Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

2024 FS CAS PML
1 Feature Engineering
1.7 Nachträge

Werner Dähler 2024

1 Feature Engineering - AGENDA

1. Feature Engineering
   11. Einführung
   12. Exploration
   13. Transformation
   14. Konstruktion
   15. Selektion
   16. Implementation
   17. Nachträge
      171. NA Imputation mit ML Methoden
      172. Automatische Feature Selektion mit ML Methoden
      173. Feature Selektion mit PCA
2. Klassifikation
3. Regression
4. Validierung und Mehr

1.7 Feature Engineering - Nachträge

1.7.1 NA Imputation mit ML Methoden

- Wiederholung zum Umgang mit Missing Values (NAs, Kapitel 1.3.1.3): die Optionen
  - entfernen von Beobachtungen mit NAs
  - entfernen von Features mit NAs
  - einsetzen (NA Imputation)
    - einsetzen eines willkürlichen Wertes
    - einsetzen eines errechneten Wertes (Mean, Median, Modalwert)
    - **einsetzen eines (mittels ML) geschätzten wahrscheinlichsten Wertes**
- dabei bieten sich, je nach Skalierung des betreffenden Features folgende Methodengruppen an:
  - metrisch: Regression
  - kategorial: Klassifikation

1.7 Feature Engineering - Nachträge

1.7.1 NA Imputation mit ML Methoden

1.7.1.1 Metrisch mit Regression: "age"

- das bisherige Vorgehen zur prädiktiven Modellierung wird dazu etwas modifiziert
- um beispielsweise für "age" gültige Werte für NAs zu ermitteln, kann wie folgt vorgegangen werden
  - laden der Rohdaten bank_data.csv und entfernen ungünstiger Variablen ("default", "pdays", "poutcome")
  - entfernen aller Beobachtungen mit NAs, ausser bei "age"
  - One-Hot Encoding auf allen nicht numerischen Variablen
  - train - test - split
    - train: alle Instanzen für welche age nicht NAn ist
    - test: alle übrigen Instanzen
  - features - target - split: das Target ist dabei die Variable, welche NAs aufweist und wo diese eingesetzt werden sollen (hier also "age")

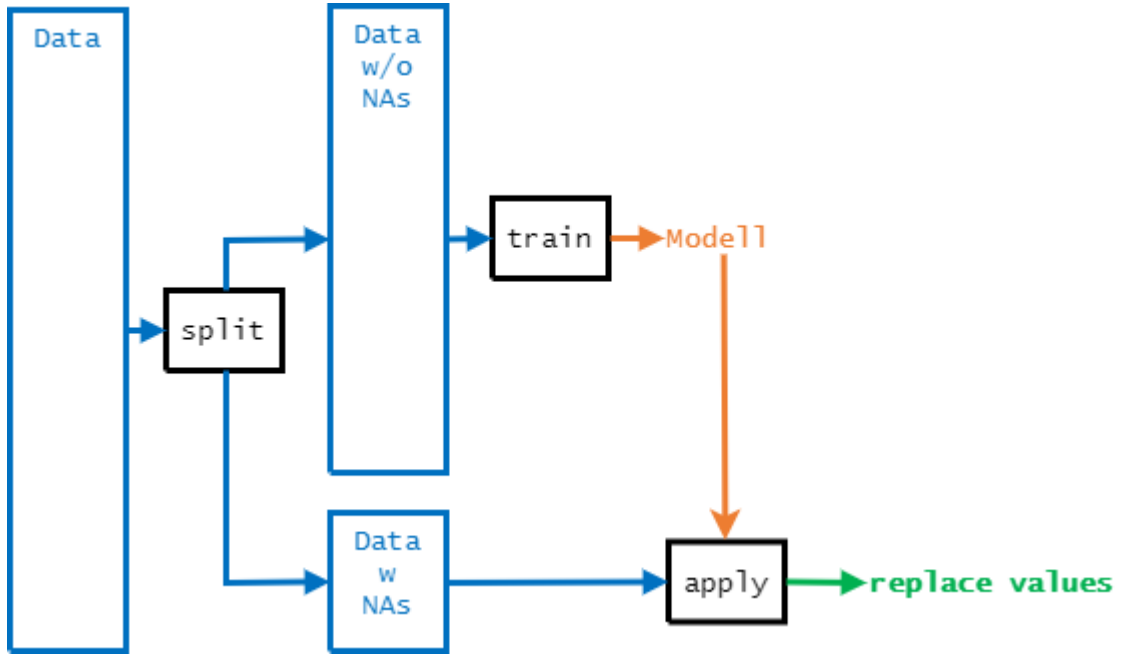
1.7 Feature Engineering - Nachträge

1.7.1 NA Imputation mit ML Methoden

1.7.1.1 Metrisch mit Regression: "age"

- Trainieren einer Linearen Regression und Vorhersagen der Werte für die Instanzen mit NAs
- Rekombinieren des Data Frame aus den Komponenten (X_train, y_train, X_test und y_test)

(vgl. [ipynb])



1.7 Feature Engineering - Nachträge

1.7.1 NA Imputation mit ML Methoden

1.7.1.1 Metrisch mit Regression: "age"

- durch das anschliessende Rekombinieren werden die Instanzen, für welche NAs ermittelt und eingesetzt wurden, am Ende des Data Frame angehängt
- zu Demozwecken wurde vorgängig noch ein Index hinterlegt, welcher die Rows mit NAs in "age" markiert

```
na_idx = data.age.isna()
print(data.age[na_idx].head())
```

```
7     NaN
20    NaN
34    NaN
101   NaN
327   NaN
```

```
print(new_data.age[na_idx].head())
```

```
7      62.270798
20     38.102409
34     41.409065
101    44.357099
327    39.134023
```

1.7 Feature Engineering - Nachträge

1.7.1 NA Imputation mit ML Methoden

1.7.1.2 Kategorial mit Klassifikation: "marital"

- analoges Vorgehen mit folgenden Modifikationen: ([ipynb])
  - One-Hot Encoding ebenfalls auf allen nicht numerischen Variablen, ausser "marital", welche hier die Rolle des Targets spielen wird
- eine Sichtung der Situation vor und nach Einsetzen zeigt untenstehende Ergebnisse

```
na_idx = data.marital.isna()
print(data.marital[na_idx].head())
```

```
553     NaN
1582    NaN
1698    NaN
1801    NaN
2274    NaN
```

```
print(new_data.marital[na_idx].head())
```

```
553     married
1582    married
1698    married
1801     single
2274     single
```

1.7 Feature Engineering - Nachträge

1.7.1 NA Imputation mit ML Methoden

1.7.1.3 sklearn.impute.KNNImputer

- die Klasse KNNImputer bietet Imputation zum Auffüllen fehlender Werte unter Verwendung des k-Nächste-Nachbarn-Ansatzes
- standardmässig wird eine euklidische Distanzmetrik, die fehlende Werte unterstützt, nan_euclidean_distances, verwendet, um die nächsten Nachbarn zu finden
- jeder fehlende Wert wird mit Hilfe der Werte der nächsten Nachbarn, die einen Wert für die Variable haben, eingesetzt (nur numerische Features!)
- die Anwendung folgt der gängigen scikit-learn API (vgl. [ipynb])

```
from sklearn.impute import KNNImputer
imp = KNNImputer()
imp.fit(data)
new_data = pd.DataFrame(imp.transform(data), columns=data.columns)
```

  - die Transformation des Ergebnisses als Data Frame ist hier angebracht, da Ergebnis von `.transform()` ein `ndarray` ist

1.7 Feature Engineering - Nachträge

1.7.1 NA Imputation mit ML Methoden

1.7.1.4 sklearn.impute.IterativeImputer

- relativ neu, daher noch als "experimental" bezeichnet
- eine Strategie zur Imputation fehlender Werte durch Modellierung jedes Merkmals mit fehlenden Werten als Funktion anderer Merkmale in einer Round-Robin-Methode
- bei jedem Schritt wird eine Merkmalsspalte als Target y bezeichnet und die anderen Merkmalsspalten werden als Features X behandelt
- ein Regressor wird an (X, y) für bekanntes y angepasst
- dann wird der Klassifikator verwendet, um die fehlenden Werte von y vorherzusagen
- dies wird für jedes Merkmal in einer iterativen Weise durchgeführt und dann für max_iter Imputationsrunden wiederholt
- die Ergebnisse der letzten Imputationsrunde werden zurückgegeben
- vgl. [ipynb]

1.7 Feature Engineering - Nachträge

1.7.2 Automatische Feature Selektion mit ML Methoden

1.7.2.1 Modellbasierte Feature Selection - 1.7.2.1.1 Klassifikationsmodelle

- wie in Kapitel 2.2.1.7 gesehen, benötigen einige Klassifikatoren (insbesondere regelbasierte Methoden) Feature Importance intern zur Modellbildung
- diese Information kann danach konsolidiert aus dem trainierten Modell extrahiert werden
- um maximale Information zur Importance zu gewinnen, sind gegenüber dem bisherigen Vorgehen zwei Punkte zu beachten
  - die Modelle werden **nicht** auf gesplitteten Daten gebildet (es sollen ja keine Modelle evaluiert werden)
  - allfällig vorangehendes One-Hot Encoding ist mit dem Parameter drop_first=False (Default) durchzuführen, um alle möglichen Ausprägungen berücksichtigen zu können
- aus den ermittelten Importances kann anschliessend beispielsweise eine Liste erstellt werden, um auf die gewünschte Anzahl Features zu filtern

1.7 Feature Engineering - Nachträge

1.7.2 Automatische Feature Selektion mit ML Methoden

1.7.2.1 Modellbasierte Feature Selection - 1.7.2.1.1 Klassifikationsmodelle

- Schritte des in [ipynb] hinterlegten Codes
  - laden der Rohdaten
  - minimales Feature Engineering
  - kein Train - Test - Split!
  - One-Hot Encoding auf den Features
  - trainieren eines Modells mit RandomForestClassifier
  - extrahieren von .feature_importances_ und mit Feature-Namen zusammenkombinieren in einem Data Frame best, welcher nach Importances abnehmend sortiert ist

	feature	importance
1	duration	0.220319
3	pdays	0.087773
8	euribor3m	0.087118
9	nr.employed	0.068184
0	age	0.057070
7	cons.conf.idx	0.053401

1.7 Feature Engineering - Nachträge

1.7.2 Automatische Feature Selektion mit ML Methoden

1.7.2.1 Modellbasierte Feature Selection - 1.7.2.1.1 Klassifikationsmodelle

- aus diesem Data Frame kann anschliessend ein Filter `sel_vars` erstellt werden, um auf die besten Features einzuschränken

```
sel_vars = best.head(6).feature.tolist()
new_X = X[sel_vars]
print(new_X.info())
```

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   duration        1832 non-null   float64
 1   pdays           1832 non-null   int64
 2   euribor3m       1832 non-null   float64
 3   nr.employed     1832 non-null   float64
 4   age             1832 non-null   float64
 5   cons.conf.idx   1832 non-null   float64
```

1.7 Feature Engineering - Nachträge

1.7.2 Automatische Feature Selektion mit ML Methoden

1.7.2.1 Modellbasierte Feature Selection - 1.7.2.1.2 Regressionsmodelle

- für die Beurteilung der Wichtigkeit der Features kann bei regelbasierten Methoden gleich vorgegangen werden wie bei Klassifikationsmodellen
- bei OLS Methoden, insbesondere Lasso und Ridge kann über ein Tuning des Parameters alpha auf die Wichtigkeit der einzelnen Features geschlossen werden (vgl. Kap. 3.2.2)
- bei Lasso Regression schränkt der Regularisierungsparameter alpha die Koeffizienten ein
- je grösser alpha, umso mehr Koeffizienten erhalten den Wert 0
- man kann davon ausgehen, dass bei kleinem alpha die Koeffizienten weniger wichtiger Features auf 0 gesetzt werden, bei kontinuierlichem Erhöhen von alpha dann jene mehr und mehr wichtiger Features

1.7 Feature Engineering - Nachträge

1.7.2 Automatische Feature Selektion mit ML Methoden

1.7.2.1 Modellbasierte Feature Selection - 1.7.2.1.2 Regressionsmodelle

- zur untenstehenden Visualisierung (vgl. [ipynb])
  - minimales Feature Engineering auf den Rohdaten
  - Features - Target - Split
  - kein Train - Test - Split (hier nicht notwendig)
  - Iteration über einen Bereich von alpha, wobei die Sequenz logarithmisch erzeugt wird
  - in jedem Iterationsschritt wird
  - das Modell trainiert
  - die ermittelten Koeffizienten in einer Liste hinterlegt
  - danach werden die Koeffizienten den Werten von alpha gegenübergestellt

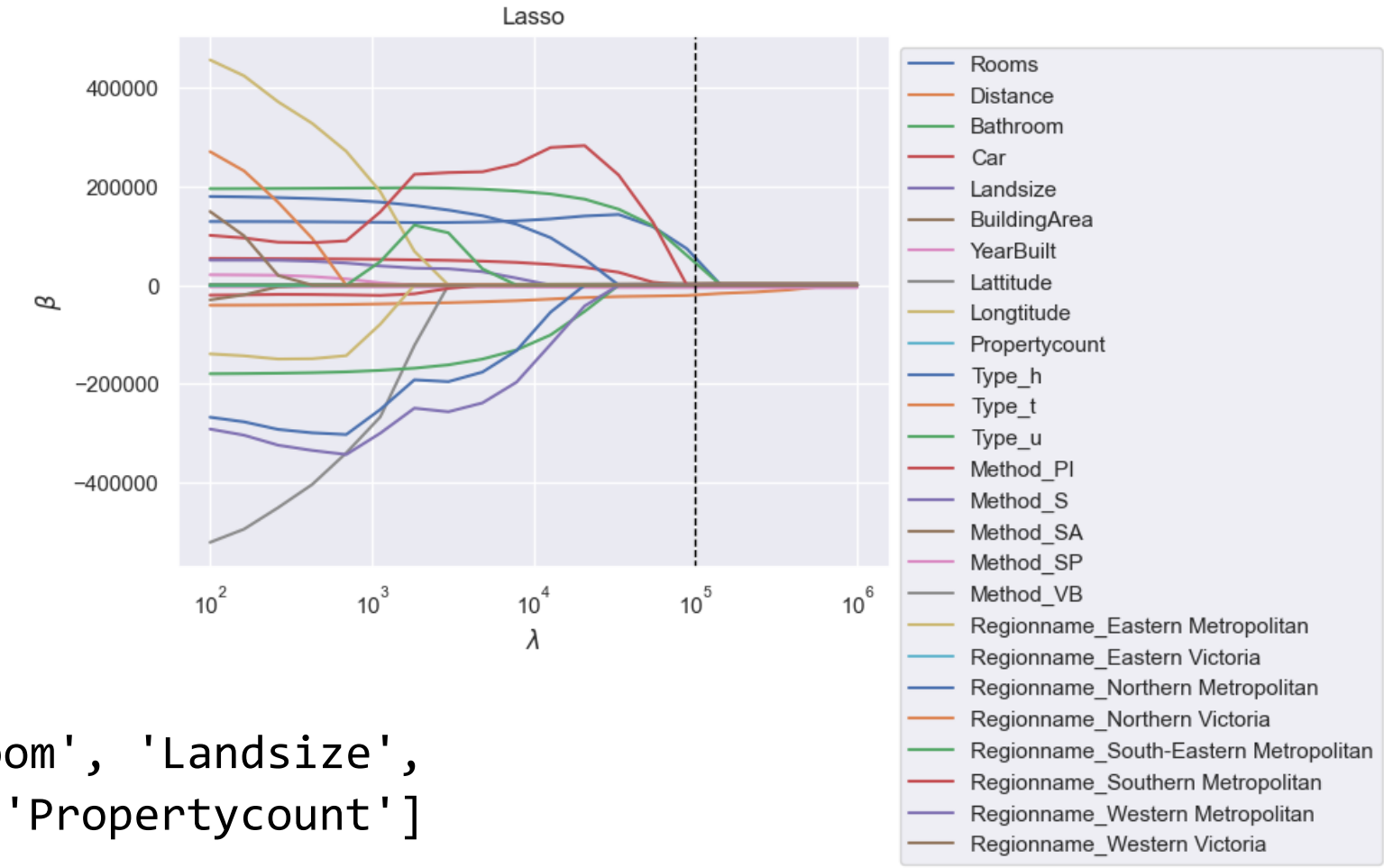
1.7 Feature Engineering - Nachträge

1.7.2 Automatische Feature Selektion mit ML Methoden

1.7.2.1 Modellbasierte Feature Selection - 1.7.2.1.2 Regressionsmodelle

- es wird ersichtlich, wie die Koeffizienten der Features mit Erhöhen von alpha nach und nach auf 0 gesetzt werden

- ein Schitt z.B. an der Stelle $alpha$=10^5 schränkt auf die folgenden 7 Features ein




```
['Rooms', 'Distance', 'Bathroom', 'Landsize',
'BuildingArea', 'YearBuilt', 'Propertycount']
```

1.7 Feature Engineering - Nachträge

1.7.2 Automatische Feature Selektion mit ML Methoden

1.7.2.1 Modellbasierte Feature Selection - 1.7.2.1.2 Regressionsmodelle

- dabei werden die Namen der Features wiederum in einer Liste hinterlegt, welche als Filter eingesetzt werden kann

```
new_X = X[sel_vars]
new_X.info()
```

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Rooms           6830 non-null   int64
 1   Distance        6830 non-null   float64
 2   Bathroom        6830 non-null   float64
 3   Landsize        6830 non-null   float64
 4   BuildingArea    6830 non-null   float64
 5   YearBuilt       6830 non-null   float64
 6   Propertycount   6830 non-null   float64
```

1.7 Feature Engineering - Nachträge

1.7.2.2 Iterative Methoden

- die modellbasierten Methoden zur Feature Importance haben aber die Schwäche, dass sich wegen möglicher Interaktionen durch Entfernen einzelner Features die Präferenzen der übrigen neu ändern könnten
- zwei Methoden, welche diesem Umstand entgegenwirken und als Funktionen in sklearn zur Verfügung stehen:
  - sklearn.feature_selection.RFE
  - sklearn.inspection.permutation_importance

- Vorbereitung:
  - aufsetzen auf den Rohdaten
  - entfernen von NAs
  - One-Hot Encoding der Features mit drop_first=False
  - kein Train - Test - Split

1.7 Feature Engineering - Nachträge

1.7.2.2.1 Iterative Methoden - RFE

(Feature ranking with recursive feature elimination)

- basiert auf einem externen Learner, welcher für die Features einen Gewichtungswert zurückgibt, z.B.
  - Koeffizienten bei linearen Modellen
  - Feature Importance bei Regelbasierten Modellen
- Ziel von Recursive Feature Elimination (RFE) ist es, rekursiv die Auswirkung kleiner werdender Subsets von Features zu untersuchen
- Vorgehen
  - festlegen, wie viele Features ausgewählt werden sollen
  - zuerst wird ein Modell mit allen Features trainiert, danach iterativ das Feature mit dem jeweils kleinsten Gewicht entfernt
  - wiederholen, bis die geforderte Anzahl von Features erreicht ist

1.7 Feature Engineering - Nachträge

1.7.2.2.1 Iterative Methoden - RFE

- mit dem Parameter n_features_to_select von RFE kann eingestellt werden, für wie viele "beste" Features der Support auf True gesetzt werden soll
- letzterer kann danach zu Filtern verwendet werden
- wird der Param auf 1 gesetzt, werden die Ränge für alle Features sichtbar (vgl. [ipynb])

Fazit:

- bestes Feature auch hier duration, gefolgt von makroökonomischen Parametern (nr.employed und euribor3m)
- wie oben gezeigt (vgl. 1.7.2.1.1), kann auch hier mit einem Performance Vergleich der Verlust an Vorhersagekraft beurteilt werden

	Feature	Ranking	Support
1	duration	1	True
9	nr.employed	2	False
8	euribor3m	3	False
0	age	4	False
3	pdays	5	False
7	cons.conf.idx	6	False

- die Ergebnisse von RFE können verwendet werden, um eine Liste der Feature-Namen zum Filtern zu erstellen ([ipynb])

1.7 Feature Engineering - Nachträge

1.7.2.2.2 Iterative Methoden - permutation_importance

- RFE (vgl. 1.7.2.2.1) bedingt einen externen Learner, welcher Gewichte der einzelnen Features zurückgibt
- permutation_importance umgeht diese "Schwäche" indem die einzelnen Features unabhängig voneinander durch mehrmaliges Permutieren neutralisiert und danach der Scorewert mit einer Baseline verglichen wird
- das Verfahren kann mit jedem beliebigen Learner (Klassifikator oder Regressor) eingesetzt werden
- zum Vorgehen
  - trainieren eines Modells mit den vorbereiteten Daten (vgl. 1.7.2.2.1)
  - ermitteln eines ersten Score Wertes als **Baseline**
  - für jedes Feature werden die Werte n mal (parameter n_repeats, default=5) zufällig durchmischt (permutiert) und danach der Score Wert unter Anwendung auf das ursprüngliche Modell erneut ermittelt
  - je mehr sich der Score Wert gegenüber der Baseline verschlechtert, umso wichtiger erscheint das betreffende Feature

1.7 Feature Engineering - Nachträge

1.7.2.2.2 Iterative Methoden - permutation_importance

- als Ergebnis liefert die Methode folgende Attribute
  - importances_mean: Mittewert der Importance für jedes Feature
  - importances_std: Standardabweichung der Importance für jedes Feature
  - importances: Einzelwerte für jede Iteration mit jedem Feature
- die Ergebnisse können danach mit geeigneten Methoden visualisiert werden (vgl. [ipynb])

	feature	mean	std
1	duration	0.137555	0.006486
3	pdays	0.042031	0.002264
8	euribor3m	0.022380	0.004200
0	age	0.013100	0.001505
2	campaign	0.007642	0.000913

- daraus kann wiederum eine Liste zum Filtern der Features erstellt werden (vgl. [ipynb])

1.7 Feature Engineering - Nachträge

1.7.3 Feature Selektion mit PCA - EXTRA

1.7.3.1 Korrelationen mit erster Hauptkomponente

- bei den bisher gezeigten Selektionsmethoden wurde jeweils der Einfluss der Features auf das Target bewertet
  d.h. sie sind insbesondere geeignet mit Sicht auf Überwachtes Lernen
- Feature Importance kann allerdings auch mit Sicht auf Unüberwachtes Lernen von Interesse sein
- in Kap. 1.4.2, Feature Konstruktion - Dimensionsreduktion mit PCA wurde die Hauptkomponentenanalyse vorgestellt
- diese transformiert die Ausgangsdaten durch Rotation im multidimensionalen Raum so um, dass die Maximale Varianz auf der ersten transformierten Dimension (1. Hauptkomponente) abgebildet wird
- in einem anschliessenden Vergleich kann mit Korrelationskoeffizienten jedes Ausgangsfeatures mit dieser ersten Hauptkomponente der Grad der Repräsentativität der einzelnen Features für den gesamten Datensatz quantifiziert werden
- im Codebeispiel wird das Target 'y' binär numerisch umcodiert, um es wie ein gewöhnliches Feature zu behandeln

1.7 Feature Engineering - Nachträge

1.7.3 Feature Selektion mit PCA - EXTRA

1.7.3.1 Korrelationen mit erster Hauptkomponente

- Vorgehen (vgl. [ipynb])
  - laden der Rohdaten mit minimalem Feature Engineering
  - binär Umcodieren des Targets ("yes" -> 1, "no" -> 0)
  - One-Hot Encoding
  - Standardisieren
  - PCA
  - hinzufügen von PC1 an vorbereitete Daten
  - berechnen der Korrelationsmatrix (Absolutwerte)
  - Korrelationsmatrix nach PC1 abnehmend sortieren
  - Zeile für PC1 entfernen
  - Spalte PC1 anzeigen (z.B. erste 6)
- das Ergebnis kann bedarfsweise wiederum als Liste zum Filtern der Features verwendet werden

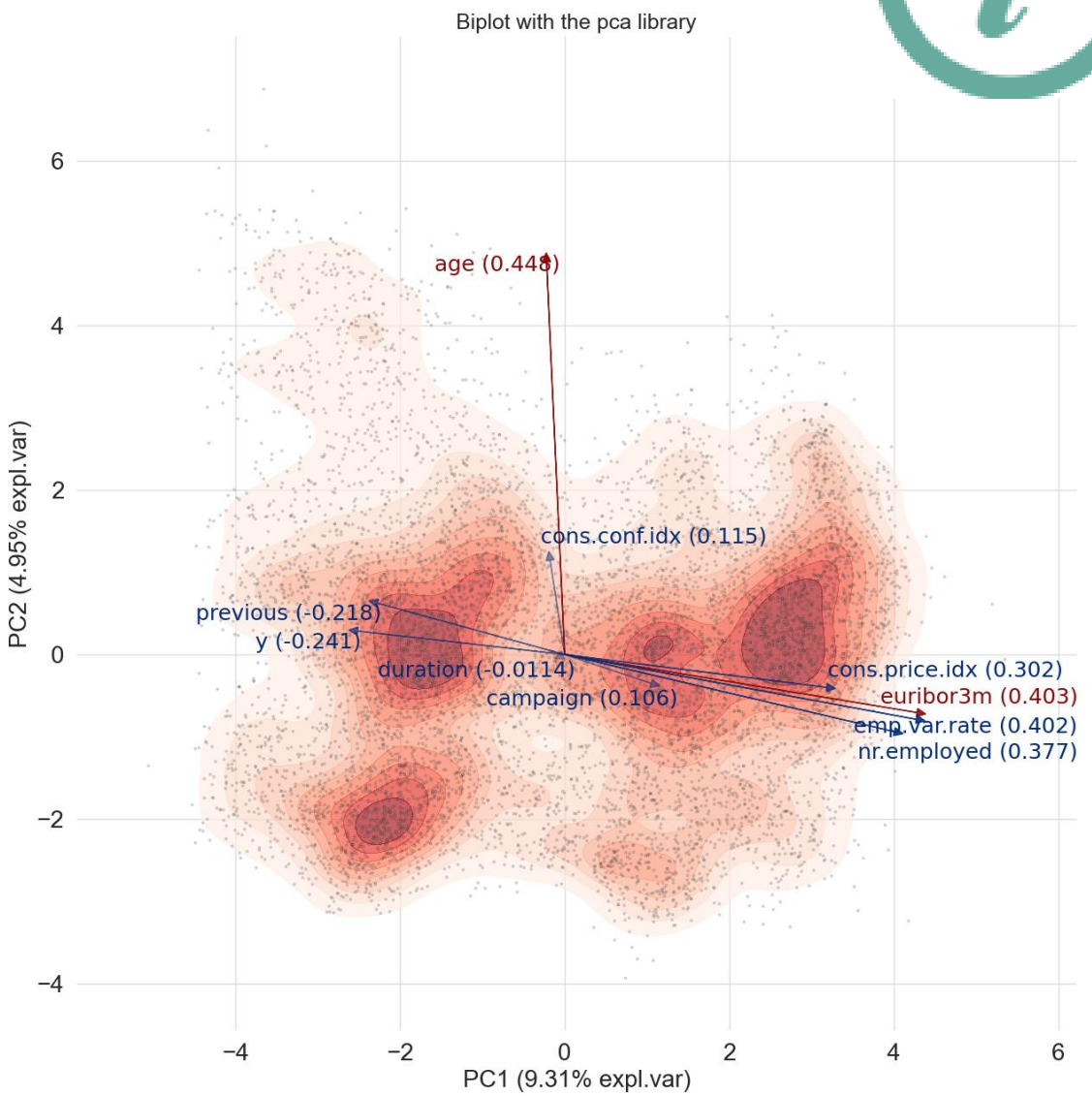
```
euribor3m          0.902998
emp.var.rate       0.901408
nr.employed        0.846323
cons.price.idx     0.677562
contact_cellular   0.660235
contact_telephone  0.660235
```

1.7 Feature Engineering - Nachträge

1.7.3 Feature Selektion mit PCA - EXTRA

1.7.3.2 Biplot aus der Library pca

- ein Biplot ist ein erweitertes Streudiagramm, das sowohl Punkte als auch Vektoren zur Darstellung der Struktur verwendet
- vgl. [ipynb]



1.7 Feature Engineering - Nachträge

Workshop 11

Gruppen zu 2 bis 4, Zeit: 45'

- ermitteln Sie die Importance der Features der Rohdaten von melb_data.csv unter Einsatz von `sklearn.inspection.permutation_importance`
- setzen Sie dazu minimales Feature Engineering wie folgt ein:
  - entfernen fragwürdiger Variablen: 'Unnamed: 0', 'Suburb', 'Address', 'SellerG', 'Postcode', 'Bedroom2', 'Date', 'CouncilArea'
  - One-Hot encoding aller verbleibenden kategorialen Variablen (der Parameter `dummy_na=True` von `pd.get_dummies()` erstellt auch Dummy-Variablen für NAs)
  - einsetzen von geschätzten Werten für NAs in verbleibenden numerischen Variablen mit `sklearn.impute.KNNImputer`

1.7 Feature Engineering - Nachträge

- danach:
  - features - target - split
  - kein train - test - split
  - ermitteln der Importance unter Einsatz von
  - `sklearn.inspection.permutation_importance`
  - `sklearn.tree.DecisionTreeRegressor`
  - tabellarische und graphische Darstellung der Ergebnisse