BESCHREIBENDE DATENANALYSE

Joern Ploennigs · AI4SC

ERSTE SCHRITTE: KENNE DEINE DATEN

Als nächstes machen wir einige Beispiele für explorative Datenanalyse mit Pandas. Hierfür laden wir als Beispiel die Liste aller aktuellen Baustellen in Rostock an, die von der Stadt Rostock im Open Data Portal zur Verfügung gestellt wird. Öffentliche Daten wie diese zeigen reale Bauaktivitäten und die Analyse hilft z.B. bei der Planung, Verkehrsführung oder Risikobewertung bei parallelen Projekten. Solche Daten können später genutzt werden, um Modelle zur Vorhersage von Bauverzögerungen, Verkehrsbelastung oder Personalplanung zu trainieren.

Wir lesen die komprimierte baustdatendatei direkt in Pandas ein. Da diese durch Komma getrennt ist, müssen keinen Trennzeichen angeben, da das Komma der Standard-Trennzeichen ist.

```
import numpy as np # Import von NumPy
import pandas as pd # Import von Pandas

baust  = pd.read_csv("../data/Baustellen/baustellen_flat.csv")
```

Als ersten Schritt können wir die Größe des Datensatzes abfragen. Datenrahmen haben ein Attribut `.shape`, genau wie numpy-Arrays:

```
baust.shape
```

```
(204, 19)
```

Die Antwort sagt uns die Anzahl der Zeilen und Spalten an. Z. B. `(204, 19)` bedeutet 204 Zeilen mit je 19 Spalten. Das hilft abzuschätzen, wie umfangreich der Datensatz ist und ob er vollständig erscheint.

Als nächstes könnten wir einen schnellen Blick auf ein paar Zeilen der Daten werfen. Die Attribute `.head()` und `.tail()` sind nützlich, um sich die ersten und letzten Zeilen anzusehen, `.sample()` extrahiert ein paar zufällige Zeilen:

```
baust.head()
```

	latitude	longitude	uuid	kreis_name	kreis_schluessel	gemeindeverband_name	gemeindeverband_schluessel	gemeinde_name	gemeinde_schluessel	strasse_name	strasse_schluessel
0	54.089282	12.111994	063a229c-db25-45af-9fba-8d963b287910	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Budapester Str.	01640
1	54.068210	12.078264	07165e95-4e5f-429c-88b6-125d261cbcda	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Satower Str.	08180
2	54.174303	12.081928	077f9448-c35e-4074-8491-7d142045a8db	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Am Markt	00460
3	54.082628	12.126542	0c04e4b5-6e82-49b6-8b60-1c2cf9c1b1cf	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Klopstockstr.	06070
4	54.074173	12.123860	0c122d6b-a6bd-4c41-b25b-09fe10a23a2e	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Ziolkowskistr.	09920

```
baust.tail(3)
```

	latitude	longitude	uuid	kreis_name	kreis_schluessel	gemeindeverband_name	gemeindeverband_schluessel	gemeinde_name	gemeinde_schluessel	strasse_name	strasse_schluessel
201	54.080468	12.126524	fd2e99b4-9ac8-4dd2-94c2-b521b6d91455	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Südring	07400
202	54.151488	12.080890	fd32f630-333c-4752-9989-8c41a8eb5b30	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Taklerring	08880
203	54.086911	12.104815	fdc7ab16-67ac-4d33-ae98-b0bbd9703f18	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Dethardingstr.	04330

```
baust.sample(4)
```

	latitude	longitude	uuid	kreis_name	kreis_schluessel	gemeindeverband_name	gemeindeverband_schluessel	gemeinde_name	gemeinde_schluessel	strasse_name	strasse_schluessel
186	54.123774	12.054765	ecabda72-7bb0-4ffd-82e1-1de9c6c83db0	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Willi-Bredel-Str.	09820
109	54.084988	12.136501	9b735785-e2c2-4fca-9c0d-677df0ed5cac	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Humboldtstr.	03740
138	54.239682	12.224892	b7e6784c-e54d-40ff-8864-0b16a423a0e8	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Torfbrücke	09000
168	54.083904	12.114623	ddfdb537-e839-4dbc-a8f9-7fd1abcfcb7d	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Fahnenstr.	02300

Diese Funktionen zeigt uns zwar im ersten Eindruck viele Spalten, aber nicht immer alle, wenn der Bildschirmplatz nicht ausreicht. Deshalb ist es bei solchen Datensätzen mit vielen Spalten sinnvoll alle Spaltennamen in einem Datensatz aufzulisten.

```
baust.columns
```

```
Index(['latitude', 'longitude', 'uuid', 'kreis_name', 'kreis_schluessel',
       'gemeindeverband_name', 'gemeindeverband_schluessel', 'gemeinde_name',
       'gemeinde_schluessel', 'strasse_name', 'strasse_schluessel', 'sparte',
       'von', 'nach', 'baubeginn', 'bauende', 'verkehrsbeeintraechtigungen',
       'baumassnahme', 'dauer'],
      dtype='str')
```

Als nächstes überprüfen wir die Datentypen der Daten. Obwohl einige, z.B. `kreis_name` und `baumassnahme` aussehen wie ein String, könnten sie tatsächlich einen anderen Typ haben. Dies kann über das Attribut `.dtypes` abgefragt werden:

```
baust.dtypes
```

```
latitude                      float64
longitude                     float64
uuid                              str
kreis_name                        str
kreis_schluessel              float64
gemeindeverband_name              str
gemeindeverband_schluessel    float64
gemeinde_name                     str
gemeinde_schluessel           float64
strasse_name                      str
strasse_schluessel                str
sparte                            str
von                               str
nach                              str
baubeginn                         str
bauende                           str
verkehrsbeeintraechtigungen       str
baumassnahme                      str
dauer                         float64
dtype: object
```

Die Datentypen geben an, dass sowohl `kreis_name` und `baumassnahme` keine Strings sondern Objekte sind. Hier bedeutet es Zeichenfolgen, im Prinzip können es auch komplexere Objekte sein. Wenn wir zum Beispiel mal die Anzahl der Werte für `kreis_name` berechnen

```
baust.kreis_name.count()
```

```
np.int64(203)
```

so ist diese Zahl niedriger als die Anzahl der Zeilen die `baust.shape` zurückgeliefert hat. Das liegt an fehlenden Werten. Dazu mehr im Abschnitt Datenvorverarbeitung.

WAS SIND DIE WERTE?

Einer der ersten Schritte mit einem neuen Datensatz ist es, die Werte der relevanten Variablen zu überprüfen. Die Pandas-Bibliothek enthält viele Tools für die beschreibenden Datenanalyse. Bei den kategorischen Variablen möchten wir normalerweise die expliziten Werte sehen, bei den numerischen können wir die minimalen und maximalen Werte überprüfen.

Verschiedene diskrete Werte

Was sind die möglichen Werte für *Kreisname*? Ein schneller Blick auf die Daten zeigt, dass es einige Werte für die Spalte `sparte`, aber gibt es noch mehr? Das können wir mit der Methode `.unique()` abfragen:

```
baust.sparte.unique()
```

```
<ArrowStringArray>
[  'Fernwärmeleitung',         'Grünpflege',      'Wasserleitung',
   'Gebäudesanierung',            'Hochbau',         'Straßenbau',
          'Stromnetz',          'Kabelnetz',         'Gasleitung',
        'Telefonnetz',  'Lichtsignalanlage', 'Straßenbeleuchtung',
            'privat',           'Gleisbau',       'Kranarbeiten']
Length: 15, dtype: str
```

Das Ergebnis zeigt uns, welche unterschiedlichen Sparten in den Baustellendaten vorkommen, zum Beispiel `Hochbau`, `Straßenbau` oder `Gebäudesanierung`. Wir können auch ein technisches Detail erkennen, nämlich dass das Ergebnis als ein numpy-Array zurückgegeben wird.

Tip

Ein Wort über Methoden und Methodenverkettung. Lassen Sie uns den vorherigen Befehl, `baust.sparte.unique()`, überprüfen. Dieser enthält drei Komponenten, und der Prozess besteht darin, diese in einer sequenziellen Reihenfolge anzuwenden:

- `baust` ist der Datensatz. Das ist das, womit wir anfangen.
- `.sparte` ist ein *Attribut* von `baust`. Hier bedeutet es, das die Spalte `sparte` aus dem Datensatz extrahiert wird und als Serie zurückgegeben wird.
- Schließlich ist `.unique()` eine *Methode*, welche die Serie an Werten auf ihrer linken Seite analysiert (hier die Serie `sparte`), und diese analysiert um die eindeutigen Werten zu identifizieren.

Einige der Methoden können entweder auf den gesamten Datenrahmen oder auf einzelne Variablen angewendet werden. Wenn sie auf den gesamten Datenrahmen angewendet werden, gelten sie für jede einzelne Variable separat, sozusagen wie eine Schleife über einzelne Variablen, auf die sie angewendet werden.

Häufig ist es nicht nur interessant bei kategorischen Werten zu verstehen, welche eindeutigen Werte auftreten, sondern wie oft diese Auftreten. Hierfür nutzt man die Funktion

```
baust.sparte.value_counts()
```

```
sparte
Hochbau               40
Gebäudesanierung      35
Straßenbau            35
Wasserleitung         25
Fernwärmeleitung      18
Telefonnetz           16
Kabelnetz             11
Stromnetz              8
Grünpflege             4
Lichtsignalanlage      3
Straßenbeleuchtung     3
Gleisbau               2
Kranarbeiten           2
Gasleitung             1
privat                 1
Name: count, dtype: int64
```

Das gibt uns schon mal ein gutes Bild der Sparten. Aber wie sieht es mit der Spalte `baumassnahme` aus?

```
baust.baumassnahme.unique()
```

```
<ArrowStringArray>
[                                          'FW-Leitung i.a. der Hanseatic',
                                                          'Baumpflanzung',
                                             'Einbau Trinkwasserschieber',
                                                                     nan,
                               'Sanierung Mischwasser- und Trinkwasserleitung',
                                                       'Containerstellung',
                                          'Kennzeichnung Baustellenausfahrten',
 'Wartungs- und Instandsetzungsarbeiten innerhalb der konzessionierten Strecke',
                                    'Grünraumarbeiten, Sanierung Außenanlagen',
                                      'Ausschleifung - Muffung, StS Bussebart',
 ...
                                                'Sanierung Trinkwasserleitung',
                                                       'Grundhafte Erneuerung',
                                                          'Neubau Ampelanlage',
                                                       'Erneuerung TW-Leitung',
                                                  'Anschluss Personenbahnhof',
                                'Arbeiten Hausanschluss Trink- und Schmutzwasser',
                      'Errichtung einer Kabelbrücke für E-Versorgung Baustelle',
                                    'Anbindung Breitband Stadtentsorgung HRO',
```

Diese Liste gibt ein viel komplexeres Bild und listet eine Vielzahl an Typen von Baumaßnahmen. Wenn wir hier die Funktion `value_counts()` verwenden, so listet diese auch nicht alle Werte auf:

```
baust.baumassnahme.value_counts()
```

```
baumassnahme
Breitbandausbau                                                 12
Erneuerung Straßenbeleuchtung                                    3
Sanierung Trinkwasserleitung                                     3
Baumpflanzung                                                    2
Sanierungsarbeiten                                               2
                                                                ..
Arbeiten Hausanschluss Trink- und Schmutzwasser                  1
Errichtung einer Kabelbrücke für E-Versorgung Baustelle          1
Anbindung Breitband Stadtentsorgung HRO                          1
Baustelleneinrichtung / Belieferung für Neubau Goetheplatzbrücke 1
Baustraße für Neubau Hort Taklerring                             1
Name: count, Length: 140, dtype: int64
```

Bei Datensätzen mit hunderttausenden von Zeilen kann das sehr lange dauern und kein verwertbares Ergebnis liefern. Deshalb ist es meist sinnvoll erstmal zu prüfen wie viele eindeutige Werte existieren. Diese Anzahl können wir mit der Methode `.nunique` abfragen:

```
baust.baumassnahme.nunique()
```

```
140
```

Numerische Werte

Für numerische Variablen sollte es vermieden werden `unique()` weil diese meist sehr große Ergebnismengen zurück geben, die selten nützlich sind. Stattdessen ist interessante den Wertebereich der Variable zu verstehen (siehe “Normierung”). Dies gibt einen schnellen Überblick darüber, ob die Werte plausibel sind, z.B. ob negative Alterswerte auftreten oder ungewöhnlich große Alterswerte:

```
baust.longitude.min(), baust.longitude.max()
```

```
(np.float64(12.0120276853359), np.float64(12.2248919062465))
```

```
baust.latitude.min(), baust.latitude.max()
```

```
(np.float64(54.057580297901), np.float64(54.2396815337323))
```

Der Wertebereich für beide ist für Rostock durchaus plausibel und wir machen uns daher keine Sorgen über zu große oder zu kleine Werte. Siehe Abschnitt “Entfernen unerwünschter Variablen” unten, um fehlende Werte zu erkennen und zu analysieren.

Pandas bietet die Methode `describe()` an, die es erlaubt diese Statistiken für alle numerische Spalten in einem Aufruf zu extrahieren. Als Ergebnis erhalten wir einen neun Dataframe mit den Statistiken.

```
baust.describe()
```

	latitude	longitude	kreis_schluessel	gemeindeverband_schluessel	gemeinde_schluessel	dauer
count	204.000000	204.000000	203.0	203.0	2.030000e+02	204.000000
mean	54.104472	12.110911	13003.0	130030000.0	1.300300e+11	229.857230
std	0.031356	0.034811	0.0	0.0	0.000000e+00	278.425382
min	54.057580	12.012028	13003.0	130030000.0	1.300300e+11	0.166667
25%	54.083985	12.084762	13003.0	130030000.0	1.300300e+11	30.572917
50%	54.090294	12.116659	13003.0	130030000.0	1.300300e+11	97.125000
75%	54.124206	12.136534	13003.0	130030000.0	1.300300e+11	336.750000
max	54.239682	12.224892	13003.0	130030000.0	1.300300e+11	1347.000000

Zählungen und Anteile

Eine weitere häufige Aufgabe, mit der wir konfrontiert sind, besteht darin, Werte zu zählen oder Anteile von bestimmten Werten zu berechnen. Die Methode `.value_counts` (siehe oben) kann verwendet werden, um die Anzahl kontinuierlicher Werte zu erhalten. Bei numerischen Werten funktioniert das aufgrund der Vielfalt and Werte nicht gut. Außerdem hilft es auch nicht Fragen die von Bedingungen abhängig sind. Zum Beispiel wollen wir wissen wie viele Baustellen im Jahr 2024 begonnen worden sind. Dafür müssen wir einen logischen Vektor erstellen, der die Bedingung beschreibt, und diesen summieren.

```
seit2024 = baust.baubeginn > "2024-01-01 00:00:00" # Erzeuge eine logische Variable die True ist wenn der baubeginn nach 2024 ist
seit2024.sum() # Zähle die Baustellen
```

```
np.int64(127)
```

Lassen Sie uns diese Schritte genauer erklären. Zuerst die Zeile

```
seit2024 = baust.baubeginn > "2024-01-01 00:00:00"
seit2024.head(5)
```

```
0     True
1     True
2     True
3     True
4    False
Name: baubeginn, dtype: bool
```

Erzeuge eine logische Variable die `True` oder `False` ist, wenn das Datum des Baubeginns nach 2024 liegt (um genau zu sein machen wir hier einen Stringvergleich und keinen Datumsvergleich. Da wir die Spalte nicht in ein Datum umgewandelt haben.). Ein Beispiel für diese Variable sieht so aus:

Als nächstes addiert `seit2024.sum()` diese Werte zusammen - denken Sie daran, sobald wir mit logischen Werten rechnen, wird `True` in “1” und `False` in “0” umgewandelt, und letzteres spielt beim Addieren keine Rolle. Es ist also so, als würden wir Baustellen zählen. In der Praxis müssen Sie keine separate Variable erstellen, sondern können es direkt berechnen

```
(baust.baubeginn > "2024-01-01 00:00:00").sum()
```

```
np.int64(127)
```

Die Berechnung von Proportionen kann auf ähnliche Weise erfolgen. Statt `.sum` verwenden wir einfach `.mean()`:

```
(baust.baubeginn > "2024-01-01 00:00:00").mean()
```

```
np.float64(0.6225490196078431)
```

Also gibt es 62% der Baustellen seit 2024.

Filtern von Daten

Schließlich, lassen wollen wir die Teilmengen der Baustellen erhalten, die entweder im “Hochbau” oder “Straßenbau” liegen. Dies kann mit der Methode `.isin()` durchgeführt werden, um zu überprüfen, ob ein String in einer gegebenen Liste vorhanden ist:

```
ab = baust[baust.sparte.isin(["Hochbau", "Straßenbau"])]
ab.head(5)
```

	latitude	longitude	uuid	kreis_name	kreis_schluessel	gemeindeverband_name	gemeindeverband_schluessel	gemeinde_name	gemeinde_schluessel	strasse_name	strasse_schluessel
5	54.147466	12.065587	0cfabba6-211c-42b0-b10c-3f2c6b2103e0	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Flensburger Str.	01930
8	54.143917	12.061527	140210b9-601a-482a-a14b-8f13a46879ef	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Möllner Str.	00600
9	54.135178	12.093463	150a9ed8-a7b1-4bf1-8e3f-95e569f480e6	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Schmarl-Dorf	08280
13	54.179967	12.084384	18728d03-7a43-4372-8564-f7519b29a64f	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Luisenstr.	06760
14	54.089303	12.108991	1a589275-aebe-497f-b981-1e8f09b81473	Rostock	13003.0	Rostock, Hanse- und Universitätsstadt	130030000.0	Rostock, Hanse- und Universitätsstadt	1.300300e+11	Waldemarstr.	09410

AUSWAHL VON VARIABLEN

Häufig enthalten Datensätze Variablen, die für die aktuelle Analyse irrelevant sind. In einem solchen Fall möchten wir entweder nur die relevanten auswählen (wenn es nur wenige solcher Variablen gibt) oder die irrelevanten entfernen (wenn es nicht zu viele davon gibt). Das Entfernen irrelevanter Informationen hilft uns, die Daten besser zu sehen, sie besser zu verstehen und somit weniger Codierfehler zu haben. Es hilft auch bei bestimmten Transformationen, bei denen wir jetzt möglicherweise auf den gesamten reduzierten Datenrahmen anstatt nur auf ausgewählte Variablen arbeiten möchten. Schließlich hilft es auch, Speicher zu sparen und die Verarbeitungsgeschwindigkeit zu erhöhen.

Auswahl der gewünschten Variablen

Wir haben besprochen, wie man die gewünschten Variablen auswählt, aber wir werden es hier auch kurz überprüfen. Sie können nur die gewünschten Variablen mit `dataframe["var1", "var2", ..."]` auswählen.

```
baust[["sparte", "baumassnahme", "verkehrsbeeintraechtigungen", "baubeginn", "bauende", "latitude", "longitude", "dauer"]].sample(4)
```

	sparte	baumassnahme	verkehrsbeeintraechtigungen	baubeginn	bauende	latitude	longitude	dauer
23	Telefonnetz	Tiefbauarbeiten für Breitbandausbau	Sicherungsmaßnahmen entlang des Gehwegs	2024-03-18 07:00:00+01:00	2024-04-26 18:00:00+02:00	54.082812	12.126326	39.416667
56	Gebäudesanierung	NaN	NaN	2024-01-22 00:00:00+01:00	2024-04-01 00:00:00+02:00	54.097021	12.093661	69.958333
94	Hochbau	Lkw Entladung am 02. & 04.04.2024	halbseitige Sperrung, Sicherungsmaßnahmen entl...	2024-04-02 00:00:00+02:00	2024-04-05 00:00:00+02:00	54.078083	12.142209	3.000000
168	Gebäudesanierung	Gerüststellung & BE für Fassadensanierung	Sicherungsmaßnahmen entlang der Straße, Sicher...	2023-09-21 00:00:00+02:00	2024-04-01 00:00:00+02:00	54.083904	12.114623	193.000000

Anstatt den gesamten Datensatz zu laden und später die gewünschten Variablen auszuwählen, können wir auch angeben, welche Spalten mit `pd.read_csv` gelesen werden sollen.

```
pd.read_csv("../data/Baustellen/baustellen_flat.csv",
            usecols=["latitude", "longitude", "sparte", "baubeginn", "bauende", "verkehrsbeeintraechtigungen", "baumassnahme", "dauer"]).sample(4)
```

	latitude	longitude	sparte	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
99	54.069910	12.117036	Kabelnetz	2024-01-15 00:00:00+01:00	2024-03-29 00:00:00+01:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	LWL- Anbindung in vorhandene Rohrtrasse	74.000000
95	54.064148	12.105620	Wasserleitung	2023-06-12 00:00:00+02:00	2024-05-04 00:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	Sanierung Trinkwasserleitung 3. BA	327.000000
70	54.071622	12.120165	Stromnetz	2024-03-18 07:00:00+01:00	2024-03-28 18:00:00+01:00	Sicherungsmaßnahmen entlang des Gehwegs	Herstellung Hausanschluss	10.458333
102	54.177265	12.080702	Hochbau	2023-10-30 00:00:00+01:00	2024-05-01 00:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	Abrissarbeiten	183.958333

Dies erreicht ein ähnliches Ergebnis, obwohl die Spalten jetzt in der ursprünglichen Reihenfolge und nicht in der angegebenen Reihenfolge sind.

Entfernen unerwünschter Variablen

Alternativ können wir, wenn wir die meisten Variablen behalten möchten, stattdessen angeben, welche entfernt werden sollen. Dies kann mit der Methode `.drop` erfolgen. Im Folgenden lassen wir eine große Anzahl von Variablen fallen:

```
baust.drop(["uuid", "kreis_name", "kreis_schluessel", "gemeindeverband_name", "gemeindeverband_schluessel", "gemeinde_name", "gemeinde_schluessel", "strasse_name", "strasse_schluessel"],
           axis=1 ).sample(4)
```

	latitude	longitude	sparte	von	nach	baubeginn	bauende	verkehrsbeeintraechtigungen	baumassnahme	dauer
154	54.074954	12.123408	Hochbau	31	34	2023-06-14 00:00:00+02:00	2025-06-28 00:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	Neubau Geschäftshaus	745.000000
164	54.093226	12.088681	Hochbau	10a	NaN	2023-02-20 00:00:00+01:00	2024-07-27 00:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	Sicherung Baufeld & Baustellenzufahrt	522.958333
84	54.093407	12.116542	Hochbau	7a	8a	2023-07-03 00:00:00+02:00	2024-05-01 00:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	Hochbauarbeiten, Zimmerarbeiten (Dachstuhl, Da...	303.000000
169	54.081561	12.135749	Gebäudesanierung	23	NaN	2023-04-01 08:00:00+02:00	2024-06-02 00:00:00+02:00	Sicherungsmaßnahmen entlang der Straße, Sicher...	Haussanierung (Gerüststellung, BE entlang des ...	427.666667

Beachten Sie, dass wir `.drop` mitteilen müssen, dass wir Spalten mit diesen Namen zu entfernen und nicht Zeilen mit diesem Index. Dies ist, was das Argument `axis=1` bewirkt. Andernfalls erhalten wir einen Fehler:

```
baust.drop(["uuid", "kreis_name", "kreis_schluessel", "gemeindeverband_name", "gemeindeverband_schluessel", "gemeinde_name", "gemeinde_schluessel", "strasse_name", "strasse_schluessel"]).sample(4)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[25], line 1
----> 1 baust.drop(["uuid", "kreis_name", "kreis_schluessel", "gemeindeverband_name", "gemeindeverband_schluessel", "gemeinde_name", "gemeinde_schluessel", "strasse_name", "strasse_schluessel"]).sample(4)

File ~/Documents/code/Lehre/IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/frame.py:6288, in DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   6284                 weight  250.0   150.0
   6285         falcon  speed   320.0   250.0
   6286                 weight  1.0     0.8
   6287         """
-> 6288         return super().drop(
   6289             labels=labels,
   6290             axis=axis,
   6291             index=index,

File ~/Documents/code/Lehre/IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/generic.py:4644, in NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
   4640         obj = self
   4641
   4642         for axis, labels in axes.items():
   4643             if labels is not None:
-> 4644                 obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4645
   4646         if inplace:
   4647             self._update_inplace(obj)

File ~/Documents/code/Lehre/IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/generic.py:4686, in NDFrame._drop_axis(self, labels, axis, level, errors, only_slice)
   4682                 if not isinstance(axis, MultiIndex):
   4683                     raise AssertionError("axis must be a MultiIndex")
   4684                 new_axis = axis.drop(labels, level=level, errors=errors)
   4685             else:
-> 4686                 new_axis = axis.drop(labels, errors=errors)
   4687             indexer = axis.get_indexer(new_axis)
   4688
   4689         # Case for non-unique axis

File ~/Documents/code/Lehre/IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/indexes/base.py:7268, in Index.drop(self, labels, errors)
   7266 if mask.any():
   7267     if errors != "ignore":
-> 7268         raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7269     indexer = indexer[~mask]
```

```
   7270 return self.delete(indexer)

KeyError: "['uuid', 'kreis_name', 'kreis_schluessel', 'gemeindeverband_name', 'gemeindeverband_schluessel', 'gemeinde_name', 'gemeinde_schluessel', 'strasse_name', 'strasse_schluessel'] not found in axis"
```

Ohne `axis=1` interpretiert Pandas die übergebenen Namen als **Zeilenindex** und nicht als Spaltennamen. Da diese Bezeichner nicht im Zeilenindex vorkommen, entsteht der gezeigte `KeyError`.

GRUPPIERTE OPERATIONEN

Wie andere große DataFrame-Bibliotheken unterstützt auch Pandas gruppierte Operationen. Gruppierte Operationen werden im Wesentlichen wie folgt durchgeführt: Zuerst wird die Daten anhand der Gruppierungsvariablenwerte in Gruppen aufgeteilt. Zweitens werden die folgenden Operationen auf allen Gruppen unabhängig voneinander durchgeführt. Und schließlich werden alle Ergebnisse wieder zusammengeführt, zusammen mit Gruppenindikatoren.

Gruppieren in Pandas kann mit der Methode `.groupby` durchgeführt werden. Es erwartet eine Liste von Gruppierungsvariablen. Wir demonstrieren dies, indem wir den Baustellendauer nach Sparte berechnen. `.groupby` erstellt lediglich ein gruppiertes Datenframe, der dann um eine Aggregationsfunktion erweitert wird um einen Ergebnisdataframe zu erhalten.

```
baust.groupby("sparte").dauer.mean()
```

```
sparte
Fernwärmeleitung       75.138889
Gasleitung            185.750000
Gebäudesanierung      179.201786
Gleisbau              939.541667
Grünpflege             68.593750
Hochbau               466.585417
Kabelnetz              51.462121
Kranarbeiten            3.500000
Lichtsignalanlage     260.472222
Straßenbau            295.767857
Straßenbeleuchtung     46.416667
Stromnetz              71.640625
Telefonnetz            54.661458
Wasserleitung         198.724167
privat                  2.250000
Name: dauer, dtype: float64
```

Das Ergebnis ist eine Serie mit zwei Werten (die numerische *dauer*) mit dem Index der kategorischen Werte von *sparte*.

Wir können auch nach mehr als einer Variable gruppieren. In diesem Fall müssen wir diese als Liste angeben, z.B. um die Baustellendauer nach Sparte und Baumaßnahme zu berechnen, können wir das tun.

```
baust.groupby(["sparte", "baumassnahme"]).dauer.mean()
```

```
sparte            baumassnahme
Fernwärmeleitung  Erweiterung Fernwärme Warnemünde                          316.375000
                  FW-Leitung i.a. der Hanseatic                             39.375000
                  Fernwärme                                                 39.333333
                  Fernwärmeerweiterung                                     186.041667
                  Fernwärmehausanschluss                                    32.000000
                                                                               ...
Wasserleitung     Sanierung Wasser/ Abwasser in versch. Bauphasen           494.958333
                  Schachtdeckel Havarie Übernahme VRAO Az.                  190.000000
                  Trennung Trinkwassergrundstücksanschluss                    3.000000
                  Umverlegung Trinkwasserleitung, SW & RW-Kanal             303.000000
privat            Fundamentarbeiten Balkonanlage -Betonpumpe & Betonmischer-    2.250000
Name: dauer, Length: 145, dtype: float64
```

Die Methode `.groupby` unterstützt weitere Optionen, z.B. kann man die Gruppenindikatoren als Variablen behalten anstatt als Index.

ZEICHENKETTEN OPERATIONEN

Pandas öffnet Zeichenkettenvariablen für eine große Liste von Zeichenkettenfunktionen mit dem Attribut `.str`. Diese replizieren größtenteils das *re* Modul, aber die Syntax ist anders und oft sind auch die Funktionsnamen unterschiedlich. Wir gehen hier durch eine Reihe von Beispielen, nämlich

- `str.contains` zum Suchen von Mustern in Zeichenketten
- `str.match` um festzustellen, ob eine Zeichenkette mit einem bestimmten Muster beginnt
- `str.replace` um Teile von Zeichenketten zu ersetzen
- `str.split` zum Zerteilen von Zeichenketten in Wörter

Viele der Zeichenkettenfunktionen nutzen reguläre Ausdrücke, daher ist es nützlich, eine Vorstellung von den Grundlagen regulärer Ausdrücke zu haben.

Lassen Sie uns Baustellendaten verwenden und analysieren, ob es ein regelmäßiges Muster in den Baumaßnahmen gibt. Wenn wir nur `unique` aufrufen, erhalten wir eine sehr unübersichtliche Liste.

```
baust.baumassnahme.unique()[:10]
```

```
<ArrowStringArray>
[                                            'FW-Leitung i.a. der Hanseatic',
                                                          'Baumpflanzung',
                                               'Einbau Trinkwasserschieber',
                                                                      nan,
                                 'Sanierung Mischwasser- und Trinkwasserleitung',
                                                       'Containerstellung',
                                          'Kennzeichnung Baustellenausfahrten',
 'Wartungs- und Instandsetzungsarbeiten innerhalb der konzessionierten Strecke',
                                     'Grünraumarbeiten, Sanierung Außenanlagen',
                                       'Ausschleifung - Muffung, StS Bussebart']
Length: 10, dtype: str
```

Allerdings sehen wir auch, dass einige Wörter wie z.B. “Sanierung” wiederholt vorkommen. Vieleicht ist es sinnvoller, die Zeichenkette in einzelne Worte zu zerlegen und diese zu zählen. Hierfür müssen wir mehrere Methoden verknüpfen, indem wir die Zeichenkette zuerst in Listen an Wörten mit `split` an allen vorkommenden Leerzeichen und Satzzeichen zerlegen. Mit `stack()` kombinieren wir diese Liste in eine neue Serie und zählen dann mit `value_counts()` wie häufig welche Wörter vorkommen.

```
baust.baumassnahme.str.split(expand=True).stack().value_counts()
```

```
Neubau             25
und                21
Sanierung          19
Breitbandausbau    13
für                12
                   ..
HRO                 1
Belieferung         1
Baustraße           1
Hort                1
Taklerring          1
Name: count, Length: 283, dtype: int64
```

Siehe da: Worte wie “Neubau” und “Sanierung” kommen durchaus oft vor.

Finde Muster in Zeichenketten

Als ersten Schritt wollen wir alle Baumaßnamen identifizieren, die das Wort “Sanierung” enthalten. Muster in Zeichenfolgen können mit `str.contains(Muster)` identifiziert werden. Diese Funktion gibt einen Vektor von Wahrheitswerten zurück, je nachdem, ob der ursprüngliche Zeichenvektor das Muster enthält:

```
baust.sparte.str.contains("sanierung", na=False, regex=False, case=False).head(4)
```

```
0    False
1    False
2    False
3     True
Name: sparte, dtype: bool
```

Das Argument `na` gibt an, was mit fehlenden Werten zu tun ist, hier zwingen wir sie, den Wert `false` zurückzugeben. `regex=False` bedeutet, dass das Muster, das wir angeben, kein regulärer Ausdruck ist. Einfache Muster sind einfacher und schneller zu verwenden als reguläre Ausdrücke, aber letztere sind viel leistungsfähiger, aber auch deutlich komplizierter. Der Parameter `case` gibt an, dass die Funktion nicht auf Groß/Kleinschreibung achten soll.

Jetzt wollen wir uns mal die Baumaßnahmen ansehen, die den Begriff Sanierung enthalten

```
baust.baumassnahme[baust.baumassnahme.str.contains("sanierung", na=False, regex=False, case=False)].value_counts()
```

```
baumassnahme
Sanierung Trinkwasserleitung                                3
Sanierungsarbeiten                                          2
Sanierung Mischwasser- und Trinkwasserleitung               1
Grünraumarbeiten, Sanierung Außenanlagen                    1
Sanierung Gehweg                                            1
Sanierung Altbau                                            1
Gehwegsanierung und Verkehrsberuhigung                      1
Gehwegsanierung                                             1
Gebäudesanierung                                            1
Sanierung TWL und Einflechtung Mischwasserkanal             1
Baustelleneinrichtung für Sanierung Kirrchplatz 10          1
Sanierung Geh- und Radweg                                   1
Sanierung Mischwasserkanal                                  1
Sanierung Gebäude                                           1
Sanierung Trinkwasserleitung 3. BA                          1
Sanierung Wohnhaus                                          1
Kabelsanierung                                              1
Sanierung Schachtdeckel                                     1
Sanierung Regen- und Trinkwasserleitung                     1
```

Bitte beachten Sie: Zuerst müssen Sie das Attribut `.str` verwenden, um eine Spalte für Zeichenfolgenoperationen zu öffnen. `.cabin.contains()` funktioniert nicht. `.contains` gibt einen logischen Wert zurück oder NA im Falle eines fehlenden Eintrags.

Ersetzen von Zeichenfolgen

Zuletzt wollen wir im Datensatz die deutschen Umlaute ersetzen, z.B. weil diese Probleme mit ML-Modellen bereiten könnten. Dies kann mit der Methode `str.replace` gemacht werden. Sie hat zwei Argumente: Muster und Ersatz. Wir geben das erstes einen Umlaut wie `ä` an und ersetzen ihn einfach durch `ae`:

```
baust.sparte.str.replace("ä", "ae").str.replace("ö", "oe").str.replace("ü", "ue").str.replace("ß", "ss").value_counts()
```

```
sparte
Hochbau                40
Gebaeudesanierung      35
Strassenbau            35
Wasserleitung          25
Fernwaermeleitung      18
Telefonnetz            16
Kabelnetz              11
Stromnetz               8
Gruenpflege             4
Lichtsignalanlage       3
Strassenbeleuchtung     3
Gleisbau                2
Kranarbeiten            2
Gasleitung              1
privat                  1
Name: count, dtype: int64
```

In praktischer Hinsicht ist es oft nützlich, den Originalspalte zu behalten, um zu prüfen, ob die Ersetzung richtig durchgeführt wurde. Dies kann erreicht werden, indem eine neue Spalte im Dataframe erstellt wird und eine Stichprobe der alten und neuen Variablen ausgedruckt wird:

```
baust["sparte_ASCII"] = baust.sparte.str.replace("ä", "ae").str.replace("ö", "oe").str.replace("ü", "ue").str.replace("ß", "ss")
baust[["sparte", "sparte_ASCII"]].value_counts()
```

```
sparte              sparte_ASCII
Hochbau             Hochbau                40
Gebäudesanierung    Gebaeudesanierung      35
Straßenbau          Strassenbau            35
Wasserleitung       Wasserleitung          25
Fernwärmeleitung    Fernwaermeleitung      18
Telefonnetz         Telefonnetz            16
Kabelnetz           Kabelnetz              11
Stromnetz           Stromnetz               8
Grünpflege          Gruenpflege             4
Lichtsignalanlage   Lichtsignalanlage       3
Straßenbeleuchtung  Strassenbeleuchtung     3
Gleisbau            Gleisbau                2
Kranarbeiten        Kranarbeiten            2
Gasleitung          Gasleitung              1
privat              privat                  1
Name: count, dtype: int64
```

UMGANG MIT ZEITSTEMPELN

Wenn man Zeitreihen analysiert, muss man immer mit Zeitstempeln umgehen. Das ist leider fehleranfälliger, als man vermuten mag, weil es sehr viele unterschiedliche Arten gibt Zeitstemmpel darzustellen und es Besonderheiten wie Zeitzonen, Sommer/Winter-Zeit, oder Schaltjahre gibt.

Prinzipiell können wir Datumsangaben und Zeiten mit der Funktion `pd.to_datetime` in ein richtiges Datetime-Objekt umwandeln, mit dem man auch korrekte Zeitberechnungen machen kann (und nicht solche Workarounds, wie oben). Bei einzelnen Werten kann die Funktion sogar Datums und Zeitformate selbst erkennen.

```
pd.to_datetime("2023-03-08")
```

```
Timestamp('2023-03-08 00:00:00')
```

```
pd.to_datetime("03/09/2023")
```

```
Timestamp('2023-03-09 00:00:00')
```

```
pd.to_datetime("10-Mar-2023")
```

```
Timestamp('2023-03-10 00:00:00')
```

```
pd.to_datetime("2023-03-11")
```

```
Timestamp('2023-03-11 00:00:00')
```

```
pd.to_datetime("12:00")
```

```
Timestamp('2026-05-11 12:00:00')
```

```
pd.to_datetime("14:30:00")
```

```
Timestamp('2026-05-11 14:30:00')
```

```
pd.to_datetime("10:00:00.400")
```

```
Timestamp('2026-05-11 10:00:00.400000')
```

```
pd.to_datetime("18:00+0030")
```

```
Timestamp('2026-05-11 18:00:00+0030', tz='UTC+00:30')
```

```
pd.to_datetime("2023-03-08 12:00")
```

```
Timestamp('2023-03-08 12:00:00')
```

```
pd.to_datetime("03/09/2023 14:30:00")
```

```
Timestamp('2023-03-09 14:30:00')
```

```
pd.to_datetime("10-Mar-2023 10:00:00.400")
```

```
Timestamp('2023-03-10 10:00:00.400000')
```

```
pd.to_datetime("2023-03-11T18:00+0030")
```

```
Timestamp('2023-03-11 18:00:00+0030', tz='UTC+00:30')
```

Wenn wir diese allerdings in einer Serie kombinieren, funktioniert das nicht mehr

```
# Beispieldaten
data = pd.DataFrame({
    "Datumzeit": ["2023-03-08 12:00","03/09/2023 14:30:00","10-Mar-2023 10:00:00.400","2023-03-11T18:00+0030"]
})
```

```
# Spalte "Uhrzeit" in Zeitformat konvertieren
data["DatumzeitDT"] = pd.to_datetime(data["Datumzeit"])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[47], line 2
      1 # Spalte "Uhrzeit" in Zeitformat konvertieren
----> 2 data["DatumzeitDT"] = pd.to_datetime(data["Datumzeit"])

File ~/Documents/code/Lehre/IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/tools/datetimes.py:1040, in to_datetime(arg, errors, dayfirst, yearfirst, utc, format, exact, unit, origin, cache)
   1038         result = arg.map(cache_array)
   1039     else:
-> 1040         values = convert_listlike(arg._values, format)
   1041         result = arg._constructor(values, index=arg.index, name=arg.name)
   1042 elif isinstance(arg, (ABCDataFrame, abc.MutableMapping)):

File ~/Documents/code/Lehre/IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/tools/datetimes.py:435, in _convert_listlike_datetimes(arg, format, name, utc, unit, errors, dayfirst, yearfirst,
    433 # `format` could be inferred, or user didn't ask for mixed-format parsing.
    434 if format is not None and format != "mixed":
--> 435     return _array_strptime_with_fallback(arg, name, utc, format, exact, errors)
    437 result, tz_parsed = objects_to_datetime64(
    438     arg,
    439     dayfirst=dayfirst,
   (...)    443     allow_object=True,
    444 )
    446 if tz_parsed is not None:
    447     # We can take a shortcut since the datetime64 numpy array
    448     # is in UTC

File ~/Documents/code/Lehre/IntroductionMachineLearning/.venv/lib/python3.12/site-packages/pandas/core/tools/datetimes.py:470, in _array_strptime_with_fallback(arg, name, utc, fmt, exact, errors)
    459 def _array_strptime_with_fallback(
    460     arg,
    461     name,
   (...)    465     errors: str,
    466 ) -> Index:
    467     """
    468     Call array_strptime, with fallback behavior depending on 'errors'.
```

```
    469     """
--> 470     result, tz_out = array_strptime(arg, fmt, exact=exact, errors=errors, utc=utc)
    471     if tz_out is not None:
    472         unit = np.datetime_data(result.dtype)[0]

File pandas/_libs/tslibs/strptime.pyx:563, in pandas._libs.tslibs.strptime.array_strptime()
--> 563 'Could not get source, probably due dynamically evaluated source code.'

File pandas/_libs/tslibs/strptime.pyx:511, in pandas._libs.tslibs.strptime.array_strptime()
--> 511 'Could not get source, probably due dynamically evaluated source code.'

File pandas/_libs/tslibs/strptime.pyx:617, in pandas._libs.tslibs.strptime._parse_with_format()
--> 617 'Could not get source, probably due dynamically evaluated source code.'

ValueError: time data "03/09/2023 14:30:00" doesn't match format "%Y-%m-%d %H:%M". You might want to try:
    - passing `format` if your strings have a consistent format;
    - passing `format='ISO8601'` if your strings are all ISO8601 but not necessarily in exactly the same format;
    - passing `format='mixed'`, and the format will be inferred for each element individually. You might want to use `dayfirst` alongside this.
```

Deshalb muss man zuerst darauf achten, dass entsprechende Zeitstempel immer in einem festen Format sind.

```
# Beispieldaten
data = pd.DataFrame({
    "Datumzeit": [
        "2023-03-08T12:00:00.000+0030",
        "2023-03-09T14:30:00.000+0030",
        "2023-03-10T10:00:00.400+0030",
        "2023-03-11T18:00:00.000+0030"]
})
```

```
# Spalte "Uhrzeit" in Zeitformat konvertieren
data["DatumzeitDT"] = pd.to_datetime(data["Datumzeit"], format='ISO8601')
data.DatumzeitDT
```

```
0          2023-03-08 12:00:00+00:30
1          2023-03-09 14:30:00+00:30
2   2023-03-10 10:00:00.400000+00:30
3          2023-03-11 18:00:00+00:30
Name: DatumzeitDT, dtype: datetime64[us, UTC+00:30]
```

Bei unsicheren Mustern kann man das Format direkt spezifizieren, Dafür wird die strftime-Notation verwendet

```
# Spalte "Uhrzeit" in Zeitformat konvertieren
data["DatumzeitDT"] = pd.to_datetime(data["Datumzeit"], format="%Y-%m-%dT%H:%M:%S.%f%z")
data.DatumzeitDT
```

```
0          2023-03-08 12:00:00+00:30
1          2023-03-09 14:30:00+00:30
2   2023-03-10 10:00:00.400000+00:30
3          2023-03-11 18:00:00+00:30
Name: DatumzeitDT, dtype: datetime64[us, UTC+00:30]
```

Mit dem gleichen Muster können Datumsangaben auch wieder in Strings umgewandelt werden.

```
data.DatumzeitDT.dt.strftime('%Y-%m-%d %H:%M')
```

```
0    2023-03-08 12:00
1    2023-03-09 14:30
2    2023-03-10 10:00
3    2023-03-11 18:00
Name: DatumzeitDT, dtype: str
```

QUESTIONS