Medical Insurance Cost Analysis

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
import requests
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore", category = UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning, message=".*use_inf_as_na.*")
from ipywidgets import interact,interactive,fixed,interact_manual
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from tqdm import tqdm
import requests
```

Importing Datasets

```
data= pd.read_csv("insurance.csv")
df = data
```

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

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

Data wrangling

```
missing_data = df.isnull()
missing_data
```

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	KnownAllergies	HistoryOfCancerInFamily	NumberOfMajorSurgeries	Pre
0	False	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	
981	False	False	False	False	False	False	False	False	False	False	
982	False	False	False	False	False	False	False	False	False	False	
983	False	False	False	False	False	False	False	False	False	False	
984	False	False	False	False	False	False	False	False	False	False	
985	False	False	False	False	False	False	False	False	False	False	

986 rows × 11 columns

```
for column in missing_data.columns.values.tolist():
    print(column)
    print(missing_data[column].value_counts())
    print("")
```

```
Age
Age
False    986
Name: count, dtype: int64

Diabetes
Diabetes
False    986
Name: count, dtype: int64

BloodPressureProblems
BloodPressureProblems
False    986
Name: count, dtype: int64

AnyTransplants
AnyTransplants
False    986
Name: count, dtype: int64

AnyChronicDiseases
AnyChronicDiseases
False    986
Name: count, dtype: int64

Height
Height
False    986
Name: count, dtype: int64

Weight
Weight
False    986
Name: count, dtype: int64

KnownAllergies
KnownAllergies
False    986
Name: count, dtype: int64

HistoryOfCancerInFamily
HistoryOfCancerInFamily
False    986
Name: count, dtype: int64

NumberOfMajorSurgeries
NumberOfMajorSurgeries
False    986
Name: count, dtype: int64

PremiumPrice
PremiumPrice
False    986
Name: count, dtype: int64
```

```
df.dtypes
```

	0
Age	int64
Diabetes	int64
BloodPressureProblems	int64
AnyTransplants	int64
AnyChronicDiseases	int64
Height	int64
Weight	int64
KnownAllergies	int64
HistoryOfCancerInFamily	int64
NumberOfMajorSurgeries	int64
PremiumPrice	int64

dtype: object

```
# Map 'Diabetes' column: yes -> 1, no -> 2
df['Diabetes'] = df['Diabetes'].map({'yes': 1, 'no': 2})
df[['PremiumPrice']] = np.round(df[['PremiumPrice']],2)
```

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

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

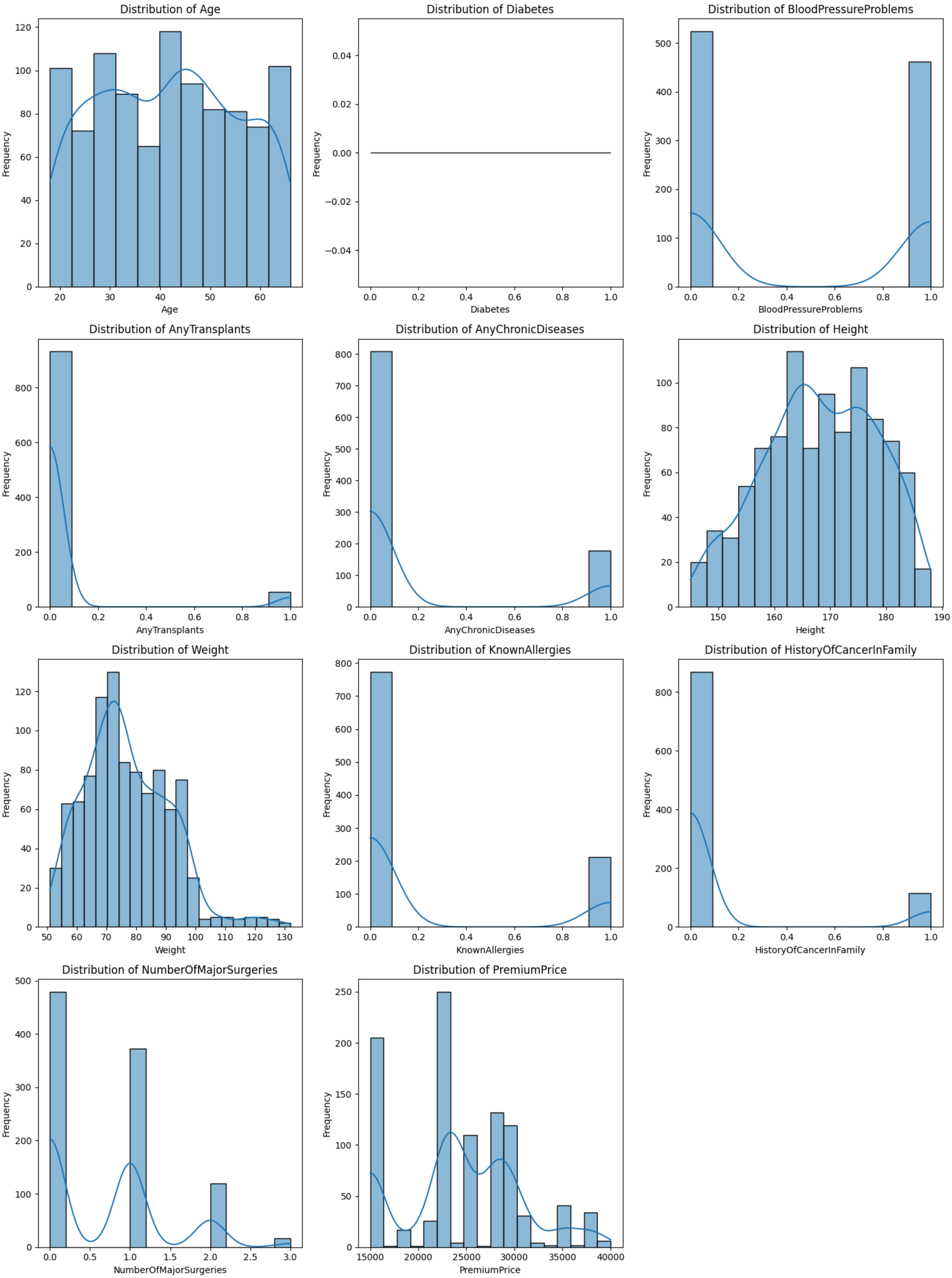
Exploratory Data Analysis (EDA)

Distribution analysis

```
df.head(), 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                 0 non-null      float64
 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: float64(1), int64(10)
memory usage: 84.9 KB
(   Age  Diabetes  BloodPressureProblems  AnyTransplants  AnyChronicDiseases  \
 0   45       NaN                      0               0                   0   
 1   60       NaN                      0               0                   0   
 2   36       NaN                      1               0                   0   
 3   52       NaN                      1               0                   1   
 4   38       NaN                      0               0                   1   

    Height  Weight  KnownAllergies  HistoryOfCancerInFamily  \
 0     155      57               0                        0   
 1     180      73               0                        0   
 2     158      59               0                        0   
 3     183      93               0                        0   
 4     166      88               0                        0   

    NumberOfMajorSurgeries  PremiumPrice  
 0                       0         25000  
 1                       0         29000  
 2                       1         23000  
 3                       2         28000  
 4                       1         23000  ,
 None)
```

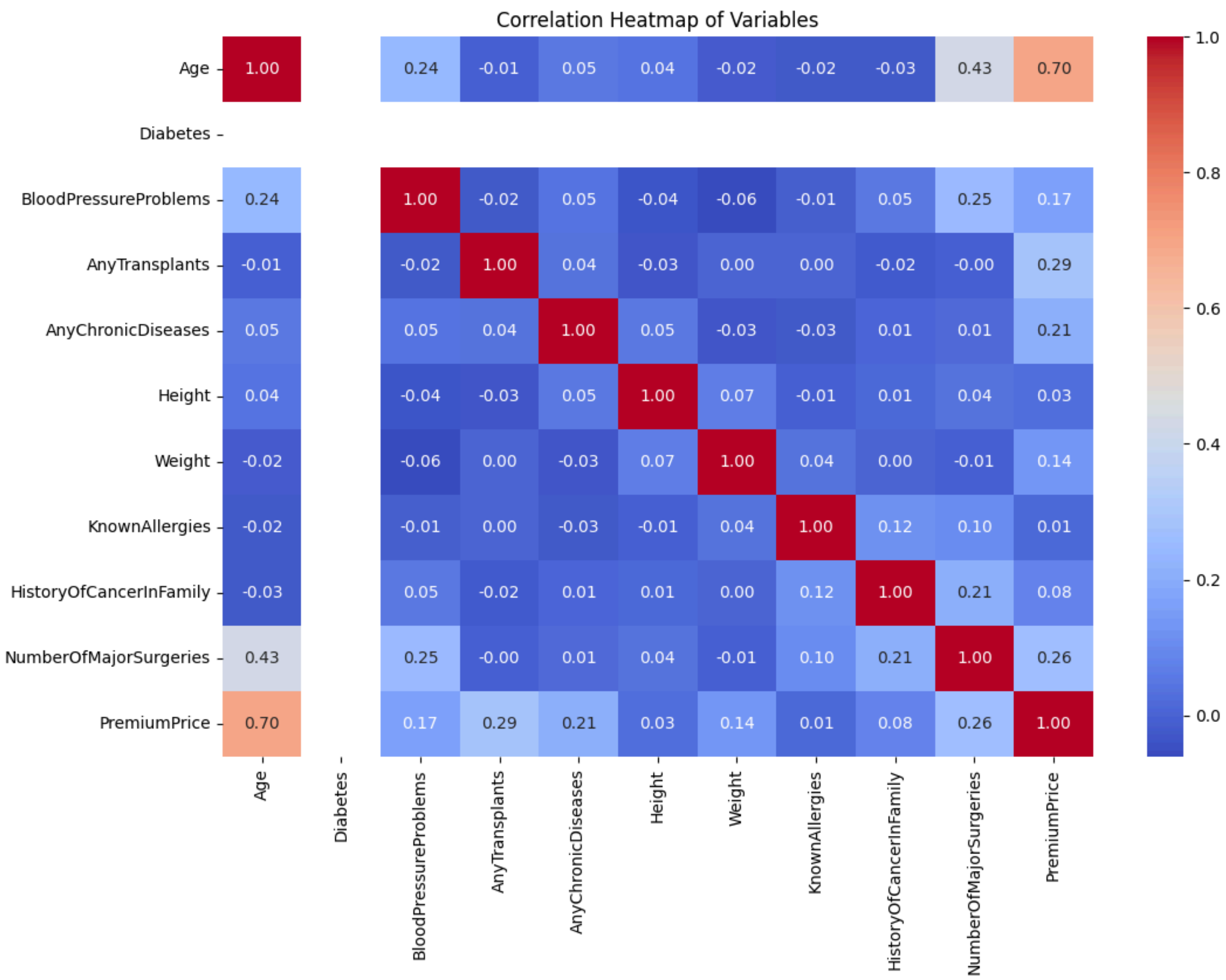
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting distribution for all numerical variables
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(15, 20))
axes = axes.flatten()

for i, column in enumerate(df.columns):
    sns.histplot(df[column], kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {column}')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Frequency')

# Remove empty subplots
for j in range(len(df.columns), len(axes)):
    fig.delaxes(axes[j])

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


Distribution of Age
Age
Frequency
Distribution of Diabetes
Diabetes
Frequency
Distribution of BloodPressureProblems
BloodPressureProblems
Frequency
Distribution of AnyTransplants
AnyTransplants
Frequency
Distribution of AnyChronicDiseases
AnyChronicDiseases
Frequency
Distribution of Height
Height
Frequency
Distribution of Weight
Weight
Frequency
Distribution of KnownAllergies
KnownAllergies
Frequency
Distribution of HistoryOfCancerInFamily
HistoryOfCancerInFamily
Frequency
Distribution of NumberOfMajorSurgeries
NumberOfMajorSurgeries
Frequency
Distribution of PremiumPrice
PremiumPrice
Frequency

The histograms above show the distributions of all variables in the dataset. Some variables like PremiumPrice and Age have broader distributions, while others, such as Diabetes and BloodPressureProblems, are binary.

Correlation Analysis

```
# Generate correlation matrix
correlation_matrix = df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
plt.title("Correlation Heatmap of Variables")
plt.show()
```



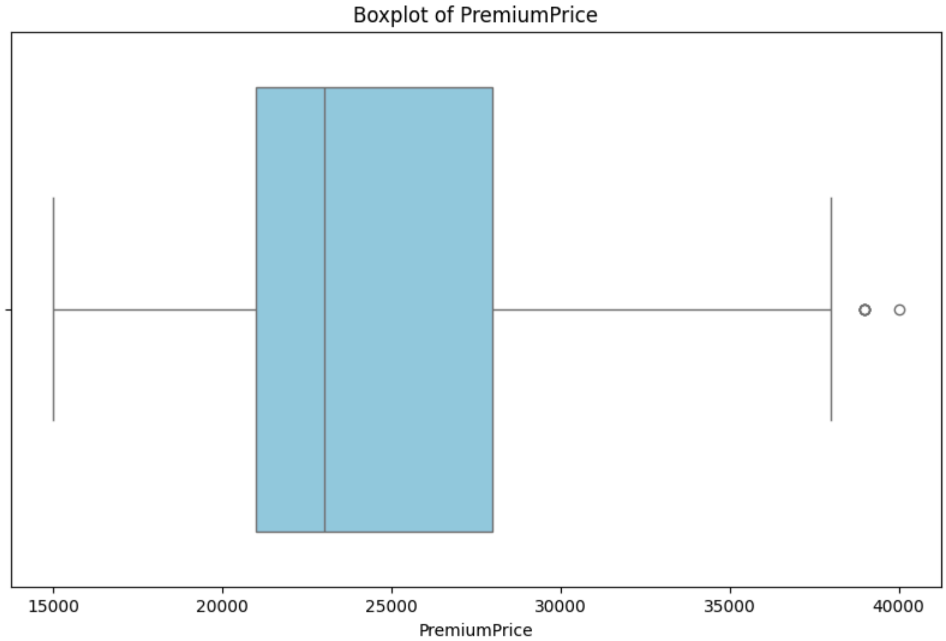
The heatmap displays the correlations between all variables. Notable points include:

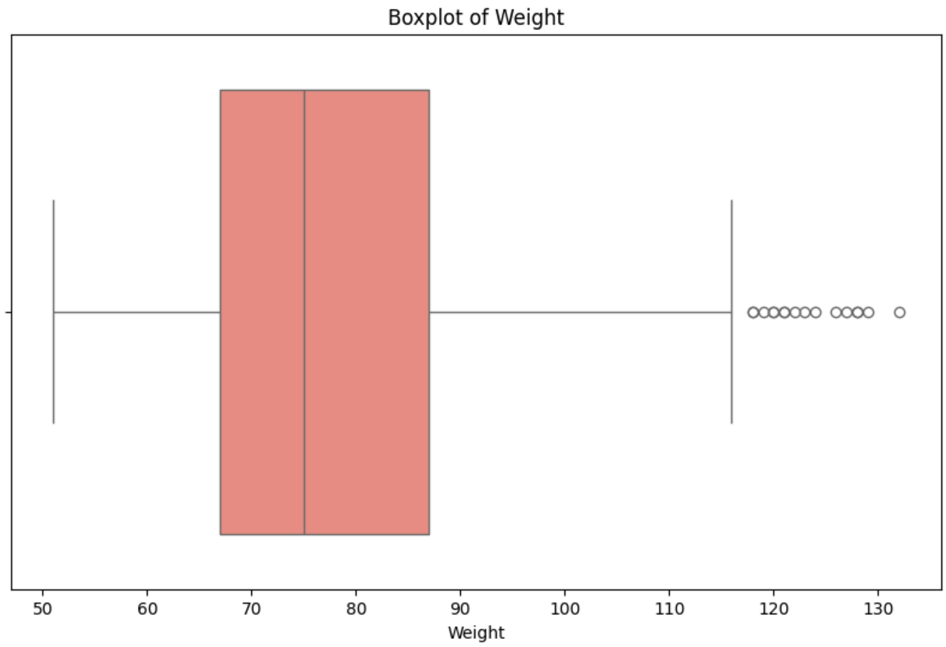
- PremiumPrice appears moderately correlated with NumberOfMajorSurgeries and Weight.
- Other variables show low or no correlation with PremiumPrice.

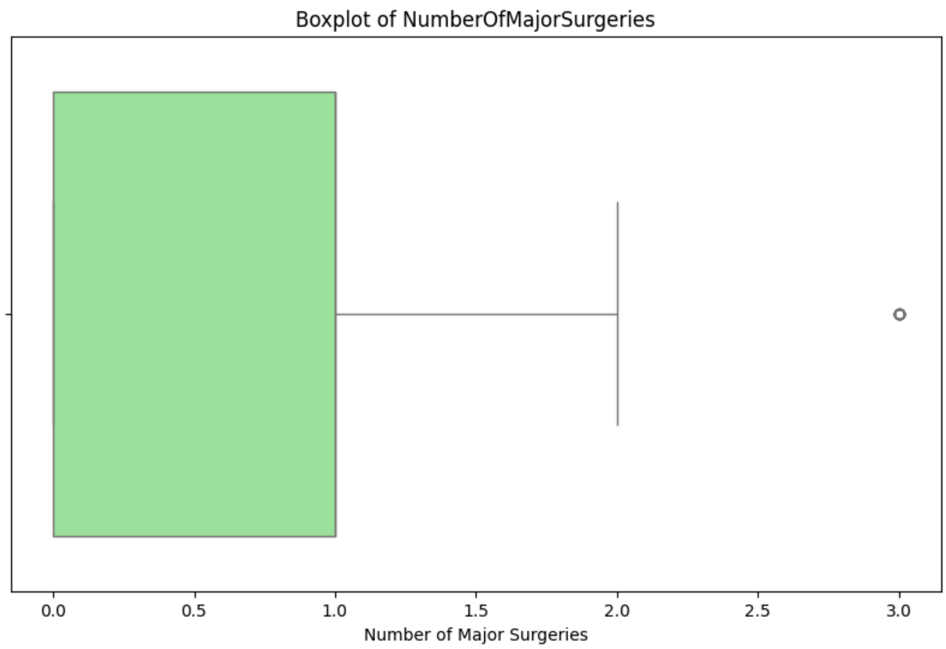
Outlier Detection

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

# Detecting outliers for PremiumPrice, Weight, and NumberOfMajorSurgeries
outliers_premium, lb_premium, ub_premium = detect_outliers_iqr(df, "PremiumPrice")
outliers_weight, lb_weight, ub_weight = detect_outliers_iqr(df, "Weight")
outliers_surgeries, lb_surgeries, ub_surgeries = detect_outliers_iqr(df, "NumberOfMajorSurgeries")

# Outputting the results
len(outliers_premium), lb_premium, ub_premium, \
len(outliers_weight), lb_weight, ub_weight, \
len(outliers_surgeries), lb_surgeries, ub_surgeries

# Boxplot for PremiumPrice
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x="PremiumPrice", color="skyblue")
plt.title("Boxplot of PremiumPrice")
```

```
plt.xlabel("PremiumPrice")
plt.show()

# Boxplot for Weight
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x="Weight", color="salmon")
plt.title("Boxplot of Weight")
plt.xlabel("Weight")
plt.show()

# Boxplot for NumberOfMajorSurgeries
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x="NumberOfMajorSurgeries", color="lightgreen")
plt.title("Boxplot of NumberOfMajorSurgeries")
plt.xlabel("Number of Major Surgeries")
plt.show()
```

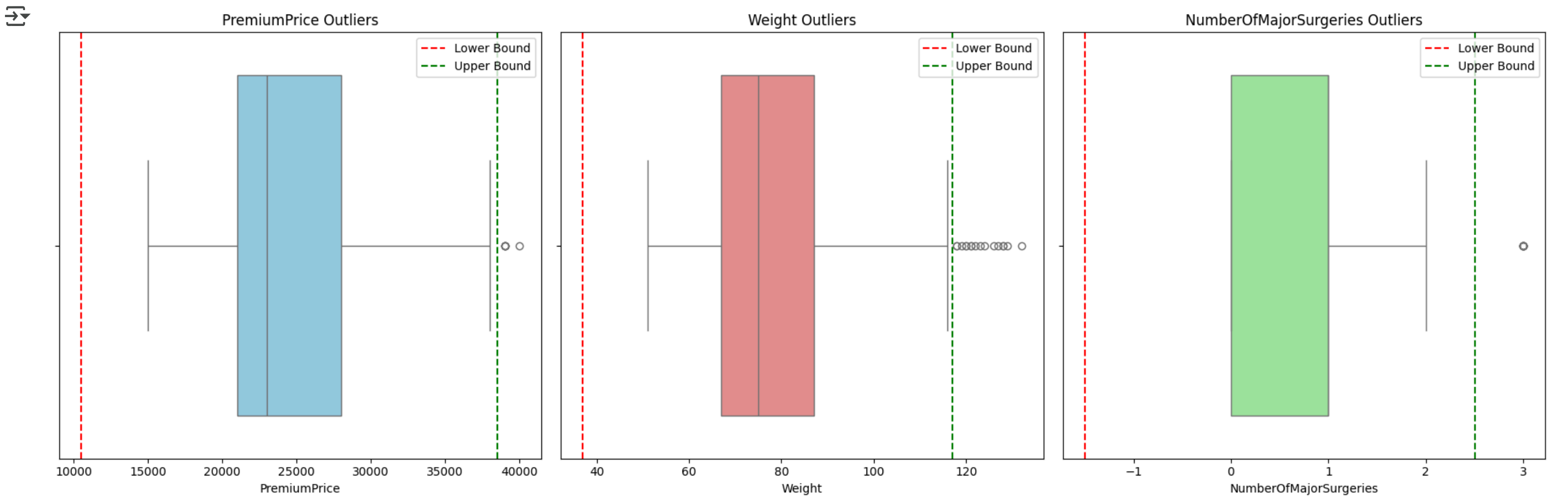







```
import matplotlib.pyplot as plt
import seaborn as sns
# Visualizing outliers for selected variables
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))

# PremiumPrice
# Ensure 'df' is defined by running the preceding cells that load and prepare the data.
sns.boxplot(data=df, x="PremiumPrice", ax=axes[0], color="skyblue")
axes[0].axvline(lb_premium, color="red", linestyle="--", label="Lower Bound")
axes[0].axvline(ub_premium, color="green", linestyle="--", label="Upper Bound")
```

```
axes[0].set_title("PremiumPrice Outliers")
axes[0].legend()

# Weight
sns.boxplot(data=df, x="Weight", ax=axes[1], color="lightcoral")
axes[1].axvline(lb_weight, color="red", linestyle="--", label="Lower Bound")
axes[1].axvline(ub_weight, color="green", linestyle="--", label="Upper Bound")
axes[1].set_title("Weight Outliers")
axes[1].legend()

# NumberOfMajorSurgeries
sns.boxplot(data=df, x="NumberOfMajorSurgeries", ax=axes[2], color="lightgreen")
axes[2].axvline(lb_surgeries, color="red", linestyle="--", label="Lower Bound")
axes[2].axvline(ub_surgeries, color="green", linestyle="--", label="Upper Bound")
axes[2].set_title("NumberOfMajorSurgeries Outliers")
axes[2].legend()

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



The boxplots above illustrate the detected outliers:

- PremiumPrice: Outliers lie significantly beyond the upper bound (38,500), indicating high premiums.
- Weight: Outliers include both underweight and overweight individuals beyond the range [37, 117].
- NumberOfMajorSurgeries: Outliers are above 2 surgeries, as expected.

Handling strategies might include:

- PremiumPrice: Winsorizing or capping at a threshold.
- Weight: Normalizing or segmenting into categories.
- NumberOfMajorSurgeries: Considering separate models for groups with many surgeries.

Hypothesis Testing

```
from scipy.stats import ttest_ind, chi2_contingency, f_oneway

# Example: T-test for smokers vs non-smokers (assuming data has such columns)
group1 = df[df['Diabetes'] == 0]['PremiumPrice']
group2 = df[df['Diabetes'] == 1]['PremiumPrice']
t_stat, p_value = ttest_ind(group1, group2)
print(f"T-test: t-statistic = {t_stat}, p-value = {p_value}")

# Example: ANOVA for NumberOfMajorSurgeries and PremiumPrice
anova_stat, anova_p = f_oneway(
    df[df['NumberOfMajorSurgeries'] == 0]['PremiumPrice'],
    df[df['NumberOfMajorSurgeries'] == 1]['PremiumPrice'],
    df[df['NumberOfMajorSurgeries'] >= 2]['PremiumPrice']
)
print(f"ANOVA: F-statistic = {anova_stat}, p-value = {anova_p}")

# Example: Chi-square test for HistoryOfCancerInFamily and AnyChronicDiseases
contingency_table = pd.crosstab(
    df['HistoryOfCancerInFamily'],
    df['AnyChronicDiseases']
)
chi2_stat, chi2_p, _, _ = chi2_contingency(contingency_table)
print(f"Chi-square test: Chi2-statistic = {chi2_stat}, p-value = {chi2_p}")
```

```
T-test: t-statistic = nan, p-value = nan
ANOVA: F-statistic = 39.2415277169692, p-value = 4.0158930660447635e-17
Chi-square test: Chi2-statistic = 0.02062393388215223, p-value = 0.8858081638149811
/usr/local/lib/python3.11/dist-packages/scipy/_lib/deprecation.py:234: SmallSampleWarning: One or more sample arguments is too small; all returned values will be N
  return f(*args, **kwargs)
```

```
from scipy.stats import f_oneway, ttest_ind

```

```
# ANOVA for PremiumPrice across NumberOfMajorSurgeries (0, 1, 2)
groups_surgeries = [df[df["NumberOfMajorSurgeries"] == group]["PremiumPrice"]
                    for group in df["NumberOfMajorSurgeries"].unique()]
anova_surgeries = f_oneway(*groups_surgeries)

# T-tests for PremiumPrice across AnyChronicDiseases (0 vs. 1)
group_chronic_0 = df[df["AnyChronicDiseases"] == 0]["PremiumPrice"]
group_chronic_1 = df[df["AnyChronicDiseases"] == 1]["PremiumPrice"]
ttest_chronic = ttest_ind(group_chronic_0, group_chronic_1, equal_var=False)

# T-tests for PremiumPrice across HistoryOfCancerInFamily (0 vs. 1)
group_cancer_0 = df[df["HistoryOfCancerInFamily"] == 0]["PremiumPrice"]
group_cancer_1 = df[df["HistoryOfCancerInFamily"] == 1]["PremiumPrice"]
ttest_cancer = ttest_ind(group_cancer_0, group_cancer_1, equal_var=False)

# Output results
anova_surgeries.pvalue, ttest_chronic.pvalue, ttest_cancer.pvalue
```

```
(np.float64(2.8711631377228097e-16),
 np.float64(1.7279736467737666e-13),
 np.float64(0.01982822652964323))
```

The results of the hypothesis tests are as follows:

- ANOVA for PremiumPrice across NumberOfMajorSurgeries:

  p-value: 2.87e-16

  Interpretation: The mean PremiumPrice differs significantly across groups (0, 1, 2 surgeries).

- T-test for PremiumPrice across AnyChronicDiseases (0 vs. 1):

  p-value: 1.73e-13

  Interpretation: There is a statistically significant difference in PremiumPrice between individuals with and without chronic diseases.

- T-test for PremiumPrice across HistoryOfCancerInFamily (0 vs. 1):

  p-value: 0.0198

  Interpretation: There is a statistically significant difference in PremiumPrice between those with and without a family history of cancer, though the effect may be less pronounced.

ML Modeling

Data Preprocessing

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
# Step 1: Handle Missing Values
# Check for missing values
print("Missing Values:\n", data.isnull().sum())

# If missing values exist, handle them
if data.isnull().sum().any():
    # Fill numerical features with median
    for col in data.select_dtypes(include=np.number).columns:
        data[col].fillna(data[col].median(), inplace=True)
    # Fill categorical features with mode
    for col in data.select_dtypes(include='object').columns:
        data[col].fillna(data[col].mode()[0], inplace=True)

# Step 2: Feature Engineering
# Example: Calculate BMI (Body Mass Index) if height and weight columns exist
if 'height' in data.columns and 'weight' in data.columns:
    data['BMI'] = data['weight'] / (data['height'] / 100) ** 2

# Split the dataset BEFORE identifying numerical/categorical features for the preprocessor
X = data.drop('PremiumPrice', axis=1)
y = data['PremiumPrice']

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

# Step 3: Scaling and Encoding
# Identify numerical and categorical columns AFTER splitting
numerical_features = X_train.select_dtypes(include=np.number).columns.tolist()
categorical_features = X_train.select_dtypes(include='object').columns.tolist()


# Preprocessing Pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features) # Added handle_unknown for robustness
    ]
)


# Apply the preprocessing pipeline
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)

print("Preprocessing Complete. Ready for Model Selection.")
```

```
Missing Values:
 Age                         0
Diabetes                    0
BloodPressureProblems       0
AnyTransplants              0
AnyChronicDiseases          0
Height                      0
Weight                      0
KnownAllergies              0
HistoryOfCancerInFamily     0
NumberOfMajorSurgeries      0
PremiumPrice                0
dtype: int64
Preprocessing Complete. Ready for Model Selection.
```

Model Selection

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor

# Step 1: Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_preds = linear_model.predict(X_test)

# Step 2: Decision Tree
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train, y_train)
tree_preds = tree_model.predict(X_test)

# Step 3: Random Forest
forest_model = RandomForestRegressor(random_state=42)
forest_model.fit(X_train, y_train)
forest_preds = forest_model.predict(X_test)

# Step 4: Gradient Boosting
gb_model = GradientBoostingRegressor(random_state=42)
gb_model.fit(X_train, y_train)
gb_preds = gb_model.predict(X_test)

# Step 5: Neural Network
nn_model = MLPRegressor(random_state=42, max_iter=500)
nn_model.fit(X_train, y_train)
nn_preds = nn_model.predict(X_test)

print("Model Training Complete. Ready for Evaluation.")
```

```
Model Training Complete. Ready for Evaluation.
```

Model Evaluation and Validation

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Function to evaluate models
def evaluate_model(name, y_test, predictions):
    mse = mean_squared_error(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)
    print(f"{name} Evaluation:")
    print(f" - MSE: {mse}")
    print(f" - MAE: {mae}")
    print(f" - R2: {r2}\n")

# Evaluate all models
evaluate_model("Linear Regression", y_test, linear_preds)
evaluate_model("Decision Tree", y_test, tree_preds)
evaluate_model("Random Forest", y_test, forest_preds)
evaluate_model("Gradient Boosting", y_test, gb_preds)
evaluate_model("Neural Network", y_test, nn_preds)
```

```
Linear Regression Evaluation:
 - MSE: 12221661.705858208
 - MAE: 2586.2253840681074
 - R2: 0.713394427027874

Decision Tree Evaluation:
 - MSE: 14893939.393939395
 - MAE: 1095.959595959596
 - R2: 0.6507278521900168

Random Forest Evaluation:
 - MSE: 5310708.080808081
 - MAE: 1034.7474747474748
 - R2: 0.8754605904647057

Gradient Boosting Evaluation:
 - MSE: 6166949.130591766
 - MAE: 1501.7274850951546
 - R2: 0.8553812049783704

Neural Network Evaluation:
 - MSE: 577411808.7458334
 - MAE: 23238.412330525465
 - R2: -12.540666258758183
```