DEPARTAMENTO CIÊNCIA E TECNOLOGIA

Estimation, Detection and Analysis II

01 - General Fundamentals

Association Rules (APRIORI)

Decision trees (classification and regression)

Simple and multiple linear regression

Initial setting

1. In Anaconda, create an *environment* called EDA2
2. In PyCharm , create a project called EDA2 and associate it with the created *environment* . In this link you can find a guide on how to associate a PyCharm project with an Anaconda *environment .*
3. Install, in the *environment* , the packages needed to solve the exercises

Association Rules (APRIORI)

The purpose of this exercise is to reproduce "Lab 6 – Data Mining : Association Rules" solved with Weka. First, you should recall the results of that exercise.

```
Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    <conf:(0.92)> lift:(1.27) lev:(0.03) [155] conv:(3.35)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    <conf:(0.92)> lift:(1.27) lev:(0.03) [149] conv:(3.28)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    <conf:(0.92)> lift:(1.27) lev:(0.03) [150] conv:(3.27)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    <conf:(0.92)> lift:(1.27) lev:(0.03) [159] conv:(3.26)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    <conf:(0.91)> lift:(1.27) lev:(0.04) [164] conv:(3.15)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    <conf:(0.91)> lift:(1.26) lev:(0.03) [151] conv:(3.06)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    <conf:(0.91)> lift:(1.26) lev:(0.03) [145] conv:(3.01)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(3)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    <conf:(0.91)> lift:(1.26) lev:(0.03) [156] conv:(3)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    <conf:(0.91)> lift:(1.26) lev:(0.04) [179] conv:(2.92)
```

Analyse the functioning of the APRIORI algorithm in Python at [https : //www.javatpoint.com/apriori-algorithm-in-machine-learning](https://www.javatpoint.com/apriori-algorithm-in-machine-learning)

In PyCharm 's EDA2 project , create a file called decisionrules_supermarket.py to hold the code for the following instructions:

1. Install the apyori package

2. Read the supermarket.csv file found in Moodle:

```
import pandas as pd # To read data
data = pd.read_csv (' file_path /supermarket.csv')
```

In this case, the file is read using the read_csv function from the pandas library. Documentation for this function can be found at this link and general documentation for the pandas library can be found at this link .

3. Convert the dataset "data" to the format expected by the algorithm:

```
records = []
for index , row in data.iterrows ():
record = []
for c in data.columns :
        if row [c] == 't':
            record.append (c)
        if c=="total":
            record.append ("total_"+ row [c])
    records.append (record)
```

Excerpt from the list (records[0], corresponding to the transaction recorded in the first line of the csv):

```
['baby needs ', ' bread and cake ', ' baking needs ', ' juice-sat-cord-ms ', ' biscuits ', ' canned vegetables ', ' cleaners-polishers ', ' coffee ', ' sauces-gravy-pkle ', ' confectionary ', ' dishcloths-scour ', ' frozen foods', ' razor blades ', ' party snack foods', ' tissues-paper prd ', ' wrapping ', ' mens toiletries ', ' cheese ', ' milk-cream ', ' margarine ', ' small goods ', ' fruit ', ' vegetables ', 'department122', '750ml white nz ', ' total_high ']
```

4. Run the algorithm and check the result

```
from apyori import a priori

rules = apriori (records, min_support =0.1, min_confidence =0.9, min_length =2, min_lift =1.25)
listrules = list (rules)
```

5. view the result

```
for item in listrules :
   print(item)
```

Excerpt from the result (first rule found):

```
RelationRecord ( items = frozenset ({' bread and cake ', ' total_high ', ' baking needs ', ' beef '}), support =0.13140263669764427, ordered_statistics = [ OrderedStatistic ( items_base = frozenset ({' total_high ', ' baking needs ', ' beef '}), items_add = frozenset ({' bread and cake '}), confidence =0.9007407407407408, lift =1.2515697920142366)])
```

This format is difficult to read. We can "split" the result to make it more readable:

```
RelationRecord (
       items = frozenset ({' bread and cake ', ' total_high ', ' baking needs ', ' beef '}),
       support =0.13140263669764427,
       ordered_statistics = [ OrderedStatistic (
              items_base = frozenset ({' total_high ', ' baking needs ', ' beef '}),
              items_add = frozenset ({' bread and cake '}),
              confidence =0.9007407407407408,
              lift =1.2515697920142366)]    )
```

Or, to make it even simpler, we can use the following code to format the result

```
for item in listrules :
    pair = item[0]
    items = [x for x in pair ]
    ant = str ( list (item[2][0][0]))[1:-1]
    cons = str ( list (item[2][0][1]))[1:-1]
    print("Rule: {" + ant + "} -> {" + cons + "}")
    print(" Support : " + str (item[1]))
    print(" Confidence : " + str (item[2][0][2]))
    print(" Lift : " + str (item[2][0][3]))
    print("=============================================")
```

Excerpt from the result (first rule found, reformatted):

```
Rule: {' beef ', ' baking needs ', ' total_high '} -> {' bread and cake '}
Support : 0.13140263669764427
Confidence : 0.9007407407407408
Lift : 1.2515697920142366
```

6. interpret the result

The first rule says that transactions containing ' beef ', ' baking needs ' and ' total_high ' should probably also have ' bread and cake '. This rule has:

- 13% support : The items ' beef ', ' baking needs ', ' total_high ' and ' bread and cake ' appear together in 13% of transactions
- 90% Confidence : Transactions with ' beef ', ' baking needs ' and ' total_hig have a 90% chance of also containing ' bread and cake '

```
•   Lift of 1.25%: as the Lift is greater than 1, the antecedent and consequent appear
    more often together than expected, that is: the occurrence of the antecedent has a
    positive effect on the occurrence of the consequent
```

decision trees

Classification

The purpose of this exercise is to reproduce “Lab 4 – Data Mining : Classification #1” solved with Weka . At first, you should recall the results of that exercise.

Model results:

```
J48 pruned tree
------------------

feathers = FALSE
|   milk = TRUE: mammal (41.0)
|   milk = FALSE
|   |   backbone = TRUE
|   |   |   fins = FALSE
|   |   |   |   tail = FALSE: amphibian (3.0)
|   |   |   |   tail = TRUE: reptile (6.0/1.0)
|   |   |   fins = TRUE: fish (13.0)
|   |   backbone = FALSE
|   |   |   airborne = FALSE
|   |   |   |   predator = TRUE: invertebrate (8.0)
|   |   |   |   predator = FALSE
|   |   |   |   |   legs <= 2: invertebrate (2.0)
|   |   |   |   |   legs > 2: insect (2.0)
|   |   |   airborne = TRUE: insect (6.0)
feathers = TRUE: bird (20.0)

Number of Leaves  :   9

Size of the tree :    17
```

model evaluation

```
Correctly Classified Instances          93               92.0792 %
Incorrectly Classified Instances         8                7.9208 %
Kappa statistic                          0.8955
Mean absolute error                      0.0225
Root mean squared error                  0.14
Relative absolute error                 10.2478 %
Root relative squared error             42.4398 %
Total Number of Instances              101

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     mammal
                 1,000    0,011    0,929      1,000    0,963      0,958    0,994     0,929     fish
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     bird
                 0,800    0,033    0,727      0,800    0,762      0,735    0,986     0,812     invertebrate
                 0,625    0,032    0,625      0,625    0,625      0,593    0,920     0,677     insect
                 0,750    0,000    1,000      0,750    0,857      0,862    0,872     0,760     amphibian
                 0,600    0,010    0,750      0,600    0,667      0,656    0,793     0,420     reptile
Weighted Avg.    0,921    0,008    0,922      0,921    0,920      0,914    0,976     0,908

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 41  0  0  0  0  0  0 |  a = mammal
  0 13  0  0  0  0  0 |  b = fish
  0  0 20  0  0  0  0 |  c = bird
  0  0  0  8  2  0  0 |  d = invertebrate
  0  0  0  3  5  0  0 |  e = insect
  0  0  0  0  0  3  1 |  f = amphibian
  0  1  0  0  1  0  3 |  g = reptile
```

In PyCharm 's EDA2 project , create a file called decisiontree_zoo.py to hold the code for the following instructions:

1. Install the sklearn package

2. Read the zoo.csv file found in Moodle:

```
import pandas as pd # To read data
data = pd.read_csv (' file_path /zoo.csv')
```

3. Define which variables are independent (X) and dependent (y)

```
X = data[[' hair ', ' feathers ', ' eggs ', ' milk ', ' airborne ', ' aquatic ', ' predator ',
        ' toothed ', ' backbone ', ' breathes ', ' venomous ', 'fins', ' legs ', ' tail ',
        ' domestic ', ' catsize ']]
Y = data[[' type ']]
```

4. Create and fit the decision tree

```
from sklearn import tree
clf = tree.DecisionTreeClassifier ()
clf = clf.fit (X, Y)
```

5. View the template created

```
from sklearn.tree import export_text
r = export_text ( clf , feature_names =[' hair ', ' feathers ', ' eggs ', ' milk ', ' airborne ',
```

```
        ' aquatic ', ' predator ', ' toothed ', ' backbone ', ' breathes ', ' venomous ',
'fins',
        ' legs ', ' tail ', ' domestic ', ' catsize '])
print(r)
```

Result

```
|--- milk <= 0.50
| |--- feathers <= 0.50
| | |--- ends <= 0.50
| | | |--- backbone <= 0.50
| | | | |--- airborne <= 0.50
| | | | | |--- predator <= 0.50
| | | | | | |--- legs <= 3.00
| | | | | | | |--- class : invertebrate
| | | | | | |--- legs > 3.00
| | | | | | | |--- class : insect
| | | | | |--- predator > 0.50
| | | | | | |--- class : invertebrate
| | | | |--- airborne > 0.50
| | | | | |--- class : insect
| | | |--- backbone > 0.50
| | | | |--- aquatic <= 0.50
| | | | | |--- class : reptile
| | | | |--- aquatic > 0.50
| | | | | |--- breathes <= 0.50
| | | | | | |--- class : reptile
| | | | | |--- breathes > 0.50
| | | | | | |--- class : amphibian
| | |--- purposes > 0.50
| | | |--- class : fish
| |--- feathers > 0.50
| | |--- class : bird
|--- milk > 0.50
| |--- class : mammal
```

6. Draw the decision tree

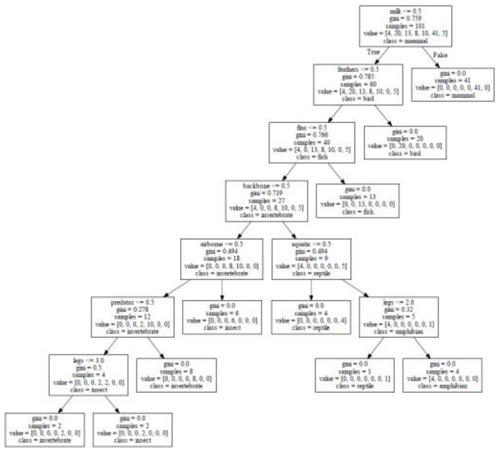
```
dot_data = tree.export_graphviz ( clf , out_file = None ,
                                feature_names =[' hair ', ' feathers ', ' eggs ', ' milk
',
        ' airborne ', ' aquatic ', ' predator ', ' toothed ', ' backbone ',
        ' breathes ', ' venomous ', 'fins', ' legs ', ' tail ',
        ' domestic ', ' catsize '],
                                class_names = clf.classes_ )

import pydotplus
graph = pydotplus.graph_from_dot_data ( dot_data )

from IPython.display import Image

Image ( graph.create_png ())
graph.write_pdf ("dt.pdf")
```

7. Check the result saved in the dt.pdf file within the EDA2 project



Note: The tree created is slightly different from the one created with Weka , which is due to the intrinsic randomness of the decision tree algorithm.

8. Evaluate the model using cross- validation (10-fold)

```
from sklearn import tree
clf = tree.DecisionTreeClassifier ()

from sklearn.model_selection import cross_val_predict
y_pred = cross_val_predict ( clf , X, Y, cv =10)

from sklearn.metrics import classification_report , confusion_matrix
print( confusion_matrix (Y, y_pred ))
print( classification_report (Y, y_pred ))
from sklearn.model_selection import cross_validate
from statistics import mean
cv_results = cross_validate ( clf , X, Y, cv =10)
print(" Accuracy :", mean ( cv_results [' test_score ']))
```

Results:

```
[[ 3 0 0 0 0 0 1]
[ 0 20 0 0 0 0 0]
[ 0 0 13 0 0 0 0]
[ 0 0 0 7 1 0 0]
[ 0 0 0 1 9 0 0]
[ 0 0 0 0 0 41 0]
[ 1 0 0 1 0 0 3]]
              precision    recall f1-score support

   amphibian 0.75 0.75 0.75 4
        bird 1.00 1.00 1.00 20
        fish 1.00 1.00 1.00 13
      insect 0.78 0.88 0.82 8
invertebrate 0.90 0.90 0.90 10
      mammal 1.00 1.00 1.00 41
     reptile 0.75 0.60 0.67 5

    accuracy 0.95 101
macro avg 0.88 0.88 0.88 101
weighted avg 0.95 0.95 0.95 101

Accuracy : 0.93
```

Note: As the created tree is slightly different from the one created with Weka , its performance is also different

Regression

The objective of this exercise is to reproduce "Lab 1 – Data Mining : Regression" solved with Weka with the LinearRegression model , but with decision trees for regression. At first, you should recall the results of that exercise.

Model results:

```
Linear Regression Model

SellingPrice =

    -26.6882 * HouseSize +
      7.0551 * LotSize +
  43166.0767 * Bedrooms +
  42292.0901 * Bathroom +
 -21661.1208

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.9945
Mean absolute error                   4053.821
Root mean squared error               4578.4125
Relative absolute error                 13.1339 %
Root relative squared error             10.51   %
Total Number of Instances                7
```

Forecast results:

```
SellingPrice = (-26.6882    * 3198) +
    (7.0551      * 9669) +
    (43166.0767 * 5) +
    (42292.0901 * 1)
    - 21661.1208

 SellingPrice = 219,328
```

PyCharm 's EDA2 project , create a file called decisiontree_regression_House.py to hold the code for the following instructions:

1. Read the House.csv and House_new files found in Moodle:

```
import pandas as pd # To read data
data = pd.read_csv (' file_path /House.csv')
data_new = pd.read_csv (' file_path /House_new.csv')
```

2. Define which variables are independent (X) and dependent (y)

```
X = data[[' HouseSize ', ' LotSize ', ' Bedrooms ', 'Granite', ' Bathroom ']]
Y = data[[' SellingPrice ']]
```

3. Create and fit the decision tree

```
from sklearn.tree import DecisionTreeRegressor
regr_1 = DecisionTreeRegressor ()
regr_1.fit(X, Y)
```

4. View the template created

```
from sklearn.tree import export_text
r = export_text (regr_1, feature_names =[' HouseSize ', ' LotSize ', ' Bedrooms ',
'Granite', ' Bathroom '])
print(r)
```

Result

```
|--- LotSize <= 17075.00
|   |--- HouseSize <= 2690.00
|   |   |--- Bathroom <= 0.50
|   |   |   |--- value : [189900.00]
|   |   |--- Bathroom > 0.50
|   |   |   |--- value : [195000.00]
|   |--- HouseSize > 2690.00
|   |   |--- HouseSize <= 3388.00
|   |   |   |--- HouseSize <= 3115.00
|   |   |   |   |--- value : [230000.00]
|   |   |   |--- HouseSize > 3115.00
|   |   |   |   |--- value : [224900.00]
|   |   |--- HouseSize > 3388.00
|   |   |   |--- Bedrooms <= 5.50
|   |   |   |   |--- value : [197900.00]
|   |   |   |--- Bedrooms > 5.50
|   |   |   |   |--- value : [205000.00]
|--- LotSize > 17075.00
|   |--- value : [325000.00]
```

5. Draw the decision tree

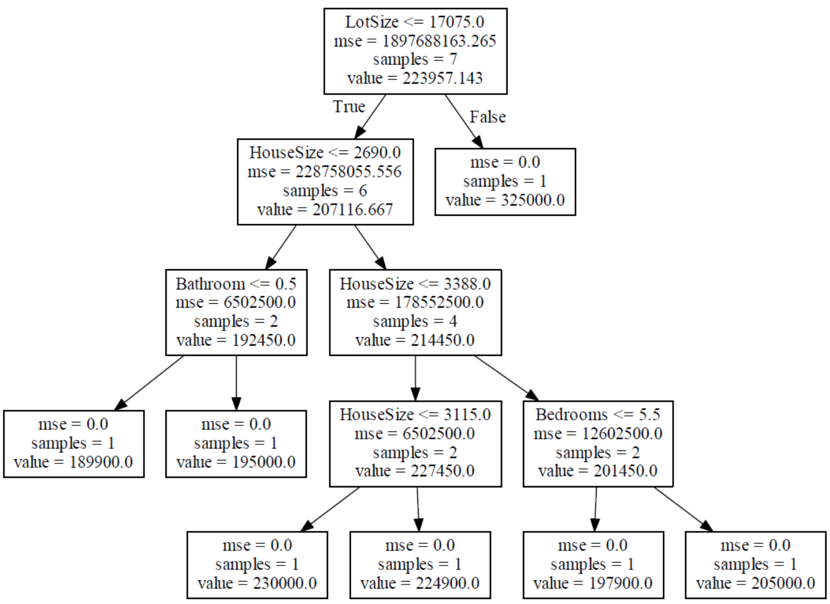
```
from sklearn import tree
# Create DOT data
dot_data = tree.export_graphviz (regr_1, out_file = None ,
                                feature_names =[' HouseSize ', ' LotSize ', ' Bedrooms
', 'Granite', ' Bathroom '],
                                class_names =regr_1.classes_)

# draw graph
import pydotplus
graph = pydotplus.graph_from_dot_data ( dot_data )

# show graph
from IPython.display import Image

Image ( graph.create_png ())
graph.write_pdf ('dt_regr.pdf')
```

6. Check the result saved in the dt_regression.pdf file within the EDA2 project



7. Evaluate the model

```
from sklearn.metrics import mean_squared_error
predictions = regr_1.predict(X=data[[' HouseSize ', ' LotSize ', 'Bedrooms', 'Granite',
'Bathroom']])
mse = mean_squared_error (data[[' SellingPrice ']], predictions)
print("MSE:", mse )
```

Result:

```
MSE: 0.0
```

Note: The result gives 0 error, because the model is *overfitted* to the training set

8. Apply the model to new data

```
predictions = regr_1.predict(X= data_final_new [[' HouseSize ', ' LotSize ', ' Bedrooms
',
        'Granite', ' Bathroom ']])
print(" Predicted price : % d\n"% predictions )
```

9. Result

```
Predicted price : 224900
```

Linear Regression

Simple Linear Regression

"Manual" Process

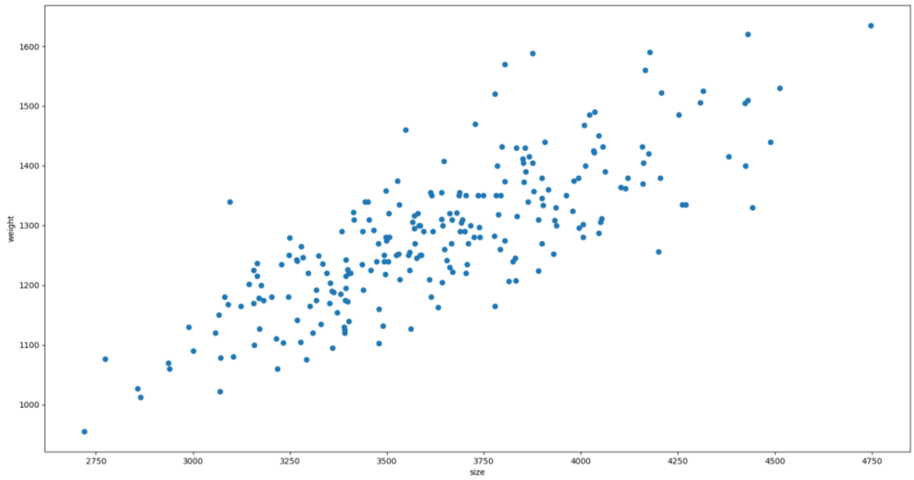
1. Read the sizeweight.csv file (found in Moodle)

```
import pandas as pd
data = pd.read_csv (' file_path /sizeweight.csv')
```

2. Draw the graph of the data

```
import matplotlib.pyplot as plt
plt.rcParams [' figure.figsize '] = (20.0, 10.0)
plt.scatter (' size ', ' weight ', data=data)
plt.xlabel (' size ')
plt.ylabel (' weight ')
plt.show ()
```

Result:



3. Define the independent (X) and dependent (Y) variables

```
X = data[' size ']. values
Y = data[' weight ']. values
```

4. Calculate β $_0$ and β $_1$ and show the equation of the line

```
import numpy as np

mean_x = np.mean (X)
mean_y = np.mean (Y)
n = len (X)

number = 0
name = 0
for i in range(n):
    numer += (X[i] - mean_x ) * (Y[i] - mean_y )
    denom += (X[i] - mean_x ) ** 2
b1 = number / denomination
b0 = mean_y - (b1 * mean_x )

print('y =',b1,'* x +', b0)
```

Result:

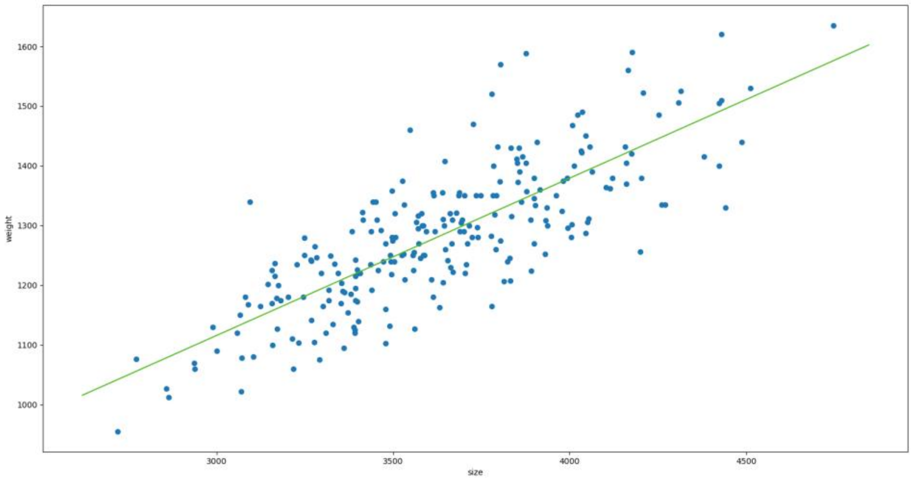
```
y = 0.26342933948939945 * x + 325.57342104944223
```

5. Draw the graph of the data and the line obtained

```
max_x = np.max (X) + 100
min_x = np.min (X) - 100
x = np.linspace ( min_x , max_x , 1000)
y = b0 + b1 * x

plt.rcParams [' figure.figsize '] = (20.0, 10.0)
plt.scatter (' size ', ' weight ', data=data)
plt.xlabel (' size ')
plt.ylabel (' weight ')
plt.plot (x,y,color='#52b920')
plt.show ()
```

Result:



6. Evaluate the model: calculate the R_2

```
ss_t = 0
ss_r = 0
for i in range( X.size ):
    y_pred = b0 + b1 * X[i]
    ss_t += (Y[i] - mean_y ) ** 2
    ss_r += (Y[i] - y_pred ) ** 2
r2 = 1 - ( ss_r / ss_t )
print("r2 (manual) =",r2)
```

Result:

```
r2 (manual) = 0.6393117199570003
```

Using " scikit learn "

7. import the library

```
from sklearn.linear_model import LinearRegression
```

8. Reformat the independent variable

```
X = X.reshape (( X.size , 1))
```

9. create the model

```
reg = LinearRegression ()
```

10. Fit the model

```
reg = reg.fit (X, Y)
```

11. Calculate model predictions

```
Y_pred = reg.predict (X)
```

12. Calculate and show the R^2

```
r2_score = reg.score (X, Y)
print("r2 ( LinearRegression ) =",r2_score)
```

Result:

```
r2( LinearRegression ) = 0.639311719957
```

Multiple Linear Regression

Example replication with Weka

The purpose of this exercise is to reproduce "Lab 1 – Data Mining : Regression" solved with Weka . At first, you should recall the results of that exercise.

Model results:

```
Linear Regression Model

SellingPrice =

    -26.6882 * HouseSize +
      7.0551 * LotSize +
  43166.0767 * Bedrooms +
  42292.0901 * Bathroom +
 -21661.1208

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correlation coefficient                  0.9945
Mean absolute error                   4053.821
Root mean squared error               4578.4125
Relative absolute error                 13.1339 %
Root relative squared error             10.51   %
Total Number of Instances                7
```

Forecast results:

```
SellingPrice = (-26.6882   * 3198) +
    (7.0551      * 9669) +
    (43166.0767 * 5) +
    (42292.0901 * 1)
    - 21661.1208

 SellingPrice = 219,328
```

PyCharm 's EDA2 project , create a file called linearRegression_House.py to hold the code for the following instructions:

13. Read the House.csv and House_Final_new.csv files (found in Moodle)

```
import pandas as pd
data = pd.read_csv (' file_path /House.csv')
data_final_new = pd.read_csv (' fic_path /House_final_new.csv')
```

14. Create a [1]linear regression model for the original dataset .

```
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression ()
linear_regressor.fit  (data[['  HouseSize  ',  '  LotSize  ',  '  Bedrooms  ',  'Granite',  '
Bathroom ']], data[' SellingPrice '])
```

The creation of the linear regression model is done using the creation of a LinearRegressor , followed by the execution of the fit of the model. For that, you need to import the LinearRegression library from sklearn.linear_model . Important documentation for this step can be found at this link . The model is intended to consider the independent variables HouseSize , LotSize , Bedrooms , Granite, Bathroom and the dependent variable SellingPrice .

15. model's coefficients and intercept .

1

```
print(" coefs :", linear_regressor.coef _, sep =" ")
print(" intercept :", linear_regressor.intercept _, sep =" ")
```

The result obtained should be as follows

```
coefs : [-2.69307835e+01 6.33452410e+00 4.42937606e+04 7.14067629e+03 4.31791999e+04]
intercept : -21739.29666506665
```

Coefficients are in scientific notation. To avoid this, we add the following lines before the prints:

```
import numpy as np
np.set_printoptions ( formatter ={' float_kind ': '{:f}'. format })
```

In this way, the result obtained is the following:

```
coefs : [-26.930784 6.334524 44293.760584 7140.676293 43179.199889]
intercept : -21739.29666506665
```

The coefficients are presented in the order of the attributes. This means that the regression equation is as follows:

```
SellingPrice = -26.930784 * HouseSize +
6.334524 * LotSize +
44293.760584 * Bedrooms +
7140.676293 * Granite +
43179.199889 * Bathroom +
-21739.29666506665
```

As we can see, this model (unlike the model generated by Weka), considers the “Granite” attribute. To stop considering this attribute, we have to eliminate it from the fit function in point 2:

```
from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression ()
linear_regressor.fit (data[[' HouseSize ', ' LotSize ', ' Bedrooms ', ' Bathroom ']],
data[' SellingPrice '])
```

When we run the code again, we get the following output:

```
coefs : [-26.688240 7.055124 43166.076944 42292.090237]
intercept : -21661.12129661304
```

Which means that the following regression equation was obtained:

```
SellingPrice = -26.688240 * HouseSize +
7.055124 * LotSize +
43166.076944 * Bedrooms +
42292.090237 * Bathroom +
-21661.12129661304
```

Which corresponds, approximately, to the equation obtained in Weka

16. Get some model performance metrics

```
from sklearn.metrics import mean_absolute_error , mean_squared_error
import math
predictions = linear_regressor.predict (data[[' HouseSize ', ' LotSize ', ' Bedrooms ',
' Bathroom ']])
print("\n=METRICS =")
print('   mean_absolute_error   :   ',   mean_absolute_error   (data['   SellingPrice   '],
predictions ), sep ="")
print(' Root mean_squared_error : ', math.sqrt ( mean_squared_error (data[' SellingPrice
'], predictions )), sep ="")
```

The metrics to be calculated are the Mean Absolute Error and Root mean Squared Error. For that, it is necessary to import the functions mean_absolute_error and mean_squared_error from the sklearn.metrics library , whose documentation can be found at this link . How the mean_squared_error function calculates the Mean Squared Error, it is necessary to apply the square root to it, using the sqrt function of the math library . The output will be:

```
= METRICS =
mean_absolute_error : 4053.8210607484966
root mean_squared_error : 4578.412476004145
```

These values are similar to those obtained with Weka

17. Calculate and visualize model predictions for new data.

```
predictions2 = linear_regressor.predict ( data_final_new [[' HouseSize ', ' LotSize ', '
Bedrooms ', ' Bathroom ']])
print("\ nmodel final predictions :", predictions2, sep =" ")
```

Predictions are calculated using the predict function . The output is the following:

```
model final predictions : [219328.357193]
```

This result is similar to that obtained with Weka .