Modern Methods of Data Analysis

Homework Project

Students:

Vitaly Sopov

Michael Florinsky

HSE

Faculty of Computer Science

Data Science

2019

1. Choosing the Dataset

For this project we selected the dataset with information about modern GPUs (graphics processing units) used in modern PCs. The dataset contains information about these GPUs' characteristics and details. It can be found on Kaggle: [https://www.kaggle.com/iliassekkaf/computerparts#All\_GPUs.csv](https://www.kaggle.com/iliassekkaf/computerparts" \l "All_GPUs.csv)

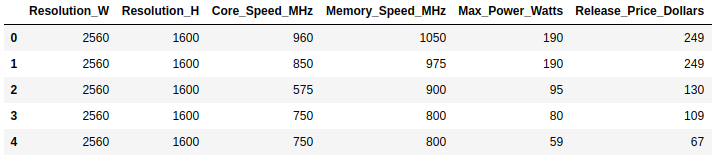
This dataset was chosen because while it is quite simple in its nature, it contains a lot of diverse types of features: there are some categorical features (which we unfortunately ignore), a lot of numerical features (a lot of them can be interesting as target features in regression problems) and time features which can be used in temporal analysis. Besides, we were interested in this dataset because we love computers and everything about them.

Original dataset contains 3405 rows of 34 features. We selected 9 meaningful features: Resolution\_W (maximal screen resolution, width, in pixels), Resolution\_H (maximal screen resolution, height, in pixels), Core\_Speed\_MHz (GPU core speed), Memory\_Speed\_MHz (GPU memory speed), Max\_Power\_Watts (power requirements), Release\_Price\_Dollars (initial price), Memory\_MB (amount if built-in GPU memory), Memory\_Bus\_Bit (size of the transport bus), Memory\_Bandwidth\_GB\_Sec (the bandwidth of the GPU memory).

Initially there were a lot of NaN values, also some numerical features were represented as strings, for example, "2560x1600" as screen resolution. We transformed this single feature into two numerical: width and height. Some rows contained incorrect values - that could not be represented as number, even if the feature is numerical. We deleted those values. All the NaNs were also deleted.

After the preprocessing the final dataset contained 220 rows and 9 features.

Example of dataset chunk:



The problems which can be attempted to be solved using this dataset include the prediction of the characteristics of future GPUs, analysis of GPUs characteristics to their price, determining some of their characteristics based on the other characteristics and general analysis of correlation of characteristics, and their evolution in time.

2. K-means

All code was written in Python 3 in jupyter environment. Numpy and pandas libraries were used for matrix computations and dataset processing. Matplotlib library was used for displaying plots. Sklearn library was used for clustering and linear regression.

The code is stored in a public GitHub repository (<https://github.com/PreFX48/modern_data_analysis/>). In the following sections we will give links to the appropriate sections of code.

First of all, we preprocess the raw dataset (we will do this in every task). It is done in file hw\_2\_3\_6.ipynb, in section "HW\_1", in cell 2: [hw\_2\_3\_6.ipynb](https://github.com/PreFX48/modern_data_analysis/blob/master/hw_2_3_6.ipynb)

Let's select features, that have wide range of values. For example, screen resolution is not suitable: it looks like categorical feature with poor values range. Afted histogram analysis it is clear, that 'Release\_Price\_Dollars', 'Core\_Speed\_MHz', 'Max\_Power\_Watts', 'Memory\_Bandwidth\_GB\_Sec' can be taken. Histograms are shown on the next page.

Изображение выглядит как снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана

Автоматически созданное описание

Then we use K-means algorithm on these features for 5 clusters and 9 clusters. The code for clusterization can be found in file *[hw\_2\_3\_6.ipynb](https://github.com/PreFX48/modern_data_analysis/blob/master/hw_2_3_6.ipynb)* in section "HW 2" in cells 6 to 10. There we use K-Means provided by sklearn library. For making our experiments reproducible we fix the random seed which is used for selecting initialization states for K-means algorithm. Let’s compute delta as 100 \* (CMean - GMean) / GMean. The results are the following:

Best result for 5 clusters on 10 random inits:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cluster N | Elements in cluster | Delta 1 | Delta 2 | Delta 3 | Delta 4 |
| 1 | 40 | 63.26 | -1.13 | 55.27 | 54.49 |
| 2 | 30 | 2.25 | -0.02 | 11.44 | 62.67 |
| 3 | 27 | 36.08 | -23.54 | 38.8 | 7.88 |
| 4 | 41 | -48.37 | -19.73 | -50.44 | -65.27 |
| **5** | **82** | **-19.37** | **18.17** | **-18.8** | **14.27** |

Inertia = 11.214650925809542

The best cluster here is cluster 5: it has 81 elements with not so big delta values (by absolute). Other clusters contain too big or too small deltas on some features.

Best result for 9 clusters on 10 random inits:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cluster N | Elements in cluster | Delta 1 | Delta 2 | Delta 3 | Delta 4 |
| 1 | 39 | -27.19 | 25.66 | -33.36 | 11.50 |
| 2 | 22 | 36.54 | -27.50 | 39.26 | 11.94 |
| **3** | **26** | **-1.32** | **16.42** | **10.55** | **23.94** |
| 4 | 9 | -24.27 | -34.39 | -22.57 | -59.09 |
| 5 | 29 | 2.64 | 0.07 | 11.24 | 64.34 |
| 6 | 27 | -56.51 | -18.23 | -61.23 | -71.97 |
| 7 | 20 | 96.49 | -0.04 | 47.93 | 65.65 |
| 8 | 25 | 30.84 | -2.99 | 57.44 | 36.66 |
| 9 | 23 | -32.26 | 2.29 | -30.47 | -24.90 |

Inertia = 6.786299682142113

The best cluster on this split is cluster 3: on all features it has small absolute delta values, comparing to other clusters in general. As we can see, it both cases delta values are within [-100, 100] range. So, it is good result. Dataset cleanup is very significant in this case.

3. Bootstrapping

The code for boostrapping can be found in file [hw\_2\_3\_6.ipynb](https://github.com/PreFX48/modern_data_analysis/blob/master/hw_2_3_6.ipynb) in section "HW 3" in cells 11 to 13.

We use bootstrapping on 5-cluster case. Histograms with the results of bootstrapping are shown below.

Confidence interval is computed for "Release\_Price\_Dollars" feature on grand mean (95%):

Изображение выглядит как снимок экрана

Автоматически созданное описание

pivotal: [176.74, 194.97]

non-pivotal: [174.73, 197.20]

Regarding feature min/max values this difference in small, so pivot and non-pivot bounds are quite similar. If we’ll recall, that these values represented in dollars, we can be sure that this is small difference.

Let’s select clusters 3 and 4 for the next computations. Comparing cluster-means for "Release\_Price\_Dollars" feature on clusters 3 and 4:

cluster3 mean: 252.78

cluster4 mean: 95.78

As we can see, they are completely different. So, these clusters demonstrate completely different values inside them. Cluster 3 is for expensive devices with high performance, cluster 4 is for cheaper devices.

Comparing mean for "Release\_Price\_Dollars" feature on cluster 3 with grand mean:

cluster3 mean: 252.78

grand mean: 185.85

As we can see, they are completely different again, but closer. It is OK, regarding clusters difference.

Изображение выглядит как снимок экрана

Автоматически созданное описаниеИзображение выглядит как снимок экрана

Автоматически созданное описание

4. Contingency Matrix

The code for this section can be found here: [contingency.ipynb](https://github.com/PreFX48/modern_data_analysis/blob/master/contingency.ipynb)

First of all we generate thee nominal features for this dataset. First feature will be a taxon feature, where we split our dataset into 4 groups of the same size.

The other two features will be created from Core\_Speed and Memory\_Bandwidth numerical features by grouping them into 4 buckets each.

Изображение выглядит как текст

Автоматически созданное описание

Looking at Core\_Speed histogram, we define bucket bounds as

[0, 780, 1050, 1450, 1800]

For Memory\_Bandwidth histogram, which is on the next page, we define bucket bounds as

[0, 140, 290, 550, 1100]

Изображение выглядит как кроссворд

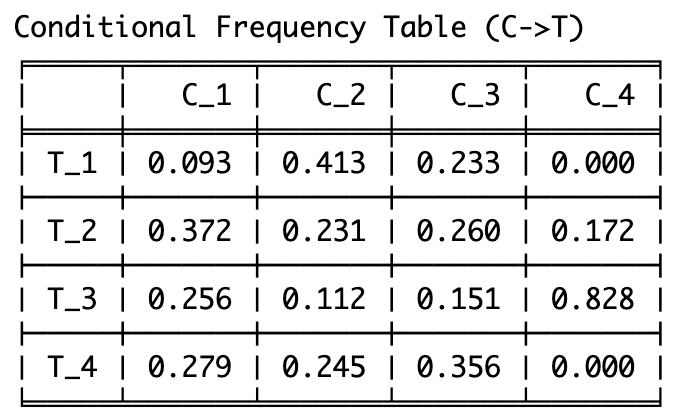
Автоматически созданное описание

The code for displaying histograms and generating splits can be found in this file: [contingency.ipynb](https://github.com/PreFX48/modern_data_analysis/blob/master/contingency.ipynb)

in section "Generating Nominal Features" in cells 2 to 6.

Then we build the required matrices. Here and lower we define C as CoreSpeed feature, M as MemoryBandwidth feature and T as taxonomial feature and will refer to their contingency tables as (C -> T) and (M -> T) tables.

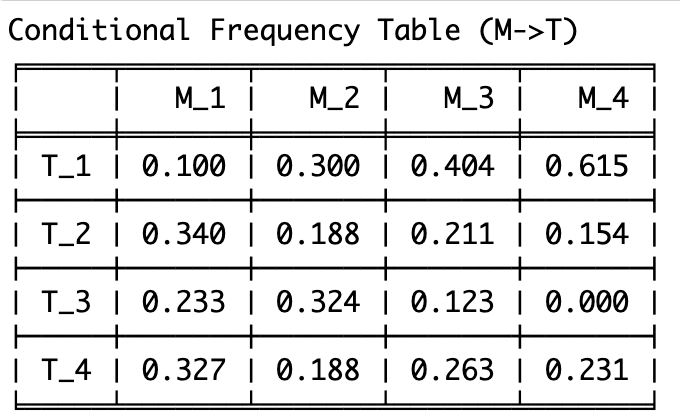
Conditional Frequency Table for (C->T) is computed in section "Building Conditional Frequency Tables" in cells 7-9. The computations use numpy library for matlab-like matrix operations.



Here we can see that C4 (high core speed) implies with high (83%) probability taxon T3, with some much lower probability (17%) – taxon T2 and with zero probability – taxons T1 and T4.

T1 is also quite improbable (with probability of 9%) to be for any sample from C1.

Conditional Frequency Table for (M->T) is computed in section "Building Conditional Frequency Tables" in cells 7, 8, 10.

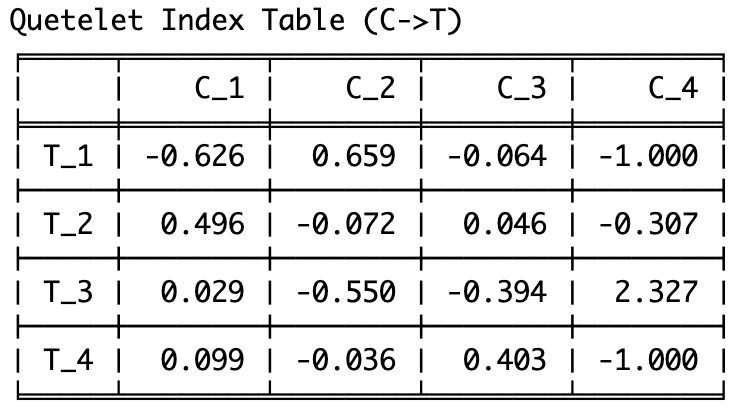


Here we can see that M4 (high memory bandwidth) implies with 62% probability taxon T1, which is a high enough probability.

It is also seen that M4 with 100% probability guarantees anything but T3.

Besides, with rather high probability (90%) GPUs with the lowers memory bandwidth will not be in T1.

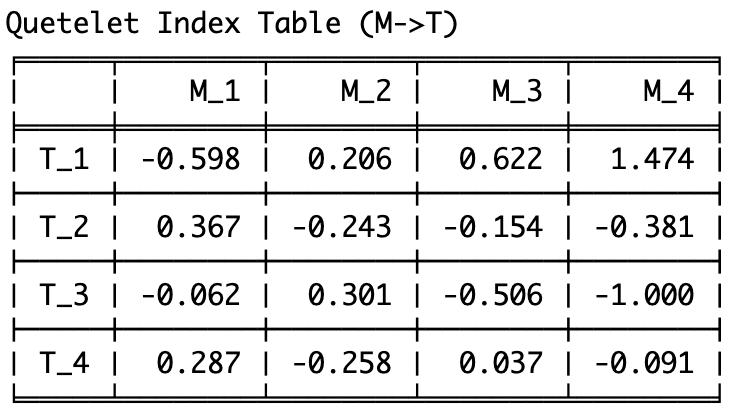
Quetelet Index Table for (C->T) is computed in section "Building Quetelet Index Tables" in cell 11.



Here we can see that after choosing sample from C4 or T3, with much higher probability it will be a sample from T3 and C4 accordingly. Probability of C4 and T1,T2,T4 are accordingly lowered.

Probability of simultaneously choosing sample from T1 and C1 is also lowered.

Quetelet Index Table for (M->T) is computed in section "Building Quetelet Index Tables" in cell 12.

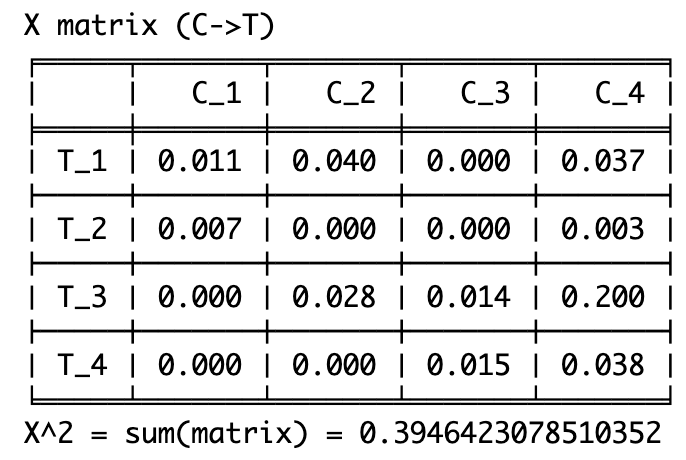


Probability of choosing sample from both T1 and M4 is much higher than normal.

At the same time simultaneous inclusions in (T1, M1) and (T3, M3) is lower than average.

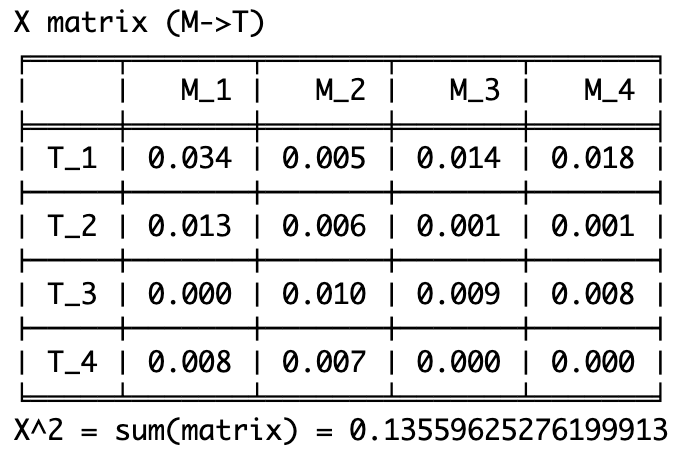
Being simultaneously in both T3 and M4 is impossible (0% probability).

Computing X2 matrix and value for (C->T) is done in section "Computing X2 values" in cell 13.



Item (C4 -> T3) with value 0.2 is the most deviated from the independence.

Computing X2 matrix and value for (M->T) is done in section "Computing X2 values" in cell 14.



Item (M1 -> T1) with value 0.034 is the most deviated from the independence, which is not that much, so the whole table shows more or less independent values.

Finally, we work with an independence criterion. Number of degrees of freedom for contingency table is (width - 1) \* (height - 1) = (4 - 1) \* (4 - 1) = 9. For this, at the 95% probability chi-squared value is less than 16.92. At the 99% probability chi-squared value is less than 21.67

**C -> T**

For X2 = 0.39464 and X2 \* N > 16.92 => N > 16.92 / 0.39464 = 42.87

That means that for any N >= 43 we can reject the hypothesis of statistical independence at 95% confidence level. For any N <= 42 the hypothesis should be accepted.

For X2 = 0.39464 and X2 \* N > 21.67 => N > 21.67 / 0.39464 = 54.91

That means that for any N >= 55 we can reject the hypothesis of statistical independence at 99% confidence level. For any N <= 54 the hypothesis should be accepted.

**M -> T**

For X2 = 0.135596 and X2 \* N > 16.92 => N > 16.92 / 0.135596 = 124.78

That means that for any N >= 125 we can reject the hypothesis of statistical independence at 95% confidence level. For any N <= 124 the hypothesis should be accepted.

For X2 = 0. 135596 and X2 \* N > 21.67 => N > 21.67 / 0. 135596 = 159.81

That means that for any N >= 160 we can reject the hypothesis of statistical independence at 99% confidence level. For any N <= 159 the hypothesis should be accepted.

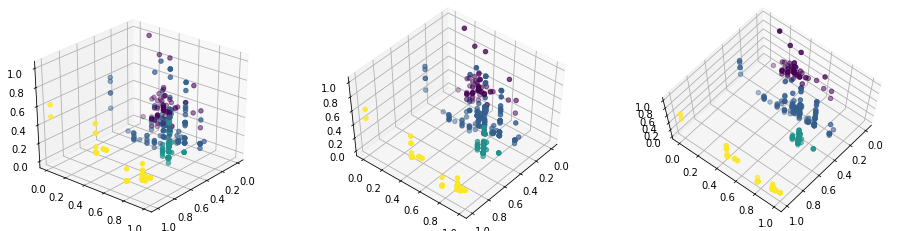
So, it can be seen that C partition (core speed) is much better associated with T partition than M partition (memory bandwidth). There are a lot more items in C partition which are more deviant from statistical independence than in M partition, and in average C partition is more dependent on T. According to independence criterion, M partition required more samples to be rejected as statistically independent, which makes it less believable to be associated with T.

5. PCA/SVD

All code for this task is in file: [svd\_pca.ipynb](https://github.com/PreFX48/modern_data_analysis/blob/master/svd_pca.ipynb)

For this task we select 3 features: the width of the screen resolution, the speed of the GPU memory and the maximal GPU power. Intuitively they should correlate rather well since in order render a higher resolution a GPU needs to have a faster GPU memory, and both rendering and memory management requires power. So maybe the representation of these features could be somehow embedded into a 2D space or approximated by a 1D ranking factor.

In section "Standardization and Visualization" in cells 3, 4 we normalize these features in visualize them as dots in 3D space. Colors here show different values of screen resolution to make the plot more graphical.



Then we compute their data scatter and impact of individual principal components in it (in section "Data Scatter and SVD" in cells 5, 6). The computation is done using numpy's linear algebra functions.

Not normalized:

data\_scatter = 11310651901

SVD: Z\_shape=(390, 390), mu\_shape=(3,), C\_shape=(3, 3)

Component 1 impact is 11242166483.89, or 99.39%

Component 2 impact is 63915164.01, or 0.57%

Component 3 impact is 4570253.10, or 0.04%

Normalized

data\_scatter = 330.2714119493378

SVD: Z\_shape=(390, 390), mu\_shape=(3,), C\_shape=(3, 3)

Component 1 impact is 296.43, or 89.75%

Component 2 impact is 23.50, or 7.11%

Component 3 impact is 10.35, or 3.13%

Then we compute the hidden ranking factor (in cells 7-9). The first Z-vector and loading vectors are negative, so we change their signs.

The loading vector that we get is [0.60923897 0.69332032 0.38488286]. After normalizing it we get the following formula for the ranking factor:

FACTOR = 0.36\*Resolution\_W + 0.41\*Memory\_Speed\_MHz + 0.23\*Max\_Power\_Watts

We can also get the vector with the values of the factor for our dataset which will be normalized in [0;1] (or [0%;100%]).

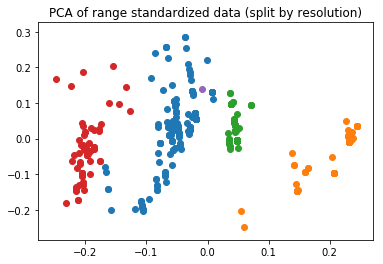
Here we can see that in this ranking the maximal power is a much less important feature than the resolution or memory speed, which is the most important.

It can be also seen that in this dataset there is no "ideal" sample in terms of our ranking that through combination if its individual features has the highest possible ranking score, and the highest ranked element has its FACTOR\_SCORE = 74%.

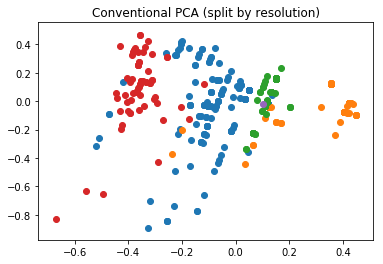
After that we use the first two principal components for visualization purposes. Here we compare visualization in features which are normalized using 1) z-scoring 2) range-standardization. For visibility purposes we color the points by their Resolution value which should split the samples in an important way. The code for visualization is in section "Visualization with first 2 PC" in cells 10-12.

Изображение выглядит как текст

Автоматически созданное описание



Besides that, we also applied conventional PCA to the normalized data and got results similar to z-scoring normalization (code is in cells 13-15):



This is because of the fact that these methods produce the same scoring vectors and loadings (except maybe for the sign of the vectors, which can be manually fixed), so it is easy to produce the same visualization.

Finally, comparing z-scoring and range-standardization, it can be seen from the plots that range-normalization preserves the structure of the dataset much better than Z-scoring and thus should be preferred. Mainly, the initial dataset has 4 distinct clusters of samples which show the supported resolution of the GPUs. The second plot (range-normalization) shows these clusters, while at the first plot (z-scoring) the distinction is almost lost.

Mainly it is because of the fact that z-scoring tries to negate the effect of the clusterization (which is mainly determined by "resolution" feature) by dividing it by standard deviation. For z-scoring it greatly affects the resolution feature and lessens its influence. Range-standardization, while still greatly affecting this feature, has a greater effect on other features, which keeps the resolution feature more distinct.

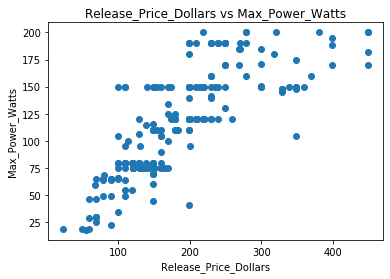
6. Correlation Coefficient

The code used to compute this task can be found in section "HW6" in this file: [hw\_2\_3\_6.ipynb](https://github.com/PreFX48/modern_data_analysis/blob/master/hw_2_3_6.ipynb)

It is seems correct that 'Release\_Price\_Dollars' will get more or less linear dependency with some other feature. Let's check that by drawing scatterplots (we do this by using matplotlib library in section "HW6", cell 14):

Изображение выглядит как текст

Автоматически созданное описание



Изображение выглядит как текст

Автоматически созданное описание

Scatter plot is the best way to visualize linear regression: we will see the distance between every value and predicted line. Even without line we can detect linear dependency just by eye: dots are grouping into something in the shape of line or stripe.

So, our second feature is 'Max\_Power\_Watts', let's use it as predictor. Let's just use linear regression (code in cell 15). We use regression provided by sklearn library:

Изображение выглядит как текст, карта

Автоматически созданное описание

Intercept: 24.063789728741483

Slope: 1.4308760915718808

Det. coefficient: 0.7207058952422762

Cor. coefficient: 0.6249264510422082

As we can see, it is predicted well. Slope in this case means tangent of the angle between x axis and predicted line. Also we found determinacy and correlation coefficient (Pearson) coefficients. Correlation coefficient is a measure of linear dependency between two chosen features. The value -1 means perfect negative relation, value 0 means no relation, value 1 means perfect linear dependency. Our value (almost 0.8) means quite strong correlation between such features.

Determinacy coefficient means scoring the extent of y-variance taken into account. It is high too in our case.

Let’s choose three dots from the set X with values:

**190, 105, 150**, and make a prediction. True values (y\_true) are:

**249, 349, 214**, predicted (y\_pred): **295.9, 174.3, 238.7**.

So, first and third predicted values are quite close to true values, the second one is predicted badly.

By applying regression code and computing MRAE on all set:

MRAE: 0.5554

This looks correct: dots on the scatterplot are dispersed around the predicted line. Comparing with determinacy from previous computation: it is smaller, but also has another meaning: the perfect value is 0. So, this pair of featured is far from perfect linear dependency, but has strong linear correlation.