Linear Regression Example

The example below uses a marketing dataset, in order to illustrate a linear regression activity.

Workflow:

1. Preparation
   A. Load the dataset from a `.csv` file and show a short description
   B. Show the two dimensional scatter plots for all the predicting variables with respect to the target
   C. Split the data into *predicting variables* `X` and *target* `y`
      a. here we set the `random_state` variable to make the experiment *repeatable*
2. First experiment: compute the regression on a single predicting variable
   A. Consider a reduced dataset containing the chosen variable and the target
   B. Fit the `LinearRegression` estimator on the training set
   C. Show the statistical significance of the fitted model
   D. Predict the target for the test set using the *fitted* estimator
   E. Compute the regression coefficients and the quality measures: *Root Mean Squared Error (RMSE)* and *coefficient of determination (r2)*
3. Second experiment: compute the regression considering all the predicting variables
   A. Repeat the steps from 2.2 to 2.5
4. Third experiment: use the `DecisionTreeRegressor` with the entire dataset
   A. Fit the tree using the default hyperparameters, in order to find the maximum depth of the unconstrained tree
   B. Use *cross-validation* to find the optimal *maximum depth* of the tree
   C. Fit the tree with the optmal `max_depth`
   D. Predict and show the *root mean squared error*
5. Fourth experiment: use the `RandomForestRegressor`
   A. Repeat steps from 4.2 to 4.4 (for simplicity, we use the maximum `max_depth` found in 4.1)

In [ ]:
```
# Code source: Claudio Sartori
# License: BSD 3 clause
```

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
from sklearn.model_selection import train_test_split

random_state = 94922767 # this will be used to guarantee the repeatability of the experiment
```

Load the dataset from a `.xlsx` file and show a short description

In [ ]:
```
# This cell allows full compatibility between execution in Google Colab and in local
try:
  import google.colab.files
  IN_COLAB = True
except:
  IN_COLAB = False
# from google.colab import files
if IN_COLAB:
    uploaded = files.upload()
```

In [ ]:
```
# The file must be available in the same directory,
# or uploaded in the Colab environment
# in the execution of the previous cell
data_fn = 'FoodUK2014.xlsx'
df0 = pd.read_excel(data_fn)
```

Data Exploration and preparation

Show a short description of the columns

In [ ]:
```
```

Out[ ]:

	hhsize	quarter	adults_n	children_n	totalexp	SexHRP	month	Gorx	Year	income	AgeHRP	qmeat	c
count	5114.000000	5114	5114.000000	5114.000000	5114.000000	5114	5114	5114	5114.0	5114.000000	5114.000000	4873.000000	3542.00
unique	NaN	4	NaN	NaN	NaN	2	12	12	NaN	NaN	NaN	NaN	
top	NaN	April to June	NaN	NaN	NaN	Male	February	South East	NaN	NaN	NaN	NaN	
freq	NaN	1341	NaN	NaN	NaN	3050	445	736	NaN	NaN	NaN	NaN	
mean	2.363707	NaN	1.841807	0.521901	519.898868	NaN	NaN	NaN	2014.0	679.542002	53.802698	10.475023	2.14
std	1.244704	NaN	0.743052	0.945622	411.543093	NaN	NaN	NaN	0.0	499.596175	16.187912	8.798118	2.03
min	1.000000	NaN	0.000000	0.000000	-246.916821	NaN	NaN	NaN	2014.0	0.000000	17.000000	0.086667	0.10
25%	1.000000	NaN	1.000000	0.000000	260.598783	NaN	NaN	NaN	2014.0	306.954000	41.000000	4.452500	0.86
50%	2.000000	NaN	2.000000	0.000000	426.977227	NaN	NaN	NaN	2014.0	548.086000	54.000000	8.374167	1.62
75%	3.000000	NaN	2.000000	1.000000	651.003763	NaN	NaN	NaN	2014.0	925.652500	67.000000	14.005333	2.81
max	9.000000	NaN	7.000000	7.000000	5859.877186	NaN	NaN	NaN	2014.0	2134.090000	80.000000	104.589333	41.34

Show the number of rows with nulls

It is computed subtracting the number of rows in the dataset without nulls from the original number of rows

In [ ]:

Out[ ]:

```
1668
```

Drop rows with nulls

In [ ]:

```
After dropping rows with nulls the dataset has 3446 rows
```

Data transformation

- Convert the alphanumeric SexHRP into numeric 0 and 1

- the `sklearn` machine learning procedures work only with numeric predicting attributes
- Generate two new columns as ratio of other columns
  - this is suggested by background information

In [ ]:

Use only the columns that the experts consider interesting

This is suggested by background information

In [ ]:

Out[ ]:

	adults_n	children_n	SexHRP	AgeHRP	qmeat_hhsize_ratio	income_hhsize_ratio	uvmeat
1	2	2	1	38	1.511250	206.130000	8.813621
2	2	0	1	54	5.890083	135.962500	7.965790
4	3	0	1	64	4.285667	165.346667	5.726323
5	2	2	1	70	8.968250	66.632500	8.451528
7	3	0	1	64	4.079111	134.393333	5.904745

Choose the target and split the data into *predicting variables* `X` and *target* `y`

In [ ]:

Show the two dimensional scatter plots for all the predicting variables with respect to the target

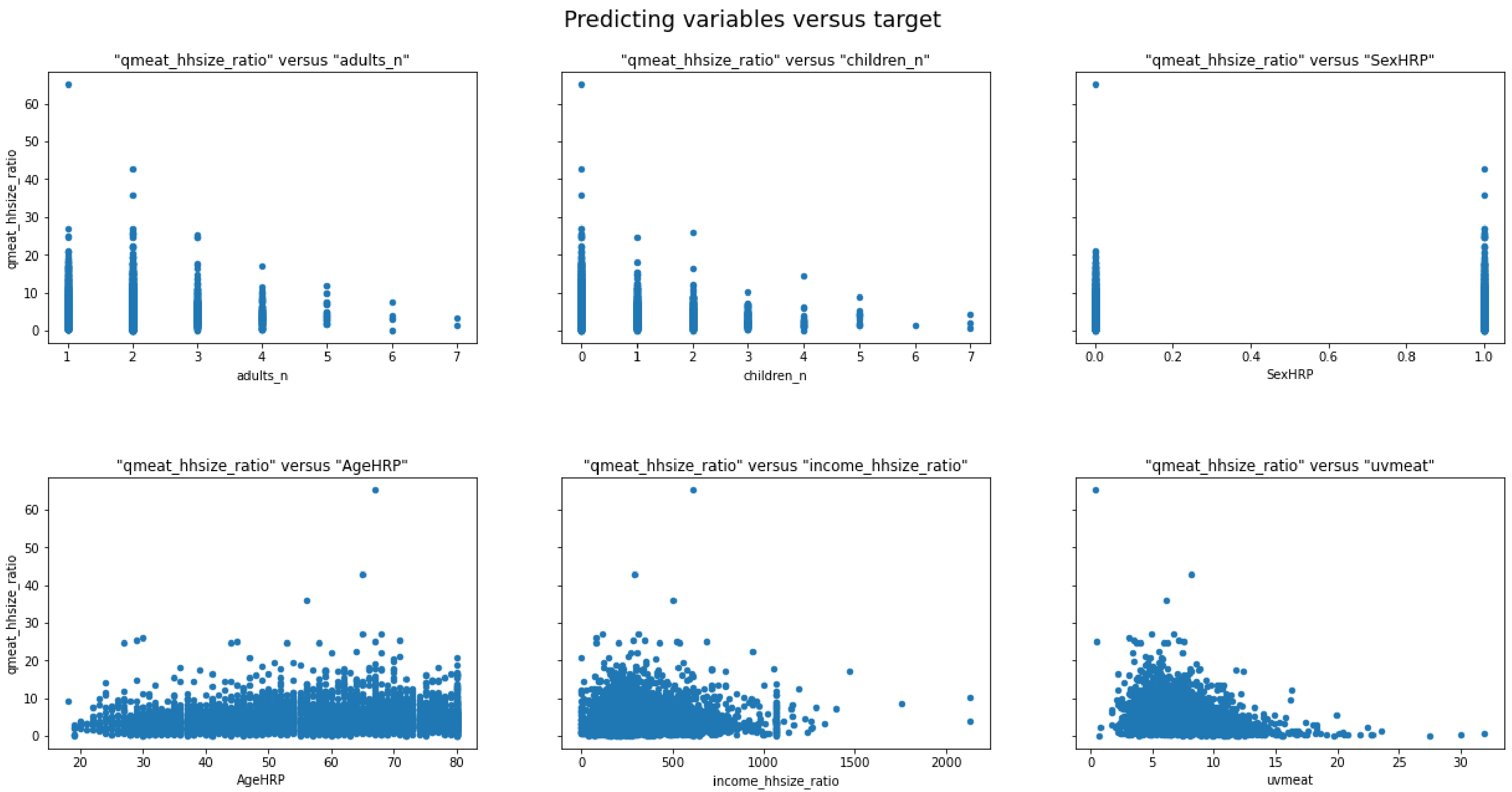
In [ ]:

```
ncols=3
import math
nrows = math.ceil((df.shape[1]-1)/ncols)
figwidth = ncols * 7
figheigth = nrows*5
```

In [ ]:

```

```



Show the *p-values* of the target with respect to the variables

In [ ]:

```python
from sklearn.feature_selection import f_regression
# Your code here
```

Out[ ]:

	Variable	p-value
0	adults_n	1.415945e-05
1	children_n	1.077386e-30
2	SexHRP	8.429827e-02
3	AgeHRP	1.710126e-21
4	income_hhsize_ratio	1.211099e-03
5	uvmeat	4.789746e-52

Split the data into *train* and *test* and show the sizes of the two parts

Here we set the `random_state` variable to make the experiment *repeatable*

In [ ]:

```
```

```
Training set and test set have 2412 and 1034 elements respectively
```

Consider a reduced dataset containing the chosen variable and the target

In [ ]:

```
```

Fit the `linear_model` estimator on the training set and predict the target for the test set using the *fitted* estimator

In [ ]:

```
```

Compute the regression coefficients and the quality measures

Create a function to compute the F-statistic and p-value of the regression model

In [ ]:

```
# Computation of F-statistic and p-value for the regression
# http://facweb.cs.depaul.edu/sjost/csc423/documents/f-test-reg.htm
```

Compute the statistical significance of the model

In [ ]:

Out[ ]:

	Univariate Linear - Value
Intercept for "adults_n"	5.646984
Coefficient for "adults_n"	-0.326893
rmse	3.886323
r2	0.007595
f-statistic	9.841162
p-value	0.001727

Second experiment: compute the regression considering all the predicting variables

Now we use the entire data in `X_train` and `X_test` for fitting and predicting

In [ ]:

Fit, predict and show the results

Now we see the *regression coefficients* resulting from the fitting.

In particular, *positive coefficients* indicate that the target *increases* with the variable, *negative coefficients* indicate a *decreasing* trend.

The absolute values of the coefficient cannot be considered directly a measure of importance, due to the possibly different orders of magnitude of the data in the different columns (observe above the outputs of `describe`).

In [ ]:

Out[ ]:

	Variable	Coefficient
0	adults_n	-0.318682
1	children_n	-0.650924
2	SexHRP	0.383162
3	AgeHRP	0.014913
4	income_hhsize_ratio	0.000989
5	uvmeat	-0.392620

Compute the statistical significance

In [ ]:

```
```

Out[ ]:

	Variable	p-value
0	adults_n	4.812836e-19
1	children_n	1.384315e-188
2	SexHRP	5.806116e-10
3	AgeHRP	3.199609e-119
4	income_hhsize_ratio	6.460728e-08
5	uvmeat	0.000000e+00

Compute the quality measures

In [ ]:

```
```

Out[ ]:

	Univariate Linear - Value
rmse	3.6651
r2	0.1173
f-statistic	57.4179
p-value	0.0000

Decision Tree Multivariate Regresson

In [ ]:
```
# Create Decision Tree regression object
from sklearn.tree import DecisionTreeRegressor
```

Fit the tree with default hyperparameters, and find the maximum depth of the unconstrained tree

In [ ]:
```
```

```
The maximum depth of the full Decision Tree Regressor is 34
```

Find the optimal value of the hyperparameter `max_depth` with *cross-validation*

The optimization searches for the *maximum tree depth* guaranteing the smallest mean squared error At the end, this operation returns also the *fitted best tree* `best_estimator_`

In [ ]:
```
```

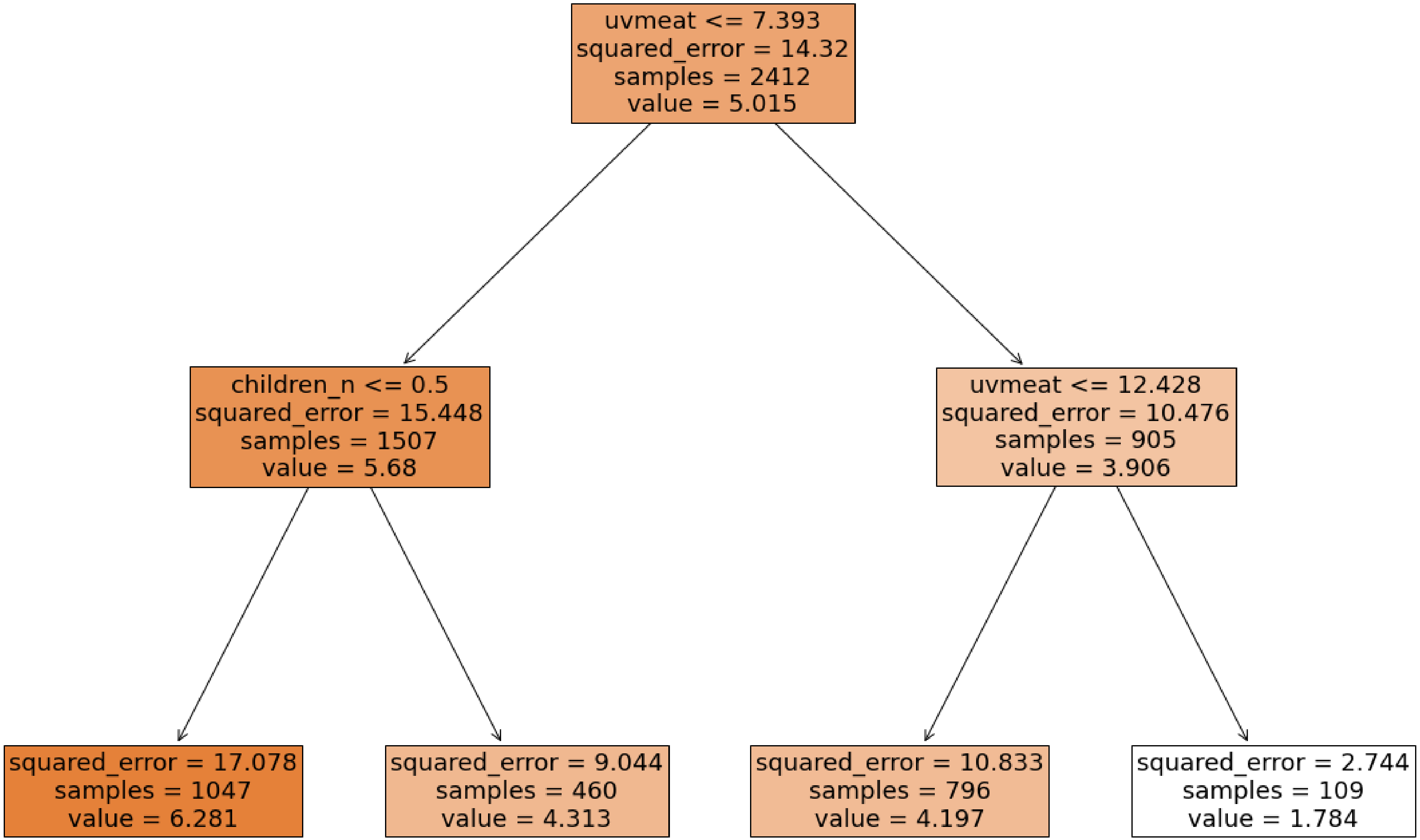
```
The optimal maximum depth for the decision tree is 2
```

In [ ]:
```
```

```
Decision Tree Regression - RMSE = 3.79
```

Show the tree

In [ ]:
```
from sklearn.tree import plot_tree
from matplotlib.pyplot import figure
# Your code here
```


uvmeat <= 7.393
squared_error = 14.32
samples = 2412
value = 5.015
children_n <= 0.5
squared_error = 15.448
samples = 1507
value = 5.68
uvmeat <= 12.428
squared_error = 10.476
samples = 905
value = 3.906
squared_error = 17.078
samples = 1047
value = 6.281
squared_error = 9.044
samples = 460
value = 4.313
squared_error = 10.833
samples = 796
value = 4.197
squared_error = 2.744
samples = 109
value = 1.784

Random Forest Multivariate Regresson

Create a Random forest regressor and fit it on the complete dataset.

For simplicity use the max_depth found in the Decision tree regressor to perform a cross validation and find the best depth for this model.

In [ ]:

```
The optimal maximum depth for the trees in the random forest is 4
```

In [ ]:

```
Random Forest Regression - RMSE = 3.58
```

Final observations

Linear regression

The multivariate regression with all the predicting variables available with respect to the univariate regression has

- lower RMSE
- higher coefficient of determination
- the p-value suggests the acceptance of both models ### Decision Tree and Random Forest regression
- Decistion Tree has an RMSE slightly higher than multivariate linear regression
- Random Forest has an RMSE slightly lower than multivariate linear regression

Control questions

1. observing the multi-variate experiment, what variable has the higher effect on the target?
2. is there a variable having an almost negligible effect on the target?
3. try to repeat the univariate experiment with the other two columns and comment the results