„AutoForecast“

PROGNOSEMODELLE AUTOMATISIERT ERZEUGEN

BIANCA HUBER | BIANCA.HUBER@MARMELADENBAUM.DE

Überblick

Über mich

Motivation

Kernidee

Anwendungsfälle

Diskussion

Überblick

Über mich

Motivation

Kernidee

Anwendungsfälle

Diskussion

- Meine Idee für Programmkonzeption vorstellen und Vorteile aufzeigen.
  Diskussion, Feedback und Inspiration für die Konzeption und technische Umsetzung.

Über mich

Akademisch

- Bachelorstudium in International Management
- Zusatzqualifikation Ingenieurwesen
- Informatik Track Business Analytics Master

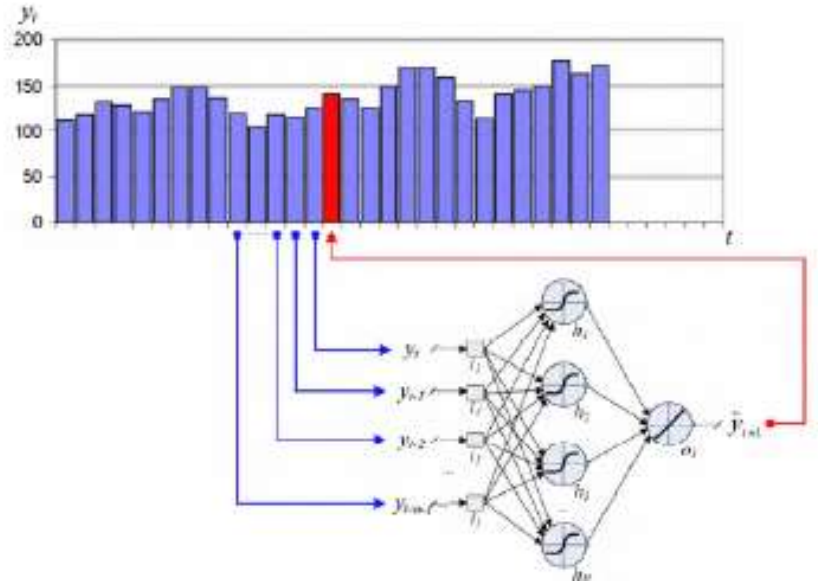
Projekterfahrung

- Process Modelling in Java
- Backtestingtool in VBA
- Prognosekorrektur in R
- Schulungen Python mit Schwerpunkt Deep Learning
- Textzerlegung in Python
- Erzexploration in R
- Zeitreihenprognose in Python
- …

Motivation

Usecase I

- Mehrere Time Lagged Neural Nets für
- Unterschiedliche Inputvektoren mit
- Kreuzvalidierung evaluieren.



*Crone, S. F./Kourentzes, N. 2010: Naive Support Vector Regression and Multilayer Perceptron benchmarks for the 2010 neural network grand competition (NNGC) on time series prediction

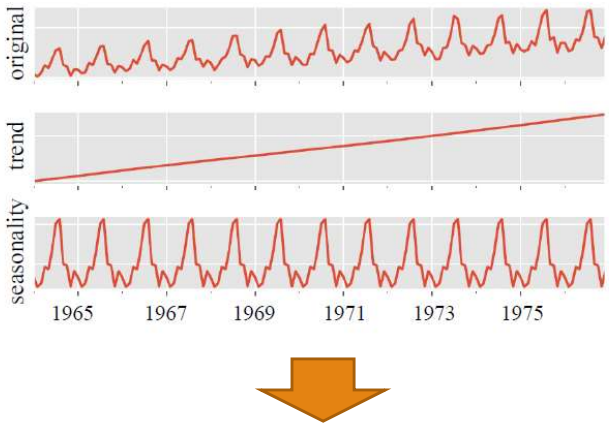
Motivation

Usecase I

- Mehrere Time Lagged Neural Nets für
- Unterschiedliche Inputvektoren mit
- Kreuzvalidierung evaluieren.

Usecase II

- Auf mehreren Aggregationsstufen und
- Mit rollierendem Zeitfenster
- Datenexploration durchführen und
- Geeignete Prognosemodelle ableiten.



Triple Exponential Smoothing

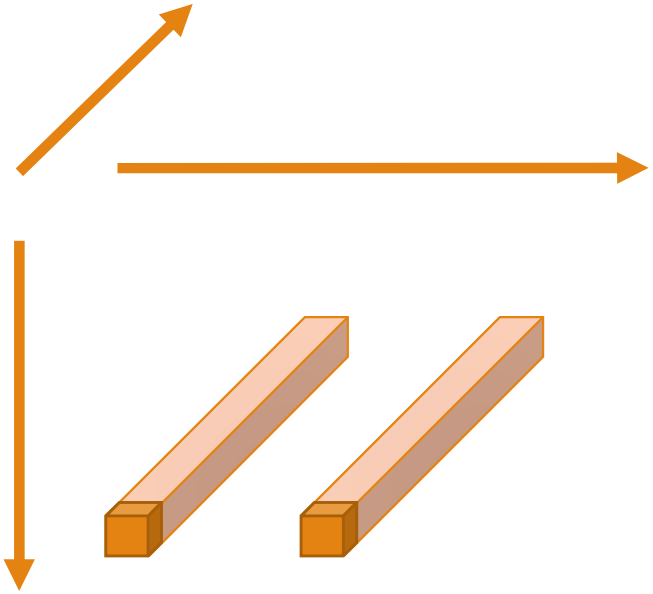
Motivation

Usecase I

- Mehrere Time Lagged Neural Nets für
- Unterschiedliche Inputvektoren mit
- Kreuzvalidierung evaluieren.

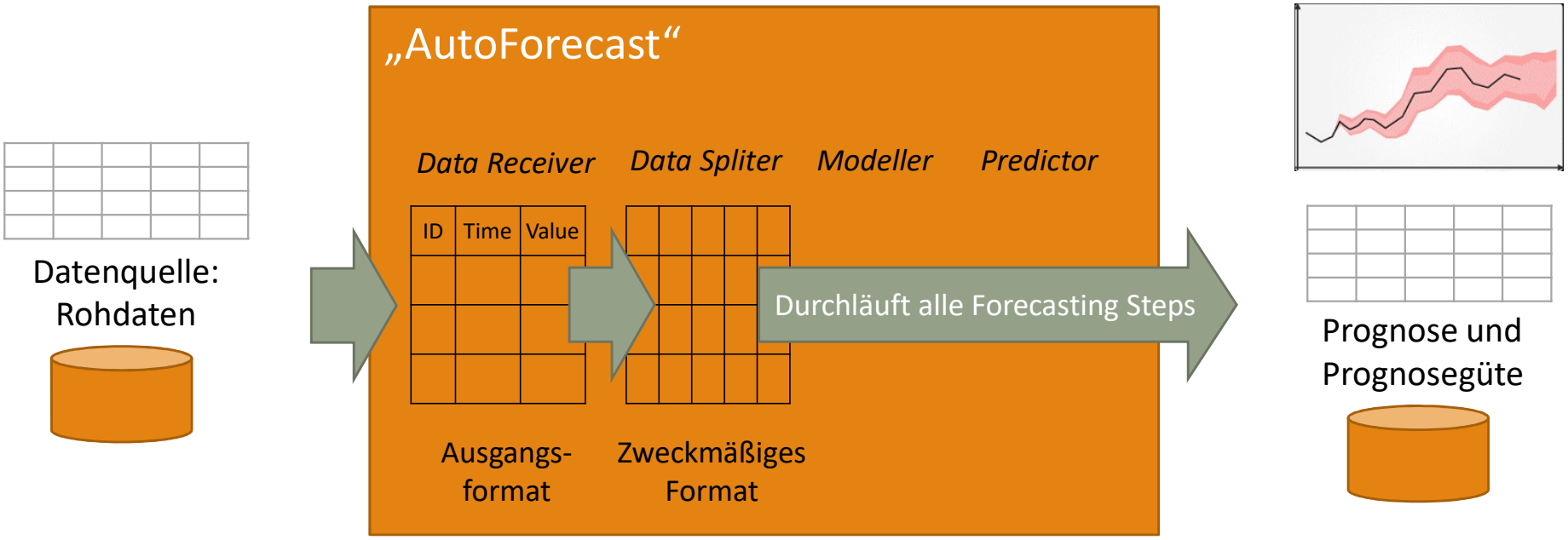
Usecase II

- Auf mehreren Aggregationsstufen und
- Mit rollierendem Zeitfenster
- Datenexploration durchführen und
- Geeignete Prognosemodelle ableiten.



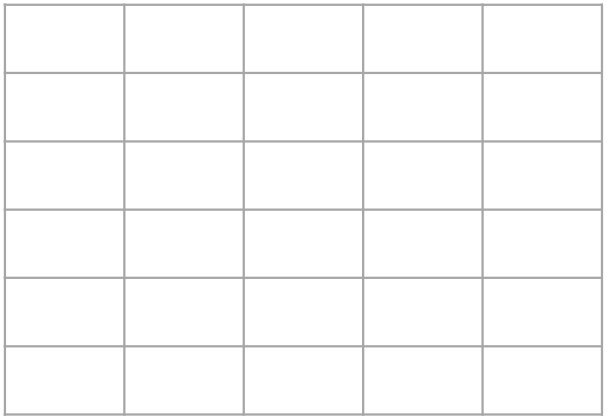
Mehrere Dimensionen sollen abgebildet werden.
Einzelne Bereiche des (diskreten) n-dimensionalen Raums sind zu vergleichen.

Einordnung


„AutoForecast“
Data Receiver
Data Spliter
Modeller
Predictor
ID
Time
Value
Durchläuft alle Forecasting Steps
Datenquelle:
Rohdaten
Ausgangs-
format
Zweckmäßiges
Format
Prognose und
Prognosegüte

Kernidee

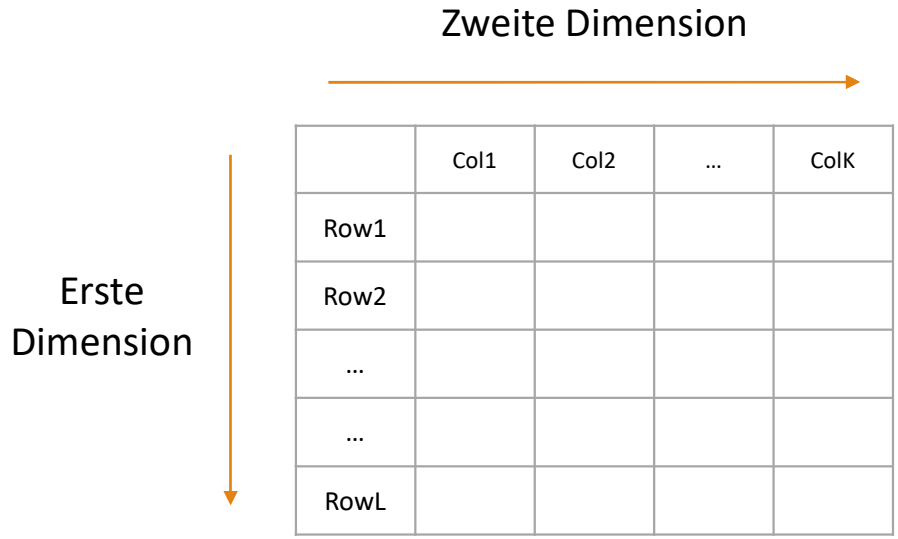
Dataframes der Pandas-Bibliothek nutzen.



Kernidee

Dataframes der Pandas-Bibliothek nutzen.

Zwei Dimensionen entlang der Zeilen und Spalten abbilden. Index und Spaltennamen nicht verändern.



Kernidee

Dataframes der Pandas-Bibliothek nutzen.

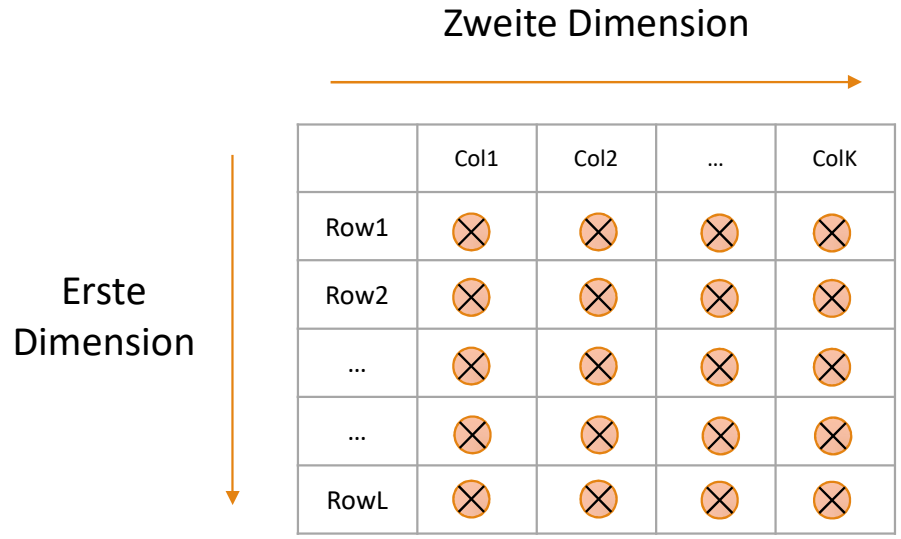
Zwei Dimensionen entlang der Zeilen und Spalten abbilden. Index und Spaltennamen nicht verändern.

Innerhalb der einzelnen Zellen können verschiedene 1…*n* - dimensionale Datentypen abgelegt werden.

- Floats,
- Arrays,
- Dictionaries,
- Modelle usw.

} 2 + n Dimensionen

Wichtig ist nur, dass alle vom gleichen Typ sind.



Kernidee

Dataframes der Pandas-Bibliothek nutzen.

Zwei Dimensionen entlang der Zeilen und Spalten abbilden. Index und Spaltennamen nicht verändern.

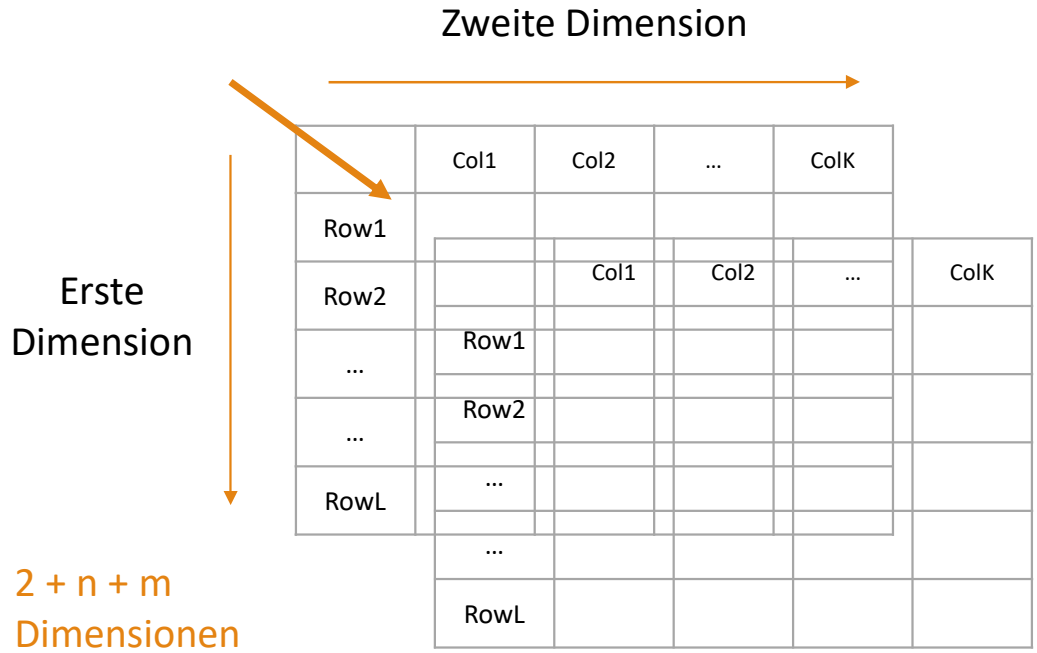
Innerhalb der einzelnen Zellen können verschiedene 1…*n* - dimensionale Datentypen abgelegt werden.

- Floats,
- Arrays,
- Dictionaries,
- Modelle usw.

Wichtig ist nur, dass alle vom gleichen Datentyp sind.

Weitere Dimensionen können auch durch *m* Dataframes mit der gleiche Struktur, aber anderen Zellformate hinzugefügt werden.

2 + n + m Dimensionen



Kernidee

Dataframes der Pandas-Bibliothek nutzen.

Zwei Dimensionen entlang der Zeilen und Spalten abbilden. Index und Spaltennamen nicht verändern.

Innerhalb der einzelnen Zellen können verschiedene 1…*n* - dimensionale Datentypen abgelegt werden.

- Floats,
- Arrays,
- Dictionaries,
- Modelle usw.

} 2 + n Dimensionen

Wichtig ist nur, dass alle vom gleichen Datentyp sind.

Weitere Dimensionen können auch durch *m* Dataframes mit der gleiche Struktur, aber anderen Zellformate hinzugefügt werden.

} 2 + n + m Dimensionen

} Je nach Fachlichkeit Dimensionen und Optionen wählen

Kernidee

Dataframes der Pandas-Bibliothek nutzen.

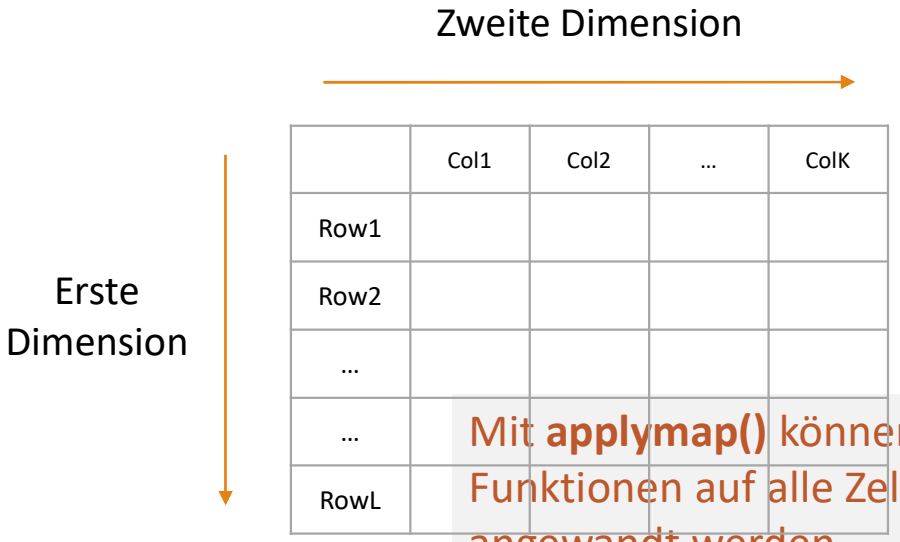
Zwei Dimensionen entlang der Zeilen und Spalten abbilden. Index und Spaltennamen nicht verändern.

Innerhalb der einzelnen Zellen können verschiedene 1…*n* - dimensionale Datentypen abgelegt werden.

- Floats,
- Arrays,
- Dictionaries,
- Modelle usw.

Wichtig ist nur, dass alle vom gleichen Datentyp sind.

Weitere Dimensionen können auch durch *m* Dataframes mit der gleiche Struktur, aber anderen Zellformate hinzugefügt werden.



Anwendungsbeispiele

Beispiel I: Originale Zeitreihen und Transformationen und Visualisierung in Data Receiver Klasse mit Funktionen wie *normalize_data(), logarithmize_data(),* etc. in *applymap*-Aufruf.

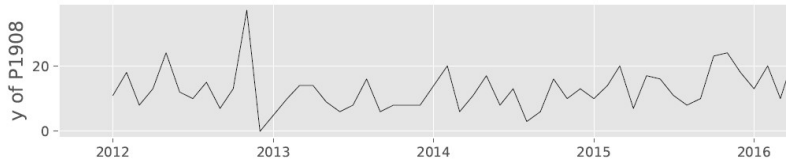
ID	time
1	[12.2012, 12.2013, 12.2014, 12.2015, 12.2015, 12.2016]
2	…
…	
L	

ID	original
1	[1, 2, 3, 4, 5, 6]
2	…
…	
…	
L	

ID	normalisiert
1	[0, 0.2, 0.4, 0.6, 0.8, 1]
2	
…	
…	
L	

…

plot_time_series(ID)



Intuitiv aufrufbar und schnell erweiterbar.

Anwendungsbeispiele

Beispiel II: Zerlegung in k-Fold Trainings- und Testdatentsätze in Data Splitter Klasse mit Funktionen wie *split(k)* in neue Dataframes

input_train,

ID	window1	…	windowK
1	[1, 2, 3]		[2, 3, 4]
2	…		
…			
L			

input_test

ID	window1	…	windowK
1	[4, 5]		[5, 6]
2	…		
…			
L			

(timeline_train…)

Flexibel und resultierende Datensätze überprüfbar.

Anwendungsbeispiele

Beispiel III: Erzeugung der Prognosemodelle in Modeller Klasse mit Kindklassen wie *SES*, *ARIMA* *und applymap*-Aufruf zur Modellgenerierung für alle gesplitteten Trainingsdatensätze.

models

ID	window1	…	windowK
1	ARIMA().fit()		ARIMA().fit()
2	…		
…			
L			

```
class Modeler(ABC):
    def __init__(self, DataSplitter):
        self.DataSplitter = DataSplitter

    def get_models(self, method_to_generate_model):
        self.models = self.DataSplitter.inputs_train.applymap(method_to_generate_model)
        self.has_models = True


class Modeler_ES(Modeler):

    def __init__(self, DataSplitter):
        super().__init__(DataSplitter)

        self.get_models(self.get_simple_exponential_smoothing_model)

    def get_simple_exponential_smoothing_model(self, input_train):
        self.typ_desc = 'Simple Exponential Smoothing'

        model = SimpleExpSmoothing(input_train).fit()
        return model
```

- Beliebige Modelle können als neue Kindklassen ergänzt werden.

Anwendungsbeispiele

Beispiel IV: Messung der Vorhersagegüte in Evaluator Klasse mit Error-Funktionen wie *MAPE() etc., die in der applymap*-Funktion auf zuvor erstellte Dictionaries angewandt wird.

Input dictionaries

ID	window1	…	windowK
1	{*true*: …, *pred*: …}		{*true*: …, *pred*: …}
2	…		
…			
L			

MAPE

ID	window1	…	windowK
1	5 %		4.9 %
2	…		
…			
L			

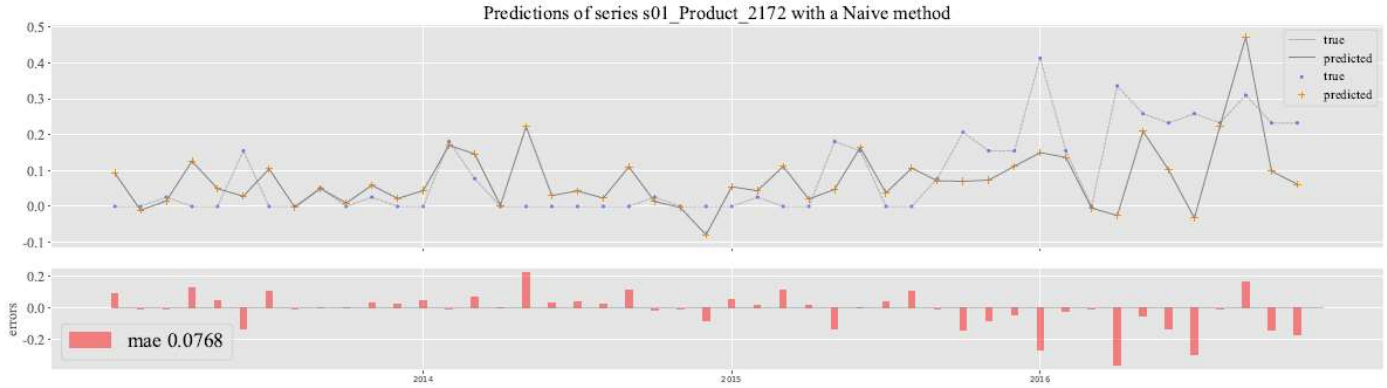
```
class Evaluator:
    def __init__(self, Predictor, DataSplitter):
        self.Predictor = Predictor
        self.DataSplitter = DataSplitter

    def create_df_for_evaluation(self):
        df_evaluate = pd.DataFrame(index = self.Predictor.predictions.index,
                                   columns = self.Predictor.predictions.columns)

        for row in df_evaluate.index:
            for col in df_evaluate.columns:
                dict_2d = {'true' : self.DataSplitter.input_test.loc[row, col],
                           'pred' : self.Predictor.predictions.loc[row, col]}
                df_evaluate.at[row, col] = dict_2d
        return df_evaluate

    def get_predictions(self, method_to_get_error):
        self.evaluations = self.evaluations.applymap(method_to_get_error)
```

- Fehler vergleichbar und um neue Error-Funktionen erweiterbar.

Anwendungsbeispiele

Beispiel V: Visualisierungen können basierend auf Dataframes aus Evaluator, DataSplitter und anderen Klassen erstellt werden und über Zeilen- und Spalten-Id aufgerufen werden.

plot_predictions_and_mae(ID, window)



Diskussion

Flexibel einsetzbar: einfache Erweiterung von einzelnen Funktionen/Features

Zwischenergebnisse zugänglich für Testzwecke

Intuitiv aufrufbar für Fachanwender

Konzept eignet sich gut für den iterativen Projektansatz: probieren - evaluieren – ändern/ergänzen

Schnittstelle für Erweiterbarkeit durch Import-/Exportfunktion im Table Style und Data Receiver.

Einzelne Schritte mit Zwischenergebnissen sind klar trennbar und setzen aufeinander auf: einzelne Elemente können weiterentwickelt werden ohne jedes Detail des Gesamtprogramms zu verstehen.

Auch für Fachanwender mit begrenzten Programmierkenntnissen ist die Wartung und Betriebsfähigkeit möglich: kleinere Features können ergänzt werden, wie z. B. neue Fehlerfunktionen.

…?

Vielen Dank!

Kontaktieren Sie mich gerne!

Bianca Huber

bianca.huber@marmeladenbaum.de