МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК

Кафедра математических методов в экономике

Ахматов Ариет Алымкулович

МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АНАЛИЗА И ПРОГНОЗА ОБЪЁМОВ ПРОДАЖ (НА ПРИМЕРЕ ДАННЫХ ПРОЕКТА «Kaggle»)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по направлению подготовки 01.03.04 Прикладная математика,
бакалаврская программа (профиль) «Математические методы в экономике»

г. Владивосток
2021

Автор работы _____
(подпись)

«_____» _____ 2021 г.

Руководитель ВКР
_зав. кафедрой к.ф.-м.н., доцент_____
(должность, уч. степень, уч. звание)

_____ А.С. Величко
(подпись)

«_____»_____2021 г.

Защищена в ГЭК с оценкой _____

Секретарь ГЭК

_____ _____А.В. Мишаков_____
подпись И.О. Фамилия

«_____» _____ 2021 г.

«Допустить к защите»

Зав. кафедрой _к.ф.-м.н., доцент
(уч. степень, уч. звание)

_____ А.С. Величко

«_____»_____ 2021 г.

УТВЕРЖДАЮ

Огнев А.В. /_____/
Ф.И.О. Подпись

И.о. директора Школы естественных наук_
Директор / наименование структурного подразделения

«_ __ _» _____ 2021 г.

В материалах данной выпускной квалификационной работы не содержатся сведения, составляющие государственную тайну, и сведения, подлежащие экспортному контролю.

Чеботарев А.Ю. /_____/
Ф.И.О. Подпись

Зав. кафедрой информатики, математического____
и компьютерного моделирования_____
Уполномоченный по экспортному контролю

«_ __ _» _____ 2021 г.

Оглавление

Введение

В настоящее время прогнозирование временных рядов — сложная задача, во многом обусловлена неожиданным изменениям в экономике и асимметрией информации. Из-за волатильности рынка в последнее время серьезную обеспокоенность вызывает качество прогнозов регрессионных моделей, которое необходимо тщательно оценивать [1, с. 15].

Применение методов машинного обучения к проблеме прогнозирование финансовых показателей в последние десятилетия активно развивается благодаря совокупности факторов: стремительному развитию технологии работы методов машинного обучения, развитию возможностей хранения и обработки большего объёма данных. В связи с этими тенденции по исследованию и применения различных методов машинного обучения к задаче прогнозирования является актуальной задачей.

Учитывая современные интерес к подобным задачам со стороны компаний, находящихся в мире быстро меняющегося рынка, исследователям требуется уметь грамотно анализировать наборы данных, понимать принцип работы методов машинного обучения и использовать эффективные методы прогнозирования, строить адекватные модели и оперативно обращать внимание на изменение на рынке для успешно. В данной работе рассмотрены и построены модели прогнозирования методом машинного обучения на примере набора данных из платформы Kaggle.

Целью выпускной квалификационной роботы является применение методов машинного обучения для анализа и прогнозирования временного ряда.

В соответствии с поставленной целью были сформулированы следующие задачи:

- Изучение теории прогнозирования временных рядов;
- Исследование методов машинного обучения для прогнозирования;
- Провести анализ набора данных из Kaggle;
- Построить модели прогноза объемов продаж;
- Провести сравнительный анализ моделей и сделать выводы по

проведенной работе.

Объектом исследования является набор данных «DT MART» полученные из платформы Kaggle, которые представляют данные об объёмах продаж различной техники: фотоаппаратов, аксессуаров/оборудования и игровых дисков.

Предметом является методы машинного обучения для анализа прогноза временных рядов.

Работа состоит из введения, трех глав, заключения, списка литературы и пяти приложений.

1 Теоретические основы прогнозирования временных рядов

Для реализации, поставленной задачи необходимо разобраться основными понятиями прогнозирования временных рядов и с постановкой задачи прогнозирование. Далее в рамках исследования моделей следует разобраться в методах прогнозирования, машинном обучении и оценки качества моделей.

1.1 Основные понятия

Один из наиболее важных элементов при принятии решений в государственном управлении и в частных областях бизнеса – это построение микроэкономических моделей для прогнозирования финансовых показателей. В течение последних десятилетий эконометрические модели с применением машинного обучения, направленные на прогнозирование, стали очень популярны.

Для лучшего понимания термина «Прогнозирование временных рядов» выделим два понятия:

- Временной ряд – это последовательность наблюдений, сделанных за некоторый период времени.
- Прогнозирование – составление научно обоснованного предсказания вероятностного развития событий в будущем [6].

Прогнозирование временного ряда составляется на основе исторических данных, полученных в прошлом. Например, составление прогноза погода, который позволяет предотвратить или снизить негативные последствия возможных природных явлений.

Временной ряд состоит из двух компонент: показатель времени и значения показателя ряда. Одна из классификаций временных рядов по показателям времени выделяет интервальные и моментальные ряды [3, 6].

Интервальный временной ряд – это последовательность значений показателя ряда, которая сформировалась за определенный интервал времени (например, количество продаж за месяц, неделю или день).

Моментальный временной ряд характеризует совокупность явлений в

определенный момент времени. Примерами могут быть последовательность финансовых индексов или температура окружающей среды.

Так же ряды разделяют на стационарные и нестационарные.

Стационарный ряд – ряд, который сохраняет свои значения относительно среднего уровня. Остальные ряды называются нестационарными. Нестационарные ряды часто приводятся к стационарным рядам при помощи разностного оператора.

Интервал времени, на который необходимо построить прогноз, называется временем упреждения. Задачи часто подразделяют по категории срочности на долгосрочные, среднесрочные и краткосрочные.

Kaggle – публичная веб-платформа, предназначенная для исследовательских проектов в области науки о данных. В системе размешен набор открытых данных различных коммерческих организаций. Отметим несколько разделов Kaggle:

- Datasets (наборы данных);
- Machine Learning Competitions (соревнования по машинному обучению);
- Learn (изучение);
- Discussion (обсуждения).

1.2 Постановка задачи прогнозирования

Последовательность действий (алгоритм), которая совершается для создания модели прогнозирования, называется *методом прогнозирования* [5, с. 54].

Модель прогнозирования – функциональное представление, валидно показывающее временной ряд и являющееся базой для получения будущих значений.

Пусть нам даны моменты времени $t = 1, 2, 3, \dots, T$. Обозначим временной ряд $f(t) = f(1), f(2), \dots, f(T)$.

Необходимо определить значение $f(t)$ в моменты времени $T + 1, T + 2, \dots, T + P$. Моменты времени $T + i$ называются моментами прогноза, а

величина P – временем упреждения. Для начала определим функциональную зависимость, которая показывает связь прошлых и будущих значений ряда.

$f(t) = Z(f(t-1), f(t-2) + \cdots) + \xi_t$ Данная зависимость называется моделью прогнозирования [5].

1.3 Методы прогнозирование

На данный момент можно насчитать более 100 классов методов [4]. Но их можно объединить в две основные группы:

- интуитивные;
- формализованные.

Интуитивные методы не предполагают создание прогнозирующей модели. Поэтому применяются в тех случаях, когда процесс очень простой, либо, наоборот, слишком сложный и невозможно описать влияние всех факторов математической моделью. К данному методу относят метод экспертных оценок, предвидения по образцу.

Формализованные методы прогнозирования выявляют зависимости в данных, что позволяет сделать математическую модель, для вычислений будущий значении процесса. Формализованные методы разделяют на статические и структурные модели.

К статическим методам относят модели, в которых зависимость между фактическим и будущим значениями описывается аналитически. К ним относят: регрессионные модели, авторегрессионные модели и модели экспоненциального сглаживания.

В структурных моделях зависимость задана структурно. Например: нейросетевые модели и модели на базе цепей Маркова [8].

В данной работе мы рассмотрим статистические методы и методы машинного обучения.

1.4 Машинное обучение

Машинное обучение — класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение в

процессе применения решений множества сходных задач.

Различают два типа обучения:

- Обучение по прецедентам, или индуктивное обучение, основано на выявлении эмпирических закономерностей в данных;
- Дедуктивное обучение предполагает формализацию знаний экспертов и их перенос в компьютер в виде базы знаний.

Дедуктивное обучение принято относить к области экспертных систем, поэтому термины машинное обучение тождественно обучению по прецедентам.

Также различают способы обучения:

- Обучение с учителем (Supervised learning);
- Обучение без учителя (Unsupervised learning);
- Обучение с частичным привлечением учителя (Semi-Supervised learning);
- Глубинное обучение (Deep learning).

В настоящее время термин «машинное обучение» означает совокупность алгоритмов, применяемые к данным для построения зависимостей между этими данными. В основном каждый из таких алгоритмов связан с конкретной математической моделью. Наиболее традиционным способом применения машинного обучения к задаче прогнозирование временных рядов является обучение с учителем (supervised learning). В качестве целевой переменной, значение которой предсказывается алгоритмом, выступает финансовый показатель за некоторый период времени [1, с. 16].

Следует отметить, что многие методы индуктивного обучения разрабатывались как альтернатива классическим статистическим подходам. Многие методы тесно связаны с извлечением информации и интеллектуальным анализом данных (data mining).

В данной работе для решения задачи прогнозирования мы рассмотрим модели, созданные на основе разных методов машинного обучения, иначе говоря, разных видов машинного обучения:

- классическое обучения;

– ансамбльное обучение;
– нейросети и глубокое обучение.

Классическое машинное обучение, строится на классических статистических алгоритмах и решает вопросы, связанные с принятием решений на основе данных. Задачами, которые решаются при помощи обучения с учителем, являются, например, классификация и регрессия.

Ансамблевые методы — это парадигма машинного обучения, где несколько моделей (часто называемых «слабыми учениками») обучаются для решения одной и той же проблемы и объединяются для получения лучших результатов. Основная гипотеза состоит в том, что при правильном сочетании слабых моделей мы можем получить более точные и/или надежные модели.

Нейронная сеть — это математическая модель, основанная на машинном обучении, представляется собой структуру, состоящую из искусственных нейронов, определенным образом связанных друг с другом и внешней средой с помощью связей, каждая из которых имеет определённый коэффициент, эти коэффициенты называют весами, они формируются в процессе обучения, который заключается в итеративной подстройке весов сети по некоторому правилу, называемому алгоритмом обучения.

1.5 Оценка качества моделей

Для оценки полученных прогнозов используем статистические оценки, иными словами оценку ошибки прогнозирования временного ряда.

Самый простой показатель — это отклонение факта от прогноза в количественном выражении. В практике рассчитывают ошибку прогнозирования по каждой отдельной позиции, а также рассчитывают среднюю ошибку прогнозирования. Следующие распространенные показатели ошибки относятся именно к показателям средних ошибок прогнозирования. К ним относятся:

Показатель RMSE (root mean square error) – среднеквадратичная ошибка используется для оценки качества предсказания модели. Он измеряет разницу

между истинным значение и предсказанным значениями. Задается формулой:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \widehat{x_i})}$$

Где N – общее число наблюдений; а $\sum_{i=1}^{N}(x_i - \widehat{x_i})$ представляет собой квадрат суммы разностей истинного значения и предсказания модели. Главным достоинством использование метрики RMSE является штрафование модели за большие ошибки.

Показатель MAPE (mean absolute percentage error) – средняя абсолютная процентная ошибка. Это коэффициент, не имеющий размерности, с очень простой интерпретацией. Его можно измерять в долях или процентах. Задается формулой:

$$MAPE = \frac{1}{N} \times \frac{\sum_{i=1}^{N}(x_i - \widehat{x_i})}{x_i} \times 100\%$$

где N – общее число наблюдений, x_i – фактическое значение временного ряда, а $\widehat{x_i}$ – прогнозное. Данная оценка применяется для временных рядов, фактические значения которых значительно больше 1. Если же фактические значения временного ряда близки к 0, то в знаменателе окажется очень маленькое число, что сделает значение MAPE близким к бесконечности – это не совсем корректно.

Но для понимания прогноз хороший или нет, следует провести сравнение с эталоном – постоянной моделью. Если неизвестно о характеристике входных данных, то и спрогнозированы нечего нельзя лучше среднего значения y. Затем можно сравнить среднеквадратичную погрешность нашей модели и нулевой модели. Эту идею формализует коэффициент детерминации.

Показатель R^2 (coefficient of determination) – коэффициент детерминации. Является одной из наиболее эффективных оценок адекватности регрессионной модели. Задается формулой:

$$R^2 = 1 - \frac{\sum_i (y_i - \widehat{y_i})^2}{\sum_i (y_i - \overline{y_i})^2} \approx 1 - \frac{MSE}{VAR(y)}$$

Здесь y_i – значение i-ого элемента, а $\widehat{y_i}$ – оценка этого элемента,

полученная с помощью регрессионной модели. Наконец $\overline{y_i}$ – среднее значение y, представляющее нулевую модель, которая всегда возвращает одно и тоже значение. Следовательно, для идеальной модели получится оценка 1, а для нулевой – оценка 0 [4, с. 171].

В данной работе будут использованы все три показателя, но следует отметить, что средняя абсолютная процентная ошибка и коэффициент детерминации наиболее просты для интерпретации результатов, будем основывается в исследовании на них.

2 Модели прогнозирования временных рядов. Статистические методы и методы машинного обучения

В данной главе проведен краткий анализ методов и моделей прогнозирования временных рядов. Кратко описана теоретическая основа и принципы работы статистических моделей, и методов машинного обучения.

2.1 Статистические методы. Метод Бокса-Дженкинса

2.1.1 Авторегрессионная модель

В основе авторегрессионной модели лежит предположение, что будущие значения линейно зависят от определенного количества предшествующих значений рассматриваемого процесса. При анализе временных рядов наиболее часто используемыми методами являются модель авторегрессии (autoregressive, AR) и модель скользящего среднего (moving average, MA) [3].

В первой модели значение процесса вычисляется как конечная линейная совокупность предшествующих значений с некоторым «белым шумом» [3, с. 141].

$$f(t) = C + \alpha_1 f(t-1) + \alpha_2 f(t-2) + .. + \alpha_p f(t-p) + \xi_t,$$

где C – вещественная константа, α- коэффициенты, ξ- ошибка модели.

Данная формула описывает процесс авторегрессии порядка p, которую в литературе обозначают как $AR(p)$. Параметры α и C находят методом наименьших квадратов или методом максимального правдоподобия.

Следующий метод используют часто вместе с предыдущим методом и называют моделью скользящего среднего, которую можно описать уравнением:

$$f(t) = \frac{1}{q}(f(t-1) + f(t-2) + .. + f(t-q) + \xi_t,$$

где q – порядок скользящего среднего, ξ_t- ошибка прогнозирования.

В литературе обозначают процесс как $MA(q)$ [2].

Для большей эффективности при подгонке модели часто объединяют эти два метода, и модель носит название авторегрессии проинтегрированного скользящего среднего – $ARIMA(p, d, q)$ (autoregression integrated moving average)

$$f(t) = AR(p) + \alpha_1 X_1(t) + .. + \alpha_s X_s(t)\ ,$$

где $\alpha_1, \dots, \alpha_s$— коэффициенты внешних факторов $X_1(t), \dots, X_s(t)$ [3, с. 151].

2.1.2 Алгоритм Бокса-Дженкинса

Эконометрические временные ряды за редким исключением не стационарны. Не стационарность чаще всего проявляется в наличии зависящей от времени неслучайной составляющей $f(x)$. Если случайный остаток, полученный вычитанием из исходного ряда его не случайной составляющее $f(x)$, представляет сбой стационарный временный ряд, то ряд называется нестационарным однородным [3, с. 156].

Для описание таких рядов используется модель авторегрессии проинтегрированного скользящего среднего (ARIMA-model). Также она известна как модель Бокса-Дженкинса. Модель ARIMA используется для описание временных рядов, обладающие следующими свойствами:

- ряд включает (аддитивно) составляющую $f(x)$, имеющий вид алгебраического полинома;
- ряд получившийся после применениям к нему процедур последовательных разностей, может быть описана медалью ARMA(p,q)

Наиболее распространены в эконометрических исследованиях модели ARIMA (p, d, q) со значениями параметров не превышающие 2. При этом параметры p и q переделяют соответственно порядок авторегресионной составляющей и порядок скользящего среднего (аналогично модели ()), а параметр порядок разностей (дискретной производной).

Методология построения ARIMA – модели включает следующие основные этапы:

- идентификация пробной модели;
- оценивание параметров модели и диагностическую проверку адекватности модели;
- использование модели для прогнозирования.

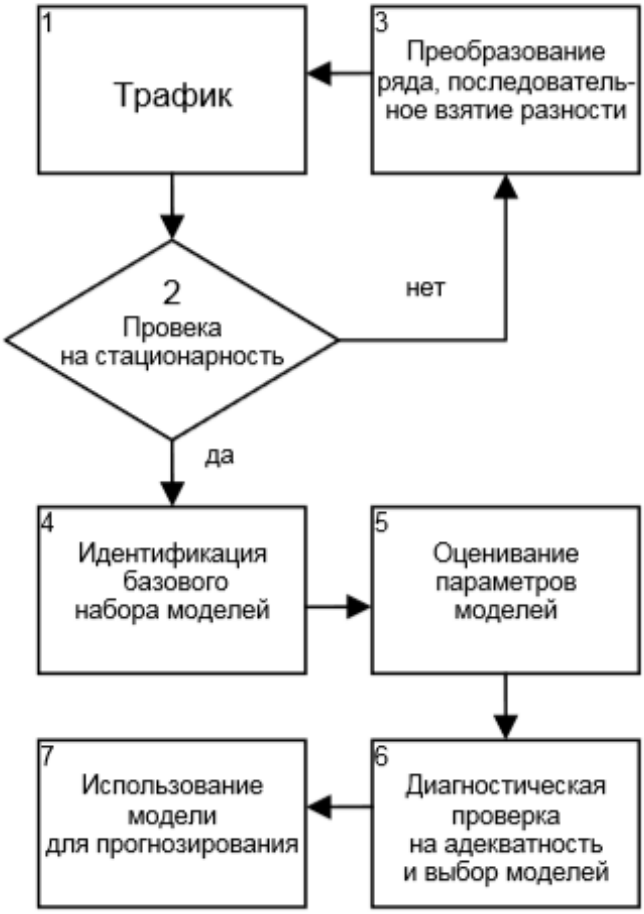

Рисунок 1 – Структурная схема подбора параметров модели ARIMA

Сначала в блоке 1, 2 необходимо получить стационарный ряд при тестировании исходных данных на стационарность прежде всего используется визуальный анализ графика. На этом этапе можно обнаружить ярко выраженную трудовую или сезонную составляющие.

В блоке 3 после получения стационарного ряда исследуется характер поведения выборочных ACF и PACF и выдвигаются гипотезы о значения параметров p (порядок авторегрессии) и q (порядок скользящего среднего). На выходе блока может формироваться базовый набор, включающий 1,2 или больше число моделей.

В блоке 4 после осуществления идентификации моделей необходимо оценить их параметры.

В блоке 5 для проверки каждой пробной модели на адекватность анализируется ее ряд остатков. У адекватной модели остатки должны быть

похожими на белый шум, т.е. их выборочные автокорреляции не должны существенно отличатся от нуля. Проверить значимости коэффициентов автокорреляции можно разными способами, например Q-статистику Бокса-Пирса или тестом Бокса-Льюнга. Если результаты проверки нескольких моделей оказываются адекватными исходным данным, то при оканчательной выборке следует два требования:

- повышение точности (качество подгонки модели);
- уменьшение числа параметров модели.

Воедино эти требования сведены в инфомациооно критерии Акайка (AIC), определяется формулой:

$$AIC = \frac{p+q}{n} + \ln\left(\frac{\sum_{t=1}^{n} e_t^2}{n}\right)$$

Где e_t - уровни ряда остатков.

Выбор следует сделать в пользу модели с меньшим значением AIC.

С помощью выбранной модели в блоке 6 можно строить точечный и интервальный прогноз на L шагов в перед.

Также сезонная модель Бокса-Дженкинса может быть представлена виде: SARIMA (p, d, q) (P_s, D_s, Q_s). Для оценки сезонных параметров требуется информация не менее за шесть полных периодов.

Успех применения мощного, гибкого, но в тоже время сложной модели ARIMA во многом зависит от практического опыта и квалификации исследователя, а процедуры автоматического выбора вида модели призвана облегчить аналитическую деятельность исследователя.

2.2 Методы машинного обучения

2.2.1 Модель Linear Regression

Линейная регрессия (Linear regression, LR) – самый простой метод регрессии. В нём используется линейный подход для моделирования отношений между зависимой (предсказанная) и независимой (предсказывающая) переменной. Одним из его достоинств является лёгкость интерпретации

результатов [7, с. 261].

Задача простой (однородной) линейной регрессии состоит в том, чтобы смоделировать связь между единственным признаком (объясняющей переменной x) и непрерывнозначным откликом (целевой переменной y). Уравнение линейной модели с одной объясняющей переменной определяется следующим образом:

$$y = \omega_0 + \omega_1 x$$

Здесь вес ω_0 – точка пересечения оси Y и ω_1 – коэффициент объясняющей переменной. Наша задача – извлечь веса линейного уравнении, чтобы описать связь между объясняющей и целевой переменными, которую затем можно использовать для предсказания откликов новых объясняющих переменных, не бывших частью тренировочного набора данных.

Основываясь на линейном уравнении, выше, линейная регрессия может понимать как нахождение оптимально прямой линии, проходящей через точки образцов данных, как показано на рисунке 2.

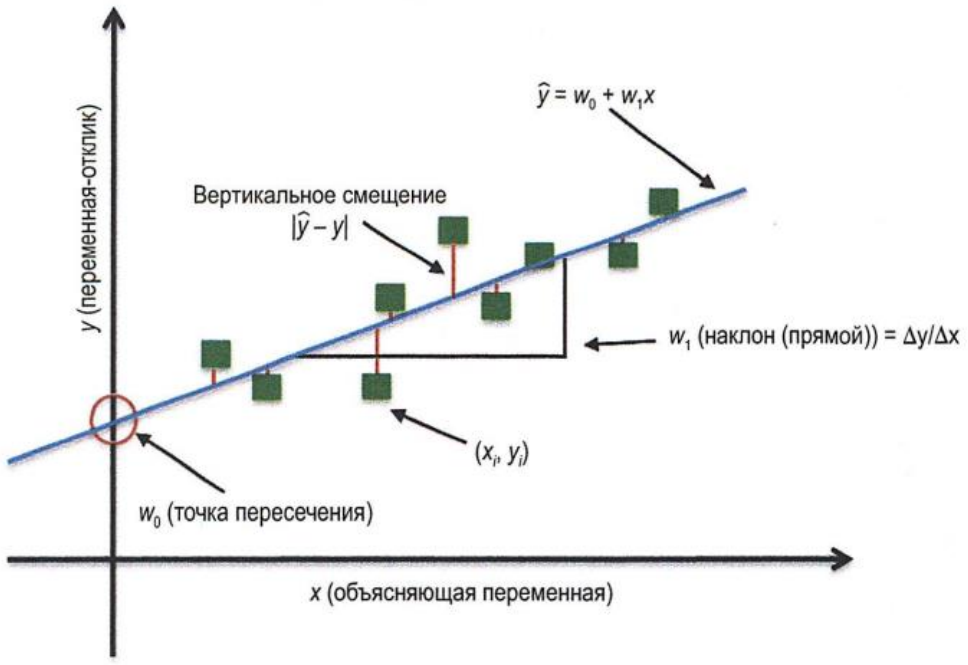

Рисунок 2 – График простой линейной регресси

Оптимальное подогнанная прямая линия также называется линией регресси, а вертикальные прямые от линии регресии до точек данных – так называемые смещенния, или остатки, – ошибки нашего предсказания.

Частный случай, состоящий из одной объясняющей переменной, назыается простой линейнной регресииией, но, мы также можем обобщить линейную регрессионную модель на две и более объяняющих переменных. Отсюда этот процесс называется множесванной линейнной регрессией[1]:

$$y = \omega_0 x_0 + \omega_1 x_1 + \cdots + \omega_m x_m = \sum_{i=0}^{m} \omega_i x_i = \omega^T x$$

Здесь ω_0 – точка пересечения оси Y при $x_0 = 1$.

[1]*Уравнение простой регрессии имеет на правой стороне точку пересечения (ω_0) и объясняющую переменную с коэффициентом наклона ($\omega_0 x$). В отличии от него уравнение множественной регрессии имеет на правой стороне две или более объясняющих переменных, каждая со своим коэффициентом наклона.*

2.2.2 Модель XGBoost

XGBoost (extreme gradient boosting) — алгоритм машинного обучения, основанный на дереве поиска решений и использующий фреймворк градиентного бустинга GBM [9, с 221].

Алгоритм GBM, основан на двух важных понятиях: *аддитивном расширении* и градиентной оптимизации алгоритмом *наискорейшего спуска*. Общая идея первого состоит в генерировании последовательности относительно простых деревьев (слабых учеников), где каждое следующее дерево добавляется вдоль градиента. Например, пусть имеются деревья M, которые агрегируют итоговые предсказания в ансамбле. Дерево в каждой итерации f_k теперь является частью намного более широкого пространства всех возможных деревьев в модели ($\emptyset$):

$$y = \sum_{m=1}^{M} f_k(x_i), f_k \in \phi$$

Новые деревья будут добавляться к предыдущим деревьям по принципу аддитивного расширения поэтапно:

$\hat{y}_i^0 = 0$;

$\hat{y}_i^1 = f_1(x_i) = \hat{y}_i^0 + f_1(x_i); -$ первое дерево

$\hat{y}_i^2 = f_1(x_i) + f_2(x_i) = \hat{y}_i^1 + f_2(x_i); -$ второе дерево добавляется к приведшему и т.д., пока не будет достигнут критерий остановки.

Предсказание ансамбля градиентного бустинга состоит из суммы предсказаний всех предыдущих деревьев и недавно добавленного дерева $\left(\hat{y}_i^{t-1}\right) + f_1(x_i)$, что в более формальном плане приводит к следующему:

$$\widehat{y_i^t} = \sum_{m=1}^{M} f_k(x_i) = \left(\hat{y}_i^{t-1}\right) + f_i(x_i).$$

Вторая важная и при этом довольно хитроумная часть алгоритма GBM – это градиентная оптимизация наискорейшим спуском. Иными словами, в аддитивную модель мы добавляем все более мощные деревья. Это достигается путем применения к новым деревьям градиентной оптимизации. Каким образом выполняются обновления градиента с деревьями? Прежде всего мы параметрзуем деревья; мы делаем это путем рекурсивного обновления значений расщепления узлов вдоль градиента, где узлы представлены вектором. Вследствие этого направление наискорейшего спуска является отрицательным градиентом функции потерь, и расщепления узлов будут обновляться и обучаться, приводя к:

λ: параметру сжатия (в данном контексте также именуемом темпом обучения), который заставит ансамбль обучаться медленно с добавлением большего количества деревьев;

γ_{mi}: параметру обновления градиента, также именуемому длиной шага.

Прогнозная отметка для каждого листа тем самым является итоговой отметкой для нового дерева, которая, в свою очередь, является результатом простого суммирования по каждому листу:

$$\widehat{y_i^t} = \sum_{m=1}^{M} f_k(x_i) = \left(\hat{y}_i^{t-1}\right) + \lambda\gamma_{mi} f_i(x_i).$$

Таким образом, если резюмировать, алгоритм GBM работает путем инкрементного добавления более точных деревьев, усвоенных вдоль градиента.

Расширяя алгоритм GBM, алгоритм экстремального градиентного бустинга XGBoost добавляет больше масштабируемых методов, которые задействуют многопоточность на одиночной машине и параллельную обработку на кластерах из многочисленных серверов (используя сегментирование) [9, с. 215].

XGBoost – это новое поколение алгоритмов градиентного бустинга GBM с серьезной доводкой исходного алгоритма бустинга деревьев GBM. Алгоритм XGBoost обеспечивает параллельную обработку; предлагаемая алгоритмом масштабируемость реализуется благодаря доработанным авторами нескольким параметрическим настройкам и добавлениям:

- *алгоритм принимает разреженные данные, в которых могут задействоваться разреженные матрицы, экономя оперативную память (отсутствует потребность в плотных матрицах) и продолжительность вычисления (нулевые значения обрабатываются особым образом);*
- *обучение приближенному дереву (взвешенный метод квантильной схемы), которое показывает аналогичные результаты, но за гораздо меньшее время, чем классический исчерпывающий просмотр возможных точек ветвления;*
- *параллельные вычисления на одиночной машине (используя многопоточность в фазе поиска лучшего расщепления) и аналогичным образом распределенные вычисления на нескольких машинах;*
- *внеядерные вычисления на одиночной машине с привлечением решения для хранения данных под названием «постолбцовый блок» (column block), которое располагает данные на диске столбцами, тем самым экономя время – данные с диска поступают в том виде, в котором их ожидает алгоритм оптимизации (который оперирует векторами-столбцами)*.

2.2.3 Модель LSTM

Долгая краткосрочная память (long short-term memory, LSTM) — тип рекуррентной нейронной сети, способный обучаться долгосрочным зависимостям. LSTM были представлены в работе [Hochreiter & Schmidhuber

(1997)], впоследствии усовершенствованы и популяризированы другими исследователями, хорошо справляются со многими задачами и до сих пор широко применяются. LSTM специально разработаны для устранения проблемы долгосрочной зависимости. Их специализация — запоминание информации в течение длительных периодов времени, поэтому их практически не нужно обучать.

Архитектуры RNN-сетей с долгой краткосрочной памятью (LSTM) имеют более сложные элементы, которые поддерживают внутреннее состояние и содержат вентили для отслеживания зависимостей между элементами входной последовательности и соответствующим образом регулируют состояния ячеек. Эти вентили соединяются рекуррентно друг с другом вместо обычных скрытых элементов. Они призваны решать проблемы исчезающих и взрывающихся градиентов, позволяя градиентам проходить без изменений. На рисунке 3 показаны развернутые элементы LSTM и их типичный вентильный механизм.

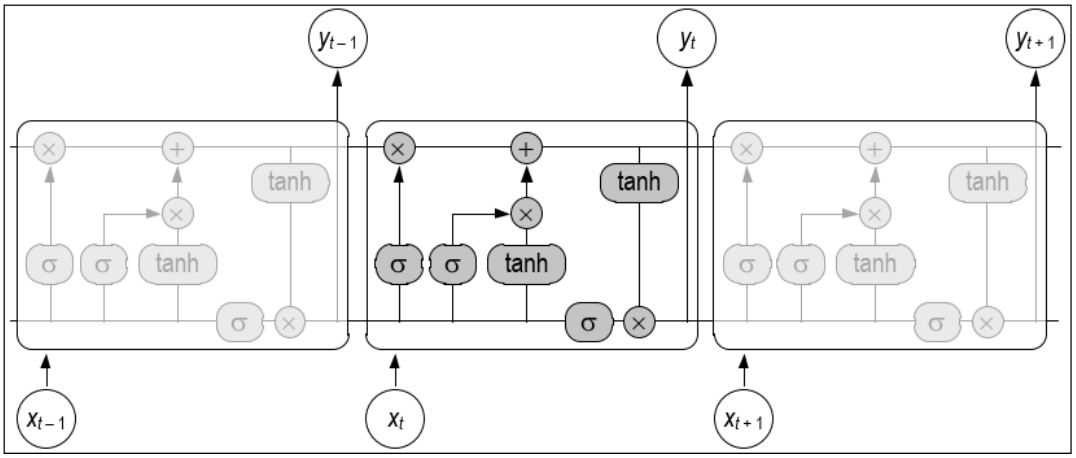

Рисунок 3. Развернутые элементы LSTM и их типичный вентильный механизм

Типичный элемент LSTM объединяет четыре параметризованных слоя, которые взаимодействуют друг с другом и состоянием ячейки путем преобразования и передачи векторов. Эти слои обычно предусматривают входной вентиль, выходной вентиль и вентиль забывания; но есть варианты, которые могут иметь дополнительные вентили или не иметь некоторых из этих механизмов. Светло-серые узлы на рисунке 4 отождествляют поэлементные

операции, а более темные серые элементы представляют слои с параметрами весов и смещения, усваиваемыми во время тренировки [10, с. 77].

Состояние ячейки с проходит вдоль горизонтального соединения в верхней части ячейки. Взаимодействие состояния клетки с различными вентилями приводит к ряду рекуррентных решений, таких как:

1. Вентиль забывания следит за тем, сколько состояний ячеек должно быть опустошено. Он получает в качестве входов предыдущее скрытое состояние h_{t-1} и текущий вход x_t, вычисляет сигмоидальную активацию и умножает результирующее значение f_t в интервале [0; 1] с состоянием ячейки, соответственно снижая или сохраняя его.

2. Входной вентиль также вычисляет сигмоидальную активацию i из h_{t-1} и x_t, которые производят кандидатов на обновление. Активация tanh в интервале [–1; 1] умножает кандидатов на обновление u_t и, в зависимости от результирующего знака, добавляет или вычитает результат из состояния ячейки.

3. Выходной вентиль фильтрует обновленное состояние ячейки с помощью сигмоидальной активации o_t и умножает его на состояние ячейки, нормализованное в интервал [–1; 1] с помощью активации tanh.

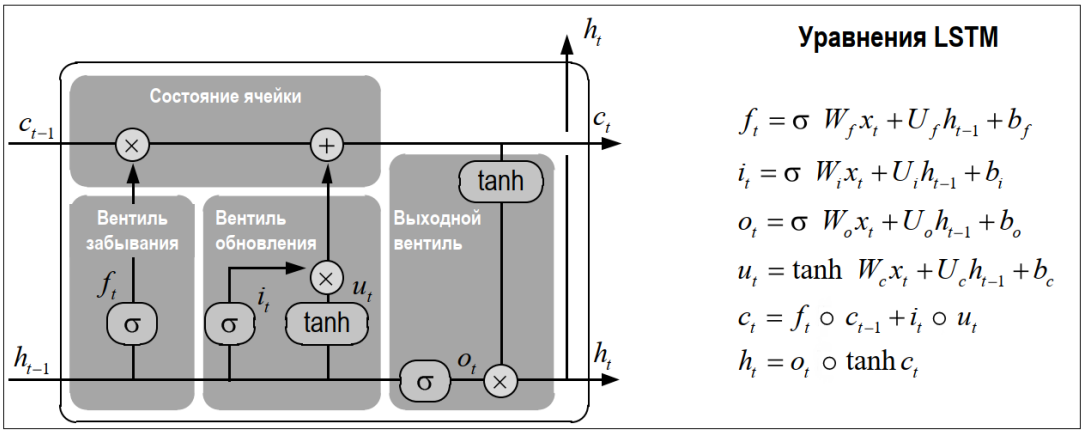

Рисунок 4. Типичный элемент LSTM

3 Построение моделей прогнозирования на примере данных Kaggle

В данной главе будет произведен анализ набора данных и построены модели, на основе методов описанных ранее. Производится сравнительный анализ методов (алгоритмов) машинного обучения для задачи прогнозирования временных рядов. Анализ также будет проводиться на реальных данных взяты с платформы Kaggle.

3.1 Описание набора данных

Набор данных «DT MART» размещен на платформе Kaggle.com и приставляет собой архив csv-файлов, в которых отражена информация разного рода, в том чисел о продажах *Sales.csv* [11]. Исходный набор данных о продажах изображён на рисунке 5. Полный набор данных временного ряда расположен в Приложении А.

CSV-файл о продажах содержит следующий признаки (характеристики):

ID: Уникальный идентификатор каждого товара.

ID_Order: Уникальный идентификационный номер каждого заказа.

ID_Item_ ordered: предположим, вы заказываете 2 разных товара в одном заказе, он генерирует 2 разных идентификатора элемента заказа для одного идентификатора заказа; заказы отслеживаются по идентификатору позиции заказа;

GMV: валовая стоимость товаров (общая денежная стоимость продажи товара);

Units_sold: количество проданных товара;

SLA: количество дней, которое требуется для доставки товара;

MRP: максимальная розничная цена (рассчитываемая производителем цена, которая представляет собой самую высокую цену, которая может взиматься за товар);

Procurement SLA: время, потраченное на закупку товара.

```
# загрузка dataset
db = pd.read_csv('Данные/DT_MART/Sales.csv','\t', parse_dates=['Date'], index_col='Date', dayfirst=True, low_memory = False) #1,048,575
```

```
df2.head(10)
```

Date	ID	ID_Order	ID_Item_ordered	GMV	Units_sold	SLA	Product_Category	Analytic_Category	Sub_category	product_analytic_vertical	MRP	Procurement_SLA
2015-08-30 13:17:00	ACCE7U2GBCJKQZZV	1.410000e+15	1.410000e+15	285	1	16	CE	CameraAccessory	CameraAccessory	Lens	395	3
2015-09-14 16:06:00	AUDEAFDA5GGESPWX	1.400000e+15	1.400000e+15	299	1	8	CE	EntertainmentSmall	AudioMP3Player	AudioMP3Player	999	1
2015-09-15 13:06:00	ACCDH7MFZJGRWJCW	3.410000e+15	3.410000e+15	284	1	5	CE	GamingHardware	GamingAccessory	GamePad	410	1
2015-09-22 11:26:00	VGLE7AUXCYJWGBRV	3.410000e+15	3.410000e+15	419	1	9	CE	EntertainmentSmall	TVVideoSmall	VideoGlasses	349	5
2015-09-22 17:17:00	REME69QDWFWZGUMB	3.410000e+15	3.410000e+15	490	1	8	CE	EntertainmentSmall	TVVideoSmall	RemoteControl	999	5
2015-09-23 13:42:00	ACCDYGCYHMNJTD3N	1.410000e+15	1.410000e+15	479	1	7	CE	EntertainmentSmall	Speaker	LaptopSpeaker	600	3
2015-09-23 17:11:00	ACCEB3YGZQM3F3UQ	4.410000e+15	4.410000e+15	649	1	5	CE	EntertainmentSmall	Speaker	MobileSpeaker	999	2
2015-09-24 13:29:00	SELDGZDDMHSHZWFF	4.410000e+15	4.410000e+15	2850	1	5	CE	EntertainmentSmall	TVVideoSmall	SelectorBox	3100	2
2015-09-25 21:10:00	ACCDJKUFZKJUGHNZ	4.410000e+15	4.410000e+15	345	1	8	CE	EntertainmentSmall	Speaker	LaptopSpeaker	650	2
2015-09-25 21:10:00	ACCE4T8WEYFGWFJU	1.410000e+15	1.410000e+15	48595	1	10	CE	EntertainmentSmall	HomeAudio	HomeAudioSpeaker	48595	3

Рисунок 5 – Набор данных о продажах

Анализ набора данных о продажах

Для построения математической модели необходимо проанализировать ключевые характеристики данных о продажах на основе исторических данных на 2015–2016 год.

Соберем необходимую информацию о продажах, из набора данных «DT MART», за период с 2015 по 2016 год, размещенных в базе Kaggle dataset. Нас интересует следующая информация о продажах:

- *Дата транзакции (Date)*
- *ID товара (ID)*
- *Категория товара (Category)*
- *Название товара (Brand)*
- *Валовая выручка (валовая стоимость товара (GMV)*
- *Предельный продукт (максимальная розничная цена) (MRP)*
- *Количество проданного товара (Units)*
- *Количество дней затраченного на доставку товара (SLA)*

Разобьем работу с данными на несколько этапов:

1. Загрузка исходного csv файла в программу и предварительная обработка данных;
2. Разведочный анализ набора данных о продажах;
3. Построение моделей прогнозирования;
4. Анализ и оценка моделей прогнозирования;

Загрузка и предварительная обработка данных

Для загрузки и обработки данных воспользуемся языком программирования Python 3 и основными библиотеками для работы с данными: NumPy, Pandas и Matplotlib. Очистим наши данные от лишней информации:

1. Выделим в данных о продажах, атрибут GMV, возьмем ее значение как целевую переменной y которую хотим предсказать.
2. В наборе данных уберем операции, в которых не хватает необходимой для нас информации.
3. Удалим случайные выбросы, которые связаны с некорректными данными.
4. Сгладим ряд. От исходной периодичности наблюдений перейдем к периодичности «один час».
5. Для построения моделей машинного обучения произведем разделение данных на тренировочный (train) 60%, проверочный (cv) 20% и тестовый (test) 30% набор без перемешиваний.

Тренировочный набор данных был использован для обучения модели, проверочный для того, чтобы настроить параметры и избежать переобучение модели, а тестовый — для оценки качества ее обучения.

3.2 Характеристика набора данных

Визуализируем прогнозируемый временной ряд объемов продаж.

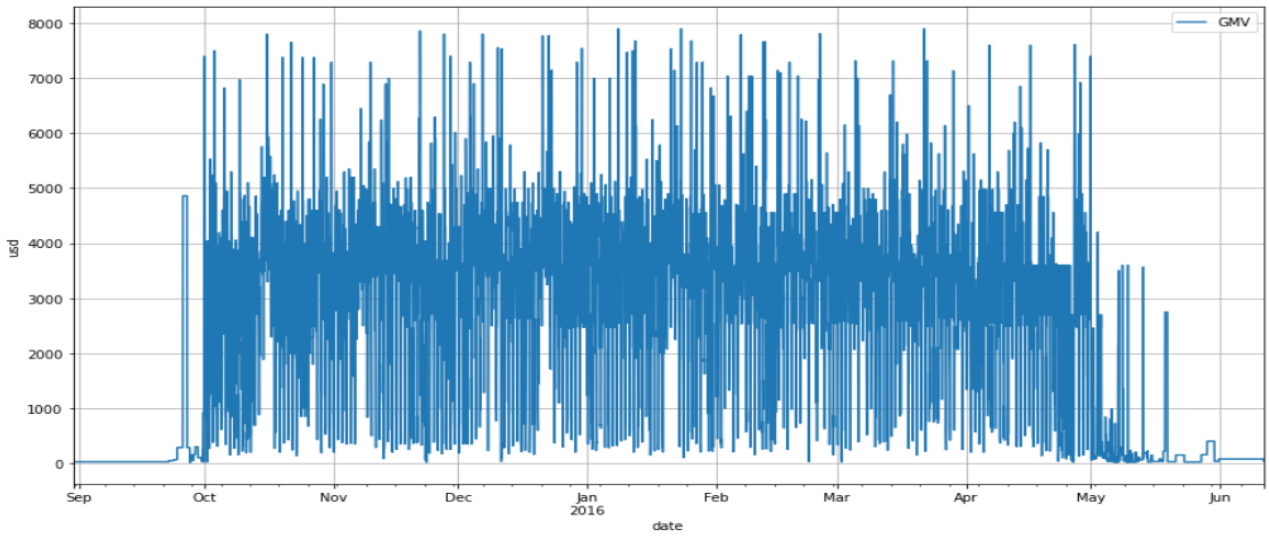

Рисунок 6 – Объемы продаж продукции с 2015–2016 г. ед. изм. в $.

Определим основные характеристики временного ряда. Мы можем заметить, что у нас относительно большой разброс ряда. На рисунке 7 изображена описательная статистика временного ряда. Рассчитаем коэффициент вариации для ряда. V = 0.719664, где V – коэффициент вариации объемов продаж.

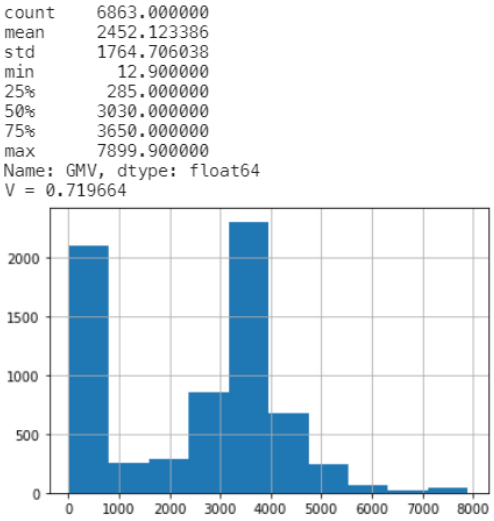

Рисунок 7 – Гистограмма и статистика временного ряда GMV

Если коэффициент вариации превышает 33%, то это говорит о неоднородности информации и необходимости исключения самых больших и самых маленьких значений.

Исходя из визуализации графика, можем сделать следующие предварительные выводы:

1. Ряд не имеют определенного тренда.
2. Ряд не имеют определенный период.
3. Ряд не имеют сезонную составляющую.
4. Необходимо снизить коэффициент вариаций.

Создадим матрицу точечных графиков, позволяющую в одном месте визуализировать в этом наборе данных попарные корреляции между разными признаками, также рассчитаем линейные коэффициенты корреляции Пирсона. Для подготовки матрицы точечных графиков напишем функцию `def plotScatterMatrix()`. Полный программный код расположен в приложении Б.

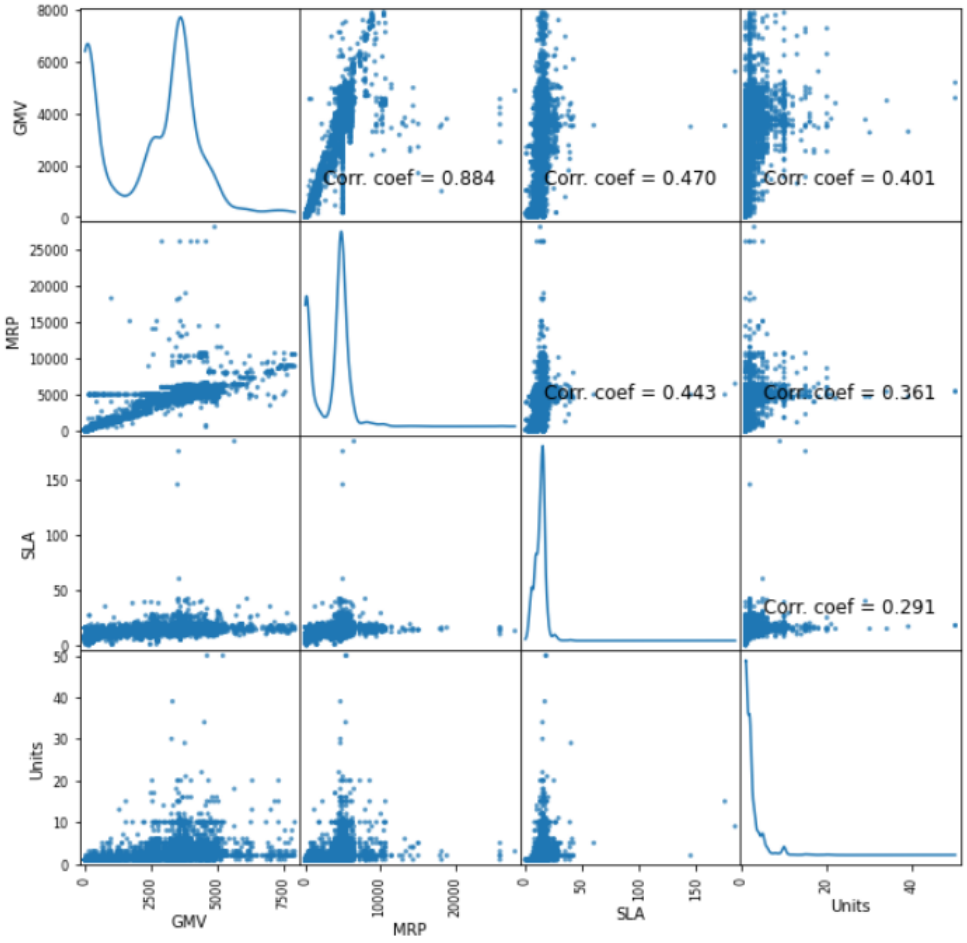

Рисунок 8 – Матрица корреляций набора данных о продажах

3.3 Построение статистических моделей прогнозирования

3.3.1 Метод скользящей средней и взвешенной средней

Для начала обратимся к самому простому методу, предположив, что последующие значения будут такие же, как предшествующие, т.е. «завтра будет похоже на вчера». Данный метод называется методом скользящей средней. Во второй главе были разобраны теоретические аспекты данной модели. Реализуем эту же функцию в Python 3 и посмотрим на прогноз, построенный по последнему наблюдаемому дню (24 часа). Приведу фрагмент кода программы (рисунок 9). Полный программный код представлен в Приложении Б.

```
def moving_average(series, n):          # M
    return np.average(series[-n:])
print(moving_average(df.GMV[5147:5868], 24))
```

```
3082.0916666666667
```

Рисунок 9 – Метод скользящей средней за последний день

Минусом данного метода является сложность построения прогноза на длительный период, так как предшествующее значение должно наблюдаться фактически. Однако плюсом данной модели является возможность выявления тренда при помощи сглаживания рядов. Построим графики для объема продаж.

На рисунке 10 представлен график сглаженного ряда с интервалом в месяц. Зеленным цветом выделена скользящая средняя, которая показывает тенденции продаж. График синего цвета соответствует актуальному объему продаж. А красный пунктир отображает доверительный интервал +/-1.96 для сглаженного значения. Из сглаженного ряда на рисунке видно, что 4-недельный интервал хорошо отражает общие изменения, связанные с резким ростом и последующим падением продаж в интервале каждого дня, в промежутке октября и в конце ноября. Объем продаж держатся в диапазоне сумм 1000-6000$.

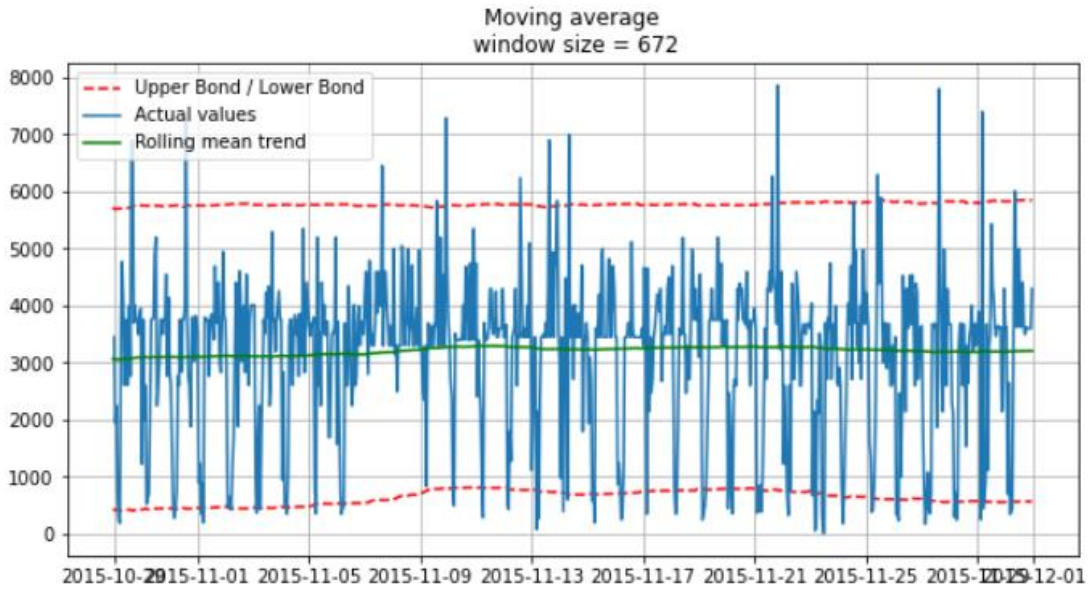

Рисунок 10 – График метода скользящей средней

Ранее было упомянуто про модификацию данного метода, который называется «метод взвешенной средней». Метод взвешенной средней отличается от метода скользящей средней тем, что в процессе наблюдения данным присваивают определенные веса, сумма которых равна единице, как правило, новым наблюдениям дают больший вес.

Реализуем метод взвешенной средней на языке программирования Python и посмотрим на прогноз. На рисунке 11 представлен фрагмент программы

построения прогноза для объемов продаж.

```
#Метод взвешенной средней
def weighted_average(series, weights):
    result = 0.0
    weights.reverse()
    for n in range(len(weights)):
        result += series[-n-1] * weights[n]
    return result

print(weighted_average(df.GMV[5147:5868], [0.8, 0.1, 0.05, 0.04, 0.01]))
```

```
3682.174
```

Рисунок 11 – Метод взвешенной средней за последний месяц

К сожалению, предыдущие методы способны строить прогнозы только на один шаг вперед, что является недостаточным для потребностей современного мира. Поэтому построим модель объединением этих двух методов, моделью авторегрессии проинтегрированного скользящего среднего (ARIMA).

3.3.2 Модель ARIMA. Метод Бокса-Дженкинса

Ранее мы говорили о такой характеристике, как стационарность. Ее важность обоснована тем, что по стационарному ряду легко строить прогноз, так как характеристики в будущем не будут отличаться от текущих наблюдений. Многие модели временных рядов могут моделировать и предсказывать данные характеристики, например, математическое ожидание или дисперсию. Однако если ряд не стационарен, то предположения будут неверны. И на практике большинство реальных моделей не являются стационарными.

Для проверки на стационарность существуют различные способы – разности различных порядков, выделение тренда или сезонности.

Поэтому при построении модели эконометрическим способом необходимо определить характеристики ряда. Всю схему работы с моделью ARIMA можно описать алгоритмом, который мы описали ранее в разделе методологии Бокса-Дженкинса.

Применим критерий Дики-Фуллера к нашей модели и представим ее графически.

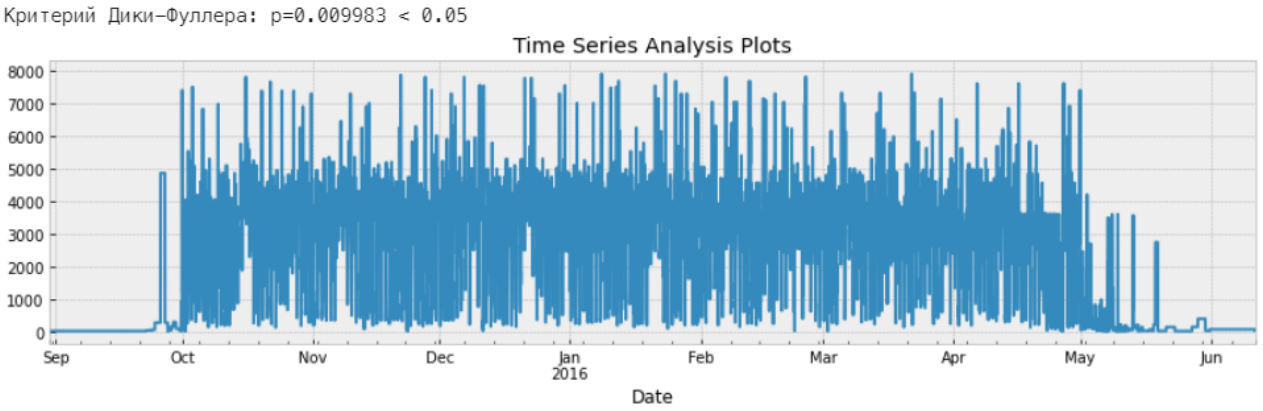

Рисунок 12 – Проверка критерия Дики-Фуллера

Видно, что ряд не является стационарным, и критерий Дики-Фуллера отверг нулевую гипотезу о наличии единичного корня.

Следующим шагом будет стабилизация дисперсии преобразованием Бокса-Кокса.

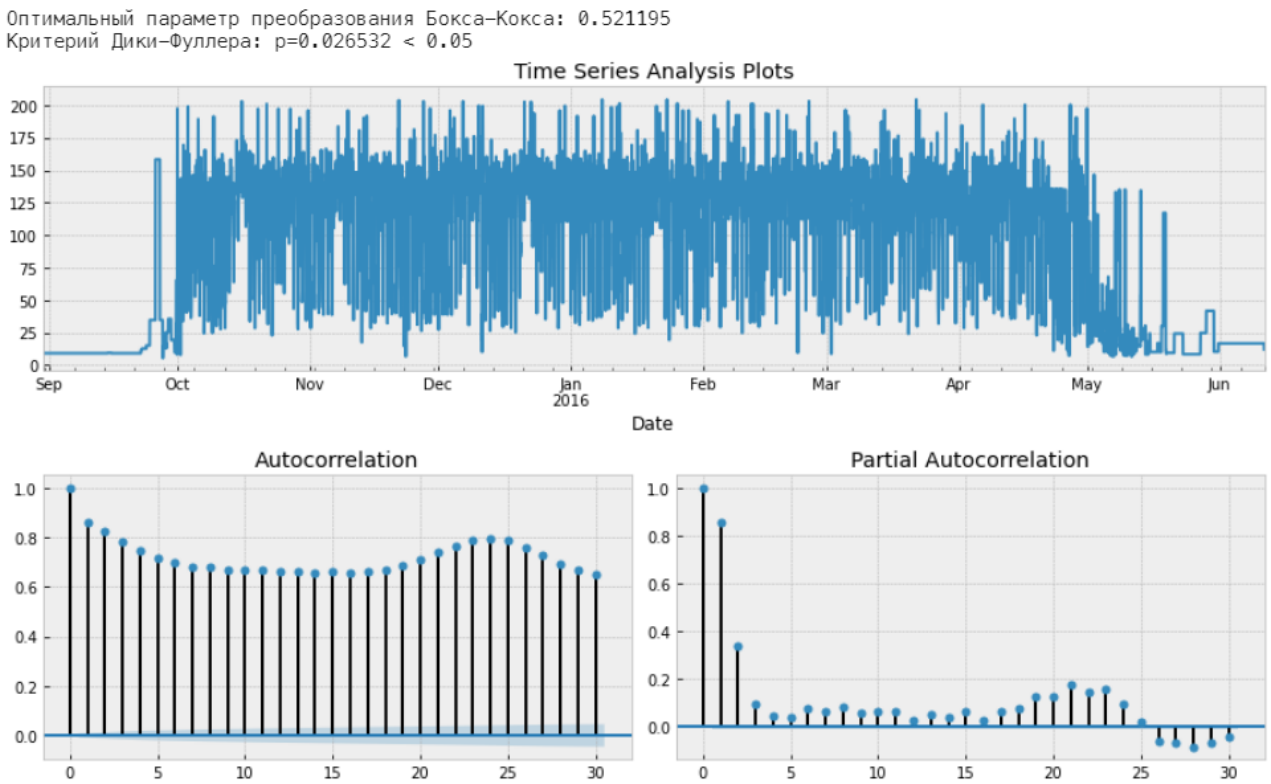

Рисунок 13 – Преобразование Бокса-Кокса

На графике 13 изображены временной ряд объемов продаж, автокорреляции (ACF) и частичной автокорреляции (PACF). Автокорреляционная функция всё ещё выглядит нехорошо из-за большого числа значимых лагов. График коррелограммы говорит о том, что существует сезонность в получившемся ряде. Возьмём сезонные разности ряда.

В итоге получили стационарный ряд по автокорреляционной и частной автокорреляционной функции (рисунок 14).

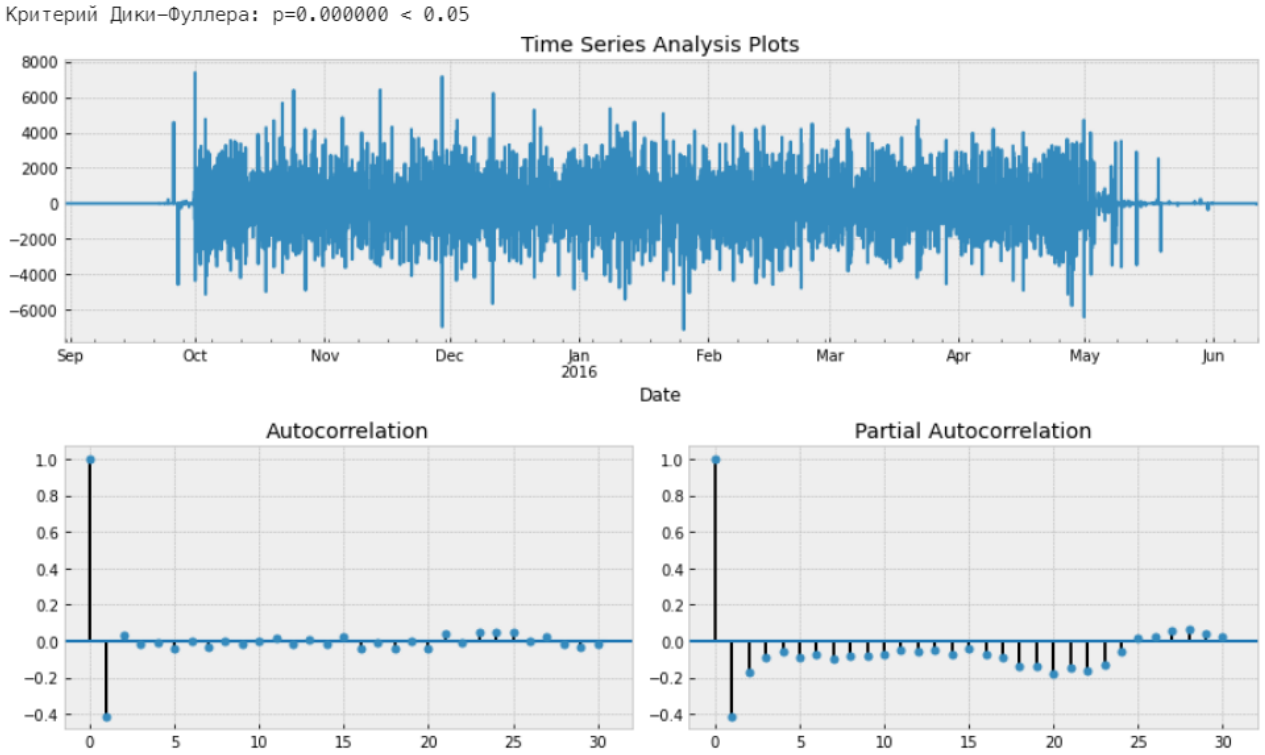

Рисунок 14 – Приведенный ряд к стационарному виду

Теперь нужно подобрать параметры для модели ARIMA.

Для начала используем начальные приближения, равные Q=3, P=4, q=1, p=1.

С помощью автоматического подбора найдем лучшие значения. В первой главе мы кратко коснулись того, как выглядит модель.

AR (p) – модель авторегрессии. Основное предположение – текущие значения ряда зависят от предыдущих значений с некоторым запаздыванием (или несколькими запаздываниями). Максимальная задержка в модели называется p. Чтобы определить начальное значение p, вам нужно взглянуть на график PACF – найти самый большой значительный лаг, после которого большинство других лагов становятся незначительными.

MA (q) – модель скользящего среднего. Моделирует погрешность временного ряда, предполагая, что текущая погрешность зависит от предыдущего с некоторым запаздыванием, которое называется q. Начальное значение можно найти на графике ACF с той же логикой.

AR (p) + MA (q) = ARMA (p, q)

У нас есть модель авторегрессии-скользящего среднего. Если ряд стационарный, его можно аппроксимировать с помощью ARMA (p, q).

Введем I (d) – порядок интеграции. Это просто количество несезонных различий, необходимых для того, чтобы сделать серию неподвижной. Мы получаем модель ARIMA, которая знает, как обрабатывать нестационарные данные с помощью несезонных различий [2, с. 156].

Теперь, зная, как установить начальные параметры, можно попробовать аналитическим методом установить приблизительные параметры модели, взглянув еще раз на графики рисунка 14.

Для точности прогноза воспользуемся функцией математического подбора наилучших параметров. Расчетные значения параметров, следующие:

order (p = 2, d = 1, q =1);

seasonal order (P =0, D = 1, q = 1).

Подставим значения параметров в модель SARIMA. Выедем информацию по получившийся модели (рисунок 15).

SARIMAX Results

Dep. Variable:	GMV	No. Observations:	6863
Model:	SARIMAX(2, 1, 1)x(0, 1, 1, 24)	Log Likelihood	-31196.650
Date:	Wed, 16 Jun 2021	AIC	62403.300
Time:	18:06:16	BIC	62437.451
Sample:	08-30-2015	HQIC	62415.082
	- 06-11-2016		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.2079	0.008	25.157	0.000	0.192	0.224
ar.L2	0.1354	0.009	15.019	0.000	0.118	0.153
ma.L1	-0.9237	0.004	-215.449	0.000	-0.932	-0.915
ma.S.L24	-0.9400	0.004	-259.936	0.000	-0.947	-0.933
sigma2	533.1671	5.558	95.931	0.000	522.274	544.060

Ljung-Box (Q):	90.55	Jarque-Bera (JB):	3981.03
Prob(Q):	0.00	Prob(JB):	0.00
Heteroskedasticity (H):	0.86	Skew:	0.04
Prob(H) (two-sided):	0.00	Kurtosis:	6.74

Рисунок 15 – Параметры для модели SARIMA

Проверка модели на адекватность

Проверим модели на соответствие стационарности, а также проанализируем коррелограммы, отражающие поведение теоретической автокорреляционной и частной автокорреляционной функций. Это может нам

помочь в определении важной информации для прогнозирования элементов регрессии (рисунок 16).

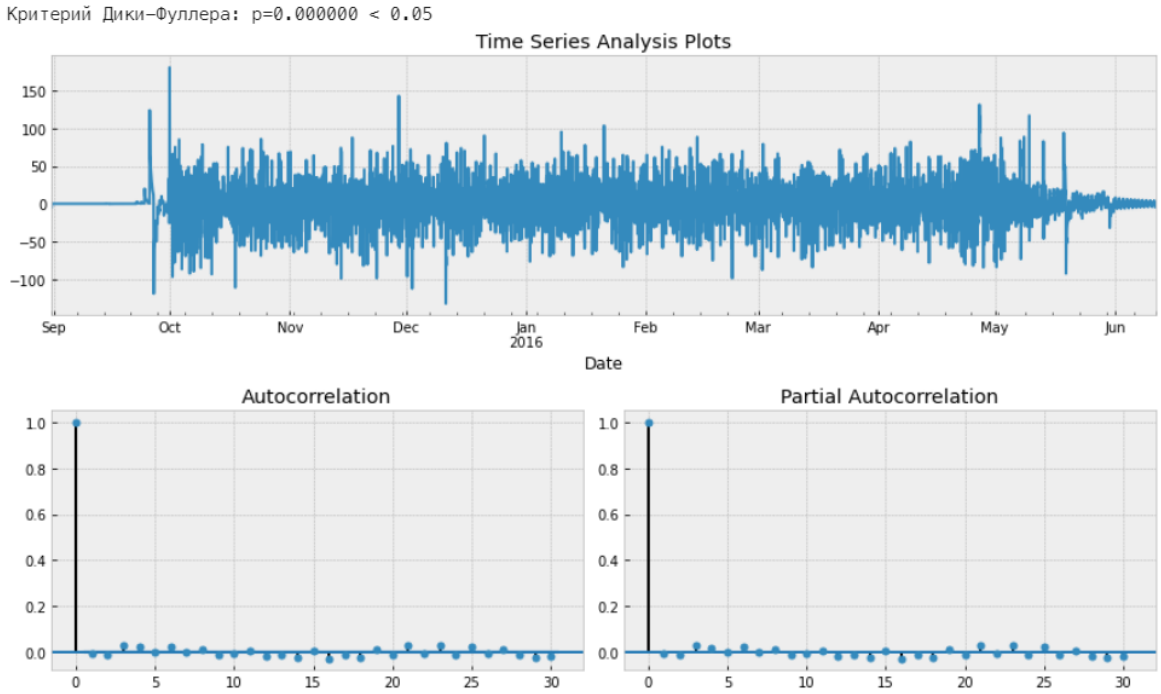

Рисунок 16 – Анализ коррелограммы на адекватность

Из рисунка 16 видно, что ряд стационарен, явных автокорреляций нет, и нормально распределены, следовательно можно строить прогноз по получившейся модели.

Осталось подставить наши параметры на основе выбранной модели SARIMA (2,1,1) (0, 1, 1) и построить график прогноза. На рисунке 17 изображены исходные данные (синий цвет) и прогнозные значение (красная цвет).

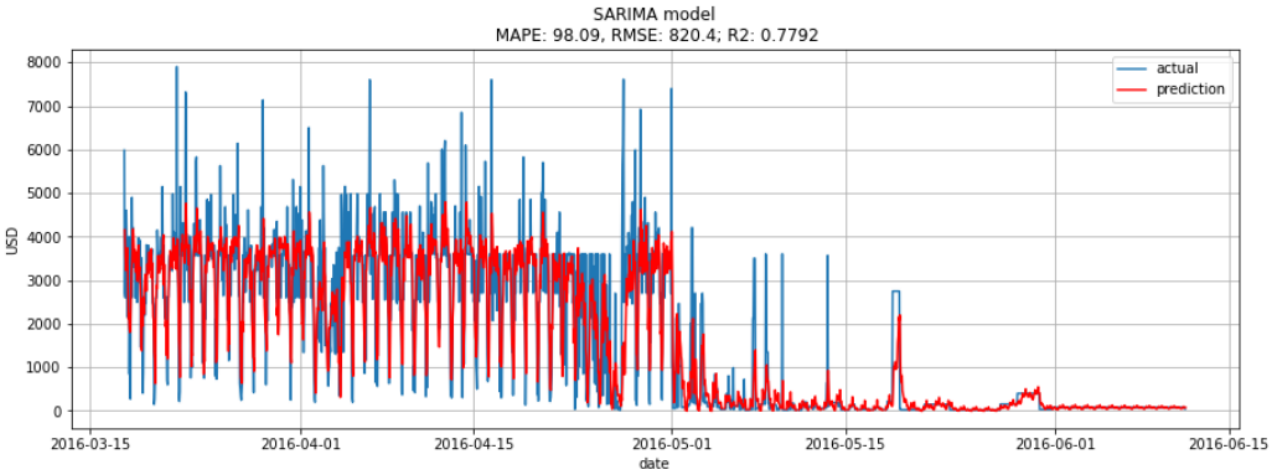

Рисунок 17 – Прогноз и исходные значения объемов продаж. Модель SARIMA

Таблица 1 – Результаты оценки модели SARIMA

Название модели	MAPE	RMSE	R^2
SARIMA	98.09	820.4	0.7792

В финале мы получаем достаточно адекватный прогноз, с коэффициентом детерминации (R^2) равным 0.779, что очень и очень неплохо, однако в модели довольно большая разность между истинными и прогнозным значениями, что видно из среднеквадратичной ошибки (RMSE). Но тем не менее суммарные затраты на подготовку данных, приведение к стационарности, определение и перебор параметров могут не оправдать затраченные усилия, для этого далее построим модели машинного обучения, и сравним их результаты.

3.4 Построение моделей машинного обучения

3.4.1 Модель линейной регрессии, МНК

При создании модели воспользуемся обычным методом наименьших квадратов (OLS), он реализован в библиотеке `scikit-learn`. МНК применяется для оценки параметров линейной регрессии, минимизирующих сумму квадратичных вертикальных расстояний (остатков или ошибок) до точек образцов.

Начнем с реализацию линейной регрессии, то есть попытаемся предсказать объемы продаж (GMV) по признакам: MRP, SLA, Units, range_mg. Но этих признаков недостаточно для построения хорошей модели. Линейная регрессия - линейный подход к моделированию зависимости целевой переменной от одной или нескольких объясняющих переменных. Способ, который мы реализуем, для построения качественной модели, заключается в том, что мы подгоним LR модель к предыдущим N значениям, и используем эту модель для прогнозирования значения на текущий день. Для этого реализуем функцию `get_preds_lr` полный программный код представлен в Приложении В.

Применим стандартную регрессию методом наименьших квадратов:

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(fit_intercept = True)
```

Импортируем класс `LinearRegression` из `sklearn` модуля и конструируем

объект этого класса.

```
>>> lr.fit(X_train, y_train) # Обучение модели
>>> prediction = lr.predict(X_N)
```

Отметим, что метод `fit` и `predict` вызываются от имени объекта `LinearRegression`. Обучившиеся на получившихся данных модель линейной регрессии.

Затем, реализуем цикл для подгонки модели на обучавшейся и проверочной выборки (train_cv), пусть N = 30:

```
>>> for N in range (1, N+1):
        pred_list = get_preds_lr(train_cv,'CLOSE',N, 0, n_train)
```

На каждом шагу добавляем в проверочный (cv) набор признак «pred_N»

```
>>> cv.loc[:, 'pred_' + str(N)] = pred_list
```

Таким образом, чтобы выбрать оптимальный признак из pred (1–30), посчитаем для каждого оценки качества модели предсказания модели, например MAPE, RMSE и R2. Построим график:

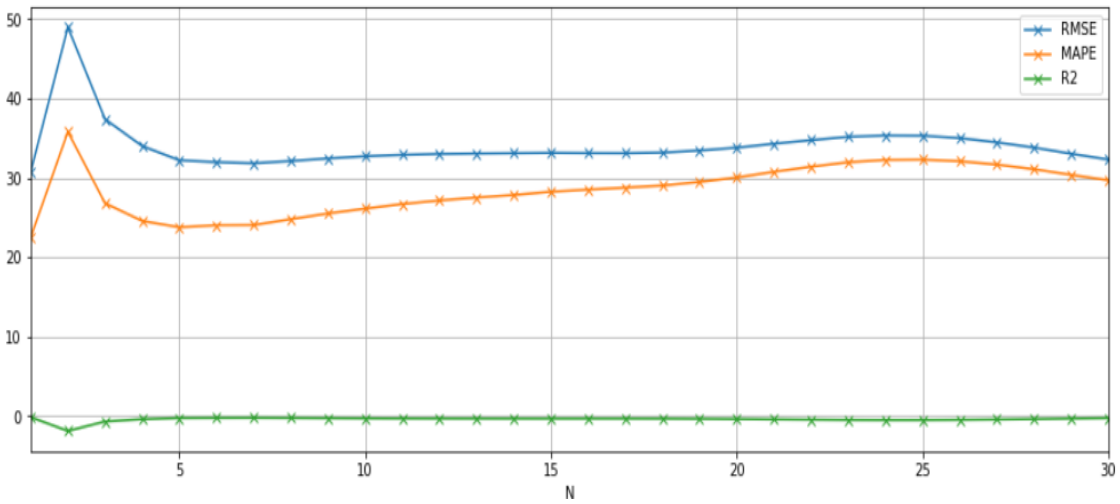

Рисунок 18 – График оценки качества модели MAPE, RMSE и R2

Отсюда можно сделать вывод что N = 6 оптимальна.

Наконец, построим финальную модель LR:

```
>>> pred_list = get_preds_lr(df, 'GMV', N_opt, 0, n_train+n_cv)
>>> test.loc[:, 'pred_' + str(N_opt)] = pred_list
```

На графике ниже показан полученная линейная регрессия объема продаж, красной линией отрисован прогноз модели, синий линией исходные значения (тестовых наборов данных).

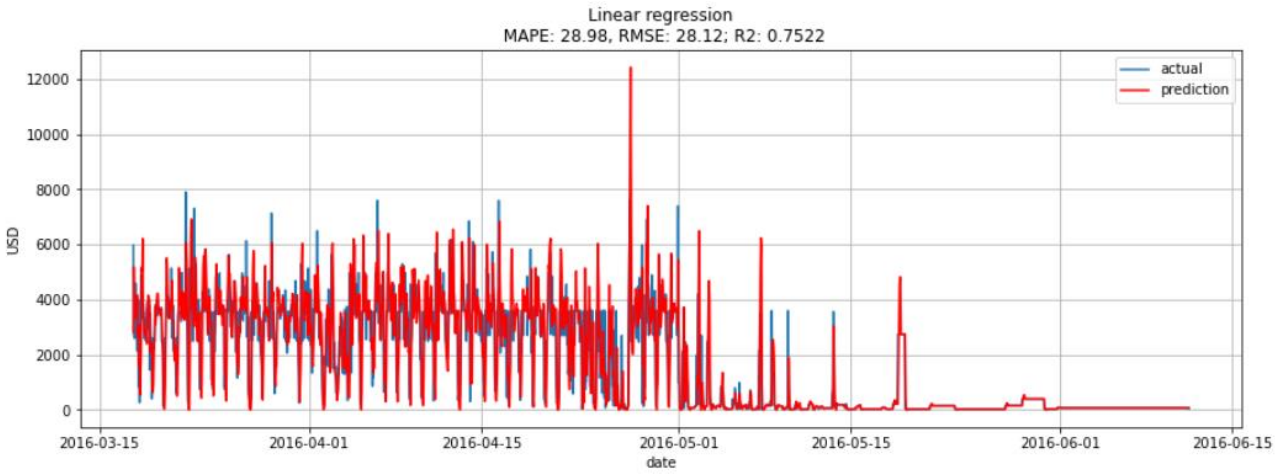

Рисунок 19 – Прогноз и исходные значения объемов продаж. Модель линейной регрессии

Теперь, количественно измерим качество аппроксимации, для сравнения различных моделей. Для этого можно измерить близость прогноза к истинным значениям воспользовавшись функциями `mean_absolute_error`, `mean_squared_error` из модуля `sklearn.metrics`:

```
>>> from sklearn.metrics import mean_absolute_error,
mean_squared_error
```

Эти функции принимают два аргумента: истинные и предсказанные значения:

```
>>> print("MAPE on train set = {:.4}".
format(get_mape(test.GMV, pred_list)))
>>> print("RMSE on train set = {:.4}".
format(math.sqrt(mean_squared_error(test.GMV, pred_list))))
```

Для расчета MAPE была реализована отдельная функция с именем `get_mape`, полный программный код представлен в Приложении В.

Но для понимания прогноз хороший или нет, проведем сравнение с эталоном – постоянной моделью. Если неизвестно о характеристике входных данных, то и спрогнозированы нечего нельзя лучше среднего значения y. Затем можно сравнить среднеквадратичную погрешность нашей модели и нулевой модели. Эту идею формализует коэффициент детерминации:

$$1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)^2} \approx 1 - \frac{MSE}{VAR(y)}$$

Здесь y_i – значение i-ого элемента, а $\widehat{y_i}$ – оценка этого элемента, полученная с помощью регрессионной модели. Наконец $\overline{y_i}$ – среднее значение y, представляющее нулевую модель, которая всегда возвращает одно и тоже значение. Следовательно, для идеальной модели получится оценка 1, а для нулевой – оценка 0 [4, с. 171].

Коэффициент детерминации вычислим с помощью функции `r2_score` из модуля `sklearn.metrics`. Ниже представлена таблица:

Таблица 2 – Результаты оценки модели линейной регрессии

Название модели	MAPE	RMSE	R^2
Линейная регрессия	28.98	28.12	0.7522

3.4.2 Модель градиентного спуска, алгоритм XGBoost

При создании модели ансамблевого метода, градиентного бустинга деревьев под название XGBoost, которым реализован в библиотеки `scikit-learn`. Методы бустинга могут быть очень мощными для регрессионных задач.

Реализацию алгоритма XGBoost выполним пример его применения на том же самом наборе данных, который мы использовали для линейной регрессии.

Будем обучать модель XGBoost на тренировочном (train) наборе, настраивать ее гиперпараметры с использованием проверочного (cv) набора и, наконец, применять модель XGBoost к тестовому (test) набору и сообщать результаты. Хорошая возможность уже к имеющимся признакам (MRP, SLA, Units, range_mg).

Но при создании модели, модель не корректно обучалось и выдавала прогноз с оценкой модели с отрицательным коэффициент детерминации. Проблема заключалась в не правильном масштабировании признаков в обучавшейся (train) выборке. Например, модель обучалась на признаках со значениями от 200 до 800 и поэтому модель может выводить только прогнозы в этом диапазоне. Когда модель пытается предсказать набор проверки и видит значения вне этого диапазона, она не может хорошо обобщать.

Решения данной проблемы — это правильное масштабирование набора признаков для обучения. Масштабировать тренировочную (train) так чтобы он

имел среднее (mean) = 0 и стандартное отклонение (std) = 1. Затем также на проверочной (cv) наборе. Например, если мы делаем прогнозы на день T, возьмем объем продаж (GMV) за последние N дней (дни T-N до T-1) и масштабирую их, чтобы они имели среднее значение 0 и дисперсию 1, то же самое и для остальных признаков: MRP, SLA, Units, range_mg.

Прогнозируемые значения также будут масштабироваться, и мы будем обратно преобразовывать их, используя их соответствующее среднее и стандартное отклонение. В ходе исследования обнаружил, что такой способ масштабирования дает наилучшую модель. Полный программный код представлен в Приложении Г.

Применим регрессию на основе алгоритма XGBOOST:

```
>>> from xgboost import XGBRegressor
```

Импортируем класс `XGBRegressor` из `xgoost` модуля и конструируем объект этого класса.

```
>>> model = XGBRegressor()
>>> model.fit(X_train_scaled, y_train_scaled)
>>> pred_scaled = model.predict(X_train_scaled)
```

Отметим, что метод `fit` и `predict` вызываются от имени объекта `XGBRegressor`. Обучаем на тренировочном (train) наборе нашу модель. Посчитаем оценки качества предсказания модели, например MAPE, RMSE и R2:

```
>>> get_mape(y_train, pred)
>>> math.sqrt(mean_squared_error(y_train, pred))
>>> r2_score(y_train, pred)
```

После обучения прогоним модель на проверочном (cv) наборе и вычислим метрики оценки качества модели:

```
>>> pred_scaled = model.predict(X_cv_scaled)
>>> get_mape(y_cv, cv.pred)
>>> math.sqrt(mean_squared_error(y_cv, cv.pred))
>>> r2_score(y_cv, cv.pred)
```

И наконец постоим финальную модель, для ее реализации был написан функция `def train_pred_xgboost` полный программный код представлен в Приложении Г.

```
rmse, mape, r2, pred = train_pred_xgboost(X_train_cv_scaled,
                                          y_train_cv_scaled,
                                          X_sample_scaled,
                                          y_sample,
                                          test.CLOSE_mean,
                                          test.CLOSE_std)
```

XGBOOST имеет множество гиперпараметров, которые можно настраивать или перебрать по сетке сколько-нибудь значительное количество комбинаций, но из-за нескольких причин в данной работе не будет реализовано. Поэтому при использовании бустинга решающих деревьев воспользуемся только самыми важными параметрами алгоритма:

```
>>> params = {
        'objective':'reg:squarederror','booster':'gbtree',
        'gamma': 0, 'n_estimators': 100
        'min_child_weight': 1,
        'max_depth': 3,'subsample': 1,
        'colsample_bytree': 1, 'colsample_bylevel': 1,
        'learning_rate': 0.1,
                }
```

Значения параметров были подобраны по умолчанию. На график ниже отображен полученная модель объемов продаж (GMV), красной линией отрисован прогноз модели, синий линией исходные значения (тестовых наборов данных).

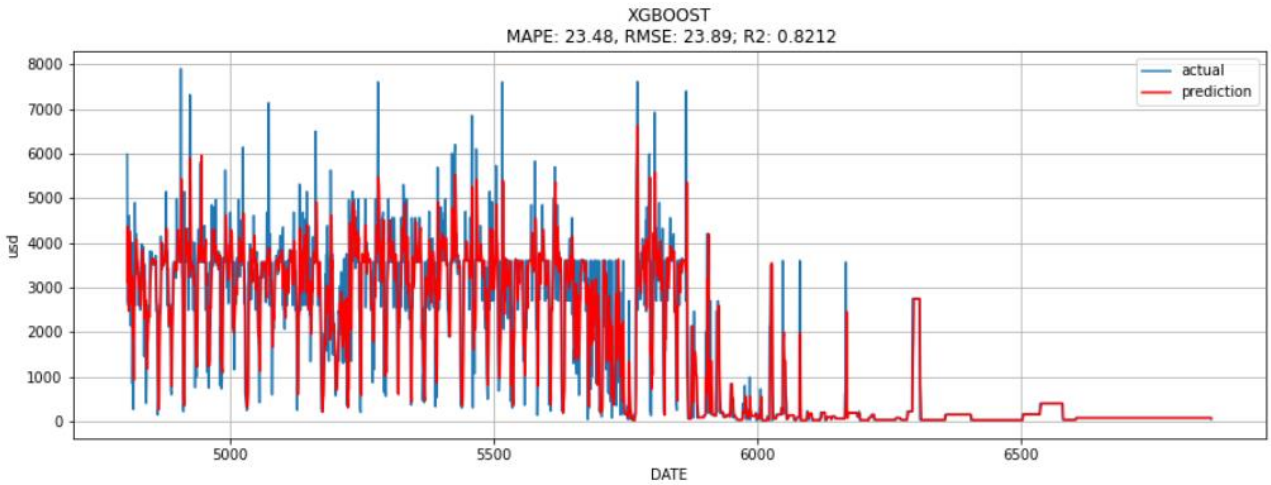

Рисунок 20 – Прогноз и исходные значения объёмов продаж. Модель XGBOOST

Таблица 3 – Результаты оценки модели XGBoost

Название модели	MAPE	RMSE	R^2
XGBOOST	23.48	23.89	0.8212

3.4.3 Модель рекуррентной нейронной сети, LSTM-сеть

LSTM — это метод глубокого обучения, разработанный для борьбы с проблемой исчезающих градиентов, встречающейся в длинных последовательностях. LSTM имеет три элемента: шлюз обновления, шлюз забывания и выходной шлюз. Ворота обновления и забывания определяют, обновляется ли каждый элемент ячейки памяти. Выходной вентиль определяет количество информации, выводимой в виде активаций на следующий уровень.

Ниже на рисунке 21 приведена архитектура LSTM, которую буду использовать. Мы будем использовать два слоя модулей LSTM и промежуточный слой, чтобы избежать чрезмерной подгонки.

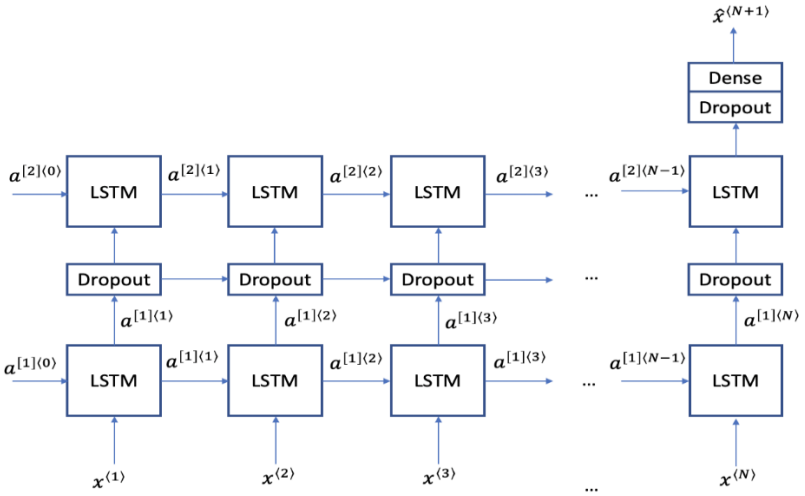

Рисунок 21 – Сетевая архитектура LSTM

Начнем реализацию модели прогнозирования на основе LSTM-сети, то есть попытаемся предсказать объем (GMV) по тем же признакам что и модели XGBOOST. Будем обучать модель LSTM-сети на тренировочном (train) наборе, настраивать ее гиперпараметры с использованием проверочного (cv) набора и, наконец, применять модель LSTM к тестовому (test) набору и сообщать результаты Для этого реализуем функцию `train_pred_eval_model()` полный программный код представлен в Приложении Д.

При создании модели LSTM, модель не корректно обучалось и выдавала прогноз с оценкой модели с отрицательным коэффициент детерминации. Проблема была все такой что и ранее для модели XGBOOST в не правильном масштабировании признаков в обучавшейся (train) выборке.

Решения данной проблемы — это правильное масштабирование набора признаков для обучения. В соответствии с этим были реализованы функции `get_x_y()` и `get_x_scaled_y()`. Полный программный код представлен в Приложении Д.

Построим модель LSTM-сети:

```
>>> from keras.models import Sequential
```

Импортируем класс `Sequential` из `keras.models` модуля и конструируем объект этого класса.

```
>>> model = Sequential()
>>> model.add(LSTM(units=lstm_units,return_sequences=True,
input_shape=(x_train_scaled.shape[1],1)))
>>> model.add(Dropout(dropout_prob))
>>> model.add(LSTM(units=lstm_units))
>>> model.add(Dropout(dropout_prob))
>>> model.add(Dense(1))
>>> model.compile(loss='mean_squared_error', optimizer=optimizer)
```

Таким образом мы добавляем параметры для модели LSTM

```
>>> model.fit(x_train_scaled, y_train_scaled, epochs=epochs,
batch_size=batch_size, verbose=2)
>>> est_scaled = model.predict(x_cv_scaled)
```

Отметим, что метод `fit` и `predict` вызываются от имени объекта `Sequential`. Обучаем на тренировочном (train) наборе нашу модель.

После обучения прогоним модель на проверочном (cv) наборе и вычислим метрики оценки качества модели, например MAPE, RMSE и R2:

```
>>> est = (est_scaled * np.array(std_cv_list).reshape(-1,1)) +
np.array(mu_cv_list).reshape(-1,1)
>>> get_mape(y_cv, est))
>>> math.sqrt(mean_squared_error(y_cv, est)))
```

```
>>> r2_score(y_cv, est)
```

И наконец постоим финальную модель, для ее реализации был написан функция `dev train_pred_eval_model` полный программный код представлен в Приложении Д.

```
rmse, mape, r2, est = train_pred_eval_model(x_train_cv_scaled,
                                            y_train_cv_scaled,
                                            x_test_scaled,
                                            y_test,
                                            mu_test_list,
                                            std_test_list)
```

LSTM имеет множество гиперпараметров, которые можно настраивать или перебрать по сетке сколько-нибудь значительное количество комбинаций.

```
>>> model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_24 (LSTM)               (None, 9, 50)             10400
_________________________________________________________________
dropout_24 (Dropout)         (None, 9, 50)             0
_________________________________________________________________
lstm_25 (LSTM)               (None, 50)                20200
_________________________________________________________________
dropout_25 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 51
=================================================================
Total params: 30,651
Trainable params: 30,651
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 22 – Сводка параметров модели LSTM

На рисунке 26 можно увидеть сводку параметров модели, в данном случае на рисунке 6 видно, что указанная модель имеет 30,651 параметров. Но из-за некоторых причин в данной работе не будет реализовано точечный подбор параметров для данной модели. Значения параметров были подобраны по умолчанию.

На график ниже отображен полученные регрессии объемов продаж (GMV), красной линией отрисован прогноз модели, синий линией исходные значения (тестовых наборов данных).

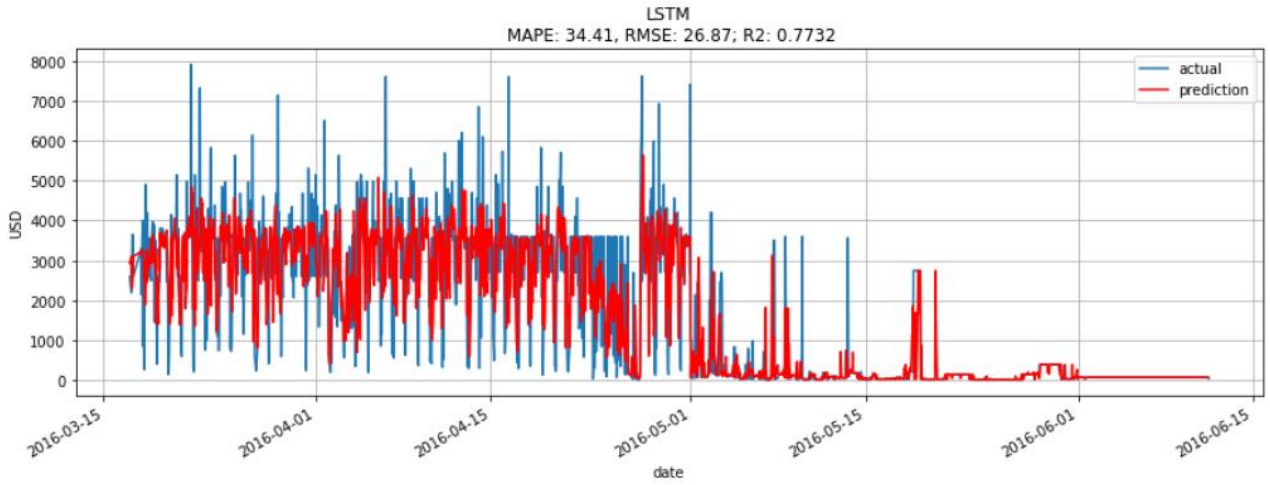

Рисунок 23 – Прогноз и исходные значения объёмов продаж. Модель LSTM

Таблица 4 – Результат оценки модели LSTM-сети

Название модели	MAPE	RMSE	R^2
LSTM	34.41	26.87	0.7732

3.5 Оценка моделей и выводы

В данной главе были построены модели по разным методам (алгоритмам), которые целесообразно применять к задачи прогнозирования. Результаты оценок на тестовой выборке моделей представлены на таблице 5. По результатам прогнозирование объёмов продаж (GMV), методы, основанные на сложной математической модели, учитывающей либо несколько простых моделей внутри себя, либо эффективно улучшающий показатель ошибки в рамках одной модели, показали, что модели пригодны для прогнозирования набора данных kaggle.

Таблица 5 – Оценки моделей прогнозирования объемов продаж

Название Модели	MAPE %	RMSE	R^2
SARIMA	98.09	820.4	0.7792
LR	28.98	28.12	0.7522
XGBOOST	23.48	23.89	0.8212
LSTM	34.41	26.87	0.7732

Сравнительный анализ алгоритмов показывает, что получившиеся модели SARIMA на 5.3%, LR на 9.1% а LSTM на 6.2 % хуже, чем модель XGBOOST.

Наилучшее значение, показал алгоритм XGBoost. Его эффективность подтверждается на практике, в связи с этим исследователям следует брать ее для

решение задач прогнозирования. Следует подчеркнуть, что другие методы тоже показали не плохие результаты, их можно использовать для прогнозирования объёмов продаж наборе данных kaggle.

Ниже представлен график полученных прогнозов за один день:

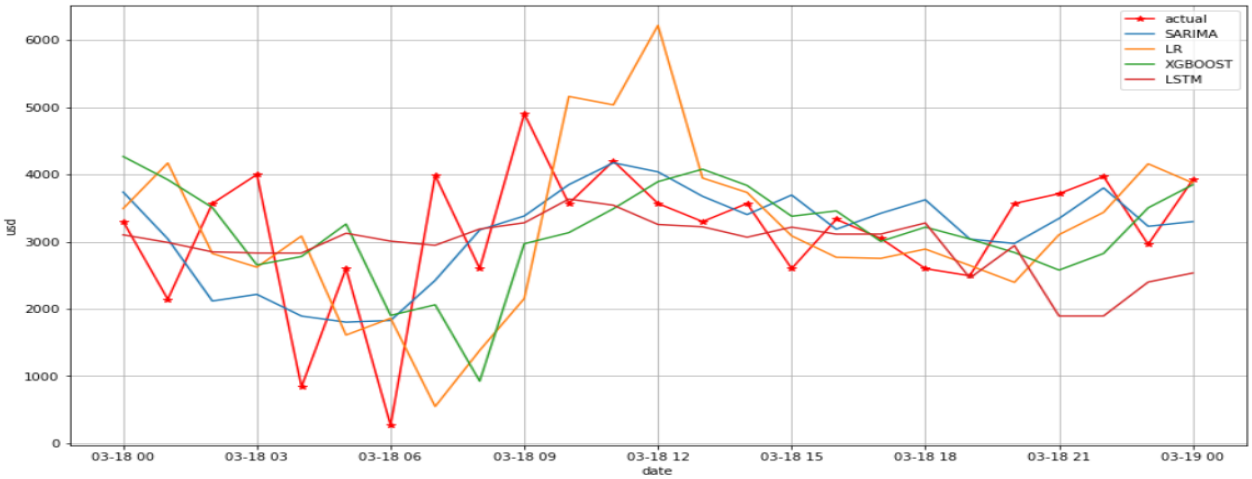

Рисунок 24 – График исходных и прогнозных значений (за один день)

При выборе алгоритма, следует учитывать специфику анализируемых данных. Авторегрессии проинтегрированного скользящего среднего, благодаря простоте, единому представления анализа и проектирования пользуется большим спросом, но в данном исследований показала время затратность при построении модели.

Линейная регрессия ее простатита и легкость интерпретации результатов, показала не плохой результат оказалось наихудшей моделью, ее слабость заключается в трудоемкости определения параметров.

Экстремальный градиентный спуск не сложен в реализации, обладает большим количеством гиперпараметров и может работать с неправильными данными. Ее преимущества оправдались на практике данный метод построил наилучшую модель.

Рекуррентная нейронная LSTM-сеть способен моделировать не линейные процессы, обладает адаптивностью и масштабируемостью. На практике показала результаты не оправдывающие силы ее реализации.

Заключение

Настоящая выпускная квалификационная работа посвящена построению моделей машинного обучения, для прогнозирования объёмов продаж. Прогнозы составлялись с применением различных методов и алгоритмов машинного обучения. Для реализации задачи использовались соответствующие инструменты по итогу написаны модули на языке программирования Python 3.

Практически полученные результаты удовлетворяют цели работы, модели справляются с поставленной задачей и их можно использовать на практике. Исследование показало, что модель экстремального градиентного спуска оптимальна на примере данных Kaggle.

Для успешного выполнения роботы были выполнены все задачи, которые мы определили в начале:

- Была изучена теория прогнозирования временных рядов;
- Исследованы методы машинного обучения для прогнозирования;
- Обработан полученный набор данных из Kaggle;
- Построены модели прогноза;
- В заключении описаны оценки и результаты по сравнительному анализу моделей.

Цель выпускной квалификационной работы достигнута и выполнена успешно.

Список литературы

1. Алжеев А. В., Кочкаров Р.А. Сравнительный анализ прогнозных моделей ARIMA и LSTM на примере акций российских компаний. Финансовый университет, Москва, Россия, 2020. – 36с [Электронный ресурс]. — Режим доступа: https://www.elibrary.ru/item.asp?id=42572916 (дата обращения 21.06.2021)

2. Беляевский, И. К. Маркетинговое исследование: информация, анализ, прогноз : учебное пособие / И. К. Беляевский. - 2-е изд., перераб. и доп. - Москва : КУРС : ИНФРА-М, 2020. - 392 с. - ISBN 978-5-905554-08-7. - Текст : электронный. - URL: (https://habr.com/ru/post/177633/) https://znanium.com/catalog/product/1054208 (дата обращения: 21.06.2021).

3. Дуброва Т. А. Статистические методы прогнозирования. [Текст] / Т. А. Дуброва М.: ЮНИТИ-ДАНА, 2003. 206 с.: ISBN 5-238-00497-4. - Текст: электронный. - URL: https://www.azstat.org/Kitweb/zipfiles/11872.pdf (дата обращения: 21.06.2021)

4. Коэльо, Л. П. Построение систем машинного обучения на языке Python / Л. П. Коэльо, В. Ричарт ; перевод с английского А. А. Слинкин. — 2-е изд. — Москва : ДМК Пресс, 2016. — 302 с. — ISBN 978-5-97060-330-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/82818 (дата обращения: 21.06.2021).

5. Крянев, А. В. Эконометрика (продвинутый уровень): Конспект лекций / Крянев А.В. - Москва :КУРС, НИЦ ИНФРА-М, 2017. - 62 с.: ISBN 978-5-906818-62-1. - Текст : электронный. - URL: https://znanium.com/catalog/product/767248 (дата обращения: 21.06.2021).

6. Магнус Я. Р., Катышев П.К., Пересецкий А.А. Эконометрика. Начальный курс: Учеб. – 6-е изд., перераб. и доп. – М.: Дело, 2004. – 576 с.: ISBN 5-7749-0055-X. - Текст : электронный. - URL: http://math.isu.ru/ru/chairs/me/files/books/magnus.pdf (дата обращения: 21.06.2021).

7. Рашка, С. Python и машинное обучение: крайне необходимое пособие по новейшей предсказательной аналитике, обязательное для более глубокого понимания методологии машинного обучения / С. Рашка ; пер. с англ. А.В. Логунова. - Москва : ДМК Пресс, 2017. - 418 с. - ISBN 978-5-97060-409-0. - Текст : электронный. - URL: https://znanium.com/catalog/product/1027758 (дата обращения: 21.06.2021).

8. Селиверстова А. В., Сравнительный анализ моделей и методов прогнозирования // Современные научные исследования и инновации. 2016. № 11 [Электронный ресурс]. URL: https://web.snauka.ru/issues/2016/11/74271 (дата обращения: 21.06.2021).

9. Шарден, Б. Крупномасштабное машинное обучение вместе с Python : учебное пособие / Б. Шарден, Л. Массарон, А. Боскетти ; перевод с английского А. В. Логунова. — Москва : ДМК Пресс, 2018. — 358 с. — ISBN 978-5-97060-506-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/105836 (дата обращения: 21.06.2021).

10. Янсен С. Машинное обучение для алгоритмической торговли на финансовых рынках. Практикум: Пер. с англ. — СПб.: БХВ-Петербург, 2020. — 560 с.: ISBN 978-5-9775-6595-0. (дата обращения: 21.06.2021).

11. Набора данных на Kaggle. URL: https://www.kaggle.com/datatattle/dt-mart-market-mix-modeling (дата обращения: 21.06.2021)

12. Программный код на GitHub. URL: https://github.com/ariakhmatov/DiplomProject-ML_predict

Приложение А

Исходный ременной ряд распложён по ссылке ниже:

https://www.kaggle.com/datatattle/dt-mart-market-mix-modeling?select=Sales.csv

Приложение Б

Программный код «МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АНАЛИЗА И ПРОГНОЗА ОБЪЁМОВ ПРОДАЖ (НА ПРИМЕРЕ ДАННЫХ ПРОЕКТА «Kaggle»)» на языке Python.

#библиотеки

```
import sys
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from statsmodels.iolib.table import SimpleTable
import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.metrics import r2_score
from scipy.ndimage.interpolation import shift
from scipy.optimize import minimize
import scipy.stats as scs
from itertools import product
from tqdm import tqdm
#------------------------------------------------------------------------------
```

#функции

```
def get_mape(y_true, y_pred):
    """
    Вычислить среднюю абсолютную процентную ошибку (MAPE)
    """
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


def plotMovingAverage(series, n): # Код для отрисовки график метода скользящей средней

    rolling_mean = series.rolling(window=n).mean() # сглаживание исходного ряда для
выявления трендов
    # Чем больше зададим ширину интервала n — тем более сглаженным окажется тренд

    # При желании, можно строить и доверительные интервалы для сглаженных значений
    rolling_std =  series.rolling(window=n).std()
```

```
    upper_bond = rolling_mean+1.96*rolling_std
    lower_bond = rolling_mean-1.96*rolling_std

    plt.figure(figsize=(10,5))
    plt.title("Moving average\n window size = {}".format(n))
    plt.plot(rolling_mean, "g", label="Rolling mean trend") # Скользящий средний тренд

    plt.plot(upper_bond, "r--", label="Upper Bond / Lower Bond")
    plt.plot(lower_bond, "r--")
    plt.plot(series[n:], label="Actual values") # Фактические значения
    plt.legend(loc="upper left")
    plt.grid(True)

def weighted_average(series, weights): # Метод взвешенной средней за последний месяц
    result = 0.0
    weights.reverse()
    for n in range(len(weights)):
        result += series[-n-1] * weights[n]
    return result

def exponential_smoothing(series, alpha): # Метод экспоненциального сглаживания
    result = [series[0]] # first value is same as series
    for n in range(1, len(series)):
        result.append(alpha * series[n] + (1 - alpha) * result[n-1])
    return result

def double_exponential_smoothing(series, alpha, beta): # Метод двойного
экспоненциального сглаживания
    result = [series[0]]
    for n in range(1, len(series)+1):
        if n == 1:
            level, trend = series[0], series[1] - series[0]
        if n >= len(series): # прогнозируем
            value = result[-1]
        else:
            value = series[n]
        last_level, level = level, alpha*value + (1-alpha)*(level+trend)
        trend = beta*(level-last_level) + (1-beta)*trend
        result.append(level+trend)
    return result
```

```
def tsplot(y, lags=None, figsize=(12, 7), style='bmh'): # Критерий Дики-Фуллера
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
    with plt.style.context(style):
        fig = plt.figure(figsize=figsize)
        layout = (2, 2)
        ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
        acf_ax = plt.subplot2grid(layout, (1, 0))
        pacf_ax = plt.subplot2grid(layout, (1, 1))

        y.plot(ax=ts_ax)
        ts_ax.set_title('Time Series Analysis Plots')
        smt.graphics.plot_acf(y, lags=lags, ax=acf_ax, alpha=0.5)
        smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax, alpha=0.5)

        plt.tight_layout()

        print("Критерий Дики-Фуллера: p=%f" % sm.tsa.stattools.adfuller(y)[1],'< 0.05')

    return

def invboxcox(y,lmbda):# обратное преобразование Бокса-Кокса
    if lmbda == 0:
        return(np.exp(y))
    else:
        return(np.exp(np.log(lmbda*y+1)/lmbda))
#-------------------------------------START-----------------------------------------
```

#Модуль программы

```
# загрузка dataset
db_ = pd.read_csv('Данные/DT_MART/Sales.csv','\t',parse_dates=['Date'],
index_col='Date', dayfirst=True, low_memory = False) #1,048,575
df2 = db_.sort_index() # сортировка

# преобразование данных, очистка пустых значений и изменение типа
df2.replace(' ', np.nan, inplace=True) #1,046,133
df2.replace(0.0, np.nan, inplace=True) #1,045,311
df2 = df2.dropna()
df2["GMV"] = pd.to_numeric(df2["GMV"])

#unique_arr = db3["SLA"].unique() # групировка содержимых значений
```

```
# удаление не нужных атрибутов
df2.drop(['ID_Order','ID_Item_ordered','Product_Category',
'Sub_category','Procurement_SLA'],axis=1, inplace=True)

# переименование
df2['Category'],df2['Brand'],df2['Units'] =
df2['Analytic_Category'],df2['product_analytic_vertical'],df2['Units_sold']
df2.drop(['Analytic_Category','product_analytic_vertical','Units_sold'], axis=1,
inplace=True)

# конечный вид данных
df2 = df2[['ID','Category','Brand','GMV','MRP','Units','SLA']]
df2.info()

#Сохранение нового файла
df2.to_csv('SalesX_.csv', sep=';', encoding='utf-8')

# загрузка dataset
db = pd.read_csv('SalesX.csv',';', parse_dates=['Date'], index_col='Date',
dayfirst=True, low_memory = False) #1,045,311

df2 = db

# Объем продаж с 2015-2016г в часах
df = df2[['GMV']].resample('h').max() #df = df[['GMV','MRP']]
df = df.fillna(method='ffill')
df.plot(figsize=(10,5))
plt.legend(loc = 'best')
plt.xlabel("date")
plt.ylabel("usd")
plt.grid(True)

df = df2[['GMV','MRP','SLA','Units']].resample('h').max()
df = df.fillna(method='ffill')
df = pd.DataFrame({'GMV': df.GMV/10,'MRP': df.MRP/10,
                   'SLA': df.SLA,'Units': df.Units
                   })

itog_GMV = df.GMV.describe()
print(itog_GMV)
df['GMV'].hist()
```

```
print ('V = %f' % (itog_GMV['std']/itog_GMV['mean']))

# сглаживаени сильных выбросов, в данных
j = 0
l = []
for i in range(0, 6863):
    if df.GMV[i] > 8000: # or df.GMV[i] < 10000:
        df.replace(df.GMV[i], np.nan, inplace=True)
        df.fillna(method='ffill',inplace=True )
        j+=1
        l.append(i)
df.dropna(inplace=True)

# проверка на стационарность ряда
test = sm.tsa.adfuller(df.GMV)
print ('adf: ', test[0])
print ('p-value: ', test[1])
print('Critical values: ', test[4])
if test[0]> test[4]['5%']:
    print ('есть единичные корни, ряд не стационарен')
else:
    print ('единичных корней нет, ряд стационарен')

# Метод скользящей средней
def moving_average(series, n):# Метод скользящей средней за последний день
    return np.average(series[-n:])
print(moving_average(df.GMV[5147:5868], 24)) # среднее за последние 24 часа
# График метод скользящей средней
plotMovingAverage(df.GMV[5147:5868], 24*7*4) # сглаживаем по дню
plotMovingAverage(df.GMV[755:2219], 24*7*4) # сглаживаем по месяцу

#Метод взвешенной средней
def weighted_average(series, weights):
    result = 0.0
    weights.reverse()
    for n in range(len(weights)):
        result += series[-n-1] * weights[n]
    return result
print(weighted_average(df.GMV[5147:5868], [0.8, 0.1, 0.05, 0.04, 0.01]))
```

```
# Модель ARIMA и SARIMA
tsplot(df.GMV, lags=30)
data = df.copy()
data['GMV_'] = data['GMV']
data['GMV'], lmbda = scs.boxcox(data.GMV)
print("Оптимальный параметр преобразования Бокса-Кокса: %f" % lmbda)
tsplot(data.GMV, lags=30)

data['GMV_1'] = data.GMV_ - data.GMV_.shift(1)
tsplot(data.GMV_1[1:], lags=30)

# добавление доп. признаков
data["date"] = data.index.date
data['range_mg'] = data.MRP - data.GMV_

# Подбор параметров модели ARIMA
# начальное приближение параметров Q=3, P=4, q=1, p=1
ps = range(0, 2)
d  = 1
qs = range(0, 2)
Ps = range(0, 3)
D  = 1
Qs = range(0, 3)
parameters = product(ps, qs, Ps, Qs)
parameters_list = list(parameters)
len(parameters_list)

# таймер
#%time
results = []
best_aic = float("inf")

for param in tqdm(parameters_list):
    #try except нужен, потому что на некоторых наборах параметров модель не обучается
    try:
        model=sm.tsa.statespace.SARIMAX(data.GMV, order=(param[0], d, param[1]),
                                        seasonal_order=(param[2], D, param[3],
24)).fit(disp=-1)

    #выводим параметры, на которых модель не обучается и переходим к следующему набору
    except ValueError:
```

```
            print('wrong parameters:', param)
            continue
        aic = model.aic

        #сохраняем лучшую модель, aic, параметры
        if aic < best_aic:
            best_model = model
            best_aic = aic
            best_param = param
        results.append([param, model.aic])

warnings.filterwarnings('default')

result_table = pd.DataFrame(results)
result_table.columns = ['parameters', 'aic']
print(result_table.sort_values(by = 'aic', ascending=True).head())

#%time
best_model = sm.tsa.statespace.SARIMAX(data.GMV, order=(2,1,1),
                                       seasonal_order=(0,1,1, 24)).fit(disp=-1)

print(best_model.summary())
tsplot(best_model.resid[24:], lags=30)

#график прогноза модели
data["arima_model"] = invboxcox(best_model.fittedvalues, lmbda)
forecast = invboxcox(best_model.predict(start = data.shape[0], end = data.shape[0]),
lmbda)
forecast = data["arima_model"].append(forecast)[-2058:]
#print(list(forecast))

actual = data.GMV_[-2058:]

plt.figure(figsize=(15, 5))
plt.plot(actual, label="actual")
plt.plot(forecast, 'r', label="prediction")
plt.legend(loc = 'best')
plt.xlabel("date")
plt.ylabel("USD")
#plt.title("SARIMA model\n Mean absolute error {} GMV".format
#          (round(mean_absolute_error(data.dropna().GMV_, data.dropna().arima_model))))
```

```
plt.title("SARIMA model\n MAPE: {:.4}, RMSE: {:.4}; R2: {:.4}".format(
     get_mape(data.dropna().GMV_[-2058:], data.dropna().arima_model[-2058:]),
     np.sqrt(mean_squared_error(data.dropna().GMV_[-2058:], data.dropna().arima_model[-
2058:])),
     r2_score(data.dropna().GMV_[-2058:], data.dropna().arima_model[-2058:])))
plt.grid(True)
#---------------------------------------- END ----------------------------------------
```

Приложение В

Программный код «МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АНАЛИЗА И ПРОГНОЗА ОБЪЁМОВ ПРОДАЖ (НА ПРИМЕРЕ ДАННЫХ ПРОЕКТА «Kaggle»)» на языке Python.

#библиотеки

```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
#-------------------------------------------------------------------------------
```

#функции

```
def invboxcox(y,lmbda):# обратное преобразование Бокса-Кокса
    if lmbda == 0:
        return(np.exp(y))
    else:
        return(np.exp(np.log(lmbda*y+1)/lmbda))

def get_mape(y_true, y_pred):
    """
    Вычислить среднюю абсолютную процентную ошибку (MAPE)
    """
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def get_preds_lr(df, target, N, pred_min, offset):
    """
    Получить прогноз на временном интервале t, используя значения из t-1, t-2, ..., t-
N.
    Inputs
        df         : dataframe со значениями, которые вы хотите предсказать. Может быть
любой длины.
        target     : имя столбца, который вы хотите предсказать, например, 'GMV'.
        N          : получить предсказание на временном интервале t по значениям из t-
1, t-2, ..., t-N
        pred_min   : все предсказания должны быть >= pred_min
        offset     : для df мы делаем предсказания только для df[offset:]. например,
```

```
offset может быть размером обучающего множества
    Outputs
        pred_list  : прогноз для target np.array длины len(df)-offset
    """
    # Создать объект линейной регрессии
    lr = LinearRegression(fit_intercept = True)
    # лист для прогноза
    pred_list = []
    for i in range(offset, len(df['GMV'])):
        X_train = np.array(range(len(df['GMV'][i-N:i]))) # e.g. [0 1 2 3 4]
        y_train = np.array(df['GMV'][i-N:i]) # e.g. [2944 3088 3226 3335 3436]
        X_train = X_train.reshape(-1, 1)     # e.g X_train =
                                             # [[0]
                                             #  [1]
                                             #  [2]
                                             #  [3]
                                             #  [4]]
        # X_train = np.c_[np.ones(N), X_train]              # add a column
        y_train = y_train.reshape(-1, 1)
    #     print X_train.shape
    #     print y_train.shape
    #     print 'X_train = \n' + str(X_train)
    #     print 'y_train = \n' + str(y_train)
        lr.fit(X_train, y_train)            # Обучение модели

        prediction = lr.predict(np.array(N).reshape(1,-1))
        #print(prediction)
        pred_list.append(prediction[0][0])  # Прогнозируйте количество шагов с помощью
модели

    # If the values are < pred_min, set it to be pred_min
    pred_list = np.array(pred_list)
    pred_list[pred_list < pred_min] = pred_min
    return pred_list
#-------------------------------------START-------------------------------------------
```

#Модуль программы

```
# загрузка dataset
db = pd.read_csv('SalesX.csv',';', parse_dates=['Date'], index_col='Date',
dayfirst=True, low_memory = False) #1,045,311
df2 = db
```

```
# 5 категории товаров, 74 вида товара
df_Camera = df2[df2.Category == 'Camera'] #5
df_CameraA = df2[df2.Category == 'CameraAccessory'] #25
df_Entertainment = df2[df2.Category == 'EntertainmentSmall'] #23
df_Gaming = df2[df2.Category == 'GamingHardware'] #18
df_GameCDDVD = df2[df2.Category == 'GameCDDVD'] #3
# график
df_Camera['GMV'].plot(figsize=(15,5), label="Camera", grid=True)
df_CameraA['GMV'].plot(figsize=(15,5),label="CameraA",grid=True)
df_Entertainment['GMV'].plot(figsize=(15,5),label="Entertainment",grid=True)
df_Gaming['GMV'].plot(figsize=(15,5),label="Gaming",grid=True)
df_GameCDDVD['GMV'].plot(figsize=(20,5),label="GameCDDVD",grid=True)
plt.legend(loc = 'best'

# Объем продаж (усредненные) с 2015-2016г за каждый час
df = df2[['GMV']].resample('h').max() #last() first() min() max() mean()
df = df.fillna(method='ffill')
df.plot(figsize=(15,5),label="GMV")
plt.grid(True)

df = df2[['GMV','MRP','SLA','Units']].resample('h').max()
df = df.fillna(method='ffill')
df = df.dropna()

df = pd.DataFrame({'GMV': df.GMV/10,'MRP': df.MRP/10,
                   'SLA': df.SLA,'Units': df.Units
                   })

itog_GMV = df.GMV.describe()
print(itog_GMV)
df['GMV'].hist()
print ('V = %f' % (itog_GMV['std']/itog_GMV['mean']))

j = 0
l = []
for i in range(0, 6863):
    if df.GMV[i] > 8000:
        df.replace(df.GMV[i], np.nan, inplace=True)
        df.fillna(method='ffill',inplace=True )
        #df.replace(df.GMV[i], df.GMV[i]*100, inplace=True)
        j+=1
```

```
        l.append(i)

# проверка на стационарность
test = sm.tsa.adfuller(df['GMV'])
print ('adf: ', test[0])
print ('p-value: ', test[1])
print('Critical values:', test[4])
if test[0]> test[4]['5%']:
    print ('есть единичные корни, ряд не стационарен')
else:
    print ('единичных корней нет, ряд стационарен')

# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there
are more than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of
kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize],
diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.5, 0.2), xycoords='axes
fraction', ha='center', va='center', size=textSize)
    plt.suptitle('Scatter and Density Plot')
    plt.show()

plotScatterMatrix(df, 10, 12)

# Correlation matrix
def plotCorrelationMatrix(df, graphWidth):
    #filename = df.dataframeName
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there
are more than 1 unique values
```

```
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant columns
({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w',
edgecolor='k')
    corrMat = plt.matshow(corr,fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    #plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()

plotCorrelationMatrix(df, 5)

df['GMV_'] = df['GMV']
df['GMV'], lmbda = scs.boxcox(df.GMV)
print("Оптимальный параметр преобразования Бокса-Кокса: %f" % lmbda)
tsplot(df.GMV, lags=30)

# добавление доп. признаков
df["date"] = df.index.date
# Вычислим разницу между
df['range_mg'] = df.MRP - df.GMV_

# Разделить на test, cv и test набор
# Разделим этот набор данных на 60% train, 20% validation и 20% test
# Количество train, cv, test наборов данных

test_size = 0.3                 # proportion of dataset to be used as test set
cv_size = 0.2                   # proportion of dataset to be used as cross-validation
set
Nmax = 30                       # for feature at day t, we use lags from t-1, t-2, ...,
t-N as features
                                # Nmax is the maximum N we are going to test

n_cv, n_test = int(cv_size*len(df)), int(test_size*len(df))
n_train =  len(df) - n_cv - n_test
print('Количество:')
```

```
print(" n_train = " + str(n_train),
        "\n n_cv = " + str(n_cv),
        "\n n_test = " + str(n_test))


# Разделить на train, cv, and test
train = df[:n_train].copy()
cv = df[n_train:n_train+n_cv].copy()
train_cv = df[:n_train+n_cv].copy()
test = df[n_train+n_cv:].copy()
print('Размерность:')
print(" train = " + str(train.shape),
        "\n cv = " + str(cv.shape), "~ train_cv = " + str(train_cv.shape),
        "\n test = " + str(test.shape))

# График деление на train, cv, и test
plt.figure(figsize=(16, 9))
plt.plot(train.GMV, 'b',label = 'train')
plt.plot(cv.GMV, 'y',label= 'cv')
plt.plot(test.GMV, 'g',label= 'test')
plt.legend(loc='best')
plt.xlabel("DATA")
plt.ylabel("RUB")
plt.grid(True)

# Простая линенная регрессия
RMSE, R2, MAPE = [],[],[]
# N is no. of samples to use to predict the next value
for N in range(1, Nmax+1):
                            #(df, target, N, pred_min, offset):
    LR,pred_list = get_preds_lr(train_cv, 'GMV', N, 0, n_train)
    # добавляем прогноз исходного ряда в качестве признаков
    cv.loc[:, 'pred_' + str(N)] = pred_list
    #append
    RMSE.append(math.sqrt(mean_squared_error(cv.GMV, pred_list)))
    MAPE.append(get_mape(cv.GMV, pred_list))
    R2.append(r2_score(cv.GMV, pred_list))

#print('RMSE = ' + str(RMSE))
#print('MAPE = ' + str(MAPE))
#print('R2 = ' + str(R2))
```

```
# График качества прогноза на проверочных данных cv
plt.figure(figsize=(15, 5))
plt.plot(range(1, Nmax+1), RMSE, 'x-',label="RMSE" )
plt.plot(range(1, Nmax+1), MAPE, 'x-',label="MAPE")
plt.plot(range(1, Nmax+1), R2, 'x-',label="R2")
plt.xlabel('N')
plt.xlim([1, Nmax])
plt.legend(loc = 'best')
plt.grid(True)

# Set optimum N
N_opt = 6
                        #(df, target, N, pred_min, offset):
pred_list = get_preds_lr(df, 'GMV', N_opt, 0, n_train+n_cv)
test.loc[:, 'pred_' + str(N_opt)] = pred_list

pred_df = pd.DataFrame({'pred_list': pred_list.reshape(-1),
                        'Date': df[n_train+n_cv:]['date']})

# График прогноза
pred_df["pred_list_"] = invboxcox(pred_df.pred_list, lmbda)

import matplotlib.pyplot as plt
plt.figure(figsize=(15, 5))
plt.plot(test.GMV_, label="actual" )
plt.plot(pred_df.pred_list_,'r' ,label="prediction")
plt.legend(loc = 'best')
plt.xlabel("date")
plt.ylabel("USD")
plt.title("Linear regression\n MAPE: {:.4}, RMSE: {:.4}; R2: {:.4}".format(
    get_mape(test.GMV,pred_df.pred_list),
    np.sqrt(mean_squared_error(test.GMV, pred_df.pred_list)),
    r2_score(test.GMV, pred_df.pred_list)))
plt.grid(True)

print("MAPE on train set = {:.4}".format(get_mape(test.GMV, pred_list)))
print("RMSE on train set = {:.4}".format(math.sqrt(mean_squared_error(test.GMV,
pred_list))))
print("R2 on train set = {:.4}".format(r2_score(test.GMV, pred_list)))
#------------------------------------ END ----------------------------------------
```

Приложение Г

Программный код «МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АНАЛИЗА И ПРОГНОЗА ОБЪЁМОВ ПРОДАЖ (НА ПРИМЕРЕ ДАННЫХ ПРОЕКТА «Kaggle»)» на языке Python.

```
#библиотеки
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from tqdm import tqdm
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from xgboost import XGBRegressor

#--------------------------------------------------------------------------------
#функции
def invboxcox(y,lmbda):# обратное преобразование Бокса-Кокса
    if lmbda == 0:
        return(np.exp(y))
    else:
        return(np.exp(np.log(lmbda*y+1)/lmbda))

def get_mape(y_true, y_pred):
    """
    Вычислить среднюю абсолютную процентную ошибку (MAPE)
    """
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def get_mov_avg_std(df, col, N):
    """
    Given a dataframe, get mean and std dev at timestep t using values from t-1, t-2,
..., t-N.
    Inputs
        df         : dataframe. Can be of any length.
        col        : name of the column you want to calculate mean and std cv
        N          : get mean and std dev at timestep t using values from t-1, t-2,
..., t-N
    Outputs
```

```
        df_out     : same as df but with additional column containing mean and std cv
    """
    mean_list = df[col].rolling(window = N, min_periods=1).mean() # len(mean_list) =
len(df)
    std_list = df[col].rolling(window = N, min_periods=1).std()   # first value will be
NaN, because normalized by N-1

    # Add one timestep to the predictions
    mean_list = np.concatenate((np.array([np.nan]), np.array(mean_list[:-1])))
    std_list = np.concatenate((np.array([np.nan]), np.array(std_list[:-1])))

    # Append mean_list to df
    df_out = df.copy()
    df_out[col + '_mean'] = mean_list
    df_out[col + '_std'] = std_list

    return df_out

def scale_row(row, feat_mean, feat_std):
    """
    Given a pandas series in row, scale it to have 0 mean and var 1 using feat_mean and
feat_std
    Inputs
        row      : pandas series. Need to scale this.
        feat_mean: mean
        feat_std : standard deviation
    Outputs
        row_scaled : pandas series with same length as row, but scaled
    """
    # If feat_std = 0 (this happens if adj_close doesn't change over N days),
    # set it to a small number to avoid division by zero
    feat_std = 0.001 if feat_std == 0 else feat_std

    row_scaled = (row-feat_mean) / feat_std

    return row_scaled

def train_pred_xgboost(X_train_scaled, \
                           y_train_scaled, \
                           X_test_scaled, \
                           y_test, \
```

```
                            col_mean, \
                            col_std, \
                            seed=100, \
                            n_estimators=100, \
                            max_depth=3, \
                            learning_rate=0.1, \
                            min_child_weight=1, \
                            subsample=1, \
                            colsample_bytree=1, \
                            colsample_bylevel=1, \
                            gamma=0):
    '''
    Train model, do prediction, scale back to original range and do evaluation
    Use XGBoost here.
    Inputs
        X_train_scaled     : features for training. Scaled to have mean 0 and variance
1
        y_train_scaled     : target for training. Scaled to have mean 0 and variance 1
        X_test_scaled      : features for test. Each sample is scaled to mean 0 and
variance 1
        y_test             : target for test. Actual values, not scaled.
        col_mean           : means used to scale each sample of X_test_scaled. Same
length as X_test_scaled and y_test
        col_std            : standard deviations used to scale each sample of
X_test_scaled. Same length as X_test_scaled and y_test
        seed               : model seed
        n_estimators       : number of boosted trees to fit
        max_depth          : maximum tree depth for base learners
        learning_rate      : boosting learning rate (xgb's "eta")
        min_child_weight   : minimum sum of instance weight(hessian) needed in a child
        subsample          : subsample ratio of the training instance
        colsample_bytree   : subsample ratio of columns when constructing each tree
        colsample_bylevel  : subsample ratio of columns for each split, in each level
        gamma              :
    Outputs
        mape               : mean absolute percentage error of y_test and pred
        rmse               : root mean square error of y_test and pred
        r2                 : r2
        pred               : predicted values. Same length as y_test
    '''
                             #seed=model_seed
```

```
    model = XGBRegressor(n_estimators=n_estimators,
                         max_depth=max_depth,
                         learning_rate=learning_rate,
                         min_child_weight=min_child_weight,
                         subsample=subsample,
                         colsample_bytree=colsample_bytree,
                         colsample_bylevel=colsample_bylevel,
                         gamma=gamma)

    # Train the model
    model.fit(X_train_scaled, y_train_scaled)

    # Get predicted labels and scale back to original range
    pred_scaled = model.predict(X_test_scaled)
    pred = pred_scaled * col_std + col_mean

    # Calculate error
    mape = get_mape(y_test, pred)
    rmse = math.sqrt(mean_squared_error(y_test, pred))
    r2 = r2_score(y_test, pred)

    return  mape, rmse, r2, pred
#--------------------------------------START-------------------------------------------
```

#Модуль программы

```
# загрузка dataset
db = pd.read_csv('SalesX.csv',';', parse_dates=['Date'], index_col='Date',
dayfirst=True, low_memory = False) #1,045,311
df2 = db

df = df2[['GMV','MRP','SLA','Units']].resample('h').max()
df = df.fillna(method='ffill')
df = pd.DataFrame({'GMV': df.GMV/10,'MRP': df.MRP/10,
                   'SLA': df.SLA,'Units': df.Units
                   })

df['GMV_'] = df.GMV
df['GMV'], lmbda = scs.boxcox(df.GMV)
print("Оптимальный параметр преобразования Бокса-Кокса: %f" % lmbda)
tsplot(df.GMV, lags=30)
```

```
# добавление доп. Признаков
df["DATE"] = df.index.date
# Вычислим разницу между
df['range_mg'] = df.MRP - df.GMV_

# Add a column 'order_day' to indicate the order of the rows by date
df['order_day'] = [x for x in list(range(len(df)))]
# merging_keys
merging_keys = ['order_day']
# List of columns that we will use to create lags
lag_cols = ['GMV', 'MRP', 'range_mg', 'SLA', 'Units']
lag_cols

N = 3

shift_range = [x+1 for x in range(N)]

for shift in tqdm(shift_range):

    train_shift = df[merging_keys + lag_cols].copy()

    # E.g. order_day of 0 becomes 1, for shift = 1.
    # So when this is merged with order_day of 1 in df, this will represent lag of 1.
    train_shift['order_day'] = train_shift['order_day'] + shift

    foo = lambda x: '{}_lag_{}'.format(x, shift) if x in lag_cols else x
    train_shift = train_shift.rename(columns=foo)

    df = pd.merge(df, train_shift, on=merging_keys, how='left') #.fillna(0)

del train_shift

# Remove the first N rows which contain NaNs
df = df[N:]

cols_list =  ['GMV', 'MRP', 'range_mg', 'SLA', 'Units']
for col in cols_list:
    df = get_mov_avg_std(df, col, N)

# Количество train, cv, test наборов данных
```

```
test_size = 0.3                # proportion of dataset to be used as test set
cv_size = 0.2                  # proportion of dataset to be used as cross-validation
set


n_cv, n_test  = int(cv_size*len(df)), int(test_size*len(df))
n_train = len(df) - n_cv - n_test
print('Количество:')
print(" n_train = " + str(n_train),
        "\n n_cv = " + str(n_cv),
        "\n n_test = " + str(n_test))


# Разделить на train, cv, and test
train = df[:n_train].copy()
cv = df[n_train:n_train+n_cv].copy()
train_cv = df[:n_train+n_cv].copy()
test = df[n_train+n_cv:].copy()

print(" train = " + str(train.shape),
        "\n cv = " + str(cv.shape), "~ train_cv = " + str(train_cv.shape),
        "\n test = " + str(test.shape))

cols_to_scale = [ "GMV" ]

for i in range(1,N+1):
    cols_to_scale.append("GMV_lag_"+str(i))
    cols_to_scale.append("MRP_lag_"+str(i))
    cols_to_scale.append("range_mg_lag_"+str(i))
    cols_to_scale.append("SLA_lag_"+str(i))
    cols_to_scale.append("Units_lag_"+str(i))


# Do scaling for train set
# Here we only scale the train dataset, and not the entire dataset to prevent
information leak
scaler = StandardScaler()
train_scaled = scaler.fit_transform(train[cols_to_scale])

print("scaler.mean_ = " + str(scaler.mean_))
print("scaler.var_ = " + str(scaler.var_))
```

```
print("train_scaled.shape = " + str(train_scaled.shape))

# Convert the numpy array back into pandas dataframe

train_scaled = pd.DataFrame(train_scaled, columns=cols_to_scale)

train_scaled[['DATE']] = train.reset_index()[['DATE']]
print("train_scaled.shape = " + str(train_scaled.shape))
train_scaled.head()

# Do scaling for train+cv set
scaler_train_cv = StandardScaler()
train_cv_scaled = scaler_train_cv.fit_transform(train_cv[cols_to_scale])
print("scaler_train_cv.mean_ = " + str(scaler_train_cv.mean_))
print("scaler_train_cv.var_ = " + str(scaler_train_cv.var_))
print("train_cv_scaled.shape = " + str(train_cv_scaled.shape))

# Convert the numpy array back into pandas dataframe
train_cv_scaled = pd.DataFrame(train_cv_scaled, columns=cols_to_scale)
train_cv_scaled[['DATE']] = train_cv.reset_index()[['DATE']]
print("train_cv_scaled.shape = " + str(train_cv_scaled.shape))
train_cv_scaled.head()

# Do scaling for cv set
cv_scaled = cv[['DATE']]
for col in tqdm(cols_list):
    feat_list = [col + '_lag_' + str(shift) for shift in range(1, N+1)]
    temp = cv.apply(lambda row: scale_row(row[feat_list], row[col+'_mean'],
row[col+'_std']), axis=1)
    cv_scaled = pd.concat([cv_scaled, temp], axis=1)

# Now the entire dev set is scaled
cv_scaled.head()

# Do scaling for test set
test_scaled = test[['DATE']]
for col in tqdm(cols_list):
    feat_list = [col + '_lag_' + str(shift) for shift in range(1, N+1)]
    temp = test.apply(lambda row: scale_row(row[feat_list], row[col+'_mean'],
row[col+'_std']), axis=1)
    test_scaled = pd.concat([test_scaled, temp], axis=1)
```

```
# Now the entire test set is scaled
test_scaled.head()

features = []
for i in range(1,N+1):
    features.append("GMV_lag_"+str(i))
    features.append("MRP_lag_"+str(i))
    features.append("range_mg_lag_"+str(i))
    features.append("SLA_lag_"+str(i))
    features.append("Units_lag_"+str(i))

target = "GMV"


# Split into X and y
X_train = train[features]
y_train = train[target]
X_cv = cv[features]
y_cv = cv[target]
X_train_cv = train_cv[features]
y_train_cv = train_cv[target]
X_sample = test[features]
y_sample = test[target]
print("X_train = " + str(X_train.shape))
print("y_train = " + str(y_train.shape))
print("X_cv = " + str(X_cv.shape))
print("y_cv = " + str(y_cv.shape))
print("X_train_cv = " + str(X_train_cv.shape))
print("y_train_cv = " + str(y_train_cv.shape))
print("X_sample = " + str(X_sample.shape))
print("y_sample = " + str(y_sample.shape))

# Split into X and y
X_train_scaled = train_scaled[features]
y_train_scaled = train_scaled[target]
X_cv_scaled = cv_scaled[features]
X_train_cv_scaled = train_cv_scaled[features]
y_train_cv_scaled = train_cv_scaled[target]
X_sample_scaled = test_scaled[features]
```

```
print("X_train_scaled = " + str(X_train_scaled.shape))
print("y_train_scaled = " + str(y_train_scaled.shape))
print("X_cv_scaled = " + str(X_cv_scaled.shape))
print("X_train_cv_scaled = " + str(X_train_cv_scaled.shape))
print("y_train_cv_scaled = " + str(y_train_cv_scaled.shape))
print("X_sample_scaled = " + str(X_sample_scaled.shape))

# График деление на train, cv, и test
plt.figure(figsize=(15, 5))
plt.plot(train.GMV, 'b',label = 'train')
plt.plot(cv.GMV, 'y',label= 'cv')
plt.plot(test.GMV, 'g',label= 'test')
plt.legend(loc='best')
plt.xlabel("DATE")
plt.ylabel("USD")
plt.grid(True)

# Create the model
#model = XGBRegressor(gamma=0, objective="reg:squarederror")
model = XGBRegressor()
'''model = XGBRegressor(seed = model_seed,
                     n_estimators = n_estimators,
                     max_depth = max_depth,
                     learning_rate = learning_rate,
                     min_child_weight = min_child_weight,
                     subsample = subsample,
                     colsample_bytree = colsample_bytree,
                     colsample_bylevel = colsample_bylevel,
                     gamma = gamma)'''

# Train the regressor
model.fit(X_train_scaled, y_train_scaled)

# Do prediction on train set
pred_scaled = model.predict(X_train_scaled)
pred = pred_scaled * math.sqrt(scaler.var_[0]) + scaler.mean_[0]

print("MAPE on train set = {:.4}".format(get_mape(y_train, pred)))
print("RMSE on train set = {:.4}".format(math.sqrt(mean_squared_error(y_train, pred))))
print("R2 on train set = {:.4}".format(r2_score(y_train, pred)))
```

```
# Do prediction on test set
pred_scaled = model.predict(X_cv_scaled)
cv['pred_scaled'] = pred_scaled
cv['pred'] = cv['pred_scaled'] * cv['GMV_std'] + cv['GMV_mean']

print("RMSE on cv set = {:.4}".format(get_mape(y_cv, cv.pred)))
print("RMSE on cv set = {:.4}".format(math.sqrt(mean_squared_error(y_cv, cv.pred))))
print("R2 on cv set = {:.4}".format(r2_score(y_cv, cv.pred)))

# Просмотрите список функций и их оценки важности
imp = list(zip(train[features], model.feature_importances_))
imp.sort(key=lambda tup: tup[1])
imp[-10:]

rmse, mape, r2, pred = train_pred_xgboost(X_train_cv_scaled,
                                      y_train_cv_scaled,
                                      X_sample_scaled,
                                      y_sample,
                                      test.GMV_mean,
                                      test.GMV_std)

# Calculate error
print("MAPE on test set = {:.4}".format(mape))
print("RMSE on test set = {:.4}".format(rmse))
print("R2 on test set = {:.4}".format(r2))

pred_df = pd.DataFrame({'pred': pred,
                        'Date': df[n_train+n_cv:]['DATE']})

# График прогноза
pred_df["pred_"] = invboxcox(pred_df.pred, lmbda)

plt.figure(figsize=(15, 5))
plt.plot(test.GMV_, label="actual" )
plt.plot(pred_df.pred_,'r' ,label="prediction")
plt.legend(loc = 'best')
plt.xlabel("DATE")
plt.ylabel("usd")
#plt.xlim([1, 30])
plt.title("XGBOOST\n MAPE: {:.4}, RMSE: {:.4}; R2: {:.4}".format(
    get_mape(test.GMV, pred),
```

```
    np.sqrt(mean_squared_error(test.GMV, pred)),
    r2_score(test.GMV, pred)))
plt.grid(True)
#---------------------------------------- END ----------------------------------------
```

Приложение Д

Программный код «МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АНАЛИЗА И ПРОГНОЗА ОБЪЁМОВ ПРОДАЖ (НА ПРИМЕРЕ ДАННЫХ ПРОЕКТА «Kaggle»)» на языке Python.

#библиотеки

```
import math
import matplotlib
import numpy as np
import pandas as pd
import seaborn as sns
import time

from datetime import date
from matplotlib import pyplot as plt
from numpy.random import seed
from pylab import rcParams
from sklearn.metrics import mean_squared_error

from tqdm import tqdm_notebook
from sklearn.preprocessing import StandardScaler

#from tensorflow import set_random_seed
from tensorflow.random import set_seed
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
#from keras.utils import plot_model
from tensorflow.keras.utils import plot_model
#from keras.utils.vis_utils import model_to_dot
#-------------------------------------------------------------------------------------
```

#функции

```
def invboxcox(y,lmbda):# обратное преобразование Бокса-Кокса
    if lmbda == 0:
        return(np.exp(y))
    else:
        return(np.exp(np.log(lmbda*y+1)/lmbda))

def get_mape(y_true, y_pred):
    """
    Compute mean absolute percentage error (MAPE)
```

```
    """
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def get_x_y(data, N, offset):
    """
    Split data into x (features) and y (target)
    """
    x, y = [], []
    for i in range(offset, len(data)):
        x.append(data[i-N:i])
        y.append(data[i])
    x = np.array(x)
    y = np.array(y)

    return x, y

def get_x_scaled_y(data, N, offset):
    """
    Split data into x (features) and y (target)
    We scale x to have mean 0 and std dev 1, and return this.
    We do not scale y here.
    Inputs
        data     : pandas series to extract x and y
        N
        offset
    Outputs
        x_scaled : features used to predict y. Scaled such that each element has mean 0
and std dev 1
        y        : target values. Not scaled
        mu_list  : list of the means. Same length as x_scaled and y
        std_list : list of the std devs. Same length as x_scaled and y
    """
    x_scaled, y, mu_list, std_list = [], [], [], []
    for i in range(offset, len(data)):
        mu_list.append(np.mean(data[i-N:i]))
        if np.std(data[i-N:i]) == 0:
            std_list.append(1)
        else:
            std_list.append(np.std(data[i-N:i]))
        x_scaled.append((data[i-N:i]-mu_list[i-offset])/std_list[i-offset])
```

```
        y.append(data[i])
    x_scaled = np.array(x_scaled)
    y = np.array(y)

    return x_scaled, y, mu_list, std_list

def train_pred_eval_model(x_train_scaled, \
                          y_train_scaled, \
                          x_cv_scaled, \
                          y_cv, \
                          mu_cv_list, \
                          std_cv_list, \
                          lstm_units=50, \
                          dropout_prob=0.5, \
                          optimizer='adam', \
                          epochs=0, \
                          batch_size=1):
    '''
    Train model, do prediction, scale back to original range and do evaluation
    Use LSTM here.
    Returns rmse, mape and predicted values
    Inputs
        x_train_scaled  : e.g. x_train_scaled.shape=(451, 9, 1). Here we are using the
past 9 values to predict the next value
        y_train_scaled  : e.g. y_train_scaled.shape=(451, 1)
        x_cv_scaled     : use this to do predictions
        y_cv            : actual value of the predictions
        mu_cv_list      : list of the means. Same length as x_scaled and y
        std_cv_list     : list of the std devs. Same length as x_scaled and y
        lstm_units      : lstm param
        dropout_prob    : lstm param
        optimizer       : lstm param
        epochs          : lstm param
        batch_size      : lstm param
    Outputs
        rmse            : root mean square error
        mape            : mean absolute percentage error
        est             : predictions
    '''
    # Create the LSTM network
    model = Sequential()
```

```
    model.add(LSTM(units=lstm_units, return_sequences=True,
input_shape=(x_train_scaled.shape[1],1)))
    model.add(Dropout(dropout_prob)) # Add dropout with a probability of 0.5
    model.add(LSTM(units=lstm_units))
    model.add(Dropout(dropout_prob)) # Add dropout with a probability of 0.5
    model.add(Dense(1))

    # Compile and fit the LSTM network
    model.compile(loss='mean_squared_error', optimizer=optimizer)
    model.fit(x_train_scaled, y_train_scaled, epochs=epochs, batch_size=batch_size,
verbose=0)

    # Do prediction
    #print(np.isnan(np.sum(x_cv_scaled))) # есть nan
    est_scaled = model.predict(x_cv_scaled)
    est = (est_scaled * np.array(std_cv_list).reshape(-1,1)) +
np.array(mu_cv_list).reshape(-1,1)

    # Calculate RMSE and MAPE
#     print("x_cv_scaled = " + str(x_cv_scaled))
#     print("est_scaled = " + str(est_scaled))
#     print("est = " + str(est))

    #print(np.isnan(np.sum(y_cv)), np.isnan(np.sum(est)))
    rmse = math.sqrt(mean_squared_error(y_cv, est))
    mape = get_mape(y_cv, est)
    r2 = r2_score(y_cv, est)

    return rmse, mape, r2 , est
#-------------------------------------START-----------------------------------------
#Модуль программы

# загрузка dataset
db = pd.read_csv('SalesX.csv',';', parse_dates=['Date'], index_col='Date',
dayfirst=True, low_memory = False) #1,045,311
df2 = db

df = df2[['GMV','MRP','SLA','Units']].resample('h').max()
df = df.fillna(method='ffill')
df = pd.DataFrame({'GMV': df.GMV/10,'MRP': df.MRP/10,
                   'SLA': df.SLA,'Units': df.Units
```

```
                  })

j = 0
l = []
for i in range(0, 6863):
    if df.GMV[i] > 8000:
        df.replace(df.GMV[i], np.nan, inplace=True)
        df.fillna(method='ffill',inplace=True )
        j+=1
        l.append(i)

df['GMV_'] = df.GMV
df['GMV'], lmbda = scs.boxcox(df.GMV)
print("Оптимальный параметр преобразования Бокса-Кокса: %f" % lmbda)
tsplot(df.GMV, lags=30)

# Convert Date column to datetime
df["Date"] = df.index.date

df.loc[:, 'Date'] = pd.to_datetime(df['Date'],format='%Y-%m-%d')

# Change all column headings to be lower case, and remove spacing
df.columns = [str(x).lower().replace(' ', '_') for x in df.columns]

# Get month of each sample
#df['month'] = df['date'].dt.month

# Sort by datetime
df.sort_values(by='date', inplace=True, ascending=True)

df['range_mg'] = df.mrp - df.gmv
#df.drop(['units'], axis=1, inplace=True)

# Get sizes of each of the datasets
test_size = 0.3                # proportion of dataset to be used as test set
cv_size = 0.2                  # proportion of dataset to be used as cross-validation
set

N = 9                          # for feature at day t, we use lags from t-1, t-2, ...,
t-N as features.
```

```
num_cv, num_test = int(cv_size*len(df)), int(test_size*len(df))
num_train = len(df) - num_cv - num_test
print('Количество:')
print("num_train = " + str(num_train),
      "\n num_cv = " + str(num_cv),
      "\n num_test = " + str(num_test))

# Разделить на train, cv, and test
train = df[:num_train][['date', 'gmv', 'gmv_']]
cv = df[num_train:num_train+num_cv][['date', 'gmv', 'gmv_']]
train_cv = df[:num_train+num_cv][['date', 'gmv', 'gmv_']]
test = df[num_train+num_cv:][['date', 'gmv', 'gmv_']]
print('Размерность:')
print(" train = " + str(train.shape),
        "\n cv = " + str(cv.shape), "~ train_cv = " + str(train_cv.shape),
        "\n test = " + str(test.shape))

# Преобразование набора данных в x_train и y_train
# Здесь мы масштабируем только обучающий набор данных, а не весь набор данных, чтобы
предотвратить утечку информации

scaler = StandardScaler()
train_scaled = scaler.fit_transform(np.array(train['gmv']).reshape(-1,1))
print("scaler.mean_ = " + str(scaler.mean_))
print("scaler.var_ = " + str(scaler.var_))

# Разделить на x и y

x_train_scaled, y_train_scaled = get_x_y(train_scaled, N, N)
print("x_train_scaled.shape = " + str(x_train_scaled.shape)) # (446, 7, 1)
print("y_train_scaled.shape = " + str(y_train_scaled.shape)) # (446, 1)

# Масштабировать набор данных cv
# Разделить на x и y

x_cv_scaled, y_cv, mu_cv_list, std_cv_list =
get_x_scaled_y(np.array(train_cv['gmv']).reshape(-1,1), N, num_train)

print("x_cv_scaled.shape = " + str(x_cv_scaled.shape))
print("y_cv.shape = " + str(y_cv.shape))
```

```
print("len(mu_cv_list) = " + str(len(mu_cv_list)))
print("len(std_cv_list) = " + str(len(std_cv_list)))

# Здесь мы масштабируем набор train_cv для окончательной модели

scaler_final = StandardScaler()
train_cv_scaled_final = scaler_final.fit_transform(np.array(train_cv['gmv']).reshape(-
1,1))
print("scaler_final.mean_ = " + str(scaler_final.mean_))
print("scaler_final.var_ = " + str(scaler_final.var_))

# # Scale the test dataset
x_test_scaled, y_test, mu_test_list, std_test_list =
get_x_scaled_y(np.array(df['gmv']).reshape(-1,1), N, num_train+num_cv)
print("x_test_scaled.shape = " + str(x_test_scaled.shape))
print("y_test.shape = " + str(y_test.shape))
print("len(mu_test_list) = " + str(len(mu_test_list)))
print("len(std_test_list) = " + str(len(std_test_list)))

#### Input params ##################

# initial value before tuning
lstm_units=50                  # lstm param. initial value before tuning.
dropout_prob=1                 # lstm param. initial value before tuning.
optimizer='adam' #  'nadam'    # lstm param. initial value before tuning.
epochs= 0                       # lstm param. initial value before tuning.
batch_size=1                   # lstm param. initial value before tuning.

model_seed = 100

fontsize = 14
ticklabelsize = 14
####################################

# Set seeds to ensure same output results
seed(101)
set_seed(model_seed)

# Create the LSTM network
```

```
model = Sequential()
model.add(LSTM(units=lstm_units, return_sequences=True,
input_shape=(x_train_scaled.shape[1],1)))
model.add(Dropout(dropout_prob)) # Add dropout with a probability of 0.5
model.add(LSTM(units=lstm_units))
model.add(Dropout(dropout_prob)) # Add dropout with a probability of 0.5
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer=optimizer)
model.fit(x_train_scaled, y_train_scaled, epochs=epochs, batch_size=batch_size,
verbose=2)

# Print model summary
model.summary()

# Do prediction
est_scaled = model.predict(x_cv_scaled)
est = (est_scaled * np.array(std_cv_list).reshape(-1,1)) +
np.array(mu_cv_list).reshape(-1,1)
print("est.shape = " + str(est.shape))

# Calculate RMSE
rmse_bef_tuning = math.sqrt(mean_squared_error(y_cv, est))
print("RMSE = %0.3f" % rmse_bef_tuning)

# Calculate MAPE
mape_pct_bef_tuning = get_mape(y_cv, est)
print("MAPE = %0.3f%%" % mape_pct_bef_tuning)

# Calculate R2
from sklearn.metrics import r2_score
r2_bef_tuning = r2_score(y_cv, est)
print("R2 = %0.3f" % r2_bef_tuning)

N_opt = 4
# Split train_cv into x and y
x_train_cv_scaled, y_train_cv_scaled = get_x_y(train_cv_scaled_final, N_opt, N_opt)

# Split test into x and y
x_test_scaled, y_test, mu_test_list, std_test_list =
get_x_scaled_y(np.array(df['gmv']).reshape(-1,1), N_opt, num_train+num_cv)
```

```
# Train, predict and eval model
rmse, mape, r2 , est = train_pred_eval_model(x_train_cv_scaled, \
                                             y_train_cv_scaled, \
                                             x_test_scaled, \
                                             y_test, \
                                             mu_test_list, \
                                             std_test_list)

# Calculate RMSE,MAPE,R2
print("RMSE on test set = {:.4}".format(rmse))
print("MAPE on test set = {:.4}%".format(mape))
print("R2 on test set = {:.4}".format(r2))

# График прогноза
est_df["est_"] = invboxcox(est_df.est, lmbda)

test.gmv_.plot(figsize=(15,5), label="actual")
est_df.est_.plot(figsize=(15,5), label="prediction", color = 'r')
plt.legend(loc = 'best')
plt.xlabel("date")
plt.ylabel("USD")
#plt.xlim([1, 30])
plt.title("LSTM\n MAPE: {:.4}, RMSE: {:.4}; R2: {:.4}".format(
    get_mape(test.gmv, est_df.est),
    np.sqrt(mean_squared_error(test.gmv, est_df.est)),
    r2_score(test.gmv, est_df.est)))
plt.grid(True)
#------------------------------------ END ----------------------------------------
```