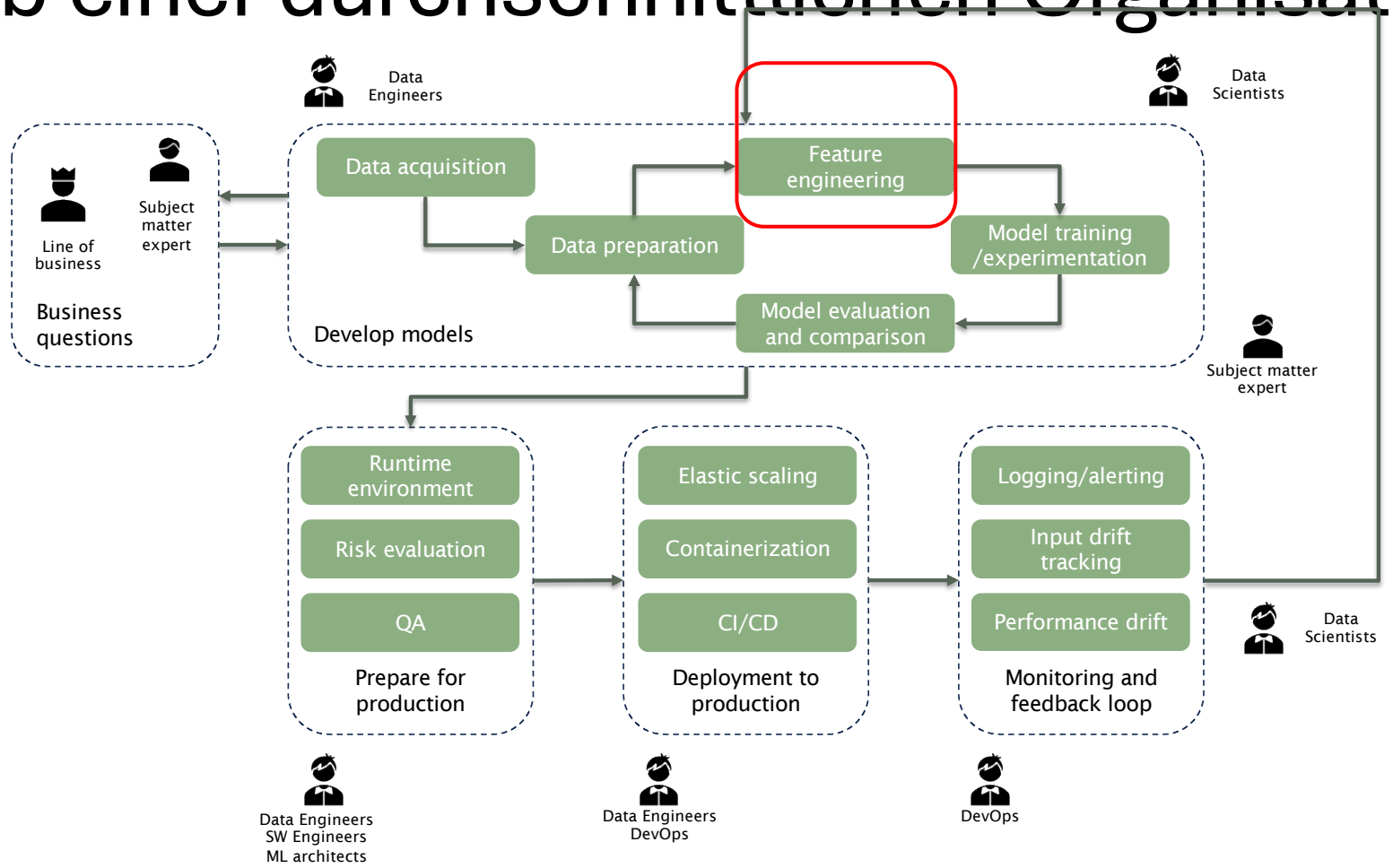
Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Unüberwachtes Lernen: Feature Engineering Praxis

Violeta Vogel

Technik und Informatik, BFH

Das realistische Bild eines ML-Lebenszyklus innerhalb einer durchschnittlichen Organisation



Feature Engineering: Transformation

- da sich einzelne Transformationsschritte aber auf andere auswirken können, wird nach dem Sandbox-Prinzip gearbeitet, d.h. die Untersuchungen werden jeweils auf einer Kopie der Ausgangsdaten (`ori_data`) durchgeführt, damit sie nicht jedes Mal neu geladen werden müssen

```
ori_data = pd.read_csv('bank_data.csv', sep=';')
```

- jeder Codeblock beginnt danach mit folgender Anweisung

```
data = ori_data.copy()
```

Data Frame: Entfernen der Beobachtungen

Nach Index

- Gezieltes Entfernen von Beobachtungen nach Index, z.B. Entfernen der Beobachtungen mit dem grössten Wert von "age" aus dem Data Frame
- dazu wird der Index der entsprechenden Beobachtung(en) ermittelt und in einer Liste hinterlegt, welche beim folgenden Aufruf von `.drop()` als Parameter mitgegeben wird

```
idx = (data.age[data.age == max(data.age)].index[0]).tolist()
data.drop(idx, inplace=True)
```

Data Frame: Entfernen der Beobachtungen

Nach Bedingung

- Entfernen von Beobachtungen nach Bedingung(en), z.B. Entfernen aller Beobachtungen, für welche in der Variable "age" der Wert >= 100 ist

```
data.drop(data[data.age >= 100].index, inplace=True)
```

- **Entfernen von Duplikaten**

```
data.drop_duplicates(ignore_index=True, inplace=True)
```

Data Frame: Entfernen der Beobachtungen

Nach Index

- Entfernen von Variablen nach deren Spaltenindex im Data Frame (beginnend bei 0), z.B. entfernen der ersten drei Variablen (von links)

```
cols_to_drop = [0, 1, 2]
data.drop(data.columns[cols_to_drop], axis=1, inplace=True)
```

- aus Transparenzgründen wird erst eine Liste mit den Indices der zu entfernenden Variablen erstellt, welche der anschliessenden Methode .drop() als Parameter mitgegeben wird

Data Frame: Entfernen der Beobachtungen

Nach Name

- gezieltes Entfernen von Variablen nach deren Namen im Data Frame, z.B. entfernen von "marital" und "education"

```
cols_to_drop = ['marital', 'education']
data.drop(cols_to_drop, axis=1, inplace=True)
```

- auch hier wird mit zwei Schritten gearbeitet, welche kompakter codiert werden könnte (vgl. [ipynb])

Data Frame: Missing Values

- Methoden:
  - entfernen aller Beobachtungen (rows) mit NAs
  - entfernen aller Variablen (columns) mit NAs
- können unter Umständen zu grossem Datenverlust führen

- Alternativen:
  - einsetzen eines willkürlichen Wertes
  - einsetzen eines errechneten Wertes (z.B. Modalwert bei Kategorialen Variablen, Median bei Numerischen Variablen), was im Folgenden gezeigt werden soll
  - einsetzen eines (mittels ML) geschätzten wahrscheinlichsten Wertes

Data Frame: Missing Values

Kategoriale Variablen

- Ersetzen von NAs bei "marital" durch den Modalwert aller nicht-NA Werte dieser Variablen

```
data.marital.fillna(data.marital.mode()[0], inplace=True)
```

- der Index `[0]` bei der Methode `.mode()` ist hier wichtig, da letztere mehrere Werte zurückgeben könnte und daher eine Liste ist

Numerische Variablen

- Ersetzen von NAs bei "age" durch den Median aller nicht-NA Werte dieser Variablen

```
data.age.fillna(data.age.median(), inplace=True)
```

Kategoriale Variablen: Kardinalität

- unter Feature Exploration wurde festgestellt, dass z.B. für die Variable "education" eine Kategorie (Level) mit vergleichsweise wenigen Werten vorliegt: "illiterate"
- falls dies vom Fach so akzeptiert wird, kann diese Kategorie mit "basic.4y" zusammengelegt werden

```
data.education = np.where(
    data.education == 'illiterate', ## condition
    'basic.4y',                     ## if true
    data.education)                 ## if false
```

- dazu besonders geeignet ist die Funktion `where` aus dem Package `numpy(np)`
- bedarfsweise können auch mehrere Kategorien zu einer zusammenkombiniert werden, indem mehrere Bedingungen verknüpft werden (vgl. [ipynb])

Kategoriale Variablen: Numerisieren

- um Informationen aus Kategorialen Variablen in Machine Learning verwenden zu können, muss diese numerisch dargestellt werden können
- dazu stehen verschiedene Möglichkeiten zur Verfügung
  - Faktorisieren
  - Ordinal Encodieren
  - Nominal Encodieren,

Kategoriale Variablen: Numerisieren - Faktorisieren

- jeder Kategorie einer Kategorialen Variablen wird ein Integer-Wert zugeordnet (beginnend bei 0)
- z.B. für die Variable "job":

```
data.job = pd.factorize(data.job)[0]
```

- tatsächlich gibt `.factorize()` ein Tupple mit folgende Komponenten zurück
  - `numpy.ndarray`: faktorisierte Werte, beginnend bei 0
  - `pandas.core.indexes.base.Index`: Zuordnungen der obigen Werte zu den Ausgangswerten (erstes Element für 0)
- mit dem oben stehenden Code können die Komponenten gleich beim Aufruf aufgetrennt werden (`[0]` nach `.factorize()`)
- per Default werden die Faktorwerte gemäss ihrem sequentiellen Auftreten vergeben
- mit dem Parameter sort=True werden die Faktorwerte lexikografisch in Bezug auf die Ausgangswerte vergeben

Kategoriale Variablen: Numerisieren – ordinal Encodieren

- eine Schwäche von Faktorisieren: die numerischen Werte werden scheinbar willkürlich zugeordnet (nach deren sequenziellen Auftreten im Dataset)
- um also eine als ordinal skaliert erkannte Variable korrekt zu numerisieren, müssen die gezielt den einzelnen Kategorien zugeordnet werden können
- am Beispiel der Variable "education" könnte eine solche Zuordnung wie folgt aussehen:

illiterate	→	0
unknown	→	0
basic.4y	→	1
basic.6y	→	2
basic.9y	→	3
professional.course	→	4
high.school	→	5
university.degree	→	6

Kategoriale Variablen: Numerisieren – ordinal Encodieren

- als dazu geeignete Funktion wird hier `.replace()` eingesetzt, welche als Parameter ein Python-Dictionary entgegennimmt, welches idealerweise in einem vorherigen Schritt definiert wird

```
replace_nums = {    ## a dictionary of dictionaries
    'education': {
        'illiterate': 0,
        'unknown': 0,
        'basic.4y': 1,
        'basic.6y': 2,
        'basic.9y': 3,
        'professional.course': 4,
        'high.school': 5,
        'university.degree': 6
    }
}
```

Kategoriale Variablen: Numerisieren – ordinal Encodieren

- der effektive Aufruf:

```
data.replace(replace_nums, inplace=True)
```

- es können auch gleich mehrere Variablen mit einem Aufruf encodiert werden (vgl. [ipynb])

- es gibt zwar folgenden Encoder von scikit-learn: `sklearn.preprocessing.OrdinalEncoder`
- macht aber tatsächlich eine Faktorisierung, für effaktiv ordinales Encodieren ist wesentlich anspruchsvollere Parametrisierung notwendig

Kategoriale Variablen: Numerisieren – ordinal Encodieren

Spezialfall: 0-1 Encodieren

- falls Kategoriale Variablen nur zwei Kategorien aufweisen, können diese auch einfach mit `numpy.where` encodiert werden
- so weist z.B. "contact" nur die Kategorien "cellular" und "telephone" auf

```
data['contact'] = np.where(data.contact == 'cellular', 1, 0)
```

- der obenstehende Aufruf weist dem Wert "cellular" neu den Wert 1 zu, allen anderen (!) den Wert 0

- aus Gründen der Transparenz kann es Sinn machen, die umcodierte Variable anschliessend umzubenennen, z.B. wie folgt:

```
data.rename(columns = {'contact' : 'contact_cellular'}, inplace=True)
```

Kategoriale Variablen: Numerisieren – one hot encoding (nominal)

oder: Erstellen von Dummy-Variablen

- für die folgenden Darstellungen wird aus praktischen Gründen eine Zufallsstichprobe der Daten erstellt

```
data = ori_data.sample(6, random_state=1234)
print(data.marital)
```

```
9056      single
9483    divorced
788       single
9554    divorced
809     divorced
4822     married
```

Kategoriale Variablen: Numerisieren – one hot encoding (nominal)

- die Pandas-Funktion .get_dummies() erstellt für jede auftretende Kategorie eine neue Variable (Dummy Variable) **und entfernt die Ausgangsvariable**
- dabei stehen verschiedene Parametrisierungsmöglichkeiten zur Verfügung
- für die folgenden Darstellungen wird das Ergebnis jeweils den Ausgangsdaten gegenübergestellt

```
new_data = pd.get_dummies(data.marital)
print(pd.merge(data.marital, new_data, left_index=True, right_index=True))
```

```
       marital  divorced  married  single
9056    single         0        0       1
9483  divorced         1        0       0
788     single         0        0       1
9554  divorced         1        0       0
809   divorced         1        0       0
4822   married         0        1       0
```

Kategoriale Variablen: Numerisieren – one hot encoding (nominal)

- mit Parameter `drop_first`:

```
new_data = pd.get_dummies(data.marital, drop_first=True)
:
```

```
       marital  married  single
9056    single        0       1
9483  divorced        0       0
788     single        0       1
9554  divorced        0       0
809   divorced        0       0
4822   married        1       0
```

- es wird eine Dummy-Variable weniger erstellt, als ursprünglich Kategorien vorhanden sind

Kategoriale Variablen: Numerisieren – one hot encoding (nominal)

- mit Parameter prefix:

```
new_data = pd.get_dummies(data.marital, prefix='marital')
:
```

	marital	marital_divorced	marital_married	marital_single
9056	single	0	0	1
9483	divorced	1	0	0
788	single	0	0	1
9554	divorced	1	0	0
809	divorced	1	0	0
4822	married	0	1	0

- dem Namen der Dummy-Variablen wird jeweils der Name der Ausgangsvariable als Präfix mitgegeben - ist vor allem dann angebracht, wenn mehrere Variablen auf diese Weise transformiert werden sollen

Kategoriale Variablen: Numerisieren – one hot encoding (nominal)

- `.get_dummies()` kann auch gleich auf mehrere Variablen gleichzeitig angewendet werden

```
data = pd.get_dummies(
    data,
    columns=['job', 'marital', 'loan'],
    drop_first=True)
```

- da im obigen Aufruf der Parameter columns eine Liste der zu berücksichtigenden Variablen enthält, ist es auch möglich, für alle zum Zeitpunkt nicht kategorialen Variablen (ausser vielleicht dem Target) mit einem Aufruf Dummy-Variablen zu erstellen

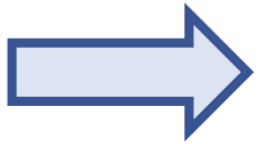
```
target = 'y'
sel_vars = data.select_dtypes(include=['object']).columns.drop(target)
data = pd.get_dummies(data, columns=sel_vars, drop_first=True)
```

Numerische Variablen: Skalieren - Normalisieren

- Vorgehen Normalisieren am Beispiel von "age"

```
data.age = (data.age - data.age.min()) / \
    (data.age.max() - data.age.min())
```

```
min     17.0
max    116.0
```

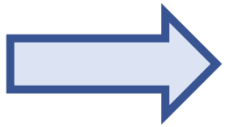


```
min    0.0
max    1.0
```

Numerische Variablen: Skalieren - Standartisieren

- Vorgehen Standardisieren am Beispiel von "age"

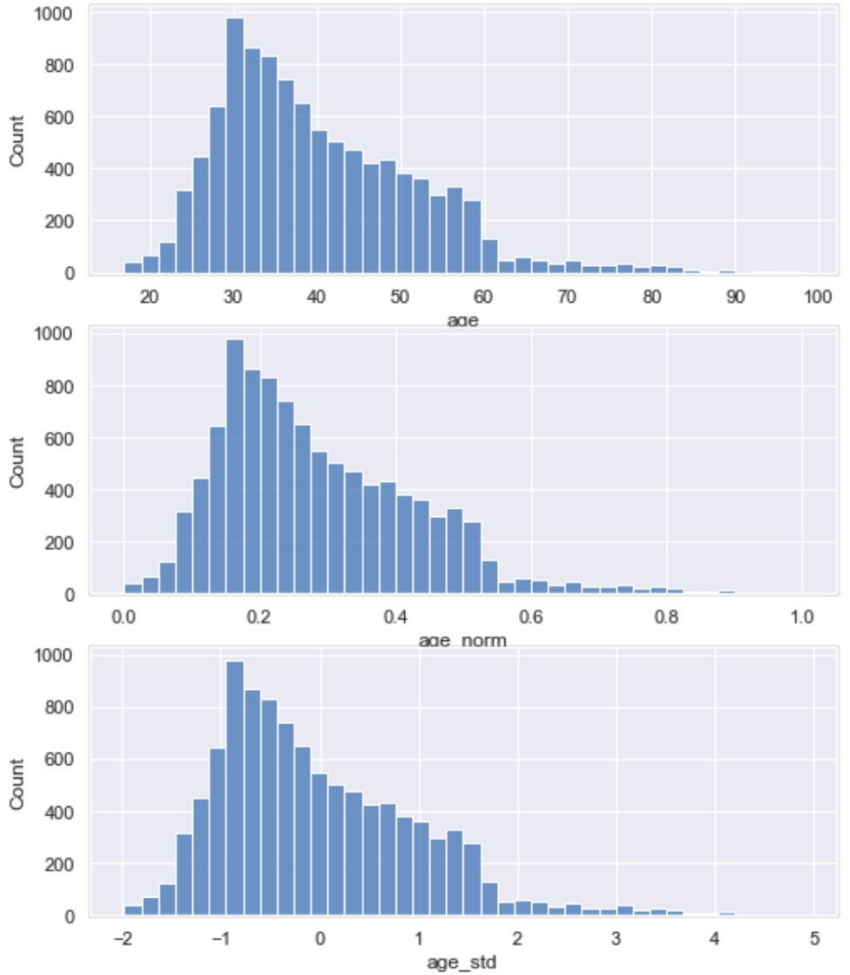
```
data.age = (data.age - data.age.mean()) / \
    data.age.std()
```

mean	40.423191		mean	1.795497e-17
std	11.915715		std	1.000000e+00

Numerische Variablen: Skalieren – Gegenüberstellung der Verfahren

- eine visuelle Gegenüberstellung der beiden Verfahren ([ipynb])
  - Ausgangswerte (oben)
  - normalisiert (mitte)
  - standardisiert (unten)

- an der Verteilung ändert sich nichts, einziger Unterschied: Skala der x-Achse



Numerische Variablen: Skalieren

- das Ganze macht nur Sinn, wenn die jeweilige Transformation gleich auf **alle** interessierenden Variablen angewendet wird
- ausserdem muss unter Umständen die Transformation für späteren Gebrauch mit neuen Daten hinterlegt werden können
- die Library scikit-learn (sklearn) bietet dazu im Modul sklearn.preprocessing (unter anderen) die folgenden beiden Funktionen an:
  - `MinMaxScaler`
  - `StandardScaler`

Numerische Variablen: Skalieren

- Vorgehen Skalieren aller numerischen Variablen des Data Frame

```
data = ori_data.select_dtypes(exclude=['object'])
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler().set_output(transform="pandas")
data = scaler.fit_transform(data)
```

- hier exemplarisch mit MinMaxScaler, welcher bei Standard-Parametrisierung eine Normalisierung (0-1 Transformation) durchführt
- die verschiedenen Funktionen sind im [ipynb] ausführlich kommentiert
- in den meisten Situationen werden diese Transformationen erst unmittelbar beim Trainieren eingesetzt, weshalb bei der Implementierung vorerst darauf verzichtet wird

Numerische Variablen: Binning

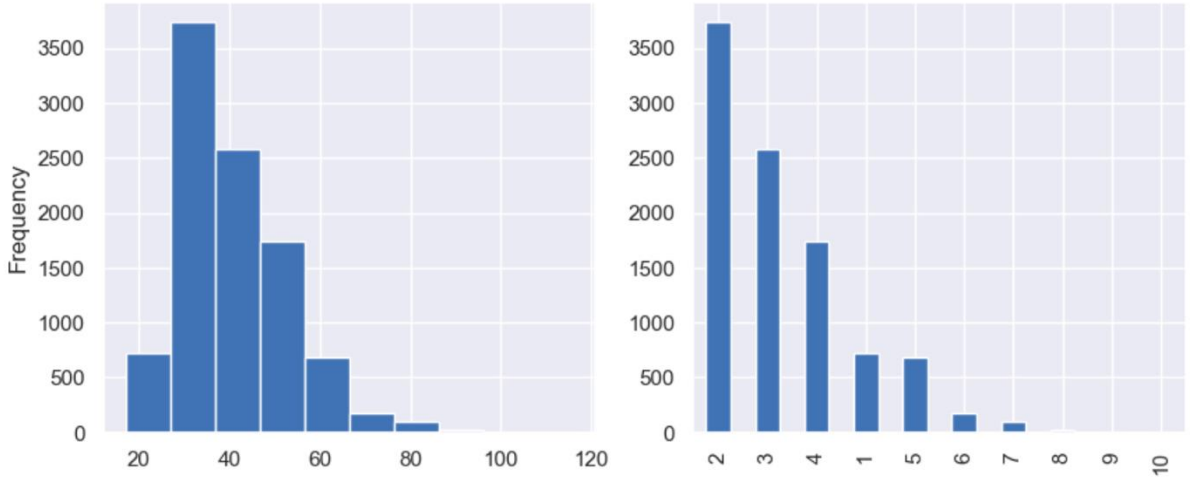
- für die Bestimmung der optimalen Anzahl Klassen bei der Erstellung von Histogrammen liegen allerdings unterschiedliche Vorstellungen vor:
  - seaborn.histplot: Binwidth gemäss Freedman–Diaconis rule
  - pandas.DataFrame.hist: 10 (dasselbe wie plt.hist), dabei wird der Bereich zwischen min und max konsequent in 10 gleich grosse Bereiche unterteilt, was wie bei seaborn.histplot zu unschönen Klassengrenzen führen kann
  - R hist(): Sturges Regel, ausserdem werden mit pretty() die Grenzen so gesetzt, dass sie durch 1, 2, oder 5 teilbar sind
  - R ggplot2::geom_histogram(): 30
- viele Autoren sind sich immerhin einig, dass es keinen "vernünftigen" Defaultwert gibt, dass also unterschiedliche Anzahlen Bins, resp. Binwidth experimentell ermittelt werden sollten

Numerische Variablen: Binning – Equal bining

- Vorgehen Equal Binning mit pd.cut am Beispiel von "age"

```
bins = 10
data.age = (pd.cut(
    data.age,
    bins = bins,
    labels = list(range(1, bins + 1))))
```

- links: Histogramm von "age" vor Binning
- rechts: Barplot nach Binning

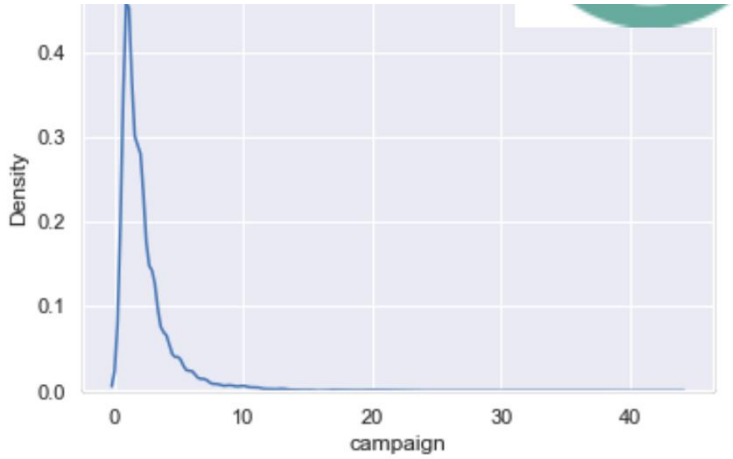


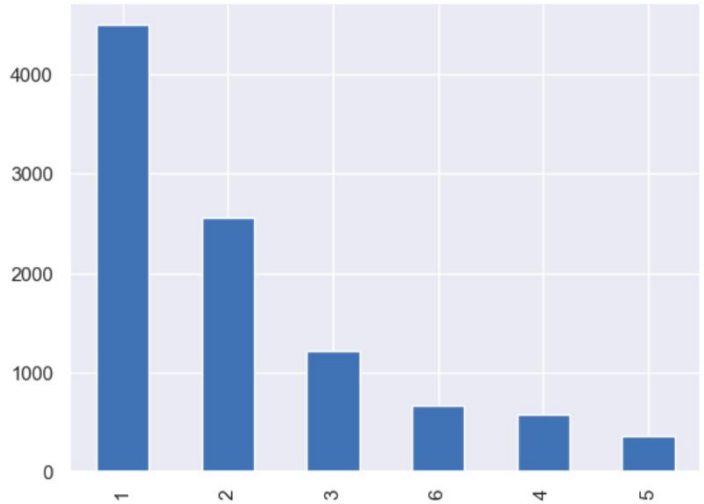
Numerische Variablen: Binning

- "campaign" zeigt eine extrem (rechts-) schiefe Verteilung
- eine Analyse mit value_counts() zeigt darüber hinaus, dass z.B. Werte ab einer bestimmten Grenze auf deren Wert zurückgeschnitten werden könnten (hier Werte > 6 → 6
- dies kann mit einer einfachen Umformung erreicht werden:

```
data.campaign = np.where(
    data.campaign <= 5, data.campaign, 6)
```

- anschliessend visualisieren mit Barplot



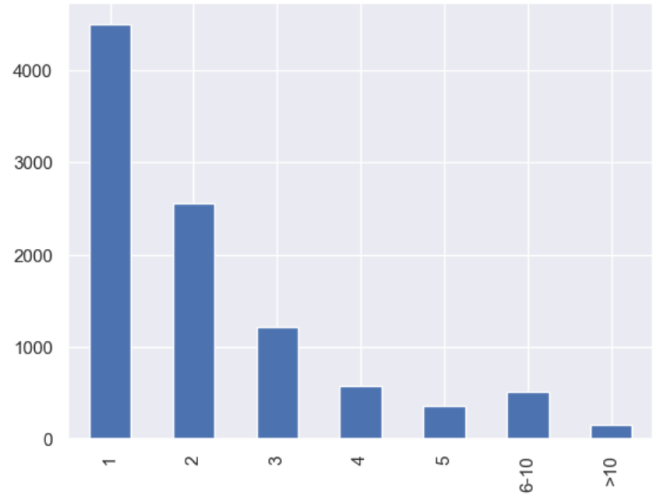


Numerische Variablen: Binning

- für unterschiedliche Bin-Bereiche können wiederum mit pandas.cut() unterschiedliche Bins definiert werden, z.B.
  - Werte von 1 bis 5 so übernehmen
  - zusätzlich je eine Klasse für
    - Werte von 6 - 10
    - Werte > 10

```
data.campaign = pd.cut(
    data.campaign,
    bins = [0, 1, 2, 3, 4, 5, 10, 1000],
    labels = [1, 2, 3, 4, 5, '6-10', '>10'])
```

- anschliessend visualisieren mit Barplot



Bereinigen von Variablennamen

- durch Nominal Encodieren (One-Hot Encoding) können seltsame Variablennamen entstehen
- die Namen der entstehenden Dummy-Variablen setzen sich zusammen aus dem Namen der Ausgangsvariable sowie der Bezeichnung der jeweiligen Kategorie
- am Beispiel von "job" z.B. wie folgt:

```
data = pd.get_dummies(ori_data)
print(data.columns[data.columns.str.contains('job_')].tolist())
```

```
['job_admin.', 'job_blue collar', 'job_entrepreneur', 'job_housemaid',
'job_management', 'job_retired', 'job_self-employed', 'job_services',
'job_student', 'job_technician', 'job_unemployed']
```

- "job_blue collar": enthält ein Leerzeichen
- "job_self-employed" enthält einen Bindestrich

Bereinigen von Variablennamen

- ausserdem enthalten die sozioökonomischen Variablen ("emp.var.rate", etc.) einen Punkt im Namen
- dies ist normalerweise kein Problem
- allerdings behandeln verschiedene Methoden des Machine Learning die Variablen (Features) als eigenständige Python-Objekte, und dort sind solche Zeichen in Bezeichnern nicht erlaubt
- mittels der Funktion `.str.contains()` und eines Regulären Ausdrucks (Regex) kann untersucht werden, welche Variablen ungünstige Zeichen enthalten, dabei werden die Namen angezeigt, welche andere als die erlaubten Zeichen (`a-z, A-Z, 0-9, _`) enthalten

```
old_names = data.columns
old_names = old_names[old_names.str.contains('[^a-zA-Z0-9_]')]
print(old_names.tolist()) ## check
```

```
['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'nr.employed', 'job_admin.',
'job_blue collar', 'job_self-employed', 'education_basic.4y',
'education_basic.6y', 'education_basic.9y', 'education_high.school',...
```

Bereinigen von Variablennamen

- mit der Funktion `.str.replace()` kann ebenfalls mit Unterstützung von Regex eine Liste erzeugt werden, in welcher alle unerlaubten Zeichen z.B. durch "_" ersetzt werden

```
new_names = old_names.str.replace('[^a-zA-Z0-9_]', '_', regex=True)
```

- die beiden oben erstellten Listen (old_names und new_names) können anschliessend verwendet werden, um die fraglichen Variablen im Data Frame umzubenennen

```
for i in range(len(old_names)):
    data.rename(columns={old_names[i]:new_names[i]}, inplace=True)
```

Konstruktion

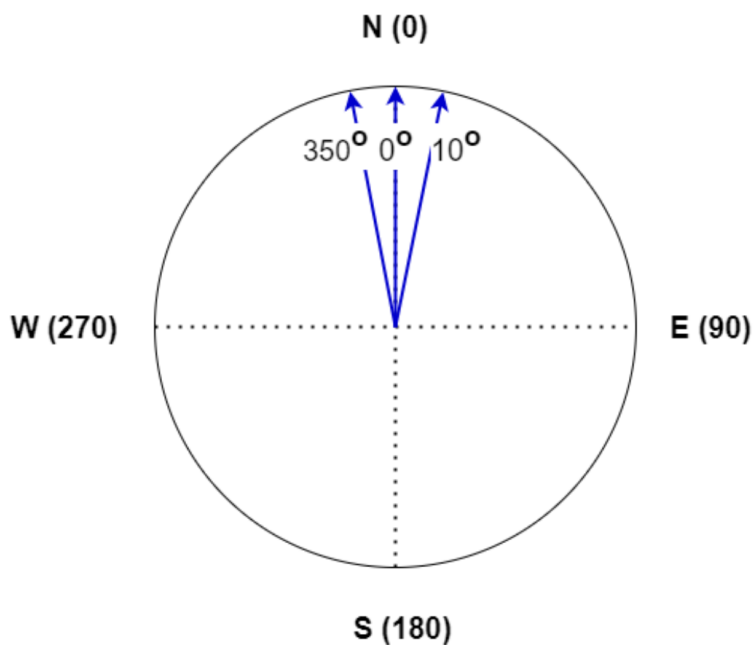
- darunter versteht man das Erstellen neuer Variablen durch Ableiten aus bestehenden
- Ziele:
  - reduzieren der Komplexität
  - vermeiden von Korrelationen

- zwei Techniken stehen hier im Vordergrund
  - gezieltes Konstruieren von neuen Variablen aus bestehenden und ersetzen der Ausgangsvariablen
  - Dimensionsreduktion mit Methoden des Nichtüberwachten Lernens (dies als Ausblick)

Konstruktion

Beispiel Wetterdaten

- die quantitative Beschreibung von Wettersituationen besteht meist aus einer Anzahl unterschiedlicher Merkmale wie
  - Temperatur, Luftdruck, Luftfeuchtigkeit, Niederschlagsmenge, Bewölkungsgrad etc.
  - ausserdem Windgeschwindigkeit und Windrichtung
- während die meisten metrisch skaliert und damit für Machine Learning unproblematisch sind, trifft dies für die Windrichtung nicht zu
- diese kann in unterschiedlicher Form vorliegen
  - nominal oder ordinal (N, E, S, W oder N, NE, E, SE, ...)
  - metrisch als Abweichung in Grad gegenüber Norden im Uhrzeigersinn
- letzteres ist insofern problematisch, als z.B. eine kleine Abweichung von N in Richtung W zu einem sehr grossen Wert führt (< 360º) während N selber 0º beträgt



Konstruktion

Abhilfe

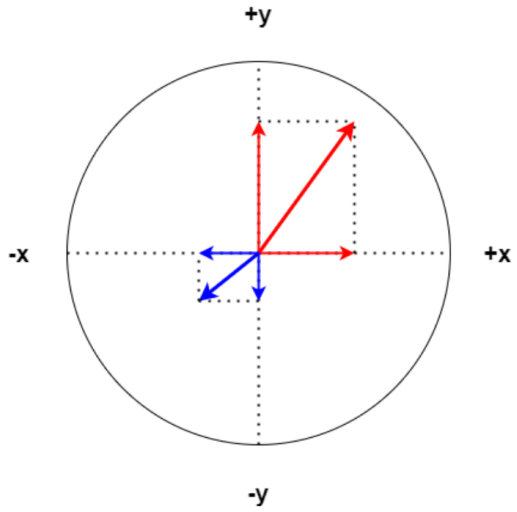
- zerlegen der Richtung in eine x- und y-Komponente mit Hilfe trigonometrischer Funktionen, z.B. die W-E Achse als x, die N-S Achse als y
- multiplizieren der neuen Komponenten mit Geschwindigkeit, formal:

$$x = sin\left(\frac{direction \cdot \pi}{180}\right) \cdot speed \qquad y = cos\left(\frac{direction \cdot \pi}{180}\right) \cdot speed$$

- oder als Code

```
import numpy as np
data['x'] = np.sin(data.direction * np.pi / 180) * data.speed
data['y'] = np.cos(data.direction * np.pi / 180) * data.speed
```

- die Ausgangsvariablen direction und speed können danach aus dem Data Frame entfernt werden



Konstruktion

Beispiel Datum Typ

- im Melbourne Housing Dataset hat es die Variable "Date", welche ein Kalenderdatum enthält

```
:
print(data.Date.head())
```

```
0    3/12/2016
1    4/02/2016
2    4/03/2017
3    4/03/2017
4    4/06/2016
```

- ohne besondere Vorkehrungen werden Timestamps als String (object) Type in Pandas Dataframes eingelesen
- im Folgenden wird dargestellt, wie solche Daten in den Typ `datetime` umgewandelt und dann einzelne Komponenten wie Jahr, Monat, Tag etc. extrahiert werden können

Konstruktion

- in einem ersten Schritt wird die Variable in den Typ `datetime` konvertiert, zu Demonstrationszwecken gleich in eine neue Variable "date_dt"

```
data['date_dt'] = pd.to_datetime(data.Date, format="%d/%m/%Y")
```

- daraus können anschliessend einzelne Komponenten wie Tag, Monat und Jahr extrahiert werden, auch hier als neue Variablen im Data Frame

```
data['year'] = data.date_dt.dt.year
data['month'] = data.date_dt.dt.month
data['day'] = data.date_dt.dt.day
```

- als weitere Möglichkeit wird hier noch die Differenz zu einem Startdatum (1.1.2016) ermittelt und im Data Frame hinterlegt

```
start_date = pd.to_datetime('1/1/2016', format="%d/%m/%Y")
data['daydiff'] = (data.date_dt - start_date).dt.days
```

Konstruktion

- das Ergebnis:

```
print(data[['Date', 'date_dt', 'day', 'month', 'year', 'daydiff']].head())
```

```
        Date    date_dt  day  month  year  daydiff
0  3/12/2016 2016-12-03    3     12  2016      337
1  4/02/2016 2016-02-04    4      2  2016       34
2  4/03/2017 2017-03-04    4      3  2017      428
3  4/03/2017 2017-03-04    4      3  2017      428
4  4/06/2016 2016-06-04    4      6  2016      155
```

- neben den hier gezeigten können weitere Komponenten extrahiert werden (falls vorhanden) wie hour, minute, second, etc.
  vgl. Online Ref: https://pandas.pydata.org/docs/reference/api/pandas.Series.dt.date.html
- ausserdem können mit der entsprechenden Parametrisierung Variablen mit Datum bereits beim Einlesen mit `.read_csv()` konvertiert werden (vgl. [ipynb])

Implementation

- beispielhaft zusammengestellt die Modifikationen, ausgehend von den Rohdaten des Bankkundendataset
- die einzelnen Schritte sind in den folgenden Blöcken organisiert:
  1. Modifikationen auf Data Frame
  2. Modifikationen kategoriale Variablen
  3. Modifikationen numerische Variablen
  4. Abschlussarbeiten inkl. Speichern des modifizierten Data Frames als CSV-File unter neuem Namen

Implementation

- das "Drehbuch" (Vorschlag)

Data Frame		
E1	Entfernen von Beobachtungen nach Bedingung	age > 100
E2	Entfernen von Duplikaten	
E3	Entfernen fragwürdiger Variablen	default, poutcome (ev. duration)
E4	Einsetzen von Werten für NAs	alle numerische: Median
		alle nicht numerische: Modalwert
Kategoriale Variablen		
E5	Reduzieren der Kardinalität	education: illiterate -> basic.4y
	Nummerisieren - Faktorisieren	hier keine
E6	Nummerisieren - Ordinal Encodieren	education, day_of_week, month
E7	Nummerisieren - Binär Encodieren	housing: no -> 0, sonst 1
		contact: cellular -> 1, sonst 0 (rename)
E8	Nummerisieren - Nominal Encodieren	alle jetzt noch nicht numerischen ausser y

Implementation

- das "Drehbuch" (Fortsetzung)

Numerische Variablen		
E9	logarithmieren	duration, campaign
E10	binär umcodieren	pdays: 999 -> 0, sonst 1
		previous: 0 -> 1 sonst 0
Andere Tätigkeiten		
	Konstruktion	unterbleibt hier
E11	Bereinigen der Variablennamen	
	Standardisieren	unterbleibt hier
E12	Speichern	als bank_data_prep.csv mit sep=',' (default)

- konkrete Implementierung: FE_6_Implementation.ipynb

Implementation Workshop 03

Workshop 03 Zeit: 60 - 90'

- in Workshop 2 haben Sie das Melbourne Housing Dataset mit Sicht auf Supervised Learning untersucht und dabei erste Empfehlungen für Feature Transformation erarbeitet

- eine Zusammenstellung konsolidierter Empfehlungen finden Sie in WS 03 Empfehlungen.xlsx

- empfohlenes Vorgehen:
  - erstellen Sie eine Kopie des Notebooks zu Kap. 1.6 (1.6 Feature Engineering - Implementation.ipynb)
  - modifizieren Sie dies mit Sicht auf die neue Fragestellung
  - erstellen Sie vom Ergebnis ein neues Dataset unter einem anderen Namen, z.B. melb_data_prep.csv