DAV

DEUTSCHE
AKTUARVEREINIGUNG e.V.

Praktische Prüfung im Vertiefungswissen

Actuarial Data Science 3 / Immersion

gemäß Prüfungsordnung 4
der Deutschen Aktuarvereinigung e. V.

Zeitraum: 11.04.2022 – 11.05.2022

Hinweise zum Lösungsvorschlag:

- Die Prüfung besteht aus den drei Blöcken A, B und C. Für die Blöcke B und C ist ein Notebook (R-Markdown oder Jupyter Notebook) mit R oder Python zu erstellen. Zusätzliche Materialen (Datensätze etc.) werden mit den Prüfungsunterlagen zur Verfügung gestellt.

- Für jeden Block ist ein PDF-Dokument einzureichen. Falls ein Notebook zu erstellen ist, muss das entsprechende PDF-Dokument den Output des erstellten Codes enthalten und das Notebook muss ebenfalls eingereicht werden.

- Für jeden Block sind die Aufgaben in der vorgegebenen Reihenfolge zu bearbeiten. Die Lösung muss direkt nach der Aufgabenstellung in einer strukturierten und übersichtlichen Weise dargestellt werden.

- Die Gesamtpunktzahl beträgt 180 Punkte. Die Prüfung ist bestanden, wenn mindestens 90 Punkte erreicht werden.

ADS Immersion-Klausur Frühjahr 2022

Teil A: Allgemeine Fragen [20 Punkte]

a) Informationstheoretische Grundlagen (Lernziele 2.1.1, 2.1.2)

A1) [4 Punkte] Erklären Sie in Ihren Worten die Funktionsweise einer (deterministischen) Turing-Maschine, die die folgende Aufgabe löst: Die unbekannte Anzahl von 0-en, die zwischen 1-en auf einem Band eingeschlossen ist, soll verdoppelt werden. (Nehmen Sie hierzu an, dass das Band eine wohlgeformte Ausgangskonfiguration hat: es existieren mindestens zwei 1-en auf dem Band.)

Beispiel: Aus 10001 soll 10000001 werden.

Lösung: Als erstes wird der Lesekopf auf ein beliebiges nicht-leeres Feld gesetzt. Dieser wird anschließend, sofern möglich, nach links bewegt, bis er die 1 einreicht und diese 1 streicht. Anschließend bewegt sich der Lesekopf ein Feld nach rechts. Sofern er dort eine 1 vorfindet, geht der Lesekopf nach rechts bis zur ersten freien Zelle und schreibt dort eine 1. Das Programm ist in diesem Fall beendet (STOP). Findet der Lesekopf in dem Feld hingegen eine 0, streicht er diese ebenfalls und geht zur ersten freien Zelle des Bandes nach rechts. Dort wird eine 0 geschrieben, der Lesekopf geht eine Zeile nach rechts und schreibt ebenfalls eine 0. Schließlich wandert der Lesekopf nach links zur ersten nichtleeren Zelle nach links und der Algorithmus wiederholt sich.

A2) [5 Punkte] Erklären Sie kurz, weshalb eine positive Antwort auf das P=NP Problem für Versicherungen ein Problem bezüglich der Verschlüsselung von Daten darstellen würde. Welche zusätzliche Gefahr bezüglich dieser Problematik stellen hierüber hinaus Quantencomputer dar, über die bekannt ist, dass sie Zahlen in polynomieller Zeitkomplexität in ihre Primfaktoren zerlegen können. (Genauer: Es ist bekannt, dass für die Klasse der polynomiell entscheidbaren Quantenprobleme BQP gilt P $\subset BQP$. Ob allerdings BQP = NP gilt, ist ebenso unbekannt.) Gibt es aus kryptographischer Sicht einen Ausweg aus dieser Gefahr durch Quantencomputer?

Lösung: Die kryptographischen Verfahren, die momentan zur Verschlüsselung von Daten verwendet werden, beruhen im Wesentlichen darauf, dass das „Erraten“ des Schlüssels sehr schwierig ist. Genauer handelt es sich um ein NP-schweres Problem. Sollte ein Beweis geliefert werden, dass NP-schwere Probleme in polynomieller Zeit gelöst werden können, würde dies praktisch das Ende der meisten aktuell verwendeten Verschlüsselungsalgorithmen bedeuten. Für Versicherungen würde dies ein großes Problem bei der Übertragung verschlüsselter Daten darstellen. Sobald Quantencomputer in der Praxis eingesetzt werden können, wird die Faktorisierung großer Zahlen ohnehin in kurzer (polynomieller) Zeit lösbar und die praktische Relevanz der Frage, ob P=NP, wird erheblich sinken. Aus diesem Grund arbeiten zahlreiche auf Kryptographie spezialisierte Unternehmen an der Entwicklung von quantensicheren Verschlüsselungsalgorithmen.

A3) [2 Punkte] Ist die Funktion $\phi = \big((\neg x \vee y) \wedge (x \wedge \neg y)\big) \wedge z$ erfüllbar?

Lösung: Nein, denn egal welche Kombination man für *x, y* und *z* einsetzt, ergibt sich immer der Wert 0 für ϕ (die Kombinationen können z.B. als Tabelle überprüft werden).

b) Systemarchitektur (Lernziel 2.2.3)

A4) [5 Punkte] Betrachten Sie nochmals die Aufgabenstellung aus Teil C) zur Schadenprognose. Stellen Sie sich vor, eine entsprechende Analysemöglichkeit soll in einem Unternehmen mittels einer Softwarelösung umgesetzt werden. Diese soll auch die Möglichkeit zur Sammlung der Daten enthalten. Aufgrund der großen zu verarbeitenden Datenmenge soll die Software in der Cloud genutzt werden. Geben Sie zwei Beispiele an, worauf bei der Entwicklung der Software aus diesem Grund zu achten ist.

Lösung: Optimalerweise würde die Software aufgrund der Anforderung als „Cloud Native“ entwickelt werden. Entsprechend müsste auf Skalierbarkeit der Applikation geachtet werden, was zum Beispiel die Parallelisierung von Berechnungsmöglichkeiten voraussetzt. Darüber hinaus sollte die Software auf Microservices basieren, beispielsweise sollten Datensammlung und Analysesoftware voneinander getrennt werden. Darüber hinaus sollten die einzelnen Teile der Analyse in einzelnen Microservices umgesetzt werden. Um den parallelen Zugriff verschiedener User auf die Software zu ermöglichen, bietet sich der Einsatz von Containern an.

c) Tools & Programme (Lernziel 3.1.3)

A5) [4 Punkte] Oft ist es bei der Datenanalyse, wie beispielsweise in Teil B), notwendig, zufällige Samples zu ziehen. Aus diesem Grund werden in der aktuariellen Data Science häufig Zufallszahlgeneratoren eingesetzt. Worauf ist bei der Verwendung zur Zufallszahlgeneratoren zu achten?

Lösung: Weil die Ergebnisse der Datenanalyse reproduzierbar sein müssen, ist ein Seed für den Zufallszahlengenerator zu setzen. Darüber hinaus sollten mathematische und statistische Qualitätskriterien an den Zufallszahlengenerator angelegt werden. Dies bedeutet, dass die vom Zufallsgenerator erstellten Pseudo-Zufallszahlen nicht von echt zufälligen Zahlen unterscheidbar sein sollten.

Immersion_2022_Teil_B

April 5, 2022

1 Musterlösung Teil B Immersion 2022

Stand: 03.01.2022

1.0.1 Bibliotheken

[1]:
```
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
from tabulate import tabulate
from sklearn import preprocessing
from sklearn import metrics
import seaborn as sns
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
from tqdm import notebook
import warnings
warnings.filterwarnings('ignore')
```

1.1 Teil B-I

Einem Kfz-Versicherer ist es gelungen, fahrzeugbasierte Telemetrie seiner Kunden zu erlangen und zu aggregieren (siehe Aufgabenteile B4-6). Zur Verifikation der Aggregation beim Service-Provider wurden einzelne Abzüge der Rohdaten in der Datei `rohdaten.csv` exportiert. Das Ziel ist es, die Rohdaten in die aggregierte Form zu bringen und so die vorgenommene Transformation zu plausibilisieren.

1.1.1 Aufgabe B1 (5 Punkte, Lernziele 3.3.3, 3.4.5, 6.1.1)

Laden Sie die Datei `rohdaten.csv` und betrachten Sie zunächst nur diesen Datensatz. Wie viele Zeilen gibt es und welche semantische Struktur liegt hier vor? Welche Arten von Anreicherungen wurden vorgenommen?

Tabelle 1: Schema für rohdaten.csv

Spaltenname

Datentyp [Einheit]

Beschreibung

Time

Int16 [-]

Enumeration der Zeitreihe auf der Basis [s*10e-1], also 10Hz

Speed

Float16 [km/h]

GPS-Geschwindigkeit

Acc_X

Float16 [mG]

Longitudinale Beschleunigung

Acc_Y

Float16 [mG]

Laterale Beschleunigung

Acc_Z

Float16 [mG]

Dorsale Beschleunigung

EventCoding

Float16 [-]

Event-Encodierung

SpeedLimitCoding

Float16 [-]

Positionsbasierte Geschwindigkeitsbegrenzung

road_type

Int16 [-]

Straßentyp (0=Innerorts, 1=Außerorts, 2=Kraftfahrstraße/Autobahn)

precipitation

Int16 [-]

Positionsbasierte Niederschlagsindikation

night

Int16 [-]

Zeitbasierte Beleuchtungsindikation

rush

Int16 [-]

Zeitbasierte Verkehrsdichteindikation

IDpol

Int16 [-]

Schlüssel zur Vertragsidentifikation

1.1.2 Laden des ersten Datensatzes: Telematik-Rohdaten ("rohdaten.csv")

[2]:
```
rohdaten_df = pd.read_csv("./rohdaten.csv")
rohdaten_df
```

[2]:

	Unnamed: 0	Time	Speed	Acc_X	Acc_Y	Acc_Z	Event_Coding	\
0	0	0	2.026	108.481	-361.961	-4.640	1.0	
1	1	1	0.443	-71.785	-117.955	-7.371	1.0	
2	2	2	1.018	-89.126	301.670	7.013	1.0	
3	3	3	-1.166	-63.524	24.188	-4.548	1.0	
4	4	4	2.078	-19.656	-164.822	4.069	1.0	
…	…	…	…	…	…	…	…	
145119	145119	55136	127.676	-49.833	-76.567	6.468	0.0	
145120	145120	55137	130.746	-335.272	-220.288	-1.899	0.0	
145121	145121	55138	127.822	69.217	-104.291	-1.793	0.0	
145122	145122	55139	130.709	43.697	-327.784	4.909	0.0	
145123	145123	55140	130.690	-168.494	-389.908	-11.148	0.0	

	Speed_Limit_Coding	road_type	precipitation	night	rush	IDpol
0	0.0	1	0	0	1	13912
1	0.0	1	0	0	1	13912
2	0.0	1	0	0	1	13912
3	0.0	1	0	0	1	13912
4	0.0	1	0	0	1	13912
…	…	…	…	…	…	…
145119	0.0	1	0	0	0	3491
145120	0.0	1	0	0	0	3491
145121	0.0	1	0	0	0	3491
145122	0.0	1	0	0	0	3491
145123	0.0	1	0	0	0	3491

[145124 rows x 13 columns]

[3]:
```
rohdaten_df = rohdaten_df.drop('Unnamed: 0',axis=1)
```

IDpol-Ausprägungen:

[4]:
```
rohdaten_df['IDpol'].unique()
```

[4]:
```
array([13912, 20917,  3491], dtype=int64)
```

- Die einzelnen Zeilen bestehen aus den Spalten Unnamed: 0, Time, Speed, Acc_X, Acc_Y, Acc_Z, Event_Coding, Speed_Limit_Coding, road_type, precipitation, night, rugh, IDpol. Es ist anzunehmen, dass letztere Spalte ein Zuordnungsschlüssel ist und die erste Spalte `Unnamed: 0` zu einer entferten Information, einem verstreuten Index oder anderweitig ungültigen Format beim Export oder vorher resultiert.
- Der Datensatz besteht aus 145124 Zeilen, die offenbar zu drei verschiedenen Verträgen gehören: 3491, 13912 und 20917.
- Es gibt Rohdaten der Beschleunigung und der Geschwindigkeit, aufgezeichnet mit einer unbekannten Abtastfrequenz. (Das Schema sagt dazu, es seien 10Hz.) Dazu kommen Codings, die sich auf "Events", "Speed Limit" (Geschwindigkeitsbegrenzung), "Road Type" (Straßentyp 1/2/3), "precipiation" (Niederschlag ja/nein), "night" (Nachtfahrt ja/nein) und "rush" (Rush-Hour-Fahrt ja/nein) beziehen.

Prüfung auf fehlende Werte:

[5]:
```
rohdaten_df.isnull().sum().sum()
```

[5]:
```
0
```

Es gibt hier keine fehlenden Werte. Abgesehen von falschen Werten und möglicherweise inkorrektem Schema gibt es an dieser Stelle erstmal nichts weiter zu tun, die Betrachtung der Daten folgt in B2.

1.2 Aufgabe B2 (15 Punkte, 3.3.3, 3.4.5, 6.1.1)

Es soll eine deskriptive Analyse der Rohdaten vorgenommen werden: Wie viele Fahrten liegen vor, sind diese vollständig und wie sind die Fahrdaten in Bezug auf die Meta-Daten (Straßentyp, Wetter, Uhrzeit) im Vergleich einzuordnen? Stellen Sie eine Hypothese auf, welche Bedeutung das Merkmal `Event_Coding` aufweist. Begründen Sie dies anhand der vorliegenden Daten, auch mittels geeigneter Visualisierung.

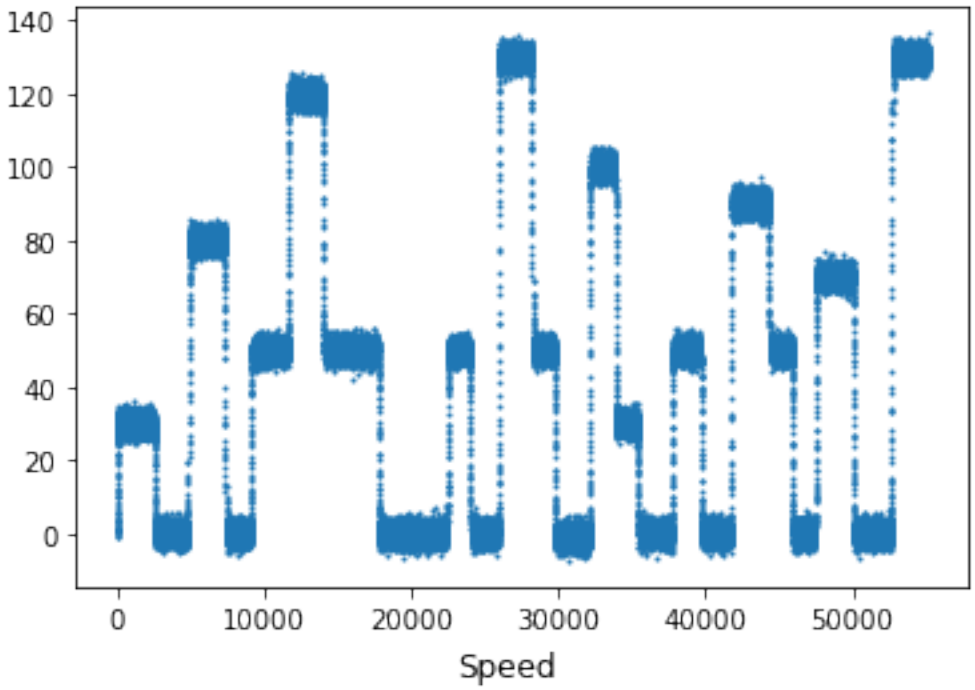
1.2.1 Antwort:

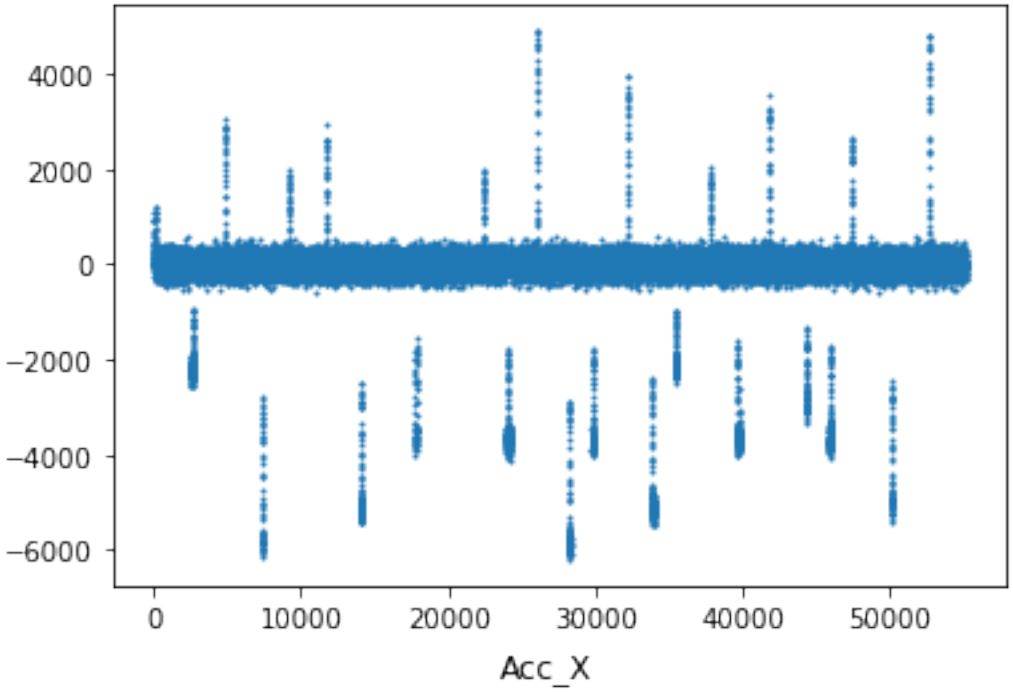
Da wir bereits festgestellt haben, dass die Daten offenbar zu mehreren Verträgen gehören, scheint es zweckmäßig, diese nach IDpol getrennt zu analysieren:

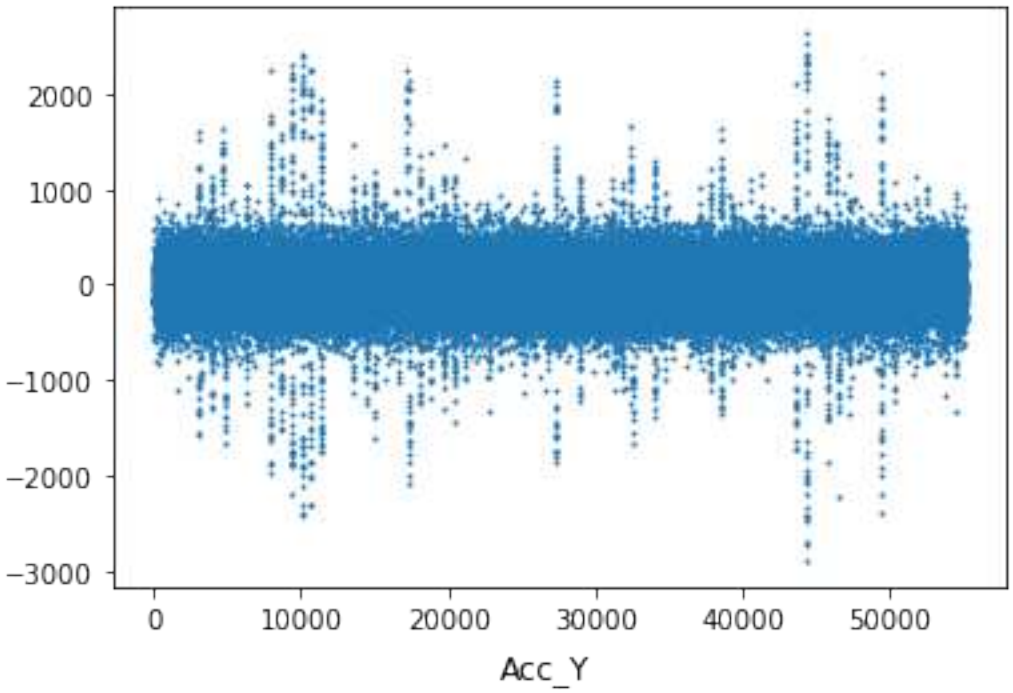
[6]:
```
trips_1 = rohdaten_df[rohdaten_df['IDpol'] == 3491]
trips_2 = rohdaten_df[rohdaten_df['IDpol'] == 13912]
trips_3 = rohdaten_df[rohdaten_df['IDpol'] == 20917]
```

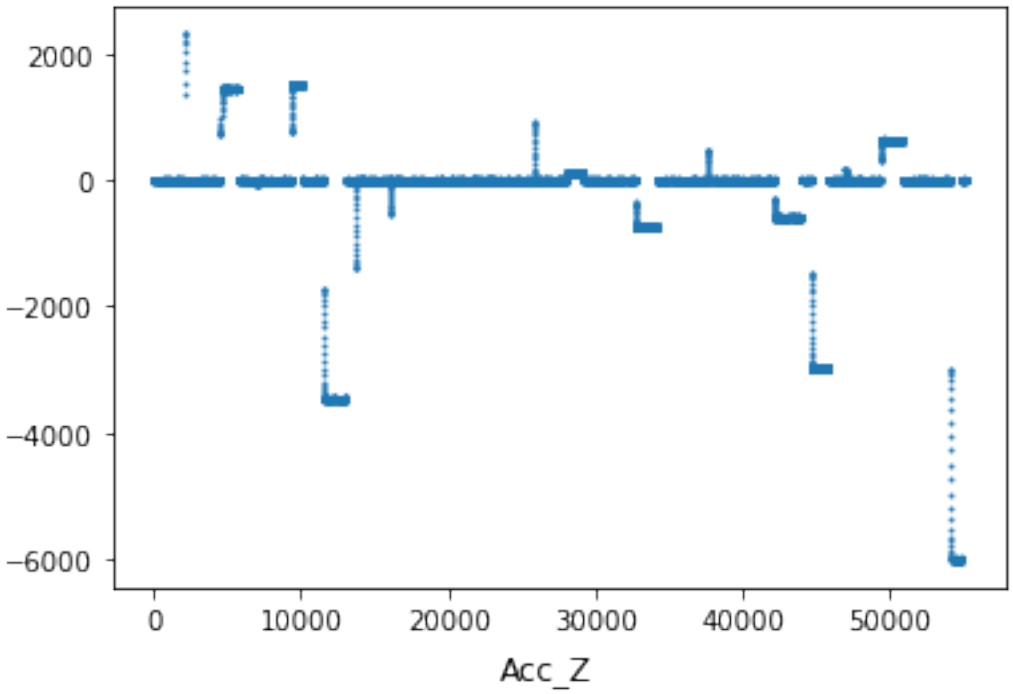
Wir wollen nun für die einzelnen Teildatensätze Scatterplots anfertigen, jeweils gegen die Zeit:

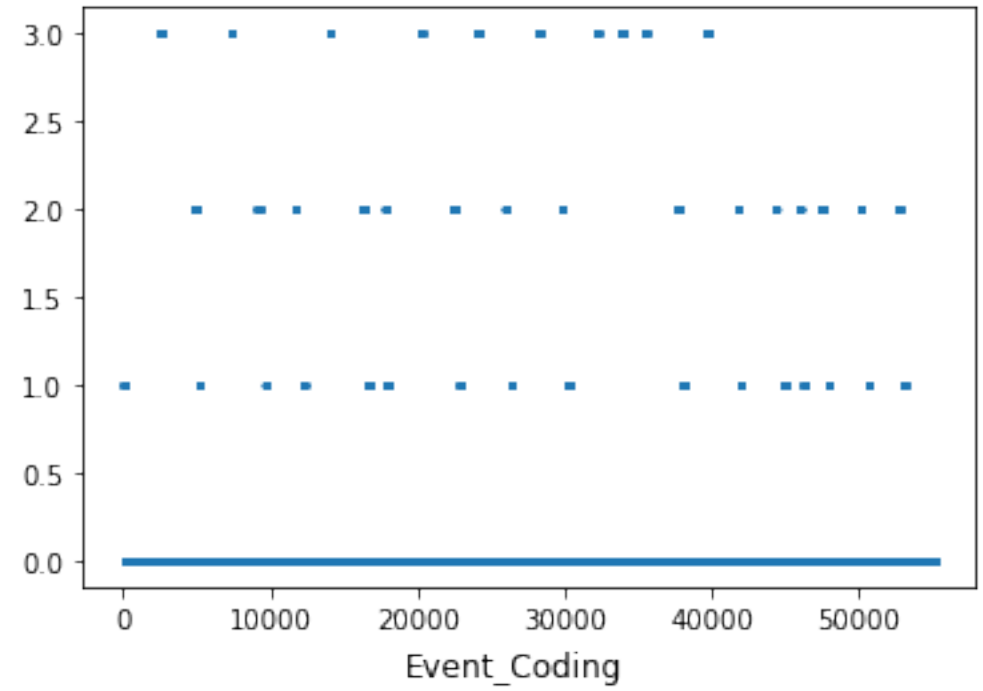
[7]:
```
# Fahrt 1
for i in range(1,11):
    figure=plt.scatter(trips_1.iloc[:,0],trips_1.iloc[:,i],s=1)
    plt.figtext(0.5, 0.01, trips_2.columns[i],wrap=True,␣
 ↪horizontalalignment='center', fontsize=12)
    plt.show()
```

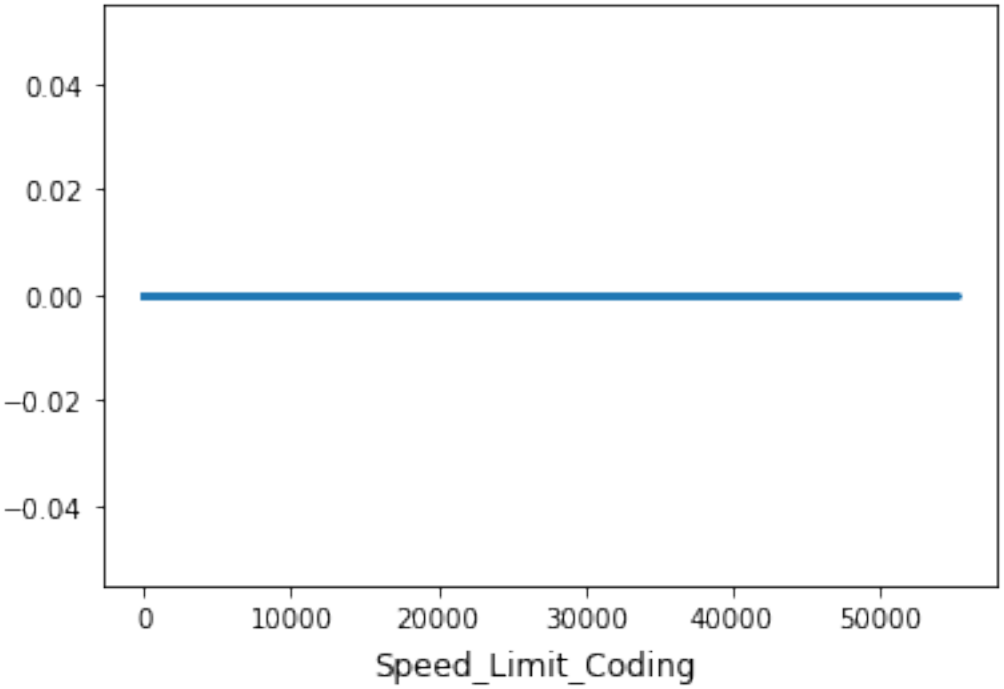

140
120
100
80
60
40
20
0
0
10000
20000
30000
40000
50000
Speed

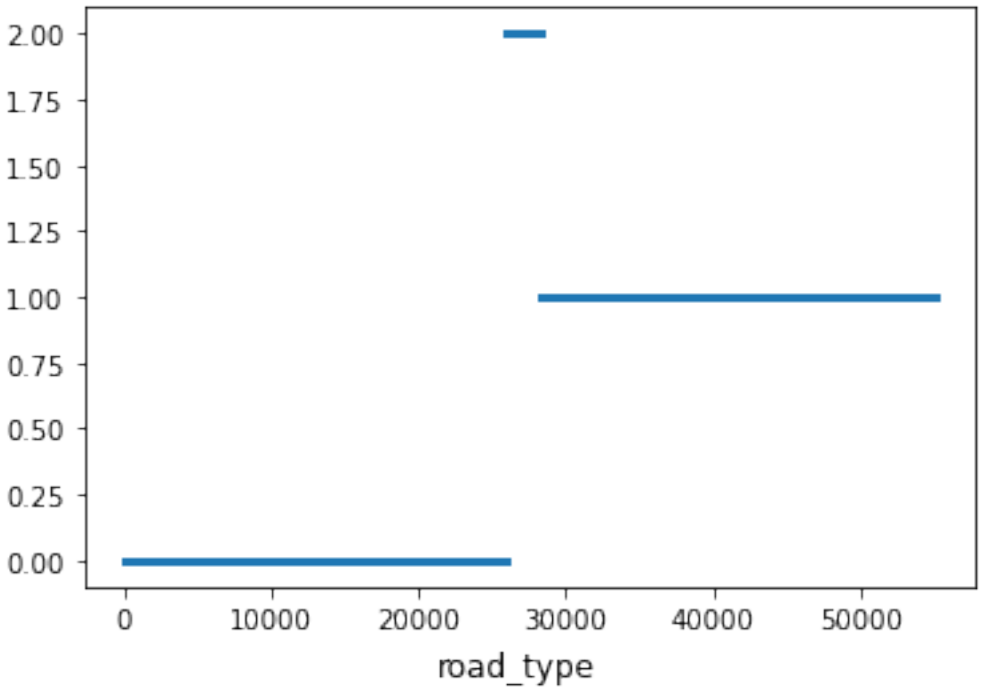

4000
2000
0
−2000
−4000
−6000
0
10000
20000
30000
40000
50000
Acc_X

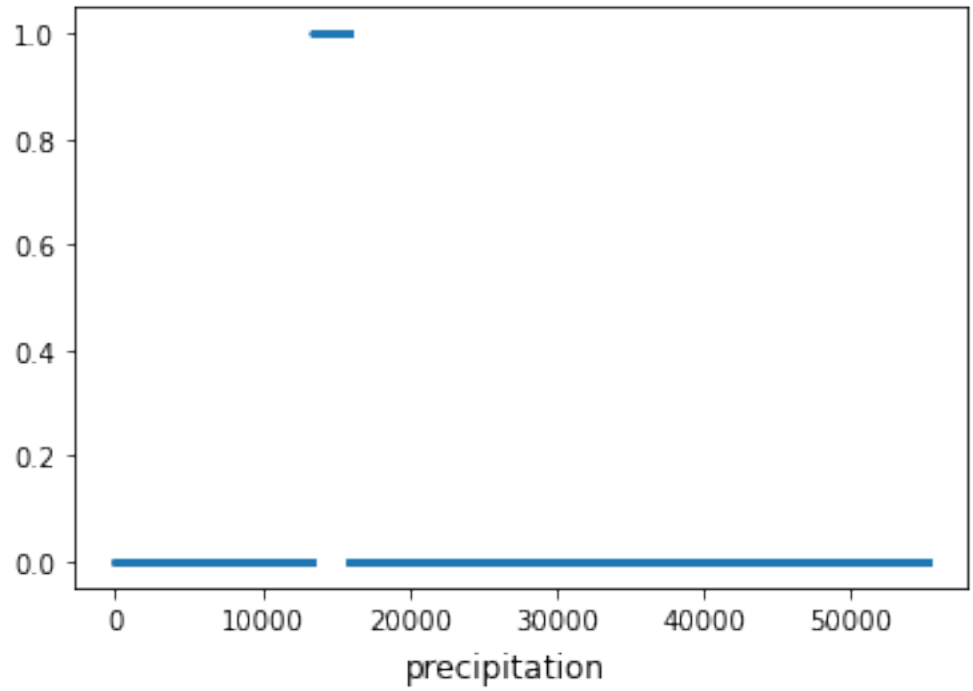

2000
1000
0
−1000
−2000
−3000
0
10000
20000
30000
40000
50000
Acc_Y

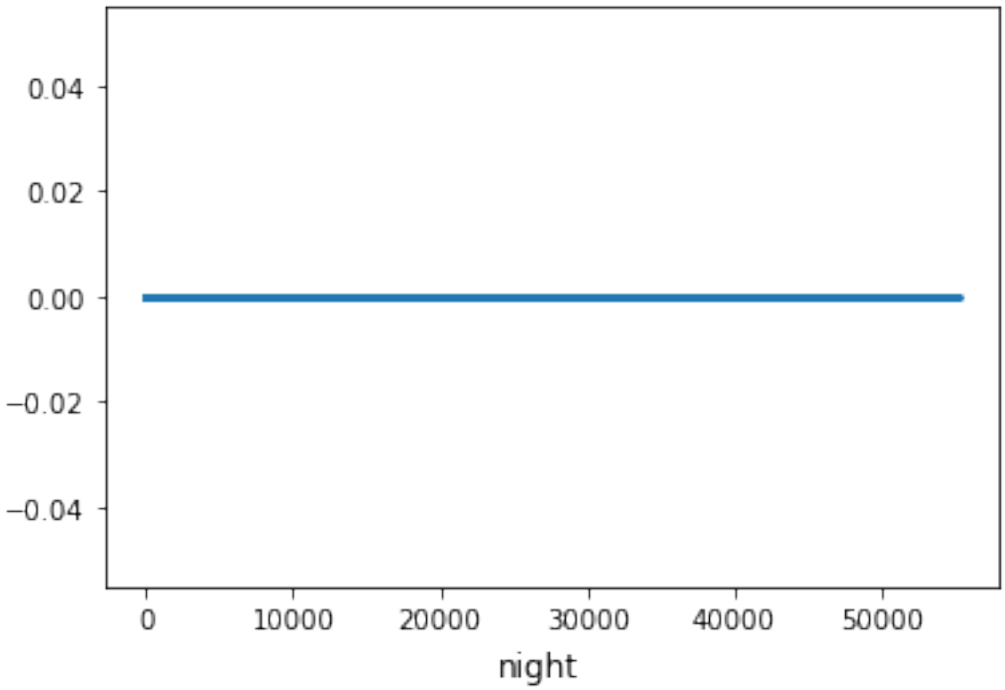

2000
0
−2000
−4000
−6000
0
10000
20000
30000
40000
50000
Acc_Z

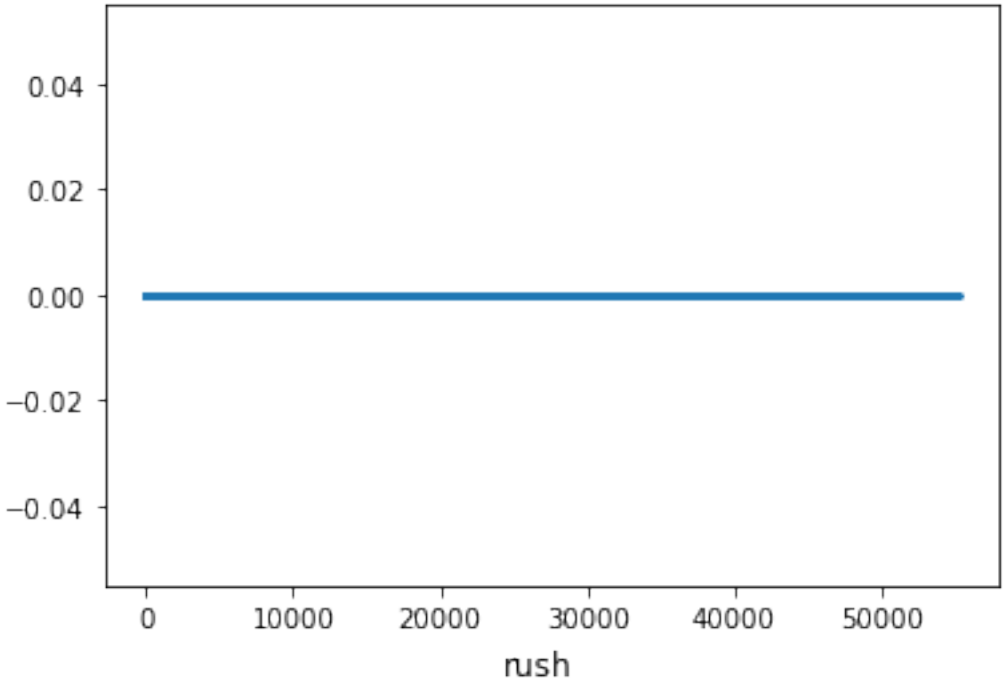

3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10000
20000
30000
40000
50000
Event_Coding


0.04
0.02
0.00
−0.02
−0.04
0
10000
20000
30000
40000
50000
Speed_Limit_Coding

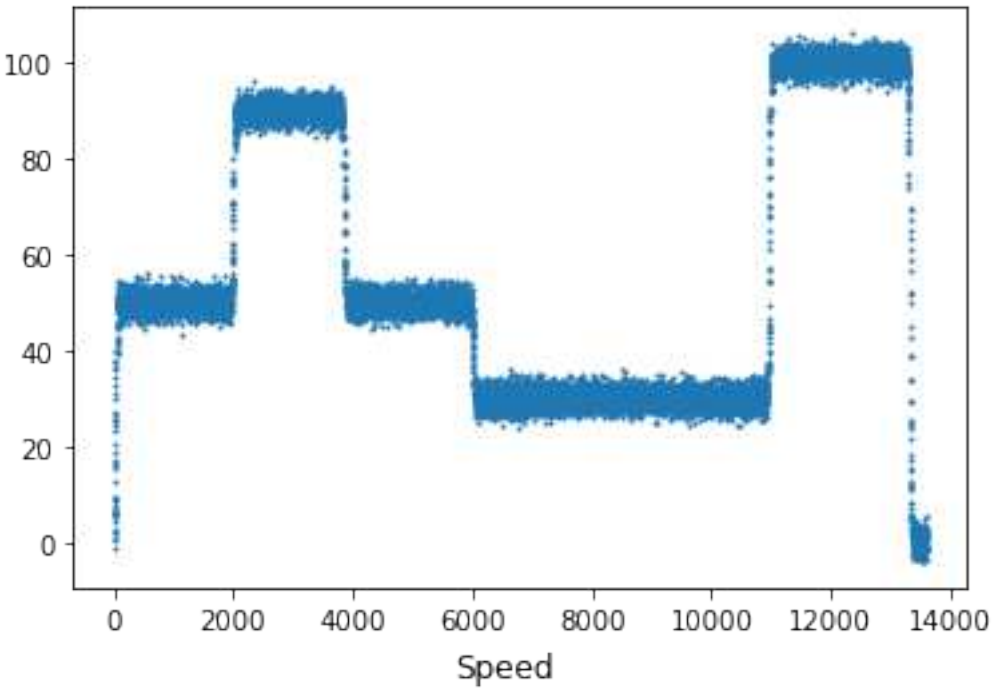

2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
10000
20000
30000
40000
50000
road_type

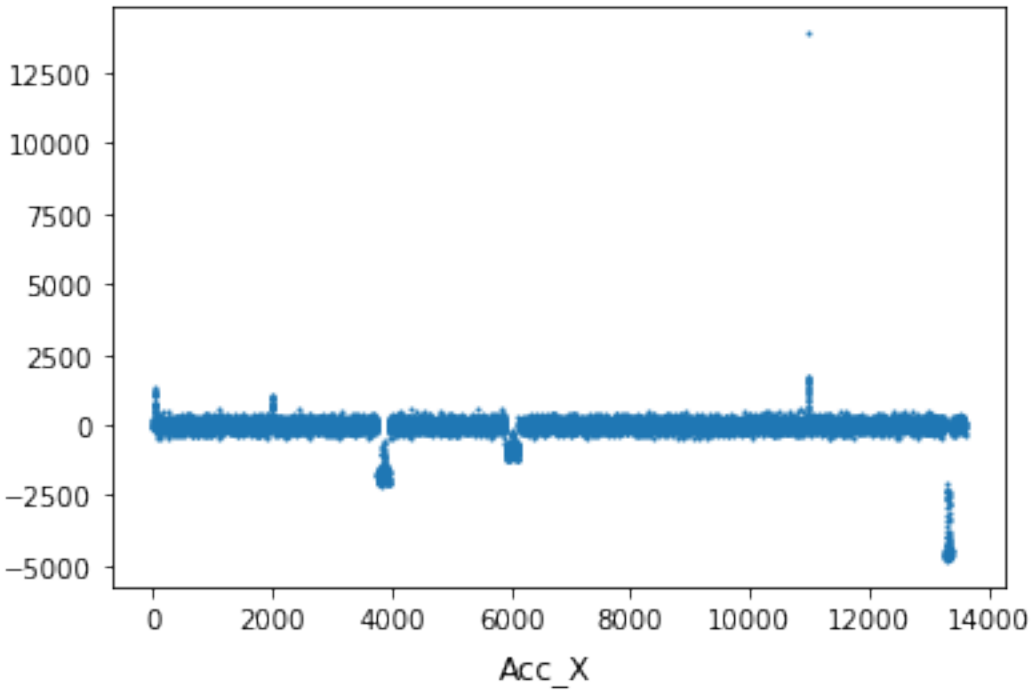

1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
precipitation

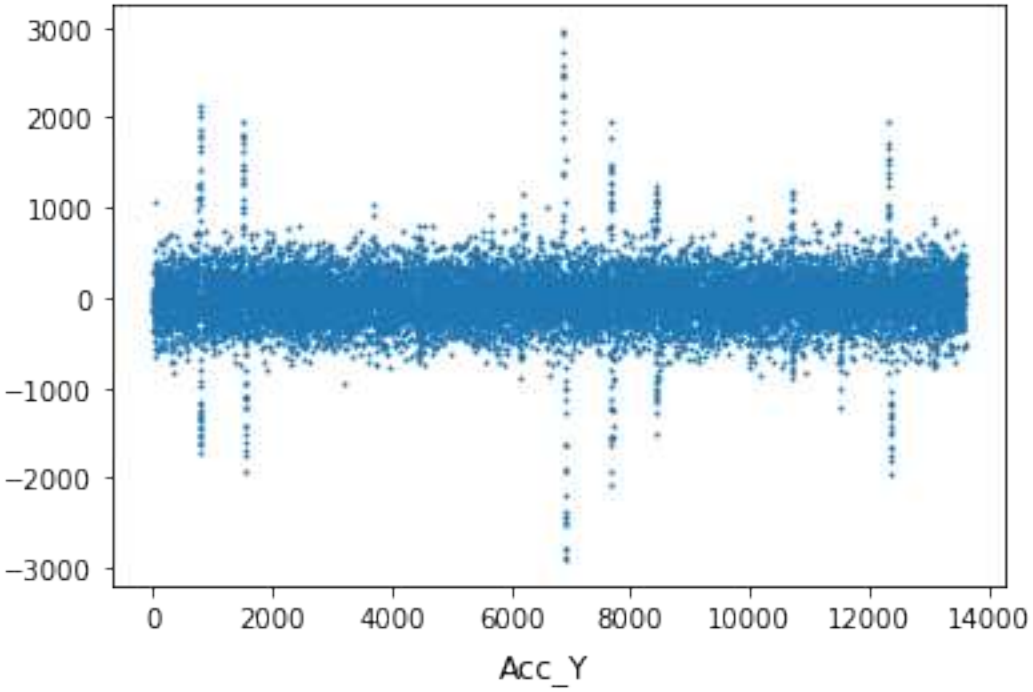

0.04
0.02
0.00
−0.02
−0.04
0
10000
20000
30000
40000
50000
night

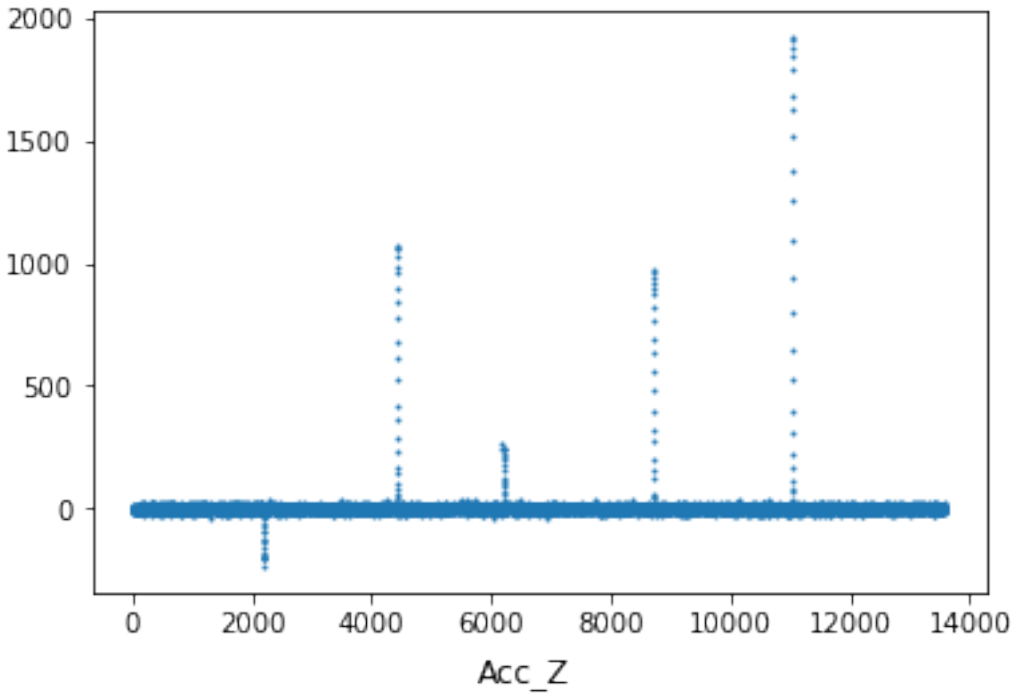

0.04
0.02
0.00
−0.02
−0.04
0
10000
20000
30000
40000
50000
rush

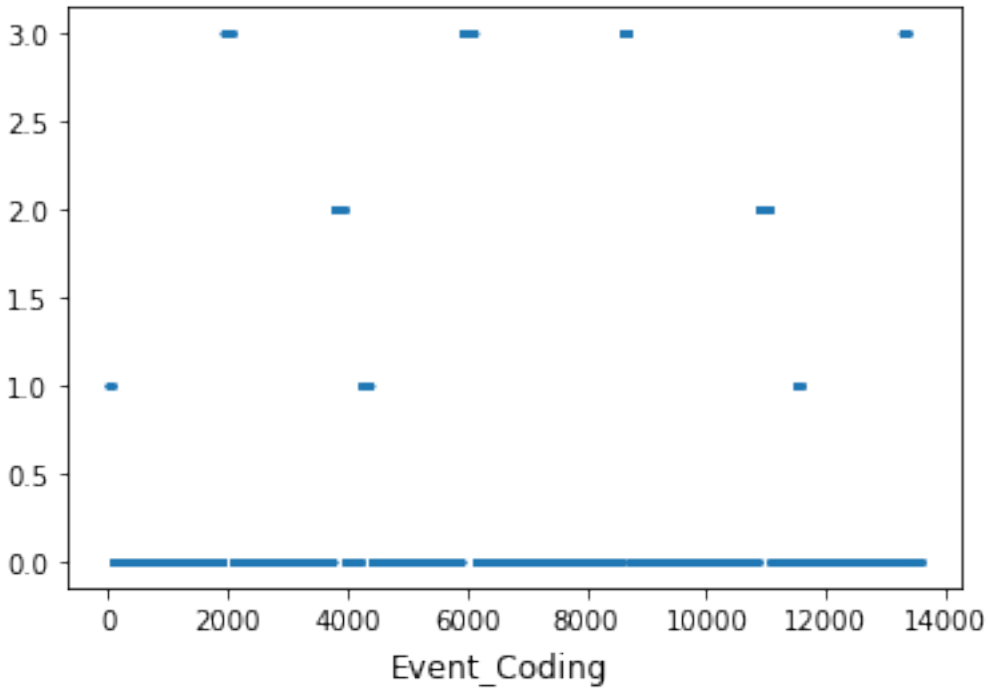
[8]:
```
# Fahrt 2
for i in range(1,11):
    figure=plt.scatter(trips_2.iloc[:,0],trips_2.iloc[:,i],s=1)
    plt.figtext(0.5, 0.01, trips_2.columns[i],wrap=True,␣
 ↪horizontalalignment='center', fontsize=12)
    plt.show()
```

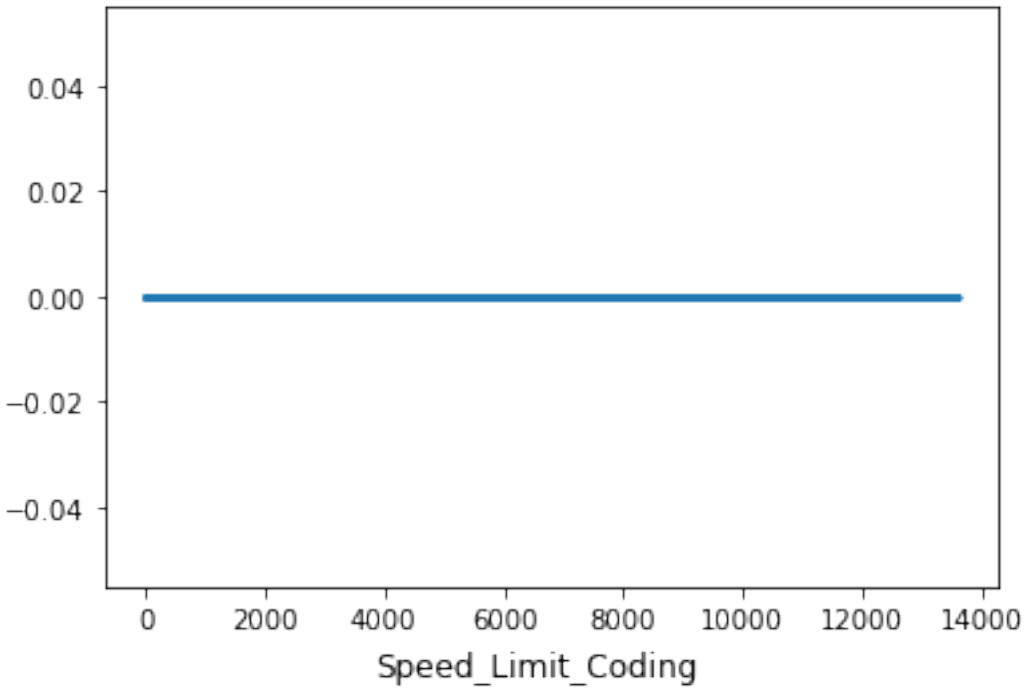


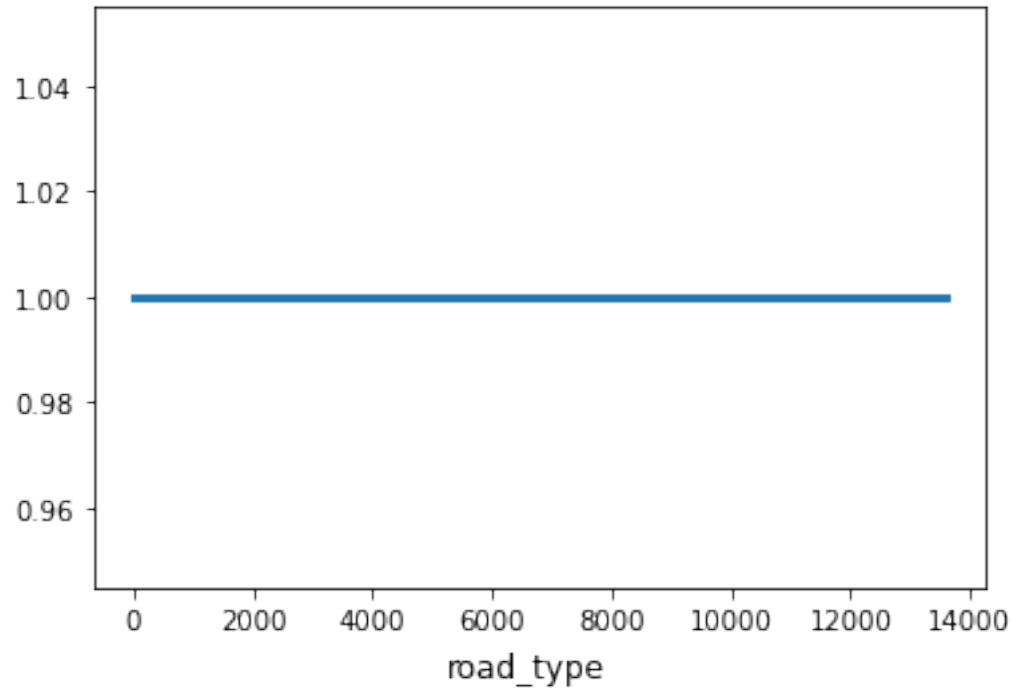

12500
10000
7500
5000
2500
0
−2500
−5000
0
2000
4000
6000
8000
10000
12000
14000
Acc_X

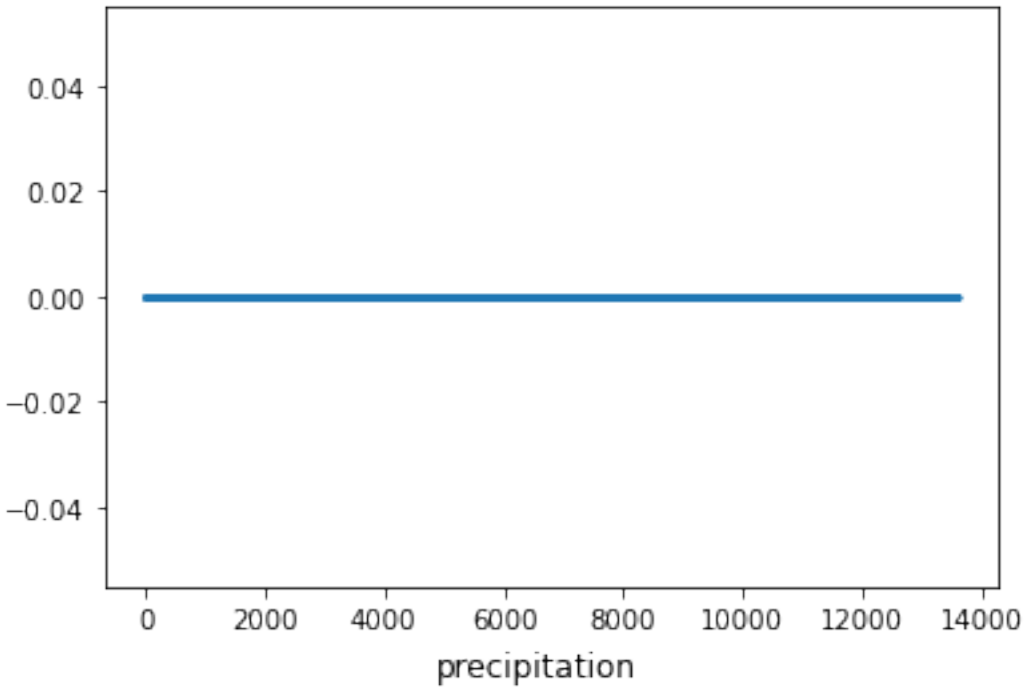

3000
2000
1000
0
−1000
−2000
−3000
0
2000
4000
6000
8000
10000
12000
14000
Acc_Y

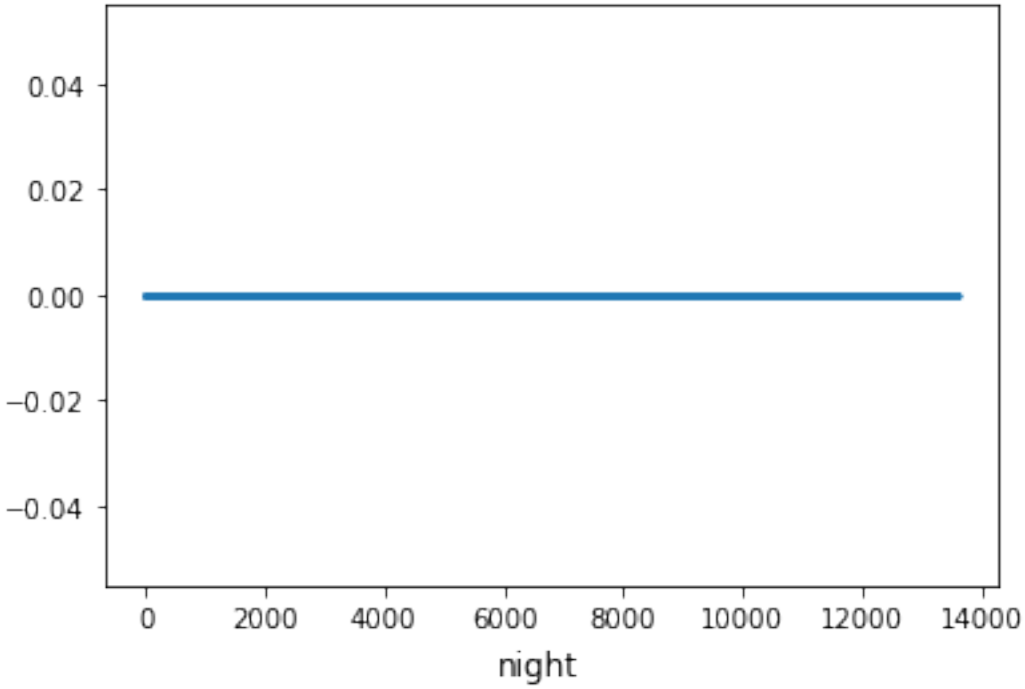

2000
1500
1000
500
0
0
2000
4000
6000
8000
10000
12000
14000
Acc_Z

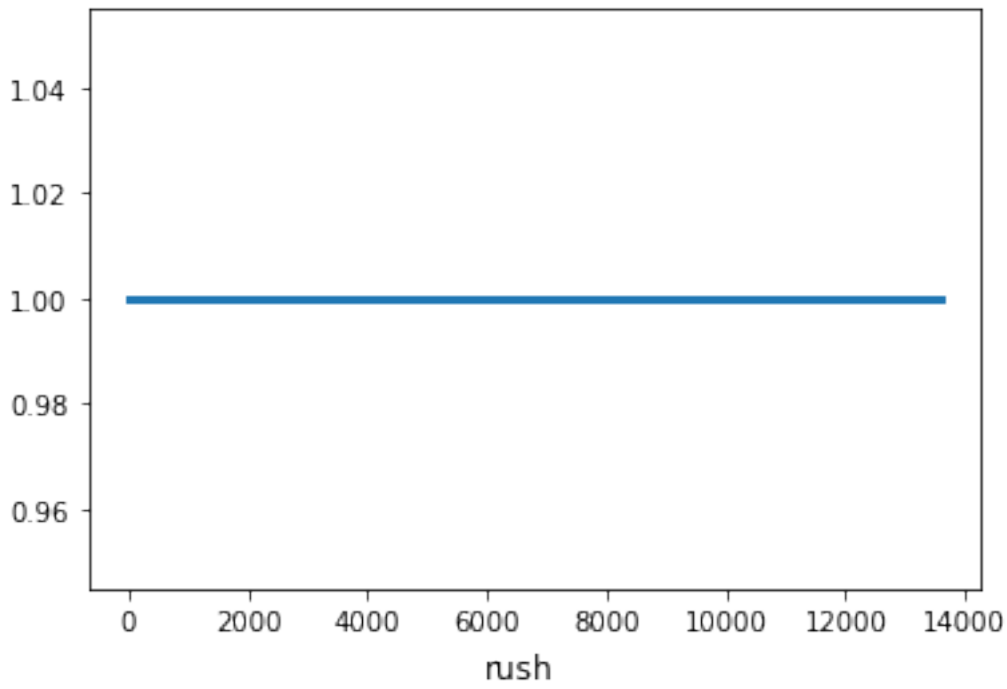

3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2000
4000
6000
8000
10000
12000
14000
Event_Coding


0.04
0.02
0.00
−0.02
−0.04
0
2000
4000
6000
8000
10000
12000
14000
Speed_Limit_Coding

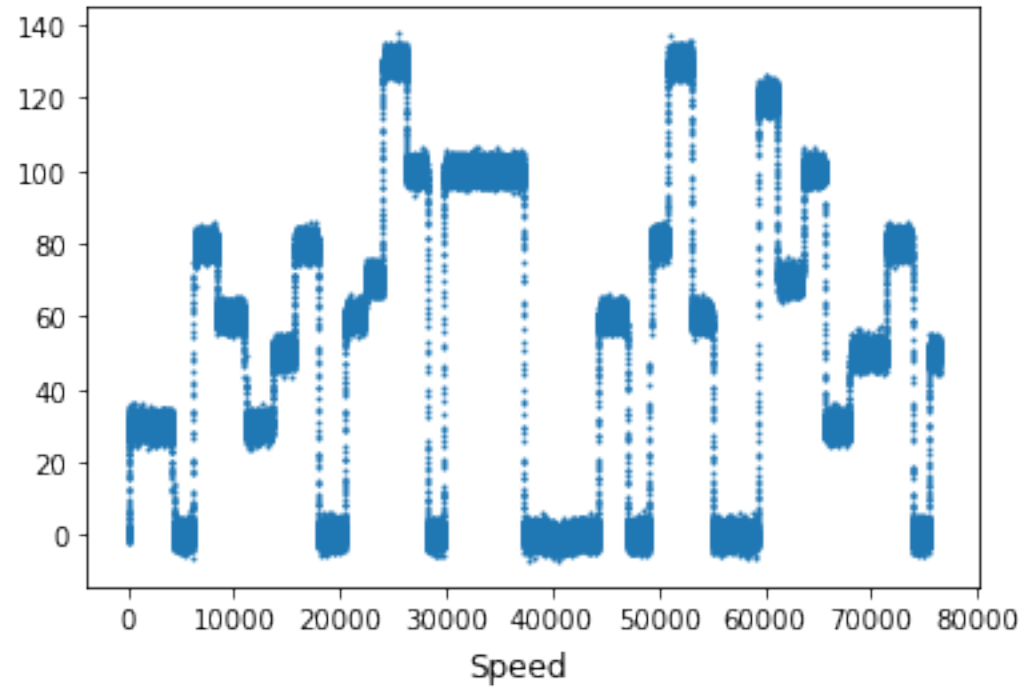

1.04
1.02
1.00
0.98
0.96
0
2000
4000
6000
8000
10000
12000
14000
road_type

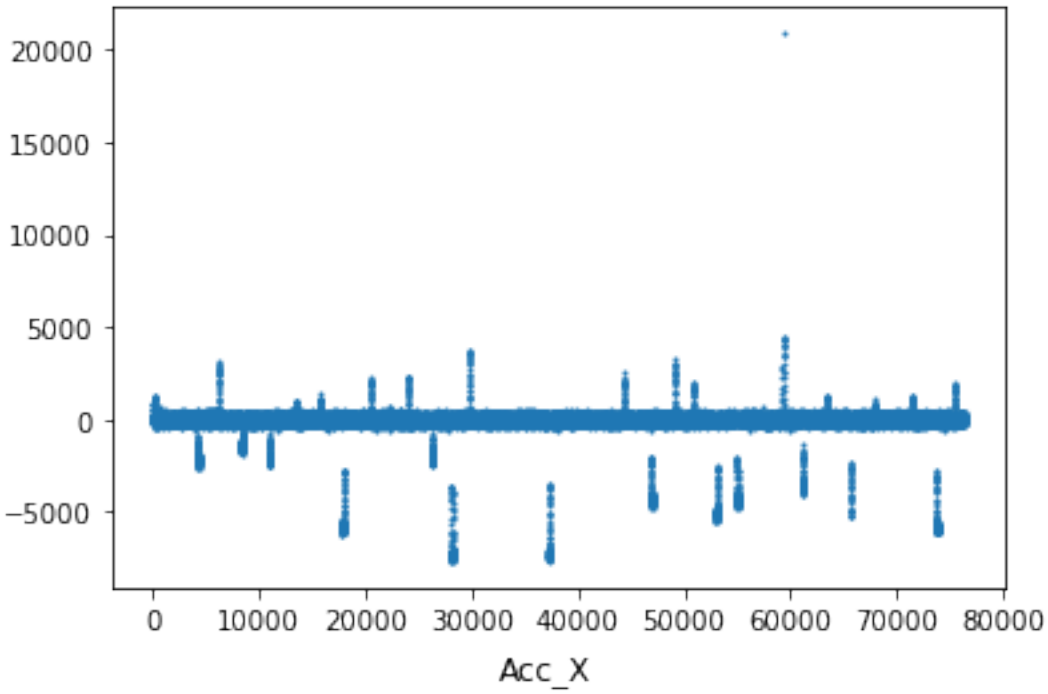

0.04
0.02
0.00
−0.02
−0.04
0
2000
4000
6000
8000
10000
12000
14000
precipitation

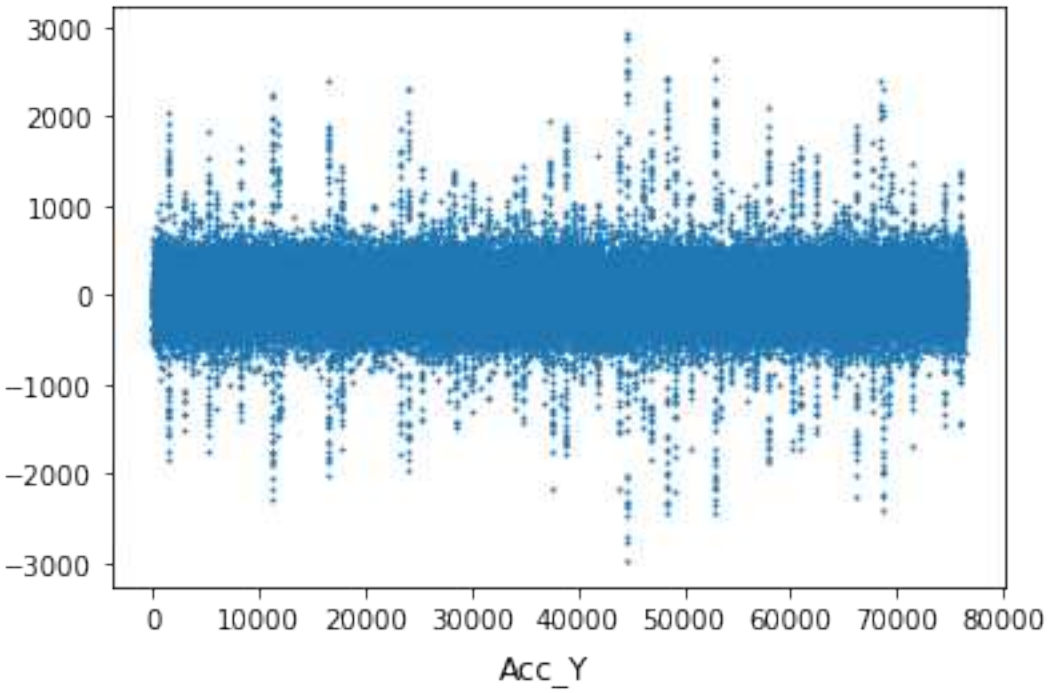

0.04
0.02
0.00
−0.02
−0.04
0
2000
4000
6000
8000
10000
12000
14000
night

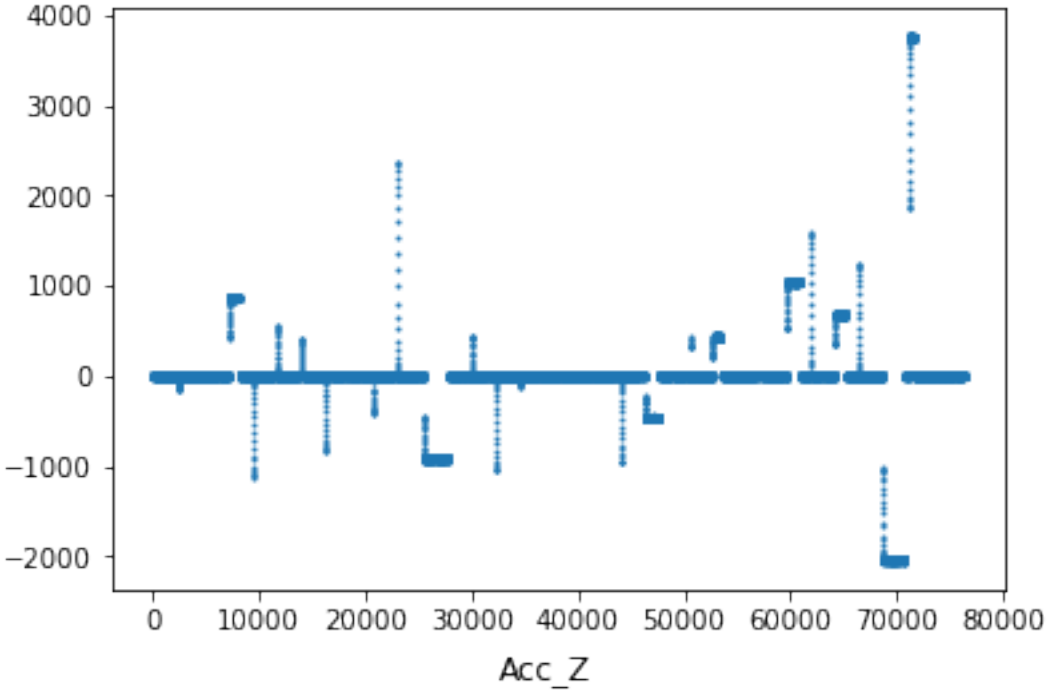


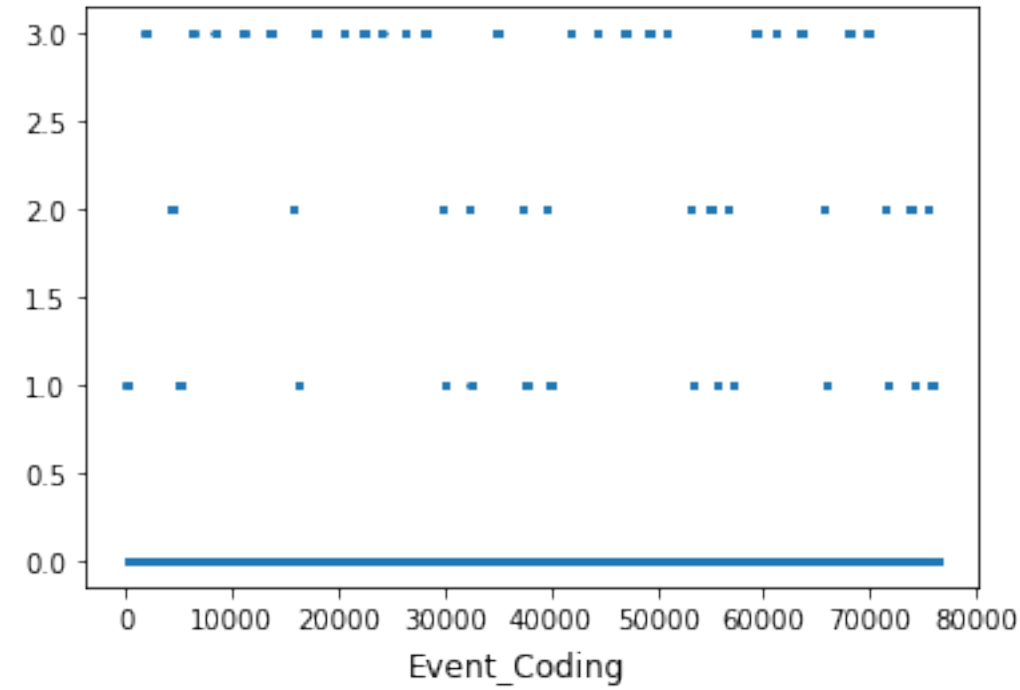
[9]:
```
# Fahrt 3
for i in range(1,11):
    figure=plt.scatter(trips_3.iloc[:,0],trips_3.iloc[:,i],s=1)
    plt.figtext(0.5, 0.01, trips_3.columns[i],wrap=True,␣
 ↪horizontalalignment='center', fontsize=12)
    plt.show()
```

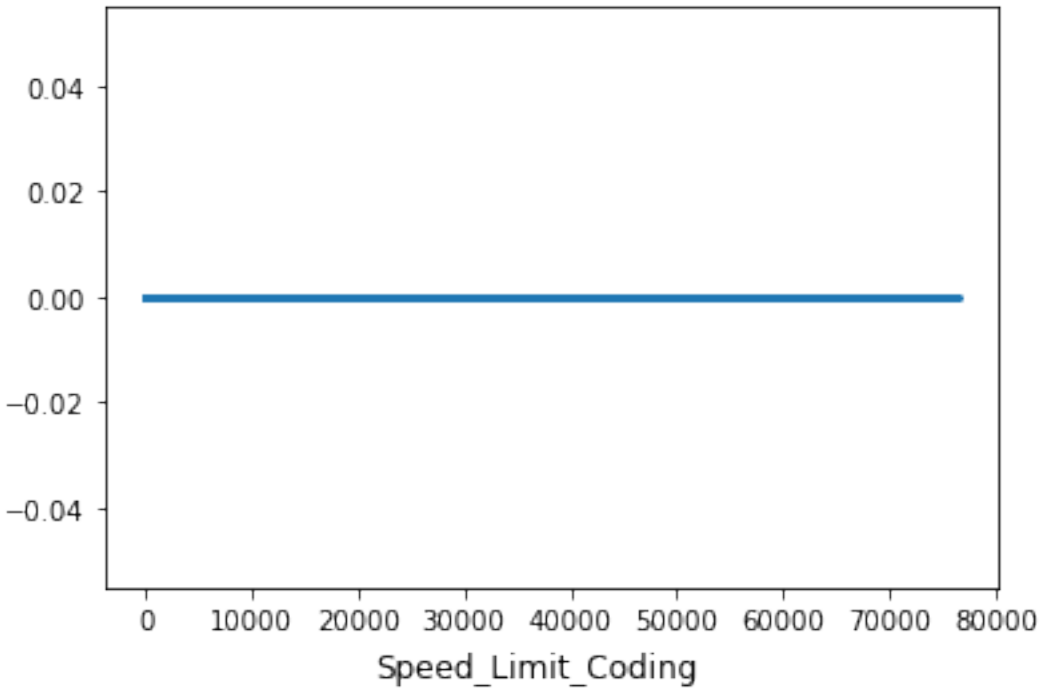

140
120
100
80
60
40
20
0
0
10000
20000
30000
40000
50000
60000
70000
80000
Speed

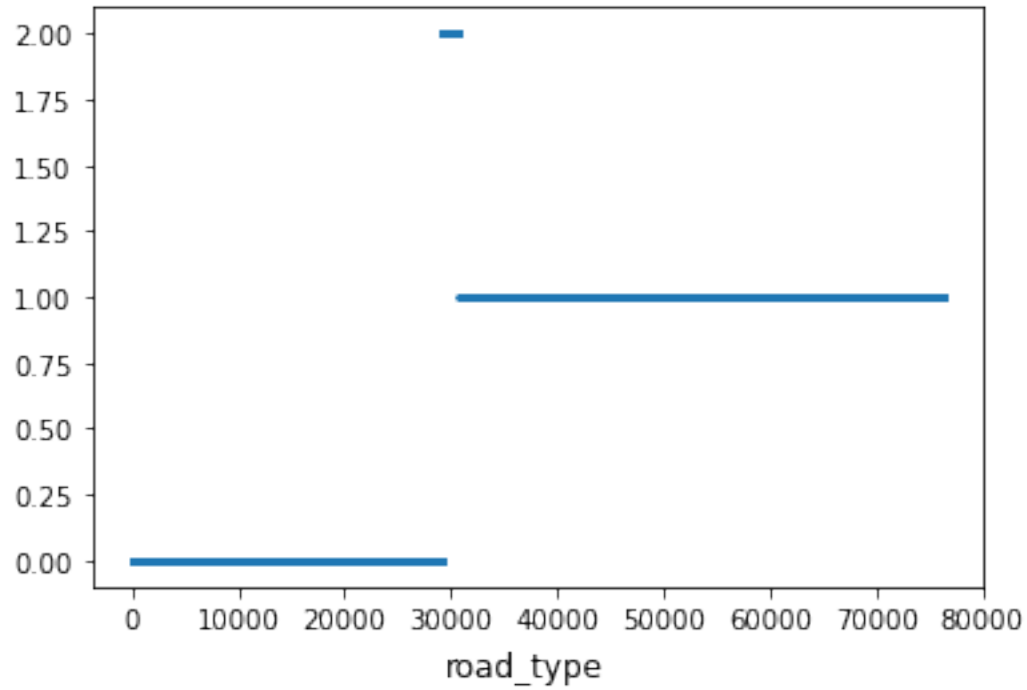

20000
15000
10000
5000
0
−5000
0
10000
20000
30000
40000
50000
60000
70000
80000
Acc_X

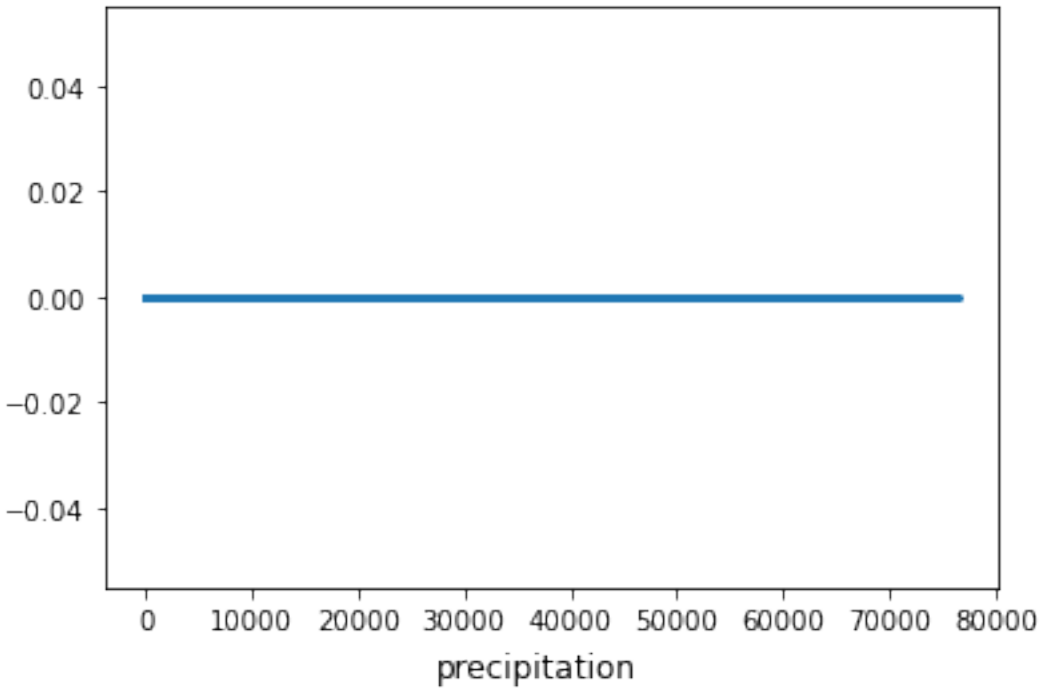

3000
2000
1000
0
−1000
−2000
−3000
0
10000
20000
30000
40000
50000
60000
70000
80000
Acc_Y

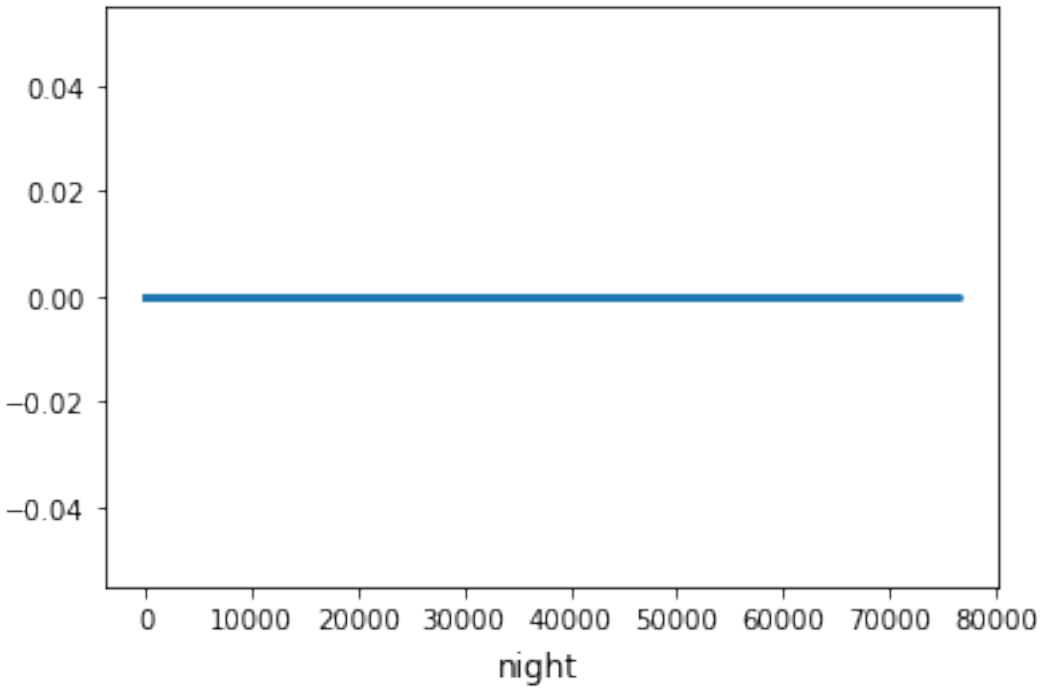

4000
3000
2000
1000
0
−1000
−2000
0
10000
20000
30000
40000
50000
60000
70000
80000
Acc_Z

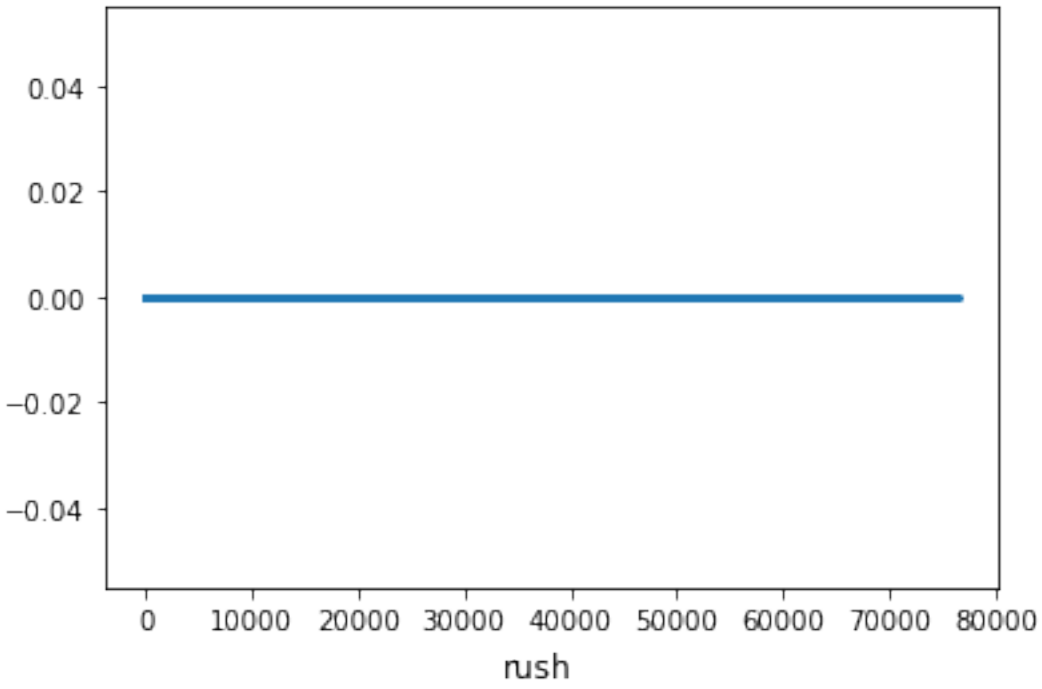

3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10000
20000
30000
40000
50000
60000
70000
80000
Event_Coding


0.04
0.02
0.00
−0.02
−0.04
0
10000
20000
30000
40000
50000
60000
70000
80000
Speed_Limit_Coding


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
10000
20000
30000
40000
50000
60000
70000
80000
road_type


0.04
0.02
0.00
−0.02
−0.04
0
10000
20000
30000
40000
50000
60000
70000
80000
precipitation

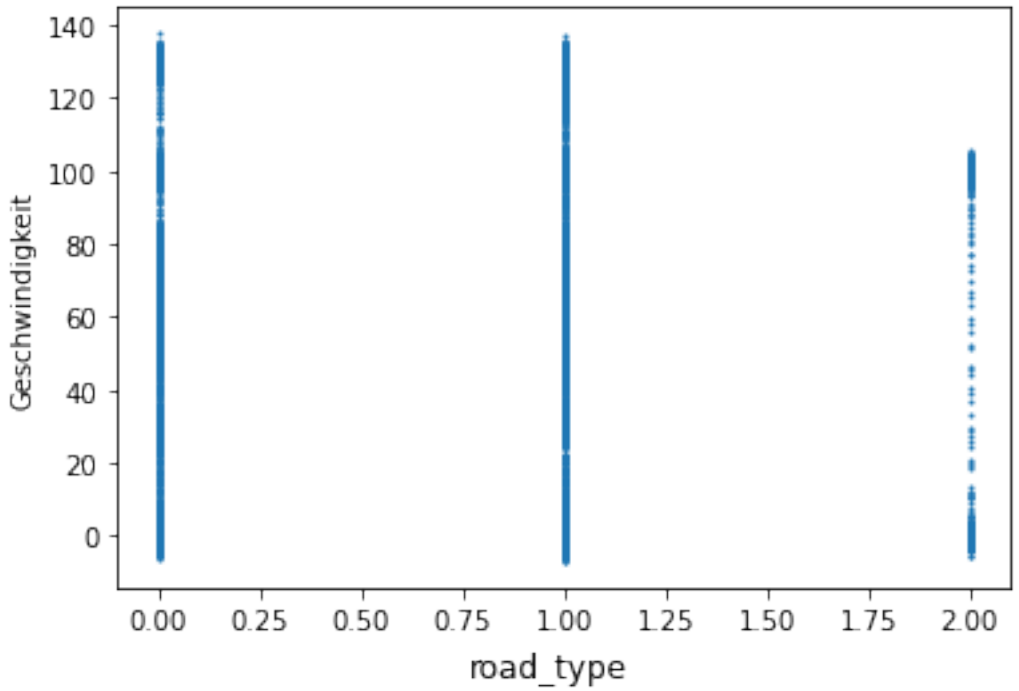




Wir können uns fragen, ob die Höchstgeschwindigkeit indikativ für den Straßentyp ist. Tatsächlich deutet nichts darauf hin, dass die Kodierung so wie im Schema angegeben plausibel wäre. Hier

muss man mit dem Dienstanbieter Rücksprache halten.

[10]:
```
figure=plt.scatter(trips_3.iloc[:,7],trips_3.iloc[:,1],s=1)
plt.figtext(0.5, 0.01, trips_2.columns[7],wrap=True,␣
 ↪horizontalalignment='center', fontsize=12)
plt.ylabel("Geschwindigkeit")
plt.show()
```



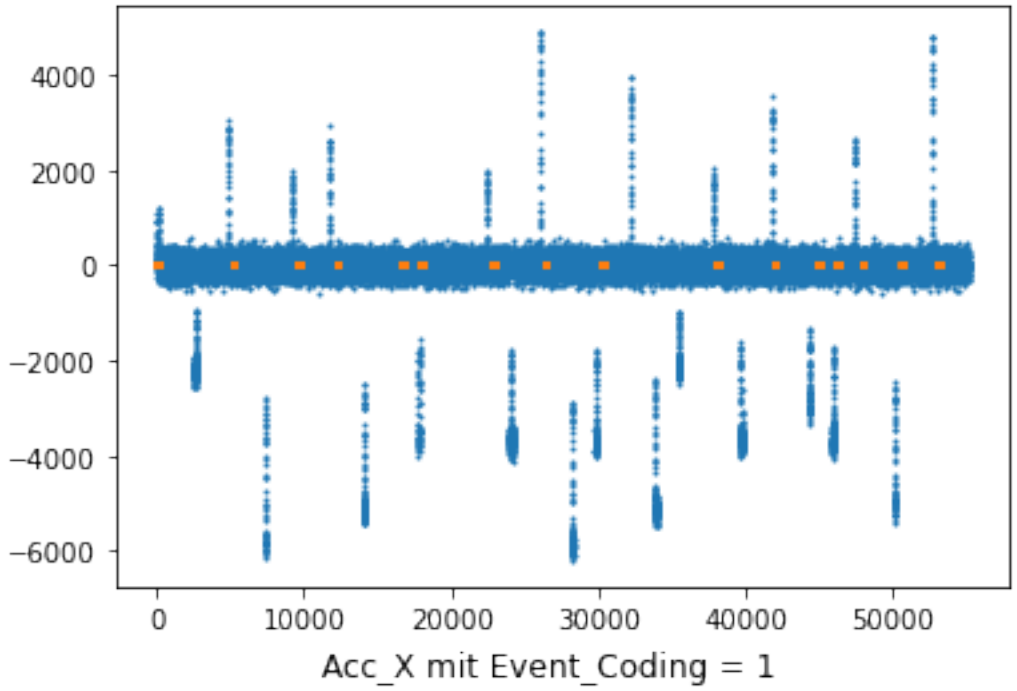
Wir beobachten Folgendes:

- Die erste Fahrt ist nicht beendet, sondern die Zeitreihe endet in einer hohen Geschwindigkeit. Auch die dritte Fahrt endet bei einer Geschwindigkeit um die 50 km/h und ist damit wohl nicht beendet.
- Das Feld `Speedlimit_Coding` hat nur eine Ausprägung und ist entweder falsch befüllt oder im Ausschnitt nicht relevant. Jedenfalls lässt sich sagen, dass das Merkmal wahrscheinlich keine Rolle spielt. Möglicherweise hat die im Schema verzeichnete Anreicherung aus Kartenmaterial nicht funktioniert.
- Das Feld `road_type` liefert nicht unbedingt konsistente Aussage: Die Bedeutungsebene der Art "Stadt, außerorts, Kraftstraße / Autobahn" kann anhand der Geschwindigkeit nicht verifiziert werden. Auch hier könnten Kartendaten falsch sein.
- Die Felder `night` und `rush` sind ebenfalls ohne Aussage, weil singulär ausgeprägt.
- Die Events bedürfen einer weiteren Prüfung. Hierzu werden die einzelnen Typen in die Beschleunigungsplots hineingetragen.

[11]:
```
trips_1[trips_1["Event_Coding"] == 1.0]['Event_Coding']
```

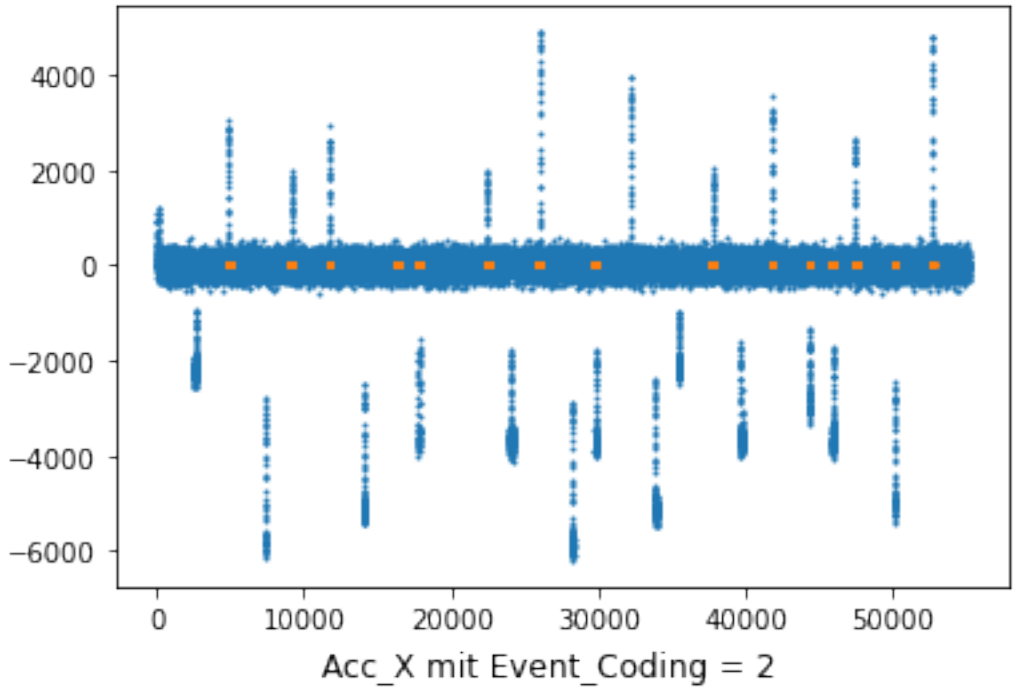
[11]:
```
89983     1.0
89984     1.0
89985     1.0
89986     1.0
89987     1.0
          …
143101    1.0
143102    1.0
143103    1.0
143104    1.0
143105    1.0
Name: Event_Coding, Length: 1873, dtype: float64
```

[12]:
```
figure=plt.scatter(trips_1.iloc[:,0],trips_1.iloc[:,2],s=1)
figure=plt.scatter(trips_1[trips_1["Event_Coding"] == 1.
 ↪0]['Time'],trips_1[trips_1["Event_Coding"] == 1.0]['Event_Coding'],s=4)
plt.figtext(0.5, 0.01, "Acc_X mit Event_Coding = 1",wrap=True,␣
 ↪horizontalalignment='center', fontsize=12)
plt.show()
```



[13]:
```
figure=plt.scatter(trips_1.iloc[:,0],trips_1.iloc[:,2],s=1)
figure=plt.scatter(trips_1[trips_1["Event_Coding"] == 2.
 ↪0]['Time'],trips_1[trips_1["Event_Coding"] == 2.0]['Event_Coding'],s=4)
```

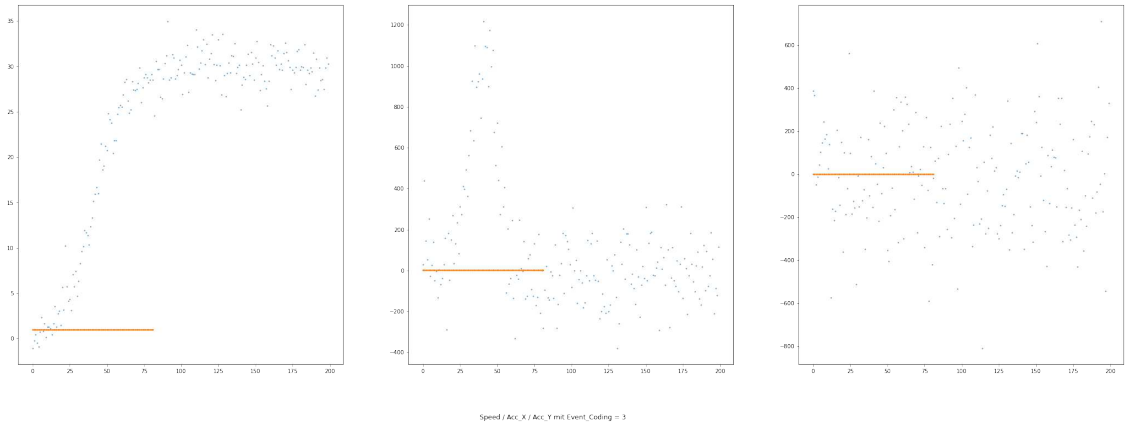
```
plt.figtext(0.5, 0.01, "Acc_X mit Event_Coding = 2",wrap=True,␣
 ↪horizontalalignment='center', fontsize=12)
plt.show()
```



Sucht man sich einen kleineren Ausschnitt aus den Zeitreihen, von dem bekannt ist, dass er ein Event_Coding enthält, so kann betrachtet werden, an welchen Stellen der Merkmalszeitreihen diese auftreten. Die Beobachtung des Codings 3 zeigt hier, dass es sowohl in `X`-, als auch in `Y`-Richtung ein Beschleunigungssignal gibt, sodass hier davon auszugehen ist, dass es sich hierbei um eine Kurvenfahrt oder ähnliches handeln muss.

[14]:
```
fig, axs = plt.subplots(1, 3)

fig.set_size_inches(36, 12)

a = 0 # Anfang des Slicings
e = 200 # Ende des Sclicings

axs[0].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,1],s=1)
axs[0].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 1.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 1.
 ↪0]['Event_Coding'],s=4)
axs[1].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,2],s=1)
```

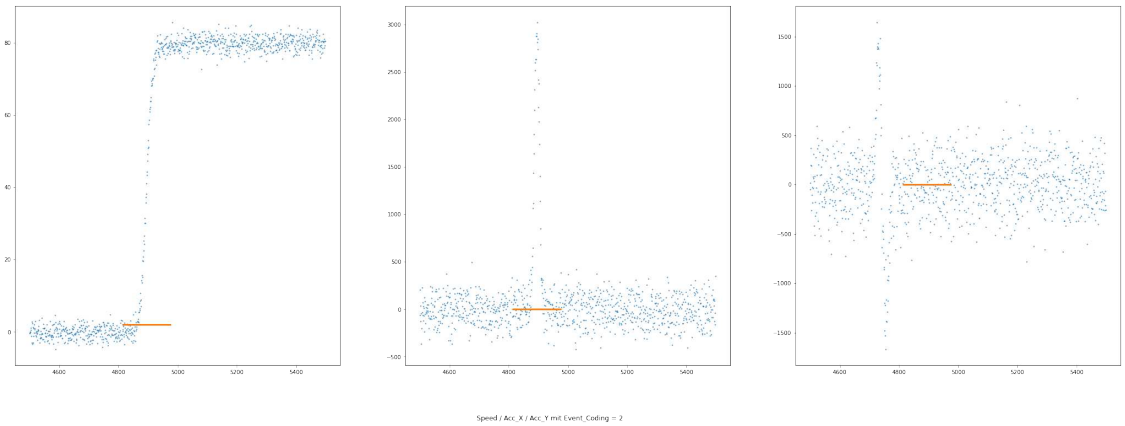
```
axs[1].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 1.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 1.
 ↪0]['Event_Coding'],s=4)
axs[2].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,3],s=1)
axs[2].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 1.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 1.
 ↪0]['Event_Coding'],s=4)
plt.figtext(0.5, 0.01, "Speed / Acc_X / Acc_Y mit Event_Coding = 3",wrap=True,␣
 ↪horizontalalignment='center', fontsize=12)

plt.show()
```



Es handelt sich bei dem Code 1 vermutlich um Beschleunigungsevents, da sie mit dem Ansteigen der Geschwindigkeit und positiven Werten von `Acc_X` zusammenfallen.

[15]:
```
fig, axs = plt.subplots(1, 3)

fig.set_size_inches(36, 12)

a = 4500 # Anfang des Slicings
e = 5500 # Ende des Sclicings

axs[0].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,1],s=1)
axs[0].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 2.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 2.
 ↪0]['Event_Coding'],s=4)
axs[1].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,2],s=1)
axs[1].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 2.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 2.
 ↪0]['Event_Coding'],s=4)
axs[2].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,3],s=1)
```

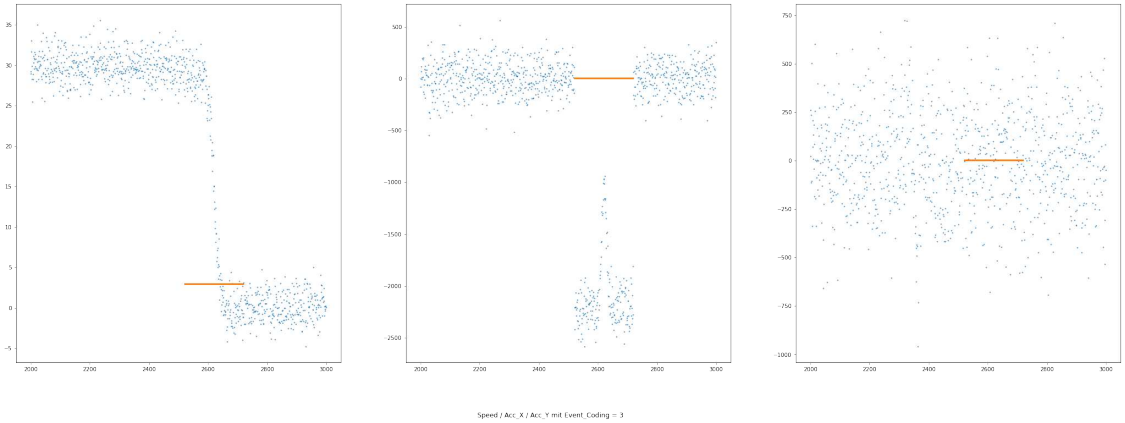
```
axs[2].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 2.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 2.
 ↪0]['Event_Coding'],s=4)
plt.figtext(0.5, 0.01, "Speed / Acc_X / Acc_Y mit Event_Coding = 2",wrap=True,␣
 ↪horizontalalignment='center', fontsize=12)

plt.show()
```



Der Visualisierung zufolge handelt es sich auch beim Code 2 um Beschleunigungsevents,da sich das gleiche Verhalten wie bei Code 1 zeigt. Möglicherweise ist ein Unterschied, dass zuvor eine Y-Beschleunigung stattfand, sodass auch die bedeutung “Beschleunigung nach Kurve” möglich ist. Dies ist anhand der vorliegenden Daten allerdings nur sehr schwierig abschließend zu beantworten.

[16]:
```
fig, axs = plt.subplots(1, 3)

fig.set_size_inches(36, 12)

a = 2000 # Anfang des Slicings
e = 3000 # Ende des Sclicings

axs[0].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,1],s=1)
axs[0].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 3.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 3.
 ↪0]['Event_Coding'],s=4)
axs[1].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,2],s=1)
axs[1].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 3.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 3.
 ↪0]['Event_Coding'],s=4)
axs[2].scatter(trips_1.iloc[a:e,0],trips_1.iloc[a:e,3],s=1)
axs[2].scatter(trips_1.iloc[a:e][trips_1["Event_Coding"] == 3.
 ↪0]['Time'],trips_1.iloc[a:e][trips_1["Event_Coding"] == 3.
 ↪0]['Event_Coding'],s=4)
```

```
plt.figtext(0.5, 0.01, "Speed / Acc_X / Acc_Y mit Event_Coding = 3",wrap=True,␣
↪horizontalalignment='center', fontsize=12)

plt.show()
```



Beim Code 3 handelt es sich ziemlich deutlich um ein Brems-Event, da sowohl Geschwindigkeit als auch X-Beschleunigung eine negative Tendenz aufweisen.

1.2.2 Aufgabe B3 (10 Punkte, Lernziele 3.3.3, 3.4.5, 5.1.7, 6.1.1)

Die Datei `telematik_agg.csv` enthält aggregierte Datensätze, die aus den Rohdaten abgeleitet wurden.

Die Zeitreihen sollen geeignet aggregiert werden, sodass die Rohdaten aus der Datei `rohdaten.csv` in das Zielformat der Spalten 1-16 von `telematik_agg.csv` überführt werden. Die Bedeutung der Spalten kann hierbei aus der Bezeichnung und den Rohdaten geeignet abgeleitet werden.

Ergeben sich unter Transformation die für unter den IDpol-Ausprägungen 3491, 13912, 20917 hinterlegten Werte der Spalten `max_accel_x min_accel_x max_accel_y min_accel_y max_speed_mw max_speed_urban max_speed_rural ratio_mw ratio_urban ratio_rural ratio_night ratio_precip ratio_rush four_accel_1 four_accel_2 four_accel_3 four_accel_4`, d.h. stimmt bspw. `np.min(trips[i][Acc_x])` für die einzelnen IDpol-Unterteilungen mit dem Eintrag in `telematik_agg` zu dieser ID überein? Wie sind etwaige Abweichungen zu erklären?

Hinweis zu den Spalten `four_accel_1`, `four_accel_2`, `four_accel_3` sowie `four_accel_4`: diese entstehen durch Fouriertransformationen, indem über 1,5 Sekunden-Fenster die ersten Koeffizienten des Aufrufs `rfft = np.abs(np.fft.rfft(trip[window]['Acc_X']))` bestimmt werden und anschließend über alle Fenster gemittelt wird.

(Hinweis: Das exakte Matching der aggregierten Werte ist nur insofern bewertungsrelevant, wie es aus der Dokumentation auch eindeutig bestimmt werden kann.)

Antwort Wir beginnen mit der Berechnung der Aggregate:

[17]:
```
rohdaten_df.head()
```

[17]:

	Time	Speed	Acc_X	Acc_Y	Acc_Z	Event_Coding	Speed_Limit_Coding	road_type	precipitation	night	rush	IDpol
0	0	2.026	108.481	-361.961	-4.640	1.0	0.0	1	0	0	1	13912
1	1	0.443	-71.785	-117.955	-7.371	1.0	0.0	1	0	0	1	13912
2	2	1.018	-89.126	301.670	7.013	1.0	0.0	1	0	0	1	13912
3	3	-1.166	-63.524	24.188	-4.548	1.0	0.0	1	0	0	1	13912
4	4	2.078	-19.656	-164.822	4.069	1.0	0.0	1	0	0	1	13912

[18]:
```
# Laden des Datensatzes:

telematik_df = pd.read_csv("./telematik_agg.csv",low_memory=False)
telematik_df = telematik_df.drop(["Unnamed: 0"],axis=1)
telematik_df.columns
```

[18]:
```
Index(['IDpol', 'max_accel_x', 'min_accel_x', 'max_accel_y', 'min_accel_y',
       'max_speed_mw', 'max_speed_urban', 'max_speed_rural', 'ratio_mw',
       'ratio_urban', 'ratio_rural', 'ratio_night', 'ratio_precip',
       'ratio_rush', 'four_accel_1', 'four_accel_2', 'four_accel_3',
       'four_accel_4'],
      dtype='object')
```

[19]:
```
telematik_df.head(5)
```

[19]:

	IDpol	max_accel_x	min_accel_x	max_accel_y	min_accel_y	max_speed_mw	max_speed_urban	max_speed_rural	ratio_mw	ratio_urban	ratio_rural	ratio_night	ratio_precip	ratio_rush	four_accel_1	four_accel_2
0	19260.0	0.12450	-2.161	2.055	-1.894	142.134	123.261	58.095	0.128	0.326	0.546	0.151116	0.162	-1725.867	-1571.672	105.482
1	1303.0	0.12139	-2.508	3.743	-1.522	260.000	176.083	70.877	0.081	0.263	0.656					
2	26535.0	0.09012	-2.173	2.780	-1.359	157.469	126.513	59.552	0.141	0.231	0.629					
3	4263.0	0.10508	-2.114	2.540	-1.775	146.727	113.567	60.670	0.131	0.292	0.577					
4	988.0	0.06733	-2.216	3.838	-2.024	250.000	165.411	73.215	0.176	0.246	0.578					

```
1     3.855576         0.116   22285.253      9028.015       537.691
2    -0.159239         0.147      79.762      2025.573         0.000
3    -3.962291         0.196   -3448.507      2936.377       113.498
4     1.282606         0.171   -1324.465      -452.557         3.748

   four_accel_3  four_accel_4
0       138.104      7571.877
1       942.168     10583.513
2       743.158     15526.304
3       644.315       556.608
4       912.984      2499.431
```

`IDpol` bekommen wir aus der entsprechenden Zeile, dies muss nicht aggregiert werden, sondern wie bereits geschehen, gruppiert werden. Die Weiteren Merkmale stellen Mittelungen, Maximumsbildungen etc. dar.

Trip 1

[20]:
```
minax = trips_1['Acc_X'].min()/1000
maxax = trips_1['Acc_X'].max()/1000
minay = trips_1['Acc_Y'].min()/1000
maxay= trips_1['Acc_Y'].max()/1000
maxmw = np.max(trips_1[trips_1.road_type == 2]['Speed'])
maxur = np.max(trips_1[trips_1.road_type == 0]['Speed'])
maxru = np.max(trips_1[trips_1.road_type == 1]['Speed'])
ratmw = trips_1[trips_1.road_type == 2]['Time'].count() / trips_1.Time.count()
ratur = trips_1[trips_1.road_type == 2]['Time'].count() / trips_1.Time.count()
ratru = trips_1[trips_1.road_type == 2]['Time'].count() / trips_1.Time.count()
ratni = trips_1[trips_1.night == 1]['Time'].count() / trips_1.Time.count()
ratpr = trips_1[trips_1.precipitation == 1]['Time'].count() / trips_1.Time.
 ↪count()
rrush = trips_1[trips_1.rush == 1]['Time'].count() / trips_1.Time.count()

### Berechnung Fourier-Komponenten

trip = trips_1
bin_length = 150
index_list = range(0,len(trip),150)
windowed_df_X = pd.
 ↪DataFrame(columns=['four_accel_1','four_accel_2','four_accel_3','four_accel_4'])
counter = 0
for i in index_list:
    counter+=1
    rfft = np.abs(np.fft.rfft(trip[i:i+bin_length]['Acc_X']))
    fft_box = np.zeros((4,1),float)
    fft_box[0] = rfft[0]
    fft_box[1] = rfft[1]
    fft_box[2] = rfft[2]
```

```
    fft_box[3] = rfft[3]
    windowed_df_X = windowed_df_X.append(pd.DataFrame([fft_box.
 ↪ravel()],columns=['four_accel_1','four_accel_2','four_accel_3','four_accel_4']))
trip_four = windowed_df_X.mean()

### Ende Fourier-berechnung

trip_1_stats = pd.DataFrame([[13912.
 ↪0,maxax,minax,maxay,minay,maxmw,maxur,maxru,ratmw,ratur,ratru,ratni,ratpr,rrush,trip_four[0
 ↪= telematik_df.columns)
```

[21]:
```
trip_1_stats
```

[21]:
```
     IDpol  max_accel_x  min_accel_x  max_accel_y  min_accel_y  max_speed_mw  \
0  13912.0     4.893402    -6.198431     2.638332    -2.893057       135.806

   max_speed_urban  max_speed_rural  ratio_mw  ratio_urban  ratio_rural  \
0          125.565          136.448  0.042618     0.042618     0.042618

   ratio_night  ratio_precip  ratio_rush  four_accel_1  four_accel_2  \
0          0.0      0.045574         0.0  19861.823269  10939.896811

   four_accel_3  four_accel_4
0   7153.213344   5512.549307
```

[22]:
```
telematik_df[telematik_df.index == 13912]
```

[22]:
```
         IDpol  max_accel_x  min_accel_x  max_accel_y  min_accel_y  \
13912  13912.0       4.8934      -6.1984       2.6383      -2.8931

       max_speed_mw  max_speed_urban  max_speed_rural  ratio_mw  ratio_urban  \
13912       135.806          125.565          136.448    0.0426       0.0426

       ratio_rural  ratio_night  ratio_precip  ratio_rush  four_accel_1  \
13912       0.0426          0.0        0.0456         0.0    19861.8233

       four_accel_2  four_accel_3  four_accel_4
13912    10939.8968     7153.2133     5512.5493
```

Trip 2:

[23]:
```
trip = trips_2

minax = trip['Acc_X'].min()/1000
maxax = trip['Acc_X'].max()/1000
minay = trip['Acc_Y'].min()/1000
maxay= trip['Acc_Y'].max()/1000
```

```
maxmw = np.max(trip[trip.road_type == 2]['Speed'])
maxur = np.max(trip[trip.road_type == 0]['Speed'])
maxru = np.max(trip[trip.road_type == 1]['Speed'])
ratmw = trip[trip.road_type == 2]['Time'].count() / trip.Time.count()
ratur = trip[trip.road_type == 2]['Time'].count() / trip.Time.count()
ratru = trip[trip.road_type == 2]['Time'].count() / trip.Time.count()
ratni = trip[trip.night == 1]['Time'].count() / trip.Time.count()
ratpr = trip[trip.precipitation == 1]['Time'].count() / trip.Time.count()
rrush = trip[trip.rush == 1]['Time'].count() / trip.Time.count()

### Berechnung Fourier-Komponenten

bin_length = 150
index_list = range(0,len(trip),150)
windowed_df_X = pd.
 ↪DataFrame(columns=['four_accel_1','four_accel_2','four_accel_3','four_accel_4'])
counter = 0
for i in index_list:
    counter+=1
    rfft = np.abs(np.fft.rfft(trip[i:i+bin_length]['Acc_X']))
    fft_box = np.zeros((4,1),float)
    fft_box[0] = rfft[0]
    fft_box[1] = rfft[1]
    fft_box[2] = rfft[2]
    fft_box[3] = rfft[3]
    windowed_df_X = windowed_df_X.append(pd.DataFrame([fft_box.
 ↪ravel()],columns=['four_accel_1','four_accel_2','four_accel_3','four_accel_4']))
trip_four = windowed_df_X.mean()

### Ende Fourier-berechnung

trip_2_stats = pd.DataFrame([[20917.
 ↪0,maxax,minax,maxay,minay,maxmw,maxur,maxru,ratmw,ratur,ratru,ratni,ratpr,rrush,trip_four[0
 ↪= telematik_df.columns)
```

[24]:
```
trip_2_stats
```

[24]:
```
      IDpol  max_accel_x  min_accel_x  max_accel_y  min_accel_y  max_speed_mw  \
0  20917.0       13.912    -4.804685     2.956455    -2.899303           NaN

   max_speed_urban  max_speed_rural  ratio_mw  ratio_urban  ratio_rural  \
0              NaN           105.91       0.0          0.0          0.0

   ratio_night  ratio_precip  ratio_rush  four_accel_1  four_accel_2  \
0          0.0           0.0         1.0  12409.307857    6924.52464

   four_accel_3  four_accel_4
```

```
0     4659.3696   4273.850289
```

[25]:
```
telematik_df[telematik_df.index == 20917]
```

[25]:
```
          IDpol  max_accel_x  min_accel_x  max_accel_y  min_accel_y  \
20917  20917.0       13.912      -4.8047       2.9565      -2.8993

       max_speed_mw  max_speed_urban  max_speed_rural  ratio_mw  ratio_urban  \
20917           NaN              NaN           105.91       0.0          0.0

       ratio_rural  ratio_night  ratio_precip  ratio_rush  four_accel_1  \
20917          0.0          0.0           0.0         1.0    12409.3079

       four_accel_2  four_accel_3  four_accel_4
20917     6924.5246     4659.3696     4273.8503
```

Trip 3

[26]:
```
trip = trips_3

minax = trip['Acc_X'].min()/1000
maxax = trip['Acc_X'].max()/1000
minay = trip['Acc_Y'].min()/1000
maxay= trip['Acc_Y'].max()/1000
maxmw = np.max(trip[trip.road_type == 2]['Speed'])
maxur = np.max(trip[trip.road_type == 0]['Speed'])
maxru = np.max(trip[trip.road_type == 1]['Speed'])
ratmw = trip[trip.road_type == 2]['Time'].count() / trip.Time.count()
ratur = trip[trip.road_type == 2]['Time'].count() / trip.Time.count()
ratru = trip[trip.road_type == 2]['Time'].count() / trip.Time.count()
ratni = trip[trip.night == 1]['Time'].count() / trip.Time.count()
ratpr = trip[trip.precipitation == 1]['Time'].count() / trip.Time.count()
rrush = trip[trip.rush == 1]['Time'].count() / trip.Time.count()

### Berechnung Fourier-Komponenten

bin_length = 150
index_list = range(0,len(trip),150)
windowed_df_X = pd.
 ↪DataFrame(columns=['four_accel_1','four_accel_2','four_accel_3','four_accel_4'])
counter = 0
for i in index_list:
    counter+=1
    rfft = np.abs(np.fft.rfft(trip[i:i+bin_length]['Acc_X']))
    fft_box = np.zeros((4,1),float)
    fft_box[0] = rfft[0]
    fft_box[1] = rfft[1]
```

```
    fft_box[2] = rfft[2]
    fft_box[3] = rfft[3]
    windowed_df_X = windowed_df_X.append(pd.DataFrame([fft_box.
 ↪ravel()],columns=['four_accel_1','four_accel_2','four_accel_3','four_accel_4']))
trip_four = windowed_df_X.mean()

### Ende Fourier-berechnung

trip_3_stats = pd.DataFrame([[20917.
 ↪0,maxax,minax,maxay,minay,maxmw,maxur,maxru,ratmw,ratur,ratru,ratni,ratpr,rrush,trip_four[0
 ↪= telematik_df.columns)
```

[27]:
```
trip_3_stats
```

[27]:
```
     IDpol  max_accel_x  min_accel_x  max_accel_y  min_accel_y  max_speed_mw  \
0  20917.0       20.917    -7.699774     2.933699    -2.972388       105.815

   max_speed_urban  max_speed_rural  ratio_mw  ratio_urban  ratio_rural  \
0          137.635          137.252  0.020671     0.020671     0.020671

   ratio_night  ratio_precip  ratio_rush  four_accel_1  four_accel_2  \
0          0.0           0.0         0.0  18802.637363   9465.701528

   four_accel_3  four_accel_4
0   6038.463578   4329.867135
```

[28]:
```
telematik_df[telematik_df.index == 3491]
```

[28]:
```
        IDpol  max_accel_x  min_accel_x  max_accel_y  min_accel_y  \
3491  20917.0       20.917      -7.6998       2.9337      -2.9724

      max_speed_mw  max_speed_urban  max_speed_rural  ratio_mw  ratio_urban  \
3491       105.815          137.635          137.252    0.0207       0.0207

      ratio_rural  ratio_night  ratio_precip  ratio_rush  four_accel_1  \
3491       0.0207          0.0           0.0         0.0    18802.6374

      four_accel_2  four_accel_3  four_accel_4
3491     9465.7015     6038.4636     4329.8671
```

Die berechneten Fourier-Transformationen passen bis auf geringe Abweichungen gut zu den vorliegenden Werten aus den angegebenen Zeilen.

Wie auch bei den Verhältnissen und den Aggregaten lässt sich beobachten, dass offenbar bei der zweiten Nachkommestelle gerundet wurde.

1.3 Teil B-II: Aggregierter Datensatz - Data Mining

In der Datei telematik_agg.csv befinden sich voraggregierte und angereicherte Sensordaten von Kfz-Policen. Das Ziel dieser Aufgabe besteht darin, geeignete Merkmale zu erzeugen, die in Form eines "Scores" in ein Tarifmodell (siehe Teil C) überführt werden können. Da in dieser Phase der Produktenwicklung das überwachte Lernen mittels verknüpften Schadendaten nicht zur Verfügung steht, muss unüberwacht gearbeitet werden. Gesucht ist eine Gruppierung der aggregierten Fahrdaten in fünf Cluster.

Die Dokumentation des Service-Providers lautet wie folgt:

Tabelle 2: Schema für telematik_agg.csv

Spaltenname

Datentyp [Einheit]

Beschreibung

max_accel_x

Float16 [G]

Maximale Beschleunigung in X-Richtung

min_accel_x

Float16 [G]

Minimale Beschleunigung in X-Richtung

max_accel_y

Float16 [G]

Maximale Beschleunigung in Y-Richtung

min_accel_y

Float16 [G]

Minimale Beschleunigung in Y-Richtung

max_speed_mw

Float16 [km/h]

Maximale Geschwindigkeit für Straßentyp Autobahn

max_speed_urban

Float16 [km/h]

Maximale Geschwindigkeit für Straßentyp innerorts

max_speed_rural

Float16 [km/h]

Maximale Geschwindigkeit für Straßentyp außerorts

ratio_mw

Float16 [-]

Anteil des Straßentyps Autobahn

ratio_urban

Float16 [-]

Anteil des Straßentyps innerorts

ratio_rural

Float16 [-]

Anteil des Straßentyps außerorts

ratio_night

Float16 [-]

Relativer Anteil Nachtfahrten

ratio_precip

Float16 [-]

Relativer Anteil Niederschlag

ratio_rush

Float16 [-]

Relativer Anteil Rush-Hour

four_accel_1

Float16 [Hz]

1. Komponente der DFT*, 1,5 s-Fenster
   four_accel_2
   Float16 [Hz]
   2. Komponente der DFT*, 1,5 s-Fenster
      four_accel_3
      Float16 [Hz]
      3. Komponente der DFT*, 1,5 s-Fenster
         four_accel_4
         Float16 [Hz]
         4. Komponente der DFT*, 1,5 s-Fenster

- Diskrete Fourier-Transformation (reellwertig mittels numpy.fft.rfft())
  Erzeugt über 1,5s-Fenster mit dem Aufruf von rfft =
- np.abs(np.fft.rfft(trip[window][‘Acc_X’])),danach gemittelt über alle Fenster.

1.3.1 Aufgabe B4 (10 Punkte, Lernziele 5.1.1 – 5.1.4)

Visualisieren Sie die Merkmale in einer Scatter-Matrix, die auf der Diagonalen eine Kerndichteschätzung aufweist. Spielen Sie aus dem Datensatz MTPLfreq_de.csv des Aufgabenteils C die Spalte DrivStyle an und bereinigen Sie etwaige Inkonsistenzen im Datensatz. Begründen Sie an Stellen, wo eine Datenbereinigung erforderlich ist, Ihre Auswahl. Visualisieren Sie für alle enthaltenen Merkmale die Beobachtungen in 3D-Scatterplots, die jeweils drei Merkmale gegeneinander abbilden (also [1,2,3],[4,5,6] usw.). Wieso ist das Vorgehen nur eingeschränkt zweckmäßig? Lassen Sich Rückschlüsse auf erste Muster in den Daten ziehen, welche für eine Clusterung relevant sein können?

[29]:
```
telematik_df
```

[29]:

	IDpol	max_accel_x	min_accel_x	max_accel_y	min_accel_y	\
0	19260.0	0.12450	-2.161	2.055	-1.894	
1	1303.0	0.12139	-2.508	3.743	-1.522	
2	26535.0	0.09012	-2.173	2.780	-1.359	
3	4263.0	0.10508	-2.114	2.540	-1.775	
4	988.0	0.06733	-2.216	3.838	-2.024	
…	…	…	…	…	…	
29210	29121.0	0.15333	-2.205	3.463	-1.580	
29211	236.0	0.13192	-2.581	3.965	-1.558	
29212	26143.0	0.10092	-2.352	3.940	-1.323	
29213	13695.0	0.08094	-2.398	3.926	-1.918	
29214	21353.0	0.14058	-2.139	3.402	-0.939	

	max_speed_mw	max_speed_urban	max_speed_rural	ratio_mw	ratio_urban	\
0	142.134	123.261	58.095	0.128	0.326	
1	260.000	176.083	70.877	0.081	0.263	
2	157.469	126.513	59.552	0.141	0.231	
3	146.727	113.567	60.670	0.131	0.292	
4	250.000	165.411	73.215	0.176	0.246	
…	…	…	…	…	…	
29210	175.175	134.486	66.659	0.173	0.243	
29211	259.581	169.929	76.348	0.070	0.233	
29212	240.000	149.296	76.555	0.111	0.271	
29213	250.000	160.034	71.075	0.192	0.233	
29214	222.605	153.817	70.839	0.183	0.207	

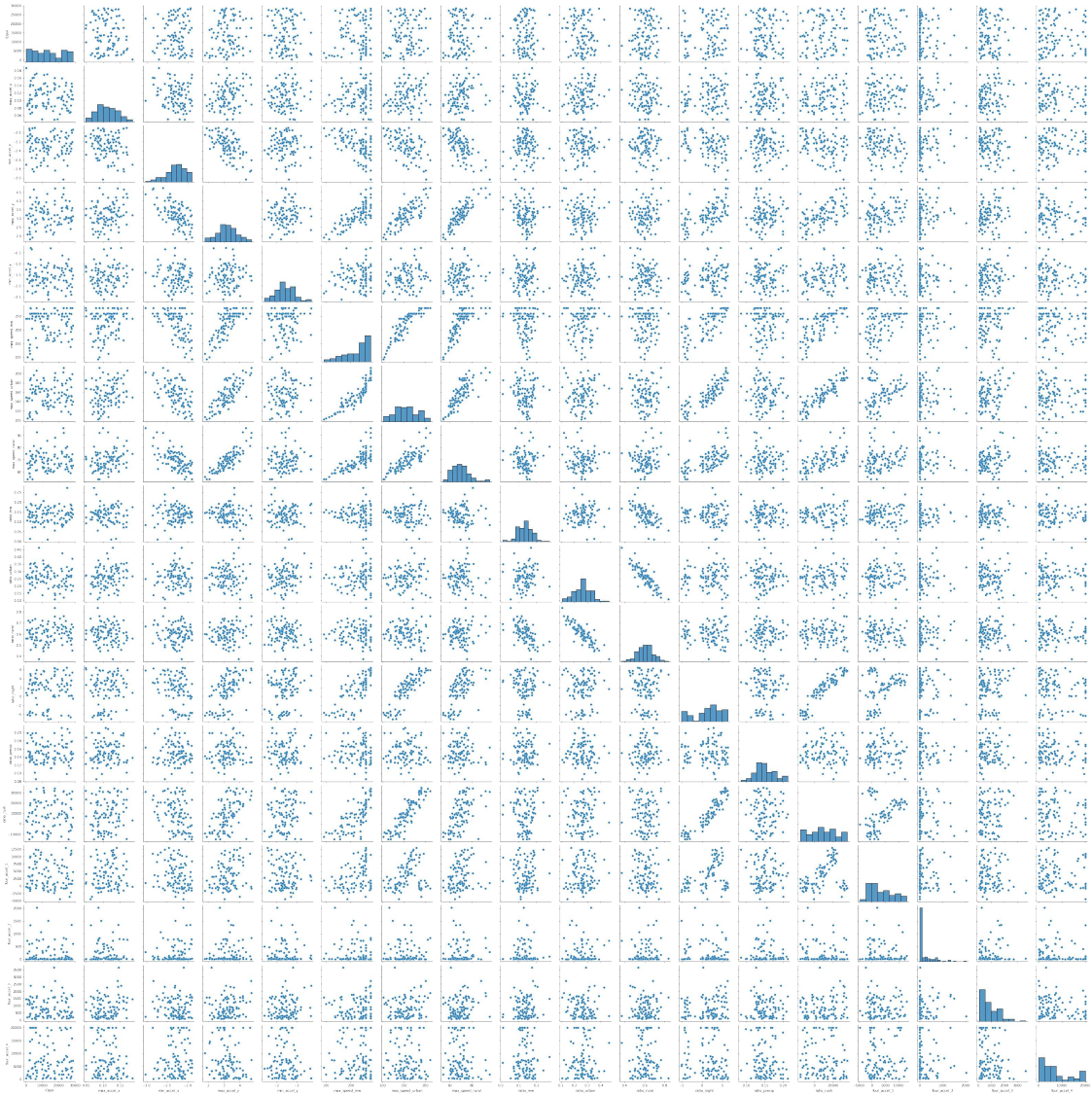
	ratio_rural	ratio_night	ratio_precip	ratio_rush	four_accel_1	\
0	0.546	0.151116	0.162	-1725.867	-1571.672	
1	0.656	3.855576	0.116	22285.253	9028.015	
2	0.629	-0.159239	0.147	79.762	2025.573	
3	0.577	-3.962291	0.196	-3448.507	2936.377	
4	0.578	1.282606	0.171	-1324.465	-452.557	
…	…	…	…	…	…	
29210	0.585	-0.586203	0.130	2283.903	703.309	
29211	0.698	3.607420	0.159	22158.770	7826.207	

```
29212        0.618     1.230867          0.131   -1869.034        -6.010
29213        0.575     2.068949          0.182   11948.122       172.997
29214        0.610     1.667215          0.117    9421.204      4441.900

       four_accel_2  four_accel_3  four_accel_4
0           105.482       138.104     7571.877
1           537.691       942.168    10583.513
2             0.000       743.158    15526.304
3           113.498       644.315      556.608
4             3.748       912.984     2499.431
…               …             …            …
29210         0.377       487.034     6240.011
29211        27.332       798.874    12995.517
29212        11.638      2464.727    20000.000
29213       435.255       381.740    20000.000
29214       547.306       772.529    11261.062

[29215 rows x 18 columns]
```

ACHTUNG: PLOT DAUERT SEHR LANGE!

[30]:
```
#sns.pairplot(telematik_df)
sns.pairplot(telematik_df.sample(100))
```

[30]:
```
<seaborn.axisgrid.PairGrid at 0x2cdd3a9f3d0>
```



Es gibt in den Merkmalen keine großen Auffälligkeiten. Eher im Gegenteil: Die Merkmale weisen weder eine große Schiefe, noch große Korrelationen untereinander auf. Es gibt bis hierher keinen Grund anzunehmen, dass sich für die spätere Analyse nutzbare Muster (zumindest visuell) identifizieren ließen.

Als nächstes soll die Spalte `DrivStyle` des in Aufgabenteil C verwendeten Datensatzes angespielt werden.

[31]:
```
MTPL_df = pd.read_csv(".\MTPLfreq-de.csv",delimiter=";")
zuordnung_df = MTPL_df[['IDpol','DrivStyle']]
zuordnung_df
```

[31]:

	IDpol	DrivStyle
0	1	2

```
1          2          1
2          3          4
3          4          1
4          5          1
…        …        …
29211  29212          2
29212  29213          4
29213  29214          4
29214  29215          2
29215  29216          2

[29216 rows x 2 columns]
```

[32]:
```
telematik_df = pd.
 ↪merge(telematik_df,zuordnung_df,right_on="IDpol",left_on="IDpol").
 ↪reindex(telematik_df.index)
telematik_df
```

[32]:
```
          IDpol  max_accel_x  min_accel_x  max_accel_y  min_accel_y  \
0      19260.0      0.12450       -2.161        2.055       -1.894
1       1303.0      0.12139       -2.508        3.743       -1.522
2      26535.0      0.09012       -2.173        2.780       -1.359
3       4263.0      0.10508       -2.114        2.540       -1.775
4        988.0      0.06733       -2.216        3.838       -2.024
…          …            …          …           …           …
29210  26143.0      0.10092       -2.352        3.940       -1.323
29211  13695.0      0.08094       -2.398        3.926       -1.918
29212  21353.0      0.14058       -2.139        3.402       -0.939
29213      NaN          NaN          NaN          NaN          NaN
29214      NaN          NaN          NaN          NaN          NaN

       max_speed_mw  max_speed_urban  max_speed_rural  ratio_mw  ratio_urban  \
0           142.134          123.261           58.095     0.128        0.326
1           260.000          176.083           70.877     0.081        0.263
2           157.469          126.513           59.552     0.141        0.231
3           146.727          113.567           60.670     0.131        0.292
4           250.000          165.411           73.215     0.176        0.246
…             …                …                …       …          …
29210       240.000          149.296           76.555     0.111        0.271
29211       250.000          160.034           71.075     0.192        0.233
29212       222.605          153.817           70.839     0.183        0.207
29213           NaN              NaN              NaN       NaN          NaN
29214           NaN              NaN              NaN       NaN          NaN

       ratio_rural  ratio_night  ratio_precip  ratio_rush  four_accel_1  \
0            0.546     0.151116         0.162   -1725.867     -1571.672
1            0.656     3.855576         0.116   22285.253      9028.015
```

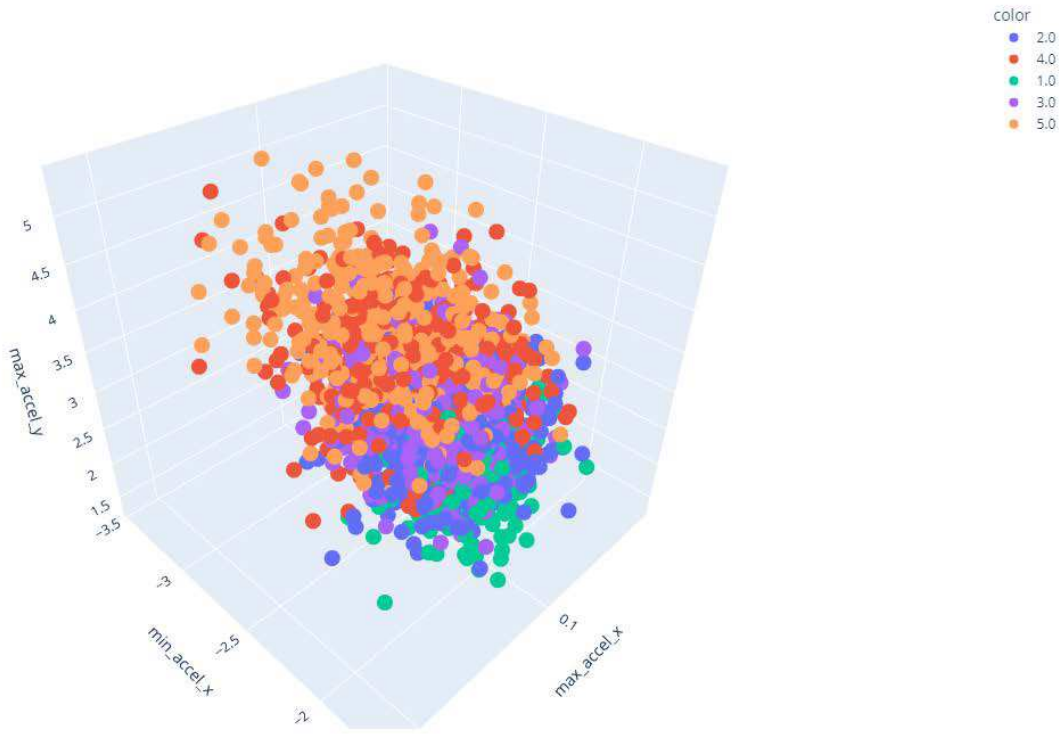
```
2            0.629    -0.159239          0.147      79.762      2025.573
3            0.577    -3.962291          0.196   -3448.507      2936.377
4            0.578     1.282606          0.171   -1324.465      -452.557
…              …           …              …          …             …
29210        0.618     1.230867          0.131   -1869.034       -6.010
29211        0.575     2.068949          0.182   11948.122       172.997
29212        0.610     1.667215          0.117    9421.204      4441.900
29213          NaN          NaN            NaN         NaN           NaN
29214          NaN          NaN            NaN         NaN           NaN

       four_accel_2  four_accel_3  four_accel_4  DrivStyle
0           105.482       138.104     7571.877        2.0
1           537.691       942.168    10583.513        4.0
2             0.000       743.158    15526.304        2.0
3           113.498       644.315      556.608        1.0
4             3.748       912.984     2499.431        3.0
…               …             …            …          …
29210        11.638      2464.727    20000.000        2.0
29211       435.255       381.740    20000.000        3.0
29212       547.306       772.529    11261.062        3.0
29213           NaN           NaN          NaN        NaN
29214           NaN           NaN          NaN        NaN

[29215 rows x 19 columns]
```

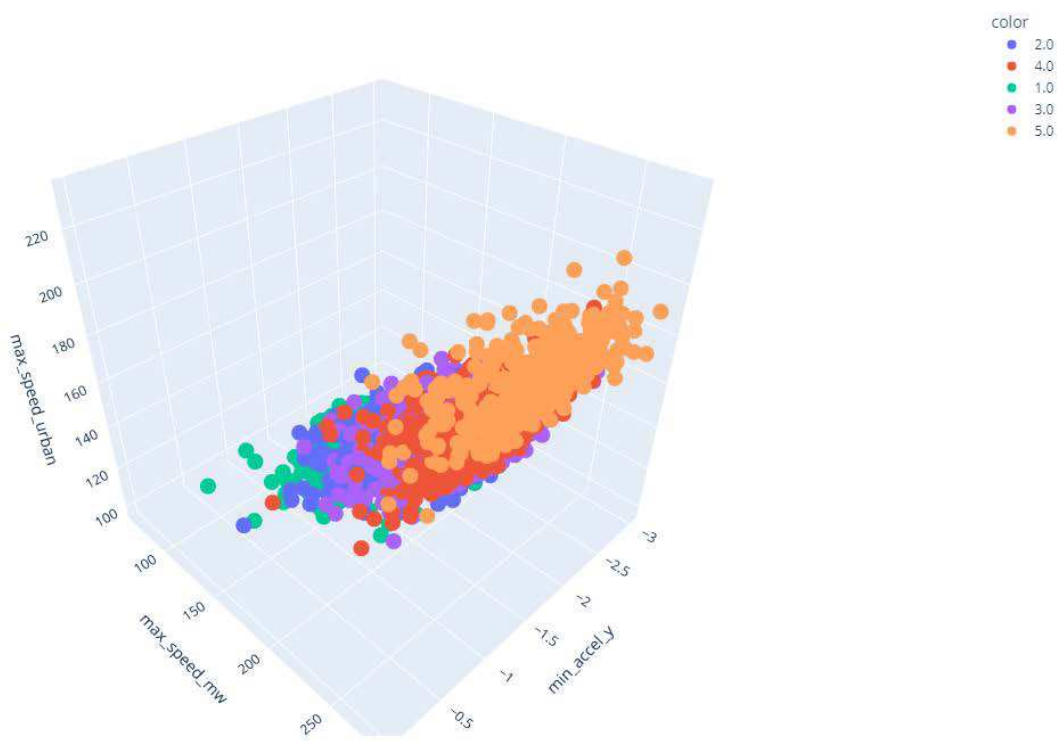
Wir stellen fest, dass `zuordnung_df` sortiert ist. Folglich können wir über Reindizierung die Driving Styles an den Dataframe `telematik_df` anhängen.

Im Folgenden werden nun 3D-Scatterplots für jeweils drei aufeinander folgende Merkmale erzeugt, dann noch für zufällig ausgewählte Kombinationen. Hierbei muss auf ein Subsampling zurückgegriffen werden, da der Aufwand sonst schnell prohibitiv werden kann.

[86]:
```
fig = px.scatter_3d(telematik_df[:2000], x=telematik_df.columns[1],␣
↪y=telematik_df.columns[2], z=telematik_df.columns[3],
              color=telematik_df['DrivStyle'][:2000].astype(str),width=1200,␣
↪height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

[87]:
```
fig = px.scatter_3d(telematik_df[:2000], x=telematik_df.columns[4],␣
↪y=telematik_df.columns[5], z=telematik_df.columns[6],
              color=telematik_df['DrivStyle'][:2000].astype(str),width=1200,␣
↪height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```


color
2.0
4.0
1.0
3.0
5.0
max_accel_y
5
4.5
4
3.5
3
2.5
2
1.5
min_accel_x
−3.5
−3
−2.5
−2
max_accel_x
0.1


color
2.0
4.0
1.0
3.0
5.0
max_speed_urban
220
200
180
160
140
120
100
max_speed_mw
100
150
200
250
min_accel_y
−0.5
−1
−1.5
−2
−2.5
−3

[88]:
```
fig = px.scatter_3d(telematik_df[:2000], x=telematik_df.columns[7],␣
↪y=telematik_df.columns[8], z=telematik_df.columns[9],
              color=telematik_df['DrivStyle'][:2000].astype(str),width=1200,␣
↪height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

[89]:
```
fig = px.scatter_3d(telematik_df[:2000], x=telematik_df.columns[10],␣
↪y=telematik_df.columns[11], z=telematik_df.columns[12],
              color=telematik_df['DrivStyle'][:2000].astype(str),width=1200,␣
↪height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

[90]:
```
fig = px.scatter_3d(telematik_df[:2000], x=telematik_df.columns[13],␣
↪y=telematik_df.columns[14], z=telematik_df.columns[15],
              color=telematik_df['DrivStyle'][:2000].astype(str),width=1200,␣
↪height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

[91]:
```
fig = px.scatter_3d(telematik_df[:2000], x=telematik_df.columns[1],␣
↪y=telematik_df.columns[2], z=telematik_df.columns[3],
              color=telematik_df['DrivStyle'][:2000].astype(str),width=1200,␣
↪height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

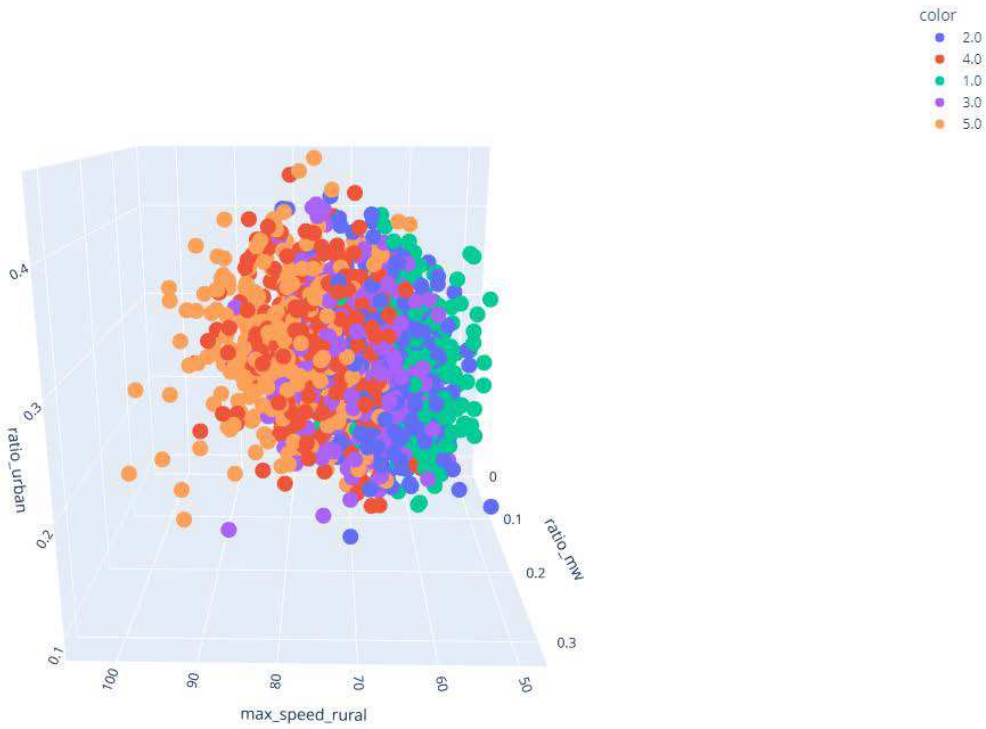
Es zeigt sich, dass 3D-Scatterplots zwar visuell beeindruckend und ästhetisch ansprechend sind, aber natürlich maximal nur drei Merkmale gegeneinander auftragen können. Muster, die mehr Dimensionen umfassen und/oder große Nicht-Linearitäten aufweisen, werden hierdurch nicht auffindbar. Insbesondere bei vielen merkmalen ist das Vorgehen, wenn man möglichst viele Kombinationen durchprüfen möchte, auch prohibitiv in Rechen- und Betrachtungsaufwand.

Hierbei ist zu erkennen, dass bezüglich der Ausprägungen von `drvstyle` durchaus Muster zu erkennen sind: So ist beispielsweise eine sphärische Auftrennung der Gruppen im Plot mit `four_accel_1` und `ratio_rush` erkennbar, die auch eine geeignete Clusterung reproduzieren müsste.

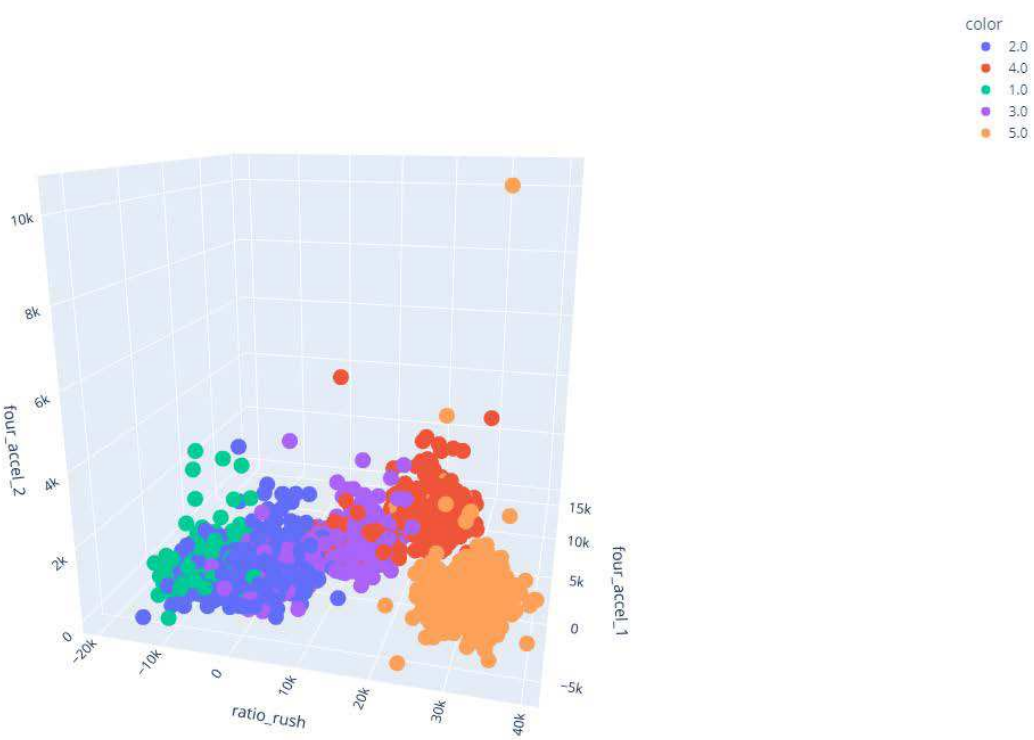
1.3.2 Aufgabe B5 (15 Punkte, 5.1.8, 5.1.9)

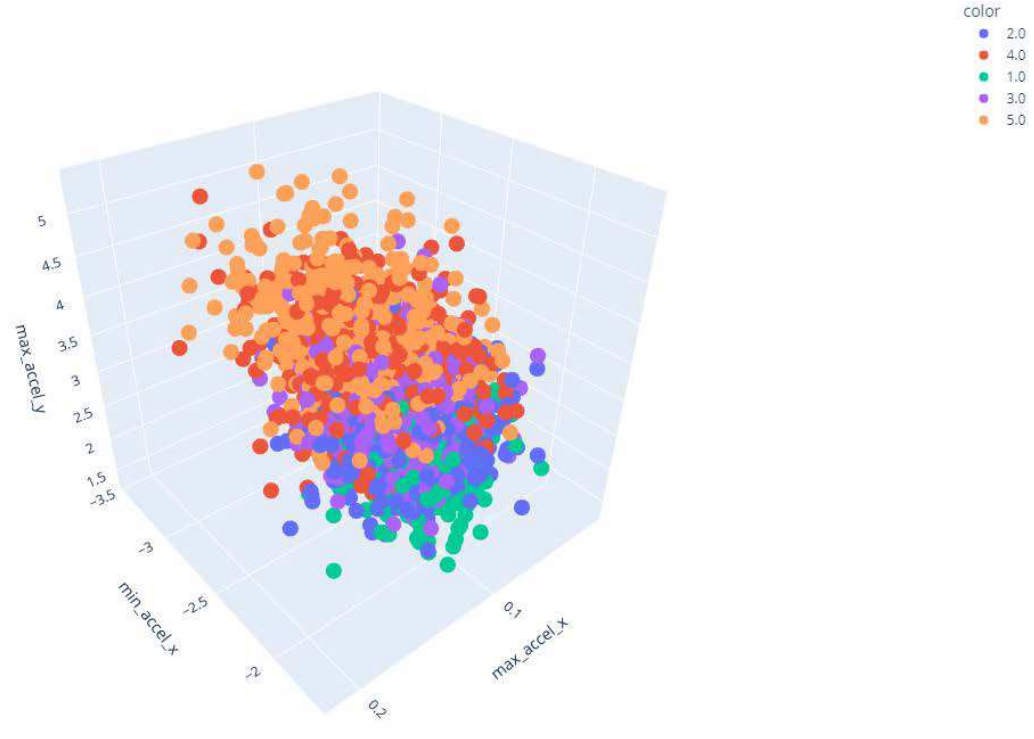
Führen Sie a) ein *agglomeratives* und b) ein *zentroidbasiertes* Clusterverfahren durch, um die Fahrer in fünf homogene Gruppen einzuteilen. Diskutieren Sie anhand geeigneter Kriterien, ob die vom Management gewählte Anzahl im Hinblick auf das unüberwachte Vorgehen zweckmäßig erscheint und begründen Sie Ihre Auswahl. Vergleichen Sie die Resultate mit einem Vorgehen, bei dem die Merkmale zuvor zunächst standardisiert wurden: Welche Unterschiede beobachten Sie und worauf sind diese zurückzuführen?

Zunächst werden `IDpol` und `drvstyle` aus dem Dataframe entfernt, da sie keine Information bzgl.


color
2.0
4.0
1.0
3.0
5.0
ratio_urban
0.4
0.3
0.2
0.1
ratio_mw
0
0.1
0.2
0.3
max_speed_rural
100
90
80
70
60
50


color
2.0
4.0
1.0
3.0
5.0
ratio_precip
0.2
0.15
0.1
ratio_rural
0.4
0.6
0.8
ratio_night
-6
-4
-2
0
2
4
6
8


color
2.0
4.0
1.0
3.0
5.0
four_accel_2
10k
8k
6k
4k
2k
0
ratio_rush
−20k
−10k
0
10k
20k
30k
40k
four_accel_1
15k
10k
5k
0
−5k


color
2.0
4.0
1.0
3.0
5.0
max_accel_y
5
4.5
4
3.5
3
2.5
2
1.5
min_accel_x
−3.5
−3
−2.5
−2
max_accel_x
0.1
0.2

des Clusterings tragen, bzw. zu Leakage führen würden.

Bevor wir hier zur Tat schreiten, ist eine zumindest minimale Verifizierung notwendig. Dazu gehört, dass wir schauen, ob es ungültige Werte gibt:

[39]:
```
telematik_df[pd.isnull(telematik_df).any(1)]
```

[39]:

	IDpol	max_accel_x	min_accel_x	max_accel_y	min_accel_y	\
3492	20917.0	13.91200	-4.8047	2.9565	-2.8993	
23424	2701.0	0.06125	-2.0410	2.2840	-1.8490	
29213	NaN	NaN	NaN	NaN	NaN	
29214	NaN	NaN	NaN	NaN	NaN	

	max_speed_mw	max_speed_urban	max_speed_rural	ratio_mw	ratio_urban	\
3492	NaN	NaN	105.910	0.0	0.0	
23424	129.889	123.839	56.825	NaN	NaN	
29213	NaN	NaN	NaN	NaN	NaN	
29214	NaN	NaN	NaN	NaN	NaN	

	ratio_rural	ratio_night	ratio_precip	ratio_rush	four_accel_1	\
3492	0.0	0.0	0.000	1.000	12409.3079	
23424	NaN	NaN	0.152	-3025.707	56.5730	
29213	NaN	NaN	NaN	NaN	NaN	
29214	NaN	NaN	NaN	NaN	NaN	

	four_accel_2	four_accel_3	four_accel_4	DrivStyle
3492	6924.5246	4659.3696	4273.8503	4.0
23424	0.0000	62.4370	12853.4050	2.0
29213	NaN	NaN	NaN	NaN
29214	NaN	NaN	NaN	NaN

In der Zeile `3492` gibt es zwei NaN-Werte, die wahrscheinlich darauf zurückzuführen sind, dass entweder eine Datenverarbeitung für die Spalten `max_speed_mw` und `max_speed_urban` fehlgeschlagen ist, oder das Verhältnis konnte aufgrund einer zu kurzen oder falschen Aufzeichnung nicht bestimmt werden.

Außerdem ist zu erkennen, dass in Zeile `23424` die Verhältnismerkmale komplett fehlen.

Hier muss man nun überlegen, wie man sinnvoll damit umgehen kann. Die Ansätze, die Werte auf wahlweise das Minimum oder Maximum zu setzen eine offensichtliche Ungleichbehandlung zur Folge haben könnten, ist hier der Mittelwert der restlichen Verteilung zu bevorzugen, da in diesem Falle typischerweise auch das mittlere Risiko liegen sollte. Eine Entfernung der Zeile ist ebenso vorstellbar.

Die beiden letzten Zeilen indes sind auf die `merge`-Operation mit `DrivStyle` zurückzuführen und müssen entfernt werden.

[40]:
```
telematik_df = telematik_df.fillna(telematik_df.mean(),limit=4)
```

Wir verifizieren, dass es keine fehlenden Werte mehr gibt, sodass die Algorithmen ausgeführt werden

können.

[41]:
```
telematik_df = telematik_df.drop([29213,29214],axis=0)
```

[42]:
```
telematik_df[pd.isnull(telematik_df).any(1)]
```

[42]:
```
Empty DataFrame
Columns: [IDpol, max_accel_x, min_accel_x, max_accel_y, min_accel_y,
max_speed_mw, max_speed_urban, max_speed_rural, ratio_mw, ratio_urban,
ratio_rural, ratio_night, ratio_precip, ratio_rush, four_accel_1, four_accel_2,
four_accel_3, four_accel_4, DrivStyle]
Index: []
```

[43]:
```
telematik_df_zuordnung = telematik_df[['IDpol','DrivStyle']]
telematik_df_cluster = telematik_df.drop(['IDpol','DrivStyle'],axis=1)
telematik_df_cluster
```

[43]:

	max_accel_x	min_accel_x	max_accel_y	min_accel_y	max_speed_mw	\
0	0.12450	-2.161	2.055	-1.894	142.134	
1	0.12139	-2.508	3.743	-1.522	260.000	
2	0.09012	-2.173	2.780	-1.359	157.469	
3	0.10508	-2.114	2.540	-1.775	146.727	
4	0.06733	-2.216	3.838	-2.024	250.000	
…	…	…	…	…	…	
29208	0.15333	-2.205	3.463	-1.580	175.175	
29209	0.13192	-2.581	3.965	-1.558	259.581	
29210	0.10092	-2.352	3.940	-1.323	240.000	
29211	0.08094	-2.398	3.926	-1.918	250.000	
29212	0.14058	-2.139	3.402	-0.939	222.605	

	max_speed_urban	max_speed_rural	ratio_mw	ratio_urban	ratio_rural	\
0	123.261	58.095	0.128	0.326	0.546	
1	176.083	70.877	0.081	0.263	0.656	
2	126.513	59.552	0.141	0.231	0.629	
3	113.567	60.670	0.131	0.292	0.577	
4	165.411	73.215	0.176	0.246	0.578	
…	…	…	…	…	…	
29208	134.486	66.659	0.173	0.243	0.585	
29209	169.929	76.348	0.070	0.233	0.698	
29210	149.296	76.555	0.111	0.271	0.618	
29211	160.034	71.075	0.192	0.233	0.575	
29212	153.817	70.839	0.183	0.207	0.610	

	ratio_night	ratio_precip	ratio_rush	four_accel_1	four_accel_2	\
0	0.151116	0.162	-1725.867	-1571.672	105.482	
1	3.855576	0.116	22285.253	9028.015	537.691	
2	-0.159239	0.147	79.762	2025.573	0.000	
3	-3.962291	0.196	-3448.507	2936.377	113.498	

```
4          1.282606          0.171   -1324.465       -452.557          3.748
…             …              …          …              …              …
29208     -0.586203          0.130    2283.903        703.309          0.377
29209      3.607420          0.159   22158.770       7826.207         27.332
29210      1.230867          0.131   -1869.034         -6.010         11.638
29211      2.068949          0.182   11948.122        172.997        435.255
29212      1.667215          0.117    9421.204       4441.900        547.306

       four_accel_3  four_accel_4
0            138.104      7571.877
1            942.168     10583.513
2            743.158     15526.304
3            644.315       556.608
4            912.984      2499.431
…               …            …
29208        487.034      6240.011
29209        798.874     12995.517
29210       2464.727     20000.000
29211        381.740     20000.000
29212        772.529     11261.062

[29213 rows x 17 columns]
```

[44]:
```
telematik_df_cluster[pd.isnull(telematik_df_cluster).any(1)]
```

[44]:
```
Empty DataFrame
Columns: [max_accel_x, min_accel_x, max_accel_y, min_accel_y, max_speed_mw,
max_speed_urban, max_speed_rural, ratio_mw, ratio_urban, ratio_rural,
ratio_night, ratio_precip, ratio_rush, four_accel_1, four_accel_2, four_accel_3,
four_accel_4]
Index: []
```

Wir wählen hier als Anzahl-Heuristik die Verwendung des **Silhouetten-Koeffizienten** aus. Dieser wird für alle vier durchgespielten Fälle geplottet: Kmeans mit/ohne Standardisierung und Ward mit/ohne Standardisierung:

Teil a): Agglomeratives Clustering

Vor dem eigentlichen Clustering wird die Vorverabreitung durchgeführt. Dazu wird ein zweiter, standardisierter Datensatz erzeugt, der über eine Hauptkomponentenanalyse reduziert wird.

[45]:
```
x = telematik_df_cluster.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
telematik_df_cluster_scaled = pd.DataFrame(x_scaled)
```

[ ]:
```

```

Wir führen nun das Clustering für den skalierten und den unskalierten Datensatz durch.

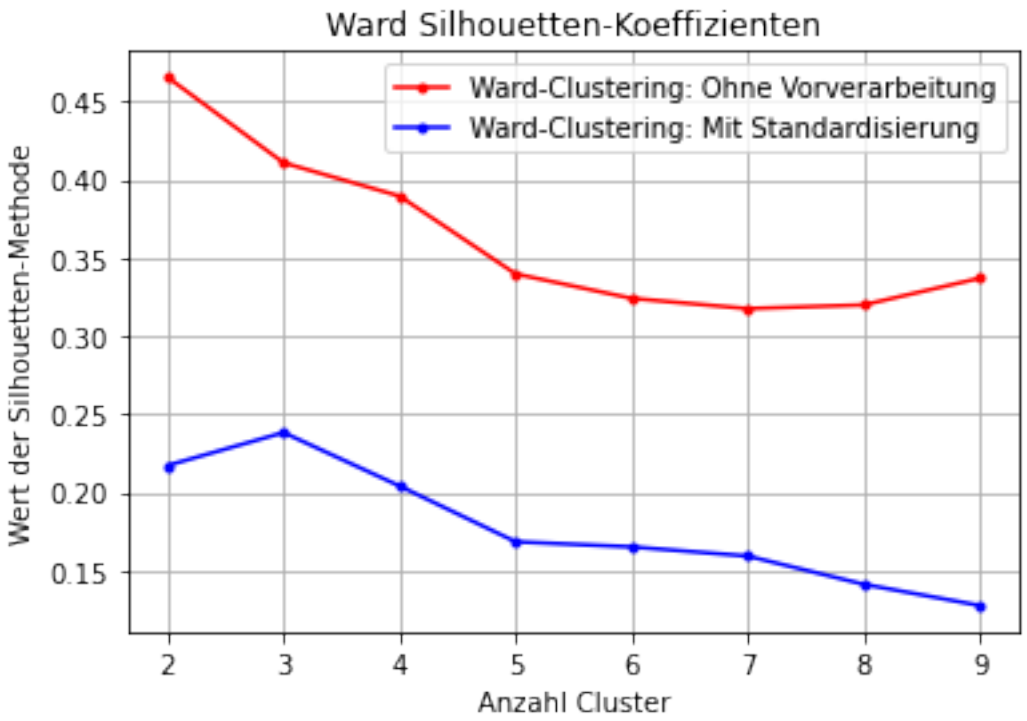
[46]:
```
ward_vanilla_clustering_results = list([0]*10)
ward_standardized_clustering_results = list([0]*10)

for hyper_num_cluster in notebook.tqdm(range(2,10),
                        desc="Berechne Clustering für Vanilla-Fall"):

    vanilla_clustering = AgglomerativeClustering(n_clusters=hyper_num_cluster)
    vanilla_clustering.fit(telematik_df_cluster)
    ward_vanilla_clustering_results[hyper_num_cluster] = vanilla_clustering.
 ↪labels_

for hyper_num_cluster in notebook.tqdm(range(2,10),
                        desc="Berechne Clustering für Standardisierten Fall"):

    standardized_clustering =␣
 ↪AgglomerativeClustering(n_clusters=hyper_num_cluster)
    standardized_clustering.fit(telematik_df_cluster_scaled)
    ward_standardized_clustering_results[hyper_num_cluster] =␣
 ↪standardized_clustering.labels_
```

```
Berechne Clustering für Vanilla-Fall:   0%|          | 0/8 [00:00<?, ?it/s]
```

```
Berechne Clustering für Standardisierten Fall:   0%|          | 0/8 [00:00<?, ?
 ↪it/s]
```

[99]:
```
silhouette_ward_vanilla = list(map(lambda x:metrics.
 ↪silhouette_score(telematik_df_cluster,x),ward_vanilla_clustering_results[2:
 ↪10]))
silhouette_ward_standardized = list(map(lambda x:metrics.
 ↪silhouette_score(telematik_df_cluster_scaled,x),ward_standardized_clustering_results[2:
 ↪10]))
```

[96]:
```
plt.plot(list(range(2,10)),silhouette_ward_vanilla,'.r-',label="Ward-Clustering:
 ↪ Ohne Vorverarbeitung")
plt.plot(list(range(2,10)),silhouette_ward_standardized,'.b-',␣
 ↪label="Ward-Clustering: Mit Standardisierung")
plt.grid(True)
plt.title("Ward Silhouetten-Koeffizienten")
plt.xlabel('Anzahl Cluster')
plt.ylabel('Wert der Silhouetten-Methode')
plt.legend()
plt.show()
```



Teil b): Zentroid-basiertes Clustering

Wir wählen für diese Teilaufgabe die Verwendung von **kmeans** aus und lassen den Algorithmus für 2 . . . 10 Zentroide als Vorgabe durchlaufen.

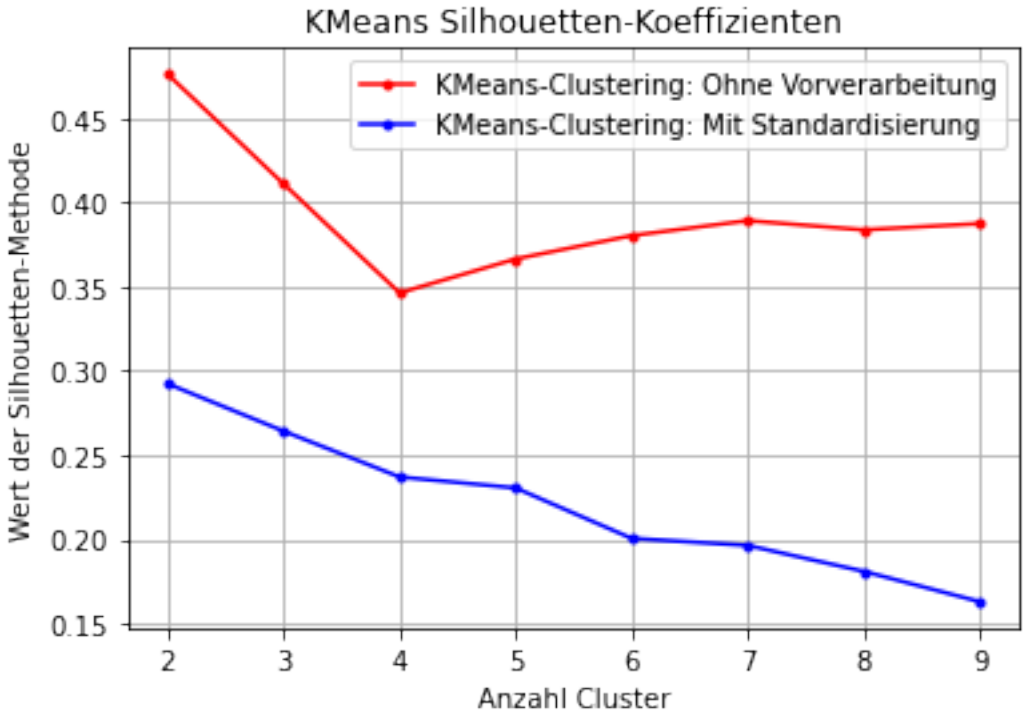
[50]:
```
kmeans_vanilla_clustering_results = list([0]*10)
kmeans_standardized_clustering_results = list([0]*10)

for hyper_num_cluster in notebook.tqdm(range(2,10),
                          desc="Berechne Clustering für Vanilla-Fall"):

    kmeans = KMeans(n_clusters=hyper_num_cluster,␣
 ↪random_state=1111,max_iter=2000).fit(telematik_df_cluster)
    kmeans_vanilla_clustering_results[hyper_num_cluster] = kmeans.labels_

for hyper_num_cluster in notebook.tqdm(range(2,10),
                          desc="Berechne Clustering für Standardisierten Fall"):

    kmeans = KMeans(n_clusters=hyper_num_cluster,␣
 ↪random_state=1111,max_iter=2000).fit(telematik_df_cluster_scaled)
    kmeans_standardized_clustering_results[hyper_num_cluster] = kmeans.labels_
```

```
Berechne Clustering für Vanilla-Fall:   0%|          | 0/8 [00:00<?, ?it/s]
```

```
Berechne Clustering für Standardisierten Fall:   0%|          | 0/8 [00:00<?, ?
 ↪it/s]
```

[51]:
```
silhouette_kmeans_vanilla = list(map(lambda x:metrics.
 ↪silhouette_score(telematik_df_cluster,x),kmeans_vanilla_clustering_results[2:
 ↪10]))
silhouette_kmeans_standardized = list(map(lambda x:metrics.
 ↪silhouette_score(telematik_df_cluster_scaled,x),kmeans_standardized_clustering_results[2:
 ↪10]))
```

[101]:
```
plt.plot(list(range(2,10)),silhouette_kmeans_vanilla,'.
 ↪r-',label="KMeans-Clustering: Ohne Vorverarbeitung")
plt.plot(list(range(2,10)),silhouette_kmeans_standardized,'.b-',␣
 ↪label="KMeans-Clustering: Mit Standardisierung")
plt.grid(True)
plt.title("KMeans Silhouetten-Koeffizienten")
plt.xlabel('Anzahl Cluster')
plt.ylabel('Wert der Silhouetten-Methode')
plt.legend()
plt.show()
```



Es ist zu erkennen, dass die grundsäzliche Größenordnung der Koeffizienten mit der Standardisierung sinkt, was auch zu erwarten ist: Während ohne Standardisierung die unterschiedlichen Skalen der Merkmale den Trennungseigenschaften schaden, scheint mit Standardisierung ein Bias

hervorzutreten, der darauf zurückzuführen ist, dass wenig Erklärungsgehalt in den Beschleunigungsmerkmalen steckt - es kommt also der Fluch der Dimensionen zum Tragen.

Das alles ändert aber nichts daran, dass die Heuristik im Großen und Ganzen unbrauchbar ist - die originalen Cluster können ohne dieses Zusatzwissen nur schwer "erkannt" werden.

1.3.3 Aufgabe B6 (5 Punkte, 5.1.8, 5.1.9, 6.1.1)

Vergleichen Sie die in B5 gebildete Gruppierung mit dem Merkmal "Telematik-Score" aus Teil C. Geben Sie mögliche Gründe an, wodurch die Diskrepanz entstanden sein könnte.

[102]:
```
kmeans = KMeans(n_clusters=5, random_state=1111,max_iter=2000).
 ↪fit(telematik_df_cluster)
predicted_values = kmeans.labels_

confusion_matrix(telematik_df_zuordnung['DrivStyle'] -1 ,predicted_values)
```

[102]:
```
array([[2787,    0, 1000,    4,    0],
       [3042,    0, 2185, 1519,    0],
       [ 394,    0,  959, 3433,  823],
       [   2,   40,  173, 1058, 5626],
       [   0, 5786,    0,    8,  374]], dtype=int64)
```

[103]:
```
perm = {1:1, 2:5, 3:2, 4:3, 5:4}
# Anwenden der neuen Zuordnung
def perm_labels(label,perm):
    return perm[label]
predicted_values_sorted = list(map(lambda x:
 ↪perm_labels(x,perm),predicted_values+1))
confusion_matrix(telematik_df_zuordnung['DrivStyle'],predicted_values_sorted)
```

[103]:
```
array([[2787, 1000,    4,    0,    0],
       [3042, 2185, 1519,    0,    0],
       [ 394,  959, 3433,  823,    0],
       [   2,  173, 1058, 5626,   40],
       [   0,    0,    8,  374, 5786]], dtype=int64)
```

[104]:
```
fig = px.scatter_3d(telematik_df_cluster[:2000], x=telematik_df_cluster.
 ↪columns[2], y=telematik_df_cluster.columns[16], z=telematik_df.columns[11],
              color=kmeans.labels_[:2000].astype(str),width=1200, height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

Wir wollen zusätzlich noch visualisieren, welche Werte hier abweichen. Dazu werden alle off-diagonalen Elemente der Confusion-Matrix mit einem Flag versehen und gegen die anderen Punkte geplottet:

[105]:
```
telematik_df['incorrect_index'] = predicted_values_sorted -␣
↪telematik_df['DrivStyle']

fig = px.scatter_3d(telematik_df_cluster[:2000], x=telematik_df_cluster.
↪columns[2], y=telematik_df_cluster.columns[16], z=telematik_df.columns[11],
              color=telematik_df['incorrect_index'][:2000].
↪astype(str),width=1200, height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

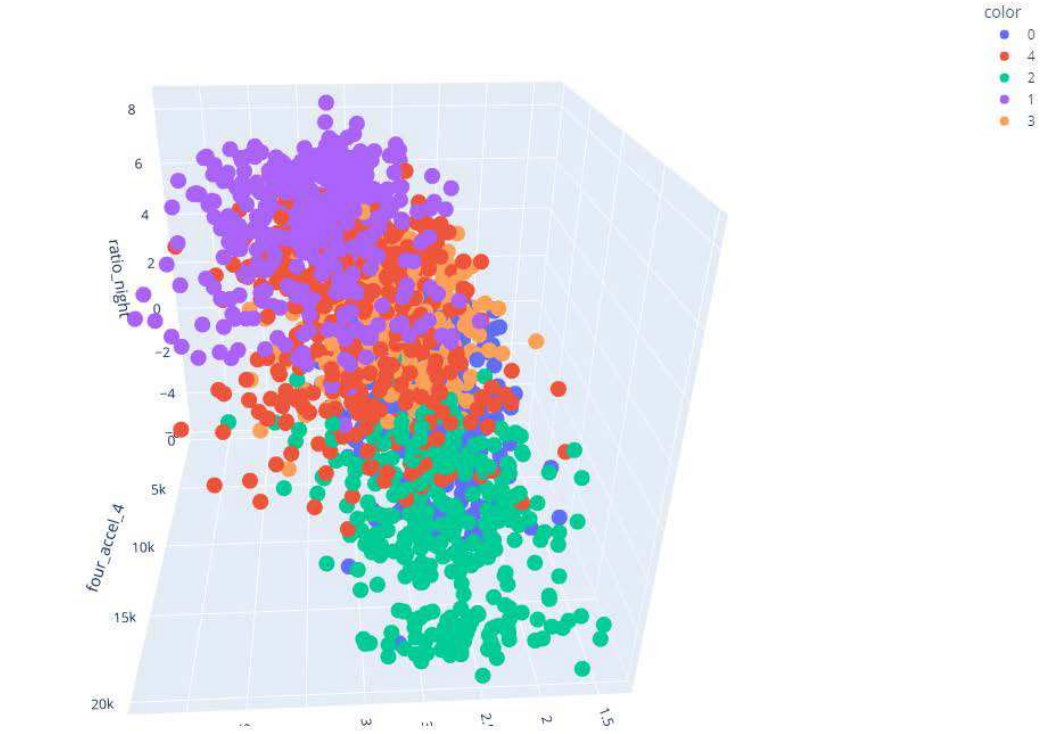
[106]:
```
telematik_df['incorrect_index'] = predicted_values_sorted -␣
↪telematik_df['DrivStyle']

fig = px.scatter_3d(telematik_df_cluster[:2000], x=telematik_df_cluster.
↪columns[9], y=telematik_df_cluster.columns[10], z=telematik_df.columns[12],
              color=telematik_df['incorrect_index'][:2000].
↪astype(str),width=1200, height=780)
fig.update_traces(marker_colorscale='viridis', selector=dict(type='scatter_3d'))
fig.show()
```

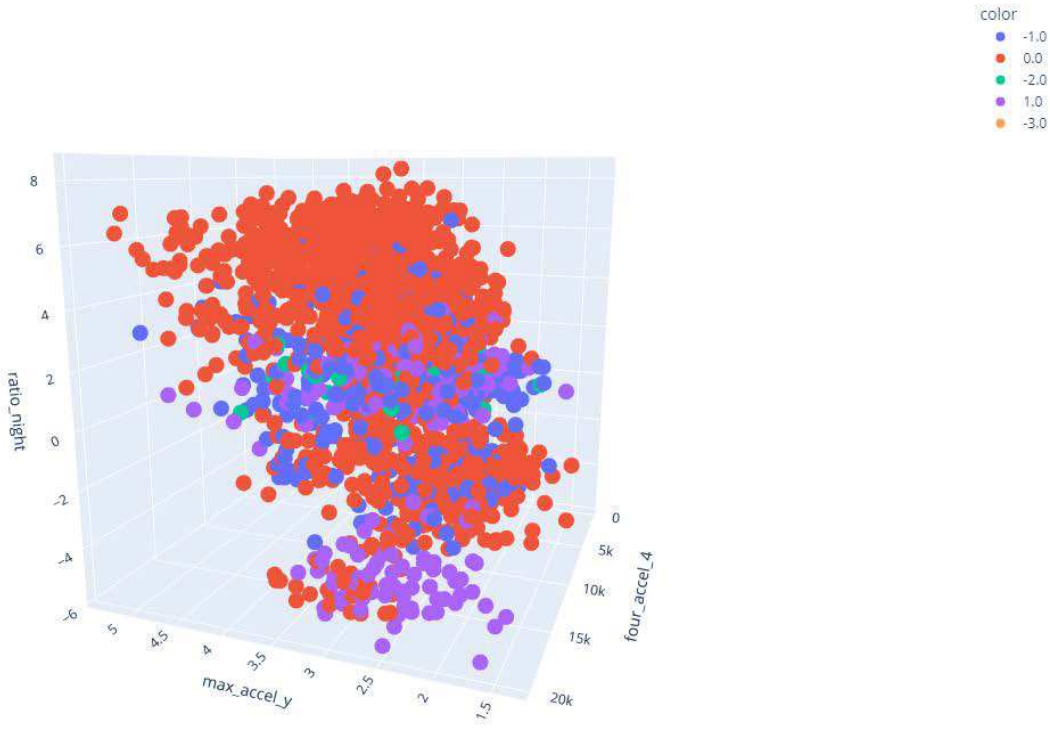
Aus dieser Darstellung erkennt man, dass in der Tat die meisten Abweichungen auf die "verschmierten" Übergangsbereiche zwischen den Clustern zurückzuführen sind. Alle Punkte, die nicht blau sind, stimmen im Label nicht überein. Die Bereiche können hier also auf Basis der vorliegenden Merkmale einfach nicht getrennt werden.

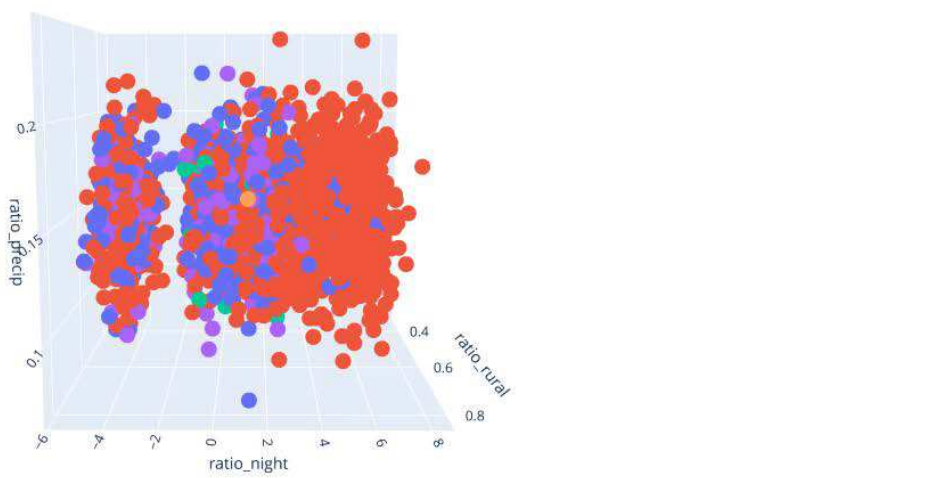
Die Hypothese lautet daher, dass wahrscheinlich weitere Merkmale in den "Score" einfließen, die im Datensatz in der vorliegenden Fassung nicht enthalten sind. Insbesondere die Trennung der Gruppen 1-2 und 3-4 sind unscharf, sodass dem weitere Aspekte zugrunde liegen dürften.

[ ]:

[ ]:


color
0
4
2
1
3
ratio_night
8
6
4
2
0
−2
−4
−6
four_accel_4
5k
10k
15k
20k


color
-1.0
0.0
-2.0
1.0
-3.0
ratio_night
8
6
4
2
0
-2
-4
-6
max_accel_y
5
4.5
4
3.5
3
2.5
2
1.5
four_accel_4
0
5k
10k
15k
20k


color
-1.0
0.0
-2.0
1.0
-3.0
0.2
0.15
0.1
ratio_precip
0.4
0.6
0.8
ratio_rural
-6
-4
-2
0
2
4
6
8
ratio_night

Block C: Schadenprognose und Tarifierung [100 Punkte]

Im Datensatz „MTPLfreq-de.csv“ sind Daten von Fahranfängern enthalten. Der (synthetische) Datensatz weist die folgenden Merkmale auf:

Feldname	Beschreibung
IDPol	Versicherungsnummer
Exposure	Versicherte Zeit als Jahresanteil (1: Ganzjährig)
ClaimNb	Schadenanzahl
Area	Großregion
Density	Fahrzeugdichte in der Region
VehAge	Fahrzeugalter
VehBrand	Fahrzeugmarke (kodiert als B1 bis B12)
VehGas	Kraftstoff
VehPower	Motorleistung
DrivKMT	Jährliche Fahrzeugnutzung in 1000 km
DrivSex	Geschlecht: „F“-Fahrerin, sonst „M“
DrivStyle	Fahrstil (Telematik-Score 1 bis 5)

Hinweise:

- Bei der Verwendung von Gradient-Boosting-Verfahren kann XGBoost durch lightGBM ersetzt werden, sofern dies durchgängig geschieht.

1. Teil: Schadenprognose (Binärklassifikation) [50 Punkte]

Ziel ist hier die Erkennung von besonders schadengefährderter Fahrerinnen und Fahrer, um diesen geeignete Unfallpräventionsmaßnahmen (wie z.B. Fahrsicherheitstraining etc.) anbieten zu können.

Aufgabe C1: Daten einlesen, aufbereiten sowie explorative Datenanalyse [8 Punkte]

Aufgabe C1: a) [Lernziel 6.1] [2 Punkte]

Bevor die eigentliche Verarbeitung beginnt, sollen die notwendigen Programmbibliotheken importiert sowie der Datensatz „freMTPL2freq.csv“ in einen Data-Frame (Python: Pandas) eingelesen werden. Dabei ist sicher zu stellen, dass auch die Merkmale VehGas, VehPower DrivKMT und DrivStyle als kategorielle Merkmale gespeichert werden. Zur Sichtprüfung sollen die letzten Datensätze (vollständig) angezeigt werden. Des Weiteren soll überprüft werden, ob der Datensatz fehlende Werte enthält. Wieviele Datensätze sind für die Verarbeitung verfügbar? Zusätzlich sollen die Ergebnisse reproduzierbar sein.

Lösungsvorschlag:

In [1]:
```
# Benötigte Bibliotheken einbinden
suppressPackageStartupMessages({
library("repr")
library("rpart")
library("rpart.plot")
library("ranger")
library("keras")
library("tidyverse")
library("dplyr")
library("corrplot")
library("pROC")
library("vip")
library("ggplot2")
```

```
library("scales")
library("xgboost")
library("party")
library("caret")
library("gains")
library("gridExtra")
library("ROCR")
library("tibble")
})
```

In [2]:
```
# Print version information about R, the OS and attached or loaded packages
sessionInfo()
```

```
R version 4.0.5 (2021-03-31)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.2 LTS

Matrix products: default
BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=C
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    grid      stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] ROCR_1.0-11       gridExtra_2.3     gains_1.2         caret_6.0-88
 [5] lattice_0.20-44   party_1.3-7       strucchange_1.5-2 sandwich_3.0-0
 [9] zoo_1.8-9         modeltools_0.2-23 mvtnorm_1.1-1     xgboost_1.4.1.1
[13] scales_1.1.1      vip_0.3.2         pROC_1.17.0.1     corrplot_0.88
[17] forcats_0.5.1     stringr_1.4.0     dplyr_1.0.7       purrr_0.3.4
[21] readr_2.0.2       tidyr_1.1.4       tibble_3.1.5      ggplot2_3.3.5
[25] tidyverse_1.3.1   keras_2.3.0.0     ranger_0.12.1     rpart.plot_3.0.9
[29] rpart_4.1-15      repr_1.1.3.9000   bigrquery_1.3.2   httr_1.4.2

loaded via a namespace (and not attached):
 [1] TH.data_1.0-10       colorspace_2.0-2     class_7.3-19
 [4] ellipsis_0.3.2       IRdisplay_1.0.0.9000 base64enc_0.1-3
 [7] fs_1.5.0             rstudioapi_0.13      bit64_4.0.5
[10] prodlim_2019.11.13   fansi_0.5.0          lubridate_1.7.10
[13] coin_1.4-1           xml2_1.3.2           codetools_0.2-18
[16] splines_4.0.5        libcoin_1.0-8        zeallot_0.1.0
[19] IRkernel_1.2.0.9000  jsonlite_1.7.2       broom_0.7.9
[22] dbplyr_2.1.1         png_0.1-7            tfruns_1.5.0
[25] compiler_4.0.5       backports_1.2.1      assertthat_0.2.1
[28] Matrix_1.3-3         fastmap_1.1.0        gargle_1.1.0
[31] cli_3.0.1            htmltools_0.5.2      tools_4.0.5
[34] gtable_0.3.0         glue_1.4.2           reshape2_1.4.4
[37] Rcpp_1.0.7           cellranger_1.1.0     vctrs_0.3.8
[40] nlme_3.1-152         iterators_1.0.13     timeDate_3043.102
[43] gower_0.2.2          rvest_1.0.0          lifecycle_1.0.1
[46] MASS_7.3-54          ipred_0.9-11         hms_1.1.1
[49] parallel_4.0.5       reticulate_1.22      stringi_1.7.5
[52] tensorflow_2.6.0     foreach_1.5.1        lava_1.6.9
[55] rlang_0.4.11         pkgconfig_2.0.3      matrixStats_0.61.0
[58] evaluate_0.14        recipes_0.1.16       bit_4.0.4
[61] tidyselect_1.1.1     plyr_1.8.6           magrittr_2.0.1
[64] R6_2.5.1             generics_0.1.0       multcomp_1.4-17
[67] pbdZMQ_0.3-5         DBI_1.1.1            pillar_1.6.3
[70] haven_2.4.1          whisker_0.4          withr_2.4.2
[73] nnet_7.3-16          survival_3.2-11      modelr_0.1.8
[76] crayon_1.4.1         uuid_0.1-4           utf8_1.2.2
[79] tzdb_0.1.2           readxl_1.3.1         data.table_1.14.2
[82] ModelMetrics_1.2.2.2 reprex_2.0.0         digest_0.6.28
[85] munsell_0.5.0
```

In [3]:
```
# Multiple plot function
# Siehe: http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
```

```
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

 if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}
```

In [4]:
```
# Reproduzierbarkeit: Seed-Wert setzen
set.seed(42)

# Zeichenketten als Faktoren verarbeiten (siehe Versionsunterschiede R 3.x zu R 4.x )
options(stringsAsFactors = TRUE)

# Daten einlesen
dat <- as_tibble(read.csv2("../input/mtplfreq/MTPLfreq-de.csv",header=TRUE))

# Gemäß Aufgabenstellung die Merkmale VehPower, DrivKMT und DrivStyle als kategorielle Merkmale anlegen
# (VehGas bereits durch Option stringsAsFactors behandelt, s.o.)
dat$VehPower <- as.factor(dat$VehPower)
dat$DrivKMT <- as.factor(dat$DrivKMT)
dat$DrivStyle <- as.factor(dat$DrivStyle)

# Ausgabe der letzten Datensätze
tail(dat)
```

```
Warning message in options(stringsAsFactors = TRUE):
“'options(stringsAsFactors = TRUE)' is deprecated and will be disabled”
```

A tibble: 6 × 12

IDpol	ClaimNb	Exposure	Area	VehPower	VehAge	VehBrand	VehGas	Density	DrivStyle	DrivSex	DrivKMT
<int>	<int>	<dbl>	<fct>	<fct>	<int>	<fct>	<fct>	<int>	<fct>	<fct>	<fct>
29211	0	0.0100	E	3	3	B12	Regular	2981	4	F	20
29212	0	0.0100	E	3	3	B12	Regular	2981	2	M	9
29213	0	0.0100	D	3	3	B10	Diesel	1530	4	F	15
29214	0	0.0055	E	1	0	B12	Regular	2981	4	F	9
29215	0	0.0100	E	2	3	B12	Regular	4160	2	M	12
29216	0	0.0055	D	4	5	B1	Regular	786	2	F	20

In [5]:
```
# Kurze Übersicht über die Merkmalswerte
summary(dat)
```

```
     IDpol          ClaimNb           Exposure      Area        VehPower
 Min.   :    1   Min.   :0.00000   Min.   :0.0027   A:4159   1      :7057
 1st Qu.: 7305   1st Qu.:0.00000   1st Qu.:0.1200   B:3083   3      :6753
 Median :14608   Median :0.00000   Median :0.3300   C:8694   2      :6485
 Mean   :14608   Mean   :0.07869   Mean   :0.4095   D:7250   4      :5155
 3rd Qu.:21912   3rd Qu.:0.00000   3rd Qu.:0.6600   E:5673   6      :1308
 Max.   :29216   Max.   :3.00000   Max.   :1.0000   F: 357   5      :1282
                                                             (Other):1176
     VehAge          VehBrand        VehGas         Density      DrivStyle
 Min.   : 0.000   B2     :8402   Diesel :11869   Min.   :   10   1:3792
 1st Qu.: 4.000   B1     :7254   Regular:17347   1st Qu.:  106   2:6748
 Median : 9.000   B3     :3874                   Median :  403   3:5610
 Mean   : 8.532   B12    :3759                   Mean   : 1269   4:6898
 3rd Qu.:13.000   B5     :1701                   3rd Qu.: 1530   5:6168
 Max.   :38.000   B4     :1524                   Max.   :10000
                  (Other):2702
 DrivSex      DrivKMT
 F:13367   9      :4938
 M:15849   12     :4447
           6      :3857
           20     :3586
           15     :3389
           40     :2933
           (Other):6066
```

In [6]:
```r
# Fehlende Werte zählen
sum(is.na(dat))
```

0

In [7]:
```r
# Anzahl Datensätze und Merkmale
dim(dat)
```

$29216 \cdot 12$

Der Datensatz enthält keine "Missings" (fehlende Werte). Insgesamt sind 29.216 Datensätze und 12 Merkmale (auch Variablen oder schlicht Spalten genannt) vorhanden.

Aufgabe C1: b) [Lernziel 6.1] [1 Punkt]

Aus dem Merkmal „ClaimNb“ soll ein neues binäres Merkmal „Claims“ („nicht schadenfrei“) mit dem Wert 0 für Schadenfreiheit und andernfalls 1 abgeleitet, dem Datensatz hinzugefügt und als Zielgröße verwendet werden. Für Claims soll eine Häufigkeitsauszählung der Merkmalswerte absolut und prozentual durchgeführt und ausgegeben werden. Warum sollten diese Angaben nicht als Schadenhäufigkeit bezeichnet werden? Exposure und die ClaimNb kann für die weiteren Analysen in diesem Abschnitt ausgeschlossen werden.

Lösungsvorschlag:

In [8]:
```r
# Ableitung des binären Merkmals „Claims“ („nicht schadenfrei“)
dat$Claims <- pmin(1,dat$ClaimNb)

# Auszählung der Häufigkeit schadenfreier / nicht schadenfreier Policen (absolut und relativ):
print("Absolute Häufigkeit schadenfreier / nicht schadenfreier Policen:")
table(dat$Claims)
paste("Relative Häufigkeit schadenfreier / nicht schadenfreier Policen (in %):")
table(dat$Claims)/dim(dat)[1]*100
```

```
[1] "Absolute Häufigkeit schadenfreier / nicht schadenfreier Policen:"
    0     1
27102  2114
```

'Relative Häufigkeit schadenfreier / nicht schadenfreier Policen (in %):'

```
        0         1
92.764239  7.235761
```

Da in die Berechnung der Schadenhäufigkeit alle Schäden eingehen und in der oben angestellten Betrachtung höchstens ein Schaden je Police gezählt wird, sollte hier zur Vermeidung von Verwechslungen der Begriff Schadenhäufigkeit nicht verwendet werden. Schadenhäufigkeit wird im Teil 2 dieses Notebooks betrachtet.

In [9]:
```r
# Neue Datei für die folgenden Aufgaben der Binärkassifikation erzeugen,
# ohne die Variablen ClaimNb und Exposure
# (die Datenmenge ist recht klein, daher sind weitere Kopien unproblematisch)

dat0 <- dat %>% select(-c(ClaimNb,Exposure))
print(dat0)
```

```
# A tibble: 29,216 × 11
   IDpol Area  VehPower VehAge VehBrand VehGas  Density DrivStyle DrivSex DrivKMT
   <int> <fct> <fct>     <int> <fct>    <fct>     <int> <fct>     <fct>   <fct>
 1     1 E     6             4 B6       Regular    8103 2         M       15
 2     2 D     6             4 B12      Regular     563 1         M       15
 3     3 D     6             4 B12      Regular     563 4         F       12
 4     4 D     7             5 B12      Regular    1530 1         M       25
 5     5 E     5             4 B14      Regular    2981 1         M       40
 6     6 E     1             1 B12      Regular    2981 4         M       3
 7     7 F     6             8 B12      Regular   10000 2         M       20
 8     8 F     6             8 B12      Regular   10000 1         M       40
 9     9 B     5             9 B12      Regular     106 2         M       15
10    10 C     5             9 B12      Regular     207 2         M       20
# … with 29,206 more rows, and 1 more variable: Claims <dbl>
```

Aufgabe C1: c) [Lernziel 6.1] [5 Punkte]

Für jedes Merkmal (ohne IDPol) sind die absoluten und relativen Häufigkeiten differenziert nach Claims als Säulengraphiken kompakt darzustellen. Numerische Merkmalswerte (z.B. Density) sind dabei geeignet zusammenzufassen. Welche Auffälligkeiten ergeben sich? Was kann, in Kombination mit den Ergebnissen aus dem vorherigen Teil, über den Datensatz und insbesondere über mögliche Performancemetriken gesagt werden?

Lösungsvorschlag:

In [10]:
```r
# Darstellung der absoluten und relativen Häufigkeit differenziert nach Claims als Säulengraphik

options(repr.plot.width=10, repr.plot.height=10)

# Claims als factor für die weiteren Darstellungen
dat_ex <- dat0 %>%
    mutate_at(vars(Claims), factor)

# Merkmal Area Type
dat1 <- dat_ex %>% group_by(Area, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g1 <- ggplot(dat_ex) +
    geom_bar(mapping = aes(x=Area, fill=Claims)) +
    labs(x = "Area Type", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g2 <- ggplot(dat1, aes(x=Area, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7) +
    labs(x = "Area Type", y = "Prozent", fill = "Claims") +
    ylim(0, 20)

# Merkmal Vehicle Brand
dat1 <- dat_ex %>% group_by(VehBrand, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g3 <- ggplot(dat_ex) +
    geom_bar(mapping = aes(x=VehBrand, fill=Claims)) +
    labs(x = "Vehicle Brand", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g4 <- ggplot(dat1, aes(x=VehBrand, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7, ylim = c(0, 20)) +
    labs(x = "Vehicle Brand", y = "Prozent", fill = "Claims") +
    ylim(0, 20)

# Merkmal Vehicle Gas
dat1 <- dat_ex %>% group_by(VehGas, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g5 <- ggplot(dat_ex) +
    geom_bar(mapping = aes(x=VehGas, fill=Claims)) +
    labs(x = "Vehicle Gas", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g6 <- ggplot(dat1, aes(x=VehGas, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7) +
    labs(x = "Vehicle Gas", y = "Prozent", fill = "Claims") +
    ylim(0, 20)

# Merkmal Driver Sex
dat1 <- dat_ex %>% group_by(DrivSex, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g7 <- ggplot(dat_ex) +
    geom_bar(mapping = aes(x=DrivSex, fill=Claims)) +
    labs(x = "Driver Sex", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g8 <- ggplot(dat1, aes(x=DrivSex, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7) +
```

```
    labs(x = "Driver Sex", y = "Prozent", fill = "Claims") +
    ylim(0, 20)

# Merkmal Vehicle Power
dat1 <- dat_ex %>% group_by(VehPower, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g9 <- ggplot(dat_ex) +
    geom_bar(mapping = aes(x=VehPower, fill=Claims)) +
    labs(x = "Vehicle Power", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g10 <- ggplot(dat1, aes(x=VehPower, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7) +
    labs(x = "Vehicle Power", y = "Prozent", fill = "Claims") +
    ylim(0, 20)

# Merkmal Driver Style
dat1 <- dat_ex %>% group_by(DrivStyle, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g11 <- ggplot(dat_ex) +
    geom_bar(mapping = aes(x=DrivStyle, fill=Claims)) +
    labs(x = "Driver Style", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g12 <- ggplot(dat1, aes(x=DrivStyle, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7) +
    labs(x = "Driver Style", y = "Prozent", fill = "Claims") +
    ylim(0, 20)

# Merkmal VehAge
dat1AgGr <- mutate(dat_ex, VehAgeGr = case_when( VehAge <= 5 ~ "<= 05"
                                                , VehAge <= 10 ~ "<= 10"
                                                , VehAge <= 15 ~ "<= 15"
                                                , VehAge <= 20 ~ "<= 20"
                                                , VehAge <= 25 ~ "<= 25"
                                                , VehAge <= 30 ~ "<= 30"
                                                , VehAge <= 35 ~ "<= 35"
                                                , VehAge <= 40 ~ "<= 40"))
dat1 <- dat1AgGr %>% group_by(VehAgeGr, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g13 <- ggplot(dat1AgGr) +
    geom_bar(mapping = aes(x=VehAgeGr, fill=Claims)) +
    labs(x = "FzgAlter", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g14 <- ggplot(dat1, aes(x=VehAgeGr, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7) +
    labs(x = "FzgAlter", y = "Prozent", fill = "Claims") +
    theme_minimal()  +
    ylim(0, 20)

# Merkmal DrivKMT
dat1 <- dat_ex %>% group_by(DrivKMT, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g15 <- ggplot(dat_ex) +
    geom_bar(mapping = aes(x=DrivKMT, fill=Claims)) +
    labs(x = "DrivKMT", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g16 <- ggplot(dat1, aes(x=DrivKMT, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7) +
    labs(x = "DrivKMT", y = "Prozent", fill = "Claims")  +
    ylim(0, 20)

# Merkmal Density
dat1Dens <- mutate(dat_ex, DensityGr = case_when( Density <= 1000 ~ "<= 01T"
                                                , Density <= 2000 ~ "<= 02T"
                                                , Density <= 3000 ~ "<= 03T"
                                                , Density <= 4000 ~ "<= 04T"
                                                , Density <= 5000 ~ "<= 05T"
                                                , Density <= 6000 ~ "<= 06T"
                                                , Density <= 7000 ~ "<= 07T"
                                                , Density <= 8000 ~ "<= 08T"
                                                , Density <= 9000 ~ "<= 09T"
                                                , Density <= 10000 ~ "<= 10T"
                                                , Density <= 11000 ~ "<= 11T"))
dat1 <- dat1Dens %>% group_by(DensityGr, Claims) %>%
    summarize(count = n()) %>% mutate( pc = count /sum(count))
g17 <- ggplot(dat1Dens) +
    geom_bar(mapping = aes(x=DensityGr, fill=Claims)) +
    labs(x = "Density", y = "Anzahl", fill = "Claims") +
    scale_fill_manual(values=c("#00BFC4","#F8766D"))
g18 <- ggplot(dat1, aes(x=DensityGr, y=pc*100, fill=Claims)) +
    geom_bar(stat="Identity", width = 0.7) +
```

```
    labs(x = "Density", y = "Prozent", fill = "Claims")  +
    ylim(0, 20)

multiplot(g1, g3, g5, g7, g9, g11, g13, g15, g17, g2, g4, g6, g8, g10, g12, g14, g16, g18, cols=2)
```

```
`summarise()` has grouped output by 'Area'. You can override using the `.groups` argument.

`summarise()` has grouped output by 'VehBrand'. You can override using the `.groups` argument.

Warning message:
“Ignoring unknown parameters: ylim”
`summarise()` has grouped output by 'VehGas'. You can override using the `.groups` argument.

`summarise()` has grouped output by 'DrivSex'. You can override using the `.groups` argument.

`summarise()` has grouped output by 'VehPower'. You can override using the `.groups` argument.

`summarise()` has grouped output by 'DrivStyle'. You can override using the `.groups` argument.

`summarise()` has grouped output by 'VehAgeGr'. You can override using the `.groups` argument.

`summarise()` has grouped output by 'DrivKMT'. You can override using the `.groups` argument.

`summarise()` has grouped output by 'DensityGr'. You can override using the `.groups` argument.

Warning message:
“Removed 6 rows containing missing values (position_stack).”
Warning message:
“Removed 11 rows containing missing values (position_stack).”
Warning message:
“Removed 2 rows containing missing values (position_stack).”
Warning message:
“Removed 2 rows containing missing values (position_stack).”
Warning message:
“Removed 10 rows containing missing values (position_stack).”
Warning message:
“Removed 5 rows containing missing values (position_stack).”
Warning message:
“Removed 8 rows containing missing values (position_stack).”
Warning message:
“Removed 10 rows containing missing values (position_stack).”
Warning message:
“Removed 7 rows containing missing values (position_stack).”
```

Interpretation:

Aus den Grafiken ist erkennbar, dass AreaType=1 prozentual gesehen ungefähr gleichverteilt ist. Gleiches gilt für den Treibstofftyp (Diesel vs. Regular). Aus dem Feature Geschlecht des Fahrers ist eine erhöhte Schadenwahrscheinlichkeit für männliche Fahrer erkennbar. Bezüglich der Motorleistung ist eine erhöhte Schadenwahrscheinlichkeit mit steigen Leistung erkennbar. Der Fahrstil scheint Personen mit hoher Schadenwahrscheinlichkeit in niedrige Klassen einzuordnen. Bei Fahrzeugalter ist erkennbar, dass bei jungen Wagen (Alter kleiner als 10 Jahr) eine erhöhte Schadenwahrscheinlichkeit besteht. Ab einem Alter von ca. 20 Jahren zeigen sich, basierend auf wenigen Beobachtungen, Ausreißer.

Mit Blick auf die Schadenzahlen ist erkennbar, dass es sich um einen unbalanzierten Datensatz handelt (rund 7 % der Policen haben Schäden). Damit ist die Accuracy bereits keine geeignete Performancemetrik (eine triviale Klassifikation, die alle Datenpunkte auf die häufigste Klasse (0) abbildet, erreicht bereits eine hohe Accuracy).

Aufgabe C2: Daten für die Modellierung vorbereiten [6 Punkte]

Aufgabenteil C2: a) [Lernziel 6.1] [2 Punkte]

Die Konzepte Holdout- und Kreuzvalidierung sind zu diskutieren. Welche Vor- und Nachteile bringen diese mit? Im Anschluss ist der Datensatz geeignet in einen Trainings- und Validierungsdatensatz zu teilen. Verifizieren Sie Ihre Ergebnisse.

Lösungsvorschlag:

Bei der Holdoutvalidierung wird der Datensatz in (mindestens) zwei Teile unterteilt:

- Trainingsdatensatz: Anteil der Daten, der für das Training des Modells verwendet wird
- Testdatensatz: Dient zur Bestimmung der Güte/Performance des Modells
- Validierungssatz: Dient zur Optimierung der Modellhyperparameter

Ziel ist eine Schätzung des Testfehlers. Das Vorgehen bei der Holdoutvalidierung und die Implementierung des Verfahrens ist einfach. Als Nachteil der Methode ist die hohe Varianz zu nennen (abhängig davon wie die Beobachtungen in Test- und Validierungsdatensatz eingeteilt werden). Zudem wird beim Trainieren des Modells immer auf einen gewissen Anteil der Beobachtungen verzichtet. Das spricht dafür, dass durch diesen Ansatz der Testfehler überschätzt wird.

Bei der (k-fachen) Kreuzvalidierung wird eine Einteilung des Datensatzen in k (möglichst gleich großen) Gruppen vorgenommen. Auf k-1 Gruppen wird das Training vorgenommen, auf der verbleibenden Gruppe ein Testfehler ermittelt. Dies wird wiederholt, wobei die Testgruppe bei jedem durchlauf eine andere ist. Bei jedem Durchlauf wird ein Testfehler ermittelt. Letztlich wird eine Schätzung des Testfehlers als arithmetisches Mittel der einzelnen Testfehlerwerte bestimmt. Im Gegensatz zur Holdoutmethode fließen alle Daten in das Training der Methode (zu einem bestimmten Zeitpunkt) ein. Zudem liefert die Holdoutmethode bedingt durch den zufälligen Split verschiedene Ergebnisse. Das ist bei der (k-fachen) Kreuzvalidierung nicht der Fall. Allerdings ist diese auch (abhängig von der Wahl von k) rechenintensiver.

In [11]:
```
# Trennung der Daten in zwei Stichproben: 80% Training und 20% Validierung
trn <- sample(c(1:nrow(dat0)), round(0.8*nrow(dat0)), replace = FALSE)

# Stichproben train/val ohne IDpol
train <- dat0[trn,c(2:11)]
val <- dat0[-trn,c(2:11)]

train_orig <- train
val_orig <- val

# Auszählung der Häufigkeit: Absolut
table(train$Claims)
table(val$Claims)
```

```
    0     1
21685  1688
   0    1
5417  426
```

In [12]:
```
# Auszählung der Häufigkeit: Prozentual (Stimmen Anteile überein?)
prop.table(table(train$Claims))
prop.table(table(val$Claims))
```

```
         0          1
0.92777992 0.07222008
```

```
        0         1
0.92709225 0.07290775
```

Aufgabenteil C2: b) [Lernziel 6.1] [2 Punkte]

Diskutieren Sie im Folgenden verschiedene Metriken für die Bewertung der Modellgüte. Gehen Sie dabei auch auf F_{beta} ein.

Lösungsvorschlag:

Basierend auf der Konfusionsmatrix

		Predicted
Actual	TN	FP
	FN	TP

lassen sich die folgenden Performance-Matriken ableiten:

- Accuracy: $\frac{TP+TN}{TP+FP+FN+TN}$

  Diese gibt den Anteil der richtigen Vorhersagen an. Diese ist, wie bereits festgestellt wurde, in vorliegenden Fall eines unbalancierten Datensatzes keine geeignete Metrik.

- Precision: $\frac{TP}{TP+FP}$

  Diese gibt den Anteil der richtig vorhergesagten Beobachtungen (TP) bezogen auf die Gesamtheit aller als positiven Ereignisse vorhergesagten Ergebnisse an. Beispielsweise wäre bei einem Spam-Filter für E-Mails eine hohe Precision wünschenswert, um wichtige E-Mails nicht zu verlieren.

- Recall: $\frac{TP}{TP+FN}$

  Der Recall gibt den Anteil der korrekt als positiv klassifizierten Ergebnisse (TP) im Bezug auf die Gesamtheit der tatsächlich positiven Ergebnisse an. Im Beispiel des Spam-Filters wäre ein hoher Recall nicht unbedingt wichtig, da nicht unbedingt jede Spam-Mail herausgefiltert werden muss. Wenn es allerdings um die Klassifikation von Filmen in die Kategorien jugendfrei (0) oder nicht-jugendfrei (1) geht, wäre ein hoher Recall wünschenswert.

- Specificity: $\frac{TN}{TN+FP}$

  Die Specificity gibt den Anteil der korrekt als negativ klassifizierten Beobachtungen (TN) in Bezug auf die Gesamtheit der tatsächlich negativen Ergebnisse an. Im Fall einer Betrugserkennung bei Transaktionen (Betrug=1) ergibt sich damit der Anteil der nicht-betrügerischen Transaktionen die im Bezug auf alle Transaktionen korrekt erkannt wurden.

- False Discovery Rate (FDR): $\frac{FP}{TP+FP}$

  Diese Metrik misst bei allen als positiv vorhergesagten Beobachtungen den Anteil, der falsch zugeordnet wurde.

Die Metrik F_{beta} ist definiert durch $F_{beta} = (1+\beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$. Der Parameter β steuert hierbei Kompromiss zwischen Precision und Recall: Für $0 < \beta < 1$ liegt der Fokus mehr auf Precision, für $\beta > 1$ mehr auf dem Recall. Für $\beta = 1$ ergibt sich F_1 als harmonisches Mittel aus Precision und Recall.

Aufgabenteil C2: c) [Lernziel 6.1] [2 Punkte]

Was ist eine ROC-Kurve? Wie kann eine ROC-Kurve bei der Bewertung der Modellgüte nutzen? Was ist die Area under curve (AUC)? Warum kann es sinnvoll sein, sich zur Berurteilung der Modellgüte neben der Anzeige der AUC auch die ROC-Kurve zu plotten?

Lösungsvorschlag:

Die ROC-Kurve (ROC steht für Receiver Operator Characteristic) zeigt für ein Modell den Trade-off zwischen der Sensitivität des Modells (der TPR - also True Positive Rate, auch Recall genannt (s.o.) TPR = TP/(TP + FN)) und der Spezifizität (1 – FPR, wobei FPR die False Positive Rate bezeichnet FPR = FP/(TN + FP)).

Je höher die Kurve an die linke obere Ecke kommt, desto besser ist das Modell. Ein zufälliger Klassifizierer wird ein Ergebnis nahe der Diagonalen liefern, wo die TPR der FPR entspricht.

Da die ROC-Kurve unabhängig ist von der Verteilung der zu schätzenden Größe (in unserem Fall die Variable Claims), ist die Darstellung als ROC-Kurve besonders hilfreich, wenn seltene Ereignisse prognostiziert werden sollen.

Die AUC beschreibt die Fläche unter der ROC-Kurve. Zum Vergleich mehrere Modelle kann es hilfreich sein, sich die AUC berechnen zu lassen. Ein Modell mit hoher AOC kann möglicherweise in einigen Bereichen schlechter performen als ein Modell mit geringerer AUC. Dies kann man in der Visualisierung der ROC-Kurve sehen und ggf. berücksichtigen.

Aufgabe C3: Klassische logistische Regression [14 Punkte]

Aufgabenteil C3: a) [Lernziel 4.1] [2 Punkte]

Für die Vorhersage der binären Variablen Claims ist ein logistisches Modell zu verwenden, welches sämtliche verfügbaren Features beinhalten soll. Eine Normalisierung / Regularisierung ist dabei nicht vorzunehmen. Das Ergebnis ist im Anschluss kurz zu interpretieren.

Lösungsvorschlag:

In [13]:
```r
# Durchführung der logistischen Regression
GLM1 <- glm(Claims ~., data=train, family=binomial)
summary(GLM1)
```

```
Call:
glm(formula = Claims ~ ., family = binomial, data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.9326  -0.4213  -0.3432  -0.2818   2.8616

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)   -3.766e+00  2.693e-01 -13.985  < 2e-16 ***
AreaB          3.033e-01  1.027e-01   2.953 0.003146 **
AreaC         -4.065e-02  8.915e-02  -0.456 0.648417
AreaD          1.935e-01  9.495e-02   2.038 0.041511 *
AreaE          1.141e-01  1.578e-01   0.723 0.469887
AreaF          9.769e-02  3.847e-01   0.254 0.799535
VehPower2      2.049e-01  8.592e-02   2.385 0.017083 *
VehPower3      2.275e-01  8.640e-02   2.633 0.008472 **
VehPower4      2.370e-01  8.902e-02   2.663 0.007753 **
VehPower5      7.808e-01  1.189e-01   6.568 5.11e-11 ***
VehPower6      7.735e-01  1.194e-01   6.479 9.24e-11 ***
VehPower7      1.150e+00  1.364e-01   8.429  < 2e-16 ***
VehPower8      1.132e+00  2.244e-01   5.044 4.56e-07 ***
VehPower9      8.971e-01  3.082e-01   2.911 0.003607 **
VehPower10     1.807e-01  3.322e-01   0.544 0.586553
VehAge        -1.699e-02  5.658e-03  -3.003 0.002673 **
VehBrandB10   -1.738e-01  2.892e-01  -0.601 0.547919
VehBrandB11    1.240e-01  1.660e-01   0.747 0.455099
VehBrandB12   -2.526e-01  1.023e-01  -2.470 0.013525 *
VehBrandB13   -7.167e-01  3.175e-01  -2.257 0.023993 *
VehBrandB14    1.979e-01  4.400e-01   0.450 0.652956
VehBrandB2     2.352e-02  7.184e-02   0.327 0.743394
VehBrandB3     5.615e-02  8.742e-02   0.642 0.520696
VehBrandB4    -1.633e-01  1.335e-01  -1.223 0.221273
VehBrandB5     1.469e-01  1.145e-01   1.283 0.199532
VehBrandB6     1.086e-01  1.209e-01   0.898 0.369146
VehGasRegular  2.483e-01  6.565e-02   3.782 0.000156 ***
Density        2.367e-05  3.146e-05   0.752 0.451886
DrivStyle2    -7.700e-03  8.058e-02  -0.096 0.923870
DrivStyle3    -2.687e-01  9.864e-02  -2.724 0.006449 **
DrivStyle4    -3.661e-01  1.066e-01  -3.433 0.000598 ***
DrivStyle5    -2.780e-01  1.222e-01  -2.274 0.022946 *
DrivSexM       4.383e-01  7.200e-02   6.087 1.15e-09 ***
DrivKMT6       3.731e-01  2.059e-01   1.812 0.070062 .
DrivKMT9       4.701e-01  2.066e-01   2.275 0.022911 *
DrivKMT12      5.632e-01  2.105e-01   2.675 0.007478 **
DrivKMT15      7.480e-01  2.147e-01   3.484 0.000494 ***
DrivKMT20      8.139e-01  2.169e-01   3.752 0.000175 ***
DrivKMT25      8.785e-01  2.256e-01   3.894 9.85e-05 ***
DrivKMT30      1.097e+00  2.300e-01   4.771 1.83e-06 ***
DrivKMT35      1.001e+00  2.407e-01   4.159 3.20e-05 ***
DrivKMT40      1.233e+00  2.215e-01   5.568 2.58e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 12123  on 23372  degrees of freedom
Residual deviance: 11612  on 23331  degrees of freedom
AIC: 11696

Number of Fisher Scoring iterations: 6
```

Das Modell der logistischen Regression zeigt signifikante Variablen insbesondere bei $VehPower$, $VehGasRegular$, beim Fahrstil $DrivStyle$, beim Geschlecht $DrivSex$ sowie bei den gefahrenen Kilometern $DrivKMT$. Diese Ergebnisse sind größtenteils in Einklang mit den Ergebnissen der explorativen Datenanalyse.

Aufgabenteil C3: b) [Lernziel 6.1] [4 Punkte]

Die Ergebnisse aus Teil C3: a) sollten genutzt werden, um unterschiedliche Vorhersagen auf den Validierungsdaten vorzunehmen. Hierfür sollen drei unterschiedliche Schwellenwerte verwendet werden, um eine binäre Vorhersage für Claims zu treffen.

Für jeden gewählten Schwellenwert soll die Confusion-Matrix sowie Accuracy, Precision, Recall (auch True Positive Rate bzw. Sensitivity) und Specificity berechnet werden. Geben Sie für jeden gewählten Schwellenwert die Confusion-Matrix aus und berechnen Sie jeweils Accuracy, Precision, Recall (auch True Positive Rate, Sensitivity) und Spezifizität.

Lösungsvorschlag:

In [14]:
```
# Vorhersagen erstellen
val$fitGLM1<-predict(GLM1, newdata=val,type="response")
head(val)
```

A tibble: 6 × 11

Area	VehPower	VehAge	VehBrand	VehGas	Density	DrivStyle	DrivSex	DrivKMT	Claims	fitGLM1
<fct>	<fct>	<int>	<fct>	<fct>	<int>	<fct>	<fct>	<fct>	<dbl>	<dbl>
D	6	4	B12	Regular	563	4	F	12	1	0.06530787
E	5	4	B14	Regular	2981	1	M	40	1	0.32056676
F	6	8	B12	Regular	10000	1	M	40	1	0.24467445
B	4	8	B2	Regular	55	2	M	9	1	0.10082029
D	7	5	B12	Regular	1530	1	M	12	1	0.18624737
D	1	0	B12	Regular	1530	2	M	9	1	0.06659740

In [15]:
```
getMeasures <- function(Claims, predictedClaims){
    CM = table(Claims, predictedClaims)
    print(CM)

    TN =CM[1,1]
    TP =CM[2,2]
    FP =CM[1,2]
    FN =CM[2,1]

    accuracy  = (TP+TN)/(TP+TN+FP+FN) # wieviele Datensätze wurden korrekt klassifiziert
    precision = (TP)/(TP+FP) # welcher Anteil der erkannten Schäden war tatsächlich ein Schaden
    recall    = (TP)/(TP+FN) # welcher Anteil der Schäden wurde als solcher erkannt
    fpr       = (FP)/(FP+TN) # welcher Anteil der Nicht-Schäden wurde als Schaden erkannt

    print(paste("Accuracy:   ", round(accuracy,3)))
    print(paste("Precision:  ", round(precision,3)))
    print(paste("Recall/TPR: ", round(recall,3)))# TPR
    print(paste("FPR:        ", round(fpr,3)))# FPR
    print(paste("Specificity:", round(1-fpr,3)))
}
```

In [16]:
```
# GLM mit einem Schwellenwert von 0.3
GLM_03 <- val %>% select(Claims, fitGLM1) %>% mutate( fitGLM_03_binary = ifelse(fitGLM1 > 0.3, 1, 0))
getMeasures(GLM_03$Claims, GLM_03$fitGLM_03_binary)
```

```
      predictedClaims
Claims    0    1
     0 5411    6
     1  422    4
[1] "Accuracy:    0.927"
[1] "Precision:   0.4"
[1] "Recall/TPR:  0.009"
[1] "FPR:         0.001"
[1] "Specificity: 0.999"
```

In [17]:
```
# GLM mit einem Schwellenwert von 0.2
GLM_02 <- val %>% select(Claims, fitGLM1) %>% mutate( fitGLM_02_binary = ifelse(fitGLM1 > 0.2, 1, 0))
getMeasures(GLM_02$Claims, GLM_02$fitGLM_02_binary )
```

```
      predictedClaims
Claims    0    1
     0 5345   72
     1  393   33
[1] "Accuracy:    0.92"
[1] "Precision:   0.314"
[1] "Recall/TPR:  0.077"
[1] "FPR:         0.013"
[1] "Specificity: 0.987"
```

In [18]:
```
# GLM mit einem Schwellenwert von 0.1
GLM_01 <- val %>% select(Claims, fitGLM1) %>% mutate( fitGLM_01_binary = ifelse(fitGLM1 > 0.1, 1, 0))
getMeasures(GLM_01$Claims, GLM_01$fitGLM_01_binary)
```

```
      predictedClaims
Claims    0    1
     0 4415 1002
     1  265  161
[1] "Accuracy:    0.783"
[1] "Precision:   0.138"
[1] "Recall/TPR:  0.378"
[1] "FPR:         0.185"
[1] "Specificity: 0.815"
```

Bei höheren als den betrachteten Schwellenwerten werden gar keine Schäden mehr erkannt. Aufgrund der Seltenheit des Schadenereignisses hat ein solches Modell jedoch noch immer eine Accuracy von über 90 Prozent, da ein großer Anteil der Datensätze ohne Schaden korrekt als Nicht-Schaden klassifiziert wird. Das simple Modell Schaden = 0 hätte eine Accuracy von 5417 / (5417 + 426) = 92,71%, das entspricht dem Anteil der Nicht-Schadenfälle in den Validierungsdaten (siehe oben).

Bei einem Schwellenwert von 0.3 werden vier der 426 Datensätze mit Schäden als solche erkannt. Es gibt 6 falsch erkannte Schadenfälle. Dennoch hat das Modell eine recht hohe Accuracy, da 5.411 der 5.417 Nicht-Schäden korrekt erkannt werden.

Bei einem Schwellenwert von 0.2 werden bereits 33 der Schäden erkannt, bei einem Schwellenwert von 0.1 werden 161 der 426 Schäden korrekt klassifiziert - dafür werden jedoch auch 1.002 Nicht-Schadenfälle als Schäden eingestuft.

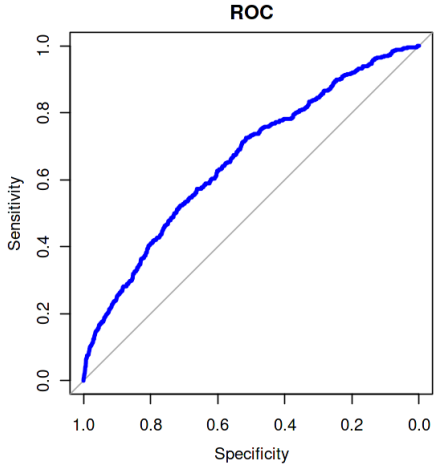
Aufgabenteil C3: c) [Lernziel 6.1] [4 Punkte]

Erstellen Sie eine ROC-Curve für das GLM. Wie finden sich hier die Ergebnisse von Teil C3: b) wieder?

Lösungsvorschlag:

In [19]:
```
options(repr.plot.width=5, repr.plot.height=5)

plot(roc(val$Claims, val$fitGLM1, direction="<"), col="blue", lwd=3, main="ROC")
```

```
Setting levels: control = 0, case = 1
```



Hier kann man nun die Werte von oben abtragen, die Specificity auf der X-Achse und Recall/TPR bzw. Sensitivity auf der Y-Achse:

Schwellenwert von 0.3: "Specificity: 0.999" "Recall/TPR: 0.009"

Schwellenwert von 0.2: "Specificity: 0.987" "Recall/TPR: 0.077"

Schwellenwert von 0.1: "Specificity: 0.815" "Recall/TPR: 0.378"

Aufgabenteil C3: d) [Lernziel 6.1] [4 Punkte]

Nehmen Sie folgende Randbedingungen für Ihr Modell an:

Ein falsch positiv klassifizierter Schaden kostet das Unternehmen 100 EUR (z.B. entgangene Prämie). Ein falsch negativ klassifizierter Schaden kostet das Unternehmen 1.000 EUR (z.B. durchschnittlicher Schaden).

Errechnen Sie mit diesen Annahmen den besten Schwellenwert für das Modell.

Wie ändert sich der Schwellenwert, wenn die Werte umgekehrt wären? Der falsch positiv klassifizierte Schaden also 1.000 EUR kosten würde und der falsch negative 100 EUR.

Lösungsvorschlag:

In [20]:
```
CostInfo <- function(val, Claims, predict, cost_fp, cost_fn )
{
    Cost_df <- data.frame(matrix(ncol = 4, nrow = 0))
    x <- c("i", "FP", "FN", "Cost")
    colnames(Cost_df) <- x

    for(i in seq(from=0.0, to=1, by=0.001)){
        data <- val %>% select(Claims, fitGLM1) %>% mutate( predictedClaims = ifelse(fitGLM1 > i, 1, 0))
        CM = CM = table(data$Claims, data$predictedClaims)

        if (length(CM)==4) # sonst stürzt es hier ab weil die Tabelle nicht gross genug ist
            #(sprich: eine Variante kommt nicht vor)
            {
                FP = CM[1,2]
                FN = CM[2,1]
                Cost = FP*cost_fp + FN*cost_fn
                new_row <- c(i, FP, FN, Cost)
                Cost_df[nrow(Cost_df) + 1, ] <- new_row
            }
    }
    return (Cost_df)
}
```

In [21]:
```
cost_fp <- 100
cost_fn <- 1000
Cost_df = CostInfo(val, val$Claims, val$fitGLM1, cost_fp, cost_fn)
```

In [22]:
```
Cost_df[which.min(Cost_df$Cost),]
```

A data.frame: 1 × 4

	i	FP	FN	Cost
	<dbl>	<dbl>	<dbl>	<dbl>
85	0.098	1053	254	359300

Bei einem Schwellenwert von 0.098 werden 254 Schäden falsch negativ prognostiziert und 1.053 Schäden falsch positiv.

Dem Unternehmen entstehen also Kosten von 254 *1.000 EUR* + *1.053* 100 EUR = 359.300 EUR

In [23]:
```
cost_fp <- 1000
cost_fn <- 100
Cost_df = CostInfo(val, val$Claims, val$fitGLM1, cost_fp, cost_fn)
```

In [24]:
```
Cost_df[which.min(Cost_df$Cost),]
```

A data.frame: 1 × 4

	i	FP	FN	Cost
	<dbl>	<dbl>	<dbl>	<dbl>
305	0.318	1	424	43400

Bei einem Schwellenwert von 0.318 werden 424 Schäden falsch negativ prognostiziert und nur 1 Schaden Falsch positiv.

Dem Unternehmen entstehen also Kosten von 424 *100 EUR* + *1* 1.000 EUR = 43.400 EUR

Aufgabe C4: Random Forest, XGB sowie abschließender Modellvergleich [16 Punkte]

Aufgabenteil C4: a) [Lernziel 4.1] [2 Punkte]

Analog zu Aufgabe C4 a) ist für die Vorhersage der binären Variablen Claims ein Random Forest zu verwenden.

Lösungsvorschlag:

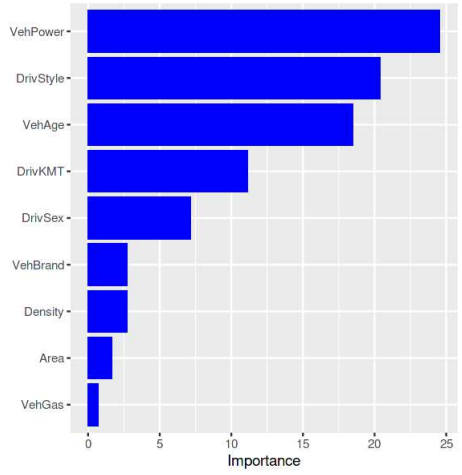
In [25]:
```
RF1 <- ranger(Claims ~.-ClaimNb, data=train,
              probability = TRUE, classification = TRUE,
              max.depth = 4, importance='impurity')
print(RF1)
```

```
Warning message in terms.formula(f, data = data):
“'varlist' has changed (from nvar=10) to new 11 after EncodeVars() -- should no longer happen!”
Ranger result

Call:
 ranger(Claims ~ . - ClaimNb, data = train, probability = TRUE,      classification = TRUE, max.depth = 4,
importance = "impurity")

Type:                             Probability estimation
Number of trees:                  500
Sample size:                      23373
Number of independent variables:  9
Mtry:                             3
Target node size:                 10
Variable importance mode:         impurity
Splitrule:                        gini
OOB prediction error (Brier s.):  0.06491931
```

Aufgabenteil C4: b) [Lernziel 4.1] [2 Punkte]

Nutzen Sie Ergebnisse aus Teil a) um eine Vorhersage auf den Validierungsdaten vorzunehmen.

In [26]:
```
# Vorhersagen erstellen
val$fitRF1 <- predict(RF1, data=val)$predictions[,2]

plot_vi_rf <- vip(RF1, width = 0.5, aesthetics = list(fill = "blue"))

options(repr.plot.width=5, repr.plot.height=5)
grid.arrange(plot_vi_rf, ncol = 1)
```



Aufgabenteil C4: c) [Lernziel 4.1] [2 Punkte]

Analog zu Aufgabe C4 a) ist für die Vorhersage der binären Variablen Claims XGBoost zu nutzen.

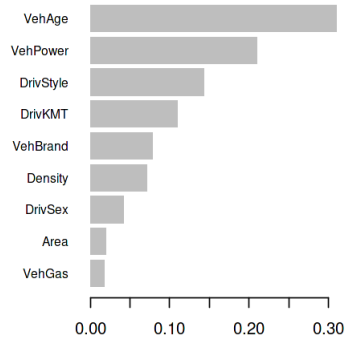
In [27]:
```
train_x <- data.matrix(select(train_orig, -Claims))
train_y <- data.matrix(select(train_orig, Claims))

val_x <- data.matrix(select(val_orig, -Claims))
val_y <- data.matrix(select(val_orig, Claims))

xgb_train <- xgb.DMatrix(data = train_x, label = train_y)
xgb_val <- xgb.DMatrix(data = val_x, label = val_y)
```

In [28]:
```
xgb <- xgb.train(data = xgb_train, nrounds = 200, max_depth = 3, eta = 0.1, verbose = 0,
                 objective = "binary:logistic", eval_metric="auc", tree_method = "hist")
```

Aufgabe C4: d) [Lernziel 4.1] [2 Punkte]

Nutzen Sie Ergebnisse aus Teil c) um eine Vorhersage auf den Validierungsdaten vorzunehmen. Geben Sie die Feature-Importance der Merkmale an.

In [29]:
```
options(repr.plot.width=5, repr.plot.height=5)

pred_y <- predict(xgb, xgb_val)
importance_matrix <- xgb.importance(colnames(xgb_train), model = xgb)
xgb.plot.importance(importance_matrix[1:9,])
val$fitxgb <- pred_y
```

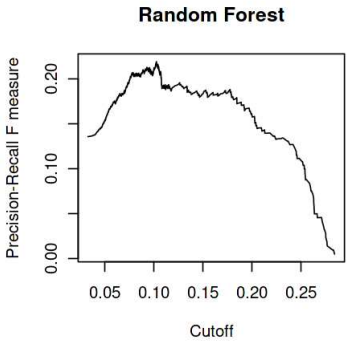


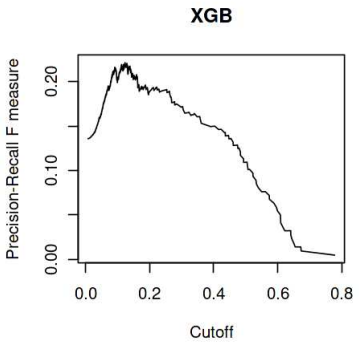
Aufgabe C4: e) [Lernziel 6.1] [3 Punkte]

Ermitteln Sie für den Random Forest und XGB einen optimalen CutOff-Point mittels des F1-Scores. Dabei ist auf eine angemessene Visualisierung zu achten.

Lösungsvorschlag:

In [30]:
```
options(repr.plot.width=4, repr.plot.height=4)

pred_rf <- prediction(data.frame(val$fitRF1), data.frame(val$Claims))
pred_xgb <- prediction(data.frame(val$fitxgb), data.frame(val$Claims))

perf_rf = performance(pred_rf, measure = "f")
perf_xgb = performance(pred_xgb, measure = "f")

plot(perf_rf, main="Random Forest")
plot(perf_xgb, main="XGB")
par(mfrow=c(1,3))
```





In [31]:
```
ind_rf <- which.max( slot(perf_rf, "y.values")[[1]] )
f1_rf <- slot(perf_rf, "y.values")[[1]][ind_rf]
cutoff_rf <- slot(perf_rf, "x.values")[[1]][ind_rf]
print(c(F1rf= f1_rf, cutoff = cutoff_rf))

ind_xgb <- which.max( slot(perf_xgb, "y.values")[[1]] )
f1_xgb <- slot(perf_xgb, "y.values")[[1]][ind_xgb]
cutoff_xgb <- slot(perf_xgb, "x.values")[[1]][ind_xgb]
print(c(F1xgb= f1_xgb, cutoff = cutoff_xgb))
```

```
     F1rf    cutoff
0.2196007 0.1028068
    F1xgb    cutoff
0.2215447 0.1230182
```

Damit ergeben sich die folgenden CutOff-Punkte:

Methode	CutOff Punkt
Random Forest	0.103
XGB	0.123

Aufgabe C4: f) [Lernziel 6.1] [3 Punkte]

Die Ergebnisse aus Teil e) sind aus Sicht des verwendeten Maßes sowie aus ökonomischer Sicht mit den Annahmen aus Aufgabenteil C3: d) zu bewerten.

Lösungsvorschlag:

In [32]:
```
RF_01 <- val %>% select(Claims, fitRF1) %>% mutate( fitRF_01_binary = ifelse(fitRF1 > 0.103, 1, 0))
XGB_01 <- val %>% select(Claims, fitxgb) %>% mutate( fitXGB_01_binary = ifelse(fitxgb > 0.123, 1, 0))
head(RF_01)
head(XGB_01)
```

A tibble: 6 × 3

Claims	fitRF1	fitRF_01_binary
<dbl>	<dbl>	<dbl>
1	0.09319775	0
1	0.26195566	1
1	0.26263440	1
1	0.07834537	0
1	0.24494369	1
1	0.06638190	0

A tibble: 6 × 3

Claims	fitxgb	fitXGB_01_binary
<dbl>	<dbl>	<dbl>
1	0.14040934	1
1	0.45356146	1
1	0.49336302	1
1	0.09160405	0
1	0.64085287	1
1	0.07933959	0

In [33]:
```
# Ergebnisse für RF
getMeasures(RF_01$Claims, RF_01$fitRF_01_binary)
```

```
      predictedClaims
Claims    0    1
     0 4868  549
     1  307  119
[1] "Accuracy:    0.853"
[1] "Precision:   0.178"
[1] "Recall/TPR:  0.279"
[1] "FPR:         0.101"
[1] "Specificity: 0.899"
```

In [34]:
```
# Ergebnisse für XGB
getMeasures(XGB_01$Claims, XGB_01$fitXGB_01_binary)
```

```
      predictedClaims
Claims    0    1
     0 4968  449
     1  317  109
[1] "Accuracy:    0.869"
[1] "Precision:   0.195"
[1] "Recall/TPR:  0.256"
[1] "FPR:         0.083"
[1] "Specificity: 0.917"
```

Für die Gütemessung und die Bestimmung des CutOff-Punktes wurde der F1-Score verwendet. Dabei wird ausgewogen zwischen FN und FP-Werten bewertet. Daher liefert der folgende Vergleich bei der ökomischen Bewertung auch keine Überraschung:

Kosten für RF: $549 \cdot 100 + 307 \cdot 1.000 = 361.900$

Kosten für XGB: $449 \cdot 100 + 317 \cdot 1.000 = 361.900$

Beide Kosten liegen oberhalb der in Teil C3: d) ermittelten Werte.

Die Vorgabe des F1-Scores sollte kritisch hinterfragt werden. Nach den gemachten Annahmen aus Teil C3: d) liegen die Kosten für FN bzw. FP weit auseinander (FN = 100; FP = 1000). Daher sollte mehr Wert auf den Recall gelegt werden, was für ein $\beta > 1$ spricht.

Aufgabe C4: g) Gütevergleich mittels ROC-Kurven [Lernziel 6.1] [2 Punkte]

Stellen Sie die Ergebnisse aus den Teilen C3 und C4 als ROC-Kurven gegenüber und interpretieren Sie das Ergebnis.

Lösungsvorschlag:

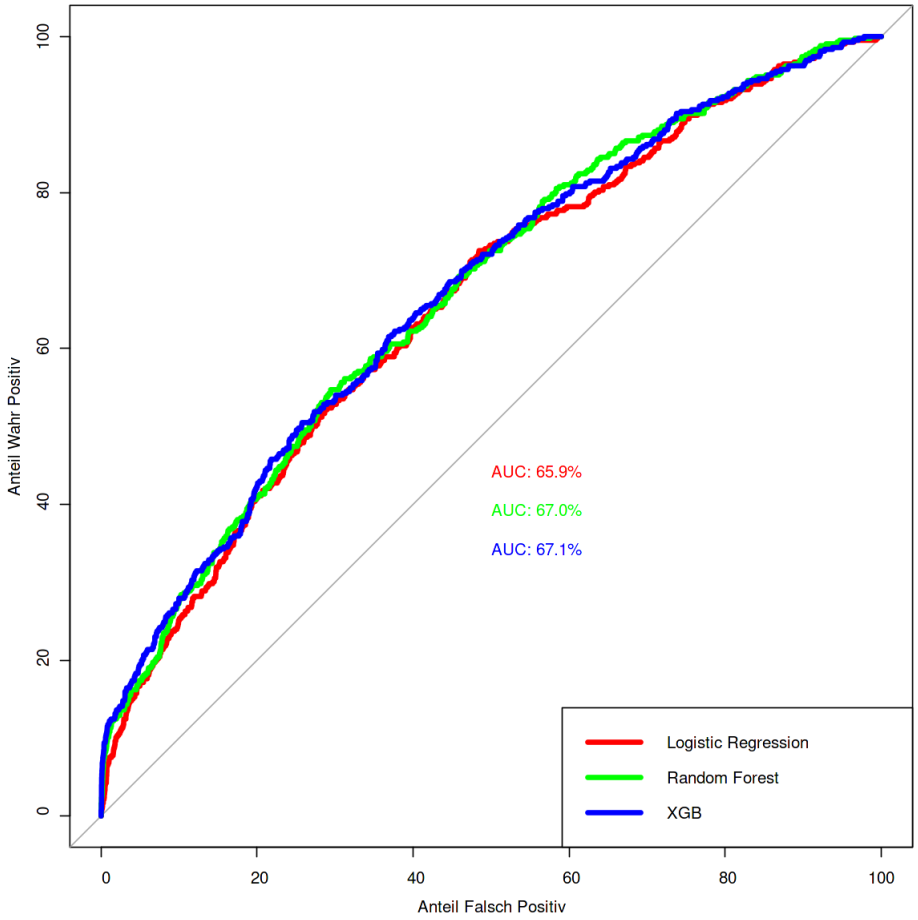
In [35]:
```
options(repr.plot.width=10, repr.plot.height=10)

# ROC-Kurve auf Basis der Validierungsdaten
par(pty = "s")
roc(val$Claims
    , val$fitGLM1
    , plot=TRUE
    , legacy.axes=TRUE
    , percent=TRUE
    , xlab="Anteil Falsch Positiv"
    , ylab="Anteil Wahr Positiv"
    , col="red"
    , lwd=4
```

```
    , print.auc=TRUE
    , print.auc.y=45)
plot.roc(val$Claims
         , val$fitRF1
         , percent=TRUE
         , col="green"
         , lwd=4
         , print.auc=TRUE
         , add=TRUE
         , print.auc.y=40)
plot.roc(val$Claims
         , val$fitxgb
         , percent=TRUE
         , col="blue"
         , lwd=4
         , print.auc=TRUE
         , add=TRUE
         , print.auc.y=35)

legend("bottomright"
       , legend=c("Logistic Regression", "Random Forest", "XGB")
       , col=c("red", "green", "blue")
       , lwd=4)
```

```
Setting levels: control = 0, case = 1

Setting direction: controls < cases
```

```
Call:
roc.default(response = val$Claims, predictor = val$fitGLM1, percent = TRUE,     plot = TRUE, legacy.axes =
TRUE, xlab = "Anteil Falsch Positiv",     ylab = "Anteil Wahr Positiv", col = "red", lwd = 4, print.auc = T
RUE,     print.auc.y = 45)

Data: val$fitGLM1 in 5417 controls (val$Claims 0) < 426 cases (val$Claims 1).
Area under the curve: 65.87%
```

```
Setting levels: control = 0, case = 1

Setting direction: controls < cases

Setting levels: control = 0, case = 1

Setting direction: controls < cases
```



Die AUC-Werte für die drei Modelle zeigen, dass Random Forest und XGB nahe beieinander liegen. Würde man sich rein auf die Maximierung des AUC-Wertes für die Modellauswahl beschränken, würde hier die Wahl auf XGB fallen. Damit ist die Frage nach einem geeigneten Cutoff-Punkt (oder, je nach Anwendungsfall, Cutoff-Punkten) noch nicht beantwortet.

Aufgabe C5: Gain- und Lift-Charts [6 Punkte]

Aufgabe C5: a) Gain-Chart [Lernziel 6.1] [3 Punkte]

Der Versicherer hat ein externes Angebot für ein Fahrsicherheitstraining bekommen. Bei Durchführung garantiert der Veranstalter zukünftig Schadenfreiheit. Man will die Schäden um 30% reduzieren - wie vielen Versicherungsnehmern muss der Versicherer das Training anbieten, um dieses Ziel zu erreichen? Welches Modell ist hier am besten geeignet? Gilt dies für alle gewählten Prozentsätze?

Die Frage soll mit einem Gain-Chart beantwortet werden.

Anmerkung: die Begriffe "Gain-Chart / -Curve" und "Lift-Chart / -Curve" werden in der Literatur und in den zur Verfügung stehenden Bibliotheken nicht eindeutig verwendet. So wird im Anschluss für den Gain Chart die caret-Funktion "Lift" verwendet und für den Lift Chart die Bibliothet "Gains".

Gute und kompakte Erklärung https://www.saedsayad.com/model_evaluation_c.htm

Lösungsvorschlag:

In [36]:
```
# Ergebnisse zusammenfassen in DataFrame fuer Lift
results      <- data.frame(Claim = val$Claims)
results$GLM  <- val$fitGLM1
results$RF   <- val$fitRF1
results$xgb  <- val$fitxgb
# results$rand <- val$fitRand

# Unterteilung in Klassen: Claim und NoClaim
results$Class[results$Claim==1] <- 'Claim'
results$Class[results$Claim==0] <- 'NoClaim'
results$Class <- as.factor(results$Class)
```

In [37]:
```
# Ergebnisse anschauen
head(results)
```

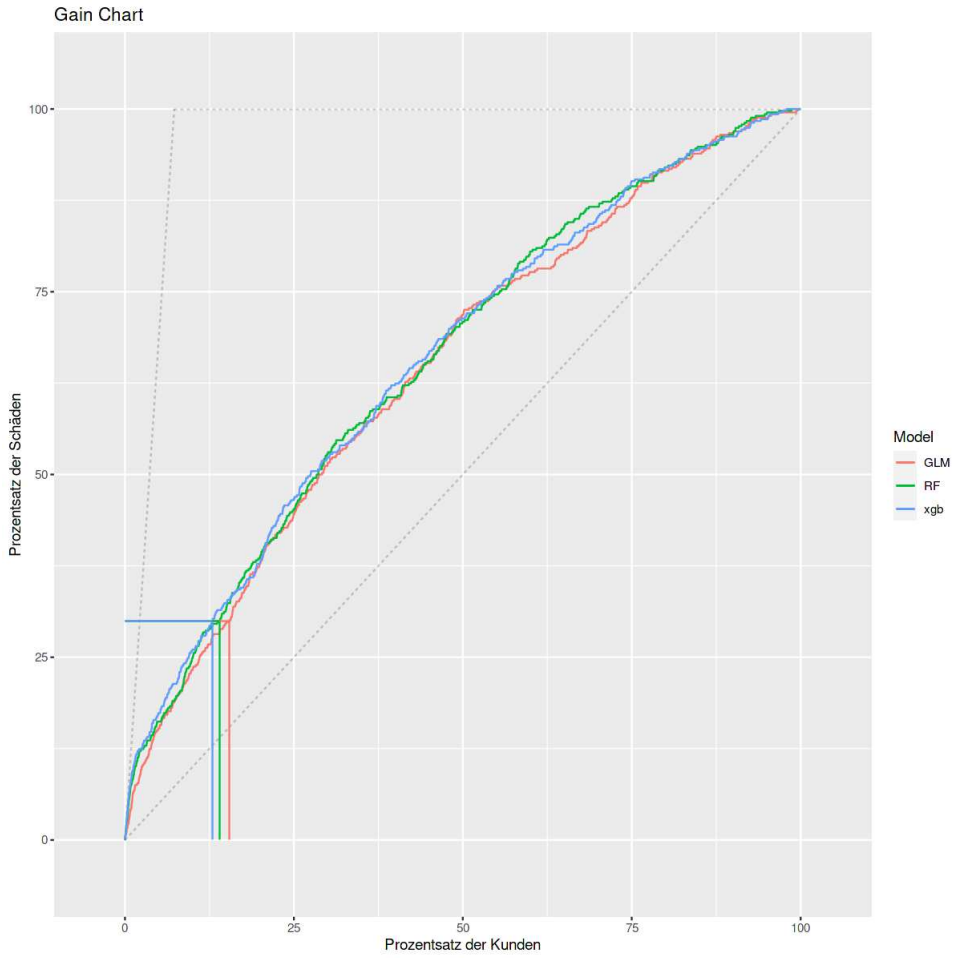
A data.frame: 6 × 5

	Claim	GLM	RF	xgb	Class
	<dbl>	<dbl>	<dbl>	<dbl>	<fct>
1	1	0.06530787	0.09319775	0.14040934	Claim
2	1	0.32056676	0.26195566	0.45356146	Claim
3	1	0.24467445	0.26263440	0.49336302	Claim
4	1	0.10082029	0.07834537	0.09160405	Claim
5	1	0.18624737	0.24494369	0.64085287	Claim
6	1	0.06659740	0.06638190	0.07933959	Claim

In [38]:
```
obj <- lift(Class ~ GLM + RF + xgb, data = results)
```

In [39]:
```
options(repr.plot.width=10, repr.plot.height=10)

plot <- ggplot(obj, values = 30)  +
    ggtitle('Gain Chart') + xlab('Prozentsatz der Kunden') + ylab('Prozentsatz der Schäden')
plot
```

```
Warning message in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
“collapsing to unique 'x' values”
Warning message in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
“collapsing to unique 'x' values”
Warning message in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
“collapsing to unique 'x' values”
```



Wenn man ca. 12,5 Prozent der Kunden anspricht, kann man zukünftig 30% der erwarteten Schäden einsparen. Bei 30% der Schäden erzielt man mit dem xgb die besten Ergebnisse - bei ca. 80% der Schäden würde RF bessere Ergebnisse liefern. Dies zeigt auch, dass eine ausschließliche Betrachtung des AUC-Wertes nicht immer zu idealen Ergebnissen führt.

Aufgabe C5: b) Decile-wise Lift-Chart [Lernziel 6.1] [3 Punkte]

Es soll die Frage beantwortet werden, um welchen Faktor die positiven Auswirkungen des Fahrtsicherheitstrainings (Schadenreduzierung) bei Anwendung des Modells erhöht werden kann, wenn 10 % der Kunden angeschrieben werden. Die Frage ist mittels eines Lift-Charts zu beantworten. Welches Modell bietet sich an?

Lösungsvorschlag:

https://rpubs.com/cassandra/PredictiveModelValidation

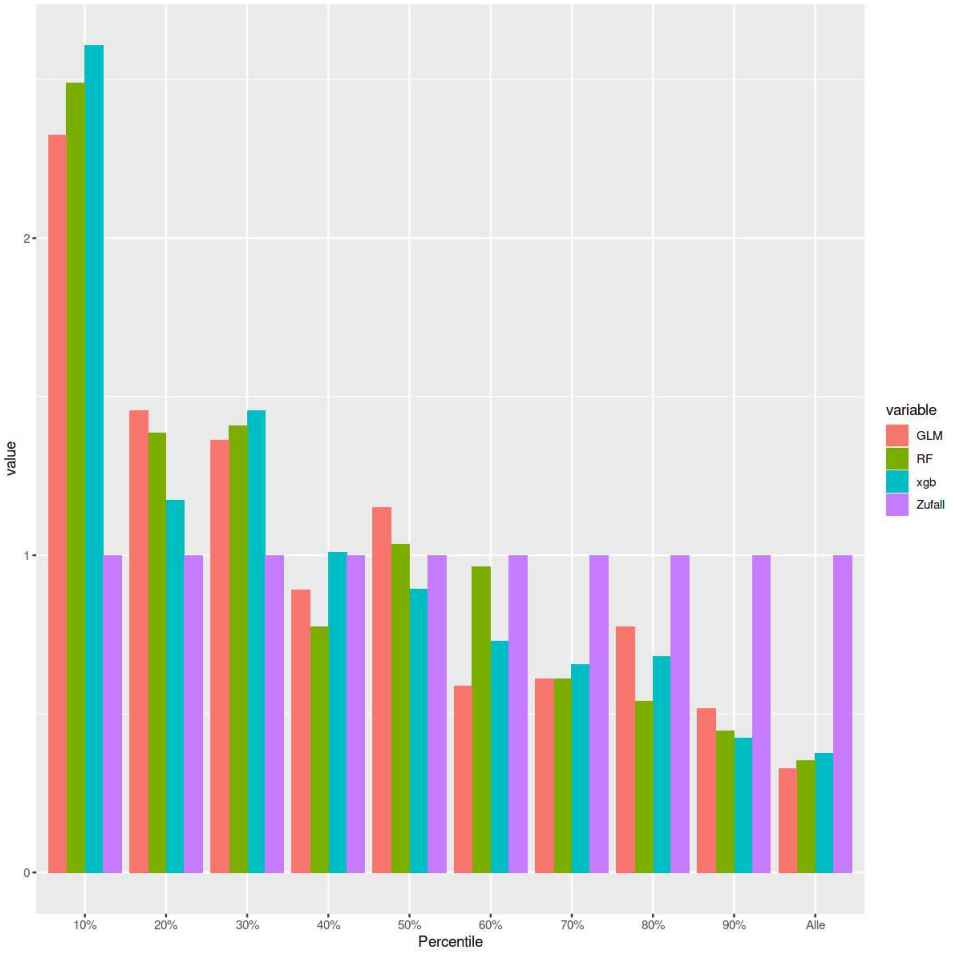
In [40]:
```
gain <- gains(results$Claim, results$xgb)
```

In [41]:
```
gain_xgb <- gains(results$Claim, results$xgb)
gain_RF  <- gains(results$Claim, results$RF)
gain_GLM <- gains(results$Claim, results$GLM)
```

In [42]:
```
xgb = gain_xgb$mean.resp / mean(results$Claim)
RF  = gain_RF$mean.resp / mean(results$Claim)
GLM = gain_GLM$mean.resp / mean(results$Claim)
Zufall = mean(results$Claim) / mean(results$Claim) # kann man auch weglassen

data <- data.frame(xgb, RF, GLM, Zufall)
```

In [43]:
```
require(tidyr)
Percentile <- c("10%", "20%", "30%", "40%", "50%", "60%", "70%", "80%", "90%", "Alle",
      "10%", "20%", "30%", "40%", "50%", "60%", "70%", "80%", "90%", "Alle" ,
      "10%", "20%", "30%", "40%", "50%", "60%", "70%", "80%", "90%", "Alle" ,
      "10%", "20%", "30%", "40%", "50%", "60%", "70%", "80%", "90%", "Alle")
data.long <- gather(data, variable, value)
```

In [44]:
```
options(repr.plot.width = 10, repr.plot.height = 10)
ggplot(data = data.long, aes(x = Percentile, y = value, fill = variable)) +
  geom_col(position = position_dodge())
```



Mit dem Random Forest und dem xbg kann jeweils ein "Lift Faktor" von über 2,5 erreicht werden - d.h. man erreicht durch das gezielte Ansprechen von Kunden, die das Modell auswählt mehr als 2,5 mal so viele Schadensfälle und potenzielle Schadenreduzierungen, als wenn man die Auswahl der Kunden zufällig trifft.

2. Teil: Tarifierung (Schadenhäufigkeit) [50 Punkte]

Ziel ist hier die Erstellung eines auf Basis der Schadenhäufigkeiten ausdifferenzierten, rechtskonformen Tarifmodells.

[Interner Hinweis: Punkteabzug, wenn geschlechtsdifferenzierte Tarifmodelle erstellt werden]

Aufgabe C6: Schadenhäufigkeit betrachten und für die Modellierung vorbereiten [6 Punkte]

Aufgabe C6: a) Schadenhäufigkeit [Lernziel 6.1] [3 Punkte]

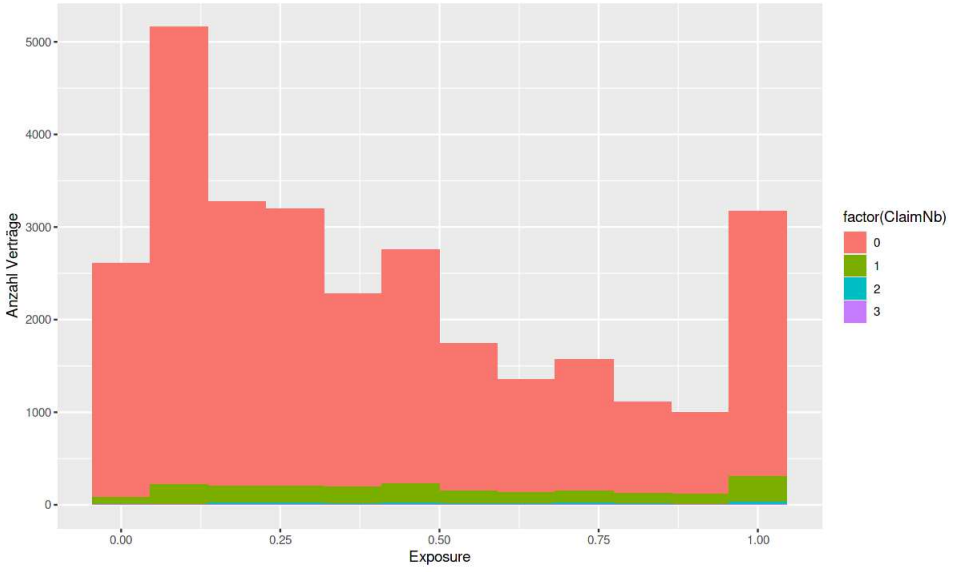
Im weiteren Aufgabenteil sollen nun die in Aufgabe C1 a) eingelesenen gesamten Daten (einschließlich ClaimNb und Exposure) betrachtet werden. Die ungewichtete Häufigkeitsverteilung der Schadenanzahl "ClaimNb", die Verteilung der Exposure sowie die exposure-gewichtete prozentuale Schadenhäufigkeit ist zu ermitteln und auszugeben. Um wie viel Prozent ist die Schadenhäufigkeit höher als die im 1. Aufgabenteil ermittelte Anteil an Policen mit mindestens einem Schaden? Was ist der Hauptgrund für die Abweichung?

Lösungsvorschlag:

In [45]:
```
# Verteilung der Schäden je Vertrag
table(dat$ClaimNb)
```

```
    0     1     2     3
27102  1942   159    13
```

In [46]:
```
# Darstellung, wie die absolute Häufigkeit der ClaimNB vom Merkmal "Exposure" abhängt
options(repr.plot.width=10, repr.plot.height=6)
ggplot(dat) +
    geom_histogram(mapping = aes(x = Exposure, fill = factor(ClaimNb)),binwidth = 1/11) +
    ylab("Anzahl Verträge")
```



In [47]:
```
# Berechnung der Schadenhäufigkeit und deren Abweichung zum Anteil nicht schadenfreier Policen:
SH=sum(dat$ClaimNb)/sum(dat$Exposure)
paste("Die Schadenhäufigkeit beträgt"
      , round(100*SH,2)
      ,"%")
paste("Die Schadenhäufigkeit ist um"
      ,round(100*SH/mean(dat$Claims)-100)
      ,"% höher als der im 1. Aufgabenteil ermittelte Anteil an Policen mit mindestens einem Schaden.")
paste("Die durchschnittliche Exposure beträgt p.a."
      , round(mean(dat$Exposure),2)
      , ", also rund"
      ,round(12*mean(dat$Exposure))
      ,"Monate.")
```

'Die Schadenhäufigkeit beträgt 19.22 %'

'Die Schadenhäufigkeit ist um 166 % höher als der im 1. Aufgabenteil ermittelte Anteil an Policen mit mindestens einem Schaden.'

'Die durchschnittliche Exposure beträgt p.a. 0.41 , also rund 5 Monate.'

Der Hauptgrund für die große Abweichung zwischen der Schadenhäufigkeit und dem Anteil nicht schadenfreier Policen liegt in der recht geringen jahresdurchschnittlichen Vertragslaufzeit von nur rund fünf Monaten.

Aufgabe C6: b) Modellierungsdateien, Funktionen und Nullmodell erstellen [Lernziel 6.1] [3 Punkte]

Die in Aufgabe C2 b) vorgenommene Aufteilung der Daten in zwei Stichproben (Exposure und ClaimNb wurden dort entfernt) ist auf die gesamten Daten anzuwenden. Die Trainings- und Teststichprobe sollen nur noch die benötigten Merkmale enthalten. Die Verteilung der ClaimNb sowie die Schadenhäufigkeit soll für beide Dateien verglichen und die Übereinstimmung bewertet werden. Darüber hinaus soll die Poissonabweichung des Nullmodells berechnet werden. Dazu soll eine entsprechende R- oder Python-Funktion definiert und angewandt werden.

Lösungsvorschlag:

Zur Merkmalsauswahl: Da geschlechtsdifferenzierte Tarife nicht zulässig sind, wird das Merkmal DrivSex aus den Trainingsdaten- und Testdaten entfernt. Die IDpol wird ebenfalls nicht benötigt und aus den Modellierungsdaten entfernt. Diese Vorgehensweise hat den Vorteil, dass bei den folgenden ML-Modellen nicht jeweils überprüft und ausgeschlossen werden muss, ob unzulässige Merkmale verwendet wurden.

In [48]:
```r
# train/test-Aufteilung (80%/20%) aus Aufgabe C2 a) verwenden
# und die train/test-Dateien aus Teil 1 überschreiben.
# Dabei IDpol und DrivSex nicht in die Modellierungsdaten übernehmen.
train <- dat[trn,c(2:10,12)]
test <- dat[-trn,c(2:10,12)]

# Sichtprüfung der Modellierungsdaten
head(train)
```

A tibble: 6 × 10

ClaimNb	Exposure	Area	VehPower	VehAge	VehBrand	VehGas	Density	DrivStyle	DrivKMT
<int>	<dbl>	<fct>	<fct>	<int>	<fct>	<fct>	<int>	<fct>	<fct>
0	0.52	E	1	1	B1	Diesel	5806	1	25
0	0.34	E	3	10	B2	Regular	4160	2	40
0	0.61	D	6	12	B1	Regular	563	4	25
0	0.24	D	1	18	B3	Diesel	563	4	9
0	0.32	C	3	5	B5	Regular	148	1	40
0	1.00	D	1	8	B1	Regular	1097	3	9

In [49]:
```r
# Auszählung der Häufigkeit: Prozentual (Stimmen Anteile überein?)
prop.table(table(train$ClaimNb))
prop.table(table(test$ClaimNb))
```

```
           0            1            2            3
0.9277799170 0.0662730501 0.0054336200 0.0005134129
          0           1           2           3
0.927092247 0.067259969 0.005476639 0.000171145
```

In [50]:
```r
# Funktion: ClaimFreqency erstellen und anwenden
ClaimFrequency <- function(txt, l.c, l.x, t.c, t.x) {
    sprintf("%s: %.2f%% / %.2f%%", txt, sum(l.c)/sum(l.x)*100, sum(t.c)/sum(t.x)*100)
}
# Schadenhäufigkeit der Trainings- und Teststichprobe vergleichen
ClaimFrequency("Schadenhäufigkeit der Trainings-/Teststichprobe"
               , train$ClaimNb,train$Exposure
               , test$ClaimNb,test$Exposure)
```

'Schadenhäufigkeit der Trainings-/Teststichprobe: 19.16% / 19.44%'

Die Verteilungen stimmen näherungsweise überein. Abweichungen können zufälligen Schwankungen bei kleinen Fallzahlen geschuldet sein.

In [51]:
```r
# Funktion: Poissonabweichung berechnen
PD <- function(pred, obs) {200*(sum(pred)-sum(obs)+sum(log((obs/pred)^(obs))))/length(pred) }

# Function: Poissonabweichung für train/test ausgeben
```

```
PD2 <- function(txt, l.c, l.x, t.c, t.x) {
    sprintf("%s, train / test: %.2f%% / %.2f%%", txt, PD(l.c, l.x), PD(t.c, t.x)) }
```

In [52]:
```
# Null-Modell für die Schadenhäufigkeit
sh <- sum(train$ClaimNb)/sum(train$Exposure)
# Print Poisson Deviance
PD2("Poissonabweichung Nullmodell", sh*train$Exposure,train$ClaimNb, sh*test$Exposure,test$ClaimNb)
```

'Poissonabweichung Nullmodell, train / test: 43.24% / 42.86%'

Aufgabe C7: Schadenhäufigkeitsmodellierung mit GLM und Entscheidungsbaum [9 Punkte]

Aufgabe C7: a) Poisson-Log-Modellierung via GLM [Lernziel 6.1] [3 Punkte]

Das GLM aus Aufgabe C3 a) soll zur Modellierung der Schadenhäufigkeit mittels eines Poisson-Log-Modells umgebaut werden. Die nötigen Änderungen sind zu beschreiben und durchzuführen. Das geänderte Modell ist an die Trainingsdaten zu fitten und die Güte des Modells ist in Form der Poissonabweichung an den Testdaten zu ermitteln.

Lösungsvorschlag:

Notwendige Anpassungen: In der Modellgleichung des GLM ist Claims durch ClaimNb zu ersetzen, die Verteilungsfamilie in Poisson zu ändern und ein Offset mit log(Exposure) anzugeben. Da typischerweise Jahresprämien berechnet werden, wird die Exposure aus der Modellgleichung entfernt.

In [53]:
```
# Umsetzung der Änderungen des GLM1
GLM1SH <- glm(ClaimNb ~.-Exposure, data=train, offset=log(Exposure),family=poisson())
summary(GLM1SH)
```

```
Call:
glm(formula = ClaimNb ~ . - Exposure, family = poisson(), data = train,
    offset = log(Exposure))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.3792  -0.4371  -0.3031  -0.1724   4.6801

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)   -2.475e+00  2.173e-01 -11.388  < 2e-16 ***
AreaB          2.921e-01  9.423e-02   3.100 0.001937 **
AreaC          9.205e-03  8.273e-02   0.111 0.911400
AreaD          2.321e-01  8.711e-02   2.664 0.007721 **
AreaE          1.758e-01  1.430e-01   1.229 0.218891
AreaF         -5.284e-02  3.432e-01  -0.154 0.877646
VehPower2      1.304e-01  7.902e-02   1.651 0.098816 .
VehPower3      2.230e-01  7.911e-02   2.819 0.004822 **
VehPower4      2.448e-01  8.149e-02   3.005 0.002660 **
VehPower5      8.414e-01  1.068e-01   7.879 3.29e-15 ***
VehPower6      8.400e-01  1.071e-01   7.842 4.43e-15 ***
VehPower7      1.094e+00  1.204e-01   9.090  < 2e-16 ***
VehPower8      1.078e+00  2.005e-01   5.377 7.56e-08 ***
VehPower9      1.327e+00  2.600e-01   5.106 3.29e-07 ***
VehPower10     4.891e-01  2.798e-01   1.748 0.080502 .
VehAge        -1.909e-02  5.259e-03  -3.631 0.000283 ***
VehBrandB10   -2.503e-01  2.649e-01  -0.945 0.344663
VehBrandB11    1.949e-01  1.458e-01   1.337 0.181341
VehBrandB12   -3.988e-02  9.210e-02  -0.433 0.664999
VehBrandB13   -6.903e-01  3.062e-01  -2.255 0.024148 *
VehBrandB14    4.572e-01  4.122e-01   1.109 0.267414
VehBrandB2     4.277e-02  6.560e-02   0.652 0.514387
VehBrandB3     8.273e-02  7.971e-02   1.038 0.299339
VehBrandB4    -2.104e-01  1.247e-01  -1.687 0.091569 .
VehBrandB5     1.594e-01  1.031e-01   1.546 0.122195
VehBrandB6     1.456e-01  1.094e-01   1.330 0.183447
VehGasRegular  1.562e-01  5.760e-02   2.712 0.006684 **
Density        4.458e-05  2.802e-05   1.591 0.111592
DrivStyle2     1.141e-01  6.759e-02   1.689 0.091249 .
DrivStyle3    -4.351e-01  7.765e-02  -5.603 2.11e-08 ***
DrivStyle4    -6.681e-01  8.069e-02  -8.280  < 2e-16 ***
DrivStyle5    -6.615e-01  8.961e-02  -7.382 1.56e-13 ***
DrivKMT6       4.473e-01  1.896e-01   2.359 0.018332 *
DrivKMT9       5.590e-01  1.887e-01   2.962 0.003058 **
DrivKMT12      6.480e-01  1.904e-01   3.403 0.000666 ***
DrivKMT15      7.767e-01  1.929e-01   4.027 5.64e-05 ***
DrivKMT20      7.977e-01  1.936e-01   4.120 3.78e-05 ***
DrivKMT25      8.335e-01  2.003e-01   4.160 3.18e-05 ***
DrivKMT30      1.032e+00  2.033e-01   5.073 3.92e-07 ***
DrivKMT35      8.947e-01  2.124e-01   4.211 2.54e-05 ***
DrivKMT40      1.050e+00  1.927e-01   5.449 5.07e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 10106.3  on 23372  degrees of freedom
Residual deviance:  9401.9  on 23332  degrees of freedom
AIC: 12950

Number of Fisher Scoring iterations: 6
```

In [54]:
```r
# Vorhersagen erstellen
train$fitGLM1SH <- fitted(GLM1SH)
test$fitGLM1SH <- predict(GLM1SH, newdata=test, type="response")

# Poissonabweichung ausgeben
PD2("Poissonabweichung GLM1SH", train$fitGLM1SH,train$ClaimNb, test$fitGLM1SH,test$ClaimNb)
```

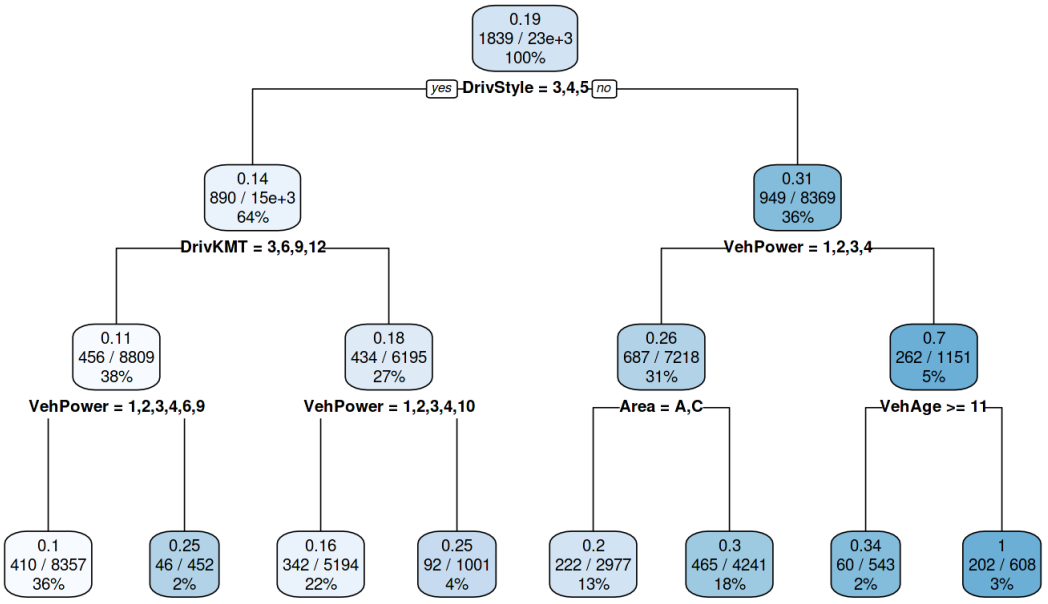
'Poissonabweichung GLM1SH, train / test: 40.23% / 39.33%'

Aufgabe C7: b) Poisson-Log-Modellierung via Entscheidungsbaum [Lernziel 4.1] [3 Punkte]

Es ist ein kleiner binärer Poisson-Entscheidungsbaum mit Tiefe 3 (also max. 8 Blätter) auf Basis der Trainingsdaten zu erstellen und zu visualisieren. Die Güte des Entscheidungsbaums ist in Form der Poissonabweichung an den Testdaten zu ermitteln.

Lösungsvorschlag:

In [55]:
```
# Entscheidungsbaum für die Schadenhäufigkeit berechnen
tree <- rpart(cbind(Exposure,ClaimNb) ~.-fitGLM1SH, train,
              method="poisson", control=rpart.control(maxdepth=3,cp=0.001))

# Entscheidungsbaum anzeigen
options(repr.plot.width=12, repr.plot.height = 7)
rpart.plot(tree) #
```



In [56]:
```
# Vorhersagen erstellen
train$fitTree1SH <- predict(tree,train)
train$fitTree1SH <- train$fitTree1SH*train$Exposure
test$fitTree1SH <- predict(tree, test)
test$fitTree1SH <- test$fitTree1SH*test$Exposure

# Poissonabweichung ausgeben
PD2("Poissonabweichung Tree1SH", train$fitTree1SH,train$ClaimNb, test$fitTree1SH,test$ClaimNb)
```

'Poissonabweichung Tree1SH, train / test: 40.44% / 39.63%'

Aufgabe C7: c) Ensemble und Vergleiche [Lernziel 4.1] [3 Punkte]

Die Vorhersagen des GLM und des Entscheidungsbaums sollen als Ensemble (Mittelwert) verwendet werden. Die Güte des Ensembles ist in Form der Poissonabweichung an den Testdaten zu ermitteln. Die Vor- und Nachteile der beiden verwendeten Basismodelle sind am vorliegenden Beispiel zu erläutern.

Lösungsvorschlag:

In [57]:
```
# Vorhersagen erstellen
train$fitMix1SH <- 0.5*(train$fitGLM1SH + train$fitTree1SH)
test$fitMix1SH <- 0.5*(test$fitGLM1SH + test$fitTree1SH)

# Poissonabweichung ausgeben
PD2("Poissonabweichung Ensemble GLM+Tree", train$fitMix1SH,train$ClaimNb, test$fitMix1SH,test$ClaimNb)
```

'Poissonabweichung Ensemble GLM+Tree, train / test: 40.12% / 39.25%'

Das GLM kann in der verwendeten einfachen Form (im Modellraum) nur lineare Zusammenhänge abbilden und daher möglichweise die volle Komplexität der Daten nicht abbilden sowie bei nicht signifikanten Parametern nicht testide Prognosen verursachen. Allerdings könnten hier durch die Zusammenlegung von Merkmalsklassen sowie durch Änderung von Wertebereichen, Polynomtermen und Interaktionstermen gute Verbesserungen erzielt werden und eine stabile Modellstruktur erreicht werden.

Der Entscheidungsbaum differenziert sehr gut zwischen den Gruppen und zeigt die wichtigsten Segmente. Auf dieser Basis könnten auch Anhaltspunkte für Merkmalsinteraktionen für das GLM abgeleitet werden. Allerdings kann der

Entscheidungsbaum bereits bei etwas anderen Daten (z.B. Datenaktualisierung oder andere Stichprobenziehung) anders aussehen.

Beide Modelle liefern vergleichbar gute Prognosen. In das Ensemble aus beiden Modelle fließen lineare und nicht lineare Zusammenhänge ein und führen zu einer Verbesserung der Prognose.

Aufgabe C8: Schadenhäufigkeitsmodellierung mit XGBoost samt Monotoniebeschränkungen [20 Punkte]

Aufgabe C8: a) Datenaufbereitung und Gradient Boosting mit XGBoost [Lernziel 4.1] [8 Punkte]

Das XGBoost-Modell aus Aufgabe C4 c) soll zur Modellierung der Schadenhäufigkeit mittels eines Poisson-Log-Modells umgebaut werden. Die nötigen Änderungen an den Datenstrukturen sind kurz zu kommentieren und durchzuführen. Das geänderte Modell ist an die Trainingsdaten zu fitten und die Güte des Modells ist in Form der Poissonabweichung an den Testdaten zu ermitteln. Des weiteren ist die Feature-Importance der Merkmale anzugeben, mit den entsprechenden Werten des Aufgabenteil C4 c) zu vergleichen und mögliche Unterschiede sind kurz zu begründen.

Lösungsvorschlag:

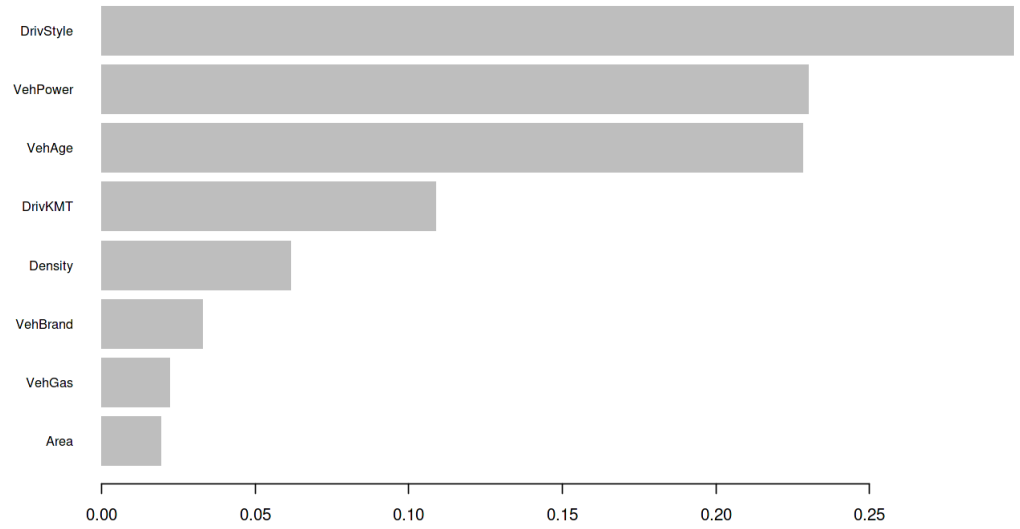
In [58]:
```
# train/test für ML: Features und label trennen
train_x <- data.matrix(train[,c(3:10)]) # nur die 8 Merkmale behalten
train_y <- data.matrix(train[,c(1)])
test_x <- data.matrix(test[,c(3:10)])
test_y <- data.matrix(test[,c(1)])

# Datenstrukturen für XGBoost anlegen
xtrain <- xgb.DMatrix(data = train_x, label = train_y)
xtest <- xgb.DMatrix(data = test_x, label = test_y)

# Exposure berücksichtigen (offset)
setinfo(xtrain,"base_margin",log(train$Exposure))
setinfo(xtest,"base_margin",log(test$Exposure))
```

TRUE
TRUE

In [59]:
```
## XGBoost aus C4 c) an SH anpassen:
# objective = "count:poisson" mit eval_metric="poisson-nloglik" (default)
XGB1 <- xgb.train(data = xtrain, nrounds = 200, max_depth = 3, eta = 0.1, verbose = 0,
                  objective = "count:poisson", tree_method = "hist")

# Vorhersagen erstellen
train$fitXGB1SH <- predict(XGB1 , newdata = xtrain)
test$fitXGB1SH <- predict(XGB1, newdata = xtest)

# Poissonabweichung ausgeben
PD2("Poissonabweichung XGB1SH", train$fitXGB1SH, train$ClaimNb, test$fitXGB1SH,test$ClaimNb)
```

'Poissonabweichung XGB1SH, train / test: 38.53% / 38.46%'

In [60]:
```
# Feature Importance ausgeben
importance_XGB1 <- xgb.importance(colnames(xtrain), model = XGB1)
xgb.plot.importance(importance_XGB1[1:8,])
```



Im Vergleich zur Feature Importance der entsprechenden Binärklassifikation haben VehAge und DrivStyle die vorderen Plätze im Ranking getauscht. DrivSyle ist nun mit Abstand das wichtigste Merkmal. Denkbar ist hier ein Zusammenhang mit der zuvor nicht betrachteten Exposure, deren Mittelwert (Exposure_mean) bei DrivStyle=2 auffällig gering ist:

In [61]:
```
train %>%
  group_by(DrivStyle) %>%
  summarise_at(vars(ClaimNb,Exposure), list(~sum(.), ~mean(.)))
```

A tibble: 5 × 5

DrivStyle	ClaimNb_sum	Exposure_sum	ClaimNb_mean	Exposure_mean
<fct>	<int>	<dbl>	<dbl>	<dbl>
1	391	1267.595	0.12998670	0.4214078
2	558	1768.436	0.10408506	0.3298706
3	319	1829.017	0.07062209	0.4049186
4	310	2437.488	0.05556551	0.4369041
5	261	2294.599	0.05317848	0.4675221

Aufgabe C8: b) XGBoost mit Monotoniebeschränkungen [Lernziel 4.1] [4 Punkte]

Das XGBooost-Modell aus dem vorangegangenen Aufgabenteil soll um zwei Monotoniebeschränkungen ergänzt werden. Die Schadenhäufigkeit soll beim Merkmal DrivKMT monoton steigen und beim Merkmal DrivStyle monoton fallen. Das geänderte Modell ist an die Trainingsdaten zu fitten und die Güte des Modells ist in Form der Poissonabweichung an den Testdaten zu ermitteln. Die Güte der beiden XGBoost-Modelle ist zu vergleichen und die beobachtete Veränderung zu interpretieren. War das so zu erwarten?

Lösungsvorschlag:

In [62]:
```
# Vektor mit Monotoniebeschränkungen erzeugen: steigend: +1, fallend: -1, unbeschränkt: 0
mb <- c(rep( 0, 8))
mb[7] <- -1 # Schätzer für Fahrstil (DrivStyle) soll monoton fallen
mb[8] <- +1 # Schätzer für Jahresfahrleistung (DrivKMT) soll monoton steigen
mb
```

0 · 0 · 0 · 0 · 0 · 0 · -1 · 1

In [63]:
```
## XGBoost aus C8 a) mit Monotoniebeschränkungen fitten
XGB2 <- xgb.train(data = xtrain, nrounds = 200, max_depth = 3, eta = 0.1, verbose = 0,
                  monotone_constraints = mb, objective = "count:poisson", tree_method = "hist")

# Vorhersagen erstellen
train$fitXGB2SH <- predict(XGB2 , newdata = xtrain)
test$fitXGB2SH <- predict(XGB2, newdata = xtest)
```

```
# Poissonabweichung Vorhersagen erstellen
PD2("Poissonabweichung XGB2SH (Monotoniebeschränkung)"
    , train$fitXGB2SH
    , train$ClaimNb
    , test$fitXGB2SH
    ,test$ClaimNb)
```

'Poissonabweichung XGB2SH (Monotoniebeschränkung), train / test: 38.64% / 38.47%'

Ergebnis: Die Monotoniebeschränkungen führen hier nur zu einer sehr kleinen Verschlechterung der Modellgüte und haben daher nur einen sehr geringen Einfluss. Tendenziell konnte man das bereits bei den Häufigkeitsverteilungen zur Aufgabe C1 c) erkennen (vor Betrachtung von Exposure, Mehrfachschäden und Interaktionen). DrivKMT zeigt dort ein monoton steigendes und DrivStyle ein monoton fallendes Verhalten für Claims=1. Daher wären große Auswirkungen auf die Modellgüte eine Überraschung gewesen, das Ergebnis ist entsprechend erwartbar.

Aufgabe C8: c) XGBoost mit Hyperparametertuning [Lernziel 4.1] [8 Punkte]

Die Hyperparameter Baumanzahl, Baumtiefe und Lernrate des XGBooost-Modell aus Aufgabe C8 b) sind geeignet zu optimieren. Beschreiben Sie kurz Ihr gewähltes Verfahren, setzten Sie es um, kommentieren Sie das Ergebnis und bewerten Sie, inwieweit das gefundene Model gut generalisiert.

Lösungsvorschlag:

Da kein Verfahren vorgegeben ist, wird im Folgenden für 20 zufällig gewählte (hier dreidimensionale) Hyperparameterkonstellationen ein XGBoost-Modell gefittet und die ermittelte Modellgüte samt Hyperparameter in eine Tabelle geschrieben. Das Modell mit der geringsten Test-Poissonabweichung wird ausgewählt. Vor- bzw. Nachteile des Verfahrens werden am Ende des Aufgabenteils diskutiert.

In [64]:
```
# Tabelle für Hyperparameter und Güte (PD) anlegen
xgb_results <- tibble(Nummer=integer(), Baeume=integer(), Tiefe=integer(), Lernrate=numeric(),
                      PD_train=numeric(), PD_test=numeric())
```

In [65]:
```
# "Randomized Search" für XGBoost-Hyperparameter: Baumanzahl, Baumtiefe und Lernrate
for (i in 1:20) {
    #  Hyperparameter: Wertebereich definieren und einen zufälligen Wert ziehen
    ntrees <- sample(2:20,1)*20
    tiefe <- sample(2:6,1)
    lernrate <- round(ifelse(ntrees <= 100, runif(1,0.06,0.30), runif(1,0.03,0.15)),2)

    # XGBoost-Modell trainieren
    xgb_search <- xgb.train(data=xtrain, nrounds=ntrees, max_depth=tiefe, eta=lernrate,
                            objective="count:poisson", eval_metric="poisson-nloglik",
                            monotone_constraints = mb, verbose=0, tree_method = "hist")

    # Gütemaß PD: Training, Test
    y_train_pred <- predict(xgb_search,newdata = xtrain)
    PD_train <- PD(y_train_pred,train$ClaimNb)
    y_test_pred <- predict(xgb_search,newdata = xtest)
    PD_test <- PD(y_test_pred,test$ClaimNb)

    # Hyperparameter und PD exportieren
    xgb_results <- xgb_results %>%
        add_row(Nummer=i, Baeume=all_of(ntrees), Tiefe=all_of(tiefe), Lernrate=all_of(lernrate),
                PD_train=PD_train, PD_test=PD_test)
 }
```

In [66]:
```
# Hyperparameter des bestes Modells auswählen
best <- which.min(xgb_results$PD_test)

# Hyperparameter und Modellgüte anzeigen (Bestes Modell ist ganz oben)
head(xgb_results[order(xgb_results$PD_test),])
```

A tibble: 6 × 6

Nummer	Baeume	Tiefe	Lernrate	PD_train	PD_test
<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>
6	220	5	0.06	37.18198	38.15196
19	300	5	0.05	36.92443	38.22664
10	220	4	0.09	37.57227	38.27441
14	100	6	0.16	35.15119	38.41563
18	220	3	0.12	38.30847	38.45058
9	360	3	0.10	38.03387	38.46124

In [67]:
```
XGB3 <- xgb.train(data = xtrain, nrounds = xgb_results$Baeume[best], max_depth = xgb_results$Tiefe[best],
                  eta = xgb_results$Lernrate[best], verbose = 0,
                  monotone_constraints = mb, objective = "count:poisson", tree_method = "hist")

# Vorhersagen erstellen
train$fitXGB3SH <- predict(XGB3 , newdata = xtrain)
test$fitXGB3SH <- predict(XGB3, newdata = xtest)

# Poissonabweichung ausgeben
PD2("Poissonabweichung XGB3SH", train$fitXGB3SH, train$ClaimNb, test$fitXGB3SH,test$ClaimNb)
```

'Poissonabweichung XGB3SH, train / test: 37.18% / 38.15%'

Ergebnis: Durch die Optimierung der Hyperparameter konnte die Poissonabweichung etwas verringert, die Güte des Modells also etwas verbessert werden. Da jedoch die Testdaten zur Optimierung der Hyperparameter verwendet wurde, kann nicht davon ausgegangen werden, dass das so gefundene Model gut generalisiert.

Eine bessere Vorgehensweise wäre die Trennung der Daten in drei Gruppen (Training, Validierung und Test) bzw. eine Kreuzvalidierung innerhalb der Trainingsdaten gewesen, diese war aber (aus Gründen der Vereinfachung) nicht vorgegeben.

Aufgabe C9: Schadenhäufigkeitsmodellierung mit dem "Combined Actuarial Neural Net" (CANN) [15 Punkte]

Als letztes Modell ist das "Combined Actuarial Neural Net" (CANN) in Orientierung an der Umsetzung von Jürg Schelldorfer und Mario V. Wüthrich (2019) mit dem Deep-Learning-Framework "Keras" zu erstellen, siehe https://github.com/JSchelldorfer/ActuarialDataScience, "3 - Nesting Classical Actuarial Models into Neural Networks".

Aufgabe C9: a) Datenaufbereitung für das neuronale Netz (CANN) [Lernziel 4.3] [4 Punkte]

Für das neuronale Netz sind die Daten geeignet aufzubereiten (das Merkmal VehBrand soll dabei über ein Embedding ins Netzwerk einfließen, siehe folgenden Aufgabenteil).

Lösungsvorschlag:

In [68]:
```
# feature pre-processing

# min-max-scaler:
PreProcess.Continuous <- function(var1, dat2){
  names(dat2)[names(dat2) == var1]  <- "V1"
  dat2$X <- as.numeric(dat2$V1)
  dat2$X <- 2*(dat2$X-min(dat2$X))/(max(dat2$X)-min(dat2$X))-1
  names(dat2)[names(dat2) == "V1"]  <- var1
  names(dat2)[names(dat2) == "X"]  <- paste(var1,"X", sep="")
  dat2
}

# pre-procecessing function:
Features.PreProcess <- function(dat2){
  dat2 <- PreProcess.Continuous("Area", dat2)
  dat2 <- PreProcess.Continuous("VehPower", dat2)
  dat2$VehAge <- pmin(dat2$VehAge,20)
  dat2 <- PreProcess.Continuous("VehAge", dat2)
  dat2$VehBrandX <- as.integer(dat2$VehBrand)-1
  dat2$VehGasX <- as.integer(dat2$VehGas)-1.5
  dat2$Density <- round(log(dat2$Density),2)
  dat2 <- PreProcess.Continuous("Density", dat2)
```

```
  dat2 <- PreProcess.Continuous("DrivStyle", dat2)
  dat2
}

dat2 <- Features.PreProcess(dat)

# Prognosewerte des GLM1 hinzufügen (für das CANN)
dat2$fitGLM1SH <- predict(GLM1SH, newdata=dat2, type="response")
```

In [69]:
```
# Transformationen an einer kleinen Zufallsstichprobe überprüfen
sample_n(dat2[c(2:3,14:21)], 5)
```

A tibble: 5 × 10

ClaimNb	Exposure	AreaX	VehPowerX	VehAgeX	VehBrandX	VehGasX	DensityX	DrivStyleX	fitGLM1SH
<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
0	0.53	-0.2	-0.7777778	0.1	9	-0.5	0.07091172	1.0	0.03938330
1	1.00	-1.0	-0.7777778	-0.2	0	0.5	-0.60636758	0.0	0.13549930
2	0.62	0.2	-0.5555556	0.0	7	-0.5	0.16642547	-0.5	0.18386167
0	1.00	-0.2	-0.5555556	0.0	6	-0.5	-0.02460203	0.5	0.10428437
0	0.52	-1.0	-0.3333333	-0.2	0	-0.5	-0.60636758	1.0	0.04333759

In [70]:
```
# Nicht benötigte bzw. nicht zulässige Merkmale entfernen
# und train/test-Stichproben wie oben (gleiche Liste) anlegen
dat2 <- dat2[c(2:3,14:21)]

# training und test sample
train9 <- dat2[trn,]
test9 <- dat2[-trn,]
```

In [71]:
```
# Merkmale definieren
features <- c(3:5, 7:9) # definition of feature variables (non-categorical)
q0 <- length(features)
print(paste("Anzahl Basisvariablen (ohne VehBrand):",q0))
```

```
[1] "Anzahl Basisvariablen (ohne VehBrand): 6"
```

In [72]:
```
# Matrizen der Trainingsdaten aufbereiten
Xtrain <- as.matrix(train9[, features])
Brtrain <- as.matrix(train9$VehBrandX)
Ytrain <- as.matrix(train9$ClaimNb)

# Matrizen der Testdaten aufbereiten
Xtest <- as.matrix(test9[, features])
Brtest <- as.matrix(test9$VehBrandX)
Ytest <- as.matrix(test9$ClaimNb)
```

Aufgabe C9: b) Netzwerkarchitektur (CANN) aufbauen und parametrisieren [Lernziel 4.3] [8 Punkte]

Es soll ein neuronales Netz mit drei verborgenen Schichten aufgebaut werden. Das Merkmal VehBrand soll über ein zweidimensionales Embedding im Netzwerk berücksichtigt werden. Als CANN-Komponente soll das Prognoseergebnis des GLM aus Aufgabe C7 a) einfließen, siehe "GLM Skip Connection" in der Referenz. Die Modellarchitektur ("Summary") ist auszugeben.

Das Modell soll geeignet gefittet, die Konvergenz der Anpassung überprüft und die Modellgüte in Form der Poissonabweichung an den Testdaten ermitteln werden. Sollte die Modellgüte des CANN schlechter als die des zugrundeliegenden GLM ausfallen sind die Ursachen aufzuklären und zu beheben.

Lösungsvorschlag:

In [73]:
```
# Das GLM ins CANN einbauen
Vtrain <- as.matrix(log(train9$fitGLM1SH))
Vtest <- as.matrix(log(test9$fitGLM1SH))
(lambda.hom <- sum(train9$ClaimNb)/sum(train9$fitGLM1SH))
```

1.06102749904331

In [74]:
```
# Hyperparameter der Netzwerkarchitektur (siehe "01 CANN approach.r")
q1 <- 16 # Anzahl Neuronen in hidden Layer 1
q2 <- 12
```

```
q3 <- 8
d <- 2   # Dimensionierung der Embeddings für kategoriale Merkmale
(BrLabel <- length(unique(train9$VehBrandX)))
```

11

In [75]:
```
# Architektur des CANN definieren
Design   <- layer_input(shape = c(q0),  dtype = 'float32', name = 'Design')
VehBrand <- layer_input(shape = c(1),   dtype = 'int32', name = 'VehBrand')
LogVol   <- layer_input(shape = c(1),   dtype = 'float32', name = 'LogVol')

BrandEmb = VehBrand %>%
  layer_embedding(input_dim = BrLabel, output_dim = d, input_length = 1, name = 'BrandEmb') %>%
  layer_flatten(name='Brand_flat')

Network = list(Design, BrandEmb) %>% layer_concatenate(name='concate') %>%
  layer_dense(units=q1, activation='tanh', name='hidden1') %>%
  layer_dense(units=q2, activation='tanh', name='hidden2') %>%
  layer_dense(units=q3, activation='tanh', name='hidden3') %>%
  layer_dense(units=1, activation='linear', name='Network',
              weights=list(array(0, dim=c(q3,1)), array(log(lambda.hom), dim=c(1))))

Response = list(Network, LogVol) %>% layer_add(name='Add') %>%
  layer_dense(units=1, activation=k_exp, name = 'Response', trainable=FALSE,
              weights=list(array(1, dim=c(1,1)), array(0, dim=c(1))))

model <- keras_model(inputs = c(Design, VehBrand,LogVol), outputs = c(Response))
model %>% compile(optimizer = optimizer_nadam(), loss = 'poisson')

summary(model)
```

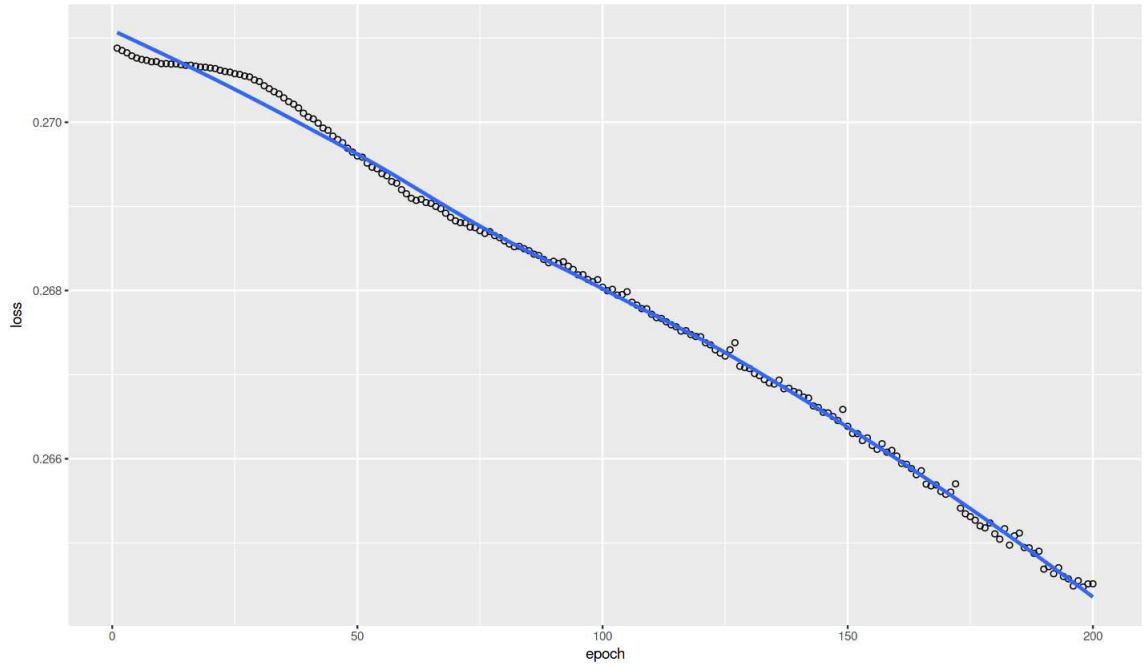
```
Model: "functional_1"
__________________________________________________________________________________
Layer (type)                Output Shape      Param #  Connected to
==================================================================================
VehBrand (InputLayer)       [(None, 1)]       0
__________________________________________________________________________________
BrandEmb (Embedding)        (None, 1, 2)      22       VehBrand[0][0]
__________________________________________________________________________________
Design (InputLayer)         [(None, 6)]       0
__________________________________________________________________________________
Brand_flat (Flatten)        (None, 2)         0        BrandEmb[0][0]
__________________________________________________________________________________
concate (Concatenate)       (None, 8)         0        Design[0][0]
                                                       Brand_flat[0][0]
__________________________________________________________________________________
hidden1 (Dense)             (None, 16)        144      concate[0][0]
__________________________________________________________________________________
hidden2 (Dense)             (None, 12)        204      hidden1[0][0]
__________________________________________________________________________________
hidden3 (Dense)             (None, 8)         104      hidden2[0][0]
__________________________________________________________________________________
Network (Dense)             (None, 1)         9        hidden3[0][0]
__________________________________________________________________________________
LogVol (InputLayer)         [(None, 1)]       0
__________________________________________________________________________________
Add (Add)                   (None, 1)         0        Network[0][0]
                                                       LogVol[0][0]
__________________________________________________________________________________
Response (Dense)            (None, 1)         2        Add[0][0]
==================================================================================
Total params: 485
Trainable params: 483
Non-trainable params: 2
__________________________________________________________________________________
```

In [76]:
```
# Das CANN fitten
{t1 <- proc.time()
  fit <- model %>% keras::fit(list(Xtrain, Brtrain, Vtrain), Ytrain, epochs=200,
                              batch_size=10000, verbose=0, validation_split=0)
  (proc.time()-t1)}
```

```
   user  system elapsed
  8.776   0.922   6.145
```

In [77]:
```
plot(fit)
```

```
`geom_smooth()` using formula 'y ~ x'
```



In [78]:
```
# Vorhersagen erstellen
train$fitCANN <- as.vector(model %>% predict(list(Xtrain, Brtrain, Vtrain)))
test$fitCANN <- as.vector(model %>% predict(list(Xtest, Brtest, Vtest)))

# Poissonabweichung ausgeben
PD2("Poissonabweichung CANN"
    , train$fitCANN, as.vector(unlist(train$ClaimNb))
    , test$fitCANN, as.vector(unlist(test$ClaimNb)))
```

'Poissonabweichung CANN, train / test: 38.98% / 38.96%'

Aufgabe C9: c) Modellvergleiche und abschließende Bewertung [Lernziel 6.1] [3 Punkte]

Abschließed sollen die Poissionabweichung (Testdaten) der Modelle aus den Aufgabenteilen C6b (Nullmodell), C7a (GLM), C9b (CANN) und C8b (XGBoost) verglichen und das Ergebnis interpretiert werden.

In [79]:
```
# Vergleich der Modellgüten auf Basis der Testdaten:
PD2("Poissonabweichung Nullmodell"
    , sh*train$Exposure, train$ClaimNb
    , sh*test$Exposure, test$ClaimNb)
PD2("Poissonabweichung GLM1"
    , train$fitGLM1SH, train$ClaimNb
    , test$fitGLM1SH, test$ClaimNb)
PD2("Poissonabweichung CANN"
    , train$fitCANN, as.vector(unlist(train$ClaimNb))
    , test$fitCANN, as.vector(unlist(test$ClaimNb)))
PD2("Poissonabweichung XGB1 "
    , train$fitXGB1SH, train$ClaimNb
    , test$fitXGB1SH, test$ClaimNb)
```

'Poissonabweichung Nullmodell, train / test: 43.24% / 42.86%'
'Poissonabweichung GLM1, train / test: 40.23% / 39.33%'
'Poissonabweichung CANN, train / test: 38.98% / 38.96%'
'Poissonabweichung XGB1 , train / test: 38.53% / 38.46%'

Interpretation, Fazit:

Insgesamt kann hier kein Modell die Poissonabweichung gegenüber dem Nullmodell (kein Modell, entspräche Einheitsprämie) um deutlich mehr als 10% reduzieren. Die Daten und Methoden erlauben also keine genaue Vorhersage von Schadenereignissen.

Trotzdem kann sich das Gradient-Boosting-Verfahren XGBoost bei der Vorhersagegüte etwas von den anderen Verfahren absetzen und ist - in allen drei gefitteten Varianten - das genaueste Modell.

Die Progosegüte des CANN ist in der hier verwendeten Parametrisierung nur leicht besser als die des zugrunde liegenden GLM. Bei beiden Modellen besteht Optimierungspotential, siehe auch Anmerkungen in Aufgabe C7 c) zum GLM.