1. Introduction

In [ ]:
```
# Importing Libraries

import os

# Data Analysis and Data Visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Setting the style for seaborn
sns.set(style="whitegrid")

# Hypothesis Testing
from scipy import stats
import statsmodels.api as sm
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from scipy.stats import chi2_contingency

# ML Pre-Processing
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
import pickle

# ML Models
from sklearn.preprocessing import  StandardScaler,OneHotEncoder
from sklearn.model_selection import cross_val_score, KFold
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor, AdaBoostRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV, Lasso, LassoCV, ElasticNet, ElasticNetCV
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,r2_score
from sklearn.exceptions import ConvergenceWarning

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.inspection import permutation_importance
from sklearn.utils import resample


import shap

from xgboost import XGBRegressor
import lightgbm as lgb

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# OLD
# from tensorflow.keras.wrappers.scikit_learn import KerasRegressor

# NEW
from scikeras.wrappers import KerasRegressor

from statsmodels.stats.outliers_influence import variance_inflation_factor


import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

2. Exploratory Data Analysis (EDA)

2.1 - Data Overview

In [4]:
```
# Load the Dataset

df = pd.read_csv(r"M:\Module - 19 - DSML Portfolio Project\1 - Insurance Cost Prediction\insurance.csv")
```

In [5]:
```
df.head()
```

Out[5]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily	NumberOfM
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	

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

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

In [7]:
```
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 [8]:
```
# Missing Values

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

Out[8]:
```
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 [9]:
```
# Renaming a few column names

df = df.rename(columns={
    "PremiumPrice":"Premium_Price",
    "BloodPressureProblems":"Blood_Pressure_Problems",
    "AnyChronicDiseases":"Any_Chronic_Diseases",
    "AnyTransplants":"Any_Transplants",
    "HistoryOfCancerInFamily":"History_of_Cancer_in_Family",
    "KnownAllergies":"Known_Allergies",
    "NumberOfMajorSurgeries":"Number_of_Major_Surgeries"
})
```

In [10]:
```
# Identify columns

TARGET = "Premium_Price"

binary_cols = [c for c in df.columns if set(df[c].dropna().unique()).issubset({0,1})]
cont_cols = [c for c in df.columns if c not in binary_cols]
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['Age', 'Diabetes', 'Blood_Pressure_Problems', 'Any_Transplants',
       'Any_Chronic_Diseases', 'Height', 'Weight', 'Known_Allergies',
       'History_of_Cancer_in_Family', 'Number_of_Major_Surgeries',
       'Premium_Price'],
      dtype='object')
```

In [12]:
```
binary_cols
```

Out[12]:
```
['Diabetes',
 'Blood_Pressure_Problems',
 'Any_Transplants',
 'Any_Chronic_Diseases',
 'Known_Allergies',
 'History_of_Cancer_in_Family']
```

In [13]:
```
cont_cols
```

Out[13]: ['Age', 'Height', 'Weight', 'Number_of_Major_Surgeries', 'Premium_Price']

In [14]:
```
# Function to Summarize Continuous Variables

def num_summary(frame, cols):
    """
    Returns a summary table with count, mean, std, min, median, max, skew, kurtosis, missing%.
    """
    out = frame[cols].agg(['count','mean','std','min','median','max']).T
    out['missing%'] = 100 * (1 - out['count']/len(frame))
    out['skew'] = frame[cols].skew()
    out['kurtosis'] = frame[cols].kurtosis()
    return out.round(3)

summary_cont = num_summary(df, cont_cols)
summary_bin = df[binary_cols].sum().to_frame(name="Yes") #* `Yes` is Count of 1s
summary_bin["No"] = len(df) - summary_bin["Yes"] #* `No` is Count of 0s


print("Continuous Variables Summary:")
display(summary_cont)

print("\nBinary Variables Summary:")
display(summary_bin)
```

Continuous Variables Summary:

	count	mean	std	min	median	max	missing%	skew	kurtosis
Age	986.0	41.745	13.963	18.0	42.0	66.0	0.0	0.030	-1.132
Height	986.0	168.183	10.098	145.0	168.0	188.0	0.0	-0.180	-0.762
Weight	986.0	76.950	14.265	51.0	75.0	132.0	0.0	0.667	0.610
Number_of_Major_Surgeries	986.0	0.667	0.749	0.0	1.0	3.0	0.0	0.861	0.066
Premium_Price	986.0	24336.714	6248.184	15000.0	23000.0	40000.0	0.0	0.098	-0.453

Binary Variables Summary:

	Yes	No
Diabetes	414	572
Blood_Pressure_Problems	462	524
Any_Transplants	55	931
Any_Chronic_Diseases	178	808
Known_Allergies	212	774
History_of_Cancer_in_Family	116	870

In [15]:
```
df.describe(include='all')
```

Out[15]:

	Age	Diabetes	Blood_Pressure_Problems	Any_Transplants	Any_Chronic_Diseases	Height	Weight	Known_Allergies	History_of_Car
count	986.000000	986.000000	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	76.950304	0.215010	
std	13.963371	0.493789	0.499264	0.229615	0.384821	10.098155	14.265096	0.411038	
min	18.000000	0.000000	0.000000	0.000000	0.000000	145.000000	51.000000	0.000000	
25%	30.000000	0.000000	0.000000	0.000000	0.000000	161.000000	67.000000	0.000000	
50%	42.000000	0.000000	0.000000	0.000000	0.000000	168.000000	75.000000	0.000000	
75%	53.000000	1.000000	1.000000	0.000000	0.000000	176.000000	87.000000	0.000000	
max	66.000000	1.000000	1.000000	1.000000	1.000000	188.000000	132.000000	1.000000	

2.2 - Distribution Analysis

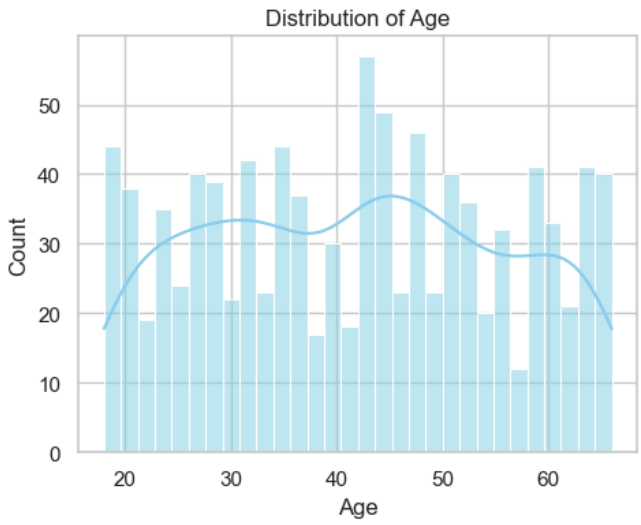
Here we are Going to Perform -

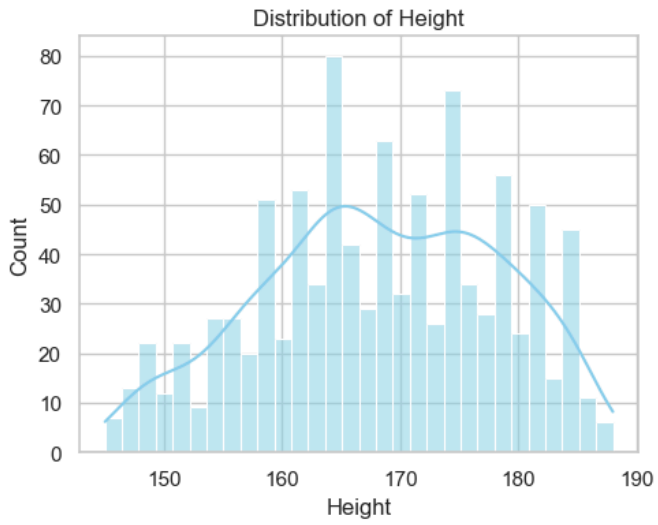
- Histograms, KDE plots for continuous vars
- Bar plots for binary vars

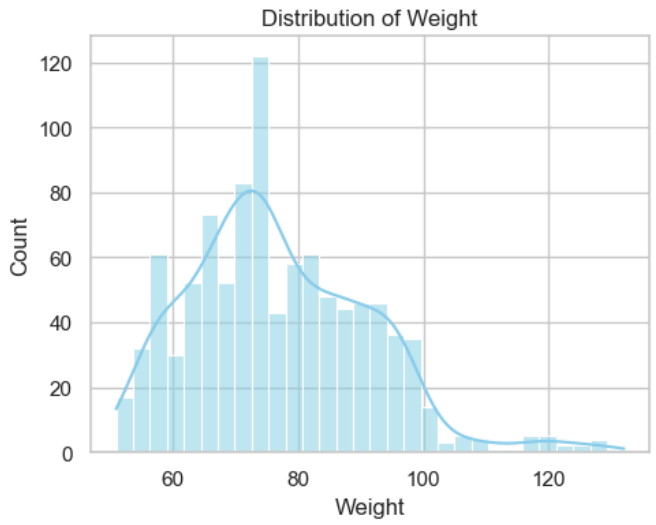
2.2.1 - Continuous Variables

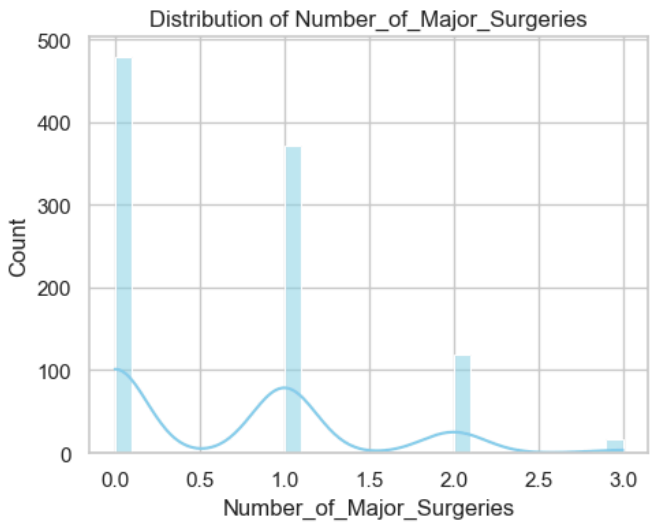
In [16]:
```
# Histograms for Continuous Variables

def plot_continuous_distributions(data, cols):
    n = len(cols)
    rows = int(np.ceil(n/3))
    plt.figure(figsize=(15, 4*rows))
```

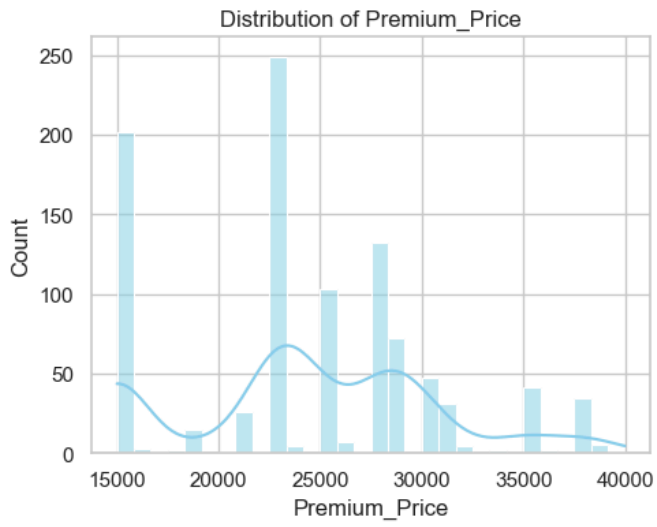
```
    for i, col in enumerate(cols, 1):
        plt.subplot(rows, 3, i)
        sns.histplot(data[col], kde=True, bins=30, color="skyblue")
        plt.title(f"Distribution of {col}")

    plt.tight_layout()
    plt.show()

plot_continuous_distributions(df, cont_cols)
```



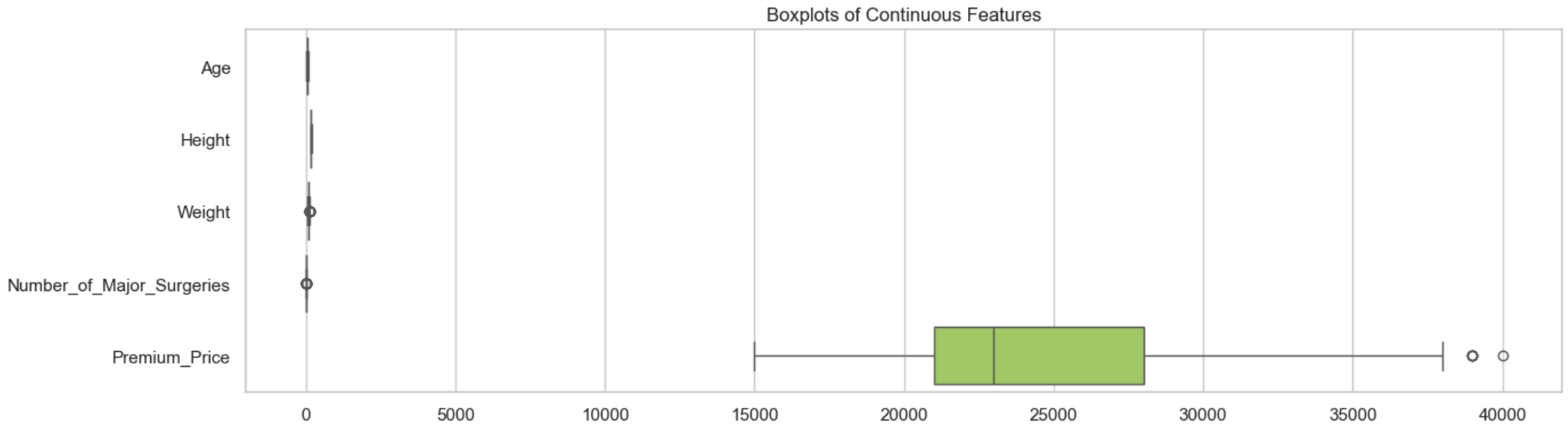






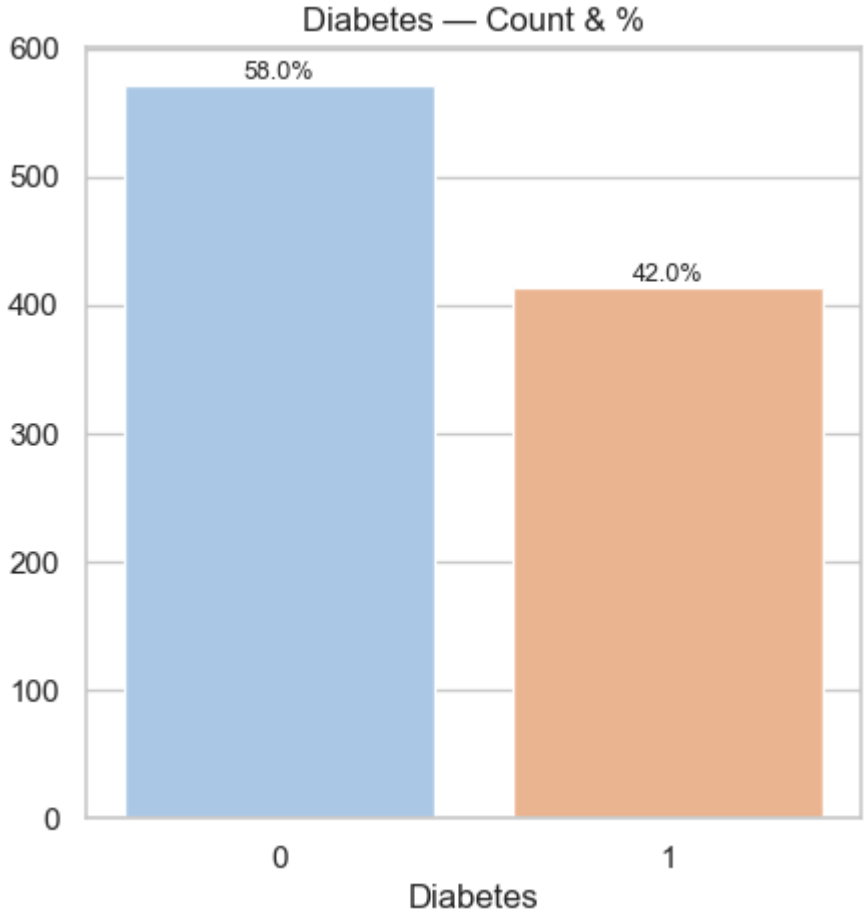


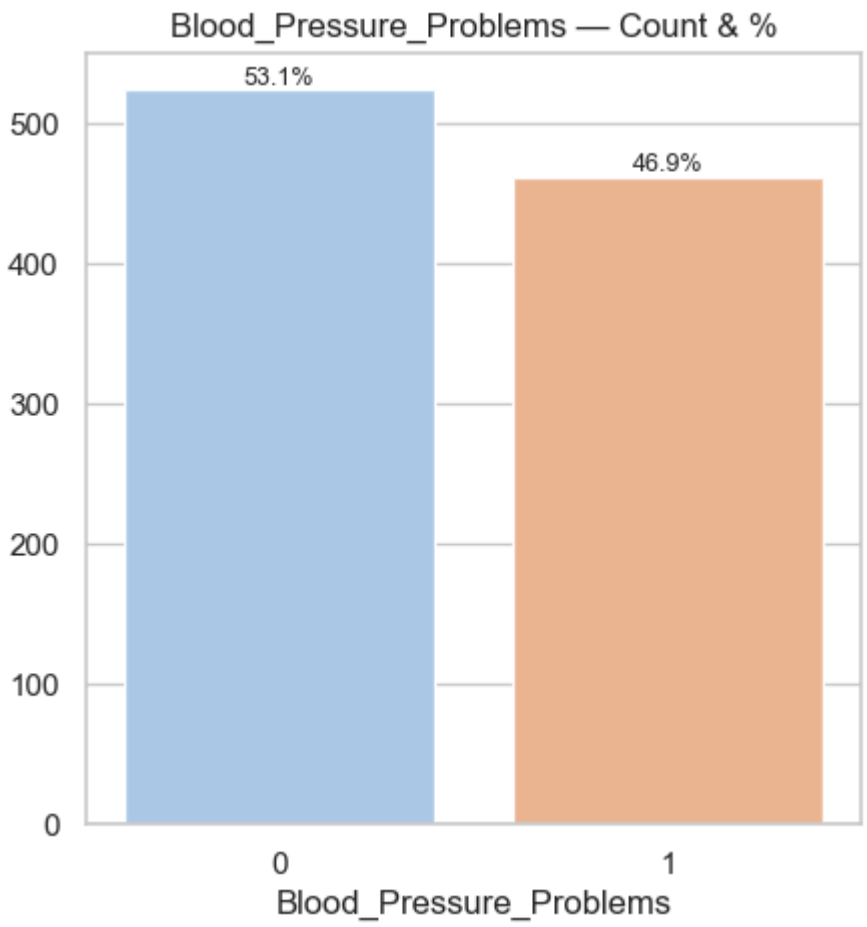
In [17]:
```
# Boxplot for Continuous Variables

def plot_boxplots(data, cols):
    plt.figure(figsize=(14, max(4, 0.5*len(cols))))
    sns.boxplot(data=data[cols], orient="h", palette="Set2")
    plt.title("Boxplots of Continuous Features")
    plt.tight_layout()
    plt.show()

plot_boxplots(df, cont_cols)
```

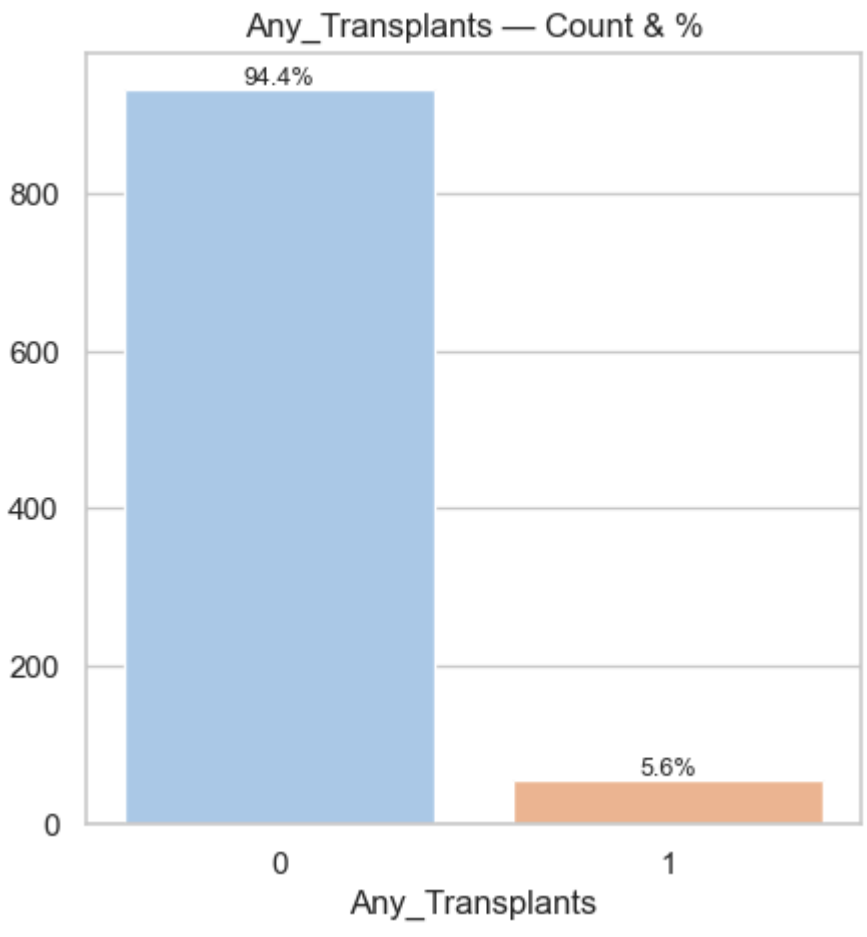


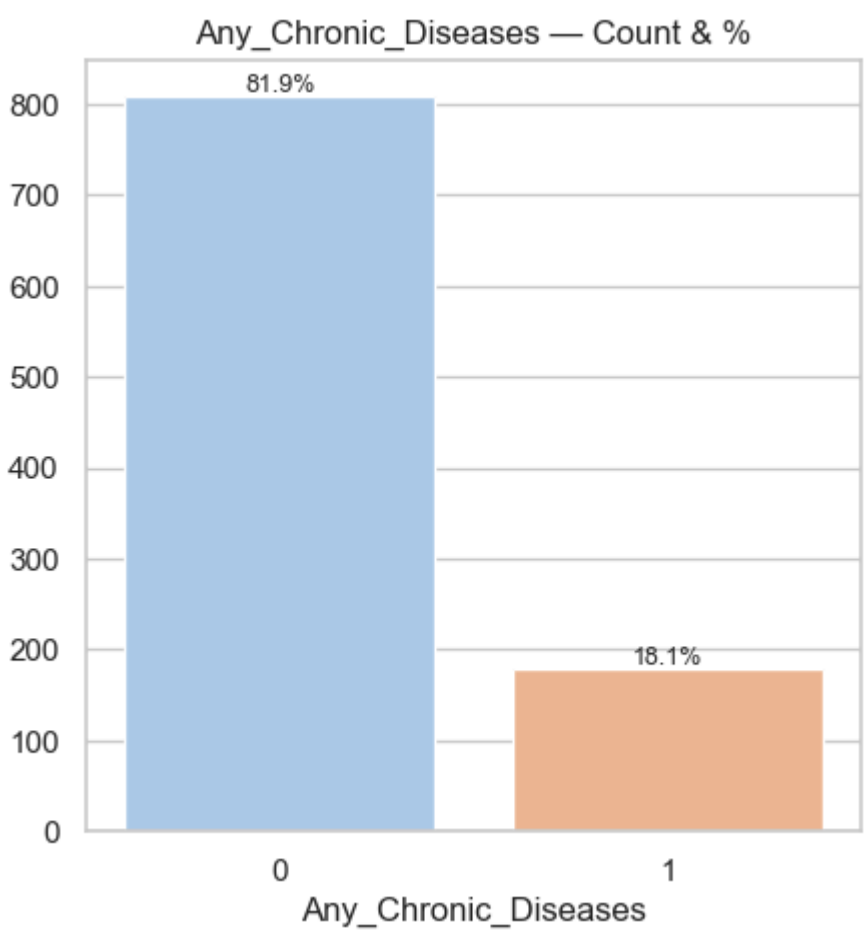
2.2.2 - Binary Variables

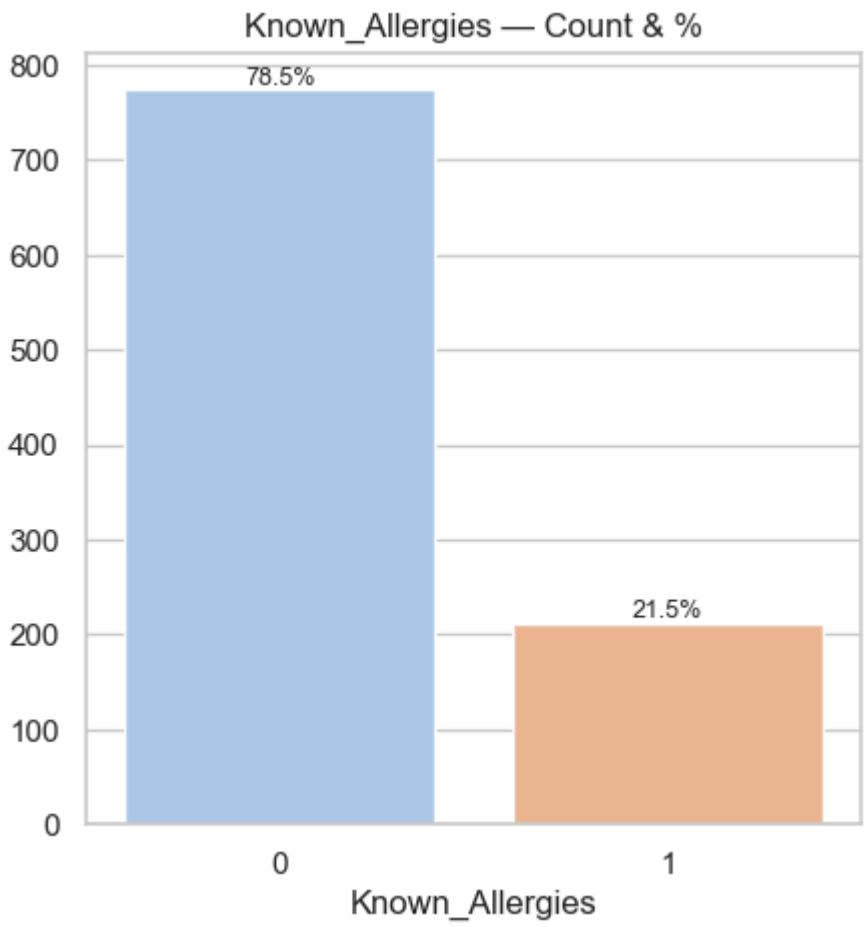
In [18]:
```
# Binary Variable Bar Plots

def plot_binary_distribution(data, col):
    counts = data[col].value_counts().sort_index()
    pct = 100 * counts / counts.sum()

    fig, ax = plt.subplots(figsize=(5,5))
    sns.barplot(x=counts.index.astype(str), y=counts.values, palette="pastel", ax=ax)
    ax.set_title(f"{col} — Count & %")
    for i, v in enumerate(counts.values):
        ax.text(i, v, f"{pct.iloc[i]:.1f}%", ha='center', va='bottom', fontsize=9)
    plt.show()

for col in binary_cols:
    plot_binary_distribution(df, col)
```

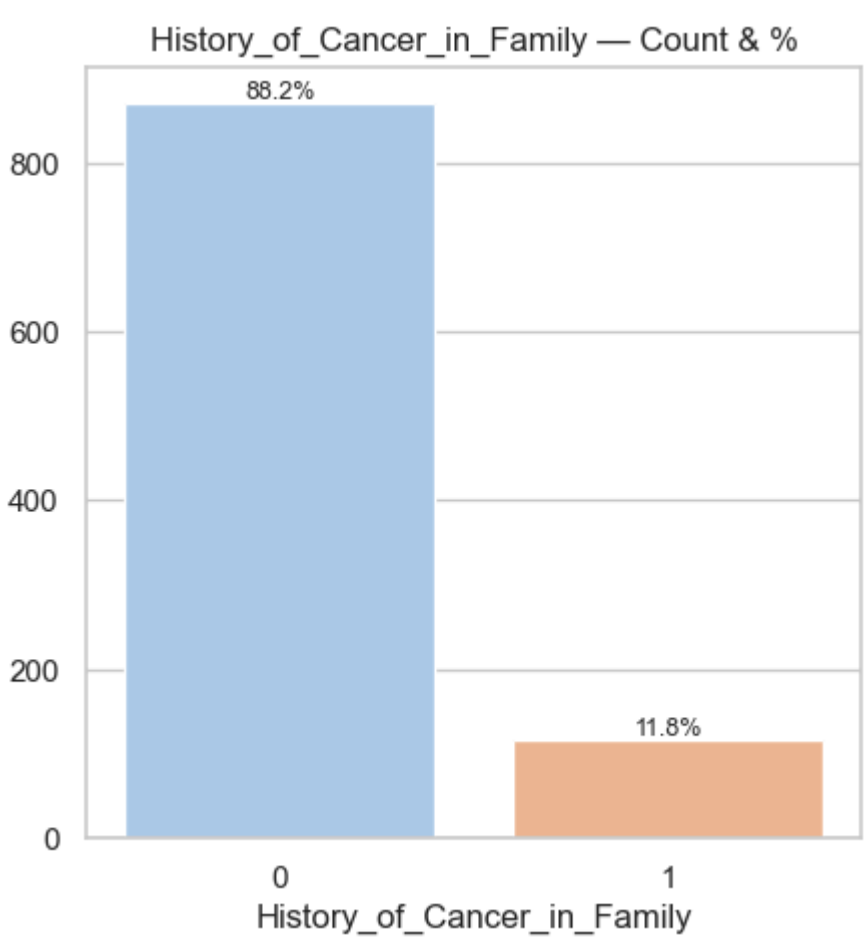

Diabetes — Count & %
600
500
400
300
200
100
0
58.0%
42.0%
0
1
Diabetes


Blood_Pressure_Problems — Count & %
500
400
300
200
100
0
53.1%
46.9%
0
1
Blood_Pressure_Problems


Any_Transplants — Count & %
800
600
400
200
0
94.4%
5.6%
0
1
Any_Transplants



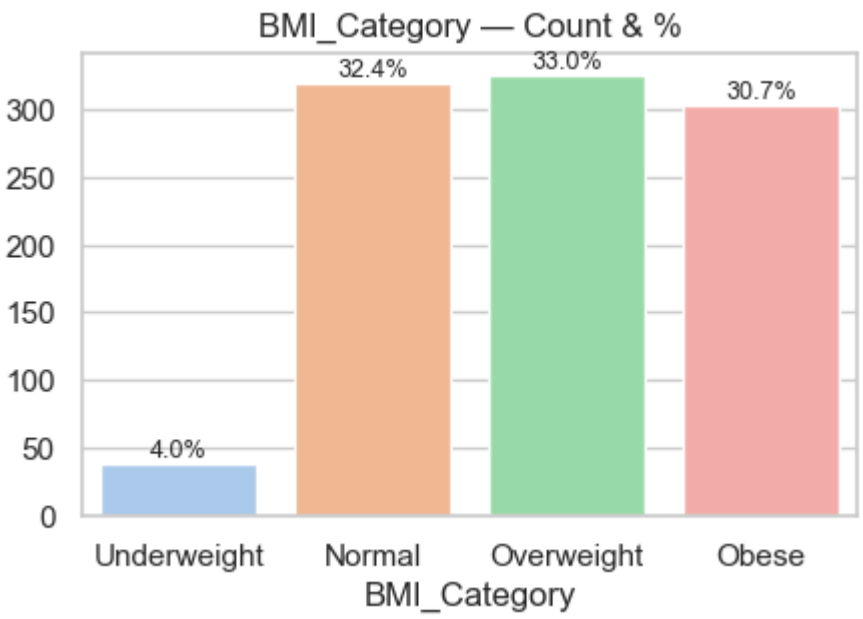


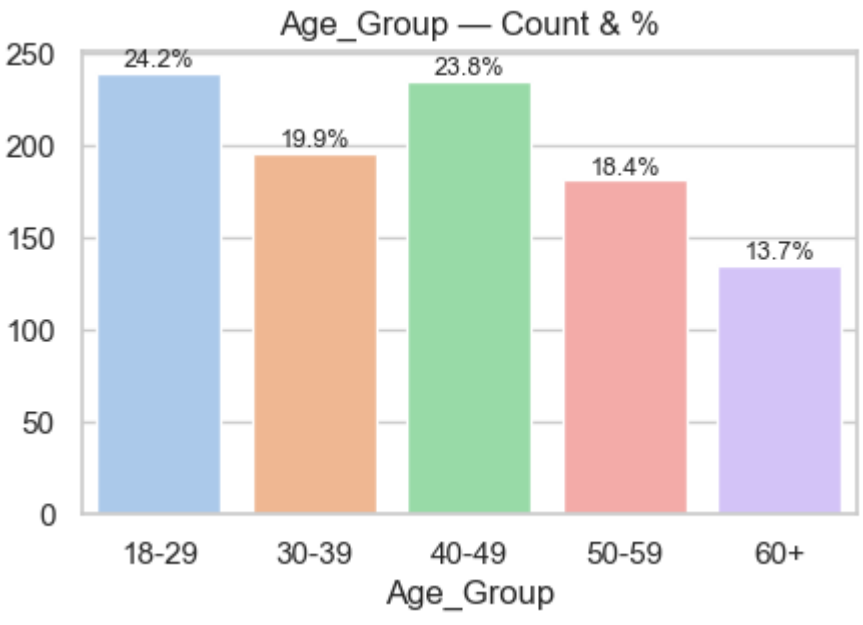


2.2.3 - Engineered features (BMI, Age Groups)

In [19]:
```
# Engineered Values

# Helper function for barplot with % labels
def plot_category_distribution(data, col, palette="pastel"):
    counts = data[col].value_counts().sort_index()
```

```
    pct = 100 * counts / counts.sum()

    fig, ax = plt.subplots(figsize=(5,3))
    sns.barplot(x=counts.index.astype(str), y=counts.values, palette=palette, ax=ax)
    ax.set_title(f"{col} — Count & %")

    for i, v in enumerate(counts.values):
        ax.text(i, v, f"{pct.iloc[i]:.1f}%", ha='center', va='bottom', fontsize=9)
    plt.show()

# ✅ BMI
df['BMI'] = df['Weight'] / ((df['Height']/100)**2)
bmi_bins = [0, 18.5, 25, 30, np.inf]
bmi_labels = ['Underweight','Normal','Overweight','Obese']
df['BMI_Category'] = pd.cut(df['BMI'], bins=bmi_bins, labels=bmi_labels, right=False)

plot_category_distribution(df, 'BMI_Category')

# ✅ Age groups
age_bins = [18, 30, 40, 50, 60, np.inf]
age_labels = ['18-29','30-39','40-49','50-59','60+']
df['Age_Group'] = pd.cut(df['Age'], bins=age_bins, labels=age_labels, right=False)

plot_category_distribution(df, 'Age_Group')
```





In [20]:
```
# Add BMI to Continuous Columns
cont_cols.append('BMI')
```

In [21]:
```
cont_cols
```

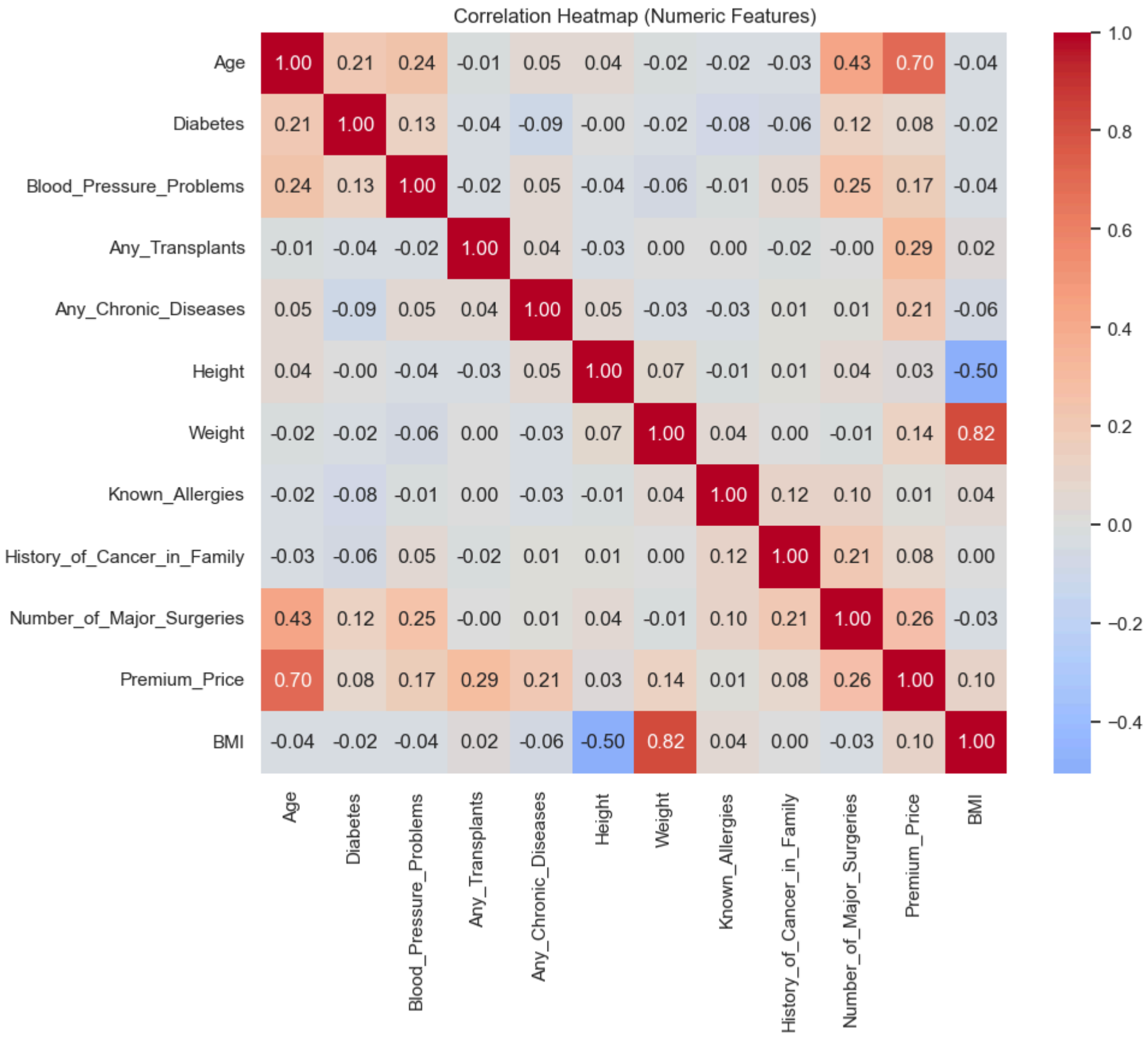
Out[21]:
```
['Age',
 'Height',
 'Weight',
 'Number_of_Major_Surgeries',
 'Premium_Price',
 'BMI']
```

2.3 - Correlation Analysis

2.3.1 - Correlation Heatmap (Numeric) & Correlation Ranking

In [22]:
```
# Heatmap of Correlation

def plot_corr_heatmap(data, target=TARGET):
    corr = data.corr(numeric_only=True)

    plt.figure(figsize=(10,8))
    sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", center=0)
    plt.title("Correlation Heatmap (Numeric Features)")
    plt.show()

    # Sort features by correlation with target
```

```
    target_corr = corr[target].drop(target).sort_values(ascending=False)
    print("🔎 Correlation of Features with PremiumPrice:")
    display(target_corr)

plot_corr_heatmap(df)
```




```
🔎 Correlation of Features with PremiumPrice:
Age                           0.697540
Any_Transplants               0.289056
Number_of_Major_Surgeries     0.264250
Any_Chronic_Diseases          0.208610
Blood_Pressure_Problems       0.167097
Weight                        0.141507
BMI                           0.103812
History_of_Cancer_in_Family   0.083139
Diabetes                      0.076209
Height                        0.026910
Known_Allergies               0.012103
Name: Premium_Price, dtype: float64
```

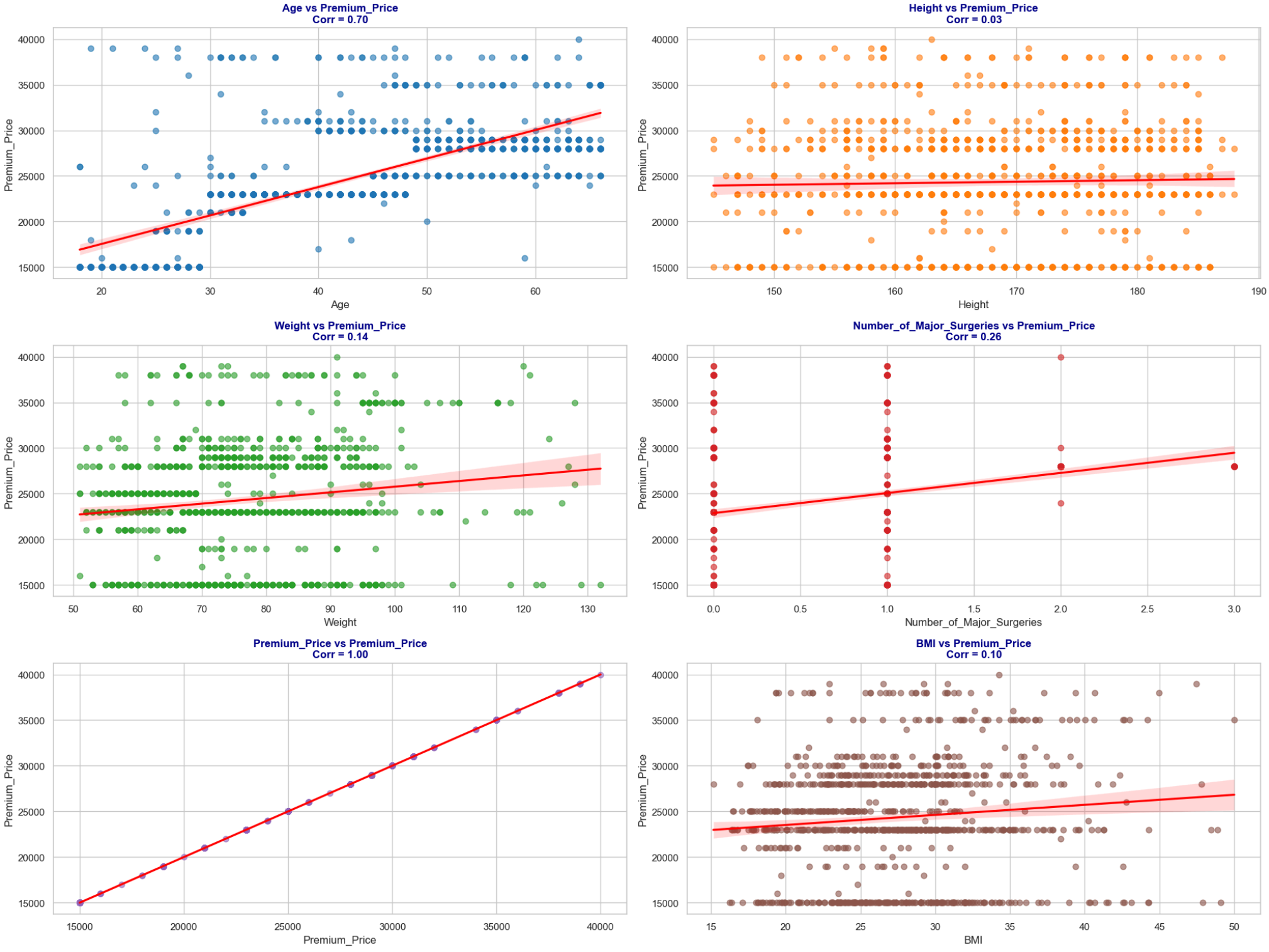
2.3.2 - Scatter Plots for Top Variables

In [23]:
```
cont_cols
```

Out[23]:
```
['Age',
 'Height',
 'Weight',
 'Number_of_Major_Surgeries',
 'Premium_Price',
 'BMI']
```

In [24]:
```
# Premium vs Continuous Features (Scatterplots + Correlation Annotations)

# Custom Colors
Colors_Palette = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b"]

# ---- Continuous Features ----
# cont_cols = ['Age', 'Height', 'Weight', 'Number_of_Major_Surgeries']

# Plot regression scatter plots with CI
plt.figure(figsize=(20, 15))
```

```
for idx, col in enumerate(cont_cols, 1):
    plt.subplot(3, 2, idx)
    sns.regplot(
        x=df[col],
        y=df['Premium_Price'],
        color=Colors_Palette[idx-1],
        scatter_kws={'alpha':0.6, 's':40},  # show all points clearly
        line_kws={'color':'red'}, ci=95
    )

    # --- Correlation Annotation ---
    corr = df[col].corr(df['Premium_Price'])
    plt.title(f"{col} vs Premium_Price\nCorr = {corr:.2f}", fontsize=12, fontweight="bold", color="darkblue")

plt.tight_layout()
plt.show()
```



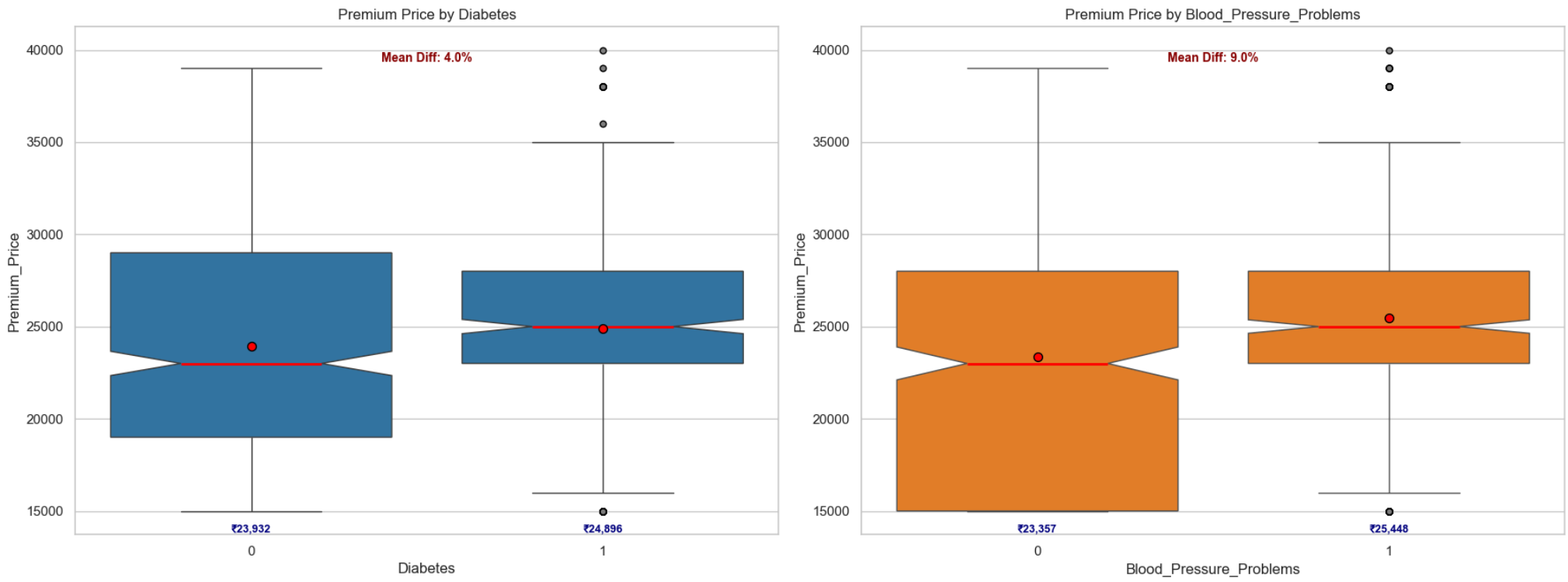
2.3.3 - Boxplot for Top Variables

In [25]:
```
binary_cols
```

Out[25]:
```
['Diabetes',
 'Blood_Pressure_Problems',
 'Any_Transplants',
 'Any_Chronic_Diseases',
 'Known_Allergies',
 'History_of_Cancer_in_Family']
```

In [26]:
```
# Premium vs Binary Features (Boxplots + % Difference)

# Custom Colors
Colors_Palette = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b"]

# Boxplots for Binary Features vs Premium Price
plt.figure(figsize=(18, 20))

for idx, col in enumerate(binary_cols, 1):
    plt.subplot(3, 2, idx)
    sns.boxplot(
        x=df[col],
        y=df['Premium_Price'],
        color=Colors_Palette[idx-1],
        notch=True,
        showmeans=True,
        meanprops={'marker': 'o','markerfacecolor': 'red',
                   'markeredgecolor': 'black','markersize': 7},
```

```
        medianprops={"color": "red", "linewidth": 2},
        flierprops={'marker': 'o','markersize': 5,
                    'markerfacecolor': 'gray','markeredgecolor': 'black'}
    )

    # calculate means for both groups
    means = df.groupby(col)['Premium_Price'].mean()
    if len(means) == 2:
        diff_pct = ((means[1] - means[0]) / means[0]) * 100

        # annotate percentage difference at the top
        plt.text(
            0.5, max(df['Premium_Price']) * 0.98,
            f"Mean Diff: {diff_pct:.1f}%",
            ha='center', va='bottom', fontsize=10, color='darkred', weight='bold'
        )

        # annotate actual mean values under each box
        for x_pos, val in enumerate(means):
            plt.text(
                x_pos, df['Premium_Price'].min() * 0.95,
                f"₹{val:,.0f}",  # formatted with commas
                ha='center', va='top', fontsize=9, color='navy', weight='bold'
            )

    plt.title(f"Premium Price by {col}", fontsize=12)

plt.tight_layout()
plt.show()
```



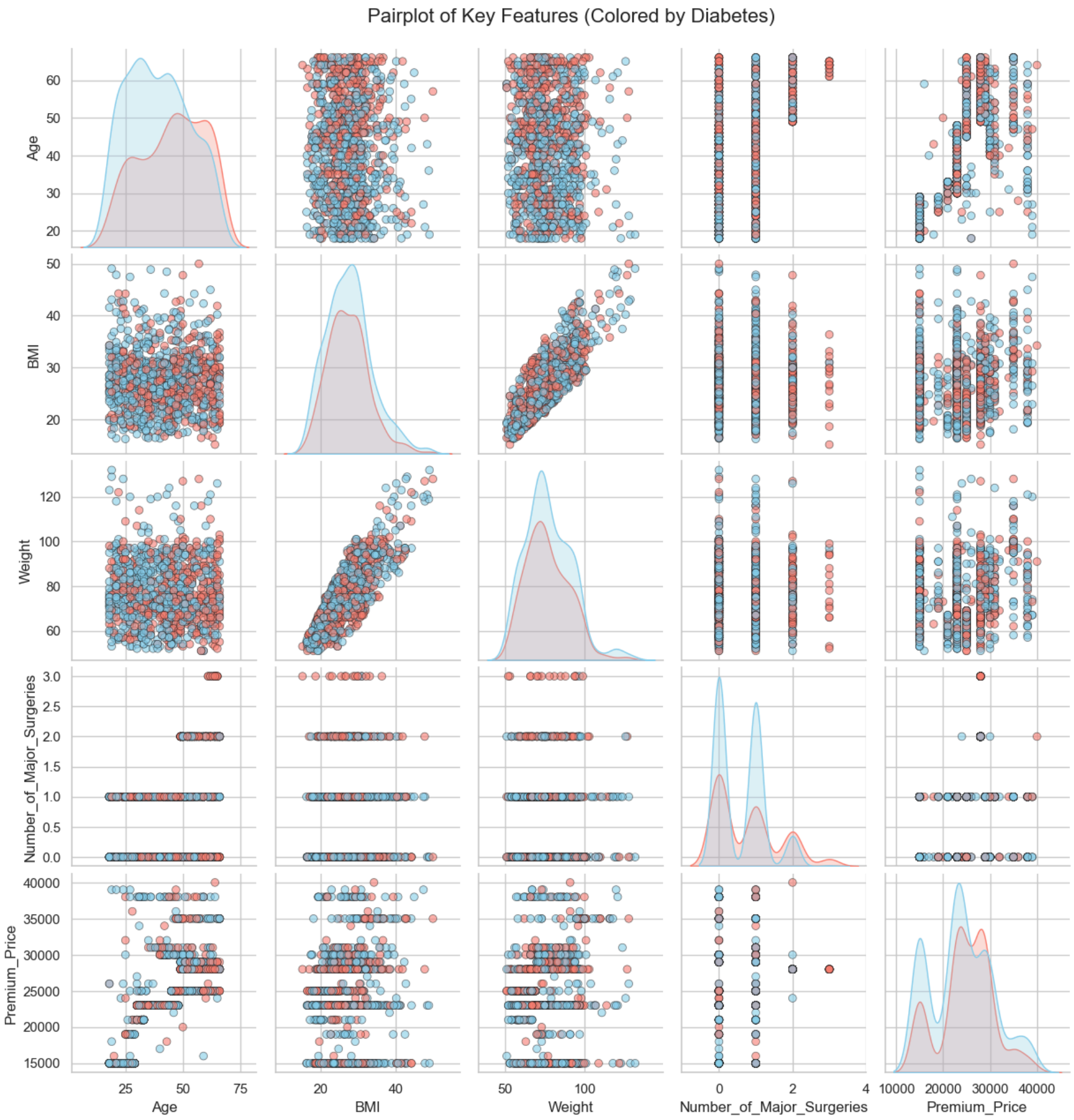
Now, a “Mean Diff: X%” annotation appears above, showing the % difference in average premium between those with (1) vs without (0) the condition.

2.3.4 - Pairplots for Top Variables

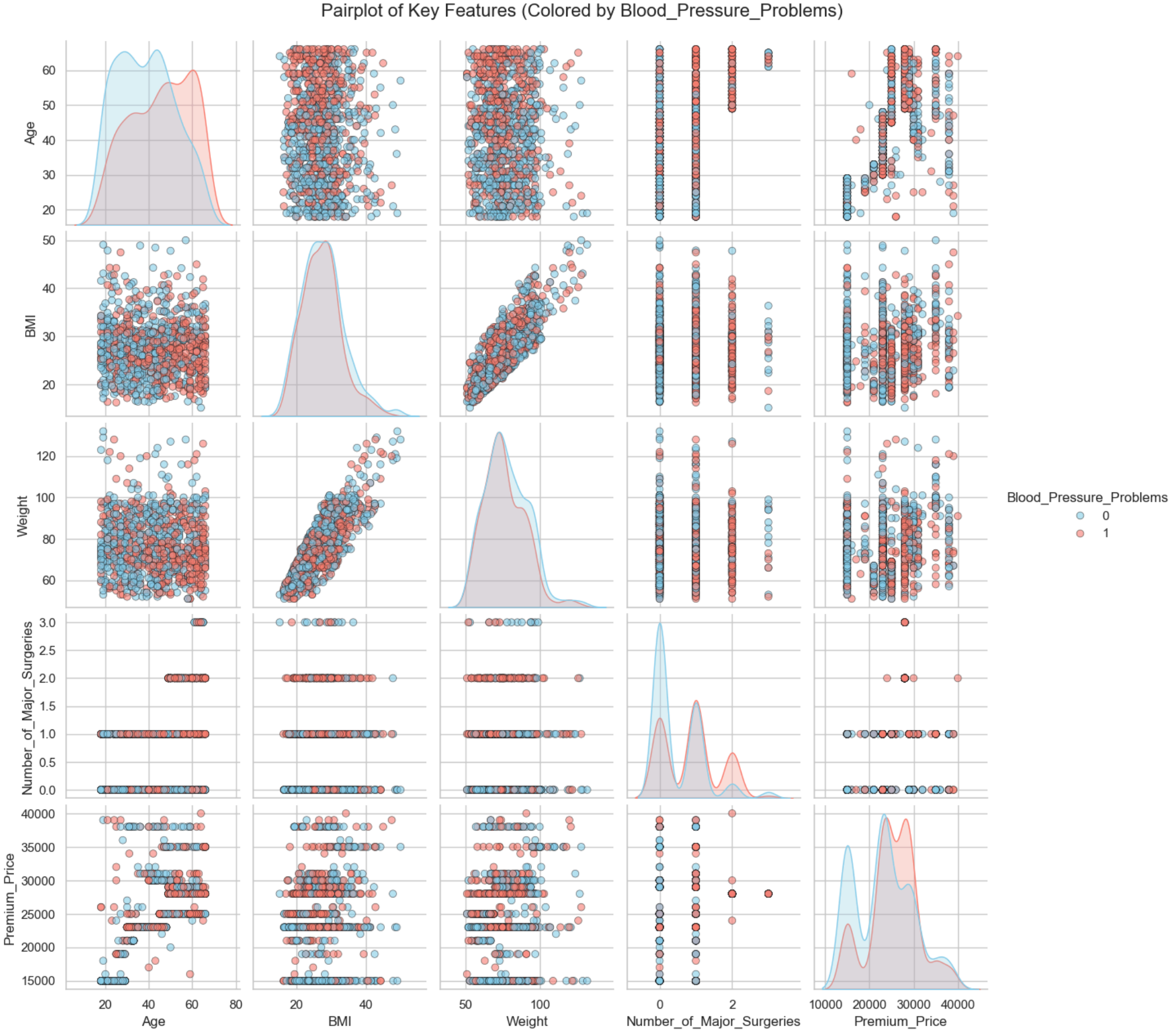
In [27]:
```python
# Pairplots for Key Features (with all binary columns)

# numeric features for pairplot
pairplot_cols = ['Age', 'BMI', 'Weight', 'Number_of_Major_Surgeries', 'Premium_Price']

# binary columns (all except Premium_Price, Age, Height, Weight, BMI, Surgeries)
binary_cols = ['Diabetes', 'Blood_Pressure_Problems', 'Any_Transplants',
               'Any_Chronic_Diseases', 'Known_Allergies', 'History_of_Cancer_in_Family']

def save_pairplots(df, num_cols, bin_cols, save_path="pairplots"):
    import os
    os.makedirs(save_path, exist_ok=True)  # create folder if not exists

    for col in bin_cols:
        print(f"Generating Pairplot for: {col}...")

        g = sns.pairplot(
            df[num_cols + [col]],
            hue=col,
            diag_kind="kde",
```

```
            palette={0: "skyblue", 1: "salmon"},
            plot_kws={'alpha':0.6, 's':40, 'edgecolor':'k'}
        )

        g.fig.suptitle(f"Pairplot of Key Features (Colored by {col})", y=1.02, fontsize=16)

        # save plot
        filename = f"{save_path}/pairplot_{col}.png"
        g.savefig(filename, dpi=150, bbox_inches='tight')
        plt.show()  # also display inline
        plt.close()

# Run the function
save_pairplots(df, pairplot_cols, binary_cols)
```

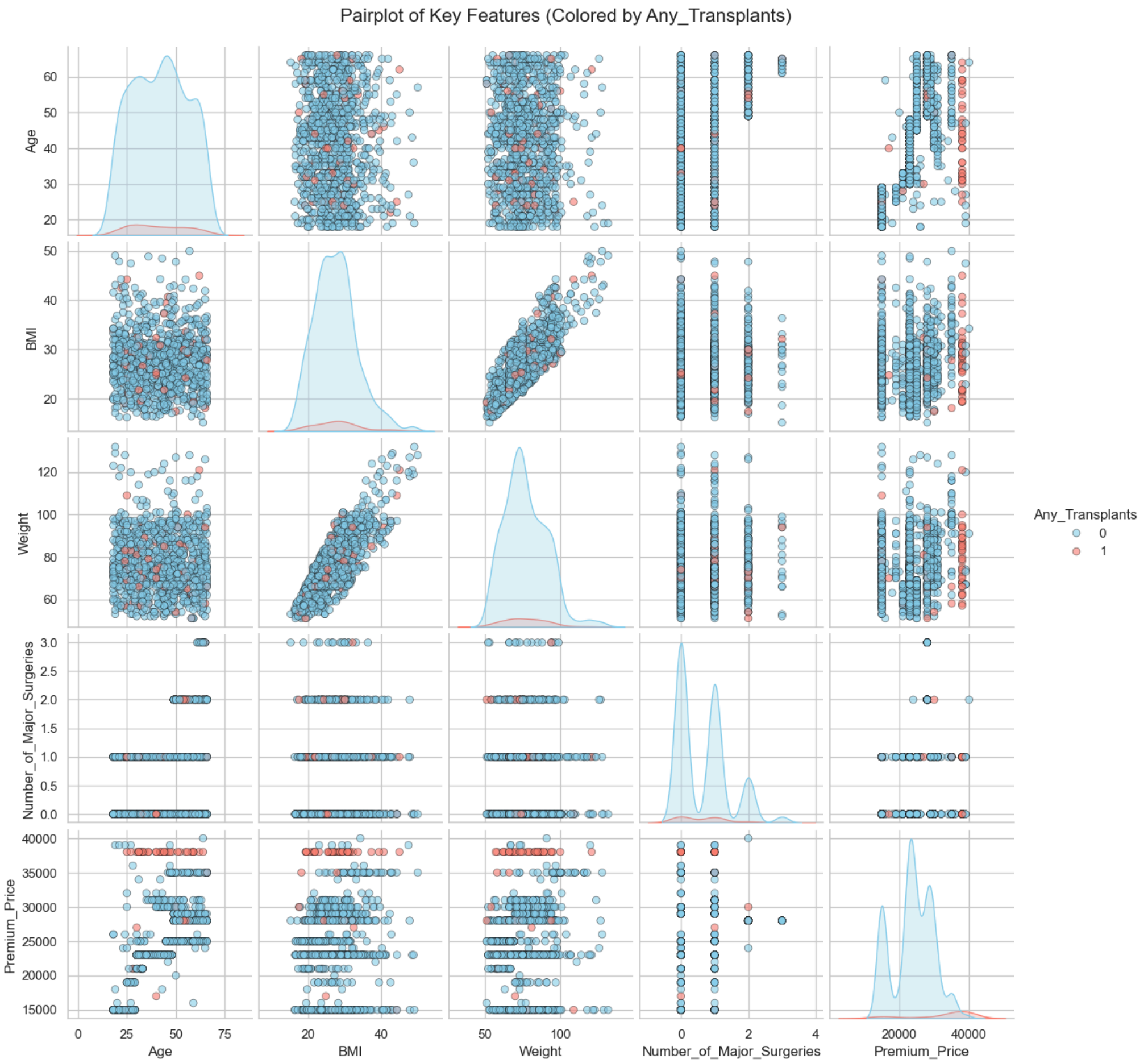
```
Generating Pairplot for: Diabetes...
```



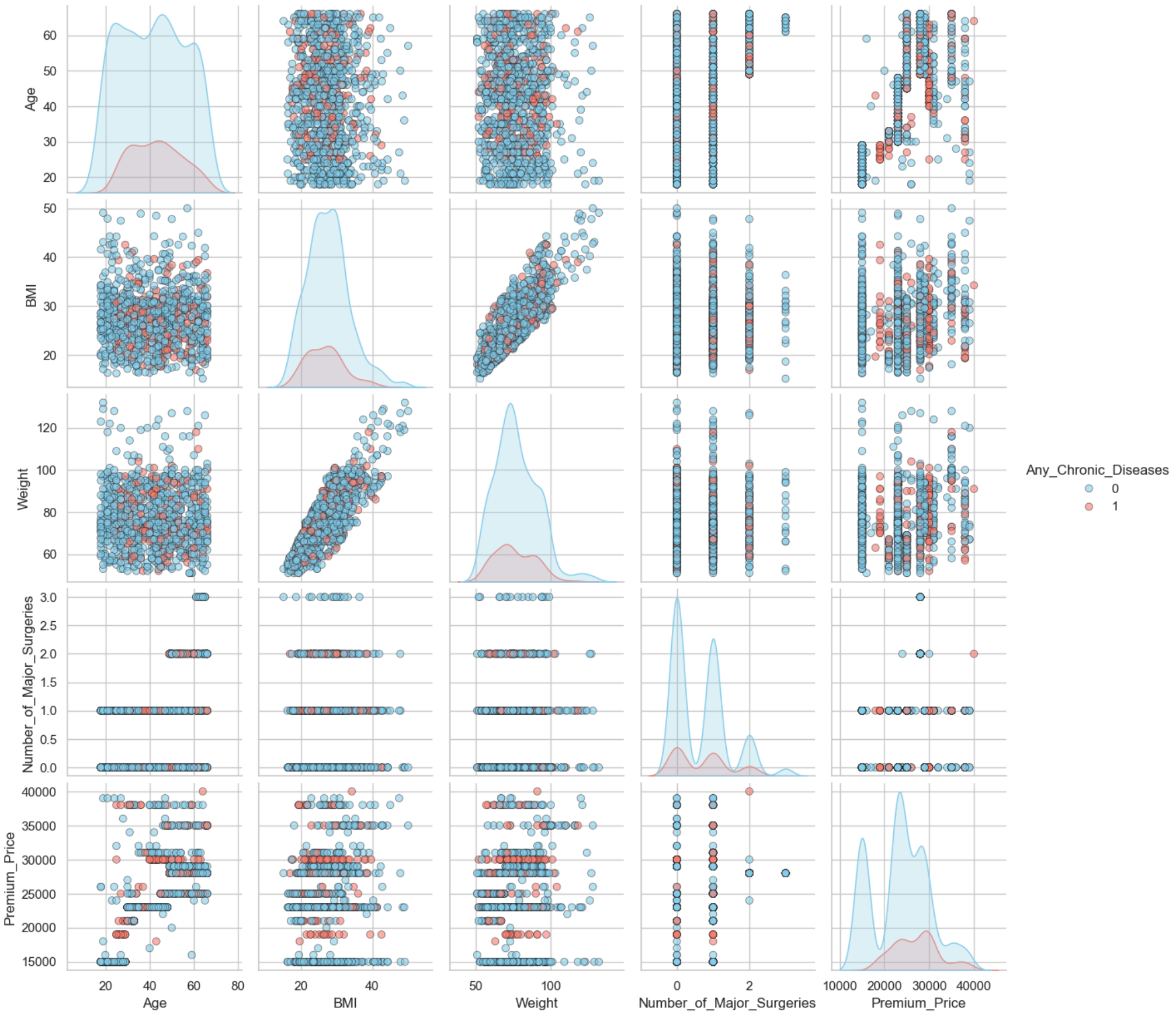

```
Generating Pairplot for: Blood_Pressure_Problems...
```



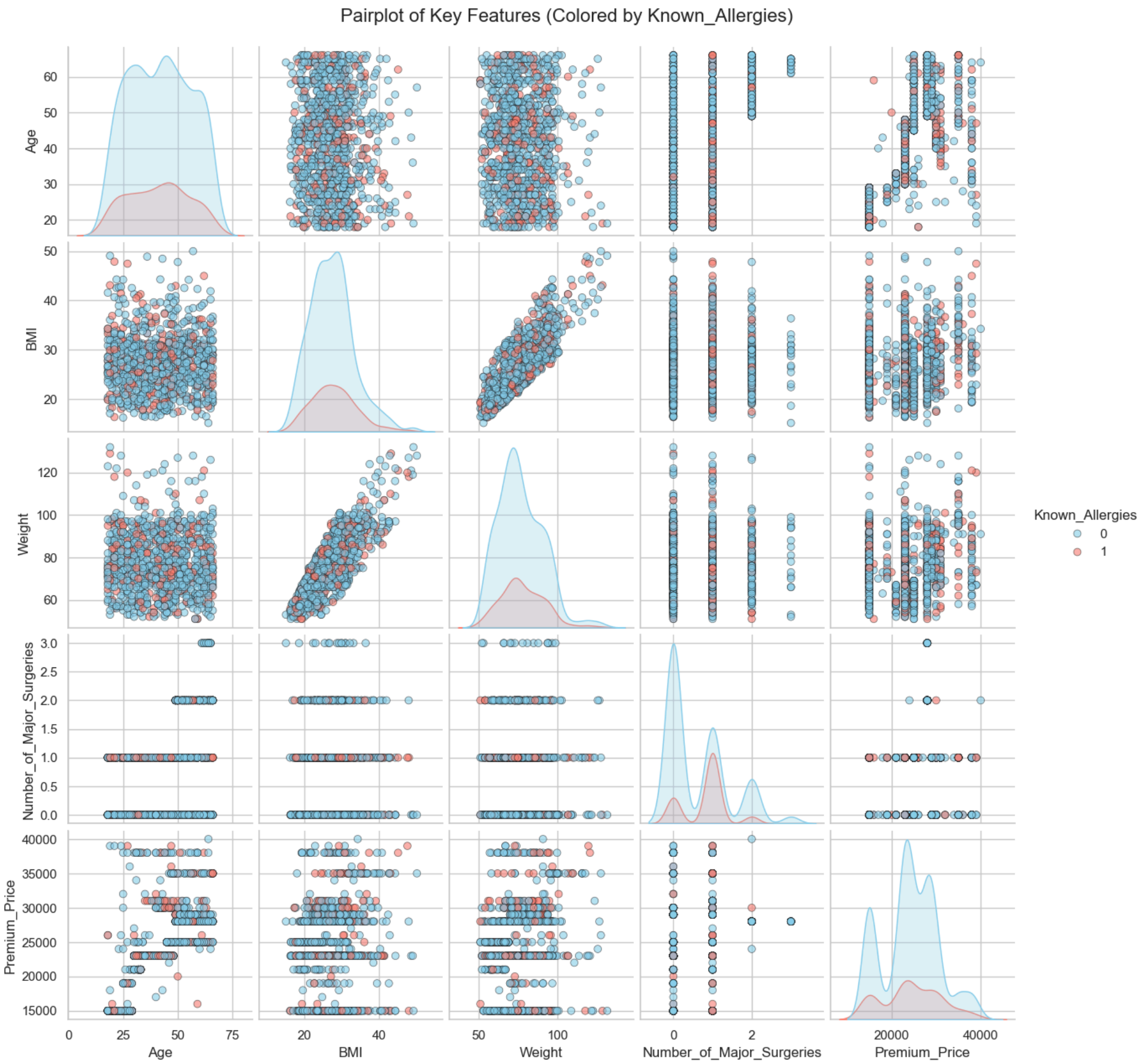

```
Generating Pairplot for: Any_Transplants...
```



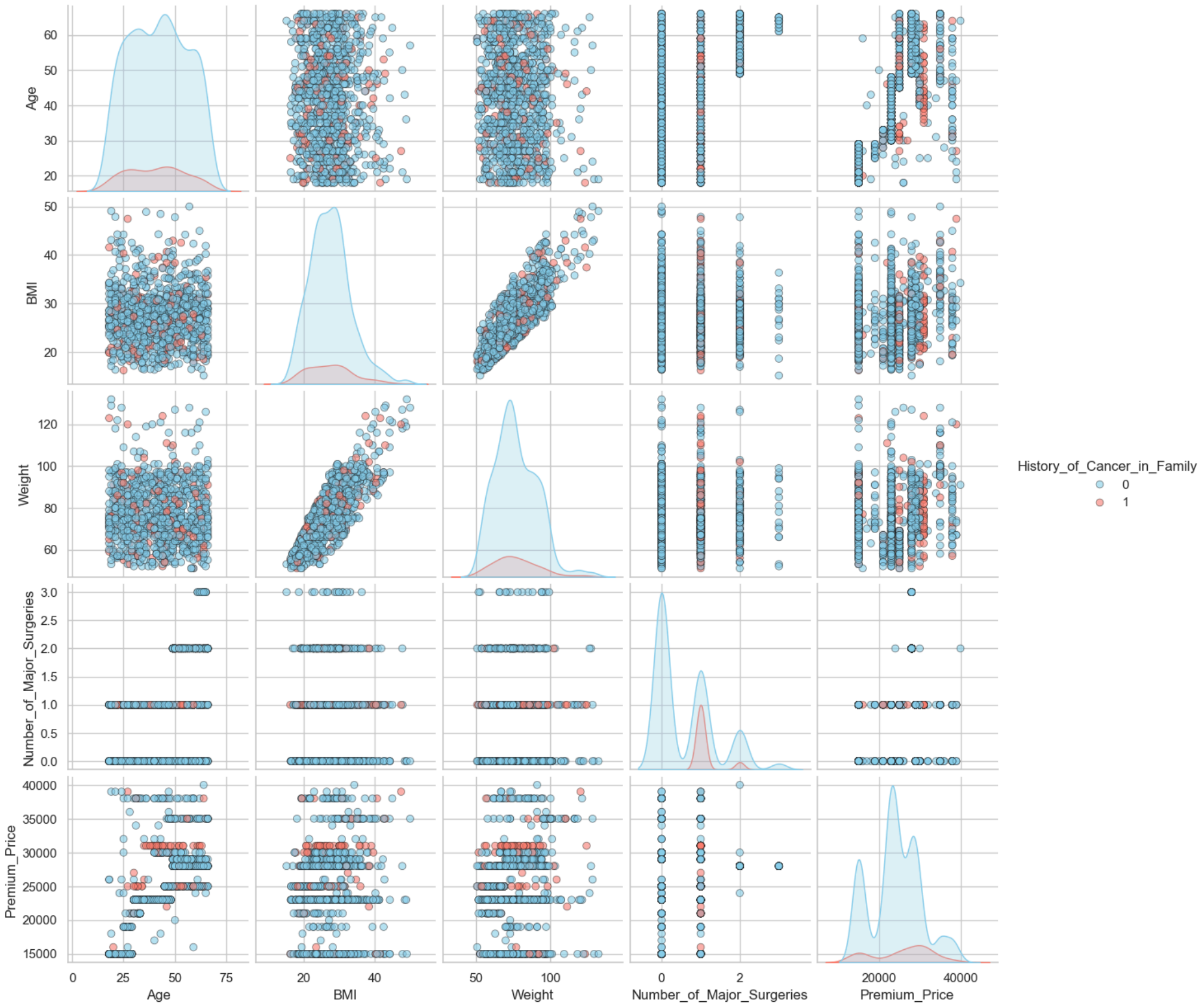
Generating Pairplot for: Any_Chronic_Diseases...



Generating Pairplot for: Known_Allergies...



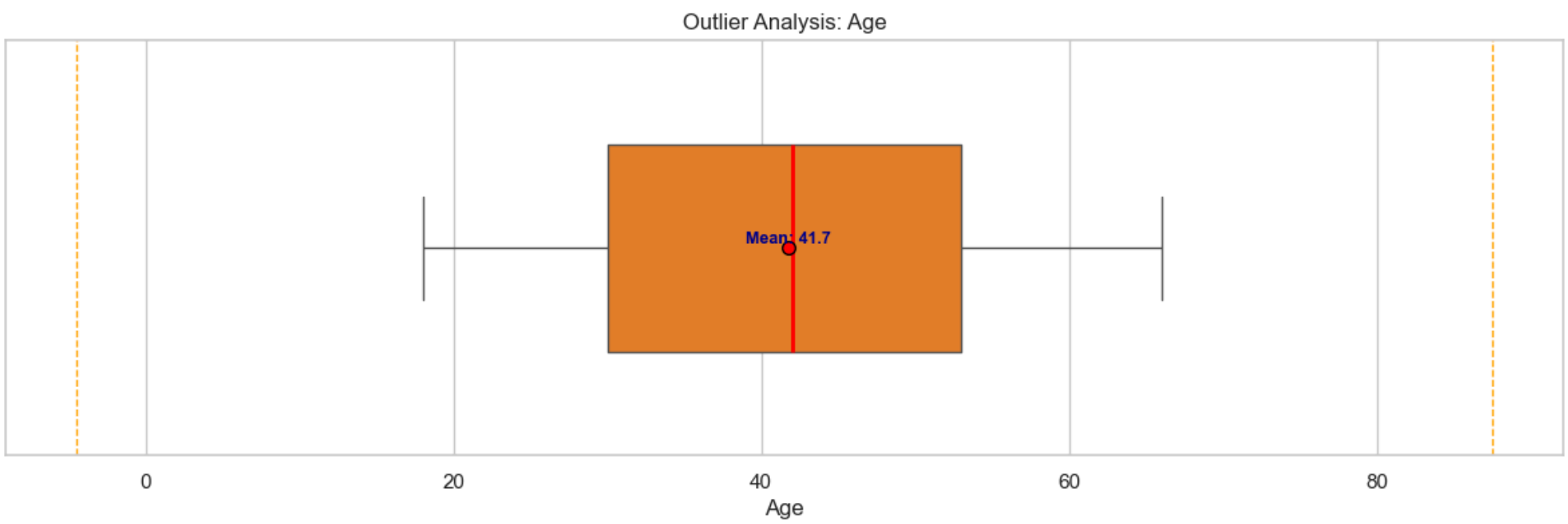
Generating Pairplot for: History_of_Cancer_in_Family...

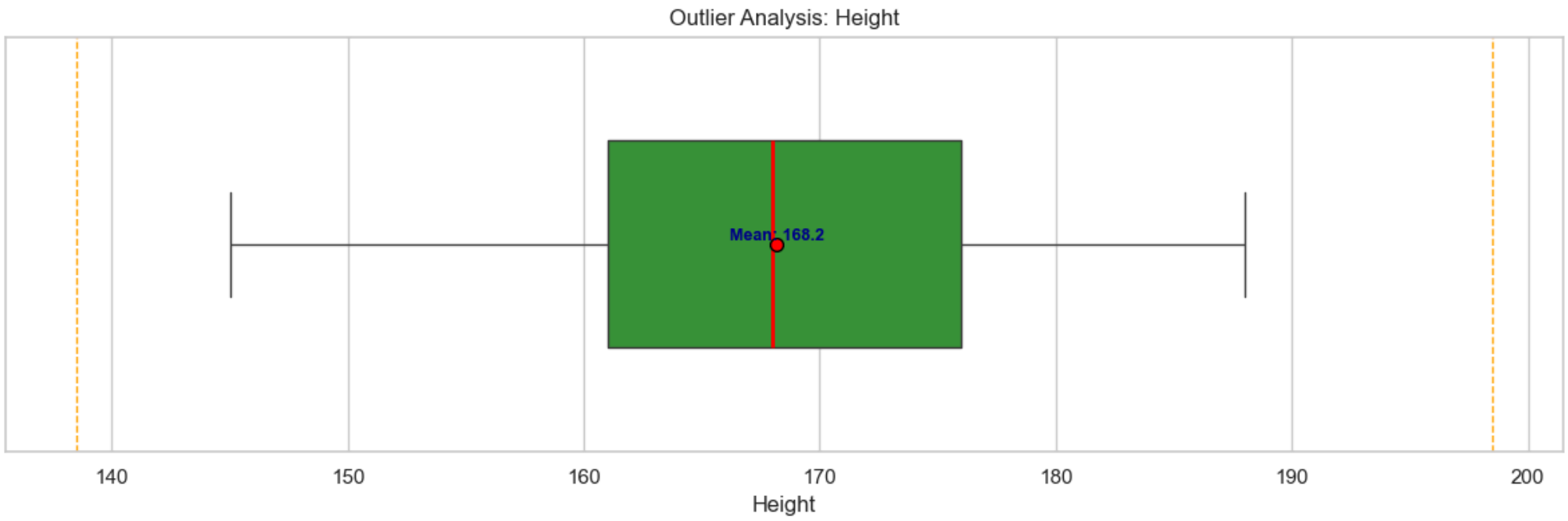


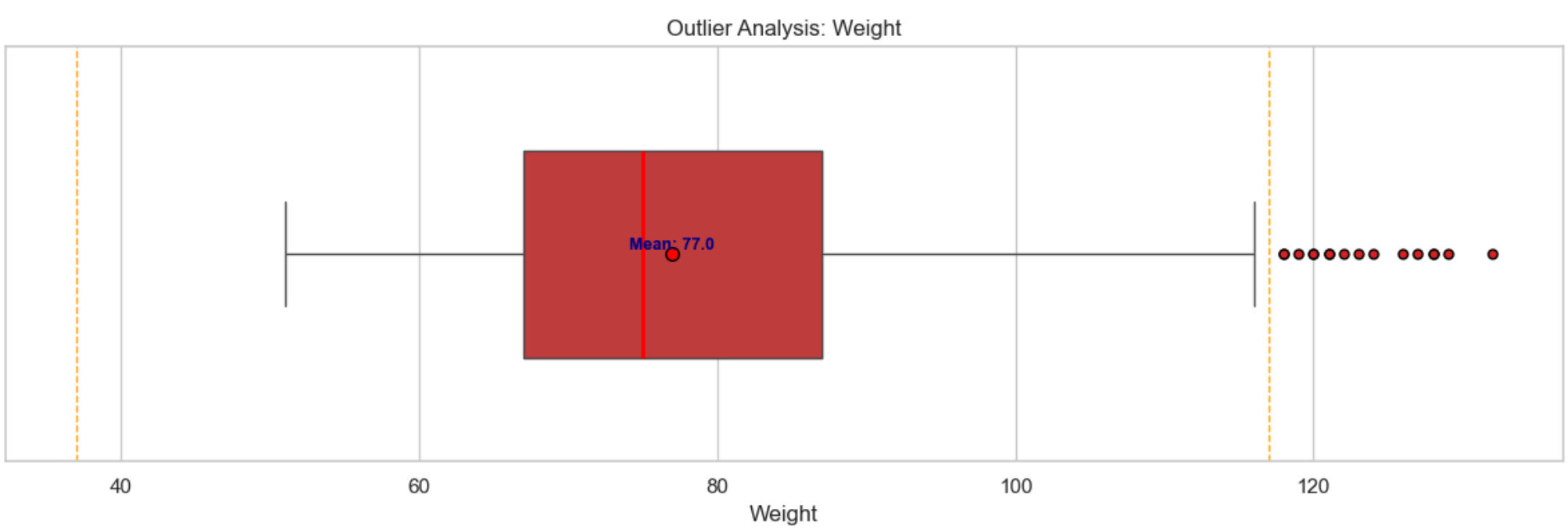
2.4 - Outlier Analysis

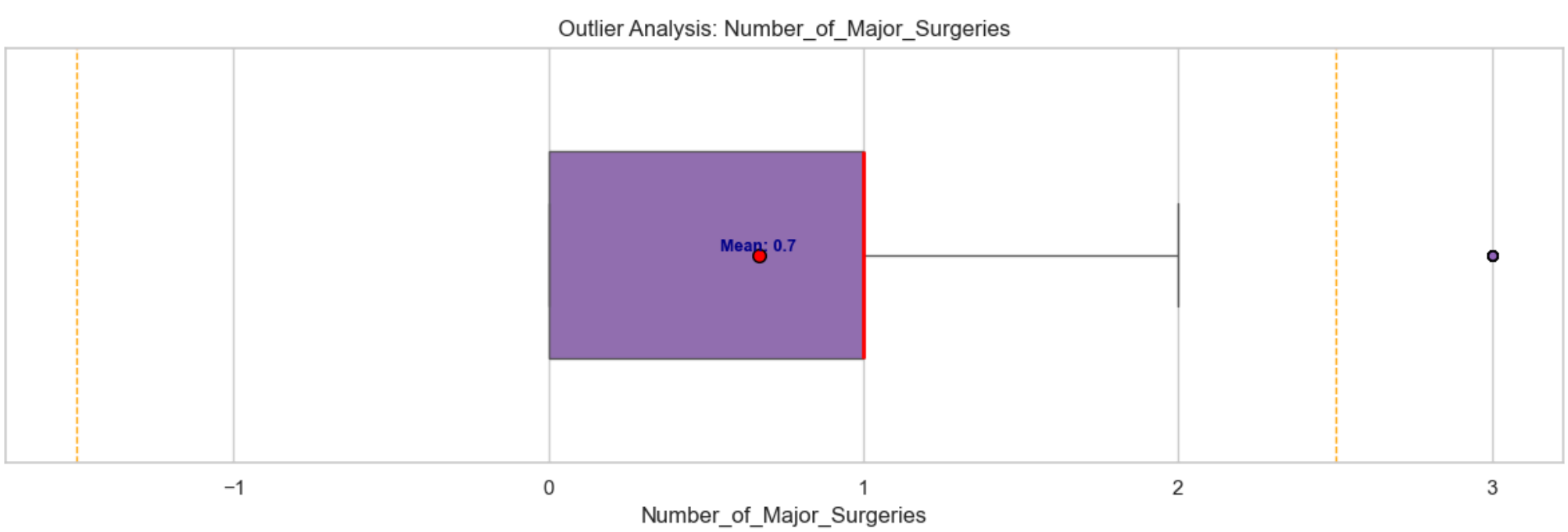
In [28]:
```
# Outlier Analysis

# Continuous variables
cont_cols = ['Age', 'Height', 'Weight', 'Number_of_Major_Surgeries', 'Premium_Price']

# Custom Colors
Colors_Palette = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b"]

# Dictionary to store outlier counts
outlier_summary = {}

plt.figure(figsize=(12, 4*len(cont_cols)))  # one row per boxplot

for i, col in enumerate(cont_cols, 1):
    plt.subplot(len(cont_cols), 1, i)
    plt.title(f"Outlier Analysis: {col}", fontsize=12)

    # Boxplot (horizontal)
    sns.boxplot(
        data=df,
        x=col,
        color=Colors_Palette[i % len(Colors_Palette)],
        width=0.5,
        fliersize=5,
        #notch=True,
        showcaps=True,
        linewidth=1,
        showmeans=True,
        meanprops={'marker': 'o',
                   'markerfacecolor': 'red',
                   'markeredgecolor': 'black',
                   'markersize': 7},
        medianprops={"color": "red", "linewidth": 2},
        flierprops={'marker': 'o',
                    'markersize': 5,
                    'markerfacecolor': Colors_Palette[i % len(Colors_Palette)],
                    'markeredgecolor': 'black',
```

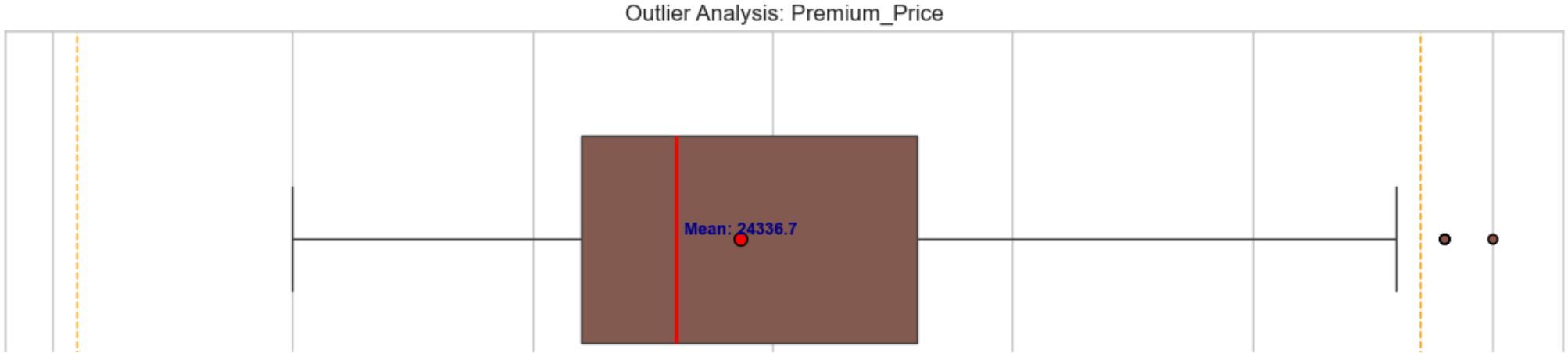
```
                  "linewidth": 1.5}
    )

    # --- Add Mean Value Annotation ---
    mean_val = df[col].mean()
    plt.text(mean_val, 0, f"Mean: {mean_val:.1f}", ha='center', va='bottom',
             fontsize=9, fontweight="bold", color="darkblue")

    # --- Outlier Detection ---
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound, upper_bound = Q1 - 1.5*IQR, Q3 + 1.5*IQR

    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)][col]
    outlier_summary[col] = {
        "Lower Bound": lower_bound,
        "Upper Bound": upper_bound,
        "Outlier Count": outliers.shape[0]
    }

    # Add IQR lines
    plt.axvline(lower_bound, color="orange", linestyle="--", lw=1)
    plt.axvline(upper_bound, color="orange", linestyle="--", lw=1)

plt.tight_layout()
plt.show()

# Convert outlier summary into a nice DataFrame
outlier_df = pd.DataFrame(outlier_summary).T
display(outlier_df)
```


Outlier Analysis: Age
Mean: 41.7
0
20
40
60
80
Age


Outlier Analysis: Height
Mean: 168.2
140
150
160
170
180
190
200
Height


Outlier Analysis: Weight
Mean: 77.0
40
60
80
100
120
Weight


Outlier Analysis: Number_of_Major_Surgeries
Mean: 0.7
−1
0
1
2
3
Number_of_Major_Surgeries


Outlier Analysis: Premium_Price
Mean: 24336.7

	Lower Bound	Upper Bound	Outlier Count
Age	-4.5	87.5	0.0
Height	138.5	198.5	0.0
Weight	37.0	117.0	16.0
Number_of_Major_Surgeries	-1.5	2.5	16.0
Premium_Price	10500.0	38500.0	6.0

2.5 - Z-Score & IQR

In [146]:
```python
# --- Z-Score Method ---
def z_score_outliers(df, vars, threshold=3, min_percent=5):
    results = {}
    for var in vars:
        if var in df.columns:
            mean = df[var].mean()
            std_dev = df[var].std()
            z_scores = np.abs(stats.zscore(df[var]))
            outliers = len(df[z_scores > threshold])
            total = len(df[var])
            percent_outliers = (outliers / total) * 100

            value_at_z_plus = mean + threshold * std_dev
            value_at_z_minus = mean - threshold * std_dev

            verdict = "Remove" if percent_outliers >= min_percent else "Keep"

            results[var] = {
                "Method": "Z-Score",
                "Lower Threshold": round(value_at_z_minus, 2),
                "Upper Threshold": round(value_at_z_plus, 2),
                "Outliers": outliers,
                "Percent Outliers": round(percent_outliers, 2),
                "Verdict": verdict
            }
    return pd.DataFrame(results).T


# --- IQR Method ---
def iqr_outliers(df, vars, min_percent=5):
    results = {}
    for var in vars:
        if var in df.columns:
            Q1 = df[var].quantile(0.25)
            Q3 = df[var].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR

            outliers = len(df[(df[var] < lower_bound) | (df[var] > upper_bound)])
            total = len(df[var])
            percent_outliers = (outliers / total) * 100

            verdict = "Remove" if percent_outliers >= min_percent else "Keep"

            results[var] = {
                "Method": "IQR",
                "Lower Threshold": round(lower_bound, 2),
                "Upper Threshold": round(upper_bound, 2),
                "Outliers": outliers,
                "Percent Outliers": round(percent_outliers, 2),
                "Verdict": verdict
            }
    return pd.DataFrame(results).T


# --- Example Usage ---
num_vars = ["Age", "BMI", "Weight", "Height", "Premium_Price"]  # update with your numeric cols
z_results = z_score_outliers(df, num_vars, threshold=3, min_percent=5)
iqr_results = iqr_outliers(df, num_vars, min_percent=5)

print(" 🔹 Z-Score Outlier Report")
display(z_results)

print("\n 🔹 IQR Outlier Report")
display(iqr_results)
```

🔹 Z-Score Outlier Report

	Method	Lower Threshold	Upper Threshold	Outliers	Percent Outliers	Verdict
Age	Z-Score	-0.14	83.64	0	0.00	Keep
BMI	Z-Score	9.82	45.10	7	0.71	Keep
Weight	Z-Score	34.16	119.75	13	1.32	Keep
Height	Z-Score	137.89	198.48	0	0.00	Keep
Premium_Price	Z-Score	5592.16	43081.27	0	0.00	Keep

🔹 IQR Outlier Report

	Method	Lower Threshold	Upper Threshold	Outliers	Percent Outliers	Verdict
Age	IQR	-4.50	87.50	0	0.00	Keep
BMI	IQR	12.34	41.81	22	2.23	Keep
Weight	IQR	37.00	117.00	16	1.62	Keep
Height	IQR	138.50	198.50	0	0.00	Keep
Premium_Price	IQR	10500.00	38500.00	6	0.61	Keep

In [147]:
```python
# --- Function to detect outliers using both methods ---
def outlier_summary(df, vars, z_threshold=3, min_percent=5):
    results = []

    for var in vars:
        if var in df.columns:
            total = len(df[var])

            # --- Z-Score Method ---
            mean = df[var].mean()
            std_dev = df[var].std()
            z_scores = np.abs(stats.zscore(df[var]))
            z_outliers = len(df[z_scores > z_threshold])
            z_percent = (z_outliers / total) * 100
            z_lower = mean - z_threshold * std_dev
            z_upper = mean + z_threshold * std_dev
            z_verdict = "Remove" if z_percent >= min_percent else "Keep"

            # --- IQR Method ---
            Q1 = df[var].quantile(0.25)
            Q3 = df[var].quantile(0.75)
            IQR = Q3 - Q1
            iqr_lower = Q1 - 1.5 * IQR
            iqr_upper = Q3 + 1.5 * IQR
            iqr_outliers = len(df[(df[var] < iqr_lower) | (df[var] > iqr_upper)])
            iqr_percent = (iqr_outliers / total) * 100
            iqr_verdict = "Remove" if iqr_percent >= min_percent else "Keep"

            results.append({
                "Variable": var,
                "Z_Lower": round(z_lower, 2),
                "Z_Upper": round(z_upper, 2),
                "Z_Outliers": z_outliers,
                "Z_%": round(z_percent, 2),
                "Z_Verdict": z_verdict,
                "IQR_Lower": round(iqr_lower, 2),
                "IQR_Upper": round(iqr_upper, 2),
                "IQR_Outliers": iqr_outliers,
                "IQR_%": round(iqr_percent, 2),
                "IQR_Verdict": iqr_verdict
            })

    return pd.DataFrame(results)


# --- Example Usage ---
num_vars = ["Age", "BMI", "Weight", "Height", "Premium_Price"]  # update with your numeric cols
outlier_report = outlier_summary(df, num_vars, z_threshold=3, min_percent=5)

print("📊 Combined Outlier Report")
display(outlier_report)
```

📊 Combined Outlier Report

	Variable	Z_Lower	Z_Upper	Z_Outliers	Z_%	Z_Verdict	IQR_Lower	IQR_Upper	IQR_Outliers	IQR_%	IQR_Verdict
0	Age	-0.14	83.64	0	0.00	Keep	-4.50	87.50	0	0.00	Keep
1	BMI	9.82	45.10	7	0.71	Keep	12.34	41.81	22	2.23	Keep
2	Weight	34.16	119.75	13	1.32	Keep	37.00	117.00	16	1.62	Keep
3	Height	137.89	198.48	0	0.00	Keep	138.50	198.50	0	0.00	Keep
4	Premium_Price	5592.16	43081.27	0	0.00	Keep	10500.00	38500.00	6	0.61	Keep

3. Hypothesis Testing

Framework

STEP - ***1***: Setup NULL and ALTERNATE Hypothesis.

- `NULL Hypothesis (H0)` - Holidays ***DO NOT have*** any effect on the Number of Electric Cycles Rented.
- `ALTERNATE Hypothesis (Ha)` - Holidays ***DO have*** some effect on the Number of Electric Cycles Rented.

STEP - ***2***: Check for Basic Assumptions (Determine the Distribution) for the Hypothesis.

- *Distribution* Check using `Q-Q Plot`
- *Homogeneity of Variances* using `Levene's Test`

STEP - ***3***: Define the Test Statistics; Distribution of T under NULL Hypothesis.

- The Assumptions for the T Test are met then we can proceed performing *T Test for Independent Samples* else we will perform the Non-Parametric test equivalent to T- Test for Independent Sample i.e., `Mann - Whitney U Rank Test` for two Independent Samples.

STEP - ***4***: Compute the P-Value and Set a Value for Alpha (Significance Level).

- We will Set the `Significance Level ( ∝ )` as ***0.05***

STEP - ***5***: Compare P-Value with Alpha (∝).

- Based on the P-Value, We will Accept or Reject the NULL Hypothesis
  - `P-Value < ∝` : We *Reject* the NULL Hypothesis
  - `P-Value > ∝` : We *Fail to Reject* the NULL Hypothesis

In [29]:
```python
results_summary = []   # global list to collect results
```

3.1 - Binary Features (0/1 variables)

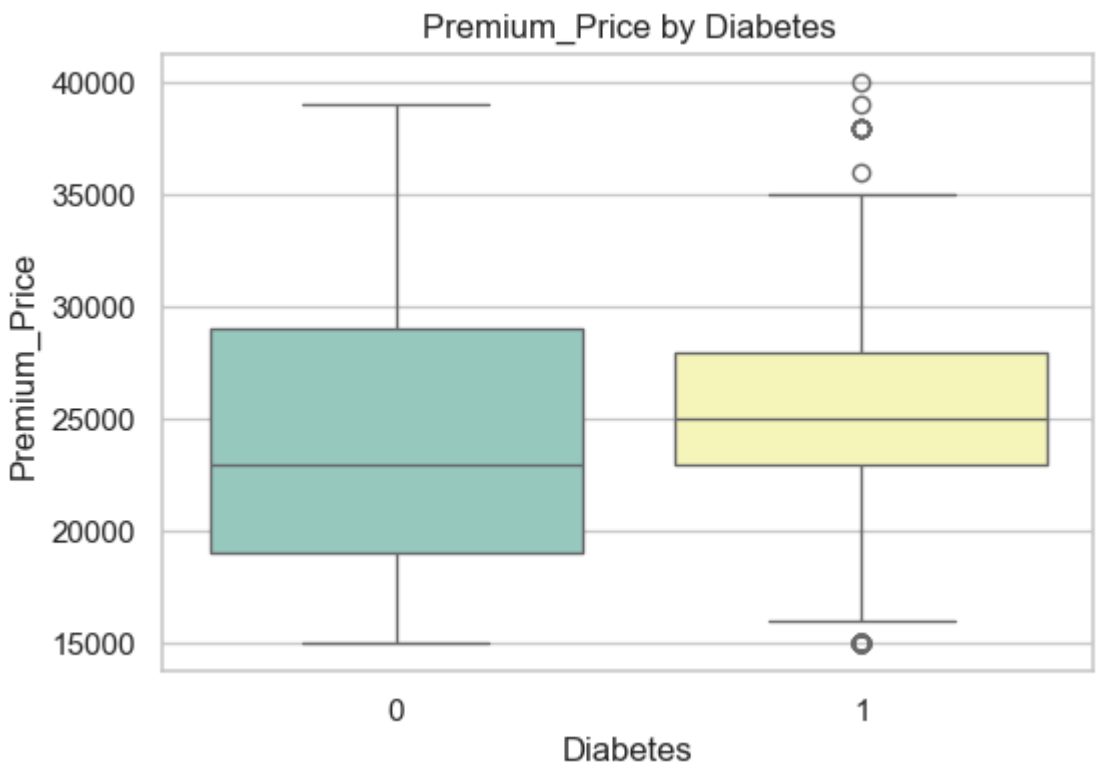
In [30]:
```python
def binary_hypothesis(data, feature, target="Premium_Price"):

    print(f"\n{'='*60}\nFeature: {feature} vs {target}\n{'='*60}")
    print(f"H0: {feature} has no significant effect on {target}")
    print(f"Ha: {feature} DOES have a significant effect on {target}\n")

    group0 = data[data[feature]==0][target]
    group1 = data[data[feature]==1][target]

    # --- Boxplot ---
    plt.figure(figsize=(6,4))
    sns.boxplot(x=feature, y=target, data=data, palette="Set3")
    plt.title(f"{target} by {feature}")
    plt.show()

    # --- Histogram + Q-Q Plots ---
    fig, axes = plt.subplots(2, 2, figsize=(10,6))
    sns.histplot(group0, kde=True, ax=axes[0,0], color="skyblue")
    axes[0,0].set_title(f"Histogram: {feature}=0")
    sm.qqplot(group0, line='s', ax=axes[0,1])
    axes[0,1].set_title(f"Q-Q Plot: {feature}=0")
    sns.histplot(group1, kde=True, ax=axes[1,0], color="salmon")
    axes[1,0].set_title(f"Histogram: {feature}=1")
    sm.qqplot(group1, line='s', ax=axes[1,1])
    axes[1,1].set_title(f"Q-Q Plot: {feature}=1")
    plt.tight_layout()
    plt.show()

    # --- Shapiro-Wilk Test ---
    stat0, p0 = stats.shapiro(group0)
    stat1, p1 = stats.shapiro(group1)
    print(f"Shapiro Test -> {feature}=0: stat={stat0:.3f}, p={p0:.5f}")
    print(f"Shapiro Test -> {feature}=1: stat={stat1:.3f}, p={p1:.5f}")

    normality = (p0 > 0.05) and (p1 > 0.05)
    if not normality:
        print("❌ Normality assumption failed (p<0.05). Will use Mann-Whitney U test.\n")
    else:
        print("✅ Normality assumption holds for both groups.\n")

    # --- Levene’s Test for Equal Variances ---
    lev_stat, lev_p = stats.levene(group0, group1)
    print(f"Levene’s Test: stat={lev_stat:.3f}, p={lev_p:.5f}")
    equal_var = lev_p > 0.05
    if normality:
        if equal_var:
            print("✅ Variances are equal. Will use Independent T-test.\n")
        else:
            print("❌ Variances unequal. Will use Welch’s T-test.\n")

    # --- Hypothesis Test Selection ---
    if normality:  # Normal distribution
        if equal_var:
            test_name = "Independent T-test"
            test_stat, test_p = stats.ttest_ind(group0, group1, equal_var=True)
            notes = "Independent T-test used (assumptions satisfied)"
        else:
            test_name = "Welch’s T-test"
            test_stat, test_p = stats.ttest_ind(group0, group1, equal_var=False)
            notes = "Welch’s T-test used (variance unequal)"
    else:  # Non-parametric fallback
        test_name = "Mann-Whitney U Test"
```

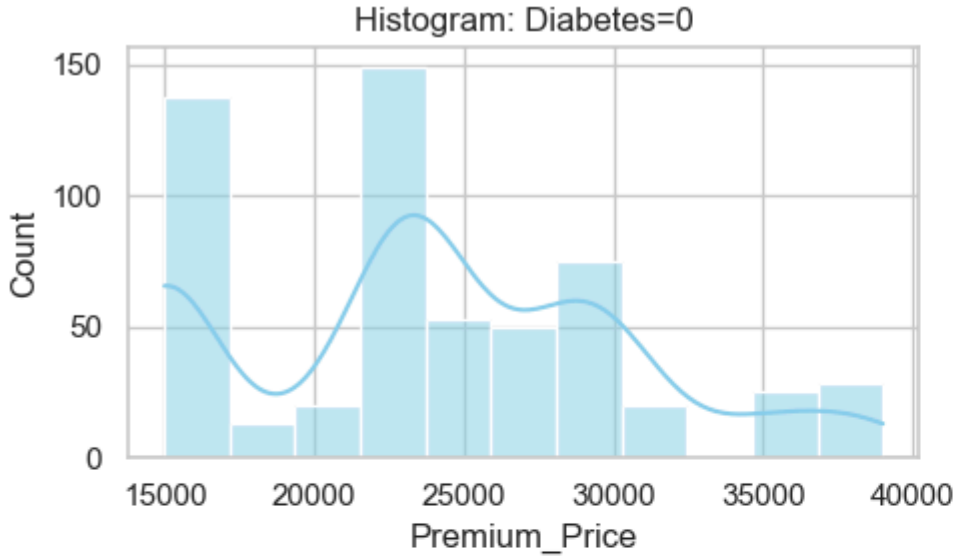
```
        test_stat, test_p = stats.mannwhitneyu(group0, group1, alternative='two-sided')
        notes = "Mann-Whitney U Test used (normality failed)"

    # --- Results ---
    print(f"{test_name} → stat={test_stat:.3f}, p={test_p:.5f}")
    alpha = 0.05
    if test_p < alpha:
        print(f"✅ Reject H0 → {feature} significantly affects {target}.")
    else:
        print(f"❌ Fail to Reject H0 → No significant effect of {feature} on {target}.")

    # --- Decision ---
    decision = "Reject H0" if test_p < alpha else "Fail to Reject H0"

    results_summary.append({
        "Feature": feature,
        "Target": target,
        "Test": test_name,
        "Stat Value": round(test_stat, 3),
        "p-value": round(test_p, 5),
        "Decision": decision,
        "Notes": notes
    })
```

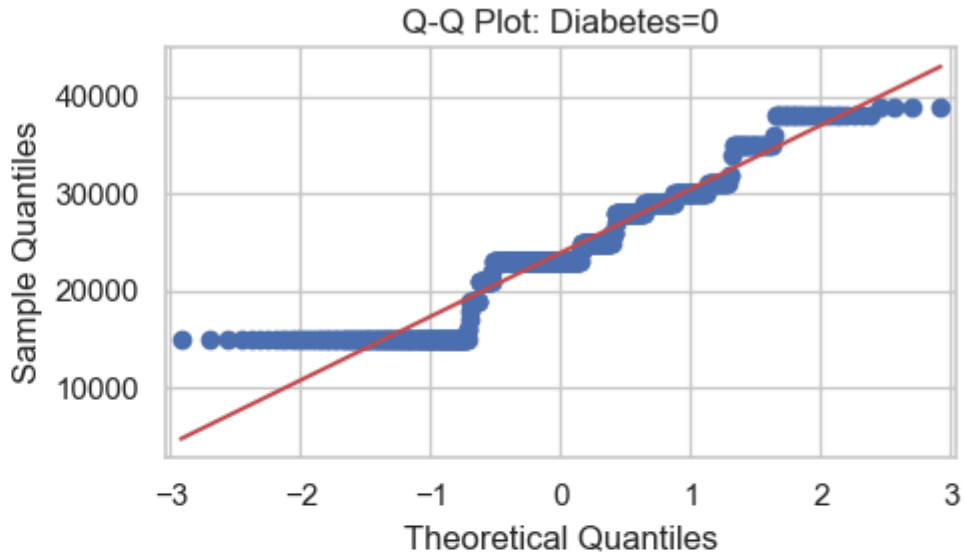
3.1.1 - Diabetes

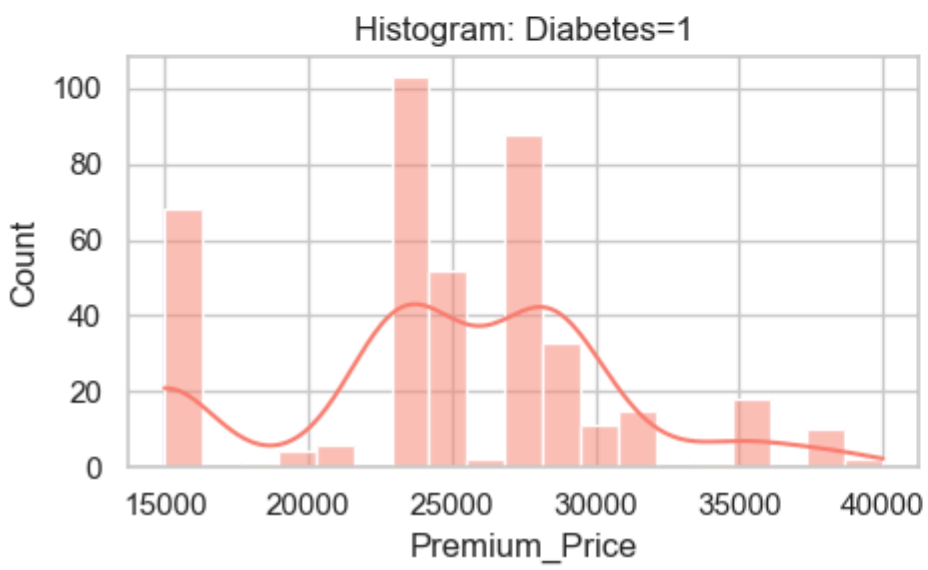
In [31]:
```
binary_hypothesis(df, "Diabetes")
```

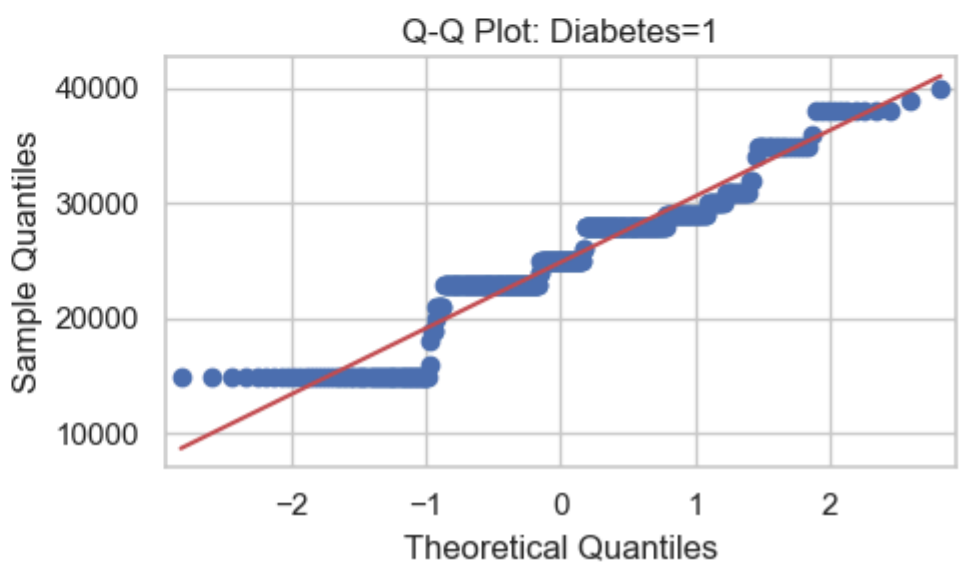
```
============================================================
Feature: Diabetes vs Premium_Price
============================================================
H0: Diabetes has no significant effect on Premium_Price
Ha: Diabetes DOES have a significant effect on Premium_Price
```











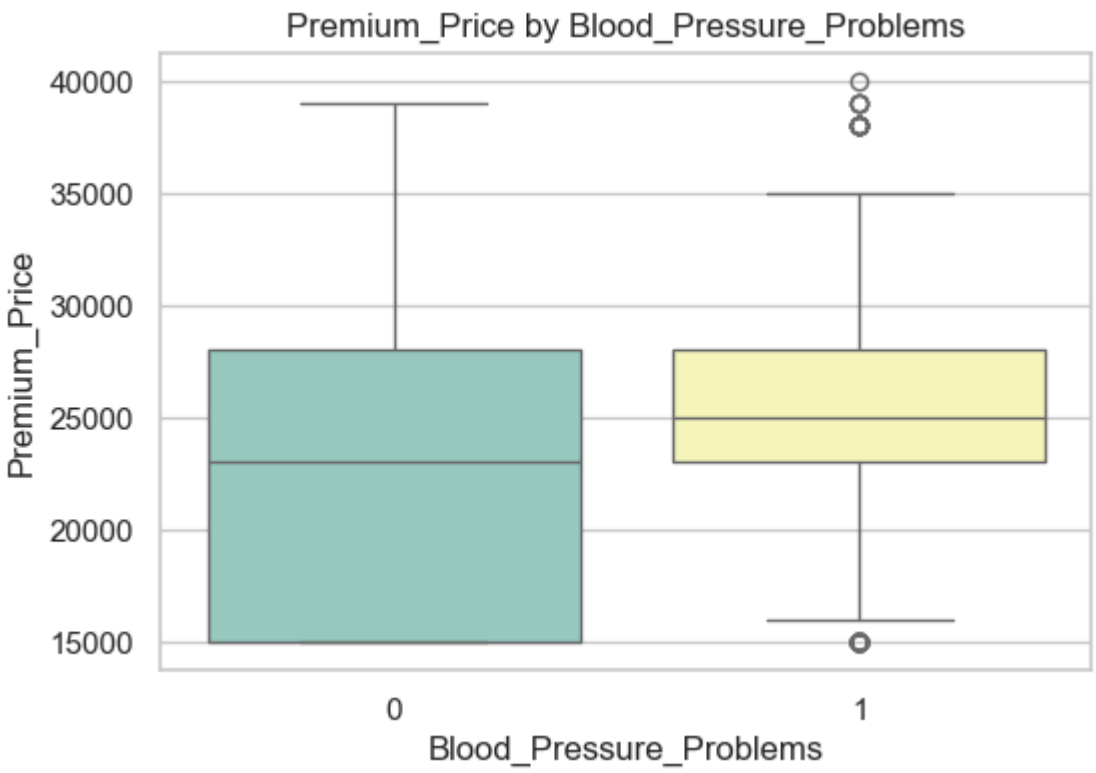
```
Shapiro Test -> Diabetes=0: stat=0.921, p=0.00000
Shapiro Test -> Diabetes=1: stat=0.923, p=0.00000
❌ Normality assumption failed (p<0.05). Will use Mann–Whitney U test.

Levene’s Test: stat=7.033, p=0.00813
Mann–Whitney U Test → stat=106563.500, p=0.00648
✅ Reject H0 → Diabetes significantly affects Premium_Price.
```

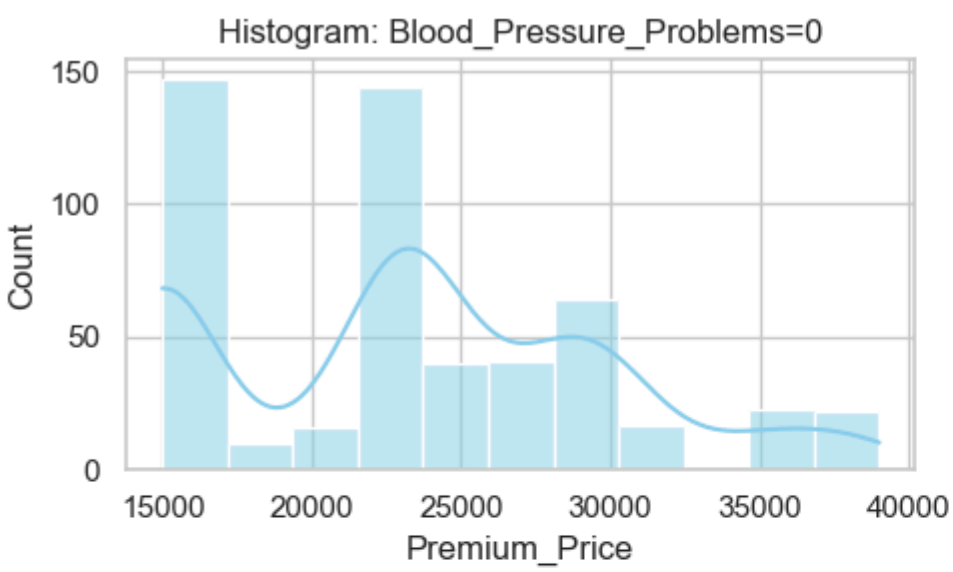
Insights

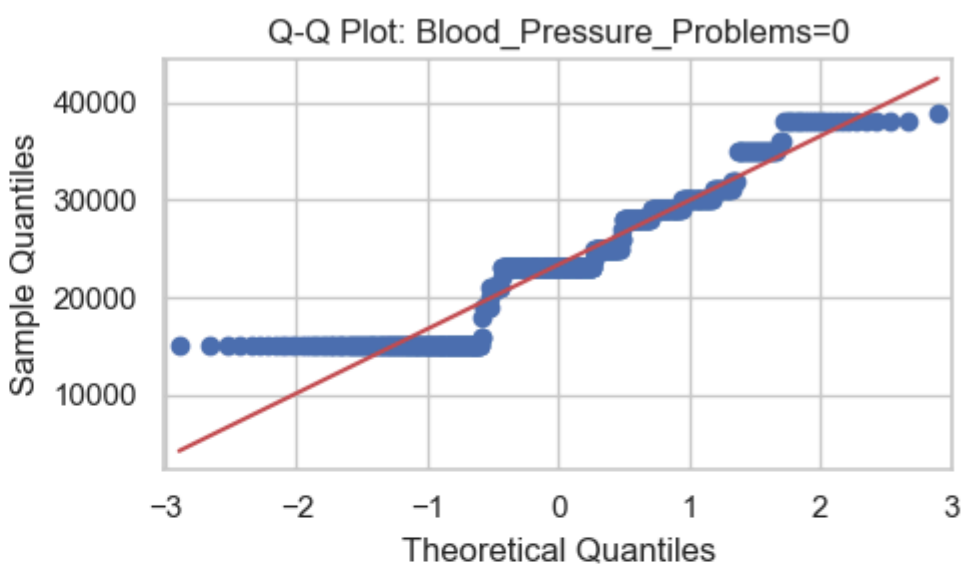
3.1.2 - Blood Pressure Problems

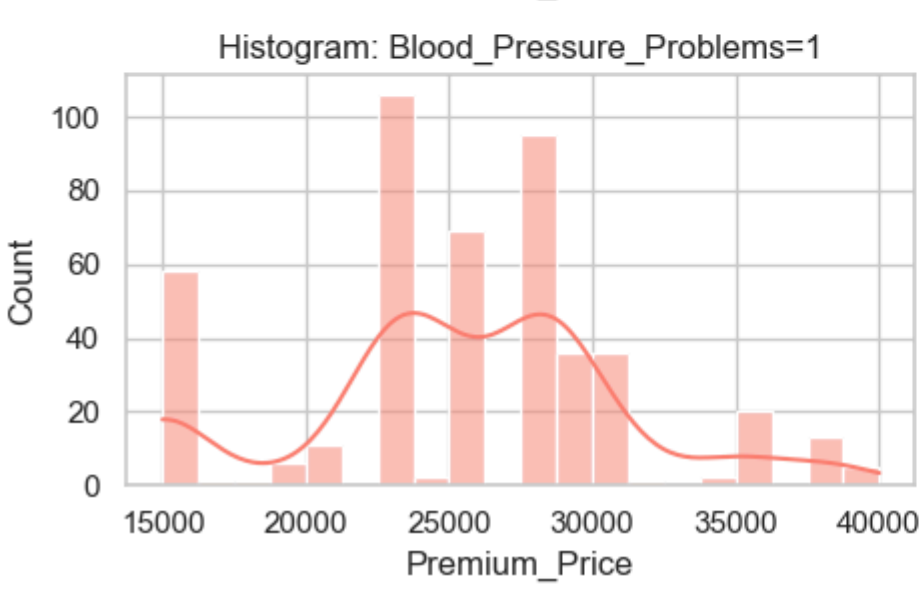
In [32]:
```
binary_hypothesis(df, "Blood_Pressure_Problems")
```

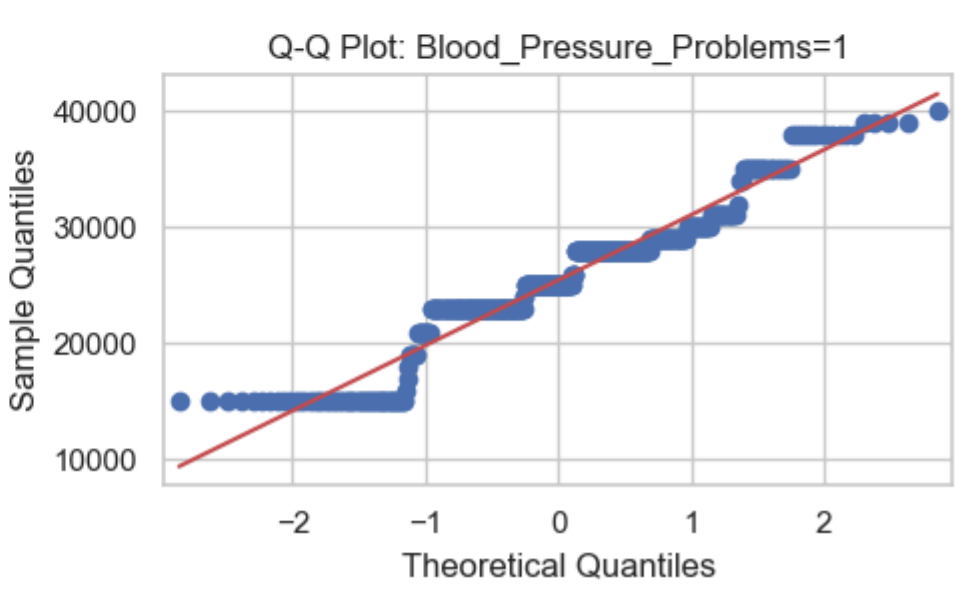
```
============================================================
Feature: Blood_Pressure_Problems vs Premium_Price
============================================================
H0: Blood_Pressure_Problems has no significant effect on Premium_Price
Ha: Blood_Pressure_Problems DOES have a significant effect on Premium_Price
```











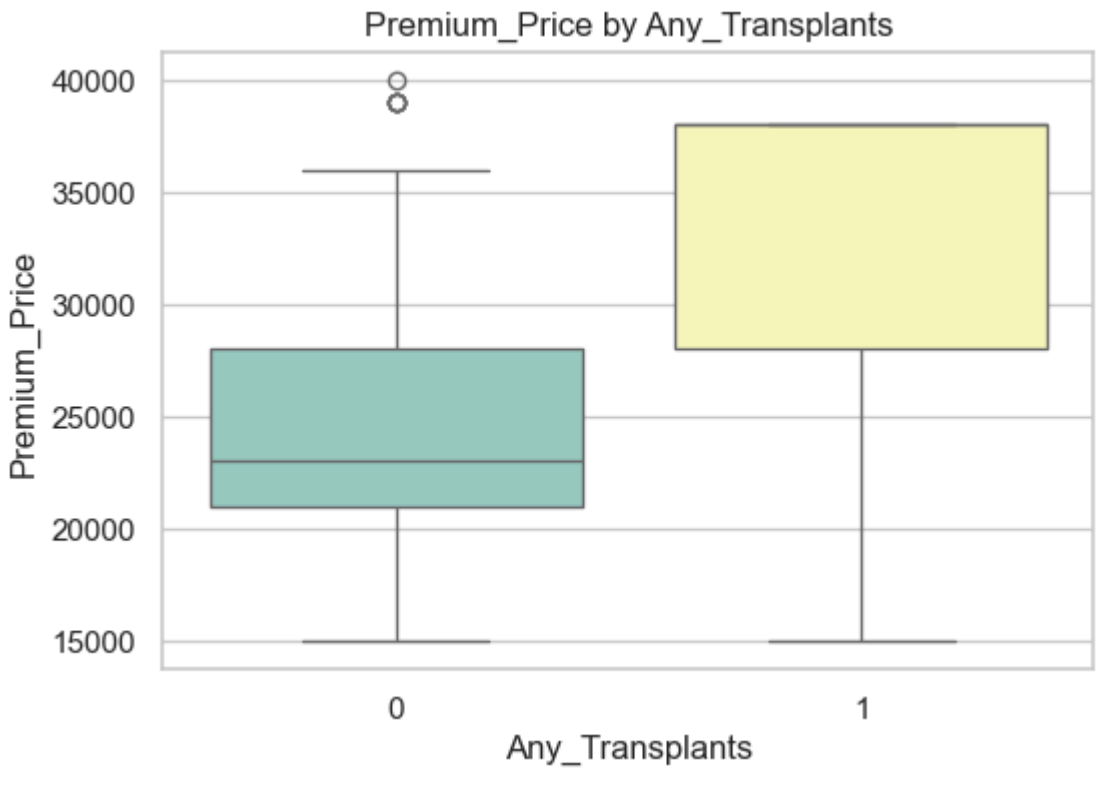

```
Shapiro Test -> Blood_Pressure_Problems=0: stat=0.906, p=0.00000
Shapiro Test -> Blood_Pressure_Problems=1: stat=0.935, p=0.00000
❌ Normality assumption failed (p<0.05). Will use Mann–Whitney U test.

Levene’s Test: stat=10.948, p=0.00097
Mann–Whitney U Test → stat=96697.000, p=0.00000
✅ Reject H0 → Blood_Pressure_Problems significantly affects Premium_Price.
```

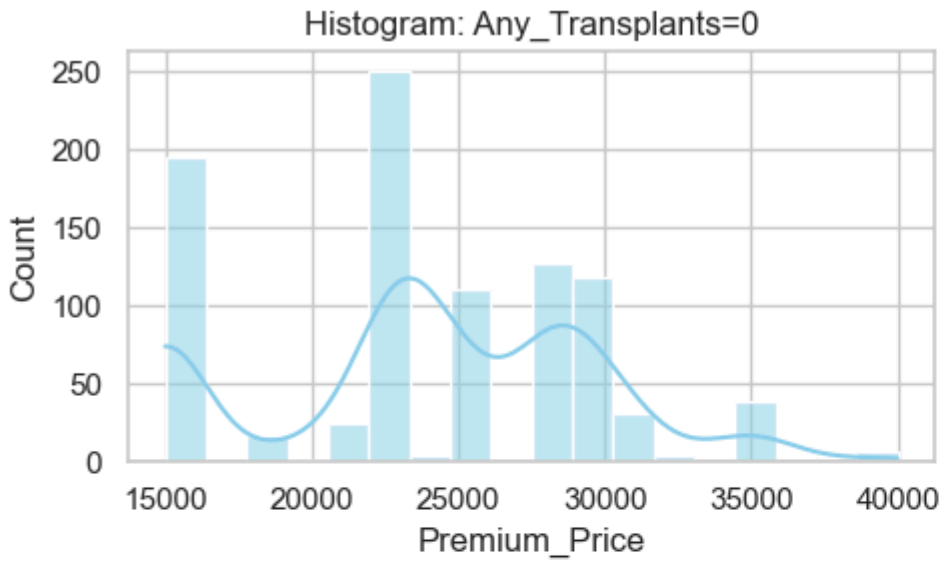
Insights

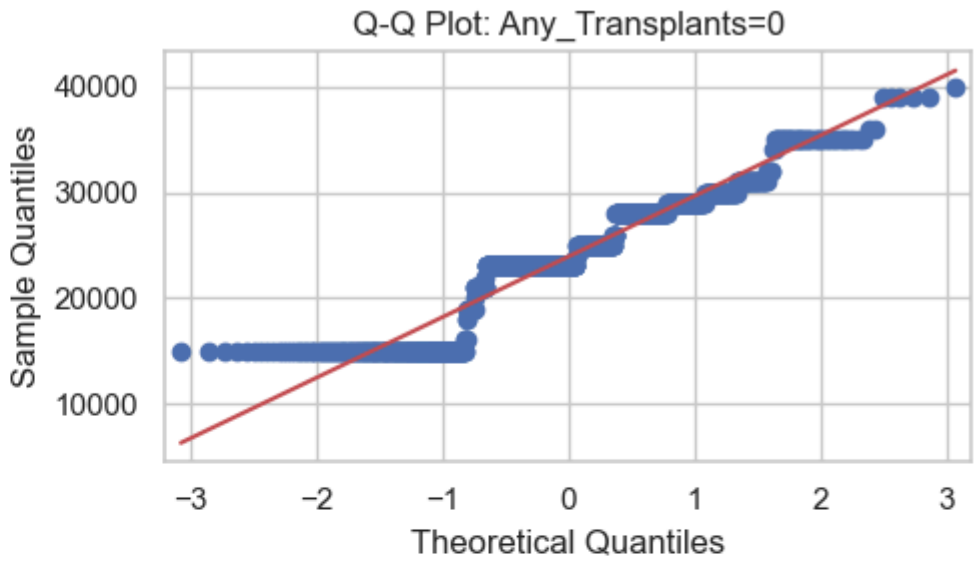
3.1.3 - Any Transplants

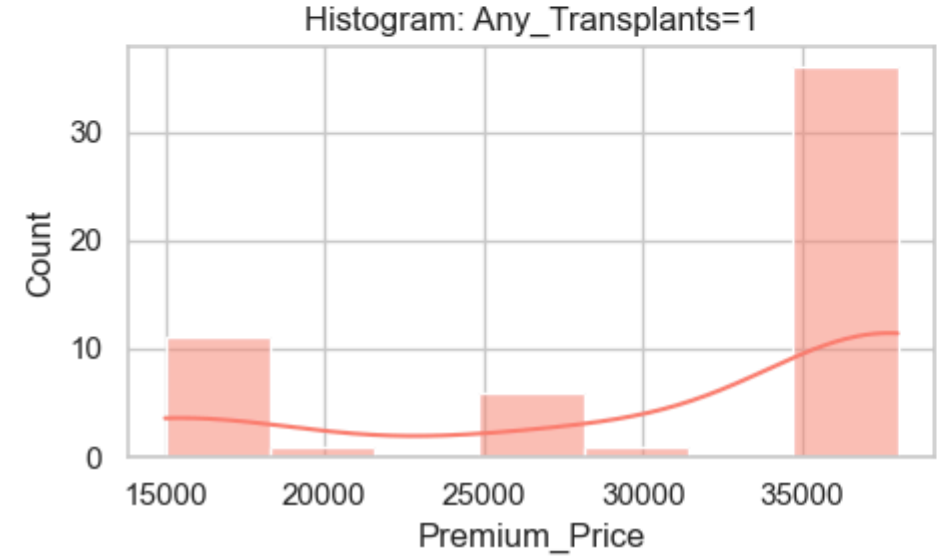
In [33]:
```
binary_hypothesis(df, "Any_Transplants")
```

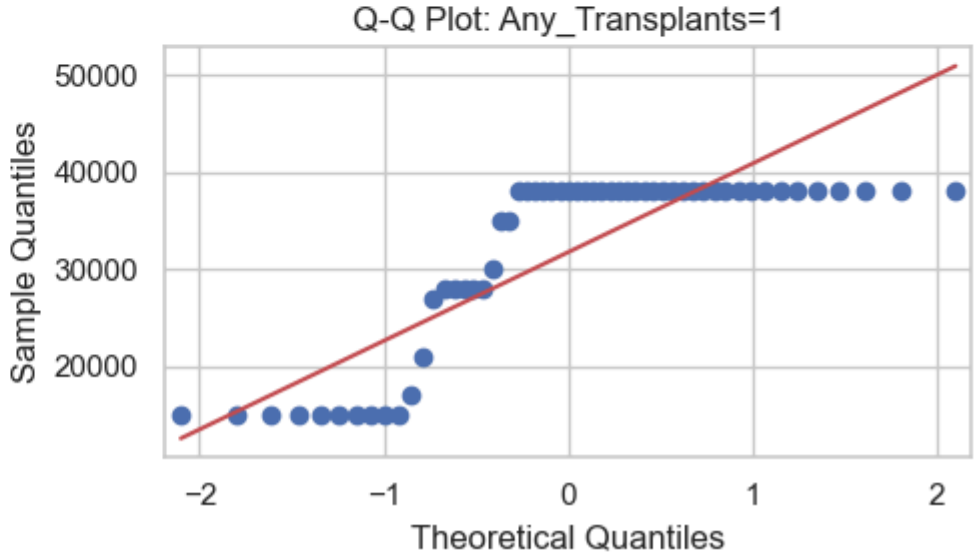
```
============================================================
Feature: Any_Transplants vs Premium_Price
============================================================
H0: Any_Transplants has no significant effect on Premium_Price
Ha: Any_Transplants DOES have a significant effect on Premium_Price
```











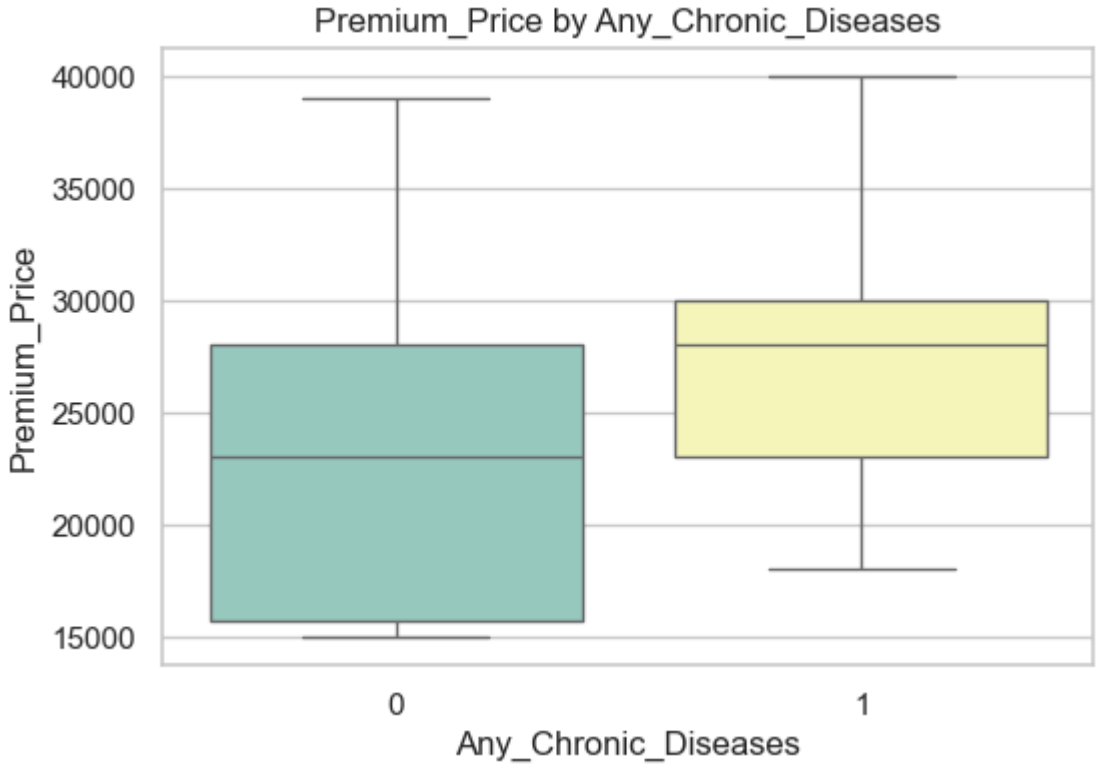

```
Shapiro Test -> Any_Transplants=0: stat=0.920, p=0.00000
Shapiro Test -> Any_Transplants=1: stat=0.663, p=0.00000
❌ Normality assumption failed (p<0.05). Will use Mann-Whitney U test.

Levene’s Test: stat=8.935, p=0.00287
Mann-Whitney U Test → stat=11814.000, p=0.00000
✅ Reject H0 → Any_Transplants significantly affects Premium_Price.
```

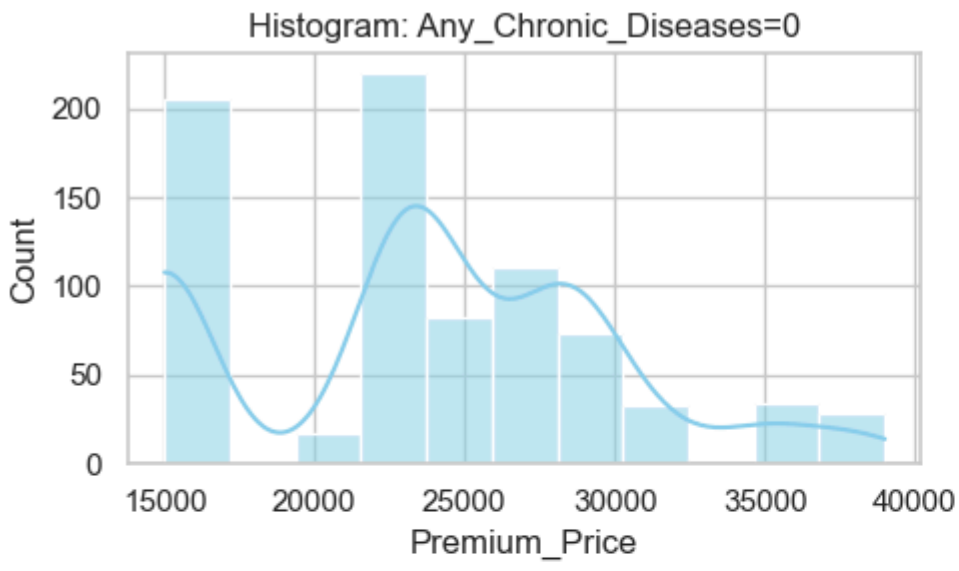
Insights

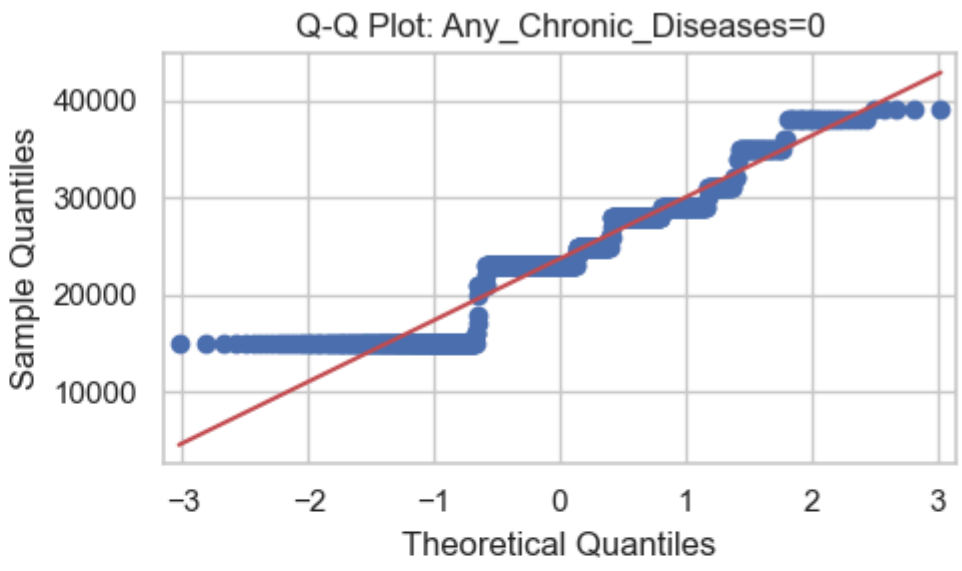
3.1.4 - Any Chronic Diseases

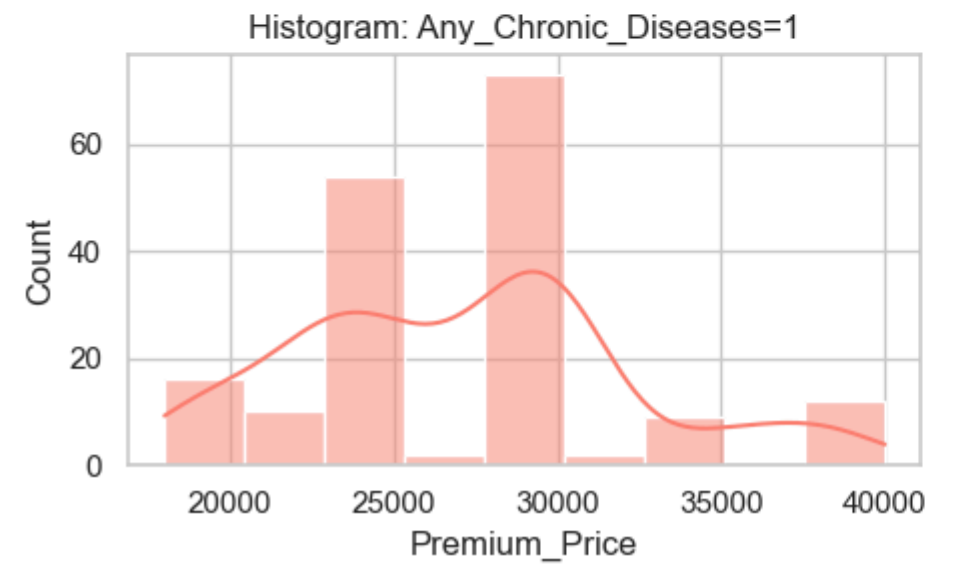
In [34]:
```
binary_hypothesis(df, "Any_Chronic_Diseases")
```

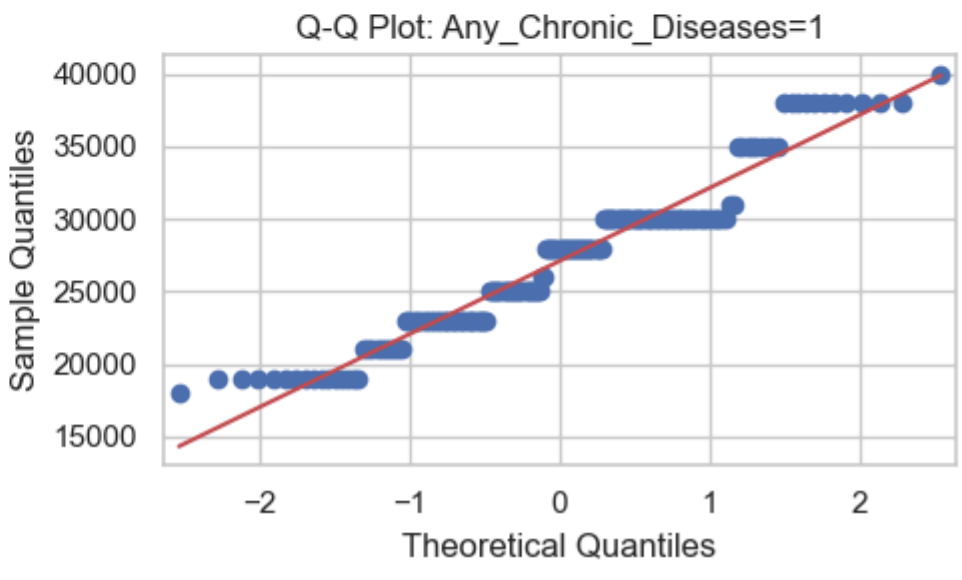
```
============================================================
Feature: Any_Chronic_Diseases vs Premium_Price
============================================================
H0: Any_Chronic_Diseases has no significant effect on Premium_Price
Ha: Any_Chronic_Diseases DOES have a significant effect on Premium_Price
```











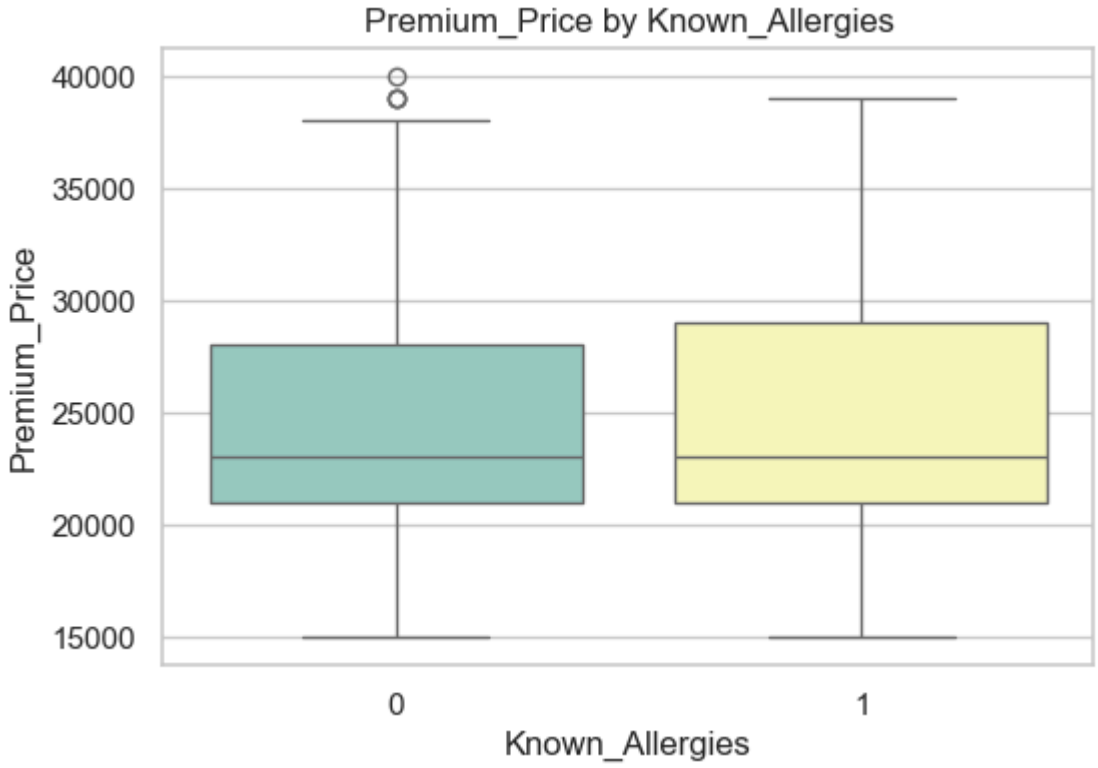

```
Shapiro Test -> Any_Chronic_Diseases=0: stat=0.910, p=0.00000
Shapiro Test -> Any_Chronic_Diseases=1: stat=0.940, p=0.00000
❌ Normality assumption failed (p<0.05). Will use Mann–Whitney U test.
Levene’s Test: stat=6.508, p=0.01089
Mann–Whitney U Test → stat=49243.500, p=0.00000
✅ Reject H0 → Any_Chronic_Diseases significantly affects Premium_Price.
```

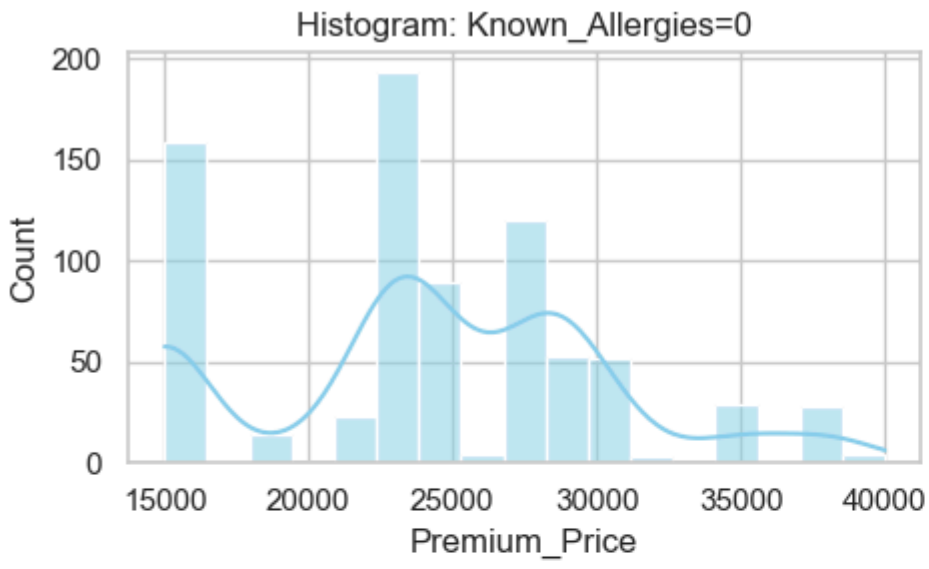
Insights

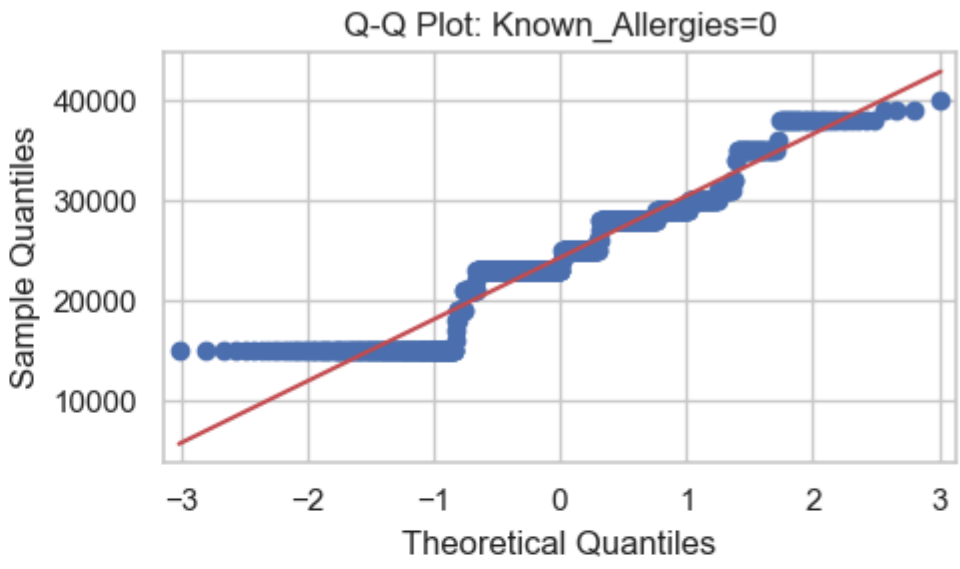
3.1.5 - Known Allergies

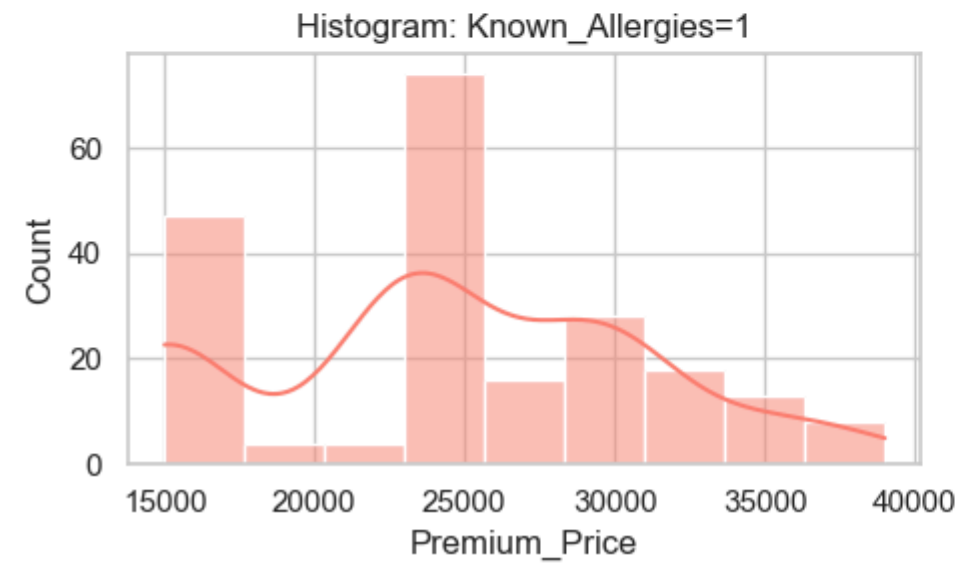
In [35]:
```
binary_hypothesis(df, "Known_Allergies")
```

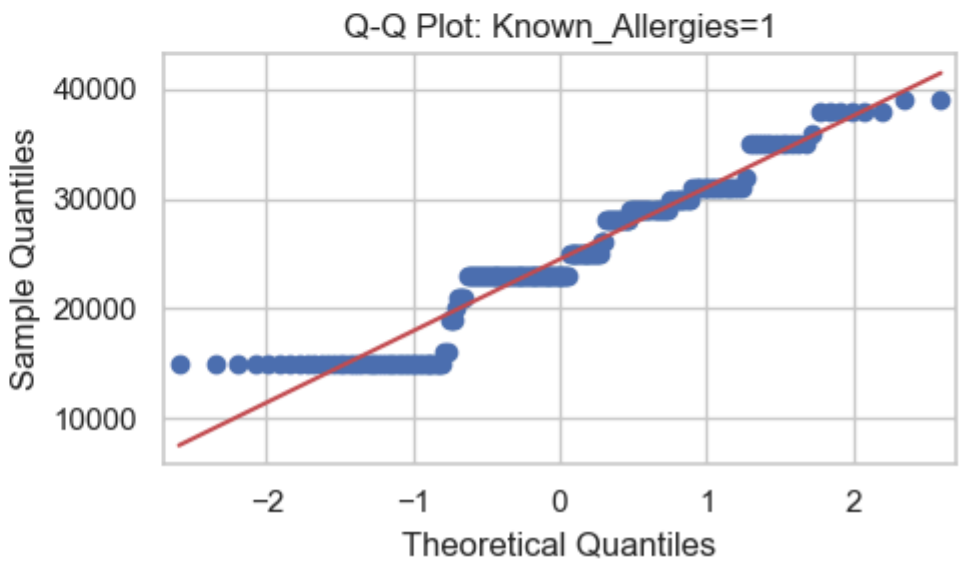
```
============================================================
Feature: Known_Allergies vs Premium_Price
============================================================
H0: Known_Allergies has no significant effect on Premium_Price
Ha: Known_Allergies DOES have a significant effect on Premium_Price
```











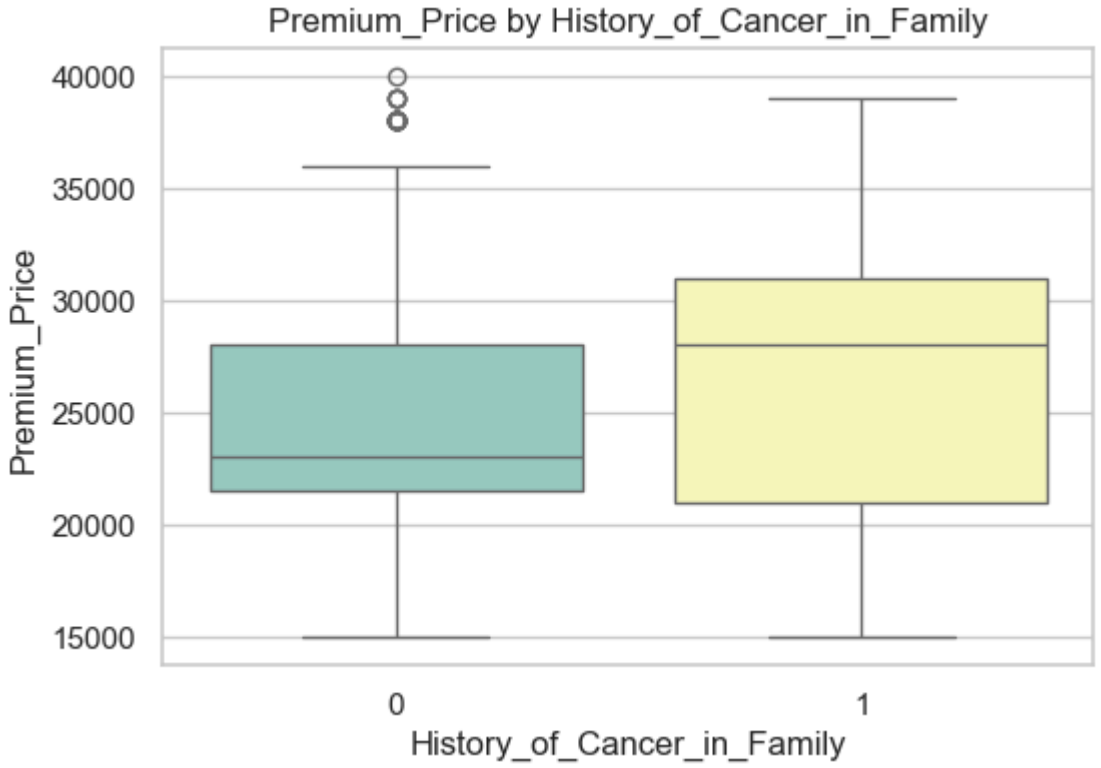

```
Shapiro Test -> Known_Allergies=0: stat=0.925, p=0.00000
Shapiro Test -> Known_Allergies=1: stat=0.927, p=0.00000
❌ Normality assumption failed (p<0.05). Will use Mann–Whitney U test.
Levene’s Test: stat=1.484, p=0.22346
Mann–Whitney U Test → stat=79964.500, p=0.56578
❌ Fail to Reject H0 → No significant effect of Known_Allergies on Premium_Price.
```

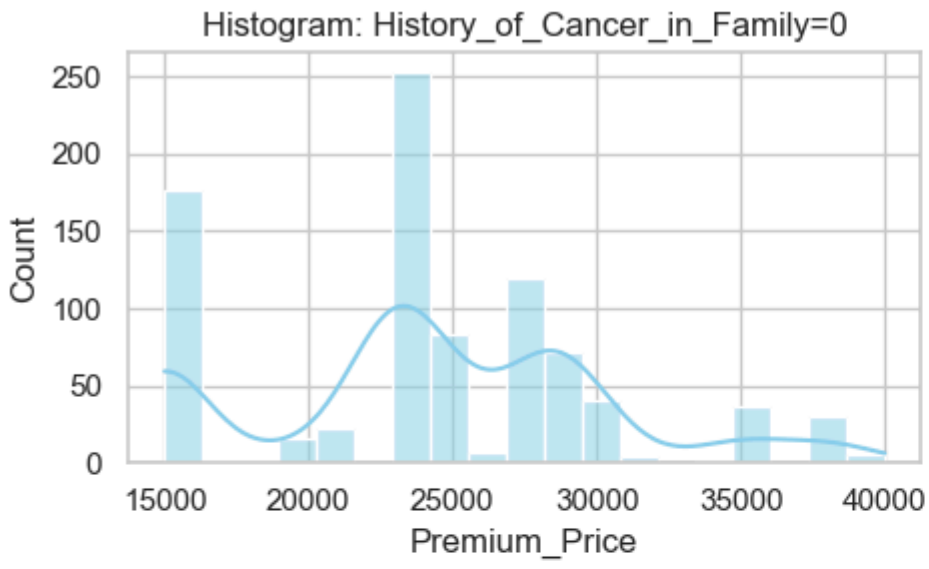
Insights

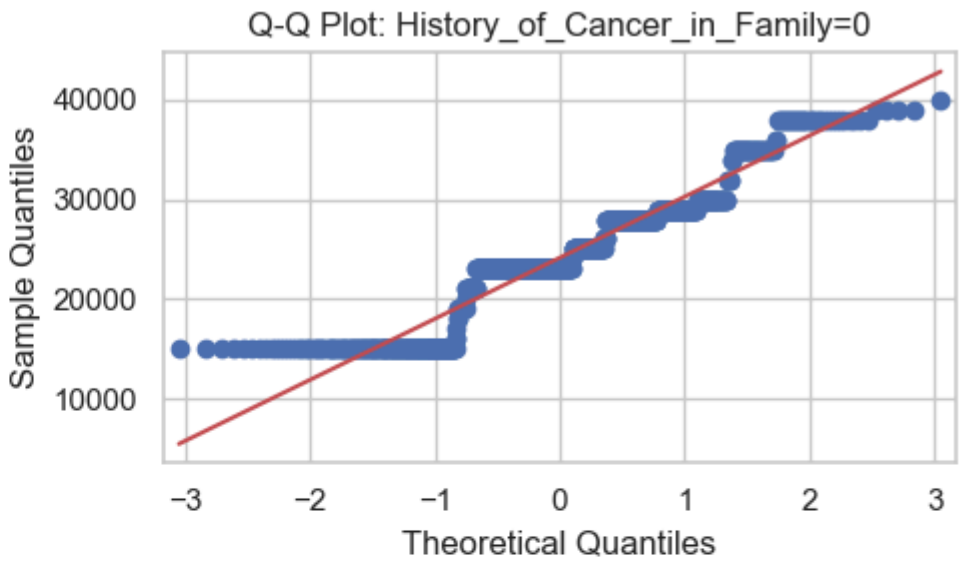
3.1.6 - History of Cancer in Family

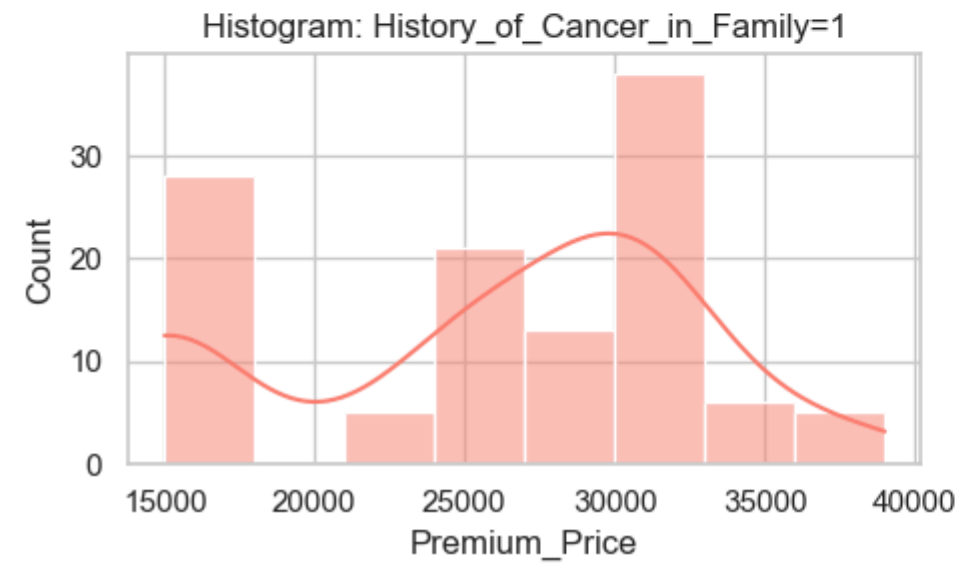
In [36]:
```
binary_hypothesis(df, "History_of_Cancer_in_Family")
```

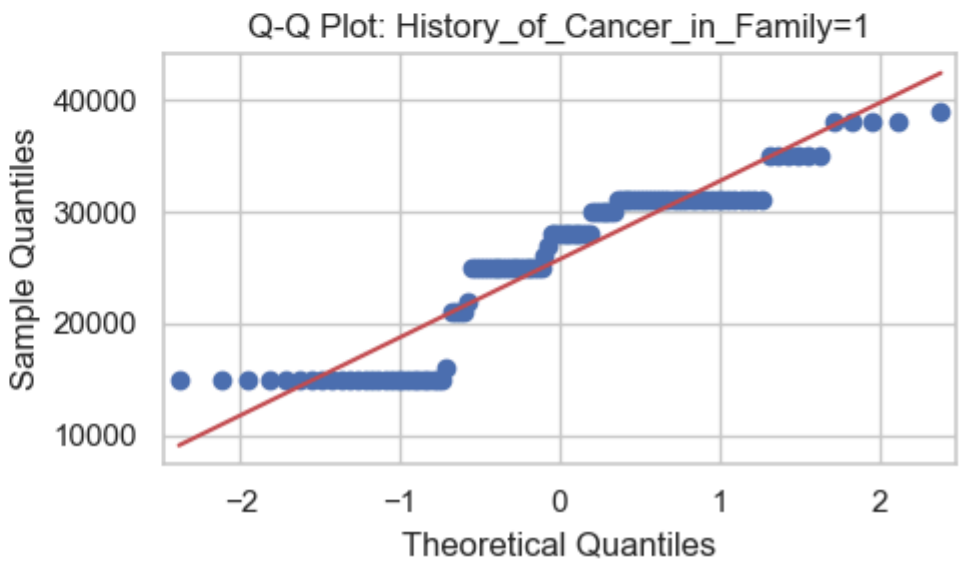
```
============================================================
Feature: History_of_Cancer_in_Family vs Premium_Price
============================================================
H0: History_of_Cancer_in_Family has no significant effect on Premium_Price
Ha: History_of_Cancer_in_Family DOES have a significant effect on Premium_Price
```












```
Shapiro Test -> History_of_Cancer_in_Family=0: stat=0.921, p=0.00000
Shapiro Test -> History_of_Cancer_in_Family=1: stat=0.876, p=0.00000
❌ Normality assumption failed (p<0.05). Will use Mann–Whitney U test.
Levene’s Test: stat=5.998, p=0.01450
Mann–Whitney U Test → stat=39412.500, p=0.00010
✅ Reject H0 → History_of_Cancer_in_Family significantly affects Premium_Price.
```

Insights

3.2 - Categorical Hypothesis Testing (3+ categories, e.g. Surgeries)

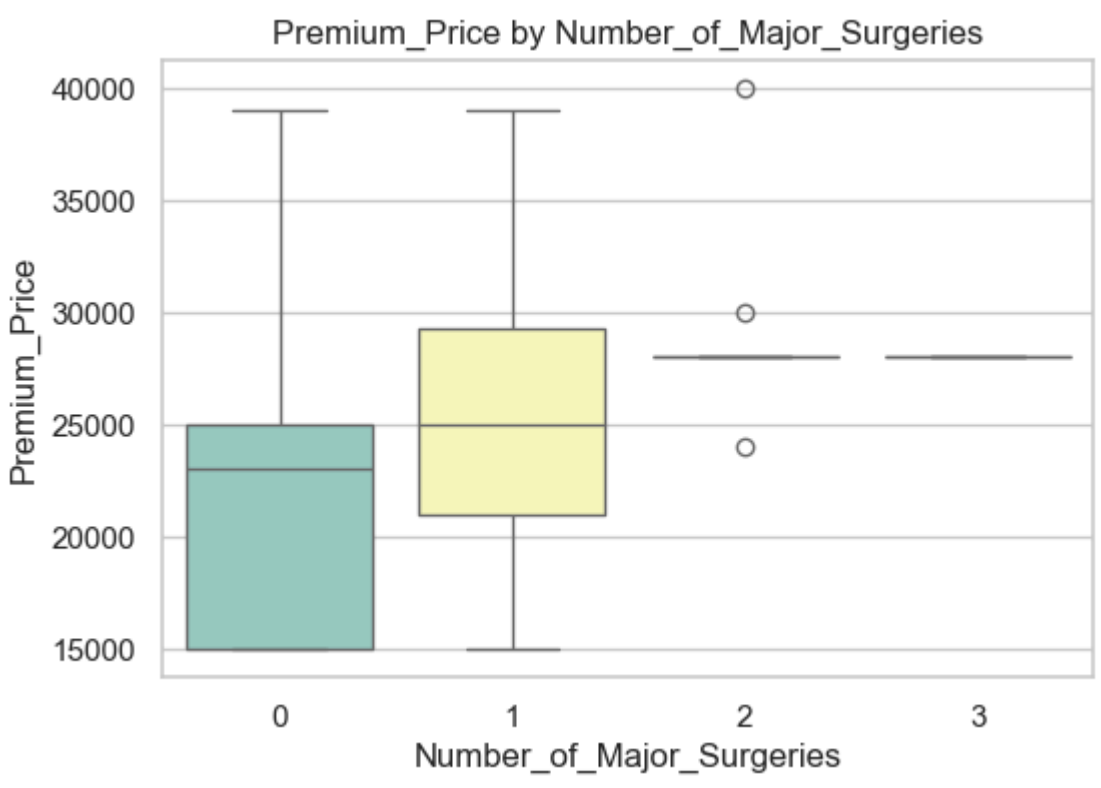
In [37]:
```python
def categorical_hypothesis(data, feature, target="Premium_Price"):
    print(f"\n{'='*60}\nFeature: {feature} vs {target}\n{'='*60}")
    print(f"H0: Mean {target} is equal across all groups of {feature}")
    print(f"Ha: At least one group of {feature} differs significantly in {target}\n")

    groups = [data[data[feature]==cat][target] for cat in sorted(data[feature].unique())]

    # --- Boxplot ---
    plt.figure(figsize=(6,4))
    sns.boxplot(x=feature, y=target, data=data, palette="Set3")
    plt.title(f"{target} by {feature}")
    plt.show()

    # --- Histogram + Q-Q for each group ---
    fig, axes = plt.subplots(len(groups), 2, figsize=(10,4*len(groups)))
    for i, g in enumerate(groups):
        sns.histplot(g, kde=True, ax=axes[i,0], color="skyblue")
        axes[i,0].set_title(f"Histogram: {feature}={i}")
        sm.qqplot(g, line='s', ax=axes[i,1])
        axes[i,1].set_title(f"Q-Q Plot: {feature}={i}")
    plt.tight_layout()
```

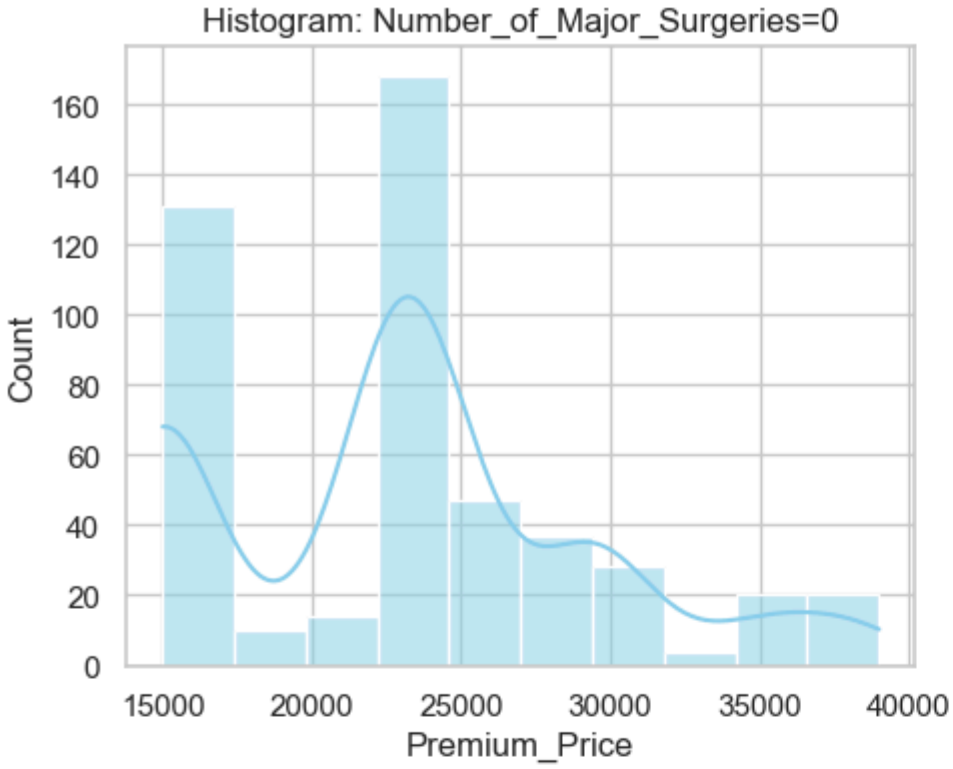
```
plt.show()

# --- Shapiro Normality Test for each group ---
normality = True
for i, g in enumerate(groups):
    stat, p = stats.shapiro(g)
    print(f"Shapiro Test -> {feature}={i}: stat={stat:.3f}, p={p:.5f}")
    if p < 0.05:
        normality = False
        print("❌ Normality failed for this group.")
if normality:
    print("✅ All groups pass normality.\n")
else:
    print("❌ At least one group failed normality → fallback to non-parametric test.\n")

# --- Levene's Test for variance homogeneity ---
lev_stat, lev_p = stats.levene(*groups)
print(f"Levene's Test: stat={lev_stat:.3f}, p={lev_p:.5f}")
equal_var = lev_p > 0.05
if equal_var:
    print("✅ Variances are equal.\n")
else:
    print("❌ Variances are unequal.\n")

# --- Decide test ---
if normality:
    if equal_var:
        test_name = "ANOVA"
        f_stat, p_val = stats.f_oneway(*groups)
        notes = "ANOVA used (all assumptions satisfied)"
    else:
        test_name = "Welch's ANOVA"
        f_stat, p_val = stats.f_oneway(*groups)  # SciPy doesn't have true Welch's ANOVA
        notes = "Welch's ANOVA used (variance unequal)"
else:
    test_name = "Kruskal-Wallis Test"
    f_stat, p_val = stats.kruskal(*groups)
    notes = "Kruskal-Wallis used (normality failed)"

# --- Results ---
print(f"{test_name} → stat={f_stat:.3f}, p={p_val:.5f}")
alpha = 0.05
if p_val < alpha:
    print(f"✅ Reject H0 → At least one group of {feature} has significantly different {target}.")
else:
    print(f"❌ Fail to Reject H0 → No significant difference in {target} across {feature}.")

# --- Decision ---
decision = "Reject H0" if p_val < alpha else "Fail to Reject H0"

results_summary.append({
    "Feature": feature,
    "Target": target,
    "Test": test_name,
    "Stat Value": round(f_stat, 3),
    "p-value": round(p_val, 5),
    "Decision": decision,
    "Notes": notes
})
```

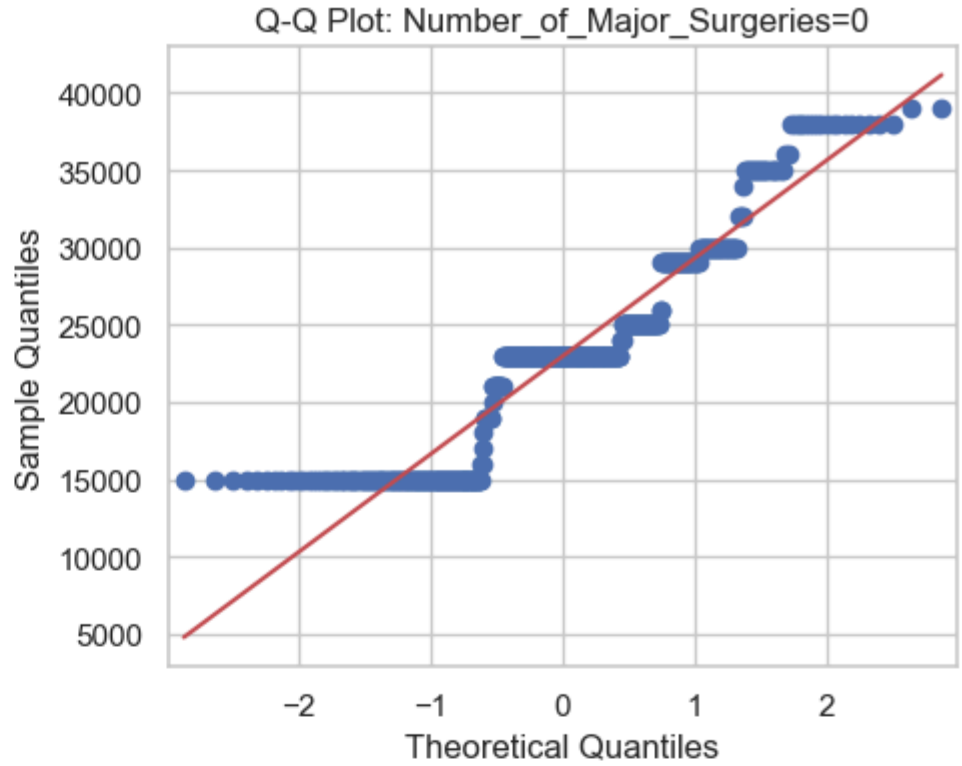
3.2.1 - Major Surgeries

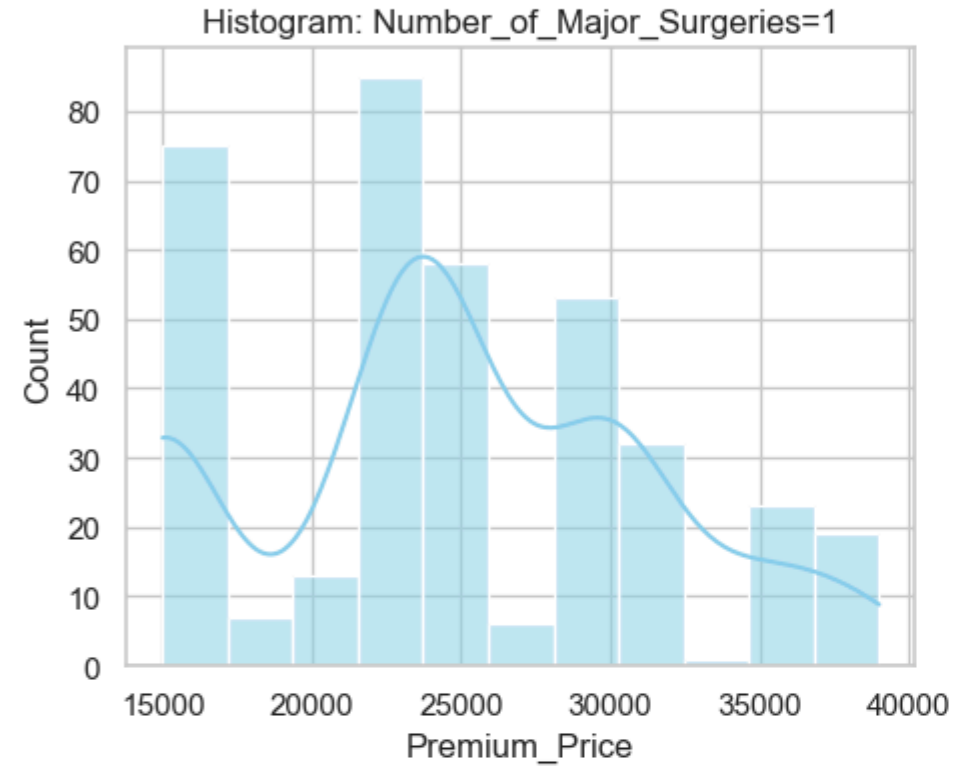
In [38]:
```
categorical_hypothesis(df, "Number_of_Major_Surgeries")
```

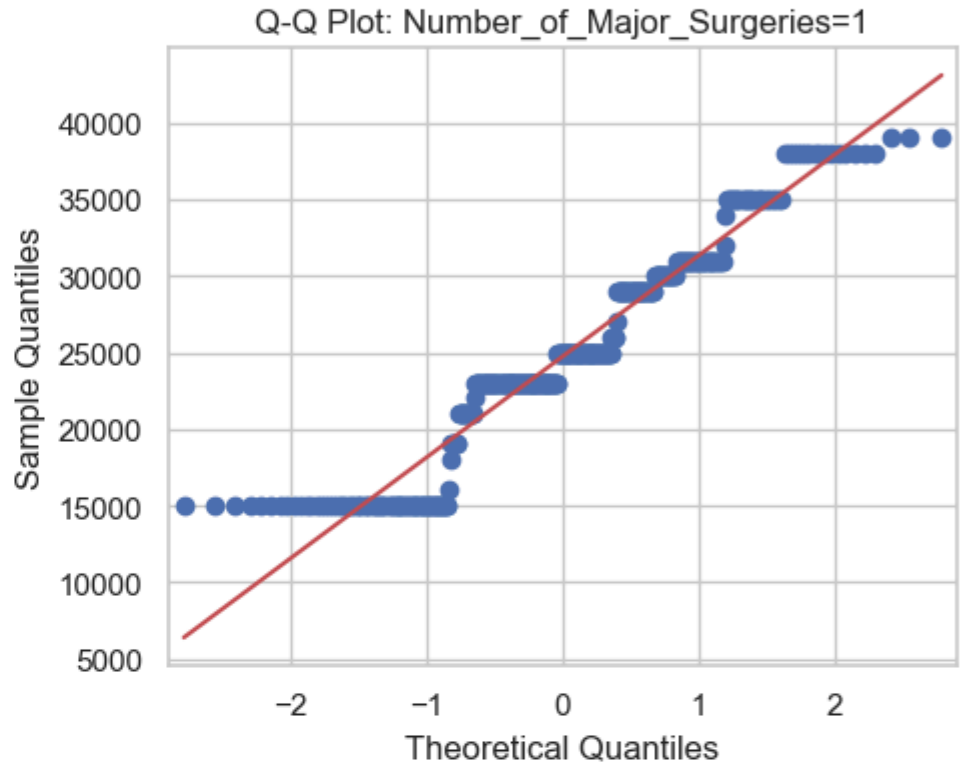
```
============================================================
Feature: Number_of_Major_Surgeries vs Premium_Price
============================================================
H0: Mean Premium_Price is equal across all groups of Number_of_Major_Surgeries
Ha: At least one group of Number_of_Major_Surgeries differs significantly in Premium_Price
```

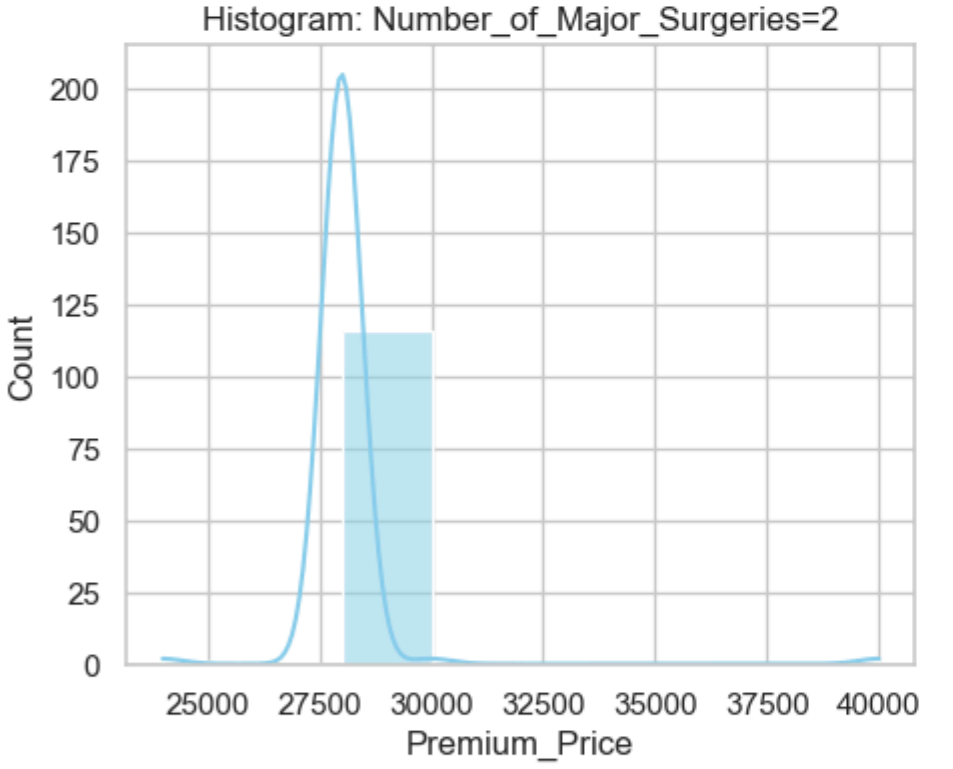

Premium_Price by Number_of_Major_Surgeries
Premium_Price
40000
35000
30000
25000
20000
15000
0
1
2
3
Number_of_Major_Surgeries

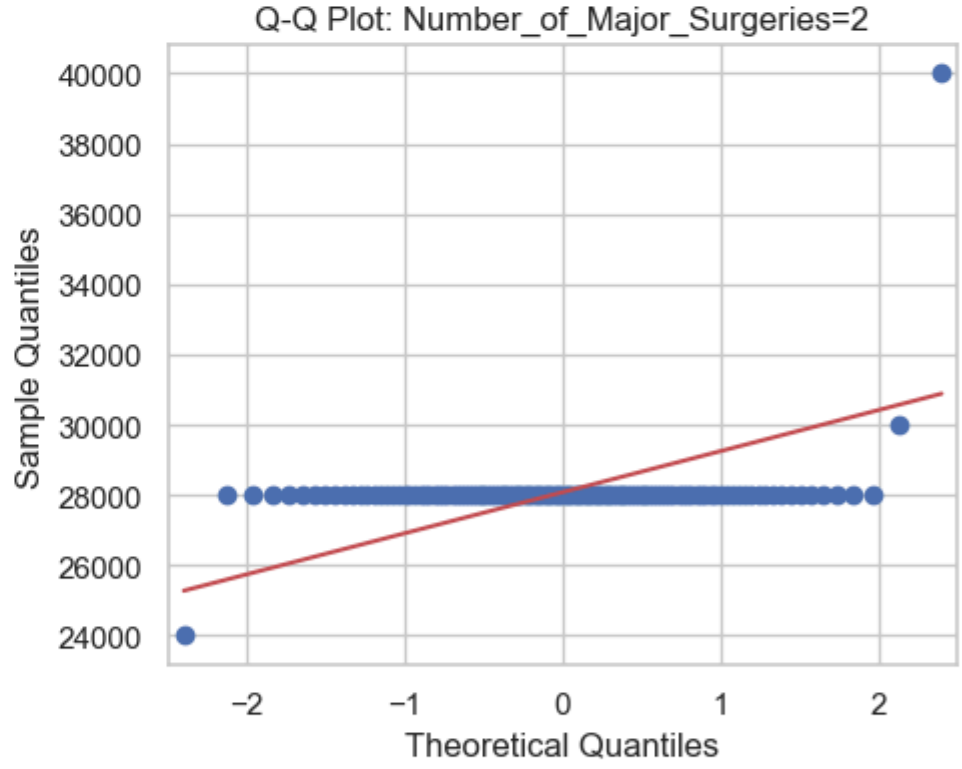

Histogram: Number_of_Major_Surgeries=0
Count
Premium_Price
0
20
40
60
80
100
120
140
160
15000
20000
25000
30000
35000
40000

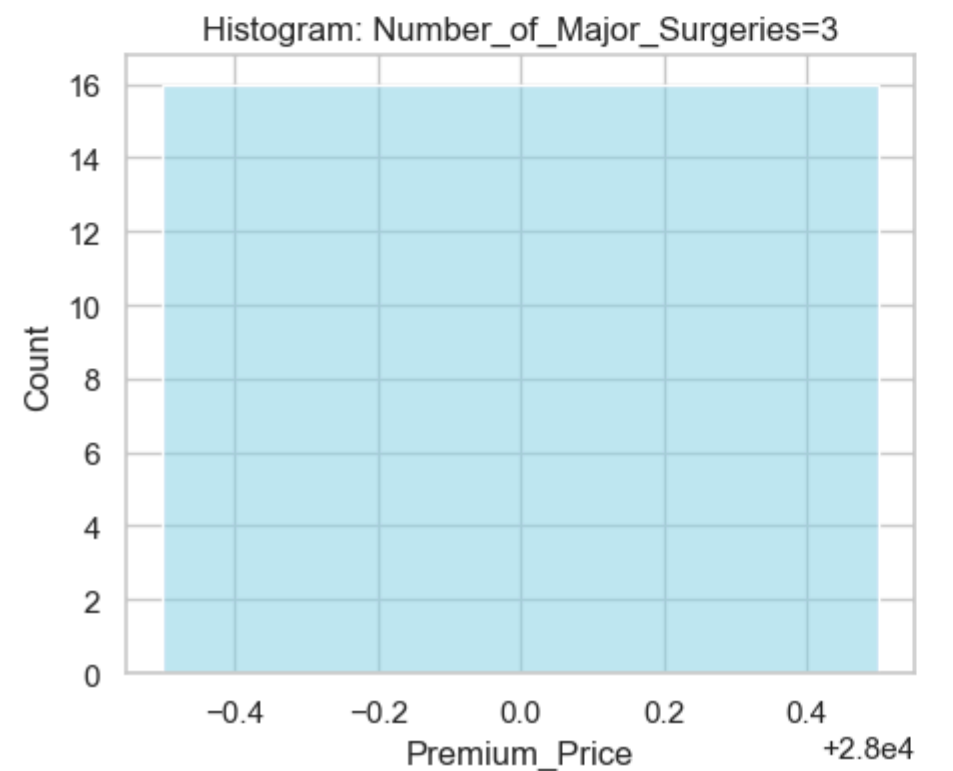

Q-Q Plot: Number_of_Major_Surgeries=0
Sample Quantiles
Theoretical Quantiles
5000
10000
15000
20000
25000
30000
35000
40000
−2
−1
0
1
2

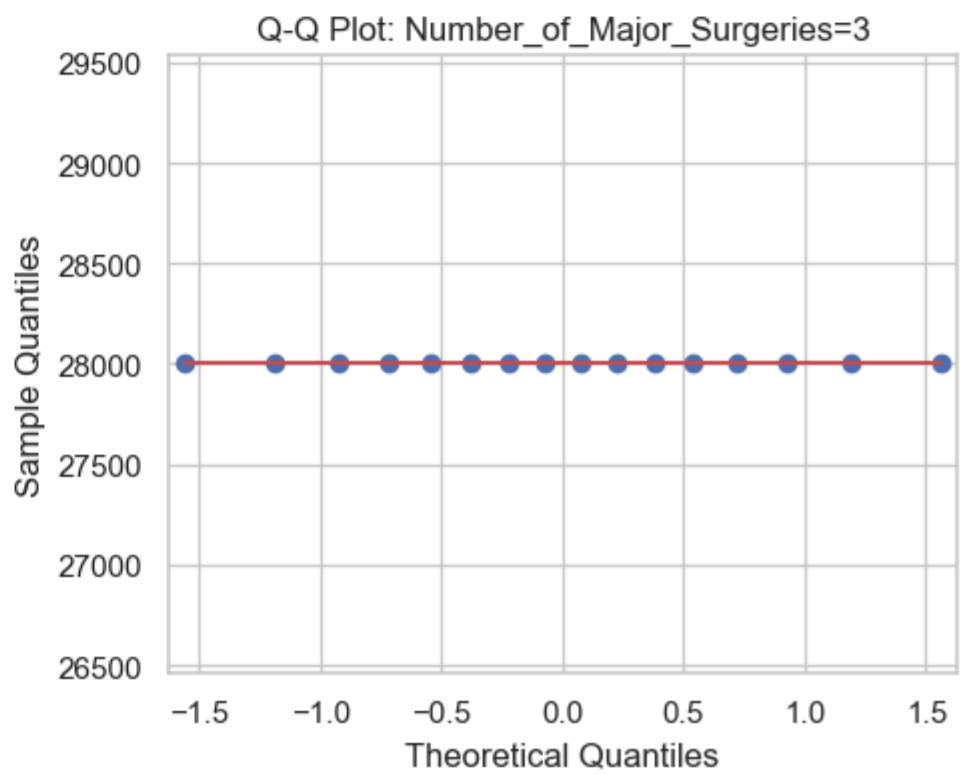

Histogram: Number_of_Major_Surgeries=1
Count
Premium_Price
0
10
20
30
40
50
60
70
80
15000
20000
25000
30000
35000
40000


Q-Q Plot: Number_of_Major_Surgeries=1
Sample Quantiles
Theoretical Quantiles
5000
10000
15000
20000
25000
30000
35000
40000
−2
−1
0
1
2


Histogram: Number_of_Major_Surgeries=2
Count
Premium_Price
0
25
50
75
100
125
150
175
200
25000
27500
30000
32500
35000
37500
40000


Q-Q Plot: Number_of_Major_Surgeries=2
Sample Quantiles
Theoretical Quantiles
24000
26000
28000
30000
32000
34000
36000
38000
40000
−2
−1
0
1
2


Histogram: Number_of_Major_Surgeries=3
Count
Premium_Price
+2.8e4
0
2
4
6
8
10
12
14
16
−0.4
−0.2
0.0
0.2
0.4


Q-Q Plot: Number_of_Major_Surgeries=3
Sample Quantiles
Theoretical Quantiles
26500
27000
27500
28000
28500
29000
29500
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

```
Shapiro Test -> Number_of_Major_Surgeries=0: stat=0.889, p=0.00000
❌ Normality failed for this group.
Shapiro Test -> Number_of_Major_Surgeries=1: stat=0.931, p=0.00000
❌ Normality failed for this group.
Shapiro Test -> Number_of_Major_Surgeries=2: stat=0.124, p=0.00000
❌ Normality failed for this group.
Shapiro Test -> Number_of_Major_Surgeries=3: stat=1.000, p=1.00000
❌ At least one group failed normality → fallback to non-parametric test.

Levene’s Test: stat=57.302, p=0.00000
❌ Variances are unequal.

Kruskal–Wallis Test → stat=93.813, p=0.00000
✅ Reject H0 → At least one group of Number_of_Major_Surgeries has significantly different Premium_Price.
```

Insights

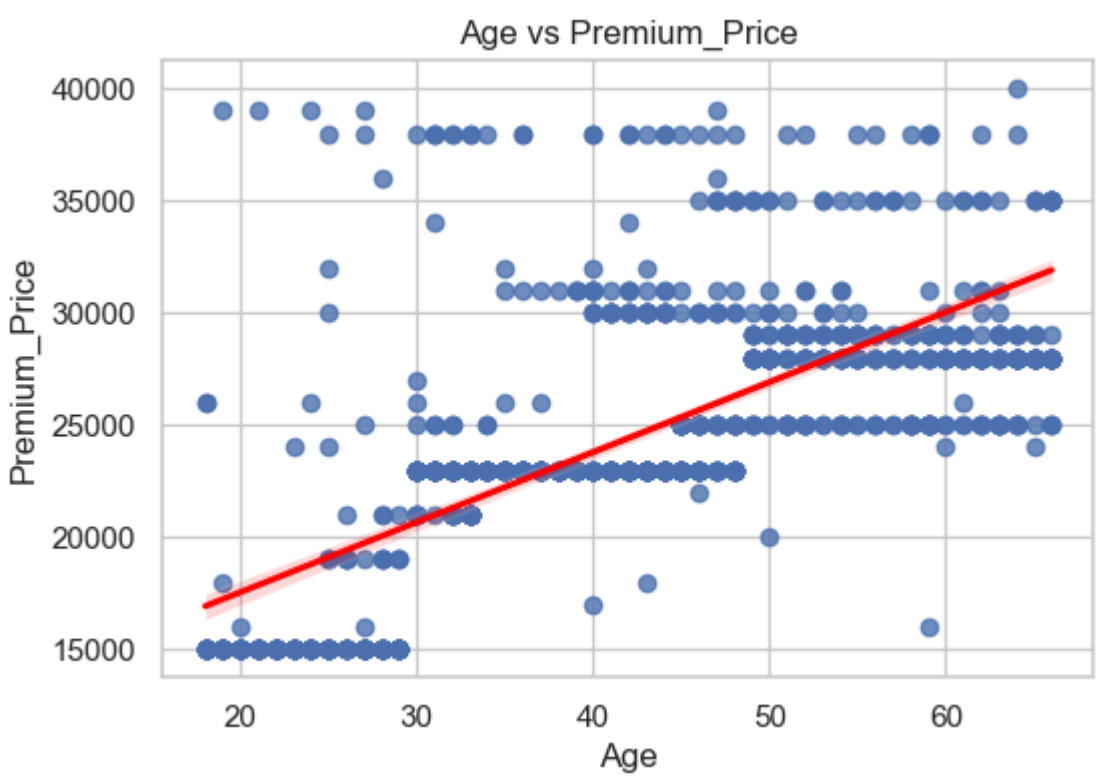
3.3 - Numeric Hypothesis Testing (Correlation / Regression)

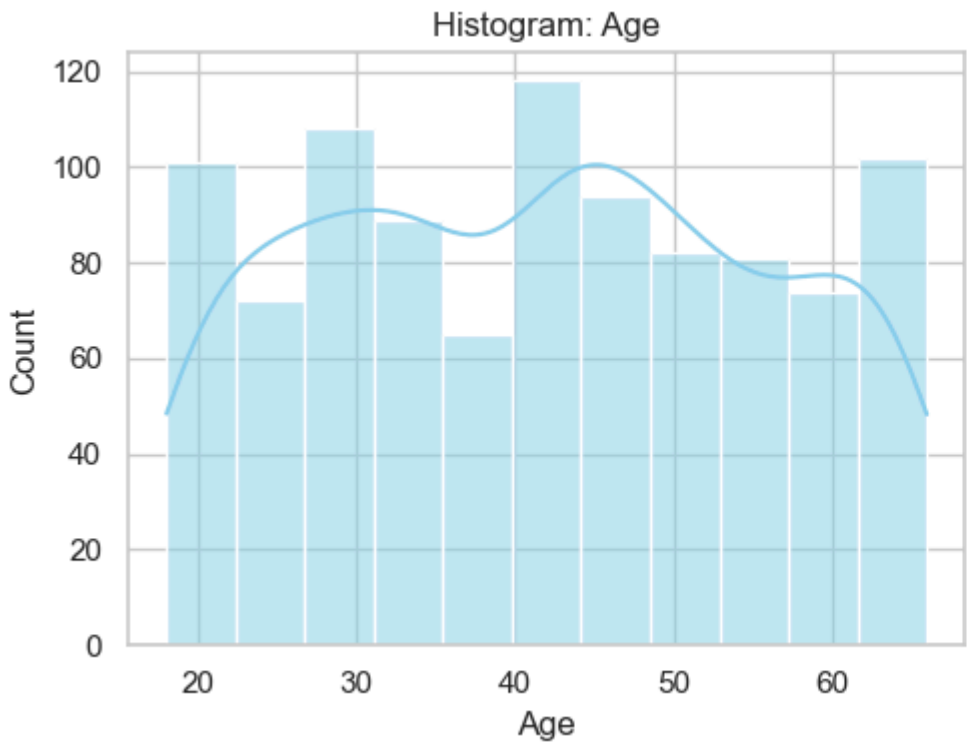
In [39]:
```python
def numeric_hypothesis(data, feature, target="Premium_Price"):
    print(f"\n{'='*60}\nFeature: {feature} vs {target}\n{'='*60}")
    print(f"H0: {feature} is NOT correlated with {target}")
    print(f"Ha: {feature} IS correlated with {target}\n")

    # --- Scatter + Regression line ---
    plt.figure(figsize=(6,4))
    sns.regplot(x=feature, y=target, data=data, line_kws={"color":"red"})
    plt.title(f"{feature} vs {target}")
    plt.show()

    # --- Hist + Q-Q Plot for feature ---
    fig, axes = plt.subplots(1, 2, figsize=(10,4))
    sns.histplot(data[feature], kde=True, ax=axes[0], color="skyblue")
    axes[0].set_title(f"Histogram: {feature}")
    sm.qqplot(data[feature], line='s', ax=axes[1])
    axes[1].set_title(f"Q-Q Plot: {feature}")
    plt.tight_layout()
    plt.show()

    # --- Shapiro Normality Test ---
    stat, p = stats.shapiro(data[feature])
    print(f"Shapiro Test -> {feature}: stat={stat:.3f}, p={p:.5f}")

    normality = p > 0.05   # Normality flag

    if normality:
        print("✅ Normality holds → Using Pearson correlation (assumes linear relationship).\n")
        corr, corr_p = stats.pearsonr(data[feature], data[target])
        test_name = "Pearson Correlation"
        stat_val = corr
        notes = "Pearson correlation used (normality passed)"
    else:
        print("❌ Normality assumption failed → Using Spearman correlation (rank-based, robust to non-normality).\n")
        corr, corr_p = stats.spearmanr(data[feature], data[target])
        test_name = "Spearman Correlation"
        stat_val = corr
        notes = "Spearman correlation used (normality failed)"

    # --- Results ---
    print(f"{test_name} → r={corr:.3f}, p={corr_p:.5f}")
    alpha = 0.05
    if corr_p < alpha:
        print(f"✅ Reject H0 → {feature} significantly correlates with {target}.")
    else:
        print(f"❌ Fail to Reject H0 → {feature} has no significant correlation with {target}.")

    # --- Decision ---
    decision = "Reject H0" if corr_p < alpha else "Fail to Reject H0"

    # --- Append to Summary Table ---
    results_summary.append({
        "Feature": feature,
        "Target": target,
        "Test": test_name,
        "Stat Value": round(stat_val, 3),
        "p-value": round(corr_p, 5),
        "Decision": decision,
        "Notes": notes
    })
```

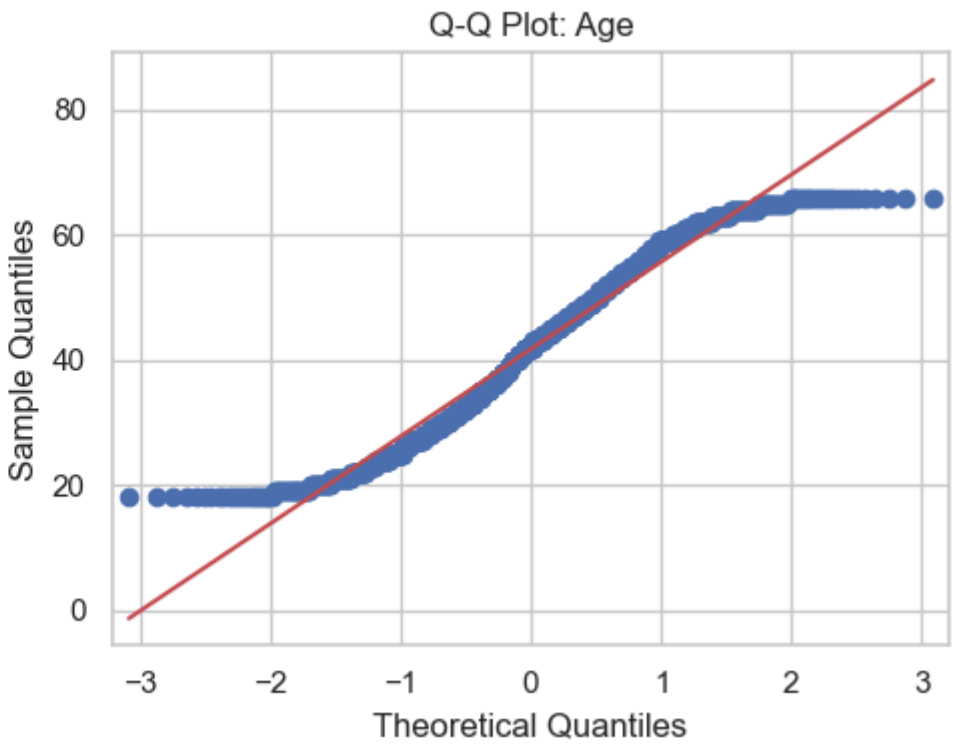
3.3.1 - Age

In [40]:
```python
numeric_hypothesis(df, "Age")
```

```
============================================================
Feature: Age vs Premium_Price
============================================================
H0: Age is NOT correlated with Premium_Price
Ha: Age IS correlated with Premium_Price
```







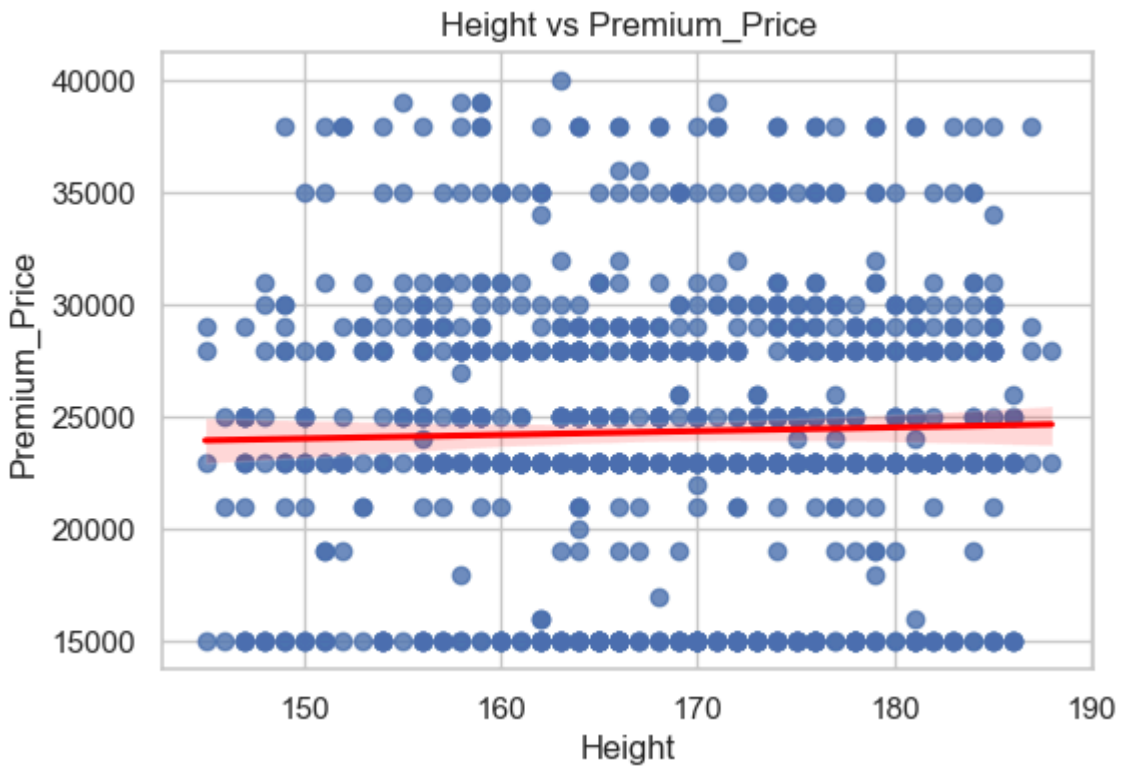

```
Shapiro Test -> Age: stat=0.959, p=0.00000
❌ Normality assumption failed → Using Spearman correlation (rank-based, robust to non-normality).

Spearman Correlation → r=0.739, p=0.00000
✅ Reject H0 → Age significantly correlates with Premium_Price.
```

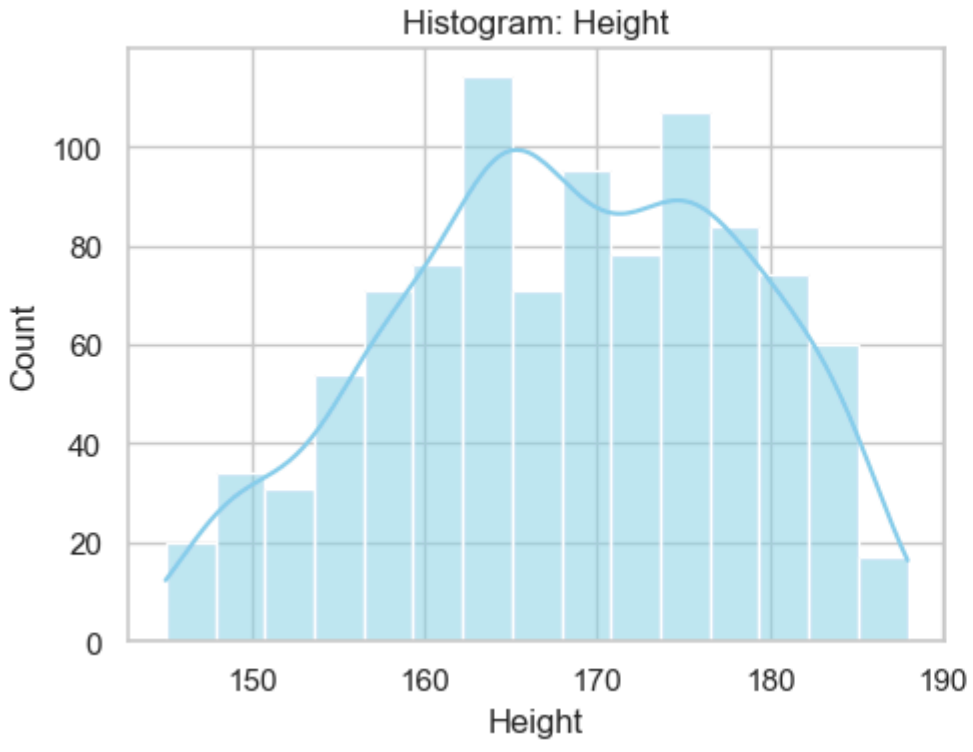
Insights

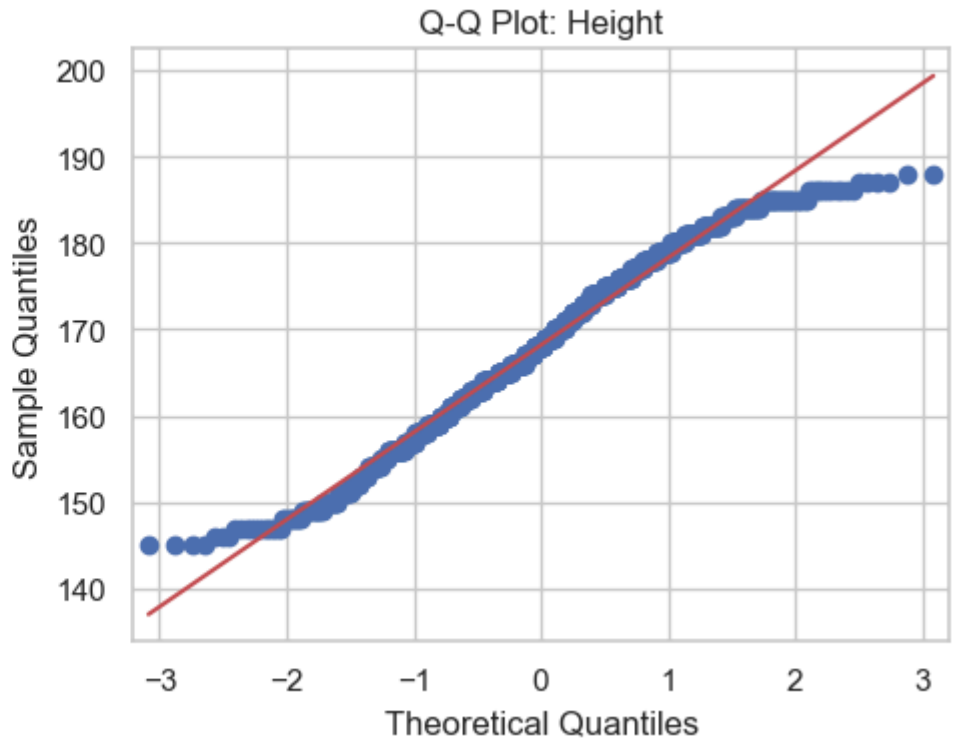
3.3.2 - Height

In [41]:
```
numeric_hypothesis(df, "Height")
```

```
============================================================
Feature: Height vs Premium_Price
============================================================
H0: Height is NOT correlated with Premium_Price
Ha: Height IS correlated with Premium_Price
```







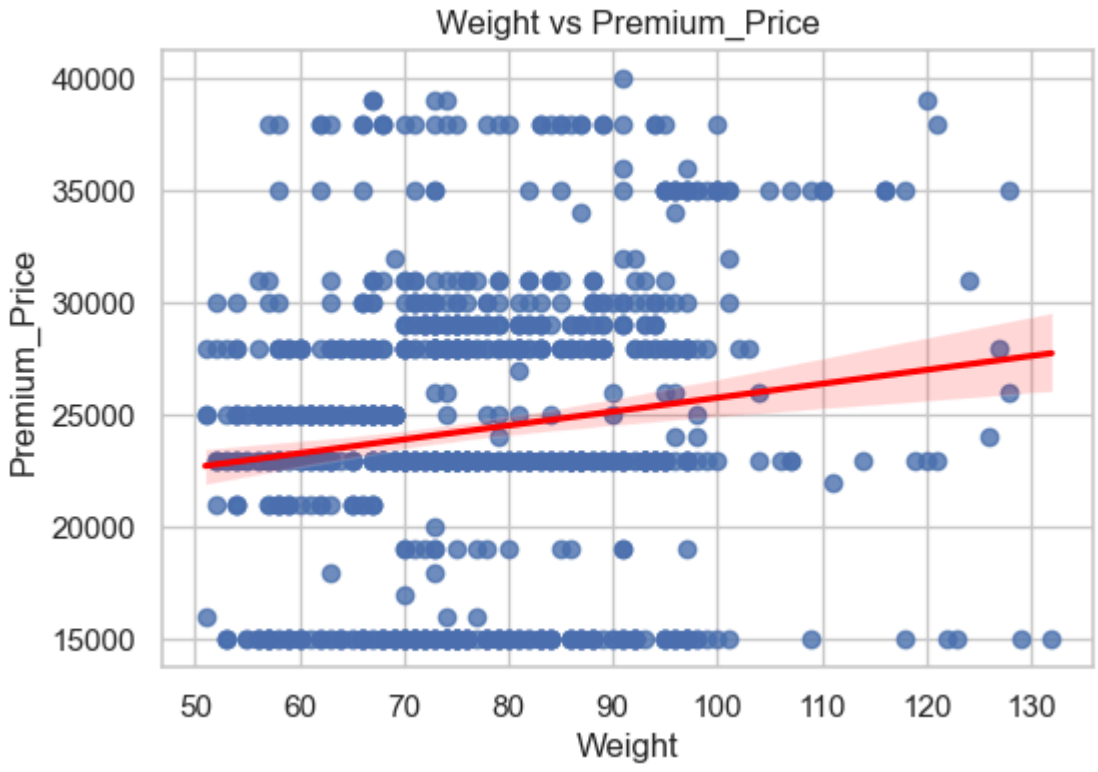

```
Shapiro Test -> Height: stat=0.980, p=0.00000
❌ Normality assumption failed → Using Spearman correlation (rank-based, robust to non-normality).

Spearman Correlation → r=0.023, p=0.46820
❌ Fail to Reject H0 → Height has no significant correlation with Premium_Price.
```

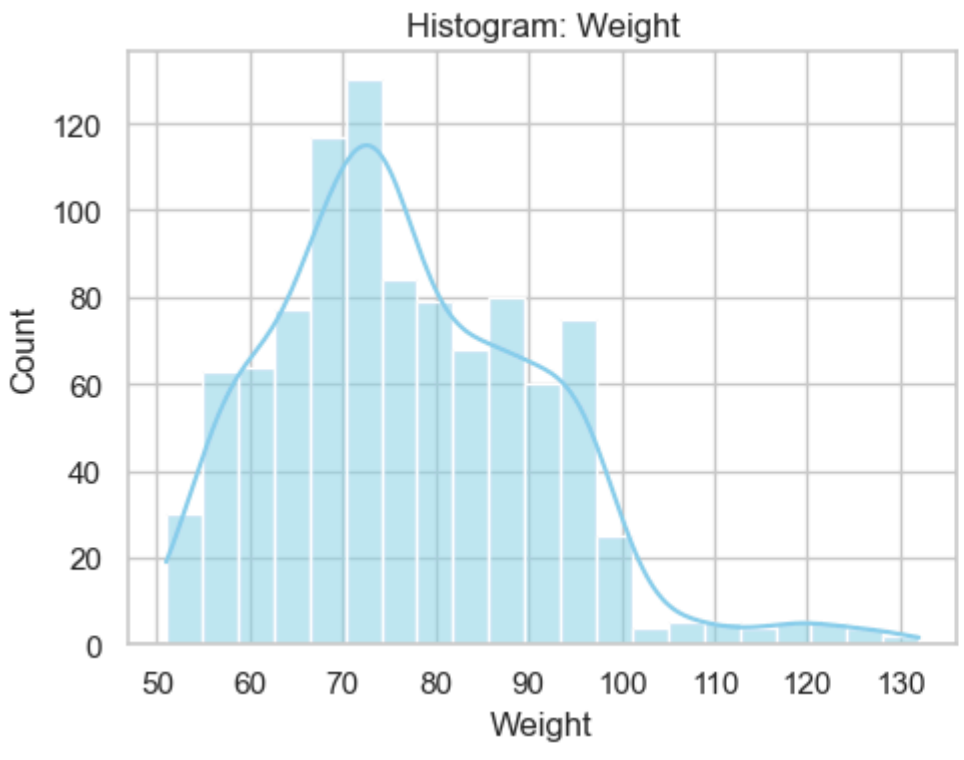
Insights

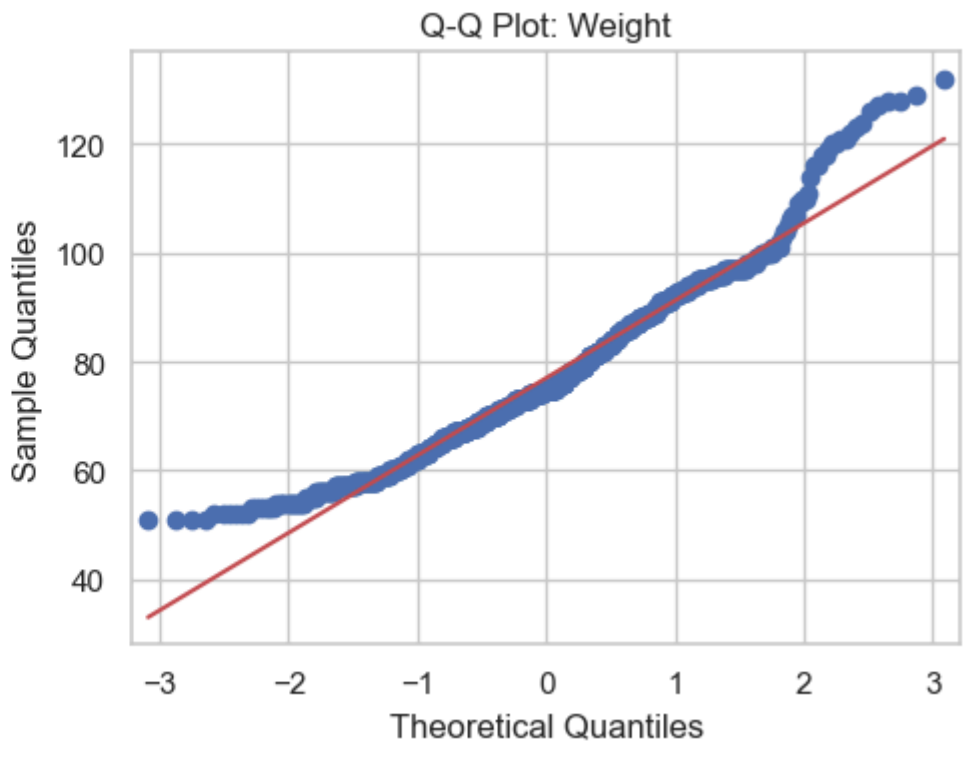
3.3.3 - Weight

In [42]:
```
numeric_hypothesis(df, "Weight")
```

```
============================================================
Feature: Weight vs Premium_Price
============================================================
H0: Weight is NOT correlated with Premium_Price
Ha: Weight IS correlated with Premium_Price
```







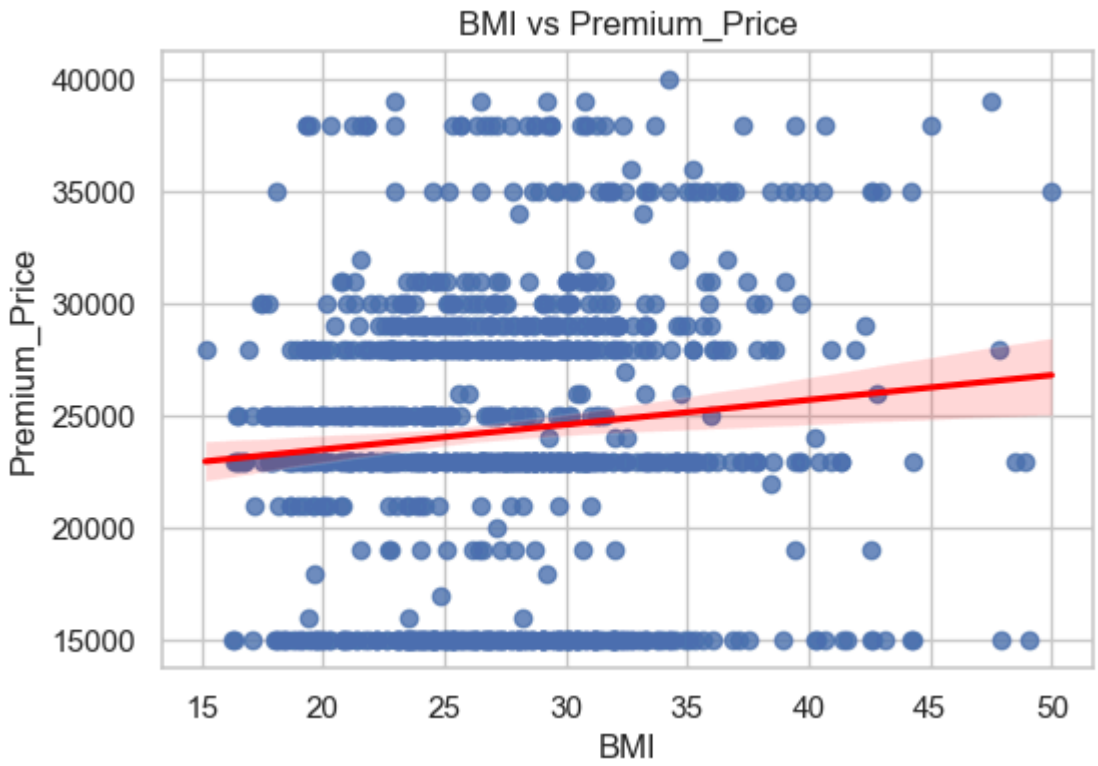
```
Shapiro Test -> Weight: stat=0.967, p=0.00000
❌ Normality assumption failed → Using Spearman correlation (rank-based, robust to non-normality).

Spearman Correlation → r=0.129, p=0.00005
✅ Reject H0 → Weight significantly correlates with Premium_Price.
```

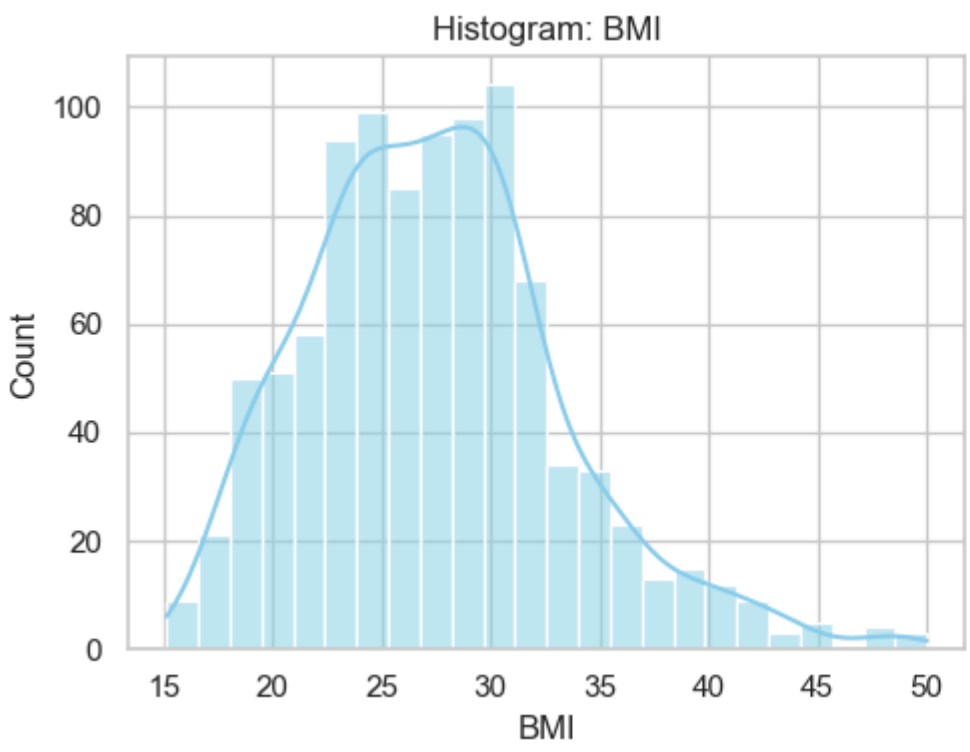
Insights

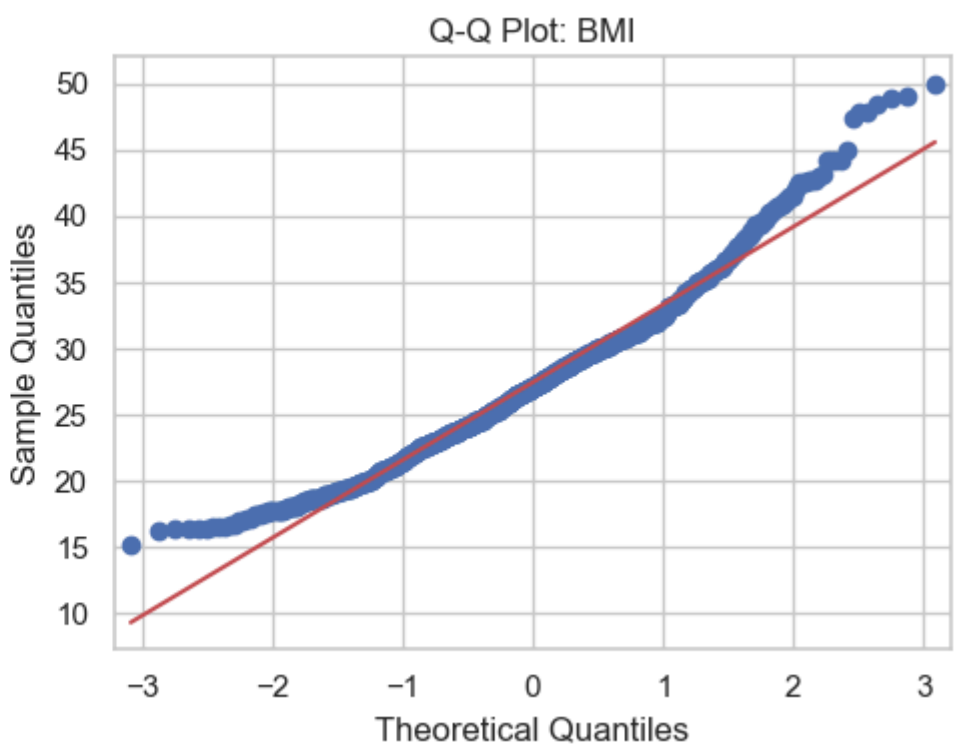
3.3.4 - BMI

In [43]:
```
numeric_hypothesis(df, "BMI")
```

```
============================================================
Feature: BMI vs Premium_Price
============================================================
H0: BMI is NOT correlated with Premium_Price
Ha: BMI IS correlated with Premium_Price
```








```
Shapiro Test -> BMI: stat=0.973, p=0.00000
❌ Normality assumption failed → Using Spearman correlation (rank-based, robust to non-normality).

Spearman Correlation → r=0.098, p=0.00209
✅ Reject H0 → BMI significantly correlates with Premium_Price.
```

Insights

3.4 - Chi - Square

In [44]:
```
cat_vars = [
    "Diabetes", "Blood_Pressure_Problems", "Any_Transplants",
    "Any_Chronic_Diseases", "Known_Allergies", "History_of_Cancer_in_Family",
    "Number_of_Major_Surgeries"
]
```

In [45]:
```
# Chi-Square Test for Independence

chi_pvals = []
chi_labels = []

for i in cat_vars:
    row = []
    label_row = []
    for j in cat_vars:
```

```
        if i == j:
            row.append(np.nan)
            label_row.append("—")
        else:
            table = pd.crosstab(df[i], df[j])
            chi2, p, dof, exp = chi2_contingency(table)
            row.append(round(p, 3))
            label_row.append("Dependent" if p < 0.05 else "Independent")
    chi_pvals.append(row)
    chi_labels.append(label_row)

chi_pvals_df = pd.DataFrame(chi_pvals, index=cat_vars, columns=cat_vars)
chi_labels_df = pd.DataFrame(chi_labels, index=cat_vars, columns=cat_vars)
```

In [46]:
```
# Display the results

print("🔹 P-values from Chi-Square Tests:")
display(chi_pvals_df)

print("\n🔹 Dependence Results (p < 0.05 means Dependent):")
display(chi_labels_df)
```

🔹 P-values from Chi-Square Tests:

	Diabetes	Blood_Pressure_Problems	Any_Transplants	Any_Chronic_Diseases	Known_Allergies	History_of_Cancer_in_Family	Nun
Diabetes	NaN	0.000	0.312	0.006	0.015	0.100	
Blood_Pressure_Problems	0.000	NaN	0.528	0.179	0.776	0.157	
Any_Transplants	0.312	0.528	NaN	0.354	1.000	0.676	
Any_Chronic_Diseases	0.006	0.179	0.354	NaN	0.447	0.886	
Known_Allergies	0.015	0.776	1.000	0.447	NaN	0.000	
History_of_Cancer_in_Family	0.100	0.157	0.676	0.886	0.000	NaN	
Number_of_Major_Surgeries	0.000	0.000	0.868	0.106	0.000	0.000	

🔹 Dependence Results (p < 0.05 means Dependent):

	Diabetes	Blood_Pressure_Problems	Any_Transplants	Any_Chronic_Diseases	Known_Allergies	History_of_Cancer_in_Family	N
Diabetes	—	Dependent	Independent	Dependent	Dependent	Independent	
Blood_Pressure_Problems	Dependent	—	Independent	Independent	Independent	Independent	
Any_Transplants	Independent	Independent	—	Independent	Independent	Independent	
Any_Chronic_Diseases	Dependent	Independent	Independent	—	Independent	Independent	
Known_Allergies	Dependent	Independent	Independent	Independent	—	Dependent	
History_of_Cancer_in_Family	Independent	Independent	Independent	Independent	Dependent	—	
Number_of_Major_Surgeries	Dependent	Dependent	Independent	Independent	Dependent	Dependent	

Insights

3.5 - Summary

In [47]:
```
# Convert results summary to DataFrame for better visualization

results_summary_df = pd.DataFrame(results_summary)
results_summary_df
```

Out[47]:

	Feature	Target	Test	Stat Value	p-value	Decision	Notes
0	Diabetes	Premium_Price	Mann–Whitney U Test	106563.500	0.00648	Reject H0	Mann–Whitney U Test used (normality failed)
1	Blood_Pressure_Problems	Premium_Price	Mann–Whitney U Test	96697.000	0.00000	Reject H0	Mann–Whitney U Test used (normality failed)
2	Any_Transplants	Premium_Price	Mann–Whitney U Test	11814.000	0.00000	Reject H0	Mann–Whitney U Test used (normality failed)
3	Any_Chronic_Diseases	Premium_Price	Mann–Whitney U Test	49243.500	0.00000	Reject H0	Mann–Whitney U Test used (normality failed)
4	Known_Allergies	Premium_Price	Mann–Whitney U Test	79964.500	0.56578	Fail to Reject H0	Mann–Whitney U Test used (normality failed)
5	History_of_Cancer_in_Family	Premium_Price	Mann–Whitney U Test	39412.500	0.00010	Reject H0	Mann–Whitney U Test used (normality failed)
6	Number_of_Major_Surgeries	Premium_Price	Kruskal–Wallis Test	93.813	0.00000	Reject H0	Kruskal–Wallis used (normality failed)
7	Age	Premium_Price	Spearman Correlation	0.739	0.00000	Reject H0	Spearman correlation used (normality failed)
8	Height	Premium_Price	Spearman Correlation	0.023	0.46820	Fail to Reject H0	Spearman correlation used (normality failed)
9	Weight	Premium_Price	Spearman Correlation	0.129	0.00005	Reject H0	Spearman correlation used (normality failed)
10	BMI	Premium_Price	Spearman Correlation	0.098	0.00209	Reject H0	Spearman correlation used (normality failed)

4. Modeling

4.1 - Data Preparation

In [48]:
```
# --- Copy of data ---
df_model = df.copy()
```

In [49]:
```
df_model
```

Out[49]:

	Age	Diabetes	Blood_Pressure_Problems	Any_Transplants	Any_Chronic_Diseases	Height	Weight	Known_Allergies	History_of_Cancer_in_Family	Nu
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	
...	...	...	...	...	...	...	...	...	...	...
981	18	0	0	0	0	169	67	0	0	
982	64	1	1	0	0	153	70	0	0	
983	56	0	1	0	0	155	71	0	0	
984	47	1	1	0	0	158	73	1	0	
985	21	0	0	0	0	158	75	1	0	

986 rows × 14 columns

In [50]:
```
# --- Features (X) and Target (y) ---

target = "Premium_Price"

X = df_model.drop(target, axis=1)
y = df_model[target]
```

In [51]:
```
# --- Identify categorical & continuous features ---
cat_features = ["BMI_Category", "Age_Group"]
cont_features = ["Age", "Height", "Weight", "BMI"]
```

In [52]:
```
# --- One-hot encode categorical features ---
X = pd.get_dummies(X, columns=cat_features, drop_first=True)
```

In [53]:
```
# --- Train-test split ---

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Convert only bool columns to int
for col in X_train.select_dtypes(include=["bool"]).columns:
    X_train[col] = X_train[col].astype(int)
    X_test[col] = X_test[col].astype(int)



# --- Scale continuous features ---

scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[cont_features] = scaler.fit_transform(X_train[cont_features])
X_test_scaled[cont_features] = scaler.transform(X_test[cont_features])


print("✅ Data Preprocessing Done")
print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

```
✅ Data Preprocessing Done
Train shape: (788, 18)
Test shape: (198, 18)
```

In [140]:
```
X_train.columns
```

Out[140]:
```
Index(['Age', 'Diabetes', 'Blood_Pressure_Problems', 'Any_Transplants',
       'Any_Chronic_Diseases', 'Height', 'Weight', 'Known_Allergies',
       'History_of_Cancer_in_Family', 'Number_of_Major_Surgeries', 'BMI',
       'BMI_Category_Normal', 'BMI_Category_Overweight', 'BMI_Category_Obese',
       'Age_Group_30-39', 'Age_Group_40-49', 'Age_Group_50-59',
       'Age_Group_60+'],
      dtype='object')
```

In [54]:
```
df_model
```

Out[54]:

	Age	Diabetes	Blood_Pressure_Problems	Any_Transplants	Any_Chronic_Diseases	Height	Weight	Known_Allergies	History_of_Cancer_in_Family	Nu
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	
...	...	...	...	...	...	...	...	...	...	
981	18	0	0	0	0	169	67	0	0	
982	64	1	1	0	0	153	70	0	0	
983	56	0	1	0	0	155	71	0	0	
984	47	1	1	0	0	158	73	1	0	
985	21	0	0	0	0	158	75	1	0	

986 rows × 14 columns

In [55]:
```
def check_splits(X_train, X_train_scaled, y_train, X_test, X_test_scaled, y_test, n=5):
    print("\n🔎 Data Split Check")
    print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
    print(f"X_test  shape: {X_test.shape}, y_test shape: {y_test.shape}")
    print(f"X_train_scaled shape: {X_train_scaled.shape}")
    print(f"X_test_scaled shape: {X_test_scaled.shape}")

    print("\n📌 First few rows of X_train (raw encoded):")
    display(X_train.head(n))

    print("\n📌 First few rows of X_train_scaled (continuous vars scaled):")
    display(X_train_scaled[["Age","Height","Weight","BMI"]].head(n))

    print("\n🎯 Target Preview (y_train):")
    display(y_train.head(n))

# ✅ Call after preprocessing
check_splits(X_train, X_train_scaled, y_train, X_test, X_test_scaled, y_test)
```

```
🔎 Data Split Check
X_train shape: (788, 18), y_train shape: (788,)
X_test  shape: (198, 18), y_test shape: (198,)
X_train_scaled shape: (788, 18)
X_test_scaled shape: (198, 18)

📌 First few rows of X_train (raw encoded):
```

	Age	Diabetes	Blood_Pressure_Problems	Any_Transplants	Any_Chronic_Diseases	Height	Weight	Known_Allergies	History_of_Cancer_in_Family	Num
762	19	0	0	0	0	146	55	0	0	
334	31	0	1	0	0	162	87	0	0	
890	58	1	0	0	0	147	75	0	0	
529	31	0	1	0	0	171	85	0	0	
468	26	1	1	0	0	167	70	0	1	

```
📌 First few rows of X_train_scaled (continuous vars scaled):
```

	Age	Height	Weight	BMI
762	-1.617536	-2.219275	-1.519089	-0.279682
334	-0.763949	-0.620930	0.691260	0.956170
890	1.156620	-2.119378	-0.137621	1.218088
529	-0.763949	0.278138	0.553113	0.269701
468	-1.119610	-0.121448	-0.482988	-0.397867

```
🎯 Target Preview (y_train):
762    15000
334    34000
890    29000
529    23000
468    15000
Name: Premium_Price, dtype: int64
```

4.2 - Linear Regression Model

In [56]:
```
X_train_scaled.dtypes
```

Out[56]:
```
Age                           float64
Diabetes                        int64
Blood_Pressure_Problems         int64
Any_Transplants                 int64
Any_Chronic_Diseases            int64
Height                        float64
Weight                        float64
Known_Allergies                 int64
History_of_Cancer_in_Family     int64
Number_of_Major_Surgeries       int64
BMI                           float64
BMI_Category_Normal             int64
BMI_Category_Overweight         int64
BMI_Category_Obese              int64
Age_Group_30-39                 int64
Age_Group_40-49                 int64
Age_Group_50-59                 int64
Age_Group_60+                   int64
dtype: object
```

In [57]:
```python
# --- Train Linear Regression on Scaled Data ---
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)

# --- Predictions ---
y_train_pred = lin_reg.predict(X_train_scaled)
y_test_pred = lin_reg.predict(X_test_scaled)

# --- Evaluation Function ---
def evaluate_model(name, y_true_train, y_pred_train, y_true_test, y_pred_test):
    metrics = {}
    metrics["Model"] = name
    metrics["Train RMSE"] = np.sqrt(mean_squared_error(y_true_train, y_pred_train))
    metrics["Test RMSE"] = np.sqrt(mean_squared_error(y_true_test, y_pred_test))
    metrics["Test MAE"] = mean_absolute_error(y_true_test, y_pred_test)
    metrics["Test R²"] = r2_score(y_true_test, y_pred_test)

    print(f"\n{name} Performance:")
    for k,v in metrics.items():
        if k != "Model":
            print(f"{k}: {v:.3f}")
    return metrics

# --- Store results ---
results = []
results.append(
    evaluate_model("Linear Regression", y_train, y_train_pred, y_test, y_test_pred)
)

# --- Coefficients (Feature Importance for Linear Model) ---
coef_df = pd.DataFrame({
    "Feature": X_train_scaled.columns,
    "Coefficient": lin_reg.coef_
}).sort_values(by="Coefficient", ascending=False)

print("\nTop Features Driving Premium Price (Linear Regression):")
print(coef_df.head(10))
```

```
Linear Regression Performance:
Train RMSE: 3381.950
Test RMSE: 3020.591
Test MAE: 2091.987
Test R²: 0.786

Top Features Driving Premium Price (Linear Regression):
                        Feature  Coefficient
16              Age_Group_50-59  7894.232735
3               Any_Transplants  7474.035658
17                Age_Group_60+  7143.039433
15              Age_Group_40-49  7034.755970
14              Age_Group_30-39  5772.278873
4           Any_Chronic_Diseases  2155.951674
0                            Age  1870.083012
8   History_of_Cancer_in_Family  1868.143282
13           BMI_Category_Obese  1861.626654
6                        Weight  1322.094472
```

In [58]:
```python
coef_df.head(20)
```

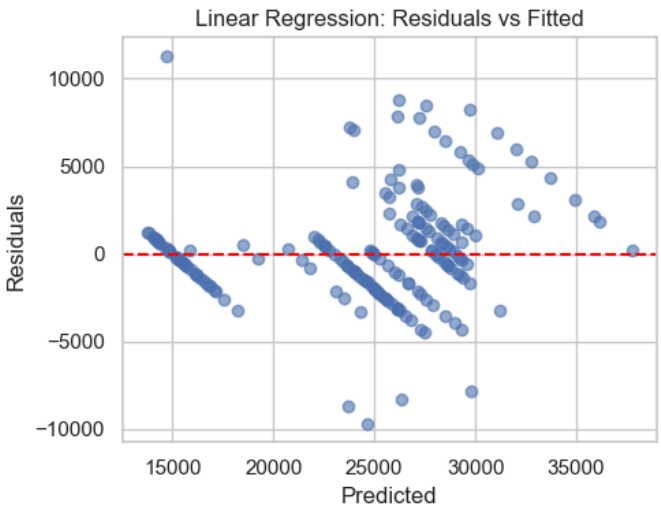
Out[58]:

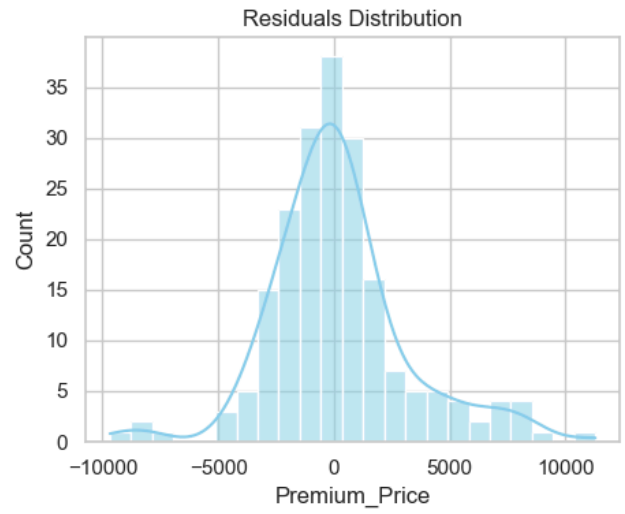
	Feature	Coefficient
16	Age_Group_50-59	7894.232735
3	Any_Transplants	7474.035658
17	Age_Group_60+	7143.039433
15	Age_Group_40-49	7034.755970
14	Age_Group_30-39	5772.278873
4	Any_Chronic_Diseases	2155.951674
0	Age	1870.083012
8	History_of_Cancer_in_Family	1868.143282
13	BMI_Category_Obese	1861.626654
6	Weight	1322.094472
12	BMI_Category_Overweight	739.449979
11	BMI_Category_Normal	209.342553
2	Blood_Pressure_Problems	181.000035
7	Known_Allergies	-36.997485
5	Height	-159.974116
9	Number_of_Major_Surgeries	-219.020388
1	Diabetes	-322.069029
10	BMI	-929.858888

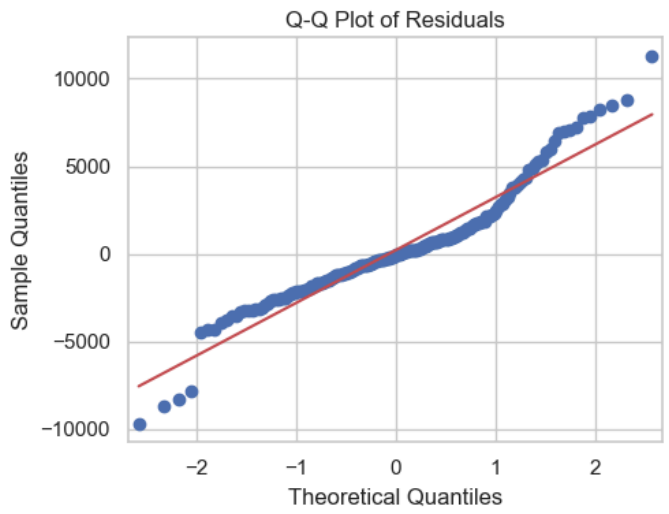
In [59]:
```python
# --- Residual Analysis ---
def plot_residuals(y_true, y_pred, model_name="Linear Regression"):

    residuals = y_true - y_pred

    fig, axes = plt.subplots(1, 3, figsize=(15,4))

    # Residuals vs Fitted
    axes[0].scatter(y_pred, residuals, alpha=0.6)
    axes[0].axhline(0, color='red', linestyle='--')
    axes[0].set_title(f"{model_name}: Residuals vs Fitted")
    axes[0].set_xlabel("Predicted")
    axes[0].set_ylabel("Residuals")

    # Histogram of residuals
    sns.histplot(residuals, kde=True, ax=axes[1], color="skyblue")
    axes[1].set_title("Residuals Distribution")

    # Q-Q Plot
    sm.qqplot(residuals, line='s', ax=axes[2])
    axes[2].set_title("Q-Q Plot of Residuals")

    plt.tight_layout()
    plt.show()


# --- Cross Validation ---
def cross_validate_model(model, X, y, cv=5):
    kf = KFold(n_splits=cv, shuffle=True, random_state=42)
    scores = cross_val_score(model, X, y, scoring="r2", cv=kf)
    rmse_scores = np.sqrt(-cross_val_score(model, X, y, scoring="neg_mean_squared_error", cv=kf))

    print(f"\nCross-Validation ({cv}-fold):")
    print(f"Mean R²: {scores.mean():.3f} | Std: {scores.std():.3f}")
    print(f"Mean RMSE: {rmse_scores.mean():.3f} | Std: {rmse_scores.std():.3f}")


# --- Statsmodels for Coefficients ---
def regression_summary(X, y):
    X_const = sm.add_constant(X)   # add intercept
    model = sm.OLS(y, X_const).fit()
    display(model.summary())
    return model
```

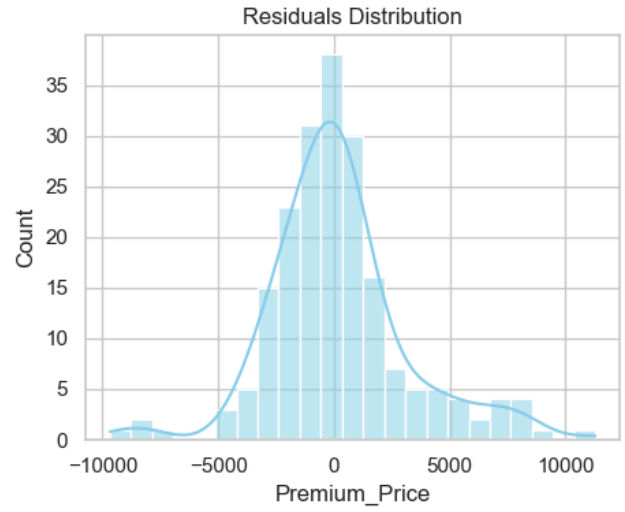
In [60]:
```python
# Residual Analysis
plot_residuals(y_test, y_test_pred, model_name="Linear Regression")



# Residual analysis - test set
plot_residuals(y_test, y_test_pred, model_name="Linear Regression - Test")


# Residual analysis - train set
plot_residuals(y_train, y_train_pred, model_name="Linear Regression - Train")

# Cross Validation
```

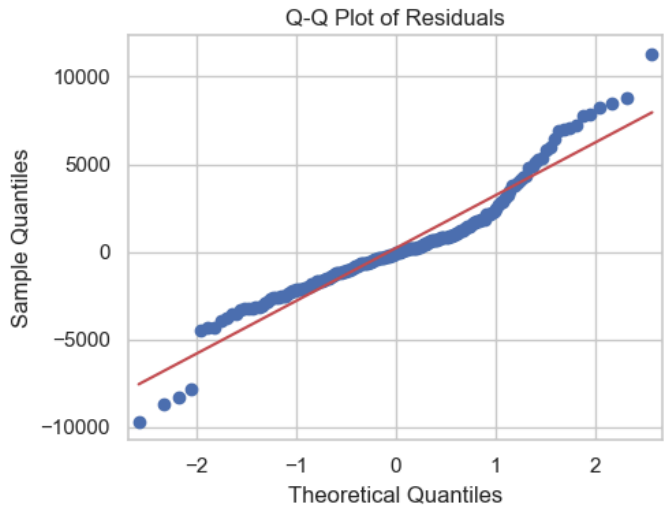
```
cross_validate_model(lin_reg, X, y, cv=5)

# Regression Coefficients / p-values
regression_summary(X_train_scaled, y_train)
```

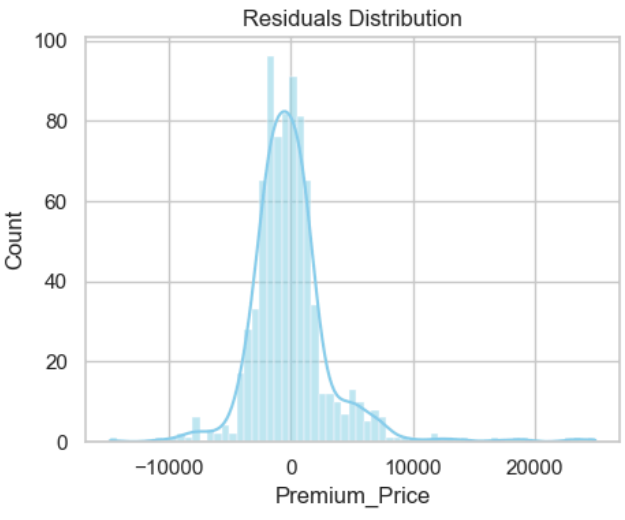


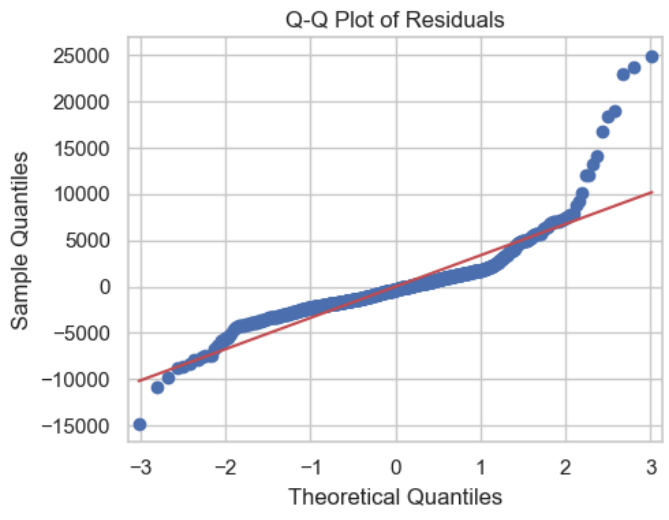













```
Cross-Validation (5-fold):
Mean R²: 0.695 | Std: 0.092
Mean RMSE: 3386.516 | Std: 436.970
```

OLS Regression Results

Dep. Variable:	Premium_Price	**R-squared:**	0.700
Model:	OLS	**Adj. R-squared:**	0.692
Method:	Least Squares	**F-statistic:**	99.45
Date:	Fri, 22 Aug 2025	**Prob (F-statistic):**	1.28e-186
Time:	00:41:32	**Log-Likelihood:**	-7521.6
No. Observations:	788	**AIC:**	1.508e+04
Df Residuals:	769	**BIC:**	1.517e+04
Df Model:	18		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.734e+04	1140.299	15.208	0.000	1.51e+04	1.96e+04
Age	1870.0830	602.316	3.105	0.002	687.705	3052.461
Diabetes	-322.0690	256.674	-1.255	0.210	-825.935	181.797
Blood_Pressure_Problems	181.0000	259.789	0.697	0.486	-328.979	690.979
Any_Transplants	7474.0357	534.556	13.982	0.000	6424.674	8523.397
Any_Chronic_Diseases	2155.9517	322.591	6.683	0.000	1522.688	2789.215
Height	-159.9741	743.985	-0.215	0.830	-1620.456	1300.508
Weight	1322.0945	1141.259	1.158	0.247	-918.259	3562.448
Known_Allergies	-36.9975	304.285	-0.122	0.903	-634.325	560.330
History_of_Cancer_in_Family	1868.1433	401.785	4.650	0.000	1079.418	2656.868
Number_of_Major_Surgeries	-219.0204	192.290	-1.139	0.255	-596.496	158.455
BMI	-929.8589	1271.471	-0.731	0.465	-3425.824	1566.107
BMI_Category_Normal	209.3426	724.406	0.289	0.773	-1212.705	1631.390
BMI_Category_Overweight	739.4500	893.305	0.828	0.408	-1014.155	2493.055
BMI_Category_Obese	1861.6267	1077.177	1.728	0.084	-252.929	3976.182
Age_Group_30-39	5772.2789	583.462	9.893	0.000	4626.912	6917.645
Age_Group_40-49	7034.7560	958.722	7.338	0.000	5152.734	8916.778
Age_Group_50-59	7894.2327	1367.591	5.772	0.000	5209.578	1.06e+04
Age_Group_60+	7143.0394	1737.466	4.111	0.000	3732.301	1.06e+04

Omnibus:	435.148	**Durbin-Watson:**	1.877
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6195.697
Skew:	2.174	**Prob(JB):**	0.00
Kurtosis:	16.030	**Cond. No.**	35.4

Notes:

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

Out[60]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x15c6af2df60>

In [61]:
```
# Calculate VIF for each feature
def calculate_vif(X):
    vif_data = pd.DataFrame()
    vif_data["Feature"] = X.columns
    vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                       for i in range(len(X.columns))]
    return vif_data

# Assuming X_train_scaled is a DataFrame
vif_df = calculate_vif(pd.DataFrame(X_train_scaled, columns=X_train.columns))
print(vif_df)
```

	Feature	VIF
0	Age	13.778605
1	Diabetes	1.856877
2	Blood_Pressure_Problems	2.089455
3	Any_Transplants	1.072316
4	Any_Chronic_Diseases	1.266204
5	Height	33.073930
6	Weight	78.157471
7	Known_Allergies	1.342614
8	History_of_Cancer_in_Family	1.238977
9	Number_of_Major_Surgeries	2.559603
10	BMI	101.872607
11	BMI_Category_Normal	6.481639
12	BMI_Category_Overweight	10.074540
13	BMI_Category_Obese	14.296498
14	Age_Group_30-39	2.990498
15	Age_Group_40-49	7.901772
16	Age_Group_50-59	13.076646
17	Age_Group_60+	16.333871

In [62]:
```
# Predictions with Confidence Intervals using Statsmodels

# Add constant for intercept
X_const = sm.add_constant(X_test)

# Fit OLS on training data
ols_model = sm.OLS(y_train, sm.add_constant(X_train_scaled)).fit()

# Get predictions with intervals
predictions = ols_model.get_prediction(X_const)
pred_summary = predictions.summary_frame(alpha=0.05)  # 95% CI

# Show first few rows
pd.set_option('display.float_format', lambda x: '%.2f' % x)
print(pred_summary.head())
```

```
         mean  mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  \
613 178183.91 73937.79       33040.06      323327.76      32884.56
451 147108.68 86028.92      -21770.71      315988.07     -21904.37
731 206672.41 76079.62       57324.04      356020.79      57172.91
436 117028.91 84215.11      -48289.86      282347.68     -48426.40
275 171690.06 75364.86       23744.79      319635.32      23592.23

     obs_ci_upper
613     323483.26
451     316121.74
731     356171.92
436     282484.22
275     319787.88
```

In [63]:
```
# VIF requires a DataFrame (not scaled arrays)
X_vif = pd.DataFrame(X_train_scaled, columns=X_train.columns)

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = X_vif.columns
vif_data["VIF"] = [variance_inflation_factor(X_vif.values, i) for i in range(X_vif.shape[1])]

print(vif_data)
```

	Feature	VIF
0	Age	13.78
1	Diabetes	1.86
2	Blood_Pressure_Problems	2.09
3	Any_Transplants	1.07
4	Any_Chronic_Diseases	1.27
5	Height	33.07
6	Weight	78.16
7	Known_Allergies	1.34
8	History_of_Cancer_in_Family	1.24
9	Number_of_Major_Surgeries	2.56
10	BMI	101.87
11	BMI_Category_Normal	6.48
12	BMI_Category_Overweight	10.07
13	BMI_Category_Obese	14.30
14	Age_Group_30-39	2.99
15	Age_Group_40-49	7.90
16	Age_Group_50-59	13.08
17	Age_Group_60+	16.33

🔎 Interpretation (later):

VIF < 5 → good (no multicollinearity issue).

VIF between 5–10 → moderate correlation, worth checking.

VIF > 10 → serious multicollinearity, consider dropping or combining features.

4.2.1 - Ridge

In [64]:
```
# --- Ridge Regression with Cross-Validation ---
alphas = np.logspace(-3, 3, 50)  # range of alpha values
```

```
ridge_cv = RidgeCV(alphas=alphas, cv=5, scoring='r2')
ridge_cv.fit(X_train_scaled, y_train)

# Predictions
y_train_pred_ridge = ridge_cv.predict(X_train_scaled)
y_test_pred_ridge = ridge_cv.predict(X_test_scaled)

# Metrics
ridge_train_r2 = r2_score(y_train, y_train_pred_ridge)
ridge_test_r2 = r2_score(y_test, y_test_pred_ridge)
ridge_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred_ridge))

print(f"Best alpha: {ridge_cv.alpha_:.4f}")
print(f"R² (Train): {ridge_train_r2:.3f}")
print(f"R² (Test): {ridge_test_r2:.3f}")
print(f"RMSE (Test): {ridge_rmse:.2f}")
```

```
Best alpha: 0.1207
R² (Train): 0.699
R² (Test): 0.786
RMSE (Test): 3022.10
```

4.2.2 - Lasso

In [65]:

```
# --- Lasso Regression with Cross-Validation ---
alphas = np.logspace(-3, 3, 50)
lasso_cv = LassoCV(alphas=alphas, cv=5, random_state=42, max_iter=5000)
lasso_cv.fit(X_train_scaled, y_train)

# Predictions
y_train_pred_lasso = lasso_cv.predict(X_train_scaled)
y_test_pred_lasso = lasso_cv.predict(X_test_scaled)

# Metrics
lasso_train_r2 = r2_score(y_train, y_train_pred_lasso)
lasso_test_r2 = r2_score(y_test, y_test_pred_lasso)
lasso_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred_lasso))

print(f"Best alpha: {lasso_cv.alpha_:.4f}")
print(f"R² (Train): {lasso_train_r2:.3f}")
print(f"R² (Test): {lasso_test_r2:.3f}")
print(f"RMSE (Test): {lasso_rmse:.2f}")

# Count features kept
n_nonzero = np.sum(lasso_cv.coef_ != 0)
print(f"Number of features kept: {n_nonzero}/{len(lasso_cv.coef_)}")
```

```
Best alpha: 1.5264
R² (Train): 0.699
R² (Test): 0.786
RMSE (Test): 3021.68
Number of features kept: 18/18
```

4.2.3 - ElasticNet

In [66]:

```
# --- Elastic Net Regression with Cross-Validation ---
l1_ratios = [0.1, 0.3, 0.5, 0.7, 0.9]  # balance between Lasso (1) and Ridge (0)
alphas = np.logspace(-3, 3, 50)

elastic_cv = ElasticNetCV(l1_ratio=l1_ratios, alphas=alphas, cv=5,
                          random_state=42, max_iter=5000)
elastic_cv.fit(X_train_scaled, y_train)

# Predictions
y_train_pred_elastic = elastic_cv.predict(X_train_scaled)
y_test_pred_elastic = elastic_cv.predict(X_test_scaled)

# Metrics
elastic_train_r2 = r2_score(y_train, y_train_pred_elastic)
elastic_test_r2 = r2_score(y_test, y_test_pred_elastic)
elastic_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred_elastic))

print(f"Best alpha: {elastic_cv.alpha_:.4f}")
print(f"Best l1_ratio: {elastic_cv.l1_ratio_:.2f}")
print(f"R² (Train): {elastic_train_r2:.3f}")
print(f"R² (Test): {elastic_test_r2:.3f}")
print(f"RMSE (Test): {elastic_rmse:.2f}")
```

```
Best alpha: 0.0010
Best l1_ratio: 0.90
R² (Train): 0.699
R² (Test): 0.786
RMSE (Test): 3021.54
```

4.2.4 - Comparison

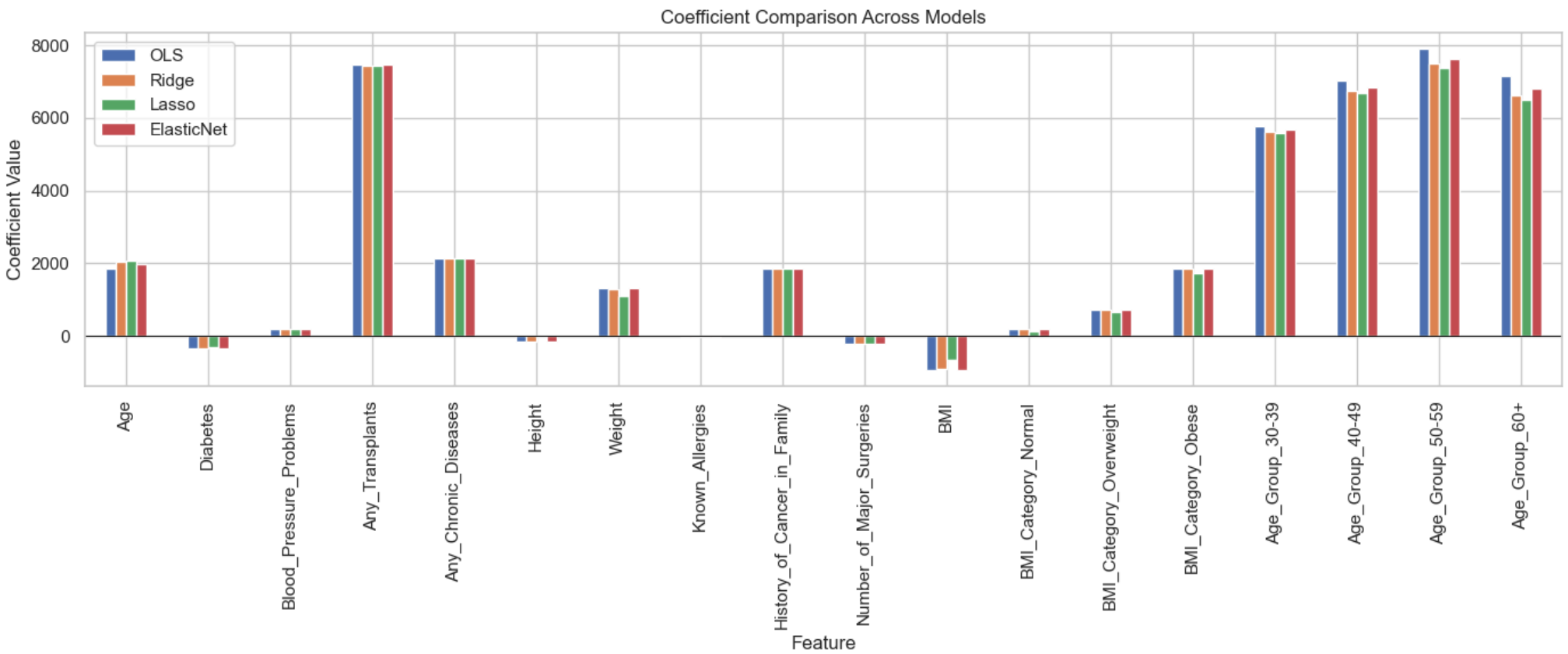
In [67]:

```
# --- Collect results into a dataframe ---
results = pd.DataFrame({
    "Model": ["Linear Regression", "Ridge", "Lasso", "Elastic Net"],
    "Train R²": [
        r2_score(y_train, y_train_pred),          # OLS
```

```
        ridge_train_r2, lasso_train_r2, elastic_train_r2
    ],
    "Test R²": [
        r2_score(y_test, y_test_pred),            # OLS
        ridge_test_r2, lasso_test_r2, elastic_test_r2
    ],
    "RMSE (Test)": [
        np.sqrt(mean_squared_error(y_test, y_test_pred)),   # OLS
        ridge_rmse, lasso_rmse, elastic_rmse
    ],
    "Best Alpha": [
        "N/A", ridge_cv.alpha_, lasso_cv.alpha_, elastic_cv.alpha_
    ],
    "l1_ratio": [
        "N/A", "0 (ridge)", "1 (lasso)", elastic_cv.l1_ratio_
    ],
    "Non-zero Features": [
        X_train_scaled.shape[1],                    # OLS keeps all
        np.sum(ridge_cv.coef_ != 0),
        np.sum(lasso_cv.coef_ != 0),
        np.sum(elastic_cv.coef_ != 0)
    ]
})

display(results)
```

	Model	Train R^2	Test R^2	RMSE (Test)	Best Alpha	l1_ratio	Non-zero Features
0	Linear Regression	0.70	0.79	3020.59	N/A	N/A	18
1	Ridge	0.70	0.79	3022.10	0.12	0 (ridge)	18
2	Lasso	0.70	0.79	3021.68	1.53	1 (lasso)	18
3	Elastic Net	0.70	0.79	3021.54	0.00	0.90	18

In [68]:
```
# --- Coefficient Comparison Plot ---
coef_df = pd.DataFrame({
    "Feature": X_train_scaled.columns,
    "OLS": lin_reg.coef_,
    "Ridge": ridge_cv.coef_,
    "Lasso": lasso_cv.coef_,
    "ElasticNet": elastic_cv.coef_
})

coef_df.set_index("Feature").plot(kind="bar", figsize=(14,6))
plt.title("Coefficient Comparison Across Models")
plt.ylabel("Coefficient Value")
plt.xticks(rotation=90)
plt.axhline(0, color='black', linewidth=0.8)
plt.legend()
plt.tight_layout()
plt.show()
```



🔎 Model Comparison Insights – Linear Models

✅ Performance Metrics

- **Linear Regression (Baseline)**:
  - Train R^2 = 0.70 | Test R^2 = 0.79 | RMSE ≈ 3020
  - Good generalization, but risk of overfitting if assumptions are violated.
- **Ridge Regression**:
  - Train R^2 = 0.70 | Test R^2 = 0.79 | RMSE ≈ 3022

- Similar performance to OLS, but coefficients are shrunk → more stable under multicollinearity.

- **Lasso Regression**:
  - Train R^2 = 0.70 | Test R^2 = 0.79 | RMSE ≈ 3021
  - No feature elimination here (all 18 features retained). However, useful in case of redundant predictors.
- **Elastic Net**:
  - Train R^2 = 0.70 | Test R^2 = 0.79 | RMSE ≈ 3021
  - Combines L1 and L2 regularization; results very close to Ridge/Lasso.

📌 **Conclusion**: All linear models perform almost identically on this dataset. Regularization does not significantly improve test performance, suggesting multicollinearity is limited and all features are relevant.

✅ Coefficient Shrinkage Insights

- **High Impact Predictors**:
  - `Any_Transplants`, `Age_Group_60+`, `Age_Group_50-59`, and `History_of_Cancer_in_Family` consistently have the largest positive coefficients across all models → strong predictors of higher premiums.
- **Negative Predictors**:
  - `BMI` (continuous), `Diabetes`, and `Number_of_Major_Surgeries` show negative coefficients in some models → might reduce premiums or act as controlled factors after considering other risks.
- **Regularization Effect**:
  - Ridge and Elastic Net slightly shrink extreme values but do not change feature importance ranking.
  - Lasso did **not** drop any features (all coefficients remain non-zero), meaning no redundant variables strong enough to be eliminated.

✅ Business Insights

- Insurance premiums are **most sensitive to age groups (esp. 50+) and transplant history**, aligning with domain expectations.
- BMI plays a complex role: while obesity categories increase premiums, raw BMI is negatively weighted (possible overlap with categorical BMI encoding).
- Since all models agree on feature importance, **linear models already capture the key drivers of premium costs**.

✅ Next Steps

- **Tree-Based Models** (Decision Tree, Random Forest, Gradient Boosting) → to check if non-linear relationships improve accuracy.
- **Model Explainability** (Permutation Importance, SHAP) → to strengthen feature-level interpretation.
- **Check Feature Engineering** → review interaction effects (e.g., Age × BMI, Chronic Diseases × Age) to see if premiums depend on combinations of factors.

4.3 - Decision Tree

4.3.1 - Baseline Model

In [69]:
```
#from sklearn.tree import DecisionTreeRegressor
#from sklearn.metrics import mean_squared_error, r2_score

# --- Baseline Decision Tree ---
dt_reg = DecisionTreeRegressor(random_state=42)

# Fit on training data
dt_reg.fit(X_train_scaled, y_train)

# Predictions
y_train_pred_dt = dt_reg.predict(X_train_scaled)
y_test_pred_dt  = dt_reg.predict(X_test_scaled)

# Metrics
train_r2_dt = r2_score(y_train, y_train_pred_dt)
test_r2_dt  = r2_score(y_test, y_test_pred_dt)
rmse_dt     = np.sqrt(mean_squared_error(y_test, y_test_pred_dt))

print("Decision Tree (Baseline)")
print(f"Train R²: {train_r2_dt:.3f}")
print(f"Test R²:  {test_r2_dt:.3f}")
print(f"Test RMSE: {rmse_dt:.2f}")
```

```
Decision Tree (Baseline)
Train R²: 1.000
Test R²:  0.711
Test RMSE: 3509.73
```

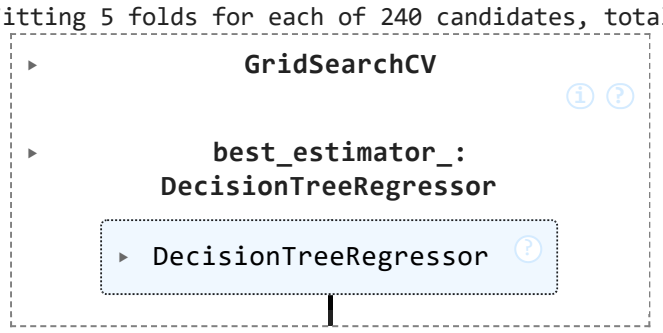
4.3.2 - Hyperparameter Tuning

In [70]:
```
# --- Hyperparameter Grid ---
param_grid = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
```

```
}

dt_reg_tuned = DecisionTreeRegressor(random_state=42)

# Grid Search with 5-fold CV
grid_search = GridSearchCV(
    estimator=dt_reg_tuned,
    param_grid=param_grid,
    scoring='r2',
    cv=5,
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
```

Out[70]:



4.3.3 - Best Model Evaluation

In [71]:

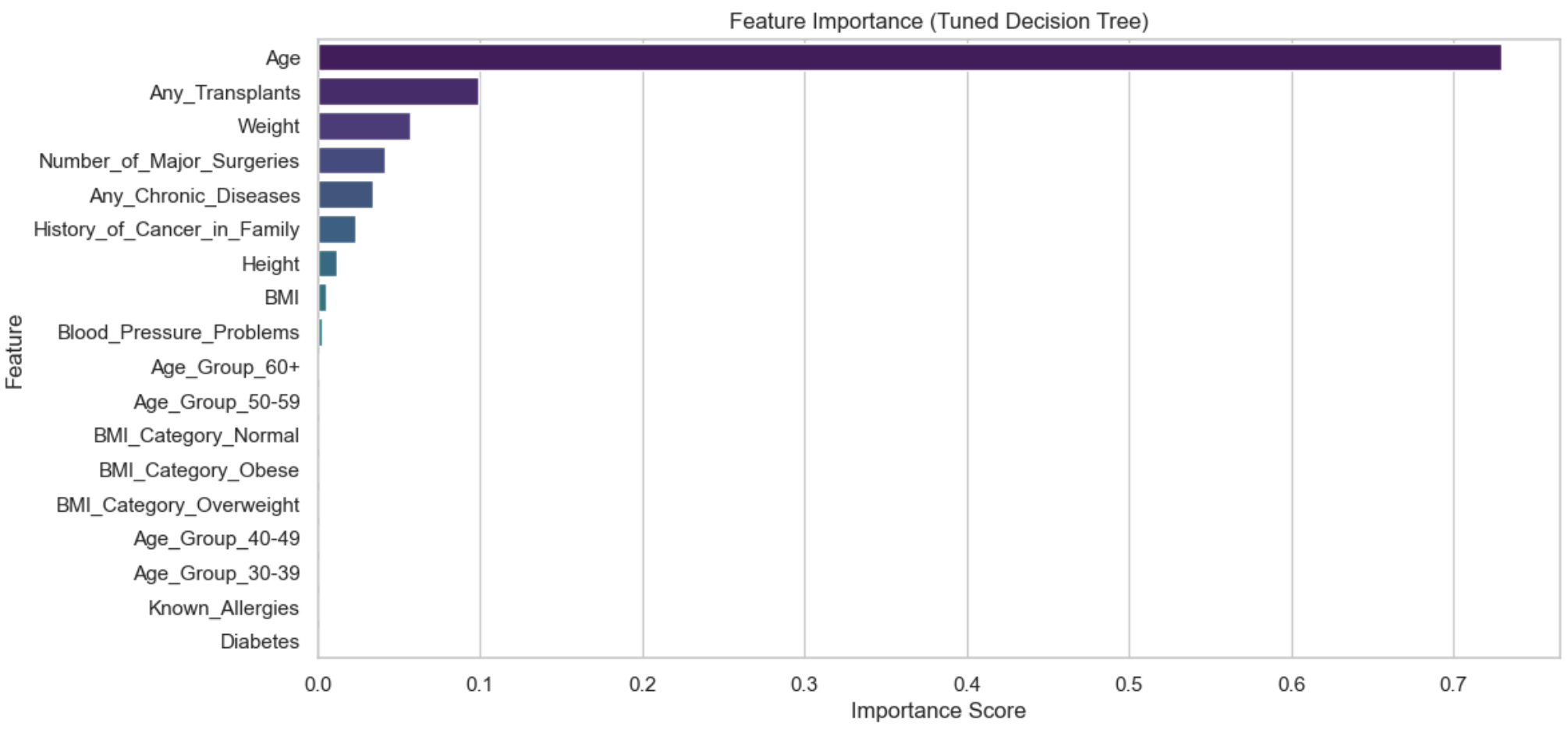
```
# Best Model
best_dt = grid_search.best_estimator_

print("Best Parameters:", grid_search.best_params_)
print("Best CV R²:", grid_search.best_score_)

# Predictions with tuned model
y_test_pred_best = best_dt.predict(X_test_scaled)

print("Tuned Decision Tree")
print(f"Train R²: {r2_score(y_train, best_dt.predict(X_train_scaled)):.3f}")
print(f"Test R²:  {r2_score(y_test, y_test_pred_best):.3f}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test, y_test_pred_best)):.2f}")
```

```
Best Parameters: {'max_depth': 10, 'max_features': None, 'min_samples_leaf': 5, 'min_samples_split': 20}
Best CV R²: 0.7299152163552399
Tuned Decision Tree
Train R²: 0.828
Test R²:  0.884
Test RMSE: 2220.20
```

4.3.4 - Feature Importance

In [72]:

```
# --- Feature Importance Plot for Decision Tree ---
importances = best_dt.feature_importances_
features = X.columns

# Sort features by importance
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(12,6))
sns.barplot(x=importances[indices], y=features[indices], palette="viridis", orient='h')
plt.title("Feature Importance (Tuned Decision Tree)")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.show()
```



4.4 - Random Forrest

4.4.1 - Baseline Model

In [73]:
```
# --- Baseline Random Forest ---
rf_reg = RandomForestRegressor(random_state=42, n_jobs=-1)
rf_reg.fit(X_train, y_train)

# Predictions
y_train_pred = rf_reg.predict(X_train)
y_test_pred = rf_reg.predict(X_test)

# Metrics
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("Random Forest (Baseline)")
print(f"Train R²: {train_r2:.3f}")
print(f"Test R²:  {test_r2:.3f}")
print(f"Test RMSE: {test_rmse:.2f}")
```

```
Random Forest (Baseline)
Train R²: 0.968
Test R²:  0.890
Test RMSE: 2165.39
```

4.4.2 - Hyperparameter Tuning

In [74]:
```
# Define parameter grid
param_dist = {
    "n_estimators": [100, 200, 300, 500],
    "max_depth": [None, 5, 10, 20],
    "min_samples_split": [2, 5, 10, 20],
    "min_samples_leaf": [1, 2, 5, 10],
    'max_features': [None, 'sqrt', 'log2']
}

# Random Forest Regressor
rf_reg = RandomForestRegressor(random_state=42, n_jobs=-1)

# Randomized Search CV
rf_random_search = RandomizedSearchCV(
    estimator=rf_reg,
    param_distributions=param_dist,
    n_iter=30,              # number of random combinations
    cv=5,                   # 5-fold CV
    scoring='r2',           # evaluation metric
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# Fit
rf_random_search.fit(X_train, y_train)

# Best parameters & CV score
print("Best Parameters:", rf_random_search.best_params_)
print("Best CV R²:", rf_random_search.best_score_)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best Parameters: {'n_estimators': 500, 'min_samples_split': 20, 'min_samples_leaf': 5, 'max_features': None, 'max_depth': 10}
Best CV R²: 0.7491941050818869
```

4.4.3 - Best Model Evaluation

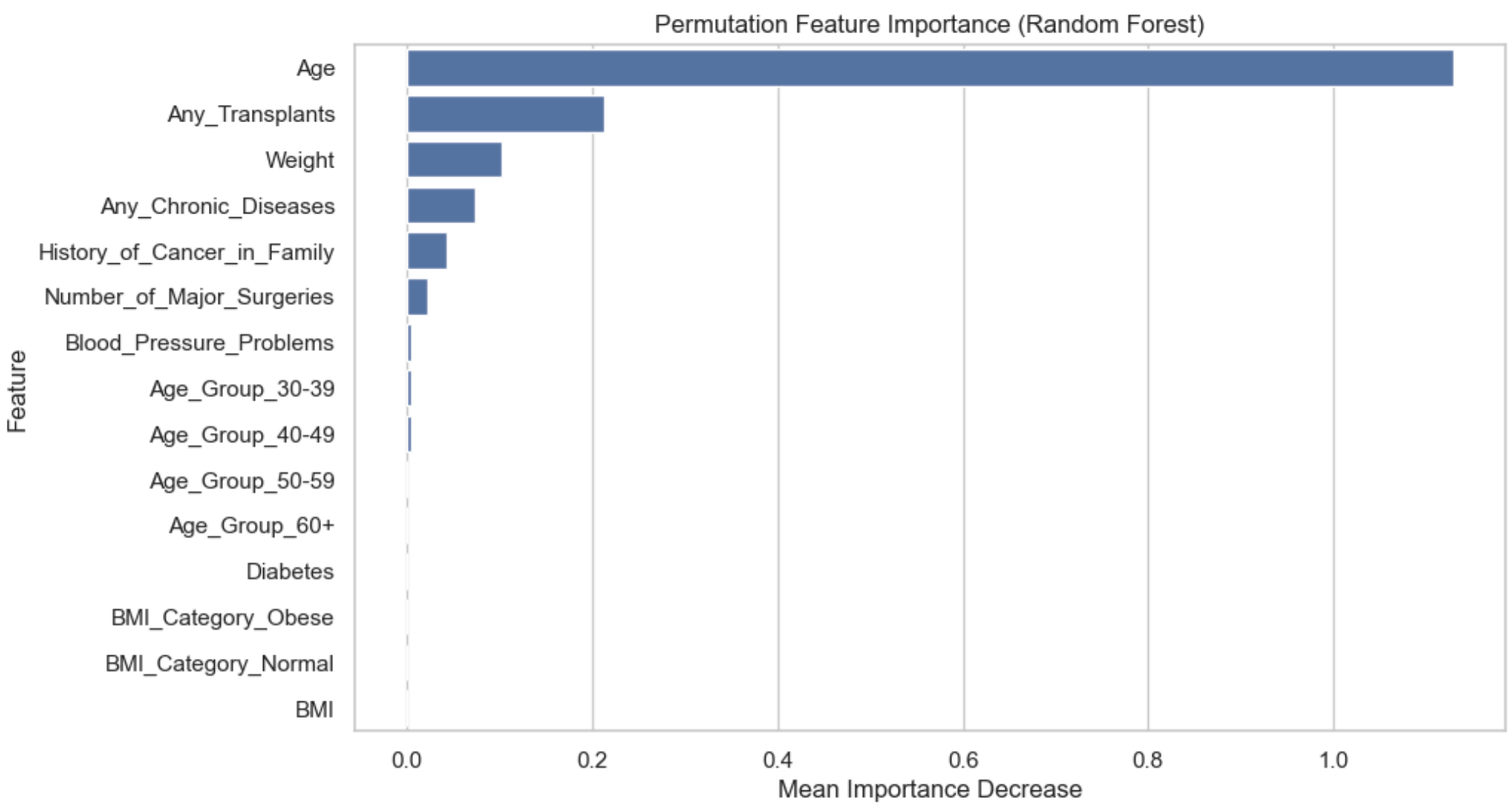
In [75]:
```python
# Best RF model
best_rf = rf_random_search.best_estimator_

# Fit on train set
best_rf.fit(X_train, y_train)

# Predictions
y_train_pred = best_rf.predict(X_train)
y_test_pred = best_rf.predict(X_test)

# Evaluation metrics
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("Tuned Random Forest")
print("Train R²:", round(train_r2, 3))
print("Test R²: ", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

```
Tuned Random Forest
Train R²: 0.822
Test R²:  0.894
Test RMSE: 2129.11
```

4.4.4 - Permutation Feature Importance

In [76]:
```python
# Compute permutation importance on the tuned RF
perm_importance = permutation_importance(
    best_rf, X_test, y_test, n_repeats=20, random_state=42, n_jobs=-1
)

# Convert to DataFrame for readability
importances_df = pd.DataFrame({
    "Feature": X.columns,
    "Importance": perm_importance.importances_mean,
    "Std": perm_importance.importances_std
}).sort_values(by="Importance", ascending=False)

# Display top features
print(importances_df.head(10))

# Plot feature importance
plt.figure(figsize=(10,6))
sns.barplot(x="Importance", y="Feature", data=importances_df.head(15))
plt.title("Permutation Feature Importance (Random Forest)")
plt.xlabel("Mean Importance Decrease")
plt.ylabel("Feature")
plt.show()
```

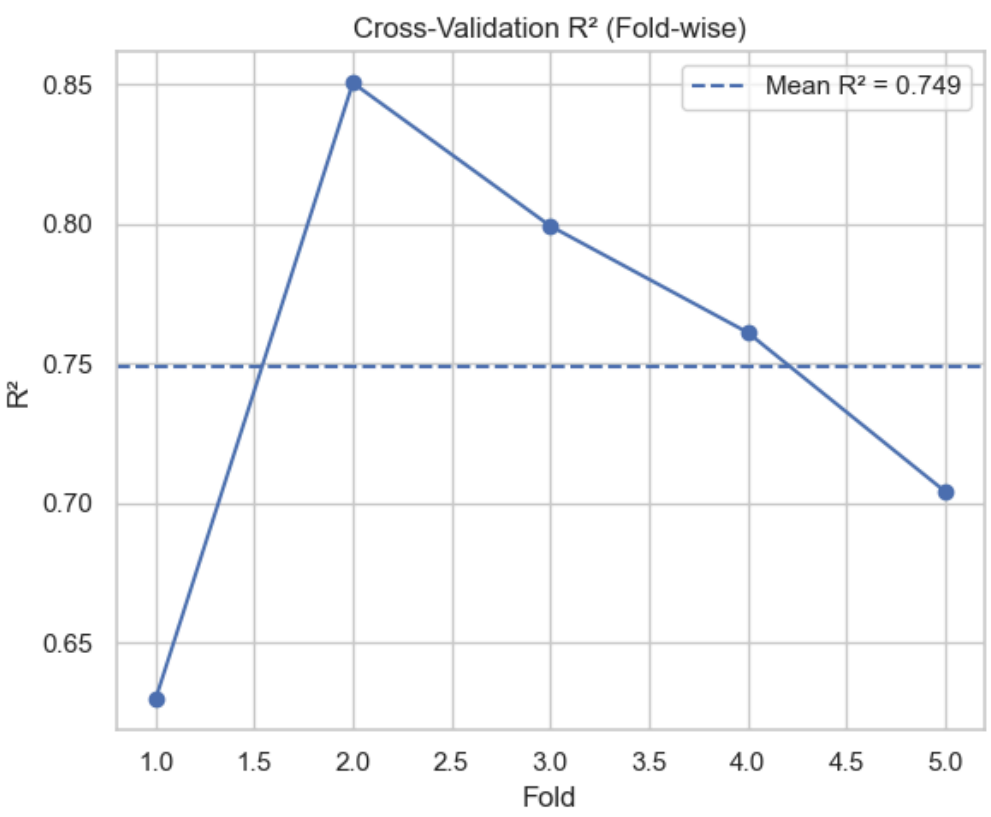
```
                      Feature  Importance  Std
0                         Age        1.13 0.09
3              Any_Transplants        0.21 0.03
6                      Weight        0.10 0.02
4         Any_Chronic_Diseases        0.07 0.02
8   History_of_Cancer_in_Family        0.04 0.01
9     Number_of_Major_Surgeries        0.02 0.01
2       Blood_Pressure_Problems        0.01 0.00
14             Age_Group_30-39        0.00 0.00
15             Age_Group_40-49        0.00 0.00
16             Age_Group_50-59        0.00 0.00
```

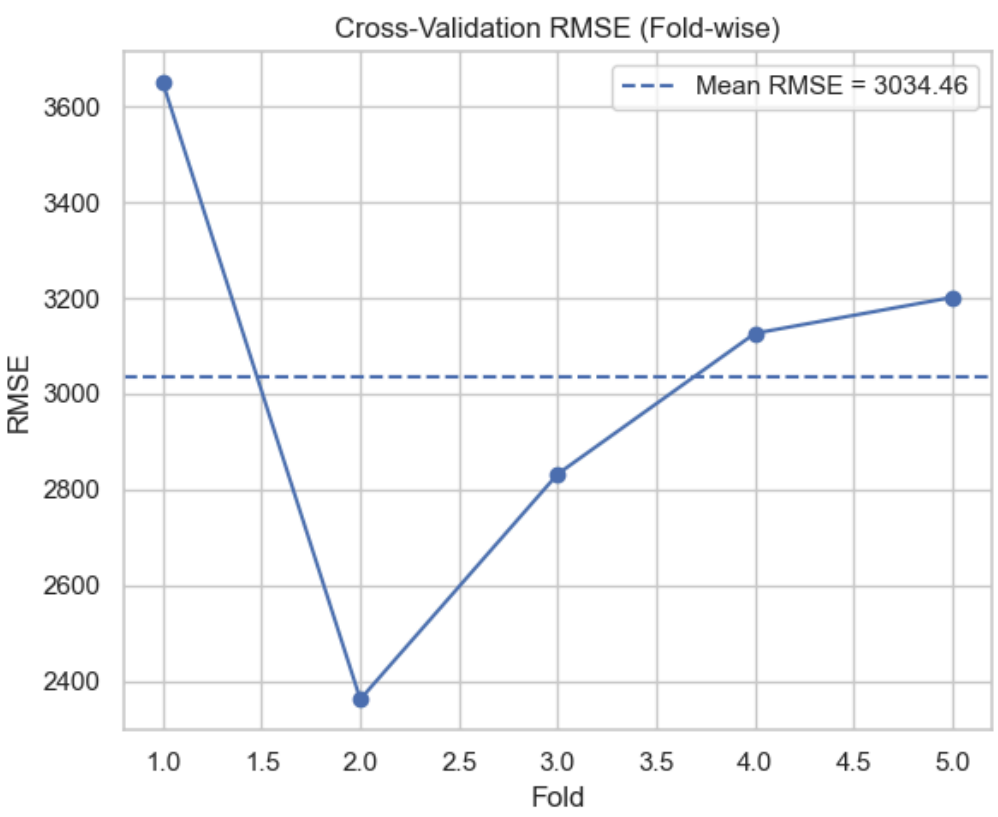


4.4.5 - Cross-Validation Stability

In [77]:
```python
# Use the tuned RF directly from your search object
best_rf = rf_random_search.best_estimator_

# 5-fold CV with shuffling for stability
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# R² across folds
r2_scores = cross_val_score(best_rf, X_train, y_train, cv=kf, scoring="r2")

# RMSE across folds (use neg_mean_squared_error and then sqrt)
neg_mse_scores = cross_val_score(best_rf, X_train, y_train, cv=kf, scoring="neg_mean_squared_error")
rmse_scores = np.sqrt(-neg_mse_scores)

# Print summary
print("Cross-Validation Results (5-Fold)")
print("-----------------------------------")
print(f"R²:   mean = {r2_scores.mean():.3f}, std = {r2_scores.std():.3f}, folds = {np.round(r2_scores, 3)}")
print(f"RMSE: mean = {rmse_scores.mean():.2f}, std = {rmse_scores.std():.2f}, folds = {np.round(rmse_scores, 2)}")

# --- Plots ---
fig, ax = plt.subplots(1, 2, figsize=(12,5))

# Fold indices for x-axis
fold_idx = np.arange(1, len(r2_scores) + 1)

# R² plot
ax[0].plot(fold_idx, r2_scores, marker='o', linestyle='-')
ax[0].axhline(r2_scores.mean(), linestyle='--', label=f"Mean R² = {r2_scores.mean():.3f}")
ax[0].set_title("Cross-Validation R² (Fold-wise)")
ax[0].set_xlabel("Fold")
ax[0].set_ylabel("R²")
ax[0].legend()

# RMSE plot
ax[1].plot(fold_idx, rmse_scores, marker='o', linestyle='-')
ax[1].axhline(rmse_scores.mean(), linestyle='--', label=f"Mean RMSE = {rmse_scores.mean():.2f}")
ax[1].set_title("Cross-Validation RMSE (Fold-wise)")
ax[1].set_xlabel("Fold")
ax[1].set_ylabel("RMSE")
ax[1].legend()

plt.tight_layout()
plt.show()
```

```
Cross-Validation Results (5-Fold)
-----------------------------------
R²:   mean = 0.749, std = 0.077, folds = [0.63  0.851 0.799 0.761 0.704]
RMSE: mean = 3034.46, std = 425.58, folds = [3648.87 2362.84 2832.76 3126.41 3201.43]
```



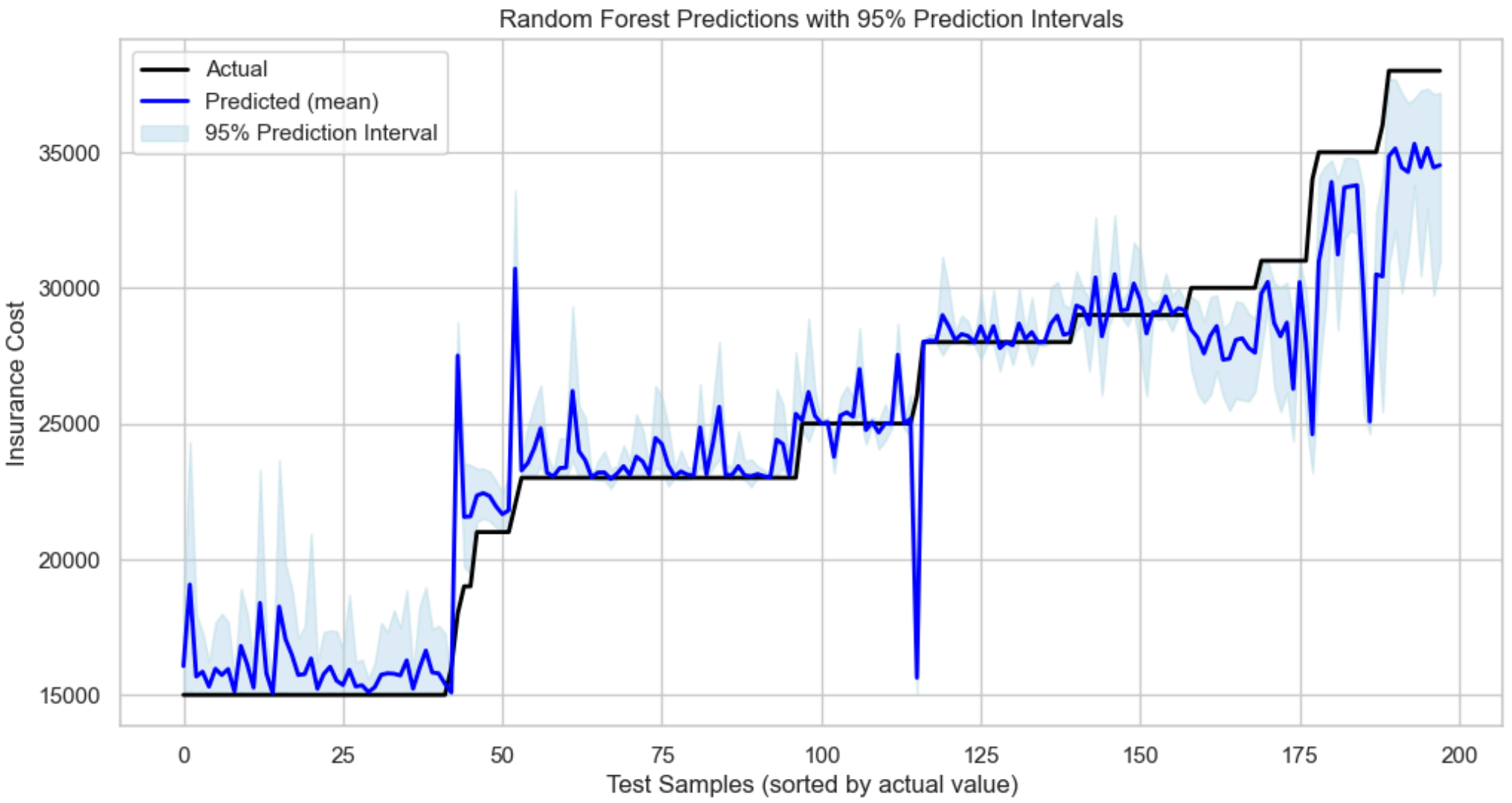


4.4.6 - Confidence/Prediction Interval

In [78]:
```
# Bootstrap settings
n_bootstraps = 100   # number of resampled models
alpha = 0.05         # 95% prediction interval (2.5% and 97.5% quantiles)

# Collect predictions for test set
all_preds = []

for i in range(n_bootstraps):
    # Resample training data
    X_resampled, y_resampled = resample(X_train, y_train, random_state=42+i)

    # Clone best RF and fit on resampled data
    rf_clone = rf_random_search.best_estimator_
    rf_clone.fit(X_resampled, y_resampled)

    # Predict on test set
    preds = rf_clone.predict(X_test)
    all_preds.append(preds)

# Convert to numpy array
all_preds = np.array(all_preds)  # shape: (n_bootstraps, n_test_samples)

# Compute mean prediction + intervals
y_pred_mean = all_preds.mean(axis=0)
lower_bound = np.percentile(all_preds, 100 * alpha/2, axis=0)
upper_bound = np.percentile(all_preds, 100 * (1 - alpha/2), axis=0)

# Show example for first 10 test samples
for i in range(10):
    print(f"Sample {i}: Pred = {y_pred_mean[i]:.2f}, "
          f"95% PI = [{lower_bound[i]:.2f}, {upper_bound[i]:.2f}]")
```

```
Sample 0: Pred = 29788.62, 95% PI = [27738.53, 30884.54]
Sample 1: Pred = 27988.80, 95% PI = [25109.84, 29886.17]
Sample 2: Pred = 30383.91, 95% PI = [29020.35, 32583.06]
Sample 3: Pred = 16061.31, 95% PI = [15016.98, 18493.36]
Sample 4: Pred = 24857.06, 95% PI = [23438.90, 26444.65]
Sample 5: Pred = 28646.95, 95% PI = [26942.74, 29568.98]
Sample 6: Pred = 24670.47, 95% PI = [24066.37, 25076.61]
Sample 7: Pred = 23644.15, 95% PI = [23003.35, 25218.36]
Sample 8: Pred = 28052.89, 95% PI = [27988.82, 28273.66]
Sample 9: Pred = 28002.16, 95% PI = [27902.49, 28100.07]
```

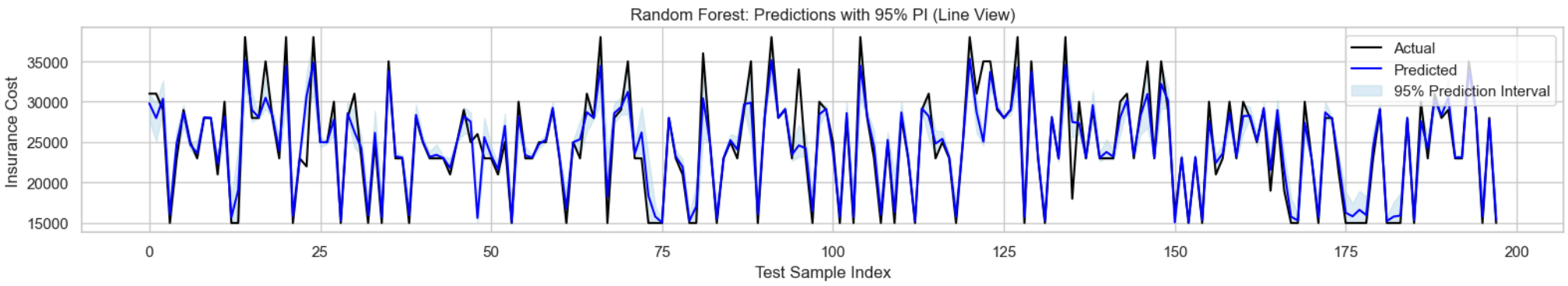
In [79]:
```
# Plotting the predictions with confidence intervals

# Sort test samples by actual values for nicer plotting
sorted_idx = np.argsort(y_test.values)
y_test_sorted = y_test.values[sorted_idx]
y_pred_mean_sorted = y_pred_mean[sorted_idx]
lower_sorted = lower_bound[sorted_idx]
upper_sorted = upper_bound[sorted_idx]

# Plot
plt.figure(figsize=(12, 6))
plt.plot(y_test_sorted, label="Actual", color="black", linewidth=2)
plt.plot(y_pred_mean_sorted, label="Predicted (mean)", color="blue", linewidth=2)
plt.fill_between(range(len(y_test_sorted)),
                 lower_sorted, upper_sorted,
                 color="lightblue", alpha=0.4, label="95% Prediction Interval")

plt.xlabel("Test Samples (sorted by actual value)")
plt.ylabel("Insurance Cost")
plt.title("Random Forest Predictions with 95% Prediction Intervals")
plt.legend()
plt.show()
```

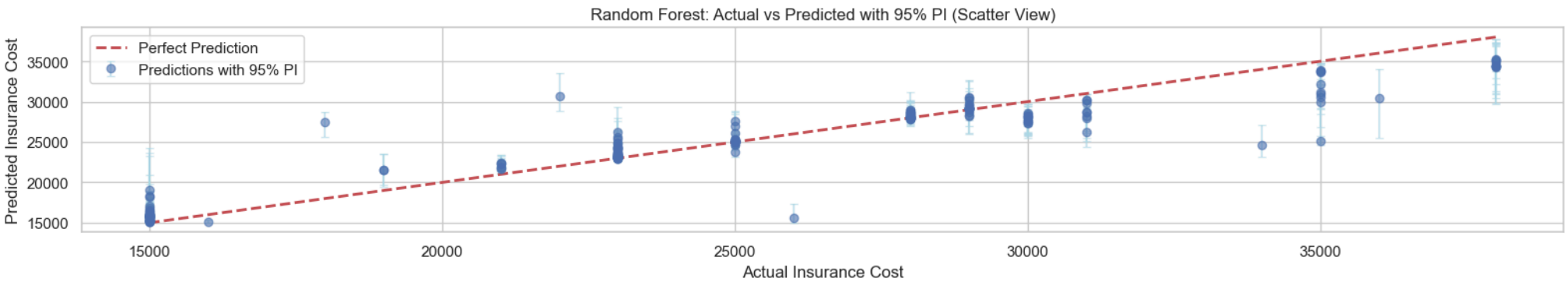


In [80]:
```
# --- Check coverage of 95% Prediction Interval ---
inside_interval = np.mean((y_test.values >= lower_bound) & (y_test.values <= upper_bound)) * 100
avg_interval_width = np.mean(upper_bound - lower_bound)

print(f"Coverage within 95% Prediction Interval: {inside_interval:.2f}% of test samples")
print(f"Average Interval Width: {avg_interval_width:.2f}")
```

```
fig, axes = plt.subplots(2, 1, figsize=(16, 6))

# --- Plot 1: Line plot with uncertainty bands ---
axes[0].plot(range(len(y_test)), y_test.values, label="Actual", color="black")
axes[0].plot(range(len(y_test)), y_pred_mean, label="Predicted", color="blue")
axes[0].fill_between(range(len(y_test)), lower_bound, upper_bound,
                     color="lightblue", alpha=0.4, label="95% Prediction Interval")

axes[0].set_title("Random Forest: Predictions with 95% PI (Line View)")
axes[0].set_xlabel("Test Sample Index")
axes[0].set_ylabel("Insurance Cost")
axes[0].legend()

# --- Plot 2: Scatter with error bars ---
axes[1].errorbar(y_test, y_pred_mean,
                 yerr=[y_pred_mean - lower_bound, upper_bound - y_pred_mean],
                 fmt='o', ecolor='lightblue', alpha=0.6, capsize=3, label="Predictions with 95% PI")

axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
             'r--', lw=2, label="Perfect Prediction")

axes[1].set_title("Random Forest: Actual vs Predicted with 95% PI (Scatter View)")
axes[1].set_xlabel("Actual Insurance Cost")
axes[1].set_ylabel("Predicted Insurance Cost")
axes[1].legend()

plt.tight_layout()
plt.show()
```

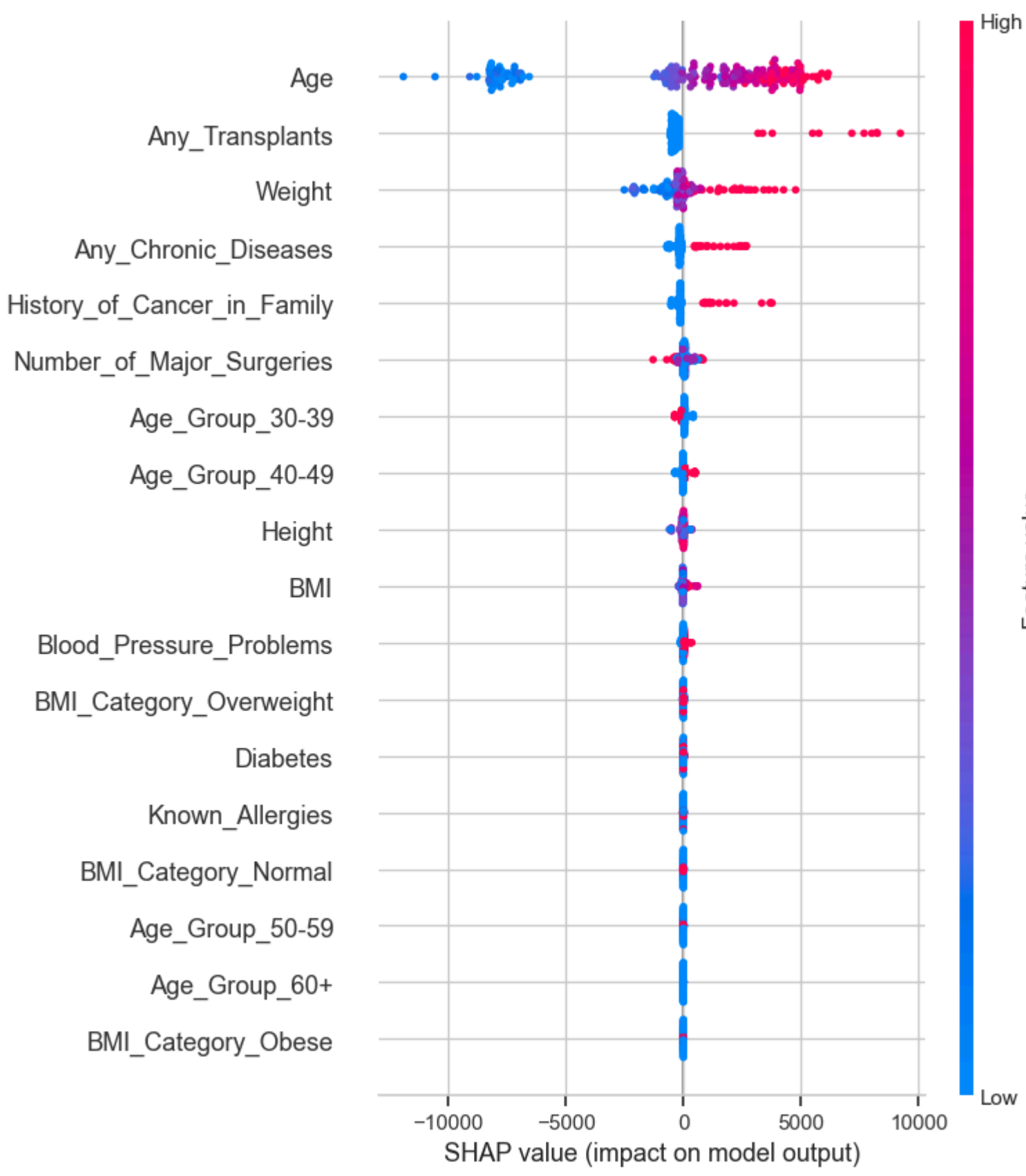
```
Coverage within 95% Prediction Interval: 37.37% of test samples
Average Interval Width: 2390.88
```

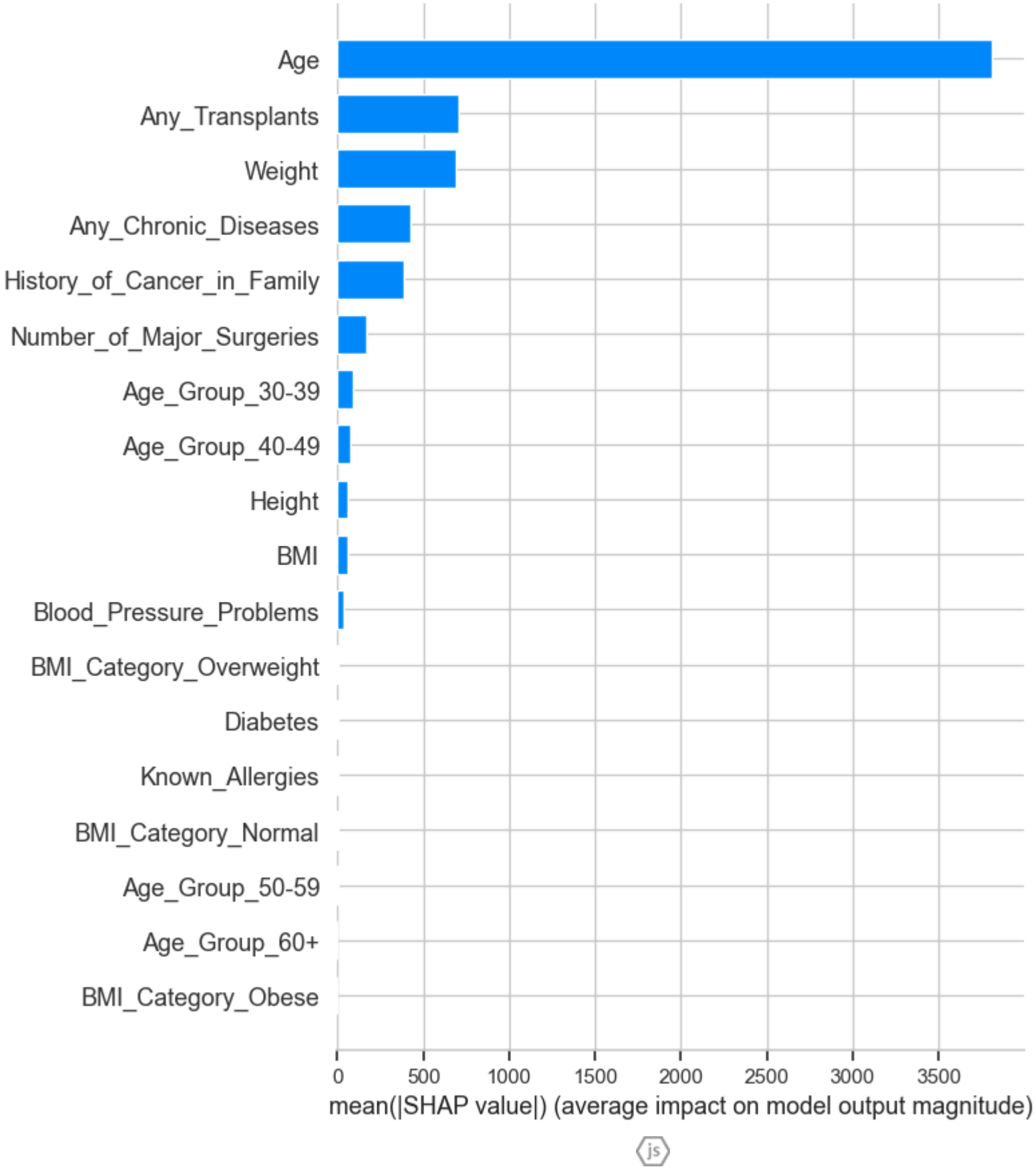




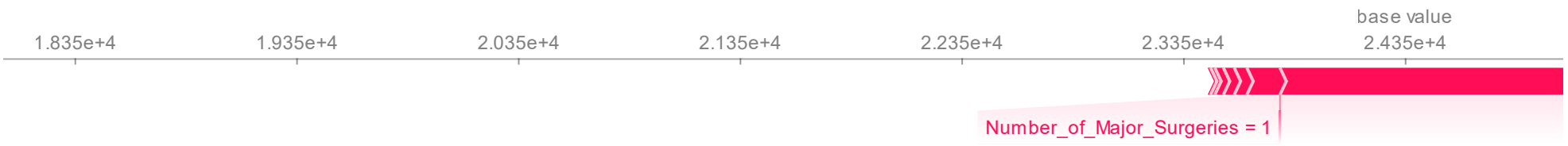
4.4.7 - SHAP Values

In [81]:
```python
# --- SHAP Values for Feature Importance ---

# --- Initialize SHAP explainer for the tuned RF model ---
explainer = shap.TreeExplainer(best_rf)
shap_values = explainer.shap_values(X_test)

# --- Summary Plot (global importance) ---
shap.summary_plot(shap_values, X_test, plot_type="dot")

# --- Bar Plot (mean absolute importance) ---
shap.summary_plot(shap_values, X_test, plot_type="bar")

# --- Example Force Plot for a single prediction ---
sample_idx = 5  # pick a test sample index
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[sample_idx,:], X_test.iloc[sample_idx,:])
```


Age
Any_Transplants
Weight
Any_Chronic_Diseases
History_of_Cancer_in_Family
Number_of_Major_Surgeries
Age_Group_30-39
Age_Group_40-49
Height
BMI
Blood_Pressure_Problems
BMI_Category_Overweight
Diabetes
Known_Allergies
BMI_Category_Normal
Age_Group_50-59
Age_Group_60+
BMI_Category_Obese
−10000
−5000
0
5000
10000
SHAP value (impact on model output)
High
Feature value
Low

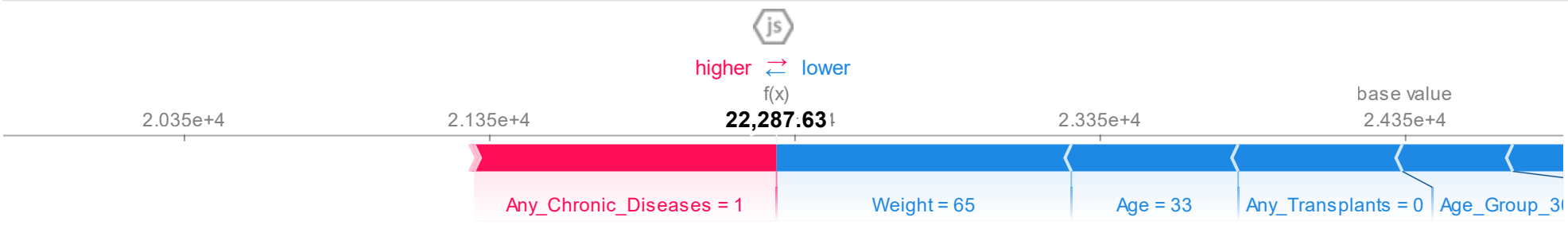


Out[81]:



In [82]:
```
# --- Example Force Plot for a single prediction ---
sample_idx = 10  # pick a test sample index
shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[sample_idx,:], X_test.iloc[sample_idx,:])
```

Out[82]:



4.5 - Gradient Boost

4.5.1 - Baseline Model

In [83]:
```
# Baseline Gradient Boosting
gbr_baseline = GradientBoostingRegressor(random_state=42)
gbr_baseline.fit(X_train, y_train)

# Predictions
y_train_pred = gbr_baseline.predict(X_train)
y_test_pred = gbr_baseline.predict(X_test)

# Metrics
```

```
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("Baseline Gradient Boosting")
print("Train R²:", round(train_r2, 3))
print("Test R²:", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

```
Baseline Gradient Boosting
Train R²: 0.888
Test R²: 0.875
Test RMSE: 2306.52
```

4.5.2 - Hyperparameter Tuning

In [84]:

```
# Define parameter grid
param_dist = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.001, 0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6, 8, 10],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5, 10],
    'max_features': ['sqrt', 'log2', None]
}

# Model
gbm = GradientBoostingRegressor(random_state=42)

# RandomizedSearchCV
gbm_random_search = RandomizedSearchCV(
    estimator=gbm,
    param_distributions=param_dist,
    n_iter=50,            # number of random combos
    scoring='r2',
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# Fit
gbm_random_search.fit(X_train, y_train)

# Best params
print("Best Parameters:", gbm_random_search.best_params_)
print("Best CV R²:", gbm_random_search.best_score_)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
Best Parameters: {'n_estimators': 500, 'min_samples_split': 20, 'min_samples_leaf': 5, 'max_features': None, 'max_depth': 4, 'learning_rate':
0.01}
Best CV R²: 0.7477850563715591
```

4.5.3 - Best Model Evaluation

In [85]:

```
# Evaluate tuned model
best_gbm = gbm_random_search.best_estimator_

y_train_pred = best_gbm.predict(X_train)
y_test_pred = best_gbm.predict(X_test)

train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("\nTuned Gradient Boosting")
print("Train R²:", round(train_r2, 3))
print("Test R²: ", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

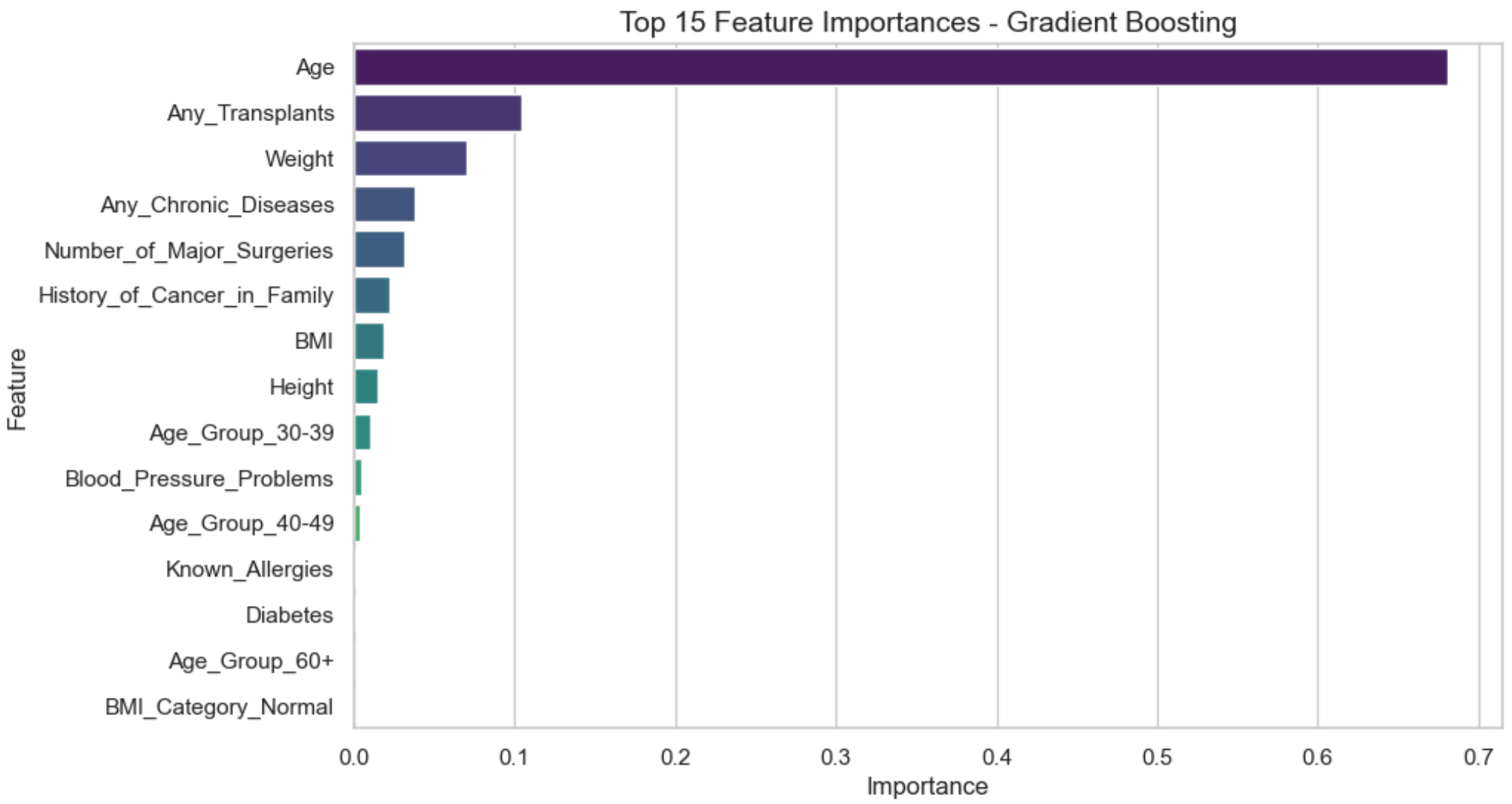
```
Tuned Gradient Boosting
Train R²: 0.853
Test R²:  0.878
Test RMSE: 2278.97
```

4.5.4 - Feature Importance

In [86]:

```
# Feature importance from tuned GBM
importances = best_gbm.feature_importances_
features = X_train.columns

# Put into DataFrame
feat_imp = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feat_imp.head(15), palette="viridis")
plt.title("Top 15 Feature Importances - Gradient Boosting", fontsize=14)
```

```
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```



4.6 - XGBoost

4.6.1 - Baseline Model

In [87]:
```
# Baseline XGBoost Regressor

xgb_model = XGBRegressor(
    random_state=42,
    objective='reg:squarederror'  # for regression
)

# Fit the model
xgb_model.fit(X_train, y_train)

# Predictions
y_train_pred = xgb_model.predict(X_train)
y_test_pred = xgb_model.predict(X_test)

# Metrics
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("Baseline XGBoost")
print("Train R²:", round(train_r2, 3))
print("Test R²: ", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

```
Baseline XGBoost
Train R²: 1.0
Test R²:  0.846
Test RMSE: 2560.05
```

4.6.2 - Hyperparameter Tuning

In [88]:
```
# Define parameter grid for XGBoost
param_dist = {
    "n_estimators": [100, 200, 300, 500],
    "max_depth": [3, 5, 7, 10, 15],
    "learning_rate": [0.01, 0.05, 0.1, 0.2],
    "subsample": [0.6, 0.8, 1.0],
    "colsample_bytree": [0.6, 0.8, 1.0],
    "gamma": [0, 0.1, 0.3, 0.5],
    "reg_alpha": [0, 0.01, 0.1, 1],
    "reg_lambda": [1, 1.5, 2, 3]
}

# XGBoost Regressor
xgb_reg = XGBRegressor(random_state=42, n_jobs=-1)

# Randomized Search CV
xgb_random_search = RandomizedSearchCV(
    estimator=xgb_reg,
```

```
    param_distributions=param_dist,
    n_iter=30,              # number of random combinations
    cv=5,                   # 5-fold CV
    scoring='r2',           # evaluation metric
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# Fit
xgb_random_search.fit(X_train, y_train)

# Best parameters & CV score
print("Best Parameters:", xgb_random_search.best_params_)
print("Best CV R²:", xgb_random_search.best_score_)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best Parameters: {'subsample': 0.8, 'reg_lambda': 2, 'reg_alpha': 0.1, 'n_estimators': 300, 'max_depth': 7, 'learning_rate': 0.05, 'gamma': 0.
1, 'colsample_bytree': 0.6}
Best CV R²: 0.7312974810600281
```

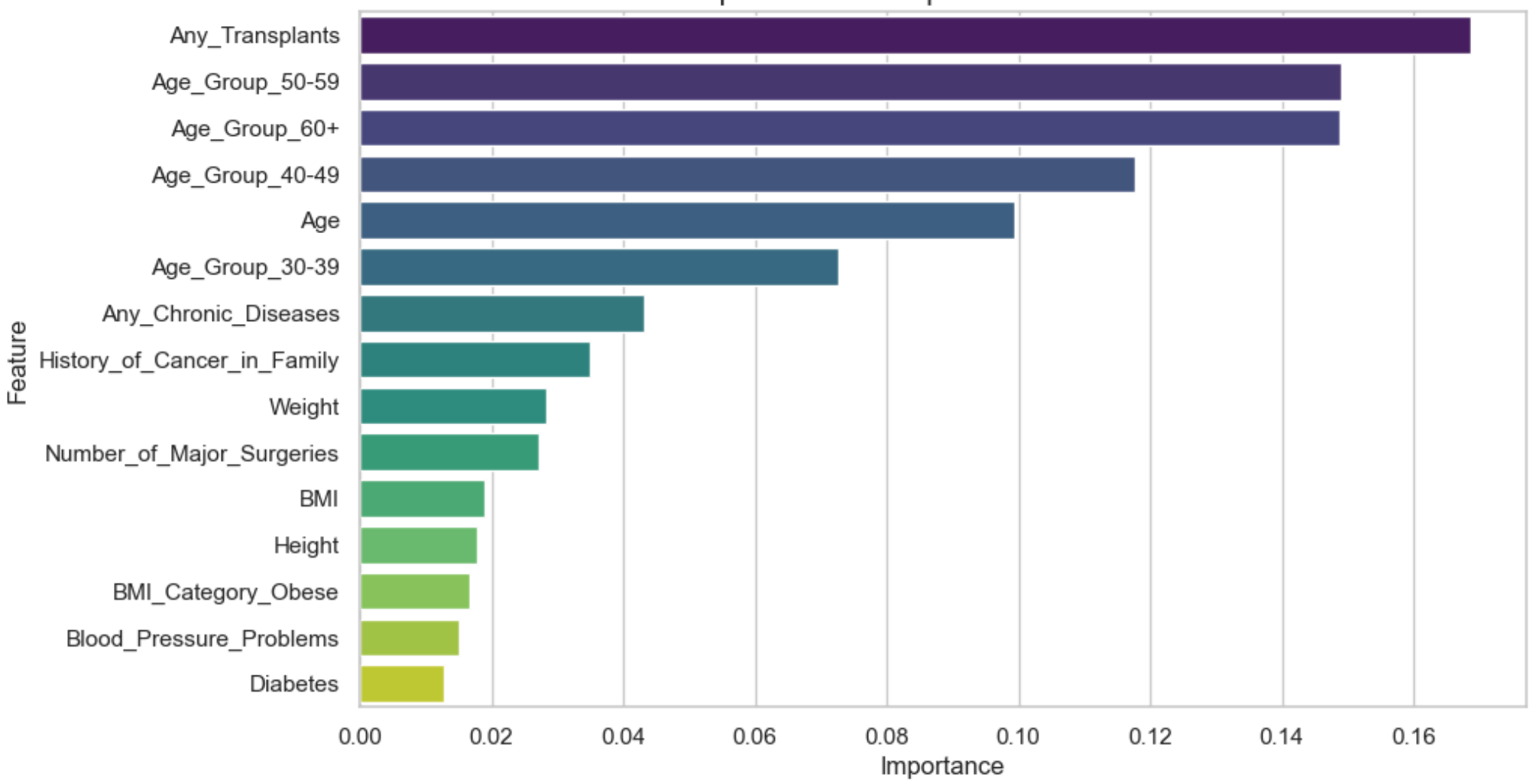
4.6.3 - Best Model Evaluation

In [89]:
```
# Evaluate tuned XGBoost model
best_xgb = xgb_random_search.best_estimator_

# Predictions
y_train_pred = best_xgb.predict(X_train)
y_test_pred = best_xgb.predict(X_test)

# Metrics
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Print results
print("\nTuned XGBoost Regressor")
print("Train R²:", round(train_r2, 3))
print("Test R²: ", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

```
Tuned XGBoost Regressor
Train R²: 0.996
Test R²:  0.858
Test RMSE: 2456.8
```

4.6.4 - Feature Importance

In [90]:
```
# Feature importance from tuned XGBoost
importances = best_xgb.feature_importances_
features = X_train.columns

# Put into DataFrame
feat_imp = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feat_imp.head(15), palette="viridis")
plt.title("Top 15 Feature Importances - XGBoost", fontsize=14)
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```



4.7 - LightGBM

4.7.1 - Baseline Model

In [91]:
```python
# Baseline LightGBM
lgb_baseline = lgb.LGBMRegressor(random_state=42, n_jobs=-1, verbose=-1, force_row_wise=True)
lgb_baseline.fit(X_train, y_train)

# Predictions
y_train_pred = lgb_baseline.predict(X_train)
y_test_pred = lgb_baseline.predict(X_test)

# Metrics
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("Baseline LightGBM")
print("Train R²:", round(train_r2, 3))
print("Test R²:", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

```
Baseline LightGBM
Train R²: 0.926
Test R²: 0.879
Test RMSE: 2275.16
```

4.7.2 - Hyperparameter Tuning

In [92]:
```python
# Parameter grid
param_dist = {
    "n_estimators": [100, 200, 300, 500],
    "max_depth": [-1, 5, 10, 15, 20],
    "learning_rate": [0.01, 0.05, 0.1, 0.2],
    "num_leaves": [31, 50, 70, 100],
    "subsample": [0.6, 0.8, 1.0],
    "colsample_bytree": [0.6, 0.8, 1.0],
    "reg_alpha": [0, 0.01, 0.1, 1],
    "reg_lambda": [1, 1.5, 2, 3]
}

# LightGBM Regressor
lgb_reg = lgb.LGBMRegressor(random_state=42, n_jobs=-1, verbose=-1, force_row_wise=True)

# Randomized Search CV
lgb_random_search = RandomizedSearchCV(
    estimator=lgb_reg,
    param_distributions=param_dist,
    n_iter=30,
    cv=5,
    scoring='r2',
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# Fit
```

```
lgb_random_search.fit(X_train, y_train)

# Best parameters & CV score
print("Best Parameters:", lgb_random_search.best_params_)
print("Best CV R²:", lgb_random_search.best_score_)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
Best Parameters: {'subsample': 0.6, 'reg_lambda': 1, 'reg_alpha': 0, 'num_leaves': 70, 'n_estimators': 500, 'max_depth': 10, 'learning_rate':
0.01, 'colsample_bytree': 0.8}
Best CV R²: 0.7332577163592962
```

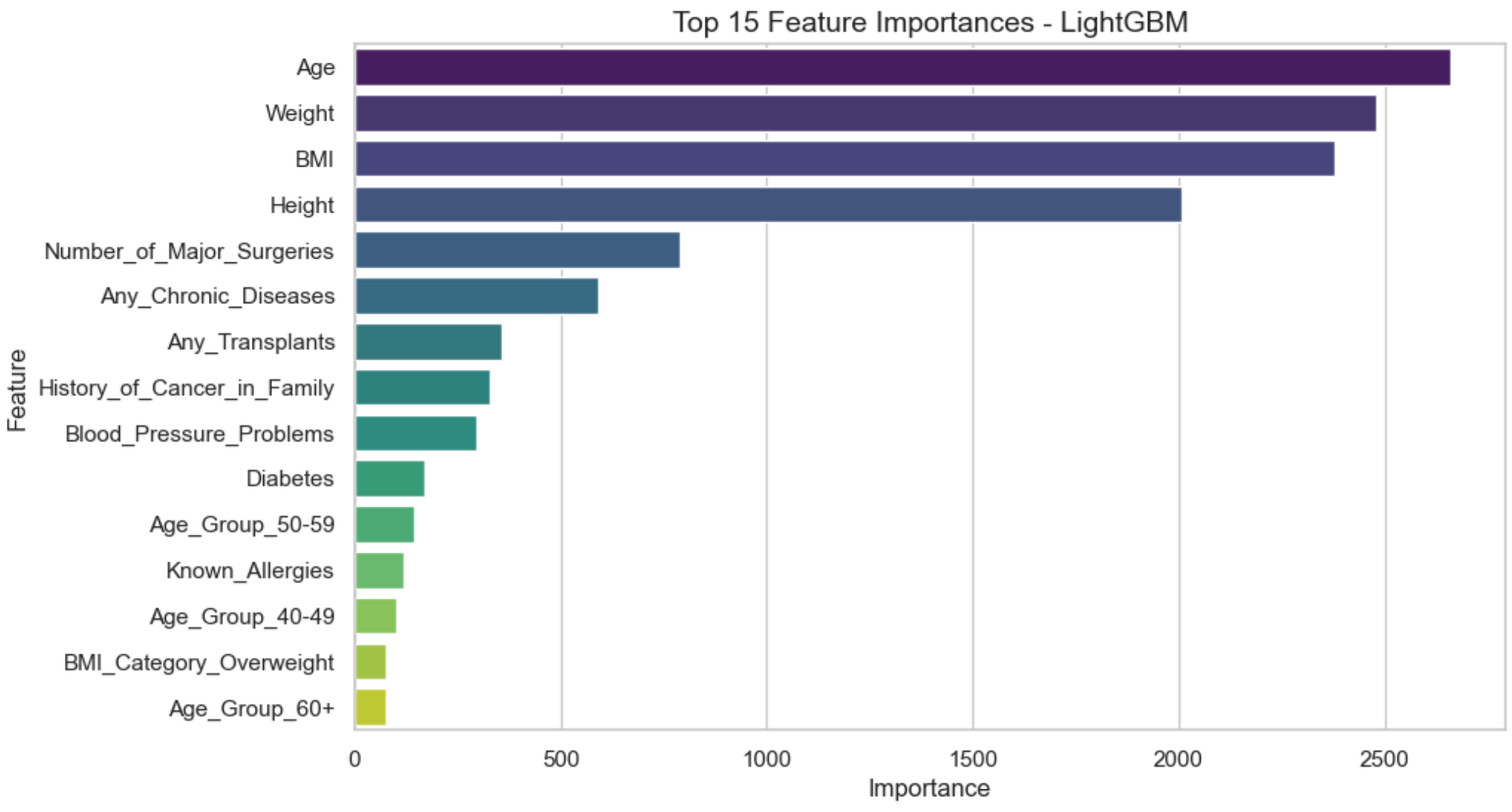
4.7.3 - Best Model Evaluation

In [93]:
```
# Evaluate tuned LightGBM
best_lgb = lgb_random_search.best_estimator_

# Predictions
y_train_pred = best_lgb.predict(X_train)
y_test_pred = best_lgb.predict(X_test)

# Metrics
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Print results
print("\nTuned LightGBM Regressor")
print("Train R²:", round(train_r2, 3))
print("Test R²: ", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

```
Tuned LightGBM Regressor
Train R²: 0.866
Test R²:  0.882
Test RMSE: 2239.31
```

4.7.4 - Feature Importance

In [94]:
```
# Feature importance from tuned LightGBM
importances = best_lgb.feature_importances_
features = X_train.columns

# Put into DataFrame
feat_imp = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feat_imp.head(15), palette="viridis")
plt.title("Top 15 Feature Importances - LightGBM", fontsize=14)
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```



4.8 - Neural Network

4.8.1 - Baseline Model

In [95]:
```
# Baseline Neural Network
nn_baseline = Sequential([
    Dense(64, input_dim=X_train.shape[1], activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)  # regression output
])

nn_baseline.compile(optimizer='adam', loss='mse')

# Fit baseline model
history = nn_baseline.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=100,
    batch_size=32,
    verbose=0
)

# Predictions
y_train_pred = nn_baseline.predict(X_train).flatten()
y_test_pred = nn_baseline.predict(X_test).flatten()

# Metrics
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("Baseline Neural Network")
print("Train R²:", round(train_r2, 3))
print("Test R²:", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
Baseline Neural Network
Train R²: 0.448
Test R²: 0.476
Test RMSE: 4727.5
```

4.8.2 - Hyperparameter Tuning

In [99]:
```
from scikeras.wrappers import KerasRegressor

# Function to create model
def create_nn_model(hidden_layer1=64, hidden_layer2=32, learning_rate=0.001):
    model = Sequential()
    model.add(Dense(hidden_layer1, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dense(hidden_layer2, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='mse')
    return model

# Wrap Keras model (scikeras)
nn_reg = KerasRegressor(model=create_nn_model, verbose=0)

# Parameter grid
param_dist = {
    "model__hidden_layer1": [32, 64, 128],
    "model__hidden_layer2": [16, 32, 64],
    "model__learning_rate": [0.001, 0.005, 0.01, 0.05],
    "batch_size": [16, 32, 64],
    "epochs": [50, 100, 150]
}

# RandomizedSearchCV
nn_random_search = RandomizedSearchCV(
    estimator=nn_reg,
    param_distributions=param_dist,
    n_iter=20,
    cv=5,
    scoring='r2',
    random_state=42,
    n_jobs=-1,
    verbose=2
)

# Fit
nn_random_search.fit(X_train, y_train)

# Best parameters & CV score
print("Best Parameters:", nn_random_search.best_params_)
print("Best CV R²:", nn_random_search.best_score_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best Parameters: {'model__learning_rate': 0.01, 'model__hidden_layer2': 32, 'model__hidden_layer1': 32, 'epochs': 150, 'batch_size': 32}
Best CV R²: 0.6491724371910095
```

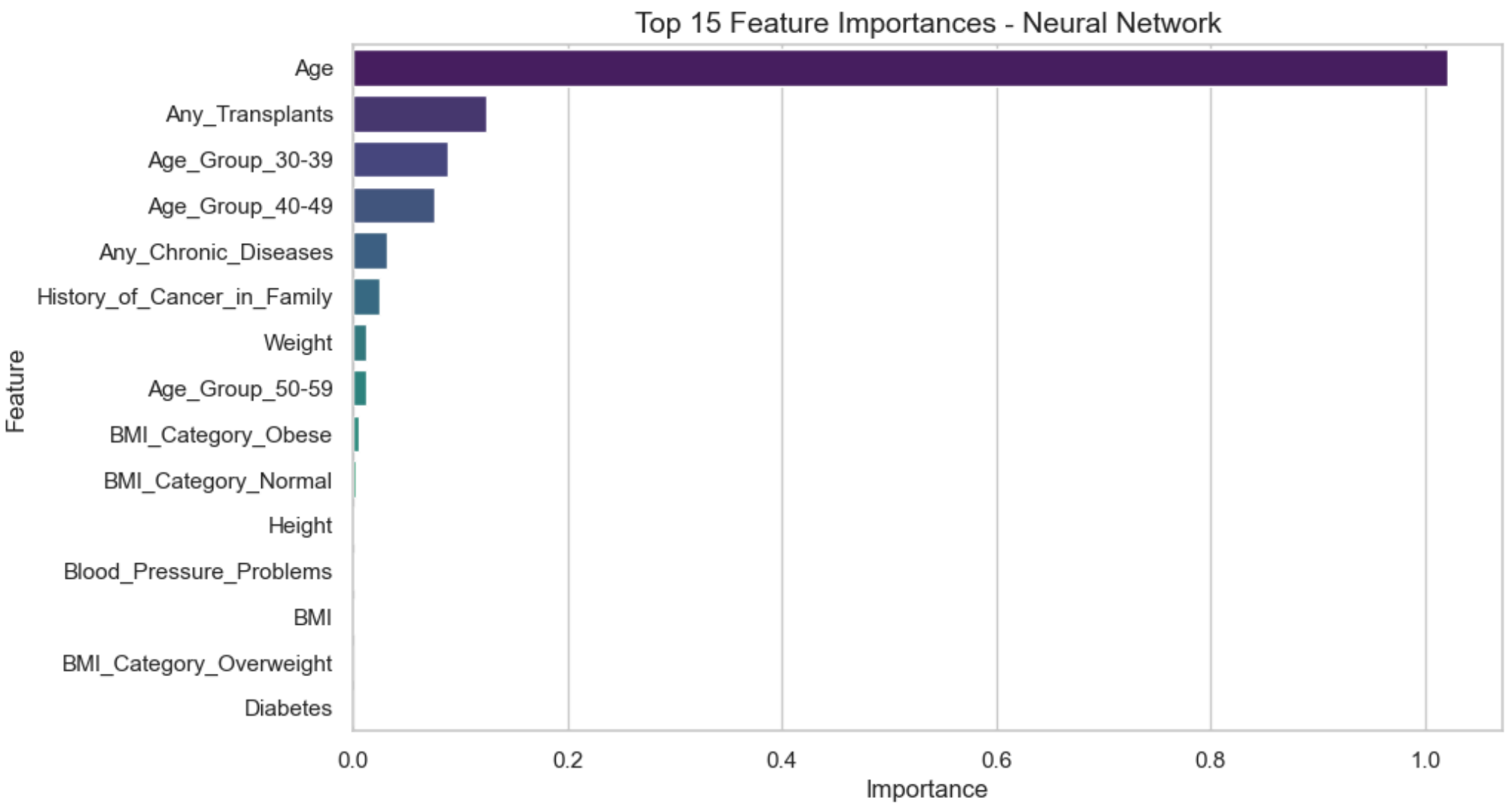
4.8.3 - Best Model Evaluation

In [100]:
```python
# Evaluate tuned Neural Network
best_nn = nn_random_search.best_estimator_

# Predictions
y_train_pred = best_nn.predict(X_train).flatten()
y_test_pred = best_nn.predict(X_test).flatten()

# Metrics
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

# Print results
print("\nTuned Neural Network Regressor")
print("Train R²:", round(train_r2, 3))
print("Test R²: ", round(test_r2, 3))
print("Test RMSE:", round(rmse_test, 2))
```

```
Tuned Neural Network Regressor
Train R²: 0.68
Test R²:  0.76
Test RMSE: 3202.08
```

4.8.4 - Permutation Feature Importance

In [101]:
```python
# Permutation importance on test set
perm_importance = permutation_importance(best_nn, X_test, y_test, n_repeats=10, random_state=42, n_jobs=-1)

# Put into DataFrame
feat_imp = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': perm_importance.importances_mean
}).sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feat_imp.head(15), palette="viridis")
plt.title("Top 15 Feature Importances - Neural Network", fontsize=14)
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

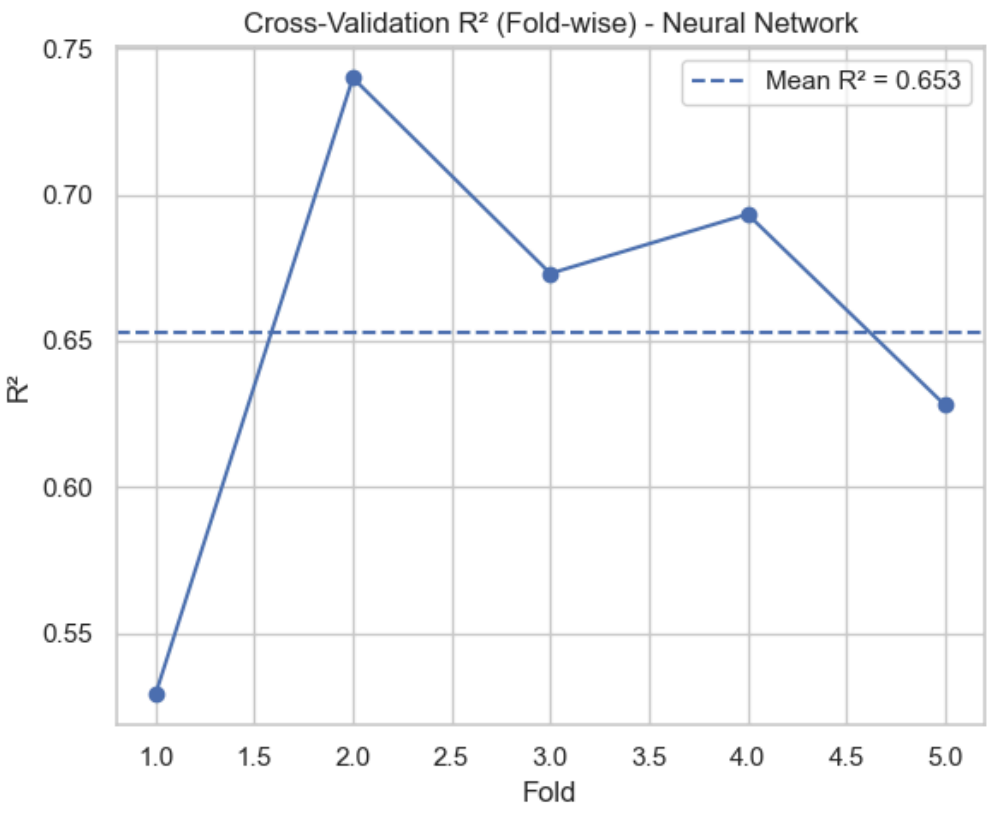


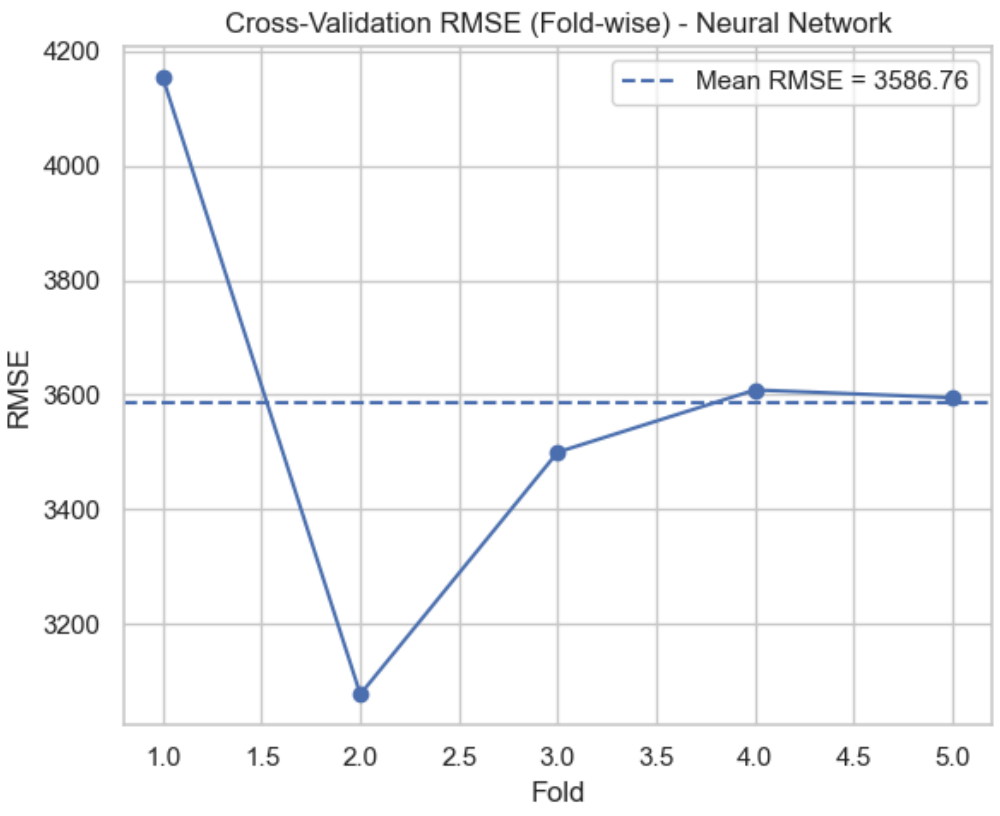
4.8.5 - Cross-Validation Stability

In [102]:
```python
# Use the tuned Neural Network
best_nn = nn_random_search.best_estimator_

# 5-fold CV with shuffling for stability
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# R² across folds
r2_scores = cross_val_score(best_nn, X_train, y_train, cv=kf, scoring="r2")

# RMSE across folds (use neg_mean_squared_error and then sqrt)
neg_mse_scores = cross_val_score(best_nn, X_train, y_train, cv=kf, scoring="neg_mean_squared_error")
rmse_scores = np.sqrt(-neg_mse_scores)

# Print summary
```

```
print("Cross-Validation Results (5-Fold) - Neural Network")
print("----------------------------------")
print(f"R²:   mean = {r2_scores.mean():.3f}, std = {r2_scores.std():.3f}, folds = {np.round(r2_scores, 3)}")
print(f"RMSE: mean = {rmse_scores.mean():.2f}, std = {rmse_scores.std():.2f}, folds = {np.round(rmse_scores, 2)}")

# --- Plots ---
fig, ax = plt.subplots(1, 2, figsize=(12,5))

# Fold indices for x-axis
fold_idx = np.arange(1, len(r2_scores) + 1)

# R² plot
ax[0].plot(fold_idx, r2_scores, marker='o', linestyle='-')
ax[0].axhline(r2_scores.mean(), linestyle='--', label=f"Mean R² = {r2_scores.mean():.3f}")
ax[0].set_title("Cross-Validation R² (Fold-wise) - Neural Network")
ax[0].set_xlabel("Fold")
ax[0].set_ylabel("R²")
ax[0].legend()

# RMSE plot
ax[1].plot(fold_idx, rmse_scores, marker='o', linestyle='-')
ax[1].axhline(rmse_scores.mean(), linestyle='--', label=f"Mean RMSE = {rmse_scores.mean():.2f}")
ax[1].set_title("Cross-Validation RMSE (Fold-wise) - Neural Network")
ax[1].set_xlabel("Fold")
ax[1].set_ylabel("RMSE")
ax[1].legend()

plt.tight_layout()
plt.show()
```

```
WARNING:tensorflow:5 out of the last 25 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x0
000015C6F4232E0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.fu
nction repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define y
our @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), ple
ase refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  mo
re details.
WARNING:tensorflow:5 out of the last 11 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x0
000015C71894310> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.fu
nction repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define y
our @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), ple
ase refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  mo
re details.
Cross-Validation Results (5-Fold) - Neural Network
----------------------------------
R²:   mean = 0.653, std = 0.071, folds = [0.529 0.74  0.673 0.693 0.628]
RMSE: mean = 3586.76, std = 343.04, folds = [4153.13 3077.32 3499.88 3608.51 3594.96]
```





4.8.6 - Confidence / Prediction Interval

In [103]:
```
# Bootstrap settings
n_bootstraps = 100   # number of resampled models
alpha = 0.05         # 95% prediction interval

# Collect predictions for test set
all_preds = []

for i in range(n_bootstraps):
    # Resample training data
    X_resampled, y_resampled = resample(X_train, y_train, random_state=42+i)

    # Clone and fit best Neural Network on resampled data
    nn_clone = nn_random_search.best_estimator_
    nn_clone.fit(X_resampled, y_resampled, verbose=0)

    # Predict on test set
    preds = nn_clone.predict(X_test).flatten()
    all_preds.append(preds)
```

```
# Convert to numpy array
all_preds = np.array(all_preds)  # shape: (n_bootstraps, n_test_samples)

# Compute mean prediction + intervals
y_pred_mean = all_preds.mean(axis=0)
lower_bound = np.percentile(all_preds, 100 * alpha/2, axis=0)
upper_bound = np.percentile(all_preds, 100 * (1 - alpha/2), axis=0)

# Show example for first 10 test samples
print("\nPrediction Intervals (first 10 test samples):")
for i in range(10):
    print(f"Sample {i}: Pred = {y_pred_mean[i]:.2f}, "
          f"95% PI = [{lower_bound[i]:.2f}, {upper_bound[i]:.2f}]")
```

```
Prediction Intervals (first 10 test samples):
Sample 0: Pred = 29358.09, 95% PI = [27917.31, 31084.29]
Sample 1: Pred = 28257.20, 95% PI = [26482.40, 30198.55]
Sample 2: Pred = 29861.31, 95% PI = [28459.47, 31752.70]
Sample 3: Pred = 18760.66, 95% PI = [17489.80, 19836.06]
Sample 4: Pred = 27002.64, 95% PI = [25676.81, 28604.32]
Sample 5: Pred = 25785.01, 95% PI = [24596.01, 26853.88]
Sample 6: Pred = 25181.62, 95% PI = [23748.02, 26565.57]
Sample 7: Pred = 24764.85, 95% PI = [23669.87, 26296.98]
Sample 8: Pred = 27261.21, 95% PI = [26022.92, 28487.68]
Sample 9: Pred = 28713.13, 95% PI = [27319.94, 30402.23]
```

4.9 - Final Model Comparison

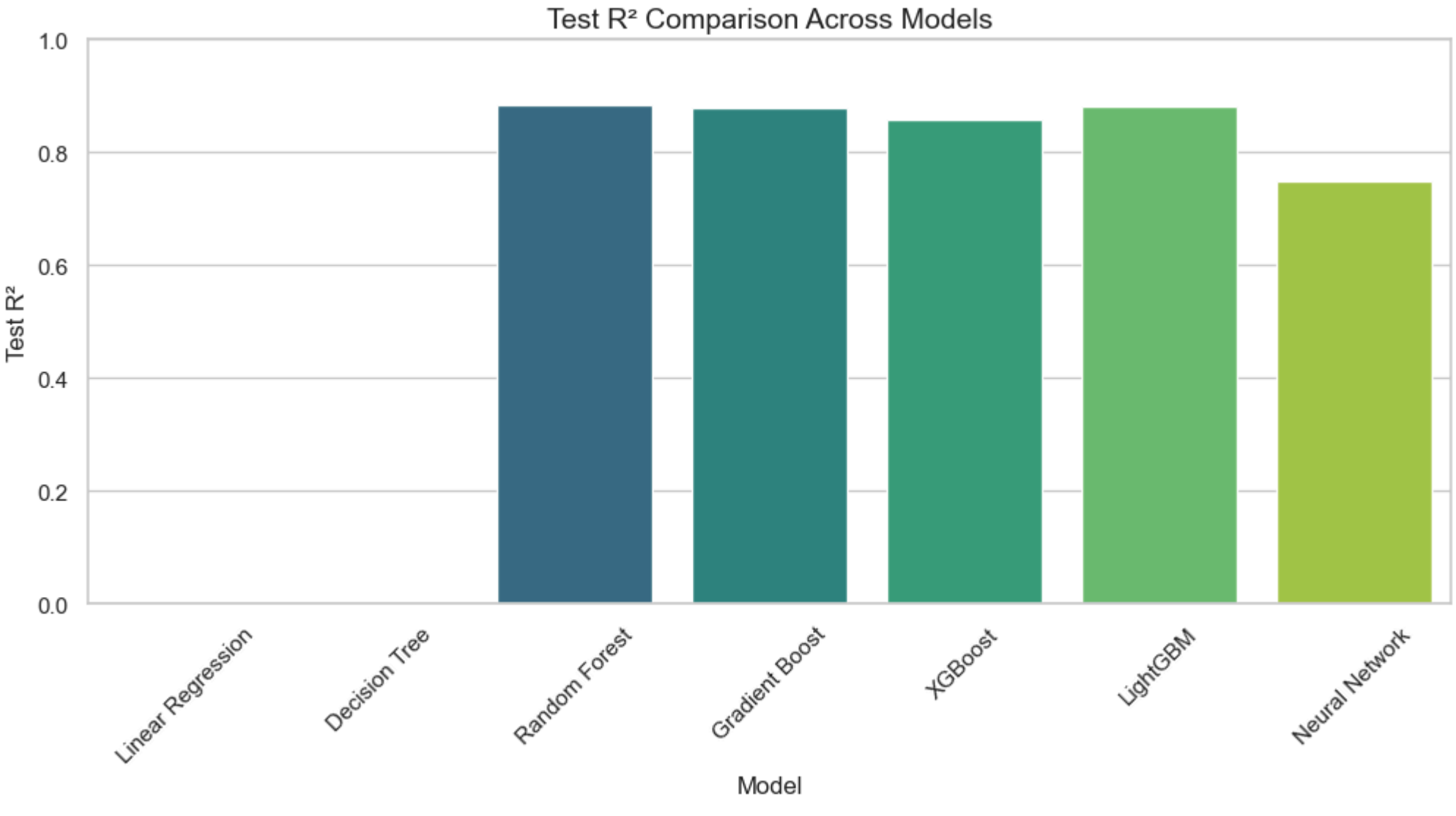
In [121]:
```
# Example names (replace if yours are different)
best_lin = lin_reg  # Linear Regression (already trained)
best_dt = grid_search.best_estimator_
best_rf = rf_random_search.best_estimator_
best_gbm = gbm_random_search.best_estimator_
best_xgb = xgb_random_search.best_estimator_
best_lgb = lgb_random_search.best_estimator_
best_nn = nn_random_search.best_estimator_  # Neural Network
```

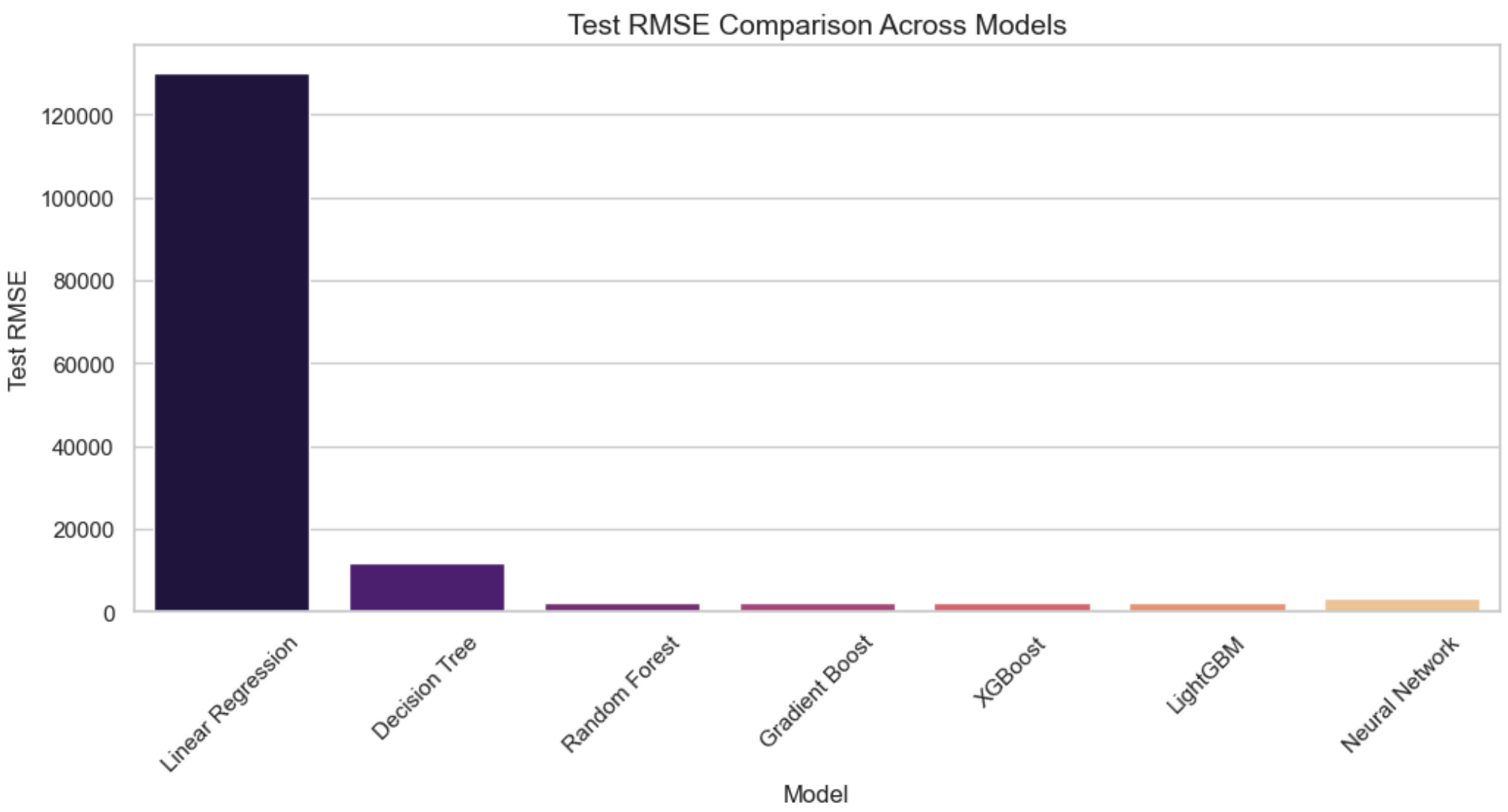
In [123]:
```
# Auto - Collect Metrics

# Create a dictionary to store results
results = {}

models = {
    'Linear Regression': best_lin,
    'Decision Tree': best_dt,
    'Random Forest': best_rf,
    'Gradient Boost': best_gbm,
    'XGBoost': best_xgb,
    'LightGBM': best_lgb,
    'Neural Network': best_nn
}

for name, model in models.items():
    # Check if it's Keras (NN) or scikit-learn
    if name == 'Neural Network':
        y_train_pred = model.predict(X_train).flatten()
        y_test_pred = model.predict(X_test).flatten()
    else:
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(X_test)

    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)
    train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
    test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))

    results[name] = {
        'Train R2': train_r2,
        'Test R2': test_r2,
        'Train RMSE': train_rmse,
        'Test RMSE': test_rmse
    }

# Convert to DataFrame
metrics_df = pd.DataFrame(results).T.reset_index().rename(columns={'index':'Model'})
metrics_df
```

Out[123]:

	Model	Train R2	Test R2	Train RMSE	Test RMSE
0	Linear Regression	-445.70	-396.00	130393.42	130112.58
1	Decision Tree	-2.85	-2.39	12099.48	12026.90
2	Random Forest	0.80	0.88	2792.87	2220.72
3	Gradient Boost	0.85	0.88	2363.95	2278.97
4	XGBoost	1.00	0.86	378.23	2456.80
5	LightGBM	0.87	0.88	2261.23	2239.31
6	Neural Network	0.66	0.75	3583.79	3275.36

In [125]:
```
# Plotting the results

# R² Comparison
plt.figure(figsize=(12,5))
sns.barplot(x='Model', y='Test R2', data=metrics_df, palette="viridis")
plt.title("Test R² Comparison Across Models", fontsize=14)
plt.ylabel("Test R²")
plt.xticks(rotation=45)
plt.ylim(0,1)
plt.show()

# RMSE Comparison
plt.figure(figsize=(12,5))
sns.barplot(x='Model', y='Test RMSE', data=metrics_df, palette="magma")
plt.title("Test RMSE Comparison Across Models", fontsize=14)
plt.ylabel("Test RMSE")
plt.xticks(rotation=45)
plt.show()


# Lower RMSE is better, higher R2 is better
metrics_df['Verdict'] = metrics_df.apply(
    lambda row: 'Best Candidate' if (row['Test R2']==metrics_df['Test R2'].max()) and (row['Test RMSE']==metrics_df['Test RMSE'].min()) else '
    axis=1
)

metrics_df
```



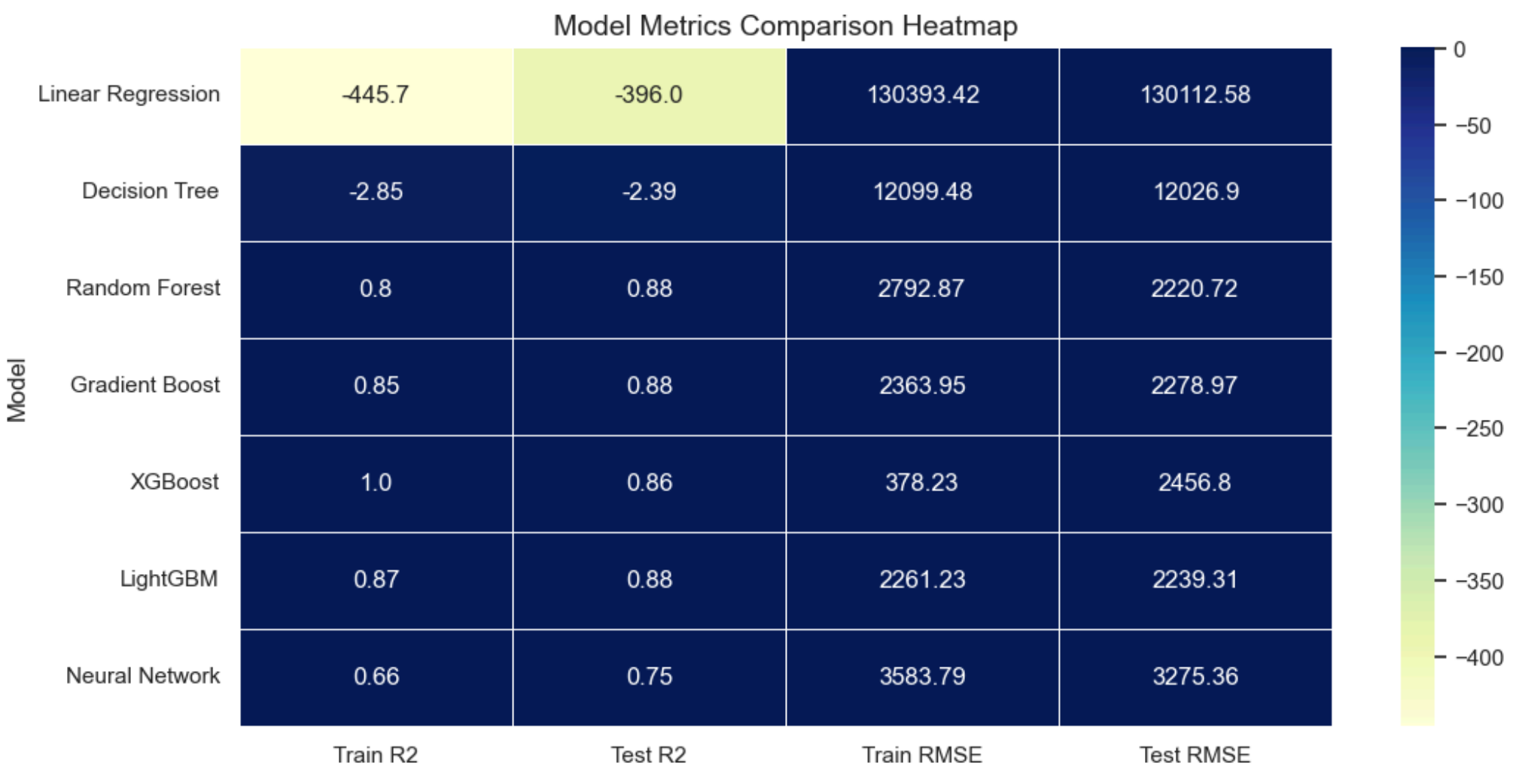


Out[125]:

	Model	Train R2	Test R2	Train RMSE	Test RMSE	Verdict
0	Linear Regression	-445.70	-396.00	130393.42	130112.58	
1	Decision Tree	-2.85	-2.39	12099.48	12026.90	
2	Random Forest	0.80	0.88	2792.87	2220.72	Best Candidate
3	Gradient Boost	0.85	0.88	2363.95	2278.97	
4	XGBoost	1.00	0.86	378.23	2456.80	
5	LightGBM	0.87	0.88	2261.23	2239.31	
6	Neural Network	0.66	0.75	3583.79	3275.36	

In [126]:
```
# Heatmap of R² and RMSE

# Select only numeric metrics
heatmap_data = metrics_df.set_index('Model')[['Train R2', 'Test R2', 'Train RMSE', 'Test RMSE']]

# Normalize RMSE for better visual comparison (optional)
# This keeps R² as-is and scales RMSE to 0-1 range for heatmap
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

heatmap_scaled = heatmap_data.copy()
heatmap_scaled[['Train RMSE','Test RMSE']] = scaler.fit_transform(heatmap_scaled[['Train RMSE','Test RMSE']])
```

In [128]:
```
# Plotting the heatmap

plt.figure(figsize=(12,6))
sns.heatmap(heatmap_scaled, annot=heatmap_data.round(2), cmap="YlGnBu", fmt='', linewidths=.5)
plt.title("Model Metrics Comparison Heatmap", fontsize=14)
plt.show()
```



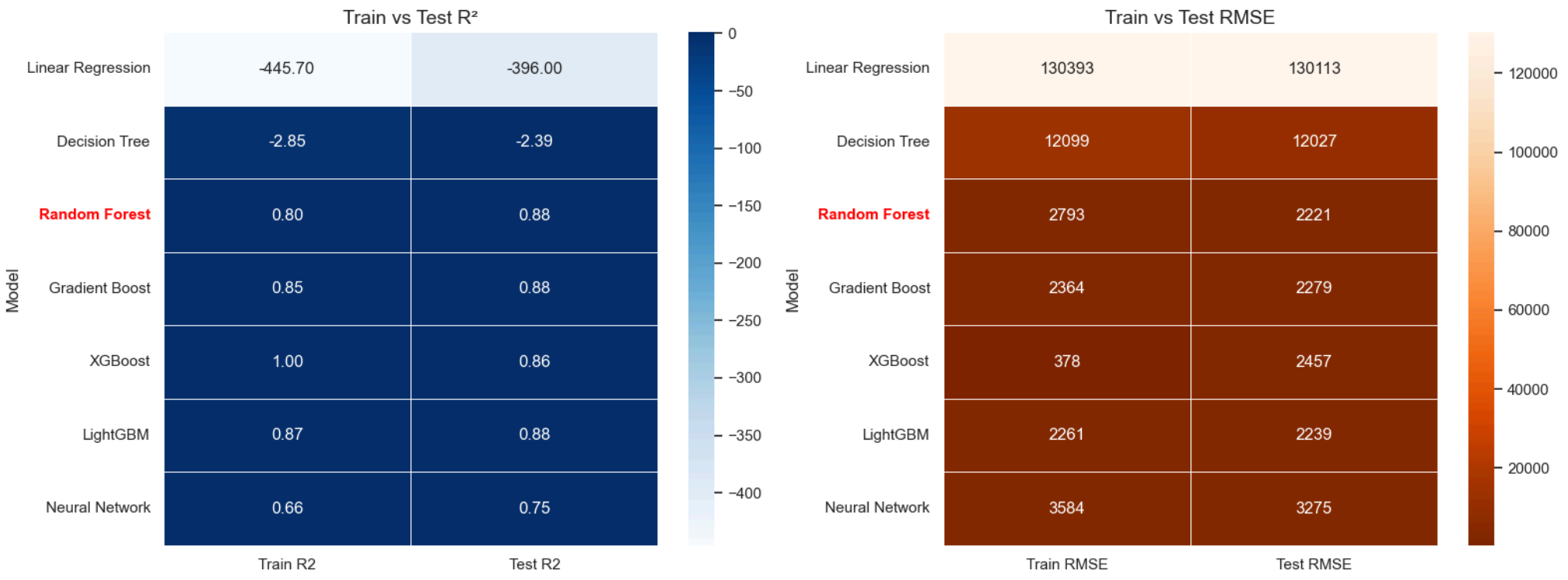
Best Model

In [ ]:
```python
# Step 1: Identify Best Model

# Best model based on highest Test R² and lowest Test RMSE
best_model_r2 = metrics_df.loc[metrics_df['Test R2'].idxmax(),'Model']
best_model_rmse = metrics_df.loc[metrics_df['Test RMSE'].idxmin(),'Model']

# Optional: pick model that satisfies both (if same)
best_model = best_model_r2 if best_model_r2 == best_model_rmse else best_model_r2
print(f"Highlighted Best Model: {best_model}")
```

```
Highlighted Best Model: Random Forest
```

In [130]:
```python
# Step 2: Separate R² and RMSE Data

# Set Model as index
metrics_df_plot = metrics_df.set_index('Model')

r2_df = metrics_df_plot[['Train R2','Test R2']]
rmse_df = metrics_df_plot[['Train RMSE','Test RMSE']]
```

In [131]:
```python
# Step 3: Plot R² and RMSE Side by Side

fig, axes = plt.subplots(1, 2, figsize=(16,6))

# --- R² Heatmap ---
sns.heatmap(r2_df, annot=True, fmt=".2f", cmap="Blues", linewidths=0.5, ax=axes[0])
axes[0].set_title("Train vs Test R²", fontsize=14)
for tick in axes[0].get_yticklabels():
    if tick.get_text() == best_model:
        tick.set_weight('bold')
        tick.set_color('red')

# --- RMSE Heatmap ---
sns.heatmap(rmse_df, annot=True, fmt=".0f", cmap="Oranges_r", linewidths=0.5, ax=axes[1])
axes[1].set_title("Train vs Test RMSE", fontsize=14)
for tick in axes[1].get_yticklabels():
    if tick.get_text() == best_model:
        tick.set_weight('bold')
        tick.set_color('red')

plt.tight_layout()
plt.show()
```



create a small Verdict Table that summarizes the best model with its key metrics and gives a short recommendation.

In [132]:
```
# Step 1: Extract Best Model Metrics

# Get metrics of the best model
best_metrics = metrics_df.loc[metrics_df['Model'] == best_model].copy()

# Add a short recommendation based on Test metrics
best_metrics['Recommendation'] = f"Use {best_model} for final predictions. High Test R² and low RMSE indicate good accuracy and generalization

best_metrics
```

Out[132]:

	Model	Train R2	Test R2	Train RMSE	Test RMSE	Verdict	Recommendation
2	Random Forest	0.80	0.88	2792.87	2220.72	Best Candidate	Use Random Forest for final predictions. High ...

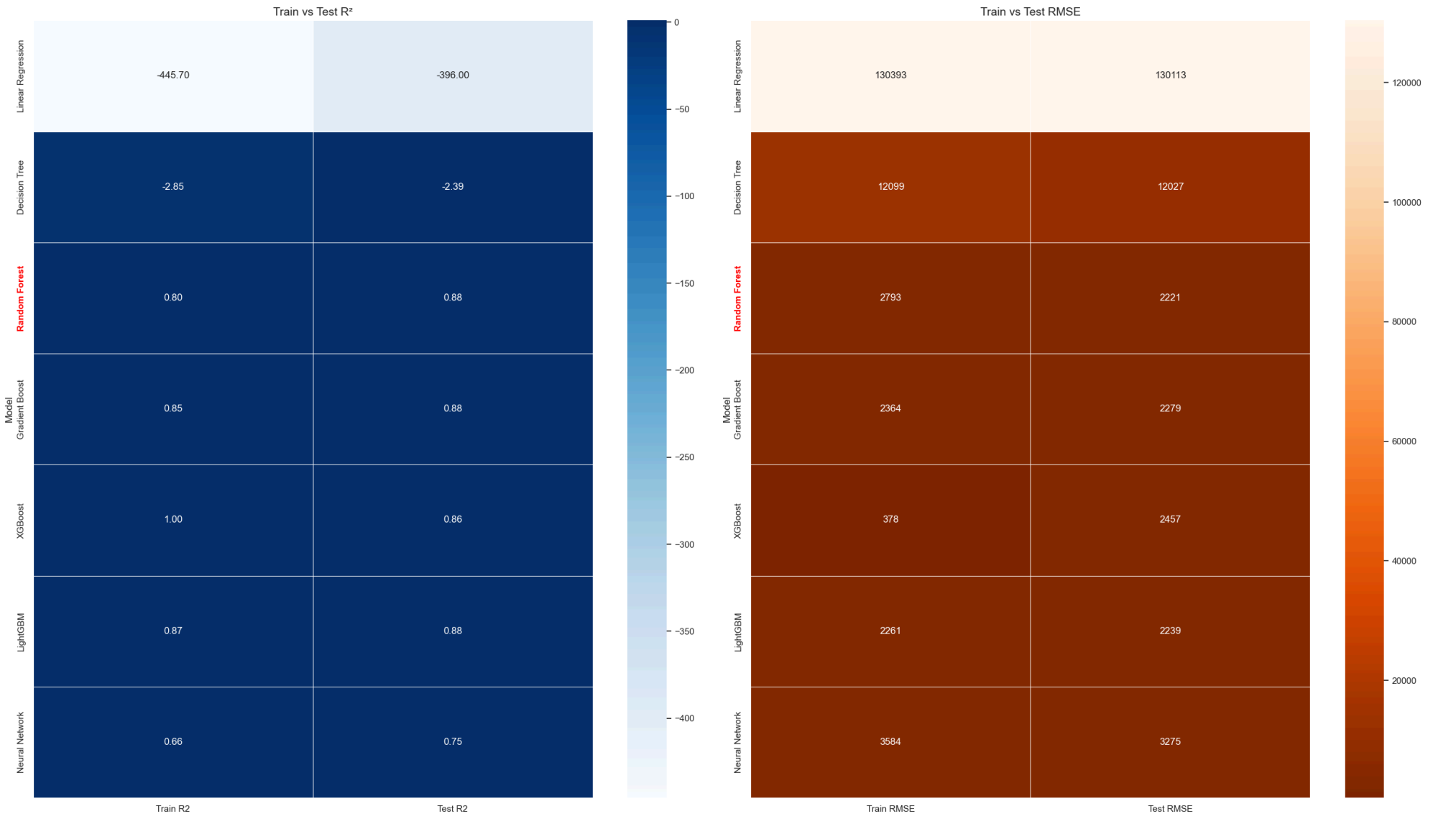
In [133]:
```
# Step 2: Display as a Clean Table

# Display only relevant columns
display_columns = ['Model', 'Test R2', 'Test RMSE', 'Recommendation']
display(best_metrics[display_columns])
```

	Model	Test R2	Test RMSE	Recommendation
2	Random Forest	0.88	2220.72	Use Random Forest for final predictions. High ...

Big Summary Pic

In [136]:
```
import matplotlib.gridspec as gridspec

# --- Identify best model ---
best_model_r2 = metrics_df.loc[metrics_df['Test R2'].idxmax(),'Model']
best_model_rmse = metrics_df.loc[metrics_df['Test RMSE'].idxmin(),'Model']
best_model = best_model_r2 if best_model_r2 == best_model_rmse else best_model_r2

# --- Prepare data ---
metrics_df_plot = metrics_df.set_index('Model')
r2_df = metrics_df_plot[['Train R2','Test R2']]
rmse_df = metrics_df_plot[['Train RMSE','Test RMSE']]

# Verdict info
best_metrics = metrics_df.loc[metrics_df['Model'] == best_model].copy()
best_metrics['Recommendation'] = f"Use {best_model} for final predictions. \nHigh Test R² and low RMSE indicate good accuracy and generalizati
display_columns = ['Model', 'Test R2', 'Test RMSE', 'Recommendation']
verdict_df = best_metrics[display_columns]

# --- Plot layout ---
fig = plt.figure(figsize=(25,20))
gs = gridspec.GridSpec(2, 2, height_ratios=[1, 0.4], width_ratios=[1,1])

# R² heatmap
ax0 = fig.add_subplot(gs[0,0])
sns.heatmap(r2_df, annot=True, fmt=".2f", cmap="Blues", linewidths=0.5, ax=ax0)
ax0.set_title("Train vs Test R²", fontsize=14)
for tick in ax0.get_yticklabels():
    if tick.get_text() == best_model:
        tick.set_weight('bold')
        tick.set_color('red')

# RMSE heatmap
ax1 = fig.add_subplot(gs[0,1])
sns.heatmap(rmse_df, annot=True, fmt=".0f", cmap="Oranges_r", linewidths=0.5, ax=ax1)
ax1.set_title("Train vs Test RMSE", fontsize=14)
for tick in ax1.get_yticklabels():
    if tick.get_text() == best_model:
        tick.set_weight('bold')
        tick.set_color('red')
```

```
# Verdict Table
ax2 = fig.add_subplot(gs[1,:])
ax2.axis('off')
table_data = verdict_df.values
col_labels = verdict_df.columns
table = ax2.table(cellText=table_data, colLabels=col_labels, cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(11)
table.scale(1,2)
ax2.set_title("Final Verdict on Best Model", fontsize=14, fontweight='bold', pad=20)

plt.tight_layout()
plt.show()
```



Final Verdict on Best Model

Model	Test R2	Test RMSE	Recommendation
Random Forest	0.884351129022682	2220.718509468573	Use Random Forest for final predictions. High Test R^2 and low RMSE indicate good accuracy and generalization.

Perfect! Let's do it step by step. First, I'll give you a **short professional guideline for interpreting your dashboard**, so it's ready for your report. Then we can move on to **deployment**.

Dashboard Interpretation Guidelines

1. **R^2 Heatmap (Left)**
   - Shows **Train vs Test R^2** for all models.
   - Higher values indicate better **explained variance**.
   - **Red-bold model** is the best performer on test set.
2. **RMSE Heatmap (Right)**
   - Shows **Train vs Test RMSE**.
   - Lower values indicate better **prediction accuracy**.
   - **Red-bold model** is the best performer on test set.
3. **Verdict Table (Bottom)**
   - Summarizes the **best model** according to Test R^2 and RMSE.
   - Includes **short recommendation** for which model to use for final predictions.
4. **Overfitting Check**
   - If Train metrics are much higher than Test metrics, model may be **overfitting**.
   - A balanced model will have **Train ≈ Test** values for both R^2 and RMSE.
5. **Portfolio Presentation Tip**

- Use **heatmap colors + bold red highlighting** to make the dashboard visually appealing.
- Reference the **Verdict Table** for the final recommendation.

Once you include this text in your portfolio, anyone reading it can **instantly understand model performance and your reasoning**.

If you're ready, we can **start the deployment part next**.

For deployment, do you want:

1. **Local web app using Streamlit/Gradio**?
2. **Cloud deployment (e.g., Hugging Face Spaces, Heroku)**?

This will determine the approach.

5. Deployment

5.1 - Save Your Trained Model

In [139]:
```
import joblib

# Save the best model (replace with your chosen model)
joblib.dump(best_rf, "best_model.pkl")
```

Out[139]:
```
['best_model.pkl']
```

```
from sklearn.pipeline import Pipeline
import joblib

# Example pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),  # your ColumnTransformer
    ('model', best_rf)
])

pipeline.fit(X_train, y_train)

# Save the pipeline
joblib.dump(pipeline, "best_pipeline.pkl")
```

In [142]:
```
print(best_rf.feature_names_in_)
```

```
['Age' 'Diabetes' 'Blood_Pressure_Problems' 'Any_Transplants'
 'Any_Chronic_Diseases' 'Height' 'Weight' 'Known_Allergies'
 'History_of_Cancer_in_Family' 'Number_of_Major_Surgeries' 'BMI'
 'BMI_Category_Normal' 'BMI_Category_Overweight' 'BMI_Category_Obese'
 'Age_Group_30-39' 'Age_Group_40-49' 'Age_Group_50-59' 'Age_Group_60+']
```

In [ ]: