```
#!pip install shap
#!pip install kagglehub
# Notwendige Bibliotheken importieren
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression,
Lasso
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

df = pd.read_csv("car details v4.csv")

# # Beispiel-Datensatz (Iris-Datensatz)
# data = load_iris()
# X = pd.DataFrame(data.data, columns=data.feature_names)
# y = pd.Series(data.target)

df
```

```
               Make                                 Model    Price
Year  \
0             Honda                   Amaze 1.2 VX i-VTEC   505000
2017
1     Maruti Suzuki                     Swift DZire VDI   450000
2014
2           Hyundai                i10 Magna 1.2 Kappa2   220000
2011
3            Toyota                            Glanza G   799000
2019
4            Toyota        Innova 2.4 VX 7 STR [2016-2020]  1950000
2018
...             ...                                   ...      ...
...
2054       Mahindra              XUV500 W8 [2015-2017]   850000
2016
2055        Hyundai                         Eon D-Lite +   275000
2014
2056           Ford          Figo Duratec Petrol ZXI 1.2   240000
2013
2057            BMW  5-Series 520d Luxury Line [2017-2019]  4290000
2018
2058       Mahindra      Bolero Power Plus ZLX [2016-2019]   670000
2017
```

```
      Kilometer Fuel Type Transmission    Location   Color   Owner  \
0         87150    Petrol       Manual        Pune    Grey   First
1         75000    Diesel       Manual    Ludhiana   White  Second
2         67000    Petrol       Manual     Lucknow  Maroon   First
3         37500    Petrol       Manual   Mangalore     Red   First
4         69000    Diesel       Manual      Mumbai    Grey   First
...         ...       ...          ...         ...     ...     ...
2054      90300    Diesel       Manual       Surat   White   First
2055      83000    Petrol       Manual   Ahmedabad   White  Second
2056      73000    Petrol       Manual       Thane  Silver   First
2057      60474    Diesel    Automatic  Coimbatore   White   First
2058      72000    Diesel       Manual    Guwahati   White   First

     Seller Type   Engine           Max Power              Max Torque
\
0      Corporate  1198 cc   87 bhp @ 6000 rpm       109 Nm @ 4500 rpm

1     Individual  1248 cc   74 bhp @ 4000 rpm       190 Nm @ 2000 rpm

2     Individual  1197 cc   79 bhp @ 6000 rpm  112.7619 Nm @ 4000 rpm

3     Individual  1197 cc   82 bhp @ 6000 rpm       113 Nm @ 4200 rpm

4     Individual  2393 cc  148 bhp @ 3400 rpm       343 Nm @ 1400 rpm

...          ...      ...                 ...                     ...

2054  Individual  2179 cc  138 bhp @ 3750 rpm       330 Nm @ 1600 rpm

2055  Individual   814 cc   55 bhp @ 5500 rpm        75 Nm @ 4000 rpm

2056  Individual  1196 cc   70 bhp @ 6250 rpm       102 Nm @ 4000 rpm

2057  Individual  1995 cc  188 bhp @ 4000 rpm       400 Nm @ 1750 rpm

2058  Individual  1493 cc   70 bhp @ 3600 rpm       195 Nm @ 1400 rpm


     Drivetrain  Length   Width  Height  Seating Capacity  Fuel Tank
Capacity
0           FWD  3990.0  1680.0  1505.0               5.0
35.0
1           FWD  3995.0  1695.0  1555.0               5.0
42.0
2           FWD  3585.0  1595.0  1550.0               5.0
35.0
3           FWD  3995.0  1745.0  1510.0               5.0
37.0
4           RWD  4735.0  1830.0  1795.0               7.0
55.0
```

```
...          ...     ...     ...     ...                  ...
...
2054         FWD  4585.0  1890.0  1785.0                  7.0
70.0
2055         FWD  3495.0  1550.0  1500.0                  5.0
32.0
2056         FWD  3795.0  1680.0  1427.0                  5.0
45.0
2057         RWD  4936.0  1868.0  1479.0                  5.0
65.0
2058         RWD  3995.0  1745.0  1880.0                  7.0
NaN

[2059 rows x 20 columns]
```

```
df[['Max Power Value', 'Max Power RPM']] = df['Max
Power'].str.extract(r'(\d+)\D*(\d+)')
df[['Max Torque Value', 'Max Torque RPM']] = df['Max
Torque'].str.extract(r'(\d+)\D*(\d+)')
df['Engine Capacity'] = df['Engine'].str.split(" ",
expand=True).iloc[:, 0]
# To make year having more impact and be more precise we would
substract them
# from max year, cause newer the car -> more value it gets
max_year = df['Year'].max()
df['Car exploitet in years'] = max_year - df['Year']

# Convert the extracted values to numeric
df['Max Power Value'] = pd.to_numeric(df['Max Power Value'])
df['Max Power RPM'] = pd.to_numeric(df['Max Power RPM'])
df['Max Torque Value'] = pd.to_numeric(df['Max Torque Value'])
df['Max Torque RPM'] = pd.to_numeric(df['Max Torque RPM'])

df = df.drop(columns=['Max Power', 'Max Torque', 'Engine', 'Year'])

# Collect columns to drop
columns_to_drop = []
object_columns = []
numeric_columns = []

# Loop through columns to check their type and unique values
for i in df.columns:
    if df[i].dtype == 'object' and len(df[i].unique()) > 50:
        columns_to_drop.append(i)

# Drop the identified columns
df = df.drop(columns=columns_to_drop)

df
```

```
                 Make    Price  Kilometer Fuel Type Transmission   Color  \
0               Honda   505000      87150    Petrol       Manual    Grey
1       Maruti Suzuki   450000      75000    Diesel       Manual   White
2             Hyundai   220000      67000    Petrol       Manual  Maroon
3              Toyota   799000      37500    Petrol       Manual     Red
4              Toyota  1950000      69000    Diesel       Manual    Grey
...               ...      ...        ...       ...          ...     ...
2054         Mahindra   850000      90300    Diesel       Manual   White
2055          Hyundai   275000      83000    Petrol       Manual   White
2056             Ford   240000      73000    Petrol       Manual  Silver
2057              BMW  4290000      60474    Diesel    Automatic   White
2058         Mahindra   670000      72000    Diesel       Manual   White

       Owner Seller Type Drivetrain  Length   Width  Height  Seating Capacity  \
0      First   Corporate        FWD  3990.0  1680.0  1505.0               5.0
1     Second  Individual        FWD  3995.0  1695.0  1555.0               5.0
2      First  Individual        FWD  3585.0  1595.0  1550.0               5.0
3      First  Individual        FWD  3995.0  1745.0  1510.0               5.0
4      First  Individual        RWD  4735.0  1830.0  1795.0               7.0
...      ...         ...        ...     ...     ...     ...               ...
2054   First  Individual        FWD  4585.0  1890.0  1785.0               7.0
2055  Second  Individual        FWD  3495.0  1550.0  1500.0               5.0
2056   First  Individual        FWD  3795.0  1680.0  1427.0               5.0
2057   First  Individual        RWD  4936.0  1868.0  1479.0               5.0
2058   First  Individual        RWD  3995.0  1745.0  1880.0               7.0

      Fuel Tank Capacity  Max Power Value  Max Power RPM  Max Torque
```

```
Value  \
0                     35.0              87.0          6000.0
109.0
1                     42.0              74.0          4000.0
190.0
2                     35.0              79.0          6000.0
112.0
3                     37.0              82.0          6000.0
113.0
4                     55.0             148.0          3400.0
343.0
...                    ...               ...             ...
...
2054                  70.0             138.0          3750.0
330.0
2055                  32.0              55.0          5500.0
75.0
2056                  45.0              70.0          6250.0
102.0
2057                  65.0             188.0          4000.0
400.0
2058                   NaN              70.0          3600.0
195.0

      Max Torque RPM  Car exploitet in years
0             4500.0                       5
1             2000.0                       8
2             7619.0                      11
3             4200.0                       3
4             1400.0                       4
...              ...                     ...
2054          1600.0                       6
2055          4000.0                       8
2056          4000.0                       9
2057          1750.0                       4
2058          1400.0                       5

[2059 rows x 19 columns]
```

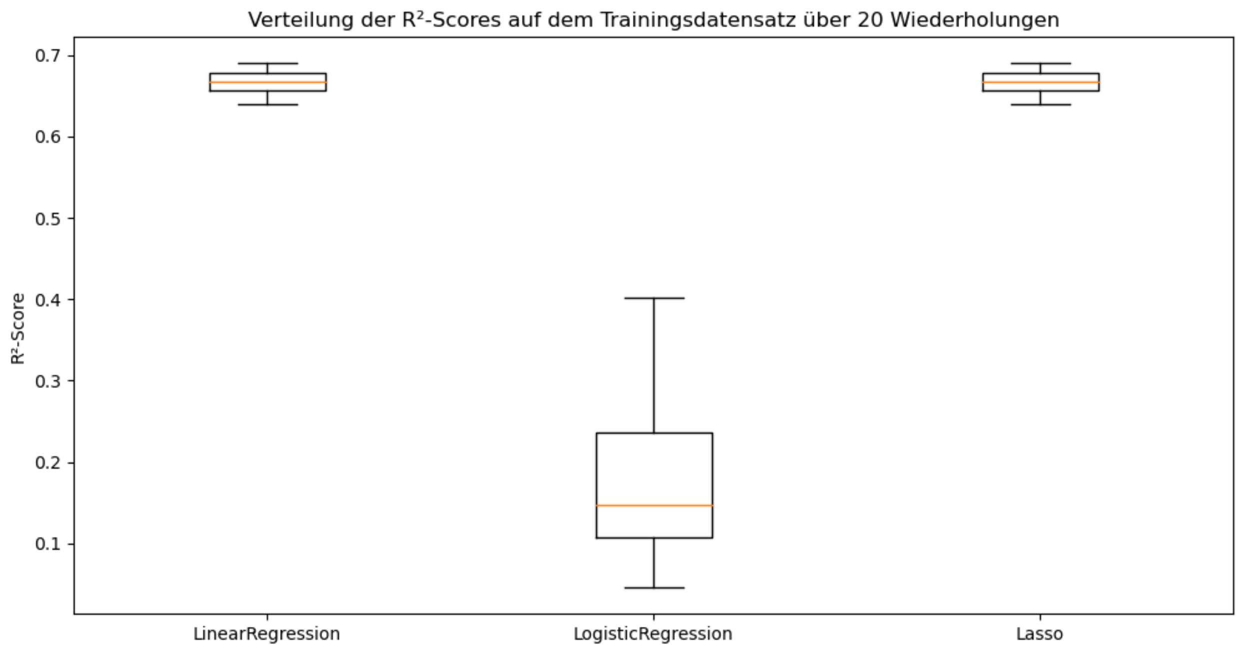
```
for i in df.columns:
    if df[i].dtype == 'object':
        object_columns.append(i)
    else:
        numeric_columns.append(i)

# Fill missing values with most occuring string
for col in df.select_dtypes(include=['object']).columns:
    most_frequent_value = df[col].mode()[0]  # Get the most frequent 
value
    df[col].fillna(most_frequent_value, inplace=True)
```

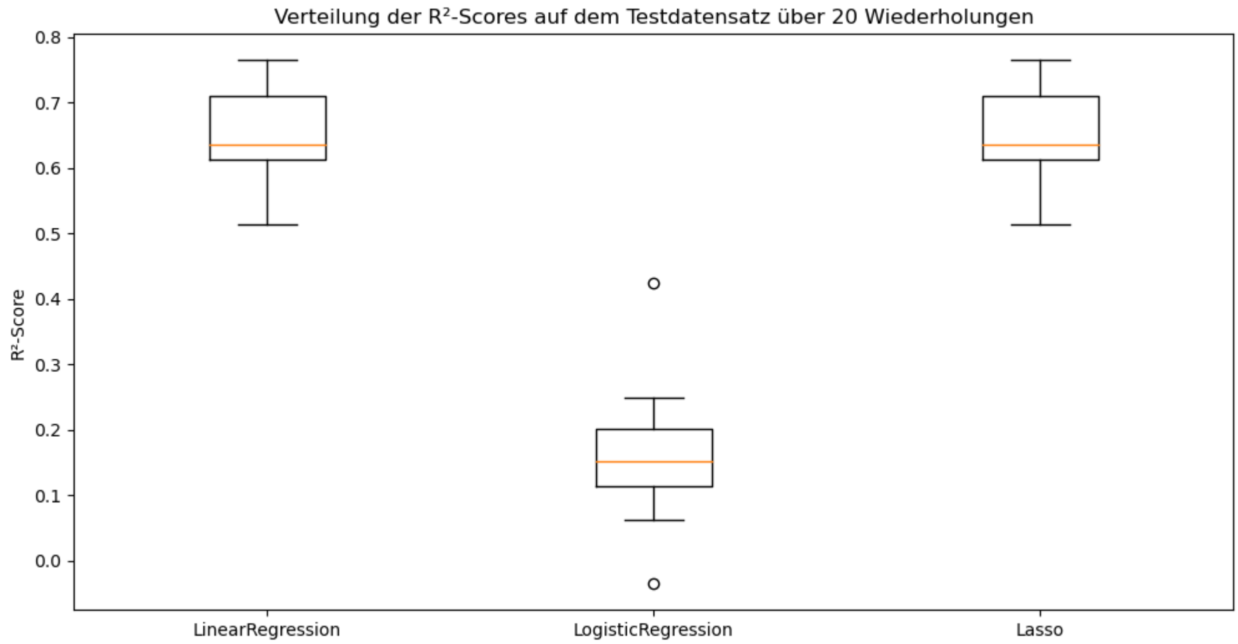
```
# Fill missing values for numeric columns with the average value
(mean)
for col in df.select_dtypes(include=['number']).columns:
    mean_value = df[col].mean()  # Get the mean value
    df[col].fillna(mean_value, inplace=True)

y = df["Price"]
X = df.drop(columns=["Price"])

#First we train without our string feautures

X = X.drop(columns=object_columns)



# Lineare Regression, Lasso, Logistische Regression



# Listen zur Speicherung der R²-Scores
r2_train_scores = {"LinearRegression": [], "LogisticRegression": [],
"Lasso": []}
r2_test_scores = {"LinearRegression": [], "LogisticRegression": [],
"Lasso": []}

# Wiederholen der Schritte 20 Mal
for i in range(20):


    # Teilen der Daten in Trainings- und Testdatensatz
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=i)

    # Standardisieren der Daten
    scaler = MinMaxScaler()

    # Fit the scaler on Xtrain and transform Xtrain
    Xtrain_scaled = scaler.fit_transform(X_train)

    # Transform Xtest using the same scaler (no fitting)
    Xtest_scaled = scaler.transform(X_test)
    # Initialisierung der Modelle
    models = {
        "LinearRegression": LinearRegression(),
        "LogisticRegression": LogisticRegression(max_iter=1000),
        "Lasso": Lasso(alpha=0.1)
    }

    # Trainieren der Modelle und Vorhersage auf dem Testdatensatz
    for model_name, model in models.items():
        model.fit(Xtrain_scaled, y_train)
```

```
        # Vorhersage auf dem Testdatensatz
        ytest_pred = model.predict(Xtest_scaled)

        # Berechnung des R²-Score für den Testdatensatz
        r2_test = r2_score(y_test, ytest_pred)
        r2_test_scores[model_name].append(r2_test)

        # Berechnung des R²-Score für den Trainingsdatensatz
        ytrain_pred = model.predict(Xtrain_scaled)
        r2_train = r2_score(y_train, ytrain_pred)
        r2_train_scores[model_name].append(r2_train)

# Berechnung der Durchschnittswerte und Standardabweichungen
mean_r2_train = {model_name: np.mean(scores) for model_name, scores in
r2_train_scores.items()}
std_r2_train = {model_name: np.std(scores) for model_name, scores in
r2_train_scores.items()}

mean_r2_test = {model_name: np.mean(scores) for model_name, scores in
r2_test_scores.items()}
std_r2_test = {model_name: np.std(scores) for model_name, scores in
r2_test_scores.items()}

# Ausgabe der Ergebnisse
for model_name in r2_train_scores.keys():
    print(f"Modell: {model_name}")
    print(f"Durchschnittlicher R²-Score auf dem Trainingsdatensatz:
{mean_r2_train[model_name]} ± {std_r2_train[model_name]}")
    print(f"Durchschnittlicher R²-Score auf dem Testdatensatz:
{mean_r2_test[model_name]} ± {std_r2_test[model_name]}")
    print()

# Erstellen eines Boxplots zur Visualisierung der Stabilität
plt.figure(figsize=(12, 6))
plt.boxplot([r2_train_scores["LinearRegression"],
r2_train_scores["LogisticRegression"], r2_train_scores["Lasso"]],
            labels=['LinearRegression', 'LogisticRegression',
'Lasso'])
plt.title('Verteilung der R²-Scores auf dem Trainingsdatensatz über 20
Wiederholungen')
plt.ylabel('R²-Score')
plt.show()

plt.figure(figsize=(12, 6))
plt.boxplot([r2_test_scores["LinearRegression"],
r2_test_scores["LogisticRegression"], r2_test_scores["Lasso"]],
            labels=['LinearRegression', 'LogisticRegression',
'Lasso'])
plt.title('Verteilung der R²-Scores auf dem Testdatensatz über 20
```

```
Wiederholungen')
plt.ylabel('R²-Score')
plt.show()
```

```
Modell: LinearRegression
Durchschnittlicher R²-Score auf dem Trainingsdatensatz:
0.6665625400898865 ± 0.013493388094843267
Durchschnittlicher R²-Score auf dem Testdatensatz: 0.6509093641868238
± 0.061456243891206075

Modell: LogisticRegression
Durchschnittlicher R²-Score auf dem Trainingsdatensatz:
0.17263511164669104 ± 0.08969069171501731
Durchschnittlicher R²-Score auf dem Testdatensatz: 0.15924237242781597
± 0.09018961926224281

Modell: Lasso
Durchschnittlicher R²-Score auf dem Trainingsdatensatz:
0.6665625400806466 ± 0.013493388092812662
Durchschnittlicher R²-Score auf dem Testdatensatz: 0.6509097608315882
± 0.061456060150388944
```

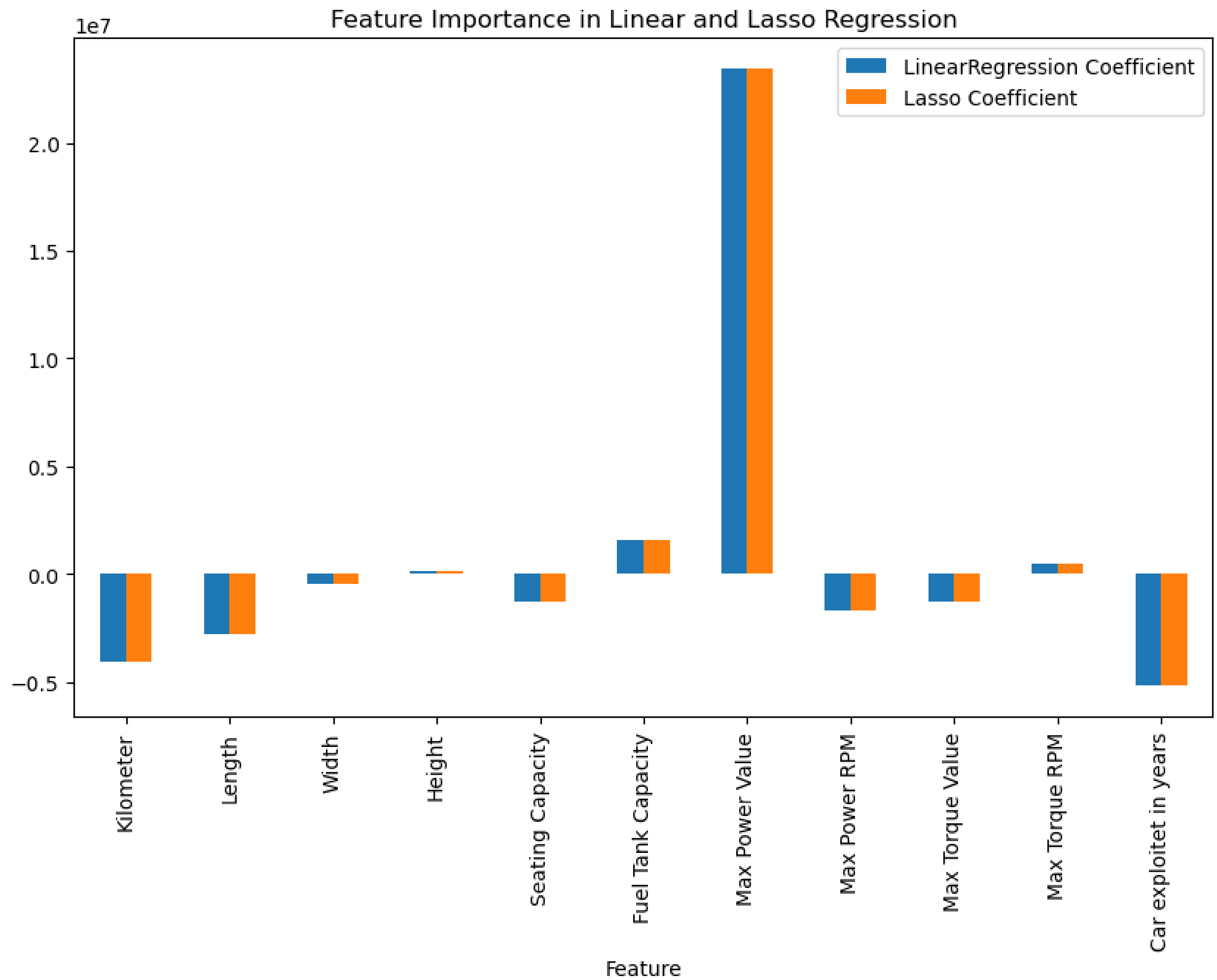





```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, Lasso
import matplotlib.pyplot as plt

# Hier verwenden wir Xtrain_scaled, Xtest_scaled und y_train, y_test

# Lineares Regressionsmodell
linear_model = LinearRegression()
linear_model.fit(Xtrain_scaled, y_train)

# Lasso-Modell
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(Xtrain_scaled, y_train)

# Koeffizienten für die Modelle extrahieren
linear_coefficients = linear_model.coef_
lasso_coefficients = lasso_model.coef_

# Feature-Namen
feature_names = X.columns

# Erstellen eines DataFrames zur Darstellung der Koeffizienten
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'LinearRegression Coefficient': linear_coefficients,
    'Lasso Coefficient': lasso_coefficients
})

# Anzeigen der Wichtigkeit der Features
```

```
print(importance_df)

# Visualisierung der Feature-Wichtigkeit
importance_df.set_index('Feature').plot(kind='bar', figsize=(10, 6),
title="Feature Importance in Linear and Lasso Regression")
plt.show()
```

```
                   Feature  LinearRegression Coefficient  Lasso
Coefficient
0                Kilometer                 -4.029741e+06      -
4.029629e+06
1                   Length                 -2.746036e+06      -
2.746024e+06
2                    Width                 -4.804101e+05      -
4.803992e+05
3                   Height                  1.874096e+05
1.873935e+05
4         Seating Capacity                 -1.297022e+06      -
1.297009e+06
5       Fuel Tank Capacity                  1.571192e+06
1.571166e+06
6          Max Power Value                  2.340806e+07
2.340782e+07
7            Max Power RPM                 -1.707493e+06      -
1.707370e+06
8         Max Torque Value                 -1.247786e+06      -
1.247634e+06
9           Max Torque RPM                  4.942981e+05
4.942693e+05
10  Car exploitet in years                 -5.183842e+06      -
5.183841e+06
```




```
# object merkmale wieder einfügen
X = df

X

                Make    Price  Kilometer Fuel Type Transmission   Color  \
0              Honda   505000      87150    Petrol       Manual    Grey

1      Maruti Suzuki   450000      75000    Diesel       Manual   White

2            Hyundai   220000      67000    Petrol       Manual  Maroon

3             Toyota   799000      37500    Petrol       Manual     Red

4             Toyota  1950000      69000    Diesel       Manual    Grey

...              ...      ...        ...       ...          ...     ...

2054        Mahindra   850000      90300    Diesel       Manual   White
```

```
2055        Hyundai   275000      83000    Petrol       Manual   White
2056           Ford   240000      73000    Petrol       Manual  Silver
2057            BMW  4290000      60474    Diesel    Automatic   White
2058       Mahindra   670000      72000    Diesel       Manual   White

       Owner Seller Type Drivetrain  Length   Width  Height  Seating
Capacity  \
0      First   Corporate        FWD  3990.0  1680.0  1505.0
5.0
1     Second  Individual        FWD  3995.0  1695.0  1555.0
5.0
2      First  Individual        FWD  3585.0  1595.0  1550.0
5.0
3      First  Individual        FWD  3995.0  1745.0  1510.0
5.0
4      First  Individual        RWD  4735.0  1830.0  1795.0
7.0
...      ...         ...        ...     ...     ...     ...
...
2054   First  Individual        FWD  4585.0  1890.0  1785.0
7.0
2055  Second  Individual        FWD  3495.0  1550.0  1500.0
5.0
2056   First  Individual        FWD  3795.0  1680.0  1427.0
5.0
2057   First  Individual        RWD  4936.0  1868.0  1479.0
5.0
2058   First  Individual        RWD  3995.0  1745.0  1880.0
7.0

      Fuel Tank Capacity  Max Power Value  Max Power RPM  Max Torque
Value  \
0               35.00000             87.0         6000.0
109.0
1               42.00000             74.0         4000.0
190.0
2               35.00000             79.0         6000.0
112.0
3               37.00000             82.0         6000.0
113.0
4               55.00000            148.0         3400.0
343.0
...                  ...              ...            ...
...
2054            70.00000            138.0         3750.0
```

```
330.0
2055            32.00000             55.0         5500.0
75.0
2056            45.00000             70.0         6250.0
102.0
2057            65.00000            188.0         4000.0
400.0
2058            52.00221             70.0         3600.0
195.0

      Max Torque RPM  Car exploitet in years
0             4500.0                       5
1             2000.0                       8
2             7619.0                      11
3             4200.0                       3
4             1400.0                       4
...              ...                     ...
2054          1600.0                       6
2055          4000.0                       8
2056          4000.0                       9
2057          1750.0                       4
2058          1400.0                       5

[2059 rows x 19 columns]
```

```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Iteriere durch alle Spalten
for i in X.columns:
    # Prüfen, ob der Datentyp der Spalte "object" ist (kategorisch)
    if X[i].dtype == 'object':
        # Wende One-Hot-Encoding auf die kategorische Spalte an
        X = pd.get_dummies(X, columns=[i])

# Wenn du es nur auf bestimmte Spalten anwenden willst, kannst du auch
explizit definieren:
# columns_to_encode = ['Fuel Type', 'Transmission', 'Location']
# X = pd.get_dummies(X, columns=columns_to_encode)

# One-Hot-Encoding mit Drop der ersten Spalte
X = pd.get_dummies(X, drop_first=True)

y = df["Price"]
X = X.drop(columns=["Price"])

# Listen zur Speicherung der R²-Scores
r2_train_scores = {"LinearRegression": [], "LogisticRegression": [],
"Lasso": []}
r2_test_scores = {"LinearRegression": [], "LogisticRegression": [],
"Lasso": []}
```

```python
# Wiederholen der Schritte 20 Mal
for i in range(20):


    # Teilen der Daten in Trainings- und Testdatensatz
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=i)

    # Standardisieren der Daten
    scaler = MinMaxScaler()

    # Fit the scaler on Xtrain and transform Xtrain
    Xtrain_scaled = scaler.fit_transform(X_train)

    # Transform Xtest using the same scaler (no fitting)
    Xtest_scaled = scaler.transform(X_test)
    # Initialisierung der Modelle
    models = {
        "LinearRegression": LinearRegression(),
        "LogisticRegression": LogisticRegression(max_iter=1000),
        "Lasso": Lasso(alpha=0.1)
    }

    # Trainieren der Modelle und Vorhersage auf dem Testdatensatz
    for model_name, model in models.items():
        model.fit(Xtrain_scaled, y_train)

        # Vorhersage auf dem Testdatensatz
        ytest_pred = model.predict(Xtest_scaled)

        # Berechnung des R²-Score für den Testdatensatz
        r2_test = r2_score(y_test, ytest_pred)
        r2_test_scores[model_name].append(r2_test)

        # Berechnung des R²-Score für den Trainingsdatensatz
        ytrain_pred = model.predict(Xtrain_scaled)
        r2_train = r2_score(y_train, ytrain_pred)
        r2_train_scores[model_name].append(r2_train)

# Berechnung der Durchschnittswerte und Standardabweichungen
mean_r2_train = {model_name: np.mean(scores) for model_name, scores in
r2_train_scores.items()}
std_r2_train = {model_name: np.std(scores) for model_name, scores in
r2_train_scores.items()}

mean_r2_test = {model_name: np.mean(scores) for model_name, scores in
r2_test_scores.items()}
std_r2_test = {model_name: np.std(scores) for model_name, scores in
r2_test_scores.items()}
```

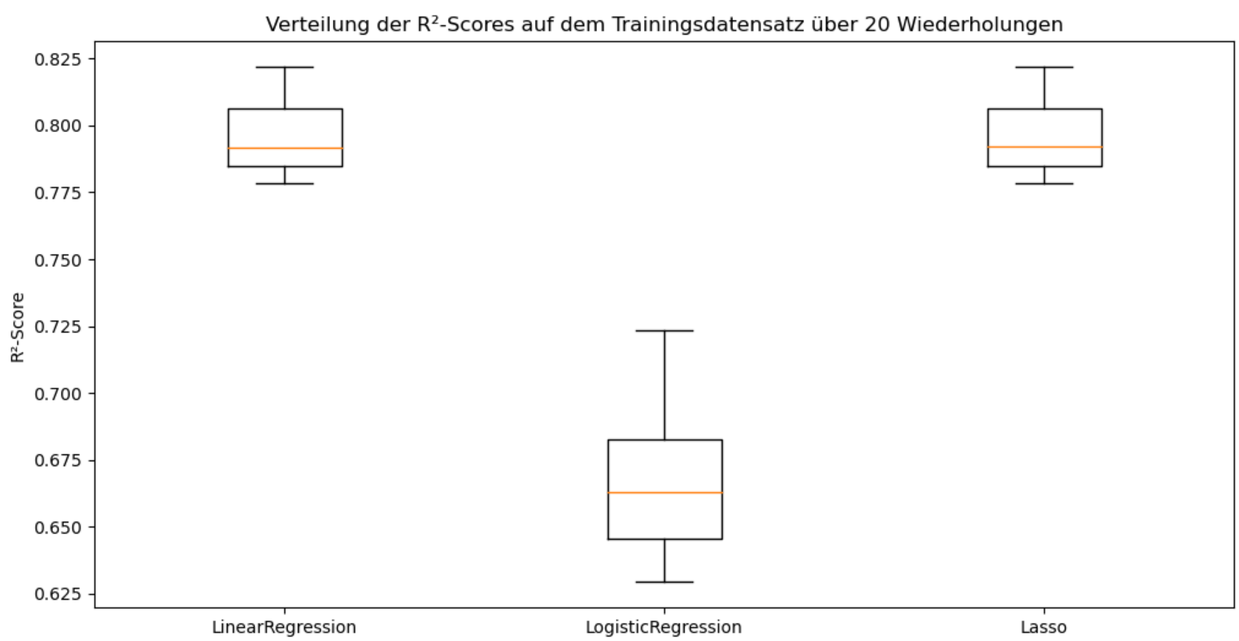
```python
# Ausgabe der Ergebnisse
for model_name in r2_train_scores.keys():
    print(f"Modell: {model_name}")
    print(f"Durchschnittlicher R²-Score auf dem Trainingsdatensatz:
{mean_r2_train[model_name]} ± {std_r2_train[model_name]}")
    print(f"Durchschnittlicher R²-Score auf dem Testdatensatz:
{mean_r2_test[model_name]} ± {std_r2_test[model_name]}")
    print()

# Erstellen eines Boxplots zur Visualisierung der Stabilität
plt.figure(figsize=(12, 6))
plt.boxplot([r2_train_scores["LinearRegression"],
r2_train_scores["LogisticRegression"], r2_train_scores["Lasso"]],
            labels=['LinearRegression', 'LogisticRegression',
'Lasso'])
plt.title('Verteilung der R²-Scores auf dem Trainingsdatensatz über 20
Wiederholungen')
plt.ylabel('R²-Score')
plt.show()

plt.figure(figsize=(12, 6))
plt.boxplot([r2_test_scores["LinearRegression"],
r2_test_scores["LogisticRegression"], r2_test_scores["Lasso"]],
            labels=['LinearRegression', 'LogisticRegression',
'Lasso'])
plt.title('Verteilung der R²-Scores auf dem Testdatensatz über 20
Wiederholungen')
plt.ylabel('R²-Score')
plt.show()
```

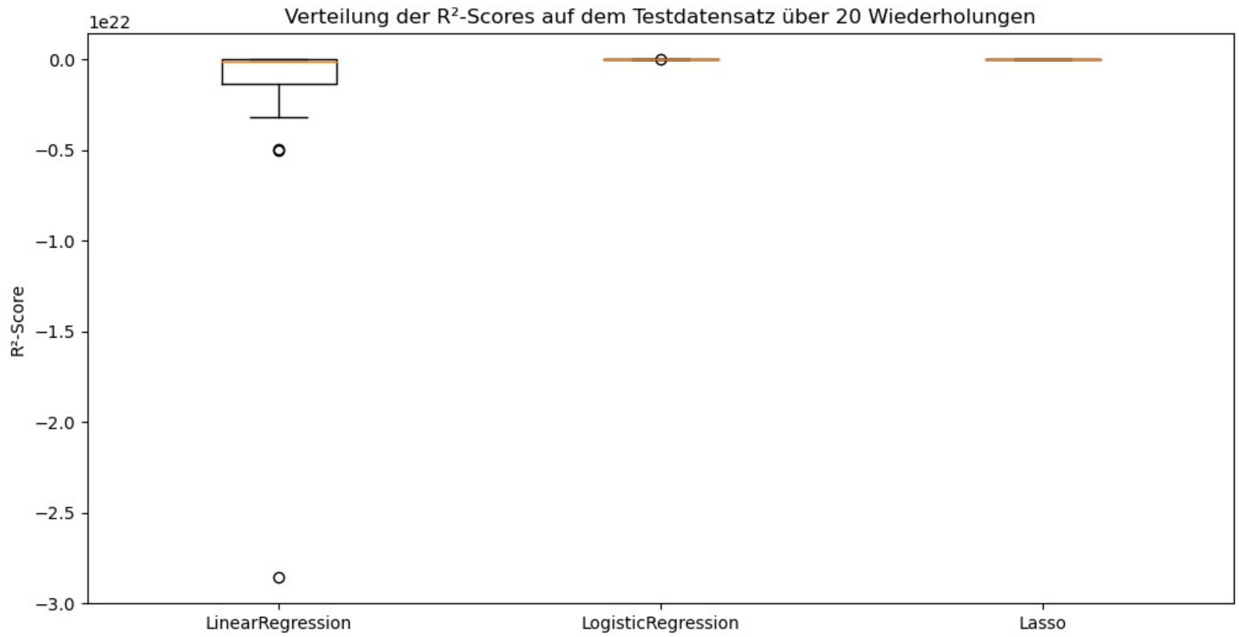
```
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 4.315e+14, tolerance: 9.234e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 3.721e+14, tolerance: 8.614e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 2.416e+14, tolerance: 1.042e+12
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
```

```
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 4.814e+14, tolerance: 1.069e+12
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 4.741e+14, tolerance: 1.022e+12
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 3.900e+14, tolerance: 8.636e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 4.673e+14, tolerance: 1.008e+12
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 2.965e+14, tolerance: 9.625e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 2.650e+14, tolerance: 1.056e+12
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 3.920e+14, tolerance: 9.447e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 3.429e+14, tolerance: 8.618e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
```

```
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 4.654e+14, tolerance: 1.058e+12
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 4.503e+14, tolerance: 9.715e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 3.724e+14, tolerance: 9.341e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 2.130e+14, tolerance: 8.770e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 2.420e+14, tolerance: 9.713e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 2.406e+14, tolerance: 1.015e+12
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 1.851e+14, tolerance: 9.274e+11
  model = cd_fast.enet_coordinate_descent(
C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 4.017e+14, tolerance: 1.002e+12
  model = cd_fast.enet_coordinate_descent(

Modell: LinearRegression
Durchschnittlicher R²-Score auf dem Trainingsdatensatz:
0.7945347920221131 ± 0.012530043874783674
```

```
Durchschnittlicher R²-Score auf dem Testdatensatz: -
2.34565075277085e+21 ± 6.213052585297884e+21

Modell: LogisticRegression
Durchschnittlicher R²-Score auf dem Trainingsdatensatz:
0.6678831825277923 ± 0.028084807635327627
Durchschnittlicher R²-Score auf dem Testdatensatz: 0.4158398914398142
± 0.12199571232250417

Modell: Lasso
Durchschnittlicher R²-Score auf dem Trainingsdatensatz:
0.7945681393707914 ± 0.01252280536709652
Durchschnittlicher R²-Score auf dem Testdatensatz: 0.6983893749203387
± 0.06949786453769216


C:\Users\mrazi\anaconda3\Lib\site-packages\sklearn\linear_model\
_coordinate_descent.py:628: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation.
Duality gap: 4.067e+14, tolerance: 9.992e+11
  model = cd_fast.enet_coordinate_descent(
```






```python
# Train your model with the new feature set (X_no_price)
model = RandomForestRegressor()
model.fit(X_no_price, y)  # y is the target variable (Price)
```

```
RandomForestRegressor()
```

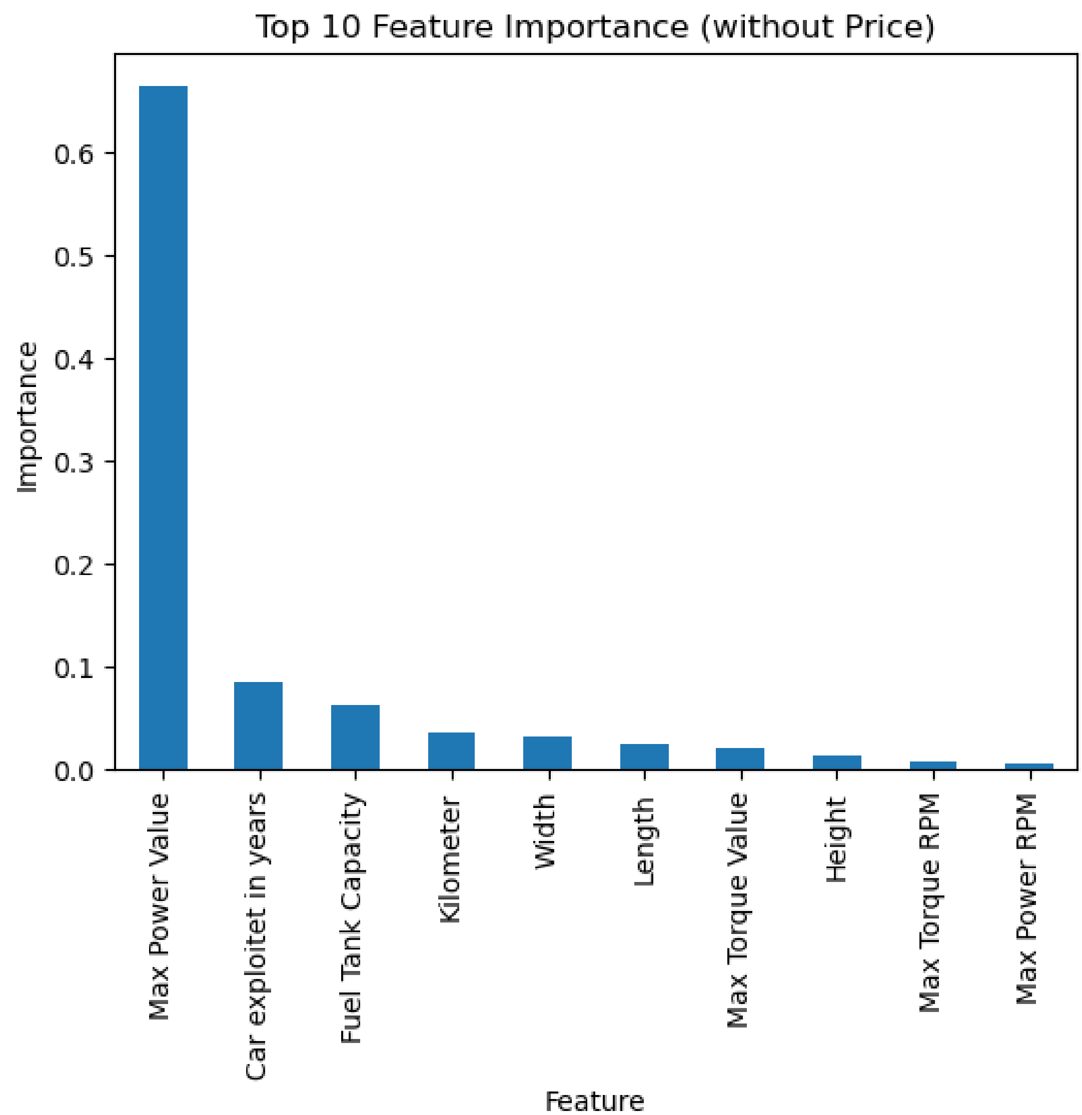
```python
# Get feature importances
importances = model.feature_importances_

# Create a DataFrame for features and their importances
importance_df = pd.DataFrame({'Feature': X_no_price.columns,
'Importance': importances})

# Sort the importance values in descending order
sorted_importance_df = importance_df.sort_values(by='Importance',
ascending=False)

# Display the sorted importance values
print(sorted_importance_df)
```

```
                   Feature    Importance
6          Max Power Value  6.624728e-01
10  Car exploitet in years  8.648842e-02
5       Fuel Tank Capacity  6.299924e-02
0                Kilometer  3.650972e-02
2                    Width  3.248926e-02
```

```
..                         ...           ...
16                   Make_Fiat  8.279298e-08
52  Fuel Type_Petrol + LPG  7.925522e-08
51  Fuel Type_Petrol + CNG  4.141852e-08
45     Fuel Type_CNG + CNG  2.471588e-08
66                  Color_Pink  2.250240e-08

[84 rows x 2 columns]
```

```
# Select top 10 features by importance
top_10_features = sorted_importance_df.head(10)

# Plot the top 10 features
plt.figure(figsize=(10, 6))
top_10_features.plot(kind='bar', x='Feature', y='Importance',
legend=False)
plt.title("Top 10 Feature Importance (without Price)")
plt.ylabel("Importance")
plt.xticks(rotation=90)  # Rotate feature names for better visibility
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```


Top 10 Feature Importance (without Price)
Importance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Max Power Value
Car exploitet in years
Fuel Tank Capacity
Kilometer
Width
Length
Max Torque Value
Height
Max Torque RPM
Max Power RPM
Feature