**DEÜ Fen Fakültesi**

**Bilgisayar Bilimleri Bölümü**

**BİL 3013 Veri Madenciliğine Giriş**

**Ödev 3. Model oluşturma ve karşılaştırma**

**Kodların Kaggle linki:**

[**https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-exploration**](https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-exploration)

[**https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-knn**](https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-knn)

[**https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-rndm-forest**](https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-rndm-forest)

[**https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-ann**](https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-ann)

[**https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-all-files**](https://www.kaggle.com/code/ozgurd5/bil3013-data-mining-assignment-3-all-files)

**Öğrenci: Özgür Dalbeler - 2022280084**

**Öğretim Üyesi: Prof. Dr. Efendi NASİBOĞLU**

**İzmir 2024**

## 1- Ödevin Tanımı

1. Python’un “**Seaborn**” kütüphanesinden “**mpg.csv**” veri setini kullanın. Veri setini

**0.9/0.1** olarak eğitim ve test setlerine ayırın.

2. “**horsepower**”, “**acceleration**” ve “**weight**” atributlarını kullanarak “**mpg**” değerini

tahmin etmek için **K-en yakın komşu (KNN), Rassal Orman (Random Forest) ve Yapay**

**Sinir Ağı (ANN)** modellerini oluşturun.

3. Modelleri test setinde deneyerek algoritmaların **başarılarını karşılaştırın**. Başarının

karşılaştırılmasında **kullandığınız kriterin formülünü ve açıklamasını raporda yazın.**

4. En başarılı modeli kullanarak **horsepower = 130, acceleration=13, weight=3500** olan

bir otomobilin **“mpg” değerini tahmin edin.**

5. Programın kodunu içeren **.py dosyasını ve Rapor.doc rapor dosyasını Sakai’ye**

**yükleyin.**

6. **Rapor dosyasında modellerin karşılaştırma sonuçlarını içeren yorumlara, tablolara**

**ve grafiklere yer verilmelidir.**

## 2- Kullanılan Yöntemler ve Teknolojiler

**Pandas:** Pandas, Python'da veri analizi ve manipülasyonu için kullanılan bir kütüphanedir. DataFrame, tablo yapısında verileri saklar ve CSV, XLSX, JSON gibi formatlardan veri okuma ve yazma imkanı sunar.

Kurulumu:  
pip install pandas

Kullanımı:  
import pandas as pd

df = pd.DataFrame(ilanlar)

**Matplotlib:** Matplotlib, Python'da veri görselleştirme için kullanılan bir kütüphanedir. Grafikler, histogramlar, çubuk grafikler ve diğer görseller oluşturmak için sıkça kullanılır.

Kurulumu:

pip install matplotlib

Kullanımı:

import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y = [10, 20, 25, 30, 35]

plt.plot(x, y, marker='o')

plt.xlabel('X Ekseni')

plt.ylabel('Y Ekseni')

plt.title('Örnek Grafik')

plt.show()

**Scikit-Learn:** Makine öğrenmesi modelleri (sınıflandırma, regresyon, kümeleme) uygulamak için kullanılan bir kütüphanedir. Veri işleme, model eğitme ve değerlendirme aşamalarında sıkça kullanılır.

Kurulumu:

pip install scikit-learn

Kullanımı:

from sklearn.linear\_model import LinearRegression

# Örnek veri

X = [[1], [2], [3], [4], [5]]

y = [2, 4, 5, 4, 5]

# Model kurma

model = LinearRegression()

model.fit(X, y)

# Tahmin

print("x=6 için tahmin:", model.predict([[6]]))

**Seaborn**: Veri görselleştirme için estetik ve istatistiksel çizimler sunan bir kütüphanedir. Özellikle veri dağılımlarını ve kategorik karşılaştırmaları görselleştirmek için kullanılır.

Kurulumu:

pip install seaborn

Kullanımı:

import seaborn as sns

import matplotlib.pyplot as plt

# Örnek veri

tips = sns.load\_dataset("tips")

# Dağılım grafiği

sns.scatterplot(x="total\_bill", y="tip", data=tips)

plt.xlabel("Toplam Hesap")

plt.ylabel("Bahşiş")

plt.title("Hesap ve Bahşiş İlişkisi")

plt.show()

**NumPy:** NumPy, Python'da bilimsel hesaplamalar için kullanılan bir kütüphanedir. Çok boyutlu diziler (array) oluşturma, matematiksel işlemler yapma ve veri manipülasyonu gibi işlemler için kullanılır.

Kurulumu:

pip install numpy

Kullanımı:

import numpy as np

# NumPy array oluşturma

array = np.array([1, 2, 3, 4, 5])

# Temel işlemler

print("Array:", array)

print("Toplam:", np.sum(array))

print("Ortalama:", np.mean(array))

**Random:** Python’da rastgele sayı üretimi ve seçim işlemleri için kullanılır. Rastgele sayılar, liste elemanları arasından seçim veya karıştırma gibi işlemleri destekler.

Kurulumu:

Python’un kendi içinde gelen bir modül. Kuruluma ihtiyaç bulunmamakta.

Kullanımı:

import random

# 1 ile 10 arasında (dahil) rastgele bir sayı üretme

random\_number = random.randint(1, 10)

print("Rastgele Sayı:", random\_number)

## 3- Uygulama

### 3.1 Keşif

Bu aşamada veri keşfi yapılmaktadır.

#### 3.1.1 Modüller

**import seaborn as sns**

**import matplotlib.pyplot as plt**

#### 3.1.2 Veri Setini Yüklemek

**# Seaborn'un mpg veri setini yükle**

**df = sns.load\_dataset("mpg")**

#### 3.1.3 Eksik Değer Tespiti

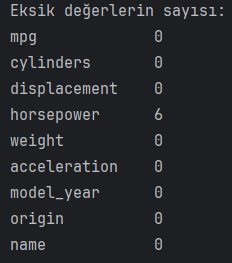
**# Veri setinde eksik değer tespiti**

**print("\nEksik değerlerin sayısı:")**

**print(df.isnull().sum())**

**# 398 veriden 6 tanesi eksik olduğu için eksik verilerin kaybı kabul edilebilir.**

**df = df.dropna(subset=["horsepower"])**

****

#### 3.1.3 İstatiksiksel Özet

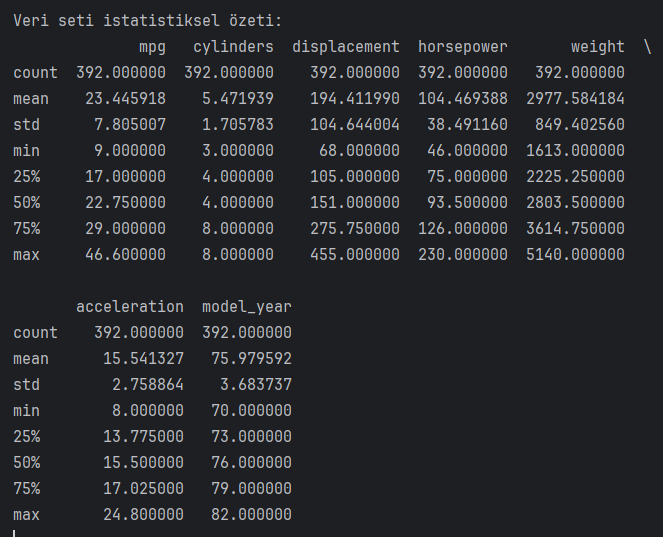
**# Temel istatistiksel özet,**

**pd.set\_option('display.max\_rows', 12) # Max 12 satır göster**

**pd.set\_option('display.max\_columns', 12) # Max 12 sütun göster**

**print("\nVeri seti istatistiksel özeti:")**

**print(df.describe())**

****

**Horsepower:**

Ortalama ~104.47, std ~38.49. 25% dilimde 75 hp, 75% dilimde 126 hp var. Bu, veri setinin yarısının 75 ile 126 hp arasında toplandığını gösterir. Yine de maksimum 230 gibi yüksek değerler mevcut. Sağdan çarpık bir dağılım söz konusu.

**Weight:**

Ortalama ~2977.58, std ~849.40, çok geniş bir aralık (1613 ile 5140). Araçların ağırlıklarının da geniş bir yelpazede değiştiğini görüyoruz. %50 medyan ~2803.5 pound, ancak 75% diliminde 3614.75’e çıkıyor. Bu yüksek varyasyon, modellerde normalleştirmenin faydalı olabileceğini düşündürüyor.

**Acceleration:**

Ortalama ~15.54, std ~2.76. %50 değer 15.5, 25%-75% aralığı 13.78-17.03. Değerler daha sıkı bir şekilde orta alanda toplanmış. Diğer özelliklere göre daha normal bir dağılım var.

**MPG:**

Ortalama (mean) ~23.45, Standart sapma (std) ~7.8, oldukça geniş bir yayılım olduğunu gösteriyor. 25%-75% aralığı (IQR) 17 ile 29 arasında; veri setinin yarısı bu aralıkta. Bu, araçların yakıt verimliliğinde ciddi bir çeşitlilik olduğunu gösteriyor.

#### 3.1.4 Horsepower Histogramı

**# Horsepower**

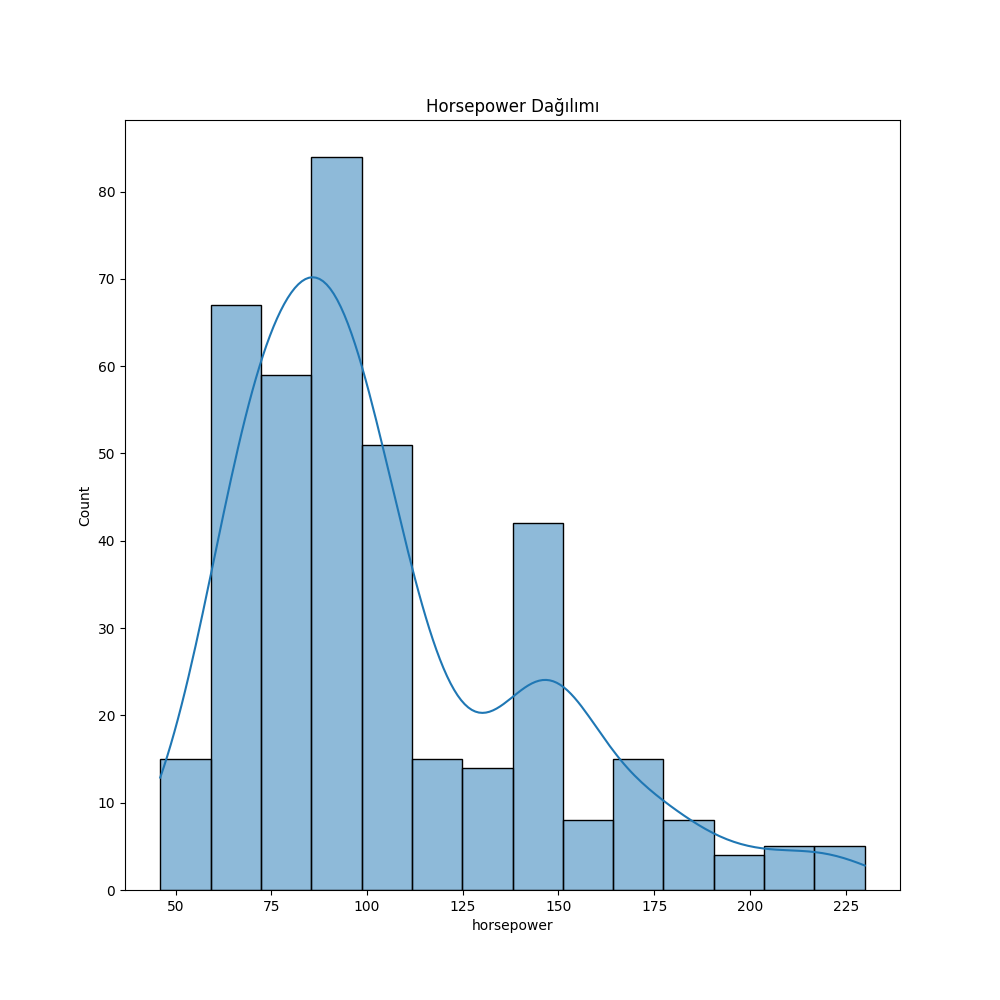
**plt.figure(figsize=(10,10))**

**sns.histplot(df["horsepower"], kde=True)**

**plt.title("Horsepower Dağılımı")**

**plt.savefig("horsepower.png")**

**plt.show()**

****

Sağdan çarpık (right-skewed) bir dağılım var. 65-105 hp aralığı yoğun. Yüksek beygir gücü (150 üzeri) daha az.

#### 3.1.5 Acceleration Histogramı

**# Acceleration**

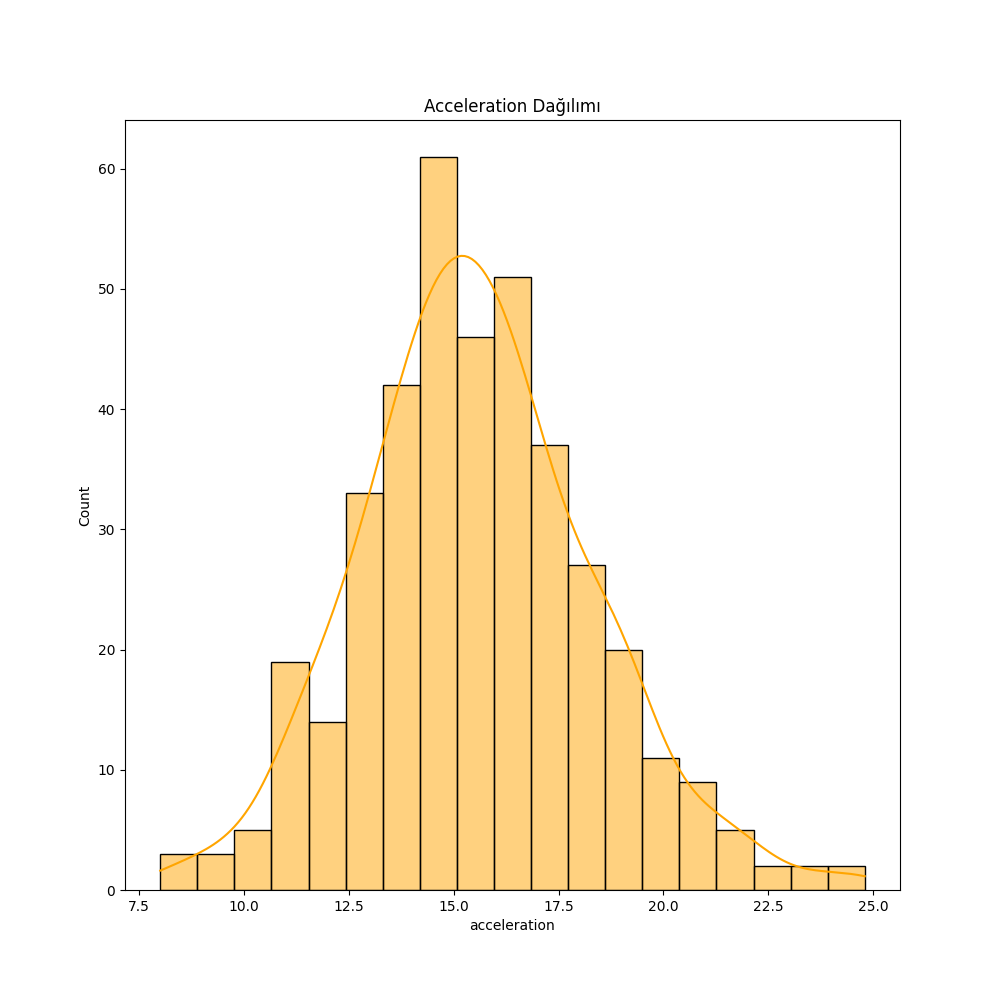
**plt.figure(figsize=(10,10))**

**sns.histplot(df["acceleration"], kde=True, color="orange")**

**plt.title("Acceleration Dağılımı")**

**plt.savefig("acceleration.png")**

**plt.show()**



Yaklaşık normal bir dağılıma sahip. Çoğu otomobil 13-17 aralığında hızlanma değerine sahip. Çok uç değerler yok gibi görünüyor.

#### 3.1.6 Weight Histogramı

**# Weight**

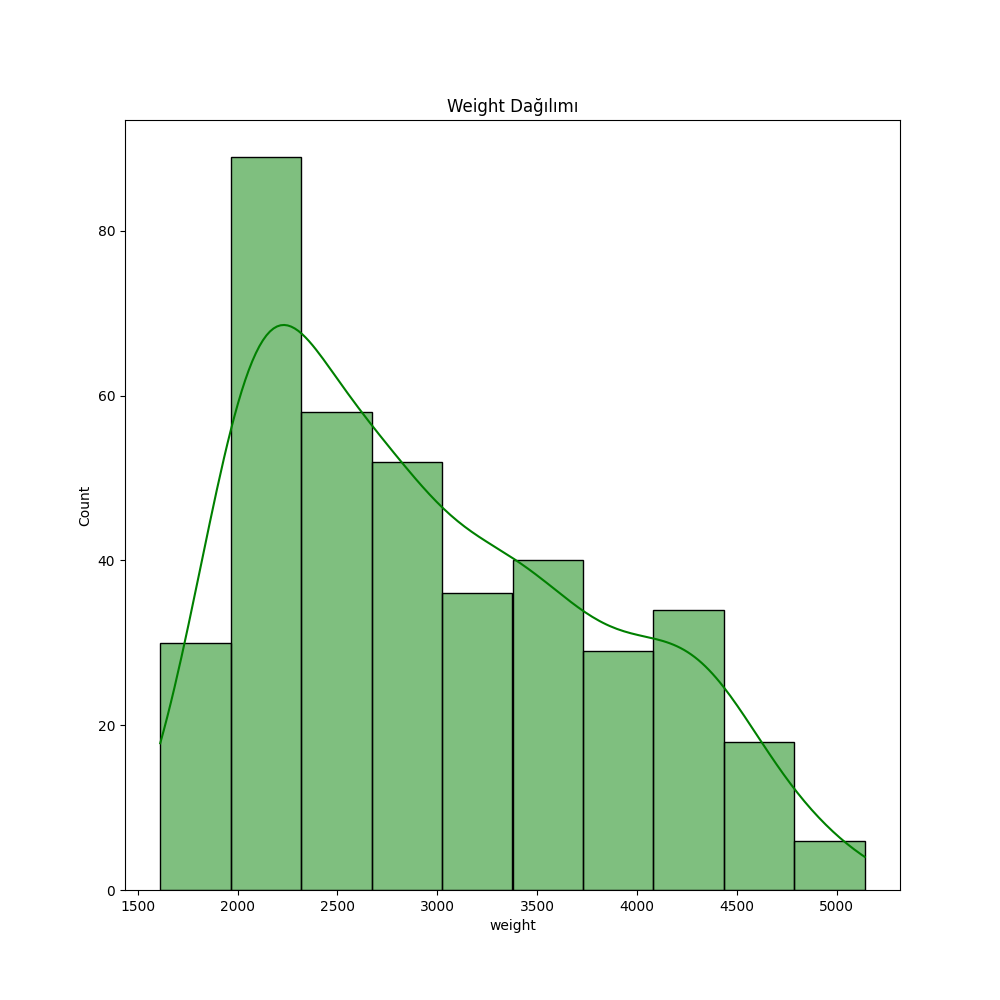
**plt.figure(figsize=(10,10))**

**sns.histplot(df["weight"], kde=True, color="green")**

**plt.title("Weight Dağılımı")**

**plt.savefig("weight.png")**

**plt.show()**



Yine sağdan çarpık bir dağılım. 2000-3000 pound aralığı en yoğun. Ağırlık arttıkça otomobil sayısı düşüyor.

#### 3.1.7 MPG (Mile Per Gallon) Histogramı

**# MPG**

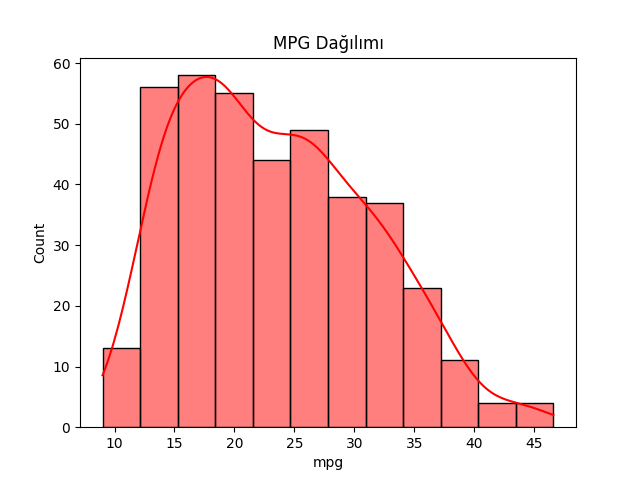
**plt.figure()**

**sns.histplot(df["mpg"], kde=True, color="red")**

**plt.title("MPG Dağılımı")**

**plt.savefig("mpg.png")**

**plt.show()**



Ağırlık ve horsepower’a benzer şekilde sağdan çarpık. 12-30 mpg aralığı en yaygın, yüksek verimli (35+ mpg) araçlar nadir.

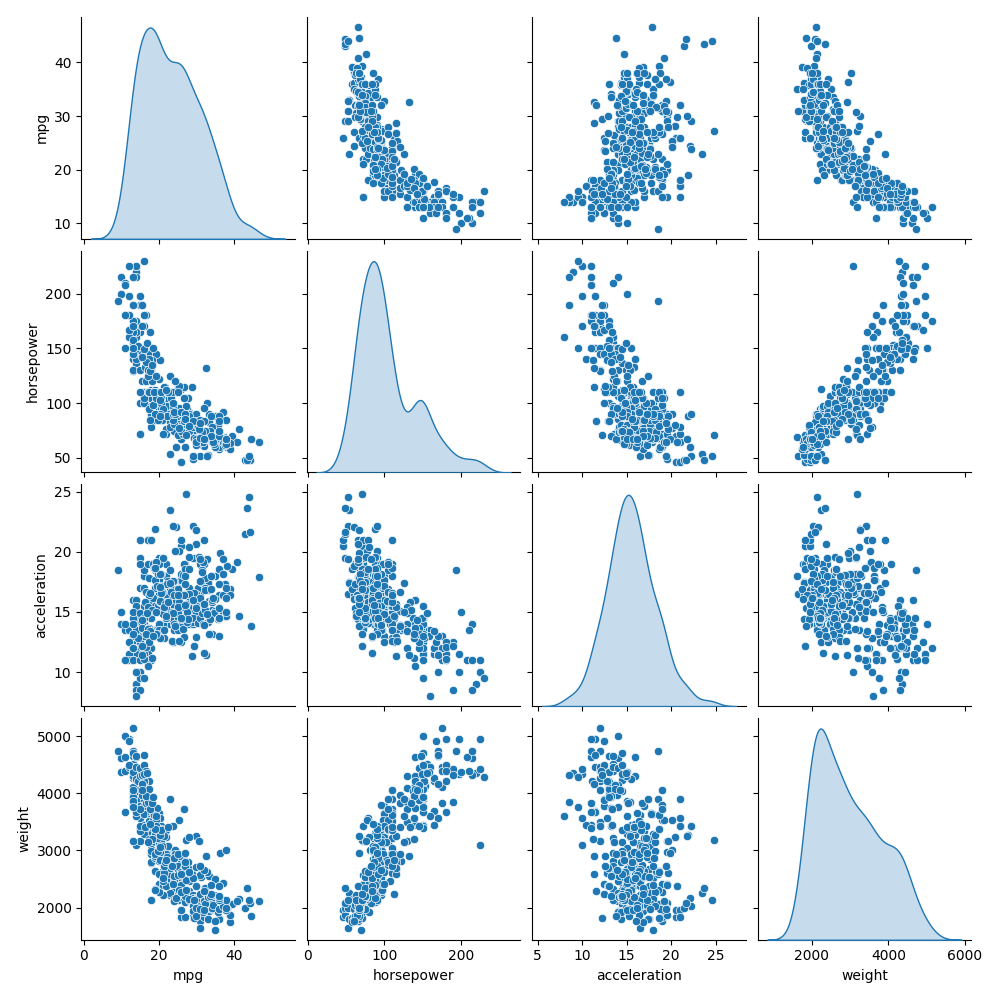
#### 3.1.8 Değişkenler Arası Pair Plot

**# Burada ilgili değişkenleri bir arada görerek ilişkilerini anlayabiliriz.**

**sns.pairplot(df[["mpg", "horsepower", "acceleration", "weight"]], diag\_kind="kde")**

**plt.savefig("pairplot.png")**

**plt.show()**

****

#### 3.1.9 Korelasyon Matrisi

**# Korelasyon matrisi**

**corr = df[["mpg", "horsepower", "acceleration", "weight"]].corr()**

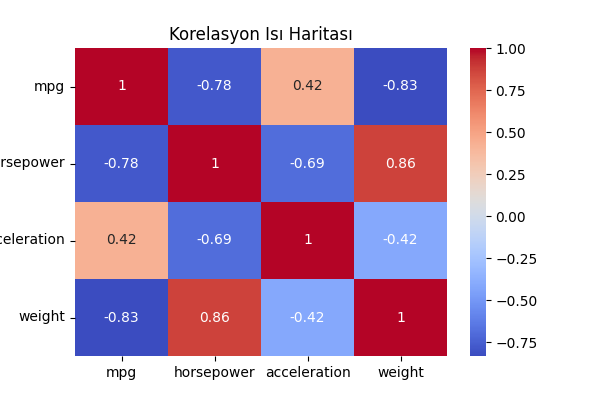
**plt.figure(figsize=(6,4))**

**sns.heatmap(corr, annot=True, cmap="coolwarm")**

**plt.title("Korelasyon Isı Haritası")**

**plt.savefig("correlation.png")**

**plt.show()**

****

**mpg** ile **horsepower** arasında -0.78; mpg ile weight arasında **-0.83** gibi **güçlü negatif** korelasyonlar var. Yani beygir gücü veya ağırlık arttıkça aracın yakıt verimliliği (mpg) düşüyor.

**mpg** ile **acceleration** arasında **pozitif** bir korelasyon var **(0.42)**. Daha hızlı hızlanan araçların genelde mpg’si daha yüksek.

**horsepower** ile **weight** arasında **0.86** gibi çok **güçlü pozitif** bir korelasyon var. Ağır araçların genelde beygir gücü yüksek. Bu da araçların genellikle ağırlaştıkça daha güçlü motora ihtiyaç duymasından kaynaklanıyor olabilir.

#### 3.1.10 Scatter Plotlar

**# Scatter plotlar için değerleri sırala**

**df["horsepower"] = df["horsepower"].sort\_values().values**

**df["acceleration"] = df["acceleration"].sort\_values().values**

**df["weight"] = df["weight"].sort\_values().values**

**df["mpg"] = df["mpg"].sort\_values().values**

**# Scatter plotlar**

**fig, axs = plt.subplots(2, 2, figsize=(12, 8))**

**# Horsepower scatter plot**

**axs[0, 0].scatter(df.index, df["horsepower"], color='blue')**

**axs[0, 0].set\_title("Horsepower Dağılımı")**

**axs[0, 0].set\_xlabel("Index")**

**axs[0, 0].set\_ylabel("Horsepower")**

**# Acceleration scatter plot**

**axs[0, 1].scatter(df.index, df["acceleration"], color='orange')**

**axs[0, 1].set\_title("Acceleration Dağılımı")**

**axs[0, 1].set\_xlabel("Index")**

**axs[0, 1].set\_ylabel("Acceleration")**

**# Weight scatter plot**

**axs[1, 0].scatter(df.index, df["weight"], color='green')**

**axs[1, 0].set\_title("Weight Dağılımı")**

**axs[1, 0].set\_xlabel("Index")**

**axs[1, 0].set\_ylabel("Weight")**

**# MPG scatter plot**

**axs[1, 1].scatter(df.index, df["mpg"], color='red')**

**axs[1, 1].set\_title("MPG Dağılımı")**

**axs[1, 1].set\_xlabel("Index")**

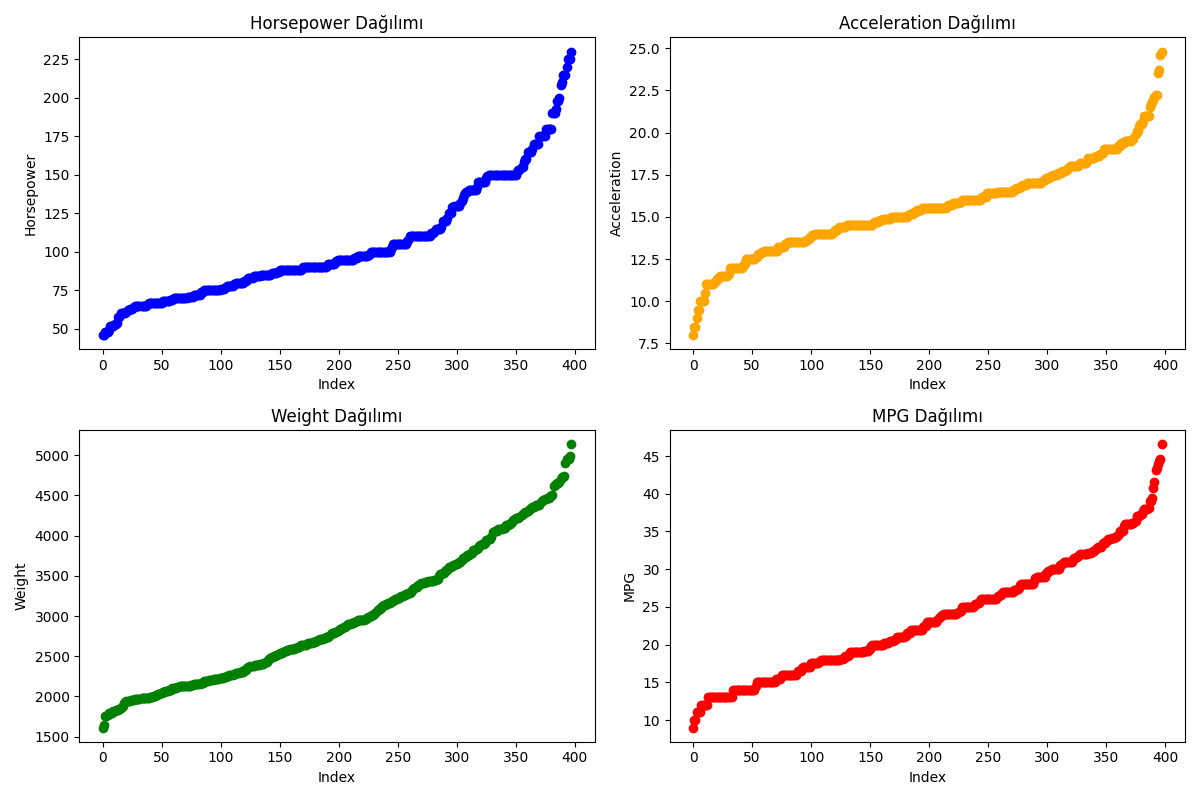
**axs[1, 1].set\_ylabel("MPG")**

**# Layout düzenlemesi ve grafiğin kaydedilmesi**

**plt.tight\_layout()**

**plt.savefig("scatter\_plots.png")**

**plt.show()**

****

#### 3.1.11 Box Plotlar ve Outlierlar

**# Boxplotlar ve outlier tespiti**

**özellikler = ["mpg", "horsepower", "acceleration", "weight"]**

**plt.figure(figsize=(10,8))**

**for i, feature in enumerate(özellikler, 1):**

**plt.subplot(2, 2, i)**

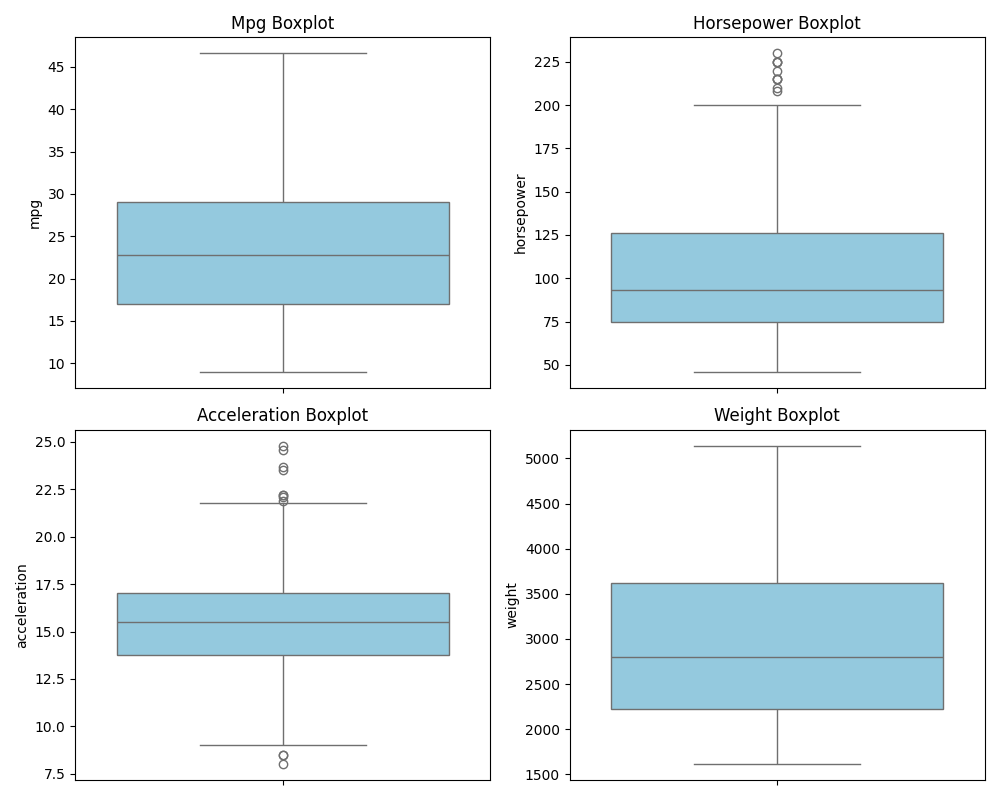
**sns.boxplot(y=df[feature], color='skyblue', whis=1.5)**

**plt.title(f"{feature.capitalize()} Boxplot")**

**plt.tight\_layout()**

**plt.savefig("boxplots.png")**

**plt.show()**



Acceleration ve horsepower’da outlier gözükse de (IQR katsayısı default olarak 1.5, bu değeri değiştirmedim) bu outlierlar kabul edilebilir değerde ve sayıdalar. Histogramlardan veya scatter plotlardan da anlaşılacağı gibi veri setimizde tüm değerler düzenli olarak artıp azalıyor, uçlara doğru ekstrem değer artışları veya azalışları yaşanmıyor.

### 3.2 KNN

#### 3.2.1 Modüller

**import seaborn as sns**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split, cross\_val\_score**

**from sklearn.preprocessing import MinMaxScaler**

**from sklearn.neighbors import KNeighborsRegressor**

**from sklearn.metrics import mean\_squared\_error, r2\_score, make\_scorer, mean\_absolute\_error**

**import random**

#### 3.2.2 Veri Setinin Yüklenmesi ve Bölümü

**# Veri setini yükle**

**df = sns.load\_dataset("mpg")**

**df = df.dropna(subset=["horsepower"]) # eksik değerleri at**

**# Özellik ve hedef seçimi**

**ozellik = df[["horsepower", "acceleration", "weight"]]**

**hedef = df["mpg"]**

**# Eğitim-test ayrımı**

**random\_state = 300 #random.randint(0, 999)**

**print("Random state:", random\_state)**

**ozellik\_egitim, ozellik\_test, hedef\_egitim, hedef\_test = train\_test\_split(ozellik, hedef, test\_size=0.1, random\_state=random\_state)**

#### 3.2.3 Normalizasyon

Horsepower 100 civarında değerler alırken, weight 3000 civarında değerler alıyor. Bu durumda weight, horsepower’dan çok daha etkili olur. Bu sorunu çözmek için normalizasyon yapmalıyız.

**# Normalizasyon**

**min\_max\_normalizator = MinMaxScaler()**

**ozellik\_egitim\_normalize = min\_max\_normalizator.fit\_transform(ozellik\_egitim)**

**ozellik\_test\_normalize = min\_max\_normalizator.transform(ozellik\_test)**

#### 3.2.4 K Sayısının Belirlenmesi

K sayısı modelimizin gücünü ve tahmin kapasitesini doğrudan belirler. En doğru k sayısını seçtiğimizi anlamamız için k-fold cross validation testi yapabiliriz. Test sonuçlarımızı 3 farklı yolla skorlayabiliriz: R Kare, MSE (Mean Squared Error) veya MAE (Mean Absolute Error).

**# K sayısının belirlenmesi**

**k\_degerleri\_adayları = range(1, 100, 2) # Sadece tek sayılar**

**cross\_validation\_df = pd.DataFrame(columns=["k", "r\_kare", "mse", "mae"])**

**for k in k\_degerleri\_adayları:**

**# Mesafeye göre ağırlıklı KNN regresyon modeli**

**knn\_modeli = KNeighborsRegressor(n\_neighbors=k, weights='distance')**

**# Cross-validation ile her k için ortalama R kare skoru**

**r\_kare\_skorları = cross\_val\_score(knn\_modeli, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=make\_scorer(r2\_score, greater\_is\_better=True))**

**r\_kare = r\_kare\_skorları.mean()**

**# Cross-validation ile her k için ortalama MSE skoru**

**mse\_skorları = cross\_val\_score(knn\_modeli, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=make\_scorer(mean\_squared\_error, greater\_is\_better=False))**

**mse = -1 \* mse\_skorları.mean()**

**# Cross-validation ile her k için ortalama MAE skoru**

**mae\_skorları = cross\_val\_score(knn\_modeli, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=make\_scorer(mean\_absolute\_error, greater\_is\_better=False))**

**mae = -1 \* mae\_skorları.mean()**

**# Sonuçları dataframe'e ekle**

**cross\_validation\_df = cross\_validation\_df.\_append({"k": k, "r\_kare": r\_kare, "mse": mse, "mae": mae}, ignore\_index=True)**

**print("k:", k, "R Kare:", r\_kare, "MSE:", mse, "MAE:", mae)**

**# R kare yöntemine göre en iyi k'yı bulma**

**en\_iyi\_r\_kare = cross\_validation\_df["r\_kare"].max()**

**en\_iyi\_r\_kare\_k = cross\_validation\_df[cross\_validation\_df["r\_kare"] == en\_iyi\_r\_kare]["k"].values[0]**

**print("En iyi R Kare:", en\_iyi\_r\_kare, "En iyi k:", en\_iyi\_r\_kare\_k)**

**# MSE yöntemine göre en iyi k'yı bulma**

**en\_iyi\_mse = cross\_validation\_df["mse"].min()**

**en\_iyi\_mse\_k = cross\_validation\_df[cross\_validation\_df["mse"] == en\_iyi\_mse]["k"].values[0]**

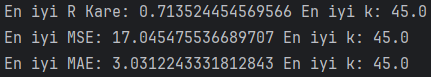
**print("En iyi MSE:", en\_iyi\_mse, "En iyi k:", en\_iyi\_mse\_k)**

**# MAE yöntemine göre en iyi k'yı bulma**

**en\_iyi\_mae = cross\_validation\_df["mae"].min()**

**en\_iyi\_mae\_k = cross\_validation\_df[cross\_validation\_df["mae"] == en\_iyi\_mae]["k"].values[0]**

**print("En iyi MAE:", en\_iyi\_mae, "En iyi k:", en\_iyi\_mae\_k)**

****

#### 3.2.5 Skor Grafikleri

**# K değerlerinin skor grafikleri**

**fig, axes = plt.subplots(1, 3, figsize=(18, 6))**

**# R Kare grafiği**

**axes[0].plot(cross\_validation\_df["k"], cross\_validation\_df["r\_kare"], label="R Kare")**

**axes[0].set\_xlabel("k")**

**axes[0].set\_ylabel("R Kare")**

**axes[0].set\_title("R Kare'ye Göre KNN Regresyon Modeli Performansı")**

**axes[0].legend()**

**axes[0].grid(True)**

**# MSE grafiği**

**axes[1].plot(cross\_validation\_df["k"], cross\_validation\_df["mse"], label="MSE")**

**axes[1].set\_xlabel("k")**

**axes[1].set\_ylabel("MSE")**

**axes[1].set\_title("MSE'ye Göre KNN Regresyon Modeli Performansı")**

**axes[1].legend()**

**axes[1].grid(True)**

**# MAE grafiği**

**axes[2].plot(cross\_validation\_df["k"], cross\_validation\_df["mae"], label="MAE")**

**axes[2].set\_xlabel("k")**

**axes[2].set\_ylabel("MAE")**

**axes[2].set\_title("MAE'ye Göre KNN Regresyon Modeli Performansı")**

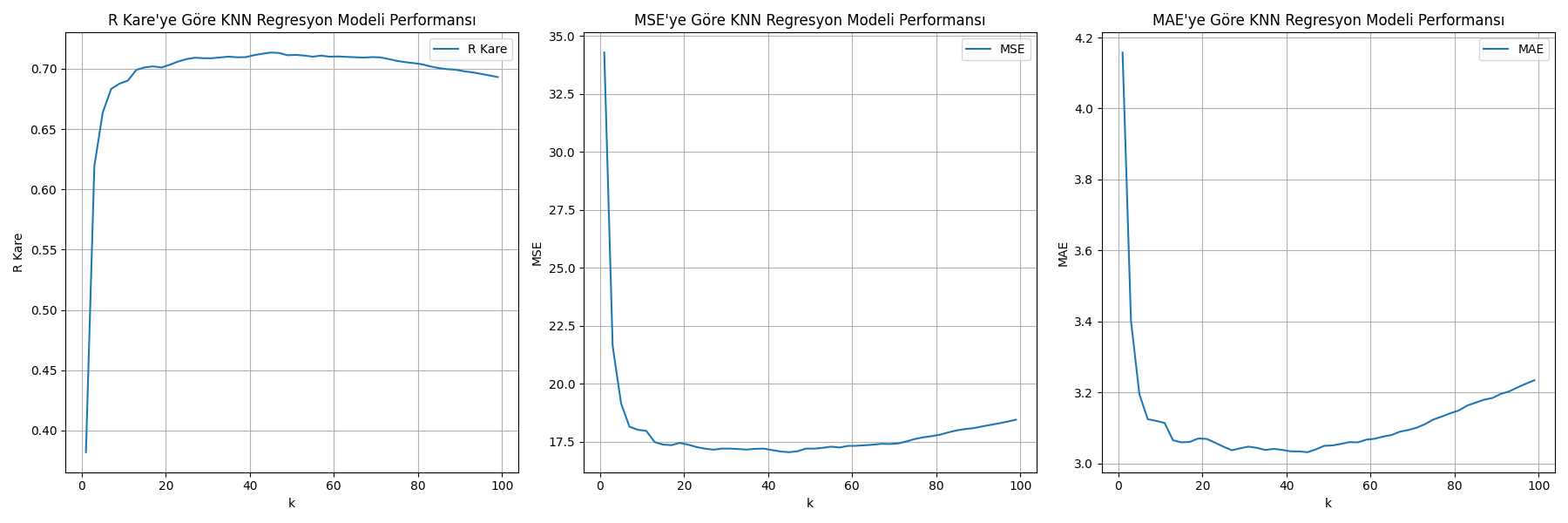
**axes[2].legend()**

**axes[2].grid(True)**

**plt.tight\_layout()**

**plt.savefig("knn.png")**

**plt.show()**



#### 3.2.6 En İyi K Değeri İle Model Eğitimi ve Tahmin

**# Tahmin Hedefleri**

**tahmin\_hedef\_horsepower = 130**

**tahmin\_hedef\_acceleration = 13**

**tahmin\_hedef\_weight = 3500**

**# En iyi R Kare sonucunu veren k değeri ile model oluşturma**

**k\_final = int(en\_iyi\_r\_kare\_k)**

**print(f"\nEn iyi k (R Kare bazlı) ile model oluşturuluyor... k={k\_final}")**

**final\_knn\_model = KNeighborsRegressor(n\_neighbors=k\_final, weights='distance')**

**final\_knn\_model.fit(ozellik\_egitim\_normalize, hedef\_egitim)**

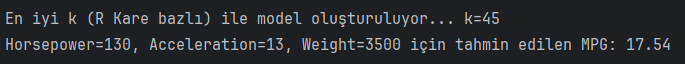
**# Yeni değerler için tahmin**

**araba\_df = pd.DataFrame([[tahmin\_hedef\_horsepower, tahmin\_hedef\_acceleration, tahmin\_hedef\_weight]], columns=ozellik\_egitim.columns)**

**araba\_df\_normalize = min\_max\_normalizator.transform(araba\_df)**

**tahmin\_mpg = final\_knn\_model.predict(araba\_df\_normalize)**

**print(f"Horsepower={tahmin\_hedef\_horsepower}, Acceleration={tahmin\_hedef\_acceleration}, Weight={tahmin\_hedef\_weight} için tahmin edilen MPG: {tahmin\_mpg[0]:.2f}")**

****

### 3.3 Random Forest

#### 3.3.1 Modüller

**import seaborn as sns**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split, cross\_val\_score**

**from sklearn.ensemble import RandomForestRegressor**

**from sklearn.metrics import make\_scorer, r2\_score, mean\_squared\_error, mean\_absolute\_error**

**import random**

#### 3.3.2 Veri Setinin Yüklenmesi

**# Veri setini yükle**

**df = sns.load\_dataset("mpg")**

**df = df.dropna(subset=["horsepower"])**

**# Özellik ve hedef seçimi**

**ozellik = df[["horsepower", "acceleration", "weight"]]**

**hedef = df["mpg"]**

**# Eğitim-test ayrımı**

**random\_state = 300 #random.randint(0, 999)**

**ozellik\_egitim, ozellik\_test, hedef\_egitim, hedef\_test = train\_test\_split(ozellik, hedef, test\_size=0.1, random\_state=random\_state)**

#### 3.3.3 N\_Estimators ve Max\_Depth Sayısının Belirlenmesi

Ağaç sayımız (n\_estimators) ve bu ağaçların büyümesine izin verdiğimiz maksimum derinlik (max\_depth) modelimizin gücünü ve doğruluğunu belirler. En doğru değerleri seçtiğimizi anlamamız için k-fold cross validation testi yapabiliriz. Test sonuçlarımızı 3 farklı yolla skorlayabiliriz: R Kare, MSE (Mean Squared Error) veya MAE (Mean Absolute Error).

**# Cross-validation sonuçlarını saklamak için DataFrame**

**cross\_validation\_sonuçlar = pd.DataFrame(columns=["n\_estimators", "max\_depth", "r\_kare", "mse", "mae"])**

**# Skorlayıcılar**

**mse\_scorer = make\_scorer(mean\_squared\_error, greater\_is\_better=False)**

**mae\_scorer = make\_scorer(mean\_absolute\_error, greater\_is\_better=False)**

**r\_kare\_scorer = make\_scorer(r2\_score, greater\_is\_better=True)**

**# Cross-validation ile sonuçların hesaplanması**

**for n\_estimators in n\_estimators\_adayları:**

**for max\_depth in max\_depth\_adayları:**

**random\_forest\_model = RandomForestRegressor(n\_estimators=n\_estimators, max\_depth=max\_depth, random\_state=random\_state)**

**# R Kare**

**r\_kare\_skorları = cross\_val\_score(random\_forest\_model, ozellik\_egitim, hedef\_egitim, cv=5, scoring=r\_kare\_scorer)**

**r\_kare = r\_kare\_skorları.mean()**

**# MSE**

**mse\_skorları = cross\_val\_score(random\_forest\_model, ozellik\_egitim, hedef\_egitim, cv=5, scoring=mse\_scorer)**

**mse = (-1) \* mse\_skorları.mean()**

**# MAE**

**mae\_skorları = cross\_val\_score(random\_forest\_model, ozellik\_egitim, hedef\_egitim, cv=5, scoring=mae\_scorer)**

**mae = (-1) \* mae\_skorları.mean()**

**# Sonucu tabloya ekle**

**cross\_validation\_sonuçlar = cross\_validation\_sonuçlar.\_append({**

**"n\_estimators": n\_estimators,**

**"max\_depth": max\_depth,**

**"r\_kare": r\_kare,**

**"mse": mse,**

**"mae": mae**

**}, ignore\_index=True)**

**print("n\_estimators:", n\_estimators, "max\_depth:", max\_depth, "R Kare:", r\_kare, "MSE:", mse, "MAE:", mae)**

**# R kare yöntemine göre en iyi n\_estimators ve max\_depth'u bulma**

**en\_iyi\_r\_kare = cross\_validation\_sonuçlar["r\_kare"].max()**

**en\_iyi\_r\_kare\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["r\_kare"] == en\_iyi\_r\_kare].iloc[0]**

**en\_iyi\_r\_kare\_n\_estimators = en\_iyi\_r\_kare\_satir["n\_estimators"]**

**en\_iyi\_r\_kare\_max\_depth = en\_iyi\_r\_kare\_satir["max\_depth"]**

**print("En iyi R Kare:", en\_iyi\_r\_kare, "En iyi n\_estimators:", en\_iyi\_r\_kare\_n\_estimators, "En iyi max\_depth:", en\_iyi\_r\_kare\_max\_depth)**

**# MSE yöntemine göre en iyi n\_estimators ve max\_depth'u bulma**

**en\_iyi\_mse = cross\_validation\_sonuçlar["mse"].min()**

**en\_iyi\_mse\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["mse"] == en\_iyi\_mse].iloc[0]**

**en\_iyi\_mse\_n\_estimators = en\_iyi\_mse\_satir["n\_estimators"]**

**en\_iyi\_mse\_max\_depth = en\_iyi\_mse\_satir["max\_depth"]**

**print("En iyi MSE:", en\_iyi\_mse, "En iyi n\_estimators:", en\_iyi\_mse\_n\_estimators, "En iyi max\_depth:", en\_iyi\_mse\_max\_depth)**

**# MAE yöntemine göre en iyi n\_estimators ve max\_depth'u bulma**

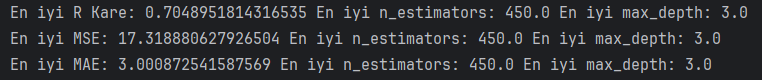
**en\_iyi\_mae = cross\_validation\_sonuçlar["mae"].min()**

**en\_iyi\_mae\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["mae"] == en\_iyi\_mae].iloc[0]**

**en\_iyi\_mae\_n\_estimators = en\_iyi\_mae\_satir["n\_estimators"]**

**en\_iyi\_mae\_max\_depth = en\_iyi\_mae\_satir["max\_depth"]**

**print("En iyi MAE:", en\_iyi\_mae, "En iyi n\_estimators:", en\_iyi\_mae\_n\_estimators, "En iyi max\_depth:", en\_iyi\_mae\_max\_depth)**

****

#### 3.3.4 Skor Grafikleri

**# Grafikler: her bir max\_depth için ayrı çizgi**

**fig, axes = plt.subplots(1, 3, figsize=(18, 6))**

**# R Kare grafiği (her max\_depth farklı çizgi)**

**for max\_depth in max\_depth\_adayları:**

**subset = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["max\_depth"] == max\_depth]**

**axes[0].plot(subset["n\_estimators"], subset["r\_kare"], marker='o', label=f"max\_depth={max\_depth}")**

**axes[0].set\_xlabel("n\_estimators")**

**axes[0].set\_ylabel("R Kare")**

**axes[0].set\_title("R Kare'ye Göre Random Forest Regresyon Modeli Performansı")**

**axes[0].legend()**

**axes[0].grid(True)**

**# MSE grafiği (her max\_depth farklı çizgi)**

**for max\_depth in max\_depth\_adayları:**

**subset = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["max\_depth"] == max\_depth]**

**axes[1].plot(subset["n\_estimators"], subset["mse"], marker='o', label=f"max\_depth={max\_depth}")**

**axes[1].set\_xlabel("n\_estimators")**

**axes[1].set\_ylabel("MSE")**

**axes[1].set\_title("MSE'ye Göre Random Forest Regresyon Modeli Performansı")**

**axes[1].legend()**

**axes[1].grid(True)**

**# MAE grafiği (her max\_depth farklı çizgi)**

**for max\_depth in max\_depth\_adayları:**

**subset = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["max\_depth"] == max\_depth]**

**axes[2].plot(subset["n\_estimators"], subset["mae"], marker='o', label=f"max\_depth={max\_depth}")**

**axes[2].set\_xlabel("n\_estimators")**

**axes[2].set\_ylabel("MAE")**

**axes[2].set\_title("MAE'ye Göre Random Forest Regresyon Modeli Performansı")**

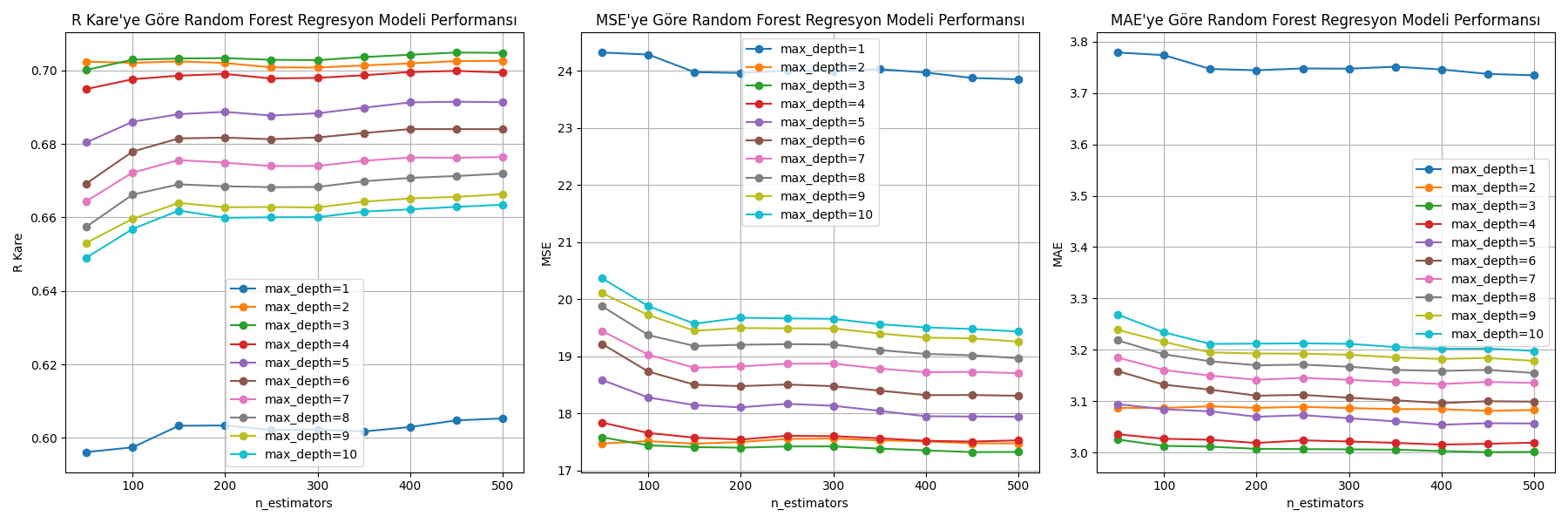
**axes[2].legend()**

**axes[2].grid(True)**

**plt.tight\_layout()**

**plt.savefig("random\_forest.png")**

**plt.show()**

****

#### 3.3.5 En İyi Değerler İle Model Eğitimi ve Tahmin

**# Tahmin Hedefleri**

**tahmin\_hedef\_horsepower = 130**

**tahmin\_hedef\_acceleration = 13**

**tahmin\_hedef\_weight = 3500**

**# En iyi R Kare sonucunu veren n\_estimators ve max\_depth değerleri ile model oluşturma**

**en\_iyi\_n\_estimators\_r\_kare = int(en\_iyi\_r\_kare\_n\_estimators)**

**en\_iyi\_max\_depth\_r\_kare = int(en\_iyi\_r\_kare\_max\_depth)**

**print(f"\nEn iyi R Kare sonucunu veren model oluşturuluyor... n\_estimators={en\_iyi\_n\_estimators\_r\_kare}, max\_depth={en\_iyi\_max\_depth\_r\_kare}")**

**final\_random\_forest\_model = RandomForestRegressor(n\_estimators=en\_iyi\_n\_estimators\_r\_kare, max\_depth=en\_iyi\_max\_depth\_r\_kare, random\_state=random\_state)**

**final\_random\_forest\_model.fit(ozellik\_egitim, hedef\_egitim)**

**# Yeni değerler için tahmin**

**araba\_df = pd.DataFrame([[tahmin\_hedef\_horsepower, tahmin\_hedef\_acceleration, tahmin\_hedef\_weight]], columns=ozellik\_egitim.columns)**

**tahmin\_mpg = final\_random\_forest\_model.predict(araba\_df)**

**print(f"Horsepower={tahmin\_hedef\_horsepower}, Acceleration={tahmin\_hedef\_acceleration}, Weight={tahmin\_hedef\_weight} için tahmin edilen MPG: {tahmin\_mpg[0]:.2f}")**



### 3.4 ANN

#### 3.4.1 Modüller

**import seaborn as sns**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split, cross\_val\_score**

**from sklearn.neural\_network import MLPRegressor**

**from sklearn.metrics import make\_scorer, r2\_score, mean\_squared\_error, mean\_absolute\_error**

**from sklearn.preprocessing import MinMaxScaler**

**import random**

#### 3.4.2 Veri Setinin Yüklenmesi

**# Veri setini yükle**

**df = sns.load\_dataset("mpg").dropna(subset=["horsepower"])**

**# Özellik ve hedef seçimi**

**ozellik = df[["horsepower", "acceleration", "weight"]]**

**hedef = df["mpg"]**

**# Eğitim-test ayrımı**

**random\_state = 300 #random.randint(0, 999)**

**ozellik\_egitim, ozellik\_test, hedef\_egitim, hedef\_test = train\_test\_split(ozellik, hedef, test\_size=0.1, random\_state=random\_state)**

#### 3.4.3 Normalizasyon

Parametreler arasındaki ölçek farkı modelimizi etkileyebilir. Bu yüzden normalizasyon yapmalıyız.

**# Normalizasyon**

**min\_max\_normalizator = MinMaxScaler()**

**ozellik\_egitim\_normalize = min\_max\_normalizator.fit\_transform(ozellik\_egitim)**

**ozellik\_test\_normalize = min\_max\_normalizator.transform(ozellik\_test)**

#### 3.4.4 Parametrelerin Değerlerinin Belirlenmesi

**# Skorlayıcılar**

**mse\_scorer = make\_scorer(mean\_squared\_error, greater\_is\_better=False)**

**mae\_scorer = make\_scorer(mean\_absolute\_error, greater\_is\_better=False)**

**r\_kare\_scorer = make\_scorer(r2\_score, greater\_is\_better=True)**

**# Hiperparametre adayları**

**hidden\_layer\_sizes\_adayları = []**

**for katman\_sayisi in range(1, 8):**

**for düğüm\_sayisi in range(2,13):**

**hidden\_layer = tuple([düğüm\_sayisi] \* katman\_sayisi)**

**hidden\_layer\_sizes\_adayları.append(hidden\_layer)**

**print("hidden\_layer\_sizes\_adayları:", hidden\_layer\_sizes\_adayları)**

**alpha\_adayları = [0.0001, 0.001]**

**max\_iter = 1000**

**# Cross-validation sonuçlarını saklayacağımız DataFrame**

**cross\_validation\_sonuçlar = pd.DataFrame(columns=["hidden\_layer\_sizes", "alpha", "r\_kare", "mse", "mae"])**

**# Cross-validation ile sonuçların hesaplanması**

**for hidden\_layers in hidden\_layer\_sizes\_adayları:**

**for alpha\_degeri in alpha\_adayları:**

**ann\_model = MLPRegressor(hidden\_layer\_sizes=hidden\_layers, alpha=alpha\_degeri, random\_state=random\_state, max\_iter=max\_iter)**

**# R Kare**

**r\_kare\_skorları = cross\_val\_score(ann\_model, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=r\_kare\_scorer)**

**r\_kare = r\_kare\_skorları.mean()**

**# MSE**

**mse\_skorları = cross\_val\_score(ann\_model, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=mse\_scorer)**

**mse = (-1) \* mse\_skorları.mean()**

**# MAE**

**mae\_skorları = cross\_val\_score(ann\_model, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=mae\_scorer)**

**mae = (-1) \* mae\_skorları.mean()**

**# Sonuçları dataframe'e ekle**

**cross\_validation\_sonuçlar = cross\_validation\_sonuçlar.\_append({**

**"hidden\_layer\_sizes": hidden\_layers,**

**"alpha": alpha\_degeri,**

**"r\_kare": r\_kare,**

**"mse": mse,**

**"mae": mae**

**}, ignore\_index=True)**

**print("hidden\_layer\_sizes:", hidden\_layers, "alpha:", alpha\_degeri, "R Kare:", r\_kare, "MSE:", mse, "MAE:", mae)**

**# R kare yöntemine göre en iyi hiperparametreleri bulma**

**en\_iyi\_r\_kare = cross\_validation\_sonuçlar["r\_kare"].max()**

**en\_iyi\_r\_kare\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["r\_kare"] == en\_iyi\_r\_kare].iloc[0]**

**en\_iyi\_r\_kare\_hidden\_layers = en\_iyi\_r\_kare\_satir["hidden\_layer\_sizes"]**

**en\_iyi\_r\_kare\_alpha = en\_iyi\_r\_kare\_satir["alpha"]**

**print("En iyi R Kare:", en\_iyi\_r\_kare, "En iyi hidden\_layer\_sizes:", en\_iyi\_r\_kare\_hidden\_layers, "En iyi alpha:", en\_iyi\_r\_kare\_alpha)**

**# MSE yöntemine göre en iyi hiperparametreleri bulma**

**en\_iyi\_mse = cross\_validation\_sonuçlar["mse"].min()**

**en\_iyi\_mse\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["mse"] == en\_iyi\_mse].iloc[0]**

**en\_iyi\_mse\_hidden\_layers = en\_iyi\_mse\_satir["hidden\_layer\_sizes"]**

**en\_iyi\_mse\_alpha = en\_iyi\_mse\_satir["alpha"]**

**print("En iyi MSE:", en\_iyi\_mse, "En iyi hidden\_layer\_sizes:", en\_iyi\_mse\_hidden\_layers, "En iyi alpha:", en\_iyi\_mse\_alpha)**

**# MAE yöntemine göre en iyi hiperparametreleri bulma**

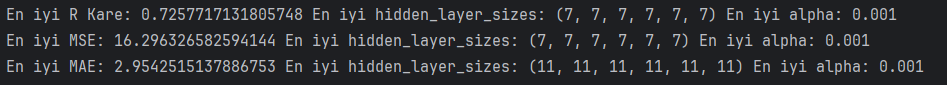
**en\_iyi\_mae = cross\_validation\_sonuçlar["mae"].min()**

**en\_iyi\_mae\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["mae"] == en\_iyi\_mae].iloc[0]**

**en\_iyi\_mae\_hidden\_layers = en\_iyi\_mae\_satir["hidden\_layer\_sizes"]**

**en\_iyi\_mae\_alpha = en\_iyi\_mae\_satir["alpha"]**

**print("En iyi MAE:", en\_iyi\_mae, "En iyi hidden\_layer\_sizes:", en\_iyi\_mae\_hidden\_layers, "En iyi alpha:", en\_iyi\_mae\_alpha)**



#### 3.4.5 Skor Grafikleri

# R Kare grafiği (her alpha için ayrı çizgi)

for alpha\_degeri in alpha\_adayları:

subset = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["alpha"] == alpha\_degeri]

unique\_layers = list(hidden\_layer\_sizes\_adayları)

x\_indices = [unique\_layers.index(h) for h in subset["hidden\_layer\_sizes"]]

axes[0].plot(x\_indices, subset["r\_kare"], marker='o', label=f"alpha={alpha\_degeri}")

axes[0].set\_xlabel("hidden\_layer\_sizes index")

axes[0].set\_ylabel("R Kare")

axes[0].set\_title("R Kare'ye Göre ANN Modeli Performansı")

axes[0].legend()

axes[0].grid(True)

# MSE grafiği

for alpha\_degeri in alpha\_adayları:

subset = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["alpha"] == alpha\_degeri]

unique\_layers = list(hidden\_layer\_sizes\_adayları)

x\_indices = [unique\_layers.index(h) for h in subset["hidden\_layer\_sizes"]]

axes[1].plot(x\_indices, subset["mse"], marker='o', label=f"alpha={alpha\_degeri}")

axes[1].set\_xlabel("hidden\_layer\_sizes index")

axes[1].set\_ylabel("MSE")

axes[1].set\_title("MSE'ye Göre ANN Modeli Performansı")

axes[1].legend()

axes[1].grid(True)

# MAE grafiği

for alpha\_degeri in alpha\_adayları:

subset = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["alpha"] == alpha\_degeri]

unique\_layers = list(hidden\_layer\_sizes\_adayları)

x\_indices = [unique\_layers.index(h) for h in subset["hidden\_layer\_sizes"]]

axes[2].plot(x\_indices, subset["mae"], marker='o', label=f"alpha={alpha\_degeri}")

axes[2].set\_xlabel("hidden\_layer\_sizes index")

axes[2].set\_ylabel("MAE")

axes[2].set\_title("MAE'ye Göre ANN Modeli Performansı")

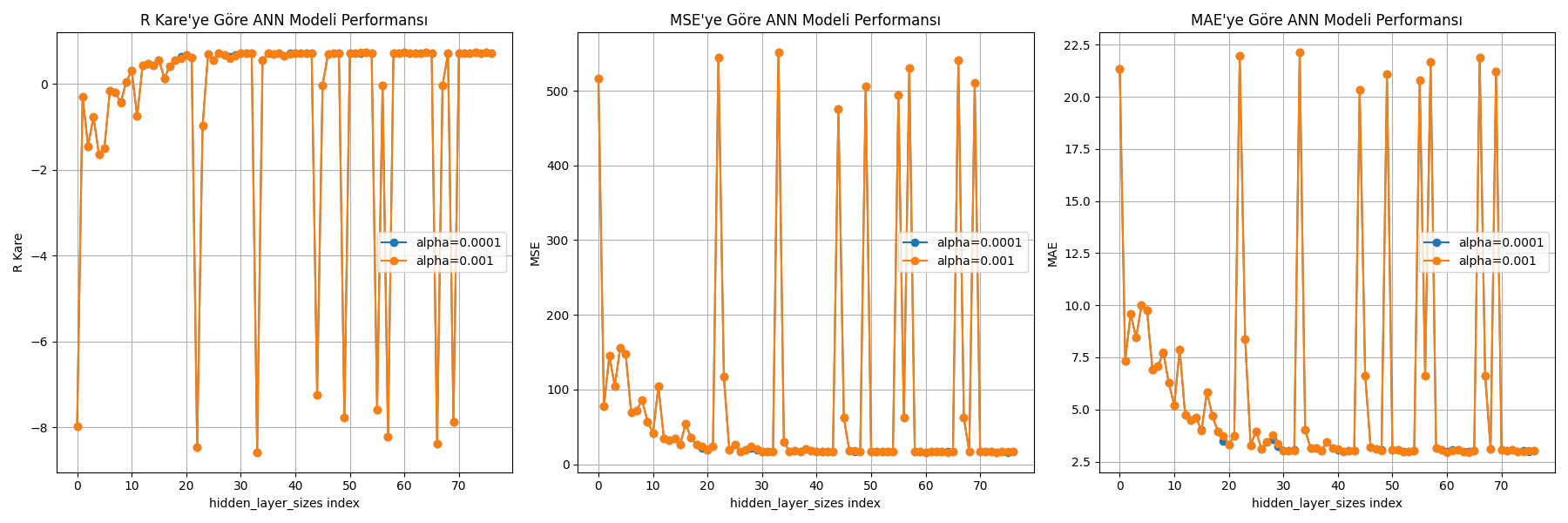
axes[2].legend()

axes[2].grid(True)

plt.tight\_layout()

plt.savefig("ann.png")

plt.show()



#### 3.4.6 En İyi Değerler İle Model Eğitimi ve Tahmin

**# Tahmin Hedefleri**

**tahmin\_hedef\_horsepower = 130**

**tahmin\_hedef\_acceleration = 13**

**tahmin\_hedef\_weight = 3500**

**# En iyi R Kare sonucunu veren hiperparametreler ile model oluşturma**

**print(f"\nEn iyi hiperparametrelerle ANN modeli oluşturuluyor...")**

**print(f"Hidden Layer Sizes: {en\_iyi\_r\_kare\_hidden\_layers}")**

**print(f"Alpha: {en\_iyi\_r\_kare\_alpha}")**

**# Final ANN modelini oluştur**

**final\_ann\_model = MLPRegressor(**

**hidden\_layer\_sizes=en\_iyi\_r\_kare\_hidden\_layers,**

**alpha=en\_iyi\_r\_kare\_alpha,**

**random\_state=random\_state,**

**max\_iter=max\_iter**

**)**

**# Modeli normalize edilmiş eğitim verileriyle eğit**

**final\_ann\_model.fit(ozellik\_egitim\_normalize, hedef\_egitim)**

**# Yeni değerler için tahmin**

**araba\_df = pd.DataFrame([[tahmin\_hedef\_horsepower, tahmin\_hedef\_acceleration, tahmin\_hedef\_weight]],**

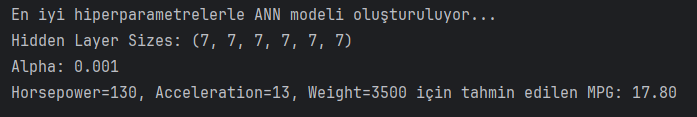
**columns=ozellik\_egitim.columns)**

**araba\_df\_normalize = min\_max\_normalizator.transform(araba\_df)**

**# Tahmin yap**

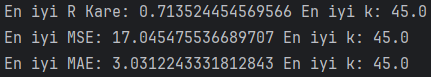
**tahmin\_mpg = final\_ann\_model.predict(araba\_df\_normalize)**

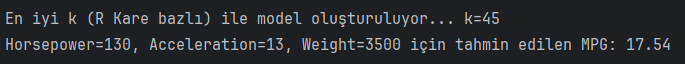
**print(f"Horsepower={tahmin\_hedef\_horsepower}, Acceleration={tahmin\_hedef\_acceleration}, Weight={tahmin\_hedef\_weight} için tahmin edilen MPG: {tahmin\_mpg[0]:.2f}")**



## 4- En İyi Model ve Tahminleri

### 4.1 KNN

****

****

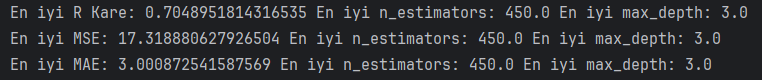
R Kare = 0.71

MSE = 17.04

MAE = 3.03

Tahmin = 17.54

### 4.2 Random Forest

****



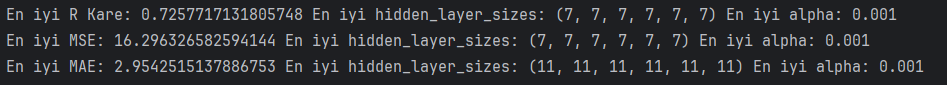
R Kare = 0.70

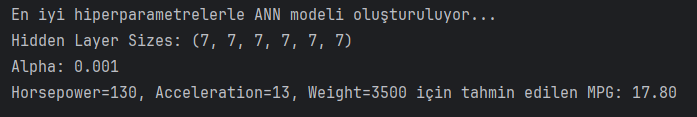
MSE = 17.31

MAE = 3.00

Tahmin = 17.06

### 4.3 ANN





R Kare = 0.72

MSE = 16.29

MAE = 2.95

Tahmin = 17.80

### 4.4 Yorum

Modellerin hepsi birbirleri ile çok yakın skora ve tahmine sahipler. KNN ve Random Forest eğitmesi ve test etmesi çok daha kolay ve işlem maliyeti bakımından ucuzdu. ANN ise çok çok daha maliyetli bir eğitim sürecine sahip. Bilgisayarımın yettiği kadarıyla pek çok deneme yaptım, hepsi 20-30dk süren işlemlerdi ve bekleme sürem çok fazlaydı. Daha fazla tune edebilirdim, bazı parametreleri değiştirmedim, kütüphanenin default değerlerini kullandım.

Değerlendirme yöntemimizi R Kare seçersek en başarılı model Random Forest ancak aralarında 0.1 fark mevcut. MSE ve MAE karşılaştırması yaparsak ANN iki durumda da en iyi model.

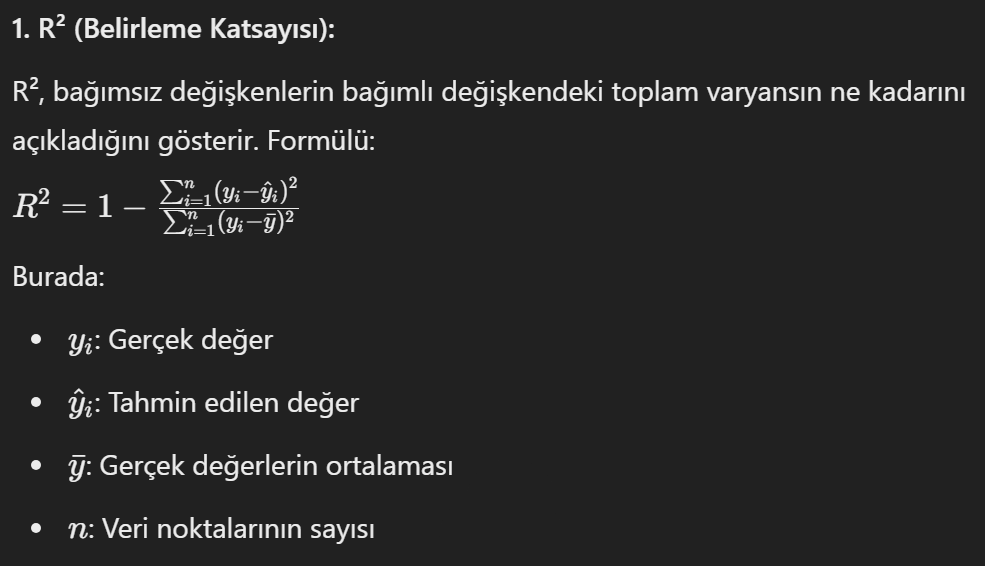
## 5- Değerlendirme

Ödeve doğrudan keşif yaparak başladım. Veri setinin pek çok özelliği ile ilgili grafikler çizdirip yorumlarda ve değerlendirmelerde bulundum. Histogramlar, scatter plotlar ve korelasyon matrisleri çizdirdim.

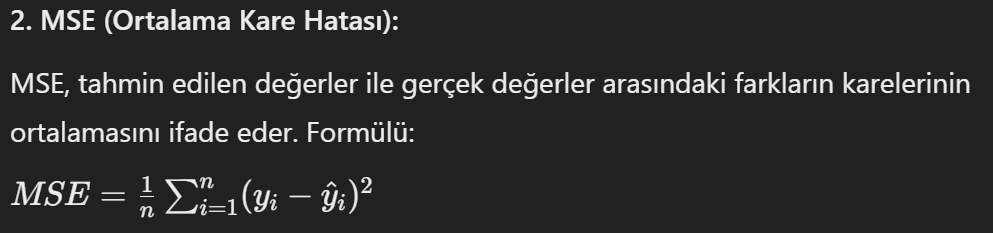
Daha sonra KNN için araştırmalarda bulundum. Bu sefer kütüphaneleri kullanmakta zorlanmadım ve ilk olarak min-max normalizasyonu ile sayısal değerleri normalize ettim çünkü özelliklerimiz aynı değer aralıklarında bulunmuyor. Weight 3000 civarında dolaşırken horsepower 100 civarında dolaşıyor.

Buraya kadar herhangi bir zorlukla karşılaşmadım. İlk zorluğum k sayısını seçme ile ilgiliydi. Uygun k sayısına ulaşıp ulaşamadığımı değerlendirmenin yollarını araştırdım. Cross validation adında aşırı yaygın kullanılan bir mantık öğrendim. Veri setini rastgele n sayıda parçaya bölüp sırayla bir parçayı test diğer parçaları ise eğitim olarak seçip modeli eğitiyor. Daha sonra test verisinin tahmin edilen ve gerçek değerlerini karşılaştırarak skorluyoruz. Skorlamanın birkaç yolu var. Bunları da öğrendim.

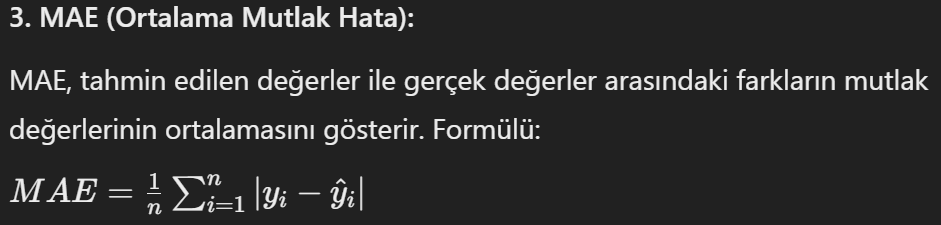
R Kare:



Mean Squared Error:



Mean Absolute Error:



Daha sonra bu değerleri bulmak için birtakım denemelerde bulundum. İlk olarak tüm değişkenleri farklı olarak düşündüm ve ayrıştırdım. İlk değişkenimiz verileri random ayrıştırırken kullanmamız gereken random state yani seed idi. İkinci değişkenimiz tabi ki de k sayımız. Üçüncü değişkenimiz cross validation yaparkenki fold yani katlama sayımız, n. Son değişkenimiz ise aslında 3 adet ölçekten oluşuyor: R Kare, MSE ve MAE.  
  
Bunun için şu kodu yazdım. K değerlerini tek sayıda seçme sebebim ise eşitlik durumunu önlemek.

**# K sayısının belirlenmesi**

**k\_degerleri = range(1, 100, 2) # Sadece tek sayılar**

**cross\_validation\_katları = range(2, 100)**

**cross\_validation\_df = pd.DataFrame(columns=["k", "n", "r\_kare", "mse", "mae"])**

**for k in k\_degerleri:**

**for n in cross\_validation\_katları:**

**# Mesafeye göre ağırlıklı KNN regresyon modeli**

**knn\_modeli = KNeighborsRegressor(n\_neighbors=k, weights='distance')**

**# Cross-validation ile her k için ortalama R kare skoru**

**r\_kare\_skorları = cross\_val\_score(knn\_modeli, ozellik\_egitim\_normalize, hedef\_egitim, cv=n, scoring=make\_scorer(r2\_score, greater\_is\_better=True))**

**r\_kare = r\_kare\_skorları.mean()**

**# Cross-validation ile her k için ortalama MSE skoru**

**mse\_skorları = cross\_val\_score(knn\_modeli, ozellik\_egitim\_normalize, hedef\_egitim, cv=n, scoring=make\_scorer(mean\_squared\_error, greater\_is\_better=False))**

**mse = -1 \* mse\_skorları.mean()**

**# Cross-validation ile her k için ortalama MAE skoru**

**mae\_skorları = cross\_val\_score(knn\_modeli, ozellik\_egitim\_normalize, hedef\_egitim, cv=n, scoring=make\_scorer(mean\_absolute\_error, greater\_is\_better=False))**

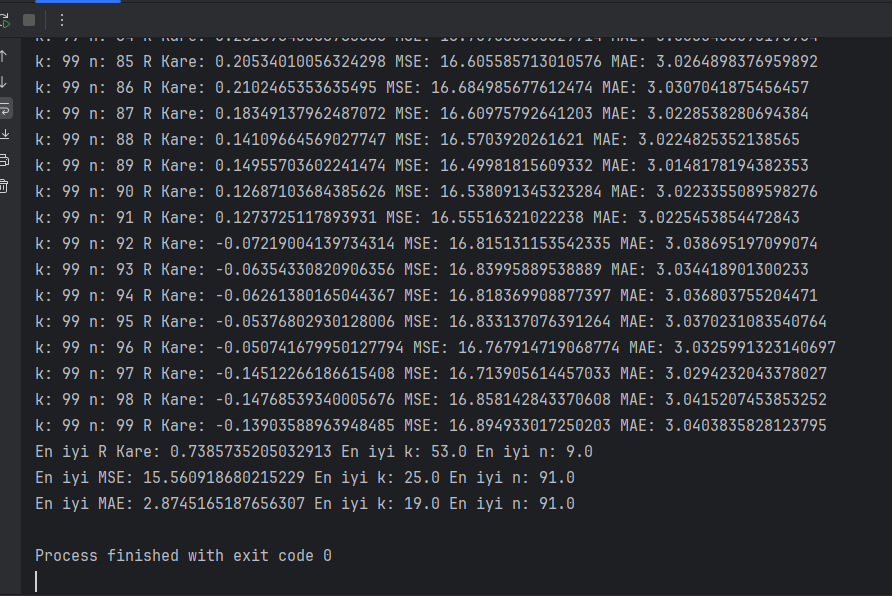
**mae = -1 \* mae\_skorları.mean()**

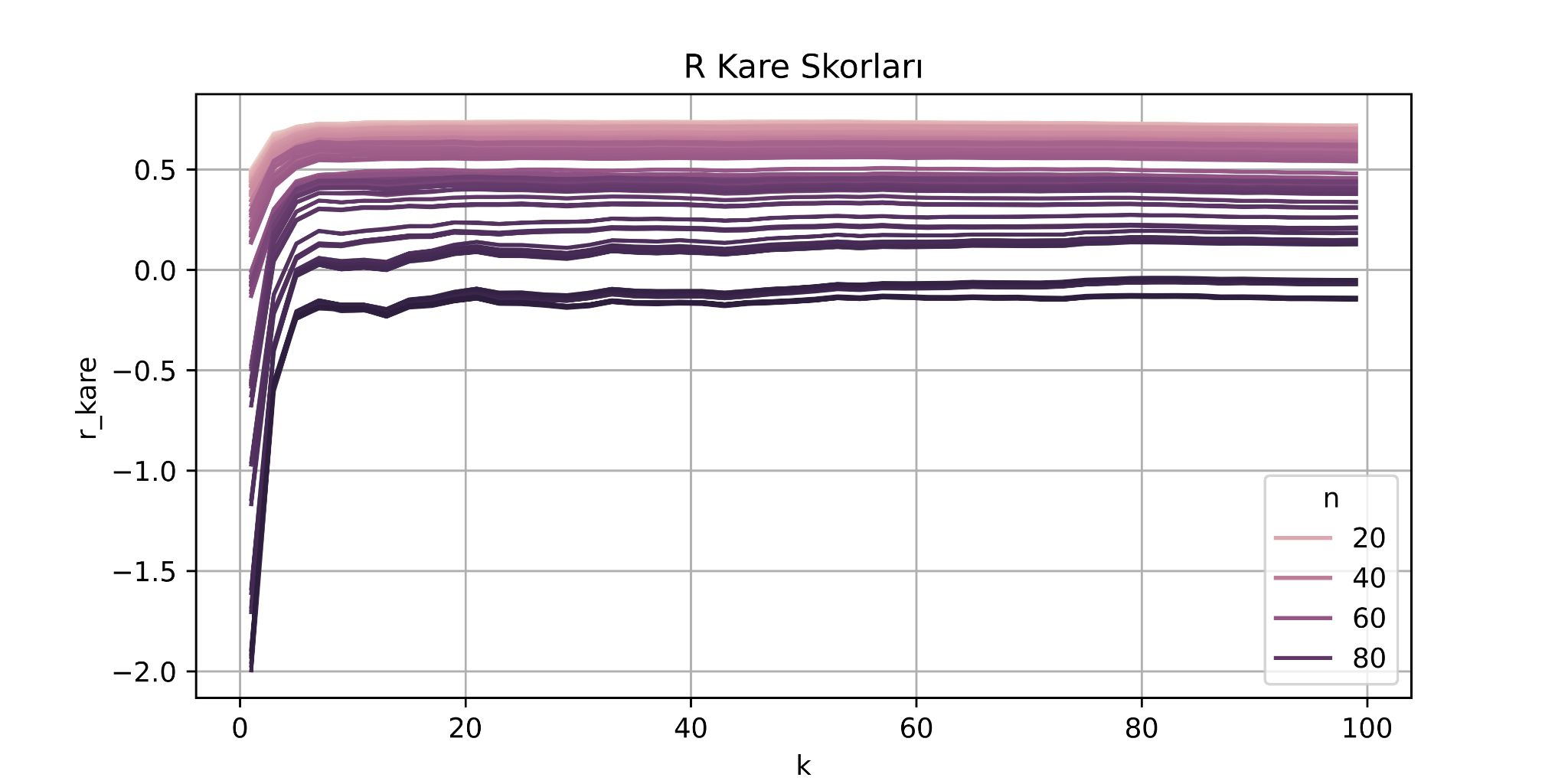
**# Sonuçları dataframe'e ekle**

**cross\_validation\_df = cross\_validation\_df.\_append({"k": k, "n": n, "r\_kare": r\_kare, "mse": mse, "mae": mae}, ignore\_index=True)**

**print("k:", k, "n:", n, "R Kare:", r\_kare, "MSE:", mse, "MAE:", mae)**

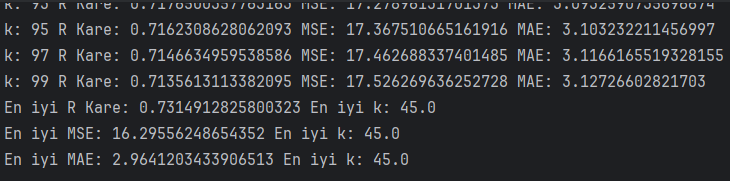
Bu kod 2-100 arasında n değerleri ve 1-100 arasındaki tek sayı k değerlerini dolaşıp üç ölçeğimizi de hesaplayıp en yüksek değerlerin k ve n sayılarını bulmamızı sağlıyor. Random seed olayını devreye katmadım. Bunun için bir for döngüsü daha açıp orada da birkaç defa birkaç farklı random state ile deneme yapmam gerekiyordu.

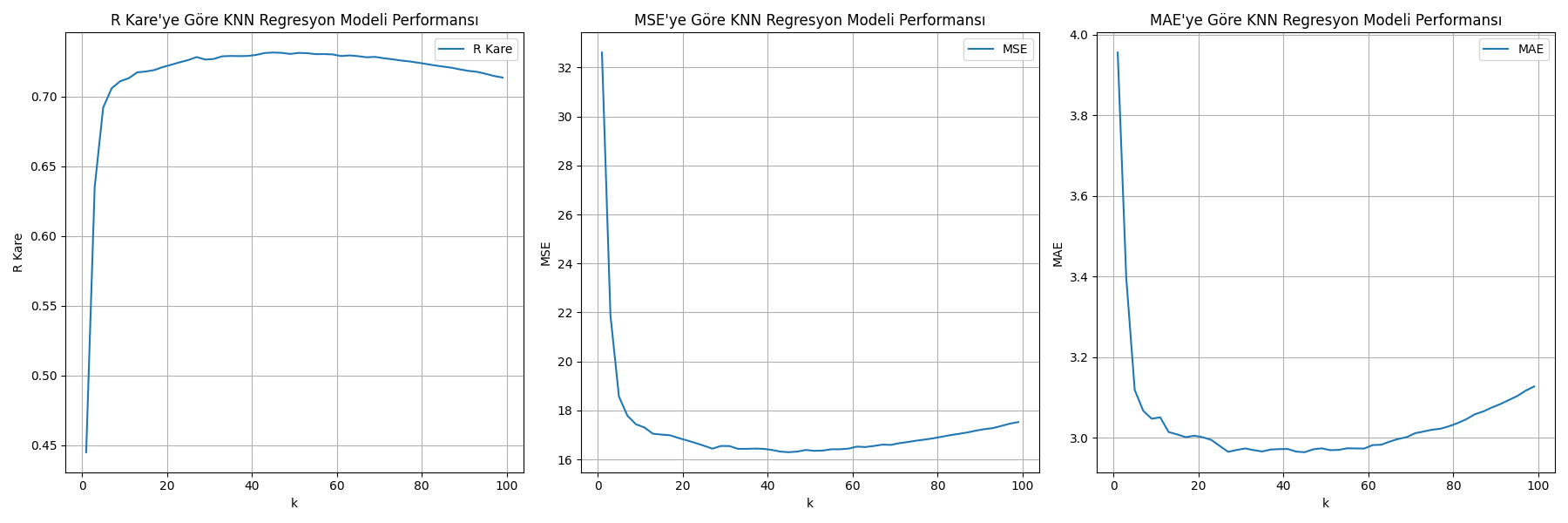




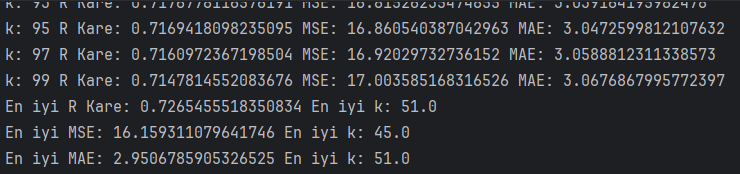
Sonuçları almam yaklaşık 30 dakika sürdü ve anlamaya çalışırken bu yolu kullanarak işin içinden çıkamayacağımı ve daha önemlisi yanlış yaptığımı fark ettim. Cross validation benim modelimi test eden bir şey ancak ben testimin parametreleri ile modelimi en iyi şekilde göstermesi için oynamaya çalışıyordum. Daha doğrusu modelimi en iyi gösteren parametreyi bulmaya çalışıyordum. Bu yanlış bir yol. Genelde cross validation için 5 veya 10 fold kullanıldığını öğrendim ve tekrar denedim. Yukarıda 3.2 KNN kısmında tanıttığım kod bu denememde kullandığım kod.

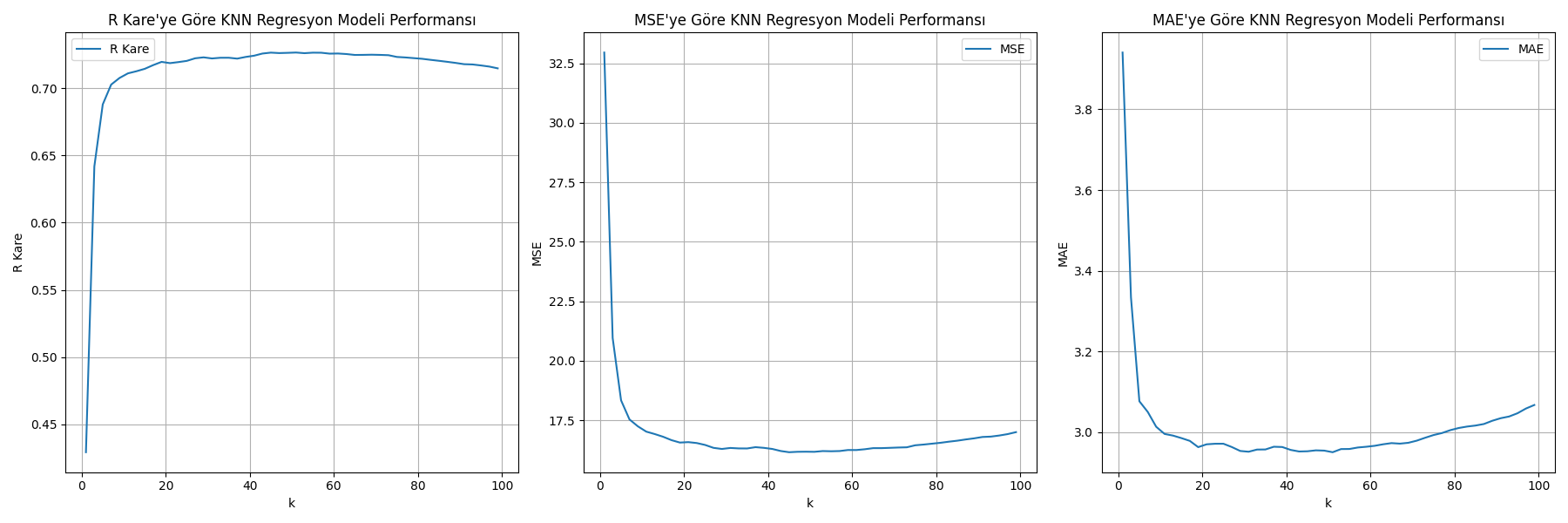
5 Fold:





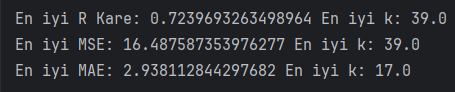
10 Fold:

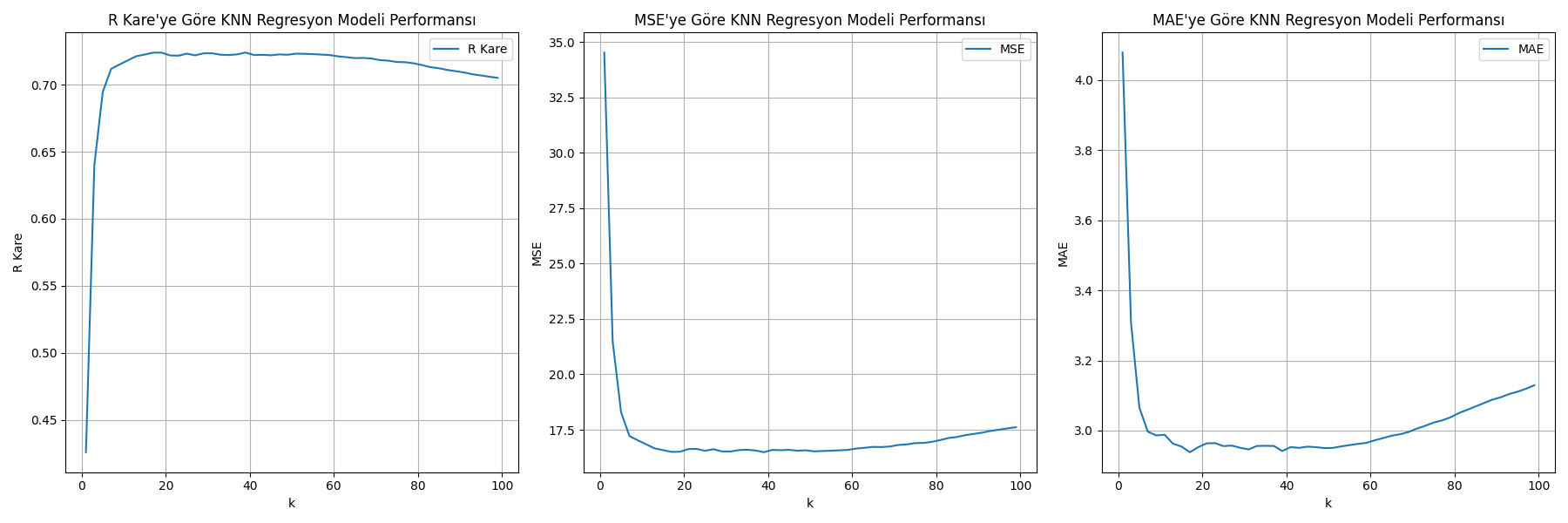




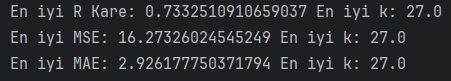
K-fold kontrol ederken katlama sayısını 5 veya 10 yapmamız sonuçlarımızı değiştirdi. 10 katlama sayısında da farklı testlerde farklı k değerleri daha uygun çıkıyor. Üstelik kullandığım seed de bunda bir etken. random-state’i 308 verdiğimde bu değerleri aldım. Değiştirdiğimde ise farklı değerler aldım.

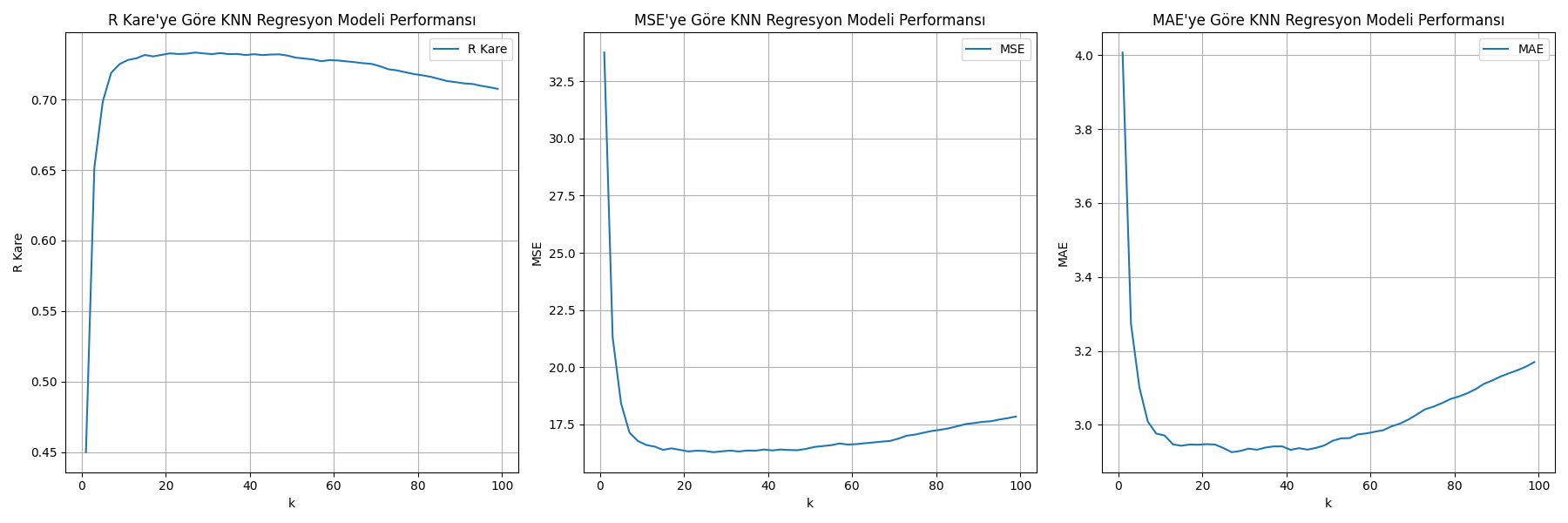
Random State 3 - Fold 10:



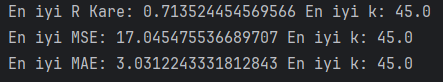


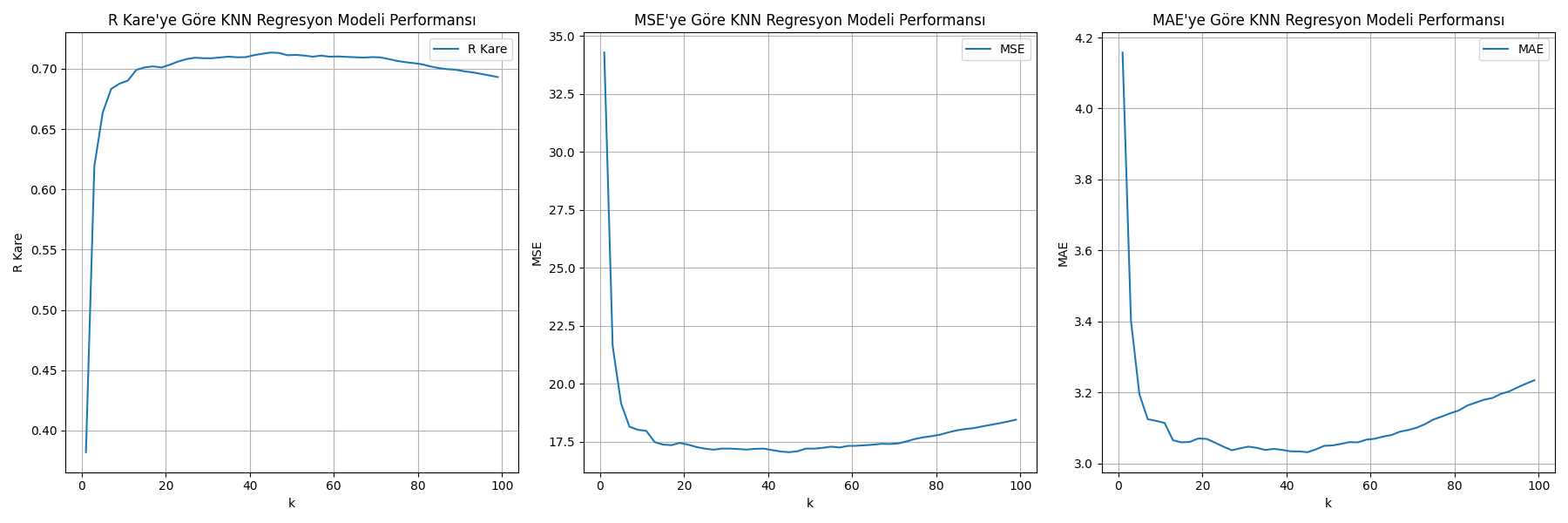
Random State 3 - Fold 5:





Birkaç defa daha farklı şeyler denedim ve 5 fold yaptığımda tüm testlerin aynı k değerini verdiğini gözlemledim. Aksi karşıma çıkmadı, belki çıkabilirdi ancak daha fazla kurcalamayı bıraktım. İnternetteki pek çok örnekte de sabit bir random state seçiliyor ve modeller o state ile kuruluyor. Random state 300 ve fold sayısı 5 olduğunda yaptığım testte k sayısının 45 olması gerektiği sonucuna ulaştım. Zaten sonrasında skorlar düşüyor.





Son olarak R Kare, MSE ve MAE gibi skorlar tek bir modele ait değil modelin tüm aşamalarına ait değerler. Bunların ortalamasını alıyorum. Medyan veya mod da alınabilirdi ancak onları denemeye gerek duymadım.

İşleyişi KNN ile öğrendiğim için Random Forest yöntemini kolaylıkla gerçekleştirdim. Elim alıştı ve yeni bir anlayış kazandım. Buraya ödevi bitirdikten sonra bir ekleme yapıyorum. Bu konuların 9. slaytta olduğunu bilmiyordum çünkü henüz işlememiştik. Ödevin sonuna doğru son slaytın ismini görünce baktım ki tüm bu konuları aslında gelecekte işleyecekmişiz. Erkenden öğrenmiş oldum ancak zordu. Okulda gördükten sonra bu ödevi yapsaydım tüm bu süreç çok daha hızlı geçerdi.

ANN geçtiğimde ise deneyecek çok parametre vardı. Katman sayısı, katmanlardaki nöron sayıları, aktivasyon fonksiyonu, solver, alpha, learning rate değişimi, learning rate, batch size, max iteration. Bunlardan solver, alpha ve batch size bilmediğim şeylerdi. Solverların ağırlık güncelleme algoritmaları olduğunu öğrendim. Alpha overfitting’i azaltan bir değer iken batch size her iterasyonda veri setinin ne kadarının kullanılacağı belirten bir değer. Bunların hepsini denedim ve iç içe 6 for döngüsü içeren bir method yaptım ve ortaya böyle bir şey çıktı:

**import seaborn as sns**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split, cross\_val\_score**

**from sklearn.neural\_network import MLPRegressor**

**from sklearn.metrics import make\_scorer, r2\_score, mean\_squared\_error, mean\_absolute\_error**

**from sklearn.preprocessing import MinMaxScaler**

**import random**

**# Veri setini yükle**

**df = sns.load\_dataset("mpg").dropna(subset=["horsepower"])**

**# Özellik ve hedef seçimi**

**ozellik = df[["horsepower", "acceleration", "weight"]]**

**hedef = df["mpg"]**

**# Eğitim-test ayrımı**

**random\_state = 300 #random.randint(0, 999)**

**ozellik\_egitim, ozellik\_test, hedef\_egitim, hedef\_test = train\_test\_split(ozellik, hedef, test\_size=0.1, random\_state=random\_state)**

**# Normalizasyon**

**min\_max\_normalizator = MinMaxScaler()**

**ozellik\_egitim\_normalize = min\_max\_normalizator.fit\_transform(ozellik\_egitim)**

**ozellik\_test\_normalize = min\_max\_normalizator.transform(ozellik\_test)**

**# Skorlayıcılar**

**mse\_scorer = make\_scorer(mean\_squared\_error, greater\_is\_better=False)**

**mae\_scorer = make\_scorer(mean\_absolute\_error, greater\_is\_better=False)**

**r\_kare\_scorer = make\_scorer(r2\_score, greater\_is\_better=True)**

**# Hiperparametre adayları**

**hidden\_layer\_sizes\_adayları = []**

**# 1'den 5'e kadar katman sayısı**

**for katman\_sayisi in range(1, 6): # 1, 2, 3, 4, 5 katman**

**for düğüm\_sayisi in range(10, 101, 10): # Her katmanda 10'dan 100'e kadar**

**# Aynı düğüm sayısını tüm katmanlara uygula**

**hidden\_layer = tuple([düğüm\_sayisi] \* katman\_sayisi)**

**hidden\_layer\_sizes\_adayları.append(hidden\_layer)**

**activation\_adayları = ["identity", "logistic", "tanh", "relu"]**

**solver\_adayları = ["lbfgs", "sgd", "adam"]**

**alpha\_adayları = [0.0001, 0.001, 0.01, 0.1, 1]**

**learning\_rate\_init\_adayları = [0.0001, 0.001, 0.01, 0.1, 1]**

**max\_iter\_adayları = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]**

**# Cross-validation sonuçlarını saklayacağımız DataFrame**

**cross\_validation\_sonuçlar = pd.DataFrame(columns=["hidden\_layer\_sizes", "activation", "solver", "alpha", "learning\_rate\_init", "max\_iter", "r\_kare", "mse", "mae"])**

**max\_iter = 1000**

**# Cross-validation ile sonuçların hesaplanması**

**for hidden\_layers in hidden\_layer\_sizes\_adayları:**

**for activation\_adayı in activation\_adayları:**

**for solver\_adayı in solver\_adayları:**

**for alpha\_degeri in alpha\_adayları:**

**for learning\_rate\_init\_adayı in learning\_rate\_init\_adayları:**

**for max\_iter\_adayı in max\_iter\_adayları:**

**ann\_model = MLPRegressor(hidden\_layer\_sizes=hidden\_layers,**

**activation=activation\_adayı,**

**solver=solver\_adayı,**

**alpha=alpha\_degeri,**

**learning\_rate\_init=learning\_rate\_init\_adayı,**

**max\_iter=max\_iter\_adayı,**

**random\_state=random\_state)**

**# R Kare**

**r\_kare\_skorları = cross\_val\_score(ann\_model, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=r\_kare\_scorer)**

**r\_kare = r\_kare\_skorları.mean()**

**# MSE**

**mse\_skorları = cross\_val\_score(ann\_model, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=mse\_scorer)**

**mse = (-1) \* mse\_skorları.mean()**

**# MAE**

**mae\_skorları = cross\_val\_score(ann\_model, ozellik\_egitim\_normalize, hedef\_egitim, cv=5, scoring=mae\_scorer)**

**mae = (-1) \* mae\_skorları.mean()**

**# Sonuçları dataframe'e ekle**

**cross\_validation\_sonuçlar = cross\_validation\_sonuçlar.\_append({**

**"hidden\_layer\_sizes": hidden\_layers,**

**"activation": activation\_adayı,**

**"solver": solver\_adayı,**

**"alpha": alpha\_degeri,**

**"learning\_rate\_init": learning\_rate\_init\_adayı,**

**"max\_iter": max\_iter\_adayı,**

**"r\_kare": r\_kare,**

**"mse": mse,**

**"mae": mae**

**}, ignore\_index=True)**

**print("hidden\_layer\_sizes:", hidden\_layers,**

**"activation:", activation\_adayı,**

**"solver:", solver\_adayı,**

**"alpha:", alpha\_degeri,**

**"learning\_rate\_init:", learning\_rate\_init\_adayı,**

**"max\_iter:", max\_iter\_adayı,**

**"R Kare:", r\_kare,**

**"MSE:", mse,**

**"MAE:", mae)**

**# R kare yöntemine göre en iyi hiperparametreleri bulma**

**en\_iyi\_r\_kare = cross\_validation\_sonuçlar["r\_kare"].max()**

**en\_iyi\_r\_kare\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["r\_kare"] == en\_iyi\_r\_kare].iloc[0]**

**en\_iyi\_r\_kare\_hidden\_layers = en\_iyi\_r\_kare\_satir["hidden\_layer\_sizes"]**

**en\_iyi\_r\_kare\_activation = en\_iyi\_r\_kare\_satir["activation"]**

**en\_iyi\_r\_kare\_solver = en\_iyi\_r\_kare\_satir["solver"]**

**en\_iyi\_r\_kare\_alpha = en\_iyi\_r\_kare\_satir["alpha"]**

**en\_iyi\_r\_kare\_learning\_rate\_init = en\_iyi\_r\_kare\_satir["learning\_rate\_init"]**

**en\_iyi\_r\_kare\_max\_iter = en\_iyi\_r\_kare\_satir["max\_iter"]**

**print("En iyi R Kare:", en\_iyi\_r\_kare,**

**"En iyi hidden\_layer\_sizes:", en\_iyi\_r\_kare\_hidden\_layers,**

**"En iyi activation:", en\_iyi\_r\_kare\_activation,**

**"En iyi solver:", en\_iyi\_r\_kare\_solver,**

**"En iyi alpha:", en\_iyi\_r\_kare\_alpha,**

**"En iyi learning\_rate\_init:", en\_iyi\_r\_kare\_learning\_rate\_init,**

**"En iyi max\_iter:", en\_iyi\_r\_kare\_max\_iter)**

**# MSE yöntemine göre en iyi hiperparametreleri bulma**

**en\_iyi\_mse = cross\_validation\_sonuçlar["mse"].min()**

**en\_iyi\_mse\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["mse"] == en\_iyi\_mse].iloc[0]**

**en\_iyi\_mse\_hidden\_layers = en\_iyi\_mse\_satir["hidden\_layer\_sizes"]**

**en\_iyi\_mse\_activation = en\_iyi\_mse\_satir["activation"]**

**en\_iyi\_mse\_solver = en\_iyi\_mse\_satir["solver"]**

**en\_iyi\_mse\_alpha = en\_iyi\_mse\_satir["alpha"]**

**en\_iyi\_mse\_learning\_rate\_init = en\_iyi\_mse\_satir["learning\_rate\_init"]**

**en\_iyi\_mse\_max\_iter = en\_iyi\_mse\_satir["max\_iter"]**

**print("En iyi MSE:", en\_iyi\_mse,**

**"En iyi hidden\_layer\_sizes:", en\_iyi\_mse\_hidden\_layers,**

**"En iyi activation:", en\_iyi\_mse\_activation,**

**"En iyi solver:", en\_iyi\_mse\_solver,**

**"En iyi alpha:", en\_iyi\_mse\_alpha,**

**"En iyi learning\_rate\_init:", en\_iyi\_mse\_learning\_rate\_init,**

**"En iyi max\_iter:", en\_iyi\_mse\_max\_iter)**

**# MAE yöntemine göre en iyi hiperparametreleri bulma**

**en\_iyi\_mae = cross\_validation\_sonuçlar["mae"].min()**

**en\_iyi\_mae\_satir = cross\_validation\_sonuçlar[cross\_validation\_sonuçlar["mae"] == en\_iyi\_mae].iloc[0]**

**en\_iyi\_mae\_hidden\_layers = en\_iyi\_mae\_satir["hidden\_layer\_sizes"]**

**en\_iyi\_mae\_activation = en\_iyi\_mae\_satir["activation"]**

**en\_iyi\_mae\_solver = en\_iyi\_mae\_satir["solver"]**

**en\_iyi\_mae\_alpha = en\_iyi\_mae\_satir["alpha"]**

**en\_iyi\_mae\_learning\_rate\_init = en\_iyi\_mae\_satir["learning\_rate\_init"]**

**en\_iyi\_mae\_max\_iter = en\_iyi\_mae\_satir["max\_iter"]**

**print("En iyi MAE:", en\_iyi\_mae,**

**"En iyi hidden\_layer\_sizes:", en\_iyi\_mae\_hidden\_layers,**

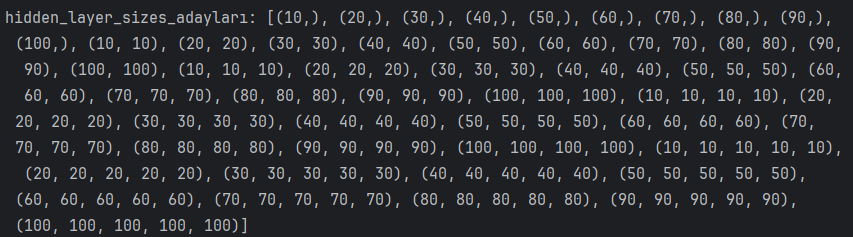
**"En iyi activation:", en\_iyi\_mae\_activation,**

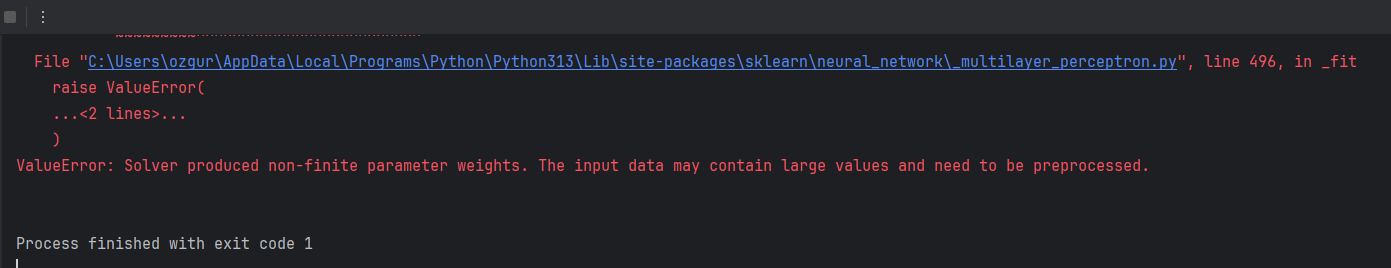
**"En iyi solver:", en\_iyi\_mae\_solver,**

**"En iyi alpha:", en\_iyi\_mae\_alpha,**

**"En iyi learning\_rate\_init:", en\_iyi\_mae\_learning\_rate\_init,**

**"En iyi max\_iter:", en\_iyi\_mae\_max\_iter)**

****



Çalışmadı ve hata aldım. Nasıl daha iyi yapabileceğimi araştırırken Grid Search adında bir yöntem olduğunu öğrendim.

**# Grid Search için parametreler**

**param\_grid = {**

**"hidden\_layer\_sizes": hidden\_layer\_sizes\_adayları,**

**"activation": activation\_adayları,**

**"solver": solver\_adayları,**

**"alpha": alpha\_adayları,**

**"learning\_rate\_init": learning\_rate\_init\_adayları,**

**"max\_iter": max\_iter\_adayları**

**}**

**# Grid Search ile model optimizasyonu**

**grid\_search = GridSearchCV(**

**MLPRegressor(random\_state=random\_state),**

**param\_grid,**

**cv=5,**

**scoring=scorers,**

**refit='r2' # R-kare skoruna göre en iyi modeli seç**

**)**

**# Grid Search'ü çalıştır**

**grid\_search.fit(ozellik\_egitim\_normalize, hedef\_egitim)**

**# Detaylı sonuçları DataFrame'e aktar**

**for params, mean\_score, scores in zip(**

**grid\_search.cv\_results\_['params'],**

**grid\_search.cv\_results\_['mean\_test\_r2'],**

**grid\_search.cv\_results\_['mean\_test\_neg\_mse']**

**):**

**cross\_validation\_sonuçlar = cross\_validation\_sonuçlar.\_append({**

**"hidden\_layer\_sizes": params['hidden\_layer\_sizes'],**

**"alpha": params['alpha'],**

**"r\_kare": mean\_score,**

**"mse": -scores, # Negatif MSE'yi pozitife çevir**

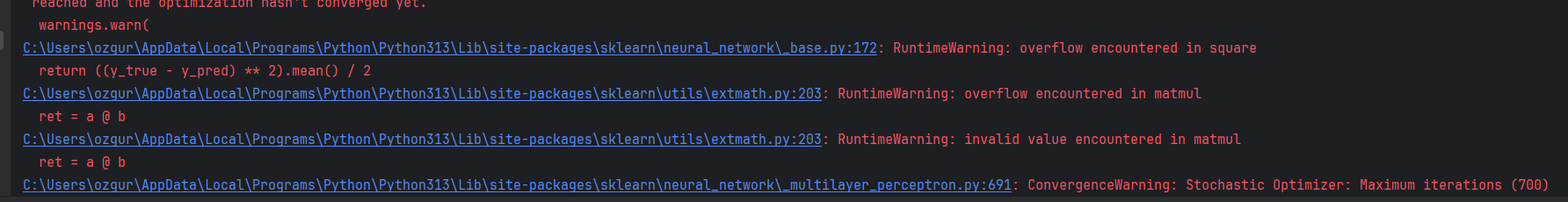
**}, ignore\_index=True)**

**# En iyi modelin parametrelerini yazdır**

**print("En iyi parametreler:", grid\_search.best\_params\_)**

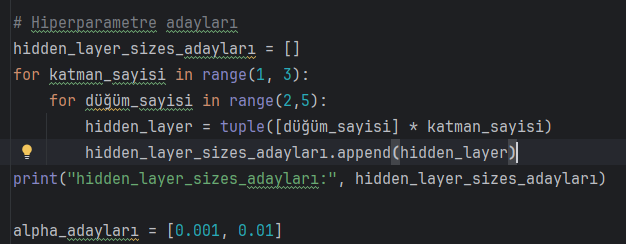
**print("En iyi R-kare skoru:", grid\_search.best\_score\_)**

Kodun gövdesini yukarıdaki blok ile değiştirdim. Yaklaşık 30 dakika boyunca sürekli hata aldım ancak kod çalışmaya devam ediyordu.

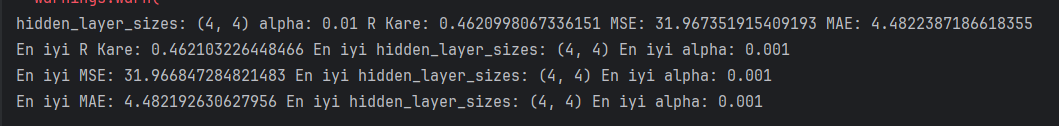


Beklemenin bir şey değiştirmeyeceğini düşünüp iptal ettim ve daha küçük, daha sınırlı bir yolu tercih ettim. Birkaç saat bekleseydim farklı şeyler olabilirdi ancak beklemedim.

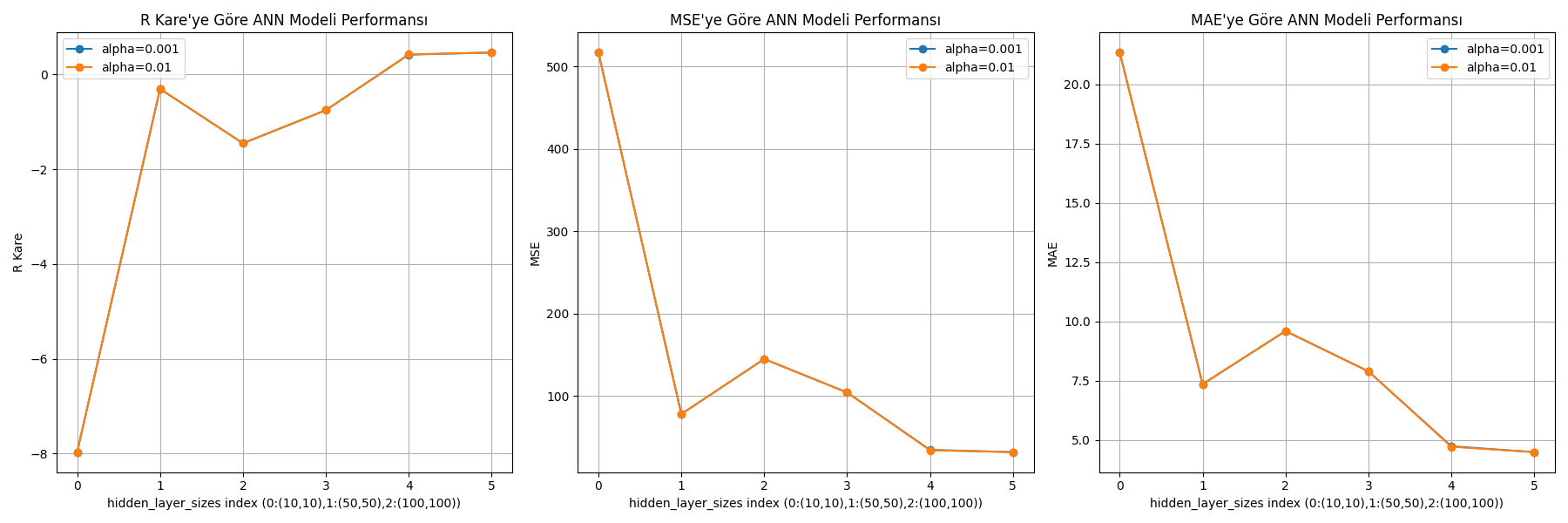
Grid Search ile grafik oluşturmak için aradaki aşamaların verisini alamadım bu yüzden for döngüsü ile devam ettim. Bir sonuç görmek için saatlerce beklememek için max iteration’u 1000’e sabitleyip geri kalan parametrelerde kütüphanenin default değerlerini kullandım.



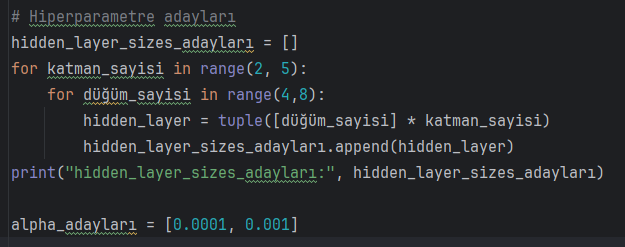


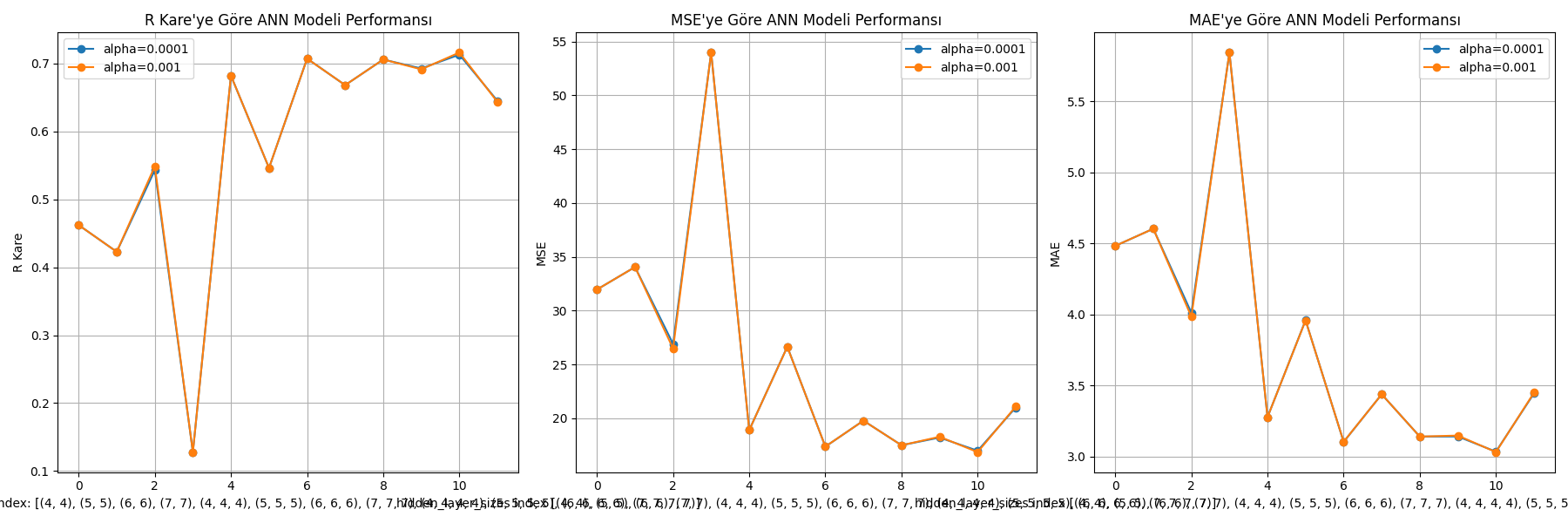


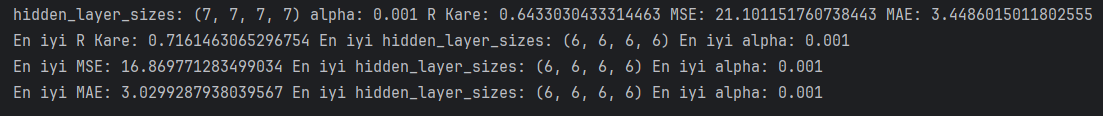
Elde ettiğim grafikte alpha = 0.01 değeri o kadar baskındı ki en iyi alpha değeri gözükmüyordu.



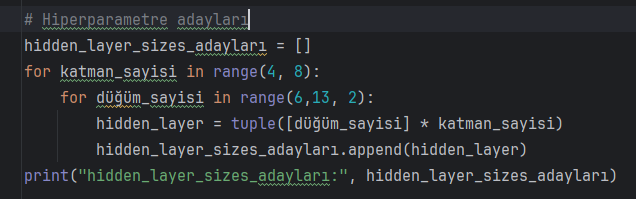
Yeni denememe alpha değerini 0.001 veya 0.0001 denemeyi seçtim. Daha sonra iki veya daha fazla layerlı ilerlemeye ve layerlarımdaki nöron sayımı 4’ten büyük yapmayı tercih ettim çünkü grafiğe göre 4. index’ten yani (3,3)ten sonra bir ilerleyiş vardı. Bu ilerleyiş tekrar düşene kadar devam etmem gerektiğini düşündüm. Global maksimumu bulduğumdan emin olamasam da en yakın lokal maksimumu bulabilirdim.

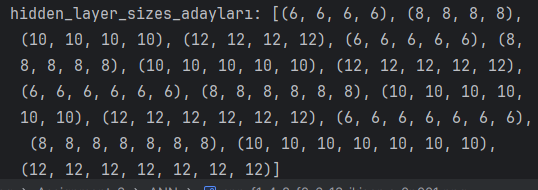


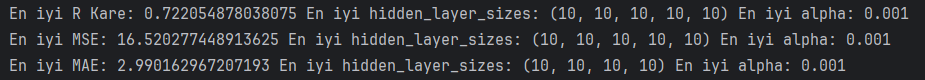


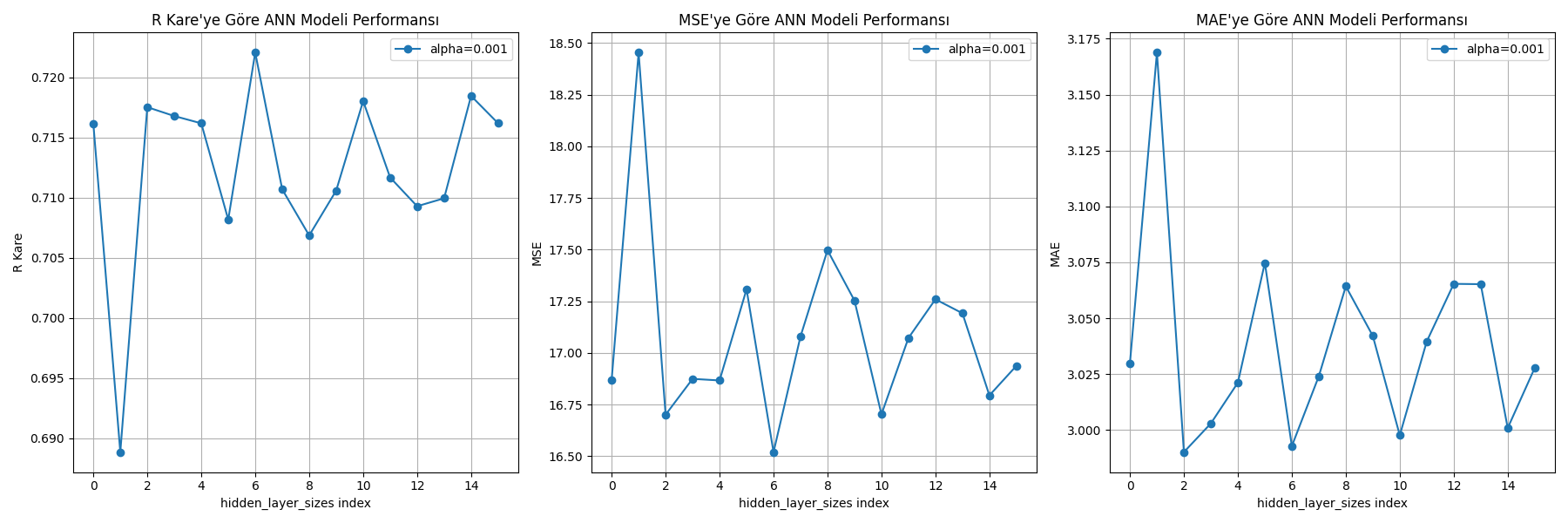


Alpha değerini daha da düşürmenin bir işe yaramadığını gözlemledim. Hala en iyi alpha 0.001. Artık bunu denememeye karar verdim. Her layer içindeki nöron sayısı arttıkça bir süre ilerleme yaşanmış ancak 6’dan sonra düşüş yaşanmış. Biraz dikkatli baktığımda tek sayılarda her zaman bir düşüş olduğunu, çift sayılarda da bir artış olduğunu fark ettim. Sadece çift sayıları seçerek bir daha denedim. Layer sayısının artışı da ilerleme sağlamıştı. Bu sefer daha uzun sürecek bir işlemle daha uzun beklemeye karar verdim.









Katman sayısının üst sınırı 7, her katmandaki nöron sayısının üst sınırı 12 olmasına rağmen artış 4 katman ve katman başı 10 nöronda sınırlı kalmış. En önemlisi bir önceki modele kıyasla R Kare değeri sadece 0.1 artmış. (4,4) sonrası neredeyse 2 katına çıkmıştı ancak burada 0.1 artmış. Bu noktada durmaya karar verdim.

Diğer parametreleri de hesaba katmak aşırı bir işlem yükü oluşturuyordu, modelimi daha da kompleks hale getirmedim. Zaten 0.7 R Kare değeri ile KNN ve Random Forest ile aynı performansa sahipti.

Son eklemelerimi de yapıp raporumu bitirmek isterim. Bu ödevin yapılması için 9. slaytın da işlenmiş olması gerekirdi. Araştırıp öğrendiğim konuların aslında görmemiz gereken ancak daha görmediğimiz şeyler olduğunu sonlara doğru fark ettim. Ödevin veriliş ve son teslim tarihlerinin zamanlaması biraz yanlış olmuş. “Model karşılaştırması yapmak için 9. slayttaki konunun işlenmesi lazım, önden bakabilir veya dersin işlenmesini bekleyebilirsiniz” diye bir uyarı olması iyi olabilirdi. Ancak sakaide tüm slaytlar önceden açık olduğu için bir sonraki hafta işlememiz gereken 9. slaytı açıp okudum. Değerlendirme yollarımın da hali hazırda ders konusu olduğunu gördüğüm için sevindim ve ekstra bir ekleme yapmadan ödevimi sonlandırdım. Bu ödeve yaklaşık 20 saat gibi bir zaman ayırdım. Çok ama çok şey öğrendim, elimi bu alanda kirletmiş ve bir şeyler üretmiş oldum. Çok öğretici ve kaliteli bir ödevdi.

## 6- Kaynaklar

Ders Slaytları

Chat GPT - Gemini - Claude

<https://www.ibm.com/topics/knn>

<https://www.geeksforgeeks.org/how-to-find-the-optimal-value-of-k-in-knn/>

<https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a>

<https://scikit-learn.org/stable/modules/cross_validation.html>

<https://www.geeksforgeeks.org/cross-validation-machine-learning/>

<https://machinelearningmastery.com/k-fold-cross-validation/>

<https://www.ibm.com/topics/random-forest>

<https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/>

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html>

<https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/>

<https://www.ibm.com/topics/neural-networks>

<https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/>