Integrating statistical thinking to the reform of contract law: towards a leximetric approach of the French legal system

Louis Daniel Georges Brule Naudet

Integrating statistical thinking to the reform of contract law: towards a leximetric approach of the French legal system

Detailed research plan for a mathematical analysis of the understanding of French law

Louis Brulé Naudet
University of Paris-Dauphine (Paris Sciences et Lettres)

Contents

1 Fractality of the legal phenomenon and mathematization of practice: an infinitely fragmented and statistically appropriable science

To this day[1], the majority tendency is based on the confrontation between the human and social sciences and the fundamental disciplines applied to the observations of nature and the constraining forces which are, in a non-limitative way, for example, physics or geology. However, to think of the brutal separation between the subjects presents itself as a logic refuted for centuries, notably, in the legal doctrine[2], thanks to the works of Gottfried Wilhelm Leibniz[3], and the structuring of the theories of language and fundamental computer science seems to go in this direction as well, effectively demonstrating the nature of complex dynamics attached to literary paradigms. In this sense, one of the points of rapprochement between computer science and literature that we will address is the rejection of ambiguity in decision making and its impact on the modeling of behaviors induced by iterative protocols[4]: thus, the objective of the study would be to demonstrate a possible mathematical representation of the practice of law in order to draw statistical conclusions on the effectiveness of the norm and its understanding by the legal actors.

[1]L. Strauss. *Anthropologie structurale II*. Fonds Claude Lévi-Strauss. Place des éditeurs, 2014.

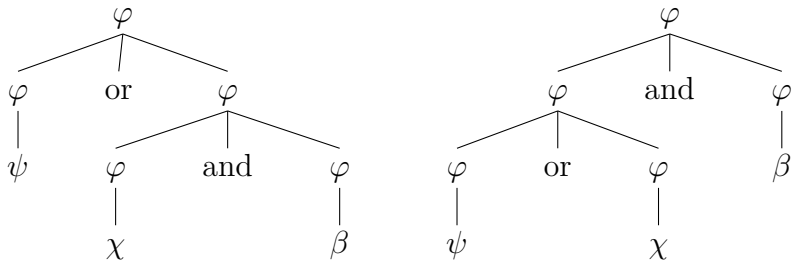
[2]Harvard University Press. *Chapter 1. Beyond Geometry: Leibniz and the Science of Law*, pages 17–27. Harvard University Press, 2009.

[3]M. H. Hoeflich. Law & geometry: Legal science from leibniz to langdell. *American Journal of Legal History*, 30(2):95–121, 1986.

[4]K. Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, 2013. ISBN 9781118762868.

1.1 Ambiguity, legal security and statistical modeling: a common paradigm for law and computer science

When we analyze the interpretation process resulting from the launching of an algorithm, whatever its language, we discover a particular characteristic that isolates computer grammar from other forms of communication practiced by human beings[5]: the rejection of algebraic ambiguity by the systematic application of precedence rules. This has two effects, the first is the neutralization of any form of uncontrolled decision, the second is the exact application of the developer's will. In natural grammar, if $\varphi \rightarrow \varphi$ *or* $\varphi \mid \varphi$ *and* $\varphi \mid \psi \mid \chi \mid \beta$, the sentence "$\psi$ *or* χ *and* β" induces a syntactic ambiguity from which two derivation trees result:



We can then understand the importance of the forced contextualization of the reasoning in the avoidance of interpretation errors by singularizing out the possible result, and from this simple demonstration can easily arise an analogy with the functioning of the syllogism applied in the practice of law[6], notably in the production of court decisions and the writing of conventions between people. From this conclusion, we can take the discussion

[5]H. M. Pandey. Advances in ambiguity detection methods for formal grammars. *Procedia Engineering*, 24:700–707, 2011.

[6]S. Brunet. La conception originelle de la sécurité juridique : l'allemagne. *Titre VII*, 5:79–90, 2020.

even further by exploring the fractal character of law in its construction as an iterated function, and this conception appears to be relatively effective in capturing the elements of hierarchical order, recursivity and self-reference in legal reasoning[7].

1.2 From the standardization of a large set of categorical variables to the realization of unbiased indicators of legal effectiveness

We have just demonstrated the importance of mathematics in the avoidance of errors of interpretation induced by the inambiguity caused with the extensive contextualization of the legal problem. Based on this fact, questioning the reform of contract law in terms of effectiveness would seem to be a major topic of research in doctrine, particularly thanks to the large production of case law and the possibility of a clear comparison between the periods before and after the adoption of ordinance No. 2016-131 of February 10, 2016 reforming contract law[8]. In this context, one particular point will attract our attention, concerning the quantitative approach to the practice of law in business and personal relationships. To speak of leximetry, in this sense, would consist of an application of econometric principles[9] to legal problems, in order to empirically test the different provisions and interpret their understanding by the actors of the juridical system. In this context, the researcher finds himself in a problematic of observation and compilation of

[7]Dalmasso & al. Comparer, mesurer, classer : L'art périlleux de la leximétrie du licenciement. *Travail et emploi*, 4:33–46, 2009.

[8]In accordance with Section 1 of Act No. 2018-287 of April 20, 2018, Ordinance No. 2016-131 of February 10, 2016, is ratified.

[9]J. M. Wooldridge. *Introductory Econometrics: A Modern Approach*. South-Western, 2009.

data directly from reality[10], which could be coupled with a survey of lawyers and jurists in contract law directly confronted with the dispositions of the 2016 reform. A general research problem could then be formulated: Is the reform of contract law a source of clarification or, on the contrary, of incomprehension for legal practitioners with regard to the judicial reading ? This would mean questioning the underlying issues, in particular whether the reform is understood by the drafters of contracts, by introducing a quantitative analysis of the number of appeals at first instance on issues arising from the reform, and whether it is correctly applied by the judges, by referring to the number of appeals lodged by the parties following referral to the court, as well as the number of appeals to the Court of Cassation and the number of rulings in favor of or against resolution of the litigation. Using the great intellectual and practical production, makes this research theme a propitious field of leximetric application, notably thanks to the important size of the observed sample and the possibility of clearly isolating the explained variable and the explanatory ones. We can also be satisfied that the result will be indicators of high quality, in particular thanks to their minimization of statistical bias[11], and, especially in the context of multiple linear regression[12], which will allow viable significant results from the estimation of the coefficients associated with the explanatory variables.

[10]D. M. Hausman. *The Philosophy of Economics: An Anthology*. Cambridge University Press, 3 edition, 2007.

[11]T. Alban. *Econométrie des Variables Qualitatives*. Dunod, 2000.

[12]H. Theil. *Principles of Econometrics*. Wiley Hamilton publication, 1979.

2 Towards a leximetry methodology for impact analysis of reforms: the example of ordinary least squares regression

One of the reasons for the development of specific statistical techniques is due to the nature of the data that are mobilized. These data are generally non-experimental and of several types, each one raising specific problems. In the context of an impact analysis, two models are possible: time series and panel data. These data are usually drawn from supposedly representative samples of the total population. This means in particular that one can only calculate an estimation of the parameters from this sample: even if it is randomly drawn, this sample is rarely perfect and there is always a risk that it is not exactly representative of the population it is supposed to represent[13], in particular, in our example, this could result from anomalies in the judicial procedure. The first property expected of an estimator is therefore that it is unbiased, i.e. that its expectation is equal to the *true* parameter. The interest of the OLS method is precisely to produce unbiased representations: the situations where the estimator overestimates and underestimates θ such that the estimated value will be correct on average.

2.1 Simple linear regression and OLS estimator, analysis of both strengths and limitations

The least squares method consists in minimizing the sum of the squares of the residuals, weighted differences in the multidimensional case, between each point of the regression cloud and its projection, parallel to the ordinate

[13]P. Kennedy. *A Guide to Econometrics, 5th Edition*, volume 1. The MIT Press, 5 edition, 2003.

axis, on the regression line, such as:

$$y = \beta_0 + \beta_1 x_1 \tag{1}$$

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x} \tag{3}$$

In this model, the β_1 coefficient will be interpreted as the marginal effect of an additional unit or variation of the explanatory variable on the explained one and for a leximetrical application[14], it could explain, as an example, the relationship between the changes in positive law and the number of appeals to the court of first instance for contract law issues[15]. From this point of view, if there is evidence of an increase in the number of referrals to the courts of first instance, it could be interpreted either as a better reading of contract law, allowing for an increase in legal production by private individuals, or, on the contrary, as a lack of readability and comprehension of the provisions, and, if this is the case, as a lesser quality of the work of the drafters, weighing down on legal security, and in particular on the vulnerable party. However, the major drawback of simple linear regression is the existence of omitted variable bias. Thus, in addition to simultaneity, correlation between the explanatory variables and the error term can occur when an omitted variable affects both the explained variable and one[16] explanatory variable. One method to

[14]The Python 3.8 source code is available in the appendix, tested with execution via Unix Shell (zsh), and aims at both a descriptive analysis of position variables and dispersion indicators, but also of simple and multiple linear regression coefficients, based on a gradient descent algorithm with empirical optimization of the learning coefficient.

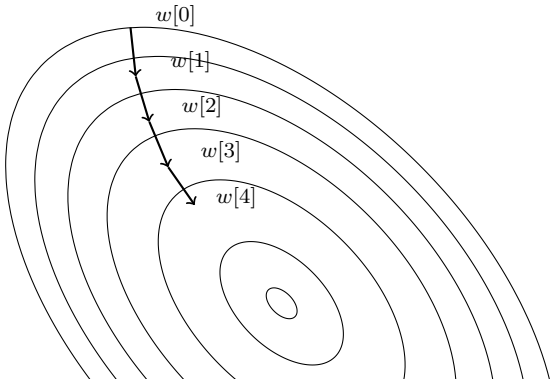
[15]If we want to analyze the impact of a purely qualitative explanatory variable, we will use a dummy variable, taking either the value 0 or 1, which will be integrated into the equation in the same way as a quantitive data.

[16](or more).

limit this bias would be to introduce several explanatory variables within a multiple linear regression.

2.2 Introduction to multiple regression by gradient descent algorithm

The gradient descent algorithm is designed for differentiable optimization: it is therefore intended to minimize a differentiable real function defined on a euclidean space[17]. By this process, it will be possible to optimize the value of the regression coefficients in order to minimize the mean square error and thereby approximate the real impact of the variation of the explanatory variables. The algorithm is iterative and proceeds by successive improvements, until convergence.



For our approach in Python, it is important to understand the mathematical decomposition of a gradient descent mechanism[18]. This model is

[17]R. Bonnin. *Machine Learning for Developers*. Packt Publishing, 2017.

[18]Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization. Springer US, 2013.

structurally based on an approximation of the value of x, the global minimum of the generalized representation of partial derivatives associated with the empirically observed function, computed by a recursive process blocked by a learning rate modulating the correction and by extension, the speed of convergence. Mathematically, the problem can be formulated as follows :

$$\eta^{t+1} = \eta^t - \alpha \nabla S^t \tag{4}$$

$$\nabla S^t = \frac{1}{n} \sum_{i=1}^{n} (h_\eta(x_i) - y) x_i \tag{5}$$

Therefore,

$$\eta^{t+1} = \eta^t - \frac{\alpha}{n} \sum_{i=1}^{n} (h_\eta(x_i) - y) x_i \tag{6}$$

Where ∇S^t represents the gradient associated with the function, α, the learning rate and η_t the estimator of the β_t coefficients. The issue of the determination of the learning rate makes an optimal analysis very complex: if the learning rate is too low, convergence can only be achieved with a very large number of iterations, requiring a lot of resources; if it is too high, the system will face a divergence and it will be impossible to approach the value of the gradient[19]. To overcome this risk, the attached algorithm, based on a matrix method of determination, has been designed for the optimization of this parameter by implementing a comparison process of the different linear coefficients of determination of Pearson from the regression[20]. So, in the context of our study, the R-squared being defined as the ratio of the variance explained by the regression to the total variance, and allowing to measure

[19]K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series. MIT Press, 2012.

[20]C. Carrère. *Statistiques descriptives: Théorie et applications*.

the proximity of the data to the fitted regression line; a value close to 1 will indicate that the model explains all the variability of the data around the mean, and conversely, a reduced coefficient will present the regression as not explaining the dependencies between the variables.

In order to establish a research schedule, it would seem wise to plan a report over several years in order to complete the dataset concerning the various judicial institutions, probably over three years for the purpose to be more or less in line with the first decade of the reform and to obtain a better statistical significance, but this field of research is promising, especially in France, where it is actually absent from legal doctrine and thought, in spite of its potential for the identification of the weaknesses of our law, especially in such a fundamental discipline as contract, which, by extension, touches on the right to property and the freedom of entrepreneurship.

References

Dalmasso & al. Comparer, mesurer, classer : L'art périlleux de la leximétrie du licenciement. *Travail et emploi*, 4:33–46, 2009.

T. Alban. *Econométrie des Variables Qualitatives*. Dunod, 2000.

R. Bonnin. *Machine Learning for Developers*. Packt Publishing, 2017.

S. Brunet. La conception originelle de la sécurité juridique : l'allemagne. *Titre VII*, 5:79–90, 2020.

C. Carrère. *Statistiques descriptives: Théorie et applications*.

K. Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, 2013. ISBN 9781118762868.

D. M. Hausman. *The Philosophy of Economics: An Anthology*. Cambridge University Press, 3 edition, 2007.

M. H. Hoeflich. Law & geometry: Legal science from leibniz to langdell. *American Journal of Legal History*, 30(2):95–121, 1986.

P. Kennedy. *A Guide to Econometrics, 5th Edition*, volume 1. The MIT Press, 5 edition, 2003.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series. MIT Press, 2012.

Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization. Springer US, 2013.

H. M. Pandey. Advances in ambiguity detection methods for formal grammars. *Procedia Engineering*, 24:700–707, 2011.

Harvard University Press. *Chapter 1. Beyond Geometry: Leibniz and the Science of Law*, pages 17–27. Harvard University Press, 2009.

L. Strauss. *Anthropologie structurale II*. Fonds Claude Lévi-Strauss. Place des éditeurs, 2014.

H. Theil. *Principles of Econometrics*. Wiley Hamilton publication, 1979.

J. M. Wooldridge. *Introductory Econometrics: A Modern Approach*. South-Western, 2009.

A Appendix: full source code in Python 3.10, or how to perform leximetric analysis from scratch

```
import numpy as np
import pandas as pd
from scipy.stats import t

def minimum(axis):
    """
    minimum : pandas.core.frame.DataFrame -> Float
    minimum(axis) Calculates the minimum of all the variations of each
        variable of interest. Return minimum_val.
    """
    try:
        list_values = axis.tolist() # Transform
            pandas.core.frame.DataFrame to List to iterate.
        minimum_val = float(min(list_values))

    return minimum_val

def maximum(axis):
    """
    max_calculus : pandas.core.frame.DataFrame -> Float
    max_calculus(axis) Calculates the maximum of all the variations of
        each variable of interest. Return maximum_val.
    """
    try:
        list_values = axis.tolist() # Transform
            pandas.core.frame.DataFrame to List to iterate.
        maximum_val = float(max(list_values))
```

```
    return maximum_val

def wide(axis):
    """
    wide : pandas.core.frame.DataFrame -> Float
    wide(axis) Calculates the delta of all the variations of each
        variable of interest. Return delta.
    """
    try:
        delta = maximum(axis) - minimum(axis)

    return delta

def average(axis):
    """
    average : pandas.core.frame.DataFrame -> Float
    average(axis) Calculates the mean of all the variations of each
        variable of interest. Return mean.
    """
    try:
        list_values = axis.tolist() # Transform
            pandas.core.frame.DataFrame to List to iterate.
        mean = sum(float(i) for i in list_values) / len(list_values)

    return mean

def median(axis):
    """
    median : pandas.core.frame.DataFrame -> Float
    median(axis) Calculates the median of all the variations of each
        variable of interest. Return median.
```

```
    """
    try:
        list_values = axis.tolist() # Transform
            pandas.core.frame.DataFrame to List to iterate.
        list_values_sorted = sorted(list_values)

        if len(list_values_sorted) % 2 == 0:
            median = (list_values_sorted[(int(len(list_values_sorted) -
                1)) // 2] +
                list_values_sorted[(int(len(list_values_sorted) + 1)) //
                2] ) / 2.0

        else:
            median = list_values_sorted[(len(list_values_sorted) - 1) //
                2]

    return median

def quartiles(axis):
    """
    quartiles : pandas.core.frame.DataFrame -> Float * Float
    quartiles(axis) Calculates the 1st and 3rd quartiles of all the
        variations of each variable of interest. Return quartile_1,
        quartile_3.
    """
    try:
        list_values = axis.tolist() # Transform
            pandas.core.frame.DataFrame to List to iterate.
        list_values_sorted = sorted(list_values)

        if len(list_values_sorted) % 4 == 0:
```

```
            quartile_1 = list_values_sorted[len(list_values_sorted) // 4
                - 1]
            quartile_3 = list_values_sorted[3 * (len(list_values_sorted)
                // 4 - 1)]

        else:
            quartile_1 = list_values_sorted[len(list_values_sorted) // 4]
            quartile_3 = list_values_sorted[3 * (len(list_values_sorted)
                // 4)]

    return quartile_1, quartile_3

def variance(axis):
    """
    variance : pandas.core.frame.DataFrame -> Float
    variance(axis) Calculates the corrected variance of all the
        variations of each variable of interest. var.
    """
    try:
        list_values = axis.tolist() # Transform
            pandas.core.frame.DataFrame to List to iterate.
        var = 0

        for i in range(0, len(list_values)):
            var = (var + (list_values[i] - average(axis)) ** 2)

        var = var / (len(list_values) - 1)

    return var

def std(axis):
```

```
    """
    std : pandas.core.frame.DataFrame -> Float
    std(axis) Calculates the standard deviation of all the variations of
        each variable of interest . Return standard_deviation.
    """
    try:
        standard_deviation = variance(axis) ** (1 / 2)

    return standard_deviation

def statistics (dataframe):
    """
     statistics : pandas.core.frame.DataFrame ->
        pandas.core.frame.DataFrame
     statistics (dataframe) Performs key statistic analysis for each
        explanatory variable, stores all data in a summary table.
        Return dataframe_stats.
    """
    dataframe_stats = pd.DataFrame(columns=["Explanatory variable",
        "Minimum", "Maximum", "Delta", "Mean", "Median", "1st
        quartile", "3rd quartile", "Adj. Variance", "Adj. Standard
        deviation"])

    for element in dataframe:
        min = minimum(dataframe[element])
        max = maximum(dataframe[element])
        delta = wide(dataframe[element])
        mean = average(dataframe[element])
        mdn = median(dataframe[element])
        quar_1, quar_3 = quartiles(dataframe[element])
        adj_var = variance(dataframe[element])
```

```
            adj_std = std(dataframe[element])

            row = {"Explanatory variable": element, "Minimum": min,
                "Maximum": max, "Delta": delta, "Mean": mean, "Median":
                mdn, "1st quartile": quar_1, "3rd quartile": quar_3, "Adj.
                Variance": adj_var, "Adj. Standard deviation": adj_std}
            dataframe_stats = dataframe_stats.append(row,
                ignore_index=True)

        return dataframe_stats

def covariance(y, x):
    """
    covariance: pandas.core.frame.DataFrame *
        pandas.core.frame.DataFrame -> Float
    covariance(y, x) Calculates the corrected covariance of the
        explanatory variables with respect to the explained variable.
        Return adj_cov.
    """
    try:
        numerator = 0

        for i in range(0, len(x), 1):
            numerator += (y[i] - average(y)) * (x[i] - average(x))

        adj_cov = numerator / (len(x) - 1)

    return adj_cov

def pearsonr(y, x):
    """
```

```
    pearsonr: pandas.core.frame.DataFrame *
        pandas.core.frame.DataFrame -> Float
    pearsonr(y, x) Calculates the Pearson's correlation coefficient of
        the explanatory variables with respect to the explained
        variable. Return corr.
    """
    try:
        corr = covariance(y, x) / (std(x) * std(y))

    return corr

def ols(y, x):
    """
    ols: pandas.core.frame.DataFrame * pandas.core.frame.DataFrame
        -> Float
    ols(y, x) Calculates the B1 and B0 coefficients of the Ordinary
        Least Mean Square Method, to perform linear regression. Return
        pearsonr_dict, pearsonr_dataframe.
    """
    try:
        numerator = 0
        denominator = 0

        for i in range(0, len(x)):
            numerator += (x[i] - average(x)) * (y[i] - average(y))
            denominator += (x[i] - average(x)) ** 2

        b1 = numerator / denominator
        b0 = average(y) - (b1 * average(x))

    return b0, b1
```

```
def r2_score(y, x):
    """
    r2_score: pandas.core.frame.DataFrame *
        pandas.core.frame.DataFrame -> Float
    r2_score(y, x) Calculates the coefficient of determination of the
        explanatory variables with respect to the explained variable.
        Return score.
    """
    try:
        b0, b1 = ols(y, x)
        sum_total_squares = 0
        sum_residuals_squares = 0

        for i in range(0, len(x)) :
            y_reg = b0 + b1 * x[i]
            sum_total_squares += (y[i] - average(y)) ** 2
            sum_residuals_squares += (y[i] - y_reg) ** 2

        score = 1 - (sum_residuals_squares / sum_total_squares)

    return score

def adj_r2_score(y, x):
    """
    adj_r2_score: pandas.core.frame.DataFrame *
        pandas.core.frame.DataFrame -> Dict * Float
    adj_r2_score(y, x) Calculates the adjusted coefficient of
        determination of the explanatory variables with respect to the
        explained variable. Return adj_score.
    """
```

```
    try:
        adj_score = 1 - (((1 - r2_score(y, x)) * (len(x) - 1)) / (len(x)
            - 1 - 1))

    return adj_score

def std_err(y, x):
    """
    std_err: pandas.core.frame.DataFrame *
        pandas.core.frame.DataFrame -> Float
    std_err(y, x) Calculates the root-mean-square deviation of the
        Ordinary Least Mean Square Method. Return rmse.
    """
    try:
        b0, b1 = ols(y, x)
        rmse = 0
        ss_xx = 0

        for i in range(len(x)):
            y_reg = b0 +(b1 * x[i])
            rmse += (y[i] - y_reg) ** 2
            ss_xx += (x[i] - average(x)) ** 2

        rmse = ((rmse / (len(x) - 2)) / ss_xx) ** (1 / 2) # Correction
            of the degree of liberties .

    return rmse

def t_stat(y, x):
    """
    t_stat: pandas.core.frame.DataFrame * pandas.core.frame.DataFrame
```

```
        -> Float
    t_stat(y, x) Calculates the test statistic of the Ordinary Least
        Mean Square Method. Return t.
    """
    try:
        b0, b1 = ols(y, x)
        t = b1 / std_err(y, x)

    return t

def p_value(y, x):
    """
    p_value: pandas.core.frame.DataFrame *
        pandas.core.frame.DataFrame -> Float
    p_value(y, x) Calculates the test statistic of the Ordinary Least
        Mean Square Method. Return p.
    """
    try:
        degrees_of_freedom = len(x) - 1
        p = (1.0 - t.cdf(abs(t_stat(y, x)), degrees_of_freedom)) * 2.0

    return p

def slr(dataframe, dep_dataframe):
    """
    slr : pandas.core.frame.DataFrame * pandas.core.frame.DataFrame
        -> pandas.core.frame.DataFrame
    slr(dataframe, dep_variable) Simple linear regression on each
        explanatory variable, using the ordinary least squares method.
        Return ols_dataframe.
    """
```

```
    try:
        ols_dataframe = pd.DataFrame(columns=["Explanatory variable",
            "Covariance", "Pearson's correlation", "R-squared", "Adj.
            R-squared", "B0 coef.", "B1 coef.", "Std err. (B1)", "t-stat.",
            "p-value"])
        k = len(dataframe.columns) # Number of explicative variables

        for element in dataframe:
            adj_cov = covariance(dep_dataframe, dataframe[element])
            corr = pearsonr(dep_dataframe, dataframe[element])
            b0, b1 = ols(dep_dataframe, dataframe[element])
            score = r2_score(dep_dataframe, dataframe[element])
            adj_score = adj_r2_score(dep_dataframe,
                dataframe[element])
            rmse = std_err(dep_dataframe, dataframe[element])
            t = t_stat(dep_dataframe, dataframe[element])
            p = p_value(dep_dataframe, dataframe[element])

            row = {"Explanatory variable": element, "Covariance":
                adj_cov, "Pearson's correlation": corr, "R-squared":
                score, "Adj. R-squared": adj_score, "B0 coef.": b0, "B1
                coef.": b1, "Std err. (B1)":rmse, "t-stat.": t,
                "p-value": p}
            ols_dataframe = ols_dataframe.append(row,
                ignore_index=True)

    return ols_dataframe

def matrix_rewrite(y, x):
    """
    matrix_rewrite : pandas.core.frame.DataFrame *
```

```
        pandas.core.frame.DataFrame -> List[List] * List[List]
    matrix_rewrite(y, x) matrixing of the pandas tables, to solve the
        multiple linear regression model. Shape of X : (n_samples,
        n_features). Return Y, X.
    """
    try:
        Y, X = [], []

        for element in y:
            Y.append(element)

        for i in range(0, len(x), 1):
            X.append(x.iloc[i].values.tolist())

    return Y, X

def mean_matrix(x):
    """
    mean_matrix : pandas.core.frame.DataFrame -> numpy.ndarray
    mean_matrix(x) Creates a matrix containing the means of the
        explanatory variables. Return mu.
    """
    mean_list = []

    for element in x:
        mean_list.append(average(x[element]))

    mu = np.array(mean_list)

    return mu
```

```
def std_matrix(x):
    """
    std_matrix : pandas.core.frame.DataFrame -> numpy.ndarray
    std_matrix(x) : Creates a matrix containing the standard deviation
        of the explanatory variables. Return sigma.
    """
    std_list = []

    for element in x:
        std_list.append(std(x[element]))

    sigma = np.array(std_list)

    return sigma

def feature_normalize(y, x):
    """
    feature_normalize : pandas.core.frame.DataFrame *
        pandas.core.frame.DataFrame -> numpy.ndarray *
        numpy.ndarray * numpy.ndarray
    feature_normalize(y, x) Normalize values to optimizing gradient
        descent. Return X_norm, Y, mu, sigma.
    """
    try:
        mu = mean_matrix(x)
        sigma = std_matrix(x)
        Y, X = matrix_rewrite(y, x)

        X_norm = (X - mu) / sigma
        X_norm = np.c_[np.ones(X_norm.shape[0]), X_norm] #
            Translates slice objects to concatenation along the second axis.
```

```
    return X_norm, Y, mu, sigma

def ulr(Y, X, theta):
    """
    ulr : numpy.ndarray * numpy.ndarray * numpy.ndarray ->
        numpy.ndarray
    ulr(X, Y, theta) Computes the dependent variable of a particular
        combinaison of theta for linear regression. Return iterate_value.
    """
    predictions = X.dot(theta)
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)

    iterate_value = 1 / (2 * len(Y)) * errors.T.dot(errors)

    return iterate_value

def accuracy(y, x, theta, mu, sigma):
    """
    accuracy: numpy.ndarray * numpy.ndarray * numpy.ndarray *
        numpy.ndarray * numpy.ndarray -> Float
    accuracy(y, x, theta, mu, sigma) Calculates R2 of the regression to
        optimizing learning rate. Return score.
    """
    try:
        sum_total_squares = 0
        sum_residuals_squares = 0

        for i in range(0, len(x)) :
            normalize_test_data = ((np.array(x.iloc[i]) - mu) / sigma)
```

```
            normalize_test_data = np.hstack((np.ones(1),
                normalize_test_data))
            prediction = normalize_test_data.dot(theta)

            sum_total_squares += (y[i] - average(y)) ** 2
            sum_residuals_squares += (y[i] - prediction) ** 2

        score = 1 - (sum_residuals_squares / sum_total_squares)

    return score

def gradient_descent(Y, X, theta, alpha, mu, sigma, iterations, y, x):
    """
    gradient_descent : numpy.ndarray * numpy.ndarray * numpy.ndarray
        * Float * Int -> numpy.ndarray * numpy.ndarray
    gradient_descent(X, Y, theta, alpha, iterations) Computes the
        dependent variables for linear regression. Return theta,
        iterate_values.
    """
    try:
        iterate_values = np.zeros(iterations)

        for i in range(iterations):
            predictions = X.dot(theta)
            errors = np.subtract(predictions, Y)
            sum_delta = (alpha / len(Y)) * X.transpose().dot(errors)
            theta -= sum_delta

            iterate_values[i] = ulr(Y, X, theta)
            progressbar(iterations, i + 1, alpha=alpha)
```

```
            if i == (iterations - 1):
                score = accuracy(y, x, theta, mu, sigma)
                progressbar(iterations, i + 1, alpha=alpha, score=score)

    return theta, iterate_values, score

def mls(y, x):
    """
    mls : pandas.core.frame.DataFrame * pandas.core.frame.DataFrame
        -> pandas.core.frame.DataFrame * numpy.ndarray *
        numpy.ndarray * numpy.ndarray
    mls(y, x) Multiple linear regression on several endogenous
        variables. Return normalized_coef_dataframe, theta, mu, sigma.
    """
    X, Y, mu, sigma = feature_normalize(y, x)

    theta = np.zeros(len(mu) + 1)
    iterations = 99999
    #alpha = [0.065, 0.50, 1.50, 2.00] # Test some learning rates to
        select the highest R-squared.
    alpha = [0.065]
    theta_list, score_list = [], []

    for element in alpha:
        print("\n testing ({}/{}):".format((alpha.index(element) + 1),
            len(alpha)))
        theta_train, iterate_values, score = gradient_descent(Y, X,
            theta, element, mu, sigma, iterations, y, x)
        theta_list.append(theta_train)
        score_list.append(score)
```

```
print("\n{}".format(theta))

normalized_coef_dataframe = pd.DataFrame(columns=["B0", "B1",
    "B2", "B3", "B4", "B5", "B6", "R-squared"])
row = {"B0":theta[0], "B1":theta[1], "B2":theta[2], "B3":theta[3],
    "B4":theta[4], "B5":theta[5], "B6":theta[6],
    "R-squared":max(score_list)}
normalized_coef_dataframe =
    normalized_coef_dataframe.append(row, ignore_index=True)

return normalized_coef_dataframe, theta, mu, sigma
```