Method

|  |  |
| --- | --- |
| transformer\_agent.py | |
| Dient zur Erstellung einer Transformatorklasse.  Input Variablen:  Anzahl an Haushalte  Beispiel zur Benutzung:  transformer = Transformer(num\_households=100)  transformer.initialize\_transformer()  capacity = transformer.get\_max\_capacity()  print(capacity) | |
| Funktion | Beschreibung |
| set\_power\_household() | Diese Funktion berechnet die Leistung eines Hausanschlussus in Watt.  Die Berechnungsformel hierfür ist:  Volt \* Ampere \* Phasen  Wir gehen davon aus, dass für einen Hausanschluss die Spannung 230 Volt, die Stromstärke 62 Ampere beträgt und eine Phase verwendet wird.  In dieser Funktion können Anpassungen vorgenommen werden, falls die Hausanschlüsse eine höhere Leistung haben sollen. |
| set\_customers\_contracted\_power() | Hierbei wird die Anschlussleistung, die mit Hilfe von set\_power\_household berechnet wurde, auf kW umgerechnet.  Anschließend wird der Wert mit der Anzahl an Haushalten multipliziert.  In unserer Simulation gehen wir derzeit davon aus, dass jedes Haus ein Elektrofahrzeug besitzt. Deshalb ist die Anzahl an Haushalten auch die Anzahl an Elektrofahrzeugen. |
| get\_c\_diversity() | Bei der Benutzung von künstlichen Lastprofilen spielt der Diversityfaktor eine Rolle, dieser kann auch Gleichzeitigkeitsfaktor genannt werden. Dieser wird eingesetzt, um das gleichzeitige Auftreten der Last abzubilden.  Find source for diversity factor:  https://en.wikipedia.org/wiki/Diversity\_factor |
| set\_transformer\_power\_capacity() | Wir berechnen mit der Formel aus dem unten zitierten Paper die Kapazität des Transformators.  Einflussfaktoren auf die Kapazität ist die Summe an Leistungswerten von allen Kunden, der Diversityfaktor, eine Sicherheitsmarge und einer Übergrößenzahl. Da ein Transformator meist eine Lebensdauer von mehreren Jahrzehnten (Source?) hat, wird eine Sicherheitsmarge und Oversized Power Capacity eingeführt. Diese dienen zur Abbildung des zukünftigen Anstieges an Last, die über den Transformator fließt, ab.  https://www.sciencedirect.com/science/article/pii/S0960148117310649?via%3Dihub |
| initialize\_transformer() | Durch diese Funktion werden die vorherig definierten Funktionen set\_power\_household(), set\_customer\_conrtacted\_power() und set\_transformer\_power\_capacity() ausgeführt, um die Berechnung der Kapazität des Transformators in einem Schritt auszuführen. |
| get\_max\_capacity() | Durch diese Funktion wird die Kapazität der Transformatorklasse ausgegeben.  Diese Funktion aufzurufen, ist nur sinnvoll nachdem die Funktion initialize\_transformer() ausgeführt wurde, da davor im Kapazitätswert des Transformators ein None wert steht. |

|  |  |
| --- | --- |
| customer\_agent.py | |
| Dient zur Erstellung einer Stromkunden Klasse mit einem Standardlastprofil. Dieses soll ermöglichen eine Grundlast auf den Transformator abzubilden, die nicht mehr für das Laden von E-Fahrzeugen zur Verfügung steht.  Input Variablen:  Jährlicher Verbrauch Haushalt  Start Datum  End Datum  Beispiel zur Benutzung:  customer = PowerCustomer(yearly\_cons\_household=3500,  start\_date=start\_date,  end\_date=end\_date)  customer.initialize\_customer()  customer.set\_current\_load(timestamp)  print(customer.current\_load\_kw) | |
| Funktion | Beschreibung |
| create\_cleaned\_h0\_profile() | Lädt aus dem Ordner input, das File h0\_profile.csv. Diese Datei wird vom BDEW unter <https://www.bdew.de/energie/standardlastprofile-strom/> zur Verfügung gestellt. Hierbei handelt es sich um die durchschnittlichen Leistungswerte in kW eines Haushaltes in Deutschland, normiert auf 1000 kWh Jahresverbrauch, in einer Auflösung von 15 Minuten.  Anschließend wir das H0-Profil bearbeitet, sodass dieses in einer Spalte dargestellt wird, mit dem Zeitstempel als Index.  Hierbei wird das Jahr im Zeitstempel auf das Jahr des Start Datums gesetzt.  Anschließend wird das aufbereitete Standardlastprofil als Datei mit dem Namen cleaned\_h0\_profil.csv in dem input Ordner gespeichert. |
| set\_standard\_load\_profile() | Lädt das bearbeitete H0 Lastprofil aus dem Ordner input und weist es der Kundenklasse zu. |
| set\_scale() | Nimmt die Variable yearly\_cons\_household, die den Jahresverbrauch eines Haushaltes in kWh angibt. Beispielsweise 3500 kWh. Dieser Verbrauch wird anschließend durch 1000 geteilt, um den Faktor zu erhalten, der in der Funktion set\_scaled\_load\_profile() verwendet wird, zu berechnen. |
| set\_scaled\_load\_profile() | Um das Standardlastprofil, das auf 1000 kWh jährlich normiert ist, zu skalieren, wird der Skalierungsfaktor aus set\_scale genommen und das Standardlastprofil mit diesem Skalierungsfaktor multipliziert.  Das Ergebnis ist ein skaliertes Standardlastprofil. |
| initialize\_customer() | Diese Funktion wird benutzt um die anderen Funktionen, set\_standard\_load\_profile(), set\_scale() und set\_scaled\_load\_profile() aufzurufen.  Mit dem Aufruf initialize\_customer() werden die Lastgänge für einen Kunden berechnet. |
| set\_current\_load() | Diese Funktion hat als input parameter einen Zeitstempel. Die Funktion gibt den Lastwert in kW des eingegebenen Zeitstempels aus. Dieser Lastwert in kW wird aus dem skalierten Lastprofil genommen. |

|  |  |
| --- | --- |
| mobility\_data.py | |
| Wird benutzt, um eine Klasse mit dem Namen MobilityDataAggregator zu erstellen. Diese Klasse enthält Funktionen, um Mobilitätsprofile für die Fahrzeuge aufzubereiten. Unter die Aufbereitung fällt die Auswahl des richtigen Zeitraumes, die Aggregation auf 15-Minuten Zeitintervalle und das erstmalige Data Cleaning.  Input Variablen:  raw\_mobility\_data: pd.DataFrame,  start\_date: str,  end\_date: str  Beispiel zur Benutzung:  data\_aggregator = MobilityDataAggregator(raw\_mobility\_data=raw\_mobility\_data,  start\_date=self.start\_date,  end\_date=self.end\_date)  print(data\_aggregator.df\_processed) | |
| Funktion | Beschreibung |
| \_create\_df\_limited\_time() | Bekommt den input Dataframe der Mobilitätsdaten für ein Fahrzeug. Dieser Dataframe ist ein dask dataframe. Als erster wird überprüft, ob das Startdatum und das Enddatum außerhalb des Bereichs der Mobilitätsdaten liegen. Falls dies der Fall ist, wird eine Fehlermeldung ausgegeben.  Anschließend wird ein Filter auf die Mobilitätsdaten angewandt, sodass nur noch Mobilitätsdaten gespeichert werden, die für die Simulation benötigt werden. |
| \_aggregate\_15\_min\_steps() | Hierbei werden die reduzierten Mobilitätsdaten auf 15 Minuten Schritte aggregiert.  Bei der Aggregation wird ein Raster aus 15 minuten Werten gebildet und die Werte bis zum nächsten 15 Minuten Zeitschritt wie angegeben aggregiert. Der Zeitstempel am Anfang jedes 15 Minuten Intervalls wird in das resampelte Ergebnis einbezogen.  Aggregationen finden wie folgt statt:   * ECONSUMPTION wird die Summe gebildet, da Gesamtwerte für die 15 Minuten gebraucht werden. * TRIPNUMBER wird der minimale Wert genommen. Tripnummern sind eine fortlaufende aufsteigende Zahl innerhalb eines Mobilitätsprofils. Falls das Auto geparkt wird, endet ein Trip und der nächste Beginnt. Das heißt wenn sich die Tripnummer innerhalb der 15 Minuten erhöht, wird die Tripnummer auf den kleinsten Wert gesetzt. * ID\_PANELSESSION wird auf das Maximum gesetzt. Hierbei schwankt der Wert zwischen 0=ignition, 1=driving und 2=engine turn-off. Falls innerhalb eines 15-Minuten Intervalls nun der Wert von ignition zu driving wechselt, wird der Wert auf driving gesetzt, und somit die gesamte Viertelstunde als driving bezeichnet. Falls es von driving auf engine turn-off wechselt, wird die gesamte Viertelstunde auf engine turn-off gesetzt. Hierbei kann immer noch ECONSUMPTION anfallen. Wenn innerhalb von einer Viertelstunde alle drei Panelsessions auftauchen, wird die Viertelstunde auf engine turn-off gesetzt. * ID\_TERMINAL ist in jeder Aggregation das gleiche, da dies die Car\_ID darstellt. Wird in der Aggregation auf den ersten vorkommenden Wert gesetzt. * CLUSTER schwankt zwischen 0=everywhere else, 1=home und 2=work. Gibt an, an welcher Stelle sich das Auto befindet. Wird auf das Minimum gesetzt. Dies hat zur Folge, dass das Auto noch mit einer 0 gelabeled wird, obwohl es bereits in dem 15 Minuten Intervall daheim angekommen ist. Das gleiche gilt auch bei work. Ein Auto muss sich deshalb mindestens 15 Minuten an einem Ort aufgehalten haben, um ein entsprechendes Label zu bekommen. * DELTAPOS bezeichnet die Positionsveränderung seit der letzten Messung in m. Hierbei wird während der Aggregation die Summe gebildet, da die absolute zurückgelegte Distanz innerhalb der 15 Minuten von Interesse ist. |
| \_data\_cleaning() | Im Datacleaning werden hauptsächlich zwei Schritte durchgeführt. Zum einen werden NAN Werte in den Spalten TRIPNUMBER und ECONSUMPTION aus den Mobilitätsdaten entfernt. Zum anderen werden die timestamps in den Mobilitätsdaten auf das richtige Format überprüft. Schlägt das Entfernen der NAN Werte oder das Überprüfen des Zeitstempels fehlt, werden die Mobilitätsdaten keinem Auto zuegordnet und es wird versucht einen neuen Agenten mit neuen Mobilitätsdaten zu erstellen. Dies geschieht, da sowohl ECONSUMPTION als auch die Zeitstempel eine besonders wichtige Rolle in der nachfolgenden Simulation spielen. |
| prepare\_mobility\_data() | Ruft die Funktionen \_create\_df\_limited\_time(),\_aggregate\_15\_min\_steps() und \_data\_cleaning() auf. |
| set\_median\_trip\_len() | Berechnet für die hinzugefügten Mobilitätsdaten die Länge eines Trips im Median. |

Beschreibung Mobilitätsdaten:

Es werden von diesen Spalten nur 7 Spalten benutzt. Um das Laden der Mobilitätsdaten schneller zu machen, werden diese Spalten in der Funktion zum Laden spezifiziert. Um die Geschwindigkeit des Ladens weiterhin zu erhöhen wird in der Ladefunktion bereits der Datentyp jeder Spalte definiert. Bei den Spalten handelt es sich um folgende Spalten mit dem dazugehörigen Datentypen {'TIMESTAMP': str, 'TRIPNUMBER': int, 'DELTAPOS': float, 'CLUSTER': int, 'ECONSUMPTIONKWH': float, 'ID\_PANELSESSION': int, 'ID\_TERMINAL': int}

Da der Zeitstempel in String Format gespeichert wird, wird dieser nachträglich im timestamp Format geparst.

909 Dateien mit Mobilitätsprofilen.