

```
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0
```

**Glückwunsch:** Du hast nun einen besseren Einblick in deine Daten! Dieser kann dir beim weiteren Vorgehen weiterhelfen.

## Train-Test-Split

In diesem Projekt brauchst du sowohl die Trainings- als auch die Test- und Zieldaten.

Die Testdaten haben wir nicht vorgegeben. Deshalb empfehlen wir dir an dieser Stelle, die Daten in ein Trainings- und Testset zu teilen und mit dem Trainingsset so zu arbeiten, als wären dies alle Daten, die du zur Verfügung hast. Wenn du dein Modell dann fertig gebaut hast, kannst du mit dem Testset simulieren, was passiert, wenn neue Daten in deine Datenpipeline hereinkommen, also beispielsweise neue Autos auf Auktionsplattformen angeboten werden.

Denke immer daran: **Always fit on Train Set only!** Das gilt insbesondere auch für die Datenbereinigung und das *Feature Engineering*. Im Idealfall fasst du dein Testset nur einmal an, und zwar, wenn du ein für dich optimales Modell gebaut und evaluiert hast und wissen möchtest, wie gut es auf ungewöhnlichen Daten performt. Stell dir einfach vor, du hättest das Testset gar nicht vorliegen.

Nutze `train_test_split` aus dem Submodul `sklearn.model_selection`, um die Daten in Test- und Trainingsset aufzuteilen. Übergebe die folgenden Parameter: `random_state=42` und `test_size=0.1`, damit du deine Vorhersagen später mit unserem Modell vergleichen kannst und eine erste Einschätzung erhältst. Speichere zusätzlich `features_test` als `features_test.csv` ab.

```
In [38]: #read in the dataframe again to remove previous changes
df = pd.read_csv("data_train.csv")

# perform train-test-split
from sklearn.model_selection import train_test_split

target = df.loc[:, 'IsBadBuy']
features = df.drop('IsBadBuy', axis=1)

features_train, features_test, target_train, target_test = train_test_split(features,
```

```
target,  
random_  
test_si
```

```
features_train_aim = features_train.copy()  
  
print(features_train.shape)  
print(target_train.shape)  
print(features_test.shape)  
print(target_test.shape)
```

```
(59058, 32)  
(59058,)  
(6562, 32)  
(6562,)
```

```
In [39]: # save features_test as 'features_test.csv'  
features_test.to_csv('features_test.csv', index=False)
```

**Glückwunsch:** Du hast jetzt einen Trainingsdatensatz. Diesen kannst du nutzen, um neue Features zu erzeugen und um die Bausteine deines Modells zu trainieren.

## Data Preparation

Die *Data Preparation* hat das Ziel, einen Weg zu finden, die Datensätze für dein Modell zu säubern (*Data Cleaning*) und in ein für dein Modell lesbares Format zu bringen (*Datatype Transformation*). Wenn diese Schritte vollzogen sind, kannst du dich daran machen, ein Möglichst repräsentatives Trainingsset auszuwählen (*Sampling*).

### Datatype Transformation

```
In [40]: df= features_train.copy()
```

```
In [41]: #Die 'PurchDate'-Spalte soll in ein Datumsformat umgewandelt werden.  
df.loc[:, 'PurchDate'] = pd.to_datetime(df.loc[:, 'PurchDate'], unit='s')  
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 59058 entries, 21430 to 15795
Data columns (total 32 columns):
 #   Column           Non-Null Count Dtype  
--- 
 0   PurchDate        59058 non-null  datetime64[ns]
 1   Auction          59058 non-null  object  
 2   VehYear          59058 non-null  int64   
 3   VehicleAge       59058 non-null  int64   
 4   Make             59058 non-null  object  
 5   Model            59058 non-null  object  
 6   Trim             57187 non-null  object  
 7   SubModel         59051 non-null  object  
 8   Color            59051 non-null  object  
 9   Transmission     59050 non-null  object  
 10  WheelTypeID      56454 non-null  float64 
 11  WheelType        56450 non-null  object  
 12  VehODO          59058 non-null  int64   
 13  Nationality      59054 non-null  object  
 14  Size              59054 non-null  object  
 15  TopThreeAmericanName 59054 non-null  object  
 16  MMRAcquisitionAuctionAveragePrice 59043 non-null  float64 
 17  MMRAcquisitionAuctionCleanPrice   59043 non-null  float64 
 18  MMRAcquisitionRetailAveragePrice  59043 non-null  float64 
 19  MMRAcquisitionRetailCleanPrice    59043 non-null  float64 
 20  MMRCurrentAuctionAveragePrice    58797 non-null  float64 
 21  MMRCurrentAuctionCleanPrice     58797 non-null  float64 
 22  MMRCurrentRetailAveragePrice    58797 non-null  float64 
 23  MMRCurrentRetailCleanPrice     58797 non-null  float64 
 24  PRIMEUNIT          2785 non-null  object  
 25  AUCGUART          2785 non-null  object  
 26  BYRNO              59058 non-null  int64   
 27  VNZIP1            59058 non-null  int64   
 28  VNST              59058 non-null  object  
 29  VehBCost          59007 non-null  float64 
 30  IsOnlineSale       59058 non-null  int64   
 31  WarrantyCost      59058 non-null  int64  
dtypes: datetime64[ns](1), float64(10), int64(7), object(14)
memory usage: 14.9+ MB

```

```

In [42]: #Transmission Spalte
print(df['Transmission'].unique())

#Textformatierung standardisieren (Manual-MANUAL)
df['Transmission'].replace('Manual', 'MANUAL', inplace=True)
print(df['Transmission'].unique())

['AUTO' 'MANUAL' nan 'Manual']
['AUTO' 'MANUAL' nan]

```

```
In [43]: df.drop(['PRIMEUNIT', 'AUCGUART'], axis=1, inplace=True)
```

## Data Imputation

Im Gegensatz zum letzten Projekt gibt es in den Daten relativ viele fehlende Werte. Dies trifft auch auf die Test- und Zieldaten zu. Bei den Test- und Zieldaten solltest du **jeden** Datenpunkt klassifizieren. Aus ihnen solltest du also keinen Datenpunkt aufgrund fehlender oder unwahrscheinlicher Werte entfernen. Stattdessen könntest du die fehlenden Werte ersetzen oder Features mit vielen fehlenden Werten entfernen.

**Tipp:** Du kannst nicht davon ausgehen, dass fehlende Werte in *features\_aim.csv* in den gleichen Spalten wie in *data\_train.csv* vorkommen. Du brauchst also Strategien für jede Spalte. Um die fehlenden Werte mehrerer Spalten gleichzeitig mit verschiedenen Werten zu füllen, kannst du der Methode `my_df.fillna()` ein `dict` überreichen, dessen `keys` die entsprechenden Spaltennamen sind. Die `values` sind die zugehörigen Werte.

```
In [44]: df.columns
```

```
Out[44]: Index(['PurchDate', 'Auction', 'VehYear', 'VehicleAge', 'Make', 'Model',
       'Trim', 'SubModel', 'Color', 'Transmission', 'WheelTypeID', 'WheelType',
       'VehOdo', 'Nationality', 'Size', 'TopThreeAmericanName',
       'MMRAcquisitionAuctionAveragePrice', 'MMRAcquisitionAuctionCleanPrice',
       'MMRAcquisitionRetailAveragePrice', 'MMRAcquisitionRetailCleanPrice',
       'MMRCurrentAuctionAveragePrice', 'MMRCurrentAuctionCleanPrice',
       'MMRCurrentRetailAveragePrice', 'MMRCurrentRetailCleanPrice', 'BYRNO',
       'VNZIP1', 'VNST', 'VehBCost', 'IsOnlineSale', 'WarrantyCost'],
      dtype='object')
```

```
In [45]: print(features_train.columns)
print(features_test.columns)
```

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

```
In [46]: from sklearn.impute import SimpleImputer
```

```
def fillna_train_test(train,test):

    train = train.copy()
    test = test.copy()

    numeric_columns = train.select_dtypes(include=['number'], exclude=['datetime'])
    #print('numeric columns: ', numeric_columns)
    categorical_columns = train.select_dtypes(include=['object']).columns
    #print('categorical columns: ', categorical_columns)

    # Impute categorical columns with the mode
    categorical_imputer = SimpleImputer(strategy='most_frequent')
    train.loc[:, categorical_columns] = categorical_imputer.fit_transform(train.loc[:, categorical_columns])

    # Impute numeric columns with the median
    numeric_imputer = SimpleImputer(strategy='median')
    train.loc[:, numeric_columns] = numeric_imputer.fit_transform(train.loc[:, numeric_columns])
```

```

# Testdaten
test.loc[:, categorical_columns] = categorical_imputer.transform(test.loc[:, categorical_columns])
test.loc[:, numeric_columns] = numeric_imputer.transform(test.loc[:, numeric_columns])

return train,test

```

In [ ]:

In [ ]:

In [ ]:

## Deal with outliers

- Gibt es Ausreißer im Trainingsset ? - Wie sollte man mit ihnen umgehen?

```

In [47]: numeric_columns = ['MMRAcquisitionAuctionAveragePrice',
                      'MMRAcquisitionAuctionCleanPrice',
                      'MMRAcquisitionRetailAveragePrice',
                      'MMRAcquisitionRetailCleanPrice',
                      'MMRCurrentAuctionAveragePrice',
                      'MMRCurrentAuctionCleanPrice',
                      'MMRCurrentRetailAveragePrice',
                      'MMRCurrentRetailCleanPrice',
                      'VehOdo',
                      'WarrantyCost']

```

```

In [48]: def outlier_detection_IQR(features, target):
    """
    Detect and remove outliers from a DataFrame using the Interquartile Range (IQR)

    Parameters:
    - features (pd.DataFrame): The input DataFrame containing numerical columns.
    - target (pd.Series or array-like): The array containing 'IsBadBuy' values for each row.

    Returns:
    - pd.DataFrame: A modified DataFrame with outliers removed.
    - pd.Series or array-like: The modified target array.

    The function iterates through numerical columns (excluding specified columns like 'X_')
    and identifies outliers based on the Interquartile Range (IQR) for each class (except 'X_').
    Outliers are defined as values falling outside the 1.5 times IQR range.

    Outliers are then printed for each column and class.

    The function modifies the original DataFrame and target array by removing rows
    """
    features, target = features.align(target, axis=0, copy=False)
    # Combine features and target into a single DataFrame
    df = pd.concat([features, target], axis=1)

    numeric_columns = set(df.select_dtypes(include='number').columns)
    excluded_columns = {'BYRNO', 'IsOnlineSale', 'WheelTypeID', 'VNZIP1', 'PurchDate'}
    X_columns = list(numeric_columns - excluded_columns)

    rows_to_remove = []

    for col in X_columns:
        # IQR calculation

```

```

Q1_0 = df.loc[df['IsBadBuy'] == 0, col].quantile(0.25)
Q3_0 = df.loc[df['IsBadBuy'] == 0, col].quantile(0.75)
IQR_0 = Q3_0 - Q1_0

Q1_1 = df.loc[df['IsBadBuy'] == 1, col].quantile(0.25)
Q3_1 = df.loc[df['IsBadBuy'] == 1, col].quantile(0.75)
IQR_1 = Q3_1 - Q1_1

# Lower and upper limit calculation
lower_limit_0 = Q1_0 - 1.5 * IQR_0
upper_limit_0 = Q3_0 + 1.5 * IQR_0

lower_limit_1 = Q1_1 - 1.5 * IQR_1
upper_limit_1 = Q3_1 + 1.5 * IQR_1

# Identify outliers
outliers_0 = df.index[(df['IsBadBuy'] == 0) & ((df[col] < lower_limit_0) | 
outliers_1 = df.index[(df['IsBadBuy'] == 1) & ((df[col] < lower_limit_1) | 

print(f"{col}: Class 0 - {len(outliers_0)} outliers, Class 1 - {len(outlier

# Collect rows to remove
rows_to_remove.extend(outliers_0)
rows_to_remove.extend(outliers_1)

# Remove duplicates from rows_to_remove
rows_to_remove = list(set(rows_to_remove))

# Removing outliers
df = df.drop(index=rows_to_remove)

# Separate features and target from the modified DataFrame
features = df.drop(columns=['IsBadBuy'])
target = df['IsBadBuy']

return features, target

```

In [49]: #RANSAC

```

def outlier_detection_RANSAC(data):

    numeric_columns = set(data.select_dtypes(include='number').columns)
    excluded_columns = {'VehBCost', 'BYRNO', 'IsOnlineSale', 'WheelTypeID', 'VNZIP1'}
    print(list(numeric_columns - excluded_columns))
    X_columns = list(numeric_columns - excluded_columns)
    y_column = 'VehBCost'

    for col in X_columns:
        d_distance = mad(data.loc[:, y_column]) * 3

        model_outlier = RANSACRegressor(residual_threshold=d_distance, random_state=42)
        X = data.loc[:, [col]]
        y = data.loc[:, y_column]
        model_outlier.fit(X=X, y=y)
        data = data.loc[model_outlier.inlier_mask_, :]

        print(f'{col}: {(~model_outlier.inlier_mask_).sum()} outliers removed')

    return data

```

In [50]: display(df.loc[df['VehOdo'].idxmax(), ['VehicleAge']])  
print(df['VehOdo'].max())

```
display(df.loc[df['VehOdo'].idxmin(), ['VehicleAge']])
print(df['VehOdo'].min())
```

```
VehicleAge      5
Name: 23240, dtype: object
114184
VehicleAge      6
Name: 17281, dtype: object
5368
```

```
In [51]: display(df.loc[df['VehBCost'].idxmax(), ['VehicleAge', 'VehOdo', 'Make']])
print(df['VehBCost'].max())
display(df.loc[df['VehBCost'].idxmin(), ['VehicleAge', 'VehOdo', 'Make']])
print(df['VehBCost'].min())
```

```
VehicleAge      2
VehOdo        10095
Make          LEXUS
Name: 56544, dtype: object
38785.0
VehicleAge      8
VehOdo        102641
Make          DODGE
Name: 1606, dtype: object
1.0
```

```
In [52]: df[df['VehBCost']<1000]
```

```
Out[52]:   PurchDate Auction VehYear VehicleAge  Make    Model Trim SubModel Color Transn
           1606  2009-01-19 OTHER     2001       8 DODGE  DAKOTA
                           PICKUP Bas EXT CAB 2.5L  RED
                           2WD 4C
```

```
In [53]: features_train.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 59058 entries, 21430 to 15795
Data columns (total 32 columns):
 #   Column           Non-Null Count Dtype  
--- 
 0   PurchDate        59058 non-null  int64   
 1   Auction          59058 non-null  object  
 2   VehYear          59058 non-null  int64   
 3   VehicleAge       59058 non-null  int64   
 4   Make             59058 non-null  object  
 5   Model            59058 non-null  object  
 6   Trim             57187 non-null  object  
 7   SubModel         59051 non-null  object  
 8   Color            59051 non-null  object  
 9   Transmission     59050 non-null  object  
 10  WheelTypeID      56454 non-null  float64 
 11  WheelType        56450 non-null  object  
 12  VehOdo           59058 non-null  int64   
 13  Nationality      59054 non-null  object  
 14  Size              59054 non-null  object  
 15  TopThreeAmericanName 59054 non-null  object  
 16  MMRAcquisitionAuctionAveragePrice 59043 non-null  float64 
 17  MMRAcquisitionAuctionCleanPrice   59043 non-null  float64 
 18  MMRAcquisitionRetailAveragePrice 59043 non-null  float64 
 19  MMRAcquisitionRetailCleanPrice   59043 non-null  float64 
 20  MMRCurrentAuctionAveragePrice   58797 non-null  float64 
 21  MMRCurrentAuctionCleanPrice    58797 non-null  float64 
 22  MMRCurrentRetailAveragePrice   58797 non-null  float64 
 23  MMRCurrentRetailCleanPrice    58797 non-null  float64 
 24  PRIMEUNIT          2785 non-null  object  
 25  AUCGUART          2785 non-null  object  
 26  BYRNO              59058 non-null  int64   
 27  VNZIP1            59058 non-null  int64   
 28  VNST              59058 non-null  object  
 29  VehBCost           59007 non-null  float64 
 30  IsOnlineSale        59058 non-null  int64   
 31  WarrantyCost        59058 non-null  int64 

dtypes: float64(10), int64(8), object(14)
memory usage: 16.9+ MB

```

**Achtung:** Wenn du dich dazu entschließt, Ausreißer zu löschen, dann setze dies nur im Trainingset um und füge diesen Schritt **nicht** deiner Pipeline oder Cleaning-Funktion hinzu. Andernfalls wirst du bei Projektabschluss womöglich nicht für jeden Wert in *features\_aim* eine Vorhersage erhalten können.

Um deine Bereinigungsschritte leicht mit den Test- und Zieldaten zu reproduzieren, solltest du eine Funktion definieren, welche die jeweiligen Schritte für dich ausführt. Nenne deine Funktion `clean_data`. Sie sollte den unbereinigten `pandas.DataFrame` als Argument erhalten und den gereinigten `pandas.DataFrame` ausgeben. Achte dabei darauf, dass keine Datenpunkte gelöscht werden, denn im Zieldatenset solltest du für jeden Wert eine Vorhersage machen.

```
In [54]: def remove_correlation(train, test):

    corr_columns = ['MMRAcquisitionAuctionAveragePrice',
                    'MMRAcquisitionAuctionCleanPrice',
                    'MMRAcquisitionRetailAveragePrice',
                    'MMRAcquisitionRetailCleanPrice',
                    'MMRCurrentAuctionAveragePrice',
                    'MMRCurrentAuctionCleanPrice',
```

```

'MMRCurrentRetailAveragePrice',
'MMRCurrentRetailCleanPrice',
'VehYear']

train = train.drop(corr_columns, axis=1)
test = test.drop(corr_columns, axis=1)

return train, test

```

In [55]:

```

def clean_data(train, test):

    #Nan-Werte in anderen Spalten mit Modus und Median füllen
    train, test = fillna_train_test(train,test)

    #Die 'PurchDate'-Spalte soll in ein Datumsformat umgewandelt werden.
    train.loc[:, 'PurchDate'] = pd.to_datetime(train.loc[:, 'PurchDate'], unit='s')
    test.loc[:, 'PurchDate'] = pd.to_datetime(test.loc[:, 'PurchDate'], unit='s')

    # zwei Spalte löschen (ca 95 % der Spalten bestehen aus NAN-Werten)
    train= train.drop(['PRIMEUNIT','AUCGUART'], axis=1)
    test= test.drop(['PRIMEUNIT','AUCGUART'], axis=1)

    # drei kategorische Spalten, die nicht umgewandelt werden können, Löschen
    train = train.drop(['SubModel','Model','Trim'], axis=1)
    test = test.drop(['SubModel','Model','Trim'], axis=1)

    #Textformatierung standardisieren (Manual-MANUAL)
    train.loc[:, 'Transmission'] = train.loc[:, 'Transmission'].replace('Manual', 'MANUAL')
    test.loc[:, 'Transmission'] = test.loc[:, 'Transmission'].replace('Manual', 'MANUAL')

    #doppelte Einträge Löschen
    train = train.drop_duplicates()
    #test = test.drop_duplicates()

    # korrelierte Spalten entfernen
    #train, test = remove_correlation(train,test)

    return train, test

```

In [56]:

```

pd.set_option('mode.chained_assignment', None)

features_train, features_test = clean_data(features_train,features_test)

target_train = target_train[features_train.index]
target_test = target_test[features_test.index]

print(features_train.shape, target_train.shape)
(59058, 27) (59058,)

```

In [57]:

```
features_train.isnull().sum()
```

```
Out[57]: PurchDate          0  
Auction            0  
VehYear            0  
VehicleAge          0  
Make               0  
Color               0  
Transmission        0  
WheelTypeID         0  
WheelType           0  
VehOdo              0  
Nationality          0  
Size                0  
TopThreeAmericanName 0  
MMRAcquisitionAuctionAveragePrice 0  
MMRAcquisitionAuctionCleanPrice    0  
MMRAcquisitionRetailAveragePrice   0  
MMRAcquisitionRetailCleanPrice     0  
MMRCurrentAuctionAveragePrice     0  
MMRCurrentAuctionCleanPrice       0  
MMRCurrentRetailAveragePrice      0  
MMRCurrentRetailCleanPrice        0  
BYRNO                0  
VNZIP1              0  
VNST                0  
VehBCost             0  
IsOnlineSale          0  
WarrantyCost          0  
dtype: int64
```

```
In [58]: features_test.isnull().sum()
```

```
Out[58]: PurchDate          0  
Auction            0  
VehYear            0  
VehicleAge          0  
Make               0  
Color               0  
Transmission        0  
WheelTypeID         0  
WheelType           0  
VehOdo              0  
Nationality          0  
Size                0  
TopThreeAmericanName 0  
MMRAcquisitionAuctionAveragePrice 0  
MMRAcquisitionAuctionCleanPrice    0  
MMRAcquisitionRetailAveragePrice   0  
MMRAcquisitionRetailCleanPrice     0  
MMRCurrentAuctionAveragePrice     0  
MMRCurrentAuctionCleanPrice       0  
MMRCurrentRetailAveragePrice      0  
MMRCurrentRetailCleanPrice        0  
BYRNO                0  
VNZIP1              0  
VNST                0  
VehBCost             0  
IsOnlineSale          0  
WarrantyCost          0  
dtype: int64
```

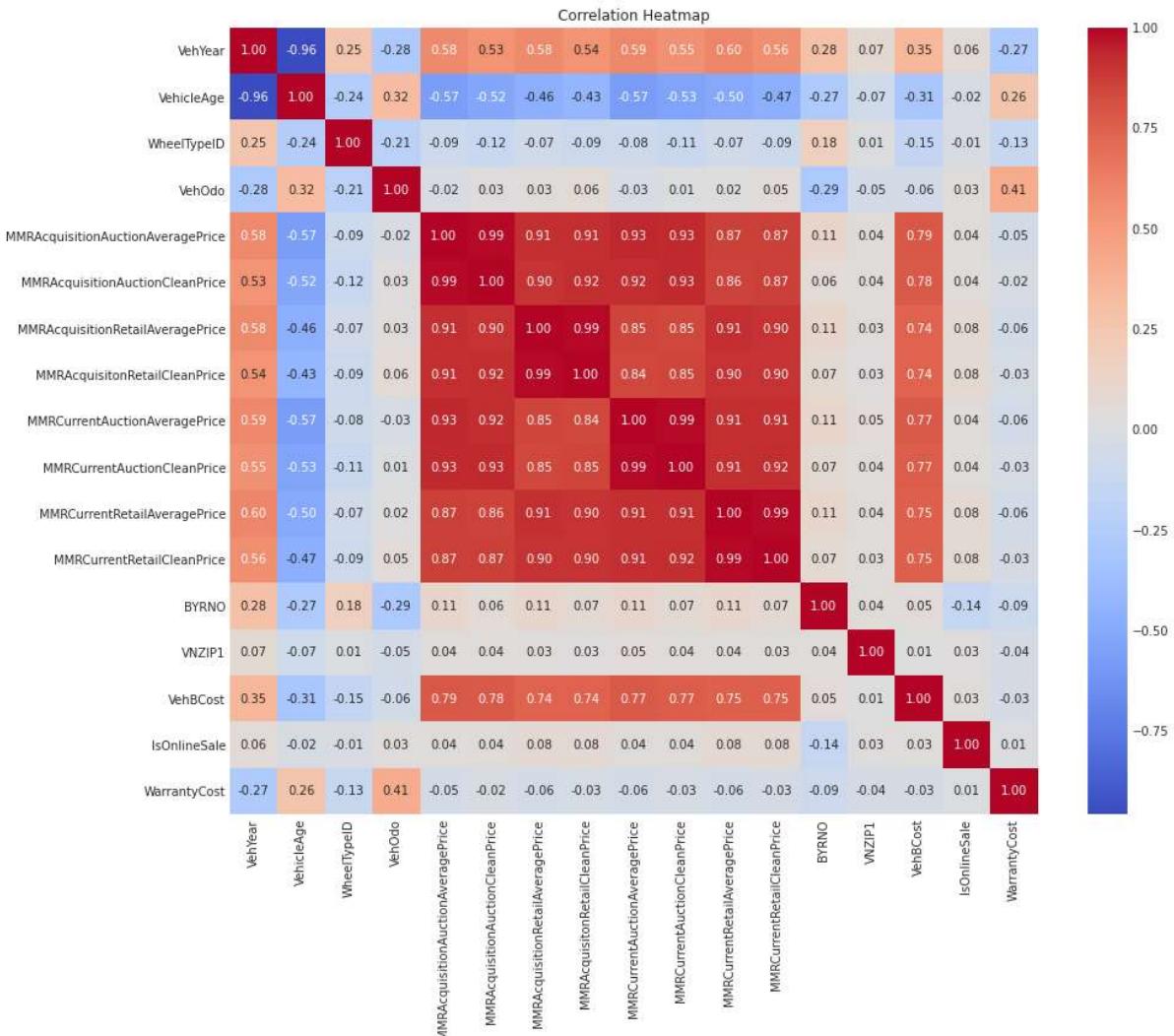
```
In [59]: a= (features_train.index == target_train.index)  
np.count_nonzero(~a)
```

```
# ~a ifadesi bu boolean diziyi tersine çevirir ve np.count_nonzero(~a) ifadesi bu t
# Bu, indislerin eşit olmadığı konumların sayısına denk gelir.
```

Out[59]: 0

In [60]:

```
# Heatmap
plt.figure(figsize=(15, 12))
sns.heatmap(features_train.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



**Glückwunsch:** Du hast deine Daten bereinigt und diese Schritte in einer Funktion zusammengefasst. So kannst du die Datenreinigung bei Bedarf unkompliziert wiederholen.

## Resample

Wie du sicherlich festgestellt hast, sind die Zielkategorien im Datensatz sehr unausgeglichen. Es kann also notwendig sein, dein Trainingsdatenset zu resampeln. In *Unausgegliche* Zeilkategorien (Modul 2, Kapitel 3) hast du verschiedene Methoden des *resampling* kennengelernt und auch gelernt, wie man `imblearn.pipeline` nutzt, um verschiedene Samplingmethoden zu testen.

In [61]:

```
'''#SMOTE
# SMOTE
from imblearn.over_sampling import SMOTE
```

```

features_copy = features_train.copy()
target_copy = target_train.copy()
#print(features_copy.shape, target_copy.shape)

# Wähle die numerischen Spalten im features_train DataFrame aus (exclude datetime64
numeric_columns = list(features_copy.select_dtypes(include='number', exclude='datet

# Konvertiere die ausgewählten numerischen Spalten in ein NumPy-Array
features_train_array = features_copy.loc[:, numeric_columns].values

# Konvertiere target_train in ein NumPy-Array
target_train_array = target_copy.values

# Initialisiere das SMOTE (Synthetic Minority Over-sampling Technique) Objekt
smotesampler = SMOTE(random_state=42)

# Trainingsdaten mit SMOTE-Methode neu sampeln
features_train_smote, target_train_smote = smotesampler.fit_resample(features_train_ar

# Klassen anzeigen
pd.crosstab(index=target_train_smote, columns='count')'''
```

Out[61]:

```
"#SMOTE
# SMOTE
from imblearn.over_sampling import SMOTE
features_copy = features_train.copy()
target_copy = target_train.copy()
#print(features_copy.shape, target_copy.shape)
# Wähle die numerischen Spalten im features_train DataFrame aus (exclude datetime64 Spalten)
numeric_columns = list(features_copy.select_dtypes(include='number', exclude='datetime64').columns)
# Konvertiere die ausgewählten numerischen Spalten in ein NumPy-Array
features_train_array = features_copy.loc[:, numeric_columns].values
# Konvertiere target_train in ein NumPy-Array
target_train_array = target_copy.values
# Initialisiere das SMOTE (Synthetic Minority Over-sampling Technique) Objekt
smotesampler = SMOTE(random_state=42)
# Trainingsdaten mit SMOTE-Methode neu sampeln
features_train_smote, target_train_smote = smotesampler.fit_resample(features_train_array, target_train_array)
# Klassen anzeigen
pd.crosstab(index=target_train_smote, columns='count')"
```

In [62]:

```
'''print(features_train_smote.shape)
print(target_train_smote.shape)'''
```

Out[62]:

```
'print(features_train_smote.shape)\nprint(target_train_smote.shape)'
```

In [63]:

```
'''features_train_smote =pd.DataFrame(features_train_smote, columns=numeric_columns)
target_train_smote =pd.DataFrame(target_train_smote, columns=['IsBadBuy'])'''
```

Out[63]:

```
"features_train_smote =pd.DataFrame(features_train_smote, columns=numeric_columns)
\ntarget_train_smote =pd.DataFrame(target_train_smote, columns=['IsBadBuy'])"
```

In [64]:

```
'''print(type(features_train_smote))
print(type(target_train_smote))'''
```

Out[64]:

```
'print(type(features_train_smote))\nprint(type(target_train_smote))'
```

## Modeling

Kommen wir nun zur Kernaufgabe eines *Data Scientist*, dem Bauen von Maschine-Learning-Modellen. Wie du es im Training bereits gelernt hast, ist das Optimieren eines Modells ein iterativer Prozess. Die nachfolgenden Schritte wirst du also sicher mehr als einmal durchführen.