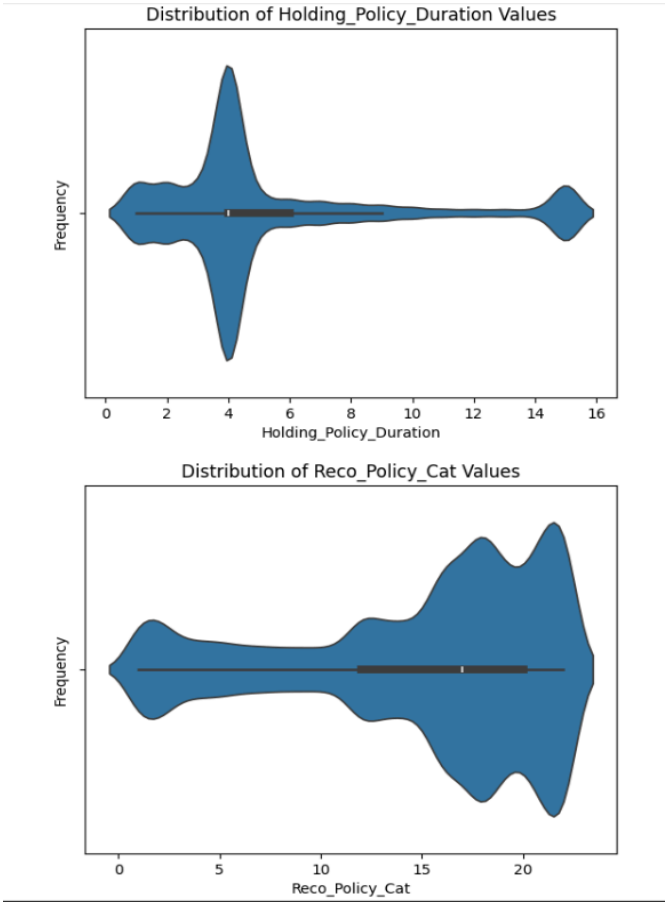
Here the Response columns show the count of response for each combination of columns.

4. Violin plots -

Plot gives a clear understanding of the data, displays the existence of outliers in the data as well as the distribution (interquartile range) and central tendency (median). Which makes comparison between groups much easier.

Performed on - Holding_Policy_Duration , Reco_Policy_Duration

```
# Ploting how the values in the Health Indicator Column are spread

Violen_column_List = ['Holding_Policy_Duration','Reco_Policy_Cat']

for Column in Violen_column_List:

    sns.violinplot(x=Column, data=Dataset)
    plt.xlabel(Column)
    plt.ylabel('Frequency')
    plt.title(f'Distribution of {Column} Values')
    plt.show()
# By this we can say that there are more number of people holding policy for 4 years
```



The Holding_policy_duration plot represents the duration for which a policyholder has held their health insurance policy, Maximum at 4 years can be drawn from the graph.
The Reco_policy_Cat is recommended for an individual based on certain factors determined by the graph. Which is maximum at 20.

5. Relation between Holding_Policy_Duration,Reco_Policy_Cat together and Response Column -

Now that we have seen how the values are ranging in the above columns, let us check how the value of Response column (Target) varies with these values.

```
#How the Columns Holding_Policy_Duration,Reco_Policy_Cat have effect on Response Column

grouped_data = Dataset.groupby(['Holding_Policy_Duration', 'Reco_Policy_Cat'])['Response'].value_counts().unstack()
grouped_data.head(60)
```

Output :

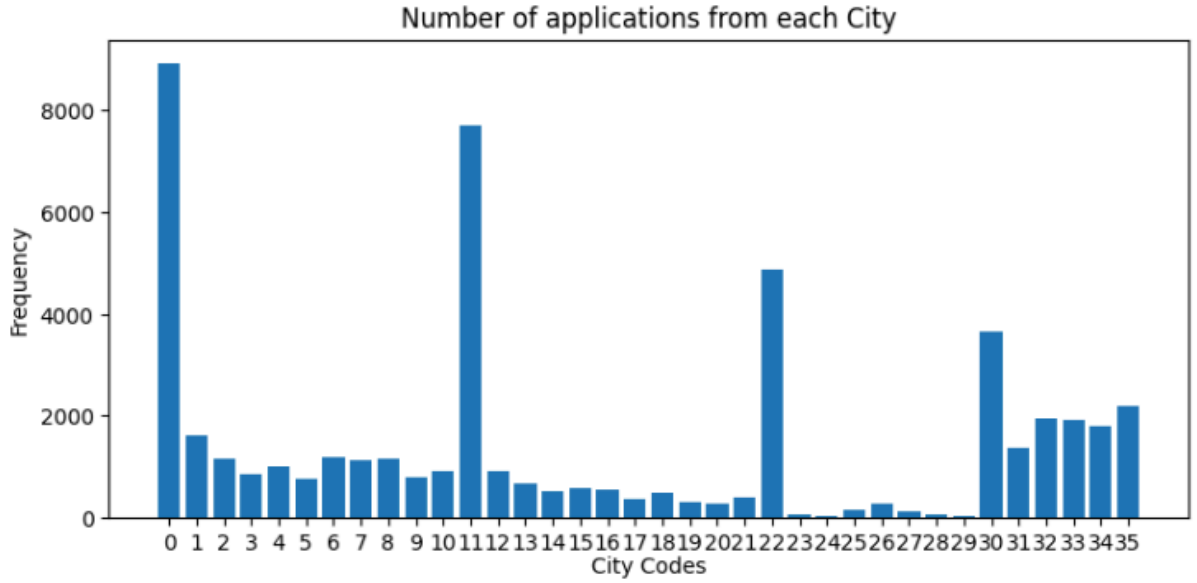
	Response	0	1
Holding_Policy_Duration	**Reco_Policy_Cat**		
1.0	**1**	211.0	1.0
	2	266.0	26.0
	3	48.0	11.0
	4	85.0	7.0
10.0	**1**	27.0	NaN
	2	18.0	3.0
	3	19.0	6.0
	4	5.0	3.0
	5	18.0	14.0
11.0	**1**	21.0	NaN
	2	12.0	2.0
	3	7.0	2.0
	4	8.0	NaN
	5	8.0	3.0

6. Bar graph of insurance applications -

The Bar chart is represented by bars, the bar's height or length reflects the percentage of the information in that class.

Performed on: City_Code

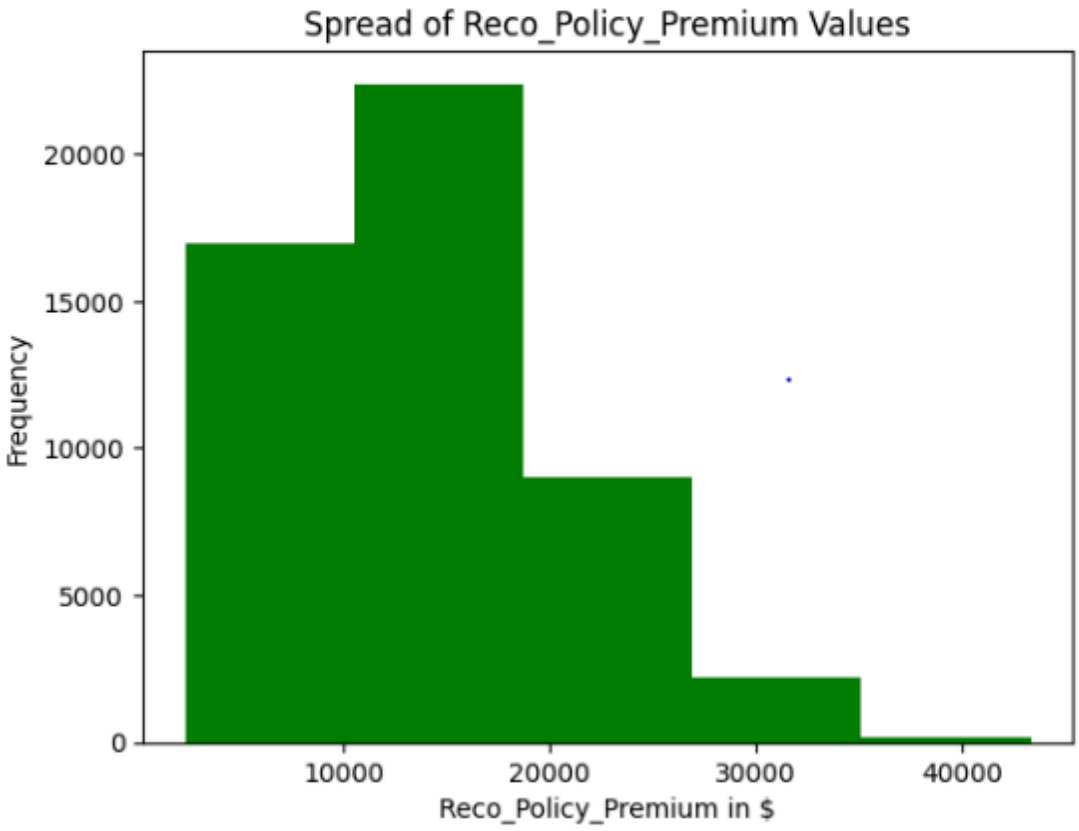
```
Range_of_values = Dataset['City_Code_encoded'].value_counts()
x_label = Range_of_values.index
y_label = Range_of_values.values
min_value = min(Dataset['City_Code_encoded'])
max_value = max(Dataset['City_Code_encoded'])

plt.figure(figsize=(9, 4))
plt.bar(x_label,y_label)
plt.xlabel('City Codes',labelpad =1)
plt.ylabel('Frequency')
plt.title('Number of applications from each City')
plt.xticks(range(int(min_value),int(max_value) + 1))


plt.show()
```



The graph depicts the number of applications from each city, giving a proportion of data points in each category as bars making it easier to understand.

7. Analyzing Reco_Policy_Premium rates -

```
# Majority of the Reco_Policy_Premium rates are between 10k and 20k
plt.hist(Dataset['Reco_Policy_Premium'], bins=5, color='green')
plt.title('Spread of Reco_Policy_Premium Values')
plt.xlabel('Reco_Policy_Premium in $')
plt.ylabel('Frequency')
plt.show()
```



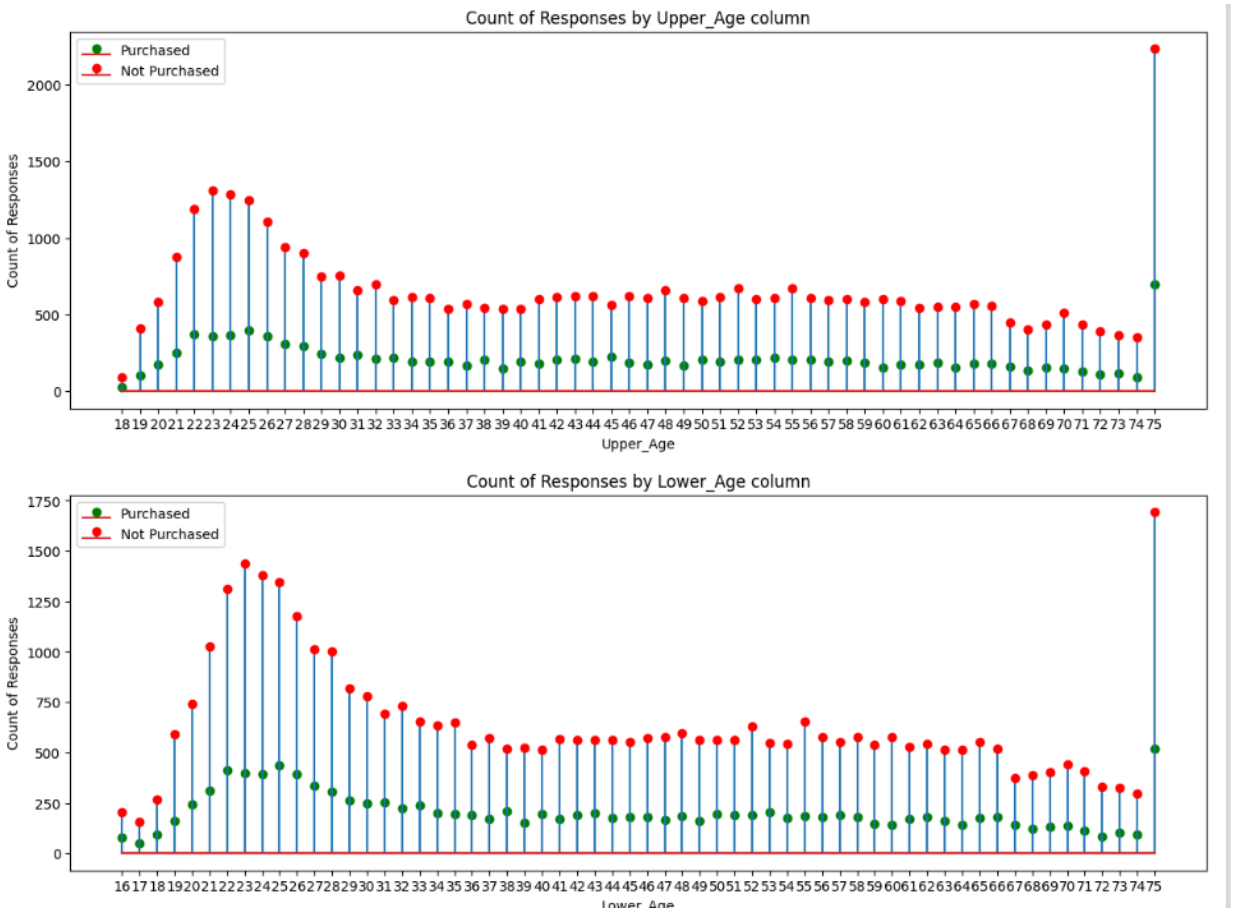
The graph represents the amount of money that an individual or group should pay for a health insurance policy that is recommended to them. Most people fall under 1000 to 2000 dollars insurance policy premium.

8. Stem plots for the count of responses -

Stem plots are also known as stem and leaf plots, which display numerical data points. It clearly depicts individual data points, offering an in depth picture of the distribution.

Performed on: Upper_Age, Lower_Age

```
Age_Column_List = ['Upper_Age','Lower_Age']
#plt.figure(figsize=(15, 3))

for Column in Age_Column_List:
    plt.figure(figsize=(15, 5))
    Response_count = Dataset.groupby([Column, 'Response']).size()
    Response_count = Response_count.unstack()
    #Response_count.head()

    #plt.figure(figsize=(15, 8))


    plt.stem(Response_count.index, Response_count[1], markerfmt = 'go' )
    plt.stem(Response_count.index, Response_count[0], markerfmt = 'ro')


    min_value = min(Dataset[Column])
    max_value = max(Dataset[Column])
    plt.xticks(range(int(min_value),int(max_value) + 1,1))


    plt.title(f'Count of Responses by {Column} column')
    plt.xlabel(Column)
    plt.ylabel('Count of Responses')
    plt.legend(['Purchased', 'Not Purchased'])
    plt.show()
```



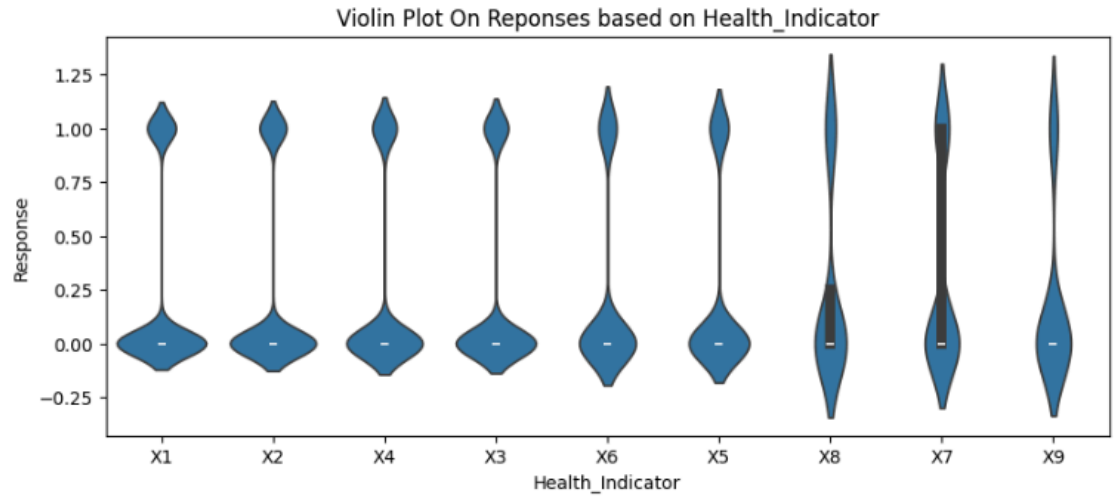
The plot shows the count of responses, if purchased or not purchased, responses of upper age are given in graph 1, responses of lower age are given in graph 2. These graphs show how the response is varied across various age groups.

9. Violin Plot on responses -

Violin plots provide the distribution of the values with a shape and this is great in utilizing space while visualizing and can communicate effectively.
Performed on: Health_Indicator column

The violin plot offers important information on how health indicators are distributed throughout the covered population.

```
# Visualization on How the Response Depends on the Health of a person
plt.figure(figsize=(10, 4))
sns.violinplot(x='Health_Indicator', y='Response', data=Dataset)
plt.title('Violin Plot On Reponses based on Health_Indicator')
plt.xlabel('Health_Indicator')
plt.ylabel('Response')

plt.show()
```



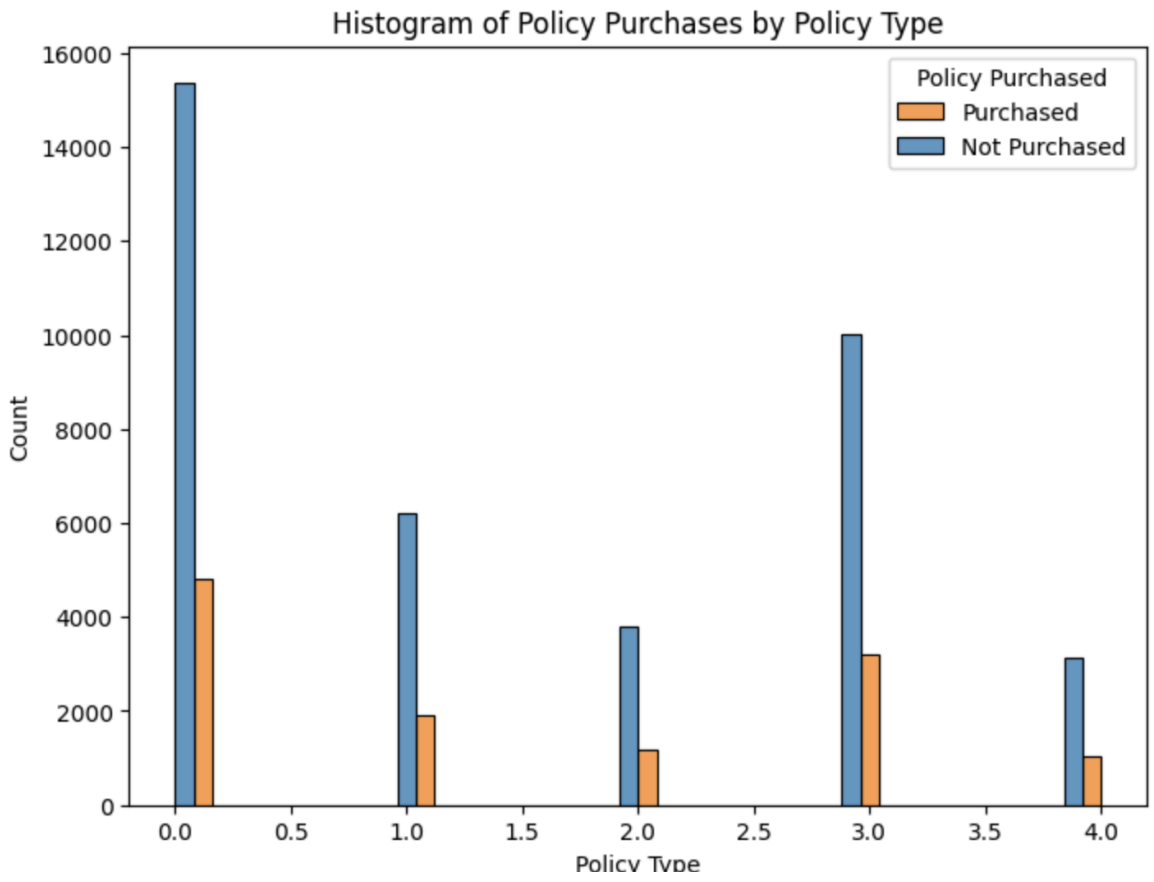
The above is the graph representing the frequency of responses of customers with various health types.

10. Histogram of policy purchases -

The histogram graph represents distribution of numeric data, by contiguous bars. Anomalies can be easily observed if found in the data.
Performed on: Holding_Policy_Type

The graph represents types of policies that are either purchased or not by the customers. The highest count of purchases is more than 15000 and not purchased is below 2000.

```
plt.figure(figsize=(8, 6))
sns.histplot(data=Dataset, x='Holding_Policy_Type', hue = 'Response',multiple='dodge')

plt.title('Histogram of Policy Purchases by Policy Type')

plt.xlabel('Policy Type')
plt.ylabel('Count')

plt.legend(title='Policy Purchased', labels=['Purchased', 'Not Purchased'])

plt.show()
```



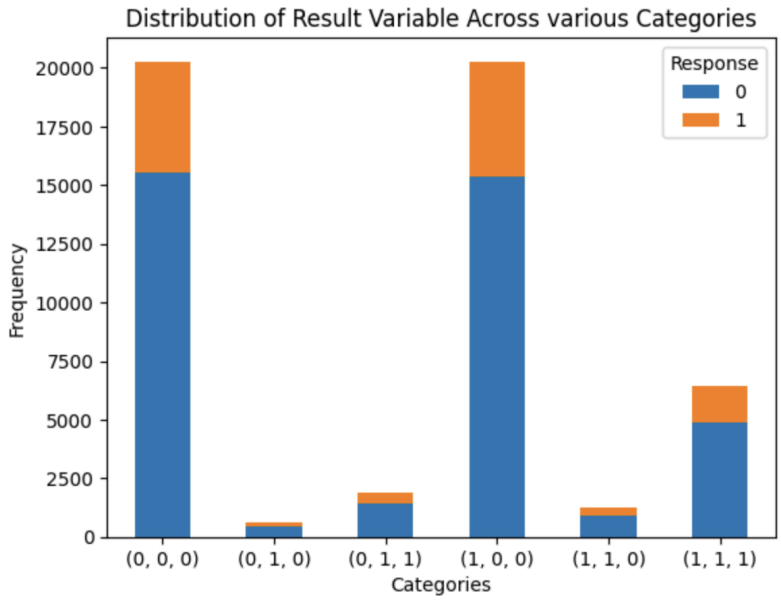
11. Stacked Bar Chart -

This is similar to a normal bar graph but we are using this in a stacked manner to compare multiple columns.This gives us a much clear view in comparing columns.

Performed on: Accomodation_Type, Reco_Insurance_Policy and Is_Spouse

```
plt.figure(figsize=(20, 6))
Column_Group = Dataset.groupby(['Accomodation_Type', 'Reco_Insurance_Type', 'Is_Spouse'])['Response'].value_counts()

temp_plot = Column_Group.unstack()

temp_plot.plot(kind='bar', stacked=True)




plt.title('Distribution of Result Variable Across various Categories')

plt.xlabel('Categories')

plt.ylabel('Frequency')

plt.legend(title='Response')

plt.xticks(rotation=360)

plt.show()
```

```
<Figure size 2000x600 with 0 Axes>
```



Here we are visualizing the effect of the Response column of the dataset on three different columns and they are Accomodation_Type, Reco_Insurance_Policy and Is_Spouse column.In the above graph we have visualized all possible combinations for this and the effect on target column.

References -

Matplotlib :

1. https://matplotlib.org/stable/plot_types/basic/bar.html#sphx-glr-plot-types-basic-bar-py
2. https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_features.html#pie-charts
3. https://matplotlib.org/stable/plot_types/basic/stem.html#sphx-glr-plot-types-basic-stem-py
4. https://matplotlib.org/stable/plot_types/stats/hist_plot.html#sphx-glr-plot-types-stats-hist-plot-py

Pandas :

1. https://pandas.pydata.org/docs/user_guide/index.html#user-guide
2. https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html#pandas.get_dummies

Seaborn :

1. https://seaborn.pydata.org/generated/seaborn.histplot.html
2. https://seaborn.pydata.org/generated/seaborn.violinplot.html