In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.preprocessing import label_binarize, StandardScaler
```

1. Veri Seti Okunması

In [2]:
```python
# Veri setini yükle
df = pd.read_csv('Automobile_data.csv')

# Veri setini genel inceleme
print(df.head())
```

```
   symboling normalized-losses         make fuel-type aspiration num-of-doors  \
0          3                 ?  alfa-romero       gas        std          two
1          3                 ?  alfa-romero       gas        std          two
2          1                 ?  alfa-romero       gas        std          two
3          2               164         audi       gas        std         four
4          2               164         audi       gas        std         four

    body-style drive-wheels engine-location  wheel-base  ...  engine-size  \
0  convertible          rwd           front        88.6  ...          130
1  convertible          rwd           front        88.6  ...          130
2    hatchback          rwd           front        94.5  ...          152
3        sedan          fwd           front        99.8  ...          109
4        sedan          4wd           front        99.4  ...          136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000       21
1         mpfi  3.47    2.68               9.0        111      5000       21
2         mpfi  2.68    3.47               9.0        154      5000       19
3         mpfi  3.19     3.4              10.0        102      5500       24
4         mpfi  3.19     3.4               8.0        115      5500       18

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500
3          30  13950
4          22  17450

[5 rows x 26 columns]
```

2. Eksik Verilerin Doldurulması

In [3]:
```python
# "?" işaretlerini NaN ile değiştirdim
df.replace("?", np.nan, inplace=True)
```

```
#print("\nEksik değerlerin sayısı:\n", df.isnull().sum())

df['normalized-losses'] = df['normalized-losses'].astype(float)
df['normalized-losses'].fillna(df['normalized-losses'].median(), inplace=True)

df['bore'] = df['bore'].astype(float)
df['stroke'] = df['stroke'].astype(float)
df['horsepower'] = df['horsepower'].astype(float)
df['peak-rpm'] = df['peak-rpm'].astype(float)
df['price'] = df['price'].astype(float)

df['bore'].fillna(df['bore'].median(), inplace=True)
df['stroke'].fillna(df['stroke'].median(), inplace=True)
df['horsepower'].fillna(df['horsepower'].median(), inplace=True)
df['peak-rpm'].fillna(df['peak-rpm'].median(), inplace=True)
df['price'].fillna(df['price'].median(), inplace=True)
df['num-of-doors'].fillna(df['num-of-doors'].mode()[0], inplace=True)

#print("\nEksik değerlerin tekrar kontrolü:\n", df.isnull().sum())
```

3. Kategorik Verilerin Dönüştürülmesi

In [4]:
```
# 'num-of-doors' ve 'num-of-cylinders' sütunlarını sayısal değerlere dönüştürdüm
df['num-of-doors'] = df['num-of-doors'].replace({'two': 2, 'four': 4})
df['num-of-cylinders'] = df['num-of-cylinders'].replace({
    'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'eight': 8, 'twelve':
})

# Kategorik sütunları sayısal değerlere dönüştürmek için One-Hot Encoding kullan
categorical_columns = ['make', 'fuel-type', 'aspiration', 'body-style', 'drive-w

df = pd.get_dummies(df, columns=categorical_columns)

print(df.head())
```

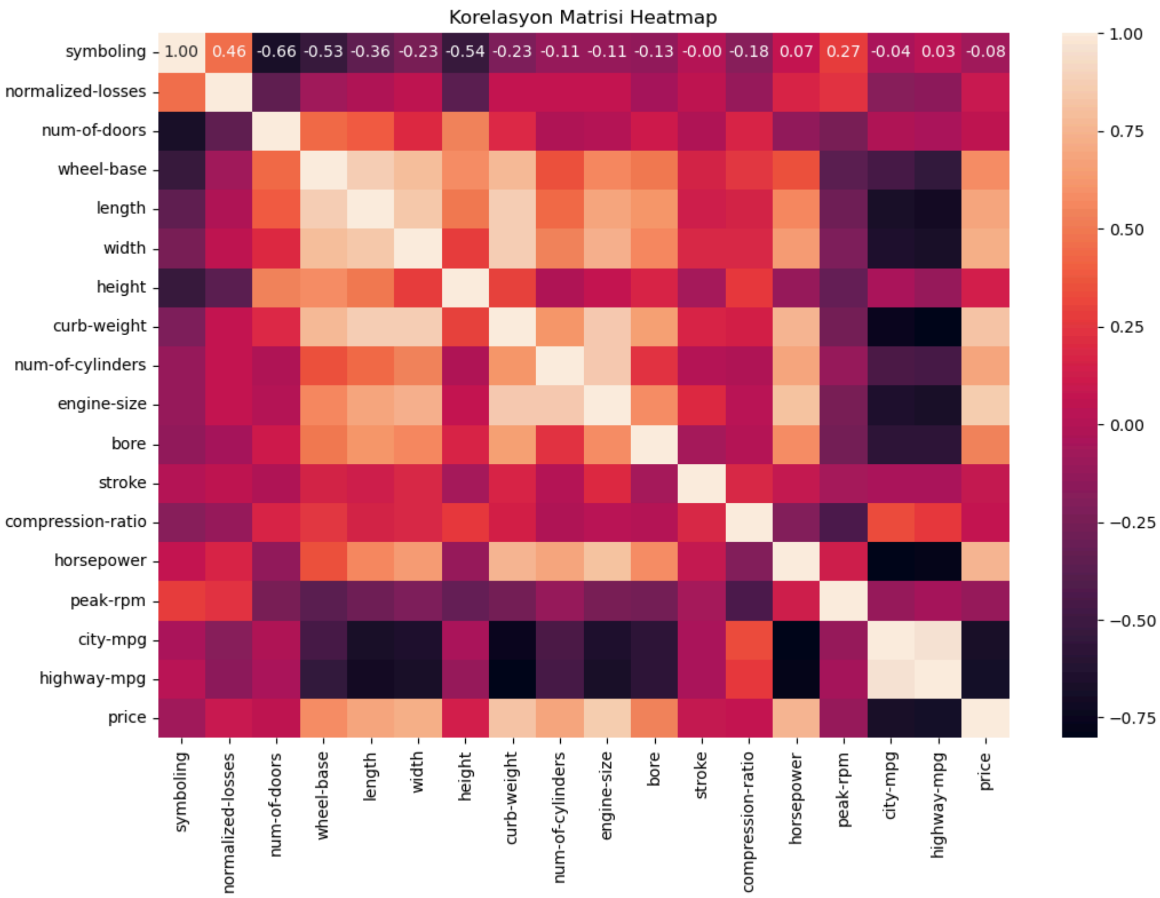
```
   symboling  normalized-losses  num-of-doors  wheel-base  length  width  \
0          3              115.0             2        88.6   168.8   64.1
1          3              115.0             2        88.6   168.8   64.1
2          1              115.0             2        94.5   171.2   65.5
3          2              164.0             4        99.8   176.6   66.2
4          2              164.0             4        99.4   176.6   66.4

   height  curb-weight  num-of-cylinders  engine-size  ...  fuel-system_mpfi  \
0    48.8         2548                 4          130  ...              True
1    48.8         2548                 4          130  ...              True
2    52.4         2823                 6          152  ...              True
3    54.3         2337                 4          109  ...              True
4    54.3         2824                 5          136  ...              True

   fuel-system_spdi  fuel-system_spfi  engine-type_dohc  engine-type_dohcv  \
0             False             False              True              False
1             False             False              True              False
2             False             False             False              False
3             False             False             False              False
4             False             False             False              False

   engine-type_l  engine-type_ohc  engine-type_ohcf  engine-type_ohcv  \
0          False            False             False             False
1          False            False             False             False
2          False            False             False              True
3          False             True             False             False
4          False             True             False             False

   engine-type_rotor
0              False
1              False
2              False
3              False
4              False

[5 rows x 69 columns]
```

3. EDA (Exploratory Data Analysis)

-Korelasyon Matrisi Heatmap

In [5]:
```python
# Sayısal verilerin özet istatistikleri
print("\nSayısal verilerin özet istatistikleri:\n", df.describe())

# Korelasyon Analizi ve Heatmap
numeric_df = df.select_dtypes(include=[np.number])  # Sadece sayısal sütunlar
numeric_df = numeric_df.replace([np.inf, -np.inf], np.nan).dropna()  # Sonsuz de

corr_matrix = numeric_df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f")
plt.title("Korelasyon Matrisi Heatmap")
plt.show()
```

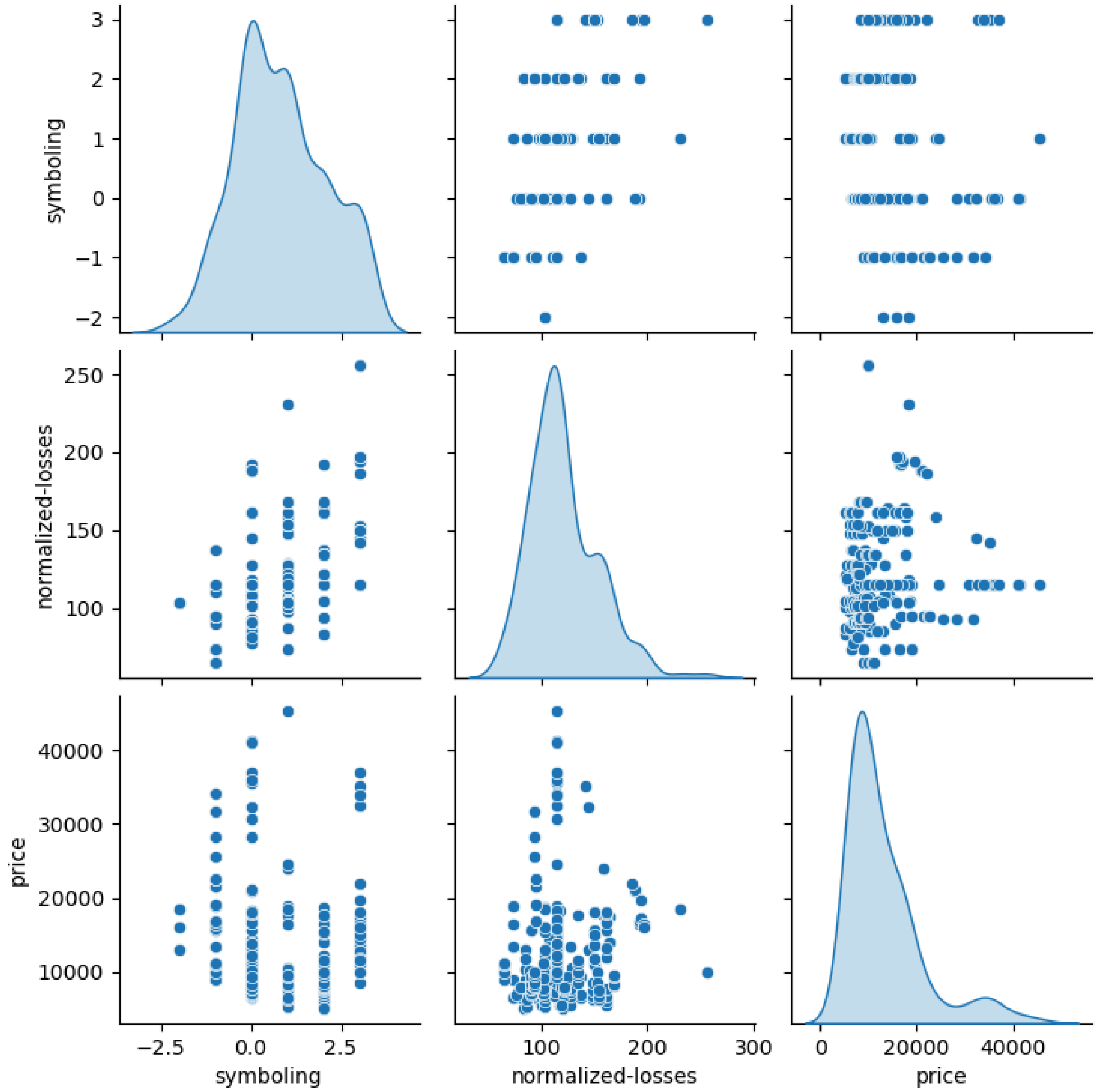
```
Sayısal verilerin özet istatistikleri:
        symboling  normalized-losses  num-of-doors  wheel-base      length  \
count  205.000000         205.000000    205.000000  205.000000  205.000000   
mean     0.834146         120.600000      3.131707   98.756585  174.049268   
std      1.245307          31.805105      0.993715    6.021776   12.337289   
min     -2.000000          65.000000      2.000000   86.600000  141.100000   
25%      0.000000         101.000000      2.000000   94.500000  166.300000   
50%      1.000000         115.000000      4.000000   97.000000  173.200000   
75%      2.000000         137.000000      4.000000  102.400000  183.100000   
max      3.000000         256.000000      4.000000  120.900000  208.100000   

            width      height  curb-weight  num-of-cylinders  engine-size  \
count  205.000000  205.000000   205.000000        205.000000   205.000000   
mean    65.907805   53.724878  2555.565854          4.380488   126.907317   
std      2.145204    2.443522   520.680204          1.080854    41.642693   
min     60.300000   47.800000  1488.000000          2.000000    61.000000   
25%     64.100000   52.000000  2145.000000          4.000000    97.000000   
50%     65.500000   54.100000  2414.000000          4.000000   120.000000   
75%     66.900000   55.500000  2935.000000          4.000000   141.000000   
max     72.300000   59.800000  4066.000000         12.000000   326.000000   

             bore      stroke  compression-ratio  horsepower     peak-rpm  \
count  205.000000  205.000000         205.000000  205.000000   205.000000   
mean     3.329366    3.256098          10.142537  104.165854  5126.097561   
std      0.270858    0.313634           3.972040   39.529733   477.035772   
min      2.540000    2.070000           7.000000   48.000000  4150.000000   
25%      3.150000    3.110000           8.600000   70.000000  4800.000000   
50%      3.310000    3.290000           9.000000   95.000000  5200.000000   
75%      3.580000    3.410000           9.400000  116.000000  5500.000000   
max      3.940000    4.170000          23.000000  288.000000  6600.000000   

         city-mpg  highway-mpg         price  
count  205.000000   205.000000    205.000000  
mean    25.219512    30.751220  13150.307317  
std      6.542142     6.886443   7879.121326  
min     13.000000    16.000000   5118.000000  
25%     19.000000    25.000000   7788.000000  
50%     24.000000    30.000000  10295.000000  
75%     30.000000    34.000000  16500.000000  
max     49.000000    54.000000  45400.000000  
```



- Pair Plot

In [6]:
```
selected_columns = ['symboling', 'normalized-losses', 'price']
sns.pairplot(df[selected_columns], diag_kind="kde")
plt.show()
```

```
E:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_a
s_na option is deprecated and will be removed in a future version. Convert inf va
lues to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
E:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_a
s_na option is deprecated and will be removed in a future version. Convert inf va
lues to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
E:\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_a
s_na option is deprecated and will be removed in a future version. Convert inf va
lues to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



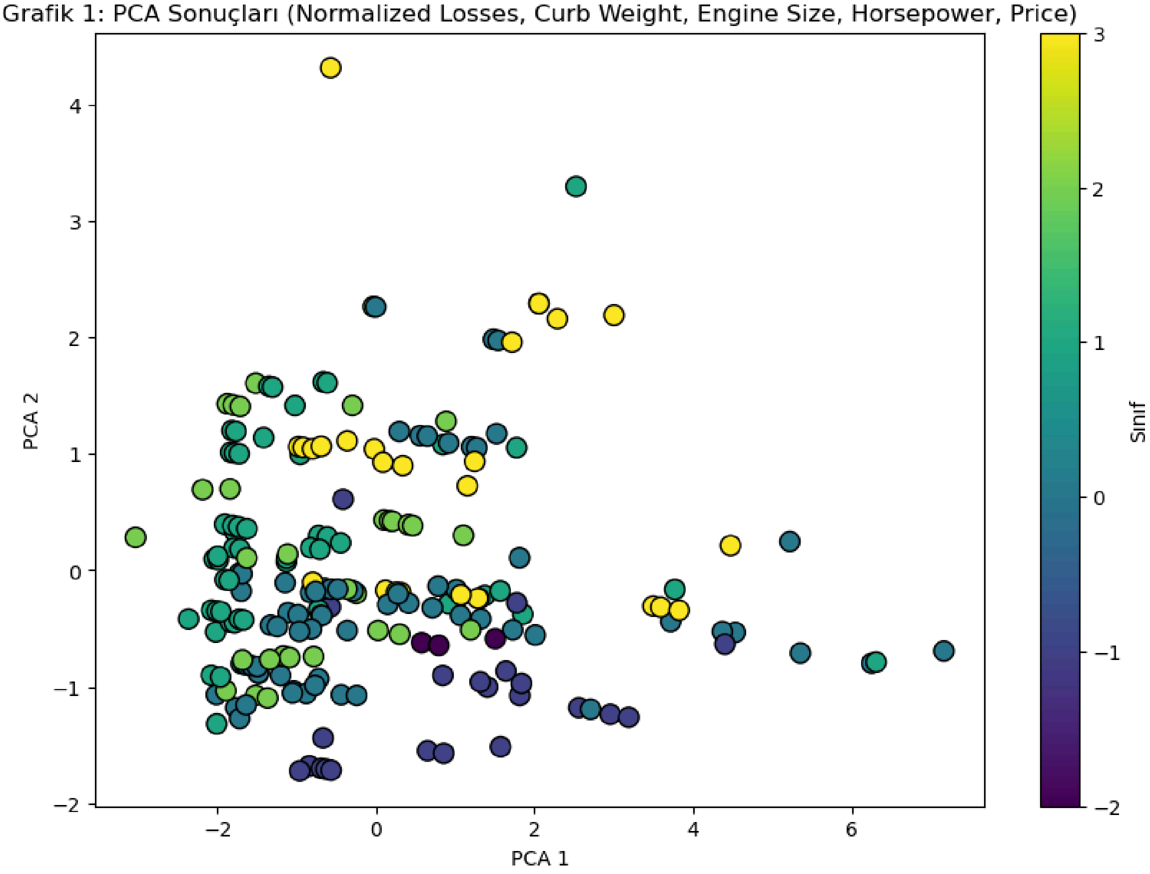
4. PCA (Principal Component Analysis)

In [7]:

```
# PCA fonksiyonu
def plot_pca(X, y, title):
    # Veriyi ölçeklendirme
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # PCA ile iki bileşene indirgeme
    pca = PCA(n_components=2)
    pca_result = pca.fit_transform(X_scaled)

    # PCA bileşenlerinin açıklanan varyans oranlarını yazdırma
    print(f"\n{title} - PCA Bileşenleri:\n", pca.explained_variance_ratio_)

    # Grafik oluşturma
    plt.figure(figsize=(10, 7))
    scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1], c=y, cmap='viridis
    plt.xlabel('PCA 1')
    plt.ylabel('PCA 2')
    plt.title(title)
```

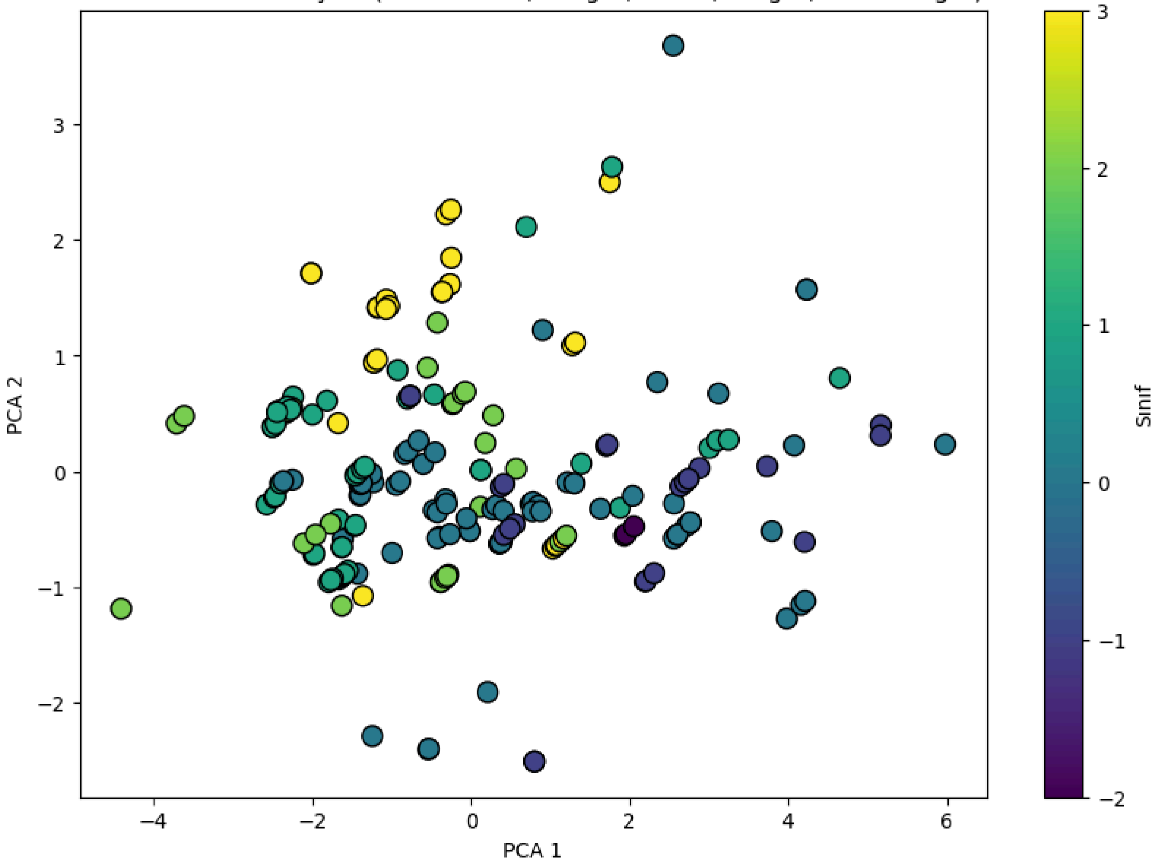
```
    # Renk barı ekleme
    plt.colorbar(scatter, label='Sınıf')

    # Grafiği göster
    plt.show()

y = numeric_df['symboling']

# Grafik 1
X1 = numeric_df[['normalized-losses', 'curb-weight', 'engine-size', 'horsepower'
plot_pca(X1, y, 'Grafik 1: PCA Sonuçları (Normalized Losses, Curb Weight, Engine

# Grafik 2
X2 = numeric_df[['wheel-base', 'length', 'width', 'height', 'curb-weight']]
plot_pca(X2, y, 'Grafik 2: PCA Sonuçları (Wheel Base, Length, Width, Height, Cur

# Grafik 3
X3 = numeric_df[['compression-ratio', 'horsepower', 'city-mpg', 'highway-mpg', '
plot_pca(X3, y, 'Grafik 3: PCA Sonuçları (Compression Ratio, Horsepower, City MP
```

```
Grafik 1: PCA Sonuçları (Normalized Losses, Curb Weight, Engine Size, Horsepower,
Price) - PCA Bileşenleri:
 [0.68774484 0.19871276]
```



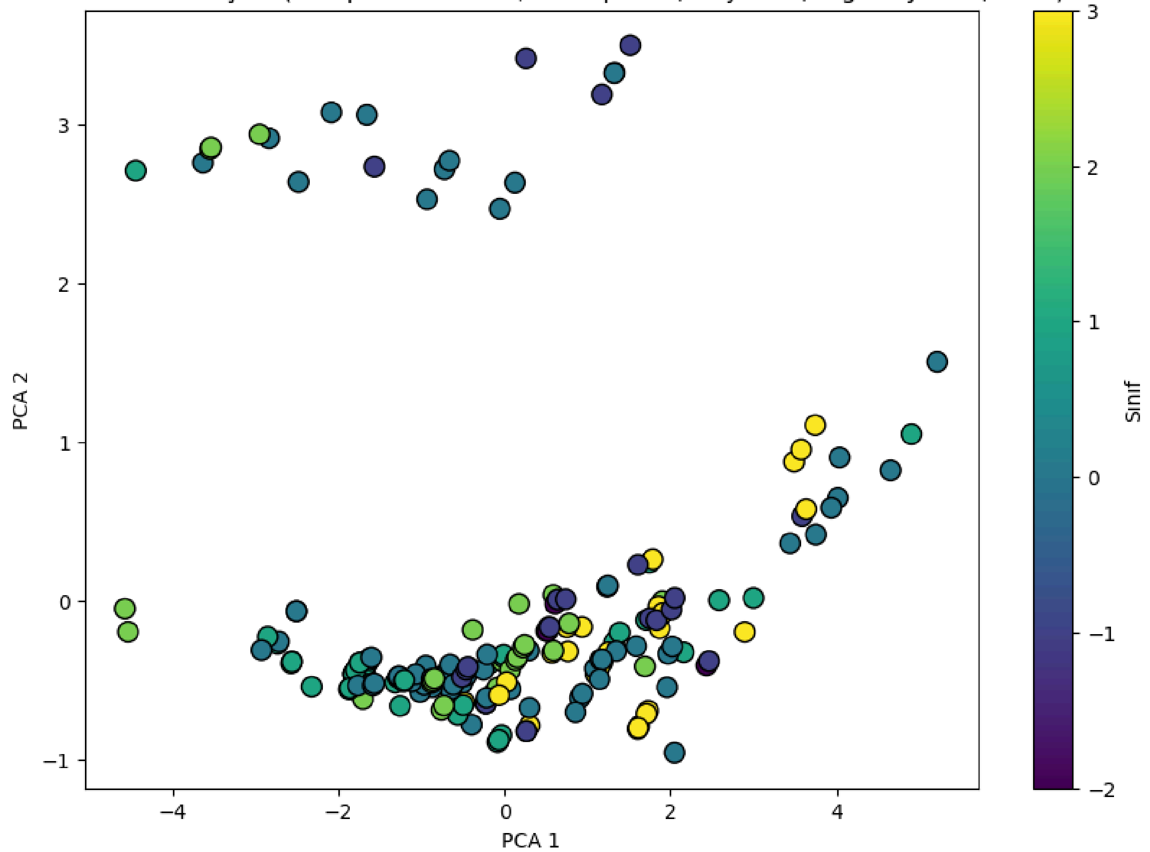

```
Grafik 2: PCA Sonuçları (Wheel Base, Length, Width, Height, Curb Weight) - PCA Bi
leşenleri:
 [0.7531455  0.17161811]
```

Grafik 2: PCA Sonuçları (Wheel Base, Length, Width, Height, Curb Weight)




```
Grafik 3: PCA Sonuçları (Compression Ratio, Horsepower, City MPG, Highway MPG, Pr
ice) - PCA Bileşenleri:
 [0.67867298 0.21272136]
```

Grafik 3: PCA Sonuçları (Compression Ratio, Horsepower, City MPG, Highway MPG, Price)



5. Öznitelik Seçimi

```
In [8]: # Korelasyon matrisine göre yüksek korelasyonlu öznitelikleri belirleme
threshold = 0.75
high_corr_features = [column for column in corr_matrix.columns if any(corr_matri
print("\nYüksek korelasyonlu öznitelikler:\n", high_corr_features)
```

```
Yüksek korelasyonlu öznitelikler:
 ['symboling', 'normalized-losses', 'num-of-doors', 'wheel-base', 'length', 'widt
h', 'height', 'curb-weight', 'num-of-cylinders', 'engine-size', 'bore', 'stroke',
'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'pric
e']
```

6. Model Eğitimleri ve Değerlendirmeler

```
In [9]: # Bağımsız ve bağımlı değişkenler
X = df.drop(['symboling'], axis=1)
X_high_corr = df[high_corr_features]
y = df['symboling']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
X_train_h, X_test_h, y_train_h, y_test_h = train_test_split(X_high_corr, y, test

print("\nEğitim ve test seti boyutları (yüksek korelasyonlu öznitelikler ile):",
print("\nEğitim ve test seti boyutları (tüm öznitelikler ile):", X_train.shape,
```

```
Eğitim ve test seti boyutları (yüksek korelasyonlu öznitelikler ile): (164, 18)
(41, 18) (164,) (41,)

Eğitim ve test seti boyutları (tüm öznitelikler ile): (164, 68) (41, 68) (164,)
(41,)
```

- Karar Ağacı

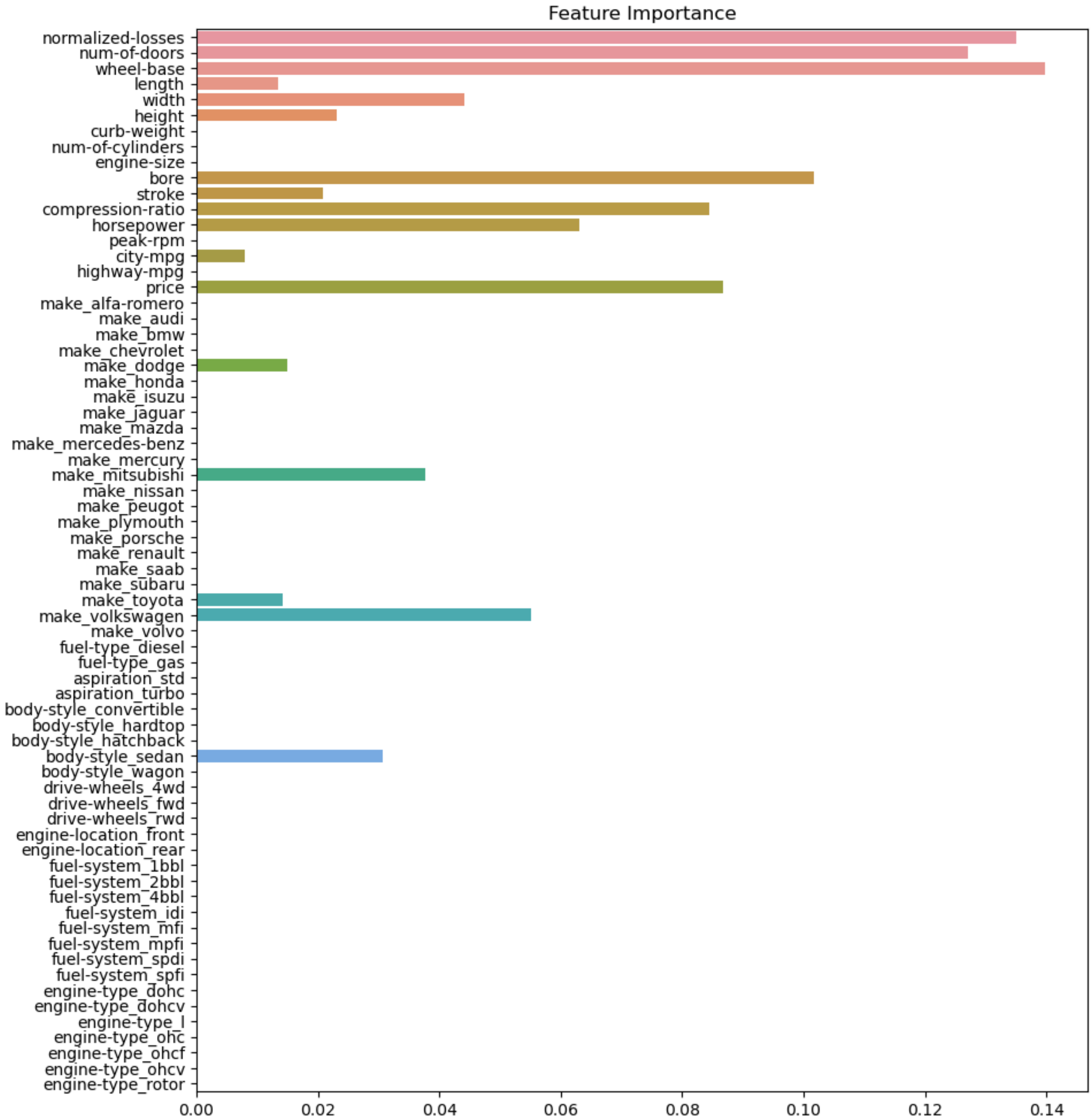
```
In [10]: # Modeli eğitme
dt_model = DecisionTreeClassifier()
dt_model_h = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_model_h.fit(X_train_h, y_train_h)

random_index = np.random.randint(0, X_test.shape[0])
random_test_input = X_test.iloc[random_index].values.reshape(1, -1)

random_prediction = dt_model.predict(random_test_input)

print(f"Rastgele Seçilen Girdi (Index: {random_index}):")
print(X_test.iloc[random_index])
print(f"Gerçek Değer: {y_test.iloc[random_index]}")
print(f"Tahmin Edilen Değer: {random_prediction[0]}")

# Tahmin ve değerlendirme (tüm öznitelikler ile)
y_pred = dt_model.predict(X_test)
train_pred_dt = dt_model.predict(X_train)
print("Karar Ağaçları Modeli Performansı (tüm öznitelikler ile)")
print(classification_report(y_test, y_pred))


feature_importance = dt_model.feature_importances_
```

```
features = X_train.columns

# Görselleştirme
plt.figure(figsize=(10, 12))
sns.barplot(x=feature_importance, y=features)
plt.title('Feature Importance')
plt.show()

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# ROC eğrisi ve AUC (tüm öznitelikler ile)
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
n_classes = y_test_binarized.shape[1]
y_score = dt_model.predict_proba(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Tüm sınıflar için ROC eğrisini çizme
plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Sınıf {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Karar Ağaçları ROC Eğrisi (tüm öznitelikler ile)')
plt.legend(loc='lower right')
plt.show()

# Tahmin ve değerlendirme (yüksek korelasyonlu öznitelikler ile)
y_pred_h = dt_model_h.predict(X_test_h)
print("Karar Ağaçları Modeli Performansı (yüksek korelasyonlu öznitelikler ile)"
print(classification_report(y_test_h, y_pred_h))

print("Confusion Matrix:")
print(confusion_matrix(y_test_h, y_pred_h))

# ROC eğrisi ve AUC (yüksek korelasyonlu öznitelikler ile)
y_test_h_binarized = label_binarize(y_test_h, classes=np.unique(y_test_h))
n_classes_h = y_test_h_binarized.shape[1]
y_score_h = dt_model_h.predict_proba(X_test_h)

fpr_h = dict()
tpr_h = dict()
roc_auc_h = dict()

for i in range(n_classes_h):
    fpr_h[i], tpr_h[i], _ = roc_curve(y_test_h_binarized[:, i], y_score_h[:, i])
    roc_auc_h[i] = auc(fpr_h[i], tpr_h[i])

# Tüm sınıflar için ROC eğrisini çizme (yüksek korelasyonlu öznitelikler)
```

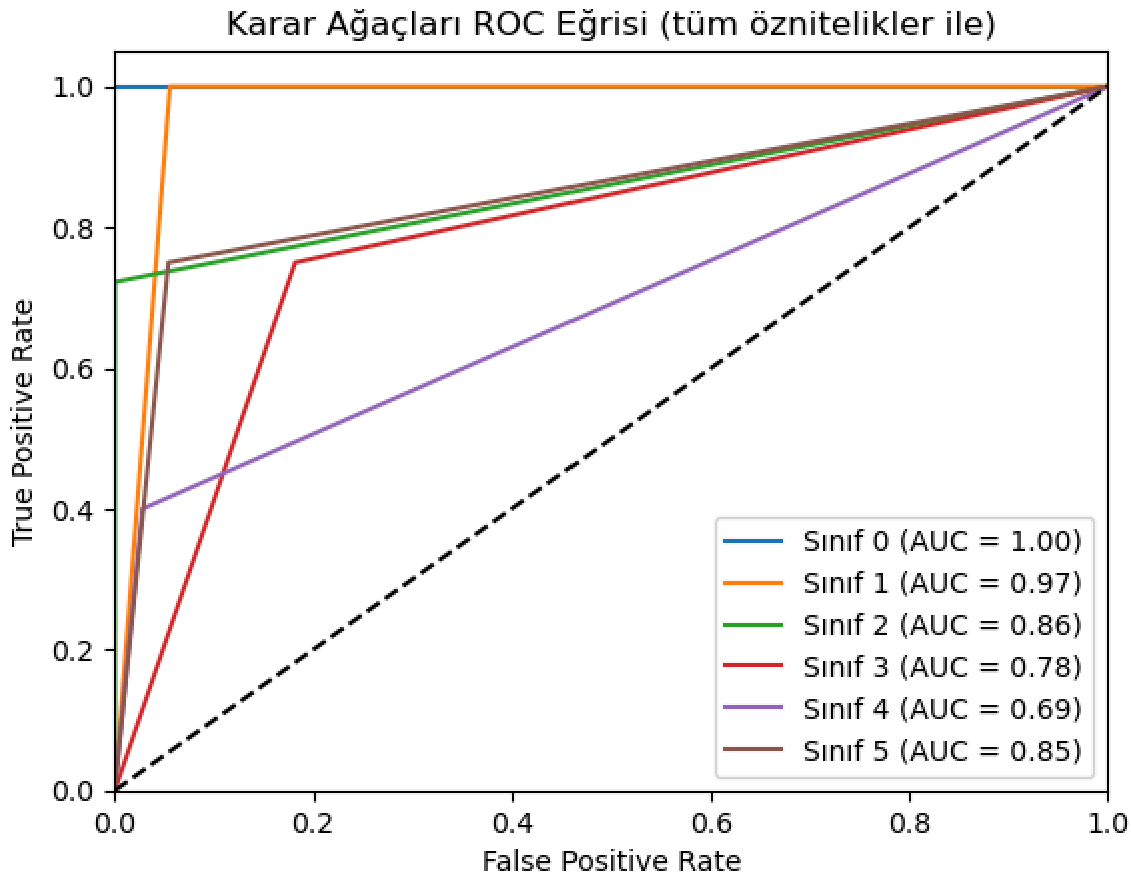
```python
plt.figure()
for i in range(n_classes_h):
    plt.plot(fpr_h[i], tpr_h[i], label=f'Sınıf {i} (AUC = {roc_auc_h[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Karar Ağaçları ROC Eğrisi (yüksek korelasyonlu öznitelikler ile)')
plt.legend(loc='lower right')
plt.show()
```

```
Rastgele Seçilen Girdi (Index: 13):
normalized-losses     65.0
num-of-doors             4
wheel-base           102.4
length               175.6
width                 66.5
                      ...
engine-type_l        False
engine-type_ohc       True
engine-type_ohcf     False
engine-type_ohcv     False
engine-type_rotor    False
Name: 175, Length: 68, dtype: object
Gerçek Değer: -1
Tahmin Edilen Değer: -1
Karar Ağaçları Modeli Performansı (tüm öznitelikler ile)
              precision    recall  f1-score   support

          -2       1.00      1.00      1.00         1
          -1       0.71      1.00      0.83         5
           0       1.00      0.72      0.84        18
           1       0.50      0.75      0.60         8
           2       0.67      0.40      0.50         5
           3       0.60      0.75      0.67         4

    accuracy                           0.73        41
   macro avg       0.75      0.77      0.74        41
weighted avg       0.79      0.73      0.74        41
```

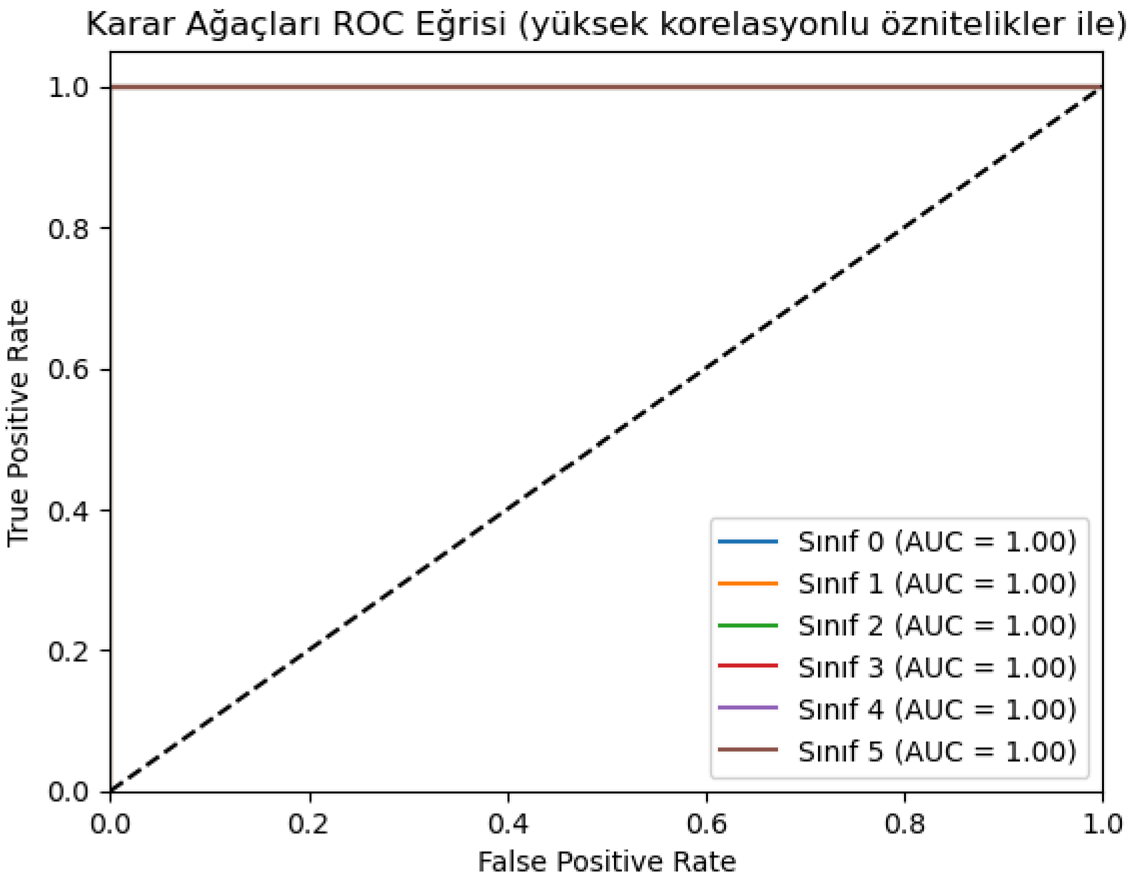
```
E:\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```




```
Confusion Matrix:
[[ 1  0  0  0  0  0]
 [ 0  5  0  0  0  0]
 [ 0  1 13  2  1  1]
 [ 0  1  0  6  0  1]
 [ 0  0  0  3  2  0]
 [ 0  0  0  1  0  3]]
```




```
Karar Ağaçları Modeli Performansı (yüksek korelasyonlu öznitelikler ile)
              precision    recall  f1-score   support

          -2       1.00      1.00      1.00         1
          -1       1.00      1.00      1.00         5
           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00         4

    accuracy                           1.00        41
   macro avg       1.00      1.00      1.00        41
weighted avg       1.00      1.00      1.00        41

Confusion Matrix:
[[ 1  0  0  0  0  0]
 [ 0  5  0  0  0  0]
 [ 0  0 18  0  0  0]
 [ 0  0  0  8  0  0]
 [ 0  0  0  0  5  0]
 [ 0  0  0  0  0  4]]
```



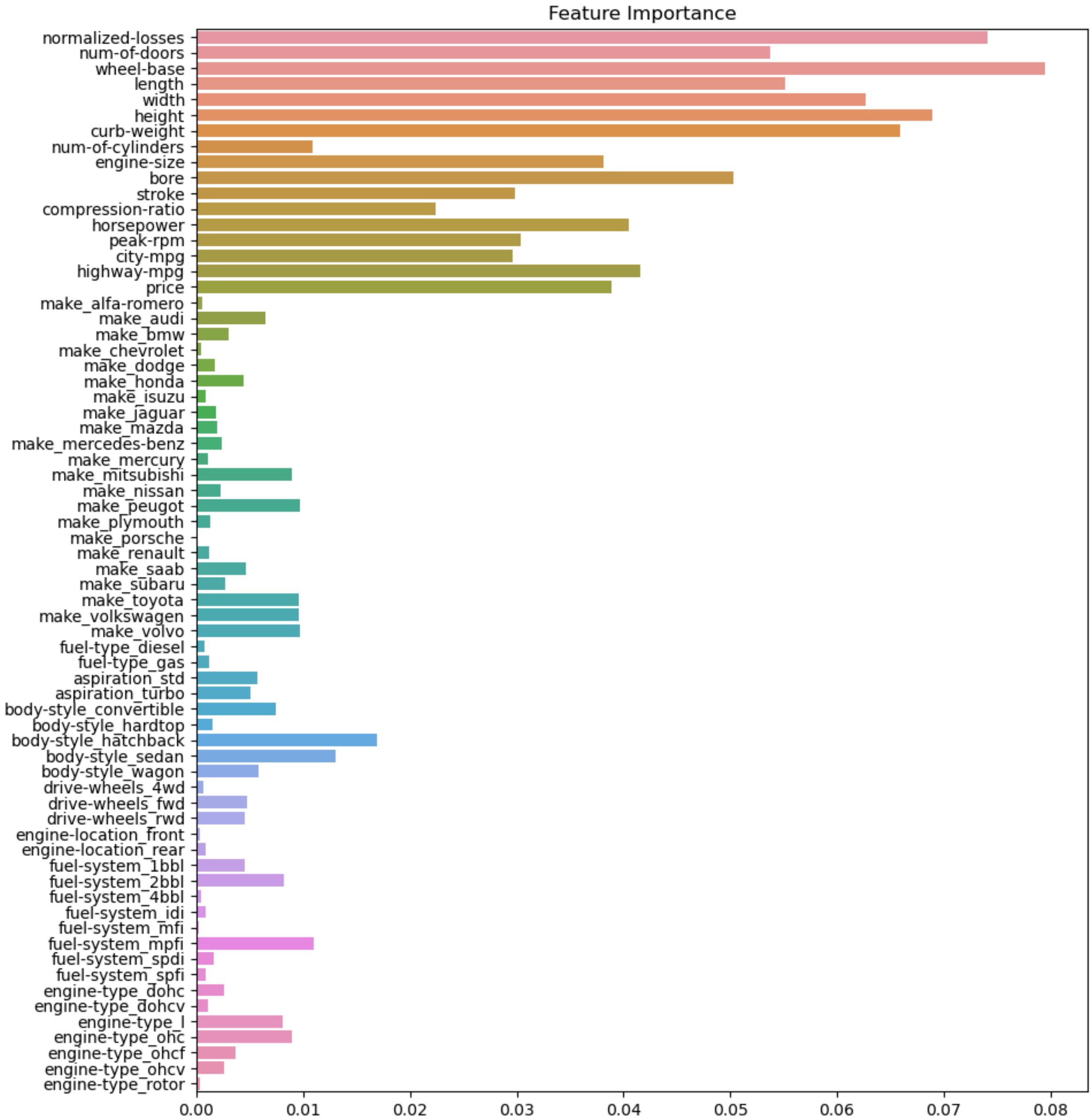
- Rastgele Orman

In [11]:
```
# Modeli eğitme
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

rf_model_h = RandomForestClassifier()
rf_model_h.fit(X_train_h, y_train_h)

random_prediction = rf_model.predict(random_test_input)

print(f"Rastgele Seçilen Girdi (Index: {random_index}):")
print(X_test.iloc[random_index])
print(f"Gerçek Değer: {y_test.iloc[random_index]}")
print(f"Tahmin Edilen Değer: {random_prediction[0]}")


# Tahmin ve değerlendirme (tüm öznitelikler ile)
y_pred = rf_model.predict(X_test)
print("\nRastgele Ormanlar Modeli Performansı (tüm öznitelikler ile)")
print(classification_report(y_test, y_pred))

feature_importance = rf_model.feature_importances_
features = X_train.columns

# Görselleştirme
plt.figure(figsize=(10, 12))
sns.barplot(x=feature_importance, y=features)
plt.title('Feature Importance')
plt.show()
```

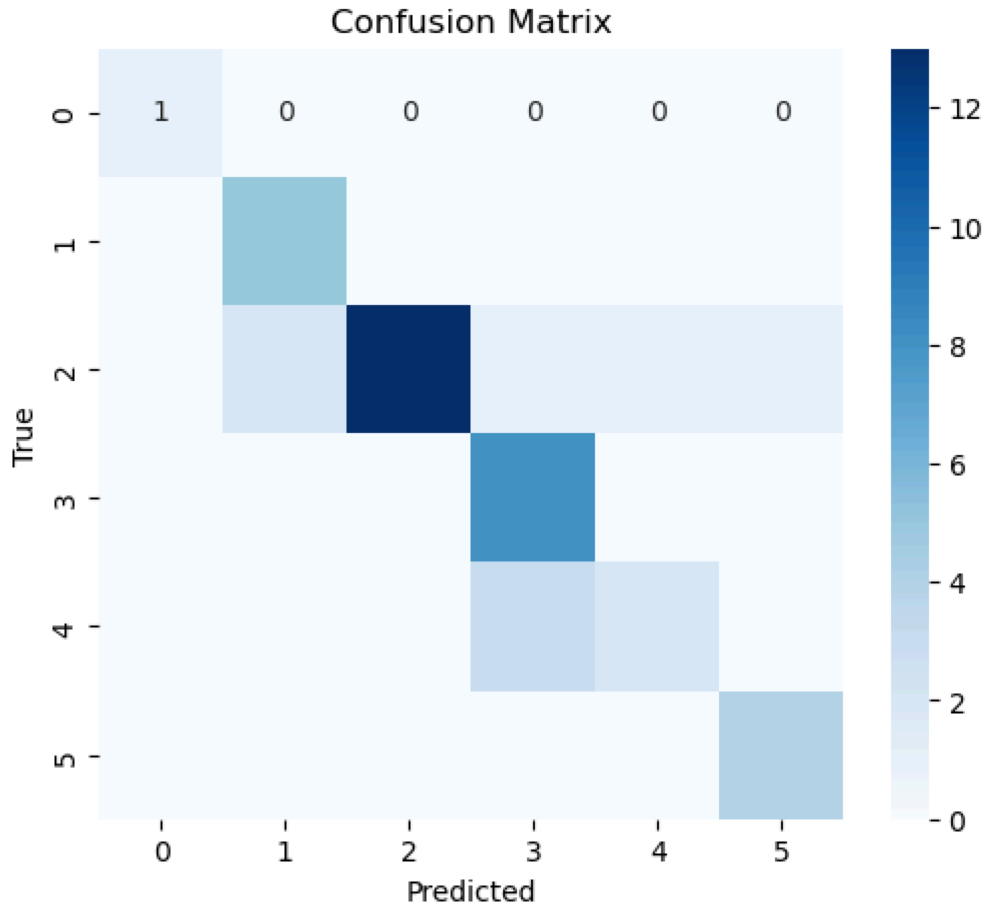
```python
print("Confusion Matrix:")
conf_matrix_rf = confusion_matrix(y_test, y_pred)
print(conf_matrix_rf)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# ROC eğrisi ve AUC (tüm öznitelikler ile)
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
n_classes = y_test_binarized.shape[1]
y_score = rf_model.predict_proba(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Tüm sınıflar için ROC eğrisini çizme
plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Sınıf {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Eğrisi (tüm öznitelikler ile)')
plt.legend(loc='lower right')
plt.show()

# Tahmin ve değerlendirme (yüksek korelasyonlu öznitelikler ile)
y_pred_h = rf_model_h.predict(X_test_h)
print("Rastgele Ormanlar Modeli Performansı (yüksek korelasyonlu öznitelikler il
print(classification_report(y_test_h, y_pred_h))

print("Confusion Matrix (high corr):")
conf_matrix_rf_h = confusion_matrix(y_test_h, y_pred_h)
print(conf_matrix_rf_h)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_rf_h, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (high corr)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# ROC eğrisi ve AUC (yüksek korelasyonlu öznitelikler ile)
y_test_h_binarized = label_binarize(y_test_h, classes=np.unique(y_test_h))
n_classes_h = y_test_h_binarized.shape[1]
y_score_h = rf_model_h.predict_proba(X_test_h)

fpr_h = dict()
```

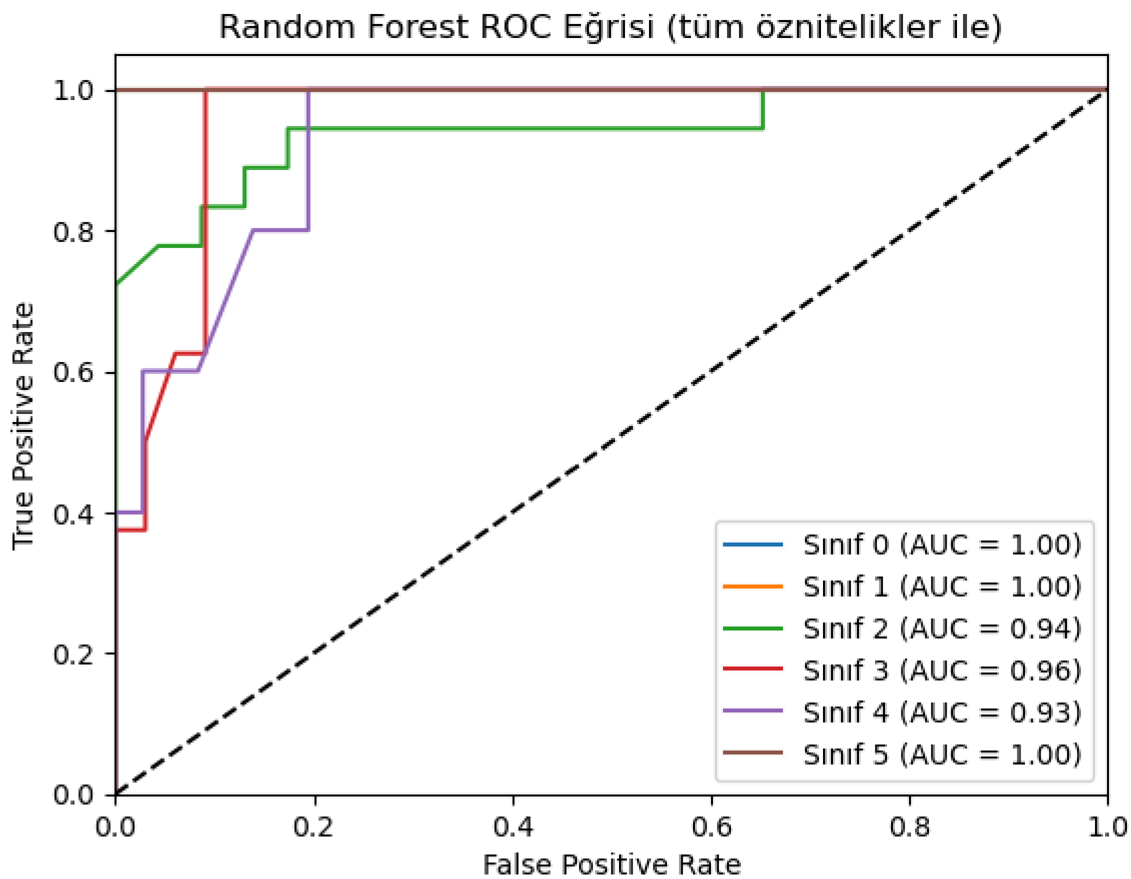
```python
tpr_h = dict()
roc_auc_h = dict()

for i in range(n_classes_h):
    fpr_h[i], tpr_h[i], _ = roc_curve(y_test_h_binarized[:, i], y_score_h[:, i])
    roc_auc_h[i] = auc(fpr_h[i], tpr_h[i])

# Tüm sınıflar için ROC eğrisini çizme (yüksek korelasyonlu öznitelikler)
plt.figure()
for i in range(n_classes_h):
    plt.plot(fpr_h[i], tpr_h[i], label=f'Sınıf {i} (AUC = {roc_auc_h[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Eğrisi (yüksek korelasyonlu öznitelikler ile)')
plt.legend(loc='lower right')
plt.show()
```

```
E:\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
Rastgele Seçilen Girdi (Index: 13):
normalized-losses     65.0
num-of-doors             4
wheel-base           102.4
length               175.6
width                 66.5
                     ...
engine-type_l        False
engine-type_ohc       True
engine-type_ohcf     False
engine-type_ohcv     False
engine-type_rotor    False
Name: 175, Length: 68, dtype: object
Gerçek Değer: -1
Tahmin Edilen Değer: -1

Rastgele Ormanlar Modeli Performansı (tüm öznitelikler ile)
              precision    recall  f1-score   support

          -2       1.00      1.00      1.00         1
          -1       0.71      1.00      0.83         5
           0       1.00      0.72      0.84        18
           1       0.67      1.00      0.80         8
           2       0.67      0.40      0.50         5
           3       0.80      1.00      0.89         4

    accuracy                           0.80        41
   macro avg       0.81      0.85      0.81        41
weighted avg       0.84      0.80      0.80        41
```

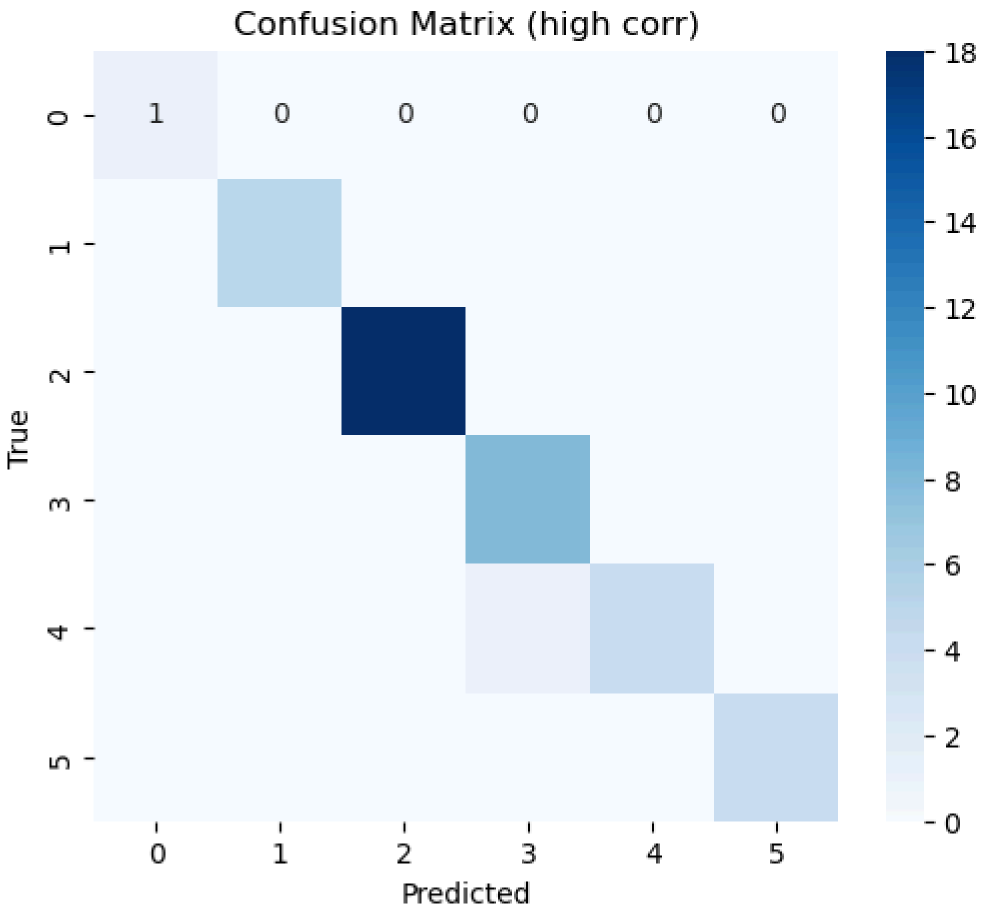


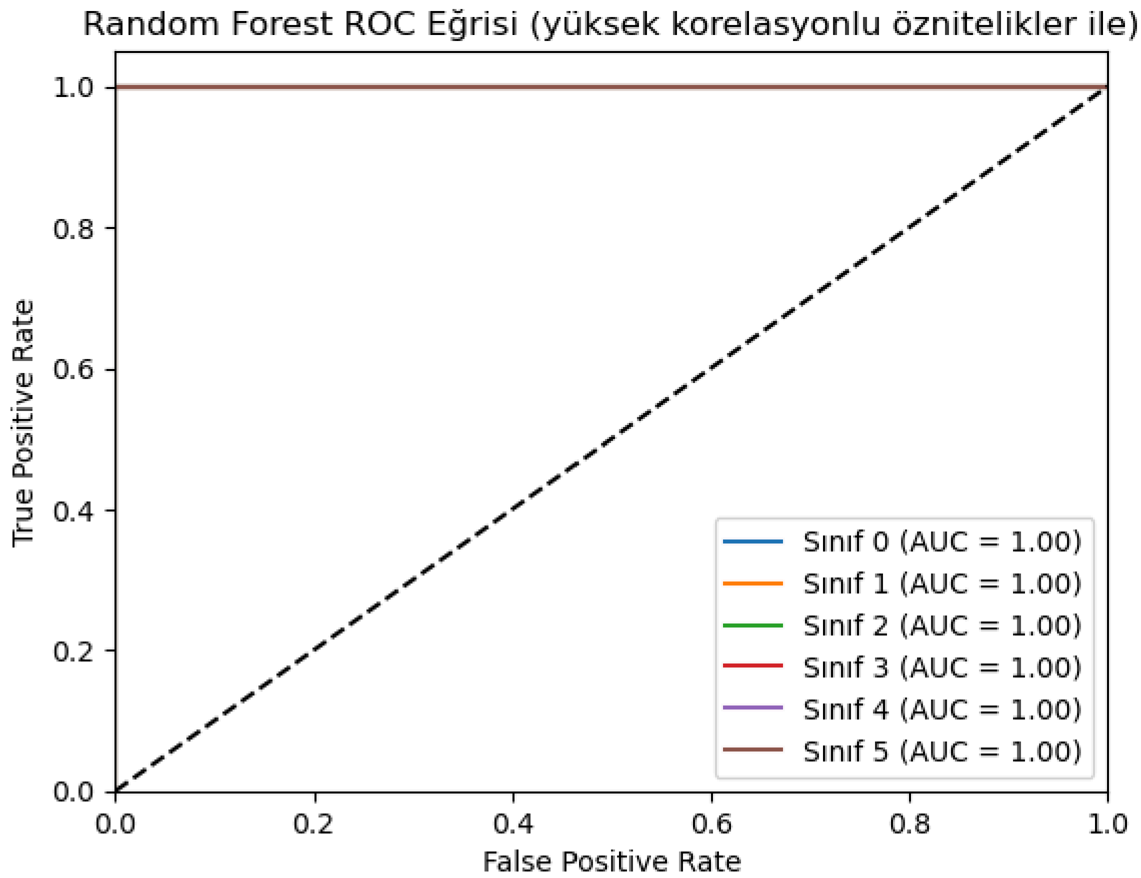

```
Confusion Matrix:
[[ 1  0  0  0  0  0]
 [ 0  5  0  0  0  0]
 [ 0  2 13  1  1  1]
 [ 0  0  0  8  0  0]
 [ 0  0  0  3  2  0]
 [ 0  0  0  0  0  4]]
```


Confusion Matrix
1
0
0
0
0
0
True
Predicted
0
1
2
3
4
5
12
10
8
6
4
2
0


Random Forest ROC Eğrisi (tüm öznitelikler ile)
True Positive Rate
False Positive Rate
Sınıf 0 (AUC = 1.00)
Sınıf 1 (AUC = 1.00)
Sınıf 2 (AUC = 0.94)
Sınıf 3 (AUC = 0.96)
Sınıf 4 (AUC = 0.93)
Sınıf 5 (AUC = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

```
Rastgele Ormanlar Modeli Performansı (yüksek korelasyonlu öznitelikler ile)
              precision    recall  f1-score   support

          -2       1.00      1.00      1.00         1
          -1       1.00      1.00      1.00         5
           0       1.00      1.00      1.00        18
           1       0.89      1.00      0.94         8
           2       1.00      0.80      0.89         5
           3       1.00      1.00      1.00         4

    accuracy                           0.98        41
   macro avg       0.98      0.97      0.97        41
weighted avg       0.98      0.98      0.97        41

Confusion Matrix (high corr):
[[ 1  0  0  0  0  0]
 [ 0  5  0  0  0  0]
 [ 0  0 18  0  0  0]
 [ 0  0  0  8  0  0]
 [ 0  0  0  1  4  0]
 [ 0  0  0  0  0  4]]
```





- Gradient Boosting

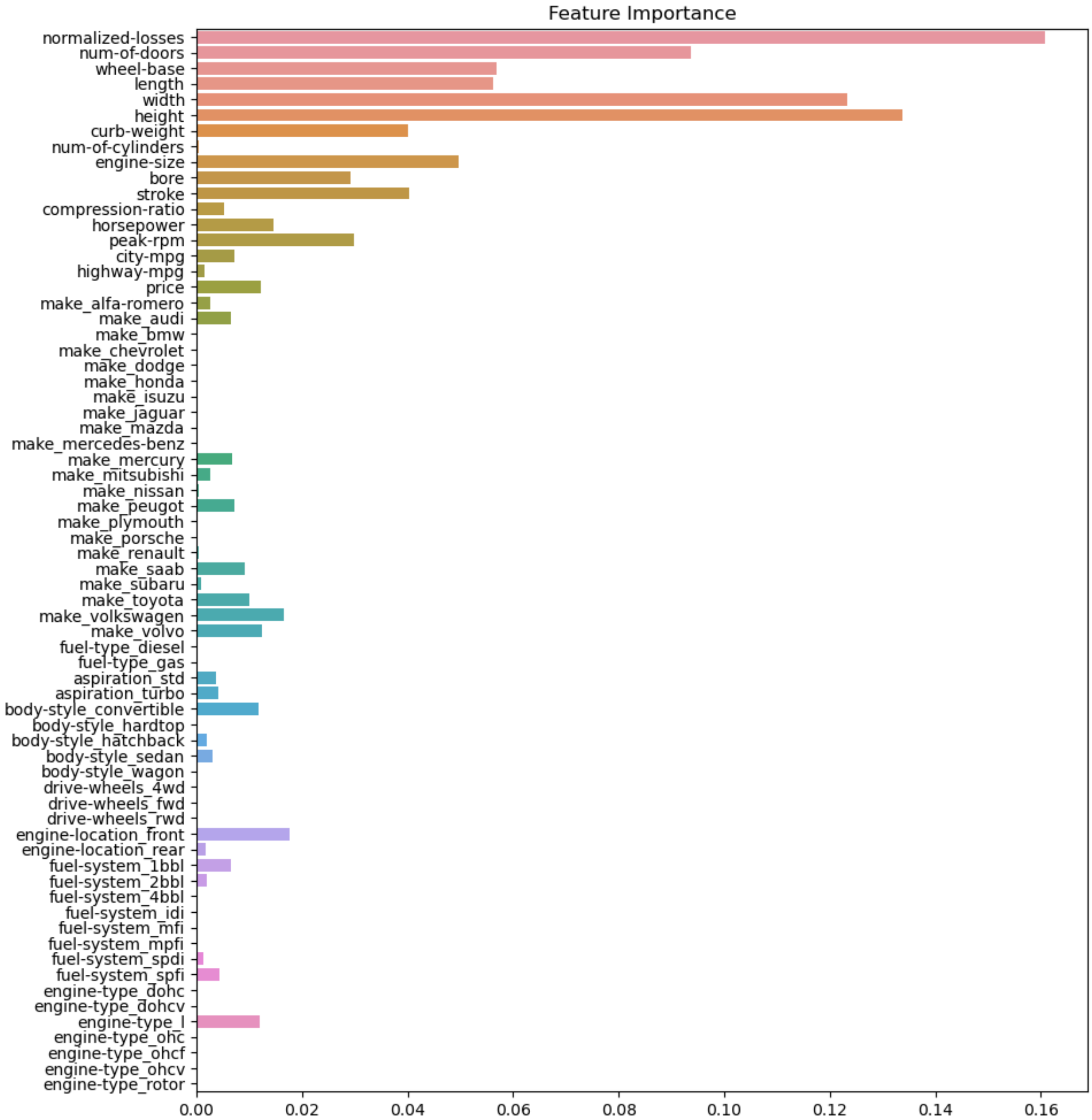
In [12]:

```
# Modeli eğitme
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)

gb_model_h = GradientBoostingClassifier()
gb_model_h.fit(X_train_h, y_train_h)

random_index = np.random.randint(0, X_test.shape[0])
random_test_input = X_test.iloc[random_index].values.reshape(1, -1)

random_prediction = gb_model.predict(random_test_input)

print(f"Rastgele Seçilen Girdi (Index: {random_index}):")
print(X_test.iloc[random_index])
print(f"Gerçek Değer: {y_test.iloc[random_index]}")
print(f"Tahmin Edilen Değer: {random_prediction[0]}")

# Tahmin ve değerlendirme (tüm öznitelikler ile)
y_pred = gb_model.predict(X_test)
print("Gradient Boosting Modeli Performansı (tüm öznitelikler ile)")
print(classification_report(y_test, y_pred))

feature_importance = gb_model.feature_importances_
features = X_train.columns

# Görselleştirme
plt.figure(figsize=(10, 12))
sns.barplot(x=feature_importance, y=features)
plt.title('Feature Importance')
```

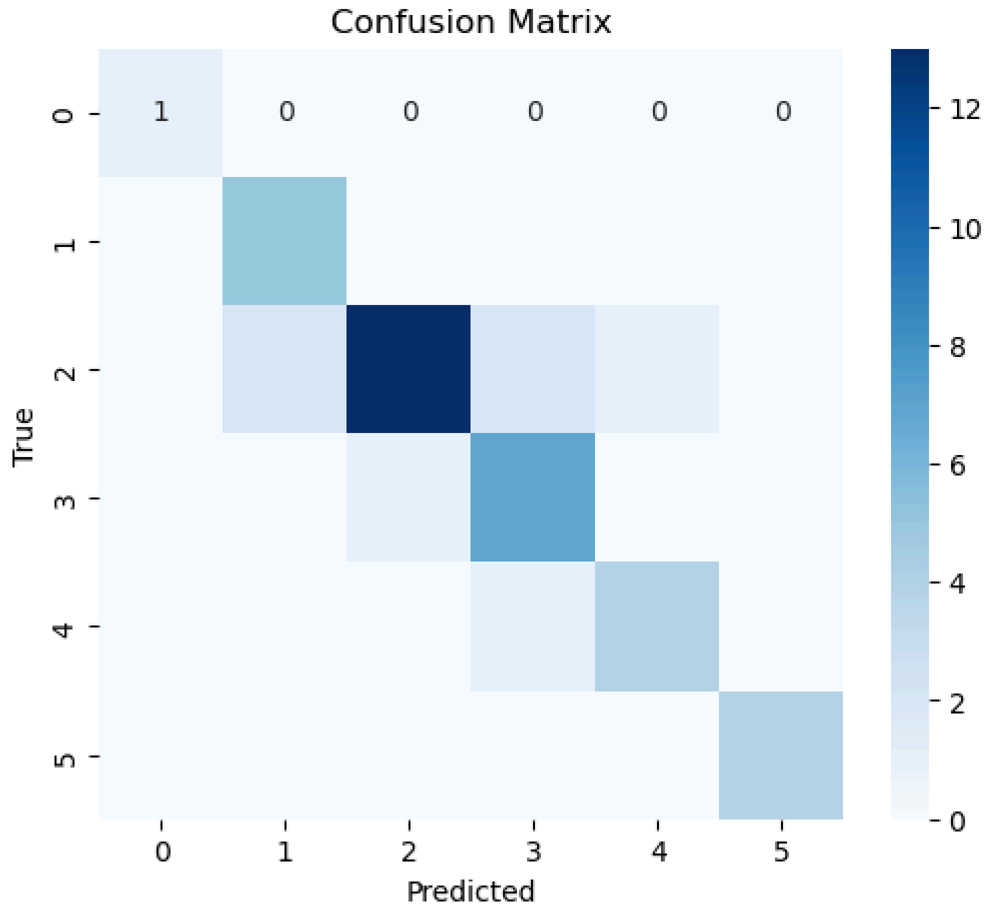
```
plt.show()


print("Confusion Matrix:")
conf_matrix_gb = confusion_matrix(y_test, y_pred)
print(conf_matrix_gb)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_gb, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# ROC eğrisi ve AUC (tüm öznitelikler ile)
y_test_binarized = label_binarize(y_test, classes=np.unique(y_test))
n_classes = y_test_binarized.shape[1]
y_score = gb_model.predict_proba(X_test)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Tüm sınıflar için ROC eğrisini çizme
plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Sınıf {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Gradient Boosting ROC Eğrisi (tüm öznitelikler ile)')
plt.legend(loc='lower right')
plt.show()


# Tahmin ve değerlendirme (yüksek korelasyonlu öznitelikler ile)
y_pred_h = gb_model_h.predict(X_test_h)
print("Gradient Boosting Modeli Performansı (yüksek korelasyonlu öznitelikler il
print(classification_report(y_test_h, y_pred_h))

print("Confusion Matrix (high corr):")
conf_matrix_gb_h = confusion_matrix(y_test_h, y_pred_h)
print(conf_matrix_gb_h)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix_gb_h, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix (high corr)')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# ROC eğrisi ve AUC (yüksek korelasyonlu öznitelikler ile)
y_test_h_binarized = label_binarize(y_test_h, classes=np.unique(y_test_h))
```

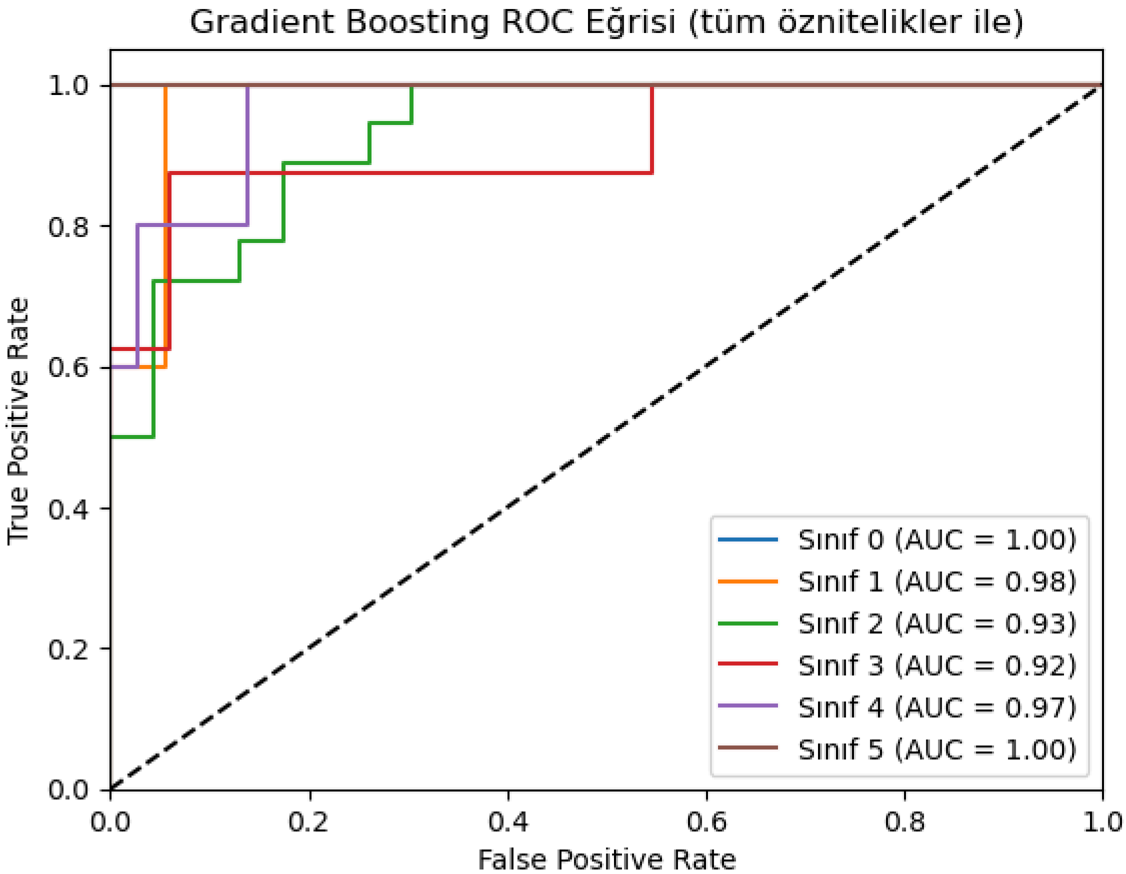
```python
n_classes_h = y_test_h_binarized.shape[1]
y_score_h = gb_model_h.predict_proba(X_test_h)

fpr_h = dict()
tpr_h = dict()
roc_auc_h = dict()

for i in range(n_classes_h):
    fpr_h[i], tpr_h[i], _ = roc_curve(y_test_h_binarized[:, i], y_score_h[:, i])
    roc_auc_h[i] = auc(fpr_h[i], tpr_h[i])

# Tüm sınıflar için ROC eğrisini çizme (yüksek korelasyonlu öznitelikler)
plt.figure()
for i in range(n_classes_h):
    plt.plot(fpr_h[i], tpr_h[i], label=f'Sınıf {i} (AUC = {roc_auc_h[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Gradient Boosting ROC Eğrisi (yüksek korelasyonlu öznitelikler ile)')
plt.legend(loc='lower right')
plt.show()
```

```
E:\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have
valid feature names, but GradientBoostingClassifier was fitted with feature names
  warnings.warn(
Rastgele Seçilen Girdi (Index: 32):
normalized-losses     81.0
num-of-doors             4
wheel-base            95.7
length               169.7
width                 63.6
                     ...
engine-type_l        False
engine-type_ohc       True
engine-type_ohcf     False
engine-type_ohcv     False
engine-type_rotor    False
Name: 154, Length: 68, dtype: object
Gerçek Değer: 0
Tahmin Edilen Değer: 0
Gradient Boosting Modeli Performansı (tüm öznitelikler ile)
              precision    recall  f1-score   support

          -2       1.00      1.00      1.00         1
          -1       0.71      1.00      0.83         5
           0       0.93      0.72      0.81        18
           1       0.70      0.88      0.78         8
           2       0.80      0.80      0.80         5
           3       1.00      1.00      1.00         4

    accuracy                           0.83        41
   macro avg       0.86      0.90      0.87        41
weighted avg       0.85      0.83      0.83        41
```

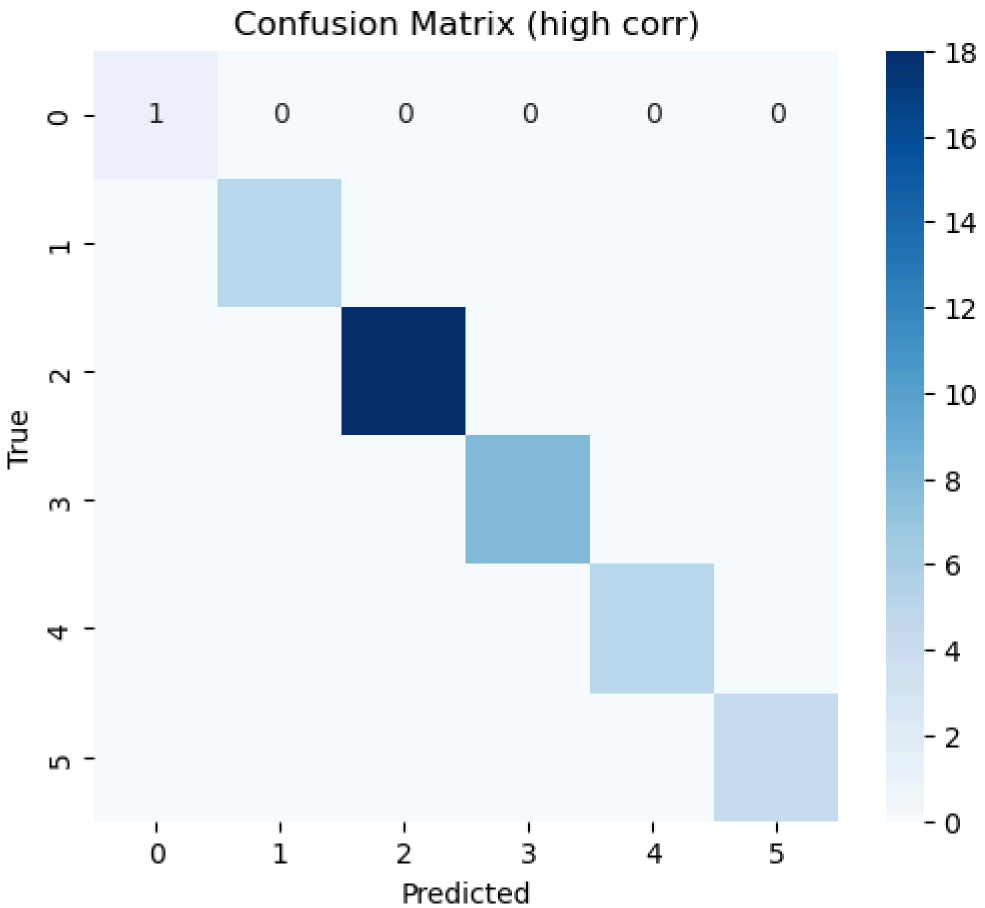


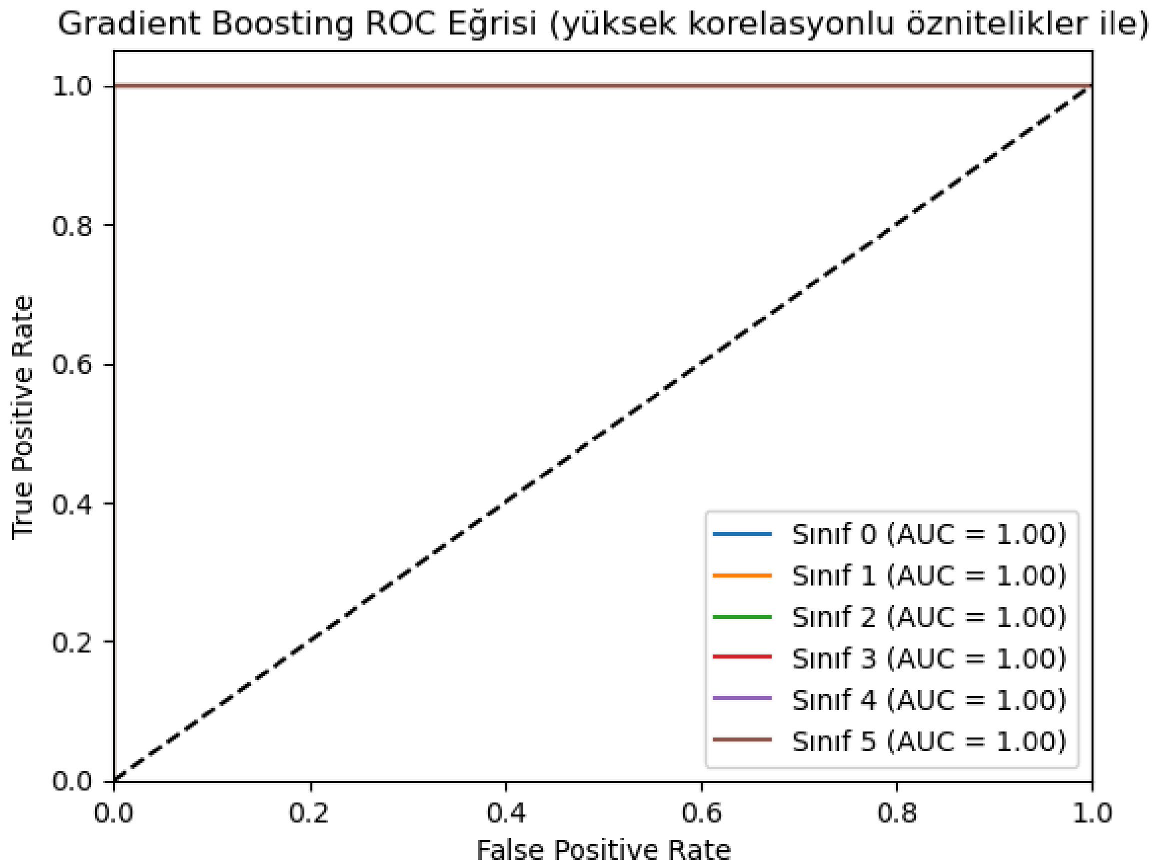

```
Confusion Matrix:
[[ 1  0  0  0  0  0]
 [ 0  5  0  0  0  0]
 [ 0  2 13  2  1  0]
 [ 0  0  1  7  0  0]
 [ 0  0  0  1  4  0]
 [ 0  0  0  0  0  4]]
```


Confusion Matrix
1
0
0
0
0
0
True
Predicted
0
1
2
3
4
5
0
2
4
6
8
10
12


Gradient Boosting ROC Eğrisi (tüm öznitelikler ile)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Sınıf 0 (AUC = 1.00)
Sınıf 1 (AUC = 0.98)
Sınıf 2 (AUC = 0.93)
Sınıf 3 (AUC = 0.92)
Sınıf 4 (AUC = 0.97)
Sınıf 5 (AUC = 1.00)

```
Gradient Boosting Modeli Performansı (yüksek korelasyonlu öznitelikler ile)
              precision    recall  f1-score   support

          -2       1.00      1.00      1.00         1
          -1       1.00      1.00      1.00         5
           0       1.00      1.00      1.00        18
           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00         4

    accuracy                           1.00        41
   macro avg       1.00      1.00      1.00        41
weighted avg       1.00      1.00      1.00        41

Confusion Matrix (high corr):
[[ 1  0  0  0  0  0]
 [ 0  5  0  0  0  0]
 [ 0  0 18  0  0  0]
 [ 0  0  0  8  0  0]
 [ 0  0  0  0  5  0]
 [ 0  0  0  0  0  4]]
```





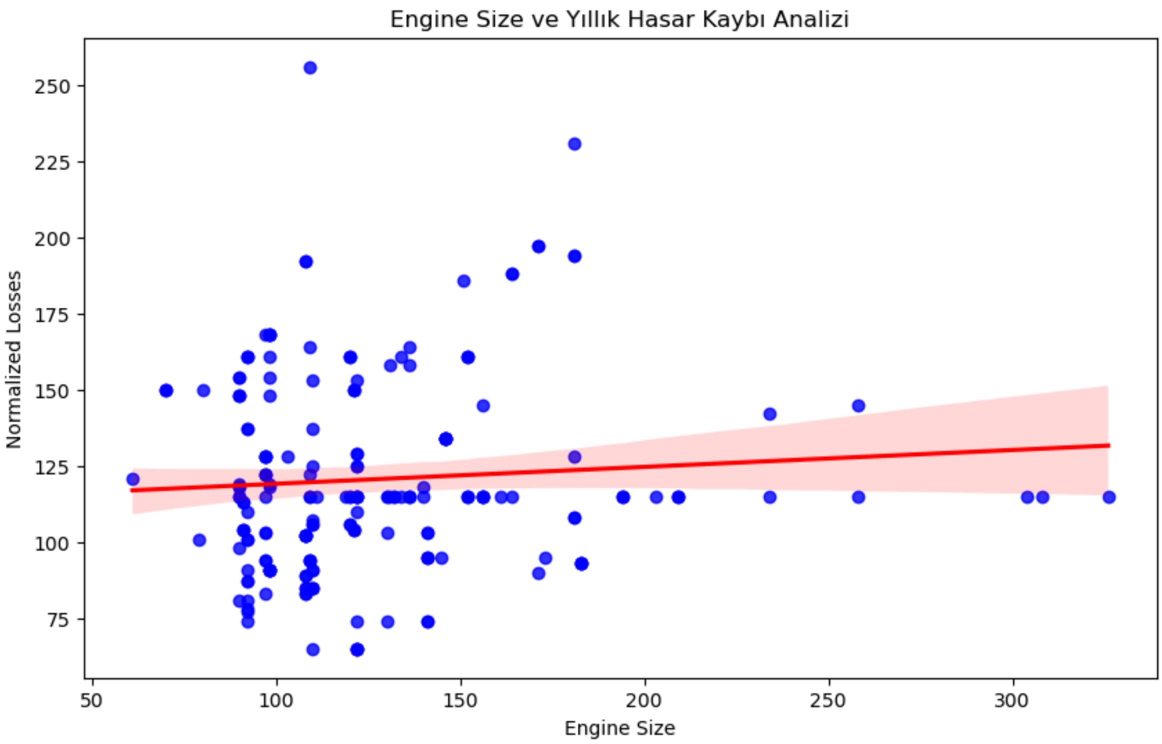
5. Ortalama Yıllık Hasar Kaybı Analizi

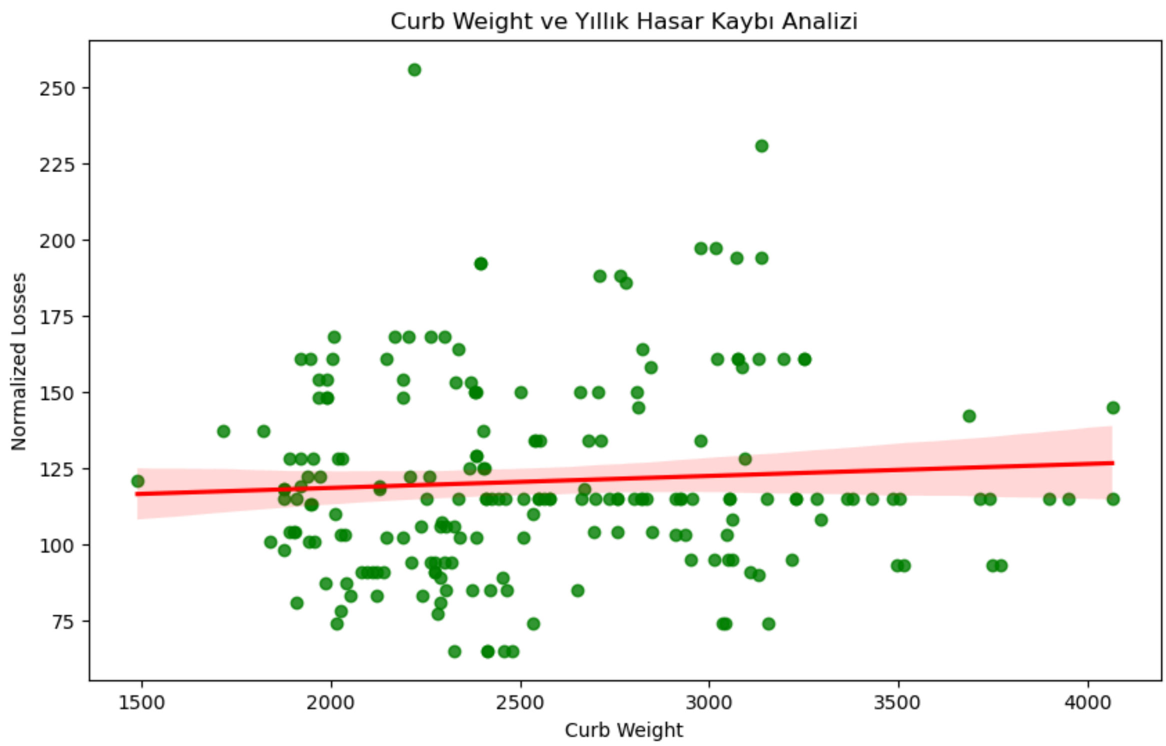
In [13]:

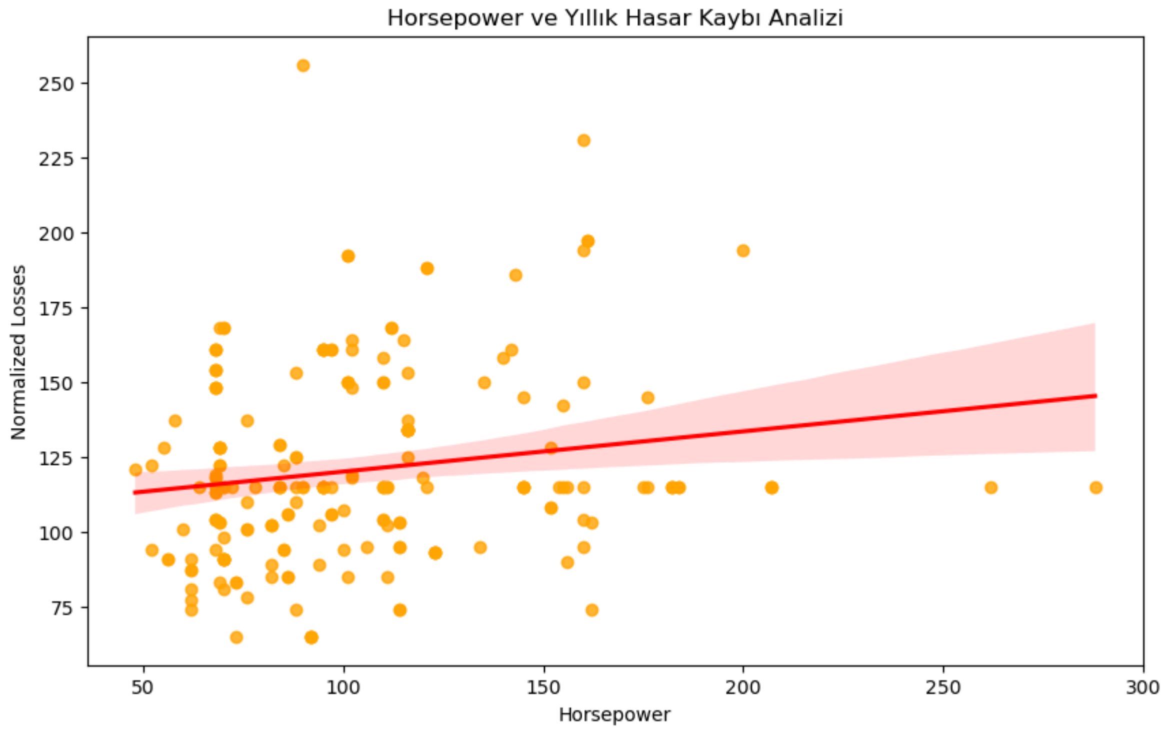
```
# Bağımsız ve bağımlı değişkenlerin tanımlanması
X = df[['engine-size', 'curb-weight', 'horsepower']]
y = df['normalized-losses']

X = sm.add_constant(X)

# Regresyon modelini oluşturma
model = sm.OLS(y, X).fit()

# Model özeti
print(model.summary())

# engine-size vs normalized-losses
plt.figure(figsize=(10, 6))
sns.regplot(x='engine-size', y='normalized-losses', data=df, scatter_kws={'color
plt.title('Engine Size ve Yıllık Hasar Kaybı Analizi')
plt.xlabel('Engine Size')
plt.ylabel('Normalized Losses')
plt.show()

# curb-weight vs normalized-losses
plt.figure(figsize=(10, 6))
sns.regplot(x='curb-weight', y='normalized-losses', data=df, scatter_kws={'color
plt.title('Curb Weight ve Yıllık Hasar Kaybı Analizi')
plt.xlabel('Curb Weight')
plt.ylabel('Normalized Losses')
plt.show()
```

```python
# horsepower vs normalized-losses
plt.figure(figsize=(10, 6))
sns.regplot(x='horsepower', y='normalized-losses', data=df, scatter_kws={'color'
plt.title('Horsepower ve Yıllık Hasar Kaybı Analizi')
plt.xlabel('Horsepower')
plt.ylabel('Normalized Losses')
plt.show()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      normalized-losses   R-squared:                       0.040
Model:                            OLS   Adj. R-squared:                  0.026
Method:                 Least Squares   F-statistic:                     2.821
Date:                Sun, 11 Aug 2024   Prob (F-statistic):             0.0400
Time:                        08:21:20   Log-Likelihood:                -995.38
No. Observations:                 205   AIC:                             1999.
Df Residuals:                     201   BIC:                             2012.
Df Model:                           3
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const         116.5116     12.048      9.671      0.000      92.755     140.268
engine-size    -0.1054      0.115     -0.913      0.362      -0.333       0.122
curb-weight    -0.0039      0.008     -0.472      0.637      -0.020       0.012
horsepower      0.2625      0.097      2.712      0.007       0.072       0.453
==============================================================================
Omnibus:                       32.204   Durbin-Watson:                   1.021
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.271
Skew:                           0.938   Prob(JB):                     1.48e-10
Kurtosis:                       4.335   Cond. No.                     1.44e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.44e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```




Curb Weight ve Yıllık Hasar Kaybı Analizi
Normalized Losses
250
225
200
175
150
125
100
75
1500
2000
2500
3000
3500
4000
Curb Weight


Horsepower ve Yıllık Hasar Kaybı Analizi
Normalized Losses
250
225
200
175
150
125
100
75
50
100
150
200
250
300
Horsepower