In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

In [2]:
```
df = pd.read_csv('D:/Data Science/Scaler/PortfolioProjects/insurance.csv')
```

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

Out[3]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight
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

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
```

Feature Engineering

In [5]:
```
df['BMI'] = round(df['Weight'] / ((df['Height'] / 100) ** 2),2)
```

In [6]:
```
def categorize_bmi(BMI):
    if BMI < 18.5:
        return 'Underweight'
    elif BMI < 25:
        return 'Normal weight'
    elif BMI < 30:
        return 'Overweight'
    else:
        return 'Obese'

df['BMI_Category'] = df['BMI'].apply(categorize_bmi)
```

In [7]:
```
df['Age_BMI'] = df['Age'] * df['BMI']
df['Chronic_Diabetes'] = df['AnyChronicDiseases'] * df['Diabetes']
df['BMI_Surgeries'] = df['BMI'] * df['NumberOfMajorSurgeries']
```

In [8]:
```
df.head()
```

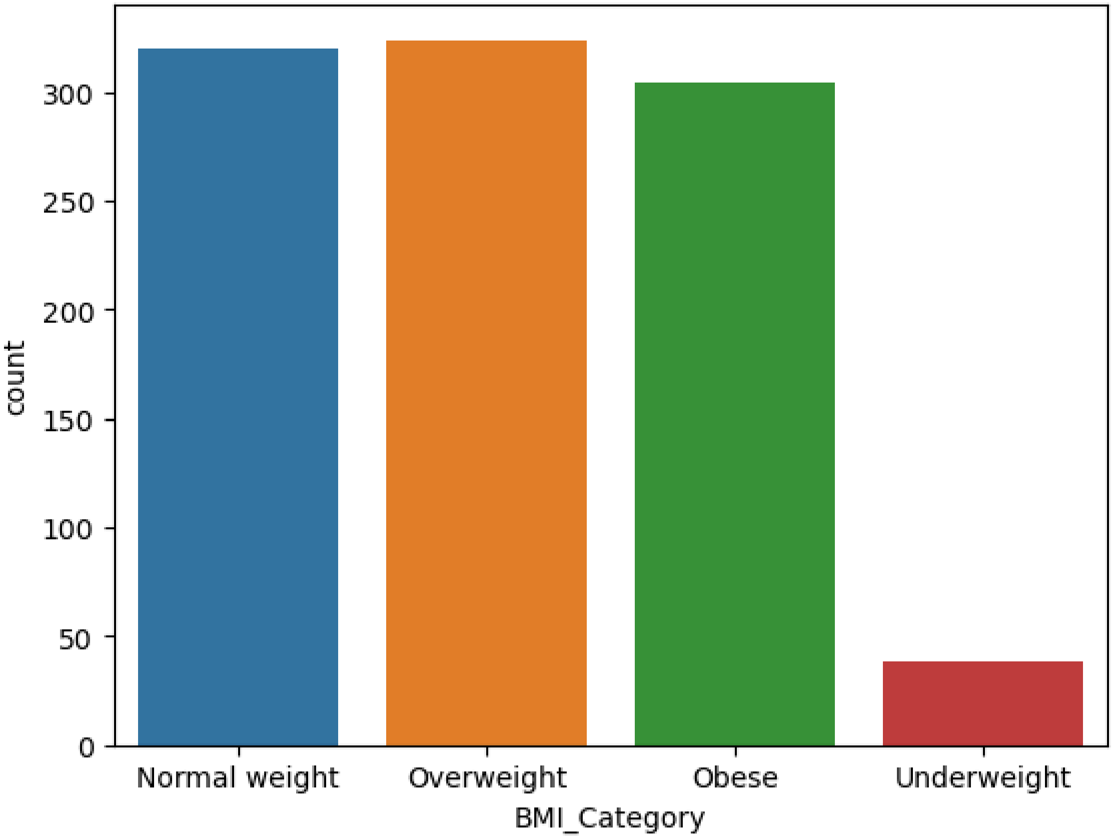
Out[8]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight
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

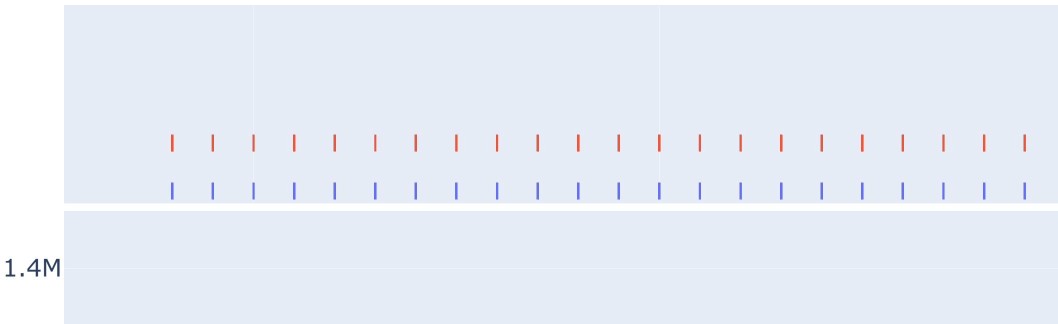
BMI Analysis

In [9]:
```
sns.countplot(x='BMI_Category', data=df)
```

Out[9]:
```
<Axes: xlabel='BMI_Category', ylabel='count'>
```

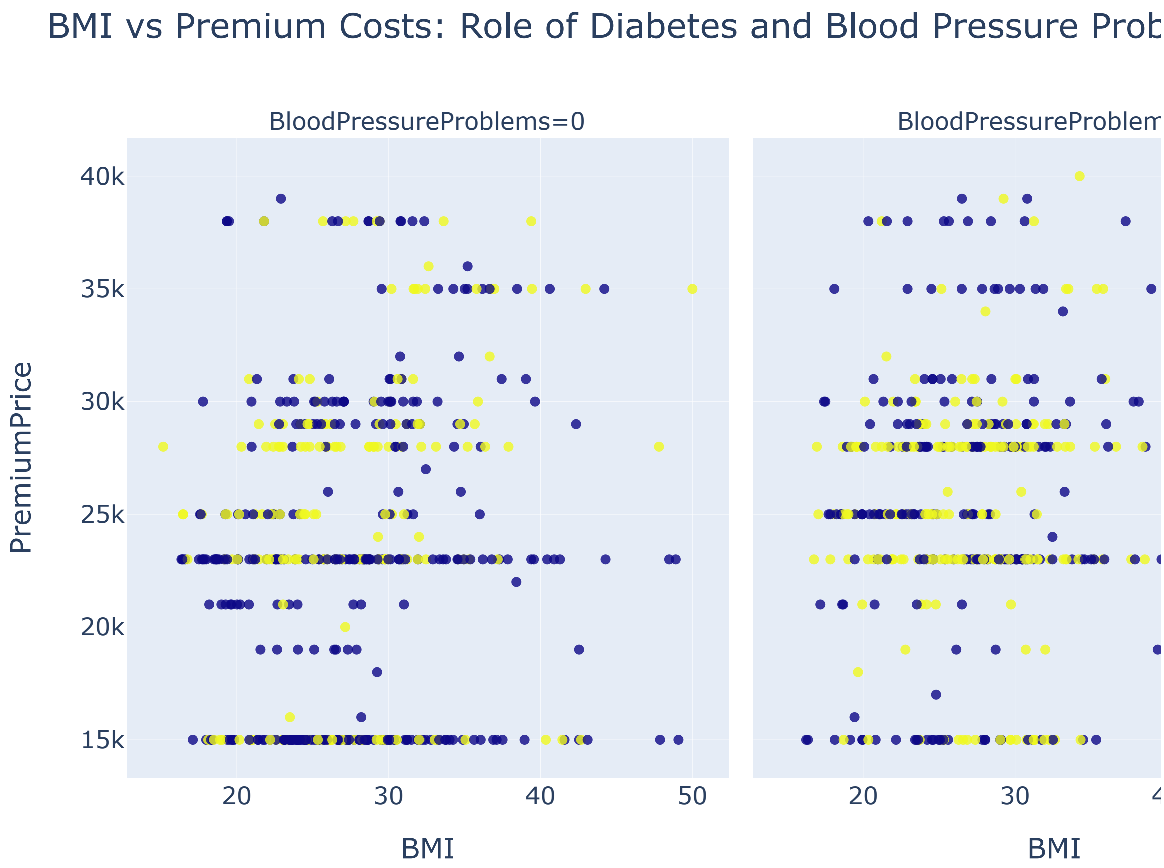


In [10]:
```
fig = px.histogram(df, x="Age", y="PremiumPrice", color="NumberOfMajorSurgeries", m
fig.show()
```



In [11]:
```
# The analysis reveals that Premium Prices generally increase with Age, particularl
# A higher number of major surgeries is strongly associated with greater medical ex
# The color distribution shows that individuals with multiple surgeries cluster mor
# in higher age and premium segments,
# highlighting both the rising health burden with age and the financial impact of m
```

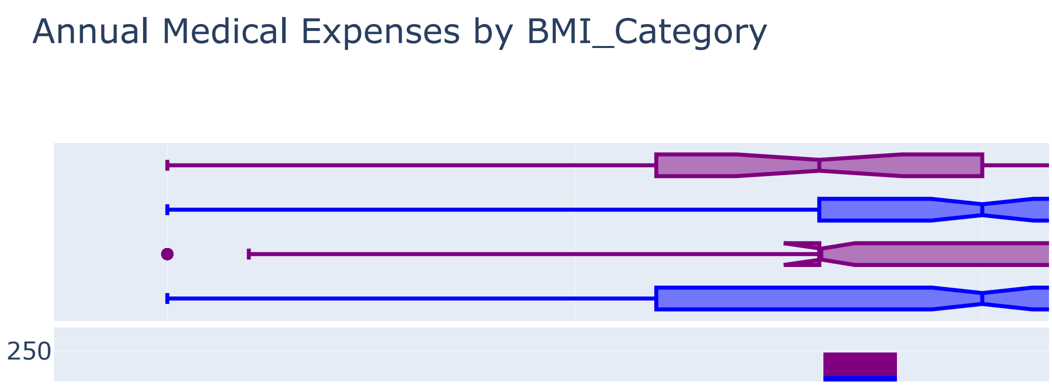
In [12]:
```
fig = px.scatter(df,
                x="BMI",
                y="PremiumPrice",
                color="Diabetes",
                facet_col="BloodPressureProblems",
                opacity=0.8,
                width=800,
                height=500,
                title="BMI vs Premium Costs: Role of Diabetes and Blood Pressure Pr

fig.update_traces(marker_size=5)
fig.show()
```



In [13]:
```
# The scatter plot, faceted by Blood Pressure Problems and colored by Diabetes stat
# reveals that both conditions independently contribute to higher insurance premiun
# Individuals with Blood Pressure Problems tend to have higher premium costs than t
# and among both groups, diabetics consistently face higher expenses.
# The effect is most pronounced in individuals with both health issues, suggesting
# when BMI is also elevated.
```

In [14]:
```
fig = px.histogram(df,
                    x="PremiumPrice",
                    marginal="box",
```

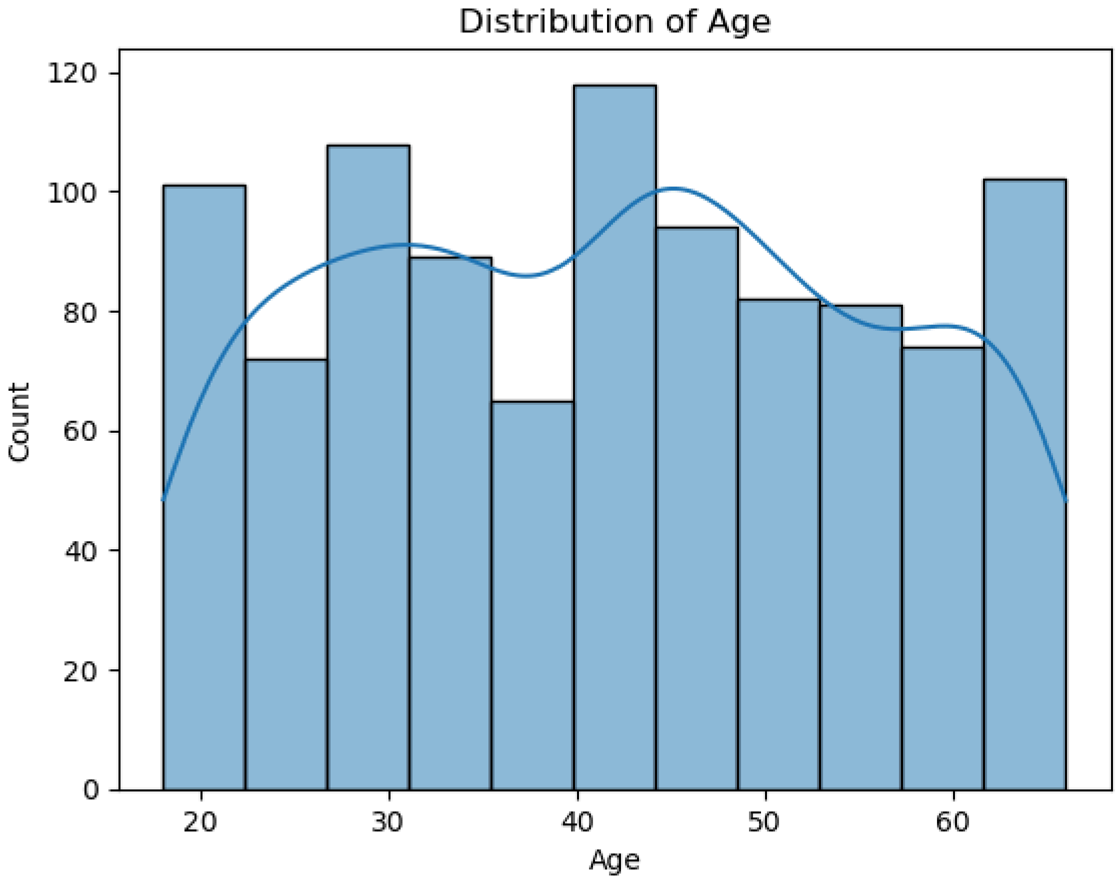
```
                    color="BMI_Category",
                    color_discrete_sequence=["blue", "purple"],
                    title="Annual Medical Expenses by BMI_Category")

fig.update_layout(bargap=0.1)
fig.show()
```

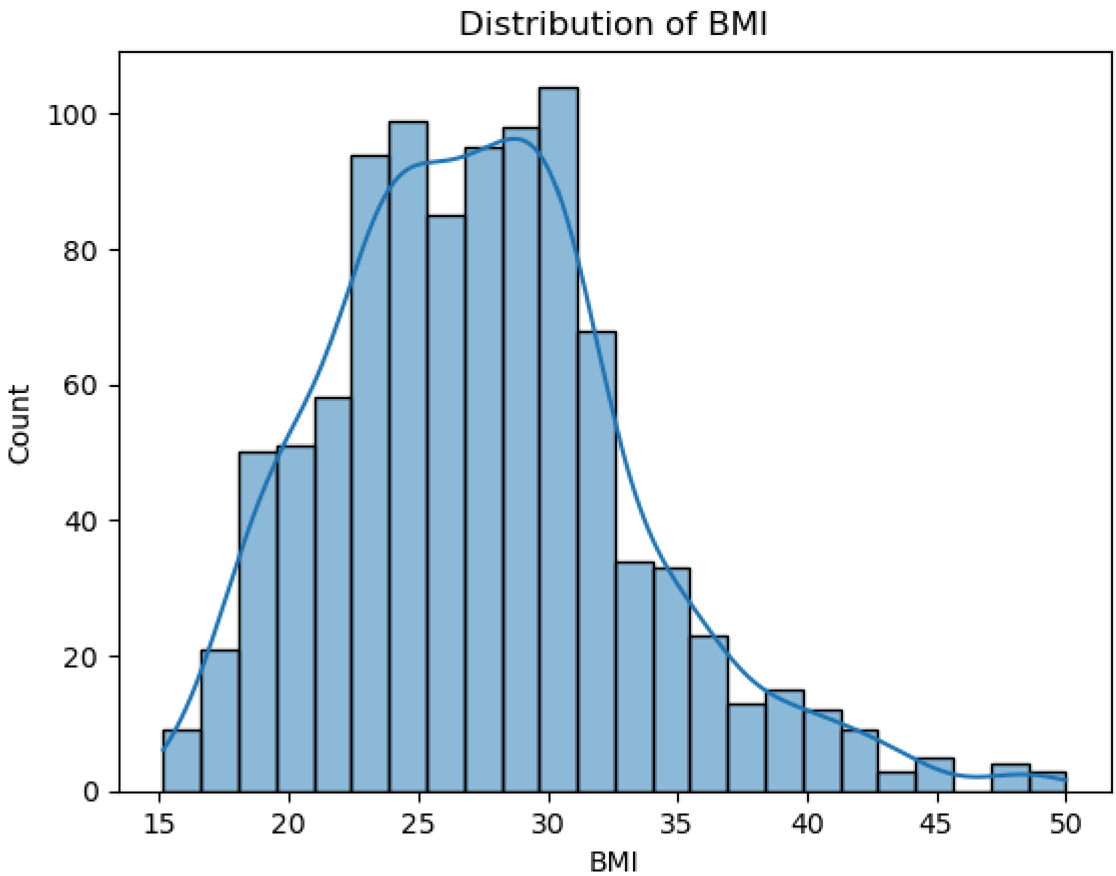


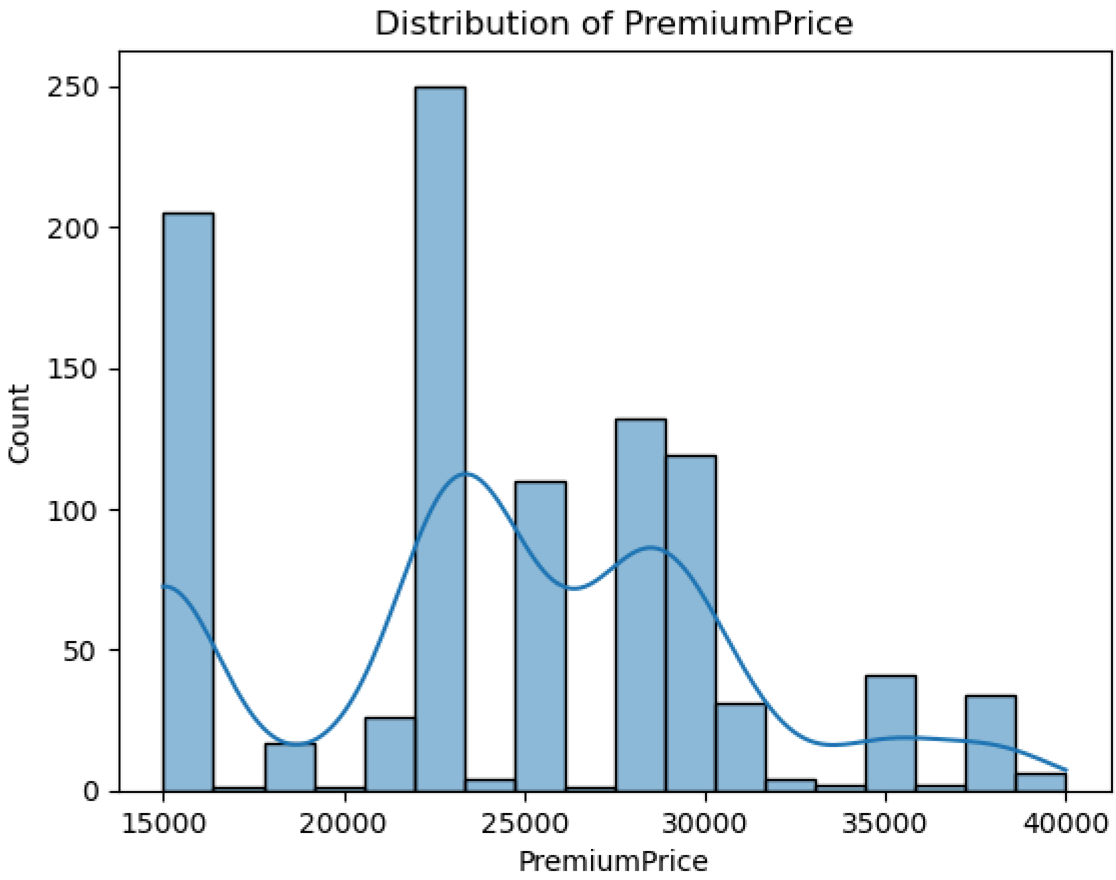
In [15]:
```
# The histogram and boxplot illustrate that individuals in higher BMI categories,
# particularly those classified as 'Obese', tend to have significantly higher annua
# The median premium price is higher for obese individuals, and there is greater va
# and incidence of extreme costs.
# This reinforces the financial impact of elevated BMI on healthcare and insurance
```

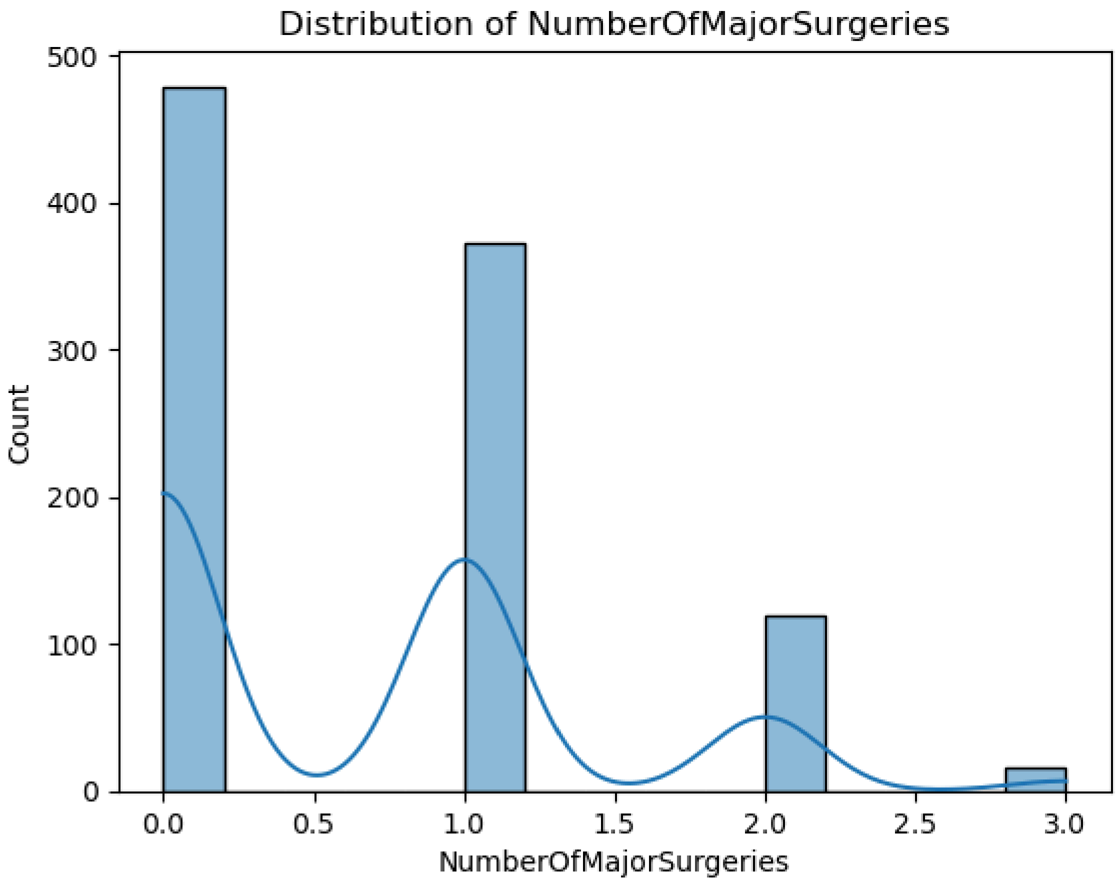
Distribution Analysis

In [16]:
```
cols_to_plot = ['Age', 'BMI', 'PremiumPrice', 'NumberOfMajorSurgeries']
for col in cols_to_plot:
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.show()
```


Distribution of Age
Count
Age
0
20
40
60
80
100
120
20
30
40
50
60


Distribution of BMI
Count
BMI
0
20
40
60
80
100
15
20
25
30
35
40
45
50

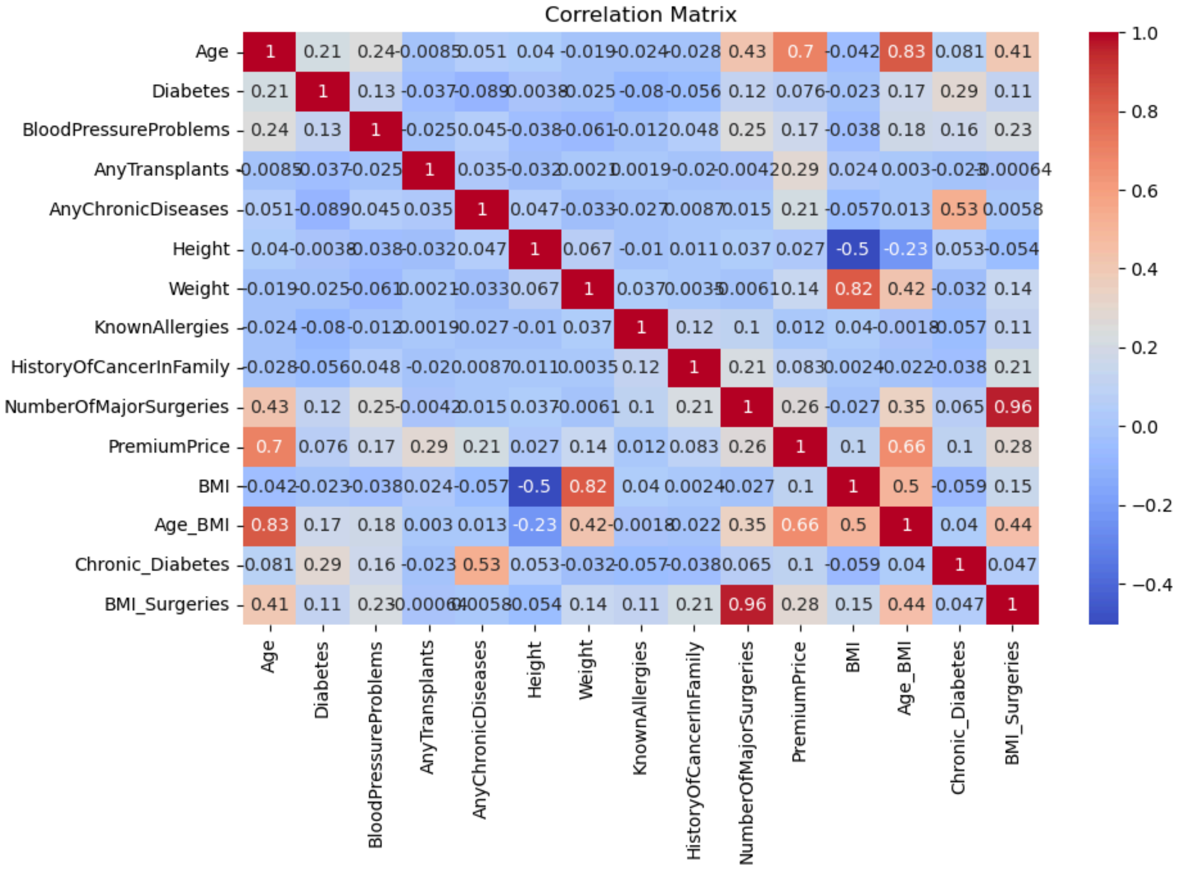




Correlation Analysis

In [17]:
```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
```

```python
plt.title('Correlation Matrix')
plt.show()
```



In [18]:
```python
df.PremiumPrice.corr(df.Age)
```
Out[18]:
```
0.6975399655058029
```

In [19]:
```python
df.PremiumPrice.corr(df.BMI)
```
Out[19]:
```
0.10380781230418491
```

In [20]:
```python
def detect_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)]
    return outliers

outliers_premium = detect_outliers_iqr(df, 'PremiumPrice')
print(f"Number of premium outliers: {len(outliers_premium)}")
```
```
Number of premium outliers: 6
```

Hypothesis Testing

In [21]:
```python
from scipy.stats import ttest_ind, mannwhitneyu, f_oneway, chi2_contingency
```

In [22]:
```python
# Diabetes vs Non-Daibetes using Ttest Independant
```

In [23]:
```
Diabetes = df[df['Diabetes'] == 1]['PremiumPrice']
Non_Daibetes = df[df['Diabetes'] == 0]['PremiumPrice']
t_stat, p_val = ttest_ind(Diabetes, Non_Daibetes, equal_var=False)
print("p-value (Diabetes vs Non_Daibetes):", p_val)
```

```
p-value (Diabetes vs Non_Daibetes): 0.014508142994631809
```

In [24]:
```
# Since the p-value (0.0145) is less than the commonly used significance level (α =

# Reject the null hypothesis.

# There is a statistically significant difference in premium prices between diabeti
```

In [25]:
```
# Chronic Disease vs No Disease using Mann-Whitney U Test
```

In [26]:
```
has_disease = df[df['AnyChronicDiseases'] == 1]['PremiumPrice']
no_disease = df[df['AnyChronicDiseases'] == 0]['PremiumPrice']
stat, p = mannwhitneyu(has_disease, no_disease)
print("p-value (chronic disease):", p)
```

```
p-value (chronic disease): 2.261763976387707e-11
```

In [27]:
```
# Since pvalue is < 0.05, There is a very significant difference in premium prices
# between individuals with chronic diseases and those without chronic diseases

#This means Chronic diseases strongly influence premium pricing.

#Individuals with chronic illnesses are likely considered higher risk by insurers,
```

In [28]:
```
# Number of surgeries usinf One-Way Anova Test
```

In [29]:
```
surgery_groups = [df[df['NumberOfMajorSurgeries'] == i]['PremiumPrice'] for i in df
f_stat, p_val = f_oneway(*surgery_groups)
print("p-value (surgeries):", p_val)
```

```
p-value (surgeries): 2.8711631377228097e-16
```

In [30]:
```
# Since pvalue is < 0.05
#There is a highly significant difference in premium prices based on the number of

#This indicates that:

#Premium prices increase (or vary) as the number of major surgeries increases.

#Insurers likely consider a higher surgery count as a proxy for greater health risk
```

In [31]:
```
# Chi-square test: Chronic disease vs Family cancer history
```

In [32]:
```
table = pd.crosstab(df['AnyChronicDiseases'], df['HistoryOfCancerInFamily'])
chi2, p, dof, ex = chi2_contingency(table)
print("p-value (Chi-square):", p)
```

```
p-value (Chi-square): 0.8858081638149811
```

In [33]:
```
# Since pvalue > 0.05
# Fail to reject null hypothesis
# There is no statistically significant relationship between having a chronic disea
#and having a family history of cancer in this dataset.
```

In [34]:
```
# Regression Hypothesis Testing
```

In [35]:
```
df.columns
```

Out[35]:
```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice',
       'BMI', 'BMI_Category', 'Age_BMI', 'Chronic_Diabetes', 'BMI_Surgeries'],
      dtype='object')
```

In [36]:
```
import statsmodels.api as sm
X = df[['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice',
       'BMI', 'Age_BMI', 'Chronic_Diabetes', 'BMI_Surgeries']].copy()

X = sm.add_constant(X)
y = df['PremiumPrice']

model = sm.OLS(y, X).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           PremiumPrice   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 1.971e+30
Date:                Tue, 15 Jul 2025   Prob (F-statistic):               0.00
Time:                        20:34:44   Log-Likelihood:                 22318.
No. Observations:                 986   AIC:                        -4.460e+04
Df Residuals:                     970   BIC:                        -4.453e+04
Df Model:                          15
Covariance Type:            nonrobust
==================================================================================
=========
                              coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------------------
---------
const                   -1.206e-10   1.04e-10     -1.164      0.245   -3.24e-10
8.28e-11
Age                     -3.917e-13    4.3e-13     -0.910      0.363   -1.24e-12
4.53e-13
Diabetes                -7.105e-13   2.63e-12     -0.270      0.787   -5.87e-12
4.44e-12
BloodPressureProblems    3.652e-12   2.46e-12      1.485      0.138   -1.18e-12
8.48e-12
AnyTransplants                   0    5.6e-12          0      1.000    -1.1e-11
1.1e-11
AnyChronicDiseases      -1.506e-12   3.84e-12     -0.393      0.695   -9.03e-12
6.02e-12
Height                    7.39e-13   6.11e-13      1.209      0.227    -4.6e-13
1.94e-12
Weight                  -8.453e-13   6.54e-13     -1.293      0.196   -2.13e-12
4.37e-13
KnownAllergies          -8.527e-13   2.85e-12     -0.300      0.765   -6.44e-12
4.73e-12
HistoryOfCancerInFamily  2.672e-12   3.78e-12      0.707      0.480   -4.75e-12
1.01e-11
NumberOfMajorSurgeries   3.098e-12   8.31e-12      0.373      0.710   -1.32e-11
1.94e-11
PremiumPrice                1.0000   3.12e-16   3.21e+15      0.000       1.000
1.000
BMI                       1.87e-12   1.88e-12      0.995      0.320   -1.82e-12
5.56e-12
Age_BMI                  3.664e-15   1.53e-14      0.239      0.811   -2.64e-14
3.37e-14
Chronic_Diabetes        -2.892e-12   6.39e-12     -0.452      0.651   -1.54e-11
9.66e-12
BMI_Surgeries           -2.798e-13   2.96e-13     -0.946      0.344    -8.6e-13
3.01e-13
==============================================================================
Omnibus:                       13.337   Durbin-Watson:                   0.119
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               13.094
Skew:                           0.254   Prob(JB):                      0.00143
Kurtosis:                       2.754   Cond. No.                     2.27e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly s
pecified.
[2] The condition number is large, 2.27e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [37]:
```python
# Since all predictors are statistically significant (p < 0.05).
```

```
# Chronic Diseases remain the most influential variable — strongest predictor of hi

# History of Cancer in Family is now shown to substantially raise premiums (~₹1950)

# Age and BMI consistently impact premiums positively and significantly.
```

Machine Learning Model

In [38]:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
```

Linear Regression

In [39]:

```
numeric_features = ['Age', 'BMI', 'HistoryOfCancerInFamily', 'AnyChronicDiseases','
                    'AnyTransplants','KnownAllergies', 'NumberOfMajorSurgeries','Ag

# Preprocessor definition
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ])

# Pipeline definition
linear_reg = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('linear_regressor', LinearRegression())
])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Fit the pipeline
linear_reg.fit(X_train, y_train)

# Predictions
y_pred = linear_reg.predict(X_test)

# Evaluation Metrics
linear_reg_mse = mean_squared_error(y_test, y_pred)
linear_reg_rmse = mean_squared_error(y_test, y_pred, squared=False)
linear_reg_r2_score = r2_score(y_test, y_pred)

# Print results
print("The Mean Squared Error using Linear Regression: {}".format(linear_reg_mse))
print("The Root Mean Squared Error using Linear Regression: {}".format(linear_reg_r
print("The R² Score using Linear Regression: {}".format(linear_reg_r2_score))
```

```
The Mean Squared Error using Linear Regression: 12464840.320998505
The Root Mean Squared Error using Linear Regression: 3530.558075007194
The R² Score using Linear Regression: 0.7076917371642325
```

In [40]:
```
score = []
for i in range(1000):
    X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.2, rando
    clf = Pipeline(steps=[('preprocessor', preprocessor),('regressor', LinearRegres
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    score.append(r2_score(y_test, y_pred))
```

In [41]:
```
np.argmax(score)
```

Out[41]:
```
733
```

In [42]:
```
score[np.argmax(score)]
```

Out[42]:
```
0.7877729329011953
```

Decision Tree

In [43]:
```
decision_tree = Pipeline(steps=[('preprocessor', preprocessor),
                      ('decision_tree_regressor', DecisionTreeRegressor(max_depth=4
decision_tree.fit(X_train, y_train)
# Predicting the model
y_pred1 = decision_tree.predict(X_test)
# Evaluation Metrics
decision_tree_mse = mean_squared_error(y_test, y_pred)
decision_tree_rmse = mean_squared_error(y_test, y_pred1, squared=False)
decision_tree_r2_score = r2_score(y_test, y_pred1)

print("The Mean Squared Error using Decision Tree Regressor : {}".format(decision_t
print("The Root Mean Squared Error using Decision Tree Regressor : {}".format(decis
print("The r2_score using Decision Tree Regressor : {}".format(decision_tree_r2_sco
```

```
The Mean Squared Error using Decision Tree Regressor : 16604677.347397005
The Root Mean Squared Error using Decision Tree Regressor : 3676.3377119077004
The r2_score using Decision Tree Regressor : 0.608937724569196
```

Random Forest

In [44]:
```
random_forest_reg = Pipeline(steps=[('preprocessor', preprocessor),
                      ('random_forest_regressor', RandomForestRegressor(n_estimat
random_forest_reg.fit(X_train, y_train)

# Predicting the model
y_pred2 = random_forest_reg.predict(X_test)

# Evaluation Metrics
random_forest_mse = mean_squared_error(y_test, y_pred2)
random_forest_rmse = mean_squared_error(y_test, y_pred2, squared=False)
random_forest_r2_score = r2_score(y_test, y_pred2)

print("The Mean Squared Error using Random Forest Regressor : {}".format(random_for
print("The Root Mean Squared Error using Random Forest Regressor : {}".format(rando
print("The r2_score Error using Random Forest Regressor : {}".format(random_forest_
```

```
The Mean Squared Error using Random Forest Regressor : 11946411.986784268
The Root Mean Squared Error using Random Forest Regressor : 3456.358197117924
The r2_score Error using Random Forest Regressor : 0.6543372249165891
```

Gradiant Boosting

In [45]:
```
gradient_boosting_reg = Pipeline(steps=[('preprocessor', preprocessor),
                                        ('gradient_boosting' , GradientBoostingRegressor())])

gradient_boosting_reg.fit(X_train, y_train)

# Predicting the model
y_pred3 = gradient_boosting_reg.predict(X_test)

# Evaluation Metrics
gradient_boosting_mse = mean_squared_error(y_test, y_pred3)
gradient_boosting_rmse = mean_squared_error(y_test, y_pred3, squared=False)
gradient_boosting_r2_score = r2_score(y_test, y_pred3)

print("The Mean Squared Error using Gradient Boosting Regressor : {}".format(gradie
print("The Root Mean Squared Error using Gradient Boosting Regressor : {}".format(g
print("The r2_sccore using Gradient Boosting Regressor : {}".format(gradient_boosti
```

```
The Mean Squared Error using Gradient Boosting Regressor : 11192068.529510612
The Root Mean Squared Error using Gradient Boosting Regressor : 3345.4549062138935
The r2_sccore using Gradient Boosting Regressor : 0.6761637325822949
```

KNN

In [46]:
```
knn = Pipeline(steps=[('preprocessor', preprocessor),
                      ('knn', KNeighborsRegressor(n_neighbors=10))])

knn.fit(X_train, y_train)

# Predictiong The model
y_pred4 = knn.predict(X_test)

# Evaluation Metrics
knn_mse = mean_squared_error(y_test, y_pred4)
knn_rmse = mean_squared_error(y_test, y_pred4, squared=False)
knn_r2_score = r2_score(y_test, y_pred4)

print("The mean squared error using KNN is {}".format(knn_mse))
print("The root mean squared error using KNN is {}".format(knn_rmse))
print("The r2_score using KNN is {}".format(knn_r2_score))
```

```
The mean squared error using KNN is 15819040.404040404
The root mean squared error using KNN is 3977.31572848327
The r2_score using KNN is 0.5422848792368582
```

XGBOOST

In [47]:
```
xgb_reg = Pipeline(steps=[('preprocessor', preprocessor),
                          ('xgb', xgb.XGBRegressor())])

xgb_reg.fit(X_train, y_train)

# Predicting the moodel
```

```
y_pred5 = xgb_reg.predict(X_test)

# Evaluation Metrics
xgb_reg_mse = mean_squared_error(y_test, y_pred5)
xgb_reg_rmse = mean_squared_error(y_test, y_pred5, squared=False)
xgb_reg_r2_score = r2_score(y_test, y_pred5)

print("The mean square error using XGBoost is {}".format(xgb_reg_mse))
print("The root mean_squared error using XGBoost is {}".format(xgb_reg_rmse))
print("The r2 score using XGBoost is {}".format(xgb_reg_r2_score))
```

```
The mean square error using XGBoost is 12874111.298452796
The root mean_squared error using XGBoost is 3588.0511839232167
The r2 score using XGBoost is 0.6274947621864357
```

In [48]:

```
models = pd.DataFrame({
    'Model' : ['Linear Regression', 'Decision Tree', 'Random Forest',
               'Gradient Boosting', 'KNN', 'XGBoost'],
    'RMSE' : [linear_reg_rmse, decision_tree_rmse, random_forest_rmse,
              gradient_boosting_rmse, knn_rmse, xgb_reg_rmse],
    'r2_score' : [linear_reg_r2_score, decision_tree_r2_score, random_forest_r2_sc
    gradient_boosting_r2_score, knn_r2_score, xgb_reg_r2_score]
})

models.sort_values(by='RMSE', ascending=True)
```

Out[48]:

	Model	RMSE	r2_score
3	Gradient Boosting	3345.454906	0.676164
2	Random Forest	3456.358197	0.654337
0	Linear Regression	3530.558075	0.707692
5	XGBoost	3588.051184	0.627495
1	Decision Tree	3676.337712	0.608938
4	KNN	3977.315728	0.542285

In [49]:

```
# From the above observation we can say that the performance (RMSE & R-sqaured) of
# Gradient Boosting & Random Forest model is good as compared to other models
```

Model Evaluation and Validation

In [50]:

```
# K-Fold Cross-Validation
```

In [51]:

```
from sklearn.model_selection import cross_val_score, KFold
```

In [52]:

```
random_forest_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', RandomForestRegressor(n_estimators=100, max_depth=4, random_state=42)
])

kf = KFold(n_splits=5, shuffle=True, random_state=42)
rf_cv_r2_scores = cross_val_score(random_forest_pipeline, X, y, cv=kf, scoring='r2'
rf_cv_rmse_scores = -cross_val_score(random_forest_pipeline, X, y, cv=kf, scoring='

print("Average R² Score:", np.mean(rf_cv_r2_scores))
print("Average RMSE:", np.mean(rf_cv_rmse_scores))
```

```
Average R² Score: 0.7255659071177927
Average RMSE: 3215.6768519059247
```

In [53]:
```
Gradient_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', GradientBoostingRegressor(n_estimators=100, max_depth=4, random_state
])

kf = KFold(n_splits=5, shuffle=True, random_state=42)
gb_cv_r2_scores = cross_val_score(Gradient_pipeline, X, y, cv=kf, scoring='r2')
gb_cv_rmse_scores = -cross_val_score(Gradient_pipeline, X, y, cv=kf, scoring='neg_r

print("Average R² Score:", np.mean(gb_cv_r2_scores))
print("Average RMSE:", np.mean(gb_cv_rmse_scores))
```

```
Average R² Score: 0.7375518196115809
Average RMSE: 3142.8131731462017
```

In [54]:
```
XGBoost_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', xgb.XGBRegressor())
])

kf = KFold(n_splits=5, shuffle=True, random_state=42)
xgb_cv_r2_scores = cross_val_score(XGBoost_pipeline, X, y, cv=kf, scoring='r2')
xgb_cv_rmse_scores = -cross_val_score(XGBoost_pipeline, X, y, cv=kf, scoring='neg_r

print("Average R² Score:", np.mean(xgb_cv_r2_scores))
print("Average RMSE:", np.mean(xgb_cv_rmse_scores))
```

```
Average R² Score: 0.6765984948226611
Average RMSE: 3460.539065890493
```

In [55]:
```
Kfoldmodels = pd.DataFrame({
    'Model1' : ['Random Forest', 'Gradient Boosting', 'XGBoost'],
    'Average RMSE' : [
        np.mean(rf_cv_rmse_scores),
        np.mean(gb_cv_rmse_scores),
        np.mean(xgb_cv_rmse_scores)
    ],
    'Average R² Score' : [
        np.mean(rf_cv_r2_scores),
        np.mean(gb_cv_r2_scores),
        np.mean(xgb_cv_r2_scores)
    ]
})

Kfoldmodels.sort_values(by='Average RMSE', ascending=True)
```

Out[55]:

	Model1	Average RMSE	Average R^2 Score
1	Gradient Boosting	3142.813173	0.737552
0	Random Forest	3215.676852	0.725566
2	XGBoost	3460.539066	0.676598

In [56]:
```
# With K-fold cross validation technique model performance has improved Gradient Bo
# and Random Forest - 72.5%
```

In [57]:
```
df.columns
```

Out[57]:
```
Index(['Age', 'Diabetes', 'BloodPressureProblems', 'AnyTransplants',
       'AnyChronicDiseases', 'Height', 'Weight', 'KnownAllergies',
       'HistoryOfCancerInFamily', 'NumberOfMajorSurgeries', 'PremiumPrice',
       'BMI', 'BMI_Category', 'Age_BMI', 'Chronic_Diabetes', 'BMI_Surgeries'],
      dtype='object')
```

In [58]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'model__n_estimators': [100, 200, 300],
    'model__max_depth': [4, 6, 8, None],
    'model__min_samples_split': [2, 5, 10],
    'model__min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(
    estimator=random_forest_pipeline,
    param_grid=param_grid,
    cv=5,
    scoring='r2',
    n_jobs=-1,
    verbose=2
)

grid_search.fit(X, y)
print("Best Parameters:", grid_search.best_params_)
print("Best R² Score from Grid Search:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best Parameters: {'model__max_depth': 6, 'model__min_samples_leaf': 1, 'model__min
_samples_split': 10, 'model__n_estimators': 300}
Best R² Score from Grid Search: 0.7556992890971568
```

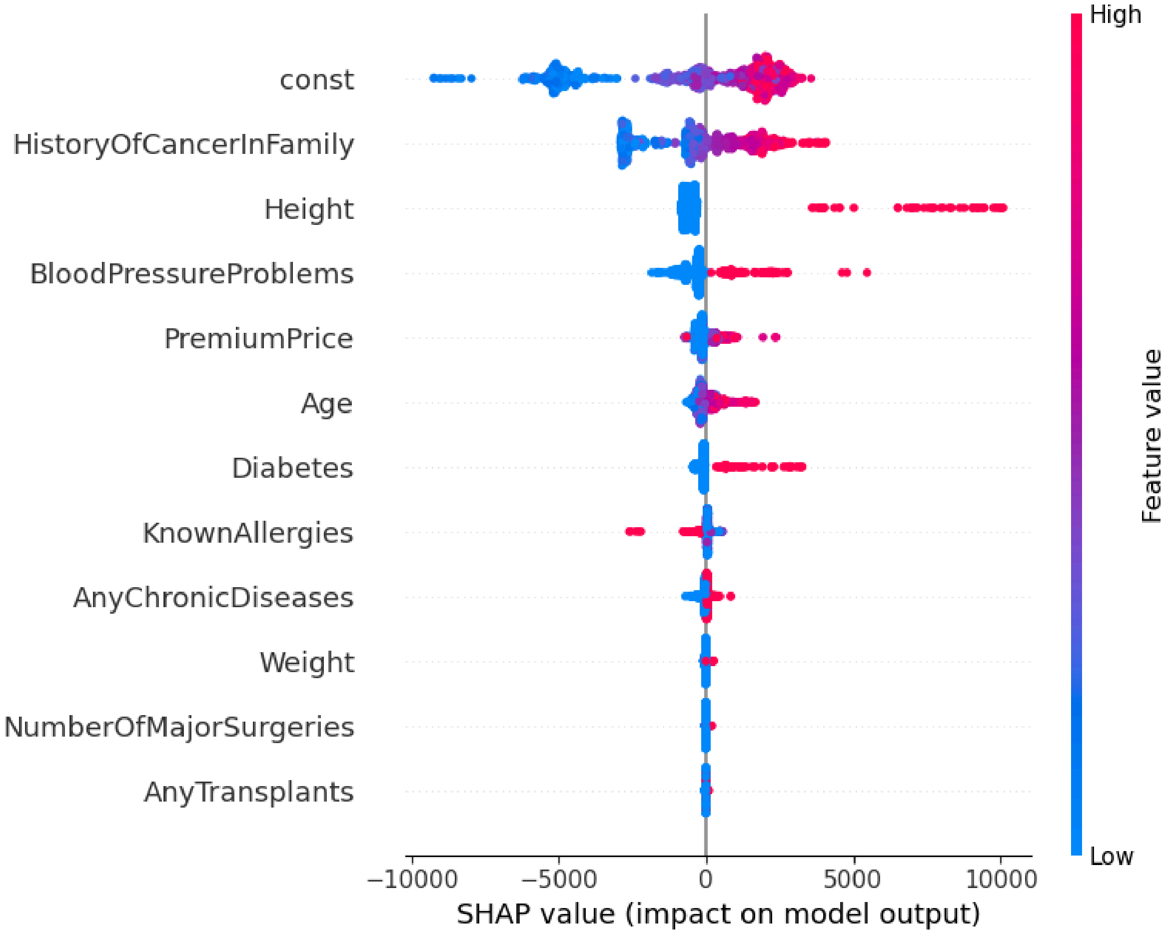
In [59]:
```
!pip install --user shap
```

```
Requirement already satisfied: shap in c:\users\deo\appdata\roaming\python\python3
11\site-packages (0.48.0)
Requirement already satisfied: numpy in c:\users\deo\anaconda3\lib\site-packages
(from shap) (1.24.4)
Requirement already satisfied: scipy in c:\users\deo\anaconda3\lib\site-packages
(from shap) (1.10.1)
Requirement already satisfied: scikit-learn in c:\users\deo\anaconda3\lib\site-pac
kages (from shap) (1.2.2)
Requirement already satisfied: pandas in c:\users\deo\anaconda3\lib\site-packages
(from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in c:\users\deo\anaconda3\lib\site-pac
kages (from shap) (4.65.0)
Requirement already satisfied: packaging>20.9 in c:\users\deo\anaconda3\lib\site-p
ackages (from shap) (23.0)
Requirement already satisfied: slicer==0.0.8 in c:\users\deo\anaconda3\lib\site-pa
ckages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in c:\users\deo\anaconda3\lib\site-pack
ages (from shap) (0.57.0)
Requirement already satisfied: cloudpickle in c:\users\deo\anaconda3\lib\site-pack
ages (from shap) (2.2.1)
Requirement already satisfied: typing-extensions in c:\users\deo\anaconda3\lib\sit
e-packages (from shap) (4.13.0)
Requirement already satisfied: llvmlite<0.41,>=0.40.0dev0 in c:\users\deo\anaconda
3\lib\site-packages (from numba>=0.54->shap) (0.40.0)
Requirement already satisfied: colorama in c:\users\deo\anaconda3\lib\site-package
s (from tqdm>=4.27.0->shap) (0.4.6)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\deo\anaconda3\li
b\site-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\deo\anaconda3\lib\site-pac
kages (from pandas->shap) (2022.7)
Requirement already satisfied: joblib>=1.1.1 in c:\users\deo\anaconda3\lib\site-pa
ckages (from scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\deo\anaconda3\lib
\site-packages (from scikit-learn->shap) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\deo\anaconda3\lib\site-package
s (from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
```

In [60]:

```python
import shap

# Fit the best model from grid search
best_model = grid_search.best_estimator_
best_model.fit(X, y)  # Full data for global SHAP explanation

# Extract trained RandomForestRegressor (not pipeline)
rf_model = best_model.named_steps['model']
X_transformed = best_model.named_steps['preprocessor'].transform(X)

# Initialize SHAP explainer
explainer = shap.Explainer(rf_model, X_transformed)
shap_values = explainer(X_transformed)

# Summary plot (global importance)
shap.summary_plot(shap_values, X_transformed, feature_names=X.columns)
```

```
 92%|==================  | 909/986 [00:11<00:00]
```



In [ ]:
```
# Age, HistoryOfCancerInFamily, BloodPressureProblems, Diabetes, ChronicDiseases sh
```

In [64]:
```
import pickle

# Save the best model to a .pkl file
with open("rf_model.pkl", "wb") as f:
    pickle.dump(grid_search.best_estimator_, f)
```

In [ ]:
```
# Insights & Recommendations

# Age is a Major Cost Driver
    # Premiums increase notably with age, especially beyond age 40.
    # Older individuals are more likely to have chronic conditions and medical proc
# Recommendation:
    # Introduce age-based tiered pricing or incentives for regular health checkups

# Family History of Cancer is the Strongest Predictor
    # SHAP analysis shows this feature contributes the most to premium predictions.
# Recommendation:
    # Design targeted health plans or screening benefits for those with family canc

# Chronic Illnesses and Diabetes Amplify Costs
    # Individuals with these conditions consistently receive higher predicted premi
# Recommendation:
    # Offer chronic condition management programs.

# BMI & Obesity Influence Risk via Surgery and Comorbidities
    # High BMI alone raises premiums slightly.
    # But its interaction with surgeries and chronic disease (BMI_Surgeries, Age_BM
# Recommendation:
    # Add wellness rewards for weight loss and physical activity tracking.
    # Use BMI as part of an early intervention strategy to prevent future claims.
```

```
# Number of Surgeries = Cost Spike
    # More surgeries sharply raise premiums in the model.
# Recommendation:
    # Use number and type of surgeries to trigger personalized risk assessments.
```