|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ | | **«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»** | | **(МОСКОВСКИЙ ПОЛИТЕХ)** | | Факультет информационных технологий | |
|  |

Кафедра «Прикладная информатика»

Форма обучения: очная

|  |  |  |
| --- | --- | --- |
| **ОТЧЕТ**  **ПО ПРОЕКТНОЙ ДЕЯТЕЛЬНОСТИ** | | |
| Тема: *«Построение и оценивание параметров систем уравнений.»* | | |
|  | | |
| Группа | *211-365* |  |
| Студенты | *\_\_\_\_\_\_\_\_\_\_\_*  *\_\_\_\_\_\_\_\_\_\_\_*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | Тимофей Игоревич Кондрашин  Денис Сергеевич Тюлюкин  Тимур Рафаэлевич Халфеев  Егор Михайлович Баннов |
| Преподаватель,  к.п.н., доцент | *\_\_\_\_\_\_\_\_\_\_\_* | Наталья Ивановна Царькова |
| Оценка работы  Дата | \_\_\_\_\_\_\_\_\_\_\_ | «\_\_» мая 2023 |
|  |  |  |
|  |  |  |

МОСКВА 2023

Оглавление

[ВВЕДЕНИЕ 5](#_Toc134381186)

[ГЛАВА 1. Теоретическое обоснование модели 8](#_Toc134381187)

[1.1. Общие положения 8](#_Toc134381188)

[1.2. Excel 8](#_Toc134381189)

[1.2.1. Избавление от мультиколлинеарности и выбросов 8](#_Toc134381190)

[1.2.2. Проверка идентификации 9](#_Toc134381191)

[1.2.3. Оценка параметров структурной модели 10](#_Toc134381192)

[1.2.4. Прогнозирование 11](#_Toc134381193)

[1.3. Python 12](#_Toc134381194)

[1.3.1. Использованные библиотеки 12](#_Toc134381195)

[1.3.2. Проверка на наличие выбросов 12](#_Toc134381196)

[1.3.3. Интерполяция данных 13](#_Toc134381197)

[1.3.4. Решение проблемы мультиколлинеарности 14](#_Toc134381198)

[1.3.5. Разделение данных на обучающую и тестовую выборки 14](#_Toc134381199)

[1.3.6. Модель линейной регрессии 16](#_Toc134381200)

[1.3.7. Прогнозирование 17](#_Toc134381201)

[ГЛАВА 2. Построение и анализ эконометрической модели 20](#_Toc134381202)

[2.1. Python 20](#_Toc134381203)

[2.1.1. Предварительный анализ и обработка данных 20](#_Toc134381204)

[2.1.1.1. Изучение данных на наличие пропусков, нулевых значений и ошибок измерений 20](#_Toc134381205)

[2.1.1.2. Проверка данных на наличие выбросов и изучение распределения данных 22](#_Toc134381206)

[2.1.1.3. Интерполяция данных 26](#_Toc134381207)

[2.1.1.3.1. Данные до интерполяции 27](#_Toc134381208)

[2.1.1.3.2. Данные после интерполяции 27](#_Toc134381209)

[2.1.1.4. Избавление от мультиколлинеарности между признаками 29](#_Toc134381210)

[2.1.1.4.1. Центрирование данных 29](#_Toc134381211)

[2.1.1.4.2. Минимаксная нормализация данных 29](#_Toc134381212)

[2.1.1.4.3. Нормализация средним (Z-нормализация) 29](#_Toc134381213)

[2.1.1.4.4. MaxAbsScaler 30](#_Toc134381214)

[2.1.1.4.5. RobustScaler 30](#_Toc134381215)

[2.1.1.5. Добавление шума в данные 31](#_Toc134381216)

[2.1.1.6. Построение матрицы коэффициентов межфакторной корреляции 33](#_Toc134381217)

[2.1.2. Разделение данных на обучающую и тестовую выборки. 34](#_Toc134381218)

[2.1.3. Построение модели линейной регрессии 35](#_Toc134381219)

[2.1.4. Отбор переменных для модели линейной регрессии 36](#_Toc134381220)

[2.1.4.1. Разделение данных на зависимые и независимые переменные 36](#_Toc134381221)

[2.1.4.2. Оценка предварительно составленных систем уравнений 37](#_Toc134381222)

[2.1.4.2.1. Система независимых уравнений 38](#_Toc134381223)

[2.1.4.2.2. Система рекурсивных уравнений 43](#_Toc134381224)

[2.1.4.2.3. Система одновременных уравнений 48](#_Toc134381225)

[2.1.5. Прогнозирование 54](#_Toc134381226)

[2.1.5.1. Система независимых уравнений 55](#_Toc134381227)

[2.1.5.2. Система рекурсивных уравнений 56](#_Toc134381228)

[2.1.5.3. Система одновременных уравнений 56](#_Toc134381229)

[2.1.6. Прогнозирование с помощью методов машинного обучения 57](#_Toc134381230)

[2.2. Excel 59](#_Toc134381231)

[2.2.1. Исходные данные 59](#_Toc134381232)

[2.2.2. Системы уравнений 60](#_Toc134381233)

[ЗАКЛЮЧЕНИЕ 64](#_Toc134381234)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 65](#_Toc134381235)

[ПРИЛОЖЕНИЕ А 66](#_Toc134381236)

[ПРИЛОЖЕНИЕ Б 69](#_Toc134381237)

[ПРИЛОЖЕНИЕ В 72](#_Toc134381238)

[ПРИЛОЖЕНИЕ Г 74](#_Toc134381239)

[ПРИЛОЖЕНИЕ Д 76](#_Toc134381240)

[ПРИЛОЖЕНИЕ Е 78](#_Toc134381241)

[ПРИЛОЖЕНИЕ Ж 80](#_Toc134381242)

[ПРИЛОЖЕНИЕ З 82](#_Toc134381243)

[ПРИЛОЖЕНИЕ И 85](#_Toc134381244)

**ВВЕДЕНИЕ**

Современный экономический статус регионов Российской Федерации требует использования разнообразных инструментов для оценки экономического развития, условий жизни, экономического равновесия. Одним из основных показателей, определяющих потенциал развития отраслей и регионов в целом, является валовый региональный продукт (ВРП), характеризирующий потенциал экономики регионов. Экономика Москвы – крупнейшая экономика среди субъектов Российской Федерации по объёму валового регионального продукта (ВРП). Москва – крупнейший в общегосударственном масштабе финансовый центр и центр управления значительной частью экономики страны. Так, например, в Москве сосредоточено более половины банков, зарегистрированных в стране, при этом на их долю приходится 90% банковских активов. Кроме того, большая часть крупнейших компаний зарегистрированы и имеют центральные офисы именно в Москве. Валовой региональный продукт – обобщающий показатель экономической деятельности региона, характеризующий процесс производства товаров и услуг для конечного использования.

ВРП позволяет решать следующий комплекс задач:

* Определение общих размеров экономики региона, уровня его экономического развития;
* Выявления общих тенденций экономического развития;
* Обоснование приоритетных направлений развития экономики;
* Изучение механизмов и факторов экономического роста;
* Оценка эффективности государственного управления в регионе.

Валовой региональный продукт помогает определить общие размеры экономики региона и оценить эффективность государственного управления в нем. Важно оценивать не только уровень валового регионального продукта, но и его динамику. Если ВРП растет, значит, экономика работает хорошо и город развивается: появляются новые производства, рабочие места, растет уровень доходов населения. Востребованность использования ВРП как на федеральном, так и на региональном уровне сделало эту тему актуальной для рассмотрения.

Целью исследования является построение и оценивание параметров систем уравнений для прогнозирования и анализа зависимости валового регионального продукта (ВРП) (млн руб.), объёма инвестиций в основной капитал за год (млн руб.), дохода населения города Москвы (млн руб.) от следующих факторов:

* Финансовый результат деятельности (чистая прибыль(сальдо)), млн руб.;
* Прямые иностранные инвестиции, млн USD;
* Среднегодовая численность занятых, тыс. чел.;
* Стоимость основных фондов, млн руб.;
* Степень износа основных фондов, %;
* Затраты на научные исследования и разработки, млн руб.;
* Объем инновационных товаров, работ и услуг, млн руб.;
* Объем экспорта, млн USD;
* Объем импорта, млн USD;
* Сумма остатков вкладов на счетах в Банке России, млн руб.;
* Прожиточный минимум субъекта Российской Федерации (город Москва), тыс. руб..

Достижение указанной цели определило постановку и решение следующих задач:

* Найти исходные данные для анализа.
* Построить эконометрические модели (систему с раздельными уравнениями, рекурсивными уравнениями, одновременными уравнениями).
* Проверить системы на необходимое условие идентификации и достаточное условие идентификации.
* Выбрать метод оценивания систем: МНК, КМНК, ДМНК.
* Исключить незначимые факторы.
* Рассчитать параметры регрессии.
* Оценить качество спецификации моделей.
* Спрогнозировать данные на 2023 год.

Объектом исследования являются вышеупомянутые факторы роста на валовой региональный продукт, объём инвестиций и доход населения города Москвы. Предметом исследования являются системы эконометрических уравнений (системы с раздельными уравнениями, рекурсивными уравнениями, одновременными уравнениями).

Структурными элементами отчета являются: титульный лист, оглавление, введение, глава 1 (теоретическое обоснование модели), глава 2 (построение и анализ эконометрической модели), заключение, список используемых источников и литературы и приложения.

# ГЛАВА 1. Теоретическое обоснование модели

## 1.1. Общие положения

Прогнозирование различных экономических факторов используется специалистами в целях разработки оптимальных алгоритмов для развития различных сфер экономики.

В свою очередь Москва является крупнейшим финансовым центром страны, по этой причине прогнозирование и анализ зависимости валового регионального продукта, объёма инвестиций в основной капитал за год и дохода населения города Москвы очень важно.

Прогнозирование в нашем проекте проводится на основе следующих экономических моделей: система с раздельными переменными, рекурсивными уравнениями и одновременными уравнениями.

Рассмотрим, что представляет собой каждая из систем уравнений:

* В системе независимых уравнений, каждая зависимая переменная y рассматривается как функция одного и того же набора факторов x.
* В системе с рекурсивными, уравнениями зависимая переменная y одного уравнения выступает в виде фактора x в другом уравнении.
* В системе одновременных уравнений, одни и те же зависимые переменные в одних уравнениях входят в левую часть, а в других уравнениях – в правую часть системы

Перед началом анализа и прогнозирования проводился поиск исходных данных, которые были получены путем совмещения разрозненных данных из различных источников в единую структуру.

## 1.2. Excel

### 1.2.1. Избавление от мультиколлинеарности и выбросов

Мультиколлинеарность — это явление, при котором два или более предикторов взаимосвязаны между собой. Это может привести к нестабильности оценок и усложнить интерпретацию результатов.

Выбросы представляют собой аномальные значения в наборе данных.

Для избавления от мультиколлинеарности и выбросов, в каждой системе строится матрица межфакторной корреляции при помощи инструмента “Анализа данных” и проверяется наличие или отсутствие сильной связи между разными факторами. В случае, если такая связь имеется, то сравнивается значение статистической значимости фактора в модели и выбрасывается переменная, имеющая меньшее значение.

Когда в системе не наблюдается межфакторная корреляция, проводится поиск и удаление выбросов. В ходе этого процесса может измениться P-значение (вероятность, позволяющая определить значимость коэффициента регрессии) и соответственно, если его значение после удаления выбросов становиться больше 0,05, то мы можем сделать вывод, что соответствующая независимая переменная практически не влияет на зависимую переменную и её стоит исключить из модели.

### 1.2.2. Проверка идентификации

При переходе системы одновременных уравнений от приведенной формы модели к структурной, возникает проблема идентификации, которая характеризует единственность соответствия между приведенной и структурной формами модели.

Соответственно возникает необходимость проверить данную систему на идентифицируемость.

Возможны следующие ситуации:

1. Структурные коэффициенты однозначно выражаются через коэффициенты приведенной формы модели. В этом случае структурную модель называют точно идентифицируемой.
2. Некоторые из структурных коэффициентов не выражаются через коэффициенты приведенной формы модели. Такую структурную модель называют неидентифицируемой.
3. Структурные коэффициенты неоднозначно выражаются через коэффициенты приведенной формы модели. Тогда структурную модель называют сверхидентифицируемой.

Для определения типа структурной модели обозначаем число эндогенных (зависимые переменные, число которых равно числу уравнений в системе) переменных в i-м уравнении системы через H, а число экзогенных (предопределенные переменные, влияющие на эндогенные переменные, но не зависящие от них) переменных, которые содержаться в системе, но не входят в данное уравнение, - через D, тогда условие идентифицируемости модели можно записать в виде следующего четного правила:

* если D + 1 = H, то уравнение идентифицируемо;
* если D + 1 < H, то уравнение неидентифицируемо;
* если D + 1 > H, то уравнение сверхидентифицируемо.

При этом структурная модель всегда представляет собой систему совместных уравнений, каждое из которых требуется проверять на идентификацию. Модель считается идентифицируемой, если каждое уравнение системы идентифицируемо. Если хотя бы одно из уравнений системы неидентифицируемо, то и вся модель считается неидентифицируемой. Сверхидентифицируемая модель содержит хотя бы одно сверхидентифицируемое уравнение.

### 1.2.3. Оценка параметров структурной модели

Оценка параметров для независимых и рекурсивных уравнений проводится через обычный метод наименьших квадратов (МНК), который позволяет получить такие оценки параметров, при которых сумма квадратов отклонений фактических значений результативного признака от теоретических минимальна.

В случае системы одновременных уравнений в зависимости от типа структурной модели, могут быть применены разные способы.

Так, если структурная модель является точно идентифицируемой, то применяется косвенный метод наименьших квадратов (КМНК). Для этого структурная модель преобразовывается в приведенную форму модели и для каждого уравнения обычным МНК оцениваются приведенные коэффициенты. После этого коэффициенты приведенной формы модели трансформируются в параметры структурной модели.

В случае, если система сверхидентифицируема, то КМНК не используется, так как не дает однозначных оценок для параметров структурной модели. Вместо этого чаще всего применяется двухшаговый метод наименьших квадратов (ДМНК). Его основная идея заключается в том, что на основе приведенной формы модели, необходимо получить теоретические значения эндогенных переменных, которые содержатся в правой части уравнения.

После этого, подставив их вместо фактических значений, применяем обычный МНК к структурной форме сверхидентифицируемого уравнения. При этом, если все уравнения системы сверхидентифицируемые, то для оценки структурных коэффициентов каждого уравнения используется ДМНК. Если же в системе есть точно идентифицируемые уравнения, то структурные коэффициенты по ним находятся из системы приведенных уравнений.

### 1.2.4. Прогнозирование

После оценивания параметров структурной модели уравнений каждой их систем, мы получаем уравнения регрессии, которые позволяют нам прогнозировать значения валового регионального продукта, объёма инвестиций в основной капитал за год и дохода населения города Москвы.

Таким образом необходимо найти расчетные значения факторов, от которых зависят прогнозируемые значения. Для этого мы можем воспользоваться временными рядами.

После нахождения расчетных значений факторов подставляем их в уравнения регрессии и получаем прогнозные значения.

## 1.3. Python

### 1.3.1. Использованные библиотеки

В ходе выполнения проекта для визуализации данных использовалась библиотека Plotly Express. Она предоставляет более 30 функций для создания различных типов фигур. При этом программный интерфейс для использования данных функций максимально последователен и прост в освоении, за счет чего позволяет легко создавать точечные и столбчатые диаграммы, гистограммы, а также диаграммы солнечных лучей.

Для работы с данными применялась библиотека Pandas. Она помогает подготовить и провести первичный анализ данных, удалять и заполнять пропуски, отфильтровывать, отсортировывать или каким-либо образом изменять данные. Кроме этого, библиотека также поддерживает основные статистические методы, которые необходимы для работы с данными. Например, расчёт средних значений, их распределение по квантилям и другие.

### 1.3.2. Проверка на наличие выбросов

Нахождение выбросов — это один из методов анализа данных, который используется для выявления аномальных значений в наборе данных. Выбросы могут возникать из-за ошибок в данных, случайных вариаций или из-за наличия редких, но реальных экстремальных значений.

Существует несколько методов нахождения выбросов, в том числе метод Z-оценки и метод межквартильного размаха (IQR):

* Метод межквартильного размаха (IQR) основан на вычислении интерквартильного размаха - разницы между 75-перцентилью и 25-перцентилем. Затем определяется верхняя и нижняя границы выбросов, которые определяются как 1,5 межквартильных размаха за пределами верхнего и нижнего квартилей.
* Метод Z-оценки основан на расчете Z-оценки для каждого значения в наборе данных и проверке, насколько далеко каждое значение от среднего значения. Значения, которые находятся на расстоянии больше, чем три стандартных отклонения от среднего, могут быть считаны выбросами.

Кроме этого, еще одним мощными инструментом для проверки данных на наличие выбросов и аномалий являются коробчатые диаграммы и диаграммы скрипок.

На языке Python мы можем построить данные диаграммы при помощи функций combine\_box\_plots и combine\_violin\_plots. Соответственно на рисунке 1 мы можем увидеть слева – коробчатую диаграмму, а справа – диаграмму скрипок.

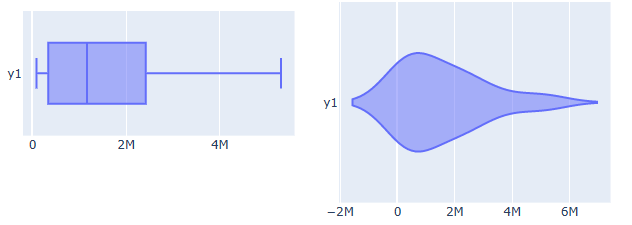


Рисунок 1 - Коробчатая диаграмма и диаграмма скрипок

В случае, если в данных присутствуют аномальные значения (выбросы), то их следует исключить.

### 1.3.3. Интерполяция данных

При применении методов машинного обучения к данным, существует проблема переобучения. Это явление возникает из-за малого количества наблюдений и приводит к тому, что модель хорошо объясняет примеры из обучающей выборки, но относительно плохо работает на примерах, которые не участвовали в обучении.

По этой причине зачастую для увеличения количества наблюдений используется метод интерполяции данных, который позволяет найти неизвестные промежуточные значения, по имеющемуся дискретному набору её известных значений.

Одним из наиболее эффективных методов интерполяции данных в Python, является метод под названием Akima.

### 1.3.4. Решение проблемы мультиколлинеарности

Чтобы избежать проблемы мультиколлинеарности можно использовать нормализацию и центрирование данных.

Существуют следующие методы нормализации данных:

* Центрирование данных: данный способ приводит все значения признаков к среднему значению. Это осуществляется путем вычитания среднего значения из каждого признака.
* Минимаксная нормализация данных: этот метод приводит все значения признаков к диапазону от 0 до 1. Это делается путем вычитания минимального значения и деления на разницу между максимальным и минимальным значениями.
* Нормализация средним (Z-нормализация): данный метод приводит все значения признаков к диапазону от -1 до 1, путем вычитания среднего значения и деления на стандартное отклонение.
* Max-нормализация данных: данный метод приводит все значения признаков к диапазону от -1 до 1. Это осуществляется путем деления каждого значения на максимальное абсолютное значение.
* Robust-нормализация данных: данный метод приводит все значения признаков к диапазону от -1 до 1. Это делается путем вычитания медианы и деления на интерквартильный размах.

### 1.3.5. Разделение данных на обучающую и тестовую выборки

Для обучения модели необходимо разделить данные на обучающую и тестовую выборки.

Обучающая выборка – это часть данных, на которых модель обучается, чтобы изучить потенциальные закономерности и отношения.

Тестовая выборка – это часть данных, которая используется для тестирования полученной модели и не используется при обучении. При этом результат мы сравниваем с фактическими значениями и исходя из этого, оцениваем точность модели.

При этом разделение данных на выборки может выполняться разными методами. По этой причине рассмотрим некоторые из них.

Валидация на отложенных данных (Hold-Out Validation) – данный способ валидации заключается в том, что данные разделяются на две выборки: обучающую и тестовую. Так модель обучается на обучающей выборке данных, а затем проверяется на тестовой выборке. При этом обычно обучающая выборка составляет 70-80% от исходных данных, а тестовая – оставшиеся 20-30%.

Кросс-валидация (Cross-Validation) – способ валидации, при котором данные разбиваются на несколько равны частей, которые называются “складками”. После этого каждая “складка” используется в качестве тестовой выборки, а оставшиеся – в качестве обучающей выборки. Таким образом, модель обучается и тестируется на каждой из “складок”, что позволяет получить оценку ее производительности на всех данных. Стоит отметить, что этот метод обеспечивает более стабильные оценки производительности, чем способ валидации на отложенных данных, однако он может быть затратным с точки зрения вычислительных ресурсов.

Поэлементная кросс-валидация (Leave-One-Out Cross Validation) – этот способ является частным случаем кросс-валидации, при котором количество складок равно количеству наблюдений в данных. Таким образом, каждое наблюдение используется в качестве тестовой выборки, а оставшиеся наблюдения – в качестве обучающей выборки. Этот метод обеспечивает наиболее точную оценку производительности модели, но может быть очень затратным с точки зрения вычислительных ресурсов.

Перекрестная проверка (Leave-p-Out Cross Validation) – данный способ почти полностью повторяет метод поэлементной кросс-валидации и отличается лишь тем, что изначально задается размер тестовой выборки, а оставшиеся наблюдения используются в качестве обучающей выборки.

### 1.3.6. Модель линейной регрессии

Линейная регрессия — это метод анализа данных, который используется для описания и прогнозирования линейной связи между зависимой и одной или несколькими независимыми переменными. Он часто используется в статистическом анализе данных для построения математической модели, которая может быть использована для прогнозирования значений зависимой переменной на основе значений независимых переменных.

Для построения модели линейной регрессии необходимо иметь набор данных, который состоит из пар значений зависимой и независимой переменных. Затем на основе этих данных проводится анализ, который позволяет оценить параметры модели - коэффициенты наклона и пересечения. Эти параметры могут быть использованы для построения уравнения линейной регрессии.

Для оценки качества модели линейной регрессии используются различные статистические показатели, такие как коэффициент детерминации (R-квадрат) или корреляционный коэффициент. Они позволяют оценить, насколько хорошо модель соответствует данным и насколько точно она может использоваться для прогнозирования значений зависимой переменной.

Важно понимать, что линейная регрессия является моделью, которая описывает только линейные отношения между переменными. Если связь между переменными не является линейной, то модель линейной регрессии может оказаться неэффективной и требовать использования других методов анализа данных.

Отбор переменных для модели линейной регрессии — это процесс выбора наиболее важных и значимых независимых переменных, которые будут использоваться в модели для прогнозирования зависимой переменной. Это важный шаг в построении модели, так как использование неподходящих переменных может привести к низкой точности прогнозов и низкому качеству модели.

Существует несколько методов отбора переменных для модели линейной регрессии, включая:

* Метод пошагового отбора переменных — это метод, который включает или исключает независимые переменные в модель пошагово на основе их значимости. На каждом шаге происходит выбор переменной, которая наиболее улучшает качество модели.
* Метод отбора на основе значимости - этот метод основан на анализе значимости каждой независимой переменной с помощью статистических тестов, таких как t-тест или F-тест. Переменные с низким уровнем значимости исключаются из модели.
* Метод регуляризации — это метод, который добавляет штрафы за большие значения коэффициентов в модель. Он позволяет автоматически отбирать переменные, которые наиболее важны для модели, и уменьшает вероятность переобучения.
* Метод отбора на основе информационных критериев - этот метод основан на использовании различных информационных критериев, таких как AIC (критерий Акаике) и BIC (критерий Шварца), чтобы выбрать модель с наименьшей ошибкой.

При этом выбор метода отбора переменных зависит от конкретной задачи и объема доступных данных.

### 1.3.7. Прогнозирование

Для прогнозирования нужно построить системы уравнений, которые будут состоять из уравнений, полученных путем отбора параметров. После этого необходимо провести прогнозирование и сравнить прогнозные значения с их реальными значениями. При этом прогнозирование проводиться для каждого уравнения по отдельности, а также для всех систем уравнений в целом.

Кроме этого, прогнозирование можно произвести и при помощи методов машинного обучения. Рассмотрим данные методы:

1. Логическая регрессия – это метод анализа данных, который использует математику для поиска взаимосвязей для поиска взаимосвязей между двумя факторами данных. Затем эта взаимосвязь используется для прогнозирования значения одного из этих факторов на основе другого.
2. Опорные вектора – основная идея этого метода заключается в переводе исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости, разделяющей классы. При этом разделяющей гиперплоскостью будет гиперплоскость, создающая наибольшее расстояние до двух параллельных гиперплоскостей. Алгоритм основан на допущении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.
3. k-ближайшие соседи – данный метрический алгоритм заключается в том, что объект присваивается тому классу, который является наиболее распространённым среди соседей данного элемента, классы которых уже известны. В случае использования метода для регрессии, объекту присваивается среднее значение по ближайшим к нему объектам, значения которых уже известны.
4. Случайный лес – этот метод состоит из нескольких отдельных деревьев решений, которые опираются на случайные особенности и обучение данным, чтобы достичь разумного предположения, которое имеет больше доверия, чем одно дерево решений.
5. Градиентный бустинг – данный метод пошагово создает каждое дерево регрессии, используя стандартную функцию потерь для измерения ошибок в каждом шаге и их исправления в следующем.
6. Нейронная сеть – это метод, который использует взаимосвязанные узлы или нейроны в слоистой структуре. Он создает адаптивную систему, с помощью которой компьютеры учатся на своих ошибках и постоянно совершенствуются.

**ГЛАВА 2. Построение и анализ эконометрической модели**

* 1. **Python**
     1. **Предварительный анализ и обработка данных**

Прежде чем приступать к построению и оценке параметров систем независимых, регрессионных и одновременных уравнений, необходимо провести предварительный анализ данных, визуализировать их, выявить и устранить выбросы, пропуски и аномалии.

* + - 1. **Изучение данных на наличие пропусков, нулевых значений и ошибок измерений**

Для начала посмотрим имеются ли в данных пропуски и нулевые значения. Для этого воспользуемся методом isna() и sum():

pd.concat([data.isna().sum(), data.isnull().sum()], axis=1, keys=['isna', 'isnull'])

Пропусков и нулевых значений в данных нет, так как нами заранее была проведена очистка и обработка данных на которую мы потратили достаточно много времени.

Подробно на этом этапе мы останавливаться не будем, так как это не является целью данной работы. Если обобщить, то при обработке данных были удалены ненужные признаки, а также были заполнены пропуски, удалены выбросы и аномалии, исправлены ошибки измерения.

Далее опишем функцию, которая поможет нам визуализировать все имеющиеся данные, а так же является мощным инструментом для поиска ошибок измерения (по графикам рассеяния можно увидеть наличие выбросов и аномалий).

Функция “combine\_scatter\_plots” строит графики рассеяния для каждой пары признаков в наборе данных. Это позволяет нам визуально оценить наличие ошибок измерений. Если ошибок нет, то точки на графике должны располагаться вблизи линии тренда. Если же точки расположены вдали от линии тренда, то это может говорить о наличии ошибок измерений. В таком случае необходимо исключить из набора данных некорректные значения. Код python с использованием данной функции представлен ниже:

def combine\_scatter\_plots(height=1000, width=1100, isyear=True):

    fig = make\_subplots(rows=5, cols=3)

    data\_copy = data.copy()

    if isyear:

        data\_copy.drop('Год', axis=1, inplace=True)

    for i, column in enumerate(data\_copy.columns.values):

        if 0 <= i < 3: row, col = 1, i + 1

        elif 3 <= i < 6: row, col = 2, i + 1 - 3

        elif 6 <= i < 9: row, col = 3, i + 1 - 6

        elif 9 <= i < 12: row, col = 4, i + 1 - 9

        else: row, col = 5, i + 1 - 12

        if 0 <= i < 3:

            fig.add\_trace(go.Scatter(y=data\_copy[f'y{i + 1}'], x=year, name=column), row=row, col=col)

        else:

            fig.add\_trace(go.Scatter(y=data\_copy[f'x{i - 2}'], x=year, name=column), row=row, col=col)fig.update\_layout(height=height, width=width, title\_text="Зависимость признаков от времени")

    fig.show()

После вызова функции получены следующие графики (рисунок 2):

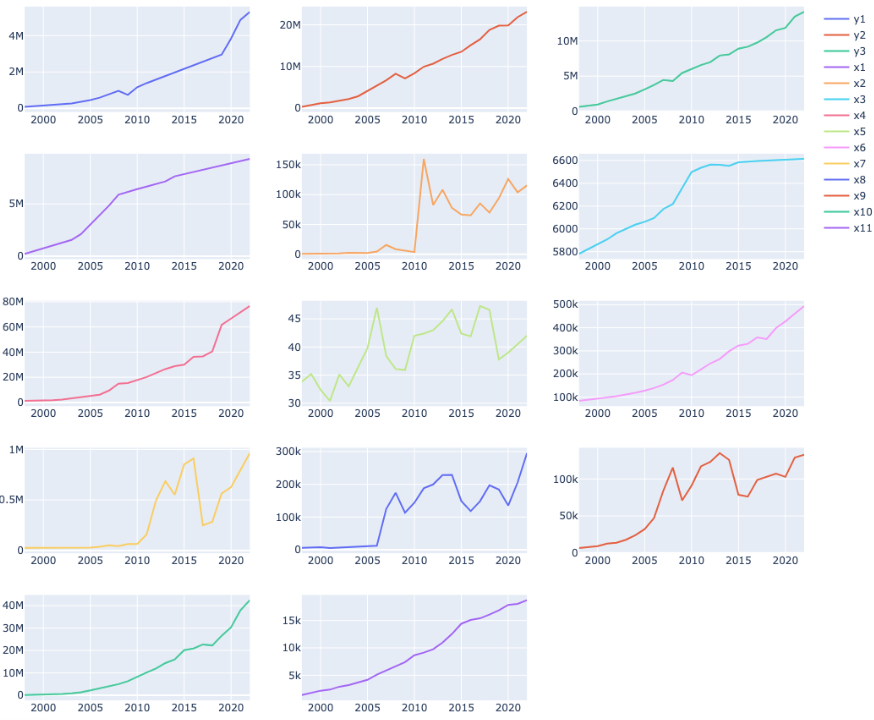


Рисунок 2 - Графики зависимости признаков от времени

На графиках рассеяния видно, что в большинстве случаев точки расположены вблизи линии тренда (к примеру *y1, y2*), что говорит об отсутствии ошибок измерений, но есть несколько признаков с достаточно сильным отклонением (к примеру *x2, x5*), несмотря на это мы не будем их удалять, так как в этих случаях данные и правда являются корректными.

* + - 1. **Проверка данных на наличие выбросов и изучение распределения данных**

На данном этапе будет искать выбросы с помощью следующих двух методов: метод межквартильного размаха и метод Z-оценки.

Метод межквартильного размаха (IQR) основан на вычислении интерквартильного размаха с помощью следующей формулы:

Реализация данного метода на python представлена ниже:

# Находим Первый квартиль (Q1) и Третий квартиль (Q3) и рассчитываем межквартильный размах

Q1 = data.quantile(q=.25)

Q3 = data.quantile(q=.75)

IQR = data.apply(stats.iqr)

# Оставляем только те значения, которые больше нижней границы и меньше верхней границы.

data\_clean = data[~((data < (Q1 - 1.5 \* IQR)) | (data > (Q3 + 1.5 \* IQR))).any(axis=1)]

data\_clean.shape

(25, 15)

Метод Z-оценки основан на расчете Z-оценки для каждого значения в наборе данных с помощью следующей формулы:

Реализация данного метода на python представлена ниже:

z = np.abs(stats.zscore(data))

data\_clean = data[(z < 3).all(axis=1)]

data\_clean.shape

(25, 15)

В результате проверки на наличие выбросов мы получили исходный набор данных. Это говорит нам о том что выбросов в данных найденно не было.

Еще одним мощными инстурментом для проверки данных на наличие выбросов и аномалий являются ящиковые диаграммы и диаграммы скрипок. Построим их для каждого признака в наборе данных. Для этого реализуем функции “combine\_box\_plots” и “combine\_violin\_plots”.

Функция “combine\_box\_plots” строит ящиковые диаграммы для каждого признака в наборе данных, где значения признака разбиваются на квартили и представляются в виде ящика. Ящик показывает границы первого и третьего квартилей, а также медиану. Черта внутри ящика показывает медиану, а концы усов - границы первого и третьего квартилей. Точки, выходящие за границы усов, считаются выбросами. Также на графике отображаются все значения признака.

Код python с использованием данной функции:

# Построение графиков

def combine\_box\_plots(height=1000, width=1100):

    fig = make\_subplots(rows=5, cols=3)

    data\_copy = data.copy()

    data\_copy.drop('Год', axis=1, inplace=True)

    for i, column in enumerate(data\_copy.columns.values):

        if 0 <= i < 3:row, col = 1, i + 1

        elif 3 <= i < 6:row, col = 2, i + 1 - 3

        elif 6 <= i < 9:row, col = 3, i + 1 - 6

        elif 9 <= i < 12:row, col = 4, i + 1 - 9

        else:row, col = 5, i + 1 - 12

        if 0 <= i < 3:

            fig.add\_trace(go.Box(x=data\_copy[f'y{i + 1}'], name=column), row=row, col=col)

        else:

            fig.add\_trace(go.Box(x=data\_copy[f'x{i - 2}'], name=column), row=row, col=col)fig.update\_layout(height=height, width=width, title\_text="Ящиковые диаграммы распределения признаков",showlegend=False)

    fig.show()

combine\_box\_plots()

Результат выполнения данного кода и построенные ящиковые диаграммы распределения признаков представлены на рисунке 3.

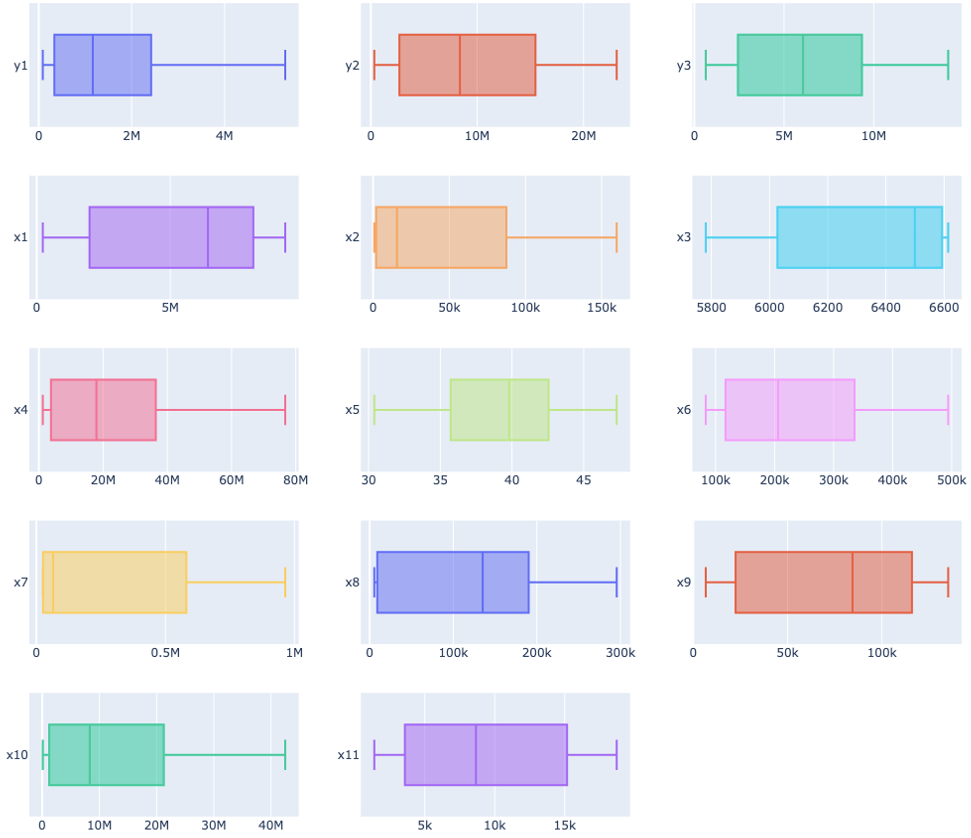


Рисунок 3 - Ящиковые диаграммы распределения признаков

Функция “combine\_violin\_plots” строит аналогичные диаграммы, но вместо ящика используется график, который представляет распределение значений признака с помощью ядерной оценки плотности:

def combine\_violin\_plots(height=1500, width=1100):

    fig = make\_subplots(rows=5, cols=3)

    data\_copy = data.copy()

    data\_copy.drop('Год', axis=1, inplace=True)

    for i, column in enumerate(data\_copy.columns.values):

        if 0 <= i < 3:row, col = 1, i + 1

        elif 3 <= i < 6:row, col = 2, i + 1 - 3

        elif 6 <= i < 9:row, col = 3, i + 1 - 6

        elif 9 <= i < 12:row, col = 4, i + 1 - 9

        else:row, col = 5, i + 1 - 12

        if 0 <= i < 3:

            fig.add\_trace(go.Violin(x=data\_copy[f'y{i + 1}'], name=column), row=row, col=col)

        else:

           fig.add\_trace(go.Violin(x=data\_copy[f'x{i - 2}'], name=column), row=row, col=col)fig.update\_layout(height=height, width=width, title\_text="Распределение признаков", showlegend=False)

    fig.show()

Результат выполнения данного кода и построенные графики распределения признаков представлены на рисунке 4.

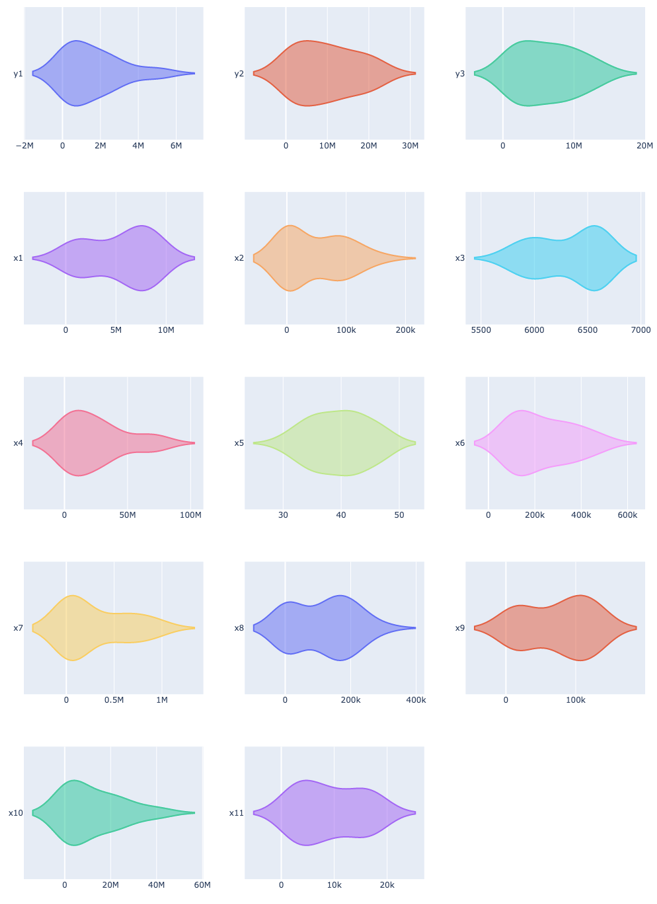


Рисунок 4 - Графики распределения признаков

Как мы можем видеть, в данных нет выбросов и аномалий. Также мы можем заметить, что распределение значений признаков не является нормальным. Имеются как унимодально распределенные признаки, так и многомодально распределенные признаки.

* + - 1. **Интерполяция данных**

В предыдущем разделе мы убедились, что в наших данных нет пропусков и нулевых значений. Однако, мы имеем всего лишь 25 наблюдений, что является недостаточным количеством для построения моделей, так как это приводит к большим ошибкам и переобучению модели. Поэтому нам необходимо увеличить количество наблюдений. Для этого мы будем использовать метод интерполяции данных.

Для этого создадим класс “InterpolateData”, который будет принимать на вход данные и метод интерполяции. По умолчанию метод интерполяции равен 'linear', а частота - 'Q' (квартальная). Для интерполяции данных мы будем использовать метод interpolate() из библиотеки pandas:  
class InterpolateData:

def \_\_init\_\_(self, data, method='linear', freq='Q'):

        self.data = data.copy()

        self.year = pd.to\_datetime(self.data['Год'].astype(str), format='%Y')

        self.data\_quarterly = data.copy()

        self.data\_quarterly['Год'] = pd.to\_datetime(self.data\_quarterly['Год'].astype(str), format='%Y')

        self.data\_quarterly.set\_index('Год', inplace=True)

        self.data\_quarterly = self.data\_quarterly.resample(freq).mean().interpolate(method=method)

    def head(self): return self.data\_quarterly.head()

    def scatter(self):

        fig = px.scatter(self.data\_quarterly)

        fig.update\_layout(title\_text="Данные после интерполяции", showlegend=True)

        return fig.show()

    def original\_scatter(self):

        fig = px.scatter(self.data.drop(columns=['Год']), x=self.year, y=self.data.drop(columns=['Год']).columns)

        fig.update\_layout(title\_text="Данные до интерполяции", showlegend=True)

        return fig.show()

    def after\_interpolate(self): return self.data\_quarterly

В нашей работе будет использоваться метод интерполяции под наименованием “akima”. Данный метод является одним из наиболее точных методов интерполяции. Для наглядности построим графики рассеяния до и после интерполяции.

* + - * 1. **Данные до интерполяции**

На рисунке 5 представлены данные до интерполяции.

InterpolateData(data).original\_scatter()

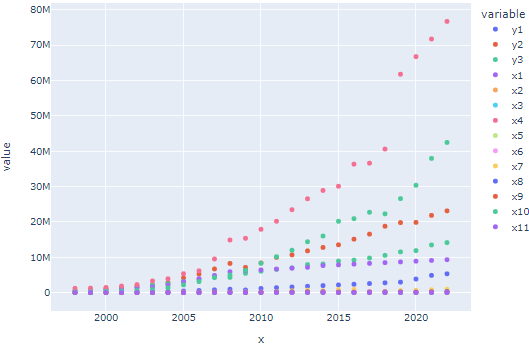


Рисунок 5 - Данные до интерполяции

* + - * 1. **Данные после интерполяции**

С помощью метода “akima” интерполируем данные. Результат данных после интерполяции представлен на рисунке 6.

# InterpolateData(data).scatter()

# InterpolateData(data, method='cubic').scatter()

# InterpolateData(data, method='quadratic').scatter()

InterpolateData(data, method='akima').scatter()

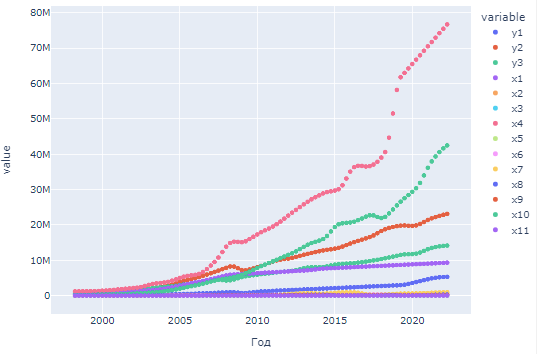


Рисунок 6 - Данные после интерполяции

После интерполяции данных мы получили ∼100 наблюдений (было 25), что является достаточным количеством для построения моделей. Снова посмотрим на то как распределены признаки (Рисунок 7). После этого перейдем к избавлению от мультиколлинеарности.



Рисунок 7- Зависимость признаков от времени после интерполяции

* + - 1. **Избавление от мультиколлинеарности между признаками**

В нашей работе мы будем использовать следующие методы нормализации данных:

* Центрирование данных;
* Минимаксная нормализация данных;
* Z-нормализация данных;
* Max-нормализация данных;
* Robust-нормализация данных.
  + - * 1. **Центрирование данных**

Этот способ приводит все значения признаков к среднему значению 0. Это делается путем вычитания среднего значения из каждого значения признака:

# data = data.apply(lambda x: x - x.mean())

* + - * 1. **Минимаксная нормализация данных**

Этот метод приводит все значения признаков к диапазону от 0 до 1. Это делается путем вычитания минимального значения и деления на разницу между максимальным и минимальным значениями:

data = data.apply(lambda x: (x - x.min()) / (x.max() - x.min()))

Именно этот метод нормализации данных мы будем использовать в нашей работе, как основным.

* + - * 1. **Нормализация средним (Z-нормализация)**

Этот метод приводит все значения признаков к диапазону от -1 до 1. Это делается путем вычитания среднего значения и деления на стандартное отклонение:

# data = data.apply(lambda x: (x - x.mean()) / x.std())

* + - * 1. **MaxAbsScaler**

Этот метод приводит все значения признаков к диапазону от -1 до 1. Это делается путем деления на максимальное абсолютное значение:

# transformer = MaxAbsScaler().fit(data)

# data = pd.DataFrame(transformer.transform(data))

* + - * 1. **RobustScaler**

Этот метод приводит все значения признаков к диапазону от -1 до 1. Это делается путем вычитания медианы и деления на интерквартильный размах:

# transformer = RobustScaler().fit(data)

# data = pd.DataFrame(transformer.transform(data))

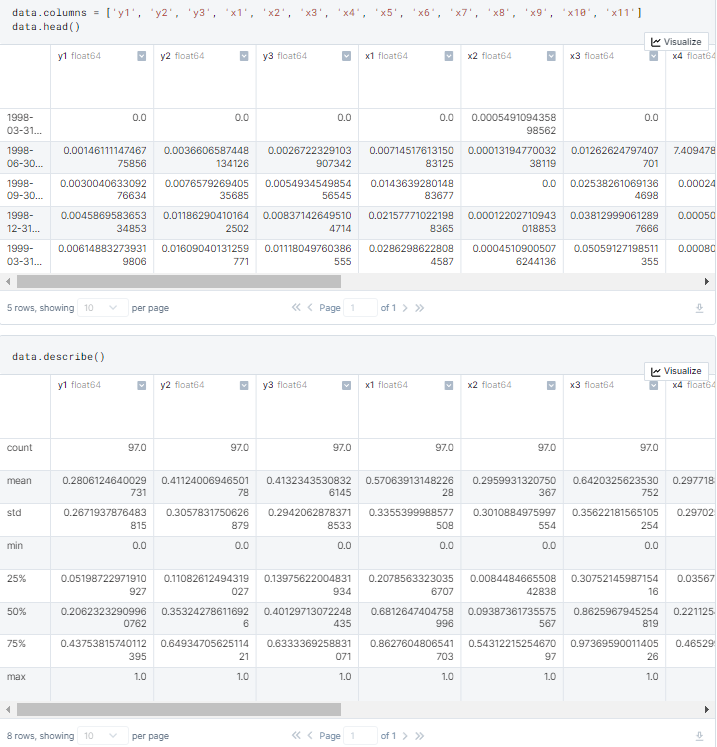


Рисунок 8 - Избавление от мультиколлинеарности

Как мы видим из части таблицы (Рисунок 8), все признаки находятся в диапазоне от 0 до 1. Так же мы видим, что среднее значение признаков близко к 0.5, а стандартное отклонение близко к 0.3. Это говорит о том, что данные нормализованы правильно. Это позволило нам избавиться от мультиколлинеарности между признаками.

* + - 1. **Добавление шума в данные**

Для объяснения добавления шума в данные, рассмотрим следующий график. На нем изображены зависимости признаков от времени (Рисунок 9).

# px.scatter(data, title="Зависимость признаков от времени", height=500)

px.scatter(data['y1'], title="Зависимость признака Y1 от времени до добавления шума", height=500)

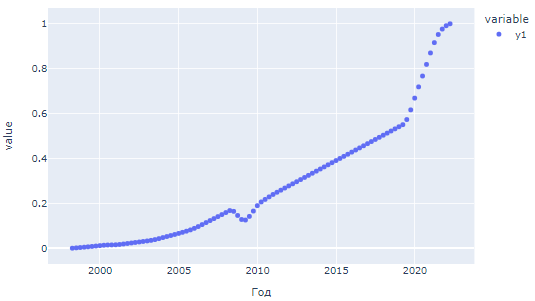


Рисунок 9 - Зависимость признака Y1 от времени до добавления шума

Как видно из графика, данные имеют очень сильную линейную структуру, из-за этого мы получаем сверх высокие метрики при обучении модели. Добавление шума в данные, позволяет сделать данные более реалистичными, так как в реальных данных всегда присутствует шум. Так же добавление шума в данные позволяет сделать данные более устойчивыми к выбросам. В следующем коде, а так же в результате его выполнения (Рисунок 10) представлено добавление шума:

np.random.seed(101)

noise = np.random.normal(0, .06, data.shape)

noisy\_data = data + noise

data = noisy\_data

data = data.apply(lambda x: (x - x.min()) / (x.max() - x.min()))

px.scatter(data['y1'], title="Зависимость признака Y1 от времени после добавления шума", height=500)

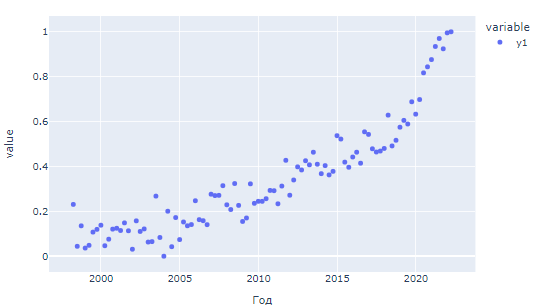


Рисунок 10 - Зависимость признака Y1 от времени до добавления шума

Далее добавим шум для всех данных (Рисунок 11):

combine\_scatter\_plots(isyear=False)

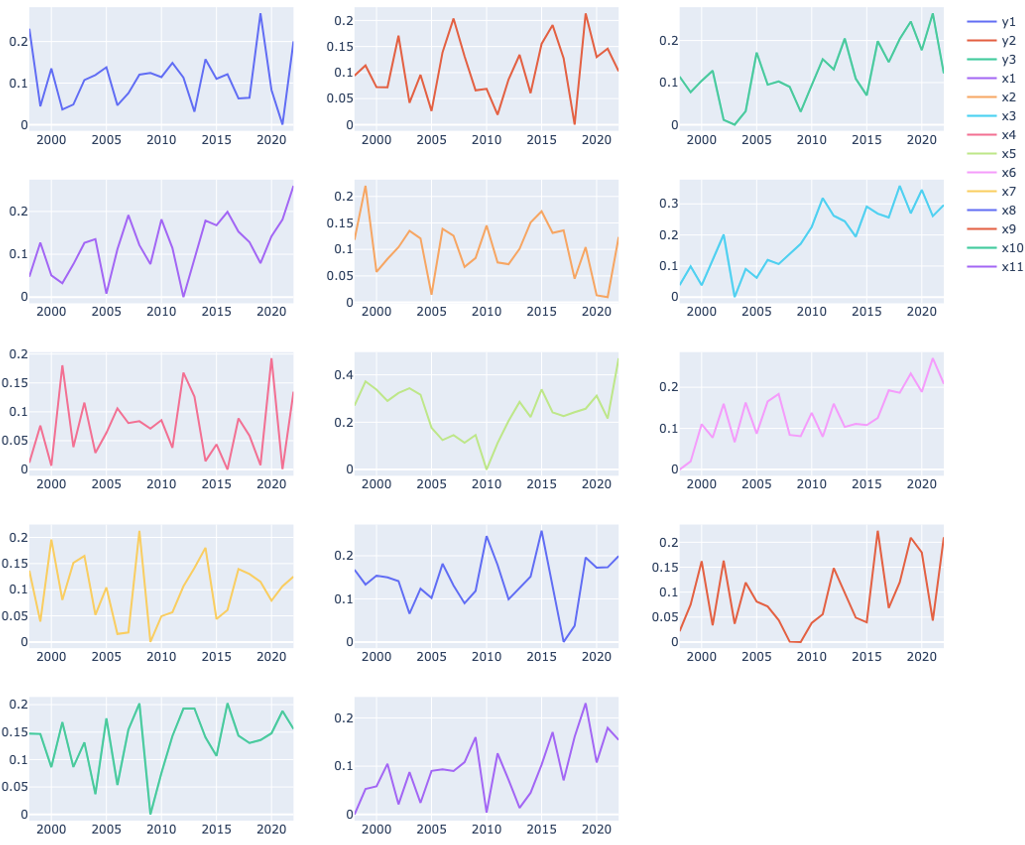


Рисунок 11 - Зависимость признакав от времени после добавления шума

Теперь наши данные выглядят более реалистично. Так же мы видим, что данные стали более устойчивыми к выбросам.

* + - 1. **Построение матрицы коэффициентов межфакторной корреляции**

Далее построим матрицу коэффициентов межфакторной корреляции (Рисунок 12):

fig = px.imshow(data.corr(), text\_auto='.2f', color\_continuous\_scale="gnbu",

                labels={'color': 'R', 'x': 'Фактор', 'y': 'Фактор'})

fig.update\_layout(height=1000, title\_text="Матрица коэффициентов межфакторной корреляции")

fig.show()

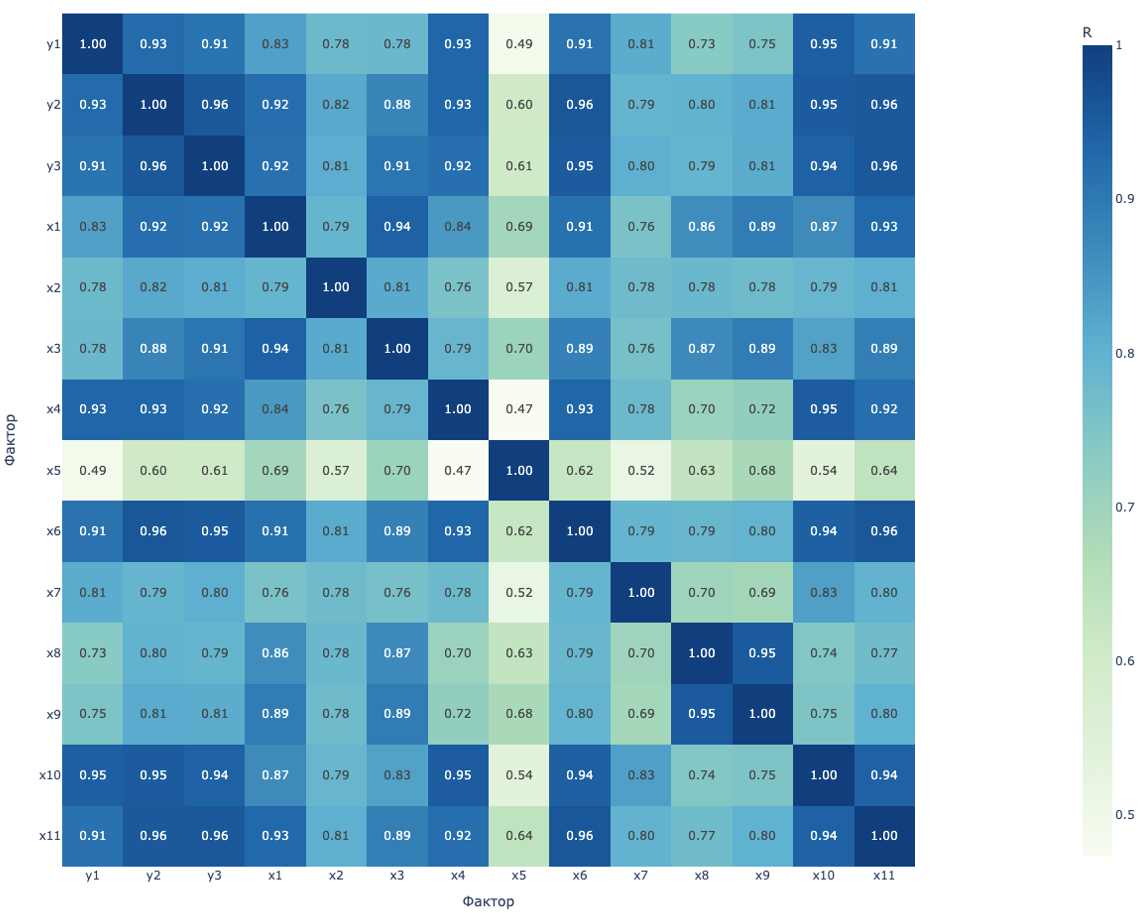


Рисунок 12 - Матрица коэффициентов межфакторной корреляции

Можно заметить, что между некоторыми признаками есть сильная линейная зависимость. Но при этом большая часть признаков слабо коррелирует между собой.

* + 1. **Разделение данных на обучающую и тестовую выборки.**

Для дальнейшего обучения модели, данные разделяются на обучающую и тестовую выборки. Обучающая выборка - это часть данных, на которых модель будет обучаться.

Валидация на отложенных данных (Hold-Out Validation) (Рисунок 13):

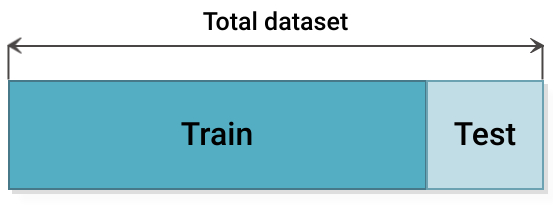


Рисунок 13 - Hold-Out Validation

Кросс-валидация (Cross-validation) (Рисунок 14):

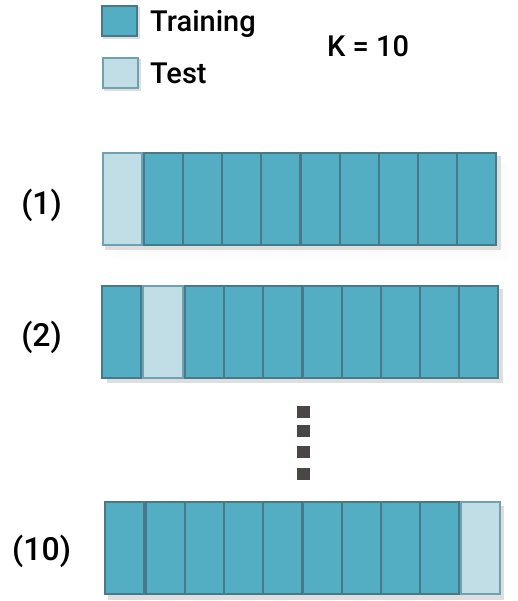


Рисунок 14 - Cross-validation

Leave-One-Out Cross-validation (LOO/LOOCV) (Рисунок 15):



Рисунок 15 - LOO/LOOCV

Leave-p-out cross-validation (LpOC) полностью повторяет метод Leave-One-Out cross-validation (LOO/LOOCV) и отличается лишь тем что изначально задается размер тестовой выборки в то время как в Leave-One-Out cross-validation он всегда равен 1.

# lpo = LeavePOut(p=5)

# for train\_index, test\_index in lpo.split(X):

#     print("TRAIN:", train\_index, "TEST:", test\_index)

#     X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]

#     y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

В итоге получаем следующие размеры выборок:

X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape

((67, 11), (30, 11), (67, 3), (30, 3))

* + 1. **Построение модели линейной регрессии**

Линейная регрессия - это метод, который используется для определения связи между зависимой и одной или несколькими независимыми переменными. Он широко применяется в статистическом анализе данных для создания математической модели, которая может быть использована для прогнозирования значений зависимой переменной на основе значений независимых переменных. Для построения этой модели необходимо иметь данные, состоящие из пар значений зависимой и независимой переменных. Затем проводится анализ, который позволяет определить параметры модели, такие как коэффициенты наклона и пересечения. С помощью этих параметров может быть построено уравнение линейной регрессии. Код построения модели линейной регрессии представлен в Приложении А

* + 1. **Отбор переменных для модели линейной регрессии**

Выбор метода отбора переменных зависит от конкретной задачи и объема доступных данных. В нашей работе мы будем использовать метод пошагового отбора переменных и метод отбора на основе значимости.

Для наглядности снова посмотрим на часть данных, которые мы будем использовать для построения модели линейной регрессии (Рисунок 16).



Рисунок 16 – Данные

Как было ранее сказано, мы имеем дело с набором данных, содержащим 11 независимых переменных и 3 зависимых переменных. Переменные x1-x11 являются независимыми переменными, а переменные y1-y3 являются зависимыми переменными. Разделим данные на независимые и зависимые переменные.

* + - 1. **Разделение данных на зависимые и независимые переменные**

Зависимые переменные - это переменные, значения которых мы хотим предсказать. В нашем случае это переменные y1, y2 и y3

Независимые переменные - это переменные, значения которых мы используем для прогнозирования зависимых переменных. В нашем случае это переменные [x1,…,x11]

* + - 1. **Оценка предварительно составленных систем уравнений**

Всего у нас три системы уравнений (Рисунок 17), которые мы можем использовать для прогнозирования зависимых переменных:

* Система независимых уравнений;
* Система рекурсивных уравнений;
* Система одновременных уравнений.

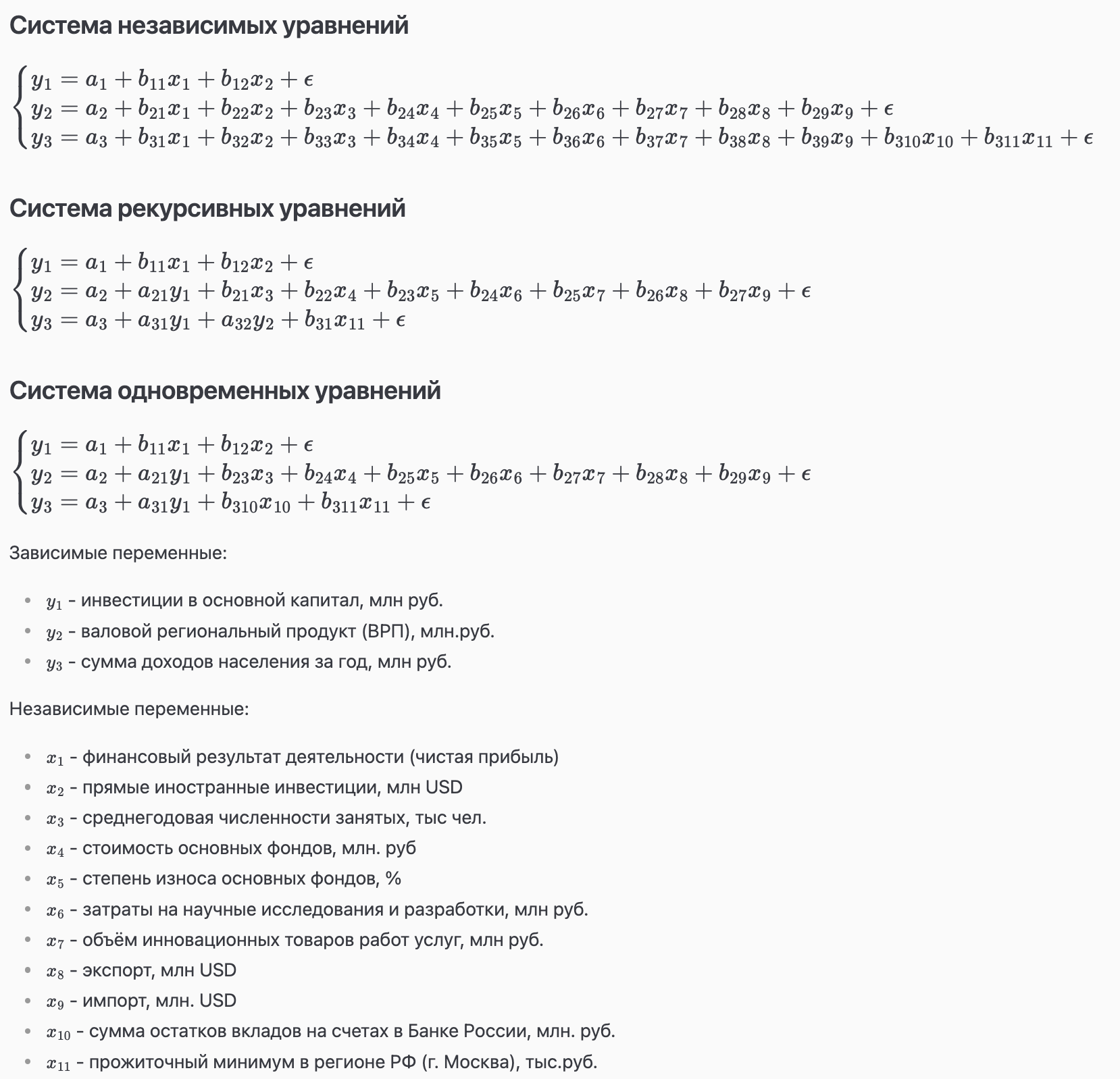


Рисунок 17 - Системы уравнений

* + - * 1. **Система независимых уравнений**

Рассмотрим нашу первую систему независимых уравнений. Обратимся к первому уравнению системы:

Исходная система уравнений представлена на рисунке 18:

fn1\_nez = pd.concat([x1\_train, x2\_train], axis=1)

CustomLinearRegression(y1\_train, fn1\_nez).summary()

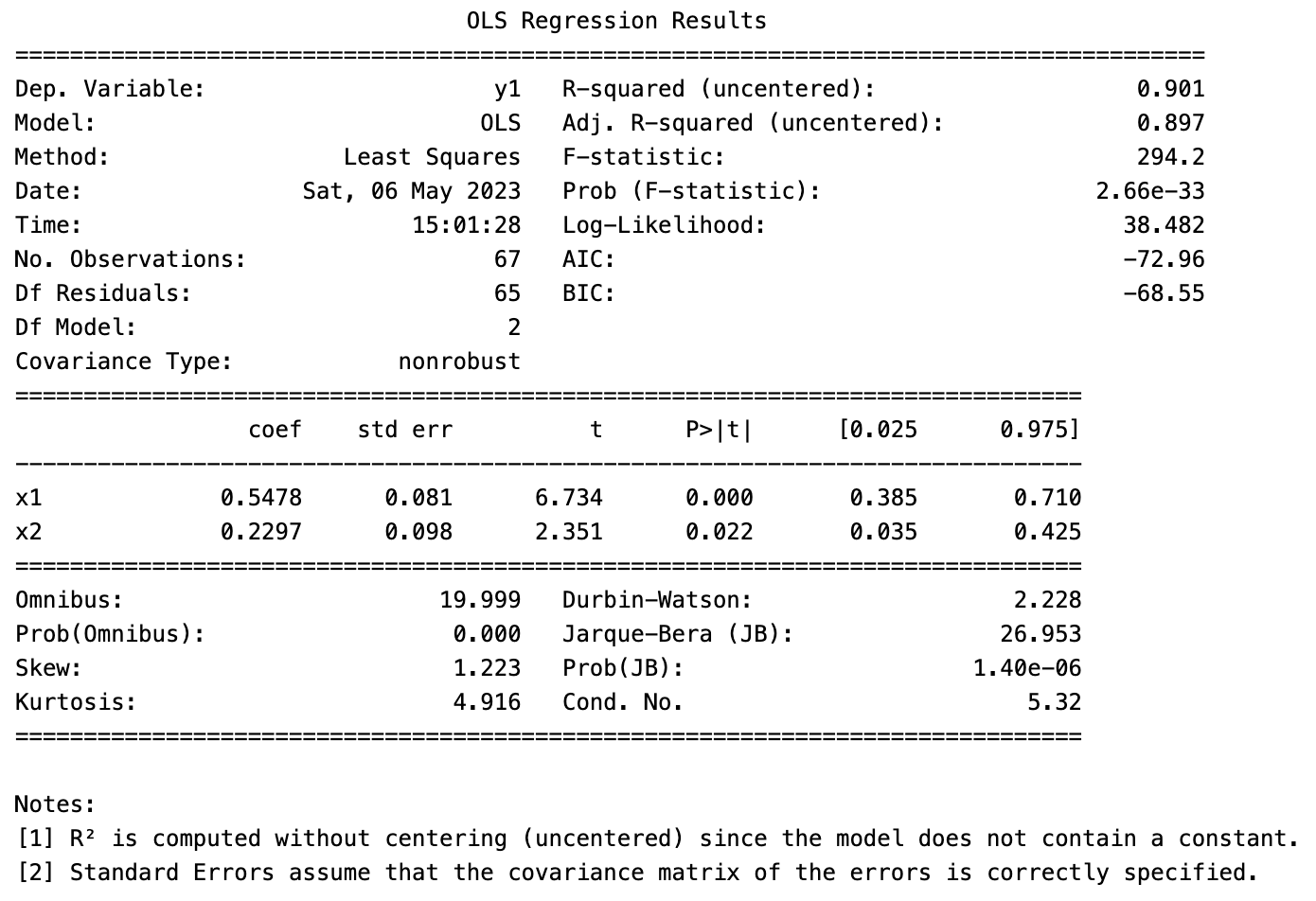


Рисунок 18 - Исходная система уравнений

Конечная система уравнений и веса:

final\_fn1\_nez = pd.concat([x1\_train, x2\_train], axis=1)

CustomLinearRegression(y1\_train, final\_fn1\_nez).get\_coefficients()

x1 0.54776

x2 0.22969

dtype: float64

$$\hat y\_1 = 0.5478x\_{1} + 0.2297x\_{2}$$

fig = make\_subplots(rows=1, cols=2)

fig.update\_yaxes(title\_text='Y1', row=1, col=1)

fig.add\_trace(px.scatter(x=x1\_train, y=y1\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='X1', row=1, col=1)

fig.add\_trace(px.scatter(x=x2\_train, y=y1\_train).data[0], row=1, col=2)

fig.update\_xaxes(title\_text='X2', row=1, col=2)

fig.update\_layout(title\_text="График рассеяния для X1 и X2 от Y1")

fig.show()

График рассеяния для x1 и x2 от y1 представлен на рисунке 19:

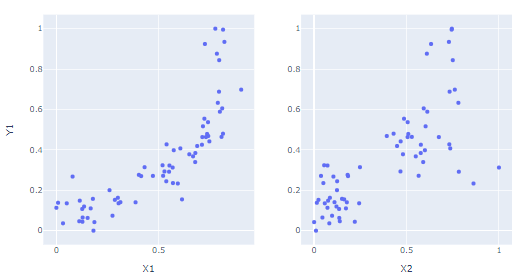


Рисунок 19 - График рассеяния для x1 и x2 от y1

Обратимся ко второму уравнению системы:

Исходная система уравнений представлена на рисунке 20:

fn2\_nez = pd.concat([x1\_train, x2\_train, x3\_train, x4\_train, x5\_train, x6\_train, x7\_train, x8\_train, x9\_train], axis=1)

CustomLinearRegression(y2\_train, fn2\_nez).summary()

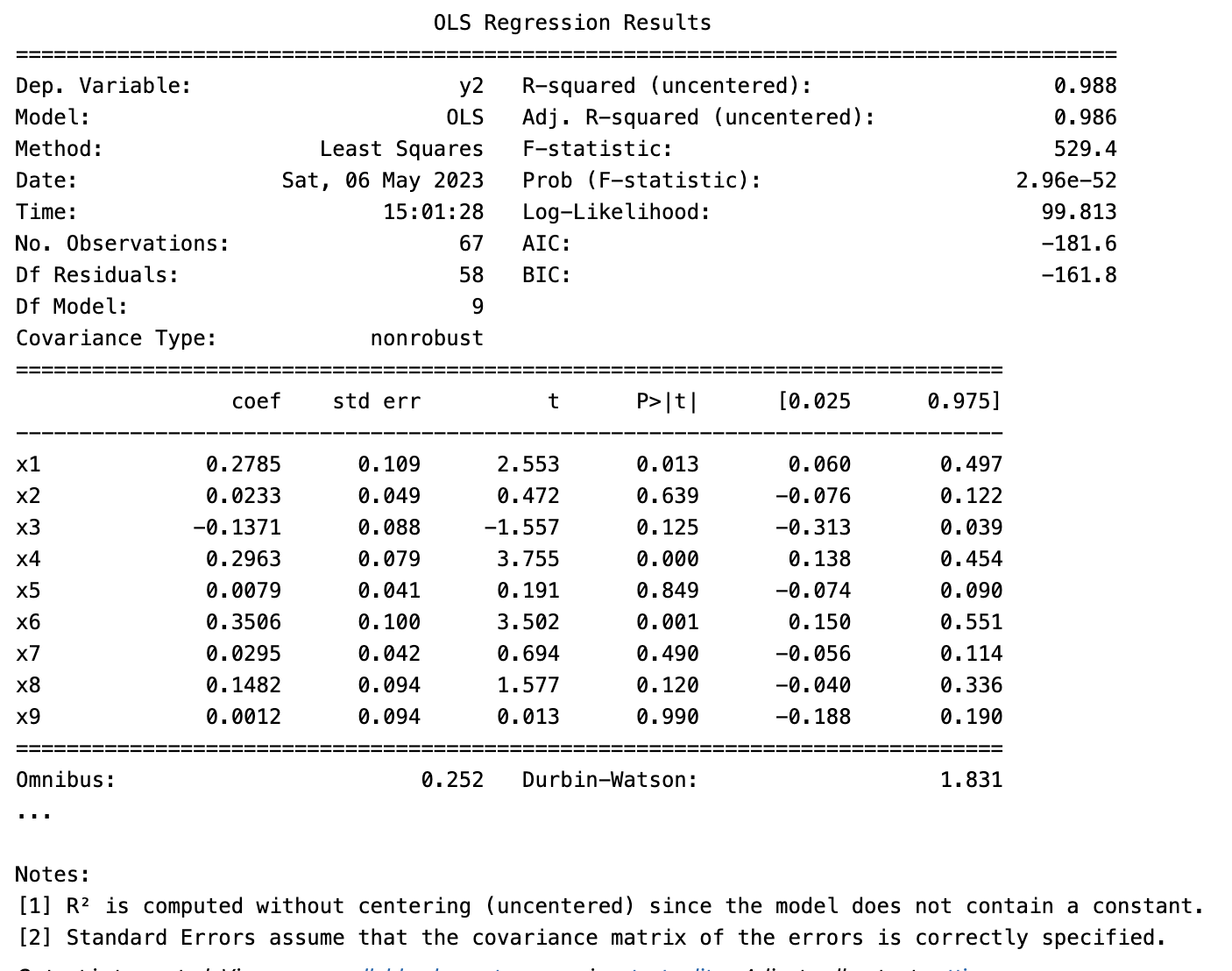


Рисунок 20 - Исходная система уравнений

Исходная система уравнений после отбора признаков методом обратного исключения (Backward elimination) представлена на рисунке 21:

CustomLinearRegression(y2\_train, fn2\_nez).backward\_elimination()

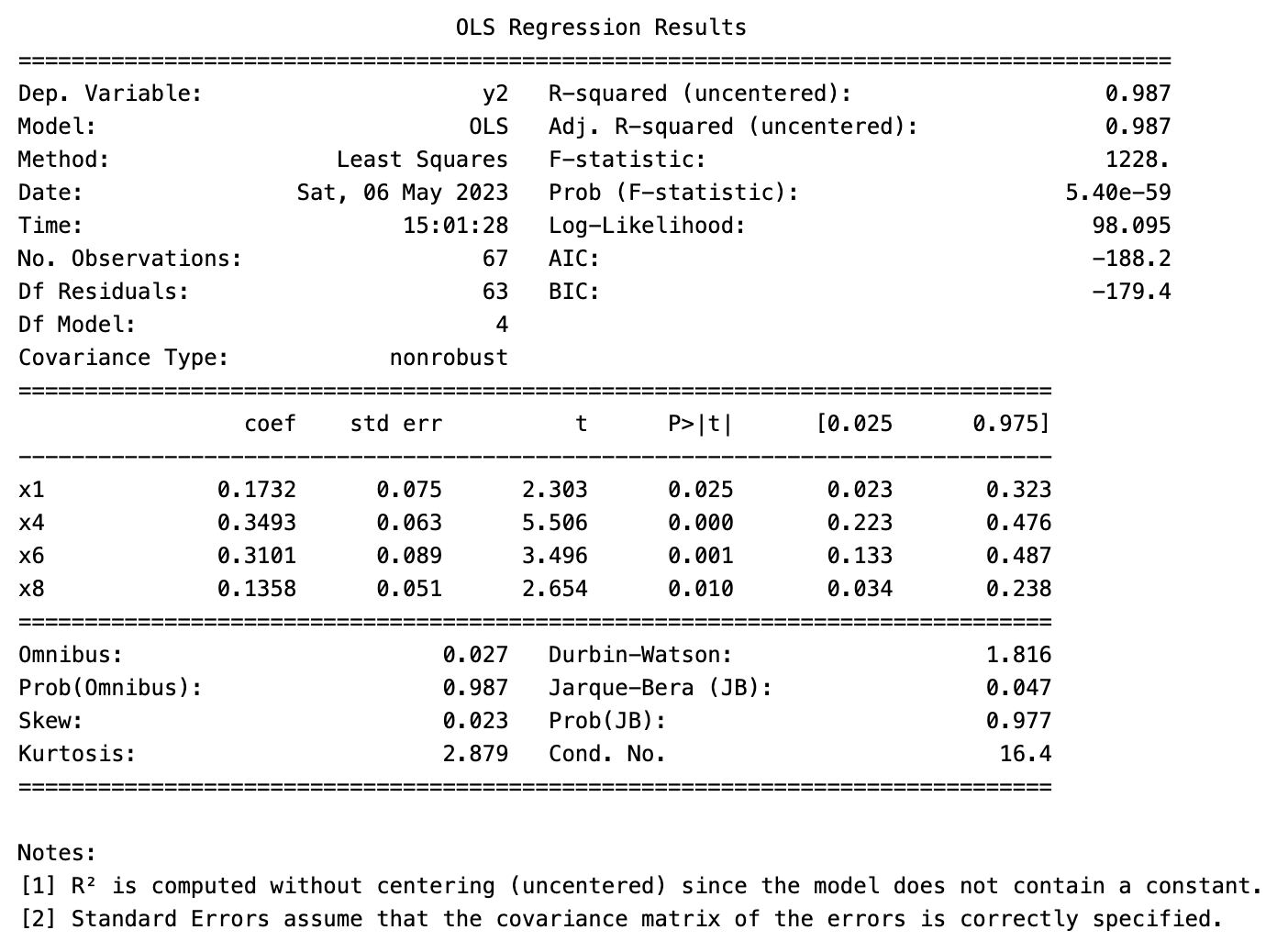


Рисунок 21 - Исходная система уравнений после отбора признаков

Конечная система уравнений и веса:

final\_fn2\_nez = pd.concat([x1\_train, x4\_train, x6\_train, x8\_train], axis=1)

CustomLinearRegression(y2\_train, final\_fn2\_nez).get\_coefficients()

x1 0.173157

x4 0.349321

x6 0.310060

x8 0.135762

dtype: float64

$$\hat y\_2 = 0.1732x\_{1} + 0.3493x\_{4} + 0.3101x\_{6} + 0.1358x\_{8}$$

fig = make\_subplots(rows=2, cols=2)

fig.update\_yaxes(title\_text='Y2', row=1, col=1)

fig.update\_yaxes(title\_text='Y2', row=2, col=1)

fig.add\_trace(px.scatter(x=x1\_train, y=y2\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='X1', row=1, col=1)

fig.add\_trace(px.scatter(x=x4\_train, y=y2\_train).data[0], row=1, col=2)

fig.update\_xaxes(title\_text='X4', row=1, col=2)

fig.add\_trace(px.scatter(x=x6\_train, y=y2\_train).data[0], row=2, col=1)

fig.update\_xaxes(title\_text='X6', row=2, col=1)

fig.add\_trace(px.scatter(x=x8\_train, y=y2\_train).data[0], row=2, col=2)

fig.update\_xaxes(title\_text='X8', row=2, col=2)

fig.update\_layout(height=800, title='Графики рассеяния для X1, X4, X6, X8 от Y2')

fig.show()

График рассеяния для x1, x4, x6, x8 от y2 представлен на рисунке 22:

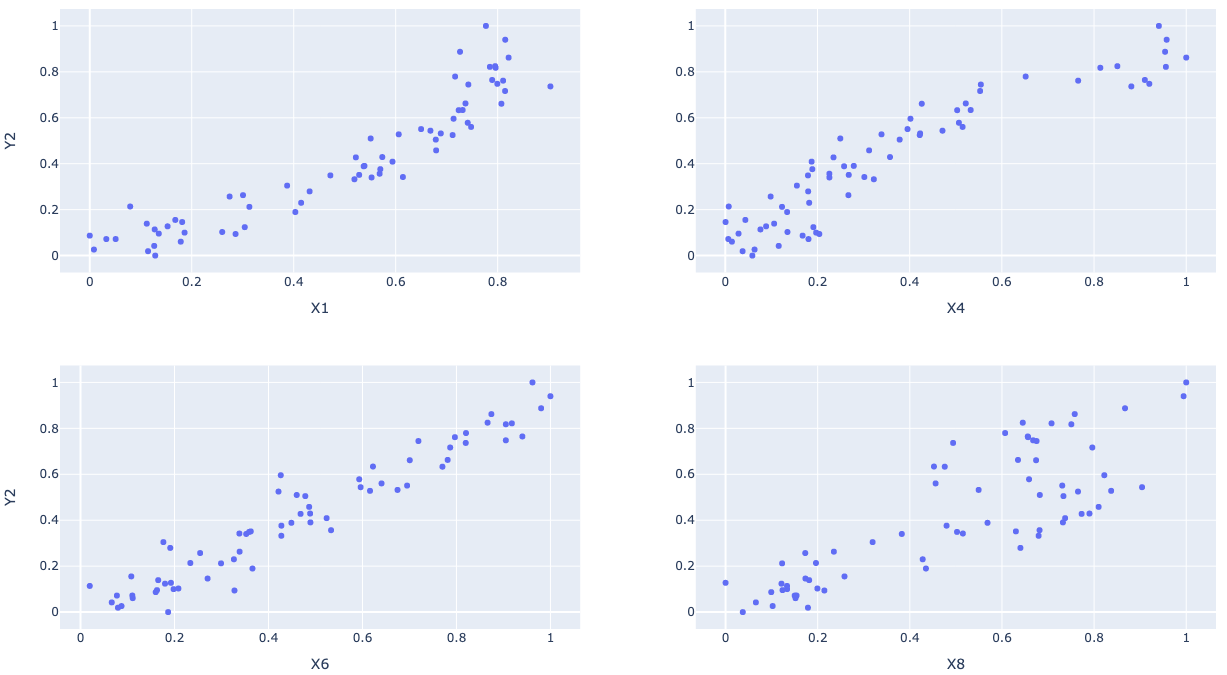


Рисунок 22 - График рассеяния для x1, x4, x6, x8 от y2

Обратимся к третьему уравнению системы:

Исходная система уравнений представлена на рисунке 23:

fn3\_nez = pd.concat(

    [x1\_train, x2\_train, x3\_train, x4\_train, x5\_train, x6\_train, x7\_train, x8\_train, x9\_train, x10\_train, x11\_train],

    axis=1)

CustomLinearRegression(y3\_train, fn3\_nez).summary()

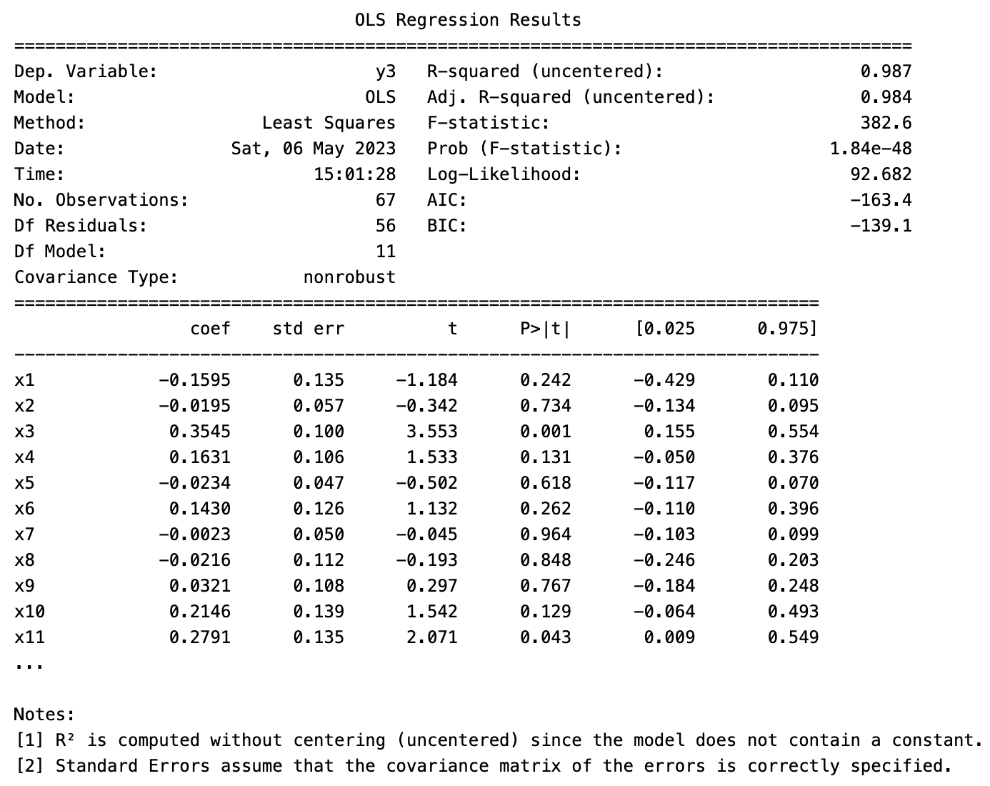


Рисунок 23 - Исходная система уравнений

Исходная система уравнений после отбора признаков методом обратного исключения (Backward elimination) представлена на рисунке 24:

CustomLinearRegression(y3\_train, fn3\_nez).backward\_elimination()

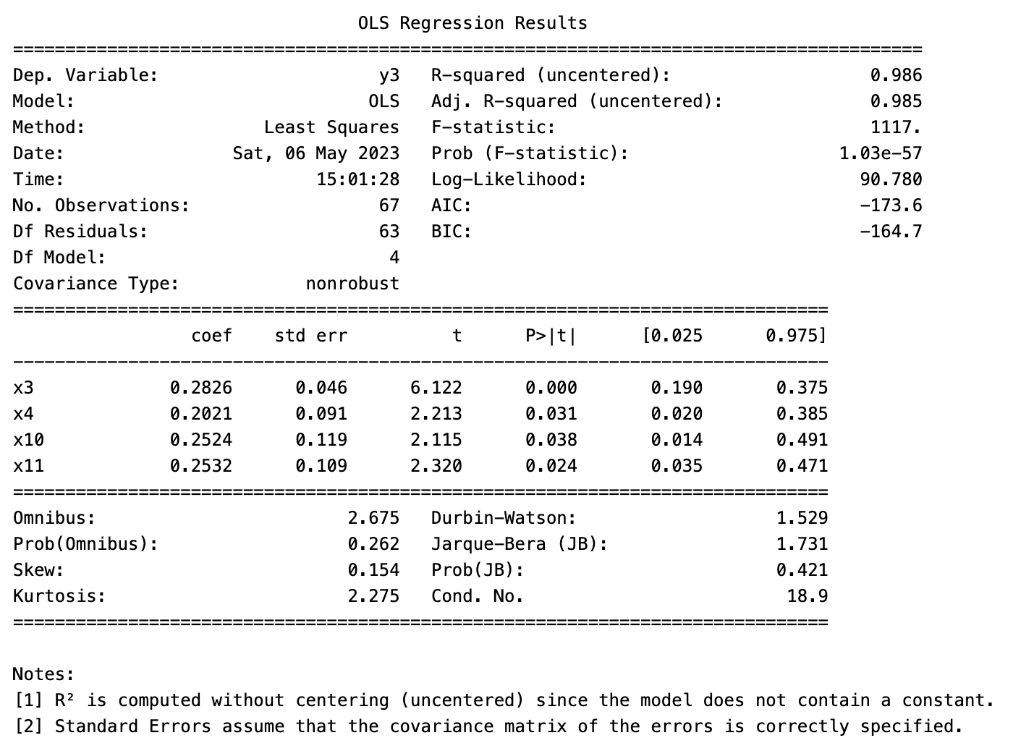


Рисунок 24 - Исходная система уравнений после отбора признаков

Конечная система уравнений и веса:

final\_fn3\_nez = pd.concat([x3\_train, x4\_train, x10\_train, x11\_train], axis=1)

CustomLinearRegression(y3\_train, final\_fn3\_nez).get\_coefficients()

x3 0.282615

x4 0.202117

x10 0.252413

x11 0.253225

dtype: float64

$$\hat y\_3 = 0.2826x\_{3} + 0.2021x\_{4} + 0.2524x\_{10} + 0.2532x\_{11}$$

fig = make\_subplots(rows=2, cols=2)

fig.update\_yaxes(title\_text='Y3', row=1, col=1)

fig.update\_yaxes(title\_text='Y3', row=2, col=1)

fig.add\_trace(px.scatter(x=x3\_train, y=y3\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='X3', row=1, col=1)

fig.add\_trace(px.scatter(x=x4\_train, y=y3\_train).data[0], row=1, col=2)

fig.update\_xaxes(title\_text='X4', row=1, col=2)

fig.add\_trace(px.scatter(x=x10\_train, y=y3\_train).data[0], row=2, col=1)

fig.update\_xaxes(title\_text='X10', row=2, col=1)

fig.add\_trace(px.scatter(x=x11\_train, y=y3\_train).data[0], row=2, col=2)

fig.update\_xaxes(title\_text='X11', row=2, col=2)

fig.update\_layout(height=800, title='Графики рассеяния для X3, X4, X10, X11 от Y3')

fig.show()

График рассеяния для x3, x4, x10, x11 от y3 представлен на рисунке 25:

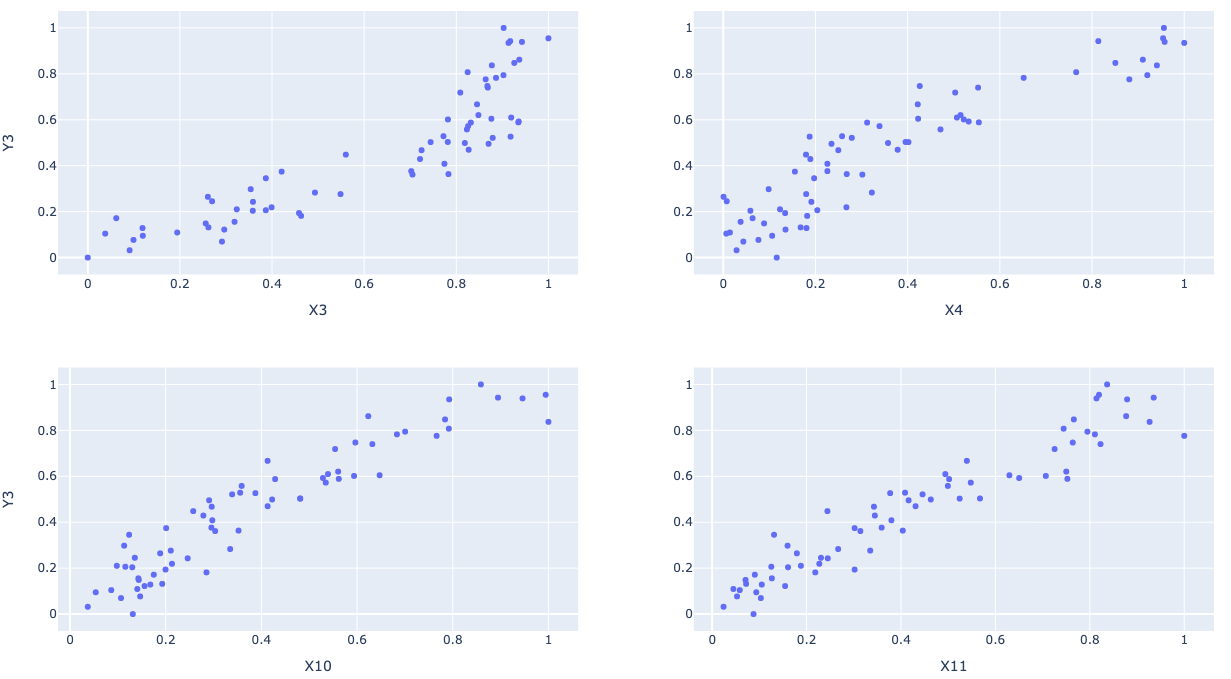


Рисунок 25 - График рассеяния для x3, x4, x10, x11 от y3

* + - * 1. **Система рекурсивных уравнений**

Рассмотрим вторую систему рекурсивных уравнений. Обратимся к первому уравнению системы:

Исходная система уравнений представлена на рисунке 26:

fn1\_rec = pd.concat([x1\_train, x2\_train], axis=1)

CustomLinearRegression(y1\_train, fn1\_rec).summary()

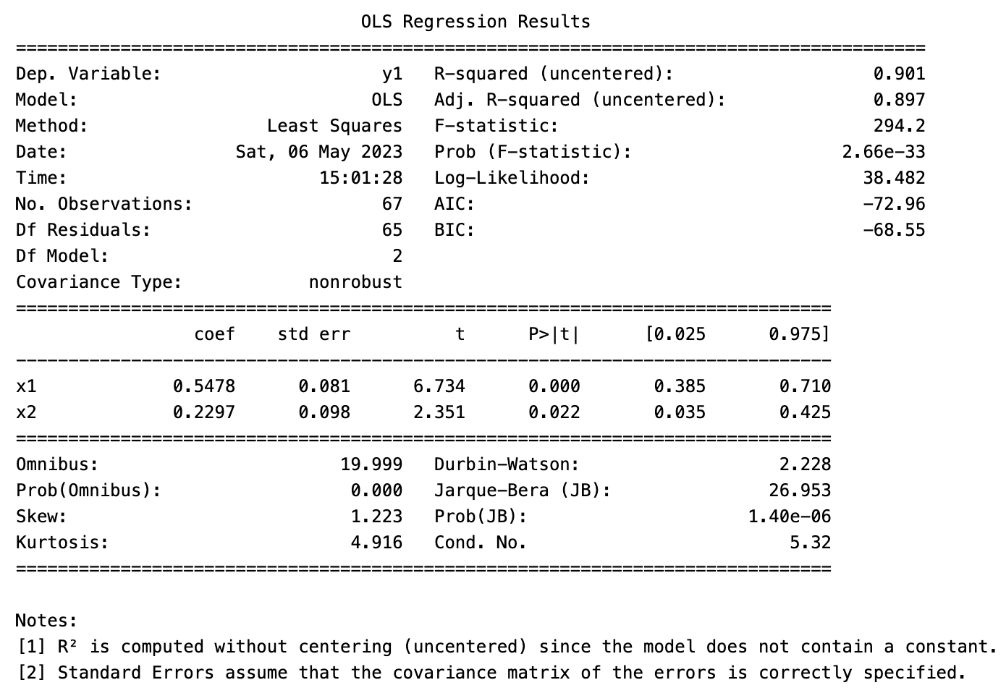


Рисунок 26 - Исходная система уравнений

Конечная система уравнений и веса:

final\_fn1\_rec = pd.concat([x1\_train, x2\_train], axis=1)

CustomLinearRegression(y1\_train, final\_fn1\_rec).get\_coefficients()

x1 0.54776

x2 0.22969

dtype: float64

$$\hat y\_1 = 0.5478x\_{1} + 0.2297x\_{2}$$

fig = make\_subplots(rows=1, cols=2)

fig.update\_yaxes(title\_text='Y1', row=1, col=1)

fig.add\_trace(px.scatter(x=x1\_train, y=y1\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='X1', row=1, col=1)

fig.add\_trace(px.scatter(x=x2\_train, y=y1\_train).data[0], row=1, col=2)

fig.update\_xaxes(title\_text='X2', row=1, col=2)

fig.update\_layout(title\_text="График рассеяния для X1 и X2 от Y1")

fig.show()

График рассеяния для x1 и x2 от y1 представлен на рисунке 27:

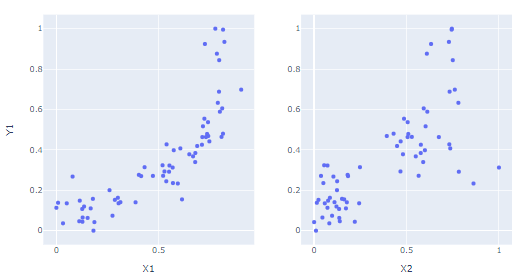


Рисунок 27 - График рассеяния для x1 и x2 от y1

Обратимся ко второму уравнению системы:

Исходная система уравнений представлена на рисунке 28:

fn2\_rec = pd.concat(

    [y1\_train, x1\_train, x2\_train, x3\_train, x4\_train, x5\_train, x6\_train, x7\_train, x8\_train, x9\_train], axis=1)

CustomLinearRegression(y2\_train, fn2\_rec).summary()

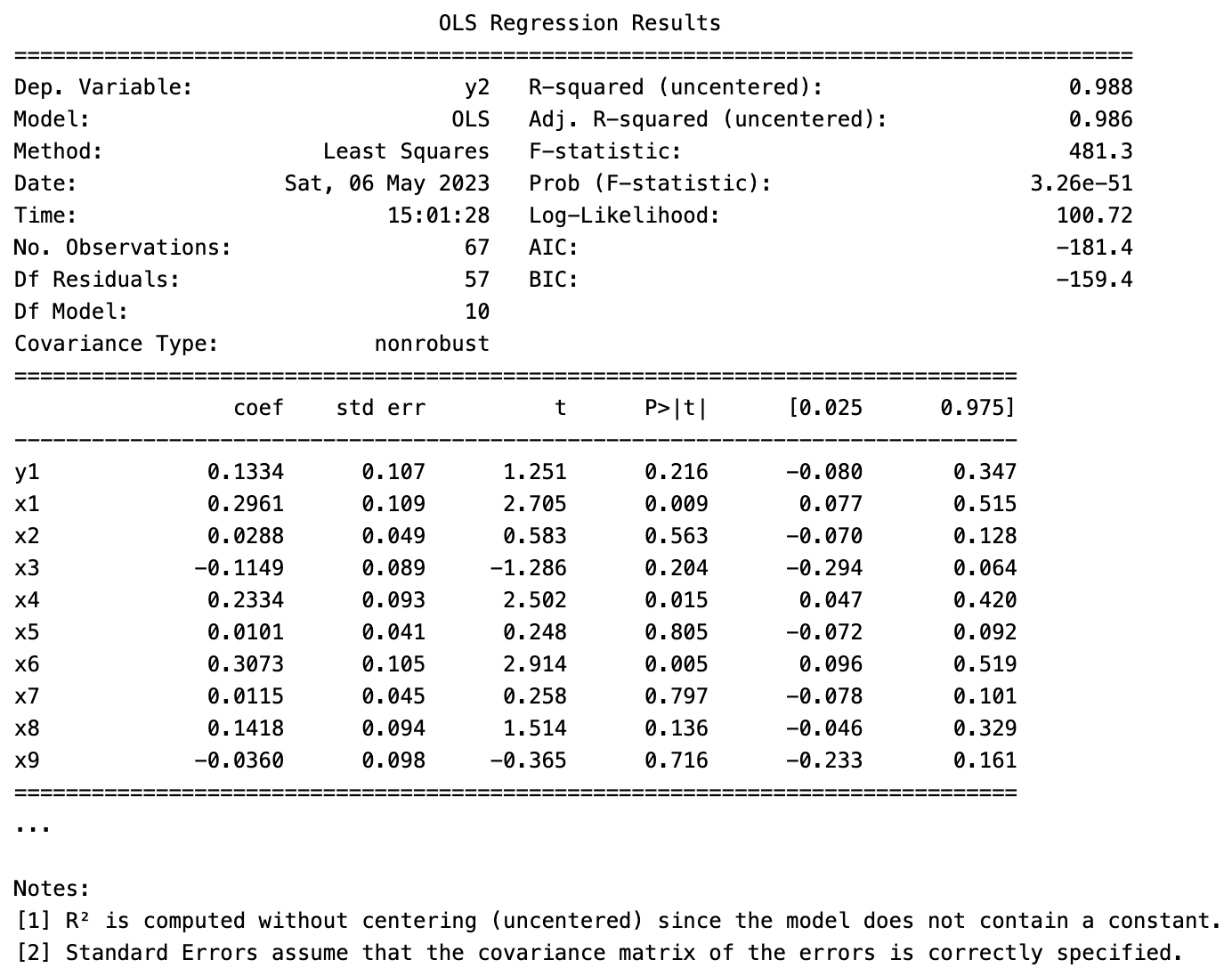


Рисунок 28 - Исходная система уравнений

Исходная система уравнений после отбора признаков методом обратного исключения (Backward elimination) представлена на рисунке 29:

CustomLinearRegression(y2\_train, fn2\_rec).backward\_elimination(start=1)

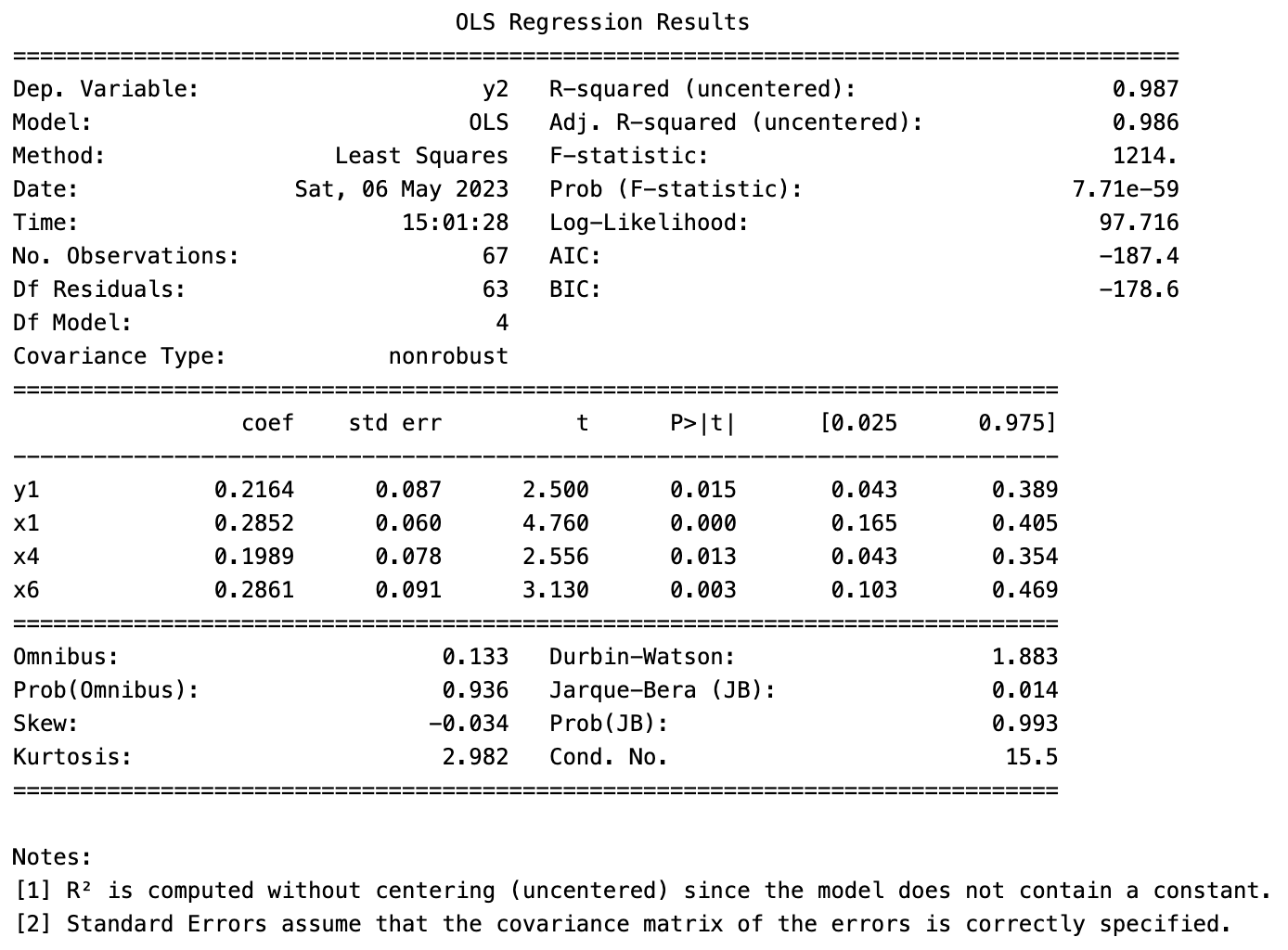


Рисунок 29 - Исходная система уравнений после отбора признаков

Конечная система уравнений и веса:

final\_fn2\_rec = pd.concat([y1\_train, x1\_train, x4\_train, x6\_train], axis=1)

CustomLinearRegression(y2\_train, final\_fn2\_rec).get\_coefficients()

y1 0.216365

x1 0.285226

x4 0.198917

x6 0.286120

dtype: float64

$$\hat y\_2 = 0.2164y\_{1} + 0.2852x\_{1} + 0.1989x\_{4} + 0.2861x\_{6}$$

fig = make\_subplots(rows=2, cols=2)

fig.update\_yaxes(title\_text='Y2', row=1, col=1)

fig.update\_yaxes(title\_text='Y2', row=2, col=1)

fig.add\_trace(px.scatter(x=y1\_train, y=y2\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='Y1', row=1, col=1)

fig.add\_trace(px.scatter(x=x1\_train, y=y2\_train).data[0], row=1, col=2)

fig.update\_xaxes(title\_text='X1', row=1, col=2)

fig.add\_trace(px.scatter(x=x4\_train, y=y2\_train).data[0], row=2, col=1)

fig.update\_xaxes(title\_text='X4', row=2, col=1)

fig.add\_trace(px.scatter(x=x6\_train, y=y2\_train).data[0], row=2, col=2)

fig.update\_xaxes(title\_text='X6', row=2, col=2)

fig.update\_layout(height=1000, title='Графики рассеяния для Y1, X1, X4, X6 от Y2')

fig.show()

График рассеяния для x1, x4, x6, x8 от y2 представлен на рисунке 30:

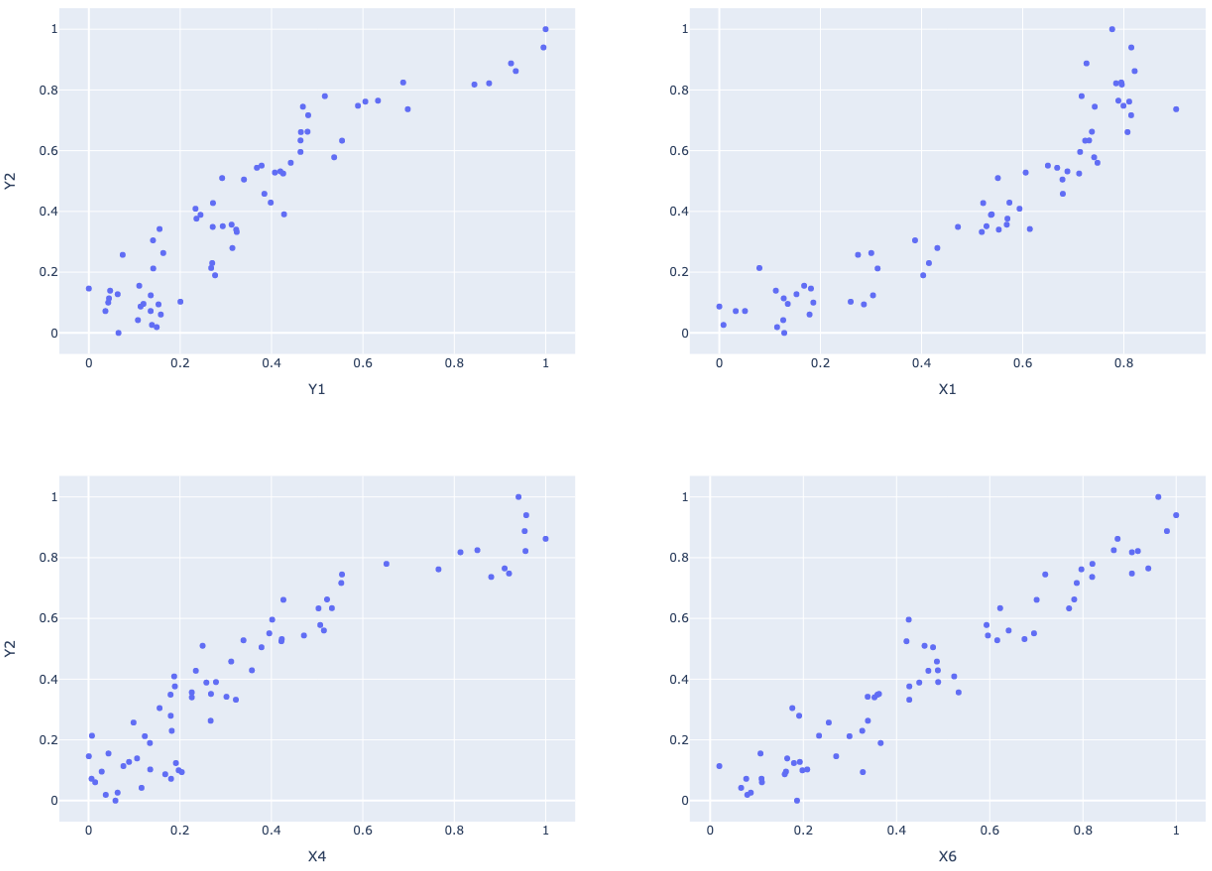


Рисунок 30 - Графики рассеяния для y1, x1, x4, x6 от y2

Обратимся к третьему уравнению системы:

Исходная система уравнений представлена на рисунке 31:

fn3\_rec = pd.concat([y1\_train, y2\_train, x11\_train], axis=1)

CustomLinearRegression(y3\_train, fn3\_rec).summary()

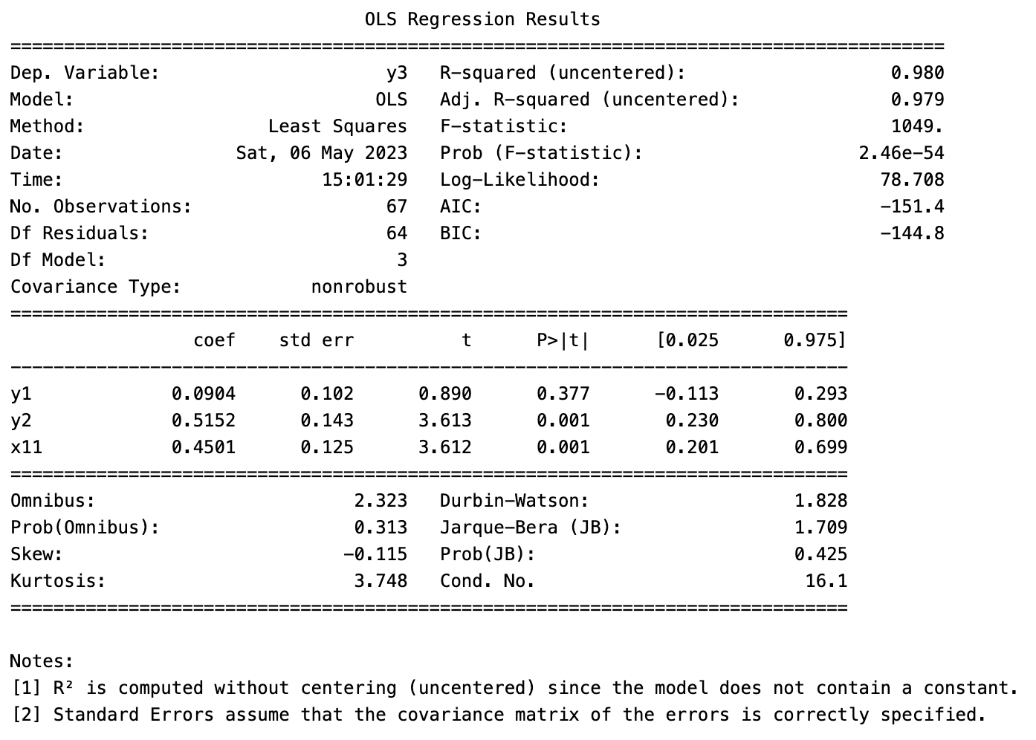


Рисунок 31 - Исходная система уравнений

Исходная система уравнений после отбора признаков методом обратного исключения (Backward elimination) представлена на рисунке 32:

CustomLinearRegression(y3\_train, fn3\_rec).backward\_elimination(start=0)

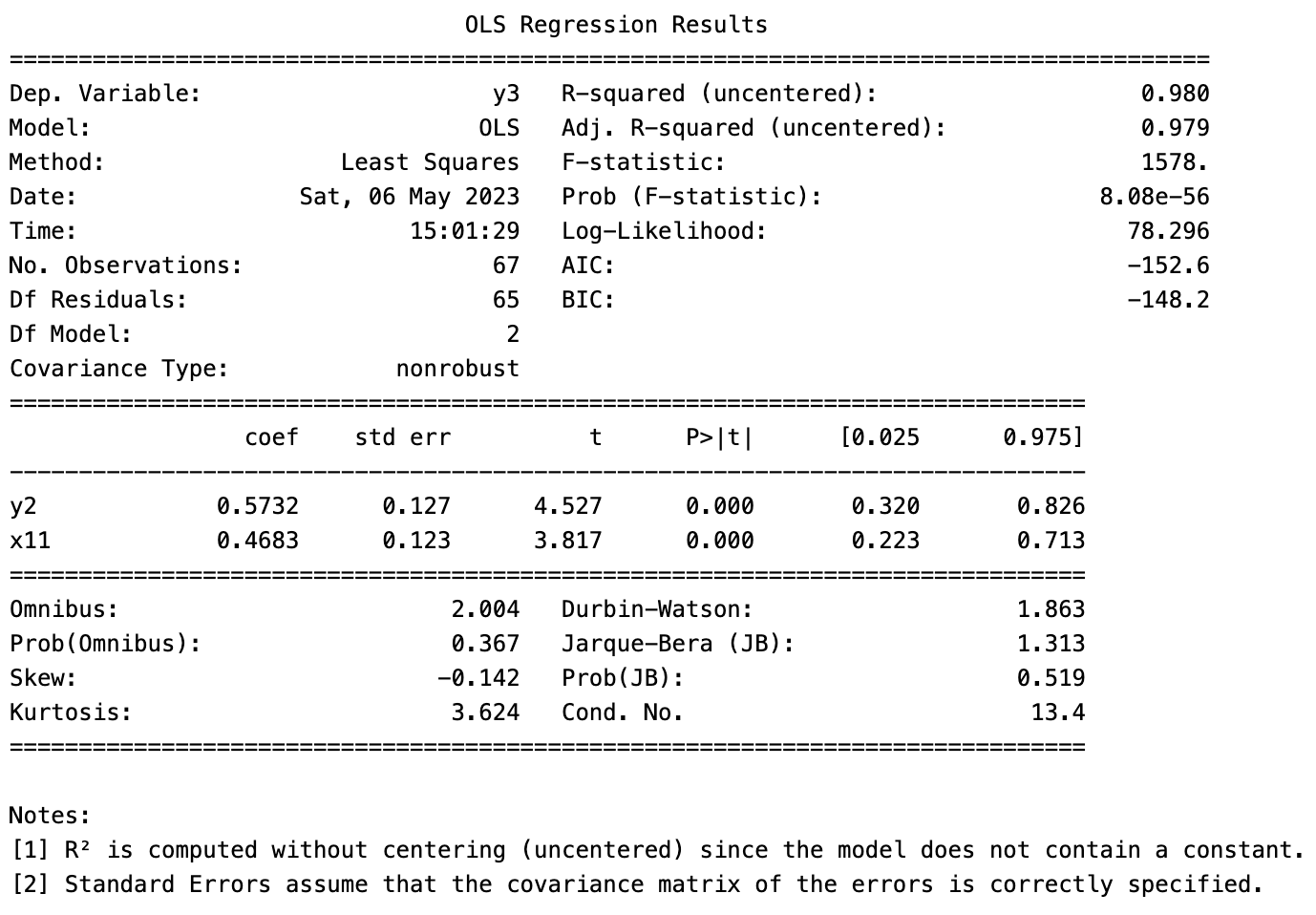


Рисунок 32 - Исходная система уравнений после отбора признаков

Конечная система уравнений и веса:

final\_fn3\_rec = pd.concat([y2\_train, x11\_train], axis=1)

CustomLinearRegression(y3\_train, final\_fn3\_rec).get\_coefficients()

y2 0.573236

x11 0.468336

dtype: float64

$$\hat y\_3 = 0.5732y\_{2} + 0.4683x\_{11}$$

fig = make\_subplots(rows=1, cols=2)

fig.update\_yaxes(title\_text='Y3', row=1, col=1)

fig.add\_trace(px.scatter(x=y2\_train, y=y3\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='Y2', row=1, col=1)

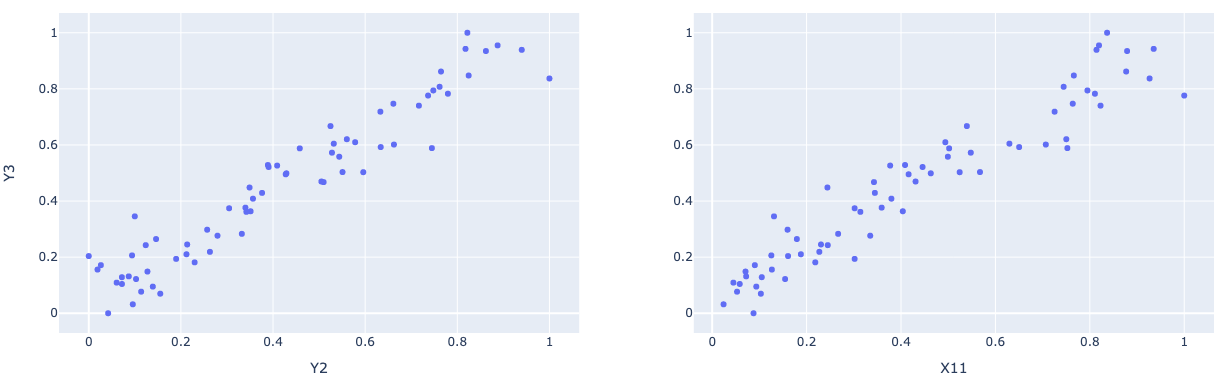
fig.add\_trace(px.scatter(x=x11\_train, y=y3\_train).data[0], row=1, col=2)

fig.update\_xaxes(title\_text='X11', row=1, col=2)

fig.update\_layout(height=500, title='Графики рассеяния для Y2, X11 от Y3')

fig.show()

График рассеяния для y2, x11 от y3 представлен на рисунке 33:



*Рисунок 33 - График рассеяния для y2, x11 от y3*

* + - * 1. **Система одновременных уравнений**

Рассмотрим третью систему одновременных уравнений. Обратимся к первому уравнению системы:

Исходная система уравнений представлена на рисунке 34:

fn1\_odn = pd.concat([x1\_train, x2\_train], axis=1)

CustomLinearRegression(y1\_train, fn1\_odn, model\_type=Poisson).summary()

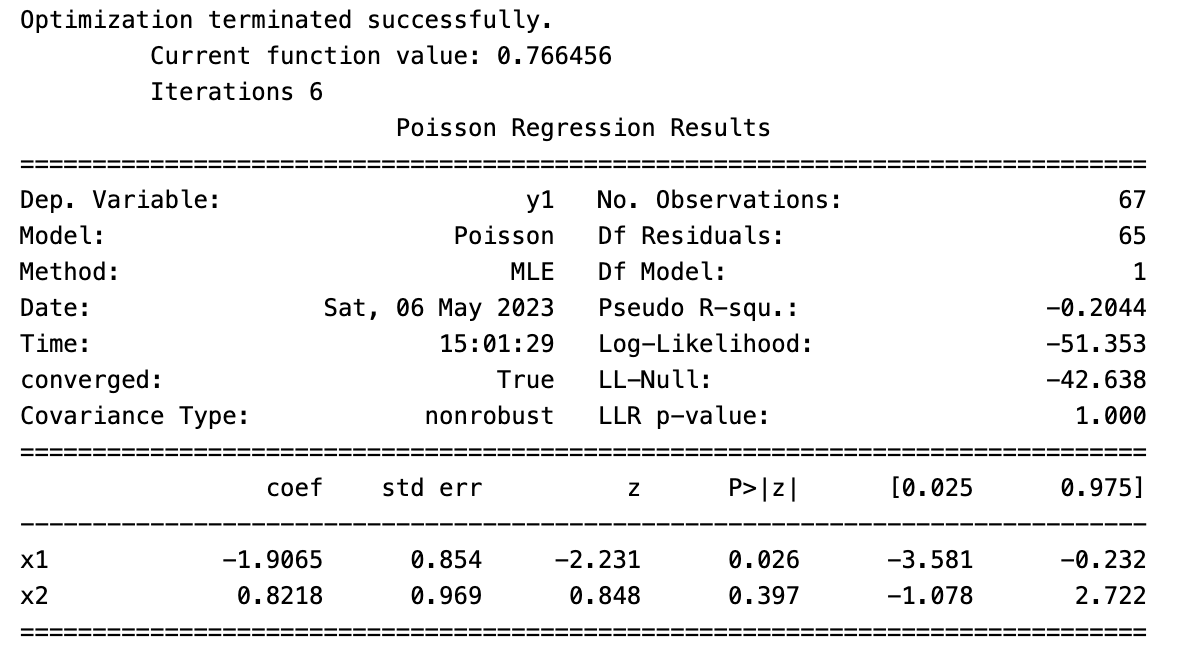


Рисунок 34 - Исходная система уравнений

Исходная система уравнений после отбора признаков методом обратного исключения (Backward elimination) представлена на рисунке 35:

CustomLinearRegression(y1\_train, fn1\_odn, model\_type=Poisson).backward\_elimination()

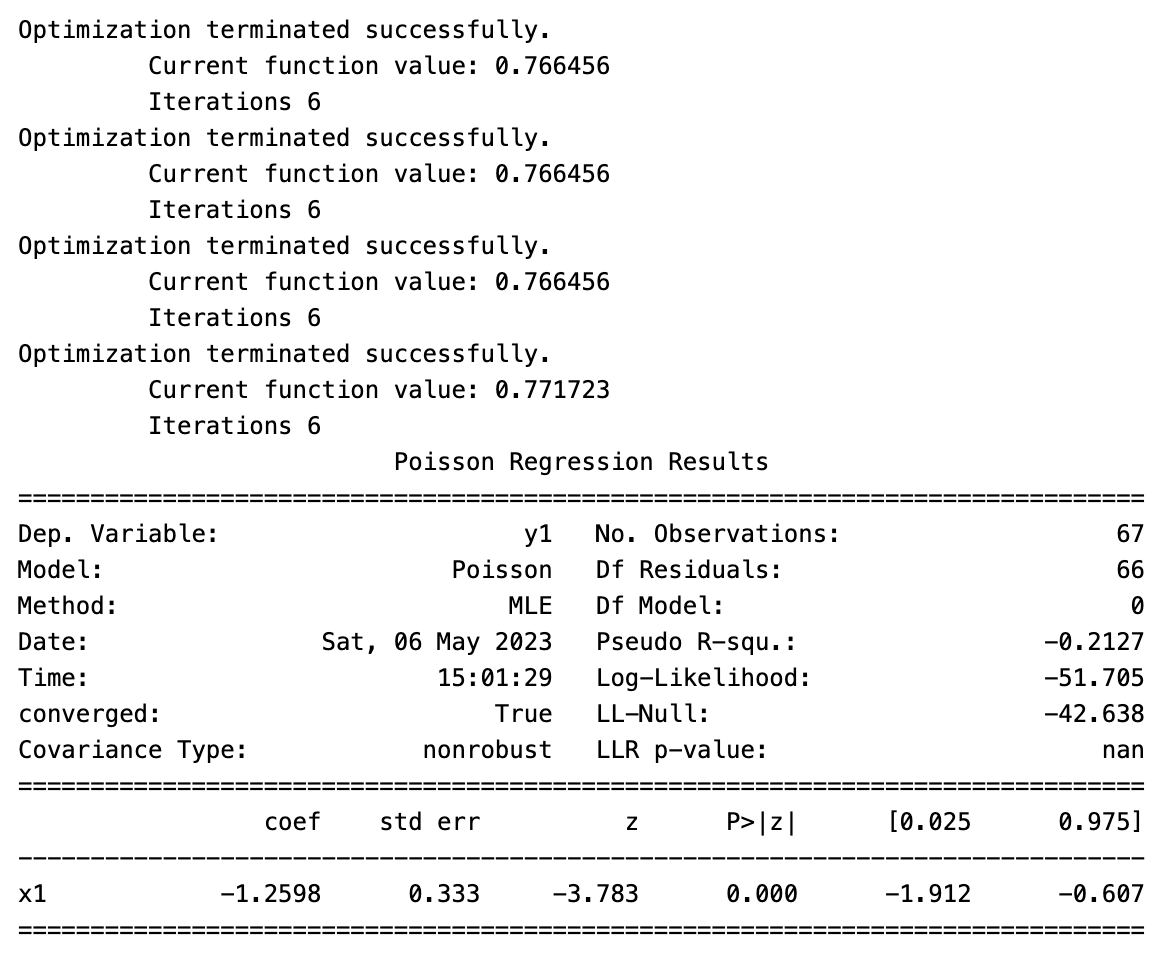


Рисунок 35 - Исходная система уравнений после отбора признаков

Конечная система уравнений и веса:

final\_fn1\_odn = pd.concat([x1\_train], axis=1)

CustomLinearRegression(y1\_train, final\_fn1\_odn, model\_type=Poisson).get\_coefficients()

Optimization terminated successfully.

Current function value: 0.771723

Iterations 6

x1 -1.259768

dtype: float64

$$\hat y\_1 = -1.2598x\_{1}$$

fig = make\_subplots(rows=1, cols=1)

fig.update\_yaxes(title\_text='Y1', row=1, col=1)

fig.add\_trace(px.scatter(x=x1\_train, y=y1\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='X1', row=1, col=1)

fig.update\_layout(height=500, title='Графики рассеяния для X1 от Y1')

fig.show()

График рассеяния для x1 от y1 представлен на рисунке 36:

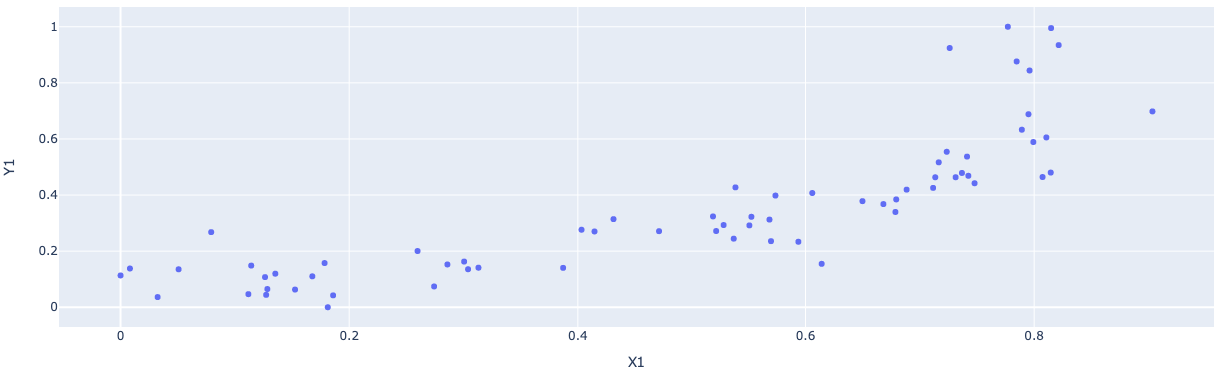


Рисунок 36 - График рассеяния для x1 от y1

Обратимся ко второму уравнению системы:

Исходная система уравнений представлена на рисунке 37:

fn2\_odn = pd.concat([y1\_train, x3\_train, x4\_train, x5\_train, x6\_train, x7\_train, x8\_train, x9\_train], axis=1)

CustomLinearRegression(y2\_train, fn2\_odn, model\_type=Poisson).summary()



Рисунок 37 - Исходная система уравнений

Исходная система уравнений после отбора признаков методом обратного исключения (Backward elimination) представлена на рисунке 38:

CustomLinearRegression(y2\_train, fn2\_odn, model\_type=Poisson).backward\_elimination(start=1)

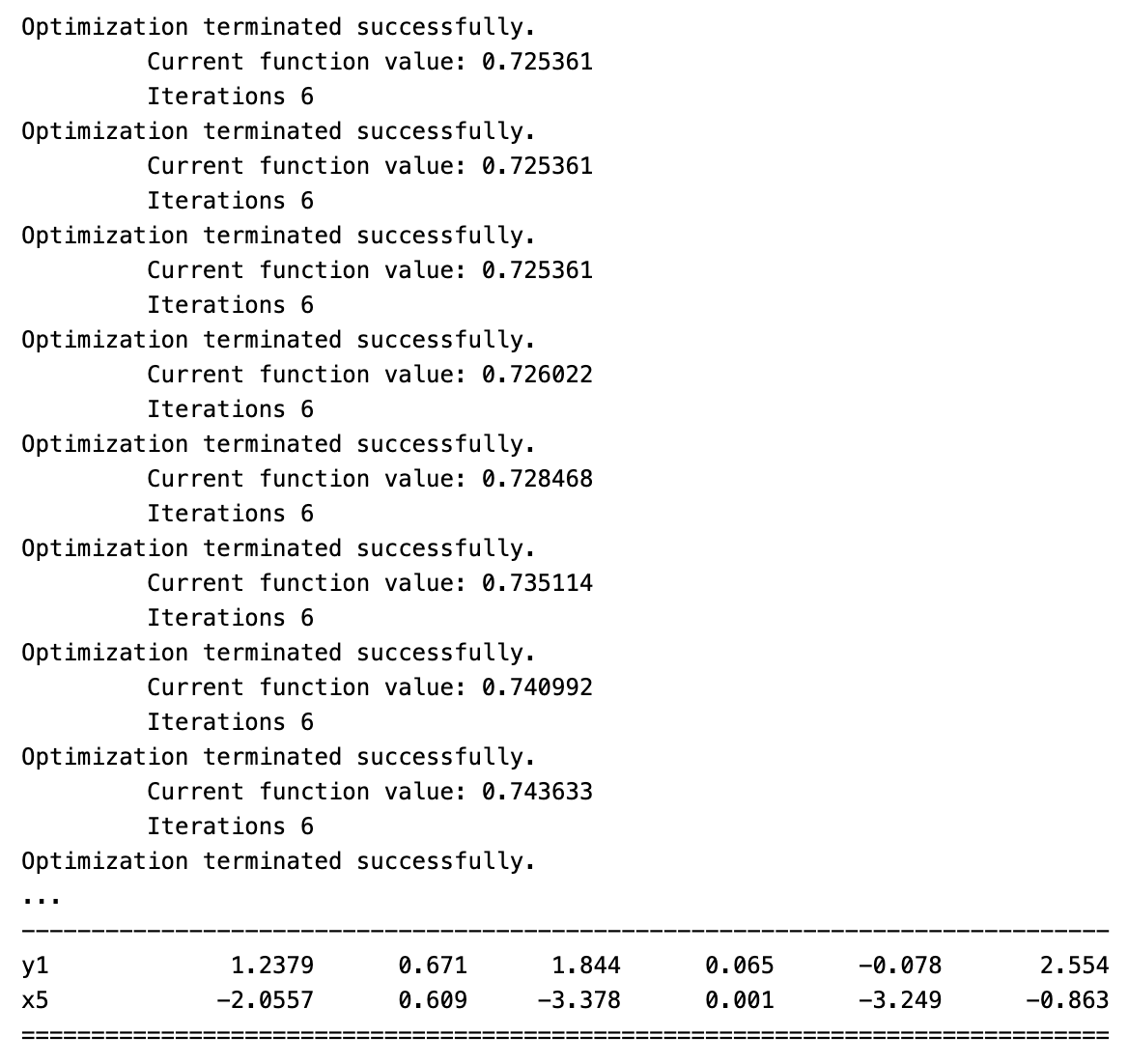


Рисунок 38 - Исходная система уравнений после отбора признаков

Конечная система уравнений и веса:

final\_fn2\_odn = pd.concat([y1\_train, x5\_train], axis=1)

CustomLinearRegression(y2\_train, final\_fn2\_odn, model\_type=Poisson).get\_coefficients()

Optimization terminated successfully.

Current function value: 0.747961

Iterations 6

y1 1.237865

x5 -2.055679

dtype: float64

$$\hat y\_2 = 1.2379y\_{1} + -2.0557x\_{5}$$

fig = make\_subplots(rows=1, cols=2)

fig.update\_yaxes(title\_text='Y2', row=1, col=1)

fig.add\_trace(px.scatter(x=y1\_train, y=y2\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='Y1', row=1, col=1)

fig.add\_trace(px.scatter(x=x5\_train, y=y2\_train).data[0], row=1, col=2)

fig.update\_xaxes(title\_text='X5', row=1, col=2)

fig.update\_layout(height=500, title='Графики рассеяния для Y1, X5 от Y2')

fig.show()

График рассеяния для x1, x4, x6, x8 от y2 представлен на рисунке 39:

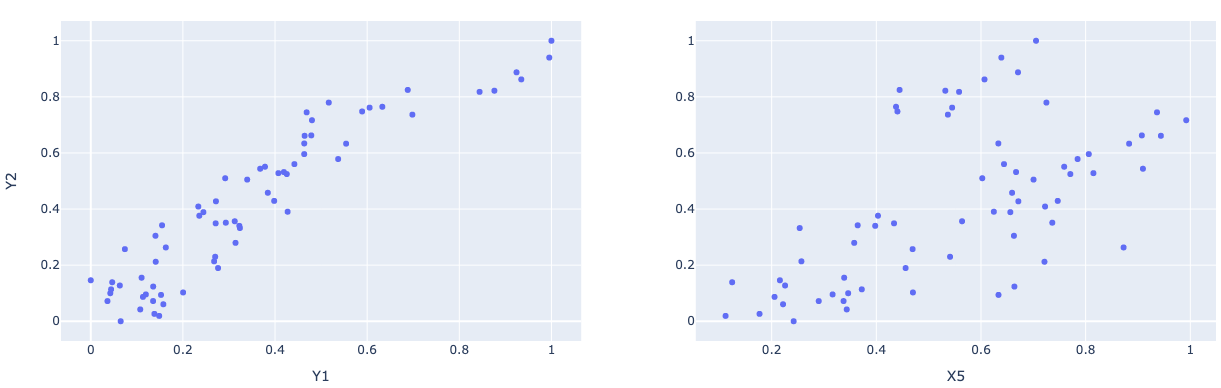


Рисунок 39 - Графики рассеяния для y1, x5 от y2

Обратимся к третьему уравнению системы:

Исходная система уравнений представлена на рисунке 40:

fn3\_odn = pd.concat([y1\_train, x10\_train, x11\_train], axis=1)

CustomLinearRegression(y3\_train, fn3\_odn, model\_type=Poisson).summary()



Рисунок 40 - Исходная система уравнений

Исходная система уравнений после отбора признаков методом обратного исключения (Backward elimination) представлена на рисунке 41:

CustomLinearRegression(y3\_train, fn3\_odn, model\_type=Poisson).backward\_elimination(start=0)

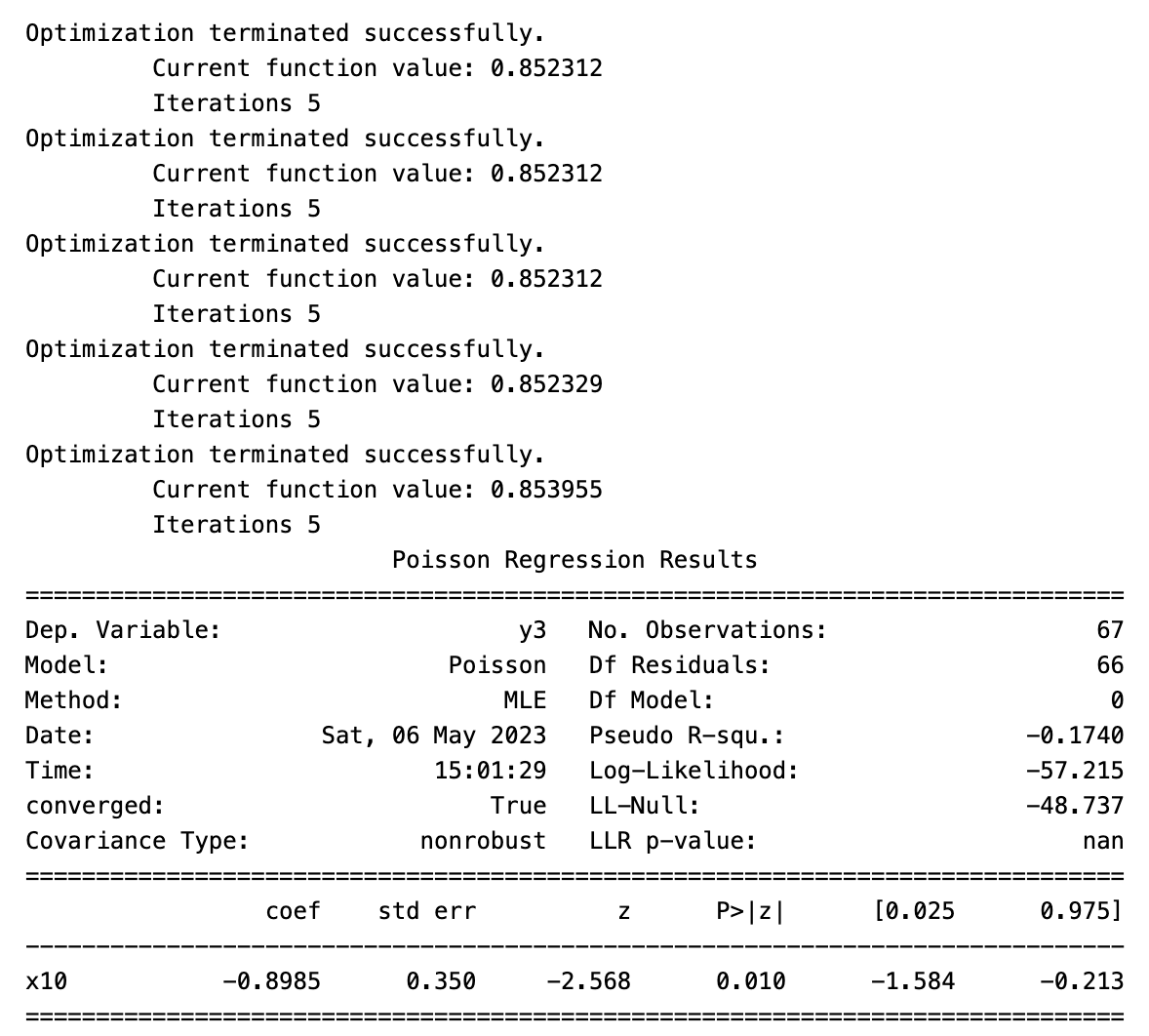


Рисунок 41 - Исходная система уравнений после отбора признаков

Конечная система уравнений и веса:

final\_fn3\_odn = pd.concat([y1\_train, x10\_train], axis=1)

CustomLinearRegression(y3\_train, final\_fn3\_odn, model\_type=Poisson).get\_coefficients()

Optimization terminated successfully.

Current function value: 0.852329

Iterations 5

y1 0.904614

x10 -1.717046

dtype: float64

$$\hat y\_3 = 0.9046y\_{1} + -1.7170x\_{10}$$

fig = make\_subplots(rows=1, cols=2)

fig.update\_yaxes(title\_text='Y3', row=1, col=1)

fig.add\_trace(px.scatter(x=y1\_train, y=y3\_train).data[0], row=1, col=1)

fig.update\_xaxes(title\_text='Y1', row=1, col=1)

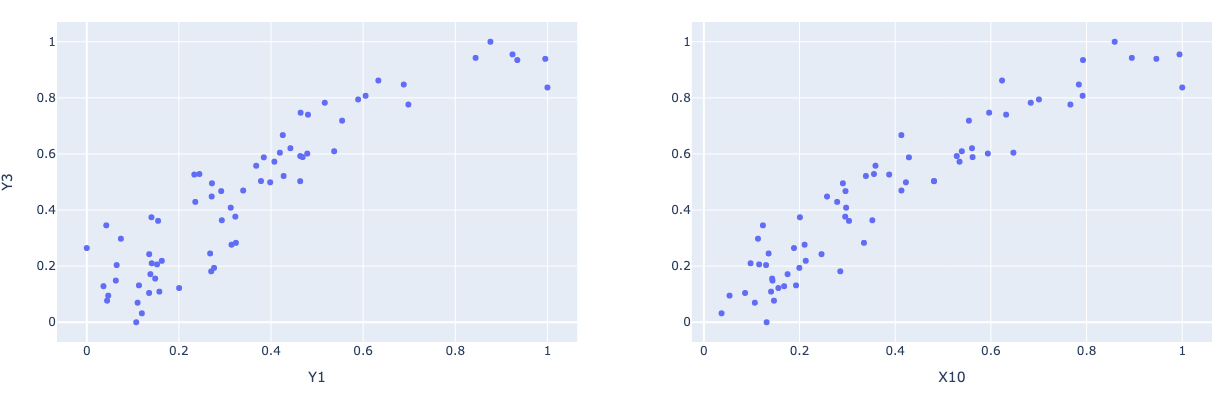
fig.add\_trace(px.scatter(x=x10\_train, y=y3\_train).data[0], row=1, col=2)

fig.update\_xaxes(title\_text='X10', row=1, col=2)

fig.update\_layout(height=500, title='Графики рассеяния для Y1, X10 от Y3')

fig.show()

График рассеяния для y1, x10 от y3 представлен на рисунке 42:



*Рисунок 42 - График рассеяния для y1, x10 от y3*

* + 1. **Прогнозирование**

На данный момент мы прошли практически все этапы построения модели. Осталось только прогнозирование. Для этого нам необходимо построить системы уравнений, которые будут состоять из уравнений, полученных путем отбора параметров на предыдущих этапах работы. После этого мы сможем провести прогнозирование и сравнить прогнозные значения с их реальными значениями. Проведем прогнозирование для каждого уравнения по отдельности, а также для всех систем уравнений в целом.

Структура этапа прогнозирования:

* Построение системы уравнений;
* Вывод параметров для каждого уравнения;
* Прогнозирование для каждого уравнения и оценка резульатов по ряду метрик.
  + - 1. **Система независимых уравнений**

Система независимых уравнений с полученными значениями параметров представлена на рисунке 43:

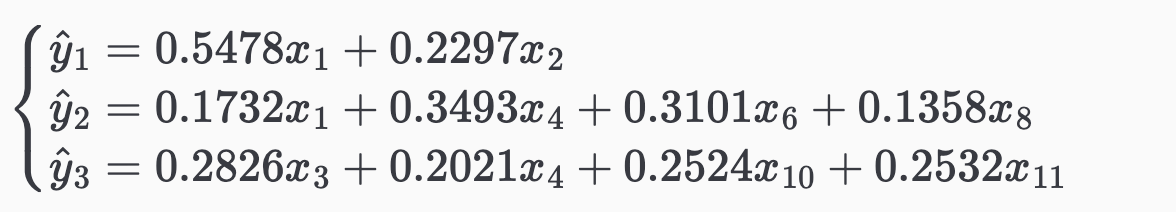
****

Рисунок 43 - Система независимых уравнений с полученными значениями параметров

print(f'Параметры первого: {final\_fn1\_nez.columns.values.tolist()}\n'

      f'Параметры второго: {final\_fn2\_nez.columns.values.tolist()}\n'

      f'Параметры третьего: {final\_fn3\_nez.columns.values.tolist()}')

Параметры первого: ['x1', 'x2']

Параметры второго: ['x1', 'x4', 'x6', 'x8']

Параметры третьего: ['x3', 'x4', 'x10', 'x11']

Первое уравнение:

CustomLinearRegression(y1\_train, final\_fn1\_nez).predict(pd.concat([x1\_test, x2\_test], axis=1), y1\_test)

Коэффициент детерминации (R^2): 78.5%

Cредняя ошибка аппроксимации (MAE): 8.3%

Cредняя квадратичная ошибка (MSE): 1.1%

Второе уравнение:

CustomLinearRegression(y2\_train, final\_fn2\_nez).predict(pd.concat([x1\_test, x4\_test, x6\_test, x8\_test], axis=1), y2\_test)

Коэффициент детерминации (R^2): 89.8%

Cредняя ошибка аппроксимации (MAE): 6.3%

Cредняя квадратичная ошибка (MSE): 0.6%

Третье уравнение:

CustomLinearRegression(y3\_train, final\_fn3\_nez).predict(pd.concat([x3\_test, x4\_test, x10\_test, x11\_test], axis=1), y3\_test)

Коэффициент детерминации (R^2): 95.5%

Cредняя ошибка аппроксимации (MAE): 4.5%

Cредняя квадратичная ошибка (MSE): 0.3%

* + - 1. **Система рекурсивных уравнений**

Система независимых уравнений с полученными значениями параметров представлена на рисунке 44:

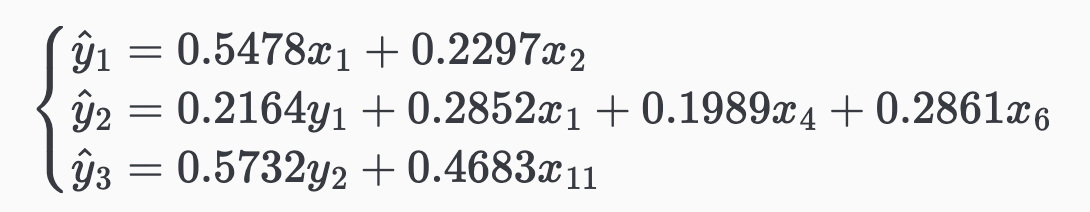


Рисунок 44 - Система независимых уравнений с полученными значениями параметров

print(f'Параметры первого: {final\_fn1\_rec.columns.values.tolist()}\n'

      f'Параметры второго: {final\_fn2\_rec.columns.values.tolist()}\n'

      f'Параметры третьего: {final\_fn3\_rec.columns.values.tolist()}')

Параметры первого: ['x1', 'x2']

Параметры второго: ['y1', 'x1', 'x4', 'x6']

Параметры третьего: ['y2', 'x11']

Первое уравнение:

CustomLinearRegression(y1\_train, final\_fn1\_rec).predict(pd.concat([x1\_test, x2\_test], axis=1), y1\_test)

Коэффициент детерминации (R^2): 78.5%

Cредняя ошибка аппроксимации (MAE): 8.3%

Cредняя квадратичная ошибка (MSE): 1.1%

Второе уравнение:

CustomLinearRegression(y2\_train, final\_fn2\_rec).predict(pd.concat([y1\_test, x1\_test, x4\_test, x6\_test], axis=1), y2\_test)

Коэффициент детерминации (R^2): 92.7%

Cредняя ошибка аппроксимации (MAE): 5.4%

Cредняя квадратичная ошибка (MSE): 0.5%

Третье уравнение:

CustomLinearRegression(y3\_train, final\_fn3\_rec).predict(pd.concat([y2\_test, x11\_test], axis=1), y3\_test)

Коэффициент детерминации (R^2): 94.1%

Cредняя ошибка аппроксимации (MAE): 5.3%

Cредняя квадратичная ошибка (MSE): 0.4%

* + - 1. **Система одновременных уравнений**

Система независимых уравнений с полученными значениями параметров представлена на рисунке 45:

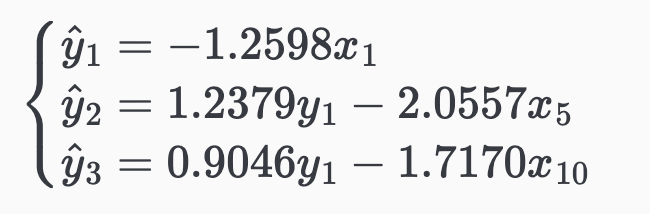


Рисунок 45 - Система независимых уравнений с полученными значениями параметров

print(f'Параметры первого: {final\_fn1\_odn.columns.values.tolist()}\n'

      f'Параметры второго: {final\_fn2\_odn.columns.values.tolist()}\n'

      f'Параметры третьего: {final\_fn3\_odn.columns.values.tolist()}')

Параметры первого: ['x1']

Параметры второго: ['y1', 'x5']

Параметры третьего: ['y1', 'x10']

Первое уравнение:

CustomLinearRegression(y1\_train, final\_fn1\_odn).predict(pd.concat([x1\_test], axis=1), y1\_test)

Коэффициент детерминации (R^2): 69.7%

Cредняя ошибка аппроксимации (MAE): 10.0%

Cредняя квадратичная ошибка (MSE): 1.6%

Второе уравнение:

CustomLinearRegression(y2\_train, final\_fn2\_odn).predict(pd.concat([y1\_test, x5\_test], axis=1), y2\_test)

Коэффициент детерминации (R^2): 85.0%

Cредняя ошибка аппроксимации (MAE): 7.9%

Cредняя квадратичная ошибка (MSE): 0.9%

Третье уравнение:

CustomLinearRegression(y3\_train, final\_fn3\_odn).predict(pd.concat([y1\_test, x10\_test], axis=1), y3\_test)

Коэффициент детерминации (R^2): 88.7%

Cредняя ошибка аппроксимации (MAE): 6.9%

Cредняя квадратичная ошибка (MSE): 0.8%

* + 1. **Прогнозирование с помощью методов машинного обучения**

Дополнительно было проведено прогнозирование с помощью методов машинного обучения.

В качестве методов машинного обучения были выбраны:

* Логистическая регрессия
* Метод опорных векторов
* Метод k-ближайших соседей
* Метод случайного леса
* Метод градиентного бустинга
* Метод нейронных сетей

Код прогнозирования с помощью методов машинного обучения на python представлен в Приложении Б.

Прогнозирование для y1 представлено на рисунке 46:

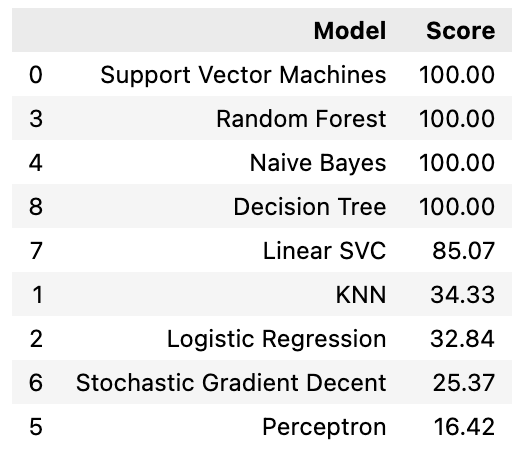


Рисунок 46 - Прогнозирование для y1

Прогнозирование для y2 представлено на рисунке 47:

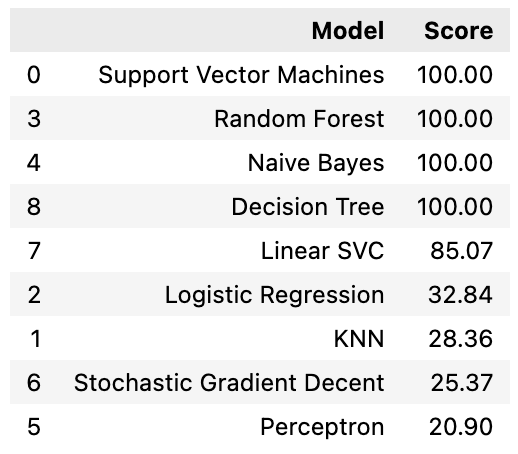


Рисунок 47- Прогнозирование для y2

Прогнозирование для y3 представлено на рисунке 48:

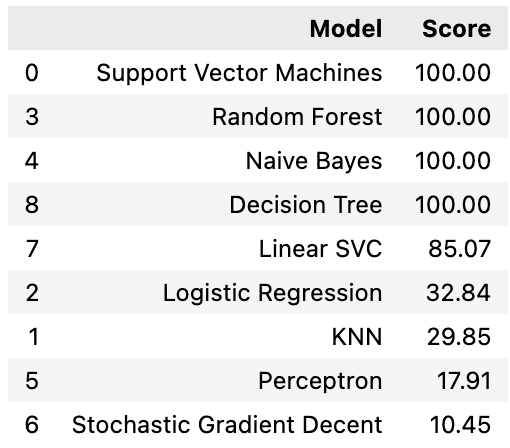


Рисунок 48 - Прогнозирование для y3

На рисунке 49 представлена важность переменных для Random Forest Classifier.

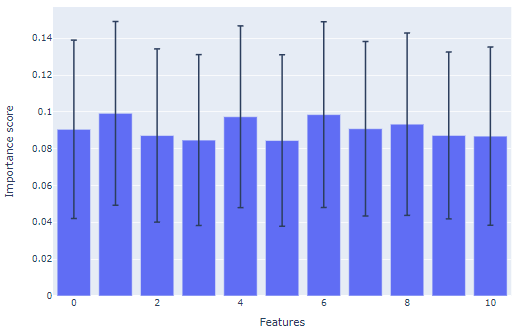


Рисунок 49 - Важность переменных для Random Forest Classifier

* 1. **Excel**
     1. **Исходные данные**

Перед началом анализа и прогнозирования был проведен поиск исходных данных, которые были получены путем совмещения разрозненных данных из различных источников в единую структуру.

В результате проведенного поиска данные выглядят следующим образом (Рисунок 50):

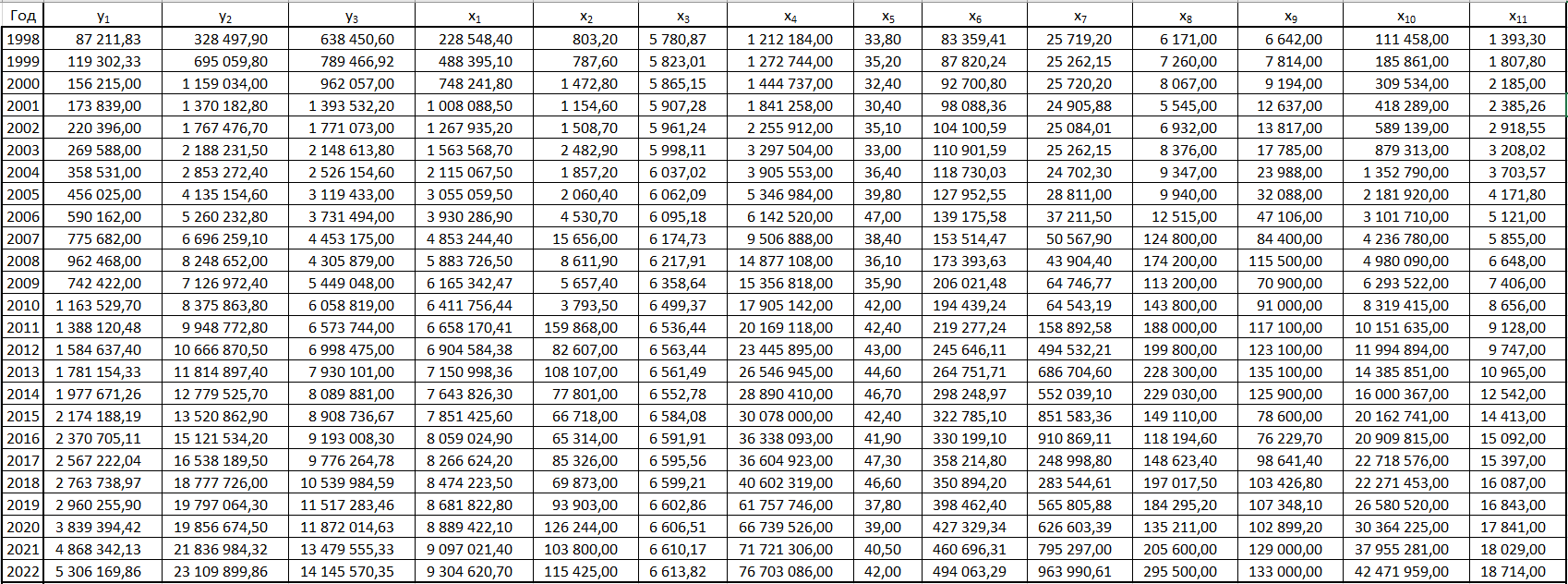


Рисунок 50 - Исходные данные

* + 1. **Системы уравнений**

Исходя из данных, были составлены 3 системы уравнений — система независимых уравнений, система рекурсивных уравнений и система одновременных уравнений (Рисунок 51).

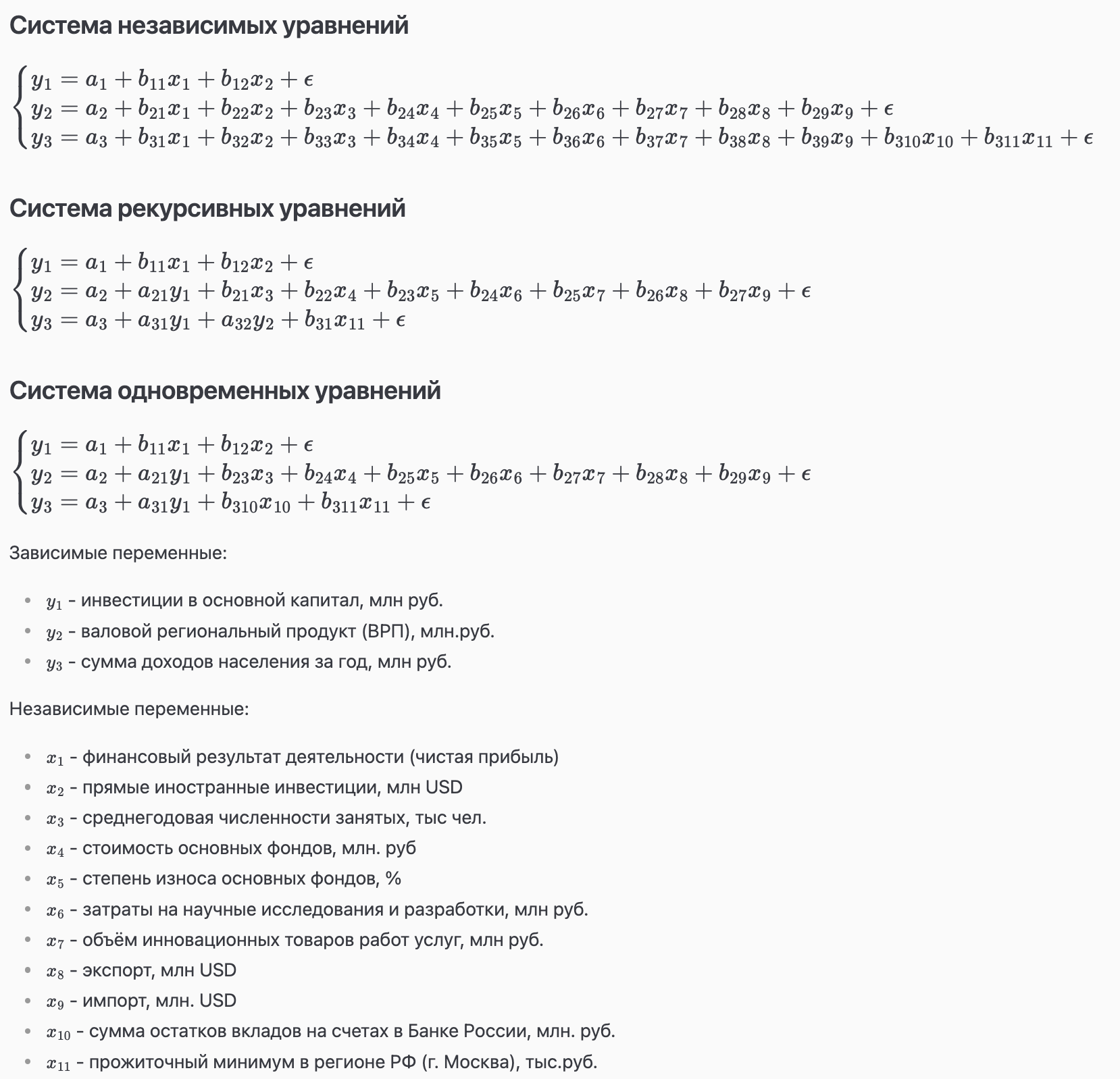


Рисунок 51 - Системы уравнений

Для применения МНК к первым двум системам, необходимо было исключить незначимые факторы и избавиться от мультиколлинеарности. С помощью регрессионного анализа (Рисунок 52) и построения матрицы корреляции (Рисунок 53) в Excel были отобраны факторы для каждого уравнения в двух системах:

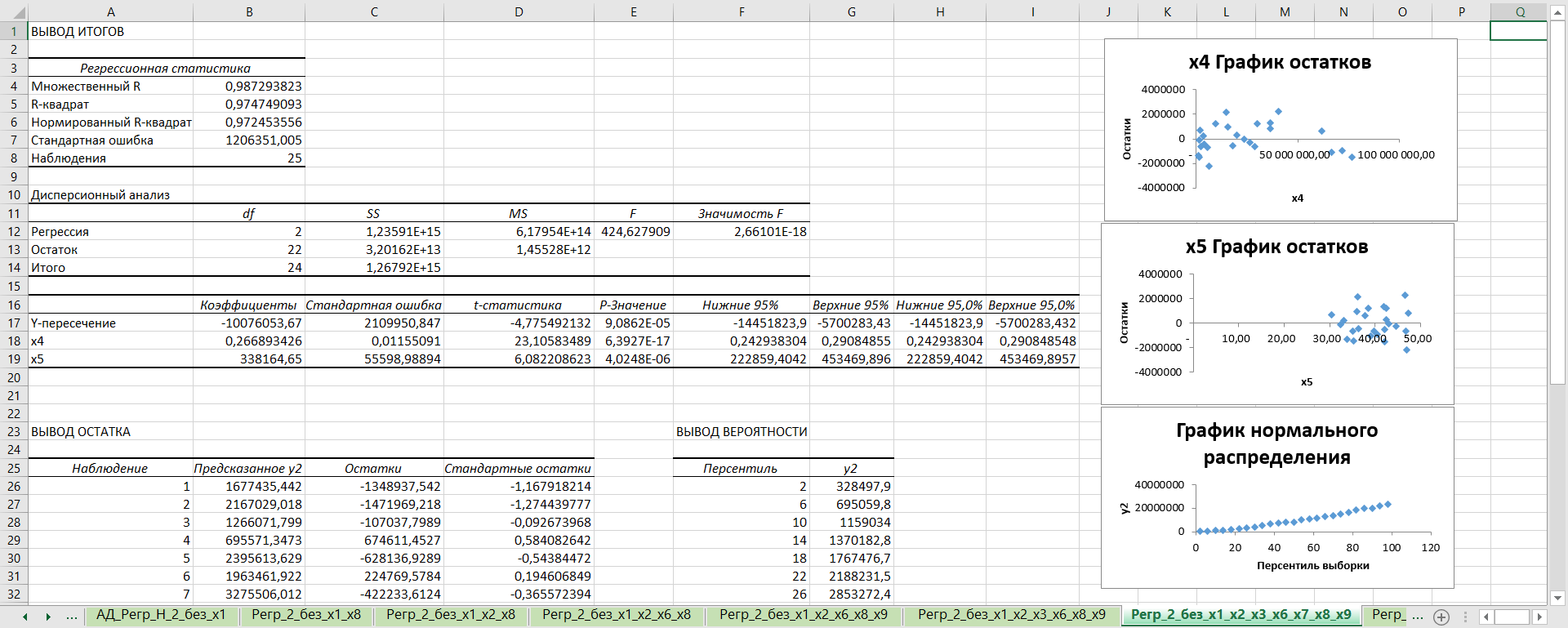


Рисунок 52 - Пример использования регрессионного анализа

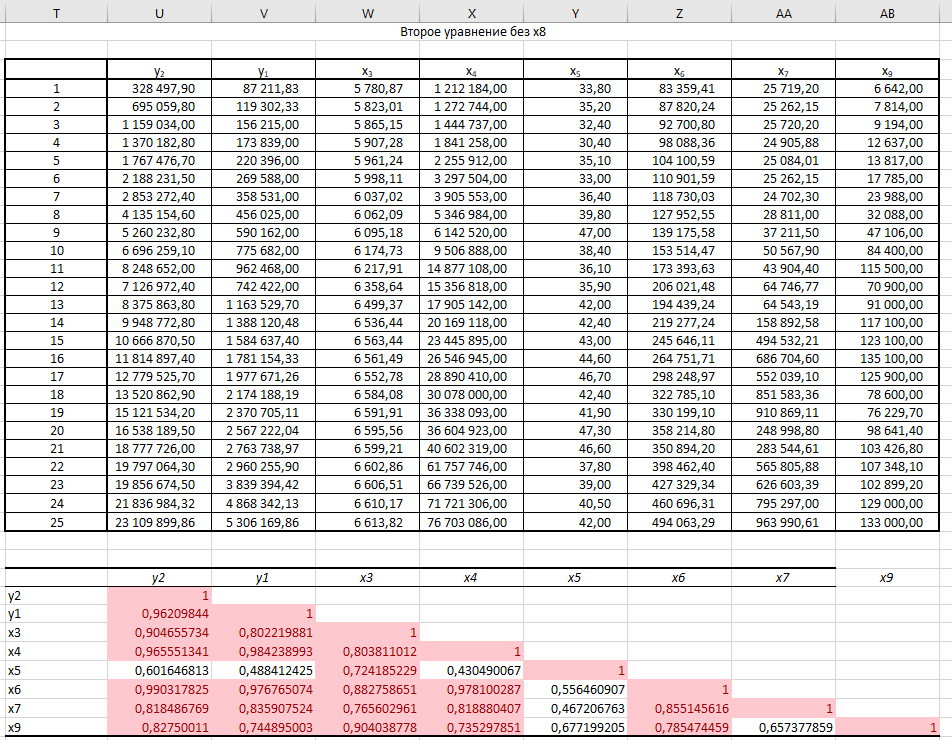


Рисунок 53 - Пример использования Корреляции

В результате отбора факторов и применения к ним МНК системы независимых (Приложение В) и рекурсивных уравнений (Приложение Г) имеют следующий вид:

Спрогнозированные значения и доверительные интервалы для 2023 года (система независимых уравнений) (Приложение Д):

Спрогнозированные значения и доверительные интервалы для 2023 года (система рекурсивных уравнений) (Приложение Е):

Для системы одновременных уравнений была проведена идентификация. Проверка на идентификацию приведена в Приложении Ж. Так как все уравнения системы сверхидентифицируемы, то и сама система сверхидентифицируема. Для поиска коэффициентов был применен ДМНК (Приложение З).

В результате применения ДМНК были найдены коэффициенты; система выглядит следующим образом:

Спрогнозированные значения и доверительные интервалы для 2023 года (система одновременных уравнений) (Приложение И):

**ЗАКЛЮЧЕНИЕ**

В заключении можно отметить, что исследование параметров систем уравнений для прогнозирования и анализа зависимости валового регионального продукта, объёма инвестиций в основной капитал за год и дохода населения города Москвы является важной задачей для оценки эффективности государственного управления в регионе.

Анализ динамики ВРП и других показателей экономического развития поможет выявить тенденции и прогнозировать будущее развитие региона. Поэтому, использование ВРП на федеральном и региональном уровнях является актуальной и востребованной темой для исследования.

В ходе выполнения работы нами были решены следующие задач:

* Поиск исходных данных для анализа.
* Построение эконометрических моделей (системы с раздельными уравнениями, рекурсивными уравнениями, одновременными уравнениями).
* Проверка системы на необходимое условие идентификации и достаточное условие идентификации.
* Выбор метода оценивания систем: МНК, КМНК, ДМНК.
* Исключение незначимых факторов.
* Рассчёт параметров регрессии.
* Прогноз данных на 2023 год.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Единое хранилище данных [Электронный ресурс]. URL: https://ehd.moscow/index.php? (дата обращения: 18.02.2023)
2. Управление Федеральной службы государственной статистики по г. Москве и Московской области [Электронный ресурс]. URL: https://77.rosstat.gov.ru/ (дата обращения: 18.02.2023)
3. Учебное пособие по эконометрике [Электронный ресурс]. URL: http://producm.ru/upload/books5/01.pdf (дата обращения: 25.02.2023)
4. Регрессионные модели в Python [Электронный ресурс]. URL: https://nagornyy.me/it/regressionnye-modeli-v-python/ (дата обращения: 02.03.2023)
5. Основные виды эконометрических моделей [Электронный ресурс]. URL: https://911zp.at.ua/\_ld/2/222\_-\_.pdf (дата обращения: 07.03.2023)
6. Типы моделей в эконометрике [Электронный ресурс]. URL: https://studfile.net/preview/9810964/page:5/ (дата обращения: 20.03.2023)
7. Идентификация моделей [Электронный ресурс]. URL: https://studopedia.ru/9\_148175\_identifikatsiya-modeley.html (дата обращения: 05.04.2023)
8. Двухшаговый метод наименьших квадратов [Электронный ресурс]. URL:https://bstudy.net/630258/ekonomika/dvuhshagovyy\_metod\_naimenshih\_kvadratov\_dmnk (дата обращения: 14.04.2023)
9. Двухшаговый метод наименьших квадратов для решения систем эконометрических уравнений [Электронный ресурс]. URL: <https://www.bibliofond.ru/view.aspx?id=883106> (дата обращения: 17.04.2023)
10. Эконометрическое прогнозирование [Электронный ресурс]. URL: https://habr.com/ru/articles/583388/ (дата обращения: 21.04.2023)

# ПРИЛОЖЕНИЕ А

class CustomLinearRegression:

    def \_\_init\_\_(self, y, x, model\_type=OLS):

        self.model\_type = model\_type

        self.y = y

        self.x = x

        self.model = self.model\_type(self.y, self.x).fit()

    def get\_coefficients(self):

        print(self.model.params)

        answer = []

        names = self.x.columns.values.tolist()

        for i, par in enumerate(self.model.params):

            answer.append(f'+ {par:.4f}{names[i][0]}\_{{{names[i][1:]}}}')

        print(f'$$\hat {self.y.name[0]}\_{self.y.name[1]} = {" ".join(answer)[2:]}$$')

    def summary(self):

        print(self.model.summary())

    def pre(self, X\_test):

        return self.model.predict(X\_test)

    def predict(self, X\_test, y\_test):

        r2 = r2\_score(y\_test, self.model.predict(X\_test))

        mae = mean\_absolute\_error(y\_test, self.model.predict(X\_test))

        mse = mean\_squared\_error(y\_test, self.model.predict(X\_test))

        print(

            f'Коэффициент детерминации (R^2): {r2 \* 100:.1f}%\nCредняя ошибка аппроксимации (MAE): {mae \* 100:.1f}%\nCредняя квадратичная ошибка (MSE): {mse \* 100:.1f}%')

    def F(self):

        return self.model.fvalue

    def current\_p(self, index):

        return self.model.f\_pvalue[index]

    def forward\_selection(self):

        xi = []

        stop = False

        x\_new = []  # список переменных, которые будут включены в модель

        x\_len = len(self.x.columns) - 1  # количество столбцов в датафрейме

        x\_test = []

        for n in range(x\_len):

            F\_max = 0  # максимальное значение F

            p\_max = 0  # максимальное значение p

            F\_max\_i = 0

            F\_max\_x = ''

            if not stop:

                for i in range(x\_len):

                    if i not in xi:

                        x\_test = x\_new.copy()

                        x\_test.append(self.x.columns[i])

                        F = CustomLinearRegression(self.y, self.x.loc[:, x\_test]).F()

                        print(f"'x{i + 1}',", F)

                        if F > F\_max:

                            F\_max = F

                            F\_max\_i = i

                            F\_max\_x = f"x{i + 1}"

                x\_new.append(F\_max\_x)

                xi.append(F\_max\_i)

                print(x\_new)

                for j in range(len(x\_new)):

                    p = CustomLinearRegression(self.y, self.x.loc[:, x\_new]).current\_p(j)

                    if p > 0.05:

                        stop = True

                        print(f'P-value: {p}')

                        break

        CustomLinearRegression(self.y, self.x.loc[:, x\_new[:-1]]).summary()

    def backward\_elimination(self, p=0.05, start=0):

        regression = self.model\_type(self.y, self.x).fit()

        futures\_num = len(self.x.columns)

        for i in range(futures\_num):

            regression = self.model\_type(self.y, self.x).fit()

            max\_p = max(regression.pvalues.values[start:])

            if max\_p > p:

                for j in range(0, futures\_num - i):

                    if regression.pvalues[j] == max\_p:

                        self.x = self.x.drop(self.x.columns[[j]], axis=1)

        print(regression.summary())

    def custom\_feature\_selection(self, x, y, model=LinearRegression(), selection\_method=RFE):

        result\_list = list()

        result = selection\_method(model).fit(x, y)

        for i, feature in enumerate(result.get\_support()):

            if feature:

                result\_list.append(f'x{i + 1}')

        CustomLinearRegression(y, x[result\_list]).summary()

    def correlation\_map(self):

        return px.imshow(self.x.corr(), text\_auto='.2f', color\_continuous\_scale="gnbu")

class CustomLinearRegressionWithConst(CustomLinearRegression):

    def \_\_init\_\_(self, y, x, model\_type=OLS):

        super().\_\_init\_\_(y, x, model\_type)

        self.x = sm.add\_constant(self.x)

        self.model = self.model\_type(self.y, self.x).fit()

    def backward\_elimination(self, p=0.05, start=1):

        regression = self.model\_type(self.y, self.x).fit()

        futures\_num = len(self.x.columns)

        for i in range(futures\_num):

            self.x = sm.add\_constant(self.x)

            regression = self.model\_type(self.y, self.x).fit()

            max\_p = max(regression.pvalues.values[start:])

            if max\_p > p:

                for j in range(0, futures\_num - i):

                    if regression.pvalues[j] == max\_p:

                        self.x = self.x.drop(self.x.columns[[j]], axis=1)

        print(regression.summary())

class CustomLinearRegressionRegularized(CustomLinearRegression):

    def \_\_init\_\_(self, y, x, model\_type=OLS, alpha=0.01, L1\_wt=0.05):

        super().\_\_init\_\_(y, x, model\_type)

        self.alpha = alpha

        self.L1\_wt = L1\_wt

        self.model = sm.OLS(self.y, self.x).fit\_regularized(alpha=self.alpha, L1\_wt=self.L1\_wt, refit=True)

    def backward\_elimination(self, p=0.05, start=0):

        regression = self.model\_type(self.y, self.x).fit\_regularized(alpha=self.alpha, L1\_wt=self.L1\_wt, refit=True)

        features\_num = len(self.x.columns)

        for i in range(features\_num):

            regression = self.model\_type(self.y, self.x).fit\_regularized(alpha=self.alpha, L1\_wt=self.L1\_wt, refit=True)

            max\_p = max(regression.pvalues[start:])

            if max\_p > p:

                for j in range(features\_num - i):

                    if regression.pvalues[j] == max\_p:

                        self.x = self.x.drop(self.x.columns[[j]], axis=1)

        print(regression.summary())

# class CustomRidgeLinearRegression(CustomLinearRegression):

#     def \_\_init\_\_(self, y, x, model\_type=OLS):

#         super().\_\_init\_\_(y, x, model\_type)

#         self.model = Ridge(alpha=0.1).fit(self.x, self.y)

# ПРИЛОЖЕНИЕ Б

class MLPrediction():

    def \_\_init\_\_(self, X\_train, X\_test, y\_train):

        self.X\_train = X\_train

        self.X\_test = X\_test

        self.y\_train = preprocessing.LabelEncoder().fit\_transform(y\_train)

    def logreg(self):  # Logistic Regression

        self.logreg = LogisticRegression()

        self.logreg.fit(self.X\_train, self.y\_train)

        Y\_pred = self.logreg.predict(self.X\_test)

        self.acc\_log = round(self.logreg.score(self.X\_train, self.y\_train) \* 100, 2)

    def logreg\_score(self):

        self.logreg()

        print(self.acc\_log)

        print(self.logreg.coef\_)

    def svc(self):  # Support Vector Machines

        self.svc = SVC(kernel='linear')

        self.svc.fit(self.X\_train, self.y\_train)

        Y\_pred = self.svc.predict(self.X\_test)

        self.acc\_svc = round(self.svc.score(self.X\_train, self.y\_train) \* 100, 2)

    def svc\_score(self):

        self.svc()

        print(self.acc\_svc)

        print(self.svc.coef\_)

    def knn(self):

        self.knn = KNeighborsClassifier(n\_neighbors=3)  # 3

        self.knn.fit(self.X\_train, self.y\_train)

        Y\_pred = self.knn.predict(self.X\_test)

        self.acc\_knn = round(self.knn.score(self.X\_train, self.y\_train) \* 100, 2)

    def knn\_score(self):

        self.knn()

        print(self.acc\_knn)

    def gaussian(self):  # Gaussian Naive Bayes

        self.gaussian = GaussianNB()

        self.gaussian.fit(self.X\_train, self.y\_train)

        Y\_pred = self.gaussian.predict(self.X\_test)

        self.acc\_gaussian = round(self.gaussian.score(self.X\_train, self.y\_train) \* 100, 2)

    def gaussian\_score(self):

        self.gaussian()

        print(self.acc\_gaussian)

    def perceptron(self):

        self.perceptron = Perceptron()

        self.perceptron.fit(self.X\_train, self.y\_train)

        Y\_pred = self.perceptron.predict(self.X\_test)

        self.acc\_perceptron = round(self.perceptron.score(self.X\_train, self.y\_train) \* 100, 2)

    def perceptron\_score(self):

        self.perceptron()

        print(self.acc\_perceptron)

    def linear\_svc(self):  # Linear SVC

        self.linear\_svc = LinearSVC()

        self.linear\_svc.fit(self.X\_train, self.y\_train)

        Y\_pred = self.linear\_svc.predict(self.X\_test)

        self.acc\_linear\_svc = round(self.linear\_svc.score(self.X\_train, self.y\_train) \* 100, 2)

    def linear\_svc\_score(self):

        self.linear\_svc()

        print(self.acc\_linear\_svc)

    def sgd(self):  # Stochastic Gradient Descent

        self.sgd = SGDClassifier()

        self.sgd.fit(self.X\_train, self.y\_train)

        Y\_pred = self.sgd.predict(self.X\_test)

        self.acc\_sgd = round(self.sgd.score(self.X\_train, self.y\_train) \* 100, 2)

    def sgd\_score(self):

        self.sgd()

        print(self.acc\_sgd)

    def decision\_tree(self):  # Decision Tree

        self.decision\_tree = DecisionTreeClassifier()

        self.decision\_tree.fit(self.X\_train, self.y\_train)

        Y\_pred = self.decision\_tree.predict(self.X\_test)

        self.acc\_decision\_tree = round(self.decision\_tree.score(self.X\_train, self.y\_train) \* 100, 2)

    def decision\_tree\_score(self):

        self.decision\_tree()

        print(self.acc\_decision\_tree)

    def random\_forest(self):  # Random Forest

        self.random\_forest = RandomForestClassifier(n\_estimators=1000, max\_depth=100, min\_samples\_leaf=1)

        self.random\_forest.fit(self.X\_train, self.y\_train)

        self.Y\_pred = self.random\_forest.predict(self.X\_test)

        self.random\_forest.score(self.X\_train, self.y\_train)

        self.acc\_random\_forest = self.random\_forest.score(self.X\_train, self.y\_train) \* 100

    def random\_forest\_score(self):

        self.random\_forest()

        print(self.acc\_random\_forest)

    def rfc\_plot(self):

        self.random\_forest()

        importances = self.random\_forest.feature\_importances\_

        std = np.std([tree.feature\_importances\_ for tree in self.random\_forest.estimators\_], axis=0)

        indices = np.argsort(importances)[::-1]

        df = pd.DataFrame({

            'Features': indices,

            'Importance': importances[indices],

            'Importance std': std[indices]

        })

        fig = px.bar(df, x='Features', y='Importance', error\_y='Importance std',

                     labels={'Importance': 'Importance score'})

        fig.update\_layout(title='Важность переменных для Random Forest Classifier')

        return fig.show()

    def compare(self):

        self.logreg()

        self.svc()

        self.knn()

        self.gaussian()

        self.perceptron()

        self.linear\_svc()

        self.sgd()

        self.decision\_tree()

        self.random\_forest()

        models = pd.DataFrame({

            'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',

                      'Random Forest', 'Naive Bayes', 'Perceptron',

                      'Stochastic Gradient Decent', 'Linear SVC',

                      'Decision Tree'],

            'Score': [self.acc\_svc, self.acc\_knn, self.acc\_log,

                      self.acc\_random\_forest, self.acc\_gaussian, self.acc\_perceptron,

                      self.acc\_sgd, self.acc\_linear\_svc, self.acc\_decision\_tree]})

        return models.sort\_values(by='Score', ascending=False)

# ПРИЛОЖЕНИЕ В

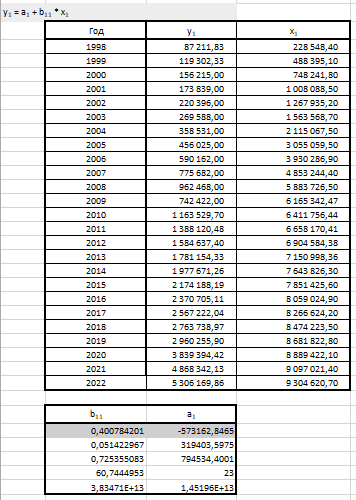


Рисунок 54 - МНК к 1-ому ур-ию незав.

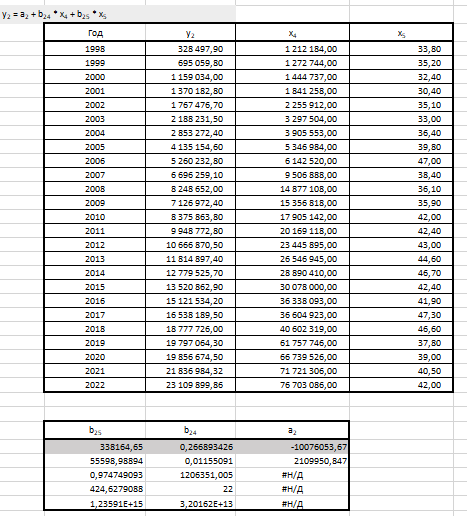


Рисунок 55 - МНК ко 2-ому ур-ию незав.

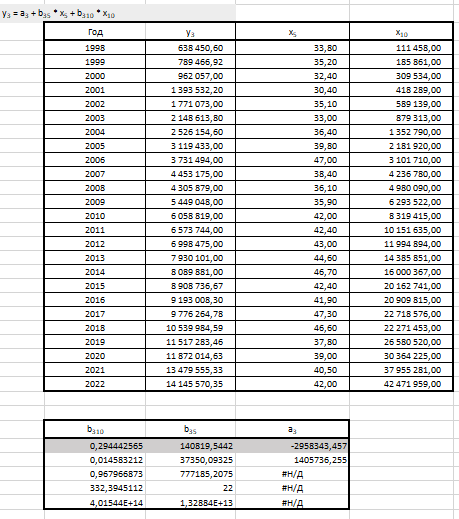


Рисунок 56 - МНК к 3-ему ур-ию незав.

**ПРИЛОЖЕНИЕ Г**

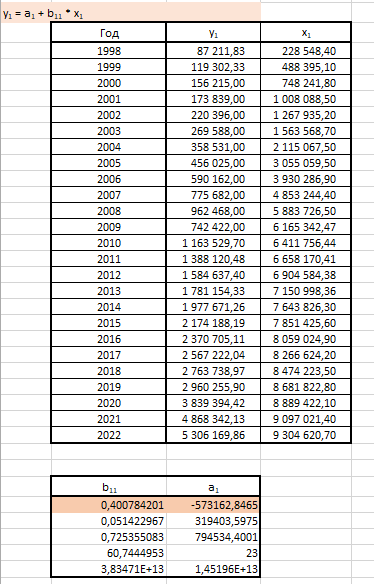


Рисунок 57 - МНК к 1-ому ур-ию рекурс..

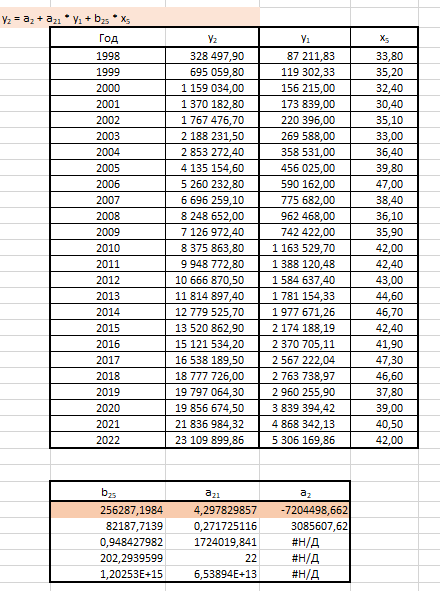


Рисунок 58 - МНК ко 2-ому ур-ию рекурс.

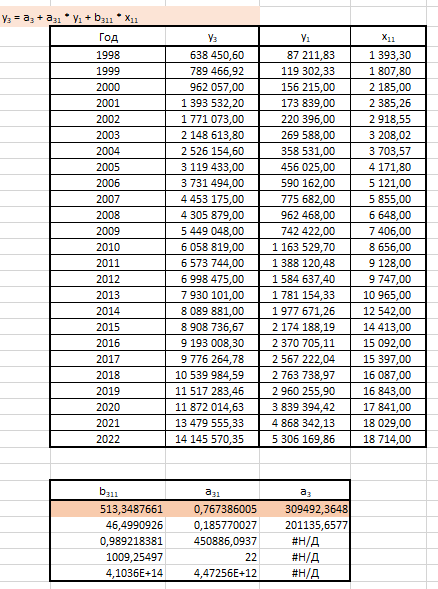


Рисунок 59 - МНК к 3-ему ур-ию рекурс..

**ПРИЛОЖЕНИЕ Д**

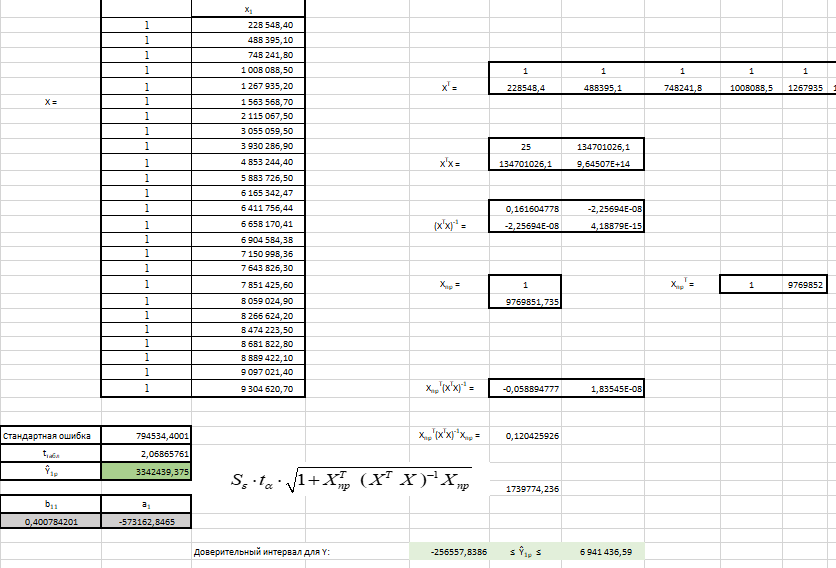


Рисунок 60 - Прогноз к 1-ому ур-ию незав.

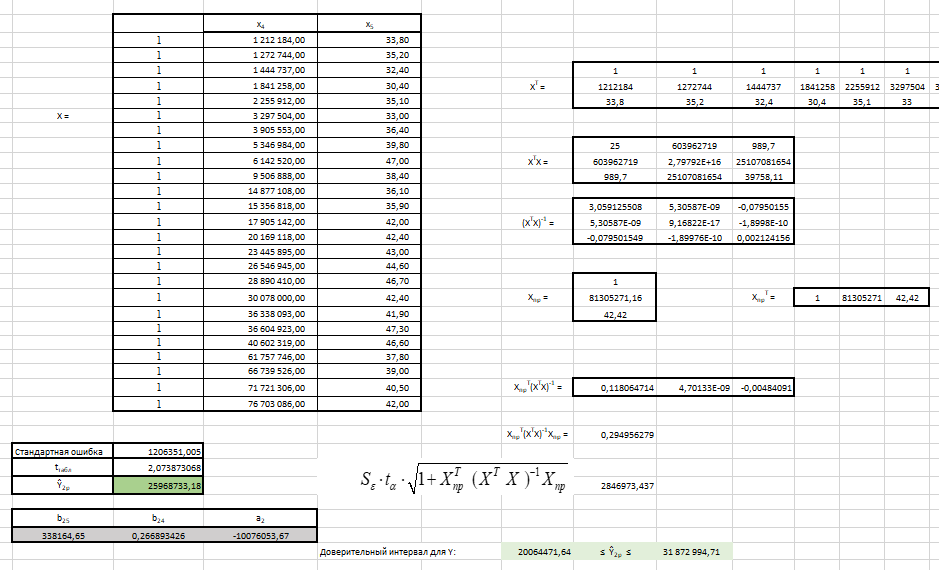


Рисунок 61 - Прогноз ко 2-ому ур-ию незав.

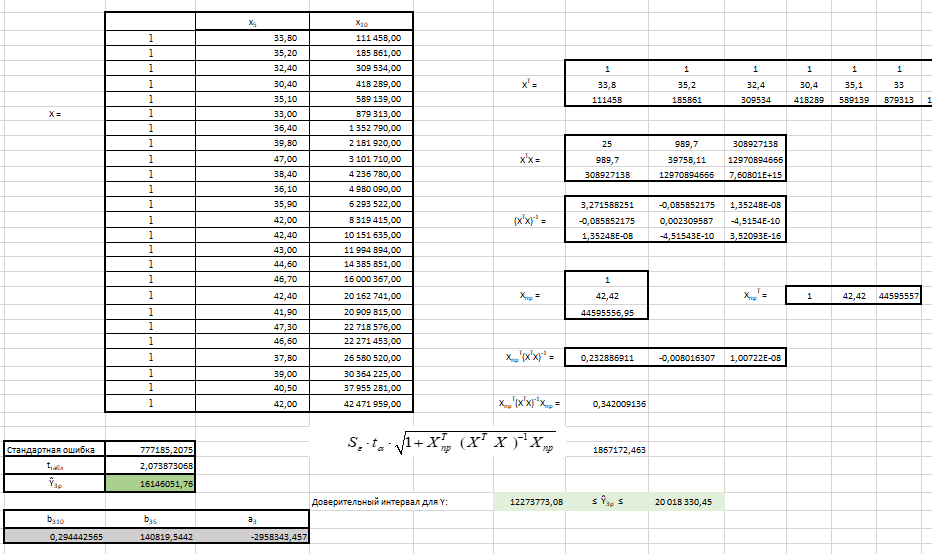


Рисунок 62 - Прогноз к 3-ему ур-ию незав.

**ПРИЛОЖЕНИЕ Е**

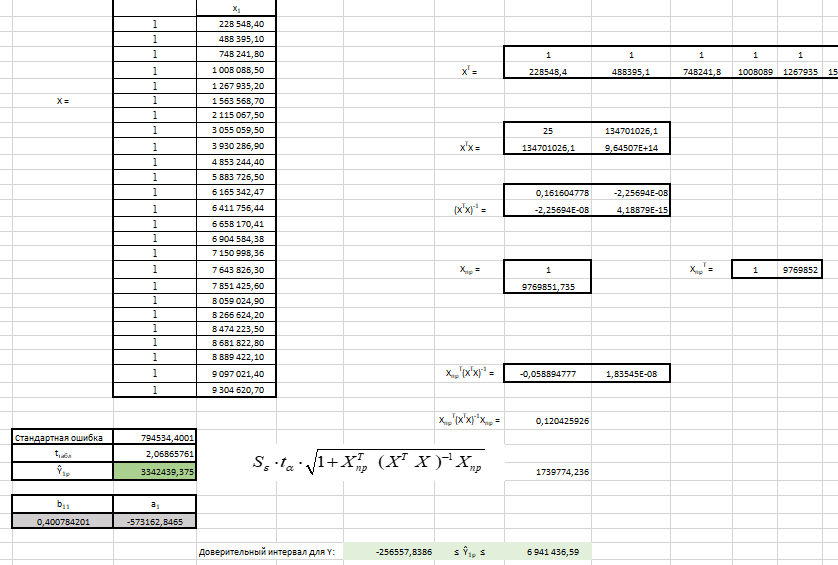


Рисунок 63 - Прогноз к 1-ому ур-ию рекурс.

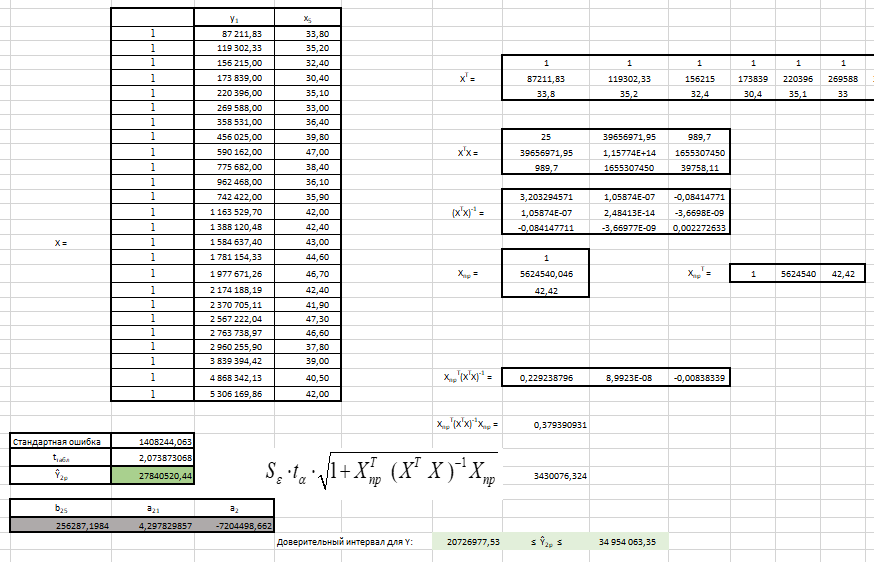


Рисунок 64 - Прогноз ко 2-ому ур-ию рекурс.

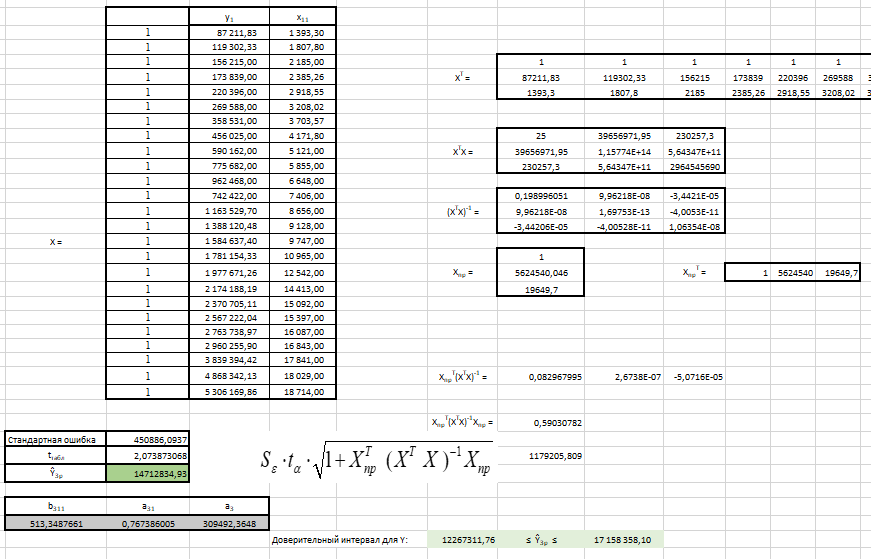


Рисунок 65 - Прогноз к 3-ему ур-ию рекурс.

**ПРИЛОЖЕНИЕ Ж**

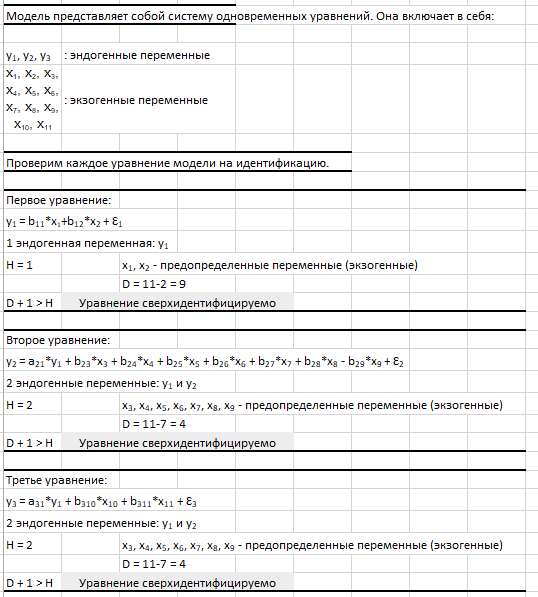


Рисунок 66 - Проверка необходимого условия идентификации

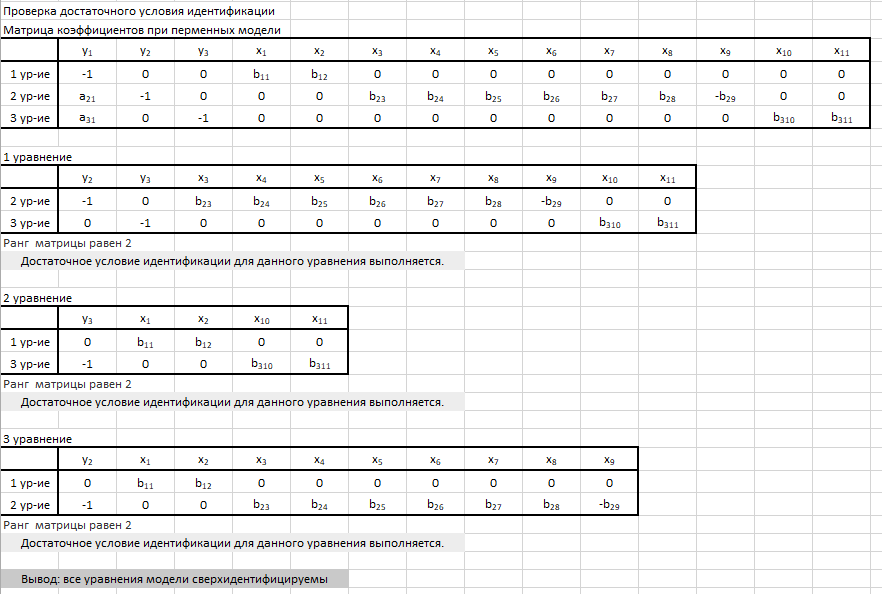


Рисунок 67 - Проверка достаточного условия идентификации

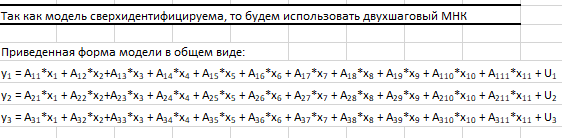


Рисунок 68 - Приведенная форма модели

**ПРИЛОЖЕНИЕ З**

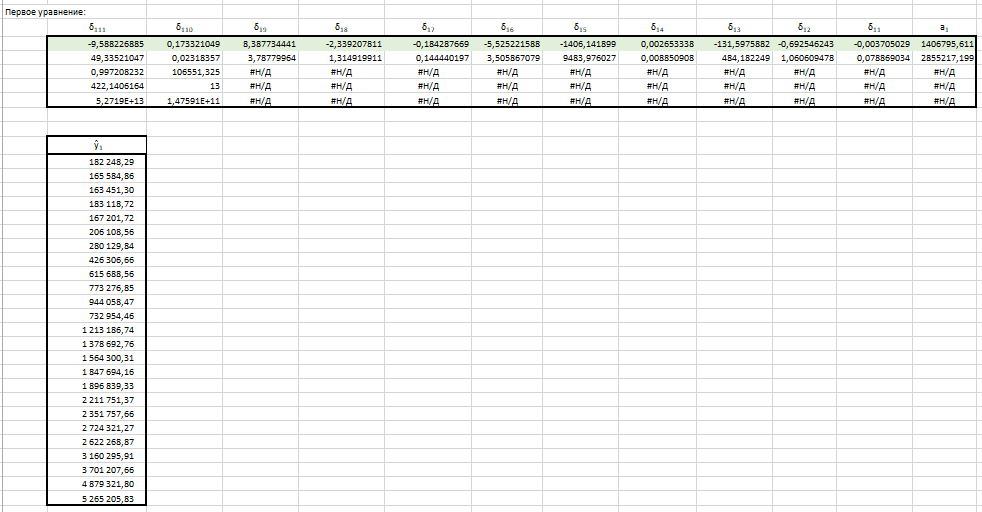


Рисунок 69 - Первый шаг ДМНК для 1-ого уравнения

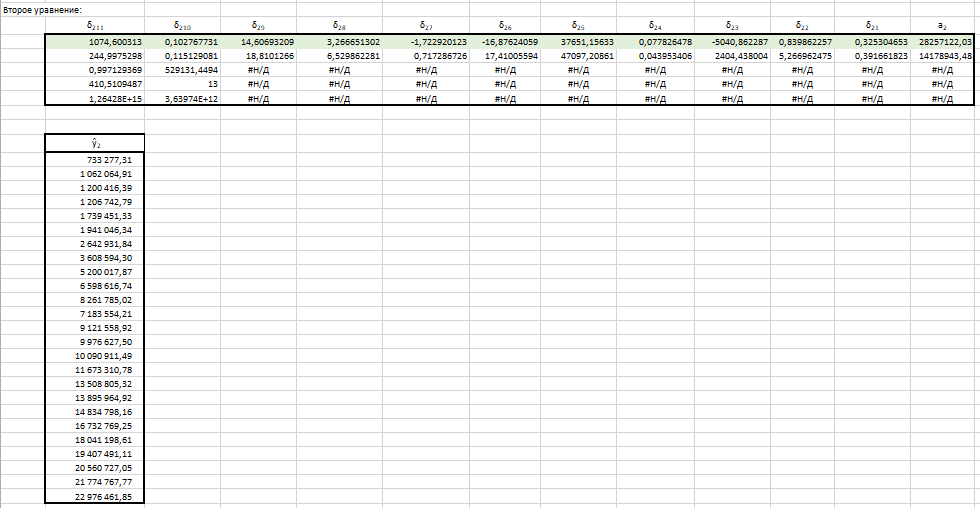


Рисунок 70 - Первый шаг ДМНК для 2-ого уравнения

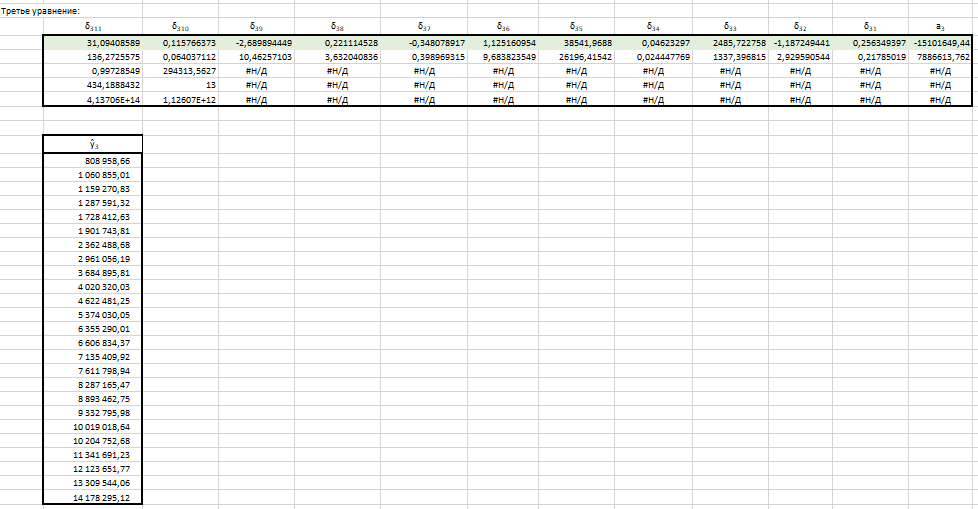


Рисунок 71 - Первый шаг ДМНК для 3-его уравнения

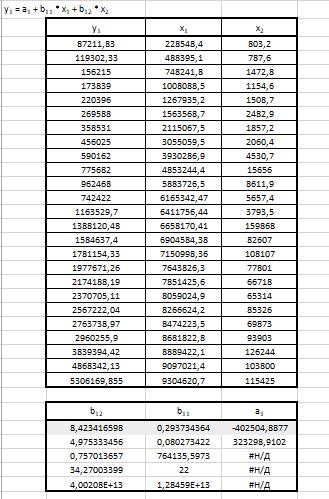


Рисунок 72 – Второй шаг ДМНК для 1-ого уравнения

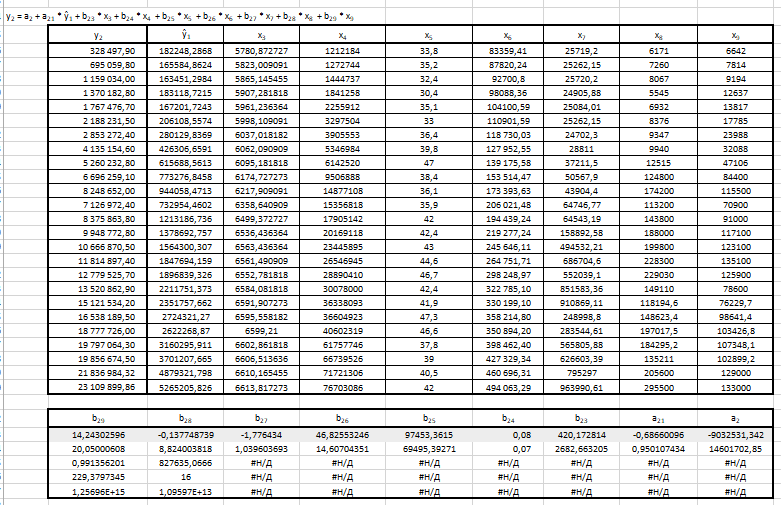


Рисунок 73 – Второй шаг ДМНК для 2-ого уравнения

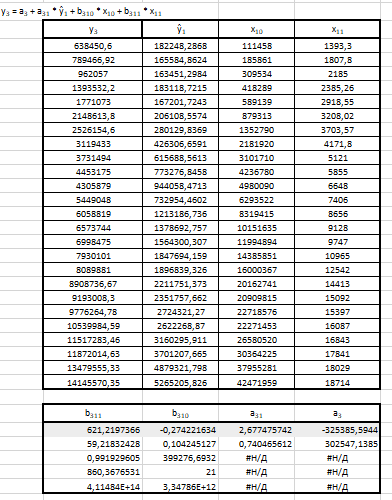


Рисунок 74 - Второй шаг ДМНК для 3-его уравнения

**ПРИЛОЖЕНИЕ И**

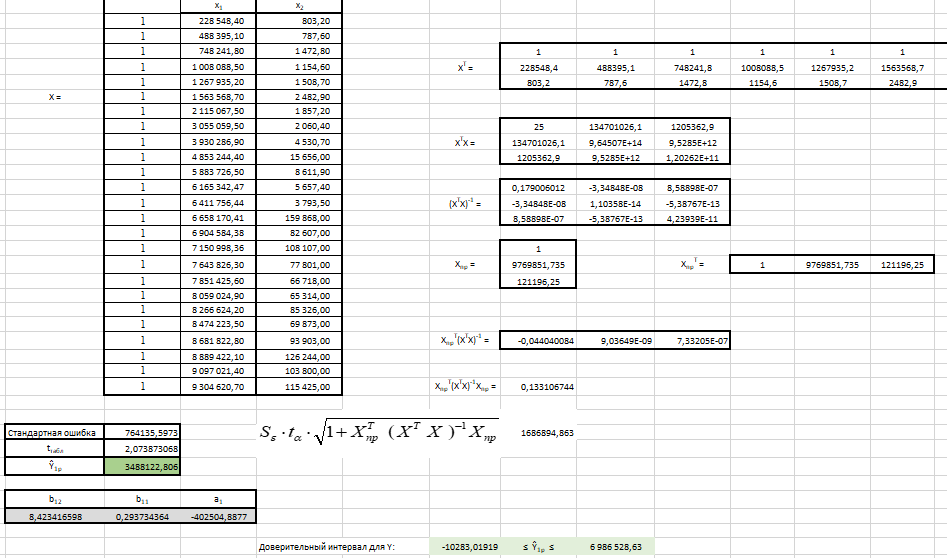


Рисунок 75 - Прогноз для 1-ого ур-ия одновр.

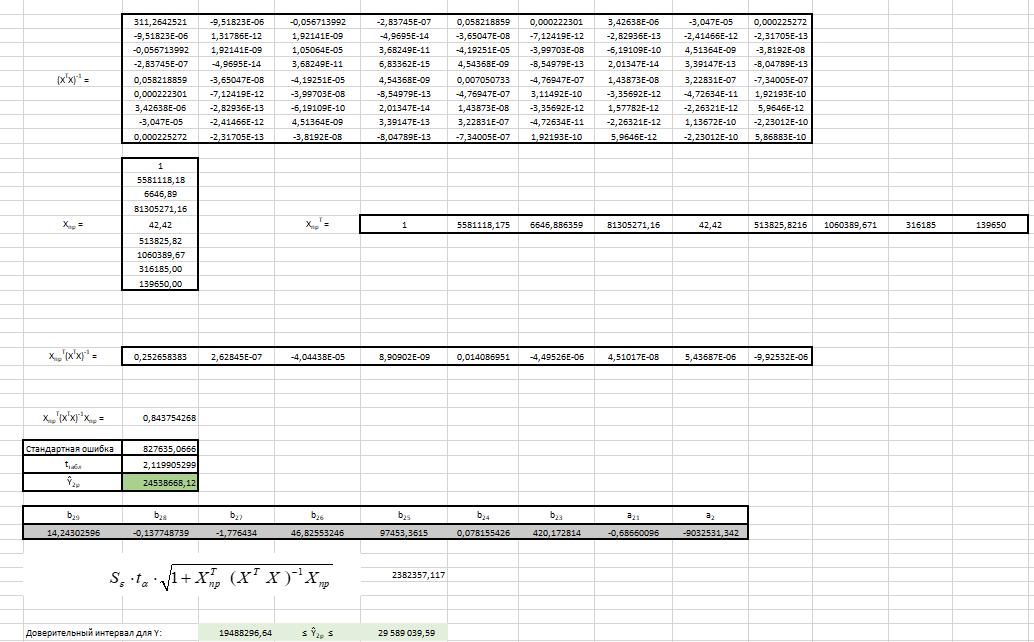


Рисунок 76 - Прогноз для 2-ого ур-ия одновр.

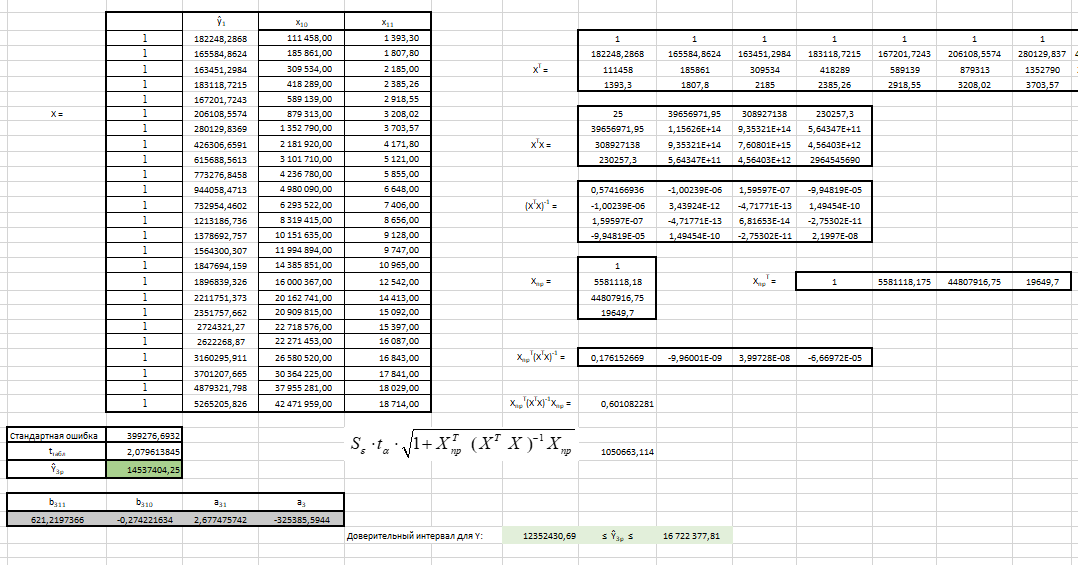


Рисунок 77 - Прогноз для 3-его ур-ия одновр.