Министерство науки и высшего образования РФ
ФГАОУ ВПО
Национальный исследовательский технологический университет «МИСиС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

КУРСОВАЯ РАБОТА

по дисциплине «Прикладной статистический анализ»

на тему «Разработка модели прогнозирования количества кредитных заявок»

Исполнители:

Раздел	Фамилия И.О.	Группа
Курсовая работа	Витязев Е.Д.	БИВТ-21-1

Проверил:
К.т.н., доцент каф. ИКТ

Маркарян А.О.

МОСКВА, 2023

Введение

В современном быстро меняющемся мире, где экономика и финансы играют ключевую роль в жизни общества, прогнозирование количества заявок на жилищные кредиты становится все более актуальным. Это исследование направлено на разработку модели, которая поможет в прогнозировании количества заявок на жилищные кредиты, основываясь на данных, предоставленных Центральным Банком Российской Федерации. Это исследование не только поможет улучшить понимание текущего состояния рынка жилищных кредитов, но и предоставит инструменты для прогнозирования будущих тенденций.

Актуальность исследования заключается в том, что точное прогнозирование количества заявок на жилищные кредиты может помочь банкам и другим финансовым учреждениям в планировании своих операций, управлении рисками и оптимизации процессов. Это, в свою очередь, может привести к улучшению финансовой стабильности и экономического роста.

Общая цель данного исследования - разработать надежную и эффективную модель прогнозирования количества заявок на жилищные кредиты. Для достижения этой цели, я поставил перед собой следующие задачи:

- Изучить и анализировать имеющиеся данные.
- Разработать и обучить модель прогнозирования.
- Провести тестирование и валидацию модели.
- Проанализировать и интерпретировать полученные результаты.

Объект исследования - это заявки на жилищные кредиты, поданные физическими лицами-резидентами. Предмет исследования - это процесс прогнозирования количества этих заявок на основе имеющихся данных.

В ходе исследования я надеюсь разработать модель, которая будет способна точно прогнозировать количество заявок на жилищные кредиты, что поможет улучшить финансовое планирование и управление рисками.

1. АНАЛИЗ ХАРАКТЕРИСТИК ОБЪЕКТА ИССЛЕДОВАНИЯ

1.1 Описание объекта исследования

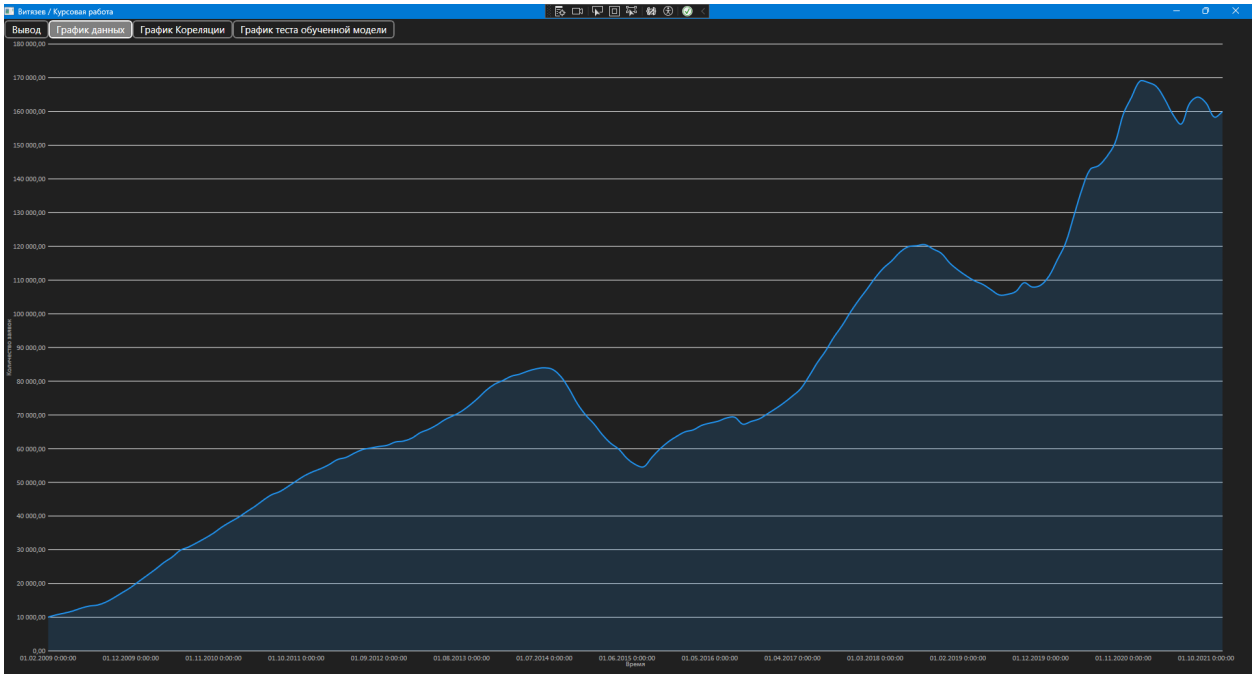
Объектом исследования в данной работе являются заявки на жилищные кредиты, представленные в виде временного ряда за период с февраля 2009 года по декабрь 2021 года, за исключением первого месяца каждого года. Данные представляют собой ежемесячное количество заявок на жилищные кредиты, поданных физическими лицами-резидентами.

1.2 Анализ объекта исследования с помощью статистических показателей

В данном разделе я провёл анализ основных статистических характеристик объекта исследования - количества заявок на жилищные кредиты, поданных физическими лицами-резидентами.

1. **Среднее значение** составляет 79857,67. Это означает, что в среднем в месяц поступает около 79857,67 заявки на жилищный кредит.
2. **Медиана равна** 69947,91. Это означает, что половина всех месячных значений лежит ниже этого числа, а другая половина - выше. Так как медиана близка к среднему значению, мы можем сказать, что распределение данных относительно симметрично.
3. **Стандартное отклонение** равно 41339,07. Это показывает, насколько велик разброс данных относительно среднего значения. Чем больше стандартное отклонение, тем больше разброс данных.
4. **Дисперсия равна** 1708918652,67. Это показывает разброс значений относительно среднего значения. Дисперсия, как и стандартное отклонение, является мерой разброса данных. Такая большая дисперсия вполне ожидаема для наших данных, учитывая, что они представляют собой количество заявок на жилищные кредиты, которые могут достигать значений более 168994,73. Большая дисперсия указывает на то, что значения заявок на жилищные кредиты могут значительно варьироваться от месяца к месяцу.
5. **Минимум и максимум** равны 10027,17 и 168994,73 соответственно. Это наименьшее и наибольшее количество заявок на жилищный кредит в месяц.
6. **Скос** равен 0,45. Это показывает, что распределение данных немного скошено вправо, то есть у распределения есть небольшой “хвост” справа.
7. **Эксцесс** равен 2,54. Это показывает, что распределение данных имеет более “тяжелые” хвосты и менее острый пик, чем у нормального распределения.

1.3 Выявление причинно-следственных связей



«Заявки на кредит по месяцам в виде графика»

Из графика мы видим стабильный рост количества жилищных кредитов в России с 1 февраля 2009 года по 1 июля 2014 года. Давайте рассмотрим причины этого роста:

- **Экономический рост:** В период с 2009 по 2014 год Россия переживала некоторое экономическое восстановление после мирового финансового кризиса 2008 года. Улучшение экономической ситуации могло способствовать увеличению спроса на жилье и, следовательно, на жилищные кредиты.
- **Государственная поддержка:** Власти могли предоставлять стимулы для развития рынка недвижимости, такие как субсидии на ипотечные кредиты, снижение процентных ставок или упрощение процедуры получения кредита. Это могло привести к росту числа жилищных кредитов.
- **Демографические факторы**: Рост населения, увеличение числа семей и стремление молодых людей к собственному жилью также могли способствовать увеличению спроса на жилищные кредиты.
- **Стабильность банковской системы:** Если банковская система была стабильной и надежной, это могло увеличить доверие кредиторов и заинтересовать больше людей в получении жилищных кредитов.
- **Инфляция и ставки:** Инфляция и уровень процентных ставок также могли повлиять на решение о взятии кредита. Низкие процентные ставки делают кредиты более доступными.

Далее на графике мы видим небольшой спад количества жилищных кредитов в августе 2015 года, когда было всего 55 614 заявок на кредит, в то время как всего пару месяцев назад их было 86-90 тысяч. Причины у этого могут быть следующими:

- **Экономические факторы:** Возможно, в это время экономическая ситуация в России начала ухудшаться. Экономический спад может снизить спрос на жилье и, следовательно, количество жилищных кредитов.
- **Изменение процентных ставок:** Если в это время произошло повышение процентных ставок по жилищным кредитам, это могло отпугнуть потенциальных заемщиков. Высокие процентные ставки делают кредиты менее привлекательными.
- **Политическая нестабильность:** В 2015 году Россия столкнулась с политическими и экономическими вызовами, такими как санкции и падение цен на нефть. Это могло повлиять на доверие кредиторов и желание людей брать кредиты.
- **Сезонные факторы:** Лето может быть менее активным временем для жилищных сделок из-за отпусков и других факторов. Это может объяснить снижение числа заявок на кредит в августе.

Перейдем к 2021 году, когда наблюдается резкий рост заявок на кредит. Давайте рассмотрим, что могло способствовать этому:

- **Снижение ставок:** В 2021 году банки активно снижали процентные ставки по ипотечным кредитам. Это сделало жилищные кредиты более доступными и привлекательными для потенциальных заемщиков.
- **Государственная поддержка:** Власти предоставили антикризисные льготные программы кредитования на приобретение жилья. Это стимулировало спрос на жилищные кредиты.
- **Восстановление после пандемии:** После тяжелых локдаунов и неопределенности 2020 года экономика начала восстанавливаться. Это могло увеличить интерес к покупке жилья и, следовательно, кредитов.
- **Увеличение доходов домохозяйств:** Несмотря на сложности, многие домохозяйства стали более финансово устойчивыми, что способствовало росту спроса на жилье и кредиты.

1.4 Постановка задачи моделирования

Моя цель - разработать модель, которая может прогнозировать будущее количество заявок на кредиты на основе исторических данных. Это может помочь банкам и кредитным учреждениям лучше планировать свои ресурсы и стратегии. Прогнозирование спроса на кредиты является важной задачей, которая может помочь финансовым учреждениям оптимизировать свои операции и улучшить качество обслуживания клиентов.

Для достижения этой цели, я планирую выполнить следующие задачи:

1. **Выбор подходящих моделей**: Я собираюсь обучить все модели, которые я могу, а именно: SDCA, OnlineGradientDescent, Poisson, fastTree, fastForest, SSA. Каждая из этих моделей имеет свои сильные и слабые стороны, и я намерен исследовать, как они могут быть применимы к моим данным.

2. **Обучение моделей**: Я обучу выбранные модели на моих исторических данных. Это может включать в себя настройку параметров моделей и проверку их производительности с помощью кросс-валидации. Я также буду использовать различные методы оптимизации для улучшения производительности каждой модели.
3. **Тестирование моделей**: Я протестирую модели на отложенной выборке данных, чтобы увидеть, насколько хорошо они могут прогнозировать будущее количество заявок на кредиты. Я буду использовать различные метрики, такие как средняя абсолютная ошибка и среднеквадратичная ошибка, для оценки производительности каждой модели.
4. **Сравнение моделей**: Я сравню все эти модели на качество на тестовых данных. Я буду использовать статистические методы для определения, какая модель дает наилучшие результаты.
5. **Интерпретация результатов**: Наконец, я проанализирую результаты прогнозирования и обсужу их в контексте моего исследования. Я также обсужу возможные ограничения моего подхода и предложу направления для дальнейшего исследования.

2. МОДЕЛИРОВАНИЕ СТАТИСТИЧЕСКИХ ЗАВИСИМОСТЕЙ

2.1 Формализация и классификация переменных

Формализация переменных - это процесс определения переменных, которые будут использоваться в модели. В контексте моего исследования, переменными могут быть дата (время) и количество заявок на кредиты. Дата будет служить независимой переменной, а количество заявок на кредиты - зависимой переменной.

Классификация переменных - это процесс группировки переменных по определенным характеристикам. В моем случае, я могу классифицировать переменные следующим образом:

1. **Независимые переменные**: Это переменные, которые я использую для прогнозирования значения зависимой переменной. В моем случае, это дата.
2. **Зависимые переменные**: Это переменные, значения которых я пытаюсь прогнозировать. В моем случае, это количество заявок на кредиты.
3. **Временные ряды**: Это наборы данных, собранные в хронологическом порядке. В моем случае, количество заявок на кредиты, собранных по датам, является временным рядом.
4. **Целевая переменная**: Это переменная, которую я пытаюсь прогнозировать или объяснить в моей модели. В моем случае, целевой переменной является количество заявок на кредиты.

2.2 Проверка гипотезы о нормальном распределении выходной величины

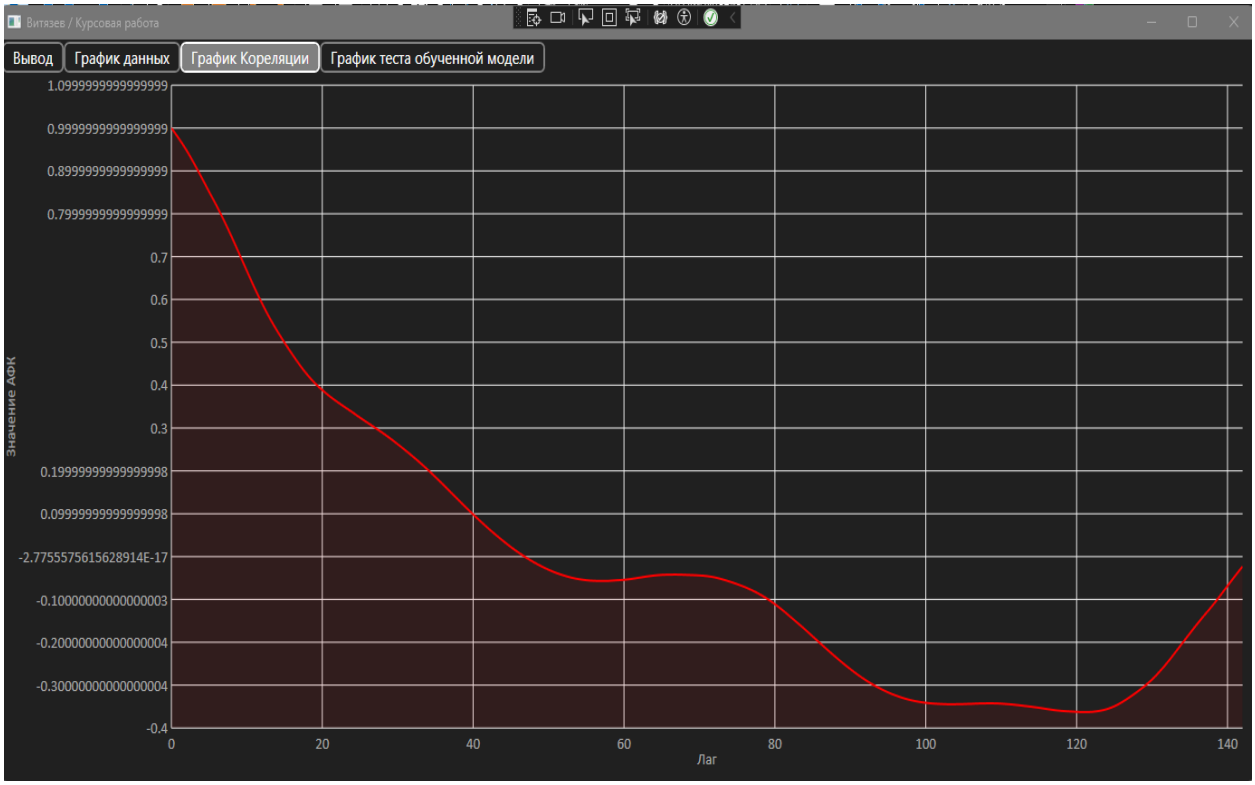
В ходе анализа были проведены два различных статистических теста для проверки гипотезы о нормальности распределения данных: тест Шапиро-Вилка и тест Андерсона-Дарлинга.

- **Тест Шапиро-Вилка** - это широко используемый тест для проверки гипотезы о нормальности распределения данных. P-значение, полученное в результате теста Шапиро-Вилка, составило 0,00011969665338406892. Обычно, если p-значение меньше выбранного уровня значимости (обычно 0,05), то нулевая гипотеза отвергается. В данном случае, p-значение значительно меньше 0,05, что указывает на то, что данные, вероятно, не подчиняются нормальному распределению.
- **Тест Андерсона-Дарлинга** - это еще один тест, используемый для проверки гипотезы о нормальности распределения данных. P-значение, полученное в результате теста Андерсона-Дарлинга, составило 5,8967710645189016E-05. Это значение значительно меньше обычного уровня значимости 0,05, что также указывает на то, что данные, вероятно, не подчиняются нормальному распределению.

Результаты тестов Шапиро-Вилка и Андерсона-Дарлинга указывают на то, что данные, вероятно, не подчиняются нормальному распределению.

2.3 Корреляционный анализ

Для проведения корреляционного анализа я использовал автокорреляционную функцию (АКФ). АКФ - это статистический инструмент, который измеряет корреляцию между значениями моего временного ряда в разные моменты времени. Это позволяет определить, насколько каждое значение в моем временном ряду связано с предыдущими значениями.



«Графическое представление АКФ»

Мои данные показывают, что автокорреляция начинается с 1 (что ожидаемо, поскольку это корреляция временного ряда с самим собой), а затем постепенно уменьшается. Это может указывать на наличие тренда в данных.

Однако, есть некоторые лаги, где автокорреляция снова возрастает. Это может указывать на наличие сезонности в данных, где некоторые паттерны повторяются через определенные промежутки времени.

2.4 Построение модели

В ходе моего исследования я обучил несколько моделей для прогнозирования будущего количества заявок на кредиты. Для этого я использовал различные алгоритмы машинного обучения, каждый из которых имеет свои особенности:

- **SDCA (Stochastic Dual Coordinate Ascent)**: Это алгоритм оптимизации, который используется для обучения линейных моделей. Он эффективен для больших наборов данных и поддерживает параллельные вычисления.
- **OnlineGradientDescent**: Это алгоритм стохастического градиентного спуска, который обновляет параметры модели на каждом шаге обучения, используя только один обучающий пример. Он подходит для обучения на больших наборах данных.
- **Poisson Regression**: Это тип регрессионного анализа, который используется для моделирования счетных данных. В этом случае, он может быть полезен для прогнозирования количества заявок на кредиты.
- **FastTree и FastForest**: Это алгоритмы, основанные на деревьях решений. Они могут моделировать сложные нелинейные зависимости между переменными.

Перед обучением моделей я подготовил данные, взяв последние 11 записей из исходных данных для тестирования моделей. Оставшиеся данные были использованы для обучения моделей.

Также я применил метод скользящего среднего для сглаживания данных перед обучением моделей. Это помогло уменьшить влияние случайных колебаний и выявить общую тенденцию в данных.

Для каждой модели я создал конвейер обработки данных, который включает объединение признаков и нормализацию, а затем применил соответствующий алгоритм обучения. После обучения модели были сохранены для дальнейшего использования.

Кроме того, я обучил модель **SSA** (Singular Spectrum Analysis), которая использует методы анализа временных рядов для прогнозирования будущего количества заявок на кредиты.

В результате я получил несколько обученных моделей, которые можно использовать для прогнозирования будущего количества заявок на кредиты. В следующих разделах я буду оценивать производительность этих моделей и сравнивать их результаты.

2.4.1 Структурная идентификация модели

Структурная идентификация модели включает определение формы модели, которая наилучшим образом описывает данные. В моем исследовании я использовал несколько различных структур моделей, включая SDCA, OnlineGradientDescent, Poisson, FastTree, FastForest и SSA.

- **SDCA (Stochastic Dual Coordinate Ascent):** Это линейная модель, которая использует стохастический алгоритм оптимизации. В этой модели я использовал дату как единственный признак.
- **OnlineGradientDescent:** Это также линейная модель, но она использует алгоритм стохастического градиентного спуска для обучения. В этой модели я также использовал дату как единственный признак и применил нормализацию MinMax к признакам.
- **Poisson Regression:** Это модель регрессии Пуассона, которая подходит для моделирования счетных данных. В этой модели я использовал дату как единственный признак и применил нормализацию MinMax к признакам.
- **FastTree и FastForest:** Это модели, основанные на деревьях решений. Они могут моделировать сложные нелинейные зависимости между переменными. В этих моделях я использовал дату как единственный признак.
- **SSA (Singular Spectrum Analysis):** Это модель временных рядов, которая использует методы анализа временных рядов для прогнозирования. В этой модели я использовал количество заявок на кредиты как единственный признак.

2.4.2 Параметрическая идентификация модели

Параметрическая идентификация модели включает оценку параметров модели. В моем исследовании я использовал различные алгоритмы обучения для оценки параметров моделей:

- **SDCA** (Stochastic Dual Coordinate Ascent): В этой модели я использовал максимальное количество итераций равное 100 для обучения модели.
- **OnlineGradientDescent**: В этой модели я не указывал специфических параметров для обучения, поэтому были использованы параметры по умолчанию.
- **Poisson Regression**: В этой модели я также не указывал специфических параметров для обучения, поэтому были использованы параметры по умолчанию.
- **FastTree и FastForest**: В этих моделях я также не указывал специфических параметров для обучения, поэтому были использованы параметры по умолчанию.
- **SSA (Singular Spectrum Analysis)**: В этой модели я использовал окно размером 11, длину серии 23, размер обучения равный количеству данных и горизонт

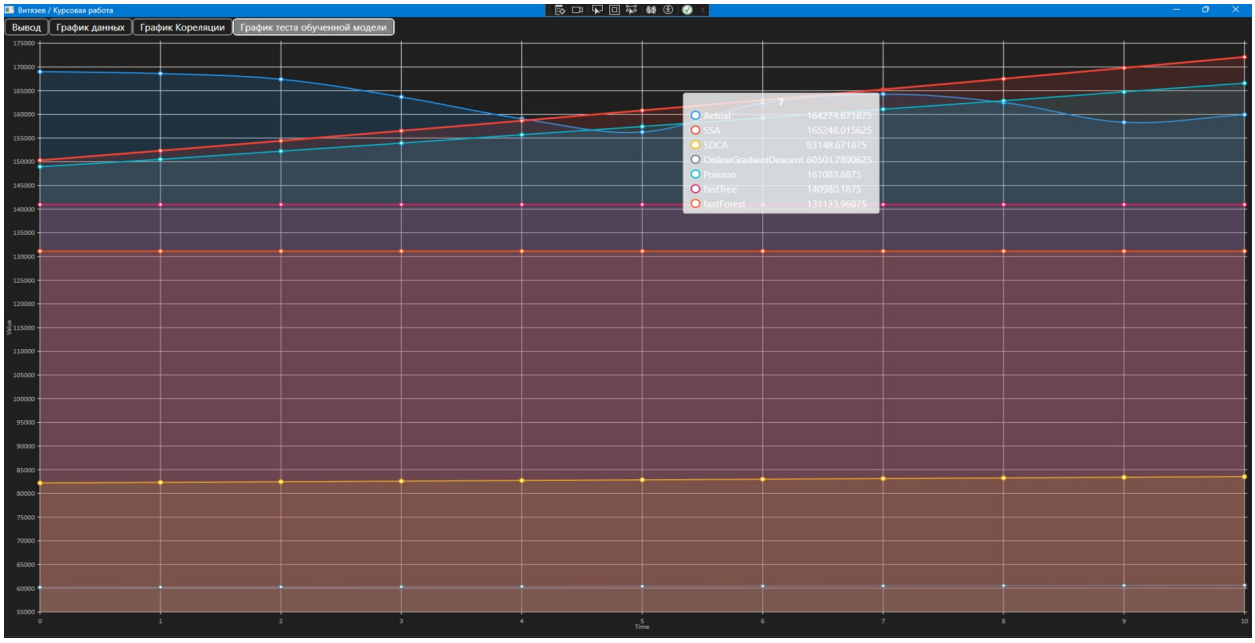
прогнозирования 11. Также я установил уровень доверия 0.9 для расчета доверительных интервалов прогнозов.

Все эти параметры были выбраны на основе характеристик данных и целей исследования. Они были оценены в процессе обучения моделей на основе исходных данных.

3. Исследование моделей

В ходе исследования я провел тестирование обученных моделей на тестовых данных. Для оценки производительности моделей я использовал среднюю абсолютную ошибку (MAE). Результаты тестирования моделей представлены ниже:

- **SSA**: MAE = 8211,678
- **SDCA**: MAE = 79982,21
- **OnlineGradientDescent**: MAE = 102448,52
- **Poisson**: MAE = 7940,05
- **FastTree**: MAE = 21880,32
- **FastForest**: MAE = 31726,54



«График предсказанных значений и актуальных значений»

Из этих результатов видно, что модель Poisson показала наименьшую среднюю абсолютную ошибку (MAE) среди всех моделей. Однако, несмотря на это, модель **SSA** лучше всего угадывает будущие значения по графику, что указывает на ее способность лучше улавливать общую динамику данных.

В итоге, с учетом результатов тестирования и способности моделей улавливать общую динамику данных, я **выбираю модель SSA** для прогнозирования будущего количества заявок на кредиты.

4. Программная реализация и численное исследование результатов моделирования

4.1 Обоснование выбора и описание программного обеспечения

Для реализации моего исследования я использовал следующее программное обеспечение:

- **Язык программирования - C#**: C# является современным, объектно-ориентированным языком программирования, разработанным Microsoft. Он обеспечивает мощные функции для разработки различных типов приложений, включая научные исследования и анализ данных.
- **IDE - Visual Studio Community 2022**: Visual Studio Community - это бесплатная версия популярной среды разработки от Microsoft. Она предлагает широкий спектр инструментов для разработки, тестирования и отладки кода.
- **Библиотеки**: Я использовал несколько библиотек для обработки данных и обучения моделей:
    - **ML.NET**: Это библиотека машинного обучения от Microsoft, которая предоставляет функции для обучения и оценки моделей прямо в C#.
    - **LiveCharts**: Эта библиотека используется для создания интерактивных графиков в WPF.
    - **MathNET**: Это библиотека для численных вычислений в .NET.
    - **Accord**: Это фреймворк .NET для научных вычислений, который включает поддержку многих алгоритмов машинного обучения.
    - **EPPlus**: Это библиотека для работы с файлами Excel в .NET.
- **Фреймворк - .NET 7**: .NET 7 - это последняя версия платформы .NET от Microsoft. Она предлагает множество функций для разработки высокопроизводительных приложений и поддерживает множество библиотек и инструментов.
- **GUI - WPF (Windows Presentation Foundation)**: WPF - это фреймворк для построения настольных приложений с богатым пользовательским интерфейсом на платформе .NET.

Выбор этих инструментов был обусловлен их мощными возможностями, широкой поддержкой сообщества и хорошей интеграцией друг с другом. Они позволили мне эффективно обрабатывать данные, обучать модели, оценивать их производительность и визуализировать результаты.

4.2 Описание основных модулей программы

4.2.1 Модуль считывания данных «ExcelReader»

```
namespace VityazevFinalWork.Solution.Misc
{
    Ссылок: 1
    internal class ExcelReader
    {
        Ссылок: 1
        public static List<TData> ReadBigExcel()
        {
            var result = new List<TData>();

            using (var package = new ExcelPackage(new FileInfo(@"D:\Downloads\Stat_morgage_tables_10 (1).xlsx")))
            {
                var worksheet = package.Workbook.Worksheets[5];
                // 2009 - 2013
                result.AddRange(ReadTables(worksheet, 139, 1, 5));
                // 2014 - 2017
                result.AddRange(ReadTables(worksheet, 86, 1, 4));
                // 2018 - 2020 (оставляю данные за 2021 год для тестов)
                result.AddRange(ReadTables(worksheet, 5, 1, 4));
            }

            return result.OrderBy(d=>d.date).ToList();
        }

        Ссылок: 3
        private static List<TData> ReadTables(ExcelWorksheet worksheet, int startI, int startJ, int count)
        {
            var result = new List<TData>();
            for (int n=0; n<count; n++)
            {
                result.AddRange( ReadTable(worksheet, startI, startJ));
                startI += 13;
            }
            return result;
        }

        Ссылок: 1
        private static List<TData> ReadTable(ExcelWorksheet worksheet,int startI, int startJ)
        {
            string year = worksheet.Cells[startI, startJ].Value.ToString().Replace(" г.", "");
            Debug.WriteLine($"Got year = {year}");
            startI += 2;

            var result = new List<TData>
            {
                new TData
                {
                    date = DateTime.ParseExact($"{worksheet.Cells[startI, startJ].Value}.{year}", "dd.MM.yyyy", CultureInfo.InvariantCulture),
                    amount = Convert.ToDouble(worksheet.Cells[startI, startJ + 1].Value)
                }
            };
            for (int i = startI + 1; i < startI + 1 + 10; i++)
            {
                double am = (double)worksheet.Cells[i, startJ + 1].Value - (double)worksheet.Cells[i - 1, startJ + 1].Value;
                result.Add(new TData
                {
                    date = DateTime.ParseExact($"{worksheet.Cells[i, startJ].Value}.{year}", "dd.MM.yyyy", CultureInfo.InvariantCulture),
                    amount = am
                });
            }
            return result;
        }
    }
}
```

Модуль “ExcelReader” представляет собой внутренний класс, который используется для чтения данных из файла Excel. Он содержит три основных метода: ReadBigExcel, ReadTables и ReadTable.

- **ReadBigExcel**: Этот метод отвечает за чтение данных из файла Excel. Он открывает файл Excel, выбирает нужный лист и считывает данные из определенных таблиц. Данные из каждой таблицы считываются с помощью метода ReadTables. В конце работы метода данные сортируются по дате и возвращаются в виде списка объектов типа TData.
- **ReadTables**: Этот метод используется для чтения данных из нескольких таблиц на одном листе файла Excel. Он принимает в качестве параметров рабочий лист,

начальные координаты и количество таблиц для чтения. Для каждой таблицы вызывается метод ReadTable, и результаты объединяются в один список.

- **ReadTable**: Этот метод считывает данные из одной таблицы. Он принимает в качестве параметров рабочий лист и начальные координаты таблицы. Сначала метод считывает год из заголовка таблицы. Затем он считывает данные из каждой строки таблицы, преобразуя их в объекты типа TData. Каждый объект TData содержит дату и количество заявок на кредиты.

Все эти методы работают вместе, чтобы считать данные из файла Excel и преобразовать их в формат, который можно использовать для дальнейшего анализа и обучения моделей. Этот модуль является важной частью системы, так как он обеспечивает первый шаг в процессе обработки данных - считывание данных.

4.2.2 Модуль вычисления характеристик «Specifications»

```
using LiveCharts;
using LiveCharts.Wpf;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using VityazevFinalWork.Solution.Models;

namespace VityazevFinalWork.Solution.Modules
{
    Ссылок: 4
    internal class Specifications
    {
        private readonly List<TData> _data;
        private readonly CartesianChart? _chart;

        Ссылок: 3
        public Specifications(List<TData> data, CartesianChart? chart)
        {
            _data = data;
            _chart = chart;
        }

        // срденее
        Ссылок: 6
        public double Mean()
        {
            return _data.Average(d => d.amount);
        }

        // медиана
        Ссылок: 1
        public double Median()
        {
            var sortedAmounts = _data.Select(d => d.amount).OrderBy(a => a).ToList();
            int count = sortedAmounts.Count;
            if (count % 2 == 0)
            {
                return (sortedAmounts[count / 2 - 1] + sortedAmounts[count / 2]) / 2;
            }
            else
            {
                return sortedAmounts[count / 2];
            }
        }

        // стандартное отклонение
        Ссылок: 4
        public double StandardDeviation()
        {
            double mean = Mean();
            double sumOfSquaresOfDifferences = _data.Select(d => Math.Pow(d.amount - mean, 2)).Sum();
            double standardDeviation = Math.Sqrt(sumOfSquaresOfDifferences / _data.Count);
            return standardDeviation;
        }
```

```
52          // мода
            Ссылок: 0
53          public double Mode()
54          {
55              return _data.GroupBy(n => n.amount)
56                          .OrderByDescending(g => g.Count())
57                          .Select(g => g.Key).FirstOrDefault();
58          }
59
60          // дисперсия
            Ссылок: 1
61          public double Variance()
62          {
63              double mean = Mean();
64              double variance = _data.Average(d => Math.Pow(d.amount - mean, 2));
65              return variance;
66          }
67
68
            Ссылок: 2
69          public (double, double) MinMax()
70          {
71              return (_data.MinBy(n => n.amount).amount, _data.MaxBy(n => n.amount).amount);
72          }
73
74          // скос
            Ссылок: 1
75          public double Skewness()
76          {
77              double mean = Mean();
78              double standardDeviation = StandardDeviation();
79              double skewness = _data.Average(d => Math.Pow(d.amount - mean, 3)) / Math.Pow(standardDeviation, 3);
80              return skewness;
81          }
82
83          // эксцесс
            Ссылок: 1
84          public double Kurtosis()
85          {
86              double mean = Mean();
87              double standardDeviation = StandardDeviation();
88              double kurtosis = _data.Average(d => Math.Pow(d.amount - mean, 4)) / Math.Pow(standardDeviation, 4);
89              return kurtosis;
90          }
91
92          // Для всяких тестов на нормальнео распрделение
            Ссылок: 2
93          public double[] GetNormolizedData()
94          {
95              double mean = Mean();
96              double stdDev = StandardDeviation();
97
98              return _data.Select(d => (d.amount - mean) / stdDev).ToArray();
99          }
```

```
            Ссылок: 1
101         public void BuildGraph()
102         {
103             if (_chart != null)
104             {
105                 _chart.Dispatcher.Invoke(() =>
106                 {
107                     LineSeries lineSeries = new LineSeries
108                     {
109                         Title = "Заявки на кредит",
110                         Values = new ChartValues<double>(),
111                         PointGeometry = null // Это убирает маркеры точек на линии
112                     };
113                     // Заполняем LineSeries данными
114                     foreach (var data in _data)
115                     {
116                         lineSeries.Values.Add(data.amount);
117                     }
118                     // Добавляем LineSeries на график
119                     _chart.Series.Add(lineSeries);
120
121                     // Устанавливаем метки для оси X
122                     _chart.AxisX.Add(new Axis
123                     {
124                         Title = "Время",
125                         Labels = _data.Select(d => d.date.ToString()).ToList(),
126                         Separator = new Separator { Step = 10, IsEnabled = false } // Настраиваем шаг меток
127                     });
128
129                     // Устанавливаем метки для оси Y
130                     _chart.AxisY.Add(new Axis
131                     {
132                         Title = "Количество заявок",
133                         LabelFormatter = value => value.ToString("N")
134                     });
135                 });
136             }
137             else
138             {
139                 MessageBox.Show("_chart is null");
140             }
141         }
142     }
```

Модуль “Specifications” представляет собой внутренний класс, который используется для вычисления статистических характеристик данных и построения графиков. Он содержит следующие методы:

- **Mean**: Этот метод вычисляет среднее значение количества заявок на кредиты. Он делает это, суммируя все значения и делит их на общее количество значений.
- **Median**: Этот метод вычисляет медиану количества заявок на кредиты. Для этого он сначала сортирует значения по возрастанию, а затем выбирает среднее значение. Если количество значений четное, то медианой будет среднее двух средних значений.
- **StandardDeviation**: Этот метод вычисляет стандартное отклонение количества заявок на кредиты. Он делает это, вычисляя среднее значение, затем находит разность между каждым значением и средним, возводит разность в квадрат, суммирует все квадраты разностей, делит сумму на количество значений и извлекает квадратный корень из результата.
- **Mode**: Этот метод вычисляет моду количества заявок на кредиты. Он делает это, группируя значения по количеству и выбирая значение с наибольшим количеством повторений.
- **Variance**: Этот метод вычисляет дисперсию количества заявок на кредиты. Он делает это, вычисляя среднее значение, затем находит разность между каждым значением и средним, возводит разность в квадрат, суммирует все квадраты разностей и делит сумму на количество значений.
- **MinMax**: Этот метод возвращает минимальное и максимальное количество заявок на кредиты. Он делает это, просто находя минимальное и максимальное значение в данных.
- **Skewness**: Этот метод вычисляет коэффициент асимметрии распределения количества заявок на кредиты. Он делает это, вычисляя среднее значение и стандартное отклонение, затем находит разность между каждым значением и средним, возводит разность в куб, суммирует все кубы разностей, делит сумму на количество значений и делит результат на куб стандартного отклонения.
- **Kurtosis**: Этот метод вычисляет коэффициент эксцесса распределения количества заявок на кредиты. Он делает это, вычисляя среднее значение и стандартное отклонение, затем находит разность между каждым значением и средним, возводит разность в четвертую степень, суммирует все четвертые степени разностей, делит сумму на количество значений и делит результат на четвертую степень стандартного отклонения.
- **GetNormolizedData**: Этот метод возвращает нормализованные данные, которые могут быть использованы для тестов на нормальность распределения. Он делает это, вычисляя среднее значение и стандартное отклонение, затем находит разность между каждым значением и средним и делит разность на стандартное отклонение.

- **BuildGraph**: Этот метод строит график количества заявок на кредиты. Он делает это, создавая новую серию данных для графика, добавляя в нее все значения из данных, и затем добавляя серию на график.

  Все эти методы работают вместе, чтобы обеспечить подробный анализ данных.

4.2.3 Модуль проверки гипотезы о нормальном распределении и построение АФК «Dependencies»

```
namespace VityazevFinalWork.Solution.Modules
{
    Ссылок: 2
    internal class Dependencies
    {
        private readonly List<TData> _data;

        Ссылок: 1
        public Dependencies(List<TData> data)
        {
            _data = data;
        }

        Ссылок: 1
        public (double w, double p) lib_ShapiroWilkTest()
        {
            var test = new ShapiroWilkTest(_data.Select(d => d.amount).ToArray());
            return (test.Statistic, test.PValue);
        }

        Ссылок: 1
        public (double w, double p) lib_AndersonDarlingTest()
        {
            var sp = new Specifications(_data, null);

            var adTest = new AndersonDarlingTest(sp.GetNormolizedData(), new Accord.Statistics.Distributions.Univariate.NormalDistribution());

            return (adTest.Statistic, adTest.PValue);
        }

        Ссылок: 1
        public double[] lib_AutoCorelation(CartesianChart? chart = null)
        {
            double[] acf = Correlation.Auto(_data.Select(d=>d.amount).ToArray());
            if (chart != null)
            {
                chart.Dispatcher.Invoke(() =>
                {
                    LineSeries lineSeries = new LineSeries
                    {
                        Title = "Автокорреляционной функция",
                        Values = new ChartValues<ObservablePoint>(),
                        Stroke = new SolidColorBrush(Color.FromRgb(255, 0, 0)),
                        Fill = new SolidColorBrush(Color.FromArgb(20, 255, 0, 0)),
                        PointGeometry = null
                    };
                    for (int i=0; i < acf.Length; i++)
                    {
                        lineSeries.Values.Add(new ObservablePoint(i, Math.Round( acf[i],4)));
                    }
                    chart.Series = new SeriesCollection() { lineSeries };

                    chart.AxisX.Clear();
                    chart.AxisY.Clear();
                    chart.AxisX.Add(new Axis { Title = "Лаг", FontSize=15 });
                    chart.AxisY.Add(new Axis { Title = "Значение АФК", FontSize=15 });
                });
            }
            return acf;
        }

    }
}

```

Модуль “Dependencies” представляет собой внутренний класс, который используется для проверки гипотезы о нормальном распределении данных и построения автокорреляционной функции (АФК). Для выполнения этих задач используются библиотеки Accord.NET и MathNet.Numerics. Они содержат следующие методы:

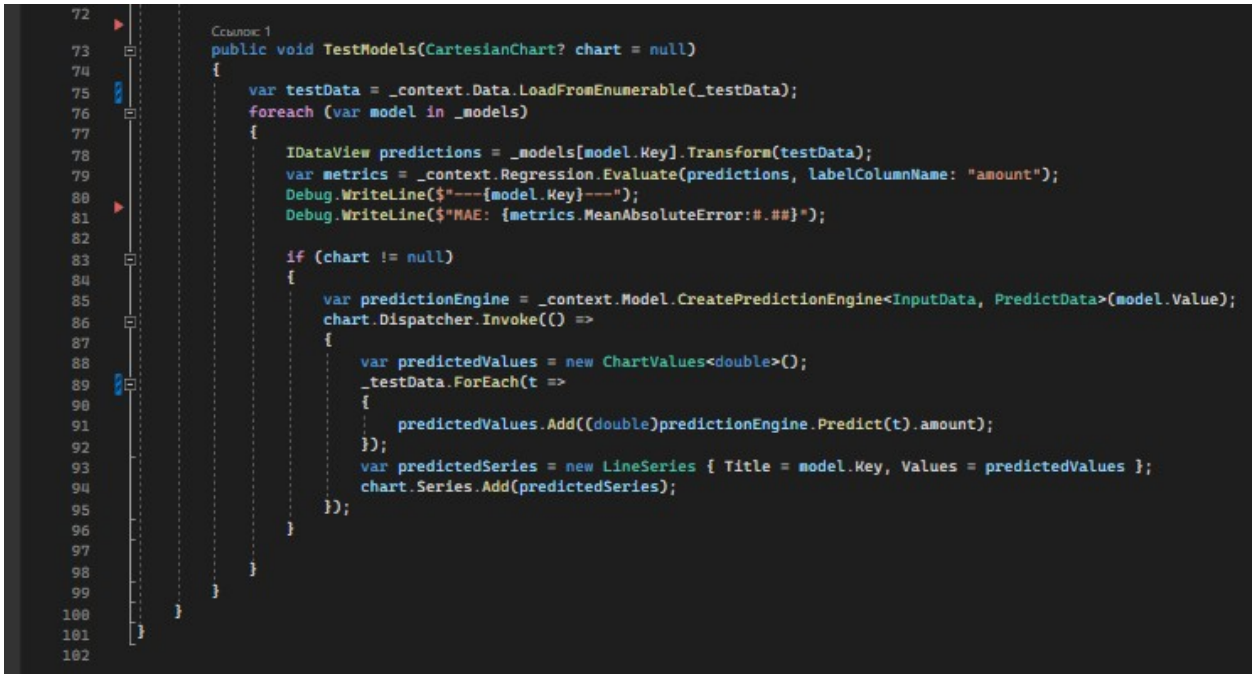
- **lib_ShapiroWilkTest**: Этот метод, основанный на библиотеке Accord.NET, выполняет тест Шапиро-Уилка на нормальность данных. Он возвращает статистику теста и p-значение.

- **lib_AndersonDarlingTest**: Этот метод, основанный на библиотеке Accord.NET, выполняет тест Андерсона-Дарлинга на нормальность данных. Он возвращает статистику теста и p-значение.
- **lib_AutoCorelation**: Этот метод, основанный на библиотеке MathNet.Numerics, вычисляет автокорреляционную функцию данных и строит график АФК, если предоставлен объект CartesianChart.

  Этот класс принимает в качестве параметров список данных.

4.2.4 Модуль обучения и тестов регрессионных моделей «RegressionModel»

```
namespace VityazevFinalWork.Solution.Modules
{
    Ссылок: 2
    internal class RegressionModel
    {
        private readonly List<InputData> _data;
        private readonly List<InputData> _testData;
        private readonly MLContext _context = new MLContext();
        private Dictionary<string, ITransformer> _models = new Dictionary<string, ITransformer>();

        Ссылок: 1
        public RegressionModel(List<TData> data)
        {
            _data = new List<InputData>();
            data.ForEach(d =>
            {
                _data.Add(d.ToInpData());
            });
            _testData = _data.TakeLast(11).ToList();
            _data = _data.Take(_data.Count - 11).ToList();
        }

        Ссылок: 1
        public void TrainAll()
        {
            var dataView = _context.Data.LoadFromEnumerable(_data);

            var sdcaPipeline = _context.Transforms.Concatenate("Features", "date")
                .Append(_context.Regression.Trainers.Sdca(labelColumnName: "amount", maximumNumberOfIterations: 100));

            var onlineGradientDescentPipeline = _context.Transforms.Concatenate("Features", "date")
                .Append(_context.Transforms.NormalizeMinMax("Features"))
                .Append(_context.Regression.Trainers.OnlineGradientDescent(labelColumnName: "amount"));

            var poissonPipeline = _context.Transforms.Concatenate("Features", "date")
                .Append(_context.Transforms.NormalizeMinMax("Features"))
                .Append(_context.Regression.Trainers.LbfgsPoissonRegression(labelColumnName: "amount"));

            var fastTree = _context.Transforms.Concatenate("Features", "date")
                .Append(_context.Regression.Trainers.FastTree(labelColumnName: "amount"));

            var fastForest = _context.Transforms.Concatenate("Features", "date")
                .Append(_context.Regression.Trainers.FastForest(labelColumnName: "amount"));

            // Обучение моделей
            _models["SDCA"] = sdcaPipeline.Fit(dataView);
            _models["OnlineGradientDescent"] = onlineGradientDescentPipeline.Fit(dataView);
            _models["Poisson"] = poissonPipeline.Fit(dataView);
            _models["fastTree"] = fastTree.Fit(dataView);
            _models["fastForest"] = fastForest.Fit(dataView);
        }

        Ссылок: 2
        public void PredictAll(DateTime date)
        {
            var data = new InputData() { date = Shared.ConvertToUnixTime(date) };

            foreach (var model in _models)
            {
                var predictionEngine = _context.Model.CreatePredictionEngine<InputData, PredictData>(model.Value);
                var result = predictionEngine.Predict(data);
                Debug.WriteLine($"Predicted amount for date: {data.date} using {model.Key} = {result.amount}");
            }
        }
```



```
72
        Ссылок: 1
73      public void TestModels(CartesianChart? chart = null)
74      {
75          var testData = _context.Data.LoadFromEnumerable(_testData);
76          foreach (var model in _models)
77          {
78              IDataView predictions = _models[model.Key].Transform(testData);
79              var metrics = _context.Regression.Evaluate(predictions, labelColumnName: "amount");
80              Debug.WriteLine($"---{model.Key}---");
81              Debug.WriteLine($"MAE: {metrics.MeanAbsoluteError:#.##}");
82
83              if (chart != null)
84              {
85                  var predictionEngine = _context.Model.CreatePredictionEngine<InputData, PredictData>(model.Value);
86                  chart.Dispatcher.Invoke(() =>
87                  {
88                      var predictedValues = new ChartValues<double>();
89                      _testData.ForEach(t =>
90                      {
91                          predictedValues.Add((double)predictionEngine.Predict(t).amount);
92                      });
93                      var predictedSeries = new LineSeries { Title = model.Key, Values = predictedValues };
94                      chart.Series.Add(predictedSeries);
95                  });
96              }
97
98          }
99      }
100 }
101 }
102
```

Модуль “RegressionModel” представляет собой внутренний класс, который используется для обучения и тестирования регрессионных моделей. Он использует библиотеку машинного обучения Microsoft ML.NET для обучения моделей и оценки их производительности. Он содержит следующие методы:

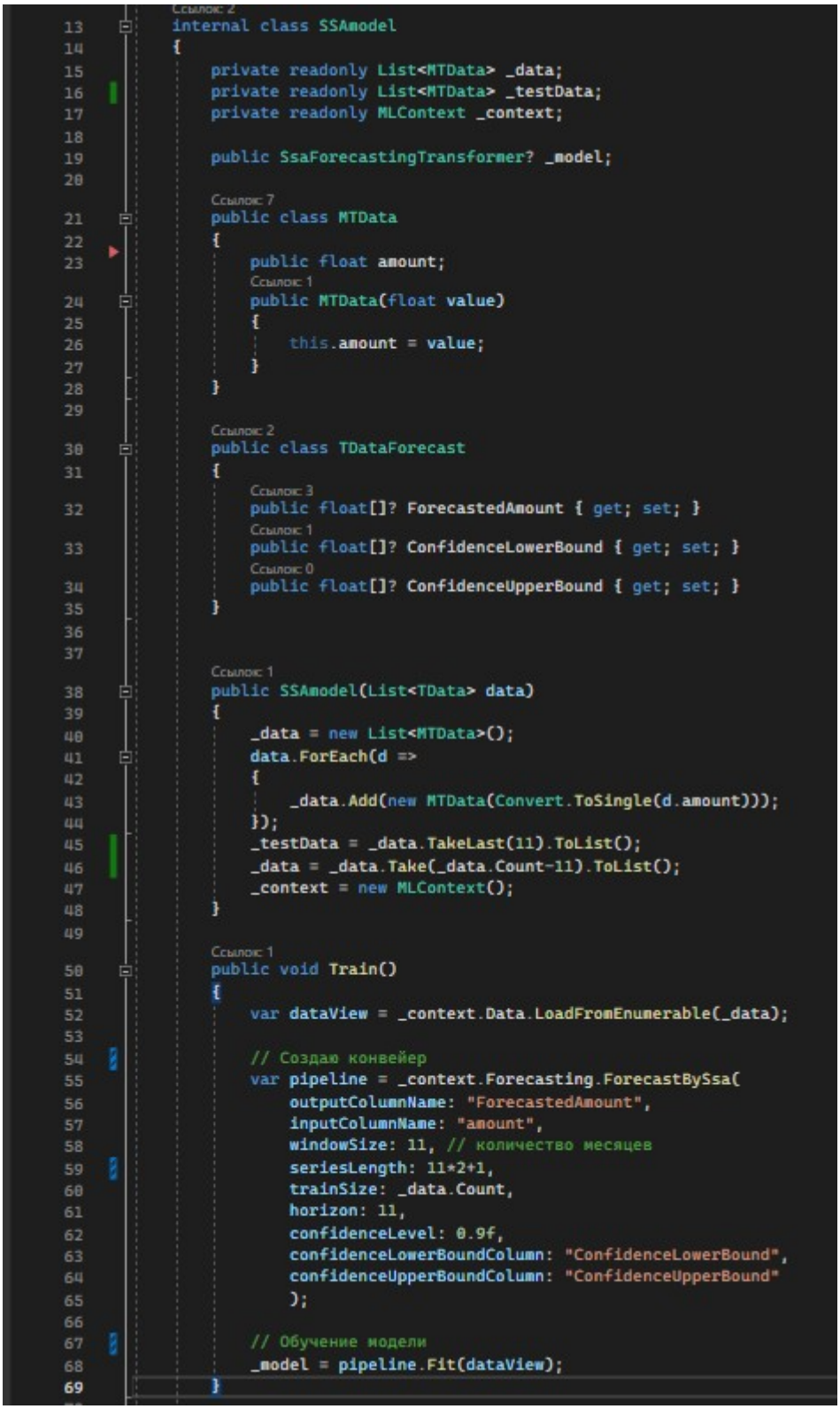
- **TrainAll**: Этот метод обучает все модели на данных. Он создает конвейеры обучения для каждой модели, включая SDCA, OnlineGradientDescent, Poisson, FastTree и FastForest, и затем обучает каждую модель на данных.
- **PredictAll**: Этот метод делает прогнозы для всех моделей на основе указанной даты. Он создает движок прогнозирования для каждой модели и делает прогноз для указанной даты.
- **TestModels**: Этот метод тестирует все модели на тестовых данных. Он преобразует тестовые данные в формат, подходящий для моделей, оценивает производительность каждой модели на тестовых данных и выводит среднюю абсолютную ошибку (MAE) для каждой модели.

В конструкторе класса “RegressionModel” происходит инициализация данных для обучения и тестирования моделей. Конструктор принимает список данных типа TData, который затем преобразуется в список объектов типа InputData. Это делается для того, чтобы данные были в формате, подходящем для обучения моделей ML.NET.

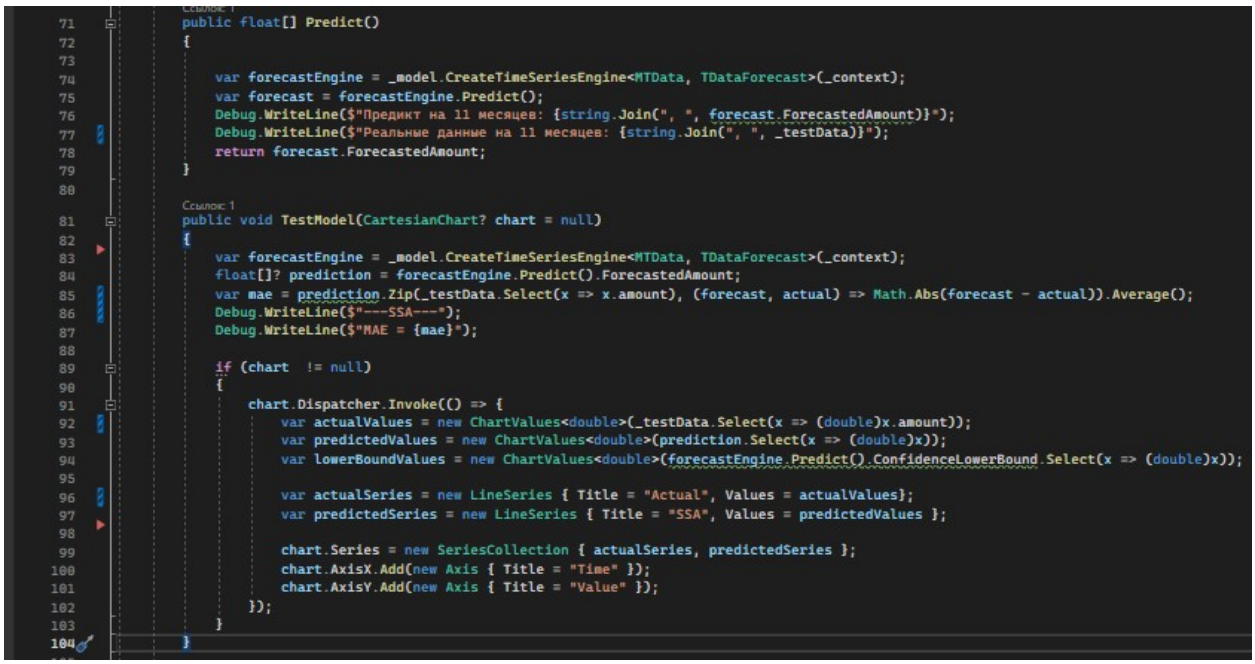
Тестовые данные выбираются из исходных данных. В частности, последние 11 записей из исходных данных отделяются и используются в качестве тестовых данных. Оставшиеся данные используются для обучения моделей. Это делается для того, чтобы обеспечить независимость тестовых данных от данных, использованных для обучения

моделей, что является важным принципом при оценке производительности моделей машинного обучения.

4.2.5 Модуль обучения и тестов модели SSA «SSAmodel»



```
Ссылок: 2
internal class SSAmodel
{
    private readonly List<MTData> _data;
    private readonly List<MTData> _testData;
    private readonly MLContext _context;

    public SsaForecastingTransformer? _model;

    Ссылок: 7
    public class MTData
    {
        public float amount;
        Ссылок: 1
        public MTData(float value)
        {
            this.amount = value;
        }
    }

    Ссылок: 2
    public class TDataForecast
    {
        Ссылок: 3
        public float[]? ForecastedAmount { get; set; }
        Ссылок: 1
        public float[]? ConfidenceLowerBound { get; set; }
        Ссылок: 0
        public float[]? ConfidenceUpperBound { get; set; }
    }


    Ссылок: 1
    public SSAmodel(List<TData> data)
    {
        _data = new List<MTData>();
        data.ForEach(d =>
        {
            _data.Add(new MTData(Convert.ToSingle(d.amount)));
        });
        _testData = _data.TakeLast(11).ToList();
        _data = _data.Take(_data.Count-11).ToList();
        _context = new MLContext();
    }

    Ссылок: 1
    public void Train()
    {
        var dataView = _context.Data.LoadFromEnumerable(_data);

        // Создаю конвейер
        var pipeline = _context.Forecasting.ForecastBySsa(
            outputColumnName: "ForecastedAmount",
            inputColumnName: "amount",
            windowSize: 11, // количество месяцев
            seriesLength: 11*2+1,
            trainSize: _data.Count,
            horizon: 11,
            confidenceLevel: 0.9f,
            confidenceLowerBoundColumn: "ConfidenceLowerBound",
            confidenceUpperBoundColumn: "ConfidenceUpperBound"
            );

        // Обучение модели
        _model = pipeline.Fit(dataView);
    }
```



```
71      public float[] Predict()
72      {
73
74          var forecastEngine = _model.CreateTimeSeriesEngine<MTData, TDataForecast>(_context);
75          var forecast = forecastEngine.Predict();
76          Debug.WriteLine($"Предикт на 11 месяцев: {string.Join(", ", forecast.ForecastedAmount)}");
77          Debug.WriteLine($"Реальные данные на 11 месяцев: {string.Join(", ", _testData)}");
78          return forecast.ForecastedAmount;
79      }
80
81      public void TestModel(CartesianChart? chart = null)
82      {
83          var forecastEngine = _model.CreateTimeSeriesEngine<MTData, TDataForecast>(_context);
84          float[]? prediction = forecastEngine.Predict().ForecastedAmount;
85          var mae = prediction.Zip(_testData.Select(x => x.amount), (forecast, actual) => Math.Abs(forecast - actual)).Average();
86          Debug.WriteLine($"---SSA---");
87          Debug.WriteLine($"MAE = {mae}");
88
89          if (chart  != null)
90          {
91              chart.Dispatcher.Invoke(() => {
92                  var actualValues = new ChartValues<double>(_testData.Select(x => (double)x.amount));
93                  var predictedValues = new ChartValues<double>(prediction.Select(x => (double)x));
94                  var lowerBoundValues = new ChartValues<double>(forecastEngine.Predict().ConfidenceLowerBound.Select(x => (double)x));
95
96                  var actualSeries = new LineSeries { Title = "Actual", Values = actualValues};
97                  var predictedSeries = new LineSeries { Title = "SSA", Values = predictedValues };
98
99                  chart.Series = new SeriesCollection { actualSeries, predictedSeries };
100                 chart.AxisX.Add(new Axis { Title = "Time" });
101                 chart.AxisY.Add(new Axis { Title = "Value" });
102             });
103         }
104     }
```

Модуль “SSAmodel” представляет собой внутренний класс, который используется для обучения и тестирования модели Singular Spectrum Analysis (SSA) для прогнозирования временных рядов. Он использует библиотеку машинного обучения Microsoft ML.NET для обучения модели и оценки ее производительности. Он содержит следующие методы:

- **Train**: Этот метод обучает модель SSA на данных. Он создает конвейер обучения для модели SSA, указывая параметры модели, такие как размер окна, длину серии, размер обучения, горизонт прогноза и уровень доверия. Затем он обучает модель на данных.
- **Predict**: Этот метод делает прогнозы для модели SSA на основе обученной модели. Он создает движок прогнозирования для модели и делает прогноз для следующих 11 месяцев.
- **TestModel**: Этот метод тестирует модель SSA на тестовых данных. Он создает движок прогнозирования для модели, делает прогнозы для тестовых данных и вычисляет среднюю абсолютную ошибку (MAE) между прогнозами и реальными данными.

Конструктор класса “SSAmodel” принимает список данных типа `TData`, который затем преобразуется в список объектов типа `MTData`. Это делается для того, чтобы данные были в формате, подходящем для обучения моделей ML.NET. Последние 11 записей из исходных данных отделяются и используются в качестве тестовых данных, а остальные данные используются для обучения модели.

4.2.6 Основной модуль «Main»

```
namespace VityazevFinalWork.Solution
{
    Ссылок: 3
    internal class Main
    {
        private readonly RichTextBox _logsBox;
        private readonly List<TData> _data;
        private readonly CartesianChart _chart, _korChart, _modelChart;

        Ссылок: 1
        public Main(RichTextBox richTextBox, CartesianChart chart, CartesianChart korChart, CartesianChart modelChart)
        {
            _logsBox = richTextBox;
            _data = ExcelReader.ReadBigExcel();
            _chart = chart;
            _korChart = korChart;
            _modelChart = modelChart;
            _data = Shared.SmoothData(_data, 11);
        }

        Ссылок: 1
        public void Run(bool async = false)
        {
            if (async)
            {
                Task.Run(MainWorker);
            }
            else
            {
                MainWorker();
            }
        }

        Ссылок: 2
        private void MainWorker()
        {
            print("1. Анализ характеристик объекта исследования", true);

            var sp = new Specifications(_data, _chart);
            print($"Среднее = {sp.Mean()}");
            print($"Медиана = {sp.Median()}");
            print($"Стандартное отклонение = {sp.StandardDeviation()}");
            print($"Диспесрсия = {sp.Variance()}");
            print($"Минимум = {sp.MinMax().Item1}");
            print($"Максимум = {sp.MinMax().Item2}");
            print($"Скос = {sp.Skewness()}");
            print($"Эксцесс = {sp.Kurtosis()}");
            sp.BuildGraph();


            print("2. Моделирование статистических зависимостей", true);
            var dp = new Dependencies(_data);
            print($"Шапиро-Вилк тест PValue: {dp.lib_ShapiroWilkTest().p}");
            print($"Андерсон-Дарлинг тест Pvalue: {dp.lib_AndersonDarlingTest().p}");
            dp.lib_AutoCorelation(_korChart);
```

```
            var ssa = new SSAmodel(_data);
            ssa.Train();
            var ssaPredict =  ssa.Predict();
            ssa.TestModel(_modelChart);

            var rm = new RegressionModel(_data);
            rm.TrainAll();
            rm.PredictAll(_data[0].date);
            rm.PredictAll(_data[_data.Count - 1].date);
            rm.TestModels(_modelChart);

            print("4. Модель SSA", true);
            print($"Предсказанные значения на 11 месяцев: {string.Join(' ', ssaPredict)}");
            print($"Реальные значения на 11 месяцев: {string.Join(' ', _data.TakeLast(11).ToArray().Select(d=>Math.Round( d.amount)).ToArray())}");
        }


        Ссылок: 15
        private void print(string text, bool isTitel = false)
        {
            _logsBox.Dispatcher.Invoke(() =>
            {
                Paragraph p1 = new Paragraph();
                p1.Inlines.Add(new Run(text)
                {
                    FontSize = isTitel ? 20 : 15,
                    FontWeight = isTitel ? FontWeights.Bold : FontWeights.Light
                });
                _logsBox.Document.Blocks.Add(p1);
            });
        }
    }
}
```

Модуль “Main” представляет собой внутренний класс, который служит в качестве основного модуля для выполнения всего процесса анализа данных и обучения моделей. Он содержит следующие методы:

- **Run**: Этот метод запускает основной рабочий процесс, который выполняет все шаги анализа данных и обучения моделей. Он может запускать рабочий процесс асинхронно или синхронно в зависимости от переданного параметра.
- **MainWorker**: Этот метод выполняет основной рабочий процесс. Он выполняет следующие шаги:
    - Анализ характеристик данных с помощью модуля “Specifications”.
    - Моделирование статистических зависимостей с помощью модуля “Dependencies”.
    - Обучение и тестирование модели SSA с помощью модуля “SSAmodel”.
    - Обучение и тестирование регрессионных моделей с помощью модуля “RegressionModel”.
- **print**: Этот метод используется для вывода текста в текстовое поле RichTextBox. Он принимает текст и флаг, указывающий, является ли текст заголовком, и выводит текст в текстовое поле.

Конструктор класса “Main” принимает объекты RichTextBox и CartesianChart, которые используются для вывода текста и построения графиков, а также список данных для анализа. Он считывает данные из файла Excel с помощью модуля “ExcelReader” и сглаживает данные с помощью функции Shared.SmoothData.

Все эти методы работают вместе, чтобы обеспечить полный процесс анализа данных и обучения моделей, начиная от считывания данных из файла Excel и заканчивая обучением и тестированием моделей для прогнозирования будущего количества заявок на кредиты. Этот модуль является важной частью программы, так как он координирует все другие модули и обеспечивает выполнение всего процесса анализа данных и обучения моделей.

4.3 Численное исследование результатов моделирования

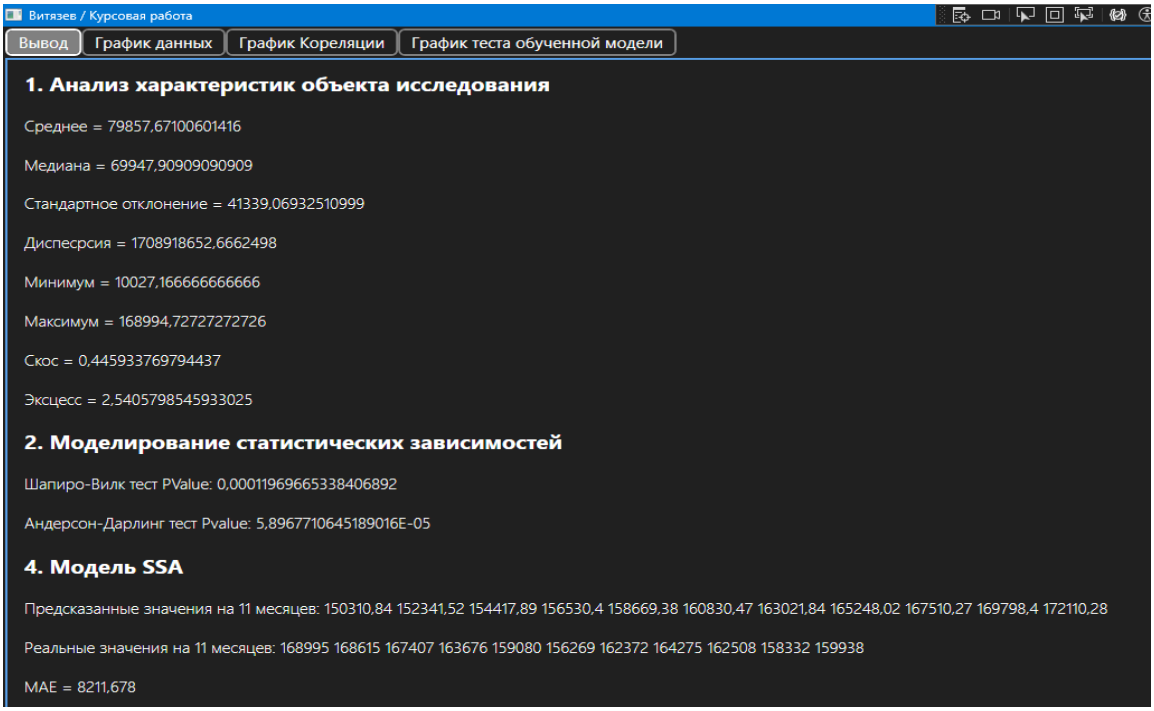
В ходе численного исследования результатов моделирования были получены следующие данные, используя модель SSA:

- Предсказанные значения на 11 месяцев: 150310,84 152341,52 154417,89 156530,4 158669,38 160830,47 163021,84 165248,02 167510,27 169798,4 172110,28
- Реальные значения на 11 месяцев: 168995 168615 167407 163676 159080 156269 162372 164275 162508 158332 159938

Для оценки точности предсказаний была использована метрика средней абсолютной ошибки (MAE). Она вычисляется как среднее значение абсолютных разностей между предсказанными и реальными значениями. Для каждого месяца была вычислена абсолютная ошибка, затем все абсолютные ошибки были усреднены, чтобы получить MAE.

В результате численного исследования было получено значение MAE = 8211,678. Учитывая, что данные измеряются в десятках тысяч, это можно считать приемлемым значением.

Также стоит отметить, что, несмотря на наличие ошибок в предсказаниях, общая динамика предсказанных значений соответствует динамике реальных данных. Это указывает на то, что модель успешно улавливает общие тренды в данных, что является важным фактором для эффективного прогнозирования.



«Вывод программы»