

**KIRIKKALE ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ**

**YAPAY ZEKA DERSİ FİNAL ÖDEVİ RAPOR DOSYASI**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**YAPAY ZEKA DERSİ FİNAL ÖDEVİ**

**2020 / 2021 YILI GÜZ DÖNEMİ**

**Danışman**

**Dr. Öğr. Üyesi ENES AYAN**

**Projeyi Hazırlayan**

**Serkan BAYKAL 160255063**

**Kırıkkale, 2021**

## **KODLAR**

#Yapay Zeka Ödevi Serkan Baykal

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

import os

#Dosya Yolu

cwd = os.getcwd()

#Veri Yolu

readCsv = cwd + "/data.csv"

#Verileri Oku

df = pd.read\_csv(readCsv)

#İlk 5 Satır

df.head()

#1- Veri setinde hastalıklı ve sağlam sayılarını sütun grafiği kullanarak çizdirin.

df.target.value\_counts()

sns.countplot(x="target", data=df, palette="bwr")

plt.show()

countNoDisease = len(df[df.target == 0])

countHaveDisease = len(df[df.target == 1])

print("Percentage of Patients Haven't Heart Disease: {:.2f}%".format((countNoDisease / (len(df.target))\*100)))

print("Percentage of Patients Have Heart Disease: {:.2f}%".format((countHaveDisease / (len(df.target))\*100)))

#2- Cinsiyete göre hasta ve sağlıklı hasta sayıları sütun grafiği ile ifade ediniz.

pd.crosstab(df.sex,df.target).plot(kind="bar",figsize=(15,6),color=['red','green' ])

plt.title('Heart Disease Frequency for Sex')

plt.xlabel('0 = Female, 1 = Male')

plt.xticks(rotation=0)

plt.legend(["Haven't Disease", "Have Disease"])

plt.ylabel('Frequency')

plt.show()

#3-Veri setindekilerin yaş dağılımını gösteren bir sütun grafiği çizdirin.

sns.countplot(x='age', data=df, palette="mako\_r")

plt.xlabel("Age")

plt.show()

#4-Veri setinde hasta olanların yaş dağılımını gösteren bir sütun grafiği çizdirin.

pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6))

plt.title('Heart Disease Frequency for Ages')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.show()

a = pd.get\_dummies(df['cp'], prefix = "cp")

b = pd.get\_dummies(df['thal'], prefix = "thal")

c = pd.get\_dummies(df['slope'], prefix = "slope")

frames = [df, a, b, c]

df = pd.concat(frames, axis = 1)

df.head()

df = df.drop(columns = ['cp', 'thal', 'slope'])

y = df.target.values

x\_data = df.drop(['target'], axis = 1)

x = (x\_data - np.min(x\_data)) / (np.max(x\_data) - np.min(x\_data)).values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size = 0.2,random\_state=0)

#Matris

x\_train = x\_train.T

y\_train = y\_train.T

x\_test = x\_test.T

y\_test = y\_test.T

#Oluşturma

def initialize(dimension):

    weight = np.full((dimension,1),0.01)

    bias = 0.0

    return weight,bias

def sigmoid(z):

    y\_head = 1/(1+ np.exp(-z))

    return y\_head

def forwardBackward(weight,bias,x\_train,y\_train):

    #Forward

    y\_head = sigmoid(np.dot(weight.T,x\_train) + bias)

    loss = -(y\_train\*np.log(y\_head) + (1-y\_train)\*np.log(1-y\_head))

    cost = np.sum(loss) / x\_train.shape[1]

    #Backward

    derivative\_weight = np.dot(x\_train,((y\_head-y\_train).T))/x\_train.shape[1]

    derivative\_bias = np.sum(y\_head-y\_train)/x\_train.shape[1]

    gradients = {"Derivative Weight" : derivative\_weight, "Derivative Bias" : derivative\_bias}

    return cost,gradients

def update(weight,bias,x\_train,y\_train,learningRate,iteration) :

    costList = []

    index = []

    #Verileri Güncelle

    for i in range(iteration):

        cost,gradients = forwardBackward(weight,bias,x\_train,y\_train)

        weight = weight - learningRate \* gradients["Derivative Weight"]

        bias = bias - learningRate \* gradients["Derivative Bias"]

        costList.append(cost)

        index.append(i)

    parameters = {"weight": weight,"bias": bias}

    plt.plot(index,costList)

    plt.xlabel("Number of Iteration")

    plt.ylabel("Cost")

    plt.show()

    return parameters, gradients

def predict(weight,bias,x\_test):

    z = np.dot(weight.T,x\_test) + bias

    y\_head = sigmoid(z)

    y\_prediction = np.zeros((1,x\_test.shape[1]))

    for i in range(y\_head.shape[1]):

        if y\_head[0,i] <= 0.5:

            y\_prediction[0,i] = 0

        else:

            y\_prediction[0,i] = 1

    return y\_prediction

#7-Logistic Regresyon kullanarak veri setini sınıflandırın

def logistic\_regression(x\_train,y\_train,x\_test,y\_test,learningRate,iteration):

    dimension = x\_train.shape[0]

    weight,bias = initialize(dimension)

    parameters, gradients = update(weight,bias,x\_train,y\_train,learningRate,iteration)

    y\_prediction = predict(parameters["weight"],parameters["bias"],x\_test)

    print("Manuel Test Accuracy: {:.2f}%".format((100 - np.mean(np.abs(y\_prediction - y\_test))\*100)))

logistic\_regression(x\_train,y\_train,x\_test,y\_test,1,100)

accuracies = {}

lr = LogisticRegression()

lr.fit(x\_train.T,y\_train.T)

acc = lr.score(x\_test.T,y\_test.T)\*100

accuracies['Logistic Regression'] = acc

print("Test Accuracy {:.2f}%".format(acc))

# 8- K-NN kullanarak veri setini sınıflandırın

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors = 2)  # n\_neighbors means k

knn.fit(x\_train.T, y\_train.T)

prediction = knn.predict(x\_test.T)

print("KNN Score: {:.2f}%".format(knn.score(x\_test.T, y\_test.T)\*100))

#Find "k" Value

scoreList = []

for i in range(1,20):

    knn2 = KNeighborsClassifier(n\_neighbors = i)  # n\_neighbors means k

    knn2.fit(x\_train.T, y\_train.T)

    scoreList.append(knn2.score(x\_test.T, y\_test.T))

plt.plot(range(1,20), scoreList)

plt.xticks(np.arange(1,20,1))

plt.xlabel("K value")

plt.ylabel("Score")

plt.show()

acc = max(scoreList)\*100

accuracies['KNN'] = acc

print("Maximum KNN Score: {:.2f}%".format(acc))

#9- Naive Bayes kullanarak veri setini sınıflandırın

from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb.fit(x\_train.T, y\_train.T)

acc = nb.score(x\_test.T,y\_test.T)\*100

accuracies['Naive Bayes'] = acc

print("Accuracy of Naive Bayes: {:.2f}%".format(acc))

#10- Karar ağaçları kullanarak veri setini sınıflandırın

from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()

dtc.fit(x\_train.T, y\_train.T)

acc = dtc.score(x\_test.T, y\_test.T)\*100

accuracies['Decision Tree'] = acc

print("Decision Tree Test Accuracy {:.2f}%".format(acc))

from sklearn.metrics import confusion\_matrix

#Keras Kullanımı İçin, Tensorflow Kütüphanesi Gerekli

from keras.models import Sequential

from keras.layers import Dense

#Modelleri Karşılaştırma

colors = ["red", "green", "orange", "magenta","turquoise","black"]

sns.set\_style("whitegrid")

plt.figure(figsize=(16,5))

plt.yticks(np.arange(0,100,10))

plt.ylabel("Accuracy %")

plt.xlabel("Algorithms")

sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()), palette=colors)

plt.show()

#Karmaşıklık Matrisi

y\_head\_lr = lr.predict(x\_test.T)

knn3 = KNeighborsClassifier(n\_neighbors = 3)

knn3.fit(x\_train.T, y\_train.T)

y\_head\_knn = knn3.predict(x\_test.T)

y\_head\_nb = nb.predict(x\_test.T)

y\_head\_dtc = dtc.predict(x\_test.T)

cm\_lr = confusion\_matrix(y\_test,y\_head\_lr)

cm\_knn = confusion\_matrix(y\_test,y\_head\_knn)

cm\_nb = confusion\_matrix(y\_test,y\_head\_nb)

cm\_dtc = confusion\_matrix(y\_test,y\_head\_dtc)

plt.figure(figsize=(24,12))

plt.suptitle("Confusion Matrixes",fontsize=24)

plt.subplots\_adjust(wspace = 0.4, hspace= 0.4)

plt.subplot(2,3,1)

plt.title("Logistic Regression Confusion Matrix")

sns.heatmap(cm\_lr,annot=True,cmap="Blues",fmt="d",cbar=False, annot\_kws={"size": 24})

plt.subplot(2,3,2)

plt.title("K Nearest Neighbors Confusion Matrix")

sns.heatmap(cm\_knn,annot=True,cmap="Blues",fmt="d",cbar=False, annot\_kws={"size": 24})

plt.subplot(2,3,4)

plt.title("Naive Bayes Confusion Matrix")

sns.heatmap(cm\_nb,annot=True,cmap="Blues",fmt="d",cbar=False, annot\_kws={"size": 24})

plt.subplot(2,3,5)

plt.title("Decision Tree Classifier Confusion Matrix")

sns.heatmap(cm\_dtc,annot=True,cmap="Blues",fmt="d",cbar=False, annot\_kws={"size": 24})

plt.show()

#11- Yapay Sinir Ağları kullanarak veri setini sınıflandırın

plt.subplots(figsize=(12,8))

sns.heatmap(df.corr(),robust=True,annot=True)

#Karışık Verileri Tespit Etme

plt.subplots(figsize=(15,6))

df.boxplot(patch\_artist=True, sym="k.")

X = df.iloc[:, 1:11].values

y = df.iloc[:, -2].values

#Ölçeklendirme

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

sc\_y = StandardScaler()

#X = np.array(X).reshape(-1,1)

X = sc\_X.fit\_transform(X)

#y = np.array(y).reshape(-1,1)

y = sc\_y.fit\_transform(y)

#Eğitim ve Test Setleri Oluşturma

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)

from sklearn.preprocessing import Imputer

imputer= Imputer(strategy='mean')

imputer = imputer.fit(X[:,4:11])

X[:,4:11]= imputer.transform(X[:,4:11])

#Yapay Sinir Ağı

classifier = Sequential()

classifier.add(Dense(output\_dim = 6, init = 'uniform', activation = 'relu', input\_dim = 10))

classifier.add(Dense(output\_dim = 6, init = 'uniform', activation = 'relu'))

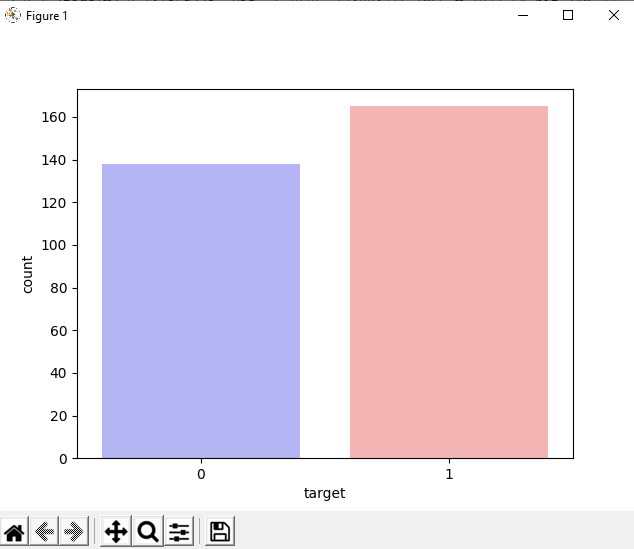
classifier.add(Dense(output\_dim = 1, init = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

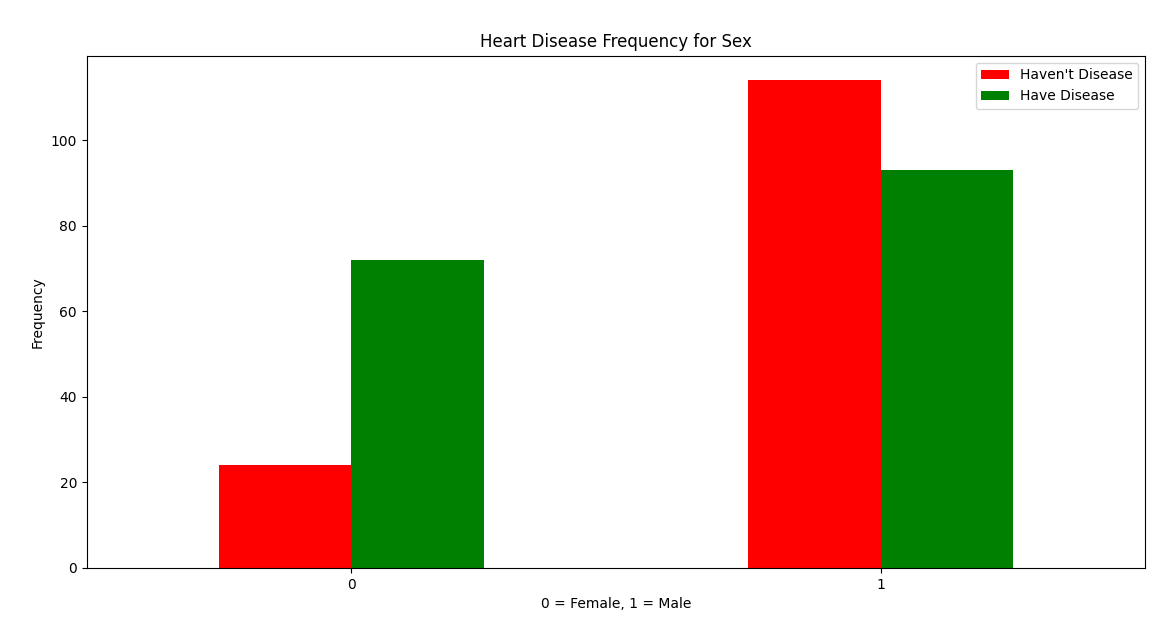
#Fit

classifier.fit(X\_train, y\_train, batch\_size = 10, nb\_epoch = 100)

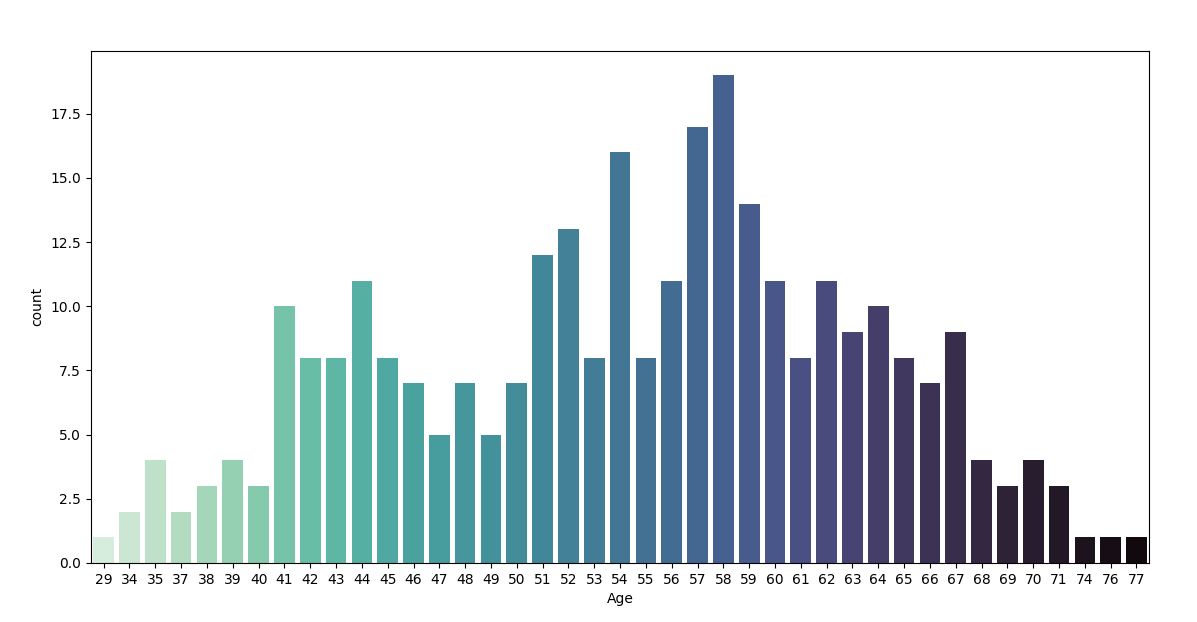
## **ÇIKTILAR**



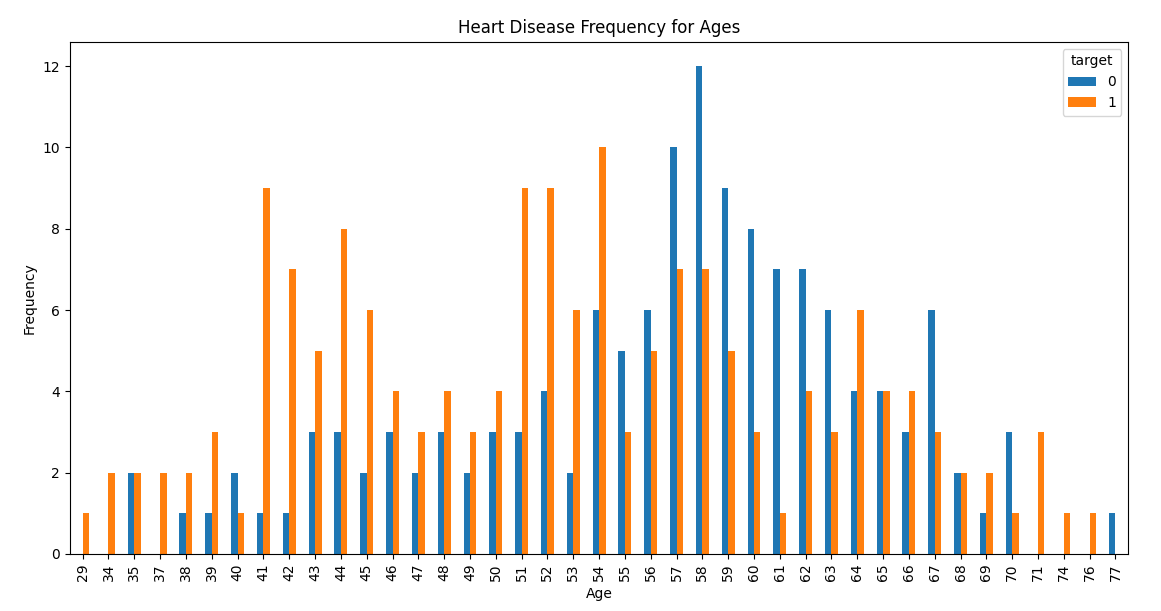
(Hastalıklı ve Sağlam Sayısı)



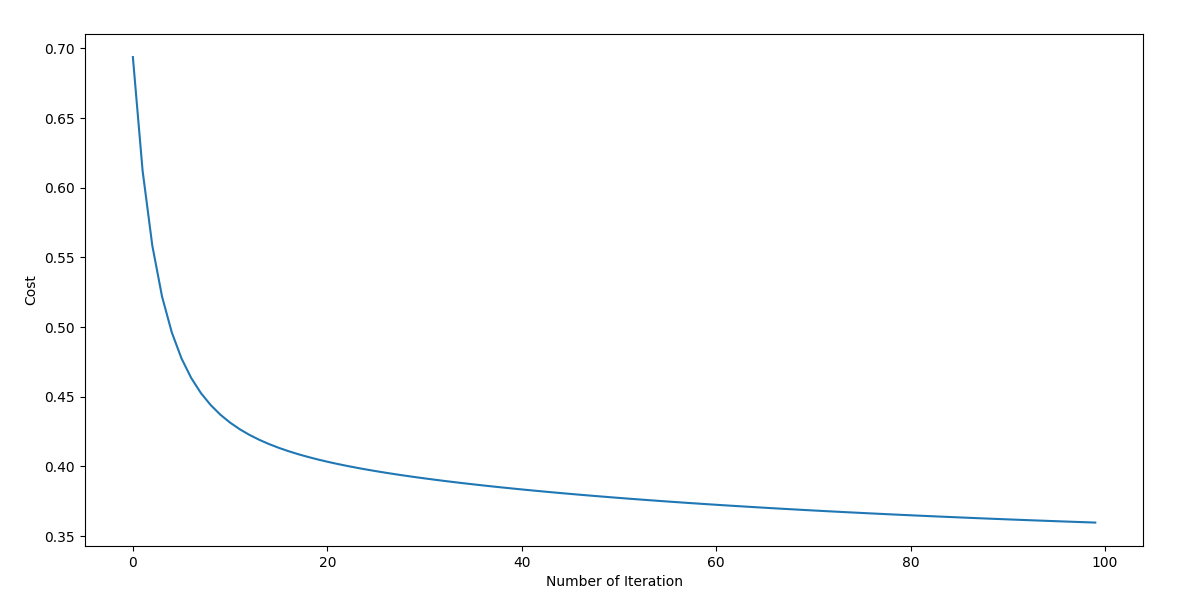
(Cinsiyete Göre Hastalıklı ve Sağlam Sayıları)



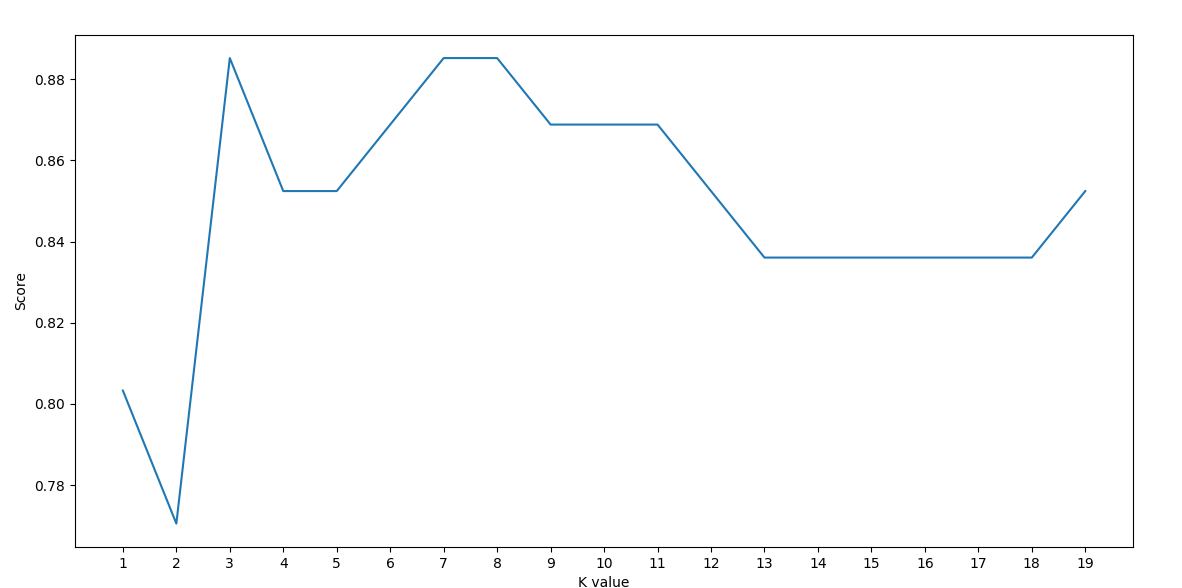
(Yaş Dağılımını Gösteren Grafik)



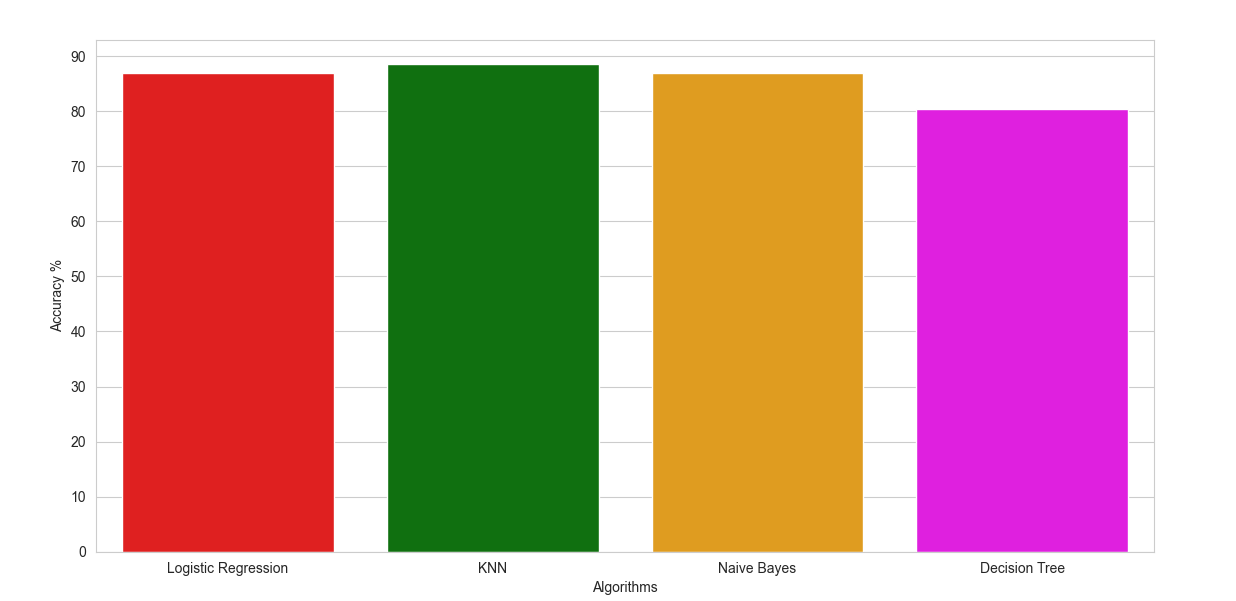
(Hasta Olanların Yaş Dağılımı)



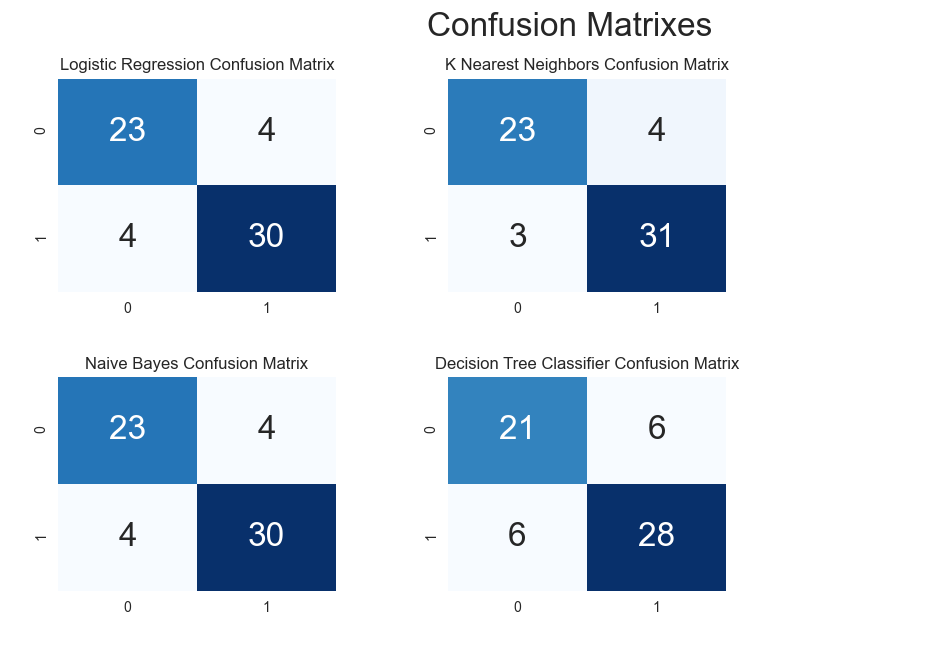
(Maliyet / İterasyon)



(Skor ve K Değeri)



(Algoritmaların Doğruluk Değerleri)



(Algoritmaların Karmaşıklık Matrisleri)

## **SONUÇ**

Algoritmaları incelediğimiz zaman en yüksek doğruluk değerine sahip olan algoritmayı “Algoritmaların Doğruluk Değerleri” adlı resimde görebiliriz. Bu resimdeki grafikten de anlaşılacağı gibi en iyi doğruluk değerini K-NN algoritması vermiştir. Fakat bu algoritmaya en başarılı diyebilmemiz için sadece bu değere bakarak yorum yapmamız doğru olmayacaktır. İşlenecek veri setlerine göre sonuç değişecektir. Aynı zamanda hesaba katılması gereken birkaç adet faktör daha göz önünde bulundurulmalıdır.

Bu veri seti üzerinden bir sonuç çıkarmak ve yorum yapmak gerekirse;

En başarısız algoritmanın Karar Ağaçları (Decision Tree) olduğunu söyleyebiliriz.

**Karar Ağaçları**

* Doğruya Doğru: 28
* Yanlışa Yanlış: 21
* Doğruya Yanlış: 6
* Yanlışa Doğru: 6

Hata matrisinden bu verileri okuduğumuzda diğer algoritmalara göre çok daha kötü bir sonuç ortaya çıkardığını görüyoruz.

En başarılı algoritmayı bulmak istediğimiz zaman diğerlerinin birbiriyle hemen hemen aynı sonuçları verdiğini görüyoruz. Naive Bayes ve Logistic Regression aynı sonucu üretmiştir. K-NN ise bu iki diğer algoritmadan farklı olarak, olumlu yönde 1 adet yanlışa doğru demek hatasına düşmemiştir ve diğer 2 algoritmadan ayrışmıştır.

**K-NN**

* Doğruya Doğru: 31
* Yanlışa Yanlış: 23
* Doğruya Yanlış: 4
* Yanlışa Doğru: 3

**KNN ve Karar Ağaçları Karşılaştırması**

* Her ikisi de parametrik olmayan yöntemlerdir.
* KNN daha maliyetlidir fakat daha hızlıdır.