API Documentation

API Documentation

December 4, 2014

Contents

1 Module BilbasenDataMining.bilbasen

Bilbasen module

This module handles all communication with bilbasen.dk, and it contains functions to communicate with bilbasen.dk as well as functions to download data from bilbasen.dk. The main part of the module runs the method download_data_to_database which is the most important method of the module, as this is where bilbasen.dk is crawled and the data is parsed and saved in a database.

1.1 Functions

connect()

This method returns a httplib connection to bilbasen.dk

extract_car_info(*listing_type*, *listing*)

This method is used by the method download_data_to_database() and extracts data from a html listing. The data extracted is all the attributes on a car given on a search result site, which is: model, link to detailed page, description, how many km it has done, which year it is from, how many horsepowers it has, how many km per liter it does, how fast it goes from 0-100 km/t, what the monthly cost is, if it can pull a trailer, and a location. The method saves this data in the namedtupe Car and returns that tuple.

get_date()

This method returns the current date

get_car_image_src(*link*)

Given a link to a car description page, this method returns a link to an image of the car.

create_car_brand_table(*conn*)

This method crawls bilbasen.dk for a list of car brands, and creates and stores these brands in a table called 'Brands' in the database. The method is called each time the method download_data_to_database is called.

insert_car_to_table(*car*, *tablename*, *cursor*)

This method takes as input a namedtuple Car, a name of a table in the database and a cursor to the database, and inserts the car into the given table.

download_data_to_database(*limit*=`None`)

This method is the most important one of this module, as this is the one crawling bilbasen.dk to extract data of cars currently on sale. The method creates a new table each time it is called, and the table will be named 'AllBrandsDDMMYY', where DDMMYY is the current date. If the table already exist, it will be deleted. The method takes an optional parameter, defining how many pages to crawl. If no parameter is given, it crawls all pages (i.e. all cars) on bilbasen.dk

main()

Main method which will run by a command 'python bilbasen.py arg', where arg is the number of pages to crawl. Do not provide any argument if all pages should be crawled.

1.2 Variables

Name	Description
__package__	**Value:** `'BilbasenDataMining'`

1.3 Class Car

```
object ─┐
        │
     tuple ─┐
            │
          BilbasenDataMining.bilbasen.Car
```

Car(model, link, description, kms, year, hk, kml, kmt, moth, trailer, location, price)

1.3.1 Methods

__getnewargs__(*self*)

Return self as a plain tuple. Used by copy and pickle.

Overrides: tuple.__getnewargs__

__getstate__(*self*)

Exclude the OrderedDict from pickling

__new__(*_cls*, *model*, *link*, *description*, *kms*, *year*, *hk*, *kml*, *kmt*, *moth*, *trailer*, *location*, *price*)

Create new instance of Car(model, link, description, kms, year, hk, kml, kmt, moth, trailer, location, price)

Return Value

a new object with type S, a subtype of T

Overrides: object.__new__

__repr__(*self*)

Return a nicely formatted representation string

Overrides: object.__repr__

Inherited from tuple

__add__(), __contains__(), __eq__(), __ge__(), __getattribute__(), __getitem__(), __getslice__(), __gt__(), __hash__(), __iter__(), __le__(), __len__(), __lt__(), __mul__(), __ne__(), __rmul__(), __sizeof__(), count(), index()

Inherited from object

__delattr__(), __format__(), __init__(), __reduce__(), __reduce_ex__(), __setattr__(), __str__(), __subclasshook__()

1.3.2 Properties

Name	Description
description	Alias for field number 2
hk	Alias for field number 5
kml	Alias for field number 6
kms	Alias for field number 3
kmt	Alias for field number 7
link	Alias for field number 1
location	Alias for field number 10
model	Alias for field number 0
moth	Alias for field number 8
price	Alias for field number 11
trailer	Alias for field number 9
year	Alias for field number 4
Inherited from object __class__	

2 Module BilbasenDataMining.database

Database module

This module contains convenient methods to communicate with the database.

2.1 Functions

connect_to_database()

Returns a MySQLdb cursor and connection to the database

check_if_table_exist(*cursor*, *table*)

Checks if the specified table exist in the database

delete_table(*cursor*, *table*)

Deletes the specified table from the database

query(*cursor*, *query*)

Executes the specified query and returns the retrieved data

commit(*cursor*, *connection*)

Commits any changes to the database - this function should be used after inserting and deleting

get_newest_table()

Returns the newest AllCars-table in the database by using the fact that the tables are named after the date they were generated (i.e. the date the data was download from bilbasen.dk)

get_car_brands()

returns a pandas series of all the brands

get_locations(*table*)

returns a pandas Series of all the sale locations present in the specified table

create_test_table()

Creates a test table to have specific data to test on

2.2 Variables

Name	Description
__package__	**Value:** `'BilbasenDataMining'`

3 Module BilbasenDataMining.datamining

Datamining module

This module contains all the data mining methods.

3.1 Functions

get_distribution_all_brands(*tablename*)

Calculates a frequency distribution of all car brands in the given table and returns a panda dataframe of the distribution with brand-names in the first column, the number a brand appears in the table in the second column and the corresponding percentage in the last column

get_distribution_one_brand(*tablename*, *brand*)

Calculates a frequency distribution of a particular car brand in the given table in terms of the different models for that brand. Returns a panda dataframe of the distribution with model-names in the first column, the number a model appears in the table in the second column, and the corresponding percentage in the last column

create_distribution(*series*, *columns*, *n_models*)

Calculates a frequency distribution (using the nltk-package) of a given panda series object. Returns a panda dataframe object containing the frequency distribution with the values of the given series as the first column, the number of time a certain value appear in the given series in the second column, and the corresponding percentage in the third column. The DataFrame object returned will have labelled columns according to the given columns list.

get_location_distribution_all_brands(*table*)

Calculates the distribution of all brands based on sale locations in the specified table. A pandas Series with the distribution and the locations as index is returned.

get_location_distribution_one_brand(*table*, *brand*)

Calculates the distribution of the specified brand based on sale locations in the specified table. A pandas Series with the distribution and the locations as index is returned.

extract_brands(*models*)

Extracts and returns the exact brand names from the given list of models, i.e. from the list [”Audi A8”, ”Mercedes SLK”], the list [”Audi”, ”Mercedes”] will be extracted and returned

simplify_model_names(*models*)

Simplifies the car model names in the given list of models, i.e. removes irrelevant information in the names, number of doors (’4d’), size of engine (’2.0L’) etc.

get_fastest_cars(*table*)

Finds the fastest car(s) in the given table (i.e. the car(s) that goes fastest from 0-100 km/t). The result is returned as a pandas DataFrame containing the entire database row of that car.

get_cheapest_cars(*table*)

Finds the cheapest car(s) in the given table, that is not 0DKK and is not a leasing car. The result is returned as a pandas DataFrame containing the entire database row of that car.

get_most_expensive_cars(*table*)

Finds the most expensive car(s) in the given table. The result is returned as a pandas DataFrame containing the entire database row of that car.

get_most_ecofriendly_cars(*table*)

Finds the most eco friendly car(s) in the given table, i.e. the car(s) that goes most KMs on a liter petrol. The result is returned as a pandas DataFrame containing the entire database row of that car.

analyze_description(*description*)

A very simple sentiment analysis function Assigns a score to a sentence, based on how many positive or negative words appear in that sentence. If there are no negative or positive words, the final score will be 0. If there are more positive than negative words, the score will be positive - how positive will depend on how many positive / negative words. As it it most likely that there will be many positive words compared to negative words, a negative word is counted twice, as the score should reflect a clear difference between a car description containing only positive words, and a car description containing words like ”buler” and ”dårligt”, which clearly is a worse car.

calculate_best_offer(*table*, *model*)

This method calculates the best offer for a given car model. The calculation is based on creating a linear regression model of the attributes: car description, mileage and age, with prices as y-values. The description of the car is analyzed to get a rank value, using the modules analyze_description method. Based on the linear regression model, a prediction of the prices is calculated, and the differences between the predicted prices and the actual prices are calculated to find the biggest difference (the largest negative value), which is the best offer. The returned result is a pandas DataFrame along with the difference.

3.2 Variables

Name	Description
__package__	**Value:** `'BilbasenDataMining'`

4 Module BilbasenDataMining.graphics

Graphics module

This module contains methods to create plots of different types to visualize the data and results from the data mining methods.

4.1 Functions

create_distribution_map(*distribution*)

Given a distribution as a pandas Series with indexes as locations and values as the number of cars at that location, this method creates a map of Denmark using Basemap, and plots a scatter plot on top of the map. Each scatter will correspond to a location, and its size will correspond to the number of cars at that location.

create_price_km_scatter(*table*, *brand*)

This method creates a scatter plot showing the coherence between prices and mileage (in km's).

create_price_year_scatter(*table*, *brand*)

This method creates a scatter plot showing the coherence between prices and age (in production year).

create_distribution_plot(*table*, *n_bars*, *plot_name*)

This method creates a bar plot showing the distribution of the data in the table, which has to be a pandas DataFrame, and saves it with the given name. The plot will only include the specified number of bars (n_bars), as otherwise it can become very crowded.

create_pie_plot(*distribution*, *plot_name*)

This method creates a pie chart of a given distribution, provided as a pandas DataFrame, and saves it with the given name.

4.2 Variables

Name	Description
__package__	**Value:** `'BilbasenDataMining'`

5 Module BilbasenDataMining.html

html module

This module contains convenient methods to create html representations of various data types.

5.1 Functions

create_HTMLtable_from_series(*series*, *listofheaders*)

This method creates a html table from a Pandas Series, with headers as provided in the listofheaders.

create_car_representation(*cars*, *attribute*)

This method creates a html div containing specific information, which depends on the given attribute. It also downloads a picture of the car to represent in the div from bilbasen.dk

5.2 Variables

Name	Description
__package__	**Value:** `'BilbasenDataMining'`

6 Module BilbasenDataMining.main

```
Main script

This is the main script of the module and is used to run the cherrypy
web application - run the web application by 'python main.py'
- the web application will run on 127.0.0.1 port 8888

It uses jinja2 to generate HTML from templates specified in the templates
folder.
It consist of a class MiningBilbasen, which contains methods for each site
of the web application (specifying their URLs).
Each method returns the HTML of the site, which is used by cherrypy
to present the site.
```

6.1 Functions

generate_index_page()

This method generates the index page of the web application and uses the jinja2 template 'index.html' to present it.

6.2 Variables

Name	Description
env	**Value:** `Environment(loader= PackageLoader('main', 'templates'))`
newest_table	**Value:** `'AllCars031214'`
__package__	**Value:** `'BilbasenDataMining'`

6.3 Class MiningBilbasen

6.3.1 Methods

index(*self*)

distributions(*self*)

This method generates the distributions page and uses the jinja2 template 'distribution_template.html' to generate the html.

distribution_of_car_brand(*self*, *brand*)

This method generates the distributions_of_car_brand page and uses the jinja2 template 'distribution_template.html' to generate the html.

location_distributions(*self*, *brand*=`'all brands'`)

This method generates the location_distributions page and uses the jinja2 template 'location_distribution_template.html' to generate the html.

price_km_year_coherence(*self*, *brand*=`'all brands'`)

This method generates the price_km_year_coherence page and uses the jinja2 template 'price_km_year_coherence_template.html' to generate the html.

best_offer(*self*, *model*=`None`)

This method generates the best_offer page and uses the jinja2 template 'best_offer_template.html' to generate the html.

Index