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-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-ann
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"])
```

```
Eksik değerlerin sayısı:
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
name            0
```

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())
```

```
Veri seti istatistiksel özeti:
              mpg   cylinders  displacement  horsepower       weight  \
count  392.000000  392.000000    392.000000  392.000000   392.000000
mean    23.445918    5.471939    194.411990  104.469388  2977.584184
std      7.805007    1.705783    104.644004   38.491160   849.402560
min      9.000000    3.000000     68.000000   46.000000  1613.000000
25%     17.000000    4.000000    105.000000   75.000000  2225.250000
50%     22.750000    4.000000    151.000000   93.500000  2803.500000
75%     29.000000    8.000000    275.750000  126.000000  3614.750000
max     46.600000    8.000000    455.000000  230.000000  5140.000000

       acceleration  model_year
count    392.000000  392.000000
mean      15.541327   75.979592
std        2.758864    3.683737
min        8.000000   70.000000
25%       13.775000   73.000000
50%       15.500000   76.000000
75%       17.025000   79.000000
max       24.800000   82.000000
```

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.

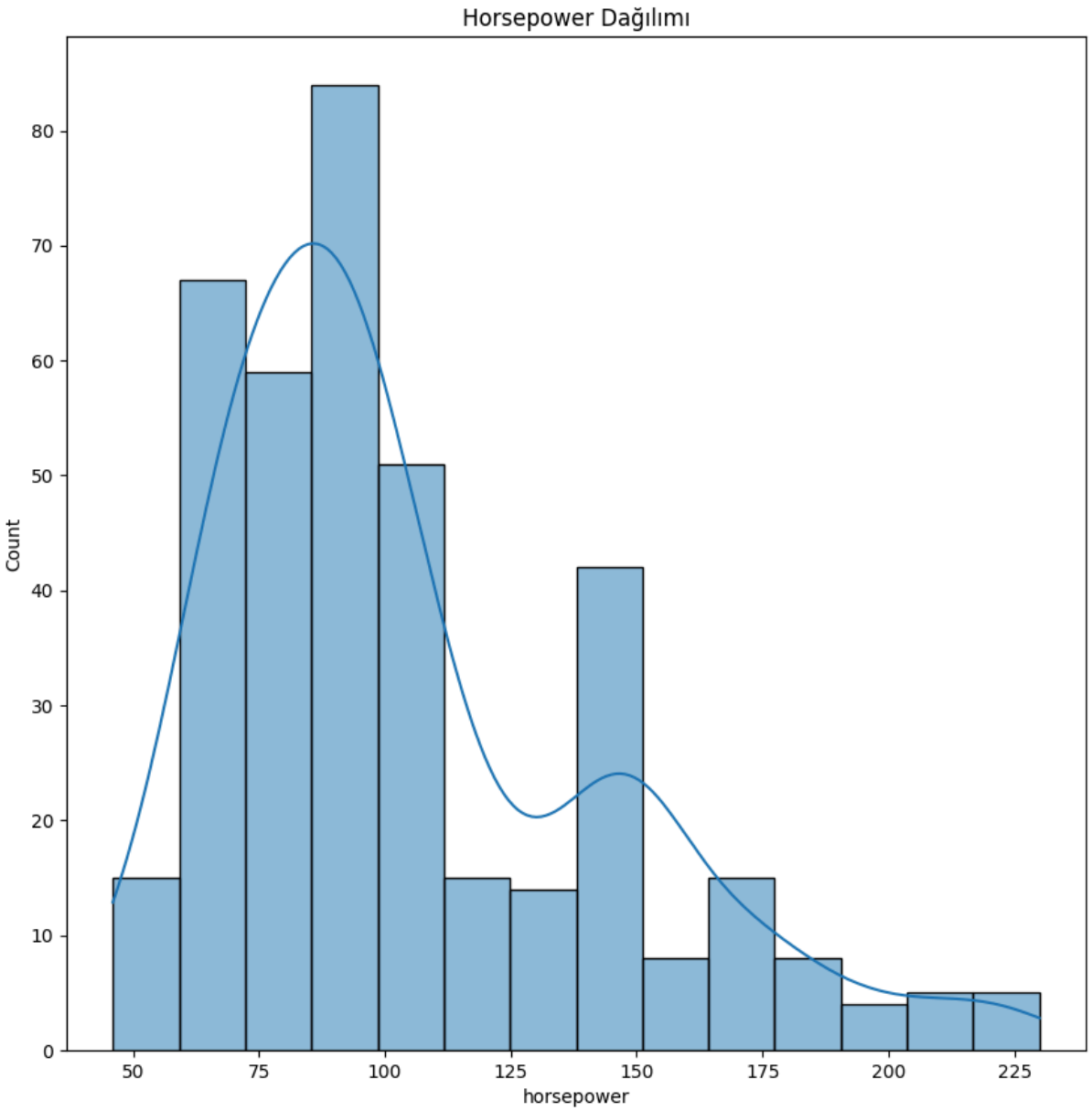
Weight:
Ortalama ~2977.58, std ~849.40, çok geniş bir aralık (1613 ile 5140). Araçların ağırlıklarının da geniş bir yelpazede değiştiğini görüyoruz. %50 medyan ~2803.5 pound, ancak 75% diliminde 3614.75'e çıkıyor. Bu yüksek varyasyon, modellerde normalleştirmenin faydalı olabileceğini düşündürüyor.

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

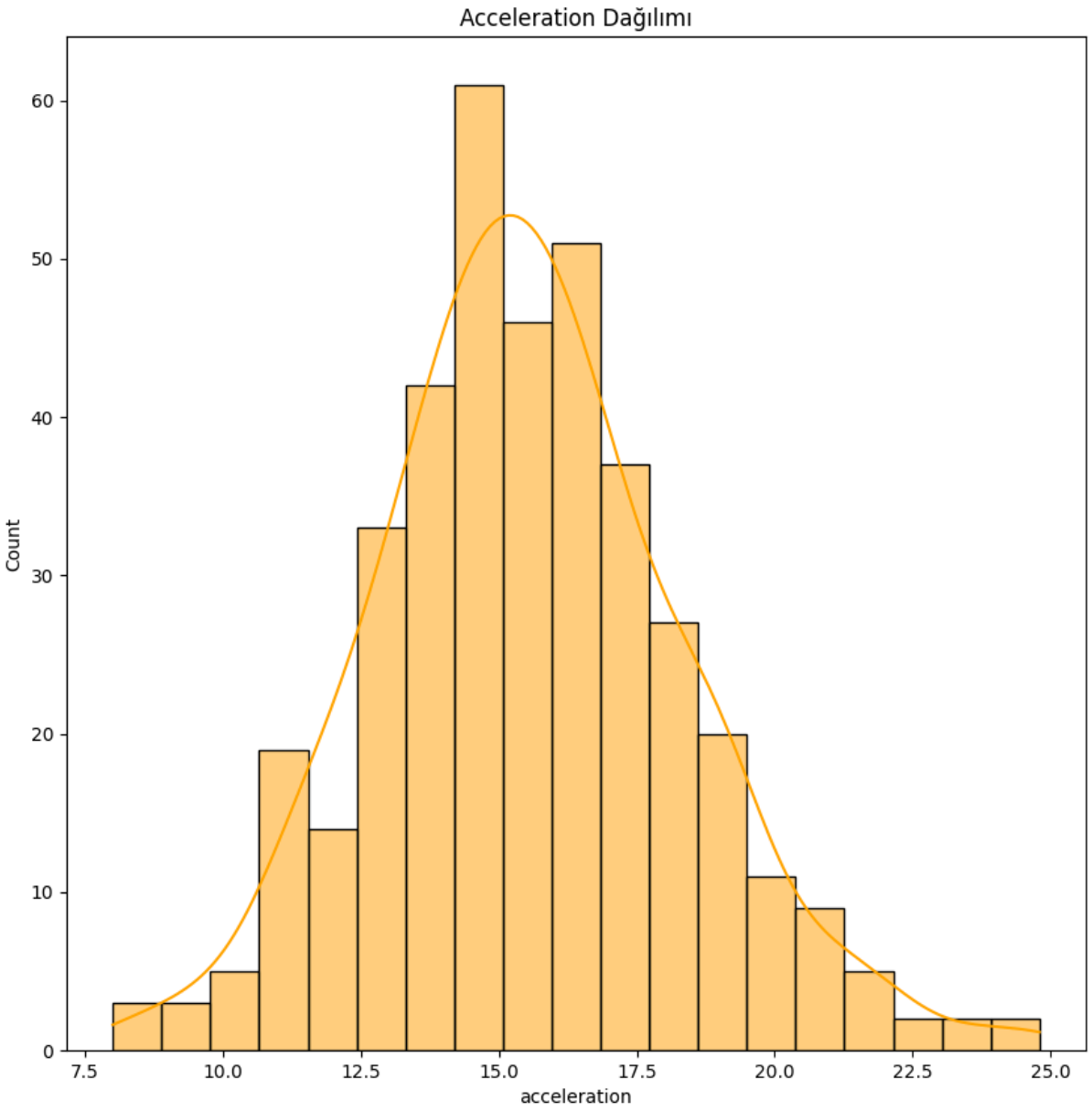
```
# Horsepower
plt.figure(figsize=(10,10))
sns.histplot(df["horsepower"], kde=True)
plt.title("Horsepower Dağılımı")
plt.savefig("horsepower.png")
plt.show()
```



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

3.1.5 Acceleration Histogramı

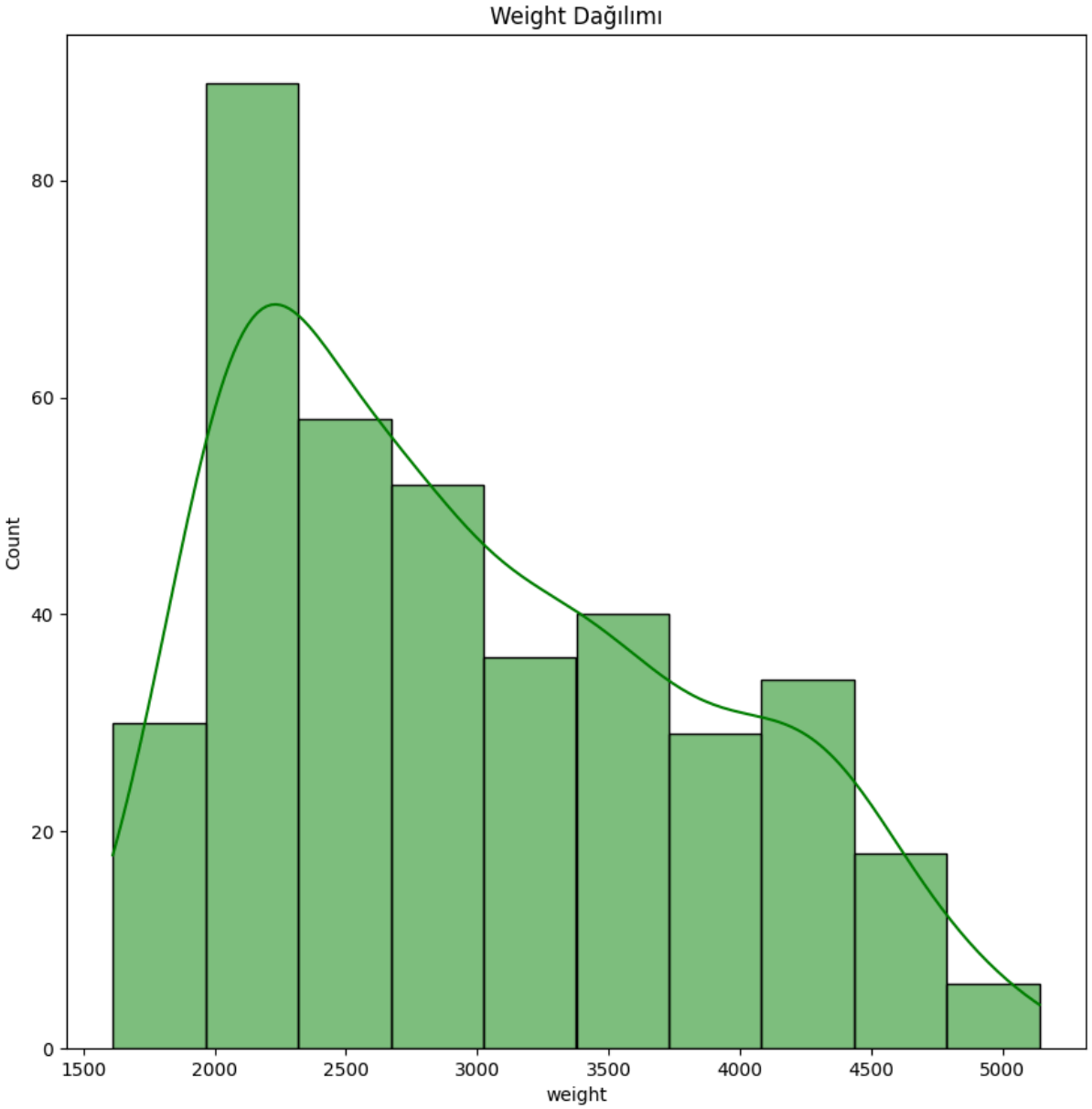
```
# Acceleration
plt.figure(figsize=(10,10))
sns.histplot(df["acceleration"], kde=True, color="orange")
plt.title("Acceleration Dağılımı")
plt.savefig("acceleration.png")
plt.show()
```



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

3.1.6 Weight Histogramı

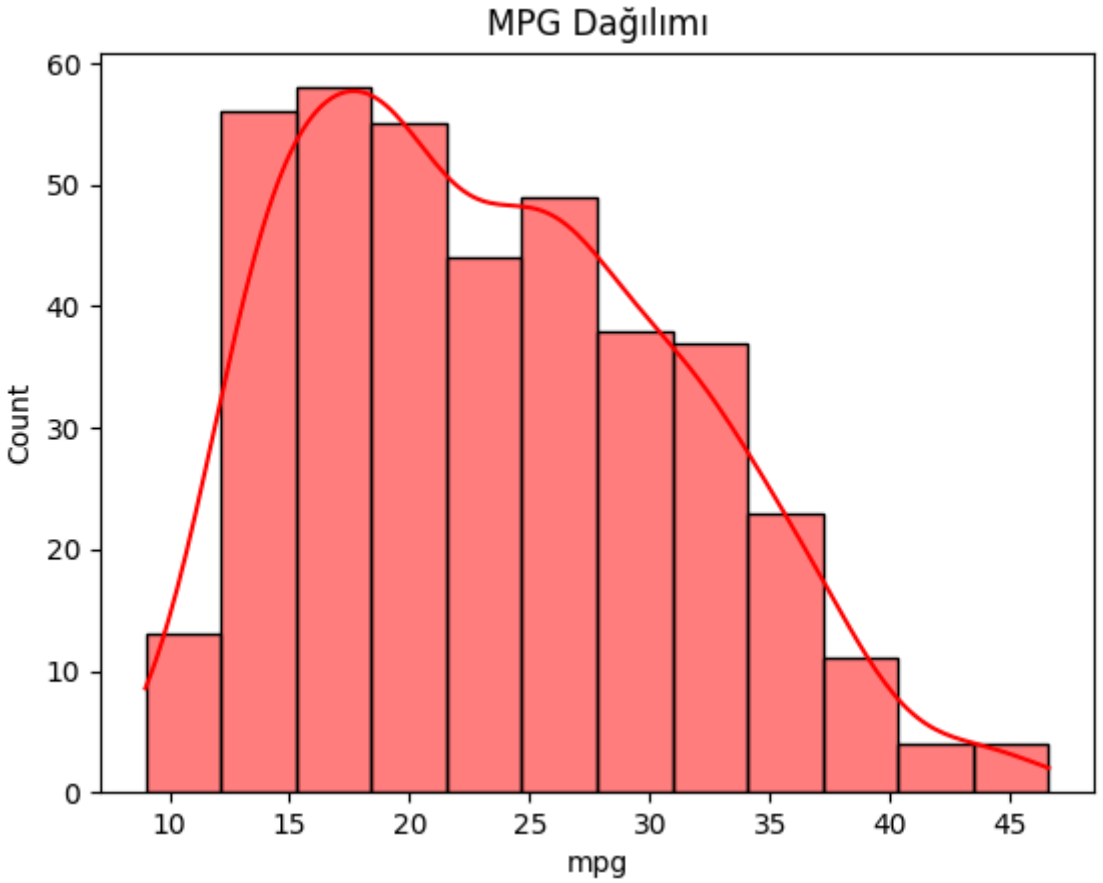
```
# Weight
plt.figure(figsize=(10,10))
sns.histplot(df["weight"], kde=True, color="green")
plt.title("Weight Dağılımı")
plt.savefig("weight.png")
plt.show()
```



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

3.1.7 MPG (Mile Per Gallon) Histogramı

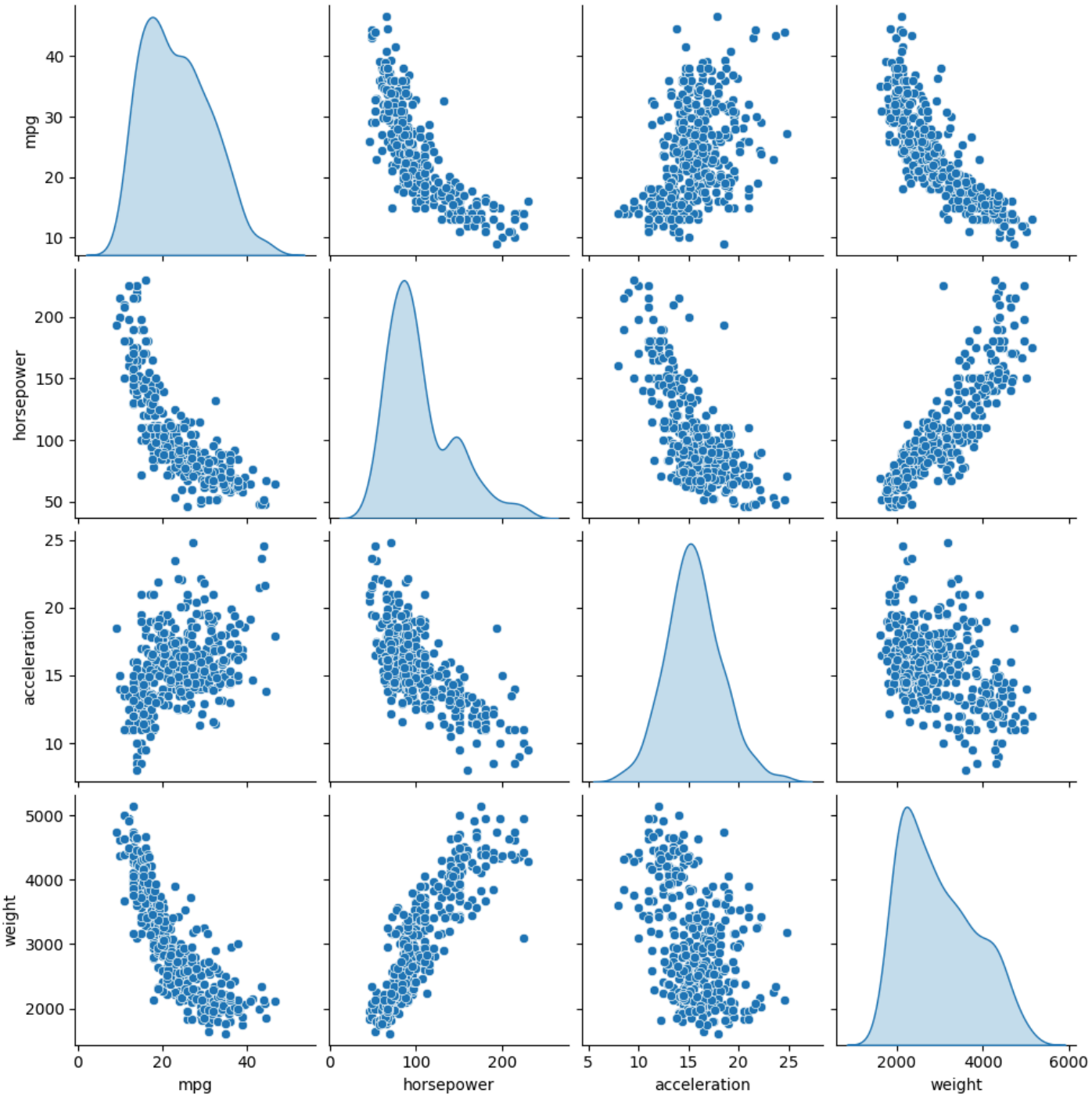
```
# MPG
plt.figure()
sns.histplot(df["mpg"], kde=True, color="red")
plt.title("MPG Dağılımı")
plt.savefig("mpg.png")
plt.show()
```



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

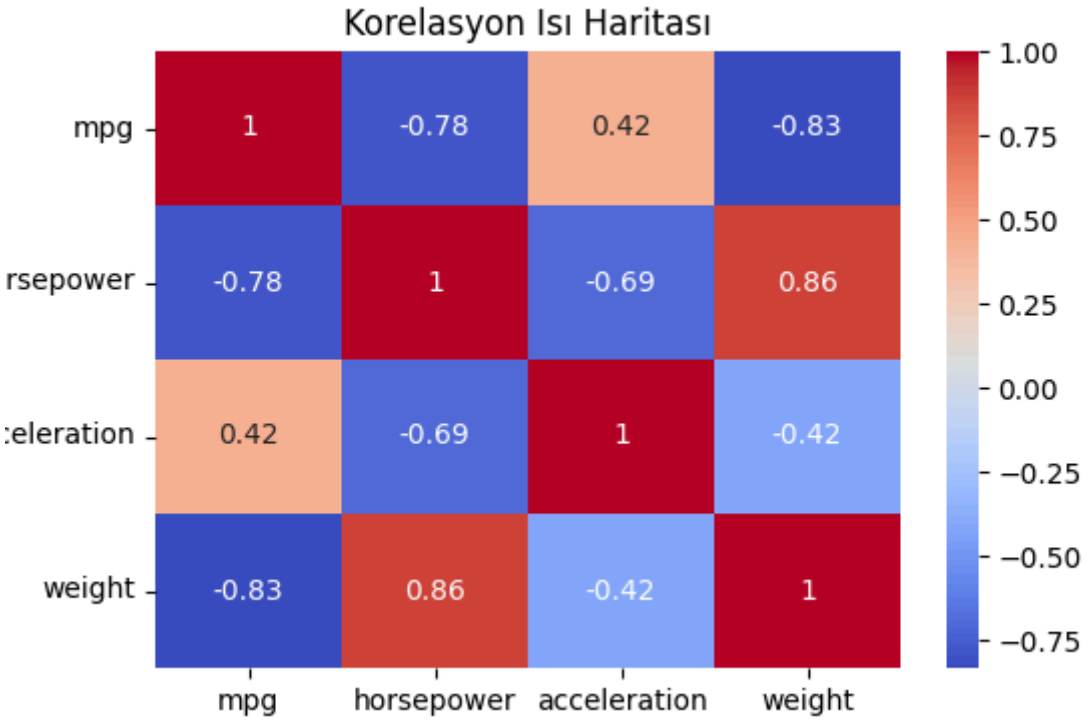
3.1.8 Değişkenler Arası Pair Plot

```
# 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.

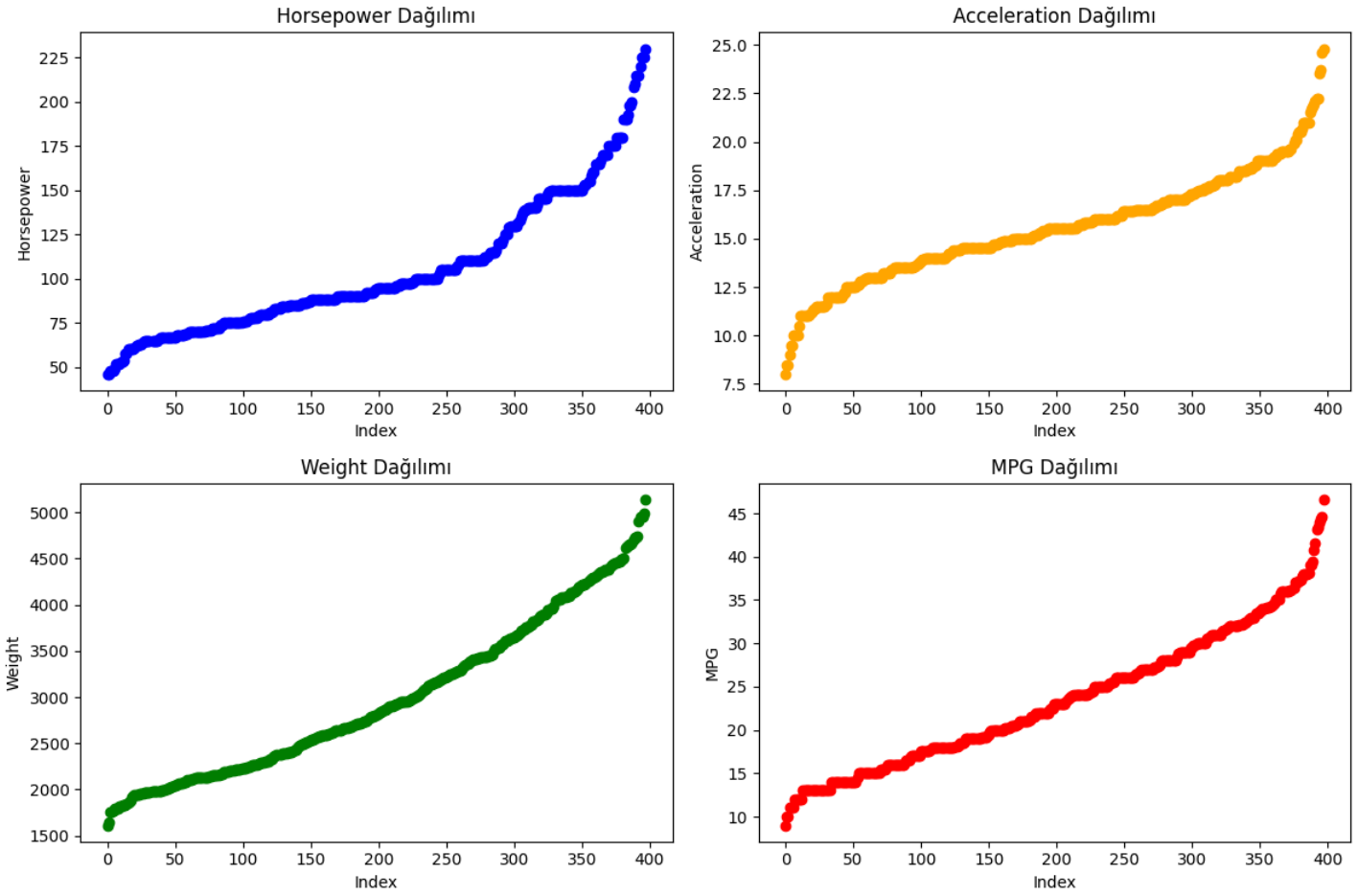
mpg ile **acceleration** arasında **pozitif** bir korelasyon var **(0.42)**. Daha hızlı hızlanan araçların genelde mpg'si daha yüksek.

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")
```

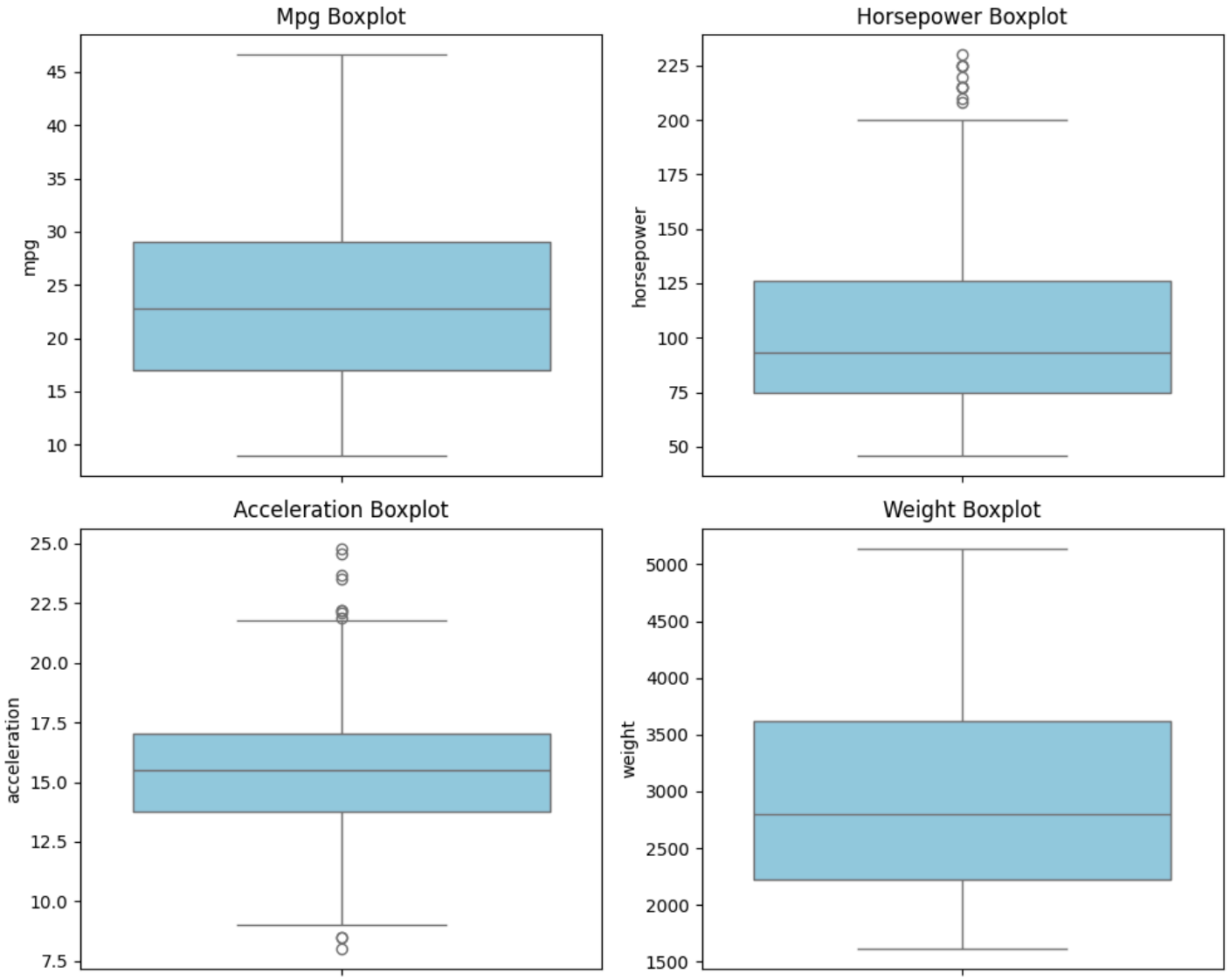
```
axs[0, 1].set_ylabel("Acceleration")

# 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))
```

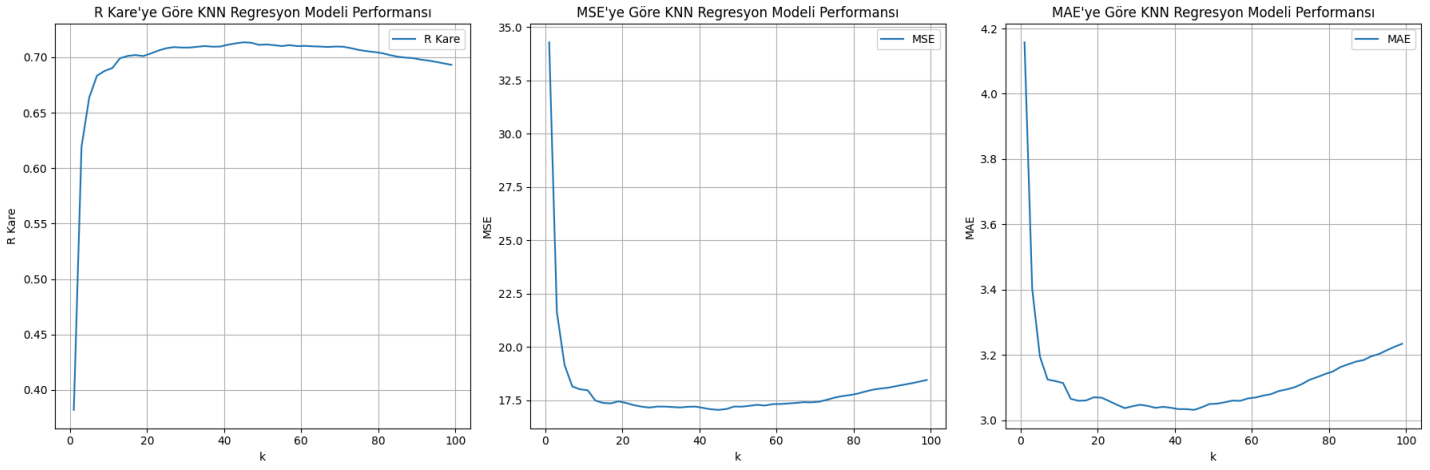
```
    mae = -1 * mae_skorları.mean()

    # 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)
```

```
En iyi R Kare: 0.713524454569566 En iyi k: 45.0
En iyi MSE: 17.045475536689707 En iyi k: 45.0
En iyi MAE: 3.0312243331812843 En iyi k: 45.0
```

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}")
```

```
En iyi k (R Kare bazlı) ile model oluşturuluyor... k=45
Horsepower=130, Acceleration=13, Weight=3500 için tahmin edilen MPG: 17.54
```

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).

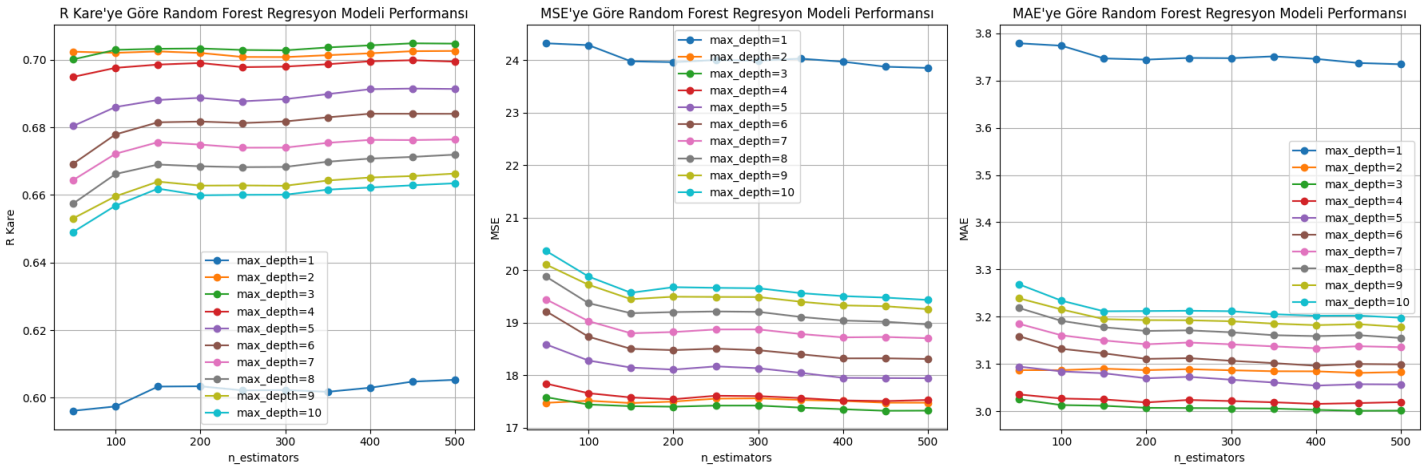
```
# Cross-validation sonuçlarını saklamak için DataFrame
cross_validation_sonuçlar = pd.DataFrame(columns=["n_estimators", "max_depth",
"r_kare", "mse", "mae"])

# 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)
```

```
En iyi R Kare: 0.7048951814316535 En iyi n_estimators: 450.0 En iyi max_depth: 3.0
En iyi MSE: 17.318880627926504 En iyi n_estimators: 450.0 En iyi max_depth: 3.0
En iyi MAE: 3.000872541587569 En iyi n_estimators: 450.0 En iyi max_depth: 3.0
```

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}")
```

```
En iyi R Kare sonucunu veren model oluşturuluyor... n_estimators=450, max_depth=3
Horsepower=130, Acceleration=13, Weight=3500 için tahmin edilen MPG: 17.06
```

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]
```

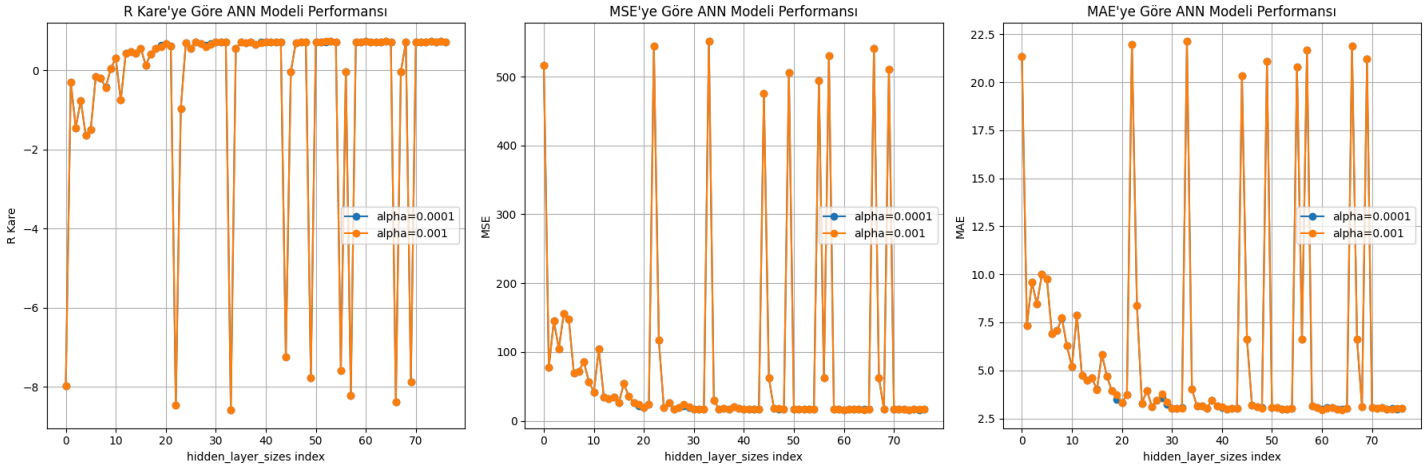
```
en_iyi_mae_hidden_layers = en_iyi_mae_satir["hidden_layer_sizes"]
en_iyi_mae_alpha = en_iyi_mae_satir["alpha"]
print("En iyi MAE:", en_iyi_mae, "En iyi hidden_layer_sizes:",
en_iyi_mae_hidden_layers, "En iyi alpha:", en_iyi_mae_alpha)
```

```
En iyi R Kare: 0.7257717131805748 En iyi hidden_layer_sizes: (7, 7, 7, 7, 7, 7) En iyi alpha: 0.001
En iyi MSE: 16.296326582594144 En iyi hidden_layer_sizes: (7, 7, 7, 7, 7, 7) En iyi alpha: 0.001
En iyi MAE: 2.9542515137886753 En iyi hidden_layer_sizes: (11, 11, 11, 11, 11, 11) En iyi alpha: 0.001
```

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}")
```

```
En iyi hiperparametrelerle ANN modeli oluşturuluyor...
Hidden Layer Sizes: (7, 7, 7, 7, 7, 7)
Alpha: 0.001
Horsepower=130, Acceleration=13, Weight=3500 için tahmin edilen MPG: 17.80
```

4- En İyi Model ve Tahminleri

4.1 KNN

```
En iyi R Kare: 0.713524454569566 En iyi k: 45.0
En iyi MSE: 17.045475536689707 En iyi k: 45.0
En iyi MAE: 3.0312243331812843 En iyi k: 45.0
```

```
En iyi k (R Kare bazlı) ile model oluşturuluyor... k=45
Horsepower=130, Acceleration=13, Weight=3500 için tahmin edilen MPG: 17.54
```

R Kare = 0.71
MSE = 17.04
MAE = 3.03
Tahmin = 17.54

4.2 Random Forest

```
En iyi R Kare: 0.7048951814316535 En iyi n_estimators: 450.0 En iyi max_depth: 3.0
En iyi MSE: 17.318880627926504 En iyi n_estimators: 450.0 En iyi max_depth: 3.0
En iyi MAE: 3.000872541587569 En iyi n_estimators: 450.0 En iyi max_depth: 3.0
```

```
En iyi R Kare sonucunu veren model oluşturuluyor... n_estimators=450, max_depth=3
Horsepower=130, Acceleration=13, Weight=3500 için tahmin edilen MPG: 17.06
```

R Kare = 0.70
MSE = 17.31
MAE = 3.00
Tahmin = 17.06

4.3 ANN

```
En iyi R Kare: 0.7257717131805748 En iyi hidden_layer_sizes: (7, 7, 7, 7, 7, 7) En iyi alpha: 0.001
En iyi MSE: 16.296326582594144 En iyi hidden_layer_sizes: (7, 7, 7, 7, 7, 7) En iyi alpha: 0.001
En iyi MAE: 2.9542515137886753 En iyi hidden_layer_sizes: (11, 11, 11, 11, 11, 11) En iyi alpha: 0.001
```

```
En iyi hiperparametrelerle ANN modeli oluşturuluyor...
Hidden Layer Sizes: (7, 7, 7, 7, 7, 7)
Alpha: 0.001
Horsepower=130, Acceleration=13, Weight=3500 için tahmin edilen MPG: 17.80
```

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:

1. R^2 (Belirleme Katsayısı):

R^2, bağımsız değişkenlerin bağımlı değişkendeki toplam varyansın ne kadarını açıkladığını gösterir. Formülü:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Burada:

- y_i: Gerçek değer
- $\hat{y}_i$: Tahmin edilen değer
- $\bar{y}$: Gerçek değerlerin ortalaması
- n: Veri noktalarının sayısı

Mean Squared Error:

2. MSE (Ortalama Kare Hatası):

MSE, tahmin edilen değerler ile gerçek değerler arasındaki farkların karelerinin ortalamasını ifade eder. Formülü:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Mean Absolute Error:

3. MAE (Ortalama Mutlak Hata):

MAE, tahmin edilen değerler ile gerçek değerler arasındaki farkların mutlak değerlerinin ortalamasını gösterir. Formülü:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

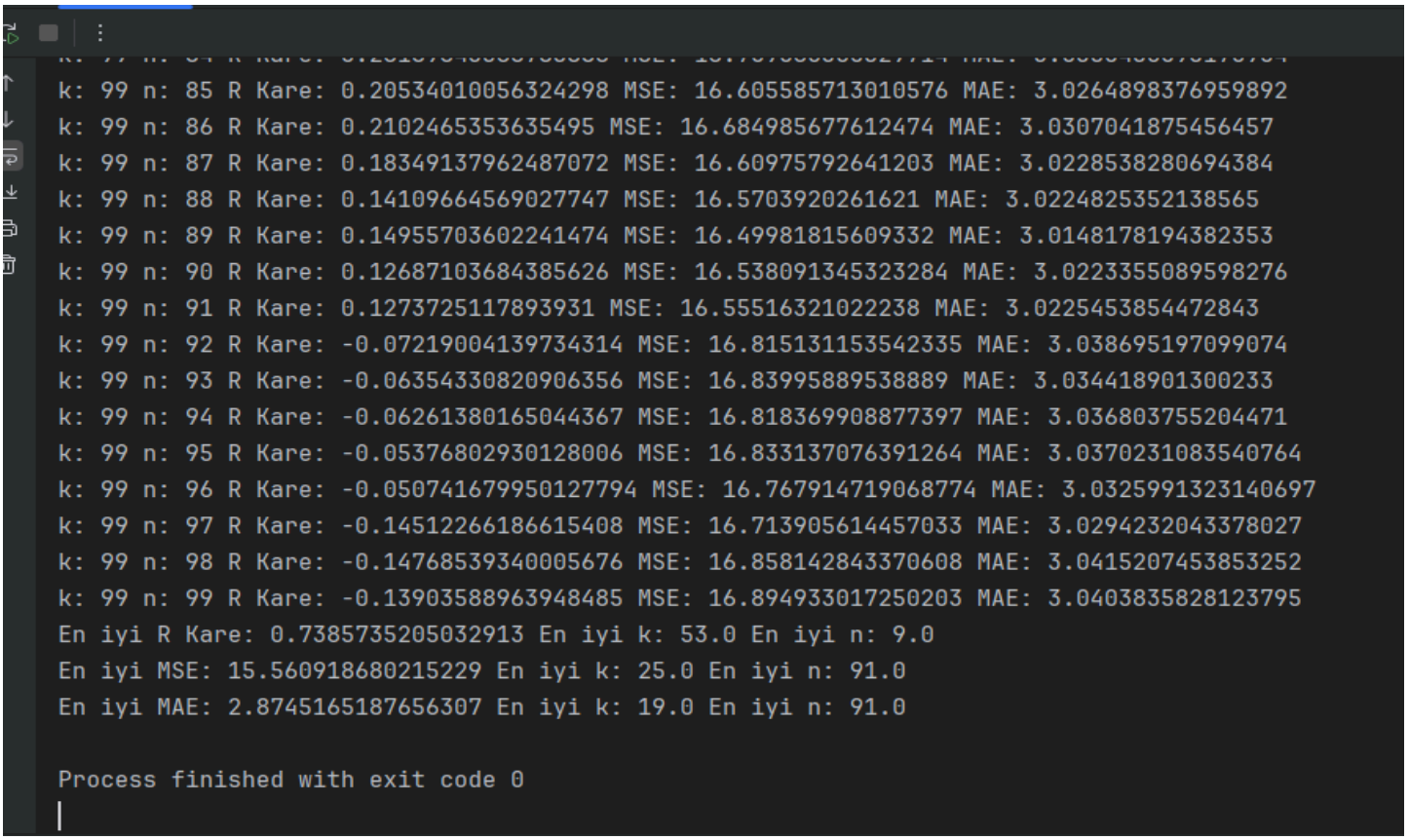
Daha sonra bu değerleri bulmak için birtakım denemelerde bulundum. İlk olarak tüm değişkenleri farklı olarak düşündüm ve ayrıştırdım. İlk değişkenimiz verileri random ayrıştırırken kullanmamız gereken random state yani seed idi. İkinci değişkenimiz tabi ki de k sayımız. Üçüncü değişkenimiz cross validation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

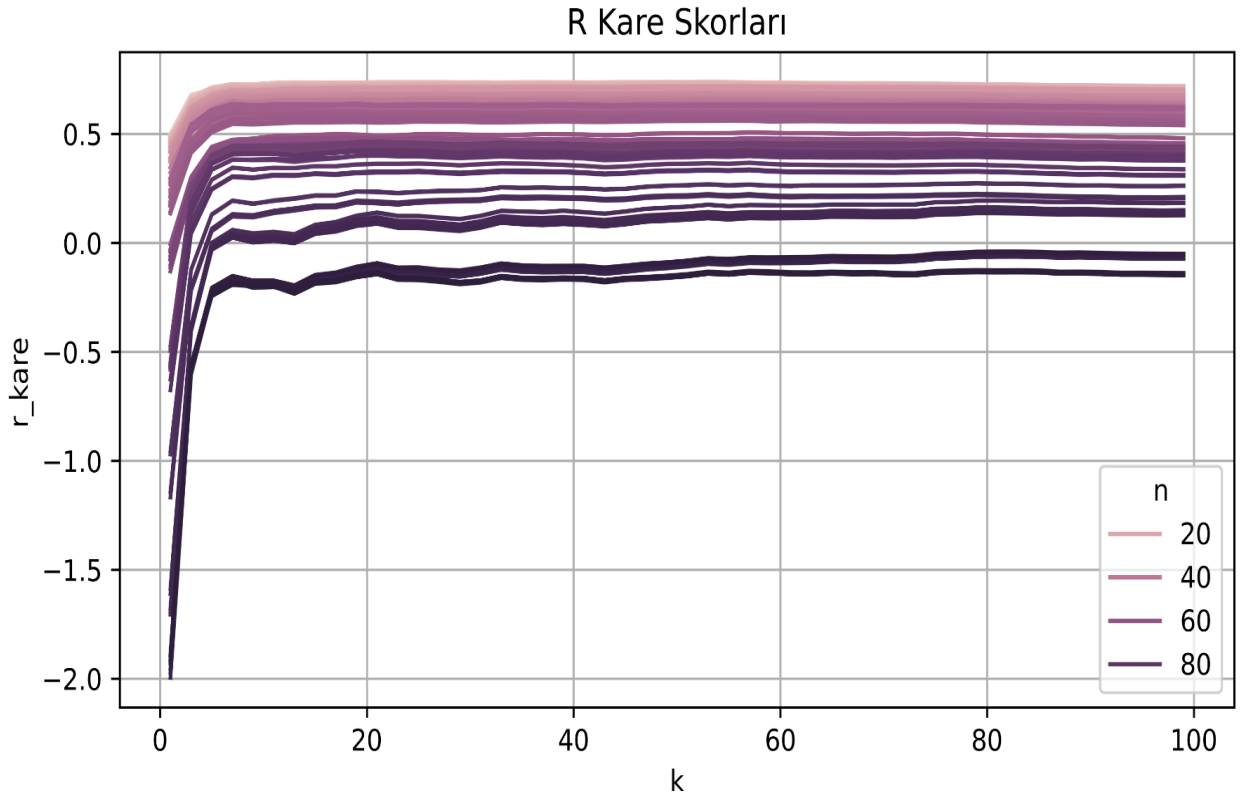
Bunun için şu kodu yazdım. K değerlerini tek sayıda seçme sebebim ise eşitlik durumunu önlemek.

```
# 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.

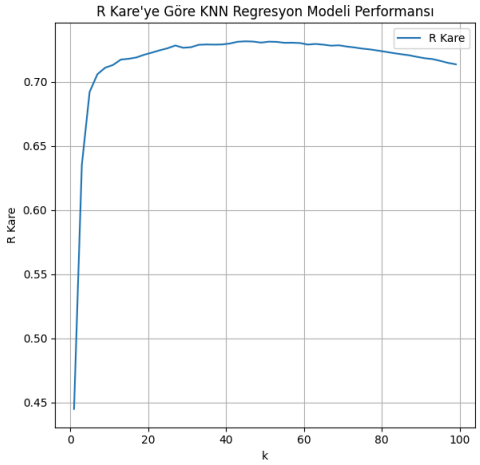


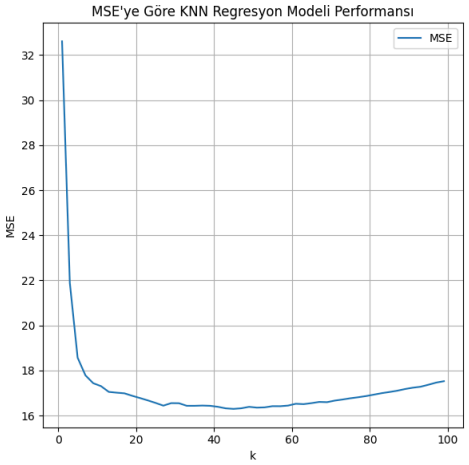


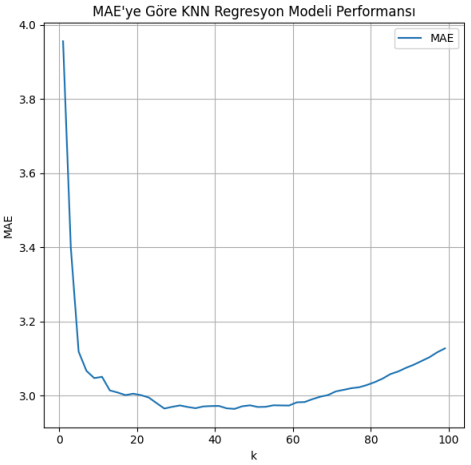
Sonuçları almam yaklaşık 30 dakika sürdü ve anlamaya çalışırken bu yolu kullanarak işin içinden çıkamayacağımı ve daha önemlisi yanlış yaptığımı fark ettim. Cross validation benim modelimi test eden bir şey ancak ben testimin parametreleri ile modelimi en iyi şekilde göstermesi için oynamaya çalışıyordum. 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:

```
k: 95 R Kare: 0.7162308628062093 MSE: 17.367510665161916 MAE: 3.103232211456997
k: 97 R Kare: 0.7146634959538586 MSE: 17.462688337401485 MAE: 3.1166165519328155
k: 99 R Kare: 0.7135613113382095 MSE: 17.526269636252728 MAE: 3.12726602821703
En iyi R Kare: 0.7314912825800323 En iyi k: 45.0
En iyi MSE: 16.29556248654352 En iyi k: 45.0
En iyi MAE: 2.9641203433906513 En iyi k: 45.0
```

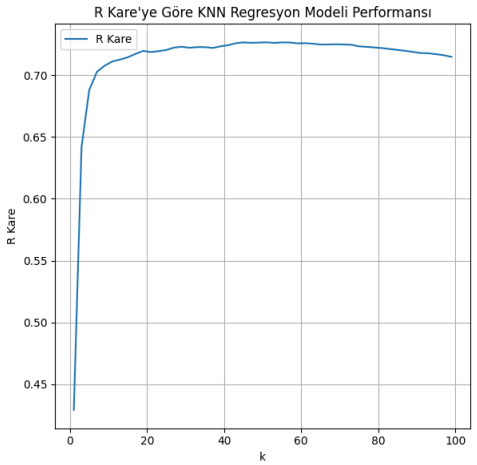


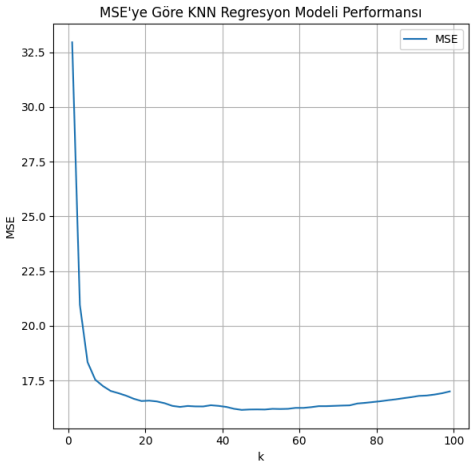


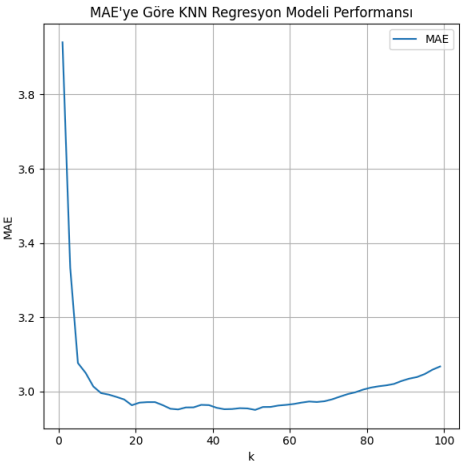


10 Fold:

```
k: 95 R Kare: 0.7169418098235095 MSE: 16.860540387042963 MAE: 3.0472599812107632
k: 97 R Kare: 0.7160972367198504 MSE: 16.92029732736152 MAE: 3.0588812311338573
k: 99 R Kare: 0.7147814552083676 MSE: 17.003585168316526 MAE: 3.0676867995772397
En iyi R Kare: 0.7265455518350834 En iyi k: 51.0
En iyi MSE: 16.159311079641746 En iyi k: 45.0
En iyi MAE: 2.9506785905326525 En iyi k: 51.0
```



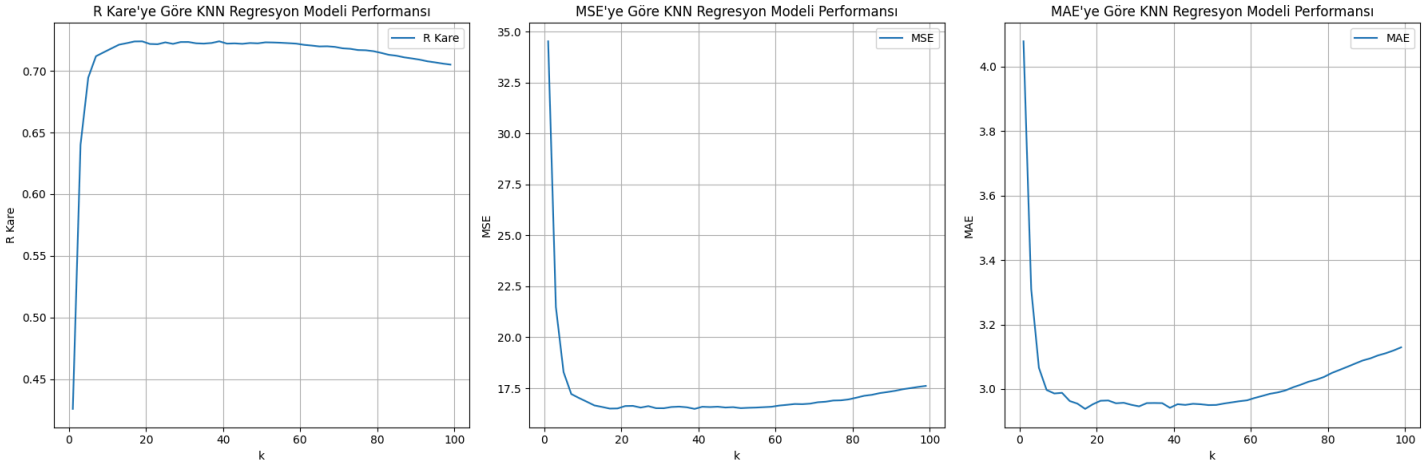




K-fold kontrol ederken katlama sayısını 5 veya 10 yapmamız sonuçlarımızı değiştirdi. 10 katlama sayısında da farklı testlerde farklı k değerleri daha uygun çıkıyor. Üstelik kullandığım seed de bunda bir etken. random-state'i 308 verdiğimde bu değerleri aldım. Değiştirdiğimde ise farklı değerler aldım.

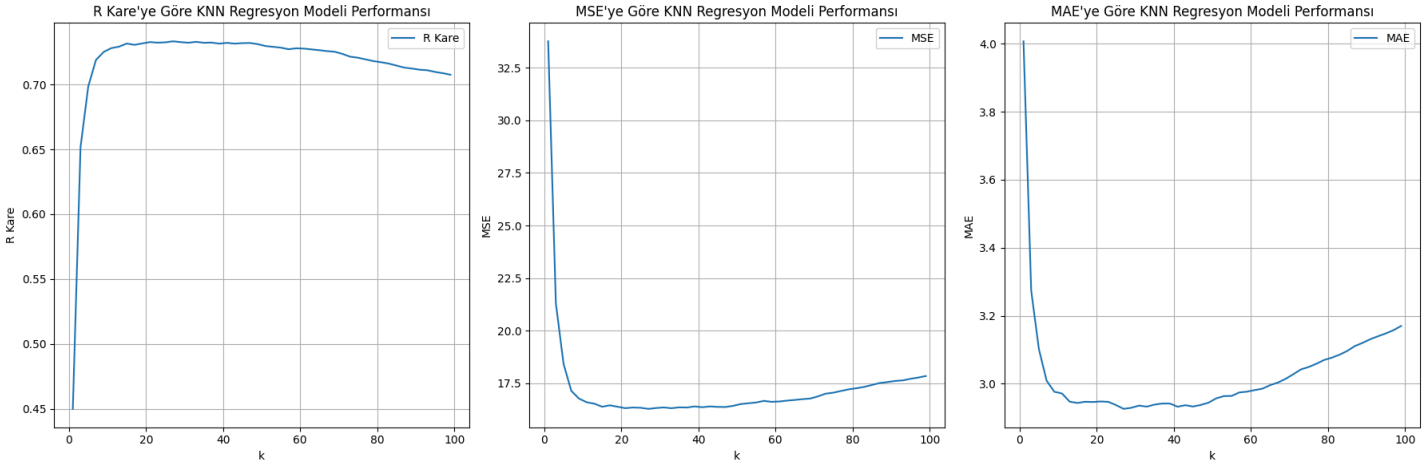
Random State 3 - Fold 10:

```
En iyi R Kare: 0.7239693263498964 En iyi k: 39.0
En iyi MSE: 16.487587353976277 En iyi k: 39.0
En iyi MAE: 2.938112844297682 En iyi k: 17.0
```



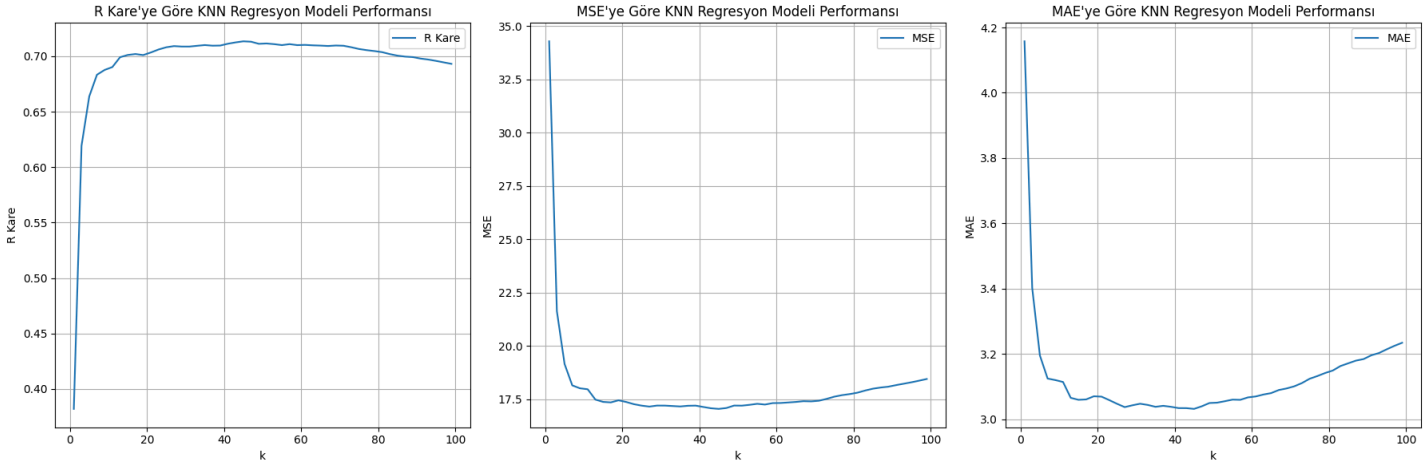
Random State 3 - Fold 5:

```
En iyi R Kare: 0.7332510910659037 En iyi k: 27.0
En iyi MSE: 16.27326024545249 En iyi k: 27.0
En iyi MAE: 2.926177750371794 En iyi k: 27.0
```



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.

```
En iyi R Kare: 0.713524454569566 En iyi k: 45.0
En iyi MSE: 17.045475536689707 En iyi k: 45.0
En iyi MAE: 3.0312243331812843 En iyi k: 45.0
```



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:

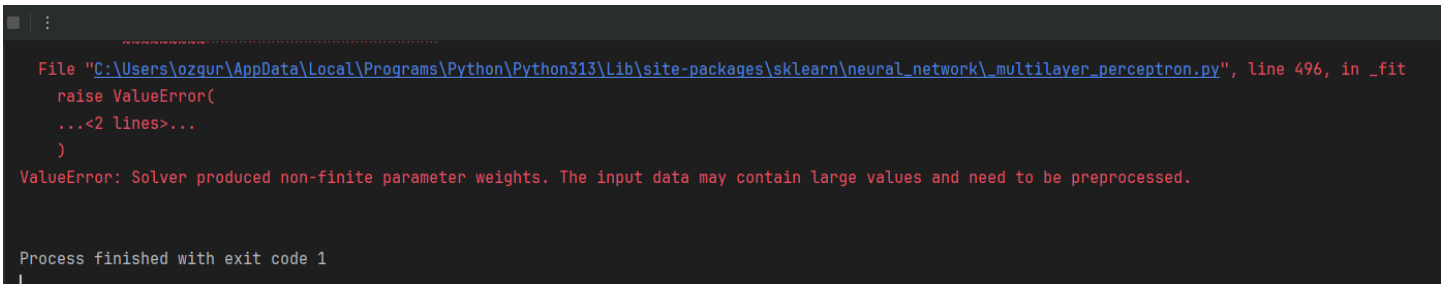
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import make_scorer, r2_score, mean_squared_error,
mean_absolute_error
from sklearn.preprocessing 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)
```

```
hidden_layer_sizes_adayları: [(10,), (20,), (30,), (40,), (50,), (60,), (70,), (80,), (90,),
 (100,), (10, 10), (20, 20), (30, 30), (40, 40), (50, 50), (60, 60), (70, 70), (80, 80), (90,
  90), (100, 100), (10, 10, 10), (20, 20, 20), (30, 30, 30), (40, 40, 40), (50, 50, 50), (60,
  60, 60), (70, 70, 70), (80, 80, 80), (90, 90, 90), (100, 100, 100), (10, 10, 10, 10), (20,
 20, 20, 20), (30, 30, 30, 30), (40, 40, 40, 40), (50, 50, 50, 50), (60, 60, 60, 60), (70,
 70, 70, 70), (80, 80, 80, 80), (90, 90, 90, 90), (100, 100, 100, 100), (10, 10, 10, 10, 10),
  (20, 20, 20, 20, 20), (30, 30, 30, 30, 30), (40, 40, 40, 40, 40), (50, 50, 50, 50, 50),
 (60, 60, 60, 60, 60), (70, 70, 70, 70, 70), (80, 80, 80, 80, 80), (90, 90, 90, 90, 90),
 (100, 100, 100, 100, 100)]
```



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.

```
reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\ozgur\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\neural_network\_base.py:172: RuntimeWarning: overflow encountered in square
  return ((y_true - y_pred) ** 2).mean() / 2
C:\Users\ozgur\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\extmath.py:203: RuntimeWarning: overflow encountered in matmul
  ret = a @ b
C:\Users\ozgur\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\utils\extmath.py:203: RuntimeWarning: invalid value encountered in matmul
  ret = a @ b
C:\Users\ozgur\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (700)
```

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

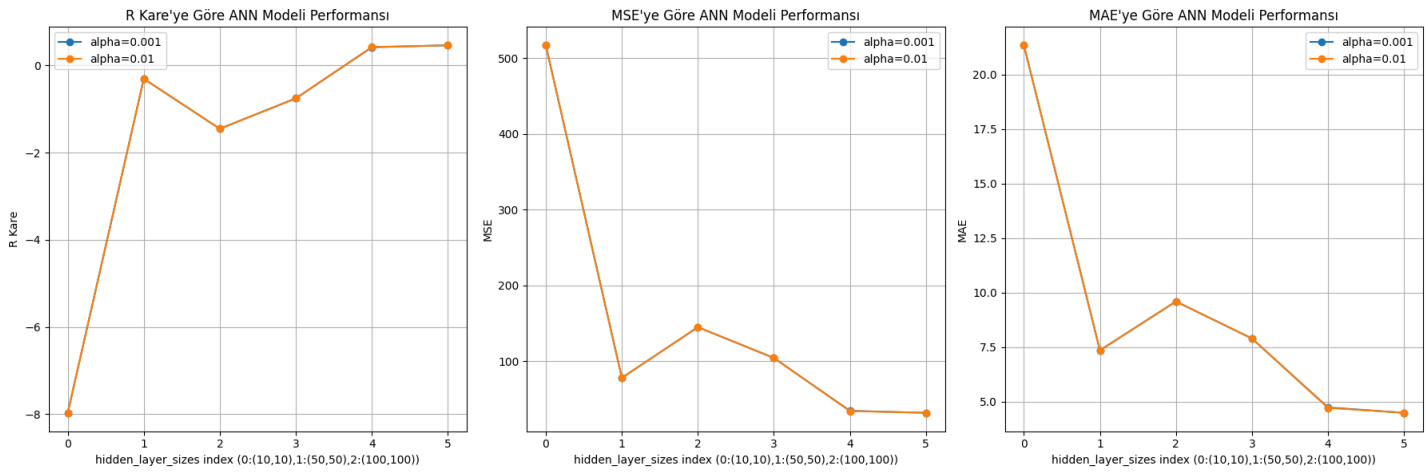
Grid Search ile grafik oluşturmak için aradaki aşamaların verisini alamadım bu yüzden for döngüsü ile devam ettim. Bir sonuç görmek için saatlerce beklememek için max iteration'u 1000'e sabitleyip geri kalan parametrelerde kütüphanenin default değerlerini kullandım.

```python
# Hiperparametre adayları
hidden_layer_sizes_adayları = []
for katman_sayisi in range(1, 3):
    for düğüm_sayisi in range(2,5):
        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 = [0.001, 0.01]
```

```
hidden_layer_sizes_adayları: [(2,), (3,),
  (4,), (2, 2), (3, 3), (4, 4)]
```

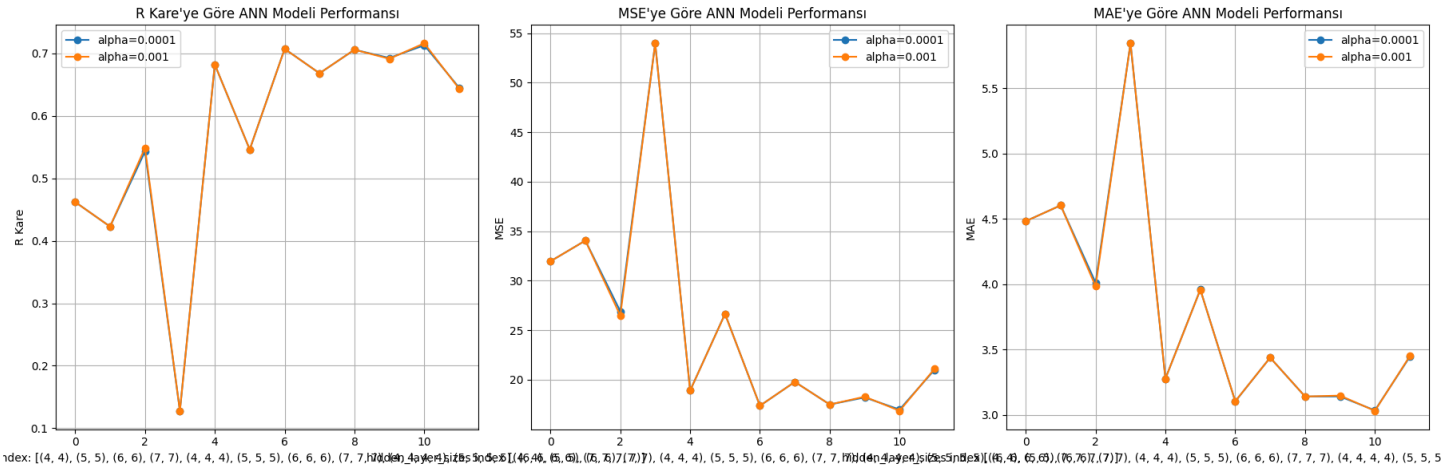
```
En iyi R Kare: 0.462103226448466 En iyi hidden_layer_sizes: (4, 4) En iyi alpha: 0.001
En iyi MSE: 31.966847284821483 En iyi hidden_layer_sizes: (4, 4) En iyi alpha: 0.001
En iyi MAE: 4.482192630627956 En iyi hidden_layer_sizes: (4, 4) En iyi alpha: 0.001
```

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.

```
# Hiperparametre adayları
hidden_layer_sizes_adayları = []
for katman_sayisi in range(2, 5):
    for düğüm_sayisi in range(4,8):
        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]
```



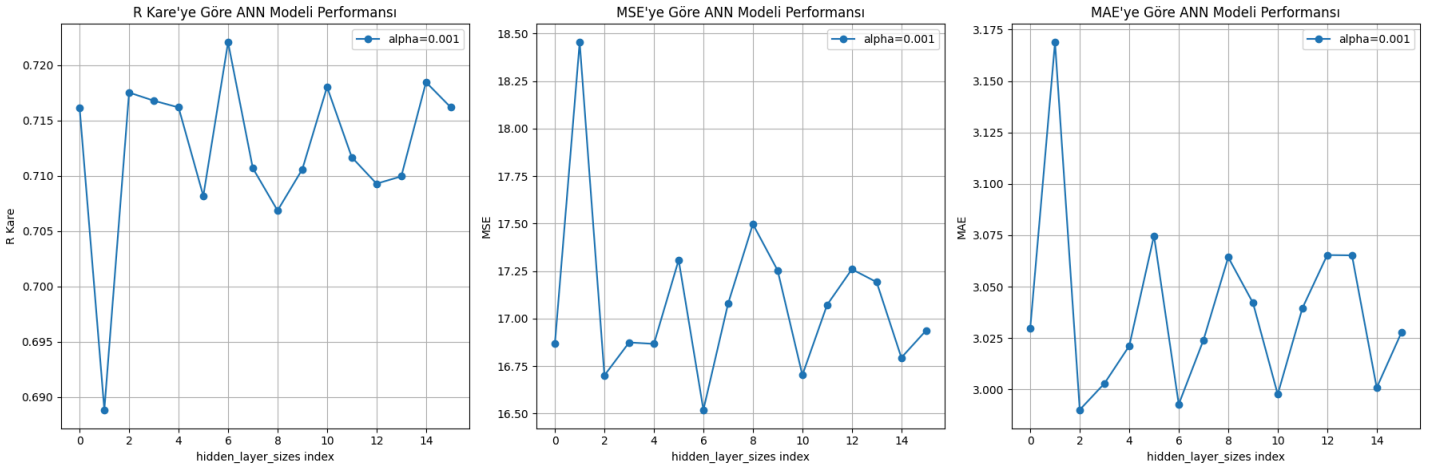

```
En iyi R Kare: 0.7161463065296754 En iyi hidden_layer_sizes: (6, 6, 6, 6) En iyi alpha: 0.001
En iyi MSE: 16.869771283499034 En iyi hidden_layer_sizes: (6, 6, 6, 6) En iyi alpha: 0.001
En iyi MAE: 3.0299287938039567 En iyi hidden_layer_sizes: (6, 6, 6, 6) En iyi alpha: 0.001
```

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.

```python
# Hiperparametre adayları
hidden_layer_sizes_adayları = []
for katman_sayisi in range(4, 8):
    for düğüm_sayisi in range(6,13, 2):
        hidden_layer = tuple([düğüm_sayisi] * katman_sayisi)
        hidden_layer_sizes_adayları.append(hidden_layer)
print("hidden_layer_sizes_adayları:", hidden_layer_sizes_adayları)
```

```
hidden_layer_sizes_adayları: [(6, 6, 6, 6), (8, 8, 8, 8),
 (10, 10, 10, 10), (12, 12, 12, 12), (6, 6, 6, 6, 6), (8,
 8, 8, 8, 8), (10, 10, 10, 10, 10), (12, 12, 12, 12, 12),
 (6, 6, 6, 6, 6, 6), (8, 8, 8, 8, 8, 8), (10, 10, 10, 10,
 10, 10), (12, 12, 12, 12, 12, 12), (6, 6, 6, 6, 6, 6, 6),
  (8, 8, 8, 8, 8, 8, 8), (10, 10, 10, 10, 10, 10, 10),
 (12, 12, 12, 12, 12, 12, 12)]
```

```
En iyi R Kare: 0.722054878038075 En iyi hidden_layer_sizes: (10, 10, 10, 10, 10) En iyi alpha: 0.001
En iyi MSE: 16.520277448913625 En iyi hidden_layer_sizes: (10, 10, 10, 10, 10) En iyi alpha: 0.001
En iyi MAE: 2.990162967207193 En iyi hidden_layer_sizes: (10, 10, 10, 10) En iyi alpha: 0.001
```



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/