S1B3_DataExploration

February 20, 2022

1 Data and Data Exploration

Let us get equiped with data and few tools to explore them.

Our data strategy:

- We will use new machine learning methods. It is always good to use datasets, which are well understood and well explored. We will test our techniques on the of-the-shelf data.
  - Use data sets you can easily find help with (friends/community/online…)
  - We focus on the datasets which are available as a part of the skleanr package
  - The data set should be constant and the same across references (price returns can be tricky…)
- In addition to the well-known data, we encourage everyone to find the data they are relevant for them and we will test the techniques on such data.

1.1 Task

- Think about data set relevant to you, which can be used for classification. Data set should contain several possible explanatory features and should not be too long (1000s observations at most, 100s is suitable).
- As a default option, we will provide you with the limit order book data and we can train prediction using standard limit order book features. The data are downloaded from Lobster (you can find them mentioned around the web).
- For the Session 2, come up with suggestion of the data you would like to explore and try in your project. If the data would be relevant to you and you will find the quantum ML giving you value, you got something!

1.2 Datasets and how to load them

https://scikit-learn.org/stable/datasets/toy_dataset.html

- The boston house-prices dataset
  - Regression
  - load_boston()
- The iris dataset
  - Classification
  - load_iris()

- The breast cancer dataset
  - Classification
  - load_breast_cancer()
- The progression of diabetes dataset
  - Regression
  - load_diabetes()
- The wine recognition dataset
  - Classification
  - load_wine()
- The digits dataset (classification).
  - Classification
  - load_digits([n_class])
- The Linnerud dataset (exercise and psychological data)
  - Multivariate regression
  - load_linnerud()

[57]:
```
from sklearn import datasets

# This is the data set we will use for classification
data=datasets.load_breast_cancer()
```

[58]:
```
data.DESCR
```

[58]:
```
'.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
dataset\n--------------------------------------------\n\n**Data Set
Characteristics:**\n\n    :Number of Instances: 569\n\n    :Number of
Attributes: 30 numeric, predictive attributes and the class\n\n    :Attribute
Information:\n        - radius (mean of distances from center to points on the
perimeter)\n        - texture (standard deviation of gray-scale values)\n
- perimeter\n        - area\n        - smoothness (local variation in radius
lengths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavity
(severity of concave portions of the contour)\n        - concave points (number
of concave portions of the contour)\n        - symmetry\n        - fractal
dimension ("coastline approximation" - 1)\n\n        The mean, standard error,
and "worst" or largest (mean of the three\n        worst/largest values) of
these features were computed for each image,\n        resulting in 30 features.
For instance, field 0 is Mean Radius, field\n        10 is Radius SE, field 20
is Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n
- WDBC-Benign\n\n    :Summary Statistics:\n\n
===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius
(mean):                              6.981  28.11\n    texture (mean):
9.71   39.28\n    perimeter (mean):                          43.79  188.5\n    area
(mean):                                143.5  2501.0\n    smoothness (mean):
0.053  0.163\n    compactness (mean):                        0.019  0.345\n
concavity (mean):                          0.0    0.427\n    concave points (mean):
0.0    0.201\n    symmetry (mean):                           0.106  0.304\n
```

```
fractal dimension (mean):            0.05   0.097\n    radius (standard error):
0.112  2.873\n    texture (standard error):            0.36   4.885\n
perimeter (standard error):          0.757  21.98\n    area (standard error):
6.802  542.2\n    smoothness (standard error):         0.002  0.031\n
compactness (standard error):        0.002  0.135\n    concavity (standard
error):           0.0    0.396\n    concave points (standard error):      0.0
0.053\n    symmetry (standard error):           0.008  0.079\n    fractal
dimension (standard error):   0.001  0.03\n    radius (worst):
7.93   36.04\n    texture (worst):                     12.02  49.54\n
perimeter (worst):                   50.41  251.2\n    area (worst):
185.2  4254.0\n    smoothness (worst):                  0.071  0.223\n
compactness (worst):                 0.027  1.058\n    concavity (worst):
0.0    1.252\n    concave points (worst):              0.0    0.291\n
symmetry (worst):                    0.156  0.664\n    fractal dimension
(worst):            0.055  0.208\n    =====================================
====== ======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution:
212 - Malignant, 357 - Benign\n\n    :Creator:  Dr. William H. Wolberg, W. Nick
Street, Olvi L. Mangasarian\n\n    :Donor: Nick Street\n\n    :Date: November,
1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic)
datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image
of a fine needle\naspirate (FNA) of a breast mass.  They
describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating
plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K.
P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of
the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp.
97-101, 1992], a classification method which uses linear\nprogramming to
construct a decision tree.  Relevant features\nwere selected using an exhaustive
search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual
linear program used to obtain the separating plane\nin the 3-dimensional space
is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust
Linear\nProgramming Discrimination of Two Linearly Inseparable
Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is
also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-
prog/cpo-dataset/machine-learn/WDBC/\n\n.. topic:: References\n\n   - W.N.
Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction \n     for
breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n
Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n
San Jose, CA, 1993.\n   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast
cancer diagnosis and \n     prognosis via linear programming. Operations
Research, 43(4), pages 570-577, \n     July-August 1995.\n   - W.H. Wolberg,
W.N. Street, and O.L. Mangasarian. Machine learning techniques\n     to diagnose
breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n
163-171.'
```

[59]:
```
data.data
```

[59]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, …, 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, …, 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, …, 2.430e-01, 3.613e-01,
         8.758e-02],
        …,
        [1.660e+01, 2.808e+01, 1.083e+02, …, 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, …, 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, …, 0.000e+00, 2.871e-01,
         7.039e-02]])
```

[60]:
```
data.target
```

[60]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

[61]:
```
# It is more convenient to load the data as X and y

X, y = datasets.load_breast_cancer(return_X_y=True)
```

[62]:
```
X
```

[62]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, …, 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, …, 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, …, 2.430e-01, 3.613e-01,
        8.758e-02],
       …,
       [1.660e+01, 2.808e+01, 1.083e+02, …, 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, …, 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, …, 0.000e+00, 2.871e-01,
        7.039e-02]])
```

[ ]:
```
from sklearn import datasets
# popular data set for regressions
data=datasets.load_boston()
```

[2]:
```
data.DESCR
```

[2]:
```
".. _boston_dataset:\n\nBoston house prices
dataset\n---------------------------\n\n**Data Set Characteristics:**  \n\n
:Number of Instances: 506 \n\n    :Number of Attributes: 13 numeric/categorical
predictive. Median Value (attribute 14) is usually the target.\n\n    :Attribute
Information (in order):\n        - CRIM     per capita crime rate by town\n
- ZN       proportion of residential land zoned for lots over 25,000 sq.ft.\n
- INDUS    proportion of non-retail business acres per town\n        - CHAS
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n        -
NOX      nitric oxides concentration (parts per 10 million)\n        - RM
average number of rooms per dwelling\n        - AGE      proportion of owner-
occupied units built prior to 1940\n        - DIS      weighted distances to
five Boston employment centres\n        - RAD      index of accessibility to
radial highways\n        - TAX      full-value property-tax rate per $10,000\n
- PTRATIO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2
where Bk is the proportion of blacks by town\n        - LSTAT    % lower status
of the population\n        - MEDV     Median value of owner-occupied homes in
$1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and
Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing
dataset.\nhttps://archive.ics.uci.edu/ml/machine-learning-
databases/housing/\n\n\nThis dataset was taken from the StatLib library which is
maintained at Carnegie Mellon University.\n\nThe Boston house-price data of
Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air',
J. Environ. Economics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh
& Welsch, 'Regression diagnostics\n…', Wiley, 1980.   N.B. Various
transformations are used in the table on\npages 244-261 of the latter.\n\nThe
```

```
Boston house-price data has been used in many machine learning papers that
address regression\nproblems.   \n     \n.. topic:: References\n\n   - Belsley,
Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources
of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining
Instance-Based and Model-Based Learning. In Proceedings on the Tenth
International Conference of Machine Learning, 236-243, University of
Massachusetts, Amherst. Morgan Kaufmann.\n"
```

[3]:
```
data.data
```

[3]:
```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, …, 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, …, 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, …, 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        …,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, …, 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, …, 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, …, 2.1000e+01, 3.9690e+02,
         7.8800e+00]])
```

[4]:
```
data.target
```

[4]:
```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
```

```
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
```

1.3 Data Exploration

In the following part, we will explore the data set and review some useful techniques to perform data exploration.

We will use following packages:

- numpy
- pandas
- scikit-learn
- matplotlib
- seaborn

[5]:
```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston

dataBoston = load_boston()

df = pd.DataFrame(dataBoston.data, columns=dataBoston.feature_names)
df['target'] = dataBoston.target
```

[6]:
```
df.head()
```

[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0

\

	PTRATIO	B	LSTAT	target
0	15.3	396.90	4.98	24.0
1	17.8	396.90	9.14	21.6
2	17.8	392.83	4.03	34.7
3	18.7	394.63	2.94	33.4
4	18.7	396.90	5.33	36.2

[7]:
```
# Data size and number of features
instance_count, attr_count = df.shape
```

[8]:
```
instance_count
```

[8]: 506

[9]:
```
attr_count
```

[9]: 14

[10]:
```
# describe data
df.describe()
```

[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000

\

	AGE	DIS	RAD	TAX	PTRATIO	B
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032
std	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864
min	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000
25%	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500
50%	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000

\

```
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT      target
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

[11]:
```
# we can calculate the quantities
df.median()
```

[11]:
```
CRIM         0.25651
ZN           0.00000
INDUS        9.69000
CHAS         0.00000
NOX          0.53800
RM           6.20850
AGE         77.50000
DIS          3.20745
RAD          5.00000
TAX        330.00000
PTRATIO     19.05000
B          391.44000
LSTAT       11.36000
target      21.20000
dtype: float64
```

[12]:
```
df.mean()
```

[12]:
```
CRIM         3.613524
ZN          11.363636
INDUS       11.136779
CHAS         0.069170
NOX          0.554695
RM           6.284634
AGE         68.574901
DIS          3.795043
RAD          9.549407
TAX        408.237154
PTRATIO     18.455534
B          356.674032
LSTAT       12.653063
```

```
target      22.532806
dtype: float64
```

[13]:
```
df.quantile(0.25)
```

[13]:
```
CRIM         0.082045
ZN           0.000000
INDUS        5.190000
CHAS         0.000000
NOX          0.449000
RM           5.885500
AGE         45.025000
DIS          2.100175
RAD          4.000000
TAX        279.000000
PTRATIO     17.400000
B          375.377500
LSTAT        6.950000
target      17.025000
Name: 0.25, dtype: float64
```

[14]:
```
# are there any nulls?
pd.isnull(df).any()
```

[14]:
```
CRIM       False
ZN         False
INDUS      False
CHAS       False
NOX        False
RM         False
AGE        False
DIS        False
RAD        False
TAX        False
PTRATIO    False
B          False
LSTAT      False
target     False
dtype: bool
```

2 Always Explore your Data

It is a good practice always explore the data.

2.1 Correlation

When we aim to start find relationship in the dataset, the correlation is the first thing to check.

Pandas provides three correlations (yes, there is more than one):

- Pearson correlation:
  - The standard correlation coefficient (covarience normalised by square-root of variances)
- Spearman's rank correlation:
  - Correlation of the ranks (for every variable, we rank the values and work with ranks instead of values themselves)
- Kendall tau correlation:
  - Normalised difference of the number of concordant pairs and the number of discordant pairs
    - Pair of data points (xi, yi) and (xj,yj) is concordant if either xi>xj and yi>yj, or xi<yi and yi<yj; otherwise it is discordant

[15]:
```python
df.corr()
```

[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE \
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339
target	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955

	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
CRIM	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455621	-0.388305
ZN	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412995	0.360445
INDUS	-0.708027	0.595129	0.720760	0.383248	-0.356977	0.603800	-0.483725
CHAS	-0.099176	-0.007368	-0.035587	-0.121515	0.048788	-0.053929	0.175260
NOX	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.590879	-0.427321
RM	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.613808	0.695360
AGE	-0.747881	0.456022	0.506456	0.261515	-0.273534	0.602339	-0.376955
DIS	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.496996	0.249929
RAD	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.488676	-0.381626
TAX	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.543993	-0.468536
PTRATIO	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.374044	-0.507787
B	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.366087	0.333461
LSTAT	-0.496996	0.488676	0.543993	0.374044	-0.366087	1.000000	-0.737663
target	0.249929	-0.381626	-0.468536	-0.507787	0.333461	-0.737663	1.000000

[16]:
```python
df.corr(method='pearson')
```

[16]:
```
            CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
CRIM     1.000000 -0.200469  0.406583 -0.055892  0.420972 -0.219247  0.352734
ZN      -0.200469  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.569537
INDUS    0.406583 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.644779
CHAS    -0.055892 -0.042697  0.062938  1.000000  0.091203  0.091251  0.086518
NOX      0.420972 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.731470
RM      -0.219247  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.240265
AGE      0.352734 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.000000
DIS     -0.379670  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.747881
RAD      0.625505 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.456022
TAX      0.582764 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.506456
PTRATIO  0.289946 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.261515
B       -0.385064  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.273534
LSTAT    0.455621 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.602339
target  -0.388305  0.360445 -0.483725  0.175260 -0.427321  0.695360 -0.376955

              DIS       RAD       TAX   PTRATIO         B     LSTAT    target
CRIM    -0.379670  0.625505  0.582764  0.289946 -0.385064  0.455621 -0.388305
ZN       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
B        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
target   0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

[17]:
```python
df.corr(method='spearman')
```

[17]:
```
            CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
CRIM     1.000000 -0.571660  0.735524  0.041537  0.821465 -0.309116  0.704140
ZN      -0.571660  1.000000 -0.642811 -0.041937 -0.634828  0.361074 -0.544423
INDUS    0.735524 -0.642811  1.000000  0.089841  0.791189 -0.415301  0.679487
CHAS     0.041537 -0.041937  0.089841  1.000000  0.068426  0.058813  0.067792
NOX      0.821465 -0.634828  0.791189  0.068426  1.000000 -0.310344  0.795153
RM      -0.309116  0.361074 -0.415301  0.058813 -0.310344  1.000000 -0.278082
AGE      0.704140 -0.544423  0.679487  0.067792  0.795153 -0.278082  1.000000
DIS     -0.744986  0.614627 -0.757080 -0.080248 -0.880015  0.263168 -0.801610
RAD      0.727807 -0.278767  0.455507  0.024579  0.586429 -0.107492  0.417983
TAX      0.729045 -0.371394  0.664361 -0.044486  0.649527 -0.271898  0.526366
PTRATIO  0.465283 -0.448475  0.433710 -0.136065  0.391309 -0.312923  0.355384
B       -0.360555  0.163135 -0.285840 -0.039810 -0.296662  0.053660 -0.228022
```

LSTAT	0.634760	-0.490074	0.638747	-0.050575	0.636828	-0.640832	0.657071
target	-0.558891	0.438179	-0.578255	0.140612	-0.562609	0.633576	-0.547562

	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
CRIM	-0.744986	0.727807	0.729045	0.465283	-0.360555	0.634760	-0.558891
ZN	0.614627	-0.278767	-0.371394	-0.448475	0.163135	-0.490074	0.438179
INDUS	-0.757080	0.455507	0.664361	0.433710	-0.285840	0.638747	-0.578255
CHAS	-0.080248	0.024579	-0.044486	-0.136065	-0.039810	-0.050575	0.140612
NOX	-0.880015	0.586429	0.649527	0.391309	-0.296662	0.636828	-0.562609
RM	0.263168	-0.107492	-0.271898	-0.312923	0.053660	-0.640832	0.633576
AGE	-0.801610	0.417983	0.526366	0.355384	-0.228022	0.657071	-0.547562
DIS	1.000000	-0.495806	-0.574336	-0.322041	0.249595	-0.564262	0.445857
RAD	-0.495806	1.000000	0.704876	0.318330	-0.282533	0.394322	-0.346776
TAX	-0.574336	0.704876	1.000000	0.453345	-0.329843	0.534423	-0.562411
PTRATIO	-0.322041	0.318330	0.453345	1.000000	-0.072027	0.467259	-0.555905
B	0.249595	-0.282533	-0.329843	-0.072027	1.000000	-0.210562	0.185664
LSTAT	-0.564262	0.394322	0.534423	0.467259	-0.210562	1.000000	-0.852914
target	0.445857	-0.346776	-0.562411	-0.555905	0.185664	-0.852914	1.000000

[18]:

```
df.corr(method='kendall')
```

[18]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE \
CRIM	1.000000	-0.462057	0.521014	0.033948	0.603361	-0.211718	0.497297
ZN	-0.462057	1.000000	-0.535468	-0.039419	-0.511464	0.278134	-0.429389
INDUS	0.521014	-0.535468	1.000000	0.075889	0.612030	-0.291318	0.489070
CHAS	0.033948	-0.039419	0.075889	1.000000	0.056387	0.048080	0.055616
NOX	0.603361	-0.511464	0.612030	0.056387	1.000000	-0.215633	0.589608
RM	-0.211718	0.278134	-0.291318	0.048080	-0.215633	1.000000	-0.187611
AGE	0.497297	-0.429389	0.489070	0.055616	0.589608	-0.187611	1.000000
DIS	-0.539878	0.478524	-0.565137	-0.065619	-0.683930	0.179801	-0.609836
RAD	0.563969	-0.234663	0.353967	0.021739	0.434828	-0.076569	0.306201
TAX	0.544956	-0.289911	0.483228	-0.037655	0.453258	-0.190532	0.360311
PTRATIO	0.312768	-0.361607	0.336612	-0.115694	0.278678	-0.223194	0.251857
B	-0.264378	0.128177	-0.192017	-0.033277	-0.202430	0.032951	-0.154056
LSTAT	0.454837	-0.386818	0.465980	-0.041344	0.452005	-0.468231	0.485359
target	-0.403964	0.339989	-0.418430	0.115202	-0.394995	0.482829	-0.387758

	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
CRIM	-0.539878	0.563969	0.544956	0.312768	-0.264378	0.454837	-0.403964
ZN	0.478524	-0.234663	-0.289911	-0.361607	0.128177	-0.386818	0.339989
INDUS	-0.565137	0.353967	0.483228	0.336612	-0.192017	0.465980	-0.418430
CHAS	-0.065619	0.021739	-0.037655	-0.115694	-0.033277	-0.041344	0.115202
NOX	-0.683930	0.434828	0.453258	0.278678	-0.202430	0.452005	-0.394995
RM	0.179801	-0.076569	-0.190532	-0.223194	0.032951	-0.468231	0.482829
AGE	-0.609836	0.306201	0.360311	0.251857	-0.154056	0.485359	-0.387758
DIS	1.000000	-0.361892	-0.381988	-0.223486	0.168631	-0.409347	0.313115
RAD	-0.361892	1.000000	0.558107	0.251913	-0.214364	0.287943	-0.248115

```
TAX     -0.381988  0.558107  1.000000  0.287769 -0.241606  0.384191 -0.414650
PTRATIO -0.223486  0.251913  0.287769  1.000000 -0.042152  0.330335 -0.398789
B        0.168631 -0.214364 -0.241606 -0.042152  1.000000 -0.145430  0.126955
LSTAT   -0.409347  0.287943  0.384191  0.330335 -0.145430  1.000000 -0.668656
target   0.313115 -0.248115 -0.414650 -0.398789  0.126955 -0.668656  1.000000
```

2.2 Computational requirements

We can use %timeit command to verify that the three correlations have different computational costs:

[19]:
```
methods=['pearson','spearman','kendall']
for m in methods:
    %timeit df.corr(method=m)
```

```
411 µs ± 2.65 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
11.3 ms ± 252 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
31.8 ms ± 452 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In practice, we are interested in correlation between the target variable and the independent variables.

This helps us to understand how each of the variables can help us to predict the target:

[20]:
```
corMatrix = df.corr(method='pearson')

predictions=corMatrix.iloc[-1][:-1]
predictions.sort_values(ascending=False)
```

[20]:
```
RM          0.695360
ZN          0.360445
B           0.333461
DIS         0.249929
CHAS        0.175260
AGE        -0.376955
RAD        -0.381626
CRIM       -0.388305
NOX        -0.427321
TAX        -0.468536
INDUS      -0.483725
PTRATIO    -0.507787
LSTAT      -0.737663
Name: target, dtype: float64
```

This allows us to identify candidates for further machine learning models.

Colinearity: Some features may show significant correlation to target.

- Problem: Some features can be strongly correlated with each other and thus adding all of them together can cause problems.

  - Ordinary least squares (linear regression) can be affected and give misleading results
- How to identify the strongly correlated features?

[21]:
```python
attrs = corMatrix.iloc[:-1,:-1]
# threshold
thresholdCorr = 0.75
# selecting only the correlations above the threshold
important_corrs = (attrs[abs(attrs) > thresholdCorr][attrs != 1.0]) \
    .unstack().dropna().to_dict()

# we need only (a,b) while (b,a) is duplicated
unique_important_corrs = pd.DataFrame(
    list(set([(tuple(sorted(key)), important_corrs[key]) \
    for key in important_corrs])), columns=['pair', 'correlation'])

# we can sort the outcome based on values
unique_important_corrs = unique_important_corrs.iloc[
    abs(unique_important_corrs['correlation']).argsort()[::-1]]
```

[22]:
```python
unique_important_corrs
```

[22]:
```
             pair  correlation
2      (RAD, TAX)     0.910228
1      (DIS, NOX)    -0.769230
0    (INDUS, NOX)     0.763651
```

[23]:
```python
# let us recall the correlations with target values
predictions.sort_values(ascending=False)
```

[23]:
```
RM         0.695360
ZN         0.360445
B          0.333461
DIS        0.249929
CHAS       0.175260
AGE       -0.376955
RAD       -0.381626
CRIM      -0.388305
NOX       -0.427321
TAX       -0.468536
INDUS     -0.483725
PTRATIO   -0.507787
LSTAT     -0.737663
Name: target, dtype: float64
```

Overall, there are three strongly correlated pairs of features. Only one of the features, INDUS, is strongly correlated with the target. The colinearity does not seem to be the primary worry initially.

2.3 Visualisation

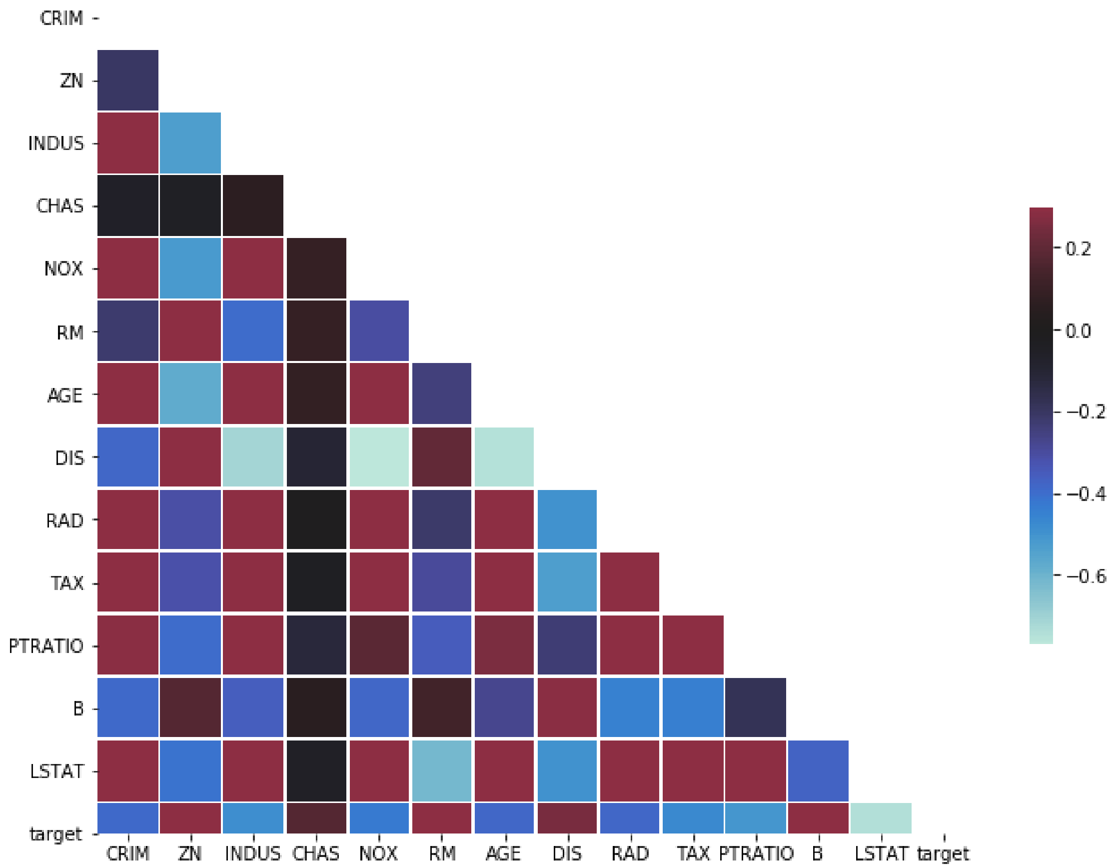
We can visualise the same information. We will use seaborn package.

[24]:
```python
import pandas as pd
import seaborn.apionly as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Correlation matrix
corr = df.corr()

# Mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Heatmap
sns.heatmap(corr, mask=mask, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
/Users/jannovotny/opt/anaconda3/lib/python3.7/_collections_abc.py:841:
MatplotlibDeprecationWarning:
The examples.directory rcparam was deprecated in Matplotlib 3.0 and will be
removed in 3.2. In the future, examples will be found relative to the 'datapath'
directory.
  self[key] = other[key]
/Users/jannovotny/opt/anaconda3/lib/python3.7/_collections_abc.py:841:
MatplotlibDeprecationWarning:
The savefig.frameon rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
  self[key] = other[key]
/Users/jannovotny/opt/anaconda3/lib/python3.7/_collections_abc.py:841:
MatplotlibDeprecationWarning:
The text.latex.unicode rcparam was deprecated in Matplotlib 3.0 and will be
removed in 3.2.
  self[key] = other[key]
/Users/jannovotny/opt/anaconda3/lib/python3.7/_collections_abc.py:841:
MatplotlibDeprecationWarning:
The verbose.fileo rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
  self[key] = other[key]
/Users/jannovotny/opt/anaconda3/lib/python3.7/_collections_abc.py:841:
MatplotlibDeprecationWarning:
The verbose.level rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
  self[key] = other[key]
```

```
/Users/jannovotny/opt/anaconda3/lib/python3.7/site-
packages/seaborn/apionly.py:9: UserWarning: As seaborn no longer sets a default
style on import, the seaborn.apionly module is deprecated. It will be removed in
a future version.
  warnings.warn(msg, UserWarning)
```

[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f928af6b310>



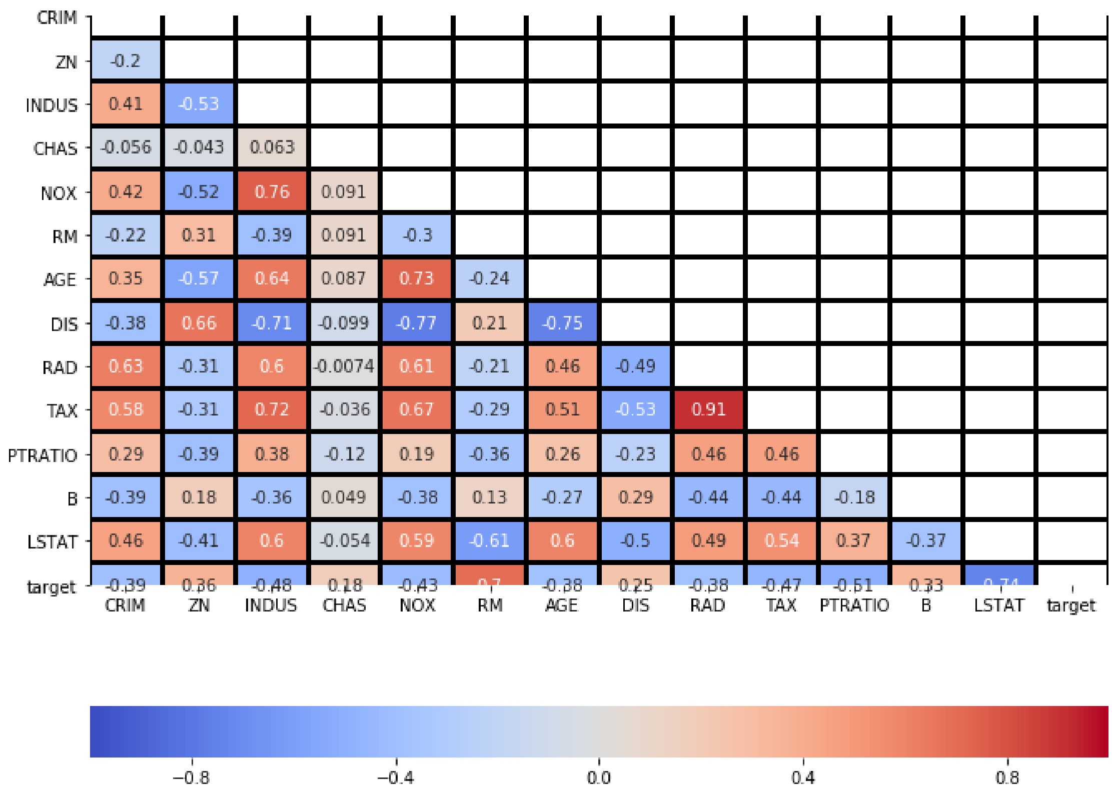
Variable CHAS is standing out.

Let us add significance to correlations. We will use stats.pearsonr from scipy.

[25]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# define function which plots correlation matrix
def plotCorMat(corr, mask=None):
    f, ax = plt.subplots(figsize=(11, 9))
```

```
    sns.heatmap(corr, ax=ax,
                mask=mask,
                # cosmetics
                annot=True, vmin=-1, vmax=1, center=0,
                cmap='coolwarm', linewidths=2, linecolor='black',␣
↪cbar_kws={'orientation': 'horizontal'})
```

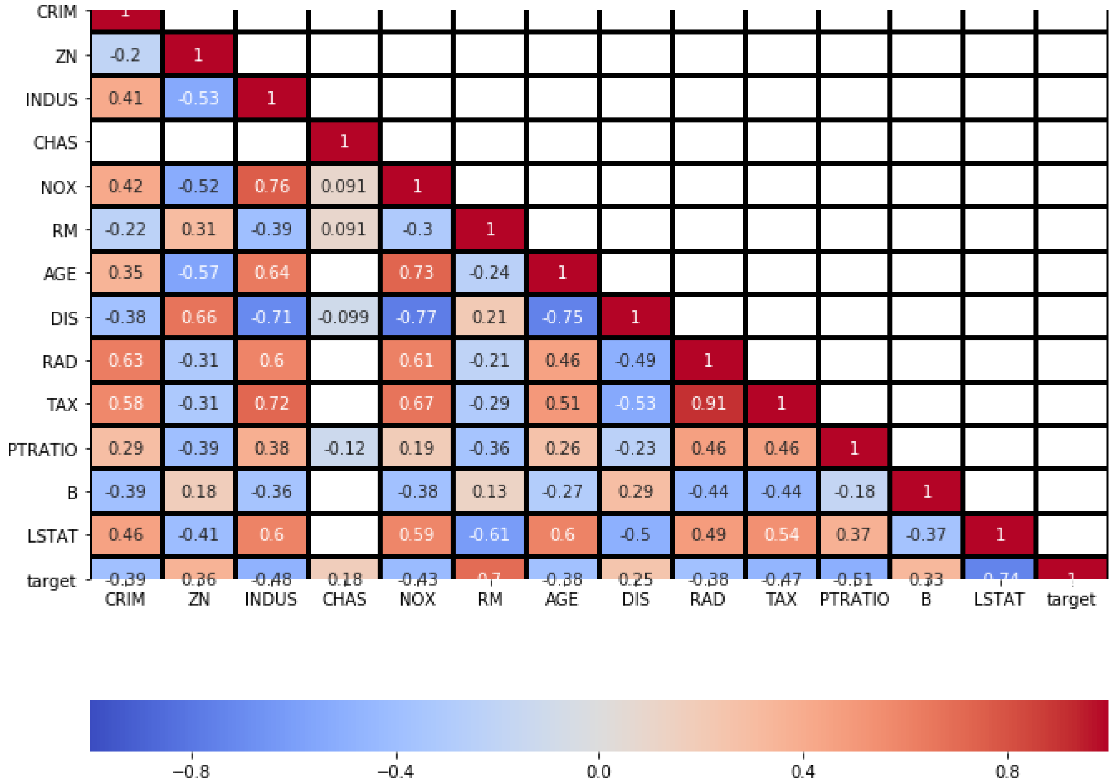
[26]:
```
# Correlation matrix
corr = df.corr()
# Upper triangular mask
mask = np.triu(corr)
# Plot the correlation matrix
plotCorMat(corr,mask)
plt.show()
```



[27]:
```
# Calculating the p-value of the correlation
def corrPVal(df=None):
    pM = np.zeros(shape=(df.shape[1],df.shape[1]))
    for col in df.columns:
        for col2 in df.drop(col,axis=1).columns:
            _ , p = stats.pearsonr(df[col],df[col2])
```

```
            pM[df.columns.to_list().index(col),df.columns.to_list().
↪index(col2)] = p
    return pM
```

[28]:
```
# Significance filter
corr = df.corr()
pVals = corrPVal(df)
mask = np.invert(np.tril(pVals<0.05))
plotCorMat(corr,mask)
```



CHAS is indeed less correlated to the other features.

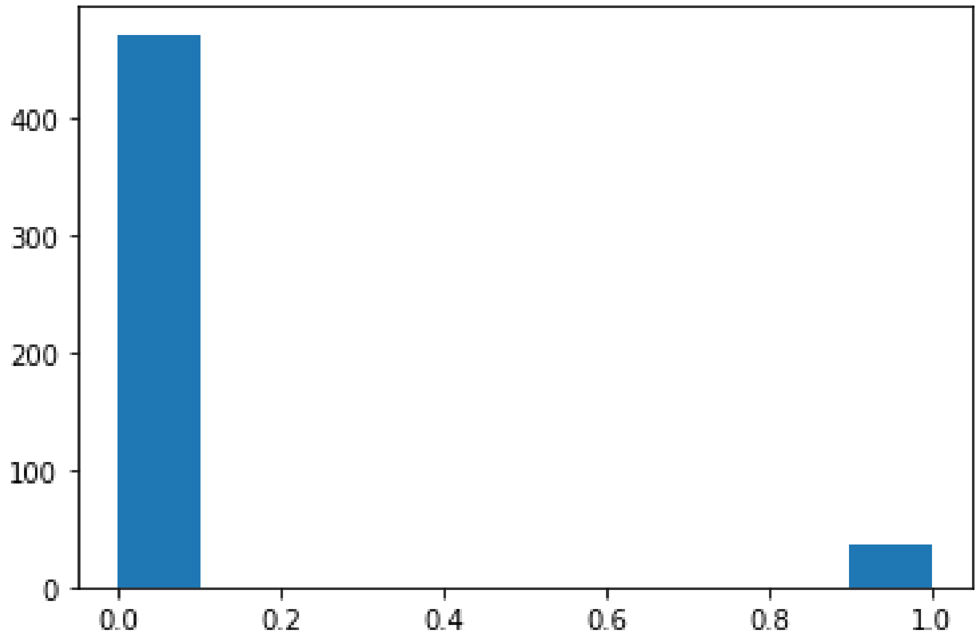
2.4 Focus on the features

Let us have a look on the feature values. Visualisation can help us to understand the features.

We can use it to undertand the CHAS variable using histogram:

[29]:
```
import matplotlib.pyplot as plt
attr = df['CHAS']
plt.hist(attr)
```

[29]: (array([471.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 35.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.]),
 <a list of 10 Patch objects>)

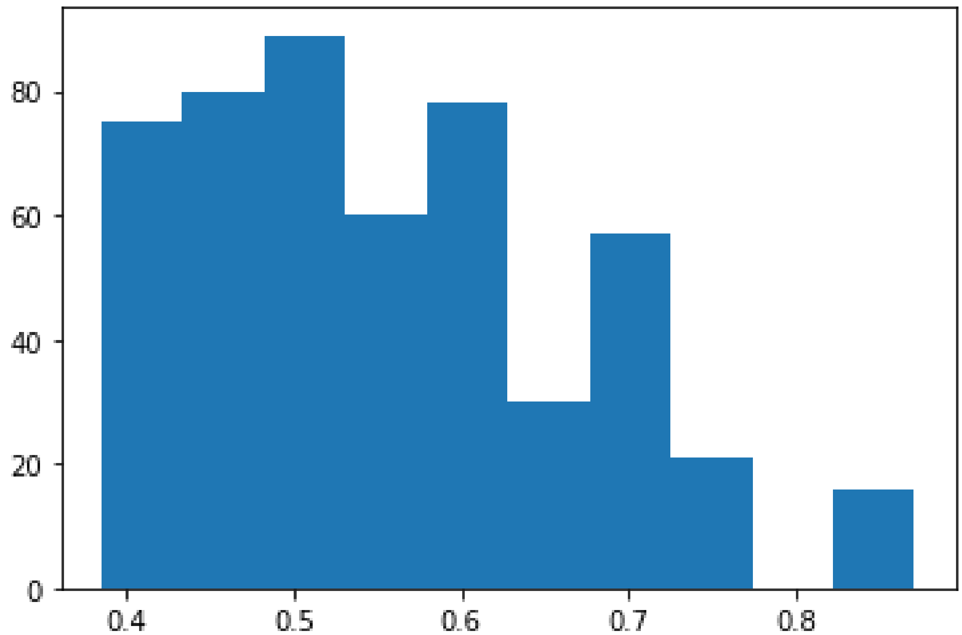


We know why we see very week correlation with other variables.

Let us review other variables.

[30]:
```
attr = df['NOX']
plt.hist(attr)
```

[30]: (array([75., 80., 89., 60., 78., 30., 57., 21., 0., 16.]),
 array([0.385 , 0.4336, 0.4822, 0.5308, 0.5794, 0.628 , 0.6766, 0.7252,
        0.7738, 0.8224, 0.871]),
 <a list of 10 Patch objects>)

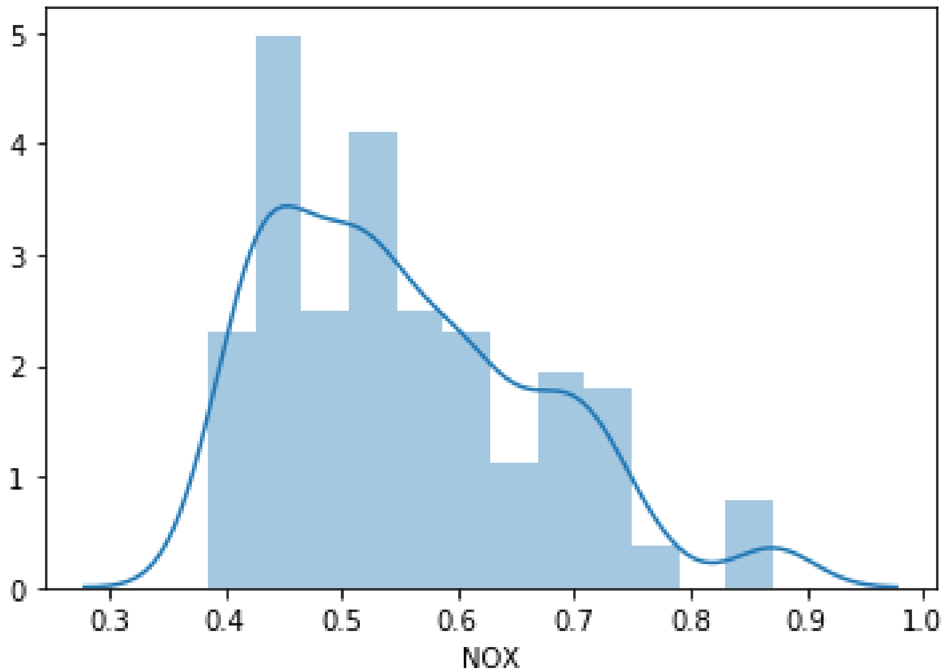


It may be useful to add the kernel density (smoothed histogram).

We use sns package for it:

[31]:
```
sns.distplot(attr)
```

[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f928ba645d0>



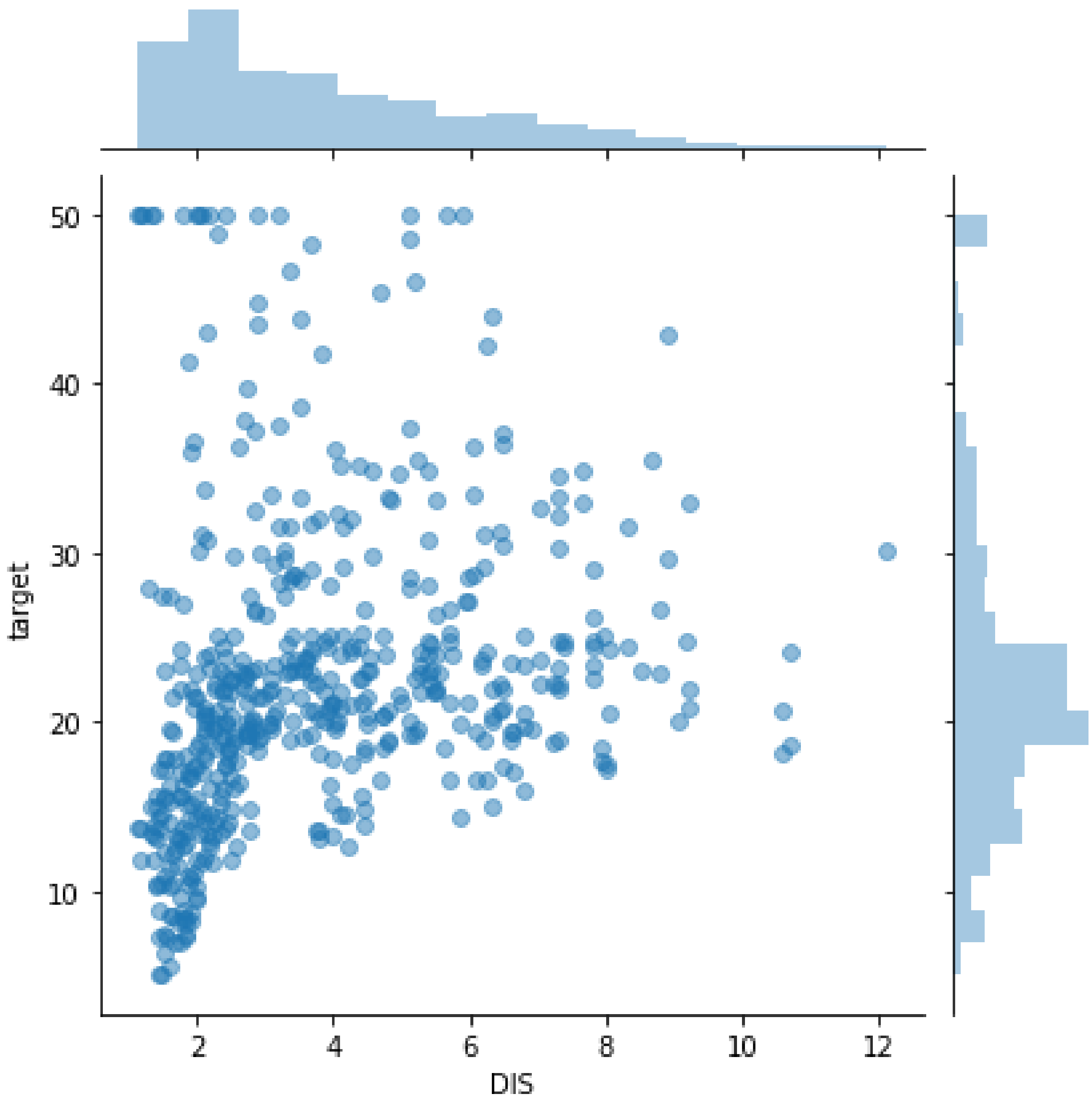
2.5 Pairwise Relationship

Another plot is a scatter plot. We can nicely visualise pairs of variables/features as they occur in the data.

- Ideas for model (linear vs non-linear)
  - Be careful of overfitting – never choose model by visually inspecting the entire dataset (even the data we will use for validation)
    * It can be used, but for very "weak" decisions
- Outliers

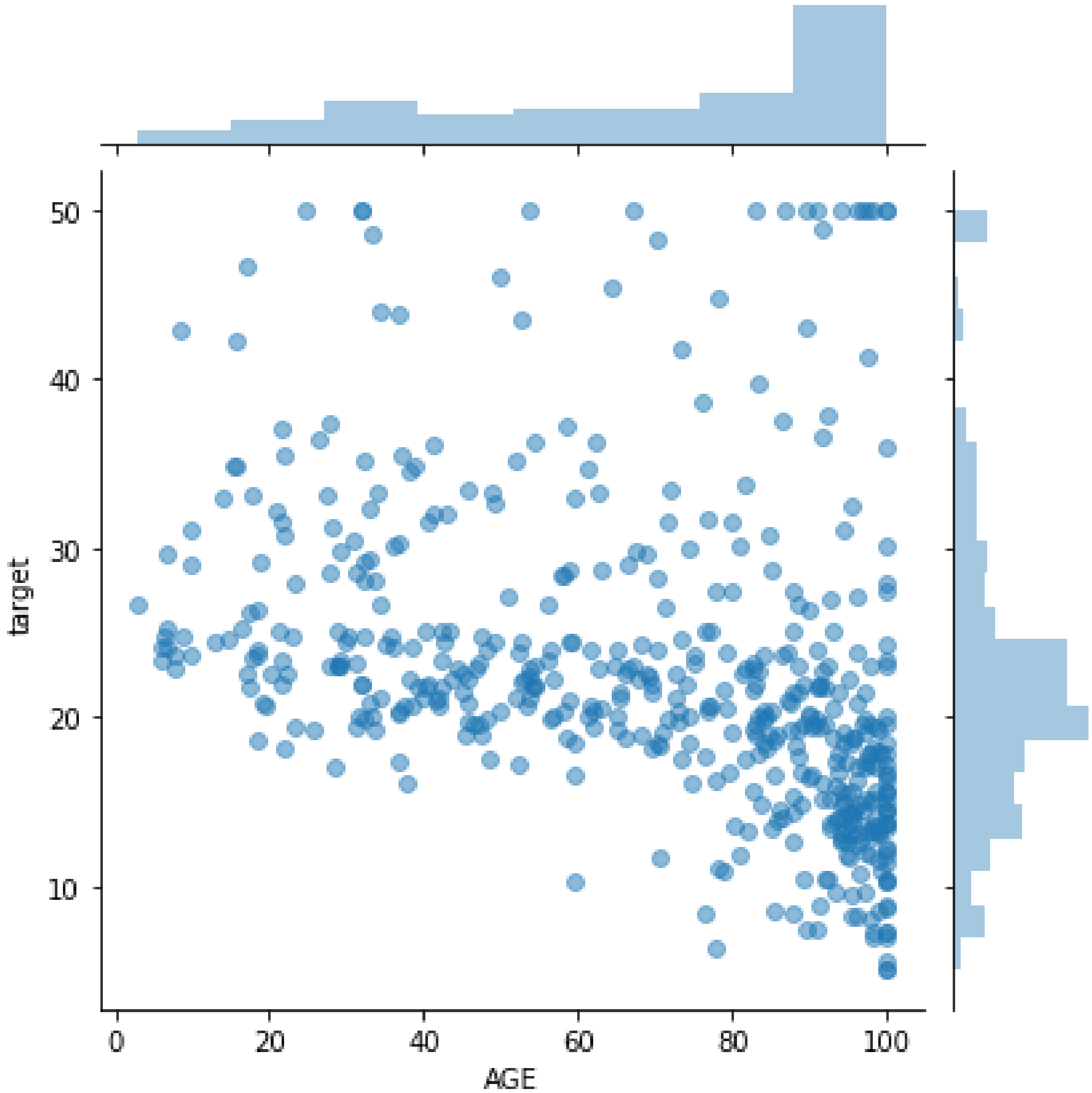
[32]:
```
# distance and price
x, y = df['DIS'], df['target']
sns.jointplot(x, y, kind='scatter', joint_kws={'alpha':0.5})
```

[32]:
```
<seaborn.axisgrid.JointGrid at 0x7f928aa14290>
```



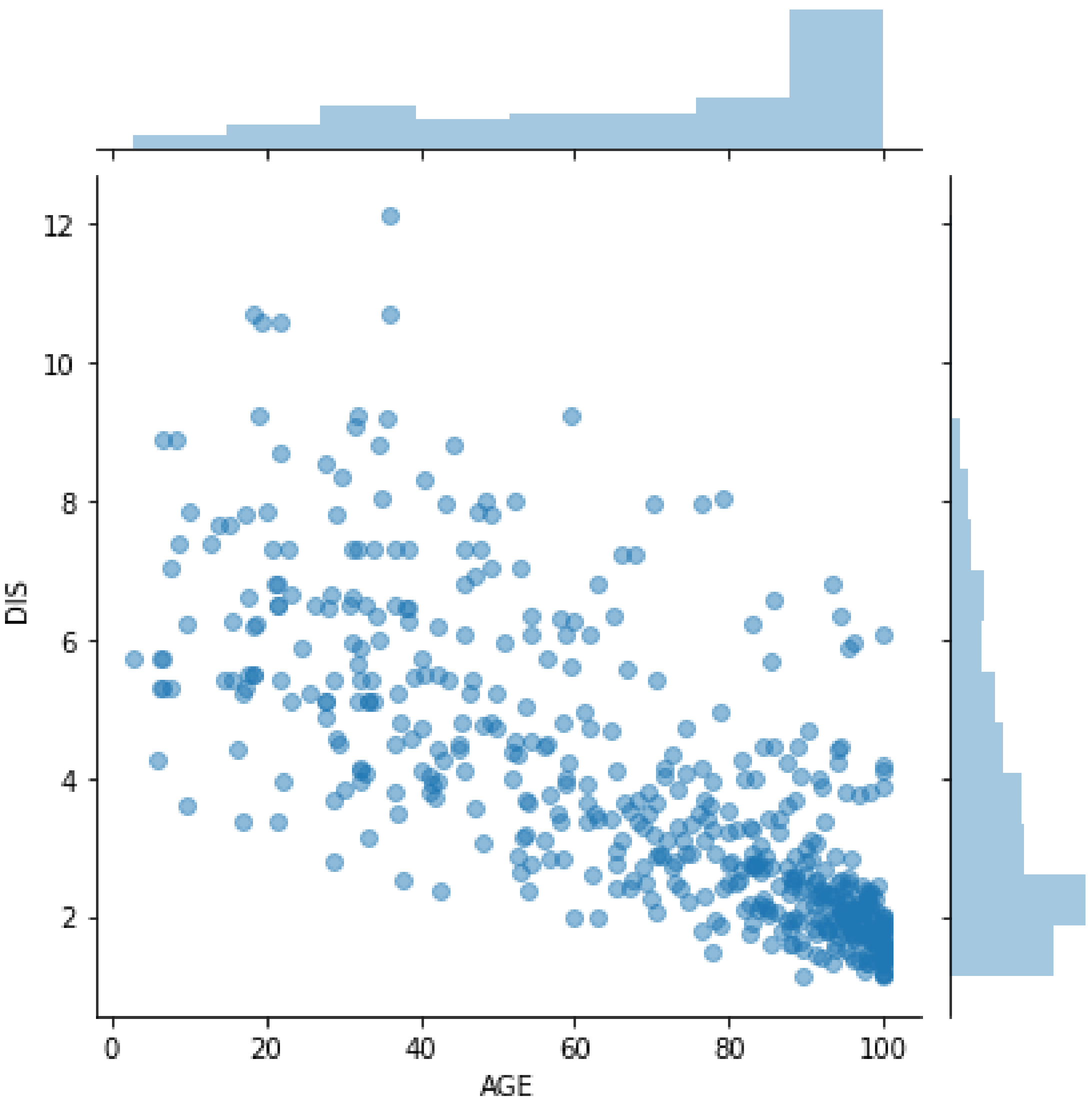
[33]:
```
# price and age
x, y = df['AGE'], df['target']
sns.jointplot(x, y, kind='scatter', joint_kws={'alpha':0.5})
```

[33]: <seaborn.axisgrid.JointGrid at 0x7f928bf30410>




```
[34]: # distance and age
      x, y = df['AGE'], df['DIS']
      sns.jointplot(x, y, kind='scatter', joint_kws={'alpha':0.5})
```

```
[34]: <seaborn.axisgrid.JointGrid at 0x7f928bef9d50>
```



In the ideal case, the features are independent. If there is large correlation (in absolute value), both variables do not add explanatory power into the model.

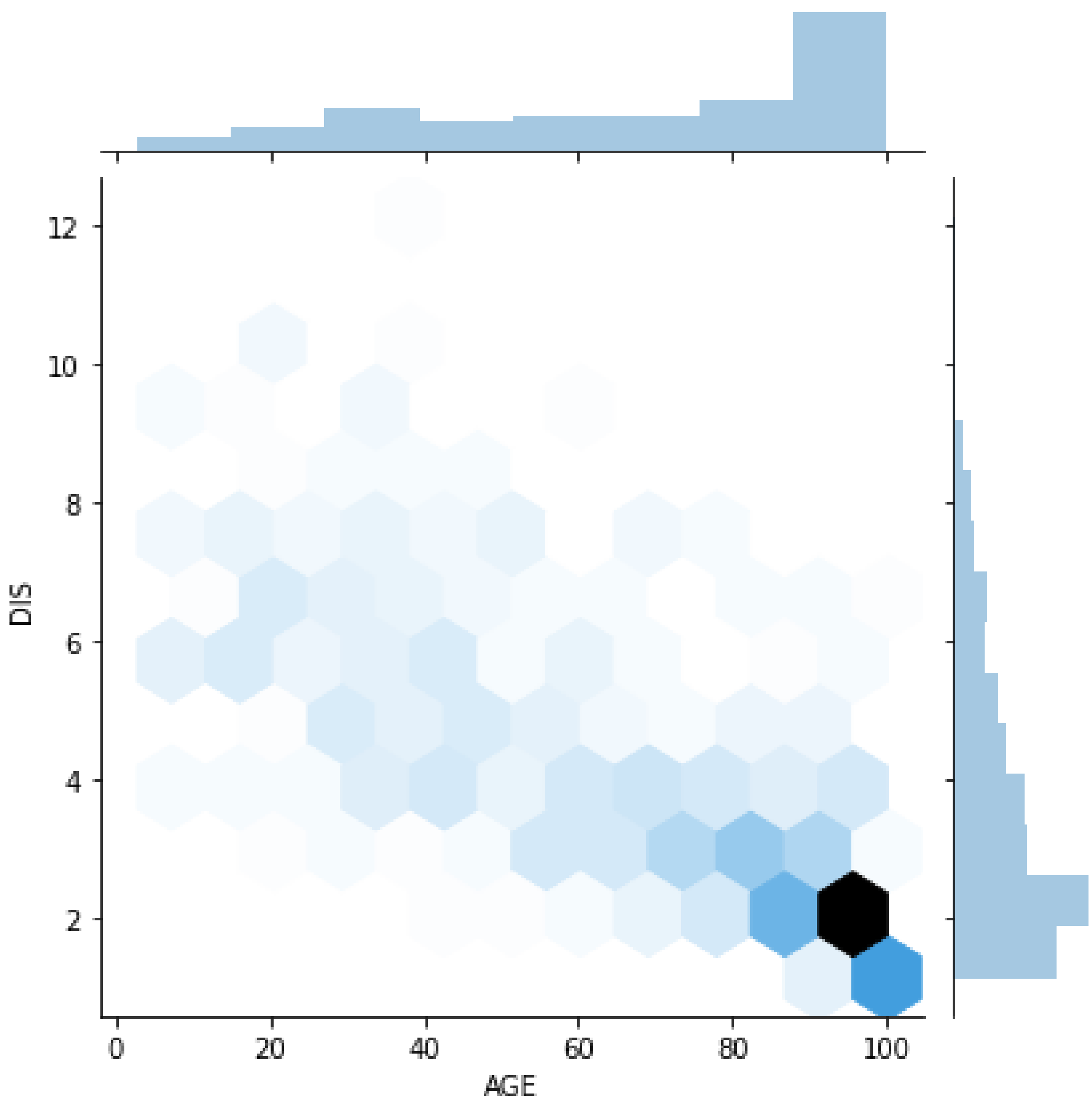
- We can explain DIS with AGE (and vice versa)
  - Large value of AGE will be likely observed with small value of DIS
  - Inferring price from large value of AGE will not be much enhanced by observing DIS as it will likely be small
    - The correlation is not perfect, there is some additional explanatory power

2.6 Multidimensional distribution functions

We can estimate the 2D PDF – surface; this represents generalisation of histograms.

[35]:
```
# hexagonal plots
sns.jointplot(df['AGE'], df['DIS'], kind='hex')
```

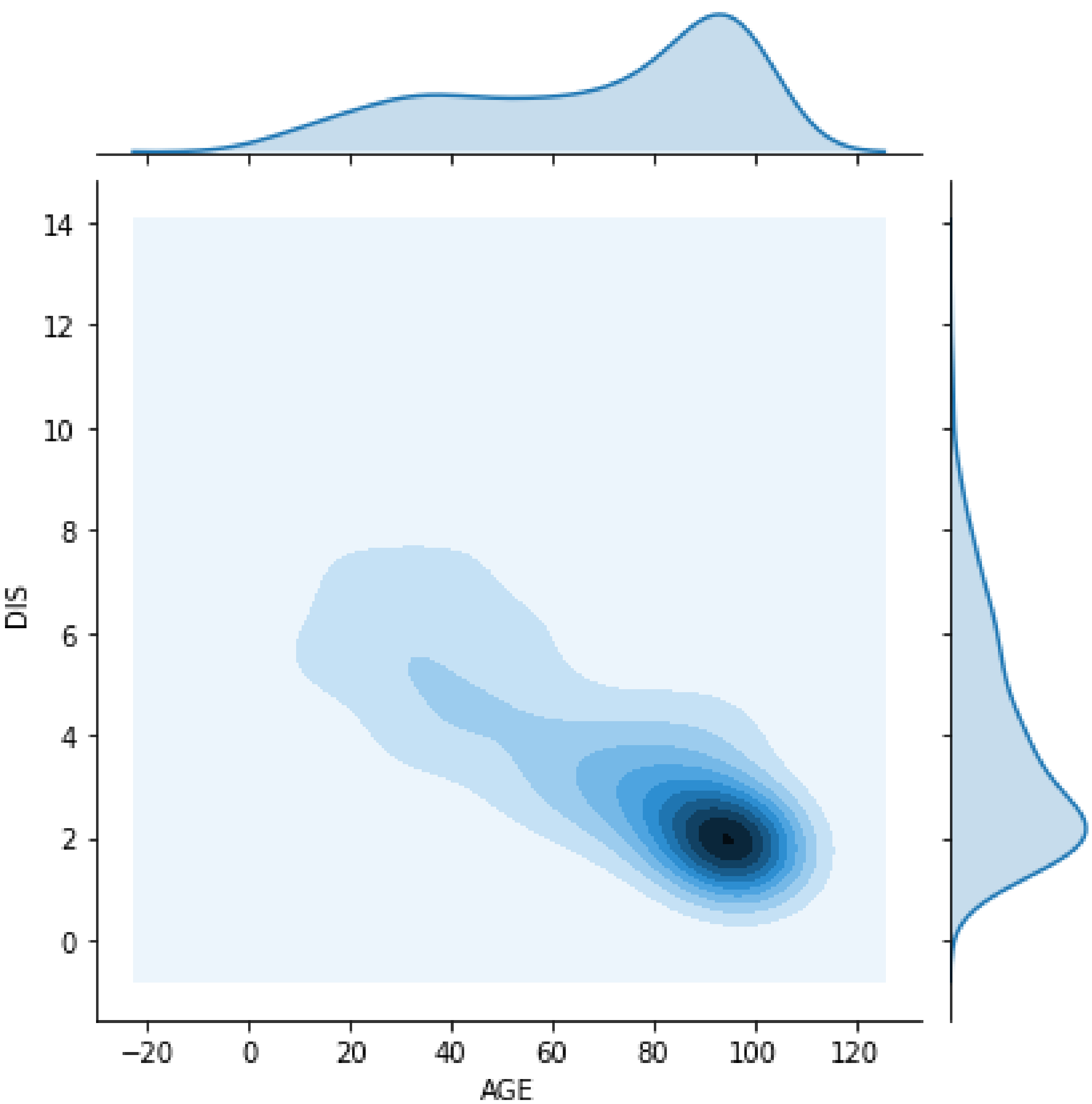
[35]: <seaborn.axisgrid.JointGrid at 0x7f928b68c350>



Another nice plot is contour plot:

[36]:
```
sns.jointplot(df['AGE'], df['DIS'], kind='kde')
```

[36]: <seaborn.axisgrid.JointGrid at 0x7f928c387c10>



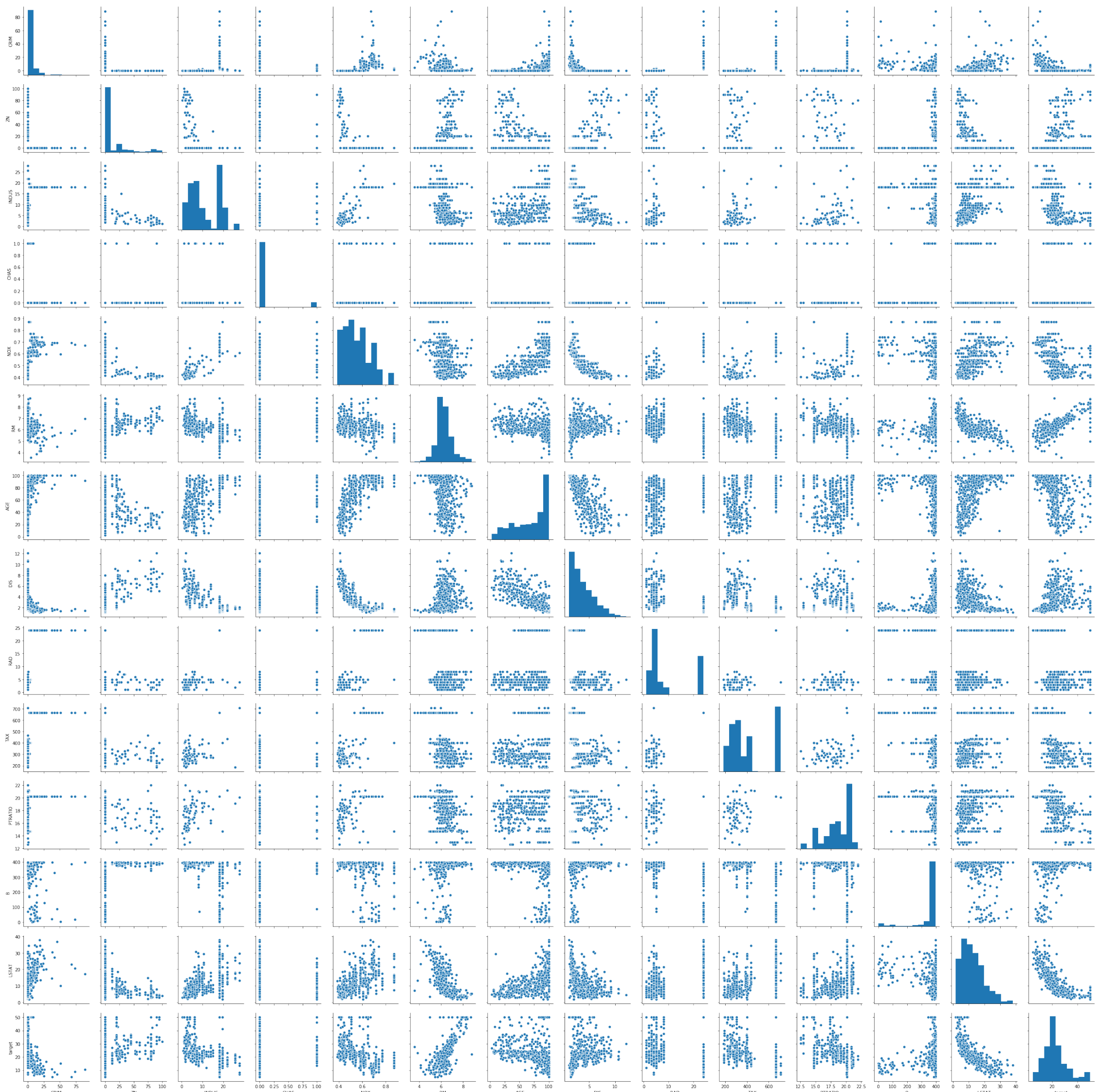
2.7 Pairwise Scatter Plots

Seaborn allows us to visualise the dependency across the entire dataset.

High-level view of the data combining the pair-wise scatter plots and distribution of individual variables.

[37]:
```
sns.pairplot(df)
```

[37]: <seaborn.axisgrid.PairGrid at 0x7f928b73e490>



3 Synthetic datasets

In many situations, we are proposing a certain ML model and we need a tailored synthetic data set. For that purpose, we can use sklearn routines to generate the data.

3.1 Classification data

[38]:
```
from sklearn.datasets import make_classification
import numpy as np
import pandas as pd
X, y = make_classification(n_samples=1000, n_features=4,n_informative=2,␣
 ↪n_redundant=0,random_state=0, shuffle=False)
```

[39]:
```python
X.shape
```

[39]: (1000, 4)

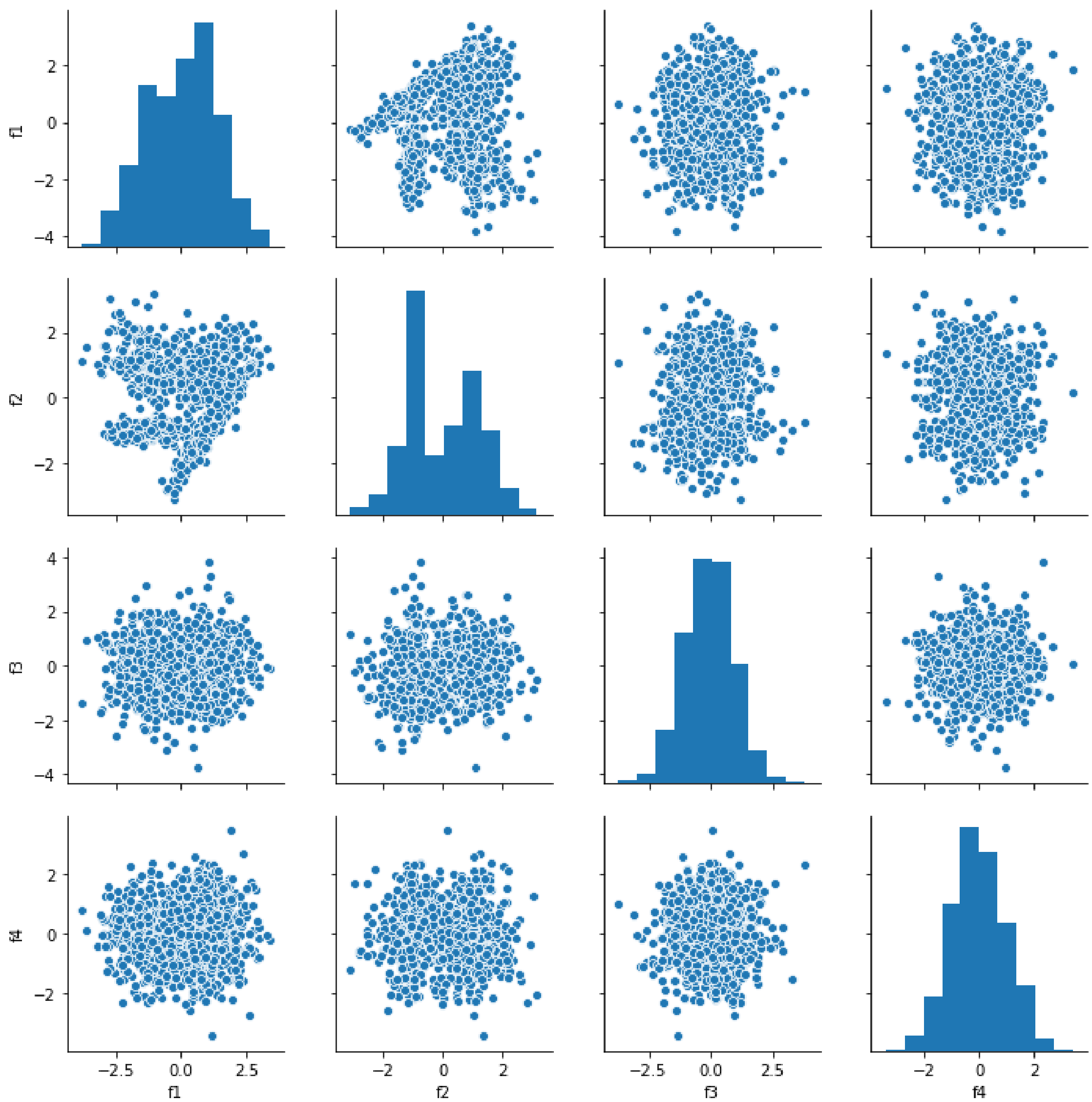
[40]:
```python
featureNames=['f1','f2','f3','f4'];
featureNames
```

[40]: ['f1', 'f2', 'f3', 'f4']

[41]:
```python
dfClass=pd.DataFrame(X, columns=featureNames)
```

[42]:
```python
import seaborn as sns
sns.pairplot(dfClass)
```

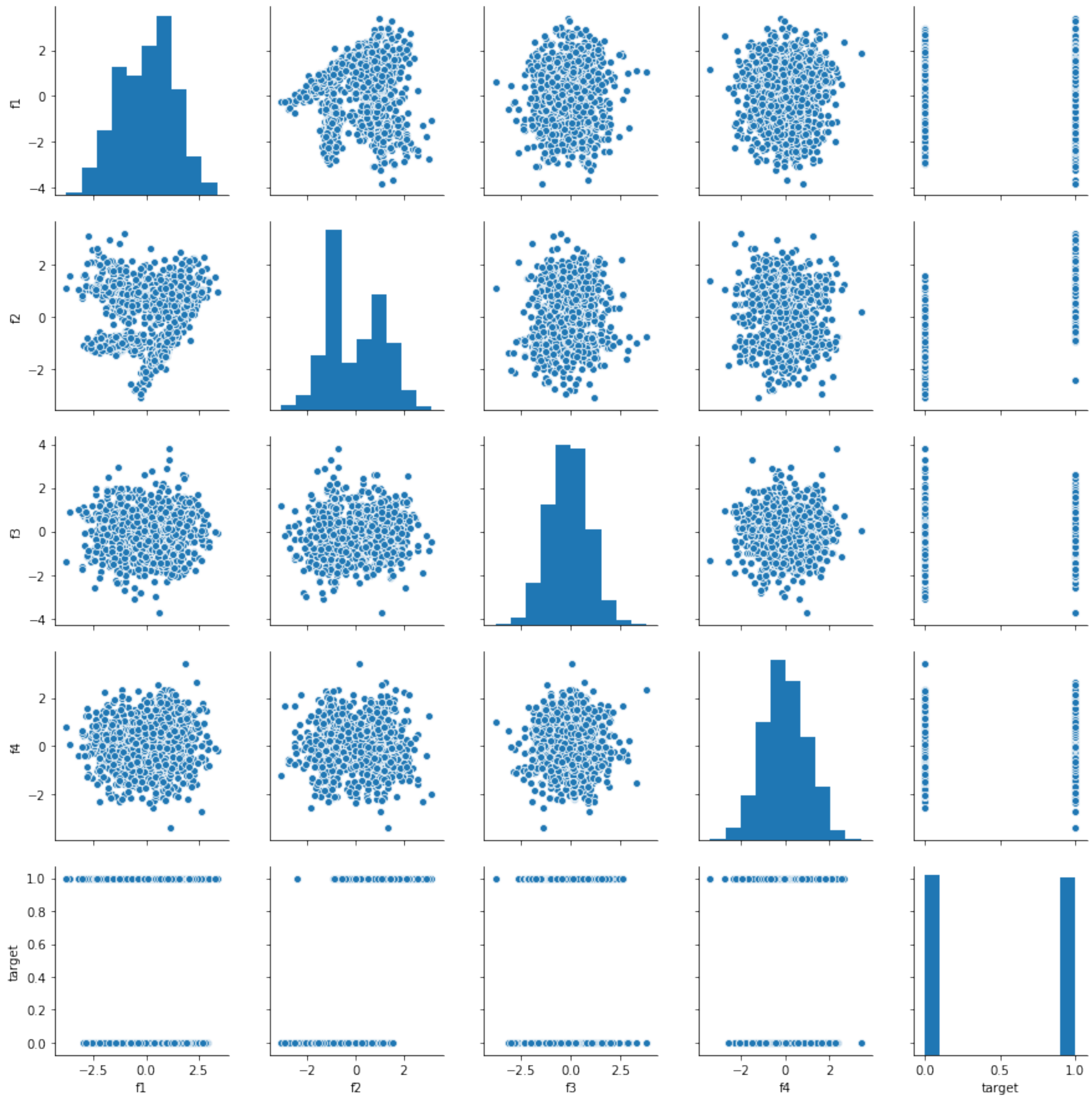
[42]: <seaborn.axisgrid.PairGrid at 0x7f927391a310>



[43]:
```
# adding class into the dataset
dfClass['target']=y
```

[44]:
```
sns.pairplot(dfClass)
```

[44]: <seaborn.axisgrid.PairGrid at 0x7f9275859f90>



[45]:
```
# Correlations
corMatrix = dfClass.corr(method='pearson')
```

```
predictions=corMatrix.iloc[-1][:-1]
predictions.sort_values(ascending=False)
```

[45]:
```
f2    0.831891
f1    0.041438
f3    0.035485
f4   -0.052465
Name: target, dtype: float64
```

Let us try several other specifications for the dataset, and look on the correlations.

[46]:
```
# Add redundant features

X, y = make_classification(n_samples=1000, n_features=4,n_informative=2,␣
 ↪n_redundant=2,random_state=0, shuffle=False)
dfClass=pd.DataFrame(X, columns=featureNames)
dfClass['target']=y

# Correlations
corMatrix = dfClass.corr(method='pearson')

predictions=corMatrix.iloc[-1][:-1]
predictions.sort_values(ascending=False)
```

[46]:
```
f2    0.837525
f4    0.723619
f3    0.334212
f1    0.035272
Name: target, dtype: float64
```

[47]:
```
# With shuffle
X, y = make_classification(n_samples=1000, n_features=4,n_informative=2,␣
 ↪n_redundant=0,random_state=0, shuffle=True)
dfClass=pd.DataFrame(X, columns=featureNames)
dfClass['target']=y

# Correlations
corMatrix = dfClass.corr(method='pearson')

predictions=corMatrix.iloc[-1][:-1]
predictions.sort_values(ascending=False)
```

[47]:
```
f4    0.831891
f3    0.041438
f1    0.035485
f2   -0.052465
Name: target, dtype: float64
```

3.2 Regression data

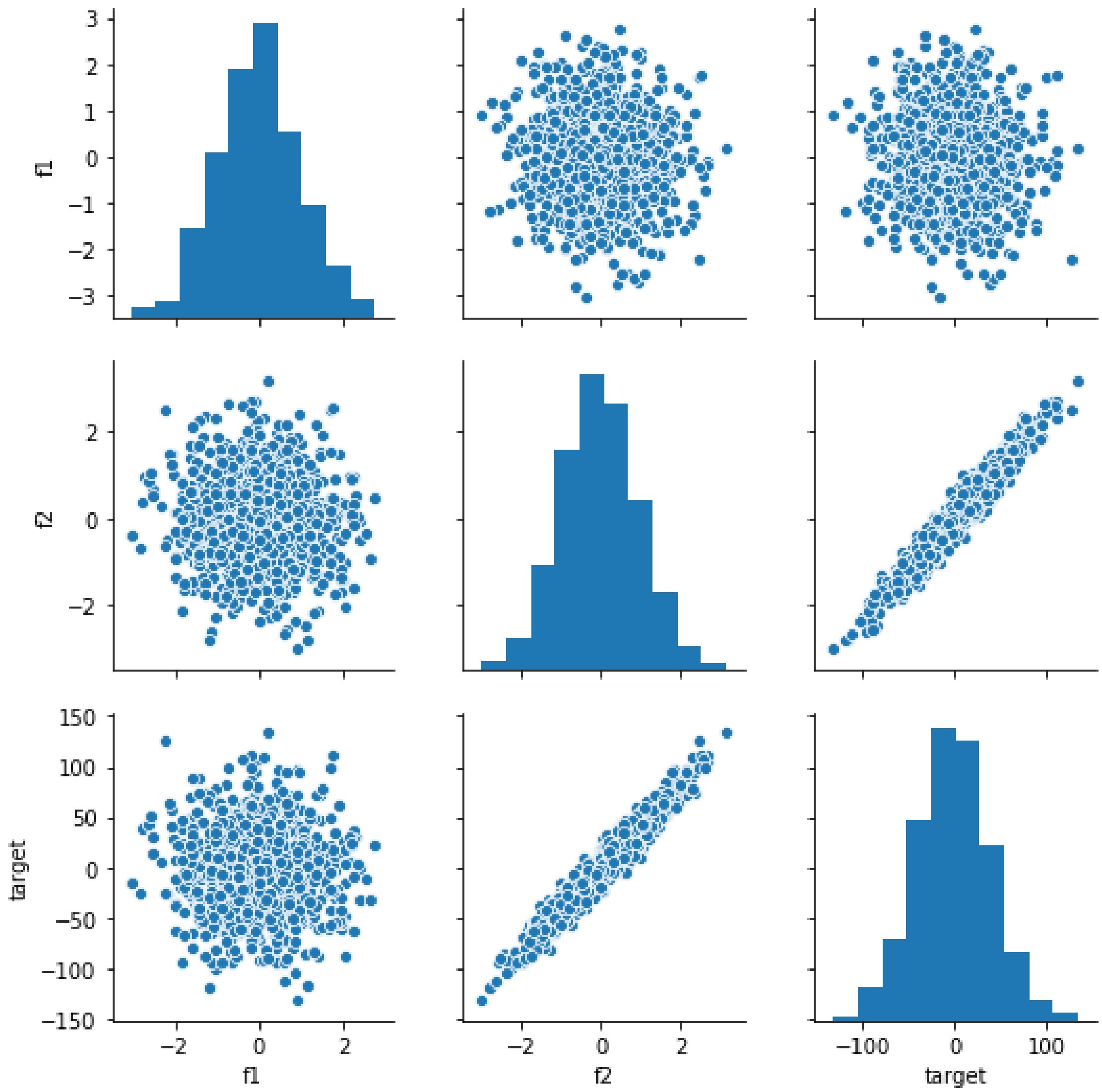
We may in certain cases need real valued data to test regression models.

[48]:
```
import numpy as np
from matplotlib import pyplot as plt

from sklearn.datasets import make_regression


n_samples = 1000

X, y, coef = make_regression(n_samples=n_samples, n_features=2,
                                      n_informative=1, noise=10,
                                      coef=True, random_state=0)
```

[49]:
```
featureNames = ['f1','f2']
dfReg=pd.DataFrame(X, columns=featureNames)
# Adding target
dfReg['target'] = y
```

[50]:
```
# Visualization

sns.pairplot(dfReg)
```

[50]:
```
<seaborn.axisgrid.PairGrid at 0x7f9278ece5d0>
```


f1
f2
target
3
2
1
0
−1
−2
−3
150
100
50
0
−50
−100
−150
−2
0
2
−100
0
100
f1
f2
target

```
[ ]:
```