1- Normalizasyon Veri setinizi normalleştirmek, farklı ölçeklerde olan özelliklerin modeller tarafından daha etkili bir şekilde işlenmesini sağlar. Bu, özellikle uzaklık tabanlı algoritmalar için önemlidir, çünkü özellikler arasındaki ölçek farklılıkları modelin performansını doğrudan etkileyebilir. En yaygın normalleştirme yöntemlerinden biri Min-Max normalleştirmesidir. Bu yöntem, tüm değerleri 0 ile 1 arasında bir ölçeğe dönüştürür. Burada, veri setinizdeki her bir özelliği (sütunu) aşağıdaki formülü kullanarak normalleştirelim:

Xnor= (X-Xmin)/(Xmax - Xmin)

Xnor normalleştirilmiş değeri, X verinin orjinal değeri, Xmin veri setindeki en düşük değeri, Xmax veri setindeki en büyük değeri

Bu işlemi Python'da pandas kütüphanesi kullanarak gerçekleştirebiliriz. Öncelikle, veri setinizi bir pandas DataFrame'e yükleyelim ve ardından Min-Max normalleştirmesini uygulayalım. Bunun için öncelikle veri setinizi gözden geçirelim ve normalleştirmeyi gerçekleştirelim.

In [8]:
```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Veri setini doğru ayırıcı ile yükleyelim
df = pd.read_csv('veri-seti.txt', header=None, sep='\t')
# Sütun adlarını belirleme
columns = ['Number of times pregnant', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', '
df.columns = columns

# Min-Max Normalizasyonu uygulayalım
df_normalized = (df - df.min()) / (df.max() - df.min())

# İlk beş satırı kontrol edelim
df_normalized.head()
```

Out[8]:

	Number of times pregnant	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	
0	0.352941	0.743719	0.590164	0.353535	0.000000	0.500745	0.234415	0.483
1	0.058824	0.427136	0.540984	0.292929	0.000000	0.396423	0.116567	0.166
2	0.470588	0.919598	0.524590	0.000000	0.000000	0.347243	0.253629	0.183
3	0.058824	0.447236	0.540984	0.232323	0.111111	0.418778	0.038002	0.000
4	0.000000	0.688442	0.327869	0.353535	0.198582	0.642325	0.943638	0.200

2-PCA ve LDA Uygulaması

PCA (Principal Component Analysis) ve LDA (Linear Discriminant Analysis) iki popüler boyut indirgeme tekniğidir. PCA, verinin varyansını maksimize ederek özellikleri dönüştürürken, LDA sınıf ayrımını maksimize eder. Her iki yöntem de genellikle yüksek boyutlu veri setlerini daha az boyutlu bir uzaya indirgemek için kullanılır. Ancak, LDA denetimli bir öğrenme yöntemidir ve sınıf etiketlerini kullanır, PCA ise denetimsiz bir yöntemdir.

In [4]:
```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import train_test_split

X = df_normalized.drop('Outcome', axis=1)
y = df_normalized['Outcome']

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

```
lda = LDA(n_components=1)
X_lda = lda.fit_transform(X, y)

print("PCA Açıklanan Varyans Oranı:", pca.explained_variance_ratio_)
print("LDA Açıklanan Varyans Oranı:", lda.explained_variance_ratio_ if hasattr(lda, 'explained_v
```

```
PCA Açıklanan Varyans Oranı: [0.31192249 0.21186663]
LDA Açıklanan Varyans Oranı: [1.]
```

PCA ve LDA sonuçlarına dayanarak, Insulin ve Glucose özelliklerinin bu veri seti için önemli olduğunu söyleyebiliriz.

3. Çoklu Doğrusal ve Multinominal Lojistik Regresyon

In [5]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# Veri setini eğitim ve test setlerine ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Lojistik Regresyon modeli eğitimi
log_reg = LogisticRegression(multi_class='multinomial', solver='lbfgs')
log_reg.fit(X_train, y_train)

# Modelin katsayılarını yazdırma
print("Katsayılar:\n", log_reg.coef_)
print("Kesme terimi:", log_reg.intercept_)

# Test verisi üzerinde performans değerlendirme
y_pred = log_reg.predict(X_test)
print("Doğruluk oranı:", accuracy_score(y_test, y_pred))
print("Sınıflandırma raporu:\n", classification_report(y_test, y_pred))
```

```
Katsayılar:
 [[ 0.4505206   2.65118931 -0.24999553  0.03039025 -0.06304149  2.16599472
   0.45399225  0.90638614]]
Kesme terimi: [-3.25065341]
Doğruluk oranı: 0.7359307359307359
Sınıflandırma raporu:
               precision    recall  f1-score   support

         0.0       0.78      0.82      0.80       151
         1.0       0.63      0.57      0.60        80

    accuracy                           0.74       231
   macro avg       0.71      0.70      0.70       231
weighted avg       0.73      0.74      0.73       231
```

Bu sonuçlar, modelin diyabet teşhisi konusunda iyi bir doğruluk oranına sahip olduğunu, ancak özellikle pozitif sınıf için (diyabet olanlar) duyarlılık ve kesinliğin daha da iyileştirilebileceğini göstermektedir. Yanlış negatif ve yanlış pozitif oranlarının azaltılması için modelin daha da optimize edilmesi gerekebilir.

4. Karar Ağacı Sınıflandırma

In [6]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Karar Ağacı modelini eğitme
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)

# Test veri seti üzerinde modelin performansını değerlendirme
y_pred_dt = decision_tree.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
class_report_dt = classification_report(y_test, y_pred_dt)
```

```
print("Doğruluk (Accuracy):", accuracy_dt)
print("Confusion Matrix:\n", conf_matrix_dt)
print("Sınıflandırma Raporu:\n", class_report_dt)
```

```
Doğruluk (Accuracy): 0.7012987012987013
Confusion Matrix:
 [[107  44]
 [ 25  55]]
Sınıflandırma Raporu:
              precision    recall  f1-score   support

         0.0       0.81      0.71      0.76       151
         1.0       0.56      0.69      0.61        80

    accuracy                           0.70       231
   macro avg       0.68      0.70      0.69       231
weighted avg       0.72      0.70      0.71       231
```

Karar Ağacı modeli, diyabet teşhisi konusunda kabul edilebilir bir doğruluk oranı sunmuş, ancak yanlış pozitif ve yanlış negatif oranları dikkate alındığında modelin performansının daha da iyileştirilebileceği görülmektedir. Diyabet olmayanları tespit etme konusunda daha yüksek bir kesinlik sunarken, diyabet olan hastaların tespitinde duyarlılığı artırılabilir.

5. Naive Bayes Sınıflandırma

In [7]:
```
from sklearn.naive_bayes import GaussianNB

# Naive Bayes modeli eğitimi
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)

# Test verisi üzerinde performans değerlendirme
y_pred_nb = naive_bayes.predict(X_test)
print("Doğruluk oranı:", accuracy_score(y_test, y_pred_nb))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
print("Sınıflandırma raporu:\n", classification_report(y_test, y_pred_nb))
```

```
Doğruluk oranı: 0.7445887445887446
Confusion Matrix:
 [[119  32]
 [ 27  53]]
Sınıflandırma raporu:
              precision    recall  f1-score   support

         0.0       0.82      0.79      0.80       151
         1.0       0.62      0.66      0.64        80

    accuracy                           0.74       231
   macro avg       0.72      0.73      0.72       231
weighted avg       0.75      0.74      0.75       231
```

Naive Bayes modeli, test verisi üzerinde %74.46 gibi bir doğruluk oranı ile kabul edilebilir bir performans göstermiştir. Model, diyabet olmayanları oldukça iyi bir kesinlikle tanırken, diyabet olan hastaların tespitinde duyarlılığı ve kesinliği daha da artırılabilir. Bu sonuçlar, modelin iyileştirilmesi için ek parametre ayarlamaları veya farklı sınıflandırma yöntemlerinin denenmesi gerektiğini göstermektedir.