Insurance Cost Prediction

Problem Statement

- Insurance companies need to accurately predict the cost of health insurance for individuals to set premiums appropriately. However, traditional methods of cost prediction often rely on broad actuarial tables and historical averages, which may not account for the nuanced differences among individuals. By leveraging machine learning techniques, insurers can predict more accurately the insurance costs tailored to individual profiles, leading to more competitive pricing and better risk management.
- Down here is a detailed data analysis on the given insurance data, finding the major factors of the individuals inflicting the final premium price and creating our own machine learning model to do the cost prediction for the given data.

EDA and Hypothesis Testing for Insurance Cost Prediction(Block 2)

Importing libraries

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

```
df = pd.read_csv("/content/drive/MyDrive/insurance.csv")
df
```

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFami
0	45	0	0	0	0	155	57	0	
1	60	1	0	0	0	180	73	0	
2	36	1	1	0	0	158	59	0	
3	52	1	1	0	1	183	93	0	
4	38	0	0	0	1	166	88	0	
...	...	...	...	...	...	...	...	...	
981	18	0	0	0	0	169	67	0	
982	64	1	1	0	0	153	70	0	
983	56	0	1	0	0	155	71	0	
984	47	1	1	0	0	158	73	1	
985	21	0	0	0	0	158	75	1	

986 rows × 11 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

Basic EDA

```
df.shape
```

```
(986, 11)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 986 entries, 0 to 985
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    986 non-null    int64
 1   Diabetes               986 non-null    int64
 2   BloodPressureProblems  986 non-null    int64
 3   AnyTransplants         986 non-null    int64
 4   AnyChronicDiseases     986 non-null    int64
```

```
 5   Height                   986 non-null    int64
 6   Weight                   986 non-null    int64
 7   KnownAllergies           986 non-null    int64
 8   HistoryOfCancerInFamily  986 non-null    int64
 9   NumberOfMajorSurgeries   986 non-null    int64
 10  PremiumPrice             986 non-null    int64
dtypes: int64(11)
memory usage: 84.9 KB
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	986.0	41.745436	13.963371	18.0	30.0	42.0	53.0	66.0
Diabetes	986.0	0.419878	0.493789	0.0	0.0	0.0	1.0	1.0
BloodPressureProblems	986.0	0.468560	0.499264	0.0	0.0	0.0	1.0	1.0
AnyTransplants	986.0	0.055781	0.229615	0.0	0.0	0.0	0.0	1.0
AnyChronicDiseases	986.0	0.180527	0.384821	0.0	0.0	0.0	0.0	1.0
Height	986.0	168.182556	10.098155	145.0	161.0	168.0	176.0	188.0
Weight	986.0	76.950304	14.265096	51.0	67.0	75.0	87.0	132.0
KnownAllergies	986.0	0.215010	0.411038	0.0	0.0	0.0	0.0	1.0
HistoryOfCancerInFamily	986.0	0.117647	0.322353	0.0	0.0	0.0	0.0	1.0
NumberOfMajorSurgeries	986.0	0.667343	0.749205	0.0	0.0	1.0	1.0	3.0
PremiumPrice	986.0	24336.713996	6248.184382	15000.0	21000.0	23000.0	28000.0	40000.0

```
df.isnull().sum()
```

	0
Age	0
Diabetes	0
BloodPressureProblems	0
AnyTransplants	0
AnyChronicDiseases	0
Height	0
Weight	0
KnownAllergies	0
HistoryOfCancerInFamily	0
NumberOfMajorSurgeries	0
PremiumPrice	0

- Hence we can conclude that there are **no NULL** values in the data

```
duplicates = df.duplicated()
print(duplicates)
```

```
0      False
1      False
2      False
3      False
4      False
       ...
981    False
982    False
983    False
984    False
985    False
Length: 986, dtype: bool
```

```
#Creating a copy of the data for extended analysis
df1 = df.copy()
df1.head()
```

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily
0	45	0	0	0	0	155	57	0	0
1	60	1	0	0	0	180	73	0	0
2	36	1	1	0	0	158	59	0	0
3	52	1	1	0	1	183	93	0	0
4	38	0	0	0	1	166	88	0	0

Next steps: Generate code with df1 View recommended plots New interactive sheet

```
df1.Age.min(), df.Age.max()
```

```
(18, 66)
```

```
df1.PremiumPrice.min(), df1.PremiumPrice.max()
```

```
(15000, 40000)
```

```
# Creating new categories in Age and Premium data received for better analysis
df1["age_cat"] = pd.cut(df1["Age"], bins = [18, 25, 35, 55, 66], labels = ["Youth", "Young-adults", "middle aged adults", "senior citize
df1["premium_cat"] = pd.cut(df1["PremiumPrice"], bins = [15000,20000,30000,40000], labels = ["low", "medium", "high"])
df1.head()
```

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily
0	45	0	0	0	0	155	57	0	0
1	60	1	0	0	0	180	73	0	0
2	36	1	1	0	0	158	59	0	0
3	52	1	1	0	1	183	93	0	0
4	38	0	0	0	1	166	88	0	0

Next steps: Generate code with df1 View recommended plots New interactive sheet

```
# Creating BMI index(numerical column)
df1["BMI"] = df1["Weight"] / (df1["Height"] / 100) ** 2
df1["BMI"] = df1["BMI"].round(2)
df1.head()
```

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily
0	45	0	0	0	0	155	57	0	0
1	60	1	0	0	0	180	73	0	0
2	36	1	1	0	0	158	59	0	0
3	52	1	1	0	1	183	93	0	0
4	38	0	0	0	1	166	88	0	0

Next steps: Generate code with df1 View recommended plots New interactive sheet

```
df1["BMI"].min(), df1["BMI"].max()
```

```
(15.16, 50.0)
```

```
# Creating BMI cat for analysis
bmi_cat = pd.cut(df1["BMI"], bins = [15.00, 18.50, 25.00, 30.00, 40.00, 50.00], labels = ["Underweight", "Normal", "Overweight", "Obesity
df1["bmi_cat"] = bmi_cat
df1.head()
```

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily
0	45	0	0	0	0	155	57	0	0
1	60	1	0	0	0	180	73	0	0
2	36	1	1	0	0	158	59	0	0
3	52	1	1	0	1	183	93	0	0
4	38	0	0	0	1	166	88	0	0

Next steps: Generate code with df1 View recommended plots New interactive sheet

```
df1.shape
```

```
(986, 15)
```

- Now the revised data has 15 columns

```
# Finding the unique entries in each column
for i in df1.columns:
  print(f"Unique entries in {i: <25} = {df1[i].nunique()}")
```

```
Unique entries in Age                       = 49
Unique entries in Diabetes                  = 2
Unique entries in BloodPressureProblems     = 2
Unique entries in AnyTransplants            = 2
Unique entries in AnyChronicDiseases        = 2
Unique entries in Height                    = 44
Unique entries in Weight                    = 74
Unique entries in KnownAllergies            = 2
Unique entries in HistoryOfCancerInFamily   = 2
Unique entries in NumberOfMajorSurgeries    = 4
Unique entries in PremiumPrice              = 24
Unique entries in age_cat                   = 4
Unique entries in premium_cat               = 3
Unique entries in BMI                       = 631
Unique entries in bmi_cat                   = 5
```

```
# Age count to show the entries with highest and lowest age group
Age_count = df1["Age"].value_counts()
Age_count.head(), Age_count.tail()
```

```
(Age
 43    30
 27    27
 42    27
 35    26
 45    25
 Name: count, dtype: int64,
 Age
 56    15
 23    13
 26    13
 57    12
 39    11
 Name: count, dtype: int64)
```

```
# Premium price count to show the premium which was received by max members and min members
pp_count = df1["PremiumPrice"].value_counts()
pp_count.head(), pp_count.tail()
```

```
(PremiumPrice
 23000    249
```

```
15000    202
28000    132
25000    103
29000     72
Name: count, dtype: int64,
PremiumPrice
22000    1
40000    1
20000    1
27000    1
17000    1
Name: count, dtype: int64)
```

```
Surgery_count = df1["NumberOfMajorSurgeries"].value_counts()
Surgery_count
```

NumberOfMajorSurgeries	count
0	479
1	372
2	119
3	16

- Shows that Maximum people in the entry have had "0" surgeries and only 16 people have had "3" surgeries.

```
Age_cat_count = df1["age_cat"].value_counts()
Age_cat_count
```

age_cat	count
middle aged adults	413
Young-adults	210
senior citizens	203
Youth	137

- Shows that maximum entries belongs to middle aged category(aged between 35-55 yrs). And only minimum entries of youth were present(aged between 18-25 yrs).

```
premium_cat_count = df1["premium_cat"].value_counts()
premium_cat_count
```

premium_cat	count
medium	642
high	120
low	22

- Maximum people have received only average premium ranging between 20,000 to 30,000.

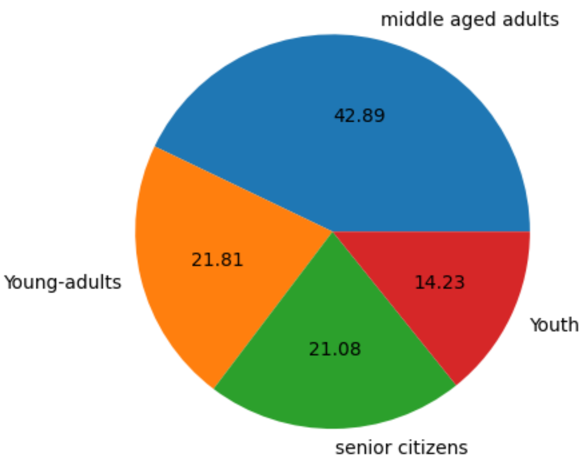
```
bmi_cat_count = df1["bmi_cat"].value_counts()
bmi_cat_count
```

bmi_cat	count
Overweight	326
Normal	319
Obesity	266
Underweight	39
Extreme Obesity	36

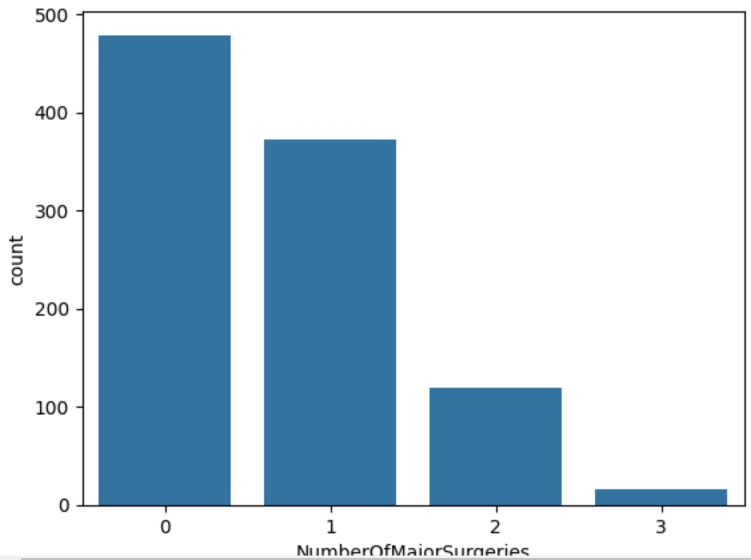
- Maximum entries are found to be "Overweight" ranging from 25.00 to 30.00 in BMI index.

Univariate visual analysis

```
from seaborn.widgets import color_palette
plt.pie(Age_cat_count, labels = Age_cat_count.index, autopct = "%.2f")
plt.show()
```



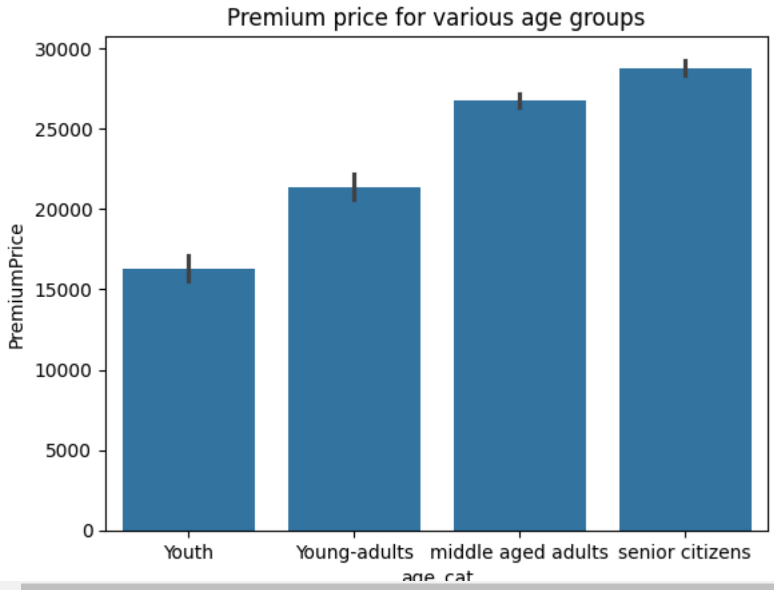

```
sns.countplot(data = df1, x = df1["NumberOfMajorSurgeries"])
plt.show()
```



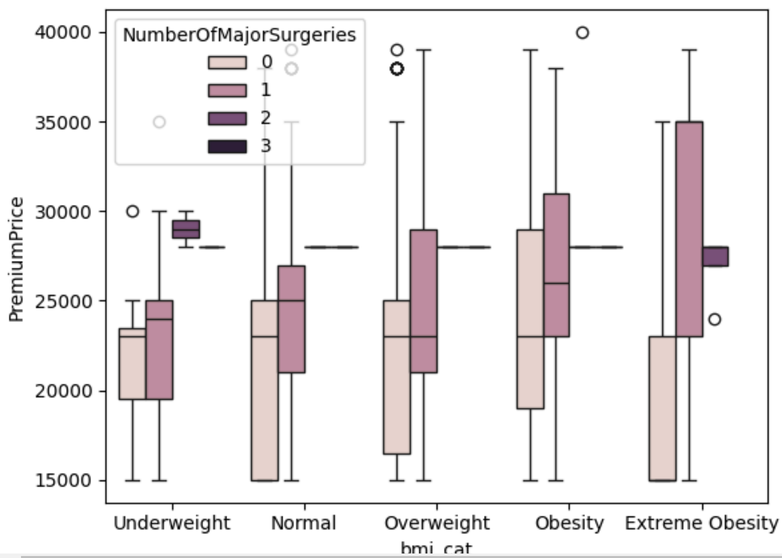
Multivariate visual analysis

```
plt.title("Premium price for various age groups")
sns.barplot(data = df1, x=df1["age_cat"], y = df1["PremiumPrice"])
```

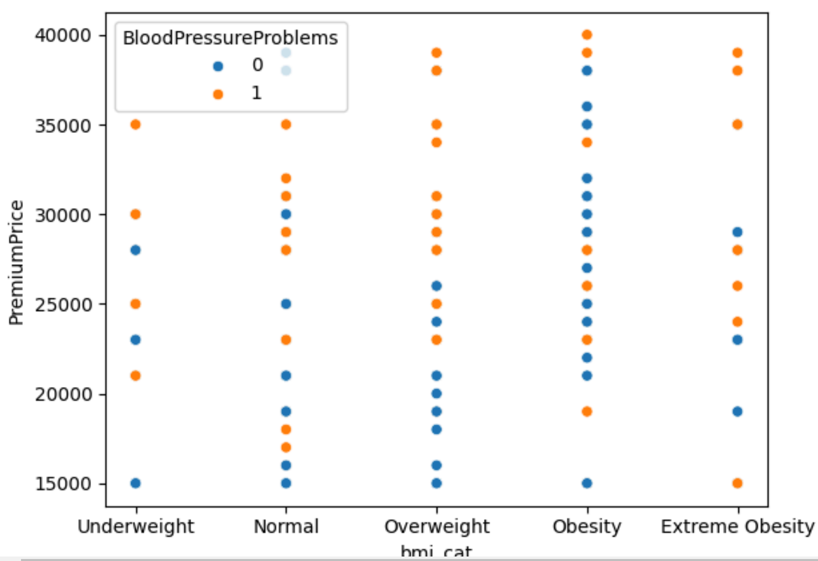
```
plt.show()
```




```
sns.boxplot(data=df1, x="bmi_cat", y="PremiumPrice", hue="NumberOfMajorSurgeries")
plt.show()
```

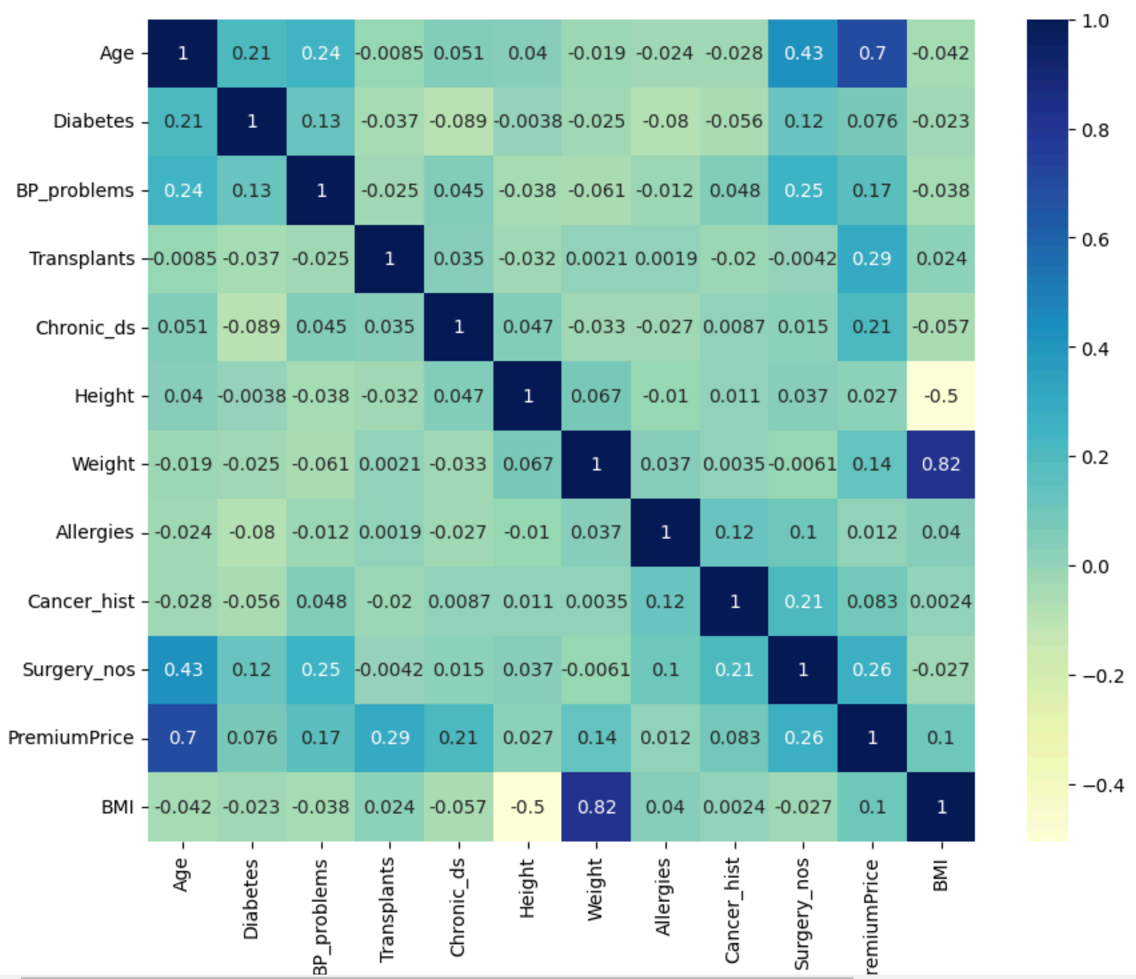



```
sns.scatterplot(data=df1, x= "bmi_cat", y="PremiumPrice", hue="BloodPressureProblems")
plt.show()
```



Correlation analysis

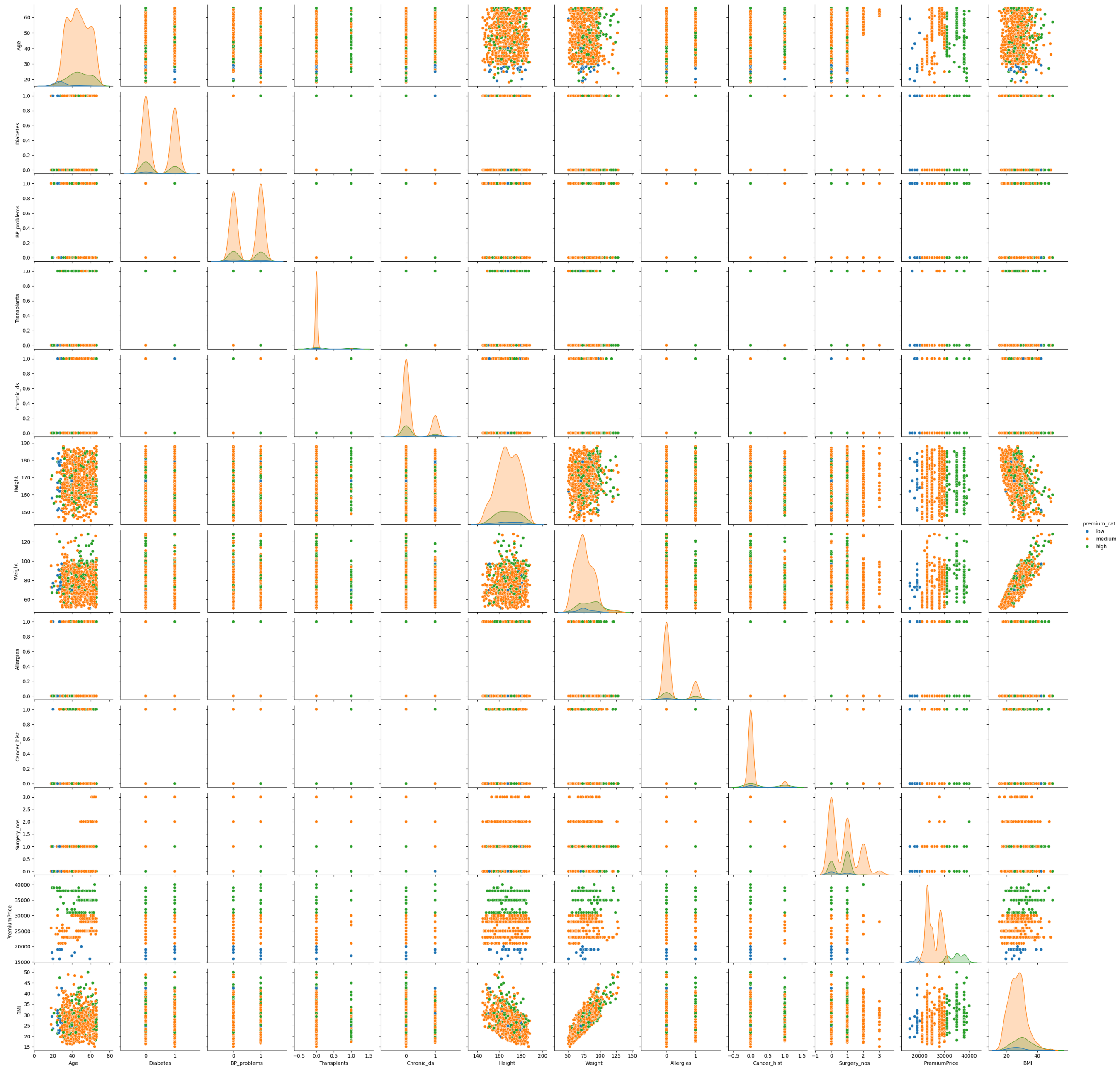
```
# Correlation analysis
df1.rename(columns={'BloodPressureProblems': 'BP_problems', 'AnyTransplants': 'Transplants', 'AnyChronicDiseases': 'Chronic_ds', 'KnownA
numerical_features = df1.select_dtypes(include=np.number).columns
plt.figure(figsize=(10,8))
sns.heatmap(df1[numerical_features].corr(), annot=True, cmap="YlGnBu")
plt.show()
```



- From the above chart it is observed that the highest correlation is found between "Weight" and "BMI" followed by "Premium Price" and "Age".
- Other notable correlations are observed between "Number of surgeries" and "Blood Pressure Problems", "Number of surgeries" and "Premium Price", "Any transplants" and "Premium Price". Like wise between "Chronic diseases" and "Premium Price"
- The above observations indicate that people with more BP problems has higher chances of having a surgery, hence also gets higher premium price as insurance.
- Similarly those who undergoes any kind of transplants or has chronic diseases tends to get higher premium price.

```
sns.pairplot(data = df1, hue = "premium_cat")
```

```
<seaborn.axisgrid.PairGrid at 0x7a3e10817a50>
```



Insights based on EDA

- Given age group ranges from 18 to 66. Of which Middle aged adults ranging from 35-55 has the maximum entries(42.89%). But on bivariate analysis against the premium price received for each age groups, it is noticed that as the age increases, price received has also increased. And that also has the maximum correlation of **0.7** with the premium price.
- Surgery numbers also seems to have a good correlation with premium pricing but intrestingly instead of maximum surgery numbers receiving the higher pricing, people with just **1 surgery and extreme obesity receives higher premium** as per the multivarieate analysis done above.
- **Blood pressure problem** is highly noticed **in "Overweight" and "Extremely obese" age groups**. And these two classes with BP problems also tends to get higher premium.
- On categorising the premium received, it is noticed that maximum premium received is between 20,000 - 30,000(642) followed by 30,000-40,000 by 120+ members.
- In the surgery numbers data, it is noticed that more than 479 people have had "0" surgeries and 372 people have only one surgery.

Outlier treatment

Age outlier

```
Q1 = np.percentile(df1["Age"],25)
Q2 = np.percentile(df1["Age"],50)
Q3 = np.percentile(df1["Age"],75)
IQR = Q3-Q1
print("Q1 :", Q1)
print("Q2 :", Q2)
print("Q3 :", Q3)
print("Age IQR :", IQR)
```

```
Q1 : 30.0
Q2 : 42.0
Q3 : 53.0
Age IQR : 23.0
```

```
Upper_whisker = Q3+ (1.5*IQR)
Lower_whisker = (max(Q1- (1.5*IQR),0))
print("Upper whisker :", Upper_whisker)
print("Lower whisker :", Lower_whisker)
```

```
Upper whisker : 87.5
Lower whisker : 0
```

Weight outlier

```
Q1 = np.percentile(df1["Weight"],25)
Q2 = np.percentile(df1["Weight"],50)
Q3 = np.percentile(df1["Weight"],75)
IQR = Q3-Q1
print("Q1 :", Q1)
print("Q2 :", Q2)
print("Q3 :", Q3)
print("Weight IQR :", IQR)
```

```
Q1 : 67.0
Q2 : 75.0
Q3 : 87.0
Weight IQR : 20.0
```

```
Upper_whisker = Q3+ (1.5*IQR)
Lower_whisker = (max(Q1- (1.5*IQR),0))
print("Upper whisker :", Upper_whisker)
print("Lower whisker :", Lower_whisker)
```

```
Upper whisker : 117.0
Lower whisker : 37.0
```

Height outlier

```
Q1 = np.percentile(df1["Height"],25)
Q2 = np.percentile(df1["Height"],50)
Q3 = np.percentile(df1["Height"],75)
IQR = Q3-Q1
print("Q1 :", Q1)
print("Q2 :", Q2)
print("Q3 :", Q3)
print("Height IQR :", IQR)
```

```
Q1 : 161.0
Q2 : 168.0
Q3 : 176.0
Height IQR : 15.0
```

```
Upper_whisker = Q3+ (1.5*IQR)
Lower_whisker = (max(Q1- (1.5*IQR),0))
print("Upper whisker :", Upper_whisker)
print("Lower whisker :", Lower_whisker)
```

```
Upper whisker : 198.5
Lower whisker : 138.5
```

BMI outlier

```
Q1 = np.percentile(df1["BMI"],25)
Q2 = np.percentile(df1["BMI"],50)
Q3 = np.percentile(df1["BMI"],75)
IQR = Q3-Q1
print("Q1 :", Q1)
print("Q2 :", Q2)
print("Q3 :", Q3)
print("BMI IQR :", IQR)
```

```
Q1 : 23.395
Q2 : 27.155
Q3 : 30.76
BMI IQR : 7.365000000000002
```

```
Upper_whisker = Q3+ (1.5*IQR)
Lower_whisker = (max(Q1- (1.5*IQR),0))
print("Upper whisker :", Upper_whisker)
print("Lower whisker :", Lower_whisker)
```

```
Upper whisker : 41.807500000000005
Lower whisker : 12.347499999999997
```

Premium outlier

```
Q1 = np.percentile(df1["PremiumPrice"],25)
Q2 = np.percentile(df1["PremiumPrice"],50)
Q3 = np.percentile(df1["PremiumPrice"],75)
IQR = Q3-Q1
print("Q1 :", Q1)
print("Q2 :", Q2)
print("Q3 :", Q3)
print("Premium price IQR :", IQR)
```

```
Q1 : 21000.0
Q2 : 23000.0
Q3 : 28000.0
Premium price IQR : 7000.0
```

```
Upper_whisker = Q3+ (1.5*IQR)
Lower_whisker = (max(Q1- (1.5*IQR),0))
print("Upper whisker :", Upper_whisker)
print("Lower whisker :", Lower_whisker)
```

```
Upper whisker : 38500.0
Lower whisker : 10500.0
```

- All the outliers present in the given data are natural variations that occur in a population dataset, hence it can be left as it is in the dataset.

Hypothesis Testing

```
from scipy.stats import norm, t, f
from scipy.stats import ttest_ind, ttest_rel, chi2_contingency
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import f, f_oneway, kruskal, shapiro, levene, kstest
```

Presence of chronic diseases lead to higher insurance premiums OR not

```
# Filtering the premium price based on the presence and absence of chronic diseases
Chronic_ds_present = df1.loc[df1["Chronic_ds"]== 1]["PremiumPrice"]
Chronic_ds_absent = df1.loc[df1["Chronic_ds"] == 0]["PremiumPrice"]
Chronic_ds_present.mean(), Chronic_ds_absent.mean()
```

```
(27112.3595505618, 23725.247524752474)
```

Assumption test

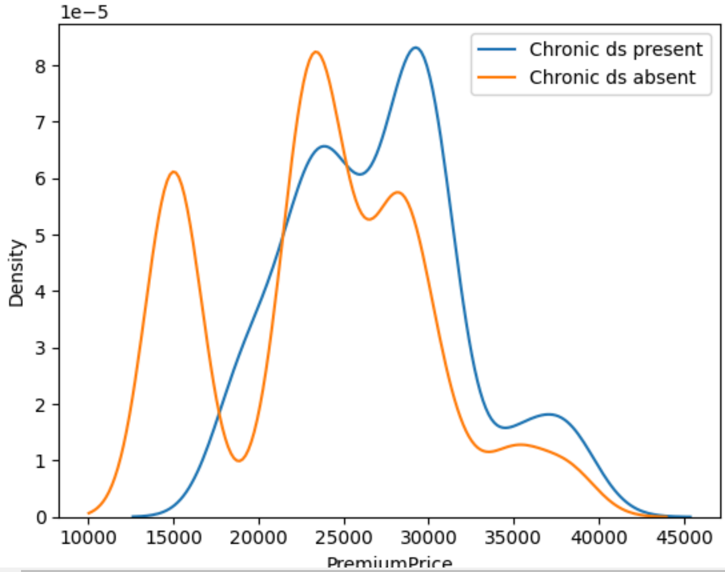
```
# Levene test
x_stat, p_value = levene(Chronic_ds_present, Chronic_ds_absent)
```

```
print("x_stat :", x_stat)
print("p_value :", p_value)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho")
else:
  print("Fail to reject Ho")
```

```
x_stat : 6.508345857175313
p_value : 0.010887284040117 98
Reject Ho
```

Visual test

```
sns.kdeplot(Chronic_ds_present)
sns.kdeplot(Chronic_ds_absent)
plt.legend(["Chronic ds present", "Chronic ds absent"])
plt.show()
```



T-test of independence (to find if presence of chronic diseases lead to higher insurance premiums OR not)

```
#Ho :  Presence of chronic diseases has no effect on higher insurance premium
#Ha : Presence of chronic diseases has effect on higher insurance premium

t_stat, p_value = ttest_ind(Chronic_ds_present, Chronic_ds_absent , alternative = "greater")
print("t_stat :", t_stat)
print("p_value :", p_value)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho : Presence of chronic diseases has effect on higher insurance premium")
else:
  print("Fail to reject Ho : Presence of chronic diseases has NO effect on higher insurance premium")
```

```
t_stat : 6.69104572734849
p_value : 1.8567068826457 41e-11
Reject Ho : Presence of chronic diseases has effect on higher insurance premium
```

Transplant operations lead to higher insurance premiums or not

```
# Filtering the premium price based on the presence and absence of transplants
Transplants_present = df1.loc[df1["Transplants"]== 1]["PremiumPrice"]
Transplants_absent = df1.loc[df1["Transplants"] == 0]["PremiumPrice"]
Transplants_present.mean(), Transplants_absent.mean()
```

```
(31763.636363636364, 23897.95918367347)
```

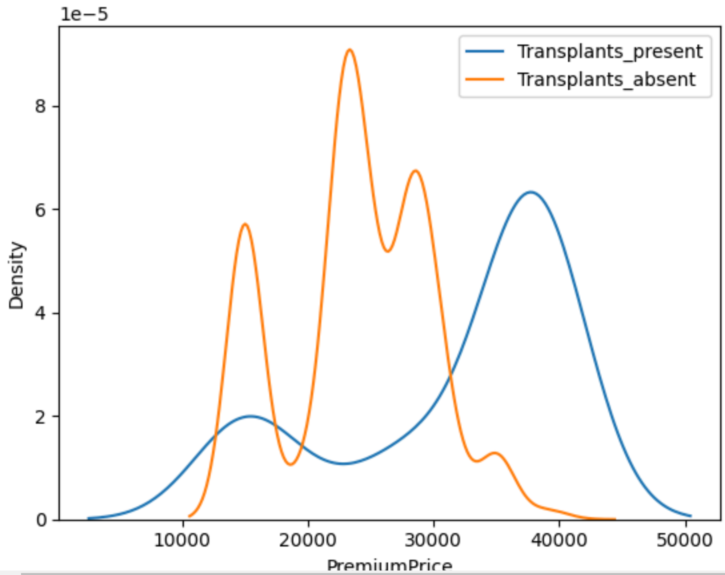
Assumption test

```
# Levene test
x_stat, p_value = levene(Transplants_present, Transplants_absent)
print("x_stat :", x_stat)
print("p_value :", p_value)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho")
else:
  print("Fail to reject Ho")
```

```
x_stat : 8.934855696615894
p_value : 0.002867336380270254
Reject Ho
```

Visual test

```
sns.kdeplot(Transplants_present)
sns.kdeplot(Transplants_absent)
plt.legend(["Transplants_present", "Transplants_absent"])
plt.show()
```



T-test of independence (to find if presence of transplants leads to higher insurance premiums OR not)

```
#Ho :  Presence of transplants has no effect on higher insurance premium
#Ha : Presence of transplants has effect on higher insurance premium

t_stat, p_value = ttest_ind(Transplants_present, Transplants_absent , alternative = "greater")
print("t_stat :", t_stat)
print("p_value :", p_value)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho : Presence of transplants has effect on higher insurance premium")
else:
  print("Fail to reject Ho : Presence of transplants has NO effect on higher insurance premium")
```

```
t_stat : 9.471654448151899
p_value : 9.893647711816386e-21
Reject Ho : Presence of transplants has effect on higher insurance premium
```

History of cancer in family has effect on insurance premium or not

```
# Filtering the premium price based on the presence and absence of cancer history in family
Cancer_present = df1.loc[df1["Cancer_hist"]== 1]["PremiumPrice"]
Cancer_absent = df1.loc[df1["Cancer_hist"] == 0]["PremiumPrice"]
Cancer_present.mean(), Cancer_absent.mean()
```

```
(25758.620689655174, 24147.126436781607)
```

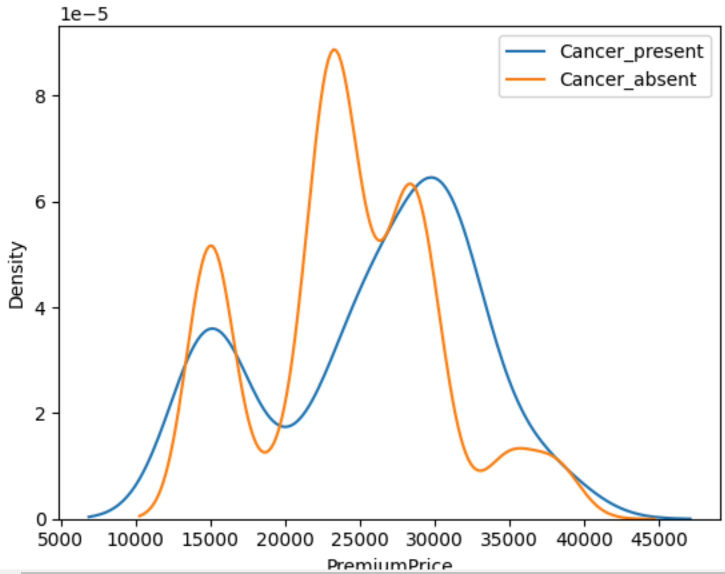
Assumption test

```
# Levene test
x_stat, p_value = levene(Cancer_present, Cancer_absent)
print("x_stat :", x_stat)
print("p_value :", p_value)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho")
else:
  print("Fail to reject Ho")
```

```
x_stat : 5.997899252289433
p_value : 0.014496453126833982
Reject Ho
```

Visual test

```
sns.kdeplot(Cancer_present)
sns.kdeplot(Cancer_absent)
plt.legend(["Cancer_present", "Cancer_absent"])
plt.show()
```



T-test of independence (to find if presence of cancer history in the family leads to higher insurance premiums OR not)

```
#Ho :  Presence of cancer history has no effect on higher insurance premium
#Ha : Presence of cancer history has effect on higher insurance premium

t_stat, p_value = ttest_ind(Cancer_present, Cancer_absent , alternative = "greater")
print("t_stat :", t_stat)
print("p_value :", p_value)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho : Presence of cancer history in family has effect on higher insurance premium")
else:
  print("Fail to reject Ho : Presence of cancer history in family has NO effect on higher insurance premium")
```

```
t_stat : 2.617041984412821
p_value : 0.004502793655223513
Reject Ho : Presence of cancer history in family has effect on higher insurance premium
```

Number of major surgeries has effect on higher insurance premium or not

```
surgery_0 = df1[df1["Surgery_nos"] == 0]["PremiumPrice"]
surgery_1 = df1[df1["Surgery_nos"] == 1]["PremiumPrice"]
surgery_2 = df1[df1["Surgery_nos"] == 2]["PremiumPrice"]
surgery_3 = df1[df1["Surgery_nos"] == 3]["PremiumPrice"]
surgery_0.mean(), surgery_1.mean(), surgery_2.mean(), surgery_3.mean()
```

```
(22968.684759916494, 24741.935483870966, 28084.03361344538, 28000.0)
```

Assumption test

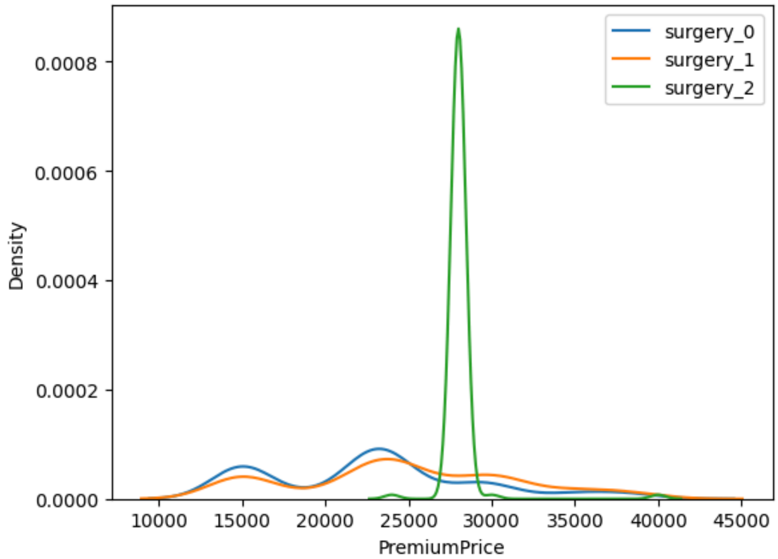
```
# kruskal test
f_stat, p_value = kruskal(surgery_0, surgery_1, surgery_2, surgery_3)
print("f_stat :", f_stat)
print("p_value :", p_value)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho : Atleast one of them is different")
else:
  print("Fail to reject Ho")
```

```
f_stat : 93.81277073618764
p_value : 3.3223412749827346e-20
Reject Ho : Atleast one of them is different
```

Visual test

```
sns.kdeplot(surgery_0)
sns.kdeplot(surgery_1)
sns.kdeplot(surgery_2)
sns.kdeplot(surgery_3)
plt.legend(["surgery_0", "surgery_1", "surgery_2", "surgery_3"])
plt.show()
```

```
<ipython-input-303-4fba6d252631>:4: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=False` to di
  sns.kdeplot(surgery_3)
```



- Visual test clearly shows the premium price varies based on the no. of surgeries performed.

ANOVA test (to find if the no. of major surgeries performed has effect over premium price or not)

```
# Ho: Premium price insurance is similar for different no. of surgeries
# Ha: Premium price insurance differs for different no. of surgeries

f_stat, p_value = f_oneway(surgery_0, surgery_1, surgery_2, surgery_3)
print("f_stat :", f_stat)
print("p_value :", p_value)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho : Premium price insurance differs for different no. of surgeries")
else:
  print("Fail to reject Ho: Premium price insurance is similar for different no. of surgeries")
```

```
f_stat : 26.13539359740762
p_value : 2.8711631377228097e-16
Reject Ho : Premium price insurance differs for different no. of surgeries
```

Number of surgeries are dependent on Blood pressure problems or not

```
bp_surgeries = pd.crosstab(df1["BP_problems"], df1["Surgery_nos"], margins= True)
bp_surgeries
```

Surgery_nos	0	1	2	3	All
BP_problems					
0	315	172	26	11	524
1	164	200	93	5	462
All	479	372	119	16	986

Next steps: Generate code with bp_surgeries View recommended plots New interactive sheet

Chi-square (Test of Independence)

```
# Ho : Number of surgeries is not dependent on Blood pressure problems
# Ha : Number of surgeries is dependent on Blood pressure problems

chi_stat, p_value, dof, expected = chi2_contingency(bp_surgeries)
print("chi_stat :", f_stat)
print("p_value :", p_value)
print("dof :", dof)
print("Expected :", expected)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho: Number of surgeries is dependent on Blood pressure problems")
else:
  print("Fail to reject Ho: Number of surgeries is not dependent on Blood pressure problems")
```

```
chi_stat : 26.13539359740762
p_value : 2.8395630390366105e-15
dof : 8
Expected : [[254.55983773 197.69574037  63.24137931   8.5030426  524.        ]
 [224.44016227 174.30425963  55.75862069   7.4969574  462.        ]
 [479.         372.         119.          16.         986.        ]]
Reject Ho: Number of surgeries is dependent on Blood pressure problems
```

Chronic diseases are dependent on history of cancer in the family or not

```
chronic_cancer = pd.crosstab(df1["Chronic_ds"], df1["Cancer_hist"], margins= True)
chronic_cancer
```

Cancer_hist	0	1	All
Chronic_ds			
0	714	94	808
1	156	22	178
All	870	116	986

Next steps: Generate code with chronic_cancer View recommended plots New interactive sheet

Chi-square (Test of Independence)

```
# Ho : Chronic diseases are not dependent on history of cancer in family
# Ha : Chronic diseases is dependent on history of cancer in family

chi_stat, p_value, dof, expected = chi2_contingency(chronic_cancer)
print("chi_stat :", f_stat)
print("p_value :", p_value)
print("dof :", dof)
print("Expected :", expected)
alpha = 0.05

if p_value < alpha:
  print("Reject Ho: Chronic diseases are dependent on history of cancer in family")
else:
  print("Fail to reject Ho: Chronic diseases are not dependent on history of cancer in family")
```

```
chi_stat : 26.13539359740762
p_value : 0.9993314302405011
dof : 4
Expected : [[712.94117647  95.05882353 808.        ]
 [157.05882353  20.94117647 178.        ]
 [870.         116.         986.        ]]
Fail to reject Ho: Chronic diseases are not dependent on history of cancer in family
```

ML Modeling -----> (Block 3)

Since the Target variable is already provided and the data type is continuous, we're choosing Linear regression as the ML model and do some analysis on it.

Regression Analysis

```
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
```

Scaling the data using min max scaler to study about Linear Regression

```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
X = pd.DataFrame(min_max_scaler.fit_transform(df1[["Age","Diabetes","BP_problems", "Transplants", "Chronic_ds", "Height", "Weight", "Al
                 columns= ["Age","Diabetes","BP_problems", "Transplants", "Chronic_ds", "Height", "Weight", "Allergies", "Cancer_hist",
X
```

	Age	Diabetes	BP_problems	Transplants	Chronic_ds	Height	Weight	Allergies	Cancer_hist	Surgery_nos	BMI
0	0.562500	0.0	0.0	0.0	0.0	0.232558	0.074074	0.0	0.0	0.000000	0.245982
1	0.875000	1.0	0.0	0.0	0.0	0.813953	0.271605	0.0	0.0	0.000000	0.211538
2	0.375000	1.0	1.0	0.0	0.0	0.302326	0.098765	0.0	0.0	0.333333	0.243111
3	0.708333	1.0	1.0	0.0	1.0	0.883721	0.518519	0.0	0.0	0.666667	0.361940
4	0.416667	0.0	0.0	0.0	1.0	0.488372	0.456790	0.0	0.0	0.333333	0.481343
...	...	...	...	...	...	...	...	...	...	...	...
981	0.000000	0.0	0.0	0.0	0.0	0.558140	0.197531	0.0	0.0	0.000000	0.238232
982	0.958333	1.0	1.0	0.0	0.0	0.186047	0.234568	0.0	0.0	1.000000	0.423077
983	0.791667	0.0	1.0	0.0	0.0	0.232558	0.246914	0.0	0.0	0.333333	0.413031
984	0.604167	1.0	1.0	0.0	0.0	0.302326	0.271605	1.0	0.0	0.333333	0.404133
985	0.062500	0.0	0.0	0.0	0.0	0.302326	0.296296	1.0	0.0	0.333333	0.427095

986 rows × 11 columns

Next steps: Generate code with X View recommended plots New interactive sheet

```
y = df1["PremiumPrice"]
```

Linear regression

Dividing the data into 80% train data and 20% test data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state= 10)


y_train = np.array(y_train)


X_sm = sm.add_constant(X_train)
model = sm.OLS(y_train, X_sm)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.635
Model:                            OLS   Adj. R-squared:                  0.630
Method:                 Least Squares   F-statistic:                     122.9
Date:                Thu, 06 Feb 2025   Prob (F-statistic):          1.14e-161
Time:                        09:08:38   Log-Likelihood:                -7607.9
No. Observations:                 788   AIC:                         1.524e+04
Df Residuals:                     776   BIC:                         1.530e+04
Df Model:                          11
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        1.498e+04   2341.572      6.399      0.000    1.04e+04    1.96e+04
Age          1.561e+04    530.543     29.429      0.000    1.46e+04    1.67e+04
Diabetes     -399.0830    285.584     -1.397      0.163    -959.693     161.527
BP_problems    91.5296    285.107      0.321      0.748    -468.143     651.202
Transplants  7960.6944    586.185     13.581      0.000    6809.998    9111.391
Chronic_ds   2780.7749    350.763      7.928      0.000    2092.219    3469.331
Height      -1071.6036   3043.789     -0.352      0.725   -7046.641    4903.433
Weight       7526.1440   6110.750      1.232      0.218   -4469.416    1.95e+04
Allergies     265.3290    336.911      0.788      0.431    -396.035     926.693
Cancer_hist  2226.9458    433.960      5.132      0.000    1375.070    3078.821
Surgery_nos -1623.9827    635.533     -2.555      0.011   -2871.550    -376.415
BMI         -2818.1260   7384.821     -0.382      0.703   -1.73e+04    1.17e+04
==============================================================================
Omnibus:                      217.772   Durbin-Watson:                   2.032
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1371.457
Skew:                           1.091   Prob(JB):                    1.56e-298
Kurtosis:                       9.084   Cond. No.                         117.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The StatsModel gives us the R squared and adjusted R squared values directly.

R-squared: 0.635

Adj. R-squared: 0.630

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

```
lr.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
lr.coef_
```

```
array([15613.54531989,  -399.08295285,    91.52956222,  7960.69438801,
        2780.77485883, -1071.60361195,  7526.14403107,   265.32903178,
        2226.9457893 , -1623.98268833, -2818.12596294])
```

```
lr.intercept_
```

```
14984.098697732361
```

```
y_pred1 = lr.predict(X_test)
y_pred1
```

```
array([21931.26175056, 17558.99168587, 20727.57104222, 26947.92418551,
       18109.19982248, 29148.69656869, 18814.39238993, 35059.40558843,
       18026.06870122, 26908.6486378 , 23111.442342  , 23768.99003413,
       25031.03503668, 24390.70392518, 23451.54221134, 29687.69821232,
       17215.34331627, 19680.83746626, 25320.5702365 , 18570.25940199,
       21585.18811352, 24042.45406489, 28910.81896992, 34462.55150937,
       23976.97629131, 27738.92410345, 25883.20235013, 30296.9652075 ,
       29015.55211228, 22435.29668989, 27734.26926719, 35498.99812922,
       29270.00765319, 30911.13011421, 24846.27779612, 23099.73072782,
       24912.14450448, 26702.42915346, 16818.64830399, 26105.29770224,
       28234.74363625, 28614.74340628, 29091.16515048, 20770.42349901,
       25087.4578149 , 30955.3043678 , 21758.63719642, 25897.03794932,
       29954.36417553, 27793.44595893, 30522.78723721, 19605.80459144,
       29047.96966618, 27915.41323062, 17192.70404376, 40690.43767601,
       20003.72770436, 15948.78558401, 31164.70395444, 20483.93799212,
       28293.72673033, 30322.90818069, 23850.34076921, 19876.65833719,
       22679.23924697, 27884.20669748, 27580.59516324, 32602.43329807,
       27861.76086845, 29101.85927035, 27908.72104846, 17868.0936554 ,
       28193.2197423 , 17603.60132373, 24782.52898271, 24403.04403588,
```

```
       31551.94364983, 27392.91599156, 25582.20794059, 27300.36695403,
       34647.99867282, 20621.58504299, 15726.61672101, 15787.23224907,
       24178.09075128, 38183.12332278, 27201.76255689, 32685.02523321,
       28822.62825605, 19719.73509947, 33061.53668378, 20451.29748639,
       15281.3472198 , 23556.86980756, 32290.24549975, 29402.65960118,
       22608.80670221, 27520.81680986, 30406.41942608, 16273.16628703,
       22372.6137952 , 29870.57742338, 22972.9403369 , 29924.84484139,
       17033.58166999, 24105.33473727, 17741.11988437, 19213.71054705,
       26952.55319403, 22594.9681091 , 23179.16542364, 27119.28912151,
       28855.24616783, 28519.4380173 , 21356.83699991, 28468.15605402,
       22279.07132672, 15175.83427544, 26773.42900626, 25614.87481638,
       25364.17038532, 28664.82714482, 19060.69248556, 25743.67528897,
       26202.07737291, 22332.4553453 , 26582.28516064, 17004.45635324,
       24237.79289277, 19361.06498665, 25007.45779366, 20812.77017961,
       24840.22346172, 28763.40423226, 25701.34803316, 26901.51200328,
       19939.32796316, 20783.91131815, 29460.56167691, 23179.15894241,
       18610.85533053, 22499.17850704, 24503.15876636, 30515.28492383,
       23586.13441208, 18060.74692458, 19367.57848158, 24991.06538287,
       23290.55612074, 19194.80913041, 19618.31633389, 23618.6916141 ,
       19302.42572815, 31233.56273832, 28705.16528207, 33190.11931418,
       21040.10985992, 23970.03716124, 17457.83288392, 25438.38608305,
       17127.32029469, 22029.81439817, 21112.85438757, 16896.464183  ,
       24921.64625237, 17861.09508158, 29312.62328696, 20995.32232451,
       23976.57900562, 28731.33335359, 17446.01576021, 25770.76371068,
       14775.84695537, 19618.8689526 , 14984.91922853, 23455.95953904,
       22606.98908617, 22128.02012693, 22040.91998453, 22375.8234624 ,
       26234.13327833, 27865.72668369, 31633.64700668, 15201.27253927,
       28583.39713487, 35069.37450963, 24351.53657929, 16943.24304226,
       25569.47317053, 20334.42163635, 24132.38005775, 21661.48593549,
       27065.432658  , 21317.82748191, 20443.23825425, 22735.76676195,
       27532.0646284 , 25545.38837047])
```

```
y_pred2 = lr.predict(X_train)
y_pred2
```

```
16013.51722462, 18783.72090057, 16365.78524798, 27937.05732195,
25278.21719282, 27710.46026511, 23915.91058459, 25601.64662697,
27905.11229842, 29339.42751145, 18135.44419157, 31331.40423669,
21868.70681714, 23560.18383233, 29109.84010287, 17251.05401693])
```

```
from sklearn.metrics import r2_score
```

```
r2_score(y_test, lr.predict(X_test))
```

```
0.6708087436472316
```

```
r2_score(y_train, lr.predict(X_train))
```

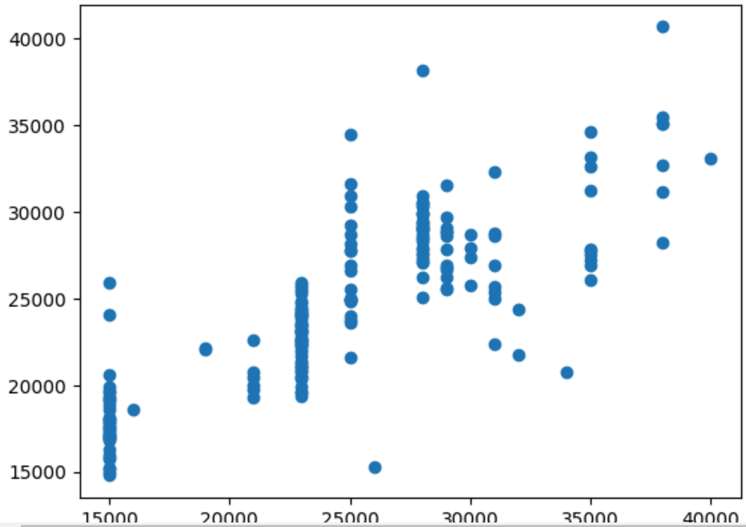
```
0.6353750141727392
```

Assumptions of Linear Regression

Linearity

```
plt.scatter(y_test, y_pred1)
```

```
<matplotlib.collections.PathCollection at 0x7a3e08bb6210>
```



- Correlation is not weak/zero. It looks positive, linearity exists

Variance Inflation Factor (To check multicollinearity)

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
X_sm
```

	const	Age	Diabetes	BP_problems	Transplants	Chronic_ds	Height	Weight	Allergies	Cancer_hist	Surgery_nos	BM
12	1.0	0.125000	0.0	0.0	0.0	0.0	0.767442	0.074074	1.0	0.0	0.333333	0.08122
578	1.0	0.250000	0.0	1.0	0.0	0.0	0.488372	0.518519	0.0	0.0	0.000000	0.53358
441	1.0	0.958333	0.0	1.0	0.0	0.0	0.674419	0.160494	0.0	0.0	0.333333	0.17164
698	1.0	0.791667	1.0	0.0	0.0	0.0	0.511628	0.469136	1.0	0.0	0.000000	0.48076
773	1.0	0.645833	1.0	0.0	0.0	0.0	0.325581	0.135802	0.0	0.0	0.000000	0.26865
...	...	...	...	...	...	...	...	...	...	...	...	.
369	1.0	0.375000	0.0	1.0	1.0	1.0	0.441860	0.086420	0.0	0.0	0.000000	0.18369
320	1.0	0.208333	0.0	0.0	0.0	1.0	0.790698	0.320988	0.0	0.0	0.000000	0.25459
527	1.0	0.500000	0.0	0.0	0.0	0.0	0.930233	0.407407	0.0	0.0	0.333333	0.26923
125	1.0	0.270833	1.0	0.0	1.0	0.0	0.976744	0.543210	1.0	0.0	0.000000	0.34471
265	1.0	0.125000	1.0	0.0	0.0	0.0	0.325581	0.407407	0.0	0.0	0.333333	0.51865

788 rows × 12 columns

Next steps: Generate code with X_sm | View recommended plots | New interactive sheet

```
vif = pd.DataFrame()
vif['features'] =X_sm.columns
vif['VIF'] = [variance_inflation_factor(X_sm, i) for i in range(X_sm.shape[1])]
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

	features	VIF
0	const	298.83
11	BMI	81.99
7	Weight	61.74
6	Height	28.61
10	Surgery_nos	1.34
1	Age	1.30
3	BP_problems	1.10
9	Cancer_hist	1.09
2	Diabetes	1.08
8	Allergies	1.03
5	Chronic_ds	1.02
4	Transplants	1.01

Next steps: Generate code with vif | View recommended plots | New interactive sheet

Height, Weight and BMI(which is derived from height and weight) are the features with VIF more than 10. Eventhough Height and Weight as individual features can be dropped to avoid multicollinearity. BMI is a very important factor to decide health in many cases, hence cannot be dropped.

Performing VIF again after dropping columns "Height" and "Weight"

```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
X1 = pd.DataFrame(min_max_scaler.fit_transform(df1[["Age","Diabetes","BP_problems", "Transplants", "Chronic_ds",  "Allergies", "Cancer_h
                 columns= ["Age","Diabetes","BP_problems", "Transplants", "Chronic_ds", "Allergies", "Cancer_hist", "Surgery_nos", "BMI'
X1
```

	Age	Diabetes	BP_problems	Transplants	Chronic_ds	Allergies	Cancer_hist	Surgery_nos	BMI
0	0.562500	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.245982
1	0.875000	1.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.211538
2	0.375000	1.0	1.0	0.0	0.0	0.0	0.0	0.333333	0.243111
3	0.708333	1.0	1.0	0.0	1.0	0.0	0.0	0.666667	0.361940
4	0.416667	0.0	0.0	0.0	1.0	0.0	0.0	0.333333	0.481343
...	...	...	...	...	...	...	...	...	...
981	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.238232
982	0.958333	1.0	1.0	0.0	0.0	0.0	0.0	1.000000	0.423077
983	0.791667	0.0	1.0	0.0	0.0	0.0	0.0	0.333333	0.413031
984	0.604167	1.0	1.0	0.0	0.0	1.0	0.0	0.333333	0.404133
985	0.062500	0.0	0.0	0.0	0.0	1.0	0.0	0.333333	0.427095

986 rows × 9 columns

Next steps: Generate code with X1 | View recommended plots | New interactive sheet

Training the new data

```
from sklearn.model_selection import train_test_split

X1_train, X1_test, y_train, y_test = train_test_split(X1,y,test_size=0.2,random_state= 10)
```

```
X1_sm = sm.add_constant(X1_train)
model = sm.OLS(y_train, X1_sm)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           PremiumPrice   R-squared:                       0.627
Model:                            OLS   Adj. R-squared:                  0.623
Method:                 Least Squares   F-statistic:                     145.6
Date:                Thu, 06 Feb 2025   Prob (F-statistic):          2.97e-160
Time:                        09:08:39   Log-Likelihood:                -7616.3
No. Observations:                 788   AIC:                         1.525e+04
Df Residuals:                     778   BIC:                         1.530e+04
Df Model:                           9
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        1.427e+04    436.165     32.726      0.000    1.34e+04    1.51e+04
Age          1.569e+04    535.148     29.323      0.000    1.46e+04    1.67e+04
Diabetes     -427.0372    288.124     -1.482      0.139    -992.631     138.556
BP_problems    10.9015    286.419      0.038      0.970    -551.345     573.148
Transplants  7819.4584    590.361     13.245      0.000    6660.570    8978.347
Chronic_ds   2822.8134    353.805      7.978      0.000    2128.288    3517.338
Allergies     275.0208    340.036      0.809      0.419    -392.475     942.517
Cancer_hist  2258.1256    437.289      5.164      0.000    1399.720    3116.531
Surgery_nos -1541.8168    641.212     -2.405      0.016   -2800.528    -283.106
BMI          4341.8806    826.562      5.253      0.000    2719.324    5964.437
==============================================================================
Omnibus:                      211.444   Durbin-Watson:                   2.025
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1218.542
Skew:                           1.082   Prob(JB):                    2.49e-265
Kurtosis:                       8.695   Cond. No.                         9.25
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
lr.fit(X1_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
lr.coef_
```

```
array([ 1.56923371e+04, -4.27037217e+02,  1.09014909e+01,  7.81945840e+03,
        2.82281337e+03,  2.75020785e+02,  2.25812562e+03, -1.54181675e+03,
        4.34188058e+03])
```

```
lr.intercept_
```

```
14273.746181982964
```

```
y_pred3 = lr.predict(X1_test)
y_pred3
```

```
array([21423.71075692, 17479.35593895, 20572.85798225, 26069.65521311,
       18343.82142407, 28652.33827551, 17958.10106677, 34092.61135043,
       19228.72340883, 27368.29058242, 22077.20749555, 23803.60172187,
       25301.12111591, 23992.20858683, 23935.8147561 , 30233.23405405,
       16908.17867739, 19463.76882594, 25088.01483756, 18841.14901355,
       21329.83908927, 24112.49467118, 29122.75914907, 34583.89296924,
       23919.4882516 , 26577.66430937, 25503.281346  , 30510.97735775,
       28742.39695165, 22237.18269198, 27624.89178061, 35380.17951745,
       29705.33192326, 31291.2601288 , 24553.5080688 , 23592.88367866,
       25311.90045907, 26272.68036318, 16507.4084738 , 26044.28688875,
       28431.47568554, 28126.69366175, 28444.40825934, 20776.63127801,
       24441.04815643, 30875.91942865, 21743.92831315, 25642.27689182,
       31227.767805  , 27768.63678028, 31526.33049208, 19737.28269865,
       29907.49191999, 28869.3741833 , 17017.52861655, 39967.34740362,
       20801.44491158, 15671.93658891, 31652.05802003, 21435.31651868,
       27913.51691023, 30063.71069836, 23523.61588825, 20092.79420396,
       22417.49312304, 27348.09818941, 27604.06678903, 32110.02755212,
       26914.23875736, 29025.21796146, 27102.95821054, 17761.91003952,
       28318.91748051, 17769.80862117, 24708.01351089, 24060.1058107 ,
       31399.26817125, 27011.14264998, 24892.36073615, 27869.79814272,
       34080.02171953, 20325.83300738, 14729.48416996, 15918.34989243,
       23724.68317221, 37607.11619658, 27002.23554335, 32408.89941358,
       28495.97863038, 19783.27142222, 33070.09710944, 20387.69059982,
       15153.69421185, 22659.13930947, 32693.68858566, 29092.87368612,
       22724.39353127, 26637.08204926, 30184.96112265, 16517.45861946,
       22932.49437936, 29933.34056393, 22848.07445908, 30238.13442291,
       16992.69959126, 24088.56352335, 18337.2600427 , 18882.63361702,
       26380.26680084, 22537.40322929, 23344.88303682, 27620.24154979,
       28863.22308282, 28925.58251119, 21615.61545989, 29095.06588978,
       22193.84758416, 15601.34466674, 25946.22926181, 24200.82690122,
       25113.75988153, 29278.29920315, 18564.596035  , 26814.57212528,
       26291.05926034, 22673.75961453, 26647.10803272, 16968.48441137,
       24091.86778341, 18766.80939377, 25860.99244297, 21825.01361935,
       26023.42021925, 29265.24831608, 25719.32963118, 27723.68267287,
       19922.87595911, 20821.38016444, 29581.61884292, 23202.05643531,
       18692.45223303, 23402.71556238, 24066.5328149 , 29401.09508866,
       23037.70756394, 17910.68165463, 19683.98295262, 25178.04763626,
       24155.46998718, 19352.10061256, 18827.8968914 , 23815.2949159 ,
       20279.2727059 , 30938.61340905, 28632.37163126, 33762.11723391,
       21608.69017197, 23568.17773768, 16987.18882156, 24297.23594429,
       16991.26223141, 22101.6733539 , 21749.79849216, 16244.14929531,
       25962.92744827, 18176.08102941, 29034.52194962, 22004.06108934,
       24174.11945175, 29052.96062465, 17255.11382178, 25859.13256861,
       15537.42784442, 19724.30371633, 15203.38112105, 23661.50214381,
       22670.7886183 , 21857.06191216, 22742.47646139, 22068.9742072 ,
       26221.9928604 , 27481.1645459 , 32295.02389306, 15308.12071829,
       28412.5008158 , 34237.333302  , 24109.64527421, 16283.84216727,
       25910.9577741 , 20241.07620636, 23872.87613352, 21545.88577741,
       26986.87838128, 21604.52579623, 20633.36630673, 22550.3177311 ,
       27726.59264755, 25761.83532098])
```

```
y_pred4 = lr.predict(X1_train)
y_pred4
```

```
30884.17283932, 29781.60685578, 28881.13924883, 18515.46685654,
28063.69793213, 27415.23720802, 28896.135418  , 24793.63641519,
24356.91869341, 25639.67549301, 21500.18321215, 30123.54983761,
31424.36039271, 18807.83089472, 21479.37652594, 39136.5039606 ,
20820.04296415, 24088.14447198, 15639.97343532, 29840.6943862 ,
26021.66824083, 26919.46293844, 20399.02226053, 27680.2748503 ,
24135.99192997, 17520.67729624, 18210.55441671, 26396.95582688,
19603.33658674, 28472.12433551, 21705.47078109, 21375.95916045,
25504.1172995 , 28086.03165829, 18492.5335963 , 22285.23297487,
21915.14847762, 24769.05520474, 26184.26022778, 33880.65757441,
25583.6144054 , 25747.19512696, 23077.53565214, 25935.96465716,
30312.32247113, 16599.20414539, 29261.59241216, 29228.68457422,
25211.02946971, 29397.03494653, 17182.42194679, 24682.17279835,
19818.6314106 , 15926.66325221, 19455.94364442, 24928.28783848,
24104.90986676, 31776.75578865, 17817.16552766, 28101.03338562,
21340.14624967, 17659.41959343, 30584.64735872, 31553.54521974,
22717.69693366, 25370.47976436, 29226.27177228, 21994.00901886,
28043.70660409, 30823.86458203, 25209.54459973, 16728.39794209,
25442.68589214, 23254.08366076, 18841.50788244, 24628.18770679,
20316.6552401 , 25570.93687147, 27935.24735358, 18639.6693247 ,
31172.87938568, 33121.87429856, 22614.6958417 , 30160.95245995,
21146.65183514, 16762.99545197, 22703.2910772 , 32484.30700121,
28049.39601169, 28129.21928718, 27646.2322408 , 22739.55382278,
17205.29972721, 21056.99873308, 28655.98560398, 25119.68113754,
34931.67526251, 22281.66550824, 19985.70688909, 19988.4744225 ,
29045.59540819, 17731.1798526 , 22846.11421973, 26860.85403807,
32392.96616789, 22610.53616861, 25653.9332265 , 19823.98237867,
30557.76846938, 28335.66540708, 31020.97276998, 25483.48141725,
34328.92329547, 30017.71108031, 20392.98995048, 24702.34259834,
16238.31750981, 19114.48482308, 16430.55222306, 27543.56608474,
25269.14237142, 27816.79607898, 25033.0801598 , 25620.58516331,
27818.4082081 , 29549.55262562, 18873.15621829, 31609.13584713,
21471.20633862, 22774.94364377, 27687.92359482, 17546.25770636])
```

```
vif = pd.DataFrame()
vif['features'] =X1_sm.columns
vif['VIF'] = [variance_inflation_factor(X1_sm, i) for i in range(X1_sm.shape[1])]
vif['VIF'] = round(vif['VIF'],2)
vif = vif.sort_values(by = 'VIF', ascending = False)
vif
```

	features	VIF
0	const	10.18
8	Surgery_nos	1.34
1	Age	1.29
3	BP_problems	1.09
2	Diabetes	1.08
7	Cancer_hist	1.08
6	Allergies	1.03
5	Chronic_ds	1.02
9	BMI	1.01
4	Transplants	1.00

Next steps: Generate code with vif View recommended plots New interactive sheet

- After dropping Height and Weight, VIF of BMI comes down to 1 and now all the given features shows **NO MULTICOLLINEARITY**. Therefore, we don't need to drop any features further.

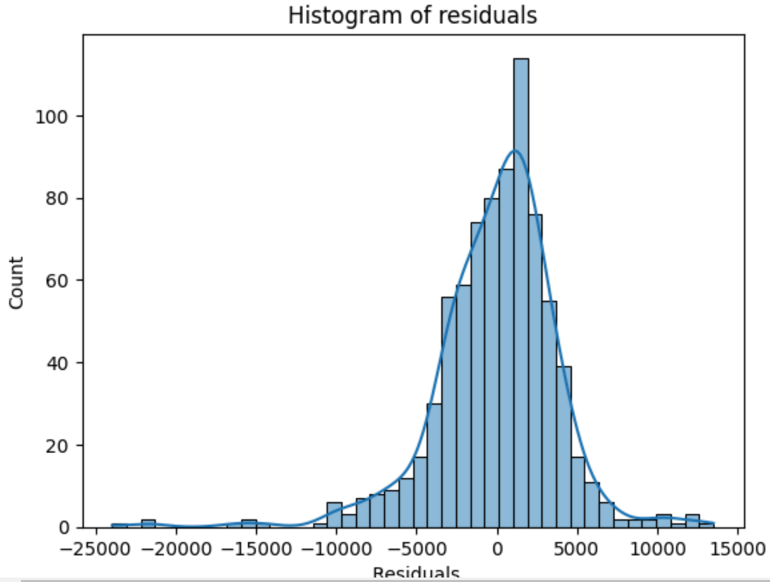
Normality of residuals

```
y_hat = results.predict(X1_sm)
```

```
errors = y_hat - y_train
```

```
sns.histplot(errors, kde=True)
plt.xlabel(" Residuals")
plt.title("Histogram of residuals")
```

```
Text(0.5, 1.0, 'Histogram of residuals')
```




```
from scipy import stats
result = stats.shapiro(errors)
result.statistic
```

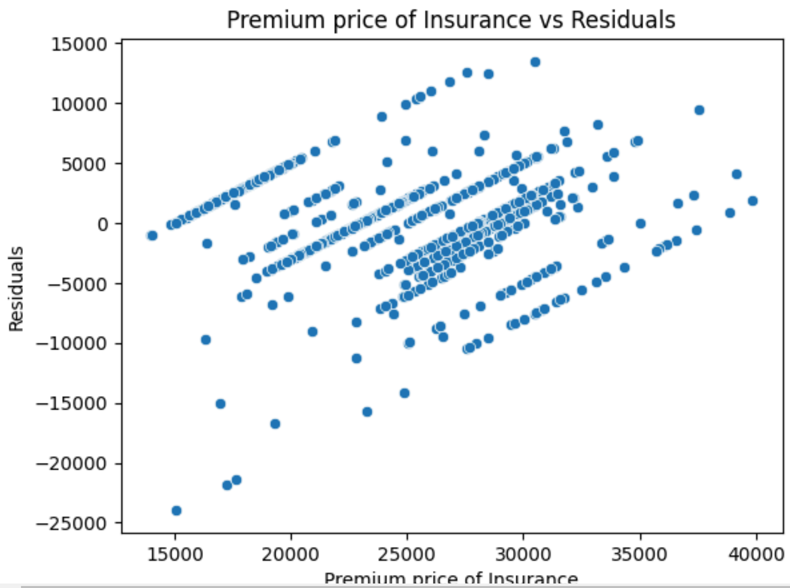
```
0.9256164367658695
```

- Since the value is closer to 1, it means the distribution is normal

Test for Homoscedasticity

```
sns.scatterplot(x=y_pred4,y=errors)
plt.xlabel("Premium price of Insurance")
plt.ylabel("Residuals")
plt.title("Premium price of Insurance vs Residuals")
```

```
Text(0.5, 1.0, 'Premium price of Insurance vs Residuals')
```



- Null Hypothesis: Heteroscedasticity is not present.
- Alternate Hypothesis: Heteroscedasticity is present.

```
# Performing the Goldfeld-Quandt test to check for Homoscedasticity -
from statsmodels.compat import lzip
import statsmodels.stats.api as sms

name= ['F statistic', 'p-value']
```

```
test = sms.het_goldfeldquandt(y_train, X1_sm)
lzip(name, test)
```

```
[('F statistic', 1.1689936277360775), ('p-value', 0.06324208336170273)]
```

- Since p-value > alpha(0.05) - Homoscedaticity is present

Auto-correlation

- There is no auto-correlation since the features height and weight have been dropped and all the other features are independent of each other except the target column.

Insights on Assumptions of Linear regression model

- Linear function EXISTS.
- Multicollinearity was earliar found between Height, Weight and BMI. so Height and weight were dropped and the Linear regression analysis was performed again. After that, **No multicollinearity** was observed as per the VIF score. Since none of the major features have VIF score of more than 5.
- Errors are normally distributed as per the histogram of residuals.
- No heteroscedasticity has been observed as per the Goldfeld-Quandt test conducted.
- No auto correlation has been observed since all the datas are independent of each other and they all have a linear relationship with the "dependent variable - Premium price of Insurance".

Regularisation (To avoid overfitting of the data)

Types of regularisation : Ridge(L2) and Lasso(L1)

```
from sklearn.linear_model import Ridge, Lasso
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
```

```
# Splitting the data into train and test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 12)
```

```
# Transform the features into polynomial features
degree = 2
scaler = StandardScaler()
polyreg_scaled = make_pipeline(PolynomialFeatures(degree), scaler, LinearRegression())
polyreg_scaled.fit(X_train, y_train)
```

```
Pipeline
  PolynomialFeatures
  StandardScaler
  LinearRegression
```

```
polyreg_scaled.steps[-1][1].coef_
```

```
array([-1.55208003e-09,  8.11732060e+03,  2.08897469e+02, -1.51917977e+03,
        1.96964872e+03,  1.93905808e+03,  1.76929720e+04, -2.50894539e+04,
       -4.63253335e+02,  2.68621379e+02, -2.61765531e+02,  3.16326701e+04,
       -5.07759010e+03,  7.84070738e+02, -8.45537070e+02,  6.74548466e+02,
       -1.24023108e+02, -5.38560952e+02,  3.15953097e+03,  3.10584341e+02,
        1.53942756e+02,  4.38376768e+02, -9.70453262e+02,  2.08897469e+02,
       -5.89316623e+01, -3.35341506e+02, -1.09196038e+01, -7.00765796e+02,
        1.13716626e+03,  2.87722878e+01,  5.52124130e+01, -6.05180044e+01,
       -1.66765505e+03, -1.51917977e+03, -2.36468457e+02,  7.85093077e+00,
        3.02561059e+03, -4.11905485e+03, -2.29008218e+02,  2.04812286e+02,
       -1.23018295e+02,  5.54090589e+03,  1.96964872e+03,  3.34520530e+02,
       -1.28094908e+03,  2.48891192e+03, -6.40615643e-01,  3.44697987e+02,
       -7.62338120e+02, -3.50942800e+03,  1.93905808e+03, -2.50077196e+03,
        3.23778643e+03,  3.59861052e+00,  1.74610880e+02, -3.40321630e+02,
       -3.89916484e+03, -1.40201386e+04,  2.94658449e+04,  6.61969476e+01,
       -4.77044706e+02,  1.71989179e+03, -2.80098018e+04, -3.21995472e+04,
        3.72984403e+02,  2.52205955e+02, -3.28588608e+03,  7.84017722e+04,
```

```
        -4.63253335e+02,  5.23638518e+02, -4.12319820e+00,  4.82058579e+02,
         2.68621379e+02, -6.10116208e+02,  1.95683036e+02, -9.92876268e+02,
         2.63177655e+03, -5.29796471e+04])
```

```
polyreg_scaled.score(X_train, y_train)
```

```
0.7396094160635869
```

```
polyreg_scaled.score(X_test, y_test)
```

```
0.68591454539357
```

- This shows that at polynomial degree of 2, "GOOD FIT" has been observed between the train and test data.
- At higher degree, the difference between train and test scores are high.

Ridge(L2 regularisation)

```
degree = 2
scaler = StandardScaler()
polyreg_scaled = make_pipeline(PolynomialFeatures(degree), scaler, Ridge())
polyreg_scaled.fit(X_train, y_train)
```

```
Pipeline
  PolynomialFeatures
  StandardScaler
  Ridge
```

```
polyreg_scaled.steps[-1][1].coef_
```

```
array([ 0.00000000e+00,  5.85822037e+03, -1.99085181e+02, -3.20466164e+02,
         1.07525485e+03,  8.53851251e+02, -9.89046535e+02, -2.99275865e+02,
        -5.28419990e+02,  6.15070419e+01,  1.10272555e+03,  3.86971258e+01,
        -4.83438916e+03,  7.67530861e+02, -8.23088767e+02,  6.61204624e+02,
        -1.22054780e+02,  1.39813290e+03,  6.31562671e+02,  2.96832014e+02,
         1.27735213e+02,  3.83350079e+02,  2.18285256e+03, -1.99085181e+02,
        -6.47134175e+01, -3.04763108e+02, -2.70217517e+00, -6.08393280e+01,
         2.41301140e+02,  1.63617364e+01,  3.40377429e+01, -2.09989785e+01,
        -4.83994810e+02, -3.20466164e+02, -2.42416199e+02,  5.37136065e+00,
         1.08101946e+03, -1.65356954e+03, -2.25977651e+02,  2.24629300e+02,
        -1.38366040e+02,  2.33596690e+03,  1.07525485e+03,  3.43890691e+02,
         3.88728359e+01,  8.39391249e+02,  2.52885269e+01,  3.45209182e+02,
        -7.02120001e+02, -1.24939676e+03,  8.53851251e+02, -7.54650963e+02,
         1.09686440e+03, -6.54649124e+00,  1.44967735e+02, -3.40920777e+02,
        -1.14320086e+03,  3.46161773e+02, -1.70852496e+02,  1.56733309e+02,
        -1.66846584e+02,  5.10064913e+02,  2.03420898e+02,  9.23269778e+02,
         2.52217482e+02, -7.84889668e+01, -1.77788061e+03,  1.26169857e+02,
        -5.28419990e+02,  5.28089456e+02, -1.95059768e+00,  6.92836855e+02,
         6.15070419e+01, -5.12005383e+02,  6.04911803e+02, -9.92934990e+02,
         7.85087219e+02, -1.33145030e+03])
```

```
polyreg_scaled.score(X_train, y_train)
```

```
0.737636017211148
```

```
polyreg_scaled.score(X_test, y_test)
```

```
0.6949579846075127
```

- As per the regularization method 2(Ridge), higher "Good fit" is again found at standard regularization value and polynomial degree of 2.

Lasso (L1 Regularisation)

```
degree = 2
scaler = StandardScaler()
polyreg_scaled = make_pipeline(PolynomialFeatures(degree), scaler, Lasso(alpha = 0.01)) #alpha - Regularisation strength.
polyreg_scaled.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning: Objective did not conve
  model = cd_fast.enet_coordinate_descent(
```

```
Pipeline
  PolynomialFeatures
  StandardScaler
  Lasso
```

```
polyreg_scaled.steps[-1][1].coef_
```

```
array([ 0.00000000e+00,  6.70744188e+03, -9.51265318e+02, -2.97221101e+03,
        3.38386873e+03,  2.74211559e+03, -1.58504397e+03,  5.33723662e+02,
       -1.01075242e+03,  6.02243651e+02,  5.73585515e+02, -1.95482963e+02,
       -5.07445554e+03,  7.84955030e+02, -8.17828660e+02,  6.60565826e+02,
       -1.14443936e+02,  8.12927390e+02,  1.33526621e+03,  3.04100777e+02,
        1.38860328e+02,  4.24815389e+02,  1.25133755e+03,  7.52523567e+02,
       -5.80182955e+01, -3.22672514e+02, -4.41343340e+00, -2.38519832e+02,
        5.25315947e+02,  2.26551432e+01,  5.39306660e+01, -4.72977840e+01,
       -8.26505808e+02,  9.21001360e+02, -2.39210825e+02,  6.91335984e+00,
        2.21664633e+03, -3.10987571e+03, -2.22563624e+02,  2.16496514e+02,
       -1.31948894e+02,  4.21361232e+03, -2.27297570e+02,  3.31718819e+02,
       -6.81572350e+02,  1.73300785e+03,  7.51376675e+00,  3.46251744e+02,
       -7.36179380e+02, -2.49132922e+03,  6.62526528e+01, -1.64916874e+03,
        2.18346001e+03,  4.70953843e-01,  1.62117391e+02, -3.48829080e+02,
       -2.54199537e+03,  8.27844391e+02, -8.57897467e+02,  3.48863969e+02,
       -9.55375212e+01,  9.83494570e+02,  2.23223036e+02,  1.25948208e+03,
        1.49557025e+01, -2.24285870e+02, -2.35150111e+03,  4.19020007e+02,
       -2.93285213e+02,  5.29119739e+02, -1.46833987e+00,  9.85716956e+02,
       -5.63212880e+02, -5.42550116e+02,  7.72956484e+02, -1.00069439e+03,
        1.49276421e+03, -2.16970304e+03])
```

```
polyreg_scaled.score(X_train, y_train)
```

```
0.7386965060272441
```

```
polyreg_scaled.score(X_test, y_test)
```

```
0.6907896197451431
```

- As per the regularization method 1(Lasso), "Good fit" is again found at regularization value of 0.01 and polynomial degree of 2 onwards.

Feature importance

```
from sklearn.ensemble import RandomForestRegressor
RF = RandomForestRegressor(n_estimators=100,
    max_depth=10,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features='sqrt',
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    bootstrap=True,
    oob_score=False,
    n_jobs=None,
    random_state=None,
    ccp_alpha=0.0085)
```

```
RF.fit(X_train,y_train)
```

```
RandomForestRegressor
RandomForestRegressor(ccp_alpha=0.0085, max_depth=10, max_features='sqrt')
```

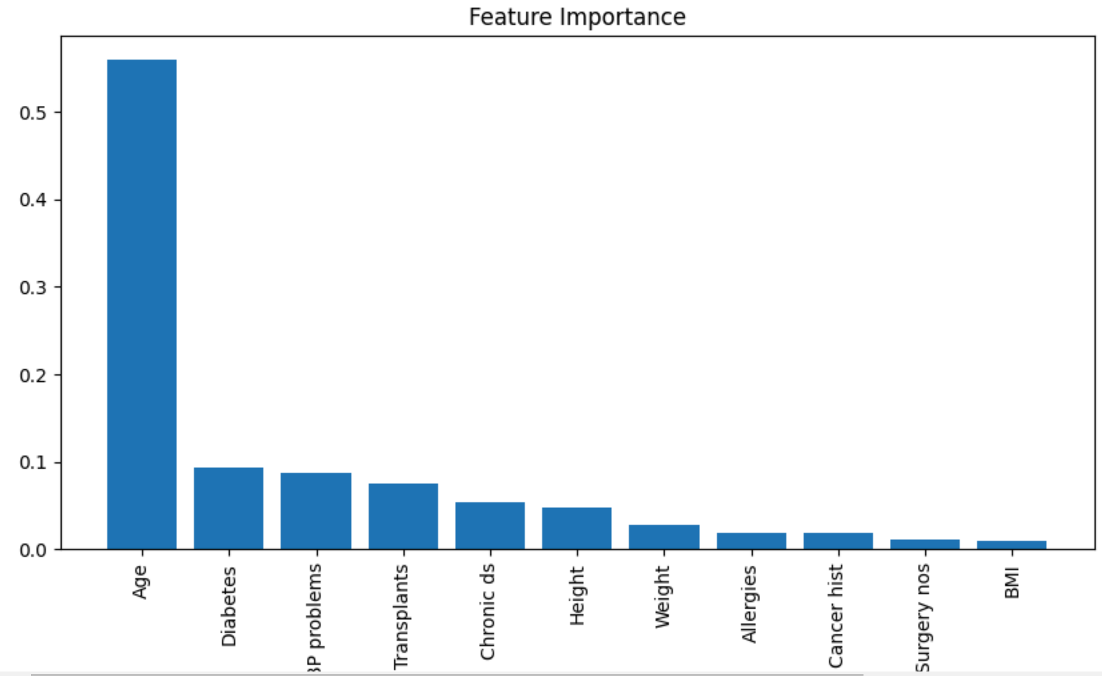
```
RF.score(X_train,y_train),RF.score(X_test,y_test)
```

```
(0.9429504504215943, 0.775308520034433)
```

```
RF.feature_importances_
```

```
array([0.55884942, 0.01042524, 0.01886764, 0.08711633, 0.02825956,
       0.05376918, 0.09369359, 0.00873804, 0.01783793, 0.04764638,
       0.07479669])
```

```
# Feature Importance
importances = RF.feature_importances_

indices = np.argsort(importances)[::-1] # Sort feature importances in descending order
names = ["Age", "Diabetes", "BP problems", "Transplants", "Chronic ds","Height", "Weight", "Allergies", "Cancer hist", "Surgery nos", "

plt.figure(figsize=(10, 5)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(len(names)), importances[indices]) # Add bars
plt.xticks(range(len(names)), names, rotation=90) # Add feature names as x-axis labels
plt.show()
```



- This shows that Age plays a major role in deciding the premium price of the individuals

```
df1.columns
```

```
Index(['Age', 'Diabetes', 'BP_problems', 'Transplants', 'Chronic_ds', 'Height',
       'Weight', 'Allergies', 'Cancer_hist', 'Surgery_nos', 'PremiumPrice',
       'age_cat', 'premium_cat', 'BMI', 'bmi_cat'],
      dtype='object')
```

```
df1.to_csv("insurance_pred.csv")
```

```
# ---------------------------------------------------------------------------------------------------------------
```

Actionable Insights and recommendations

- After applying all the tests above we can conclude that the data provided can be modeled using a Linear Regression model.
- Based on all the analysis and study done above, it is sure that certain factors such as "Age", "Diabetes", "BMI","Blood pressure problems", "Transplants", "Presence of chronic diseases", "Number of surgeries done" plays a major part in deciding the premium price of the beneficiary.
- Feature importance analysis shows that Age, Diabetes and BP problems play a major role in impacting the premium price and less impact will be made by "BMI", "No. of surgeries", yet those two factors play a small role in final prediction price
- Hence an app will be created to predict the premium price using these 7 factors alone.
- Below are the steps towards the development of the app.

```
# ---------------------------------------------------------------------------------------------------------------
```

```
X = df1[['Age', 'BMI', 'BP_problems', 'Transplants', 'Chronic_ds', 'Surgery_nos', 'Diabetes']]
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.