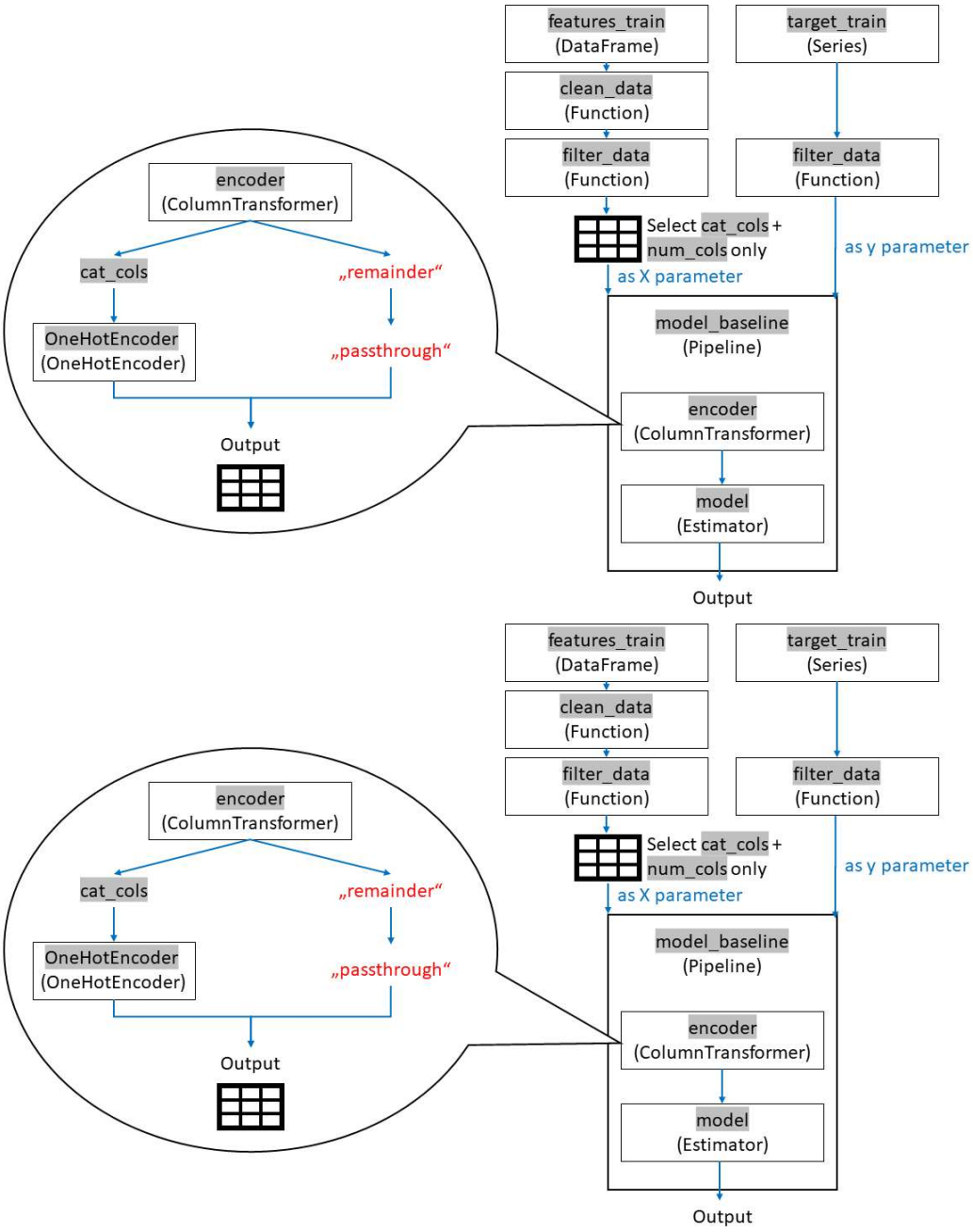
Build a simple Baselinemodel



Bevor du dich nun der Erzeugung und Auswahl neuer Features widmest, solltest du erst einmal sehen, von welchem Punkt du deine Bemühungen startest und ein erstes einfaches Modell ohne jegliche Hyperparameteroptimierung erzeugen. So kannst du hinterher immer wieder überprüfen, ob die folgenden Schritte in der Modelloptimierung eine Verbesserung bringen. Am besten erzeugst du eine Pipeline, die neben der Modellierung auch das Encoden nicht-numerischer Features übernimmt. Ich schlage folgendes Vorgehen vor:

1. Numerische Features (`num_cols`) und kategorische Features (`cat_cols`) definieren,
2. Algorithmus für das Baselinemodel aussuchen und als `model` instanziieren,
3. Pipeline (`model_baseline`) nach dem Schema unten erstellen,
4. Das gereinigte und ggf. gefilterte `features_train` sowie das entsprechende `target_train` zum *fitten* verwenden,
5. mit `model_baseline` Vorhersagen treffen auf das gereinigte `features_test` .

In [65]:
```
# define num_cols and cat_cols
# Categorical columns
cat_cols = list(features_train.select_dtypes(include='object').columns)
cat_cols
```

Out[65]:
```
['Auction',
 'Make',
 'Color',
 'Transmission',
 'WheelType',
 'Nationality',
 'Size',
 'TopThreeAmericanName',
 'VNST']
```

In [66]:
```
# Numeric columns
num_cols = list(features_train.select_dtypes(include=['number']).columns)
num_cols
```

Out[66]:
```
['VehYear',
 'VehicleAge',
 'WheelTypeID',
 'VehOdo',
 'MMRAcquisitionAuctionAveragePrice',
 'MMRAcquisitionAuctionCleanPrice',
 'MMRAcquisitionRetailAveragePrice',
 'MMRAcquisitonRetailCleanPrice',
 'MMRCurrentAuctionAveragePrice',
 'MMRCurrentAuctionCleanPrice',
 'MMRCurrentRetailAveragePrice',
 'MMRCurrentRetailCleanPrice',
 'BYRNO',
 'VNZIP1',
 'VehBCost',
 'IsOnlineSale',
 'WarrantyCost']
```

In [67]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn.impute import SimpleImputer
from sklearn.metrics import recall_score, precision_score, f1_score, accuracy_score

features_train_basic = features_train.drop('PurchDate', axis=1)
features_test_basic = features_test.drop('PurchDate', axis=1)
print(features_train_basic.columns)
print(features_test_basic.columns)
```

```
Index(['Auction', 'VehYear', 'VehicleAge', 'Make', 'Color', 'Transmission',
       'WheelTypeID', 'WheelType', 'VehOdo', 'Nationality', 'Size',
       'TopThreeAmericanName', 'MMRAcquisitionAuctionAveragePrice',
       'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice',
       'MMRAcquisitonRetailCleanPrice', 'MMRCurrentAuctionAveragePrice',
       'MMRCurrentAuctionCleanPrice', 'MMRCurrentRetailAveragePrice',
       'MMRCurrentRetailCleanPrice', 'BYRNO', 'VNZIP1', 'VNST', 'VehBCost',
       'IsOnlineSale', 'WarrantyCost'],
      dtype='object')
Index(['Auction', 'VehYear', 'VehicleAge', 'Make', 'Color', 'Transmission',
       'WheelTypeID', 'WheelType', 'VehOdo', 'Nationality', 'Size',
       'TopThreeAmericanName', 'MMRAcquisitionAuctionAveragePrice',
       'MMRAcquisitionAuctionCleanPrice', 'MMRAcquisitionRetailAveragePrice',
       'MMRAcquisitonRetailCleanPrice', 'MMRCurrentAuctionAveragePrice',
       'MMRCurrentAuctionCleanPrice', 'MMRCurrentRetailAveragePrice',
       'MMRCurrentRetailCleanPrice', 'BYRNO', 'VNZIP1', 'VNST', 'VehBCost',
       'IsOnlineSale', 'WarrantyCost'],
      dtype='object')
```

In [68]:
```python
import warnings
from sklearn.exceptions import ConvergenceWarning

warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

In [69]:
```python
#Basic Model with LogisticRegression

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)
    ],
    remainder='passthrough'
)

pipeline_basic = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(random_state=42))
])

# Pipeline'i eğit
pipeline_basic.fit(features_train_basic, target_train)

# Test verisi üzerinde tahmin yap
y_pred = pipeline_basic.predict(features_test_basic)

# Recall, Precision ve F1 Score'ları hesapla ve yazdır
recall_basic = recall_score(target_test, y_pred)
precision_basic = precision_score(target_test, y_pred)
f1_basic = f1_score(target_test, y_pred)
accuracy_score = accuracy_score(target_test, y_pred)

print("Recall Score:", recall_basic)
print("Precision Score:", precision_basic)
print("F1 Score:", f1_basic)
print("Accuracy Score:", accuracy_score)
```

```
Recall Score: 0.002364066193853428
Precision Score: 0.3333333333333333
F1 Score: 0.004694835680751174
Accuracy Score: 0.8707711063700091
```

In [70]:
```python
# Der Recall misst Ihre Fähigkeit, beschädigte Fahrzeuge zu erkennen.
# Es zeigt also, wie gut Sie tatsächlich beschädigte Fahrzeuge identifizieren könne
```

```
# Precision zeigt, wie zuverlässig Sie Fahrzeuge als Montagsauto kennzeichnen könne
# Ein niedriger Precision-Wert bedeutet,
        #dass das Model zwar viele True Positives identifiziert, aber auch viele Fa
```

In [71]:
```
# Mein Schwellenwert für accuracy_score---> Die Antwort auf die Frage, warum accura

'''from sklearn.dummy import DummyClassifier

# 'most_frequent' stratejisiyle DummyClassifier oluşturun
dummy_model = DummyClassifier(strategy='most_frequent')

# Modeli eğitin (aslında eğitime gerek yok, çünkü stratejiye göre sabit bir tahmin
dummy_model.fit(features_train_basic, target_train)

# Test seti üzerinde tahmin yapın
dummy_predictions = dummy_model.predict(features_test_basic)

# Doğruluk (accuracy) skoru
dummy_accuracy = accuracy_score(target_test, dummy_predictions)
print(dummy_accuracy)'''

# accuracy_score beim Dummymodell ist 0.87
```

Out[71]:
```
"from sklearn.dummy import DummyClassifier\n\n# 'most_frequent' stratejisiyle Dumm
yClassifier oluşturun\ndummy_model = DummyClassifier(strategy='most_frequent')\n\n
# Modeli eğitin (aslında eğitime gerek yok, çünkü stratejiye göre sabit bir tahmin
yapacak)\ndummy_model.fit(features_train_basic, target_train)\n\n# Test seti üzeri
nde tahmin yapın\ndummy_predictions = dummy_model.predict(features_test_basic)\n\n
# Doğruluk (accuracy) skoru\ndummy_accuracy = accuracy_score(target_test, dummy_pr
edictions)\nprint(dummy_accuracy)"
```

In [72]:
```
# Beim Basismodell wurde es festgestellt, dass der Accuracy_Score nicht als Modelgü
# weil die Daten eine unausgeglichene Verteilung aufwiesen.
# Wie hier zu sehen ist, liegt der schlechteste Genauigkeitswert für das Dummy-Mode
# Obwohl dies auf den ersten Blick wie ein sehr guter Wert erscheint,
# heißt das nicht, dass dies bei meinem Modell der Fall ist.
```

Glückwunsch: Du hast ein *baseline model* erstellt und kannst dich nun daran machen deine Metriken zu verbesssern.

Feature Engineering

Das Datenset bietet bereits einige Features. Viele davon sind aber textbasiert und somit für die meisten Modelle noch nicht nutzbar. Bedenke beim Erzeugen neuer Features, dass die Rechendauer und der Arbeitsspeicherbedarf deiner Modelle mit jedem weiteren Feature ansteigen. Die meisten Automodelle haben eine unübersichtlich große Anzahl an verschiedenen Modellen, Untermodellen und Ausstattungsvarianten. Die bisherige Erfahrung des Autohändlers zeigt aber, dass der Einfluss der Automarke auf die Wiederverkäuflichkeit deutlich überwiegt.

In [73]:
```
from sklearn.preprocessing import OneHotEncoder

def engineer_features(train, test):


    def map_season(month):
        if month in [12, 1, 2]:
            return 'Winter'
        elif month in [3, 4, 5]:
```

```
            return 'Spring'
        elif month in [6, 7, 8]:
            return 'Summer'
        elif month in [9, 10, 11]:
            return 'Autumn'

    # Date column
    train['Month'] = train['PurchDate'].dt.month
    train['Year'] = train['PurchDate'].dt.year
    train['Day'] = train['PurchDate'].dt.day
    train['Season'] = train['Month'].map(map_season)

    test['Month'] = test['PurchDate'].dt.month
    test['Year'] = test['PurchDate'].dt.year
    test['Day'] = test['PurchDate'].dt.day
    test['Season'] = test['Month'].map(map_season)

    train.drop(['PurchDate'], axis=1, inplace=True)
    test.drop(['PurchDate'], axis=1, inplace=True)

    train.drop(['Month'], axis=1, inplace=True)
    test.drop(['Month'], axis=1, inplace=True)

    # Zuverlässige IDs identifizieren:
    reliable_ID_index = [16369, 11210, 10510, 10420, 10410, 1055,
                         1082, 1141, 1086, 1125, 99750,
                         1151, 99761, 1031, 1051, 10315,
                         18881, 1081, 1235, 5546, 52598]

    # Neue Spalte "reliable_BYRNO" erstellen:

    # Die Filterkriterien sind die Gruppen, deren Durchschnitt der "IsBadBuy"-Spalt
    # Wenn eine Gruppe dieses Kriterium erfüllt, wird sie 'zuverlässig' genannt und
    # zuverlässig=1 nicht zuverlässig=0
    train['reliable_BYRNO'] = train['BYRNO'].isin(reliable_ID_index).astype(int)
    test['reliable_BYRNO'] = test['BYRNO'].isin(reliable_ID_index).astype(int)

    train = train.drop(['BYRNO', 'VNST'], axis=1)
    test = test.drop(['BYRNO', 'VNST'], axis=1)

    # OneHotEncoder kullanarak kategorik sütunları dönüştürme
    categorical_columns = train.select_dtypes(include=['object']).columns
    encoder = OneHotEncoder(drop='first', sparse=False)

    for col in categorical_columns:
        encoded_values_train = encoder.fit_transform(train[[col]])
        encoded_values_test = encoder.transform(test[[col]])
        # Her bir kategori için yeni sütun adları oluştur
        new_columns = [f"{col}_{cat}" for cat in encoder.get_feature_names([col])]
        # DataFrame'e yeni sütunları ekle
        train[new_columns] = pd.DataFrame(encoded_values_train, index=train.index)
        test[new_columns] = pd.DataFrame(encoded_values_test, index=test.index)
        # Orijinal kategorik sütunu düşür
        train.drop(col, axis=1, inplace=True)
        test.drop(col, axis=1, inplace=True)

    return train, test


'''
Schritte:
1. Gruppieren Sie den DataFrame nach „BYRNO“ und berechnen Sie den Mittelwert von „
2. Identifizieren Sie zuverlässige IDs, indem Sie Gruppen mit einem durchschnittlic
3. Erstellen Sie eine neue Spalte „zuverlässig“ im DataFrame und initialisieren Sie
```

```
4. Setzen Sie die Werte in der Spalte „zuverlässig“ auf 1 für Zeilen, die „BYRNO“-W
'''
```

Out[73]:
```
'\nSchritte:\n1. Gruppieren Sie den DataFrame nach „BYRNO“ und berechnen Sie den M
ittelwert von „IsBadBuy“ für jede Gruppe.\n2. Identifizieren Sie zuverlässige IDs,
indem Sie Gruppen mit einem durchschnittlichen „IsBadBuy“ von weniger als 0,10 aus
wählen.\n3. Erstellen Sie eine neue Spalte „zuverlässig“ im DataFrame und initiali
sieren Sie sie mit Nullen.\n4. Setzen Sie die Werte in der Spalte „zuverlässig“ au
f 1 für Zeilen, die „BYRNO“-Werten in zuverlässigen IDs entsprechen.\n'
```

Du hast nun sicherlich einige neue Features erzeugt oder gelöscht und damit die Trainingsdaten modifiziert. Um mit neuen Datensätzen das Gleiche tun zu können, solltest du deine durchgeführten Schritte wieder in eine Funktion kopieren. Nenne diese Funktion `engineer_features` .

In [74]:
```
#features_train, features_test = clean_data(features_train,features_test)
features_train, features_test = engineer_features(features_train,features_test)
```

In [75]:
```
target_train = target_train[features_train.index]
target_test = target_test[features_test.index]
print(features_train.shape, target_train.shape)
print(features_test.shape, target_test.shape)
```

```
(59058, 87) (59058,)
(6562, 87) (6562,)
```

In [76]:
```
features_train.to_csv('features_train_ohe.csv', index=False)
features_test.to_csv('features_test_ohe.csv', index=False)
```

In [77]:
```
target_train.to_csv('target_train_ohe.csv', index=False)
target_test.to_csv('target_test_ohe.csv', index=False)
```

Glückwunsch: Mit den neuen Features hast du eine gute Basis für ein erstes richtiges Modell!

In [78]:
```
features_train.isna().sum()
```

Out[78]:

VehYear	0
VehicleAge	0
WheelTypeID	0
VehOdo	0
MMRAcquisitionAuctionAveragePrice	0
MMRAcquisitionAuctionCleanPrice	0
MMRAcquisitionRetailAveragePrice	0
MMRAcquisitonRetailCleanPrice	0
MMRCurrentAuctionAveragePrice	0
MMRCurrentAuctionCleanPrice	0
MMRCurrentRetailAveragePrice	0
MMRCurrentRetailCleanPrice	0
VNZIP1	0
VehBCost	0
IsOnlineSale	0
WarrantyCost	0
Year	0
Day	0
reliable_BYRNO	0
Auction_Auction_MANHEIM	0
Auction_Auction_OTHER	0
Make_Make_BUICK	0
Make_Make_CHEVROLET	0
Make_Make_CHRYSLER	0
Make_Make_DODGE	0
Make_Make_FORD	0
Make_Make_GMC	0
Make_Make_HONDA	0
Make_Make_HYUNDAI	0
Make_Make_INFINITI	0
Make_Make_ISUZU	0
Make_Make_JEEP	0
Make_Make_KIA	0
Make_Make_LEXUS	0
Make_Make_LINCOLN	0
Make_Make_MAZDA	0
Make_Make_MERCURY	0
Make_Make_MINI	0
Make_Make_MITSUBISHI	0
Make_Make_NISSAN	0
Make_Make_OLDSMOBILE	0
Make_Make_PONTIAC	0
Make_Make_SATURN	0
Make_Make_SCION	0
Make_Make_SUBARU	0
Make_Make_SUZUKI	0
Make_Make_TOYOTA	0
Make_Make_VOLKSWAGEN	0
Make_Make_VOLVO	0
Color_Color_BLACK	0
Color_Color_BLUE	0
Color_Color_BROWN	0
Color_Color_GOLD	0
Color_Color_GREEN	0
Color_Color_GREY	0
Color_Color_MAROON	0
Color_Color_NOT AVAIL	0
Color_Color_ORANGE	0
Color_Color_OTHER	0
Color_Color_PURPLE	0
Color_Color_RED	0
Color_Color_SILVER	0
Color_Color_WHITE	0
Color_Color_YELLOW	0

```
Transmission_Transmission_MANUAL                    0
WheelType_WheelType_Covers                          0
WheelType_WheelType_Special                         0
Nationality_Nationality_OTHER                       0
Nationality_Nationality_OTHER ASIAN                 0
Nationality_Nationality_TOP LINE ASIAN              0
Size_Size_CROSSOVER                                 0
Size_Size_LARGE                                     0
Size_Size_LARGE SUV                                 0
Size_Size_LARGE TRUCK                               0
Size_Size_MEDIUM                                    0
Size_Size_MEDIUM SUV                                0
Size_Size_SMALL SUV                                 0
Size_Size_SMALL TRUCK                               0
Size_Size_SPECIALTY                                 0
Size_Size_SPORTS                                    0
Size_Size_VAN                                       0
TopThreeAmericanName_TopThreeAmericanName_FORD      0
TopThreeAmericanName_TopThreeAmericanName_GM        0
TopThreeAmericanName_TopThreeAmericanName_OTHER     0
Season_Season_Spring                                0
Season_Season_Summer                                0
Season_Season_Winter                                0
dtype: int64
```

Data Scaling

Modelle performen in der Regel besser, wenn du die Daten vorher skalierst. Am besten nutzt du hierfür die *transformer* aus `sklearn.preprocessing` . Du kannst Sie hier instanziieren, testen und dann später in deine Pipeline einbauen.

In [79]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

In [80]:
```
# Initialize StandardScaler
scaler = StandardScaler()

# Fit the scaler on features_train and transform both features_train and features_t
features_train_scaled = scaler.fit_transform(features_train)
features_test_scaled = scaler.transform(features_test)
```

Dimensionality Reduction

Hast du in deiner EDA stark korellierte Features entdeckt oder einfach insgesamt zu viele Features? Dann empfehle ich dir, zu prüfen, ob Dimensionsreduzierung z.B. durch eine `PCA` sinnvoll ist. Wir haben das Vorgehen hierzu in *Modul 1, Kapitel 4* behandelt.

In [81]:
```
from sklearn.decomposition import PCA

# Veri setini yükleyin (features_train olarak varsayalım)
# features_train.shape[1] ile toplam öznitelik sayısını alıyoruz
total_features = features_train_scaled.shape[1]

# PCA uygulayın
pca = PCA(n_components=total_features)
pca.fit(features_train_scaled)

# Açıklanan varyans oranlarını alın
explained_variance_ratios = pca.explained_variance_ratio_
```

```python
# Toplam varyans oranlarını bir DataFrame'e ekleyin
pca_results_df = pd.DataFrame({
    'Component': range(1, len(explained_variance_ratios) + 1),
    'Explained Variance Ratio': explained_variance_ratios,
    'Cumulative Explained Variance': explained_variance_ratios.cumsum()
})

# DataFrame'i yazdırın
print(pca_results_df)
```

	Component	Explained Variance Ratio	Cumulative Explained Variance
0	1	1.049726e-01	0.104973
1	2	4.579143e-02	0.150764
2	3	3.901709e-02	0.189781
3	4	3.054292e-02	0.220324
4	5	2.461727e-02	0.244941
5	6	2.413316e-02	0.269075
6	7	2.134100e-02	0.290416
7	8	1.955574e-02	0.309971
8	9	1.786882e-02	0.327840
9	10	1.759424e-02	0.345434
10	11	1.706979e-02	0.362504
11	12	1.629313e-02	0.378797
12	13	1.555898e-02	0.394356
13	14	1.538642e-02	0.409743
14	15	1.496032e-02	0.424703
15	16	1.447904e-02	0.439182
16	17	1.432505e-02	0.453507
17	18	1.427813e-02	0.467785
18	19	1.356070e-02	0.481346
19	20	1.334410e-02	0.494690
20	21	1.325098e-02	0.507941
21	22	1.288490e-02	0.520826
22	23	1.270413e-02	0.533530
23	24	1.260245e-02	0.546132
24	25	1.251469e-02	0.558647
25	26	1.247060e-02	0.571118
26	27	1.236231e-02	0.583480
27	28	1.230035e-02	0.595780
28	29	1.213975e-02	0.607920
29	30	1.203728e-02	0.619957
30	31	1.196563e-02	0.631923
31	32	1.192384e-02	0.643847
32	33	1.184986e-02	0.655697
33	34	1.175704e-02	0.667454
34	35	1.172457e-02	0.679178
35	36	1.170688e-02	0.690885
36	37	1.166331e-02	0.702549
37	38	1.158370e-02	0.714132
38	39	1.155372e-02	0.725686
39	40	1.152401e-02	0.737210
40	41	1.150427e-02	0.748714
41	42	1.148910e-02	0.760203
42	43	1.144407e-02	0.771647
43	44	1.140842e-02	0.783056
44	45	1.133351e-02	0.794389
45	46	1.131532e-02	0.805705
46	47	1.123019e-02	0.816935
47	48	1.117611e-02	0.828111
48	49	1.106462e-02	0.839176
49	50	1.103605e-02	0.850212
50	51	1.100738e-02	0.861219
51	52	1.081438e-02	0.872033
52	53	1.064764e-02	0.882681
53	54	1.052047e-02	0.893201
54	55	1.022990e-02	0.903431
55	56	9.936725e-03	0.913368
56	57	9.560618e-03	0.922929
57	58	9.085207e-03	0.932014
58	59	9.053066e-03	0.941067
59	60	8.185208e-03	0.949252
60	61	7.891935e-03	0.957144
61	62	7.490851e-03	0.964635
62	63	7.068304e-03	0.971703

63	64	5.282926e-03	0.976986
64	65	4.658677e-03	0.981645
65	66	3.960683e-03	0.985606
66	67	3.933929e-03	0.989540
67	68	2.879083e-03	0.992419
68	69	2.862241e-03	0.995281
69	70	2.235240e-03	0.997516
70	71	8.126437e-04	0.998329
71	72	7.632003e-04	0.999092
72	73	2.864972e-04	0.999378
73	74	2.678417e-04	0.999646
74	75	2.025200e-04	0.999849
75	76	6.809607e-05	0.999917
76	77	3.425063e-05	0.999951
77	78	2.451504e-05	0.999976
78	79	1.376318e-05	0.999989
79	80	7.677451e-06	0.999997
80	81	1.390503e-06	0.999998
81	82	1.187886e-06	1.000000
82	83	2.194294e-07	1.000000
83	84	1.262927e-07	1.000000
84	85	5.280861e-33	1.000000
85	86	2.824902e-33	1.000000
86	87	1.315719e-33	1.000000

Features Selection

Die Auswahl der richtigen Features ist eine gute Methode, um *Overiffting* zu vermeiden und dein Modell insgesamt zu verbessern.

In [ ]:

Train model

Nun ist es an der Zeit, ein Modell zu erzeugen und zu evaluieren. Beim Erzeugen des Modells kann es hilfreich sein, sich folgende Fragen zu stellen:

- Welches Modell soll verwendet werden? Du hast im Training folgende Klassifizierungs-Modelle kennengelernt:
  - `sklearn.neighbors.KNeighborsClassifier` (siehe *Modul 1, Kapitel 2, k Nearest Neighbors*)
  - `sklearn.linear_model.LogisticRegression` (siehe *Modul 2, Kapitel 2, Lineare Regression versus logistische Regression*)
  - `sklearn.tree.DecisionTreeClassifier` (siehe *Modul 2, Kapitel 3, Entscheidungsbaum*)
  - `sklearn.ensemble.RandomForestClassifier` (siehe *Modul 2, Kapitel 3, Mit Ensembling von Entscheidungsbäumen zu Entscheidungswäldern*)
  - `sklearn.svm.SVC` (siehe *Modul 2, Kapitel 4, Lineare Support Vector Machine*)
  - Künstliches neuronales Netz mit `keras` (siehe *Modul 2, Kapitel 5, keras – Ein Baukasten für neuronale Netze*)
- Wie gut sind die Vorhersagen auf ungesehenen Daten? Dabei können dir folgende Objekte weiterhelfen:

- `sklearn.pipeline.Pipeline` (siehe *Modul 1, Kapitel 1, Kreuzvalidierung mit Pipelines*)
- `sklearn.model_selection.KFold` (siehe *Modul 1, Kapitel 1, Modelle validieren mit Kreuzvalidierung*)
- `sklearn.model_selection.cross_val_score()` (siehe *Modul 1, Kapitel 1, Kreuzvalidierung mit Pipelines*)
- `sklearn.model_selection.validation_curve()` (siehe *Modul 1, Kapitel 1, Gittersuche*)
- `sklearn.model_selection.GridSearchCV` (siehe *Modul 1, Kapitel 1Gittersuche*)

- Welche Features sind wichtig für dein Modell und auf welche kannst du verzichten?

Ich empfehle dir, folgende Codezelle auszuführen, damit du die Datenkonvertierungs-Warnung nicht siehst:

In [82]:
```
from sklearn.exceptions import DataConversionWarning
import warnings
warnings.filterwarnings(action='ignore', category=DataConversionWarning)
```

Damit du weniger Tipparbeit hast, habe ich schon eine Import-Befehle vorgegeben. Du kannst sie gerne ergänzen:

In [83]:
```
#useful imports
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import GridSearchCV
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import StratifiedKFold
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score, recall_score, precision_score, accuracy_score
```

In [84]:
```
'''features_train = pd.read_csv('features_train_ohe.csv')
features_test = pd.read_csv('features_test_ohe.csv')
target_train = pd.read_csv('target_train_ohe.csv')
target_test = pd.read_csv('target_test_ohe.csv')'''
```

Out[84]:
```
"features_train = pd.read_csv('features_train_ohe.csv')\nfeatures_test = pd.read_c
sv('features_test_ohe.csv')\ntarget_train = pd.read_csv('target_train_ohe.csv')\nt
arget_test = pd.read_csv('target_test_ohe.csv')"
```

Baue zuerst ein einfaches Modell und evaluiere es, um einen Vergleichswert zu haben.

In [85]:
```
#Logistic Regression with Standardscaler

log = LogisticRegression(class_weight= 'balanced')

log.fit(features_train_scaled, target_train)

#predict
y_pred_log = log.predict(features_test_scaled)

#evaluate
recall_log_basic = recall_score(target_test, y_pred_log)
```

```
precision_log_basic = precision_score(target_test, y_pred_log)
f1_log_basic = f1_score(target_test, y_pred_log)

print("LogisticRegression\n")
print('recall: ', recall_log_basic, '\nprecision: ',precision_log_basic, '\nf1_scor
```

```
LogisticRegression

recall:  0.6382978723404256
precision:  0.2082529888160432
f1_score:  0.31404478045943596
```

In [86]:

```
# Finden der besten Parameter mit der Gittersuche für die logistische Regression

# parameters
'''param_log = {
    'C': [60, 65, 70, 75,80],
    'max_iter': [100, 250, 500],
    'solver': ['liblinear']
}

log_grid = LogisticRegression(class_weight= 'balanced')

# GridSearchCV
grid_search_log= GridSearchCV(log_grid, param_grid=param_log, scoring='f1', cv=5)

# fitting
grid_search_log.fit(features_train_scaled, target_train)

# best parameters and best model
best_params_log = grid_search_log.best_params_
best_model_log = grid_search_log.best_estimator_

print("Best parameters :", best_params_log)
print("Best Model :", best_model_log)

# predict
y_pred_log = best_model_log.predict(features_test_scaled)

# evaluate
recall_best_log = recall_score(target_test, y_pred_log)
precision_best_log = precision_score(target_test, y_pred_log)
f1_best_log = f1_score(target_test, y_pred_log)

print("Best LogisticRegression with Gridsearch\n-------------")
print('recall: ', recall_best_log, '\nprecision: ', precision_best_log, '\nf1_score
```

Out[86]:

```
'param_log = {\n    \'C\': [60, 65, 70, 75,80],\n    \'max_iter\': [100, 250, 50
0],\n    \'solver\': [\'liblinear\']\n}\n\nlog_grid = LogisticRegression(class_wei
ght= \'balanced\')\n\n# GridSearchCV\ngrid_search_log= GridSearchCV(log_grid, para
m_grid=param_log, scoring=\'f1\', cv=5)\n\n# fitting\ngrid_search_log.fit(features
_train_scaled, target_train)\n\n# best parameters and best model\nbest_params_log
= grid_search_log.best_params_\nbest_model_log = grid_search_log.best_estimator_\n
\nprint("Best parameters :", best_params_log)\nprint("Best Model :", best_model_lo
g)\n\n# predict\ny_pred_log = best_model_log.predict(features_test_scaled)\n\n# ev
aluate\nrecall_best_log = recall_score(target_test, y_pred_log)\nprecision_best_lo
g = precision_score(target_test, y_pred_log)\nf1_best_log = f1_score(target_test,
y_pred_log)\n\nprint("Best LogisticRegression with Gridsearch\n-------------")\npr
int(\'recall: \', recall_best_log, \'\nprecision: \', precision_best_log, \'\nf1_s
core: \', f1_best_log)'
```

In [87]:

```
'''Best parameters : {'C': 60, 'max_iter': 100, 'solver': 'liblinear'}
Best Model : LogisticRegression(C=60, class_weight='balanced', solver='liblinear')
Best LogisticRegression with Gridsearch
-------------
```

```
recall:  0.6382978723404256
precision:  0.20809248554913296
f1_score:  0.31386224934612034 '''
```

Out[87]:
```
"Best parameters : {'C': 60, 'max_iter': 100, 'solver': 'liblinear'}\nBest Model :
LogisticRegression(C=60, class_weight='balanced', solver='liblinear')\nBest Logist
icRegression with Gridsearch\n-------------\nrecall:  0.6382978723404256 \nprecisi
on:  0.20809248554913296 \nf1_score:  0.31386224934612034 "
```

In [88]:
```
# Logistische Regression mit besten Parametern

log_best = LogisticRegression(C=60, class_weight='balanced', solver='liblinear')

log_best.fit(features_train_scaled, target_train)

#predict
y_pred_log_best = log_best.predict(features_test_scaled)

#evaluate
recall_log_best = recall_score(target_test, y_pred_log)
precision_log_best = precision_score(target_test, y_pred_log)
f1_log_best = f1_score(target_test, y_pred_log)

print("Logistische Regression mit besten Parametern\n")
print('recall: ', recall_log_best, '\nprecision: ',precision_log_best, '\nf1_score:
```

```
Logistische Regression mit besten Parametern

recall:  0.6382978723404256
precision:  0.2082529888160432
f1_score:  0.31404478045943596
```

In [89]:
```
# Logistische Regression mit besten Parametern und PCA

pipeline_log = Pipeline([
    ('pca', PCA(n_components=None)),
    ('log', LogisticRegression(C=60, class_weight='balanced', solver='liblinear'))
])

# Ich habe den Parameter n_components auf 55 gesetzt, um 90 Prozent der Daten darzu

pipeline_log.fit(features_train_scaled, target_train)

#predict
y_pred_log_pca = pipeline_log.predict(features_test_scaled)

#evaluate
recall_log_pca = recall_score(target_test, y_pred_log_pca)
precision_log_pca = precision_score(target_test, y_pred_log_pca)
f1_log_pca = f1_score(target_test, y_pred_log_pca)

print("Logistische Regression mit besten Parametern\n")
print('recall: ', recall_log_pca, '\nprecision: ',precision_log_pca, '\nf1_score: '
```

```
Logistische Regression mit besten Parametern

recall:  0.6382978723404256
precision:  0.20809248554913296
f1_score:  0.31386224934612034
```

In [90]:
```
# n_componenets
# 55 --> recall:  0.6394799054373522  precision:  0.20262172284644195  f1_score:  0
# None --> recall:  0.6382978723404256  precision:  0.20809248554913296  f1_score:
```

In [91]:
```
# Best LogisticRegression with cross-validation
from sklearn.model_selection import cross_val_score, KFold

log_cv = LogisticRegression(C=0.02, max_iter=100, class_weight='balanced', solver='

# cross-validation
cv_recall = cross_val_score(log_cv, features_train_scaled, target_train, cv=5, scor
cv_precision = cross_val_score(log_cv, features_train_scaled, target_train, cv=5, s
cv_f1 = cross_val_score(log_cv, features_train_scaled, target_train, cv=5, scoring=

#print("Recall Scores:", cv_recall)
print("Recall Scores Mean:", cv_recall.mean())

#print("Precision Scores:", cv_precision)
print("Precision Scores Mean:", cv_precision.mean())

#print("F1 Scores:", cv_f1)
print("F1 Scores Mean:", cv_f1.mean())

# fitting
log_cv.fit(features_train_scaled, target_train)

# Evaluierung des Modells am realen Testsatz
predictions = log_cv.predict(features_test_scaled)

# Recall, Precision ve F1 Score
test_recall = recall_score(target_test, predictions)
test_precision = precision_score(target_test, predictions)
test_f1 = f1_score(target_test, predictions)

print("\nLogistische Regression mit Kreuzvalidierung\n")
print("\nRecall Scores Mean:", test_recall)
print("Precision Scores Mean:", test_precision)
print("F1 Scores Mean:", test_f1)
```

```
Recall Scores Mean: 0.636674811519746
Precision Scores Mean: 0.1978717822622636
F1 Scores Mean: 0.3019002227927392

Logistische Regression mit Kreuzvalidierung


Recall Scores Mean: 0.640661938534279
Precision Scores Mean: 0.20926640926640927
F1 Scores Mean: 0.3154831199068685
```

In [92]:
```
#Best LogisticRegression with Cross Validation and PCA

# PCA uygulayalım
pca = PCA(n_components=55)
features_train_pca = pca.fit_transform(features_train_scaled)
features_test_pca = pca.transform(features_test_scaled)

# Lojistik Regresyon modeli oluşturalım
log_cv_pca = LogisticRegression(C=0.02, max_iter=100, class_weight='balanced', solv

# Cross-validation ile modeli değerlendirelim
cv_recall = cross_val_score(log_cv_pca, features_train_pca, target_train, cv=5, scc
cv_precision = cross_val_score(log_cv_pca, features_train_pca, target_train, cv=5,
cv_f1 = cross_val_score(log_cv_pca, features_train_pca, target_train, cv=5, scoring

print("Cross-Validation Recall Scores Mean:", cv_recall.mean())
print("Cross-Validation Precision Scores Mean:", cv_precision.mean())
print("Cross-Validation F1 Scores Mean:", cv_f1.mean())
```

```
# fitting
log_cv_pca.fit(features_train_pca, target_train)

# Evaluierung des Modells am realen Testsatz
predictions = log_cv_pca.predict(features_test_pca)

# Recall, Precision ve F1 Score
test_recall_pca = recall_score(target_test, predictions)
test_precision_pca = precision_score(target_test, predictions)
test_f1_pca = f1_score(target_test, predictions)

print("\nLogistische Regression mit Kreuzvalidierung und PCA\n")
print("Recall Scores Mean:", test_recall_pca)
print("Precision Scores Mean:", test_precision_pca)
print("F1 Scores Mean:", test_f1_pca)
```

```
Cross-Validation Recall Scores Mean: 0.6351603245030975
Cross-Validation Precision Scores Mean: 0.19168003589439853
Cross-Validation F1 Scores Mean: 0.2944751342744452

Logistische Regression mit Kreuzvalidierung und PCA

Recall Scores Mean: 0.6371158392434988
Precision Scores Mean: 0.20286036883703426
F1 Scores Mean: 0.3077362260919212
```

In [93]:
```
# Random Forest

# Model
rf_basic =  RandomForestClassifier(class_weight='balanced', random_state=42)
# fitting
rf_basic.fit(features_train_scaled, target_train)
#predict
y_pred_rf = rf_basic.predict(features_test_scaled)

#evaluate
recall_rf_basic = recall_score(target_test, y_pred_rf)
precision_rf_basic = precision_score(target_test, y_pred_rf,zero_division=1)
f1_rf_basic = f1_score(target_test, y_pred_rf)

print("RandomForestClassifier\n")
print('recall: ', recall_rf_basic, '\nprecision: ',precision_rf_basic, '\nf1_score:
```

```
RandomForestClassifier

recall:  0.02127659574468085
precision:  0.6428571428571429
f1_score:  0.04118993135011441
```

In [94]:
```
# Feature importance for Random Forest
feature_importance = rf_basic.feature_importances_

# column names
feature_names = features_train.columns

#Combine and sort feature importance values and names
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': featu
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascendir
feature_importance_df
```

Out[94]:

	Feature	Importance
13	VehBCost	0.063727
3	VehOdo	0.062279
4	MMRAcquisitionAuctionAveragePrice	0.058988
9	MMRCurrentAuctionCleanPrice	0.058716
5	MMRAcquisitionAuctionCleanPrice	0.056976
8	MMRCurrentAuctionAveragePrice	0.056914
11	MMRCurrentRetailCleanPrice	0.054921
10	MMRCurrentRetailAveragePrice	0.053845
6	MMRAcquisitionRetailAveragePrice	0.053494
7	MMRAcquisitonRetailCleanPrice	0.051900
12	VNZIP1	0.049216
17	Day	0.045648
15	WarrantyCost	0.044035
1	VehicleAge	0.026023
0	VehYear	0.023033
65	WheelType_WheelType_Covers	0.016460
2	WheelTypeID	0.016195
19	Auction_Auction_MANHEIM	0.012459
18	reliable_BYRNO	0.008587
84	Season_Season_Spring	0.008402
61	Color_Color_SILVER	0.008256
85	Season_Season_Summer	0.008139
86	Season_Season_Winter	0.007787
62	Color_Color_WHITE	0.007499
20	Auction_Auction_OTHER	0.007479
50	Color_Color_BLUE	0.006793
74	Size_Size_MEDIUM	0.006262
16	Year	0.006165
54	Color_Color_GREY	0.005950
82	TopThreeAmericanName_TopThreeAmericanName_GM	0.005894
49	Color_Color_BLACK	0.005841
60	Color_Color_RED	0.005535
22	Make_Make_CHEVROLET	0.005327
24	Make_Make_DODGE	0.005207
52	Color_Color_GOLD	0.004906
81	TopThreeAmericanName_TopThreeAmericanName_FORD	0.004590

	Feature	Importance
25	Make_Make_FORD	0.004494
23	Make_Make_CHRYSLER	0.004313
75	Size_Size_MEDIUM SUV	0.004085
71	Size_Size_LARGE	0.003980
53	Color_Color_GREEN	0.003900
64	Transmission_Transmission_MANUAL	0.003689
83	TopThreeAmericanName_TopThreeAmericanName_OTHER	0.003663
80	Size_Size_VAN	0.003252
68	Nationality_Nationality_OTHER ASIAN	0.003166
41	Make_Make_PONTIAC	0.002952
55	Color_Color_MAROON	0.002706
42	Make_Make_SATURN	0.002280
73	Size_Size_LARGE TRUCK	0.002137
14	IsOnlineSale	0.002047
76	Size_Size_SMALL SUV	0.001976
69	Nationality_Nationality_TOP LINE ASIAN	0.001937
28	Make_Make_HYUNDAI	0.001903
32	Make_Make_KIA	0.001836
39	Make_Make_NISSAN	0.001645
31	Make_Make_JEEP	0.001431
70	Size_Size_CROSSOVER	0.001414
77	Size_Size_SMALL TRUCK	0.001333
35	Make_Make_MAZDA	0.001138
36	Make_Make_MERCURY	0.001116
45	Make_Make_SUZUKI	0.001098
78	Size_Size_SPECIALTY	0.001018
38	Make_Make_MITSUBISHI	0.000994
21	Make_Make_BUICK	0.000992
72	Size_Size_LARGE SUV	0.000980
46	Make_Make_TOYOTA	0.000944
79	Size_Size_SPORTS	0.000906
66	WheelType_WheelType_Special	0.000833
26	Make_Make_GMC	0.000784
51	Color_Color_BROWN	0.000765
59	Color_Color_PURPLE	0.000741
57	Color_Color_ORANGE	0.000639

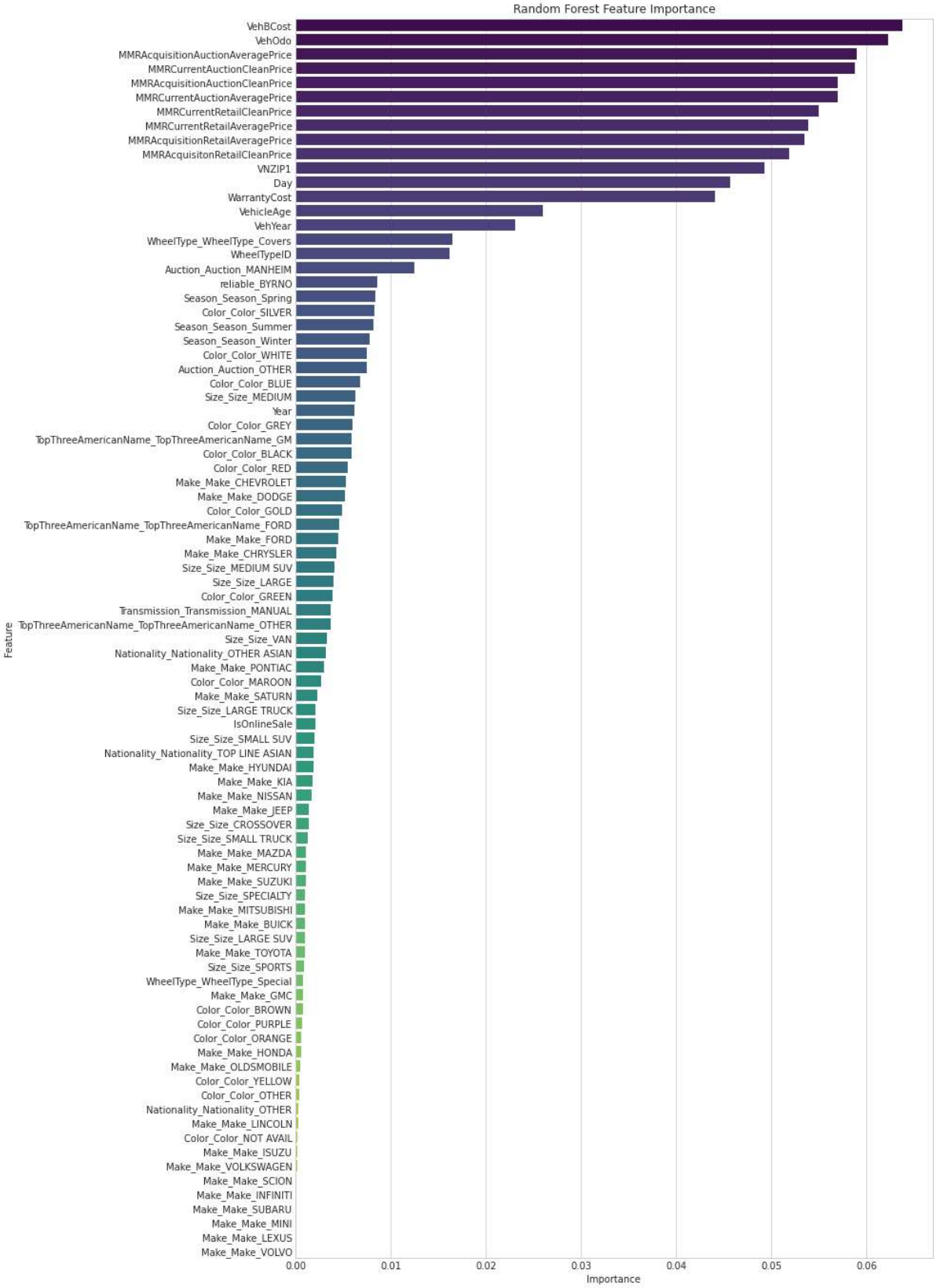
	Feature	Importance
27	Make_Make_HONDA	0.000609
40	Make_Make_OLDSMOBILE	0.000513
63	Color_Color_YELLOW	0.000381
58	Color_Color_OTHER	0.000374
67	Nationality_Nationality_OTHER	0.000310
34	Make_Make_LINCOLN	0.000260
56	Color_Color_NOT AVAIL	0.000230
30	Make_Make_ISUZU	0.000204
47	Make_Make_VOLKSWAGEN	0.000190
43	Make_Make_SCION	0.000140
29	Make_Make_INFINITI	0.000066
44	Make_Make_SUBARU	0.000059
37	Make_Make_MINI	0.000059
33	Make_Make_LEXUS	0.000038
48	Make_Make_VOLVO	0.000032

In [95]:
```
#Threshold value set for important columns

important_columns = feature_importance_df[feature_importance_df['Importance'] > 0.0
len(important_columns)
```

Out[95]:
```
39
```

In [96]:
```
# Visualisation
plt.figure(figsize=(12, 24))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df, palette='viric
plt.title('Random Forest Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Optimiere dann die Hyperparameter. Verbessert sich so die Vorhersage?

In [97]:

```
# Random Forest with important-columns

# Feature selection based on important_columns
features_train_rf = features_train[important_columns]
features_test_rf = features_test[important_columns]

scaler = StandardScaler()
features_train_rf_scaled= scaler.fit_transform(features_train_rf)
features_test_rf_scaled= scaler.transform(features_test_rf)
```

```
# Model
rf_imp_cols =  RandomForestClassifier(class_weight='balanced', random_state=42)
# fitting
rf_imp_cols.fit(features_train_rf_scaled, target_train)
#predict
y_pred_rf_imp = rf_imp_cols.predict(features_test_rf_scaled)

#evaluate
recall_rf_imp_cols = recall_score(target_test, y_pred_rf_imp)
precision_rf_imp_cols = precision_score(target_test, y_pred_rf_imp)
f1_rf_imp_cols = f1_score(target_test, y_pred_rf_imp)

print("RandomForestClassifier\n")
print('recall: ', recall_rf_imp_cols, '\nprecision: ',precision_rf_imp_cols, '\nf1_
```

```
RandomForestClassifier

recall:  0.022458628841607566
precision:  0.6551724137931034
f1_score:  0.043428571428571434
```

In [98]:

```
# tuning hyperparameters for Random Forest

'''# Create RandomForestClassifier
rf = RandomForestClassifier(class_weight = 'balanced', random_state=42)

# Feature selection based on important_columns
features_train_rf = features_train[important_columns]
features_test_rf = features_test[important_columns]

scaler = StandardScaler()
features_train_rf_scaled= scaler.fit_transform(features_train_rf)
features_test_rf_scaled= scaler.transform(features_test_rf)

# Parametre range
param_grid = {
    'n_estimators': [ 50, 100],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# GridSearchCV
grid_search = GridSearchCV(rf, param_grid=param_grid, scoring='f1', cv=5)

# fit
grid_search.fit(features_train_rf_scaled, target_train)

# best parameters and best model
best_params_grid = grid_search.best_params_
best_model_grid = grid_search.best_estimator_

# best parameters
print("Best parameters with Grid Search:", best_params_grid)

# Predict using the best model
y_pred_rf_grid = best_model_grid.predict(features_test_rf_scaled)

# Evaluate
recall_rf_grid = recall_score(target_test, y_pred_rf_grid)
precision_rf_grid = precision_score(target_test, y_pred_rf_grid)
f1_rf_grid = f1_score(target_test, y_pred_rf_grid)
```

```
print("Random Forest with Grid Search\n-------------")
print('recall: ', recall_rf_grid, '\nprecision: ', precision_rf_grid, '\nf1_score:
```

Out[98]:

```
'# Create RandomForestClassifier\nrf = RandomForestClassifier(class_weight = \'bal
anced\', random_state=42)\n\n# Feature selection based on important_columns\nfeatu
res_train_rf = features_train[important_columns]\nfeatures_test_rf = features_test
[important_columns]\n\nscaler = StandardScaler()\nfeatures_train_rf_scaled= scale
r.fit_transform(features_train_rf)\nfeatures_test_rf_scaled= scaler.transform(feat
ures_test_rf)\n\n# Parametre range\nparam_grid = {\n    \'n_estimators\': [ 50, 10
0],\n    \'max_depth\': [None, 5, 10],\n    \'min_samples_split\': [2, 5],\n
\'min_samples_leaf\': [1, 2]\n}\n\n# GridSearchCV\ngrid_search = GridSearchCV(rf,
param_grid=param_grid, scoring=\'f1\', cv=5)\n\n# fit\ngrid_search.fit(features_tr
ain_rf_scaled, target_train)\n\n# best parameters and best model\nbest_params_grid
= grid_search.best_params_\nbest_model_grid = grid_search.best_estimator_\n\n# bes
t parameters\nprint("Best parameters with Grid Search:", best_params_grid)\n\n# Pr
edict using the best model\ny_pred_rf_grid = best_model_grid.predict(features_test
_rf_scaled)\n\n# Evaluate\nrecall_rf_grid = recall_score(target_test, y_pred_rf_gr
id)\nprecision_rf_grid = precision_score(target_test, y_pred_rf_grid)\nf1_rf_grid
= f1_score(target_test, y_pred_rf_grid)\n\nprint("Random Forest with Grid Search\n
-------------")\nprint(\'recall: \', recall_rf_grid, \'\nprecision: \', precision_
rf_grid, \'\nf1_score: \', f1_rf_grid)'
```

In [99]:

```
'''Best parameters with Grid Search: {'max_depth': 10, 'min_samples_leaf': 1, 'min_
Random Forest with Grid Search
-------------
recall:  0.5508274231678487
precision:  0.23836317135549873
f1_score:  0.3327383077472332'''
```

Out[99]:

```
"Best parameters with Grid Search: {'max_depth': 10, 'min_samples_leaf': 1, 'min_s
amples_split': 2, 'n_estimators': 100}\nRandom Forest with Grid Search\n----------
---\nrecall:  0.5508274231678487 \nprecision:  0.23836317135549873 \nf1_score:  0.
3327383077472332"
```

In [100…

```
# Random Forest mit besten Parametern basierend auf wichtigen Spalten

# Create RandomForestClassifier
rf_best = RandomForestClassifier(class_weight='balanced',
                                 min_samples_leaf= 1,
                                 min_samples_split= 2,
                                 max_depth=10,
                                 n_estimators= 100,
                                 random_state=42)

# fitting
rf_best.fit(features_train_rf_scaled, target_train)
#predict
y_pred_rf_best = rf_best.predict(features_test_rf_scaled)

#evaluate
recall_rf_best = recall_score(target_test, y_pred_rf_best)
precision_rf_best = precision_score(target_test, y_pred_rf_best)
f1_rf_best = f1_score(target_test, y_pred_rf_best)

print("Best RandomForestClassifier based on important columns\n")
print('recall: ', recall_rf_best, '\nprecision: ',precision_rf_best, '\nf1_score: '
```

```
Best RandomForestClassifier based on important columns

recall:  0.5508274231678487
precision:  0.23836317135549873
f1_score:  0.3327383077472332
```

In [101…

```
# Best Random Forest mit GridSearch and PCA --> basierend auf wichtigen Spalten
```

```
#Pipeline
rf_best_pca = Pipeline([
    ('pca', PCA(n_components=None)),
    ('rf' , RandomForestClassifier(class_weight='balanced',
                                   max_depth= 10,
                                   min_samples_leaf= 1,
                                   min_samples_split= 2,
                                   n_estimators= 100,
                                   random_state=42))
    ])

#fitting
rf_best_pca.fit(features_train_rf_scaled, target_train)

#predict
y_pred_rf_best_pca = rf_best_pca.predict(features_test_rf_scaled)

#evaluate
recall_rf_best_pca = recall_score(target_test, y_pred_rf_best_pca)
precision_rf_best_pca = precision_score(target_test, y_pred_rf_best_pca)
f1_rf_best_pca = f1_score(target_test, y_pred_rf_best_pca)

print("Best RandomForestClassifier with PCA based on important columns\n----------
print('recall: ', recall_rf_best_pca, '\nprecision: ',precision_rf_best_pca, '\nf1_
```

```
Best RandomForestClassifier with PCA based on important columns
-------------
recall:  0.4787234042553192
precision:  0.2854122621564482
f1_score:  0.35761589403973515
```

In [102…

```
# Best Random Forest with GridSearch, PCA and Cross-Validation

# kFolds
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Modell
rf_cv_pca = RandomForestClassifier(class_weight='balanced',
                                   max_depth= 10,
                                   min_samples_leaf= 1,
                                   min_samples_split= 2,
                                   n_estimators= 100,
                                   random_state=42)


# PCA
pca = PCA(n_components=None)
features_train_pca = pca.fit_transform(features_train_rf_scaled)
features_test_pca = pca.transform(features_test_rf_scaled)


# Cross-validation ile modeli değerlendirelim
cv_recall = cross_val_score(rf_cv_pca, features_train_pca, target_train, cv=5, scor
cv_precision = cross_val_score(rf_cv_pca, features_train_pca, target_train, cv=5, s
cv_f1 = cross_val_score(rf_cv_pca, features_train_pca, target_train, cv=5, scoring=

print("Cross-Validation Recall Scores Mean:", cv_recall.mean())
print("Cross-Validation Precision Scores Mean:", cv_precision.mean())
print("Cross-Validation F1 Scores Mean:", cv_f1.mean())

# fitting
rf_cv_pca.fit(features_train_pca, target_train)

# Evaluierung des Modells am realen Testsatz
```

```
predictions = rf_cv_pca.predict(features_test_pca)

# Recall, Precision ve F1 Score
test_recall_pca = recall_score(target_test, predictions)
test_precision_pca = precision_score(target_test, predictions)
test_f1_pca = f1_score(target_test, predictions)

print("\nLogistische Regression mit Kreuzvalidierung und PCA\n")
print("Recall Scores Mean:", test_recall_pca)
print("Precision Scores Mean:", test_precision_pca)
print("F1 Scores Mean:", test_f1_pca)
```

```
Cross-Validation Recall Scores Mean: 0.43895850301777256
Cross-Validation Precision Scores Mean: 0.27705053036556343
Cross-Validation F1 Scores Mean: 0.33963733909135274

Logistische Regression mit Kreuzvalidierung und PCA

Recall Scores Mean: 0.4787234042553192
Precision Scores Mean: 0.2854122621564482
F1 Scores Mean: 0.35761589403973515
```

In [103…

```
#Random Forest with best parameters - feature_importance - Cross Validation with St

# RandomForestClassifier
rf_best = RandomForestClassifier(class_weight='balanced',
                                 min_samples_leaf= 1,
                                 min_samples_split= 2,
                                 max_depth=10,
                                 n_estimators= 100,
                                 random_state=42)


# StratifiedKFold ile manuel olarak cross-validation yapma
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Modelin en iyi parametrelerle fit edilmesi
f1_scores = []
precision_scores = []
recall_scores = []

for train_index, val_index in cv.split(features_train_rf, target_train):
    X_train_cv, X_val_cv = features_train_rf.iloc[train_index].values, features_tra
    y_train_cv, y_val_cv = target_train.iloc[train_index], target_train.iloc[val_ir

    # Modeli fit etme
    rf_best.fit(X_train_cv, y_train_cv)

    # Tahminler ve metrikler
    y_pred_cv = rf_best.predict(X_val_cv)
    f1_rf_cv = f1_score(y_val_cv, y_pred_cv)
    precision_rf_cv = precision_score(y_val_cv, y_pred_cv)
    recall_rf_cv = recall_score(y_val_cv, y_pred_cv)
    ac_cv = accuracy_score(y_val_cv, y_pred_cv)

    f1_scores.append(f1_rf_cv)
    precision_scores.append(precision_rf_cv)
    recall_scores.append(recall_rf_cv)

print("F1 Scores:", f1_scores)
print("Precision Scores:", precision_scores)
print("Recall Scores:", recall_scores)

# mean
```

```
mean_f1_rf = sum(f1_scores) / len(f1_scores)
mean_precision_rf = sum(precision_scores) / len(precision_scores)
mean_recall_rf = sum(recall_scores) / len(recall_scores)


print("Mean Precision:", mean_precision_rf)
print("Mean Recall:", mean_recall_rf)
print("Mean F1 Score:", mean_f1_rf)
```

```
F1 Scores: [0.32551194539249145, 0.3135112418575331, 0.3418693982074263, 0.3187782
318778232, 0.33347511697150145]
Precision Scores: [0.2357849196538937, 0.2255820985787723, 0.24768089053803338, 0.
23170731707317074, 0.2411565672100892]
Recall Scores: [0.5254820936639119, 0.5137741046831956, 0.5516528925619835, 0.5106
822880771882, 0.5403170227429359]
Mean Precision: 0.23638235861079188
Mean Recall: 0.528381680345843
Mean F1 Score: 0.32662918686135506
```

In [104…

```
# Decision Tree with Data Scaling

from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectFromModel

# Model
dt_basic = DecisionTreeClassifier(class_weight = 'balanced' , random_state=42)

# fitting
dt_basic.fit(features_train_scaled, target_train)

# predict
y_pred_dt_basic = dt_basic.predict(features_test_scaled)

# evaluate
recall_dt_basic = recall_score(target_test, y_pred_dt_basic)
precision_dt_basic = precision_score(target_test, y_pred_dt_basic)
f1_dt_basic = f1_score(target_test, y_pred_dt_basic)

print("DecisionTreeClassifier\n-------------")
print('recall: ', recall_dt_basic, '\nprecision: ', precision_dt_basic, '\nf1_score
```

```
DecisionTreeClassifier
-------------
recall:  0.23049645390070922
precision:  0.23353293413173654
f1_score:  0.23200475907198098
```

In [105…

```
# Decision Tree mit Feature Importance

# Decision Tree
dt_model = DecisionTreeClassifier(class_weight = 'balanced' , random_state=42)

# fitting
dt_model.fit(features_train, target_train)

# Özellik önem sıralamasını alın
feature_importances = dt_model.feature_importances_

# Önemli özellikleri seçin
sfm = SelectFromModel(dt_model, threshold=0.005)  # Önem sıralamasına göre bir eşik
sfm.fit(features_train, target_train)

# Önemli sütunları seçin
important_columns_dt = features_train.columns[sfm.get_support()]
print(important_columns_dt)
```

```
# Önemli sütunlara sahip verileri seçin
features_train_dt = features_train[important_columns_dt]
features_test_dt = features_test[important_columns_dt]
```

```
Index(['VehYear', 'VehicleAge', 'WheelTypeID', 'VehOdo',
       'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice',
       'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitonRetailCleanPrice',
       'MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice',
       'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice', 'VNZIP1',
       'VehBCost', 'WarrantyCost', 'Day', 'reliable_BYRNO',
       'Auction_Auction_MANHEIM', 'Make_Make_DODGE', 'Color_Color_BLACK',
       'Color_Color_BLUE', 'Color_Color_GOLD', 'Color_Color_GREEN',
       'Color_Color_GREY', 'Color_Color_RED', 'Color_Color_SILVER',
       'Color_Color_WHITE', 'Size_Size_MEDIUM', 'Season_Season_Spring',
       'Season_Season_Summer'],
      dtype='object')
```

In [106…

```
# Model
dt_basic = DecisionTreeClassifier(class_weight='balanced', random_state=42)
# fitting
dt_basic.fit(features_train_dt, target_train)
# predict
y_pred_dt = dt_basic.predict(features_test_dt)

# evaluate
recall_dt = recall_score(target_test, y_pred_dt)
precision_dt = precision_score(target_test, y_pred_dt)
f1_dt = f1_score(target_test, y_pred_dt)

print("DecisionTreeClassifier\n-------------")
print('recall: ', recall_dt, '\nprecision: ', precision_dt, '\nf1_score: ', f1_dt)
```

```
DecisionTreeClassifier
-------------
recall:  0.22576832151300236
precision:  0.217787913340935
f1_score:  0.22170632617527566
```

In [107…

```
from sklearn.model_selection import GridSearchCV
# Best Decision Tree with Gridsearch and Feature Importance
'''
dt_grid = DecisionTreeClassifier(class_weight = 'balanced' , random_state=42)


# GridSearchCV için parametre aralıklarını belirleyin
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
}

# GridSearchCV objesini oluşturun
grid_search_dt = GridSearchCV(dt_grid, param_grid= params, scoring='f1', cv=5, n_jo

# Modeli eğitin
grid_search_dt.fit(features_train_scaled, target_train)

# En iyi parametreleri ve en iyi tahmini bulun
best_params_dt = grid_search_dt.best_params_
best_dt_model = grid_search_dt.best_estimator_
print(best_dt_model)
y_pred_dt_best_grid = best_dt_model.predict(features_test_scaled)

# Değerlendirme yapın
```

```
recall_dt_best_grid = recall_score(target_test, y_pred_dt_best_grid)
precision_dt_best_grid = precision_score(target_test, y_pred_dt_best_grid)
f1_dt_best_grid = f1_score(target_test, y_pred_dt_best_grid)

print("DecisionTreeClassifier with GridSearchCV\n-------------")
print("Best Parameters:", best_params_dt)
print('recall: ', recall_dt_best_grid, '\nprecision: ', precision_dt_best_grid, '\r
```

Out[107]:

```
'\ndt_grid = DecisionTreeClassifier(class_weight = \'balanced\' , random_state=42)
\n\n\n# GridSearchCV için parametre aralıklarını belirleyin\nparams = {\n    \'cri
terion\': [\'gini\', \'entropy\'],\n    \'max_depth\': [None, 5, 10, 20],\n    \'m
in_samples_split\': [2, 5],\n    \'min_samples_leaf\': [1, 2],\n}\n\n# GridSearchC
V objesini oluşturun\ngrid_search_dt = GridSearchCV(dt_grid, param_grid= params, s
coring=\'f1\', cv=5, n_jobs=-1)\n\n# Modeli eğitin\ngrid_search_dt.fit(features_tr
ain_scaled, target_train)\n\n# En iyi parametreleri ve en iyi tahmini bulun\nbest_
params_dt = grid_search_dt.best_params_\nbest_dt_model = grid_search_dt.best_estim
ator_\nprint(best_dt_model)\ny_pred_dt_best_grid = best_dt_model.predict(features_
test_scaled)\n\n# Değerlendirme yapın\nrecall_dt_best_grid = recall_score(target_t
est, y_pred_dt_best_grid)\nprecision_dt_best_grid = precision_score(target_test, y
_pred_dt_best_grid)\nf1_dt_best_grid = f1_score(target_test, y_pred_dt_best_grid)
\n\nprint("DecisionTreeClassifier with GridSearchCV\n-------------")\nprint("Best
Parameters:", best_params_dt)\nprint(\'recall: \', recall_dt_best_grid, \'\nprecis
ion: \', precision_dt_best_grid, \'\nf1_score: \', f1_dt_best_grid)'
```

In [108…

```
'''DecisionTreeClassifier(class_weight='balanced', max_depth=5, random_state=42)
DecisionTreeClassifier with GridSearchCV
-------------
Best Parameters: {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 1, 'min_
recall:  0.6477541371158393
precision:  0.2003656307129799
f1_score:  0.3060597598436191'''
```

Out[108]:

```
"DecisionTreeClassifier(class_weight='balanced', max_depth=5, random_state=42)\nDe
cisionTreeClassifier with GridSearchCV\n-------------\nBest Parameters: {'criterio
n': 'gini', 'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}\nrecal
l:  0.6477541371158393 \nprecision:  0.2003656307129799 \nf1_score:  0.30605975984
36191"
```

In [109…

```
# Decision Tree with best Parameters

# Model
dt_best = DecisionTreeClassifier(class_weight='balanced',
                                 criterion= 'gini',
                                 min_samples_leaf= 1,
                                 max_depth=5,
                                 min_samples_split= 2,
                                 random_state=42)
# fitting
dt_best.fit(features_train_scaled, target_train)
# predict
y_pred_dt_best = dt_best.predict(features_test_scaled)

# evaluate
recall_dt_best = recall_score(target_test, y_pred_dt_best)
precision_dt_best = precision_score(target_test, y_pred_dt_best)
f1_dt_best = f1_score(target_test, y_pred_dt_best)

print("DecisionTreeClassifier with best parameters\n-------------")
print('recall: ', recall_dt_best, '\nprecision: ', precision_dt_best, '\nf1_score:
```

```
DecisionTreeClassifier with best parameters
-------------
recall:  0.6477541371158393
precision:  0.2003656307129799
f1_score:  0.3060597598436191
```

In [110…

```
# Decision Tree with best parameters and PCA


#Pipeline
dt_best_pca = Pipeline([
    ('pca', PCA(n_components=None)),
    ('dt' , DecisionTreeClassifier(class_weight='balanced',
                                   criterion= 'gini',
                                   min_samples_leaf= 1,
                                   max_depth=5,
                                   min_samples_split= 2,
                                   random_state=42))
    ])

#fitting
dt_best_pca.fit(features_train_scaled, target_train)

#predict
y_pred_dt_best_pca = dt_best_pca.predict(features_test_scaled)

#evaluate
recall_dt_best_pca = recall_score(target_test, y_pred_dt_best_pca)
precision_dt_best_pca = precision_score(target_test, y_pred_dt_best_pca)
f1_dt_best_pca = f1_score(target_test, y_pred_dt_best_pca)

print("Best Decision Tree with PCA based on important columns\n-------------")
print('recall: ', recall_dt_best_pca, '\nprecision: ',precision_dt_best_pca, '\nf1_
```

```
Best Decision Tree with PCA based on important columns
-------------
recall:  0.5177304964539007
precision:  0.20956937799043063
f1_score:  0.2983651226158038
```

In [111…

```
# Decision Tree with best Parameters and Cross Validation

# kFolds
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Pipeline
dt_cv = DecisionTreeClassifier(class_weight='balanced',
                                   criterion= 'gini',
                                   min_samples_leaf= 1,
                                   max_depth=5,
                                   min_samples_split= 2,
                                   random_state=42)

# Cross-validation yapın
cv_scores_recall_dt = cross_val_score(dt_cv, features_train_scaled, target_train, s
cv_scores_precision_dt = cross_val_score(dt_cv, features_train_scaled, target_trair
cv_scores_f1_dt = cross_val_score(dt_cv, features_train_scaled, target_train, scori

# Ortalamaları alın
cv_mean_recall_dt_cv = cv_scores_recall_dt.mean()
cv_mean_precision_dt_cv = cv_scores_precision_dt.mean()
cv_mean_f1_dt_cv = cv_scores_f1_dt.mean()

print("DecisionTreeClassifier\n-------------")
print('Mean recall: ', cv_mean_recall_dt_cv)
print('Mean precision: ', cv_mean_precision_dt_cv)
print('Mean f1_score: ', cv_mean_f1_dt_cv)

# Modeli eğitin
dt_cv.fit(features_train_scaled, target_train)
```

```
# Test verileri üzerinde tahmin yapın
test_predictions_dt = dt_cv.predict(features_test_scaled)

# Değerlendirme yapın
recall_test_dt_cv = recall_score(target_test, test_predictions_dt)
precision_test_dt_cv = precision_score(target_test, test_predictions_dt)
f1_test_dt_cv = f1_score(target_test, test_predictions_dt)

print("\nDecisionTreeClassifier on Test Data\n-------------")
print('recall: ', recall_test_dt_cv)
print('precision: ', precision_test_dt_cv)
print('f1_score: ', f1_test_dt_cv)
```

```
DecisionTreeClassifier
-------------
Mean recall:  0.6257836823182819
Mean precision:  0.1950472983545679
Mean f1_score:  0.297240106833476

DecisionTreeClassifier on Test Data
-------------
recall:  0.6477541371158393
precision:  0.2003656307129799
f1_score:  0.3060597598436191
```

In [112…

```
'''DecisionTreeClassifier with GridSearchCV
-------------
Best Parameters: {'dt__class_weight': 'balanced', 'dt__max_depth': 5, 'dt__min_samp
recall:  0.648936170212766
precision:  0.20021881838074398
f1_score:  0.3060200668896321'''
```

Out[112]:

```
"DecisionTreeClassifier with GridSearchCV\n-------------\nBest Parameters: {'dt__c
lass_weight': 'balanced', 'dt__max_depth': 5, 'dt__min_samples_leaf': 1, 'dt__min_
samples_split': 2}\nrecall:  0.648936170212766 \nprecision:  0.20021881838074398
\nf1_score:  0.3060200668896321"
```

In [113…

```
# kFolds
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Pipeline
pipe_dt_pca = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.90)),
    ('dt', DecisionTreeClassifier(class_weight='balanced',
                                  criterion= 'gini',
                                  min_samples_leaf= 1,
                                  max_depth=5,
                                  min_samples_split= 2,
                                  random_state=42))
])

# Cross-validation yapın
cv_scores_recall_dt = cross_val_score(pipe_dt_pca, features_train, target_train, sc
cv_scores_precision_dt = cross_val_score(pipe_dt_pca, features_train, target_train,
cv_scores_f1_dt = cross_val_score(pipe_dt_pca, features_train, target_train, scorir

# Ortalamaları alın
cv_mean_recall_dt = cv_scores_recall_dt.mean()
cv_mean_precision_dt = cv_scores_precision_dt.mean()
cv_mean_f1_dt = cv_scores_f1_dt.mean()

print("DecisionTreeClassifier\n-------------")
print('Mean recall: ', cv_mean_recall_dt)
```

```
print('Mean precision: ', cv_mean_precision_dt)
print('Mean f1_score: ', cv_mean_f1_dt)

# Modeli eğitin
pipe_dt_pca.fit(features_train, target_train)

# Test verileri üzerinde tahmin yapın
test_predictions_dt = pipe_dt_pca.predict(features_test)

# Değerlendirme yapın
recall_test_dt_cv = recall_score(target_test, test_predictions_dt)
precision_test_dt_cv = precision_score(target_test, test_predictions_dt)
f1_test_dt_cv = f1_score(target_test, test_predictions_dt)

print("\nDecisionTreeClassifier on Test Data\n-------------")
print('recall: ', recall_test_dt_cv)
print('precision: ', precision_test_dt_cv)
print('f1_score: ', f1_test_dt_cv)
```

```
DecisionTreeClassifier
-------------
Mean recall:  0.6247690884250385
Mean precision:  0.1794308221353898
Mean f1_score:  0.2780245501059108

DecisionTreeClassifier on Test Data
-------------
recall:  0.6749408983451537
precision:  0.19039679893297765
f1_score:  0.29700910273081926
```

In [114…

```
# kFolds
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Pipeline
pipe_dt_pca = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=None)),
    ('dt', DecisionTreeClassifier(class_weight='balanced',
                                  criterion= 'gini',
                                  min_samples_leaf= 1,
                                  max_depth=5,
                                  min_samples_split= 2,
                                  random_state=42))
])

# Modelin en iyi parametrelerle fit edilmesi ve çapraz doğrulama
f1_scores_dt_pca = []
precision_scores_dt_pca = []
recall_scores_dt_pca = []


for train_index, val_index in kf.split(features_train, target_train):
    X_train_cv, X_val_cv = features_train.iloc[train_index].values, features_train.
    y_train_cv, y_val_cv = target_train.iloc[train_index], target_train.iloc[val_ir

    # Modeli fit etme
    pipe_dt_pca.fit(X_train_cv, y_train_cv)

    # Tahminler ve metrikler
    y_pred_cv_dt_pca = pipe_dt_pca.predict(X_val_cv)
    f1_dt_pca = f1_score(y_val_cv, y_pred_cv_dt_pca)
    precision_dt_pca = precision_score(y_val_cv, y_pred_cv_dt_pca)
    recall_dt_pca = recall_score(y_val_cv, y_pred_cv_dt_pca)
```

```
    f1_scores_dt_pca.append(f1_dt_pca)
    precision_scores_dt_pca.append(precision_dt_pca)
    recall_scores_dt_pca.append(recall_dt_pca)


print("DecisionTreeClassifier with PCA\n-------------")
print("F1 Scores:", f1_scores_dt_pca)
print("Precision Scores:", precision_scores_dt_pca)
print("Recall Scores:", recall_scores_dt_pca)


# mean
mean_f1_dt_pca = sum(f1_scores_dt_pca) / len(f1_scores_dt_pca)
mean_precision_dt_pca = sum(precision_scores_dt_pca) / len(precision_scores_dt_pca)
mean_recall_dt_pca = sum(recall_scores_dt_pca) / len(recall_scores_dt_pca)

print("Mean Precision:", mean_precision_dt_pca)
print("Mean Recall:", mean_recall_dt_pca)
print("Mean F1 Score:", mean_f1_dt_pca)
```

```
DecisionTreeClassifier with PCA
-------------
F1 Scores: [0.30215550423402615, 0.28445797395569083, 0.28659476117103233, 0.28856
361589190926, 0.3041982105987612]
Precision Scores: [0.21028663273506562, 0.18697198755002223, 0.1979776476849388,
0.18892165122156698, 0.20392156862745098]
Recall Scores: [0.5365686944634313, 0.5943462897526501, 0.5188284518828452, 0.6106
194690265486, 0.5985104942450914]
Mean Precision: 0.19761589756380893
Mean Recall: 0.5717746798741133
Mean F1 Score: 0.293194013170284
```

In [115…

```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# Initialize StandardScaler
scaler = StandardScaler()

# Fit the scaler on features_train and transform both features_train and features_t
features_train_scaled = scaler.fit_transform(features_train)
features_test_scaled = scaler.transform(features_test)

# Best Logistik Regresyon
log_best = LogisticRegression(C=60, class_weight='balanced', solver='liblinear')

# Best Decision Tree
dt_best = DecisionTreeClassifier(class_weight='balanced',
                                 criterion= 'gini',
                                 min_samples_leaf= 1,
                                 max_depth=5,
                                 min_samples_split= 2,
                                 random_state=42)

# Best Random Forest
rf_best =RandomForestClassifier(class_weight='balanced',
                                 min_samples_leaf= 1,
                                 min_samples_split= 2,
                                 max_depth=10,
                                 n_estimators= 100,
                                 random_state=42)
```

```
# Ensemble
ensemble_model = VotingClassifier(estimators=[
    ('logreg', log_best),
    ('decision_tree', dt_best),
    ('random_forest', rf_best)
], voting='soft')  # 'soft' kullanarak model tahminlerini birleştiriyoruz.

# fitting
ensemble_model.fit(features_train_scaled, target_train)

# predict
y_pred_ensemble = ensemble_model.predict(features_test_scaled)

# evaluate
recall_ensemble = recall_score(target_test, y_pred_ensemble)
precision_ensemble = precision_score(target_test, y_pred_ensemble)
f1_ensemble = f1_score(target_test, y_pred_ensemble)

print("Ensemble Model (Logistic Regression + Decision Tree + Random Forest with PCA
print('Recall: ', recall_ensemble, '\nPrecision: ', precision_ensemble, '\nF1 Score
```

```
Ensemble Model (Logistic Regression + Decision Tree + Random Forest with PCA)
-------------
Recall:  0.6170212765957447
Precision:  0.21472645002056767
F1 Score:  0.31858407079646023
```

In [116…

```
dict_log_baseline = {'Model': ['LogisticRegression Basic'],
                     'Recall': recall_basic,
                     'Precision': precision_basic,
                     'F1 Score': f1_basic}

dict_log_basic = {'Model': ['LogisticRegression Basic'],
                     'Recall': recall_log_basic,
                     'Precision': precision_log_basic,
                     'F1 Score': f1_log_basic}

dict_log_best= {'Model': ['Best LogisticRegression with Gridsearch'],
                     'Recall': recall_log_best,
                     'Precision': precision_log_best,
                     'F1 Score':f1_log_best}

dict_log_best_pca= {'Model': ['Best LogisticRegression with Gridsearch and PCA'],
                     'Recall': recall_log_pca,
                     'Precision': precision_log_pca,
                     'F1 Score':f1_log_pca}

dict_log_best_cv= {'Model': ['Best LogisticRegression with Gridsearch and Cross Val
                     'Recall': test_recall,
                     'Precision': test_precision,
                     'F1 Score':test_f1}

dict_log_best_cv_pca = {'Model': ['Best LogisticRegression with PCA and Cross Valic
                     'Recall': test_recall_pca,
                     'Precision': test_precision_pca,
                     'F1 Score':test_f1_pca}

dict_rf_basic = {'Model': ['Random Forest Basic(RF)'],
                     'Recall': recall_rf_basic,
                     'Precision': precision_rf_basic,
                     'F1 Score': f1_rf_basic}

dict_rf_imp_cols = {'Model': ['Random Forest Basic(RF) with important columns'],
                     'Recall': recall_rf_imp_cols,
```

```
                  'Precision': precision_rf_imp_cols,
                  'F1 Score': f1_rf_imp_cols}

dict_rf_grid = {'Model': ['Best RandomForest with GridSearch and feature importance
                  'Recall': recall_rf_best,
                  'Precision': precision_rf_best,
                  'F1 Score': f1_rf_best}

dict_rf_grid_pca = {'Model': ['Best RandomForest with Gridsearch and PCA'],
                  'Recall': recall_rf_best_pca,
                  'Precision': precision_rf_best_pca,
                  'F1 Score': f1_rf_best_pca}

dict_rf_grid_pca_cv = {'Model': ['Best RandomForest with Gridsearch, PCA and Cross
                  'Recall': test_recall_pca,
                  'Precision':test_precision_pca,
                  'F1 Score': test_f1_pca}

dict_rf_grid_cv = {'Model': ['Best RandomForest with Gridsearch, Feature Importance
                  'Recall': mean_recall_rf,
                  'Precision':mean_precision_rf,
                  'F1 Score': mean_f1_rf}

dict_dt_basic = {'Model': ['Decision Tree Basic'],
                  'Recall': recall_dt_basic,
                  'Precision': precision_dt_basic,
                  'F1 Score': f1_dt_basic}

dict_dt= {'Model': ['Decision Tree with GridSearch'],
                  'Recall': recall_dt,
                  'Precision': precision_dt,
                  'F1 Score': f1_dt}

dict_dt_best= {'Model': ['Decision Tree with GridSearch and Feature Importance'],
                  'Recall': recall_dt_best,
                  'Precision': precision_dt_best,
                  'F1 Score': f1_dt_best}

dict_dt_best_pca= {'Model': ['Decision Tree with GridSearch without Feature Importa
                  'Recall': recall_dt_best_pca,
                  'Precision': precision_dt_best_pca,
                  'F1 Score': f1_dt_best_pca}

dict_dt_best_cv = {'Model': ['Decision Tree with GridSearch and Cross Validation wi
                  'Recall': recall_test_dt_cv,
                  'Precision': precision_test_dt_cv,
                  'F1 Score': f1_test_dt_cv}

dict_dt_grid_cv= {'Model': ['Decision Tree with GridSearch without Feature Importar
                  'Recall': mean_recall_dt_pca,
                  'Precision': mean_precision_dt_pca,
                  'F1 Score': mean_f1_dt_pca}

ensemble_model= {'Model': ['Logistic Regression - Random Forest - Decision Tree'],
                  'Recall': recall_ensemble,
                  'Precision': precision_ensemble,
                  'F1 Score': f1_ensemble}

results = [dict_log_baseline,
           dict_log_basic,
           dict_log_best,
           dict_log_best_cv,
           dict_log_best_pca,
           dict_rf_basic,
```

```
            dict_rf_imp_cols,
            dict_rf_grid,
            dict_rf_grid_pca,
            dict_rf_grid_pca_cv,
            dict_rf_grid_cv,
            dict_dt_basic,
            dict_dt,
            dict_dt_best,
            dict_dt_best_pca,
            dict_dt_best_cv,
            dict_dt_grid_cv,
            ensemble_model
           ]

pd.set_option('display.max_colwidth', None)
df_results = pd.DataFrame(results)
df_results = df_results.round(3)
df_results
```

Out[116]:

	Model	Recall	Precision	F1 Score
0	[LogisticRegression Basic]	0.002	0.333	0.005
1	[LogisticRegression Basic]	0.638	0.208	0.314
2	[Best LogisticRegression with Gridsearch]	0.638	0.208	0.314
3	[Best LogisticRegression with Gridsearch and Cross Validation]	0.641	0.209	0.315
4	[Best LogisticRegression with Gridsearch and PCA]	0.638	0.208	0.314
5	[Random Forest Basic(RF)]	0.021	0.643	0.041
6	[Random Forest Basic(RF) with important columns]	0.022	0.655	0.043
7	[Best RandomForest with GridSearch and feature importance]	0.551	0.238	0.333
8	[Best RandomForest with Gridsearch and PCA]	0.479	0.285	0.358
9	[Best RandomForest with Gridsearch, PCA and Cross Validation]	0.479	0.285	0.358
10	[Best RandomForest with Gridsearch, Feature Importance and Cross Validation(StratifiedKFold)]	0.528	0.236	0.327
11	[Decision Tree Basic]	0.230	0.234	0.232
12	[Decision Tree with GridSearch]	0.226	0.218	0.222
13	[Decision Tree with GridSearch and Feature Importance]	0.648	0.200	0.306
14	[Decision Tree with GridSearch without Feature Importance]	0.518	0.210	0.298
15	[Decision Tree with GridSearch and Cross Validation without Feature Importance]	0.675	0.190	0.297
16	[Decision Tree with GridSearch without Feature Importance]	0.572	0.198	0.293
17	[Logistic Regression - Random Forest - Decision Tree]	0.617	0.215	0.319

In [ ]:

Model selection

Wähle das beste Modell aus. Entscheide dabei selbst, welche Metrik dir am wichtigsten ist. Mit `confusion_matrix()` aus `sklearn.metrics` kannst du genau sehen, wie viele