Medical Insurance Premium Prediction


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import pandas_profiling as pp
```

In [2]:
```
df=pd.read_csv('Medicalpremium.csv')
```

In [3]:
```
df.head(10)
```

Out[3]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownA
0	45	0	0	0	0	155	57	
1	60	1	0	0	0	180	73	
2	36	1	1	0	0	158	59	
3	52	1	1	0	1	183	93	
4	38	0	0	0	1	166	88	
5	30	0	0	0	0	160	69	
6	33	0	0	0	0	150	54	
7	23	0	0	0	0	181	79	
8	48	1	0	0	0	169	74	
9	38	0	0	0	0	182	93	

In [4]:
```
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
```

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Age                        0
Diabetes                   0
BloodPressureProblems      0
AnyTransplants             0
AnyChronicDiseases         0
Height                     0
Weight                     0
KnownAllergies             0
HistoryOfCancerInFamily    0
NumberOfMajorSurgeries     0
PremiumPrice               0
dtype: int64
```

In [6]:
```
df.shape
```

Out[6]:
```
(986, 11)
```

In [7]:
```
df.describe()
```

Out[7]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height
count	986.000000	986.000000	986.000000	986.000000	986.000000	986.000000
mean	41.745436	0.419878	0.468560	0.055781	0.180527	168.182556
std	13.963371	0.493789	0.499264	0.229615	0.384821	10.098155
min	18.000000	0.000000	0.000000	0.000000	0.000000	145.000000
25%	30.000000	0.000000	0.000000	0.000000	0.000000	161.000000
50%	42.000000	0.000000	0.000000	0.000000	0.000000	168.000000
75%	53.000000	1.000000	1.000000	0.000000	0.000000	176.000000
max	66.000000	1.000000	1.000000	1.000000	1.000000	188.000000

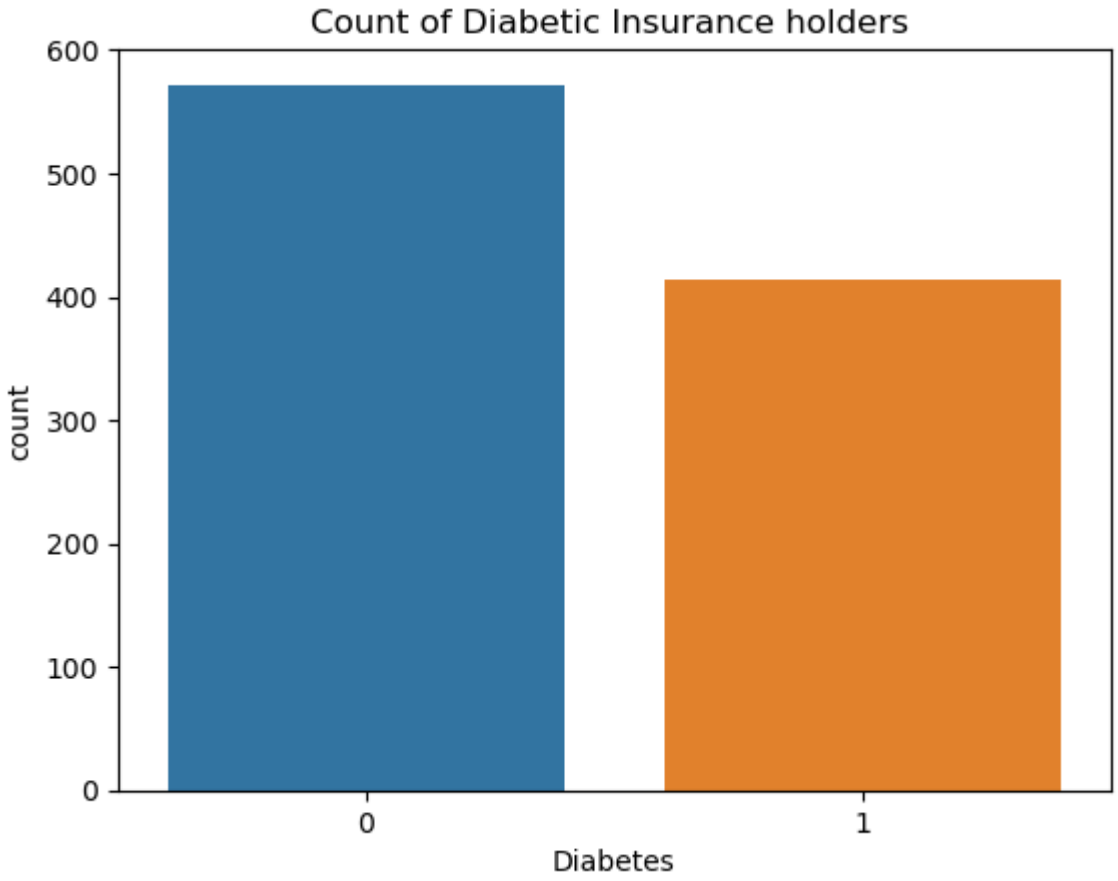
In [8]:
```
# EDA (we will analyse the data through graph and also add new columns to dataset in
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice'],
      dtype='object')
```

In [10]:
```
sns.countplot(data=df, x='Diabetes')
plt.title('Count of Diabetic Insurance holders ')
```

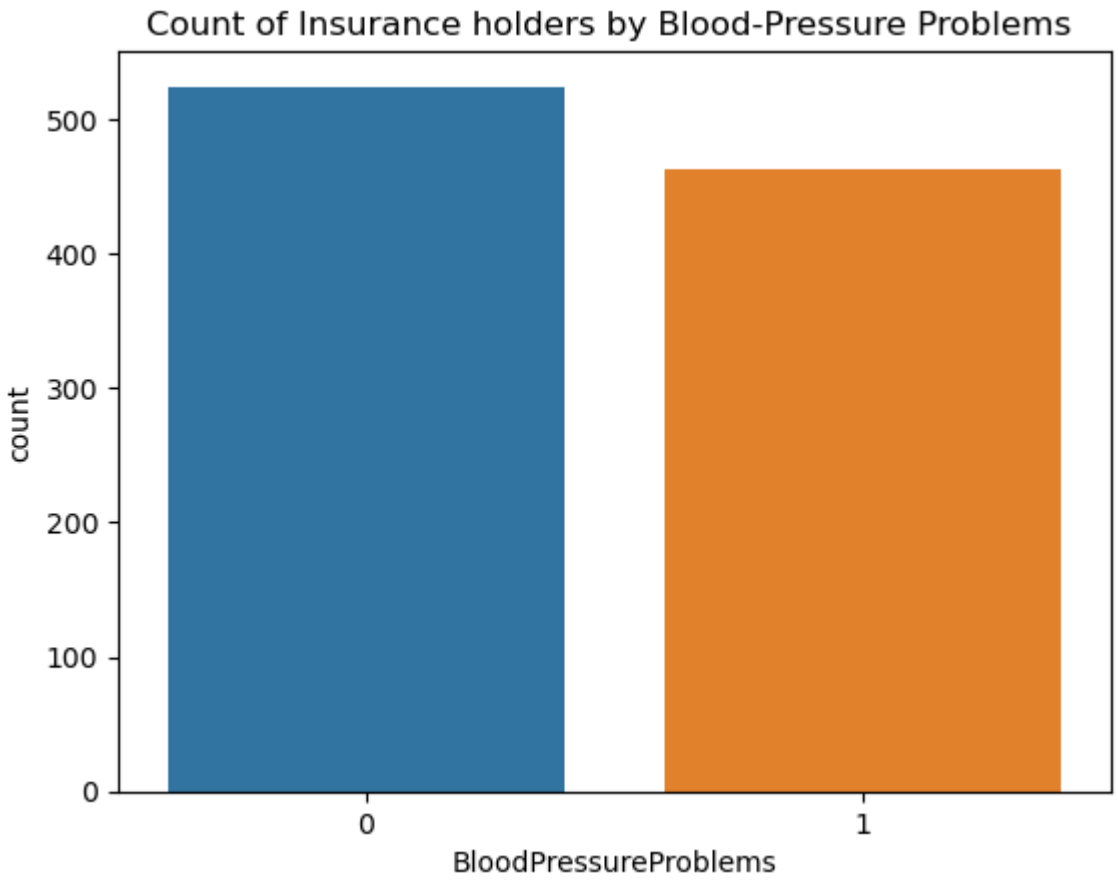
Out[10]:
```
Text(0.5, 1.0, 'Count of Diabetic Insurance holders ')
```



In [11]:
```
# we can figure out that non Diebetic people holds medicail insurance than Diebetic p
```

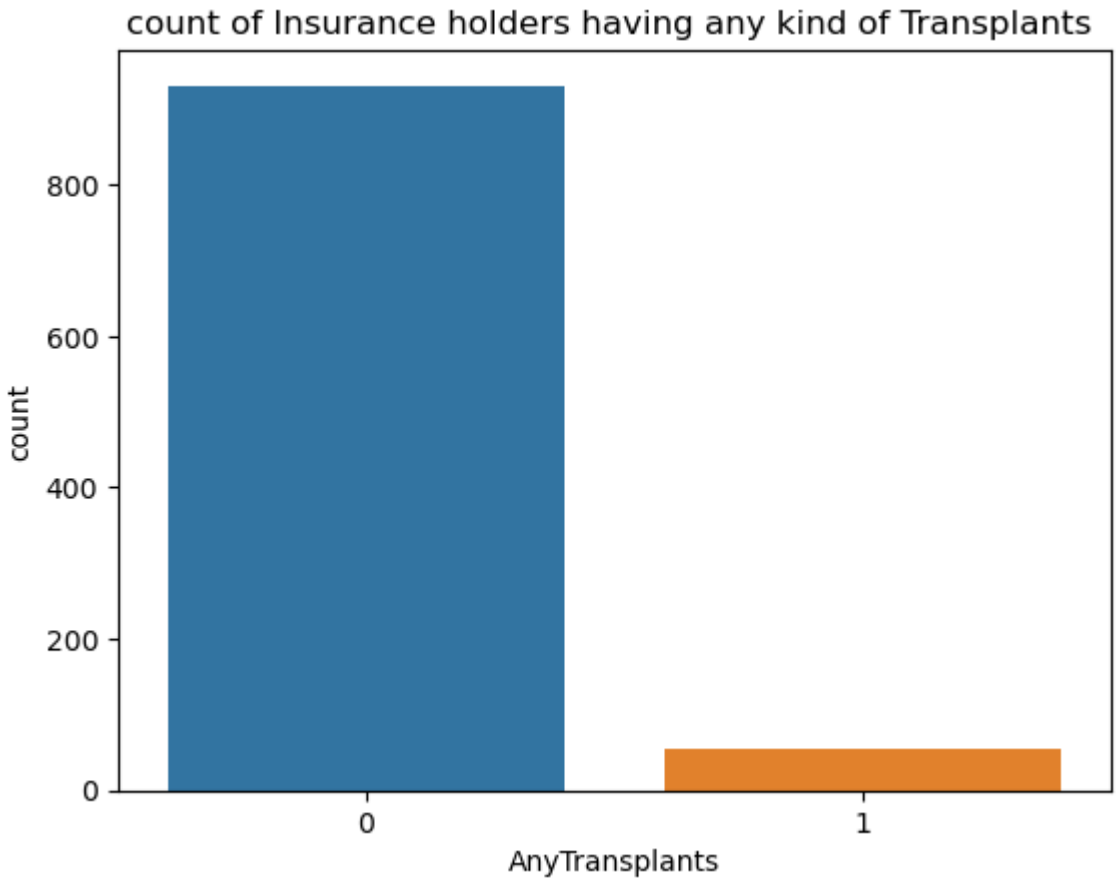
In [12]:
```
sns.countplot(data=df, x='BloodPressureProblems')
plt.title('Count of Insurance holders by Blood-Pressure Problems ')
```

Out[12]: Text(0.5, 1.0, 'Count of Insurance holders by Blood-Pressure Problems ')



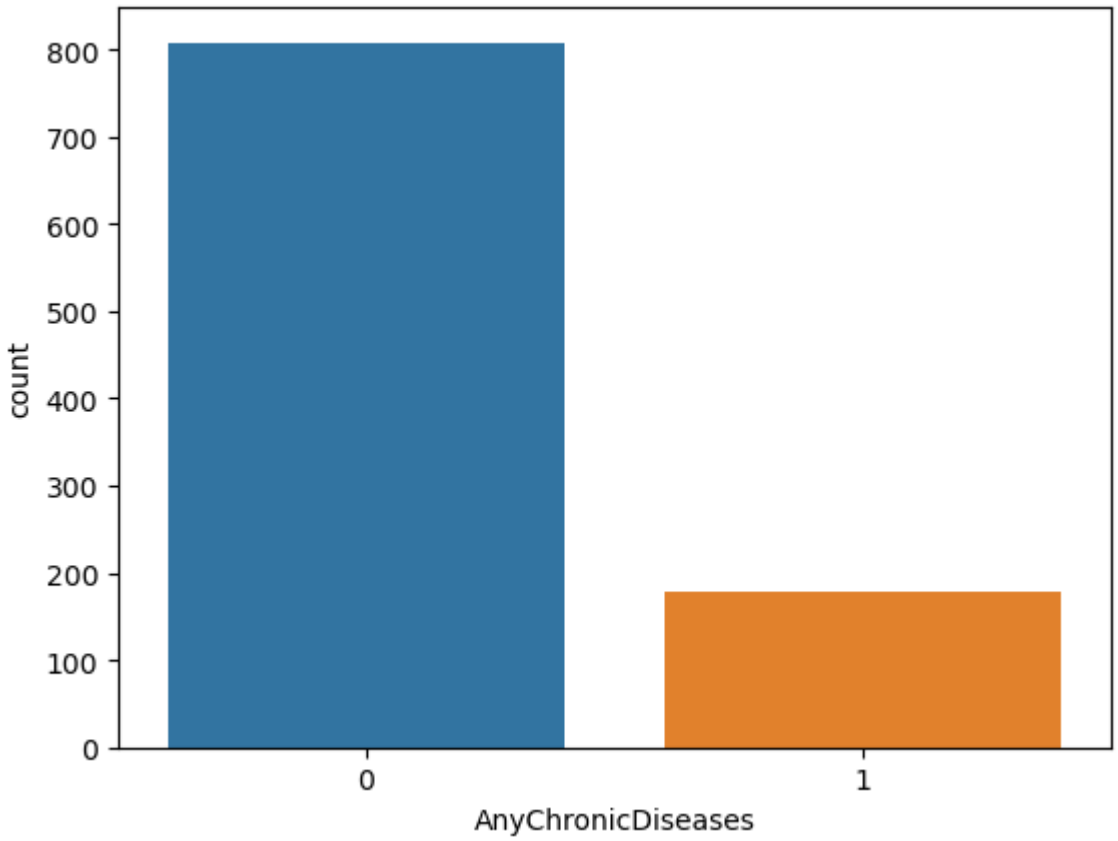
In [13]:
```
sns.countplot(data=df, x='AnyTransplants')
plt.title('count of Insurance holders having any kind of Transplants ')
```

Out[13]: Text(0.5, 1.0, 'count of Insurance holders having any kind of Transplants ')



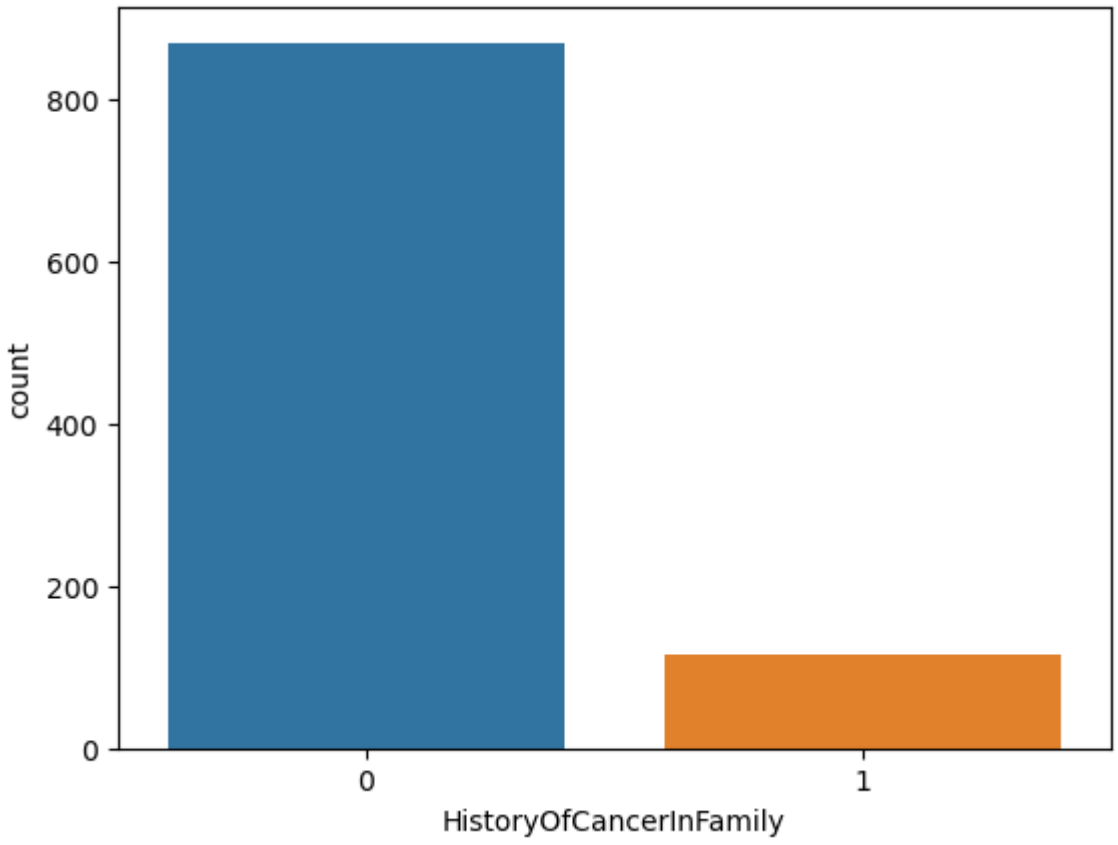
In [14]:
```
sns.countplot(data=df, x='AnyChronicDiseases')
```

Out[14]: <AxesSubplot:xlabel='AnyChronicDiseases', ylabel='count'>



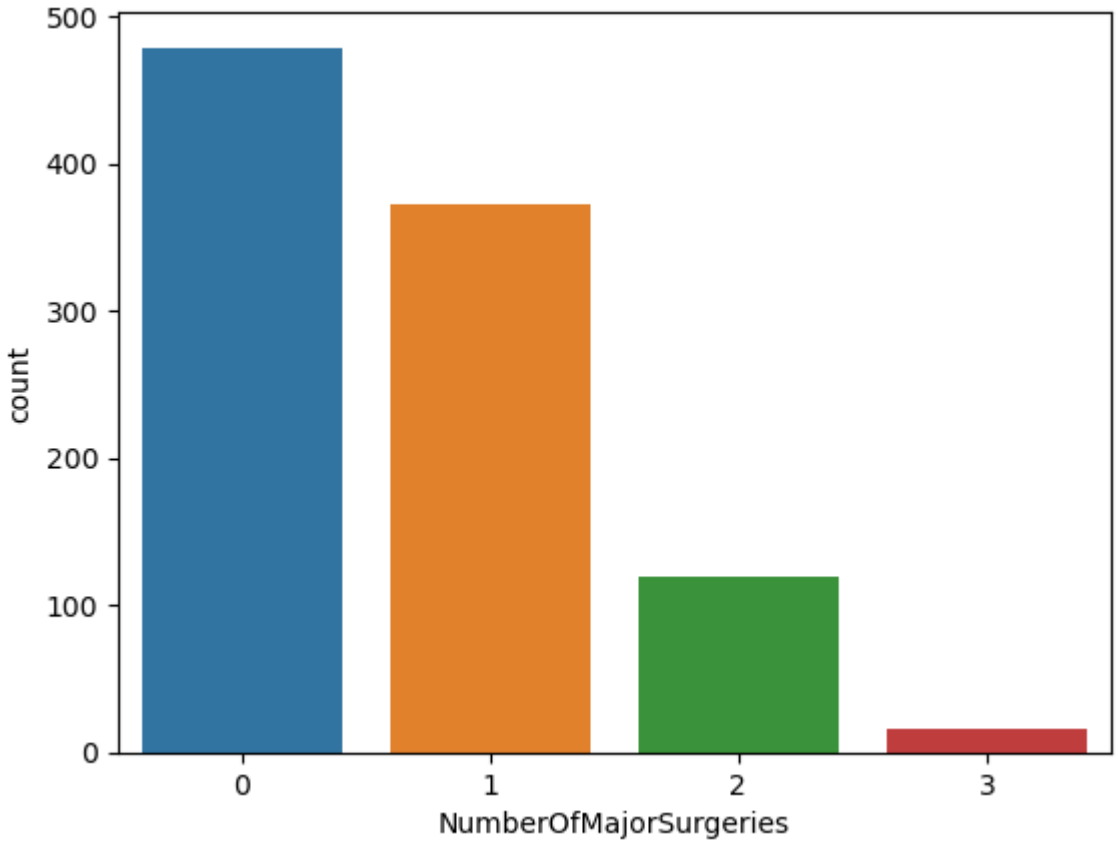
In [15]:
```
sns.countplot(data=df,x='HistoryOfCancerInFamily')
```

Out[15]: <AxesSubplot:xlabel='HistoryOfCancerInFamily', ylabel='count'>

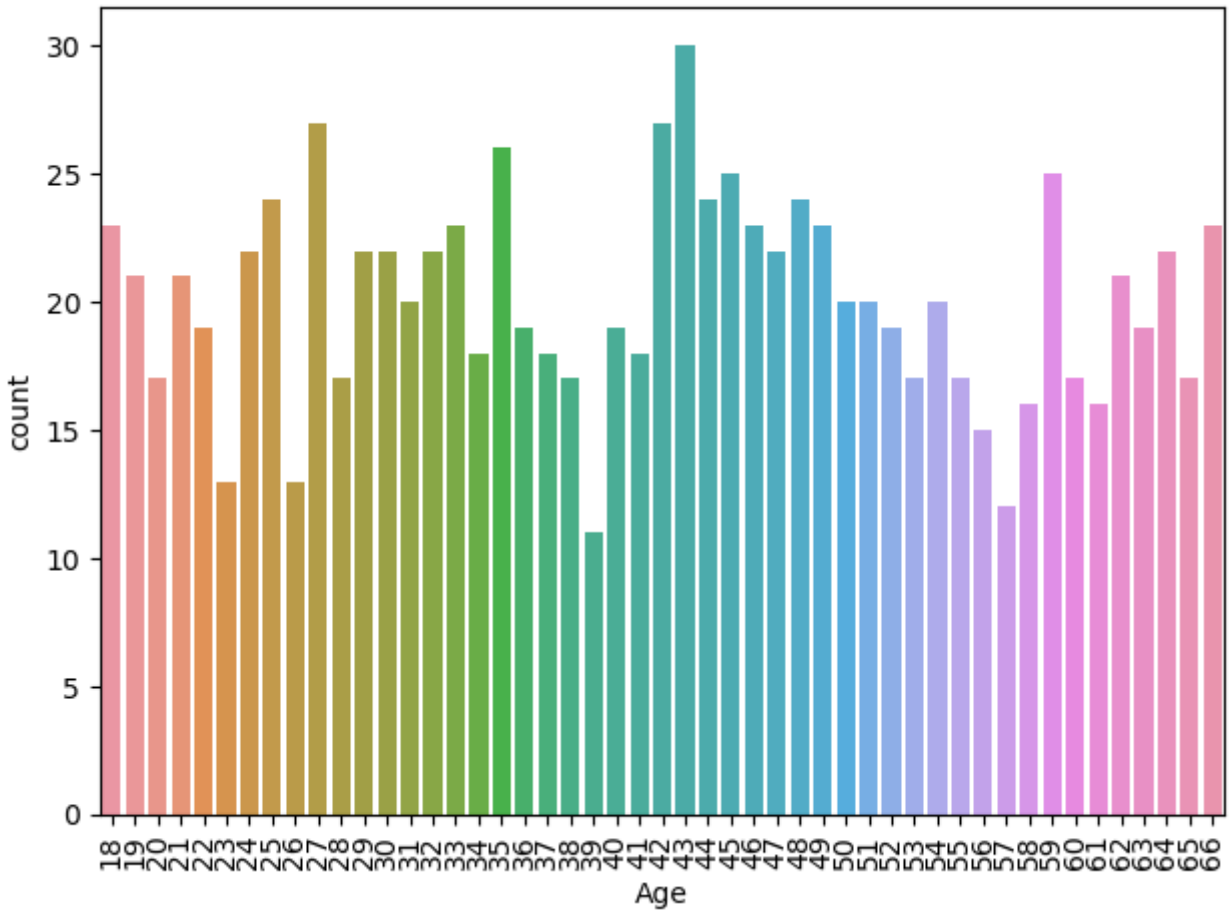


In [16]:
```
sns.countplot(data=df,x='NumberOfMajorSurgeries')
```

Out[16]:
```
<AxesSubplot:xlabel='NumberOfMajorSurgeries', ylabel='count'>
```



In [17]:
```
sns.countplot(data=df, x='Age')
plt.xticks(rotation=90)
plt.tight_layout()
```



In [18]:
```
# we will make Age categories
# 18 - 29 - young
# 30 - 40 - Middle age
# 41 - 55 - Average Age
# 56 - 60+ - Senior
```

In [19]:
```
bins = [15,29,40,55,70]
labels = ['Young','Middle age','Average age','Senior']

df['Age_category'] = pd.cut(df['Age'],bins=bins,labels=labels,right=False)
```

In [20]:
```
df
```

Out[20]:

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

986 rows × 12 columns

In [21]:
```
pp.ProfileReport(df)
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[21]:

In [22]:
```
# lets check the outliers if any
```

In [23]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 986 entries, 0 to 985
Data columns (total 12 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   
 11  Age_category             986 non-null    category
dtypes: category(1), int64(11)
memory usage: 86.0 KB
```

In [24]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Age_category']=le.fit_transform(df['Age_category'])
```

In [25]:
```
df
```

Out[25]:

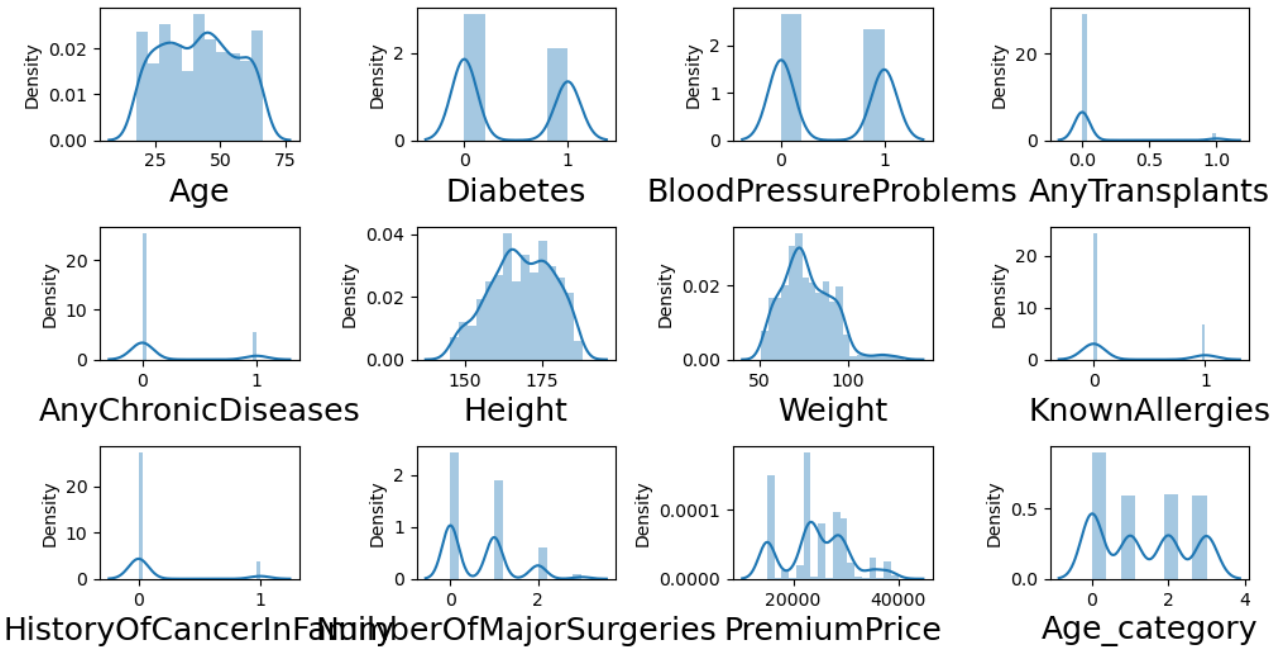
	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	Know
0	45	0	0	0	0	155	57	
1	60	1	0	0	0	180	73	
2	36	1	1	0	0	158	59	
3	52	1	1	0	1	183	93	
4	38	0	0	0	1	166	88	
...	...	...	...	...	...	...	...	
981	18	0	0	0	0	169	67	
982	64	1	1	0	0	153	70	
983	56	0	1	0	0	155	71	
984	47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

986 rows × 12 columns

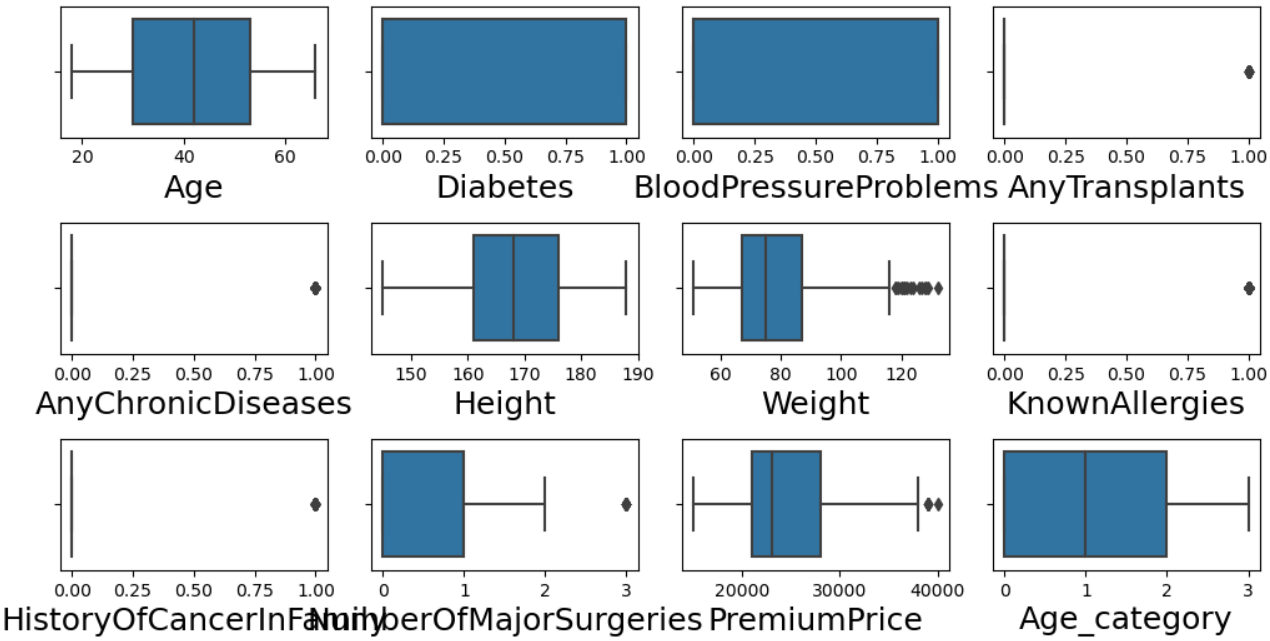
In [26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 986 entries, 0 to 985
Data columns (total 12 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
 11  Age_category              986 non-null    int32
dtypes: int32(1), int64(11)
memory usage: 88.7 KB
```

In [27]:
```
plt.figure(figsize=(10,10))
pltn = 1

for i in df:
    if pltn<=25:
        ax = plt.subplot(6,4,pltn)
        sns.distplot(df[i])
        plt.xlabel(i,fontsize=18)
    pltn=pltn+1
plt.tight_layout()
```



In [28]:
```
plt.figure(figsize=(10,10))
pltn=1

for i in df:
    if pltn<=25:
        ax = plt.subplot(6,4,pltn)
        sns.boxplot(df[i])
        plt.xlabel(i,fontsize=18)
    pltn=pltn+1
plt.tight_layout()
```



In [29]:
```
# lets remove the outliers
```

In [30]:
```
from scipy.stats import zscore
```

In [31]:
```
z=np.abs(zscore(df))
new_data=df[(z<=3).all(axis=1)]
```

In [32]:
```
new_data
```

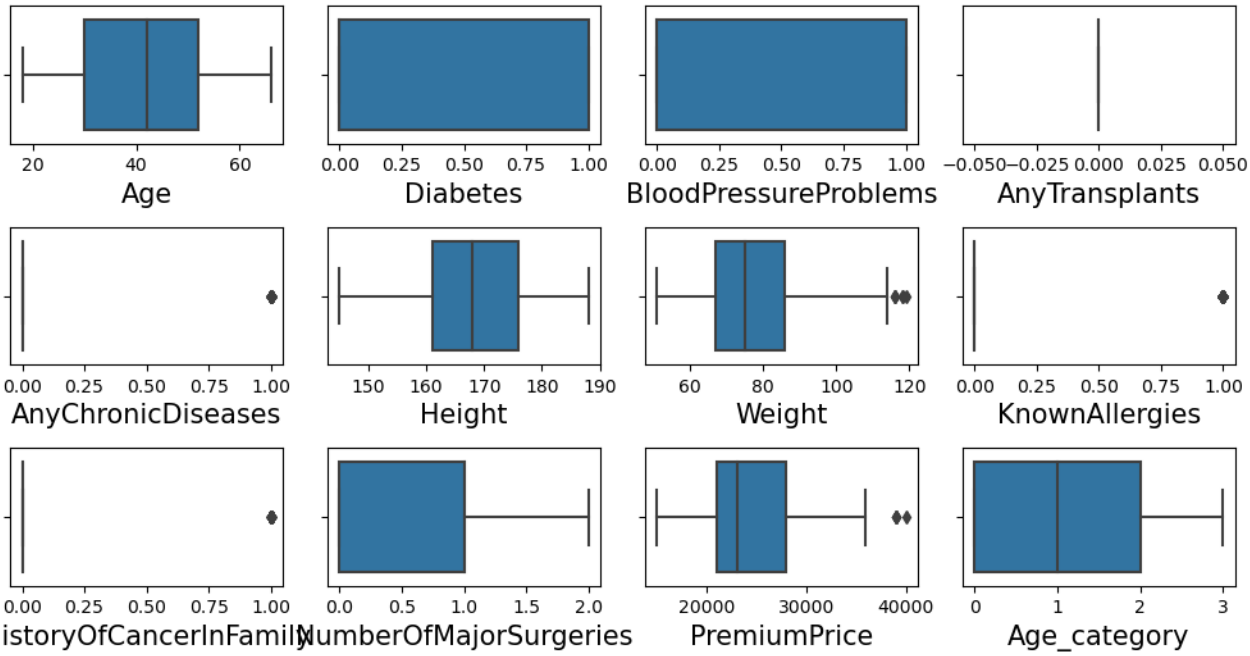
Out[32]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	Know
0	45	0	0	0	0	155	57	
1	60	1	0	0	0	180	73	
2	36	1	1	0	0	158	59	
3	52	1	1	0	1	183	93	
4	38	0	0	0	1	166	88	
...	...	...	...	...	...	...	...	
979	24	0	0	0	0	161	71	
981	18	0	0	0	0	169	67	
983	56	0	1	0	0	155	71	
984	47	1	1	0	0	158	73	
985	21	0	0	0	0	158	75	

904 rows × 12 columns

In [33]:
```
# checking outlier after removing (still few are there)
```

In [34]:
```
plt.figure(figsize=(10,10))
pltn=1

for y in new_data:
    if pltn<=25:
        ax=plt.subplot(6,4,pltn)
        sns.boxplot(new_data[y])
        plt.xlabel(y,fontsize=15)
    pltn=pltn+1
plt.tight_layout()
```



In [35]:
```
new_data.skew()
```

Out[35]:
```
Age                         0.032571
Diabetes                    0.332464
BloodPressureProblems       0.102071
AnyTransplants              0.000000
AnyChronicDiseases          1.646529
Height                     -0.192481
Weight                      0.371147
KnownAllergies              1.360976
HistoryOfCancerInFamily     2.350399
NumberOfMajorSurgeries      0.645781
PremiumPrice               -0.084733
Age_category                0.242663
dtype: float64
```

In [36]:
```
new_data.shape
```

Out[36]:
```
(904, 12)
```

In [37]:
```
df.shape
```

Out[37]:
```
(986, 12)
```

Dataloss = 8%

In [38]:
```
Dataloss = ((986-904)/986)*100
Dataloss
```

Out[38]:
```
8.316430020283976
```

In [39]:
```
pp.ProfileReport(new_data)
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[39]:

In [40]:
```
new_data.columns
```

Out[40]:
```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice',
       'Age_category'],
      dtype='object')
```

In [41]:
```
x = new_data.drop('PremiumPrice',axis=1)
y = new_data['PremiumPrice']
```

In [42]:
```
x
```

Out[42]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	Know
0	45	0	0	0	0	155	57	
1	60	1	0	0	0	180	73	
2	36	1	1	0	0	158	59	
3	52	1	1	0	1	183	93	
4	38	0	0	0	1	166	88	
...	...	...	...	...	...	...	...	
979	24	0	0	0	0	161	71	
981	18	0	0	0	0	169	67	
983	56	0	1	0	0	155	71	
984	47	1	1	0	0	158	73	
985	21	0	0	0	0	158	75	

904 rows × 11 columns

In [43]:
```
y
```

Out[43]:
```
0      25000
1      29000
2      23000
3      28000
4      23000
       ...
979    15000
981    15000
983    29000
984    39000
985    15000
Name: PremiumPrice, Length: 904, dtype: int64
```

scaling

In [44]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)
```

In [45]:
```
x
```

Out[45]:
```
array([[ 0.25454296, -0.84772117, -0.9503462 , ..., -0.36834548,
        -0.90751679, -1.1251223 ],
       [ 1.34475222,  1.17963316, -0.9503462 , ..., -0.36834548,
        -0.90751679,  0.60318791],
       [-0.3995826 ,  1.17963316,  1.05224812, ..., -0.36834548,
         0.5343025 , -0.26096719],
       ...,
       [ 1.05402975, -0.84772117,  1.05224812, ..., -0.36834548,
         0.5343025 ,  0.60318791],
       [ 0.39990419,  1.17963316,  1.05224812, ..., -0.36834548,
         0.5343025 , -1.1251223 ],
       [-1.48979186, -0.84772117, -0.9503462 , ..., -0.36834548,
         0.5343025 ,  1.46734301]])
```

In [46]:
```
# Modeling building
```

In [47]:
```
from sklearn.model_selection import train_test_split
```

In [48]:
```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=30)
```

In [49]:
```
xtrain
```

Out[49]:
```
array([[ 1.34475222,  1.17963316,  1.05224812, ..., -0.36834548,
        -0.90751679,  0.60318791],
       [-1.19906939,  1.17963316,  1.05224812, ..., -0.36834548,
         0.5343025 ,  1.46734301],
       [-0.18154075, -0.84772117, -0.9503462 , ...,  2.71484261,
         0.5343025 , -0.26096719],
       ...,
       [-0.32690198, -0.84772117,  1.05224812, ..., -0.36834548,
        -0.90751679, -0.26096719],
       [-1.12638877,  1.17963316,  1.05224812, ...,  2.71484261,
         0.5343025 ,  1.46734301],
       [ 0.69062666,  1.17963316,  1.05224812, ..., -0.36834548,
         0.5343025 , -1.1251223 ]])
```

In [50]:
```
ytrain
```

Out[50]:
```
948    25000
343    15000
451    31000
263    23000
398    28000
       ...
153    23000
549    19000
882    23000
468    15000
874    29000
Name: PremiumPrice, Length: 632, dtype: int64
```

In [51]:
```
# lets import metrix
```

In [52]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [53]:
```
# lets import the Regression models
```

In [54]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
```

In [55]:
```
model = [LinearRegression(), Ridge(alpha=10), Lasso(alpha=10), DecisionTreeRegressor(

model_name = ['Linear Regression','Redge L2','Lasso L1','Decision Tree Regressor', 'R

mae=[]
mse=[]
rmse=[]
r2_score_Accuracy=[]


for i in model:
    i.fit(xtrain, ytrain)
    ypred = i.predict(xtest)
    mae.append(mean_absolute_error(ytest,ypred))
    mse.append(mean_squared_error(ytest,ypred))
    rmse.append(np.sqrt(mse))
    r2_score_Accuracy.append(r2_score(ytest,ypred))

model_comparision = pd.DataFrame({'Model': model_name, 'MAE': mae, 'MSE': mse, 'RMSE'
model_comparision.sort_values(by='Accuracy',ascending=False)
```

Out[55]:

	Model	MAE	MSE	RMSE	Accuracy
4	Random Forest Regressor	722.610294	4.845216e+06	[2836.2160865993383, 2837.0170501450907, 2835....	0.849898
7	GradientBoosting Regressor	1298.600491	6.620214e+06	[2836.2160865993383, 2837.0170501450907, 2835....	0.794909
8	XGboost Regressor	1201.243563	6.671332e+06	[2836.2160865993383, 2837.0170501450907, 2835....	0.793325
3	Decision Tree Regressor	720.588235	7.875000e+06	[2836.2160865993383, 2837.0170501450907, 2835....	0.756036
2	Lasso L1	2038.492693	8.039691e+06	[2836.2160865993383, 2837.0170501450907, 2835....	0.750934
0	Linear Regression	2035.256625	8.044122e+06	[2836.2160865993383]	0.750797
1	Redge L2	2043.964605	8.048666e+06	[2836.2160865993383, 2837.0170501450907]	0.750656
6	KNN Regressor	1903.676471	1.055632e+07	[2836.2160865993383, 2837.0170501450907, 2835....	0.672970
5	SVR regressor	4410.234294	3.229284e+07	[2836.2160865993383, 2837.0170501450907, 2835....	-0.000416

In [56]:
```
lr = LinearRegression()
lr.fit(xtrain,ytrain)

l2 = Ridge(alpha=10)
l2.fit(xtrain,ytrain)

l1 = Lasso(alpha=10)
l1.fit(xtrain,ytrain)

dtc = DecisionTreeRegressor()
dtc.fit(xtrain,ytrain)

rf = RandomForestRegressor()
rf.fit(xtrain,ytrain)

svr = SVR()
svr.fit(xtrain,ytrain)

knn = KNeighborsRegressor()
knn.fit(xtrain,ytrain)

gb = GradientBoostingRegressor()
gb.fit(xtrain,ytrain)

xg = XGBRegressor()
xg.fit(xtrain,ytrain)
```

Out[56]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

In [57]:
```
ypred1 = lr.predict(xtest)

ypred2 = l2.predict(xtest)

ypred3 = l1.predict(xtest)

ypred4 = dtc.predict(xtest)

ypred5 = rf.predict(xtest)

ypred6 = svr.predict(xtest)

ypred7 = knn.predict(xtest)

ypred8 = gb.predict(xtest)

ypred9 = xg.predict(xtest)
```

In [58]:
```
# lets see the result of the medels in dataframe
```

In [59]:
```
algo_train = pd.DataFrame({'Actual': ytest, 'Linear Regression': ypred1, 'Ridge L2':
```

In [60]:
```
algo_train
```

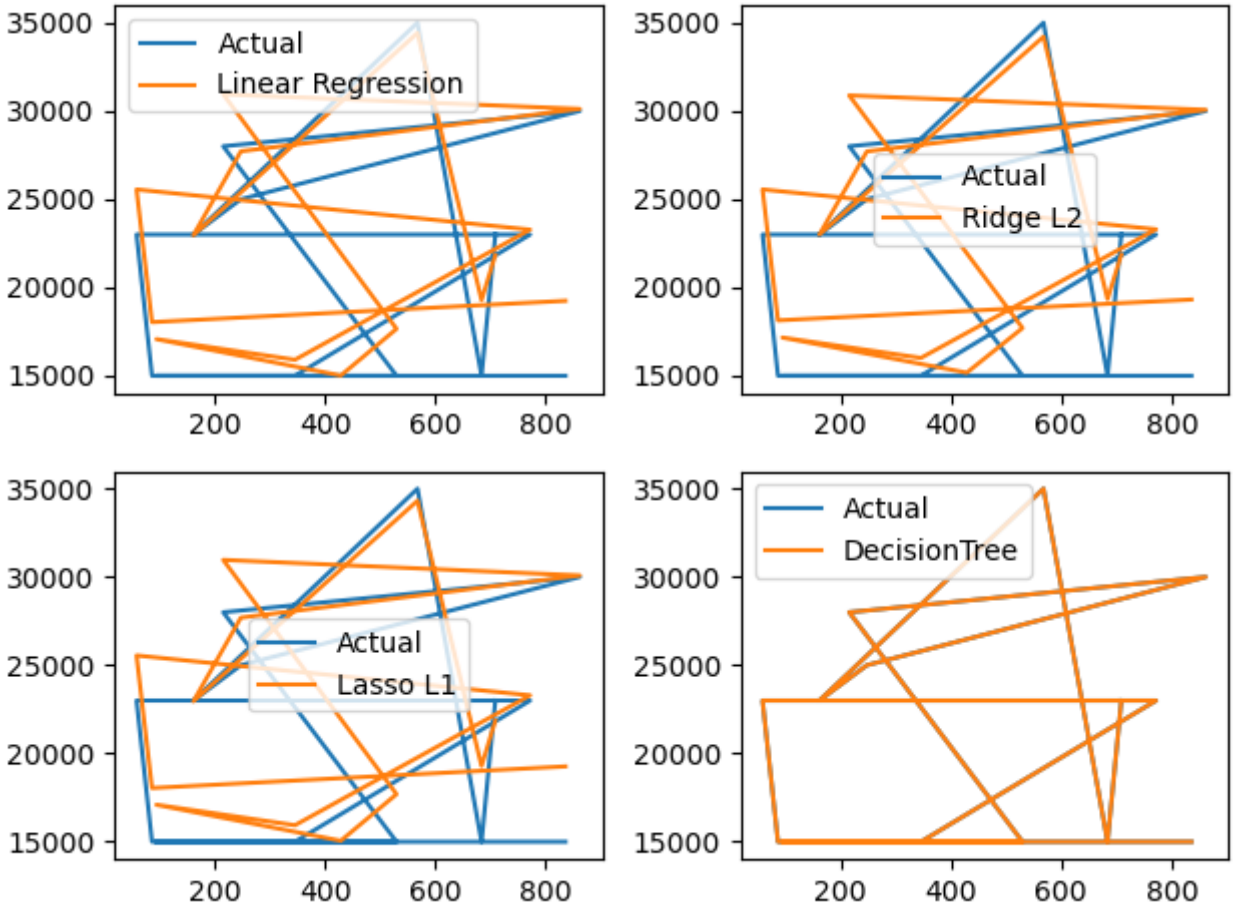
Out[60]:

	Actual	Linear Regression	Ridge L2	Lasso L1	DecisionTree	RandomForest	SVR	K
709	23000	21841.966282	21861.811820	21859.575014	23000.0	24410.0	22999.806130	2300
684	15000	19287.189287	19341.099724	19290.843155	15000.0	15100.0	23023.655670	1520
568	35000	34419.042346	34207.745219	34316.520722	35000.0	34920.0	23050.708227	2840
162	23000	22960.763200	22997.389620	22958.896369	23000.0	23390.0	23012.850335	2380
248	25000	27715.743021	27707.828119	27695.692376	25000.0	25260.0	23049.241323	2760
...	...	...	...	...	...	...	...	
164	15000	19981.829448	20033.723262	20008.436513	15000.0	15560.0	22987.774512	2260
685	25000	27040.599425	27011.926105	27006.253781	25000.0	25100.0	23038.220075	2420
648	28000	27016.982672	27017.543487	27082.429891	28000.0	28000.0	23064.772412	2800
548	15000	15004.800196	15107.031752	15007.185021	15000.0	17780.0	22980.147753	2460
18	25000	25584.858294	25499.343947	25585.933208	25000.0	25100.0	23035.670310	2660

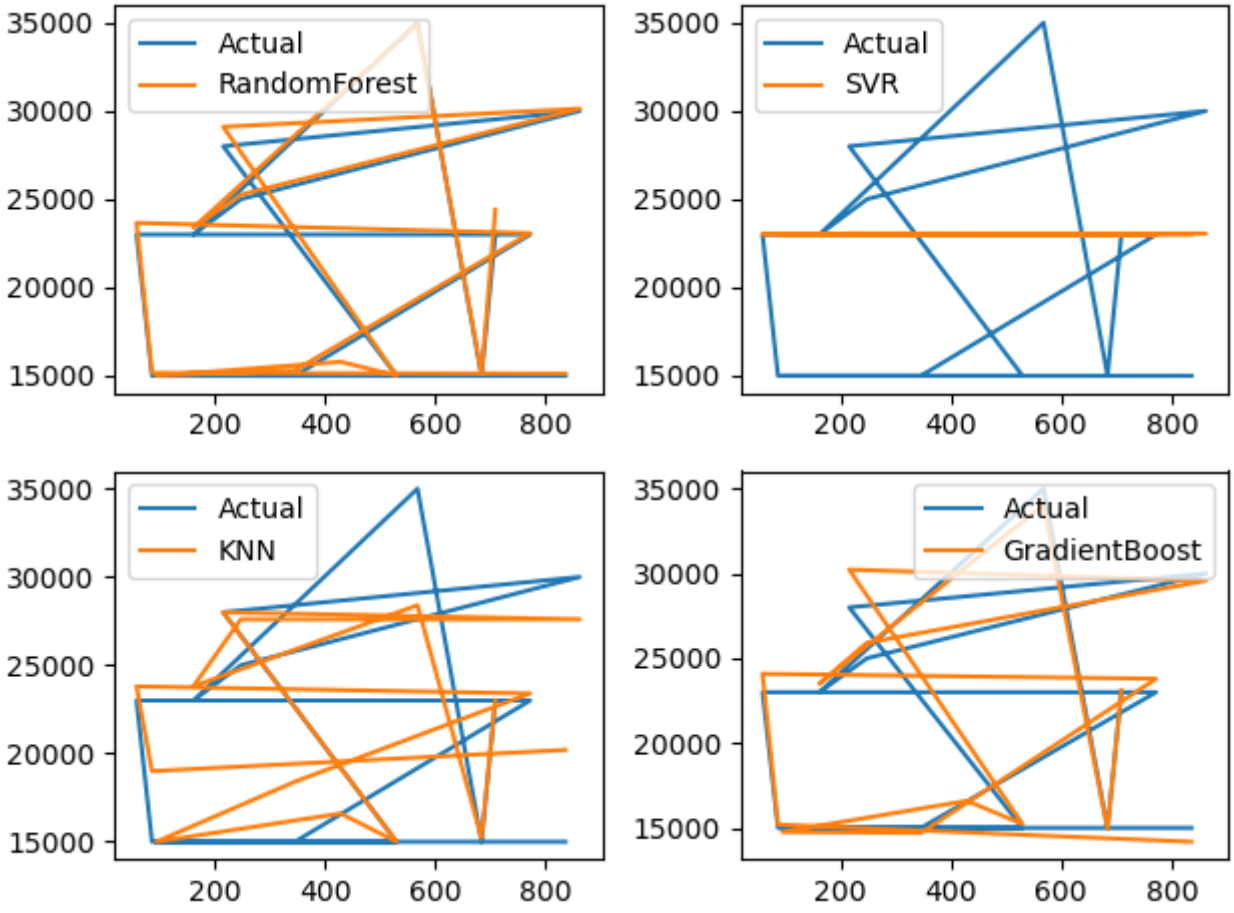
272 rows × 10 columns

In [61]:
```
import matplotlib.pyplot as plt
```

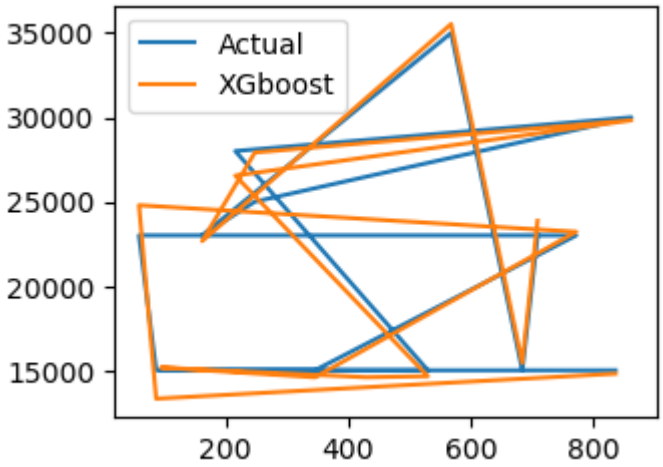
In [62]:
```
# here we check first 4 algos through graph
# blue line show the actual values
# Orange line represents the predicted values

plt.subplot(221)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['Linear Regression'].iloc[0:15], label='Linear Regression')
plt.legend()

plt.subplot(222)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['Ridge L2'].iloc[0:15], label='Ridge L2')
plt.legend()

plt.subplot(223)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['Lasso L1'].iloc[0:15], label='Lasso L1')
plt.legend()

plt.subplot(224)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['DecisionTree'].iloc[0:15], label='DecisionTree')
plt.legend()

plt.tight_layout()
```



In [63]:
```
# here we check another 4 algos

plt.subplot(221)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['RandomForest'].iloc[0:15], label='RandomForest')
plt.legend()

plt.subplot(222)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['SVR'].iloc[0:15], label='SVR')
plt.legend()

plt.subplot(223)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['KNN'].iloc[0:15], label='KNN')
plt.legend()

plt.subplot(224)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['GradientBoost'].iloc[0:15], label='GradientBoost')
plt.legend()

plt.tight_layout()
```



In [64]:
```
plt.subplot(221)
plt.plot(algo_train['Actual'].iloc[0:15], label='Actual')
plt.plot(algo_train['XGboost'].iloc[0:15], label='XGboost')
plt.legend()

plt.tight_layout()
```



In [65]:
```
# from the above graph we can see that our Randomforest model predicting values verym
# will consider Random forest as a perect model for this dataset
```

In [66]:
```
# lets check r2_score
```

In [67]:
```
from sklearn import metrics
```

In [68]:
```
score1 = metrics.r2_score(ytest,ypred1)
score2 = metrics.r2_score(ytest,ypred2)
score3 = metrics.r2_score(ytest,ypred3)
score4 = metrics.r2_score(ytest,ypred4)
score5 = metrics.r2_score(ytest,ypred5)
score6 = metrics.r2_score(ytest,ypred6)
score7 = metrics.r2_score(ytest,ypred7)
score8 = metrics.r2_score(ytest,ypred8)
score9 = metrics.r2_score(ytest,ypred9)
```

In [69]:
```
print(score1,score2,score3,score4,score5,score6,score7,score8,score9)
```

```
0.7507971412700817 0.7506563687875405 0.750934406678438 0.758997722095672 0.8451688
154897494 -0.00041614293302805194 0.6729703872437358 0.7930620507619563 0.793325466
7272512
```

In [70]:
```
# here we can see that the score5 i.e Random forest provides the highest accuracy wit
```

In [71]:
```
# lets check MSE
```

In [72]:
```
s1 = metrics.mean_absolute_error(ytest,ypred1)
s2 = metrics.mean_absolute_error(ytest,ypred2)
s3 = metrics.mean_absolute_error(ytest,ypred3)
s4 = metrics.mean_absolute_error(ytest,ypred4)
s5 = metrics.mean_absolute_error(ytest,ypred5)
s6 = metrics.mean_absolute_error(ytest,ypred6)
s7 = metrics.mean_absolute_error(ytest,ypred7)
s8 = metrics.mean_absolute_error(ytest,ypred8)
s9 = metrics.mean_absolute_error(ytest,ypred9)
```

In [73]:
```
print(s1,s2,s3,s4,s5,s6,s7,s8,s9)
```

```
2035.256625030842 2043.9646047550943 2038.4926928105774 683.8235294117648 762.35294
11764706 4410.234293633487 1903.6764705882354 1302.3837261779981 1201.2435625861672
```

In [74]:
```
# The Mean Squared Error (MSE) - given lowest by s5 i.e 5th model (random Forest)
# here lower value of MSE is better the performance of the model
# s5 provides 757.7205882352941 which is lowest from other models
```

In [75]:
```
df.columns
```

Out[75]:
```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice',
       'Age_category'],
      dtype='object')
```

In [76]:
```
# further we will use Randomforest  for the redictition of the new data.
```

In [77]:
```
# we will create a small data in python dictionary

data = {'age':60,
        'Diabetes':0,
       'Blood Pressure':1,
       'AnyTransplants':0,
       'AnyChronicDiseases':0,
       'Height':162,
       'weight':70,
       'KnownAllergies':0,
       'HistoryOfCancerInFamily':0,
       'NumberOfMajorSurgeries':0,
       'Age_category':2}

ab = pd.DataFrame(data,index=[0])
ab
```

Out[77]:

	age	Diabetes	Blood Pressure	AnyTransplants	AnyChronicDiseases	Height	weight	KnownAllergies	Histor
0	60	0	1	0	0	162	70	0	

In [78]:
```
# now we will predict the charges based on the above data (ab)
# for prediction we will use our best model i.e Randomforest
```

In [79]:
```
new_pred = rf.predict(ab)
print(new_pred)
```

```
[33820.]
```

In [80]:
```
# lets save the model using job lib
# this stage we will call it as Deployment stage.
# before deployment first we have to train our model on entire dataset
# since erlier we found out that Random forest Regressor is best model
```

In [81]:
```
rf = RandomForestRegressor()
rf.fit(x,y)
```

Out[81]:
```
RandomForestRegressor()
```

In [82]:
```
import joblib
```

In [83]:
```
joblib.dump(rf,'model_joblib_rf')
```

Out[83]:
```
['model_joblib_rf']
```

In [84]:
```
#using above model we can perform prediction
#creating a variable for model

model = joblib.load('model_joblib_rf')  # model loading
```

In [85]:
```
model.predict(x)
```

Out[85]:
```
array([24940., 29000., 23000., 28640., 23030., 22980., 21000., 15000.,
       23660., 23000., 28000., 25460., 15000., 32250., 23000., 29950.,
       23000., 25000., 15740., 28090., 15100., 28040., 23000., 34970.,
       21020., 28000., 23000., 21020., 15110., 19450., 15160., 15000.,
       28050., 23000., 25000., 30090., 16650., 27960., 15170., 29000.,
       15000., 23000., 25820., 34960., 25000., 15010., 23200., 28000.,
       28000., 28310., 25000., 23000., 28910., 28000., 22530., 23660.,
       23000., 25000., 15380., 23000., 28000., 15850., 28070., 15960.,
       23020., 21020., 15570., 30060., 25000., 28000., 28070., 15080.,
       23000., 28000., 23000., 25000., 18550., 23000., 25000., 25130.,
       16180., 28760., 21020., 25630., 31000., 28000., 28000., 15110.,
       23000., 23600., 25000., 24370., 15090., 31210., 15440., 15100.,
       23000., 25000., 24790., 20540., 29210., 23000., 15250., 23000.,
       25130., 20650., 28000., 25000., 25900., 22100., 28000., 28820.,
       23000., 15420., 23000., 29880., 30580., 28370., 16370., 15240.,
       28000., 23200., 15090., 23270., 15000., 23000., 28140., 29000.,
       30930., 28000., 21040., 21500., 28000., 29000., 23000., 28120.,
       24370., 28000., 34930., 28490., 23090., 15000., 35000., 25000.,
       15010., 24800., 34650., 15520., 23090., 29000., 15000., 23000.,
```

In [86]:
```
#above result and predicted results are different
#since here we have trained model on entire dataset
#so it is more accurate than train test split model
```

lets make GUI: For Insurance Cost Prediction

In [87]:
```
# harges for newly arrived customer
from tkinter import *
```

In [88]:
```
import joblib #saved model is sucessfully imported
```

In [92]:

```
# here we enter the function


def show_entry():
    p1 = float(e1.get())
    p2 = float(e2.get())
    p3 = float(e3.get())
    p4 = float(e4.get())
    p5 = float(e5.get())
    p6 = float(e6.get())
    p7 = float(e7.get())
    p8 = float(e8.get())
    p9 = float(e9.get())
    p10 = float(e10.get())
    p11= float(e11.get())

    model = joblib.load('model_joblib_rf')  #load the model using joblib with model n
    result =model.predict([[p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11]])

    Label(master,text="Medical Insurance Cost").grid(row=12)
    Label(master,text=result).grid(row=13)


master = Tk()
master.title("Predict your Medical Insurance Permium ") # title for GUI

#creating labels for input from user

label = Label(master,text = "Insurance Amount Prediction ", bg='black',
              fg='white').grid(row=0,columnspan=2)

Label(master,text = "Enter your Age").grid(row=1)
Label(master,text = "Do you have any Diabetes - Yes press 1 / No press 0 ").grid(row=
Label(master,text = "Do you have Blood Pressure - Yes press 1 / No press 0 ").grid(ro
Label(master, text = "Did you had any transplant? - Yes press 1 / No press 0 ").grid(
Label(master,text = "Do you have any Chronic Diseases? - Yes press 1 / No press 0 ").
Label(master,text = " Your Height in cm ").grid(row=6)
Label(master,text = "Your weight in kg ").grid(row=7)
Label(master,text = "Do you have any Allergies? - Yes press 1 / No press 0 ").grid(ro
Label(master,text = "Any of the family member had Cancer? - Yes press 1 / No press 0
Label(master,text = "Specify the number of Major Surgeries ").grid(row=10)
Label(master,text = "Mention the Age Crietria <29 press 3, 30-40 = 1,41-55 = 0, >=56

#creating objects for 6 entries from user

e1 = Entry(master)
e2 = Entry(master)
e3 = Entry(master)
e4 = Entry(master)
e5 = Entry(master)
e6 = Entry(master)
e7 = Entry(master)
e8 = Entry(master)
e9 = Entry(master)
e10 = Entry(master)
e11= Entry(master)

#with help of grid creating blank boxes to input data

e1.grid(row=1,column=1)
e2.grid(row=2,column=1)
e3.grid(row=3,column=1)
e4.grid(row=4,column=1)
e5.grid(row=5,column=1)
e6.grid(row=6,column=1)
```

```
e7.grid(row=7,column=1)
e8.grid(row=8,column=1)
e9.grid(row=9,column=1)
e10.grid(row=10,column=1)
e11.grid(row=11,column=1)

#creating button

Button(master,text="Predict",command=show_entry).grid()

mainloop()
```

In [ ]: