PROJECT REPORT

ON

“HEALTHCARE INSURANCE SYSTEM ANALYSIS AND CROSS SELL PREDICTION”

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE COURSE OF

PG DIPLOMA IN BIG DATA ANALYTICS

SUBMITTED BY

1. MANVENDRA CHOUHAN (220320525004)
2. ROHIT SAWANT (220320525017)
3. SHARDUL RAJE (220320525028)
4. AKSHAY MAMIDWAR (220320525039)

UNDER THE GUIDANCE OF

PROF. NIMESH DAGUR

CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING NOIDA

CERTIFICATE

This is to certify that the project entitled **"Healthcare Insurance System Analysis and Cross Sell Prediction"** is done by **"Manvendra Chouhan (220320525004), Rohit Sawant (220320525017), Shardul Raje (220320525028) and Akshay Mamidwar (220320525039)"** in partial fulfillment of the requirement for the award of Post Graduate Diploma in Big Data Analytics Course.

Mr. Nimesh Dagur

(Project Guide)

ACKNOWLEDGEMENT

We would like to express our gratitude to our guide Prof. Nimesh Dagur for his constant encouragement and mentoring during the course of this project. We consider ourselves extremely fortunate to have the opportunity of associating ourselves with him. This report was made possible by his patience and persistence. Finally, we would like to thank the entire faculty and staff members of CDAC, Noida who have been a strong support system throughout the course of this project.

Manvendra Chouhan

Rohit Sawant

Shardul Raje

Akshay Mamidwar

PG - DBDA March 2022 Batch

CDAC, Noida.

INDEX

ABSTRACT

A HealthCare Insurance Company is facing challenges in enhancing its revenue and understanding the customers so it wants to analyse the competitors company data received from varieties of sources, namely through scrapping and third-party sources.

This analysis will help them to track the behaviour, condition of customers so that to customize offers for them to buy insurance policies and also calculate royalties to those customers who buy policies in past. In addition to this, it will also help the company to predict the cross sell i.e., whether existing customers which bought health insurance are interested to buy company's vehicle insurance too or not. This analysis will help to increase the revenue of the company.

So, the goal of the project is to create data pipelines for the Healthcare insurance company which will help the company to make appropriate business strategies to enhance their revenue by analysing customers behaviours, send offers & royalties to customers respectively and predict cross sell.

CHAPTER 1: INTRODUCTION

1.1 Introduction

A Healthcare insurance is becoming so vital in our life and we seek to buy policies. Having a health insurance policy protects your savings from getting drained due to medical treatments. Medical emergencies are unpredictable, and with rising healthcare costs, quality management can get very expensive. Without medical insurance, there can be a rapid loss of savings. On the other hand, every healthcare insurance company is facing challenges in enhancing its revenue and understanding the customers so it wants to take help of Big Data Ecosystem to analyze the Competitors company data received from varieties of sources, namely through scrapping and third-party sources. This analysis will help them to track the behavior, condition of customers so that to customize offers for them to buy insurance policies and also calculate royalties to those customers who buy policies in past, this in turn will enhance their revenues.

In the financial services industry, cross-selling is a popular term. Cross-selling involves selling complementary products to existing customers. It is one of the highly effective techniques in the marketing industry. To understand better, suppose you are a bank representative and you try to sell a mutual fund or insurance policy to your existing customer.

The main objective behind this method is to increase sales revenue and profit from the already acquired customer base of a company. Cross-selling is perhaps one of the easiest ways to grow the business as they have already established a relationship with the client.

Further, it is more profitable as the cost of acquiring a new customer is comparatively higher.

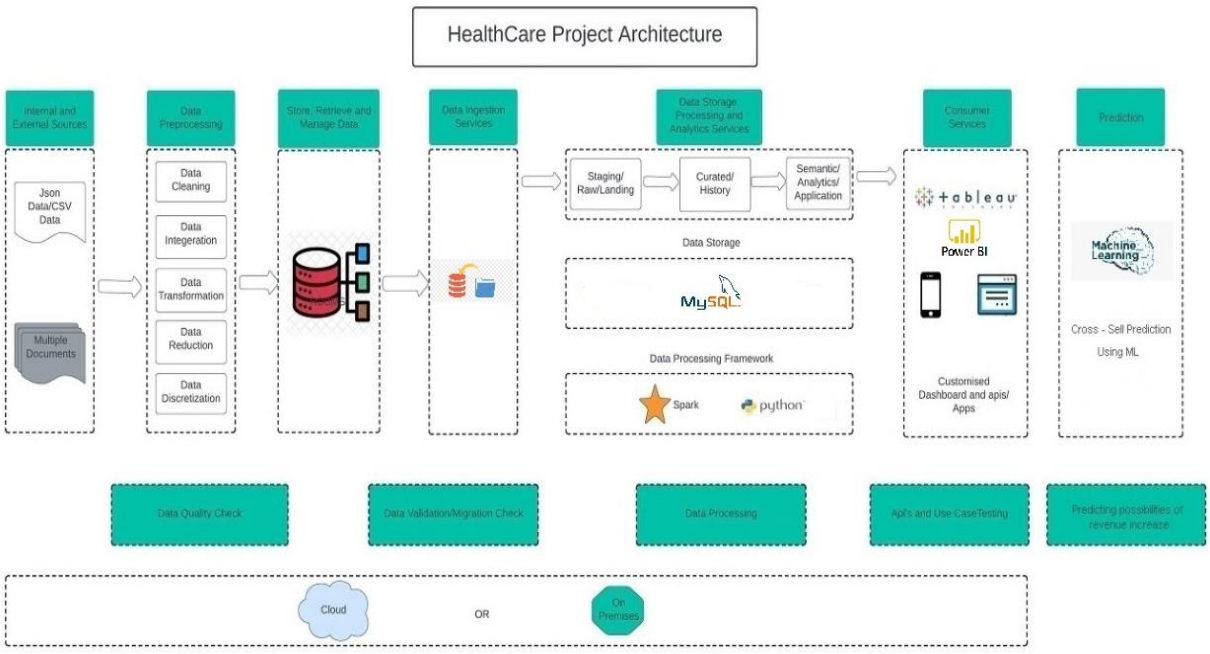
CHAPTER 2: PROCESS FLOW/ ARCHITECTURE

2.1 Process Flow

1. Data files are received in the form of json and csv. These files are coming from the third-party sources based on user interaction methodology.
2. The data files are validated, enriched and processed before loading into RDBMS System.
3. After validating the data files, we are creating the data model for RDBMS so that we can store the data files into the RDBMS.
4. After storing the data into the RDBMS, we transform according to our business requirements.
5. Finally, data needs to be analysed by some analytical queries.
6. After analytical queries, we test the result and use the result to enhance the revenue.
7. Then we train and test the data on various model in order to predict the cross sell in order to enhance revenue.

2.2 Project Architecture

A schematic flow of operations is shown below:



CHAPTER 3: DATASET EXPLAINATION

3.1 Dataset Explanation

Data coming from third-party sources in csv and json format. Data file contains below fields:

1. json file-1 (claims.json)

   - claim_id
   - patient_id
   - disease_name
   - sub_id
   - claimed_or_rej
   - claim_type
   - claim_amount
   - claim_date

2. csv file-2 (disease.csv)

   - subgrp_id
   - disease_id
   - disease_name

3. csv file-3 (groups.csv)

   - country
   - premium_return
   - pincode
   - grp_id
   - grp_name
   - grp_type
   - city
   - year_est
   - grp_sk

4. csv file-4 (hospital.csv)

- hospital_id
- hospital_name
- city
- state
- country

5. csv file-5 (patient.csv)

- patient_id
- patient_name
- gender
- birth_date
- phone
- disease_name
- city
- hospital_id

6. csv file-6 (subgroup.csv)

- subgrp_id
- subgrp_name
- monthly_premium
- subgrp_sk

7. csv file-7 (subscriber.csv)

- s_key
- sub_id
- first_name
- last_name
- street
- birth_date
- gender

- phone
- country
- city
- pincode
- subgrp_id
- elig_ind
- e_date
- t_date

8. csv file-8 (customer.csv)

- id
- gender
- age
- driving_license
- region_code
- previously_insured
- vehicle_age
- vehicle_damage
- annual_premium
- policy_sales_channel
- vintage
- response

CHAPTER 4: DATA PRE-PROCESSING

4.1 Data Cleaning

Following data cleaning steps are carried out:

1. Empty string replacements with actual null
2. Checking duplicate values
3. Data type checks (including date format) and corrections/ rejections
4. File name checks and empty file checks
5. Malformed record checks and rejections

4.2 Data Enriching

Following steps are carried out for data enrichment:

1. Validate the data from the input file and load only valid records into the target table according to the constraints mentioned in the target table
2. Load only the members who are currently effective i.e., SYSDATE BETWEEN EFFT_DT AND TERM_DT
3. Reject records if the Subscriber_Id has less than 9 characters.
4. Populate leading zeroes in the fields GROUP_ID and SUBGRP_ID while populating data into the Target table
5. Also validate the Group Id and Subgrp_Id against the Subgrp table and load only matching data into the target table

4.3 Implementation/ Coding

1. Pre-processing on file patients.csv:

```
# Importing the processing libray..
import numpy as np
import pandas as pd
```

```
# Read the Patient Details
df = pd.read_csv('../../Raw dataset/Patient_records.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70 entries, 0 to 69
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Patient_id          70 non-null     int64 
 1   Patient_name        53 non-null     object
 2   patient_gender      70 non-null     object
 3   patient_birth_date  70 non-null     object
 4   patient_phone       68 non-null     object
 5   disease_name        70 non-null     object
 6   city                70 non-null     object
 7   hospital_id         70 non-null     object
dtypes: int64(1), object(7)
memory usage: 4.5+ KB
```

```
df.describe
```

```
<bound method NDFrame.describe of     Patient_id Patient_name patient_gender patient_birth_date   patient_phone  \
0       187158       Harbir         Female         1924-06-30  +91 0112009318   
1       112766     Brahmdev         Female         1948-12-20  +91 1727749552   
2       199252      Ujjawal           Male         1980-04-16  +91 8547451606   
3       133424      Ballari         Female         1969-09-25  +91 0106026841   
4       172579      Devnath         Female         1946-05-01  +91 1868774631   
..         ...          ...            ...                ...             ...   
65      191132       Dipesh         Female         1949-04-01  +91 5851958964   
66      105686          NaN           Male         1930-09-01  +91 7061843400   
67      160140       Kishan           Male         1923-05-12  +91 9067652693   
68      114252          NaN         Female         1927-02-26  +91 4984346995   
69      188365  Bhageeratha           Male         1973-03-21  +91 0590662722   

      disease_name          city hospital_id  
0     Galactosemia      Rourkela       H1001  
1   Bladder cancer  Tiruvottiyur       H1016  
2    Kidney cancer     Berhampur       H1009  
3          Suicide  Bihar Sharif       H1017  
4     Food allergy   Bidhannagar       H1019  
..             ...           ...         ...  
65        Glaucoma         Kochi       H1016  
66       Hepatitis      Kolhapur       H1008  
67   Rett Syndrome    Srikakulam       H1002  
68        Diabetes     Ambarnath       H1014  
69     Pet allergy       Sonipat       H1017  

[70 rows x 8 columns]>
```

```
df.head(10)
```

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
0	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001
1	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016
2	199252	Ujjawal	Male	1980-04-16	+91 8547451606	Kidney cancer	Berhampur	H1009
3	133424	Ballari	Female	1969-09-25	+91 0106026841	Suicide	Bihar Sharif	H1017
4	172579	Devnath	Female	1946-05-01	+91 1868774631	Food allergy	Bidhannagar	H1019
5	171320	Atasi	Male	1967-10-02	+91 9747336855	Whiplash	Amravati	H1013
6	107794	Manish	Male	1967-06-06	+91 4354294043	Sunbathing	Panvel	H1004
7	130339	Aakar	Female	1925-03-05	+91 2777633911	Drug consumption	Bihar Sharif	H1000
8	110377	Gurudas	Male	1945-05-06	+91 1232859381	Dengue	Kamarhati	H1001
9	149367	NaN	Male	1925-06-12	+91 1780763280	Head banging	Bangalore	H1013

```
# Count the total Null values
df.isnull().sum()
```

```
Patient_id            0
Patient_name         17
patient_gender        0
patient_birth_date    0
patient_phone         2
disease_name          0
city                  0
hospital_id           0
dtype: int64
```

```
# Replace the null values
df["Patient_name"].fillna("NA", inplace = True)
```

```
df.head()
```

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
0	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001
1	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016
2	199252	Ujjawal	Male	1980-04-16	+91 8547451606	Kidney cancer	Berhampur	H1009
3	133424	Ballari	Female	1969-09-25	+91 0106026841	Suicide	Bihar Sharif	H1017
4	172579	Devnath	Female	1946-05-01	+91 1868774631	Food allergy	Bidhannagar	H1019

```
# Replace the null values
df['patient_phone'].fillna("NA", inplace = True)
```

```
# Count the total Null values
df.isnull().sum()
```

```
Patient_id            0
Patient_name          0
patient_gender        0
patient_birth_date    0
patient_phone         0
disease_name          0
city                  0
hospital_id           0
dtype: int64
```

```
# Check the duplicates
df.duplicated()
```

```
0     False
1     False
2     False
3     False
4     False
      ...
65    False
66    False
67    False
68    False
69    False
Length: 70, dtype: bool
```

```
# Drop duplicates
df.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)
```

	Patient_id	Patient_name	patient_gender	patient_birth_date	patient_phone	disease_name	city	hospital_id
0	187158	Harbir	Female	1924-06-30	+91 0112009318	Galactosemia	Rourkela	H1001
1	112766	Brahmdev	Female	1948-12-20	+91 1727749552	Bladder cancer	Tiruvottiyur	H1016
2	199252	Ujjawal	Male	1980-04-16	+91 8547451606	Kidney cancer	Berhampur	H1009
3	133424	Ballari	Female	1969-09-25	+91 0106026841	Suicide	Bihar Sharif	H1017
4	172579	Devnath	Female	1946-05-01	+91 1868774631	Food allergy	Bidhannagar	H1019
...	...	...	...	...	...	...	...	...
65	191132	Dipesh	Female	1949-04-01	+91 5851958964	Glaucoma	Kochi	H1016
66	105686	NA	Male	1930-09-01	+91 7061843400	Hepatitis	Kolhapur	H1008
67	160140	Kishan	Male	1923-05-12	+91 9067652693	Rett Syndrome	Srikakulam	H1002
68	114252	NA	Female	1927-02-26	+91 4984346995	Diabetes	Ambarnath	H1014
69	188365	Bhageeratha	Male	1973-03-21	+91 0590662722	Pet allergy	Sonipat	H1017

70 rows × 8 columns

```
# Check Patient Id not less than 6 digit
count = 0
for i in df['Patient_id'].values:

    if len(str(i))<6:
        df.drop([count], axis=0, inplace=True)
    elif len(str(i))>6:
        df.drop([count], axis=0, inplace=True)
    count = count+1
```

```
# Count the patient_gender types
df['patient_gender'].unique()
```

```
array(['Female', 'Male'], dtype=object)
```

```
# Export the Data
df.to_csv("../patient.csv",index=False)
```

2. Pre-processing on file subscriber.csv:

```
# Read the subscriber Details
df = pd.read_csv('../../Raw dataset/subscriber.csv')
```

```
df.head()
```

	sub _id	first_name	last_name	Street	Birth_date	Gender	Phone	Country	City	Zip Code	Subgrp_id	Elig_ind	eff_date	term_date
0	SUBID10000	Harbir	Vishwakarma	Baria Marg	1924-06-30	Female	+91 0112009318	India	Rourkela	767058	S107	Y	1944-06-30	1954-01-14
1	SUBID10001	Brahmdev	Sonkar	Lala Marg	1948-12-20	Female	+91 1727749552	India	Tiruvottiyur	34639	S105	Y	1968-12-20	1970-05-16
2	SUBID10002	Ujjawal	Devi	Mammen Zila	1980-04-16	Male	+91 8547451606	India	Berhampur	914455	S106	N	2000-04-16	2008-05-04
3	SUBID10003	Ballari	Mishra	Sahni Zila	1969-09-25	Female	+91 0106026841	India	Bihar Sharif	91481	S104	N	1989-09-25	1995-06-05
4	SUBID10004	Devnath	Srivastav	Magar Zila	1946-05-01	Female	+91 1868774631	India	Bidhannagar	531742	S110	N	1966-05-01	1970-12-09

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   sub _id     100 non-null    object
 1   first_name  73 non-null     object
 2   last_name   100 non-null    object
 3   Street      100 non-null    object
 4   Birth_date  100 non-null    object
 5   Gender      100 non-null    object
 6   Phone       97 non-null     object
 7   Country     100 non-null    object
 8   City        100 non-null    object
 9   Zip Code    100 non-null    int64 
 10  Subgrp_id   98 non-null     object
 11  Elig_ind    96 non-null     object
 12  eff_date    100 non-null    object
 13  term_date   100 non-null    object
dtypes: int64(1), object(13)
memory usage: 11.1+ KB
```

```
# Count the total Null values
df.isnull().sum()
```

```
sub _id        0
first_name    27
last_name      0
Street         0
Birth_date     0
Gender         0
Phone          3
Country        0
City           0
Zip Code       0
Subgrp_id      2
Elig_ind       4
eff_date       0
term_date      0
dtype: int64
```

```
# Replace the null values
df["first_name"].fillna("NA", inplace = True)
```

```
# Replace the null values
df["Phone"].fillna("NA", inplace = True)
```

```
# Replace the null values
df["Subgrp_id"].fillna("NA", inplace = True)
```

```
# Replace the null values
df["Elig_ind"].fillna("N", inplace = True)
```

```
# Check the duplicates
df.duplicated()
```

```
0     False
1     False
2     False
3     False
4     False
      ...
95    False
96    False
97    False
98    False
99    False
Length: 100, dtype: bool
```

```
# Drop duplicates
df.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)
```

```
# Check Subscriber Id not less than 9 digit
count = 0
for i in df['sub _id'].values:

    if len(i)<9:
        df.drop([count], axis=0, inplace=True)
    elif len(str(i))>10:
        df.drop([count], axis=0, inplace=True)
    count = count+1
```

```
# Check the Elig_ind types
df['Elig_ind'].unique()
```

```
array(['Y', 'N'], dtype=object)
```

```
# Check always eff_date is greater than term_date
count = 0
for x,y in df[['eff_date','term_date']].values:
    dob1 = datetime.strptime(x,'%Y-%m-%d').date()
    dob2 = datetime.strptime(y,'%Y-%m-%d').date()
    if dob1 > dob2:
        df.drop([count], axis=0, inplace=True)
    count = count + 1
```

```
# Export the Data
df.to_csv('../subscriber.csv',index=False)
```

3. Pre-processing on file claims.json:

```
# Importing the processing Libray..
import numpy as np
import pandas as pd
from datetime import datetime
```

```
df = pd.read_json('../../Raw dataset/claims.json')
```

```
df.head()
```

	claim_id	patient_id	disease_name	SUB_ID	Claim_Or_Rejected	claim_type	claim_amount	claim_date
0	0	187158	Galactosemia	SUBID1000	N	claims of value	79874	1949-03-14
1	1	112766	Bladder cancer	SUBID10001	NaN	claims of policy	151142	1970-03-16
2	2	199252	Kidney cancer	SUBID10002	NaN	claims of value	59924	2008-02-03
3	3	133424	Suicide	SUBID10003	NaN	claims of fact	143120	1995-02-08
4	4	172579	Food allergy	SUBID10004	Y	claims of value	168634	1967-05-23

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70 entries, 0 to 69
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   claim_id            70 non-null     int64 
 1   patient_id          70 non-null     int64 
 2   disease_name        70 non-null     object
 3   SUB_ID              70 non-null     object
 4   Claim_Or_Rejected   70 non-null     object
 5   claim_type          70 non-null     object
 6   claim_amount        70 non-null     int64 
 7   claim_date          70 non-null     object
dtypes: int64(3), object(5)
memory usage: 4.5+ KB
```

```
# Count the total Null values
df.isnull().sum()
```

```
claim_id            0
patient_id          0
disease_name        0
SUB_ID              0
Claim_Or_Rejected   0
claim_type          0
claim_amount        0
claim_date          0
dtype: int64
```

```
# Replace the null values
df['Claim_Or_Rejected'].mask(df['Claim_Or_Rejected'] == 'NaN', 'N', inplace=True)
```

```
# Check the duplicates
df.duplicated()
```

```
0     False
1     False
2     False
3     False
4     False
      ...
65    False
66    False
67    False
68    False
69    False
Length: 70, dtype: bool
```

```
# Drop duplicates
df.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)
```

	claim_id	patient_id	disease_name	SUB_ID	Claim_Or_Rejected	claim_type	claim_amount	claim_date
0	0	187158	Galactosemia	SUBID1000	N	claims of value	79874	1949-03-14
1	1	112766	Bladder cancer	SUBID10001	N	claims of policy	151142	1970-03-16
2	2	199252	Kidney cancer	SUBID10002	N	claims of value	59924	2008-02-03
3	3	133424	Suicide	SUBID10003	N	claims of fact	143120	1995-02-08
4	4	172579	Food allergy	SUBID10004	Y	claims of value	168634	1967-05-23
...	...	...	...	...	...	...	...	...
65	65	191132	Glaucoma	SUBID1065	Y	claims of policy	81980	1969-05-31
66	66	105686	Hepatitis	SUBID10066	N	claims of fact	13667	1957-09-12
67	67	160140	Rett Syndrome	SUBID1067	N	claims of value	109433	1944-12-25

```
# Check Subscriber Id not Less than 9 digit
count = 0
for i in df['SUB_ID'].values:

    if len(i)<9:
        df.drop([count], axis=0, inplace=True)
    elif len(str(i))>10:
        df.drop([count], axis=0, inplace=True)
    count = count+1
```

```
# Check Patient Id not Less than 6 digit
count = 0
for i in df['patient_id'].values:

    if len(str(i))<6:
        df.drop([count], axis=0, inplace=True)
    elif len(str(i))>6:
        df.drop([count], axis=0, inplace=True)
    count = count+1
```

```
# Check the Claim_Or_Rejected types
df['Claim_Or_Rejected'].unique()
```

```
array(['N', 'Y'], dtype=object)
```

```
# Check the claim_type types
df['claim_type'].unique()
```

```
array(['claims of value', 'claims of policy', 'claims of fact'],
      dtype=object)
```

```
# Export the Data
df.to_json('../claims.json')
```

4. Pre-processing on group.csv and subgroup.csv:

```
# Importing the processing Library..
import numpy as np
import pandas as pd
```

```
# Read the Groups Details
df1 = pd.read_csv('../../Raw dataset/group.csv',header=None)
```

```
# Read the Subgroups Details
df2 = pd.read_csv('../../Raw dataset/subgroup.csv',header=None)
```

```
df1.head()
```

	0	1	2	3	4	5	6	7
0	India	72000	482018	GRP101	Life Insurance Corporation of India	Govt.	Mumbai	1956
1	India	45000	482049	GRP102	HDFC Standard Life Insurance Co. Ltd.	Private	Mumbai	2000
2	India	64000	482030	GRP103	Max Life Insurance Co. Ltd.	Private	Delhi	2000
3	India	59000	482028	GRP104	ICICI Prudential Life Insurance Co. Ltd.	Private	Mumbai	2000
4	India	37000	482014	GRP105	Kotak Mahindra Life Insurance Co. Ltd.	Private	Mumbai	2001

```
df1.head()
```

	0	1	2	3	4	5	6	7
0	India	72000	482018	GRP101	Life Insurance Corporation of India	Govt.	Mumbai	1956
1	India	45000	482049	GRP102	HDFC Standard Life Insurance Co. Ltd.	Private	Mumbai	2000
2	India	64000	482030	GRP103	Max Life Insurance Co. Ltd.	Private	Delhi	2000
3	India	59000	482028	GRP104	ICICI Prudential Life Insurance Co. Ltd.	Private	Mumbai	2000
4	India	37000	482014	GRP105	Kotak Mahindra Life Insurance Co. Ltd.	Private	Mumbai	2001

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58 entries, 0 to 57
Data columns (total 8 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       58 non-null     object
 1   1       58 non-null     int64 
 2   2       58 non-null     int64 
 3   3       58 non-null     object
 4   4       58 non-null     object
 5   5       58 non-null     object
 6   6       58 non-null     object
 7   7       58 non-null     int64 
dtypes: int64(3), object(5)
memory usage: 3.8+ KB
```

```
# Count the total Null values
df1.isnull().sum()
```

```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
dtype: int64
```

```
df2.head()
```

	0	1	2	3
0	S101	Deficiency Diseases	3000	GRP101,GRP105
1	S102	Accident	1000	GRP110,GRP150,GRP136
2	S103	Physiology	2000	GRP122,GRP108,GRP138,GRP148
3	S104	Therapy	1500	GRP103,GRP113,GRP123,GRP133,GRP143
4	S105	Allergies	2300	GRP153,GRP104,GRP114,GRP124

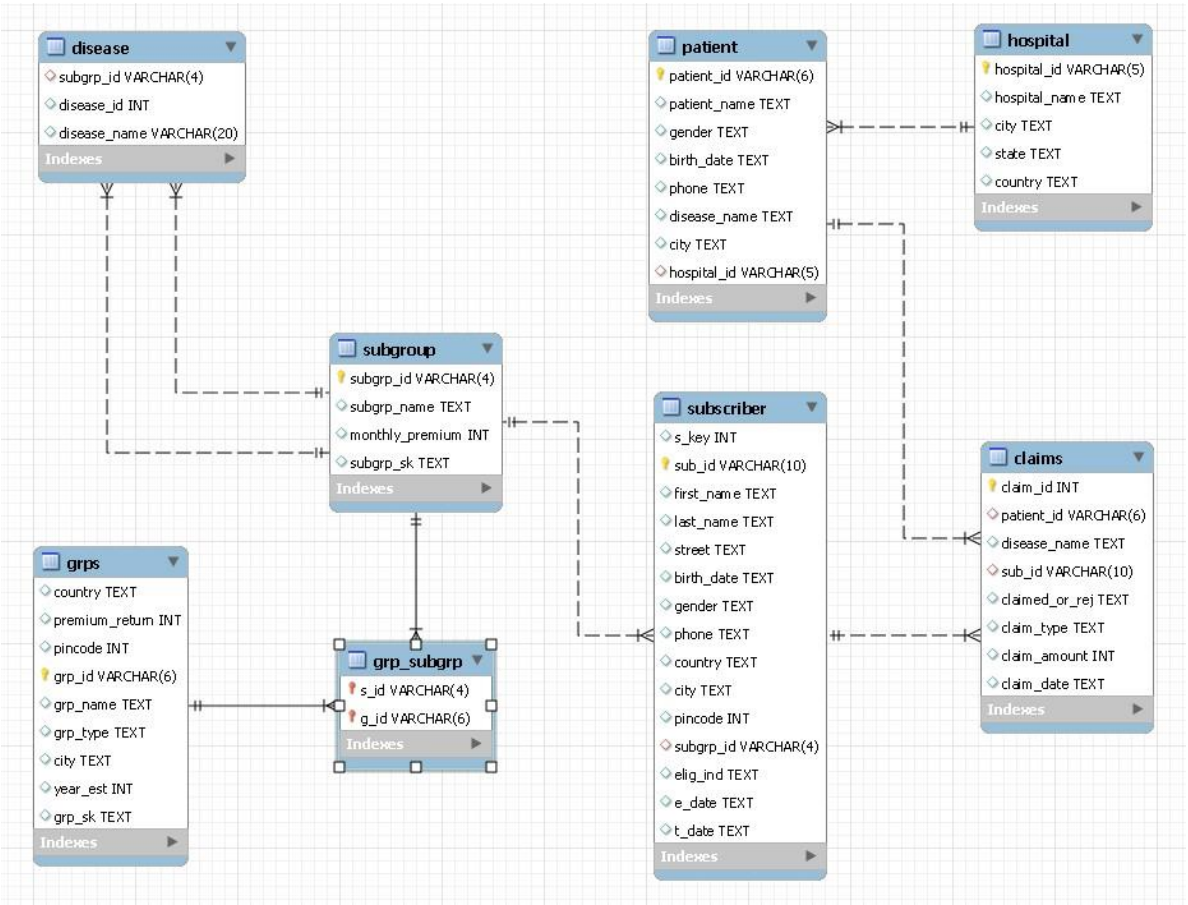
```
df2.isnull().sum()
```

```
0    0
1    0
2    0
3    0
dtype: int64
```

```
# Export Data
df1.to_csv("../group.csv",index=False)
df2.to_csv("../subgroup.csv",index=False)
```

CHAPTER 5: DATABASE/ SCHEMA DESIGN

5.1 Schema Design for SQL Database



CHAPTER 6: DATA ANALYSIS

6.1 Analytical Queries

Once the data is ready for analysis, below analysis has been performed on a batch basis:

1. Find those Subscribers having age less than 30 and they subscribe any subgroup. The output can be in form of a file with columns.
2. Which groups of policies subscriber subscribe mostly Government or private. The output can be in form of a file with columns.
3. List female patients over the age of 40 that have undergone knee surgery in the past year. The output can be in form of a file with columns.
4. Give the Most Profitable subgroup which subscribe the greatest number of times. The output can be in form of a file with columns.
5. Give out which groups has maximum subgroups (Policies Groups). The output can be in form of a file with columns.
6. Give the result from where most of the claims are coming (city). The output can be in form of a file with columns.
7. List all the patients whose age is below 18 and who admit for cancer in the hospital. The output can be in form of a file with columns.
8. List patients who have cashless insura0nce and have total charges greater than or equal for Rs. 50,000. The output can be in form of a file with columns.
9. Find out total number of claims which were rejected by the groups (insurance companies). The output can be in form of a file with columns.
10. Give out which disease having maximum number of claims. The output can be in form of a file with columns.

6.2 Implementation/ Coding (Spark)

After uploading the data into RDBMS, we analysed the data with the help of pyspark by establishing connection by using MySQL connector.

Some snippet of the code and result is as follows:

Data Analysis (Spark)

```
# Import Spark
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
```

```
'C:\\Program Files\\spark'
```

```
# Start Spark Session
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
conf = pyspark.SparkConf().setAppName('Spark App').setMaster('local')
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)
```

```
# Connect to MySQL
import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="mysql"
)
print(mydb)
```

```
<mysql.connector.connection_cext.CMySQLConnection object at 0x000001D82CBD1D30>
```

```
# List Databases
mycursor = mydb.cursor()
mycursor.execute("show databases")
myresult = mycursor.fetchall()
for x in myresult:
    print(x)
```

```
('cdacnoida',)
('classwork',)
('exercise',)
('healthcare_system',)
('information_schema',)
('mysql',)
('performance_schema',)
('practice',)
('project',)
('sakila',)
('sys',)
('vitawa_sevakendra',)
('world',)
```

```
# Establish connection to database
import mysql
import mysql.connector
connection=mysql.connector.Connect(host='localhost',database='healthcare_system',user='root',password='mysql')
if connection.is_connected():
    db_info = connection.get_server_info()
    print("Connection successfully established to",db_info)
else:
    print("Connection Failed")
```

```
Connection successfully established to 8.0.28
```

```
cursor=connection.cursor()
cursor.execute("select database();")
record=cursor.fetchone()
print("You are connected to",record)
```

```
You are connected to ('healthcare_system',)
```

```
# List Tables
cursor=connection.cursor()
cursor.execute("Show Tables;")
record=cursor.fetchall()
for x in record:
    print(x)
```

```
('claims',)
('disease',)
('grp_subgrp',)
('grps',)
('hospital',)
('patient',)
('subgroup',)
('subscriber',)
```

```
# Import sql functions
from pyspark.sql.functions import *
```

```
# Describe the disease table
df=spark.read.csv("C:/Users/shard/OneDrive/CDAC/Project/OurWork/disease.csv",sep=";",inferSchema=True,header=True)
df.createOrReplaceTempView("disease")
spark.sql("desc disease").show()
```

```
+------------+---------+-------+
|    col_name|data_type|comment|
+------------+---------+-------+
|   subgrp_id|   string|   null|
|  disease_id|      int|   null|
|disease_name|   string|   null|
+------------+---------+-------+
```

```
# Describe the groups table
df1=spark.read.csv("C:/Users/shard/OneDrive/CDAC/Project/OurWork/groups.csv",sep=";",inferSchema=True,header=True)
df1.createOrReplaceTempView("groups")
spark.sql("desc groups").show()
```

```
+--------------+---------+-------+
|      col_name|data_type|comment|
+--------------+---------+-------+
|       country|   string|   null|
|premium_return|      int|   null|
|       pincode|      int|   null|
|        grp_id|   string|   null|
|      grp_name|   string|   null|
|      grp_type|   string|   null|
|          city|   string|   null|
|      year_est|      int|   null|
|        grp_sk|   string|   null|
+--------------+---------+-------+
```

```
# Describe the grp_subgrp table
df2=spark.read.csv("C:/Users/shard/OneDrive/CDAC/Project/OurWork/group_subgroup.csv",sep=";",inferSchema=True,header=Tr
df2.createOrReplaceTempView("grp_subgrp")
spark.sql("desc grp_subgrp").show()
```

```
+--------+---------+-------+
|col_name|data_type|comment|
+--------+---------+-------+
|    s_id|   string|   null|
|    g_id|   string|   null|
+--------+---------+-------+
```

```
# Describe the subscriber table
df3=spark.read.csv("C:/Users/shard/OneDrive/CDAC/Project/OurWork/subscriber.csv",sep=";",inferSchema=True,header=True)
df3.createOrReplaceTempView("subscriber")
spark.sql("desc subscriber").show()
```

```
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|     s_key|      int|   null|
|    sub_id|   string|   null|
|first_name|   string|   null|
| last_name|   string|   null|
|    street|   string|   null|
|birth_date|timestamp|   null|
|    gender|   string|   null|
|     phone|   string|   null|
|   country|   string|   null|
|      city|   string|   null|
|   pincode|      int|   null|
| subgrp_id|   string|   null|
|  elig_ind|   string|   null|
|    e_date|timestamp|   null|
|    t_date|timestamp|   null|
+----------+---------+-------+
```

```
# Describe the hospital table
df4=spark.read.csv("C:/Users/shard/OneDrive/CDAC/Project/OurWork/hospital.csv",sep=";",inferSchema=True,header=True)
df4.createOrReplaceTempView("hospital")
spark.sql("desc hospital").show()
```

```
+-------------+---------+-------+
|     col_name|data_type|comment|
+-------------+---------+-------+
|  hospital_id|   string|   null|
|hospital_name|   string|   null|
|         city|   string|   null|
|        state|   string|   null|
|      country|   string|   null|
+-------------+---------+-------+
```

```
# Describe the patient table
df5=spark.read.csv("C:/Users/shard/OneDrive/CDAC/Project/OurWork/patient.csv",sep=",",inferSchema=True,header=True)
df5.createOrReplaceTempView("patient")
spark.sql("desc patient").show()
```

```
+------------+---------+-------+
|    col_name|data_type|comment|
+------------+---------+-------+
|  patient_id|      int|   null|
|patient_name|   string|   null|
|      gender|   string|   null|
|  birth_date|timestamp|   null|
|       phone|   string|   null|
|disease_name|   string|   null|
|        city|   string|   null|
| hospital_id|   string|   null|
+------------+---------+-------+
```

```
# Describe the subgroup table
df6=spark.read.csv("C:/Users/shard/OneDrive/CDAC/Project/OurWork/subgroup.csv",sep=";",inferSchema=True,header=True)
df6.createOrReplaceTempView("subgroup")
spark.sql("desc subgroup").show()
```

```
+---------------+---------+-------+
|       col_name|data_type|comment|
+---------------+---------+-------+
|      subgrp_id|   string|   null|
|    subgrp_name|   string|   null|
|monthly_premium|      int|   null|
|      subgrp_sk|   string|   null|
+---------------+---------+-------+
```

```
# Describe the claims table
df7=spark.read.csv("C:/Users/shard/OneDrive/CDAC/Project/OurWork/claims.csv",sep=";",inferSchema=True,header=True)
df7.createOrReplaceTempView("claims")
spark.sql("desc claims").show()
```

```
+--------------+---------+-------+
|      col_name|data_type|comment|
+--------------+---------+-------+
|      claim_id|      int|   null|
|    patient_id|      int|   null|
|  disease_name|   string|   null|
|        sub_id|   string|   null|
|claimed_or_rej|   string|   null|
|    claim_type|   string|   null|
|  claim_amount|      int|   null|
|    claim_date|timestamp|   null|
+--------------+---------+-------+
```

```
# Which disease having maximum number of claims.
results = spark.sql("select disease_name, count(claim_id) as max from claims group by disease_name order by max desc")
results.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query1.csv')
results.show(10)
```

```
+---------------+---+
|   disease_name|max|
+---------------+---+
|       Glaucoma|  3|
|   Head banging|  3|
|        Anthrax|  3|
|   Galactosemia|  3|
|    Pet allergy|  3|
|Phenylketonuria|  3|
|         Stroke|  2|
|       Diabetes|  2|
| Bladder cancer|  2|
|Fanconi anaemia|  2|
+---------------+---+
only showing top 10 rows
```

```
# Find those Subscribers having age less than 30 and they subscribe any subgroup
subscriber_ages = spark.sql("select birth_date,current_date() as CurrentDate,\
          year(current_date())-year(birth_date) as age from subscriber")
subscriber_ages.sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query2.csv')
subscriber_ages.show()
```

```
+-------------------+-----------+---+
|         birth_date|CurrentDate|age|
+-------------------+-----------+---+
|1924-06-30 00:00:00| 2022-09-23| 98|
|1948-12-20 00:00:00| 2022-09-23| 74|
|1980-04-16 00:00:00| 2022-09-23| 42|
|1969-09-25 00:00:00| 2022-09-23| 53|
|1946-05-01 00:00:00| 2022-09-23| 76|
|1967-10-02 00:00:00| 2022-09-23| 55|
```

```
|1925-03-05 00:00:00| 2022-09-23| 97|
|1945-05-06 00:00:00| 2022-09-23| 77|
|1925-06-12 00:00:00| 2022-09-23| 97|
|1955-01-22 00:00:00| 2022-09-23| 67|
|1964-04-29 00:00:00| 2022-09-23| 58|
|1991-11-11 00:00:00| 2022-09-23| 31|
|1981-01-25 00:00:00| 2022-09-23| 41|
|1966-07-24 00:00:00| 2022-09-23| 56|
|1933-11-20 00:00:00| 2022-09-23| 89|
|1996-10-15 00:00:00| 2022-09-23| 26|
|1935-09-16 00:00:00| 2022-09-23| 87|
|1924-11-09 00:00:00| 2022-09-23| 98|
|1923-09-15 00:00:00| 2022-09-23| 99|
|1920-11-13 00:00:00| 2022-09-23|102|
+-------------------+-----------+---+
only showing top 20 rows
```

```
res = subscriber_ages.filter(subscriber_ages.age < 30).count()
print("Subscribers having age less than 30 --> ",res)
```

```
Subscribers having age less than 30 -->  4
```

```
# Find out which group has maximum subgroups.
sparkdf = spark.sql("select g_id,count(s_id) as max from grp_subgrp group by g_id order by max desc")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query3.csv')
sparkdf.show()
```

```
+------+---+
|  g_id|max|
+------+---+
|GRP143|  2|
|GRP147|  2|
|GRP104|  2|
|GRP105|  1|
|GRP108|  1|
|GRP131|  1|
|GRP102|  1|
|GRP103|  1|
|GRP112|  1|
|GRP101|  1|
|GRP123|  1|
|GRP114|  1|
|GRP127|  1|
|GRP133|  1|
|GRP122|  1|
|GRP138|  1|
|GRP148|  1|
|GRP142|  1|
|GRP157|  1|
|GRP126|  1|
+------+---+
only showing top 20 rows
```

```
# Find out hospital which serve most number of patients
sparkdf = spark.sql("select hospital_name,count(patient_id) as total_patient \
          from hospital join patient on hospital.hospital_id=patient.hospital_id \
          group by hospital_name order by total_patient desc")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query4.csv')
sparkdf.show()
```

```
+--------------------+-------------+
|       hospital_name|total_patient|
+--------------------+-------------+
|   Manipal Hospitals|            9|
|Apollo Hospitals ...|            8|
|Medanta The Medicity|            7|
|Jaslok Hospital a...|            6|
|Indraprastha Apol...|            5|
|Fortis Hospital M...|            4|
|PGIMER - Postgrad...|            4|
|Apollo Hospital -...|            4|
|Bombay Hospital &...|            3|
|Yashoda Hospital ...|            3|
|Apollo Health Cit...|            3|
|King Edward Memor...|            3|
|Lilavati Hospital...|            2|
|The Christian Med...|            2|
|Fortis Hiranandan...|            2|
|Fortis Flt. Lt. R...|            1|
|P. D. Hinduja Nat...|            1|
|Sir Ganga Ram Hos...|            1|
|All India Institu...|            1|
|Breach Candy Hosp...|            1|
+--------------------+-------------+
```

```
# Find out which subgroups subscribe most number of times
sparkdf = spark.sql("select subgrp_name, \
          count(sub_id) as cunt from subgroup join \
          subscriber on subgroup.subgrp_id = subscriber.subgrp_id  group by subgrp_name order by cunt desc")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query5.csv')
sparkdf.show()
```

```
+-------------------+----+
|        subgrp_name|cunt|
+-------------------+----+
|            Therapy|  14|
|              Viral|  11|
|Deficiency Diseases|  11|
|         Hereditary|  11|
|          Allergies|  10|
|         Physiology|  10|
|           Accident|  10|
|             Cancer|   9|
|     Self inflicted|   7|
| Infectious disease|   7|
+-------------------+----+
```

```
# Find out total number of claims which were rejected
sparkdf = spark.sql("select claimed_or_rej,count(claim_id) from claims group by claimed_or_rej")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query6.csv')
sparkdf.show()
```

```
+--------------+---------------+
|claimed_or_rej|count(claim_id)|
+--------------+---------------+
|             Y|             18|
|             N|             52|
+--------------+---------------+
```

```
# From where most claims are comming (city)
sparkdf = spark.sql("select patient.city,count(claim_id) as maxclaim from patient join claims on \
          claims.patient_id = patient.patient_id group by patient.city order by maxclaim desc limit 10")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query7.csv')
sparkdf.show()
```

```
+------------+--------+
|        city|maxclaim|
+------------+--------+
|      Mysore|       2|
|    Amravati|       2|
|   Kamarhati|       2|
|    Jabalpur|       2|
|Bihar Sharif|       2|
|   Ghaziabad|       2|
|       Morbi|       2|
|  Karimnagar|       2|
|   Bangalore|       1|
|     Udaipur|       1|
+------------+--------+
```

```
# Which groups of policies subscriber subscribe mostly Goverment or private
sparkdf = spark.sql("select grp_type,count(grp_id) from groups group by grp_type")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query8.csv')
sparkdf.show()
```

```
+--------+-------------+
|grp_type|count(grp_id)|
+--------+-------------+
|   Govt.|            7|
| Private|           51|
+--------+-------------+
```

```
res = spark.sql("select grp_subgrp.g_id from subscriber \
          join grp_subgrp on grp_subgrp.s_id = subscriber.subgrp_id")

res.registerTempTable("grp_tble")

sparkdf = spark.sql("select grp_type,count(grp_tble.g_id) from groups \
          join grp_tble on groups.grp_id = grp_tble.g_id group by grp_type")

sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query9.csv')

sparkdf.show()
```

```
+--------+-----------+
|grp_type|count(g_id)|
+--------+-----------+
|   Govt.|         35|
| Private|        347|
+--------+-----------+
```

```
# Average monthly premium subscriber pay to insurance company subgroup.
sparkdf = spark.sql("select subgroup.subgrp_id,avg(monthly_premium) from subscriber right join \
          subgroup on subscriber.subgrp_id = subgroup.subgrp_id group by \
          subgroup.subgrp_id order by subgroup.subgrp_id")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query10.csv')
sparkdf.show()
```

```
+---------+--------------------+
|subgrp_id|avg(monthly_premium)|
+---------+--------------------+
|     S101|              3000.0|
|     S102|              1000.0|
|     S103|              2000.0|
|     S104|              1500.0|
|     S105|              2300.0|
|     S106|              1200.0|
|     S107|              3200.0|
|     S108|              1500.0|
|     S109|              2000.0|
|     S110|              1000.0|
+---------+--------------------+
```

```
# Find out Which group is most profitable
sparkdf = spark.sql("select grp_id,premium_return from groups order by premium_return desc")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query11.csv')
sparkdf.show()
```

```
+------+--------------+
|grp_id|premium_return|
+------+--------------+
|GRP147|         99000|
|GRP131|         99000|
|GRP123|         99000|
|GRP118|         97000|
|GRP154|         95000|
|GRP133|         93000|
|GRP157|         92000|
|GRP134|         90000|
|GRP143|         90000|
|GRP106|         89000|
|GRP129|         88000|
|GRP152|         87000|
|GRP153|         86000|
|GRP150|         84000|
|GRP124|         81000|
|GRP122|         79000|
|GRP115|         79000|
|GRP121|         78000|
|GRP109|         78000|
|GRP101|         72000|
+------+--------------+
only showing top 20 rows
```

```
# List all the patients below age of 18 who admit for cancer
sparkdf = spark.sql("select patient_id, patient_name,\
          year(current_date())-year(birth_date) as age from patient \
          where disease_name like '%cancer' order by age limit 2")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query12.csv')
sparkdf.show()
```

```
+----------+------------+---+
|patient_id|patient_name|age|
+----------+------------+---+
|    194166|          NA|  9|
|    197441|    Deependu| 18|
+----------+------------+---+
```

```
# List patients who have cashless insurance and have total charges greater than or equal for Rs. 50,000.
sparkdf = spark.sql("select patient_name, gender, birth_date \
          from patient join claims on patient.patient_id = claims.patient_id \
          where claim_amount >= 50000 and claim_type = 'claims of value'")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query13.csv')
sparkdf.show()
```

```
+------------+------+-------------------+
|patient_name|gender|         birth_date|
+------------+------+-------------------+
|      Harbir|Female|1960-02-24 00:00:00|
|     Ujjawal|  Male|1965-12-31 00:00:00|
|     Devnath|Female|1982-02-22 00:00:00|
|       Aakar|Female|1990-04-24 00:00:00|
```

```
|          NA|  Male|1955-06-03 00:00:00|
|  Dharmadaas|  Male|1964-10-25 00:00:00|
|          NA|Female|1953-04-04 00:00:00|
|     Bhagvan|Female|2011-02-26 00:00:00|
|          NA|Female|1959-01-06 00:00:00|
|       Umang|Female|2017-02-26 00:00:00|
|  Vaijayanti|  Male|1969-04-06 00:00:00|
|       Ekant|  Male|1969-11-01 00:00:00|
|      Gensho|  Male|1991-07-27 00:00:00|
|   Anjushree|  Male|1982-06-28 00:00:00|
|       Saroj|Female|1953-07-21 00:00:00|
|          NA|  Male|1956-04-04 00:00:00|
|  Chitranjan|Female|2020-10-27 00:00:00|
|          NA|  Male|2013-10-30 00:00:00|
|       Lalit|Female|1978-04-30 00:00:00|
|      Kishan|  Male|1955-06-30 00:00:00|
+------------+------+-------------------+
```

```
# List female patients over the age of 40 that have undergone knee surgery in the past year
sparkdf = spark.sql("select patient_name from patient \
                    where gender = 'Female' and disease_name = 'Heart Attack'")
sparkdf.toPandas().to_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query14.csv')
sparkdf.show()
```

```
+------------+
|patient_name|
+------------+
|     Upasana|
+------------+
```

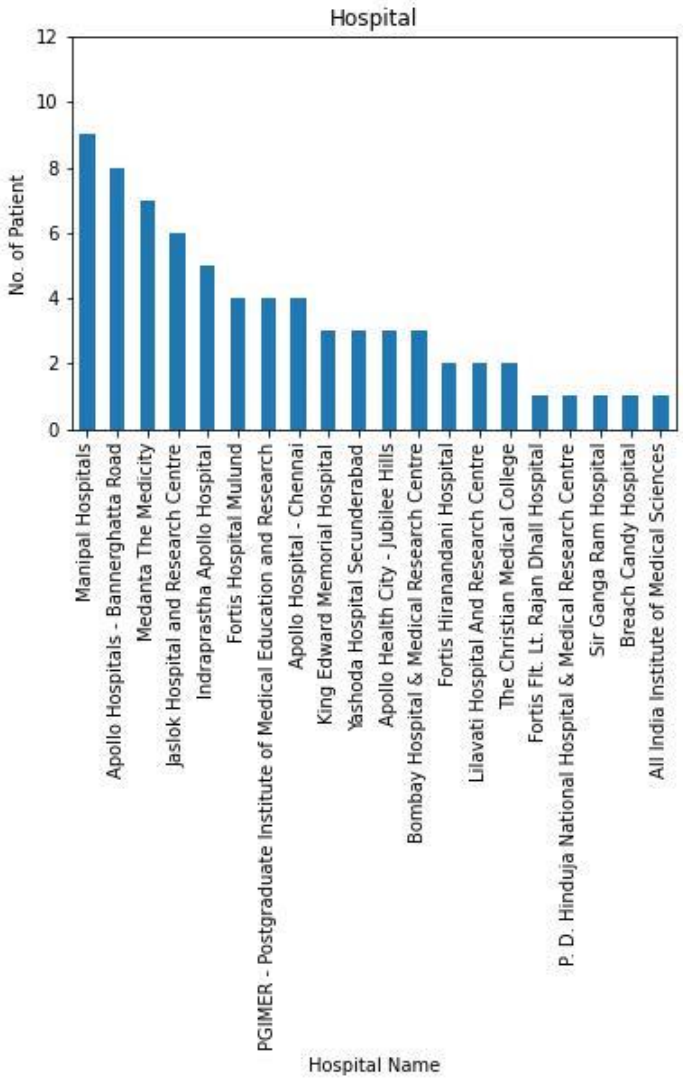
CHAPTER 7: OUTPUT

7.1 Data Visualisation

We used Matplotlib and seaborn to visualize our use cases which will be better to take business decision.

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

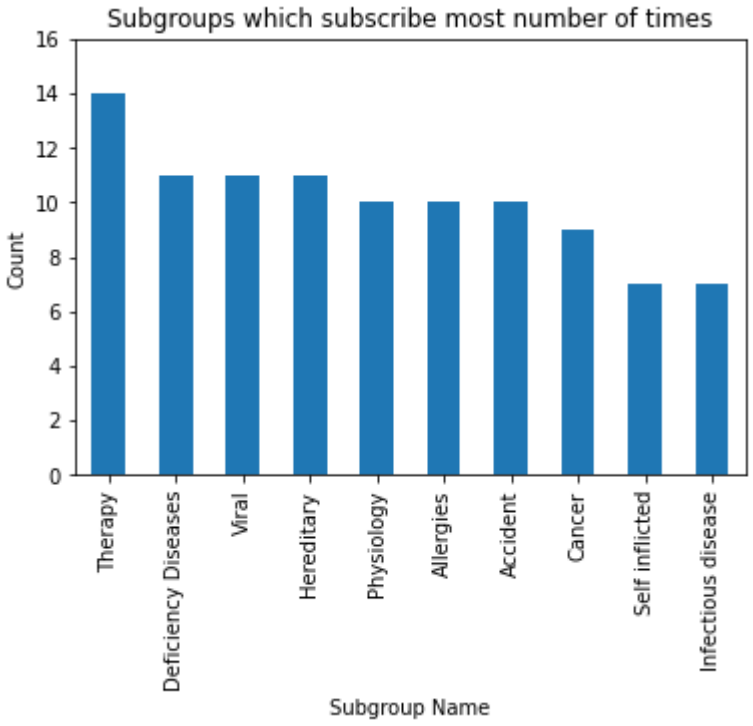
```
# Find out hospital which serve most number of patients
df=pd.read_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query4.csv')
df.plot(x='hospital_name',y='total_patient',kind='bar',legend=None)
plt.title("Hospital")
plt.ylabel('No. of Patient')
plt.xlabel("Hospital Name")
plt.ylim(ymin=0,ymax=12)
plt.savefig('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Visualization/Hospital with most patient.jpg',bbox_inches='ti
```



```
## Find out which subgroups subscribe most number of times
df1=pd.read_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query5.csv')
df1
```

	Unnamed: 0	subgrp_name	cunt
0	0	Therapy	14
1	1	Deficiency Diseases	11
2	2	Viral	11
3	3	Hereditary	11
4	4	Physiology	10
5	5	Allergies	10
6	6	Accident	10
7	7	Cancer	9
8	8	Self inflicted	7
9	9	Infectious disease	7

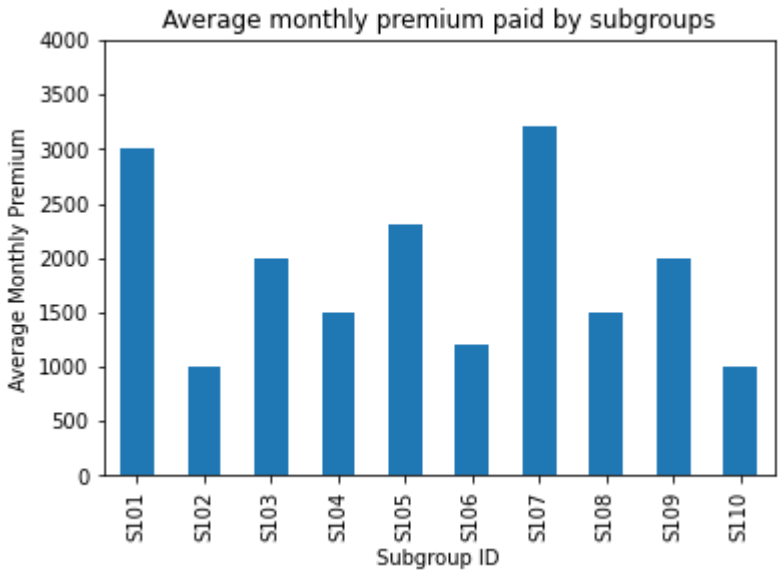
```
df1.plot(x='subgrp_name',y='cunt',kind='bar',legend=None)
plt.title("Subgroups which subscribe most number of times")
plt.ylabel('Count')
plt.xlabel("Subgroup Name")
plt.ylim(ymin=0,ymax=16)
plt.savefig('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Visualization/Subgroups which subscribe most number of times.
```




```
# Average monthly premium subscriber pay to insurance company subgroup.
df2=pd.read_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query10.csv')
df2
```

	Unnamed: 0	subgrp_id	avg(monthly_premium)
0	0	S101	3000.0
1	1	S102	1000.0
2	2	S103	2000.0
3	3	S104	1500.0
4	4	S105	2300.0
5	5	S106	1200.0

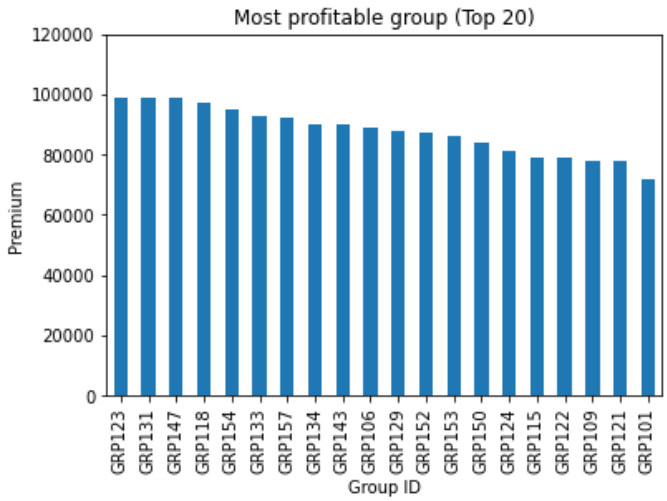
```
df2.plot(x='subgrp_id',y='avg(monthly_premium)',kind='bar',legend=None)
plt.title("Average monthly premium paid by subgroups")
plt.ylabel('Average Monthly Premium')
plt.xlabel("Subgroup ID")
plt.ylim(ymin=0,ymax=4000)
plt.savefig('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Visualization/Average Monthly Premium Paid by Subgroups.jpg',
```




```
# Find out Which group is most profitable
df3=pd.read_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query11.csv')
df3.head()
```

	Unnamed: 0	grp_id	premium_return
0	0	GRP123	99000
1	1	GRP131	99000
2	2	GRP147	99000
3	3	GRP118	97000
4	4	GRP154	95000

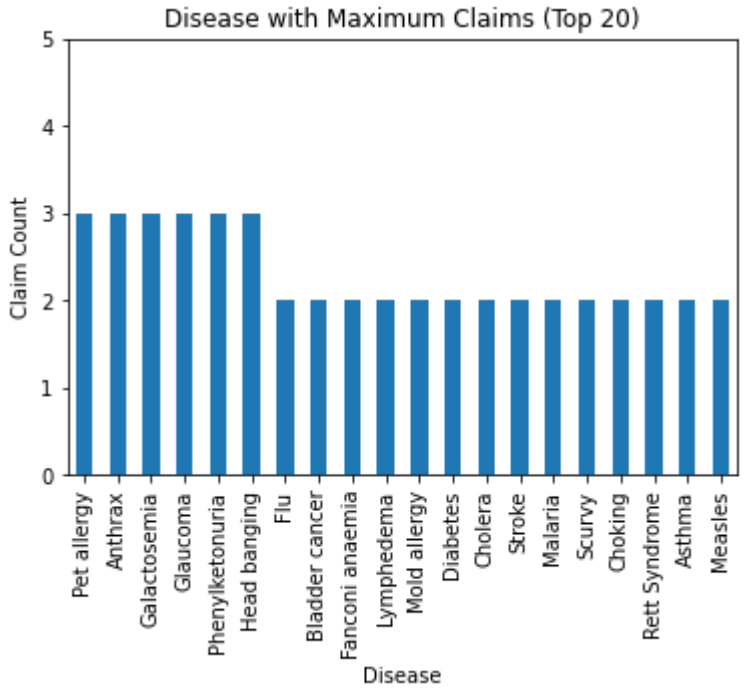
```
df3.head(n=20).plot(x='grp_id',y='premium_return',kind='bar',legend=None)
plt.title("Most profitable group (Top 20)")
plt.xlabel("Group ID")
plt.ylabel("Premium")
plt.ylim(ymin=0,ymax=120000)
plt.savefig('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Visualization/Most Profitable Groups.jpg',bbox_inches='tight'
```



```
# which disease having maximum number of claims.
df4=pd.read_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query1.csv')
df4.head()
```

	Unnamed: 0	disease_name	max
0	0	Pet allergy	3
1	1	Anthrax	3
2	2	Galactosemia	3
3	3	Glaucoma	3
4	4	Phenylketonuria	3

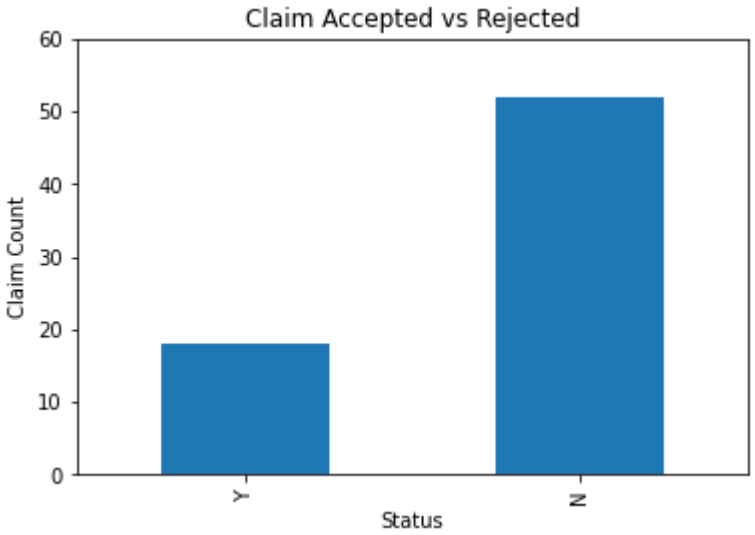
```
df4.head(n=20).plot(x='disease_name',y='max',kind='bar',legend=None)
plt.title("Disease with Maximum Claims (Top 20)")
plt.xlabel("Disease")
plt.ylabel("Claim Count")
plt.ylim(ymin=0,ymax=5)
plt.savefig('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Visualization/Disease with max Claim.jpg',bbox_inches='tight'
```




```
# Find out total number of claims which were rejected
df5=pd.read_csv('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Spark/query6.csv')
df5
```

	Unnamed: 0	claimed_or_rej	count(claim_id)
0	0	Y	18
1	1	N	52

```
df5.head(n=20).plot(x='claimed_or_rej',y='count(claim_id)',kind='bar',legend=None)
plt.title("Claim Accepted vs Rejected")
plt.xlabel("Status")
plt.ylabel("Claim Count")
plt.ylim(ymin=0,ymax=60)
plt.savefig('C:/Users/shard/OneDrive/CDAC/Project/OurWork/Visualization/Claim Accepted vs Rejected.jpg',bbox_inches='ti
```


Claim Accepted vs Rejected
Claim Count
60
50
40
30
20
10
0
Y
N
Status

CHAPTER 8: CROSS SELL PREDICTION

8.1 Hypothesis Generation

After looking at the problem statement, we will now move into hypothesis generation. It is the process of listing out all the possible factors that can affect the outcome.

What is hypothesis generation?

This is a very important stage in any data science/machine learning pipeline. It involves understanding the problem in detail by brainstorming as many factors as possible which can impact the outcome. It is done by understanding the problem statement thoroughly and before looking at the data.

Below are some of the factors which I think can affect the Response (dependent variable for this cross-sell prediction problem):

Gender: Males are more likely to buy Vehicle Insurance.

Age: It is generally said that it is profitable to buy Insurance as early as possible so more likely btw Customers of age between 25-40 age are likely to buy Insurance.

Driving_License: Customers who generally have Driving_License take Insurance.

Previously_Insured: Customers generally take One Vehicle insurance.

Vehicle_Age: The more the vehicle_age the better.

Annual_Premium: Customers generally opt for Insurance where Premium is not too high.

These are some of the factors which can affect the target variable; you can come up with many more factors.

We will be using Python for this course along with the below listed libraries. The version of these libraries is mentioned below:

• pandas • seaborn • sklearn

8.2 Data

For this practice problem, two CSV files are given: train and test.

- Train file will be used for training the model, i.e., our model will learn from this file. It contains all the independent variables and the target variable.
- Test file contains all the independent variables, but not the target variable. We will apply the model to predict the target variable for the test data.

Reading data

```
train=pd.read_csv('/content/train.csv')
test=pd.read_csv('/content/test.csv')
```

Understanding the Data

In this section, we will look at the structure of the train and test datasets. Firstly, we will check the features present in our data and then we will look at their data types.

train

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1
1	2	Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0
2	3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1
3	4	Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0
4	5	Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0
...	...	...	...	...	...	...	...	...	...	...	...	...
381104	381105	Male	74	1	26.0	1	1-2 Year	No	30170.0	26.0	88	0
381105	381106	Male	30	1	37.0	1	< 1 Year	No	40016.0	152.0	131	0
381106	381107	Male	21	1	30.0	1	< 1 Year	No	35118.0	160.0	161	0
381107	381108	Female	68	1	14.0	0	> 2 Years	Yes	44617.0	124.0	74	0
381108	381109	Male	46	1	29.0	0	1-2 Year	No	41777.0	26.0	237	0

381109 rows × 12 columns

We have 11 independent variables and 1 target variable, i.e., Response in the train dataset. Let's also have a look at the columns of test dataset.

test

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	381110	Male	25	1	11.0	1	< 1 Year	No	35786.0	152.0	53
1	381111	Male	40	1	28.0	0	1-2 Year	Yes	33762.0	7.0	111
2	381112	Male	47	1	28.0	0	1-2 Year	Yes	40050.0	124.0	199
3	381113	Male	24	1	27.0	1	< 1 Year	Yes	37356.0	152.0	187
4	381114	Male	27	1	28.0	1	< 1 Year	No	59097.0	152.0	297
...	...	...	...	...	...	...	...	...	...	...	...
127032	508142	Female	26	1	37.0	1	< 1 Year	No	30867.0	152.0	56
127033	508143	Female	38	1	28.0	0	1-2 Year	Yes	28700.0	122.0	165
127034	508144	Male	21	1	46.0	1	< 1 Year	No	29802.0	152.0	74
127035	508145	Male	71	1	28.0	1	1-2 Year	No	62875.0	26.0	265
127036	508146	Male	41	1	29.0	1	1-2 Year	No	27927.0	124.0	231

127037 rows × 11 columns

We have similar features in the test dataset as the train dataset except the Response. We will predict the Response using the model built using the train data.

Given below is the description for each variable:

Variable	Definition
id	Unique ID for the customer
Gender	Gender of the customer
Age	Age of the customer
Driving_License	0: Customer does not have DL, 1: Customer already has DL
Region_Code	Unique code for the region of the customer
Previously_Insured	1: Customer already has Vehicle Insurance, 0: Customer doesn't have Vehicle Insurance
Vehicle_Age	Age of the Vehicle
Vehicle_Damage	1: Customer got his/her vehicle damaged in the past. 0: Customer didn't get his/her vehicle damaged in the past.
Variable	**Definition**
Annual_Premium	The amount customer needs to pay as premium in the year
Policy_Sales_Channel	Anonymised Code for the channel of outreaching to the customer ie. Different Agents, Over Mail, Over Phone, In Person, etc.
Vintage	Number of Days, Customer has been associated with the company
Response	1: Customer is interested, 0: Customer is not interested

The training data set has 12 variables (see above) and Test has 11 (excluding Response).Let's look at the shape of the dataset.

train.shape

```
(381109, 12)
```

test.shape

```
(127037, 11)
```

8.3 Basic EDA

Print data types for each variable

train.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381109 entries, 0 to 381108
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   id                    381109 non-null  int64
 1   Gender                381109 non-null  object
 2   Age                   381109 non-null  int64
 3   Driving_License       381109 non-null  int64
 4   Region_Code           381109 non-null  float64
 5   Previously_Insured    381109 non-null  int64
 6   Vehicle_Age           381109 non-null  object
 7   Vehicle_Damage        381109 non-null  object
 8   Annual_Premium        381109 non-null  float64
 9   Policy_Sales_Channel  381109 non-null  float64
 10  Vintage               381109 non-null  int64
 11  Response              381109 non-null  int64
dtypes: float64(3), int64(6), object(3)
memory usage: 34.9+ MB
```

We can see there are three format of data types:

- **object:** Object format means variables are categorical. Categorical variables in our dataset are: Loan_ID, Gender, Married, Dependents, Education, Self_Employed, Property_Area, Loan_Status

- **int64:** It represents the integer variables.id, Age, Driving_License, Previously_Insured, Vintage, Response is of this format.

- **float64:** It represents the variable which have some decimal values involved. They are also numerical variables. Numerical variables in our dataset are: Region_Code, Annual_Premium, Policy_Sales_Channel

By looking at the info of the dataset we can get a rough idea on the numeric and the string columns.

```
col=train.columns.tolist()
col.remove('id')
train[col].describe(percentiles = [.25,.50,.75,.95,.99])
```

	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage	Response
count	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000
mean	38.822584	0.997869	26.388807	0.458210	30564.389581	112.034295	154.347397	0.122563
std	15.511611	0.046110	13.229888	0.498251	17213.155057	54.203995	83.671304	0.327936
min	20.000000	0.000000	0.000000	0.000000	2630.000000	1.000000	10.000000	0.000000
25%	25.000000	1.000000	15.000000	0.000000	24405.000000	29.000000	82.000000	0.000000
50%	36.000000	1.000000	28.000000	0.000000	31669.000000	133.000000	154.000000	0.000000
75%	49.000000	1.000000	35.000000	1.000000	39400.000000	152.000000	227.000000	0.000000
95%	69.000000	1.000000	47.000000	1.000000	55176.000000	160.000000	285.000000	1.000000
99%	77.000000	1.000000	50.000000	1.000000	72963.000000	160.000000	297.000000	1.000000
max	85.000000	1.000000	52.000000	1.000000	540165.000000	163.000000	299.000000	1.000000

By looking at the summary of the data we can infer the mean,standard deviation, min and max of the data.

We will be able to get a idea on the outliers here by the percentiles (In the Annual_Premium the 99th percentile is 72963 and the max is 540165 this represents the outliers in this column).

8.4 Univariate Analysis

In this section, we will do univariate analysis. It is the simplest form of analyzing data where we examine each variable individually.

For categorical features we can use frequency table or bar plots which will calculate the number of each category in a particular variable.

For numerical features, probability density plots can be used to look at the distribution of the variable.

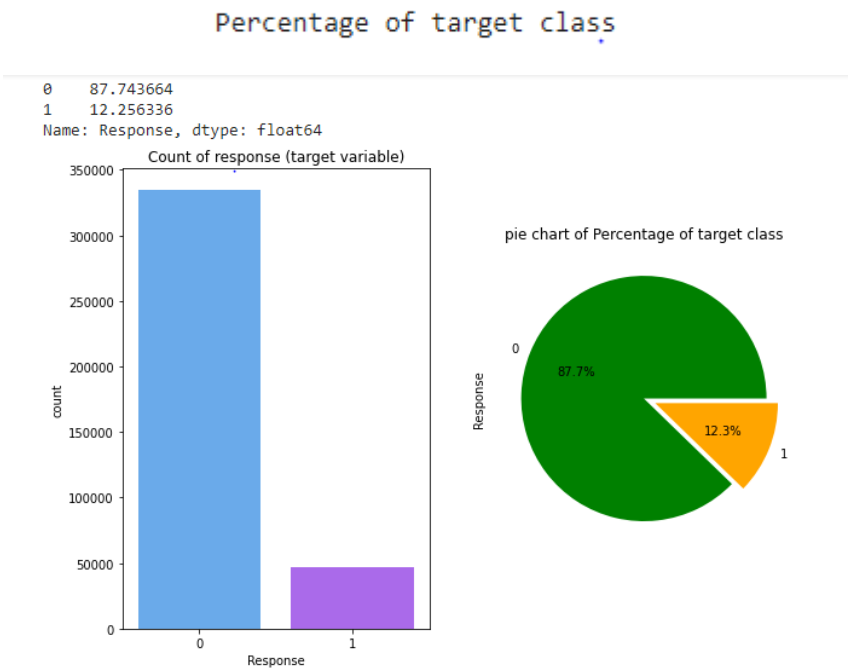
Target Variable

We will first look at the target variable, i.e., **Response**.

As it is a binary classification (1/0) variable, let us look at its frequency table, percentage distribution and bar plot.

Frequency table of a variable will give us the count of each category in that variable.

```
plt.subplot(1,2,1)
sns.countplot(train['Response'],palette='cool')
plt.title("Count of response (target variable)")
plt.subplot(1,2,2)
count=train['Response'].value_counts()
count.plot.pie(autopct = '%1.1f%%',colors=['green','orange'], figsize = (10,7),explode = [0,0
.1],title = "pie chart of Percentage of target class")
print( "Percentage of target class\n")
print(train['Response'].value_counts()/len(train)*100)
```



By the plot we can say that this is the problem of imbalance binary classification problem

The indivisuals interested is 87 % as compared to the othe one.

Now let's visualize each variable separately. Different types of variables are Categorical, ordinal and numerical.

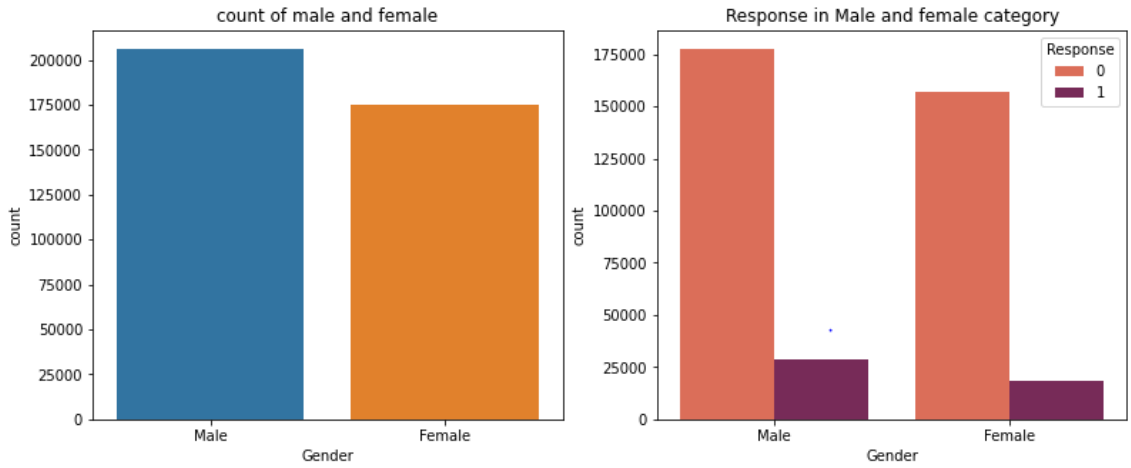
- **Categorical features:** These features have categories (Gender, Previously_Insured, Vehicle_Damage, Response, Driving_License)
- **Ordinal features:** Variables in categorical features having some order involved (Vehicle_Age)
- **Numerical features:** These features have numerical values (id, Age, Region_Code, Annual_Premium, Policy_Sales_Channel Vintage)

Let's visualize the categorical and ordinal features first.

Gender variable

```
plt.figure(figsize = (13,5))
plt.subplot(1,2,1)
sns.countplot(train['Gender'])
plt.title("count of male and female")
```

```
plt.subplot(1,2,2)
sns.countplot(train['Gender'], hue = train['Response'],palette="rocket_r")
plt.title("Response in Male and female category")
plt.show()
```



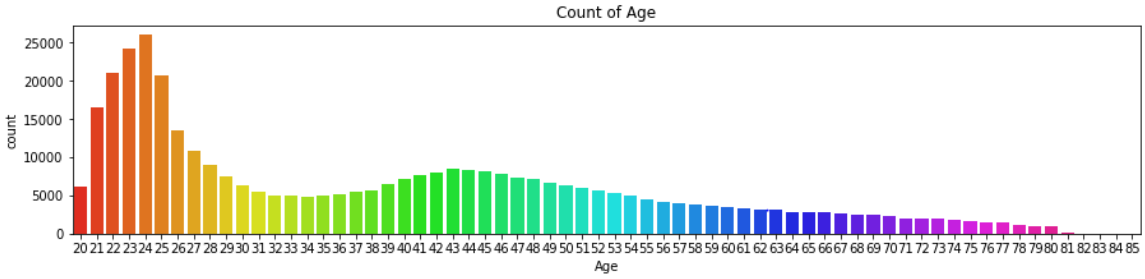
It can be inferred from the above bar plots that:

- The gender variable in the dataset is almost equally distributed
- Male category is slightly greater than that of female and chances of buying the insurance is also little high.

Age variable

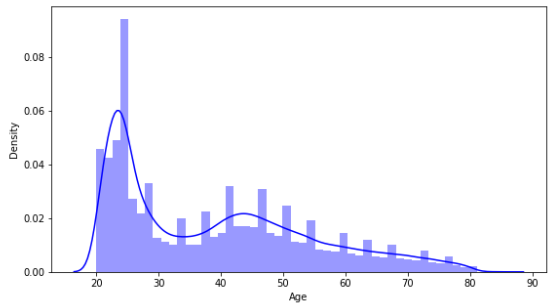
Now let's visualize the numerical variables. Let's look at the distribution of Age first.

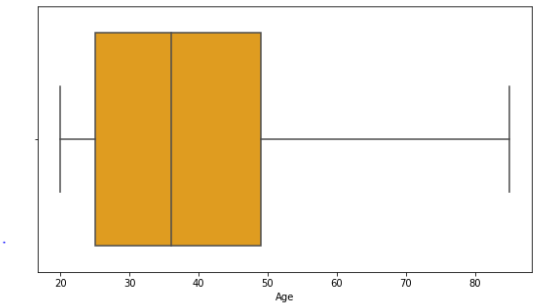
```
plt.figure(figsize = (15,3))
sns.countplot(train['Age'], palette = 'hsv')
plt.title('Count of Age')
plt.show()
```



```
f,ax = plt.subplots(nrows=1,ncols=2,figsize=(20,5))
axx = ax.flatten()
sns.distplot(train['Age'] ,ax = axx[0],color='Blue')
sns.boxplot(train['Age'],ax = axx[1],color='Orange')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0a370d1d0>
```





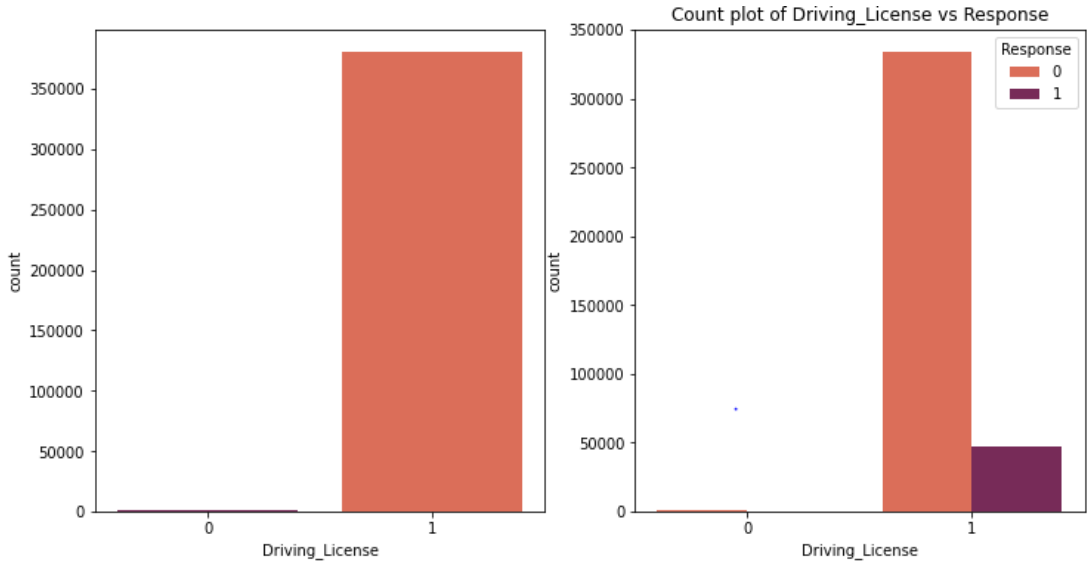
Following inferences can be made from the above plots:

- Count of the indivisuals with age 24 are greater in the dataset
- Variable Age looks like right skewed
- From the boxplot we observe that here is not serious outliers in the data

Driving License

```
print("Percentage of  Driving_License feature\n ")
print(train['Driving_License'].value_counts()/len(train)*100)
f,ax = plt.subplots(nrows=1,ncols=2,figsize=(12,6))
axx = ax.flatten()
plt.title("Count plot of Driving_License vs Response")
sns.countplot(train['Driving_License'],ax = axx[0],palette = 'rocket')
sns.countplot('Driving_License', hue = 'Response',ax =axx[1],data = train,palette="rocket_r"
)
```

```
Percentage of  Driving_License feature

1    99.786938
0     0.213062
Name: Driving_License, dtype: float64
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0acef0610>
```



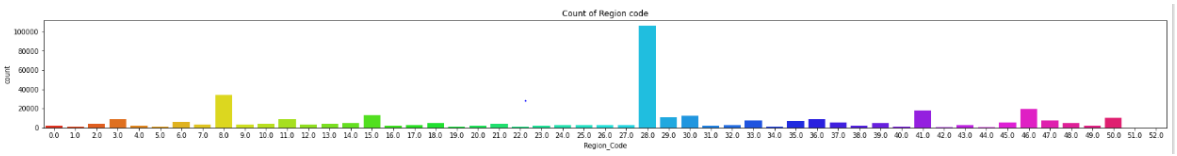
Following inferences can be made from the above bar plots:

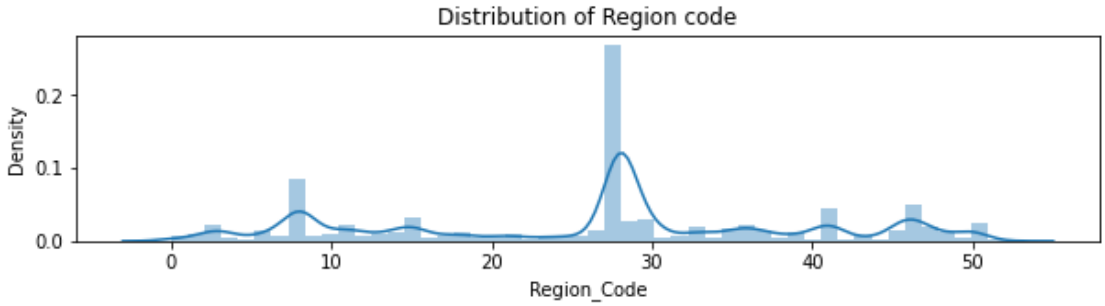
- Customers who have the DL are 99%
- Customers who are interested in Vehicle Insurance almost all have driving license

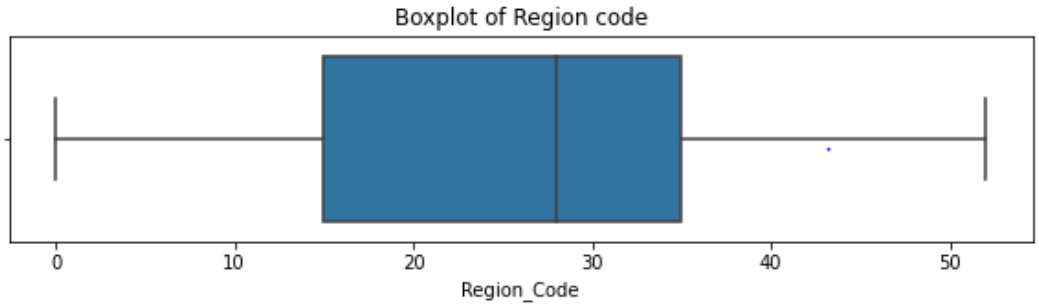
Region Code

Now lets check the bar plot,distribution plot and box plot for the region code variable.

```
plt.figure(figsize = (30,10))
plt.subplot(3,1,1)
sns.countplot(train['Region_Code'], palette = 'hsv')
plt.title('Count of Region code')
plt.figure(figsize = (10,7))
plt.subplot(3,1,2)
sns.distplot(train['Region_Code'])
plt.title('Distribution of Region code')
plt.figure(figsize = (10,7))
plt.subplot(3,1,3)
sns.boxplot(train['Region_Code'])
plt.title('Boxplot of Region code')
plt.show()
```







Following inferences can be made from the above plots:

- The indivisuals with region code 28 the highest as compared to the other ones
- From the box plot it looks like there is no outliers in the data
- Further we can analyze which region has highest intrested customers

Previously Insured

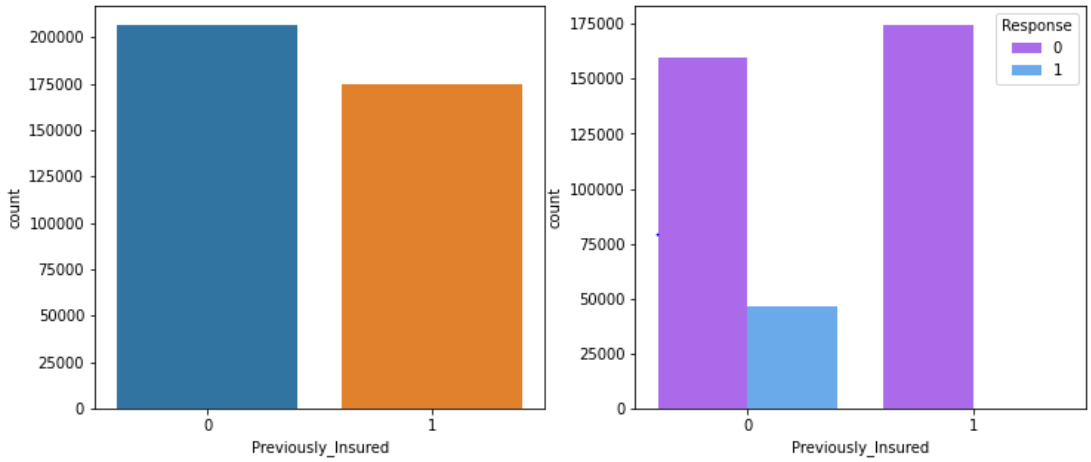
After looking at every variable individually in univariate analysis, we will now explore them again with respect to the target variable.

First of all we will find the relation between target variable and categorical independent variables. Let us look at the column chart now which will give us the previously insured customer is interested or not .

```
print("Percentage ofPreviously_Insured feature\n ")
print(train['Previously_Insured'].value_counts()/len(train)*100)
f,ax = plt.subplots(nrows=1,ncols=2,figsize=(12,5))
axx = ax.flatten()
sns.countplot(train['Previously_Insured'],ax = axx[0])
```

sns.countplot('Previously_Insured', hue = 'Response',ax =axx[1],data = train,palette="cool_r ")

```
Percentage ofPreviously_Insured feature

0    54.178988
1    45.821012
Name: Previously_Insured, dtype: float64
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0a33febd0>
```



- The variable previously insured almost has equal count
- Customer who are not previously insured are likely to be interested

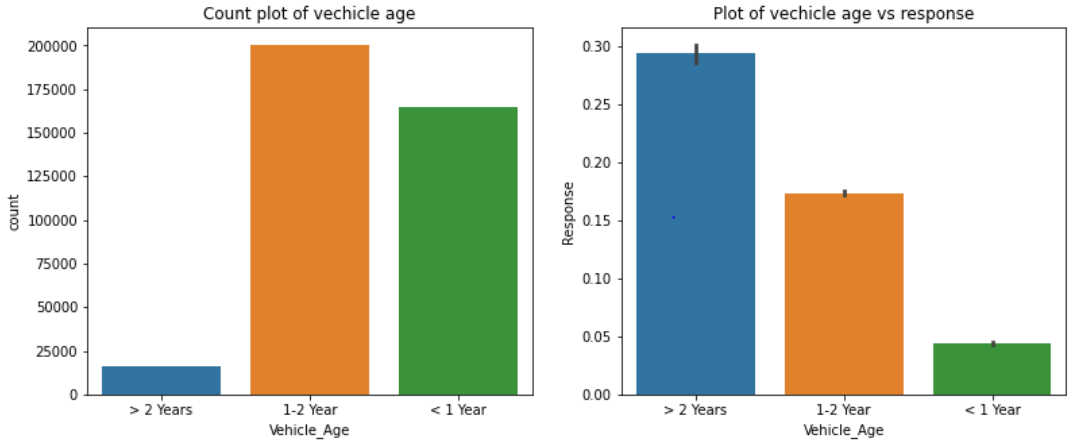
Vehicle Age

Now lets check the relationship between the vehicle age and response.

Lets us look at bar chart of vehicle age with respect to response count.

```
print("Percentage of vechicle age feature with respect to mean responce\n ")
print(train.groupby('Vehicle_Age')['Response'].mean())
plt.figure(figsize = (13,5))
plt.subplot(1,2,1)
sns.countplot(train['Vehicle_Age'])
plt.title("Count plot of vechicle age")
plt.subplot(1,2,2)
plt.title("Plot of vechicle age vs response")
sns.barplot(x='Vehicle_Age',y='Response',data=train)
```

```
Percentage of vechicle age feature with respect to mean responce

Vehicle_Age
1-2 Year     0.173755
< 1 Year     0.043705
> 2 Years    0.293746
Name: Response, dtype: float64
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0a3370210>
```



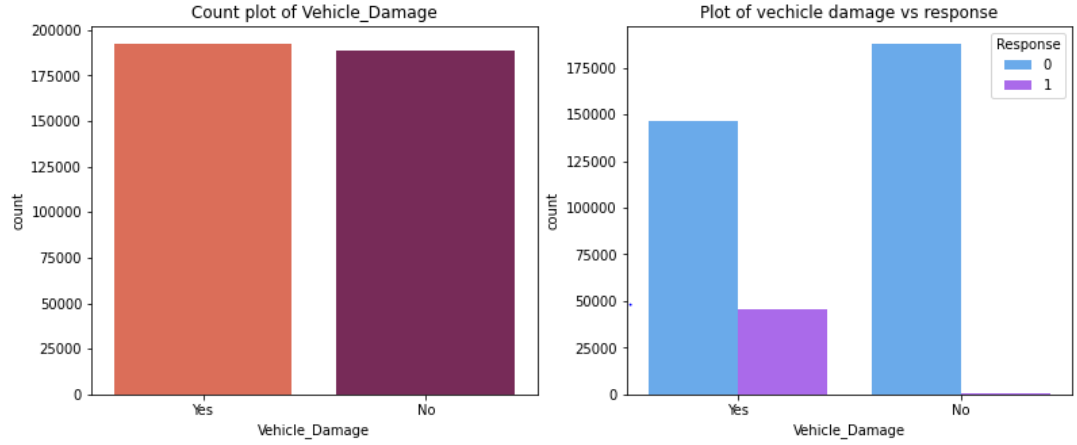
From the above figure it is clear that the more the age of vehicle the better making the vehicle insurance cheaper

Vehicle damage

```
print("Percentage of vechicle damage feature\n ")
print(train['Vehicle_Damage'].value_counts()/len(train)*100)
plt.figure(figsize = (13,5))
plt.subplot(1,2,1)
sns.countplot(train['Vehicle_Damage'],palette="rocket_r")
plt.title("Count plot of Vehicle_Damage")
plt.subplot(1,2,2)
plt.title("Plot of vechicle damage vs response")
sns.countplot('Vehicle_Damage', hue = 'Response',data = train,palette="cool")
```

- Customers with vehicle damage(Yes and NO) are equally distributed with (50.48% , 49.51%)
- Customers with vehicle damage are more interested in Vehicle Insurance

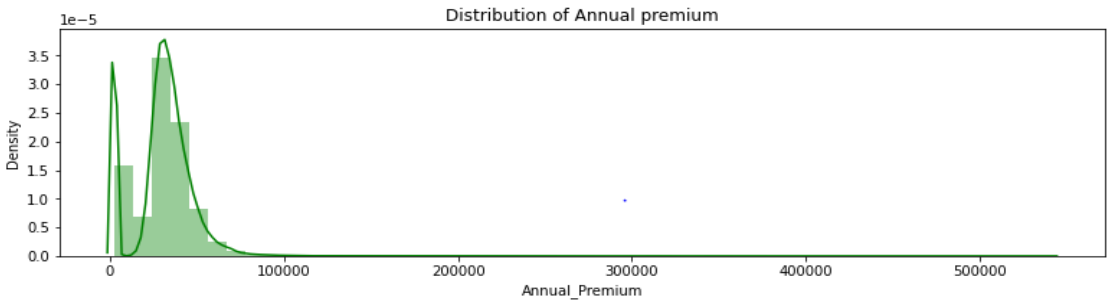
```
Percentage of vechicle damage feature

Yes    50.487656
No     49.512344
Name: Vehicle_Damage, dtype: float64
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0a3280f10>
```

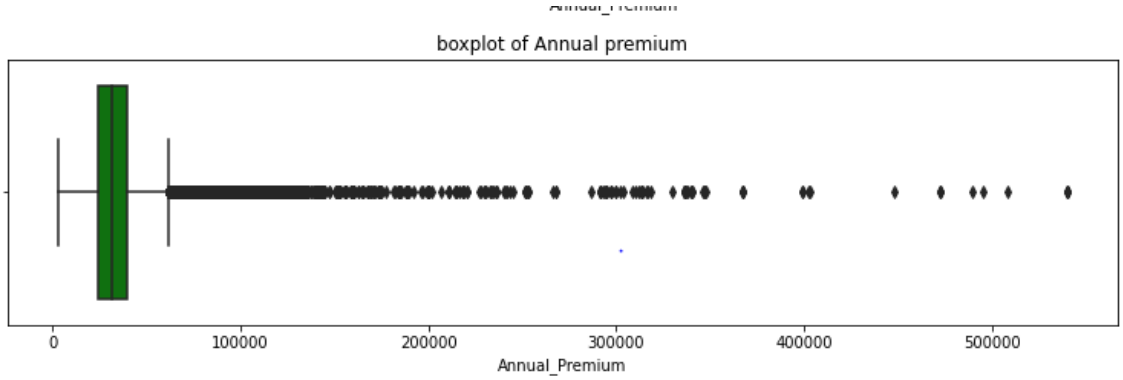


Annual_Premium

Now lets check the distribution plot for the numerical variable Annual_Premium and check the box plot also.

```
plt.figure(figsize=(13,7))
plt.subplot(2,1,1)
sns.distplot(train['Annual_Premium'], color='green')
plt.title("Distribution of Annual premium")
plt.show()
#print("-----------------------------------")
plt.figure(figsize=(13,7))
plt.subplot(2,1,2)
sns.boxplot(train['Annual_Premium'],color='green')
plt.title("boxplot of Annual premium")
plt.show()
```



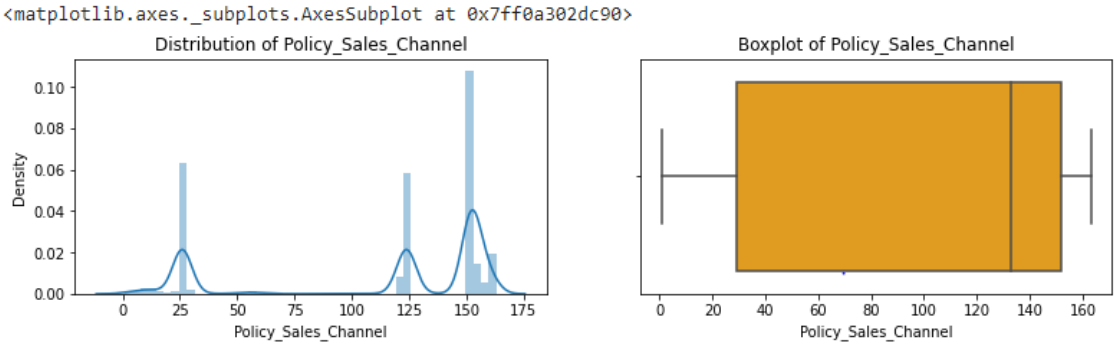


From the distribution plot we can infer that the annual premium variable is right skewed

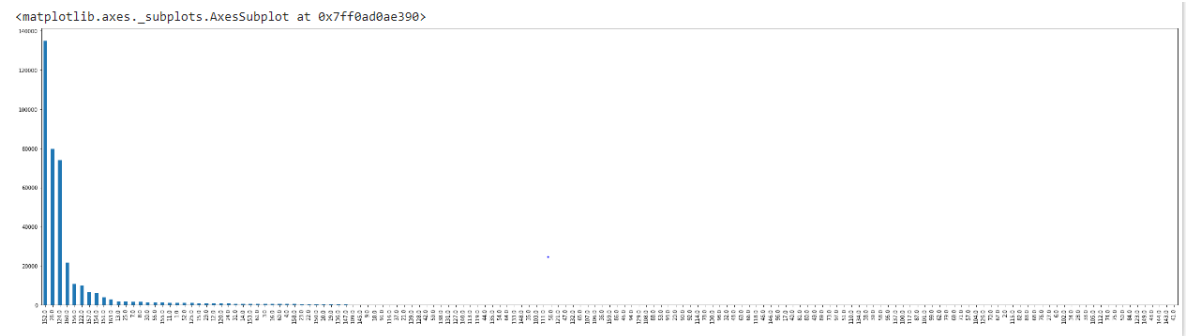
- From the boxplot we can observe lot of outliers in the variable
- We can see that customers generally opt for Premium which is not too high.

Policy_Sales_Channel

```
plt.figure(figsize = (20,3))
plt.subplot(1,3,1)
plt.title("Distribution of Policy_Sales_Channel")
sns.distplot(train['Policy_Sales_Channel'])
plt.subplot(1,3,2)
plt.title("Boxplot of Policy_Sales_Channel")
sns.boxplot(train['Policy_Sales_Channel'],color='Orange')
```



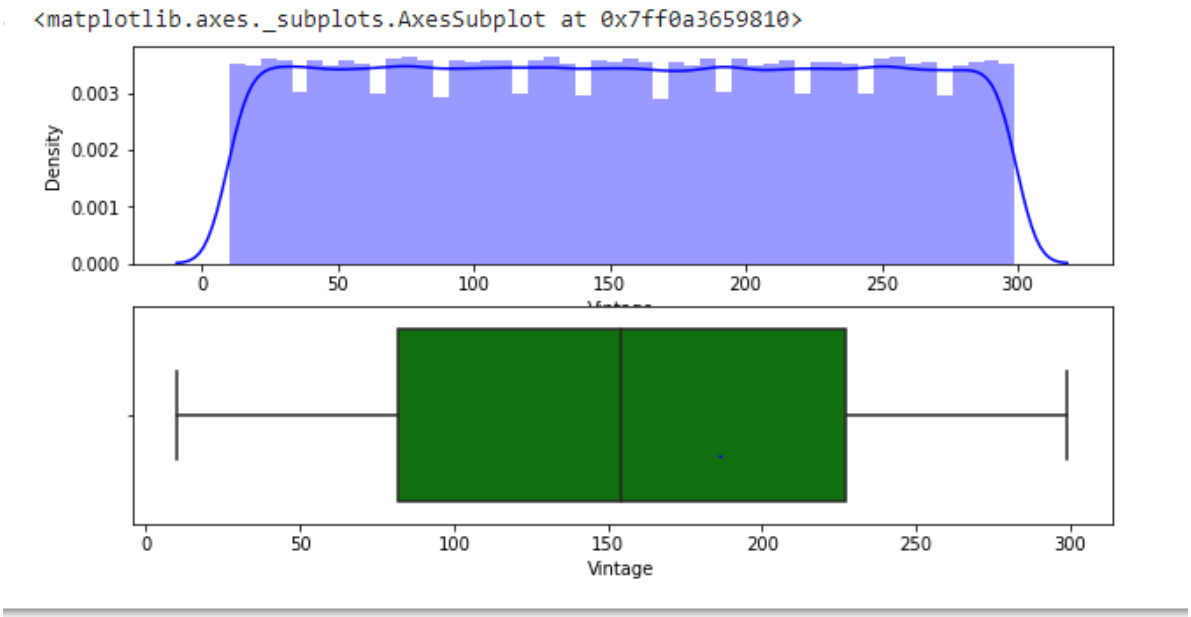

```
plt.figure(figsize=(40,10))
train['Policy_Sales_Channel'].value_counts().plot.bar()
```



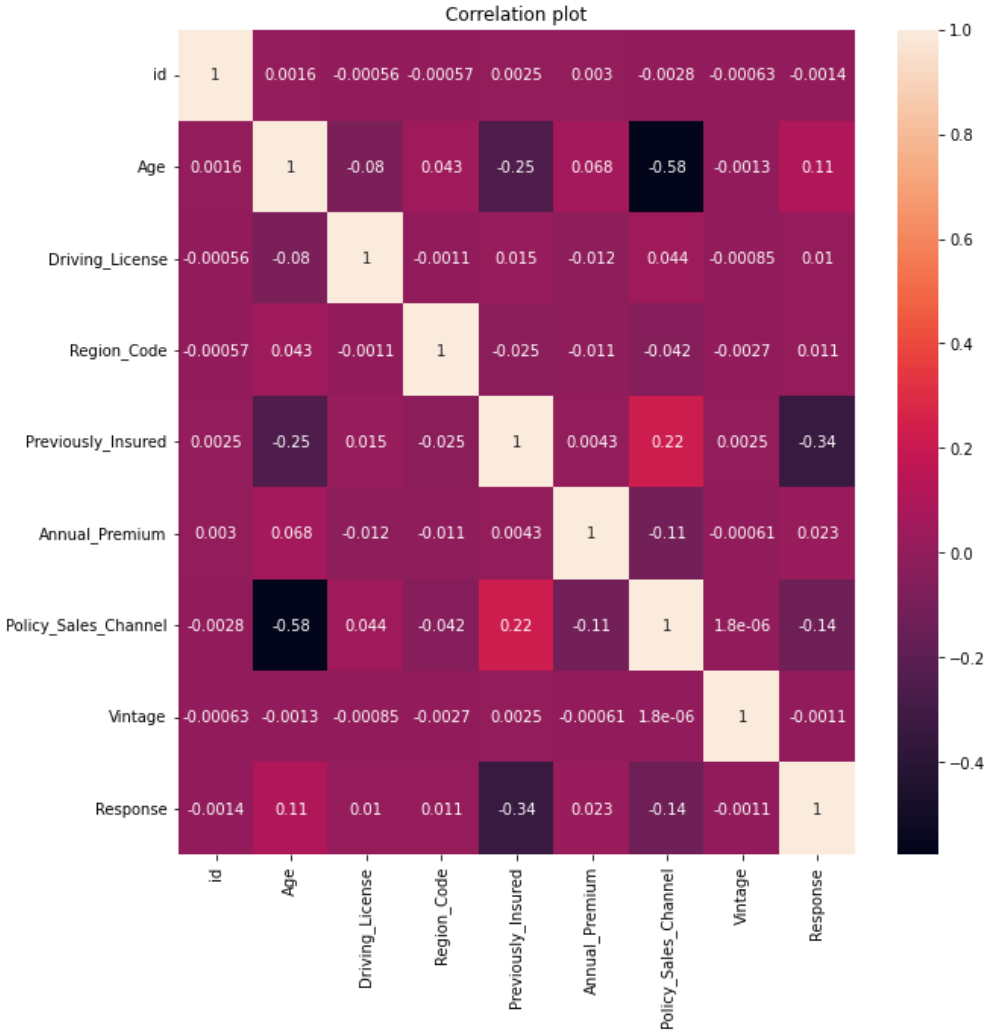
- Policy_Sales_Channel no. 152 have highest number of customers.
- Policy_Sales_Channel no. [152,26,124,160,156,122,157,154,151,163] have most of the customers.

Vintage

```
f,ax = plt.subplots(nrows=2,ncols=1,figsize=(10,5))
axx = ax.flatten()
sns.distplot(train['Vintage'],ax=axx[0], color='Blue')
sns.boxplot(train['Vintage'],ax=axx[1],color='green')
```




```
plt.figure(figsize=(10,10))
plt.title("Correlation plot")
sns.heatmap(train.corr(), annot=True,)
```



From the correlation plot we observe that policy sales channel has slightly greater correlation with Age variable, this may be the indication of multicollinearity. We can further use VIF to check this.

8.5 Improving Model Performance

After exploring all the variables in our data, we can now impute the missing values and treat the outliers because missing data and outliers can have adverse effect on the model performance.

Missing value imputation

Let's list out feature-wise count of missing values.

```
train.isnull().sum()
```

```
id                    0
Gender                0
Age                   0
Driving_License       0
Region_Code           0
Previously_Insured    0
Vehicle_Age           0
Vehicle_Damage        0
Annual_Premium        0
Policy_Sales_Channel  0
Vintage               0
Response              0
dtype: int64
```

There are missing values in Annual_premium, Policy_Sales_Channel, Vintage, Response features.

We will treat the missing values in all the features one by one.

We can consider these methods to fill the missing values:

- **For numerical variables:** imputation using mean or median
- **For categorical variables:** imputation using mode

There are very less missing values in Annual_premium, Policy_Sales_Channel, Vintage, Response features so we can fill them using the mode of the features.

```
train['Annual_Premium'].fillna(train['Annual_Premium'].mode()[0],inplace=True)
train['Policy_Sales_Channel'].fillna(train['Policy_Sales_Channel'].mode()[0],inplace=True)
train['Vintage'].fillna(train['Vintage'].mode()[0],inplace=True)
train['Response'].fillna(train['Response'].mode()[0],inplace=True)
```

Now lets check whether all the missing values are filled in the dataset.

```
train.isnull().sum()
```

```
id                    0
Gender                0
Age                   0
Driving_License       0
Region_Code           0
Previously_Insured    0
Vehicle_Age           0
Vehicle_Damage        0
Annual_Premium        0
Policy_Sales_Channel  0
Vintage               0
Response              0
dtype: int64
```

Lets check the missing values in test dataset also.

```
test.isnull().sum()
```

```
id                    0
Gender                0
Age                   0
Driving_License       0
Region_Code           0
Previously_Insured    0
Vehicle_Age           0
Vehicle_Damage        0
Annual_Premium        0
Policy_Sales_Channel  0
Vintage               0
dtype: int64
```

There is no missing values in the test dataset.

Label Encoding

Now we will do label encoding for the categorical variables. label encoding turns categorical variables into a series of 0 and 1,2 making them lot easier to quantify and compare. Let us understand the process of label encoding first:

• Consider the “Gender” variable. It has two classes, Male and Female.

• As logistic regression takes only the numerical values as input, we have to change male and female into numerical value.

• Once we apply label encoding to this variable, it will convert the “Gender” variable values into two values(0 and 1), i.e. Male and Female.

• Male will have a value of 1 and if the gender is Female value is 0.

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
train['Vehicle_Age'] = le.fit_transform(train['Vehicle_Age'])
train['Gender'] = le.fit_transform(train['Gender'])
train['Vehicle_Damage'] = le.fit_transform(train['Vehicle_Damage'])
test['Gender'] = le.fit_transform(test['Gender'])
test['Vehicle_Age'] = le.fit_transform(test['Vehicle_Age'])
test['Vehicle_Damage'] = le.fit_transform(test['Vehicle_Damage'])
```

```
train.head()
```

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	1	44	1	28.0	0	2	1	40454.0	26.0	217	1
1	2	1	76	1	3.0	0	0	0	33536.0	26.0	183	0
2	3	1	47	1	28.0	0	2	1	38294.0	26.0	27	1
3	4	1	21	1	11.0	1	1	0	28619.0	152.0	203	0
4	5	0	29	1	41.0	1	1	0	27496.0	152.0	39	0

```
test.head()
```

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	381110	1	25	1	11.0	1	1	0	35786.0	152.0	53
1	381111	1	40	1	28.0	0	0	1	33762.0	7.0	111
2	381112	1	47	1	28.0	0	0	1	40050.0	124.0	199
3	381113	1	24	1	27.0	1	1	1	37356.0	152.0	187
4	381114	1	27	1	28.0	1	1	0	59097.0	152.0	297

Outlier Treatment

As we saw earlier in univariate analysis, Annual_Premium contains outliers so we have to treat them as the presence of outliers affects the distribution of the data. Let's examine what can happen to a data set with outliers. For the sample data set:

1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4

We find the following: mean, median, mode, and standard deviation

Mean = 2.58

Median = 2.5

Mode = 2

Standard Deviation = 1.08

If we add an outlier to the data set:

1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 400

The new values of our statistics are:

Mean = 35.38

Median = 2.5

Mode = 2

Standard Deviation = 114.74

It can be seen that having outliers often has a significant effect on the mean and standard deviation and hence affecting the distribution. We must take steps to remove outliers from our data sets.

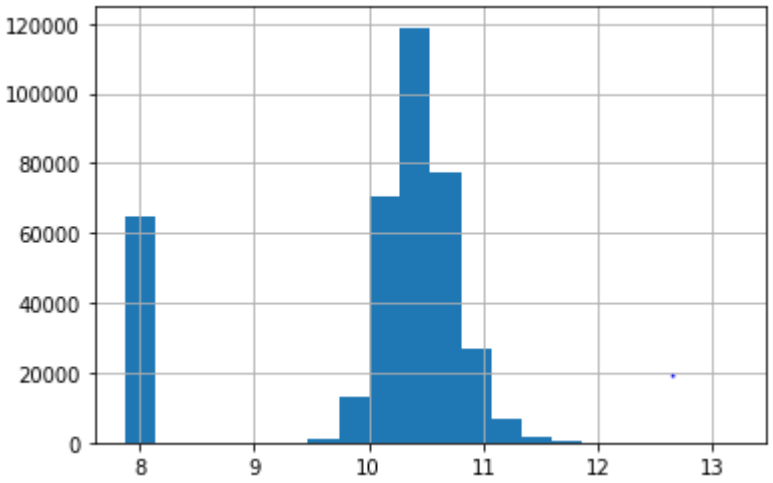
Due to these outliers bulk of the data in the Annual_premium is at the left and the right tail is longer. This is called **right skewness**.

One way to remove the skewness is by doing the log transformation. As we take the log transformation, it does not affect the smaller values much, but reduces the larger values.

So, we get a distribution similar to normal distribution.

Let’s visualize the effect of log transformation. We will do the similar changes to the test file simultaneously.

```
train['Annual_Premium_log']=np.log(train['Annual_Premium'])
train['Annual_Premium_log'].hist(bins=20)
test['Annual_Premium_log']=np.log(test['Annual_Premium'])
```



Now the distribution looks much closer to normal and effect of extreme values has been significantly subsided.

Let’s build a logistic regression model and make predictions for the test dataset.

Feature Engineering

In this section we will try the Discretisation of continous column using Decision Tree. Discretisation is a process of converting the continous variable into discrete variable with the help of bins.

In this section we will convert the continous variable('Age','Annual_Premium') into discrete variable .

Steps to Follow:

Spit the data into train_test and fit the Decision Tree(depth=1,2,3,4) using the X=continous variable ; y=Target

The continous variable are then replaced by the predicted_probability.

```
from sklearn.model_selection import train_test_split
X=train[['Age','Annual_Premium','Response']]
y=train['Response']

x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=2)
from sklearn.tree import DecisionTreeClassifier
dtree=DecisionTreeClassifier(max_depth=2)
dtree.fit(x_train.Age.to_frame(),y_train)
x_train['Age_tree']=dtree.predict_proba(x_train.Age.to_frame())[:,1]
x_train.head()
```

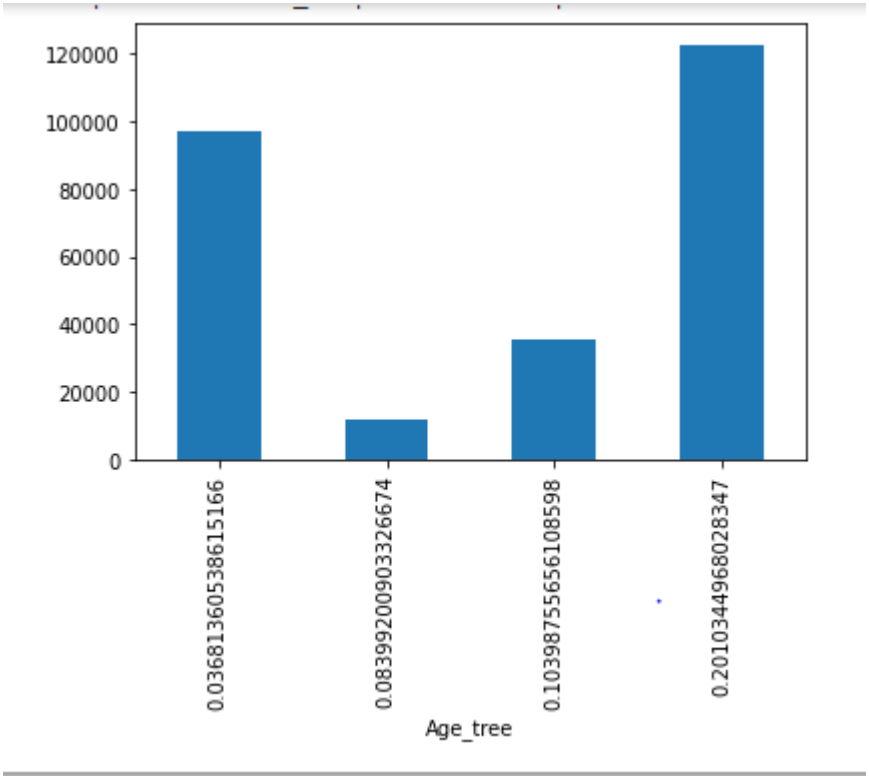
	Age	Annual_Premium	Response	Age_tree
275905	33	41453.0	0	0.201034
195653	22	20089.0	0	0.036814
293289	52	42310.0	0	0.201034
6065	45	2630.0	0	0.201034
18646	25	35506.0	0	0.036814

```
x_train['Age_tree'].unique()
```

```
array([0.2010345 , 0.03681361, 0.08399201, 0.10398756])
```

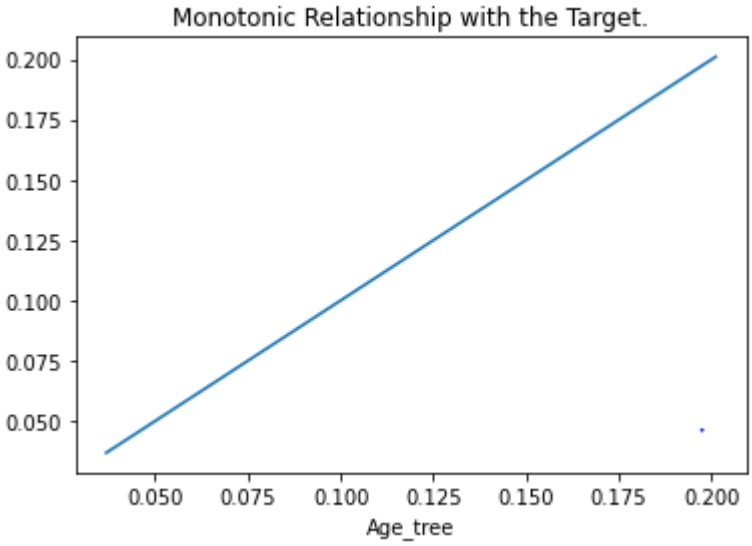
Now lets visualize the age_tree variable with respect to the response count.

```
x_train.groupby('Age_tree')['Response'].count().plot.bar()
```



Check the monotonic relationship with the target variable.

```
x_train.groupby('Age_tree')['Response'].mean().plot()
plt.title('Monotonic Relationship with the Target.')
plt.show()
```



- We can see that the new column: Age_tree is a good predictor of the Target.

- We can use the Predicted_probability to create the Bins.

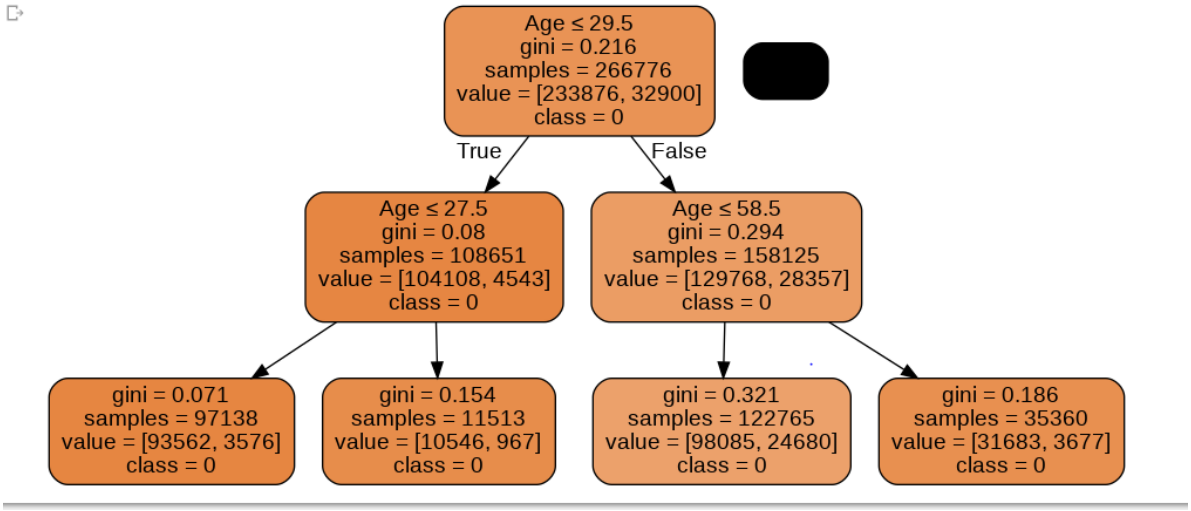
To find the bins we have to find out the min and max age range between the probability values of age ie. Age_tree.

```
age_limit=pd.DataFrame({'min_age':x_train.groupby('Age_tree')['Age'].min(),'max_age':x_train.groupby('Age_tree')['Age'].max()})
age_limit
```

Age_tree	min_age	max_age
0.036814	20	27
0.083992	28	29
0.103988	59	85
0.201034	30	58

To visualize the decision tree by graphviz we import the export_graphviz function from the tree class. It will split the age column with respect to response variable.

```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data,filled=True,
                rounded=True,
               special_characters=True,feature_names=['Age'],class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```



Hyper-parameter Tuning to check the best depth.

```
from sklearn.model_selection import cross_val_score
score_ls=[]
score_std_ls=[]
for tree_depth in [1,2,3,4]:
  tree_model=DecisionTreeClassifier(max_depth=tree_depth)
  score=cross_val_score(tree_model,x_train.Age.to_frame(),y_train,cv=3,scoring='roc_auc')
  score_ls.append(np.mean(score))
  score_std_ls.append(np.std(score))
temp=pd.concat([pd.Series([1,2,3,4]),pd.Series(score_ls),pd.Series(score_std_ls)],axis=1)
temp.columns=['depth', 'roc_auc_mean', 'roc_auc_std']
print(temp)
```

	depth	roc_auc_mean	roc_auc_std
0	1	0.653528	0.002830
1	2	0.683938	0.002064
2	3	0.689069	0.002108
3	4	0.695761	0.002520

We can see that with depth=2 is a better choice to avoid overfitting.

We replaced the continous colum with the bins.

```
train.loc[(train['Age']>=20) & (train['Age']<27),'Age_label']='Teenangers'
```

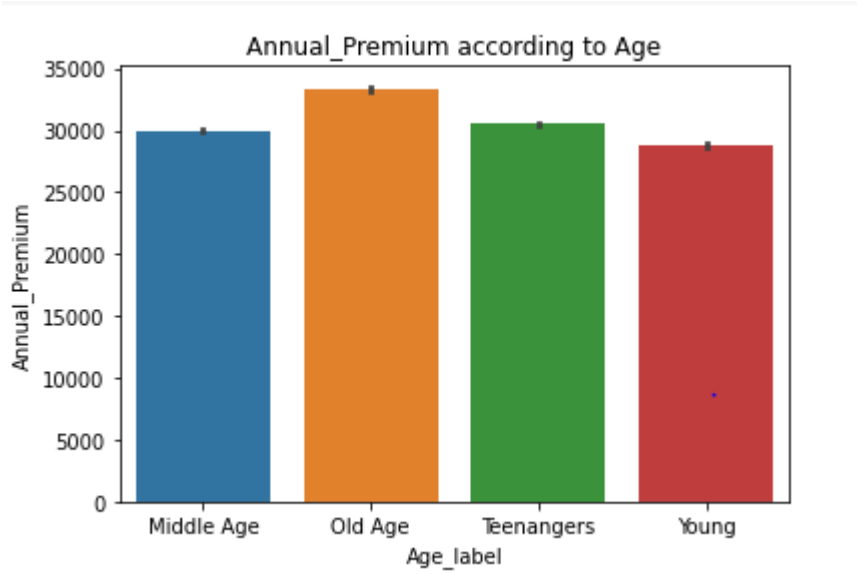
```
train.loc[(train['Age']>=27) & (train['Age']<29),'Age_label']='Young'
train.loc[(train['Age']>=29) & (train['Age']<58),'Age_label']='Middle Age'
train.loc[(train['Age']>=58) & (train['Age']<=85),'Age_label']='Old Age'

train.loc[(train['Age']>=20) & (train['Age']<27),'Age']=0
train.loc[(train['Age']>=27) & (train['Age']<29),'Age']=1
train.loc[(train['Age']>=29) & (train['Age']<58),'Age']=2
train.loc[(train['Age']>=58) & (train['Age']<=85),'Age']=3
```

```
train.head()
```

id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response	Annual_Premium_log	Age_label
1	1	2	1	28.0	0	2	1	40454.0	26.0	217	1	10.607921	Middle Age
2	1	3	1	3.0	0	0	0	33536.0	26.0	183	0	10.420375	Old Age
3	1	2	1	28.0	0	2	1	38294.0	26.0	27	1	10.553049	Middle Age
4	1	0	1	11.0	1	1	0	28619.0	152.0	203	0	10.261826	Teenangers
5	0	2	1	41.0	1	1	0	27496.0	152.0	39	0	10.221796	Middle Age

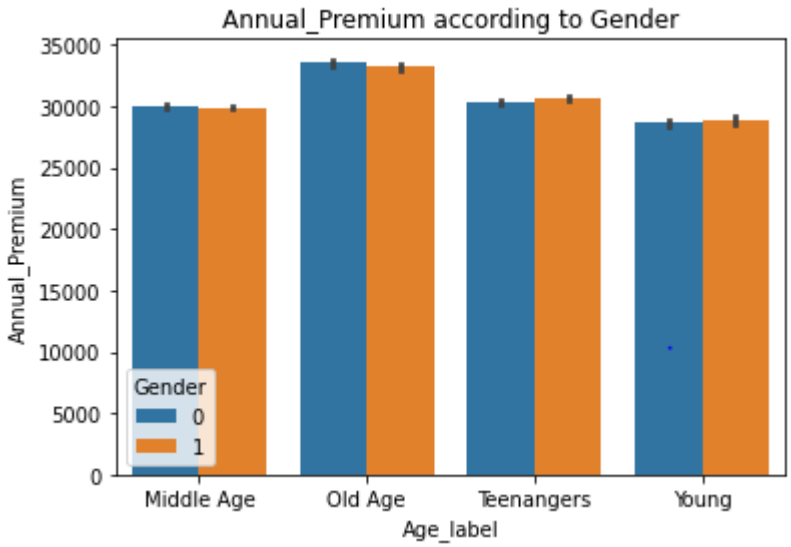
Now lets check the Annual_Premium according to Age_label by using the bar plot.

```
sns.barplot(train['Age_label'],train['Annual_Premium'])
plt.title('Annual_Premium according to Age')
plt.show()
```



- From the above Plot we can see that Annual_Premium is directly dependent on the Age of the Customer. The higher the age higher the Annual_Premium.

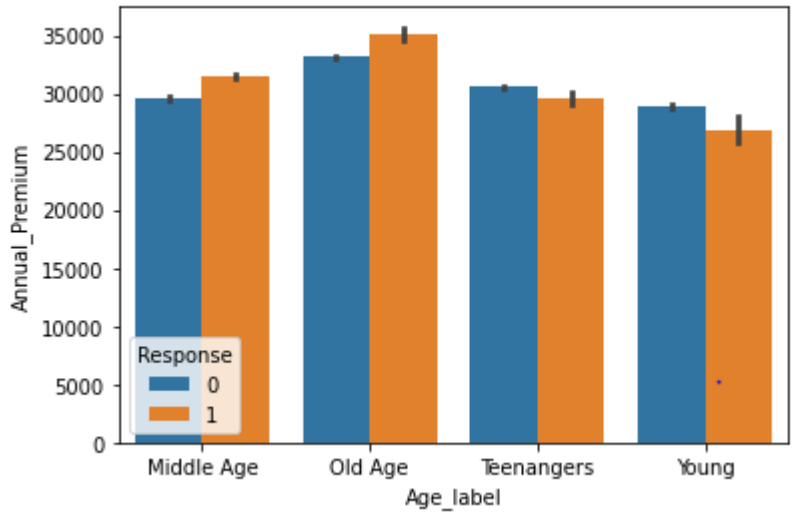
```
sns.barplot(train['Age_label'],train['Annual_Premium'],hue=train['Gender'])
plt.title('Annual_Premium according to Gender')
plt.show()
```



- There is not much to see from the above plot.

```
sns.barplot(train['Age_label'],train['Annual_Premium'],hue=train['Response'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0a3320750>
```



- We can see that:Age group 20-27 usually do not take Insurance as they are just starting with their lives and may not have money to pay the Premium.
- Age group from 29-85 opt for Insurance as they have driving License and resources to pay the Premium.

We shall do the same for Annual_Premium column as well.

```
tree=DecisionTreeClassifier(max_depth=2)
tree.fit(x_train.Annual_Premium.to_frame(),y_train)
```

```
x_train['Annual_premium_tree']=tree.predict_proba(x_train.Annual_Premium.to_frame())[:
,1]
x_train['Annual_premium_tree'].unique()
```

```
array([0.14019373, 0.10877935, 0.13156156, 0.15365861])
```

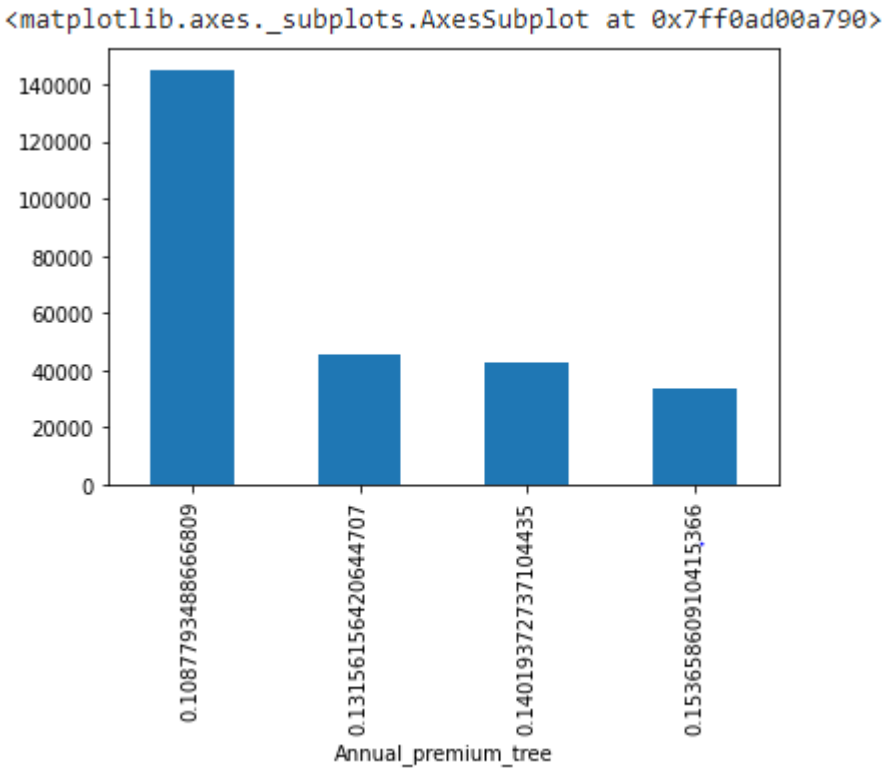
x_train

	Age	Annual_Premium	Response	Age_tree	Annual_premium_tree
275905	33	41453.0	0	0.201034	0.140194
195653	22	20089.0	0	0.036814	0.108779
293289	52	42310.0	0	0.201034	0.140194
6065	45	2630.0	0	0.201034	0.131562
18646	25	35506.0	0	0.036814	0.108779
...	...	...	...	...	...
84434	43	34569.0	0	0.201034	0.108779
95816	54	44982.0	0	0.201034	0.140194
203245	59	2630.0	0	0.103988	0.131562
100879	58	2630.0	0	0.201034	0.131562
351400	46	2630.0	0	0.201034	0.131562

266776 rows × 5 columns

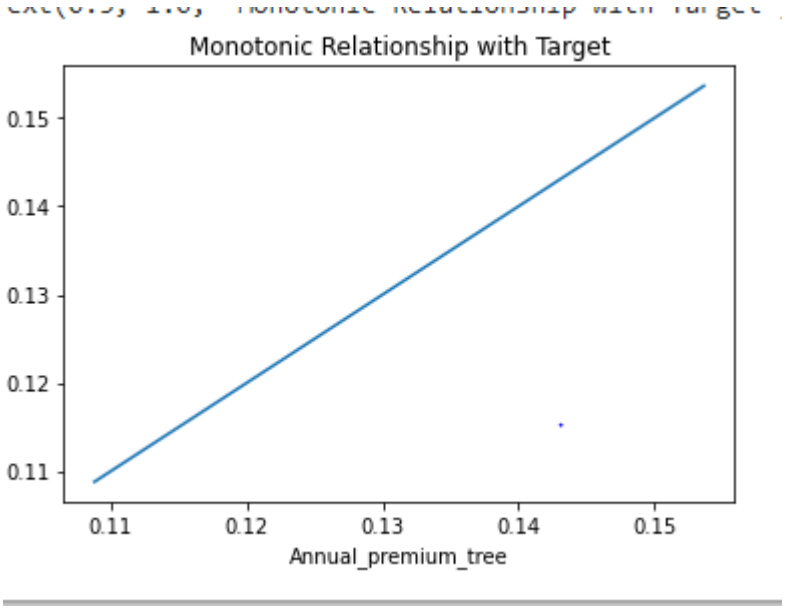
Check the Annual_Premium_tree with respect to Response count using the bar plot.

```
x_train.groupby('Annual_premium_tree')['Response'].count().plot.bar()
```



Now check the relationship between the Annual_premium_tree with the mean of response variable.

```
x_train.groupby('Annual_premium_tree')['Response'].mean().plot()
plt.title('Monotonic Relationship with Target')
```



- We can see that the new column: Annual_premium_tree is a good predictor of the Target.
- We can use the Predicted_probability to create the Bins.

To find the bins values we have to find the min and max values lies between the Annual Premium probability values.

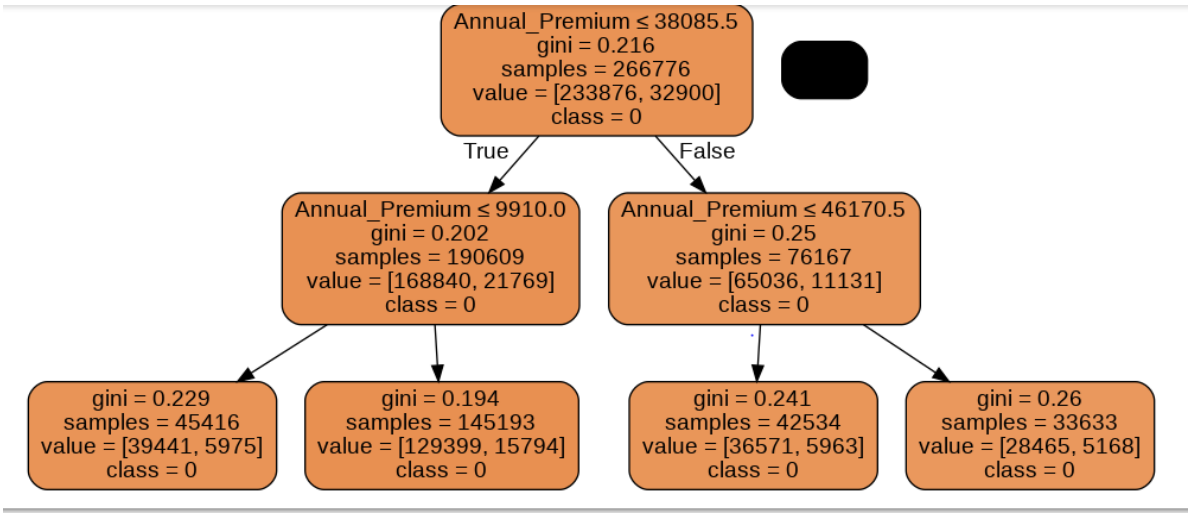
```
Annual_premium_tree_limit=pd.concat([x_train.groupby('Annual_premium_tree')['Annual_Premium'].min(),x_train.groupby('Annual_premium_tree')['Annual_Premium'].max()],axis=1)
Annual_premium_tree_limit
```

Annual_premium_tree	Annual_Premium	Annual_Premium
0.108779	10004.0	38085.0
0.131562	2630.0	9816.0
0.140194	38086.0	46170.0
0.153659	46171.0	540165.0

To check the graph wise presentation of decision tree we plot the graph of decision tree using export_graphviz function.it will split the Annual_Premium column with respect to response.

```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data=StringIO()
export_graphviz(tree,out_file=dot_data,feature_names=['Annual_Premium'],class_names=['0','1'],rounded=True,filled=True,special_characters=True)
```

```
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



Now make the changes in the Annual_Premium column and Annual_Premium_label as per the bins value we get from the above min max value chart.

```
train.loc[(train['Annual_Premium']>=2630.0) &(train['Annual_Premium']<9816.0),'Annual_Premium_label']='Low_Premium'
train.loc[(train['Annual_Premium']>=9816.0) & (train['Annual_Premium']<38085.0),'Annual_Premium_label']='Average_Premium'
train.loc[(train['Annual_Premium']>=38085.0) & (train['Annual_Premium']<46170.0),'Annual_Premium_label']='Above_avg_premium'
train.loc[(train['Annual_Premium']>=46170.0) & (train['Annual_Premium']<=540165.0),'Annual_Premium_label']='High_premium'
```

```
train.loc[(train['Annual_Premium']>=2630.0) &(train['Annual_Premium']<9816.0),'Annual_Premium']=0
train.loc[(train['Annual_Premium']>=9816.0) & (train['Annual_Premium']<38085.0),'Annual_Premium']=1
train.loc[(train['Annual_Premium']>=38085.0) & (train['Annual_Premium']<46170.0),'Annual_Premium']=2
```

```
train.loc[(train['Annual_Premium']>=46170.0) & (train['Annual_Premium']<=540165.0),'Annual_Premium']=3
```

lets find the best depth value for the decision tree .

```
score_ls=[]
score_std_ls=[]
for tree_depth in [1,2,3,4]:
  tree_model=DecisionTreeClassifier(max_depth=tree_depth)
  score=cross_val_score(tree_model,x_train.Annual_Premium.to_frame(),y_train,cv=3,scoring='roc_auc')
  score_ls.append(np.mean(score))
  score_std_ls.append(np.std(score))
temp=pd.concat([pd.Series([1,2,3,4]),pd.Series(score_ls),pd.Series(score_std_ls)],axis=1)
temp.columns = ['depth', 'roc_auc_mean', 'roc_auc_std']
print(temp)
```

	depth	roc_auc_mean	roc_auc_std
0	1	0.529599	0.000910
1	2	0.540527	0.000477
2	3	0.547035	0.001000
3	4	0.547878	0.001473

We can see that there is not much difference in roc_auc score we we will use depth=2.

```
train.head()
```

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	1	2	1	28.0	0	2	1	2.0	26.0	217	1
1	2	1	3	1	3.0	0	0	0	1.0	26.0	183	0
2	3	1	2	1	28.0	0	2	1	2.0	26.0	27	1
3	4	1	0	1	11.0	1	1	0	1.0	152.0	203	0
4	5	0	2	1	41.0	1	1	0	1.0	152.0	39	0

Annual_Premium_log	Age_label	Annual_Premium_label
10.607921	Middle Age	Above_avg_premium
10.420375	Old Age	Average_Premium
10.553049	Middle Age	Above_avg_premium
10.261826	Teenangers	Average_Premium
10.221796	Middle Age	Average_Premium

test.head()

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	381110	1	25	1	11.0	1	1	0	35786.0	152.0	53
1	381111	1	40	1	28.0	0	0	1	33762.0	7.0	111
2	381112	1	47	1	28.0	0	0	1	40050.0	124.0	199
3	381113	1	24	1	27.0	1	1	1	37356.0	152.0	187
4	381114	1	27	1	28.0	1	1	0	59097.0	152.0	297

Annual_Premium_log
10.485312
10.427091
10.597884
10.528249
10.986935

Lets drop the columns which we created for the visualization purpose.

```
train.drop(['id','Age_label','Annual_Premium_label','Annual_Premium'],axis=1,inplace=True)
test.drop(['id','Annual_Premium'],axis=1,inplace=True)
```

train.head()

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Policy_Sales_Channel	Vintage	Response	Annual_Premium_log
0	1	2	1	28.0	0	2	1	26.0	217	1	10.607921
1	1	3	1	3.0	0	0	0	26.0	183	0	10.420375
2	1	2	1	28.0	0	2	1	26.0	27	1	10.553049
3	1	0	1	11.0	1	1	0	152.0	203	0	10.261826
4	0	2	1	41.0	1	1	0	152.0	39	0	10.221796

Evaluation Metrics for Classification Problems

The process of model building is not complete without evaluation of model's performance. Suppose we have the predictions from the model, how can we decide whether the predictions are accurate? We can plot the results and compare them with the actual values, i.e. calculate the distance between the predictions and actual values. Lesser this distance more accurate will be the predictions. Since this is a classification problem, we can evaluate our models using any one of the following evaluation metrics:

•**Accuracy**: Let us understand it using the confusion matrix which is a tabular representation of Actual vs Predicted values.

• True Positive - Targets which are actually true(Y) and we have predicted them true(Y)

• True Negative - Targets which are actually false(N) and we have predicted them false(N)

• False Positive - Targets which are actually false(N) but we have predicted them true(T)

• False Negative - Targets which are actually true(T) but we have predicted them false(N)

Using these values, we can calculate the accuracy of the model. The accuracy is given by:

•**Precision**: It is a measure of correctness achieved in true prediction i.e. of observations labeled as true, how many are actually labeled true.

Precision = TP / (TP + FP)

•**Recall**(**Sensitivity**) - It is a measure of actual observations which are predicted correctly i.e. how many observations of true class are labeled correctly. It is also known as 'Sensitivity'.

Recall = TP / (TP + FN)

•**Specificity** - It is a measure of how many observations of false class are labeled correctly.

Specificity = TN / (TN + FP)

Specificity and Sensitivity plays a crucial role in deriving ROC curve.

• **Receiver Operating Characteristic**(**ROC**) summarizes the model's performance by evaluating the trade offs between true positive rate (sensitivity) and false positive rate(1-specificity).

The area under curve (AUC), referred to as index of accuracy(A) or concordance index, is a perfect performance metric for ROC curve. Higher the area under curve, better the prediction power of the model.

8.6 Model Building:

Let us make our first model to predict the target variable. We will start with Logistic Regression which is used for predicting binary outcome.

• Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.

• Logistic regression is an estimation of Logit function. Logit function is simply a log of odds in favor of the event.

• This function creates a s-shaped curve with the probability estimate, which is very similar to the required step wise function

Splitting data for training and validation

Sklearn requires the target variable in a separate dataset.

So, we will drop our target variable from the train dataset and save it in another dataset.

We will build the following models in this section.

• **Logistic Regression**

• **Decision Tree**

• **Random Forest**

• **XGBoost**

• **KNeighbour Classifier**

We will use scikit-learn (sklearn) for making different models which is an open source library for Python. It is one of the most efficient tool which contains many inbuilt functions that can be used for modeling in Python**.**

Sklearn requires the target variable in a separate dataset. So, we will drop our target variable from the train dataset and save it in another dataset.

```
X=train.drop('Response',axis=1)
y=train['Response']
```

Now we will train the model on training dataset and make predictions for the test dataset. But can we validate these predictions? One way of doing this is we can divide our train dataset into two parts: train and validation. We can train the model on this train part and using that make predictions for the validation part. In this way we can validate our predictions as we have the true predictions for the validation part (which we do not have for the test dataset).

We will use the train_test_split function from sklearn to divide our train dataset. So, first let us import train_test_split.

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.30)
```

The dataset has been divided into training and validation part. Let us import LogisticRegression and accuracy_score from sklearn and fit the logistic regression model.

Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(solver='liblinear',random_state=0)
lr.fit(X_train,y_train)
```

```
LogisticRegression(random_state=0, solver='liblinear')
```

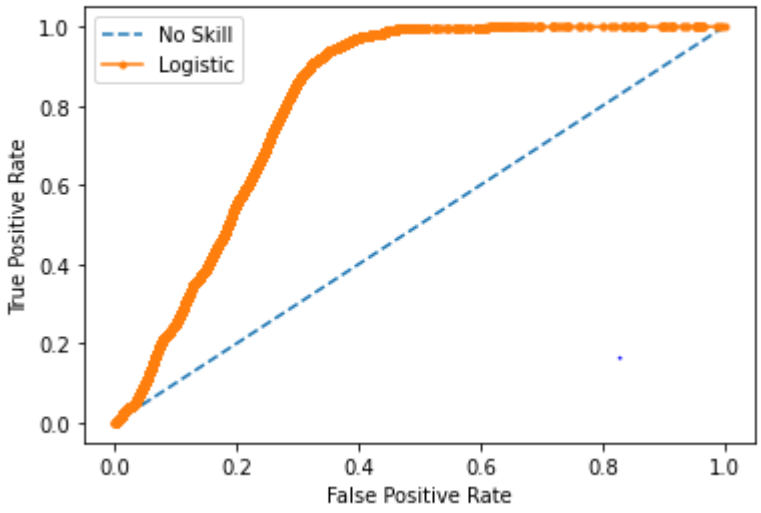
Let's predict the Response for validation set and calculate its accuracy.

ROC CURVE

```
from sklearn.metrics import roc_curve
ns_probs = [0 for _ in range(len(y_test))]
```

```
lr_probs = lr.predict_proba(X_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot

plt.show()
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.813
```



- We can see that with using simplest Classification Algorithm we are able to get ROC-AUC score as 0.814.
- We can see that its is a case of Imbalanced dataset. So we will apply some techniques.

```
y_pred=lr.predict(X_test)
```

Let us calculate how accurate our predictions are by calculating the accuracy.

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

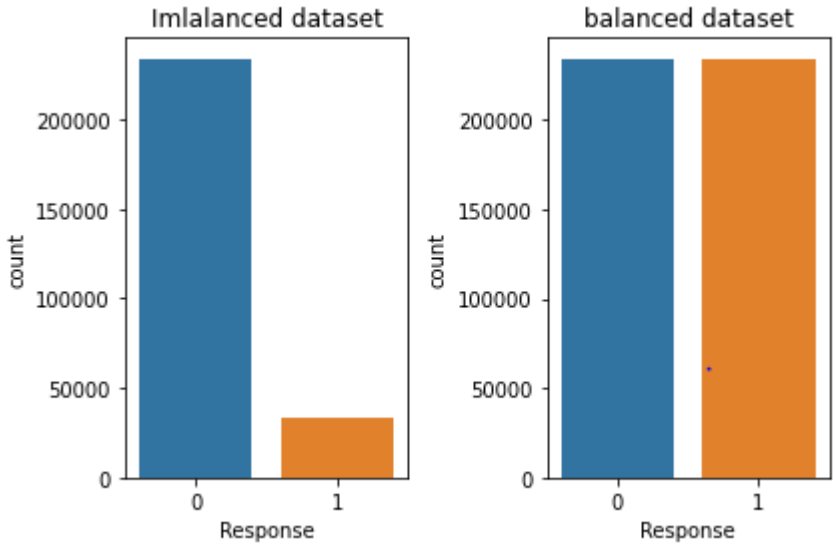
```
0.8774369604576107
```

So our predictions are almost 88% accurate, i.e. we have identified 88% of the Response correctly.

Handling an Unbalanced Dataset

We have earlier seen that the Response variable has an unbalanced ratio in labels. So we will apply some techniques to balance the ratio so that.

```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
X_balanced,Y_balanced=sm.fit_resample(X,y)
x_train_balc,x_test_balc,y_train_balc,y_test_balc=train_test_split(X_balanced,Y_balanced,
test_size=0.3,random_state=5)
plt.subplot(121)
sns.countplot(y_train)
plt.title('Imlalanced dataset')


plt.subplot(122)
sns.countplot(y_train_balc)
plt.title('balanced dataset')
plt.tight_layout()
plt.show()
```



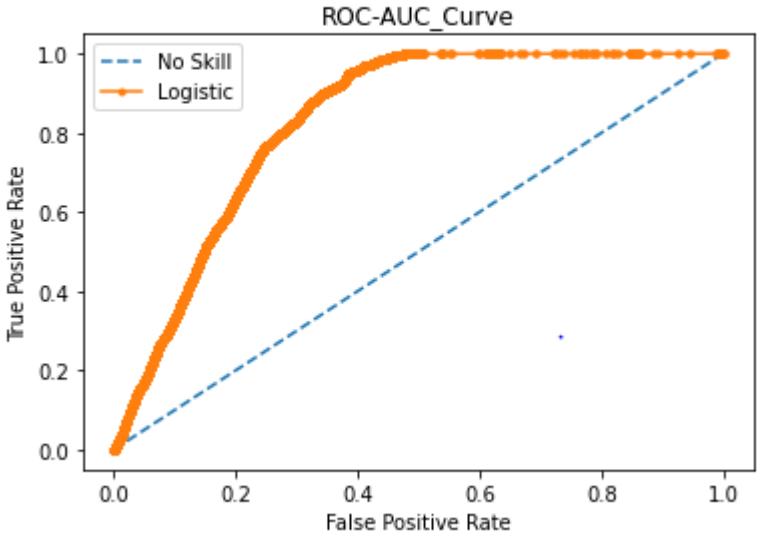
After balancing the response variable apply logistic regression model and check accuracy score is incresed or not.

```
lr_balanced=LogisticRegression(solver='liblinear',random_state=0)
lr_balanced.fit(x_train_balc,y_train_balc)
```

ROC CURVE

```
## After Balancing
ns_probs = [0 for _ in range(len(y_test_balc))]


lr_probs = lr_balanced.predict_proba(x_test_balc)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_test_balc, ns_probs)
lr_auc = roc_auc_score(y_test_balc, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test_balc, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test_balc, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot

plt.title('ROC-AUC_Curve')

plt.show()
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.830
```



We can clearly see that after Balancing the ratio of target variable . Our ROC-AUC score has increased to from 0.81 to 0.83.

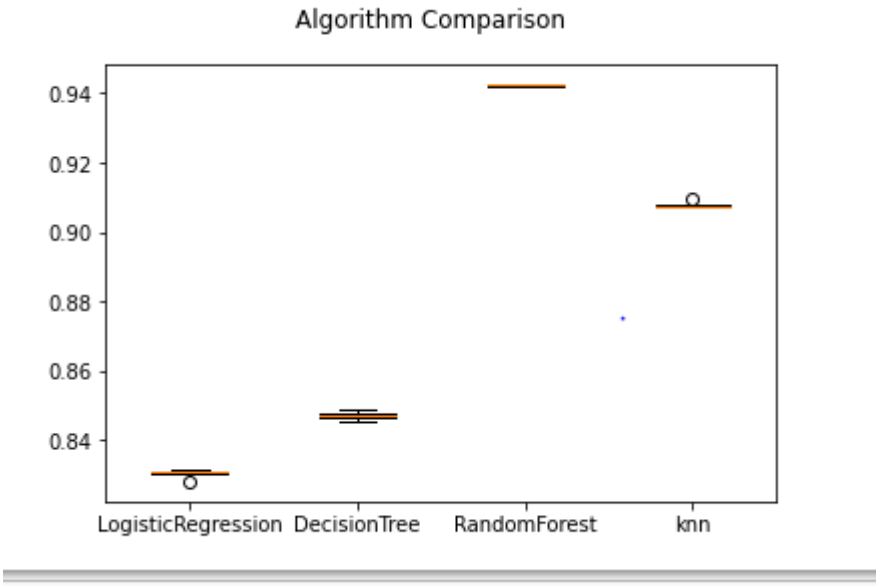
We will be using Cross-Validation to find the model which has the best accuracy.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
models=[]
models.append(('LogisticRegression',LogisticRegression(solver="liblinear", random_state=5)))
models.append(('DecisionTree',DecisionTreeClassifier(random_state=5)))
models.append(('RandomForest',RandomForestClassifier(random_state=5)))

models.append(('knn',KNeighborsClassifier()))
from xgboost.compat import KFold
results=[]
names=[]
for name,model in models:
  kf=KFold(n_splits=5,shuffle=True,random_state=5)
  score=cross_val_score(model,X_balanced,Y_balanced,cv=kf,scoring='roc_auc',verbose=1
)
```

```
results.append(score)
names.append(name)
msg='%s :%f (%f)'%(name,score.mean(),score.std())
print(msg)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
LogisticRegression :0.830226 (0.001165)
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   25.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
DecisionTree :0.846939 (0.001262)
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed: 11.6min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
RandomForest :0.942293 (0.000135)
knn :0.907821 (0.000953)
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   51.4s finished
```

```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Logistic Regression using stratified k-folds cross validation

To check how robust our model is to unseen data, we can use Validation. It is a technique which involves reserving a particular sample of a dataset on which you do not train the model. Later, you test your model on this sample before finalizing it. Some of the common methods for validation are listed below:

- **The validation set approach**
- **k-fold cross validation**
- **Leave one out cross validation (LOOCV)**
- **Stratified k-fold cross validation**

In this section we will learn about **stratified k-fold cross validation**. Let us understand how it works:

- • Stratification is the process of rearranging the data so as to ensure that each fold is a good representative of the whole.
- • For example, in a binary classification problem where each class comprises of 50% of the data, it is best to arrange the data such that in every fold, each class comprises of about half the instances.
- • It is generally a better approach when dealing with both bias and variance.
- • A randomly selected fold might not adequately represent the minor class, particularly in cases where there is a huge class imbalance.

Let's import StratifiedKFold from sklearn and fit the model.

```
from sklearn.model_selection import StratifiedKFold
```

Now let's make a cross validation logistic model with stratified 5 folds and make predictions for test dataset.

```
i=1
kf = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
  print('\n{} of kfold {}'.format(i,kf.n_splits))
  xtr,xvl = X.loc[train_index],X.loc[test_index]
  ytr,yvl = y[train_index],y[test_index]
```

```
model = LogisticRegression(random_state=1)
model.fit(xtr, ytr)

pred_test = model.predict(xvl)
score = accuracy_score(yvl,pred_test)
print('accuracy_score',score)

i+=1
pred_test = model.predict(test)
pred=model.predict_proba(xvl)[:,1]
```

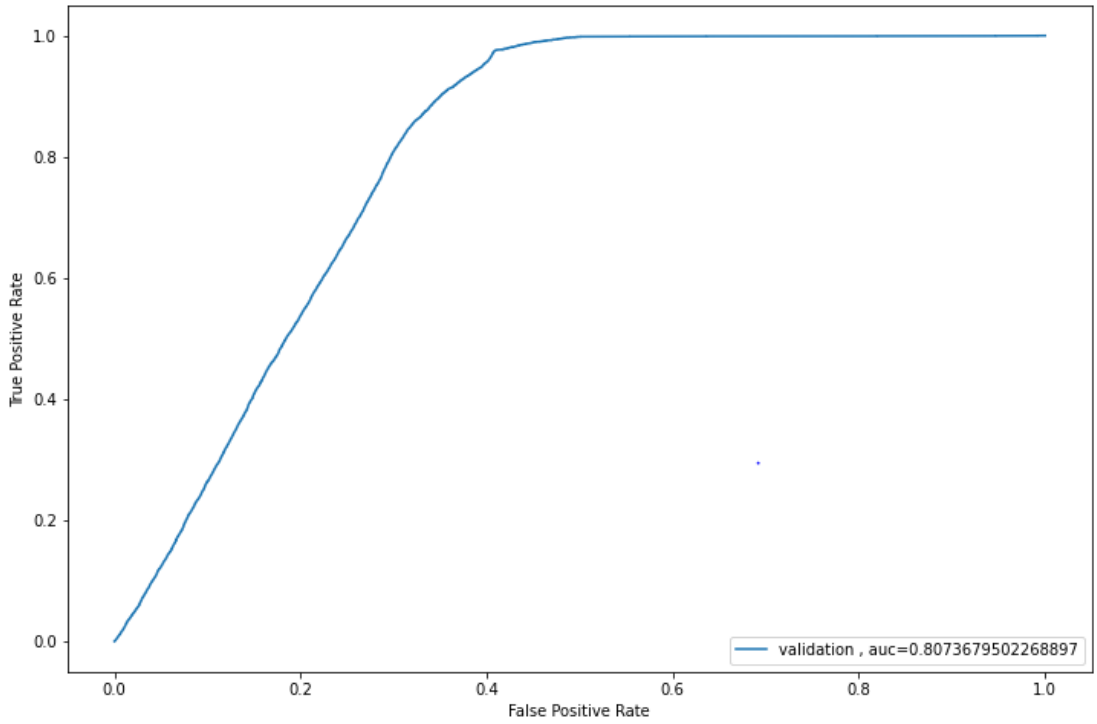
```
1 of kfold 5
accuracy_score 0.8773582430269476

2 of kfold 5
accuracy_score 0.8763217968565506

3 of kfold 5
accuracy_score 0.8767678622969747

4 of kfold 5
accuracy_score 0.8774107213140563

5 of kfold 5
accuracy_score 0.8773303945106992
```

The mean validation accuracy for this model turns out to be 0.87. Let us visualize the roc curve.

ROC CURVE

```
from sklearn import metrics
fpr,tpr,_=metrics.roc_curve(yvl,pred)
auc=metrics.roc_auc_score(yvl,pred)
plt.figure(figsize=(12,8))
plt.plot(fpr,tpr,label='validation , auc='+str(auc))
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```



We got an auc value of 0.80.

Decision Tree Classifier

Decision tree is a type of supervised learning algorithm(having a pre-defined target variable) that is mostly used in classification problems. In this technique, we split the population or sample into two or more homogeneous sets(or sub-populations) based on most significant splitter / differentiator in input variables.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable.

```
from sklearn.tree import DecisionTreeClassifier
```

Let’s fit the decision tree model with 5 folds of cross validation.

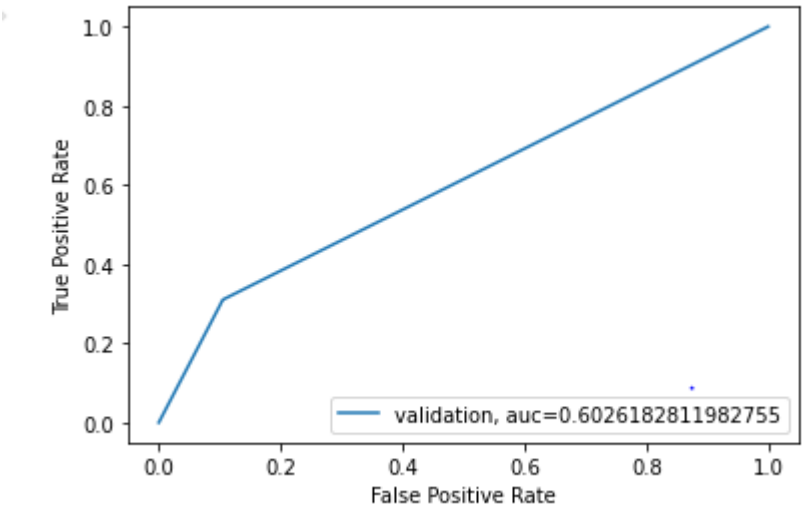
```
i=1
kf=StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
  print('\n {} of kfold {} '.format(i,kf.n_splits))
  xtr,xvl=X.loc[train_index],X.loc[test_index]
  ytr,yvl=y[train_index],y[test_index]

  model=DecisionTreeClassifier(random_state=1)
  model.fit(xtr,ytr)
  pred_test=model.predict(xvl)
  score=accuracy_score(yvl,pred_test)
  print('accuracy score : ',score)
  i+=1
  pred_test1=model.predict(test)
  pred_prob=model.predict_proba(xvl)[:,1]
```

```
 1 of kfold 5
accuracy score :  0.822741465718559

 2 of kfold 5
accuracy score :  0.8232006507307601

 3 of kfold 5
accuracy score :  0.8230169767258797

 4 of kfold 5
accuracy score :  0.8247225210569127

 5 of kfold 5
accuracy score :  0.8246677424856667
```

The mean validation accuracy for this model turns out to be 0.82. Let us visualize the roc curve.

ROC CURVE

```
fpr,tpr,_=roc_curve(yvl,pred_prob)
auc=roc_auc_score(yvl,pred_prob)
plt.plot(fpr,tpr,label='validation, auc='+str(auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```



We got an auc value of 0.60.

We got an accuracy of 0.82 which is much lesser than the accuracy from logistic regression model. So let's build another model, i.e. Random Forest, a tree based ensemble algorithm and try to improve our model by improving the accuracy.

Random Forest Classifier

- RandomForest is a tree based bootstrapping algorithm wherein a certain no. of weak learners (decision trees) are combined to make a powerful prediction model.
- For every individual learner, a random sample of rows and a few randomly chosen variables are used to build a decision tree model.
- Final prediction can be a function of all the predictions made by the individual learners.
- In case of regression problem, the final prediction can be mean of all the predictions.

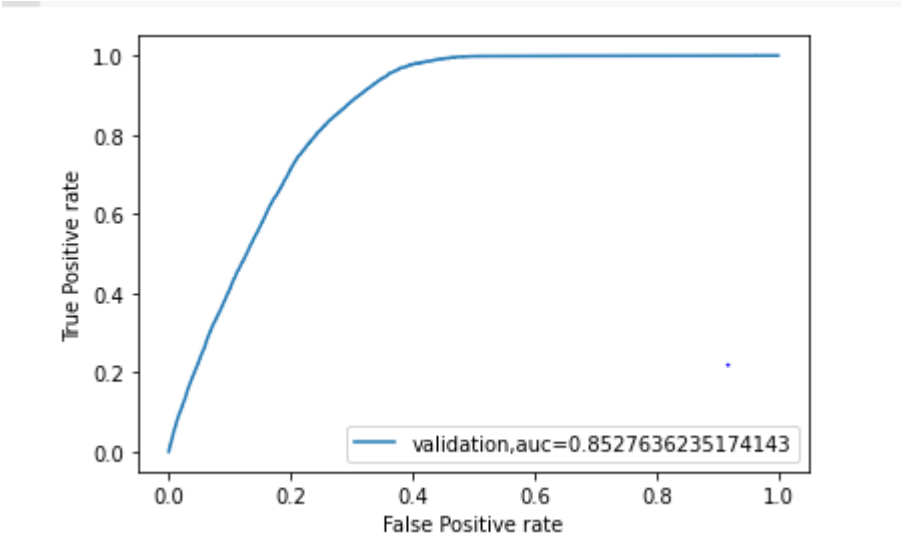
```
from sklearn.ensemble import RandomForestClassifier
n=1
kf=StratifiedKFold(n_splits=5,shuffle=True,random_state=1)
for train_index,test_index in kf.split(X,y):
  print('\n {} of kfold {} '.format(n,kf.n_splits))
  xtr,xvl=X.loc[train_index],X.loc[test_index]
  ytr,yvl=y[train_index],y[test_index]

  model=RandomForestClassifier(random_state=1,max_depth=10)
  model.fit(xtr,ytr)
  pred_test=model.predict(xvl)
  acc=accuracy_score(yvl,pred_test)
  print('accuracy score : ',acc)
  n+=1

  pred_test=model.predict(test)
  pred_prob=model.predict_proba(xvl)[:,1]
```

```
 1 of kfold 5
accuracy score :  0.8774369604576107

 2 of kfold 5
accuracy score :  0.8774369604576107

 3 of kfold 5
accuracy score :  0.8774369604576107

 4 of kfold 5
accuracy score :  0.8774238408858335

 5 of kfold 5
accuracy score :  0.87743535246192
```

The mean validation accuracy for this model turns out to be 0.87. Let us visualize the roc curve.

ROC CURVE

```
fpr,tpr,_=roc_curve(yvl,pred_prob)
auc=roc_auc_score(yvl,pred_prob)
plt.plot(fpr,tpr,label='validation,auc='+str(auc))
plt.xlabel('False Positive rate')
plt.ylabel('True Positive rate')
plt.legend(loc=4)
plt.show()
```



We got an auc value of 0.85.

Grid-search

We will try to improve the accuracy by tuning the hyperparameters for this model. We will use grid search to get the optimized values of hyper parameters.

Grid-search is a way to select the best of a family of hyper parameters, parametrized by a grid of parameters. We will tune the max_depth and n_estimators parameters. max_depth decides the maximum depth of the tree and n_estimators decides the number of trees that will be used in random forest model.

```
from sklearn.model_selection import GridSearchCV
paramgrid={'max_depth':list(range(1,20,2)),'n_estimators':list(range(1,80,20))}
grid_cv=GridSearchCV(RandomForestClassifier(max_depth=1,random_state=1),param_grid=paramgrid)
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
```

```
grid_cv.fit(x_train,y_train)
```

```
GridSearchCV(estimator=RandomForestClassifier(max_depth=1, random_state=1),
             param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19],
                         'n_estimators': [1, 21, 41, 61]})
```

```
grid_cv.best_estimator_
```

```
RandomForestClassifier(max_depth=11, n_estimators=41, random_state=1)
```

So, the optimized value for the max_depth variable is 11 and for n_estimator is 41. Now let's build the model using these optimized values.

```
i=1
kf=StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
  print('\n {} of kfold {} '.format(i,kf.n_splits))
```

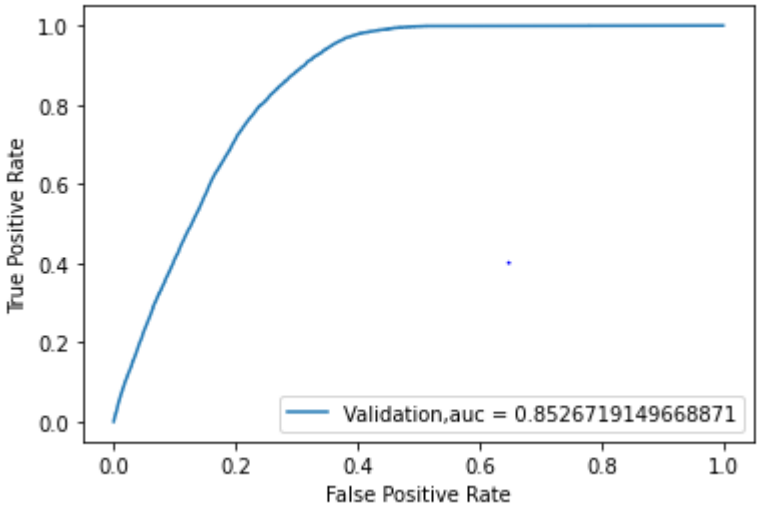
```
xtr,xvl=X.loc[train_index],X.loc[test_index]
ytr,yvl=y[train_index],y[test_index]

model=RandomForestClassifier(n_estimators=41,max_depth=11,random_state=1)
model.fit(xtr,ytr)
pred_test=model.predict(xvl)
acc=accuracy_score(yvl,pred_test)
print('accuracy score : ',acc)
i+=1
pred_test=model.predict(test)
pred2=model.predict_proba(xvl)[:,1]
```

```
 1 of kfold 5
accuracy score :  0.8774238408858335

 2 of kfold 5
accuracy score :  0.8774763191729422

 3 of kfold 5
accuracy score :  0.8774369604576107

 4 of kfold 5
accuracy score :  0.8774107213140563

 5 of kfold 5
accuracy score :  0.87743535246192
```

The mean validation accuracy for this model turns out to be 0.87. Let us visualize the roc curve.

ROC CURVE

```
fpr,tpr,_=roc_curve(yvl,pred2)
auc=roc_auc_score(yvl,pred2)
plt.plot(fpr,tpr,label='Validation,auc = '+str(auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```

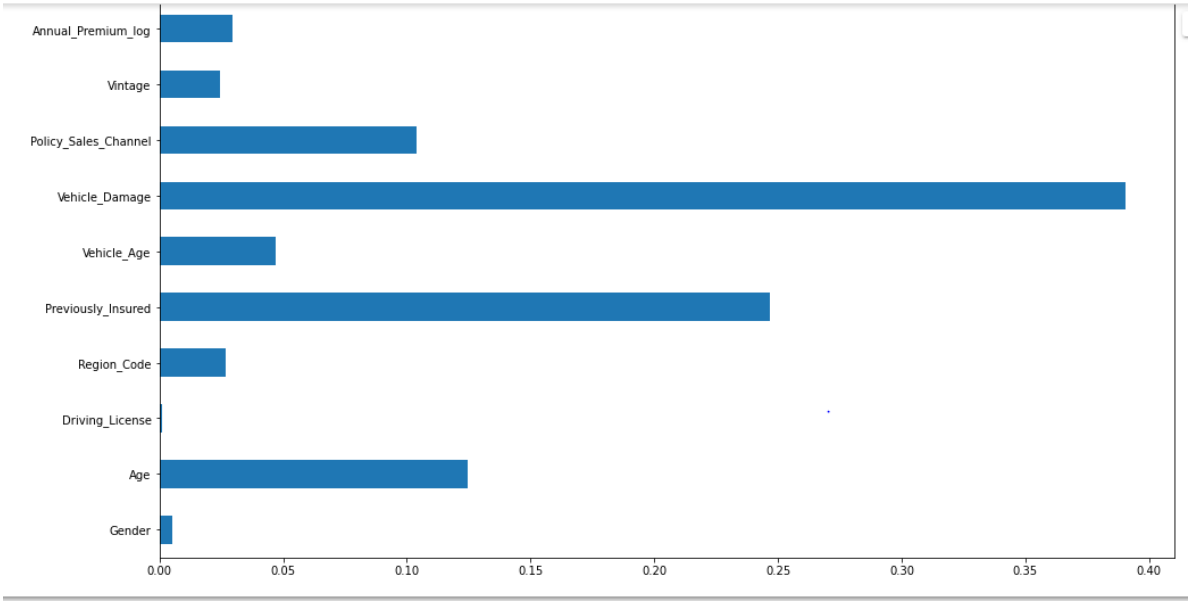


We got an auc value of 0.85.

feature importance

Let us find the feature importance now, i.e. which features are most important for this problem. We will use feature_importances_ attribute of sklearn to do so.

```
importances=pd.Series(model.feature_importances_,index=X.columns)
importances.plot(kind='barh',figsize=(16,9))
```



We can see that vehicle_damage is the most important feature followed by previously_insured,Age, policy_sales_channel. So, feature engineering helped us in predicting our target variable.

XGBoost

XGBoost is a fast and efficient algorithm and has been used to by the winners of many data science competitions.

XGBoost works only with numeric variables and we have already replaced the categorical variables with numeric variables. Let's have a look at the parameters that we are going to use in our model.

• **n_estimator:** This specifies the number of trees for the model.

• **max_depth:** We can specify maximum depth of a tree using this parameter.

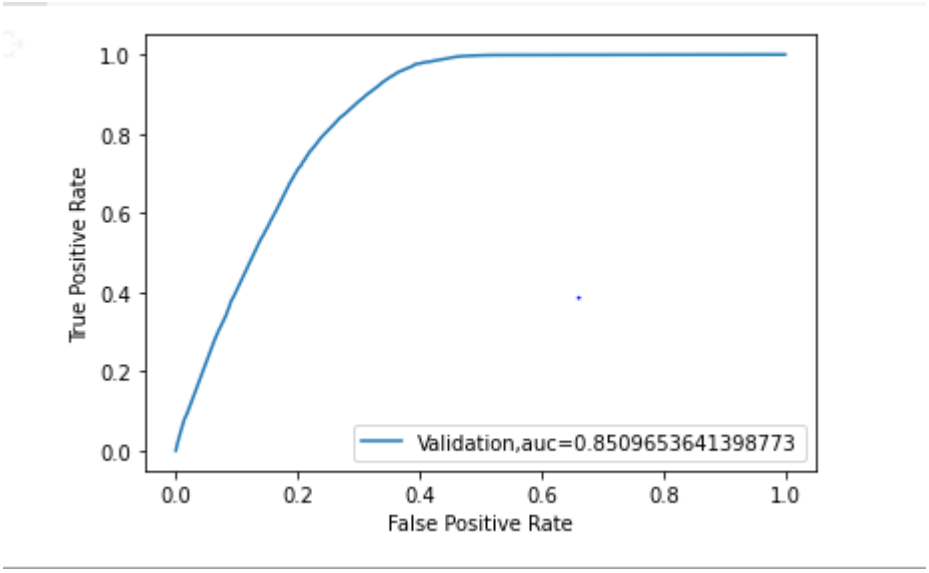
```
from xgboost import XGBClassifier
n=1
kf=StratifiedKFold(n_splits=5,shuffle=True,random_state=1)
for train_index,test_index in kf.split(X,y):
  print('\n {} of kfold {} :'.format(n,kf.n_splits))
  xtr,xvl=X.loc[train_index],X.loc[test_index]
  ytr,yvl=y[train_index],y[test_index]

  model=XGBClassifier(n_estimators=50,max_depth=4)
  model.fit(xtr,ytr)
  pred_test=model.predict(xvl)
  acc=accuracy_score(yvl,pred_test)
  print('accuracy score : ',acc)
  n+=1

  pred_test=model.predict(test)
  pred3=model.predict_proba(xvl)[:,1]
```

```
 1 of kfold 5 :
accuracy score :  0.8774369604576107

 2 of kfold 5 :
accuracy score :  0.8774369604576107

 3 of kfold 5 :
accuracy score :  0.8774369604576107

 4 of kfold 5 :
accuracy score :  0.8774369604576107

 5 of kfold 5 :
accuracy score :  0.87743535246192
```

The mean validation accuracy for this model turns out to be 0.87. Let us visualize the roc curve.

ROC CURVE

```
fpr,tpr,_=roc_curve(yvl,pred3)
auc=roc_auc_score(yvl,pred3)
plt.plot(fpr,tpr,label='Validation,auc='+str(auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```



We got an auc value of 0.85.

KNeighbour Classifier

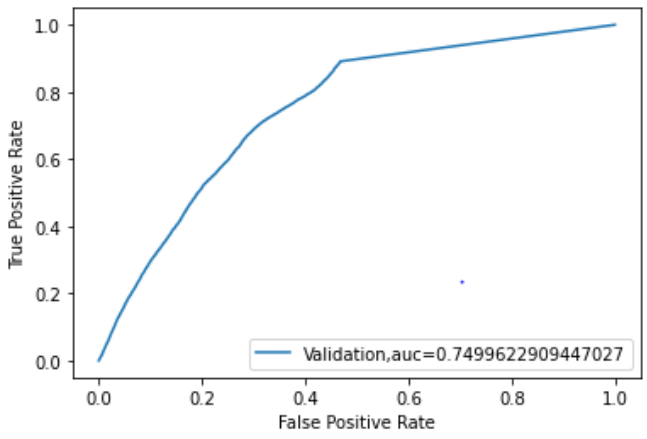
```
from sklearn.neighbors import KNeighborsClassifier
n=1
kf=StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
  print('\n {} of kfold {} : '.format(n,kf.n_splits))
  xtr,xvl=X.loc[train_index],X.loc[test_index]
  ytr,yvl=y[train_index],y[test_index]
  model=KNeighborsClassifier(n_neighbors=10,weights='distance',p=2,leaf_size=10)
  model.fit(xtr,ytr)
  pred_test=model.predict(xvl)
  acc=accuracy_score(yvl,pred_test)
  print('accuracy score : ',acc)
  n+=1

  pred_test=model.predict(test)
  pred4=model.predict_proba(xvl)[:,1]
```

```
 1 of kfold 5 :
accuracy score :  0.8561701346068065

 2 of kfold 5 :
accuracy score :  0.8588727663929049

 3 of kfold 5 :
accuracy score :  0.8581905486604917

 4 of kfold 5 :
accuracy score :  0.8566293196190077

 5 of kfold 5 :
accuracy score :  0.8577819760958266
```

The mean validation accuracy for this model turns out to be 0.85. Let us visualize the roc curve.

ROC CURVE

```
fpr,tpr,_=roc_curve(yvl,pred4)
auc=roc_auc_score(yvl,pred4)
plt.plot(fpr,tpr,label='Validation,auc='+str(auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```



We got an auc value of 0.74.

We can see that **RandomForest** has performed the best so we will go with it.

we will apply the random forest model on the balanced response variable.

```
rtree=RandomForestClassifier(random_state=5)
rtree.fit(x_train_balc,y_train_balc)
```

```
RandomForestClassifier(random_state=5)
```

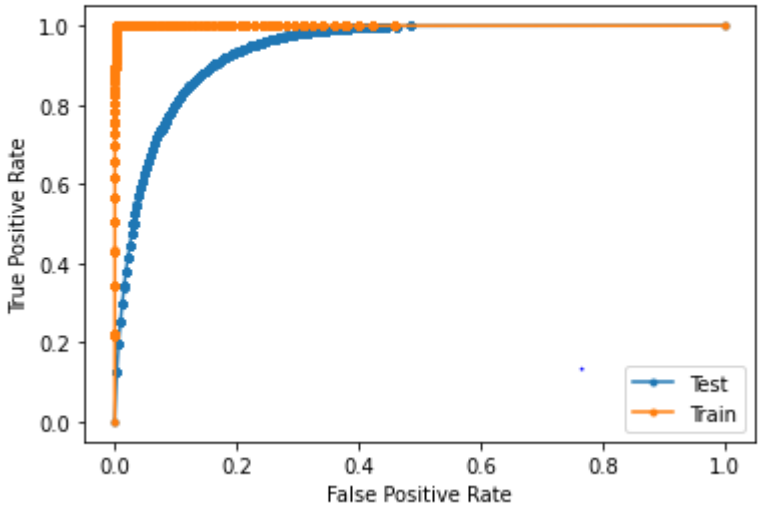
```
y_pred=rtree.predict(x_test_balc)
print('roc_auc_score for Random Forest',roc_auc_score(y_test_balc,y_pred))
```

```
roc_auc_score for Random Forest 0.8693563914062096
```

The roc auc score of y actual and y predicted for random forest model is 0.86

```
test_prob=rtree.predict_proba(x_test_balc)[:,1]
train_prob=rtree.predict_proba(x_train_balc)[:,1]
test_auc=roc_auc_score(y_test_balc,test_prob)
train_auc=roc_auc_score(y_train_balc,train_prob)
print('Test Score : ROC_AUC=%.3f'%(test_auc))
print('Train Score : ROC_AUC=%.3f'%(train_auc))
test_fpr,test_tpr,_=roc_curve(y_test_balc,test_prob)
train_fpr,train_tpr,_=roc_curve(y_train_balc,train_prob)
plt.plot(test_fpr,test_tpr,marker='.',label='Test')
plt.plot(train_fpr,train_tpr,marker='.',label='Train')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

```
Test Score : ROC_AUC=0.940
Train Score : ROC_AUC=1.000
```



We got an auc value of 0.94.

We will apply **XGB** on balanced response variable and check the accuracy rate

```
xgb=XGBClassifier(random_state=5,max_depth=5)
xgb.fit(x_train_balc,y_train_balc)
```

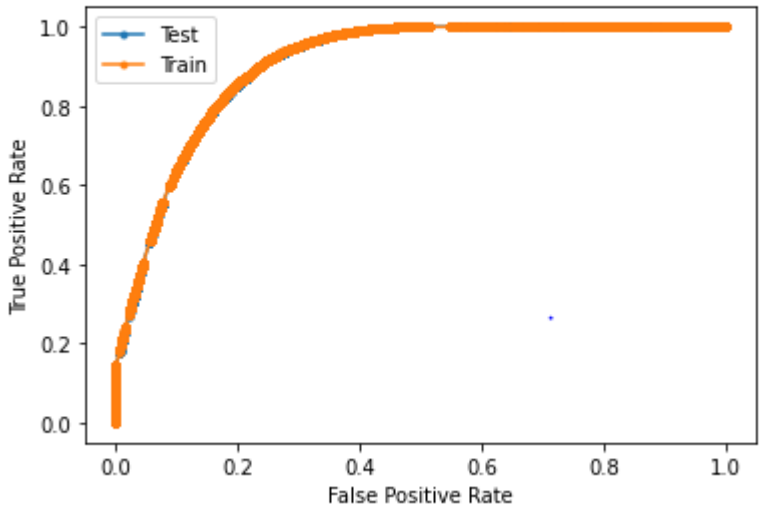
```
XGBClassifier(max_depth=5, random_state=5)
```

ROC CURVE

```
test_prob=xgb.predict_proba(x_test_balc)[:,1]
train_prob=xgb.predict_proba(x_train_balc)[:,1]
auc_test=roc_auc_score(y_test_balc,test_prob)
auc_train=roc_auc_score(y_train_balc,train_prob)
print('Test_Score :ROC_AUC=%.3f'%(auc_test))
print('Train_Score:ROC_AUC=%.3f'%(auc_train))
test_fpr,test_tpr,_=roc_curve(y_test_balc,test_prob)
train_fpr,train_tpr,_=roc_curve(y_train_balc,train_prob)
plt.plot(test_fpr,test_tpr,marker='.',label="Test")
```

```
plt.plot(train_fpr,train_tpr,marker='.',label="Train")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

```
Test_Score :ROC_AUC=0.906
Train_Score:ROC_AUC=0.906
```



We got an auc value for test data is 0.90.

We got an auc value for train data is 0.90.

```
y_pred=xgb.predict(x_test_balc)
print('ROC AUC Score of XGBClassifier :',roc_auc_score(y_test_balc,y_pred))
```

```
ROC AUC Score of XGBClassifier : 0.8329002797011592
```

The roc auc score of y actual value and y predicted value is 0.83

8.7 Final Model

After applying all the models random forest model is best model for this dataset.Because the accuracy rate of random forest model is greater than all of the other model.So we will make final prediction with the random forest model.

Making final model

```
pred_test=rtree.predict(test)
pred_test
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
submission=pd.DataFrame({'id':test_org['id'],'Response':pred_test})
```

```
submission['Response'].value_counts()
```

```
0    115440
1     11597
Name: Response, dtype: int64
```

```
submission.to_csv('rtree_submission.csv',index=False)
```

CHAPTER 9: CONCLUSION

We have collected data from various 3rd party sources and processed them and we computed the data to visualize some of necessary use case. Based on the above analysis the healthcare insurance company will create a new business strategy to acquire more customers, engagement and send offers. As well as fetching the company and customer details and provide easy access to information regarding customers.

For cross sell prediction, the aim of our project was to classify whether customer will take vehicle insurance or not. We used dataset from kaggle and performed exploratory data analysis for descriptive statistics. So, for this dataset Random Forest Classifier is best model because its accuracy score is greater than all of the other model.

CHAPTER 10: FUTURE SCOPE

This project has a very vast scope in future in this field. There are various use cases that can be achieved by this project. Some of future scopes are below:

- ✓ Real time data can also be used for real time processing.
- ✓ We can automate the whole procedure where data coming from sources and getting executed at a same time.
- ✓ Betterment of results using different hyperparameters for tuning
- ✓ Implementing more models to gain better results
- ✓ Using the same method to predict response for various other kinds of insurances
- ✓ Combining all the processing of insurance offer advertisements and achieving the best customers for the same.
- ✓ Not in the Healthcare industry we can generalized the whole procedure to other sectors like cars, online education system etc.

CHAPTER 11: REFERENCES/ BIBLIOGRAPHY

Books and Reports

[1] Beranger, Jérôme. 2016. Ethics in Big Data: the medical datasphere. London: Elsevier.

[2] Davis, Cord and Patterson, Doug. 2012. Ethics of Big Data. Farnham, O'Reilly.

[3] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers.

Web Sites

[1] Big Data Ethics

[2] GeekforGeeks (GeeksforGeeks | A computer science portal for geeks)

[3] Analytics Vidhya (Analytics Vidhya - Learn Machine learning, artificial intelligence, business analytics, data science, big data, data visualizations tools and techniques. | Analytics Vidhya)