Gliederung

Die Arbeit wurde komplett in Python erstellt. Dazu wurden zwei Programme angwendet. Die IDE (integrated development environment) Pycharm wurde für den ersten Teil der Arbeit verwendet. Jupyter Notebook in dem auch diese Gesamtübersicht erstellt wurde, wurde für die weiteren Teile der Arbeit verwendet. Pycharm eignet sich besonders gut um Fehler im Code zu beheben. Jupyter Notebook ist besser zur Darstellung, Übersicht und Navigation des Codes geeignet.

Die Arbeit ist untergliedert in vier Teile:

1. Extrahieren der Daten und Auswahl an Prädiktoren
2. Deskripitive Statistik - Erstellen von Schaubildern
3. Induktive Statistik - Regressionen
4. Das aktuelle Sortiment (nicht in der schriftlichen Arbeit enthalten)

Verwendete Module zum Erstellen des Datensatzes:

In [ ]:
```
import json, re, csv, requests, sys, json, time
from math import sqrt
from collections import Counter
from checking_numbers import checking_number, getting_attributes, crea
ting_new_fieldnames
from bs4 import BeautifulSoup
from multiprocessing.pool import ThreadPool
```

Verwendete Module zur grafischen Darstellung der Merkmale:

In [12]:
```
import pandas as pd
import numpy as np
import matplotlib as mp
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
from matplotlib import colors
from mpl_toolkits.axes_grid1 import make_axes_locatable
import matplotlib.patches as mpatches
```

Verwendete Module zum Erstellen der Regressionen:

In [13]:
```
import statsmodels.api as sm
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
import statsmodels.stats.outliers_influence as outliers_influence
from patsy import dmatrices
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.linear_model import LinearRegression, RidgeCV
```

Extrahieren der Daten und Auswahl an Prädiktoren

Zurück zur Gliederung

Parsen des HTML-Codes und Speichern der Links zu Laptops

Zurück zu Extrahieren der Daten und Auswahl an Prädiktoren

Um die Daten aus dem Code zu extrahieren werden zunächst die einzelnen URLs aufgerufen. Die URL der ersten Seite aller Notebooks auf Alternate.de ist:

https://www.alternate.de/Alle-Notebooks/html/listings/1444138486045?lk=21363&size=120&hideFilter=false (https://www.alternate.de/Alle-Notebooks/html/listings/1444138486045?lk=21363&size=120&hideFilter=false)

Die zweite Seite der Laptops:

https://www.alternate.de/Alle-Notebooks/html/listings/1444138486045?lk=21363&page=2&size=120&hideFilter=false (https://www.alternate.de/Alle-Notebooks/html/listings/1444138486045?lk=21363&page=2&size=120&hideFilter=false)

Die dritte Seite der Laptops:

https://www.alternate.de/Alle-Notebooks/html/listings/1444138486045?lk=21363&page=3&size=120&hideFilter=false (https://www.alternate.de/Alle-Notebooks/html/listings/1444138486045?lk=21363&page=3&size=120&hideFilter=false)

usw...

Die zweite und die dritte Seite unterscheiden sich also nur hinsichtlich "page=...", auf der ersten Seite wurde "page=..." ausgelassen

Um also alle Links zu sämtlichen Seiten zu bekommen, auf denen sich die Links zu den einzelnen Laptops befinden, kann in allen Links die entsprechende Stelle für "page=..." angepasst werden. Die generierten Links werden einer Liste beigefügt:

In [3]:
```
BaseURLs = []
def getting_base_urls():
    BaseURLs.append("https://www.alternate.de/Alle-Notebooks/html/listings/1444138486045?lk=21363&size=120&hideFilter=false")
    for number in range(2, 16):
        BaseURLs.append(r"https://www.alternate.de/Alle-Notebooks/html/listings/1444138486045?lk=21363&page="+str(number)+"&size=120&hideFilter=false")
```

In der Liste "BaseURLs" befinden sich also alle Links unter denen die einzelnen Laptops aufgerufen werden können. Mittels des Moduls *requests* lässt sich der HTML-Code von einer beliebigen Webseite in den Speicher des Rechners laden.

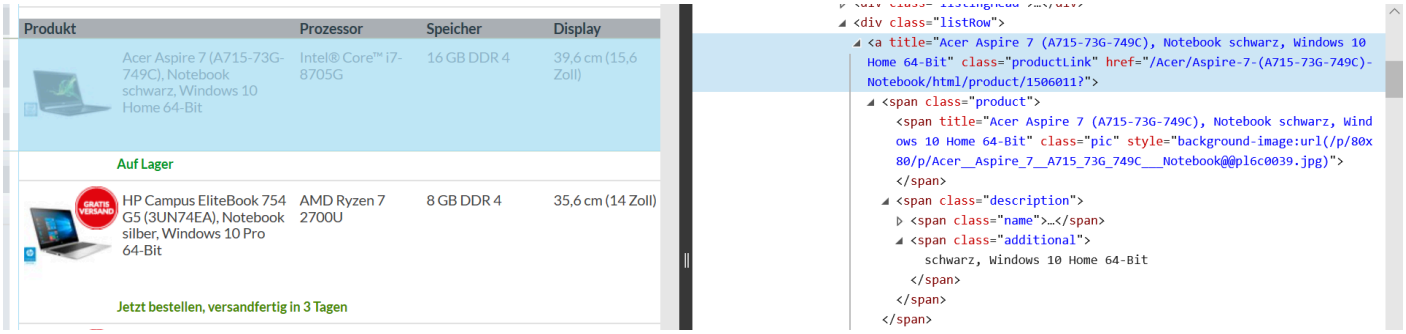
In Zeile 4 des untenstehenden Codes wird damit der Code geladen. Um den Server mit Anfragen nicht zu überladen und somit zu riskieren, dass die IP-Adresse des Rechners gesperrt wird, wird nur alle drei Sekunden eine Anfrage zum Laden einer neuen Seite gestellt. In Zeile 3 des untenstehenden Codes wurde diese Methode angewendet.

Anschließend werden mittels des Moduls BeautifulSoup und des LXML-Parsers sämtliche Links zu allen Laptops gefunden. Links in einem HTML-Code sind überlicherweise mit *href* gekennzeichnet. So auch in diesem Fall. Der Screenshot zeigt, dass sich die Links der Laptops in einem *<a-Tag* befinden (dunkelrot markiert). Es wird versucht aus sämtlichen <a-Tags der Klasse productLink:

```
for article in soup.find_all("a", class_= "productLink"):
```

das Attribut href, also den Link, in den Speicher des Rechners zu lesen:

```
URLs.append(article.get("href")+"\n")
```



In [ ]:
```
def getting_hyperlinks(BaseURLs):
    global BaseURL_searched
    time.sleep(3)
    source = requests.get(BaseURLs).text
    soup = BeautifulSoup(source, "lxml")
    for article in soup.find_all("a", class_= "productLink"):
        URLs.append(article.get("href")+"\n")
        print(article.get("href"))
    BaseURL_searched += 1
    sys.stdout.write("\r%i BaseURLs searched" % BaseURL_searched)
```

Extrahieren der Daten zu Laptops

Zurück zu Extrahieren der Daten und Auswahl an Prädiktoren

Um nun aus den Links zu den einzelnen Laptops die Daten zu extrahieren, die als Prädiktoren den Preis erklären sollen, wird erneut die HTML-Struktur der Webeseiten untersucht. Allerdings wird diesmal der Code einer Ebene tiefer untersucht, also der Code der einzelnen Laptops. Für jeden Laptops wird ein Dictionary angelegt, in dem sich alle Informationen des Laptops befinden:

```
indiv_information = {}
```

Erneut werden die Links, die mittels des obenstehenden Codes gefunden und gespeichert wurden aufgerufen und mittels des LXML-Parsers leicht zugänglich gemacht:

```
joined_url = "https://www.alternate.de"+URL
source = requests.get(joined_url).text
soup = BeautifulSoup(source, "lxml")
```

Der Preis wird mit derselben Methode, mit der auch die Links der einzelnen Laptops gefunden wurden, ausfindig gemacht:

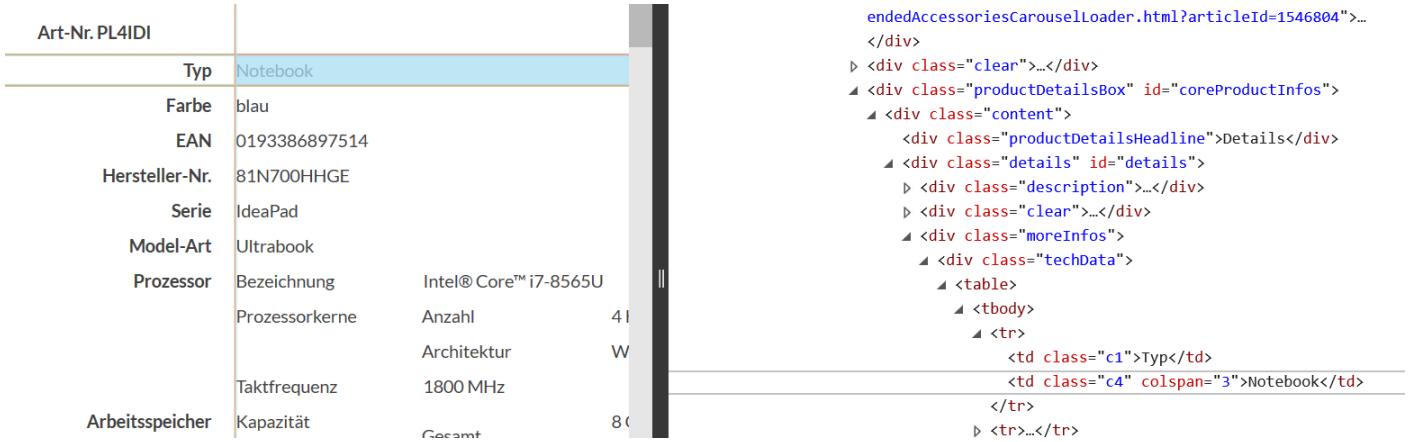
```
price = soup.find("span", itemprop= "price")
indiv_information.update({"Laptop_ID": laptops_found, "Preis": price.get("co
ntent")})
print(joined_url)
```

Ebenso wird mit den Prädiktoren verfahren. Aus der Grafik ist ersichtlich, dass die Liste der Eigenschaften des Laptops *tech Data* benannt wurde (blau markiert) und sich in einem <div-Tag befindet.

```
specification_table = soup.find("div", class_="techData")
```

Die einzelnen Merkmale in der Auflistung *tech Data* befinden sich in -Tags. Die erste Untergruppe in den -Tags bezieht sich auf das Merkmal und die zweite Untergruppe auf die Merkmalsausprägung. In vielen Fällen sind allerdings mehr als zwei Untergruppen vorhanden. In der Grafik ist ersichtlich, dass beispielsweise der Prozessor vier weitere Untergruppen besitzt: Prozessor-Bezeichnung, Prozessor-Prozessorkerne-Anzahl, Prozessor-Prozessorkerne-Architektur und Prozessor-Taktfrequenz. Es wurde also eine Funktion (getting_cat) erstellt, die die am weitesten rechts stehende (bzw. im HTML-Code die am weitesten unten stehende) Information als Ausprägung und alle davon links stehenden (bzw. darüber stehenden) Beschriftungen als Merkmal speichert. Ist links neben der Beschriftung kein Eintrag vorhanden wurde in der darüberliegenden Zeile die linksstehende Beschriftung dem Merkmal beigefügt. So ergibt sich für die Anzahl der Prozessorkerne: *Architektur-Prozessorkerne-Prozessor*

```
for specification in specification_table.find_all("tr"):
    getting_cat(specification, indiv_information)
indiv_information.update({"URL": joined_url})
return indiv_information
```


Art-Nr. PL4IDI
Typ
Notebook
Farbe
blau
EAN
0193386897514
Hersteller-Nr.
81N700HHGE
Serie
IdeaPad
Model-Art
Ultrabook
Prozessor
Bezeichnung
Intel® Core™ i7-8565U
Prozessorkerne
Anzahl
Architektur
Taktfrequenz
1800 MHz
Arbeitsspeicher
Kapazität
endedAccessoriesCarouselLoader.html?articleId=1546804">…

…

Details

…
…

<table>
<tbody>
<tr>
<td class="c1">Typ</td>
<td class="c4" colspan="3">Notebook</td>
</tr>
<tr>…</tr>

In [ ]:
```
def getting_information(URL):
    global laptops_found
    laptops_found += 1
    sys.stdout.write("\rData downloaded for %i laptops" % laptops_foun
d)
    indiv_information = {}
    joined_url = "https://www.alternate.de"+URL
    source = requests.get(joined_url).text
    soup = BeautifulSoup(source, "lxml")
    #Preis
    price = soup.find("span", itemprop= "price")
    indiv_information.update({"Laptop_ID": laptops_found, "Preis": pri
ce.get("content")})
    print(joined_url)
    #Prädiktoren
    specification_table = soup.find("div", class_="techData")
    for specification in specification_table.find_all("tr"):
        getting_cat(specification, indiv_information)
    indiv_information.update({"URL": joined_url})
    return indiv_information

def getting_cat(specification, indiv_information):
    namelist = []
    number = "c4"
    value = specification.find("td", class_=number).get_text()
    while number != "c1":
        key = None
        while key is None:
            try:
                key = specification.find("td", class_=number).previous
_sibling
                keytext = key.string
                if keytext is None:
                    specification = specification.previous_sibling
                else:
                    number = "".join(key["class"])
                    namelist.append(keytext)
            except AttributeError:
                specification = specification.previous_sibling
    keytext = "-".join(namelist)
    indiv_information.update({keytext: value})
```

Da bisher nur Funktionen definiert wurden (def ...), diese aber nicht ausgeführt wurden, werden diese nun ausgeführt. Da außerdem zu über 1700 Laptops die Daten mittels Parser übersetzt und extrahiert werden sollen und dieser Prozess lange dauern könnte, wird Multithreading angewendet, also das Auslasten aller verfügbaren Prozessorkerne. In Python lässt sich Multithreading unter anderem mit dem Modul multiprocessing und der zugehörigen Funktion ThreadPool durchführen:

```
with ThreadPool() as p:
hyperlinks = p.map(getting_hyperlinks, BaseURLs)
individual_information = p.map(getting_information, URLs)
```

Die Daten der Laptops werden in einer Json-Datei gespeichert und der ThreadPool nach Ausführen des Programms geschlossen:

```
json.dump(individual_information, json_file)
p.close()
p.join()
```

In [ ]:
```
if __name__ == '__main__':

    BaseURLs = []  # Links der Seiten, unter denen die Laptops aufgeru
fen werden können
    URLs = []  # Links der einzelnen Laptops
    laptops_found = 0
    BaseURL_searched = 0

    json_file = open("laptop_information_neu.json", "w")
    getting_base_urls()
    with ThreadPool() as p:
        hyperlinks = p.map(getting_hyperlinks, BaseURLs)
        individual_information = p.map(getting_information, URLs)
        json.dump(individual_information, json_file)
        p.close()
        p.join()

    json_file.close()
```

Die gespeicherten Daten werden dann zur Auswahl der Prädiktoren und Normierung erneut geladen. Dazu wird ein eigenhändig geschriebenes Modul (creating_new_fieldnames) angewendet, dass eine Liste der Merkmale erstellt:

In [ ]:
```
with open("laptop_information.json", "r") as old_data:
    reader = json.load(old_data)
Fieldnames = creating_new_fieldnames(reader)
c = Counter(Fieldnames) # Überprüft die Anzahl der Merkmale
```

Zwei weitere eigenhändig geschriebene Module erlauben es zu überprüfen, wie häufig die einzelnen Merkmalsausprägungen auftreten und wie viele Merkmalsausprägungen vorhanden sind. Die Module sind nicht wesentlicher Bestandteil der Arbeit und werden deshalb nicht beschrieben. Eine zusätzliche Eigenschaft der Funktion _getting*attributes* ist, dass damit eine Excel-Tabelle der vier häufigsten Ausprägungen erstellt wird, die auf der letzten Seite der Arbeit abgebildet ist.

In [ ]:
```
checking_number(reader)
getting_attributes(reader, Fieldnames, Excel_Tabelle_speichern=True, N
ame_Excel_Tabelle="Übersicht Merkmale")
```

Beschränken der Merkmale auf analyserelevante Prädiktoren

Zurück zu Extrahieren der Daten und Auswahl an Prädiktoren

Anschließend wurde unterschieden zwischen Merkmalen die *MCAR* und *MNAR* zuzuschreiben sind, die in Listen zusammengefasst wurden. Es wurde unteschieden zwischen qualitativen und quantitativen *MNAR* -Merkmalen (_nicht_vorhanden*qualitativ* _, nicht_vorhanden*quantitativ), MCAR* -Merkmalen (behalten_liste), Eigenschaften die den Preis vermutlich stark beeinflussen, aber zu denen nur wenige Observationen Ausprägungen besitzen (_laptop_löschen*liste*) und Kathegorien, die nicht berücksichtigt wurden (_remove_c*list*):

In [ ]:
```
nicht_vorhanden_qualitativ = ["Tastatur-Beleuchtung", "Fingerabdrucksensor-Eingabe", "Mikrofon-Sound",
                  "Touchscreen-Eingabe","Mobilefunk Standards-Konnektivität", "NFC-Konnektivität", "Hinweis"]
nicht_vorhanden_quantitativ = ["Speicher 2-Datenspeicher", "Speicher 3-Datenspeicher", "Gesamt-Speicher-Grafik"]
behalten_liste = ["Anzahl-Prozessorkerne-Prozessor", "Bild-Display", "Speicher 1-Datenspeicher",
                "Taktfrequenz-Prozessor", "Typ-Arbeitsspeicher", "Auflösung-Display",
                "Lautsprecher-Sound", "Typ-Optisches Laufwerk", "Stromquelle-Akku-Stromversorgung"]
laptop_löschen_liste = ["Gehäuse-Beleuchtung", "G-Sync-Display", "Handvenensensor-Eingabe"]
remove_c_list =["Hersteller-Nr.", "Typ", "Layout-Tastatur-Eingabe", "Farbe", "Abmessungen", "vorhanden-Zubehör",
                "Bluetooth-Konnektivität", "Gewicht", "EAN", "Anzahl -Webcam-Kamera", "Kartenleser", "Art-Display",
                "Weitere Informationen", 'davon-Speicher-Grafik', "Gesamtkapazität-Datenspeicher",
                "Funktionen-Tastatur-Eingabe", "Maus/Touchpad-Eingabe", "Funktion",
                "SoundChip-Sound", "Architektur-Prozessorkerne-Prozessor", "Serie", "WLAN-Konnektivität",
                "Eigenschaft-Display", "LAN-Konnektivität", "Slots-Arbeitsspeicher",
                "Seitenverhältnis-Display", "Besonderheiten-Webcam-Kamera",
                "Dauer-Akku-Stromversorgung", "max. unterstützt -Kapazität-Arbeitsspeicher", "Helligkeit-Display",
                "Modellbezeichnung", "Detail-Display", "Sprache-Beleuchtung", "Webcam-Auflösung 1-Webcam-Kamera",
                "Frequenz-Display", "Model-Art", "Kapazität-Akku-Stromversorgung", "Feature", "Kontrast-Display", "Lesen-Formate-Optisches Laufwerk",
                "Schreiben-Formate-Optisches Laufwerk", "Chipsatz", "Sprache-Eingabe",
                "max. Leistungsaufnahme-Prozessor", "Besonderheiten-Rückseite-Kamera", "Anzahl Objektive-Rückseite-Kamera",
                "Auflösung 1-Rückseite-Kamera", "intern-Schnittstellen", "Zusätzliche Info-Eingabe",
                "Ladezeit-Akku-Stromversorgung", "Ortung", "Lizenz-Beleuchtung", "HDR-Display",
                "Zusätzliche Info-Beleuchtung", "optional-Zubehör", "Lizenz-Eingabe", "SLI-Grafik"]
```

Es werden also drei Funktionen erstellt, die erste Funktion (_deleting*observations*) löscht Laptops mit Ausprägungungen der in _laptop_löschen*liste* definierten Merkmale sowie Laptops, die die Merkmale aus _behalten*liste* nicht vorweisen. Zu quantitativen und qualitativen Merkmalen werden in _replacing_missing*data* bei nicht vorhandenen Ausprägungen geschlussfolgert, dass die Ausprägungen entweder _nicht*vorhanden* bzw. 0 ist. Nicht analyserelevante Merkmale werden in _removing_from*laptop* gelöscht:

In [ ]:
```
def deleting_observations(keeping_list, deleting_list):
    for keep_element in keeping_list:
        for delete_element in deleting_list:
            for laptop in reader:
                if keep_element not in laptop:
                    reader.remove(laptop)
                else:
                    if delete_element in laptop:
                        reader.remove(laptop)

deleting_observations(behalten_liste, laptop_löschen_liste)

def replacing_missing_data(list,  new_value):
    for attribute in list:
        for laptop in reader:
            if attribute not in laptop:
                    laptop[attribute] = new_value

replacing_missing_data(nicht_vorhanden_qualitativ, "nicht vorhanden")
replacing_missing_data(nicht_vorhanden_quantitativ, "0")

def removing_from_laptop(dellist):
    for laptop in reader:
        for element in dellist:
            if element in laptop:
                del laptop[element]
            try:
                Fieldnames.remove(element)
            except ValueError:
                continue

removing_from_laptop(remove_c_list)
```

Es wird erneut eine Excel-Tabelle mit den restlichen Ausprägungen erstellt:

In [ ]:
```
getting_attributes(reader, Fieldnames, Excel_Tabelle_speichern=True, N
ame_Excel_Tabelle="Übersicht Merkmale")
```

Vereinzelt waren die Informationen unvollständig. Wenn die Speicherart nicht ausdrücklich genannt wurde oder zwei Grafikkarten vorhanden waren (für ein Laptop der Fall), so werden auch diese Laptops aus der Analyse ausgeschlossen:

In [ ]:
```
def deleting_certain_obs(attribute, value):
    for laptop in reader:
        if laptop[attribute] == value:
            reader.remove(laptop)
            print("gelöscht aus", attribute, ":", laptop[attribute])
print("so viele Laptops sind noch im Datensatz: ", len(reader))
# Bei den folgenden Datenspeichern wurden keine Angaben zur Speicherar
t gemacht, weshalb sie gelöscht werden
deleting_certain_obs("Speicher 1-Datenspeicher", "512 GB")
deleting_certain_obs("Speicher 1-Datenspeicher", "256 GB")
deleting_certain_obs("Speicher 1-Datenspeicher", "1.000 GB [5400 U/mi
n]")
deleting_certain_obs("Speicher 1-Datenspeicher", "64 GB [Flash]")
deleting_certain_obs("Speicher 2-Datenspeicher", "1.000 GB [5400 U/mi
n]")
deleting_certain_obs("Speicher 3-Datenspeicher", "undefined [5400 U/mi
n]")
print("so viele Daten sind noch im Datenset: ", len(reader))
# Ein Laptop der zwei Grafikkarten besitzt wird gelöscht
deleting_certain_obs("Typ-Grafik", "2x NVIDIA GeForce RTX 2080")

print("so viele Laptops sind noch im Datensatz: ", len(reader))
```

Normieren der Prädiktoren

Zurück zu Extrahieren der Daten und Auswahl an Prädiktoren

Um die Daten zu normieren werden reguläre Ausdücke angewendet. Auch hier wird eine Funktion erstellt, die vergleicht, welchen Wert die Ausprägung vor und nach der Normierung angenommen hat:

In [ ]:
```
def general_regex(regex, subst, ct):
    already_printed = [] # wurde die Ausprägung und die folgende Normi
erung schon gedruckt, soll die Ausprägung nicht erneut
    # dargestellt werden, weshalb hierfür eine Liste bereits gedruckte
r Ausprägungen erstellt wird
    print(ct.center(80, "="))
    for laptop in reader:
        pass_statement = False
        if regex.fullmatch(laptop[ct]):
            if laptop[ct] not in already_printed:
                print("vor Normierung:".ljust(40, "-") + laptop[ct].rj
ust(40, "-"), 10 * " ", laptop["URL"], end= "")
                already_printed.append(laptop[ct])
                pass_statement = True
            laptop[ct] = regex.sub(subst, laptop[ct]).upper()
            if pass_statement is True:
                print("nach Normierung:".ljust(40, "-") + laptop[ct].r
just(40, "-"))
        else:
            if laptop[ct] not in already_printed:
                already_printed.append(laptop[ct])
```

Die einzelnen Regulären Ausdrücke werden erstellt und auf die Merkmale mittels der erstellen Funktion _general*regex* angewendet.

Besonderheiten dabei:

1. Für die Festplatte wird eine eigene Funktion erstellt, die den Speicher in SSD, HDD und EMMC unterteilt, wobei die Laptops mit EMMC-Speicherkarte allerdings nicht berücksichtigt werden
2. Für den Akku wird ebenfalls eine Funktion erstellt, die in Akku-Typ und Akku-Kapazität unterteilt

In [ ]:
```
# Prozessor
Prozessor_Kerne_regex = re.compile(r"(\d)(\sKerne)")
general_regex(Prozessor_Kerne_regex, r"\1", "Anzahl-Prozessorkerne-Pro
zessor")

# Arbeitsspeicher, Taktfrequenz des Prozessors
Taktfr_Arbeitssp_regex = re.compile(r"(\d\d?\d?\d?)(.*)")
general_regex(Taktfr_Arbeitssp_regex, r"\1", "Gesamt-Kapazität-Arbeits
speicher")
general_regex(Taktfr_Arbeitssp_regex, r"\1", "Taktfrequenz-Prozessor")

# Festplatte
HDD_SSD_eMMC_regex = re.compile(r"(\d?.?\d?\d\d) GB \[(.*)?(HDD|SSD|eM
MC)(.*)?\]", re.IGNORECASE)
SSD_regex = re.compile(r"(\d?.\d?\d\d) GB \[(.*)(M\.2|PCIe|SSHD)(.*)",
re.IGNORECASE)
for laptop in reader:
    laptop["HDD-Kapazität"], laptop["HDD-Anzahl"] = 0, 0
    laptop["SSD-Kapazität"], laptop["SSD-Anzahl"] = 0, 0
    laptop["EMMC-Kapazität"], laptop["EMMC-Anzahl"]= 0, 0
def split_Festplatte(storage, type):
    for laptop in reader:
        if type in str(laptop[storage]):
            xy = laptop[storage].split(" ")
            laptop[type+"-Anzahl"] += 1
            laptop[type + "-Kapazität"] += int(xy[0].replace(".", ""))
            laptop[storage] = int(xy[0].replace(".", ""))
            laptop[storage+"-Typ"] = type
# 1 Festplatte
general_regex(HDD_SSD_eMMC_regex, r"\1 \3", "Speicher 1-Datenspeiche
r")
general_regex(SSD_regex, r"\1 SSD", "Speicher 1-Datenspeicher")
split_Festplatte("Speicher 1-Datenspeicher", "HDD")
split_Festplatte("Speicher 1-Datenspeicher", "SSD")
split_Festplatte("Speicher 1-Datenspeicher", "EMMC")
# 2 Festplatte
general_regex(HDD_SSD_eMMC_regex, r"\1 \3", "Speicher 2-Datenspeiche
r")
general_regex(SSD_regex, r"\1 SSD", "Speicher 2-Datenspeicher")
split_Festplatte("Speicher 2-Datenspeicher", "HDD")
split_Festplatte("Speicher 2-Datenspeicher", "SSD")
split_Festplatte("Speicher 2-Datenspeicher", "EMMC")
for laptop in reader:
    if "Speicher 2-Datenspeicher-Typ" not in laptop:
        laptop["Speicher 2-Datenspeicher-Typ"] = "nicht vorhanden"
# 3 Festplatte
general_regex(HDD_SSD_eMMC_regex, r"\1 \3", "Speicher 3-Datenspeiche
r")
general_regex(SSD_regex, r"\1 SSD", "Speicher 3-Datenspeicher")
split_Festplatte("Speicher 3-Datenspeicher", "HDD")
split_Festplatte("Speicher 3-Datenspeicher", "SSD")
split_Festplatte("Speicher 3-Datenspeicher", "EMMC")
for laptop in reader:
    if "Speicher 3-Datenspeicher-Typ" not in laptop:
        laptop["Speicher 3-Datenspeicher-Typ"] = "nicht vorhanden"
removing_from_laptop(["Speicher 1-Datenspeicher", "Speicher 2-Datenspe
icher", "Speicher 3-Datenspeicher"])
```

```
removing_from_laptop(["Speicher 1-Datenspeicher-Typ", "Speicher 2-Datenspeicher-Typ", "Speicher 3-Datenspeicher-Typ"])

# Display-Größe
Display_regex = re.compile(r"(.*)((\d\d(,\d)?) Zoll)(.*)")
general_regex(Display_regex, r"\3", "Bild-Display")

# Laufwerk
Laufwerk_regex = re.compile(r"(DVD-Brenner|DVD-Combo|DVD-ROM|Intern CD/DVD|Blu-ray-Combo)")
general_regex(Laufwerk_regex, "vorhanden", "Typ-Optisches Laufwerk")

# Typ Arbeitsspeicher
Arbeitsspeicher_regex = re.compile(r"(DDR \d)(.*)")
general_regex(Arbeitsspeicher_regex, r"\1", "Typ-Arbeitsspeicher")

# Lautsprecher
Lautsprecher1_regex = re.compile(r"(1.0 Kanal|2.0 Kanal)")
Lautsprecher2_regex = re.compile(r"(4.0 Kanal|2.1 Kanal|2.2 Kanal|7.1 Kanal)")
general_regex(Lautsprecher1_regex, "Standard Lautsprecher", "Lautsprecher-Sound")
general_regex(Lautsprecher2_regex, "Überdurchschn Lautsprecher", "Lautsprecher-Sound")

# Akku
Akku_regex = re.compile(r"(.*)(Polymer|Ion)(.*)(\d\d(.\d)?)(.*)", re.IGNORECASE)
general_regex(Akku_regex, r"\2 \4", "Stromquelle-Akku-Stromversorgung")
Akku_mit_Info = re.compile(r"(ION|POLYMER) (\d\d)(,\d)?")
def split_Akku():
    for laptop in reader:
        if Akku_mit_Info.fullmatch(laptop["Stromquelle-Akku-Stromversorgung"]):
            xy = laptop["Stromquelle-Akku-Stromversorgung"].split(" ")
            laptop["Akku-Typ"] = xy[0]
            laptop["Akku-Kapazität"] = xy[1].replace(",", ".")
        else:
            del laptop["Stromquelle-Akku-Stromversorgung"]
split_Akku()
removing_from_laptop(["Stromquelle-Akku-Stromversorgung"])

# Marke
for laptop in reader:
    laptop["Marke"] = laptop["URL"].split("/")[3]
Marke_regex = re.compile(r"OMEN-by-(HP)|OMEN-X-by-(HP)|(H)ewlett-(P)ackard")
general_regex(Marke_regex, r"\1\2\3\4", "Marke")

# Betriebssystem
Betriebssystem_regex = re.compile(r"(.*?)(Windows|Mac|Linux|DOS|Google|ohne Betriebssystem)(.*)", re.IGNORECASE)
general_regex(Betriebssystem_regex, r"\2", "Betriebssystem")

# externe Schnittstellen
Schnittstellen_regex = re.compile(r"(.*?)(((\d)x)(.*?))*")
```

```
general_regex(Schnittstellen_regex, (r"\4 "), "extern-Schnittstellen")
for laptop in reader:
    laptop["extern-Schnittstellen"] = (filter(None, re.sub("( )*", "
", laptop["extern-Schnittstellen"]).split(" ")))
    laptop["extern-Schnittstellen"] = sum([int(i) for i in laptop["ext
ern-Schnittstellen"]])

# Generalüberholt
for laptop in reader:
    laptop["Zustand"] = "neu"
    if "Generalüberholt" in laptop["URL"].split("/")[4]:
        laptop["Zustand"] = "generalüberholt"

# Rabattaktion
for laptop in reader:
    laptop["Rabatt_Lehreinrichtung"] = "nicht vorhanden"
    if "Kaufberechtigt sind:" in laptop["Hinweis"]:
        laptop["Rabatt_Lehreinrichtung"] = "vorhanden"
    del laptop["Hinweis"]
```

Einige Merkmalsausprägungen werden zusammegefasst. Sämtliche Ausprägungen des Mobilfunks und der Tastatur-Beleuchtung werden zusammegefasst zu *vorhanden,* wenn die Ausprägung nicht *nicht vorhanden* ist

In [ ]:
```
def standardizing_remaining(pass_statement, else_value, new_value, att
ribute = "Typ-Grafik"):
    for laptop in reader:
        if pass_statement:
            if laptop[attribute] != else_value:
                laptop[attribute] = new_value
        else:
            if laptop[attribute] == else_value:
                laptop[attribute] = new_value

# Mobilfunk
standardizing_remaining(True, "nicht vorhanden", "vorhanden", "Mobilef
unk Standards-Konnektivität")
# Tastatur-Beleuchtung
standardizing_remaining(True, "nicht vorhanden", "vorhanden", "Tastatu
r-Beleuchtung")
```

Erstellen der Pixeldichte

Zurück zu Extrahieren der Daten und Auswahl an Prädiktoren

Zur Berechnung der Pixeldichte wird ebenfalls eine eigene Funktion erstellt:

In [ ]:
```
def generating_ppi():
    for laptop in reader:
        xy = laptop["Auflösung-Display"].split(" x ")
        x = xy[1].split(" ")
        y = xy[0].replace(".", "")
        x = x[0].replace(".", "")
        laptop["Bild-Display"] = float(laptop["Bild-Display"].replace
(",", "."))
        temp2 = sqrt(int(y)**2+int(x)**2)
        laptop["PPI-Display"] = int(temp2/laptop["Bild-Display"])

generating_ppi()
```

Beifügen der Daten zu Grafikkarte

Zurück zu Extrahieren der Daten und Auswahl an Prädiktoren

Da an dieser Stelle der Datensatz der Alternate GMBH und der Datensatz der Grafikkarten zusammengefasst werden soll, werden einige Ausprägungen abgeändert, nachdem überprüft worden ist, ob es sich hierbei um dieselben Grafikkarten handelt. Die Bezeichnung der Grafikkarten selbt ist hier Mittel um die zwei Datensätze zusammenzufügen:

In [ ]:
```
# Grafikkarte
standardizing_remaining(False, "AMD Radeon Vega 8", "AMD Radeon RX Veg
a 8")
standardizing_remaining(False, "AMD Radeon Vega 3", "AMD Radeon RX Veg
a 3")
standardizing_remaining(False, "AMD Radeon Vega 10", "AMD Radeon RX Ve
ga 10")
standardizing_remaining(False, "AMD Radeon Vega 6", "AMD Radeon RX Veg
a 6")
standardizing_remaining(False, "AMD Radeon 540X", "AMD Radeon RX 540
X")
standardizing_remaining(False, "AMD Radeon Pro WX 3100", "AMD Radeon P
ro WX 3100 Mobile")
standardizing_remaining(False, "AMD Radeon RX Vega M GL", "AMD Radeon
RX Vega M GL / 870")
standardizing_remaining(False, "AMD Radeon R3", "AMD Radeon R3 (Mullin
s/Beema)")
standardizing_remaining(False, "Intel HD Graphics 400", "Intel HD Grap
hics 400 (Braswell)")
standardizing_remaining(False, "Intel HD Graphics 405", "Intel HD Grap
hics 405 (Braswell)")

for laptop in reader:
    laptop["Typ-Grafik"] = laptop["Typ-Grafik"].replace("®", "")
```

Einige Grafikkarten sind nicht integriert und besitzen daher einen Grafikspeicher. In einigen Fällen wurde diese Angabe nicht gemacht:

In [ ]:
```
# Fälschlicherweise vom Verkäufer ausgelassen:
for laptop in reader:
    if laptop["Typ-Grafik"] == "NVIDIA GeForce GTX 1060":
        laptop["Gesamt-Speicher-Grafik"] = "6.144 MB"
    if laptop["Typ-Grafik"] == "NVIDIA GeForce MX150":
        laptop["Gesamt-Speicher-Grafik"] = "4.096 MB"
    if laptop["Typ-Grafik"] == "NVIDIA GeForce RTX 2080":
        laptop["Gesamt-Speicher-Grafik"] = "8.192 MB"
    if laptop["Typ-Grafik"] == "NVIDIA Quadro P1000":
        laptop["Gesamt-Speicher-Grafik"] = "4.096 MB"
# Die Endungen MB werden entfernt
    laptop["Gesamt-Speicher-Grafik"] = laptop["Gesamt-Speicher-Grafi
k"].strip(" MB").replace(".", "")
```

Der Datensatz der Alternate GMBH und der Datensatz der Grafikkarten werden zusammengefügt. Das Erstellen des Datensatzes der Grafikkarten funktioniert dabei nach dem selben Prinzip wie der Datensatz der Alternate GMBH.

In [ ]:
```
with open(r"C:\Users\tobias\PycharmProjects\Alternate\Grafikkarten\gra
fik_information.json", "r") as json_file:
    information = json.load(json_file)
    Grafikkarten = []
    for alleGrafikkarten in information:
        Grafikkarten = alleGrafikkarten["Grafikkarten"]
        for Grafikkarte in Grafikkarten:
            # Es wird lediglich die Anzahl der Shader berücksichtigt
            del Grafikkarte["Architektur"]
            del Grafikkarte["Kerntakt"]
    for laptop in reader:
        # Die Grafikkarten der Laptops werden abgeglichen und durch di
e Anzahl der Pixelshader ergänzt, wenn die
        # Grafikkarte in beiden Datensätzen vertreten ist
        for Grafikkarte in Grafikkarten:
            if laptop["Typ-Grafik"]+ (" (Laptop)") == Grafikkarte["Mod
ell"]:
                laptop.update(Grafikkarte)
            elif laptop["Typ-Grafik"] == Grafikkarte["Modell"]:
                laptop.update(Grafikkarte)
```

Es wird erneut überprüft, welche Ausprägungen die einzelnen Merkmale besitzen und eine Excel Tabelle mit den vier häufigsten Ausprägungen mit den Namen *Übersicht Merkmale bearbeitet* erstellt:

In [ ]:
```
checking_number(reader)
Fieldnames = creating_new_fieldnames(reader)
getting_attributes(reader, Fieldnames, Excel_Tabelle_speichern=True, N
ame_Excel_Tabelle="Übersicht Merkmale bearbeitet")
print("im Datenset sind noch:", len(reader), "Laptops")
```

Der fertige Datensatz wird als CSV-Datei gespeichert:

In [ ]:
```
with open("edited_laptops.csv", "w", newline= "") as newfile:
    with open(r"C:\Users\tobias\Desktop\Uni\Marburg\Bachelorarbeit\Alternate\edited_laptops.csv", "w", newline="") as newfile2:
        with open(r"C:\Users\tobias\AnacondaProjects\Alternate\edited_laptops.csv", "w", newline="") as newfile3:
            writer = csv.DictWriter(newfile, Fieldnames)
            writer2 = csv.DictWriter(newfile2, Fieldnames)
            writer3 = csv.DictWriter(newfile3, Fieldnames)
            writer.writeheader()
            writer2.writeheader()
            writer3.writeheader()
            for laptop in reader:
                laptop["URL"] = laptop["URL"].rstrip("\n")
                writer.writerow(laptop)
                writer2.writerow(laptop)
                writer3.writerow(laptop)
```

Deskriptive Statistik

Zurück zur Gliederung

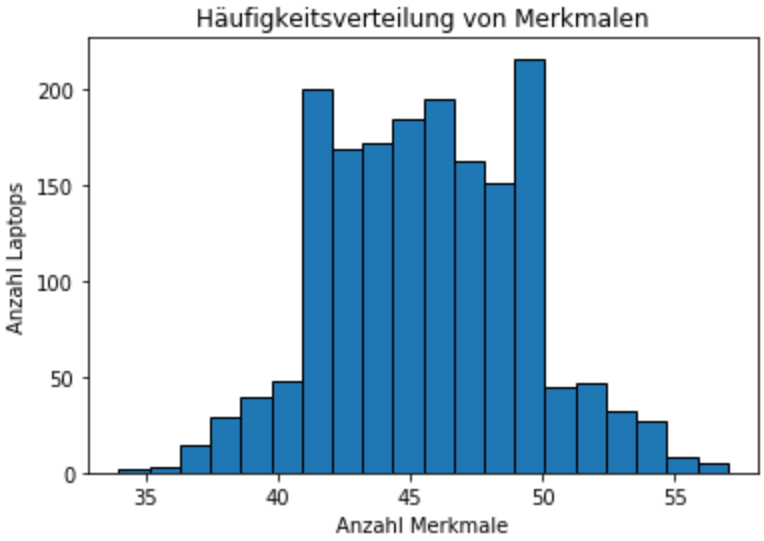
Der ursprüngliche Datensatz

Zurück zu Deskriptive Statistik

In [882]:
```
pd.set_option('display.max_rows', 5)
```

In [883]:
```
df = pd.read_json("laptop_information.json")
```

In [884]:
```
df['Anzahl_Merkmale'] = df.apply(lambda x: x.count(), axis=1)
df_hist = plt.hist(df.Anzahl_Merkmale, bins= 20, edgecolor="black")
plt.title("Häufigkeitsverteilung von Merkmalen")
plt.ylabel("Anzahl Laptops")
plt.xlabel("Anzahl Merkmale")
plt.savefig("hfgt_merkmale_datenset_alt");
```



Im Durchschnitt besitzt ein Merkmalsträger Informationen zu rund 46 Merkmalen:

In [885]:
```
df.Anzahl_Merkmale.mean()
```

Out[885]:
```
45.64779874213836
```

Der bearbeitete Datensatz

Zurück zu Deskriptive Statistik

In [886]:
```
df = pd.read_csv('edited_laptops.csv', encoding='cp1252')
pd.options.display.max_columns = None
```

In [887]:
```
df
```

Out[887]:

	Laptop_ID	Preis	Bezeichnung-Prozessor	Anzahl-Prozessorkerne-Prozessor	Taktfrequenz-Prozessor	Gesamt-Kapazität-Arbeitsspeicher	Arbeitss
0	1	1699.0	Intel® Core™ i7-8750H	6	2200	16	
1	2	299.0	AMD Ryzen 3 3200U	2	2600	8	
...	...	...	...	...	...	...	
1437	1747	949.0	Intel® Core™ i5-8300H	4	2300	8	
1438	1748	1299.0	AMD Ryzen 7 3750H	4	2300	16	

1439 rows × 37 columns

die automatisch generierten Namen der Merkmale werden umbenannt:

In [888]:
```
df.columns = df.columns.str.replace("-", "_")
df = df.rename(columns={"Bezeichnung_Prozessor":"Prozessor",
                        "Anzahl_Prozessorkerne_Prozessor":"Kerne",
                        "Taktfrequenz_Prozessor": "Takt_Prozessor",
                        "Gesamt_Kapazität_Arbeitsspeicher": "Arbeitsspeicher",
                        "SSD_Kapazität":"SSD",
                        "HDD_Kapazität":"HDD",
                        "Typ_Optisches Laufwerk": "Laufwerk",
                        "Gesamt_Speicher_Grafik": "Speicher_Grafik",
                        "Lautsprecher_Sound":"Lautsprecher",
                        "Fingerabdrucksensor_Eingabe": "Fingerabdrucksensor",
                        "Touchscreen_Eingabe": "Touchscreen",
                        "Mobilefunk Standards_Konnektivität": "Mobilfunk",
                        "NFC_Konnektivität": "NFC",})
```

Im bearbeiteten Datensatz kommen vier Merkmale vor, zu denen Angaben fehlen, alle anderen Variablen kamen 1439 mal vor:

In [889]:
```
df_anzahl = df.count().rename_axis('Merkmale').reset_index(name='Anzahl').sort_values(["Anzahl"], ascending=False)
# Die letzten 5 Merkmale mit den wenigsten Ausprägungen
df_anzahl.tail(5)
```

Out[889]:

	Merkmale	Anzahl
14	Mikrofon_Sound	1439
35	Modell	1401
36	Pixelshader	1401
29	Akku_Typ	1294
30	Akku_Kapazität	1294

Der berarbeitete Datensatz ohne fehlende Werte

Zurück zu Deskriptive Statistik

Anzahl an Observationen, wenn fehlende Merkmale ausgeschlossen werden:

In [890]:
```
df_neu = df.dropna()
del df
df_neu.set_index("Laptop_ID", inplace=True)
print(len(df_neu))
```

```
1260
```

Anzahl an Observationen, wenn Duplikate ausgeschlossen werden:

In [891]:
```
exogene_var = list(set(df_neu.columns) - set(["Laptop_ID", "Preis", "URL"]))
df_neu_exogen = df_neu[exogene_var]
```

In [892]:
```
df_neu_exogen.drop_duplicates()
len(df_neu_exogen)
```

Out[892]:
```
1260
```

Es sind also keine Duplikate im Datenset, die Anzahl beträgt auch nach Löschen der Duplikate 1260

Alle Merkmale haben dieselbe Anzahl:

In [893]:
```
df_anzahl_neu = df_neu.count().rename_axis('Merkmale').reset_index(nam
e='Anzahl').sort_values(["Anzahl"], ascending=False)
df_anzahl_neu
```

Out[893]:

	Merkmale	Anzahl
0	Preis	1260
1	Prozessor	1260
...	...	...
15	Betriebssystem	1260
35	Pixelshader	1260

36 rows × 2 columns

Löschen einiger Merkmale und Merkmalsausprägungen

Zurück zu Deskriptive Statistik

Typ-Grafik und Modell entsprechen einander. Die Merkmale _Prozessor, Auflösung_Display, Typ*Grafik* und *Modell* sind die einzigen String-Variablen, die nicht als Dummy-Variablen dargestellt werden sollen und jeweils andere Merkmale haben, die den Zusammenhang zu dem entsprechenden Preis besser erkären und werden deshalb gelöscht:

In [894]:
```
pd.set_option('display.max_colwidth', -1)
df_neu[["Prozessor", "Auflösung_Display", "Typ_Grafik", "Modell", "URL"]].head(4)
```

Out[894]:

Laptop_ID	Prozessor	Auflösung_Display	Typ_Grafik	Modell	URL
1	Intel® Core™ i7-8750H	1.920 x 1.080 Pixel	NVIDIA GeForce GTX 1070	NVIDIA GeForce GTX 1070 (Laptop)	https://www.alternate.de/MSI/GS65-8RF-078-Stealth-Thin-Notebook/html/product/1433639?
2	AMD Ryzen 3 3200U	1.920 x 1.080 Pixel	AMD Radeon RX Vega 3	AMD Radeon RX Vega 3	https://www.alternate.de/HP/15-db1224ng-Notebook/html/product/1556331?
3	Intel® Core™ i7-8750H	1.920 x 1.080 Pixel	NVIDIA GeForce GTX 1060	NVIDIA GeForce GTX 1060 (Laptop)	https://www.alternate.de/Razer/Blade-15-Dual-Storage-(RZ09-02705G75-R3G1)-Notebook/html/product/1493919?
4	Intel® Core™ i7-8750H	1.920 x 1.080 Pixel	NVIDIA GeForce GTX 1060	NVIDIA GeForce GTX 1060 (Laptop)	https://www.alternate.de/OMEN-by-HP/15-dc0003ng-Notebook/html/product/1462417?

In [895]:
```
df_neu = df_neu.drop(["Prozessor", "Auflösung_Display", "Typ_Grafik", "Modell"], axis = 1)
```

Das gekürzte Datenset besteht nur aus neuen Laptops, das Merkmal Zustand wird deshalb ebenfalls gelöscht:

In [896]:
```
print("Zustand:\n", df_neu.Zustand.value_counts())
df_neu = df_neu.drop("Zustand", axis= 1);
```

```
Zustand:
 neu    1260
Name: Zustand, dtype: int64
```

die Anzahl der Datenspeicher setzt sich aus der Summe von Anzahl an SSDs und HDDs zusammen. Da ein grundlegender Unterschied zwischen diesen beiden Festplattentypen sollte, wird dieses Merkmal ebenfalls gelöscht

In [897]:
```
df_neu[["Anzahl_Datenspeicher", "HDD_Anzahl", "SSD_Anzahl", "EMMC_Anzahl"]].head(5)
```

Out[897]:

Laptop_ID	Anzahl_Datenspeicher	HDD_Anzahl	SSD_Anzahl	EMMC_Anzahl
1	1	0	1	0
2	1	0	1	0
3	2	1	1	0
4	2	1	1	0
5	1	0	1	0

In [898]:
```
df_neu = df_neu.drop("Anzahl_Datenspeicher", axis=1);
```

Einzelne Marken sind selten vertreten im Datenset und werden ebenfalls gelöscht. Die Wahl fiehl dabei auf die sechs größten Marken gemessen an ihrem Marktanteil. Marken, die weniger als 44 mal vertreten waren wurden deshalb gelöscht:

In [899]:
```
Anzahl_Marke = df_neu.groupby(["Marke"], as_index=False).size().to_frame("Anzahl").reset_index(level=0)
Liste = Anzahl_Marke[Anzahl_Marke.Anzahl < 44]
for Markealt in Liste.Marke:
    print("Laptops mit der Marke",Markealt, "werden gelöscht")
    df_neu = df_neu[df_neu["Marke"] != Markealt]
```

```
Laptops mit der Marke ALTERNATE werden gelöscht
Laptops mit der Marke AORUS werden gelöscht
Laptops mit der Marke Alienware werden gelöscht
Laptops mit der Marke Fujitsu werden gelöscht
Laptops mit der Marke GIGABYTE werden gelöscht
Laptops mit der Marke Huawei werden gelöscht
Laptops mit der Marke MSI werden gelöscht
Laptops mit der Marke Razer werden gelöscht
Laptops mit der Marke Schenker werden gelöscht
Laptops mit der Marke TrekStor werden gelöscht
Laptops mit der Marke XMG werden gelöscht
```

Anzahl an Observationen nach Löschen der Marken:

In [900]:
```
len(df_neu)
```

Out[900]: 1084

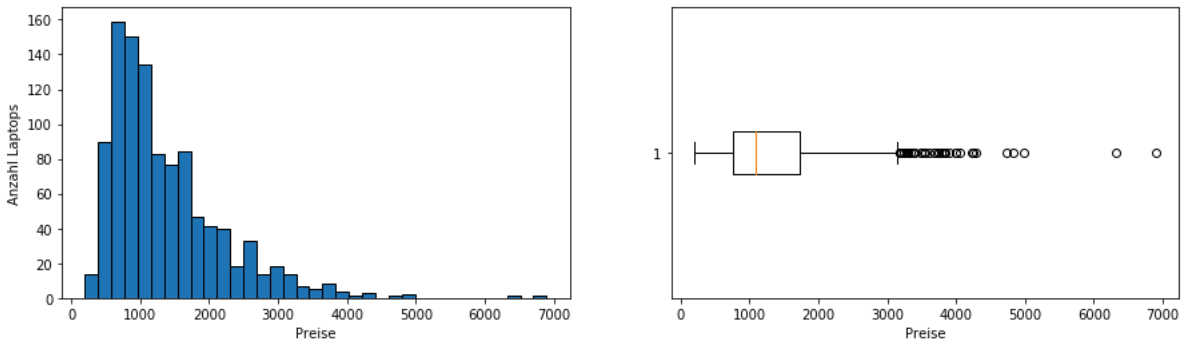
Laptops die eine eMMC Speicherkarte haben, werden gelöscht (35 Observationen):

In [901]:
```
df_neu = df_neu[df_neu.EMMC_Kapazität == 0]
del df_neu["EMMC_Kapazität"]
del df_neu["EMMC_Anzahl"]
len(df_neu)
```

Out[901]: 1049

Es werden nur Laptops bis einschließlich 4000 Euro betrachtet. Die Wahl von 4000 Euro war dabei recht willkürlich, aber als runde Zahl ansehnlicher

In [902]:
```
plt.subplot(1,2,1)
plt.hist(df_neu.Preis, bins = 35, edgecolor="black")
plt.ylabel("Anzahl Laptops")
plt.xlabel("Preise")
plt.subplot(1,2,2)
plt.xlabel("Preise")
plt.gcf().set_size_inches(15, 4, forward=True)
plt.boxplot(df_neu.Preis, vert = False, showbox = True);
#plt.yticks(None)
plt.ylabel(None)
plt.savefig("hfgkt_preisspanne_ungekürzt");
```



Laptops über 4000 Euro werden aus der Analyse ausgeschlossen, der neue Datensatz erneut gespeichert:

In [903]:
```
df_neu = df_neu[df_neu.Preis<=4000]
df_neu.to_json(r'analysierte merkmale.json')
df_neu.to_json(r'C:\Users\tobia\PycharmProjects\Alternate\analysierte merkmale.json')
len(df_neu)
```

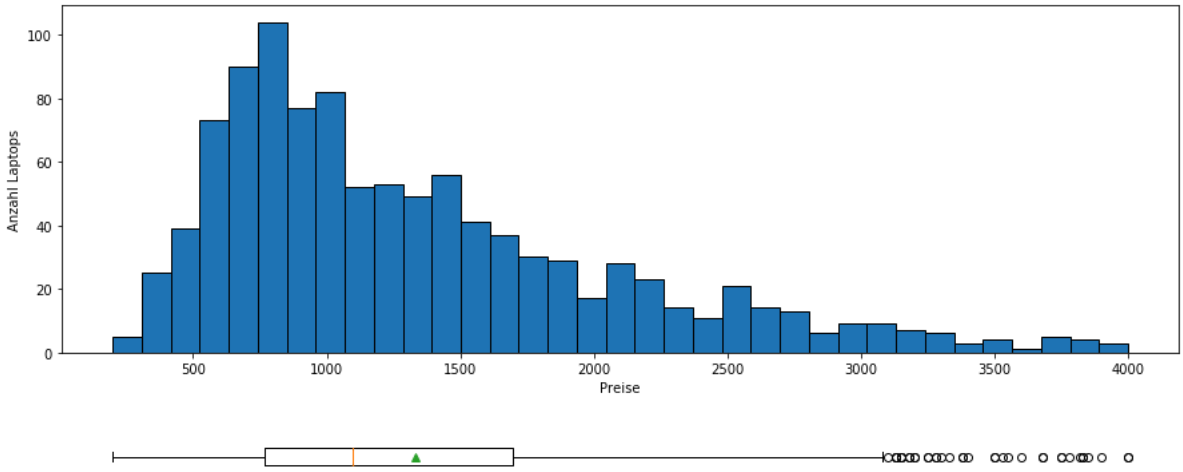
Out[903]: 1040

Klassieren in Preisspannen

Zurück zu Deskriptive Statistik

In [904]:
```
fig, (ax1, ax2) =plt.subplots(2,1, figsize= (15,7),gridspec_kw={'heigh
t_ratios': [3, 1]})
ax1.hist(df_neu.Preis, bins = 35, edgecolor="black")
ax1.set_ylabel("Anzahl Laptops")
ax1.set_xlabel("Preise")
ax2.boxplot(df_neu.Preis, vert=False, showmeans= True)
ax2.axis("off")

plt.savefig("hfgkt_preisspanne_gekürzt");
```



75% Quantil des Preises:

In [905]:
```
df_neu.Preis.quantile(q=0.75)
```

Out[905]:
```
1699.0
```

Durchschnittspreis:

In [906]:
```
df_neu.Preis.mean()
```

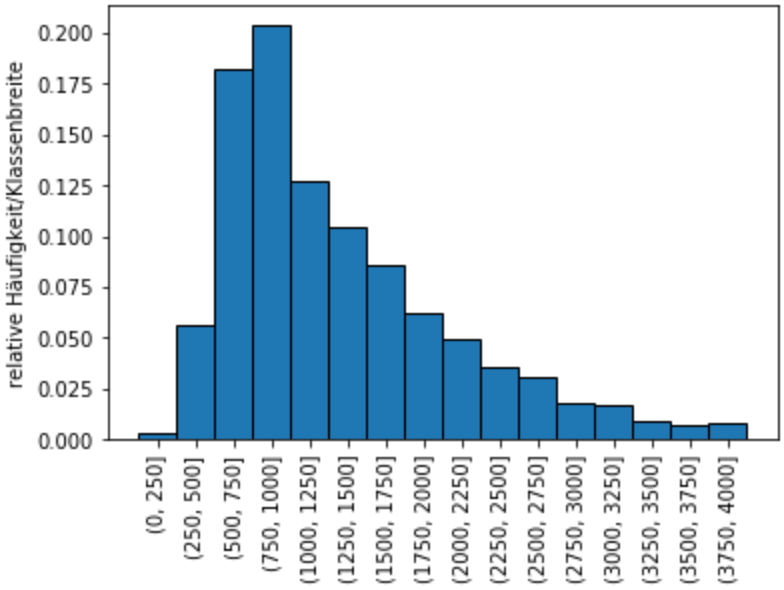
Out[906]:
```
1334.2133846153847
```

Die Laptops werden in Preisklassen unterteilt:

In [907]:
```
Unterteilung_Preis = np.linspace(0, 4000, 17).astype(int)
Klassenbreite = [(b-a)/250 for a, b in zip(Unterteilung_Preis.tolist()
[:-1], Unterteilung_Preis.tolist()[1:])]
Index = [((b-a)/2+a)/250 for a, b in zip(Unterteilung_Preis.tolist()[:
-1], Unterteilung_Preis.tolist()[1:])]
df_neu["Preisspanne"] = pd.cut(df_neu["Preis"], Unterteilung_Preis)
Kathegorien = df_neu["Preisspanne"].cat.categories
```

Graphische Darstellung der Preisklassen:

In [908]:
```
gruppenhäufigkeit = df_neu.groupby("Preisspanne").size()
relative_gruppenhäufigkeit = (gruppenhäufigkeit/len(df_neu))/Klassenbr
eite
plt.bar(Index, relative_gruppenhäufigkeit, edgecolor="black", width=Kl
assenbreite)
plt.ylabel("relative Häufigkeit/Klassenbreite")
plt.xticks(Index, Kathegorien, rotation = 90);
```



qualitative Merkmale

Zurück zu Deskriptive Statistik

die ersten zwei Ausprägungen des Datensatzes:

In [909]:
```
df_neu.head(2)
```

Out[909]:

Laptop_ID	Preis	Kerne	Takt_Prozessor	Arbeitsspeicher	Typ_Arbeitsspeicher	Bild_Display	Spei
2	299.0	2	2600	8	DDR 4	15.6	
4	899.0	6	2200	8	DDR 4	15.6	

Die qualitativen Merkmale sind noch in ihrer ursprünglichen Form:

In [910]:
```
df_neu[["Typ_Arbeitsspeicher", "Laufwerk", "Lautsprecher", "Tastatur_Beleuchtung", "Betriebssystem",
        "Fingerabdrucksensor", "Touchscreen"]].head(2)
```

Out[910]:

Laptop_ID	Typ_Arbeitsspeicher	Laufwerk	Lautsprecher	Tastatur_Beleuchtung	Betriebss
2	DDR 4	VORHANDEN	STANDARD LAUTSPRECHER	nicht vorhanden	BETRIEBSS
4	DDR 4	nicht vorhanden	STANDARD LAUTSPRECHER	vorhanden	WIN

Also müssen Dummy-Variablen erstellt werden, was mit dem Befehl _pandas.get*dummies()* möglich ist:

In [911]:
```
binäre_variablen = pd.get_dummies(df_neu[["Typ_Arbeitsspeicher", "Laufwerk", "Lautsprecher", "Tastatur_Beleuchtung",
                    "Fingerabdrucksensor", "Touchscreen", "Mobilfunk", "NFC",
                    "Akku_Typ", "Marke", "Betriebssystem", "Mikrofon_Sound", "Rabatt_Lehreinrichtung"]])
```

Zur graphischen Darstellung wird der Datensatz der qualitativen Merkmale um den Preis und die Preisspanne erweitert:

In [912]:
```
binäre_variablen["Preis"] = df_neu["Preis"]
binäre_variablen["Preisspanne"] = df_neu["Preisspanne"]
Merkmale = binäre_variablen.columns.tolist()
Merkmale = Merkmale[-1:] + Merkmale[:-1]
binäre_variablen = binäre_variablen[Merkmale]
```

In [913]:
```
binäre_variablen.head(2)
```

Out[913]:

Laptop_ID	Preisspanne	Typ_Arbeitsspeicher_DDR3	Typ_Arbeitsspeicher_DDR4	Laufwerk_VORHAND
2	(250, 500]	0	1	
4	(750, 1000]	0	1	

Häufig befinden sich Leerzeichen im Namen der durch _get*dummies* generierten Merkmale, welche ersetzt werden durch "_":

In [914]:
```
for Merkmal in Merkmale:
    binäre_variablen.columns = binäre_variablen.columns.str.replace(" ", "_")
binäre_variablen.head(2)
```

Out[914]:

Laptop_ID	Preisspanne	Typ_Arbeitsspeicher_DDR_3	Typ_Arbeitsspeicher_DDR_4	Laufwerk_VORH
2	(250, 500]	0	1	
4	(750, 1000]	0	1	

Qualitative Merkmale mit mehr als zwei Ausprägungen

Zurück zu Deskriptive Statistik

Im bearbeiteten Datensatz gibt es zwei Merkmale mit mehr als zwei Ausprägungen.

1. Marke
2. Betriebssystem

Standard-Graph für Marke und Betriebssystem:

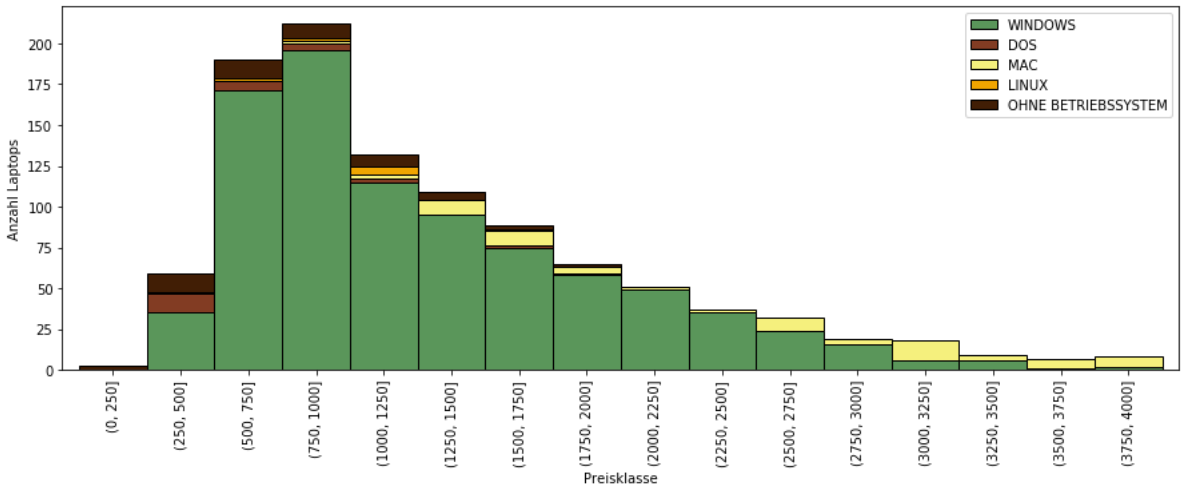
In [915]:
```
def gestapelt_balken(titel, zweite_gruppe, reihenfolge,farbe, klassieren=False, gegeben=False, ax=None):
    unterklasse = df_neu.groupby(["Preisspanne", zweite_gruppe]).size().to_frame("Anzahl")
    fnl = unterklasse.pivot_table("Anzahl", ["Preisspanne"], zweite_gruppe)
    fnl = fnl.fillna(0).rename_axis(None).rename_axis("Preisspanne", axis=1)
    fnl["Andere"] = 0
    if klassieren:
        for column in fnl:
            if fnl[column].sum()<43:
                fnl["Andere"] += fnl[column]
                del fnl[column]
    if gegeben:
        fnl = fnl[reihenfolge]
    graph =fnl.plot.bar(stacked= True, width=1, ax=ax, colormap= farbe, edgecolor= "black", legend= True, title=titel)
    ax.set_title(titel, fontsize=14)
    patches, labels = graph.get_legend_handles_labels()
    ax.legend(patches, labels, loc='best')
    ax.set_ylabel("Anzahl Laptops")
    ax.set_xlabel("Preisklasse")
```

Jeder Marke wird eine Farbe zugewiesen:

In [916]:
```
cm = ["#96D65E", "#5B995A", "#F8F17F", "#F3A800", "#824026", "#431F0
7"]
#print(plt.rcParams['axes.prop_cycle'].by_key()['color'])
farben = colors.ListedColormap(cm)
df_neu.loc[df_neu.Marke == "ASUS", "Farbe"] = "#96D65E"
df_neu.loc[df_neu.Marke == "Acer", "Farbe"] = "#5B995A"
df_neu.loc[df_neu.Marke =="Apple", "Farbe"] = "#F8F17F"
df_neu.loc[df_neu.Marke == "Dell", "Farbe"] = "#F3A800"
df_neu.loc[df_neu.Marke == "HP", "Farbe"] = "#824026"
df_neu.loc[df_neu.Marke == "Lenovo", "Farbe"] = "#431F07"
```

Übersicht Betriebssystem:

In [917]:
```
fig, ax = plt.subplots(1,1, figsize=(15,5))
gestapelt_balken(None, "Betriebssystem",
                 ["WINDOWS", "DOS", "MAC", "LINUX", "OHNE BETRIEBSSYST
EM"],
                 colors.ListedColormap([cm[i] for i in (1,4,2,3,5)]),
False, True, ax=ax)
plt.savefig("betriebssystem_in_preiskl", bbox_inches = "tight")
```

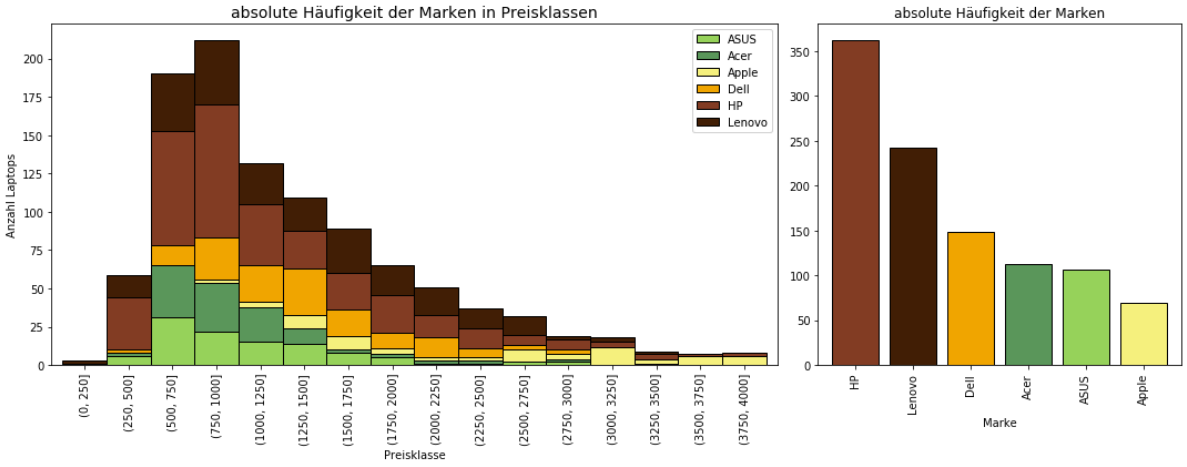


In [918]:
```
for sstm in df_neu.Betriebssystem.unique().tolist():
    print("Durchschnittlicher Preis für das Betriebssystem", sstm,
":", df_neu[df_neu.Betriebssystem == sstm].Preis.mean())
```

```
Durchschnittlicher Preis für das Betriebssystem OHNE BETRIEBSSYSTEM :
859.3294117647059
Durchschnittlicher Preis für das Betriebssystem WINDOWS : 1295.1483257
918553
Durchschnittlicher Preis für das Betriebssystem DOS : 699.576923076923
1
Durchschnittlicher Preis für das Betriebssystem LINUX : 984.5
Durchschnittlicher Preis für das Betriebssystem MAC : 2475.52173913043
5
```

Übersicht Marke:

In [919]:
```
fig, (ax1, ax2) = plt.subplots(1,2,figsize=(15,6),gridspec_kw={'width_ratios': [2, 1]})
gestapelt_balken("absolute Häufigkeit der Marken in Preisklassen", "Marke", [], farben, klassieren=True, ax=ax1)
df_neu.groupby(["Marke"])
bar = df_neu.groupby(["Marke"]).size().sort_values(ascending=False)
farbe = [x for _,x in sorted(zip(df_neu.groupby(["Marke"]).size(),mp.cm.get_cmap(farben)(np.linspace(0,1,6))))][::-1]
bar.plot.bar(ax=ax2, title="absolute Häufigkeit der Marken", width=0.8, edgecolor="black", color =farbe)
plt.tight_layout()
plt.savefig("marke_in_preiskl_mit_bar")
```



In [920]:
```
for Marke in df_neu.Marke.unique().tolist():
    print("Durchschnittlicher Preis für die Marke", Marke, ":", df_neu[df_neu.Marke == Marke].Preis.mean())
```

```
Durchschnittlicher Preis für die Marke HP : 1222.4856077348068
Durchschnittlicher Preis für die Marke ASUS : 1088.018691588785
Durchschnittlicher Preis für die Marke Acer : 1035.2083035714286
Durchschnittlicher Preis für die Marke Lenovo : 1377.9247933884296
Durchschnittlicher Preis für die Marke Dell : 1408.1891891891892
Durchschnittlicher Preis für die Marke Apple : 2475.521739130435
```

Qualtiative Merkmale mit zwei Ausprägungen

Zurück zu Deskriptive Statistik

Im Datenset gibt es 11 qualitative Merkmale mit zwei Ausprägungen:

1. Tastatur Beleuchtung
2. Fingerabdrucksensor
3. Typ-Arbeitsspeicher
4. Tochscreen
5. Lautsprecher
6. Laufwerk
7. Mobilfunk
8. NFC
9. Akku-Typ
10. Mikrofon-Sound
11. Rabatt-Lehreinrichtung

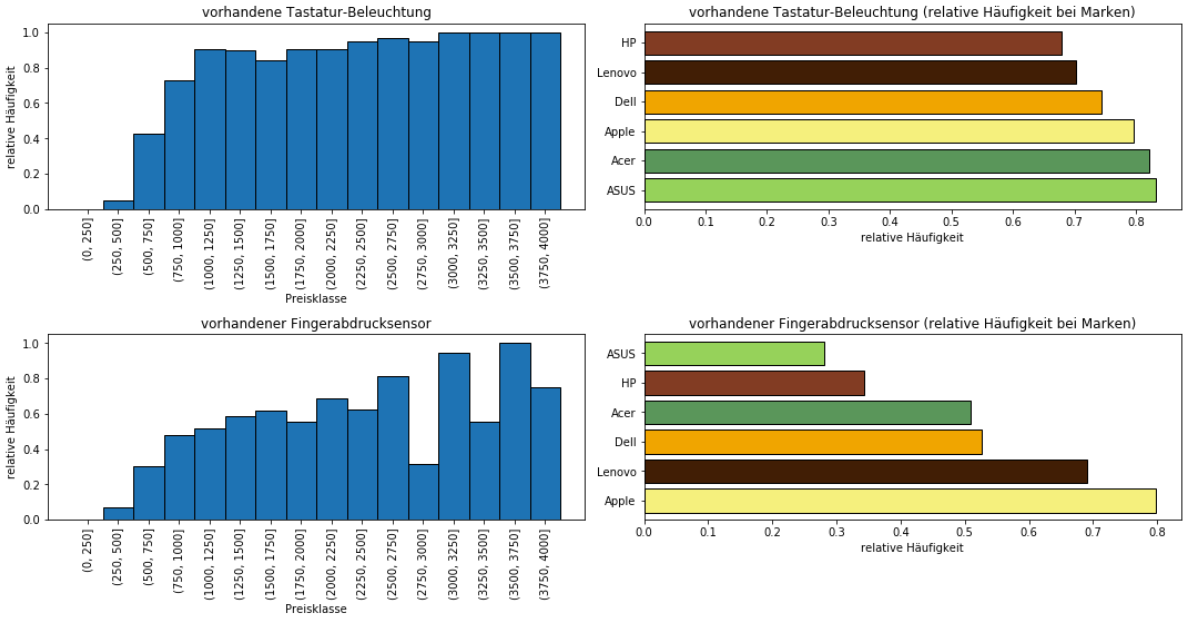
Es wurde ebenfalls eine Grafik erstellt, die Informationen für sämtliche qualitative Merkmale mit zwei Ausprägungen beinhaltet. Da sich allerdings bereits eine Reihe anderer Graphen in der Arbeit befanden und die Arbeit ansonsten mit Graphen überladen worden wäre, wurde diese Art von Graphen nicht in der Arbeit berücksichtigt. Exemplarisch sind untenstehend zwei solcher Graphen abgebildet:

In [921]:
```
def standard_bar(Überschrift, ylim, y_variable1, y_variable2, zeilen=
1, spalten=2, position=1, gegeben=False):
    y = binäre_variablen.groupby("Preisspanne")[y_variable1].mean()

    plot1 = plt.subplot(zeilen, spalten, position)
    plt.title(Überschrift)
    plt.ylabel("relative Häufigkeit")
    plt.xlabel("Preisklasse")
    plt.ylim(0,ylim)
    Graph = plt.bar(Index, y, edgecolor="black", width=Klassenbreite)
    plt.xticks(Index, Kathegorien, rotation = 90)
    if gegeben:
        plt.xticks([])
        plt.xlabel("")

    plot2 = plt.subplot(zeilen, spalten, position+1)
    Ausprägung = "".join(y_variable1.rsplit(y_variable2)).lstrip("_").
replace("_", " ")
    y_nicht_sortiert = df_neu[df_neu[y_variable2]==Ausprägung].groupby
("Marke").size()/df_neu.groupby("Marke").size().dropna()
    y = y_nicht_sortiert.sort_values(ascending=False)
    cm = []
    for Marke in y.index:
        cm.append(df_neu[df_neu.Marke == Marke].Farbe.unique().tolist
()[0])
    farbe = colors.ListedColormap(cm)
    farbe = mp.cm.get_cmap(farbe)(np.linspace(0,1,len(cm)))
    y.plot(kind="barh", title=Überschrift+" (relative Häufigkeit bei M
arken)", width=0.8, edgecolor="black", color=farbe)
    plt.ylabel("")
    plt.xlabel("relative Häufigkeit")
    plt.tight_layout()
```

In [922]:
```
plt.gcf().set_size_inches(15, 30, forward=True)
standard_bar("vorhandene Tastatur-Beleuchtung",1.05,"Tastatur_Beleuchtung_vorhanden", "Tastatur_Beleuchtung",8,2,1, False)
standard_bar("vorhandener Fingerabdrucksensor", 1.05, "Fingerabdrucksensor_vorhanden", "Fingerabdrucksensor",8,2,3, False)
```



Durchschnittliche Preise und Anzahl:

In [923]:
```
for Merkmal in ["Typ_Arbeitsspeicher", "Laufwerk", "Lautsprecher", "Tastatur_Beleuchtung", "Fingerabdrucksensor", "Touchscreen", "Mobilfunk", "NFC", "Akku_Typ","Mikrofon_Sound", "Rabatt_Lehreinrichtung"]:
    for Ausprägung in df_neu[Merkmal].unique():
        print(Merkmal.ljust(22, "-"), Ausprägung.ljust(26, "-"), "Preis:", str(round(df_neu[df_neu[Merkmal] == Ausprägung].Preis.mean(), 2)).ljust(9, "-"), "Anzahl:", df_neu[df_neu[Merkmal] == Ausprägung].Preis.count())
    print()
```

```
Typ_Arbeitsspeicher--- DDR 4-------------------- Preis: 1306.36-- Anz
ahl: 915
Typ_Arbeitsspeicher--- DDR 3-------------------- Preis: 1538.1--- Anz
ahl: 125

Laufwerk-------------- VORHANDEN----------------- Preis: 673.75--- Anz
ahl: 112
Laufwerk-------------- nicht vorhanden----------- Preis: 1413.92-- Anz
ahl: 928

Lautsprecher---------- STANDARD LAUTSPRECHER----- Preis: 1304.13-- Anz
ahl: 984
Lautsprecher---------- ÜBERDURCHSCHN LAUTSPRECHER Preis: 1862.75-- Anz
ahl: 56

Tastatur_Beleuchtung-- nicht vorhanden----------- Preis: 822.82--- Anz
ahl: 278
Tastatur_Beleuchtung-- vorhanden----------------- Preis: 1520.78-- Anz
ahl: 762

Fingerabdrucksensor--- nicht vorhanden----------- Preis: 1132.18-- Anz
ahl: 529
Fingerabdrucksensor--- vorhanden----------------- Preis: 1543.36-- Anz
ahl: 511

Touchscreen----------- nicht vorhanden----------- Preis: 1307.43-- Anz
ahl: 843
Touchscreen----------- vorhanden----------------- Preis: 1448.84-- Anz
ahl: 197

Mobilfunk------------- nicht vorhanden----------- Preis: 1294.85-- Anz
ahl: 956
Mobilfunk------------- vorhanden----------------- Preis: 1782.21-- Anz
ahl: 84

NFC------------------- nicht vorhanden----------- Preis: 1309.31-- Anz
ahl: 1013
NFC------------------- vorhanden----------------- Preis: 2268.63-- Anz
ahl: 27

Akku_Typ-------------- ION---------------------- Preis: 1173.52-- Anz
ahl: 816
Akku_Typ-------------- POLYMER------------------ Preis: 1919.61-- Anz
ahl: 224

Mikrofon_Sound-------- vorhanden----------------- Preis: 1329.09-- Anz
ahl: 853
Mikrofon_Sound-------- nicht vorhanden----------- Preis: 1357.59-- Anz
ahl: 187

Rabatt_Lehreinrichtung nicht vorhanden----------- Preis: 1336.16-- Anz
ahl: 1005
Rabatt_Lehreinrichtung vorhanden----------------- Preis: 1278.43-- Anz
ahl: 35
```

quantitative Merkmale

Zurück zu Deskriptive Statistik

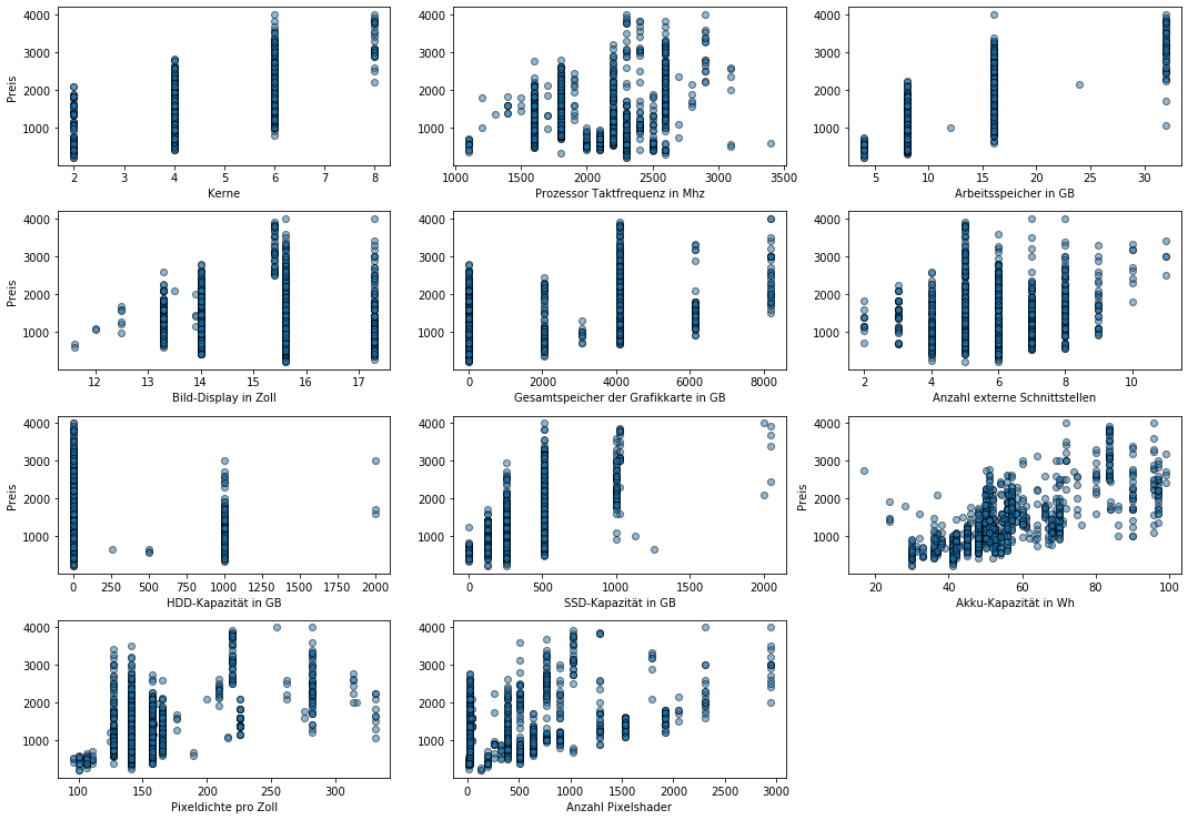
Das Datenset beinhalten 11 quantitative Merkmale

1. Kerne
2. Takt_Prozessor
3. Arbeitsspeicher
4. Bild_Display
5. Speicher_Grafik
6. extern_Schnittstellen
7. HDD
8. SSD
9. Akku_Kapazität
10. PPI_Display
11. Pixelshader

Der untenstehende Graph bietet eine grobe Übersicht über den Zusammenhang zwischen den Preisen und Ausprägungen der Eigenschaften:

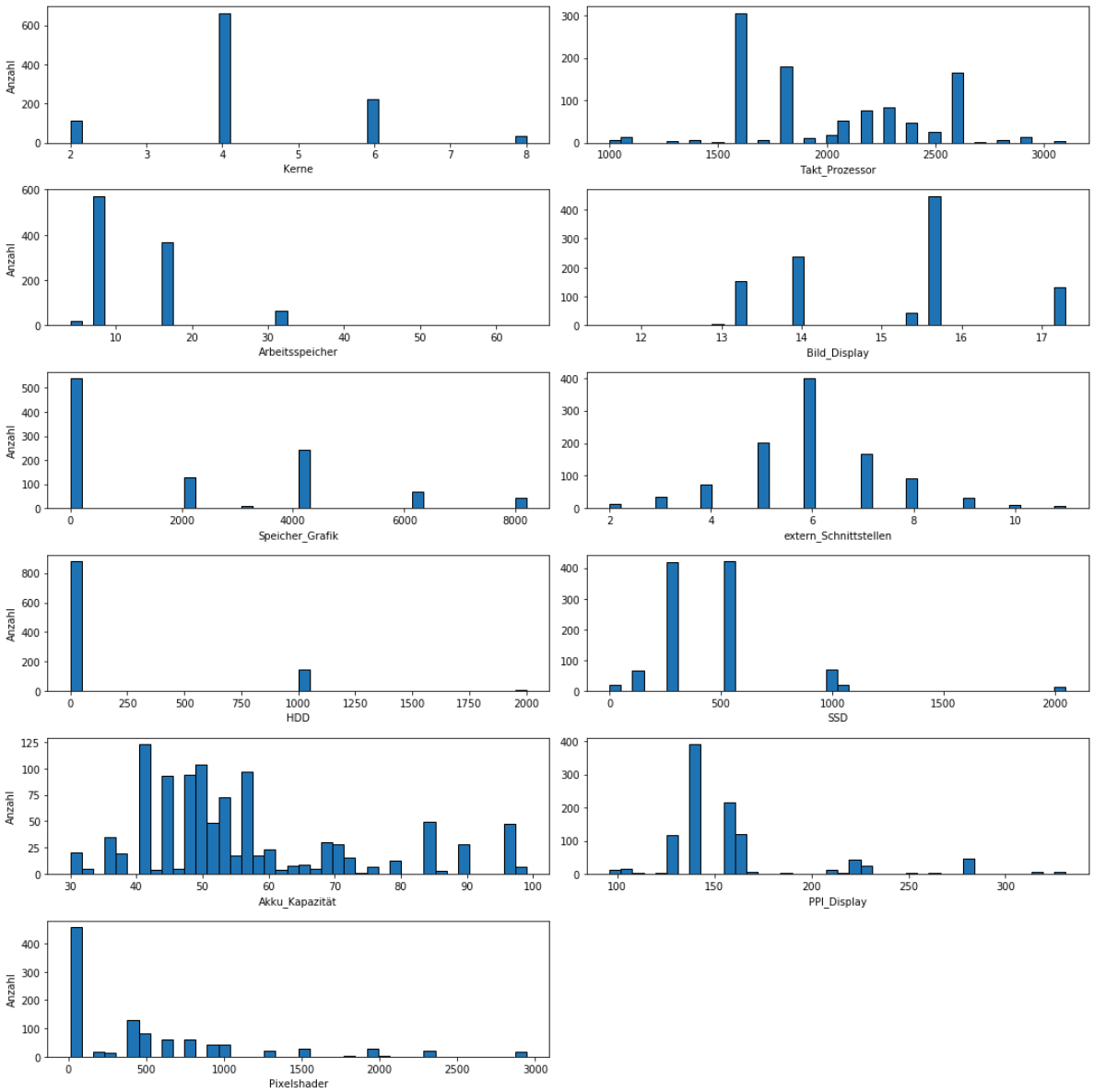
In [924]:
```
def standard_scatter(position, titel, x_variable, y_variable= df_neu.Preis, ylabel= "Preis", x=5, y=3, gegeben=False):
    plt.subplot(x, y, position)
    if gegeben:
        plt.ylabel(ylabel)
    plt.xlabel(titel)
    Graph = plt.scatter(x_variable, y_variable, alpha= 0.5, edgecolors="black")
```

In [925]:
```
plt.gcf().set_size_inches(15, 12.5, forward=True)
standard_scatter(1, "Kerne", df_neu.Kerne, gegeben=True)
standard_scatter(2, "Prozessor Taktfrequenz in Mhz", df_neu.Takt_Prozessor)
standard_scatter(3, "Arbeitsspeicher in GB", df_neu.Arbeitsspeicher)
standard_scatter(4, "Bild-Display in Zoll", df_neu["Bild_Display"], gegeben=True)
standard_scatter(5, "Gesamtspeicher der Grafikkarte in GB", df_neu["Speicher_Grafik"])
standard_scatter(6, "Anzahl externe Schnittstellen", df_neu["extern_Schnittstellen"])
standard_scatter(7, "HDD-Kapazität in GB", df_neu["HDD"], gegeben=True)
standard_scatter(8, "SSD-Kapazität in GB", df_neu["SSD"])
standard_scatter(9, "Akku-Kapazität in Wh", df_neu["Akku_Kapazität"], gegeben=True)
standard_scatter(10, "Pixeldichte pro Zoll", df_neu["PPI_Display"])
standard_scatter(11, "Anzahl Pixelshader", df_neu["Pixelshader"])
plt.tight_layout()
plt.savefig("grobe_übersicht_quantitativ")
```



Häufigkeitesverteilung der quantiativen Merkmale:

In [926]:
```
i=0
quant_red = quantitative_variablen[["Kerne", "Takt_Prozessor", "Arbeitsspeicher", "Bild_Display", "Speicher_Grafik", "extern_Schnittstellen",
                "HDD", "SSD", "Akku_Kapazität", "PPI_Display", "Pixelshader"]]
for Merkmal in quant_red:
    i+=1
    plt.gcf().set_size_inches(15,15, forward=True)
    plt.subplot(6, 2, i)
    plt.xlabel(Merkmal)
    plt.hist(quantitative_variablen[Merkmal], edgecolor="black", bins=40)
    if i%2 != 0:
        plt.ylabel("Anzahl")
    plt.tight_layout()
```



Übersicht nach Komponenten

Zurück zu Deskriptive Statistik

Die Hauptkomponenten der Laptops sind:

1. Speicherkarten
2. Grafikkarte
3. Prozessor
4. Arbeitsspeicher
5. Bildschirm
6. Akku

Speicherkarten

Zurück zu Übersicht nach Komponenten

SSD

In [927]:
```
print("HDD-Anzahl bei Laptops mit zwei SSDs\n", df_neu[df_neu["SSD_Anzahl"]== 2]["HDD_Anzahl"].value_counts())
print("\nHDD-Anzahl bei Laptops mit einer SSD\n",df_neu[df_neu["SSD_Anzahl"] == 1]["HDD_Anzahl"].value_counts())
print("\SSD-Anzahl",df_neu["SSD_Anzahl"].value_counts())
```

```
HDD-Anzahl bei Laptops mit zwei SSDs
 0    8
1    1
Name: HDD_Anzahl, dtype: int64

HDD-Anzahl bei Laptops mit einer SSD
 0    832
1    170
Name: HDD_Anzahl, dtype: int64
\SSD-Anzahl 1    1002
0    29
2    9
Name: SSD_Anzahl, dtype: int64
```

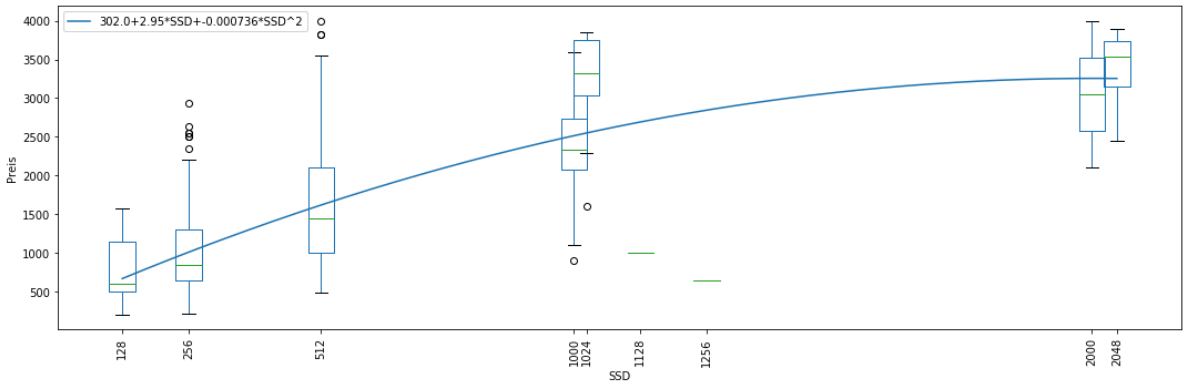
In [928]:
```
def standard_boxplot(regression, axes, label, endog_variable, exog_variablen, Titel="", Balkenbreite=10,
                     Bedingungen=(df_neu.Preis==df_neu.Preis), Klassieren=False, Klassen=False, rotieren=0, yvar = "Preis"):
# unabhängige Variablen werden als Instanz einer Liste gespeichert:
    if not isinstance(exog_variablen, list):
        exog_variablen = [exog_variablen[Bedingungen]]
    else: exog_variablen = [exog_variable[Bedingungen] for exog_variable in exog_variablen]
# Wenn Klassiert werden soll, werden die Abstände aus den Mittelwerten der Klasse berechnet:
    if Klassieren:
        Unterteilung = np.around(np.linspace(exog_variablen[0].min(), exog_variablen[0].max(), Klassen), 1)
        df_neu["Skalierung"] = pd.cut(exog_variablen[0][Bedingungen], Unterteilung)
        Kathegorien = df_neu["Skalierung"].cat.categories
# Mittelpunkte der Klassen:
        Skalierung = df_neu.groupby("Skalierung")[label].mean().tolist()
        plt.xticks(Skalierung, Kathegorien, rotation = 90);
# sind zwei exogene Variablen vorhanden, die untersucht werden sollen, also len(exog_variablen)== 2, wird der zweite Boxplot
# bei einer Distanz von 1.2 Mal der Balkenbreite positioniert:
    else:
        Skalierung = sorted(exog_variablen[0][Bedingungen].unique().tolist());
        if len(exog_variablen) ==2:
            Skalierung = sorted(Skalierung + [Index+1.2*Balkenbreite for Index in Skalierung])
    if regression:
# Für die einzelnen untersuchten Merkmale werden Regressionen durchgeführt:
        for exog_variable in exog_variablen:
            X, y= exog_variable[Bedingungen], endog_variable[Bedingungen];
# Ein quadratischer Term wird miteinbezogen:
            X = pd.concat([X, exog_variable[Bedingungen]**2], axis=1);
# Eine Kostante wird miteinbezogen:
            X = sm.add_constant(X.values);
            fit = sm.OLS(y, X).fit();
            print(fit.rsquared)
            achsenabschn, x1, x2 =str(round(fit.params[0])), str(round(fit.params[1],2)), str(round(fit.params[2],6))
# Die in den Darstellungen abgebildeten Label für die Regeressionskurven werden erstellt:
            graph_label=achsenabschn+"+"+x1+"*"+str(label)+"+"+x2+"*"+str(label)+"^2"
# Die Werte für die die Kurve eingezeichnet werden soll reicht von dem Mittelpunkkt der Klasse mit den kleinsten Werten zu
# dem Mittelpunkt der Klasse mit den größten Werten:
            x1werte = pd.Series(np.arange(Skalierung[0],Skalierung[-1], 0.1))
            regression_xwerte = pd.concat([x1werte, x1werte**2], axis=1)
            regression_xwerte = sm.add_constant(regression_xwerte.values)
```

```
            variable, prediction = regression_xwerte, fit.predict(regr
ession_xwerte)
# Regressionskurve wird dargestellt:
            axes.plot(x1werte, prediction, label=graph_label);
            axes.legend();
#Boxplots werden dargestellt:
    if Klassieren: exog_variablen = df_neu.Skalierung[Bedingungen]
    df_neu[Bedingungen].boxplot(column=yvar, widths=Balkenbreite, by=e
xog_variablen, rot=rotieren,
                          positions=Skalierung, grid=False, ax=axes);
# Wenn die Residuen oder die geschätzen Werte die abhängigen Variablen
sind, werden den Plots andere Eigenschaften zugewiesen,
# die Plots kamen allerdings nicht zur Anwendung:
    if "resid" in yvar:
        axes.axhline(0, color='black', linestyle="--")
    if "fit" in yvar:
        axes.plot([199, 4000], [199, 4000], color = 'black', linestyle
="--")
# Graph wird beschriftet:
    axes.set_xlabel(label)
    axes.set_ylabel(yvar)
    axes.set_title(Titel)
    fig.suptitle("")
    fig.tight_layout()
```

Die oben erstellte Funktion zur Darstellung der Merkmale wird auf die SSD-Karten angewendet:

In [929]:
```
fig, (ax1) = plt.subplots(1,1,figsize=(15,5))
standard_boxplot(True, ax1, "SSD", df_neu.Preis, df_neu.SSD, Bedingung
en=(df_neu.HDD ==0), Balkenbreite=50, rotieren=90)
plt.savefig("SSD-Kapazität")
```

```
0.38328221230921855
```



HDD

In [930]:
```
print("HDD-Kapazität bei Laptops ohne SSD\n", df_neu[df_neu["SSD_Anzahl"]== 0]["HDD"].value_counts())
```

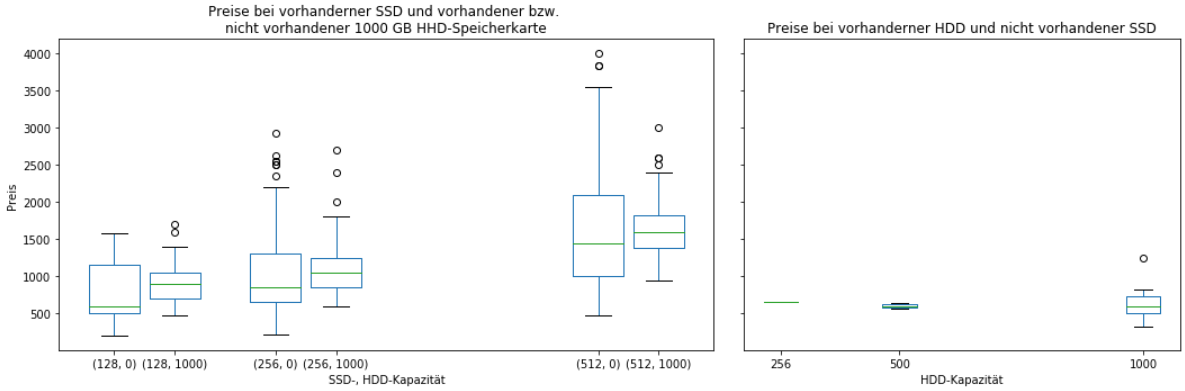
```
HDD-Kapazität bei Laptops ohne SSD
 1000    25
500     3
256     1
Name: HDD, dtype: int64
```

In [931]:
```
print("HDD-Kapazität bei Laptops mit SSD \n", df_neu[df_neu.SSD >0].HDD.value_counts())
```

```
HDD-Kapazität bei Laptops mit SSD
 0       840
1000    168
2000    3
Name: HDD, dtype: int64
```

Die oben erstellte Funktion wird auf die HDD-Festplatten angewendet:

In [932]:
```
fig ,(ax1, ax2) = plt.subplots(1,2, figsize=(15,5),gridspec_kw={'width_ratios': [1.5, 1]}, sharey=True)
standard_boxplot(False, ax1, "SSD-, HDD-Kapazität", df_neu.Preis, [df_neu.SSD, df_neu.HDD],
Titel="Preise bei vorhanderner SSD und vorhandener bzw. \nnicht vorhandener 1000 GB HHD-Speicherkarte", Balkenbreite=40,
Bedingungen= ((df_neu.HDD ==1000) | (df_neu.HDD ==0)) & (df_neu.SSD != 0)&
            (df_neu.SSD <1000))
standard_boxplot(False, ax2, "HDD-Kapazität", df_neu.Preis, df_neu.HDD,
Titel="Preise bei vorhanderner HDD und nicht vorhandener SSD", Balkenbreite=70,
Bedingungen= ((df_neu.HDD !=0) & (df_neu.SSD ==0)))
plt.ylabel("")
plt.savefig("HDD-Kapazität");
```



Auszug aus der Pivot-Tabelle, die die Häufigkeit von HDD und SSD-Kapazität darstellt, als zu zählendes Merkmal wurde der Preis gewählt, allerdings hätte auch jedes andere Merkmal gewählt werden können:

In [933]:
```
pd.pivot_table(df_neu,values="Preis",index=["HDD", "SSD"],aggfunc="count")[0:10]
```

Out[933]:

		Preis
HDD	**SSD**	
0	**128**	27
	256	385
	...	...
	2048	4
256	**0**	1

10 rows × 1 columns

Grafikkarte

Zurück zu Übersicht nach Komponenten

Es gibt zwei Merkmale, die die Grafikkarte beschreiben

1. Speicher
2. Pixelshader

Pixelshader und der Gesamtspeicher der Grafikkarte korrelieren stark miteinander:

In [934]:
```
korr_grafik = df_neu[["Speicher_Grafik", "Pixelshader"]]
korr_grafik.corr()
```

Out[934]:

	Speicher_Grafik	Pixelshader
Speicher_Grafik	1.000000	0.886274
Pixelshader	0.886274	1.000000

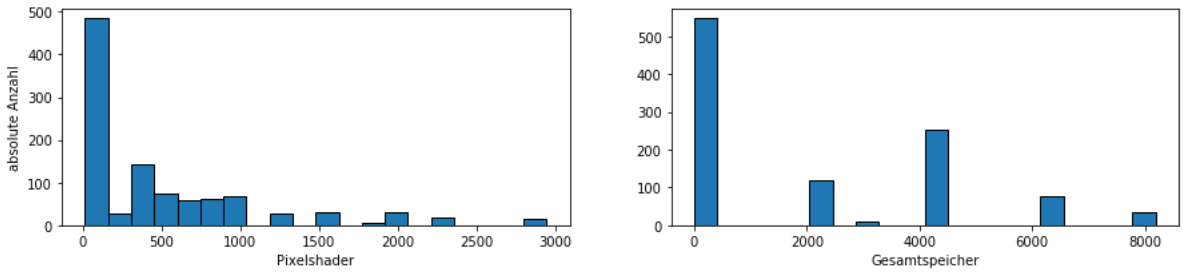
Der Grafikspeicher kann den Wert Null annehmen. Dabei handelt es sich dann um integrierte Grafikkarten

In [935]:
```
print("Minimum Speicher:", df_neu["Speicher_Grafik"].min())
print("Minimum Pixelshader:", df_neu["Pixelshader"].min())
```

```
Minimum Speicher: 0
Minimum Pixelshader: 12.0
```

Häufigkeitsverteilung von Speicher und Pixelshader:

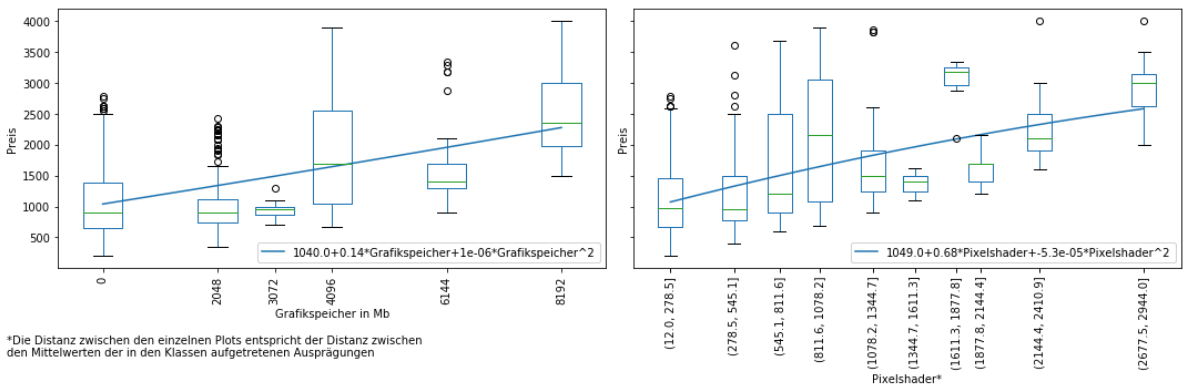
In [936]:
```
plt.gcf().set_size_inches(15, 3, forward=True)
plt.subplot(1,2,1)
plt.ylabel("absolute Anzahl")
plt.xlabel("Pixelshader")
plt.hist(df_neu["Pixelshader"], bins = 20, edgecolor="black");
plt.subplot(1,2,2)
plt.xlabel("Gesamtspeicher")
plt.hist(df_neu["Speicher_Grafik"], bins = 20, edgecolor="black")
plt.savefig("hfkt_Grafikkarte");
```



Es zeigt sich, dass es teilweise unerwartete Abweichungen bei Pixelshader und Grafikspeicher bezüglich des Preises gibt. Die Abweichungen wurden nur knapp in der Arbeit erwähnt, wurden aber dennoch untersucht:

In [937]:
```
fig, (ax1, ax2) = plt.subplots(1,2,figsize=(15,5), sharey=True)
standard_boxplot(True, ax1, "Grafikspeicher", df_neu.Preis, df_neu.Spe
icher_Grafik, rotieren=90,Balkenbreite=700)
ax1.set_xlabel("Grafikspeicher in Mb")
standard_boxplot(True, ax2, "Pixelshader", df_neu.Preis, df_neu.Pixels
hader, rotieren=90, Balkenbreite=150,
                 Klassieren=True, Klassen=12)
ax2.set_xlabel("Pixelshader*")
plt.figtext(0.01, 0.1, "*Die Distanz zwischen den einzelnen Plots ents
pricht der Distanz zwischen\nden Mittelwerten \
der in den Klassen aufgetretenen Ausprägungen", horizontalalignment='l
eft')
plt.savefig("Grafikspeicher_Pixelshader", bbox_inches = "tight");
```

```
0.2271536983110869
0.21500707531185126
```



durchschnittliche Preise für Laptops mit und ohne Touchscreen:

In [938]:
```
print(df_neu[df_neu.Touchscreen == "nicht vorhanden"].Preis.mean())
print(df_neu[df_neu.Touchscreen == "vorhanden"].Preis.mean())
```

```
1307.4257651245553
1448.8426395939086
```

Untersucht man die Laptops mit einem 4096 GB Grafik-Speicher, die von der Norm abweichen, fallen folgende Eigenschaften auf:

Laptops mit 4096 GB Grafik-Speicher sind durchschnittlich etwa 14 mal häufiger Apple-Laptops als im Durchschnitt aller anderen Grafik-Speicher-Formate. Die Anzahl wie viele Observationen die anderen Speicher-Formate aufweisen wurde allerdings hier nicht berücksichtigt:

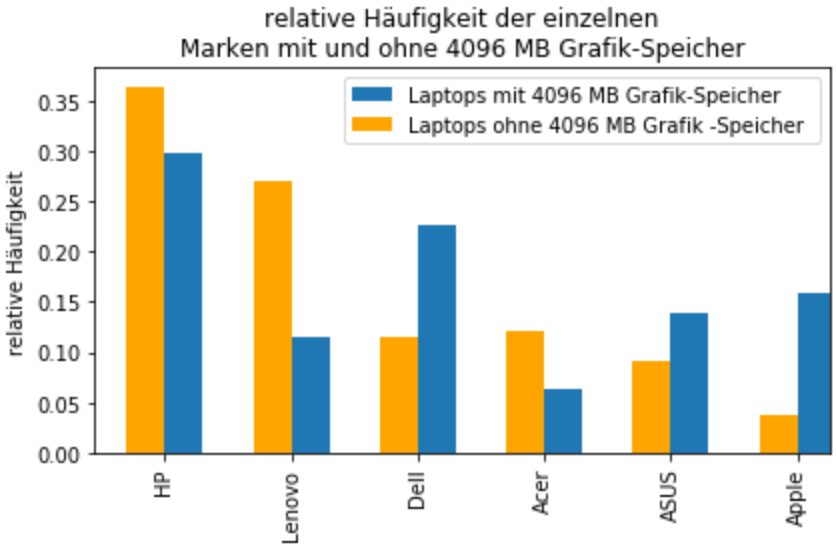
In [939]:
```
tabelle0 = pd.concat([binäre_variablen, df_neu["Speicher_Grafik"]], ax
is=1)
tabelle1 = pd.pivot_table(tabelle0, index="Speicher_Grafik")
tabelle1 = pd.pivot_table(tabelle0[tabelle0.Speicher_Grafik != 4096],
index="Speicher_Grafik").mean()
tabelle2 = pd.pivot_table(tabelle0[tabelle0.Speicher_Grafik == 4096],
index="Speicher_Grafik")
ergebnis = ((tabelle1-tabelle2)/tabelle1).sort_values(by= 4096, axis=
1, ascending=False)
ergebnis_spalten = abs((tabelle1-tabelle2)/tabelle1).sort_values(by= 4
096, axis=1, ascending=False).columns
ergebnis = ergebnis[ergebnis_spalten]
ergebnis.iloc[:, : 4]
```

Out[939]:

Speicher_Grafik	Betriebssystem_MAC	Marke_Apple	Lautsprecher_ÜBERDURCHSCHN_LAUTSPREC
4096	-13.969896	-13.969896	-3.08

wird berücksichtigt, wie häufig die einzelnen Ausprägungen des Grafik-Speichers auftreten ergibt sich folgendes Bild:

In [940]:
```
Marken = df_neu.Marke.value_counts().index.tolist()
fig, ax1 = plt.subplots()
df_neu[df_neu["Speicher_Grafik"] == 4096]["Marke"].value_counts().divi
de(len(df_neu[df_neu["Speicher_Grafik"] ==\
4096])).reindex(index = Marken).plot(kind='bar', ax= ax1, label="Lapto
ps mit 4096 MB Grafik-Speicher", width = 0.3, position=0)
df_neu[df_neu["Speicher_Grafik"] != 4096]["Marke"].value_counts().divi
de(len(df_neu[df_neu["Speicher_Grafik"] !=\
4096])).reindex(index = Marken).plot(kind='bar', ax= ax1, color="orang
e", label="Laptops ohne 4096 MB Grafik -Speicher ",
width=0.3, position=1)
ax1.legend(loc="upper right")
ax1.set_ylabel("relative Häufigkeit")
ax1.set_title("relative Häufigkeit der einzelnen\nMarken mit und ohne
4096 MB Grafik-Speicher")
fig.tight_layout()
plt.savefig("hfkt_4096_grafik_speicher");
```



Die überwiegende Anzahl an Apple Laptops besitzt einen 4096 MB fassenden Grafikspeicher:

In [941]:
```
df_neu[df_neu.Marke == "Apple"]["Speicher_Grafik"].value_counts()
```

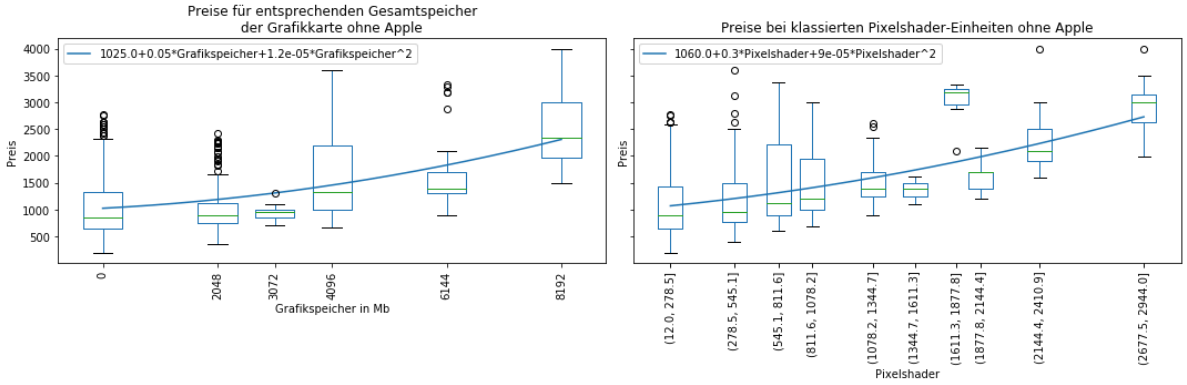
Out[941]:
```
4096    40
0       29
Name: Speicher_Grafik, dtype: int64
```

Ohne Apple-Laptops ergibt sich somit folgendes Bild:

In [942]:
```
fig, (ax1, ax2) = plt.subplots(1,2,figsize=(15,5), sharey=True)
standard_boxplot(True, ax1, "Grafikspeicher", df_neu.Preis, df_neu.Spe
icher_Grafik,Bedingungen=(df_neu.Marke!="Apple"),
Titel="Preise für entsprechenden Gesamtspeicher\nder Grafikkarte ohne
Apple", Balkenbreite=700, rotieren=90)
ax1.set_xlabel("Grafikspeicher in Mb")
standard_boxplot(True, ax2, "Pixelshader", df_neu.Preis, df_neu.Pixels
hader, Bedingungen=(df_neu.Marke!="Apple"),
Titel="Preise bei klassierten Pixelshader-Einheiten ohne Apple", Klass
ieren=True, Klassen=12, Balkenbreite=150, rotieren=90)

plt.savefig("Grafikspeicher_Pixelshader_ohne_Apple", bbox_inches = "ti
ght");
```

```
0.23579127362419605
0.21506769535765147
```



Prozessor

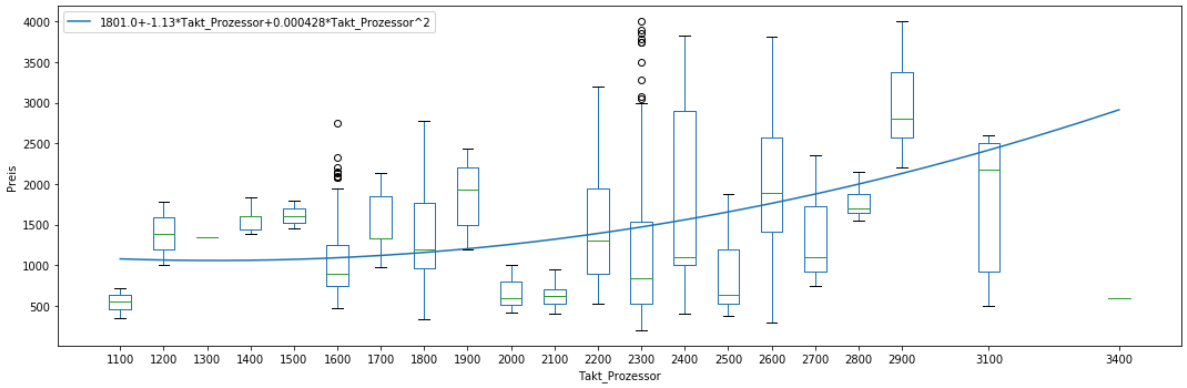
Zurück zu Übersicht nach Komponenten

Merkmale des Prozessors:

1. Kerne
2. Prozessor Takt

In [943]:
```
fig, ax1 = plt.subplots(1,1,figsize=(15,5))
standard_boxplot(True, ax1, "Takt_Prozessor", df_neu.Preis, df_neu.Takt_Prozessor , Balkenbreite=50)
plt.savefig("Prozessor-Takt");
```

```
0.13716936518288747
```

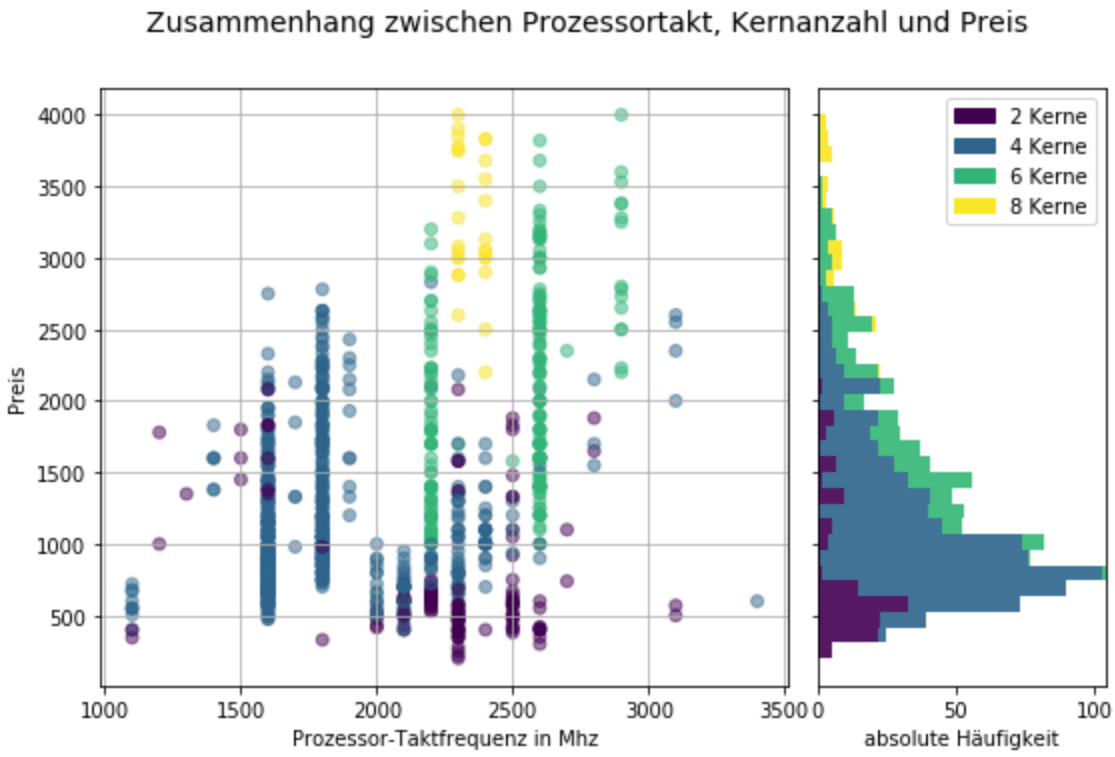


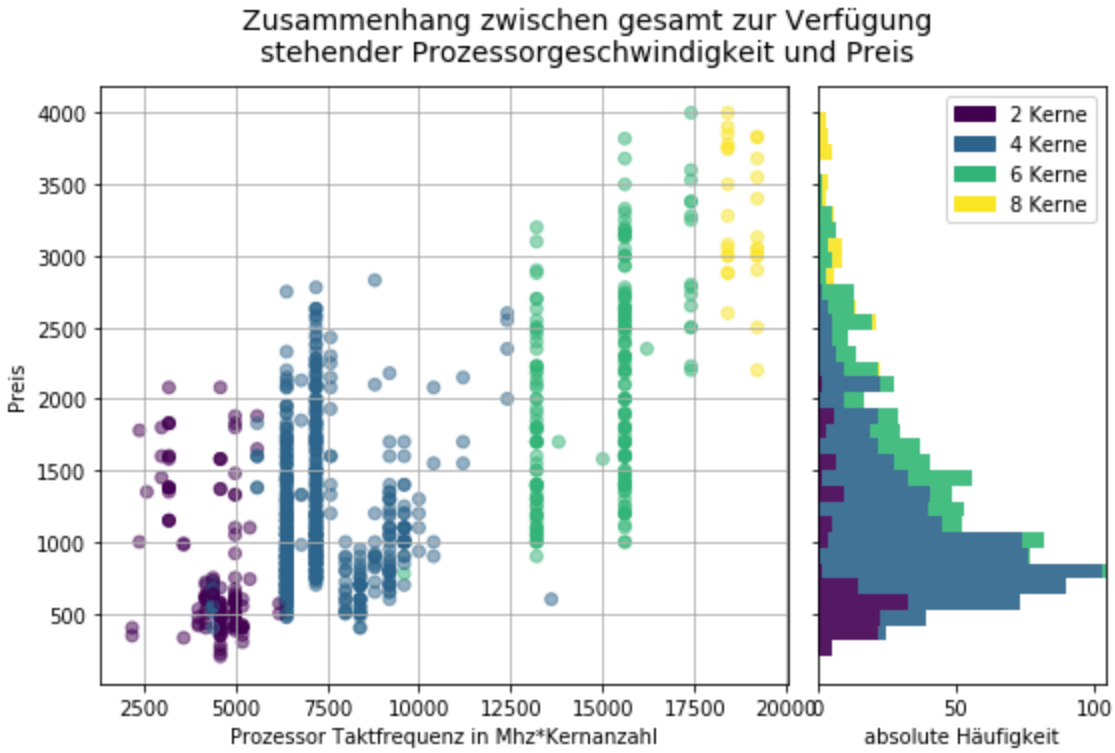
In [944]:
```
def übers(x_variable, x_label, titel):
    Farben = mp.cm.get_cmap('viridis')(np.linspace(0,1,4))
    plt.viridis()
    fig, axScatter = plt.subplots(figsize=(9, 5.5));
    fig.suptitle(titel, fontsize=14);
    axScatter.scatter(x_variable, df_neu.Preis, alpha = 0.5, c=df_neu.Kerne);
    axScatter.grid(b=True);
    plt.ylabel("Preis")
    plt.xlabel(x_label)
    divider = make_axes_locatable(axScatter);
    axHisty = divider.append_axes("right", 2, pad=0.2, sharey=axScatter);
    axHisty.yaxis.set_tick_params(labelleft=False,)
    axHisty.hist([df_neu.Preis[df_neu.Kerne == Kerne] for Kerne  in sorted(df_neu["Kerne"].unique().tolist())],
                 orientation='horizontal', bins=35, alpha= 0.9, color=Farben, stacked="True");
    plt.legend(handles=[mpatches.Patch(color=Farben[i],
    label="{} Kerne".format(x)) for i,x in enumerate(sorted(df_neu["Kerne"].unique().tolist()))]);
    plt.xlabel("absolute Häufigkeit");
```

Die folgende Übersicht stellt den Zusammenhang zwischen Prozessor-Kernen, Taktfrequenz und dem Preis dar:

In [945]:
```
übers(df_neu.Takt_Prozessor, "Prozessor-Taktfrequenz in Mhz", "Zusammenhang zwischen Prozessortakt, Kernanzahl und Preis");
übers(df_neu.Takt_Prozessor*df_neu.Kerne, "Prozessor Taktfrequenz in Mhz*Kernanzahl",
      "Zusammenhang zwischen gesamt zur Verfügung\nstehender Prozessor geschwindigkeit und Preis" );
```

```
<Figure size 432x288 with 0 Axes>
```



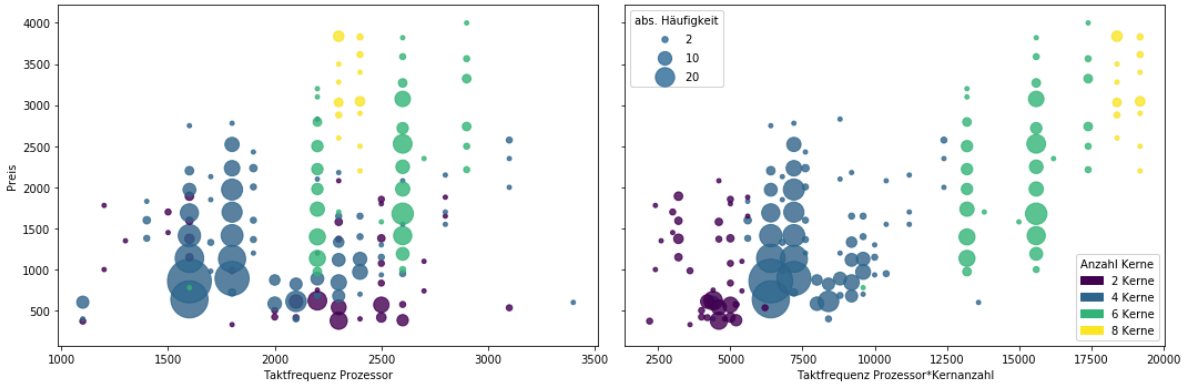


Da die obenstehdende Abbildung nur bedingt informativ ist, wurde eine weitere Abbildung erstellt, die die Größe der Punkte als absolute Häufigkeit darstellt:

In [946]:
```
Klassieren(1000,3500,30, "Takt_Prozessor_Unterteilung", "Takt_Prozessor")
Klassieren(199,3999,15, "Preisspanne", "Preis")
scatter_prozessor = df_neu.groupby(["Takt_Prozessor_Unterteilung", "Preisspanne", "Kerne"])["Takt_Prozessor"].mean().to_frame("durchschn_Takt").reset_index()
scatter_preis = df_neu.groupby(["Takt_Prozessor_Unterteilung", "Preisspanne", "Kerne"])["Preis"].mean().to_frame("durchschn_Preis").reset_index()
scatter_anzahl = df_neu.groupby(["Takt_Prozessor_Unterteilung", "Preisspanne", "Kerne"]).size().to_frame("Anzahl").reset_index()
scatter_var = pd.concat([scatter_anzahl, scatter_preis, scatter_prozessor], axis=1)
scatter_var = scatter_var.T.drop_duplicates().T.apply(pd.to_numeric, errors='ignore')
del scatter_prozessor, scatter_preis, scatter_anzahl
```

In [947]:
```
def Klassieren(kleinster_wert, größter_wert, klassen, name, klasse_zu_teilen):
    Unterteilung = np.linspace(kleinster_wert, größter_wert, klassen).astype(int)
    df_neu[name] = pd.cut(df_neu[klasse_zu_teilen], Unterteilung)
```

In [948]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,5), sharey= True)
ax1.scatter(scatter_var.durchschn_Takt, scatter_var.durchschn_Preis, s
=scatter_var.Anzahl*15, c=scatter_var.Kerne, alpha=0.8)
ax1.set_xlabel("Taktfrequenz Prozessor")
ax1.set_ylabel("Preis");
scatter = ax2.scatter(scatter_var.durchschn_Takt*scatter_var.Kerne, sc
atter_var.durchschn_Preis, s=scatter_var.Anzahl*15,
            c=scatter_var.Kerne, alpha=0.8)
kw = dict(prop="sizes", num=[2, 10, 20], color="#306998", func=lambda
s: s/15)
legende1 = ax2.legend(*scatter.legend_elements(**kw),loc="best", title
="abs. Häufigkeit")
Farben = mp.cm.get_cmap('viridis')(np.linspace(0,1,4))
legende2 = ax2.legend(title= "Anzahl Kerne", loc="lower right", handle
s=[mpatches.Patch(color=Farben[i],
label="{} Kerne".format(x)) for i,x in enumerate(sorted(df_neu["Kern
e"].unique().tolist()))]);
ax2.add_artist(legende1)
ax2.set_xlabel("Taktfrequenz Prozessor*Kernanzahl")
plt.tight_layout()
plt.savefig("Kerntakt_Kerne");
```



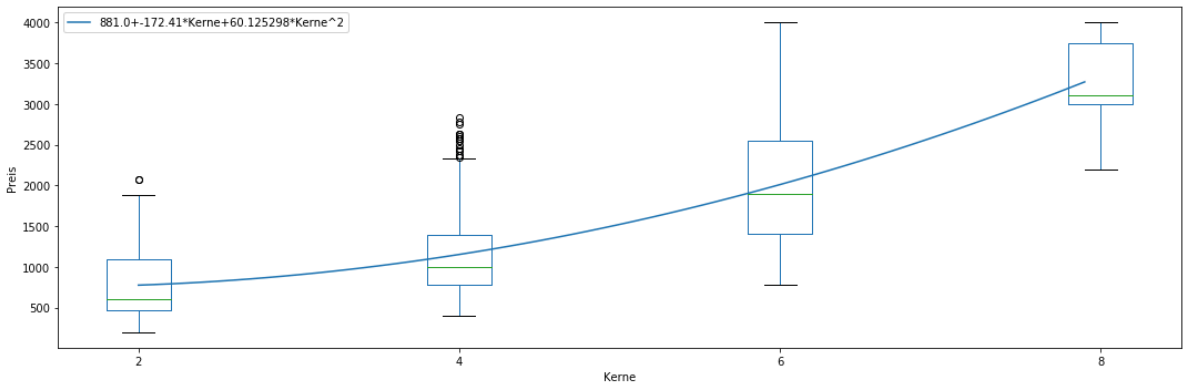
Wie auch in der Grafik, zeigt sich, dass Prozessoren mit zwei Kernen eine durchschnittlich höhere Taktfrequenz aufweisen, als Laptops mit vier Kernen:

In [949]:
```
print(df_neu[df_neu.Kerne == 2].Takt_Prozessor.mean())
print(df_neu[df_neu.Kerne == 4].Takt_Prozessor.mean())
```

```
2202.127659574468
1808.623298033283
```

In [950]:
```
fig, ax1 = plt.subplots(1,1,figsize=(15,5))
standard_boxplot(True, ax1, "Kerne", df_neu.Preis, df_neu.Kerne , Balk
enbreite=0.4)
plt.tight_layout()
plt.savefig("Kerne");
```

```
0.4714208695602562
```



Arbeitsspeicher

Zurück zu Übersicht nach Komponenten

Merkmale des Arbeitsspeichers:

1. Typ (DDR3/DDR4)
2. Speicher

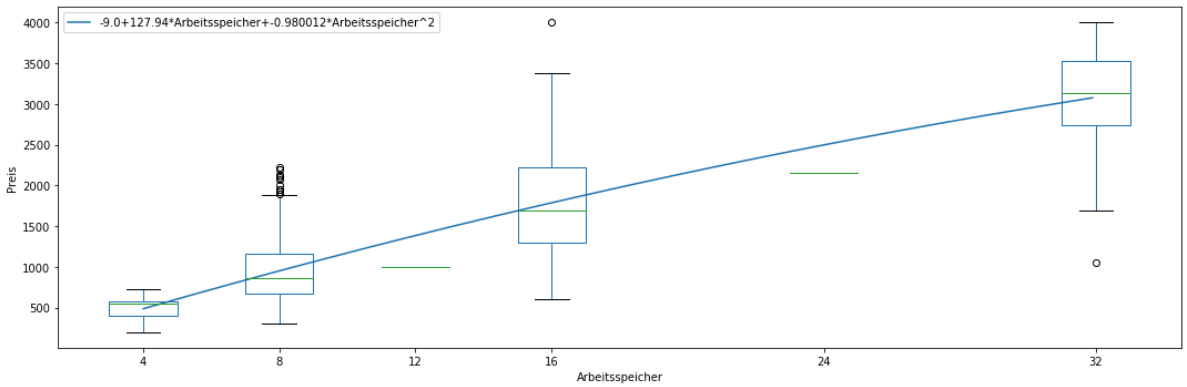
Der Typ des Arbeitsspeichers und die Kapazität des Arbeitsspeichers korrelieren kaum miteinander:

In [951]:
```
binäre_variablen.Typ_Arbeitsspeicher_DDR_4.corr(df_neu.Arbeitsspeiche
r)
```

Out[951]:
```
0.04389714028870925
```

In [952]:
```
fig,ax1 = plt.subplots(1,1, figsize=(15,5))
standard_boxplot(True, ax1, "Arbeitsspeicher", df_neu.Preis, [df_neu.A
rbeitsspeicher], Balkenbreite=2)
plt.tight_layout()
plt.savefig("arbeitsspeicher");
```

```
0.5832796326902863
```



Durchschnittliche Preise ausgewählter Laptops für bestimmte Kapazitäten an Arbeitsspeicher:

In [953]:
```
print(df_neu[df_neu.Arbeitsspeicher == 4].Preis.median())
print(df_neu[df_neu.Arbeitsspeicher == 8].Preis.median())
print(df_neu[df_neu.Arbeitsspeicher == 16].Preis.median())
```

```
549.0
869.0
1699.0
```

Bildschirm

Zurück zu Übersicht nach Komponenten

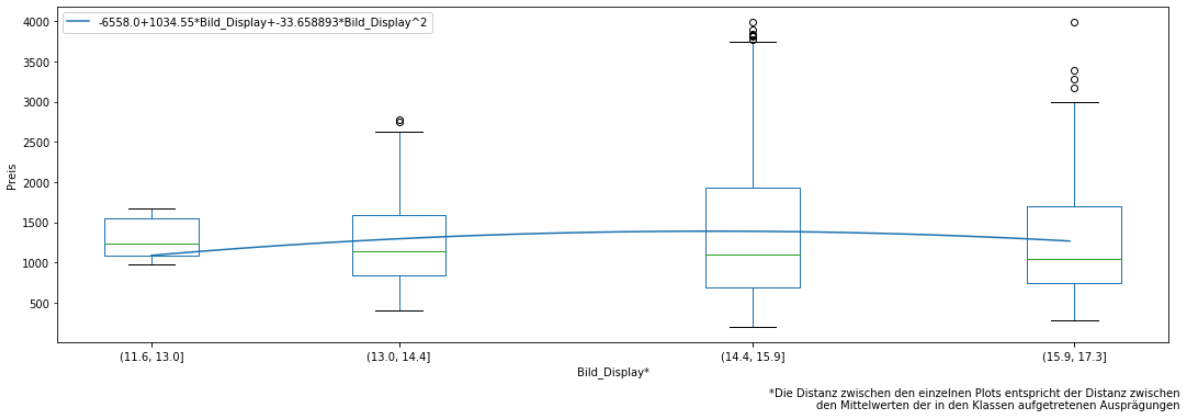
Merkmale des Bildschirms:

1. Pixeldichte
2. Größe
3. Touchscreen

In [954]:
```
fig, (ax1) = plt.subplots(1,1, sharey=True, figsize=(15,5))
standard_boxplot(True, ax1, "Bild_Display", df_neu.Preis, df_neu.Bild_
Display, Klassieren=True, Klassen=5, Balkenbreite=0.5)

plt.figtext(1, -.05, "*Die Distanz zwischen den einzelnen Plots entspr
icht der Distanz zwischen\nden Mittelwerten \
der in den Klassen aufgetretenen Ausprägungen", horizontalalignment='r
ight')
ax1.set_xlabel("Bild_Display*")
plt.tight_layout()
plt.savefig("display_größe", bbox_inches = "tight");
```

```
0.007398598463080663
```



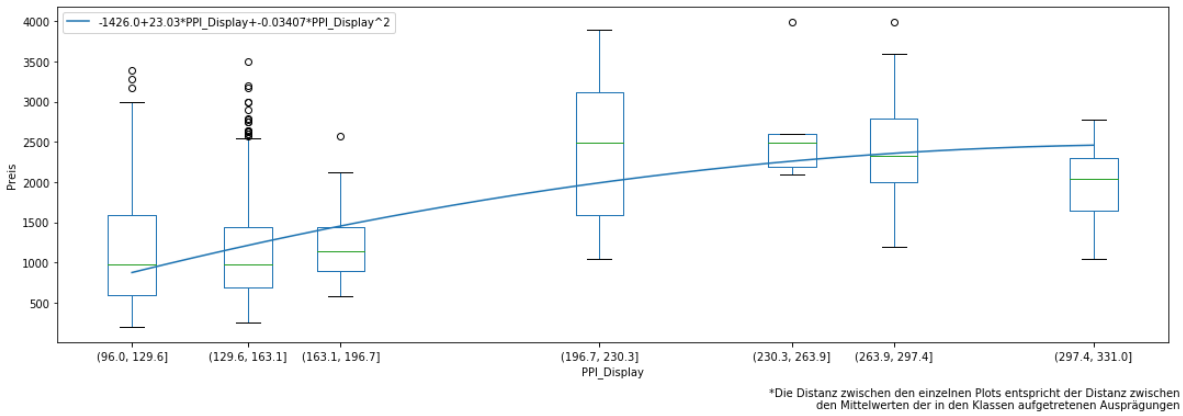
In [955]:
```
print("durchschnittliche Bildschirm-Größe:",df_neu.Bild_Display.mean
())
print("durchschnittlicher Preis für Laptops über durchschn. Bildschirm
-Größe:",df_neu[df_neu.Bild_Display>=15].Preis.mean())
print("durchschnittlicher Preis für Laptops über durchschn. Bildschirm
-Größe:",df_neu[df_neu.Bild_Display<15].Preis.mean())
```

```
durchschnittliche Bildschirm-Größe: 15.073557692307721
durchschnittlicher Preis für Laptops über durchschn. Bildschirm-Größe:
1381.8342211838008
durchschnittlicher Preis für Laptops über durchschn. Bildschirm-Größe:
1257.397864321608
```

In [956]:
```
fig, ax1 = plt.subplots(1,1,figsize=(15,5))
standard_boxplot(True, ax1, "PPI_Display", df_neu.Preis, df_neu.PPI_Di
splay, Klassieren=True, Klassen=8, Balkenbreite=10)

plt.figtext(1, -.05, "*Die Distanz zwischen den einzelnen Plots entspr
icht der Distanz zwischen\nden Mittelwerten \
der in den Klassen aufgetretenen Ausprägungen", horizontalalignment='r
ight')
plt.tight_layout()
plt.savefig("pixeldichte", bbox_inches = "tight");
```

```
0.2904569914830817
```



Akku

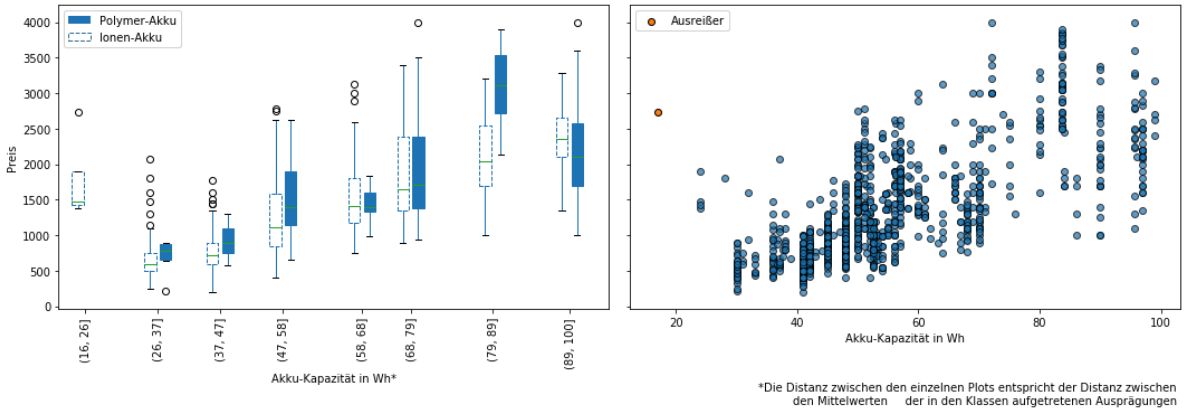
Zurück zu Übersicht nach Komponenten

Merkmale des Akkus:

1. Speichertyp
2. Akku-Kapazität

für den Akku wird eine eigene Funktion erstellt:

In [957]:
```
def Akku():
    Klassieren(16, 100, 9, "Akku_Unterteilung", "Akku_Kapazität")
    skalierung = df_neu.groupby(["Akku_Unterteilung"])["Akku_Kapazität"].mean().tolist()
    skalierung = [Mittelwert/skalierung[0] for Mittelwert in skalierung]
    fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,5), sharey= True)
    boxprops = dict(linestyle='--')
    bp1 = df_neu[df_neu.Akku_Typ=="ION"].boxplot(column="Preis", by=["Akku_Unterteilung", "Akku_Typ"],ax=ax1,
    positions=[x*4 for x in skalierung], boxprops=boxprops, rot= 90, widths=(0.3), grid=False);
    bp2 = df_neu[df_neu.Akku_Typ=="POLYMER"].boxplot(column="Preis", by=["Akku_Unterteilung"],ax=ax1,
    patch_artist=True,positions=[x*4+0.4 for x in skalierung], rot= 90, widths=(0.3), grid=False, manage_ticks=False);
    Ionen_patch = mpatches.Patch(label="Polymer-Akku")
    poly_patch = mpatches.Patch(label="Ionen-Akku", linestyle="--", facecolor="white", edgecolor="#306998")
    ax1.legend(handles=[Ionen_patch, poly_patch])
    ax1.set_xticks(ticks=[x*4+0.2 for x in skalierung])
    ax1.set_xlabel("Akku-Kapazität in Wh*")
    ax1.set_ylabel("Preis")
    ax1.set_title("")
    fig.suptitle("");
    plt.figtext(0.98, -0.02, "*Die Distanz zwischen den einzelnen Plots entspricht der Distanz zwischen\nden Mittelwerten \
    der in den Klassen aufgetretenen Ausprägungen", horizontalalignment='right')
    plt.tight_layout()
    ax2.scatter(df_neu[~df_neu.index.isin([650])].Akku_Kapazität, df_neu[~df_neu.index.isin([650])].Preis,
                edgecolor="black", alpha=0.7)
    ax2.scatter(df_neu.loc[ 650 , "Akku_Kapazität"], df_neu.loc[ 650 , "Preis"], edgecolor="black",
                label="Ausreißer")
    ax2.legend()
    ax2.set_xlabel("Akku-Kapazität in Wh")
Akku()
plt.savefig("akku", bbox_inches = "tight");
```



Löschen des Ausreißers:

Zurück zu Deskriptive Statistik

In [958]:
```
df_neu.loc[ 650 , "Akku_Kapazität"]
```

Out[958]: 17.0

Die Anzahl an Observationen reduziert sich also auf 1039:

In [959]:
```
df_neu = df_neu[~df_neu.index.isin([650])]
binäre_variablen = binäre_variablen[~binäre_variablen.index.isin([650])]
len(df_neu)
```

Out[959]: 1039

Löschen jeweils einer Dummy-Variable:

In [960]:
```
binäre_variablen = binäre_variablen.drop(["Preis", "Preisspanne", "Typ_Arbeitsspeicher_DDR_3", "Laufwerk_nicht_vorhanden",
                          "Lautsprecher_STANDARD_LAUTSPRECHER", "Tastatur_Beleuchtung_nicht_vorhanden",
                         "Fingerabdrucksensor_nicht_vorhanden", "Touchscreen_nicht_vorhanden",
                         "Mobilfunk_nicht_vorhanden", "NFC_nicht_vorhanden", "Betriebssystem_WINDOWS", "Betriebssystem_MAC",
                         "Akku_Typ_POLYMER", "Marke_HP",
                          "Mikrofon_Sound_nicht_vorhanden", "Rabatt_Lehreinrichtung_nicht_vorhanden"], axis=1)
```

Erstellen quantitativer Merkmale:

In [961]:
```
quantitative_variablen = df_neu.drop(["Typ_Arbeitsspeicher", "Laufwerk", "Lautsprecher", "Tastatur_Beleuchtung",
                         "Fingerabdrucksensor", "Touchscreen", "Mobilfunk", "NFC", "Preisspanne",
                         "Akku_Typ", "Betriebssystem", "Mikrofon_Sound", "Rabatt_Lehreinrichtung",
                        "HDD_Anzahl", "SSD_Anzahl", "Marke", "Farbe", "Skalierung",
                        "Takt_Prozessor_Unterteilung", "Akku_Unterteilung"], axis=1)
```

Zusammenfügen quantitativer und qualtitativer Merkmale:

In [962]:
```
x = pd.concat([quantitative_variablen, binäre_variablen], axis="columns")
x.to_csv(r'C:\Users\tobia\Desktop\Uni\Marburg\Bachelorarbeit\Alternate\fertiges_datenset.csv')
x = x.drop(["URL"], axis=1)
quantitative_variablen = quantitative_variablen.drop(["URL", "Preis"], axis=1)
```

In [963]:
```
x
```

Out[963]:

Laptop_ID	Preis	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	Speicher_Grafik	extern_
2	299.0	2	2600	8	15.6	0	
4	899.0	6	2200	8	15.6	6144	
...	...	...	...	...	...	...	
1747	949.0	4	2300	8	15.6	4096	
1748	1299.0	4	2300	16	15.6	6144	

1039 rows × 31 columns

Induktive Statistik

Zurück zur Gliederung

1. Generelle Kollinearität
2. Erstellen Transformierter Werte
3. Modelle (Modellübersicht)
4. Residuen-Plots
5. Kreuzvalidierung
6. Modellsauswahl und Darstellung
7. Weitere in der Arbeit nicht berücksichtigte Modelle

Laden des oben erstellen, fertigen Datensatzes:

In [1300]:
```
x = pd.read_csv(r'C:\Users\tobia\Desktop\Uni\Marburg\Bachelorarbeit\Alternate\fertiges_datenset.csv')
x.set_index("Laptop_ID", inplace=True)
x = x.drop(["URL"], axis=1)
```

Die ersten zwei Observationen:

In [1301]:
```
x.head(2)
```

Out[1301]:

Laptop_ID	Preis	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	Speicher_Grafik	extern_S
2	299.0	2	2600	8	15.6	0	
4	899.0	6	2200	8	15.6	6144	

Der Preis gehört nicht zu den Prädiktoren und wird als separater Datensatz behandelt, zudem werden logarithmische Werte des Preises erstellt:

In [1302]:
```
y = pd.DataFrame()
y["Preis"] = x.pop("Preis")
y["Log_Preis"] = np.log(y.Preis)
```

Überprüfung möglicher Multikollinearität - generell:

Zurück zu Induktive Statistik

Die drei am stärksten miteinander korrelierenden Merkmale:

In [1303]:
```
c = x.corr()
c =c.mask(np.tril(np.ones(c.shape)).astype(np.bool)).abs()
corr = c.unstack()
corr = pd.DataFrame(corr.sort_values(kind="quicksort")).reset_index()
corr1 = corr[corr.level_0 != corr.level_1].sort_values(by=[0], ascendi
ng=False).reset_index().drop("index", axis=1).dropna()
corr1.head(3)
```

Out[1303]:

	level_0	level_1	0
0	Pixelshader	Speicher_Grafik	0.886291
1	Speicher_Grafik	Kerne	0.670456
2	Akku_Kapazität	Kerne	0.660291

Ein Wert über 10 für den Varianz-Inflations-Faktor gilt als kritisch. Hier haben alle Faktoren einen Wert unter 10:

In [1304]:
```
def vif(var):
    from statsmodels.stats.outliers_influence import variance_inflatio
n_factor
    vif = pd.DataFrame()
    vif["VIF"] = [variance_inflation_factor(var.values, i) for i in ra
nge(var.shape[1])]
    vif["Merkmale"] = var.columns
    vif = vif[vif.Merkmale != "Konstante"]
    return(vif.sort_values(by=["VIF"], ascending=False).reset_index(dr
op=True))
```

In [1305]:
```
vif(x)
```

Out[1305]:

	VIF	Merkmale
0	122.188786	Bild_Display
1	53.908087	Takt_Prozessor
...	...	...
28	1.137903	Betriebssystem_DOS
29	1.061592	Betriebssystem_LINUX

30 rows × 2 columns

Transformierte Werte

Zurück zu Induktive Statistik

In [1306]:
```
x["Konstante"] = 1
```

In [1307]:
```
quantitativ = x[['Kerne', 'Takt_Prozessor',
           'Arbeitsspeicher',
           'Bild_Display',
           'Speicher_Grafik',
           'extern_Schnittstellen',
           'HDD', 'SSD',
           'Akku_Kapazität',
           'PPI_Display',
           'Pixelshader']]
```

In [1308]:
```
qualitativ = x[['Laufwerk_VORHANDEN','Tastatur_Beleuchtung_vorhanden',
'Fingerabdrucksensor_vorhanden',
           'Touchscreen_vorhanden', 'Mobilfunk_vorhanden', 'NFC_vorhan
den', 'Mikrofon_Sound_vorhanden',
           'Rabatt_Lehreinrichtung_vorhanden','Akku_Typ_ION',
           'Marke_ASUS', 'Marke_Acer', 'Marke_Apple', 'Marke_Dell','Ma
rke_Lenovo',
           'Betriebssystem_DOS', 'Betriebssystem_LINUX','Betriebssyste
m_OHNE_BETRIEBSSYSTEM',
           'Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER', 'Typ_Arbeitsspei
cher_DDR_4', "Konstante"]]
```

In [1309]:
```
def x_var_erstellen(quant, qual):
    return(pd.concat([quant, qual], axis="columns"))
```

Funktionen um Merkmale gegebenfalls standardisieren und zentrieren zu können:

In [1310]:
```
def standardisieren(var):
    varn= (var-var.mean())/var.std()
    return(varn)
def zentrieren(var):
    varn =(var-var.mean())
    return(varn)
```

Erstellen quadratischer Werte:

In [1311]:
```
x["Kerne_quad"] = x.Kerne**2
x["Takt_Prozessor_quad"] = x.Takt_Prozessor**2
x["Arbeitsspeicher_quad"] = x.Arbeitsspeicher**2
x["Bild_Display_quad"] = x.Bild_Display**2
x["Speicher_Grafik_quad"] = x.Speicher_Grafik**2
x["extern_Schnittstellen_quad"] = x.extern_Schnittstellen**2
x["HDD_quad"] = x.HDD**2
x["SSD_quad"] = x.SSD**2
x["Akku_Kapazität_quad"] = x.Akku_Kapazität**2
x["PPI_Display_quad"] =x.PPI_Display**2
x["Pixelshader_quad"] =x.Pixelshader**2
```

Erstellen von Logarithmen:

In [1312]:
```
x["log_Kerne"] =np.log(x.Kerne)
x["log_Takt_Prozessor"] =np.log(x.Takt_Prozessor)
x["log_Arbeitsspeicher"] = np.log(x.Arbeitsspeicher)
x["log_Bild_Display"] =np.log(x.Bild_Display)
x["log_Speicher_Grafik"] = np.log(x.Speicher_Grafik+1)
x["log_extern_Schnittstellen"] = np.log(x.extern_Schnittstellen)
x["log_HDD"] = np.log(x.HDD+1)
x["log_SSD"] =np.log(x.SSD+1)
x["log_Akku_Kapazität"] =np.log(x.Akku_Kapazität)
x["log_PPI_Display"] =np.log(x.PPI_Display)
x["log_Pixelshader"] =np.log(x.Pixelshader)
```

Erstellen zentrierter Werte:

In [1313]:
```
x["Kerne_z"] = x.Kerne-x.Kerne.mean()
x["Kerne_z_quad"] = x.Kerne_z**2
x["Takt_Prozessor_z"] = x.Takt_Prozessor-x.Takt_Prozessor.mean()
x["Takt_Prozessor_z_quad"] = x.Takt_Prozessor_z**2
x["Arbeitsspeicher_z"] = x.Arbeitsspeicher-x.Arbeitsspeicher.mean()
x["Arbeitsspeicher_z_quad"] = x.Arbeitsspeicher_z**2
x["Bild_Display_z"] = x.Bild_Display-x.Bild_Display.mean()
x["Bild_Display_z_quad"] = x.Bild_Display_z**2
x["Speicher_Grafik_z"] = x.Speicher_Grafik-x.Speicher_Grafik.mean()
x["Speicher_Grafik_z_quad"] = x.Speicher_Grafik_z**2
x["extern_Schnittstellen"] = x.extern_Schnittstellen
x["HDD_z"] = x.HDD-x.HDD.mean()
x["HDD_z_quad"] = x.HDD_z**2
x["SSD_z"] = x.SSD-x.SSD.mean()
x["SSD_z_quad"] = x.SSD_z**2
x["Akku_Kapazität_z"] = x.Akku_Kapazität-x.Akku_Kapazität.mean()
x["Akku_Kapazität_z_quad"] = x.Akku_Kapazität_z**2
x["PPI_Display_z"] = x.PPI_Display-x.PPI_Display.mean()
x["PPI_Display_z_quad"] = x.PPI_Display_z**2
x["Pixelshader_z"] = x.Pixelshader-x.Pixelshader.mean()
x["Pixelshader_z_quad"] = x.Pixelshader_z**2
```

Datensatz wird erneut gespeichert:

In [1314]:
```
x["Preis"] = y["Preis"]
x["Log_Preis"] = y["Log_Preis"]
x.to_csv(r'C:\Users\tobia\Desktop\Uni\Marburg\Bachelorarbeit\Alternate
\fertiges_datenset_transform.csv')
x = x.drop(["Preis", "Log_Preis"], axis=1)
```

Modellübersicht

Zurück zu Induktive Statistik

Die Vorgehensweise ist identisch für die drei Modelle. Es wird ein nicht reduziertes Modell erstellt, überprüft, ob Heteroskedastizität vorliegt und mögliche Multikollinearität überprüft. Da für Modell2 und Modell3 bereits Vergleichswerte vorhanden sind, werden die reduzierten und zentrierten Modelle Modell2 und Modell3 anhand des Bestimmtheitsmaßes bezügliche des Preises mit Modell1 verglichen. Die Resdiuen-Plots werden für die drei Modelle dargestellt, für jedes Modell eine fünf-fache Kreuzvalidierung durchgeführt und ein Modell ausgewählt, welches den Zusammenhang zwischen Prädiktoren und Zielgröße vermutlich am Besten beschreiben kann, wofür dann eine Gesamtübersicht erstellt wird.

1. Modell1
    - nicht zentriert und nicht reduziert
        - Regression
        - Breusch-Pagan
        - VIF
    - zentriert und nicht reduziert
        - VIF
    - zentriert und reduziert
2. Modell2
    - zentriert und nicht reduziert
        - Breusch-Pagan
        - Vergleich Modell1 und Modell2
    - zentriert und reduziert
3. Modell3
    - zentriert und nicht reduziert
        - Breusch-Pagan
        - VIF
    - zentriert und reduziert
        - Vergleich Modell1, Modell2 und Modell3

Modell1

nicht zentriert und nicht reduziert

Zurück zu Modellübersicht

quantitative erklärende Variablen Modell1 und Modell2:

In [1315]:
```
x_quant = x[['Kerne', "Kerne_quad", 'Takt_Prozessor', "Takt_Prozessor_
quad",
             'Arbeitsspeicher', "Arbeitsspeicher_quad",
             'Bild_Display', "Bild_Display_quad",
             'Speicher_Grafik', "Speicher_Grafik_quad",
             'extern_Schnittstellen',
             'HDD', "HDD_quad", 'SSD', 'SSD_quad',
             'Akku_Kapazität', "Akku_Kapazität_quad",
             'PPI_Display', "PPI_Display_quad",
             'Pixelshader', "Pixelshader_quad"]]
```

Zusammenfügen quantitativer und qualitativer Merkmale:

In [1316]:
```
x_neu1 = x_var_erstellen(x_quant, qualitativ)
```

Es erscheint hier eine Warnung, dass die Regressionsfunktion keinen vollen Rang hat. Deshalb wurden sämtliche Regressionen auch in Stata durchgeführt um zu überprüfen, ob auch hier eine Warnung erscheint oder Variablen ausgelassen wurden. Allerdings waren die Ergebnisse identisch

In [1317]:
```
model = sm.OLS(y.Preis, x_neu1)
mod_lin0 = model.fit(cov_type="HC0")
x["fit_mod_lin0"] = mod_lin0.fittedvalues
x["resid_mod_lin0"] = mod_lin0.resid
mod_lin0.summary()
```

```
C:\Users\tobia\Anaconda3\lib\site-packages\statsmodels\base\model.py:1
752: ValueWarning: covariance of constraints does not have full rank.
The number of constraints is 40, but rank is 36
  'rank is %d' % (J, J_), ValueWarning)
```

Out[1317]:

OLS Regression Results

Dep. Variable:	Preis	**R-squared:**	0.885
Model:	OLS	**Adj. R-squared:**	0.881
Method:	Least Squares	**F-statistic:**	306.7
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	0.00
Time:	18:10:00	**Log-Likelihood:**	-7225.4
No. Observations:	1039	**AIC:**	1.453e+04
Df Residuals:	998	**BIC:**	1.474e+04
Df Model:	40		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025
Kerne	-13.4993	38.246	-0.353	0.724	-88.459
Kerne_quad	12.5020	4.079	3.065	0.002	4.508
Takt_Prozessor	-0.4064	0.237	-1.717	0.086	-0.870
Takt_Prozessor_quad	0.0001	5.68e-05	2.382	0.017	2.4e-05
Arbeitsspeicher	31.7303	6.892	4.604	0.000	18.221
Arbeitsspeicher_quad	-0.0681	0.183	-0.372	0.710	-0.426
Bild_Display	-322.2233	148.314	-2.173	0.030	-612.914
Bild_Display_quad	11.0136	4.880	2.257	0.024	1.449
Speicher_Grafik	0.0790	0.019	4.200	0.000	0.042
Speicher_Grafik_quad	-1.194e-05	3.9e-06	-3.064	0.002	-1.96e-05
extern_Schnittstellen	13.6340	8.723	1.563	0.118	-3.463
HDD	0.1424	0.055	2.611	0.009	0.035
HDD_quad	-0.0002	3.34e-05	-5.538	0.000	-0.000
SSD	0.2886	0.097	2.972	0.003	0.098
SSD_quad	3.098e-05	4.68e-05	0.662	0.508	-6.07e-05
Akku_Kapazität	5.8603	4.918	1.192	0.233	-3.779
Akku_Kapazität_quad	0.0493	0.039	1.265	0.206	-0.027
PPI_Display	4.6292	2.163	2.140	0.032	0.390
PPI_Display_quad	-0.0063	0.005	-1.308	0.191	-0.016
Pixelshader	-0.3556	0.073	-4.902	0.000	-0.498
Pixelshader_quad	0.0002	3.25e-05	6.785	0.000	0.000
Laufwerk_VORHANDEN	14.8041	33.541	0.441	0.659	-50.934
Tastatur_Beleuchtung_vorhanden	137.2391	22.965	5.976	0.000	92.228

Fingerabdrucksensor_vorhanden	25.8538	21.315	1.213	0.225	-15.923
Touchscreen_vorhanden	82.2398	27.135	3.031	0.002	29.057
Mobilfunk_vorhanden	334.1124	37.750	8.851	0.000	260.123
NFC_vorhanden	244.0871	55.699	4.382	0.000	134.919
Mikrofon_Sound_vorhanden	24.5174	26.379	0.929	0.353	-27.184
Rabatt_Lehreinrichtung_vorhanden	31.9003	57.805	0.552	0.581	-81.395
Akku_Typ_ION	-48.1620	31.674	-1.521	0.128	-110.243
Marke_ASUS	-22.4712	30.641	-0.733	0.463	-82.527
Marke_Acer	-29.5293	27.186	-1.086	0.277	-82.814
Marke_Apple	446.6866	60.108	7.431	0.000	328.878
Marke_Dell	-23.8031	32.137	-0.741	0.459	-86.791
Marke_Lenovo	157.3070	27.165	5.791	0.000	104.064
Betriebssystem_DOS	-226.4823	36.245	-6.249	0.000	-297.520
Betriebssystem_LINUX	-31.3285	63.970	-0.490	0.624	-156.708
Betriebssystem_OHNE_BETRIEBSSYSTEM	-265.5741	36.641	-7.248	0.000	-337.390
Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER	134.1329	47.923	2.799	0.005	40.205
Typ_Arbeitsspeicher_DDR_4	-137.4838	40.639	-3.383	0.001	-217.135
Konstante	2041.9514	1156.233	1.766	0.077	-224.224

Omnibus:	67.402	**Durbin-Watson:**	1.749
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	89.573
Skew:	0.563	**Prob(JB):**	3.54e-20
Kurtosis:	3.894	**Cond. No.**	2.80e+09

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
[2] The condition number is large, 2.8e+09. This might indicate that there are strong multicollinearity or other numerical problems.

Breusch-Pagan Modell1

Der Breusch-Pagan Test wird mittels den quadrierten Werten der Resiuden als Zielgröße durchgeführt. Der in der Arbeit abgebildete P-Wert der F-Statistik ist in der untenstehenden Abbildung mit **Prob (F-statistic)** gekennzeichnet.

In [1318]:
```
x["residquad0"] = mod_lin0.resid**2
X_test = x[["Kerne", "Takt_Prozessor", "Arbeitsspeicher", "HDD", "SS
D", "Akku_Kapazität", "PPI_Display","Bild_Display",
           "Speicher_Grafik", "Pixelshader", "extern_Schnittstelle
n"]]
model = sm.OLS(x["residquad0"], sm.add_constant(X_test))
homoskedastizität = model.fit()
homoskedastizität.summary()
```

```
C:\Users\tobia\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2
389: FutureWarning: Method .ptp is deprecated and will be removed in a
future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

Out[1318]:

OLS Regression Results

Dep. Variable:	residquad0	**R-squared:**	0.067
Model:	OLS	**Adj. R-squared:**	0.057
Method:	Least Squares	**F-statistic:**	6.745
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	5.20e-11
Time:	18:10:00	**Log-Likelihood:**	-13492.
No. Observations:	1039	**AIC:**	2.701e+04
Df Residuals:	1027	**BIC:**	2.707e+04
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.468e+05	5.85e+04	2.507	0.012	3.19e+04	2.62e+05
Kerne	-9850.2844	4238.671	-2.324	0.020	-1.82e+04	-1532.839
Takt_Prozessor	-4.3560	11.542	-0.377	0.706	-27.005	18.293
Arbeitsspeicher	1006.7544	807.164	1.247	0.213	-577.125	2590.634
HDD	-17.7264	9.803	-1.808	0.071	-36.963	1.510
SSD	-9.8656	17.725	-0.557	0.578	-44.648	24.917
Akku_Kapazität	1220.2572	328.854	3.711	0.000	574.955	1865.560
PPI_Display	55.3150	105.977	0.522	0.602	-152.641	263.271
Bild_Display	-1.05e+04	3572.456	-2.940	0.003	-1.75e+04	-3493.918
Speicher_Grafik	1.2094	3.377	0.358	0.720	-5.417	7.836
Pixelshader	24.8270	12.750	1.947	0.052	-0.193	49.847
extern_Schnittstellen	5347.9573	2879.427	1.857	0.064	-302.275	1.1e+04

Omnibus:	798.863	**Durbin-Watson:**	1.952
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	16345.896
Skew:	3.398	**Prob(JB):**	0.00
Kurtosis:	21.204	**Cond. No.**	6.41e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.41e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Varianz-Inflations-Faktor

Die Werte ersten und zweiten Grades korrelieren erwartungsgemäß stark untereinander

In [1319]:
```
vif(x_neu1)
```

Out[1319]:

	VIF	Merkmale
0	610.312254	Bild_Display
1	603.397703	Bild_Display_quad
...	...	...
38	1.119246	Betriebssystem_DOS
39	1.067588	Betriebssystem_LINUX

40 rows × 2 columns

Modell1

zentriert und nicht reduziert

Zurück zu Modellübersicht

In [1320]:
```
x_quant_z = x[['Kerne_z', "Kerne_z_quad", 'Takt_Prozessor_z', "Takt_Pr
ozessor_z_quad",
           'Arbeitsspeicher_z', "Arbeitsspeicher_z_quad",
           'Bild_Display_z', "Bild_Display_z_quad",
           'Speicher_Grafik_z', "Speicher_Grafik_z_quad",
           'extern_Schnittstellen',
           'HDD_z', "HDD_z_quad", 'SSD_z', 'SSD_z_quad',
           'Akku_Kapazität_z', "Akku_Kapazität_z_quad",
           'PPI_Display_z', "PPI_Display_z_quad",
           'Pixelshader_z', "Pixelshader_z_quad"]]
```

In [1321]:
```
x_zentriert1 = x_var_erstellen(x_quant_z, qualitativ)
```

In der Arbeit auf Seite 34:

In [1322]:
```
model = sm.OLS(y.Preis, x_zentriert1)
mod_lin0_z = model.fit(cov_type="HC0")
x["fit_mod_lin0_z"] = mod_lin0_z.fittedvalues
x["resid_mod_lin0_z"] = mod_lin0_z.resid
print(mod_lin0_z.aic)
mod_lin0_z.summary()
```

```
14532.86452621716
```

```
C:\Users\tobia\Anaconda3\lib\site-packages\statsmodels\base\model.py:1
752: ValueWarning: covariance of constraints does not have full rank.
The number of constraints is 40, but rank is 39
  'rank is %d' % (J, J_), ValueWarning)
```

Out[1322]:

OLS Regression Results

Dep. Variable:	Preis	**R-squared:**	0.885
Model:	OLS	**Adj. R-squared:**	0.881
Method:	Least Squares	**F-statistic:**	313.4
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	0.00
Time:	18:10:00	**Log-Likelihood:**	-7225.4
No. Observations:	1039	**AIC:**	1.453e+04
Df Residuals:	998	**BIC:**	1.474e+04
Df Model:	40		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025	
Kerne_z	92.4853	11.355	8.145	0.000	70.231	1
Kerne_z_quad	12.5020	4.079	3.065	0.002	4.508	2
Takt_Prozessor_z	0.1376	0.037	3.675	0.000	0.064	
Takt_Prozessor_z_quad	0.0001	5.68e-05	2.382	0.017	2.4e-05	
Arbeitsspeicher_z	30.1168	3.194	9.429	0.000	23.856	3
Arbeitsspeicher_z_quad	-0.0681	0.183	-0.372	0.710	-0.426	
Bild_Display_z	9.7935	12.669	0.773	0.440	-15.038	3
Bild_Display_z_quad	11.0136	4.880	2.257	0.024	1.449	2
Speicher_Grafik_z	0.0316	0.008	3.748	0.000	0.015	
Speicher_Grafik_z_quad	-1.194e-05	3.9e-06	-3.064	0.002	-1.96e-05	-4
extern_Schnittstellen	13.6340	8.723	1.563	0.118	-3.463	3
HDD_z	0.0708	0.043	1.659	0.097	-0.013	
HDD_z_quad	-0.0002	3.34e-05	-5.538	0.000	-0.000	
SSD_z	0.3130	0.067	4.702	0.000	0.183	
SSD_z_quad	3.098e-05	4.68e-05	0.662	0.508	-6.07e-05	
Akku_Kapazität_z	11.2837	1.117	10.101	0.000	9.094	1
Akku_Kapazität_z_quad	0.0493	0.039	1.265	0.206	-0.027	
PPI_Display_z	2.6020	0.683	3.811	0.000	1.264	
PPI_Display_z_quad	-0.0063	0.005	-1.308	0.191	-0.016	
Pixelshader_z	-0.1494	0.054	-2.775	0.006	-0.255	
Pixelshader_z_quad	0.0002	3.25e-05	6.785	0.000	0.000	

Laufwerk_VORHANDEN	14.8041	33.541	0.441	0.659	-50.934	8
Tastatur_Beleuchtung_vorhanden	137.2391	22.965	5.976	0.000	92.228	18
Fingerabdrucksensor_vorhanden	25.8538	21.315	1.213	0.225	-15.923	6
Touchscreen_vorhanden	82.2398	27.135	3.031	0.002	29.057	13
Mobilfunk_vorhanden	334.1124	37.750	8.851	0.000	260.123	40
NFC_vorhanden	244.0871	55.699	4.382	0.000	134.919	35
Mikrofon_Sound_vorhanden	24.5174	26.379	0.929	0.353	-27.184	7
Rabatt_Lehreinrichtung_vorhanden	31.9003	57.805	0.552	0.581	-81.395	14
Akku_Typ_ION	-48.1620	31.674	-1.521	0.128	-110.243	1
Marke_ASUS	-22.4712	30.641	-0.733	0.463	-82.527	3
Marke_Acer	-29.5293	27.186	-1.086	0.277	-82.814	2
Marke_Apple	446.6866	60.108	7.431	0.000	328.878	56
Marke_Dell	-23.8031	32.137	-0.741	0.459	-86.791	3
Marke_Lenovo	157.3070	27.165	5.791	0.000	104.064	21
Betriebssystem_DOS	-226.4823	36.245	-6.249	0.000	-297.520	-15
Betriebssystem_LINUX	-31.3285	63.970	-0.490	0.624	-156.708	9
Betriebssystem_OHNE_BETRIEBSSYSTEM	-265.5741	36.641	-7.248	0.000	-337.390	-19
Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER	134.1329	47.923	2.799	0.005	40.205	22
Typ_Arbeitsspeicher_DDR_4	-137.4838	40.639	-3.383	0.001	-217.135	-5
Konstante	1134.1852	80.908	14.018	0.000	975.609	129

Omnibus:	67.402	**Durbin-Watson:**	1.749
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	89.573
Skew:	0.563	**Prob(JB):**	3.54e-20
Kurtosis:	3.894	**Cond. No.**	1.02e+08

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
[2] The condition number is large, 1.02e+08. This might indicate that there are strong multicollinearity or other numerical problems.

VIF (in der Arbeit auf Seite 33)

In [1323]:
```
vif(x_zentriert1)
```

Out[1323]:

	VIF	Merkmale
0	16.768947	Pixelshader_z
1	11.533474	PPI_Display_z
...	...	...
38	1.119246	Betriebssystem_DOS
39	1.067588	Betriebssystem_LINUX

40 rows × 2 columns

Modell1

zentriert und reduziert

Zurück zu Modellübersicht

In [1324]:
```
mod_lin0_z.wald_test("Arbeitsspeicher_z_quad= Bild_Display_z = extern_Schnittstellen = SSD_z_quad = Akku_Kapazität_z_quad = PPI_Display_z_quad =  Laufwerk_VORHANDEN = Fingerabdrucksensor_vorhanden = Mikrofon_Sound_vorhanden = Rabatt_Lehreinrichtung_vorhanden = Akku_Typ_ION = Marke_ASUS = Marke_Acer = Marke_Dell= Betriebssystem_LINUX= 0", use_f=True)
```

Out[1324]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=array([[0.95380624]]), p=0.503074329708254, df_denom=998, df_num=15>
```

In der Arbeit auf Seite 35:

In [1325]:
```
x_zentriert2= x_zentriert1.drop(["Arbeitsspeicher_z_quad", "Bild_Displ
ay_z", "extern_Schnittstellen", "SSD_z_quad", "Akku_Kapazität_z_quad",
"PPI_Display_z_quad",  "Laufwerk_VORHANDEN", "Fingerabdrucksensor_vorh
anden", "Mikrofon_Sound_vorhanden", "Rabatt_Lehreinrichtung_vorhande
n", "Akku_Typ_ION","Marke_ASUS", "Marke_Acer", "Marke_Dell", "Betriebs
system_LINUX"], axis=1)
model = sm.OLS(y.Preis, x_zentriert2)
mod_lin1_z = model.fit(cov_type="HC0")
x["fit_mod_lin1_z"] = mod_lin1_z.fittedvalues
x["resid_mod_lin1_z"] = mod_lin1_z.resid
print(mod_lin1_z.aic)
mod_lin1_z.summary()
```

```
14520.353195317091
```

```
C:\Users\tobia\Anaconda3\lib\site-packages\statsmodels\base\model.py:1
752: ValueWarning: covariance of constraints does not have full rank.
The number of constraints is 25, but rank is 24
  'rank is %d' % (J, J_), ValueWarning)
```

Out[1325]:

OLS Regression Results

Dep. Variable:	Preis	**R-squared:**	0.883
Model:	OLS	**Adj. R-squared:**	0.881
Method:	Least Squares	**F-statistic:**	485.2
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	0.00
Time:	18:10:01	**Log-Likelihood:**	-7234.2
No. Observations:	1039	**AIC:**	1.452e+04
Df Residuals:	1013	**BIC:**	1.465e+04
Df Model:	25		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025	
Kerne_z	94.1942	11.042	8.531	0.000	72.553	1
Kerne_z_quad	12.7691	3.862	3.306	0.001	5.200	
Takt_Prozessor_z	0.1401	0.036	3.865	0.000	0.069	
Takt_Prozessor_z_quad	0.0001	5.67e-05	2.398	0.016	2.48e-05	
Arbeitsspeicher_z	30.0927	2.291	13.137	0.000	25.603	
Bild_Display_z_quad	9.7033	4.822	2.012	0.044	0.253	
Speicher_Grafik_z	0.0370	0.008	4.718	0.000	0.022	
Speicher_Grafik_z_quad	-1.288e-05	3.84e-06	-3.353	0.001	-2.04e-05	-
HDD_z	0.0929	0.035	2.619	0.009	0.023	
HDD_z_quad	-0.0002	2.86e-05	-7.216	0.000	-0.000	
SSD_z	0.3312	0.045	7.388	0.000	0.243	
Akku_Kapazität_z	12.9188	0.948	13.622	0.000	11.060	
PPI_Display_z	1.8087	0.349	5.188	0.000	1.125	
Pixelshader_z	-0.1842	0.052	-3.524	0.000	-0.287	
Pixelshader_z_quad	0.0002	3.17e-05	7.461	0.000	0.000	
Tastatur_Beleuchtung_vorhanden	135.4728	18.227	7.432	0.000	99.748	1
Touchscreen_vorhanden	78.7942	26.422	2.982	0.003	27.009	1
Mobilfunk_vorhanden	340.9167	35.845	9.511	0.000	270.661	4
NFC_vorhanden	274.4997	50.380	5.449	0.000	175.757	3
Marke_Apple	510.8667	41.246	12.386	0.000	430.027	5
Marke_Lenovo	188.2366	23.047	8.168	0.000	143.066	2

Betriebssystem_DOS	-219.9611	33.469	-6.572	0.000	-285.559	-1
Betriebssystem_OHNE_BETRIEBSSYSTEM	-242.6585	30.994	-7.829	0.000	-303.405	-1
Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER	135.0046	45.851	2.944	0.003	45.138	2
Typ_Arbeitsspeicher_DDR_4	-108.8859	36.541	-2.980	0.003	-180.505	-
Konstante	1171.1419	46.300	25.295	0.000	1080.396	12

Omnibus:	89.877	**Durbin-Watson:**	1.708
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	132.470
Skew:	0.655	**Prob(JB):**	1.72e-29
Kurtosis:	4.160	**Cond. No.**	7.80e+07

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
[2] The condition number is large, 7.8e+07. This might indicate that there are strong multicollinearity or other numerical problems.

Modell2

zentriert und nicht reduziert

Zurück zu Modellübersicht

(In der Arbeit auf Seite 37) Es werden dieselben Prädiktoren wie in Modell1 gewählt, die bereits zentriert sind, da sich gezeigt hat, dass bei unzentrierten Werten Multikollinearität vorliegt

In [1326]:
```
model = sm.OLS(y.Log_Preis, x_zentriert1)
mod_log0_z = model.fit(cov_type="HC0")
x["resid_mod_log0_z"] =mod_log0_z.resid
x["fit_mod_log0_z"] = mod_log0_z.fittedvalues
mod_log0_z.summary()
```

```
C:\Users\tobia\Anaconda3\lib\site-packages\statsmodels\base\model.py:1
752: ValueWarning: covariance of constraints does not have full rank.
The number of constraints is 40, but rank is 39
  'rank is %d' % (J, J_), ValueWarning)
```

Out[1326]:

OLS Regression Results

Dep. Variable:	Log_Preis	**R-squared:**	0.875
Model:	OLS	**Adj. R-squared:**	0.870
Method:	Least Squares	**F-statistic:**	367.6
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	0.00
Time:	18:10:01	**Log-Likelihood:**	237.45
No. Observations:	1039	**AIC:**	-392.9
Df Residuals:	998	**BIC:**	-190.1
Df Model:	40		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025	0.975
Kerne_z	0.0647	0.009	7.439	0.000	0.048	0.08
Kerne_z_quad	-0.0066	0.003	-2.189	0.029	-0.012	-0.00
Takt_Prozessor_z	9.407e-05	3.12e-05	3.011	0.003	3.28e-05	0.00
Takt_Prozessor_z_quad	-2.215e-08	4.49e-08	-0.493	0.622	-1.1e-07	6.59e-0
Arbeitsspeicher_z	0.0262	0.002	11.079	0.000	0.022	0.03
Arbeitsspeicher_z_quad	-0.0008	0.000	-7.232	0.000	-0.001	-0.00
Bild_Display_z	0.0064	0.009	0.681	0.496	-0.012	0.02
Bild_Display_z_quad	0.0215	0.004	5.479	0.000	0.014	0.02
Speicher_Grafik_z	4.989e-05	6.57e-06	7.595	0.000	3.7e-05	6.28e-0
Speicher_Grafik_z_quad	-6.368e-09	2.34e-09	-2.725	0.006	-1.09e-08	-1.79e-0
extern_Schnittstellen	0.0069	0.007	1.040	0.298	-0.006	0.02
HDD_z	6.813e-05	3.16e-05	2.154	0.031	6.15e-06	0.00
HDD_z_quad	-1.127e-07	2.52e-08	-4.480	0.000	-1.62e-07	-6.34e-0
SSD_z	0.0003	4.69e-05	5.438	0.000	0.000	0.00
SSD_z_quad	-9.963e-08	3.02e-08	-3.304	0.001	-1.59e-07	-4.05e-0
Akku_Kapazität_z	0.0099	0.001	9.500	0.000	0.008	0.01
Akku_Kapazität_z_quad	-6.773e-05	3.11e-05	-2.180	0.029	-0.000	-6.84e-0
PPI_Display_z	0.0028	0.000	6.507	0.000	0.002	0.00
PPI_Display_z_quad	-1.342e-05	3.29e-06	-4.085	0.000	-1.99e-05	-6.98e-0

Pixelshader_z	-0.0002	4.04e-05	-4.263	0.000	-0.000	-9.3e-0
Pixelshader_z_quad	1.403e-07	1.91e-08	7.341	0.000	1.03e-07	1.78e-0
Laufwerk_VORHANDEN	-0.0802	0.025	-3.204	0.001	-0.129	-0.03
Tastatur_Beleuchtung_vorhanden	0.1178	0.019	6.311	0.000	0.081	0.15
Fingerabdrucksensor_vorhanden	0.0247	0.017	1.481	0.139	-0.008	0.05
Touchscreen_vorhanden	0.0554	0.021	2.636	0.008	0.014	0.09
Mobilfunk_vorhanden	0.2727	0.027	9.943	0.000	0.219	0.32
NFC_vorhanden	0.0588	0.042	1.418	0.156	-0.023	0.14
Mikrofon_Sound_vorhanden	0.0019	0.019	0.101	0.919	-0.034	0.03
Rabatt_Lehreinrichtung_vorhanden	0.0458	0.037	1.227	0.220	-0.027	0.11
Akku_Typ_ION	-0.0415	0.022	-1.896	0.058	-0.084	0.00
Marke_ASUS	-0.0512	0.025	-2.069	0.039	-0.100	-0.00
Marke_Acer	-0.0068	0.023	-0.293	0.770	-0.052	0.03
Marke_Apple	0.2481	0.042	5.947	0.000	0.166	0.33
Marke_Dell	0.0342	0.024	1.447	0.148	-0.012	0.08
Marke_Lenovo	0.1092	0.020	5.358	0.000	0.069	0.14
Betriebssystem_DOS	-0.2932	0.031	-9.455	0.000	-0.354	-0.23
Betriebssystem_LINUX	-0.0658	0.048	-1.374	0.170	-0.160	0.02
Betriebssystem_OHNE_BETRIEBSSYSTEM	-0.2575	0.032	-8.014	0.000	-0.320	-0.19
Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER	0.0590	0.036	1.630	0.103	-0.012	0.13
Typ_Arbeitsspeicher_DDR_4	-0.1494	0.033	-4.548	0.000	-0.214	-0.08
Konstante	7.0832	0.063	113.277	0.000	6.961	7.20

Omnibus:	36.128	**Durbin-Watson:**	1.693
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	42.802
Skew:	0.397	**Prob(JB):**	5.08e-10
Kurtosis:	3.598	**Cond. No.**	1.02e+08

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
[2] The condition number is large, 1.02e+08. This might indicate that there are strong multicollinearity or other numerical problems.

Breusch-Pagan Modell2

In [1327]:
```
x["logresidquad0"]=x.resid_mod_log0_z**2
X_test = x[["Kerne", "Takt_Prozessor", "Arbeitsspeicher", "HDD", "SS
D", "Akku_Kapazität", "PPI_Display","Bild_Display",
            "Speicher_Grafik", "Pixelshader", "extern_Schnittstelle
n"]]
X_test = standardisieren(X_test)
model = sm.OLS(x["logresidquad0"], sm.add_constant(X_test))
homoskedastizität = model.fit()
homoskedastizität.summary()
```

```
C:\Users\tobia\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2
389: FutureWarning: Method .ptp is deprecated and will be removed in a
future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

Out[1327]:

OLS Regression Results

Dep. Variable:	logresidquad0	**R-squared:**	0.054
Model:	OLS	**Adj. R-squared:**	0.043
Method:	Least Squares	**F-statistic:**	5.277
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	3.75e-08
Time:	18:10:01	**Log-Likelihood:**	1481.7
No. Observations:	1039	**AIC:**	-2939.
Df Residuals:	1027	**BIC:**	-2880.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0371	0.002	20.434	0.000	0.034	0.041
Kerne	-0.0087	0.003	-2.851	0.004	-0.015	-0.003
Takt_Prozessor	-0.0006	0.003	-0.240	0.810	-0.006	0.004
Arbeitsspeicher	-0.0023	0.003	-0.858	0.391	-0.008	0.003
HDD	-0.0054	0.002	-2.487	0.013	-0.010	-0.001
SSD	-0.0035	0.003	-1.396	0.163	-0.008	0.001
Akku_Kapazität	0.0033	0.003	1.134	0.257	-0.002	0.009
PPI_Display	-0.0028	0.003	-1.129	0.259	-0.008	0.002
Bild_Display	-0.0119	0.003	-4.764	0.000	-0.017	-0.007
Speicher_Grafik	0.0041	0.004	0.915	0.360	-0.005	0.013
Pixelshader	0.0037	0.004	0.860	0.390	-0.005	0.012
extern_Schnittstellen	0.0046	0.002	2.017	0.044	0.000	0.009

Omnibus:	667.066	**Durbin-Watson:**	1.966
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6897.292
Skew:	2.882	**Prob(JB):**	0.00
Kurtosis:	14.229	**Cond. No.**	6.83

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Vergleich Modell 1 und Modell 2

In [1328]:
```
Bestimmtheitsmaß_Preis_m2 = y.Preis.corr(np.exp(x.fit_mod_log0_z))**2
print("Bestimmtheitsmaß:", Bestimmtheitsmaß_Preis_m2)
print("adj. Bestimmtheitsmaß:",1-(1-Bestimmtheitsmaß_Preis_m2)*(1039-
1)/(1039-mod_log0_z.df_model-1))
```

```
Bestimmtheitsmaß: 0.8791271684947216
adj. Bestimmtheitsmaß: 0.8742825660295802
```

Modell2

zentriert und reduziert

Zurück zu Modellübersicht

(In der Arbeit auf Seite 38)

In [1329]:
```
mod_log0_z.wald_test("Takt_Prozessor_z_quad=Bild_Display_z=extern_Schn
ittstellen= Fingerabdrucksensor_vorhanden = Mikrofon_Sound_vorhanden =
Rabatt_Lehreinrichtung_vorhanden= Marke_Acer = Marke_Dell = Betriebssy
stem_LINUX  = 0", use_f=True)
```

Out[1329]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=array([[0.89941477]]), p=0.5250713970053864, df_denom=998,
df_num=9>
```

In [1330]:
```
x_zentriert3 = x_zentriert1.drop(["Takt_Prozessor_z_quad","Bild_Displa
y_z","extern_Schnittstellen", "Fingerabdrucksensor_vorhanden", "Mikrof
on_Sound_vorhanden", "Rabatt_Lehreinrichtung_vorhanden", "Marke_Acer",
"Marke_Dell", "Betriebssystem_LINUX"], axis=1)

model = sm.OLS(y.Log_Preis, x_zentriert3)
mod_log1_z = model.fit(cov_type="HC0")
x["resid_mod_log1_z"] =mod_log1_z.resid
x["fit_mod_log1_z"] = mod_log1_z.fittedvalues
mod_log1_z.summary()
```

```
C:\Users\tobia\Anaconda3\lib\site-packages\statsmodels\base\model.py:1
752: ValueWarning: covariance of constraints does not have full rank.
The number of constraints is 31, but rank is 30
  'rank is %d' % (J, J_), ValueWarning)
```

Out[1330]:

OLS Regression Results

Dep. Variable:	Log_Preis	**R-squared:**	0.874
Model:	OLS	**Adj. R-squared:**	0.870
Method:	Least Squares	**F-statistic:**	455.7
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	0.00
Time:	18:10:01	**Log-Likelihood:**	232.97
No. Observations:	1039	**AIC:**	-401.9
Df Residuals:	1007	**BIC:**	-243.7
Df Model:	31		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025	0.975
Kerne_z	0.0657	0.009	7.633	0.000	0.049	0.08
Kerne_z_quad	-0.0066	0.003	-2.292	0.022	-0.012	-0.00
Takt_Prozessor_z	8.489e-05	2.78e-05	3.050	0.002	3.03e-05	0.00
Arbeitsspeicher_z	0.0263	0.002	11.359	0.000	0.022	0.03
Arbeitsspeicher_z_quad	-0.0008	0.000	-7.517	0.000	-0.001	-0.00
Bild_Display_z_quad	0.0214	0.004	5.256	0.000	0.013	0.02
Speicher_Grafik_z	5.07e-05	6.45e-06	7.862	0.000	3.81e-05	6.33e-0
Speicher_Grafik_z_quad	-6.692e-09	2.33e-09	-2.867	0.004	-1.13e-08	-2.12e-0
HDD_z	7.847e-05	3.13e-05	2.504	0.012	1.7e-05	0.00
HDD_z_quad	-1.222e-07	2.39e-08	-5.116	0.000	-1.69e-07	-7.54e-0
SSD_z	0.0003	4.71e-05	5.605	0.000	0.000	0.00
SSD_z_quad	-1.055e-07	3.06e-08	-3.447	0.001	-1.66e-07	-4.55e-0
Akku_Kapazität_z	0.0105	0.001	10.560	0.000	0.009	0.01
Akku_Kapazität_z_quad	-6.285e-05	2.98e-05	-2.112	0.035	-0.000	-4.52e-0
PPI_Display_z	0.0027	0.000	7.323	0.000	0.002	0.00
PPI_Display_z_quad	-1.301e-05	2.9e-06	-4.482	0.000	-1.87e-05	-7.32e-0
Pixelshader_z	-0.0002	3.91e-05	-4.534	0.000	-0.000	-0.00
Pixelshader_z_quad	1.435e-07	1.88e-08	7.620	0.000	1.07e-07	1.8e-0

Laufwerk_VORHANDEN	-0.0793	0.024	-3.295	0.001	-0.126	-0.03
Tastatur_Beleuchtung_vorhanden	0.1239	0.018	6.947	0.000	0.089	0.15
Touchscreen_vorhanden	0.0533	0.020	2.653	0.008	0.014	0.09
Mobilfunk_vorhanden	0.2782	0.027	10.407	0.000	0.226	0.33
NFC_vorhanden	0.0749	0.039	1.899	0.058	-0.002	0.15
Akku_Typ_ION	-0.0433	0.021	-2.023	0.043	-0.085	-0.00
Marke_ASUS	-0.0643	0.021	-3.108	0.002	-0.105	-0.02
Marke_Apple	0.2287	0.036	6.348	0.000	0.158	0.29
Marke_Lenovo	0.1070	0.018	5.930	0.000	0.072	0.14
Betriebssystem_DOS	-0.2912	0.029	-10.109	0.000	-0.348	-0.23
Betriebssystem_OHNE_BETRIEBSSYSTEM	-0.2461	0.030	-8.173	0.000	-0.305	-0.18
Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER	0.0431	0.035	1.237	0.216	-0.025	0.11
Typ_Arbeitsspeicher_DDR_4	-0.1395	0.030	-4.630	0.000	-0.199	-0.08
Konstante	7.1310	0.042	171.126	0.000	7.049	7.21

Omnibus:	33.975	**Durbin-Watson:**	1.694
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	39.268
Skew:	0.391	**Prob(JB):**	2.97e-09
Kurtosis:	3.544	**Cond. No.**	8.39e+07

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
[2] The condition number is large, 8.39e+07. This might indicate that there are strong multicollinearity or other numerical problems.

Modell3

zentriert und nicht reduziert

Zurück zu Modellübersicht

(In der Arbeit auf Seite 39) Die Werte für HDD wurden zentriert, da hier ein quadratischer Term miteinbezogen wurde

In [1331]:
```
x_neu2 = x[['log_Kerne', 'log_Takt_Prozessor', 'log_Arbeitsspeicher',
"log_Bild_Display",
       'Speicher_Grafik', 'extern_Schnittstellen', "HDD_z", "HDD_z_qua
d",'log_SSD',
       'log_Akku_Kapazität', 'log_PPI_Display', 'log_Pixelshader']]
x_neu2 = x_var_erstellen(x_neu2, qualitativ)

model = sm.OLS(y.Log_Preis, x_neu2)
mod_logn0_z = model.fit(cov_type="HC0")
x["resid_mod_logn0_z"] =mod_logn0_z.resid
x["fit_mod_logn0_z"] = mod_logn0_z.fittedvalues
mod_logn0_z.summary()
```

Out[1331]:

OLS Regression Results

Dep. Variable:	Log_Preis	**R-squared:**	0.864
Model:	OLS	**Adj. R-squared:**	0.860
Method:	Least Squares	**F-statistic:**	333.2
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	0.00
Time:	18:10:01	**Log-Likelihood:**	193.60
No. Observations:	1039	**AIC:**	-323.2
Df Residuals:	1007	**BIC:**	-164.9
Df Model:	31		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025	0.975]
log_Kerne	0.2271	0.033	6.929	0.000	0.163	0.291
log_Takt_Prozessor	0.1468	0.052	2.815	0.005	0.045	0.249
log_Arbeitsspeicher	0.3245	0.022	15.052	0.000	0.282	0.367
log_Bild_Display	0.0187	0.123	0.152	0.879	-0.222	0.259
Speicher_Grafik	6.085e-05	6.07e-06	10.021	0.000	4.89e-05	7.28e-05
extern_Schnittstellen	0.0109	0.007	1.587	0.112	-0.003	0.024
HDD_z	-6.847e-06	3.59e-05	-0.191	0.849	-7.72e-05	6.35e-05
HDD_z_quad	-2.963e-08	3.61e-08	-0.820	0.412	-1e-07	4.12e-08
log_SSD	0.0258	0.007	3.619	0.000	0.012	0.040
log_Akku_Kapazität	0.4989	0.049	10.186	0.000	0.403	0.595
log_PPI_Display	0.2849	0.042	6.778	0.000	0.203	0.367
log_Pixelshader	-0.0447	0.008	-5.438	0.000	-0.061	-0.029
Laufwerk_VORHANDEN	-0.0644	0.027	-2.396	0.017	-0.117	-0.012
Tastatur_Beleuchtung_vorhanden	0.1142	0.018	6.426	0.000	0.079	0.149
Fingerabdrucksensor_vorhanden	0.0264	0.017	1.566	0.117	-0.007	0.059
Touchscreen_vorhanden	0.0651	0.021	3.120	0.002	0.024	0.106
Mobilfunk_vorhanden	0.2804	0.028	10.000	0.000	0.225	0.335
NFC_vorhanden	0.0235	0.040	0.583	0.560	-0.056	0.103
Mikrofon_Sound_vorhanden	-0.0069	0.019	-0.368	0.713	-0.044	0.030
Rabatt_Lehreinrichtung_vorhanden	0.0474	0.041	1.163	0.245	-0.032	0.127
Akku_Typ_ION	-0.0485	0.022	-2.180	0.029	-0.092	-0.005
Marke_ASUS	-0.0591	0.027	-2.213	0.027	-0.111	-0.007

Marke_Acer	0.0162	0.024	0.682	0.495	-0.030	0.063
Marke_Apple	0.2615	0.036	7.265	0.000	0.191	0.332
Marke_Dell	0.0305	0.024	1.260	0.208	-0.017	0.078
Marke_Lenovo	0.1225	0.021	5.973	0.000	0.082	0.163
Betriebssystem_DOS	-0.2747	0.035	-7.920	0.000	-0.343	-0.207
Betriebssystem_LINUX	-0.1217	0.050	-2.422	0.015	-0.220	-0.023
Betriebssystem_OHNE_BETRIEBSSYSTEM	-0.2511	0.034	-7.347	0.000	-0.318	-0.184
Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER	0.0706	0.036	1.977	0.048	0.001	0.141
Typ_Arbeitsspeicher_DDR_4	-0.1888	0.030	-6.383	0.000	-0.247	-0.131
Konstante	1.3276	0.515	2.580	0.010	0.319	2.336

Omnibus:	29.417	**Durbin-Watson:**	1.721
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	32.055
Skew:	0.383	**Prob(JB):**	1.09e-07
Kurtosis:	3.393	**Cond. No.**	2.82e+07

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
[2] The condition number is large, 2.82e+07. This might indicate that there are strong multicollinearity or other numerical problems.

Breusch-Pagan Modell3

In [1332]:
```
x["logresid1quad"] = x.resid_mod_logn0_z**2
X_test = x[['log_Kerne', 'log_Takt_Prozessor', 'log_Arbeitsspeicher',
"log_Bild_Display",
       'Speicher_Grafik', 'extern_Schnittstellen', "HDD_z", "HDD_z_qua
d",'log_SSD',
       'log_Akku_Kapazität', 'log_PPI_Display', "log_Pixelshader"]]
model = sm.OLS(x["logresid1quad"], sm.add_constant(X_test))
homoskedastizität = model.fit()
homoskedastizität.summary()
```

Out[1332]:

OLS Regression Results

Dep. Variable:	logresid1quad	**R-squared:**	0.054
Model:	OLS	**Adj. R-squared:**	0.043
Method:	Least Squares	**F-statistic:**	4.927
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	5.75e-08
Time:	18:10:01	**Log-Likelihood:**	1437.2
No. Observations:	1039	**AIC:**	-2848.
Df Residuals:	1026	**BIC:**	-2784.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.6259	0.142	4.397	0.000	0.347	0.905
log_Kerne	-0.0366	0.009	-4.037	0.000	-0.054	-0.019
log_Takt_Prozessor	-0.0149	0.014	-1.069	0.286	-0.042	0.012
log_Arbeitsspeicher	-0.0073	0.006	-1.193	0.233	-0.019	0.005
log_Bild_Display	-0.1488	0.032	-4.607	0.000	-0.212	-0.085
Speicher_Grafik	4.544e-06	1.72e-06	2.636	0.009	1.16e-06	7.93e-06
extern_Schnittstellen	0.0046	0.002	2.752	0.006	0.001	0.008
HDD_z	-1.566e-05	1.46e-05	-1.075	0.282	-4.42e-05	1.29e-05
HDD_z_quad	1.392e-09	1.86e-08	0.075	0.940	-3.51e-08	3.79e-08
log_SSD	-0.0020	0.002	-0.886	0.376	-0.007	0.002
log_Akku_Kapazität	0.0031	0.011	0.293	0.769	-0.018	0.024
log_PPI_Display	-0.0093	0.012	-0.790	0.430	-0.033	0.014
log_Pixelshader	0.0018	0.002	0.782	0.434	-0.003	0.006

Omnibus:	653.208	**Durbin-Watson:**	1.953
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6726.205
Skew:	2.800	**Prob(JB):**	0.00
Kurtosis:	14.136	**Cond. No.**	2.53e+07

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.53e+07. This might indicate that there are strong multicollinearity or other numerical problems.

Varianz-Inflations-Faktor

In Modell3 liegt keine besonders starke Multikollinearität vor (In der Arbeit auf Seite 32)

In [1333]:
```
vif(x_neu2)
```

Out[1333]:

	VIF	Merkmale
0	9.866593	HDD_z
1	8.264706	HDD_z_quad
...	...	...
29	1.108941	Betriebssystem_DOS
30	1.049231	Betriebssystem_LINUX

31 rows × 2 columns

Vergleich Modell1, Modell2 und Modell3

In [1334]:
```
Bestimmtheitsmaß_Preis_log = y.Preis.corr(np.exp(x.fit_mod_logn0_z))**2
print("Bestimmtheitsmaß:",Bestimmtheitsmaß_Preis_log)
print("adj.Bestimmtheitsmaß:",1-(1-Bestimmtheitsmaß_Preis_log)*(1039-1)/(1039-mod_logn0_z.df_model-1))
```

```
Bestimmtheitsmaß: 0.8631416774355989
adj.Bestimmtheitsmaß: 0.8589285612494058
```

Modell3

zentriert und reduziert

Zurück zu Modellübersicht

In [1335]:
```
mod_logn0_z.wald_test("log_Bild_Display = extern_Schnittstellen=HDD_z = HDD_z_quad=Fingerabdrucksensor_vorhanden = NFC_vorhanden = Marke_Acer =Marke_Dell = Mikrofon_Sound_vorhanden =  Rabatt_Lehreinrichtung_vorhanden = 0", use_f=True)
```

Out[1335]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=array([[1.35842544]]), p=0.1946781596982428, df_denom=1.01e+03, df_num=10>
```

In [1336]:
```
x_neu3 = x_neu2.drop(["log_Bild_Display", "extern_Schnittstellen", "HD
D_z", "HDD_z_quad", "Fingerabdrucksensor_vorhanden", "NFC_vorhanden",
"Marke_Acer", "Marke_Dell", "Mikrofon_Sound_vorhanden",  "Rabatt_Lehre
inrichtung_vorhanden"], axis=1)
model = sm.OLS(y.Log_Preis, x_neu3)
mod_logn1_z = model.fit(cov_type="HC0")
x["resid_mod_logn1_z"] =mod_logn1_z.resid
x["fit_mod_logn1_z"] = mod_logn1_z.fittedvalues
mod_logn1_z.summary()
```

Out[1336]:

OLS Regression Results

Dep. Variable:	Log_Preis	**R-squared:**	0.863
Model:	OLS	**Adj. R-squared:**	0.860
Method:	Least Squares	**F-statistic:**	468.3
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	0.00
Time:	18:10:01	**Log-Likelihood:**	187.87
No. Observations:	1039	**AIC:**	-331.7
Df Residuals:	1017	**BIC:**	-222.9
Df Model:	21		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025	0.975]
log_Kerne	0.2346	0.033	7.145	0.000	0.170	0.299
log_Takt_Prozessor	0.1438	0.050	2.873	0.004	0.046	0.242
log_Arbeitsspeicher	0.3205	0.021	15.433	0.000	0.280	0.361
Speicher_Grafik	6.025e-05	5.79e-06	10.404	0.000	4.89e-05	7.16e-05
log_SSD	0.0300	0.006	4.758	0.000	0.018	0.042
log_Akku_Kapazität	0.5427	0.044	12.361	0.000	0.457	0.629
log_PPI_Display	0.2863	0.037	7.644	0.000	0.213	0.360
log_Pixelshader	-0.0481	0.008	-5.842	0.000	-0.064	-0.032
Laufwerk_VORHANDEN	-0.0636	0.025	-2.509	0.012	-0.113	-0.014
Tastatur_Beleuchtung_vorhanden	0.1231	0.017	7.351	0.000	0.090	0.156
Touchscreen_vorhanden	0.0600	0.020	2.990	0.003	0.021	0.099
Mobilfunk_vorhanden	0.2913	0.026	11.282	0.000	0.241	0.342
Akku_Typ_ION	-0.0509	0.021	-2.371	0.018	-0.093	-0.009
Marke_ASUS	-0.0758	0.021	-3.545	0.000	-0.118	-0.034
Marke_Apple	0.2289	0.028	8.309	0.000	0.175	0.283
Marke_Lenovo	0.1193	0.018	6.712	0.000	0.084	0.154
Betriebssystem_DOS	-0.2751	0.033	-8.392	0.000	-0.339	-0.211
Betriebssystem_LINUX	-0.1160	0.050	-2.330	0.020	-0.214	-0.018
Betriebssystem_OHNE_BETRIEBSSYSTEM	-0.2380	0.032	-7.373	0.000	-0.301	-0.175
Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER	0.0599	0.032	1.855	0.064	-0.003	0.123
Typ_Arbeitsspeicher_DDR_4	-0.1816	0.026	-7.078	0.000	-0.232	-0.131
Konstante	1.2824	0.389	3.296	0.001	0.520	2.045

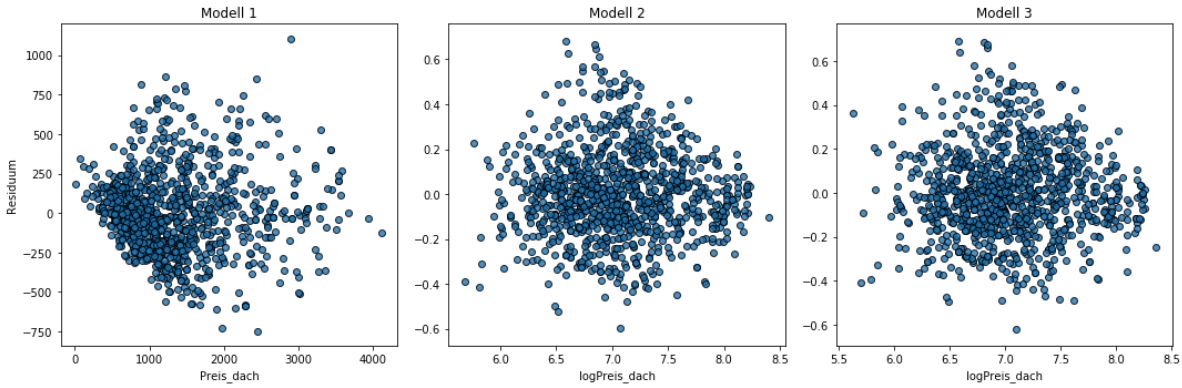
Omnibus:	30.163	**Durbin-Watson:**	1.712
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	32.846
Skew:	0.390	**Prob(JB):**	7.37e-08
Kurtosis:	3.388	**Cond. No.**	1.88e+05

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
[2] The condition number is large, 1.88e+05. This might indicate that there are strong multicollinearity or other numerical problems.

Residuen Plot Modelle 1-3 nicht reduziert

Zurück zu Induktive Statistik

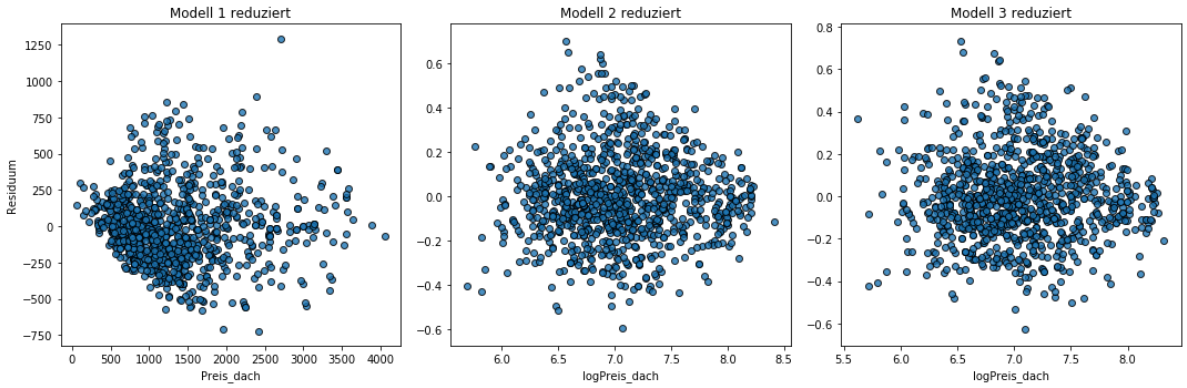
In [1337]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(15,5))
ax1.scatter(x.fit_mod_lin0_z, x.resid_mod_lin0_z, edgecolor="black", a
lpha=0.8,label="Preis")
ax2.scatter(x.fit_mod_log0_z, x.resid_mod_log0_z, edgecolor="black", a
lpha=0.8, label="logPreis")
ax3.scatter(x.fit_mod_logn0_z, x.resid_mod_logn0_z, edgecolor="black",
alpha=0.8, label="logPreis")
ax1.set_ylabel("Residuum")
ax1.set_xlabel("Preis_dach")
ax1.set_title("Modell 1")
ax2.set_xlabel("logPreis_dach")
ax2.set_xlabel("logPreis_dach")
ax2.set_title("Modell 2")
ax3.set_xlabel("logPreis_dach")
ax3.set_title("Modell 3")
plt.tight_layout()
plt.savefig("residuen_ols_log_ols");
```



Residuen Plot Modelle 1-3 reduziert

Zurück zu Induktive Statistik

In [1338]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(15,5))
ax1.scatter(x.fit_mod_lin1_z, x.resid_mod_lin1_z, edgecolor="black", a
lpha=0.8,label="Preis")
ax2.scatter(x.fit_mod_log1_z, x.resid_mod_log1_z, edgecolor="black", a
lpha=0.8, label="logPreis")
ax3.scatter(x.fit_mod_logn1_z, x.resid_mod_logn1_z, edgecolor="black",
alpha=0.8, label="logPreis")
ax1.set_ylabel("Residuum")
ax1.set_xlabel("Preis_dach")
ax1.set_title("Modell 1 reduziert")
ax2.set_xlabel("logPreis_dach")
ax2.set_xlabel("logPreis_dach")
ax2.set_title("Modell 2 reduziert")
ax3.set_xlabel("logPreis_dach")
ax3.set_title("Modell 3 reduziert")
plt.tight_layout()
plt.savefig("residuen_ols_log_ols_red");
```



Kreuzvalidierung

Zurück zu Induktive Statistik

Es wird eine Funktion erstellt, mit deren Hilfe die Bestimmtheitsmaße aus den logarithmierten und linearen Modellen verglichen werden können. Für die logarithmierten Preise wird also die Approximation aus Wooldrige (2012, S.214) verwendet

In [1339]:
```
def kreuzval(unabh, abh, ja_nein):
    from sklearn.metrics import r2_score
    from sklearn.model_selection import KFold
    daten = unabh.copy().reset_index(drop=True).values
    Preis = abh.copy().reset_index(drop=True).values
    Preis_lin = y.Preis.copy().reset_index(drop=True).values
# Datensatz wird in fünf Teile unterteilt, shuffle= ja_nein bietet die
Option,
# dass die Werte zufällig in Gruppen unterteilt werden:
    kf = KFold(n_splits=5, shuffle=ja_nein)
    r_sq_preis, gesamt_rsqu, gesamt_adj_rsqu = [],[], []
    i=1
# Die abhängigen und unabhängigen Merkmale werden nach Indizes unterte
ilt
    for train_index, test_index in kf.split(daten):
        X_train, X_test = daten[train_index], daten[test_index]
        y_train, y_test = Preis[train_index], Preis[test_index]
        y_train_lin, y_test_lin = Preis_lin[train_index], Preis_lin[te
st_index]
# Regression wird durchgeführt
        reg = LinearRegression().fit(X_train, y_train)
        test, train = pd.DataFrame(), pd.DataFrame()
# Test-Daten werden geschätzt (ersten 208 Observationen)
        test["schätzungen_test"], train["schätzungen_train"] = reg.pre
dict(X_test),reg.predict(X_train)
        test["y_test"], train["y_train"] = y_test, y_train
        test["y_test_lin"], train["y_train_lin"] = y_test_lin, y_train
_lin
# Bestimmtheitsmaß ergibt sich aus quadriertem Korrelationskoeffizient
en
        in_sample = train["schätzungen_train"].corr(train["y_train"])*
*2
        out_of_sample = test["schätzungen_test"].corr(test["y_test"])*
*2
# Hier ist der Teil der sich auf Wooldrige S.214 bezieht
        if abh.equals(y.Log_Preis):
            in_sample = train["y_train_lin"].corr(np.exp(train["schätz
ungen_train"]))**2
            out_of_sample = test["y_test_lin"].corr(np.exp(test["schät
zungen_test"]))**2
        adj_in_sample = 1-(1-in_sample)*(len(daten)-1)/(len(daten)-(le
n(unabh.columns))-1)
        adj_out_of_sample = 1-(1-out_of_sample)*(len(daten)-1)/(len(da
ten)-(len(unabh.columns))-1)
        r_sq_preis.append(["R2-train,{} Regression:".format(i), in_sam
ple,"R2-test,{} Regression:".format(i),out_of_sample])
        gesamt_rsqu.append(out_of_sample)
        gesamt_adj_rsqu.append(adj_out_of_sample)
        i+=1
    for Regression in r_sq_preis:
        print(Regression)
    print("\nDurchschnitt Bestimmtheitsmaß Test-Datensätze:", sum(gesa
mt_rsqu) / len(gesamt_rsqu))
    print("\nDurchschnitt adj. Bestimmtheitsmaß Test-Datensätze:", sum
(gesamt_adj_rsqu) / len(gesamt_adj_rsqu))
```

Der Datensatz wurde in fünf Gruppen unterteilt. Für die Gruppen ergeben sich also 208 Observationen, für die letzte Gruppe 207. Es wird anhand von Modell1 und Modell2 überprüft, ob die oben erstellte Funktion den Werten einer Regression in einem anderen Python-Modul entsprechen

Die folgenden Schritte dienen lediglich dazu, zu überprüfen, ob die obenstehende Funktion *kreuzval* richtig erstellt wurde. Angewendet wird die Funktion darunter

Überprüfung Modell1

In [1340]:
```
kreuzval(x_zentriert1, y.Preis, False)
```

```
['R2-train,1 Regression:', 0.8969433643691942, 'R2-test,1 Regressio
n:', 0.8106956559369656]
['R2-train,2 Regression:', 0.8908369972930695, 'R2-test,2 Regressio
n:', 0.8561903991321893]
['R2-train,3 Regression:', 0.891639149375759, 'R2-test,3 Regression:',
0.8574012796270066]
['R2-train,4 Regression:', 0.8857818551784511, 'R2-test,4 Regressio
n:', 0.8759368104109732]
['R2-train,5 Regression:', 0.8698654430437649, 'R2-test,5 Regressio
n:', 0.9257673794859109]

Durchschnitt Bestimmtheitsmaß Test-Datensätze: 0.8651983049186092

Durchschnitt adj. Bestimmtheitsmaß Test-Datensätze: 0.859654804920277
```

Die erste Regression "['R2-train,1 Regression:', 0.8969433643691942, 'R2-test,1 Regression:', 0.8106956559369656]" wird abgeglichen mithilfe eines anderen Python-Moduls, welches bisher Anwendung gefunden hat. Es werden also die Ergebnisse von Statsmodels mit denen von Scikit verglichen:

In [1341]:
```
def überprüfung():
    model = sm.OLS(y.Preis.tail(831), x_zentriert1.tail(831))
    mod = model.fit()
    print("R2-train,1 Regression:", mod.rsquared)
    print("R2-test,1 Regression:",y.Preis.head(208).corr(mod.predict(x
_zentriert1.head(208)))**2)
überprüfung()
```

```
R2-train,1 Regression: 0.8969433643691942
R2-test,1 Regression: 0.8106956559046014
```

Die Regressionen beider Module liefern also dasselbe Ergebnis

Überprüfung Modell2

In [1342]:
```
kreuzval(x_zentriert1, y.Log_Preis, False)
```

```
['R2-train,1 Regression:', 0.8883469556847965, 'R2-test,1 Regressio
n:', 0.815091409334155]
['R2-train,2 Regression:', 0.8842523156347785, 'R2-test,2 Regressio
n:', 0.8354759195081171]
['R2-train,3 Regression:', 0.886029230025885, 'R2-test,3 Regression:',
0.8511846718444118]
['R2-train,4 Regression:', 0.8787533202027488, 'R2-test,4 Regressio
n:', 0.8778489287144409]
['R2-train,5 Regression:', 0.8652112488264458, 'R2-test,5 Regressio
n:', 0.9195239750215249]

Durchschnitt Bestimmtheitsmaß Test-Datensätze: 0.85982498088453

Durchschnitt adj. Bestimmtheitsmaß Test-Datensätze: 0.8540605116932216
```

Die erste Regression ['R2-train,1 Regression:', 0.8883469556847965, 'R2-test,1 Regression:', 0.815091409334155] wird wieder abgeglichen

In [1343]:
```
def andere_überprüfung():
    model = sm.OLS(y.Log_Preis.tail(831), x_zentriert1.tail(831))
    mod = model.fit()
    temp = pd.DataFrame()
    temp["fit"]= np.exp(mod.predict(x_zentriert1)).head(208)
    print("R2-train,1 Regression:", y.Preis.tail(831).corr(np.exp(mod.
fittedvalues))**2)
    print("R2-test,1 Regression", y.Preis.head(208).corr(temp.fit)**2)
andere_überprüfung()
```

```
R2-train,1 Regression: 0.8883469557314146
R2-test,1 Regression 0.8150914093093111
```

Es ergeben sich für sowohl Lin-Lin (oder Modell1) als auch Log-Lin (Modell2) dieselben Werte , weshalb kann angenommen werden, dass die Funktion richtig erstellt wurde. **Da sich der Datensatz allerdings immer noch in der selben Reihenfolge befindet, in der er von Alternate.de runtergeladen wurde, und dort vermutlich die Daten nicht in zufälliger Reihenfolge angeboten werden, werden die Klassen der Kreuzvalidierung zufällig gewählt. Was auffällig ist: Die Preise der Laptops, die auf den hinteren Seiten aufgeführt werden, lassen sich besser erklären**

Die Ergebnisse der oben genannte Funktion, die unten abgebildet sind befinden sich in der Arbeit auf Seite 27, allerdings sind die Ergebnisse anders, da die Klassen der Kreuzvalidierung zufällig gewählt werden

Modell1

zentriertes und nicht gekürztes Modell1

In [1344]:
```
kreuzval(x_zentriert1,y.Preis, True)
```

```
['R2-train,1 Regression:', 0.8896484105114979, 'R2-test,1 Regressio
n:', 0.8628156179207908]
['R2-train,2 Regression:', 0.8844745586266287, 'R2-test,2 Regressio
n:', 0.8818976301072396]
['R2-train,3 Regression:', 0.8823863308995948, 'R2-test,3 Regressio
n:', 0.8907751915939155]
['R2-train,4 Regression:', 0.8928962337739647, 'R2-test,4 Regressio
n:', 0.8464704953405596]
['R2-train,5 Regression:', 0.8833985656141589, 'R2-test,5 Regressio
n:', 0.8892361910449947]

Durchschnitt Bestimmtheitsmaß Test-Datensätze: 0.8742390252015001

Durchschnitt adj. Bestimmtheitsmaß Test-Datensätze: 0.8690673100894253
```

zentriertes und gekürztes Modell1:

In [1345]:
```
kreuzval(x_zentriert2,y.Preis, True)
```

```
['R2-train,1 Regression:', 0.8811983111117142, 'R2-test,1 Regressio
n:', 0.8904158819881294]
['R2-train,2 Regression:', 0.887175517257734, 'R2-test,2 Regression:',
0.8643937092601707]
['R2-train,3 Regression:', 0.884252948120156, 'R2-test,3 Regression:',
0.8751826842871829]
['R2-train,4 Regression:', 0.8809576132911684, 'R2-test,4 Regressio
n:', 0.8947677453270528]
['R2-train,5 Regression:', 0.8877796675373032, 'R2-test,5 Regressio
n:', 0.8613655395145804]

Durchschnitt Bestimmtheitsmaß Test-Datensätze: 0.8772251120754232

Durchschnitt adj. Bestimmtheitsmaß Test-Datensätze: 0.8740708165358588
```

Modell2

zentriertes und nicht gekürztes Modell2

In [1346]:
```
kreuzval(x_zentriert1,y.Log_Preis, True)
```

```
['R2-train,1 Regression:', 0.8753562439177369, 'R2-test,1 Regressio
n:', 0.8830340118446075]
['R2-train,2 Regression:', 0.8813073566607543, 'R2-test,2 Regressio
n:', 0.8632605456810831]
['R2-train,3 Regression:', 0.8852125740023745, 'R2-test,3 Regressio
n:', 0.8508758346152868]
['R2-train,4 Regression:', 0.8828510475270008, 'R2-test,4 Regressio
n:', 0.8643632639286921]
['R2-train,5 Regression:', 0.8760413596675257, 'R2-test,5 Regressio
n:', 0.8774602059362812]

Durchschnitt Bestimmtheitsmaß Test-Datensätze: 0.8677987724011903

Durchschnitt adj. Bestimmtheitsmaß Test-Datensätze: 0.8623622123896043
```

zentriertes und gekürztes Modell2:

In [1347]:
```
kreuzval(x_zentriert3,y.Log_Preis, True)
```

```
['R2-train,1 Regression:', 0.8799449568515673, 'R2-test,1 Regressio
n:', 0.8742506518066501]
['R2-train,2 Regression:', 0.8760229601457507, 'R2-test,2 Regressio
n:', 0.895028821487362]
['R2-train,3 Regression:', 0.8828470437854512, 'R2-test,3 Regressio
n:', 0.8654184871799119]
['R2-train,4 Regression:', 0.8798796974815367, 'R2-test,4 Regressio
n:', 0.8666743255016278]
['R2-train,5 Regression:', 0.8849410582764706, 'R2-test,5 Regressio
n:', 0.8573491020544027]

Durchschnitt Bestimmtheitsmaß Test-Datensätze: 0.8717442776059908

Durchschnitt adj. Bestimmtheitsmaß Test-Datensätze: 0.8676645727187063
```

Modell3

zentriertes und nicht gekürztes Modell3

In [1348]:
```
kreuzval(x_neu2,y.Log_Preis, True)
```

```
['R2-train,1 Regression:', 0.8641113464002071, 'R2-test,1 Regressio
n:', 0.8530326808976857]
['R2-train,2 Regression:', 0.8698124224683011, 'R2-test,2 Regressio
n:', 0.8347622607904829]
['R2-train,3 Regression:', 0.8595201035682958, 'R2-test,3 Regressio
n:', 0.8792538930569677]
['R2-train,4 Regression:', 0.8658635504044163, 'R2-test,4 Regressio
n:', 0.8525283636075279]
['R2-train,5 Regression:', 0.8613818217955702, 'R2-test,5 Regressio
n:', 0.8540013090463068]

Durchschnitt Bestimmtheitsmaß Test-Datensätze: 0.8547157014797943

Durchschnitt adj. Bestimmtheitsmaß Test-Datensätze: 0.8500943321431673
```

zentriertes und gekürztes Modell3

In [1349]:
```
kreuzval(x_neu3,y.Log_Preis, True)
```

```
['R2-train,1 Regression:', 0.8576793783113561, 'R2-test,1 Regressio
n:', 0.8822499671302843]
['R2-train,2 Regression:', 0.8668818347531115, 'R2-test,2 Regressio
n:', 0.8447802307614317]
['R2-train,3 Regression:', 0.8606649167489482, 'R2-test,3 Regressio
n:', 0.8732670032883489]
['R2-train,4 Regression:', 0.8662140419469013, 'R2-test,4 Regressio
n:', 0.846197137420691]
['R2-train,5 Regression:', 0.8656822642240113, 'R2-test,5 Regressio
n:', 0.8506122997688251]

Durchschnitt Bestimmtheitsmaß Test-Datensätze: 0.8594213276739161

Durchschnitt adj. Bestimmtheitsmaß Test-Datensätze: 0.8563773013046507
```

Modellauswahl und Darstellung

Erstellen einer Gesamtübersicht für Modell1

Zurück zu Induktive Statistik

Nach den Bestimmtheitsmaßen der Kreuzvalidierungen bezüglich des Preises zu schließen, kann Modell1 den Zusammenhang der Merkmale und dem Preis am besten erklären.

In den folgenden Schritten wird ein Schaubild zur Darstellung der Ergebnisse erstellt. Das Resulat der folgenden Schritte wurde **hier** in der Funktion *Übersicht* zusammengefasst.

In [1350]:
```
xl = pd.DataFrame()
xl["fit_mod_lin0_z"] = x.fit_mod_lin1_z
xl["Preis"] = y.Preis
```

In [1351]:
```
x
```

Out[1351]:

Laptop_ID	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	Speicher_Grafik	extern_Schnitts
2	2	2600	8	15.6	0	
4	6	2200	8	15.6	6144	
...	...	...	...	...	...	
1747	4	2300	8	15.6	4096	
1748	4	2300	16	15.6	6144	

1039 rows × 90 columns

In [1352]:
```
modell_parameter = mod_lin1_z.params.to_frame(name="beta").reset_index
().rename_axis(None).rename(columns={"index": "Merkmal"})
modell_parameter
```

Out[1352]:

	Merkmal	beta
0	Kerne_z	94.194187
1	Kerne_z_quad	12.769128
...	...	...
24	Typ_Arbeitsspeicher_DDR_4	-108.885931
25	Konstante	1171.141911

26 rows × 2 columns

In [1353]:
```
for Merkmal in list(modell_parameter.Merkmal):
    xl["{}_geschätzt".format(Merkmal)]= x[Merkmal].apply(lambda x: x*m
odell_parameter[modell_parameter.Merkmal == Merkmal].beta)
```

Die Merkmalsausprägungen werden mit den zugehörigen Paramtern multipliziert, um den Anteil der Schätzungen zu bekommen

Für das Merkmal _Kernez des ersten Merkmalsträgers ergibt sich: **94,19*-2,239 = -210,89**

Für das Merkmal _Kerne_quadz des ersten Merkmalsträgers ergibt sichergibt sich: **12,77*5,012=64,00**

Für den ersten Merkmalsträger reduziert also die geringe Anzahl an Kernen den Preis um **-210,89 + 64,00 = -146,9**

Die ersten zwei Merkmalsträger aus dem ursprünglichen Datensatz:

In [1354]:
```
x.head(2)
```

Out[1354]:

Laptop_ID	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	Speicher_Grafik	extern_Schnitts
2	2	2600	8	15.6	0	
4	6	2200	8	15.6	6144	

In dieser Tabelle sind die Einzelwerte aus der eben genannten Rechnung eingetragen:

In [1355]:
```
xl.head(2)
```

Out[1355]:

Laptop_ID	fit_mod_lin0_z	Preis	Kerne_z_geschätzt	Kerne_z_quad_geschätzt	Takt_Prozessor_z_g
2	345.804516	299.0	-210.871682	63.995521	8
4	1269.259661	899.0	165.905064	39.612508	2

die Werte aus erstem und zweiten Grad der Merkmale werden zusammengefasst

In [1356]:
```
for Merkmal in list(xl.columns):
# es werden nur quadratische Merkmale betrachtet
    if "_quad" in Merkmal:
# die Merkmale werden jeweils bei _quad geteilt. Für Kerne_z_quad_gesc
hätzt ergibt sich: "Kerne_z" und "_geschätzt"
        Merkmal_neu = Merkmal.split("_quad")[0]
        try:
            xl[Merkmal_neu] = xl["{}_geschätzt".format(Merkmal_neu)]+x
l[Merkmal]
            del xl[Merkmal], xl["{}_geschätzt".format(Merkmal_neu)]
        except KeyError:
            continue
```

In dieser Tabelle sind die Summen der Werte aus den Termen ersten und zweiten Grades eingetragen (-146,9 für Kerne)

In [1357]:
```
xl.head(2)
```

Out[1357]:

Laptop_ID	fit_mod_lin0_z	Preis	Arbeitsspeicher_z_geschätzt	Bild_Display_z_quad_geschätzt	SSD
2	345.804516	299.0	-115.852365	2.694378	
4	1269.259661	899.0	-115.852365	2.694378	

Zur Veranschlaulichung wird eine Klasse für die zwei billigsten Laptops erstellt. Die beiden billigsten Laptops haben folgende Werte:

In [1358]:
```
x["Preis"] = y.Preis
x[x.Preis<220]
```

Out[1358]:

Laptop_ID	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	Speicher_Grafik	extern_Schnitts
5	2	2300	4	15.6	0	
598	2	2300	4	15.6	0	

Die beiden Laptops haben folgende geschätze Werte:

In [1359]:
```
xl[xl.Preis<220]
```

Out[1359]:

Laptop_ID	fit_mod_lin0_z	Preis	Arbeitsspeicher_z_geschätzt	Bild_Display_z_quad_geschätzt	SSD
5	52.171737	199.0	-236.222972	2.694378	
598	140.691662	219.9	-236.222972	2.694378	

In [1360]:
```
xl["Preis"] = y.Preis
preisklassen = [0,220,750,1500,2250,4000]
xl["Preisspanne"] = pd.cut(xl["Preis"], preisklassen)
gruppiert = xl.groupby("Preisspanne").sum()
gruppiert = gruppiert.replace(np.nan,0)
gruppiert
```

Out[1360]:

Preisspanne	fit_mod_lin0_z	Preis	Arbeitsspeicher_z_geschätzt	Bild_Display_z_quad_geschätz
(0, 220]	192.863399	418.90	-472.445943	5.38875
(220, 750]	163221.058488	148371.54	-31250.132702	3268.86442
(750, 1500]	517685.093578	489023.38	-19860.686730	8254.77548
(1500, 2250]	353902.859695	378148.10	11879.964884	3480.69300
(2250, 4000]	349851.044735	368891.00	39703.300490	1262.44798

Die Merkmalsausprägungen lassen sich untergliedern in solche, die den Preis erhöhen (anteil_positiv) und solche, die den Preis verringern (anteil_negativ), der Preis, der zum Klassieren verwendet wurde, muss vorher noch gelöscht werden:

In [1361]:
```
gruppiert = gruppiert.drop("Preis", axis=1)
gruppiert["anteil_positiv"] = gruppiert.where(gruppiert > 0).sum(axis=
1)
gruppiert["anteil_negativ"] = gruppiert.where(gruppiert<0).sum(axis=1)
gruppiert
```

Out[1361]:

Preisspanne	fit_mod_lin0_z	Arbeitsspeicher_z_geschätzt	Bild_Display_z_quad_geschätzt	SSD_z_g
(0, 220]	192.863399	-472.445943	5.388756	-13
(220, 750]	163221.058488	-31250.132702	3268.864427	-1251
(750, 1500]	517685.093578	-19860.686730	8254.775489	-792
(1500, 2250]	353902.859695	11879.964884	3480.693006	668
(2250, 4000]	349851.044735	39703.300490	1262.447984	1389

Der preisliche Einfluss der einzelnen Komponenten wird durch den gesamten Einfluss geteilt:

In [1362]:
```
gruppiert = gruppiert.div(gruppiert.anteil_positiv+gruppiert.anteil_negativ.abs(), axis=0)
gruppiert
```

Out[1362]:

Preisspanne	fit_mod_lin0_z	Arbeitsspeicher_z_geschätzt	Bild_Display_z_quad_geschätzt	SSD_z_g
(0, 220]	0.034204	-0.083788	0.000956	-(
(220, 750]	0.235269	-0.045044	0.004712	-(
(750, 1500]	0.389544	-0.014945	0.006211	-(
(1500, 2250]	0.446802	0.014998	0.004394	(
(2250, 4000]	0.469482	0.053280	0.001694	(

Die Merkmale _anteil*negativ* und _anteil*positiv* sollen nicht Teil der Grafik sein

In [1363]:
```
gruppiert = gruppiert.drop(["anteil_negativ", "anteil_positiv"], axis=1)
```

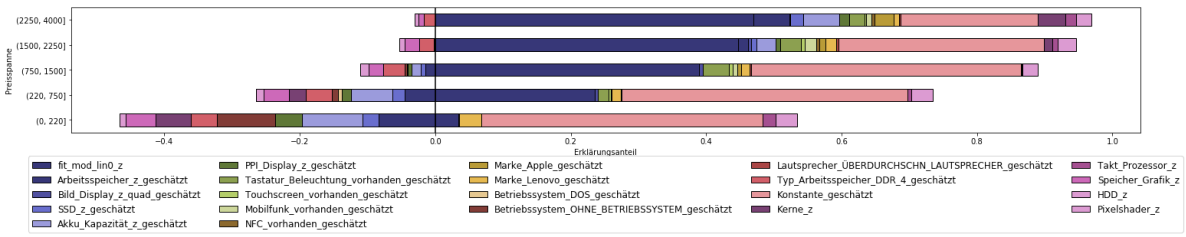
In der Summe muss sich in den einzelnen Klassen also der Wert 1 ergeben, was der Fall ist:

In [1364]:
```
gruppiert.abs().sum(axis=1).mean()
```

Out[1364]: 1.0

Aus den übrigen Merkmalen lässt sich ein Graph erstellen:

In [1365]:
```
fig,ax=plt.subplots(1,1, figsize=(25,3))
gruppiert.plot.barh(stacked= True, width=0.5, ax=ax, edgecolor= "black", legend= True, colormap="tab20b")
ax.set_xlabel("Erklärungsanteil")
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=5, fontsize="large")
ax.axvline(x=0, color ="black");
```



Der Graph ist ziemlich unübersichtlich es werden also nur Merkmale betrachtet, deren Einfluss größer als 2% des Gesamteinflusses ist.

Die Endungen "_geschätzt" und leere Spalten werden gelöscht:

In [1366]:
```
gruppiert[gruppiert.abs()<0.02] =0
gruppiert = gruppiert.loc[:, (gruppiert != 0).any(axis=0)]
for Merkmal in gruppiert.columns:
    try:
        gruppiert = gruppiert.rename(columns= {Merkmal: Merkmal.split
("_geschätzt")[0]})
    except Error:
        continue
```

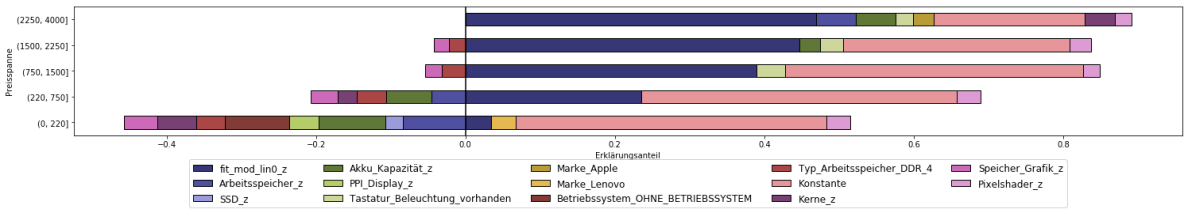
Bei den neuen Werten wurden vorherige Werte unter 0,02 durch 0 ersetzt:

In [1367]:
```
gruppiert
```

Out[1367]:

Preisspanne	fit_mod_lin0_z	Arbeitsspeicher_z	SSD_z	Akku_Kapazität_z	PPI_Display_z	Tasta
(0, 220]	0.034204	-0.083788	-0.023699	-0.089372	-0.038963	
(220, 750]	0.235269	-0.045044	0.000000	-0.061306	0.000000	
(750, 1500]	0.389544	0.000000	0.000000	0.000000	0.000000	
(1500, 2250]	0.446802	0.000000	0.000000	0.028191	0.000000	
(2250, 4000]	0.469482	0.053280	0.000000	0.053327	0.000000	

In [1368]:
```
fig,ax=plt.subplots(1,1, figsize=(25,3))
gruppiert.plot.barh(stacked= True, width=0.5, ax=ax, edgecolor= "blac
k", legend= True, colormap="tab20b")
ax.set_xlabel("Erklärungsanteil")
ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=5, fon
tsize="large")
ax.axvline(x=0, color ="black");
```



Die oben genannten Schritte lassen sich in einer Funktion zusammenfassen:

In [1369]:
```
def übersicht(fit, modell, mini,maxi):
    xl = pd.DataFrame()
    xl["fit"] = fit
    modell_parameter = modell.params.to_frame(name="beta").reset_index
().rename_axis(None).rename(columns={"index": "Merkmal"})
    for Merkmal in list(modell_parameter.Merkmal):
        xl["{}_geschätzt".format(Merkmal)]= x[Merkmal].apply(lambda x:
x*modell_parameter[modell_parameter.Merkmal == Merkmal].beta)
    for Merkmal in list(xl.columns):
    # es werden nur quadratische Merkmale betrachtet
        if "_quad" in Merkmal:
    # die Merkmale werden jeweils bei _quad geteilt. Für Kerne_z_quad_
geschätzt ergibt sich: "Kerne_z" und "_geschätzt"
            Merkmal_neu = Merkmal.split("_quad")[0]
            try:
                xl[Merkmal_neu] = xl["{}_geschätzt".format(Merkmal_ne
u)]+xl[Merkmal]
                del xl[Merkmal], xl["{}_geschätzt".format(Merkmal_ne
u)]
            except KeyError:
                continue
    xl = xl.drop("fit", axis=1)
    xl["Preis"] = y.Preis
    preisklassen = [0,750,1500,2250,4000]
    xl["Preisspanne"] = pd.cut(xl["Preis"], preisklassen)
    gruppiert = xl.groupby("Preisspanne").sum()
    gruppiert = gruppiert.replace(np.nan,0)
    gruppiert1 = gruppiert.drop("Preis", axis=1)
    gruppiert1["anteil_positiv"] = gruppiert1.where(gruppiert > 0).sum
(axis=1)
    gruppiert1["anteil_negativ"] = gruppiert1.where(gruppiert<0).sum(a
xis=1)
    gruppiert = gruppiert1.div(gruppiert1.anteil_positiv+gruppiert1.an
teil_negativ.abs(), axis=0)
    gruppiert = gruppiert.drop(["anteil_negativ", "anteil_positiv"], a
xis=1)
# Der folgende Schritt schließt Merkmale unter 2% Einfluss aus
    gruppiert[gruppiert.abs() < 0.02] = 0
    gruppiert = gruppiert.loc[:, (gruppiert != 0).any(axis=0)]
    for Merkmal in gruppiert.columns:
        try:
            gruppiert = gruppiert.rename(columns= {Merkmal: Merkmal.sp
lit("_geschätzt")[0]})
        except Error:
            continue
    fig,ax=plt.subplots(1,1, figsize=(22,3))
    gruppiert.plot.barh(stacked= True, width=0.65, ax=ax, edgecolor=
"black", legend= True, colormap="tab20b")
    ax.set_xlabel("Erklärungsanteil", fontsize="x-large")
    ax.set_ylabel("Preisspanne", fontsize="x-large")
    ax.legend(loc='upper center', bbox_to_anchor=(0.5, -0.3), ncol=5,
fontsize="x-large")
    ax.set_xlim(mini, maxi)
    ax.axvline(x=0, color ="black");
    return(gruppiert)
```

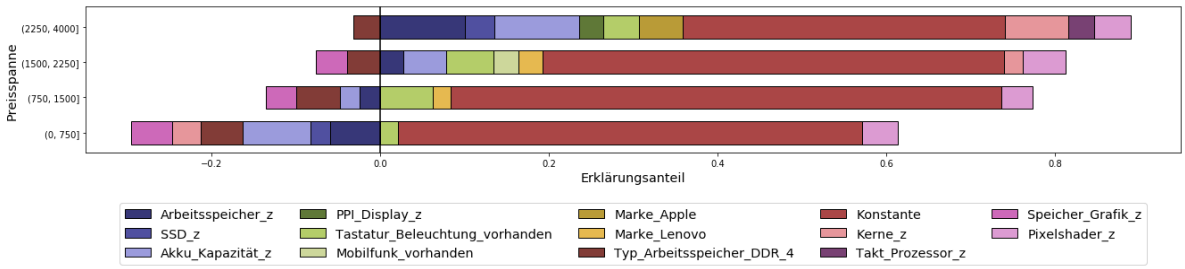
Einflussfaktoren Modell1 zentriert und reduziert

Der sich aus der Funktion ergebende Graph müsste dem obigen Graphen entsprechen, was er auch tut. Die Klasse der billigsten zwei Laptops wurde allerdings entfernt. Es sind nur noch 4 Klassen vorhanden

In [1370]:
```
übersicht(x.fit_mod_lin1_z, mod_lin1_z, -0.35, 0.95)
```

Out[1370]:

Preisspanne	Arbeitsspeicher_z	SSD_z	Akku_Kapazität_z	PPI_Display_z	Tastatur_Beleuchtung
(0, 750]	-0.059185	-0.023598	-0.080292	0.000000	
(750, 1500]	-0.024481	0.000000	-0.023460	0.000000	
(1500, 2250]	0.027112	0.000000	0.050959	0.000000	
(2250, 4000]	0.100430	0.035142	0.100519	0.027872	



Modell1 nicht zentriert, reduziert

(in der Arbeit auf Seite 36)

In [1371]:
```
x_neu4= x_neu1.drop(["Arbeitsspeicher_quad", "Bild_Display", "extern_S
chnittstellen", "SSD_quad", "Akku_Kapazität_quad", "PPI_Display_quad",
"Laufwerk_VORHANDEN", "Fingerabdrucksensor_vorhanden", "Mikrofon_Sound
_vorhanden", "Rabatt_Lehreinrichtung_vorhanden", "Akku_Typ_ION","Marke
_ASUS", "Marke_Acer", "Marke_Dell", "Betriebssystem_LINUX"], axis=1)
model = sm.OLS(y.Preis, x_neu4)
mod_lin2 = model.fit(cov_type="HC0")
x["fit_mod_lin2"] = mod_lin2.fittedvalues
x["resid_mod_lin2"] = mod_lin2.resid
print((y.Preis.corr(x.fit_mod_lin2))**2)
mod_lin2.summary()
```

```
0.8830120725397439
```

```
C:\Users\tobia\Anaconda3\lib\site-packages\statsmodels\base\model.py:1
752: ValueWarning: covariance of constraints does not have full rank.
The number of constraints is 25, but rank is 24
  'rank is %d' % (J, J_), ValueWarning)
```

Out[1371]:

OLS Regression Results

Dep. Variable:	Preis	**R-squared:**	0.883
Model:	OLS	**Adj. R-squared:**	0.880
Method:	Least Squares	**F-statistic:**	442.5
Date:	Sun, 20 Oct 2019	**Prob (F-statistic):**	0.00
Time:	18:11:31	**Log-Likelihood:**	-7235.9
No. Observations:	1039	**AIC:**	1.452e+04
Df Residuals:	1013	**BIC:**	1.465e+04
Df Model:	25		
Covariance Type:	HC0		

	coef	std err	z	P>|z|	[0.025	
Kerne	-13.3780	36.420	-0.367	0.713	-84.759	
Kerne_quad	12.6533	3.878	3.263	0.001	5.052	
Takt_Prozessor	-0.4657	0.234	-1.987	0.047	-0.925	
Takt_Prozessor_quad	0.0001	5.63e-05	2.633	0.008	3.79e-05	
Arbeitsspeicher	29.8923	2.293	13.038	0.000	25.399	
Bild_Display_quad	0.2492	0.316	0.788	0.431	-0.371	
Speicher_Grafik	0.0831	0.018	4.551	0.000	0.047	
Speicher_Grafik_quad	-1.214e-05	3.86e-06	-3.147	0.002	-1.97e-05	
HDD	0.1698	0.047	3.631	0.000	0.078	
HDD_quad	-0.0002	2.96e-05	-6.862	0.000	-0.000	
SSD	0.3329	0.045	7.393	0.000	0.245	
Akku_Kapazität	12.8978	0.954	13.514	0.000	11.027	
PPI_Display	1.7943	0.364	4.936	0.000	1.082	
Pixelshader	-0.4064	0.070	-5.790	0.000	-0.544	
Pixelshader_quad	0.0002	3.13e-05	7.567	0.000	0.000	
Tastatur_Beleuchtung_vorhanden	138.8726	19.067	7.283	0.000	101.502	1
Touchscreen_vorhanden	85.0864	26.352	3.229	0.001	33.437	1
Mobilfunk_vorhanden	341.1363	36.127	9.443	0.000	270.329	4
NFC_vorhanden	255.2760	50.111	5.094	0.000	157.061	3
Marke_Apple	515.2218	40.990	12.569	0.000	434.882	5
Marke_Lenovo	191.1857	23.038	8.299	0.000	146.032	2
Betriebssystem_DOS	-215.8241	32.415	-6.658	0.000	-279.356	-1

Betriebssystem_OHNE_BETRIEBSSYSTEM	-242.0150	31.315	-7.728	0.000	-303.391	-1
Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER	140.5240	45.210	3.108	0.002	51.914	2
Typ_Arbeitsspeicher_DDR_4	-126.4133	35.542	-3.557	0.000	-196.073	-
Konstante	-187.1016	284.946	-0.657	0.511	-745.586	3

Omnibus:	90.511	**Durbin-Watson:**	1.710
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	133.651
Skew:	0.658	**Prob(JB):**	9.51e-30
Kurtosis:	4.165	**Cond. No.**	6.30e+08

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC0)
[2] The condition number is large, 6.3e+08. This might indicate that there are strong multicollinearity or other numerical problems.

Übersicht Anzahl Laptops mit positivem Einfluss durch Merkmal:

Prozentsatz Laptops die positiv beeinflusst werden durch Komponente (In der Arbeit auf Seite 28)

In [1372]:
```
# Kerne
print(len(x[x.Kerne>1])/len(x))
# Takt_Prozessor
print(len(x[x.Takt_Prozessor>3146])/len(x))
# Speicher_Grafik
print(len(x[x.Speicher_Grafik<6925])/len(x))
# HDD
print(len(x[x.HDD<836])/len(x))
# Pixelshader
print(len(x[x.Pixelshader>1714])/len(x))
```

```
1.0
0.0009624639076034649
0.9653512993262753
0.8113570741097209
0.06352261790182868
```

In [1373]:
```
bil_m  =x.Bild_Display.min()**2*0.249181
bil_me =x.Bild_Display.mean()**2*0.249181
ppi_m  =x.PPI_Display.min()*1.794331
ppi_me = x.PPI_Display.mean()*1.794331
(bil_me+ppi_me)-(bil_m+ppi_m)
```

Out[1373]:
```
139.23821594213842
```

In [1374]:
```
91.11882998171846*-0.406427+91.11882998171846**2*0.000237
```

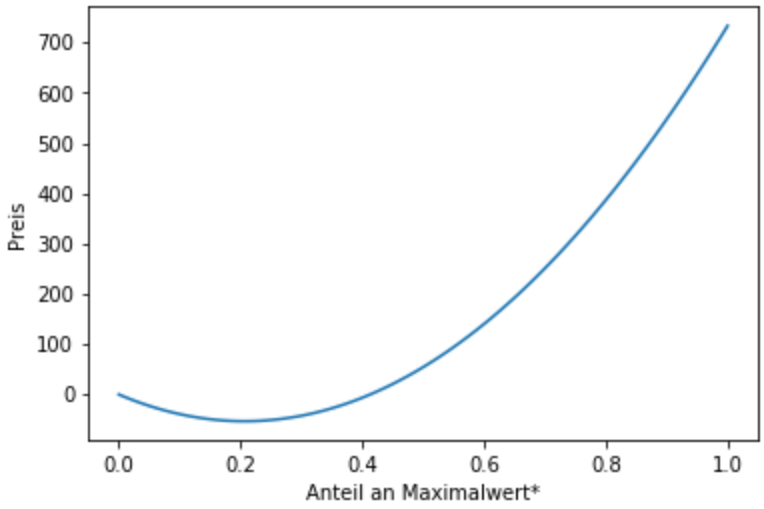
Out[1374]:
```
-35.06542675397465
```

Einfluss Grafikkarte (In der Arbeit auf Seite 29)

In [1375]:
```
p_max = x.Pixelshader.max()
s_max = x.Speicher_Grafik.max()

def pixel(wert):
    Pixel = wert*-0.406427+wert**2*0.000237
    return(Pixel)
def gr_speicher(wert):
    grspeicher = wert*0.083123+wert**2*-0.000012
    return(grspeicher)
def grafik(s1,p1):
    return(gr_speicher(s1)+pixel(p1))
def schätzungen_grafik(anzahl):
    x_achse = []
    y_achse = []
    for x in np.linspace(0,1,anzahl):
        s = s_max*x
        p = p_max*x
        x_achse.append(x)
        y_achse.append(grafik(s,p))
    plt.plot(x_achse,y_achse)
    plt.ylabel("Preis")
    plt.xlabel("Anteil an Maximalwert*")
    liste = [x for _,x in sorted(zip(y_achse,x_achse))][0]
    plt.figtext(0, -0.05, "*für sowohl Speicher_Grafik als auch Pixels
hader", horizontalalignment='left')
    plt.savefig("grafikkarte_graph", bbox_inches = "tight")
    print(liste)
schätzungen_grafik(100)
print(grafik(s_max*0.2,p_max*0.2),)
```

```
0.20202020202020204
-53.16329984000001
```

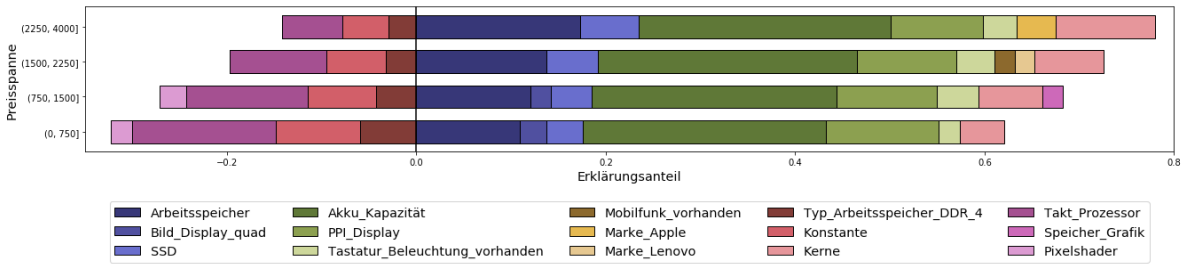


*für sowohl Speicher_Grafik als auch Pixelshader

In [1376]:
```
def einf_prz(zahl):
    Einfluss = (-13.377970*x[x.Kerne==zahl].Kerne+12.653288*x[x.Kerne==zahl].Kerne_quad-0.465746*
                x[x.Kerne==zahl].Takt_Prozessor+0.000148*x[x.Kerne==zahl].Takt_Prozessor_quad).mean()
    return(Einfluss)
Kerne2 = einf_prz(2)
Kerne4 = einf_prz(4)
Kerne6 = einf_prz(6)
Kerne8 = einf_prz(8)
print(x[x.Kerne>=6].Preis.mean())
print(Kerne4)
print(Kerne4-Kerne2)
print(Kerne6-Kerne4)
print(Kerne8-Kerne6)
print(Kerne8-Kerne2)
```

```
2185.9456962025315
-196.10027502571975
66.92373992463487
334.00904407835174
287.2538958045108
688.1866798074975
```

In [1377]:
```
gruppiert2 = übersicht(x.fit_mod_lin2, mod_lin2, -0.35, 0.8)
plt.savefig("modell1_nichtz_r", bbox_inches = "tight")
```



In [1378]:
```
gruppiert2
```

Out[1378]:

Preisspanne	Arbeitsspeicher	Bild_Display_quad	SSD	Akku_Kapazität	PPI_Display	Tastatur_
(0, 750]	0.109191	0.028149	0.038411	0.256775	0.118935	
(750, 1500]	0.120118	0.021516	0.043874	0.258079	0.105862	
(1500, 2250]	0.137352	0.000000	0.054650	0.272905	0.105089	
(2250, 4000]	0.172433	0.000000	0.062529	0.265719	0.097498	

In [1379]:
```
gruppiert2.mean().sort_values()
```

Out[1379]:
```
Takt_Prozessor    -0.111300
Konstante         -0.068196
                      ...
Arbeitsspeicher    0.134774
Akku_Kapazität     0.263369
Length: 15, dtype: float64
```

Weitere Modelle die in der Arbeit nicht berücksichtigt wurden

Zurück zu Induktive Statistik

Ridge Regression

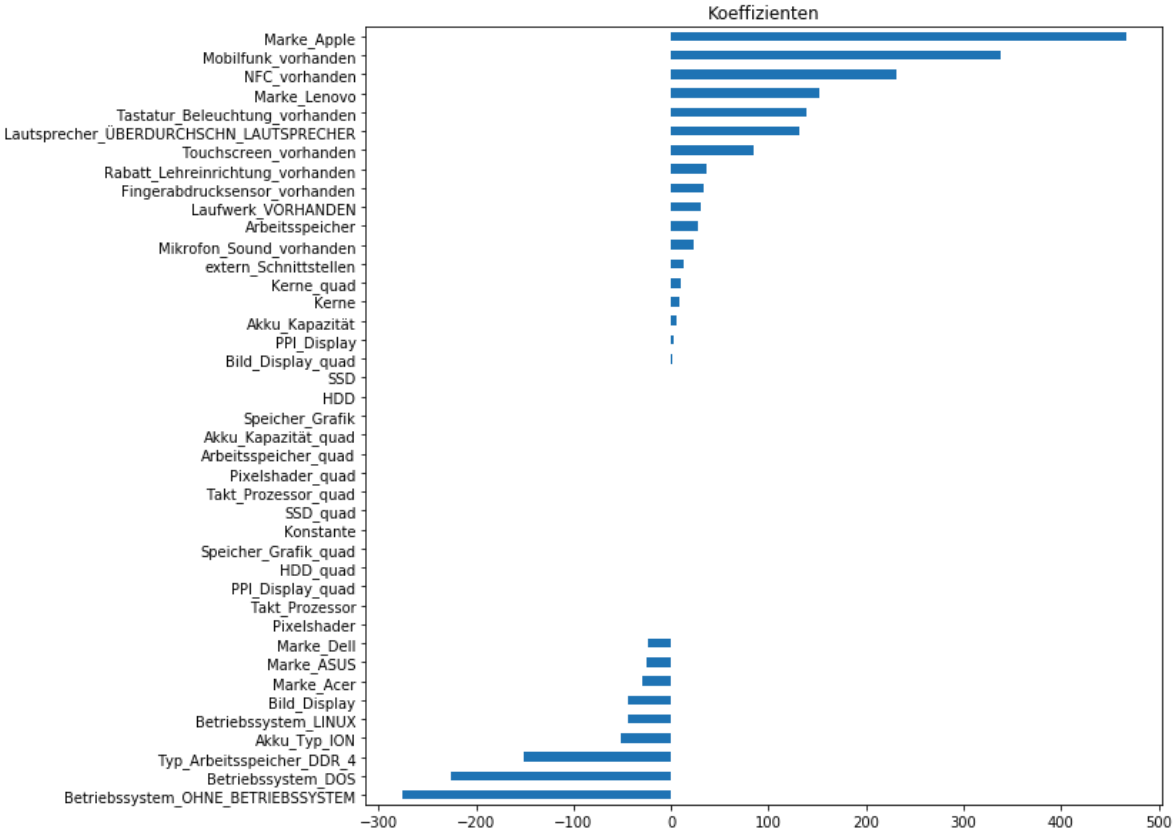
Die Ridge-Regression lieferte etwas bessere Ergebnisse als die OLS-Modelle

In [1380]:
```
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
X_train, X_test , y_train, y_test = train_test_split(x_neu1, y.Preis,
test_size=0.3, random_state=1)
alphas = 10**np.linspace(10,-2,100)*0.5
ridgecv = RidgeCV(alphas = alphas, scoring = 'neg_mean_squared_error',
normalize = True)
ridgecv.fit(X_train, y_train)
ridgecv.alpha_
print("alpha:",ridgecv.alpha_)

ridge = Ridge(alpha = ridgecv.alpha_, normalize = True)
ridge.fit(X_test, y_test)
print("Bestimmtheitsmaß für das Testset:", ridge.score(X_test, y_tes
t))

ridge.fit(x_neu1, y.Preis)
Koeffizienten = ridge.coef_
Koeffizienten_ridge = pd.Series(Koeffizienten, index = x_neu1.column
s).sort_values()
plt.gcf().set_size_inches(10,10,forward=True)
Koeffizienten_ridge.plot(kind='barh', title='Koeffizienten');
```

```
alpha: 0.005
Bestimmtheitsmaß für das Testset: 0.8938297993808044
```



Hauptkomponentenanalyse

In [1381]:
```
# from sklearn.decomposition import PCA
# pca = PCA(n_components=5)
# principalComponents = pca.fit_transform(X)
# principalDf = pd.DataFrame(data = principalComponents)
```

Regression der partiellen kleinsten Quadrate

In [1382]:
```
# from pandas import DataFrame
# from sklearn.cross_decomposition import PLSRegression

# def regPLS1(X_var,y_var):
#     _COMPS_ = len(X_var.columns) # all latent variables
#     model = PLSRegression(_COMPS_).fit( X_var, y_var )
#     return model.coef_
# regPLS1(X_log0,Log_Preis0)
```

In [1383]:
```
# from sklearn.cross_decomposition import PLSRegression
# from sklearn.metrics import mean_squared_error, r2_score
# from sklearn.model_selection import cross_val_predict

# pls = PLSRegression(n_components=10)

# # Fit
# pls.fit(X_log0,Log_Preis0)

# # Cross-validation
# y_cv = cross_val_predict(pls,X_log0,Log_Preis0, cv=10)

# # Calculate scores
# score = r2_score(Log_Preis0, y_cv)
# mse = mean_squared_error(Log_Preis0, y_cv)
# print(score)
```

In [1384]:
```
def parameter_einfl(modell_):
    parameter_einfluss = pd.DataFrame()
    modell = modell_
    basis0_koef = modell.params.to_frame(name="beta").reset_index().re
name_axis(None).rename(columns={"index":"Merkmal"})
    for Merkmal in basis0_koef.Merkmal:
        try:
            Koeffizient = basis0_koef[basis0_koef.Merkmal == Merkmal].
beta
            if ":" not in Merkmal:
                df = pd.DataFrame({"Merkmal":[Merkmal],
                                   "Parameter": Koeffizient,"min. Wert:":x[Me
rkmal].min()*Koeffizient, "max. Wert":x[Merkmal].max()*Koeffizient})
                parameter_einfluss = parameter_einfluss.append(df)
            else:
                Merkmal = Merkmal.split(":")
                print(Merkmal[1])
                df = pd.DataFrame({"Merkmal":[Merkmal],
                                   "Parameter": Koeffizient,"min. Wert:":x[Me
rkmal[0]].min()*x[Merkmal[1]].min()*Koeffizient, "max. Wert":x[Merkmal
[0]].max()*x[Merkmal[1]].max()*Koeffizient})
                parameter_einfluss = parameter_einfluss.append(df)
        except KeyError: continue
        return(parameter_einfluss)
parameter_einfl(mod_lin0)
```

Out[1384]:

	Merkmal	Parameter	min. Wert:	max. Wert
0	Kerne	-13.499252	-26.998505	-107.994019

In [1385]:
```
df_neu[df_neu.Touchscreen == "vorhanden"].Bild_Display.max()
```

Out[1385]: 17.3

Verallgemeinerte Kleinste-Quadrate-Schätzung

In [1386]:
```
# x["log_u_dach_quad"]= np.log(basis_modell.resid**2)
# print(x.log_u_dach_quad.head(2))
# y, X = dmatrices("log_u_dach_quad ~ Takt_Prozessor+ Takt_Prozessor_q
uad+ Kerne_quad +\
#                  Arbeitsspeicher +\
#                  HDD + SSD + EMMC + HDD_quad + SSD_quad + EMMC_quad
+\
#                  Akku_Kapazität_quad +\
#                  PPI_Display + PPI_Display_quad +\
#                  Bild_Display*Touchscreen_vorhanden+\
#                  Pixelshader + Pixelshader_quad +\
#                  Typ_Arbeitsspeicher_DDR_4+\
#                  Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER + \
#                  Tastatur_Beleuchtung_vorhanden+\
#                  Fingerabdrucksensor_vorhanden+\
#                  Mobilfunk_vorhanden+\
#                  NFC_vorhanden+\
#                  Betriebssystem_DOS + Betriebssystem_OHNE_BETRIEBSSY
STEM+\
#                  Marke_ASUS + Marke_Acer + Marke_Apple + Marke_Dell
+ Marke_Lenovo" ,data=x, return_type='dataframe')
# model = sm.OLS(y, X)
# basis_modell_resid = model.fit()
# #basis_modell_resid.summary()
```

In [1387]:
```
# x["gewicht"] = 1/np.exp(basis_modell_resid.fittedvalues)
# y, X = dmatrices("Preis ~ Takt_Prozessor+ Takt_Prozessor_quad+ Kerne
_quad +\
#                  Arbeitsspeicher +\
#                  HDD + SSD + EMMC + HDD_quad + SSD_quad + EMMC_quad
+\
#                  Akku_Kapazität_quad +\
#                  PPI_Display + PPI_Display_quad +\
#                  Bild_Display*Touchscreen_vorhanden+\
#                  Pixelshader + Pixelshader_quad +\
#                  Typ_Arbeitsspeicher_DDR_4+\
#                  Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER + \
#                  Tastatur_Beleuchtung_vorhanden+\
#                  Fingerabdrucksensor_vorhanden+\
#                  Mobilfunk_vorhanden+\
#                  NFC_vorhanden+\
#                  Betriebssystem_DOS + Betriebssystem_OHNE_BETRIEBSSY
STEM+\
#                  Marke_ASUS + Marke_Acer + Marke_Apple + Marke_Dell
+ Marke_Lenovo" ,data=x, return_type='dataframe')
# model = sm.WLS(y, X, weights=x["gewicht"])
# basis_modell_wls = model.fit()
# basis_modell_wls.summary()
```

In [1388]:
```
# df_neu["fit1"] = basis_modell_wls.fittedvalues
# df_neu["resid1"] = basis_modell_wls.resid
# plt.scatter(df_neu.fit_mod_lin0, df_neu.fit_mod_lin0, edgecolor="bla
ck", label="OLS")
# plt.scatter(df_neu.fit1, df_neu.resid1, edgecolor="black", alpha=0.
7, label="WLS")
# plt.legend()
# plt.title("Prognosewerte für den Preis und zugehörige Residuen")
# plt.xlabel("Prognose in Euro")
# plt.ylabel("Residuum")
# plt.savefig("residuen_ols_wls");
```

Das aktuelle Sortiment

Zurück zur Gliederung

Um zu überprüfen, ob die Schätzer die Preise des neuen Sortiments noch gut beschreiben können, wurden erneut alle Daten der Laptops aus den entspreichenden Seiten geparsed. Der Datensatz wurde am 29.09 erstellt, also rund eineinhalb Monate nach dem erstellen des ersten Datensatzes. Die Vorgehensweise war dabei identisch. Nur eine Rubrik wurde umbenannt (Speicher in Datenspeicher). Auch die weiteren Schritte entsprachen weitestgehend der oben abgebildeten Vorgehensweise.

In [1389]:
```
df = pd.read_csv('edited_laptops_neu.csv', encoding='cp1252')
```

die Merkmale werden umbenannt

In [1390]:
```
df.columns = df.columns.str.replace("-", "_")
df = df.rename(columns={"Bezeichnung_Prozessor":"Prozessor",
                        "Anzahl_Prozessorkerne_Prozessor":"Kerne",
                        "Taktfrequenz_Prozessor": "Takt_Prozessor",
                        "Gesamt_Kapazität_Arbeitsspeicher": "Arbeitssp
eicher",
                        "Anzahl Objektive_Rückseite_Kamera": "Kameras_
Rückseite",
                        "SSD_Kapazität":"SSD",
                        "HDD_Kapazität":"HDD",
                        "EMMC_Kapazität":"EMMC",
                        "EMMC_Kapazität_sq": "EMMC_quad",
                        "HDD_Kapazität_sq": "HDD_quad",
                        "SSD_Kapazität_sq": "SSD_quad",
                        "Typ_Optisches Laufwerk": "Laufwerk",
                        "Gesamt_Speicher_Grafik": "Speicher_Grafik",
                        "Lautsprecher_Sound":"Lautsprecher",
                        "Layout_Tastatur_Eingabe": "Tastatur_Layout",
                        "Fingerabdrucksensor_Eingabe": "Fingerabdrucks
ensor",
                        "Touchscreen_Eingabe": "Touchscreen",
                        "Mobilefunk Standards_Konnektivität": "Mobilfu
nk",
                        "NFC_Konnektivität": "NFC",})
```

Übersicht des aktuellen Datensatzes:

In [1391]:
```
df
```

Out[1391]:

	Laptop_ID	Preis	Prozessor	Kerne	Takt_Prozessor	Arbeitsspeicher	Typ_Arbeitsspeiche
0	4	579.00	Intel® Core™ i5-8265U	4	1600	8	DDR 4
1	11	1040.95	Intel® Core™ i7-8565U	4	1800	8	DDR 4
...	...	...	...	...	...	...	...
1458	1738	2729.00	Intel® Core™ i7	6	2600	16	DDR 4
1459	1740	4279.00	Intel® Core™ i9	8	2300	32	DDR 4

1460 rows × 42 columns

In [1392]:
```
df_anzahl = df.count().rename_axis('Merkmale').reset_index(name='Anzahl').sort_values(["Anzahl"], ascending=False)
df_anzahl.tail(8)
```

Out[1392]:

	Merkmale	Anzahl
14	Betriebssystem	1460
15	extern_Schnittstellen	1460
...	...	...
34	Akku_Kapazität	1332
41	benötigt_Zubehör	1

8 rows × 2 columns

Die Observation mit der Ausprägung bei benötigt_Zubehör wird gelöscht:

In [1393]:
```
indexNames = df[df["benötigt_Zubehör"] == "Tastatur" ].index
df.drop(indexNames , inplace=True)
df = df.drop(columns="benötigt_Zubehör")
```

Anzahl an Merkmalen:

In [1394]:
```
len(df.columns)
```

Out[1394]: 41

gekürzte Kopie des berarbeiteten Datensets, ohne fehlende Werte

Anzahl an Observationen, wenn fehlende Merkmale ausgeschlossen werden:

In [1395]:
```
df_neu = df.dropna()
len(df_neu)
df_neu.set_index("Laptop_ID", inplace=True)
```

Anzahl an Observationen, wenn Duplikate ausgeschlossen werden:

In [1396]:
```
exogene_var = list(set(df_neu.columns) - set(["Laptop_ID", "Preis", "URL"]))
df_neu_exogen = df_neu[exogene_var]
```

In [1397]:
```
df_neu_exogen.drop_duplicates()
len(df_neu_exogen)
```

Out[1397]: 1262

Es sind also keine Duplikate im Datenset

Alle Merkmale haben dieselbe Anzahl

In [1398]:
```
df_anzahl_neu = df_neu.count().rename_axis('Merkmale').reset_index(name='Anzahl').sort_values(["Anzahl"], ascending=False)
df_anzahl_neu
```

Out[1398]:

	Merkmale	Anzahl
0	Preis	1262
1	Prozessor	1262
...	...	...
17	Tastatur_Beleuchtung	1262
39	Pixelshader	1262

40 rows × 2 columns

Löschen einiger Merkmale und Merkmalsausprägungen

Typ-Grafik und Modell entsprechen einander. Diese vier Merkmale sind die Einzigen, die nicht als Dummy-Variablen dargestellt werden sollen und jeweils andere Merkmale haben, die den Zusammenhang zu dem entsprechenden Preis besser erkären und werden deshalb gelöscht

In [1399]:
```
pd.set_option('display.max_colwidth', -1)
df_neu[["Prozessor", "Auflösung_Display", "Typ_Grafik", "Modell", "URL"]].head(4)
```

Out[1399]:

Laptop_ID	Prozessor	Auflösung_Display	Typ_Grafik	Modell	
4	Intel® Core™ i5-8265U	1.366 x 768 Pixel	Intel UHD Graphics 620	Intel UHD Graphics 620	https://www.alternate.de/Acer/Asp (A315-54-5 Notebook/html/product/153
11	Intel® Core™ i7-8565U	1.920 x 1.080 Pixel	NVIDIA GeForce MX250	NVIDIA GeForce MX250	https://www.alternate.de/Acer/S (SF314-55G-7 Notebook/html/product/152
14	Intel® Core™ i7-8550U	1.920 x 1.080 Pixel	Intel UHD Graphics 620	Intel UHD Graphics 620	https://www.alternate.de/HP/Probook G4-(3UP6 Notebook/html/product/150
22	Intel® Core™ i7-8665U	2.560 x 1.440 Pixel	NVIDIA Quadro P520	NVIDIA Quadro P520	https://www.alternate.de/Lenovo/Thin P43s-(20RH001 Notebook/html/product/156

In [1400]:
```
df_neu = df_neu.drop(["Prozessor", "Auflösung_Display", "Typ_Grafik", "Modell"], axis = 1)
```

Das gekürzte Datenset besteht nur aus neuen Laptops, das Merkmal Zustand wird deshalb ebenfalls gelöscht

In [1401]:
```
print("Zustand:\n", df_neu.Zustand.value_counts())
df_neu = df_neu.drop("Zustand", axis= 1);
```

```
Zustand:
 neu    1262
Name: Zustand, dtype: int64
```

die Anzahl der Datenspeicher setzt sich aus der Summe von Anzahl an SSDs und HDDs zusammen. Da ein grundlegender Unterschied zwischen diesen beiden Festplattentypen sollte, wird dieses Merkmal ebenfalls gelöscht

In [1402]:
```
df_neu[["Anzahl_Datenspeicher", "HDD_Anzahl", "SSD_Anzahl", "EMMC_Anzahl"]].head(5)
```

Out[1402]:

Laptop_ID	Anzahl_Datenspeicher	HDD_Anzahl	SSD_Anzahl	EMMC_Anzahl
4	1	0	1	0
11	1	0	1	0
14	1	0	1	0
22	1	0	1	0
26	1	0	1	0

In [1403]:
```
df_neu = df_neu.drop("Anzahl_Datenspeicher", axis=1);
```

Einzelne Marken sind selten vertreten im Datenset und werden ebenfalls gelöscht

In [1404]:
```
Anzahl_Marke = df_neu.groupby(["Marke"], as_index=False).size().to_frame("Anzahl").reset_index(level=0)
Liste = Anzahl_Marke[Anzahl_Marke.Anzahl < 44]
for Markealt in Liste.Marke:
    print("Laptops mit der Marke",Markealt, "werden gelöscht")
    df_neu = df_neu[df_neu["Marke"] != Markealt]
```

```
Laptops mit der Marke ALTERNATE werden gelöscht
Laptops mit der Marke AORUS werden gelöscht
Laptops mit der Marke Alienware werden gelöscht
Laptops mit der Marke Fujitsu werden gelöscht
Laptops mit der Marke GIGABYTE werden gelöscht
Laptops mit der Marke MSI werden gelöscht
Laptops mit der Marke Razer werden gelöscht
Laptops mit der Marke Schenker werden gelöscht
Laptops mit der Marke TrekStor werden gelöscht
Laptops mit der Marke XMG werden gelöscht
```

In [1405]:
```
len(df_neu)
```

Out[1405]: 1084

Laptops die eine eMMC Speicherkarte haben, werden gelöscht:

In [1406]:
```
df_neu = df_neu[df_neu.EMMC == 0]
del df_neu["EMMC"]
del df_neu["EMMC_Anzahl"]
del df_neu["EMMC_quad"]
```

Auch die quadratischen Terme von HDD und SSD werden gelöscht, da sich diese auf die Einzelwerte und nicht wie im ursprünglichen Datensatz auf die Summe der Einzelwerte bezieht. Beispiel: SSD_quad für die Einzelwerte 1000 GB und 256 GB ist hier 1000^2+256^2 und nicht 1256^2

In [1407]:
```
del df_neu["HDD_quad"]
del df_neu["SSD_quad"]
```

Löschen von Laptops mit einem Preis von über 4000 Euro:

In [1408]:
```
df_neu = df_neu[df_neu.Preis<=4000]
```

Die Laptops sind etwas teurer geworden:

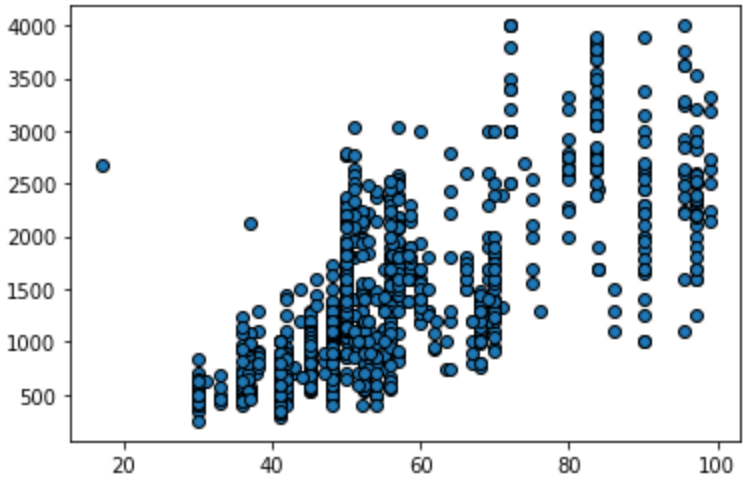
In [1409]:
```
df_neu.Preis.mean()
```

Out[1409]:
```
1412.3689092664092
```

Der Ausreißer ist noch im Sortiment und wird wieder gelöscht

In [1410]:
```
plt.scatter(df_neu.Akku_Kapazität,df_neu.Preis, edgecolor="black")
```

Out[1410]:
```
<matplotlib.collections.PathCollection at 0x266c43acdd8>
```



In [1411]:
```
df_neu = df_neu[df_neu.Akku_Kapazität > 20]
```

qualitative Merkmale

In [1412]:
```
df_neu[["Typ_Arbeitsspeicher", "Laufwerk", "Lautsprecher", "Tastatur_Beleuchtung", "Betriebssystem",
        "Fingerabdrucksensor", "Touchscreen", "Rabatt_Lehreinrichtung"]].head(2)
```

Out[1412]:

Laptop_ID	Typ_Arbeitsspeicher	Laufwerk	Lautsprecher	Tastatur_Beleuchtung	Betriebssysten
4	DDR 4	nicht vorhanden	STANDARD LAUTSPRECHER	nicht vorhanden	WINDOWS
11	DDR 4	nicht vorhanden	STANDARD LAUTSPRECHER	vorhanden	WINDOWS

In [1413]:
```
for Merkmal in df_neu[["Typ_Arbeitsspeicher", "Laufwerk", "Lautsprecher", "Tastatur_Beleuchtung",
                       "Fingerabdrucksensor", "Touchscreen", "Mobilfunk", "NFC",
                       "Akku_Typ", "Marke", "Betriebssystem", "Mikrofon_Sound", "Rabatt_Lehreinrichtung"]]:
    print(Merkmal.ljust(30,"-"), df_neu[Merkmal].unique())
```

```
Typ_Arbeitsspeicher---------- ['DDR 4' 'DDR 3']
Laufwerk--------------------- ['nicht vorhanden' 'VORHANDEN' 'Extern CD/DVD']
Lautsprecher----------------- ['STANDARD LAUTSPRECHER' 'ÜBERDURCHSCHN LAUTSPRECHER']
Tastatur_Beleuchtung--------- ['nicht vorhanden' 'vorhanden']
Fingerabdrucksensor---------- ['nicht vorhanden' 'vorhanden']
Touchscreen------------------ ['nicht vorhanden' 'vorhanden']
Mobilfunk-------------------- ['nicht vorhanden' 'vorhanden']
NFC-------------------------- ['nicht vorhanden' 'vorhanden']
Akku_Typ--------------------- ['ION' 'POLYMER']
Marke------------------------ ['Acer' 'HP' 'Lenovo' 'Dell' 'ASUS' 'Apple']
Betriebssystem--------------- ['WINDOWS' 'OHNE BETRIEBSSYSTEM' 'LINUX' 'MAC' 'DOS' '-']
Mikrofon_Sound--------------- ['vorhanden' 'nicht vorhanden']
Rabatt_Lehreinrichtung-------- ['nicht vorhanden' 'vorhanden']
```

Die Ausprägungen der qualitativen Merkmale sind identisch, lediglich in ezwei Fällen (Laufwerk und Betriebssystem) ergab sich ein neuer Wert. Die Observationen mit "Extern CD/DVD" und "-" werden deshalb gelöscht

In [1414]:
```
df_neu = df_neu[df_neu.Betriebssystem != "-"]
df_neu = df_neu[df_neu.Laufwerk != "Extern CD/DVD"]
```

In [1415]:
```
binäre_variablen = pd.get_dummies(df_neu[["Rabatt_Lehreinrichtung","Typ_Arbeitsspeicher", "Laufwerk", "Lautsprecher", "Tastatur_Beleuchtung",
                      "Fingerabdrucksensor", "Touchscreen", "Mobilfunk", "NFC",
                      "Akku_Typ", "Marke", "Betriebssystem", "Mikrofon_Sound"]])
```

In [1416]:
```
Merkmale = binäre_variablen.columns.tolist()
Merkmale = Merkmale[-1:] + Merkmale[:-1]
binäre_variablen = binäre_variablen[Merkmale]
```

In [1417]:
```
for Merkmal in Merkmale:
    binäre_variablen.columns = binäre_variablen.columns.str.replace(" ", "_")
del Merkmale
binäre_variablen.head(2)
```

Out[1417]:

Laptop_ID	Mikrofon_Sound_vorhanden	Rabatt_Lehreinrichtung_nicht_vorhanden	Rabatt_Lehreinricl
4	1	1	
11	1	1	

In [1418]:
```
binäre_variablen = binäre_variablen.drop(["Typ_Arbeitsspeicher_DDR_3", "Laufwerk_nicht_vorhanden",
                       "Lautsprecher_STANDARD_LAUTSPRECHER", "Tastatur_Beleuchtung_nicht_vorhanden",
                      "Fingerabdrucksensor_nicht_vorhanden", "Touchscreen_nicht_vorhanden",
                      "Mobilfunk_nicht_vorhanden", "NFC_nicht_vorhanden", "Betriebssystem_WINDOWS", "Betriebssystem_MAC",
                      "Akku_Typ_POLYMER", "Marke_HP",
                       "Mikrofon_Sound_nicht_vorhanden", "Rabatt_Lehreinrichtung_nicht_vorhanden"], axis=1)
```

Quantitative Merkmale

Anmerkung: Der Preis ist hier in den quantiativen Merkmalen enthalten

In [1419]:
```
quantitative_variablen = df_neu.drop(["Typ_Arbeitsspeicher", "Laufwerk", "Lautsprecher", "Tastatur_Beleuchtung",
                      "Fingerabdrucksensor", "Touchscreen", "Mobilfunk", "NFC","vorhanden_Zubehör",
                      "Akku_Typ", "Betriebssystem", "Mikrofon_Sound", "Rabatt_Lehreinrichtung",
                     "HDD_Anzahl", "SSD_Anzahl", "Marke"], axis=1)
```

Zusammenfügen qualitativer und quantitativer Merkmale

In [1420]:
```
len(binäre_variablen)
```

Out[1420]: 1031

In [1421]:
```
x = pd.concat([quantitative_variablen, binäre_variablen], axis="columns")
```

In [1422]:
```
for Marke in df_neu.Marke.unique().tolist():
    print("Durchschnittlicher Preis für die Marke", Marke, ":", df_neu[df_neu.Marke == Marke].Preis.mean())
```

```
Durchschnittlicher Preis für die Marke Acer : 1111.4503448275862
Durchschnittlicher Preis für die Marke HP : 1330.8710263157895
Durchschnittlicher Preis für die Marke Lenovo : 1461.3881516587678
Durchschnittlicher Preis für die Marke Dell : 1496.5694444444443
Durchschnittlicher Preis für die Marke ASUS : 1066.125
Durchschnittlicher Preis für die Marke Apple : 2517.7323943661972
```

Laden der alten Daten

In [1423]:
```
df_alt = pd.read_csv(r'C:\Users\tobia\Desktop\Uni\Marburg\Bachelorarbeit\Alternate\fertiges_datenset.csv')
df_alt.set_index("Laptop_ID", inplace=True)
```

In [1424]:
```
len(df_alt)
```

Out[1424]: 1039

In [1425]:
```
def xyz():
    überprüfen1 = set(x.URL)-set(df_alt.URL)
    print(len(überprüfen1))
    return(x[x['URL'].isin(list(überprüfen1))])
x = xyz().drop("URL", axis=1)
```

```
236
```

In [1426]:
```
x.head(2)
```

Out[1426]:

Laptop_ID	Preis	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	extern_Schnittstellen	Sp
31	849.0	4	1800	8	15.6	6	
91	2099.0	2	2800	8	13.3	4	

In [1427]:
```
x = x[["Preis", 'Kerne', 'Takt_Prozessor', 'Arbeitsspeicher', 'Bild_Display','Speicher_Grafik',
       'extern_Schnittstellen', 'HDD', 'SSD',
       "Akku_Kapazität", 'PPI_Display', 'Pixelshader',
       'Typ_Arbeitsspeicher_DDR_4', 'Laufwerk_VORHANDEN',
       'Lautsprecher_ÜBERDURCHSCHN_LAUTSPRECHER',
       'Tastatur_Beleuchtung_vorhanden', 'Fingerabdrucksensor_vorhanden',
       'Touchscreen_vorhanden', 'Mobilfunk_vorhanden', 'NFC_vorhanden',
       'Akku_Typ_ION', 'Marke_ASUS', 'Marke_Acer', 'Marke_Apple', 'Marke_Dell',
       'Marke_Lenovo', 'Betriebssystem_DOS', 'Betriebssystem_LINUX',
       'Betriebssystem_OHNE_BETRIEBSSYSTEM',
       'Mikrofon_Sound_vorhanden', 'Rabatt_Lehreinrichtung_vorhanden',]]
```

In [1428]:
```
x.head(2)
```

Out[1428]:

Laptop_ID	Preis	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	Speicher_Grafik	extern_
31	849.0	4	1800	8	15.6	2048	
91	2099.0	2	2800	8	13.3	0	

In [1429]:
```
df_alt = df_alt.drop("URL", axis=1)
df_alt.head(2)
```

Out[1429]:

Laptop_ID	Preis	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	Speicher_Grafik	extern_S
2	299.0	2	2600	8	15.6	0	
4	899.0	6	2200	8	15.6	6144	

In [1430]:
```
x[~x.isin(df_alt)].dropna()
```

Out[1430]:

Laptop_ID	Preis	Kerne	Takt_Prozessor	Arbeitsspeicher	Bild_Display	Speicher_Grafik	extern_
31	849.0	4.0	1800.0	8.0	15.6	2048.0	
232	1449.0	4.0	1800.0	16.0	13.3	0.0	
...	...	...	...	...	...	...	
1716	949.0	4.0	1600.0	8.0	14.0	0.0	
1728	999.0	4.0	1800.0	16.0	14.0	2048.0	

103 rows × 31 columns

In [1431]:
```
del df_alt
len(x)
```

Out[1431]: 236

erstellen quadratischer Werte und der Konstante

In [1432]:
```
x["Konstante"] = 1
x["Kerne_quad"] = x.Kerne**2
x["Takt_Prozessor_quad"] = x.Takt_Prozessor**2
x["Arbeitsspeicher_quad"] = x.Arbeitsspeicher**2
x["Bild_Display_quad"] = x.Bild_Display**2
x["Speicher_Grafik_quad"] = x.Speicher_Grafik**2
x["extern_Schnittstellen_quad"] = x.extern_Schnittstellen**2
x["HDD_quad"] = x.HDD**2
x["SSD_quad"] = x.SSD**2
x["Akku_Kapazität_quad"] = x.Akku_Kapazität**2
x["PPI_Display_quad"] =x.PPI_Display**2
x["Pixelshader_quad"] =x.Pixelshader**2
```

In [1433]:
```
x_Modell1_neu = x[x_neu4.columns]
mod_lin2_neu =  mod_lin2.predict(x_Modell1_neu)
Bestimmtheitsmaß_Preis_m2 = x.Preis.corr(mod_lin2_neu)**2
print("Bestimmtheitsmaß:", Bestimmtheitsmaß_Preis_m2)
```

```
Bestimmtheitsmaß: 0.773559330443713
```

Das Bestimmtheitsmaß der neuen Werte liegt also deutlich unter den bisherigen Bestimmtheitsmaßen von rund 0,88. Die neuen Preise werden im Durchschnitt eher unterschätzt, was möglicherweise durch eine zeitliche Komponente entstehen könnte:

In [1434]:
```
(x.Preis-mod_lin2_neu).mean()
```

Out[1434]: 125.88847525804611